Complete Listing of CWP Free Program Self-Documentations

generated by GENDOCS,
a shell script by John Stockwell
Center for Wave Phenomena
Colorado School of Mines

April 1, 2016

Center for Wave Phenomena
Colorado School of Mines

Names and Short descriptions of the Codes

CWPROOT = /usr/local/cwp

Mains:

In CWPROOT/src/cwp/main:
* CTRLSTRIP - Strip non-graphic characters
* DOWNFORT - change Fortran programs to lower case, preserving strings
* FCAT - fast cat with 1 read per file
* ISATTY - pass on return from isatty(2)
* MAXINTS - Compute maximum and minimum sizes for integer types
* PAUSE - prompt and wait for user signal to continue
* T - time and date for non-military types
* UPFORT - change Fortran programs to upper case, preserving strings

In CWPROOT/src/par/main:
A2B - convert ascii floats to binary
A2I - convert Ascii to binary Integers
ADDRVL3D - Add a random velocity layer (RVL) to a gridded
B2A - convert binary floats to ascii
CELLAUTO - Two-dimensional CELLular AUTOmata
char* sdoc[] = {
CSHOTPLOT - convert CSHOT data to files for CWP graphers
DZDV - determine depth derivative with respect to the velocity ",
FARITH - File ARITHmetic -- perform simple arithmetic with binary files
FLOAT2IBM - convert native binary FLOATS to IBM tape FLOATS
FTNSTRIP - convert a file of binary data plus record delimiters created
FTNUNSTRIP - convert C binary floats to Fortran style floats
GRM - Generalized Reciprocal refraction analysis for a single layer
H2B - convert 8 bit hexidecimal floats to binary
HTI2STIFF - convert HTI parameters alpha, beta, d(V), e(V), gamma
 HUDSON - compute effective parameters of anisotropic solids
I2A - convert binary integers to ascii
IBM2FLOAT - convert IBM tape FLOATS to native binary FLOATS
KAPERTURE - generate the k domain of a line scatterer for a seismic array
LINRORT - linearized P-P, P-S1 and P-S2 reflection coefficients

MAKEVEL - MAKE a VELocity function v(x,y,z)
MKPARFILE - convert ascii to par file format
MRAFXZWT - Multi-Resolution Analysis of a function F(X,Z) by Wavelet
PDFHISTOGRAM - generate a HISTOGRAM of the Probability Density function
PRPLOT - PRinter PLOT of 1-D arrays f(x1) from a 2-D function f(x1,x2)

RANDVEL3D - Add a random velocity layer (RVL) to a gridded
RAYT2DAN -- P- and SV-wave raytracing in 2D anisotropic media
RAYT2D - traveltime Tables calculated by 2D paraxial RAY tracing
RECAST - RECAST data type (convert from one data type to another)
REFREALAZIHTI - REAL AZImuthal REFL/transm coeff for HTI media
REFREALVTI - REAL REFL/transm coeff for VTI media and symmetry-axis
REGRID3 - REwrite a [ni3][ni2][ni1] GRID to a [no3][no2][no1] 3-D grid
RESAMP - RESAMPle the 1st dimension of a 2-dimensional function f(x1,x2)

SMOOTH2 --- SMOOTH a uniformly sampled 2d array of data, within a user-
SMOOTH3D - 3D grid velocity SMOOTHing by the damped least squares
SMOOTHINT2 --- SMOOTH non-uniformly sampled INTerfaces, via the damped
STIFF2VEL - Transforms 2D elastic stiffnesses to (vp,vs,epsilon,delta)
SUBSET - select a SUBSET of the samples from a 3-dimensional file
SWAPBYTES - SWAP the BYTES of various data types
THOM2HTI - Convert Thompson parameters V_p0, V_s0, eps, gamma,
THOM2STIFF - convert Thomsen's parameters into (density normalized)
TRANSP3D - TRANSPose an n1 by n2 by n3 element matrix
TRANSP - TRANSPose an n1 by n2 element matrix
TVNMOQC - Check tnmo-vnmo pairs; create t-v column files
UNIF2ANISO - generate a 2-D UNIFormly sampled profile of elastic
UNIF2 - generate a 2-D UNIFormly sampled velocity profile from a layered
UNIF2TI2 - generate a 2-D UNIFormly sampled profile of stiffness
UNISAM2 - UNIformly SAMple a 2-D function f(x1,x2)
UNISAM - UNIformly SAMple a function y(x) specified as x,y pairs
UTMCONV - CONVert longitude and latitude to UTM, and vice versa
VEL2STIFF - Transforms VELocities, densities, and Thomsen or Sayers
VELCONV - VELocity CONVersion
VELPERTAN - Velocity PERTerbation analysis in ANisotropic media to
VELPERT - estimate velocity parameter perturbation from covariance

VTLVZ -- Velocity as function of Time for Linear V(Z);
WKBJ - Compute WKBJ ray theoretic parameters, via finite differencing
XY2Z - converts (X,Y)-pairs to spike Z values on a uniform grid
Z2XYZ - convert binary floats representing Z-values to ascii

In CWPROOT/src/psplot/main:
PSBBOX - change BoundingBOX of existing PostScript file
PSCONTOUR - PostScript CONTOURing of a two-dimensional function f(x1,x2)
PSCUBE - PostScript image plot with Legend of a data CUBE
PSCCONTOUR - PostScript Contour plot of a data CUBE
PSEPSI - add an EPSI formatted preview bitmap to an EPS file
PSGRAPH - PostScript GRAPHer

```
PSIMAGE - PostScript IMAGE plot of a uniformly-sampled function f(x1,x2)
PSLABEL - output PostScript file consisting of a single TEXT string
PSMANAGER - printer MANAGER for HP 4MV and HP 5Si Mx Laserjet
PSMERGE - MERGE PostScript files
PSMOVIE - PostScript MOVIE plot of a uniformly-sampled function f(x1,x2,x3)
PSWIGB - PostScript WIGgle-trace plot of f(x1,x2) via Bitmap
PSWIGP - PostScript WIGgle-trace plot of f(x1,x2) via Polygons

In CWPROOT/src/xplot/main:
* LCMAP - List Color Map of root window of default screen
* LPROP - List PROPerties of root window of default screen of display
* SCMAP - set default standard color map (RGB_DEFAULT_MAP)
XCONTOUR - X CONTOUR plot of f(x1,x2) via vector plot call
* XESPB - X windows display of Encapsulated PostScript as a single Bitmap
* XEPSP - X windows display of Encapsulated PostScript
XIMAGE - X IMAGE plot of a uniformly-sampled function f(x1,x2)
XPICKER - X wiggle-trace plot of f(x1,x2) via Bitmap with PICKing
* XPSP - Display conforming PostScript in an X-window
XWIGB - X WIGgle-trace plot of f(x1,x2) via Bitmap

In CWPROOT/src/Xtcwp/main:
XGRAPH - X GRAPHer
XMOVIE - image one or more frames of a uniformly sampled function f(x1,x2)
XRECTS - plot rectangles on a two-dimensional grid

In CWPROOT/src/Xmcwp/main:
* FFTLAB - Motif-X based graphical 1D Fourier Transform

In CWPROOT/src/su/graphics/psplot:
SUPSCONTOUR - PostScript CONTOUR plot of a segy data set
SUPSCUBE - PostScript CUBE plot of a segy data set
SUPSCUBECONTOUR - PostScript CUBE plot of a segy data set
SUPSGRAPH - PostScript GRAPH plot of a segy data set
SUPSIMAGE - PostScript IMAGE plot of a segy data set
SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace
SUPSMOVIE - PostScript MOVIE plot of a segy data set
SUPSWIGB - PostScript Bit-mapped WIGgle plot of a segy data set
SUPSWIGP - PostScript Polygon-filled WIGgle plot of a segy data set

In CWPROOT/src/su/main/amplitudes:
SUCENTSAMP - CENTRoid SAMPle seismic traces
SUDIPDIVCOR - Dip-dependent Divergence (spreading) correction
SUDIVCOR - Divergence (spreading) correction
```

```
SUGAIN - apply various types of gain
SUNAN - remove NaNs & Infs from the input stream
SUNORMALIZE - Trace NORMALIZation by rms, max, or median       ",
SUPGC   -   Programmed Gain Control--apply agc like function
SUWEIGHT - weight traces by header parameter, such as offset
SUZERO -- zero-out (or set constant) data within a time window

In CWPROOT/src/su/main/attributes_parameter_estimation:
SUATTRIBUTES - instantaneous trace ATTRIBUTES

SUHISTOGRAM - create histogram of input amplitudes
SUMAX - get trace by trace local/global maxima, minima, or absolute maximum
SUMEAN - get the mean values of data traces ",
SUQUANTILE - display some quantiles or ranks of a data set

In CWPROOT/src/su/main/convolution_correlation:
SUACOR - auto-correlation
SUACORFRAC -- general FRACtional Auto-CORrelation/convolution
SUCONV - convolution with user-supplied filter
SUREFCON -  Convolution of user-supplied Forward and Reverse
SUXCOR - correlation with user-supplied filter

In CWPROOT/src/su/main/data_compression:
DCTCOMP - Compression by Discrete Cosine Transform
SUPACK1 - pack segy trace data into chars
SUPACK2 - pack segy trace data into 2 byte shorts
SUUNPACK1 - unpack segy trace data from chars to floats
SUUNPACK2 - unpack segy trace data from shorts to floats

In CWPROOT/src/su/main/data_conversion:
DT1TOSU - Convert ground-penetrating radar data in the
SEGYCLEAN - zero out unassigned portion of header
SEGYHDRMOD - replace the text header on a SEGY file
SEGYHDRS - make SEG-Y ascii and binary headers for segywrite
SEGYREAD - read an SEG-Y tape
SEGYSCAN -- SCANs SEGY file trace headers for min-max in  several
SEGYWRITE - write an SEG-Y tape
SETBHED - SET the fields in a SEGY Binary tape HEaDer file, as would be
SUASCII - print non zero header values and data in various formats
SUINTVEL - convert stacking velocity model to interval velocity model
SUOLDTONEW - convert existing su data to xdr format
SUSTKVEL - convert constant dip layer interval velocity model to the
SUSWAPBYTES - SWAP the BYTES in SU data to convert data from big endian
```

```
SWAPBHED - SWAP the BYTES in a SEGY Binary tape HEaDer file

In CWPROOT/src/su/main/datuming:
SUDATUMFD - 2D zero-offset Finite Difference acoustic wave-equation
SUDATUMK2DR - Kirchhoff datuming of receivers for 2D prestack data
SUDATUMK2DS - Kirchhoff datuming of sources for 2D prestack data
 SUKDMDCR - 2.5D datuming of receivers for prestack, common source
 SUKDMDCS - 2.5D datuming of sources for prestack common receiver

In CWPROOT/src/su/main/decon_shaping:
SUCDDECON - DECONvolution with user-supplied filter by straightforward
SUFXDECON - random noise attenuation by FX-DECONvolution
SUPEF - Wiener (least squares) predictive error filtering
SUPHIDECON - PHase Inversion Deconvolution
SUSHAPE - Wiener shaping filter

In CWPROOT/src/su/main/dip_moveout:
SUDMOFK - DMO via F-K domain (log-stretch) method for common-offset gathers
SUDMOFKCW - converted-wave DMO via F-K domain (log-stretch) method for
SUDMOTIVZ - DMO for Transeversely Isotropic V(Z) media for common-offset
SUDMOTX - DMO via T-X domain (Kirchhoff) method for common-offset gathers
SUDMOVZ - DMO for V(Z) media for common-offset gathers
SUTIHALEDMO - TI Hale Dip MoveOut (based on Hale's PhD thesis)

In CWPROOT/src/su/main/filters:
SUBFILT - apply Butterworth bandpass filter
SUCCFILT -  FX domain Correlation Coefficient FILTER
SUDIPFILT - DIP--or better--SLOPE Filter in f-k domain
SUFILTER - applies a zero-phase, sine-squared tapered filter
SUFRAC -- take general (fractional) time derivative or integral of
SUFWATRIM - FX domain Alpha TRIM
SUK1K2FILTER - symmetric box-like K-domain filter defined by the
SUKFILTER - radially symmetric K-domain, sin^2-tapered, polygonal
SUKFRAC - apply FRACtional powers of i|k| to data, with phase shift
SULFAF -  Low frequency array forming ",
SUMEDIAN - MEDIAN filter about a user-defined polygonal curve with
SUPHASE - PHASE manipulation by linear transformation
SUSMGAUSS2 --- SMOOTH a uniformly sampled 2d array of velocities
SUTVBAND - time-variant bandpass filter (sine-squared taper)

In CWPROOT/src/su/main/headers:
BHEDTOPAR - convert a Binary tape HEaDer file to PAR file format
SU3DCHART - plot x-midpoints vs. y-midpoints for 3-D data
```

SUABSHW - replace header key word by its absolute value
SUADDHEAD - put headers on bare traces and set the tracl and ns fields
SUAHW - Assign Header Word using another header word
SUAZIMUTH - compute trace AZIMUTH, offset, and midpoint coordinates
SUCDPBIN - Compute CDP bin number
SUCHART - prepare data for x vs. y plot
SUCHW - Change Header Word using one or two header word fields
SUCLIPHEAD - Clip header values
SUCOUNTKEY - COUNT the number of unique values for a given KEYword.
SUDUMPTRACE - print selected header values and data.
SUEDIT - examine segy diskfiles and edit headers
SUGEOM - Fill up geometry in trace headers.
SUGETHW - sugethw writes the values of the selected key words
SUHTMATH - do unary arithmetic operation on segy traces with
SUKEYCOUNT - sukeycount writes a count of a selected key
SULCTHW - Linear Coordinate Transformation of Header Words
SULHEAD - Load information from an ascii column file into HEADERS based
SUPASTE - paste existing SU headers on existing binary data
surandhw - set header word to random variable
SURANGE - get max and min values for non-zero header entries
SUSEHW - Set the value the Header Word denoting trace number within
SUSHW - Set one or more Header Words using trace number, mod and
SUSTRIP - remove the SEGY headers from the traces
SUTRCOUNT - SU program to count the TRaces in infile
SUUTM - UTM projection of longitude and latitude in SU trace headers
SUXEDIT - examine segy diskfiles and edit headers

In CWPROOT/src/su/main/interp_extrap:
SUINTERP - interpolate traces using automatic event picking
SUINTERPFOWLER - interpolate output image from constant velocity panels
SUOCEXT - smaller Offset EXTrapolation via Offset Continuation

In CWPROOT/src/su/main/migration_inversion:
SUGAZMIGQ - SU version of Jeno GAZDAG's phase-shift migration
SUINVXZCO - Seismic INVersion of Common Offset data for a smooth
SUINVZCO3D - Seismic INVersion of Common Offset data with V(Z) velocity
SUKDMIG2D - Kirchhoff Depth Migration of 2D poststack/prestack data
SUKDMIG3D - Kirchhoff Depth Migration of 3D poststack/prestack data
SUKTMIG2D - prestack time migration of a common-offset
SUMIGFD - 45-90 degree Finite difference depth migration for
SUMIGFFD - Fourier finite difference depth migration for
SUMIGGBZOAN - MIGration via Gaussian beams ANisotropic media (P-wave)
SUMIGGBZO - MIGration via Gaussian Beams of Zero-Offset SU data

```
SUMIGPREFD --- The 2-D prestack common-shot 45-90 degree
SUMIGPREFFD - The 2-D prestack common-shot Fourier finite-difference
char *sdoc[] = {
SUMIGPRESP - The 2-D prestack common-shot split-step Fourier ",
SUMIGPS - MIGration by Phase Shift with turning rays
SUMIGPSPI - Gazdag's phase-shift plus interpolation depth migration
SUMIGPSTI - MIGration by Phase Shift for TI media with turning rays
SUMIGSPLIT - Split-step depth migration for zero-offset data.
SUMIGTK - MIGration via T-K domain method for common-midpoint stacked data
SUMIGTOPO2D - Kirchhoff Depth Migration of 2D postack/prestack data
SUSTOLT - Stolt migration for stacked data or common-offset gathers
SUTIFOWLER   VTI constant velocity prestack time migration

In CWPROOT/src/su/main/multicomponent:
SUALFORD - trace by trace Alford Rotation of shear wave data volumes
SUEIPOFI - EIgenimage (SVD) based POlarization FIlter for
SUHROT - Horizontal ROTation of three-component data
SULTT - trace by trace, sample by sample, rotation of shear wave data
SUPOFILT - POlarization FILTer for three-component data
SUPOLAR - POLarization analysis of three-component data

In CWPROOT/src/su/main/noise:
SUADDNOISE - add noise to traces
SUHARLAN - signal-noise separation by the invertible linear
SUJITTER - Add random time shifts to seismic traces

In CWPROOT/src/su/main/operations:
SUFLIP - flip a data set in various ways
SUFWMIX -  FX domain multidimensional Weighted Mix
SUMATH - do math operation on su data
SUMIX - compute weighted moving average (trace MIX) on a panel
SUOP2 - do a binary operation on two data sets
SUOP - do unary arithmetic operation on segys
SUPERMUTE - permute or transpose a 3d datacube
SUSIMAT - Correlation similarity matrix for two traces.
SUVCAT -  append one data set to another, with or without an  ",
SUVLENGTH - Adjust variable length traces to common length

In CWPROOT/src/su/main/picking:
SUFBPICKW - First break auto picker
SUFNZERO - get Time of First Non-ZERO sample by trace
SUPICKAMP - pick amplitudes within user defined and resampled window
```

In CWPROOT/src/su/main/stacking:
SUCVS4FOWLER --compute constant velocity stacks for input to Fowler codes
SUDIVSTACK - Diversity Stacking using either average power or peak
SUPWS - Phase stack or phase-weighted stack (PWS) of adjacent traces
SURECIP - sum opposing offsets in prepared data (see below)
SUSTACK - stack adjacent traces having the same key header word

In CWPROOT/src/su/main/statics:
SUADDSTATICS - ADD random STATICS on seismic data
SURANDSTAT - Add RANDom time shifts STATIC errors to seismic traces
SURESSTAT - Surface consistent source and receiver statics calculation
SUSTATICB - Elevation static corrections, apply corrections from
SUSTATIC - Elevation static corrections, apply corrections from
SUSTATICRRS - Elevation STATIC corrections, apply corrections from

In CWPROOT/src/su/main/stretching_moveout_resamp:
SUILOG -- time axis inverse log-stretch of seismic traces
SULOG -- time axis log-stretch of seismic traces
SUNMO_a - NMO for an arbitrary velocity function of time and CDP with
SUNMO - NMO for an arbitrary velocity function of time and CDP
SUREDUCE - convert traces to display in reduced time ",
SURESAMP - Resample in time
SUSHIFT - shifted/windowed traces in time
SUTAUPNMO - NMO for an arbitrary velocity function of tau and CDP
SUTSQ -- time axis time-squared stretch of seismic traces
SUTTOZ - resample from time to depth
SUZTOT - resample from depth to time

In CWPROOT/src/su/main/supromax:
SUGET - Connect SU program to file descriptor for input stream.
SUPUT - Connect SU program to file descriptor for output stream.

In CWPROOT/src/su/main/synthetics_waveforms_testpatterns:

SUDGWAVEFORM - make Gaussian derivative waveform in SU format
SUEA2DF - SU version of (an)elastic anisotropic 2D finite difference
SUFCTANISMOD - Flux-Corrected Transport correction applied to the 2D
SUFDMOD1 - Finite difference modelling (1-D 1rst order) for the
SUFDMOD2 - Finite-Difference MODeling (2nd order) for acoustic wave equation
SUFDMOD2_PML - Finite-Difference MODeling (2nd order) for acoustic wave
SUGOUPILLAUD - calculate 1D impulse response of
SUGOUPILLAUDPO - calculate Primaries-Only impulse response of a lossless
SUIMP2D - generate shot records for a line scatterer

SUIMP3D - generate inplane shot records for a point
SUIMPEDANCE - Convert reflection coefficients to impedances.
SUKDSYN2D - Kirchhoff Depth SYNthesis of 2D seismic data from a
SUNHMOSPIKE - generates SPIKE test data set with a choice of several
SUNULL - create null (all zeroes) traces
SUPLANE - create common offset data file with up to 3 planes
SURANDSPIKE - make a small data set of RANDom SPIKEs
SUREMAC2D - Acoustic 2D Fourier method modeling with high accuracy
SUREMEL2DAN - Elastic anisotropic 2D Fourier method modeling with high
SUSPIKE - make a small spike data set
SUSYNCZ - SYNthetic seismograms for piecewise constant V(Z) function
SUSYNLV - SYNthetic seismograms for Linear Velocity function
SUSYNLVCW - SYNthetic seismograms for Linear Velocity function
SUSYNLVFTI - SYNthetic seismograms for Linear Velocity function in a
SUSYNVXZ - SYNthetic seismograms of common offset V(X,Z) media via
SUSYNVXZCS - SYNthetic seismograms of common shot in V(X,Z) media via
SUVIBRO - Generates a Vibroseis sweep (linear, linear-segment,
SUWAVEFORM - generate a seismic wavelet

In CWPROOT/src/su/main/tapering:
SUGAUSSTAPER - Multiply traces with gaussian taper
SURAMP - Linearly taper the start and/or end of traces to zero.
SUTAPER - Taper the edge traces of a data panel to zero.
SUTXTAPER - TAPER in (X,T) the edges of a data panel to zero.

In CWPROOT/src/su/main/transforms:
SUAMP - output amp, phase, real or imag trace from
SUANALYTIC - use the Hilbert transform to generate an ANALYTIC
SUCCEPSTRUM - Compute the complex CEPSTRUM of a seismic trace "
SUCCWT - Complex continuous wavelet transform of seismic traces
SUCEPSTRUM - transform to the CEPSTRal domain
SUCLOGFFT - fft real time traces to complex log frequency domain traces
SUCWT - generates Continous Wavelet Transform amplitude, regularity
SUFFT - fft real time traces to complex frequency traces
SUGABOR - Outputs a time-frequency representation of seismic data via
SUHILB - Hilbert transform
SUICEPSTRUM - fft of complex log frequency traces to real time traces
SUICLOGFFT - fft of complex log frequency traces to real time traces
SUIFFT - fft complex frequency traces to real time traces
SUMINPHASE - convert input to minimum phase
SUPHASEVEL - Multi-mode PHASE VELocity dispersion map computed
SURADON - compute forward or reverse Radon transform or remove multiples
SUSLOWFT - Fourier Transforms by a (SLOW) DFT algorithm (Not an FFT)

```
SUSLOWIFT - Fourier Transforms by (SLOW) DFT algorithm (Not an FFT)
SUSPECFK - F-K Fourier SPECtrum of data set
SUSPECFX - Fourier SPECtrum (T -> F) of traces
SUSPECK1K2 - 2D (K1,K2) Fourier SPECtrum of (x1,x2) data set
SUTAUP - forward and inverse T-X and F-K global slant stacks
SUWFFT - Weighted amplitude FFT with spectrum flattening 0->Nyquist
SUZEROPHASE - convert input to zero phase equivalent

In CWPROOT/src/su/main/velocity_analysis:
SURELANAN - REsiduaL-moveout semblance ANalysis for ANisotropic media
SURELAN - compute residual-moveout semblance for cdp gathers based
 SUTIVEL -  SU Transversely Isotropic velocity table builder
SUVEL2DF - compute stacking VELocity semblance for a single time in
SUVELAN - compute stacking velocity semblance for cdp gathers
SUVELAN_NCCS - compute stacking VELocity panel for cdp gathers
SUVELAN_NSEL - compute stacking VELocity panel for cdp gathers
SUVELAN_UCCS - compute stacking VELocity panel for cdp gathers
SUVELAN_USEL - compute stacking velocity panel for cdp gathers

In CWPROOT/src/su/main/well_logs:
LAS2SU - convert las2 format well log curves to su traces
SUBACKUS - calculate Thomsen anisotropy parameters from
SUBACKUSH - calculate Thomsen anisotropy parameters from
SUGASSMAN - Model reflectivity change with rock/fluid properties
SULPRIME - find appropriate Backus average length for
SUWELLRF - convert WELL log depth, velocity, density data into a

In CWPROOT/src/su/main/windowing_sorting_muting:
SUCOMMAND - pipe traces having the same key header word to command
SUGETGTHR - Gets su files from a directory and put them
SUGPRFB - SU program to remove First Breaks from GPR data
SUKILL - zero out traces
SUMIXGATHERS - mix two gathers
SUMUTE - MUTE above (or below) a user-defined polygonal curve with ",
SUPAD - Pad zero traces
SUPUTGTHR - split the stdout flow to gathers on the bases of given
SUSORT - sort on any segy header keywords
SUSORTY - make a small 2-D common shot off-end
SUSPLIT - Split traces into different output files by keyword value
SUWIND - window traces by key word
SUWINDPOLY - WINDow data to extract traces on or within a respective

In CWPROOT/src/su/graphics/psplot:
```

```
SUPSCONTOUR - PostScript CONTOUR plot of a segy data set
SUPSCUBE - PostScript CUBE plot of a segy data set
SUPSCUBECONTOUR - PostScript CUBE plot of a segy data set
SUPSGRAPH - PostScript GRAPH plot of a segy data set
SUPSIMAGE - PostScript IMAGE plot of a segy data set
SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace
SUPSMOVIE - PostScript MOVIE plot of a segy data set
SUPSWIGB - PostScript Bit-mapped WIGgle plot of a segy data set
SUPSWIGP - PostScript Polygon-filled WIGgle plot of a segy data set

In CWPROOT/src/su/graphics/xplot:
SUXCONTOUR - X CONTOUR plot of Seismic UNIX tracefile via vector plot call
SUXGRAPH - X-windows GRAPH plot of a segy data set
SUXIMAGE - X-windows IMAGE plot of a segy data set
SUXMAX - X-windows graph of the MAX, min, or absolute max value on
SUXMOVIE - X MOVIE plot of a 2D or 3D segy data set
SUXPICKER - X-windows  WIGgle plot PICKER of a segy data set
SUXWIGB - X-windows Bit-mapped WIGgle plot of a segy data set

In CWPROOT/src/tri/main:
GBBEAM - Gaussian beam synthetic seismograms for a sloth model
NORMRAY - dynamic ray tracing for normal incidence rays in a sloth model
TRI2UNI - convert a TRIangulated model to UNIformly sampled model
TRIMODEL - make a triangulated sloth (1/velocity^2) model
TRIRAY - dynamic RAY tracing for a TRIangulated sloth model
TRISEIS - Gaussian beam synthetic seismograms for a sloth model
UNI2TRI - convert UNIformly sampled model to a TRIangulated model

In CWPROOT/src/xtri:
SXPLOT - X Window plot a triangulated sloth function s(x1,x2)

In CWPROOT/src/tri/graphics/psplot:
SPSPLOT - plot a triangulated sloth function s(x,z) via PostScript

In CWPROOT/src/comp/dct/main:
DCTCOMP - Compression by Discrete Cosine Transform
DCTUNCOMP - Discrete Cosine Transform Uncompression
ENTROPY - compute the ENTROPY of a signal
WPTCOMP - Compression by Wavelet Packet Compression
WPTUNCOMP - Uncompress  WPT compressed data
WTCOMP - Compression by Wavelet Transform
WTUNCOMP - UNCOMPression of WT compressed data
```

```
In CWPROOT/src/comp/dwpt/1d/main:
WPC1COMP2 --- COMPress a 2D seismic section trace-by-trace using
WPC1UNCOMP2 --- UNCOMPRESS a 2D seismic section, which has been

In CWPROOT/src/comp/dwpt/2d/main:
WPCCOMPRESS --- COMPRESS a 2D section using Wavelet Packets
WPCUNCOMPRESS --- UNCOMPRESS a 2D section

Shells:

In CWPROOT/src/cwp/shell:
# ARGV - give examples of dereferencing char **argv
# COPYRIGHT - insert CSM COPYRIGHT lines at top of files in working directory
# CPALL , RCPALL - for local and remote directory tree/file transfer
# CWPFIND - look for files with patterns in CWPROOT/src/cwp/lib
# Grep  - recursively call egrep in pwd
# DIRTREE - show DIRectory TREE
# FILETYPE - list all files of given type
# NEWCASE - Changes the case of all the filenames in a directory, dir
# OVERWRITE - copy stdin to stdout after EOF
# PRECEDENCE - give table of C precedences from Kernighan and Ritchie
# REPLACE - REPLACE string1 with string2  in files
# THIS_YEAR - print the current year
# TIME_NOW - prints time in ZULU format with no spaces
# TODAYS_DATE - prints today's date in ZULU format with no spaces
# USERNAMES - get list of all login names
# VARLIST - list variables used in a Fortran program
# WEEKDAY - prints today's WEEKDAY designation
# ZAP - kill processes by name

In CWPROOT/src/par/shell:
# GENDOCS - generate complete list of selfdocs in latex form
# STRIPTOTXT -  put files from $CWPROOT/src/doc/Stripped into a new
# UPDATEDOCALL - put self-docs in ../doc/Stripped
# UPDATEDOC - put self-docs in ../doc/Stripped and ../doc/Headers
# UPDATEHEAD - update ../doc/Headers/Headers.all

In CWPROOT/src/psplot/shell:
# MERGE2 - put 2 standard size PostScript figures on one page
# MERGE4 - put 4 standard size PostScript plots on one page

In CWPROOT/src/su/shell:
# LOOKPAR - show getpar lines in SU code with defines evaluated
```

```
# MAXDIFF - find absolute maximum difference in two segy data sets
# RECIP - sum opposing (reciprocal) offsets in cdp sorted data
# RMAXDIFF - find percentage maximum difference in two segy data sets
# SUAGC - perform agc on SU data
# SUBAND - Trapezoid-like Sin squared tapered Bandpass filter via  SUFILTER
# SUDIFF, SUSUM, SUPROD, SUQUO, SUPTDIFF, SUPTSUM,
# SUDOC - get DOC listing for code
# SUENV - Instantaneous amplitude, frequency, and phase via: SUATTRIBUTES
# SUFIND - get info from self-docs
# SUFIND - get info from self-docs
# SUGENDOCS - generate complete list of selfdocs in latex form
# SUHELP - list the CWP/SU programs and shells
# SUKEYWORD -- guide to SU keywords in segy.h
# SUNAME - get name line from self-docs
# UNGLITCH - zonk outliers in data

Libs:

In CWPROOT/src/cwp/lib:
ABEL - Functions to compute the discrete ABEL transform:
AIRY - Approximate the Airy functions  Ai(x), Bi(x) and their respective
ALLOC - Allocate and free multi-dimensional arrays
ANTIALIAS - Butterworth anti-aliasing filter
AXB - Functions to solve a linear system of equations Ax=b by LU
BIGMATRIX - Functions to manipulate 2-dimensional matrices that are too big
BUTTERWORTH - Functions to design and apply Butterworth filters:
COMPLEX - Functions to manipulate complex numbers
COMPLEXD - Functions to manipulate double-precision complex numbers
COMPLEXF  - Subroutines to perform operations on complex numbers.
COMPLEXFD  - Subroutines to perform operations on double complex numbers.
Conjugate Gradient routines -
CONVOLUTION - Compute z = x convolved with y
CUBICSPLINE - Functions to compute CUBIC SPLINE interpolation coefficients
DBLAS - Double precision Basic Linear Algebra subroutines
DGE - Double precision Gaussian Elimination matrix subroutines  adapted
DIFFERENTIATE - simple DIFFERENTIATOR codes
PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place
*CWP_Exit - exit subroutine for CWP/SU codes
FRANNOR - functions to generate a pseudo-random float normally distributed
FRANUNI - Functions to generate a pseudo-random float uniformly distributed
HANKEL - Functions to compute discrete Hankel transforms
Hartley - routines for fast Hartley transform
HILBERT - Compute Hilbert transform y of x
```

```
HOLBERG1D - Compute coefficients of Holberg's 1st derivative filter
INTCUB - evaluate y(x), y'(x), y''(x), ... via piecewise cubic interpolation
INTL2B - bilinear interpolation of a 2-D array of bytes
INTLIN - evaluate y(x) via linear interpolation of y(x[0]), y(x[1]), ...
INTLINC - evaluate complex y(x) via linear interpolation of y(x[0]), y(x[1]), ...
INTLIRR2B - bilinear interpolation of a 2-D array of bytes
INTSINC8 - Functions to interpolate uniformly-sampled data via 8-coeff. sinc
INTTABLE8 -  Interpolation of a uniformly-sampled complex function y(x)
LINEAR_REGRESSION - Compute linear regression of (y1,y2,...,yn) against
maxmin - subroutines that pertain to maximum and minimum values
MKDIFF - Make an n-th order DIFFerentiator via Taylor's series method.
MKHDIFF - Compute filter approximating the bandlimited HalF-DIFFerentiator.
MKSINC - Compute least-squares optimal sinc interpolation coefficients.
MNEWT - Solve non-linear system of equations f(x) = 0 via Newton's method
ORTHPOLYNOMIALS - compute ORTHogonal POLYNOMIALS
PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place
POLAR - Functions to map data in rectangular coordinates to polar and vise versa
PRINTERPLOT - Functions to make a printer plot of a 1-dimensional array
QUEST - Functions to ESTimate Quantiles:
RESSINC8 - Functions to resample uniformly-sampled data  via 8-coefficient sinc
RFWTVA - Rasterize a Float array as Wiggle-Trace-Variable-Area.
RFWTVAINT - Rasterize a Float array as Wiggle-Trace-Variable-Area, with
SBLAS - Single precision Basic Linear Algebra Subroutines
SCAXIS - compute a readable scale for use in plotting axes
SGA - Single precision general matrix functions adapted from LINPACK FORTRAN:
SHFS8R - Shift a uniformly-sampled real-valued function y(x) via
SINC - Return SINC(x) for as floats or as doubles
SORT - Functions to sort arrays of data or arrays of indices
SQR - Single precision QR decomposition functions adapted from LINPACK FORTRAN:
STOEP - Functions to solve a symmetric Toeplitz linear system of equations
STRSTUFF -- STRing manuplation subs
SWAPBYTE - Functions to SWAP the BYTE order of binary data
SYMMEIGEN - Functions solving the eigenvalue problem for symmetric matrices

TRIDIAGONAL - Functions to solve tridiagonal systems of equations Tu=r for u.
UNWRAP_PHASE - routines to UNWRAP phase of fourier transformed data
VANDERMONDE - Functions to solve Vandermonde system of equations Vx=b
WAVEFORMS   Subroutines to define some wavelets for modeling of seismic
WINDOW - windowing routines
wrapArray - wrap an array
XCOR - Compute z = x cross-correlated with y
XINDEX - determine index of x with respect to an array of x values
YCLIP - Clip a function y(x) defined by linear interpolation of the
```

YXTOXY - Compute a regularly-sampled, monotonically increasing function x(y)
ZASC - routine to translate ncharacters from ebcdic to ascii
ZEBC - routine to translate ncharacters from ascii to ebcdic

In CWPROOT/src/par/lib:
ATOPKGE - convert ascii to arithmetic and with error checking
DOCPKGE - Function to implement the CWP self-documentation facility
EALLOC - Allocate and free multi-dimensional arrays with error reports.
ERRPKGE - routines for reporting errors
FILESTAT - Functions to determine and output the type of a file from file
GETPARS - Functions to GET PARameterS from the command line. Numeric
LINCOEFF - subroutines to create linearized reflection coefficients
MINFUNC - routines to MINimize FUNCtions
MODELING - Seismic Modeling Subroutines for SUSYNLV and clones
REFANISO - Reflection coefficients for Anisotropic media
RKE - integrate a system of n-first order ordinary differential equations
SMOOTH - Functions to compute smoothing of 1-D or 2-D input data
SUBCALLS - routines for system functions with error checking
SYSCALLS - routines for SYSTEM CALLs with error checking
TAUP - Functions to perform forward and inverse taup transforms (radon or
UPWEIK - Upwind Finite Difference Eikonal Solver
VND - large out-of-core multidimensional block matrix transpose
WTLIB - Functions for wavelet transforms

In CWPROOT/src/su/lib:
FGETGTHR - get gathers from SU datafiles
fgethdr - get segy tape identification headers from the file by file pointer
FGETTR - Routines to get an SU trace from a file
FPUTGTHR - put gathers to a file
FPUTTR - Routines to put an SU trace to a file
HDRPKGE - routines to access the SEGY header via the hdr structure.
TABPLOT - TABPLOT selected sample points on selected trace
VALPKGE - routines to handle variables of type Value

In CWPROOT/src/psplot/lib:
BASIC - Basic C function interface to PostScript
PSAXESBOX3 - Functions draw an axes box via PostScript, estimate bounding box
PSAXESBOX - Functions to draw PostScript axes and estimate bounding box
PSCAXESBOX - Draw an axes box for cube via PostScript
PSCONTOUR - draw contour of a two-dimensional array via PostScript
PSDRAWCURVE - Functions to draw a curve from a set of points
PSLEGENDBOX - Functions to draw PostScript axes and estimate bounding box
PSWIGGLE - draw wiggle-trace with (optional) area-fill via PostScript

In CWPROOT/src/xplot/lib:
AXESBOX - Functions to draw axes in X-windows graphics
COLORMAP - Functions to manipulate X colormaps:
DRAWCURVE - Functions to draw a curve from a set of points
IMAGE - Function for making the image in an X-windows image plot
LEGENDBOX - draw a labeled axes box for a legend (i.e. colorscale)
RUBBERBOX - Function to draw a rubberband box in X-windows plots
WINDOW - Function to create a window in X-windows graphics
XCONTOUR - draw contour of a two-dimensional array via X vectorplot calls

In CWPROOT/src/Xtcwp/lib:
AXES - the Axes Widget
COLORMAP - Functions to manipulate X colormaps:
FX - Functions to support floating point coordinates in X
MISC - Miscellaneous X-Toolkit functions
RESCONV - general purpose resource type converters
RUBBERBOX - Function to draw a rubberband box in X-windows plots

In CWPROOT/src/Xmcwp/lib:
RADIOBUTTONS - convenience functions creating and using radio buttons
SAMPLES - Motif-based Graphics Functions

In CWPROOT/src/tri/lib:
CHECK - CHECK triangulated models
CIRCUM - define CIRCUMcircles for Delaunay triangulation
COLINEAR - determine if edges or vertecies are COLINEAR in triangulated
CREATE - create model, boundary edge triangles, edge face, edge vertex, add
DELETE - DELETE vertex, model, edge, or boundary edge from triangulated model
DTE - Distance to Edge
FIXEDGES - FIX or unFIX EDGES between verticies
INSIDE - Is a vertex or point inside a circum circle, etc. of a triangulated
NEAREST - NEAREST edge or vertex in triangulated model
PROJECT - project to edge in triangulated model
READWRITE - READ or WRITE a triangulated model

In CWPROOT/src/cwputils:
CPUSEC - return cpu time (UNIX user time) in seconds
CPUTIME - return cpu time (UNIX user time) in seconds using ANSI C built-ins
WALLSEC - Functions to time processes
WALLTIME - Function to show time a process takes to run

In CWPROOT/src/comp/dct/lib:

```
BUFFALLOC - routines to ALLOCate/initialize and free BUFFers
DCT1 - 1D Discreet Cosine Transform Routines
DCT2 - 2D Discrete Cosine Transform Routines
DCTALLOC - ALLOCate space for transform tables for 1D DCT
GETFILTER - GET wavelet FILTER type
HUFFMAN - Routines for in memory Huffman coding/decoding
LCT1 - functions used to perform the 1D Local Cosine Transform (LCT)
LPRED - Lateral Prediction of Several Plane Waves
PENCODING - Routines to en/decode the quantized integers for lossless
QUANT - QUANTization routines
RLE - routines for in memory silence en/decoding
WAVEPACK1 - 1D wavelet packet transform
WAVEPACK2 - 2D Wavelet PACKet transform
WAVEPACK1 - 1D wavelet packet transform
WAVETRANS2 - 2D wavelet transform by tensor-product of two 1D transforms

In CWPROOT/src/comp/dct/lib:
BUFFALLOC - routines to ALLOCate/initialize and free BUFFers
DCT1 - 1D Discreet Cosine Transform Routines
DCT2 - 2D Discrete Cosine Transform Routines
DCTALLOC - ALLOCate space for transform tables for 1D DCT
GETFILTER - GET wavelet FILTER type
HUFFMAN - Routines for in memory Huffman coding/decoding
LCT1 - functions used to perform the 1D Local Cosine Transform (LCT)
LPRED - Lateral Prediction of Several Plane Waves
PENCODING - Routines to en/decode the quantized integers for lossless
QUANT - QUANTization routines
RLE - routines for in memory silence en/decoding
WAVEPACK1 - 1D wavelet packet transform
WAVEPACK2 - 2D Wavelet PACKet transform
WAVEPACK1 - 1D wavelet packet transform
WAVETRANS2 - 2D wavelet transform by tensor-product of two 1D transforms

In CWPROOT/src/comp/dwpt/1d/lib:
WBUFFALLOC -  routines to allocate/initialize and free buffers in wavelet
WPC1 - routines for compress a single seismic trace using wavelet packets
WPC1CODING - routines for encoding the integer symbols in 1D WPC
wpc1Quant - quantize
WPC1TRANS - routines to perform a 1D wavelet packet transform

In CWPROOT/src/comp/dwpt/2d/lib:
WPCBUFFAL - routines to allocate/initialize and free buffers
WPC - routines for compress a 2D seismic section using wavelet packets
```

```
WPCCODING - Routines for in memory coding of the quantized coeffiecients
WPCENDEC -  Wavelet Packet Coding, Encoding and Decoding routines
WPCHUFF -Routines for in memory Huffman coding/decoding
WPCPACK2 - routine to perform a 2D wavelet packet transform
WPCQUANT - quantization routines for WPC
WPCSILENCE - routines for in memory silence en/decoding


To search on a program name fragment, type:
      suname name_fragment <CR>

For more information type: program_name <CR>

  Items labeled with an asterisk (*) are C programs  that may
  or may not have this self documentation feature.

  Items labeled with a pound sign (#) are shell  scripts that may,
  or may not have the self documentation feature.
```

Self Documentations

Mains:

```
------
 CTRLSTRIP - Strip non-graphic characters

 ctrlstrip <dirtyfile >cleanfile



------
 DOWNFORT - change Fortran programs to lower case, preserving strings

 Usage:   downfort < infile.f > outfile.f




 Credits:
   Brian Sumner  c. 1984


------
 FCAT - fast cat with 1 read per file

 Usage: fcat file1 file2 ... > file3




 Credits:
Shuki

 This program belongs to the Center for Wave Phenomena
 Colorado School of Mines
/

------
 ISATTY - pass on return from isatty(2)

 Usage:  isatty filedes

 See:   man isatty     for further information
```

```
 Credits:
CWP: Shuki

 This program belongs to the Center for Wave Phenomena
 Colorado School of Mines


------
 MAXINTS - Compute maximum and minimum sizes for integer types
(quick and dirty)

 Usage:  maxints

 Note: These results will be in limits.h on most systems




 Credits:
CWP: Jack K. Cohen

 This program belongs to the Center for Wave Phenomena
 Colorado School of Mines


------
 PAUSE - prompt and wait for user signal to continue

 Usage:   pause [optional arguments]

 Note:
 Default prompt is "press return key to continue" which is *evoked
 by calling pause with no arguments.  The word,
 "continue", is replaced by any optional arguments handed to pause.
 Thus, the command "pause do plot" will evoke the prompt,
 "press return key to do plot".



------
 T - time and date for non-military types
```

```
 Usage: t

 Credit: Jack


------
 UPFORT - change Fortran programs to upper case, preserving strings

 Usage:   upfort < infile.f > outfile.f

 Reverse of: downfort
```

```
 A2B - convert ascii floats to binary

 a2b <stdin >stdout outpar=/dev/null

 Required parameters:
  none

 Optional parameters:
  n1=2 floats per line in input file

  outpar=/dev/null output parameter file, contains the
number of lines (n=)
  other choices for outpar are: /dev/tty,
  /dev/stderr, or a name of a disk file

 Credits:
CWP: Jack K. Cohen, Dave Hale
Hans Ecke 2002: Replaced line-wise file reading via gets() with
float-wise reading via fscanf(). This makes it
much more robust: it does not impose a specific
structure on the input file.
```

```
 A2I - convert Ascii to binary Integers

 a2i <stdin >stdout outpar=/dev/tty

 Required parameters:
  none

 Optional parameters:
  n1=2 integers per line in input file

  outpar=/dev/tty output parameter file, contains the
number of lines (n=)
```

```
 ADDRVL3D - Add a random velocity layer (RVL) to a gridded
            v(x,y,z) velocity model

addrvl3d <infile n1= n2= >outfile [parameters]

 Required Parameters:
 n1= number of samples along 1st dimension
 n2= number of samples along 2nd dimension

 Optional Parameters:

 n3=1           number of samples along 3rd dimension

 mode=1              add single layer populated with random vels
                     =2 add nrvl layers of random thickness and vel
 seed=from_clock     random number seed (integer)

 ---->New layer geometry info
 i1beg=1        1st dimension beginning sample
 i1end=n1/5     1st dimension ending sample
 i2beg=1        2nd dimension beginning sample
 i2end=n2       2nd dimension ending sample
 i3beg=1        3rd dimension beginning sample
 i3end=n3       3rd dimension ending sample
 ---->New layer velocity info
 vlsd=v/3      range (std dev) of random velocity in layer,
                where v=v(0,0,i1) and i1=(i1beg+i1end)/2
 add=1          add random vel to original vel (v_orig) at that point
                =0 replace vel at that point with (v_orig+v_rand)
 how=0          random vels can be higher or lower than v_orig
                =1 random vels are always lower than v_orig
                =2 random vels are always higher than v_orig
 cvel=2000      layer filled with constant velocity cvel
                (overides vlsd,add,how params)
 ---->Smoothing parameters (0 = no smoothing)
 r1=0.0 1st dimension operator length in samples
 r2=0.0 2nd dimension operator length in samples
 r3=0.0 3rd dimension operator length in samples
 slowness=0 =1 smoothing on slowness; =0 smoothing on velocity

 nrvl=n1/10     number of const velocity layers to add
 pdv=10.        percentage velocity deviation (max) from input model
```

```
Notes:
1. Smoothing radii usually fall in the range of [0,20].
2. Smoothing radii can be used to set aspect ratio of random velocity
   anomalies in the new layer.  For example (r1=5,r2=0,r3=0) will
   result in vertical vel streaks that mimick vertical fracturing.
3. Smoothing on slowness works better to preserve traveltimes relative
   to the unsmoothed case.
4. Default case is a random velocity (+/-30%) near surface layer whose
   thickness is 20% of the total 2D model thickness.
5. Each layer vel is a random perturbation on input model at that level.
6. The depth dimension is assumed to be along axis 1.

Example:
1. 2D RVL with no smoothing
  makevel nz=250 nx=200 | addrvl3d n1=250 n2=200 | ximage n1=250
2. 3D RVL with no smoothing
  makevel nz=250 nx=200 ny=220 |
  addrvl3d n1=250 n2=200 n3=220 |
  xmovie n1=250 n2=200



Author:  Saudi Aramco: Chris Liner Jan/Feb 2005
         Based on smooth3d (CWP: Zhenyue Liu  March 1995)
```

```
 B2A - convert binary floats to ascii

 b2a <stdin >stdout

 Required parameters:
  none

 Optional parameters:
  n1=2 floats per line in output file
       format=0 scientific notation
  =1 long decimal float form

  outpar=/dev/tty output parameter file, contains the
number of lines (n=)
                       other choices for outpar are: /dev/tty,
                       /dev/stderr, or a name of a disk file

 Note:
 Parameter: ",
  format=0 uses printf("%15.10e ", x[i1])
  format=1 uses printf("%15.15f ", x[i1])

 Credits:
CWP: Jack K. Cohen
```

```
 CELLAUTO - Two-dimensional CELLular AUTOmata

   cellauto > stdout [optional params]

 Optional Parameters:
 n1=500 output dimensions of image (n1 x n1 pixels)
 rule=30 CA rule (Wolfram classification)
  Others: 54,60,62,90,94,102,110,122,126
                       150,158,182,188,190,220,222,225,226,250
 fill=0 Don't fill image (=1 fill image)
 f0=330 fill zero values with f0
 f1=3000 fill non-zero values with f1
 ic=1 initial condition for centered unit value at t=0
               = 2 for multiple random units
 nc=20 number of random units (if ic=2)
 tc=1 random initial units at t=0 (if ic=2)
               = 2 for initial units at random (t,x)
 verbose=0 silent operation
               = 1 echos 'porosity' of the CA in bottom half of image
 seed=from_clock     random number seed (integer)

 Notes:
 This program generates a select set of Wolframs fundamental cellular
 automata. This may be useful for constructing rough, vuggy wavespeed
 profiles. The numbering scheme follows Stephen Wolfram's.

 Example:
  cellauto rule=110 ic=2 nc=100 fill=1 f1=3000 | ximage n1=500 nx=500 &

 Here we simulate a complex near surface with air-filled
 vugs in hard country rock, with smoothing applied via smooth2

  cellauto rule=110 ic=2 nc=100 fill=1 f1=3000 n1=500 |
  smooth2 n1=500 n2=500 r1=5 r2=5 > vfile.bin


 Credits:
UHouston: Chris Liner

 Trace header fields accessed:  ns
 Trace header fields modified:  ns and delrt
```

```
char* sdoc[] = {

   CPFTREND - generate picks of the Cumulate Probability Function

 Required parameters:
  ix=      - column containing X variable
  iy=      - column containing Y variable
  min_x=   - minimum X bin
  max_x=   - maximum X bin
  min_y=   - minimum Y bin
  max_y=   - maximum Y bin

 Optional parameters:
  nx=100    - number of X bins
  ny=100    - number of Y bins
  logx=0   - =1 use logarithmic scale for X axis
  logy=0   - =1 use logarithmic scale for Y axis
  ir=       - column containing reject variable
  rmin=     - reject values below rmin
  rmax=     - reject values above rmax
              NOTE: only one, rmin or rmax, may be used
 NOTES:
  cpftrend makes picks on the 2D cumulate representing the
  probability density function of the input data.

   Commandline options allow selecting any of several normalizations
   to apply to the distributions.


  cpftrend(1) makes picks on the 2D cumulate representing the
  probability density function of the input data.

   Commandline options allow selecting any of several normalizations
   to apply to the distributions.

 Credits:  Reginald H. Beardsley                      rhb@acm.org
     Copyright 2006 Exploration Software Consultants Inc.
```

```
 CSHOTPLOT - convert CSHOT data to files for CWP graphers

 cshotplot <cshot1plot [optional parameter file]

 Required parameters:
  none

 Optional parameter:
  outpar=/dev/tty output parameter file, contains:
number of plots (n2=)
points in each plot (n1=)
colors for plots (linecolor=)
```

```
 DZDV - determine depth derivative with respect to the velocity ",
  parameter, dz/dv,  by ratios of migrated data with the primary
  amplitude and those with the extra amplitude

 dzdv <infile afile=afile dfile=dfile>outfile [parameters]

 Required Parameters:
 infile= input of common image gathers with primary amplitude
 afile= input of common image gathers with extra amplitude
 dfile= output of imaged depths in common image gathers
 outfile= output of dz/dv at the imaged points
 nx=          number of migrated traces
 nz=         number of points in migrated traces
 dx= horizontal spacing of migrated trace
 dz=         vertical spacing of output trace
 fx=         x-coordinate of first migrated trace
 fz=         z-coordinate of first point in migrated trace
 off0=         first offset in common image gathers
 noff=         number of offsets in common image gathers
 doff=         offset increment in common image gathers
 cip=x1,z1,r1,..., cip=xn,zn,rn         description of input CIGS
x x-value of a common image point
z z-value of a common image point at zero offset
r r-parameter in a common image gather

 Optional Parameters:
 nxw, nzw=0 window widths along x- and z-directions in
which points are contributed in solving dz/dv.


 Notes:
 This program is used as part of the velocity analysis technique developed
 by Zhenyue Liu, CWP:1995.

 Author: CWP: Zhenyue Liu,  1995

 Reference:
 Liu, Z. 1995, "Migration Velocity Analysis", Ph.D. Thesis, Colorado
      School of Mines, CWP report #168.
```

```
 FARITH - File ARITHmetic -- perform simple arithmetic with binary files

 farith <infile >outfile [optional parameters]

 Optional Parameters:
 in=stdin input file
 out=stdout output file
 in2=    second input file (required for binary operations)
   if it can't be opened as a file, it might be a scalar
 n=size_of_in,  fastest dimension (used only for op=cartprod is set)
 isig=  index at which signum function acts (used only for
op=signum)
 scale= value to scale in by, used only for op=scale)
 bias= value to bias in by, used only for op=bias)

 op=noop   noop for out = in
   neg  for out = -in
   abs  for out = abs(in)
   scale for out = in *scale
   bias for out = in + bias
   exp  for out = exp(in)
   sin  for out = sin(in)
   cos  for out = cos(in)
   log  for out = log(in)
   sqrt for out = (signed) sqrt(in)
   sqr  for out = in*in
   degrad  for out = in*PI/180
   raddeg  for out = in*180/PI
   pinv  for out = (punctuated) 1 / in
   pinvsqr  for out = (punctuated) 1 /in*in
   pinvsqrt for out = (punctuated signed) 1 /sqrt(in)
   add  for out = in + in2
   sub  for out = in - in2
   mul  for out = in * in2
   div  for out = in / in2
cartprod for out = in x in2
requires: n=size_of_in, fastest dimension in output
signum for out[i] = in[i] for i< isig  and
= -in[i] for i>= isig
requires: isig=point where signum function acts
 Seismic operations:
   sloth   for  out =  1/in^2 Sloth from velocity in
   slowp   for  out =  1/in - 1/in2 Slowness perturbation
```

```
  slothp  for  out =  1/in^2 - 1/in2^2   Sloth perturbation

Notes:
op=sqrt takes sqrt(x) for x>=0 and -sqrt(ABS(x)) for x<0 (signed sqrt)

op=pinv takes y=1/x for x!=0,  if x=0 then y=0. (punctuated inverse)

The seismic operations assume that in and in2 are wavespeed profiles.
"Slowness" is 1/wavespeed and "sloth" is  1/wavespeed^2.
Use "suop" and "suop2" to perform unary and binary operations on
data in the SU (SEGY trace) format.

The options "pinvsq" and "pinvsqrt" are also useful for seismic
computations involving converting velocity to sloth and vice versa.

The option "cartprod" (cartesian product) requires also that the
parameter n=size_of_in be set. This will be the fastest dimension
of the rectangular array that is output.

The option "signum" causes a flip in sign for all values with index
greater than "isig" (really -1*signum(index)).

For file operations on SU format files, please use:  suop, suop2



  AUTHOR:  Dave Hale, Colorado School of Mines, 07/07/89
Zhaobo Meng added scale and cartprod, 10/01/96
Zhaobo Meng added signum, 9 May 1997
Tony Kocurko added scalar operations, August 1997
      John Stockwell added bias option 4 August 2004
```

```
 FLOAT2IBM - convert native binary FLOATS to IBM tape FLOATS

 float2ibm <stdin >stdout

 Required parameters:
  none

 Optional parameters:
 endian= byte order of your system (autodetected)
 outpar=/dev/tty output parameter file, contains the
number of values (n=)
       other choices for outpar are: /dev/tty,
       /dev/stderr, or a name of a disk file

 Notes:
 endian=1 (big endian) endian=0 (little endian) byte order
 You probably will not have to set this, as the byte order of
 your system is autodetected by the program.
 This program is usable for writing SEG Y traces with the headers
 stripped off.

 Credits:
CWP: John Stockwell, based on code by Jack K. Cohen
```

```
 FTNSTRIP - convert a file of binary data plus record delimiters created
      via Fortran to a file containing only binary values (as created via C)

 ftnstrip <ftn_data >c_data

 Caveat: this code assumes the conventional Fortran format of header
         and trailer integer containing the number of byte in the
         record.  This is overwhelmingly common, but not universal.


 Credits:
CWP: Jack K. Cohen
```

```
 FTNUNSTRIP - convert C binary floats to Fortran style floats

 ftnunstrip <stdin >stdout

 Required parameters:
  none

 Optional parameters:
  n1=1 floats per line in output file

  outpar=/dev/tty output parameter file, contains the
number of lines (n=)
  other choices for outpar are: /dev/tty,
  /dev/stderr, or a name of a disk file

 Notes: This program assumes that the record length is constant
 throughout the input and output files.
 In fortran code reading these floats, the following implied
 do loop syntax would be used:
        DO i=1,n2
                 READ (10) (someARRAY(j), j=1,n1)
        END DO
 Here n1 is the number of samples per record, n2 is the number
 of records, 10 is some default file (fort.10, for example), and
 someArray(j) is an array dimensioned to size n1


 Credits:
CWP: John Stockwell, Feb 1998,
            based on ftnstrip by: Jack K. Cohen
```

```
 GRM - Generalized Reciprocal refraction analysis for a single layer

     grm <stdin >stdout  [parameters]

 Required parameters:
 nt= Number of arrival time pairs
 dx= Geophone spacing (m)
 v0= Velocity in weathering layer (m/s)
 abtime= If set to 0, use last time as a-b, else give time (ms)

 Optional parameters:
 XY=      Value of XY if you want to override the optimum XY
  algorithm in the program. If it is not an integer multiple of
 dx, then it will be converted to the closest
 one.
XYmax   Maximum offset distance allowed when searching for
optimum XY (m)  (Default is 2*dx*10)
depthres  Size of increment in x during verical depth search(m)
  (Default is 0.5m)
 Input file:
4 column ASCII - x,y, forward time, reverse time
 Output file:
1) XYoptimum
2) apparent refractor velcocity
3) x, y, z(x,y), y-z(x,y)
z(x,y) = calculated (GRM) depth below (x y)
y-z(x,y) = GRM depth subtracted from y - absolute depth
      ..............
      4) x, y, d(x,y), y-d(x,y), (error)
d(x,y) = dip corrected depth estimate below (x,y)
y-d(x,y) = dip corrected absolute depth
error = estimated error in depth due only to the inexact
      matching of tangents to arcs in dip estimate.

      If the XY calculation is bypassed and XY specified, the values
      used will precede 1) above.  XYoptimum will still be calculated
      and displayed for reference.

 Notes:
      Uses average refactor velocity along interface.

  Credits:
```

CWP: Steven D. Sheaffer

D. Palmer, "The Generalized Reciprocal Method of Seismic Refraction Interpretation", SEG, 1982.

```
 H2B - convert 8 bit hexidecimal floats to binary

 h2b <stdin >stdout outpar=/dev/tty

 Required parameters:
  none

 Optional parameters:
  outpar=/dev/tty output parameter file, contains the
number of lines (n=)
  other choices for outpar are: /dev/tty,
  /dev/stderr, or a name of a disk file

 Note: this code may be used to recover binary data from PostScript
 bitmaps. To do this, strip away all parts of the PSfile that
 are not the actual hexidecimal bitmap and run through h2b.

 Note: that the binary file may need to be transposed using
 "transp" to appear to be the same as input data.

 Note: output will be floats with the values 0-255
```

```
 HTI2STIFF - convert HTI parameters alpha, beta, d(V), e(V), gamma
into stiffness tensor

   hti2stiff  [optional parameter] (output is to   outpar)

 Optional Parameters
 alpha=2     isotropy-plane p-wave velocity
 beta=1      fast isotropy-plan s-wave velocity
 ev=0 e(V)
 dv=0 d(V)
 gamma=0     shear-wave splitting parameter
 rho=1 density
 sign      sign of c13+c55 ( for most materials sign=1)
 outpar=/dev/tty    output parameter file

 Output:
  c_ijkl     stiffness components for x1=symmetry axis
    x3= vertical



 Credits:   Andreas Rueger, CWP Aug 01, 1996

 Reference: Andreas Rueger, P-wave reflection coefficients for
    transverse isotropy with vertical and horizontal axis
    of symmetry,  GEOPHYSICS
```

```
 HUDSON - compute  effective parameters of anisotropic solids
  using Hudson's crack theory.

 Required paramters: <none>

 Optional parameters

 vp=4.5        p-wave velocity uncracked solid
 vs=2.53       s-wave velocity uncracked solid
 rho=2.8       density
 cdens=0 crack density
 aspect=0      aspect ratio
 fill=0        gas filled cracks
               =1 water filled
 outpar        =/dev/tty   output file

Notes:
 The cracks are assumed to be vertically aligned, penny-shaped and the
 matrix is isotropic. The resulting anisotropic solid is of HTI symmetry.

 Output:
 Computes(a) stiffness elements
         (b) density normalized stiffness components
         (c) generic Thomsen parameters (vp0,vs0,eps,delta,gamma)
         (d) equivalent VTI parameters (alpha,beta,ev,dv,gv)




 AUTHOR:: Andreas Rueger, Colorado School of Mines, 10/10/96

 Additional notes:
  The routine can be easily modified to allow for any
  filling adding attenuation is not trivial

 Technical Reference:
  Hudson's theory: Hudson, 1981: Wave speed and attenuation of elastic
                                 waves in material containing cracks.
                                 Geophys. J. R. astr. Soc 64, 133-150
                   Crampin, 1984: Effective anisotropic elastic constants
                                  for waves propagating through cracked
                                  solids:
```

Geophys. J. R. astr. Soc 76, 135-145
Equivalent VTI : Rueger, 1996: Reflection coefficients in transversely
isotropic media with vertical and
horizontal axis of symmetry: Geophysics

```
 I2A - convert binary integers to ascii

 i2a <stdin >stdout

 Required parameters:
  none

 Optional parameters:
  n1=2 floats per line in output file

  outpar=/dev/tty output parameter file, contains the
number of lines (n=)


 Credits:
 Potash Corporation: c. 2008, Balazs Nemeth,  Saskatoon, Saskatchewan.
   based on b2a.c by:  CWP: Jack K. Cohen
```

```
 IBM2FLOAT - convert IBM tape FLOATS to native binary FLOATS

 ibm2float <stdin >stdout

 Required parameters:
  none

 Optional parameters:
 endian= byte order of your system (autodetected)
 outpar=/dev/tty output parameter file, contains the
number of values (n=)
       other choices for outpar are: /dev/tty,
       /dev/stderr, or a name of a disk file

 Notes:
 endian=1 (big endian) endian=0 (little endian) byte order
 You probably will not have to set this, as the byte order of
 your system is autodetected by the program.
 This program is usable for reading SEG Y files with the headers
 stripped off.

 Credits:
CWP: John Stockwell, based on code by Jack K. Cohen
```

```
KAPERTURE - generate the k domain of a line scatterer for a seismic array

kaperture [optional parameters] >stdout

Optional parameters
 x0=1000 point scatterer location
 z0=1000 point scatterer location
 nshot=1 number of shots
 sxmin=0 first shot location
 szmin=0 first shot location
 dsx=100 x-steps in shot location
 dsz=0 z-steps in shot location
 ngeo=1 number of receivers
 gxmin=0 first receiver location
 gzmin=0 first receiver location
 dgx=100 x-steps in receiver location
 dgz=0 z-steps in receiver location
 fnyq=125 Nyquist frequency  (Hz)
 fmax=125 maximum frequency  (Hz)
 fmin=5 minimum frequency  (Hz)
 nfreq=2 number of frequencies
 both=0 = 1 gives negative freqs too
 nstep=60 points on Nyquist circle
 c=5000 speed
 outpar=/dev/tty output parameter file, contains:
 xmin, xmax, ymin, ymax
 and npairs (needed for psgraph or xgraph)
 other choices for outpar are: /dev/tty,
 /dev/stderr, or a name of a disk file
Notes:
      nfreq=1 produces fmin
      nstep=0 suppresses the Nyquist circle
 and npairs
Examples:

Default case: both=0 nfreq=2

 kaperture nshot=NSHOT ngeo=NGEO nstep=NSTEP |
 psgraph n=NPAIRS,NSTEP mark=1,0 marksize=1,0 linewidth=0,1 |...
 WHERE: NPAIRS=NSHOT*NGEO

Other cases:
```

```
both=0 nfreq=NFREQ > 2
 kaperture both=0 nfreq=NFREQ nshot=NSHOT ngeo=NGEO nstep=NSTEP |
 psgraph n=NPAIRS,NSTEP mark=1,0 marksize=1,0 linewidth=0,1 |...
 WHERE: NPAIRS=NFREQ*NSHOT*NGEO

both=1 nfreq=NFREQ > 2
 kaperture both=1 nfreq=NFREQ nshot=NSHOT ngeo=NGEO nstep=NSTEP |
 psgraph n=NPAIRS,NSTEP mark=1,0 marksize=1,0 linewidth=0,1 |...
  WHERE: NPAIRS=NFREQ*NSHOT*NGEO*2

When in doubt to the size of NPAIRS, redirect output of kaperture to
/dev/tty the first time to get npairs=:
kaperture [optional parameters] > /dev/tty
```

```
 LINRORT - linearized P-P, P-S1 and P-S2 reflection coefficients
for a horizontal interface separating two of any of the
following halfspaces: ISOTROPIC, VTI, HTI and ORTHORHOMBIC.

   linrort [optional parameters]

 hspace1=ISO medium type of the incidence halfspace:
=ISO ... isotropic
=VTI ... VTI anisotropy
=HTI ... HTI anisotropy
=ORT ... ORTHORHOMBIC anisotropy
 for ISO:
 vp1=2   P-wave velocity, halfspace1
 vs1=1 S-wave velocity, halfspace1
 rho1=2.7 density, halfspace1

 for VTI:
 vp1=2 P-wave vertical velocity (V33), halfspace1
 vs1=1 S-wave vertical velocity (V44=V55), halfspace1
 rho1=2.7 density, halfspace1
 eps1=0 Thomsen's generic epsilon, halfspace1
 delta1=0 Thomsen's generic delta, halfspace1
 gamma1=0 Thomsen's generic gamma, halfspace1 ",

 for HTI:
 vp1=2  P-wave vertical velocity (V33), halfspace1
 vs1=1  "fast" S-wave vertical velocity (V44), halfspace1
 rho1=2.7 density, halfspace1
 eps1_v=0 Tsvankin's "vertical" epsilon, halfspace1
 delta1_v=0 Tsvankin's "vertical" delta, halfspace1
 gamma1_v=0 Tsvankin's "vertical" gamma, halfspace1 ",

 for ORT:
 vp1=2 P-wave vertical velocity (V33), halfspace1
 vs1=1  x2-polarized S-wave vertical velocity (V44), halfspace1
 rho1=2.7 density, halfspace1
 eps1_1=0 Tsvankin's epsilon in [x2,x3] plane, halfspace1
 delta1_1=0 Tsvankin's delta in [x2,x3] plane, halfspace1
 gamma1_1=0 Tsvankin's gamma in [x2,x3] plane, halfspace1
 eps1_2=0 Tsvankin's epsilon in [x1,x3] plane, halfspace1
 delta1_2=0 Tsvankin's delta in [x1,x3] plane, halfspace1
 gamma1_2=0 Tsvankin's gamma in [x1,x3] plane, halfspace1
 delta1_3=0 Tsvankin's delta in [x1,x2] plane, halfspace1
```

```
 hspace2=ISO medium type of the reflecting halfspace (the same
convention as above)

 medium parameters of the 2nd halfspace follow the same convention
 as above:

 vp2=2.5  vs2=1.2 rho2=3.0
 eps2=0   delta2=0
 eps2_v=0 delta2_v=0 gamma2_v=0
 eps2_1=0 delta2_1=0 gamma2_1=0
 eps2_2=0 delta2_2=0 gamma2_2=0
 delta2_3=0

(note you do not need "gamma2" parameter for evaluation
of weak-anisotropy reflection coefficients)

 a_file=-1 the string '-1' ... incidence and azimuth angles are
generated automatically using the setup values below
a_file=file_name ... incidence and azimuth angles are
read from a file "file_name"; the program expects a
file of two columns [inc. angle, azimuth]

 in the case of a_file=-1:
 fangle=0 first incidence phase angle
 langle=30 last incidence angle
 dangle=1 incidence angle increment
 fazim=0 first azimuth (in deg)
 lazim=0 last azimuth  (in deg)
 dazim=1 azimuth increment (in deg)

 kappa=0. azimuthal rotation of the lower halfspace2 (e.t. a
symmetry axis plane for HTI, or a symmetry plane for
ORTHORHOMBIC) with respect to the x1-axis

 out_inf=info.out information output file
 out_P=Rpp.out file with Rpp reflection coefficients
 out_S=Rps.out file with Rps reflection coefficients
 out_SVSH=Rsvsh.out  file with SV and SH projections of reflection
coefficients
 out_Error=error.out file containing error estimates evaluated during
the computation of the reflection coefficients;
```

```
Output:
out_P:
inc. phase angle, azimuth, reflection coefficient; for a_file=-1, the
inc. angle is the fast dimension
out_S:
inc. phase angle, azimuth, Rps1, Rps2, cos(PHI), sin(PHI); for
a_file=-1, the inc. angle is the fast dimension ",
out_SVSH:
inc. phase angle, azimuth, Rsv, Rsh, cos(PHI), sin(PHI); for
a_file=-1, the inc. angle is the fast dimension
out_Error:
error estimates of Rpp, Rpsv and Rpsh approximations; global error is
analysed as well as partial contributions to the error due to the
isotropic velocity contrasts, and due to anisotropic  upper and lower
halfspaces. The error file is self-explanatory, see also descriptions
of subroutines P_err_2nd_order, SV_err_2nd_order and SH_err_2nd_order.


Adopted Convention:

The right-hand Cartesian coordinate system with the x3-axis pointing
upward has been chosen. The upper halfspace (halfspace1)
contains the incident P-wave. Incidence angles can vary from <0,PI/2),
azimuths are unlimited, +azimuth sense counted from x1->x2 axes
(azimuth=0 corresponds to the direction of x1-axis). In the current
version, the coordinate system is attached to the halfspace1 (e.t.
the symmetry axis plane of HTI halfspace1, or one of symmetry planes
of ORTHORHOMBIC halfspace1, is aligned with the x1-axis), however, the
halfspace2 can be arbitrarily rotated along the x3-axis with respect
to the halfspace1. The positive weak-anisotropy polarization of the
reflected P-P wave (e.t. positive P-P reflection coefficient) is close
to the direction of isotropic slowness vector of the wave (pointing
outward the interface). Similarly, weak-anisotropy S-wave reflection
coefficients are described in terms of "SV" and "SH" isotropic
polarizations, "SV" and "SH" being unit vectors in the plane
perpendicular to the isotropic slowness vector. Then, the positive
"SV" polarization vector lies in the incidence plane and points
towards the interface, and positive "SH" polarization vector is
perpendicular to the incidence plane, aligned with the positive
x2-axis, if azimuth=0. Rotation angle "PHI", characterizing a
rotation of "the best projection" of the S1-wave polarization
vector in the isotropic SV-SH plane in the incidence halfspace1, is
```

```
 counted in the positive sense from "SV" axis (PHI=0) towards the
 "SH" axis (PHI=PI/2). Of course, S2 is perpendicular to S1, and
 the projection of S1 and S2 polarizations onto the SV-SH plane
 coincides with SV and SH directions, respectively, for PHI=0.

 The units for velocities are km/s, angles I/O are in degrees

 Additional Notes:
The coefficients are computed as functions of phase incidence
angle and azimuth (determined by the incidence slowness vector).
Vertical symmetry planes of the HTI and
ORTHORHOMBIC halfspaces can be arbitrarily rotated along the
x3-axis. The linearization is based on the assumption of weak ",
contrast in elastic medium parameters across the interface,
and the assumption of weak anisotropy in both halfspaces.
See the "Adopted Convention" paragraph below for a proper
input.


  Author: Petr Jilek, CSM-CWP, December 1999.
```

```
 LORENZ - compute the LORENZ attractor

  lorenz > [stdout]

 Required Parameters: none
 Optional Parameters:
 rho=28.0 parameter for lorenz equations
 sigma=10.0 parameter for lorenz equations
 eta=1.6666667 parameter for lorenz equations
 y0=1.0 initial value of y[0]
 y1=-1.0 initial value of y[1]
 y2=1.0 initial value of y[2]
 h=.01 increment in time
 tol=1.e-08 error tolerance
 stepmax=500 maximum number of steps to compute
 mode=xy xy-pairs, =yz yz-pairs, =xz xz-pairs,
=xyz xyz-triplet, =x only, =y only, =z only
 Notes:
 This program is really just a demo showing how to use the
 differential equation solver rke_solve written by Francois
 Pinard, based on a modified form of the 4th order Runge-Kutta
 method, which employs the error checking method of R. England
 1969.

 The output consists of unformated C-style binary floats, of
 either pairs or triplets as specified by the "mode" paramerter.

 Examples:
 lorenz stepmax=1000 mode=xy | xgraph n=1000 &
 lorenz stepmax=1000 mode=yz | xgraph n=1000 &
 lorenz stepmax=1000 mode=xz | xgraph n=1000 &

 lorenz stepmax=1000 mode=x | suaddhead ns=1000 | suxwigb &
 lorenz stepmax=1000 mode=y | suaddhead ns=1000 | suxwigb &
 lorenz stepmax=1000 mode=z | suaddhead ns=1000 | suxwigb &

 GNUPLOT 3D plot example:
 lorenz stepmax=2000 mode=xyz > lorenz.bin
 ...when you run gnuplot type the following command ...
 splot "lorenz.bin" binary record=2000:2000:2000 with points pointsize .1
```

```
 The lorenz equations describe a simplified model of a convection
 cell, and are given by the autonomous system of ODE's

x'(t) = sigma * ( y - x )
y'(t) = x * ( rho - z ) - y
z'(t) = x * y - eta * z

 Author: CWP: Aug 2004: John Stockwell
```

```
MAKEVEL - MAKE a VELocity function v(x,y,z)

makevel > outfile nx= nz= [optional parameters]

Required Parameters:
nx=                     number of x samples (3rd dimension)
nz=                     number of z samples (1st dimension)

Optional Parameters:
ny=1                    number of y samples (2nd dimension)
dx=1.0                  x sampling interval
fx=0.0                  first x sample
dy=1.0                  y sampling interval
fy=0.0                  first y sample
dz=1.0                  z sampling interval
fz=0.0                  first z sample
v000=2.0                velocity at (x=0,y=0,z=0)
dvdx=0.0                velocity gradient with respect to x
dvdy=0.0                velocity gradient with respect to y
dvdz=0.0                velocity gradient with respect to z
vlens=0.0               velocity perturbation in parabolic lens
tlens=0.0               thickness of parabolic lens
dlens=0.0               diameter of parabolic lens
xlens=                  x coordinate of center of parabolic lens
ylens=                  y coordinate of center of parabolic lens
zlens=                  z coordinate of center of parabolic lens
lambda=1.0              make lambda larger to sharpen edge of lens
vran=0.0 standard deviation of random perturbation
vzfile=                 file containing v(z) profile
vzran=0.0               standard deviation of random perturbation to v(z)
vzc=0.0                 v(z) chirp amplitude
z1c=fz                  z at which to begin chirp
z2c=fz+(nz-1)*dz        z at which to end chirp
l1c=dz                  wavelength at beginning of chirp
l2c=dz                  wavelength at end of chirp
exc=1.0                 exponent of chirp
```

```
 MKPARFILE - convert ascii to par file format

 mkparfile <stdin >stdout

 Optional parameters:
  string1="par1" first par string
  string2="par2" second par string

 This is a tool to convert values written line by line to parameter
 vectors in the form expected by getpar.  For example, if the input
 file looks like:
  t0 v0
  t1 v1
...
 then
mkparfile <input >output string1=tnmo string2=vnmo
 yields:
tnmo=t0,t1,...
vnmo=v0,v1,...
```

```
 MRAFXZWT - Multi-Resolution Analysis of a function F(X,Z) by Wavelet
 Transform. Modified to perform different levels of resolution
        analysis for each dimension and also to allow to transform
        back only the lower level of resolution.

    mrafxzwt [parameters] < infile > mrafile

 Required Parameters:
 n1= size of first (fast) dimension
 n2= size of second (slow) dimension

 Optional Parameters:
 p1= maximum integer such that 2^p1 <= n1
 p2= maximum integer such that 2^p2 <= n2
 order=6 order of Daubechies wavelet used (even, 4<=order<=20)
 mralevel1=3   maximum multi-resolution analysis level in dimension 1
 mralevel2=3   maximum multi-resolution analysis level in dimension 2
 trunc=0.0 truncation level (percentage) of the reconstruction
 verbose=0 =1 to print some useful information
 reconfile=    reconstructed data file to write
 reconmrafile= reconstructed data file in MRA domain to write
 dfile= difference between infile and reconfile to write
 dmrafile= difference between mrafile and reconmrafile to write
 dconly=0      =1 keep only dc component of MRA
 verbose=0     =1 to print some useful information
 if (n1 or n2 is not integer powers of 2) specify the following:
  nc1=n1/2 center of trimmed image in the 1st dimension
  nc2=n2/2 center of trimmed image in the 2nd dimension
trimfile= if given, output the trimmed file

 Notes:
 This program performs multi-resolution analysis of an input function
 f(x,z) via the wavelet transform method. Daubechies's least asymmetric
 wavelets are used. The smallest wavelet coefficient retained is given
 by trunc times the absolute maximum size coefficient in the MRA.

 The input dimensions of the data must be expressed by (p1,p2) which

  Author: Zhaobo Meng, 11/25/95, Colorado School of Mines             *
  Modified:  Carlos E. Theodoro, 06/25/97, Colorado School of Mines   *
Included options for:                                    *
- different level of resolutionf or each dimension;            *
- transform back the lower level of resolution, only. *
```

```
*
 Reference: *
 Daubechies, I., 1988, Orthonormal Bases of Compactly Supported *
 Wavelets, Communications on Pure and Applied Mathematics, Vol. XLI,  *
 909-996.   *
```

```
 PDFHISTOGRAM - generate a HISTOGRAM of the Probability Density function

  pdfhistogram < stdin > sdtout [Required params] (Optional params)

 Required parameters:
 ix= column containing X variable
 iy= column containing Y variable
 min_x= minimum X bin
 max_x= maximum X bin
 min_y= minimum Y bin
 max_y= maximum Y bin
 logx=0 =1 use logarithmic scale for X axis
 logy=0 =1 use logarithmic scale for Y axis
 norm= selected normalization type
sqrt - bin / sqrt( xnct*ycnt)
avg_cnt - 0.5* bin / (xcnt + ycnt)
avg_sum - (bin / xcnt + bin / ycnt ) / 2
xcnt - bin / xcnt
ycnt - bin / ycnt
log - log(bin)
total - bin / total
 Optional parameters:
  nx=100 - number of X bins
  ny=100 - number of Y bins
  ir= - column containing reject variable
  rmin= - reject values below rmin
  rmax= - reject values above rmax
NOTE: only one, rmin or rmax, may be used
 Notes:
 PDFHISTOGRAM creates a 2D histogram representing the probability density
 function of the input data. The output is in the form of a binary array
 that can then be plotted via ximage.
 Commandline options allow selecting any of several normalizations
 to apply to the distributions.


 Credits:
  Reginald H. Beardsley rhb@acm.org
Copyright 2006 Exploration Software Consultants Inc.
```

```
PRPLOT - PRinter PLOT of 1-D arrays f(x1) from a 2-D function f(x1,x2)

prplot <infile >outfile [optional parameters]

Optional Parameters:
n1=all                  number of samples in 1st dimension
d1=1.0                  sampling interval in 1st dimension
f1=d1                   first sample in 1st dimension
n2=all                  number of samples in 2nd dimension
d2=1.0                  sampling interval in 2nd dimension
f2=d2                   first sample in 2nd dimension
label2=Trace            label for 2nd dimension
```

```
 RANDVEL3D - Add a random velocity layer (RVL) to a gridded
           v(x,y,z) velocity model

randvel3d <infile n1= n2= >outfile [parameters]

 Required Parameters:
 n1= number of samples along 1st dimension
 n2= number of samples along 2nd dimension

 Optional Parameters:

 n3=1          number of samples along 3rd dimension

 mode=1             add single layer populated with random vels
                    =2 add nrvl layers of random thickness and vel
 seed=from_clock    random number seed (integer)

 ---->New layer geometry info
 i1beg=1       1st dimension beginning sample
 i1end=n1/5    1st dimension ending sample
 i2beg=1       2nd dimension beginning sample
 i2end=n2      2nd dimension ending sample
 i3beg=1       3rd dimension beginning sample
 i3end=n3      3rd dimension ending sample
 ---->New layer velocity info
 vlsd=v/3     range (std dev) of random velocity in layer,
               where v=v(0,0,i1) and i1=(i1beg+i1end)/2
 add=1         add random vel to original vel (v_orig) at that point
               =0 replace vel at that point with (v_orig+v_rand)
 how=0         random vels can be higher or lower than v_orig
               =1 random vels are always lower than v_orig
               =2 random vels are always higher than v_orig
 cvel=2000     layer filled with constant velocity cvel
               (overides vlsd,add,how params)
 ---->Smoothing parameters (0 = no smoothing)
 r1=0.0 1st dimension operator length in samples
 r2=0.0 2nd dimension operator length in samples
 r3=0.0 3rd dimension operator length in samples
 slowness=0 =1 smoothing on slowness; =0 smoothing on velocity

 nrvl=n1/10    number of const velocity layers to add
 pdv=10.       percentage velocity deviation (max) from input model
```

```
Notes:
1. Smoothing radii usually fall in the range of [0,20].
2. Smoothing radii can be used to set aspect ratio of random velocity
   anomalies in the new layer.  For example (r1=5,r2=0,r3=0) will
   result in vertical vel streaks that mimick vertical fracturing.
3. Smoothing on slowness works better to preserve traveltimes relative
   to the unsmoothed case.
4. Default case is a random velocity (+/-30%) near surface layer whose
   thickness is 20% of the total 2D model thickness.
5. Each layer vel is a random perturbation on input model at that level.
6. The depth dimension is assumed to be along axis 1.

Example:
1. 2D RVL with no smoothing
  makevel nz=250 nx=200 | randvel3d n1=250 n2=200 | ximage n1=250
2. 3D RVL with no smoothing
  makevel nz=250 nx=200 ny=220 |
  randvel3d n1=250 n2=200 n3=220 |
  xmovie n1=250 n2=200



Author:  U Houston: Chris Liner c. 2008
         Based on smooth3d (CWP: Zhenyue Liu  March 1995)
```

```
RAYT2DAN -- P- and SV-wave raytracing in 2D anisotropic media

rayt2dan > ttime parameterfiles= nt= nx= nz= [optional parameters]

Required Parameters:
VP0file=  name of file containing VP0(x,z)
nt=                    number of time samples for each ray
nx=                    number of samples (x) for the parameter fields
nz=                    number of samples (z) for the parameter fields

Optional Parameters:
SV=0  for P-waves, SV=1 for Shear waves

Parameters defining the velocity field
dt=0.008                time sampling interval ",
fx=0                    first lateral sample (x) in parameter field
fz=0                    first lateral sample (z) in parameter field
dx=100.0                sample spacing (x) for the parameter fields
dz=100.0                sample spacing (z) for the parameter fields

Parameters defining the takeoff angle of a ray at a source position ",
fa=-60                  first take-off angle of rays (degrees)
na=61                   number of rays ",
da=2                    increment of take-off angle
amin=0                  minimum emergence angle; must be > -90 degrees
amax=90                 maximum emergence angle; must be < 90 degrees

Parameters defining the output traveltime table
fxo=fx                  first lateral sample in traveltime table
nxo=nx                  number of later samples in traveltime table
dxo=dx                  lateral interval in traveltime table
fzo=fz                  first depth sample in traveltime table
nzo=nz                  number of depth samples in traveltime table
dzo=dz                  depth interval in traveltime table
fac=0.01                factor to determine the radius of extrap.

Parameters defining the source positions
fsx=fx                  x-coordinate of first source
nsx=1                   number of sources
dsx=2*dxo               x-coordinate increment of sources
aperx=0.5*nx*dx         ray tracing aperature in x-direction

Files for general anisotropic parameters confined to a vertical plane:
```

```
 a1111file name of file containing a1111(x,z)
 a1133file           name of file containing a1133(x,z)
 VS0file         name of file containing VS0(x,z)
 a1113file           name of file containing a1113(x,z)
 a3313file           name of file containing a3313(x,z)

 For transversely isotropic media Thomsen's parameters could be used:
 deltafile name of file containing delta(x,z)
 epsilonfile name of file containing epsilon(x,z)

 if anisotropy parameters are not given the program considers ",
 isotropic media. ",


 Credits:
      Debashish Sarkar


   Technical Reference:

Cerveny, V., 1972, Seismic rays and ray intensities
in inhomogeneous anisotropic media:
Geophys. J. R. Astr. Soc., 29, 1--13.

  Hangya, A., 1986, Gaussian beams in anisotropic elastic media:
      Geophys. J. R. Astr. Soc., 85, 473--563.

Gajewski, D. and Psencik, I., 1987, Computation of high frequency
seismic wavefields in 3-D  laterally inhomogeneous anisotropic
  media: Geophys. J. R. Astr. Soc., 91, 383-411.
```

```
 RAYT2D - traveltime Tables calculated by 2D paraxial RAY tracing

     rayt2d vfile= tfile= [optional parameters]

 Required parameters:
 vfile=stdin file containning velocity v[nx][nz]
 tfile=stdout file containning traveltime tables
t[nxs][nxo][nzo]

 Optional parameters
 dt=0.008 time sample interval in ray tracing
 nt=401 number of time samples in ray tracing

 fz=0 first depth sample in velocity
 nz=101 number of depth samples in velocity
 dz=100 depth interval in velocity
 fx=0 first lateral sample in velocity
 nx=101 number of lateral samples in velocity
 dx=100 lateral interval in velocity

 fzo=fz first depth sample in traveltime table
 nzo=nz number of depth samples in traveltime table
 dzo=dz depth interval in traveltime table
 fxo=fx first lateral sample in traveltime table
 nxo=nx number of lateral samples in traveltime table
 dxo=dx lateral interval in traveltime table

 surf="0,0;99999,0"  Recording surface "x1,z1;x2,z2;x3,z3;...
 fxs=fx x-coordinate of first source
 nxs=1 number of sources
 dxs=2*dxo x-coordinate increment of sources
 aperx=0.5*nx*dx   ray tracing aperature in x-direction

 fa=-60 first take-off angle of rays (degrees)
 na=61 number of rays
 da=2 increment of take-off angle
 amin=0 minimum emergence angle
 amax=90 maximum emergence angle

 fac=0.01 factor to determine radius for extrapolation
 ek=1 flag of implementing eikonal in shadow zones
 ms=10 print verbal information at every ms sources
 restart=n job is restarted (=y yes; =n no)
```

```
 npv=0 flag of computing quantities for velocity analysis
 if npv>0 specify the following three files
 pvfile=pvfile input file of velocity variation pv[nxo][nzo]
 tvfile=tvfile output file of traveltime variation tables
tv[nxs][nxo][nzo]
 csfile=csfile output file of cosine tables cs[nxs][nxo][nzo]

 Notes:
 1. Each traveltime table is calculated by paraxial ray tracing; then
    traveltimes in shadow zones are compensated by solving eikonal
    equation.
 2. Input velocity is uniformly sampled and smooth one preferred.
 3. Traveltime table and source ranges must be within velocity model.
 4. Ray tracing aperature can be chosen as sum of migration aperature
    plus half of maximum offset.
 5. Memory requirement for this program is about
      [nx*nz+4*mx*nz+3*nxo*nzo+na*(nx*nz+mx*nz+3*nxo*nzo)]*4 bytes
    where mx = min(nx,2*(1+aperx/dx)).

 Note: spatial units of v(z,x) must be the same as those of dx.
 v(z,x) is represented numerically in C-style binary floats v[xn][zn],
 where the depth direction is the fast direction in the data. Such
 models can be created with unif2 or makevel.     ",


 Author:  Zhenyue Liu, 10/11/94,  Colorado School of Mines

          Trino Salinas, 01/01/96 included the option to handle nonflat
          reference surfaces.
          Subroutines from Dave Hale's modeling library were adapted in
          this code to define topography using cubic splines.

 References:

 Beydoun, W. B., and Keho, T. H., 1987, The paraxial ray method:
   Geophysics, vol. 52, 1639-1653.

 Cerveny, V., 1985, The application of ray tracing to the numerical
   modeling of seismic wavefields in complex structures, in Dohr, G.,
   ED., Seismic shear waves (part A: Theory): Geophysical Press,
   Number 15 in Handbook of Geophysical Exploration, 1-124.
```

```
 RECAST - RECAST data type (convert from one data type to another)

 recast <stdin [optional parameters]  >stdout

 Required parameters:
  none

 Optional parameters:
 in=float input type (float)
  =double (double)
  =int (int)
  =char (char)
=uchar (unsigned char)
  =short (short)
  =long (long)
  =ulong (unsigned long)

 out=double output type (double)
  =float (float)
  =int (int)
  =char (char)
  =uchar (unsigned char)
  =short (short)
  =long (long)
  =ulong (unsigned long)

 outpar=/dev/tty output parameter file, contains the
number of values (n1=)
  other choices for outpar are: /dev/tty,
  /dev/stderr, or a name of a disk file

 Notes: Converting bigger types to smaller is hazardous. For float
 or double conversions to integer types, the results are
 rounded to the nearest integer.


 Credits:

CWP: John Stockwell, Jack K. Cohen
```

```
 REFREALAZIHTI -  REAL AZImuthal REFL/transm coeff for HTI media

 refRealAziHTI  [optional parameters] >coeff.data

 Optional parameters:
 vp1=2         p-wave velocity medium 1 (with respect to symm.axes)
 vs1=1         s-wave velocity medium 1 (with respect to symm.axes)
 eps1=0        epsilon medium1
 delta1=0 delta medium 1
 gamma1=0 gamma medium 1
 rho1=2.7 density medium 1
 vp2=2         p-wave velocity medium 2 (with respect to symm.axes)
 vs2=1         s-wave velocity medium 2 (with respect to symm.axes)
 eps2=0        epsilon medium 2
 delta2=0 delta medium 2
 gamma2=0 gamma medium 2
 rho2=2.7 density medium 2
 modei=0  incident mode is qP
=1 incident mode is qSV
=2 incident mode is SP
 modet=0  scattered mode
 rort=1  reflection(1) or transmission (0)
 azimuth=0 azimuth with respect to x1-axis (clockwise)
 fangle=0 first incidence angle
 langle=45 last incidence angle
 dangle=1 angle increment
 iscale=0      default: angle in degrees
=1 angle-axis in rad
               =2 axis  horizontal slowness
               =3 sin^2 of incidence angle
 ibin=1  binary output
=0 Ascci output
 outparfile =outpar parameter file for plotting
 coeffile =coeff.data coefficient-output file
 test=1  activate testing routines in code
 info=0  output intermediate results

 Notes:
 Axes of symmetry have to coincide in both media.  This code computes
 all 6 REAL reflection/transmissions coefficients on the fly. However,
 the set-up is such Real reflection/transmission coefficients in
 HTI-media with coinciding symmetry axes.
 However, the set-up is such that currently only one coefficient is
```

```
 dumped into the output. This is easily changed.  The solution of the
 scattering problem is obtained numerically and involves the Gaussian
 elimination of a 6X6 matrix. ",



 AUTHOR:: Andreas Rueger, Colorado School of Mines, 02/10/95
               original name of code <graebnerTIH.c>
           modified, extended version of this code <refTIH3D>

  Technical references:

  Sebastian Geoltrain: Asymptotic solutions to direct
and inverse scattering in anisotropic elastic media;
CWP 082.
Graebner, M.; Geophysics, Vol 57, No 11:
Plane-wave reflection and transmission coefficients
for a transversely isotropic solid.
Cerveny, V., 1972, Seismic rays and ray intensities in inhomogeneous
anisotropic media: Geophys. J. R. astr. Soc., 29, 1-13.

.. and some derivations by Andreas Rueger.

 If propagation is perpendicular or
 parallel to the symmetry axis, the solution is analytic (see
 graebner2D.c and rtRealIso.c
```

```
 REFREALVTI -  REAL REFL/transm coeff for VTI media and symmetry-axis
                planes of HTI media

 refRealVTI  [Optional parameters]

 Optional parameters:
 vp1=2          p-wave velocity medium 1 (along symm.axes)
 vs1=1          s-wave velocity medium 1 (along symm.axes)
 eps1=0         Thomsen's epsilon medium 1
 delta1=0  Thomsen's delta medium 1
 rho1=2.7  density medium 1
 axis1=0  medium 1 is VTI
 =1 medium 1 is HTI
 vp2=2.5         p-wave velocity medium 2 (along symm.axes)
 vs2=1.2          s-wave velocity medium 2 (along symm.axes)
 eps2=0  epsilon medium2
 delta2=0 delta medium 2
 rho2=3.0 density medium 2
 axis2=0 medium 2 is VTI
=2 medium 2 is HTI
 modei=0  incident mode is qP
=1 incident mode is qSV
 modet=0  scattered mode
 rort=1  reflection(1) or transmission (0)
 fangle=0 first angle
 langle=45 last angle
 dangle=1 angle increment
 iscale=0       =1 angle-axis in rad
                =2 axis  horizontal slowness
                =3 sin^2 of incidence angle
 ibin=1  binary output
=0 Ascci output
 outparfile =outpar parameter file for plotting
 coeffile =coeff.data coefficient-output file

 Notes:
 Coefficients are based on Graebner's 1992 Geophysics paper. Note the
 mistype in the equation for K1. The algorithm can be used for VTI
 and HTI media on the incidence and scattering side.



 AUTHOR:: Andreas Rueger, Colorado School of Mines, 01/20/95
```

```
      original name of algorithm: graebner1.c

  Technical reference: Graebner, M.; Geophysics, Vol 57, No 11:
Plane-wave reflection and transmission coefficients
for a transversely isotropic solid.
Rueger, A.; Geophysics 1996 (accepted):
P-wave reflection coefficients ...
```

```
REGRID3 - REwrite a [ni3][ni2][ni1] GRID to a [no3][no2][no1] 3-D grid

regrid3 < oldgrid > newgrid [parameters]

Optional parameters:
ni1=1   fastest (3rd) dimension in input grid
ni2=1   second fastest (2nd) dimension in input grid
ni3=1   slowest (1st) dimension in input grid
no1=1   fastest (3rd) dimension in output grid
no2=1   second fastest (2nd) dimension in output grid                ",
no3=1   slowest (1st) dimension in output grid
Optional Parameters:
verbose=0 =1 print some useful information

Notes:
REGRID3 can be used to span a 1-D grid to a 2-D grid,  or a 2-D grid
to a 3-D grid; or to change grid parameters within the dimensions.
Together with MUL and UNIF3, most 3-D velocity model can be


Credits:
  CWP: Zhaobo Meng, 1996, Colorado School of Mines
```

```
RESAMP - RESAMPle the 1st dimension of a 2-dimensional function f(x1,x2)

resamp <infile >outfile [optional parameters]

Required Parameters:

Optional Parameters:
n1=all                  number of samples in 1st (fast) dimension
n2=all                  number of samples in 2nd (slow) dimension
d1=1.0                  sampling interval in 1st dimension
f1=d1                   first sample in 1st dimension
n1r=n1                  number of samples in 1st dimension after resampling
d1r=d1                  sampling interval in 1st dimension after resampling
f1r=f1                  first sample in 1st dimension after resampling

NOTE:  resamp currently performs NO ANTI-ALIAS FILTERING before resampling!
Caveat: this program resamples data that are oscillatory in the fast
   dimension only, such as seismic data with no SU headers. To resample
   other 2d data, such as velocity profiles, use "unisam" or "unisam2
```

```
 ROSSLER - compute the ROSSLER attractor

  rossler > [stdout]

 Required Parameters: none
 Optional Parameters:
 a=.2 parameter for rossler equations
 b=.2 parameter for rossler equations
 c=5.7 parameter for rossler equations
 y0=1.0 initial value of y[0]
 y1=-1.0 initial value of y[1]
 y2=1.0 initial value of y[2]
 h=.01 increment in time
 tol=1.e-08 error tolerance
 stepmax=500 maximum number of steps to compute
 mode=xy xy-pairs, =yz yz-pairs, =xz xz-pairs,
=xyz xyz-triplet, =x only, =y only, =z only
 Notes:
 This program is really just a demo showing how to use the
 differential equation solver rke_solve written by Francois
 Pinard, based on a modified form of the 4th order Runge-Kutta
 method, which employs the error checking method of R. England
 1969.

 The output consists of unformated C-style binary floats, of
 either pairs or triplets as specified by the "mode" paramerter.

 Examples:
 rossler stepmax=1000 mode=xy | xgraph n=1000 &
 rossler stepmax=1000 mode=yz | xgraph n=1000 &
 rossler stepmax=1000 mode=xz | xgraph n=1000 &

 rossler stepmax=1000 mode=x | suaddhead ns=1000 | suxwigb &
 rossler stepmax=1000 mode=y | suaddhead ns=1000 | suxwigb &
 rossler stepmax=1000 mode=z | suaddhead ns=1000 | suxwigb &

 GNUPLOT 3D plot example:
 rossler stepmax=2000 mode=xyz > rossler.bin
 ...when you run gnuplot type the following command ...
 splot "rossler.bin" binary record=2000:2000:2000 with points pointsize .1
```

```
 The rossler equations describe a simple example of a chaotic system
 and are given by the autonomous system of ODE's

x'(t) = - y - z
y'(t) = x + a y
z'(t) = b + z(x - c)

 Author: CWP: Aug 2013: John Stockwell
```

```
 SMOOTH2 --- SMOOTH a uniformly sampled 2d array of data, within a user-
defined window, via a damped least squares technique

 smooth2 < stdin n1= n2= [optional parameters ] > stdout

 Required Parameters:
 n1= number of samples in the 1st (fast) dimension
 n2= number of samples in the 2nd (slow) dimension

 Optional Parameters:
 r1=0 smoothing parameter in the 1 direction
 r2=0 smoothing parameter in the 2 direction
 win=0,n1,0,n2 array for window range
 rw=0 smoothing parameter for window function
 efile=                 =efilename if set write relative error(x1) to
efilename

 Notes:
 Larger r1 and r2 result in a smoother data. Recommended ranges of r1  ",
 and r2 are from 1 to 20.

 The file verror gives the relative error between the original velocity
 and the smoothed one, as a function of depth. If the error is
 between 0.01 and 0.1, the smoothing parameters are suitable. Otherwise,
 consider increasing or decreasing the smoothing parameter values.

 Smoothing can be implemented in a selected window. The range of 1st
 dimension for window is from win[0] to win[1]; the range of 2nd
 dimension is from win[2] to win[3].

 Smoothing the window function (i.e. blurring the edges of the window)
 may be done by setting a nonzero value for rw, otherwise the edges
 of the window will be sharp.



Credits:
CWP: Zhen-yue Liu,
adapted for par/main by John Stockwell 1 Oct 92
Windowing feature added by Zliu on 16 Nov 1992
```

```
 SMOOTH3D - 3D grid velocity SMOOTHing by the damped least squares

smooth3d <infile >outfile [parameters]

 Required Parameters:
 n1= number of samples along 1st dimension
 n2= number of samples along 2nd dimension
 n3= number of samples along 3rd dimension

 Optional Parameters:

 Smoothing parameters (0 = no smoothing)
 r1=0.0 operator length in 1st dimension
 r2=0.0 operator length in 2nd dimension
 r3=0.0 operator length in 3rd dimension

 Sample intervals:
 d1=1.0 1st dimension
 d2=1.0 2nd dimension
 d3=1.0 3rd dimension
 iter=2 number of iteration used
 time=0 which dimension the time axis is (0 = no time axis)
 depth=1 which dimension the depth axis is (ignored when time>0)
 mu=1 the relative weight at maximum depth (or time)
 verbose=0 =1 for printing minimum wavelengths
 slowness=0 =1 smoothing on slowness; =0 smoothing on velocity
 vminc=0 velocity values below it are cliped before smoothing
 vmaxc=99999 velocity values above it are cliped before smoothing

 Notes:
 1. The larger the smoothing parameters, the smoother the output velocity.
    These parameters are lengths of smoothing operators in each dimension.
 2. iter controls the orders of derivatives to be smoothed in the output
    velocity. e.g., iter=2 means derivatives until 2nd order smoothed.
 3. mu is the multipler of smoothing parameters at the bottom compared to
    those at the surface.
 4. Minimum wavelengths of each dimension and the total may be printed
   for the resulting output velocity is. To compute these parameters for
   the input velocity, use r1=r2=r3=0.
 5. Smoothing directly on slowness works better to preserve traveltime.
   So the program optionally converts the input velocity into slowness ",
   and smooths the slowness, then converts back into velocity.
```

```
 Author:  CWP: Zhenyue Liu  March 1995

 Reference:
 Liu, Z., 1994,"A velocity smoothing technique based on damped least squares
in Project Reveiew, May 10, 1994, Consortium Project on
Seismic Inverse Methods for Complex Stuctures.
```

```
 SMOOTHINT2 --- SMOOTH non-uniformly sampled INTerfaces, via the damped
   least-squares technique

  smoothint2 <input ninf= >output [optional parameters]

 Required Parameters:
 <input                  file containing original interfaces
 >output                 file containing smoothed interfaces

 Optional Parameters:
 ninf=5                 number of interfaces
 r=100 smoothing parameter
 npmax=101 maximum number of points in interfaces

 Notes:
 The input file is an ASCII file. Each interface is represented by pairs
 (non-uniform sampling) of x and z values, with one pair of values on
 each line, separated by spaces or tabs. Each interface is separated with
 an entry with a large negative z value for example: 1.0     -9999.
 There is no entry for the surface. The surface is assumed to be flat
 with z=0.
 This is similar to a CSHOT model file without a surface entry and
 without comments.

 The smoothing method is analogous to a moving window averaging process
 (but not the same) with the parameter "r" being analogous to the "width
 of the window. Thus, the size of "r" must be chosen to by compatible
 with the scale (wavelengths) of the variations of the interfaces in the
 model being smoothed.

 Example using the test data set generated by unif2:
 unif2 tfile=tfilename
 Compare the unsmoothed interface model:
 unif2 < tfilename method=interpolation_method |
  psimage n1=100 n2=100 d1=10 d2=10 | ...
 To the smoothed interface model:
 smoothint2 r=100 < tfilename | unif2 method=interpolation_method | ",
psimage n1=100 n2=100 d1=10 d2=10 | ...



 Credits:
  CWP: Zhenyue Liu, Jan 1994
```

Reference:
Liu, Zhenyue, 1994, Velocity smoothing: theory and implementation, Project Review, 1994, Consortium Project on Seismic Inverse Methods for Complex Stuctures (in review)

```
STIFF2VEL - Transforms 2D elastic stiffnesses to (vp,vs,epsilon,delta)

stiff2vel  nx=  nz=  [optional parameters]

Required parameters:
nx=    number of x samples (2nd dimension)
nz=    number of z samples (1st dimension)
rho_file='rho.bin'     input file containing rho(x,z)
c11_file='c11.bin'     input file containing c11(x,z)
c13_file='c13.bin'     input file containing c13(x,z)
c33_file='c33.bin'     input file containing c33(x,z)
c44_file='c44.bin'     input file containing c44(x,z)

Optional Parameters:
vp_file='vp.bin' output file containing P-wave velocities
vs_file='vs.bin' output file containing S-wave velocities
rho_file='rho.bin' output file containing densities
eps_file='eps.bin' output file containing Thomsen epsilon
delta_file='delta.bin' output file containing Thomsen delta

Notes:
1. All quantities in MKS units
2. Isotropy implied by c11(x,z)=c33(x,z)=0
3. Vertical symmetry axis is assumed.



 Coded:
 Aramco: Chris Liner 9/25/2005
         (based on vel2stiff.c)
```

```
SUBSET - select a SUBSET of the samples from a 3-dimensional file

subset <infile >outfile [optional parameters]

Optional Parameters:
n1=nfloats              number of samples in 1st dimension
n2=nfloats/n1           number of samples in 2nd dimension
n3=nfloats/(n1*n2)      number of samples in 3rd dimension
id1s=1                  increment in samples selected in 1st dimension
if1s=0                  index of first sample selected in 1st dimension
n1s=1+(n1-if1s-1)/id1s number of samples selected in 1st dimension
ix1s=if1s,if1s+id1s,...indices of samples selected in 1st dimension
id2s=1                  increment in samples selected in 2nd dimension
if2s=0                  index of first sample selected in 2nd dimension
n2s=1+(n2-if2s-1)/id2s number of samples selected in 2nd dimension
ix2s=if2s,if2s+id2s,...indices of samples selected in 2nd dimension
id3s=1                  increment in samples selected in 3rd dimension
if3s=0                  index of first sample selected in 3rd dimension
n3s=1+(n3-if3s-1)/id3s number of samples selected in 3rd dimension
ix3s=if3s,if3s+id3s,...indices of samples selected in 3rd dimension

For the 1st dimension, output is selected from input as follows:
      output[i1s] = input[ix1s[i1s]], for i1s = 0 to n1s-1
Likewise for the 2nd and 3rd dimensions.



AUTHOR:  Dave Hale, Colorado School of Mines, 07/07/89
```

```
 SWAPBYTES - SWAP the BYTES of various  data types

 swapbytes <stdin [optional parameters]  >stdout

 Required parameters:
  none

 Optional parameters:
 in=float input type (float)
  =double (double)
  =short (short)
  =ushort (unsigned short)
  =long (long)
  =ulong (unsigned long)
  =int (int)

 outpar=/dev/tty output parameter file, contains the
number of values (n1=)
  other choices for outpar are: /dev/tty,
  /dev/stderr, or a name of a disk file

 Notes:
 The byte order of the mantissa of binary data values on PC's and DEC's
 is the reverse of so called "big endian" machines (IBM RS6000, SUN,etc.)
 hence the need for byte-swapping capability. The subroutines in this code
 have been tested for swapping between PCs and big endian machines, but
 have not been tested for DEC products.

 Caveat:
 2 byte short, 4 byte long, 4 byte float, 4 byte int,
 and 8 bit double assumed.


 Credits:
CWP: John Stockwell (Jan 1994)
 Institut fur Geophysik, Hamburg: Jens Hartmann supplied byte swapping
subroutines
```

```
THOM2HTI - Convert Thompson parameters V_p0, V_s0, eps, gamma,
             to the HTI parameters alpha, beta, epsilon(V), delta(V),
      gamma

thom2hti  [optional parameter]


vp=2         symm.axis p-wave velocity
vs=1         symm.axis s-wave velocity
eps=0        Thomsen's (generic) epsilon
gamma=0      Thomsen's generic gamma
weak=1       compute weak approximation
outpar=/dev/tty output parameter file

Outputs:
  alpha, beta, e(V), d(V), gamma

Notes:
Output is dumped to the screen and, if selected to outpar

Code can be used to find models that satisfy the constraints
that are imposed on HTI models caused by vertically fractured
layers. For definition and use of the HTI parameter set see CWP-235.


 Credits: Andreas Rueger, CWP
 For definition and use of the HTI parameter set see CWP-235.
```

```
THOM2STIFF - convert Thomsen's parameters into (density normalized)
               stiffness components for transversely isotropic
    material with in-plane-tilted axis of symmetry

thom2stiff [optional parameter]

vp=2         symm.axis p-wave velocity
vs=1         symm.axis s-wave velocity
eps=0        Thomsen's (generic) epsilon
delta=0      Thomsen's (generic) delta
gamma=0      Thomsen's (generic) gamma
rho=1        density
phi=0        angle(DEG) vertical --> symmetry axes (clockwise)
sign=1       sign of c11+c44 (generally sign=1)
outpar=/dev/tty output parameter file

Output:
aijkl,cijkl (density normalized) stiffness components


Author: CWP: Andreas Rueger  1995
```

```
TRANSP3D - TRANSPose an n1 by n2 by n3 element matrix

transp3d <infile >outfile n1= n2= [optional parameters]

Required Parameters:
n1 number of elements in 1st (fast) dimension of matrix
n2 number of elements in 2nd (middle) dimension of matrix

Optional Parameters:
n3=all number of elements in 3rd (slow) dimension of matrix
perm=231 desired output axis ordering
nbpe=sizeof(float) number of bytes per matrix element
scratchdir=/tmp directory for scratch files
scratchstem=foo stem prefix for scratch file names
verbose=0 =1 for diagnostic information
```

```
TRANSP - TRANSPose an n1 by n2 element matrix

transp <infile >outfile n1= [optional parameters]

Required Parameters:
n1                      number of elements in 1st (fast) dimension of matrix

Optional Parameters:
n2=all                  number of elements in 2nd (slow) dimension of matrix
nbpe=sizeof(float)      number of bytes per matrix element
verbose=0               =1 for diagnostic information
```

```
TVNMOQC - Check tnmo-vnmo pairs; create t-v column files

tvnmoqc [parameters] cdp=... tnmo=... vnmo=...

  Example:
tvnmoqc mode=1 \\
cdp=15,35 \\
tnmo=0.0091,0.2501,0.5001,0.7501,0.9941 \\
vnmo=1497.0,2000.0,2500.0,3000.0,3500.0 \\
tnmo=0.0082,0.2402,0.4902,0.7402,0.9842 \\
vnmo=1495.0,1900.0,2400.0,2900.0,3400.0

Required Parameter:
  prefix=        Prefix of output t-v file(s)
                 Required only for mode=2

Optional Parameter:
  mode=1         1=qc: check that tnmo values increase
                 2=qc and output t-v files

mode=1
  TVNMOQC checks that there is a tnmo and vnmo series for each CDP
    and it checks that each tnmo series increases in time.

mode=2
  TVNMOQC does mode=1 checking, plus ...

  TVNMOQC converts par (MKPARFILE) values written as:

         cdp=15,35,...,95 \\
         tnmo=t151,t152,...,t15n \\
         vnmo=v151,v152,...,v15n \\
         tnmo=t351,t352,...,t35n \\
         vnmo=v351,v352,...,v35n \\
         tnmo=... \\
         vnmo=... \\
         tnmo=t951,t952,...,t95n \\
         vnmo=v951,v952,...,v95n \\

  to column format. The format of each output file is:

         t1 v1
         t2 v2
```

```
          ...
         tn vn

  One file is output for each input pair of tnmo-vnmo series.

  A CDP VALUE MUST BE SUPPLIED FOR EACH TNMO-VNMO ROW PAIR.

  Prefix of each output file is the user-supplied value of
    parameter PREFIX.
  Suffix of each output file is the cdp value.
  For the example above, output files names are:
    PREFIX.15  PREFIX.35  ...  PREFIX.95


Credits:
     MTU: David Forel (adapted from SUNMO)
```

```
 UNIF2ANISO - generate a 2-D UNIFormly sampled profile of elastic
constants from a layered model.

  unif2aniso < infile [Parameters]

 Required Parameters:
 none

 Optional Parameters:
 ninf=5 number of interfaces
 nx=100 number of x samples (2nd dimension)
 nz=100 number of z samples (1st dimension)
 dx=10 x sampling interval
 dz=10 z sampling interval

 npmax=201 maximum number of points on interfaces

 fx=0.0 first x sample
 fz=0.0 first z sample


 x0=0.0,0.0,...,  distance x at which vp00 and vs00 are specified
 z0=0.0,0.0,...,  depth z at which vp00 and vs00 are specified

 vp00=1500,2000,...,P-velocity at each x0,z0 (m/sec)
 vs00=866,1155...,S-velocity at each x0,z0 (m/sec)
 rho00=1000,1100,...,density at each x0,z0 (kg/m^3)
 q00=110,120,130,..,attenuation Q, at each x0,z0 (kg/m^3)

 eps00=0,0,0...,Thomsen or Sayers epsilon
 delta00=0,0,0...,Thomsen or Sayers delta
 gamma00=0,0,0...,Thomsen or Sayers gamma

 dqdx=0.0,0.0,...,x-derivative of Q (d q/dx)
 dqdz=0.0,0.0,...,z-derivative of Q (d q/dz)

 drdx=0.0,0.0,...,x-derivative of density (d rho/dx)
 drdz=0.0,0.0,...,z-derivative of density (d rho/dz)

 dvpdx=0.0,0.0,...,x-derivative of P-velocity (dvp/dx)
 dvpdz=0.0,0.0,...,z-derivative of P-velocity (dvs/dz)

 dvsdx=0.0,0.0,...,x-derivative of S-velocity (dvs/dx)
```

```
 dvsdz=0.0,0.0,...,z-derivative of S-velocity (dvs/dz)

 dedx=0.0,0.0,...,x-derivative of epsilon (de/dx)
 dedz=0.0,0.0,...,z-derivative of epsilon with depth z (de/dz)

 dddx=0.0,0.0,...,x-derivative of delta (dd/dx)
 dddz=0.0,0.0,...,z-derivative of delta (dd/dz)

 dgdz=0.0,0.0,...,x-derivative of gamma (dg/dz)
 dgdx=0.0,0.0,...,z-derivative of gamma (dg/dx)

 phi00=0,0,...,  rotation angle(s) in each layer

 ...output filenames
 c11_file=c11_file output filename for c11 values
 c13_file=c13_file output filename for c13 values
 c15_file=c15_file output filename for c15 values
 c33_file=c33_file output filename for c33 values
 c35_file=c35_file output filename for c35 values
 c44_file=c44_file output filename for c44 values
 c55_file=c55_file output filename for c55 values
 c66_file=c66_file output filename for c66 values

 rho_file=rho_file output filename for density values
 q_file=q_file output filename for Q values

 paramtype=1   =1 Thomsen parameters, =0 Sayers parameters(see below)
 method=linear for linear interpolation of interface
  =mono for monotonic cubic interpolation of interface
=akima for Akima's cubic interpolation of interface
=spline for cubic spline interpolation of interface

 tfile= =testfilename  if set, a sample input dataset is
   output to "testfilename".

 Notes:
 The input file is an ASCII file containing x z values representing a
 piecewise continuous velocity model with a flat surface on top.

 The surface and each successive boundary between media is represented
 by a list of selected x z pairs written column form. The first and
 last x values must be the same for all boundaries. Use the entry
 1.0  -99999  to separate the entries for successive boundaries. No
```

```
boundary may cross another. Note that the choice of the method of
interpolation may cause boundaries to cross that do not appear to
cross in the input data file.

The number of interfaces is specified by the parameter "ninf". This
number does not include the top surface of the model. The input data
format is the same as a CSHOT model file with all comments removed.

The algorithm works by transforming the P-wavespeed , S-wavespeed,
density and the Thomsen or Sayers parameters epsilon, delta, and gamma
into elastic stiffness coefficients. Furthermore, the user can specify
rotations, phi, to the elasticity tensor in each layer.

Common ranges of Thomsen parameters are
 epsilon:  0.0 -> 0.5
 delta:   -0.2 -> 0.4
 gamma: 0.0 -> 0.4

If only P-wave, S-wave velocities and density is given as input,
the model is, by definition,  isotropic.

If files containing Thomsen/Sayers parameters are given, the model
will be assumed to have VTI symmetry.

Example using test input file generating feature:
unif2aniso tfile=testfilename  produces a 5 interface demonstration model
unif2aniso < testfilename
ximage < c11_file n1=100 n2=100
ximage < c13_file n1=100 n2=100
ximage < c15_file n1=100 n2=100
ximage < c33_file n1=100 n2=100
ximage < c35_file n1=100 n2=100
ximage < c44_file n1=100 n2=100
ximage < c55_file n1=100 n2=100
ximage < c66_file n1=100 n2=100
ximage < rho_file n1=100 n2=100
ximage < q_file   n1=100 n2=100



 Credits:
CWP: John Stockwell, April 2005.
  CWP: based on program unif2 by Zhenyue Liu, 1994
```

```
 UNIF2 - generate a 2-D UNIFormly sampled velocity profile from a layered
    model. In each layer, velocity is a linear function of position.

  unif2 < infile > outfile [parameters]

 Required parameters:
 none

 Optional Parameters:
 ninf=5 number of interfaces
 nx=100 number of x samples (2nd dimension)
 nz=100 number of z samples (1st dimension)
 dx=10 x sampling interval
 dz=10 z sampling interval

 npmax=201 maximum number of points on interfaces

 fx=0.0 first x sample
 fz=0.0 first z sample

 x0=0.0,0.0,...,  distance x at which v00 is specified
 z0=0.0,0.0,...,  depth z at which v00 is specified
 v00=1500,2000,2500...,velocity at each x0,z0 (m/sec)
 dvdx=0.0,0.0,...,derivative of velocity with distance x (dv/dx)
 dvdz=0.0,0.0,...,derivative of velocity with depth z (dv/dz)

 method=linear for linear interpolation of interface
  =mono for monotonic cubic interpolation of interface
=akima for Akima's cubic interpolation of interface
=spline for cubic spline interpolation of interface

 tfile= =testfilename  if set, a sample input dataset is
   output to "testfilename".

 Notes:
 The input file is an ASCII file containing x z values representing a
 piecewise continuous velocity model with a flat surface on top. The surface
 and each successive boundary between media are represented by a list of
 selected x z pairs written column form. The first and last x values must
 be the same for all boundaries. Use the entry   1.0  -99999  to separate
 entries for successive boundaries. No boundary may cross another. Note
 that the choice of the method of interpolation may cause boundaries
 to cross that do not appear to cross in the input data file.
```

```
The number of interfaces is specified by the parameter "ninf". This
number does not include the top surface of the model. The input data
format is the same as a CSHOT model file with all comments removed.

Example using test input file generating feature:
unif2 tfile=testfilename    produces a 5 interface demonstration model
unif2 < testfilename | psimage n1=100 n2=100 d1=10 d2=10 | ...



Credits:
 CWP: Zhenyue Liu, 1994
     CWP: John Stockwell, 1994, added demonstration model stuff.
```

```
 UNIF2TI2 - generate a 2-D UNIFormly sampled profile of stiffness
  coefficients of a layered medium (only for P waves).

  unif2ti2 < infile [Parameters]

 Required Parameters:
 none

 Optional Parameters:
 ninf=5 number of interfaces
 nx=100 number of x samples (2nd dimension)
 nz=100 number of z samples (1st dimension)
 dx=10 x sampling interval
 dz=10 z sampling interval
 ns=0          number of samples in vertical direction for smoothing
parameters across boundary

 npmax=201 maximum number of points on interfaces

 fx=0.0 first x sample
 fz=0.0 first z sample


 x0=0.0,0.0,...,  distance x at which vp00 and vs00 are specified
 z0=0.0,0.0,...,  depth z at which vp00 and vs00 are specified

 vp00=1500,2000,...,P-velocity at each x0,z0 (m/sec)
 eps00=0,0,0...,Thomsen parameter epsilon at each x0,z0
 delta00=0,0,0...,Thomsen parameter delta at each x0,z0
 rho00=1000,1100,...,density at each x0,z0 (kg/m^3)

 dvpdx=0.0,0.0,...,x-derivative of P-velocity (dvp/dx)
 dvpdz=0.0,0.0,...,z-derivative of P-velocity (dvs/dz)

 dedx=0.0,0.0,...,x-derivative of epsilon (de/dx)
 dedz=0.0,0.0,...,z-derivative of epsilon with depth z (de/dz)

 dddx=0.0,0.0,...,x-derivative of delta (dd/dx)
 dddz=0.0,0.0,...,z-derivative of delta (dd/dz)

 drdx=0.0,0.0,...,x-derivative of density (d rho/dx)
 drdz=0.0,0.0,...,z-derivative of density (d rho/dz)
```

```
 nufile=  binary file containning tilt value at each grid point


 ...output filenames
 c11_file=c11_file output filename for c11 values
 c13_file=c13_file output filename for c13 values
 c15_file=c15_file output filename for c15 values
 c33_file=c33_file output filename for c33 values
 c35_file=c35_file output filename for c35 values
 c55_file=c55_file output filename for c55 values


 method=linear for linear interpolation of interface
  =mono for monotonic cubic interpolation of interface
=akima for Akima's cubic interpolation of interface
=spline for cubic spline interpolation of interface

 tfile= =testfilename  if set, a sample input dataset is
   output to "testfilename".

 Notes:
 The input file is an ASCII file containing x z values representing a
 piecewise continuous velocity model with a flat surface on top.

 The surface and each successive boundary between media is represented
 by a list of selected x z pairs written column form. The first and
 last x values must be the same for all boundaries. Use the entry
 1.0  -99999  to separate the entries for successive boundaries. No
 boundary may cross another. Note that the choice of the method of
 interpolation may cause boundaries to cross that do not appear to
 cross in the input data file.

 The number of interfaces is specified by the parameter "ninf". This
 number does not include the top surface of the model. The input data
 format is the same as a CSHOT model file with all comments removed.

 The algorithm works by transforming the P-wavespeed, Thomsen parameters
 epsilon and delta, and the tilt of the symmetry axis into density-normalized",
 stiffness coefficients.

 At this stage, the tilt-field file can be prepared using the
 Matlab M-file nu_mod.m based on 2D interpolation between interfaces.  ",
 The binary file contains nu values at each grid point.
```

```
The interfaces are obtained by interpolation on the picked ones
stored in the infile, and the symmetry axis at each point of interface
is assumed to be parallel to the normal direction.

Common ranges of Thomsen parameters are
 epsilon:  0.0 -> 0.5
 delta:   -0.2 -> 0.4


If the tilt-field file is not given, the model will be assumed to
have VTI symmetry.

Example using test input file generating feature:
unif2aniso tfile=testfilename  produces a 5 interface demonstration model
unif2aniso < testfilename
ximage < c11_file n1=100 n2=100
ximage < c13_file n1=100 n2=100
ximage < c15_file n1=100 n2=100
ximage < c33_file n1=100 n2=100
ximage < c35_file n1=100 n2=100
ximage < c55_file n1=100 n2=100
ximage < rho_file n1=100 n2=100



Credits:
     Modified by Pengfei Cai (CWP), Dec 2011
     Modified by Xiaoxiang Wang (CWP), Aug 2010
 Based on program unif2aniso by John Stockwell, 2005
```

```
UNISAM2 - UNIformly SAMple a 2-D function f(x1,x2)

unisam2 [optional parameters] <inputfile >outputfile

Required Parameters:
none
Optional Parameters:
x1=             array of x1 values at which input f(x1,x2) is sampled
... Or specify a unform linear set of values for x1 via:
nx1=1           number of input samples in 1st dimension
dx1=1           input sampling interval in 1st dimension
fx1=0           first input sample in 1st dimension
...
n1=1            number of output samples in 1st dimension
d1=             output sampling interval in 1st dimension
f1=             first output sample in 1st dimension
x2=             array of x2 values at which input f(x1,x2) is sampled
... Or specify a unform linear set of values for x2 via:
nx2=1           number of input samples in 2nd dimension
dx2=1           input sampling interval in 2nd dimension
fx2=0           first input sample in 2nd dimension
...
n2=1            number of output samples in 2nd dimension
d2=             output sampling interval in 2nd dimension
f2=             first output sample in 2nd dimension
...
method1=linear  =linear for linear interpolation
                =mono for monotonic bicubic interpolation
                =akima for Akima bicubic interpolation
                =spline for bicubic spline interpolation
method2=linear  =linear for linear interpolation
                =mono for monotonic bicubic interpolation
                =akima for Akima bicubic interpolation
                =spline for bicubic spline interpolation

NOTES:
The number of input samples is the number of x1 values times the
number of x2 values.  The number of output samples is n1 times n2.
The output sampling intervals (d1 and d2) and first samples (f1 and f2)
default to span the range of input x1 and x2 values.  In other words,
d1=(x1max-x1min)/(n1-1) and f1=x1min; likewise for d2 and f2.

Interpolation is first performed along the 2nd dimension for each
```

```
value of x1 specified.  Interpolation is then performed along the
1st dimension.


AUTHOR:  Dave Hale, Colorado School of Mines, 01/12/91\n"
```

```
UNISAM - UNIformly SAMple a function y(x) specified as x,y pairs

  unisam xin= yin= nout= [optional parameters] >binaryfile
   ... or ...
  unisam xfile= yfile= npairs= nout= [optional parameters] >binaryfile
   ... or ...
  unisam xyfile= npairs= nout= [optional parameters] >binaryfile

Required Parameters:
xin=,,,array of x values (number of xin = number of yin)
yin=,,,array of y values (number of yin = number of xin)
 ... or
xfile= binary file of x values
yfile= binary file of y values
 ... or
xyfile= binary file of x,y pairs
npairs= number of pairs input (active only if xfile= and yfile=
 or xyfile= are set)

nout=  number of y values output to binary file

Optional Parameters:
dxout=1.0  output x sampling interval
fxout=0.0  output first x
method=linear  =linear for linear interpolation (continuous y)
=mono for monotonic cubic interpolation (continuous y')
=akima for Akima's cubic interpolation (continuous y')
=spline for cubic spline interpolation (continuous y'')
isint=,,, where these sine interpolations to apply
amp=,,, amplitude of sine interpolations
phase0=,,, starting phase (defaults: 0,0,0,...,0)
totalphase=,,, total phase (default pi,pi,pi,...,pi.)
nwidth=0       apply window smoothing if nwidth>0
sloth=0  apply interpolation in input (velocities)
=1 apply interpolation to 1/input (slowness),
  =2 apply interpolation to 1/input (sloth), and write
  out velocities in each case.
smooth=0  apply damped least squares smoothing to output
r=10   ... damping coefficient, only active when smooth=1


AUTHOR:  Dave Hale, Colorado School of Mines, 07/07/89
         Zhaobo Meng, Colorado School of Mines,
```

```
     added sine interpolation and window smoothing, 09/16/96
         CWP: John Stockwell,  added file input options, 24 Nov 1997

Remarks: In interpolation, suppose you need 2 pieces of
     sine interpolation before index 3 to 4, and index 20 to 21
   then set: isint=3,20. The sine interpolations use a sine
   function with starting phase being phase0, total phase
   being totalphase (i.e. ending phase being phase0+totalphase
   for each interpolation).
```

```
UTMCONV - CONVert longitude and latitude to UTM, and vice versa

utmconv <stdin >stdout [optional parameters]

Optional parameters:
   idx=23          reference ellipsoid index (default is WGS 1984)
   format=%.3f     output number format (printf style for one float)
   a=(from idx)    user-specified semimajor axis of ellipsoid
   f=(from idx)    user-specified flattening of ellipsoid
   letter=0        =1: use UTM letter designator for latitude/Northing
   invert=0        =0: convert latitude and longitude to UTM
                   =1: convert UTM to latitude and longitude
   verbose=0       =1: echo parameters and number of converted coords

   lon0=           central meridian for TM projection in degrees
                   (default uses the 60 standard UTM longitude zones)
   xoff=500000     false Easting (default: UTM)
   ysoff=10000000  false Northing, southern hemisphere (default: UTM)
   ynoff=0         false Northing, northern hemisphere (default: UTM)

Notes:
   Universal Transverse Mercator (UTM) coordinates are defined between
   latitudes 80S (-80) and 84N (84). Longitude values must be between
   -180 degrees (west) and 179.999... degrees (east).

   Input and output is in ASCII format. For a conversion from lon/lat
   to UTM (invert=0), input is a two-column table of longitude and
   latitude in decimal degrees. Output is a three-column table of
   UTM Easting, UTM Northing, and UTM zone. The zone is given either
   by the zone number only (default, negative on southern hemisphere)
   or by the positive zone number plus a letter designator (letter=1).

Example:
   Convert 40.822N, 14.125E to UTM with zone number and letter,
   output values rounded to nearest integer:
       echo 14.125 40.822 | utmconv letter=1 format=%.0f
   The output is "426213 4519366 33T" (Easting, Northing, UTM zone).

Reference ellipsoids:
   An ellipsoid may be specified by its semimajor axis a and its
   flattening f, or one of the following ellipsoids may be selected
   by its index idx (semimajor axes in meters):
    0  Sphere with radius of 6371000 m
```

```
 1  Airy 1830
 2  Australian National 1965
 3  Bessel 1841 (Ethiopia, Indonesia, Japan, Korea)
 4  Bessel 1841 (Namibia)
 5  Clarke 1866
 6  Clarke 1880
 7  Everest (Brunei, E. Malaysia)
 8  Everest (India 1830)
 9  Everest (India 1956)
10  Everest (Pakistan)
11  Everest (W. Malaysia, Singapore 1948)
12  Everest (W. Malaysia 1969)
13  Geodetic Reference System 1980 (GRS 1980)
14  Helmert 1906
15  Hough 1960
16  Indonesian 1974
17  International 1924 / Hayford 1909
18  Krassovsky 1940
19  Modified Airy
20  Modified Fischer 1960
21  South American 1969
22  World Geodetic System 1972 (WGS 1972)
23  World Geodetic System 1984 (WGS 1984) / NAD 1983
```

UTM grid:
The Universal Transverse Mercator (UTM) system is a world wide coordinate system defined between 80S and 84N. It divides the Earth into 60 six-degree zones. Zone number 1 has its central meridian at 177W (-177 degrees), and numbers increase eastward.

Within each zone, an Easting of 500,000 m is assigned to its central meridian to avoid negative coordinates. On the northern hemisphere, Northings start at 0 m at the equator and increase northward. On the southern hemisphere a false Northing of 10,000,000 m is applied, i.e. Northings start at 10,000,000 m at the equator and decrease southward. Letters are sometimes used to identify different zones of latitude. The letters C-M indicate zones on the southern and the letters N-X zones on the northern hemisphere.

Author:
Nils Maercklin, RISSC, University of Naples, Italy, March 2007

References:
NIMA (2000). Department of Defense World Geodetic System 1984 -
its definition and relationships with local geodetic systems.
Technical Report TR8350.2. National Imagery and Mapping Agency,
Geodesy and Geophysics Department, St. Louis, MO. 3rd edition.
J. P. Snyder (1987). Map Projections - A Working Manual.
U.S. Geological Survey Professional Paper 1395, 383 pages.
U.S. Government Printing Office.

```
 VEL2STIFF - Transforms VELocities, densities, and Thomsen or Sayers
parameters to elastic STIFFnesses

 vel2stiff  [Required parameters] [Optional Parameters] > stdout

 Required parameters:
 vpfile= file with P-wave velocities
 vsfile= file with S-wave velocities
 rhofile= file with densities

 Optional Parameters:
 epsfile= file with Thomsen/Sayers epsilon
 deltafile= file with Thomsen/Sayers delta
 gammafile= file with Thomsen/Sayers gamma
 phi_file= angle of axis of symmetry from vertical (radians)

 c11_file=c11_file     output filename for c11 values
 c13_file=c13_file     output filename for c13 values
 c15_file=c15_file     output filename for c15 values
 c33_file=c33_file     output filename for c33 values
 c35_file=c35_file     output filename for c35 values
 c44_file=c44_file     output filename for c44 values
 c55_file=c55_file     output filename for c55 values
 c66_file=c66_file     output filename for c66 values

 paramtype=1  (1) Thomsen parameters, (0) Sayers parameters(see below)

 nx=101 number of x samples 2nd (slow) dimension
 nz=101 number of z samples 1st (fast) dimension

 Notes:
 Transforms velocities, density and Thomsen/Sayers parameters
 epsilon, delta, and gamma into elastic stiffness coefficients.

 If only P-wave, S-wave velocities and density is given as input,
 the model is assumed to be isotropic.

 If files containing Thomsen/Sayers parameters are given, the model
 will be assumed to have VTI symmetry.

 All input files  vpfile, vsfile, rhofile etc. are assumed to consist
 only of C style binary floating point numbers representing the
 corresponding  material values of vp, vs, rho etc. Similarly, the output
```

files consist of the coresponding stiffnesses as C style binary floats.
If the output files are to be used as input for a modeling program,
such as suea2df, then further, the contents are assumed be arrays of
floating point numbers of the form of Array[n2][n1], where the fast
dimension, dimension 1, represents depth.

Author:
CWP: Sverre Brandsberg-Dahl 1999

Extended:
CWP: Stig-Kyrre Foss 2001
- to include the option to use the parameters by Sayers (1995)
instead of the Thomsen parameters

Technical reference:
Sayers, C. M.: Simplified anisotropy parameters for transversely
isotropic sedimentary rocks. Geophysics 1995, pages 1933-1935.

```
VELCONV - VELocity CONVersion

velconv <infile >outfile intype= outtype= [optional parameters]

Required Parameters:
intype=                 input data type (see valid types below)
outtype=                output data type (see valid types below)

Valid types for input and output data are:
vintt           interval velocity as a function of time
vrmst           RMS velocity as a function of time
vintz           velocity as a function of depth
zt              depth as a function of time
tz              time as a function of depth

Optional Parameters:
nt=all                  number of time samples
dt=1.0                  time sampling interval
ft=0.0                  first time
nz=all                  number of depth samples
dz=1.0                  depth sampling interval
fz=0.0                  first depth
nx=all                  number of traces

Example:  "intype=vintz outtype=vrmst" converts an interval velocity
          function of depth to an RMS velocity function of time.

Notes:  nt, dt, and ft are used only for input and output functions
        of time; you need specify these only for vintt, vrmst, orzt.
        Likewise, nz, dz, and fz are used only for input and output
        functions of depth.

The input and output data formats are C-style binary floats.


 AUTHOR:  Dave Hale, Colorado School of Mines, 07/07/89
```

```
 VELPERTAN - Velocity PERTerbation analysis in ANisotropic media to
             determine the model update required to flatten image gathers",

 velpertan boundary= par=cig.par refl1= refl2= npicks1=
npicks2= cdp1= cdp2= vfile= efile= dfile= nx= dx= fx=
ncdp= dcdp= fcdp= off0= noff= doff= >outfile [parameters]

 Required Parameters:
 refl1= file with picks on the 1st reflector
 refl2= file with picks on the 2nd reflector
 vfile= file defining VP0 at all grid points from prev. iter. ",
 efile= file defining eps at all grid points from prev. iter. ",
 dfile= file defining del at all grid points from prev. iter. ",
 boundary= file defining the boundary above which
         parameters are known; update is done below this ",
boundary ",
 npicks1= number of picks on the 1st reflector
 npicks2= number of picks on the 2nd reflector
 ncdp= number of cdp's
 dcdp= cdp spacing
 fcdp= first cdp
 off0= first offset in common image gathers
 noff= number of offsets in common image gathers
 doff= offset increment in common image gathers
 cip1=x1,r1,r2,..., cip=xn,r1n,r2n         description of input CIGS
 cip2=x2,r1,r2,..., cip=xn,r1n,r2n         description of input CIGS
x x-value of a common image point
r1 hyperbolic component of the residual moveout
r2 non-hyperbolic component of residual moveout
 Optional Parameters:
 method=akima         for linear interpolation of the interface
                       =mono for monotonic cubic interpolation of interface
                       =akima for Akima's cubic interpolation of interface
                       =spline for cubic spline interpolation of interface
 VP0=2000 Starting value for vertical velocity at a point in the
target layer
 x00=0.0 x-coordinate at which VP0 is defined
 z00=0.0 z-coordinate at which VP0 is defined
 eps=0.0 Starting value for Thomsen's parameter epsilon
 del=0.0 Starting value for Thomsen's parameter delta
 kz=0.0 Starting value for the vertical gradient in VP0
 kx=0.0 Starting value for the lateral gradient in VP0
 nx=100 number of nodes in the horizontal direction for the
```

```
velocity grid
 nz=100 number of nodes in the vertical direction for the
velocity grid
 dx=10 horizontal grid increment
 dz=10 vertical grid increment
 fx=0 first horizontal grid point
 fz=0 first vertical grid point
 dt=0.008 traveltime increment
 nt=500 no. of points on the ray
 amax=360 max. angle of emergence
 amin=0 min. angle of emergence

 Smoothing parameters:
 r1=0                  smoothing parameter in the 1 direction
 r2=0                  smoothing parameter in the 2 direction
 win=0,n1,0,n2         array for window range
 rw=0                  smoothing parameter for window function
 nbound=2 number of points picked on the boundary
 tol=0.1 tolerance in computing the offset (m)
 Notes:
 This program is used as part of the velocity analysis technique developed
 by Debashish Sarkar, CWP:2003.

 Notes:
 The output par file contains the coefficients describing the residual
 moveout. This program is used in conjunction with surelanan.


 Author: CSM: Debashish Sarkar, December 2003
 based on program: velpert.c written by Zhenuye Liu.
```

```
 VELPERT - estimate velocity parameter perturbation from covariance
 of imaged depths in common image gathers (CIGs)

 verpert <dfile dzfile=dzfile >outfile [parameters]

 Required Parameters:
 dfile input of imaged depths in CIGs
 dzfile=dzfile input of dz/dv at the imaged depths in CIGs
 outfile output of the estimated parameter
 noff number of offsets
 ncip number of common image gathers

 Optional Parameters:
 moff=noff number of first offsets used in velocity estimation

 Notes:
 1. This program is part of Zhenyue Liu's velocity analysis technique.
    The input dzdv values are computed using the program dzdv.
 2. For given depths, using moff smaller than noff may avoid poor
    values of dz/dv at far offsets. However, a too small moff used
    will the sensitivity of velocity error to the imaged depth.
 3. Outfile contains three parts:
    dlambda correction of the velocity paramter. dlambda plus
     the initial parameter (used in migration) will be
the updated one.
    deviation to measure how close imaged depths are to each other
     in CIGs. Old deviation corresponds to the initial
parameter; new deviation corresponds to the updated one.
    sensitivity  to predict how sensitive the error in the estimated
parameter is to an error in the measurement of imaged
depths.

       error of parameter <= sensitivity * error of depth.



 Author:  Zhenyue Liu, 12/29/93,  Colorado School of Mines

 Reference:
 Liu, Z. 1995, "Migration Velocity Analysis", Ph.D. Thesis, Colorado
      School of Mines, CWP report #168.
```

```
 VERHULST - solve the VERHULST logistic equation

  verhulst > [stdout]

 Required Parameters: none
 Optional Parameters:
 a1=1.0 parameter for verhulst equation
 a2=2000 parameter for verhulst equation
 y0=10 initial value of y[0]
 h=.01 increment in time
 tol=1.e-08 error tolerance
 stepmax=2000 maximum number of steps to compute
 mode=x xy-pairs, =yz yz-pairs, =xz xz-pairs,
=xyz xyz-triplet, =x only, =y only, =z only
 Notes:
 This program is really just a demo showing how to use the
 differential equation solver rke_solve written by Francois
 Pinard, based on a modified form of the 4th order Runge-Kutta
 method, which employs the error checking method of R. England
 1969.

 The output consists of unformated C-style binary floats, of
 either pairs or triplets as specified by the "mode" paramerter.

 Examples:
 x is the population
 verhulst stepmax=2000 mode=x | suaddhead ns=2000 | suxwigb &
 y is dx/dt, the rate of growth of the population
 verhulst stepmax=2000 mode=y | suaddhead ns=2000 | suxwigb &

 In the Verhulst equation, a1 is the reproduction rate and
 a2 is the carrying capacity
  x'(t) = a1 * x * ( 1 - x/a2 )


 The verhulst equation describes a simplified model of a population
 reproducing in an environment with limited resources,
 and are given by the autonomous system of ODE's
y'(t) = a1 * y ( 1 - y/a2 )

 Author: CWP: Aug 2009: John Stockwell
```

```
VTLVZ -- Velocity as function of Time for Linear V(Z);
         writes out a vector of velocity = v0 exp(a t/2)

vtlvz > velfile nt= dt= v0= a=

Required parameters
nt= number of time samples
dt= time sampling interval
v0= velocity at the surface
a= velocity gradient
```

```
 WKBJ - Compute WKBJ ray theoretic parameters, via finite differencing

 wkbj <vfile >tfile nx= nz= xs= zs= [optional parameters]

 Required Parameters:
 <vfile file containing velocities v[nx][nz]
 nx= number of x samples (2nd dimension)
 nz= number of z samples (1st dimension)
 xs= x coordinate of source
 zs= z coordinate of source

 Optional Parameters:
 dx=1.0  x sampling interval
 fx=0.0  first x sample
 dz=1.0  z sampling interval
 fz=0.0  first z sample
 sfile=sfile file containing sigmas sg[nx][nz]
 bfile=bfile file containing incident angles bet[nx][nz]
 afile=afile file containing propagation angles a[nx][nz]

 Notes:
 Traveltimes, propagation angles, sigmas, and incident angles in WKBJ
 by finite differences  in polar coordinates. Traveltimes are calculated
 by upwind scheme; sigmas and incident angles by a Crank-Nicolson scheme.

 Credits:
CWP: Zhenyue Liu, Dave Hale, pre 1992.
```

```
 XY2Z - converts (X,Y)-pairs to spike Z values on a uniform grid

    xy2z < stdin npairs= [optional parameters] >stdout

 Required parameter:
 npairs=  number of pairs input

 Optional parameter:
 scale=1.0 value to scale spikes by
 nx1=100  dimension of first (fast) dimension of output array
 nx2=100  dimension of second (slow) dimension of output array
 x1pad=2 zero padding in x1 dimension
 x2pad=2 zero padding in x2 dimension
 yx=0 assume (x,y) pairs
  =1 assume (y,x) pairs

 Notes:
 Converts ordered (x,y) pairs to spike x1values, of height=scale on a
 uniform grid.


 Credits:
CWP: John Stockwell, Nov 1995
```

```
 Z2XYZ - convert binary floats representing Z-values to ascii
     form in X Y Z ordered triples

    z2xyz <stdin >stdout

 Required parameters:
 n1= number of floats in 1st (fast) dimension

 Optional parameters:

 outpar=/dev/tty  output parameter file

 Notes: This program is useful for converting panels of float
 data (representing evenly spaced z values) to the x y z
 ordered triples required for certain 3D plotting packages.

 Example of NXplot3d usage on a NeXT:
 suplane | sufilter | z2xyz n1=64 > junk.ascii

 Now open junk.ascii as a mesh data file with NXplot3d.
 (NXplot3d is a NeXTStep-only utility for viewing 3d data sets


 Credits:
CWP: John Stockwell based on "b2a" by Jack Cohen
```

```
 SUCENTSAMP - CENTRoid SAMPle seismic traces

  sucentsamp <stdin [optional parameters] >sdout

 Required parameters:
  none

 Optional parameters:
  dt=from header sampling interval
verbose=1 =0 to stop advisory messages

 Notes:
 This program takes seismic traces as input, and returns traces
 consisting of spikes of height equal to the area of each lobe
 of each oscillation, located at the centroid of the lobe in
 question. The height of each spike equal to the area of the
 corresponding lobe.

 Caveat: No check is made that the data are real time traces!


 Credits:

Providence Technologies: Tom Morgan

 Trace header fields accessed: ns, dt
```

```
 SUDIPDIVCOR - Dip-dependent Divergence (spreading) correction

sudipdivcor <stdin >stdout  [optional parms]

 Required Parameters:
dxcdp distance between sucessive cdps in meters

 Optional Parameters:
np=50 number of slopes
tmig=0.0 times corresponding to rms velocities in vmig
vmig=1500.0 rms velocities corresponding to times in tmig
vfile=binary (non-ascii) file containing velocities vmig(t)
conv=0 =1 to apply the conventional divergence correction
trans=0 =1 to include transmission factors
verbose=0 =1 for diagnostic print

 Notes:
 The tmig, vmig arrays specify an rms velocity function of time.
 Linear interpolation and constant extrapolation is used to determine
 rms velocities at times not specified.  Values specified in tmig
 must increase monotonically.

 Alternatively, rms velocities may be stored in a binary file
 containing one velocity for every time sample.  If vfile is
 specified, then the tmig and vmig arrays are ignored.
 The time of the first sample is assumed to be constant, and is
 taken as the value of the first trace header field delrt.

 Whereas the conventional divergence correction (sudivcor) is
 valid only for horizontal reflectors, which have zero reflection
 slope, the dip-dependent divergence correction is valid for any
 reflector dip or reflection slope.  Only the conventional
 correction will be applied to the data if conv=1 is specified.
 Note that the conventional correction over-amplifies
 reflections from dipping beds

 The transmission factor should be applied when the divergence
 corrected data is to be migrated with a reverse time migration
 based on the constant density wave equation.

 Trace header fields accessed:  ns, dt, delrt
```

```
 SUDIVCOR - Divergence (spreading) correction

 sudivcor <stdin >stdout  [optional parms]

 Required Parameters:
 none

 Optional Parameters:
 trms=0.0 times corresponding to rms velocities in vrms
 vrms=1500.0 rms velocities corresponding to times in trms
 vfile= binary (non-ascii) file containing velocities vrms(t)

 Notes:
 The trms, vrms arrays specify an rms velocity function of time.
 Linear interpolation and constant extrapolation is used to determine
 rms velocities at times not specified.  Values specified in trms
 must increase monotonically.

 Alternatively, rms velocities may be stored in a binary file
 containing one velocity for every time sample.  If vfile is specified,
 then the trms and vrms arrays are ignored.

 The time of the first sample is assumed to be constant, and is taken
 as the value of the first trace header field delrt.

 Credits:
CWP: Jack K. Cohen, Francesca Fazarri

 Trace header fields accessed:  ns, dt, delrt
```

```
 SUGAIN - apply various types of gain

 sugain <stdin >stdout [optional parameters]

 Required parameters:
none (no-op)

 Optional parameters:
panel=0         =1  gain whole data set (vs. trace by trace)
tpow=0.0 multiply data by t^tpow
epow=0.0 multiply data by exp(epow*t)
etpow=1.0 multiply data by exp(epow*t^etpow)
gpow=1.0 take signed gpowth power of scaled data
agc=0    flag; 1 = do automatic gain control
gagc=0   flag; 1 = ... with gaussian taper
wagc=0.5 agc window in seconds (use if agc=1 or gagc=1)
trap=none zero any value whose magnitude exceeds trapval
clip=none clip any value whose magnitude exceeds clipval
pclip=none clip any value greater than clipval
nclip=none clip any value less than  clipval
qclip=1.0 clip by quantile on absolute values on trace
qbal=0   flag; 1 = balance traces by qclip and scale
pbal=0   flag; 1 = bal traces by dividing by rms value
mbal=0   flag; 1 = bal traces by subtracting the mean
maxbal=0 flag; 1 = balance traces by subtracting the max
scale=1.0 multiply data by overall scale factor
norm=0.0 divide data by overall scale factor
bias=0.0 bias data by adding an overall bias value
jon=0     flag; 1 means tpow=2, gpow=.5, qclip=.95
verbose=0 verbose = 1 echoes info
mark=0 apply gain only to traces with tr.mark=0
=1 apply gain only to traces with tr.mark!=0
vred=0   reducing velocity of data to use with tpow

  tmpdir= if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
the CWP_TMPDIR environment variable is set use
its value for the path; else use tmpfile()

 Operation order:
 if (norm) scale/norm

 out(t) = scale * BAL{CLIP[AGC{[t^tpow * exp(epow * t^tpow) * ( in(t)-bias )]^gpow}]}
```

```
 Notes:
The jon flag selects the parameter choices discussed in
Claerbout's Imaging the Earth, pp 233-236.

Extremely large/small values may be lost during agc. Windowing
these off and applying a scale in a preliminary pass through
sugain may help.

Sugain only applies gain to traces with tr.mark=0. Use sushw,
suchw, suedit, or suxedit to mark traces you do not want gained.
See the selfdocs of sushw, suchw, suedit, and suxedit for more
information about setting header fields. Use "sukeyword mark
for more information about the mark header field.

      debias data by using mbal=1

      option etpow only becomes active if epow is nonzero

 Credits:
SEP: Jon Claerbout
CWP: Jack K. Cohen, Brian Sumner, Dave Hale

 Note: Have assumed tr.deltr >= 0 in tpow routine.

 Technical Reference:
Jon's second book, pages 233-236.

 Trace header fields accessed: ns, dt, delrt, mark, offset
```

```
SUNAN - remove NaNs & Infs from the input stream

   sunan < in.su >out.su

Optional parameters:
verbose=1 echo locations of NaNs or Infs to stderr
       =0 silent
...user defined ...

value=0.0 NaNs and Inf replacement value
... and/or....
interp=0 =1 replace NaNs and Infs by interpolating
                  neighboring finite values

Notes:
A simple program to remove NaNs and Infs from an input stream.
The program sets NaNs and Infs to "value" if interp=0. When
interp=1 NaNs are replaced with the average of neighboring values
provided that the neighboring values are finite, otherwise
NaNs and Infs are replaced by "value".


Author: Reginald H. Beardsley  2003   rhb@acm.org

 A simple program to remove NaNs & Infs from an input stream. They
 shouldn't be there, but it can be hard to find the cause and fix
 the problem if you can't look at the data.

 Interpolation idea comes from a version of sunan modified by
 Balasz Nemeth while at Potash Corporation in Saskatchewan.
```

```
SUNORMALIZE - Trace NORMALIZation by rms, max, or median       ",
              or median balancing

  sunormalize <stdin >stdout t0=0 t1=TMAX norm=rms

Required parameters:
   dt=tr.dt    if not set in header, dt is mandatory
   ns=tr.ns    if not set in header, ns is mandatory

Optional parameters:
   norm=rms    type of norm rms, max, med , balmed
   t0=0.0      startimg time for window
   t1=TMAX     ending time for window

Notes:
Traces are divided by either the root mean squared amplitude,
trace maximum, or the median value. The option "balmed" is
median balancing which is a shift of the amplitudes by the
median value of the amplitudes.



Author: Ramone Carbonell,
        Inst. Earth Sciences-CSIC Barcelona, Spain, April 1998.
Modifications: Nils Maercklin,
        RISSC, University of Naples, Italy, September 2006
        (fixed user input of ns, dt, if values are not set in header).

Trace header fields accessed: ns, dt
Trace header fields modified: none
```

```
SUPGC   -   Programmed Gain Control--apply agc like function
             but the same function to all traces preserving
             relative amplitudes spatially.
Required parameter:
file=             name of input file

Optional parameters:
ntrscan=200       number of traces to scan for gain function
lwindow=1.0       length of time window in seconds



Author: John Anderson (visitor to CWP from Mobil)

Trace header fields accessed: ns, dt
```

```
SUWEIGHT - weight traces by header parameter, such as offset

  suweight < stdin > stdout [optional parameters]

Required Parameters:
  <none>

Optional parameters:
key=offset keyword of header field to weight traces by
a=1.0 constant weighting parameter (see notes below)
b=.0005 variable weighting parameter (see notes below)

... or use values of a header field for the weighting ...

key2= keyword of header field to draw weights from
scale=.0001 scale factor to apply to header field values

inv=0 weight by header value
 =1 weight by inverse of header value

Notes:
This code is initially written with offset weighting in mind, but may
be used for other, user-specified schemes.

The rationale for this program is to correct for unwanted linear
amplitude trends with offset prior to either CMP stacking or AVO work.
The code has to be edited should other functions of a keyword be required.

The default form of the weighting is to multiply the amplitudes of the
traces by a factor of:    ( a + b*keyword).

If key2=  header field is  set then this program uses the weighting
values read from that header field, instead. Note, that because most
header fields are integers, the scale=.0001 permits 10001 in the header
to represent 1.0001.

To see the list of available keywords, type:    sukeyword  -o  <CR>


Credits:
Author: CWP: John Stockwell  February 1999.
Written for Chris Walker of UniqueStep Ltd., Bedford, U.K.
inv option added by Garry Perratt (Geocon).
```

```
header fields accessed: ns, keyword
```

```
 SUZERO -- zero-out (or set constant) data within a time window

 suzero itmax= < indata > outdata

 Required parameters
  itmax= last time sample to zero out

 Optional parameters
  itmin=0 first time sample to zero out
  value=0 value to set

 See also: sukill, sumute


 Credits:
CWP: Chris
Geocon: Garry Perratt (added value= option)

 Trace header fields accessed: ns
```

```
 SUATTRIBUTES - instantaneous trace ATTRIBUTES

 suattributes <stdin >stdout mode=amp

 Required parameters:
  none

 Optional parameter:
  mode=amp output flag
           =amp envelope traces
           =phase phase traces
           =freq frequency traces
=bandwith Instantaneous bandwidth
=normamp Normalized Phase (Cosine Phase)
           =fdenv 1st envelope traces derivative
           =sdenv 2nd envelope traces derivative
           =q Ins. Q Factor
  ... unwrapping related options ....
unwrap= default unwrap=0 for mode=phase
  default unwrap=1 for freq, uphase, freqw, Q
  dphase_min=PI/unwrap
       trend=0 =1 remove the linear trend of the inst. phase
  zeromean=0 =1 assume instantaneous phase is zero mean

=freqw Frequency Weighted Envelope
=thin  Thin-Bed (inst. freq - average freq)
wint= windowing for freqw
windowing for thin
default=1
  o--------o--------o
  data-1 data data+1

 Notes:
 This program performs complex trace attribute analysis. The first three
 attributes, amp,phase,freq are the classical Taner, Kohler, and
 Sheriff, 1979.

 The unwrapping algorithm is the "simple" unwrapping algorithm that
 searches for jumps in phase.

 The quantity dphase_min is the minimum change in the phase angle taken
 to be the result of phase wrapping, rather than natural phase
 variation in the data. Setting unwrap=0 turns off phase-unwrapping
```

```
 alltogether. Choosing  unwrap > 1 makes the unwrapping function more
 sensitive to instantaneous phase changes.
 Setting unwrap > 1 may be necessary to resolve higher frequencies in
 data (or sample data more finely).

 Examples:
 suvibro f1=10 f2=50 t1=0 t2=0 tv=1 | suattributes2 mode=amp | ...
 suvibro f1=10 f2=50 t1=0 t2=0 tv=1 | suattributes2 mode=phase | ...
 suvibro f1=10 f2=50 t1=0 t2=0 tv=1 | suattributes2 mode=freq | ...
 suplane | suattributes mode=... | supswigb |...


 Credits:
CWP: Jack K. Cohen
      CWP: John Stockwell (added freq and unwrap features)
UGM (Geophysics Students): Agung Wiyono
   email:aakanjas@gmail.com (others) added more attributes


 Algorithm:
c(t) = hilbert_tranform_kernel(t) convolved with data(t)

  amp(t) = sqrt( c.re^2(t) + c.im^2(t))
  phase(t) = arctan( c.im(t)/c.re(t))
  freq(t) = d(phase)/dt

 Reference: Taner, M. T., Koehler, A. F., and  Sheriff R. E.
 "Complex seismic trace analysis", Geophysics,  vol.44, p. 1041-1063, 1979

 Trace header fields accessed: ns, trid
 Trace header fields modified: d1, trid
```

```
SUCMP   - CoMPare two seismic data sets, returns 0 to the shell ",
            if the same and 1 if different

 sucmp file_A file_B

Required parameters:
     none

  Optional parameters:
     limit=1.0e-4    normalized difference threshold value

Notes:
This program is the seismic equivalent of the Unix cmp(1)
command.  However, unlike cmp(1), it understands seismic data
and will consider files which have only small numerical
differences to be the same.

Sucmp first checks that the number of traces and number of samples
are the same. It then compares the trace headers bit for bit.
Finally it checks that the fractional difference of A & B is
less than limit.

This program is intended as an aid in regression testing changes to
seismic processing programs.

Expected usage is in shell scripts or Makefiles, e.g.
  #!/bin/sh
   #-------------------------------------------------------
   # Run a test data set and verify the result is correct
   # If the data doesn't match show the data on the screen.
  #-------------------------------------------------------

 ./fubar par=tst1.par
  sucmp tst1.su ref/tst1.su
  if [ $? ]
     then
     suxwigb <tst1.su &
     suxwigb <ref/tst1.su & "
  fi


Author:  Reginald H. Beardsley
```

```
rhb@acm.org

  sucmp - compare two seismic files in CWP/SU format to see if they
         are the same within the user specified limit.

  Algorithm:

  Loop over both input files comparing data values.  To be
  considered the same files must have:

    - same number of traces
    - same number of samples per trace
    - trace values within limits of each other

 Note that the program exits as soon as the files fail to match.

 For readability, explicit temporary variables are used which the
 compiler will optimize away. Braces are used on all conditionals
 so that a breakpoint can be set to stop the debugger only if the
 condition is true.

 Because of the overloading of trace header fields in CWP/SU, the
 headers are compared bit for bit.
```

```
SUHISTOGRAM - create histogram of input amplitudes

   suhistogram <in.su >out.dat

Required parameters:
min= minimum bin
max= maximum bin
bins= number of bins

Optional parameters
trend=0 =0 1-D histogram
  =1 2-D histogram picks on cumulate
  =2 2-D histogram in trace format

clip=     threshold value to drop outliers

dt= sample rate in feet or milliseconds.  Defaults  to
    tr.dt*1e-3
datum=  header key to get datum shift if desired (e.g. to
hang from water bottom)

Notes:
trend=0 produces a two column ASCII output for use w/ gnuplot.
Extreme values are counted in the end bins.

trend=1 produces a 6 column ASCII output for use w/ gnuplot
The columns are time/depth and picks on the cumulate
at 2.28%, 15.87%, 50%, 84.13% & 97.72% of the total points
corresponding to the median and +- 1 or 2 standard deviations
for a Gaussian distribution.

trend=2 produces an SU trace panel w/ one trace per bin that
can be displayed w/ suximage, etc.

Example for plotting with xgraph:
suhistogram < data.su min=MIN max=MAX bins=BINS |
a2b n1=2 | xgraph n=BINS nplot=1


Author: Reginald H. Beardsley  2006   rhb@acm.org
```

```
 SUMAX - get trace by trace local/global maxima, minima, or absolute maximum

 sumax <stdin >stdout [optional parameters]

 Required parameters:
none

 Optional parameters:
output=ascii  write ascii data to outpar
=binary for binary floats to stdout
=segy for SEGY traces to stdout

mode=maxmin output both minima and maxima
=max maxima only
=min minima only
=abs absolute maxima only
=rms RMS
=thd search first max above threshold

threshamp=0 threshold amplitude value
threshtime=0 tmin to start search for threshold

verbose=0  writes global quantities to outpar
=1 trace number, values, sample location
=2 key1 & key2 instead of trace number
key1=fldr key for verbose=2
key2=ep key for verbose=2

outpar=/dev/tty output parameter file; contains output
from verbose

 Examples:
 For global max and min values:  sumax < segy_data
 For local and global max and min values:  sumax < segy_data verbose=1
 To plot values specified by mode:
    sumax < segy_data output=binary mode=modeval | xgraph n=npairs
 To plot seismic data with the only values nonzero being those specified
 by mode=modeval:
    sumax < segy_data output=segy mode=modeval | suxwigb

 Note: while traces are counted from 1, sample values are counted from 0.
Also, if multiple min, max, or abs max values exist on a trace,
       only the first one is captured.
```

```
 See also: suxmax, supsmax

 Credits:
CWP : John Stockwell (total rewrite)
Geocon : Garry Perratt (all ASCII output changed from %e to %e)
                       (added mode=rms).
      ESCI: Reginald Beardsley (added header key option)
based on an original program by:
SEP: Shuki Ronen
CWP: Jack K. Cohen
      IFM-GEOMAR: Gerald Klein (added threshold option)

 Trace header fields accessed: ns dt & user specified keys
```

```
 SUMEAN - get the mean values of data traces ",

 sumean < stdin > stdout [optional parameters]

 Required parameters:
   power = 2.0 mean to the power
(e.g. = 1.0 mean amplitude, = 2.0 mean energy)

 Optional parameters:
   verbose = 0 writes mean value of section to outpar
= 1 writes mean value of each trace / section to
outpar
   outpar=/dev/tty   output parameter file
   abs = 1             average absolute value
                       = 0 preserve sign if power=1.0

 Notes:
 Each sample is raised to the requested power, and the sum of all those
 values is averaged for each trace (verbose=1) and the section.
 The values power=1.0 and power=2.0 are physical, however other powers
 represent other mathematical L-p norms and may be of use, as well.


 Credits:
  Bjoern E. Rommel, IKU, Petroleumsforskning / October 1997
    bjorn.rommel@iku.sintef.no
```

```
 SUQUANTILE - display some quantiles or ranks of a data set

 suquantile <stdin >stdout [optional parameters]

 Required parameters:
       none (no-op)

 Optional parameters:
panel=1 flag; 0 = do trace by trace (vs. whole data set)
quantiles=1 flag; 0 = give ranks instead of quantiles
  verbose=0 verbose = 1 echoes information

  tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()


 Credits:
      CWP: Jack K. Cohen


 Trace header fields accessed: ns, tracl, mark
```

```
 SUACORFRAC -- general FRACtional Auto-CORrelation/convolution

 suacorfrac power= [optional parameters] <indata >outdata

 Optional parameters:
 a=0 exponent of complex amplitude
 b=0 multiplier of complex phase
 dt=(from header) time sample interval (in seconds)
 verbose=0 =1 for advisory messages
 ntout=tr.ns number of time samples output
 sym=0 if non-zero, produce a symmetric output from
lag -(ntout-1)/2 to lag +(ntout-1)/2
 Notes:
 The calculation is performed in the frequency domain.
 The fractional autocorrelation/convolution is obtained by raising
 Fourier coefficients to seperate real powers
(a,b) for amp and phase:
     Aout exp[-i Pout] = Ain Ain^a exp[-i (1+b) Pin]
where A=amplitude  P=phase.
 Some special cases:
(a,b)=(1,1) -->auto-correlation
(a,b)=(0.5,0.5) -->half-auto-correlation
(a,b)=(0,0) -->no change to data
(a,b)=(0.5,-0.5)-->half-auto-convolution
(a,b)=(1,-1) -->auto-convolution


 Credits:
UHouston: Chris Liner, Sept 2009
CWP: Based on Hale's crpow

 Trace header fields accessed: ns, dt, trid, d1
/
```

```
 SUACOR - auto-correlation

 suacor <stdin >stdout [optional parms]

 Optional Parameters:
 ntout=101 odd number of time samples output
 norm=1 if non-zero, normalize maximum absolute output to 1
 sym=1 if non-zero, produce a symmetric output from
lag -(ntout-1)/2 to lag +(ntout-1)/2

 Credits:
CWP: Dave Hale

 Trace header fields accessed:  ns
 Trace header fields modified:  ns and delrt
```

```
 SUCONV - convolution with user-supplied filter

 suconv <stdin >stdout  filter= [optional parameters]

 Required parameters: ONE of
 sufile= file containing SU trace to use as filter
 filter= user-supplied convolution filter (ascii)

 Optional parameters:
 panel=0 use only the first trace of sufile
  =1 convolve corresponding trace in sufile with
  trace in input data

 Trace header fields accessed: ns
 Trace header fields modified: ns

 Notes: It is quietly assumed that the time sampling interval on the
 single trace and the output traces is the same as that on the traces
 in the input file.  The sufile may actually have more than one trace,
 but only the first trace is used in panel=0. In panel=1 the corresponding
 trace from the sufile are convolved with its counterpart in the data.
 Caveat, in panel=1 there have to be at least as many traces in sufile
 as in the input data. If not, a warning is returned, and later traces
 in the dataset are returned unchanged.

 Examples:
suplane | suwind min=12 max=12 >TRACE
suconv<DATA sufile=TRACE | ...
 Here, the su data file, "DATA", is convolved trace by trace with the
 the single su trace, "TRACE".

suconv<DATA filter=1,2,1 | ...
 Here, the su data file, "DATA", is convolved trace by trace with the
 the filter shown.


 Credits:
CWP: Jack K. Cohen, Michel Dietrich

  CAVEATS: no space-variable or time-variable capacity.
     The more than one trace allowed in sufile is the
     beginning of a hook to handle the spatially variant case.
```

```
Trace header fields accessed: ns
Trace header fields modified: ns
```

```
 SUREFCON -  Convolution of user-supplied Forward and Reverse
refraction shots using XY trace offset in reverse shot

surefcon <forshot sufile=revshot  xy=trace offseted  >stdout

 Required parameters:
 sufile= file containing SU trace to use as reverse shot
 xy= Number of traces offseted from the 1st trace in sufile

 Optional parameters:
 none

 Trace header fields accessed: ns
 Trace header fields modified: ns

 Notes:
 This code implements the Refraction Convolution Section (RCS) method
 of generalized reciprocal refraction traveltime analysis developed by
 Derecke Palmer and Leoni Jones.

 The time sampling interval on the output traces is half of that on the
 traces in the input files.

 Example:

 surefcon <DATA sufile=DATA xy=1 | ...

 Here, the su data file, "DATA", convolved the nth trace by
 (n+xy)th trace in the same file



 Credits: (based on suconv)
CWP: Jack K. Cohen, Michel Dietrich
UNSW: D. Palmer, K.T. LEE
  CAVEATS: no space-variable or time-variable capacity.
The more than one trace allowed in sufile is the
beginning of a hook to handle the spatially variant case.

 Trace header fields accessed: ns
 Trace header fields modified: ns
 Notes:
 This code implements the refraction convolution
```

section (RCS) method
method described in:

Palmer, D, 2001a, Imaging refractors with the convolution section,
Geophysics 66, 1582-1589.
Palmer, D, 2001b, Resolving refractor ambiguities with amplitudes,
Geophysics 66, 1590-1593.

Exploration Geophysics (2005) 36, 1825
Butsuri-Tansa (Vol. 58, No.1)
Mulli-Tamsa (Vol. 8,
A simple approach to refraction statics with the
Generalized Main Reciprocal Method and the Refraction
Convolution Section Heading
by Derecke Palmer Leonie Jones

```
 SUXCOR - correlation with user-supplied filter

 suxcor <stdin >stdout  filter= [optional parameters]

 Required parameters: ONE of
 sufile= file containing SU traces to use as filter
 filter= user-supplied correlation filter (ascii)

 Optional parameters:
 vibroseis=0 =nsout for correlating vibroseis data
 first=1 supplied trace is default first element of
  correlation.  =0 for it to be second.
 panel=0 use only the first trace of sufile as filter
  =1 xcor trace by trace an entire gather
 ftwin=0 first sample on the first trace of the window
  (only with panel=1)
 ltwin=0 first sample on the last trace of the window
  (only with panel=1)
 ntwin=nt number of samples in the correlation window
  (only with panel=1)
 ntrc=48 number of traces on a gather
  (only with panel=1)

 Trace header fields accessed: ns
 Trace header fields modified: ns

 Notes: It is quietly assumed that the time sampling interval on the
 single trace and the output traces is the same as that on the traces
 in the input file.  The sufile may actually have more than one trace,
 but only the first trace is used when panel=0. When panel=1 the number
 of traces in the sufile MUST be the same as the number of traces in
 the input.

 Examples:
suplane | suwind min=12 max=12 >TRACE
suxcor<DATA sufile=TRACE |...
 Here, the su data file, "DATA", is correlated trace by trace with the
 the single su trace, "TRACE".

suxcor<DATA filter=1,2,1 | ...
 Here, the su data file, "DATA", is correlated trace by trace with the
 the filter shown.
```

```
 Correlating vibroseis data with a vibroseis sweep:
 suxcor < data sufile=sweep vibroseis=nsout  |...

 is equivalent to, but more efficient than:

 suxcor < data sufile=sweep |
 suwind itmin=nsweep itmax=nsweep+nsout | sushw key=delrt a=0.0 |...

 sweep=vibroseis sweep in SU format, nsweep=number of samples on
 the vibroseis sweep, nsout=desired number of samples on output

 or
 suxcor < data sufile=sweep |
 suwind itmin=nsweep itmax=nsweep+nsout | sushw key=delrt a=0.0 |...

 tsweep=sweep length in seconds, tout=desired output trace length in seconds

 In the spatially variant case (panel=1), a window with linear slope
 can be defined:
  ftwin is the first sample of the first trace in the gather,
  ltwin is the first sample of the last trace in the gather,
  ntwin is the lengthe of the window,
  ntrc is the the number of traces in a gather.

  If the data consists of a number gathers which need to be
correlated with the same number gathers in the sufile, ntrc
assures that the correlating window re-starts for each gather.

The default window is non-sloping and takes the entire trace
into account (ftwin=ltwin=0, ntwin=nt).


 Credits:
CWP: Jack K. Cohen, Michel Dietrich
      CWP: modified by Ttjan to include cross correlation of panels
   permitting spatially and temporally varying cross correlation.
      UTK: modified by Rick Williams for vibroseis correlation option.

  CAVEATS:
     In the option, panel=1 the number of traces in the sufile must be
     the same as the number of traces on the input.

 Trace header fields accessed: ns
```

```
Trace header fields modified: ns
```

```
DCTCOMP - Compression by Discrete Cosine Transform

  dctcomp < stdin n1= n2=   [optional parameter] > sdtout

Required Parameters:
n1= number of samples in the fast (first) dimension
n2= number of samples in the slow (second) dimension
Optional Parameters:
blocksize1=16 blocksize in direction 1
blocksize2=16 blocksize in direction 2
error=0.01 acceptable error



Author:  CWP: Tong Chen   Dec 1995
         fixed by Graham Ganssle, Sandstone Oil & Gas, Sept 2015
```

```
 SUPACK1 - pack segy trace data into chars

 supack1 <segy_file >packed_file gpow=0.5

 Required parameters:
none

 Optional parameter:
gpow=0.5 exponent used to compress the dynamic
range of the traces


 Credits:
CWP: Jack K. Cohen, Shuki Ronen, Brian Sumner

 Caveats:
This program is for single site use.  Use segywrite to make
a portable tape.

We are storing the local header words, ungpow and unscale,
required by suunpack1 as floats.  Although not essential
(compare the handling of such fields as dt), it allows us
to demonstrate the convenience of using the natural data type.
In any case, the data itself is non-portable floats in general,
so we aren't giving up any intrinsic portability.

 Notes:
ungpow and unscale are defined in segy.h
trid = CHARPACK is defined in su.h and segy.h

 Trace header fields accessed: ns
 Trace header fields modified: ungpow, unscale, trid
```

```
 SUPACK2 - pack segy trace data into 2 byte shorts

 supack2 <segy_file >packed_file gpow=0.5

 Required parameters:
none

 Optional parameter:
gpow=0.5 exponent used to compress the dynamic
range of the traces


 Credits:
CWP: Jack K. Cohen, Shuki Ronen, Brian Sumner

 Revised: 7/4/95  Stewart A. Levin  Mobil
          Changed encoding to ensure 2 byte length (short is
    8 bytes on Cray).

 Caveats:
This program is for single site use.  Use segywrite to make
a portable tape.

We are storing the local header words, ungpow and unscale,
required by suunpack2 as floats.

 Notes:
ungpow and unscale are defined in segy.h
trid = SHORTPACK is defined in su.h and segy.h

 Trace header fields accessed: ns
 Trace header fields modified: ungpow, unscale, trid
```

```
SUUNPACK1 - unpack segy trace data from chars to floats

    suunpack1 <packed_file >unpacked_file

suunpack1 is the approximate inverse of supack1


 Credits:
CWP: Jack K. Cohen, Shuki Ronen, Brian Sumner

 Caveats:
This program is for single site use with supack1.  See the
supack1 header comments.

 Notes:
ungpow and unscale are defined in segy.h
trid = CHARPACK is defined in su.h and segy.h


 Trace header fields accessed: ns, trid, ungpow, unscale
 Trace header fields modified:     trid, ungpow, unscale
```

```
 SUUNPACK2 - unpack segy trace data from shorts to floats

    suunpack2 <packed_file >unpacked_file

 suunpack2 is the approximate inverse of supack2


 Credits:
CWP: Jack K. Cohen, Shuki Ronen, Brian Sumner

 Revised:  7/4/95 Stewart A. Levin  Mobil
          Changed decoding to parallel 2 byte encoding of supack2

 Caveats:
This program is for single site use with supack2.  See the
supack2 header comments.

 Notes:
ungpow and unscale are defined in segy.h
trid = SHORTPACK is defined in su.h and segy.h

 Trace header fields accessed: ns, trid, ungpow, unscale
 Trace header fields modified:     trid, ungpow, unscale
```

```
 DT1TOSU - Convert ground-penetrating radar data in the
Sensors & Software X.DT1 GPR format to SU format.

 dt1tosu < gpr_data_in_dt1_format  > stdout

 Optional parameters:
 ns=from header number of samples per trace
 dt=.8 time sample interval (see below)
 swap=endian endian is auto-determined =1 (big endian) swap
=0 don't swap bytes (little endian machines)
 verbose=0 silent
=1 S & S header values from first trace
sent to outpar
=2 S & S header values from all traces
sent to outpar
 outpar=/dev/tty output parameter file
 list=0 silent
=1 list explaining labels used in verbose
     is printed to stderr

 Caution: An incorrect ns field will munge subsequent processing.

 Notes:
 For compatiblity with SEGY header, apparent dt is set to
 .8 ms (800 microsecs).  Actual dt is .8 nanosecs.
 Using TRUE DISTANCES, this scales velocity
 and frequency by a factor of 1 million.
Example: v_air = 9.83X10^8 ft/s  (real)
 v_air = 983 ft/s (apparent for su)
Example: fnyquist = 625 MHz (real)
fnyquist = 625 Hz (apparent for su)

 IBM RS6000, NeXT, SUN are examples of big endian machines
 PC's and DEC are examples of little endian machines

 Caveat:
 This program has not been tested on DEC, some modification of the
 byte swapping routines may be required.


 Credits:
CWP: John Stockwell, Jan 1994   Based on a code "sugpr" by
UTULSA: Chris Liner & Bill Underwood  (Dec93)
```

```
modifications permit S & S dt1 header information to be transferred
directly to SU header

March 2012: CWP John Stockwell  updated for the revised
S&S DT1, which they still call "DT1" though it is different.

Trace header fields set: ns, tracl, tracr, dt, delrt, trid,
   hour, minute, second

Reference: Sensors & Software pulseEKKO and Noggin^plus Data File
    Formats
Publication of:
Sensors & Software: suburface imaging solutions
1091 Brevik Place
Mississauga, ON L4W 3R7 Canada
Sensors & Software In
Tel: (905) 624-8909
Fax (905) 624-9365
E-mail: sales@sensoft.ca
Website: www.sensoft.ca
```

```
 SEGYCLEAN - zero out unassigned portion of header

 segyclean <stdin >stdout

 Since "foreign" SEG-Y tapes may use the unassigned portion
 of the trace headers and since SU now uses it too, this
 program zeros out the fields meaningful to SU.

  Example:
   segyread trmax=200 | segyclean | suximage



 Credits:
CWP: Jack Cohen
```

```
 SEGYHDRMOD - replace the text header on a SEGY file

   segyhdrmod text=file data=file

   Required parameters:

   text=      name of file containing new 3200 byte text header
   data=      name of file containing SEGY data set

 Notes:
 This program simply does a replacement of the content of the first
 3200 bytes of the SEGY file with the contents of the file specified
 by the text= parameter. If the text header in the SEGY standard
 ebcdic format, the user will need to supply an ebcdic format file
 as the text=  as input file. A text file may be converted from
 ascii to ebcdic via:
   dd if=ascii_filename of=ebcdic_filename conv=ebcdic ibs=3200 count=1
 or from ebcdic to ascii via:
   dd if=ebcdic_filename of=ascii_filename ibs=3200 conv=ascii count=1



=====================================================================*\

   sgyhdrmod - replace the text header on a SEGY data file in place

   This program only reads and writes 3200 bytes

   Reginald H. Beardsley                          rhb@acm.org

\*=====================================================================*/
```

```
 SEGYHDRS - make SEG-Y ascii and binary headers for segywrite

 segyhdrs [ < sudata ] [optional parameters] [ > copy of sudata ]

 Required parameters:
ns=  if no input trace header
dt=  if no input trace header
 Optional parameters:
  ns=tr.ns from header    number of samples on input traces
  dt=tr.dt from header sample rate (microseconds) from traces
  bfile=binary name of file containing binary block
  hfile=header name of file containing ascii block
   Some binary header fields are set:
  jobid=1 job id field
  lino=1 line number (only one line per reel)
  reno=1 reel number
  format=1 data format

 All other fields are set to 0, by default.
 To set any binary header field, use sukeyword to find out
 the appropriate keyword, then use the getpar form:
  keyword=value to set keyword to value

 The header file is created as ascii and is translated to ebcdic
 by segywrite before being written to tape.  Its contents are
 formal but can be edited after creation as long as the forty
 line format is maintained.

 Caveat: This program has not been tested under XDR for machines
 not having a 2 byte unsigned short integral data type.


 Credits:

CWP: Jack K. Cohen,  John Stockwell
      MOBIL: Stew Levin
```

```
 SEGYREAD - read an SEG-Y tape

   segyread > stdout tape=

   or

   SEG-Y data stream ... | segyread tape=-  > stdout

 Required parameter:
 tape= input tape device or seg-y filename (see notes)

 Optional parameters:
 buff=1 for buffered device (9-track reel tape drive)
=0 possibly useful for 8mm EXABYTE drives
 verbose=0 silent operation
=1 ; echo every 'vblock' traces
 vblock=50 echo every 'vblock' traces under verbose option
 hfile=header file to store ebcdic block (as ascii)
 bfile=binary file to store binary block
 xfile=xhdrs file to store extended text block
 over=0 quit if bhed format not equal 1, 2, 3, 5, or 8
= 1 ; override and attempt conversion
 format=bh.format if over=1 try to convert assuming format value
 conv=1 convert data to native format
= 0 ; assume data is in native format
 ebcdic=1 perform ebcdic to ascii conversion on 3200 byte textural
               header. =0 do not perform conversion
 ns=bh.hns number of samples (use if bhed ns wrong)
 trcwt=1 apply trace weighting factor (bytes 169-170)
=0, do not apply.  (Default is 0 for formats 1 and 5)
 trmin=1 first trace to read
 trmax=INT_MAX last trace to read
 endian=(autodetected) =1 for big-endian,  =0 for little-endian byte order
 swapbhed=endian swap binary reel header?
 swaphdrs=endian swap trace headers?
 swapdata=endian swap data?
 errmax=0 allowable number of consecutive tape IO errors
 remap=...,... remap key(s)
 byte=...,... formats to use for header remapping

 Notes:
 Traditionally tape=/dev/rmt0.  However, in the modern world tape device
 names are much less uniform.  The magic name can often be deduced by
```

```
 "ls /dev".  Likely man pages with the names of the tape devices are:
 "mt", "sd" "st".  Also try "man -k scsi", " man mt", etc.
 Sometimes "mt status" will tell the device name.

 For a SEG-Y diskfile use tape=filename.

 The xfile argument will only be used if the file contains extended
 text headers.

 Remark: a SEG-Y file is not the same as an su file. A SEG-Y file
 consists of three parts: an ebcdic header, a binary reel header, and
 the traces.  The traces are (usually) in 32 bit IBM floating point
 format.  An SU file consists only of the trace portion written in the
 native binary floats.

 Formats supported:
 1: IBM floating point, 4 byte (32 bits)
 2: two's complement integer, 4 byte (32 bits)
 3: two's complement integer, 2 byte (16 bits)
 5: IEEE floating point, 4 byte (32 bits)
 8: two's complement integer, 1 byte (8 bits)

 tape=-   read from standard input. Caveat, under Solaris, you will
 need to use the buff=1 option, as well.

 Header remap:
 The value of header word remap is mapped from the values of byte

 Map a float at location 221 to sample spacing d1:
segyread <data >outdata remap=d1 byte=221f

 Map a long at location 225 to source location sx:
segyread <data >outdata remap=sx byte=225l

 Map a short at location 229 to gain constant igc:
segyread <data >outdata remap=igc byte=229s

 Or all combined:
segyread <data >outdata remap=d1,sx,igc byte=221f,225l,229s

 Segy header words are accessed as Xt where X denotes the byte number
 starting at 1 in correspondance with the SEGY standard (1975)
 Known types include: f float (4 bytes)
```

```
  l long int (4 bytes)
  s short int (2 bytes)
  b byte (1 bytes)

  type:   sudoc segyread   for further information


 Note:
      If you have a tape with multiple sequences of ebcdic header,
binary header,traces, use the device that
invokes the no-rewind option and issue multiple segyread
commands (making an appropriate shell script if you
want to save all the headers). Consider using >> if
you want a single trace file in the end.  Similar
considerations apply for multiple reels of tapes,
but use the standard rewind on end of file.

 Note: For buff=1 (default) tape is accessed with 'read', for buff=0
tape is accessed with fread. We suggest that you try buff=1
even with EXABYTE tapes.
 Caveat: may be slow on an 8mm streaming (EXABYTE) tapedrive
 Warning: segyread or segywrite to 8mm tape is fragile. Allow sufficient
time between successive reads and writes.
 Warning: may return the error message "efclose: fclose failed"
intermittently when segyreading/segywriting to 8mm (EXABYTE) tape
even if actual segyread/segywrite is successful. However, this
error message may be returned if your tape drive has a fixed
block size set.
 Caution: When reading or writing SEG-Y tapes, the tape
drive should be set to be able to read variable block length
tape files.


 Credits:
SEP: Einar Kjartansson
CWP: Jack K. Cohen, Brian Sumner, Chris Liner
   : John Stockwell (added 8mm tape stuff)
 conv parameter added by:
Tony Kocurko
Department of Earth Sciences
Memorial University of Newfoundland
St. John's, Newfoundland
```

```
 read from stdin via tape=-  added by Tony Kocurko
 bhed format = 2,3 conversion by:
Remco Romijn (Applied Geophysics, TU Delft)
J.W. de Bruijn (Applied Geophysics, TU Delft)
 bhed format = 8 conversion by: John Stockwell
 header remap feature added by:
  Matthias Imhof, Virginia Tech
--------------------------
 Additional Notes:
Brian's subroutine, ibm_to_float, which converts IBM floating
point to IEEE floating point is NOT portable and must be
altered for non-IEEE machines. See the subroutine notes below.

A direct read by dd would suck up the entire tape; hence the
dancing around with buffers and files.
```

```
SEGYSCAN -- SCANs SEGY file trace headers for min-max in  several
    possible formats.

  segyscan < segyfile

Notes:
The SEGY file trace headers are scanned assuming short, ushort, int,
uint, float, and double and the results are printed as tables.



Credits: Stew Levin, June 2013
```

```
 SEGYWRITE - write an SEG-Y tape

 segywrite <stdin tape=

 Required parameters:
tape= tape device to use (see sudoc segyread)

 Optional parameter:
 verbose=0 silent operation
=1 ; echo every 'vblock' traces
 vblock=50 echo every 'vblock' traces under verbose option
 buff=1 for buffered device (9-track reel tape drive)
=0 possibly useful for 8mm EXABYTE drive
 conv=1 =0 don't convert to IBM format
 ebcdic=1 convert text header to ebcdic, =0 leave as ascii
 hfile=header ebcdic card image header file
 bfile=binary binary header file
 trmin=1 first trace to write
 trmax=INT_MAX  last trace to write
 endian=(autodetected) =1 for big-endian and =0 for little-endian byte order
 errmax=0 allowable number of consecutive tape IO errors
 format= override value of format in binary header file

 Note: The header files may be created with  'segyhdrs'.


 Note: For buff=1 (default) tape is accessed with 'write', for buff=0
tape is accessed with fwrite. Try the default setting of buff=1
for all tape types.
 Caveat: may be slow on an 8mm streaming (EXABYTE) tapedrive
 Warning: segyread or segywrite to 8mm tape is fragile. Allow time
   between successive reads and writes.
 Precaution: make sure tapedrive is set to read/write variable blocksize
   tapefiles.

 For more information, type: sudoc <segywrite>



 Warning: may return the error message "efclose: fclose failed"
 intermittently when segyreading/segywriting to 8mm EXABYTE tape,
 even if actual segyread/segywrite is successful. However, this
 may indicate that your tape drive has been set to a fixed block
```

```
 size. Tape drives should be set to variable block size before reading
 or writing tapes in the SEG-Y format.

 Credits:
SEP: Einar Kjartansson
CWP: Jack, Brian, Chris
   : John Stockwell (added EXABYTE functionality)
 Notes:
Brian's subroutine, float_to_ibm, for converting IEEE floating
point to IBM floating point is NOT portable and must be
altered for non-IEEE machines. See the subroutine notes below.

On machines where shorts are not 2 bytes and/or ints are not
4 bytes, routines to convert SEGY 16 bit and 32 bit integers
will be required.

The program, segyhdrs, can be used to make the ascii and binary
files required by this code.
```

```
 SETBHED - SET the fields in a SEGY Binary tape HEaDer file, as would be
      produced by segyread and segyhdrs

 setbhed par= [optional parameters]

 Required parameter:
  none
 Optional parameters:
bfile=binary output binary tape header file
par= =parfile
 Set field by field, if desired:
  jobid= job id field
  lino= line number (only one line per reel)
  reno= reel number
  format= data format
 ... etc....
 To set any binary header field, use sukeyword to find out
 the appropriate keyword, then use the getpar form:
  keyword=value to set keyword to value
 Notes:
 As with all other programs in the CWP/SU package that use getpars,
 (GET PARameters from the command line) a file filled with such
 statments may be included via option par=parfile. In particular, a
 parfile created by   "bhedtopar"  may be used as input for the program
 "setbhed".

 The binary header file that results from running segyread may have the
 wrong byte order. You will need to use "swapbhed" to change the byte,"
 order before applying this program.

 Example:
   segyread tape=yourdata.segy bfile=yourdata.b > yourdata.su
 If
   bhedtopar < yourdata.b | more
 shows impossible values, then apply
   swapbhed < yourdata.b > swapped.b
 then apply
   bhedtopar < swapped.b | more
   bhedtopar < swapped.b outpar=parfile
 hand edit parfile, and then apply
  setbhed par=parfile bfile=swapped.b > new.b
 then apply
   segywrite tape=fixeddata.segy bfile=new.b < yourdata.su
```

```
 Caveat: This program breaks if a "short" isn't 2 bytes since
         the SEG-Y standard demands a 2 byte integer for ns.

 Credits:

CWP: John Stockwell  11 Nov 1994
```

```
SUASCII - print non zero header values and data in various formats

suascii <stdin >ascii_file

Optional parameter:
   bare=0     print headers and data
       =1     print only data
       =2     print headers only
       =3     print data in print data in .csv format, e.g. for Excel
       =4     print data as tab delimited .txt file, e.g. for GnuPlot
       =5     print data as .xyz file, e.g. for plotting with GMT

   ntr=50     maximum number of output traces (bare=3 or bare=4 only)
   index=0    don't include time/depth index in ascii file (bare=4)
        =1    include time/depth index in ascii file

   key=       if set, name of keyword containing x-value
              in .xyz output (bare=5 only)
   sep=       if set, string separating traces in .xyz output
              (bare=5; default is no separation)

   verbose=0  =1 for detailed information

Notes:
   The programs suwind and suresamp provide trace selection and
   subsampling, respectively.
   With bare=0 and bare=1 traces are separated by a blank line.

   With bare=3 a maximum of ntr traces are output in .csv format
   ("comma-separated value"), e.g. for import into spreadsheet
   applications like Excel.

   With bare=4 a maximum of ntr traces are output in as tab delimited
   columns. Use bare=4 for plotting in GnuPlot.

   With bare=5 traces are written as "x y z" triples as required
   by certain plotting programs such as the Generic Mapping Tools
   (GMT). If sep= is set, traces are separated by a line containing
   the string provided, e.g. sep=">" for GMT multisegment files.

   "option=" is an acceptable alias for "bare=".

Related programs: sugethw, sudumptrace
```

```
Credits:
   CWP: Jack K. Cohen  c. 1989
   CENPET: Werner M. Heigl 2006 - bug fixes & extensions
   RISSC:  Nils Maercklin 2006

Trace header field accessed: ns, dt, delrt, d1, f1, trid
```

```
 SUINTVEL - convert stacking velocity model to interval velocity model

 suintvel vs= t0= outpar=/dev/tty

 Required parameters:
vs= stacking velocities
t0= normal incidence times

 Optional parameters:
mode=0 output h= v= ; =1 output v=  t=
outpar=/dev/tty output parameter file in the form:
h=layer thicknesses vector
v=interval velocities vector
....or ...
t=vector of times from t0
v=interval velocities vector

 Examples:
    suintvel vs=5000,5523,6339,7264 t0=.4,.8,1.125,1.425 outpar=intpar

    suintvel par=stkpar outpar=intpar

 If the file, stkpar, contains:
    vs=5000,5523,6339,7264
    t0=.4,.8,1.125,1.425
 then the two examples are equivalent.

 Note: suintvel does not have standard su syntax since it does not
      operate on seismic data.  Hence stdin and stdout are not used.

 Note: may go away in favor of par program, velconv, by Dave


 Credits:
CWP: Jack

 Technical Reference:
The Common Depth Point Stack
William A. Schneider
Proc. IEEE, v. 72, n. 10, p. 1238-1254
1984

 Formulas:
```

```
     Note: All sums on i are from 1 to k

From Schneider:
Let h[i] be the ith layer thickness measured at the cmp and
v[i] the ith interval velocity.
Set:
t[i] = h[i]/v[i]
Define:
t0by2[k] = 0.5 * t0[k] = Sum h[i]/v[i]
vh[k] = vs[k]*vs[k]*t0by2[k] = Sum v[i]*h[i]
Then:
dt[i] = h[i]/v[i] = t0by2[i] - t0by2[i-1]
dvh[i] = h[i]*v[i] = vh[i] - vh[i-1]
h[i] = sqrt(dvh[i] * dt[i])
v[i] = sqrt(dvh[i] / dt[i])
```

```
 SUOLDTONEW - convert existing su data to xdr format

 suoldtonew <oldsu >newsu

 Required parameters:
none

 Optional parameters:
none

 Notes:
 This program is used to convert native machine datasets to
 xdr-based, system-independent format.



 Author: Stewart A. Levin, Mobil, 1966
```

```
 SUSTKVEL - convert constant dip layer interval velocity model to the
   stacking velocity model required by sunmo

 sustkvel v= h= dip=0.0 outpar=/dev/tty

 Required parameters:
v= interval velocities
h= layer thicknesses at the cmp

 Optional parameters:
dip=0.0 (constant) dip of the layers (degrees)
outpar=/dev/tty output parameter file in the form
required by sunmo:
tv=zero incidence time pick vector
v=stacking velocities vector

 Examples:
    sustkvel v=5000,6000,8000,10000 h=1000,1200,1300,1500 outpar=stkpar
    sunmo <data.cdp par=stkpar >data.nmo

    sustkvel par=intpar outpar=stkpar
    sunmo <data.cdp par=stkpar >data.nmo

 If the file, intpar, contains:
    v=5000,6000,8000,10000
    h=1000,1200,1300,1500
 then the two examples are equivalent.  The created parameter file,
 stkpar, is in the form of the velocity model required by sunmo.

 Note: sustkvel does not have standard su syntax since it does not
      operate on seismic data.  Hence stdin and stdout are not used.

 Caveat: Does not accept a series of interval velocity models to
produce a variable velocity file for sunmo.


 Credits:
CWP: Jack

 Technical Reference:
The Common Depth Point Stack
William A. Schneider
Proc. IEEE, v. 72, n. 10, p. 1238-1254
```

1984

Formulas:
    Note: All sums on i are from 1 to k

From Schneider:
Let h[i] be the ith layer thickness measured at the cmp and
v[i] the ith interval velocity.
Set:
t[i] = h[i]/v[i]
t0[k] = 2 Sum t[i] * cos(dip)
vs[k] = (1.0/cos(dip)) sqrt(Sum v[i]*v[i]*t[i] / Sum t[i])
Define:
t0by2[k] = Sum h[i]/v[i]
vh[k]   = Sum v[i]*h[i]
Then:
t0[k] = 2 * t0by2[k] * cos(dip)
vs[k] = sqrt(vh[k] / t0by2[k]) / cos(dip)

```
 SUSWAPBYTES - SWAP the BYTES in SU data to convert data from big endian
               to little endian byte order, and vice versa

 suswapbytes < stdin [optional parameter] > sdtout

  format=0 foreign to native
  =1 native to foreign
swaphdr=1 swap the header byte order
  =0 do not change the header byte order
swapdata=1 swap the data byte order
  =0 do not change the data byte order
  ns=from header if ns not set in header, must be set by hand
 Notes:
  The 'native'endian is the endian (byte order) of the machine you are
  running this program on. The 'foreign' endian is the opposite byte order.

 Examples of big endian machines are: IBM RS6000, SUN, NeXT
 Examples of little endian machines are: PCs, DEC

 Caveat: this code has not been tested on DEC


 Credits:
CWP: adapted for SU by John Stockwell
based on a code supplied by:
Institute fur Geophysik, Hamburg: Jens Hartmann (June 1993)

 Trace header fields accessed: ns
```

```
 SWAPBHED - SWAP the BYTES in a SEGY Binary tape HEaDer file

 swapbhed < binary_in > binary out

 Required parameter:
  none
 Optional parameters:
none


 Credits:

CWP: John Stockwell  13 May 2011
```

```
SUDATUMFD - 2D zero-offset Finite Difference acoustic wave-equation
DATUMing

sudatumfd <stdin > stdout [optional parameters]

Required parameters:

nt=    number of time samples on each trace
nx=    number of receivers per shot gather
nsx=   number of shot gathers
nz=    number of downward continuation depth steps
dz=    depth sampling interval (in meters)
mx=    number of horizontal samples in the velocity model
mz=    number of vertical samples in the velocity model
vfile1=       velocity file used for thin-lens term
vfile2=       velocity file used for diffraction term
dx=           horizontal sampling interval (in meters)

Optional parameters:

dt=.004       time sampling interval (in seconds)
buff=5 number of zero traces added to each side of each
    shot gather as a pad
tap_len=5     taper length (in number of traces)
x_0=0.0       x coordinate of leftmost position in velocity model

Notes:
The algorithm is a 45-degree implicit-finite-difference scheme based
on the one-way wave equation.  It works on poststack (zero-offset)
data only.  The two velocity files, vfile1 and vfile2, are binary
files containing floats with the format v[ix][iz].  There are two
potentially different velocity files for the thin-lens and
diffraction terms to allow for the use of a zero-velocity layer
which allows for datuming from an irregular surface.

Source and receiver locations must be set in the header values in
order for the datuming to work properly.  The leftmost position of
of the velocity models given in vfile1 and vfile2 must also be given.

Author:  Chris Robinson, 10/16/00, CWP, Colorado School of Mines
```

References:
Beasley, C., and Lynn, W., 1992, The zero-velocity layer: migration from irregular surfaces: Geophysics, 57, 1435-1443.

Claerbout, J. F., 1985, Imaging the earth's interior: Blackwell Scientific Publications.

```
SUDATUMK2DR - Kirchhoff datuming of receivers for 2D prestack data
(shot gathers are the input)

    sudatumk2dr  infile=  outfile=  [parameters]

 Required parameters:
 infile=stdin file for input seismic traces
 outfile=stdout file for common offset migration output
 ttfile= file for input traveltime tables
   The following 9 parameters describe traveltime tables:
 fzt=  first depth sample in traveltime table
 nzt=  number of depth samples in traveltime table
 dzt= depth interval in traveltime table
 fxt= first lateral sample in traveltime table
 nxt= number of lateral samples in traveltime table
 dxt= lateral interval in traveltime table
 fs=  x-coordinate of first source
 ns=  number of sources
 ds=  x-coordinate increment of sources

 fxi=                  x-coordinate of the first surface location
 dxi=                  horizontal spacing on surface
 nxi=                  number of input surface locations
 sgn= Sign of the datuming process (up=-1 or down=1)

 Optional Parameters:
 dt= or from header (dt)  time sampling interval of input data
 ft= or from header (ft)  first time sample of input data
 surf="0,0;99999,0"  The first surface defined the recording surface
 surf="0,0;99999,0"  and the second one, the new datum.
                       "x1,z1;x2,z2;x3,z3;...
 fzo=fzt z-coordinate of first point in output trace
 dzo=0.2*dzt vertical spacing of output trace
 nzo=5*(nzt-1)+1  number of points in output trace ",
 fxso=fxt x-coordinate of first shot
 dxso=0.5*dxt shot horizontal spacing
 nxso=2*(nxt-1)+1   number of shots
 fxgo=fxt x-coordinate of first receiver
 dxgo=0.5*dxt receiver horizontal spacing
 nxgo=nxso number of receivers per shot
 fmax=0.25/dt frequency-highcut for input traces
 offmax=99999 maximum absolute offset allowed in migration
 aperx=nxt*dxt/2   migration lateral aperature
```

```
angmax=60 migration angle aperature from vertical
v0=1500(m/s) reference velocity value at surface
dvz=0.0   reference velocity vertical gradient
antiali=1             Antialiase filter (no-filter = 0)
jpfile=stderr job print file name
mtr=100   print verbal information at every mtr traces
ntr=100000 maximum number of input traces to be migrated

verbose=0 silent, =1 chatty

Notes:
1. Traveltime tables were generated by program rayt2d (or other ones)
   on relatively coarse grids, with dimension ns*nxt*nzt. In the
   datuming process, traveltimes are interpolated into shot/gephone
   positions and output grids.
2. Input traces must be SU format and organized in common rec. gathers
3. If the offset value of an input trace is not in the offset array
   of output, the nearest one in the array is chosen.
4. Amplitudes are computed using the reference velocity profile, v(z),
   specified by the parameters v0= and dvz=.
5. Input traces must specify source and receiver positions via the header
   fields tr.sx and tr.gx. Offset is computed automatically.


Author:  Trino Salinas, 05/01/96,  Colorado School of Mines

This code is based on sukzmig2d.c written by Zhenyue Liu, 03/01/95.
Subroutines from Dave Hale's modeling library were adapted in
this code to define topography using cubic splines.

This code implements a Kirchhoff extraplolation operator that allows to
transfer data from one reference surface to another.  The formula used in
this application is a far field approximation of the Berryhill's original
formula (Berryhill, 1979).  This equation is the result of a stationary
phase analysis to get an analog asymptotic expansion for the two-and-one
half dimensional extrapolation formula (Bleistein, 1984).

The extrapolation formula permits the downward continuation of upgoing
waves and upward continuation of downgoing waves. For upward conti-
nuation of upgoing waves and downward continuation of downgoing waves,
the conjugate transpose of the equation is used (Bevc, 1993).

References :
```

Berryhill, J.R., 1979, Wave equation datuming: Geophysics, 44, 1329--1344.

_____, 1984, Wave equation datuming before stack (short note) : Geophysics, 49, 2064--2067.

Bevc, D., 1993, Data parallel wave equation datuming with irregular acquisition topography : 63rd Ann. Internat. Mtg., SEG, Expanded Abstracts, 197--200.

Bleistein, N., 1984, Mathematical methods for wave phenomena, Academic Press Inc. (Harcourt Brace Jovanovich Publishers), New York.

```
SUDATUMK2DS - Kirchhoff datuming of sources for 2D prestack data
  (input data are receiver gathers)

    sudatumk2ds  infile=  outfile=  [parameters]

 Required parameters:
 infile=stdin file for input seismic traces
 outfile=stdout file for common offset migration output
 ttfile= file for input traveltime tables
   The following 9 parameters describe traveltime tables:
 fzt= first depth sample in traveltime table
 nzt=  number of depth samples in traveltime table
 dzt= depth interval in traveltime table
 fxt= first lateral sample in traveltime table
 nxt= number of lateral samples in traveltime table
 dxt= lateral interval in traveltime table
 fs=  x-coordinate of first source
 ns=  number of sources
 ds=  x-coordinate increment of sources

 fxi=                   x-coordinate of the first surface location
 dxi=                   horizontal spacing on surface
 nxi=                   number of input surface locations
 sgn=                   Sign of the datuming process (up=-1 or down=1)

 Optional Parameters:
 dt= or from header (dt)  time sampling interval of input data
 ft= or from header (ft)  first time sample of input data
 surf="0,0;99999,0"  The first surface defined the recording surface
 surf="0,0;99999,0"  and the second one, the new datum.
                       "x1,z1;x2,z2;x3,z3;...
 fzo=fzt z-coordinate of first point in output trace
 dzo=0.2*dzt vertical spacing of output trace
 nzo=5*(nzt-1)+1  number of points in output trace ",
 fxso=fxt x-coordinate of first shot
 dxso=0.5*dxt shot horizontal spacing
 nxso=2*(nxt-1)+1   number of shots
 fxgo=fxt x-coordinate of first receiver
 exgo=fxgo+(nxgo-1)*dxgo x-coordinate of the last receiver
 dxgo=0.5*dxt receiver horizontal spacing
 nxgo=nxso number of receivers per shot
 fmax=0.25/dt frequency-highcut for input traces
 offmax=99999 maximum absolute offset allowed in migration
```

```
aperx=nxt*dxt/2   migration lateral aperature
angmax=60 migration angle aperature from vertical
v0=1500(m/s) reference velocity value at surface
dvz=0.0   reference velocity vertical gradient
antiali=1             Antialiase filter (no-filter = 0)
jpfile=stderr job print file name
mtr=100   print verbal information at every mtr traces
ntr=100000 maximum number of input traces to be migrated

Notes:
1. Traveltime tables were generated by program rayt2d (or other ones)
   on relatively coarse grids, with dimension ns*nxt*nzt. In the
   migration process, traveltimes are interpolated into shot/gephone
   positions and output grids.
2. Input traces must be SU format and organized in common shot gathers
3. If the offset value of an input trace is not in the offset array
   of output, the nearest one in the array is chosen.
4. Amplitudes are computed using the reference velocity profile, v(z),
   specified by the parameters v0= and dvz=.
5. Input traces must specify source and receiver positions via the header
   fields tr.sx and tr.gx. Offset is computed automatically.


Author:  Trino Salinas, 05/01/96,  Colorado School of Mines

This code is based on sukzmig2d.c written by Zhenyue Liu, 03/01/95.
Subroutines from Dave Hale's modeling library were adapted in
this code to define topography using cubic splines.

This code implements a Kirchhoff extraplolation operator that allows to
transfer data from one reference surface to another.  The formula used in
this application is a far field approximation of the Berryhill's original
formula (Berryhill, 1979).  This equation is the result of a stationary
phase analysis to get an analog asymptotic expansion for the two-and-one
half dimensional extrapolation formula (Bleistein, 1984).

The extrapolation formula permits the downward continuation of upgoing
waves and upward continuation of downgoing waves.  For upward conti-
nuation of upgoing waves and downward continuation of downgoing waves,
the conjugate transpose of the equation is used (Bevc, 1993).

References :
```

Berryhill, J.R., 1979, Wave equation datuming: Geophysics, 44, 1329--1344.

_____, 1984, Wave equation datuming before stack (short note) : Geophysics, 49, 2064--2067.

Bevc, D., 1993, Data parallel wave equation datuming with irregular acquisition topography : 63rd Ann. Internat. Mtg., SEG, Expanded Abstracts, 197--200.

Bleistein, N., 1984, Mathematical methods for wave phenomena, Academic Press Inc. (Harcourt Brace Jovanovich Publishers), New York.

```
 SUKDMDCR - 2.5D datuming of receivers for prestack, common source
           data using constant-background data mapping formula.
           (See selfdoc for specific survey requirements.)

   sukdmdcr  infile=  outfile=  [parameters]

Required file parameters:
infile=stdin file for input seismic traces
outfile=stdout file for output
ttfile file for input traveltime tables

Required parameters describing the traveltime tables:
fzt  first depth sample in traveltime table
nzt  number of depth samples in traveltime table
dzt depth interval in traveltime table
fxt first lateral sample in traveltime table
nxt number of lateral samples in traveltime table
dxt lateral interval in traveltime table
fs  x-coordinate of first source in table
ns  number of sources in table
ds  x-coordinate increment of sources in table

Parameters describing the input data:
nxso                  number of shots
dxso                  shot interval
fxso=0                x-coordinate of first shot
nxgo                  number of receiver offsets per shot
dxgo                  receiver offset interval
fxgo=0                first receiver offset
dt= or from header (dt)       time sampling interval of input data
ft= or from header (ft)       first time sample of input data

Parameters describing the domain of the problem:
dzo=0.2*dzt vertical spacing in surface determination
offmax=99999 maximum absolute offset allowed

Parameters describing the recording and datuming surfaces:
recsurf=0             recording surface (horizontal=0, topographic=1)
zrec                  defines recording surface when recsurf=0
recfile=              defines recording surface when recsurf=1
datsurf=0             datuming surface (horizontal=0, irregular=1)
zdat                  defines datuming surface when datsurf=0
datfile=              defines datuming surface when datsurf=1
```

```
Optional parameters describing the extrapolation:
aperx=nxt*dxt/2   lateral half-aperture
v0=1500(m/s) reference wavespeed
freq=50               dominant frequency in data, used to determine
                      the minimum distance below the datum that
                      the stationary phase calculation is valid.
scale=1.0             user defined scale factor for output
jpfile=stderr job print file name
mtr=100   print verbal information at every mtr traces
ntr=100000 maximum number of input traces to be datumed
```

Computational Notes:

1. Input traces must be SU format and organized in common shot gathers.

2. Traveltime tables were generated by program rayt2d (or equivalent)
   on any grid, with dimension ns*nxt*nzt. In the extrapolation process,
   traveltimes are interpolated into shot/geophone locations and
   output grids.

3. If the offset value of an input trace is not in the offset array
   of output, the nearest one in the array is chosen.

4. Amplitudes are computed using the constant reference wavepeed v0.

5. Input traces must specify source and receiver positions via the header
   fields tr.sx and tr.gx.

6. Recording and datuming surfaces are defined as follows: If recording
   surface is horizontal, set recsurf=0 (default). Then, zrec will be a
   required parameter, set to the depth of surface. If the recording
   surface is topographic, then set recsurf=1. Then, recfile is a required
   input file. The input file recfile should be a single column ascii file
   with the depth of the recording surface at every surface location (first
   source to last offset), with spacing equal to dxgo.

   The same holds for the datuming surface, using datsurf, zdat, and datfile.

Assumptions and limitations:

1. This code implements a 2.5D extraplolation operator that allows to
   transfer data from one reference surface to another. The formula used in
   this application is an adaptation of Bleistein & Jaramillo's 2.5D data
   mapping formula for receiver extrapolation. This is the result of a
   stationary phase analysis of the data mapping equation in the case of
   a constant source location (shot gather).

Credits:

Authors: Steven D. Sheaffer (CWP), 11/97

References: Sheaffer, S., 1999, "2.5D Downward Continuation of the Seismic
Wavefield Using Kirchhoff Data Mapping." M.Sc. Thesis,
Dept. of Mathematical & Computer Sciences,
Colorado School of Mines.

```
 SUKDMDCS - 2.5D datuming of sources for prestack common receiver
      data, using constant-background data-mapping formula.
           (See selfdoc for specific survey geometry requirements.)

   sukdmdcs  infile=  outfile=  [parameters]

Required parameters:
infile=stdin file for input seismic traces
outfile=stdout file for output
ttfile file for input traveltime tables

Required parameters describing the traveltime tables:
fzt  first depth sample in traveltime table
nzt  number of depth samples in traveltime table
dzt depth interval in traveltime table
fxt first lateral sample in traveltime table
nxt number of lateral samples in traveltime table
dxt lateral interval in traveltime table
fs  x-coordinate of first source in table
ns  number of sources in table
ds  x-coordinate increment of sources in table

Parameters describing the input data:
nxso                  number of shots
dxso                  shot interval
fxso=0                x-coordinate of first shot
nxgo                  number of receiver offsets per shot
dxgo                  receiver offset interval
fxgo=0                first receiver offset
dt= or from header (dt)       time sampling interval of input data
ft= or from header (ft)       first time sample of input data
dc=0                  flag for previously datumed receivers:
                         dc=0 receivers on recording surface
                         dc=1 receivers on datum                    ",

Parameters descrbing the domain of the problem:
dzo=0.2*dzt vertical spacing in surface determination
offmax=99999 maximum absolute offset allowed

Parameters describing the recording and datuming surfaces:
recsurf=0             recording surface (horizontal=0, topographic=1)
zrec                  defines recording surface when recsurf=0
recfile=              defines recording surface when recsurf=1
```

```
datsurf=0             datuming surface (horizontal=0, irregular=1)
zdat                  defines datuming surface when datsurf=0
datfile=              defines datuming surface when datsurf=1

Parameters describing the extrapolation:
aperx=nxt*dxt/2   lateral aperture
v0=1500(m/s) reference wavespeed
freq=50               dominant frequency in data, used to determine
                      the minimum distance below the datum that
                      the stationary phase calculation is valid.
scale=1.0             user defined scale factor for output
jpfile=stderr job print file name
mtr=100   print verbal information at every mtr traces
ntr=100000 maximum number of input traces to be datumed
```

Computational Notes:

1. Input traces must be SU format and organized in common receiver gathers.

2. Traveltime tables were generated by program rayt2d (or equivalent) on any grid, with dimension ns*nxt*nzt. In the extrapolation process, traveltimes are interpolated into shot/geophone locations and output grids.

3. If the offset value of an input trace is not in the offset array of output, the nearest one in the array is chosen.

4. Amplitudes are computed using the constant reference wavespeed v0.

5. Input traces must specify source and receiver positions via the header fields tr.sx and tr.gx.

6. Recording and datuming surfaces are defined as follows: If recording surface is horizontal, set recsurf=0 (default). Then, zrec will be a required parameter, set to the depth of surface. If the recording surface is topographic, then set recsurf=1. Then, recfile is a required input file. The input file recfile should be a single column ascii file with the depth of the recording surface at every surface location (first source to last offset), with spacing equal to dxgo.

   The same holds for the datuming surface, using datsurf, zdat, and datfile.

Assumptions and limitations:

1. This code implements a 2.5D extraplolation operator that allows to
   transfer data from one reference surface to another. The formula used in
   this application is an adaptation of Bleistein & Jaramillo's 2.5D data
   mapping formula for receiver extrapolation. This is the result of a
   stationary phase analysis of the data mapping equation in the case of
   a constant input receiver location (receiver gather).

Credits:

Authors: Steven D. Sheaffer (CWP), 11/97

References: Sheaffer, S., 1999, "2.5D Downward Continuation of the Seismic
Wavefield Using Kirchhoff Data Mapping." M.Sc. Thesis,
Dept. of Mathematical & Computer Sciences,
Colorado School of Mines.

```
SUCDDECON - DECONvolution with user-supplied filter by straightforward
       Complex Division in the frequency domain

sucddecon <stdin >stdout [optional parameters]

Required parameters:
filter=  ascii filter values separated by commas
 ...or...
sufile= file containing SU traces to use as filter
                      (must have same number of traces as input data
  for panel=1)
Optional parameters:
panel=0 use only the first trace of sufile as filter
 =1 decon trace by trace an entire gather
pnoise=0.001 white noise factor for stabilizing results
 (see below)
sym=0 not centered, =1 center the output on each trace
verbose=0 silent, =1 chatty

Notes:
For given time-domain input data I(t) (stdin) and deconvolution
filter F(t), the frequency-domain deconvolved trace can be written as:

I(f) I(f) * complex_conjugate[F(f)]
D(f) = ----- ===> D(f) = ------------------------
F(f) |F(f)|^2 + delta

The real scalar delta is introduced to prevent the resulting deconvolved
trace to be dominated by frequencies at which the filter power is close
to zero. As described above, delta is set to some fraction (pnoise) of
the mean of the filter power spectra. Time sampling rate must be the
same in the input data and filter traces. If panel=1 the two input files
must have the same number of traces. Data and filter traces don't need to
necessarily have the same number of samples, but the filter trace length
length be always equal or shorter than the data traces.

Caveat:
You may need to apply frequency filtering to get acceptable output
  sucddecon  ...| sufilter f=f1,f2,f3,f4
where f1,f2,f3,f4 are an acceptable frequency range, and you may need
to mute artifacts that appear at the beginning of the output, as well.

Trace header fields accessed: ns, dt
```

```
 Trace header fields modified: none


 Credits:
CWP: Ivan Vasconcelos
             some changes by John Stockwell
  CAVEATS:
In the option, panel=1 the number of traces in the sufile must be
the same as the number of traces on the input.

 Trace header fields accessed: ns,dt
 Trace header fields modified: none
```

```
 SUFXDECON - random noise attenuation by FX-DECONvolution

 sufxdecon <stdin >stdout [...]

 Required Parameters:

 Optional Parameters:
 taper=.1 length of taper
 fmin=6.       minimum frequency to process in Hz  (accord to twlen)
 fmax=.6/(2*dt)  maximum frequency to process in Hz
 twlen=entire trace  time window length (minimum .3 for lower freqs)
 ntrw=10       number of traces in window
 ntrf=4        number of traces for filter (smaller than ntrw)
 verbose=0 =1 for diagnostic print
 tmpdir= if non-empty, use the value as a directory path prefix
for storing temporary files; else, if the CWP_TMPDIR
environment variable is set, use its value for the path;
else use tmpfile()

 Notes: Each trace is transformed to the frequency domain.
        For each frequency, Wiener filtering, with unity prediction in
        space, is used to predict the next sample.
        At the end of the process, data is mapped back to t-x domain.  ",



 Credits:

CWP: Carlos E. Theodoro (10/07/97)

 References:
Canales(1984):'Random noise reduction' 54th. SEGM
Gulunay(1986):'FXDECON and complex Wiener Predicition
                            filter' 56th. SEGM
Galbraith(1991):'Random noise attenuation by F-X
                            prediction: a tutorial' 61th. SEGM

 Algorithm:
- read data
- loop over time windows
- select data
- FFT (t -> f)
- loop over space windows
```

```
- select data
- loop over frequencies
- autocorelation
- matrix problem
- construct filter
- filter data
- loop along space window
- FFT (f -> t)
- reconstruct data
  - output data

 Trace header fields accessed: ns, dt, d1
 Trace header fields modified:
```

```
 SUPEF - Wiener (least squares) predictive error filtering

 supef <stdin >stdout  [optional parameters]

 Required parameters:
 dt is mandatory if not set in header

 Optional parameters:
 cdp=  CDPs for which minlag, maxlag, pnoise, mincorr,
maxcorr are set (see Notes)
 minlag=dt first lag of prediction filter (sec)
 maxlag=last lag default is (tmax-tmin)/20
 pnoise=0.001 relative additive noise level
 mincorr=tmin start of autocorrelation window (sec)
 maxcorr=tmax end of autocorrelation window (sec)
 wienerout=0 =1 to show Wiener filter on each trace
 mix=1,...   array of weights (floats) for moving
average of the autocorrelations
 outpar=/dev/null output parameter file, contains the Wiener filter
  if wienerout=1 is set
 method=linear  for linear interpolation of cdp values
       =mono for monotonic cubic interpolation of cdps
       =akima for Akima's cubic interpolation of cdps
       =spline for cubic spline interpolation of cdps

 Trace header fields accessed: ns, dt
 Trace header fields modified: none

 Notes:

 1) To apply spiking decon (Wiener filtering with no gap):

 Run the following command

    suacor < data.su | suximage perc=95

 You will see horizontal stripe running across the center of your plot.
 This is the autocorrelation wavelet for each trace. The idea of spiking
 decon is to apply a Wiener filter with no gap to the data to collapse
 the waveform into a spike. The idea is to pick the width of the
 autocorrelation waveform _from beginning to end_ (not trough to trough)
 and use this time for MAXLAG_SPIKING:
```

```
 supef < data.su maxlag=MAXLAG_SPIKING  > dataspiked.su

2) Prediction Error Filter (i.e. gapped Wiener filtering)
The purpose of gapped decon is to suppress repetitions in the data
such as those caused by water bottom multiples.

To look for the period of the repetitions

   suacor ntout=1000 < dataspiked.su | suximage perc=95
or
   suacor ntout=1000 < dataspiked.su | sustack key=dt |suxwigb

The value of ntout must be larger than the default 100. The idea is
to look for repetitions in the autocorrelation. These repetitions will
appear as a family of parallel stripes above and below the main
autocorrelation waveform. Or, if you stack the data, these will be
repetitive spikes.  This repetition time is the GAP. We set
MINLAG_PEF to the value of this repetition time.

We set the minlag to MINLAG_PEF = GAP

We set the maxlag to MAXLAG_PEF = GAP + MAXLAG_SPIKING

 supef < dataspiked.su minlag=MINLAG_PEF maxlag=MAXLAG_PEF > datapef.su

Some experimentation may be required to get a satisfactory result.
In particular you may find that you need to reduce the value of the
minlag

3) It may be effective to sort your data into cdp gathers with susort,
and perform sunmo correction to the water speed with sunmo, prior to
attempts to suppress water bottom multiples. After applying supef, the
user should apply inverse nmo to undo the nmo to water speed prior to
further processing. Or, do the predictive decon on fully nmo-corrected
gathers.

If you flatten your data with sunmo, then make sure that you turn off
the stretch mute by using smute=20

 | sunmo vnmo=v1,v2,... tnmo=t1,t2,... smute=20 | supef ...

For a filter expressed as a function of cdp, specify the array
    cdp=cdp1,cdp2,...
```

```
 and for each cdp specified, specify the minlag and maxlag arrays as
      minlag=min1,min2,...     maxlag=max1,max2,...

 It is required that the number of minlag and maxlag values be equal to
 the number of cdp's specified.  If the number of
 values in these arrays does not equal the number of cdp's, only the first
 value will be used.

 Caveat:
 The wienerout=1 option writes out the wiener filter to outpar, and
 the prediction error filter to stdout, which is ",
     1,0,0,...,-wiener[0],...,-wiener[imaxlag-1]
 where the sample value of -wiener[0], is  iminlag in the pe-filter.
 The pe-filter is output as a SU format datafile, one pe-filter for each
 trace input.
...| supef ... wienerout | suxwigb
 shows the prediction error filters

 Credits:
CWP: Shuki Ronen, Jack K. Cohen, Ken Larner
      CWP: John Stockwell, added mixing feature (April 1998)
      CSM: Tanya Slota (September 2005) added cdp feature

      Technical Reference:
A. Ziolkowski, "Deconvolution", for value of maxlag default:
page 91: imaxlag < nt/10.  I took nt/20.

 Notes:
The prediction error filter is 1,0,0...,0,-wiener[0], ...,
so no point in explicitly forming it.

If imaxlag < 2*iminlag - 1, then we don't need to compute the
autocorrelation for lags:
imaxlag-iminlag+1, ..., iminlag-1
It doesn't seem worth the duplicated code to implement this.

 Trace header fields accessed: ns
```

```
 SUPHIDECON - PHase Inversion Deconvolution

    suphidecon < stdin > stdout

 Required parameters:
none
 Optional parameters:
 ... time range used for wavelet extraction:
 tm=-0.1 Pre zero time (maximum phase component )
 tp=+0.4 Post zero time (minimum phase component + multiples)
 pad=.1 percentage padding for nt prior to cepstrum calculation

 pnoise=0.001 Pre-withening (assumed noise to prevent division by zero)

 Notes:
 The wavelet is separated from the reflectivity and noise based on
 their different 'smoothness' in the pseudo cepstrum domain.
 The extracted wavelet also includes multiples.
 The wavelet is reconstructed in frequency domain, end removed ",
 from the trace. (Method by Lichman and Northwood, 1996.)



 Credits: Potash Corporation, Saskatechwan  Balasz Nemeth
 given to CWP by Potash Corporation 2008 (originally as supid.c)

 Reference:
 Lichman,and Northwood, 1996; Phase Inversion deconvolution for
 long and short period multiples attenuation, in
 SEG Deconvolution 2, Geophysics reprint Series No. 17
 p. 701-718, originally presented at the 54th EAGE meeting, Paris,
 June 1992, revised March 1993, revision accepted September 1994.
```

```
 SUSHAPE - Wiener shaping filter

  sushape <stdin >stdout  [optional parameters]

 Required parameters:
 w= vector of input wavelet to be shaped or ...
 ...or ...
 wfile=        ... file containing input wavelet in SU (SEGY trace) format
 d= vector of desired output wavelet or ...
 ...or ...
 dfile=        ... file containing desired output wavelet in SU format
 dt=tr.dt if tr.dt is not set in header, then dt is mandatory

 Optional parameters:
 nshape=trace length of shaping filter
 pnoise=0.001 relative additive noise level
 showshaper=0 =1 to show shaping filter

 verbose=0 silent; =1 chatty

Notes:

 Example of commandline input wavelets:
sushape < indata  w=0,-.1,.1,... d=0,-.1,1,.1,... > shaped_data

sushape < indata  wfile=inputwavelet.su dfile=desire.su > shaped_data

 To get the shaping filters into an ascii file:
 ... | sushape ... showwshaper=1 2>file | ...   (sh or ksh)
 (... | sushape ... showshaper=1 | ...) >&file  (csh)


 Credits:
CWP: Jack Cohen
CWP: John Stockwell, added wfile and dfile  options

 Trace header fields accessed: ns, dt
 Trace header fields modified: none
```

```
SUDMOFKCW - converted-wave DMO via F-K domain (log-stretch) method for
 common-offset gathers

sudmofkcw <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

Required Parameters:
cdpmin   minimum cdp (integer number) for which to apply DMO
cdpmax   maximum cdp (integer number) for which to apply DMO
dxcdp    distance between adjacent cdp bins (m)
noffmix  number of offsets to mix (see notes)

Optional Parameters:
tdmo=0.0 times corresponding to rms velocities in vdmo (s)
vdmo=1500.0 rms velocities corresponding to times in tdmo (m/s)
gamma=0.5  velocity ratio, upgoing/downgoing
ntable=1000  number of tabulated z/h and b/h (see notes)
sdmo=1.0 DMO stretch factor; try 0.6 for typical v(z)
flip=0  =1 for negative shifts and exchanging s1 and s2
  (see notes below)
fmax=0.5/dt maximum frequency in input traces (Hz)
verbose=0 =1 for diagnostic print

Notes:
Input traces should be sorted into common-offset gathers.  One common-
offset gather ends and another begins when the offset field of the trace
headers changes.

The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
the cdp location expressed in units of meters or feet.

The number of offsets to mix (noffmix) should typically equal the ratio of
the shotpoint spacing to the cdp spacing.  This choice ensures that every
cdp will be represented in each offset mix.  Traces in each mix will
contribute through DMO to other traces in adjacent cdps within that mix.

The tdmo and vdmo arrays specify a velocity function of time that is
used to implement a first-order correction for depth-variable velocity.
The times in tdmo must be monotonically increasing. The velocity function
is assumed to have been gotten by traditional NMO.

For each offset, the minimum time at which a non-zero sample exists is
used to determine a mute time.  Output samples for times earlier than this
mute time will be zeroed.  Computation time may be significantly reduced
```

```
 if the input traces are zeroed (muted) for early times at large offsets.

 z/h is horizontal-reflector depth normalized to half source-reciver offset
 h.  Normalized shift of conversion point is b/h.  The code now does not
 support signed offsets, therefore it is recommended that only end-on data,
 not split-spread, be used as input (of course after being sorted into
 common-offset gathers).  z/h vs b/h depends on gamma (see Alfaraj's Ph.D.
 thesis, 1993).

 Flip factor = 1 implies positive shift of traces (in the increasing CDP
 bin number direction).  When processing split-spread data, for example,
 if one side of the spread is processed with flip=0, then the other side
 of the spread should be processed with flip=1.  The flip factor also
 determines the actions of the factors s1 and s2, i.e., stretching or
 squeezing.

 Trace header fields accessed:  nt, dt, delrt, offset, cdp.


 Credits:
CWP: Mohamed Alfaraj
Dave Hale

 Technical Reference:
Transformation to zero offset for mode-converted waves
Mohammed Alfaraj, Ph.D. thesis, 1993, Colorado School of Mines

Dip-Moveout Processing - SEG Course Notes
Dave Hale, 1988
```

```
 SUDMOFK - DMO via F-K domain (log-stretch) method for common-offset gathers

 sudmofk <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

 Required Parameters:
 cdpmin minimum cdp (integer number) for which to apply DMO
 cdpmax maximum cdp (integer number) for which to apply DMO
 dxcdp distance between adjacent cdp bins (m)
 noffmix number of offsets to mix (see notes)

 Optional Parameters:
 tdmo=0.0 times corresponding to rms velocities in vdmo (s)
 vdmo=1500.0 rms velocities corresponding to times in tdmo (m/s)
 sdmo=1.0 DMO stretch factor; try 0.6 for typical v(z)
 fmax=0.5/dt maximum frequency in input traces (Hz)
 verbose=0 =1 for diagnostic print
 tmpdir= if non-empty, use the value as a directory path prefix
for storing temporary files; else if the CWP_TMPDIR
environment variable is set use its value for the path;
else use tmpfile()

 Notes:
 Input traces should be sorted into common-offset gathers.  One common-
 offset gather ends and another begins when the offset field of the trace
 headers changes.

 The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
 the cdp location expressed in units of meters or feet.

 The number of offsets to mix (noffmix) should typically no smaller than
 the ratio of the shotpoint spacing to the cdp spacing.  This choice
 ensures that every cdp will be represented in each offset mix.  Traces
 in each mix will contribute through DMO to other traces in adjacent cdps
 within that mix. (Values of noffmix 2 or 3 times the ratio of shotpoint
 spacing to the cdp spacing often yield better results.)

 The tdmo and vdmo arrays specify a velocity function of time that is
 used to implement a first-order correction for depth-variable velocity.
 The times in tdmo must be monotonically increasing.

 For each offset, the minimum time at which a non-zero sample exists is
 used to determine a mute time.  Output samples for times earlier than this",
 mute time will be zeroed.  Computation time may be significantly reduced
```

```
 if the input traces are zeroed (muted) for early times at large offsets.

 Credits:
CWP: Dave

 Technical Reference:
Dip-Moveout Processing - SEG Course Notes
Dave Hale, 1988

 Trace header fields accessed:  ns, dt, delrt, offset, cdp.
```

```
SUDMOTIVZ - DMO for Transeversely Isotropic V(Z) media for common-offset
           gathers

sudmotivz <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

Required Parameters:
cdpmin          minimum cdp (integer number) for which to apply DMO
cdpmax          maximum cdp (integer number) for which to apply DMO
dxcdp           distance between adjacent cdp bins (m)
noffmix         number of offsets to mix (see notes)

Optional Parameters:
vnfile=         binary (non-ascii) file containing NMO interval
                  velocities (m/s)
vfile=          binary (non-ascii) file containing interval velocities (m/s)
etafile=        binary (non-ascii) file containing eta interval values (m/s)
tdmo=0.0        times corresponding to interval velocities in vdmo (s)
vndmo=1500.0    NMO interval velocities corresponding to times in tdmo (m/s)
vdmo=vndmo     interval velocities corresponding to times in tdmo (m/s)
etadmo=1500.0   eta interval values corresponding to times in tdmo (m/s)
fmax=0.5/dt     maximum frequency in input traces (Hz)
smute=1.5       stretch mute used for NMO correction
speed=1.0       twist this knob for speed (and aliasing)
verbose=0       =1 for diagnostic print

Notes:
Input traces should be sorted into common-offset gathers.  One common-
offset gather ends and another begins when the offset field of the trace
headers changes.

The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
the cdp location expressed in units of meters or feet.

The number of offsets to mix (noffmix) should typically equal the ratio of
the shotpoint spacing to the cdp spacing.  This choice ensures that every
cdp will be represented in each offset mix.  Traces in each mix will
contribute through DMO to other traces in adjacent cdps within that mix.

vnfile, vfile and etafile should contain the regularly sampled interval
values of NMO velocity, velocity and eta respectivily as a
function of time.  If, for example, vfile is not supplied, the interval
velocity function is defined by linear interpolation of the values in the
tdmo and vdmo arrays.  The times in tdmo must be monotonically increasing.
```

If vfile or vdmo are not given it will be equated to vnfile or vndmo.

For each offset, the minimum time to process is determined using the smute parameter. The DMO correction is not computed for samples that have experienced greater stretch during NMO.

Trace header fields accessed: nt, dt, delrt, offset, cdp.

```
 SUDMOTX - DMO via T-X domain (Kirchhoff) method for common-offset gathers

 sudmotx <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [optional parms]

 Required Parameters:
 cdpmin                  minimum cdp (integer number) for which to apply DMO
 cdpmax                  maximum cdp (integer number) for which to apply DMO
 dxcdp                   distance between successive cdps
 noffmix                 number of offsets to mix (see notes)

 Optional Parameters:
 offmax=3000.0           maximum offset
 tmute=2.0               mute time at maximum offset offmax
 vrms=1500.0             RMS velocity at mute time tmute
 verbose=0               =1 for diagnostic print
 tmpdir= if non-empty, use the value as a directory path prefix
for storing temporary files; else if the CWP_TMPDIR
environment variable is set use its value for the path;
else use tmpfile()


 Notes:
 Input traces should be sorted into common-offset gathers.  One common-
 offset gather ends and another begins when the offset field of the trace
 headers changes.

 The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
 the cdp location expressed in units of meters or feet.

 The number of offsets to mix (noffmix) should typically equal the ratio of
 the shotpoint spacing to the cdp spacing.  This choice ensures that every
 cdp will be represented in each offset mix.  Traces in each mix will
 contribute through DMO to other traces in adjacent cdps within that mix.

 The defaults for offmax and vrms are appropriate only for metric units.
 If distances are measured in feet, then these parameters should be
 specified explicitly.

 offmax, tmute, and vrms need not be specified precisely.
 If these values are unknown, then one should overestimate offmax
 and underestimate tmute and vrms.

 No muting is actually performed.  The tmute parameter is used only to
```

```
 determine parameters required to perform DMO.

 Credits:
CWP: Dave Hale

 Technical Reference:
      A non-aliased integral method for dip-moveout
      Dave Hale
      submitted to Geophysics, June, 1990

 Trace header fields accessed:  ns, dt, delrt, offset, cdp.
```

```
SUDMOVZ - DMO for V(Z) media for common-offset gathers

sudmovz <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

Required Parameters:
cdpmin          minimum cdp (integer number) for which to apply DMO
cdpmax          maximum cdp (integer number) for which to apply DMO
dxcdp           distance between adjacent cdp bins (m)
noffmix         number of offsets to mix (see notes)

Optional Parameters:
vfile=          binary (non-ascii) file containing interval velocities (m/s)
tdmo=0.0        times corresponding to interval velocities in vdmo (s)
vdmo=1500.0     interval velocities corresponding to times in tdmo (m/s)
fmax=0.5/dt     maximum frequency in input traces (Hz)
smute=1.5       stretch mute used for NMO correction
speed=1.0       twist this knob for speed (and aliasing)
verbose=0       =1 for diagnostic print

Notes:
Input traces should be sorted into common-offset gathers.  One common-
offset gather ends and another begins when the offset field of the trace
headers changes.

The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
the cdp location expressed in units of meters or feet.

The number of offsets to mix (noffmix) should typically equal the ratio of
the shotpoint spacing to the cdp spacing.  This choice ensures that every
cdp will be represented in each offset mix.  Traces in each mix will
contribute through DMO to other traces in adjacent cdps within that mix.

vfile should contain the regularly sampled interval velocities as a
function of time.  If vfile is not supplied, the interval velocity
function is defined by linear interpolation of the values in the tdmo
and vdmo arrays.  The times in tdmo must be monotonically increasing.

For each offset, the minimum time to process is determined using the
smute parameter.  The DMO correction is not computed for samples that
have experienced greater stretch during NMO.

Trace header fields accessed:  nt, dt, delrt, offset, cdp.
```

```
 SUTIHALEDMO - TI Hale Dip MoveOut (based on Hale's PhD thesis)

  sutihaledmo <infile >outfile [optional parameters]


 Required Parameters:
 nxmax   maximum number of midpoints in common offset gather

 Optional Parameters:
 option=1 1 = traditional Hale DMO (from PhD thesis)
2 = Bleistein's true amplitude DMO
3 = Bleistein's cos*cos weighted DMO
4 = Zhang's DMO
5 = Tsvankin's anisotropic DMO
6 = Tsvankin's VTI DMO weak anisotropy approximation
 dx=50.  midpoint sampling interval between traces
in a common offset gather.  (usually shot
interval in meters)
 v=1500.0 velocity (in meters/sec)
(must enter a positive value for option=3)
(for excluding evanescent energy)
 h=200.0 source-receiver half-offset (in meters)
 ntpad=0 number of time samples to pad
 nxpad=h/dx number of midpoints to pad
 file=vnmo name of file with vnmo as a function of p
used for option=5--otherwise not used
(Generate this file by running program
sutivel with appropriate list of Thomsen's
parameters.)
 e=0. Thompsen's epsilon
 d=0. Thompsen's delta

Note:

 This module assumes a single common offset gather after NMO is
 to be input, DMO corrected, and output.  It is useful for computing
 theoretical DMO impulse responses.  The Hale algorithm is
 computationally intensive and not commonly used for bulk processing
 of all of the offsets on a 2-D line as there are cheaper alternative
 algorithms.  The Hale algorithm is commonly used in theoretical studies.
 Bulk processing for multiple common offset gathers is typically done
 using other modules.
```

```
 Test run:   suspike | sutihaledmo nxmax=32 option=1 v=1500 | suxwigb &


 Author:  (Visitor to CSM from Mobil) John E. Anderson Spring 1994
 References: Anderson, J.E., and Tsvankin, I., 1994, Dip-moveout by
Fourier transform in anisotropic media, CWP-146
```

```
 SUBFILT - apply Butterworth bandpass filter

 subfilt <stdin >stdout [optional parameters]

 Required parameters:
  if dt is not set in header, then dt is mandatory

 Optional parameters: (nyquist calculated internally)
  zerophase=1 =0 for minimum phase filter
  locut=1 =0 for no low cut filter
  hicut=1 =0 for no high cut filter
  fstoplo=0.10*(nyq) freq(Hz) in low cut stop band
  astoplo=0.05 upper bound on amp at fstoplo
  fpasslo=0.15*(nyq) freq(Hz) in low cut pass band
  apasslo=0.95 lower bound on amp at fpasslo
  fpasshi=0.40*(nyq) freq(Hz) in high cut pass band
  apasshi=0.95 lower bound on amp at fpasshi
  fstophi=0.55*(nyq) freq(Hz) in high cut stop band
  astophi=0.05 upper bound on amp at fstophi
  verbose=0 =1 for filter design info
  dt = (from header) time sampling interval (sec)

  ... or  set filter by defining  poles and 3db cutoff frequencies
npoleselo=calculated     number of poles of the lo pass band
npolesehi=calculated     number of poles of the lo pass band
f3dblo=calculated frequency of 3db cutoff frequency
f3dbhi=calculated frequency of 3db cutoff frequency

 Notes:
 Butterworth filters were originally of interest because they
 can be implemented in hardware form through the combination of
 inductors, capacitors, and an amplifier. Such a filter can be
 constructed in such a way as to have very small oscillations
 in the flat portion of the bandpass---a desireable attribute.
 Because the filters are composed of LC circuits, the impulse
 response is an ordinary differential equation, which translates
 into a polynomial in the transform domain. The filter is expressed
 as the division by this polynomial. Hence the poles of the filter
 are of interest.

 The user may define low pass, high pass, and band pass filters
 that are either minimum phase or are zero phase.  The default
 is to let the program calculate the optimal number of poles in
```

```
 low and high cut bands.

 Alternately the user may manually define the filter by the 3db
 frequency and by the number of poles in the low and or high
 cut region.

 The advantage of using the alternate method is that the user
 can control the smoothness of the filter. Greater smoothness
 through a larger pole number results in a more bell shaped
 amplitude spectrum.

 For simple zero phase filtering with sin squared tapering use
 "sufilter".

 Credits:
CWP: Dave Hale c. 1993 for bf.c subs and test drivers
CWP: Jack K. Cohen for su wrapper c. 1993
      SEAM Project: Bruce Verwest 2009 added explicit pole option
                    in a program called "subfiltpole"
      CWP: John Stockwell (2012) combined Bruce Verwests changes
           into the original subfilt.

 Caveat: zerophase will not do good if trace has a spike near
   the end.  One could make a try at getting the "effective"
   length of the causal filter, but padding the traces seems
   painful in an already expensive algorithm.


 Theory:
 The

 Trace header fields accessed: ns, dt, trid
```

```
SUCCFILT -  FX domain Correlation Coefficient FILTER

  sucff < stdin > stdout [optional parameters]

Optional parameters:
cch=1.0 Correlation coefficient high pass value
ccl=0.3 Correlation coefficient low pass value
key=ep ensemble identifier
padd=25 FFT padding in percentage

Notes:
This program uses "get_gather" and "put_gather" so requires that
the  data be sorted into ensembles designated by "key", with the ntr
field set to the number of traces in each respective ensemble.

Example:
susort ep offset < data.su > datasorted.su
suputgthr dir=Data verbose=1 < datasorted.su
sugetgthr dir=Data verbose=1 > dataupdated.su
succfilt  < dataupdated.su > ccfiltdata.su

(Work in progress, editing required)


Credits:
 Potash Corporation: Balazs Nemeth, Saskatoon Canada. c. 2008
```

```
 SUDIPFILT - DIP--or better--SLOPE Filter in f-k domain

 sudipfilt <infile >outfile [optional parameters]

 Required Parameters:
 dt=(from header) if not set in header then mandatory
 dx=(from header, d1) if not set in header then mandatory

 Optional parameters:
 slopes=0.0 monotonically increasing slopes
 amps=1.0 amplitudes corresponding to slopes
 bias=0.0 slope made horizontal before filtering

 verbose=0 verbose = 1 echoes information

 tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

 Notes:
 d2 is an acceptable alias for dx in the getpar

 Slopes are defined by delta_t/delta_x, where delta
 means change. Units of delta_t and delta_x are the same
 as dt and dx. It is sometimes useful to fool the program
 with dx=1 dt=1, thus avoiding units and small slope values.

 Linear interpolation and constant extrapolation are used to
 determine amplitudes for slopes that are not specified.
 Linear moveout is removed before slope filtering to make
 slopes equal to bias appear horizontal.  This linear moveout
 is restored after filtering.  The bias parameter may be
 useful for spatially aliased data.  The slopes parameter is
 compensated for bias, so you need not change slopes when you
 change bias.


 Credits:

CWP: Dave (algorithm--originally called slopef)
     Jack (reformatting for SU)
```

```
Trace header fields accessed: ns, dt, d2
```

```
 SUFILTER - applies a zero-phase, sine-squared tapered filter

 sufilter <stdin >stdout [optional parameters]

 Required parameters:
       if dt is not set in header, then dt is mandatory

 Optional parameters:
       f=f1,f2,...             array of filter frequencies(HZ)
       amps=a1,a2,...          array of filter amplitudes
       dt = (from header)      time sampling interval (sec)
verbose=0 =1 for advisory messages

 Defaults:f=.10*(nyquist),.15*(nyquist),.45*(nyquist),.50*(nyquist)
                         (nyquist calculated internally)
          amps=0.,1.,...,1.,0.  trapezoid-like bandpass filter

 Examples of filters:
 Bandpass:   sufilter <data f=10,20,40,50 | ...
 Bandreject: sufilter <data f=10,20,30,40 amps=1.,0.,0.,1. | ..
 Lowpass:    sufilter <data f=10,20,40,50 amps=1.,1.,0.,0. | ...
 Highpass:   sufilter <data f=10,20,40,50 amps=0.,0.,1.,1. | ...
 Notch:      sufilter <data f=10,12.5,35,50,60 amps=1.,.5,0.,.5,1. |..

 Credits:
      CWP: John Stockwell, Jack Cohen
CENPET: Werner M. Heigl - added well log support

 Possible optimization: Do assignments instead of crmuls where
 filter is 0.0.

 Trace header fields accessed: ns, dt, d1
```

```
 SUFRAC -- take general (fractional) time derivative or integral of
    data, plus a phase shift.  Input is CAUSAL time-indexed
    or depth-indexed data.

 sufrac power= [optional parameters] <indata >outdata

 Optional parameters:
power=0 exponent of (-i*omega)
=0  ==> phase shift only
>0  ==> differentiation
<0  ==> integration

sign=-1 sign in front of i * omega
dt=(from header) time sample interval (in seconds)
phasefac=0 phase shift by phase=phasefac*PI
verbose=0 =1 for advisory messages

 Examples:
  preprocess to correct 3D data for 2.5D migration
         sufrac < sudata power=.5 sign=1 | ...
  preprocess to correct susynlv, susynvxz, etc. (2D data) for 2D migration
         sufrac < sudata phasefac=.25 | ...
 The filter is applied in frequency domain.
 if dt is not set in header, then dt is mandatory

 Algorithm:
g(t) = Re[INVFTT{ ( (sign) iw)^power FFT(f)}]
 Caveat:
 Large amplitude errors will result if the data set has too few points.

 Good numerical integration routine needed!
 For example, see Gnu Scientific Library.


 Credits:
CWP: Chris Liner, Jack K. Cohen, Dave Hale (pfas)
      CWP: Zhenyue Liu and John Stockwell added phase shift option
CENPET: Werner M. Heigl - added well log support

 Trace header fields accessed: ns, dt, trid, d1
/
```

```
 SUFWATRIM - FX domain Alpha TRIM

  sufwatrim  <stdin > stdout [optional parameters]

 Required parameters:
 key=key1,key2,.. Header words defining mixing dimesnion
 dx=d1,d2,.. Distance units for each header word
 Optional parameters:
 keyg=ep Header word indicating the start of gather
 vf=0 =1 Do a frequency dependent mix
 vmin=5000 Velocity of the reflection slope
than should not be attenuated
 Notes:
 Trace with the header word mark set to one will be
 the output trace


 Credits: Potash Corporation of Saskatchewan, Balasz Nemeth
 Code given to CWP in 2008
```

```
 SUK1K2FILTER - symmetric box-like K-domain filter defined by the
  cartesian product of two sin^2-tapered polygonal
  filters defined in k1 and k2

      suk1k2filter <infile >outfile [optional parameters]

 Optional parameters:
 k1=val1,val2,... array of K1 filter wavenumbers
 k2=val1,val2,... array of K2 filter wavenumbers
 amps1=a1,a2,... array of K1 filter amplitudes
 amps2=a1,a2,... array of K2 filter amplitudes
 d1=tr.d1 or 1.0 sampling interval in first (fast) dimension
 d2=tr.d1 or 1.0 sampling interval in second (slow) dimension
 quad=0 =0 all four quandrants
=1 (quadrants 1 and 4)
=2 (quadrants 2 and 3)

 Defaults:
 k1=.10*(nyq1),.15*(nyq1),.45*(nyq1),.50*(nyq1)
 k2=.10*(nyq2),.15*(nyq2),.45*(nyq2),.50*(nyq2)
 amps1=0.,1.,...,1.,0.  trapezoid-like bandpass filter
 amps2=0.,1.,...,1.,0.  trapezoid-like bandpass filter

 The nyquist wavenumbers, nyq1 and nyq2, are computed internally.

 verbose=0 verbose = 1 echoes information

 tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

 Notes:
 The filter is assumed to be symmetric, to yield real output

 Because the data are assumed to be purely spatial (i.e. non-seismic),
 the data are assumed to have trace id (30), corresponding to (z,x) data

 The relation: w = 2 pi F is well known for frequency, but there
 doesn't seem to be a commonly used letter corresponding to F for the
 spatial conjugate transform variables.  We use K1 and K2 for this.
 More specifically we assume a phase:
-i(k1 x1 + k2 x2) = -2 pi i(K1 x1 + K2 x2).
```

```
and K1, K2 define our respective wavenumbers.


Credits:
    CWP: John Stockwell, November 1995.

Trace header fields accessed: ns, d1, d2
```

```
SUKFILTER - radially symmetric K-domain, sin^2-tapered, polygonal
 filter

    sukfilter <infile >outfile [optional parameters]

Optional parameters:
k=val1,val2,... array of K filter wavenumbers
amps=a1,a2,... array of K filter amplitudes
d1=tr.d1 or 1.0 sampling interval in first (fast) dimension
d2=tr.d1 or 1.0 sampling interval in second (slow) dimension

Defaults:
k=.10*(nyq),.15*(nyq),.45*(nyq),.50*(nyq)
amps=0.,1.,...,1.,0.  trapezoid-like bandpass filter

The nyquist wavenumbers, nyq=sqrt(nyq1^2 + nyq2^2) is  computed
internally.

Notes:
The filter is assumed to be symmetric, to yield real output.

Because the data are assumed to be purely spatial (i.e. non-seismic),
the data are assumed to have trace id (30), corresponding to (z,x) data

The relation: w = 2 pi F is well known for frequency, but there
doesn't seem to be a commonly used letter corresponding to F for the
spatial conjugate transform variables.  We use K1 and K2 for this.
More specifically we assume a phase:
-i(k1 x1 + k2 x2) = -2 pi i(K1 x1 + K2 x2).
and K1, K2 define our respective wavenumbers.


Credits:
    CWP: John Stockwell, June 1997.

Trace header fields accessed: ns, d1, d2
```

```
 SUKFRAC - apply FRACtional powers of i|k| to data, with phase shift

     sukfrac <infile >outfile [optional parameters]

 Optional parameters:
  power=0 exponent of (i*sqrt(k1^2 + k2^2))^power
=0 ===> phase shift only
>0 ===> differentiation
<0 ===> integration
  sign=1 sign on transform exponent
  d1=1.0 x1 sampling interval
  d2=1.0 x2 sampling interval
  phasefac=0 phase shift by phase=phasefac*PI
  ...directional derivative, active only if angle= is set ....
  angle= if set applies operator directionally in the
direction specified by the value of angle,
-360.0 <= angle < 359.99999 degress

 Notes:
 The relation: w = 2 pi F is well known for frequency, but there
 doesn't seem to be a commonly used letter corresponding to F for the
 spatial conjugate transform variables.  We use K1 and K2 for this.
 More specifically we assume a phase:
-i(k1 x1 + k2 x2) = -2 pi i(K1 x1 + K2 x2).
 and K1, K2 define our respective wavenumbers.

 Algorithms
  g(x1,x2)=Re[2DINVFFT{ ( (sign) i |k|)^power 2DFFT(f)}e^i(phase)]
 where:
       |k| = sqrt[ (k1)^2 + (k2)^2 ]
 or when angle= is set
       |k| = sqrt[ (k1  cos(angle) )^2 + (k2 sin(angle) )^2 ]

 In the default mode a factor of (i|k|)^(power) is applied in the
 transform domain. For time data the time axis direction is taken to
 be the k1-direction.  The effect of this filter is to differentiate
 the input in the normal direction to any curvilinear features in
 in the data, and thus be a non-directional-specific edge enhancer.

 If angle= is set, then the intended effect is a derivative in the
 direction specified by the angle, with the k1-direction being angle=0,
 corresponding to curves whose normals lie in the x1-direction.
```

```
Caveat:
Large amplitude errors will result of the data set has too few points.

Examples:
Edge sharpening:
Laplacean :
   sukfrac < image_data  power=2 phasefac=-1 | ...

Image enhancement:
  Derivative filter:
   sukfrac < image_data  power=1 phasefac=-.5 | ...

Image enhancement:
  Half derivative (this one is the best for photographs):
   sukfrac < image_data  power=.5 phasefac=-.25 | ...


Credits:
    CWP: John Stockwell, June 1997, based on sufrac.

Trace header fields accessed: ns, d1, d2
```

```
 SULFAF -  Low frequency array forming ",

  sulfaf < stdin > stdout [optional parameters]

 Optional Parameters:
 key=ep header keyword describing the gathers
 f1=3 lower frequency cutof
 f2=20 high frequency cutof
 fr=5 frequency ramp

 vel=1000 surface wave velocity
 dx=10 trace spacing
 maxmix=tr.ntr default is the entire gather
 adb=1.0 add back ratio 1.0=pure filtered 0.0=origibal

 tri=0 1 Triangular weight in mixing window

 Notes:
 The traces transformed into the freqiency domain
 where a trace mix is performed in the specifed frequency range
 as Mix=vel/(freq*dx)

 This program uses "get_gather" and "put_gather" so requires that
 the  data be sorted into ensembles designated by "key", with the ntr
 field set to the number of traces in each respective ensemble.

 Example:
 susort ep offset < data.su > datasorted.su
 suputgthr dir=Data verbose=1 < datasorted.su
 sugetgthr dir=Data verbose=1 > dataupdated.su
 sulfaf  < dataupdated.su > ccfiltdata.su

 (Work in progress, editing required)

define LOOKFAC 1 /* Look ahead factor for npfaro
define PFA_MAX 720720  /* Largest allowed nfft */
define PIP2 PI/2.0 /* IP/2 */

int
main( int argc, char *argv[] )
{
cwp_String key; /* header key word from segy.h */
cwp_String type; /* ... its type */
```

```
Value val; /* ... its value */
segy **rec_o; /* trace header+data matrix */
int first=0; /* true when we passed the first gather
int ng=0;
float dt; /* time sampling interval */
int nt; /* num time samples per trace */
int ntr; /* num of traces per ensemble */

int nfft=0; /* lenghth of padded array */
float snfft; /* scale factor for inverse fft
int nf=0; /* number of frequencies */
float d1; /* frequency sampling interval. */
float *rt=NULL; /* real trace */
complex *ct=NULL; /* complex trace */
float **ffdr=NULL; /* frequency domain data */
float **ffdi=NULL; /* frequency domain data */
float **ffdrm=NULL; /* frequency domain mixed data */
float **ffdim=NULL; /* frequency domain mixed data */

int verbose; /* flag: =0 silent; =1 chatty */


float f1; /* minimum frequency */
int if1; /* ...    ...  integerized */
float f2; /* maximum frequency */
int if2; /* ...    ...  integerized */
float fr; /* slope of frequency ramp */
int ifr; /* ...    ...  integerized */
float vel; /* velocity of guided waves */
float dx; /* spatial sampling intervall */
int maxmix; /* size of mix */
int tri; /* flag: =1 trianglular window */
float adb; /* add back ratio */

/* Initialize
initargs(argc, argv);
requestdoc(1);

if (!getparstring("key", &key)) key = "ep";
if (!getparfloat("f1", &f1)) f1 = 3.0;
if (!getparfloat("f2", &f2)) f2 = 20.0;
if (!getparfloat("dx", &dx)) dx = 10;
if (!getparfloat("vel", &vel)) vel = 1000;
```

```
if (!getparfloat("fr", &fr)) fr = 5;
if (!getparint("maxmix", &maxmix)) maxmix = -1;
if (!getparint("tri", &tri)) tri = 0;
if (!getparfloat("adb", &adb)) adb = 1.0;

if (!getparint("verbose", &verbose)) verbose = 0;

/* get the first record
rec_o = get_gather(&key,&type,&val,&nt,&ntr,&dt,&first);
if(ntr==0) err("Can't get first record\n");

/* set up the fft
nfft = npfaro(nt, LOOKFAC * nt);
 if (nfft >= SU_NFLTS || nfft >= PFA_MAX)
  err("Padded nt=%d--too big", nfft);
 nf = nfft/2 + 1;
 snfft=1.0/nfft;
d1=1.0/(nfft*dt);

ct=ealloc1complex(nf);
rt=ealloc1float(nfft);

if1=NINT(f1/d1);
if2=NINT(f2/d1);
ifr=NINT(fr/d1);

do {
if(maxmix==-1) maxmix=ntr;
ng++;

/* Allocate arrays for fft
ffdr = ealloc2float(nf,ntr);
ffdi = ealloc2float(nf,ntr);
ffdrm = ealloc2float(if2+ifr,ntr);
ffdim = ealloc2float(if2+ifr,ntr);
{ int itr,iw;
for(itr=0;itr<ntr;itr++) {

memcpy( (void *) rt, (const void *) (*rec_o[itr]).data, nt*FSIZE);

memset( (void *) &rt[nt], 0,(nfft-nt)*FSIZE);

pfarc(1,nfft,rt,ct);
```

```
for(iw=0;iw<nf;iw++) {
ffdr[itr][iw] = ct[iw].r;
ffdi[itr][iw] = ct[iw].i;
}
}
}

/* Mixing
{ int mix,iw,nmix;
  int ims,ime;
  int itr,itrm,iww,ws;
  float tmpr,tmpi,wh;

for(iw=if1;iw<if2+ifr;iw++) {

mix=MIN(NINT(vel/iw*d1*dx),maxmix);
if(!ISODD(mix)) mix -=1;
if (verbose) warn(" %f %d",iw*d1,mix);

for(itr=0;itr<ntr;itr++) {

ims=MAX(itr-mix/2,0);
ime=MIN(ims+mix,ntr-1);

tmpr=0.0; tmpi=0.0;
wh=1.0; ws=mix/2;
nmix=0;
for(itrm=ims,iww=-mix/2;itrm<ime;++itrm,++iww) {
++nmix;
if(tri) wh = (float)(ws-abs(iww));
tmpr+=ffdr[itrm][iw]*wh;
tmpi+=ffdi[itrm][iw]*wh;
}
ffdrm[itr][iw]=tmpr/nmix;
ffdim[itr][iw]=tmpi/nmix;
}
}

for(iw=if1;iw<if2;iw++) {
for(itr=0;itr<ntr;itr++) {
ffdr[itr][iw]=ffdrm[itr][iw]*adb+ffdr[itr][iw]*(1.0-adb);
ffdi[itr][iw]=ffdim[itr][iw]*adb+ffdi[itr][iw]*(1.0-adb);
```

```
}
}

for(iw=if2,iww=0;iw<if2+ifr;iw++,iww++) {

wh=(float)(1.0-(float)iww/(float)ifr);

for(itr=0;itr<ntr;itr++) {
ffdr[itr][iw] = (wh*ffdrm[itr][iw]+ffdr[itr][iw]*(1.0-wh))*adb+ffdr[itr][iw]*(1.0-adb);
ffdi[itr][iw] = (wh*ffdim[itr][iw]+ffdi[itr][iw]*(1.0-wh))*adb+ffdi[itr][iw]*(1.0-adb);
}
}

}


{ int itr,iw;
for(itr=0;itr<ntr;itr++) {

for(iw=0;iw<nf;iw++) {
ct[iw].r = ffdr[itr][iw]*snfft;
ct[iw].i = ffdi[itr][iw]*snfft;
}

pfacr(-1,nfft,ct,rt);
memcpy( (void *) (*rec_o[itr]).data, (const void *) rt, nt*FSIZE);

}
}

rec_o = put_gather(rec_o,&nt,&ntr);

free2float(ffdr);
free2float(ffdi);
free2float(ffdrm);
free2float(ffdim);
rec_o = get_gather(&key,&type,&val,&nt,&ntr,&dt,&first);

} while(ntr);

warn("Number of gathers %10d\n",ng);

return EXIT_SUCCESS;
```

```
}
```

```
SUMEDIAN - MEDIAN filter about a user-defined polygonal curve with
  the distance along the curve specified by key header word

sumedian <stdin >stdout xshift= tshift= [optional parameters]

Required parameters:
xshift=               array of position values as specified by
                      the 'key' parameter
tshift=               array of corresponding time values (sec)
 ... or input via files:
nshift=               number of x,t values defining median times
xfile=                file containing position values as specified by
                      the 'key' parameter
tfile=                file containing corresponding time values (sec)

Optional parameters:
key=tracl             Key header word specifying trace number
                      =offset  use trace offset instead

mix=.6,1,1,1,.6       array of weights for mix (weighted moving average)
median=0              =0  for mix
                      =1  for median filter
nmed=5                odd no. of traces to median filter
sign=-1               =-1  for upward shift
                      =+1  for downward shift
subtract=1            =1  subtract filtered data from input
                      =0  don't subtract
verbose=0             =1  echoes information

tmpdir=   if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
        the CWP_TMPDIR environment variable is set use
        its value for the path; else use tmpfile()

Notes:
------
Median filtering is a process for suppressing a particular moveout on
seismic sections. Its advantage over traditional dip filtering is that
events with arbitrary moveout may be suppressed. Median filtering is
commonly used in up/down wavefield separation of VSP data.

The process generally consists of 3 steps:
1. A copy of the data panel is shifted such that the polygon in x,t
```

specifying moveout is flattened to horizontal. The x,t pairs are specified either by the vector xshift,tshift or by the values in the datafiles xfile,tfile. For fractional shift, the shifted data is interpolated.
2. Then a mix (weighted moving average) is performed over the shifted panel to emphasize events with the specified moveout and suppress events with other moveouts.
3. The panel is then shifted back (and interpolated) to its original moveout, and subtracted from the original data. Thus all events with the user-specified moveout are removed.

For VSP data the following modifications are provided:
1. The moveout polygon in x,t is usually the first break times for each trace. The parameter sign allows for downward shift in order align upgoing events.
2. Alternative to a mix, a median filter can be applied by setting the parameter median=1 and nmed= to the number of traces filtered.
3. By setting subtract=0 the filtered panel is only shifted back but not subtracted from the original data.

The values of tshift are linearly interpolated for traces falling between given xshift values. The tshift interpolant is extrapolated to the left by the smallest time sample on the trace and to the right by the last value given in the tshift array.

The files tfile and xfile are files of binary (C-style) floats.

The number of values defined by mix=val1,val2,... determines the number of traces to be averaged, the values determine the weights.

Caveat:
The median filter may perform poorly on the edges of a section. Choosing larger beginning and ending mix values may help, but may also introduce additional artifacts.

Examples:

Credits:

CWP: John Stockwell, based in part on sumute, sureduce, sumix
CENPET: Werner M. Heigl - fixed various errors, added VSP functionality

```
 U of Durham, UK: Richard Hobbs - fixed the program so it applies the
                                  median filter
 ideas for improvement:
a versatile median filter needs to do:
shift traces by fractional amounts -> needs sinc interpolation
positive and negative shifts similar to SUSTATIC
make subtraction of filtered events a user choice
provide a median stack as well as a weighted average stack
 Trace header fields accessed: ns, dt, delrt, key=keyword
```

```
SUPHASE - PHASE manipulation by linear transformation

 suphase  <stdin >sdout

Required parameters:
none
Optional parameters:
a=90 constant phase shift
b=180/PI              linear phase shift
c=0.0 phase = a +b*(old_phase)+c*f;

Notes:
A program that allows the user to experiment with changes in the phase
spectrum of a signal.
```

```
 SUSMGAUSS2 --- SMOOTH a uniformly sampled 2d array of velocities
using a Gaussian filter specified with correlation
lengths a1 and a2.

 susmgauss2 < stdin [optional parameters ] > stdout

 Optional Parameters:
 a1=0 smoothing parameter in the 1 direction
 a2=0 smoothing parameter in the 2 direction

 Notes:
 Larger a1 and a2 result in a smoother velocity. The velocities are
 first transformed to slowness and then a Gaussian filter is applied
 in the wavenumber domain.

 Input file must be in SU format. The output file is smoothed velocity



Credits:
CWP: Carlos Pacheco, 2005
```

```
SUTVBAND - time-variant bandpass filter (sine-squared taper)

sutvband <stdin >stdout tf= f=

Required parameters:
      dt = (from header)      time sampling interval (sec)
      tf=             times for which f-vector is specified
      f=f1,f2,f3,f4   Corner frequencies corresponding to the
                      times in tf. Specify as many f= as
                      there are entries in tf.

The filters are applied in frequency domain.

Example:
sutvband <data tf=.2,1.5 f=10,12.5,40,50 f=10,12.5,30,40 | ...

Credits:
     CWP: Jack, Ken

Trace header fields accessed:  ns, dt, delrt
```

```
 BHEDTOPAR - convert a Binary tape HEaDer file to PAR file format

     bhedtopar < stdin outpar=parfile

 Required parameter:
  none
 Optional parameters:
swap=0  =1 to swap bytes
  outpar=/dev/tty =parfile  name of output param file

 Notes:
 This program dumps the contents of a SEGY binary tape header file, as
 would be produced by segyread and segyhdrs to a file in "parfile" format.
 A "parfile" is an ASCII file containing entries of the form param=value.
 Here "param" is the keyword for the binary tape header field and
 "value" is the value of that field. The parfile may be edited as
 any ASCII file. The edited parfile may then be made into a new binary tape
 header file via the program    setbhed.

 See    sudoc  setbhed   for examples


 Credits:

CWP: John Stockwell  11 Nov 1994
```

```
 SU3DCHART - plot x-midpoints vs. y-midpoints for 3-D data

 su3dchart <stdin >stdout

 Optional parameters:
 outpar=null name of parameter file
 degree=0 =1 convert seconds of arc to degrees

 The output is the (x, y) pairs of binary floats

 Example:
 su3dchart <segy_data outpar=pfile >plot_data
 psgraph <plot_data par=pfile \\
linewidth=0 marksize=2 mark=8 | ...
 rm plot_data

 su3dchart <segy_data | psgraph n=1024 d1=.004 \\
linewidth=0 marksize=2 mark=8 | ...

 Note:  sx, etc., are declared double because float has only 7
 significant numbers, that's not enough, for example,
 when tr.scalco=100 and coordinates are in second of arc
 and located near 30 degree latitude and 59 degree longitude



 Credits:
CWP: Shuki Ronen
Toralf Foerster

 Trace header fields accessed: sx, sy, gx, gy, counit, scalco.
```

```
 SUABSHW - replace header key word by its absolute value

 suabshw <stdin >stdout key=offset

 Required parameters:
  none

 Optional parameter:
  key=offset header key word


 Credits:
CWP: Jack K. Cohen
```

```
 SUADDHEAD - put headers on bare traces and set the tracl and ns fields

 suaddhead <stdin >stdout ns= ftn=0

 Required parameter:
  ns=the number of samples per trace

 Optional parameter:
ifdef SU_LINE_HEADER
head=           file to read headers in
                       not supplied --  will generate headers
                       given        --  will read in headers and attach
                                        floating point arrays to form
                                        traces  ",
                       (head can be created via sustrip program)
endif
  ftn=0 Fortran flag
  0 = data written unformatted from C
  1 = data written unformatted from Fortran
       tsort=3         trace sorting code:
                                1 = as recorded (no sorting)
                                2 = CDP ensemble
                                3 = single fold continuous profile
                                4 = horizontally stacked ",
       ntrpr=1         number of data traces per record
                       if tsort=2, this is the number of traces per cdp",

 Trace header fields set: ns, tracl
 Use sushw/suchw to set other needed fields.

 Caution: An incorrect ns field will munge subsequent processing.
 Note:    n1 and nt are acceptable aliases for ns.

 Example:
 suaddhead ns=1024 <bare_traces | sushw key=dt a=4000 >segy_traces

 This command line adds headers with ns=1024 samples.  The second part
 of the pipe sets the trace header field dt to 4 ms. See also the
 selfdocs of related programs  sustrip and supaste.
 See:   sudoc supaste
 Related Programs:  supaste, sustrip
```

```
 SUAHW - Assign Header Word using another header word

  suahw <stdin >stdout [optional parameters]

 Required parameters:
  key1=ep output key
  key2=fldr input key
  a= array of key1 output values
  b= array of key2 input values
  mode=extrapolate how to assign a key1-value, when the key2-value
is not found in b:
=interpolate interpolate
=extrapolate interpolate and extrapolate
=zero zero key1-values
=preserve preserve key1-values
=transfer transfer key2-values to key1

 Optional parameters:
  key3=tracf input key
  c= array of key3 input values

 The key1-value is assigned based on the key2-value and the arrays a,b.
 If the header value of key2= equals the n'th element in b=, then the
 header value key1= is set to the n'th element in a=.
 The arrays a= and b= must have the same size, and the elements of b=
 must be in ascending order.

 The mode-switch decides what to do when a trace header has a key2-value
 that is not an element of the b-array:
    zero - the key1-value will be set to zero
    preserve - the key1-value will not be modified
    transfer - the key2-value will be assigned to key1
    interpolate - if the key2-value is greater than the n'th element
and less than the (n+1)'th element of b=, then the key1-value
will be interpolated accordingly from the n'th and (n+1)'th
element of a=. Otherwise, key1 will not be changed.
    extrapolate - same as interpolate, plus, if the key2-value is
smaller/greater than the first/last element of b=, then the
key1-value will be set to the first/last element of a=

 The array c= can be used to prevent the modification of trace headers
 with certain key3-values. The number of elements in c= is independent
 of the other arrays.
```

```
 The key1-value will not be modified, if the mode-switch is set to
    zero, preserve, transfer - and the key3-value is an element of c=
    interpolate, extrapolate - and the key3-value is outside of c=
(smaller than the first or greater than
the last element of c=)

 Examples:
  Assign shot numbers 1-3 to field file ID 1009,1011,1015 and 0 to the
  remaining FFID (fldr):
    suahw <data a=1,2,3 b=1009,1011,1015 mode=zero

  Use channel numbers (tracf) to assign stations numbers (tracr) for a
  split spread with a gap:
    suahw <data key1=tracr a=151,128,124,101 key2=tracf b=1,24,25,48

  Assign shot-statics:
    suahw <data key1=sstat key2=ep a=-32,13,-4 b=1,2,3

  Set trid to 0 for channel 1-24, but only for the record 1016:
    suahw <data key1=trid key2=tracf key3=fldr a=0,0 b=1,24 c=1016

 Credits:
Florian Bleibinhaus, U Salzburg, Austria
cloned from suchw of Einar Kajartansson, SEP
```

```
SUAZIMUTH - compute trace AZIMUTH, offset, and midpoint coordinates
            and set user-specified header fields to these values

 suazimuth <stdin >stdout [optional parameters]

Required parameters:
    none

Optional parameters:
  key=otrav      header field to store computed azimuths in
  scale=1.0      value(key) = scale * azimuth
  az=0           azimuth convention flag
                  0: 0-179.999 deg, reciprocity assumed
                  1: 0-359.999 deg, points from receiver to source
                 -1: 0-359.999 deg, points from source to receiver
  sector=1.0     if set, defines output in sectors of size
                 sector=degrees_per_sector, the default mode is
                 the full range of angles specified by az

  offset=0       if offset=1 then set offset header field
  signedflag=0   is offset signed? if =1 signedflag=1
  offkey=offset  header field to store computed offsets in
  offkey=offset  header field to store computed offsets in

  cmp=0          if cmp=1, then compute midpoint coordinates and
                 set header fields for (cmpx, cmpy)
  mxkey=ep       header field to store computed cmpx in
  mykey=cdp      header field to store computed cmpy in

Notes:
  All values are computed from the values in the coordinate fields
  sx,sy (source) and gx,gy (receiver).
  The output field "otrav" for the azimuth was chosen arbitrarily as
  an example of a little-used header field, however, the user may
  choose any field that is convenient for his or her application.

  Setting the sector=number_of_degrees_per_sector sets key field to
  sector number rather than an angle in degrees.

  For az=0, azimuths are measured from the North, however, reciprocity
  is assumed, so azimuths go from 0 to 179.9999 degrees. If sector
  option is set, then the range is from 0 to 180/sector.
```

```
  For az=1, azimuths are measured from the North, with the assumption
  that the direction vector points from the receiver to the source.
  For az=-1, the direction vector points from the source to the
  receiver. No reciprocity is assumed in these cases, so the angles go
  from 0 to 359.999 degrees.
  If the sector option is set, then the range is from 0 to 360/sector.

Caveat:
  This program honors the value of scalco in scaling the values of
  sx,sy,gx,gy. Type "sukeyword scalco" for more information.

  Type "sukeyword -o" to see the keywords and descriptions of all
  header fields.

  To plot midpoints, use: su3dchart


Credits:
 based on suchw, su3dchart
     CWP: John Stockwell and  UTulsa: Chris Liner, Oct. 1998
     UTulsa: Chris Liner added offset option, Feb. 2002
        cll: fixed offset option and added cmp option, May 2003
     RISSC: Nils Maercklin added key options for offset and
            midpoints, and added azimuth direction option, Sep. 2006

 Algorithms:
     offset = osign * sqrt( (gx-sx)*(gx-sx) + (gy-sy)*(gy-sy) )
              with osign = sgn( min((sx-gx),(sy-gy)) )

     midpoint x  value  xm = (sx + gx)/2
     midpoint y  value  ym = (sy + gy)/2

 Azimuth will be defined as the angle, measured in degrees,
 turned from North, of a vector pointing to the source from the midpoint,
 or from the midpoint to the source. Azimuths go from 0-179.000 degrees
 or from 0-180.0 degrees.

 value(key) = scale*[90.0 - (180.0/PI)*(atan((sy - ym)/(sx - xm))) ]
     or
 value(key) = scale*[180.0 - (180.0/PI)*(atan2((ym - sy),(xm - sx)) ]

 Trace header fields accessed: sx, sy, gx, gy, scalco.
 Trace header fields modified: (user-specified keys)
```

```
SUCDPBIN - Compute CDP bin number

sucdpbin <stdin >stdout xline= yline= dcdp=

Required parameters:
xline= array of X defining the CDP line
yline= array of Y defining the CDP line
dcdp= distance between bin centers

Optional parameters
verbose=0 <>0 output informations
cdpmin=1001 min cdp bin number
distmax=dcdp search radius

xline,yline defines the CDP line made of continuous straight lines.
If a smoother line is required, use unisam to interpolate.
Bin centers are located at dcdp constant interval on this line.
Each trace will be numbered with the number of the closest bin. If no
bin center is found within the search radius. cdp is set to 0


Credits:
2009 Dominique Rousset - Mohamed Hamza
     Universit de Pau et des Pays de l'Adour (France)
```

```
 SUCHART - prepare data for x vs. y plot

 suchart <stdin >stdout key1=sx key2=gx

 Required parameters:
  none

 Optional parameters:
  key1=sx   abscissa
  key2=gx ordinate
outpar=null name of parameter file

 The output is the (x, y) pairs of binary floats

 Examples:
 suchart < sudata outpar=pfile >plot_data
 psgraph <plot_data par=pfile title="CMG" \\
linewidth=0 marksize=2 mark=8 | ...
 rm plot_data

 suchart < sudata | psgraph n=1024 d1=.004 \\
linewidth=0 marksize=2 mark=8 | ...

 fold chart:
 suchart < stacked_data key1=cdp key2=nhs |
            psgraph n=NUMBER_OF_TRACES d1=.004 \\
linewidth=0 marksize=2 mark=8 > chart.ps



 Credits:
SEP: Einar Kjartansson
CWP: Jack K. Cohen

 Notes:
The vtof routine from valpkge converts values to floats.
```

```
 SUCHW - Change Header Word using one or two header word fields

  suchw <stdin >stdout [optional parameters]

 Required parameters:
 none

 Optional parameters:
 key1=cdp,... output key(s)
 key2=cdp,... input key(s)
 key3=cdp,... input key(s)
 a=0,... overall shift(s)
 b=1,... scale(s) on first input key(s)
 c=0,... scale on second input key(s)
 d=1,... overall scale(s)
 e=1,... exponent on first input key(s)
 f=1,... exponent on second input key(s)

 The value of header word key1 is computed from the values of
 key2 and key3 by:

val(key1) = (a + b * val(key2)^e + c * val(key3)^f) / d

 Examples:
 Shift cdp numbers by -1:
suchw <data >outdata a=-1

 Add 1000 to tracr value:
  suchw key1=tracr key2=tracr a=1000 <infile >outfile

 We set the receiver point (gx) field by summing the offset
 and shot point (sx) fields and then we set the cdp field by
 averaging the sx and gx fields (we choose to use the actual
 locations for the cdp fields instead of the conventional
 1, 2, 3, ... enumeration):

   suchw <indata key1=gx key2=offset key3=sx b=1 c=1 |
   suchw key1=cdp key2=gx key3=sx b=1 c=1 d=2 >outdata

 Do both operations in one call:

 suchw<indata key1=gx,cdp key2=offset,gx key3=sx,sx b=1,1 c=1,1 d=1,2 >outdata
```

```
 Credits:
SEP: Einar Kjartansson
CWP: Jack K. Cohen
      CWP: John Stockwell, 7 July 1995, added array of keys feature
      Delphi: Alexander Koek, 6 November 1995, changed calculation so
              headers of different types can be expressed in each other
```

```
 SUCLIPHEAD - Clip header values

 sucliphead <stdin >stdout [optional parameters]

 Required parameters:
none

 Optional parameters:
key=cdp,... header key word(s) to clip
min=0,... minimum value to clip
max=ULONG_MAX,ULONG_MAX,... maximum value to clip



 Credits:
Geocon: Garry Perratt
```

```
SUCOUNTKEY - COUNT the number of unique values for a given KEYword.

sucountkey < input.su key=[sx,gx,cdp,...]
Required parameter:
key= array of SU header keywords being counted
Optional parameters:
verbose=1 chatty, =0 just print keyword number
Example:
 suplane | sucountkey key=tracl,tracr,offset



Credits: Baoniu Han, bhan@mines.edu, Nov, 2000
```

```
SUDUMPTRACE - print selected header values and data.
              Print first num traces.
              Use SUWIND to skip traces.

sudumptrace < stdin [> ascii_file]

Optional parameters:
    num=4                     number of traces to dump
    key=key1,key2,...         key(s) to print above trace values
    hpf=0                     header print format is float
                              =1 print format is exponential

Examples:
  sudumptrace < inseis.su            PRINTS: 4 traces, no headers
  sudumptrace < inseis.su key=tracf,offset
  sudumptrace < inseis.su num=7 key=tracf,offset > info.txt
  sudumptrace < inseis.su num=7 key=tracf,offset hpf=1 > info.txt

Related programs: suascii, sugethw


Credits:
  MTU: David Forel, Jan 2005

Trace header field accessed: nt, dt, delrt
```

```
SUEDIT - examine segy diskfiles and edit headers

suedit diskfile  (open for possible header modification if writable)
suedit <diskfile  (open read only)

The following commands are recognized:
number read in that trace and print nonzero header words
<CR>go to trace one step away (step is initially -1)
+ read in next trace (step is set to +1)
-read in previous trace (step is set to -1)
dN advance N traces (step is set to N)
% print some percentiles of the trace data
r print some ranks (rank[j] = jth smallest datum)
p [n1 [n2]]   tab plot sample n1 to n2 on current trace
g [tr1 tr2]   ximage plot the trace [traces tr1 to tr2]
w [tr1 tr2]   xwigb plot the trace [traces tr1 to tr2]
f [tr1 tr2]   ximage plot the amplitude spectra of the trace
u [tr1 tr2]   apply user pipeline to specified traces
! key=val   change a value in a field (e.g. ! tracr=101)
? print help file
q quit

NOTE: sample numbers are 1-based (first sample is 1).

'u 1000000  1000100 suwind >subset.su' will quickly extract a few
traces from the middle of a large dataset


Credits:
SEP: Einar Kjartansson, Shuki Ronen, Stew Levin
CWP: Jack K. Cohen
Unocal: Reg Beardsley
Trace header fields accessed: ns
Trace header fields modified: ntr (only for internal plotting)
```

```
SUGEOM - Fill up geometry in trace headers.

sugeom rps= sps= xps=  <stdin >stdout [optional parameters]

Required parameters:
      rps=  filename for the receiver points definitions. (AKA R file)
      sps=  filename for the source   points definitions. (AKA S file)
      xps=  filename for the relation specification.      (AKA X file)

OBS:  if 'prefix' (see bellow) is given the last three parameters become
      optional, but any one can be used as an override for the name
      built using prefix.
         Ex: sugeom prefix=blah  xps=bleh.xxx
             Used filenames:  blah.s blah.r and bleh.xxx

Optional parameters:
      prefix=  prefix name for the SPS files.  If given the filename will
               be constructed as '<prefix>.s', '<prefix>.r' and
               '<prefix>.x'.
      rev=0             for SPS revision 0 format.
         =2 (Default)   for SPS revision 2.1 format.
  verbose=0 (Default)   silent run.
         =1             verbose stats.
    ibin=0  (default)   inline binsize (cdp interval)
                        0 -> do not compute cdp number

 For SPS format specification consult:
   http://www.seg.org/resources/publications/misc/technical-standards


 WARNING: This is not a full fledged geometry program.

 The SPS format is fully described in two documents available at the
 above SEG address.

 To make short what can be a loooong discussion, the SPS file format is
 intended to describe completely a land (or tranzition zone) survey. The
 main difference between land and marine survey is that in land all
 points are meant to be previously located and remain fixed along
 acquisition operation.  In a marine survey the cables and boat can be
 dragged by current and wind, making the prosition of source and receivers
 unpredictable.  The planning and survey description of a marine program
 must take into account all moving points, being necessary a full
```

positioning description of source and receivers at every single shot.

The SPS format standard is composed of three text files with 80 columns each. Two of those files are Point Record Specification and are used to describe all survey points, that is the receiver stations and source points. The remaining file is the Relation Record Specification and is used to describe how each source point is related to a set of corresponding receiver points during the registration of each record (fldr).

These files usually have a number of lines at the start describing the survey and the projections used for coordinates. This program just skip them. Those are header entries and have an 'H' in first column.

Each line (entry) of the Point Record Specification contain the information of a single point. The only difference of a source point specification entry and a receiver point specification entry is the first column of the file, it has an 'S' for a source point description and an 'R' for a receiver point description. For each point entry there is an identification with a pair of informations, the Line it belongs and the Point Number, a set of coordinates (X, Y, surface Elevation, depth of element), and the static correction. All source points description are in a single file (known as S-File), and all receiver station informations are in another file (R-File).

The Relation Record Specification (X-File) is a file with as many entries (text lines) as necessary to completely describe each record (fldr) acquired. Each entry containing a record information starts with an 'X' in first column. The informations in the entry starts with the tape number in which the register was recorded (not used for this program) and the Field Record Number (fldr). Next comes the source point description (the same Line and Point(ep) from the S-File). Then a sequence of channels (tracf) numbers (just first and last) and a channel increment. Next comes the receiver description using the same Line and Point/station from the R-File). The Receiver Points are specified as the first and last station used at that line. Finally comes the recording time. If the spread has a gap, very common for older lines, it will require at least two entries (text lines) to describe this record, one describing the channels and receiver stations before the gap, and another for the channels and stations after the gap. The initial informations (tape, record, and source) are repeated for all entries. In a 3D there is one entry for each line of the patch.

To use this program it is necessary to use all three SPS files. The S-File and the R-File can describe more points than will be used for the processing. For example, if one is processing just a section of a 2D line the Point Record Specification files can describe all points of the complete line, the information in excess will be disregarded.

The X-File must have the information in the same order of the input SU file, if not so all files (fldr) that do not match the order will be skipped until the program find the fldr corresponding to the sequence of the X-File.

Although this program read all of the SPS files, it is not ready for 3D geometry processing. It is just a basic 2D straight line geometry processing program that (hopefully) will fill correctly the informations in header. It can be used for a crooked line for the coordinates (X, Y Elevation, and element depth), the static correction, and offset. The cdp numbering will need an specialized program for computation.

All coordinates are expected to be an UTM projection in meters or feet. ",

The X file order must be the same as the input file. Upon reading a trace whose trcf is not the next record to be processed in X file, this record, and the following, will be skipped until a match is found for the current entry in X file. This is a way to, e.g., skip a noisy record, just remove its entry from X file.

If a trace (understand fldr/tracf pair) is not represented in the X file it will be skipped until a trace in current fldr matches the current set of X entry for that fldr. If it is desirable to process just a subset of channels, just keep in the X-File information about those channels.

Updated trace header positions:

tracl - it keeps counting to overcome possible trace skipping.
ep - If zeroed out in header it uses the relation (X) file info.

sx, sy, gx, gy - will be updated with the coordinates in point files
R and S. The scalco value used is the already
stored in header, almost ever equal zero.
gelev, selev, sdepth - will be updated with values in point files
R and S. The scalel value used is the already
stored in header, almost ever equal zero.

```
 sstat, gstatt - these values are filled with the information of the
          Point Record Specification files.  If not available will be zero.
 offset - is computed from source and receiver coordinates.  The offset
          signal is made negative if source station number is greater
          than the receiver station number.  It consider that source and
          receiver numbering are the same.   Fractionary stations, e.g.
          source point halfway between stations are ok.
 cdp    - if parameter 'ibin' is passed it is considered the cdp spacing
          and the cdp number will be computed as follow:
          The cdp position is computed as the midpoint between source
          and receiver.  The distance from this point to the first
          station of R point file is divided by 'ibin' and added to 100.
          This 100 is absolutely artitrary.  ;)


Credits:
Fernando Roxo <petro@roxo.org>

Trace header fields accessed:
fldr, tracf, scalco, scalel, ep, sx, sy, gx, gy, gelev, selev, sdepth, cdp
```

```
 SUGETHW - sugethw writes the values of the selected key words

   sugethw key=key1,... [output=] <infile [>outfile]

 Required parameters:
 key=key1,... At least one key word.

 Optional parameters:
 output=ascii output written as ascii for display
  =binary for output as binary floats
  =geom   ascii output for geometry setting
 verbose=0  quiet
  =1 chatty

 Output is written in the order of the keys on the command
 line for each trace in the data set.

 Example:
  sugethw <stdin key=sx,gx
 writes sx, gx values as ascii trace by trace to the terminal.

 Comment:
 Users wishing to edit one or more header field (as in geometry setting)
 may do this via the following sequence:
     sugethw < sudata output=geom key=key1,key2,... > hdrfile
 Now edit the ASCII file hdrfile with any editor, setting the fields
 appropriately. Convert hdrfile to a binary format via:
     a2b < hdrfile n1=nfields > binary_file
 Then set the header fields via:
     sushw < sudata infile=binary_file key=key1,key2,... > sudata.edited


 Credits:

SEP: Shuki Ronen
CWP: Jack K. Cohen
      CWP: John Stockwell, added geom stuff, and getparstringarray
```

```
 SUHTMATH - do unary arithmetic operation on segy traces with
     headers values

 suhtmath <stdin >stdout

 Required parameters:
none

 Optional parameter:
key=tracl header word to use
op=nop operation flag
nop   : no operation
add   : add header to trace
mult  : multiply trace with header
div   : divide trace by header

scale=1.0 scalar multiplier for header value
const=0.0 additive constant for header value

 Operation order: ",

 op=add: out(t) = in(t) + (scale * key + const)
 op=mult: out(t) = in(t) * (scale * key + const)
 op=div: out(t) = in(t) / (scale * key + const)

 Credits:
Matthias Imhof, Virginia Tech, Fri Dec 27 09:17:29 EST 2002
```

```
SUKEYCOUNT - sukeycount writes a count of a selected key

  sukeycount key=keyword < infile [> outfile]

Required parameters:
key=keyword      One key word.

Optional parameters:
verbose=0  quiet
       =1  chatty

Writes the key and the count to the terminal or a text
  file when a change of key occurs. This does not provide
  a unique key count (see SUCOUNTKEY for that).
Note that for key values  1 2 3 4 2 5
  value 2 is counted once per occurrence since this program
  only recognizes a change of key, not total occurrence.

Examples:
   sukeycount < stdin key=fldr
   sukeycount < stdin key=fldr > out.txt


Credits:

  MTU: David Forel, Jan 2005
```

```
SULCTHW - Linear Coordinate Transformation of Header Words

  sulcthw <infile >outfile

xt=0.0 Translation of X
yt=0.0 Translation of Y
zt=0.0 Translation of Z
xr=0.0 Rotation around X in degrees
yr=0.0 Rotation aroun Y  in degrees
zr=0.0 Rotation around Z in degrees

Notes:
Translation:
x = x'+ xt;y = y'+ yt;z = z' + zt;

Rotations:
Around Z axis
X = x*cos(zr)+y*sin(zr);
Y = y*cos(zr)-x*sin(zr);
Around Y axis
Z = z*cos(yr)+x*sin(yr);
X = x*cos(yr)-z*sin(yr);
Around X axis
Y = y*cos(xr)+z*sin(xr);
Z = Z*cos(xr)-y*sin(xr);

Header words triplets that are transformed
sx,sy,selev
gx,gy,gelev

The header words restored as 32 bit integers using SEG-Y
convention (with coordinate scalers scalco and scalel).

After transformation they are converted back to integers and stored.


 Credits: Potash Corporation of Saskatchewan: Balasz Nemeth   c. 2008
```

```
SULHEAD - Load information from an ascii column file into HEADERS based
   on the value of the user specified header field
  sulhead < inflie > outfile cf=Column_file key=..  [ optional parameters]

 Required parameters:
 cf=Name of column file
 key=key1,key2,...Number of column entires
 Optional parameters:
 mc=1 Column number to use to match rows to traces

Notes:
 Caveat: This is not simple trace header setting, but conditional
 setting.

 This utility reads the column file and loads the values into the
 specified header locations. Each column represents one set of header
 words, one of them (#mc) is used to match the rows to the traces
 using header tr.key[mc].

 Example:
 key=cdp,ep,sx   mc=1 cf=file
 file contains:
1  2  3
2  3  4

 if tr.cdp = 1 then tr.ep and tr.sx will be set to 2 and 3
 if tr.cdp = 2 then tr.ep and tr.sx will be set to 3 and 4
 if tr.cdp=other than tr.trid=3

 Caveat: the user has to make it sure that number of entires in key=
 is equal the number of columns stored in the file.

 For simple mass setting of header words, see selfdoc of:  sushw



 Credits: Balasz Nemeth, Potash Corporation, Saskatoon Saskatchewan
 Given to CWP in 2008
```

```
 SUPASTE - paste existing SU headers on existing binary data

 supaste <bare_data >segys  ns= head=headers ftn=0

 Required parameter:
ns=the number of samples per trace

 Optional parameters:
  head=headers file with segy headers
ftn=0 Fortran flag
0 = unformatted data from C
1 = ... from Fortran
verbose=0 1= echo number of traces pasted
 Caution:
An incorrect ns field will munge subsequent processing.

 Notes:
 This program is used when the option head=headers is used in
 sustrip. See:   sudoc sustrip    for more details.

 Related programs:  sustrip, suaddhead

 Credits:
CWP:  Jack K. Cohen, November 1990
```

```
surandhw - set header word to random variable

surandhw <stdin >stdout key=tstat a=0 min=0 max=1

Required parameters:
 none (no op)

Optional parameters:
 key=tstat header key word to set
 a=0 =1 flag to add original value to final key
 noise=gauss noise probability distribution
 =flat for uniform; default Gaussian
 seed=from_clock random number seed (integer)
 min=0 minimum random number
 max=1 maximum radnom number

NOTES:
The value of header word key is computed using the formula:
 val(key) = a * val(key) + rand

Example:
  surandhw <indata key=tstat a=0 min=0 max=10  > outdata
```

```
 SURANGE - get max and min values for non-zero header entries

 surange <stdin

 Optional parameters:
key= Header key(s) to range (default=all)

 Note: Gives partial results if interrupted

 Output is:
 number of traces
 keyword min max (first - last)
 north-south-east-west limits of shot, receiver and midpoint


 Credits:
      Stanford: Stewart A. Levin
              Added print of eastmost, northmost, westmost,
              southmost coordinates of shots, receivers, and
              midpoints.  These coordinates have had any
              nonzero coscal header value applied.
Geocon: Garry Perratt (output one header per line;
option to specify headers to range;
added first & last values where min<max)
Based upon original by:
SEP: Stew Levin
CWP: Jack K. Cohen

 Note: the use of "signal" is inherited from BSD days and may
       break on some UNIXs.  It is dicey in that the responsibility
 for program termination is lateraled back to the main.
```

```
 SUSEHW - Set the value the Header Word denoting trace number within
     an Ensemble defined by the value of another header word

     susehw <stdin >stdout [options]

 Required Parameters:
none

 Optional Parameters:
 key1=cdp Key header word defining the ensemble
 key2=cdpt Key header word defining the count within the ensemble
 a=1 starting value of the count in the ensemble
 b=1 increment or decrement within the ensemble

 Notes:
 This code was written because suresstat requires cdpt to be set.
 The computation is
  val(key2) = a + b*i

 The input data must first be sorted into constant key1 gathers.
 Example: setting the cdpt field ",
        susetehw < cdpgathers.su a=1 b=1 key1=cdp key2=cdpt > new.su


 Credits:
  CWP: John Stockwell (Feb 2008) in answer to a question by Warren Franz
        based on various codes, including susplit, susshw, suchw
```

```
SUSHW - Set one or more Header Words using trace number, mod and
integer divide to compute the header word values or input
the header word values from a file

... compute header fields
  sushw <stdin >stdout key=cdp,.. a=0,..  b=0,.. c=0,.. d=0,.. j=..,..

... or read headers from a binary file
  sushw <stdin > stdout  key=key1,..    infile=binary_file


Required Parameters for setting headers from infile:
key=key1,key2 ... is the list of header fields as they appear in infile
infile=  binary file of values for field specified by
 key1,key2,...

Optional parameters ():
key=cdp,... header key word(s) to set
a=0,... value(s) on first trace
b=0,... increment(s) within group
c=0,... group increment(s)
d=0,... trace number shift(s)
j=ULONG_MAX,ULONG_MAX,... number of elements in group

Notes:
Fields that are getparred must have the same number of entries as key
words being set. Any field that is not getparred is set to the default
value(s) above. Explicitly setting j=0 will set j to ULONG_MAX.

The value of each header word key is computed using the formula:
 i = itr + d
 val(key) = a + b * (i % j) + c * (int(i / j))
where itr is the trace number (first trace has itr=0, NOT 1)

Examples:
1. set every dt field to 4ms
 sushw <indata key=dt a=4000 |...
2. set the sx field of the first 32 traces to 6400, the second 32 traces
   to 6300, decrementing by -100 for each 32 trace groups
  ...| sushw key=sx a=6400 c=-100 j=32 |...
3. set the offset fields of each group of 32 traces to 200,400,...,6400
  ...| sushw key=offset a=200 b=200 j=32 |...
4. perform operations 1., 2., and 3. in one call
```

```
 ..| sushw key=dt,sx,offset a=4000,6400,200 b=0,0,200 c=0,-100,0 j=0,32,32 |

In this example, we set every dt field to 4ms.  Then we set the first
32 shotpoint fields to 6400, the second 32 shotpoint fields to 6300 and
so forth.  Next we set each group of 32 offset fields to 200, 400, ...,
6400.

Example of a typical processing sequence using suchw:
 sushw <indata key=dt a=4000 |
 sushw key=sx a=6400 c=-100 j=32 |
 sushw key=offset a=200 b=200 j=32 |
 suchw key1=gx key2=offset key3=sx b=1 c=1 |
 suchw key1=cdp key2=gx key3=sx b=1 c=1 d=2 >outdata

Again, it is possible to eliminate the multiple calls to both sushw and
sushw, as in Example 4.

Reading header values from a binary file:
If the parameter infile=binary_file is set, then the values that are to
be set for the fields specified by key=key1,key2,... are read from that
file. The values are read sequentially from the file and assigned trace
by trace to the input SU data. The infile consists of C (unformated)
binary floats in the form of an array of size (nkeys)*(ntraces) where
nkeys is the number of floats in the first (fast) dimension and ntraces
is the number of traces.

Comment:
Users wishing to edit one or more header fields (as in geometry setting)
may do this via the following sequence:
    sugethw < sudata output=geom  key=key1,key2 ... > hdrfile
Now edit the ASCII file hdrfile with any editor, setting the fields
appropriately. Convert hdrfile to a binary format via:
    a2b < hdrfile n1=nfields > binary_file
Then set the header fields via:
    sushw < sudata infile=binary_file key=key1,key2,... > sudata.edited

Caveat:
If the (number of traces)*(number of key words) exceeds the number of
values in the infile then the user may still set a single header field
on the remaining traces via the parameters key=keyword a,b,c,d,j.

Example:
   sushw < sudata=key1,key2 ... infile=binary_file [Optional Parameters]
```

```
 Credits:
SEP: Einar Kajartansson
CWP: Jack K. Cohen
      CWP: John Stockwell, added multiple fields and infile= options

 Caveat:
All constants are cast to doubles.
```

```
 SUSTRIP - remove the SEGY headers from the traces

 sustrip <stdin >stdout head=/dev/null outpar=/dev/tty ftn=0

 Required parameters:
  none

 Optional parameters:
  head=/dev/null file to save headers in

  outpar=/dev/tty output parameter file, contains:
  number of samples (n1=)
  number of traces (n2=)
  sample rate in seconds (d1=)

  ftn=0 Fortran flag
  0 = write unformatted for C
  1 = ... for Fortran

 Notes:
 Invoking head=filename will write the trace headers into filename.
 You may paste the headers back onto the traces with supaste
 See:  sudoc  supaste   for more information
 Related programs: supaste, suaddhead

 Credits:
SEP: Einar Kjartansson  c. 1985
CWP: Jack K. Cohen        April 1990

 Trace header fields accessed: ns, dt
```

```
SUTRCOUNT - SU program to count the TRaces in infile

  sutrcount < infile
Required parameters:
      none
Optional parameter:
   outpar=stdout
Notes:
Once you have the value of ntr, you may set the ntr header field
via:
      sushw key=ntr a=NTR < datain.su  > dataout.su
Where NTR is the value of the count obtained with sutrcount


Credits:  B.Nemeth, Potash Corporation, Saskatchewan
 given to CWP in 2008 with permission of Potash Corporation
```

```
SUUTM - UTM projection of longitude and latitude in SU trace headers

suutm <stdin >stdout [optional parameters]

Optional parameters:
   counit=(from header) input coordinate units code:
                  =1: length (meters or feet; no UTM projection)
                  =2: seconds of arc
                  =3: decimal degrees
                  =4: degrees, minutes, seconds
   idx=23         reference ellipsoid index (default is WGS 1984)
   a=(from idx)   user-specified semimajor axis of ellipsoid
   f=(from idx)   user-specified flattening of ellipsoid
   zkey=          if set, header key to store UTM zone number
   verbose=0      =1: echo ellipsoid parameters

   lon0=          central meridian for TM projection in degrees
                  (default uses the 60 standard UTM longitude zones)
   xoff=500000    false Easting (default: UTM)
   ysoff=10000000 false Northing, southern hemisphere (default: UTM)
   ynoff=0        false Northing, northern hemisphere (default: UTM)

Notes:
   Universal Transverse Mercator (UTM) coordinates are defined between
   latitudes 80S (-80) and 84N (84). Longitude values must be between
   -180 degrees (west) and 179.999... degrees (east).

   Latitudes are read from sy and gy (N positive), and longitudes
   are read from sx and gx (E positive).
   The UTM zone is determined from the receiver coordinates gy and gx.

   Use suazimuth to calculate shot-receiver azimuths and offsets.

Reference ellipsoids:
   An ellipsoid may be specified by its semimajor axis a and its
   flattening f, or one of the following ellipsoids may be selected
   by its index idx (semimajor axes in meters):
    0  Sphere with radius of 6371000 m
    1  Airy 1830
    2  Australian National 1965
    3  Bessel 1841 (Ethiopia, Indonesia, Japan, Korea)
    4  Bessel 1841 (Namibia)
    5  Clarke 1866
```

```
 6  Clarke 1880
 7  Everest (Brunei, E. Malaysia)
 8  Everest (India 1830)
 9  Everest (India 1956)
10  Everest (Pakistan)
11  Everest (W. Malaysia, Singapore 1948)
12  Everest (W. Malaysia 1969)
13  Geodetic Reference System 1980 (GRS 1980)
14  Helmert 1906
15  Hough 1960
16  Indonesian 1974
17  International 1924 / Hayford 1909
18  Krassovsky 1940
19  Modified Airy
20  Modified Fischer 1960
21  South American 1969
22  World Geodetic System 1972 (WGS 1972)
23  World Geodetic System 1984 (WGS 1984) / NAD 1983


UTM grid:
The Universal Transverse Mercator (UTM) system is a world wide
coordinate system defined between 80S and 84N. It divides the
Earth into 60 six-degree zones. Zone number 1 has its central
meridian at 177W (-177 degrees), and numbers increase eastward.

Within each zone, an Easting of 500,000 m is assigned to its
central meridian to avoid negative coordinates. On the northern
hemisphere, Northings start at 0 m at the equator and increase
northward. On the southern hemisphere a false Northing of
10,000,000 m is applied, i.e. Northings start at 10,000,000 m at
the equator and decrease southward.

Coordinate encoding (sx,sy,gx,gy):
   counit=1  units of length (coordinates are not converted)
   counit=2  seconds of arc
   counit=3  decimal degrees
   counit=4  degrees, minutes and seconds encoded as integer DDDMMSS
             with scalco=1 or DDDMMSS.ss with scalco=-100 (see segy.h)
Units of length are also assumed, if counit <= 0 or counit >= 5.


Author:
```

Nils Maercklin, RISSC, University of Naples, Italy, March 2007

References:

NIMA (2000). Department of Defense World Geodetic System 1984 -
its definition and relationships with local geodetic systems.
Technical Report TR8350.2. National Imagery and Mapping Agency,
Geodesy and Geophysics Department, St. Louis, MO. 3rd edition.

J. P. Snyder (1987). Map Projections - A Working Manual.
U.S. Geological Survey Professional Paper 1395, 383 pages.
U.S. Government Printing Office.

Trace header fields accessed: sx, sy, gx, gy, scalco, counit
Trace header fields modified: sx, sy, gx, gy, scalco, counit

```
 SUXEDIT - examine segy diskfiles and edit headers

 suxedit diskfile  (open for possible header modification if writable)
 suxedit <diskfile  (open read only)

 The following commands are recognized:
 number read in that trace and print nonzero header words
 <CR>go to trace one step away (step is initially -1)
 + read in next trace (step is set to +1)
 -read in previous trace (step is set to -1)
 dN advance N traces (step is set to N)
 % print some percentiles of the trace data
 r print some ranks (rank[j] = jth smallest datum)
 p [n1 [n2]]   tab plot sample n1 to n2 on current trace
 g [tr1 tr2] ["opts"]  wiggle plot (graph) the trace
[traces tr1 to tr2]
 f wiggle plot the Fourier transform of the trace
 ! key=val   change a value in a field (e.g. ! tracr=101)
 ? print help file
 q quit

 NOTE: sample numbers are 1-based (first sample is 1).


 Credits:
SEP: Einar Kjartansson, Shuki Ronen, Stew Levin
CWP: Jack K. Cohen

 Trace header fields accessed: ns
 Trace header fields modified: ntr (only for internal plotting)
```

```
 SUINTERPFOWLER - interpolate output image from constant velocity panels
   built by SUTIFOWLER or CVS

 These parameters should be specified the same as in SUTIFOWLER:
 vmin=1500. minimum velocity
 vmax=2500. maximum velocity
 nv=21 number of velocity panels
 etamin=0.10 minimum eta value
 etamax=0.25 maximum eta value
 neta=11 number of eta values
 ncdps=1130 number of cdp points

 If these parameters are specified so that nvstack>5, then the input
 data are assumed to come from CVS and the SUTIFOWLER parameters are ignored.
 nvstack=0 number of constant velocity stack panels output by CVS
 vminstack=1450 minimum velocity specified for CVS
 vscale=1.0 scale factor for velocity functions

 These parameters specify the desired output (time,velocity,eta) model
 at each cdp location. The sequential cdp numbers should be specified in
 increasing order from 0 to 'ncdps-1' at from 1 to 'ncdps' control point
 locations. (Time values are in seconds.)
 cdp=0 cdp number for (t,v,eta) triplets (specify more than
  once if needed)
 t=0. array of times for (t,v,eta) triplets (specify more
than once if needed)
 v=1500. array of velocities for (t,v,eta) triplets (specify
more than once if needed)
 eta=0. array of etas for (t,v,eta) triplets (specify more
than once if needed)

 Note: This is a simple research code based on linear interpolation.
 There are no protections against aliasing built into the code beyond
 suggesting that this program have a knowledgable user. A final version
 should do a better job taking care of endpoint conditions.



 Author: (Visitor from Mobil) John E. Anderson, Spring 1994
```

```
SUINTERP - interpolate traces using automatic event picking

          suinterp < stdin > stdout

ninterp=1    number of traces to output between each pair of input traces
nxmax=500    maximum number of input traces
freq1=4.     starting corner frequency of unaliased range
freq2=20.    ending corner frequency of unaliased range
deriv=0      =1 means take vertical derivative on pick section
             (useful if interpolating velocities instead of seismic)
linear=0     =0 means use 8 point sinc temporal interpolation
             =1 means use linear temporal interpolation
             (useful if interpolating velocities instead of seismic)
lent=5       number of time samples to smooth for dip estimate
lenx=1       number of traces to smooth for dip estimate
lagc=400     number of ms agc for dip estimate
xopt=0       0 compute spatial derivative via FFT
                (assumes input traces regularly spaced and relatively
                 noise-free)
             1 compute spatial derivative via differences
                (will work on irregulary spaced data)
iopt=0     0 = interpolate
           1 = output low-pass model: useful for QC if interpolator failing
           2 = output dip picks in units of samples/trace

verbose=0 verbose = 1 echoes information

tmpdir=   if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
        the CWP_TMPDIR environment variable is set use
        its value for the path; else use tmpfile()

Notes:
This program outputs 'ninterp' interpolated traces between each pair of
input traces.  The values for lagc, freq1, and freq2 are only used for
event tracking. The output data will be full bandwidth with no agc.  The
default parameters typically will do a satisfactory job of interpolation
for dips up to about 12 ms/trace.  Using a larger value for freq2 causes
the algorithm to do a better job on the shallow dips, but to fail on the
steep dips.  Only one dip is assumed at each time sample between each pair
of input traces.

The key assumption used here is that the low frequency data are unaliased
```

and can be used for event tracking. Those dip picks are used to interpolate
the original full-bandwidth data, giving some measure of interpolation
at higher frequencies which otherwise would be aliased. Using iopt equal
to 1 allows you to visually check whether the low-pass picking model is
aliased.

Trace headers for interpolated traces are not updated correctly.
The output header for an interpolated traces equals that for the preceding
trace in the original input data. The original input traces are passed
through this module without modification.

The place this code is most likely to fail is on the first breaks.

Example run: suplane | suinterp | suxwigb &

Credit: John Anderson (visiting scholar from Mobil) July 1994

Trace header fields accessed: ns, dt

```
 SUOCEXT - smaller Offset EXTrapolation via Offset Continuation
           method for common-offset gathers

 suocext <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= offextr= [...]

 Required Parameters:
 cdpmin= minimum cdp (integer number) for which to apply DMO
 cdpmax= maximum cdp (integer number) for which to apply DMO
 dxcdp= distance between adjacent cdp bins (m)
 noffmix= number of offsets to mix (see notes)
 offextr= offset to extrapolate

 Optional Parameters:
 tdmo=0.0 times corresponding to rms velocities in vdmo (s)
 vdmo=1500.0 rms velocities corresponding to times in tdmo (m/s)
 sdmo=1.0 DMO stretch factor; try 0.6 for typical v(z)
 fmax=0.5/dt maximum frequency in input traces (Hz)
 verbose=0 =1 for diagnostic print
 tmpdir= if non-empty, use the value as a directory path prefix
for storing temporary files; else if the CWP_TMPDIR
environment variable is set use its value for the path;
else use tmpfile()

 Notes:
 Input traces should be sorted into common-offset gathers.  One common-
 offset gather ends and another begins when the offset field of the trace
 headers changes. One common-offset gather usually is enough.

 The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
 the cdp location expressed in units of meters or feet.

 The number of offsets to mix (noffmix) should typically equal the ratio of
 the shotpoint spacing to the cdp spacing.  This choice ensures that every
 cdp will be represented in each offset mix.  Traces in each mix will
 contribute through DMO to other traces in adjacent cdps within that mix.

 The tdmo and vdmo arrays specify a velocity function of time that is
 used to implement a first-order correction for depth-variable velocity.
 The times in tdmo must be monotonically increasing.

 For each offset, the minimum time at which a non-zero sample exists is
 used to determine a mute time.  Output samples for times earlier than this",
 mute time will be zeroed.  Computation time may be significantly reduced
```

```
 if the input traces are zeroed (muted) for early times at large offsets.

 A term for better amplitude reconstruction was added to Hale's formulation.

 Credits: Carlos E. Theodoro (modification of Hale's SUDMOFK program)

 Technical Reference:
C. Theodoro & K. Larner, 1998
      Extrapolation of seismic data to small offsets (CWP-276).

Dip-Moveout Processing - SEG Course Notes
Dave Hale, 1988

Bleistein, Cohen & Jaramillo, 1997
      True amplitude transformation to zero offset of data from
      curved reflectors (CWP-262).

 Trace header fields accessed:  ns, dt, delrt, offset, cdp.
 Trace header fields modified:  offset.
```

```
SUGAZMIGQ - SU version of Jeno GAZDAG's phase-shift migration
    for zero-offset data, with attenuation Q.

sugazmig <infile >outfile vfile= [optional parameters]

Optional Parameters:
dt=from header(dt) or .004 time sampling interval
dx=from header(d2) or 1.0 midpoint sampling interval
ft=0.0 first time sample
ntau=nt(from data) number of migrated time samples
dtau=dt(from header) migrated time sampling interval
ftau=ft first migrated time sample
tmig=0.0 times corresponding to interval velocities in vmig
vmig=1500.0 interval velocities corresponding to times in tmig
vfile= name of file containing velocities
Q=1e6 quality factor
ceil=1e6 gain ceiling beyond which migration ceases

verbose=0 verbose = 1 echoes information

tmpdir=   if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
        the CWP_TMPDIR environment variable is set use
        its value for the path; else use tmpfile()

Note: ray bending effects not accounted for in this version.

The tmig and vmig arrays specify an interval velocity function of time.
Linear interpolation and constant extrapolation is used to determine
interval velocities at times not specified.  Values specified in tmig
must increase monotonically.

Alternatively, interval velocities may be stored in a binary file
containing one velocity for every time sample in the data that is to be
migrated.  If vfile is specified, then the tmig and vmig arrays are ignored.

Caveat: Adding Q is a first attempt to address GPR issues.


Credits:
 Constant Q attenuation correction by Chuck Oden 5 May 2004
 CWP John Stockwell 12 Oct 1992
 Based on a constant v version by Dave Hale.
```

```
Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: ns, dt, delrt
```

```
SUINVXZCO - Seismic INVersion of Common Offset data for a smooth
            velocity function V(X,Z) plus a slowness perturbation vp(x,z)

    suinvvxzco <infile >outfile [optional parameters]

Required Parameters:
vfile                   file containing velocity array v[nx][nz]
nx=                     number of x samples (2nd dimension) in velocity
nz=                     number of z samples (1st dimension) in velocity
nxm= number of midpoints of input traces

Optional Parameters:
dt= or from header (dt)  time sampling interval of input data
offs= or from header (offset)  source-receiver offset
dxm= or from header (d2)  sampling interval of midpoints
fxm=0 first midpoint in input trace
nxd=5 skipped number of midpoints (see note)
dx=50.0 x sampling interval of velocity
fx=0.0 first x sample of velocity
dz=50.0 z sampling interval of velocity
nxb=nx/2 band centered at midpoints (see note)
nxc=0 hozizontal range in which velocity is changed
nzc=0 vertical range in which velocity is changed
fxo=0.0 x-coordinate of first output trace
dxo=15.0 horizontal spacing of output trace
nxo=101 number of output traces  ",
fzo=0.0 z-coordinate of first point in output trace
dzo=15.0 vertical spacing of output trace
nzo=101 number of points in output trace ",
fmax=0.25/dt Maximum frequency set for operator antialiasing
ang=180 Maximum dip angle allowed in the image
ls=0 =1 for line source; =0 for point source
pert=0 =1 calculate time correction from v_p[nx][nz]
vpfile file containing slowness perturbation array v_p[nx][nz]
verbose=1               =1 to print some useful information

Notes:
Traveltime and amplitude are calculated by finite difference which
is done only in one of every NXD midpoints; in the skipped midpoint,
interpolation is used to calculate traveltime and amplitude. ",
For each midpoint, traveltime and amplitude are calculated in the
horizontal range of (xm-nxb*dx, xm+nxb*dx). Velocity is changed by
constant extropolation in two upper trianglar corners whose width is
```

nxc*dx and height is nzc*dz.

Eikonal equation will fail to solve if there is a polar turned ray.
In this case, the program shows the related geometric information.
There are three ways to remove the turned ray: smoothing velocity,
reducing nxb, and increaing nxc and nzc (if the turned ray occurs
in the shallow areas). To prevent traveltime distortion from a over
smoothed velocity, traveltime is corrected based on the slowness
perturbation.

Offsets are signed - may be positive or negative.

Author: Zhenyue Liu, 08/28/93, Colorado School of Mines

Reference:
Bleistein, N., Cohen, J. K., and Hagin, F., 1987,
Two-and-one-half dimensional Born inversion with an arbitrary reference
Geophysics Vol. 52, no.1, 26-36.

```
SUINVZCO3D - Seismic INVersion of Common Offset data with V(Z) velocity
            function in 3D

    suinvzco3d <infile >outfile [optional parameters]

Required Parameters:
vfile                   file containing velocity array v[nz]
nz=                     number of z samples (1st dimension) in velocity
nxm= number of midpoints of input traces
ny= number of lines

Optional Parameters:
dt= or from header (dt)  time sampling interval of input data
offs= or from header (offset)  source-receiver offset
dxm= or from header (d2)  sampling interval of midpoints
fxm=0                   first midpoint in input trace
nxd=5 skipped number of midpoints (see note)
dx=50.0                 x sampling interval of velocity
fx=0.0                  first x sample of velocity
dz=50.0                 z sampling interval of velocity
nxb=nx/2 band centered at midpoints (see note)
fxo=0.0                 x-coordinate of first output trace
dxo=15.0 horizontal spacing of output trace
nxo=101                 number of output traces  ",
fyo=0.0 y-coordinate of first output trace
dyo=15.0 y-coordinate spacing of output trace
nyo=101 number of output traces in y-direction
fzo=0.0                 z-coordinate of first point in output trace
dzo=15.0                vertical spacing of output trace
nzo=101                 number of points in output trace ",
fmax=0.25/dt Maximum frequency set for operator antialiasing
ang=180 Maximum dip angle allowed in the image
verbose=1               =1 to print some useful information

Notes:

This algorithm is based on formula (50) in Geophysics Vol. 51,
1552-1558, by Cohen, J., Hagin, F., and Bleistein, N.

Traveltime and amplitude are calculated by ray tracing.
Interpolation is used to calculate traveltime and amplitude. ",
For each midpoint, traveltime and amplitude are calculated in the
horizontal range of (xm-nxb*dx, xm+nxb*dx). Velocity is changed by
```

```
linear interpolation in two upper trianglar corners whose width is
nxc*dx and height is nzc*dz. ",

Eikonal equation will fail to solve if there is a polar turned ray.
In this case, the program shows the related geometric information.

Offsets are signed - may be positive or negative.  ",
```

```
SUKDMIG2D - Kirchhoff Depth Migration of 2D poststack/prestack data

    sukdmig2d  infile=  outfile=  ttfile=   [parameters]

 Required parameters:
 infile=stdin file for input seismic traces
 outfile=stdout file for common offset migration output
 ttfile= file for input traveltime tables

 ...  The following 9 parameters describe traveltime tables:
 fzt=  first depth sample in traveltime table
 nzt=  number of depth samples in traveltime table
 dzt= depth interval in traveltime table
 fxt= first lateral sample in traveltime table
 nxt= number of lateral samples in traveltime table
 dxt= lateral interval in traveltime table
 fs=  x-coordinate of first source
 ns=  number of sources
 ds=  x-coordinate increment of sources

 Optional Parameters:
 dt= or from header (dt)  time sampling interval of input data
 ft= or from header (ft)  first time sample of input data
 dxm= or from header (d2)  sampling interval of midpoints
 fzo=fzt     z-coordinate of first point in output trace
 dzo=0.2*dzt vertical spacing of output trace
 nzo=5*(nzt-1)+1  number of points in output trace ",
 fxo=fxt     x-coordinate of first output trace
 dxo=0.5*dxt horizontal spacing of output trace
 nxo=2*(nxt-1)+1   number of output traces  ",
 off0=0     first offest in output
 doff=99999 offset increment in output
 noff=1   number of offsets in output  ",
 absoff=0      flag for using absolute offsets of input traces
               =0 means use offset=gx-sx
               =1 means use abs(gx-sx)
 limoff=0      flag for only using input traces that fall within the range
               of defined output offset bins (off0,doff,noff)
               =0 means use all input traces
               =1 means limit traces used by offset
 fmax=0.25/dt frequency-highcut for input traces
 offmax=99999 maximum absolute offset allowed in migration
 aperx=nxt*dxt/2   migration lateral aperature
```

```
 angmax=60 migration angle aperature from vertical
 v0=1500(m/s) reference velocity value at surface ",
 dvz=0.0   reference velocity vertical gradient

 ls=1 flag for line source
 jpfile=stderr job print file name

 mtr=100   print verbal information at every mtr traces
 ntr=100000 maximum number of input traces to be migrated
 npv=0 flag of computing quantities for velocity analysis
 rscale=1000.0  scaling for roundoff error suppression

   ...if npv>0 specify the following three files:
 tvfile=tvfile input file of traveltime variation tables
tv[ns][nxt][nzt]
 csfile=csfile input file of cosine tables cs[ns][nxt][nzt]
 outfile1=outfile1 file containning additional migration output
with extra amplitude

 Notes:
 1. Traveltime tables were generated by program rayt2d (or other ones)
    on relatively coarse grids, with dimension ns*nxt*nzt. In the
    migration process, traveltimes are interpolated into shot/gephone
    positions and output grids.
 2. Input seismic traces must be SU format and can be any type of
    gathers (common shot, common offset, common CDP, and so on). ",
 3. Migrated traces are output in CDP gathers if velocity analysis
    is required, with dimension nxo*noff*nzo.   ",
 4. If the offset value of an input trace is not in the offset array
    of output, the nearest one in the array is chosen.
 5. Memory requirement for this program is about
     [ns*nxt*nzt+noff*nxo*nzo+4*nr*nzt+5*nxt*nzt+npa*(2*ns*nxt*nzt
+noff*nxo*nzo+4*nxt*nzt)]*4 bytes
    where nr = 1+min(nxt*dxt,0.5*offmax+aperx)/dxo.
 6. Amplitudes are computed using the reference velocity profile, v(z),
    specified by the parameters v0= and dvz=.
 7. Input traces must specify source and receiver positions via the header
    fields tr.sx and tr.gx. Offset is computed automatically.
 8. if limoff=0, input traces from outside the range defined by off0, doff,
    noff, will get migrated into the extremal offset bins/planes.  E.g. if
    absoff=0 and limoff=0, all traces with gx<sx will get migrated into the
    off0 bin.
```

```
Author:  Zhenyue Liu, 03/01/95,  Colorado School of Mines
Modifcations:
   Gary Billings, Talisman Energy, Sept 2005:  added absoff, limoff.

Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: sx, gx
```

```
SUKDMIG3D - Kirchhoff Depth Migration of 3D poststack/prestack data

  sukdmig3d datain= dataout= [parameters]

 Required parameters:
 ttfile  file for input tttables

 Optional Parameters:

 datain=stdin  file for input seismic traces
 dataout=stdout file for common offset migration output
 crfile=NULL    file for cos theta and ray paths

   The following 17 parameters describe tttables: (from ttfile header)
 fxgd= or from header (f1) first x-sample in tttable
 nxt= or from header (ns) number of x-samples in tttable
 dxgd= or from header (d1) x-interval in tttable
 fygd= or from header (f2) first y-sample in tttable
 nyt= or from header (ntr) number of y-samples in tttable
 dygd= or from header (d2) y-interval in tttable
 ixsf= or from header (sdel)          x in dxgd of first source
 nxs= or from header (nhs)  number of sources in x
 ixsr= or from header (swdep)          ratio of source & gd spacing
 iysf= or from header (gdel)         y in dygd of first source
 nys= or from header (nvs) number of sources in y
 iysr= or from header (gwdep)         ratio of source & gd spacing
 fzs= or from header (sdepth/1000) first depth sample in tttable
 nzs= or from header (duse)  number of depth samples in tttable
 dzs= or from header (ep/1000) depth interval in tttable
 nxgd= or from header (selev) x size of the traveltime region
 nygd= or from header (gelev) y size of the traveltime region
 multit= or from header (scalel)       number of multivalued traveltime

   The following two parameters are from data header
 dt= or from header (dt) time sampling interval of input data
 ft= or from header (ft) first time sample of input data
 dxm= or from header (d2)      mid point spacing of input data

 Default: output is 5 times finer in depth and 2 times finer in lateral
 fzo=fzs z-coordinate of first point in output trace
 dzo=0.2*dzs vertical spacing of output trace (5 times finer)
 nzo=5*(nzs-1)+1 number of points in output trace (5 times finer) ",
 fxo=fxgd x-coordinate of first output trace
```

```
dxo=0.5*dxgd horizontal spacing of output trace (2 times finer)
nxo=2*(nxgd-1)+1 number of output traces (2 times finer)
fyo=fygd y-coordinate of first output trace
dyo=0.5*dygd horizontal spacing of output trace (2 times finer)
nyo=2*(nygd-1)+1 number of output traces (2 times finer)

Default: poststack migration ",
fxoffset=0 first offest in output in x
fyoffset=0 first offest in output in y
dxoffset=99999 offset increment in output in x
dyoffset=99999 offset increment in output in y
nxoffset=1 number of offsets in output in x
nyoffset=1 number of offsets in output in y ",
xoffsetmax=99999 x-maximum absolute offset allowed in migration
yoffsetmax=99999 y-maximum absolute offset allowed in migration
xaper=nxt*dxgd/2.5 migration lateral aperature in x
yaper=nyt*dygd/2.5 migration lateral aperature in y
angmax=60             max angle to handle
fmax=0.25/dt          max frequency in the data
jpfile=stderr job print file name
pptr=100 print verbal information at every pptr traces
ntrmax=100000 maximum number of input traces to be migrated
ls=0                  point =0 line source =1

Notes:
1. Traveltime tables were generated by program SUTETRARAY (or other
   ones) on very sparse tetrahedral model, with dimension nys*nxs*nzs
   *nyt*nxt.
2. Input seismic traces must be SU format and can be any type of
   gathers (common shot, common offset, common CDP, and so on). ",
3. Migrated traces are output in CDP gathers if velocity analysis
   is required, with dimension nyoffset*nxoffset*nyo*nxo*nzo. ",
4. If the offset value of an input trace is not in the offset array
   of output, the nearest one in the array is chosen.
5. Memory requirement for this program is about
   [nys*nxs*nzs*nyt*nxt+nyoffset*nxoffset*nxo*nyo*nzo+
   nys*nxo*nzo*nyt*nxt]
6. Input traces must specify source and receiver positions via header
   fields tr.sx and tr.gx, as well as tr.sy and tr.gy. Offset is
   computed automatically.

Disclaimer:
This is a research code that will take considerable work to get into
```

```
the form of a a production level 3D migration code. The code is
offered as is, along with tetramod and sutetraray, to provide a starting
point for researchers who wish to write their own 3D migration codes.


Author:  Zhaobo Meng, 01/10/97,  Colorado School of Mines

Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: sx, gx
```

```
 SUKTMIG2D - prestack time migration of a common-offset
section with the double-square root (DSR) operator


   suktmig2d < infile vfile= [parameters]  > outfile

 Required Parameters:
 vfile= rms velocity file (units/s) v(t,x) as a function
of time
 dx= distance (units) between consecutive traces

 Optional parameters:
 fcdpdata=tr.cdp first cdp in data
 firstcdp=fcdpdata first cdp number in velocity file
 lastcdp=from header last cdp number in velocity file
 dcdp=from header number of cdps between consecutive traces
 angmax=40 maximum aperture angle for migration (degrees)
 hoffset=.5*tr.offset half offset (m)
 nfc=16 number of Fourier-coefficients to approximate
low-pass
filters. The larger nfc the narrower the filter
 fwidth=5  high-end frequency increment for the low-pass
  filters
  in Hz. The lower this number the more the number
of lowpass filters to be calculated for each
input trace.

 Caveat: this code may need some work
 Notes:
 Data must be preprocessed with sufrac to correct for the
 wave-shaping factor using phasefac=.25 for 2D migration.

 Input traces must be sorted into offset and cdp number.
 The velocity file consists of rms velocities for all CMPs as a
 function of vertical time and horizontal position v(t,x)
 in C-style binary floating point numbers.  It's easiest to
 supply v(t,x) that has the same dimensions as the input data to
 be migrated. Note that time t is the fast dimension in these
 the input velocity file.

 The units may be feet or meters, as long as these are
 consistent.
 Antialias filter is performed using (Gray,1992, Geoph. Prosp),
```

```
 using nc low- pass filtered copies of the data. The cutoff
 frequencies are calculated  as fractions of the Nyquist
 frequency.

 The maximum allowed angle is 80 degrees(a 10 degree taper is
 applied to the end of the aperture)

define LOOKFAC 2       /* Look ahead factor for npfaro
define PFA_MAX 720720  /* Largest allowed nfft


 Prototype of functions used internally
void lpfilt(int nfc, int nfft, float dt, float fhi, float *filter);

segy intrace;  /* input traces
segy outtrace; /* migrated output traces

int
main(int argc, char **argv)
{
int i,k,imp,iip,it,ix,ifc; /* counters
int ntr,nt; /* x,t

int verbose; /* is verbose? */
int nc; /* number of low-pass filtered versions */
/*  of the data for antialiasing */
int nfft,nf; /* number of frequencies */
int nfc; /* number of Fourier coefficients for low-pass filter
int fwidth; /* high-end frequency increment for the low-pass
/* filters  */
int firstcdp=0; /* first cdp in velocity file */
int lastcdp=0; /* last cdp in velocity file */
int oldcdp=0; /* temporary storage */
int fcdpdata=0; /* first cdp in the data */
int olddeltacdp=0;
int deltacdp;
int ncdp=0; /* number of cdps in the velocity file */
int dcdp=0; /* number of cdps between consecutive traces

float dx=0.0; /* cdp sample interval
float hoffset=0.0;  /* half receiver-source
float p=0.0; /* horizontal slowness of the migration operator
float pmin=0.0; /* maximum horizontal slowness for which there's
```

```
/* no aliasing of the operator
float dt; /* t sample interval
float h; /* offset
float x; /* aperture distance
float xmax=0.0; /* maximum aperture distance

float obliq; /* obliquity factor
float geoms; /* geometrical spreading factor
float angmax;   /* maximum aperture angle

float mp,ip; /* mid-point and image-point coordinates
float t; /* time
float t0; /* vertical traveltime
float tmax; /* maximum time

float fnyq; /* Nyquist frequency
float ang; /* aperture angle
float angtaper=0.0; /* aperture-angle taper
float v; /* velocity

float *fc=NULL; /* cut-frequencies for low-pass filters
float *filter=NULL; /* array of low-pass filter values

float **vel=NULL; /* array of velocity values from vfile
float **data=NULL; /* input data array*/
float **lowpass=NULL;   /* low-pass filtered version of the trace
float **mig=NULL; /* output migrated data array

register float *rtin=NULL,*rtout=NULL;/* real traces
register complex *ct=NULL;   /* complex trace

/* file names
char *vfile=""; /* name of velocity file
FILE *vfp=NULL;
FILE *tracefp=NULL; /* temp file to hold traces*/
FILE *hfp=NULL; /* temp file to hold trace headers

float datalo[8], datahi[8];
int itb, ite;
float firstt, amplo, amphi;

cwp_Bool check_cdp=cwp_false; /* check cdp in velocity file */
```

```
/* Hook up getpar to handle the parameters
initargs(argc,argv);
requestdoc(0);

/* Get info from first trace
if (!gettr(&intrace))  err("can't get first trace");
nt=intrace.ns;
dt=(float)intrace.dt/1000000;
tmax=(nt-1)*dt;

MUSTGETPARFLOAT("dx",&dx);
MUSTGETPARSTRING("vfile",&vfile);
if (!getparfloat("angmax",&angmax)) angmax=40;
if (!getparint("firstcdp",&firstcdp)) firstcdp=intrace.cdp;
if (!getparint("fcdpdata",&fcdpdata)) fcdpdata=intrace.cdp;
if (!getparfloat("hoffset",&hoffset)) hoffset=.5*intrace.offset;
if (!getparint("nfc",&nfc)) nfc=16;
if (!getparint("fwidth",&fwidth)) fwidth=5;
if (!getparint("verbose",&verbose)) verbose=0;

h=hoffset;

/* Store traces in tmpfile while getting a count of number of traces
tracefp = etmpfile();
hfp = etmpfile();
ntr = 0;
do {
++ntr;

/* get new deltacdp value
deltacdp=intrace.cdp-oldcdp;

/* read headers and data
efwrite(&intrace,HDRBYTES, 1, hfp);
efwrite(intrace.data, FSIZE, nt, tracefp);

/* error trappings.
/* ...did cdp value interval change?
if ((ntr>3) && (olddeltacdp!=deltacdp)) {

if (verbose) {
warn("cdp interval changed in data");
warn("ntr=%d olddeltacdp=%d deltacdp=%d"
```

```
,ntr,olddeltacdp,deltacdp);
  check_cdp=cwp_true;
}
}

/* save cdp and deltacdp values
oldcdp=intrace.cdp;
olddeltacdp=deltacdp;

} while (gettr(&intrace));

/* get last cdp  and dcdp
if (!getparint("lastcdp",&lastcdp)) lastcdp=intrace.cdp;
if (!getparint("dcdp",&dcdp)) dcdp=deltacdp - 1;


checkpars();

/* error trappings
if ( (firstcdp==lastcdp)
|| (dcdp==0)
|| (check_cdp==cwp_true) ) warn("Check cdp values in data!");

/* rewind trace file pointer and header file pointer
erewind(tracefp);
erewind(hfp);

/* total number of cdp's in data
ncdp=lastcdp-firstcdp+1;

/* Set up FFT parameters
nfft = npfaro(nt, LOOKFAC*nt);
if(nfft>= SU_NFLTS || nfft >= PFA_MAX)
  err("Padded nt=%d -- too big",nfft);
nf = nfft/2 + 1;

/* Determine number of filters for antialiasing
fnyq= 1.0/(2*dt);
nc=ceil(fnyq/fwidth);
if (verbose)
warn(" The number of filters for antialiasing is nc= %d",nc);

/* Allocate space
```

```
data = alloc2float(nt,ntr);
lowpass=alloc2float(nt,nc+1);
mig=   alloc2float(nt,ntr);
vel=   alloc2float(nt,ncdp);
fc = alloc1float(nc+1);
rtin= ealloc1float(nfft);
rtout= ealloc1float(nfft);
ct= ealloc1complex(nf);
filter= alloc1float(nf);

/* Read data from temporal array
for (ix=0; ix<ntr; ++ix){
efread(data[ix],FSIZE,nt,tracefp);
}

/* read velocities
vfp=efopen(vfile,"r");
efread(vel[0],FSIZE,nt*ncdp,vfp);
efclose(vfp);

/* Zero all arrays
memset((void *) mig[0], 0,nt*ntr*FSIZE);
memset((void *) rtin, 0, nfft*FSIZE);
memset((void *) filter, 0, nf*FSIZE);
memset((void *) lowpass[0], 0,nt*(nc+1)*FSIZE);

/* Calculate cut frequencies for low-pass filters
for(i=1; i<nc+1; ++i){
fc[i]= fnyq*i/nc;
}

/* Start the migration process
/* Loop over input mid-points first
if (verbose) warn("Starting migration process...\n");
for(imp=0; imp<ntr; ++imp){
float perc;

mp=imp*dx;
perc=imp*100.0/(ntr-1);
if(fmod(imp*100,ntr-1)==0 && verbose)
warn("migrated %g\n ",perc);

/* Calculate low-pass filtered versions
```

```
/* of the data to be used for antialiasing
for(it=0; it<nt; ++it){
rtin[it]=data[imp][it];
}
for(ifc=1; ifc<nc+1; ++ifc){
memset((void *) rtout, 0, nfft*FSIZE);
memset((void *) ct, 0, nf*FSIZE);
lpfilt(nfc,nfft,dt,fc[ifc],filter);
pfarc(1,nfft,rtin,ct);

for(it=0; it<nf; ++it){
ct[it] = crmul(ct[it],filter[it]);
}
pfacr(-1,nfft,ct,rtout);
for(it=0; it<nt; ++it){
lowpass[ifc][it]= rtout[it];
}
}

/* Loop over vertical traveltimes
for(it=0; it<nt; ++it){
int lx,ux;

t0=it*dt;
v=vel[imp*dcdp+fcdpdata-1][it];
xmax=tan((angmax+10.0)*PI/180.0)*v*t0;
lx=MAX(0,imp - ceil(xmax/dx));
ux=MIN(ntr,imp + ceil(xmax/dx));

/* loop over output image-points to the left of the midpoint
for(iip=imp; iip>lx; --iip){
float ts,tr;
int fplo=0, fphi=0;
float ref,wlo,whi;

ip=iip*dx;
x=ip-mp;
ts=sqrt( pow(t0/2,2) + pow((x+h)/v,2) );
tr=sqrt( pow(t0/2,2) + pow((h-x)/v,2) );
t= ts + tr;
if(t>=tmax) break;
geoms=sqrt(1/(t*v));
   obliq=sqrt(.5*(1 + (t0*t0/(4*ts*tr))
```

```
- (1/(ts*tr))*sqrt(ts*ts - t0*t0/4)*sqrt(tr*tr - t0*t0/4)));
   ang=180.0*fabs(acos(t0/t))/PI;
   if(ang<=angmax) angtaper=1.0;
   if(ang>angmax) angtaper=cos((ang-angmax)*PI/20);
   /* Evaluate migration operator slowness p to determine
/* the low-pass filtered trace for antialiasing
pmin=1/(2*dx*fnyq);
p=fabs((x+h)/(pow(v,2)*ts) + (x-h)/(pow(v,2)*tr));
if(p>0){fplo=floor(nc*pmin/p);}
if(p==0){fplo=nc;}
ref=fmod(nc*pmin,p);
wlo=1-ref;
fphi=++fplo;
whi=ref;
itb=MAX(ceil(t/dt)-3,0);
ite=MIN(itb+8,nt);
firstt=(itb-1)*dt;
/* Move energy from CMP to CIP
if(fplo>=nc){
for(k=itb; k<ite; ++k){
datalo[k-itb]=lowpass[nc][k];
}
ints8r(8,dt,firstt,datalo,0.0,0.0,1,&t,&amplo);
mig[iip][it] +=geoms*obliq*angtaper*amplo;
} else if(fplo<nc){
for(k=itb; k<ite; ++k){
datalo[k-itb]=lowpass[fplo][k];
datahi[k-itb]=lowpass[fphi][k];
}
ints8r(8,dt,firstt,datalo,0.0,0.0,1,&t,&amplo);
ints8r(8,dt,firstt,datahi,0.0,0.0,1,&t,&amphi);
mig[iip][it] += geoms*obliq*angtaper*(wlo*amplo + whi*amphi);
}
}

/* loop over output image-points to the right of the midpoint
for(iip=imp+1; iip<ux; ++iip){
float ts,tr;
int fplo=0, fphi;
float ref,wlo,whi;

ip=iip*dx;
x=ip-mp;
```

```
t0=it*dt;
ts=sqrt( pow(t0/2,2) + pow((x+h)/v,2) );
tr=sqrt( pow(t0/2,2) + pow((h-x)/v,2) );
t= ts + tr;
if(t>=tmax) break;
geoms=sqrt(1/(t*v));
obliq=sqrt(.5*(1 + (t0*t0/(4*ts*tr))
- (1/(ts*tr))*sqrt(ts*ts
- t0*t0/4)*sqrt(tr*tr
- t0*t0/4)));
ang=180.0*fabs(acos(t0/t))/PI;
if(ang<=angmax) angtaper=1.0;
if(ang>angmax) angtaper=cos((ang-angmax)*PI/20.0);

/* Evaluate migration operator slowness p to determine the
/* low-pass filtered trace for antialiasing
pmin=1/(2*dx*fnyq);
p=fabs((x+h)/(pow(v,2)*ts) + (x-h)/(pow(v,2)*tr));
if(p>0){
fplo=floor(nc*pmin/p);
}
if(p==0){
fplo=nc;
}

ref=fmod(nc*pmin,p);
wlo=1-ref;
fphi=fplo+1;
whi=ref;
itb=MAX(ceil(t/dt)-3,0);
ite=MIN(itb+8,nt);
firstt=(itb-1)*dt;

/* Move energy from CMP to CIP
if(fplo>=nc){
for(k=itb; k<ite; ++k){
datalo[k-itb]=lowpass[nc][k];
}
ints8r(8,dt,firstt,datalo,0.0,0.0,1,&t,&amplo);
mig[iip][it] +=geoms*obliq*angtaper*amplo;
} else if(fplo<nc){
for(k=itb; k<ite; ++k){
datalo[k-itb]=lowpass[fplo][k];
```

```
datahi[k-itb]=lowpass[fphi][k];
}
ints8r(8,dt,firstt,datalo,0.0,0.0,1,&t,&amplo);
ints8r(8,dt,firstt,datahi,0.0,0.0,1,&t,&amphi);
mig[iip][it] += geoms*obliq*angtaper*(wlo*amplo + whi*amphi);
}
}

}
}

/* Output migrated data
erewind(hfp);
for (ix=0; ix<ntr; ++ix) {
efread(&outtrace, HDRBYTES, 1, hfp);
for (it=0; it<nt; ++it) {
outtrace.data[it] = mig[ix][it];
}
puttr(&outtrace);
}

efclose(hfp);

return(CWP_Exit());
}

void
lpfilt(int nfc, int nfft, float dt, float fhi, float *filter)
lpfilt -- low-pass filter using Lanczos Smoothing
(R.W. Hamming:"Digital Filtering",1977)
Input:
nfc number of Fourier coefficients to approximate ideal filter
nfft number of points in the fft
dt time sampling interval
fhi cut-frequency

Output:
filter  array[nf] of filter values
Notes: Filter is to be applied in the frequency domain
Author: CWP: Carlos Pacheco   2006
{
int i,j;  /* counters
int nf;   /* Number of frequencies (including Nyquist)
```

```
float onfft;  /* reciprocal of nfft
float fn; /* Nyquist frequency
float df; /* frequency interval
float dw; /* frequency interval in radians
float whi;/* cut-frequency in radians
float w;  /* radian frequency

nf= nfft/2 + 1;
onfft=1.0/nfft;
fn=1.0/(2*dt);
df=onfft/dt;
whi=fhi*PI/fn;
dw=df*PI/fn;

for(i=0; i<nf; ++i){
filter[i]= whi/PI;
w=i*dw;

for(j=1; j<nfc; ++j){
float c= sin(whi*j)*sin(PI*j/nfc)*2*nfc/(PI*PI*j*j);
filter[i] +=c*cos(j*w);
}
}
}
```

```
 SUMIGFD - 45-90 degree Finite difference depth migration for
           zero-offset data.

   sumigfd <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz= number of depth sapmles
 dz= depth sampling interval
 vfile= name of file containing velocities
  (see Notes below concerning format of this file)

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
 dx=from header(d2) or 1.0 midpoint sampling interval
 dip=45,65,79,80,87,89,90   Maximum angle of dip reflector

 tmpdir= if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
the CWP_TMPDIR environment variable is set use
its value for the path; else use tmpfile()

 Notes: ",
 The computation cost by dip angle is 45=65=79<80<87<89<90

 The input velocity file \'vfile\' consists of C-style binary floats. ",
 The structure of this file is vfile[iz][ix]. Note that this means that
 the x-direction is the fastest direction instead of z-direction! Such a
 structure is more convenient for the downward continuation type
 migration algorithm than using z as fastest dimension as in other SU
 programs. (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
 denotes a v(z,x) array, the opposite of what Matlab and Fortran
 programmers may expect.) ",

 Because most of the tools in the SU package (such as  unif2, unisam2,
 and makevel) produce output with the structure vfile[ix][iz], you will
 need to transpose the velocity files created by these programs. You may
 use the SU program \'transp\' in SU to transpose such files into the
 required vfile[iz][ix] structure.



 Credits: CWP Baoniu Han, April 20th, 1998
```

```
Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: ns, dt, delrt
```

```
 SUMIGFFD - Fourier finite difference depth migration for
    zero-offset data. This method is a hybrid migration which
    combines the advantages of phase shift and finite difference",
    migrations.

 sumigffd <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz=    number of depth sapmles   ",
 dz=    depth sampling interval
 vfile= name of file containing velocities

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
 dx=from header(d2) or 1.0     midpoint sampling interval
 ft=0.0 first time sample
 fz=0.0 first depth sample

 tmpdir= if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
the CWP_TMPDIR environment variable is set use
its value for the path; else use tmpfile()

 The input velocity file \'vfile\' consists of C-style binary floats.  ",
 The structure of this file is vfile[iz][ix]. Note that this means that
 the x-direction is the fastest direction instead of z-direction! Such a
 structure is more convenient for the downward continuation type
 migration algorithm than using z as fastest dimension as in other SU  ",
 programs. (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
 denotes a v(z,x) array, the opposite of what Matlab and Fortran
 programmers may expect.) ",

 Because most of the tools in the SU package (such as  unif2, unisam2, ",
 and makevel) produce output with the structure vfile[ix][iz], you will
 need to transpose the velocity files created by these programs. You may
 use the SU program \'transp\' in SU to transpose such files into the
 required vfile[iz][ix] structure.



 Credits: CWP Baoniu Han, July 21th, 1997
```

```
Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: ns, dt, delrt
```

```
 SUMIGGBZOAN - MIGration via Gaussian beams ANisotropic media (P-wave)

 sumiggbzoan <infile >outfile vfile= nt= nx= nz= [optional parameters]

 Required Parameters:
 a3333file= name of file containing a3333(x,z)
 nx=                    number of inline samples (traces)
 nz=                    number of depth samples

 Optional Parameters:
 dt=tr.dt               time sampling interval
 dx=tr.d2               inline sampling interval (trace spacing)
 dz=1.0                 depth sampling interval
 fmin=0.025/dt          minimum frequency
 fmax=10*fmin           maximum frequency
 amin=-amax             minimum emergence angle; must be > -90 degrees
 amax=60                maximum emergence angle; must be < 90 degrees
 bwh=0.5*vavg/fmin      beam half-width; vavg denotes average velocity
 verbose=0  silent, =1 chatty

 Files for general anisotropic parameters confined to a vertical plane:
 a1111file= name of file containing a1111(x,z)
 a1133file=           name of file containing a1133(x,z)
 a1313file=           name of file containing a1313(x,z)
 a1113file=           name of file containing a1113(x,z)
 a3313file=           name of file containing a3313(x,z)

 For transversely isotropic media Thomsen's parameters could be used:
 deltafile= name of file containing delta(x,z)
 epsilonfile= name of file containing epsilon(x,z)
 a1313file=           name of file containing a1313(x,z)

 if anisotropy parameters are not given the program considers ",
 the medium to be isotropic.


 Credits:
CWP: Tariq Alkhalifah,  based on MIGGBZO by Dave Hale
      CWP: repackaged as an SU program by John Stockwell, April 2006

   Technical Reference:

      Alkhailfah, T., 1993, Gaussian beam migration for
```

anisotropic media: submitted to Geophysics.

Cerveny, V., 1972, Seismic rays and ray intensities
in inhomogeneous anisotropic media:
Geophys. J. R. Astr. Soc., 29, 1--13.

Hale, D., 1992, Migration by the Kirchhoff,
slant stack, and Gaussian beam methods:
CWP,1992 Report 121, Colorado School of Mines.

Hale, D., 1992, Computational Aspects of Gaussian
Beam migration:
CWP,1992 Report 121, Colorado School of Mines.

```
SUMIGGBZO - MIGration via Gaussian Beams of Zero-Offset SU data

sumiggbzo <infile >outfile vfile=  nz= [optional parameters]

Required Parameters:
vfile=                  name of file containing v(z,x)
nz=                     number of depth samples

Optional Parameters:
dt=from header time sampling interval
dx=from header(d2) or 1.0 spatial sampling interval
dz=1.0                  depth sampling interval
fmin=0.025/dt           minimum frequency
fmax=10*fmin            maximum frequency
amin=-amax              minimum emergence angle; must be > -90 degrees
amax=60                 maximum emergence angle; must be < 90 degrees
bwh=0.5*vavg/fmin       beam half-width; vavg denotes average velocity
verbose=0  =0 silent; =1 chatty

Note: spatial units of v(z,x) must be the same as those of dx.
v(z,x) is represented numerically in C-style binary floats v[x][z],
where the depth direction is the fast direction in the data. Such
models can be created with unif2 or makevel.

(In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
denotes a v(z,x) array, the opposite of what Matlab and Fortran
programmers may expect.) ",

Caveat:
In the event of a "Segmentation Violation" try reducing the value of
the "bwh" parameter. Run program with verbose=1 do see the default
value.

Credits:

CWP: Dave Hale (algorithm), Jack K. Cohen, and John Stockwell
(reformatting for SU)
```

```
 SUMIGPREFD --- The 2-D prestack common-shot 45-90 degree
finite-difference depth migration.

    sumigprefd <indata >outfile [parameters]  ",

 Required Parameters: ",
 nxo= number of total horizontal output samples
 nxshot= number of shot gathers to be migrated
 nz= number of depth sapmles
 dx= horizontal sampling interval ",
 dz= depth sampling interval
 vfile= velocity profile, it must be binary format (see Notes)

 Optional Parameters:
 dip=79 the maximum dip to migrate, possible values are:
45,65,79,80,87,89,90 degrees
The computation cost is 45=65=79<80<87<89<90
 fmax=25 peak frequency of Ricker wavelet used as source wavelet
 f1=5,f2=10,f3=40,f4=50  frequencies to build a Hamming window

 lpad=9999,rpad=9999 number of zero traces padded on both
sides of depth section to determine the
migration aperature, the default values
are using the full aperature.
 verbose=0 silent, =1 additional runtime information

 Notes:
 The input velocity file \'vfile\' consists of C-style binary floats.
 The structure of this file is vfile[iz][ix]. Note that this means that
 the x-direction is the fastest direction instead of z-direction! Such a
 structure is more convenient for the downward continuation type
 migration algorithm than using z as fastest dimension as in other SU
 programs.

 Because most of the tools in the SU package (such as  unif2, unisam2, ",
 and makevel) produce output with the structure vfile[ix][iz], you will
 need to transpose the velocity files created by these programs. You may
 use the SU program \'transp\' in SU to transpose such files into the
 required vfile[iz][ix] structure.
 (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
 denotes a v(z,x) array, the opposite of what Matlab and Fortran
 programmers may expect.) ",
```

```
Also, sx must be monotonically increasing throughout the dataset, and
and gx must be monotonically increasing within a shot. You may resort
your data with \'susort\', accordingly.

The scalco header field is honored so this field must be set correctly.
See selfdocs of \'susort\', \'suchw\'. Also:   sukeyword scalco



Credits: CWP, Baoniu Han, bhan@dix.mines.edu, April 19th, 1998
 Modified: Chris Stolk, 11 Dec 2005, - changed data input
   to remove erroneous time delay.
 Modified: CWP, John Stockwell 26 Sept 2006 - replaced Han's
 "goto-loop" in two places with "do { }while loops".
 Fixed it so that sx, gx, and scalco are honored.



Trace header fields accessed: ns, dt, delrt, d2, sx, gx,
Trace header fields modified: ns, dt, delrt
```

```
SUMIGPREFFD - The 2-D prestack common-shot Fourier finite-difference
depth  migration.

  sumigpreffd <indata >outfile [parameters] ",

 Required Parameters: ",
 nxo=    number of total horizontal output samples
 nxshot= number of shot gathers to be migrated
 nz= number of depth sapmles
 dx= horizontal sampling interval
 dz= depth sampling interval
 vfile=  velocity profile, it must be binary format.

 Optional Parameters:
 fmax=25 the peak frequency of Ricker wavelet used as source wavelet
 f1=5,f2=10,f3=40,f4=50 frequencies to build a Hamming window
 lpad=9999,rpad=9999 number of zero traces padded on both
sides of depth section to determine the
migration aperature, the default
values are using the full aperature.
 verbose=0 silent, =1 additional runtime information

 Notes:
 The input velocity file consists of C-style binary floats. ",
 The structure of this file is vfile[iz][ix]. Note that this means that
 the x-direction is the fastest direction instead of z-direction! Such a
 structure is more convenient for the downward continuation type
 migration algorithm than using z as fastest dimension as in other SU  ",
 programs.

 Because most of the tools in the SU package (such as  unif2, unisam2, ",
 and makevel) produce output with the structure vfile[ix][iz], you will
 need to transpose the velocity files created by these programs. You may
 use the SU program \'transp\' in SU to transpose such files into the
 required vfile[iz][ix] structure.
 (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
 denotes a v(z,x) array, the opposite of what Matlab and Fortran
 programmers may expect.) ",

 Also, sx must be monotonically increasing throughout the dataset, and
 and gx must be monotonically increasing within a shot. You may resort
 your data with \'susort\', accordingly.
```

```
The scalco header field is honored so this field must be set correctly.
See selfdocs of \'susort\', \'suchw\'. Also:   sukeyword scalco



Credits: CWP, Baoniu Han, bhan@dix.mines.edu, April 19th, 1998

 Modified: Chris Stolk, 11 Dec 2005, - changed data input
   to remove erroneous time delay.
 Modified: CWP, John Stockwell 26 Sept 2006 - replaced Han's
 "goto-loop" with  "do { }while loops".
 Fixed it so that sx, gx, and scalco are honored.



Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: ns, dt, delrt
```

```
char *sdoc[] = {

" SUMIGPREPSPI --- The 2-D PREstack commom-shot Phase-Shift-Plus
" interpolation depth MIGration.

"   sumigprepspi <indata >outfile [parameters]  ",

" Required Parameters:

" nxo=     number of total horizontal output samples
" nxshot=  number of shot gathers to be migrated
" nz=      number of depth samples
" dx=      horizontal sampling interval    ",
" dz=      depth sampling interval
" vfile=   velocity profile, it must be binary format.

" Optional Parameters:
" fmax=25    the peak frequency of Ricker wavelet used as source wavelet
" f1=5,f2=10,f3=40,f4=50     frequencies to build a Hamming window
" lpad=9999,rpad=9999        number of zero traces padded on both
"                            sides of depth section to determine the
"                            migration aperture, the default values
"                            are using the full aperture.
" nflag=0    normalization of cross-correlation:
"            0: none, 1: by source wave field
" verbose=0  silent, =1 additional runtime information

" Notes:
" The input velocity file \'vfile\' consists of C-style binary floats. ",
" The structure of this file is vfile[iz][ix]. Note that this means that
" the x-direction is the fastest direction instead of z-direction! Such a
" structure is more convenient for the downward continuation type
" migration algorithm than using z as fastest dimension as in other SU  ",
" programs.

" Because most of the tools in the SU package (such as  unif2, unisam2, ",
" and makevel) produce output with the structure vfile[ix][iz], you will
" need to transpose the velocity files created by these programs. You may
" use the SU program \'transp\' in SU to transpose such files into the
" required vfile[iz][ix] structure.

" (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
" denotes a v(z,x) array, the opposite of what Matlab and Fortran
```

```
" programmers may expect.) ",

" Also, sx must be monotonically increasing throughout the dataset, and
" and gx must be monotonically increasing within a shot. You may resort
" your data with \'susort\', accordingly.

" The scalco header field is honored so this field must be set correctly.
" See selfdocs of \'susort\', \'suchw\'. Also:   sukeyword scalco



 * Credits: CWP, Baoniu Han, bhan@dix.mines.edu, April 19th, 1998
 *   Modified: Chris Stolk, 11 Dec 2005, - changed data input
 *     to remove erroneous time delay.
 *   Modified: CWP, John Stockwell 26 Sept 2006 - replaced Han's
 *   "goto-loop" in two places with "do { }while loops".
 *   Fixed it so that sx, gx, and scalco are honored.
 *
 *
 * Trace header fields accessed: ns, dt, delrt, d2
 * Trace header fields modified: ns, dt, delrt
```

```
 SUMIGPRESP - The 2-D prestack common-shot split-step Fourier ",
migration

   sumigpresp <indata >outfile [parameters] ",

 Required Parameters:
 nxo=    number of total horizontal output samples
 nxshot= number of shot gathers to be migrated
 nz=     number of depth sapmles
 dx=     horizontal sampling interval
 dz=     depth sampling interval
 vfile=  velocity profile, it must be binary format.

 Optional Parameters:
 fmax=25 The peak frequency of Ricker wavelet used as source wavelet
 f1=5,f2=10,f3=40,f4=50  frequencies to build a Hamming window
 lpad=9999,rpad=9999     number of zero traces padded on both
sides of depth section to determine the
migration aperature, the default values
are using the full aperature.
 verbose=0             silent, =1 additional runtime information

 Notes:
 The input velocity file consists of C-style binary floats.
 The structure of this file is vfile[iz][ix]. Note that this means that
 the x-direction is the fastest direction instead of z-direction! Such a
 structure is more convenient for the downward continuation type
 migration algorithm than using z as fastest dimension as in other SU
 programs.

 Because most of the tools in the SU package (such as  unif2, unisam2, ",
 and makevel) produce output with the structure vfile[ix][iz], you will
 need to transpose the velocity files created by these programs. You may
 use the SU program \'transp\' in SU to transpose such files into the
 required vfile[iz][ix] structure.
 (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
 denotes a v(z,x) array, the opposite of what Matlab and Fortran
 programmers may expect.) ",

 Also, sx must be monotonically increasing throughout the dataset, and
 and gx must be monotonically increasing within a shot. You may resort
 your data with \'susort\', accordingly.
```

```
The scalco header field is honored so this field must be set correctly.
See selfdocs of \'susort\', \'suchw\'. Also:   sukeyword scalco


Credits: CWP, Baoniu Han, bhan@dix.mines.edu, April 19th, 1998
         Modified: Chris Stolk, 11 Dec 2005, - changed data input
                   to remove erroneous time delay.
         Modified: CWP, John Stockwell 26 Sept 2006 - replaced Han's
         "goto-loop" in two places with "do { }while loops".
         Fixed it so that sx, gx, and scalco are honored.


Trace header fields accessed: ns, dt, delrt, d2, sx, gx, scalco
Trace header fields modified: ns, dt, delrt
```

```
 SUMIGPSPI - Gazdag's phase-shift plus interpolation depth migration
             for zero-offset data, which can handle the lateral
             velocity variation.

 sumigpspi <infile >outfile vfile= [optional parameters]

 Required Parameters:
 nz= number of depth sapmles
 dz= depth sampling interval
 vfile= name of file containing velocities
(Please see Notes below concerning the format of vfile)

 Optional Parameters:
 dt=from header(dt) or .004    time sampling interval
 dx=from header(d2) or 1.0     midpoint sampling interval

 tmpdir=        if non-empty, use the value as a directory path
                prefix for storing temporary files; else if the
                the CWP_TMPDIR environment variable is set use
                its value for the path; else use tmpfile()

 Notes:
 The input velocity file 'vfile' consists of C-style binary floats.
 The structure of this file is vfile[iz][ix]. Note that this means that
 the x-direction is the fastest direction instead of z-direction! Such a
 structure is more convenient for the downward continuation type
 migration algorithm than using z as fastest dimension as in other SU
 programs. (In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
 denotes a v(z,x) array, the opposite of what Matlab and Fortran
 programmers may expect.)

 Because most of the tools in the SU package (such as  unif2, unisam2,
 and makevel) produce output with the structure vfile[ix][iz], you will
 need to transpose the velocity files created by these programs. You may
 use the SU program 'transp' in SU to transpose such files into the
 required vfile[iz][ix] structure.




 Credits: CWP, Baoniu Han, April 20th, 1998

 Trace header fields accessed: ns, dt, delrt, d2
```

```
Trace header fields modified: ns, dt, delrt
```

```
SUMIGPS - MIGration by Phase Shift with turning rays

sumigps <stdin >stdout [optional parms]

Required Parameters:
 None

Optional Parameters:
dt=from header(dt) or .004 time sampling interval
dx=from header(d2) or 1.0 distance between sucessive cdp's
ffil=0,0,0.5/dt,0.5/dt  trapezoidal window of frequencies to migrate
tmig=0.0 times corresponding to interval velocities in vmig
vmig=1500.0 interval velocities corresponding to times in tmig
vfile= binary (non-ascii) file containing velocities v(t)
nxpad=0 number of cdps to pad with zeros before FFT
ltaper=0 length of linear taper for left and right edges",
verbose=0 =1 for diagnostic print


tmpdir=   if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
        the CWP_TMPDIR environment variable is set use
        its value for the path; else use tmpfile()

Notes:
Input traces must be sorted by either increasing or decreasing cdp.

The tmig and vmig arrays specify an interval velocity function of time.
Linear interpolation and constant extrapolation is used to determine
interval velocities at times not specified.  Values specified in tmig
must increase monotonically.

Alternatively, interval velocities may be stored in a binary file
containing one velocity for every time sample.  If vfile is specified,
then the tmig and vmig arrays are ignored.

The time of first sample is assumed to be zero, regardless of the value
of the trace header field delrt.

Credits:
CWP: Dave Hale (originally called supsmig.c)

 Trace header fields accessed:  ns, dt, d2
```

```
SUMIGPSTI - MIGration by Phase Shift for TI media with turning rays

sumigpsti <stdin >stdout [optional parms]

Required Parameters:
 None

Optional Parameters:
dt=from header(dt) or .004 time sampling interval
dx=from header(d2) or 1.0 distance between sucessive cdp's
ffil=0,0,0.5/dt,0.5/dt  trapezoidal window of frequencies to migrate
tmig=0.0 times corresponding to interval velocities in vmig
vnmig=1500.0 interval NMO velocities corresponding to times in tmig
vmig=1500.0 interval velocities corresponding to times in tmig
etamig=0.0 interval eta values corresponding to times in tmig
vnfile= binary (non-ascii) file containing NMO velocities vn(t)
vfile= binary (non-ascii) file containing velocities v(t)
etafile= binary (non-ascii) file containing eta values eta(t)
nxpad=0 number of cdps to pad with zeros before FFT
ltaper=0 length of linear taper for left and right edges ",
verbose=0 =1 for diagnostic print

Notes:
Input traces must be sorted by either increasing or decreasing cdp.

The tmig, vnmig, vmig and etamig arrays specify an interval values
function of time. Linear interpolation and constant extrapolation is
used to determine interval velocities at times not specified.  Values
specified in tmig must increase monotonically.
Alternatively, interval velocities may be stored in a binary file
containing one velocity for every time sample.  If vnfile is specified,
then the tmig and vnmig arrays are ignored.

The time of first sample is assumed to be zero, regardless of the value
of the trace header field delrt.

Trace header fields accessed:  ns and dt
```

```
SUMIGSPLIT - Split-step depth migration for zero-offset data.

sumigsplit <infile >outfile vfile= [optional parameters]

Required Parameters:
nz=                   number of depth sapmles
dz=                   depth sampling interval
vfile=                name of file containing velocities

Optional Parameters:
dt=from header(dt) or .004    time sampling interval
dx=from header(d2) or 1.0     midpoint sampling interval
ft=0.0                        first time sample
fz=                           first depth sample

tmpdir=        if non-empty, use the value as a directory path
               prefix for storing temporary files; else if the
               the CWP_TMPDIR environment variable is set use
               its value for the path; else use tmpfile()


Notes:
The input velocity file \'vfile\' consists of C-style binary floats.
The structure of this file is vfile[iz][ix]. Note that this means that
the x-direction is the fastest direction instead of z-direction! Such a
structure is more convenient for the downward continuation type
migration algorithm than using z as fastest dimension as in other SU
programs.

Because most of the tools in the SU package (such as  unif2, unisam2,
and makevel) produce output with the structure vfile[ix][iz], you will
need to transpose the velocity files created by these programs. You may
use the SU program \'transp\' in SU to transpose such files into the
required vfile[iz][ix] structure.
(In C  v[iz][ix] denotes a v(x,z) array, whereas v[ix][iz]
denotes a v(z,x) array, the opposite of what Matlab and Fortran
programmers may expect.) ",



Credits: CWP Baoniu Han, July 21th, 1997

Reference:
```

Stoffa, P. L. and Fokkema, J. T. and Freire, R. M. and Kessinger, W. P., 1990, Split-step Fourier migration, Geophysics, 55, 410-421.

Trace header fields accessed: ns, dt, delrt, d2
Trace header fields modified: ns, dt, delrt

```
SUMIGTK - MIGration via T-K domain method for common-midpoint stacked data

sumigtk <stdin >stdout dxcdp= [optional parms]

Required Parameters:
dxcdp                   distance between successive cdps

Optional Parameters:
fmax=Nyquist            maximum frequency
tmig=0.0                times corresponding to interval velocities in vmig
vmig=1500.0             interval velocities corresponding to times in tmig
vfile=                  binary (non-ascii) file containing velocities v(t)
nxpad=0                 number of cdps to pad with zeros before FFT
ltaper=0                length of linear taper for left and right edges",
verbose=0               =1 for diagnostic print

tmpdir=   if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
        the CWP_TMPDIR environment variable is set use
        its value for the path; else use tmpfile()

Notes:
Input traces must be sorted by either increasing or decreasing cdp.

The tmig and vmig arrays specify an interval velocity function of time.
Linear interpolation and constant extrapolation is used to determine
interval velocities at times not specified.  Values specified in tmig
must increase monotonically.

Alternatively, interval velocities may be stored in a binary file
containing one velocity for every time sample.  If vfile is specified,
then the tmig and vmig arrays are ignored.

The time of first sample is assumed to be zero, regardless of the value
of the trace header field delrt.

The migration is a reverse time migration in the (t,k) domain. In the
first step, the data g(t,x) are Fourier transformed x->k into the ",
the time-wavenumber domain g(t,k).

Then looping over wavenumbers, the data are then reverse-time
finite-difference migrated, wavenumber by wavenumber.  The resulting
migrated data m(tau,k), now in the tau (migrated time) and k domain,
```

```
 are inverse fourier transformed back into m(tau,xout) and written out.",


 Credits:
CWP: Dave Hale

 Trace header fields accessed:  ns and dt
```

```
SUMIGTOPO2D - Kirchhoff Depth Migration of 2D postack/prestack data
    from the (variable topography) recording surface

   sumigtopo2d  infile=  outfile=  [parameters]

Required parameters:
infile=stdin file for input seismic traces
outfile=stdout file for common offset migration output
ttfile file for input traveltime tables
  The following 9 parameters describe traveltime tables:
fzt  first depth sample in traveltime table
nzt  number of depth samples in traveltime table
dzt depth interval in traveltime table
fxt first lateral sample in traveltime table
nxt number of lateral samples in traveltime table
dxt lateral interval in traveltime table
fs  x-coordinate of first source
ns  number of sources
ds  x-coordinate increment of sources

fxi                   x-coordinate of the first input trace
dxi                   horizontal spacing of input data
nxi                   number of input trace locations in surface

Optional Parameters:
dt= or from header (dt)  time sampling interval of input data
ft= or from header (ft)  first time sample of input data
dxm= or from header (d2)  sampling interval of midpoints
surf="0,0;99999,0"  Recording surface "x1,z1;x2,z2;x3,z3;...
fzo=fzt               z-coordinate of first point in output trace
dzo=0.2*dzt vertical spacing of output trace
nzo=5*(nzt-1)+1  number of points in output trace ",
fxo=fxt               x-coordinate of first output trace
dxo=0.5*dxt horizontal spacing of output trace
nxo=2*(nxt-1)+1   number of output traces  ",
off0=0                first offest in output
doff=99999 offset increment in output
noff=1        number of offsets in output  ",
fmax=0.25/dt frequency-highcut for input traces
offmax=99999 maximum absolute offset allowed in migration
aperx=nxt*dxt/2   migration lateral aperature
angmax=60 migration angle aperature from vertical
v0=1500(m/s) reference velocity value at surface ",
```

```
dvz=0.0   reference velocity vertical gradient
ls=1                  flag for line source
jpfile=stderr job print file name
mtr=100   print verbal information at every mtr traces
ntr=100000 maximum number of input traces to be migrated

Notes:
1. Traveltime tables were generated by program rayt2dtopo (or any
   other one that considers topography )on relatively coarse grids,
   with dimension ns*nxt*nzt. In the migration process, traveltimes
   are interpolated into shot/gephone positions and output grids.
2. Input seismic traces must be SU format and can be any type of
   gathers (common shot, common offset, common CDP, and so on). ",
3. Migrated traces are output in CDP gathers if velocity analysis
   is required, with dimension nxo*noff*nzo.   ",
4. If the offset value of an input trace is not in the offset array
   of output, the nearest one in the array is chosen.
5. Amplitudes are computed using the reference velocity profile, v(z),
   specified by the parameters v0= and dvz=.
6. Input traces must specify source and receiver positions via the header
   fields tr.sx and tr.gx. Offset is computed automatically.


Author:  Zhenyue Liu, 03/01/95,  Colorado School of Mines

   Trino Salinas, 07/01/96, Colorado School of Mines,
         extended the code to migrate data from a nonflat
         recording surface.

References :

Bleistein, N., Cohen, J., and Hagin, F., 1987, Two and one-half
  dimensional Born inversion with arbitrary reference: Geophysics,
  52, 26-36.

Liu,Z., 1993, A Kirchhoff approach to seismic modeling and
  pre-stack depth migration: CWP Annual Report, CWP, Colorado
  School of Mines.

Wiggins, J. W., 1984, Kirchhoff integral extrapolation and
  migration of nonplanar data: Geophysics, 49, 1239-
  1248.
```

```
 SUSTOLT - Stolt migration for stacked data or common-offset gathers

 sustolt <stdin >stdout cdpmin= cdpmax= dxcdp= noffmix= [...]

 Required Parameters:
 cdpmin=   minimum cdp (integer number) in dataset
 cdpmax=   maximum cdp (integer number) in dataset
 dxcdp=   distance between adjacent cdp bins (m)

 Optional Parameters:
 noffmix=1 number of offsets to mix (for unstacked data only)
 tmig=0.0 times corresponding to rms velocities in vmig (s)
 vmig=1500.0 rms velocities corresponding to times in tmig (m/s)
 smig=1.0 stretch factor (0.6 typical if vrms increasing)
 vscale=1.0 scale factor to apply to velocities
 fmax=Nyquist maximum frequency in input data (Hz)
 lstaper=0 length of side tapers (# of traces)
 lbtaper=0 length of bottom taper (# of samples)
 verbose=0 =1 for diagnostic print
 tmpdir= if non-empty, use the value as a directory path
prefix for storing temporary files; else if the
the CWP_TMPDIR environment variable is set use
its value for the path; else use tmpfile()

 Notes:
 If unstacked traces are input, they should be NMO-corrected and sorted
 into common-offset  gathers.  One common-offset gather ends and another
 begins when the offset field of the trace headers changes. If both
 NMO and DMO are applied, then this is equivalent to prestack time
 migration (though the velocity profile is assumed v(t), only).

 The cdp field of the input trace headers must be the cdp bin NUMBER, NOT
 the cdp location expressed in units of meters or feet.

 The number of offsets to mix (noffmix) should be specified for
 unstacked data only. noffmix should typically equal the ratio of the
 shotpoint spacing to the cdp spacing.  This choice ensures that every
 cdp will be represented in each offset mix.  Traces in each mix will
 contribute through migration to other traces in adjacent cdps within
 that mix.

 The tmig and vmig arrays specify a velocity function of time that is
 used to implement Stolt's stretch for depth-variable velocity.  The
```

```
 stretch factor smig is often referred to as the "W" factor.
 The times in tmig must be monotonically increasing.

 Credits:
CWP: Dave Hale c. 1990

 Trace header fields accessed:  ns, dt, delrt, offset, cdp
```

```
 SUTIFOWLER   VTI constant velocity prestack time migration
      velocity analysis via Fowler's method

 sutifowler ncdps=250 vmin=1500 vmax=6000 dx=12.5

 Required Parameter:
 ncdps= number of input cdp's
 Optional Parameters:
 choose=1 1 do full prestack time migration
2 do only DMO
3 do only post-stack migrations
4 do only stacking velocity analysis
 getcvstacks=0 flag to set to 1 if inputting precomputed cvstacks
(vmin, nvstack, and ncdps must match SUCVS4FOWLER job)
 vminstack=vmin minimum velocity panel in m/s in input cvstacks
 etamin=0. minimum eta (see paper by Tariq Alkhalifah)
 etamax=0.5 maximum eta (see paper by Tariq Alkhalifah)
 neta=1 number of eta values to image
 d=0. Thomsen's delta
 vpvs=0.5 assumed vp/vs ratio (not critical -- default almost always ok)
 dx=25. cdp x increment
 vmin=1500. minimum velocity panel in m/s to output
 vmax=8000. maximum velocity panel in m/s to output
 nv=75  number of velocity panels to output
 nvstack=180 number of stacking velocity panels to compute
     ( Let offmax be the maximum offset, fmax be
     the maximum freq to preserve, and tmute be
     the starting mute time in sec on offmax, then
     the recommended value for nvstack would be
     nvstack = 4 +(offmax*offmax*fmax)/(0.6*vmin*vmin*tmute)
     ---you may want to make do with less---)
 nxpad=0   number of traces to padd for spatial fft
     Ideally nxpad = (0.5*tmax*vmax+0.5*offmax)/dx
 lmute=24  length of mute taper in ms
 nonhyp=1   1 if do mute at 2*offset/vmin to avoid non-hyperbolic
moveout, 0 otherwise
 lbtaper=0 length of bottom taper in ms
 lstaper=0 length of side taper in traces
 dtout=1.5*dt output sample rate in s,   note: typically
fmax=salias*0.5/dtout
 mxfold=120 maximum number of offsets/input cmp
 salias=0.8 fraction of output frequencies to force within sloth
     antialias limit.  This controls muting by offset of
```

```
    the input data prior to computing the cv stacks
    for values of choose=1 or choose=2.
 file=sutifowler root name for temporary files
 p=not Enter a path name where to put temporary files.
   specified  Can enter multiple times to use multiple disk
systems.
    The default uses information from the .VND file
    in the current directory if it exists, or puts
    unique temporary files in the current directory.
 ngroup=20 Number of cmps per velocity analysis group.
 printfile=stderr    The output file for printout from this program.

 Required trace header words on input are ns, dt, cdp, offset.
 On output, trace headers are rebuilt from scratch with
 ns - number of samples
 dt - sample rate in usec
 cdp - the output cmp number (0 based)
 offset - the output velocity
 tracf - the output velocity index (0 based)
 fldr - index for velocity analysis group (0 based, groups of ngroup cdps)
 ep - central cmp for velocity analysis group
 igc - index for choice of eta (0 based)
 igi - eta*100
 sx=gx - x coordinate as icmp*dx
 tracl=tracr -sequential trace count (1 based)

 Note: Due to aliasing considerations, the small offset-to-depth
 ratio assumption inherent in the TI DMO derivation, and the
 poor stacking of some large-offset events associated with TI non-hyperbolic
 moveout, a fairly stiff initial mute is recommended for the
 long offsets.  As a result, this method may not work well
 where you have multiple reflections to remove via stacking.

 Note: The temporary files can be split over multiple disks by building
 a .VND file in your working directory.  The .VND file is ascii text
 with the first line giving the number of directories followed by
 successive lines with one line per directory name.

 Note: The output data order has primary key equal to cdp, secondary
 key equal to eta, and tertiary key equal to velocity.

 Credits:
CWP: John Anderson (visitor to CSM from Mobil) Spring 1993
```

```
 SUALFORD - trace by trace Alford Rotation of shear wave data volumes

 sualford inS11=file1 inS22=file2 inS12=file3 inS21=file4
 outS11=file5 outS22=file6 outS12=file7 outS21=file8 [optional
 parameters]

 Required Parameters:
 inS11= input data volume for the 11 component
 inS12= input data volume for the 12 component
 inS21= input data volume for the 21 component
 inS22= input data volume for the 22 component
 outS11= output data volume for the 11 component
 outS12= output data volume for the 11 component
 outS21= output data volume for the 11 component
 outS22= output data volume for the 11 component

 Optional parameters:
 angle_inc=               sets the increment to the angle by which
                         the data sets are rotated. The minimum is
                         set to be 1 degree and default is 5.
 Az_key=                  to set the header field storing the azimuths
                         for the fast shear wave on the output volumes
 Q_key=                   to set the header field storing the quality
                         factors of performed optimum rotations
 lag_key=                 to set the header field storing the lag in
                         miliseconds the fast and slow shear components
 xcorr_key=               to set the header field containing the maxi-
                         mum normalized cross-correlation between the
                         and slow shear waves.
 taper=   2*taper+1 is the length of the sample overlap
  between the unrotated data with the rotated
  data on the traces. The boundary between them
    is defined by time windowning.
taper = -1, for no-overlap
taper = 0, for overlap of one sample
taper =>1, for use of cosine scale to
   to interpolate between the
   unrotated and rotated parts
   of the traces

 taperwin=               another taper used to taper the data within
  the window of analysis to diminish the effect
                         of data near the window edges.In this way one
```

```
                        can focus on a given reflector. Also given in
                        number of samples

 maxlag=   maximum limit in ms for the lag between fast
    and slow shear waves. If this threshold is
  attained or surpassed, the quality factor for
  the rotation is zeroed as well as all the
  parameters found for that certain rotation


 ntraces=   number of traces to be used per computation
  ntraces=3 will use three adjacent traces to
          compute the angle of rotation                "

 Notes:

 The Alford Rotation is a method to rotate the four components
 of a shear wave survey into its natural coordinate system, where
 the fast and slow shear correspond to the inline to inline shear (S11)
 and xline to xline (S22) volumes, respectively.

 This Alford Rotation code tries to maximize the energy in the
 diagonal volumes, i.e., S11 and S22, while minimizing the energy
 in the off-diagonals, i.e., in volumes S12 and S21, in a trace by
 trace manner. It then returns the new rotated volumes, saving the
 the quality factor for the rotation and azimuth angle of the fast
 shear wave direction for each trace headers of the new rotated S11
 volume.

 The fields in the header containing the Azimuth and Quality factor
 and the sample lag between fast and slow shear are otrav, grnolf and
 grnofr, respectively, by default. The values are multiplied by ten in
 the case of the angles and by a thousand for quality factors. To
 change this defaults use the optional parameters Az_key, Q_key and
 lag_key


 modified header fields:
 the ones specified by Az_key, Q_key, lag_key and xcorr_key. By default
 these are otrav, grnlof, tstat and grnors, respectively.

 Credits:
CWP: Rodrigo Felicio Fuck
```

Code translated and adapted from original version in Fortran
by Ted Schuck (1993)

Schuck, E. L. , 1993, Multicomponent, three dimensional seismic characterization of a fractured coalbed methane reservoir, Cedar Hill Field, San Juan County, New Mexico, Ph.D. Thesis, Colorado School of Mines

```
SUEIPOFI - EIgenimage (SVD) based POlarization FIlter for
           three-component data

sueipofi <stdin >stdout [optional parameters]

Required parameters:
   none

Optional parameters:
   dt=(from header)  time sampling intervall in seconds
   wl=0.1            SVD time window length in seconds
   pwr=1.0           exponent of filter weights
   interp=cubic      interpolation between initially calculated
                     weights, choose "cubic" or "linear
   verbose=0         1 = echo additional information

   file=polar        base name for additional output file(s) of
                     filter weights (see flags below)
   rl1=0             1 = rectilinearity along first principal axis
   rl2=0             1 = rectilinearity along second principal axis
   pln=0             1 = planarity


Notes:
   Three adjacent traces are considered as one three-component
   dataset.

   The filter is the sum of the first two eigenimages of the singular
   value decomposition (SVD) of the signal matrix (time window).
   Weighting functions depending on linearity and planarity of the
   signal are applied, additionally. To avoid edge effects, these are
   interpolated linearily or via cubic splines between initially
   calculated values of non-overlapping time windows.
   The algorithm is based on the assumption that the particle motion
   trajectory is essentially 2D (elliptical polarization).

Caveat:
   Cubic spline interpolation may result in filter weights exceeding
   the set of values of initial weights. Weights outside the valid
   interval [0.0, 1.0] are clipped.
```

Author: Nils Maercklin,
GeoForschungsZentrum (GFZ) Potsdam, Germany, 2001.
E-mail: nils@gfz-potsdam.de

References:

Franco, R. de, and Musacchio, G., 2000: Polarization Filter with Singular Value Decomposition, submitted to Geophysics and published electronically in Geophysics online (www.geo-online.org).

Jurkevics, A., 1988: Polarization analysis of three-comomponent array data, Bulletin of the Seismological Society of America, vol. 78, no. 5.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. 1996: Numerical Recipes in C - The Art of Scientific Computing, Cambridge University Press, Cambridge.

Trace header fields accessed: ns, dt
Trace header fields modified: none

```
 SUHROT - Horizontal ROTation of three-component data

 suhrot <stdin >stdout [optional parameters]

 Required parameters:
 none

 Optional parameters:
 angle=rad unit of angles, choose "rad", "deg", or "gon
 inv=0 1 = inverse rotation (counter-clockwise)
 verbose=0 1 = echo angle for each 3-C station

 a=... array of user-supplied rotation angles
 x=0.0,... array of corresponding header value(s)
 key=tracf header word defining 3-C station ("x")

 ... or input angles from files:
 n=0  number of x and a values in input files
 xfile=...   file containing the x values as specified by the
  "key" parameter
 afile=...   file containing the a values

 Notes:
 Three adjacent traces are considered as one three-component
 dataset.
 By default, the data will be rotated from the Z-North-East (Z,N,E)
 coordinate system into Z-Radial-Transverse (Z,R,T).

If one of the parameters "a=" or "afile=" is set, the data
are rotated by these user-supplied angles. Specified x values
must be monotonically increasing or decreasing, and afile and
xfile are files of binary (C-style) floats.



 Author: Nils Maercklin,
 Geophysics, Kiel University, Germany, 1999.


 Trace header fields accessed: ns, sx, sy, gx, gy, key=keyword
 Trace header fields modified: trid
```

```
 SULTT - trace by trace, sample by sample, rotation of shear wave data
  volumes using the Linear Transform Technique of Li & Crampin
  (1993)

 sultt inS11=file1 inS22=file2 inS12=file3 inS21=file4 [optional
 parameters]

 optional parameters:

 mode determines what the linear transform will compute
mode=1, computes asymmetry indexes
  mode=2, computes Polarization and main
reflectivity series.
mode=3, same as above, but using eigenvalues

  mode=3 is more robust estimation for Polarization angle
than mode=2. In both cases the reflectivity series is
computed in the same way. mode=1 outputs two other data
volumes, each containing the asymmetry parameters theta,
and gamma. The other two modes only output an extra da-
ta volume, the instant polarization alpha

 outSij defines the names of the output seismic data files,
i and j equal either 1 or 2

 alpha, gamma name for optional output volumes: instanteneous polarity
 theta , alpha; theta, for angle misalignment between source
and receiver; gamma, the medium asymmetric response
coefficient

 wl running window acting on traces (in samples)

 ntraces number of traces to be average for each location

 CAVEAT

 Naming convention for off-diagonal volumes:
 S12 - Inline source, Xline receiver
 S21 - Xline source, Inline receiver

 the running will always have an odd number of samples, despite the
 input length.
```

Credits:
CWP/RCP: Rodrigo Felicio Fuck
Code based on algorithms presented in Li & Crampin (1993) and Li & MacBeth (1997)

Li, X.Y., and Crampin, S., 1993, Linear-transform techniques for processing shear-wave anisotropy in four-component seismic data, Geophysics, 58, 240-256.
Li, X.Y., and MacBeth, C., 1997, Data-Matrix asymmetry and polarization changes from multicomponent surface seismics

```
SUPOFILT - POlarization FILTer for three-component data

supofilt <stdin >stdout [optional parameters]

Required parameters:
   dfile=polar.dir   file containing the 3 components of the
                     direction of polarization
   wfile=polar.rl    file name of weighting polarization parameter

Optional parameters:
   dt=(from header)  time sampling intervall in seconds
   smooth=1          1 = smooth filter operators, 0 do not
   sl=0.05           smoothing window length in seconds
   wpow=1.0          raise weighting function to power wpow
   dpow=1.0          raise directivity functions to power dpow
   verbose=0         1 = echo additional information


Notes:
   Three adjacent traces are considered as one three-component
   dataset.

   This program SUPOFILT is an extension to the polarization analysis
   program supolar. The files wfile and dfile are SU files as written
   by SUPOLAR.



Author: Nils Maercklin,
        GeoForschungsZentrum (GFZ) Potsdam, Germany, 1999-2000.
        E-mail: nils@gfz-potsdam.de


References:
   Benhama, A., Cliet, C. and Dubesset, M., 1986: Study and
      Application of spatial directional filtering in three
      component recordings.
      Geophysical Prospecting, vol. 36.
   Kanasewich, E. R., 1981: Time Sequence Analysis in Geophysics,
      The University of Alberta Press.
   Kanasewich, E. R., 1990: Seismic Noise Attenuation,
      Handbook of Geophysical Exploration, Pergamon Press, Oxford.
```

```
Trace header fields accessed: ns, dt
```

```
SUPOLAR - POLarization analysis of three-component data

supolar <stdin [optional parameters]

Required parameters:
   none

Optional parameters:
   dt=(from header)  time sampling intervall in seconds
   wl=0.1            correlation window length in seconds
   win=boxcar        correlation window shape, choose "boxcar",
                     "hanning", "bartlett", or "welsh
   file=polar        base of output file name(s)
   rl=1              1 = rectilinearity evaluating 2 eigenvalues,
                     2, 3 = rectilinearity evaluating 3 eigenvalues
   rlq=1.0           contrast parameter for rectilinearity
   dir=1             1 = 3 components of direction of polarization
                         (the only three-component output file)
   tau=0             1 = global polarization parameter
   ellip=0           1 = principal, subprincipal, and transverse
                         ellipticities e21, e31, and e32
   pln=0             1 = planarity measure
   f1=0              1 = flatness or oblateness coefficient
   l1=0              1 = linearity coefficient
   amp=0             1 = amplitude parameters: instantaneous,
                         quadratic, and eigenresultant ir, qr, and er
   theta=0           1, 2, 3 = incidence angle of principal axis
   phi=0             1, 2, 3 = horizontal azimuth of principal axis
   angle=rad         unit of angles theta and phi, choose "rad",
                     "deg", or "gon
   all=0             1, 2, 3 = set all output flags to that value
   verbose=0         1 = echo additional information


Notes:
   Three adjacent traces are considered as one three-component
   dataset.
   Correct calculation of angles theta and phi requires the first of
   these traces to be the vertical component, followed by the two
   horizontal components (e.g. Z, N, E, or Z, inline, crossline).
   Significant signal energy on Z is necessary to resolve the 180 deg
   ambiguity of phi (options phi=2,3 only).
```

```
   Each calculated polarization attribute is written into its own
   SU file. These files get the same base name (set with "file=")
   and the parameter flag as an extension (e.g. polar.rl).

   In case of a tapered correlation window, the window length wl may
   have to be increased compared to the boxcar case, because of their
   smaller effective widths (Bartlett, Hanning: 1/2, Welsh: 1/3).

Range of values:
   parameter     option  interval
   rl            1, 2    0.0 ... 1.0   (1.0: linear polarization)
   rl            3      -1.0 ... 1.0
   tau, l1       1       0.0 ... 1.0   (1.0: linear polarization)
   pln, f1       1       0.0 ... 1.0   (1.0: planar polarization)
   e21, e31, e32 1       0.0 ... 1.0   (0.0: linear polarization)
   theta         1      -pi/2... pi/2  rad
   theta         2, 3    0.0 ... pi/2  rad
   phi           1      -pi/2... pi/2  rad
   phi           2      -pi  ... pi    rad   (see notes above)
   phi           3       0.0 ... 2 pi  rad   (see notes above)



Author: Nils Maercklin,
        GeoForschungsZentrum (GFZ) Potsdam, Germany, 1998-2001.
        E-mail: nils@gfz-potsdam.de


References:
   Jurkevics, A., 1988: Polarization analysis of three-component
      array data. Bulletin of the Seismological Society of America,
      vol. 78, no. 5.
   Kanasewich, E. R., 1981: Time Sequence Analysis in Geophysics.
      The University of Alberta Press.
   Kanasewich, E. R., 1990: Seismic Noise Attenuation.
      Handbook of Geophysical Exploration, Pergamon Press, Oxford.
   Meyer, J. H. 1988: First Comparative Results of Integral and
      Instantaneous Polarization Attributes for Multicomponent Seismic
      Data. Institut Francais du Petrole.
   Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P.
      1996: Numerical Recipes in C - The Art of Scientific Computing.
      Cambridge University Press, Cambridge.
```

Samson, J. C., 1973: Description of the Polarisation States of Vector
Processes: Application to ULF Electromagnetic Fields.
Geophysical Journal vol. 34, p. 403-419.
Sheriff, R. E., 1991: Encyclopedic Dictionary of Exploration
Geophysics. 3rd ed., Society of Exploration Geophysicists, Tulsa.

Trace header fields accessed: ns, dt
Trace header fields modified: none

```
 SUADDNOISE - add noise to traces

 suaddnoise <stdin >stdout  sn=20  noise=gauss  seed=from_clock

 Required parameters:
  if any of f=f1,f2,... and amp=a1,a2,... are specified by the user
and if dt is not set in header, then dt is mandatory

 Optional parameters:
  sn=20 signal to noise ratio
  noise=gauss noise probability distribution
  =flat for uniform; default Gaussian
  seed=from_clock random number seed (integer)
f=f1,f2,... array of filter frequencies (as in sufilter)
amps=a1,a2,... array of filter amplitudes
  dt= (from header) time sampling interval (sec)
verbose=0 =1 for echoing useful information

  tmpdir=  if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

 Notes:
 Output = Signal +  scale * Noise

 scale = (1/sn) * (absmax_signal/sqrt(2))/sqrt(energy_per_sample)

 If the signal is already band-limited, f=f1,f2,... and amps=a1,a2,...
 can be used, as in sufilter, to bandlimit the noise traces to match
 the signal band prior to computing the scale defined above.

 Examples of noise bandlimiting:
 low freqency:    suaddnoise < data f=40,50 amps=1,0 | ...
 high freqency:   suaddnoise < data f=40,50 amps=0,1 | ...
 near monochromatic: suaddnoise < data f=30,40,50 amps=0,1,0 | ...
 with a notch:    suaddnoise < data f=30,40,50 amps=1,0,1 | ...
 bandlimited:     suaddnoise < data f=20,30,40,50 amps=0,1,1,0 | ...


 Credits:
CWP: Jack Cohen, Brian Sumner, Ken Larner
John Stockwell (fixed filtered noise option)
```

```
 Notes:
At S/N = 2, the strongest reflector is well delineated, so to
see something 1/nth as strong as this dominant reflector
requires S/N = 2*n.

 Trace header field accessed: ns
```

```
 SUHARLAN - signal-noise separation by the invertible linear
    transformation method of Harlan, 1984

   suharlan <infile >outfile  [optional parameters]

 Required Parameters:
 <none>

 Optional Parameters:
 FLAGS:
 niter=1 number of requested iterations
 anenv=1 =1 for positive analytic envelopes
=0 for no analytic envelopes (not recommended)
 scl=0 =1 to scale output traces (not recommended)
 plot=3 =0 for no plots. =1 for 1-D plots only
=2 for 2-D plots only. =3 for all plots
 norm=1 =0 not to normalize reliability values
 verbose=1 =0 not to print processing information
 rgt=2 =1 for uniform random generator
=2 for gaussian random generator
 sts=1 =0 for no smoothing (not recommended)

 tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

 General Parameters:
 dx=20 offset sampling interval (m)
 fx=0    offset on first trace (m)
 dt=0.004 time sampling interval (s)

 Tau-P Transform Parameters:
 gopt=1 =1 for parabolic transform. =2 for Foster/Mosher
=3 for linear. =4 for absolute value of linear
 pmin1=-400 minimum moveout at farthest offset for fwd transf(ms)
 pmax1=400 maximum moveout at farthest offset for fwd transf(ms)
 pmin2=pmin1 minimum moveout at farthest offset for inv transf(ms)
 pmax2=pmax1 maximum moveout at farthest offset for inv transf(ms)
 np=100 number of p-values for taup transform
 prewhite=0.01 prewhitening value (suggested between 0.1 and 0,01)
 offref=2000 reference offset for p-values (m)
 depthref=500 reference depth for Foster/Mosher taup (if gopt=4)
```

```
pmula=pmax1 maximum p-value preserved in the data (ms)
pmulb=pmax1 minimum p-value muted on the data (ms)
ninterp=0 number of traces to interpolate in input data

Extraction Parameters:
nintlh=50 number of intervals (bins) in histograms
sditer=5 number of steepest descent iterations to compute ps
c=0.04 maximum noise allowed in a sample of signal(%)
rel1=0.5 reliability value for first pass of the extraction
rel2=0.75 reliability value for second pass of the extraction

Smoothing Parameters: ",
r1=10 number of points for damped lsq vertical smoothing
r2=2 number of points for damped lsq horizontal smoothing


Output Files:
signal=out_signal  name of output file for extracted signal
noise=out_noise  name of output file for extracted noise


Notes:
The signal-noise separation algorithm was developed by Dr. Bill Harlan
in 1984. It can be used to separate events that can be focused by a
linear transformation (signal) from events that can't (noise). The
linear transform is whatever is well siuted for the application at
hand. Here, only the discrete Radon transform is used, so the program
is capable of separating events focused by that transform (linear,
parabolic or time-invariantly hyperbolic). Should other transform be
required, the changes to the program will be relatively
straightforward.

The reliability parameter is the most critical one to determine what
to extract as signal and what to reject as noise. It should be tested
for every dataset. The way to test it is to start with a small value,
say 0.1 or 0.01. If too much noise is present in the extracted noise,
it is too low. If too much signal was extracted, that is, part of the
signal was lost, it is too big. All other parameters have good default
values and should perhaps not be changed in a first encounter with the
program. The transform parameters are also critical. They should be
chosen such that no aliasing is present and such that the range of
interesting slopes is spanned by the transform but not much more. The
program suradon.c has more documentation on the transform paramters.
```

Credits:
Gabriel Alvarez CWP (1995)
Some subroutines are direct translations to C from Fortran versions
written by Dr. Bill Harlan (1984)

References:

Harlan, S., Claerbout, J., and Roca, F. (1984), Signal/noise
separation and velocity estimation, Geophysics, v. 49, no. 11,
p 1869-1880.

Harlan, S. (1988), Separation of signal and noise applied to
vertical seismic profiles, Geophysics, v. 53, no. 7,
p 932-946.

Alvarez, G. (1995), Comparison of moveout-based approaches to
ground roll and multiple suppression, MSc., Department of
Geophysics, Colorado School of Mines, (Chapter 3 deals
exclusively with this method).

```
 SUJITTER - Add random time shifts to seismic traces

     sujitter <stdin >stdout  [optional parameters]

 Required parameters:
none
 Optional Parameters:
  seed=from_clock     random number seed (integer)
min=1  minimum random time shift (samples)
max=1  maximum random time shift (samples)
pon=1  shift can be positive or negative
=0 shift is positive only
fldr=0  each trace has new shift
=1 new shift when fldr header field changes
 Notes:
 Useful for simulating random statics. See also:  suaddstatics


 Credits:
U of Houston: Chris Liner
UH:  Chris added fldr, min, pon options 12/10/08
```

```
 SUFLIP - flip a data set in various ways

 suflip <data1 >data2 flip=1 verbose=0

 Required parameters:
  none

 Optional parameters:
  flip=1  rotational sense of flip
  +1  = flip 90 deg clockwise
  -1  = flip 90 deg counter-clockwise
   0  = transpose data
   2  = flip right-to-left
   3  = flip top-to-bottom
  tmpdir=  if non-empty, use the value as a directory path
 prefix for storing temporary files; else if
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

  verbose=0 verbose = 1 echoes flip info

 NOTE:  tr.dt header field is lost if flip=-1,+1.  It can be
        reset using sushw.

 EXAMPLE PROCESSING SEQUENCES:
   1. suflip flip=-1 <data1 | sushw key=dt a=4000 >data2

   2. suflip flip=2 <data1 | suflip flip=2 >data1_again

   3. suflip tmpdir=/scratch <data1 | ...

 Caveat:  may fail on large files.

 Credits:
CWP: Chris Liner, Jack K. Cohen, John Stockwell

 Caveat:
right-left flip (flip = 2) and top-bottom flip (flip = 3)
don't require the matrix approach.  We sacrificed efficiency
for uniform coding.

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt, tracl
```

```
 SUFWMIX -  FX domain multidimensional Weighted Mix

sufwmix < stdin > stdout [optional parameters]

 Required parameters:
 key=key1,key2,.. Header words defining mixing dimension
 dx=d1,d2,.. Distance units for each header word
 Optional parameters:
 keyg=ep Header word indicating the start of gather
 vf=0 =1 Do a frequency dependent mix
 vmin=5000 Velocity of the reflection slope
than should not be attenuated
 Notes:
 Trace with the header word mark set to one will be
 the output trace
  (a work in progress)


 Credits:

  Potash Corporation: Balazs Nemeth, Saskatoon Saskatchewan CA,
   given to CWP in 2008
```

```
 SUMATH - do math operation on su data

 suop <stdin >stdout op=mult

 Required parameters:
none

 Optional parameter:
op=mult operation flag
--------- operations -----------------
add   : o = i + a    (o=out; i=in)
sub   : o = i - a
mult  : o = i * a
div   : o = i / a
pow   : o = i ^ a
spow  : o = sgn(i) * abs(i) ^ a
--------- operation parameter --------
a=1
copy=1 n>1 copy each trace n times

 Note:
 There is overlap between this program and "sugain" and
 "suop

 Credits:

U Arkansas: Chris Liner Jun 2013

 Notes:
```

```
 SUMIX - compute weighted moving average (trace MIX) on a panel
  of seismic data

 sumix <stdin >sdout
 mix=.6,1,1,1,.6 array of weights for weighted average


 Note:
 The number of values defined by mix=val1,val2,... determines the number
 of traces to be averaged, the values determine the weights.

 Examples:
 sumix <stdin mix=.6,1,1,1,.6 >sdout  (default) mix over 5 traces weights
 sumix <stdin mix=1,1,1 >sdout  simple 3 trace moving average


 Author:
CWP: John Stockwell, Oct 1995

 Trace header fields accessed: ns
```

```
 SUOP2 - do a binary operation on two data sets

 suop2 data1 data2 op=diff [trid=111] >stdout

 Required parameters:
  none

 Optional parameter:
  op=diff difference of two panels of seismic data
  =sum  sum of two panels of seismic data
  =prod product of two panels of seismic data
  =quo quotient of two panels of seismic data
  =ptdiff differences of a panel and single trace
  =ptsum sum of a panel and single trace
  =ptprod product of a panel and single trace
  =ptquo quotient of a panel and single trace
  =zipper do "zipper" merge of two panels
=zippol convert polar to rectangular and then zip

  trid=FUNPACKNYQ output trace identification code. (This option
  is active only for op=zipper)
For SU version 39-43 FUNPACNYQ=111
  (See: sukeyword trid     for current value)


 Note1: Output = data1 "op" data2 with the header of data1

 Note2: For convenience and backward compatibility, this
  program may be called without an op code as:

 For:  panel "op" panel  operations:
  susum  file1 file2 == suop2 file1 file2 op=sum
  sudiff file1 file2 == suop2 file1 file2 op=diff
  suprod file1 file2 == suop2 file1 file2 op=prod
  suquo  file1 file2 == suop2 file1 file2 op=quo

 For:  panel "op" trace  operations:
  suptsum  file1 file2 == suop2 file1 file2 op=ptsum
  suptdiff file1 file2 == suop2 file1 file2 op=ptdiff
  suptprod file1 file2 == suop2 file1 file2 op=ptprod
  suptquo  file1 file2 == suop2 file1 file2 op=ptquo

 Note3: If an explicit op code is used it must FOLLOW the
```

```
filenames.

 Note4: With op=quo and op=ptquo, divide by 0 is trapped and 0 is returned.

 Note5: Weighted operations can be specified by setting weighting
coefficients for the two datasets:
w1=1.0
w2=1.0

 Note6: With op=zipper, it is possible to set the output trace id code
  (See: sukeyword trid)
  This option processes the traces from two files interleaving its samples.
  Both files must have the same trace length and must not longer than
  SU_NFLTS/2  (as in SU 39-42  SU_NFLTS=32768).

  Being "tr1" a trace of data1 and "tr2" the corresponding trace of
  data2, The merged trace will be :

  tr[2*i]= tr1[i]
  tr[2*i+1] = tr2[i]

  The default value of output tr.trid is that used by sufft and suifft,
  which is the trace id reserved for the complex traces obtained through
  the application of sufft. See also, suamp.

 Note 7: op=zippol is like op=zipper, but the input samples are polar
(amplitude and phase) and are converted to cartesian (real, imag)
before interleaving them.

 For operations on non-SU binary files  use:farith

 Credits:
SEP: Shuki Ronen
CWP: Jack K. Cohen
CWP: John Stockwell, 1995, added panel op trace options.
: Fernando M. Roxo da Motta <petro@roxo.org> - added zipper op

 Notes:
If efficiency becomes important consider inverting main loop
and repeating operation code within the branches of the switch.
```

```
 SUOP - do unary arithmetic operation on segys

 suop <stdin >stdout op=abs

 Required parameters:
none

 Optional parameter:
op=abs operation flag
abs   : absolute value
avg   : remove average value
ssqrt : signed square root
sqr   : square
ssqr  : signed square
sgn   : signum function
exp   : exponentiate
sexp  : signed exponentiate
slog  : signed natural log
slog2 : signed log base 2
slog10: signed common log
cos   : cosine
sin   : sine
tan   : tangent
cosh  : hyperbolic cosine
sinh  : hyperbolic sine
tanh  : hyperbolic tangent
cnorm : norm complex samples by modulus ",
norm  : divide trace by Max. Value
db    : 20 * slog10 (data)
neg   : negate value
posonly : pass only positive values
negonly : pass only negative values
                      sum   : running sum trace integration
                      diff  : running diff trace differentiation
                      refl  : (v[i+1] - v[i])/(v[i+1] + v[i])
mod2pi : modulo 2 pi
inv   : inverse
rmsamp : rms amplitude
                      s2v   : sonic to velocity (ft/s) conversion
                      s2vm  : sonic to velocity (m/s) conversion
                      d2m   : density (g/cc) to metric (kg/m^3) conversion
                      drv2  : 2nd order vertical derivative
                      drv4  : 4th order vertical derivative
```

```
                        integ : top-down integration
                        spike : local extrema to spikes
                        saf   : spike and fill to next spike
                        freq  : local dominant freqeuncy
                        lnza  : preserve least non-zero amps
                        --------- window operations -----------
                        mean  : arithmetic mean
                        despike  : despiking based on median filter
                        std   : standard deviation
                        var   : variance
       nw=21            number of time samples in window
                        ---------------------------------------
nop   : no operation

 Note: Binary ops are provided by suop2.
 Operations inv, slog, slog2, and slog10 are "punctuated",",
 meaning that if, the input contains 0 values,
 0 values are returned. ",

 For file operations on non-SU format binary files use:  farith

 Credits:

 CWP: Shuki Ronen, Jack K Cohen (c. 1987)
  Toralf Foerster: norm and db operations, 10/95.
  Additions by Reg Beardsley, Chris Liner, and others.

 Notes:
If efficiency becomes important consider inverting main loop
      and repeating operation code within the branches of the switch.

Note on db option.  The following are equivalent:
... | sufft | suamp | suop op=norm | suop op=slog10 |\
sugain scale=20| suxgraph style=normal

... | sufft | suamp | suop op=db | suxgraph style=normal
```

```
 SUPERMUTE - permute or transpose a 3d datacube

 supermute <stdin >sdout

 Required parameters:
 none

 Optional parameters:
 n1=ns from header number of samples in the fast direction
 n2=ntr from header number of samples in the med direction ",
 n3=1 number of samples in the slow direction

 o1=1 new fast direction
 o2=2 new med direction
 o3=3 new slow direction

 d1=1 output interval in new fast direction
 d2=1 output interval in new med direction
 d3=1 output interval in new slow direction

 Notes:
 header fields d1 and d2 default to d1=1.0 and d2=1.0

 Credits:

VT: Matthias Imhof

 Trace header fields accessed: ns, ntr
 Trace header fields modified: d1=1, f1=1, d2=1, f2=1, ns, ntr
```

```
SUSIMAT - Correlation similarity matrix for two traces.
  Output is zero lag of cross-correlation of traces,
  or linear regression correlation coefficient.
  Horizontal axis is time of trace 1, vertical is trace 2.

susimmat <data12 sufile=data2 >dataout

Required parameters:
sufile= file containing SU traces to use as filter

Optional parameters:
panel=0 use only the first trace of sufile as filter
        =1 compute sim matrix trace by trace an entire
                        gather
mt=21 operator window length (odd integer)
eps=1e-3 stability parameter
taper=0 no taper to data fragments
 =1 apply exponential taper (1/e at ends)
method=1 use xcorrelation as similarity meausure
 =2 same but normalized by (rms+eps)
 =3 use linear regression CC
 =4 use mt-dimensional vector angle

EXAMPLE PROCESSING SEQUENCES:
  1. Look for all possible alignments of OBC P and Z data
   susimmat < P_Z.su sufile=OBC_P.su  mt=71 > P_Zxcor.su

Note:  xcor window is collapsed as needed to compute edge values
It is quietly assumed that the time sampling interval on the  single
trace
and the output traces is the same as that on the traces in the input
file.
The sufile may actually have more than one trace, but only the first
trace is used when panel=0. When panel=1 the number of traces in the
sufile MUST be  the same as the number of traces in the input.

Credits:
U Arkansas: Chris Liner (originally 11/2009 at U Houston)
     CWP: John Stockwell, some i/o modifications  Jul 2015

References:
Liner, Christopher L., and Robert G. Clapp. "Nonlinear pairwise
alignment of seismic traces." The Leading Edge 23.11 (2004): 1146-1150.
```

Liner, C. L, and R. G. Clapp (2004), Nonlinear pairwise alignment of seismic traces GEOPHYSICS, VOL. 69, NO. 6
(NOVEMBER-DECEMBER 2004); P. 15521559, 7 FIGS. 10.1190/1.1836828

Caveat:

Trace header fields accessed: ns, dt
Trace header fields modified: ns, dt, tracl

```
SUVCAT -  append one data set to another, with or without an  ",
          overlapping region.  Data in the overlap may be
          determined by one of several methods.

suvcat data1 data2 >stdout

Required parameters:
       none

Optional parameters for overlapping sections:

 taplen=0    Length of overlap in integer number of
                 samples.(Default is 0.)

 taptype=0    Type of taper or combination method in the
                 overlap region.  0 - average
                                  1 - maximum magnitude
                                  2 - cosine scaled
                                  3 - summation

Computational Notes:

This program vertically concatenates traces from data2 onto
the end of the corresponding traces in data1, with a region
of overlap, defined by taplen.  Data in the overlapping       ",
region is combined by the method specified by taptype. The
currently available methods are:

    taptype=0    output is assigned the unweighted average of
                 each point in the overlap
    taptype=1    output is assigned the value of the maximum
                 absolute value of each point in the overlap
    taptype=2    output is assigned the weighted average of
                 each point in the overlap, where the output
                 is the sum of cos(x) times the values on the
                 first section, and 1-cos(x) times the values
                 on the second section, where x is factor that
                 goes from 0 to pi/2 across the overlap. This
                 favors the upper section in the upper part of
                 the overlap, and favors the lower section in
                 the lower part of the overlap.
    taptype=3    output is assigned the sum of the amplitudes
                 at each sample in the overlap
```

```
 Credits:
CWP: Jack K. Cohen, Michel Dietrich (Original SUVCAT)
     Steven D. Sheaffer (modifed to include overlap)
 IfG Kiel: Thies Beilecke (added taptype=3)

 Trace header fields accessed:  ns
 Trace header fields modified:  ns
```

```
SUVLENGTH - Adjust variable length traces to common length

suvlength <vdata >stdout

Required parameters:
 none

Optional parameters:
 ns= output number of samples (default: 1st trace ns)
```

```
 SUFBPICKW - First break auto picker

   sufbpickw < infile >outfile

 Required parameters:
  none
 Optional parameters:
 keyg=ep
 window=.03 Length of forward and backward windows (s)
 test=1 Output the characteristic function
This can be used for testing window size
 Template
 o= offset...
 t= time pairs for defining first break search
window centre
 tdv=.05 Half length of the search window

 If the template is specified the maximum value of the
 characteristic function is searched in the window
  defined by the template only.Default is the whole trace.

  The time of the pick is stored in header word unscale


 segy data
segy *trp; /* SEGY trace array
segy trtp; /* SEGY trace
int
main( int argc, char *argv[] )
{

segy **rec_o;    /* trace header+data matrix

cwp_String keyg;
cwp_String typeg;
Value valg;

int first=0; /* true when we passed the first gather
int ng=0;
float dt;
int nt;
int ntr;
```

```
unsigned int np; /* Number of points in pick template
float *t=NULL; /* array defining pick template times
float *o=NULL; /* array defining pick template offsets

float window;
int iwindow;
int *itimes;
int *itimes2;
float *offset;
float tdv;
int itdv;
float *find;
int nowindow=0;
int test=0;

FILE *ttp;

initargs(argc, argv);
    requestdoc(1);

if (!countparval("t")) {
np=2;
nowindow=1;
} else {
np=countparval("t");
}

if(!nowindow) {
t  = ealloc1float(np);
o  = ealloc1float(np);

if( np == countparval("o")) {
getparfloat("t",t);
getparfloat("o",o);
} else {
err(" t and o has different number of elements\n");
}
}



ttp = efopen("test.su","w");
```

```
if (!getparstring("keyg", &keyg)) keyg ="ep";

if (!getparfloat("window",&window)) window =0.02;
if (!getparfloat("tdv",&tdv)) tdv = -1.0;
if (!getparint("test",&test)) test = 1;

        checkpars();
/* get information from the first header
rec_o = get_gather(&keyg,&typeg,&valg,&nt,&ntr,&dt,&first);

iwindow=NINT(window/dt);
if(tdv==-1.0) {
itdv=nt;
} else {
itdv=NINT(2.0*tdv/dt);
}

if(ntr==0) err("Can't get first record\n");
do {
ng++;

itimes = ealloc1int(ntr);
itimes2 = ealloc1int(ntr);
offset = ealloc1float(ntr);

/* Phase 1
/* Loop through traces
{ int itr,ifbt;
  int it;
  float fbt;
  float ampb;
  float ampf;
  float *wf,*wb;
  float *cf; /* Characteristic function


  cf=ealloc1float(nt);
  find=ealloc1float(nt);
  for(it=0;it<nt;it++)
   find[it]=(float)it;

  for(itr=0;itr<ntr;itr++) {
```

```
   memset( (void *) cf, (int) '\0', nt*FSIZE);

if(nowindow) {
ifbt=0;
} else {
   /* Linear inperpolation of estimtated fb time
offset[itr] =(*rec_o[itr]).offset;
   intlin(np,o,t,t[0],t[np-1],1,&offset[itr],&fbt);
   ifbt = NINT(fbt/dt);
}


wb = &(*rec_o[itr]).data[0];
wf = &(*rec_o[itr]).data[iwindow];
ampb = sasum(iwindow,wb,1)+FLT_EPSILON;
ampf = sasum(iwindow,wf,1)+FLT_EPSILON;
   for(it=iwindow;it<nt-iwindow-1;it++) {
cf[it] = ampf/ampb;
/* setup next window
ampb -= fabs((*rec_o[itr]).data[it-iwindow]);
ampf -= fabs((*rec_o[itr]).data[it]);
ampb += fabs((*rec_o[itr]).data[it]);
ampf += fabs((*rec_o[itr]).data[it+iwindow]);
}
/* Smooth the characteristic function
smooth_segmented_array(&find[iwindow],&cf[iwindow],nt-iwindow-1,iwindow,1,5);

/* find the maximum*/
it=MIN(MAX(ifbt-itdv/2,0),nt-1);


wb = &cf[it];

itimes[itr] = isamax(itdv,wb,1)+it;

/* Final check
if(itimes[itr] == ifbt-itdv/2 || itimes[itr] == ifbt+itdv/2)
itimes[itr]=0;

(*rec_o[itr]).unscale=dt*itimes[itr];

if(test)
memcpy( (void *) &(*rec_o[itr]).data[0], (const void *) cf, nt*FSIZE);
```

```
  }
  free1float(cf);
  free1float(find);
}

free1int(itimes);
free1int(itimes2);
free1float(offset);

rec_o = put_gather(rec_o,&nt,&ntr);
rec_o = get_gather(&keyg,&typeg,&valg,&nt,&ntr,&dt,&first);
} while(ntr);
return EXIT_SUCCESS;

}
```

```
 SUFNZERO - get Time of First Non-ZERO sample by trace

  sufnzero <stdin >stdout [optional parameters]

 Required parameters:
none

 Optional parameters:
mode=first    Output time of first non-zero sample
              =last for last non-zero sample
              =both for both first & last non-zero samples

min=0    Threshold value for considering as zero
Any abs(sample)<min is considered to be zero

key=key1,... Keyword(s) to print


 Credits:
      Geocon : Garry Perratt
based on surms by the same, itself based on sugain & sumax by:
CWP : John Stockwell
```

```
 SUPICKAMP - pick amplitudes within user defined and resampled window

   supickamp <stdin >stdout d2=  [optional parameters]

 Required parameters:
 d2=    sampling interval for slow dimension
(required if key-parameter not specified)
 Optional parameters:
 key=  Key header word specifying trace offset
  (alternatively, specify d2,x2beg)

 x_above= array of lateral position values
  (upper window corner)
 t_above= array of time values
  (upper window corner)

  ... or input via files:
 t_xabove= file containing time and lateral position values
  (upper window corner)
 t_xbelow= file containing time and lateral position values
  (lower window corner)
 wl=     window width if t_xbelow is not specified
(No windowing if not specified)

 dt_resamp=dt   resampling interval within pick window
(dt has to come from trace headers)
 tmin=0 minimum time in input trace
 x2beg=0 first lateral position
 format=ascii  write ascii data to stdout
 =binary for binary floats to stdout
 verbose=1  writes complete  pick information into outpar
=2 writes complete pick information into outpar
   in tab-delimited column format
 outpar=/dev/tty output parameter file; contains output
from verbose
 arg1=max output (first dimension) to stdout
 arg2=i2 output (second dimension) to stdout
(see notes for other options)
 Notes:

 Window can be defined using
 (1)   vectors x_above, t_above, [wl]
 (2)   file  t_xabove, [wl] or
```

```
(3)   files t_xabove, t_xbelow

files t_xabove, t_xbelow can be generated using xwigb's picking
algorithm. The lateral positions have to be monotonically increasing
or decreasing for both vector and file input.
verbose=1 or 2 writes min, max, abs[max], energy and associated times
tmin,tmax,tabs to outpar, together with global values. verbose=0
only outputs global values.
Acceptable arg-parameters for lateral positions are
(1) x2   (2) i2 = trace number

If key=keyword is set, then the values of x2 are taken from the header
field represented by the keyword (for example key=offset)
Type sukeyword -o   to see the complete list of SU keywords.



Credits:

CWP: Andreas Rueger July 06, 1996
MTU: David Forel,   Jan. 26, 2005  Add verbose=2 option
```

```
 SUCVS4FOWLER --compute constant velocity stacks for input to Fowler codes

 Required Parameter:
 ncdps= number of input cdp gathers
 Optional Parameters:
 vminstack=1500. minimum velocity panel in m/s to output
 nvstack=180 number of stacking velocity panels to compute
( Let offmax be the maximum offset, fmax be
the maximum freq to preserve, and tmute be
the starting mute time in sec on offmax, then
the recommended value for nvstack would be
nvstack = 4 +(offmax*offmax*fmax)/(0.6*vmin*vmin*tmute)
---you may want to make do with less---)
 lmute=24 length of mute taper in ms
 nonhyp=1 1 if do mute at 2*offset/vhyp to avoid
non-hyperbolic moveout, 0 otherwise
 vhyp=2500. velocity to use for non-hyperbolic moveout mute
 lbtaper=0 length of bottom taper in ms
 lstaper=0 length of side taper in traces
 dtout=1.5*dt output sample rate in s,
note: typically fmax=salias*0.5/dtout
 mxfold=120 maximum number of offsets/input cmp
 salias=0.8 fraction of output frequencies to force within sloth
antialias limit.  This controls muting by offset of
the input data prior to computing the cv stacks
for values of choose=1 or choose=2.
 Required trace header words on input are ns, dt, cdp, offset.



 Author:  (Visitor to CSM from Mobil): John E. Anderson, Spring 1994

References:

Fowler, P., 1988, Ph.D. Thesis, Stanford University.
Anderson, J.E., Alkhalifah, T., and Tsvankin, I., 1994, Fowler
DMO and time migration for transversely isotropic media,
1994 CWP project review
```

```
SUDIVSTACK -  Diversity Stacking using either average power or peak
              power within windows


Required parameters:
   none

Optional parameters:
   key=tracf        key header word to stack on
   winlen=0.064     window length in seconds.
                    typical choices: 0.064, 0.128, 0.256,
                                     0.512, 1.024, 2.048, 4.096
                    if not specified the entire trace is used

   peak=1           peak power option default is average power

Notes:
   Diversity stacking is a noise reduction technique used in the
   summation of duplicate data. Each trace is scaled by the inverse
   of its average power prior to stacking.  The composite trace is
   then renormalized by dividing by the sum of the scalers used.

   This program stacks adjacent traces having the same key header
   word, which can be specified by the key parameter. The default
   is "tracf" (trace number within field record).
   For more information on key header words, type "sukeyword -o".

Examples:
   For duplicate field shot records:
       susort < field.data tracf | sudivstack > stack.data
   For CDP ordered data:
       sudivstack < cdp.data key=cdp > stack.data



Author: Mary Palen-Murphy,
        Masters' candidate, Colorado School of Mines,
        Geophysics Department, 1994

Implementation of "key=" option: Nils Maercklin,
        GeoForschungsZentrum (GFZ) Potsdam, Germany, 2002.

References:
```

Embree, P.,1968, Diversity seismic record stacking method and system: U.S. patent 3,398,396.
Gimlin, D. R., and Smith, J. W., 1980, A comparison of seismic trace summing techniques: Geophysics, vol. 45, pages 1017-1041.

Trace header fields accessed: ns, dt, key=keyword
Trace header fields modified: tracl

```
 SUPWS - Phase stack or phase-weighted stack (PWS) of adjacent traces
 having the same key header word

 supws <stdin >stdout [optional parameters]

 Required parameters:
none

 Optional parameters:
key=cdp    key header word to stack on
pwr=1.0    raise phase stack to power pwr
dt=(from header)  time sampling intervall in seconds
sl=0.0 window length in seconds used for smoothing
of the phase stack (weights)
ps=0 0 = output is PWS, 1 = output is phase stack
verbose=0  1 = echo additional information

 Note:
Phase weighted stacking is a tool for efficient incoherent noise
reduction. An amplitude-unbiased coherency measure is designed
based on the instantaneous phase, which is used to weight the
samples of an ordinary, linear stack. The result is called the
phase-weighted stack (PWS) and is cleaned from incoherent noise.
PWS thus permits detection of weak but coherent arrivals.

The phase-stack (coherency measure) has values between 0 and 1.

If the stacking is over cdp and the PWS option is set, then the
offset header field is set to zero. Otherwise, output traces get
their headers from the first trace of each data ensemble to stack,
including the offset field. Use "sushw" afterwards, if this is
not acceptable.



 Author: Nils Maercklin,
 GeoForschungsZentrum (GFZ) Potsdam, Germany, 2001.
 E-mail: nils@gfz-potsdam.de

 References:
B. L. N. Kennett, 2000: Stacking three-component seismograms.
 Geophysical Journal International, vol. 141, p. 263-269.
M. Schimmel and H. Paulssen, 1997: Noise reduction and detection
```

of weak , coherent signals through phase-weighted stacks.
Geophysical Journal International, vol. 130, p. 497-505.
M. T. Taner, A. F. Koehler, and R. E. Sheriff, 1979: Complex
seismic trace analysis. Geophysics, vol. 44, p. 1041-1063.

Trace header fields accessed: ns
Trace header fields modified: nhs, offset

```
 SURECIP - sum opposing offsets in prepared data (see below)

 surecip <stdin >stdout

 Sum traces with equal positive and negative offsets (i.e. assume
 reciprocity holds).

 Usage:
suabshw <data >absdata
susort cdp offset <absdata | surecip >sumdata

 Note that this processing stream can be simply evoked by:

recip data sumdata


 Credits:
SEP: Shuki Ronen
CWP: Jack Cohen

 Caveat:
The assumption is that this operation is not a mainstay processing
item.  Hence the recommended implemention via the 'recip' shell
script.  If it becomes a mainstay, then a much faster code can
quickly drummed up by incorporating portions of suabshw and
susort.

 Trace header fields accessed: ns
 Trace header fields modified: nhs, tracl, sx, gx
```

```
 SUSTACK - stack adjacent traces having the same key header word

     sustack <stdin >stdout [Optional parameters]

 Required parameters:
  none

 Optional parameters:
  key=cdp header key word to stack on
  normpow=1.0 each sample is divided by the
normpow'th number of non-zero values
stacked (normpow=0 selects no division)
repeat=0 =1 repeats the stack trace nrepeat times
nrepeat=10 repeats stack trace nrepeat times in
           output file
  verbose=0 verbose = 1 echos information

 Notes:
 ------
 The offset field is set to zero on the output traces, unless
 the user is stacking with key=offset. In that case, the value
 of the offset field is left unchanged.

 Sushw can be used afterwards if this is not acceptable.

 For VSP users:
 The stack trace appears ten times in the output file when
 setting repeat=1 and nrepeat=10. Corridor stacking can be
 achieved by properly muting the upgoing data with SUMUTE
 before stacking.


 Credits:
SEP: Einar Kjartansson
CWP: Jack K. Cohen, Dave Hale
CENPET: Werner M. Heigl - added repeat trace functionality

 Note:
The "valxxx" subroutines are in su/lib/valpkge.c.  In particular,
      "valcmp" shares the annoying attribute of "strcmp" that
if (valcmp(type, val, valnew) {
...
}
```

will be performed when val and valnew are different.

Trace header fields accessed: ns
Trace header fields modified: nhs, tracl, offset

```
 SUADDSTATICS - ADD random STATICS on seismic data

 suaddstatics required parameters [optional parameters] > stdout

 Required parameters:
 shift= the static shift will be generated
randomly in the interval [+shift,-shif] (ms)
 sources= number of source locations
 receivers= number of receiver locations
 cmps= number of common mid point locations
 maxfold= maximum fold of input data
 datafile= name and COMPLETE path of the input file

 Optional parameters:
 dt=tr.dt time sampling interval (ms)
 seed=getpid()  seed for random number generator
 verbose=0 =1 print useful information

 Notes:
 Input data should be sorted into cdp gathers.

 SUADDSTATICS applies static time shifts in a surface consistent way on
 seismic data sets. SUADDSTATICS writes the static time shifts in the
 header field TSTAT. To perform the actual shifts the user should use
 the program SUSTATIC after SUADDSTATICS. SUADDSTATICS outputs the
 corrupted data set to stdout.

 Header field used by SUADDSTATICS: cdp, sx, gx, tstat, dt.



 Credits: CWP Wences Gouveia, 11/07/94,  Colorado School of Mines
```

```
 SURANDSTAT - Add RANDom time shifts STATIC errors to seismic traces

     surandstat <stdin >stdout  [optional parameters]

 Required parameters:
none
 Optional Parameters:
  seed=from_clock     random number seed (integer)
max=tr.dt  maximum random time shift (ms)
scale=1.0 scale factor for shifts


 Credits:
U Houston: Chris Liner c. 2009
```

```
 SURESSTAT - Surface consistent source and receiver statics calculation

   suresstat fn=  [optional parameters]

 Required parameters:
 fn= seismic file
 ssol= output file source statics
 rsol= output file receiver statics

 Optional parameters:
 ntpick=50  maximum static shift (samples)
 niter=5  number of iterations
 imax=100000  largest shot (fldr),reciver(tracf) or cmp(cdp) number
 sub=0  subtract super trace 1 from super trace 2 (=1)
  sub=0 strongly biases static to a value of 0
 mode=0  use global maximum in cross-correllation window
=1 choose the peak perc=percent smaller than the global max.
 perc=10.  percent of global max (used only for mode=1)
 verbose=0  print diagnostic output (verbose=1)

 Notes:
 Estimates surface-consistent source and receiver statics, meaning that
 there is one static correction value estimated for each shot and receiver
 position.

 The method employed here is based on the method of Ronen and Claerbout:
 Geophysics 50, 2759-2767 (1985).

 The input data are NMO-corrected and sorted into shot gathers ( fldr)
 Rreceiver id position should be stored in headerword tracf
 The output files are binary files containing the source and receiver
 statics, as a function of shot number (trace header fldr) and
 receiver station number (trace header tracf).

 The code builds a supertrace1 and supertrace2, which are subsequently
 cross-correllated. The program then picks the time lag associated with
 the largest peak in the cross-correllation according to two possible
 criteria set by the parameter "mode". If mode=0, the maximum of the
 cross-correllation window is chosen. If mode=1, the program will pick
 a peak which is up to perc=percent smaller than the global maximum, but
 closer to zero lag than the global maximum. (Choosing mode=0 is
 recommended.)
```

```
 The geometry can be irregular: the program simply computes a static
 correction for each shot record (fldr=1 to fldr=nshot), with any missing
 shots being assigned a static of 0.  A static correction for each
 receiver station (tracf=1 to tracf=nr) is calculated, with missing
 receivers again assigned a static of 0.                                ",
 To window out the most cohherent region use suwind tmin= tmax= and
 save the result into a file. This will reduce the amount of time
 the code will spent on scaning the file,since the file is much smaller
 The ntpick parameter sets the maximum allowable shift desired (in
   samples NOT time).

 To apply the static corrections, use sustatic with hdrs=3

 Reference:

  Ronen, J. and Claerbout, J., 1985, Surface-consistent residual statics
      estimation  by stack-power maximization: Geophysics, vol. 50,
      2759-2767.

 Credits:
CWP: Timo Tjan, 4 October 1994

      rewritten by Thomas Pratt, USGS, Feb. 2000.
      Modified by A. Bitri, BRGM-France, Apr. 2015
 Trace header fields accessed: ns, dt, tracf, fldr, cdp
```

```
 SUSTATICB - Elevation static corrections, apply corrections from
      headers or from a source and receiver statics file
      (beta submitted by J. W. Neese)

     sustaticB <stdin >stdout  [optional parameters]

 Required parameters:
none
 Optional Parameters:
v0=v1 or user-defined or from header, weathering velocity
v1=user-defined or from header, subweathering velocity
hdrs=0 =1 to read statics from headers
  =2 to read statics from files
=3 to read from output files of suresstat
sign=1 apply static correction (add tstat values)
=-1 apply negative of tstat values
 Options when hdrs=2 and hdrs=3:
sou_file= input file for source statics (ms)
rec_file= input file for receiver statics (ms)
ns=240  (2)number of sources; (3) max fldr
nr=335  number of receivers
no=96  number of offsets

 Notes:
 For hdrs=1, statics calculation is not performed, statics correction
 is applied to the data by reading statics (in ms) from the header.

 For hdrs=0, field statics are calculated, and
  input field sut is assumed measured in ms.
  output field sstat = 10^scalel*(sdel - selev + sdepth)/swevel
  output field gstat = sstat - sut/1000.
  output field tstat = sstat + gstat + 10^scalel*(selev - gelev)/wevel

 For hdrs=2, statics are surface consistently obtained from the
 statics files. The geometry should be regular.
 The source- and receiver-statics files should be unformated C binary
 floats and contain the statics (in ms) as a function of surface location.

 For hdrs=3, statics are read from the output files of suresstat, with
 the same options as hdrs=2 (but use no=max traces per shot and assume
 that ns=max fldr number and nr=max receiver number).
 For each shot number (trace header fldr) and each receiver number
 (trace header tracf) the program will look up the appropriate static
```

```
 correction.  The geometry need not be regular as each trace is treated
 independently.

 Caveat:  The static shifts are computed with the assumption that the
 desired datum is sea level (elevation=0). You may need to shift the
 selev and gelev header values via  suchw.
 Example: subtracting min(selev,gelev)=25094431

 suchw < CR290.su key1=selev,gelev key2=selev,gelev key3=selev,gelev \\
            a=-25094431,-25094431 b=1,1 c=0,0 > CR290datum.su

 Credits:
CWP: Jamie Burns

CWP: Modified by Mohammed Alfaraj, 11/10/1992, for reading
     statics from headers and including sign (+-) option

      CWP: Modified by Timo Tjan, 29 June 1995, to include input of
           source and receiver statics from files.

modified by Thomas Pratt, USGS, Feb, 2000 to read statics from
       the output files of suresstat

 Logic changed by JWN to fix options hdrs=2,3 ???

 Trace header fields accessed:  ns, dt, delrt, gelev, selev,
sdepth, gdel, sdel, swevel, sut, scalel, fldr, tracf
 Trace header fields modified:  sstat, gstat, tstat
```

```
 SUSTATICRRS - Elevation STATIC corrections, apply corrections from
      headers or from a source and receiver statics file,
      includes application of Residual Refraction Statics

     sustaticrrs <stdin >stdout  [optional parameters]

 Required parameters:
none
 Optional Parameters:
v0=v1 or user-defined or from header, weathering velocity
v1=user-defined or from header, subweathering velocity
hdrs=0 =1 to read statics from headers
  =2 to read statics from files
sign=1 =-1 to subtract statics from traces(up shift)
 Options when hdrs=2:
sou_file= input file for source statics (ms)
rec_file= input file for receiver statics (ms)
ns=240  number of sources
nr=335  number of receivers
no=96  number of offsets

 Options when hdrs=3:
       blvl_file=              base of the near-surface model file (sampled
                                  at CMP locations)
       refr_file=              horizontal reference datum file (sampled at
                                  CMP locations)
       nsamp=                  number of midpoints on line
       fx=                     first x location in velocity model
       dx=                     midpoint interval
       V_r=                    replacement velocity
       mx=                     number of velocity model samples in
                                  lateral direction
       mz=                     number of velocity model samples in
                                  vertical direction
       dzv=                    velocity model depth interval
       vfile=                  near-surface velocity model

 Options when hdrs=4:
       nsamp=                  number of midpoints on line
       fx=                     first x location in velocity model      ",
       dx=                     midpoint interval                       ",

 Options when hdrs=5:
```

```
      none

 Notes:
 For hdrs=1, statics calculation is not performed, statics correction
 is applied to the data by reading statics (in ms) from the header.

 For hdrs=0, field statics are calculated, and
  input field sut is assumed measured in ms.
  output field sstat = 10^scalel*(sdel - selev + sdepth)/swevel
  output field gstat = sstat - sut/1000.
  output field tstat = sstat + gstat + 10^scalel*(selev - gelev)/wevel

 For hdrs=2, statics are surface consistently obtained from the
 statics files. The geometry should be regular.
 The source- and receiver-statics files should be unformated C binary
 floats and contain the statics (in ms) as a function of surface location.

 For hdrs=3, residual refraction statics and average refraction statics
 are computed.  For hdrs=4, residual refraction statics are applied,
 and for hdrs=5, average refraction statics are applied (Cox, 1999).
 These three options are coupled in many data processing sequences:
 before stack residual and average refraction statics are computed but
 only residual refractions statics are applied, and after stack average
 refraction statics are applied.  Refraction statics are often split
 like this to avoid biasing stacking velocities.  The files blvl_file
 and refr_file are the base of the velocity model defined in vfile and
 the final reference datum, as described by Cox (1999), respectively.
 Residual refraction statics are stored in the header field gstat, and
 the average statics are stored in the header field tstat.  V_r is the
 replacement velocity as described by Cox (1999).  The velocity file,
 vfile, is designed to work with a horizontal upper surface defined in
 refr_file.  If the survey has irregular topography, the horizontal
 upper surface should be above the highest topographic point on the
 line, and the velocity between this horizontal surface and topography
 should be some very large value, such as 999999999, so that the
 traveltimes through that region are inconsequential.

 Credits:
CWP: Jamie Burns

CWP: Modified by Mohammed Alfaraj, 11/10/1992, for reading
     statics from headers and including sign (+-) option
```

```
      CWP: Modified by Timo Tjan, 29 June 1995, to include input of
           source and receiver statics from files.

      CWP: Modified by Chris Robinson, 11/2000, to include the splitting
           of refraction statics into residuals and averages

 Trace header fields accessed:  ns, dt, delrt, gelev, selev,
sdepth, gdel, sdel, swevel, sut, scalel
 Trace header fields modified:  sstat, gstat, tstat

 References:

 Cox, M., 1999, Static corrections for seismic reflection surveys:
    Soc. Expl. Geophys.
```

```
 SUSTATIC - Elevation static corrections, apply corrections from
      headers or from a source and receiver statics file

     sustatic <stdin >stdout  [optional parameters]

 Required parameters:
none
 Optional Parameters:
v0=v1 or user-defined or from header, weathering velocity
v1=user-defined or from header, subweathering velocity
hdrs=0 =1 to read statics from headers
  =2 to read statics from files
=3 to read from output files of suresstat
sign=1 apply static correction (add tstat values)
=-1 apply negative of tstat values
 Options when hdrs=2 and hdrs=3:
sou_file= input file for source statics (ms)
rec_file= input file for receiver statics (ms)
ns=240  number of souces
nr=335  number of receivers
no=96  number of offsets

 Notes:
 For hdrs=1, statics calculation is not performed, statics correction
 is applied to the data by reading statics (in ms) from the header.

 For hdrs=0, field statics are calculated, and
  input field sut is assumed measured in ms.
  output field sstat = 10^scalel*(sdel - selev + sdepth)/swevel
  output field gstat = sstat - sut/1000.
  output field tstat = sstat + gstat + 10^scalel*(selev - gelev)/wevel

 For hdrs=2, statics are surface consistently obtained from the
 statics files. The geometry should be regular.
 The source- and receiver-statics files should be unformated C binary
 floats and contain the statics (in ms) as a function of surface location.

 For hdrs=3, statics are read from the output files of suresstat, with
 the same options as hdrs=2 (but use no=max traces per shot and assume
 that ns=max shot number and nr=max receiver number).
 For each shot number (trace header fldr) and each receiver number
 (trace header tracf) the program will look up the appropriate static
 correction.  The geometry need not be regular as each trace is treated
```

```
 independently.

 Caveat:  The static shifts are computed with the assumption that the
 desired datum is sea level (elevation=0). You may need to shift the
 selev and gelev header values via  suchw.
 Example: subtracting min(selev,gelev)=25094431

 suchw < CR290.su key1=selev,gelev key2=selev,gelev key3=selev,gelev \\
            a=-25094431,-25094431 b=1,1 c=0,0 > CR290datum.su

 Credits:
CWP: Jamie Burns

CWP: Modified by Mohammed Alfaraj, 11/10/1992, for reading
     statics from headers and including sign (+-) option

      CWP: Modified by Timo Tjan, 29 June 1995, to include input of
           source and receiver statics from files.

modified by Thomas Pratt, USGS, Feb, 2000 to read statics from
       the output files of suresstat

 Trace header fields accessed:  ns, dt, delrt, gelev, selev,
sdepth, gdel, sdel, swevel, sut, scalel, fldr, tracf
 Trace header fields modified:  sstat, gstat, tstat
```

```
 SUILOG -- time axis inverse log-stretch of seismic traces

 suilog nt= ntmin=  <stdin >stdout

 Required parameters:
  nt=  nt output from sulog prog
  ntmin=  ntmin output from sulog prog
  dt=  dt output from sulog prog
 Optional parameters:
  none

 NOTE:  Parameters needed by suilog to reconstruct the
original data may be input via a parameter file.

 EXAMPLE PROCESSING SEQUENCE:
  sulog outpar=logpar <data1 >data2
  suilog par=logpar <data2 >data3

  where logpar is the parameter file



 Credits:
CWP: Shuki Ronen, Chris Liner

 Caveats:
  amplitudes are not well preserved

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt
```

```
 SULOG -- time axis log-stretch of seismic traces

 sulog [optional parameters] <stdin >stdout

 Required parameters:
none

 Optional parameters:
ntmin= .1*nt minimum time sample of interest
outpar=/dev/tty output parameter file, contains:
number of samples (nt=)
minimum time sample (ntmin=)
output number of samples (ntau=)
m=3 length of stretched data
is set according to
ntau = nextpow(m*nt)
ntau= pow of 2 override for length of stretched
data (useful for padding zeros
to avoid aliasing)

 NOTES:
ntmin is required to avoid taking log of zero and to
keep number of outsamples (ntau) from becoming enormous.
        Data above ntmin is zeroed out.

The output parameters will be needed by suilog to
reconstruct the original data.

 EXAMPLE PROCESSING SEQUENCE:
sulog outpar=logpar <data1 >data2
suilog par=logpar <data2 >data3


 Credits:
CWP: Shuki, Chris

 Caveats:
  Amplitudes are not well preserved.

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt
```

```
 SUNMO_a - NMO for an arbitrary velocity function of time and CDP with
experimental Anisotropy options
  sunmo <stdin >stdout [optional parameters]

 Optional Parameters:
 tnmo=0,... NMO times corresponding to velocities in vnmo
 vnmo=1500,.. NMO velocities corresponding to times in tnmo
 anis1=0 two anisotropy coefficients making up quartic term
 anis2=0 in traveltime curve, corresponding to times in tnmo
 cdp= CDPs for which vnmo & tnmo are specified (see Notes)
 smute=1.5 samples with NMO stretch exceeding smute are zeroed
 lmute=25 length (in samples) of linear ramp for stretch mute
 sscale=1 =1 to divide output samples by NMO stretch factor
 invert=0 =1 to perform (approximate) inverse NMO
 upward=0 =1 to scan upward to find first sample to kill

 Notes:
 For constant-velocity NMO, specify only one vnmo=constant and omit tnmo.

 NMO interpolation error is less than 1% for frequencies less than 60% of
 the Nyquist frequency.

 Exact inverse NMO is impossible, particularly for early times at large
 offsets and for frequencies near Nyquist with large interpolation errors.

 The "offset" header field must be set.
 Use suazimuth to set offset header field when sx,sy,gx,gy are all
 nonzero.

 For NMO with a velocity function of time only, specify the arrays
   vnmo=v1,v2,... tnmo=t1,t2,...
 where v1 is the velocity at time t1, v2 is the velocity at time t2, ...
 The times specified in the tnmo array must be monotonically increasing.
 Linear interpolation and constant extrapolation of the specified velocities
 is used to compute the velocities at times not specified.
 The same holds for the anisotropy coefficients as a function of time only.

 For NMO with a velocity function of time and CDP, specify the array
   cdp=cdp1,cdp2,...
 and, for each CDP specified, specify the vnmo and tnmo arrays as described
 above. The first (vnmo,tnmo) pair corresponds to the first cdp, and so on.
 Linear interpolation and constant extrapolation of 1/velocity^2 is used
 to compute velocities at CDPs not specified.
```

```
 Anisotropy option:
 Caveat, this is an experimental option,

 The anisotropy coefficients anis1, anis2 permit non-hyperbolicity due
 to layering, mode conversion, or anisotropy. Default is isotropic NMO.

 The same holds for the anisotropy coefficients as a function of time and
 CDP.

 Moveout is defined by

   1  anis1
  --- x^2 + ------------- x^4.
  v^2      1 + anis2 x^2

 Note: In general, the user should set the cdp parameter.  The default is
to use tr.cdp from the first trace and assume only one cdp.
 Caveat:
 Nmo cannot handle negative moveout as in triplication caused by
 anisotropy. But negative moveout happens necessarily for negative anis1 at
 sufficiently large offsets. Then the error-negative moveout- is printed.
 Check anis1. An error (anis2 too small) is also printed if the
 denominator of the quartic term becomes negative. Check anis2. These errors
 are prompted even if they occur in traces which would not survive the
 NMO-stretch threshold. Chop off enough far-offset traces (e.g. with suwind)
 if anis1, anis2 are fine for near-offset traces.


 Credits:
SEP: Shuki, Chuck Sword
CWP: Shuki, Jack, Dave Hale, Bjoern Rommel
      Modified: 08/08/98 - Carlos E. Theodoro - option for lateral offset
      Modified: 07/11/02 - Sang-yong Suh -
  added "upward" option to handle decreasing velocity function.
      CWP: Sept 2010: John Stockwell
  replaced Carlos Theodoro's fix
  and added the instruction in the selfdoc to use suazimuth to set
    offset so that it accounts for lateral offset
      note that by the segy standard "scalel" does not scale the offset
      field
 Technical Reference:
The Common Depth Point Stack
```

William A. Schneider
Proc. IEEE, v. 72, n. 10, p. 1238-1254
1984

Trace header fields accessed: ns, dt, delrt, offset, cdp, scalel

```
 SUNMO - NMO for an arbitrary velocity function of time and CDP

  sunmo <stdin >stdout [optional parameters]

 Optional Parameters:
 tnmo=0,... NMO times corresponding to velocities in vnmo
 vnmo=1500,... NMO velocities corresponding to times in tnmo
 cdp= CDPs for which vnmo & tnmo are specified (see Notes)
 smute=1.5 samples with NMO stretch exceeding smute are zeroed
 lmute=25 length (in samples) of linear ramp for stretch mute
 sscale=1 =1 to divide output samples by NMO stretch factor
 invert=0 =1 to perform (approximate) inverse NMO
 upward=0 =1 to scan upward to find first sample to kill
 voutfile= if set, interplolated velocity function v[cdp][t] is
output to named file.
 Notes:
 For constant-velocity NMO, specify only one vnmo=constant and omit tnmo.

 NMO interpolation error is less than 1% for frequencies less than 60% of
 the Nyquist frequency.

 Exact inverse NMO is impossible, particularly for early times at large
 offsets and for frequencies near Nyquist with large interpolation errors.

 The "offset" header field must be set.
 Use suazimuth to set offset header field when sx,sy,gx,gy are all
 nonzero.

 For NMO with a velocity function of time only, specify the arrays
   vnmo=v1,v2,... tnmo=t1,t2,...
 where v1 is the velocity at time t1, v2 is the velocity at time t2, ...
 The times specified in the tnmo array must be monotonically increasing.
 Linear interpolation and constant extrapolation of the specified velocities
 is used to compute the velocities at times not specified.
 The same holds for the anisotropy coefficients as a function of time only.

 For NMO with a velocity function of time and CDP, specify the array
   cdp=cdp1,cdp2,...
 and, for each CDP specified, specify the vnmo and tnmo arrays as described
 above. The first (vnmo,tnmo) pair corresponds to the first cdp, and so on.
 Linear interpolation and constant extrapolation of 1/velocity^2 is used
 to compute velocities at CDPs not specified.
```

```
 The format of the output interpolated velocity file is unformatted C floats
 with vout[cdp][t], with time as the fast dimension and may be used as an
 input velocity file for further processing.

 Note that this version of sunmo does not attempt to deal with anisotropy.
 The version of sunmo with experimental anisotropy support is "sunmo_a


 Credits:
SEP: Shuki Ronen, Chuck Sword
CWP: Shuki Ronen, Jack, Dave Hale, Bjoern Rommel
      Modified: 08/08/98 - Carlos E. Theodoro - option for lateral offset
      Modified: 07/11/02 - Sang-yong Suh -
  added "upward" option to handle decreasing velocity function.
      CWP: Sept 2010: John Stockwell
  1. replaced Carlos Theodoro's fix
  2. added  the instruction in the selfdoc to use suazimuth to set
      offset so that it accounts for lateral offset.
        3. removed  Bjoren Rommel's anisotropy stuff. sunmo_a is the
           version with the anisotropy parameters left in.
        4. note that scalel does not scale the offset field in
           the segy standard.
 Technical Reference:
The Common Depth Point Stack
William A. Schneider
Proc. IEEE, v. 72, n. 10, p. 1238-1254
1984

 Trace header fields accessed: ns, dt, delrt, offset, cdp, scalel
```

```
 SUREDUCE - convert traces to display in reduced time ",

 sureduce <stdin >stdout rv=

 Required parameters:
dt=tr.dt if not set in header, dt is mandatory

 Optional parameters:
rv=8.0 reducing velocity in km/sec ",

 Note: Useful for plotting refraction seismic data.
 To remove reduction, do:
 suflip < reduceddata.su flip=3 | sureduce rv=RV > flip.su
 suflip < flip.su flip=3 > unreduceddata.su

 Trace header fields accessed: dt, ns, offset
 Trace header fields modified: none


 Author: UC Davis: Mike Begnaud  March 1995


 Trace header fields accessed: ns, dt, offset
```

```
SURESAMP - Resample in time

suresamp <stdin >stdout  [optional parameters]

Required parameters:
    none

Optional Parameters:
   nt=tr.ns    number of time samples on output
   dt=         time sampling interval on output
               default is:
               tr.dt/10^6     seismic data
               tr.d1          non-seismic data
   tmin=       time of first sample in output
               default is:
               tr.delrt/10^3  seismic data
               tr.f1          non-seismic data
   rf=         resampling factor;
               if defined, set nt=nt_in*rf and dt=dt_in/rf
   verbose=0   =1 for advisory messages


Example 1: (assume original data had dt=.004 nt=256)
   sufilter <data f=40,50 amps=1.,0. |
   suresamp nt=128 dt=.008 | ...
Using the resampling factor rf, this example translates to:
   sufilter <data f=40,50 amps=1.,0. | suresamp rf=0.5 | ...

Note the typical anti-alias filtering before sub-sampling!

Example 2: (assume original data had dt=.004 nt=256)
   suresamp <data nt=512 dt=.002 | ...
or use:
   suresamp <data rf=2 | ...

Example 3: (assume original data had d1=.1524 nt=8192)
   sufilter <data f=0,1,3,3.28 amps=1,1,1,0 |
   suresamp <data nt=4096 dt=.3048 | ...

Example 4: (assume original data had d1=.5 nt=4096)
   suresamp <data nt=8192 dt=.25 | ...
```

```
Credits:
   CWP: Dave (resamp algorithm), Jack (SU adaptation)
   CENPET: Werner M. Heigl - modified for well log support
   RISSC: Nils Maercklin 2006 - minor fixes, added rf option

Algorithm:
   Resampling is done via 8-coefficient sinc-interpolation.
   See "$CWPROOT/src/cwp/lib/intsinc8.c" for technical details.

Trace header fields accessed:  ns, dt, delrt, d1, f1, trid
Trace header fields modified:  ns, dt, delrt (only when set tmin)
                               d1, f1 (only when set tmin)
```

```
SUSHIFT - shifted/windowed traces in time

sushift <stdin >stdout [tmin= ] [tmax= ]

tmin= min time to pass
tmax= max time to pass
dt=                    sample rate in microseconds
fill=0.0               value to place in padded samples

(defaults for tmin and tmax are calculated from the first trace.
verbose= 1 echos parameters to stdout

Background :
tmin and tmax must be given in seconds

In the high resolution single channel seismic profiling the sample
interval is short, the shot rate and the number of samples are high.
To reduce the file size the delrt time is changed during a profiling
trip. To process and display a seismic section a constant delrt is
needed. This program does this job.

The SEG-Y header variable delrt (delay in ms) is a short integer.
That's why in the example shown below delrt is rounded to 123 !

  ... | sushift tmin=0.1234 tmax=0.2234 | ...

The dt= and fill= options are intended for manipulating velocity
volumes in trace format.  In particular models which were hung
from the water bottom when created & which then need to have the
water layer added.



Author:

Toralf Foerster
Institut fuer Ostseeforschung Warnemuende
Sektion Marine Geologie
Seestrasse 15
D-18119 Rostock, Germany

Trace header fields accessed: ns, delrt
Trace header fields modified: ns, delrt
```

```
SUTAUPNMO - NMO for an arbitrary velocity function of tau and CDP

 sutaupnmo <stdin >stdout [optional parameters]

Optional Parameters:
tnmo=0,... NMO times corresponding to velocities in vnmo
vnmo=1500,... NMO velocities corresponding to times in tnmo
cdp= CDPs for which vnmo & tnmo are specified (see Notes)
smute=1.5 samples with NMO stretch exceeding smute are zeroed
lmute=25 length (in samples) of linear ramp for stretch mute
sscale=1 =1 to divide output samples by NMO stretch factor

Notes:

For constant-velocity NMO, specify only one vnmo=constant and omit tnmo.

For NMO with a velocity function of tau only, specify the arrays
  vnmo=v1,v2,... tnmo=t1,t2,...
where v1 is the velocity at tau t1, v2 is the velocity at tau t2, ...
The taus specified in the tnmo array must be monotonically increasing.
Linear interpolation and constant extrapolation of the specified velocities
is used to compute the velocities at taus not specified.

For NMO with a velocity function of tau and CDP, specify the array
  cdp=cdp1,cdp2,...
and, for each CDP specified, specify the vnmo and tnmo arrays as described
above. The first (vnmo,tnmo) pair corresponds to the first cdp, and so on.
Linear interpolation and constant extrapolation of velocity^2 is used
to compute velocities at CDPs not specified.

Moveout is defined by

 tau^2 + tau^2.p^2.vel^2

 Note: In general, the user should set the cdp parameter.  The default is
to use tr.cdp from the first trace and assume only one cdp.
 Caveat:
 Taunmo should handle triplication

 NMO interpolation error is less than 1% for frequencies less than 60% of
 the Nyquist frequency.

 Exact inverse NMO is not implemented, nor has anisotropy
```

```
 Example implementation:
   sutaup dx=25 option=2 pmin=0 pmax=0.0007025 < cmpgather.su |
   supef minlag=0.2 maxlag=0.8 |
   sutaupnmo tnmo=0.5,2,4 vnmo=1500,2000,3200 smute=1.5 |
   sumute key=tracr mode=1 ntaper=20 xmute=1,30,40,50,85,15
 tmute=7.8,7.8,4.5,3.5,2.0,0.35 |
   sustack key=cdp | ... [...]


 Credits:
 Durham, Richard Hobbs modified from SUNMO credited below
SEP: Shuki Ronen, Chuck Sword
CWP: Shuki Ronen, Jack K. Cohen , Dave Hale

 Technical Reference:
van der Baan papers in geophysics (2002 & 2004)

 Trace header fields accessed: ns, dt, delrt, offset, cdp, sy
```

```
 SUTSQ -- time axis time-squared stretch of seismic traces

 sutsq [optional parameters] <stdin >stdout

 Required parameters:
none

 Optional parameters:
       tmin= .1*nt*dt  minimum time sample of interest
                        (only needed for forward transform)
       dt= .004        output sample rate
                        (only needed for inverse transform)
       flag= 1         1=forward transform: time to time squared
                      -1=inverse transform: time squared to time

 Note: The output of the forward transform always starts with
 time squared equal to zero.  'tmin' is used to avoid aliasing
 the early times.



 Caveats:
  Amplitudes are not well preserved.

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt
```

```
 SUTTOZ - resample from time to depth

 suttoz <stdin >stdout [optional parms]

 Optional Parameters:
 nz=1+(nt-1)*dt*vmax/(2.0*dz)   number of depth samples in output
 dz=vmin*dt/2 depth sampling interval (defaults avoids aliasing)
 fz=v(ft)*ft/2 first depth sample
 t=0.0,... times corresponding to interval velocities in v
 v=1500.0,... interval velocities corresponding to times in v
 vfile=   binary (non-ascii) file containing velocities v(t)
 verbose=0 >0 to print depth sampling information

 Notes:
 Default value of nz set to avoid aliasing
 The t and v arrays specify an interval velocity function of time.

 Note that t and v are given  as arrays of floats separated by commas,
 for example:
 t=0.0,0.01,.2,... v=1500.0,1720.0,1833.5,... with the number of t values
 equaling the number of v values. The velocities are linearly interpolated
 to make a continuous, piecewise linear v(t) profile.

 Linear interpolation and constant extrapolation is used to determine
 interval velocities at times not specified.  Values specified in t
 must increase monotonically.

 Alternatively, interval velocities may be stored in a binary file
 containing one velocity for every time sample.  If vfile is specified,
 then the t and v arrays are ignored.

 see selfdoc of suztot  for depth to time conversion

 Trace header fields accessed:  ns, dt, and delrt
 Trace header fields modified:  trid, ns, d1, and f1


 Credits:
CWP: Dave Hale c. 1992
```

```
 SUZTOT - resample from depth to time

 suztot <stdin >stdout [optional parms]

 Optional Parameters:
 nt=1+(nz-1)*2.0*dz/(vmax*dt)    number of time samples in output
 dt=2*dz/vmin time sampling interval (defaults avoids aliasing)
 ft=2*fz/v(fz) first time sample
 z=0.0,... depths corresponding to interval velocities in v
 v=1500.0,... interval velocities corresponding to depths in v
 vfile= binary (non-ascii) file containing velocities v(z)
 verbose=0 >0 to print depth sampling information

 Notes:
 Default value of nt set to avoid aliasing
 The z and v arrays specify an interval velocity function of depth.

 Note that z and v are given  as arrays of floats separated by commas,
 for example:
 z=0.0,100,200,... v=1500.0,1720.0,1833.5,... with the number of z values
 equaling the number of v values. The velocities are linearly interpolated
 to produce a piecewise linear v(z) profile. This fact must be taken into
 account when attempting to use this program as the inverse of suttoz.

 Linear interpolation and constant extrapolation is used to determine
 interval velocities at times not specified.  Values specified in z
 must increase monotonically.

 Alternatively, interval velocities may be stored in a binary file
 containing one velocity for every time sample.  If vfile is specified,
 then the z and v arrays are ignored.

 see the selfdoc of   suttoz  for time to depth conversion
 Trace header fields accessed:  ns, dt, and delrt
 Trace header fields modified:  trid, ns, d1, and f1


 Credits:
CWP: John Stockwell, 2005,
            based on suttoz.c written by Dave Hale c. 1992
```

```
SUGET  - Connect SU program to file descriptor for input stream.

   suget fd=$1 | next_su_module

This program is for interfacing " outside processing systems
with SU. Typically, an outside system would execute the su command file
and a file descriptor would be passed by an outside system to
the su command file so that output data from the outside system
could be piped into the su programs executing inside the command file.

Example:    suget fd=$1 | next_su_module

     fd=-1        file_descriptor_for_input_stream
     verbose=0    minimal listing
                  =1  asks for message with each trace processed.


Author: John Anderson (visiting scholar from Mobil) July 1994
```

```
SUPUT - Connect SU program to file descriptor for output stream.

      su_module | suput fp=$1

This program is for interfacing " outside processing systems
with SU. Typically, the outside system would execute the SU command file.
The outside system provides the file descriptor it would like to read
from to the command file to be an argument for suput.

Example: su_module | suput fp=$1

      fd=-1       file_descriptor_for_output_stream_from_su
      verbose=0   minimal listing
                  =1  asks for message with each trace processed.


Author: John Anderson (visiting scholar from Mobil) July 1994
```

```
SUADDEVENT - add a linear or hyperbolic moveout event to seismic data

suaddevent <stdin >stdout [optional parameters]

Required parameters:
      none

Optional parameters:
    type=nmo    =lmo for linear event
    t0=1.0      zero-offset intercept time IN SECONDS
    vel=3000.   moveout velocity in m/s
    amp=1.      amplitude
    dt=  must provide if 0 in headers (seconds)

Typical usage:
    sunull nt=500 dt=0.004 ntr=100 | sushw key=offset a=-1000 b=20 \\
    | suaddevent v=1000 t0=0.05 type=lmo | suaddevent v=1800 t0=0.8 \
    | sufilter f=8,12,75,90 | suxwigb clip=1 &



Credits:
     Gary Billings, Talisman Energy, May 1996, Apr 2000, June 2001

 Note:  code is inefficient in that to add a single "spike", with sinc
interpolation, an entire trace is generated and added to
the input trace.  In fact, only a few points needed be created
and added, but the current coding avoids the bookkeeping re
which are the relevant points!
```

```
 SUDGWAVEFORM - make Gaussian derivative waveform in SU format

  sudgwaveform >stdout  [optional parameters]


 Optional parameters:
 n=2     order of derivative (n>=1)
 fpeak=35 peak frequency
 nfpeak=n*n max. frequency = nfpeak * fpeak
 nt=128 length of waveform
 shift=0 additional time shift in s (used for plotting)
 sign=1 use =-1 to change sign
 verbose=0 =0 don't display diagnostic messages
               =1 display diagnostic messages
 Notes:
 This code computes a waveform that is the n-th order derivative of a
 Gaussian. The variance of the Gaussian is specified through its peak
 frequency, i.e. the frequency at which the amplitude spectrum of the
 Gaussian has a maximum. nfpeak is used to compute maximum frequency,
 which in turn is used to compute the sampling interval. Increasing
 nfpeak gives smoother plots. In order to have a (pseudo-) causal
 pulse, the program computes a time shift equal to sqrt(n)/fpeak. An
 additional shift can be applied with the parameter shift. A positive
 value shifts the waveform to the right.

 Examples:
 2-loop Ricker: dgwaveform n=1 >ricker2.su
 3-loop Ricker: dgwaveform n=2 >ricker3.su
 Sonic transducer pulse: dgwaveform n=10 fpeak=300 >sonic.su

 To display use suxgraph. For example:
 dgwaveform n=10 fpeak=300 | suxgraph style=normal &

 For other seismic waveforms, please use "suwaveform".


 Credits:

Werner M. Heigl, February 2007

 This copyright covers parts that are not part of the original
 CWP/SU: Seismic Un*x codes called by this program:
```

```
 Copyright (c) 2007 by the Society of Exploration Geophysicists.
 For more information, go to http://software.seg.org/2007/0004 .
 You must read and accept usage terms at:
 http://software.seg.org/disclaimer.txt before use.

 Revision history:
 Original SEG version by Werner M. Heigl, Apache E&P Technology,
 February 2007.

 Jan 2010 - subroutines deriv_n_gauss and hermite_n_polynomial moved
 to libcwp.a
/
```

```
 SUEA2DF - SU version of (an)elastic anisotropic 2D finite difference
forward modeling, 4th order in space

 suea2df > outfile c11file= c55file  [optional parameters]

 Required Parameters:
 c11file=c11_file c11 voigt elasticity parameter filename
 c55file=c55_file c55 voigt elasticity parameter filename

 Optional Parameters:
 rhofile=rho_file density filename
(if rhofile is not set, rho=1000 is assumed)
 Anisotropy parameters:
 aniso=0    =1 - include anisotropy parameters
 mode=0 =0 output particle velocity, =1 output stresses
(snapshots only)

 ... the next 3 parameters become active only when aniso=1....
 c13file=c13_file c13 voigt elasticity parameter filename
 c33file=c33_file c33 voigt elasticity parameter filename
 c15file=c15_file c15 voigt elasticity parameter filename
 c35file=c35_file c35 voigt elasticity parameter filename

 Attenuation parameters:
 qsw=0  switch to include attenuation =1 - include
 ... the next parameter becomes active only when qsw=1....
 qfile=Q_file   Q parameter filename

 dt=0.001 time sampling interval (s)
 ft=0.0  first time (s)
 lt=1.0  last time (s)

 nx=200 number of values in slow (x-direction)
 dx=10.0   spatial sampling interval (m) x-coor
 fx=-1000 first x coor (m)

 nz=100 number of values in fast (z)-dimension
 dz=dx spatial sampling interval (m) z-coor
 fz=0 firstz coor (m)

 Source parameters:
 sx=0 source x position (m)
 sz=500 source location (m)
```

```
 stype='p'source type
  p: P-wave
  v: velocity
 pw: P plane-wave
 sang=0 for stype='pw': plane wave angle
 wtype='dg'wave type
   dg: Gaussian derivative
   ga: Gaussian
   ri: Ricker
   sp: spike, sp2: double spike
 ts=0.05 source duration (s)
 favg=50 source average frequency

 Attenuation parameters:
 qsw=0   switch to include attenuation =1 - include

 Boundary condition parameters:
 bc=10,10,10,10  Top,left,bottom,right boundary condition
  =0 none
  =1 symmetry
  =2 free surface (top only)
  >2 absorbing (value indicates width of absorbing
layer
 bc_a=0.95; bc initial taper value for absorbing boundary
 bc_r=0.; bc exponential factor for absorbing boundary
  variables are scaled by bc_a*pow(i,-bc_r)

 Optional output parameters:
 sofile= name of source file
 snfile= name of file containing for snapshots
 snaptime= times of snapshots i.e. snaptime=0.1,0.2,0.3

 vsx= x coordinate of vertical line of seismograms
 hsz= z coordinate of horizontal line of seismograms
 vrslfile="vsp.su" output file for vertical line of seismograms[nz][nt]
 hsfile="hs.su"  output file for horizontal line of seismograms[nx][nt]
 tsw=0   switch to use shear stress only in non-fluid
media - may help reduce dispersion tsw=1. If
tsw=0 then standard calculation
 verbose=0 =1 to print progress on screen

 Notes:
 1) The outfile contains information generated by the input parameters,
```

```
   such as memory allocation, stability, etc. If your input file does
   not work, check this file first.

 The model is specified as binary files of stiffness parameters and
 densities. These may be created any way the user desires. The program
 unif2 or makevel may be used to generate densities, and the program
 unif2aniso may be used to generate the stiffnesses. You will need to
 transpose these files (stiffnesses and densities), as the input
 format for suea2df assumes that the fast dimesion is the horizontal or
 the x-dimension. You may do this via

  transp n1=nz < c11_file > transp_c11_file

 If aniso=1 then the program will expect the additional stiffnesss files.

 If qsw=1 unif2anis can be used to generate the Q values on a grid
 These value also need to be transposed, as with the stiffnesses.

Output files (always generated)
hsfile
vrslfile
hsfile.chd - header for hsfile
vrslfile.chd - header for vrslfile
hsfile.mod - model file

 Output files (if requested)
sofile - ascii source file
snfile  - su format snapshots file

 Caveat:
 A common error in using this program is to compute stiffnesses with
 a specified density, but forget to specify this density as the rhofile.


 Credits: UU GEOPHY Chris Juhlin 15 May 1999
 Copyright (c) Uppsala University, 1998.
 All rights reserved.
 Parts of program use Seismic Unix Package - CSM
 Changes - C. Juhlin

 1. Fixed upgrading of stresses. There was an error in the coding for
 the Tzz term, c15 was being used instead of c35. This only caused
 problems for dipping anisotropic layers
```

```
 2. Added some header information for hutput of snapshots.

 3. 2001-01-30: Added option to set absorbing bc constants bc_a and bc_r

 4. 2001-02-23: Corrected bug in outputting model boundaries to standard
 output in
 routine get_econst

 5. 2001-04-26: Added option for updating velocities to only use
 shear stress if material is non-fluid, this appears to reduce dispersion at
 near grazing angles for fluid-solid boundary. Set tsw=1 to invoke

 6. 2001-05-14: Changed loop in free-surface boundary condition for velocties
 Thanks goes to Mike Holzrichter for pointing out this problem and the wrong
 scaling factor in the updating.

 7. 2001-05-16: Changed set_layers function to avoid negative indexing.
 Thanks goes to Mike Holzrichter for pointing out this problem

 8. 2001-05-17: Modified make_seis to take into account VSP geometry
 correctly and not store unnecessary data.

 9. 2001-08-21: Fixed set_layers so mode fills properly in depth. Earlier
 versions were accessing incorrect array locations at last defined depth.

 10. 2003-04-21: Fixed boundary conditions.

 11. 2003-05-02: Extended the model area by half the grid spacing on the RHS.
 This makes the model area symmetric allowing a plane wave source to be
 introduced into the model (stype=pw). The w, txx and tzz grids contain now
 one more column than the u and txz grids.

 12. 2003-05-02: Added routines to allow plane waves to be introduced at a
 specified angle (sang) into the model with functions add_pw_source_V and
 add_pw_source_S.

 13. 15 Oct 2005 -- tossed all the model building stuff. Read models
from binary files made by  unif2aniso (CWP:John Stockwell)

 14. 25 Feb 2008 -- Fixed attenutation option (qsw=1) so that Q values are
from binary files made by makevel or similar program
```

```
 15. 1 April 2010 -- Changed free surface velocity BC back to original.
Someone had changed the scaling factor from 2.0 to 0.5 in fs4v_bc_top

 Algorithm based on Juhlin (1995, Geophys. Prosp.)
and Levander (1988, Geophysics)
 Attenuation included as in Graves (1996, BSSA)
```

```
 SUFCTANISMOD - Flux-Corrected Transport correction applied to the 2D
  elastic wave equation for finite difference modeling in
  anisotropic media

 sufctanismod > outfile [optional parameters]
outfile is the final wavefield snapshot x-component
x-component of wavefield snapshot is in snapshotx.data
y-component of wavefield snapshot is in snapshoty.data
z-component of wavefield snapshot is in snapshotz.data

 Optional Output Files:
 reflxfile= reflection seismogram file name for x-component
no output produced if no name specified
 reflyfile= reflection seismogram file name for y-component
no output produced if no name specified
 reflzfile= reflection seismogram file name for z-component
no output produced if no name specified
 vspxfile= VSP seismogram file name for x-component
no output produced if no name specified
 vspyfile= VSP seismogram file name for y-component
no output produced if no name specified
 vspzfile= VSP seismogram file name for z-component
no output produced if no name specified

 suhead=1      To get SU-header output seismograms (else suhead=0)

 New parameter:

 Optional Parameters:
 mt=1          number of time steps per output snapshot
 dofct=1  1 do the FCT correction
0 do not do the FCT correction
 FCT Related parameters:
 eta0=0.03 diffusion coefficient
typical values ranging from 0.008 to 0.06
about 0.03 for the second-order method
about 0.012 for the fourth-order method
 eta=0.04 anti-diffusion coefficient
typical values ranging from 0.008 to 0.06
about 0.04 for the second-order method
about 0.015 for the fourth-order method
 fctxbeg=0  x coordinate to begin applying the FCT correction
 fctzbeg=0  z coordinate to begin applying the FCT correction
```

```
 fctxend=nx  x coordinate to stop applying the FCT correction
 fctzend=nz  z coordinate to stop applying the FCT correction

 deta0dx=0.0 gradient of eta0 in x-direction  d(eta0)/dx
 deta0dz=0.0 gradient of eta0 in z-direction  d(eta0)/dz
 detadx=0.0 gradient of eta in x-direction   d(eta)/dx
 detadz=0.0 gradient of eta in z-direction   d(eta)/dz

 General Parameters:
 order=2 2 second-order finite-difference
4 fourth-order finite-difference

 nt=200        number of time steps
 dt=0.004 time step

 nx=100  number of grid points in x-direction
 nz=100  number of grid points in z-direction

 dx=0.02 spatial step in x-direction
 dz=0.02 spatial step in z-direction

 sx=nx/2 source x-coordinate (in gridpoints)
 sz=nz/2 source z-coordinate (in gridpoints)

 fpeak=20 peak frequency of the wavelet

 receiverdepth=sz  depth of horizontal receivers (in gridpoints)
 vspnx=sx x grid loc of vsp

 verbose=0     silent operation
=1 for diagnostic messages, =2 for more

 wavelet=1 1 AKB wavelet
  2 Ricker wavelet
3 impulse
4 unity

 isurf=2 1 absorbing surface condition
2 free surface condition
3 zero surface condition

 source=1 1 point source
  2 sources are located on a given refelector          ",
```

```
(two horizontal and one dipping reflectors)
  3 sources are located on a given dipping refelector     ",

 sfile=  the name of input source file, if no name specified then
use default source location. (source=1 or 2)

 Density and Elastic Parameters:
 dfile=  the name of input density file,
               if no name specified then
assume a linear density profile with ...
 rho00=2.0 density at (0, 0)
 drhodx=0.0 density gradient in x-direction  d(rho)/dx
 drhodz=0.0 density gradient in z-direction  d(rho)/dz

 afile=  name of input elastic param.  (c11) aa file, if no name
specified then, assume a linear profile with ...
 aa00=2.0 elastic parameter at (0, 0)
 daadx=0.0 parameter gradient in x-direction  d(aa)/dx
 daadz=0.0 parameter gradient in z-direction  d(aa)/dz

 cfile=  name of input elastic param. (c33)  cc file, if no name
specified then, assume a linear profile with ...
 cc00=2.0 elastic parameter at (0, 0)
 dccdx=0.0 parameter gradient in x-direction  d(cc)/dx
 dccdz=0.0 parameter gradient in z-direction  d(cc)/dz

 ffile=  name of input elastic param.  (c13) ff file, if no name
specified then, assume a linear profile with ...
 ff00=2.0 elastic parameter at (0, 0)
 dffdx=0.0 parameter gradient in x-direction  d(ff)/dx
 dffdz=0.0 parameter gradient in z-direction  d(ff)/dz

 lfile=  name of input elastic param.  (c44) ll file, if no name
specified then, assume a linear profile with ...
 ll00=2.0 elastic parameter at (0, 0)
 dlldx=0.0 parameter gradient in x-direction  d(ll)/dx
 dlldz=0.0 parameter gradient in z-direction  d(ll)/dz

 nfile=  name of input elastic param. (c66)  nn file, if no name
specified then, assume a linear profile with ...
 nn00=2.0 elastic parameter at (0, 0)
 dnndx=0.0 parameter gradient in x-direction  d(nn)/dx
 dnndz=0.0 parameter gradient in z-direction  d(nn)/dz
```

```
 Optimizations:
 The moving boundary option permits the user to restrict the computations
 of the wavefield to be confined to a specific range of spatial coordinates.
 The boundary of this restricted area moves with the wavefield
 movebc=0 0 do not use moving boundary optimization
1 use moving boundaries



 Author: Tong Fei,Center for Wave Phenomena,
Colorado School of Mines, Dec 1993
 Some additional features by: Stig-Kyrre Foss, CWP
Colorado School of Mines, Oct 2001
 New features (Oct 2001):
 - setting receiver depth
 - outputfiles with SU-headers
 - additional commentary
 Modifications (Mar 2010) Chris Liner, U Houston
 - added snapshot mt param to parallel sufdmod2d functionality
 - added verbose and some basic info echos
 - error check that source loc is in grid
 - dropped mbx1 etc from selfdoc (they were internally computed)
 - moved default receiver depth to source depth
 - added vspnx to selfdoc and moved default vspnx to source x
 - changed sy in selfdoc to sz (typo)
 - fixed bug in vsp file(s) allocation: was [nt,nx] now is [nt,nz]



Notes:
This program performs seismic modeling for elastic anisotropic
media with vertical axis of symmetry.
The finite-difference method with the FCT correction is used.

Stability condition: vmax*dt /(sqrt(2)*min(dx,dz)) < 1

Two major stages are used in the algorithm:
(1) conventional finite-difference wave extrapolation
(2) followed by an FCT correction

References:
The detailed algorithm description is given in the article
```

"Elimination of dispersion in finite-difference modeling
and migration" in CWP-137, project review, page 155-174.

Original reference to the FCT method:
Boris, J., and Book, D., 1973, Flux-corrected transport. I.
SHASTA, a fluid transport algorithm that works:
Journal of Computational Physics, vol. 11, p. 38-69.

/

```
 SUFDMOD1 - Finite difference modelling (1-D 1rst order) for the
 acoustic wave equation "

 sufdmod1 <vfile >sfile nz= tmax= sz= [optional parameters]

 Required parameters :
 <vfile or vfile= binary file containing velocities[nz]
 >sfile or sfile= SU file containing seimogram[nt]
 nz=  number of z samples
 tmax= maximum propagation time
 sz=  z coordinate of source

 Optional parameters :
 dz=1    z sampling interval
 fz=0.0  first depth sample
 rz=1    coordinate of receiver
 sz=1    coordinate of source
 dfile=  binary input file containing density[nz]
 wfile=  output file for wave field (snapshots in a SU trace panel)
 abs=0,1 absorbing conditions on top and bottom
 styp=0  source type (0: gauss, 1: ricker 1, 2: ricker 2)
 freq=15.0 approximate source center frequency (Hz)
 nt=1+tmax/dt   number od time samples (dt determined for numerical
 stability)
 zt=1    trace undersampling factor for trace and snapshots
 zd=1    depth undersampling factor for snapshots
 press=1 to record the pressure field; 0 records the particle
velocity
 verbose=0 =1 for diagnostic messages

 Notes :
  This program uses a first order explicit velocity/pressure  finite
  difference equation.
  The source function is applied on the pressure component.
  If no density file is given, constant density is assumed
  Wavefield  can be easily viewed with suximage, user must provide f2=0
  to the ximage program in order to  get correct time labelling
  Seismic trace is shifted in order to get a zero phase source
  Source begins and stop when it's amplitude is 10^-4 its maximum
  Time and depth undersampling only modify the output trace and snapshots.
  These parameters are useful for keeping snapshot file small and
  the number of samples under SU_NFLTS.
```

```
NULL  };

float source (float t, int styp, float dt, float dz, float t0, float alpha);

int main (int argc, char **argv)
{
float *rv; /* array of rock velocity from cfile
float *rd; /* array of rock density from dfile on p knots
float *rd1_5; /* array of rock density from dfile on v knots
float *p; /* pressure
float *v; /* particle velocity
float tmax, dt, t0; /* maximum time , time step,   */
/* time delay for near causal source */
float vmax; /* maximum rock velocity */
int verbose; /* is verbose? */
int nz, nt; /* number of z samples, time samples */
float fz, dz; /* first sample depth spatial depth */
float sz; /* source coordinate */
int abs[2]; /* array of absorbing conditions */
int isz; /* source location index */
float rz; /* receiver depth
int irz; /* zcoordinate (in samples) of the source
int iz, it, itsis; /* counter
int ies; /* end of source index
int press; /* to choose  between pressure or particle
/* velocity
float t; /* time
int td=1, zd=1; /* time and depth decimation
segy snapsh, sismo; /* recording of the seismic field,
/*  seismogram
char *dfile=""; /* density file name
char *wfile=""; /* seismogram file name
char *velfile="";   /* velocity file name
char *sfile=""; /* velocity file name
float freq=0.0; /* source center freq
float alpha=0.0; /* source exp */
float epst0=0.0; /* source first amp ratio
int styp; /* source type

FILE *seisfp=stdout; /* pointer to seismic trace output file
FILE *wavefp=NULL; /* pointer to wave field output file
FILE *velocityfp=stdin; /* pointer to input velocity file
FILE *densityfp=NULL; /* pointer to input density file
```

```
/* hook up getpar to handle the parameters
initargs (argc, argv);
requestdoc(0);

/* verbose
if (!getparint ("verbose",&verbose)) verbose=0;

/* get required parameters
if (!getparint ("nz",&nz)) err("must specify nz ! ");
if (verbose) warn("nz= %d", nz);

if (!getparfloat ("tmax",&tmax)) err("must specify tmax ! ");
if (verbose) warn("tmax= %f", tmax);

if (!getparfloat("sz", &sz)) err ("must specify sz ! ");
if (verbose) warn("sz= %f", sz);



/* get optional parameters
if (!getparint ("nt", &nt)) nt=0;
if (verbose) warn("nt= %d", nt);
if (!getparint ("styp", &styp)) styp=0;
if (verbose) warn("styp= %d ", styp);
if (!getparfloat ("dz", &dz)) dz=1;
if (!getparfloat ("fz", &fz)) fz=0.0;

/* source coordinates to samples
isz=NINT((sz-fz)/dz);
if (verbose) warn( "source on knot number %d ", isz);


if (!getparfloat("rz", &rz)) rz=0.0;
irz = NINT ((rz-fz)/dz);
if (verbose) warn("receiver depth : %f on knot # %d ", rz, irz);

if (!getparfloat("freq", &freq)) freq=15.0;
if (verbose) warn("frequency : %f  Hz",freq);

getparstring ("velfile", &velfile);
if (verbose) {
```

```
if (*velfile != '\0' ) warn("Velocity file : %s ",velfile);
else warn("Velocity file supplied via stdin");
}

getparstring ("sfile", &sfile);
if (verbose) {
if (*sfile != '\0' ) warn("Output trace file : %s ",sfile);
else warn("Output trace via stdout");
}

getparstring ("dfile", &dfile);
if (verbose) {
if (*dfile != '\0' ) warn("Density file : %s ",dfile);
else warn("No density file supplied ");
}

getparstring ("wfile", &wfile);
if (verbose) {
if (*wfile != '\0' ) warn("Wave file : %s ",wfile);
else warn("No wave file requested ");
}

if ( NINT((float) nz/((float) zd)) + 1  >SU_NFLTS) {
warn ("Too many depth points : impossible to output wave field. Increase zd ?");
*wfile='\0';
}


/* get absorbing conditions
if (!getparint("abs",abs)) {  abs[0]=0; abs[1]=1;  }
if (verbose) {
if (abs[0]==1) warn("absorbing condition on top ");
if (abs[1]==1) warn("absorbing condition on bottom ");
}
/* get decimation coefficients
if (!getparint("td",&td)) td=1 ;
if (verbose) warn("time decimation ccoefficent: %d ",td);
if (!getparint("zd",&zd)) zd=1 ;
if (verbose) warn("depth decimation ccoefficent: %d ",zd);

/* choose pressure or particle velocity
if (!getparint("press", &press)) press=1 ;
if ((press != 0) && (press != 1)) err ("press must equal 0 or 1");
```

```
if (verbose) {
if (press==1) warn( "program will output pressure values");
else if (press==0) warn( "program will output particle velocity values");
}


/* allocate space
p=alloc1float(nz);
v=alloc1float(nz);
rv=alloc1float(nz);
rd=alloc1float(nz);
rd1_5=alloc1float(nz);

/* read velocity file
if (*velfile != '\0' ) {
if ((velocityfp=fopen(velfile,"r"))=='\0') err("cannot open velfile=%s ",velfile);
}
if (efread (rv, sizeof(float), nz, velocityfp)!=nz)
   err("cannot read %d velocity values ", nz);

/* read density file  and linearly inderpolate on corrrect location
if (*dfile != '\0') {
if ((densityfp=fopen(dfile,"r"))=='\0') err("cannot open dfile=%s ",dfile);
if (fread(rd,sizeof(float), nz, densityfp)!=nz) err("error reading dfile %s",dfile);
fclose(densityfp);
}
else for (iz=0; iz<nz; iz++) rd[iz]=2500;
for (iz=0; iz<nz-1; iz++) rd1_5[iz]=(rd[iz]+rd[iz+1])/2;
rd1_5[nz-1]=rd[nz-1];

/* time step computation
vmax=0;
for (iz=0; iz<nz; iz++) if (rv[iz]>vmax) vmax=rv[iz];if (verbose) warn( "vmax= %f ", vm
dt=dz/1.414/vmax/2; if (verbose) warn( "time step dt= %f ", dt);

/* maximum number of iterations
if (nt==0) nt=1+tmax/dt;
if (verbose) warn( "number of time steps nt= %d ", nt);
if (NINT( (float) nt/((float)td))+1>SU_NFLTS) err("too many time steps. Increase td ?")

/* source parameter computation
   alpha=2*9.8696*freq*freq;
```

```
/* time shift to get a t0 centered source

if ((styp==0) || (styp == 2)) epst0=fabs(source (0, styp, dt, dz, 0, alpha) / 1e4);
else if (styp==1) epst0=fabs(source (1/sqrt(2*alpha), styp, dt, dz, 0, alpha)) / 1e4;
if (verbose) warn( "epst0 = %f ", epst0);

t=tmax+dt;
do t=t-dt; while (fabs(source(t, styp, dt, dz, 0, alpha))<epst0);
t0=t;
ies=2*t/dt;

if (verbose) warn("time shift t0 = %f s", t0);

 array initialization
for (iz=0; iz<nz; iz++) {  v[iz]=0; p[iz]=0;  }

if (*wfile != '\0') {
wavefp=fopen (wfile,"w");
snapsh.d1=dz*zd; snapsh.f1=fz ; snapsh.ns=nz/zd+1; snapsh.d2=dt*td; snapsh.f2=0;
/* snapsh.f2=0 is useless since 0 is the "no value" code for SU headers
}
/* propagation computation
itsis=0;
for (it=0; it<=nt; it++) {
t=it*dt;
if (abs[0]==1) p[0]=(p[0]*(1-rv[0]*dt/dz)+2*rd[0]*rv[0]*rv[0]*dt/dz*v[0])/(1+rv[0]*dt/d
else p[0]=0;
for (iz=1; iz<nz; iz++) p[iz]=p[iz]+rd[iz]*rv[iz]*rv[iz]*dt/dz*(v[iz]-v[iz-1]);
if (abs[1]!=1) p[nz-1]=0;
if (it<ies) {
p[isz]=p[isz]+source(t, styp, dt, dz, t0, alpha);
}

for (iz=0; iz<nz-1; iz++) v[iz]=v[iz]+dt/rd1_5[iz]/dz*(p[iz+1]-p[iz]);

if (abs[1] != 1) v[nz-1]=0;
else
v[nz-1]=((rd1_5[nz-1]*dz-dt*rd[nz-1]*rv[nz-1])*v[nz-1]-2*dt*p[nz-1])/(rd1_5[nz-1]*dz+dt

  if (it % td == 0) {
   if (press==1)
sismo.data[itsis]=p[irz];
   else
```

```
sismo.data[itsis]=v[irz];
   itsis++;
}

if ((*wfile!='\0') && (it % td == 0)) {
if (press==1)
for (iz=0; iz<nz/zd; ++iz) snapsh.data[iz]=p[iz*zd];
else
for (iz=0; iz<nz/zd; ++iz) snapsh.data[iz]=v[iz*zd];

    fputtr(wavefp, &snapsh);
}

}

if (*wfile!='\0') fclose (wavefp);

sismo.dt=td*dt*1e6;
sismo.ns=itsis;
sismo.delrt=-t0*1000;
sismo.trid=TREAL;
sismo.tracl=1;

if (*sfile != '\0') seisfp=efopen(sfile,"w");
fputtr (seisfp, &sismo);


return(CWP_Exit());
}

float source (float t, int styp, float dt, float dz, float t0, float alpha)
{
float x=0.0, sou=0.0;
x=-alpha*(t-t0)*(t-t0);
if (x>-40) {
 if (styp==0) sou=exp(x);
  else if (styp==1) sou=-2*alpha*(t-t0)*exp(x);
  else if (styp==2) sou=2*alpha*(1+2*x)*exp(x);
}
else sou=0;
return sou/dz*dt*1e8;
}
```

```
 SUFDMOD2_PML - Finite-Difference MODeling (2nd order) for acoustic wave
    equation with PML absorbing boundary conditions.
 Caveat: experimental PML absorbing boundary condition version,
may be buggy!

 sufdmod2_pml <vfile >wfile nx= nz= tmax= xs= zs= [optional parameters]

 Required Parameters:
 <vfile file containing velocity[nx][nz]
 >wfile file containing waves[nx][nz] for time steps
 nx= number of x samples (2nd dimension)
 nz= number of z samples (1st dimension)
 xs= x coordinates of source
 zs= z coordinates of source
 tmax= maximum time

 Optional Parameters:
 nt=1+tmax/dt number of time samples (dt determined for stability)
 mt=1 number of time steps (dt) per output time step

 dx=1.0 x sampling interval
 fx=0.0 first x sample
 dz=1.0 z sampling interval
 fz=0.0 first z sample

 fmax = vmin/(10.0*h) maximum frequency in source wavelet
 fpeak=0.5*fmax peak frequency in ricker wavelet

 dfile= input file containing density[nx][nz]
 vsx= x coordinate of vertical line of seismograms
 hsz= z coordinate of horizontal line of seismograms
 vsfile= output file for vertical line of seismograms[nz][nt]
 hsfile= output file for horizontal line of seismograms[nx][nt]
 ssfile= output file for source point seismograms[nt]
 verbose=0 =1 for diagnostic messages, =2 for more

 abs=1,1,1,1 Absorbing boundary conditions on top,left,bottom,right
  sides of the model.
  =0,1,1,1 for free surface condition on the top

 ...PML parameters....
 pml_max=1000.0        PML absorption parameter
 pml_thick=0           half-thickness of pml layer (0 = do not use PML)
```

Notes:
This program uses the traditional explicit second order differencing
method.

Two different absorbing boundary condition schemes are available. The
first is a traditional absorbing boundary condition scheme created by
Hale, 1990. The second is based on the perfectly matched layer (PML)
method of Berenger, 1995.

Authors: CWP:Dave Hale
CWP:modified for SU by John Stockwell, 1993.
CWP:added frequency specification of wavelet: Craig Artley, 1993
TAMU:added PML absorbing boundary condition:
Michael Holzrichter, 1998
CWP/WesternGeco:corrected PML code to handle density variations:
Greg Wimpey, 2006

References: (Hale's absobing boundary conditions)
Clayton, R. W., and Engquist, B., 1977, Absorbing boundary conditions
for acoustic and elastic wave equations, Bull. Seism. Soc. Am., 6,
1529-1540.

Clayton, R. W., and Engquist, B., 1980, Absorbing boundary conditions
for wave equation migration, Geophysics, 45, 895-904.

Hale, D., 1990, Adaptive absorbing boundaries for finite-difference
modeling of the wave equation migration, unpublished report from the
Center for Wave Phenomena, Colorado School of Mines.

Richtmyer, R. D., and Morton, K. W., 1967, Difference methods for
initial-value problems, John Wiley & Sons, Inc, New York.

Thomee, V., 1962, A stable difference scheme for the mixed boundary problem
for a hyperbolic, first-order system in two dimensions, J. Soc. Indust.
Appl. Math., 10, 229-245.

Toldi, J. L., and Hale, D., 1982, Data-dependent absorbing side boundaries,
Stanford Exploration Project Report SEP-30, 111-121.

References: (PML boundary conditions)

Jean-Pierre Berenger, ‘‘A Perfectly Matched Layer for the Absorption of Electromagnetic Waves,’’ Journal of Computational Physics, vol. 114, pp. 185-200.

Hastings, Schneider, and Broschat, ‘‘Application of the perfectly matched layer (PML) absorbing boundary condition to elastic wave propogation,’’ Journal of the Accoustical Society of America, November, 1996.

Allen Taflove, ‘‘Electromagnetic Modeling: Finite Difference Time Domain Methods’’, Baltimore, Maryland: Johns Hopkins University Press, 1995, chap. 7, pp. 181-195.

Trace header fields set: ns, delrt, tracl, tracr, offset, d1, d2, sdepth, trid

```
 SUFDMOD2 - Finite-Difference MODeling (2nd order) for acoustic wave equation

 sufdmod2 <vfile >wfile nx= nz= tmax= xs= zs= [optional parameters]

 Required Parameters:
 <vfile file containing velocity[nx][nz]
 >wfile file containing waves[nx][nz] for time steps
 nx= number of x samples (2nd dimension)
 nz= number of z samples (1st dimension)
 xs= x coordinates of source, or, alternatively, the name
of a file that contains the x- and z-coordinates,
with the number of pairs as the first record and
the actual pairs of (x,z) locations following.
 zs= z coordinates of source
 tmax= maximum time

 Optional Parameters:
 sstrength=1.0 strength of source
 pw=0 use point or extended source geometry parameters
  =1  use horizontal plane wave source
 pwt=20 amp taper on ends of line src (in grid points)
 mono=0 use ricker wavelet as source function
  =1  use single frequency src (freq=2*fpeak)
 nt=1+tmax/dt number of time samples (dt determined for stability)
 mt=1 number of time steps (dt) per output time step

 dx=1.0 x sampling interval
 fx=0.0 first x sample
 dz=1.0 z sampling interval
 fz=0.0 first z sample

 fmax = vmin/(10.0*h) maximum frequency in source
 fpeak=0.5*fmax peak frequency in ricker wavelet

 dfile= input file containing density[nx][nz]
 vsx= x coordinate of vertical line of seismograms
 hsz= z coordinate of horizontal line of seismograms
 vsfile= output file for vertical line of seismograms[nz][nt]
 hsfile= output file for horizontal line of seismograms[nx][nt]
 ssfile= output file for source point seismograms[nt]
 verbose=0 =1 for diagnostic messages, =2 for more
 abs=1,1,1,1 absorbing boundary conditions on top,left,bottom,right
sides of the model.
```

```
 =0,1,1,1 for free surface condition on the top

Notes:

This program uses the traditional explicit second order differencing
method.



Authors:  CWP:Dave Hale
          CWP:modified for SU by John Stockwell, 1993.
          U Houston: added plane wave and monochromatic wave
                      source options.  Chris Liner, 2010


Trace header fields set: sx, gx, ns, delrt, tracl, tracr, offset, d1, d2,
                         sdepth, trid

Modifications: Tony Kocurko (TK:)
               Memorial University in Newfoundland and Labrador
               - Allow user to supply the name of a file containing
                 shot point locations, rather than supplying them
                 as values to the xs= and zs= command line arguments.
               - Correct the calculation of izs[is].

Technical reference:
Kelly, K. R., R. W. Ward, S. Treitel, and R. M. Alford (1976),
Synthetic Seismograms: A finite-difference approach,
Geophysics, Vol. 41. No. I (February, 1976), p. 2-27.
```

```
 SUGOUPILLAUDPO - calculate Primaries-Only impulse response of a lossless
      GOUPILLAUD medium for plane waves at normal incidence

 sugoupillaudpo < stdin > stdout [optional parameters]

 Required parameters:
none

 Optional parameters:
l=1    source layer number; 1 <= l <= tr.ns
      Source is located at the top of layer l.
k=1    receiver layer number; 1 <= k
      Receiver is located at the top of layer k.
tmax   number of output time-samples;
      default: long enough to capture all primaries
pV=1   flag for vector field seismogram
      (displacement, velocity, acceleration);
      =-1 for pressure seismogram.
verbose=0     silent operation, =1 list warnings

 Input: Reflection coefficient series:

       impedance[i]-impedance[i+1]
       r[i] = ---------------------------
       impedance[i]+impedance[i+1]

r[0]= surface refl. coef. (as seen from above)
r[n]= refl. coef. of the deepest interface

 Input file is to be in SU format, i.e., binary floats with a SU header.

 Remarks:
 1. For vector fields, a buried source produces a spike of amplitude 1
 propagating downwards and a spike of amplitude -1 propagating upwards.
 A buried pressure source produces spikes of amplitude 1 both in the up-
 and downward directions.
    A surface source induces only a downgoing spike of amplitude 1 at the
 top of the first layer (both for vector and pressure fields).
 2. The sampling interval dt in the header of the input reflectivity file
 is interpreted as a two-way traveltime thicknes of the layers. The sampling
 interval of the output seismogram is the same as that of the input file.
```

```
 Credits:
CWP: Albena Mateeva, April 2001.
```

```
 SUGOUPILLAUD - calculate 1D impulse response of
     non-absorbing Goupillaud medium

 sugoupillaud < stdin > stdout [optional parameters]

 Required parameters:
none

 Optional parameters:
l=1source layer number; 1 <= l <= tr.ns
Source is located at the top of layer l.
k=1 receiver layer number; 1 <= k
Receiver is located at the top of layer k.
tmax  number of output time-samples; default:
tmax=NINT((2*tr.ns-(l-1)-(k-1))/2)  if k < tr.ns
tmax=k if k >=tr.ns
pV=1  flag for vector field seismogram
(displacement, velocity, acceleration);
=-1 for pressure seismogram.
verbose=0  silent operation, =1 list warnings

 Input: Reflection coefficient series:

 impedance[i]-impedance[i+1]
 r[i] = ----------------------------
 impedance[i]+impedance[i+1]

r[0]= surface refl. coef. (as seen from above)
r[n]= refl. coef. of the deepest interface

 Input file is to be in SU format, i.e., binary floats with a SU header.

 Remarks:
 1. For vector fields, a buried source produces a spike of amplitude 1
 propagating downwards and a spike of amplitude -1 propagating upwards.
 A buried pressure source produces spikes of amplitude 1 both in the up
 and downward directions.

 A surface source induces only a downgoing spike of amplitude 1 at the
 top of the first layer (both for vector and pressure fields).
 2. The sampling interval dt in the header of the input reflectivity file
 is interpreted as a two-way traveltime thicknes of the layers. The sampling
 interval of the output seismogram is the same as that of the input file.
```

```
 Credits:
CWP: Albena Mateeva, May 2000, a summer project at Western Geophysical


 ANOTATION used in the code comments [arises from the use of z-transforms]:
Z-sampled: sampling interval equal to the TWO-way
traveltime of the layers;
z-sampled: sampling interval equal to the ONE-way
traveltime of the layers;

 REFERENCES:

1. Ganley, D. C., 1981, A method for calculating synthetic seismograms
which include the effects of absorption and dispersion.
Geophysics, Vol.46, No. 8, p. 1100-1107.

The burial of the source is based on the Appendix of that article.

2. Robinson, E. A., Multichannel Time Series Analysis with Digital
Computer Programs: 1983 Goose Pond Press, 2nd edition.

The recursive polynomials Q, P used in this code are described
in Chapter 3 of the book: Wave Propagation in Layered Media.

My polynomial multiplication and division functions "prod" and
"pratio" are based on Robinson's Fortran subroutines in Chapter 1.

4. Clearbout, J. F., Fundamentals of Geophysical Data Processing with
Applications to Petroleum Prospecting: 1985 Blackwell Scientific
Publications.

Chapter 8, Section 3: Introduces recursive polynomials F, G in a
more intuitive way than Robinson.

The connection between the Robinson's P_k, Q_k and Clearbout's
F_k, G_k is:
P_k(Z) = F_k(Z)
Q_k(Z) = - Z^(k) G_k(1/Z)
```

```
 SUIMP2D - generate shot records for a line scatterer
           embedded in three dimensions using the Born
    integral equation ",

 suimp2d [optional parameters] >stdout

 Optional parameters
nshot=1 number of shots
nrec=1 number of receivers
c=5000 speed
dt=.004 sampling rate
nt=256 number of samples
x0=1000 point scatterer location
z0=1000 point scatterer location
sxmin=0 first shot location
szmin=0 first shot location
gxmin=0 first receiver location
gzmin=0 first receiver location
dsx=100 x-step in shot location
dsz=0   z-step in shot location
dgx=100 x-step in receiver location
dgz=0 z-step in receiver location

 Example:
suimp2d nrec=32 | sufilter | supswigp | ...


 Credits:
CWP: Norm Bleistein, Jack K. Cohen


 Theory: Use the 3D Born integral equation (e.g., Geophysics,
 v51, n8, p1554(7)). Use 2-D delta function for alpha and do
 remaining y-integral by stationary phase.

 Note: Setting a 2D offset in a single offset field beats the
       hell out of us.  We did _something_.

 Trace header fields set: ns, dt, tracl, tracr, fldr, sx, selev,
                          gx, gelev, offset
```

```
SUIMP3D - generate inplane shot records for a point
          scatterer embedded in three dimensions using
          the Born integral equation ",

suimp3d [optional parameters] >stdout

Optional parameters
nshot=1 number of shots
nrec=1 number of receivers
c=5000 speed
dt=.004 sampling rate
nt=256 number of samples
x0=1000 point scatterer location
y0=0 point scatterer location
z0=1000 point scatterer location
   dir=0 do not include direct arrival
             =1 include direct arrival
sxmin=0 first shot location
symin=0 first shot location
szmin=0 first shot location
gxmin=0 first receiver location
gymin=0 first receiver location
gzmin=0 first receiver location
dsx=100 x-step in shot location
dsy=0   y-step in shot location
dsz=0   z-step in shot location
dgx=100 x-step in receiver location
dgy=0 y-step in receiver location
dgz=0 z-step in receiver location

 Example:
       suimp3d nrec=32 | sufilter | supswigp | ...


 Credits:
CWP: Norm Bleistein, Jack K. Cohen
  UHouston: Chris Liner 2010 (added direct arrival option)



 Theory: Use the 3D Born integral equation (e.g., Geophysics,
 v51, n8, p1554(7)). Use 3-D delta function for alpha.
```

```
Note: Setting a 3D offset in a single offset field beats the
      hell out of us.  We did _something_.

Trace header fields set: ns, dt, tracl, tracr, fldr, tracf,
                         sx, sy, selev, gx, gy, gelev, offset
```

```
 SUIMPEDANCE - Convert reflection coefficients to impedances.

 suimpedance <stdin >stdout [optional parameters]

 Optional Parameters:
 v0=1500. Velocity at first sample (m/sec)
 rho0=1.0e6 Density at first sample  (g/m^3)

 Notes:
 Implements recursion [1-R(k)]Z(k) = [1+R(k)]Z(k-1).
 The input traces are assumed to be reflectivities, and thus are
 expected to have amplitude values between -1.0 and 1.0.


 Credits:
SEP: Stew Levin

 Trace header fields accessed: ns
```

```
SUKDSYN2D - Kirchhoff Depth SYNthesis of 2D seismic data from a
            migrated seismic section

  sukdsyn2d  infile  outfile [parameters]

Required parameters:
infile=stdin input migrated section
outfile=stdout file for output seismic traces
ttfile file for input traveltime tables

The following 9 parameters describe traveltime tables:
fzt=  first depth sample
nzt=  number of depth samples
dzt= depth interval
fxt= first lateral sample
nxt= number of lateral samples
dxt= lateral interval
fs= x-coordinate of first source
ns= number of sources
ds= x-coordinate increment of sources

The following 6 parameters describe the migration section:
fz=                   first z-coordinate in migrated section
dz=      vertical spacing of migrated section
nz=            number of depth points in migrated section
fx=                   first x-coordinate of migrated section
dx=      horizontal spacing of migrated section
nx=            number of lateral points in migrated section

Optional Parameters:
nt=501          number of time samples
dt=0.004        time sampling interval (sec)
ft=0.0          first time (sec)
nxo=1                  number of source-receiver offsets
dxo=25                 offset sampling interval
fxo=0.0                first offset
nxs=101                number of shotpoints
dxs=25                 shotpoint sampling interval
fxs=0.0                first shotpoint
fmax=1/(4*dt)          maximum frequency in migration section (Hz)
aperx=nxt*dxt/2   modeling lateral aperature
angmax=60 modeling angle aperature from vertical
v0=1500(m/s) reference velocity value at surface
```

```
dvz=0.0   reference velocity vertical gradient
ls=1                 flag for line source
jpfile=stderr job print file name
mtr=100   print verbal information at every mtr traces

Notes:
This program takes a migrated seismic section and a set of travel time
tables generated using rayt2d for a specific background velocity model
and generates synthetic seismic data in the form of common shot gathers.
(Common offset gathers may be generated by using nxo=1.) (Demigration.)

This program is a tool which may be used for the migration residual
statics estimation technique of Tjan, Audebert, and Larner 1994.

1. The traveltime tables are generated by program rayt2d (or other ones)
   on relatively coarse grids, with dimension ns*nxt*nzt. In the
   modeling process, traveltimes are interpolated into shot/geophone
   positions and migration section grids.
2. The input migration section must be an array of binary floats (no SU
   headers). ",
3. The synthesized traces are output in common-shot gathers in SU format.
4. The memory requirement for this program is about
     (ns*nxt*nzt+nx*nz+4*nr*nzt+3*nxt*nzt)*4 bytes
    where nr = 1+min(nxt*dxt,0.5*offmax+aperx)/dxo.


Author:  CWP: Zhenyue Liu, 07/24/95,  Colorado School of Mines

References:

Tjan, T., F. Audebert, and K. Larner, 1994,
   Prestack migration for residual statics estimation in complex media
   (Appeared in 1994 Project Review, CWP-153.)

Tjan, T., 1995, Residual statics estimation for data from structurally
   complex areas using prestack depth migration: M.Sc. thesis, Colorado
   School of Mines. (In progress.)

Larner, K., and Tjan, T., 1995, Simultaneous statics and velocity
   estimation for data from structurally complex areas.
   (Appeared in 1995 Project Review, CWP-185.)
```

```
Trace header fields set: ns, dt, delrt, tracl, tracr, fldr, tracf
                         offset, sx, gx, trid, counit
```

```
 SUNHMOSPIKE - generates SPIKE test data set with a choice of several
   Non-Hyperbolic MOveouts

   sunhmospike [optional parameters] > out_data_file

 Optional parameters:
nt=300   number of time samples
ntr=20   number of traces
dt=0.001 time sample rate in seconds
offref=2000 reference offset

gopt= 1 = parabolic transform model
2 = Foster/Mosher pseudo hyperbolic option model
3 = linear tau-p model
depthref=400 reference depth used when gopt=2
offinc=100 offset increment
nspk=4   number of events

p1 = 0 event moveout for event #1 in ms on reference offset
t1 = 100 intercept time ms event #1
a1 = 1.0 amplitude for event #1

p2 = 200 event moveout for event #2 in ms on reference offset
t2 = 100 intercept time ms for spike #2
a2 = 1.0 amplitude for event #2

p3 = 0; event moveout for event #3 in ms on reference offset
t3 = 200 intercept time for spike #3
a3 = 1.0 amplitude for event #3
p4 = 120  event moveout for event #4 in ms on reference offset

t4 = 200 intercept time s for spike #4
a4 = 1.0 amplitude for event #4

cdp = 1  output cdp number

 Notes:
 Creates a common cdp su data file with up to four spike events
 for impulse response studies for suradon, and sutifowler


 Credits:
CWP: Shuki Ronen, Chris Liner,
```

```
Modified: CWP   by John Anderson, April, 1994, to have
 appropriate trace header words and default values
 for SUTIFOWLER tests
```

```
 SUNULL - create null (all zeroes) traces

 sunull nt=   [optional parameters] >outdata

 Required parameter
  nt= number of samples per trace

 Optional parameters
  ntr=5 number of null traces to create
  dt=0.004 time sampling interval

 Rationale: It is sometimes useful to insert null traces
 between "panels" in a shell loop.

 See also: sukill, sumute, suzero


 Credits:
CWP: Jack K. Cohen

 Trace header fields set: ns, dt, tracl
```

```
SUPLANE - create common offset data file with up to 3 planes

suplane [optional parameters] >stdout

Optional Parameters:
 npl=3 number of planes
 nt=64  number of time samples
 ntr=32 number of traces
 taper=0 no end-of-plane taper
= 1 taper planes to zero at the end
 offset=400  offset
 dt=0.004   time sample interval in seconds
...plane 1 ...
dip1=0 dip of plane #1 (ms/trace)
  len1= 3*ntr/4 HORIZONTAL extent of plane (traces)
ct1= nt/2 time sample for center pivot
cx1= ntr/2 trace for center pivot
...plane 2 ...
dip2=4 dip of plane #2 (ms/trace)
len2= 3*ntr/4 HORIZONTAL extent of plane (traces)
ct2= nt/2 time sample for center pivot
cx2= ntr/2 trace for center pivot
...plane 3 ...
dip3=8 dip of plane #3 (ms/trace)
len3= 3*ntr/4 HORIZONTAL extent of plane (traces)
ct3= nt/2 time sample for center pivot
cx3= ntr/2 trace for center pivot

 liner=0 use parameters
= 1 parameters set for 64x64 data set
with separated dipping planes.

 Credits:
CWP: Chris Liner

 Trace header fields set: ns, dt, offset, tracl
```

```
 SURANDSPIKE - make a small data set of RANDom SPIKEs

   surandspike [optional parameters] > out_data_file

 Creates a common offset su data file with random spikes

 Optional parameters:
n1=500  number of time samples
n2=200 number of traces
  dt=0.002  time sample rate in seconds
nspk=20 number of spikes per trace
amax=0.2 abs(max) spike value
mode=1 different spikes on each trace
=2 same spikes on each trace
  seed=from_clock     random number seed (integer)


 Credits:
ARAMCO: Chris Liner

 Trace header fields set: ns, dt, offset
```

```
SUREMAC2D - Acoustic 2D Fourier method modeling with high accuracy
            Rapid Expansion Method (REM) time integration

suremac2d [parameters]

Required parameters:

opflag=     0: variable density wave equation
            1: constant density wave equation
            2: non-reflecting wave equation

nx=         number of grid points in horizontal direction
nz=         number of grid points in vertical direction
nt=         number of time samples
dx=         spatial increment in horizontal direction
dz=         spatial increment in vertical direction
dt=         time sample interval in seconds
isx=        grid point # of horizontal source positions
isz=        grid point # of vertical source positions

Optional parameters:
fx=0.0      first horizontal coordinate
fz=0.0      first vertical coordinate
irx=        horizontal grid point # of vertical receiver lines
irz=        vertical grid point # of horizontal receiver lines
w=0.1       width of spatial source distribution (see notes)
sflag=2     source time function
            0: user supplied source function
            1: impulse (spike at t=0)
            2: Ricker wavelet
fmax=       maximum frequency of Ricker (default) wavelet
amps=1.0    amplitudes of sources
prec=0      1: precompute Bessel coefficients b_k (see notes)
            2: use precomputed Bessel coefficients b_k
fsflag=0    1: perform run with free surface b.c.
vmaxu=      user-defined maximum velocity
dtsnap=0.0  time interval in seconds of wave field snapshots
iabso=1     apply absorbing boundary conditions (0: none)
abso=0.1    damping parameter for absorbing boundaries
nbwx=20     horizontal width of absorbing boundary
nbwz=20     vertical width of absorbing boundary
verbose=0   1: show parameters used
            2: print maximum amplitude at every expansion term
```

```
velfile=vel          velocity filename
densfile=dens        density filename
sname=wavelet.su     user supplied source time function filename
sepxname=sectx.su    x-direction pressure sections filename
sepzname=sectz.su    z-direction pressure sections filename
snpname=snap.su      pressure snapshot filename
jpfile=stderr        diagnostic output

Notes:
 0. The combination of the Fourier method with REM time integration
    allows the computation of synthetic seismograms which are free
    of numerical grid dispersion. REM has no restriction on the
    time step size dt. The Fourier method requires at least two
    grid points per shortest wavelength.
 1. nx and nz must be valid numbers for pfafft transform lengths.
    nx and nz must be odd numbers (unless opflag=1). For valid
    numbers see e.g. numbers in structure 'nctab' in source file
    $CWPROOT/src/cwp/lib/pfafft.c.
 2. Velocities (and densities) are stored as plain C style files
    of floats where the fast dimension is along the z-direction.
 3. Units must be consistent, e.g. m, s and m/s.
 4. A 20 grid points wide border at the sides and the bottom of
    the modeling grid is used for sponge boundary conditions
    (default: iabso=1).
    Source and receiver lines should be placed some (e.g. 10) grid
    points away from the absorbing boundaries in order to reduce
    reflections due to obliquely incident wavefronts.
 5. Dominant frequency is about fmax/2 (sflag=2), absolute maximum
    is delayed by 3/fmax from beginning of wavelet.
 6. If opflag!=1 the source should be not a spike in space; the
    parameter w determines at which distance (in grid points) from
    the source's center the Gaussian weight decays to 10 percent
    of its maximum. w=2 may be a reasonable choice; however, the
    waveform will be distorted.
 7. Horizontal and vertical receiver line sections are written to
    separate files. Each file can hold more than one line.
 8. Parameter vmaxu may need to be chosen larger than the highest
    propagation velocity if the modeling run becomes unstable.
    This happens if the largest eigenvalue of the modeling
    operator L is larger than estimated from the largest velocity
    due to variations of the density.
    In particular if using the variable density acoustic wave
```

equation the eigenvalues depend also on the density and it is impossible to estimated the largest eigenvalue analytically.

9. Bessel coefficients can be precomputed (prec=1) and stored on disk to save CPU time when several shots need to be run.
   In this case computation of Bessel coefficients can be skipped and read from disk file for reuse (prec=2).
   For reuse of Bessel coefficients the user may need to define the overall maximum velocity (vmaxu).
10. If snapshots are not required, a spike source (sflag=1) may be applied and the resulting impulse response seismograms can be convolved later with a desired wavelet.
11. The free surface (fsflag=1) does not coincide with the first vertical grid index (0). It appears to be half a grid spacing above that position.

Acoustic 2D Fourier method modeling with REM time integration

Reference:
Kosloff, D., Fihlo A.Q., Tessmer, E. and Behle, A., 1989,
Numerical solution of the acoustic and elastic wave equations by a new rapid expansion method, Geophysical Prospecting, 37, 383-394

Credits:
University of Hamburg: Ekkehart Tessmer, October 2012

```
SUREMEL2DAN - Elastic anisotropic 2D Fourier method modeling with high
              accuracy Rapid Expansion Method (REM) time integration

suremel2dan [parameters]

Required parameters:

nx=         number of grid points in horizontal direction
nz=         number of grid points in vertical direction
nt=         number of time samples
dx=         spatial increment in horizontal direction
dz=         spatial increment in vertical direction
dt=         time sample interval in seconds
isx=        grid point # of horizontal source positions
isz=        grid point # of vertical source positions
styp=       source types (pressure, shear, single forces)
samp=       amplitudes of sources
amode=      0: isotropic,  1: anisotropic
vmax=       global maximum velocity (only if amode=1)
vmin=       global minimum velocity (only if amode=1)

Optional parameters:
fx=0.0      first horizontal coordinate
fz=0.0      first vertical coordinate
irx=        horizontal grid point # of vertical receiver lines
irz=        vertical grid point # of horizontal receiver lines
rxtyp=      types of horizontal receiver lines
rztyp=      types of vertical receivers lines
sntyp=      types of snapshots
            0: P,  1: S,  2: UX,  3: UZ
w=0.1       width of spatial source distribution (see notes)
sflag=2     source time function
            0: user supplied source function
            1: impulse (spike at t=0)
            2: Ricker wavelet
fmax=       maximum frequency of Ricker (default) wavelet
amps=1.0    amplitudes of sources
prec=0      1: precompute Bessel coefficients b_k (see notes)
            2: use precomputed Bessel coefficients b_k
vmaxu=      user-defined maximum velocity
dtsnap=0.0  time interval in seconds of wave field snapshots
iabso=1     apply absorbing boundary conditions (0: none)
abso=0.1    damping parameter for absorbing boundaries
```

```
nbwx=20     horizontal width of absorbing boundary
nbwz=20     vertical width of absorbing boundary
verbose=0   1: show parameters used
            2: print maximum amplitude at every expansion term

c11file=c11       c11 filename
c13file=c13       c13 filename
c15file=c15       c15 filename
c33file=c33       c33 filename
c35file=c35       c35 filename
c55file=c55       c55 filename
vpfile=vp         P-velocity filename
vsfile=vs         S-velocity filename
densfile=dens     density filename

sname=wavelet.su  user supplied source time function filename

Basenames of seismogram and snapshot files:
xsect=xsect_     x-direction section files basename
zsect=zsect_     z-direction section files basename
snap=snap_       snapshot files basename

jpfile=stderr        diagnostic output

Notes:
 0. The combination of the Fourier method with REM time integration
    allows the computation of synthetic seismograms which are free
    of numerical grid dispersion. REM has no restriction on the
    time step size dt. The Fourier method requires at least two
    grid points per shortest wavelength.
 1. nx and nz must be valid numbers for pfafft transform lengths.
    nx and nz must be odd numbers. For valid numbers see e.g.
    numbers in structure 'nctab' in source file
    $CWPROOT/src/cwp/lib/pfafft.c.
 2. Velocities and densities are stored as plain C style files
    of floats where the fast dimension is along the z-direction.
 3. Units must be consistent, e.g. m, s and m/s.
 4. A 20 grid points wide border at the sides and the bottom of
    the modeling grid is used for sponge boundary conditions
    (default: iabso=1).
    Source and receiver lines should be placed some (e.g. 10) grid
    points away from the absorbing boundaries in order to reduce
    reflections due to obliquely incident wavefronts.
```

```
 5. Dominant frequency is about fmax/2 (sflag=2), absolute maximum
    is delayed by 3/fmax from beginning of wavelet.
 6. If source is not single force (i.e. pressure or shear source)
    it should be not a spike in space; the parameter w determines
    at which distance (in grid points) from the source's center
    the Gaussian weight decays to 10 percent of its maximum.
    w=2 may be a reasonable choice; however, the waveform will be
    distorted.
 7. Horizontal and vertical receiver line sections are written to
    separate files. Each file can hold more than one line.
 8. Parameter vmaxu may need to be chosen larger than the highest
    propagation velocity if the modeling run becomes unstable.
    This happens if the largest eigenvalue of the modeling
    operator L is larger than estimated from the largest velocity
    due to variations of the density.
 9. Bessel coefficients can be precomputed (prec=1) and stored on
    disk to save CPU time when several shots need to be run.
    In this case computation of Bessel coefficients can be skipped
    and read from disk file for reuse (prec=2).
    For reuse of Bessel coefficients the user may need to define
    the overall maximum velocity (vmaxu).
10. If snapshots are not required, a spike source (sflag=1) may be
    applied and the resulting impulse response seismograms can be
    convolved later with a desired wavelet.
11. Output is written to SU style files.                          ",
    Basenames of seismogram and snapshot output files will be
    extended by the type of the data (p, s, ux, or uz).
    Additionally seismogram files will be consecutively numbered.

Caveat:
    Time sections and snapshots are kept entirely in memory during
    run time. Therefore, lots of time section and snapshots may
    eat up a large amount of memory.


 Elastic anisotropic 2D Fourier method modeling with REM time integration

 Reference:
 Kosloff, D., Fihlo A.Q., Tessmer, E. and Behle, A., 1989,
   Numerical solution of the acoustic and elastic wave equations by a
   new rapid expansion method, Geophysical Prospecting, 37, 383-394

Credits:
```

University of Hamburg: Ekkehart Tessmer, July 2013

```
 SUSPIKE - make a small spike data set

 suspike [optional parameters] > out_data_file

 Creates a common offset su data file with up to four spikes
 for impulse response studies

 Optional parameters:
nt=64  number of time samples
ntr=32 number of traces
  dt=0.004  time sample rate in seconds
  offset=400  offset
nspk=4 number of spikes
ix1= ntr/4 trace number (from left) for spike #1
it1= nt/4  time sample to spike #1
ix2 = ntr/4 trace for spike #2
it2 = 3*nt/4  time for spike #2
ix3 = 3*ntr/4; trace for spike #3
it3 = nt/4; time for spike #3
ix4 = 3*ntr/4; trace for spike #4
it4 = 3*nt/4; time for spike #4


 Credits:
CWP: Shuki Ronen, Chris Liner

 Trace header fields set: ns, dt, offset
```

```
 SUSYNCZ - SYNthetic seismograms for piecewise constant V(Z) function
   True amplitude (primaries only) modeling for 2.5D

  susyncz > outfile [parameters]

 Required parameters:
 none

 Optional Parameters:
 ninf=4        number of interfaces (not including upper surface)
 dip=5*i       dips of interfaces in degrees (i=1,2,3,4)
 zint=100*i    z-intercepts of interfaces at x=0 (i=1,2,3,4)
 v=1500+ 500*i velocities below surface & interfaces (i=0,1,2,3,4)
 rho=1,1,1,1,1 densities below surface & interfaces (i=0,1,2,3,4)
 nline=1 number of (identical) lines
 ntr=32        number of traces
 dx=10         trace interval
 tdelay=0      delay in recording time after source initiation
 dt=0.004      time interval
 nt=128        number of time samples

 Notes:
 The original purpose of this code was to create some nontrivial
 data for Brian Sumner's CZ suite.

 The program produces zero-offset data over dipping reflectors.

 In the original fortran code, some arrays had the index
 interval 1:ninf, as a natural way to index over the subsurface
 reflectors.  This indexing was preserved in this C translation.
 Consequently, some arrays in the code do not use the 0 "slot".

 Example:
susyncz | sufilter | sugain tpow=1 | display_program

 Trace header fields set: tracl, ns, dt, delrt, ntr, sx, gx


 Credits:
  CWP: Brian Sumner, 1983, 1985, Fortran design and code
      CWP: Stockwell & Cohen, 1995, translation to C
```

```
SUSYNLVCW - SYNthetic seismograms for Linear Velocity function
 for mode Converted Waves

susynlvcw >outfile [optional parameters]

Optional Parameters:
nt=101 number of time samples
dt=0.04 time sampling interval (sec)
ft=0.0 first time (sec)
nxo=1 number of source-receiver offsets
dxo=0.05 offset sampling interval (km)
fxo=0.0 first offset (km, see notes below)
xo=fxo,fxo+dxo,... array of offsets (use only for non-uniform offsets)
nxm=101 number of midpoints (see notes below)
dxm=0.05 midpoint sampling interval (km)
fxm=0.0 first midpoint (km)
nxs=101 number of shotpoints (see notes below)
dxs=0.05 shotpoint sampling interval (km)
fxs=0.0 first shotpoint (km)
x0=0.0 distance x at which v00 is specified
z0=0.0 depth z at which v00 is specified
v00=2.0 velocity at x0,z0 (km/sec)
gamma=1.0 velocity ratio, upgoing/downgoing
dvdx=0.0 derivative of velocity with distance x (dv/dx)
dvdz=0.0 derivative of velocity with depth z (dv/dz)
fpeak=0.2/dt peak frequency of symmetric Ricker wavelet (Hz)
ref="1:1,2;4,2" reflector(s):  "amplitude:x1,z1;x2,z2;x3,z3;...
smooth=0 =1 for smooth (piecewise cubic spline) reflectors
er=0 =1 for exploding reflector amplitudes
ls=0 =1 for line source; default is point source
ob=1 =1 to include obliquity factors
sp=1 =1 to account for amplitude spreading
 =0 for constant amplitudes throught out
tmin=10.0*dt minimum time of interest (sec)
ndpfz=5 number of diffractors per Fresnel zone
verbose=0 =1 to print some useful information

Notes:

Offsets are signed - may be positive or negative.  Receiver locations
are computed by adding the signed offset to the source location.

Specify either midpoint sampling or shotpoint sampling, but not both.
```

```
 If neither is specified, the default is the midpoint sampling above.

 More than one ref (reflector) may be specified.  When obliquity factors
 are included, then only the left side of each reflector (as the x,z
 reflector coordinates are traversed) is reflecting.  For example, if x
 coordinates increase, then the top side of a reflector is reflecting.
 Note that reflectors are encoded as quoted strings, with an optional
 reflector amplitude: preceding the x,z coordinates of each reflector.
 Default amplitude is 1.0 if amplitude: part of the string is omitted.

 Note that gamma<1 implies P-SV mode conversion, gamma>1 implies SV-P,
 and gamma=1 implies no mode conversion.



 based on Dave Hale's code susynlv, but modified
 by Mohammed Alfaraj to handle mode conversion
 Date of modification: 01/07/92

 Trace header fields set: trid, counit, ns, dt, delrt,
tracl. tracr, fldr, tracf,
cdp, cdpt, d2, f2, offset, sx, gx
```

```
SUSYNLVFTI - SYNthetic seismograms for Linear Velocity function in a
             Factorized Transversely Isotropic medium

susynlvfti >outfile [optional parameters]

Optional Parameters:
nt=101 number of time samples
dt=0.04 time sampling interval (sec)
ft=0.0 first time (sec)
kilounits=1            input length units are km or kilo-feet
                       =0 for m or ft
                       Note: Output (sx,gx,offset) are always m or ft
nxo=1 number of source-receiver offsets
dxo=0.05 offset sampling interval (kilounits)
fxo=0.0 first offset (kilounits, see notes below)
xo=fxo,fxo+dxo,...    array of offsets (use only for non-uniform offsets)
nxm=101 number of midpoints (see notes below)
dxm=0.05 midpoint sampling interval (kilounits)
fxm=0.0 first midpoint (kilounits)
nxs=101 number of shotpoints (see notes below)
dxs=0.05 shotpoint sampling interval (kilounits)
fxs=0.0 first shotpoint (kilounits)
x0=0.0 distance x at which v00 is specified
z0=0.0 depth z at which v00 is specified
v00=2.0 velocity at x0,z0 (kilounits/sec)
dvdx=0.0 derivative of velocity with distance x (dv/dx)
dvdz=0.0 derivative of velocity with depth z (dv/dz)
fpeak=0.2/dt peak frequency of symmetric Ricker wavelet (Hz)
ref=1:1,2;4,2 reflector(s):  "amplitude:x1,z1;x2,z2;x3,z3;...
smooth=0 =1 for smooth (piecewise cubic spline) reflectors
er=0 =1 for exploding reflector amplitudes
ls=0 =1 for line source; default is point source
ob=0 =1 to include obliquity factors
tmin=10.0*dt minimum time of interest (sec)
ndpfz=5 number of diffractors per Fresnel zone
verbose=1 =1 to print some useful information

For transversely isotropic media:
angxs=0.0 angle of symmetry axis with the vertical (degrees)
define the media using either
a=1.0 corresponding to the ratio of elastic coef.(c1111/c3333)
f=0.4 corresponding to the ratio of elastic coef. (c1133/c3333)
l=0.3 corresponding to the ratio of elastic coef. (c1313/c3333)
```

```
Alternately use Tompson\'s parameters:
delta=0 Thomsen's 1986 defined parameter
epsilon=0 Thomsen's 1986 defined parameter
ntries=40 number of iterations in Snell's law and offset searches
epsx=.001 lateral offset tolerance
epst=.0001 reflection time tolerance
nitmax=12 max number of iterations in travel time integrations

Notes:

Offsets are signed - may be positive or negative.  Receiver locations
are computed by adding the signed offset to the source location.

Specify either midpoint sampling or shotpoint sampling, but not both.
If neither is specified, the default is the midpoint sampling above.

More than one ref (reflector) may be specified.  When obliquity factors
are included, then only the left side of each reflector (as the x,z
reflector coordinates are traversed) is reflecting.  For example, if x
coordinates increase, then the top side of a reflector is reflecting.
Note that reflectors are encoded as quoted strings, with an optional
reflector amplitude: preceding the x,z coordinates of each reflector.
Default amplitude is 1.0 if amplitude: part of the string is omitted.

Concerning the choice of delta and epsilon. The difference between delta",
and epsilon should not exceed one. A possible break down of the program
is the result. This is caused primarly by the break down in the two point",
ray-tracing. Also keep the values of delta and epsilon between 2 and -2.
```

```
 SUSYNLV - SYNthetic seismograms for Linear Velocity function

 susynlv >outfile [optional parameters]

 Optional Parameters:
 nt=101                  number of time samples
 dt=0.04                 time sampling interval (sec)
 ft=0.0                  first time (sec)
 kilounits=1             input length units are km or kilo-feet
 =0 for m or ft
                         Note: Output (sx,gx,offset) are always m or ft
 nxo=1                   number of source-receiver offsets
 dxo=0.05                offset sampling interval (kilounits)
 fxo=0.0                 first offset (kilounits, see notes below)
 xo=fxo,fxo+dxo,...      array of offsets (use only for non-uniform offsets)
 nxm=101                 number of midpoints (see notes below)
 dxm=0.05                midpoint sampling interval (kilounits)
 fxm=0.0                 first midpoint (kilounits)
 nxs=101                 number of shotpoints (see notes below)
 dxs=0.05                shotpoint sampling interval (kilounits)
 fxs=0.0                 first shotpoint (kilounits)
 x0=0.0                  distance x at which v00 is specified
 z0=0.0                  depth z at which v00 is specified
 v00=2.0                 velocity at x0,z0 (kilounits/sec)
 dvdx=0.0                derivative of velocity with distance x (dv/dx)
 dvdz=0.0                derivative of velocity with depth z (dv/dz)
 fpeak=0.2/dt            peak frequency of symmetric Ricker wavelet (Hz)
 ref="1:1,2;4,2"         reflector(s):  "amplitude:x1,z1;x2,z2;x3,z3;...
 smooth=0                =1 for smooth (piecewise cubic spline) reflectors
 er=0                    =1 for exploding reflector amplitudes
 ls=0                    =1 for line source; default is point source
 ob=1                    =1 to include obliquity factors
 tmin=10.0*dt            minimum time of interest (sec)
 ndpfz=5                 number of diffractors per Fresnel zone
 verbose=0               =1 to print some useful information

Notes:
Offsets are signed - may be positive or negative.  Receiver locations
are computed by adding the signed offset to the source location.

Specify either midpoint sampling or shotpoint sampling, but not both.
If neither is specified, the default is the midpoint sampling above.
```

```
More than one ref (reflector) may be specified. Do this by putting
additional ref= entries on the commandline. When obliquity factors
are included, then only the left side of each reflector (as the x,z
reflector coordinates are traversed) is reflecting.  For example, if x
coordinates increase, then the top side of a reflector is reflecting.
Note that reflectors are encoded as quoted strings, with an optional
reflector amplitude: preceding the x,z coordinates of each reflector.
Default amplitude is 1.0 if amplitude: part of the string is omitted.


 Credits: CWP Dave Hale, 09/17/91,  Colorado School of Mines
    UTulsa Chris Liner 5/22/03 added kilounits flag

 Trace header fields set: trid, counit, ns, dt, delrt,
tracl. tracr, fldr, tracf,
cdp, cdpt, d2, f2, offset, sx, gx
```

```
SUSYNVXZCS - SYNthetic seismograms of common shot in V(X,Z) media via
 Kirchhoff-style modeling

susynvxzcs<vfile >outfile  nx= nz= [optional parameters]

Required Parameters:
<vfile        file containing velocities v[nx][nz]
>outfile      file containing seismograms of common ofset
nx=           number of x samples (2nd dimension) in velocity
nz=           number of z samples (1st dimension) in velocity

Optional Parameters:
nt=501         number of time samples
dt=0.004       time sampling interval (sec)
ft=0.0         first time (sec)
fpeak=0.2/dt peak frequency of symmetric Ricker wavelet (Hz)
nxg= number of receivers of input traces
dxg=15 receiver sampling interval (m)
fxg=0.0 first receiver (m)
nxd=5          skipped number of receivers
nxs=1 number of offsets
dxs=50 shot sampling interval (m)
fxs=0.0 first shot (m)
dx=50          x sampling interval (m)
fx=0.          first x sample (m)
dz=50          z sampling interval (m)
nxb=nx/2     band width centered at midpoint (see note)
nxc=0         hozizontal range in which velocity is changed
nzc=0         vertical range in which velocity is changed
pert=0        =1 calculate time correction from v_p[nx][nz]
vpfile        file containing slowness perturbation array v_p[nx][nz]
ref="1:1,2;4,2" reflector(s):  "amplitude:x1,z1;x2,z2;x3,z3;...
smooth=0 =1 for smooth (piecewise cubic spline) reflectors
ls=0 =1 for line source; =0 for point source
tmin=10.0*dt minimum time of interest (sec)
ndpfz=5 number of diffractors per Fresnel zone
cable=1 roll reciever spread with shot
 =0 static reciever spread
verbose=0 =1 to print some useful information

Notes:
This algorithm is based on formula (58) in Geo. Pros. 34, 686-703,
by N. Bleistein.
```

```
 Traveltime and amplitude are calculated by finite difference which
 is done only in one of every NXD receivers; in skipped receivers,
 interpolation is used to calculate traveltime and amplitude. ",
 For each receiver, traveltime and amplitude are calculated in the
 horizontal range of (xg-nxb*dx, xg+nxb*dx). Velocity is changed by
 constant extropolation in two upper trianglar corners whose width is
 nxc*dx and height is nzc*dz.

 Eikonal equation will fail to solve if there is a polar turned ray.
 In this case, the program shows the related geometric information.
 There are three ways to remove the turned rays: smoothing velocity,
 reducing nxb, and increaing nxc and nzc (if the turned ray occurs
 in shallow areas). To prevent traveltime distortion from an over-
 smoothed velocity, traveltime is corrected based on the slowness
 perturbation.

 More than one ref (reflector) may be specified.
 Note that reflectors are encoded as quoted strings, with an optional
 reflector amplitude: preceding the x,z coordinates of each reflector.
 Default amplitude is 1.0 if amplitude: part of the string is omitted.



Author: Zhenyue Liu, 07/20/92, Center for Wave Phenomena
Many subroutines borrowed from Dave Hale's program: SUSYNLV

Trino Salinas, 07/30/96, fixed a bug in the geometry
setting to allow the spread move with the shots.

Chris Liner 12/10/08  added cable option, set fldr header word

 Trace header fields set: trid, counit, ns, dt, delrt,
tracl. tracr, fldr, tracf,
sx, gx
```

```
SUSYNVXZ - SYNthetic seismograms of common offset V(X,Z) media via
 Kirchhoff-style modeling

susynvxz >outfile [optional parameters]

Required Parameters:
<vfile file containing velocities v[nx][nz]
nx= number of x samples (2nd dimension)
nz= number of z samples (1st dimension)
Optional Parameters:
nxb=nx band centered at midpoint
nxd=1 skipped number of midponits
dx=100 x sampling interval (m)
fx=0.0 first x sample
dz=100 z sampling interval (m)
nt=101 number of time samples
dt=0.04 time sampling interval (sec)
ft=0.0 first time (sec)
nxo=1   number of offsets
dxo=50 offset sampling interval (m)
fxo=0.0 first offset (m)
nxm=101 number of midpoints
dxm=50 midpoint sampling interval (m)
fxm=0.0 first midpoint (m)
fpeak=0.2/dt peak frequency of symmetric Ricker wavelet (Hz)
ref="1:1,2;4,2" reflector(s):  "amplitude:x1,z1;x2,z2;x3,z3;...
smooth=0 =1 for smooth (piecewise cubic spline) reflectors
ls=0 =1 for line source; default is point source
tmin=10.0*dt minimum time of interest (sec)
ndpfz=5 number of diffractors per Fresnel zone
verbose=0 =1 to print some useful information

Notes:
This algorithm is based on formula (58) in Geo. Pros. 34, 686-703,
by N. Bleistein.

Offsets are signed - may be positive or negative. ",
Traveltime and amplitude are calculated by finite differences which
is done only in part of midpoints; in the skiped midpoint, interpolation
is used to calculate traveltime and amplitude. ",

More than one ref (reflector) may be specified.
Note that reflectors are encoded as quoted strings, with an optional
```

```
 reflector amplitude: preceding the x,z coordinates of each reflector.
 Default amplitude is 1.0 if amplitude: part of the string is omitted.


   CWP:  Zhenyue Liu, 07/20/92
Many subroutines borrowed from Dave Hale's program: SUSYNLV

 Trace header fields set: trid, counit, ns, dt, delrt,
tracl. tracr,
cdp, cdpt, d2, f2, offset, sx, gx
```

```
 SUVIBRO - Generates a Vibroseis sweep (linear, linear-segment,
dB per Octave, dB per Hertz, T-power)

 suvibro [optional parameters] > out_data_file

 Optional Parameters:
 dt=0.004 time sampling interval
 sweep=1    linear sweep
     =2 linear-segment
     =3 decibel per octave
     =4 decibel per hertz
     =5 t-power
 swconst=0.0 sweep constant (see note)
 f1=10.0 sweep frequency at start
 f2=60.0 sweep frequency at end
 tv=10.0 sweep length
 phz=0.0 initial phase (radians=1 default)
 radians=1 =0 degrees
 fseg=10.0,60.0 frequency segments (see notes)
 tseg=0.0,10.0 time segments (see notes)
 t1=1.0 length of taper at start (see notes)
 t2=1.0 length of taper at end (see notes)
 taper=1 linear
   =2 sine
=3 cosine
=4 gaussian (+/-3.8)
=5 gaussian (+/-2.0)

 Notes:
 The default tapers are linear envelopes. To eliminate the
 taper, choose t1=t2=0.0.

 "swconst" is active only with nonlinear sweeps, i.e. when
 sweep=3,4,5. ",
 "tseg" and "fseg" arrays are used when only sweep=2

 Sweep is a modulated cosine function.


 Author: CWP: Michel Dietrich
   Rewrite: Tagir Galikeev, CWP,  7 October 1994

 Trace header fields set: ns, dt, tracl, sfs, sfe, slen, styp
```

```
 SUWAVEFORM - generate a seismic wavelet

 suwaveform <stdin >stdout [optional parameters]

 Required parameters:
one of the optional parameters listed below

 Optional parameters:
type=akb wavelet type
   akb: AKB wavelet defined by max frequency fpeak
   berlage: Berlage wavelet
   gauss:   Gaussian wavelet defined by frequency fpeak
   gaussd:  Gaussian first derivative wavelet
   ricker1: Ricker wavelet defined by frequency fpeak
   ricker2: Ricker wavelet defined by half and period
   spike:   spike wavelet, shifted by time tspike
   unit: unit wavelet, i.e. amplitude = 1 = const.

dt=0.004 time sampling interval in seconds
ns= if set, number of samples in  output trace

fpeak=20.0 peak frequency of a Berlage, Ricker, or Gaussian,
   and maximum frequency of an AKB wavelet in Hz

half=1/fpeak   Ricker wavelet "ricker2": half-length
period=c*half  Ricker wavelet "ricker2": period (c=sqrt(6)/pi)
distort=0.0 Ricker wavelet "ricker2": distortion factor
decay=4*fpeak  Berlage wavelet: exponential decay factor in 1/sec
tn=2    Berlage wavelet: time exponent
ipa=-90 Berlage wavelet: initial phase angle in degrees
tspike=0.0 Spike wavelet: time at spike in seconds
verbose=0 1: echo output wavelet length


 Notes:
If ns is not defined, the program determines the trace length
depending on the dominant signal period.

The Ricker wavelet "ricker1" and the Gaussian wavelet "gauss
are zero-phase. For these two wavelets, the trace header word
delrt is set such that the peak amplitude is at t=0 seconds.
If this is not acceptable, use "sushw key=delrt a=0".
```

```
The Ricker wavelets can be defined either by the peak frequency
fpeak ("ricker1") or by its half-length, the period, and a
distortion factor ("ricker2"). "ricker" is an acceptable
alias for "ricker1".

The Berlage wavelet is defined by the peak frequency fpeak, a time
time exponent tn describing the wavelet shape at its beginning,
and an exponential decay factor describing the amplitude decay
towards later times. The parameters tn and decay are non-negative,
real numbers; tn is typically a small integer number and decay a
multiple of the dominant signal period 1/fpeak. Additionally, an
initial phase angle can be given; use -90 or 90 degrees for
zero-amplitude at the beginning.

For an AKB wavelet, fpeak is the maximum frequency; the peak
frequency is about 1/3 of the fpeak value.

The output wavelet can be normalized or scaled with "sugain".
Use "suvibro" to generate a Vibroseis sweep.

 Example:
 A normalized, zero-phase Ricker wavelet with a peak frequency
 of 15 Hz is generated and convolved with a spike dataset:

suwaveform type=ricker1 fpeak=15 | sugain pbal=1 > wavelet.su
suplane npl=1 | suconv sufile=wavelet.su | suxwigb

 Gaussian and derivatives of Gaussians:
 Use "sudgwaveform" to generate these

 Caveat:
This program does not check for aliasing.



 Author:
Nils Maercklin, RISSC, University of Napoli, Italy, 2006

 References:
Aldridge, D. F. (1990). The Berlage wavelet.
Geophysics, vol. 55(11), p. 1508-1511.
Alford, R., Kelly, K., and Boore, D. (1947). Accuracy
of finite-difference modeling of the acoustic wave
```

```
equation. Geophysics, vol. 39, p. 834-842. (AKB wavelet)
Sheriff, R. E. (2002). Encyclopedic dictionary of
applied geophysics. Society of Exploration Geophysicists,
Tulsa. (Ricker wavelet, page 301)

 Notes:
For more information on the wavelets type "sudoc waveforms"
or have a look at "$CWPROOT/src/cwp/lib/waveforms.c".

 Credits:
CWP, the authors of the subroutines in "waveforms.c".

 Trace header fields set: ns, dt, trid, delrt
```

```
 SUGAUSSTAPER - Multiply traces with gaussian taper

 sugausstaper < stdin > stdout [optional parameters]

 Required Parameters:
   <none>

 Optional parameters:
 key=offset    keyword of header field to weight traces by
 x0=300        key value defining the center of gaussian window",
 xw=50         width of gaussian window in units of key value

 Notes:
 Traces are multiplied with a symmetrical gaussian taper
   w(t)=exp(-((key-x0)/xw)**2)
 unlike "sutaper" the value of x0 defines center of the taper
 rather than the edges of the data.

 Credits:

Thomas Bohlen, formerly of TU Bergakademie, Freiberg GDR
      most recently of U Karlsruhe
          04.01.2002

 Trace header fields accessed: ns
```

```
 SURAMP - Linearly taper the start and/or end of traces to zero.

 suramp <stdin >stdout [optional parameters]

 Required parameters:
  if dt is not set in header, then dt is mandatory

 Optional parameters
tmin=tr.delrt/1000 end of starting ramp (sec)
tmax=(nt-1)*dt beginning of ending ramp (sec)
  dt = (from header) sampling interval (sec)

 The taper is a linear ramp from 0 to tmin and/or tmax to the
 end of the trace.  Default is a no-op!


 Credits:

CWP: Jack K. Cohen, Ken Larner

 Trace header fields accessed: ns, dt, delrt
```

```
 SUTAPER - Taper the edge traces of a data panel to zero.


 sutaper <stdin >stdout [optional parameters]

 Optional Parameters:
 ntr=tr.ntr number of traces. If tr.ntr is not set, then
  ntr is mandatory
 tr1=0  number of traces to be tapered at beginning
 tr2=tr1 number of traces to be tapered at end
 min=0. minimum amplitude factor of taper
 tbeg=0 length of taper (ms) at trace start
 tend=0 length of taper (ms) at trace end
 taper=1 taper type
 =1 linear (default)
 =2 sine
 =3 cosine
 =4 gaussian (+/-3.8)
 =5 gaussian (+/-2.0)

 Notes:
   To eliminate the taper, choose tbeg=0. and tend=0. and tr1=0


 Credits:

CWP: Chris Liner, Jack K. Cohen

 Trace header fields accessed: ns, ntr

 Rewrite: Tagir Galikeev, October 2002
```

```
 SUTXTAPER - TAPER in (X,T) the edges of a data panel to zero.

 sutxtaper <stdin >stdout [optional parameters]

 Optional Parameters:
 low=0.     minimum amplitude factor of taper
 tbeg=0     length of taper (ms) at trace start
 tend=0      length of taper (ms) at trace end
 taper=1       taper type
                 =1 linear (default)
                 =2 sine
                 =3 cosine
                 =4 gaussian (+/-3.8)
                 =5 gaussian (+/-2.0)
 key=tr set key to compute x-domain taper weights
               default is using internal tracecount (tr)
 tr1=0         number of traces to be tapered at beg (key=tr)
 tr2=tr1       number of traces to be tapered at end (key=tr)

 min=0. minimum value of key where taper starts (amp=1.)
 max=0. maximum value of key where taper starts (amp=1.)
 dx=1.  length of taper (in key units)
if key=tr (unset) length is tr1 and (ntr-tr2)

 Notes:
   Taper type is used for trace (x-domain) tapering as well
   as for time domain tapering.
   The taper is applied to all traces <tr1 (or key<min) and
   >tr2 (or key >max) and all time samples <tbeg and >tend.
   Taper weights are amp*1 for traces n tr1<n<tr2 and computed
   for all other traces corresponding to the taper typ.
   If key is given the taper length is defined by dx, otherwise
   the length of taper is tr1 and (ntr-tr2) respectively.
   To eliminate the taper, choose tbeg=0. and tend=0. and tr1=0
   If key is set, min,max values take precedence over tr1,tr2.


 Credits: (based on sutaper)

CWP: Chris Liner, Jack K. Cohen

 Trace header fields accessed: ns
```

Rewrite: Tagir Galikeev, October 2002
Rewrite: Gerald Klein, IFM-GEOMAR, April 2004

```
 SUAMP - output amp, phase, real or imag trace from
  (frequency, x) domain data

 suamp <stdin >stdout mode=amp

 Required parameters:
 none
 Optional parameter:
 mode=amp output flag
  =amp output amplitude traces
  =logamp output log(amplitude) traces
  =phase output phase traces
  =ouphase output unwrapped phase traces (oppenheim)
  =suphase output unwrapped phase traces (simple)
  =real output real parts
        =imag output imag parts
 jack=0 =1  divide value at zero frequency by 2
(operative only for mode=amp)

 .... phase unwrapping options .....
 unwrap=1  |dphase| > pi/unwrap constitutes a phase wrapping
(operative only for mode=suphase)
 trend=1 remove linear trend from the unwrapped phase
 zeromean=0 assume phase(0)=0.0, else assume phase is zero mean
 smooth=0 apply damped least squares smoothing to unwrapped phase
 r=10.0     ... damping coefficient, only active when smooth=1

 Notes:
  The trace returned is half length from 0 to Nyquist.

 Example:
  sufft <data | suamp >amp_traces
 Example:
sufft < data > complex_traces
   suamp < complex_traces mode=real > real_traces
   suamp < complex_traces mode=imag > imag_traces

 Note: the inverse of the above operation is:
suop2 real_traces imag_traces op=zipper > complex_traces

 Note: Explanation of jack=1
 The amplitude spectrum is the modulus of the complex output of
 the fft. f(0) is thus the average over the range of integration
```

```
 of the transform. For causal functions, or equivalently, half
 transforms, f(0) is 1/2 of the average over the full range.
 Most oscillatory functions encountered in wave applications are
 zero mean, so this is usually not an issue.

 Note: Phase unwrapping:

 The mode=ouphase uses the phase unwrapping method of Oppenheim and
 Schaffer, 1975.
 The mode=suphase generates unwrapped phase assuming that jumps
 in phase larger than pi/unwrap constitute a phase wrapping.

 Credits:
CWP: Shuki Ronen, Jack K. Cohen c.1986

 Notes:
If efficiency becomes important consider inverting main loop
      and repeating extraction code within the branches of the switch.

 Trace header fields accessed: ns, trid
 Trace header fields modified: ns, trid
```

```
 SUANALYTIC - use the Hilbert transform to generate an ANALYTIC
(complex) trace

 suanalytic <stdin >sdout

 Optional Parameter:
 phaserot= phase rotation in degrees of complex trace

 Notes:

 The output are complex valued traces. The analytic trace is defined as",
   ctr[ i ] = indata[i] + i hilb[indata[t]]
 where the imaginary part is the hilbert tranform of the original trace

 The Hilbert transform is computed in the direct (time) domain

 If phaserot is set, then a phase rotated complex trace is produced
   ctr[ i ] = cos[phaserot]*indata[i] + i sin[phaserot]* hilb[indata[t]]

 Use "suamp" to extract real, imaginary, amplitude (modulus), etc
 Exmple:
 suanalytic < sudata | suamp mode=amp | suxgraph



 Use "suattributes" for instantaneous phase, frequency, etc.


 Credits:
    CWP: John Stockwell, based on suhilb by Jack K. Cohen.

 Trace header fields accessed: ns, trid

 Technical references:
 Oppenheim, A. V. and Schafer, R. W. (1999).
     Discrete-Time Signal Processing. Prentice Hall Signal Processing Series.
     Prentice Hall, New Jersey, 2.
 Taner, M. T., F. Koehler, and R. E. Sheriff, 1979, Complex seismic
    trace analysis: Geophysics, 44, 1041-1063.
```

```
 SUCCEPSTRUM - Compute the complex CEPSTRUM of a seismic trace  "

  sucepstrum < stdin > stdout

 Required parameters:
none
 Optional parameters:
 sign1=1 sign of real to complex transform
 sign2=-1 sign of complex to complex (inverse) transform

 ...phase unwrapping .....
 mode=ouphase Oppenheim's algorithm for phase unwrapping
=suphase  simple unwrap phase
 unwrap=1  |dphase| > pi/unwrap constitutes a phase wrapping
(operative only for mode=suphase)

 trend=1 deramp the phase, =0 do not deramp the phase
 zeromean=0 assume phase starts at 0,  =1 phase is zero mean

 Notes:
 The cepstrum is defined as the fourier transform of the the decibel
 spectrum, as though it were a time domain signal.

 CC(t) = FT[ln[T(omega)] ] = FT[ ln|T(omega)| + i phi(omega) ]
T(omega) = |T(omega)| exp(i phi(omega))
       phi(omega) = unwrapped phase of T(omega)

 Phase unwrapping:
 The mode=ouphase uses the phase unwrapping method of Oppenheim and
 Schaffer, 1975, which operates integrating the derivative of the phase

 The mode=suphase generates unwrapped phase assuming that jumps
 in phase larger than dphase=pi/unwrap constitute a phase wrapping. In this case
 the jump in phase is replaced with the average of the jumps in phase
 on either side of the location where the suspected phase wrapping occurs.

 In either mode, the user has the option of de-ramping the phase, by
 removing its linear trend via trend=1 and of deciding whether the
 phase starts at phase=0 or is of  zero mean via zeromean=1.


 Author: John Stockwell, Dec 2010
  based on sucepstrum.c by:
```

Credits:
Balazs Nemeth of Potash Corporation of Saskatchewan Inc.
given to CWP in 2008

```
 SUCCWT - Complex continuous wavelet transform of seismic traces

 succwt < tdata.su > tfdata.su [optional parameters]

 Required Parameters:
 None

 Optional Parameters:
 noct=5 Number of octaves (int)
 nv=10 Number of voices per octave (int)
 fmax=Nyq Highest frequency in transform
 p=-0.5 Power of scale value normalizing CWT
=0 for amp-preserved spec. decomp.
 c=1/(2*fmax) Time-domain inverse gaussian damping parameter
(bigger c means more wavelet oscillations,
default gives minimal oscillations)
 k=1 Use complex Morlet as wavelet transform kernel
=2 use Fourier kernel ... Exp[i 2 pi f t]
 fs=1 Use dyadic freq sampling (CWT standard, honors
noct, nv)
=2 use linear freq sampling (Fourier standard)
 df=1 Frequency sample interval in Hz (used only for fs=2)
NOTE: not yet implimented (hardwired to df=1)
 dt=(from tr.dt) Sample interval override (in secs, if time data)
 verbose=0  Run silent, except echo c value. (=1 for more info)

 Examples:
 This generates amplitude spec of the CWT impulse response (IR).
  suspike ntr=1 ix1=1 nt=125 | succwt | suamp | suximage &
 Real part of Fourier IR with linear freq sampling:
 suspike ntr=1 ix1=1 nt=125 | succwt k=2 fs=2 | suamp mode=real | suximage &
 Real part of Fourier IR with dyadic freq sampling:
 suspike ntr=1 ix1=1 nt=125 | succwt k=2 | suamp mode=real | suximage &

 Inverse CWT: (within a constant scale factor)
... | succwt p=-1 | suamp mode=real | sustack key=cdp > inv.su

 Notes:
 1. Total number of scales: nscale = noct*nv
 2. Each input trace spawns nscale complex output traces
 3. Lowest frequency in the transform is fmax/( 2^(noct-1/nv) )
 4. Header field (cdp) used as cwt spectrum counter
 5. Header field (cdpt) used as scale counter within cwt spectrum
```

```
 6. Header field (gut) holds number of cwt scales ‘na’
 7. Header field (unscale) holds CWT scale ‘a’

 Header fields set: tracl, cdp, cdpt, unscale, gut



 Copyright (c) University of Tulsa, 2003-4.
 All rights reserved.
 Author: UTulsa: Chris Liner, SEP: Bob Clapp

 todo:
fix fs=2 case to allow df not equal to 1
 History:
 6/18/04
major overhaul by Clapp, including fourier implementation.
Speedup ~ 41 times (4100 %)
 2/20/04
made p=-0.5 default
 2/16/04
added p option to experiment with CWT normalization
 2/12/04
replace fb (bandwidth parameter) with c (t-domain gaussian damping const.)
 2/10/04 --- in sync with EAS paper in prep
changed morlet scaling (c = 1) to preserve time-domain peak amplitude
changed morlet exp sign to std CWT definition (conjugate) and
mathematica result that only gives positive freq gaussian with neg exp
 1/26/04
added linear frequency sampling option
 1/23/04
figured out fb and made it a getpar
key: Look at real ccwt output and determine fb by number of
oscillations desired: Default gives -+-+-+-
 1/20/04
beefed up verbose output
dimension wavelet to length 2*nt and change correlation call
... this is done to avoid conv edge effects
 1/19/04
added fourier wavlet option for comparison with Fourier Transform action
 1/17/04
complex morlet amp scaling now set to preserve first scale amp with IR
 1/16/04
added dt getpar to handle depth input properly
```

```
preserves first tracl so tracl is ok after spice
 11/11/03
initial version

 Trace header fields set: tracl, cdp, cdpt, unscale, gut
```

```
 SUCEPSTRUM - transform to the CEPSTRal domain

  sucepstrum <stdin >sdout sign1=1

 Required parameters:
 none

 Optional parameters:
 sign1=1 sign in exponent of fft
 sign2=-1 sign in exponent of ifft
 dt=from header sampling interval
 verbose=1 =0 to stop advisory messages

 .... phase unwrapping options .....
 mode=ouphase Oppenheim's phase unwrapping
=suphase simple jump detecting phase unwrapping
 unwrap=1       |dphase| > pi/unwrap constitutes a phase wrapping
    =0 no phase unwrapping (in mode=suphase  only)
 trend=1 remove linear trend from the unwrapped phase
 zeromean=0     assume phase(0)=0.0, else assume phase is zero mean
 smooth=0      apply damped least squares smoothing to unwrapped phase
 r=10     ... damping coefficient, only active when smooth=1

 Notes:
 The complex log fft of a function F(t) is given by:
 clogfft(F(t)) = log(FFT(F(t)) = log|F(omega)| + iphi(omega)
 where phi(omega) is the unwrapped phase. Note that 0< unwrap<= 1.0
 allows phase unwrapping to be tuned, if desired.

 The ceptrum is the inverse Fourier transform of the log fft of F(t)
 F(t_c) =cepstrum(F(t)) = INVFFT[log(FFT(F(t))]
                        =INVFFT[log|F(omega)| + iphi(omega)]
 Here t_c is the cepstral time domain.

 To facilitate further processing, the sampling interval
 in quefrency and first quefrency (0) are set in the
 output header.

 Caveats:
 No check is made that the data ARE real time traces!

 Use suminphase to make minimum phase representations of signals
```

```
 Credits:
      CWP: John Stockwell, June 2013 based on
sufft by:
CWP: Shuki Ronen, Chris Liner, Jack K. Cohen
CENPET: Werner M. Heigl - added well log support
U Montana: Bob Lankston - added m_unwrap_phase feature

 Note: leave dt set for later inversion

 Trace header fields accessed: ns, dt, d1, f1
 Trace header fields modified: ns, d1, f1, trid
```

```
SUCLOGFFT - fft real time traces to complex log frequency domain traces

suclogftt <stdin >sdout sign=1

Required parameters:
none

Optional parameters:
sign=1 sign in exponent of fft
dt=from header sampling interval
verbose=1 =0 to stop advisory messages

.... phase unwrapping options .....
mode=suphase simple jump detecting phase unwrapping
 =ouphase  Oppenheim's phase unwrapping
unwrap=1       |dphase| > pi/unwrap constitutes a phase wrapping
   =0 no phase unwrapping (in mode=suphase  only)
trend=1 remove linear trend from the unwrapped phase
zeromean=0     assume phase(0)=0.0, else assume phase is zero mean

Notes:
clogfft(F(t)) = log(FFT(F(t)) = log|F(omega)| + iphi(omega)
where phi(omega) is the unwrapped phase. Note that 0< unwrap<= 1.0
allows phase unwrapping to be tuned, if desired.

To facilitate further processing, the sampling interval
in frequency and first frequency (0) are set in the
output header.

suclogfft unwrap=0 | suiclogfft is not quite a no-op since the trace
length will usually be longer due to fft padding.

Caveats:
No check is made that the data ARE real time traces!

Output is type complex. To view amplitude, phase or real, imaginary
parts, use    suamp
PI/unwrap = minimum dphase is assumed to constitute a wrap in phase
for suphase unwrapping only

Examples:
suclogfft < stdin | suamp mode=real | ....
suclogfft < stdin | suamp mode=imag | ....
```

```
 The real and imaginary parts of the complex log spectrum are the
 respective amplitude and phase (unwrapped) phase spectra of the input
 signal.

 Example:  Homomorphic wavelet estimation
 suclogfft < shotgather | suamp mode=real | sustack key=dt > real.su
 suclogfft < shotgather | suamp mode=imag | sustack key=dt > imag.su
 suop2 real.su imag.su op=zipper | suiclogfft | suminphase > wavelet.su




 Credits:
      CWP: John Stockwell, Dec 2010 based on
sufft by:
CWP: Shuki Ronen, Chris Liner, Jack K. Cohen
CENPET: Werner M. Heigl - added well log support
U Montana: Bob Lankston - added m_unwrap_phase feature

 Note: leave dt set for later inversion

 Trace header fields accessed: ns, dt, d1, f1
 Trace header fields modified: ns, d1, f1, trid
```

```
 SUCWT - generates Continous Wavelet Transform amplitude, regularity
         analysis in the wavelet basis

     sucwt < stdin [Optional parameters ] > stdout

 Required Parameters:
 none

 Optional Parameters:
 base=10 Base value for wavelet transform scales
 first=-1 First exponent value for wavelet transform scales
 expinc=0.01 Exponent increment for wavelet transform scales
 last=1.5 Last exponent value for wavelet transform scales

 Wavelet Parameters:
 wtype=0 2nd derivative of Gaussian (Mexican hat)
=1 4th derivative of Gaussian (witch's hat)
=2 6th derivative of Gaussian (wizard's hat)
 nwavelet=1024 number of samples in the wavelet
 xmin=-20 minimum x value wavelet is computed
 xcenter=0 center x value  wavelet is computed
 xmax=20 maximum x value wavelet is computed
 sigma=1 sharpness parameter ( sigma > 1 sharper)

 verbose=0 silent, =1 chatty
 holder=0 =1 compute Holder regularity estimate
 divisor=1.0 a floating point number >= 1.0 (see notes)

 Notes:
 This is the CWT version of the time frequency analysis notion that is
 applied in sugabor.
 The parameter base is the base of the power that is applied to scale
 the wavelet. Some mathematical literature assume base 2. Base 10 works
 well here.

 Default option yields an output similar to that of sugabor. With the
 parameter holder=1 an estimate of the instantaneous Holder regularity
 (the Holder exponent) is output for each input data value. The result
 is a Holder exponent trace for each corresponding input data trace.

 The strict definition of the Holder exponent is the maximum slope of
 the rise of the spectrum in the log(amplitude) versus log(scale) domain:
```

```
 divisor=1.0 means the exponent is computed simply by fitting a line
 through all of the values in the transform. A value of divisor>1.0
 indicates that the Holder exponent is determined as the max of slopes
 found in (total scales)/divisor length segments.

 Some experimentation with the parameters nwavelet, first, last, and
 expinc may be necessary before a desirable output is obtained. The
 most effective way to proceed is to perform a number of tests with
 holder=0 to determine the range of first, last, and expinc that best
 represents the data in the wavelet domain. Then experimentation with
 holder=1 and values of divisor>=1.0 may proceed.



 Credits:
CWP: John Stockwell, Nov 2004
 inspired in part by "bhpcwt" in the BHP_SU package, code written by
BHP: Michael Glinsky,c. 2002, based loosely on a Matlab CWT function

 References:

 Li C.H., (2004), Information passage from acoustic impedence to
 seismogram: Perspectives from wavelet-based multiscale analysis,
 Journal of Geophysical Research, vol. 109, B07301, p.1-10.

 Mallat, S. and  W. L. Hwang, (1992),  Singularity detection and
 processing with wavelets,  IEEE Transactions on information, v 38,
 March 1992, p.617 - 643.
```

```
 SUFFT - fft real time traces to complex frequency traces

 suftt <stdin >sdout sign=1

 Required parameters:
 none

 Optional parameters:
 sign=1 sign in exponent of fft
 dt=from header sampling interval
 verbose=1 =0 to stop advisory messages

 Notes: To facilitate further processing, the sampling interval
 in frequency and first frequency (0) are set in the
 output header.

 sufft | suifft is not quite a no-op since the trace
 length will usually be longer due to fft padding.

 Caveats:
 No check is made that the data IS real time traces!

 Output is type complex. To view amplitude, phase or real, imaginary
 parts, use    suamp

 Examples:
 sufft < stdin | suamp mode=amp | ....
 sufft < stdin | suamp mode=phase | ....
 sufft < stdin | suamp mode=uphase | ....
 sufft < stdin | suamp mode=real | ....
 sufft < stdin | suamp mode=imag | ....


 Credits:

CWP: Shuki Ronen, Chris Liner, Jack K. Cohen
CENPET: Werner M. Heigl - added well log support

 Note: leave dt set for later inversion

 Trace header fields accessed: ns, dt, d1, f1
 Trace header fields modified: ns, d1, f1, trid
```

```
 SUGABOR -  Outputs a time-frequency representation of seismic data via
        the Gabor transform-like multifilter analysis technique
presented by Dziewonski, Bloch and  Landisman, 1969.

    sugabor <stdin >stdout [optional parameters]

 Required parameters:
if dt is not set in header, then dt is mandatory

 Optional parameters:
dt=(from header) time sampling interval (sec)
fmin=0 minimum frequency of filter array (hz)
fmax=NYQUIST  maximum frequency of filter array (hz)
beta=3.0 ln[filter peak amp/filter endpoint amp]
band=.05*NYQUIST filter bandwidth (hz)
alpha=beta/band^2 filter width parameter
verbose=0 =1 supply additional info
holder=0 =1 output Holder regularity estimate
=2 output linear regularity estimate

 Notes: This program produces a muiltifilter (as opposed to moving window)
 representation of the instantaneous amplitude of seismic data in the
 time-frequency domain. (With Gaussian filters, moving window and multi-
 filter analysis can be shown to be equivalent.)

 An input trace is passed through a collection of Gaussian filters
 to produce a collection of traces, each representing a discrete frequency
 range in the input data. For each of these narrow bandwidth traces, a
 quadrature trace is computed via the Hilbert transform. Treating the narrow
 bandwidth trace and its quadrature trace as the real and imaginary parts
 of a "complex" trace permits the "instantaneous" amplitude of each
 narrow bandwidth trace to be compute. The output is thus a representation
 of instantaneous amplitude as a function of time and frequency.

 Some experimentation with the "band" parameter may necessary to produce
 the desired time-frequency resolution. A good rule of thumb is to run
 sugabor with the default value for band and view the image. If band is
 too big, then the t-f plot will consist of stripes parallel to the frequency
 axis. Conversely, if band is too small, then the stripes will be parallel
 to the time axis.

 Caveat:
 The Gabor transform is not a wavelet transform, but rather are sharp
```

```
frame basis. However, it is nearly a Morlet continuous wavelet transform
so the concept of Holder regularity may have some meaning. If you are
computing Holder regularity of, say, a migrated seismic section, then
set band to 1/3 of the frequency band of your data.

Examples:
   suvibro | sugabor | suximage
   suvibro | sugabor | suxmovie n1= n2= n3=
    (because suxmovie scales it's amplitudes off of the first panel,
     may have to experiment with the wclip and bclip parameters
   suvibro | sugabor | supsimage | ... ( your local PostScript utility)


Credits:

CWP: John Stockwell, Oct 1994
      CWP: John Stockwell Oct 2004, added holder=1 option
Algorithm:

This programs takes an input seismic trace and passes it
through a collection of truncated Gaussian filters in the frequency
domain.

The bandwidth of each filter is given by the parameter "band". The
decay of these filters is given by "alpha", and the number of filters
is given by nfilt = (fmax - fmin)/band. The result, upon inverse
Fourier transforming, is that nfilt traces are created, with each
trace representing a different frequency band in the original data.

For each of the resulting bandlimited traces, a quadrature (i.e. pi/2
phase shifted) trace is computed via the Hilbert transform. The
bandlimited trace constitutes a "complex trace", with the bandlimited
trace being the "real part" and the quadrature trace being the
"imaginary part".  The instantaneous amplitude of each bandlimited
trace is then computed by computing the modulus of each complex trace.
(See Taner, Koehler, and Sheriff, 1979, for a discussion of complex
trace analysis.

The final output for a given input trace is a map of instantaneous
amplitude as a function of time and frequency.

This is not a wavelet transform, but rather a redundant frame
representation.
```

References: Dziewonski, Bloch, and Landisman, 1969, A technique for the analysis of transient seismic signals, Bull. Seism. Soc. Am., 1969, vol. 59, no.1, pp.427-444.

Taner, M., T., Koehler, F., and Sheriff, R., E., 1979, Complex seismic trace analysis, Geophysics, vol. 44, pp.1041-1063.

Chui, C., K.,1992, Introduction to Wavelets, Academic Press, New York.

Trace header fields accessed: ns, dt, trid, ntr
Trace header fields modified: tracl, tracr, d1, f2, d2, trid, ntr

```
 SUHILB - Hilbert transform

 suhilb <stdin >sdout

 Note: the transform is computed in the direct (time) domain


 Credits:
CWP: Jack Cohen
      CWP: John Stockwell, modified to use Dave Hale's hilbert() subroutine

 Trace header fields accessed: ns, trid
```

```
 SUICEPSTRUM - fft of complex log frequency traces to real time traces

  suicepstrum <stdin >sdout sign2=-1

 Required parameters:
  none

 Optional parameter:
  sign1=1 sign in exponent of first fft
  sign2=-1 sign in exponent of inverse fft
sym=0 =1 center  output
dt=tr.dt time sampling interval (s) from header
if not set assumed to be .004s
 Output traces are normalized by 1/N where N is the fft size.

 Note:
 The forward  cepstral transform is the
   F(t_c) = InvFFT[ln[FFT(F(t))]]
 The inverse  cepstral transform is the
   F(t) = InvFFT[exp[FFT(F(t_c))]]

 Here t_c is the cepstral time (quefrency) domain

 Credits:

   CWP: John Stockwell, Dec 2010 based on
     suifft.c by:
CWP: Shuki Ronen, Chris Liner, Jack K. Cohen,  c. 1989

 Trace header fields accessed: ns, trid
 Trace header fields modified: ns, trid
```

```
 SUICLOGFFT - fft of complex log frequency traces to real time traces

  suiclogftt <stdin >sdout sign=-1

 Required parameters:
  none

 Optional parameter:
  sign=-1 sign in exponent of inverse fft
sym=0 =1 center  output
 Output traces are normalized by 1/N where N is the fft size.

 Note:
 Nominally this is the inverse to the complex log fft, but
 suclogfft | suiclogfft is not quite a no-op since the trace
  length will usually be longer due to fft padding.


 Example:  Homomorphic wavelet estimation
 suclogfft < shotgather | suamp mode=real | sustack key=dt > real.su
 suclogfft < shotgather | suamp mode=imag | sustack key=dt > imag.su
 suop2 real.su imag.su op=zipper | suiclogfft | suminphase > wavelet.su




 Credits:

   CWP: John Stockwell, Dec 2010 based on
     suifft.c by:
CWP: Shuki Ronen, Chris Liner, Jack K. Cohen,  c. 1989

 Trace header fields accessed: ns, trid
 Trace header fields modified: ns, trid
```

```
 SUIFFT - fft complex frequency traces to real time traces

 suiftt <stdin >sdout sign=-1

 Required parameters:
  none

 Optional parameter:
  sign=-1 sign in exponent of inverse fft

 Output traces are normalized by 1/N where N is the fft size.

 Note: sufft | suifft is not quite a no-op since the trace
  length will usually be longer due to fft padding.


 Credits:

CWP: Shuki, Chris, Jack

 Trace header fields accessed: ns, trid
 Trace header fields modified: ns, trid
```

```
 SUMINPHASE - convert input to minimum phase

 suminphase <stdin >stdout [optional parameters]

 Required parameters:
if dt is not set in header, then dt is mandatory

 Optional parameters:
sign1=1 sign of first transform (1 or -1)
sign2=-1 sign of second transform (-1 or 1)
     pnoise=1.e-9    white noise in spectral routine
verbose=0 =1 for advisory messages


 Example:  Homomorphic wavelet estimation

 suclogfft < shotgather | suamp mode=real | sustack key=dt > real.su
 suclogfft < shotgather | suamp mode=imag | sustack key=dt > imag.su
 suop2 real.su imag.su op=zipper | suiclogfft | suminphase > wavelet.su


 Credits:
      SEAM Project: Bruce VerWest c. 2013

 Trace header fields accessed: ns, dt, d1
```

```
 SUPHASEVEL - Multi-mode PHASE VELocity dispersion map computed
              from shot record(s)

 suphasevel <infile >outfile [optional parameters]

 Optional parameters:
 fv=330 minimum phase velocity (m/s)
 nv=100 number of phase velocities
 dv=25 phase velocity step (m/s)
 fmax=50 maximum frequency to process (Hz)
=0 process to nyquist
 norm=0 do not normalize by amplitude spectrum
=1 normalize by amplitude spectrum
 verbose=0 verbose = 1 echoes information

 Notes:  Offsets read from headers.
  1. output is complex frequency data
  2. offset header word must be set (signed offset is ok)
  3. norm=1 tends to blow up aliasing and other artifacts
  4. For correct suifft later, use fmax=0
  5. For later processing outtrace.dt=domega
  6. works for 2D or 3D shots in any offset order


 Using this program:

 First: use
  suspecfx < shotrecord.su | suximage
 to see what the maximum bandwidth is in your data. This will
 give you an idea about the possible value for fmax.

 Second: Plot your data or some subset of your data via:
     suxwigb < shotrecord.su key=offset

 You can then estimate the range of phase velocities by looking
 at the maximum and minimum slopes of arrivals in your data.
 This will allow you do set first velocity fv and the increment
 in velocity dv, that make sense for your data.

 You can pick values of offset and time by placing the cursor
 on the desired location on the plot and pressing the \'s\' key
 The picks will appear in your terminal window.
```

```
 When displaying, don't forget to use suamp to compute the
 modulus of the complex values that this program puts out.

   suphasevel < shotrecord.su [parameters] | suamp | suximage



 Credits:

UHouston: Chris Liner June2008 (cloned from suspecfk)

  This code implements the following integral transform

             _
            /
  u(w,v) = / k(w,x,v) u(w,x) dx
          _/
  where
u(w,v) is the phase velocity dispersion image
k(w,x,v) is the transform kernel.... exp(-i w x / v)
u(w,x) = FT[u(t,x)] is the input shot record(s)
          _/
 Reference: Park, Miller, and Xia (1998, SEG Abstracts)

 Trace header fields accessed: dt, offset, ns
 Trace header fields modified: nx,dt,trid,d1,f1,d2,f2,tracl
```

```
SURADON - compute forward or reverse Radon transform or remove multiples
          by using the parabolic Radon transform to estimate multiples
          and subtract.

    suradon <stdin >stdout [Optional Parameters]

Optional Parameters:
choose=0    0  Forward Radon transform
            1  Compute data minus multiples
            2  Compute estimate of multiples
            3  Compute forward and reverse transform
            4  Compute inverse Radon transform
igopt=1     1  parabolic transform: g(x) = offset**2
            2  Foster/Mosher psuedo hyperbolic transform
                  g(x) = sqrt(depth**2 + offset**2)
            3  Linear tau-p: g(x) = offset
            4  abs linear tau-p: g(x) = abs(offset)
offref=2000.    reference maximum offset to which maximum and minimum
                moveout times are associated
interoff=0.     intercept offset to which tau-p times are associated
pmin=-200       minimum moveout in ms on reference offset
pmax=400        maximum moveout in ms on reference offset
dp=16           moveout increment in ms on reference offset
pmula=80        moveout in ms on reference offset where multiples begin
                    at maximum time
pmulb=200       moveout in ms on reference offset where multiples begin
                    at zero time
depthref=500.   Reference depth for Foster/Mosher hyperbolic transform
nwin=1          number of windows to use through the mute zone
f1=60.          High-end frequency before taper off
f2=80.          High-end frequency
prewhite=0.1    Prewhitening factor in percent.
cdpkey=cdp      name of header word for defining ensemble
offkey=offset   name of header word with spatial information
nxmax=240       maximum number of input traces per ensemble
ltaper=7   taper (integer) for mute tapering function

Optimizing Parameters:
The following parameters are occasionally used to avoid spatial aliasing
problems on the linear tau-p transform.  Not recommended for other
transforms...
ninterp=0      number of traces to interpolate between each input trace
                  prior to computing transform
```

```
 freq1=4.0      low-end frequency in Hz for picking (good default: 3 Hz)
                (Known bug: freq1 cannot be zero)
 freq2=20.0     high-end frequency in Hz for picking (good default: 20 Hz)
 lagc=400       length of AGC operator for picking (good default: 400 ms)
 lent=5         length of time smoother in samples for picker
                     (good default: 5 samples)
 lenx=7         length of space smoother in samples for picker
                     (good default: 1 sample)
 xopt=1         1 = use differences for spatial derivative
                        (works with irregular spacing)
                0 = use FFT derivative for spatial derivatives
                      (more accurate but requires regular spacing and
                      at least 16 input tracs--will switch to differences
                      automatically if have less than 16 input traces)


 Credits:
CWP: John Anderson (visitor to CSM from Mobil) Spring 1993

 Multiple removal notes:
Usually the input data are NMO corrected CMP gathers.  The
first pass is to compute a parabolic Radon transform and
  identify the multiples in the transform domain.  Then, the
  module is run on all the data using "choose=1" to estimate
  and subtract the multiples.  See the May, 1993 CWP Project
Review for more extensive documentation.

 NWIN notes:
The parabolic transform runs with higher resolution if the
  mute zone is honored.  When "nwin" is specified larger than
    one (say 6), then multiple windows are used through the mute
  zone.  It is assumed in this case that the input data are
  sorted by the offkey header item from small offset to large
  offset.  This causes the code to run 6 times longer.  The
      mute time is taken from the "muts" header word.
      You may have to manually set this header field yourself, if
      it is not already set.

 References:
 Anderson, J. E., 1993, Parabolic and linear 2-D, tau-p transforms
       using the generalized radon tranform, in May 11-14, 1993
       Project Review, Consortium Project on Seismic Inverse methods
       for Complex Structures, CWP-137, Center for Wave Phenomena
```

internal report.
Other References cited in above paper:
Beylkin, G,.1987, The discrete Radon transform: IEEE Transactions of Acoustics, Speech, and Signal Processing, 35, 162-712.
Chapman, C.H.,1981, Generalized Radon transforms and slant stacks: Geophysical Journal of the Royal Astronomical Society, 66, 445-453.
Foster, D. J. and Mosher, C. C., 1990, Multiple supression using curvilinear Radon transforms: SEG Expanded Abstracts 1990, 1647-1650.
Foster, D. J. and Mosher, C. C., 1992, Suppression of multiples using the Radon transform: Geophysics, 57, No. 3, 386-395.
Gulunay, N., 1990, F-X domain least-squares Tau-P and Tau-Q: SEG Expanded Abstracts 1990, 1607-1610.
Hampson, D., 1986, Inverse velocity stacking for multiple elimination: J. Can. Soc. Expl. Geophs., 22, 44-55.
Hampson, D., 1987, The discrete Radon transform: a new tool for image enhancement and noise suppression: SEG Expanded Abstracts 1978, 141-143.
Johnston, D.E., 1990, Which multiple suppression method should I use? SEG Expanded Abstracts 1990, 1750-1752.

Trace header words accessed: ns, dt, cdpkey, offkey, muts

```
 SUSLOWFT - Fourier Transforms by a (SLOW) DFT algorithm (Not an FFT)

 suslowft <stdin >sdout sign=1

 Required parameters:
  none

 Optional parameters:
  sign=1 sign in exponent of fft
  dt=from header sampling interval

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt, trid

 Notes: To facilitate further processing, the sampling interval
        in frequency and first frequency (0) are set in the
output header.
 Warning: This program is *not* fft based. Use only for demo
     purposes, *not* for large data processing.

  No check is made that the data are real time traces!
 suslowft | suslowift is not quite a no-op since the trace
 length will usually be longer due to fft padding.

 Caveats:
 No check is made that the data IS real time traces!

 Output is type complex. To view amplitude, phase or real, imaginary
 parts, use    suamp

 Examples:
 suslowft < stdin | suamp mode=amp | ....
 suslowft < stdin | suamp mode=phase | ....
 suslowft < stdin | suamp mode=real | ....
 suslowft < stdin | suamp mode=imag | ....



 Credits:

CWP: Shuki, Chris, Jack

 Note: leave dt set for later inversion
```

```
 SUSLOWIFT - Fourier Transforms by (SLOW) DFT algorithm (Not an FFT)
             complex frequency to real time domain traces

 suslowift <stdin >sdout sign=-1

 Required parameters:
  none

 Optional parameters:
  sign=-1 sign in exponent of fft
  dt=from header sampling interval

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt, trid

 Notes: To facilitate further processing, the sampling interval
       in frequency and first frequency (0) are set in the
output header.

 Warning: This program is *not* fft based. Use only for demo
     purposes, *not* for large data processing.

  No check is made that the data are real time traces!


 Credits:

CWP: John Stockwell c. 1993
       based on suifft:   Shuki Ronen, Chris Liner, Jack K. Cohen

 Note: leave dt set for later inversion
```

```
 SUSPECFK - F-K Fourier SPECtrum of data set

 suspecfk <infile >outfile [optional parameters]

 Optional parameters:

 dt=from header time sampling interval
 dx=from header(d2) or 1.0 spatial sampling interval

 verbose=0 verbose = 1 echoes information

 tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

 Note: To facilitate further processing, the sampling intervals
       in frequency and wavenumber as well as the first
frequency (0) and the first wavenumber are set in the
output header (as respectively d1, d2, f1, f2).

 Note: The relation: w = 2 pi F is well known, but there
doesn't seem to be a commonly used letter corresponding
to F for the spatial conjugate transform variable.  We
use K for this.  More specifically we assume a phase:
i(w t - k x) = 2 pi i(F t - K x).
and F, K define our notion of frequency, wavenumber.


 Credits:

CWP: Dave (algorithm), Jack (reformatting for SU)

 Trace header fields accessed: ns, dt, d2
 Trace header fields modified: tracl, ns, dt, trid, d1, f1, d2, f2
```

```
 SUSPECFX - Fourier SPECtrum (T -> F) of traces

 suspecfx <infile >outfile

 Note: To facilitate further processing, the sampling interval
       in frequency and first frequency (0) are set in the
output header.


 Credits:

CWP: Dave (algorithm), Jack (reformatting for SU)

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, dt, trid, d1, f1
```

```
SUSPECK1K2 - 2D (K1,K2) Fourier SPECtrum of (x1,x2) data set

suspeck1k2 <infile >outfile [optional parameters]

Optional parameters:

d1=from header(d1) or 1.0 spatial sampling interval in first (fast)
  dimension
d2=from header(d2) or 1.0 spatial sampling interval in second
(slow)  dimension

verbose=0 verbose = 1 echoes information

tmpdir=    if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Notes:
Because the data are assumed to be purely spatial (i.e. non-seismic),
the data are assumed to have trace id (30), corresponding to (z,x) data

To facilitate further processing, the sampling intervals in wavenumber
as well as the first frequency (0) and the first wavenumber are set in
the output header (as respectively d1, d2, f1, f2).

The relation: w = 2 pi F is well known for frequency, but there
doesn't seem to be a commonly used letter corresponding to F for the
spatial conjugate transform variables.  We use K1 and K2 for this.
More specifically we assume a phase:
-i(k1 x1 + k2 x2) = -2 pi i(K1 x1 + K2 x2).
and K1, K2 define our respective wavenumbers.


Credits:
    CWP: John Stockwell, 26 April 1995, based on original code by
         Dave Hale and Jack Cohen

Trace header fields accessed: ns, d1, d2, trid
Trace header fields modified: tracl, ns, dt, trid, d1, f1, d2, f2
```

```
 SUTAUP - forward and inverse T-X and F-K global slant stacks

    sutaup <infile >outfile  [optional parameters]

 Optional Parameters:
 option=1 =1 for forward F-K domian computation
=2 for forward T-X domain computation
=3 for inverse F-K domain computation
=4 for inverse T-X domain computation
 dt=tr.dt (from header)  time sampling interval (secs)
 nx=ntr   (counted from data) number of horizontal samples (traces)
 dx=1 horizontal sampling interval (m)
 npoints=71 number of points for rho filter
 pmin=0.0 minimum slope for Tau-P transform (s/m)
 pmax=.006 maximum slope for Tau-P transform (s/m)
 np=nx number of slopes for Tau-P transform
 ntau=nt number of time samples in Tau-P domain
 fmin=3 minimum frequency of interest
 xmin=0 offset on first trace

 verbose=0 verbose = 1 echoes information

 tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

 Notes:
 The cascade of a forward and inverse  tau-p transform preserves the
 relative amplitudes in a data panel, but not the absolute amplitudes
 meaning that a scale factor must be applied to data output by such a
 a cascade before the output may be compared to the original data.
 This is a characteristic of the algorithm employed in this program.
 (Suradon does not have this problem.)



 Credits: CWP: Gabriel Alvarez, 1995.

 Reference:
    Levin, F., editor, 1991, Slant-Stack Processing, Geophysics Reprint
         Series #14, SEG Press, Tulsa.
```

```
Trace header fields accessed: ns, dt
Trace header fields modified: dt,d2,f2
```

```
 SUWFFT - Weighted amplitude FFT with spectrum flattening 0->Nyquist

 suwfft <stdin | suifft >sdout

 Required parameters:
 none

 Optional parameters:
 w0=0.75 weight for AmpSpectrum[f-df]
 w1=1.00 weight for AmpSpectrum[f].. center value
 w2=0.75 weight for AmpSpectrum[f+df]

 Notes:
 1. output format is same as sufft
 2. suwfft | suifft is not quite a no-op since the trace
    length will usually be longer due to fft padding.
 3. using w0=0 w1=1 w2=0  gives truly flat spectrum, for other
    weight choices the spectrum retains some of its original topograpy

 Examples:
 1. boost data bandwidth to 10-90 Hz
     suwfft < data.su | suifft | sufilter f=5,8,90,100 | suximage
 1. view amplitude spectrum after flattening
     suwfft < data.su | suamp | suximage

 Caveat: The process of cascading the forward and inverse Fourier
  transforms may result in the output trace length being greater than
  the input trace length owing to zero padding. The user may wish to
  apply suwind to return the number of samples per trace to the original
  value:  Here NS is the number of samples per trace on the original data
   ... | suwind itmax=NS | ...

 Credits:

UHouston: Chris Liner

 Note: Concept from UTulsa PhD thesis of Bassel Al-Moughraby

 Trace header fields accessed: ns, dt
 Trace header fields modified: ns, d1, f1, trid
```

```
 SUZEROPHASE - convert input to zero phase equivalent

 suzerophase <stdin >stdout [optional parameters]

 Required parameters:
if dt is not set in header, then dt is mandatory

 Optional parameters:
     t0=1.0 time of peak value t0
verbose=0 =1 for advisory messages



 Credits:
c. 2011 Bruce VerWest as part of the SEAM project

 Trace header fields accessed: ns, dt, d1
```

```
 SURELANAN - REsiduaL-moveout semblance ANalysis for ANisotropic media

 surelan refl= npicks=    [optional parameters]

 Required parameters:
 reflector file: reflec =
 number of points in the reflector file =

 Optional Parameters:
 nr1=51 number of r1-parameter samples
 dr1=0.01              r1-parameter sampling interval
 fr1=-0.25             first value of r1-parameter
 nr2=51 number of r2-parameter samples
 dr2=0.01              r2-parameter sampling interval
 fr2=-0.25             first value of r2-parameter
 dzratio=5             ratio of output to input depth sampling intervals
 nsmooth=dzratio*2+1   length of semblance num and den smoothing window
 verbose=0             =1 for diagnostic print on stderr
 method=linear for linear interpolation of the interface
  =mono for monotonic cubic interpolation of interface
  =akima for Akima's cubic interpolation of interface
  =spline for cubic spline interpolation of interface

 Note:
 1. This program is part of Debashish Sarkar's anisotropic model building
 technique.
 2. Input migrated traces should be sorted by cdp - surelan outputs a
    group of semblance traces every time cdp changes.  Therefore, the
    output will be useful only if cdp gathers are input.
 3. The residual-moveout semblance for cdp gathers is based
on z(h)*z(h) = z(0)*z(0) + r1*h^2 + r2*h^4/[h^2+z(0)^2] where z
depth and h is the half-offset.
```

```
 SURELAN - compute residual-moveout semblance for cdp gathers based
on z(h)*z(h) = z(0)*z(0) + r*h*h where z depth and h offset.

 surelan <stdin >stdout   [optional parameters]

 Optional Parameters:
 nr=51 number of r-parameter samples
 dr=0.01               r-parameter sampling interval
 fr=-0.25               first value of b-parameter
 smute=1.5             samples with RMO stretch exceeding smute are zeroed
 dzratio=5             ratio of output to input depth sampling intervals
 nsmooth=dzratio*2+1   length of semblance num and den smoothing window
 verbose=0             =1 for diagnostic print on stderr

 Note:
 1. This program is part of Zhenyue Liu's velocity analysis technique.
 2. Input migrated traces should be sorted by cdp - surelan outputs a
    group of semblance traces every time cdp changes.  Therefore, the
    output will be useful only if cdp gathers are input.
 3. The parameter r may take negative values. The range of r can be
     controlled by maximum of (z(h)*z(h)-z(0)*z(0))/(h*h)
```

```
  SUTIVEL -  SU Transversely Isotropic velocity table builder
computes vnmo or vphase as a function of Thomsen's parameters and
theta and optionally interpolate to constant increments in slowness

 Optional Parameters:
 a=2500. alpha (vertical p velocity)
 b=1250. beta (vertical sv velocity)
 e=.20 epsilon (horiz p-wave anisotropy)
 d=.10 delta (strange parameter)
 maxangle=90.0 max angle in degrees
 nangle=9001 number of angles to compute
 verbose=0 set to 1 to see full listing
 np=8001 number of slowness values to output
 option=1 1=output vnmo(p) (result used for TI DMO)
2=output vnmo(theta) in degrees
3=output vnmo(theta) in radians
4=output vphase(p)
5=output vphase(theta) in degrees
6=output vphase(theta) in radians
7=output first derivative vphase(p)
8=output first derivative vphase(theta) in degrees
9=output first derivative vphase(theta) in radians
10=output second derivative vphase(p)
11=output second derivative vphase(theta) in degrees
12=output second derivative vphase(theta) in radians
13=( 1/vnmo(0)^2 -1/vnmo(theta)^2 )/p^2 test vs theta
   (result should be zero for all theta for d=e)
14=return vnmo(p) for weak anisotropy
 normalize=0 =1 means scale vnmo by cosine and scale vphase by
      1/sqrt(1+2*e*sin(theta)*sin(theta)
     (only useful for vphase when d=e for constant
result)
=0 means output vnmo or vphase unnormalized

 Output on standard output is ascii text with:
 line   1: number of values
 line   2: abscissa increment (p or theta increment, always starts at zero)
 line 3-n: one value per line



 Author: (visitor to CSM form Mobil) John E. Anderson, Spring 1994
```

```
 SUVEL2DF - compute stacking VELocity semblance for a single time in
    over Vnmo and eta in 2-D

    suvel2df <stdin >stdout [optional parameters]

 Required Parameters:
 tn zero-offset time of reflection
 offsetm Maximum offset considerd

 Optional Parameters:
 nv=50 number of velocities
 dv=50.0 velocity sampling interval
 fv=1500.0 first velocity
 nvh=50 number of horizotal velocities
 dvh=50.0 horizontal velocity sampling interval
 fvh=1500.0 first horizontal velocity
 xod=1.5 maximum offset-to-depth ratio to resolve
 dtratio=5 ratio of output to input time sampling intervals
 nsmooth=dtratio*2+1 length of semblance num and den smoothing window
 verbose=0 =1 for diagnostic print on stderr
 vavg=fv+0.5*(nv-1)*dv   average velocity used in the search

 Notes:
 Semblance is defined by the following quotient:

 n-1
[ sum q(t,j) ]^2
 j=0
s(t) = ------------------
 n-1
n sum [q(t,j)]^2
 j=0

 where n is the number of non-zero samples after muting.
 Smoothing (nsmooth) is applied separately to the numerator and denominator
 before computing this semblance quotient.

 Input traces should be sorted by cdp - suvel2df outputs a group of
 semblance traces every time cdp changes.  Therefore, the output will
 be useful only if cdp gathers are input.


 Credits:
```

```
CWP: Tariq Alkhalifah,  February 1997
 Trace header fields accessed:  ns, dt, delrt, offset, cdp.
 Trace header fields modified:  ns, dt, offset.
```

```
 SUVELAN_NCCS - compute stacking VELocity panel for cdp gathers
using Normalized CrossCorrelation Sum

 suvelan_uccs <stdin >stdout [optional parameters]

 Optional Parameters:
 nx=tr.cdpt              number of traces in cdp
 nv=50                   number of velocities
 dv=50.0                 velocity sampling interval
 fv=1500.0               first velocity
 smute=1.5               samples with NMO stretch exceeding smute are zeroed
 dtratio=5               ratio of output to input time sampling intervals
 nsmooth=dtratio*2+1     length of smoothing window
 verbose=0               =1 for diagnostic print on stderr
 pwr=1.0                 semblance value to the power

 Notes:
 Normalized CrossCorrelation sum: sum all possible crosscorrelation
 trace pairs in a CMP gather for each trial velocity and zero-offset
 two-way travel time inside a time window. This coherence measure is
 normalized by dividing each crosscorrelation trace pair by the geometric
 mean of the energy, inside the chosen time window, of each trace pair
 involved in each crosscorrelation. Then, to achieve a maximum amplitude
 of unity, the result is multiplied by  2/(M(M-1)), which is the inverse
 of the total number of crosscorrelation. The normalization allows to
 bring out weak reflection as long as these reflections have moveouts close
 to a hyperbola.



 Credits:

 CWP:  Valmore Celis, Sept 2002

 Based on the original code: suvelan.c
    Colorado School of Mines:  Dave Hale, c. 1989

 Trace header fields accessed:  ns, dt, delrt, offset, cdp, cdpt
 Trace header fields modified:  ns, dt, offset, cdp

 Reference: Neidell, N.S., and Taner, M.T., 1971, Semblance and
             other coherency measures for multichannel data:
             Geophysics, 36, 498-509.
```

```
 SUVELAN_NSEL - compute stacking VELocity panel for cdp gathers
using the Normalized Selective CrossCorrelation sum

 suvelan_usel <stdin >stdout [optional parameters]

 Optional Parameters:
 nx=tr.cdpt              number of traces in cdp
 dx=tr.d2            offset increment
 nv=50                   number of velocities
 dv=100.0                velocity sampling interval
 fv=1500.0               first velocity
 tau=0.5                 threshold for significance values
 smute=1.5               samples with NMO stretch exceeding smute are zeroed
 dtratio=5               ratio of output to input time sampling intervals
 nsmooth=dtratio*2+1     length of smoothing window
 verbose=0               =1 for diagnostic print on stderr
 pwr=1.0                 semblance value to the power

 Notes:
 Normalized Selective CrossCorrelation Sum: is based on the coherence
 measure known as crosscorrelation sum. The difference is that the selective
 approach sum only crosscorrelation pairs with relatively large differential
 moveout, thus increasing the resolving power in the velocity spectra
 compared to that achieved by conventional methods. The normalization is
 achieved in much the same way of normalizing the conventional
 crosscorrelation sum.

 Each crosscorrelation is divided by the geometric mean
 of the energy of the traces involved, and the multiplying by a constant to
 achieve maximum amplitude of unity. The constant is just the inverse of the
 total number of crosscorrelations included in the sum.  The selection is
 made using a parabolic approximation of the differential moveout and
 imposing a threshold for those differential moveouts.

 That threshold is the parameter tau in this program, which varies between 0
 to 1.  A value of tau=0, means conventional crosscorrelation sum is applied
 implying that all crosscorrelations are included in the sum. In contrast,
 a value of tau=1 (not recomended) means that only the crosscorrelation
 formed by the trace pair involving the shortest and longest offset is
 included in the sum. Intermediate values will produce percentages of the
 crosscorrelations included in the sum that will be shown in the screen
 before computing the velocity spectra. Typical values for tau are between
 0.2 and 0.6, producing approximated percentages of crosscorrelations summed
```

between 60% and 20%. The higher the value of tau the lower the percentage and higher the increase in the resolving power of velocity spectra.

Keeping the percentage of crosscorrelations included in the sum between 20% and 60% will increase resolution and avoid the precense of artifacts in the results. In data contaminated by random noise or statics distortions is recomended to mantaing the percentage of crosscorrelations included in the sum above 25%. After computing the velocity spectra one might want to adjust the level and number of contours before velocity picking.

Credits: CWP: Valmore Celis, Sept 2002

Based on the original code: suvelan.c
Colorado School of Mines: Dave Hale c. 1989

References:
Neidell, N.S., and Taner, M.T., 1971, Semblance and other coherency measures for multichannel data: Geophysics, 36, 498-509.
Celis, V. T., 2002, Selective-correlation velocity analysis: CSM thesis.

Trace header fields accessed: ns, dt, delrt, offset, cdp
Trace header fields modified: ns, dt, offset, cdp

```
SUVELAN - compute stacking velocity semblance for cdp gathers

suvelan <stdin >stdout [optional parameters]

Optional Parameters:
nv=50                   number of velocities
dv=50.0                 velocity sampling interval
fv=1500.0               first velocity
anis1=0.0               quartic term, numerator of an extended quartic term
anis2=0.0               in denominator of an extended quartic term
smute=1.5               samples with NMO stretch exceeding smute are zeroed
dtratio=5               ratio of output to input time sampling intervals
nsmooth=dtratio*2+1     length of semblance num and den smoothing window
verbose=0               =1 for diagnostic print on stderr
pwr=1.0                 semblance value to the power

Notes:
Velocity analysis is usually a two-dimensional screen for optimal values of
the vertical two-way traveltime and stacking velocity. But if the travel-
time curve is no longer close to a hyperbola, the quartic term of the
traveltime series should be considered. In its easiest form (with anis2=0)
the optimizion of all parameters requires a three-dimensional screen. This
is done by a repetition of the conventional two-dimensional screen with a
variation of the quartic term. The extended quartic term is more accurate,
though the function is no more a polynomial. When screening for optimal
values the theoretical dependencies between these paramters can be taken
into account. The traveltime function is defined by

               1            anis1
t^2 = t_0^2 + --- x^2 + ------------- x^4
              v^2       1 + anis2 x^2

The coefficients anis1, anis2 are assumed to be small, that means the non-
hyperbolicity is assumed to be small. Triplications cannot be handled.

Semblance is defined by the following quotient:

                n-1
               [ sum q(t,j) ]^2
                j=0
      s(t) = ------------------
                n-1
               n sum [q(t,j)]^2
```

```
                j=0

 where n is the number of non-zero samples after muting.
 Smoothing (nsmooth) is applied separately to the numerator and denominator
 before computing this semblance quotient.

 Then, the semblance is set to the power of the parameter pwr. With pwr > 1
 the difference between semblance values is stretched in the upper half of
 the range of semblance values [0,1], but compressed in the lower half of
 it; thus, the few large semblance values are enhanced. With pwr < 1 the
 many small values are enhanced, thus more discernible against background
 noise. Of course, always at the expanse of the respective other feature.

 Input traces should be sorted by cdp - suvelan outputs a group of
 semblance traces every time cdp changes.  Therefore, the output will
 be useful only if cdp gathers are input.

 Credits:
CWP, Colorado School of Mines:
           Dave Hale (everything except ...)
           Bjoern Rommel (... the quartic term)
      SINTEF, IKU Petroleumsforskning
           Bjoern Rommel (... the power-of-semblance function)

 Trace header fields accessed:  ns, dt, delrt, offset, cdp
 Trace header fields modified:  ns, dt, offset, cdp
```

```
 SUVELAN_UCCS - compute stacking VELocity panel for cdp gathers
using UnNormalized CrossCorrelation Sum

 suvelan_uccs <stdin >stdout [optional parameters]

 Optional Parameters:
 nx=tr.cdpt              number of traces in cdp
 nv=50                   number of velocities
 dv=50.0                velocity sampling interval
 fv=1500.0               first velocity
 smute=1.5               samples with NMO stretch exceeding smute are zeroed
 dtratio=5               ratio of output to input time sampling intervals
 nsmooth=dtratio*2+1     length of smoothing window
 verbose=0               =1 for diagnostic print on stderr
 pwr=1.0                 semblance value to the power

Notes:
 Unnormalized crosscorrelation sum: sum all possible crosscorrelation trace
 pairs in a CMP gather for each trial velocity and zero-offset two-way
 travel time inside a time window. This unnormalized coherency measure
 produces large spectral amplitudes for strong reflections and small
 spectral amplitudes for weaker ones. If M is the number of traces in the
 CMP gather M(M-1)/2 is the total number of crosscorrelations for each trial
 velocity and zero-offset two-way traveltime.


 Credits: CWP: Valmore Celis, Sept 2002

 Based on the original code: suvelan.c
    Colorado School of Mines:  Dave Hale c. 1989


 Reference: Neidell, N.S., and Taner, M.T., 1971, Semblance and other
            coherency measures for multichannel data: Geophysics, 36, 498-509.

 Trace header fields accessed:  ns, dt, delrt, offset, cdp
 Trace header fields modified:  ns, dt, offset, cdp
```

```
 SUVELAN_USEL - compute stacking velocity panel for cdp gathers
using the UnNormalized Selective CrossCorrelation Sum

 suvelan_usel <stdin >stdout [optional parameters]

 Optional Parameters:
 nx=tr.cdpt              number of traces in cdp
 dx=tr.d2                offset increment
 nv=50                   number of velocities
 dv=50.0                 velocity sampling interval
 fv=1500.0               first velocity
 tau=0.5                 threshold for significance values
 smute=1.5               samples with NMO stretch exceeding smute are zeroed
 dtratio=5               ratio of output to input time sampling intervals
 nsmooth=dtratio*2+1     length of smoothing window
 verbose=0               =1 for diagnostic print on stderr
 pwr=1.0                 semblance value to the power

 Notes:
 UnNormalized Selective CrossCorrelation sum: is based on the coherence
 measure known as crosscorrelation sum. The difference is that the selective
 approach sum only crosscorrelation pairs with relatively large differential
 moveout, thus increasing the resolving power in the velocity spectra
 compared to that achieved by conventional methods.  The selection is made
 using a parabolic approximation of the differential moveout and imposing a
 threshold for those differential moveouts.

 That threshold is the parameter tau in this program, which varies between
 0 to 1.  A value of tau=0, means conventional crosscorrelation sum is
 applied implying that all crosscorrelations are included in the sum. In
 contrast, a value of tau=1 (not recomended) means that only the
 crosscorrelation formed by the trace pair involving the shortest and longest
 offset is included in the sum. Intermediate values will produce percentages
 of the crosscorrelations included in the sum that will be shown in the
 screen before computing the velocity spectra. Typical values for tau are
 between 0.2 and 0.6, producing approximated percentages of crosscorrelations
 summed between 60% and 20%. The higher the value of tau the lower the
 percentage and higher the increase in the resolving power of velocity
 spectra.

 Keeping the percentage of crosscorrelations included in the sum between 20%
 and 60% will increase resolution and avoid the precense of artifacts in the
 results.  In data contaminated by random noise or statics distortions is
```

recomended to mantaing the percentage of crosscorrelations included in the sum above 25%. After computing the velocity spectra one might want to adjust the level and number of contours before velocity picking.

Credits: CWP: Valmore Celis, Sept 2002

Based on the original code: suvelan_.c
   Colorado School of Mines: Dave Hale c. 1989

References:
Neidell, N.S., and Taner, M.T., 1971, Semblance and other coherency
            measures for multichannel data: Geophysics, 36, 498-509.
Celis, V. T., 2002, Selective-correlation velocity analysis: CSM thesis.

Trace header fields accessed: ns, dt, delrt, offset, cdp
Trace header fields modified: ns, dt, offset, cdp

```
 LAS2SU - convert las2 format well log curves to su traces

  las2su <stdin nskip= ncurve= >stdout [optional params]

 Required parameters:
 none
 Optional parameters:
 ncurve=automatic number of log curves in the las file
 dz=0.5 input depth sampling (ft)
 m=0 output (d1,f1) in feet
=1 output (d1,f1) in meters
 ss=0 do not subsample (unless nz > 32767 )
=1 pass every other sample
 verbose=0 =1 to echo las header lines to screen
 outhdr=las_header.asc name of file for las headers

 Notes:
 1. It is recommended to run LAS_CERTIFY available from CWLS
    at http://cwls.org.
 2. First log curve MUST BE depth.
 3. If number of depth levels > 32767 (segy NT limit)
    then input is subsampled by factor of 2 or 4 as needed
 4. Logs may be isolated using tracl header word (1,2,...,ncurve)
    tracl=1 is depth

 If the input LAS file contains sonic data as delta t or interval
 transit time and you plan to use these data for generating a
 reflection coefficient time series in suwellrf, convert the sonic
 trace to velocity using suop with op=s2v (sonic to velocity)
 before input of the velocity trace to suwellrf. ",

 Caveat: ",
 No trap is made for the commonly used null value in LAS files
 (-999.25). The null value will be output as ?999.25, which
 really messes up a suxwigb display of density data because the
 ?999.25 skews the more or less 2.5 values of density.
 The user needs to edit out null values (-999.25) before running
 other programs, such as "suwellrf".


 Credits:
  * Chris Liner based on code by Ferhan Ahmed and a2b.c (June 2005)
  *            (Based on code by Ferhan Ahmed and a2b.c)
```

```
*            I gratefully acknowledge Saudi Aramco for permission
*            to release this code developed while I worked for the
*            EXPEC-ARC research division.
* CWP: John Stockwell 31 Oct 2006, combining lasstrip and
* CENPET: lasstrip 2006 by Werner Heigl
*
*     Rob Hardy: allow the ncurve parameter to work correctly if set
*    - change string length to 400 characters to allow more curves
*    - note nskip in header is totally ignored !
*
* Ideas for improvement:
* add option to chop off part of logs (often shallow
*    portions are not of interest
* cross check sampling interval from header against
*    values found from first log curve (=depth)
*
```

```
 SUBACKUSH - calculate Thomsen anisotropy parameters from
       well log (vp,vs,rho) data and optionally include
       intrinsic VTI shale layers based on gramma ray log
       via BACKUS averaging
 subackush < vp_vs_rho.su >stdout [options]
 subackush < vp_vs_rho_gr.su  gr=1 >stdout [options]

 Required parameters:
 none

 Optional parameter:
 navg=101 number of depth samples in Backus avg window

  Intrinsic anisotropy of shale layers can be included ...
 gr=0 no gamma ray log input for shale
=1 input is vp_vs_rho_gr
 grs=100 pure shale gamma ray value (API units)
 grc=10 0% shale gamma ray value (API units)
 smode=1 include shale anis params prop to shale volume
=0 include shale anis for pure shale only
 se=0.209 shale epsilon (Thomsen parameter)
 sd=0.033 shale delta (Thomsen parameter)
 sg=0.203 shale gamma (Thomsen parameter)

 Notes:
 1. Input are (vp,vs,rho) traces in metric units
 2. Output are
    tracl =(1,2,3,4,5,6)
    quantity =(vp0,vs0,<rho>,epsilon,delta,gamma)
    units =(m/s,m/s,kg/m^3,nd,nd,nd) nd=dimensionless
    tracl =(7,8)
    quantity =(Vsh,shaleEps) Vsh=shale volume fraction
    units =(nd,nd)
 3. (epsilon,delta,etc.) can be isolated by tracl header field
 4. (vp0,vs0) are backus averaged vertical wavespeeds
 5. <rho> is backus averaged density, etc.

 Example:
 las2su < logs.las nskip=34 nlog=4 > logs.su
 suwind < logs.su  key=tracl min=2 max=3 | suop op=s2vm > v.su
 suwind < logs.su  key=tracl min=4 max=4 | suop op=d2m > d.su
 fcat v.su d.su > vp_vs_rho.su
 subackus < vp_vs_rho.su > vp0_vs0_rho_eps_delta_gamma.su
```

```
 In this example we start with a well las file containing
 34 header lines and 4 log tracks (depth,p_son,s_son,den).
 This is converted to su format by las2su.  Then we pull off
 the sonic logs and convert them to velocity in metric units.
 Then the density log is pulled off and converted to metric.
 All three metric curves are bundled into one su file which
 is the input to subackus.  ",

 Related programs: subackus, sulprime


 Credits:

UHouston: Chris Liner
              I gratefully acknowledge Saudi Aramco for permission
              to release this code developed while I worked for the
              EXPEC-ARC research division.

 References:
 Anisotropy parameters: Thomsen, 2002, DISC Notes (SEG)
 Backus Method: Berryman, Grechka, and Berge, 1997, SEP94
 Shale params: Wang, 2002, Geophysics, p. 1427
```

```
 SUBACKUS - calculate Thomsen anisotropy parameters from
       well log (vp,vs,rho) data via BACKUS averaging

 subackus < vp_vs_rho.su >stdout [options]

 Required parameters:
 none

 Optional parameter:
 navg=201 number of depth samples in Backus avg window
 all=0 =1 to output extra parameters
(<vp0>,<vs0>,<rho>,eta,vang,a,f,c,l,A,B,<lam>,<mu>)
 ang=30 angle (deg) for use in vang

 Notes:
 1. Input are (vp,vs,rho) traces in metric units
 2. Output are (epsilon,delta,gamma) dimensionless traces
    tracl=(1,2,3)=(epsilon,delta,gamma)
(all=1 output optional traces listed below)
    tracl=(4,5,6,7,8)=(vp0,vs0,<rho>,eta,vang)
    tracl=(9,10,11,12)=(a,f,c,l)=(c11,c13,c33,c44) backus avg'd
    tracl=(13,14)=(<lam/lamp2mu>^2,4<mu*lampmu/lamp2mu>)=(A,B)
       used to analyze eps=(a-c)/2c; a=c11=A*x+B;  c=c33=x
    tracl=(15,16)=(<lambda>,<mu>)
       for fluid analysis (lambda affected by fluid, mu not)
    tracl=(17,18,19)=(vp,vs,rho)  orig log values
    tracl=(20)=(m)=(c66) Backus avg'd
    tracl=(21,22,23,24,25)=(a,f,c,l,m)=(c11,c13,c33,c44,c66) orig
 3. (epsilon,delta,etc.) can be isolated by tracl header field
 4. (vp0,vs0) are backus averaged vertical wavespeeds
 5. <rho> is backus averaged density, etc.
 6. eta = (eps - delta) / (1 + 2 delta)
 7. vang=vp(ang_deg)/vp0  phase velocity ratio
    The idea being that if vang~vp0 there are small time effects
    (30 deg comes from ~ max ray angle preserved in processing)

 Example:
 las2su < logs.las nskip=34 nlog=4 > logs.su
 suwind < logs.su  key=tracl min=2 max=3 | suop op=s2vm > v.su
 suwind < logs.su  key=tracl min=4 max=4 | suop op=d2m > d.su
 fcat v.su d.su > vp_vs_rho.su
 subackus < vp_vs_rho.su > eps_delta_gamma.su
 In this example we start with a well las file containing
```

```
 34 header lines and 4 log tracks (depth,p_son,s_son,den).
 This is converted to su format by las2su.  Then we pull off
 the sonic logs and convert them to velocity in metric units.
 Then the density log is pulled off and converted to metric.
 All three metric curves are bundled into one su file which
 is the input to subackus.  ",

 Related codes: sulprime subackush

 Credits:

UHouston: Chris Liner
              I gratefully acknowledge Saudi Aramco for permission
              to release this code developed while I worked for the
              EXPEC-ARC research division.
 References:
 Anisotropy parameters: Thomsen, 2002, DISC Notes (SEG)
 Backus Method: Berryman, Grechka, and Berge, 1997, SEP94
```

```
 SUGASSMAN - Model reflectivity change with rock/fluid properties

sugassman [optional parameters] > data.su

 Optional parameters:
 nt=500  number of time samples
 ntr=200 number of traces
 dt=0.004  time sampling interval in seconds
 mode=0 model isolated gassmann refl coefficient
=1 embed gassmann RC in random RC series
=2 R0 parameter sensitivity output
 p=0.15  parameter sensitivity test range (if mode=2)
 .... Environment variables ...
 temp=140  Temperature in degrees C
 pres=20  Pressure in megaPascals
 .... Caprock variables ....
 v1=37900  caprock P-wave speed (m/s)
 r1=44300  caprock mass density (g/cc)
 .... Reservoir fluid variables ....
 g=0.56 Gas specific gravity 0.56 (methane)-1.8 (condensate)
 api=50  Gas specific gravity 10 (heavy)-50 (ultra light)
 s=35 Brine salinity in ppm/(1000 000
 so=.7  Oil saturation (0-1)
 sg=.2  Gas saturation (0-1)
 .... Reservoir rock frame variables ....
 kmin=37900  Bulk modulus (MPa) of rock frame mineral(s) [default=quartz]
 mumin=44300  Shear modulus (MPa) of rock frame mineral(s) [default=quartz]
 rmin=2.67  Mass density (g/cc) of rock frame mineral(s) [default=quartz]
 phi=.24  Rock frame porosity (0-1)
 a=1  Fitting parameters: Mdry/Mmineral ~ 1/(a + b phi^c)
 b=15  ... where M is P-wave modulus and defaults are for
 c=1  ... Glenn sandstone [see Liner (2nd ed, table 26.2)]
h=20  Reservoir thickness (m)

 Notes:
 Creates a reflection coefficient series based on Gassmann
 theory of velocity and density for porous elastic media


 Credits: UHouston: Chris Liner 9/23/2009

 trace header fields set:
```

```
 SULPRIME - find appropriate Backus average length for
  a given log suite, frequency, and purpose

 sulprime < vp_vs_rho.su  [options] or
 sulprime < vp_vs_rho_gr.su   [options]

 Required parameters:
 none

 Optional parameter:
 b=2.0 target value of Backus number
b=2 is transmission limit (ok for proc, mig, etc.)
b=0.3 is scattering limit (ok for modeling)
 dz=1 input depth sample interval (ft)
 f=60 frequency (Hz)... dominant or max (to be safe)
 nmax=301 maximum averaging length (samples)
 verbose=1 print intermediate results
=0 print final result only

 Notes:
 1. input is in sync with subackus, but vp and gr not used
     (gr= gamma ray log)
 Related codes:  subackus, subackush


 Credits:
UHouston: Chris Liner Sept 2008
              I gratefully acknowledge Saudi Aramco for permission
              to release this code developed while I worked for the
              EXPEC-ARC research division.
 Reference:
     The Backus Number (Liner and Fei, 2007, TLE)
```

```
 SUWELLRF - convert WELL log depth, velocity, density data into a
uniformly sampled normal incidence Reflectivity Function of time

 suwellrf [required parameters] [optional parameters] > [stdout]

 Required Parameters:
 dvrfile= file containing depth, velocity, and density values
 ...or...
 dvfile= file containing depth and velocity values
 drfile= file containing depth and density values
 ...or...
 dfile= file containing depth values
 vfile= file containing velocity log values
 rhofile= file containing density log values
 nval=  number of triplets of d,v,r values if dvrfile is set,
  number of pairs of d,v and d,r values dvfile and drfile
are set, or number of values if dfile, vfile, and rhofile
are set.

 Optional Parameters:
 dtout=.004 desired time sampling interval (sec) in output
 ntr=1         number of traces to output

 Notes:
 The format of the input file(s) is C-style binary float. These files
 may be constructed from ascii file via:

       a2b n1=3 < dvrfile.ascii > dvrfile.bin
 ...or...
       a2b n1=2 < dvfile.ascii > dvfile.bin
       a2b n1=2 < drfile.ascii > drfile.bin
 ...or...
       a2b n1=1 < dfile.ascii > dfile.bin
       a2b n1=1 < vfile.ascii > dfile.bin
       a2b n1=1 < rhofile.ascii > rhofile.bin

 A raw normal-incidence impedence reflectivity as a function of time is
 is generated using the smallest two-way traveltime implied by the
 input velocities as the time sampling interval. This raw reflectivity
 trace is then resampled to the desired output time sampling interval
 via 8 point sinc interpolation. If the number of samples on the output
 exceeds SU_NFLTS the output trace will be truncated to that value.
```

Caveat:
This program is really only a first rough attempt at creating a well log utility. User input and modifications are welcome.

See also: suresamp

Author: CWP: John Stockwell, Summer 2001, updated Summer 2002.
inspired by a project by GP grad student Leo Brown

```
 SUCOMMAND - pipe traces having the same key header word to command

     sucommand <stdin >stdout [Optional parameters]

 Required parameters:
  none

 Optional parameters:
  verbose=0 wordy output
  key=cdp header key word to pipe on
  command="suop nop"    command piped into
dir=0 0:  change of header key
-1: break in monotonic decrease of header key
+1: break in monotonic increase of header key


Notes:
 This program permits limited parallel processing capability by opening
 pipes for processes, signaled by a change in key header word value.



 Credits:
VT: Matthias Imhof

 Note:
The "valxxx" subroutines are in su/lib/valpkge.c.  In particular,
      "valcmp" shares the annoying attribute of "strcmp" that
if (valcmp(type, val, valnew) {
...
}
will be performed when val and valnew are different.
```

```
 SUGETGTHR - Gets su files from a directory and put them
             throught the unix pipe. This creates continous data flow.

  sugetgthr  <stdin >sdout

 Required parameters:

 dir=            Name of directory to fetch data from
            Every file in the directory is treated as an su file
 Optional parameters:
 verbose=0 =1 more chatty
 vt=0 =1 allows gathers with variable length traces
  no header checking is done!
 ns= must be specified if vt=1; number of samples to read




segy tr;

int
main(int argc, char **argv)
{

cwp_String dir=""; /* input directory containng the gathers
char *fname=NULL;
char *ffname=NULL;

DIR *dp=NULL;
struct dirent *d=NULL;
struct stat __st;
FILE *fp=NULL;
int fd=0;

int verbose;
int vt=0;
ssize_t nread;
int ns=0;


/* Initialize
initargs(argc, argv);
requestdoc(1);
```

```
        /* Get parameters
        MUSTGETPARSTRING("dir", &dir);
        if (!getparint   ("verbose", &verbose)) verbose = 0;
        if (!getparint   ("vt", &vt)) vt = 0;
if(vt)MUSTGETPARINT("ns",&ns);
        checkpars();

/* Open the directory
if ((dp = opendir(dir)) == NULL)
err(" %s directory not found\n",dir);

/* For each file in directory
while (( d = readdir(dp)) !=NULL) {

fname = ealloc1(strlen(d->d_name)+1,sizeof(char));
strcpy(fname,d->d_name);

/* Skip . and .. directory entries
if(strcmp(fname,".") && strcmp(fname,"..")) {
ffname = ealloc1(strlen(d->d_name)+strlen(dir)+2,sizeof(char));

/* Create full filename
sprintf(ffname, "%s/%s",dir,fname);
if(verbose==1) warn("%s",ffname);

/* get some info from the file
stat(ffname,&__st);
if(__st.st_size > 0) {

/* Open the file and read traces into stdout*/
  if(vt) {
fd = open(ffname,O_RDONLY|CWP_O_LARGEFILE);
/* nread=fread(&tr,(size_t) HDRBYTES,1,fp);
nread=read(fd,&tr,(size_t) HDRBYTES);
memset((void *) &tr.data[tr.ns], (int) '\0' ,MAX(ns-tr.ns,0)*FSIZE);
/* nread+=fread(&tr.data[0],(size_t) tr.ns*FSIZE,1,fp);
nread+=read(fd,&tr.data[0],(size_t) tr.ns*FSIZE);
} else {
fp = efopen(ffname, "r");
nread=fgettr(fp, &tr);
}
do {
```

```
if(vt) {
tr.ns=ns;
fwrite(&tr,ns*FSIZE+HDRBYTES,1,stdout);
} else {
puttr(&tr);
}
if(vt) {
/* nread=fread(&tr,(size_t) HDRBYTES,1,fp);
nread=read(fd,&tr,(size_t) HDRBYTES);
memset((void *) &tr.data[tr.ns], (int) '\0' ,MAX(ns-tr.ns,0)*FSIZE);
/* nread+=fread(&tr.data[0],(size_t) tr.ns*FSIZE,1,fp);
nread+=read(fd,&tr.data[0],(size_t) tr.ns*FSIZE);
} else {
nread=fgettr(fp, &tr);
}
} while(nread);
if(vt) close(fd);
else efclose(fp);
} else {
warn(" File %s has zero size, skipped.\n",ffname);
}
free1(ffname);
}
free1(fname);

}
closedir(dp);
        return(CWP_Exit());
}
```

```
SUGPRFB - SU program to remove First Breaks from GPR data

  sugprfb < radar traces >outfile

 nx=51 number of traces to sum to create pilot trace (odd)
 fbt=60 length of first break in number of samples

 Notes:
 The first fbt samples from nx traces are stacked to form a pilot
 first break trace, this is fitted to the actual traces by shifting
 and scaling.  The nx traces long spatial window is
 slided along the section and a new pilot traces is formed for each
 position. The scalers in percent and the time shifts are stored in
 header words trwf and grnors.


 Segy data constans
segy  tr; /* SEGY trace

void remove_fb(float *data,float *wavelet,int n,short *scaler,short *shft);

int
main( int argc, char *argv[] )
{


int nx;
int fbt;
int nt;

float *stacked=NULL;
int *nnz=NULL;
int itr=0;


initargs(argc, argv);
    requestdoc(1);

if (!getparint("nx", &nx)) nx = 51;
if( !ISODD(nx) ) {
nx++;
warn(" nx has been changed to %d to be odd.\n",nx);
}
```

```
if (!getparint("fbt", &fbt)) fbt = 60;
        checkpars();

/* Get info from first trace
if (!gettr(&tr))  err("can't get first trace");
nt = tr.ns;

stacked = ealloc1float(fbt);
nnz = ealloc1int(fbt);
memset((void *) nnz, (int) '\0', fbt*ISIZE);
memset((void *) stacked, (int) '\0', fbt*FSIZE);

/* read nx traces and stack them
/* The first trace is already read

{ int i,it;
  float **tr_b;
  char  **hdr_b;
  int NXP2=nx/2;
  short shft,scaler;

/* ramp on read the first nx traces and create stack

   tr_b = ealloc2float(nt,nx);
hdr_b = (char**)ealloc2(HDRBYTES,nx,sizeof(char));

memcpy((void *) hdr_b[0], (const void *) &tr, HDRBYTES);
memcpy((void *) tr_b[0], (const void *) &tr.data, nt*FSIZE);

for(i=1;i<nx;i++) {
gettr(&tr);
memcpy((void *) hdr_b[i], (const void *) &tr, HDRBYTES);
memcpy((void *) tr_b[i], (const void *) &tr.data, nt*FSIZE);
}

for(i=0;i<nx;i++)
for(it=0;it<fbt;it++)
stacked[it] += tr_b[i][it];


for(it=0;it<fbt;it++)
stacked[it] /=(float)nx;
```

```
/* filter and write out the first nx/2 +1 traces
for(i=0;i<NXP2+1;i++) {
memcpy((void *) &tr, (const void *) hdr_b[i], HDRBYTES);
memcpy((void *) tr.data, (const void *) tr_b[i], nt*FSIZE);

remove_fb(tr.data,stacked,fbt,&scaler,&shft);
tr.trwf = scaler;
tr.grnors = shft;

puttr(&tr);
++itr;
}

/* do the rest of the traces
gettr(&tr);

do {

/* Update the stacked trace  - remove old
for(it=0;it<fbt;it++)
stacked[it] -= tr_b[0][it]/(float)nx;

/* Bump up the storage arrays
/* This is not very efficient , but good enough
{int ib;
for(ib=1;ib<nx;ib++) {
    memcpy((void *) hdr_b[ib-1],
(const void *) hdr_b[ib], HDRBYTES);
memcpy((void *) tr_b[ib-1],
(const void *) tr_b[ib], nt*FSIZE);
}
}

/* Store the new trace
memcpy((void *) hdr_b[nx-1], (const void *) &tr, HDRBYTES);
memcpy((void *) tr_b[nx-1], (const void *) &tr.data, nt*FSIZE);

/* Update the stacked array  - add new
for(it=0;it<fbt;it++)
stacked[it] += tr_b[nx-1][it]/(float)nx;
```

```
/* Filter and write out the middle one NXP2+1
memcpy((void *) &tr, (const void *) hdr_b[NXP2], HDRBYTES);
memcpy((void *) tr.data, (const void *) tr_b[NXP2], nt*FSIZE);

remove_fb(tr.data,stacked,fbt,&scaler,&shft);

tr.trwf = scaler;
tr.grnors = shft;
puttr(&tr);
++itr;


} while(gettr(&tr));

/* Ramp out - write ot the rest of the traces
/* filter and write out the last nx/2 traces
for(i=NXP2+1;i<nx;i++) {
memcpy((void *) &tr, (const void *) hdr_b[i], HDRBYTES);
memcpy((void *) tr.data, (const void *) tr_b[i], nt*FSIZE);

remove_fb(tr.data,stacked,fbt,&scaler,&shft);

tr.trwf = scaler;
tr.grnors = shft;
puttr(&tr);
itr++;

}


}



free1float(stacked);
free1int(nnz);
    return EXIT_SUCCESS;
}

void remove_fb(float *yp,float *ym,int n,short *scaler,short *shft)
 Find Scale and timeshift

Yp = data trace
```

```
Ym = wavelet

F=min(Yp(x) - Ym(x)*a)^2 is a function to minimize for scaler

find shift first with xcorrelation then

solve for a - scaler
/



{

define RAMPR 5

void find_p(float *ym,float *yp,float *a,int *b,int n);

int it,ir;

float a=1.0;
int b=0;
int ramps;
float *a_ramp;

ramps=NINT(n-n/RAMPR);

find_p(ym,yp,&a,&b,n);

a_ramp = ealloc1float(n);

for(it=0;it<ramps;it++)
a_ramp[it]=1.0;

for(it=ramps,ir=0;it<n;it++,ir++) {
a_ramp[it]=1.0-(float)ir/(float)(n-ramps-1);
}



if (b<0) {
for(it=0;it<n+b;it++)
yp[it-b] -=ym[it]*a*a_ramp[it-b];
} else {
```

```
for(it=0;it<n-b;it++)
yp[it] -=ym[it-b]*a*a_ramp[it+b];
}

*scaler = NINT((1.0-a)*100.0);
*shft = b;

free1float(a_ramp);
}

void find_p(float *ym,float *yp,float *a,int *b,int n)
{
define NP 1

float *y,**x;
int n_s;

int k;
float *res;
int jpvt[NP];
float qraux[NP];
float work[NP];


x = ealloc2float(n,1);
y = ealloc1float(n);
res = ealloc1float(n);

memcpy((void *) &x[0][0], (const void *) &ym[0], n*FSIZE);
memset((void *) y, (int) '\0', n*FSIZE);

/* Solve for shift
  xcor (n,0,ym,n,0,yp,n,-n/2,y);
/* pick the maximum
*b = -max_index(n,y,1)+n/2;

n_s = n-abs(*b);

if (*b < 0) {
memcpy((void *) &x[0][0], (const void *) &ym[*b], n_s*FSIZE);
} else {
memcpy((void *) &x[0][*b], (const void *) &ym[0], n_s*FSIZE);
```

```
}
/* Solve for scaler
sqrst(x, n_s, 1,yp,0.0,a,res,&k,&jpvt[0],&qraux[0],&work[0]);



free2float(x);
free1float(res);
free1float(y);
}
```

```
 SUKILL - zero out traces

 sukill <stdin >stdout [optional parameters]

 Optional parameters:
key=trid header name to select traces to kill
a=2 header value identifying tracces to kill
 or
  min=  first trace to kill (one-based)
  count=1 number of traces to kill

 Notes:
If min= is set it overrides selecting traces by header.


 Credits:
CWP: Chris Liner, Jack K. Cohen
header-based trace selection: Florian Bleibinhaus

 Trace header fields accessed: ns
```

```
 SUMIXGATHERS - mix two gathers

 sumixgathers file1 file2 > stdout [optional parameters]

 Required Parameters:
 ntr=tr.ntr if ntr header field is not set, then ntr is mandatory

 Optional Parameters: none

 Notes: Both files have to be sorted by offset
 Mixes two gathers keeping only the traces of the first file
 if the offset is the same. The purpose is to substitute only
 traces non existing in file1 by traces interpolated store in file2.  ",

 Example. If file1 is original data file and file 2 is obtained by
 resampling with Radon transform, then the output contains original
 traces with gaps filled



 Credits:
Daniel Trad. UBC
 Trace header fields accessed: ns, dt, ntr
 Copyright (c) University of British Columbia, 1999.
 All rights reserved.
```

```
 SUMUTE - MUTE above (or below) a user-defined polygonal curve with ",
   the distance along the curve specified by key header word

 sumute <stdin >stdout xmute= tmute= [optional parameters]

 Required parameters:
 xmute= array of position values as specified by
  the ‘key’ parameter
 tmute= array of corresponding time values (sec)
  in case of air wave muting, correspond to
  air blast duration
  ... or input via files:
 nmute= number of x,t values defining mute
 xfile= file containing position values as specified by
  the ‘key’ parameter
 tfile= file containing corresponding time values (sec)
  ... or via header:
 hmute= key header word specifying mute time

 Optional parameters:
 key=offset Key header word specifying trace offset
  =tracl  use trace number instead
 ntaper=0 number of points to taper before hard
mute (sine squared taper)
 mode=0    mute ABOVE the polygonal curve
=1 to zero BELOW the polygonal curve
=2 to mute below AND above a straight line. In this case
  xmute,tmute describe the total time length of
the muted zone as a function of xmute the slope
of the line is given by the velocity linvel=
  =3 to mute below AND above a constant velocity hyperbola
as in mode=2 xmute,tmute describe the total time
length of the mute zone as a function of xmute,
the velocity is given by the value of linvel=
  =4 to mute below AND above a user defined polygonal line
given by xmute, tmute pairs. The widths in time ",
of the muted zone are given by the twindow vector
 linvel=330    constant velocity for linear or hyperbolic mute
 tm0=0    time shift of linear or hyperbolic mute at
 \’key\’=0
 twindow= vector of mute zone widths in time, operative only in mode=4
  ... or input via file:
 twfile=
```

```
 Notes:
 The tmute interpolant is extrapolated to the left by the smallest time
 sample on the trace and to the right by the last value given in the
 tmute array.

 The files tfile and xfile are files of binary (C-style) floats.

 In the context of this program "above" means earlier time and
 "below" means later time (above and below as seen on a seismic section.

 The mode=2 option is intended for removing air waves. The mute is
 is over a narrow window above and below the line specified by the
 the line specified by the velocity "linvel". Here the values of
 tmute, xmute or tfile and xfile define the total time width of the mute.

 If data are spatial, such as the (z-x) output of a migration, then
 depth values are used in place of times in tmute and tfile. The value
 of the depth sampling interval is given by the d1 header field
 You must use the option key=tracl in sumute in this case.

 Caveat: if data are seismic time sections, then tr.dt must be set. If
 data are seismic depth sections, then tr.trid must be set to the value
 of TRID_DEPTH and the tr.d1 header field must be set.
 To find the value of TRID_DEPTH:
 type:
     sukeyword trid
and look for the entry for "Depth-Range (z-x) traces


 Credits:

SEP: Shuki Ronen
CWP: Jack K. Cohen, Dave Hale, John Stockwell
DELPHI: Alexander Koek added airwave mute.
      CWP: John Stockwell added modes 3 and 4
USBG: Florian Bleibinhaus added hmute + some range checks on mute times
 Trace header fields accessed: ns, dt, delrt, key=keyword, trid, d1
 Trace header fields modified: muts or mute
```

```
 SUPAD - Pad zero traces

  supad <stdin >stdout min= max= [optional parameters]

 Required parameters:
  min= trace key start
  max= trace key end

 Optional parameters:
  key1=ep panel key
  key2=tracf trace key
  key3=trid flag key
  val3=2 value assigned to padded traces
  d=1 trace key spacing

 Notes:
  In contrast to most SU codes, supad recognizes panels, or ensembles.
  If the input consists of several panels, each panel will be padded
  individually.
  key1 and key2 are the primary and secondary sort key of the data set.
  The sort order of key1 does not matter at all.
  The sort order of key2 must be monotonous - if key2 is descending,
supply a negative value for the spacing d.
  Traces with a key2-value outside the min/max range will be lost.
  Traces with a key2-value that is not a multiple of the spacing from
the min-value (the max-value, if the spacing is negative) will
not be lost. Instead, they will shift the series of key2-values.
  By default the dead trace flag will be raised for the padded traces.
  This should make it easy to remove the zero traces later on, if need be.

 Examples:
suplane | supad min=1 max=40 key1=offset key2=tracr | ...
... appends eight empty traces.

suplane | supad min=1 max=32 key1=offset key2=tracr d=0.5 | ...
... inserts a zero trace after each trace (even though the
header tracr is integer and cannot properly store the floats)

suplane | supad min=1 max=32 | ...
... produces an error because the panel and trace key are all 0.


 Credits:
```

Florian Bleibinhaus, U Salzburg, Austria

```
 SUPUTGTHR - split the stdout flow to gathers on the bases of given
  key parameter.

suputgthr <stdin   dir= [Optional parameters]

 Required parameters:
 dir= Name of directory where to put the gathers
 Optional parameters:
 key=ep header key word to watch
 suffix=".hsu" extension of the output files
 verbose=0 verbose = 1 echos information
 numlength=7 Length of numeric part of filename

 Notes:
 The name of the file is constructed from the key parameter. Traces
 are put into a temporary disk file, and renamed when key parameter
 changes in the input flow to "key.suffix". The result is that the
 directory "dir" contains separate files by "key" ensemble.  ",

 Header field modified:  ntr  to be the number of traces in a given
 ensemble.
 Related programs: sugetgthr, susplit


 Credits: Balazs Nemeth, Potash Corporation, Saskatoon Saskatchewan
 given to CWP in 2008
 Note:
The "valxxx" subroutines are in su/lib/valpkge.c.  In particular,
"valcmp" shares the annoying attribute of "strcmp" that
if (valcmp(type, val, valnew) {
...
}
will be performed when val and valnew are different.
```

```
 SUSORT - sort on any segy header keywords

 susort <stdin >stdout [[+-]key1 [+-]key2 ...]

 Susort supports any number of (secondary) keys with either
 ascending (+, the default) or descending (-) directions for
 each.  The default sort key is cdp.

 Note: Only the following types of input/output are supported
Disk input --> any output
Pipe input --> Disk output

 Caveat:  On some Linux systems Pipe input and or output often
fails
Disk input ---> Disk output is recommended

 Note: If the the CWP_TMPDIR environment variable is set use
its value for the path; else use tmpfile()

 Example:
 To sort traces by cdp gather and within each gather
 by offset with both sorts in ascending order:

  susort <INDATA >OUTDATA cdp offset

 Caveat: In the case of Pipe input a temporary file is made
to hold the ENTIRE data set.  This temporary is
either an actual disk file (usually in /tmp) or in some
implementations, a memory buffer.  It is left to the
user to be SENSIBLE about how big a file to pipe into
susort relative to the user's computer.


 Credits:
SEP: Einar Kjartansson , Stew Levin
CWP: Shuki Ronen,  Jack K. Cohen

 Caveats:
Since the algorithm depends on sign reversal of the key value
to obtain a descending sort, the most significant figure may
be lost for unsigned data types.  The old SEP support for tape
input was removed in version 1.16---version 1.15 is in the
Portability directory for those who may want to input SU data
```

stored on tape.

Trace header fields modified: tracl, tracr

```
 SUSORTY - make a small 2-D common shot off-end
    data set in which the data show geometry
    values to help visualize data sorting.

  susorty [optional parameters] > out_data_file

 Optional parameters:
nt=100  number of time samples
nshot=10  number of shots
dshot=10  shot interval (m)
noff=20  number of offsets
doff=20  offset increment (m)

 Notes:
 Creates a common shot su data file for sort visualization
       time samples           quantity
       ----------------      ----------
       first   25%           shot coord
       second  25%           rec coord
       third   25%           offset
       fourth  25%           cmp coord


 1. default is shot ordered (hsv2 cmap looks best to me)
 susorty | suximage legend=1 units=meters cmap=hsv2

 2. sort on cmp (note random order within a cmp)
 susorty | susort cdp > junk.su
 suximage < junk.su legend=1 units=meters cmap=hsv2

 3. sort to cmp and subsort on offset
 susorty | susort cdp offset > junk.su
 suximage < junk.su legend=1 units=meters cmap=hsv2


 Credits:
CWP: Chris Liner  10.09.01

 Trace header fields set: ns, dt, sx, gx, offset, cdp, tracl
```

```
 SUSPLIT - Split traces into different output files by keyword value

     susplit <stdin >stdout [options]

 Required Parameters:
none

 Optional Parameters:
key=cdp Key header word to split on (see segy.h)
stem=split_ Stem name for output files
middle=key middle of name of output files
suffix=.su Suffix for output files
numlength=7 Length of numeric part of filename
verbose=0 =1 to echo filenames, etc.
close=1 =1 to close files before opening new ones

 Notes:
 The most efficient way to use this program is to presort the input data
 into common keyword gathers, prior to using susplit. "

 Use "suputgthr" to put SU data into SU data directory format.

 Credits:
Geocon: Garry Perratt hacked together from various other codes
```

```
 SUWINDPOLY - WINDow data to extract traces on or within a respective
POLYgonal line or POLYgon with coordinates specified by header
keyword values

  suwindpoly <stdin [Required parameters] [Optional params] file=outfile

 Required parameters:
 x=x1,x2,... list of X coordinates for vertices
 y=y1,y2,... list of Y coordinates for vertices
 file=file1,file2,.. output filename(s)

 Optional parameters
 xkey=fldr X coordinate header key
 ykey=ep Y coordinate header key
 pass=0  polyline mode: pass traces near the polygonal line
=1 pass all traces interior to polygon
=2 pass all traces exterior to polygon

 Optional parameters used in polyline pass=0 mode only:
 The following need to be given if the unit increments in the X & Y
 directions are not equal.  For example, if fldr increments by 1 and
 ep increments by 4 to form 25 x 25 m bins specify dx=25.0 & dy=6.25.
 The output binning key will be converted to integers by the scaling
 with the smaller of the two values.

 dx=1.0 unit increment distance in X direction
 dy=1.0 unit increment distance in Y direction
 ilkey=tracl key for resulting inline index in polyline mode
 xlkey=tracr key for resulting xline index in polyline mode
 dw=1.0 distance in X-Y coordinate units of extracted line
to pass points to output.  Width of resulting line is
2*dw.  Ignored if polygon mode is specified.
 Notes:
 In polyline mode (pass=0), a single trace may be output multiple times
 if it meets the acceptance criteria (distance from line segment < dw)
 for multiple line segments. However, the headers will be distinct
 and will associate the output trace with a line segment. This
 behavior facilitates creation of 3D supergathers from polyline
 output. Use susort after running in polyline mode.

 x=& y=lists should be repeated for as many polygons as needed when
 pass=1 or pass=2.
```

```
In polygon mode, the polygon closes itself from the last vertex to
the first.

Example:
 suwindpoly <input.su x=10,20,50 y=0,30,60 dw=10 pass=0 file=out.su



Credits:  Reginald H. Beardsley rhb@acm.org
   originally: suxarb.c adapted from the SLT/SU package.
```

```
 SUWIND - window traces by key word

  suwind <stdin >stdout [options]

 Required Parameters:
  none

 Optional Parameters:
 verbose=0 =1 for verbose
 key=tracl Key header word to window on (see segy.h)
 min=LONG_MIN min value of key header word to pass
 max=LONG_MAX max value of key header word to pass

 abs=0 =1 to take absolute value of key header word
 j=1 Pass every j-th trace ...
 s=0 ... based at s  (if ((key - s)%j) == 0)
 skip=0 skip the initial N traces
 count=ULONG_MAX ... up to count traces
 reject=none Skip traces with specified key values
 accept=none Pass traces with specified key values(see notes)
processing, but do no window the data
 ordered=0 =1 if traces sorted in increasing keyword value
=-1  if traces are sorted in a decreasing order

 Options for vertical windowing (time gating):
 dt=tr.dt (from header) time sampling interval (sec) (seismic data)
   =tr.d1  (nonseismic)
 f1=tr.delrt (from header) first sample (seismic data)
   =tr.f1  (nonseismic)

 tmin=0.0 min time to pass
 tmax=(from header) max time to pass
 itmin=0 min time sample to pass
 itmax=(from header)   max time sample to pass
 nt=itmax-itmin+1 number of time samples to pass

 Notes:
 On large data sets, the count parameter should be set if
 possible.  Otherwise, every trace in the data set will be
 examined.  However, the count parameter overrides the accept
 parameter, so you can't specify count if you want true
 unconditional acceptance.
```

```
 The skip= option allows the user to skip over traces, which helps
 for selecting traces far from the beginning of the dataset.
 Caveat: skip only works with disk input.

 The ordered= option will speed up the process if the data are
 sorted in according to the key.

 The accept option is a bit strange--it does NOT mean accept ONLY
 the traces on the accept list!  It means accept these traces,
 even if they would otherwise be rejected (except as noted in the
 previous paragraph).  To implement accept-only, you can use the
 max=0 option (rejecting everything).  For example, to accept
 only the tracl values 4, 5 and 6:
... | suwind max=0 accept=4,5,6 | ...

 Another example is the case of suppressing nonseismic traces in
 a seismic data set. By the SEGY standard header field trace id,
 trid=1 designates traces as being seismic traces. Other traces,
 such as calibration traces may be designated by another value.

 Example:  trid=1 seismic and trid=0 is nonseismic. To reject
       nonseismic traces
       ... | suwind key=trid reject=0 | ...

 On most 32 bit machines, LONG_MIN, LONG_MAX and ULONG_MAX are
 about -2E9,+2E9 and 4E9, they are defined in limits.h.

 Selecting times beyond the maximum in the data induces
 zero padding (up to SU_NFLTS).

 The time gating here is to the nearest neighboring sample or
 time value. Gating to the exact temporal value requires
 resampling if the selected times fall between samples on the
 trace. Use suresamp to perform the time gating in this case.

 It doesn't really make sense to specify both itmin and tmin,
 but specifying itmin takes precedence over specifying tmin.
 Similarly, itmax takes precedence over tmax and tmax over nt.
 If dt in header is not set, then dt is mandatory


 Credits:
SEP: Einar Kjartansson
```

```
CWP: Shuki Ronen, Jack Cohen, Chris Liner
Warnemuende: Toralf Foerster
CENPET: Werner M. Heigl (modified to include well log data)

 Trace header fields accessed: ns, dt, delrt, keyword
 Trace header fields modified: ns, delrt, ntr
```

```
 SUPSCONTOUR - PostScript CONTOUR plot of a segy data set

 supscontour <stdin [optional parameters] | ...

 Optional parameters:

 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 See the pscontour selfdoc for the remaining parameters.

 On NeXT: supscontour < infile [optional parameters]  | open

 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2

 Credits:
```

```
CWP: Dave Hale and Zhiming Li (pscontour, etc.)
   Jack Cohen and John Stockwell (supscontour, etc.)
      Delphi: Alexander Koek, added support for irregularly spaced traces
      Aarhus University: Morten W. Pedersen copied everything from the xwigb
                         source and just replaced all occurencies of the word

 Notes:
When the number of traces isn't known, we need to count
the traces for pscontour.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

If your system does make a disk file, consider altering
the code to remove the file on interrupt.  This could be
done either by trapping the interrupt with "signal"
or by using the "tmpnam" routine followed by an immediate
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SUPSCUBECONTOUR - PostScript CUBE plot of a segy data set

supscubecontour <stdin [optional parameters] | ...

Optional parameters:

n2 is the number of traces per frame.  If not getparred then it
is the total number of traces in the data set.

n3 is the number of frames.  If not getparred then it
is the total number of frames in the data set measured by ntr/n2

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the pscubecontour selfdoc for the remaining parameters.

example:   supscubecontour < infile [optional parameters]  | gv -
```

```
 Credits:

CWP: Dave Hale and Zhiming Li (pscube)
     Jack K. Cohen  (suxmovie)
     John Stockwell (supscubecontour)

 Notes:
When n2 isn't getparred, we need to count the traces
for pscube. Although we compute ntr, we don't allocate a 2-d array
and content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SUPSCUBE - PostScript CUBE plot of a segy data set

supscube <stdin [optional parameters] | ...

Optional parameters:

n2 is the number of traces per frame.  If not getparred then it
is the total number of traces in the data set.

n3 is the number of frames.  If not getparred then it
is the total number of frames in the data set measured by ntr/n2

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the pscube selfdoc for the remaining parameters.

On NeXT:     supscube < infile [optional parameters]  | open
```

```
 Credits:

CWP: Dave Hale and Zhiming Li (pscube)
     Jack K. Cohen  (suxmovie)
     John Stockwell (supscube)

 Notes:
When n2 isn't getparred, we need to count the traces
for pscube. Although we compute ntr, we don't allocate a 2-d array
and content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSGRAPH - PostScript GRAPH plot of a segy data set

 supsgraph <stdin [optional parameters] | ...

 Optional parameters:
 style=seismic seismic is default here, =normal is alternate
(see psgraph selfdoc for style definitions)

 nplot is the number of traces (ntr is an acceptable alias for nplot)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to over-ride the header values if
 not.

 See the psgraph selfdoc for the remaining parameters.

 On NeXT:     supsgraph < infile [optional parameters]  | open

 Credits:
```

```
CWP: Dave Hale and Zhiming Li (pswigp, etc.)
   Jack Cohen and John Stockwell (supswigp, etc.)

 Notes:
When the number of traces isn't known, we need to count
the traces for pswigp.  You can make this value "known"
either by getparring nplot or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSIMAGE - PostScript IMAGE plot of a segy data set

 supsimage <stdin [optional parameters] | ...

 Optional parameters:

 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 See the psimage selfdoc for the remaining parameters.

 On NeXT: supsimage < infile [optional parameters]  | open

 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2

 Credits:
```

CWP: Dave Hale and Zhiming Li (psimage, etc.)
   Jack Cohen and John Stockwell (supsimage, etc.)

 Notes:
When the number of traces isn't known, we need to count
the traces for psimage. You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header. A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.

```
SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace
    of a SEGY (SU) data set

  supsmax <stdin >postscript file [optional parameters]

Optional parameters:
mode=max max value
 =min min value
 =abs absolute max value

n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the sumax selfdoc for additional parameter.
See the psgraph selfdoc for the remaining parameters.
```

```
 Credits:

CWP: John Stockwell, based on Jack Cohen's SU JACKet

 Notes:
When the number of traces isn't known, we need to count
the traces for psgraph.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.
```

```
SUPSMOVIE - PostScript MOVIE plot of a segy data set

supsmovie <stdin [optional parameters] | ...

Optional parameters:

n2 is the number of traces per frame.  If not getparred then it
is the total number of traces in the data set.

n3 is the number of frames.  If not getparred then it
is the total number of frames in the data set measured by ntr/n2

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the psmovie selfdoc for the remaining parameters.

On NeXT:     supsmovie < infile [optional parameters]  | open
```

```
 Credits:

CWP: Dave Hale and Zhiming Li (psmovie)
     Jack K. Cohen  (suxmovie)
     John Stockwell (supsmovie)

 Notes:
When n2 isn't getparred, we need to count the traces
for psmovie.  In this case:
we are using tmpfile because on many machines it is
implemented as a memory area instead of a disk file.
Although we compute ntr, we don't allocate a 2-d array
and content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSWIGB - PostScript Bit-mapped WIGgle plot of a segy data set

 supswigb <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
 vsp (same as normal with axis 2 reversed)
 Note: vsp requires use of a keyword

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type   sukeyword -o   to see the complete list of SU keywords.

 This program is really just a wrapper for the plotting program: pswigb
 See the pswigb selfdoc for the remaining parameters.
```

```
 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2, keyword (if set)

 Credits:

CWP: Dave Hale and Zhiming Li (pswigb, etc.)
   Jack Cohen and John Stockwell (supswigb, etc.)
      Delphi: Alexander Koek, added support for irregularly spaced traces

Modified by Brian Zook, Southwest Research Institute, to honor
 scale factors, added vsp style

 Notes:
When the number of traces isn't known, we need to count
the traces for pswigb.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSWIGP - PostScript Polygon-filled WIGgle plot of a segy data set

 supswigp <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
 vsp (same as normal with axis 2 reversed)
 Note: vsp requires use of a keyword

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type   sukeyword -o   to see the complete list of SU keywords.

 This program is really just a wrapper for the plotting program: pswigp
 See the pswigp selfdoc for the remaining parameters.
```

```
 On NeXT: supswigp < infile [optional parameters]  | open

 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2, key specified by key

 Credits:

CWP: Dave Hale and Zhiming Li (pswigp, etc.)
   Jack Cohen and John Stockwell (supswigp, etc.)
Delphi: Alexander Koek, added support for irregularly spaced traces

Modified by Brian Zook, Southwest Research Institute, to honor
 scale factors, added vsp style

 Notes:
When the number of traces isn't known, we need to count
the traces for pswigp.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

If your system does make a disk file, consider altering
the code to remove the file on interrupt.  This could be
done either by trapping the interrupt with "signal"
or by using the "tmpnam" routine followed by an immediate
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXCONTOUR - X CONTOUR plot of Seismic UNIX tracefile via vector plot call

 suxwigb <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type   sukeyword -o   to see the complete list of SU keywords.

 This program is really just a wrapper for the plotting program: xcontour
 See the xcontour selfdoc for the remaining parameters.



 Credits:
```

```
CWP: Dave Hale and Zhiming Li (xwigb, etc.)
   Jack Cohen and John Stockwell (suxwigb, etc.)
      Delphi: Alexander Koek, added support for irregularly spaced traces
      Aarhus University: Morten W. Pedersen copied everything from the xwigb
                         source and just replaced all occurencies of the word
                         xwigb with xcountour ;-)

 Notes:
When the number of traces isn't known, we need to count
the traces for xcontour.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXGRAPH - X-windows GRAPH plot of a segy data set

 suxgraph <stdin [optional parameters] | ...

 Optional parameters:
 (see xgraph selfdoc for optional parametes)

 nplot= number of traces (ntr is an acceptable alias for nplot)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to over-ride the header values if
 not.

 See the xgraph selfdoc for the remaining parameters.

 On NeXT:     suxgraph < infile [optional parameters]  | open

 Credits:

CWP: Dave Hale and Zhiming Li (pswigp, etc.)
```

Jack Cohen and John Stockwell (supswigp, etc.)

Notes:
When the number of traces isn't known, we need to count
the traces for pswigp. You can make this value "known"
either by getparring nplot or by having the ntr field set
in the trace header. A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.

```
SUXIMAGE - X-windows IMAGE plot of a segy data set

suximage infile= [optional parameters] | ...  (direct I/O)
 or
suximage <stdin [optional parameters] | ... (sequential I/O)

Optional parameters:

infile=NULL SU data to be ploted, default stdin with sequential access
            if 'infile' provided, su data read by (fast) direct access

     with ftr,dtr and n2 suximage will pass a subset of data
     to the plotting program-ximage:
ftr=1       First trace to be plotted
dtr=1       Trace increment to be plotted
n2=tr.ntr   (Max) number of traces to be plotted (ntr is an alias for n2)
     Priority: first try to read from parameter line;
       if not provided, check trace header tr.ntr;
       if still not provided, figure it out using ftello

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set or was set to 0)

key= key for annotating d2 (slow dimension)
 If annotation is not at proper increment, try
 setting d2; only first trace's key value is read

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0             =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()
```

```
 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 See the ximage selfdoc for the remaining parameters.


 Credits:

CWP: Dave Hale and Zhiming Li (ximage, etc.)
   Jack Cohen and John Stockwell (suximage, etc.)
MTU: David Forel, June 2004, added key for annotating d2
      ConocoPhillips: Zhaobo Meng, Dec 2004, added direct I/O

 Notes:

      When provide ftr and dtr and infile, suximage can be used to plot
      multi-dimensional volumes efficiently.  For example, for a Offset-CDP
      dataset with 32 offsets, the command line
      suximage infile=volume3d.su ftr=1 dtr=32 ... &
      will display the zero-offset common offset data with ranrom access.
      It is highly recommend to use infile= to view large datasets, since
      using stdin only allows sequential access, which is very slow for
      large datasets.

When the number of traces isn't known, we need to count
the traces for ximage.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXMAX - X-windows graph of the MAX, min, or absolute max value on
each trace of a SEGY (SU) data set

   suxmax <stdin [optional parameters]

 Optional parameters:
 mode=max max value
  =min min value
  =abs absolute max value

 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to over-ride the header values if
 not.

 See the sumax selfdoc for additional parameter.
 See the xgraph selfdoc for the remaining parameters.
```

```
 Credits:

CWP: John Stockwell, based on Jack Cohen's SU JACKet

 Notes:
When the number of traces isn't known, we need to count
the traces for xgraph.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.
```

```
SUXMOVIE - X MOVIE plot of a 2D or 3D segy data set

suxmovie <stdin [optional parameters]

Optional parameters:

n1=tr.ns                number of samples per trace
ntr=tr.ntr            number of traces in the data set
n2=tr.shortpad or tr.ntr number of traces in inline direction
n3=ntr/n2            number of traces in crossline direction

d1=tr.d1 or tr.dt/10^6    sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2     sampling interval in the slow dimension
  =1.0  (if not set)

d3=1.0     sampling interval in the slowest dimension

f1=tr.f1 or 0.0       first sample in the z dimension
f2=tr.f2 or 1.0           first sample in the x dimension
f3=1.0

mode=0          0= x,z slice movie through y dimension (in line)
                1= y,z slice movie through x dimension (cross line)
                2= x,y slice movie through z dimension (time slice)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Notes:
For seismic data, the "fast dimension" is either time or
depth and the "slow dimension" is usually trace number.
The 3D data set is expected to have n3 sets of n2 traces representing
the horizontal coverage of n2*d2 in x  and n3*d3 in y direction.

The data is read to memory with and piped to xmovie with the
respective sampling parameters.
```

See the xmovie selfdoc for the remaining parameters and X functions.

```
 SUXPICKER - X-windows  WIGgle plot PICKER of a segy data set

 suxpicker <stdin [optional parameters] | ...

 Optional parameters:

 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information


 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type  sukeyword -o to see the complete list of SU keywords.
```

```
 See the xpicker selfdoc for the remaining parameters.


 Credits:

CWP: Dave Hale and Zhiming Li (xpicker, etc.)
   Jack Cohen and John Stockwell (suxpicker, etc.)

 Notes:
When the number of traces isn't known, we need to count
the traces for xpicker.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

If your system does make a disk file, consider altering
the code to remove the file on interrupt.  This could be
done either by trapping the interrupt with "signal"
or by using the "tmpnam" routine followed by an immediate
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXWIGB - X-windows Bit-mapped WIGgle plot of a segy data set
 This is a modified suxwigb that uses the depth or coordinate scaling
 when such values are used as keys.

 suxwigb <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0 (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
 vsp (same as normal with axis 2 reversed)
 Note: vsp requires use of a keyword
 verbose=0              =1 to print some useful information


 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type  sukeyword -o to see the complete list of SU keywords.
```

```
 This program is really just a wrapper for the plotting program: xwigb
 See the xwigb selfdoc for the remaining parameters.


 Credits:

CWP: Dave Hale and Zhiming Li (xwigb, etc.)
   Jack Cohen and John Stockwell (suxwigb, etc.)
Delphi: Alexander Koek, added support for irregularly spaced traces

Modified by Brian Zook, Southwest Research Institute, to honor
 scale factors, added vsp style

 Notes:
When the number of traces isn't known, we need to count
the traces for xwigb.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
PSBBOX - change BoundingBOX of existing PostScript file

psbbox < PostScriptfile [optional parameters] > PostScriptfile

Optional Parameters:
llx= new llx
lly= new lly
urx= new urx
ury= new ury
verbose=1 =1 for info printed on stderr (0 for no info)
```

```
PSCONTOUR - PostScript CONTOURing of a two-dimensional function f(x1,x2)

pscontour n1= [optional parameters] <binaryfile >postscriptfile

Required Parameters:
n1                      number of samples in 1st (fast) dimension

Optional Parameters:
d1=1.0                  sampling interval in 1st dimension
f1=d1                   first sample in 1st dimension
x1=f1,f1+d1,...         array of monotonic sampled values in 1st dimension
n2=all                  number of samples in 2nd (slow) dimension
d2=1.0                  sampling interval in 2nd dimension
f2=d2                   first sample in 2nd dimension
x2=f2,f2+d2,...         array of monotonic sampled values in 2nd dimension
nc=5                    number of contour values
dc=(zmax-zmin)/nc       contour interval
fc=min+dc               first contour
c=fc,fc+dc,...          array of contour values
cwidth=1.0,...          array of contour line widths
cgray=0.0,...           array of contour grays (0.0=black to 1.0=white)
ccolor=none,...         array of contour colors; none means use cgray
cdash=0.0,...           array of dash spacings (0.0 for solid)
labelcf=1               first labeled contour (1,2,3,...)
labelcper=1             label every labelcper-th contour
nlabelc=nc              number of labeled contours (0 no contour label)
nplaces=6               number of decimal places in contour label
xbox=1.5                offset in inches of left side of axes box
ybox=1.5                offset in inches of bottom side of axes box
wbox=6.0                width in inches of axes box
hbox=8.0                height in inches of axes box
x1beg=x1min             value at which axis 1 begins
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
x2beg=x2min             value at which axis 2 begins
x2end=x2max             value at which axis 2 ends
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
```

```
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelfont=Helvetica     font name for axes labels
labelsize=18            font size for axes labels
title=                  title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24            font size for title
labelcfont=Helvetica-Bold font name for contour labels
labelcsize=6            font size of contour labels
labelccolor=black       color of contour labels
titlecolor=black        color of title
axescolor=black         color of axes
gridcolor=black         color of grid
axeswidth=1             width (in points) of axes
ticwidth=axeswidth      width (in points) of tic marks
gridwidth=axeswidth     width (in points) of grid lines
style=seismic           normal (axis 1 horizontal, axis 2 vertical) or
                        seismic (axis 1 vertical, axis 2 horizontal)

Note.
The line width of unlabeled contours is designed as a quarter of that
of labeled contours.

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic
type:   sudoc pscontour    for more information


Notes:
```

```
For nice even-numbered contours, use the parameters  fc and dc

Example: if the range of the z values of a data set range between
approximately -1000 and +1000, then use fc=-1000 nc=10 and dc=100
to get contours spaced by even 100's.



AUTHOR:  Dave Hale, Colorado School of Mines, 05/29/90
MODIFIED:  Craig Artley, Colorado School of Mines, 08/30/91
           BoundingBox moved to top of PostScript output
MODIFIED:  Zhenyue Liu, Colorado School of Mines, 08/26/93
     Values are labeled on contours
MODIFIED:  Craig Artley, Colorado School of Mines, 12/16/93
           Added color options (Courtesy of Dave Hale, Advance Geophysical).
Modified: Morten Wendell Pedersen, Aarhus University, 23/3-97
          Added ticwidth,axeswidth, gridwidth parameters
```

```
 PSCCONTOUR - PostScript Contour plot of a data CUBE

 pscubecontour n1= n2= n3= [optional parameters] <binaryfile >postscriptfile
    or
 pscubecontour n1= n2= n3= front= side= top= [optional parameters] >postscriptfile

 Data formats supported:
1. Entire cube read from stdin (n1*n2*n3 floats) [default format]
2. Faces read from stdin (n1*n2 floats for front, followed by n1*n3
   floats for side, and n2*n3 floats for top) [specify faces=1]
3. Faces read from separate data files [specify filenames]

 Required Parameters:
 n1                    number of samples in 1st (fastest) dimension
 n2                    number of samples in 2nd dimension
 n3                    number of samples in 3rd (slowest) dimension

 Optional Parameters:
 front                 name of file containing front panel
 side                  name of file containing side panel
 top                   name of file containing top panel
 faces=0               =1 to read faces from stdin (data format 2)
 d1=1.0                sampling interval in 1st dimension
 f1=0.0                first sample in 1st dimension
 d2=1.0                sampling interval in 2nd dimension
 f2=0.0                first sample in 2nd dimension
 d3=1.0                sampling interval in 3rd dimension
 f3=0.0                first sample in 3rd dimension
 d1s=1.0               factor by which to scale d1 before imaging
 d2s=1.0               factor by which to scale d2 before imaging
 d3s=1.0               factor by which to scale d3 before imaging
 nc=5                  number of contour values
 dc=(zmax-zmin)/nc     contour interval
 fc=min+dc             first contour
 c=fc,fc+dc,...        array of contour values
 cwidth=1.0,...        array of contour line widths
 cgray=0.0,...         array of contour grays (0.0=black to 1.0=white)
 ccolor=none,...       array of contour colors; none means use cgray
 cdash=0.0,...         array of dash spacings (0.0 for solid)
 labelcf=1             first labeled contour (1,2,3,...)
 labelcper=1           label every labelcper-th contour
 nlabelc=nc            number of labeled contours (0 no contour label)
 nplaces=6             number of decimal places in contour label
```

```
xbox=1.5                offset in inches of left side of axes box
ybox=1.5                offset in inches of bottom side of axes box
size1=4.0               size in inches of 1st axes (vertical)
size2=4.0               size in inches of 2nd axes (horizontal)
size3=3.0               size in inches of 3rd axes (projected)
angle=45                projection angle of cube in degrees (0<angle<90)
                        (angle between 2nd axis and projected 3rd axis)
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
x2beg=x2min             value at which axis 2 begins
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
x3end=x3max             value at which axis 3 ends
d3num=0.0               numbered tic interval on axis 3 (0.0 for automatic)
f3num=x3min             first numbered tic on axis 3 (used if d3num not 0.0)
n3tic=1                 number of tics per numbered tic on axis 3
grid3=none              grid lines on axis 3 - none, dot, dash, or solid
label3=                 label on axis 3
labelfont=Helvetica     font name for axes labels
labelsize=18            font size for axes labels
title=                  title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24            font size for title
titlecolor=black        color of title
labelcfont=Helvetica-Bold font name for contour labels
labelcsize=6            font size of contour labels
labelccolor=black       color of contour labels
axescolor=black         color of axes
gridcolor=black         color of grid

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Note: The values of x1beg=x1min, x2end=x2max and x3beg=x3min cannot
be changed.
```

```
Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic



(Original codes pscontour and pscube)

AUTHOR:  Craig Artley, Colorado School of Mines, 03/12/93
NOTE:  Original written by Zhiming Li & Dave Hale, CSM, 07/01/90
 Completely rewritten, the code now bears more similarity to
 psimage than the previous pscube.  Faces of cube now rendered
 as three separate images, rather than as a single image.  The
 output no longer suffers from stretching artifacts, and the
 code is simpler.  -Craig
MODIFIED:  Craig Artley, Colorado School of Mines, 12/17/93
   Added color options.

PSCCONTOUR: mashed together from pscube and pscontour
to generate 3d contour plots by Claudia Vanelle, Institute of Geophysics,
University of Hamburg, Germany somewhen in 2000

PSCUBE was "merged" with PSCONTOUR to create PSCUBECONTOUR
by Claudia Vanelle, Applied Geophysics Group Hamburg
somewhen in 2000
```

```
 PSCUBE - PostScript image plot with Legend of a data CUBE

 pscube n1= n2= n3= [optional parameters] <binaryfile >postscriptfile
    or
 pscube n1= n2= n3= front= side= top= [optional parameters] >postscriptfile

 Data formats supported:
1. Entire cube read from stdin (n1*n2*n3 floats) [default format]
2. Faces read from stdin (n1*n2 floats for front, followed by n1*n3
   floats for side, and n2*n3 floats for top) [specify faces=1]
3. Faces read from separate data files [specify filenames]

 Required Parameters:
 n1                     number of samples in 1st (fastest) dimension
 n2                     number of samples in 2nd dimension
 n3                     number of samples in 3rd (slowest) dimension

 Optional Parameters:
 front                  name of file containing front panel
 side                   name of file containing side panel
 top                    name of file containing top panel
 faces=0                =1 to read faces from stdin (data format 2)
 d1=1.0                 sampling interval in 1st dimension
 f1=0.0                 first sample in 1st dimension
 d2=1.0                 sampling interval in 2nd dimension
 f2=0.0                 first sample in 2nd dimension
 d3=1.0                 sampling interval in 3rd dimension
 f3=0.0                 first sample in 3rd dimension
 perc=100.0             percentile used to determine clip
 clip=(perc percentile) clip used to determine bclip and wclip
 bperc=perc             percentile for determining black clip value
 wperc=100.0-perc       percentile for determining white clip value
 bclip=clip             data values outside of [bclip,wclip] are clipped
 wclip=-clip            data values outside of [bclip,wclip] are clipped
 brgb=0.0,0.0,0.0       red, green, blue values corresponding to black
 wrgb=1.0,1.0,1.0       red, green, blue values corresponding to white
 bhls=0.0,0.0,0.0       hue, lightness, saturation corresponding to black
 whls=0.0,1.0,0.0       hue, lightness, saturation corresponding to white
 bps=12                 bits per sample for color plots, either 12 or 24
 d1s=1.0                factor by which to scale d1 before imaging
 d2s=1.0                factor by which to scale d2 before imaging
 d3s=1.0                factor by which to scale d3 before imaging
 verbose=1              =1 for info printed on stderr (0 for no info)
```

```
xbox=1.5                offset in inches of left side of axes box
ybox=1.5                offset in inches of bottom side of axes box
size1=4.0               size in inches of 1st axes (vertical)
size2=4.0               size in inches of 2nd axes (horizontal)
size3=3.0               size in inches of 3rd axes (projected)
angle=45                projection angle of cube in degrees (0<angle<90)
                        (angle between 2nd axis and projected 3rd axis)
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
x2beg=x2min             value at which axis 2 begins
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
x3end=x3max             value at which axis 3 ends
d3num=0.0               numbered tic interval on axis 3 (0.0 for automatic)
f3num=x3min             first numbered tic on axis 3 (used if d3num not 0.0)
n3tic=1                 number of tics per numbered tic on axis 3
grid3=none              grid lines on axis 3 - none, dot, dash, or solid
label3=                 label on axis 3
labelfont=Helvetica     font name for axes labels
labelsize=18            font size for axes labels
title=                  title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24            font size for title
titlecolor=black        color of title
axescolor=black         color of axes
gridcolor=black         color of grid
legend=0                =1 display the color scale
                        if ==1, resize xbox,ybox,width,height
lstyle=vertleft        Vertical, axis label on left side
                        =vertright (Vertical, axis label on right side)
                        =horibottom (Horizontal, axis label on bottom)
units=                  unit label for legend
legendfont=times_roman10    font name for title
following are defaults for lstyle=0. They are changed for other lstyles
lwidth=1.2              colorscale (legend) width in inches
lheight=height/3        colorscale (legend) height in inches
```

```
lx=1.0                  colorscale (legend) x-position in inches
ly=(height-lheight)/2+xybox    colorscale (legend) y-position in pixels
lbeg= lmin or wclip-5*perc    value at which legend axis begins
lend= lmax or bclip+5*perc    value at which legend axis ends
ldnum=0.0      numbered tic interval on legend axis (0.0 for automatic)
lfnum=lmin     first numbered tic on legend axis (used if d1num not 0.0)
lntic=1        number of tics per numbered tic on legend axis
lgrid=none     grid lines on legend axis - none, dot, dash, or solid

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic
Note: The values of x1beg=x1min, x2end=x2max and x3beg=x3min cannot
be changed.
```

```
PSEPSI - add an EPSI formatted preview bitmap to an EPS file

psepsi <epsfile >epsifile

Note:
This application requires
(1) that gs (the Ghostscript interpreter) exist, and
(2) that the input EPS file contain a BoundingBox and EndComments.
Ghostscript is used to build the preview bitmap, which is then
merged with the input EPS file to make the output EPSI file.
```

```
PSGRAPH - PostScript GRAPHer
Graphs n[i] pairs of (x,y) coordinates, for i = 1 to nplot.

psgraph n= [optional parameters] <binaryfile >postscriptfile

Required Parameters:
n                       array containing number of points per plot

Data formats supported:
1.a. x1,y1,x2,y2,...,xn,yn
  b. x1,x2,...,xn,y1,y2,...,yn (must set pairs=0)
2.   y1,y2,...,yn (must give non-zero d1[]=)
3.   x1,x2,...,xn (must give non-zero d2[]=)
4.   nil (must give non-zero d1[]= and non-zero d2[]=)
  The formats may be repeated and mixed in any order, but if
  formats 2-4 are used, the d1 and d2 arrays must be specified including
  d1[]=0.0 d2[]=0.0 entries for any internal occurences of format 1.
  Similarly, the pairs array must contain place-keeping entries for
  plots of formats 2-4 if they are mixed with both formats 1.a and 1.b.
  Also, if formats 2-4 are used with non-zero f1[] or f2[] entries, then
  the corresponding array(s) must be fully specified including f1[]=0.0
  and/or f2[]=0.0 entries for any internal occurences of format 1 or
  formats 2-4 where the zero entries are desired.

  Available colors are all the common ones and many more. The complete
  list of 68 colors is in the file $CWPROOT/src/psplot/basic.c.

Optional Parameters:
nplot=number of n's     number of plots
d1=0.0,...              x sampling intervals (0.0 if x coordinates input)
f1=0.0,...              first x values (not used if x coordinates input)
d2=0.0,...              y sampling intervals (0.0 if y coordinates input)
f2=0.0,...              first y values (not used if y coordinates input)
pairs=1,...             =1 for data pairs in format 1.a, =0 for format 1.b
linewidth=1.0,...       line widths (in points) (0.0 for no lines)
linegray=0.0,...        line gray levels (black=0.0 to white=1.0)
linecolor=none,...      line colors; none means use linegray
                        Typical use: linecolor=red,yellow,blue,...
lineon=1.0,...          length of line segments for dashed lines (in points)
lineoff=0.0,...         spacing between dashes (0.0 for solid line)
mark=0,1,2,3,...        indices of marks used to represent plotted points
marksize=0.0,0.0,...    size of marks (0.0 for no marks)
xbox=1.5                offset in inches of left side of axes box
```

```
ybox=1.5                offset in inches of bottom side of axes box
wbox=6.0                width in inches of axes box
hbox=8.0                height in inches of axes box
x1beg=x1min             value at which axis 1 begins
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
x2beg=x2min             value at which axis 2 begins
x2end=x2max             value at which axis 2 ends
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelfont=Helvetica     font name for axes labels
labelsize=18            font size for axes labels
title=                  title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24            font size for title
titlecolor=black        color of title
axescolor=black         color of axes
gridcolor=black         color of grid
axeswidth=1             width (in points) of axes
ticwidth=axeswidth      width (in points) of tic marks
gridwidth=axeswidth     width (in points) of grid lines
style=normal            normal (axis 1 horizontal, axis 2 vertical) or
                        seismic (axis 1 vertical, axis 2 horizontal)
reverse=0               =1 to reverse sequence of plotting curves      ",
Note: n1 and n2 are acceptable aliases for n and nplot, respectively.

mark index:
1. asterisk
2. x-cross
3. open triangle
4. open square
5. open circle
6. solid triangle
7. solid square
8. solid circle
```

```
All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Example:
psgraph n=50,100,20 d1=2.5,1,0.33 <datafile >psfile
 plots three curves with equally spaced x values in one plot frame
 x1-coordinates are x1(i) = f1+i*d1 for i = 1 to n (f1=0 by default)
 number of x2's and then x2-coordinates for each curve are read
 sequentially from datafile.

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic
```

```
PSIMAGE - PostScript IMAGE plot of a uniformly-sampled function f(x1,x2)
          with the option to display a second attribute

psimage n1= [optional parameters] <binaryfile >postscriptfile

Required Parameters:
n1  number of samples in 1st (fast) dimension

Optional Parameters:
d1=1.0  sampling interval in 1st dimension
f1=0.0  first sample in 1st dimension
n2=all  number of samples in 2nd (slow) dimension
d2=1.0  sampling interval in 2nd dimension
f2=0.0  first sample in 2nd dimension
perc=100.0  percentile used to determine clip
clip=(perc percentile) clip used to determine bclip and wclip
bperc=perc  percentile for determining black clip value
wperc=100.0-perc  percentile for determining white clip value
bclip=clip  data values outside of [bclip,wclip] are clipped
wclip=-clip  data values outside of [bclip,wclip] are clipped
                      bclip and wclip will be set to be inside
                      [lbeg,lend] if lbeg and/or lend are supplied
threecolor=1  supply 3 color values instead of only two,
                      using not only black and white, but f.e. red,
                      green and blue
brgb=0.0,0.0,0.0  red, green, blue values corresponding to black
grgb=1.0,1.0,1.0  red, green, blue values corresponding to grey
wrgb=1.0,1.0,1.0  red, green, blue values corresponding to white
bhls=0.0,0.0,0.0  hue, lightness, saturation corresponding to black
ghls=0.0,1.0,0.0  hue, lightness, saturation corresponding to grey
whls=0.0,1.0,0.0  hue, lightness, saturation corresponding to white
bps=12  bits per sample for color plots, either 12 or 24
d1s=1.0  factor by which to scale d1 before imaging
d2s=1.0  factor by which to scale d2 before imaging
verbose=1  =1 for info printed on stderr (0 for no info)
xbox=1.5  offset in inches of left side of axes box
ybox=1.5  offset in inches of bottom side of axes box
width=6.0  width in inches of axes box
height=8.0  height in inches of axes box
x1beg=x1min  value at which axis 1 begins
x1end=x1max  value at which axis 1 ends
d1num=0.0  numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min  first numbered tic on axis 1 (used if d1num not 0.0)
```

```
n1tic=1  number of tics per numbered tic on axis 1
grid1=none  grid lines on axis 1 - none, dot, dash, or solid
label1=  label on axis 1
x2beg=x2min  value at which axis 2 begins
x2end=x2max  value at which axis 2 ends
d2num=0.0  numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min  first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1  number of tics per numbered tic on axis 2
grid2=none  grid lines on axis 2 - none, dot, dash, or solid
label2=  label on axis 2
labelfont=Helvetica  font name for axes labels
labelsize=18  font size for axes labels
title=  title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24   font size for title
titlecolor=black  color of title
axescolor=black  color of axes
gridcolor=black  color of grid
axeswidth=1            width (in points) of axes
ticwidth=axeswidth     width (in points) of tic marks
gridwidth=axeswidth    width (in points) of grid lines
style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
seismic (axis 1 vertical, axis 2 horizontal)
legend=0          =1 display the color scale
lnice=0                =1 nice legend arrangement
                       (overrides ybox,lx,width,height parameters)
lstyle=vertleft  Vertical, axis label on left side
=vertright (Vertical, axis label on right side)
=horibottom (Horizontal, axis label on bottom)
units=  unit label for legend
legendfont=times_roman10    font name for title
following are defaults for lstyle=0. They are changed for other lstyles
lwidth=1.2  colorscale (legend) width in inches
lheight=height/3       colorscale (legend) height in inches
lx=1.0  colorscale (legend) x-position in inches
ly=(height-lheight)/2+xybox    colorscale (legend) y-position in pixels
lbeg= lmin or wclip-5*perc    value at which legend axis begins
lend= lmax or bclip+5*perc    value at which legend axis ends
ldnum=0.0  numbered tic interval on legend axis (0.0 for automatic)
lfnum=lmin  first numbered tic on legend axis (used if d1num not 0.0)
lntic=1  number of tics per numbered tic on legend axis
lgrid=none  grid lines on legend axis - none, dot, dash, or solid
```

```
curve=curve1,curve2,...  file(s) containing points to draw curve(s)
npair=n1,n2,n2,...            number(s) of pairs in each file
curvecolor=black,..  color of curve(s)
curvewidth=axeswidth  width (in points) of curve(s)
curvedash=0            solid curve(s), dash indices 1,...,11 produce
                       curve(s) with various dash styles

infile=none            filename of second attribute n1xn2 array
                       values must be from range 0.0 - 1.0
                       (plain unformatted C-style file)
bckgr=0.5              background gray value

NOTES:
The curve file is an ascii file with the points specified as x1 x2
pairs, one pair to a line.  A "vector" of curve files and curve
colors may be specified as curvefile=file1,file2,etc.
and curvecolor=color1,color2,etc, and the number of pairs of values
in each file as npair=npair1,npair2,... .

You may eliminate the blocky appearance of psimages by adjusting the
d1s= and d2s= parameters, so that psimages appear similar to ximages.

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Some example colormap settings:
red white blue: wrgb=1.0,0,0 grgb=1.0,1.0,1.0 brgb=0,0,1.0
white red blue: wrgb=1.0,1.0,1.0 grgb=1.0,0.0,0.0 brgb=0,0,1.0
blue red white: wrgb=0.0,0.0,1.0 grgb=1.0,0.0,0.0 brgb=1.0,1.0,1.0
red green blue: wrgb=1.0,0,0 grgb=0,1.0,0 brgb=0,0,1.0
orange light-blue green: wrgb=1.0,.5,0 grgb=0,.7,1.0 brgb=0,1.0,0
red light-blue dark blue: wrgb=0.0,0,1.0 grgb=0,1.0,1.0 brgb=0,0,1.0

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
```

```
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic



AUTHOR:  Dave Hale, Colorado School of Mines, 05/29/90
MODIFIED:  Craig Artley, Colorado School of Mines, 08/30/91
   BoundingBox moved to top of PostScript output
MODIFIED:  Craig Artley, Colorado School of Mines, 12/16/93
   Added color options (Courtesy of Dave Hale, Advance Geophysical).
Modified: Morten Wendell Pedersen, Aarhus University, 23/3-97
          Added ticwidth,axeswidth, gridwidth parameters
MODIFIED: Torsten Schoenfelder, Koeln, Germany 006/07/97
         colorbar (legend) (as in ximage (by Berend Scheffers, Delft))
MODIFIED: Brian K. Macy, Phillips Petroleum, 01/14/99
   Added curve plotting option
MODIFIED: Torsten Schoenfelder, Koeln, Germany 02/10/99
         color scale with interpolation of three colors
MODIFIED: Ekkehart Tessmer, University of Hamburg, Germany, 08/22/2007
         Added dashing option to curve plotting
```

```
 PSLABEL - output PostScript file consisting of a single TEXT string
          on a specified background. (Use with psmerge to label plots.)

 pslabel t= [t=] [optional parameters] > epsfile

Required Parameters (can have multiple specifications to mix fonts):
  t=                 text string to write to output

Optional Parameters:
  f=Times-Bold       font for text string
                      (multiple specifications for each t)
  size=30            size of characters in points (72 points/inch)
  tcolor=black       color of text string
  bcolor=white       color of background box
  nsub=0             number of characters to subtract when
                     computing size of background box (not all
                     characters are the same size so the
                     background box may be too big at times.)

 Example:
 pslabel t="(a) " f=Times-Bold t="h" f=Symbol t="=0.04" nsub=3 > epsfile

 This example yields the PostScript equivalent of the string
  (written here in LaTeX notation) $ (a)\\; \\eta=0.04 $

 Notes:
 This program produces a (color if desired) PostScript text string that
 can be positioned and pasted on a PostScript plot using   psmerge
     see selfdoc of   psmerge for further information.

 Possible fonts:   Helvetica, Helvetica-Oblique, Helvetica-Bold,
  Helvetica-BoldOblique,Times-Roman,Times-Italic,Times-Bold,
  Times-BoldItalic,Courier,Courier-Bold,Courier-Oblique,
  Courier-BoldOblique,Symbol

 Possible colors:  greenyellow,yellow,goldenrod,dandelion,apricot,
  peach,melon,yelloworange,orange,burntorange,bittersweet,redorange,
  mahogany,maroon,brickred,red,orangered,rubinered,wildstrawberry,
  salmon,carnationpink,magenta,violetred,rhodamine,mulberry,redviolet,
  fuchsia,lavender,thistle,orchid,darkorchid,purple,plum,violet,royalpurple,
  blueviolet,periwinkle,cadetblue,cornflowerblue,midnightblue,naveblue,
  royalblue,blue,cerulean,cyan,processblue,skyblue,turquoise,tealblue,
  aquamarine,bluegreen,emerald,junglegreen,seagreen,green,forestgreen,
```

```
 pinegreen,limegreen,yellowgreen,springgreen,olivegreen,rawsienna,sepia,
 brown,tan,white,black,gray

All color specifications may also be made in X Window style Hex format
example:   tcolor=#255

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic



AUTHOR:  John E. Anderson, Visiting Scientist from Mobil, 1994
```

```
PSMANAGER - printer MANAGER for HP 4MV and HP 5Si Mx Laserjet
               PostScript printing

  psmanager < stdin  [optional parameters] > stdout

Required Parameters:
 none
Optional Parameters:
papersize=0 paper size  (US Letter default)
 =1       US Legal
 =2  A4
 =3       11x17

orient=0 paper orientation (Portrait default)
  =1    Landscape

tray=3        printing tray (Bottom tray default)
  =1 tray 1 (multipurpose slot)
  =2 tray 2

manual=0 no manual feed
  =1     (Manual Feed)

media=0 regular paper
  =1     Transparency
  =2     Letterhead
  =3     Card Stock
  =4     Bond
  =5     Labels
  =6     Prepunched
  =7     Recyled
  =8     Preprinted
  =9     Color (printing on colored paper)

Notes:
The option manual=1 implies tray=1. The media options apply
only to the HP LaserJet 5Si MX model printer.

Examples:
  overheads:
   psmanager <  postscript_file manual=1 media=1 | lpr
  labels:
   psmanager <  postscript_file manual=1 media=5 | lpr
```

```
 Notes:  This code was reverse engineered using output from
         the NeXTStep  printer manager.

 Author:  John Stockwell, June 1995, October 1997

 Reference:
PostScript Printer Description File Format Specification,
version 4.2, Adobe Systems Incorporated
```

```
PSMERGE - MERGE PostScript files

psmerge in= [optional parameters] >postscriptfile

Required Parameters:
in=                     postscript file to merge

Optional Parameters:
origin=0.0,0.0          x,y origin in inches
scale=1.0,1.0           x,y scale factors
rotate=0.0              rotation angle in degrees
translate=0.0,0.0       x,y translation in inches

Notes:
More than one set of in, origin, scale, rotate, and translate
parameters may be specified.  Output x and y coordinates are
determined by:
          x = tx + (x-ox)*sx*cos(d) - (y-oy)*sy*sin(d)
          y = ty + (x-ox)*sx*sin(d) + (y-oy)*sy*cos(d)
where tx,ty are translate coordinates, ox,oy are origin coordinates,
sx,sy are scale factors, and d is the rotation angle.  Note that the
order of operations is shift (origin), scale, rotate, and translate.

If the number of occurrences of a given parameter is less than the number
of input files, then the last occurrence of that parameter will apply to
all subsequent files.
```

```
PSMOVIE - PostScript MOVIE plot of a uniformly-sampled function f(x1,x2,x3)

psmovie n1= [optional parameters] <binaryfile >postscriptfile

Required Parameters:
n1                      number of samples in 1st (fast) dimension

Optional Parameters:
d1=1.0                  sampling interval in 1st dimension
f1=0.0                  first sample in 1st dimension
n2=all                  number of samples in 2nd (slow) dimension
d2=1.0                  sampling interval in 2nd dimension
f2=0.0                  first sample in 2nd dimension
perc=100.0              percentile used to determine clip
clip=(perc percentile) clip used to determine bclip and wclip
bperc=perc              percentile for determining black clip value
wperc=100.0-perc        percentile for determining white clip value
bclip=clip              data values outside of [bclip,wclip] are clipped
wclip=-clip             data values outside of [bclip,wclip] are clipped
d1s=1.0                 factor by which to scale d1 before imaging
d2s=1.0                 factor by which to scale d2 before imaging
verbose=1               =1 for info printed on stderr (0 for no info)
xbox=1.0                offset in inches of left side of axes box
ybox=1.5                offset in inches of bottom side of axes box
wbox=6.0                width in inches of axes box
hbox=8.0                height in inches of axes box
x1beg=x1min             value at which axis 1 begins
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
x2beg=x2min             value at which axis 2 begins
x2end=x2max             value at which axis 2 ends
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelfont=Helvetica     font name for axes labels
labelsize=18            font size for axes labels
title=                  title of plot
```

```
titlefont=Helvetica-Bold font name for title
titlesize=24           font size for title
style=seismic          normal (axis 1 horizontal, axis 2 vertical) or
                       seismic (axis 1 vertical, axis 2 horizontal)
n3=1                   number of samples in third dimension
title2=                second title to annotate different frames
loopdsp=3              display loop type (1=loop over n1; 2=loop over n2;
                                          3=loop over n3)
d3=1.0                 sampling interval in 3rd dimension
f3=d3                  first sample in 3rd dimension

NeXT: view movie via:   psmovie < infile n1= [optional params...] | open
Note: currently only the Preview Application can handle the multipage
      PostScript output by this program.

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic
```

```
PSWIGB - PostScript WIGgle-trace plot of f(x1,x2) via Bitmap
Best for many traces.  Use PSWIGP (Polygon version) for few traces.

pswigb n1= [optional parameters] <binaryfile >postscriptfile

Required Parameters:
n1                      number of samples in 1st (fast) dimension

Optional Parameters:
d1=1.0                  sampling interval in 1st dimension
f1=0.0                  first sample in 1st dimension
n2=all                  number of samples in 2nd (slow) dimension
d2=1.0                  sampling interval in 2nd dimension
f2=0.0                  first sample in 2nd dimension
x2=f2,f2+d2,...         array of sampled values in 2nd dimension
bias=0.0                data value corresponding to location along axis 2
perc=100.0              percentile for determining clip
clip=(perc percentile) data values < bias+clip and > bias-clip are clipped
xcur=1.0                wiggle excursion in traces corresponding to clip
wt=1                    =0 for no wiggle-trace; =1 for wiggle-trace
va=1                    =0 for no variable-area; =1 for variable-area fill
                        =2 for variable area, solid/grey fill
                        SHADING: 2<= va <=5  va=2 lightgrey, va=5 black",
nbpi=72                 number of bits per inch at which to rasterize
verbose=1               =1 for info printed on stderr (0 for no info)
xbox=1.5                offset in inches of left side of axes box
ybox=1.5                offset in inches of bottom side of axes box
wbox=6.0                width in inches of axes box
hbox=8.0                height in inches of axes box
x1beg=x1min             value at which axis 1 begins
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
x2beg=x2min             value at which axis 2 begins
x2end=x2max             value at which axis 2 ends
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
```

```
labelfont=Helvetica    font name for axes labels
labelsize=18           font size for axes labels
title=                 title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24           font size for title
titlecolor=black       color of title
axescolor=black        color of axes
gridcolor=black        color of grid
axeswidth=1            width (in points) of axes
ticwidth=axeswidth     width (in points) of tic marks
gridwidth=axeswidth    width (in points) of grid lines
style=seismic          normal (axis 1 horizontal, axis 2 vertical) or
                       seismic (axis 1 vertical, axis 2 horizontal)
interp=0  no display interpolation
=1 use 8 point sinc interpolation
curve=curve1,curve2,...  file(s) containing points to draw curve(s)
npair=n1,n2,n2,...             number(s) of pairs in each file
curvecolor=black,..    color of curve(s)
curvewidth=axeswidth   width (in points) of curve(s)
curvedash=0            solid curve(s), dash indices 1,...,11 produce
                       curve(s) with various dash styles

Notes:
The interp option may be useful for high nbpi values, however, it
tacitly assumes that the data are purely oscillatory. Non-oscillatory
data will not be represented correctly when this option is set.

The curve file is an ascii file with the points specified as x1 x2
pairs, one pair to a line.  A "vector" of curve files and curve
colors may be specified as curvefile=file1,file2,etc. and
curvecolor=color1,color2,etc, and the number of pairs of values in each
file as npair=npair1,npair2,... .

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
```

```
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic
```

```
 PSWIGP - PostScript WIGgle-trace plot of f(x1,x2) via Polygons
 Best for few traces.  Use PSWIGB (Bitmap version) for many traces.

 pswigp n1= [optional parameters] <binaryfile >postscriptfile

 Required Parameters:
 n1                     number of samples in 1st (fast) dimension

 Optional Parameters:
 d1=1.0                 sampling interval in 1st dimension
 f1=0.0                 first sample in 1st dimension
 n2=all                 number of samples in 2nd (slow) dimension
 d2=1.0                 sampling interval in 2nd dimension
 f2=0.0                 first sample in 2nd dimension
 x2=f2,f2+d2,...        array of sampled values in 2nd dimension
 bias=0.0               data value corresponding to location along axis 2
 perc=100.0             percentile for determining clip
 clip=(perc percentile) data values < bias+clip and > bias-clip are clipped
 xcur=1.0               wiggle excursion in traces corresponding to clip
 fill=1 =0 for no fill;
>0 for pos. fill;
<0 for neg. fill
                               =2 for pos. fill solid, neg. fill grey
                               =-2for neg. fill solid, pos. fill grey
                       SHADING: 2<=abs(fill)<=5  2=lightgrey 5=black
 linewidth=1.0         linewidth in points (0.0 for thinest visible line)
 tracecolor=black       color of traces; should contrast with background
 backcolor=none         color of background; none means no background
 verbose=1              =1 for info printed on stderr (0 for no info)
 xbox=1.5               offset in inches of left side of axes box
 ybox=1.5               offset in inches of bottom side of axes box
 wbox=6.0               width in inches of axes box
 hbox=8.0               height in inches of axes box
 x1beg=x1min            value at which axis 1 begins
 x1end=x1max            value at which axis 1 ends
 d1num=0.0              numbered tic interval on axis 1 (0.0 for automatic)
 f1num=x1min            first numbered tic on axis 1 (used if d1num not 0.0)
 n1tic=1                number of tics per numbered tic on axis 1
 grid1=none             grid lines on axis 1 - none, dot, dash, or solid
 label1=                label on axis 1
 x2beg=x2min            value at which axis 2 begins
 x2end=x2max            value at which axis 2 ends
 d2num=0.0              numbered tic interval on axis 2 (0.0 for automatic)
```

```
f2num=x2min            first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                number of tics per numbered tic on axis 2
grid2=none             grid lines on axis 2 - none, dot, dash, or solid
label2=                label on axis 2
labelfont=Helvetica    font name for axes labels
labelsize=18           font size for axes labels
title=                 title of plot
titlefont=Helvetica-Bold font name for title
titlesize=24           font size for title
titlecolor=black       color of title
axescolor=black        color of axes
gridcolor=black        color of grid
axeswidth=1            width (in points) of axes
ticwidth=axeswidth     width (in points) of tic marks
gridwidth=axeswidth    width (in points) of grid lines
frame=1  =0 wiggle traces only, no frame
style=seismic          normal (axis 1 horizontal, axis 2 vertical) or
                       seismic (axis 1 vertical, axis 2 horizontal)

curve=curve1,curve2,...  file(s) containing points to draw curve(s)
npair=n1,n2,n2,...             number(s) of pairs in each file
curvecolor=black,..    color of curve(s)
curvewidth=axeswidth   width (in points) of curve(s)
curvedash=0            solid curve(s), dash indices 1,...,11 produce
                       curve(s) with various dash styles

Note:  linewidth=0.0 produces the thinest possible line on the output.
device.  Thus the result is device-dependent, put generally looks the
best for seismic traces.

The curve file is an ascii file with the points specified as x1 x2 pairs,
one pair to a line.  A "vector" of curve files and curve colors may
be specified as curvefile=file1,file2,etc. and similarly
curvecolor=color1,color2,etc, and the number of pairs of values
in each file as npair=npair1,npair2,... .

All color specifications may also be made in X Window style Hex format
example:   axescolor=#255

Legal font names are:
AvantGarde-Book AvantGarde-BookOblique AvantGarde-Demi AvantGarde-DemiOblique"
Bookman-Demi Bookman-DemiItalic Bookman-Light Bookman-LightItalic
Courier Courier-Bold Courier-BoldOblique Courier-Oblique
```

```
Helvetica Helvetica-Bold Helvetica-BoldOblique Helvetica-Oblique
Helvetica-Narrow Helvetica-Narrow-Bold Helvetica-Narrow-BoldOblique
Helvetica-Narrow-Oblique NewCentrySchlbk-Bold"
NewCenturySchlbk-BoldItalic NewCenturySchlbk-Roman Palatino-Bold
Palatino-BoldItalic Palatino-Italics Palatino-Roman
SanSerif-Bold SanSerif-BoldItalic SanSerif-Roman
Symbol Times-Bold Times-BoldItalic
Times-Roman Times-Italic ZapfChancery-MediumItalic
```

```
LCMAP - List Color Map of root window of default screen

Usage:   lcmap
```

```
LPROP - List PROPerties of root window of default screen of display

Usage:  lprop
```

```
SCMAP - set default standard color map (RGB_DEFAULT_MAP)

Usage:   scmap
```

```
XCONTOUR - X CONTOUR plot of f(x1,x2) via vector plot call

xcontour n1= [optional parameters] <binaryfile [>psplotfile]

X Functionality:
Button 1 Zoom with rubberband box
Button 2 Show mouse (x1,x2) coordinates while pressed
q or Q key Quit
s key Save current mouse (x1,x2) location to file
p or P key Plot current window with pswigb (only from disk files)

Required Parameters:
n1                      number of samples in 1st (fast) dimension

Optional Parameters:
d1=1.0                  sampling interval in 1st dimension
f1=0.0                  first sample in 1st dimension
x1=f1,f1+d1,...         array of sampled values in 1nd dimension
n2=all                  number of samples in 2nd (slow) dimension
d2=1.0                  sampling interval in 2nd dimension
f2=0.0                  first sample in 2nd dimension
x2=f2,f2+d2,...         array of sampled values in 2nd dimension
mpicks=/dev/tty         file to save mouse picks in
verbose=1               =1 for info printed on stderr (0 for no info)
nc=5                    number of contour values
dc=(zmax-zmin)/nc       contour interval
fc=min+dc               first contour
c=fc,fc+dc,...          array of contour values
cwidth=1.0,...          array of contour line widths
ccolor=none,...         array of contour colors; none means use cgray
cdash=0.0,...           array of dash spacings (0.0 for solid)
labelcf=1               first labeled contour (1,2,3,...)
labelcper=1             label every labelcper-th contour
nlabelc=nc              number of labeled contours (0 no contour label)
nplaces=6               number of decimal places in contour labeling
xbox=50                 x in pixels of upper left corner of window
ybox=50                 y in pixels of upper left corner of window
wbox=550                width in pixels of window
hbox=700                height in pixels of window
x1beg=x1min             value at which axis 1 begins
x1end=x1max             value at which axis 1 ends
d1num=0.0               numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min             first numbered tic on axis 1 (used if d1num not 0.0)
```

```
n1tic=1                 number of tics per numbered tic on axis 1
grid1=none              grid lines on axis 1 - none, dot, dash, or solid
x2beg=x2min             value at which axis 2 begins
x2end=x2max             value at which axis 2 ends
d2num=0.0               numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min             first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1                 number of tics per numbered tic on axis 2
grid2=none              grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelfont=Erg14         font name for axes labels
title=                  title of plot
titlefont=Rom22         font name for title
windowtitle=xwigb       title on window
labelcolor=blue         color for axes labels
titlecolor=red          color for title
gridcolor=blue          color for grid lines
labelccolor=black       color of contour labels                          ",
labelcfont=fixed        font name for contour labels
style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
seismic (axis 1 vertical, axis 2 horizontal)


Notes:
For some reason the contour might slight differ from ones generated
by pscontour (propably due to the pixel nature of the plot
coordinates)

The line width of unlabeled contours is designed as a quarter of that
of labeled contours.


AUTHOR: Morten Wendell Pedersen, Aarhus University

All the coding is based on snippets taken from xwigb, ximage and pscontour
All I have done is put the parts together and put in some bugs ;-)

So credits should go to the authors of these packages...

Caveats and Notes:
The code has been developed under Linux 1.3.20/Xfree 3.1.2E (X11 6.1)
with gcc-2.7.0 But hopefully it should work on other platforms as well

Since all the contours are drawn by Vector plot call's everytime the
```

Window is exposed, the exposing can be darn slow
OOPS This should be history... Now I keep my window content with backing store so I won't have to redraw my window unless I really have to...

Portability Question: I guess I should check if the display supports backingstore and redraw if it doesn't (see DoesBackingStore(3))
I have to be able to use CWBackingStore==Always (other values can be NonUseful and WhenMapped

Since I put the contour labels everytime I draw one contour level the area that contains the label will be crossed by the the next contour lines... (this bug also seems to be present in pscontour)
To fix this I have to redraw all the labels after been through all the contour calls
Right now I can't see a way to fix this without actually to through the entire label positioning again....Overkill I would say

The relative short length of the contour segments will propably mask the cdash settings
which means it is disposable (but I know how to draw dashed lines :)
A way of fixing this could be to get all connected point and then use XDrawlines or XDrawSegments... just an idea...No idea if it'll work.

I think there is a bug in xContour since my plot coordinates increase North and west ward instead of south and eastward

I need to check the Self Doc so if the right parameters are described (I have been through it a couple of times but....)

All functions need a heavy cleanup for unused variables
I suppose there is a couple of memory leaks due to missing free'ing of numerous pointers (especially fonts,GC's & colors could be a problem...

I have to browse through the internal pscontour call... basically what I have done is just putting pscontour instead of pswigb... Instead of repositioning the input file pointer (which doesnt work with pipes) one should consider the use of temporary file
or write your zoombox to pscontour (...how one does that?)

Wish List:
The use of cgray's unused until now... I guess I'll need to allocate a gray Colormap -> meaning that the code not will run at other display

than 8 bit Pseudocolor :((with the use of present version of the colormap library (code in $CWPROOT/src/xplot/lib))

The format of contour label should be open for the user..

It could be nice if one could choose to have a pixmap (like ximage) underlying the contours... this should be defined either by the input data or by a seperate file
eg useful for viewing traveltime contours on top a plot of the velocity field

```
XESPB - X windows display of Encapsulated PostScript as a single Bitmap

Usage:   xepsb < stdin

Caveat: your system must have Display PostScript Graphics

NOTE: This program is included as a demo of EPS -> X programming.
See:  xepsp and xpsp these are more advanced versions
```

```
XEPSP - X windows display of Encapsulated PostScript

Usage:   xepsp < stdin

Caveat: your system must have Display PostScript Graphics
Note:
this program is a more advanced version of   xepsb. See also:  xepsp
```

```
XIMAGE - X IMAGE plot of a uniformly-sampled function f(x1,x2)

ximage n1= [optional parameters] <binaryfile

X Functionality:
Button 1 Zoom with rubberband box
Button 2 Show mouse (x1,x2) coordinates while pressed
q or Q key Quit
s key Save current mouse (x1,x2) location to file
p or P key Plot current window with pswigb (only from disk files)
a or page up keys enhance clipping by 10%
c or page down keys reduce clipping by 10%
up,down,left,right keys move zoom window by half width/height
i or +(keypad)  zoom in by factor 2
o or -(keypad)  zoom out by factor 2

... change colormap interactively
r      install next RGB - colormap
R      install previous RGB - colormap
h      install next HSV - colormap
H      install previous HSV - colormap
H      install previous HSV - colormap
(Move mouse cursor out and back into window for r,R,h,H to take effect)

Required Parameters:
n1  number of samples in 1st (fast) dimension

Optional Parameters:
d1=1.0  sampling interval in 1st dimension
f1=0.0  first sample in 1st dimension
n2=all  number of samples in 2nd (slow) dimension
d2=1.0  sampling interval in 2nd dimension
f2=0.0  first sample in 2nd dimension
mpicks=/dev/tty  file to save mouse picks in
perc=100.0  percentile used to determine clip
clip=(perc percentile) clip used to determine bclip and wclip
bperc=perc  percentile for determining black clip value
wperc=100.0-perc  percentile for determining white clip value
bclip=clip  data values outside of [bclip,wclip] are clipped
wclip=-clip  data values outside of [bclip,wclip] are clipped
balance=0  bclip & wclip individually
=1 set them to the same abs value
  if specified via perc (avoids colorbar skew)
```

```
 cmap=hsv\'n\' or rgb\'m\'\'n\' is a number from 0 to 13
\'m\' is a number from 0 to 11
cmap=rgb0 is equal to cmap=gray
cmap=hsv1 is equal to cmap=hue
(compatibility to older versions)
 legend=0         =1 display the color scale
 units= unit label for legend
 legendfont=times_roman10    font name for title
 verbose=1 =1 for info printed on stderr (0 for no info)
 xbox=50 x in pixels of upper left corner of window
 ybox=50 y in pixels of upper left corner of window
 wbox=550 width in pixels of window
 hbox=700 height in pixels of window
 lwidth=16 colorscale (legend) width in pixels
 lheight=hbox/3 colorscale (legend) height in pixels
 lx=3 colorscale (legend) x-position in pixels
 ly=(hbox-lheight)/3   colorscale (legend) y-position in pixels
 x1beg=x1min value at which axis 1 begins
 x1end=x1max value at which axis 1 ends
 d1num=0.0 numbered tic interval on axis 1 (0.0 for automatic)
 f1num=x1min first numbered tic on axis 1 (used if d1num not 0.0)
 n1tic=1 number of tics per numbered tic on axis 1
 grid1=none grid lines on axis 1 - none, dot, dash, or solid
 label1= label on axis 1
 x2beg=x2min value at which axis 2 begins
 x2end=x2max value at which axis 2 ends
 d2num=0.0 numbered tic interval on axis 2 (0.0 for automatic)
 f2num=x2min first numbered tic on axis 2 (used if d2num not 0.0)
 n2tic=1 number of tics per numbered tic on axis 2
 grid2=none grid lines on axis 2 - none, dot, dash, or solid
 label2= label on axis 2
 labelfont=Erg14 font name for axes labels
 title= title of plot
 titlefont=Rom22 font name for title
 windowtitle=ximage title on window
 labelcolor=blue color for axes labels
 titlecolor=red color for title
 gridcolor=blue color for grid lines
 style=seismic         normal (axis 1 horizontal, axis 2 vertical) or
seismic (axis 1 vertical, axis 2 horizontal)
 blank=0 This indicates what portion of the lower range
to blank out (make the background color).  The
value should range from 0 to 1.
```

```
plotfile=plotfile.ps  filename for interactive ploting (P)
curve=curve1,curve2,...  file(s) containing points to draw curve(s)
npair=n1,n2,n2,...            number(s) of pairs in each file
curvecolor=color1,color2,...  color(s) for curve(s)
blockinterp=0       whether to use block interpolation (0=no, 1=yes)


NOTES:
The curve file is an ascii file with the points  specified as x1 x2
pairs separated by a space, one pair to a line.  A "vector" of curve
files and curve colors may be specified as curvefile=file1,file2,etc.
and curvecolor=color1,color2,etc, and the number of pairs of values
in each file as npair=npair1,npair2,... .


AUTHOR:  Dave Hale, Colorado School of Mines, 08/09/90

Stewart A. Levin, Mobil - Added ps print option

Brian Zook, Southwest Research Institute, 6/27/96, added blank option

Toralf Foerster, Baltic Sea Research Institute, 9/15/96, new colormaps

Berend Scheffers, Delft, colorbar (legend)

Brian K. Macy, Phillips Petroleum, 11/27/98, added curve plotting option

G.Klein, GEOMAR Kiel, 2004-03-12, added cursor scrolling and
                                  interactive change of zoom and clipping.

Zhaobo Meng, ConocoPhillips, 12/02/04, added amplitude display

Garry Perratt, Geocon, 08/04/05, modified perc handling to center colorbar if balance=

INTL2B_block - blocky interpolation of a 2-D array of bytes

intl2b_block blocky interpolation of a 2-D array of bytes

Function Prototype:
void intl2b_block(int nxin, float dxin, float fxin,
int nyin, float dyin, float fyin, unsigned char *zin,
int nxout, float dxout, float fxout,
int nyout, float dyout, float fyout, unsigned char *zout);
```

```
Input:
nxin number of x samples input (fast dimension of zin)
dxin x sampling interval input
fxin first x sample input
nyin number of y samples input (slow dimension of zin)
dyin y sampling interval input
fyin first y sample input
zin array[nyin][nxin] of input samples (see notes)
nxout number of x samples output (fast dimension of zout)
dxout x sampling interval output
fxout first x sample output
nyout number of y samples output (slow dimension of zout)
dyout y sampling interval output
fyout first y sample output

Output:
zout array[nyout][nxout] of output samples (see notes)

Notes:
The arrays zin and zout must passed as pointers to the first element of
a two-dimensional contiguous array of unsigned char values.

Constant extrapolation of zin is used to compute zout for
output x and y outside the range of input x and y.

Author:  James Gunning, CSIRO Petroleum 1999. Hacked from
intl2b() by Dave Hale, Colorado School of Mines, c. 1989-1991
```

```
 XPICKER - X wiggle-trace plot of f(x1,x2) via Bitmap with PICKing

 xpicker n1= [optional parameters] <binaryfile

 X Menu functionality:
    Pick Filename Window default is pick_file
    Load load an existing Pick Filename
    Save save to Pick Filename
    View only/Pick default is View, click to enable Picking
    Add/Delete default is Add, click to delete picks
    Cross off/on default is Cross off, click to enable Crosshairs

 In View mode:
    a or page up keys          enhance clipping by 10%
    c or page down keys        reduce clipping by 10%
    up,down,left,right keys    move zoom window by half width/height
    i or +(keypad)             zoom in by factor 2
    o or -(keypad)             zoom out by factor 2
    l  lock the zoom while moving the coursor
    u  unlock the zoom

 Notes:
Menu selections and toggles ("clicks") are made with button 1
Pick selections are made with button 3
Edit a pick selection by dragging it with button 3 down or
by making a new pick on that trace
Reaching the window limits while moving within changes the zoom
factor in this direction. The use of zoom locking(l) disables it

 Other X Mouse functionality:
 Mouse Button 1 Zoom with rubberbox
 Mouse Button 2 Show mouse (x1,x2) coordinates while pressed

 The following keys are active in View Only mode:

 Required Parameters:
 n1= number of samples in 1st (fast) dimension

 Optional Parameters:
 mpicks=pick_file name of output (input) pick file
 d1=1.0 sampling interval in 1st dimension
 f1=d1   first sample in 1st dimension
 n2=all  number of samples in 2nd (slow) dimension
```

```
 d2=1.0  sampling interval in 2nd dimension
 f2=d2   first sample in 2nd dimension
 x2=f2,f2+d2,... array of sampled values in 2nd dimension
 bias=0.0        data value corresponding to location along axis 2
 perc=100.0      percentile for determining clip
 clip=(perc percentile) data values < bias+clip and > bias-clip are clipped
 xcur=1.0        wiggle excursion in traces corresponding to clip
 wt=1    =0 for no wiggle-trace; =1 for wiggle-trace
 va=1    =0 for no variable-area; =1 for variable-area fill
                      =2 for variable area, solid/grey fill
                      SHADING: 2<=va<=5  va=2 light grey, va=5 black
 verbose=1       =1 for info printed on stderr (0 for no info)
 xbox=50 x in pixels of upper left corner of window
 ybox=50 y in pixels of upper left corner of window
 wbox=550        width in pixels of window
 hbox=700 height in pixels of window
 x1beg=x1min value at which axis 1 begins
 x1end=x1max value at which axis 1 ends
 d1num=0.0  numbered tic interval on axis 1 (0.0 for automatic)
 f1num=x1min first numbered tic on axis 1 (used if d1num not 0.0)
 n1tic=1 number of tics per numbered tic on axis 1
 grid1=none grid lines on axis 1 - none, dot, dash, or solid
 label1= label on axis 1
 x2beg=x2min value at which axis 2 begins
 x2end=x2max value at which axis 2 ends
 d2num=0.0  numbered tic interval on axis 2 (0.0 for automatic)
 f2num=x2min first numbered tic on axis 2 (used if d2num not 0.0)
 n2tic=1 number of tics per numbered tic on axis 2
 grid2=none grid lines on axis 2 - none, dot, dash, or solid
 label2= label on axis 2
 labelfont=Erg14 font name for axes labels
 title= title of plot
 titlefont=Rom22 font name for title
 labelcolor=blue color for axes labels
 titlecolor=red color for title
 gridcolor=blue color for grid lines
 style=seismic normal (axis 1 horizontal, axis 2 vertical) or
       seismic (axis 1 vertical, axis 2 horizontal)
 endian= =0 little endian, =1 big endian
 interp=0 no sinc interpolation
=1 perform sinc interpolation
 x2file= file of "acceptable" x2 values
 x1x2=1 save picks in the order (x1,x2)
```

```
 =0 save picks in the order (x2,x1)

Notes:
Xpicker will try to detect the endian value of the X-display and will
set it to the right value. If it gets obviously wrong information the
endian value will be set to the endian value of the machine that is
given at compile time as the value of CWPENDIAN defined in cwp.h
and set via the compile time flag ENDIANFLAG in Makefile.config.

The only time that you might want to change the value of the endian
variable is if you are viewing traces on a machine with a different
byte order than the machine you are creating the traces on AND if for
some reason the automaic detection of the display byte order fails.
Set endian to that of the machine you are viewing the traces on.

The interp flag is useful for making better quality wiggle trace for
making plots from screen dumps. However, this flag assumes that the
data are purely oscillatory. This option may not be appropriate for all
data sets.

If the x2file=  option is set, then the values from the specified file
will define the set of "acceptable" values of x2 for xpicker to
output. The format is a single column of ASCII values. The number of
specified values is arbitrary.

Such a file can be built from an SU data set via:
    sugethw < sudata key=offset output=geom > x2example

If the value of x2file= is not set, then
xpicker will use the values specified via: x2=.,.,.,. or those that are",
computed from the values of f2=  and d2= as being the "acceptible
values.

See the selfdoc of  suxpicker  for information on using key fields from
the SU trace headers directly.



AUTHOR:  Dave Hale, Colorado School of Mines, 08/09/90
with picking by Wenying Cai of University of Utah.
Endian stuff by Morten Pedersen and John Stockwell of CWP.
Interp stuff by Tony Kocurko of Memorial University of Newfoundland
Modified to include acceptable values by Bill Lutter of the
```

```
    Department of Geology, University of Wisconsin 10/96
 MODIFIED:  P. Michaels, Boise State Univeristy  29 December 2000
            Added solid/grey color scheme for peaks/troughs

 G.Klein, IFG Kiel University, 2003-09-29, added cursor scrolling and
            interactive change of zoom and clipping.

 NOTES:
 Interactive picker improved to allow x-axis of picks to be
 coordinated with "key=header" parameter set in driver routine
 suxpicker. Multiple picks per trace are now allowed.

  Input:
  The command line of suxpicker is unchanged.  The parameter"key=header"
  set in  suxpicker controls a) trace x-axis  displayed via xpicker and
  b) the header values in the first column of a pick file either read in
     or written out from xpicker c) header values expected in optional file
      or written out from xpicker c) header values expected in optional file
     x2file= which reads into xpicker allowable trace x-axis values.

   a) example command line:  suxpicker key=offset < shot10.plotpik

   b) pick file format:
x-axis_value_1 time_1
x-axis_value_2 time_2
x-axis_value_3 time_3
etc.
x-axis_value_n time_n

pick file example:
         1000.000000 0.500000
         2000.000000 1.000000
         3000.000000 1.500000
         4000.000000 2.000000
         5000.000000 2.500000

  c)  format of optional file x2file=:
        header_value_1
     header_value_2
   etc.
   header_val_m

       If file "x2file=" exists in directory from which suxpicker is
```

```
    invoked, then these trace header x-axis values are the only allowable
    x-axis pick values used in the pick "add" or "delete" menu operation.
    Header values do not need to be sorted or 1 to 1 with input traces.
    Further, pick file x-axis values can be read into xpicker via load
    operation without having to match key_pickx1_val x-axis values and
    can also be rewritten out an output pickfile.  As indicated, only
    the "add" and "delete" pick operations are influenced by existence
     of this file.

    Offset header values for "x2file=" can be generated by the
    command line:

    sugethw < su_segyfile key=offset output=geom >  x2examplefile=

    Output: Only change is in format of pick_file (format described above).
    If x2file= file exists then x-axis value of added picks
    will be forced to nearest allowable trace x-axis value (input values
    of x2file= file). If x2file= is not set, then the values of x2
    that are either assigned uniformly to the traces via f2 and d2,
    or by the vector of values of x2=.,.,.,.    will be the "acceptable"
    values.

  Strategy:
 a) malloc() and realloc() used to dynamically allocate memory
for array of x-axis value read in from optional file
x2file=.  This is done in function read_keyval().

b) The pick file dimensions are set in main program via malloc()
and then initialized (*apick)[i].picked = FALSE) in function
init_picks().  The pick file is declared as pick_t **apick, in
order to use realloc() as needed in functions load_picks where the
pick file is read in and edit_picks where picks are added.  The
call to realloc() and further initializing is performed in
function realloc_picks().

c) If x2file= file exists the mouse derived x-axis value
for a pick to be added is checked against allowable x-axis values
to find the closest match via function add_pick called from
edit_picks.  If the pick is to be deleted, first a search is done
to find the closest x-axis value, then the existing pick values
are searched to find the closest radial value (x**2 + t**2) via
function del_pick() invoked from edit_picks.
```

d) Code modifications are limited to above mentioned functions, except for additional parameters passed to functions edit_picks, load_picks, save_picks, and check_buttons.

```
XPSP - Display conforming PostScript in an X-window

xpsp < stdin

Note: this is the most advanced version of xepsb and xepsp.
Caveat: your  system must have Display PostScript Graphics
```

```
 XWIGB - X WIGgle-trace plot of f(x1,x2) via Bitmap

 xwigb n1= [optional parameters] <binaryfile

 X Functionality:
 Button 1 Zoom with rubberband box
 Button 2 Show mouse (x1,x2) coordinates while pressed
 q or Q key Quit
 s key Save current mouse (x1,x2) location to file
 p or P key Plot current window with pswigb (only from disk files)
 a or page up keys enhance clipping by 10%
 c or page down keys reduce clipping by 10%
 up,down,left,right keys move zoom window by half width/height
 i or +(keypad)  zoom in by factor 2
 o or -(keypad)  zoom out by factor 2
    l  lock the zoom while moving the coursor
    u  unlock the zoom
 1,2,...,9 Zoom/Move factor of the window size

 Notes:
Reaching the window limits while moving within changes the zoom
factor in this direction. The use of zoom locking(l) disables it

 Required Parameters:
 n1  number of samples in 1st (fast) dimension

 Optional Parameters:
 d1=1.0  sampling interval in 1st dimension
 f1=0.0  first sample in 1st dimension
 n2=all  number of samples in 2nd (slow) dimension
 d2=1.0  sampling interval in 2nd dimension
 f2=0.0  first sample in 2nd dimension
 x2=f2,f2+d2,...  array of sampled values in 2nd dimension
 mpicks=/dev/tty  file to save mouse picks in
 bias=0.0  data value corresponding to location along axis 2
 perc=100.0  percentile for determining clip
 clip=(perc percentile) data values < bias+clip and > bias-clip are clipped
 xcur=1.0  wiggle excursion in traces corresponding to clip
 wt=1  =0 for no wiggle-trace; =1 for wiggle-trace
 va=1  =0 for no variable-area; =1 for variable-area fill
                         =2 for variable area, solid/grey fill
                         SHADING: 2<=va<=5  va=2 light grey, va=5 black
 verbose=0  =1 for info printed on stderr (0 for no info)
```

```
xbox=50  x in pixels of upper left corner of window
ybox=50  y in pixels of upper left corner of window
wbox=550  width in pixels of window
hbox=700  height in pixels of window
x1beg=x1min  value at which axis 1 begins
x1end=x1max  value at which axis 1 ends
d1num=0.0  numbered tic interval on axis 1 (0.0 for automatic)
f1num=x1min  first numbered tic on axis 1 (used if d1num not 0.0)
n1tic=1  number of tics per numbered tic on axis 1
grid1=none  grid lines on axis 1 - none, dot, dash, or solid
x2beg=x2min  value at which axis 2 begins
x2end=x2max  value at which axis 2 ends
d2num=0.0  numbered tic interval on axis 2 (0.0 for automatic)
f2num=x2min  first numbered tic on axis 2 (used if d2num not 0.0)
n2tic=1  number of tics per numbered tic on axis 2
grid2=none  grid lines on axis 2 - none, dot, dash, or solid
label2=  label on axis 2
labelfont=Erg14  font name for axes labels
title=  title of plot
titlefont=Rom22  font name for title
windowtitle=xwigb  title on window
labelcolor=blue  color for axes labels
titlecolor=red  color for title
gridcolor=blue  color for grid lines
style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
seismic (axis 1 vertical, axis 2 horizontal)
endian=  =0 little endian =1 big endian
interp=0  no interpolation in display
=1 use 8 point sinc interpolation
wigclip=0  If 0, the plot box is expanded to accommodate
the larger wiggles created by xcur>1. If this
flag is non-zero, the extra-large wiggles are
are clipped at the boundary of the plot box.
plotfile=plotfile.ps   filename for interactive ploting (P)
curve=curve1,curve2,...  file(s) containing points to draw curve(s)
npair=n1,n2,n2,...            number(s) of pairs in each file
curvecolor=color1,color2,...  color(s) for curve(s)

Notes:
Xwigb will try to detect the endian value of the X-display and will
set it to the right value. If it gets obviously wrong information the
endian value will be set to the endian value of the machine that is
given at compile time as the value of CWPENDIAN defined in cwp.h
```

```
and set via the compile time flag ENDIANFLAG in Makefile.config.

The only time that you might want to change the value of the endian
variable is if you are viewing traces on a machine with a different
byte order than the machine you are creating the traces on AND if for
some reason the automaic detection of the display byte order fails.
Set endian to that of the machine you are viewing the traces on.

The interp flag is useful for making better quality wiggle trace for
making plots from screen dumps. However, this flag assumes that the
data are purely oscillatory. This option may not be appropriate for all
data sets.

The curve file is an ascii file with the points specified as x1 x2
pairs, separated by a space, one pair to a line.  A "vector" of curve
files and curve colors may be specified as curvefile=file1,file2,etc.
and curvecolor=color1,color2,etc, and the number of pairs of values
in each file as npair=npair1,npair2,... .


AUTHOR:  Dave Hale, Colorado School of Mines, 08/09/90

Endian stuff by:
   Morten Wendell Pedersen, Aarhus University (visiting CSM, June 1995)
 & John Stockwell, Colorado School of Mines, 5 June 1995

Stewart A. Levin, Mobil - Added ps print option
John Stockwell - Added optional sinc interpolation
Stewart A. Levin, Mobil - protect title, labels in pswigb call

Brian J. Zook, SwRI - Added style=normal and wigclip flag

Brian K. Macy, Phillips Petroleum, 11/27/98, added curve plotting option
Curve plotting notes:
MODIFIED:  P. Michaels, Boise State Univeristy  29 December 2000
           Added solid/grey color scheme for peaks/troughs

G.Klein, IFG Kiel University, 2002-09-29, added cursor scrolling and
           interactive change of zoom and clipping.
         IFM-GEOMAR Kiel, 2004-03-12, added zoom locking
         IFM-GEOMAR Kiel, 2004-03-25, interactive plotting fixed
```

```
XGRAPH - X GRAPHer
Graphs n[i] pairs of (x,y) coordinates, for i = 1 to nplot.

xgraph n= [optional parameters] <binaryfile

X Functionality:
q or Q key    Quit

Required Parameters:
n                       array containing number of points per plot

Optional Parameters:
nplot=number of n's     number of plots
d1=0.0,...              x sampling intervals (0.0 if x coordinates input)
f1=0.0,...              first x values (not used if x coordinates input)
d2=0.0,...              y sampling intervals (0.0 if y coordinates input)
f2=0.0,...              first y values (not used if y coordinates input)
pairs=1,...             =1 for data pairs in format 1.a, =0 for format 1.b
linewidth=1,1,...       line widths in pixels (0 for no lines)
linecolor=2,3,...       line colors (black=0, white=1, 2,3,4 = RGB, ...)
mark=0,1,2,3,...        indices of marks used to represent plotted points
marksize=0,0,...        size of marks in pixels (0 for no marks)
x1beg=x1min             value at which axis 1 begins
x1end=x1max             value at which axis 1 ends
x2beg=x2min             value at which axis 2 begins
x2end=x2max             value at which axis 2 ends
reverse=0               =1 to reverse sequence of plotting curves ",

Optional resource parameters (defaults taken from resource database):
windowtitle=        title on window
width=                  width in pixels of window
height=                 height in pixels of window
nTic1=                  number of tics per numbered tic on axis 1
grid1=                  grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
nTic2=                  number of tics per numbered tic on axis 2
grid2=                  grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelFont=              font name for axes labels
title=                  title of plot
titleFont=              font name for title
titleColor=             color for title
axesColor=              color for axes
```

```
gridColor=              color for grid lines
style=                  normal (axis 1 horizontal, axis 2 vertical) or
                        seismic (axis 1 vertical, axis 2 horizontal)

Data formats supported:
 1.a. x1,y1,x2,y2,...,xn,yn
   b. x1,x2,...,xn,y1,y2,...,yn
 2. y1,y2,...,yn (must give non-zero d1[]=)
 3. x1,x2,...,xn (must give non-zero d2[]=)
 4. nil (must give non-zero d1[]= and non-zero d2[]=)
  The formats may be repeated and mixed in any order, but if
  formats 2-4 are used, the d1 and d2 arrays must be specified including
  d1[]=0.0 d2[]=0.0 entries for any internal occurences of format 1.
  Similarly, the pairs array must contain place-keeping entries for
  plots of formats 2-4 if they are mixed with both formats 1.a and 1.b.
  Also, if formats 2-4 are used with non-zero f1[] or f2[] entries, then
  the corresponding array(s) must be fully specified including f1[]=0.0
  and/or f2[]=0.0 entries for any internal occurences of format 1 or
  formats 2-4 where the zero entries are desired.
mark index:
1. asterisk
2. x-cross
3. open triangle
4. open square
5. open circle
6. solid triangle
7. solid square
8. solid circle

Note: n1 and n2 are acceptable aliases for n and nplot, respectively.

Example:
xgraph n=50,100,20 d1=2.5,1,0.33 <datafile
  plots three curves with equally spaced x values in one plot frame
  x1-coordinates are x1(i) = f1+i*d1 for i = 1 to n (f1=0 by default)
  number of x2's and then x2-coordinates for each curve are read
  sequentially from datafile.
```

```
XMOVIE - image one or more frames of a uniformly sampled function f(x1,x2)

xmovie n1= n2= [optional parameters] <fileoffloats

X Functionality:
Button 1 Zoom with rubberband box
Button 2  reverse the direction of the movie.
Button 3  stop and start the movie.
q or Q key Quit
s or S key stop display and switch to Step mode
b or B key set frame direction to Backward
f or F key set frame direction to Forward
n or N key same as 'f'
c or C key set display mode to Continuous mode

Required Parameters:
n1=     number of samples in 1st (fast) dimension
n2=     number of samples in 2nd (slow) dimension

Optional Parameters:
d1=1.0  sampling interval in 1st dimension
f1=0.0  first sample in 1st dimension
d2=1.0  sampling interval in 2nd dimension
f2=0.0  first sample in 2nd dimension
perc=100.0      percentile used to determine clip
clip=(perc percentile) clip used to determine bclip and wclip
bperc=perc      percentile for determining black clip value
wperc=100.0-perc       percentile for determining white clip value
bclip=clip      data values outside of [bclip,wclip] are clipped
wclip=-clip     data values outside of [bclip,wclip] are clipped
x1beg=x1min     value at which axis 1 begins
x1end=x1max     value at which axis 1 ends
x2beg=x2min     value at which axis 2 begins
x2end=x2max     value at which axis 2 ends
fframe=1        value corresponding to first frame
dframe=1        frame sampling interval
loop=0  =1 to loop over frames after last frame is input
=2 to run movie back and forth
interp=1        =0 for a non-interpolated, blocky image
verbose=1       =1 for info printed on stderr (0 for no info)
idm=0   =1 to set initial display mode to stepmode

Optional resource parameters (defaults taken from resource database):
```

```
 windowtitle=        title on window and icon
 width=  width in pixels of window
 height= height in pixels of window
 nTic1=  number of tics per numbered tic on axis 1
 grid1=  grid lines on axis 1 - none, dot, dash, or solid
 label1= label on axis 1
 nTic2=  number of tics per numbered tic on axis 2
 grid2=  grid lines on axis 2 - none, dot, dash, or solid
 label2= label on axis 2
 labelFont=      font name for axes labels
 title=  title of plot
 titleFont=      font name for title
 titleColor=     color for title
 axesColor=      color for axes
 gridColor=      color for grid lines
 style=  normal (axis 1 horizontal, axis 2 vertical) or
seismic (axis 1 vertical, axis 2 horizontal)
 sleep=  delay between frames in seconds (integer)

 Color options:
 cmap=gray     gray, hue, saturation, or default colormaps may be specified
 bhue=0 hue mapped to bclip (hue and saturation maps)
 whue=240      hue mapped to wclip (hue and saturation maps)
 sat=1  saturation (hue map only)
 bright=1      brightness (hue and saturation maps)
 white=(bclip+wclip)/2  data value mapped to white (saturation map only)

 Notes:
 Colors are specified using the HSV color wheel model:
   Hue:  0=360=red, 60=yellow, 120=green, 180=cyan, 240=blue, 300=magenta
   Saturation:  0=white, 1=pure color
   Value (brightness):  0=black, 1=maximum intensity
 For the saturation mapping (cmap=sat), data values between white and bclip
   are mapped to bhue, with saturation varying from white to the pure color.
   Values between wclip and white are similarly mapped to whue.
 For the hue mapping (cmap=hue), data values between wclip and bclip are
   mapped to hues between whue and bhue.  Intermediate hues are found by
   moving counterclockwise around the circle from bhue to whue.  To reverse
   the polarity of the image, exchange bhue and whue.  Equivalently,
   exchange bclip and wclip (setting perc=0 is an easy way to do this).
   Hues in excess of 360 degrees can be specified in order to reach the
   opposite side of the color circle, or to wrap around the circle more
   than once.
```

```
The title string may contain a C printf format string containing a
  conversion character for the frame number.  The frame number is
  computed from dframe and fframe.  E.g., try setting title="Frame %g".
```

```
XRECTS - plot rectangles on a two-dimensional grid

xrects x1min= x1max= x2min= x2max= [optional parameters] <rectangles

Required Parameters:
x1min                   minimum x1 coordinate
x1max                   maximum x1 coordinate
x2min                   minimum x2 coordinate
x2max                   maximum x2 coordinate

Optional Parameters:
color=red               color used for rectangules

Optional resource parameters (defaults taken from resource database):
width=                  width in pixels of window
height=                 height in pixels of window
nTic1=                  number of tics per numbered tic on axis 1
grid1=                  grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
nTic2=                  number of tics per numbered tic on axis 2
grid2=                  grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelFont=              font name for axes labels
title=                  title of plot
titleFont=              font name for title
titleColor=             color for title
axesColor=              color for axes
gridColor=              color for grid lines
style=                  normal (axis 1 horizontal, axis 2 vertical) or
                        seismic (axis 1 vertical, axis 2 horizontal)
```

```
FFTLAB - Motif-X based graphical 1D Fourier Transform

Usage:  fftlab

Caveat: you must have the Motif Developer's package to install this code
```

```
 SUPSCONTOUR - PostScript CONTOUR plot of a segy data set

 supscontour <stdin [optional parameters] | ...

 Optional parameters:

 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 See the pscontour selfdoc for the remaining parameters.

 On NeXT: supscontour < infile [optional parameters]  | open

 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2

 Credits:
```

```
CWP: Dave Hale and Zhiming Li (pscontour, etc.)
   Jack Cohen and John Stockwell (supscontour, etc.)
      Delphi: Alexander Koek, added support for irregularly spaced traces
      Aarhus University: Morten W. Pedersen copied everything from the xwigb
                         source and just replaced all occurencies of the word

 Notes:
When the number of traces isn't known, we need to count
the traces for pscontour.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

If your system does make a disk file, consider altering
the code to remove the file on interrupt.  This could be
done either by trapping the interrupt with "signal"
or by using the "tmpnam" routine followed by an immediate
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SUPSCUBECONTOUR - PostScript CUBE plot of a segy data set

supscubecontour <stdin [optional parameters] | ...

Optional parameters:

n2 is the number of traces per frame.  If not getparred then it
is the total number of traces in the data set.

n3 is the number of frames.  If not getparred then it
is the total number of frames in the data set measured by ntr/n2

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the pscubecontour selfdoc for the remaining parameters.

example:   supscubecontour < infile [optional parameters]  | gv -
```

```
 Credits:

CWP: Dave Hale and Zhiming Li (pscube)
     Jack K. Cohen  (suxmovie)
     John Stockwell (supscubecontour)

 Notes:
When n2 isn't getparred, we need to count the traces
for pscube. Although we compute ntr, we don't allocate a 2-d array
and content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SUPSCUBE - PostScript CUBE plot of a segy data set

supscube <stdin [optional parameters] | ...

Optional parameters:

n2 is the number of traces per frame.  If not getparred then it
is the total number of traces in the data set.

n3 is the number of frames.  If not getparred then it
is the total number of frames in the data set measured by ntr/n2

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the pscube selfdoc for the remaining parameters.

On NeXT:     supscube < infile [optional parameters]  | open
```

```
 Credits:

CWP: Dave Hale and Zhiming Li (pscube)
     Jack K. Cohen  (suxmovie)
     John Stockwell (supscube)

 Notes:
When n2 isn't getparred, we need to count the traces
for pscube. Although we compute ntr, we don't allocate a 2-d array
and content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSGRAPH - PostScript GRAPH plot of a segy data set

 supsgraph <stdin [optional parameters] | ...

 Optional parameters:
 style=seismic seismic is default here, =normal is alternate
(see psgraph selfdoc for style definitions)

 nplot is the number of traces (ntr is an acceptable alias for nplot)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to over-ride the header values if
 not.

 See the psgraph selfdoc for the remaining parameters.

 On NeXT:     supsgraph < infile [optional parameters]  | open

 Credits:
```

```
CWP: Dave Hale and Zhiming Li (pswigp, etc.)
   Jack Cohen and John Stockwell (supswigp, etc.)

 Notes:
When the number of traces isn't known, we need to count
the traces for pswigp.  You can make this value "known"
either by getparring nplot or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSIMAGE - PostScript IMAGE plot of a segy data set

 supsimage <stdin [optional parameters] | ...

 Optional parameters:

 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 See the psimage selfdoc for the remaining parameters.

 On NeXT: supsimage < infile [optional parameters]  | open

 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2

 Credits:
```

```
CWP: Dave Hale and Zhiming Li (psimage, etc.)
   Jack Cohen and John Stockwell (supsimage, etc.)

 Notes:
When the number of traces isn't known, we need to count
the traces for psimage.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SUPSMAX - PostScript of the MAX, min, or absolute max value on each trace
    of a SEGY (SU) data set

  supsmax <stdin >postscript file [optional parameters]

Optional parameters:
mode=max max value
 =min min value
 =abs absolute max value

n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the sumax selfdoc for additional parameter.
See the psgraph selfdoc for the remaining parameters.
```

```
 Credits:

CWP: John Stockwell, based on Jack Cohen's SU JACKet

 Notes:
When the number of traces isn't known, we need to count
the traces for psgraph.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.
```

```
SUPSMOVIE - PostScript MOVIE plot of a segy data set

supsmovie <stdin [optional parameters] | ...

Optional parameters:

n2 is the number of traces per frame.  If not getparred then it
is the total number of traces in the data set.

n3 is the number of frames.  If not getparred then it
is the total number of frames in the data set measured by ntr/n2

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the psmovie selfdoc for the remaining parameters.

On NeXT:     supsmovie < infile [optional parameters]  | open
```

```
 Credits:

CWP: Dave Hale and Zhiming Li (psmovie)
     Jack K. Cohen  (suxmovie)
     John Stockwell (supsmovie)

 Notes:
When n2 isn't getparred, we need to count the traces
for psmovie.  In this case:
we are using tmpfile because on many machines it is
implemented as a memory area instead of a disk file.
Although we compute ntr, we don't allocate a 2-d array
and content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSWIGB - PostScript Bit-mapped WIGgle plot of a segy data set

 supswigb <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
 vsp (same as normal with axis 2 reversed)
 Note: vsp requires use of a keyword

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type   sukeyword -o   to see the complete list of SU keywords.

 This program is really just a wrapper for the plotting program: pswigb
 See the pswigb selfdoc for the remaining parameters.
```

```
 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2, keyword (if set)

 Credits:

CWP: Dave Hale and Zhiming Li (pswigb, etc.)
   Jack Cohen and John Stockwell (supswigb, etc.)
      Delphi: Alexander Koek, added support for irregularly spaced traces

Modified by Brian Zook, Southwest Research Institute, to honor
 scale factors, added vsp style

 Notes:
When the number of traces isn't known, we need to count
the traces for pswigb.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUPSWIGP - PostScript Polygon-filled WIGgle plot of a segy data set

 supswigp <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
 vsp (same as normal with axis 2 reversed)
 Note: vsp requires use of a keyword

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type   sukeyword -o   to see the complete list of SU keywords.

 This program is really just a wrapper for the plotting program: pswigp
 See the pswigp selfdoc for the remaining parameters.
```

```
 On NeXT: supswigp < infile [optional parameters]  | open

 Trace header fields accessed: ns, ntr, tracr, tracl, delrt, trid,
dt, d1, d2, f1, f2, key specified by key

 Credits:

CWP: Dave Hale and Zhiming Li (pswigp, etc.)
   Jack Cohen and John Stockwell (supswigp, etc.)
Delphi: Alexander Koek, added support for irregularly spaced traces

Modified by Brian Zook, Southwest Research Institute, to honor
 scale factors, added vsp style

 Notes:
When the number of traces isn't known, we need to count
the traces for pswigp.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

If your system does make a disk file, consider altering
the code to remove the file on interrupt.  This could be
done either by trapping the interrupt with "signal"
or by using the "tmpnam" routine followed by an immediate
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXCONTOUR - X CONTOUR plot of Seismic UNIX tracefile via vector plot call

 suxwigb <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type   sukeyword -o   to see the complete list of SU keywords.

 This program is really just a wrapper for the plotting program: xcontour
 See the xcontour selfdoc for the remaining parameters.



 Credits:
```

```
CWP: Dave Hale and Zhiming Li (xwigb, etc.)
   Jack Cohen and John Stockwell (suxwigb, etc.)
      Delphi: Alexander Koek, added support for irregularly spaced traces
      Aarhus University: Morten W. Pedersen copied everything from the xwigb
                         source and just replaced all occurencies of the word
                         xwigb with xcountour ;-)

 Notes:
When the number of traces isn't known, we need to count
the traces for xcontour.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SUXGRAPH - X-windows GRAPH plot of a segy data set

suxgraph <stdin [optional parameters] | ...

Optional parameters:
(see xgraph selfdoc for optional parametes)

nplot= number of traces (ntr is an acceptable alias for nplot)

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set)

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Note that for seismic time domain data, the "fast dimension" is
time and the "slow dimension" is usually trace number or range.
Also note that "foreign" data tapes may have something unexpected
in the d2,f2 fields, use segyclean to clear these if you can afford
the processing time or use d2= f2= to over-ride the header values if
not.

See the xgraph selfdoc for the remaining parameters.

On NeXT:     suxgraph < infile [optional parameters]  | open

Credits:

CWP: Dave Hale and Zhiming Li (pswigp, etc.)
```

Jack Cohen and John Stockwell (supswigp, etc.)

Notes:
When the number of traces isn't known, we need to count
the traces for pswigp. You can make this value "known"
either by getparring nplot or by having the ntr field set
in the trace header. A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.

```
SUXIMAGE - X-windows IMAGE plot of a segy data set

suximage infile= [optional parameters] | ...  (direct I/O)
 or
suximage <stdin [optional parameters] | ... (sequential I/O)

Optional parameters:

infile=NULL SU data to be ploted, default stdin with sequential access
            if 'infile' provided, su data read by (fast) direct access

     with ftr,dtr and n2 suximage will pass a subset of data
     to the plotting program-ximage:
ftr=1       First trace to be plotted
dtr=1       Trace increment to be plotted
n2=tr.ntr   (Max) number of traces to be plotted (ntr is an alias for n2)
     Priority: first try to read from parameter line;
       if not provided, check trace header tr.ntr;
       if still not provided, figure it out using ftello

d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2 sampling interval in the slow dimension
  =1.0  (if not set or was set to 0)

key= key for annotating d2 (slow dimension)
 If annotation is not at proper increment, try
 setting d2; only first trace's key value is read

f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
  =1.0 for seismic     (if not set)
  =d2 for nonseismic     (if not set)

verbose=0             =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()
```

```
 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 See the ximage selfdoc for the remaining parameters.


 Credits:

CWP: Dave Hale and Zhiming Li (ximage, etc.)
   Jack Cohen and John Stockwell (suximage, etc.)
MTU: David Forel, June 2004, added key for annotating d2
      ConocoPhillips: Zhaobo Meng, Dec 2004, added direct I/O

 Notes:

      When provide ftr and dtr and infile, suximage can be used to plot
      multi-dimensional volumes efficiently.  For example, for a Offset-CDP
      dataset with 32 offsets, the command line
      suximage infile=volume3d.su ftr=1 dtr=32 ... &
      will display the zero-offset common offset data with ranrom access.
      It is highly recommend to use infile= to view large datasets, since
      using stdin only allows sequential access, which is very slow for
      large datasets.

When the number of traces isn't known, we need to count
the traces for ximage.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXMAX - X-windows graph of the MAX, min, or absolute max value on
each trace of a SEGY (SU) data set

   suxmax <stdin [optional parameters]

 Optional parameters:
 mode=max max value
  =min min value
  =abs absolute max value

 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information

 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to over-ride the header values if
 not.

 See the sumax selfdoc for additional parameter.
 See the xgraph selfdoc for the remaining parameters.
```

```
 Credits:

CWP: John Stockwell, based on Jack Cohen's SU JACKet

 Notes:
When the number of traces isn't known, we need to count
the traces for xgraph.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.
```

```
SUXMOVIE - X MOVIE plot of a 2D or 3D segy data set

suxmovie <stdin [optional parameters]

Optional parameters:

n1=tr.ns                number of samples per trace
ntr=tr.ntr            number of traces in the data set
n2=tr.shortpad or tr.ntr number of traces in inline direction
n3=ntr/n2            number of traces in crossline direction

d1=tr.d1 or tr.dt/10^6    sampling interval in the fast dimension
  =.004 for seismic  (if not set)
  =1.0 for nonseismic (if not set)

d2=tr.d2     sampling interval in the slow dimension
  =1.0  (if not set)

d3=1.0     sampling interval in the slowest dimension

f1=tr.f1 or 0.0       first sample in the z dimension
f2=tr.f2 or 1.0           first sample in the x dimension
f3=1.0

mode=0          0= x,z slice movie through y dimension (in line)
                1= y,z slice movie through x dimension (cross line)
                2= x,y slice movie through z dimension (time slice)

verbose=0              =1 to print some useful information

tmpdir=   if non-empty, use the value as a directory path
 prefix for storing temporary files; else if the
         the CWP_TMPDIR environment variable is set use
         its value for the path; else use tmpfile()

Notes:
For seismic data, the "fast dimension" is either time or
depth and the "slow dimension" is usually trace number.
The 3D data set is expected to have n3 sets of n2 traces representing
the horizontal coverage of n2*d2 in x  and n3*d3 in y direction.

The data is read to memory with and piped to xmovie with the
respective sampling parameters.
```

See the xmovie selfdoc for the remaining parameters and X functions.

```
 SUXPICKER - X-windows  WIGgle plot PICKER of a segy data set

 suxpicker <stdin [optional parameters] | ...

 Optional parameters:

 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)

 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic  (if not set)
   =1.0 for nonseismic (if not set)

 d2=tr.d2 sampling interval in the slow dimension
   =1.0  (if not set)

 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension

 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 verbose=0              =1 to print some useful information


 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type  sukeyword -o to see the complete list of SU keywords.
```

```
 See the xpicker selfdoc for the remaining parameters.


 Credits:

CWP: Dave Hale and Zhiming Li (xpicker, etc.)
   Jack Cohen and John Stockwell (suxpicker, etc.)

 Notes:
When the number of traces isn't known, we need to count
the traces for xpicker.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we do have to count the traces, we use the "tmpfile"
routine because on many machines it is implemented
as a memory area instead of a disk file.

If your system does make a disk file, consider altering
the code to remove the file on interrupt.  This could be
done either by trapping the interrupt with "signal"
or by using the "tmpnam" routine followed by an immediate
"remove" (aka "unlink" in old unix).

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
 SUXWIGB - X-windows Bit-mapped WIGgle plot of a segy data set
 This is a modified suxwigb that uses the depth or coordinate scaling
 when such values are used as keys.

 suxwigb <stdin [optional parameters] | ...

 Optional parameters:
 key=(keyword) if set, the values of x2 are set from header field
specified by keyword
 n2=tr.ntr or number of traces in the data set (ntr is an alias for n2)
 d1=tr.d1 or tr.dt/10^6 sampling interval in the fast dimension
   =.004 for seismic (if not set)
   =1.0 for nonseismic (if not set)
 d2=tr.d2 sampling interval in the slow dimension
   =1.0 (if not set)
 f1=tr.f1 or tr.delrt/10^3 or 0.0  first sample in the fast dimension
 f2=tr.f2 or tr.tracr or tr.tracl  first sample in the slow dimension
   =1.0 for seismic     (if not set)
   =d2 for nonseismic     (if not set)

 style=seismic  normal (axis 1 horizontal, axis 2 vertical) or
 vsp (same as normal with axis 2 reversed)
 Note: vsp requires use of a keyword
 verbose=0              =1 to print some useful information


 tmpdir=   if non-empty, use the value as a directory path
  prefix for storing temporary files; else if the
          the CWP_TMPDIR environment variable is set use
          its value for the path; else use tmpfile()

 Note that for seismic time domain data, the "fast dimension" is
 time and the "slow dimension" is usually trace number or range.
 Also note that "foreign" data tapes may have something unexpected
 in the d2,f2 fields, use segyclean to clear these if you can afford
 the processing time or use d2= f2= to override the header values if
 not.

 If key=keyword is set, then the values of x2 are taken from the header
 field represented by the keyword (for example key=offset, will show
 traces in true offset). This permit unequally spaced traces to be plotted.
 Type  sukeyword -o to see the complete list of SU keywords.
```

```
 This program is really just a wrapper for the plotting program: xwigb
 See the xwigb selfdoc for the remaining parameters.


 Credits:

CWP: Dave Hale and Zhiming Li (xwigb, etc.)
   Jack Cohen and John Stockwell (suxwigb, etc.)
Delphi: Alexander Koek, added support for irregularly spaced traces

Modified by Brian Zook, Southwest Research Institute, to honor
 scale factors, added vsp style

 Notes:
When the number of traces isn't known, we need to count
the traces for xwigb.  You can make this value "known"
either by getparring n2 or by having the ntr field set
in the trace header.  A getparred value takes precedence
over the value in the trace header.

When we must compute ntr, we don't allocate a 2-d array,
but just content ourselves with copying trace by trace from
the data "file" to the pipe into the plotting program.
Although we could use tr.data, we allocate a trace buffer
for code clarity.
```

```
SXPLOT - X Window plot a triangulated sloth function s(x1,x2)

sxplot <modelfile [optional parameters]

Optional Parameters:
edgecolor=cyan          color to draw fixed edges
tricolor=yellow         color to draw non-fixed edges of triangles
=none non-fixed edges of triangles are not shown
bclip=minimum sloth     sloth value corresponding to black
wclip=maximum sloth     sloth value corresponding to white
x1beg=x1min             value at which x1 axis begins
x1end=x1max             value at which x1 axis ends
x2beg=x2min             value at which x2 axis begins
x2end=x2max             value at which x2 axis ends
cmap=gray               gray, hue, or default colormaps may be specified

Optional resource parameters (defaults taken from resource database):
width=                  width in pixels of window
height=                 height in pixels of window
nTic1=                  number of tics per numbered tic on axis 1
grid1=                  grid lines on axis 1 - none, dot, dash, or solid
label1=                 label on axis 1
nTic2=                  number of tics per numbered tic on axis 2
grid2=                  grid lines on axis 2 - none, dot, dash, or solid
label2=                 label on axis 2
labelFont=              font name for axes labels
title=                  title of plot
titleFont=              font name for title
titleColor=             color for title
axesColor=              color for axes
gridColor=              color for grid lines
style=                  normal (axis 1 horizontal, axis 2 vertical) or
                       seismic (axis 1 vertical, axis 2 horizontal)



AUTHOR:  Dave Hale, Colorado School of Mines, 05/17/91
```

```
 GBBEAM - Gaussian beam synthetic seismograms for a sloth model

 gbbeam <rayends >syntraces xg= zg= [optional parameters]

 Required Parameters:
 xg=              x coordinates of receiver surface
 zg=              z coordinates of receiver surface

 Optional Parameters:
 ng=101           number of receivers (uniform distributed along surface)
 krecord=1        integer index of receiver surface (see notes below)
 ang=0.0          array of angles corresponding to amplitudes in amp
 amp=1.0          array of amplitudes corresponding to angles in ang
 bw=0             beamwidth at peak frequency
 nt=251           number of time samples
 dt=0.004         time sampling interval
 ft=0.0           first time sample
 reftrans=0       =1 complex refl/transm. coefficients considered
 prim             =1, only single-reflected rays are considered ",
                  =0, only direct hits are considered
 atten=0          =1 add noncausal attenuation
                  =2 add causal attenuation
 lscale=          if defined restricts range of extrapolation
 aperture=        maximum angle of receiver aperture
 fpeak=0.1/dt     peak frequency of ricker wavelet
 infofile         ASCII-file to store useful information
 NOTES:
 Only rays that terminate with index krecord will contribute to the
 synthetic seismograms at the receiver (xg,zg) locations.  The
 receiver locations are determined by cubic spline interpolation
 of the specified (xg,zg) coordinates.



 AUTHOR:  Dave Hale, Colorado School of Mines, 02/09/91
 MODIFIED:  Andreas Rueger, Colorado School of Mines, 08/18/93
Modifications include: 2.5-D amplitudes, computation of reflection/
transmission losses, attenuation,
timewindow, lscale, aperture, beam width, etc.
```

```
NORMRAY - dynamic ray tracing for normal incidence rays in a sloth model

   normray <modelfile >rayends [optional parameters]

Optional Parameters:
caustic  0: show all rays 1: show only caustic rays
nonsurface  0: show rays which reach surface 1: show all rays
surface  0: shot ray from subsurface 1: from surface
nrays   number of location to shoot rays
dangle   increment of ray angle for one location
nangle   number of rays shot from one location
ashift   shift first taking off angle
xs1           x of shooting location
zs1           z of shooting location
nangle=101     number of takeoff angles
fangle=-45     first takeoff angle (in degrees)
rayfile        file of ray x,z coordinates of ray-edge intersections
nxz=101        number of (x,z) in optional rayfile (see notes below)
wavefile       file of ray x,z coordinates uniformly sampled in time
nt=101         number of (x,z) in optional wavefile (see notes below)
infofile       ASCII-file to store useful information
fresnelfile    used if you want to plot the fresnel volumes.
               default is <fresnelfile.bin>
outparfile     contains parameters for the plotting software.
               default is <outpar>
krecord        if specified, only rays incident at interface with index
               krecord are displayed and stored
prim           =1, only single-reflected rays are plotted  ",
               =0, only direct hits are displayed
ffreq=-1       FresnelVolume frequency
refseq=1,0,0   index of reflector followed by sequence of reflection (1)
               transmission(0) or ray stops(-1).
               The default rayend is at the model boundary.
               NOTE:refseq must be defined for each reflector
NOTES:
The rayends file contains ray parameters for the locations at which
the rays terminate.

The rayfile is useful for making plots of ray paths.
nxz should be larger than twice the number of triangles intersected
by the rays.

The wavefile is useful for making plots of wavefronts.
```

```
 The time sampling interval in the wavefile is tmax/(nt-1),
 where tmax is the maximum time for all rays.

 The infofile is useful for collecting information along the
 individual rays. The fresnelfile contains data used to plot
 the Fresnel volumes. The outparfile stores information used
 for the plotting software.



 AUTHOR:  Dave Hale, Colorado School of Mines, 02/16/91
 MODIFIED:  Andreas Rueger, Colorado School of Mines, 08/12/93
Modifications include: functions writeFresnel, checkIfSourceIsOnEdge;
options refseq=, krecord=, prim=, infofile=;
computation of reflection/transmission losses, attenuation.
 MODIFIED: Boyi Ou, Colorado School of Mines, 4/14/95

 Notes:
 This code can shoot rays from specified interface by users, normally you
need to use gbmodel2 to generate interface parameters for this code, both
code have a parameter named nrays, it should be same. If you just want to
shoot rays from one specified location, you need to specify xs1,zs1,
otherwise, leave them alone. If you want to shoot rays from surface, you need
to define surface equal to 1. The rays from one location will be
approximately symetric with direction Normal_direction - ashift.(if nangle is
odd, it is symetric, even, almost symetric. The formula for the first take
off angle is: angle=normal_direction-nangle/2*dangle-ashift. If you only want to
see caustics, you specify caustic=1, if you want to see rays which does not
reach surface, you specify nonsurface=1.
/
```

```
TRI2UNI - convert a TRIangulated model to UNIformly sampled model

tri2uni <triangfile >uniformfile n2= n1= [optional parameters]

Required Parameters:
n1=                     number of samples in the first (fast) dimension
n2=                     number of samples in the second dimension

Optional Parameters:
d1=1.0                 sampling interval in first (fast) dimension
d2=1.0                 sampling interval in second dimension
f1=0.0                 first value in dimension 1 sampled
f2=0.0                 first value in dimension 2 sampled

Note:
The triangulated/uniformly-sampled quantity is assumed to be sloth=1/v^2



AUTHOR:  Dave Hale, Colorado School of Mines, 04/23/91
```

```
TRIMODEL - make a triangulated sloth (1/velocity^2) model

trimodel >modelfile [optional parameters]

Optional Parameters:
xmin=0.0                minimum horizontal coordinate (x)
xmax=1.0                maximum horizontal coordinate (x)
zmin=0.0                minimum vertical coordinate (z)
zmax=1.0                maximum vertical coordinate (z)
xedge=                  x coordinates of an edge
zedge=                  z coordinates of an edge
sedge=                  sloth along an edge
kedge=                  array of indices used to identify edges
normray                0:do not generate parameters 1: does it
normface               specify which interface to shoot rays
nrays                  number of locations to shoot rays
sfill=                  x, z, x0, z0, s00, dsdx, dsdz to fill a region
densfill=               x, z, dens to fill a region
qfill=                  x, z, Q-factor to fill a region
maxangle=5.0            maximum angle (deg) between adjacent edge segments

Notes:
More than set of xedge, zedge, and sedge parameters may be
specified, but the numbers of these parameters must be equal.

Within each set, vertices will be connected by fixed edges.

Edge indices in the k array are used to identify edges
specified by the x and z parameters.  The first k index
corresponds to the first set of x and z parameters, the
second k index corresponds to the second set, and so on.

After all vertices and their corresponding sloth values have
been inserted into the model, the optional sfill parameters
are used to fill closed regions bounded by fixed edges.
Let (x,z) denote any point inside a closed region.  Sloth inside
this region is determined by s(x,z) = s00+(x-x0)*dsdx+(z-z0)*dsdz.
The (x,z) component of the sfill parameter is used to identify a
closed region.
```

AUTHOR: Dave Hale, Colorado School of Mines, 02/12/91
MODIFIED: Andreas Rueger, Colorado School of Mines, 01/18/93
   Fill regions with attenuation Q-factors and density values.
MODIFIED: Craig Artley, Colorado School of Mines, 03/27/94
   Corrected bug in computing s00 in makeSlothForTri() function.
MODIFIED: Boyi Ou, Colorado School of Mines, 4/14/95
    Make code to generate interface parameters for shooting rays
    from specified interface

NOTE:
When you use normface to specify interface, the number of interface might not be the number of interface in the picture, for example, you build a one interface model, this interface is very long, so in the shell, you use three part of xedge,zedge,sedge to make this interface, so when you use normface to specify interface, this interface is just part of whole interface. If you want see the normal rays from entire interface, you need to maek normal ray picture few times, and merge them together.

```
 TRIRAY - dynamic RAY tracing for a TRIangulated sloth model

  triray <modelfile >rayends [optional parameters]

 Optional Parameters:
 xs=(max-min)/2 x coordinate of source (default is halfway across model)
 zs=min  z coordinate of source (default is at top of model)
 nangle=101 number of takeoff angles
 fangle=-45 first takeoff angle (in degrees)
 langle=45 last takeoff angle (in degrees)
 rayfile= file of ray x,z coordinates of ray-edge intersections
 nxz=101 number of (x,z) in optional rayfile (see notes below)
 wavefile= file of ray x,z coordinates uniformly sampled in time
 nt=101 number of (x,z) in optional wavefile (see notes below)
 infofile= ASCII-file to store useful information
 fresnelfile=  used if you want to plot the fresnel volumes.
default is <fresnelfile.bin>
 outparfile= contains parameters for the plotting software.
default is <outpar>
 krecord= if specified, only rays incident at interface with index
krecord are displayed and stored
 prim=    =1, only single-reflected rays are plotted
=0, only direct hits are displayed
 ffreq=-1 FresnelVolume frequency
 refseq=1,0,0  index of reflector followed by sequence of reflection (1)
transmission(0) or ray stops(-1).
The default rayend is at the model boundary.
NOTE:refseq must be defined for each reflector
 NOTES:
 The rayends file contains ray parameters for the locations at which
 the rays terminate.

 The rayfile is useful for making plots of ray paths.
 nxz should be larger than twice the number of triangles intersected
 by the rays.

 The wavefile is useful for making plots of wavefronts.
 The time sampling interval in the wavefile is tmax/(nt-1),
 where tmax is the maximum time for all rays.

 The infofile is useful for collecting information along the
 individual rays. The fresnelfile contains data used to plot
 the Fresnel volumes. The outparfile stores information used
```

```
 for the plotting software.


 AUTHOR:  Dave Hale, Colorado School of Mines, 02/16/91
 MODIFIED:  Andreas Rueger, Colorado School of Mines, 08/12/93
Modifications include: functions writeFresnel, checkIfSourceIsOnEdge;
options refseq=, krecord=, prim=, infofile=;
computation of reflection/transmission losses, attenuation.
```

```
TRISEIS - Gaussian beam synthetic seismograms for a sloth model

 triseis <modelfile >seisfile xs= zs= xg= zg= [optional parameters]

Required Parameters:
xs=             x coordinates of source surface
zs=             z coordinates of source surface
xg=             x coordinates of receiver surface
zg=             z coordinates of receiver surface

Optional Parameters:
ns=1            number of sources uniformly distributed along s surface
ds=             increment between source locations (see notes below)
fs=0.0          first source location (relative to start of s surface)
ng=101          number of receivers uniformly distributed along g surface
dg=             increment between receiver locations (see notes below)
fg=0.0          first receiver location (relative to start of g surface)
dgds=0.0        change in receiver location with respect to source location
krecord=1       integer index of receiver surface (see notes below)
kreflect=-1     integer index of reflecting surface (see notes below)
prim            =1, only single-reflected rays are considered  ",
                =0, only direct hits are considered
bw=0            beamwidth at peak frequency
nt=251          number of time samples
dt=0.004        time sampling interval
ft=0.0          first time sample
nangle=101      number of ray takeoff angles
fangle=-45      first ray takeoff angle (in degrees)
langle=45       last ray takeoff angle (in degrees)
reftrans=0      =1 complex refl/transm. coefficients considered
atten=0         =1 add noncausal attenuation
                =2 add causal attenuation
lscale=         if defined restricts range of extrapolation
fpeak=0.1/dt    peak frequency of ricker wavelet
aperture=       maximum angle of receiver aperture

NOTES:
Only rays that terminate with index krecord will contribute to the
synthetic seismograms at the receiver (xg,zg) locations.  The
source and receiver locations are determined by cubic spline
interpolation of the specified (xs,zs) and (xg,zg) coordinates.
The default source location increment (ds) is determined to span
the source surface defined by (xs,zs).  Likewise for dg.
```

```
 AUTHOR:  Dave Hale, Colorado School of Mines, 02/09/91
 MODIFIED:  Andreas Rueger, Colorado School of Mines, 08/18/93
Modifications include: 2.5-D amplitudes, correction for ref/transm,
timewindow, lscale, aperture, beam width, etc.
```

```
UNI2TRI - convert UNIformly sampled model to a TRIangulated model

uni2tri <slothfile >modelfile n2= n1= [optional parameters]

Required Parameters:
n1=                     number of samples in first (fast) dimension
n2=                     number of samples in second dimension

Optional Parameters:
d1=1.0                  sampling interval in dimension 1
d2=1.0                  sampling interval in dimension 2
f1=0.0                  first value in dimension 1
f2=0.0                  first value in dimesion 2
ifile=                  triangulated model file used as initial model
errmax=                 maximum sloth error (see notes below)
verbose=1               =0 for silence
                        =1 to report maximum error at each stage to stderr
                        =2 to also write the normalized error to efile
efile=emax.dat          filename for error file (for verbose=2)
mm=0 output every mm-th intermediate model (0 for none)
mfile=intmodel         intermediate models written to intmodel%d
method=3               =1 add 1 vertex at maximum error
                       =2 add vertex to every triangle that exceeds errmax
                       =3 method 2, but avoid closely spaced vertices
tol=10                 closeness criterion for (in samples)
sfill=                  x, z, x0, z0, s00, dsdx, dsdz to fill a region

Notes:
Triangles are constructed until the maximum error is
not greater than the user-specified errmax.  The default errmax
is 1% of the maximum value in the sampled input file.

After the uniform values have been triangulated, the optional sfill
parameters are used to fill closed regions bounded by fixed edges.
Let (x,z) denote any point inside a closed region.  Values inside
this region is determined by s(x,z) = s00+(x-x0)*dsdx+(z-z0)*dsdz.
The (x,z) component of the sfill parameter is used to identify a
closed region.

The uniformly sampled quantity is assumed to be sloth=1/v^2.
```

```
AUTHOR:  Craig Artley, Colorado School of Mines, 03/31/94
NOTE:  After a program outlined by Dave Hale, 12/27/90.
```

```
SPSPLOT - plot a triangulated sloth function s(x,z) via PostScript

spsplot <modelfile >postscriptfile [optional parameters]

Optional Parameters:
gedge=0.0             gray to draw fixed edges (in interval [0.0,1.0])
gtri=1.0              gray to draw non-fixed edges of triangles
gmin=0.0              min gray to shade triangles (in interval [0.0,1.0])
gmax=1.0              max gray to shade triangles (in interval [0.0,1.0])
sgmin=minimum s(x,z)  s(x,y) corresponding to gmin
sgmax=maximum s(x,z)  s(x,y) corresponding to gmax
xbox=1.5              offset in inches of left side of axes box
ybox=1.5              offset in inches of bottom side of axes box
wbox=6.0              width in inches of axes box
hbox=8.0              height in inches of axes box
xbeg=xmin             value at which x axis begins
xend=xmax             value at which x axis ends
dxnum=0.0             numbered tic interval on x axis (0.0 for automatic)
fxnum=xmin            first numbered tic on x axis (used if dxnum not 0.0)
nxtic=1               number of tics per numbered tic on x axis
gridx=none            grid lines on x axis - none, dot, dash, or solid
labelx=               label on x axis
zbeg=zmin             value at which z axis begins
zend=zmax             value at which z axis ends
dznum=0.0             numbered tic interval on z axis (0.0 for automatic)
fznum=zmin            first numbered tic on z axis (used if dynum not 0.0)
nztic=1               number of tics per numbered tic on z axis
gridz=none            grid lines on z axis - none, dot, dash, or solid
labelz=               label on z axis
labelfont=Helvetica   font name for axes labels
labelsize=12          font size for axes labels
title=                title of plot
titlefont=Helvetica-Bold  font name for title
titlesize=24          font size for title
titlecolor=black      color of title
axescolor=black       color of axes
gridcolor=black       color of grid
style=seismic         normal (z axis horizontal, x axis vertical) or
                      seismic (z axis vertical, x axis horizontal)

Note:  A value of gedge or gtri outside the interval [0.0,1.0]
results in that class of edge not being drawn.
```

```
AUTHOR:  Dave Hale, Colorado School of Mines, 10/18/90
MODIFIED: Craig Artley, Colorado School of Mines, 03/27/94
   Tweaks to improve PostScript header, add basic color support.

NOTE:  Have observed errors in output when compiled with optimization
   under NEXTSTEP 3.1.  Caveat Emptor.

Modified: Morten Wendell Pedersen, Aarhus University, 23/3-97
          Added ticwidth,axeswidth, gridwidth parameters
```

Shells:

```
------
 ARGV - give examples of dereferencing char **argv

 Usage: argv


------
 COPYRIGHT - insert CSM COPYRIGHT lines at top of files in working directory

 Usage: copyright file(s)


------
 CPALL , RCPALL - for local and remote directory tree/file transfer

 Usage: cpall sourcedir destinationdir
 Caveat: destinationdir must exist and be writeable by the user

 Usage: rcpall sourcedir remotemachine  destinationdir

 If user name is different on the remote machine, then second"
 entry is "remotemachine -l remoteusername"
 Caveats: rsh, copy, and write permissions required
         You must be on the source machine,
 destinationdir must exist and be writeable by the user.

 Notes: Both of these shell scripts use tar to do the transfer.


------
 CWPFIND - look for files with patterns in CWPROOT/src/cwp/lib

 Usage: cwpfind  pattern_fragment
        cwpfind -e exact_pattern


------
 Grep  - recursively call egrep in pwd

 Usage: Grep [-egrep_options] pattern

 Caution:  Do NOT redirect into file in pwd, either use something
like  >../Grep.out or perhaps pipe output into mail to yourself.
```

```
 Author: Jack, 7/95


------
 DIRTREE - show DIRectory TREE

 Usage: dirtree


------
 FILETYPE - list all files of given type

 Usage: filetype string_from_file_output

 Examples:
filetype text      - list printable files
       filetype stripped  - list unstripped files

------
 NEWCASE - Changes the case of all the filenames in a directory, dir

 Usage: newcase -l dir  change all filenames to lower case
                -u dir  change all filenames to upper case

 Notes: Useful for files downloaded from VAX.


------
 OVERWRITE - copy stdin to stdout after EOF

 This shell is called from the shell script:    replace


------
 PRECEDENCE - give table of C precedences from Kernighan and Ritchie

 Usage: precedence


------
 REPLACE - REPLACE string1 with string2  in files

 Usage:  replace string1 string2 files
```

```
------
 THIS_YEAR - print the current year

 Usage: this_year

 NOTES - useful for building dated filenames, etc.


------
 TIME_NOW - prints time in ZULU format with no spaces

 Usage: time_now

 Note: Useful for building dated filenames


------
 TODAYS_DATE - prints today's date in ZULU format with no spaces

 Usage: todays_date

 Note: Useful for building dated filenames


------
 USERNAMES - get list of all login names

 Usage: usernames


------
 VARLIST - list variables used in a Fortran program

 Usage: varlist file.f ...

 Output is in the file: vars.file


------
 WEEKDAY - prints today's WEEKDAY designation

 Usage: weekday

 Note: Useful for building dated filenames
```

```
------
 ZAP - kill processes by name

 Typical usages:
zap ximage
zap 'xmovie|xgraph'

 Zap accepts full pattern matching for the process names

 Caveat: zap assumes that the FIRST field produced by the
Unix "ps" command is the pid (process identifier) number.
If not, change the number in the awk print statement to
the appropriate field.

 Author: Jack, 6/95 -- after Kernighan and Pike's zap
```

```
------
 GENDOCS - generate complete list of selfdocs in latex form

 Usage: gendocs -o  output filename is:  selfdocs.tex


------
 STRIPTOTXT -  put files from $CWPROOT/src/doc/Stripped into a new
               directory in the form $CWPROOT/src/TXT/NAME.txt

 Usage: striptotxt

 Author: John Stockwell,  Sept 2001

------
 UPDATEDOCALL - put self-docs in ../doc/Stripped

 Usage: updatedocall

 Note: this shell uses updatedoc to update the  database used by
       suname and gendocs


------
 UPDATEDOC - put self-docs in ../doc/Stripped and ../doc/Headers

 Usage: updatedoc  path

 Notes:
 Paths include: cwp/main cwp/lib cwp/shell par/main par/lib par/shell
     xplot/main xplot/lib psplot/main psplot/lib psplot/shell
     Xtcwp/main Xtcwp/lib Sfio/main
      su/main/amplitudes su/main/attributes_parameter_estimation
      su/main/convolution_correlation /su/main/data_compression
      su/main/data_conversion su/main/datuming su/main/decon_shaping
      su/main/dip_moveout su/main/filters su/main/headers su/main/interp_extrap
      su/main/migration_inversion su/main/multicomponent su/main/noise
      su/main/operations su/main/picking su/main/stacking su/main/statics
      su/main/stretching_moveout_resamp su/main/supromax
      su/main/synthetics_waveforms_testpatterns su/main/tapering
      su/main/transforms su/main/velocity_analysis su/main/well_logs
      su/main/windowing_sorting_muting
     su/lib su/shell su/graphics/psplot
     su/graphics/xplot tri/main tri/lib xtri tri/graphics/psplot
```

```
    tetra/lib tetra/main
    comp/dct/lib comp/dct/main comp/dct/libutil comp/dwpt/1d/lib
    comp/dwpt/1d/main comp/dwpt/2d/lib comp/dwpt/2d/main

 Use: updatedocall to update full directory, use updatehead to
      to update the master header file.

 This shell builds the database used by  suname and gendocs

------
 UPDATEHEAD - update ../doc/Headers/Headers.all

 Usage: updatehead

 Notes:

 This file builds the database used by  suname
```

```
------
 LOOKPAR - show getpar lines in SU code with defines evaluated

 Usage: lookpar filename ...


------
 MAXDIFF - find absolute maximum difference in two segy data sets

 Usage: maxdiff file1 file2


------
 RECIP - sum opposing (reciprocal) offsets in cdp sorted data

 Usage: recip <stdin >stdout


------
 RMAXDIFF - find percentage maximum difference in two segy data sets

 Usage: rmaxdiff file1 file2


------
 SUAGC - perform agc on SU data

 Note: this is an interface to sugain for backward compatibility
 See selfdoc of:   sugain   for more information

------
 SUBAND - Trapezoid-like Sin squared tapered Bandpass filter via  SUFILTER

 Usage:   suband < stdin > stdout

 Note: this shell mimmics the old program SUBAND, supersceded by SUFILTER
 See selfdoc of:   sufilter   for more information

------
 SUDIFF, SUSUM, SUPROD, SUQUO, SUPTDIFF, SUPTSUM,
 SUPTPROD, SUPTQUO - difference, sum, product, quotient of two SU data
                     sets via suop2

 Usage:
```

```
 sudiff file1 file2 > stdout
 susum file1 file2 > stdout
 ...etc

 Note: uses   suop2  to perform the computation

------
 SUDOC - get DOC listing for code

 Usage: sudoc name

 Note: Use this shell script to get selfdoc information for
 codes labeled with and asterisk (*) or pound sign (#) in suname list

------
 SUENV - Instantaneous amplitude, frequency, and phase via: SUATTRIBUTES

 Usage:   suenv < stdin > stdout

 Note: this shell mimmics the old program SUENV, supersceded by SUATTRIBUTES
 See selfdoc of:   suattributes   for more information

------
 SUFIND - get info from self-docs

 Usage: sufind [-v -n] string

 sufind string    gives a brief synopsis
 sufind -v string  is a verbose hunt for relevant items
 sufind -n name_fragment      searches for command name


------
 SUFIND - get info from self-docs

 Usage: sufind [-v -n] string

 sufind string    gives a brief synopsis
 sufind -v string  is a verbose hunt for relevant items
 sufind -n name_fragment      searches for command name

 Author: CWP: Jack K. Cohen,  1992
 Modified by: CWP: S. Narahara, 04/11/1998.
```

```
------
 SUGENDOCS - generate complete list of selfdocs in latex form

 Usage: sugendocs -o  output filename is: selfdocs.tex
 Note: this shell simply calls    gendocs


------
 SUHELP - list the CWP/SU programs and shells

 Usage:   suhelp


------
 SUKEYWORD -- guide to SU keywords in segy.h

 Usage: sukeyword -o            to begin at the top of segy.h
        sukeyword [string]      to find [string]

 Note:  keyword=  occurs in many SU programs.

------
 SUNAME - get name line from self-docs

 Usage: suname [name]

 Note: dummy selfdocs have been included in all cwp and shell programs
       that don't have automatic selfdocs.

------
 UNGLITCH - zonk outliers in data

 Usage: unglitch < stdin

 Note: this shell just invokes:  sugain < stdin qclip=.99 > stdout
 See selfdoc of:   sugain   for further information
```

```
------
 MERGE2 - put 2 standard size PostScript figures on one page

 Usage: merge2 fig1 fig2

 Notes: Translation values are hard-coded numbers that work well for
standard size (8.5 x 11) figures.
 See selfdoc of:   psmerge for details


------
 MERGE4 - put 4 standard size PostScript plots on one page

 Usage: merge4 ulfig urfig llfig lrfig

 Note: Translation values are hard-coded numbers that work well for
standard size (8.5 x 11) figures.
 See selfdoc of:   psmerge   for further information
```

Libs:

```
BASIC - Basic C function interface to PostScript

beginps write PostScript prolog (including %%Pages comment)
endps write PostScript trailer (including %%Pages comment)
begineps write encapsulated PostScript prolog (no %%Pages comment)
endeps write encapsulated PostScript trailer (no %%Pages comment)
boundingbox set BoundingBox to llx lly urx ury
newpage print "%%%%Page: label ordinal" to stdout
showpage print "showpage" to stdout
gsave print "GS" to stdout
grestore print "GR" to stdout
newpath print "NP" to stdout
closepath print "CP" to stdout
clip print "clip" to stdout
translate print "tx ty TR" to stdout, tx,ty = translation in x,y
scale print "sx sy SC" to stdout, sx,sy = scaling in x,y
rotate print "angle RO" to stdout, angle = rotation angle
concat print "m[0] m[1] m[2] m[3] m[4] m[5] CAT" to stdout
setgray print "gray setgray" to stdout, gray is 0-255 gray level
setrgbcolor print "red green blue setrgbcolor" to stdout
red,green,blue = 0-255 red,green,blue levels
setcolor set color by name based on definition in color structure
setlinewidth print "width SLW" to stdout, width = desired line width
setlinejoin print "code setlinejoin"
setdash print "[ dash ] offset setdash" to stdout
dash = array defining dash, offset = dash offset
moveto print "x y M" to stdout,   move to x,y
rmoveto print "x y RM" to stdout,  move to x,y
lineto print "x y L" to stdout, draw a line to x,y
rlineto print "x y RL" to stdout, draw a line to x,y
arc print "x y r ang1 ang2 arc" to stdout, draw an arc
x,y = vertex  r = radius from ang1 to ang2
stroke print "S" to stdout
fill print "F" to stdout
show  print "str SH" to stdout, show a string
justshow justify and show a string
image write a sampled gray-scale image
rgbimage write sampled color (rgb) image
setfont execute findfont, scalefont, and setfont for specified font
 and size
fontbbox determine font bounding box for specified font and size
fontheight return maximum height for specified font and size
```

```
fontwidth return maximum width for specified font and size
fontcapheight return maximum capheight for specified font and size
fontxheight return maximum xheight for specified font and size
fontdescender return maximum descender for specified font and size
polyline draw a segmented line
markto draw a mark at specified location
rectclip set a rectangular clipping path
rectfill draw a filled rectangle
strokerect stroke a rectangle

Function Prototypes:
void beginps (void);
void endps (void);
void begineps (void);
void endeps (void);
void newpage (const char *label, int ordinal);
void boundingbox (int llx, int lly, int urx, int ury);
void showpage (void);
void gsave (void);
void grestore (void);
void newpath (void);
void closepath (void);
void clip(void);
void translate (float tx, float ty);
void scale (float sx, float sy);
void rotate (float angle);
void concat (float m[]);
void setgray (float gray);
void setrgbcolor (float red, float green, float blue);
void setcolor (const char *name);
void setlinewidth (float width);
void setlinejoin (int code);
void setdash (float dash[], int ndash, float offset);
void moveto (float x, float y);
void rmoveto (float x, float y);
void lineto (float x, float y);
void rlineto (float x, float y);
void arc (float x, float y, float r, float ang1, float ang2);
void stroke (void);
void fill (void);
void show (const char *str);
void justshow (float just, const char *str);
void image (int w, int h, int bps, float m[], unsigned char *samples);
```

```
void rgbimage (int w, int h, int bpc, float m[], unsigned char *samples);
void cymkimage (int w, int h, int bpc, float m[], unsigned char *samples);
void setfont (const char *fontname, float fontsize);
void fontbbox (const char *fontname, float fontsize, float bbox[]);
float fontheight (const char *fontname, float fontsize);
float fontwidth (const char *fontname, float fontsize);
float fontcapheight (const char *fontname, float fontsize);
float fontxheight (const char *fontname, float fontsize);
float fontdescender (const char *fontname, float fontsize);
float fontascender (const char *fontname, float fontsize);
void polyline (const float *x, const float *y, int n);
void markto (float x, float y, int index, float size);
void rectclip (float x, float y, float width, float height);
void rectfill (float x, float y, float width, float height);
void rectstroke (float x, float y, float width, float height);

justshow:
Input:
just justification factor
str string

image:
Input:
w width of image (in samples)
h height of image (in samples)
bps number of bits per sample
m array[6] containing image matrix
samples array[w*h] of sample values

rgbimage:
Input:
w width of image (in samples)
h height of image (in samples)
bpc number of bits per component
m array[6] containing image matrix
samples array[3*w*h] of sample values

cymkimage:
Input:
w width of image (in samples)
h height of image (in samples)
bpc number of bits per component
m array[6] containing image matrix
```

```
samples array[4*w*h] of sample values


polyline:
Input:
x array[n] of x-coordinates
y array[n] of y-coordinates
n number of points

markto:
Input:
x x-coordinate of mark
y y-coordinate of mark
index type of mark to draw
size size of mark

rectclip:
Input:
x x-coordinate of clipping path origin
y y-coordinate of clipping path origin
width width of clipping path
height height of clipping path

rectfill:
Input:
x x-coordinate of rectangle origin
y y-coordinate of rectangle origin
width width of rectangle
height height of rectangle

strokerect:
Input:
x x-coordinate of rectangle origin
y y-coordinate of rectangle origin
width width of rectangle
height height of rectangle

Notes:
The majority of these routines are self explanatory. They are just
C wrappers that echo PostScript graphics commands.

justshow:
The justification factor positions the string relative to the current point.
```

```
just" may assume any value, but the common uses are:
-1.0 right-justify the string
-0.5 center the string on the current point
 0.0 left-justify the string (like using "show")

image:
Level 1 PostScript implementations support 1, 2, 4, and 8 bits per
sample.  Level 2 adds support for 12 bits per sample.
Samples are hex-encoded, and output lines are limited to 78 characters.

rgbimage:
In general, Level 1 PostScript implementations do not support rgbimage.
Level 2 supports 1, 2, 4, 8, and 12 bits per color component.  The
samples array should contain three color components (in R,G,B... order)
for each sample value.  Samples are hex-encoded, and output lines are
limited to 78 characters.

polyline:
The path is stroked every 200 points.
References:

Author:  Dave Hale, Colorado School of Mines, 1989
with modifications by Craig Artley, Colorado School of Mines, 1991, and
additions by Dave Hale, Advance Geophysical, 1992.
```

```
PSAXESBOX3 -  Functions draw an axes box via PostScript, estimate bounding box
               these are versions of psAxesBox and psAxesBox3 for psmovie.

psAxesBox3 draw an axes box via PostScript
psAxesBBox3 estimate bounding box for an axes box drawn via psAxesBox3

Function Prototypes:
void psAxesBox3(
float x, float y, float width, float height,
float x1Beg, float x1End, float p1Beg, float p1End,
float d1Num, float f1Num, int n1Tic, int grid1, char *label1,
float x2Beg, float x2End, float p2Beg, float p2End,
float d2Num, float f2Num, int n2Tic, int grid2, char *label2,
char *labelFont, float labelSize,
char *title, char *titleFont, float titleSize,
int style, char *title2);
void psAxesBBox3(
float x, float y, float width, float height,
char *labelFont, float labelSize,
char *titleFont, float titleSize,
int style, int bbox[]);

Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
x1Beg   axis value at beginning of axis 1
x1End axis value at end of axis 1
p1Beg   pad value at beginning of axis 1
p1End pad value at end of axis 1
d1Num numbered tic increment for axis 1 (0.0 for automatic)
f1Num first numbered tic for axis 1
n1Tic number of horizontal tics per numbered tic for axis 1
grid1 grid code for axis 1:  NONE, DOT, DASH, or SOLID
label1 label for axis 1
x2Beg   axis value at beginning of axis 2
x2End axis value at end of axis 2
p2Beg   pad value at beginning of axis 2
p2End pad value at end of axis 2
d2Num vertical numbered tic increment (0.0 for automatic)
f2Num first numbered vertical tic
n2Tic number of vertical tics per numbered tic
```

```
grid2 grid code for vertical axis:  NONE, DOT, DASH, or SOLID
label2 vertical axis label
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
title axes box title
titleFont name of font to use for title
titleSize size of font to use for title
style NORMAL (axis 1 on bottom, axis 2 on left)
  SEISMIC (axis 1 on left, axis 2 on top)
title2 second title

psAxesBBox3:
Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
titleFont name of font to use for title
titleSize size of font to use for title
style NORMAL (axis 1 on bottom, axis 2 on left)
  SEISMIC (axis 1 on left, axis 2 on top)
Output:
bbox bounding box (bbox[0:3] = llx, lly, ulx, uly)

Notes:
psAxesBox3:
psAxesBox will determine the numbered tic increment and first
numbered tic automatically, if the specified increment is zero.

Pad values must be specified in the same units as the corresponding
axes values.  These pads are useful when the contents of the axes box
requires more space than implied by the axes values.  For example,
the first and last seismic wiggle traces plotted inside an axes box
will typically extend beyond the axes values corresponding to the
first and last traces.  However, all tics will lie with the limits
specified in the axes values (x1Beg, x1End, x2Beg, x2End).

psAxesBBox3:
psAxesBBox uses font sizes to estimate the bounding box for
an axes box drawn with psAxesBox.  To be on the safe side,
psAxesBBox overestimates.
```

```
psAxesBBox assumes that the axes labels and titles do not extend
beyond the corresponding edges of the axes box.

References:
(see references in basic.c)
Author:  Dave Hale, Colorado School of Mines, 06/27/89
 modified by Zhiming Li, CSM, 7/1/90
```

```
PSAXESBOX - Functions to draw PostScript axes and estimate bounding box

psAxesBox Draw an axes box via PostScript
psAxesBBox estimate bounding box for an axes box drawn via psAxesBox

Function Prototypes:
void psAxesBox(
float x, float y, float width, float height,
float x1Beg, float x1End, float p1Beg, float p1End,
float d1Num, float f1Num, int n1Tic, int grid1, char *label1,
float x2Beg, float x2End, float p2Beg, float p2End,
float d2Num, float f2Num, int n2Tic, int grid2, char *label2,
char *labelFont, float labelSize,
char *title, char *titleFont, float titleSize,
char *titleColor, char *axesColor, char *gridColor,
float ticwidth, float axeswidth, float gridwidth,
int style);

void psAxesBBox(
float x, float y, float width, float height,
char *labelFont, float labelSize,
char *titleFont, float titleSize,
int style, int bbox[]);

psAxesBox:
Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
x1Beg   axis value at beginning of axis 1
x1End axis value at end of axis 1
p1Beg   pad value at beginning of axis 1
p1End pad value at end of axis 1
d1Num numbered tic increment for axis 1 (0.0 for automatic)
f1Num first numbered tic for axis 1
n1Tic number of horizontal tics per numbered tic for axis 1
grid1 grid code for axis 1:  NONE, DOT, DASH, or SOLID
label1 label for axis 1
x2Beg   axis value at beginning of axis 2
x2End axis value at end of axis 2
p2Beg   pad value at beginning of axis 2
p2End pad value at end of axis 2
```

```
d2Num vertical numbered tic increment (0.0 for automatic)
f2Num first numbered vertical tic
n2Tic number of vertical tics per numbered tic
grid2 grid code for vertical axis:  NONE, DOT, DASH, or SOLID
label2 vertical axis label
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
title axes box title
titleFont name of font to use for title
titleSize size of font to use for title
titleColor color to use for title
axesColor color to use for axes and axes labels
gridColor color to use for grid lines
axeswidth   width (in points) of axes
ticwidth    width (in points) of tic marks
gridwidth   width (in points) of grid lines
style NORMAL (axis 1 on bottom, axis 2 on left)
SEISMIC (axis 1 on left, axis 2 on top)

psAxesBBox:
Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
titleFont name of font to use for title
titleSize size of font to use for title
style NORMAL (axis 1 on bottom, axis 2 on left)
  SEISMIC (axis 1 on left, axis 2 on top)
Output:
bbox bounding box (bbox[0:3] = llx, lly, ulx, uly)

Notes:
psAxesBox:
psAxesBox will determine the numbered tic increment and first numbered
tic automatically, if the specified increment is zero.  Axis numbering
is in scientific notation, if necessary and is plotted to four
significant digits.

Pad values must be specified in the same units as the corresponding
axes values.  These pads are useful when the contents of the axes box
```

requires more space than implied by the axes values. For example, the first and last seismic wiggle traces plotted inside an axes box will typically extend beyond the axes values corresponding to the first and last traces. However, all tics will lie with the limits specified in the axes values (x1Beg, x1End, x2Beg, x2End).

psAxesBBox:
psAxesBBox uses font sizes to estimate the bounding box for
an axes box drawn with psAxesBox. To be on the safe side,
psAxesBBox overestimates.

psAxesBBox assumes that the axes labels and titles do not extend
beyond the corresponding edges of the axes box.

References:
(see References for basic.c)
Author: Dave Hale, Colorado School of Mines, 06/27/89
Modified: Ken Larner, Colorado School of Mines, 08/30/90
Modified: Dave Hale, Advance Geophysical, 10/18/92
Added color parameters for title, axes, and grid.
Modified: Morten Wendell Pedersen, Aarhus University, 23/3-97
Added ticwidth,axeswidth, gridwidth parameters

```
PSCAXESBOX - Draw an axes box for cube via PostScript

psCubeAxesBox Draw an axes box for cube via PostScript

Function Prototype:
void psCubeAxesBox(
float x, float y, float size1, float size2, float size3, float angle,
float x1Beg, float x1End, float p1Beg, float p1End,
float d1Num, float f1Num, int n1Tic, int grid1, char *label1,
float x2Beg, float x2End, float p2Beg, float p2End,
float d2Num, float f2Num, int n2Tic, int grid2, char *label2,
float x3Beg, float x3End, float p3Beg, float p3End,
float d3Num, float f3Num, int n3Tic, int grid3, char *label3,
char *labelFont, float labelSize,
char *title, char *titleFont, float titleSize,
char *titleColor, char *axesColor, char *gridColor);

Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
size1 size of 1st dimension of the input cube
size2 size of 2nd dimension of the input cube
size3 size of 3rd dimension of the input cube
angle projection angle of the cube
x1Beg   axis value at beginning of axis 1
x1End axis value at end of axis 1
p1Beg   pad value at beginning of axis 1
p1End pad value at end of axis 1
d1Num numbered tic increment for axis 1 (0.0 for automatic)
f1Num first numbered tic for axis 1
n1Tic number of tics for axis 1
grid1 grid code for axis 1:  NONE, DOT, DASH, or SOLID
label1 label for axis 1
x2Beg   axis value at beginning of axis 2
x2End axis value at end of axis 2
p2Beg   pad value at beginning of axis 2
p2End pad value at end of axis 2
d2Num numbered tic increment for axis 2 (0.0 for automatic)
f2Num first numbered tic for axis 2
n2Tic number of tics for axis 2
grid2 grid code for axis 2:  NONE, DOT, DASH, or SOLID
label2 label for axis 2
x3Beg   axis value at beginning of axis 3
```

```
x3End axis value at end of axis 3
p3Beg   pad value at beginning of axis 3
p3End pad value at end of axis 3
d3Num numbered tic increment for axis 3 (0.0 for automatic)
f3Num first numbered tic for axis 3
n3Tic number of tics for axis 3
grid3 grid code for axis 3:  NONE, DOT, DASH, or SOLID
label3 label for axis 3
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
title axes box title
titleFont name of font to use for title
titleSize size of font to use for title
titleColor color to use for title
axesColor color to use for axes and axes labels
gridColor color to use for grid lines
Authors:   Zhiming Li & Dave Hale,  Colorado School of Mines, 6/90
Modified: Craig Artley, Colorado School of Mines, 3/12/93
Changed name to psCubeAxesBox (from psAxes3), fixed minor bugs.
Modified: Craig Artley, Colorado School of Mines, 12/16/93
Added color parameters for title, axes, and grid.
```

PSCONTOUR - draw contour of a two-dimensional array via PostScript

psContour draw contour of a two-dimensional array via PostScript

Function Prototype:
void psContour (float c, int nx, float x[], int ny, float y[], float z[],
float lcs, char *lcf, char *lcc, float *w, int nplaces);

Input:
c contour value
nx number of x-coordinates
x array of x-coordinates (see notes below)
ny number of y-coordinates
y array of y-coordinates (see notes below)
lcs font size of contour label
lcf font name of contour label
lcc color of contour label
LSB flag arrays (see Notes):
z array of nx*ny z(x,y) values (see notes below)
w array of nx*ny z(x,y) values (see notes below)

Notes:
The two-dimensional array z is actually passed as a one-dimensional
array containing nx*ny values, stored with nx fast and ny slow.

The x and y arrays define a grid that is not necessarily
uniformly-sampled. Linear interpolation of z values on the
grid defined by the x and y arrays is used to determine z values
between the gridpoints.

The two least significant bits of z are used to mark intersections
of the contour with the x,y grid; therefore, the z values will
almost always be altered (slightly) by contour.

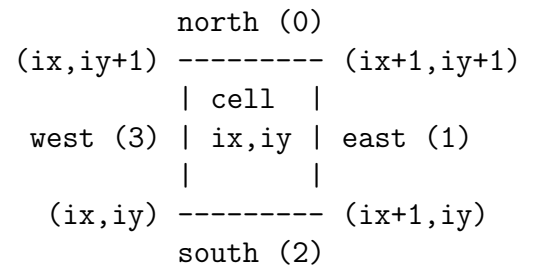
pscontour isolates the use of PostScript to four internal functions:

void coninit(void) - called before any contour drawing
void conmove(float x, float y) - moves current position to x,y
void condraw(float x, float y) - draws from current position to x,y
void condone(void) - called when contour drawing is done

These functions can usually be replaced with equivalent functions in
other graphics environments.

The w array is used to restrict the range of contour labeling that
occurs only if w<1.

As suggested in the reference, the following scheme is used to refer
to a cell of the two-dimensional array z:



Reference:
Cottafava, G. and Le Moli, G., 1969, Automatic contour map:
Commuincations of the ACM, v. 12, n. 7, July, 1969.

Author: Dave Hale, Colorado School of Mines, 06/28/89
contour labeling added by: Zhenyue Liu, August 1993

```
PSDRAWCURVE - Functions to draw a curve from a set of points

psDrawCurve Draw a curve from a set of points via PostScript
Function Prototypes:
void psDrawCurve(
float x, float y, float width, float height,
float x1Beg, float x1End, float p1Beg, float p1End,
float x2Beg, float x2End, float p2Beg, float p2End,
float *x1curve, float *x2curve, int ncurve,
char *curveColor, float curvewidth, int curvedash, int style);
psDrawCurve:
Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
x1Beg   axis value at beginning of axis 1
x1End axis value at end of axis 1
p1Beg   pad value at beginning of axis 1
p1End pad value at end of axis 1
x2Beg   axis value at beginning of axis 2
x2End axis value at end of axis 2
p2Beg   pad value at beginning of axis 2
p2End pad value at end of axis 2
x1curve vector of x1 coordinates for points along curve
x2curve vector of x2 coordinates for points along curve
ncurve number of points along curve
curveColor color to use for curve
curvewidth width (in points) of curve
style NORMAL (axis 1 on bottom, axis 2 on left)
SEISMIC (axis 1 on left, axis 2 on top)

Author: Brian Macy, Phillips Petroleum Co., 11/20/98
(Adapted after Dave Hale and other's psAxesBox routine)
```

```
PSLEGENDBOX - Functions to draw PostScript axes and estimate bounding box

psLegendBox Draw an legend box via PostScript
psLegendBBox estimate bounding box for an legend box drawn via psLegendBox

Function Prototypes:
void psLegendBox(
float x, float y, float width, float height,
float x1Beg, float x1End, float p1Beg, float p1End,
float d1Num, float f1Num, int n1Tic, int grid1, char *label1,
char *labelFont, float labelSize,
char *axesColor, char *gridColor,
int style);

void psLegendBBox(
float x, float y, float width, float height,
char *labelFont, float labelSize,
int style, int bbox[]);


psLegendBox:
Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
x1Beg   axis value at beginning of axis 1
x1End axis value at end of axis 1
p1Beg   pad value at beginning of axis 1
p1End pad value at end of axis 1
d1Num numbered tic increment for axis 1 (0.0 for automatic)
f1Num first numbered tic for axis 1
n1Tic number of horizontal tics per numbered tic for axis 1
grid1 grid code for axis 1:  NONE, DOT, DASH, or SOLID
label1 label for axis 1
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
axesColor color to use for axes and axes labels
gridColor color to use for grid lines
style VERTLEFT (Vertical, axis label on left side)
VERTRIGHT (Vertical, axis label on right side)
HORIBOTTOM (Horizontal, axis label on bottom)
```

```
psLegendBBox:
Input:
x x coordinate of lower left corner of box
y y coordinate of lower left corner of box
width width of box
height height of box
labelFont name of font to use for axes labels
labelSize size of font to use for axes labels
style VERTLEFT (Vertical, axis label on left side)
VERTRIGHT (Vertical, axis label on right side)
HORIBOTTOM (Horizontal, axis label on bottom)
Output:
bbox bounding box (bbox[0:3] = llx, lly, ulx, uly)

Notes:
psLegendBox:
psLegendBox will determine the numbered tic increment and first numbered
tic automatically, if the specified increment is zero.  Axis numbering
is in scientific notation, if necessary and is plotted to four
significant digits.

Pad values must be specified in the same units as the corresponding
Legend values.  These pads are useful when the contents of the Legend box
requires more space than implied by the Legend values.  For example, the
first and last seismic wiggle traces plotted inside an Legend box will
typically extend beyond the Legend values corresponding to the first and
last traces.  However, all tics will lie with the limits specified in
the Legend values (x1Beg, x1End, x2Beg, x2End).

psLegendBBox:
psLegendBBox uses font sizes to estimate the bounding box for
an Legend box drawn with psLegendBox.  To be on the safe side,
psLegendBBox overestimates.

psLegendBBox assumes that the Legend labels and titles do not extend
beyond the corresponding edges of the Legend box.

References:
(see References for basic.c)
Author:   Dave Hale,  Colorado School of Mines, 06/27/89
Modified: Ken Larner, Colorado School of Mines, 08/30/90
Modified: Dave Hale, Advance Geophysical, 10/18/92
Added color parameters for title, axes, and grid.
```

```
Modified: Torsten Schoenfelder, Koeln, Germany, 07/06/97
        Display a legend for ps file, move axis from left to right
Modified: Torsten Schoenfelder, Koeln, Germany, 10/02/98
        Corrected width of bbox to include legend title
```

```
PSWIGGLE - draw wiggle-trace with (optional) area-fill via PostScript

psWiggle  draw wiggle-trace with (optional) area-fill via PostScript

Function Prototype:
void psWiggle (int n, float z[], float zmin, float zmax, float zbase,
float yzmin, float yzmax, float xfirst, float xlast, int fill);

Inputs:
n number of samples to draw
z array to draw
zmin z values below zmin will be clipped
zmax z values above zmax will be clipped
zbase z values between zbase and either zmin or zmax will be filled
yzmin y-coordinate corresponding to zmin
yzmax y-coordinate corresponding to zmax
xfirst x-coordinate corresponding to z[0]
xlast x-coordinate corresponding to z[n-1]
fill = 0 for no fill
  > 0 for fill between zbase and zmax
  < 0 for fill between zbase and zmin
+2 for fill solid between zbase and zmax grey between zbase and zmin
-2 for fill solid between zbase and zmin grey between zbase and zmax
SHADING: 2<= abs(fill) <=5   abs(fill)=2 light grey  abs(fill)=5 black

NOTES:
psWiggle reduces PostScript output by eliminating linetos when
z values are essentially constant.  The tolerance for detecting
constant" z values is ZEPS*(zmax-zmin), where ZEPS is a fraction
defined below.

A more complete optimization would eliminate all connected line
segments that are essentially colinear.

psWiggle breaks up the wiggle into segments that can be drawn
without exceeding the PostScript pathlimit.

Author:  Dave Hale, Colorado School of Mines, 07/03/89
Modified:  Craig Artley, Colorado School of Mines, 04/13/92
           Corrected dead trace bug.  Now the last point of each segment
           is guaranteed to be drawn.
MODIFIED: Paul Michaels, Boise State University, 29 December 2000
           added fill=+/-2 option of solid/grey color scheme
```

```
AXESBOX - Functions to draw axes in X-windows graphics

xDrawAxesBox draw a labeled axes box
xSizeAxesBox determine optimal origin and size for a labeled axes box

Function Prototypes:
void xDrawAxesBox (Display *dpy, Window win,
int x, int y, int width, int height,
float x1beg, float x1end, float p1beg, float p1end,
float d1num, float f1num, int n1tic, int grid1, char *label1,
float x2beg, float x2end, float p2beg, float p2end,
float d2num, float f2num, int n2tic, int grid2, char *label2,
char *labelfont, char *title, char *titlefont,
char *axescolor, char *titlecolor, char *gridcolor,
int style);
void xSizeAxesBox (Display *dpy, Window win,
char *labelfont, char *titlefont, int style,
int *x, int *y, int *width, int *height);

xDrawAxesBox:
Input:
dpy display pointer
win window
x x coordinate of upper left corner of box
y y coordinate of upper left corner of box
width width of box
height height of box
x1beg axis value at beginning of axis 1
x1end axis value at end of axis 1
p1beg pad value at beginning of axis 1
p1end pad value at end of axis 1
d1num numbered tic increment for axis 1 (0.0 for automatic)
f1num first numbered tic for axis 1
n1tic number of tics per numbered tic for axis 1
grid1 grid code for axis 1:  NONE, DOT, DASH, or SOLID
label1 label for axis 1
x2beg axis value at beginning of axis 2
x2end axis value at end of axis 2
p2beg pad value at beginning of axis 2
p2end pad value at end of axis 2
d2num numbered tic increment for axis 2 (0.0 for automatic)
f2num first numbered tic for axis 2
n2tic number of tics per numbered tic for axis 2
```

```
grid2 grid code for axis 2:  NONE, DOT, DASH, or SOLID
label2 label for axis 2
labelfont name of font to use for axes labels
title axes box title
titlefont name of font to use for title
axescolor name of color to use for axes
titlecolor name of color to use for title
gridcolor name of color to use for grid
int style NORMAL (axis 1 on bottom, axis 2 on left)
SEISMIC (axis 1 on left, axis 2 on top)

xSizeAxesBox:
Input:
dpy display pointer
win window
labelfont name of font to use for axes labels
titlefont name of font to use for title
int style NORMAL (axis 1 on bottom, axis 2 on left)
SEISMIC (axis 1 on left, axis 2 on top)

Output:
x x coordinate of upper left corner of box
y y coordinate of upper left corner of box
width width of box
height height of box
{
XFontStruct *fa,*ft;
Notes:
xDrawAxesBox:
will determine the numbered tic incremenet and first
numbered tic automatically, if the specified increment is zero.

Pad values must be specified in the same units as the corresponding
axes values.  These pads are useful when the contents of the axes box
requires more space than implied by the axes values.  For example,
the first and last seismic wiggle traces plotted inside an axes box
will typically extend beyond the axes values corresponding to the
first and last traces.  However, all tics will lie within the limits
specified in the axes values (x1beg, x1end, x2beg, x2end).

xSizeAxesBox:
is intended to be used prior to xDrawAxesBox.
```

```
An "optimal" axes box is one that more or less fills the window,
with little wasted space around the edges of the window.

Author: Dave Hale, Colorado School of Mines, 01/27/90
```

```
COLORMAP - Functions to manipulate X colormaps:

xCreateRGBDefaultMap create XA_RGB_DEFAULT_MAP property of root window if
it does not already exist
xGetFirstPixel return first pixel in range of contiguous pixels in
XA_RGB_DEFAULT_MAP
xGetLastPixel return last pixel in range of contiguous pixels in
XA_RGB_DEFAULT_MAP
xCreateHSVColormap create a 2 ramp colormap (HSV - Model)
xCreateRGBColormap create a 2 ramp colormap (RGB - Model)

Function Prototypes:
Status xCreateRGBDefaultMap (Display *dpy, XStandardColormap *scmap);
unsigned long xGetFirstPixel (Display *dpy);
unsigned long xGetLastPixel (Display *dpy);
Colormap xCreateRGBColormap (Display *dpy, Window win,
char *str_cmap, int verbose)
Colormap xCreateHSVColormap (Display *dpy, Window win,
char *str_cmap, int verbose)

xCreateRGBDefaultMap:
Input:
dpy display

Output:
scmap the standard colormap structure

xGetFirstPixel, xGetLastPixel:
Input:
dpy display

Notes:
PROBLEM
-------

Most mid-range display devices today support what X calls
the "PseudoColor visual".  Typically, only 256 colors (or gray
levels) may be displayed simultaneously.  Although these 256 colors
may be chosen from a much larger (4096 or more) set of available
colors, only 256 colors can appear on a display at one time.

These 256 colors are indexed by pixel values in a table called
the colormap.  Each window can have its own colormap, but only
```

one colormap can be installed in the display hardware at a time. (Again, only 256 colors may be displayed at one time.) The window manager is responsible for installing a window's colormap when that window becomes the key window.

Many of the applications we are likely to write require a large, contiguous range of pixels (entries in the colormap). In this range, we must be able to:
(1) given a color (or gray), determine the corresponding pixel.
(2) given a pixel, determine the corresponding color (or gray).
An example would be an imaging application that uses a gray scale to display images in shades of gray between black and white. Such applications are also likely to require a few additional colors for drawing axes, text, etc.

The problem is to coordinate the use of the limited number of 256 simultaneous colors so that windows for different applications appear reasonable, even when their particular colormaps are not installed in the display hardware. For example, we might expect an analog xclock's hands to be visible even when xclock's window is not the key window, when its colormap is not installed.

We should ensure that the range of contiguous pixels used by one application (perhaps for imaging) does not conflict with the pixels used by other applications to draw text, clock hands, etc.

SOLUTION

Applications that do not require special colormaps should simply use the default colormap inherited from the root window when new top-level windows are created.

Applications that do require a special colormap MUST create their own colormap. They must not assume that space will be available in the default colormap for a contiguous range of read/write pixels, because the server or window manager may have already allocated these pixels as read-only. Even if sufficient pixels are available in the default colormap, they should not be allocated by a single application. The default colormap should be used only for windows requiring a limited number of typical colors, such as red, yellow, etc.

Applications that require a contiguous range of read/write pixels should allocate these pixels in their window's private colormaps. They should determine which contiguous pixels to allocate from parameters in the standard colormap XA_RGB_DEFAULT_MAP. In particular, the first pixel in the range of contiguous pixels should be
base_pixel
and the last pixel in the range should be
base_pixel+red_max*red_mult+green_max*green_mult+blue_max*blue_mult,
where base_pixel, red_max, etc. are members in the XStandardColormap structure. On an 8-bit display, this range will typically provide 216 contiguous pixels, which may be set to a gray scale, color scale, or whatever. This leaves 40 colors for drawing text, axes, etc.

If the XA_RGB_DEFAULT_MAP does not exist, it should be created to consist of various colors composed of an equal number of reds, greens, and blues. For example, if 216 colors are to be allocated, then red_max=green_max=blue_max=5, red_mult=36, green_mult=6, and blue_mult=1. Because of the difficulty in forcing a particular pixel to correspond to a particular color in read-only color cells, these 216 colors will likely be read/write color cells unless created by the X server. In any case, these 216 colors should not be modified by any application. In creating custom colormaps, the only use of XA_RGB_DEFAULT_MAP should be in determining which 216 pixels to allocate for contiguous pixels.

In creating a custom colormap for a window, the application should initialize this colormap to the colors already contained in the window's colormap, which was inherited initially from its parent. This will ensure that typical colors already allocated by other applications will be consistent with pixels used by the application requiring the custom colormap. Ideally, windows might have different colormaps, but the only differences would be in the range of contiguous colors used for imaging, rendering, etc. Ideally, the pixels corresponding to colors used to draw text, axes, etc. would be consistent for all windows.

Unfortunately, it is impractical to maintain complete consistency among various private colormaps. For example, suppose a custom colormap is created for a window before other applications have had the opportunity to allocate their colors from the default colormap. Then, when the window with the custom colormap becomes the key window, the windows of the other applications may be displayed with false colors, since the colormap of the key window

```
may not contain the true colors.  The colors used by the other
applications did not exist when the custom colormap was created.
One solution to this problem might be to initially allocate a
set of "common" colors in the default colormap before launching
any applications.  This will increase the likelihood that typical
colors will be consistent among various colormaps.

Functions are provided below to
(1) create the standard colormap XA_RGB_DEFAULT_MAP, if it does not exist,
(2) determine the first and last pixels in the contiguous range of pixels,
(3) create some common private colormaps

xCreateRGBDefaultMap:
This function returns 0 if the XA_RGB_DEFAULT_MAP property does not exist
and cannot be created.  At least 8 contiguous color cells must be free
in the default colormap to create the XA_RGB_DEFAULT_MAP.  If created, the
red_max, green_max, and blue_max values returned in scmap will be equal.

xGetFirstPixel, xGetLastPixel:
If it does not already exist, XA_RGB_DEFAULT_MAP will be created.
If XA_RGB_DEFAULT_MAP does not exist and cannot be created, then
this function returns 0.

xCreateRGBColormap, xCreateHSVColormap:
The returned colormap is only created; the window's colormap attribute
is not changed, and the colormap is not installed by this function.
The returned colormap is a copy of the window's current colormap, but
with an RGB color scale allocated in the range of contiguous cells
determined by XA_RGB_DEFAULT_MAP.  If it does not already exist,
XA_RGB_DEFAULT_MAP will be created.

Author:  Dave Hale, Colorado School of Mines, 09/30/90
```

```
DRAWCURVE - Functions to draw a curve from a set of points

xDrawCurve draw a curve from a set of points
Function Prototypes:
void xDrawCurve(Display *dpy, Window win,
int x, int y, int width, int height,
float x1beg, float x1end, float p1beg, float p1end,
float x2beg, float x2end, float p2beg, float p2end,
float *x1curve, float *x2curve, int ncurve,
char *curvecolor, int style);
xDrawCurve:
Input:
dpy display pointer
win window
x x coordinate of upper left corner of box
y y coordinate of upper left corner of box
width width of box
height height of box
x1beg axis value at beginning of axis 1
x1end axis value at end of axis 1
p1beg pad value at beginning of axis 1
p1end pad value at end of axis 1
x2beg axis value at beginning of axis 2
x2end axis value at end of axis 2
p2beg pad value at beginning of axis 2
p2end pad value at end of axis 2
x1curve vector of x1 coordinates for points along curve
x2curve vector of x2 coordinates for points along curve
ncurve number of points along curve
curvecolor name of color to use for axes
int style NORMAL (axis 1 on bottom, axis 2 on left)
SEISMIC (axis 1 on left, axis 2 on top)

Author: Brian Macy, Phillips Petroleum Co., 11/14/98
(Adapted after Dave Hale's xDrawAxesBox routine)
```

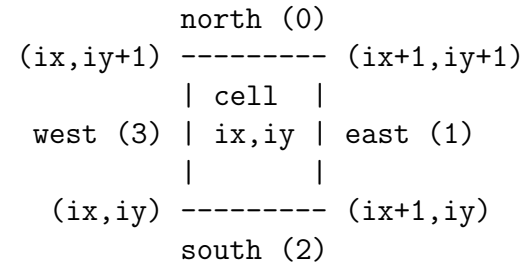
```
IMAGE - Function for making the image in an X-windows image plot

xNewImage make a new image of pixels from bytes

Function Prototype:
XImage *xNewImage (Display *dpy, unsigned long pmin, unsigned long pmax,
int width, int height, float blank, unsigned char *bytes);

Input:
dpy display pointer
pmin minimum pixel value (corresponding to byte=0)
pmax maximum pixel value (corresponding to byte=255)
width number of bytes in x dimension
height number of bytes in y dimension
blank portion for blanking (0 to 1)
bytes unsigned bytes to be mapped to an image

Author:   Dave Hale, Colorado School of Mines, 06/08/90
Revision: Brian Zook, Southwest Research Institute, 6/27/96 added blank option
  This allows replacing the low end by the background.
```

```
LEGENDBOX - draw a labeled axes box for a legend (i.e. colorscale)

Function Prototype:
void xDrawLegendBox (Display *dpy, Window win,
int x, int y, int width, int height,
float bclip, float wclip, char *units, char *legendfont,
char *labelfont, char *title, char *titlefont,
char *axescolor, char *titlecolor, char *gridcolor,
int style);

Input:
dpy display pointer
win window
x x coordinate of upper left corner of box
y y coordinate of upper left corner of box
width width of box
height height of box
units label for legend
legendfont name of font to use for legend labels
labelfont name of font to use for axes labels
title axes box title
titlefont name of font to use for title
axescolor name of color to use for axes
titlecolor name of color to use for title
gridcolor name of color to use for grid
int style NORMAL (axis 1 on bottom, axis 2 on left)
SEISMIC (axis 1 on left, axis 2 on top)
Notes:
xDrawLegendBox will determine the numbered tic incremenet and first
numbered tic automatically, if the specified increment is zero.

Pad values must be specified in the same units as the corresponding
axes values.  These pads are useful when the contents of the axes box
requires more space than implied by the axes values.  For example,
the first and last seismic wiggle traces plotted inside an axes box
will typically extend beyond the axes values corresponding to the
first and last traces.  However, all tics will lie within the limits
specified in the axes values (x1beg, x1end, x2beg, x2end).
Author: Dave Hale, Colorado School of Mines, 01/27/90
Author: Berend Scheffers , TNO Delft, 06/11/92
```

```
RUBBERBOX -  Function to draw a rubberband box in X-windows plots

xRubberBox Track pointer with rubberband box

Function Prototype:
void xRubberBox (Display *dpy, Window win, XEvent event,
int *x, int *y, int *width, int *height);

Input:
dpy display pointer
win window ID
event event of type ButtonPress

Output:
x x of upper left hand corner of box in pixels
y y of upper left hand corner of box in pixels
width width of box in pixels
height height of box in pixels

Notes:
xRubberBox assumes that event is a ButtonPress event for the 1st button;
i.e., it tracks motion of the pointer while the 1st button is down, and
it sets x, y, w, and h and returns after a ButtonRelease event for the
1st button.

Before calling xRubberBox, both ButtonRelease and Button1Motion events
must be enabled.

This is the same rubberbox.c as in Xtcwp/lib, only difference is
that xRubberBox here is XtcwpRubberBox there, and a shift has been
added to make the rubberbox more visible.

Author: Dave Hale, Colorado School of Mines, 01/27/90
```

```
WINDOW - Function to create a window in X-windows graphics

xNewWindow Create a new window and return the window ID

Function Prototype:
Window xNewWindow (Display *dpy, int x, int y, int width, int height,
int border, int background, char *name);

Input:
dpy display pointer
x x in pixels of upper left corner
y y in pixels of upper left corner
width width in pixels
height height in pixels
border border pixel
background background pixel
name name of window (also used for icon)

Notes:
The parent window is the root window.
The border_width is 4 pixels.

Author: Dave Hale, Colorado School of Mines, 01/06/90
```

```
XCONTOUR - draw contour of a two-dimensional array via X vectorplot calls

xContour draw contour of a two-dimensional array via X vector plot calls

Function Prototype:
void xContour(Display *dpy, Window win,GC gcc, GC gcl,
       float *cp,int nx, float x[], int ny, float y[], float z[],
       char lcflag,char *lcf,char *lcc, float *w, int nplaces)
Input:
c contour value
nx number of x-coordinates
x array of x-coordinates (see notes below)
ny number of y-coordinates
y array of y-coordinates (see notes below)
lcf font name of contour label
lcc color of contour label
Least Significat Bits:
z array of nx*ny z(x,y) values (see notes below)
w array of nx*ny z(x,y) values (see notes below)
Notes:
The two-dimensional array z is actually passed as a one-dimensional
array containing nx*ny values, stored with nx fast and ny slow.

The x and y arrays define a grid that is not necessarily
uniformly-sampled.  Linear interpolation of z values on the
grid defined by the x and y arrays is used to determine z values
between the gridpoints.

The two least significant bits of z are used to mark intersections
of the contour with the x,y grid; therefore, the z values will
almost always be altered (slightly) by contour.

xContour is a modified version of psContour where the use of conmove
and condraw call have been changed to match X-Windows.

Since XDrawLine requires a start and end point, the use of a manually update
of the position variables x0 and y0 is used instead of conmove.

The w array is used to restrict the range of contour labeling that
occurs only if w<1.

As suggested in the reference, the following scheme is used to refer
to a cell of the two-dimensional array z:
```



Reference:
Cottafava, G. and Le Moli, G., 1969, Automatic contour map:
Commuincations of the ACM, v. 12, n. 7, July, 1969.

Author: Morten Wendell Pedersen Aarhus University 07/20/96
Heavily based on psContour by
Dave Hale, Colorado School of Mines, 06/28/89
and with contour labeling added by: Zhenyue Liu, June 1993
(actually most of the credit should go to these two guys)

AXES - the Axes Widget

XtcwpPointInAxesRectangle returns TRUE if point is inside axes
rectangle, otherwise FALSE
XtcwpSetAxesValues set axes values
XtcwpSetAxesPads set axes pads

Function Prototype:
Boolean XtcwpPointInAxesRectangle (Widget w, Position x, Position y);
void XtcwpSetAxesValues (Widget w, float x1beg, float x1end, float x2beg,
float x2end);

void XtcwpSetAxesPads (Widget w, float p1beg, float p1end, float p2beg,
float p2end);
XtcwpPointInAxesRectangle:
Input:
w axes widget
x x coordinate of point
y y coordinate of point

XtcwpSetAxesValues:
Input:
w axes widget
x1beg axis value at beginning of axis 1
x1end axis value at end of axis 1
x2beg axis value at beginning of axis 2
x2end axis value at end of axis 2

XtcwpSetAxesPads:
Input:
w axes widget
p1beg axis pad at beginning of axis 1
p1end axis pad at end of axis 1
p2beg axis pad at beginning of axis 2
p2end axis pad at end of axis 2

Notes:
XtcwpPointInAxesRectangle:
This function is useful for determining whether or not input events
occured with the pointer inside the axes rectangle. I.e., the input
callback function will typically call this function.

XtcwpSetAxesPads:

Pad values must be specified in the same units as the corresponding
axes values. These pads are useful when the contents of the axes box
require more space than implied by the axes values. For example, the
first and last seismic wiggle traces plotted inside an axes box
will typically extend beyond the axes values corresponding to the
first and last traces. However, all tics will lie within the limits
specified in the axes values (x1beg, x1end, x2beg, and x2end).

Author: Dave Hale, Colorado School of Mines, 08/28/90
Modified: Craig Artley, Colorado School of Mines, 06/03/93, Rotate label for
   vertical axis (Courtesy Dave Hale, Advance Geophysical).

COLORMAP - Functions to manipulate X colormaps:

XtcwpCreateRGBDefaultMap create XA_RGB_DEFAULT_MAP property of root
window if it does not already exist
XtcwpGetFirstPixel return first pixel in range of contiguous
pixels in XA_RGB_DEFAULT_MAP
XtcwpGetLastPixel return last pixel in range of contiguous
pixels in XA_RGB_DEFAULT_MAP
XtcwpCreateRGBColormap create a colormap with an RGB color scale in
contiguous cells
XtcwpCreateGrayColormap create a colormap with a gray scale in contiguous cells
XtcwpCreateHueColormap create a colormap with varying hues (blue to red)
in contiguous cells
XtcwpCreateSatColormap create a colormap with varying saturations in
contiguous cells

Function Prototypes:
Status XtcwpCreateRGBDefaultMap (Display *dpy, XStandardColormap *scmap);
unsigned long XtcwpGetFirstPixel (Display *dpy);
unsigned long XtcwpGetLastPixel (Display *dpy);
Colormap XtcwpCreateRGBColormap (Display *dpy, Window win);
Colormap XtcwpCreateGrayColormap (Display *dpy, Window win);
Colormap XtcwpCreateHueColormap (Display *dpy, Window win);
Colormap XtcwpCreateSatColormap (Display *dpy, Window win,
float fhue, float lhue, float wfrac, float bright)

XtcwpCreateRGBDefaultMap:
Input:
dpy display

Output:
scmap the standard colormap structure

XtcwpGetFirstPixel, XtcwpGetLastPixel:
Input:
dpy display

XtcwpCreateRGBColormap, XtcwpCreateGrayColormap, XtcwpCreateHueColormap:
Input:
dpy display
win window

XtcwpCreateSatColormap:

Input:
dpy display
win window
fhue first hue in colormap (saturation=1)
lhue last hue in colormap (saturation=1)
wfrac fractional position of white within the colormap (saturation=0)
bright brightness

Notes:
PROBLEM

Most mid-range display devices today support what X calls
the "PseudoColor visual". Typically, only 256 colors (or gray
levels) may be displayed simultaneously. Although these 256 colors
may be chosen from a much larger (4096 or more) set of available
colors, only 256 colors can appear on a display at one time.

These 256 colors are indexed by pixel values in a table called
the colormap. Each window can have its own colormap, but only
one colormap can be installed in the display hardware at a time.
(Again, only 256 colors may be displayed at one time.) The window
manager is responsible for installing a window's colormap when that
window becomes the key window.

Many of the applications we are likely to write require a large,
contiguous range of pixels (entries in the colormap). In this
range, we must be able to:
(1) given a color (or gray), determine the corresponding pixel.
(2) given a pixel, determine the corresponding color (or gray).
An example would be an imaging application that uses a gray scale
to display images in shades of gray between black and white.
Such applications are also likely to require a few additional colors
for drawing axes, text, etc.

The problem is to coordinate the use of the limited number of
256 simultaneous colors so that windows for different applications
appear reasonable, even when their particular colormaps are not
installed in the display hardware. For example, we might expect
an analog xclock's hands to be visible even when xclock's window
is not the key window, when its colormap is not installed.

We should ensure that the range of contiguous pixels used by one

application (perhaps for imaging) does not conflict with the pixels used by other applications to draw text, clock hands, etc.

SOLUTION

Applications that do not require special colormaps should simply use the default colormap inherited from the root window when new top-level windows are created.

Applications that do require a special colormap MUST create their own colormap. They must not assume that space will be available in the default colormap for a contiguous range of read/write pixels, because the server or window manager may have already allocated these pixels as read-only. Even if sufficient pixels are available in the default colormap, they should not be allocated by a single application. The default colormap should be used only for windows requiring a limited number of typical colors, such as red, yellow, etc.

Applications that require a contiguous range of read/write pixels should allocate these pixels in their window's private colormaps. They should determine which contiguous pixels to allocate from parameters in the standard colormap XA_RGB_DEFAULT_MAP. In particular, the first pixel in the range of contiguous pixels should be
base_pixel
and the last pixel in the range should be
base_pixel+red_max*red_mult+green_max*green_mult+blue_max*blue_mult,
where base_pixel, red_max, etc. are members in the XStandardColormap structure. On an 8-bit display, this range will typically provide 216 contiguous pixels, which may be set to a gray scale, color scale, or whatever. This leaves 40 colors for drawing text, axes, etc.

If the XA_RGB_DEFAULT_MAP does not exist, it should be created to consist of various colors composed of an equal number of reds, greens, and blues. For example, if 216 colors are to be allocated, then red_max=green_max=blue_max=5, red_mult=36, green_mult=6, and blue_mult=1. Because of the difficulty in forcing a particular pixel to correspond to a particular color in read-only color cells, these 216 colors will likely be read/write color cells unless created by the X server. In any case, these 216 colors should not be modified by any application. In creating custom colormaps, the only use of XA_RGB_DEFAULT_MAP should be in determining which 216

pixels to allocate for contiguous pixels.

In creating a custom colormap for a window, the application should initialize this colormap to the colors already contained in the window's colormap, which was inherited initially from its parent. This will ensure that typical colors already allocated by other applications will be consistent with pixels used by the application requiring the custom colormap. Ideally, windows might have different colormaps, but the only differences would be in the range of contiguous colors used for imaging, rendering, etc. Ideally, the pixels corresponding to colors used to draw text, axes, etc. would be consistent for all windows.

Unfortunately, it is impractical to maintain complete consistency among various private colormaps. For example, suppose a custom colormap is created for a window before other applications have had the opportunity to allocate their colors from the default colormap. Then, when the window with the custom colormap becomes the key window, the windows of the other applications may be displayed with false colors, since the colormap of the key window may not contain the true colors. The colors used by the other applications did not exist when the custom colormap was created. One solution to this problem might be to initially allocate a set of "common" colors in the default colormap before launching any applications. This will increase the likelihood that typical colors will be consistent among various colormaps.

Functions are provided below to
(1) create the standard colormap XA_RGB_DEFAULT_MAP, if it does not exist,
(2) determine the first and last pixels in the contiguous range of pixels,
(3) create some common private colormaps - gray scale, hue scale, etc.

XtcwpCreateRGBDefaultMap:
This function returns 0 if the XA_RGB_DEFAULT_MAP property does not exist and cannot be created. At least 8 contiguous color cells must be free in the default colormap to create the XA_RGB_DEFAULT_MAP. If created, the red_max, green_max, and blue_max values returned in scmap will be equal.

XtcwpGetFirstPixel, XtcwpGetLastPixel:
If it does not already exist, XA_RGB_DEFAULT_MAP will be created. If XA_RGB_DEFAULT_MAP does not exist and cannot be created, then this function returns 0.

XtcwpCreateRGBColormap, XtcwpCreateGrayColormap, XtcwpCreateHueColormap,
XtcwpCreateSatColormap:
The returned colormap is only created; the window's colormap attribute
is not changed, and the colormap is not installed by this function.
The returned colormap is a copy of the window's current colormap, but
with an RGB color scale allocated in the range of contiguous cells
determined by XA_RGB_DEFAULT_MAP. If it does not already exist,
XA_RGB_DEFAULT_MAP will be created.

Author: Dave Hale, Colorado School of Mines, 09/30/90

```
FX - Functions to support floating point coordinates in X

FMapFX map float x to x
FMapFY map float y to y
FMapFWidth map float width to width
FMapFHeight map float height to height
FMapFAngle map float angle to angle
FMapFPoint map float x,y to x,y
FMapFPoints map float points to points
FMapX inverse map x to float x
FMapY inverse map y to float y
FMapWidth inverse map width to float width
FMapHeight inverse map height to float height
FMapAngle inverse map angle to float angle
FMapPoint map x,y to float x,y
FMapPoints map points to float points
FSetGC set graphics context
FSetMap set map (scales and shifts)
FSetClipRectangle set clip rectangle
FClipOn turn clip on
FClipOff turn clip off
FClipPoint clip point
FClipLine clip line
FClipRectangle clip rectangle
FXCreateFGC create float graphics context
FXFreeFGC free float graphic context
FXDrawPoint draw point at float x,y (with clipping)
FXDrawPoints draw float points (with clipping)
FXDrawLine draw line from float x1,y1 to float x2,y2
(with clipping)
FXDrawLines draw lines between float points (with clipping)
FXDrawRectangle draw rectangle with float x,y,width,height
(with clipping)
FXDrawArc draw arc with float x,y,width,height,angle1,angle2
FXDrawString draw string at float x,y
FXFillRectangle fill rectangle with float x,y,width,height (with
clipping)

Function Prototypes:
int FMapFX (FGC fgc, float fx);
int FMapFY (FGC fgc, float fy);
int FMapFWidth (FGC fgc, float fwidth);
int FMapFHeight (FGC fgc, float fheight);
```

```
int FMapFAngle (FGC fgc, float fangle);
void FMapFPoint (FGC fgc, float fx, float fy, int *x_return, int *y_return);
void FMapFPoints (FGC fgc, FXPoint fpoints[], int npoints,
XPoint points_return[]);
float FMapX (FGC fgc, int x);
float FMapY (FGC fgc, int y);
float FMapWidth (FGC fgc, int width);
float FMapHeight (FGC fgc, int height);
float FMapAngle (FGC fgc, int angle);
void FMapPoint (FGC fgc, int x, int y, float *fx_return, float *fy_return);
void FMapPoints (FGC fgc, XPoint points[], int npoints,
FXPoint fpoints_return[]);
void FSetGC (FGC fgc, GC gc);
void FSetMap (FGC fgc, int x, int y, int width, int height,
float fx, float fy, float fwidth, float fheight);
void FSetClipRectangle(FGC fgc, float fxa, float fya, float fxb, float fyb);
void FClipOn (FGC fgc);
void FClipOff (FGC fgc);
int FClipPoint (FGC fgc, float fx, float fy);
int FClipLine (FGC fgc, float fx1, float fy1, float fx2, float fy2,
float *fx1c, float *fy1c, float *fx2c, float *fy2c);
int FClipRectangle (FGC fgc, float fx, float fy, float fwidth, float fheight,
float *fxc, float *fyc, float *fwidthc, float *fheightc);
FGC FXCreateFGC (GC gc, int x, int y, int width, int height,
float fx, float fy, float fwidth, float fheight);
void FXFreeFGC (FGC fgc);
void FXDrawPoint (Display *display, Drawable d, FGC fgc, float fx, float fy);
void FXDrawPoints (Display *display, Drawable d, FGC fgc,
FXPoint fpoints[], int npoints, int mode);
void FXDrawLine (Display *display, Drawable d, FGC fgc,
float fx1, float fy1, float fx2, float fy2);
void FXDrawLines (Display *display, Drawable d, FGC fgc,
FXPoint fpoints[], int npoints, int mode);
void FXDrawRectangle (Display *display, Drawable d, FGC fgc,
float fx, float fy, float fwidth, float fheight);
void FXDrawArc (Display *display, Drawable d, FGC fgc,
float fx, float fy, float fwidth, float fheight,
float fangle1, float fangle2);
void FXDrawString (Display *display, Drawable d, FGC fgc,
float fx, float fy, char *string, int length);
void FXFillRectangle (Display *display, Drawable d, FGC fgc,
float fx, float fy, float fwidth, float fheight);
```

Notes:
The functions defined below are designed to resemble the equivalent
X functions. For example, FXDrawLine() is analogous to XDrawLine.
Each of the FXDraw<xxx>() functions requires an FGC instead of a GC
(graphics context). An FGC contains a GC, along with the information
required to transform floating point coordinates to integer (pixel)
coordinates.

Additional functions are provided to transform floating point coordinates
to integer coordinates and vice versa. Where feasible, macros are also
provided to perform these coordinate transformations.

Clipping of floating point coordinates is supported, because clipping
after mapping to integer coordinates is not valid when the mapped
integer coordinates overflow the range of short integers. By clipping
the floating point coordinates before mapping to integers, this overflow
can be avoided. By default, clipping is turned off until a clip rectangle
is specified or until clipping is explicitly turned on. Clipping is
not currently supported for all FXDraw functions.

Author: Dave Hale, Colorado School of Mines, 07/24/90
Modified: Dave Hale, Colorado School of Mines, 05/18/91
Added floating point clipping capability to some FXDraw functions.

MISC - Miscellaneous X-Toolkit functions

XtcwpDrawString90 Draw a string rotated 90 degrees counter-clockwise

Function Prototype:
void XtcwpDrawString90 (Display *dpy, Drawable d, GC gc,
int x, int y, char *string, int count);

Input:
dpy X display
d X drawable
gc X graphics context
x,y coordinates of baseline starting position of the string
string array[count] of characters to be drawn
count number of characters in string

Author: Dave Hale, Advance Geophysical, 06/03/93

```
RESCONV - general purpose resource type converters

XtcwpStringToFloat convert  string to float in resource

Function Prototype:
void XtcwpStringToFloat (XrmValue *args, int *nargs,
XrmValue *fromVal, XrmValue *toVal);

Author:  Dave Hale, Colorado School of Mines, 08/28/90
```

RUBBERBOX - Function to draw a rubberband box in X-windows plots

XtcwpRubberbox Track pointer with rubberband box

Function Prototype:
void XtcwpRubberbox (Display *dpy, Window win, XEvent event,
int *x, int *y, int *width, int *height);

Input:
dpy display pointer
win window ID
event event of type ButtonPress

Output:
x x of upper left hand corner of box in pixels
y y of upper left hand corner of box in pixels
width width of box in pixels
height height of box in pixels

Notes:
XtcwpRubberbox assumes that event is a ButtonPress event for the 1st button;
i.e., it tracks motion of the pointer while the 1st button is down, and
it sets x, y, w, and h and returns after a ButtonRelease event for the
1st button.

Before calling XtcwpRubberbox, both ButtonRelease and Button1Motion events
must be enabled.

Author: Dave Hale, Colorado School of Mines, 01/27/90

```
RADIOBUTTONS -  convenience functions creating and using radio buttons

XtcwpCreateStringRadioButtons create an XmFrame containing radio
buttons labeled with strings

Function Prototypes:
Widget XtcwpCreateStringRadioButtons (Widget parent, char *label,
int nstrings, char **strings, int first,
void (*callback)(int selected, void *clientdata), void *clientdata);

Input:
parent parent widget
label label for this collection of radio bottons
nstrings number of strings
strings array[nstrings] of character strings, one per button
first index of button to be initially selected
callback function called when radio buttons change state
clientdata pointer to client data to be passed to callback

Notes:
This code depends on the Motif Developer's Package.

An integer index of the selected button (along with the clientdata pointer)
is passed to the callback function.

The returned XmFrame is not managed.

Author:  Dave Hale, Colorado School of Mines, 08/28/90
```

SAMPLES - Motif-based Graphics Functions

samplesCreate
samplesDraw
samplesSetN
samplesSetData
samplesSetPlotValue
samplesSetEditMode
samplesSetOrigin

Function Prototypes:

```
Samples *samplesCreate (Widget parent, char *title,
void (*editDone)(Samples *s));
void samplesDraw (Samples *s);
void samplesSetN (Samples *s, int n);
void samplesSetData (Samples *s, float *d);
void samplesSetPlotValue (Samples *s, float pv);
void samplesSetEditMode (Samples *s, EditMode m);
void samplesSetOrigin (Samples *s, int i);
```

Notes:
Watch this space.

Author: Dave Hale, Colorado School of Mines

```
FGETGTHR - get gathers from SU datafiles

fget_gather - get a gather from a file
get_gather - get a gather from stdin


Function Prototypes:
segy **fget_gather(FILE *fp, cwp_String *key,cwp_String *type,Value *n_val,
int *nt,int *ntr, float *dt,int *first);
segy **get_gather(cwp_String *key,cwp_String *type,Value *n_val,
int *nt,int *ntr, float *dt,int *first)

fget_gather - get a gather from a file
Input:
fp file pointer of input file
key header key of ensemble
type header value type
value value of ensemble key word
nt number of time samples
ntr number of traces in ensemble
dt time sampling interval
first flag is this the first ensemble being read?
Notes:
The input seismic dataset must be sorted into ensembles defined by key
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.

get_gather - get a gather from stdin
Input:
key header key of ensemble
type header value type
value value of ensemble key word
nt number of time samples
ntr number of traces in ensemble
dt time sampling interval
first flag is this the first ensemble being read?
Notes:
The input seismic dataset must be sorted into ensembles defined by key
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.

segy tr;
```

```
segy **fget_gather(FILE *fp, cwp_String *key,cwp_String *type,Value *n_val,
int *nt,int *ntr, float *dt,int *first)
fget_gather - get a gather from a file
Input:
fp file pointer of input file
key header key of ensemble
type header value type
value value of ensemble key word
nt number of time samples
ntr number of traces in ensemble
dt time sampling interval
first flag is this the first ensemble being read?
Notes:
The input seismic dataset must be sorted into ensembles defined by key
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.
{
int nsegy;
FILE *tracefp=NULL;
FILE *headerfp=NULL;
static FILE *ntracefp=NULL; /* first different trace pointer
static FILE *nheaderfp=NULL;
segy **rec=NULL;
int ntrr=0;
int indx=0;
static Value val;


*type = hdtype(*key);
indx = getindex(*key);
        *ntr = 0;


if(*first==0) {
        /* get info from first trace
nsegy = fvgettr(fp,&tr);
if (nsegy==0)  err("can't get first trace");
*nt = tr.ns;
         *dt = (float) tr.dt/1000000.0;
         ++ntrr;
gethval(&tr, indx, n_val);
ntracefp = etmpfile();
nheaderfp = etmpfile();
```

```
*first=1;
} else {
/* This is the first trace of the nex gather
erewind(nheaderfp);
         erewind(ntracefp);
                fread (&tr,HDRBYTES, 1, nheaderfp);
                fread (tr.data,FSIZE, *nt, ntracefp);
gethval(&tr, indx, n_val);
}



        /* Store traces in tmpfile while getting a count
tracefp = etmpfile();
headerfp = etmpfile();
        do {
                efwrite(&tr, 1, HDRBYTES, headerfp);
                efwrite(tr.data, FSIZE, *nt, tracefp);

/* read the next trace
*ntr+=1;
val=*n_val;
nsegy = fvgettr(fp,&tr);
if(nsegy) ntrr++;
gethval(&tr, indx, n_val);
} while (nsegy && !valcmp(*type,val,*n_val));


/* If there are no more traces then return
if(nsegy==0 && ntrr==0 ) {
 *ntr=0;
 efclose(nheaderfp);
 efclose(ntracefp);
 return(rec=NULL);
} else {
/* store the first trace of the next gather
erewind(nheaderfp);
         erewind(ntracefp);
                efwrite(&tr, 1, HDRBYTES, nheaderfp);
                efwrite(tr.data, FSIZE, *nt, ntracefp);
}

/* allocate memory for the record
```

```
{ register int i;
rec = ealloc1(*ntr,sizeof(segy *));
for(i=0;i<*ntr;i++)
rec[i] = (segy *)ealloc1((*nt*FSIZE+HDRBYTES),sizeof(char));
}


/* load traces into an array and close temp file
erewind(headerfp);
        erewind(tracefp);
{ register int ix;
         for (ix=0; ix<*ntr; ix++)
                 fread (rec[ix],HDRBYTES, 1, headerfp);
         efclose (headerfp);
for(ix=0; ix<*ntr; ix++)
                 fread ((*rec[ix]).data,FSIZE, *nt, tracefp);
         efclose (tracefp);
}

return(rec);
}

segy tr;

segy **get_gather(cwp_String *key,cwp_String *type,Value *n_val,
int *nt,int *ntr, float *dt,int *first)
get_gather - get a gather from stdin
Input:
key header key of ensemble
type header value type
value value of ensemble key word
nt number of time samples
ntr number of traces in ensemble
dt time sampling interval
first flag is this the first ensemble being read?
Notes:
The input seismic dataset must be sorted into ensembles defined by key
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.
{
int nsegy;
FILE *tracefp=NULL;
FILE *headerfp=NULL;
```

```
static FILE *ntracefp=NULL; /* first different trace pointer
static FILE *nheaderfp=NULL;
segy **rec=NULL;
int ntrr=0;
int indx=0;
static Value val;


*type = hdtype(*key);
indx = getindex(*key);
        *ntr = 0;


if(*first==0) {
        /* get info from first trace
nsegy = vgettr(&tr);
if (nsegy==0)  err("can't get first trace");
*nt = tr.ns;
         *dt = (float) tr.dt/1000000.0;
         ++ntrr;
gethval(&tr, indx, n_val);
ntracefp = etmpfile();
nheaderfp = etmpfile();
*first=1;
} else {
/* This is the first trace of the nex gather
erewind(nheaderfp);
         erewind(ntracefp);
                fread (&tr,HDRBYTES, 1, nheaderfp);
                fread (tr.data,FSIZE, *nt, ntracefp);
gethval(&tr, indx, n_val);
}



        /* Store traces in tmpfile while getting a count
tracefp = etmpfile();
headerfp = etmpfile();
        do {
                efwrite(&tr, 1, HDRBYTES, headerfp);
                efwrite(tr.data, FSIZE, *nt, tracefp);

/* read the next trace
```

```
*ntr+=1;
val=*n_val;
nsegy = vgettr(&tr);
if(nsegy) ntrr++;
gethval(&tr, indx, n_val);
} while (nsegy && !valcmp(*type,val,*n_val));


/* If there are no more traces then return
if(nsegy==0 && ntrr==0 ) {
 *ntr=0;
 efclose(nheaderfp);
 efclose(ntracefp);
 return(rec=NULL);
} else {
/* store the first trace of the next gather
erewind(nheaderfp);
         erewind(ntracefp);
                efwrite(&tr, 1, HDRBYTES, nheaderfp);
                efwrite(tr.data, FSIZE, *nt, ntracefp);
}

/* allocate memory for the record
{ register int i;
rec = ealloc1(*ntr,sizeof(segy *));
for(i=0;i<*ntr;i++)
rec[i] = (segy *)ealloc1((*nt*FSIZE+HDRBYTES),sizeof(char));
}


/* load traces into an array and close temp file
erewind(headerfp);
        erewind(tracefp);
{ register int ix;
         for (ix=0; ix<*ntr; ix++)
                 fread (rec[ix],HDRBYTES, 1, headerfp);
         efclose (headerfp);
for(ix=0; ix<*ntr; ix++)
                 fread ((*rec[ix]).data,FSIZE, *nt, tracefp);
         efclose (tracefp);
}

return(rec);
```

```
}
```

```
fgethdr - get segy tape identification headers from the file by file pointer

Input:
fp         file pointer

Output:
chdr       3200 bytes of segy character header
bhdr        400 bytes of segy binary header
Authors:  zhiming li  and j. dulac ,   unocal
 modified for CWP/SU: R. Beardsley
```

```
FGETTR - Routines to get an SU trace from a file

fgettr get a fixed-length segy trace from a file by file pointer
fvgettr get a variable-length segy trace from a file by file pointer
fgettra get a fixed-length trace from disk file by trace number
gettr macro using fgettr to get a trace from stdin
vgettr macro using vfgettr to get a trace from stdin
gettra macro using fgettra to get a trace from stdin by trace number

Function Prototype:
int fgettr(FILE *fp, segy *tp);
int fvgettr(FILE *fp, segy *tp);
int fgettra(FILE *fp, segy *tp, int itr);

Returns:
fgettr, fvgettr:
int: number of bytes read on current trace (0 after last trace)

fgettra:
int: number of traces in disk file

Macros defined in segy.h
define gettr(x) fgettr(stdin, (x))
define vgettr(x) fgettr(stdin, (x))

Usage example:
  segy tr;
  ...
  while (gettr(&tr)) {
  tr.offset = abs(tr.offset);
  puttr(&tr);
  }
  ...

Authors: SEP: Einar Kjartansson, Stew Levin CWP: Shuki Ronen, Jack Cohen

 Revised: 7/2/95 Stewart A. Levin   Mobil
     Major rewrite:  Use xdr library for portable su output file
     format.   Merge fgettr and fgettra into same source file.
     Make input from multiple streams work (at long last!).
 Revised: 11/22/95 Stewart A. Levin  Mobil
     Always set ntr for DISK input.  This fixes susort failure.
 Revised: 1/9/96  jkc CWP
```

```
    Set lastfp on nread <=0 return, too.
Revised: 28 Mar, 2006 Stewart A. Levin   Landmark Graphics
    Reworked XDR to support random seeks on > 2GB files
    and to read big endian SHORTPACK data on little endian machines.
```

```
FPUTGTHR - put gathers to a file

fput_gather - put a gather to a file
put_gather - put a gather to stdout

Function Prototypes:
segy **fput_gather(FILE *fp, segy **rec,int *nt, int *ntr);
segy **put_gather(segy **rec,int *nt, int *ntr)

fput_gather - put a gather to a file
Input:
fp pointer to output file
rec array of segy traces
nt number of time samples per trace
ntr number of traces in ensemble
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.

put_gather - put a gather to stdout
Input:
rec array of segy traces
nt number of time samples per trace
ntr number of traces in ensemble
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.

include "su.h"
include "segy.h"
include "header.h"


segy tr;

segy **fput_gather(FILE *fp, segy **rec,int *nt, int *ntr)
fput_gather - put a gather to a file
Input:
fp pointer to output file
rec array of segy traces
nt number of time samples per trace
ntr number of traces in ensemble
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.
{
```

```
segy tr;

{ register int i;
for(i=0;i<*ntr;i++) {
memcpy( (void *) &tr, (const void *) rec[i],
*nt*FSIZE+HDRBYTES);
fvputtr(fp,&tr);
free1((void *)rec[i]);
}
}
return(rec=NULL);
}


segy **put_gather(segy **rec,int *nt, int *ntr)
put_gather - put a gather to stdout
Input:
rec array of segy traces
nt number of time samples per trace
ntr number of traces in ensemble
Author: Potash Corporation Sascatchewan, Balasz Nemeth
given to CWP in 2008.
{
segy tr;

{ register int i;
for(i=0;i<*ntr;i++) {
memcpy( (void *) &tr, (const void *) rec[i],
*nt*FSIZE+HDRBYTES);
vputtr(&tr);
free1((void *)rec[i]);
}
}
return(rec=NULL);
}
```

```
FPUTTR - Routines to put an SU trace to a file

fputtr put a segy trace to a file by file pointer
fvputtr put a segy trace to a file by file pointer (variable ns)
puttr macro using fputtr to put a trace to stdin
vputtr macro using fputtr to put a trace to stdin (variable ns)

Function Prototype:
void fputtr(FILE *fp, segy *tp);
void fvputtr(FILE *fp, segy *tp);

Returns:

void

Notes:

The functions puttr(x) vputtr(x) are macros defined in segy.h
define puttr(x) fputtr(stdin, (x))
define vputtr(x) fvputtr(stdin, (x))

Usage example:
  segy tr;
  ...
  while (gettr(&tr)) {
  tr.offset = abs(tr.offset);
  puttr(&tr);
  }
  ...

Authors: SEP: Einar Kjartansson, Stew Levin CWP: Shuki Ronen, Jack Cohen
```

```
HDRPKGE - routines to access the SEGY header via the hdr structure.

gethval get a trace header word by index
puthval put a trace header word by index
getbhval get a binary header word by index
putbhval put a binary header word by index
gethdval get a trace header word by name
puthdval put a trace header word by name


hdtype get the data type of a trace header word by name
getkey get the name of a trace header word from its index

getindex get the index of a trace header word from the name

swaphval swap the trace header words by index
swapbhval swap the binary header words by index
gettapehval get a tape trace header word by index
puttapehval put a tape trace header word by index
gettapebhval get a tape binary header word by index
puttapebhval put a tape binary header word by index
printheader display non-null header field values

Function Prototypes:
void gethval(const segy *tr, int index, Value *valp);
void puthval(segy *tr, int index, Value *valp);
void putbhval(bhed *bh, int index, Value *valp);
void getbhval(const bhed *bh, int index, Value *valp);
void gethdval(const segy *tr, char *key, Value *valp);
void puthdval(segy *tr, char *key, Value *valp);
char *hdtype(const char *key);
char *getkey(const int index);
int getindex(const char *key);
void swaphval(segy *tr, int index);
void swapbhval(bhed *bh, int index);
void gettapehval(tapesegy *tapetr, int index, Value *valp);
void puttapehval(tapesegy *tapetr, int index, Value *valp);
void gettapebhval(tapebhed *tapetr, int index, Value *valp);
void puttapebhval(tapebhed *tapetr, int index, Value *valp);
void printheader(const segy *tp);

Notes:
This package includes only those routines that directly access
```

the "hdr" or "bhdr" structures. It does not include routines
such as printfval, printftype, printfhead that use the routines
in this package to indirectly access these structures.

Note that while gethdval and puthdval are more convenient to use
than gethval and puthval, they incur an inefficiency in the
common case of iterating code over a set of traces with a fixed
key or keys. In such cases, it is advisable to set the index
or indices outside the loop using getindex.

swaphval:
Byte-swapping is needed for converting SU data from big-endian to little-
endian formats, and vice versa. The swap_.... subroutines are based
on subroutines provided by Jens Hartmann of the Institut fur Geophysik
in Hamburg. These are found in .../cwp/lib/swapbyte.c.
Authors: SEP: Einar Kjartansson CWP: Jack Cohen, Shuki Ronen
swaphval: CWP: John Stockwell

TABPLOT - TABPLOT selected sample points on selected trace

tabplot tabplot selected sample points on selected trace

Function Prototype:
void tabplot(segy *tp, int itmin, int itmax);

Input:
tp pointer to a segy
itmin minimum time sample printed
itmax maximum time sample printed

Authors: CWP: Brian Sumner, Jack K. Cohen

```
VALPKGE - routines to handle variables of type Value

vtoi cast Value variable as an int
vtol cast Value variable as a long
vtof cast Value variable as a float
vtod cast Value variable as a double
atoval convert ascii to Value
valtoabs take absolute value of a Value variable
valcmp compare Value variables
printfval printf a Value variable
fprintfval fprintf a Value variable
scanfval scanf a Value variable
printftype printf for the type of a segy header word

Function Prototypes:
int vtoi(register cwp_String type, Value val);
long vtol(register cwp_String type, Value val);
float vtof(register cwp_String type, Value val);
double vtod(register cwp_String type, Value val);
void atoval(cwp_String type, cwp_String keyval, Value *valp);
Value valtoabs(cwp_String type, Value val);
int valcmp(register cwp_String type, Value val1, Value val2);
void printfval(register cwp_String type, Value val);
void fprintfval(FILE *stream, register cwp_String type, Value val);
void scanfval(register cwp_String type, Value *valp);

Notes:
A Value is defined by the following in .../su/include/su.h:

typedef union { * storage for arbitrary type *
char s[8];
short h;
unsigned short u;
long l;
unsigned long v;
int i;
unsigned int p;
float f;
double d;
unsigned int U:16;
unsigned int P:32;
} Value;
```

The use of the valpkge routines, as well as the hdrpkge routines,
permits the user to change the definition of the types of the
various fields of the segy data type, without breaking codes that
look at part or all of these fields.

Authors: CWP: Jack K. Cohen, Shuki Ronen

```
ABEL - Functions to compute the discrete ABEL transform:

abelalloc allocate and return a pointer to an Abel transformer
abelfree  free an Abel transformer
abel compute the Abel transform

Function prototypes:
void *abelalloc (int n);
void abelfree (void *at);
void abel (void *at, float f[], float g[]);

Input:
ns number of samples in the data to be transformed
f[] array of floats, the function being transformed

Output:
at pointer to Abel transformer returned by abelalloc(int n)
g[] array of floats, the transformed data returned by
abel(*at,f[],g[])

Notes:
The Abel transform is defined by:

         Infinity
g(y) = 2 Integral dx f(x)/sqrt(1-(y/x)^2)
   |y|

Linear interpolation is used to define the continuous function f(x)
corresponding to the samples in f[].  The first sample f[0] corresponds
to f(x=0) and the sampling interval is assumed to be 1.  Therefore, the
input samples correspond to 0 <= x <= n-1.  Samples of f(x) for x > n-1
are assumed to be zero.  These conventions imply that

g[0] = f[0] + 2*f[1] + 2*f[2] + ... + 2*f[n-1]

References:
Hansen, E. W., 1985, Fast Hankel transform algorithm:  IEEE Trans. on
Acoustics, Speech and Signal Processing, v. ASSP-33, n. 3, p. 666-671.
(Beware of several errors in the equations in this paper!)

Authors:  Dave Hale and Lydia Deng, Colorado School of Mines, 06/01/90
```

```
AIRY - Approximate the Airy functions  Ai(x), Bi(x) and their respective
       derivatives  Ai'(x), Bi'(x)

airya return approximation of Ai(x)
airypa return approximation of Ai'(x)
airyb return approximation of Bi(x)
airybp return approximation of Bi'(x)

Function Prototypes:
float airya (float x);
float airyap (float x);
float airyb (float x);
float airybp (float x);

Input:
x value at which to evaluate Ai(x)

Returned:
airya Ai(x)
airypa Ai'(x)
airyb Bi(x)
airybp Bi'(x)

Reference:
The approximation is derived from tables and formulas in Abramowitz
and Stegun, p. 475-477.

Author:  Dave Hale, Colorado School of Mines, 06/06/89
```

```
ALLOC - Allocate and free multi-dimensional arrays

alloc1 allocate a 1-d array
realloc1 re-allocate a 1-d array
free1 free a 1-d array
alloc2 allocate a 2-d array
free2 free a 2-d array
alloc3 allocate a 3-d array
free3 free a 3-d array
alloc4 allocate a 4-d array
free4 free a 4-d array
alloc5 allocate a 5-d array
free5 free a 5-d array
alloc6 allocate a 6-d array
free6 free a 6-d arrayalloc1int
allocate a 1-d array of ints
realloc1int re-allocate a 1-d array of ints
free1int free a 1-d array of ints
alloc2int allocate a 2-d array of ints
free2int free a 2-d array of ints
alloc3int allocate a 3-d array of ints
free3int free a 3-d array of ints
alloc1float allocate a 1-d array of floats
realloc1float re-allocate a 1-d array of floats
free1float free a 1-d array of floats
alloc2float allocate a 2-d array of floats
free2float free a 2-d array of floats
alloc3float allocate a 3-d array of floats
free3float free a 3-d array of floats
alloc4float allocate a 4-d array of floats
free4float      free a 4-d array of floats
alloc5float     allocate a 5-d array of floats
free5float      free a 5-d array of floats
alloc6float     allocate a 6-d array of floats
free6float      free a 6-d array of floats
alloc4int       allocate a 4-d array of ints
free4int        free a 4-d array of ints
alloc5int       allocate a 5-d array of ints
free5int        free a 5-d array of ints
alloc5uchar allocate a 5-d array of unsigned chars
free5uchar free a 5-d array of unsiged chars
alloc5ushort    allocate a 5-d array of unsigned shorts
free5ushort     free a 5-d array of unsiged shorts
```

```
alloc6ushort    allocate a 6-d array of unsigned shorts
free6ushort     free a 6-d array of unsiged shorts
alloc1double allocate a 1-d array of doubles
realloc1double re-allocate a 1-d array of doubles
free1double free a 1-d array of doubles
alloc2double allocate a 2-d array of doubles
free2double free a 2-d array of doubles
alloc3double allocate a 3-d array of doubles
free3double free a 3-d array of doubles
alloc1complex allocate a 1-d array of complexs
realloc1complex re-allocate a 1-d array of complexs
free1complex free a 1-d array of complexs
alloc2complex allocate a 2-d array of complexs
free2complex free a 2-d array of complexs
alloc3complex allocate a 3-d array of complexs
free3complex free a 3-d array of complexs

alloc1dcomplex   allocate a 1-d array of complexs
realloc1dcomplex re-allocate a 1-d array of complexs
free1dcomplex    free a 1-d array of complexs
alloc2dcomplex   allocate a 2-d array of complexs
free2dcomplex    free a 2-d array of complexs
alloc3dcomplex   allocate a 3-d array of complexs
free3dcomplex    free a 3-d array of complexs

Function Prototypes:
void *alloc1 (size_t n1, size_t size);
void *realloc1 (void *v, size_t n1, size_t size);
void free1 (void *p);
void **alloc2 (size_t n1, size_t n2, size_t size);
void free2 (void **p);
void ***alloc3 (size_t n1, size_t n2, size_t n3, size_t size);
void free3 (void ***p);
void ****alloc4 (size_t n1, size_t n2, size_t n3, size_t n4, size_t size);
void free4 (void ****p);
                  size_t size);
int *alloc1int (size_t n1);
int *realloc1int (int *v, size_t n1);
void free1int (int *p);
int **alloc2int (size_t n1, size_t n2);
void free2int (int **p);
int ***alloc3int (size_t n1, size_t n2, size_t n3);
void free3int (int ***p);
```

```
float *alloc1float (size_t n1);
float *realloc1float (float *v, size_t n1);
void free1float (float *p);
float **alloc2float (size_t n1, size_t n2);
void free2float (float **p);
float ***alloc3float (size_t n1, size_t n2, size_t n3);
void free3float (float ***p);
float ****alloc4float (size_t n1, size_t n2, size_t n3, size_t n4);
void free4float (float ****p);
                         size_t n6);
int ****alloc4int (size_t n1, size_t n2, size_t n3, size_t n4);
void free4int (int ****p);
size_t n5);
        size_t n5);
        size_t n5,size_t n6);
double *alloc1double (size_t n1);
double *realloc1double (double *v, size_t n1);
void free1double (double *p);
double **alloc2double (size_t n1, size_t n2);
void free2double (double **p);
double ***alloc3double (size_t n1, size_t n2, size_t n3);
void free3double (double ***p);
complex *alloc1complex (size_t n1);
complex *realloc1complex (complex *v, size_t n1);
void free1complex (complex *p);
complex **alloc2complex (size_t n1, size_t n2);
void free2complex (complex **p);
complex ***alloc3complex (size_t n1, size_t n2, size_t n3);
void free3complex (complex ***p);

complex *alloc1dcomplex (size_t n1);
complex *realloc1dcomplex (dcomplex *v, size_t n1);
void free1dcomplex (dcomplex *p);
complex **alloc2dcomplex (size_t n1, size_t n2);
void free2dcomplex (dcomplex **p);
complex ***alloc3dcomplex (size_t n1, size_t n2, size_t n3);
void free3dcomplex (dcomplex ***p);


Notes:
The functions defined below are intended to simplify manipulation
of multi-dimensional arrays in scientific programming in C.  These
functions are useful only because true multi-dimensional arrays
```

```
in C cannot have variable dimensions (as in FORTRAN).  For example,
the following function IS NOT valid in C:
void badFunc(a,n1,n2)
float a[n2][n1];
{
a[n2-1][n1-1] = 1.0;
}
However, the following function IS valid in C:
void goodFunc(a,n1,n2)
float **a;
{
a[n2-1][n1-1] = 1.0;
}
Therefore, the functions defined below do not allocate true
multi-dimensional arrays, as described in the C specification.
Instead, they allocate and initialize pointers (and pointers to
pointers) so that, for example, a[i2][i1] behaves like a 2-D array.

The array dimensions are numbered, which makes it easy to add
functions for arrays of higher dimensions.  In particular,
the 1st dimension of length n1 is always the fastest dimension,
the 2nd dimension of length n2 is the next fastest dimension,
and so on.  Note that the 1st (fastest) dimension n1 is the
first argument to the allocation functions defined below, but
that the 1st dimension is the last subscript in a[i2][i1].
(This is another important difference between C and Fortran.)

The allocation of pointers to pointers implies that more storage
is required than is necessary to hold a true multi-dimensional array.
The fraction of the total storage allocated that is used to hold
pointers is approximately 1/(n1+1).  This extra storage is unlikely
to represent a significant waste for large n1.

The functions defined below are significantly different from similar
functions described by Press et al, 1988, NR in C.
In particular, the functions defined below:
(1) Allocate arrays of arbitrary size elements.
(2) Allocate contiguous storage for arrays.
(3) Return NULL if allocation fails (just like malloc).
(4) Do not provide arbitrary lower and upper bounds for arrays.

Contiguous storage enables an allocated multi-dimensional array to
be passed to a C function that expects a one-dimensional array.
```

```
For example, to allocate and zero an n1 by n2 two-dimensional array
of floats, one could use
a = alloc2(n1,n2,sizeof(float));
zeroFloatArray(n1*n2,a[0]);
where zeroFloatArray is a function defined as
void zeroFloatArray(int n, float *a)
{
int i;
for (i=0; i<n; i++)
a[i] = 0.0;
}

Internal error handling and arbitrary array bounds, if desired,
should be implemented in functions that call the functions defined
below, with the understanding that these enhancements may limit
portability.

Author:     Dave Hale, Colorado School of Mines, 12/31/89
                Zhaobo Meng, added 4D, 5D and 6D functions, 1996
```

```
ANTIALIAS - Butterworth anti-aliasing filter

antialias use before increasing the sampling interval of data
 i.e. subsampling

Function Prototype:
void antialias (float frac, int phase, int n, float p[], float q[]);

Input:
frac current sampling interval / future interval (should be <= 1)
phase =0 for zero-phase filter; =1 for minimum-phase filter
n number of samples
p array[n] of input samples

Output:
q array[n] of output (anti-alias filtered) samples

Notes:
The anti-alias filter is a recursive (Butterworth) filter.  For zero-phase
anti-alias filtering, the recursive filter is applied forwards and backwards.

Author:  Dave Hale, Colorado School of Mines, 06/06/90
```

```
AXB - Functions to solve a linear system of equations Ax=b by LU
decomposition, invert a square matrix or directly multiply an
inverse matrix by another matrix (without explicitely computing
the inverse).

LU_decomposition Decompose a matrix (A) into a lower triangular (L)
and an upper triangular (U) such that A=LU

backward_substitution Apply backward substitution to an LU decomposed
matrix to solve the linear system of equations Ax=b

inverse_matrix compute the inverse of a square non-singular matrix

inverse_matrix_multiply computes the product A^(-1)*B without explicitely
computing the inverse matrix

Function prototypes:
void LU_decomposition (int nrows, float **matrix, int *idx, float *d);
void backward_substitution (int nrows, float **matrix, int *idx, float *b);
void inverse_matrix (int nrows, float **matrix);
void inverse_matrix_multiply (int nrows1, float **matrix1, int ncols2,
        int nrows2, float **matrix2, float **out_matrix);

LU_decomposition:
Input:
nrows number of rows of matrix to invert
matrix matrix of coefficients in linear system Ax=b

Output:
matrix matrix containing LU decomposition (original matrix destroyed)
idx vector recording the row permutations effected by partial
pivoting
d +/- 1 depending on whether the number of row interchanges
was even or odd
backward_substitution
Input:
nrows number of rows (and columns) of input matrix
matrix matrix of coefficients (after LU decomposition)
idx permutation vector obtained from routine LU_decomposition
b right hand side vector in equation Ax=b

Output:
b vector with the solution
```

```
inverse_matrix
Input:
nrows number of rows (and columns) of input matrix
matrix matrix to invert

Output:
matrix inverse of input matrix
inverse_matrix_multiply
nrows1          number of rows (and columns) of matrix to invert
matrix1         square matrix to invert
ncols2          number of coulmns of second matrix
nrows2 number of rows of second matrix
matrix          second matrix (multiplicator)

Output Parameters:
out_matrix      matrix containing the product of the inverse of the first
                matrix by the second one.
Note:
matrix1 and matrix2 are not destroyed, (not clobbered)
Notes:
To solve the set of linear equations Ax=b, first do the LU decomposition of
A (which will clobber A with its LU decomposition) and then do the backward
substitution with this new matrix and the right-hand side vector b. The vector
b will be clobbered with the solution. Both, the original matrix and vector B,
will have been destroyed.

The LU decomposition is carried out with the Crout's method with implicit
partial pivoting that guaratees that the maximum pivot is used in every
step of the algorithm.

The operation count to solve a linear system of equations via LU decomposition
is 1/3N^3 and is a factor of 3 better than the standard Gauss-Jordan algorithm
To invert a matrix the count is the same with both algorithms: N^3.

Once a linear system Ax=b has been solved, to solve another linear system
with the same matrix A but with different vetor b, ONLY the back substitution
has to be repeated with the new b (remember that the matrix in backsubstitution
is not the original matrix but its LU decomposition)

If you want to compute A^(-1)*B from matrices A and B, it is better to
use the subroutine inverse_matrix_multiply rather than explicitely computing
the inverse. This saves a whole martix multiplication and is also more accurate.
```

Refferences:
Press, Teukolsky, Vettering and Flannery, Numerical Recipes in C:
The art of scientific computing. Cambridge University Press.
second edition. (1992).
Golub and Van Loan, Matrix Computations. John Hopkins University Press.
Second Edition. (1989).
Horn and Johnson, Matrix Analysis. Cambridge University Press. (1985).
Credits:
Adapted from discussions in Numerical Recipes, by Gabriel Alvarez (1995)

```
BIGMATRIX - Functions to manipulate 2-dimensional matrices that are too big
    to fit in real memory, but that are small enough to fit in
virtual memory:

bmalloc allocate a big matrix
bmfree free a big matrix
bmread read a vector from a big matrix
bmwrite write a vector to a big matrix

Function Prototypes:
void *bmalloc (int nbpe, int n1, int n2);
void bmfree (void *bm);
void bmread (void *bm, int dir, int k1, int k2, int n, void *v);
void bmwrite (void *bm, int dir, int k1, int k2, int n, void *v);

bmalloc:
Input:
nbpe number of bytes per matrix element
n1 number of elements in 1st (fastest) dimension
n2 number of elements in 2nd (slowest) dimension

Returned:
bm pointer to big matrix

bmfree:
Input:
bm pointer to big matrix state (returned by bmalloc)

bmread:
Input:
bm     pointer to big matrix state (returned by bmalloc)
d   = 1 or 2:  direction in which to read matrix elements
k1 1st dimension index of first matrix element to read
k2 2nd dimension index of first matrix element to read
n number of elements to read

Output:
v array[n] to contain matrix elements read

bmwrite:
Input:
bm     pointer to big matrix state (returned by bmalloc)
d   = 1 or 2:  direction in which to write matrix elements
```

```
k1 1st dimension index of first matrix element to write
k2 2nd dimension index of first matrix element to write
n number of elements to write
v array[n] containing matrix elements to write

Notes:
The bm functions provide access to a big 2-dimensional matrix along
either the 1st or 2nd dimensions.  Although, the matrix must be small
enough to fit in virtual memory, it may be too large to fit in real memory.
These functions provide equally efficient (or equally inefficient) access
to vectors in a big matrix along either the 1st or 2nd dimensions.

For example, the following algorithm will efficiently transpose an
n1 by n2 array of (n1*n2) floats stored in a file:

void *bm;
float *v;
bm = bmalloc(sizeof(float),n1,n2);
for (i2=0; i2<n2; i2++) {
(read n1 floats from input file into array v);
bmwrite(bm,1,0,i2,n1,(char*)v);
}
for (i1=0; i1<n1; i1++) {
bmread(bm,2,i1,0,n2,(char*)v);
(write n2 floats in array v to output file);
}
bmfree(bm);

Author:  Dave Hale, Colorado School of Mines, 05/17/89
```

```
BUTTERWORTH - Functions to design and apply Butterworth filters:

bfdesign design a Butterworth filter
bfhighpass apply a high-pass Butterworth filter
bflowpass apply a low-pass Butterworth filter

Function Prototypes:
void bfhighpass (int npoles, float f3db, int n, float p[], float q[]);
void bflowpass (int npoles, float f3db, int n, float p[], float q[]);
void bfdesign (float fpass, float apass, float fstop, float astop,
int *npoles, float *f3db);

bfdesign:
Input:
fpass frequency in pass band at which amplitude is >= apass
apass amplitude in pass band corresponding to frequency fpass
fstop  frequency in stop band at which amplitude is <= astop
astop amplitude in stop band corresponding to frequency fstop

Output:
npoles number of poles
f3db frequency at which amplitude is sqrt(0.5) (-3 db)

bfhighpass and bflowpass:
Input:
npoles number of poles (and zeros); npoles>=0 is required
f3db 3 db frequency; nyquist = 0.5; 0.0<=f3db<=0.5 is required
n length of p and q
p array[n] to be filtered

Output:
q filtered array[n] (may be equivalent to p)

Notes:
(1) Nyquist frequency equals 0.5

(2) The following conditions must be true:
(0.0<fpass && fpass<0.5) &&
(0.0<fstop && fstop<0.5) &&
(fpass!=fstop) &&
(0.0<astop && astop<apass && apass<1.0)

(3) if (fpass<fstop)
```

```
bfdesign:
Butterworth filter:  compute number of poles and -3 db frequency
for a low-pass or high-pass filter, given a frequency response
constrained at two frequencies.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
COMPLEX - Functions to manipulate complex numbers

cadd add two complex numbers
csub subtract two complex numbers
cmul multiply two complex numbers
cdiv divide two complex numbers
cmplx make a complex number from two real numbers
conjg complex conjugate of a complex number
cneg negate a complex number
cinv invert a complex number
cwp_csqrt complex square root of a complex number
cwp_cexp complex exponential of a complex number
crmul multiply a complex number by a real number
rcabs real magnitude of a complex number

Structure:
typedef struct _complexStruct {  complex number
float r,i;
} complex;

Function Prototypes:
complex cadd (complex a, complex b);
complex csub (complex a, complex b);
complex cmul (complex a, complex b);
complex cdiv (complex a, complex b);
float rcabs (complex z);
complex cmplx (float re, float im);
complex conjg (complex z);
complex cneg (complex z);
complex cinv (complex z);
complex cwp_csqrt (complex z);
complex cwp_cexp (complex z);
complex crmul (complex a, float x);

Notes:
The function "rcabs" was originally called "fcabs". This produced
a collision on some systems so a new name was chosen.

Reference:
Adapted from Press et al, 1988, Numerical Recipes in C (Appendix E).

Author:  Dave Hale, Colorado School of Mines, 06/02/89
Modified:  Dave Hale, Colorado School of Mines, 04/26/90
```

```
Added function cinv().
```

```
COMPLEXD - Functions to manipulate double-precision complex numbers

dcadd add two dcomplex numbers
dcsub subtract two dcomplex numbers
dcmul multiply two dcomplex numbers
dcdiv divide two dcomplex numbers
dcmplx make a dcomplex number from two real numbers
dconjg dcomplex conjugate of a dcomplex number
dcneg negate a dcomplex number
dcinv invert a dcomplex number
dcsqrt dcomplex square root of a dcomplex number
dcexp dcomplex exponential of a dcomplex number
dcrmul multiply a dcomplex number by a real number
drcabs real magnitude of a dcomplex number

Structure:
typedef struct _dcomplexStruct {  dcomplex number
double r,i;
} dcomplex;

Function Prototypes:
dcomplex dcadd (dcomplex a, dcomplex b);
dcomplex dcsub (dcomplex a, dcomplex b);
dcomplex dcmul (dcomplex a, dcomplex b);
dcomplex dcdiv (dcomplex a, dcomplex b);
double drcabs (dcomplex z);
dcomplex dcmplx (double re, double im);
dcomplex dconjg (dcomplex z);
dcomplex dcneg (dcomplex z);
dcomplex dcinv (dcomplex z);
dcomplex dcsqrt (dcomplex z);
dcomplex dcexp (dcomplex z);
dcomplex dcrmul (dcomplex a, double x);

Notes:
The function "drcabs" was originally called "fcabs". This produced
a collision on some systems so a new name was chosen.

Reference:
Adapted from Press et al, 1988, Numerical Recipes in C (Appendix E).

Author:  Dave Hale, Colorado School of Mines, 06/02/89
Modified:  Dave Hale, Colorado School of Mines, 04/26/90
```

```
Added function dcinv().
```

```
COMPLEXF  - Subroutines to perform operations on complex numbers.
This set of functions complement the one in complex.c
of the CWP library

cipow raise a complex number to an integer power
crpow raise a complex number to a real power
rcpow raise a real number to a complex power
ccpow raise a complex number to a complex power
cwp_ccos compute the complex cosine of a complex angle
cwp_csin compute the complex sine of a complex angle
cwp_ccosh compute the complex hyperbolic cosine of a complex angle
cwp_csinh compute the complex hyperbolic sine of a complex angle
cwp_cexp1 compute the complex exponential of a complex number
cwp_clog compute the complex logarithm of a complex number
Function Prototypes:
complex cipow(complex a, int p);
complex crpow(complex a, float p);
complex rcpow(float a, complex p);
complex ccpow (complex a, complex p)
complex cwp_ccos(complex a);
complex cwp_csin(complex a);
complex cwp_ccosh(complex a);
complex cwp_csinh(complex a);
complex cwp_cexp1(complex a);
complex cwp_clog(complex a);
Credits:
Dave Hale, original version in C++
Gabriel Alvarez, translation to C
```

```
COMPLEXFD  - Subroutines to perform operations on double complex numbers.
This set of functions complement the one in complexd.c
of the CWP library

dcipow raise a double complex number to an integer power
dcrpow raise a double complex number to a real power
rdcpow raise a real number to a double complex power
dcdcpow raise a double complex number to a double complex power
dccos compute the double complex cosine of a double complex angle
dcsin compute the double complex sine of a double complex angle
dccosh compute the double complex hyperbolic cosine of a double complex angle
dcsinh compute the double complex hyperbolic sine of a double complex angle
dcexp1 compute the double complex exponential of a double complex number
dclog compute the double complex logarithm of a double complex number
Function Prototypes:
dcomplex dcipow(dcomplex a, int p);
dcomplex dcrpow(dcomplex a, float p);
dcomplex rdcpow(float a, dcomplex p);
dcomplex dcdcpow (dcomplex a, dcomplex p)
dcomplex dccos(dcomplex a);
dcomplex dcsin(dcomplex a);
dcomplex dccosh(dcomplex a);
dcomplex dcsinh(dcomplex a);
dcomplex dcexp1(dcomplex a);
dcomplex dclog(dcomplex a);
Credits:
Dave Hale, original version in C++
Gabriel Alvarez, translation to C
```

Conjugate Gradient routines -

simple_conj_gradient - simple conjugate gradient routine.

Function Prototypes:
void simple_conj_gradient(int n, float *x, int m, float *b, float **a, int niter);

References:
Claerbout, J. F. (1985), Conjugate gradients for beginners, SEP 44,
Stanford Exploration Project.
Shewchuk, Jonathan Richard (1994), An introduction to the conjugate
gradient method without the agonizing pain,
edition 1 1/4, School of Computer Science Carnegie Mellon University
Credits:
CWP: John Stockwell, May 2014
Based on the subroutine cg() which appeared in (Claerbout, 1985)

```
CONVOLUTION - Compute z = x convolved with y

convolve_cwp compute the convolution of two input vector arrays

Input:
lx length of x array
ifx sample index of first x
x array[lx] to be convolved with y
ly length of y array
ify sample index of first y
y array[ly] with which x is to be convolved
lz length of z array
ifz sample index of first z

Output:
z array[lz] containing x convolved with y

Function Prototype:
void convolve_cwp (int lx, int ifx, float *x, int ly, int ify, float *y,
int lz, int ifz, float *z);

Notes:
The operation z = x convolved with y is defined to be
           ifx+lx-1
    z[i] =   sum    x[j]*y[i-j]  ;  i = ifz,...,ifz+lz-1
            j=ifx
The x samples are contained in x[0], x[1], ..., x[lx-1]; likewise for
the y and z samples.  The sample indices of the first x, y, and z values
determine the location of the origin for each array.  For example, if
z is to be a weighted average of the nearest 5 samples of y, one might
use
...
x[0] = x[1] = x[2] = x[3] = x[4] = 1.0/5.0;
conv(5,-2,x,lx,0,y,ly,0,z);
...
In this example, the filter x is symmetric, with index of first sample = -2.

This function is optimized for architectures that can simultaneously perform
a multiply, add, and one load from memory; e.g., the IBM RISC System/6000.
Because, for each value of i, it accumulates the convolution sum z[i] in a
scalar, this function is not likely to be optimal for vector architectures.

Author:  Dave Hale, Colorado School of Mines, 11/23/91
```

```
CUBICSPLINE - Functions to compute CUBIC SPLINE interpolation coefficients

cakima compute cubic spline coefficients via Akima's method
  (continuous 1st derivatives, only)
cmonot compute cubic spline coefficients via the Fritsch-Carlson method
  (continuous 1st derivatives, only)
csplin compute cubic spline coefficients for interpolation
  (continuous 1st and 2nd derivatives)

chermite compute cubic spline coefficients via Hermite Polynomial
  (continuous 1st derivatives only)
Function Prototypes:
void cakima   (int n, float x[], float y[], float yd[][4]);
void cmonot   (int n, float x[], float y[], float yd[][4]);
void csplin   (int n, float x[], float y[], float yd[][4]);
void chermite (int n, float x[], float y[], float yd[][4]);

Input:
n number of samples
x   array[n] of monotonically increasing or decreasing abscissae
y array[n] of ordinates

Output:
yd array[n][4] of cubic interpolation coefficients (see notes)

Notes:
The computed cubic spline coefficients are as follows:
yd[i][0] = y(x[i])    (the value of y at x = x[i])
yd[i][1] = y'(x[i])   (the 1st derivative of y at x = x[i])
yd[i][2] = y''(x[i])  (the 2nd derivative of y at x = x[i])
yd[i][3] = y'''(x[i]) (the 3rd derivative of y at x = x[i])

To evaluate y(x) for x between x[i] and x[i+1] and h = x-x[i],
use the computed coefficients as follows:
y(x) = yd[i][0]+h*(yd[i][1]+h*(yd[i][2]/2.0+h*yd[i][3]/6.0))

Akima's method provides continuous 1st derivatives, but 2nd and
3rd derivatives are discontinuous.  Akima's method is not linear,
in that the interpolation of the sum of two functions is not the
same as the sum of the interpolations.

The Fritsch-Carlson method yields continuous 1st derivatives, but 2nd
and 3rd derivatives are discontinuous.  The method will yield a
```

monotonic interpolant for monotonic data. 1st derivatives are set
to zero wherever first divided differences change sign.

The method used by "csplin" yields continuous 1st and 2nd derivatives.

References:
See Akima, H., 1970, A new method for
interpolation and smooth curve fitting based on local procedures,
Journal of the ACM, v. 17, n. 4, p. 589-602.

For more information, see Fritsch, F. N., and Carlson, R. E., 1980,
Monotone piecewise cubic interpolation: SIAM J. Numer. Anal., v. 17,
n. 2, p. 238-246.
Also, see the book by Kahaner, D., Moler, C., and Nash, S., 1989,
Numerical Methods and Software, Prentice Hall. This function was
derived from SUBROUTINE PCHEZ contained on the diskette that comes
with the book.

For more general information on spline functions of all types see the book by:
Greville, T.N.E, 1969, Theory and Applications of Spline Functions,
Academic Press.

Author: Dave Hale, Colorado School of Mines c. 1989, 1990, 1991

```
DBLAS - Double precision Basic Linear Algebra subroutines
(adapted from LINPACK FORTRAN):

idamax return index of element with maximum absolute value
dasum return sum of absolute values
daxpy compute y[i] = a*x[i]+y[i]
dcopy copy x[i] to y[i] (i.e., set y[i] = x[i])
ddot return sum of x[i]*y[i] (i.e., return the dot product of x and y)
dnrm2 return square root of sum of squares of x[i]
dscal compute x[i] = a*x[i]
dswap swap x[i] and y[i]

Function Prototypes:
int idamax (int n, double *sx, int incx);
double dasum (int n, double *sx, int incx);
void daxpy (int n, double sa, double *sx, int incx, double *sy, int incy);
void dcopy (int n, double *sx, int incx, double *sy, int incy);
double ddot (int n, double *sx, int incx, double *sy, int incy);
double dnrm2 (int n, double *sx, int incx);
void dscal (int n, double sa, double *sx, int incx);
void dswap (int n, double *sx, int incx, double *sy, int incy);

idmax:
Input:
n number of elements in array
sx array[n] of elements
incx increment between elements

Returned:
index of element with maximum absolute value (idamax)

dasum:
Input:
n number of elements in array
sx array[n] of elements
incx increment between elements

Returned:
sum of absolute values (dasum)

daxpy:
Input:
n number of elements in arrays
```

```
sa the scalar multiplier
sx array[n] of elements to be scaled and added
incx increment between elements of sx
sy array[n] of elements to be added
incy increment between elements of sy

Output:
sy array[n] of accumulated elements

dcopy:
Input:
n number of elements in arrays
sx array[n] of elements to be copied
incx increment between elements of sx
incy increment between elements of sy

Output:
sy array[n] of copied elements

ddot:
Input:
n number of elements in arrays
sx array[n] of elements
incx increment between elements of sx
sy array[n] of elements
incy increment between elements of sy

Returned: dot product of the two arrays

dnrm2:
Input:
n number of elements in array
sx array[n] of elements
incx increment between elements

Returned: square root of sum of squares of x[i]

dscal:
Input:
n number of elements in array
sa the scalar multiplier
sx array[n] of elements
incx increment between elements
```

```
Output:
sx array[n] of scaled elements

Author:  Dave Hale, Colorado School of Mines, 10/01/89
```

```
DGE - Double precision Gaussian Elimination matrix subroutines  adapted
from LINPACK FORTRAN:

dgefa Gaussian elimination to obtain the LU factorization of a matrix.
dgeco Gaussian elimination to obtain the LU factorization and condition
number of a matrix.
dgesl Solve linear system Ax = b or A'x = b after LU factorization.

Function Prototypes:
void dgefa (double **a, int n, int *ipvt, int *info);
void dgeco (double **a, int n, int *ipvt, double *rcond, double *z);
void dgesl (double **a, int n, int *ipvt, double *b, int job);

dgfa:
Input:
a matrix[n][n] to be factored (see notes below)
n dimension of a

Output:
a matrix[n][n] factored (see notes below)
ipvt indices of pivot permutations (see notes below)
info index of last zero pivot (or -1 if no zero pivots)

dgeco:
Input:
a matrix[n][n] to be factored (see notes below)
n dimension of a

Output:
a matrix[n][n] factored (see notes below)
ipvt indices of pivot permutations (see notes below)
rcond reciprocal of condition number (see notes below)

Workspace:
z array[n]

dgesl:
Input:
a matrix[n][n] that has been LU factored (see notes below)
n dimension of a
ipvt indices of pivot permutations (see notes below)
b right-hand-side vector[n]
job =0 to solve Ax = b
```

```
=1 to solve A'x = b

Output:
b solution vector[n]

Notes:
These functions were adapted from LINPACK FORTRAN.  Because two-dimensional
arrays cannot be declared with variable dimensions in C, the matrix a
is actually a pointer to an array of pointers to floats, as declared
above and used below.

Elements of a are stored as follows:
a[0][0]    a[1][0]    a[2][0]   ... a[n-1][0]
a[0][1]    a[1][1]    a[2][1]   ... a[n-1][1]
a[0][2]    a[1][2]    a[2][2]   ... a[n-1][2]
.                                     .
.              .                      .
.                        .            .
.                                     .
a[0][n-1]  a[1][n-1]  a[2][n-1] ... a[n-1][n-1]

Both the factored matrix a and the pivot indices ipvt are required
to solve linear systems of equations via dgesl.


dgeco:
Given the reciprocal of the condition number, rcond, and the double
epsilon, DBL_EPSILON, the number of significant decimal digits, nsdd,
in the solution of a linear system of equations may be estimated by:
nsdd = (int)log10(rcond/DBL_EPSILON)

Author:  Dave Hale, Colorado School of Mines, 1989
```

```
DIFFERENTIATE - simple DIFFERENTIATOR codes

differentiate - 1D two point centered difference based derivative

Function Prototype:
void differentiate(int n, float h, float *f, float *fprime);
void ddifferentiate(int n, double h, double *f, double *fprime);

differentiate:
Input:
n number of samples
h sample rate
f array[n] of input values
Output:
fprime array[n], the derivative of f

fprime:
Notes:
This is a simple 2 point centered-difference differentiator.
The derivatives at the endpoints are computed via 2 point leading and
lagging differences.

Author: John Stockwell, CWP, 1994
```

```
PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place

npfa_d return valid n for complex-to-complex PFA
npfar_d return valid n for real-to-complex/complex-to-real PFA
npfao_d return optimal n for complex-to-complex PFA
npfaro_d return optimal n for real-to-complex/complex-to-real PFA
pfacc_d 1D PFA complex to complex
pfacr_d 1D PFA complex to real
pfarc_d 1D PFA real to complex
pfamcc_d multiple PFA complex to real
pfa2cc_d 2D PFA complex to complex
pfa2cr_d 2D PFA complex to real
pfa2rc_d 2D PFA real to complex

Function Prototypes:
int npfa_d (int nmin);
int npfao_d (int nmin, int nmax);
int npfar_d (int nmin);
int npfaro_d (int nmin, int nmax);
void pfacc_d (int isign, int n, real_complex z[]);
void pfacr_d (int isign, int n, real_complex cz[], real rz[]);
void pfarc_d (int isign, int n, real rz[], real_complex cz[]);
void pfamcc_d (int isign, int n, int nt, int k, int kt, real_complex z[]);
void pfa2cc_d (int isign, int idim, int n1, int n2, real_complex z[]);
void pfa2cr_d (int isign, int idim, int n1, int n2, real_complex cz[], real rz[]);
void pfa2rc_d (int isign, int idim, int n1, int n2, real rz[], real_complex cz[]);

npfa_d:
Input:
nmin lower bound on returned value (see notes below)

Returned: valid n for prime factor fft

npfao_d
Input:
nmin lower bound on returned value (see notes below)
nmax desired (but not guaranteed) upper bound on returned value

Returned: valid n for prime factor fft

npfar_d
Input:
nmin lower bound on returned value
```

```
Returned: valid n for real-to-real_complex/real_complex-to-real prime factor fft

npfaro_d:
Input:
nmin lower bound on returned value
nmax desired (but not guaranteed) upper bound on returned value

Returned: valid n for real-to-real_complex/real_complex-to-real prime factor fft

pfacc_d:
Input:
isign sign of isign is the sign of exponent in fourier kernel
n length of transform (see notes below)
z array[n] of real_complex numbers to be transformed in place

Output:
z array[n] of real_complex numbers transformed

pfacr_d:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
n           length of transform (see notes below)
cz          array[n/2+1] of real_complex values (may be equivalenced to rz)

Output:
rz          array[n] of real values (may be equivalenced to cz)

pfarc_d:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
n           length of transform; must be even (see notes below)
rz          array[n] of real values (may be equivalenced to cz)

Output:
cz          array[n/2+1] of real_complex values (may be equivalenced to rz)

pfamcc_d:
Input:
isign        sign of isign is the sign of exponent in fourier kernel
n            number of real_complex elements per transform (see notes below)
nt           number of transforms
k            stride in real_complex elements within transforms
```

```
kt           stride in real_complex elements between transforms
z            array of real_complex elements to be transformed in place

Output:
z array of real_complex elements transformed

pfa2cc_d:
Input:
isign        sign of isign is the sign of exponent in fourier kernel
idim         dimension to transform, either 1 or 2 (see notes)
n1           1st (fast) dimension of array to be transformed (see notes)
n2           2nd (slow) dimension of array to be transformed (see notes)
z            array[n2][n1] of real_complex elements to be transformed in place

Output:
z array[n2][n1] of real_complex elements transformed

pfa2cr_d:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
idim        dimension to transform, which must be either 1 or 2 (see notes)
n1          1st (fast) dimension of array to be transformed (see notes)
n2          2nd (slow) dimension of array to be transformed (see notes)
cz          array of real_complex values (may be equivalenced to rz)

Output:
rz          array of real values (may be equivalenced to cz)

pfa2rc_d:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
idim        dimension to transform, which must be either 1 or 2 (see notes)
n1          1st (fast) dimension of array to be transformed (see notes)
n2          2nd (slow) dimension of array to be transformed (see notes)
rz          array of real values (may be equivalenced to cz)

Output:
cz          array of real_complex values (may be equivalenced to rz)

Notes:
Table of valid n and cost for prime factor fft.  For each n, cost
was estimated to be the inverse of the number of ffts done in 1 sec
on an IBM RISC System/6000 Model 320H, by Dave Hale, 08/04/91.
```

```
(Redone by Jack Cohen for 15 sec to rebuild NTAB table on advice of
David and Gregory Chudnovsky, 05/03/94).
Cost estimates are least accurate for very small n.  An alternative method
for estimating cost would be to count multiplies and adds, but this
method fails to account for the overlapping of multiplies and adds
that is possible on some computers, such as the IBM RS/6000 family.

npfa_d:
The returned n will be composed of mutually prime factors from
the set {2,3,4,5,7,8,9,11,13,16}.  Because n cannot exceed
720720 = 5*7*9*11*13*16, 720720 is returned if nmin exceeds 720720.

npfao_d:
The returned n will be composed of mutually prime factors from
the set {2,3,4,5,7,8,9,11,13,16}.  Because n cannot exceed
720720 = 5*7*9*11*13*16, 720720 is returned if nmin exceeds 720720.
If nmin does not exceed 720720, then the returned n will not be
less than nmin.  The optimal n is chosen to minimize the estimated
cost of performing the fft, while satisfying the constraint, if
possible, that n not exceed nmax.

npfar and npfaro:
Current implemenations of real-to-real_complex and real_complex-to-real prime
factor ffts require that the transform length n be even and that n/2
be a valid length for a real_complex-to-real_complex prime factor fft.  The
value returned by npfar satisfies these conditions.  Also, see notes
for npfa.

pfacc:
n must be factorable into mutually prime factors taken
from the set {2,3,4,5,7,8,9,11,13,16}.  in other words,
n = 2**p * 3**q * 5**r * 7**s * 11**t * 13**u
where
0 <= p <= 4,  0 <= q <= 2,  0 <= r,s,t,u <= 1
is required for pfa to yield meaningful results.  this
restriction implies that n is restricted to the range
1 <= n <= 720720 (= 5*7*9*11*13*16)

pfacr:
Because pfacr uses pfacc to do most of the work, n must be even
and n/2 must be a valid length for pfacc.  The simplest way to
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.
```

```
pfarc:
Because pfarc uses pfacc to do most of the work, n must be even
and n/2 must be a valid length for pfacc.  The simplest way to
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.

pfamcc:
To perform a two-dimensional transform of an n1 by n2 real_complex array
(assuming that both n1 and n2 are valid "n"), stored with n1 fast
and n2 slow:
    pfamcc(isign,n1,n2,1,n1,z); (to transform 1st dimension)
    pfamcc(isign,n2,n1,n1,1,z); (to transform 2nd dimension)

pfa2cc:
Only one (either the 1st or 2nd) dimension of the 2-D array is transformed.

If idim equals 1, then n2 transforms of n1 real_complex elements are performed;
else, if idim equals 2, then n1 transforms of n2 real_complex elements are
performed.

Although z appears in the argument list as a one-dimensional array,
z may be viewed as an n1 by n2 two-dimensional array:  z[n2][n1].

Valid n is computed via the "np" subroutines.

To perform a two-dimensional transform of an n1 by n2 real_complex array
(assuming that both n1 and n2 are valid "n"), stored with n1 fast
and n2 slow:  pfa2cc(isign,1,n1,n2,z);  pfa2cc(isign,2,n1,n2,z);

pfa2cr:
If idim equals 1, then n2 transforms of n1/2+1 real_complex elements to n1 real
elements are performed; else, if idim equals 2, then n1 transforms of n2/2+1
real_complex elements to n2 real elements are performed.

Although rz appears in the argument list as a one-dimensional array,
rz may be viewed as an n1 by n2 two-dimensional array:  rz[n2][n1].
Likewise, depending on idim, cz may be viewed as either an n1/2+1 by
n2 or an n1 by n2/2+1 two-dimensional array of real_complex elements.

Let n denote the transform length, either n1 or n2, depending on idim.
Because pfa2rc uses pfa2cc to do most of the work, n must be even
and n/2 must be a valid length for pfa2cc.  The simplest way to
```

```
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.

pfa2rc:
If idim equals 1, then n2 transforms of n1 real elements to n1/2+1 real_complex
elements are performed; else, if idim equals 2, then n1 transforms of n2
real elements to n2/2+1 real_complex elements are performed.

Although rz appears in the argument list as a one-dimensional array,
rz may be viewed as an n1 by n2 two-dimensional array:  rz[n2][n1].
Likewise, depending on idim, cz may be viewed as either an n1/2+1 by
n2 or an n1 by n2/2+1 two-dimensional array of real_complex elements.

Let n denote the transform length, either n1 or n2, depending on idim.
Because pfa2rc uses pfa2cc to do most of the work, n must be even
and n/2 must be a valid length for pfa2cc.  The simplest way to
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.

References:
Temperton, C., 1985, Implementation of a self-sorting
in-place prime factor fft algorithm:  Journal of
Computational Physics, v. 58, p. 283-299.

Temperton, C., 1988, A new set of minimum-add rotated
rotated dft modules: Journal of Computational Physics,
v. 75, p. 190-198.

Press et al, 1988, Numerical Recipes in C, p. 417.

Author:  Dave Hale, Colorado School of Mines, 04/27/89
Revised by Baoniu Han to handle double precision. 12/14/98
```

CWP_Exit - exit subroutine for CWP/SU codes

```
FRANNOR - functions to generate a pseudo-random float normally distributed
with N(0,1); i.e., with zero mean and unit variance.

frannor return a normally distributed random float
srannor seed random number generator for normal distribution

Function Prototypes:
float frannor (void);
void srannor (int seed);

frannor:
Input: (none)
Returned: normally distributed random float

srannor:
Input:
seed different seeds yield different sequences of random numbers.

Notes:
Adapted from subroutine rnor in Kahaner,  Moler and Nash (1988)
which in turn was based on  an algorithm by
Marsaglia and Tsang (1984).

References:
Numerical Methods and Software", D.
Kahaner, C. Moler, S. Nash, Prentice Hall, 1988.

Marsaglia G. and Tsang, W. W., 1984,
A fast, easily implemented method for sampling from decreasing or symmetric
unimodal density functions:  SIAM J. Sci. Stat. Comput., v. 5, no. 2,
p. 349-359.

Author:  Dave Hale, Colorado School of Mines, 01/21/89
```

```
FRANUNI - Functions to generate a pseudo-random float uniformly distributed
on [0,1); i.e., between 0.0 (inclusive) and 1.0 (exclusive)

franuni return a random float
sranuni seed random number generator

Function Prototypes:
float franuni (void);
void sranuni (int seed);

franuni:
Input: (none)
Returned: pseudo-random float

sranuni:
seed different seeds yield different sequences of random numbers.

Notes:
Adapted from subroutine uni in Kahaner, Moler, and Nash (1988).
This book references a set of unpublished notes by
Marsaglia.

According to the reference, this random
number generator "passes all known tests and has a period that is ...
approximately 10^19".

References:
Numerical Methods and Software", D. Kahaner, C. Moler, S. Nash,
Prentice Hall, 1988.

Marsaglia G., "Comments on the perfect uniform random number generator
Unpublished notes, Wash S. U.

Author:  Dave Hale, Colorado School of Mines, 12/30/89
```

```
HANKEL - Functions to compute discrete Hankel transforms

hankelalloc allocate and return a pointer to a Hankel transformer
hankelfree  free a Hankel transformer
hankel0 compute the zeroth-order Hankel transform
hankel1 compute the first-order Hankel transform

Function Prototypes:
void *hankelalloc (int nfft);
void hankelfree (void *ht);
void hankel0 (void *ht, float f[], float h[]);
void hankel1 (void *ht, float f[], float h[]);

hankelalloc:
Input:
nfft valid length for real to complex fft  (see notes below)

Returned:
pointer to Hankel transformer

hankelfree:
Input:
ht pointer to Hankel transformer (as returned by hankelalloc)

hankel0:
Input:
ht pointer to Hankel transformer (as returned by hankelalloc)
f array[nfft/2+1] to be transformed

Output:
h array[nfft/2+1] transformed

hankel1:
Input:
ht pointer to Hankel transformer (as returned by hankelalloc)
f array[nfft/2+1] to be transformed

Output:
h array[nfft/2+1] transformed

Notes:
The zeroth-order Hankel transform is defined by:
```

```
        Infinity
h0(k) = Integral dr r j0(k*r) f(r)
   0
```

where j0 denotes the zeroth-order Bessel function.

The first-order Hankel transform is defined by:

```
        Infinity
h1(k) = Integral dr r j1(k*r) f(r)
   0
```

where j1 denotes the first-order Bessel function.

The Hankel transform is its own inverse.

The Hankel transform is computed by an Abel transform followed by
a Fourier transform.

References:
Hansen, E. W., 1985, Fast Hankel transform algorithm: IEEE Trans. on
Acoustics, Speech and Signal Processing, v. ASSP-33, n. 3, p. 666-671.
(Beware of several errors in the equations in this paper!)

Authors: Dave Hale, Colorado School of Mines, 06/04/90

```
Hartley - routines for fast Hartley transform

srfht -  in-place FHT using the split-radix algorithm
dsrfht - in-place FHT using the split-radix algorithm (double precision)
r4fht -in-place FHT using the radix-4 algorithm
nextpow2 - find m such that n=2^m via lookup table
nextpow4 - find m such that n=4^m via lookup table
srfht - split-radix in-place FHT

Function Prototypes:

void srfht(int *n, int *m, float *f);
void dsrfht(int *n, int *m, double *f);
void r4fht(int n, int m, float *f);
int nextpow2(int n);
int nextpow4(int n);



Notes:
srfht -  in-place FHT using the split-radix algorithm
(single precision)
does not include division by n
n and m are not modified
usage:  srfht(int *n, int *m, float *f)
n  length of input sequence n=2^m
m  exponent such that n=2^m
   f  input sequence to FHT

dsrfht - in-place FHT using the split-radix algorithm
(double precision)
does not include division by n
n and m are not modified
usage:  srfht(int *n, int *m, double *f)
n  length of input sequence n=2^m
m  exponent such that n=2^m
   f  input sequence to FHT

r4fht -in-place FHT using the radix-4 algorithm
does not include division by n
usage:  r4fht(int n, int m, float *f)
n  length of input sequence n=4^m
m  exponent such that n=4^m
```

```
   f  input sequence to FHT

nextpow2 - find m such that n=2^m via lookup table
max(n)=2^24

nextpow4 - find m such that n=4^m via lookup table
max(n)=4^12

srfht - split-radix in-place FHT
  n length of input sequence n=2^m
  m exponent such that n=2^m
    f input sequence to FHT
   Reference:
       Sorensen, H. V., Jones, D. L., Burrus, C. S, & Heideman, M. T.,
       1985, On computing the Hartley transform: IEEE Trans. Acoust.,
       Speech, and Signal Proc., v. ASSP-33, no. 4, p. 1231-1238.

possible improvements:
find optimum length for FHT (nfhto)
use bit shift operators in r4fht
  Credits: CENPET: Werner M. Heigl, April 2006
```

```
HILBERT - Compute Hilbert transform y of x

hilbert compute the Hilbert transform

Function Prototype:
void hilbert (int n, float x[], float y[]);

Input:
n length of x and y
x array[n] to be Hilbert transformed

Output:
y array[n] containing Hilbert transform of x

Notes:
The Hilbert transform is computed by convolving x with a
windowed (approximate) version of the ideal Hilbert transformer.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

HOLBERG1D - Compute coefficients of Holberg's 1st derivative filter

holberg1d comput the coefficients of Holberg's 1st derivative filter

Function Prototype:
void holbergd1 (float e, int n, float d[]);

Input:
e maximum relative error in group velocity
n number of coefficients in filter (must be 2, 4, 6, or 8)

Output:
d array[n] of coefficients

Notes:
Coefficients are output in a form suitable for convolution. The
derivative is centered halfway between coefficients d[n/2-1] and d[n/2].

Coefficients are computed via the power series method of Kindelan et al.,
1990, On the construction and efficiency of staggered numerical
differentiators for the wave equation: Geophysics 55, 107-110.
See also, Holberg, 1987, Computational aspects of the choice of
operator and sampling interval for numerical differentiation in
large-scale simulation of wave phenomena: Geophys. Prosp., 35, 629-655

Reference:
Kindelan et al., 1990,
On the construction and efficiency of staggered numerical
differentiators for the wave equation: Geophysics 55, 107-110.

See also, Holberg, 1987, Computational aspects of the choice of
operator and sampling interval for numerical differentiation in
large-scale simulation of wave phenomena: Geophys. Prosp., 35, 629-655

Author: Dave Hale, Colorado School of Mines, 06/06/91

```
INTCUB - evaluate y(x), y'(x), y''(x), ... via piecewise cubic interpolation

intcub evaluate y(x), y'(x), y''(x), ... via piecewise spline
interpolation

Function Prototype:
void intcub (int ideriv, int nin, float xin[], float ydin[][4],

Input:
ideriv =0 if y(x) desired; =1 if y'(x) desired, ...
nin length of xin and ydin arrays
xin array[nin] of monotonically increasing or decreasing x values
ydin array[nin][4] of y(x), y'(x), y''(x), and y'''(x)
nout length of xout and yout arrays
xout array[nout] of x values at which to evaluate y(x), y'(x), ...

Output:
yout array[nout] of y(x), y'(x), ... values

Notes:
xin values must be monotonically increasing or decreasing.

Extrapolation of the function y(x) for xout values outside the range
spanned by the xin values is performed using the derivatives in
ydin[0][0:3] or ydin[nin-1][0:3], depending on whether xout is closest
to xin[0] or xin[nin-1], respectively.

Reference:
See: Greville, T. N., Theory and Application of Spline Functions, Academic Press
for a general discussion of spline functions.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
INTL2B - bilinear interpolation of a 2-D array of bytes

intl2b bilinear interpolation of a 2-D array of bytes

Function Prototype:
void intl2b (int nxin, float dxin, float fxin,
int nyin, float dyin, float fyin, unsigned char *zin,
int nxout, float dxout, float fxout,
int nyout, float dyout, float fyout, unsigned char *zout);

Input:
nxin number of x samples input (fast dimension of zin)
dxin x sampling interval input
fxin first x sample input
nyin number of y samples input (slow dimension of zin)
dyin y sampling interval input
fyin first y sample input
zin array[nyin][nxin] of input samples (see notes)
nxout number of x samples output (fast dimension of zout)
dxout x sampling interval output
fxout first x sample output
nyout number of y samples output (slow dimension of zout)
dyout y sampling interval output
fyout first y sample output

Output:
zout array[nyout][nxout] of output samples (see notes)

Notes:
The arrays zin and zout must passed as pointers to the first element of
a two-dimensional contiguous array of unsigned char values.

Constant extrapolation of zin is used to compute zout for
output x and y outside the range of input x and y.

For efficiency, this function builds a table of interpolation
coefficents pre-multiplied by byte values.  To keep the table
reasonably small, the interpolation does not distinguish
between x and y values that differ by less than dxin/ICMAX
and dyin/ICMAX, respectively, where ICMAX is a parameter
defined above.

Author:  Dave Hale, Colorado School of Mines, c. 1989-1991.
```

```
INTLINC - evaluate complex y(x) via linear interpolation of y(x[0]), y(x[1]), ...

Function Prototype:
void intlinc (int nin, float xin[], complex yin[], complex yinl, complex yinr,
int nout, float xout[], complex yout[]);

Input:
nin length of xin and yin arrays
xin array[nin] of monotonically increasing or decreasing x values
yin array[nin] of input y(x) values
yinl value used to extraplate y(x) to left of input yin values
yinr value used to extraplate y(x) to right of input yin values
nout length of xout and yout arrays
xout array[nout] of x values at which to evaluate y(x)

Output:
yout array[nout] of linearly interpolated y(x) values

Notes:
xin values must be monotonically increasing or decreasing.

Extrapolation of the function y(x) for xout values outside the range
spanned by the xin values in performed as follows:

For monotonically increasing xin values,
yout=yinl if xout<xin[0], and yout=yinr if xout>xin[nin-1].

For monotonically decreasing xin values,
yout=yinl if xout>xin[0], and yout=yinr if xout<xin[nin-1].

If nin==1, then the monotonically increasing case is used.
/
```

```
INTLIN - evaluate y(x) via linear interpolation of y(x[0]), y(x[1]), ...

intlin evaluate y(x) via linear interpolation of y(x[0]), y(x[1]), ...

Function Prototype:
void intlin (int nin, float xin[], float yin[], float yinl, float yinr,
int nout, float xout[], float yout[]);

Input:
nin length of xin and yin arrays
xin array[nin] of monotonically increasing or decreasing x values
yin array[nin] of input y(x) values
yinl value used to extraplate y(x) to left of input yin values
yinr value used to extraplate y(x) to right of input yin values
nout length of xout and yout arrays
xout array[nout] of x values at which to evaluate y(x)

Output:
yout array[nout] of linearly interpolated y(x) values

Notes:
xin values must be monotonically increasing or decreasing.

Extrapolation of the function y(x) for xout values outside the range
spanned by the xin values in performed as follows:

For monotonically increasing xin values,
yout=yinl if xout<xin[0], and yout=yinr if xout>xin[nin-1].

For monotonically decreasing xin values,
yout=yinl if xout>xin[0], and yout=yinr if xout<xin[nin-1].

If nin==1, then the monotonically increasing case is used.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
INTLIRR2B - bilinear interpolation of a 2-D array of bytes

intlirr2b bilinear interpolation of a 2-D array of bytes
                where x spacings are irregular

Function Prototype:
void intlirr2b (int nxin, float *xin,
int nyin, float dyin, float fyin, unsigned char *zin,
int nxout, float dxout, float fxout,
int nyout, float dyout, float fyout, unsigned char *zout);

Input:
nxin number of x samples input (fast dimension of zin)
xin array of (irregular) input x-coordinates
nyin number of y samples input (slow dimension of zin)
dyin y sampling interval input
fyin first y sample input
zin array[nyin][nxin] of input samples (see notes)
nxout number of x samples output (fast dimension of zout)
dxout x sampling interval output
fxout first x sample output
nyout number of y samples output (slow dimension of zout)
dyout y sampling interval output
fyout first y sample output

Output:
zout array[nyout][nxout] of output samples (see notes)

Notes:
The arrays zin and zout must passed as pointers to the first element of
a two-dimensional contiguous array of unsigned char values.

Constant extrapolation of zin is used to compute zout for
output x and y outside the range of input x and y.

For efficiency, this function builds a table of interpolation
coefficents pre-multiplied by byte values.  To keep the table
reasonably small, the interpolation does not distinguish
between x and y values that differ by less than dxin/ICMAX
and dyin/ICMAX, respectively, where ICMAX is a parameter
defined above.

Author:  Dave Hale, Colorado School of Mines, c. 1989-1991.
```

```
INTSINC8 - Functions to interpolate uniformly-sampled data via 8-coeff. sinc
approximations:

ints8c interpolation of a uniformly-sampled complex function y(x) via an
 8-coefficient sinc approximation.
ints8r Interpolation of a uniformly-sampled real function y(x) via a
table of 8-coefficient sinc approximations

Function Prototypes:
void ints8c (int nxin, float dxin, float fxin, complex yin[],
complex yinl, complex yinr, int nxout, float xout[], complex yout[]);
void ints8r (int nxin, float dxin, float fxin, float yin[],
float yinl, float yinr, int nxout, float xout[], float yout[]);

Input:
nxin number of x values at which y(x) is input
dxin x sampling interval for input y(x)
fxin x value of first sample input
yin array[nxin] of input y(x) values:  yin[0] = y(fxin), etc.
yinl value used to extrapolate yin values to left of yin[0]
yinr value used to extrapolate yin values to right of yin[nxin-1]
nxout number of x values a which y(x) is output
xout array[nxout] of x values at which y(x) is output

Output:
yout array[nxout] of output y(x):  yout[0] = y(xout[0]), etc.

Notes:
Because extrapolation of the input function y(x) is defined by the
left and right values yinl and yinr, the xout values are not restricted
to lie within the range of sample locations defined by nxin, dxin, and
fxin.

The maximum error for frequiencies less than 0.6 nyquist is less than
one percent.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
INTTABLE8 -  Interpolation of a uniformly-sampled complex function y(x)
via a table of 8-coefficient interpolators

intt8c interpolation of a uniformly-sampled complex function y(x)
via a table of 8-coefficient interpolators
intt8r interpolation of a uniformly-sampled real function y(x) via a
table of 8-coefficient interpolators

Function Prototype:
void intt8c (int ntable, float table[][8],
int nxin, float dxin, float fxin, complex yin[],
complex yinl, complex yinr, int nxout, float xout[], complex yout[]);
void intt8r (int ntable, float table[][8],
int nxin, float dxin, float fxin, float yin[],
float yinl, float yinr, int nxout, float xout[], float yout[]);

Input:
ntable number of tabulated interpolation operators; ntable>=2
table array of tabulated 8-point interpolation operators
nxin number of x values at which y(x) is input
dxin x sampling interval for input y(x)
fxin x value of first sample input
yin array of input y(x) values:  yin[0] = y(fxin), etc.
yinl value used to extrapolate yin values to left of yin[0]
yinr value used to extrapolate yin values to right of yin[nxin-1]
nxout number of x values a which y(x) is output
xout array of x values at which y(x) is output

Output:
yout array of output y(x) values:  yout[0] = y(xout[0]), etc.

NOTES:
ntable must not be less than 2.

The table of interpolation operators must be as follows:

Let d be the distance, expressed as a fraction of dxin, from a particular
xout value to the sampled location xin just to the left of xout.  Then,
for d = 0.0,

table[0][0:7] = 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0

are the weights applied to the 8 input samples nearest xout.
```

Likewise, for d = 1.0,

table[ntable-1][0:7] = 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0

are the weights applied to the 8 input samples nearest xout. In general, for d = (float)itable/(float)(ntable-1), table[itable][0:7] are the weights applied to the 8 input samples nearest xout. If the actual sample distance d does not exactly equal one of the values for which interpolators are tabulated, then the interpolator corresponding to the nearest value of d is used.

Because extrapolation of the input function y(x) is defined by the left and right values yinl and yinr, the xout values are not restricted to lie within the range of sample locations defined by nxin, dxin, and fxin.

AUTHOR: Dave Hale, Colorado School of Mines, 06/02/89

```
LINEAR_REGRESSION - Compute linear regression of (y1,y2,...,yn) against
(x1,x2,...,xn)
Function Prototype:
void linear_regression(float *y, float *x, int n, float coeff[4]);
Input:
y array of y values
x array of x values
n length of y and x arrays
Output:
coeff[4] where:

coeff[0] slope of best fit line
coeff[1] intercept of best fit line
coeff[2] correlation coefficient of fit (1 = perfect) [dimensionless]
coeff[3] standard error of fit (0 = perfect) [dimensions of y]
Notes:

y(x)
    |      *  .    fit is  y(x) = a x + b
    |       .
    |     .  *
    | * .
    | . *
     ------------------- x

         n Sum[x*y] - Sum[x]*Sum[y]
     a = --------------------------
         n Sum[x*x] - Sum[x]*Sum[x]

         Sum[y] - a*Sum[x]
     b = -----------------
                n

     cc = std definition

     se = std definition

Author:  Chris Liner, UTulsa, 11/16/03
```

```
maxmin - subroutines that pertain to maximum and minimum values
min_index - find the value of the index where an array is a minimum

function prototypes:
int max_index(int n, float *a,int inc);
int min_index(int n, float *a,int inc);

Author: Balasz Nemeth: Potash Corporation, given to CWP in 2008
```

```
MKDIFF - Make an n-th order DIFFerentiator via Taylor's series method.

mkdiff make discrete Taylor series approximation to n'th derivative.

Function Prototype:
void mkdiff (int n, float a, float h, int l, int m, float d[]);

Input:
n order of desired derivative (n>=0 && n<=m-l is required)
a fractional distance from integer sampling index (see notes)
h sampling interval
l sampling index of first coefficient (see notes below)
m sampling index of last coefficient (see notes below)

Output:
d array[m-l+1] of coefficients for n'th order differentiator

Notes:
The abscissae x of a sampled function f(x) can always be expressed as
x = (j+a)*h, where j is an integer, a is a fraction, and h is the
sampling interval.  To approximate the n'th order derivative fn(x)
of the sampled function f(x) at x = (j+a)*h, use the m-l+1 coefficients
in the output array d[] as follows:

fn(x) = d[0]*f(j-l) + d[1]*f(j-l-1) +...+ d[m-l]*f(j-m)

i.e., convolve the coefficients in d with the samples in f.

m-l+1 (the number of coefficients) must not be greater than the
NCMAX parameter specified below.

For best approximations,
when n is even, use a = 0.0, l = -m
when n is odd, use a = 0.5, l = -m-1

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
MKHDIFF - Compute filter approximating the bandlimited HalF-DIFFerentiator.

mkhdiff - Compute filter approximating the bandlimited half-differentiator.

Function Prototype:
void mkhdiff (float h, int l, float d[]);

Input:
h sampling interval
l half-length of half-differentiator (length = 1+2*l is odd)

Output:
d array[1+2*l] of coefficients for half-differentiator

Notes:
The half-differentiator is defined by

  pi
    d[l+j] = sqrt(1/h)/(2pi) * integral dw sqrt(-iw)*exp(-iwj)
 -pi

  pi
           = sqrt(2/h)/(2pi) * integral dw sqrt(w)*(cos(wj)-sin(wj))
  0

    for j = -l, -l+1, ... , l.

An alternative definition is that f'(j) = d(j)*d(j)*f(j), where
f'(j) denotes the derivative of a sampled function f(j) and *
denotes a convolution sum.

The half-derivative g(j) of f(j) may be computed by the following sum:

g(j) = d[0]*f(j+l) + d[1]*f(j+l-1) + ... + d[2*l]*f(j-l)

The integral over frequency is evaluated numerically using Simpson's
method.  Although the Filon method of numerical integration is more
appropriate for this integral, the truncation of d[l+j] for |j| > l
is probably the greatest source of error.  In any case, d[l+j] is
cosine-tapered to reduce these truncation errors.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
MKSINC - Compute least-squares optimal sinc interpolation coefficients.

mksinc Compute least-squares optimal sinc interpolation coefficients.

Function Prototype:
void mksinc (float d, int lsinc, float sinc[]);

Input:
d fractional distance to interpolation point; 0.0<=d<=1.0
lsinc length of sinc approximation; lsinc%2==0 and lsinc<=20

Output:
sinc array[lsinc] containing interpolation coefficients

Notes:
The coefficients are a least-squares-best approximation to the ideal
sinc function for frequencies from zero up to a computed maximum
frequency.  For a given interpolator length, lsinc, mksinc computes
the maximum frequency, fmax (expressed as a fraction of the nyquist
frequency), using the following empirically derived relation (from
a Western Geophysical Technical Memorandum by Ken Larner):

fmax = min(0.066+0.265*log(lsinc),1.0)

Note that fmax increases as lsinc increases, up to a maximum of 1.0.
Use the coefficients to interpolate a uniformly-sampled function y(i)
as follows:

            lsinc-1
    y(i+d) =  sum  sinc[j]*y(i+j+1-lsinc/2)
              j=0

Interpolation error is greatest for d=0.5, but for frequencies less
than fmax, the error should be less than 1.0 percent.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
MNEWT - Solve non-linear system of equations f(x) = 0 via Newton's method

mnewt Solve non-linear system of equations f(x) = 0 via Newton's method

Function Prototype:
int mnewt (int maxiter, float ftol, float dxtol, int n, float *x, void *aux,
void (*fdfdx)(int n, float *x, float *f, float **dfdx, void *aux));

Input:
maxiter maximum number of iterations
ftol converged when sum of absolute values of f less than ftol
dxtol converged when sum of absolute values of dx less than dxtol
n number of equations
x array[n] containing initial guess of solution
aux pointer to auxiliary parameters to be passed to fdfdx
fdfdx pointer to function to evaluate f(x) and f'(x)

Output:
x array[n] containing solution

Returned: number of iterations; -1 if failed to converge in maxiter

Input to the user-supplied function fdfdx:
n number of equations
x array[n] of x0, x1, ...
aux pointer to auxiliary variables required by fdfdx.

Output from the user-supplied function fdfdx:
f array[n] of f0(x), f1(x), ...
dfdx array[n][n] of f'(x);  dfdx[j][i] = dfi/dxj

Author:  Dave Hale, Colorado School of Mines, 06/06/91
```

```
ORTHPOLYNOMIALS - compute ORTHogonal POLYNOMIALS

hermite_n_polynomial:  Compute n-th order generalized Hermite polynomial.

hermite_n_polynomial:

Input:
h0 array of size nt holding H_{n-1}
h1 array of size nt holding H_{n}
t array of size nt holding time vector
nt size of arrays, no. of samples
n order of polynomial
sigma variance
Output:
h array of size nt holding H_{n+1}
Notes: Note that n in the function call is the order of the derivative and
        j in the code below is the n in the recurrence relation
Copyright (c) 2007 by the Society of Exploration Geophysicists.
For more information, go to http://software.seg.org/2007/0004 .
You must read and accept usage terms at:
http://software.seg.org/disclaimer.txt before use.
Author: Werner M. Heigl, Apache Corporation, E&P Technology, December 2006



include "cwp.h"



void
hermite_n_polynomial(double *h, double *h0, double *h1, double *t, int nt, int n, doubl
  hermite_n_polynomial:  Compute n-th order generalized Hermite polynomial.
Input:
h0 array of size nt holding H_{n-1}
h1 array of size nt holding H_{n}
t array of size nt holding time vector
nt size of arrays, no. of samples
n order of polynomial
sigma variance
Output:
h array of size nt holding H_{n+1}
Note: Note that n in the function call is the order of the derivative and
        j in the code below is the n in the recurrence relation
```

```
Copyright (c) 2007 by the Society of Exploration Geophysicists.
For more information, go to http://software.seg.org/2007/0004 .
You must read and accept usage terms at:
http://software.seg.org/disclaimer.txt before use.
Author: Werner M. Heigl, Apache Corporation, E&P Technology, December 2006
{
int i; /* loop variable
int j=1; /* recurrence counter

/* as long as necessary use recurrence relation
do {
/* current instance of recurrence relation
for (i = 0; i < nt; ++i)
h[i] = ( t[i] * h1[i] - j * h0[i] ) / sigma;

/* update inputs to recurrence relation
memcpy((void *) h0, (const void *) h1, DSIZE * nt);
memcpy((void *) h1, (const void *) h , DSIZE * nt);

/* update counter
++j;

} while (j < n);

}
```

```
PFAFFT - Functions to perform Prime Factor (PFA) FFT's, in place

npfa return valid n for complex-to-complex PFA
npfar return valid n for real-to-complex/complex-to-real PFA
npfao return optimal n for complex-to-complex PFA
npfaro return optimal n for real-to-complex/complex-to-real PFA
pfacc 1D PFA complex to complex
pfacr 1D PFA complex to real
pfarc 1D PFA real to complex
pfamcc multiple PFA complex to real
pfa2cc 2D PFA complex to complex
pfa2cr 2D PFA complex to real
pfa2rc 2D PFA real to complex

Function Prototypes:
int npfa (int nmin);
int npfao (int nmin, int nmax);
int npfar (int nmin);
int npfaro (int nmin, int nmax);
void pfacc (int isign, int n, complex z[]);
void pfacr (int isign, int n, complex cz[], float rz[]);
void pfarc (int isign, int n, float rz[], complex cz[]);
void pfamcc (int isign, int n, int nt, int k, int kt, complex z[]);
void pfa2cc (int isign, int idim, int n1, int n2, complex z[]);
void pfa2cr (int isign, int idim, int n1, int n2, complex cz[], float rz[]);
void pfa2rc (int isign, int idim, int n1, int n2, float rz[], complex cz[]);

npfa:
Input:
nmin lower bound on returned value (see notes below)

Returned: valid n for prime factor fft

npfao
Input:
nmin lower bound on returned value (see notes below)
nmax desired (but not guaranteed) upper bound on returned value

Returned: valid n for prime factor fft

npfar
Input:
nmin lower bound on returned value
```

```
Returned: valid n for real-to-complex/complex-to-real prime factor fft

npfaro:
Input:
nmin lower bound on returned value
nmax desired (but not guaranteed) upper bound on returned value

Returned: valid n for real-to-complex/complex-to-real prime factor fft

pfacc:
Input:
isign sign of isign is the sign of exponent in fourier kernel
n length of transform (see notes below)
z array[n] of complex numbers to be transformed in place

Output:
z array[n] of complex numbers transformed

pfacr:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
n           length of transform (see notes below)
cz          array[n/2+1] of complex values (may be equivalenced to rz)

Output:
rz          array[n] of real values (may be equivalenced to cz)

pfarc:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
n           length of transform; must be even (see notes below)
rz          array[n] of real values (may be equivalenced to cz)

Output:
cz          array[n/2+1] of complex values (may be equivalenced to rz)

pfamcc:
Input:
isign        sign of isign is the sign of exponent in fourier kernel
n            number of complex elements per transform (see notes below)
nt           number of transforms
k            stride in complex elements within transforms
```

```
kt           stride in complex elements between transforms
z            array of complex elements to be transformed in place

Output:
z array of complex elements transformed

pfa2cc:
Input:
isign        sign of isign is the sign of exponent in fourier kernel
idim         dimension to transform, either 1 or 2 (see notes)
n1           1st (fast) dimension of array to be transformed (see notes)
n2           2nd (slow) dimension of array to be transformed (see notes)
z            array[n2][n1] of complex elements to be transformed in place

Output:
z array[n2][n1] of complex elements transformed

pfa2cr:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
idim        dimension to transform, which must be either 1 or 2 (see notes)
n1          1st (fast) dimension of array to be transformed (see notes)
n2          2nd (slow) dimension of array to be transformed (see notes)
cz          array of complex values (may be equivalenced to rz)

Output:
rz          array of real values (may be equivalenced to cz)

pfa2rc:
Input:
isign       sign of isign is the sign of exponent in fourier kernel
idim        dimension to transform, which must be either 1 or 2 (see notes)
n1          1st (fast) dimension of array to be transformed (see notes)
n2          2nd (slow) dimension of array to be transformed (see notes)
rz          array of real values (may be equivalenced to cz)

Output:
cz          array of complex values (may be equivalenced to rz)

Notes:
Table of valid n and cost for prime factor fft.  For each n, cost
was estimated to be the inverse of the number of ffts done in 1 sec
on an IBM RISC System/6000 Model 320H, by Dave Hale, 08/04/91.
```

(Redone by Jack Cohen for 15 sec to rebuild NTAB table on advice of
David and Gregory Chudnovsky, 05/03/94).
Cost estimates are least accurate for very small n. An alternative method
for estimating cost would be to count multiplies and adds, but this
method fails to account for the overlapping of multiplies and adds
that is possible on some computers, such as the IBM RS/6000 family.

npfa:
The returned n will be composed of mutually prime factors from
the set {2,3,4,5,7,8,9,11,13,16}. Because n cannot exceed
720720 = 5*7*9*11*13*16, 720720 is returned if nmin exceeds 720720.

npfao:
The returned n will be composed of mutually prime factors from
the set {2,3,4,5,7,8,9,11,13,16}. Because n cannot exceed
720720 = 5*7*9*11*13*16, 720720 is returned if nmin exceeds 720720.
If nmin does not exceed 720720, then the returned n will not be
less than nmin. The optimal n is chosen to minimize the estimated
cost of performing the fft, while satisfying the constraint, if
possible, that n not exceed nmax.

npfar and npfaro:
Current implemenations of real-to-complex and complex-to-real prime
factor ffts require that the transform length n be even and that n/2
be a valid length for a complex-to-complex prime factor fft. The
value returned by npfar satisfies these conditions. Also, see notes
for npfa.

pfacc:
n must be factorable into mutually prime factors taken
from the set {2,3,4,5,7,8,9,11,13,16}. in other words,
n = 2**p * 3**q * 5**r * 7**s * 11**t * 13**u
where
0 <= p <= 4, 0 <= q <= 2, 0 <= r,s,t,u <= 1
is required for pfa to yield meaningful results. this
restriction implies that n is restricted to the range
1 <= n <= 720720 (= 5*7*9*11*13*16)

pfacr:
Because pfacr uses pfacc to do most of the work, n must be even
and n/2 must be a valid length for pfacc. The simplest way to
obtain a valid n is via n = npfar(nmin). A more optimal n can be
obtained with npfaro.

```
pfarc:
Because pfarc uses pfacc to do most of the work, n must be even
and n/2 must be a valid length for pfacc.  The simplest way to
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.

pfamcc:
To perform a two-dimensional transform of an n1 by n2 complex array
(assuming that both n1 and n2 are valid "n"), stored with n1 fast
and n2 slow:
    pfamcc(isign,n1,n2,1,n1,z); (to transform 1st dimension)
    pfamcc(isign,n2,n1,n1,1,z); (to transform 2nd dimension)

pfa2cc:
Only one (either the 1st or 2nd) dimension of the 2-D array is transformed.

If idim equals 1, then n2 transforms of n1 complex elements are performed;
else, if idim equals 2, then n1 transforms of n2 complex elements are
performed.

Although z appears in the argument list as a one-dimensional array,
z may be viewed as an n1 by n2 two-dimensional array:  z[n2][n1].

Valid n is computed via the "np" subroutines.

To perform a two-dimensional transform of an n1 by n2 complex array
(assuming that both n1 and n2 are valid "n"), stored with n1 fast
and n2 slow:  pfa2cc(isign,1,n1,n2,z);  pfa2cc(isign,2,n1,n2,z);

pfa2cr:
If idim equals 1, then n2 transforms of n1/2+1 complex elements to n1 real
elements are performed; else, if idim equals 2, then n1 transforms of n2/2+1
complex elements to n2 real elements are performed.

Although rz appears in the argument list as a one-dimensional array,
rz may be viewed as an n1 by n2 two-dimensional array:  rz[n2][n1].
Likewise, depending on idim, cz may be viewed as either an n1/2+1 by
n2 or an n1 by n2/2+1 two-dimensional array of complex elements.

Let n denote the transform length, either n1 or n2, depending on idim.
Because pfa2rc uses pfa2cc to do most of the work, n must be even
and n/2 must be a valid length for pfa2cc.  The simplest way to
```

```
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.

pfa2rc:
If idim equals 1, then n2 transforms of n1 real elements to n1/2+1 complex
elements are performed; else, if idim equals 2, then n1 transforms of n2
real elements to n2/2+1 complex elements are performed.

Although rz appears in the argument list as a one-dimensional array,
rz may be viewed as an n1 by n2 two-dimensional array:  rz[n2][n1].
Likewise, depending on idim, cz may be viewed as either an n1/2+1 by
n2 or an n1 by n2/2+1 two-dimensional array of complex elements.

Let n denote the transform length, either n1 or n2, depending on idim.
Because pfa2rc uses pfa2cc to do most of the work, n must be even
and n/2 must be a valid length for pfa2cc.  The simplest way to
obtain a valid n is via n = npfar(nmin).  A more optimal n can be
obtained with npfaro.

References:
Temperton, C., 1985, Implementation of a self-sorting
in-place prime factor fft algorithm:  Journal of
Computational Physics, v. 58, p. 283-299.

Temperton, C., 1988, A new set of minimum-add rotated
rotated dft modules: Journal of Computational Physics,
v. 75, p. 190-198.

Differ significantly from Press et al, 1988, Numerical Recipes in C, p. 417.

Author:  Dave Hale, Colorado School of Mines, 04/27/89
```

```
POLAR - Functions to map data in rectangular coordinates to polar and vise versa

recttopolar convert a function p(x,y) to a function q(a,r)
polartorect convert a function q(a,r) to a function p(x,y)

Function Prototypes:
void recttopolar ( int nx, float dx, float fx, int ny, float dy, float fy,
float **p, int na, float da, float fa, int nr,
float dr, float fr, float **q);
void polartorect ( int na, float da, float fa, int nr, float dr, float fr,
float **q, int nx, float dx, float fx, int ny,
float dy, float fy, float **p)

recttopolar:
Input:
nx number of x samples
dx x sampling interval
fx first x sample
ny number of y samples
dy y sampling interval
fy first y sample
p array[ny][nx] containing samples of p(x,y)
na number of a samples
da a sampling interval
fa first a sample
nr number of r samples
dr r sampling interval
fr first r sample

Output:
q array[nr][na] containing samples of q(a,r)

polartorect:
Input:
na number of a samples
da a sampling interval
fa first a sample
nr number of r samples
dr r sampling interval
fr first r sample
nx number of x samples
dx x sampling interval
fx first x sample
```

```
ny number of y samples
dy y sampling interval
fy first y sample
q array[nr][na] containing samples of q(a,r)

Output:
p array[ny][nx] containing samples of p(x,y)

Notes:
The polar angle a is measured in radians,
x = r*cos(a) and y = r*sin(a).

recttopolar:
Linear extrapolation is used to determine the value of p(x,y) for
x and y coordinates not in the range corresponding to nx, dx, ....

polartorect:
Linear extrapolation is used to determine the value of q(a,r) for
a and r coordinates not in the range corresponding to na, da, ....

Author:  Dave Hale, Colorado School of Mines, 06/15/90
```

```
PRINTERPLOT - Functions to make a printer plot of a 1-dimensional array

pp1d printer plot of 1d array
pplot1 printer plot of 1d array

Function Prototypes:
void pp1d (FILE *fp, char *title, int lx, int ifx, float x[]);
void pplot1 (FILE *fp, char *title, int nx, float ax[]);

pp1d:
Input:
fp file pointer for output (e.g., stdout, stderr, etc.)
title title of plot
lx length of x
ifx index of first x
x array[lx] to be plotted

pplot1:
Input:
fp file pointer for printed output (e.g., stdout, stderr, etc.)
title title of plot
nx number of x values to be plotted
ax array[nx] of x values

Notes:
Are two subroutines to do this really necessary?

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
QUEST - Functions to ESTimate Quantiles:

quest returns estimate - use when entire array is in memory
questalloc returns quantile estimator - use before questupdate
questupdate updates and returns current estimate - use for large
numbers of floats, too big to fit in memory at one time
questfree frees quantile estimator

quest:
Input:
p quantile to be estimated (0.0<=p<=1.0 is required)
n number of samples in array x (n>=5 is required)
x array[n] of floats

Returned: the estimate of a specified quantile

questalloc:
Input:
p quantile to be estimated (0.0<=p<=1.0 is required)
n number of samples in array x (n>=5 is required)
x array[n] of floats

Returned: pointer to a quantile estimator

questupdate:
Input:
q pointer to quantile estimator (as returned by questalloc)
n number of samples in array x
x array[n] of floats

Returned: quantile estimate

questfree:
q pointer to quantile estimator (as returned by questalloc)

Notes:
quest:
The estimate should approach the sample quantile in the limit of large n.

The estimate is most accurate for cumulative distribution functions
that are smooth in the neighborhood of the quantile specified by p.

This function is an implementation of the algorithm published by
```

Jain and Chlamtac (1985).

questalloc:
This function must be called before calling function questupdate.
See also notes in questupdate.

questupdate:
The estimate should approach the sample quantile in the limit of large n.

The estimate is most accurate for cumulative distribution functions
that are smooth in the neighborhood of the quantile specified by p.

This function is an implementation of the algorithm published by
Jain, R. and Chlamtac, I., 1985, The PP algorithm for dynamic
calculation of quantiles and histograms without storing observations:
Comm. ACM, v. 28, n. 10.

Reference:
Jain, R. and Chlamtac, I., 1985, The PP algorithm for dynamic
calculation of quantiles and histograms without storing observations:
Comm. ACM, v. 28, n. 10.

Author: Dave Hale, Colorado School of Mines, 05/07/89

```
RESSINC8 - Functions to resample uniformly-sampled data  via 8-coefficient sinc
 approximations:

ress8c resample a uniformly-sampled complex function via 8-coeff. sinc approx.
ress8r resample a uniformly-sampled real function via 8-coeff. sinc approx.

Function Prototypes:
void ress8r (int nxin, float dxin, float fxin, float yin[],
float yinl, float yinr,
int nxout, float dxout, float fxout, float yout[]);
void ress8c (int nxin, float dxin, float fxin, complex yin[],
complex yinl, complex yinr,
int nxout, float dxout, float fxout, complex yout[]);

Input:
nxin number of x values at which y(x) is input
dxin x sampling interval for input y(x)
fxin x value of first sample input
yin array[nxin] of input y(x) values:  yin[0] = y(fxin), etc.
yinl value used to extrapolate yin values to left of yin[0]
yinr value used to extrapolate yin values to right of yin[nxin-1]
nxout number of x values at which y(x) is output
dxout x sampling interval for output y(x)
fxout x value of first sample output

Output:
yout array[nxout] of output y(x) values:  yout[0] = y(fxout), etc.

Notes:
Because extrapolation of the input function y(x) is defined by the
left and right values yinl and yinr, the output x values defined
by nxout, dxout, and fxout are not restricted to lie within the range
of input x values defined by nxin, dxin, and fxin.

The maximum error for frequiencies less than 0.6 nyquist is
less than one percent.

Author:  Dave Hale, Colorado School of Mines, 06/06/90
```

```
RFWTVA - Rasterize a Float array as Wiggle-Trace-Variable-Area.

rfwtva rasterize a float array as wiggle-trace-variable-area.

Function Prototype:
void rfwtva (int n, float z[], float zmin, float zmax, float zbase,
int yzmin, int yzmax, int xfirst, int xlast,
int wiggle, int nbpr, unsigned char *bits, int endian);

Input:
n number of samples in array to rasterize
z array[n] to rasterize
zmin z values below zmin will be clipped
zmax z values above zmax will be clipped
zbase z values between zbase and zmax will be filled (see notes)
yzmin horizontal raster coordinate corresponding to zmin
yzmax horizontal raster coordinate corresponding to zmax
xfirst vertical raster coordinate of z[0] (see notes)
xlast vertical raster coordinate of z[n-1] (see notes)
wiggle =0 for no wiggle (VA only); =1 for wiggle (with VA)
wiggle 2<=wiggle<=5 for solid/grey coloring of VA option
                shade of grey: wiggle=2 light grey, wiggle=5 black
nbpr number of bytes per row of bits
bits pointer to first (top,left) byte in image
endian byte order  =1 big endian  =0 little endian

Output:
bits pointer to first (top,left) byte in image

Notes:
The raster coordinate of the (top,left) bit in the image is (0,0).
In other words, x increases downward and y increases to the right.
Raster scan lines run from left to right, and from top to bottom.
Therefore, xfirst, xlast, yzmin, and yzmax should not be less than 0.
Likewise, yzmin and yzmax should not be greater than nbpr*8-1, and
care should be taken to ensure that xfirst and xlast do not cause bits
to be set outside (off the bottom) of the image.

Variable area fill is performed on the right-hand (increasing y) side
of the wiggle.  If yzmin is greater than yzmax, then z values between
zmin will be plotted to the right of zmax, and z values between zbase
and zmin are filled.  Swapping yzmin and yzmax is an easy way to
reverse the polarity of a wiggle.
```

```
The variable "endian" must have a value of 1 or 0. If this is
not a case an error is returned.

Author:  Dave Hale, Colorado School of Mines, 07/01/89
MODIFIED:  Paul Michaels, Boise State University, 29 December 2000
           Added solid/grey shading scheme, wiggle>=2 option for peaks/troughs
MODIFIED:  Kris Vanneste, Royal Observatory of Belgium, July 2005
           Added missing wiggle trace bits in y (amplitude) direction
```

```
RFWTVAINT - Rasterize a Float array as Wiggle-Trace-Variable-Area, with
    8 point sinc INTerpolation.

rfwtvaint rasterize a float array as wiggle-trace-variable-area, and
apply sinc interploation for display purposes.

Function Prototype:
void rfwtvaint (int n, float z[], float zmin, float zmax, float zbase,
int yzmin, int yzmax, int xfirst, int xlast,
int wiggle, int nbpr, unsigned char *bits, int endian);

Input:
n number of samples in array to rasterize
z array[n] to rasterize
zmin z values below zmin will be clipped
zmax z values above zmax will be clipped
zbase z values between zbase and zmax will be filled (see notes)
yzmin horizontal raster coordinate corresponding to zmin
yzmax horizontal raster coordinate corresponding to zmax
xfirst vertical raster coordinate of z[0] (see notes)
xlast vertical raster coordinate of z[n-1] (see notes)
wiggle =0 for no wiggle (VA only); =1 for wiggle (with VA)
                wiggle 2<=wiggle<=5 for solid/grey coloring of VA option
                shade of grey: wiggle=2 light grey, wiggle=5 black
nbpr number of bytes per row of bits
bits pointer to first (top,left) byte in image
endian byte order  =1 big endian  =0 little endian

Output:
bits pointer to first (top,left) byte in image

Notes:
The raster coordinate of the (top,left) bit in the image is (0,0).
In other words, x increases downward and y increases to the right.
Raster scan lines run from left to right, and from top to bottom.
Therefore, xfirst, xlast, yzmin, and yzmax should not be less than 0.
Likewise, yzmin and yzmax should not be greater than nbpr*8-1, and
care should be taken to ensure that xfirst and xlast do not cause bits
to be set outside (off the bottom) of the image.

Variable area fill is performed on the right-hand (increasing y) side
of the wiggle.  If yzmin is greater than yzmax, then z values between
zmin will be plotted to the right of zmax, and z values between zbase
```

```
and zmin are filled.  Swapping yzmin and yzmax is an easy way to
reverse the polarity of a wiggle.

The variable "endian" must have a value of 1 or 0. If this is
not a case an error is returned.

The interpolation is by the 8 point sinc interpolation routine s8r.

Author:  Dave Hale, Colorado School of Mines, 07/01/89
Memorial University of Newfoundland: Tony Kocurko, Sept 1995.
 Added sinc interpolation.
MODIFIED: Paul Michaels, Boise State University, 29 December 2000
          added solid/grey color scheme for peaks/troughs  wiggle=2 option
MODIFIED:  Kris Vanneste, Royal Observatory of Belgium, July 2005
           Added missing wiggle trace bits in y (amplitude) direction
```

```
SBLAS - Single precision Basic Linear Algebra Subroutines
(adapted from LINPACK FORTRAN):

isamax return index of element with maximum absolute value
sasum return sum of absolute values
saxpy compute y[i] = a*x[i]+y[i]
scopy copy x[i] to y[i] (i.e., set y[i] = x[i])
sdot return sum of x[i]*y[i] (i.e., return the dot product of x and y)
snrm2 return square root of sum of squares of x[i]
sscal compute x[i] = a*x[i]
sswap swap x[i] and y[i]

Function Prototypes:
int isamax (int n, float *sx, int incx);
float sasum (int n, float *sx, int incx);
void saxpy (int n, float sa, float *sx, int incx, float *sy, int incy);
void scopy (int n, float *sx, int incx, float *sy, int incy);
float sdot (int n, float *sx, int incx, float *sy, int incy);
float snrm2 (int n, float *sx, int incx);
void sscal (int n, float sa, float *sx, int incx);
void sswap (int n, float *sx, int incx, float *sy, int incy);

isamax:
Input:
n number of elements in array
sx array[n] of elements
incx increment between elements

Returned: index of element with maximum absolute value

sasum:
Input:
n number of elements in array
sx array[n] of elements
incx increment between elements

Returned: sum of absolute values

saxpy:
Input:
n number of elements in arrays
sa the scalar multiplier
sx array[n] of elements to be scaled and added
```

```
incx increment between elements of sx
sy array[n] of elements to be added
incy increment between elements of sy

Output:
sy array[n] of accumulated elements

scopy:
Input:
n number of elements in arrays
sx array[n] of elements to be copied
incx increment between elements of sx
incy increment between elements of sy

Output:
sy array[n] of copied elements

sdot:
Input:
n number of elements in arrays
sx array[n] of elements
incx increment between elements of sx
sy array[n] of elements
incy increment between elements of sy

Returned: dot product of the two arrays

snrm2
Input:
n number of elements in array
sx array[n] of elements
incx increment between elements

Returned: square root of sum of squares of x[i]

sscal:
Input:
n number of elements in array
sa the scalar multiplier
sx array[n] of elements
incx increment between elements

Output:
```

```
sx array[n] of scaled elements

sswap:
Input:
n number of elements in arrays
sx array[n] of elements
incx increment between elements of sx
sy array[n] of elements
incy increment between elements of sy

Output:
sx array[n] of swapped elements
sy array[n] of swapped elements

Notes:
Adapted from Linpack Fortran.

snrm2:
This simple version may cause overflow or underflow!

Author:  Dave Hale, Colorado School of Mines, 10/01/89
```

```
SCAXIS - compute a readable scale for use in plotting axes

scaxis compute a readable scale for use in plotting axes

Function Prototype:
void scaxis (float x1, float x2, int *nxnum, float *dxnum, float *fxnum);

Input:
x1 first x value
x2 second x value
nxnum desired number of numbered values

Output:
nxnum number of numbered values
dxnum increment between numbered values (dxnum>0.0)
fxnum first numbered value

Notes:
scaxis attempts to honor the user-specified nxnum.  However, nxnum
will be modified if necessary for readability.  Also, fxnum and nxnum
will be adjusted to compensate for roundoff error; in particular,
fxnum will not be less than xmin-eps, and fxnum+(nxnum-1)*dxnum
will not be greater than xmax+eps, where eps = 0.0001*(xmax-xmin).
xmin is the minimum of x1 and x2.  xmax is the maximum of x1 and x2.

Author:  Dave Hale, Colorado School of Mines, 01/13/89
```

```
SGA - Single precision general matrix functions adapted from LINPACK FORTRAN:

sgefa Gaussian elimination to obtain the LU factorization of a matrix.
sgeco Gaussian elimination to obtain the LU factorization and
condition number of a matrix.
sgesl Solve linear system Ax = b or A'x = b after LU factorization.

Function Prototypes:
void sgefa (float **a, int n, int *ipvt, int *info);
void sgeco (float **a, int n, int *ipvt, float *rcond, float *z);
void sgesl (float **a, int n, int *ipvt, float *b, int job);

sgefa:
Input:
a matrix[n][n] to be factored (see notes below)
n dimension of a

Output:
a matrix[n][n] factored (see notes below)
ipvt indices of pivot permutations (see notes below)
info index of last zero pivot (or -1 if no zero pivots)

sgeco:
Input:
a matrix[n][n] to be factored (see notes below)
n dimension of a

Output:
a matrix[n][n] factored (see notes below)
ipvt indices of pivot permutations (see notes below)
rcond reciprocal condition number (see notes below)

Workspace:
z array[n]

sgesl
Input:
a matrix[n][n] that has been LU factored (see notes below)
n dimension of a
ipvt indices of pivot permutations (see notes below)
b right-hand-side vector[n]
job =0 to solve Ax = b
=1 to solve A'x = b
```

```
Output:
b solution vector[n]

Notes:
These functions were adapted from LINPACK FORTRAN.  Because two-dimensional
arrays cannot be declared with variable dimensions in C, the matrix a
is actually a pointer to an array of pointers to floats, as declared
above and used below.

Elements of a are stored as follows:
a[0][0]    a[1][0]    a[2][0]   ... a[n-1][0]
a[0][1]    a[1][1]    a[2][1]   ... a[n-1][1]
a[0][2]    a[1][2]    a[2][2]   ... a[n-1][2]
.                                     .
.              .                      .
.                        .            .
.                                     .
a[0][n-1]  a[1][n-1]  a[2][n-1] ... a[n-1][n-1]

Both the factored matrix a and the pivot indices ipvt are required
to solve linear systems of equations via sgesl.

sgeco:
Given the reciprocal of the condition number, rcond, and the float
epsilon, FLT_EPSILON, the number of significant decimal digits, nsdd,
in the solution of a linear system of equations may be estimated by:
nsdd = (int)log10(rcond/FLT_EPSILON)

This function was adapted from LINPACK FORTRAN.  Because two-dimensional
arrays cannot be declared with variable dimensions in C, the matrix a
is actually a pointer to an array of pointers to floats, as declared
above and used below.

Author:  Dave Hale, Colorado School of Mines, 10/01/89
```

```
SHFS8R - Shift a uniformly-sampled real-valued function y(x) via
a table of 8-coefficient sinc approximations.

shfs8r  shift a uniformly-sampled real function via a table of 8-coeff. sinc
approximations.

Function Prototypes:
void shfs8r (float dx, int nxin, float fxin, float yin[],
float yinl, float yinr, int nxout, float fxout, float yout[]);

Input:
dx       x sampling interval for both input and output y(x)
nxin     number of x values at which y(x) is input
fxin     x value of first sample input
yin      array[nxin] of input y(x) values:  yin[0] = y(fxin), etc.
yinl     value used to extrapolate yin values to left of yin[0]
yinr     value used to extrapolate yin values to right of yin[nxin-1]
nxout    number of x values a which y(x) is output
fxout    x value of first sample output

Output:
yout     array[nxout] of output y(x) values:  yout[0] = y(fxout), etc.

Notes:
Because extrapolation of the input function y(x) is defined by the
left and right values yinl and yinr, the output samples defined by
dx, nxout, and fxout are not restricted to lie within the range of
input sample locations defined by dx, nxin, and fxin.

The maximum error for frequencies less than 0.6*nyquist is less
than one percent.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
SINC - Return SINC(x) for as floats or as doubles

fsinc return float value of sinc(x) for x input as a float
dsinc return double precision sinc(x) for double precision x

Function Prototype:
float fsinc (float x);
double dsinc (double x);

Input:
x value at which to evaluate sinc(x)

Returned:  sinc(x)

Notes:
    sinc(x) = sin(PI*x)/(PI*x)

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
SORT - Functions to sort arrays of data or arrays of indices

hpsort sort an array of floats by the heap sort method
qkisort sort an array of indices i[] so that
a[i[0]] <= a[i[1]] <= ... <= a[i[n-1]]
uses the "quick sort" method
qkifind partially sort an array of indices i[] so that the
index i[m] has the value it would have if the entire
array of indices were sorted such that
a[i[0]] <= a[i[1]] <= ... <= a[i[n-1]]
uses the "quick sort" method
qksort sort an array of floats such that a[0] <= a[1] <= ... <= a[n-1]
uses the "quick sort" method
qkfind partially sort an array of floats  so that the element a[m] has
the value it would have if the entire array were sorted
such that a[0] <= a[1] <= ... <= a[n-1]
uses the "quick sort" method

Function Prototypes:
void hpsort (int n, float a[]);
void qkisort (int n, float a[], int i[]);
void qkifind (int m, int n, float a[], int i[]);
void qksort (int n, float a[]);
void qkfind (int m, int n, float a[]);

hpsort:
Input:
n number of elements in a
a array[n] to be sorted

Output:
a array[n] sorted

qkisort:
Input:
n number of elements in array a
a array[n] elements
i array[n] indices to be sorted

Output:
i array[n] indices sorted

qkifind:
```

```
Input:
m index of element to be found
n number of elements in array a
a array[n] elements
i array[n] indices to be partially sorted

Output:
i array[n] indices partially sorted sorted

qksort:
Input:
n number of elements in array a
a array[n] containing elements to be sorted

Output:
a array[n] containing sorted elements

qkfind:
Input:
m index of element to be found
n number of elements in array a
a array[n] to be partially sorted

Output:
a array[n] partially sorted


Notes:
hpsort:
The heap sort algorithm is, at worst, N log_2 N, and in most cases
is 20% faster.  Adapted from Standish.

qkisort, qkifind, qksort, qkfind:
n must be less than 2^NSTACK, where NSTACK is defined above.

qkisort:
This function is adapted from procedure quicksort by
Hoare, C.A.R., 1961, Communications of the ACM, v. 4, p. 321;
the main difference is that recursion is accomplished
explicitly via a stack array for efficiency; also, a simple
insertion sort is used to sort subarrays too small to be
partitioned efficiently.
```

```
qkifind:
This function is adapted from procedure find by
Hoare, C.A.R., 1961, Communications of the ACM, v. 4, p. 321.

qksort:
This function is adapted from procedure quicksort by
Hoare, C.A.R., 1961, Communications of the ACM, v. 4, p. 321;
the main difference is that recursion is accomplished
explicitly via a stack array for efficiency; also, a simple
insertion sort is used to sort subarrays too small to be
partitioned efficiently.

qkfind:
This function is adapted from procedure find by Hoare 1961.

Reference:
hpsort:
Standish, T. A., Data Structure Techniques, p. 91.
See also Press, W. A., et al., Numerical Recipes in C.

quick sort:
Hoare, C.A.R., 1961, Communications of the ACM, v. 4, p. 321.

Author:  Dave Hale, Colorado School of Mines, 12/26/89
```

```
SQR - Single precision QR decomposition functions adapted from LINPACK FORTRAN:

sqrdc Compute QR decomposition of a matrix.
sqrsl Use QR decomposition to solve for coordinate transformations,
projections, and least squares solutions.
sqrst Solve under-determined or over-determined least squares problems,
with a user-specified tolerance.

Function Prototypes:
void sqrdc (float **x, int n, int p, float *qraux, int *jpvt,
float *work, int job);
void sqrsl (float **x, int n, int k, float *qraux,
float *y, float *qy, float *qty,
float *b, float *rsd, float *xb, int job, int *info);
void sqrst (float **x, int n, int p, float *y, float tol,
float *b, float *rsd, int *k,
int *jpvt, float *qraux, float *work);

sqrdc:
Input:
x matrix[p][n] to decompose (see notes below)
n number of rows in the matrix x
p number of columns in the matrix x
jpvt array[p] controlling the pivot columns (see notes below)
job =0 for no pivoting;
=1 for pivoting

Output:
x matrix[p][n] decomposed (see notes below)
qraux array[p] containing information required to recover the
orthogonal part of the decomposition
jpvt array[p] with jpvt[k] containing the index of the original
matrix that has been interchanged into the k-th column, if
pivoting is requested.

Workspace:
work array[p] of workspace

sqrsl:
Input:
x matrix[p][n] containing output of sqrdc.
n number of rows in the matrix xk; same as in sqrdc.
k number of columns in the matrix xk; k must not be greater
```

```
than MIN(n,p), where p is the same as in sqrdc.
qraux array[p] containing auxiliary output from sqrdc.
y array[n] to be manipulated by sqrsl.
job specifies what is to be computed.  job has the decimal
expansion ABCDE, with the following meaning:
if A != 0, compute qy.
if B, C, D, or E != 0, compute qty.
if C != 0, compute b.
if D != 0, compute rsd.
if E != 0, compute xb.
Note that a request to compute b, rsd, or xb automatically
triggers the computation of qty, for which an array must
be provided.

Output:
qy array[n] containing Qy, if its computation has been
requested.
qty array[n] containing Q'y, if its computation has
been requested.  Here Q' denotes the transpose of Q.
b array[k] containing solution of the least squares problem:
minimize norm2(y - xk*b),
if its computation has been requested.  (Note that if
pivoting was requested in sqrdc, the j-th component of
b will be associated with column jpvt[j] of the original
matrix x that was input into sqrdc.)
rsd array[n] containing the least squares residual y - xk*b,
if its computation has been requested.  rsd is also the
orthogonal projection of y onto the orthogonal complement
of the column space of xk.
xb array[n] containing the least squares approximation xk*b,
if its computation has been requested.  xb is also the
orthogonal projection of y onto the column space of x.
info =0 unless the computation of b has been requested and R
is exactly singular.  In this case, info is the index of
the first zero diagonal element of R and b is left
unaltered.

sqrst:
Input:
x matrix[p][n] of coefficients (x is destroyed by sqrst.)
n number of rows in the matrix x (number of observations)
p number of columns in the matrix x (number of parameters)
y array[n] containing right-hand-side (observations)
```

```
tol relative tolerance used to determine the subset of
columns of x included in the solution.  If tol is zero,
a full complement of the MIN(n,p) columns is used.
If tol is non-zero, the problem should be scaled so that
all the elements of X have roughly the same absolute
accuracy eps.  Then a reasonable value for tol is roughly
eps divided by the magnitude of the largest element.

Output:
x matrix[p][n] containing output of sqrdc
b array[p] containing the solution (parameters); components
corresponding to columns not used are set to zero.
rsd array[n] of residuals y - Xb
k number of columns of x used in the solution
jpvt array[p] containing pivot information from sqrdc.
qraux array[p] containing auxiliary information from sqrdc.

Workspace:
work array[p] of workspace.

Notes:
!!! WARNING !!!
These functions have many options, not all of which have been tested!
(Dave Hale, 12/31/89)


This function was adapted from LINPACK FORTRAN.  Because two-dimensional
arrays cannot be declared with variable dimensions in C, the matrix x
is actually a pointer to an array of pointers to floats, as declared
above and used below.

Elements of x are stored as follows:
x[0][0]    x[1][0]    x[2][0]   ... x[p-1][0]
x[0][1]    x[1][1]    x[2][1]   ... x[p-1][1]
x[0][2]    x[1][2]    x[2][2]   ... x[p-1][2]
.                                     .
.             .                       .
.                        .            .
.                                     .
x[0][n-1]  x[1][n-1]  x[2][n-1] ... x[p-1][n-1]

sqrdc:
Uses Householder transformations to compute the QR decomposition of an n by p
```

matrix x. Column pivoting based on the 2-norms of the reduced columns may be performed at the user's option.

After decomposition, x contains in its upper triangular matrix R of the QR decomposition. Below its diagonal x contains information from which the orthogonal part of the decomposition can be recovered. Note that if pivoting has been requested, the decomposition is not that of the original matrix x but that of x with its columns permuted as described by jpvt.

The selection of pivot columns is controlled by jpvt as follows.
The k-th column x[k] of x is placed in one of three classes according to the value of jpvt[k].
if jpvt[k] > 0, then x[k] is an initial column.
if jpvt[k] == 0, then x[k] is a free column.
if jpvt[k] < 0, then x[k] is a final column.
Before the decomposition is computed, initial columns are moved to the beginning of the array x and final columns to the end. Both initial and final columns are frozen in place during the computation and only free columns are moved. At the k-th stage of the reduction, if x[k] is occupied by a free column it is interchanged with the free column of largest reduced norm. jpvt is not referenced if job == 0.

sqrsl:
Uses the output of sqrdc to compute coordinate transformations, projections, and least squares solutions. For k <= MIN(n,p), let xk be the matrix
xk = (x[jpvt[0]], x[jpvt[1]], ..., x[jpvt[k-1]])
formed from columns jpvt[0], jpvt[1], ..., jpvt[k-1] of the original n by p matrix x that was input to sqrdc. (If no pivoting was done, xk consists of the first k columns of x in their original order.) sqrdc produces a factored orthogonal matrix Q and an upper triangular matrix R such that
xk = Q * (R)
         (0)
This information is contained in coded form in the arrays x and qraux.

The parameters qy, qty, b, rsd, and xb are not referenced if their computation is not requested and in this case can be replaced by NULL pointers in the calling program. To save storage, the user may in some cases use the same array for different parameters in the calling sequence. A frequently occuring example is when one wishes to compute any of b, rsd, or xb and does not need y or qty. In this case one may equivalence y, qty, and one of b, rsd, or xb, while providing separate arrays for anything else that is to be computed. Thus the calling

```
sequence
sqrsl(x,n,k,qraux,y,NULL,y,b,y,NULL,110,&info)
will result in the computation of b and rsd, with rsd overwriting y.
More generally, each item in the following list contains groups of
permissible equivalences for a single calling sequence.
1. (y,qty,b) (rsd) (xb) (qy)
2. (y,qty,rsd) (b) (xb) (qy)
3. (y,qty,xb) (b) (rsd) (qy)
4. (y,qy) (qty,b) (rsd) (xb)
5. (y,qy) (qty,rsd) (b) (xb)
6. (y,qy) (qty,xb) (b) (rsd)
In any group the value returned in the array allocated to the group
corresponds to the last member of the group.

sqrst:
Computes least squares solutions to the system
Xb = y
which may be either under-determined or over-determined.  The user may
supply a tolerance to limit the columns of X used in computing the solution.
In effect, a set of columns with a condition number approximately bounded
by 1/tol is used, the other components of b being set to zero.

On return, the arrays x, jpvt, and qraux contain the usual output from
sqrdc, so that the QR decomposition of x with pivoting is fully available
to the user.  In particular, columns jpvt[0], jpvt[1], ..., jpvt[k-1]
were used in the solution, and the condition number associated with
those columns is estimated by ABS(x[0][0]/x[k-1][k-1]).

Author:  Dave Hale, Colorado School of Mines, 12/29/89
```

```
STOEP - Functions to solve a symmetric Toeplitz linear system of equations
 Rf=g for f

stoepd solve a symmetric Toeplitz system - doubles
stoepf solve a symmetric Toeplitz system - floats

Function Prototypes:
void stoepd (int n, double r[], double g[], double f[], double a[]);
void stoepf (int n, float r[], float g[], float f[], float a[]);

Input:
n dimension of system
r array[n] of top row of Toeplitz matrix
g array[n] of right-hand-side column vector

Output:
f array[n] of solution (left-hand-side) column vector
a array[n] of solution to Ra=v (Claerbout, FGDP, p. 57)

Notes:
These routines do NOT solve the case when the main diagonal is zero, it
just silently returns.

The left column of the Toeplitz matrix is assumed to be equal to the top
row (as specified in r); i.e., the Toeplitz matrix is assumed symmetric.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
STRSTUFF -- STRing manuplation subs

cwp_strdup -  duplicate a string
strchop - chop off the tail end of a string "s" after a "," returning
  the front part of "s" as "t".
cwp_strrev - reverse a string



Input:
char *str input string

Output:
none

Returns:
char *   duplicated string

Notes:
This local definition of strdup is necessary because some systems
do not have it.

Author: John Stockwell, Spring 2000.
```

SWAPBYTE - Functions to SWAP the BYTE order of binary data

swap_short_2 swap a short integer
swap_u_short_2 swap an unsigned short integer
swap_int_4 swap a 4 byte integer
swap_u_int_4 swap an unsigned integer
swap_long_4 swap a long integer
swap_u_long_4 swap an unsigned long integer
swap_float_4 swap a float
swap_double_8 swap a double

Function Prototypes:
```
void swap_short_2(short *tni2);
void swap_u_short_2(unsigned short *tni2);
void swap_int_4(int *tni4);
void swap_u_int_4(unsigned int *tni4);
void swap_long_4(long *tni4);
void swap_u_long_4(unsigned long *tni4);
void swap_float_4(float *tnf4);
void swap_double_8(double *tndd8);
```

Notes:
These routines are necessary for reversing the byte order of binary data for transportation between big-endian and little-endian machines. Examples of big-endian machines are IBM RS6000, SUN, NeXT. Examples of little endian machines are PC's and DEC.

These routines have been tested with PC data and run on PC's running several PC versions of UNIX, but have not been tested on DEC.

Also, the number appended to the name of the routine refers to the number of bytes that the item is assumed to be.

Authors: Jens Hartmann, Institut fur Geophysik, Hamburg, Jun 1993
John Stockwell, CWP, Colorado School of Mines, Jan 1994

```
SYMMEIGEN - Functions solving the eigenvalue problem for symmetric matrices
eig_jacobi - find eigenvalues and corresponding eigenvectors via
         the jacobi algorithm for symmetric matrices
sort_eigenvalues -  sort eigenvalues and corresponding eigenvectors
in descending order

Function Prototypes:
void eig_jacobi(float **a, float d[], float **v, int n);
void sort_eigenvalues(float d[], float **v, int n);
 (inspired by Press et. al., 1996)

 Macro used internally
define ROTATE(a,i,j,k,l) g=a[i][j];h=a[k][l];a[i][j]=g-s*(h+g*tau);\
    a[k][l]=h+s*(g-h*tau);

void eig_jacobi(float **a, float d[], float **v, int n)
eig_jacobi - find eigenvalues and corresponding eigenvectors via
         the jacobi algorithm for symmetric matrices

Function Prototype:
void eig_jacobi(float **a, float d[], float **v, int n);
 (inspired by Press et. al., 1996)
{
    int j,iq,ip,i;
    float tresh,theta,tau,t,sm,s,h,g,c,*b,*z;

    /* allocate space temporarily
    b=alloc1float(n); b-=1;
    z=alloc1float(n); z-=1;

    /* initialize v to the identity matrix
    for (ip=1;ip<=n;ip++) {
        for (iq=1;iq<=n;iq++) v[ip][iq]=0.0;
        v[ip][ip]=1.0;
    }
    /* initialilize to the diagonal on matrix a
    for (ip=1;ip<=n;ip++) {
        b[ip]=d[ip]=a[ip][ip];
        z[ip]=0.0;
    }

    /* main iteration loop
    for (i=1;i<=50;i++) {
```

```
sm=0.0;
for (ip=1;ip<=n-1;ip++) {
    for (iq=ip+1;iq<=n;iq++)
        sm += fabs(a[ip][iq]);
}
/* normal return
if (sm == 0.0) {
    z+=1; free1float(z);
    b+=1; free1float(b);
    return;
}
/* tresh values for first 3 sweeps and therafter
if (i < 4)
    tresh=0.2*sm/(n*n);
else
    tresh=0.0;
for (ip=1;ip<=n-1;ip++) {
    for (iq=ip+1;iq<=n;iq++) {
        g=100.0*fabs(a[ip][iq]);
        if (i > 4 && (float)(fabs(d[ip])+g) == (float)fabs(d[ip])
            && (float)(fabs(d[iq])+g) == (float)fabs(d[iq]))
            a[ip][iq]=0.0;
        else if (fabs(a[ip][iq]) > tresh) {
            h=d[iq]-d[ip];
            if ((float)(fabs(h)+g) == (float)fabs(h))
                t=(a[ip][iq])/h;
            else {
                theta=0.5*h/(a[ip][iq]);
                t=1.0/(fabs(theta)+sqrt(1.0+theta*theta));
                if (theta < 0.0) t = -t;
            }
            c=1.0/sqrt(1+t*t);
            s=t*c;
            tau=s/(1.0+c);
            h=t*a[ip][iq];
            z[ip] -= h;
            z[iq] += h;
            d[ip] -= h;
            d[iq] += h;
            a[ip][iq]=0.0;

            /* Jacobi rotations
```

```
                    for (j=1;j<=ip-1;j++) {
                        ROTATE(a,j,ip,j,iq)
                    }
                    for (j=ip+1;j<=iq-1;j++) {
                        ROTATE(a,ip,j,j,iq)
                    }
                    for (j=iq+1;j<=n;j++) {
                        ROTATE(a,ip,j,iq,j)
                    }
                    for (j=1;j<=n;j++) {
                        ROTATE(v,j,ip,j,iq)
                    }
                }
            }
        }
        for (ip=1;ip<=n;ip++) {
            b[ip] += z[ip];
            d[ip]=b[ip];
            z[ip]=0.0;
        }
    }
    /* this will not happen, hopefully
    fprintf(stderr,"jacobi iteration does not converge\n");
}


void sort_eigenvalues(float d[], float **v, int n)
sort_eigenvalues -  sort eigenvalues and corresponding eigenvectors
in descending order

Function Prototypes:
void sort_eigenvalues(float d[], float **v, int n);
 (inspired by Press et. al., 1996)
{
    int k,j,i;
    float p;

    for (i=1;i<n;i++) {
        p=d[k=i];
        for (j=i+1;j<=n;j++)
            if (d[j] >= p) p=d[k=j];
        if (k != i) {
            d[k]=d[i];
```

```
            d[i]=p;
            for (j=1;j<=n;j++) {
                p=v[j][i];
                v[j][i]=v[j][k];
                v[j][k]=p;
            }
        }
    }
}
```

```
include "cwp.h"

TEMPORARY_FILENAME - Creates a file name in a user-specified directory.

Function prototypes:
FILE *temporary_stream (char *tempfile);
char *temporary_filename(char *tempfile);

temporary_stream:
Input:
tempfile pointer to directory prefix string (eg. /usr/tmp/)

Output:
filestream pointer to temporary file stream
temporary_filename:
Input:
tempfile pointer to directory prefix string (eg. /usr/tmp/)

Output:
tempfile pointer to filename string (eg. /usr/tmp/1206aaa)

Notes:
temporary_stream creates a file stream by appending a sequence of
numbers and letters (which is created by mkstemp) to the prefix string
passed as its argument.

Author:  Andreas Klaedtke, 12/2/2009

temporary_filename creates a file name by appending a sequence of
numbers and letters (which is created by tmpnam) to the prefix string
passed as its argument.  On return the input argument points to the
(now augmented) prefix string.

It is duty of the calling program to provide room for the augmented
string.  The resulting string is typically used as a name for a
temporary file; in this case it is the calling program's job to make
sure that the supplied prefix ends with a slash.

This routine was written to supplement the ANSI C function tmpnam
which also creates a temporary filename, but within a fixed directory,
usually the /tmp directory.  Unfortunately, some /tmp directories are
too small to hold typical seismic data sets, so this routine allows
```

the user to specify a directory with sufficient capacity. Also note that on many systems, the tmpfile() call avoids this problem by simulating a temporary file with a memory buffer. However, this is not a panacea as the file size might exceed available memory and on some systems this call does actually create a file (again, usually in tmp).
Author: Jack K. Cohen, Colorado School of Mines, 12/12/95

```
TRIDIAGONAL - Functions to solve tridiagonal systems of equations Tu=r for u.

tridif Solve a tridiagonal system of equations (float version)
tridid Solve a tridiagonal system of equations (double version)
tripd Solve a positive definite, symmetric tridiagonal system
tripp Solve an unsymmetric tridiagonal system that uses
Gaussian elimination with partial pivoting

Function Prototypes:
void tridif (int n, float a[], float b[], float c[], float r[], float u[]);
void tridid (int n, double a[], double b[], double c[], double r[], double u[]);
void tripd (float *d, float *e, float *b, int n);
void tripp(int n, float *d, float *e, float *c, float *b);

tridif, tridid:
Input:
n dimension of system
a array[n] of lower sub-diagonal of T (a[0] ignored)
b array[n] of diagonal of T
c array[n] of upper super-diagonal of T (c[n-1] ignored)
r array[n] of right-hand-side column vector

Output:
u array[n] of solution (left-hand-side) column vector

tripd:
Input:
d array[n], the diagonal of A
e array[n], the superdiagonal of A
b array[n], the rhs column vector of Ax=b

Output:
b b is overwritten with the solution to Ax=b
tripp:
Input:
d diagonal vector of matrix
e       upper-diagonal vector of matrix
c       lower-diagonal vector of matrix
b       right-hand vector
n       dimension of matrix

Output:
b       solution vector
```

```
Notes:
For example, a tridiagonal system of dimension 4 is specified as:

    |b[0]    c[0]     0       0  | |u[0]|     |r[0]|
    |a[1]    b[1]    c[1]     0  | |u[1]|  =  |r[1]|
    | 0      a[2]    b[2]    c[2]| |u[2]|     |r[2]|
    | 0       0      a[3]    b[3]| |u[3]|     |r[3]|

The tridiagonal matrix is assumed to be non-singular.

tripd:
Given an n-by-n symmetric, tridiagonal, positive definite matrix A and
 n-vector b, following algorithm overwrites b with the solution to Ax = b

Authors:  tridif, tridid: Dave Hale, Colorado School of Mines, 10/03/89
tripd, tripp: Zhenyue Liu, Colorado School of Mines,  1992-1993
```

```
UNWRAP_PHASE - routines to UNWRAP phase of fourier transformed data

oppenheim_unwrap_phase - using the method of Oppenheim and Schafer (1975)
simple_unwrap_phase - by searching for phase jumps


Function Prototype:
void oppenheim_unwrap_phase(int n, int trend, int zeromean,
float df, float *xr, float *xi, float *phase);
void simple_unwrap_phase(int n, int trend, int zeromean,
float w,  float *phase);
oppenheim_unwrap_phase:
Input:
n number of samples
df frequency sampling interval
trend remove linear trend from unwrapped phase
xr real part of signal
xi imaginary part of signal

Output:
phase array[n] of output unwrapped phase values

simple_unwrap_phase:
Input:
n               number of samples
trend           remove linear trend from phase
zeromean        =0 assume phase(0)=0.0, else assume zero mean
w               unwrapping parameter; returns an error if w=0
phase           array[n] of inpu

Output:
phase           array[n] of output phase values

oppenheim_unwrap_phase:
Notes:
1) The phase unwrapping method proposed by Oppenheim and Schaffer
   1975 calculates the unwrapped phase by integrating the derivative
   with respect to frequency of the phase of a signal F(w) .

   Let u(w) = Re F(w) and v(w) = Im F(w)

   phase(w) = arctan[v(w)/u(w)]
```

```
   Taking the derivative with respect to w of both sides

d/dw [ phase(w) ] = d/dw ( arctan (v/u) )
=   [ 1/ (1 +(v/u)^2) ] ( v'/u - vu'/u^2 ) = ( v'u - vu' )/(u^2 +v^2)

2) Then, the d/dw phase   is integrated with respect to w
   to produce the phase function

   phase(w) = integral phase'(w)  dw

3) the user has the option of removing the linear trend in the phase

   The approach allows us to avoid the principle branch behavior of
   the arctangent function.

References:
Oppenheim A.V. and R.W. Schafer, Digital Signal Processing,
Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

Tria M., M. Van Der Baan 2, A. Larue, J. Mars 1, 2007,
Wavelet estimation in homomorphic domain by spectral
averaging, for deconvolution of seismic data
For the BLISS Project, Universit de Leeds,
ITF Consorsium collaboration(s) (2007)

simple_unwrap_phase:
Notes:
The strategy is to look at the change in phase (dphase) with each
time step. If |dphase| >  PI/w, then use the previous value of
dphase. No attempt is made at smoothing the dphase curve.

oppenheim_unwrap_phase:
Author: John Stockwell, CWP, 2010

simple_unwrap_phase
Author: John Stockwell, CWP, 2010
```

```
VANDERMONDE - Functions to solve Vandermonde system of equations Vx=b

vanded solve Vandermonde system of doubles
vandef solve Vandermonde system of floats

Function Prototypes:
void vanded (int n, double v[], double b[], double x[]);
void vandef (int n, float v[], float b[], float x[]);

Input:
n dimension of system
v array[n] of 2nd row of Vandermonde matrix (1st row is all ones)
b array[n] of right-hand-side column vector

Output:
x array[n] of solution column vector

Notes:
The arrays b and x may be equivalenced.

Reference:
Adapted from Algorithm 5.6-2 in Golub, G. H., and Van Loan, C. F., 1983,
Matrix Computations, John-Hopkins University Press.

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
WAVEFORMS   Subroutines to define some wavelets for modeling of seismic
            data

ricker1_wavelet     Compute the time response of a source function as
            a Ricker wavelet with peak frequency "fpeak" Hz.

ricker2_wavelet     Compute a Ricke wavelet with a given period, amplitude
            and distorsion factor

akb_wavelet         Compute the time response of a source function as
            a wavelet based on a wavelet used by Alford, Kelly,
            and Boore.

spike_wavelet       Compute the time response of a source function as
            a spike.

unit_wavelet        Compute the time response of a source function as
            a constant unit shift.

zero_wavelet        Compute the time response of a source function as
            zero everywhere.

berlage_wavelet     Compute the time response of a source function as a
            Berlage wavelet with peak frequency "fpeak" Hz,
            exponential decay factor "decay", time exponent
            "tn", and initial phase angle "ipa".

gaussian_wavelet    Compute the time response of a source function as a
            Gaussian wavelet with peak frequency "fpeak" in Hz.

gaussderiv_wavelet  Compute the time response of a source function as a
            Gaussian first derivative wavelet with peak frequency "fpeak"
            in Hz.
deriv_n_gauss  Compute the n-th derivative of a gaussian in double precision

Function Prototypes:
void ricker1_wavelet (int nt, float dt, float fpeak, float *wavelet);
void ricker2_wavelet (int hlw, float dt, float period, float ampl,
    float distort, float *wavelet);
void akb_wavelet (int nt, float dt, float fpeak, float *wavelet);
void spike_wavelet (int nt, int tindex, float *wavelet);
void unit_wavelet (int nt, float *wavelet);
void zero_wavelet (int nt, float *wavelet);
```

```
void berlage_wavelet (int nt, float dt, float fpeak, float ampl, float tn,
                      float decay, float ipa, float *wavelet);
void gaussian_wavelet (int nt, float dt, float fpeak, float *wavelet);
void gaussderiv_wavelet (int nt, float dt, float fpeak, float *wavelet);

Authors: Tong Fei, Ken Larner
Author: Nils Maercklin, February 2007
```

```
WINDOW - windowing routines

hanningnWindow - returns an n element long hanning window

Function prototypes:
void hanningnWindow(int n,float *w);
Author: Potash Corporation, Sascatchewan: Balasz Nemeth given to CWP 2008
```

```
wrapArray - wrap an array

Function prototype:
void wrapArray(void *base,size_t nmemb,size_t size,int f)
Author: Potash Corporation: Balasz Nemeth, given to CWP 2008
```

```
XCOR - Compute z = x cross-correlated with y

xcor compute z= x cross-correlated with y

Function Prototype:
void xcor (int lx, int ifx, float *x, int ly, int ify, float *y ,
int lz, int ifz, float *z);

Input:
lx length of x array
ifx sample index of first x
x array[lx] to be cross-correlated with y
ly length of y array
ify sample index of first y
y array[ly] with which x is to be cross-correlated
lz length of z array
ifz sample index of first z

Output:
z array[lz] containing x cross-correlated with y

Notes:
See notes for convolution function convolve_cwp().

The operation "x cross correlated with y"  is defined to be:

           ifx+lx-1
    z[i] =   sum    x[j]*y[i+j]  ;  i = ifz,...,ifz+lz-1
            j=ifx

This function performs cross-correlation by
(1) reversing the samples in the x array while copying
    them to a temporary array, and
(2) calling function convolve_cwp() with ifx set to 1-ifx-lx.
Assuming that the overhead of reversing the samples in x is negligible,
this method enables cross-correlation to be performed as efficiently as
convolution, while reducing the amount of code that must be optimized
and maintained.

Author:  Dave Hale, Colorado School of Mines, 11/23/91
```

```
XINDEX - determine index of x with respect to an array of x values

xindex determine index of x with respect to an array of x values

Input:
nx number of x values in array ax
ax array[nx] of monotonically increasing or decreasing x values
x the value for which index is to be determined
index index determined previously (used to begin search)

Output:
index for monotonically increasing ax values, the largest index
for which ax[index]<=x, except index=0 if ax[0]>x;
for monotonically decreasing ax values, the largest index
for which ax[index]>=x, except index=0 if ax[0]<x

Notes:
This function is designed to be particularly efficient when called
repeatedly for slightly changing x values; in such cases, the index
returned from one call should be used in the next.

Author:  Dave Hale, Colorado School of Mines, 12/25/89
```

```
YCLIP - Clip a function y(x) defined by linear interpolation of the
uniformly sampled values:  y(fx), y(fx+dx), ..., y(fx+(nx-1)*dx).
Returns the number of samples in the clipped function.

yclip clip a function y(x) defined by linear interplolation of
uniformly sampled values

Function Prototype:
int yclip (int nx, float dx, float fx, float y[], float ymin, float ymax,
float xc[], float yc[]);

Input:
nx number of x (and y) values
dx x sampling interval
fx first x
y array[nx] of uniformly sampled y(x) values
ymin minimum y value; must not be greater than ymax
ymax maximum y value; must not be less than ymin

Output:
xc array[?] of x values for clipped y(x)
yc array[?] of y values for clipped y(x)

Returned: number of samples in output arrays xc and yc

Notes:
The output arrays xc and yc should contain space 2*nx values, which
is the maximum possible number (nc) of xc and yc returned.

Author:  Dave Hale, Colorado School of Mines, 07/03/89
```

```
YXTOXY - Compute a regularly-sampled, monotonically increasing function x(y)
from a regularly-sampled, monotonically increasing function y(x) by
inverse linear interpolation.

yxtoxy compute a regularly sampled function x(y) from a regularly
sampled, monotonically increasing function y(x)

Function Prototype:
void yxtoxy (int nx, float dx, float fx, float y[],
int ny, float dy, float fy, float xylo, float xyhi, float x[]);

Input:
nx number of samples of y(x)
dx x sampling interval; dx>0.0 is required
fx first x
y array[nx] of y(x) values; y[0] < y[1] < ... < y[nx-1] required
ny number of samples of x(y)
dy y sampling interval; dy>0.0 is required
fy first y
xylo x value assigned to x(y) when y is less than smallest y(x)
xyhi x value assigned to x(y) when y is greater than largest y(x)

Output:
x array[ny] of x(y) values

Notes:
User must ensure that:
(1) dx>0.0 && dy>0.0
(2) y[0] < y[1] < ... < y[nx-1]

Author:  Dave Hale, Colorado School of Mines, 06/02/89
```

```
ZASC - routine to translate ncharacters from ebcdic to ascii

zasc - convert n characters from ebcdic to ascii format

Input:
nchar number of characters to be translated
ainput pointer to input characters

Output:
aoutput pointer to output characters

Function Prototype:
int zasc(char *ainput, char *aoutput, integer nchar);
Notes:
 translated by f2c.  Horribly inefficient, but little used

Author: Stew Levin of Mobil, 1997
```

```
ZEBC - routine to translate ncharacters from ascii to ebcdic

zebc - convert n characters from ascii to ebcdic format

Input:
nchar number of characters to be translated
ainput pointer to input characters

Output:
aoutput pointer to output characters

Function Prototype:
int zebc(char *ainput, char *aoutput, integer nchar);
Notes:
 translated by f2c.  Horribly inefficient, but little used

Author: Stew Levin of Mobil, 1997
```

CHECK - CHECK triangulated models

badModel bad Model flag
checkVertexUse check Vertex Use
checkEdgeUse check Edge Use
checkFace check Face
checkModel check Model

Function Prototypes:

```
void badModel(void);
void checkVertexUse (VertexUse *vu);
void checkEdgeUse (EdgeUse *eu);
void checkFace (Face *f);
void checkModel (Model *m);
```

checkVertexUse:
Input:
vu Pointer to VertexUse

checkEdgeUse:
Input:
eu pointer to EdgeUse

checkFace:
Input:
f pointer to Face

checkModel:
Input:
m pointer to Model

Notes: Routines for checking triangulated models.

Author: Dave Hale, Colorado School of Mines, 07/09/90

```
CIRCUM - define CIRCUMcircles for Delaunay triangulation

circum - compute center and radius-squared of circumcircle of 3 (x,y)
          locations
circumTri - compute center and radius-squared of circumcircle of
            triangular face

Function Prototypes:
void circum (float x1, float y1, float x2, float y2, float x3, float y3,
float *xc, float *yc, float *rs);
void circumTri (Tri *t);
circum:
Input:
x1 x-coordinate of first point
y1 y-coordinate of first point
x2 x-coordinate of second point
y2 y-coordinate of second point
x3 x-coordinate of third point
y3 y-coordinate of third point

Output:
xc pointer to x-coordinate of center of circumcircle
yc pointer to y-coordinate of center of circumcircle
rs pointer radius^2 of circumcircle

circumTri:
Input:
t Pointer to Tri

Returns:
xc x-coordinate of circumcircle
yc      y-coordinate of circumcircle
rs      radius^2 of circumcircle

Author:  Dave Hale, Colorado School of Mines, Fall 1990.
```

```
COLINEAR - determine if edges or vertecies are COLINEAR in triangulated
           model
edgesColinear see whether or not two edges are colinear
vertexBetweenVertices determine whether or not a vertex is on a line
                          between two other vertices
Function Prototypes:
int edgesColinear (Edge *e1, Edge *e2);
int vertexBetweenVertices (Vertex *v, Vertex *v1, Vertex *v2);

edgesColinear:
Input:
e1 pointer to first Edge
e2 pointer to second Edge

Returns: (int)
1 if colinear

vertexBetweenVertices:
Input:
v pointer to first Vertex in question
v1 pointer to first reference Vertex
v2 pointer to second reference Vertex

Returns: integer
1 if v=v1 or v=v2 or if v is between v1 and v2
0 otherwise

Author:  Dave Hale, Colorado School of Mines, Fall 1990.
```

```
CREATE - create model, boundary edge triangles, edge face, edge vertex, add
         a vertex

makeModel Make and return a pointer to a new model
makeBoundaryEdgeTri Create a boundary edge and triangle
makeEdgeFace Create an edge by connecting two vertices
makeEdgeVertex Create an edge connecting an existing vertex (v1) to a
                        new vertex
addVertexToModel Add a vertex to model, and return pointer to new vertex
insideTriInModel return pointer to triangle in model containing
                    specified (x,y) coordinates

Function Prototypes:
Model *makeModel (float xmin, float ymin, float xmax, float ymax);
void makeBoundaryEdgeTri (Vertex *v, Edge **enew, Tri **tnew);
void makeEdgeFace (Vertex *v1, Vertex *v2, Edge **enew, Face **fnew);
Vertex* addVertexToModel (Model *m, float x, float y);
Tri* insideTriInModel (Model *m, Tri *start, float x, float y);

makeModel:
Input:
xmin minimum x-coordinate
ymin minimum y-coordinate
xmax maximum x-coordinate
ymax maximum y-coordinate

Returns: pointer to a new Model

makeBoundaryEdgeTri:
Input:
v specified boundary Vertex

Output:
enew new boundary Edge
tnew new boundary triangle

Notes:
The specified vertex and the adjacent vertices on the boundary
are assumed to be colinear.  Therefore, the resulting
boundary triangle has zero area, and is intended to enable
deletion of the specified vertex from the boundary.

makeEdgeFace:
```

```
Input:
v1 First Vertex
v2 second Vertex

Output:
enew new Edge
fnew new Face

Notes:
The vertices must be adjacent to a single common face.
This face is closed off by the new edge, and a new edge and
a new face are made and returned.

addVertexToModel:
Input:
m model
x x-coordinate of new vertex
y y-coordinate of new vertex

Notes:
If the new vertex is close to an existing vertex, this function returns NULL.

insideTriInModel:
Input:
m Model
start triangle to look at first (NULL to begin looking anywhere)
x x-coordinate
y y-coordinate

Notes:
Points on an edge of a triangle are assumed to be inside that triangle.
An edge may be used by two triangles, so two triangles may "contain"
a point that lies on an edge.  The first triangle found to contain
the specified point is returned.

Author:  Dave Hale, Colorado School of Mines, Fall 1990.
```

```
DELETE - DELETE vertex, model, edge, or boundary edge from triangulated model

deleteVertexFromModel Delete a vertex from model
killModel Delete a model along with everything in it
killEdge Delete an edge
killBoundaryEdge Kill a boundary edge

Function Prototypes:
void deleteVertexFromModel (Model *m, Vertex *v);
void killModel (Model *m);
void killEdge (Edge *e, Face **fs);
void killBoundaryEdge (Edge *e);

deleteVertexFromModel:
Input:
m   pointer to Model
v pointer to Vertex to be deleted

killModel:
Input:
m pointer to Model

killEdge:
Input:
e Edge to delete

Output:
fs surviving Face

killBoundaryEdge:
Input:
e  boundary Edge

Notes:
Killing a boundary edge is typically done after a new boundary vertex
is inserted on an existing boundary edge.

Author:  Dave Hale, Colorado School of Mines, Fall 1990.
```

```
DTE - Distance to Edge

distanceToEdge - return distance to edge from specified (x,y) coordinates

Function Prototype:
float distanceToEdge (Edge *e, float x, float y);
distanceToEdge:
Input:
e edge to which distance is to be computed
x x-coordinate
y y-coordinate

Author:  Dave Hale, Colorado School of Mines, 09/11/90
```

```
FIXEDGES - FIX or unFIX EDGES between verticies

fixEdgeBetweenVertices Fix edge(s) between vertices, creating new
                              colinear edges as necessary.
unfixEdge unfix edge

fixEdgesBetweenVertices:
Input:
v1 pointer to first Vertex
v2 pointer to second Vertex

Returns:
0 if unable to fix edges
1 otherwise

unfixEdge:
Input:
e edge to be unfixed

Returns:
0 if unable to unfix edge
1 otherwise
Author:  Dave Hale, Colorado School of Mines, 06/04/91
```

```
INSIDE -  Is a vertex or point inside a circum circle, etc. of a triangulated
          model

inCircum determine whether or not a vertex is inside a circum circle
inCircumTri determine whether or not a vertex is inside a circum circle of
                 a triangle
in3Vertices determine whether or not a vertex is inside triangle (v1,v2,v3)
inTri determine whether or not a vertex is inside a triangle

Function Prototypes:
int inCircum (float x, float y, float xc, float yc, float rs);
int inCircumTri (float x, float y, Tri *t);
int in3Vertices (float x, float y, Vertex *v1, Vertex *v2, Vertex *v3);
int inTri (float x, float y, Tri *t);
inCircum:
Input:
x x-coordinate of vertex
y y-coordinate of vertex
xc x-coordinate of center of circumcircle
yc y-coordinate of center of circumcircle
rs radius^2 of circumcircle

Returns:
1 if x,y inside of circumcircle
0 otherwise

Notes:
A vertex exactly on the edge of a circumcircle is taken as being outside

inCircumTri:
Input:
x x-coordinate of vertex
y y-coordinate of vertex
t pointer to Tri

Returns:
1 if x,y inside of circumcircle of a triangle
0 otherwise

Notes:
A vertex exactly on the edge of a circumcircle is taken as being outside

in3Vertices:
```

```
Input:
x x-coordinate of vertex
y y-coordinate of vertex
v1 pointer to first Vertex
v2 pointer to second Vertex
v3 pointer to third Vertex

Returns:
1 if x,y inside of v1,v2,v3
0 otherwise

Notes:
A vertex exactly on an edge of the triangle is taken as being inside

inTri:
Input:
x x-coordinate of vertex
y y-coordinate of vertex
t pointer to Tri

Returns:
1 if x,y inside a triangle
0 otherwise
Notes:
A vertex exactly on the edge of a triangle is inside

Author:  Dave Hale, Colorado School of Mines, 06/04/91
```

```
NEAREST - NEAREST edge or vertex in triangulated model

nearestEdgeInModel Return pointer to edge in model nearest to
                     specified (x,y) coordinates

nearestVertexInModel Return pointer to vertex in model nearest
                    to specified (x,y) coordinates

Function Prototypes:
Vertex* nearestVertexInModel (Model *m, Vertex *start, float x, float y);
Edge* nearestEdgeInModel (Model *m, Edge *start, float x, float y);

nearestEdgeInModel:
Input:
m model
start edge to look at first (NULL to begin looking anywhere)
x x-coordinate
y y-coordinate

Returns: pointer to nearest Edge

nearestVertexInModel:
Input:
m model
start vertex to look at first (NULL to begin looking anywhere)
x x-coordinate
y y-coordinate

Returns: pointer to nearest Vertex

Author:  Dave Hale, Colorado School of Mines, Fall 1990
```

```
PROJECT - project to edge in triangulated model

projectToEdge - Project point with specified (x,y) coordinates to specified
                edge
Function Prototype:
void projectToEdge (Edge *e, float *x, float *y)

Input:
e edge to which point is to be projected
x x-coordinate before projection
y y-coordinate before projection

Output:
x x-coordinate after projection
y y-coordinate after projection

Author:  Dave Hale, Colorado School of Mines, 09/11/90
```

```
READWRITE - READ or WRITE a triangulated model

readModel Read a model in the form produced by writeModel
writeModel Write a model to a file

Function Prototypes:
Model *readModel (FILE *fp);
void writeModel (Model *m, FILE *fp);

readModel:
Input:
fp file pointer to file containing model

writeModel:
Input:
m pointer to Model
fp file pointer

Author:  Jack K. Cohen, Center for Wave Phenomena, 09/21/90
Modified:  Dave Hale, Center for Wave Phenomena, 11/30/90
Converted representation of model from ascii to binary for speed.
Added code to read attributes.
Modified (writeModel):  Craig Artley, Center for Wave Phenomena, 04/08/94
Corrected bug;  previously the edgeuses and vertextuses of the
exterior face were not written.
```