AsturFit user's and reference guide

Author: Víctor Luaña (VLC) and Alberto Otero-de-la-Roza (AOR)
Contact: victor@carbono.quimica.uniovi.es
Contact: aoterodelaroza@gmail.com
Address: Departamento de Química Física y Analítica, Universidad de Oviedo,
Principado de Asturias,
Julián Clavería 8, 33007 Oviedo, Spain
Version: 1.0 (2010-12-18)

Contents

1 1 Introduction

AsturFit (see [fit1]) is a collection of octave routines developed with the purpose of fitting energy versus volume theoretical data and determine the equilibrium properties of crystals and the derivatives of the E(V) curve. The routines can be used interactively or they can form part of a sophisticated script.

Several well known Equations of State (EOS) can be fitted to the data, like the Birch-Murnaghan and Poirier-Tarantola families, Vinet et al EOS, or Holzapfel's AP2 form. The fitting strategy that we recommend is, however, the average of a strain polynomial family [fit1]. In this way not only the best information available from the can be extracted, but the error in the fitting is also obtained, thus providing a detailed control of the procedure.

The AsturFit package contains also techniques able to detect, and eventually eliminate, common problems of the theoretical datafiles, like the presence of a few outliers and jumps in the energy. The techniques are discussed in refs. [fit2] and [fit1].

The package is made of three main tasks:

- allfits: fit many different EOS to a dataset.
- asturfit: the recommended fitting task.
- checknoise: analysis of the possible noise in a dataset.

These three standalone codes call to the different routines:

- avgstrainfit: fit an average of strain polynomials to some E(V) data.
- errformatf: convert a number, x, and error, dx, to a compact x(dx) form.
- figures: determine the number of significant figures of a number.
- guessjumps3: check for the existence of jumps in some curve.
- nlf: non-linear fitting of several EOS.
- noisify: add artificial noise to a curve.
- readEVdata: read a file with E(V) data.
- strain2volume: convert several strain forms into the volume values.
- strainbootstrap: bootstrap resampling applied to a strain polynomial of fixed degree or to an average of strain polynomials.
- strainevalE: evaluate the energy for a strain polynomial.
- straineval: evaluate a collection of properties, including the energy, for a strain polynomial.
- strainfit: fit a strain polynomial of fixed degree to some E(V) data.
- strainjumpfit: iterative fitting to a strain polynomial and to a step function. The guessjumps3 routine does a better and simpler job.
- strainmin: get the minimum of a strain E(V) polynomial.
- strainplot: plot an E(V) or E(f) curve.
- strainspinodal: determine the spinodal point of a E(V) curve.
- strainstepevalE: evaluate a strain polynomial with jumps.
- volume2strain: convert volumes into one of several strain forms.

The user can also call to the independent rutines to produce a complex run not available in the standalone tasks. We have provided a set of tests that can be examined to see the package in action and learn by the example the use of the routines.

1.1 1.1 Performing the standard fits

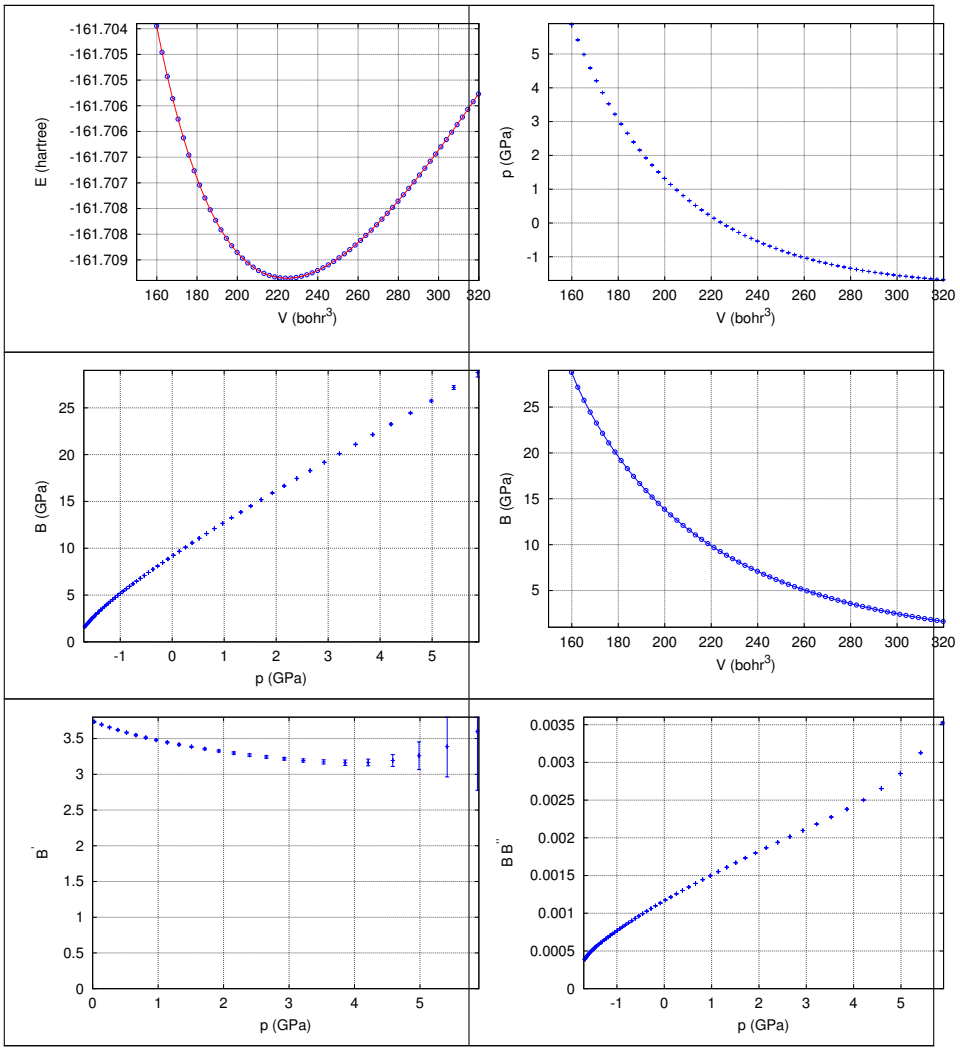
Three high level tasks have been designed with the purpose of producing the most relevant tasks in a very simple way. The first task:

```
asturfit filedat1 filedat2 ... > logfile
```

will perform an standard analysis for all the indicated data files. This task includes:

- checking if the file suffers from some type of noise;
- determining the best degree for the average of eulerian strain polynomials;
- printing the equilibrium properties, including the estimated standard error of the fitting;
- producing a set of plots of the most important properties: p(V), B(V), B(p), ...;
- producing a output file with the most relevant properties, in a format useful for parsing with further programs, like gnuplot.

The asturfit task automaticaly generates a set of the most useful plots.



Similarly, the second task:

```
allfits filedat1 filedat2 ... > logfile
```

will apply a selection of all the types of nonlinear and linear EOS fittings to everyone of the indicated data files. And the third task:

```
checknoise filedat1 filedat2 ... > logfile
```

will analyze the datafiles searching for noise in the form of separated outlier points or sets of neighbor points forming stairs.

1.2 1.2 Installation of the package on a unix-like operating system

The asturfit package is distributed as a single compressed tar file. Let us assume that you want to install it in one of your personal directories, for instance in "~/src/". Then you should

```
mv asturfit.tgz ~/src
cd ~/src
tar xtvf asturfit.tgz
```

This will create the tree of directories

```
~/src/asturfit/doc/
~/src/asturfit/src/
~/src/asturfit/test/
```

`doc` contains this documentation, `src` is the home of all the routines forming the package, and `test` contains a set of scripts and data files for testing. The `src` directory must be added to the path where octave looks for files and routines. This can be done adding the next line to the `~/.octaverc` configuration file:

```
addpath("~/src/asturfit/src/");
```

Finally, we have found very useful to create a symbolic link of the main tasks in the binary directory where most of the personal executables can be found

```
cd ~/bin
ln -s ~/src/asturfit/src/asturfit.m asturfit
ln -s ~/src/asturfit/src/allfits.m allfits
ln -s ~/src/asturfit/src/checknoise.m checknoise
```

In this way, just by making sure that `~/bin` is included in the path of binaries, we can use asturfit and the rest of tasks on any working directory.

The package can also be installed in a system directory for the access of all the users and added to the general octave path.

Asturfit (1.0) has been designed for octave, versions 3.0 and 3.2.

1.3 1.3 Citation of this package

Please, consider citing ref. [fit1] if you find the AsturFit package useful for your work. Some of the techniques, particularly the treatment of noisy data, were discussed previously in ref. [fit2]. The average of polynomials was first discussed in the PhD work of Miguel Álvarez Blanco [fit3], and it was used systematically within the gibbs code [fit4], that implements a quasi harmonic Debye model for the estimation of thermal effects on crystals. Notice that the linear fitting within Debye is not as robust as it should, and suffers from numerical instability for high degree polynomials. The new version of gibbs [fit5] uses the techniques developed with AsturFit and it is a robust code with a much improved thermodynamic treatment.

1.4 1.4 Compatibility with MatLab

AsturFit is made of octave routines and scripts. No attempt has been made to ensure compatibility with MatLab. However, we anticipate only a few potential problems:

(1) the ”endfunction”, ”endfor”, and ”endif” should be converted to simple ”end”s;

(2) the ”leasqr()” nonlinear fitting routine (used inside ”nlf.m”) should be adapted to the use of a MatLab equivalent.

(3) the ”gamma_inc()” should also be adapted. Notice that the gammai(a,b) routine, formally the required gamma incomplete function, fails to work for integer and negative values of a, that are required for Holzapfel’s AP2 EOS.

If you adapt AsturFit to MatLab or work with it on a MSWindows environment we would like to hear from your experience.

2 2 Format of datafiles

The form of data files is very simple: two columns, the first with volumes, the second with energy values. Blank lines are allowed as well as comments starting with the "#" character. The only special feature is that the comment lines can have declarations of the units used in the data file. The code understands:

- the energy unit (eV, hartree, rydberg, ...) that is converted to hartree, the default unit used in AsturFit.
- the volume unit (bohr^3, angstrom^3, ...) converted to the default bohr^3.
- the value of "Z", the number of molecules per unit cell of the crystal. As a default, AsturFit works with the energy and volume per molecule, and it is assumed to read values per cell.
- the value of "nelectrons", the number of electrons in a molecule. This value is used in the AP2 equation of state and it is transmitted in the "global nelectrons" declaration.

The data file reading is done by the "readEVdata()" function, and the test01 performs a very detailed inspection of its capabilities. A data file example would be:

```
# octave
# Calculation on an unknown crystal
# volume ang^3
# energy ry
# nelectrons 2
# z 4
31.5176 -14.5664202
32.4533 -14.5804702
33.4073 -14.5938196
34.3799 -14.6065011
35.3711 -14.6185620
36.3812 -14.6300054
37.4103 -14.6408790
```

3 3 Some examples

3.1 3.1 Test01: Checking the readEVdata routine

The routine readEVdata() has been designed as the natural data reading method of the asturfit package. In addition to reading the volume and energy, the routine recognizes several variables and data within the comments, typically at the beginning of the data file. This test reads a standard datafile and then copies of the same data from files that contain potentially problematic things, like blank an comment lines within the body of the data, strange keywords in the header, etc. If things work as they should, octave will read identical data from all the sources and the consistency test will pass:

```
octave -q test01.m
Test 1 (test01a.dat): pass
Test 2 (test01b.dat): pass
Test 3 (test01c.dat): pass
Test 4 (test01d.dat): pass
Test 5 (test01e.dat): pass
Test 6 (test01f.dat): pass
Test 7 (test01g.dat): pass
Test 8 (test01h.dat): pass
```

It is interesting to analyze the test itself:

```
# octave
% This is a test of the correct behavior of the readEVdata() routine.

global nelectrons
```

```
addpath("../src/");

# test list
tests = {"test01a.dat", # original source
         "test01b.dat", # spaces and tabs in header and data
         "test01c.dat", # strange keywords in header
         "test01d.dat", # lines inside the body of the data
         "test01e.dat", # spurious fields in header and data
         "test01f.dat", # original source with DOS line terminators
         "test01g.dat", # original source with MAC line terminators
         "test01h.dat",
         };

# reference
load "test01_ref.dat";
angtobohr = 1.88972613288564;
vref = test01_ref(:,1) * angtobohr^3;
eref = test01_ref(:,2) / 2 * 4;
nelectronsref = 2;
clear test01_ref;

# do the tests
for i = 1:length(tests)
   [v,e] = readEVdata(tests{i},0);
   if (isequal(v,vref) && isequal(e,eref) && nelectrons==nelectronsref)
      str = "pass";
   else
      str = "fail";
   endif
   printf("Test %d (%s): %s \n",i,tests{i},str);
endfor
```

3.2 3.2 Test02: A rather complete analysis of a datafile

A rather complex task with a number of details that deserve comment:

(1) This test has been the base for the standard asturfit run.

(2) test02.m will work both as a routine that can be called within an interactive octave session and also as an independent script that can be executed from the operating system. To achieve the first role the file "test02.m" starts with the function "test02", so octave will recognize the routine. To achieve the second role, the "argn" and "nargin" internal variables are tested outside the function. In the course of an standalone run "argn" will be zero and "nargin" will be set to the number of datafiles.

(3) The routine tries to determine the best degree to stop averaging polynomials. To this end, the routine examines the relative error of the volume, bulk modulus, and bulk modulus derivatives for a collection of fittings that start averaging up to 4th degree polynomials, then up to 5th degree, and so on. The increase in the maximum degree stops when the error in two or more of controlled properties increases significantly.

(4) Five different eps file plots are produced for each datafile. The on screen viewing is deactivated within the routine. Whereas the control offered by octave is not enough, in some cases, for producing a publishable plot, the result is quite useful for monitoring purposes.

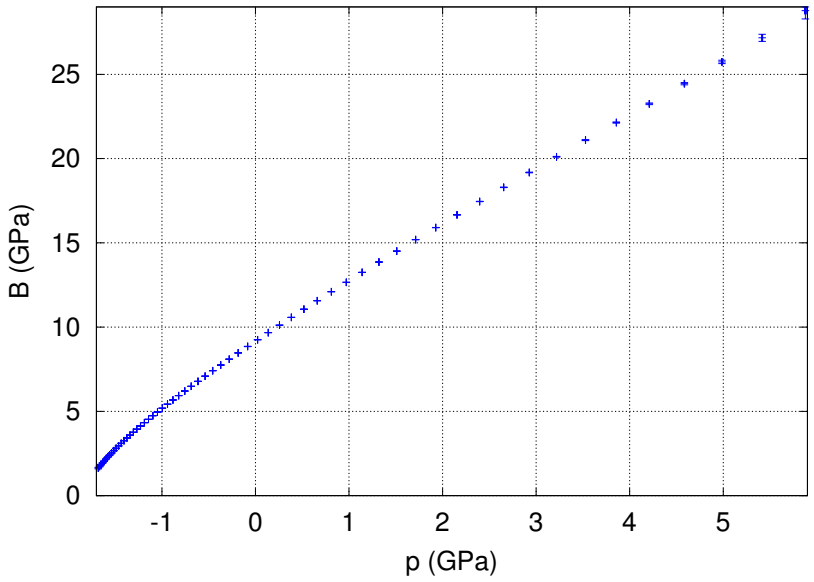


Figure 1: B(p) curve for the Na bcc phase. This is one of the plots produced by the test02.m script. There are error bars in both properties, but the bars are very small in most cases.

```
# octave
function status = test02 (filein, rootname)
%
% TEST02: Standard fit to a set of data files.
%
% Run as:   test02.m file1.dat file2.dat ...

global nelectrons
hybohr3togpa = 2*14710.50498740275538944426;
status = 0;

addpath("../src/");

printf("+++++++++++++++++++++++++++++++++\n");
printf("test02: filein and rootname --> %s, %s\n", filein, rootname);
printf("+++++++++++++++++++++++++++++++++\n");

[v,e] = readEVdata(filein,1);
vref = median(v);

# Strain fitting. Decide up to wich degree do the average.
n = 4;
newerr = ones(4,1)*1e10;
rk = [1 3 4 5];
strain = "eulerian";
printf("Fit to polynomials of %s strain\n", strain);
printf("Select the best number of polynomials to average:\n");
printf("Checking err(X) = (std(X)/mean(X))^2 for:\n");
printf(" --n-- --volume-- ----B----- ----B1p--- ----B2p---\n");
do
   olderr = newerr;
   [cf,sf] = avgstrainfit(v,e,vref,n,:,strain,0);
   newerr = (sf.eqstd(rk)./sf.eqmean(rk)).^2;
   printf("%6d", n);
   printf("  %.3e", newerr);
   printf("\n");
   n++;
until (sum(newerr > 1.5*olderr)>1 | n > 18)
[cf,sf] = avgstrainfit(v,e,vref,--n,:,strain,1);
```

```
# Figures will not appear on screen:
set(gcf(),"visible","off");

# Figures: E(V), p(V), B(p), B(V), B1(p), B*B2(p)
fileplot = sprintf("%sev.eps", rootname);
plot(v,e,'ob', v,sf.Efit,'-r');
xlabel('V (bohr^3)'); ylabel('E (hartree)'); grid('on');
print(fileplot, '-depsc');
printf("E(V) plot created in: %s\n", fileplot);

fileplot = sprintf("%spv.eps", rootname);
f = hybohr3togpa;
errorbar(v, sf.pmean*f, sf.pstd*f, '~');
xlabel('V (bohr^3)'); ylabel('p (GPa)'); grid('on');
print(fileplot, '-FHelvetica:22', '-depsc');
printf("p(V) plot created in: %s\n", fileplot);

fileplot = sprintf("%sbp.eps", rootname);
f = hybohr3togpa;
errorbar(sf.pmean*f, sf.Bmean*f, sf.pstd*f, sf.Bstd*f, '~>');
xlabel('p (GPa)'); ylabel('B (GPa)'); grid('on');
print(fileplot, '-depsc');
printf("B(p) plot created in: %s\n", fileplot);

fileplot = sprintf("%sbv.eps", rootname);
plot(v,sf.Bmean*f,'-ob');
xlabel('V (bohr^3)'); ylabel('B (GPa)'); grid('on');
print(fileplot, '-depsc');
printf("B(V) plot created in: %s\n", fileplot);

fileplot = sprintf("%sb1p.eps", rootname);
errorbar(sf.pmean*f, sf.B1pmean, sf.pstd*f, sf.B1pstd, '~>');
xlabel('p (GPa)'); ylabel("B^'"); grid('on');
print(fileplot, '-depsc');
printf("B1(p) plot created in: %s\n", fileplot);

fileplot = sprintf("%sbb2p.eps", rootname);
errorbar(sf.pmean*f, sf.Bmean.*sf.B1pmean, sf.pstd*f, sf.Bstd.*sf.B1pstd,
'~>');xlabel('p (GPa)'); ylabel("B B^{''}"); grid('on');
print(fileplot, '-depsc');
printf("B*B2p(p) plot created in: %s\n", fileplot);

# Reactivate the viewing on screen:
set(gcf(),"visible","on");

endfunction

## Use it as a script
if (!exist("argn"))
   if (nargin > 0)
      args = argv();
      for i = 1 : nargin
         test02(args{i}, sprintf("test02-c%02d-",i));
      endfor
   else
      printf("Use as: test02.m file(s)\n");
   endif
endif
```

3.3 3.3 Test03: Fit all the types of fitting included

AsturFit includes a collection of traditional EOS that are fitted non linearly to the E(V) data, and some polynomial strain EOS, for several definitions of the strain, that are fitted by solving linear equations. In this test all the techniques are applied over one or more data sets. Following a technique already described ”test03” can be used as a function within an octave session or as a standalone executable script. For instance,

```
test03.m w2k-lda-li.dat
```

will produce a long output from which we have taken, and edited to adapt the spacing, the next excerpt:

```
# octave
++++++++++++++++++++++++++++++
test03: filein and rootname --> w2k-lda-li.dat, test03-c01-
++++++++++++++++++++++++++++++
readEVdata: Reading input file -> w2k-lda-li.dat
Input volume unit: bohr^3
Input energy unit: ry
* Energy and volume ranges
  V-range (bohr^3) : [31.5176000000 , 251.9701000000] dV = 220.4525000000
  E-range (Ry) : [-7.4103176500 , -7.2832101000] dE = 0.1271075500
  Number of data points: 103


Non-linear fitting: BM3: 3rd order Birch-Murnaghan EOS
Parameters (4) start / converged
E0 (Hy)                 -7.410318        -7.410112
V0 (bohr^3)            128.319495       126.495155
B0 (GPa)                15.070313        14.834320
B1p                      3.357205         3.679199
Convergence (1=yes)? 1
Iterations: 5
SSerr, SStot, R2, 1-R2: 7.077958e-06 9.753421e-02 0.999927431027 7.26e-05
Correlation matrix of the parameters:
     1.000000
     0.294749     1.000000
    -0.608654    -0.815611     1.000000
     0.619735     0.362550    -0.820253     1.000000

Non-linear fitting: BM4: 4th order Birch-Murnaghan EOS
Parameters (5) start / converged
E0 (Hy)                 -7.410318        -7.410326
V0 (bohr^3)            128.319495       127.670979
B0 (GPa)                15.070313        15.327032
B1p                      3.357205         3.463365
B2p (1/GPa)             -0.178271        -0.218486
Convergence (1=yes)? 1
Iterations: 6
SSerr, SStot, R2, 1-R2: 3.511925e-07 9.753421e-02 0.999996399290 3.60e-06
Correlation matrix of the parameters:
     1.000000
    -0.092325     1.000000
    -0.703806    -0.240639     1.000000
     0.558956    -0.565995    -0.610126     1.000000
    -0.664088     0.254684     0.838697    -0.937838     1.000000
[......]

++++++++++++++++++++++++++++++++++++++++
Fit to polynomials of eulerian strain
++++++++++++++++++++++++++++++++++++++++
Select the best number of polynomials to average:
```

```
Checking err(X) = (std(X)/mean(X))^2 for:
 --n-- --volume-- ----B----- ----B1p--- ----B2p---
     4  1.982e-05  2.459e-04  8.617e-04  3.715e-03
     5  4.054e-06  1.444e-05  1.743e-04  2.218e-03
     6  1.236e-07  8.300e-06  5.861e-06  7.772e-04
     7  1.535e-08  2.273e-06  9.345e-07  2.399e-04
     8  5.628e-09  1.155e-06  1.271e-06  1.184e-04
     9  5.587e-09  7.776e-07  1.885e-06  7.785e-05
    10  4.787e-09  6.557e-07  1.821e-06  7.915e-05
    11  4.687e-09  6.328e-07  2.226e-06  1.200e-04

avgpolyfit: AVERAGE OF eulerian STRAIN POLYNOMIALS
Volume reference (V0): 106.323900
Range of degrees: 2--11
Number of polynomials: 10
Properties at the minimum of each polynomial:
i- pol data par -SSerr-- ---w---- ---Vmin--- --Emin--- --Bmin--- -B1min-- ---B2min---
 1   7  103   8 1.73e-10 0.175640 128.320978 -7.410319 15.076740 3.356396 -0.182539454
 2   8  103   9 1.57e-10 0.175296 128.314562 -7.410319 15.080917 3.359675 -0.184018715
 3   9  103  10 1.46e-10 0.174658 128.316375 -7.410318 15.073954 3.359028 -0.181595351
 4  10  103  11 1.35e-10 0.174389 128.312400 -7.410318 15.068134 3.362834 -0.179569158
 5   6  103   7 2.35e-10 0.172350 128.331294 -7.410319 15.079508 3.351434 -0.182540895
 6   5  103   6 9.91e-10 0.126514 128.335781 -7.410322 15.107342 3.347302 -0.186176120
 7   4  103   5 1.42e-08 0.001153 128.240909 -7.410330 15.199074 3.364375 -0.197318194
 8   3  103   4 3.51e-07 0.000000 127.670979 -7.410326 15.327032 3.463365 -0.218485797
 9   1  103   2 2.24e-04 0.000000 130.719151 -7.408701 10.394788 4.000000 -0.374119122
10   2  103   3 7.08e-06 0.000000 126.495155 -7.410112 14.834320 3.679199 -0.247466776

Average (mode 1) properties:
------ ---volume-- ---energy-- --B-(GPa)-- ----B1p---- B2p-(1/GPa) B3p--(1/GPa^2)
-mean-  128.321112   -7.410319   15.079975    3.356557   -0.182593    0.081639331
stdvev    0.008785    0.000001    0.011996    0.005008    0.002000    0.002004795
[......]
```

Let us examine the test. An important consideration is that the "leasqr()" function from the "optim" package and the "gamma_inc" from the "gsl" package, both included in octaveforge, must be correctly installed.

```
# octave
#! /usr/bin/octave -q

function status = test03 (filein, rootname)
%
% TEST03: Perform a complete set of fittings to a set of data files.
%
% Run as:   test03.m file1.dat file2.dat ...

global nelectrons
hybohr3togpa = 2*14710.50498740275538944426;
status = 0;

addpath("../src/");

printf("++++++++++++++++++++++++++++++++\n");
printf("test03: filein and rootname --> %s, %s\n", filein, rootname);
printf("++++++++++++++++++++++++++++++++\n");

[v,e] = readEVdata(filein,1);
vref = median(v);

nltypes = {'bm3', 'bm4', 'bm5', 'pt3', 'pt4', 'pt5', 'murn', 'ap2'};
for i = 1 : length(nltypes)
```

```
   nlres{i} = nlf(v,e,nltypes{i},[],1,1);
endfor

sttypes = {'eulerian', 'natural', 'lagrangian', 'infinitesimal'};
for i = 1 : length(sttypes)
   # Strain fitting. Decide up to wich degree do the average.
   n = 4;
   newerr = ones(4,1)*1e10;
   rk = [1 3 4 5];
   strain = sttypes{i};
   printf("\n\n");
   printf("+++++++++++++++++++++++++++++++++++++++\n");
   printf("Fit to polynomials of %s strain\n", strain);
   printf("+++++++++++++++++++++++++++++++++++++++\n");
   printf("Select the best number of polynomials to average:\n");
   printf("Checking err(X) = (std(X)/mean(X))^2 for:\n");
   printf(" --n-- --volume-- ----B----- ----B1p--- ----B2p---\n");
   do
      olderr = newerr;
      [cf,sf] = avgstrainfit(v,e,vref,n,:,strain,0);
      newerr = (sf.eqstd(rk)./sf.eqmean(rk)).^2;
      printf("%6d", n);
      printf("  %.3e", newerr);
      printf("\n");
      n++;
   until (sum(newerr > 1.2*olderr)>1 | n > 18)
   [cf,sf] = avgstrainfit(v,e,vref,--n,:,strain,1);
endfor

endfunction

## Use it as a script
if (!exist("argn"))
   if (nargin > 0)
      args = argv();
      for i = 1 : nargin
         test03(args{i}, sprintf("test03-c%02d-",i));
      endfor
   else
      printf('Use as: test03.m file(s)\n');
   endif
endif
```

3.4 3.4 Test04: Working with a collection of crystals

Comparing the properties of the bcc phase of the alkaline elements Na to Rb.

```
# octave
addpath("../src/");
global nelectrons

# elements
element = {"Na", "K", "Rb"};

hybohr3togpa = 2*14710.504987402755389444426;
fac = [1,1,hybohr3togpa,1,1/hybohr3togpa,1/hybohr3togpa^2];

# Table: equilibrium properties of the elements
printf("El");
printf(" -Eceil-");
printf(" --V0-(bohr^3)---");
```

```
printf(" --E0-(hartree)--");
printf(" ----B0-(GPa)----");
printf(" ------B1p0------");
printf(" --B2p0-(1/GPa)--");
printf("\n");
for i = 1 : length(element)
   el = tolower(element{i});
   filein = sprintf("w2k-lda-%s.dat", el);
   [v{i},e{i}] = readEVdata(filein,1);
   vref{i} = median(v{i});
   eceil{i} = ceil(max(e{i}));
   e{i} -= eceil{i};
   [cf{i},sf{i}] = avgstrainfit(v{i},e{i},vref{i},6,:,:,0);
   printf("%-2s", element{i});
   printf(" %7d", eceil{i});
   for k = 1 : 5
      printf(" %-16s", errformatf(sf{i}.eqmean(k)*fac(k), sf{i}.eqstd(k)*fac(k)));
   endfor
   printf("\n");
   rk = find(sf{i}.pmean > 0);
   vpos{i} = v{i}(rk);
endfor

# Figures will not appear on screen:
set(gcf(),"visible","off");

# Figure: E(V) for all the elements.
# We will make sure that all elements have the same number of points.
V = E = [];
for i = 1 : length(element)
   vv = linspace(min(v{i}), max(v{i}), 201)';
   ee = strainevalE(cf{i}, vref{i}, vv);
   V = [V, vv/sf{i}.eqmean(1)];
   E = [E, (ee-sf{i}.eqmean(2))/(sf{i}.eqmean(1)*sf{i}.eqmean(3))];
   key{i} = sprintf("-;%s;", element{i});
endfor
fileplot = "w2k-lda-ev.eps";
plot(V,E,key);
xlabel('V/V_0'); ylabel('(E-E_0)/(B_0V_0)'); grid('on');
print(fileplot, '-depsc');
printf("E(V/V_0) plot created in: %s\n", fileplot);

# Figures: B(p), B(V), B1p(p), B*B2p(p)
V = P = B = B1 = BB2 = [];
npts = 201; rk = 1:npts;
for i = 1 : length(element)
   vv = linspace(min(vpos{i}), max(vpos{i}), npts)';
   vv = vv(rk);
   s = straineval(cf{i}, vref{i}, vv);
   V = [V, vv/sf{i}.eqmean(1)];
   P = [P, s.p*hybohr3togpa];
   B = [B, s.B*hybohr3togpa];
   B1 = [B1, s.B1p];
   BB2 = [BB2, s.B.*s.B2p];
   key{i} = sprintf("-;%s;", element{i});
endfor

fileplot = "w2k-lda-bp.eps";
plot(P,B,key);
xlabel('p_{static} (GPa)'); ylabel('B (GPa)'); grid('on');
print(fileplot, '-depsc');
printf("B(p) plot created in: %s\n", fileplot);
```

```
fileplot = "w2k-lda-bv.eps";
plot(V,B,key);
xlabel('V/V_0'); ylabel('B (GPa)'); grid('on');
print(fileplot, '-depsc');
printf("B(V/V_0) plot created in: %s\n", fileplot);

fileplot = "w2k-lda-b1p.eps";
plot(P,B1,key);
xlabel('p_{static} (GPa)'); ylabel("B^'"); grid('on');
print(fileplot, '-depsc');
printf("B1p(p) plot created in: %s\n", fileplot);

fileplot = "w2k-lda-b2pbp.eps";
plot(P,BB2,key);
xlabel('p_{static} (GPa)'); ylabel('B^{''}B'); grid('on');
print(fileplot, '-depsc');
printf("B2pB(p) plot created in: %s\n", fileplot);
```

3.5 3.5 Test05: Gluing together several datafiles

First we will read a dataset formed with points around the equilibrium geometry, and we will fit an average of Birch-Murnaghan (default) polynomials to determine the equilibrium properties. Then we will read two other datasets for very small and very large volumes. The equilibrium properties will feed a Vinet and a Holzapfel AP2 EOS, that will be compared with the real E(V) data for the whole volume range. Check the two plots that will be produced.

```
# octave
% Test: gluing together two data files and using an analytical EOS to
% extrapolate to smaller volumes.

global nelectrons

addpath("../src/");

# read the central dataset
[v1,e1] = readEVdata('mgo-sety1.dat',1);
vref = median(v1);

# determine the equilibrium properties
[cf,sf] = avgstrainfit(v1,e1,vref,:,:,:,1);
plot(v1,e1,'ob', v1,sf.Efit,'-r');
xlabel('V (bohr^3)'); ylabel('E (hartree)'); grid('on');
print('test05a.eps', '-FHelvetica:38', '-depsc');

# read the additional datasets for very small and very large volumes
[v2,e2] = readEVdata('mgo-sety1-compress.dat',1);
[v3,e3] = readEVdata('mgo-sety1-expand.dat',1);
v = [v2;v1;v3]; e = [e2;e1;e3];
# volumes are already sorted, otherwise:
# [v,isort] = sort(v); e = e(isort);

# evaluate (do not fit) Vinet and AP2 EOS with the equilibrium
# parameters on the whole volume range
pin = sf.eqmean([2,1,3,4])
rvinet = nlf(v, e, 'vinet', pin, 0, 1);
rap2 = nlf(v, e, 'ap2', pin, 0, 1);

plot(v,e,'ob;Data;', v,rvinet.Efit,'-r;Vinet;', v,rap2.Efit, '-g;AP2;');
xlabel('V (bohr^3)'); ylabel('E (hartree)'); grid('on');
print('test05b.eps', '-FHelvetica:38', '-depsc');
```

3.6 3.6 Test06: Working with a subset of the data

Octave offers a powerful and simple mechanism to select parts of a vector or matrix. For instance, "v(1:2:end)" and "v(1:4:end)" select one every two and every four elements of the vector "v", and "v(find(v>100 & v <300))" selects those elements that belong to the 100--300 range. The test06.m script uses this capability to check the stability of the equilibrium properties when the dataset becomes less dense but it maintains the range. We start with 129 points for the ground state of MgO and select and fit subsets of 65, 33, and 17.

Let us see first the result:

```
# octave
readEVdata: Reading input file -> mgo-sety1.dat
Input volume unit: bohr3
Input energy unit: ry
* Energy and volume ranges
  V-range (bohr^3) : [71.5195563798 , 143.0391127595] dV = 71.5195563796
  E-range (Ry) : [-17.1721409750 , -17.0038769900] dE = 0.1682639850
  Number of data points: 129
  Number of electrons: 20

-n- data --V-(bohr^3)-- --E-(hartree)- ---B-(GPa)--  ----B'----  B''(1/GPa)
  1  129  124.82338(86)  -17.172142(0)  169.316(22)  4.1629(34) -0.02516(30)
  2   65  124.82318(82)  -17.172142(0)  169.313(21)  4.1637(30) -0.02520(27)
  4   33  124.82332(87)  -17.172142(0)  169.309(19)  4.1643(31) -0.02522(32)
  8   17  124.8234(10)   -17.172142(0)  169.327(29)  4.1623(31) -0.02522(45)
```

The test:

```
# octave
% Test06: working with subsets of the initial data.
addpath("../src/");
hybohr3togpa = 2*14710.504987402755389444 26;

# read the initial dataset, former by 129 points
[v,e] = readEVdata('mgo-sety1.dat',1);
vref = median(v);

# select one every n points, for n being [1, 2, 4, 8]:
# determine the equilibrium properties for each set
f = [1, 1, hybohr3togpa, 1, 1/hybohr3togpa, 1/hybohr3togpa^2];
printf("-n- data --V-(bohr^3)-- --E-(hartree)- ---B-(GPa)----");
printf("  ----B'----  B''(1/GPa)-\n");
for n = [1, 2, 4, 8]
   vset = v(1:n:end); eset = e(1:n:end); nset = length(vset);
   [cf,sf] = avgstrainfit(vset,eset,vref,10,:,:,0);
   printf("%3d %4d ", n, nset);
   printf(" %-14s", errformatf(sf.eqmean(1), sf.eqstd(1)));
   printf(" %-14s", errformatf(sf.eqmean(2), sf.eqstd(2)));
   printf(" %-14s", errformatf(sf.eqmean(3)*f(3), sf.eqstd(3)*f(3)));
   printf(" %-10s", errformatf(sf.eqmean(4)*f(4), sf.eqstd(4)*f(4)));
   printf(" %-10s", errformatf(sf.eqmean(5)*f(5), sf.eqstd(5)*f(5)));
   printf("\n");
endfor
```

3.7 3.7 Test07: Analyzing a phase transition

Two or more phases of a crystal can be analyzed together and the transition induced by hydrostatic pressure can eventually be determined. The plot of the E(V) curves contains this information, but we must find the common tangent to the two curves and determine its slope (minus this slope is the transition pressure). Far easier is to plot the enthalpies versus pressure: the phase with smaller enthalpy is the more stable one under each pressure. It is even better to depict the enthalpy difference and

look for the zero (see the fugure). Our test is pretty basic. With some more work we could determine the actual value of the transition pressure and the properties of both phases at this point.

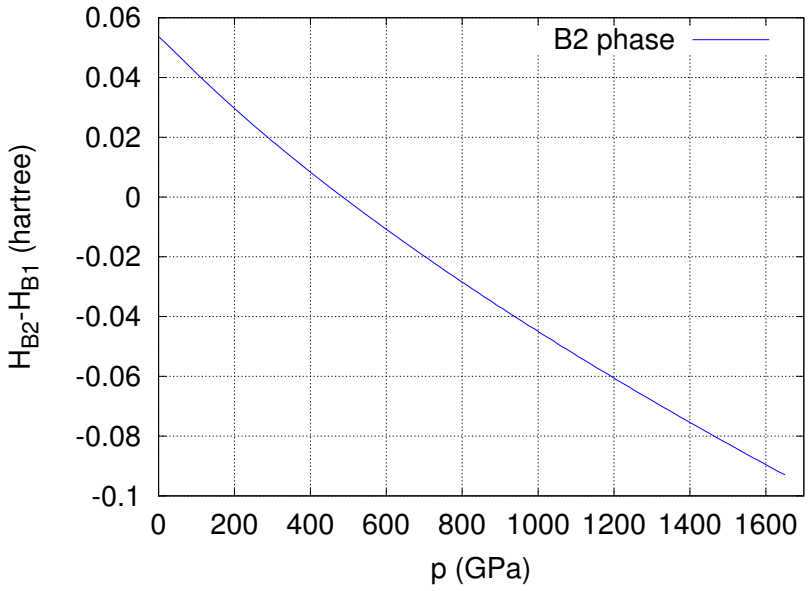


Figure 2: Relative stability of the B2 (CsCl like) and B1 (rock-salt) phases of MgO according to DFT calculations with the PBE functional.

```
# octave
% Test07: transition between two phases under high pressure
addpath("../src/");
global nelectrons
hybohr3togpa = 2*14710.50498740275538944426;

# read the data of the two phases
[vb1,eb1] = readEVdata('mgo-pbe-b1.dat',1);
vrefb1 = median(vb1);
[vb2,eb2] = readEVdata('mgo-pbe-b2.dat',1);
vrefb2 = median(vb2);

# fit both phases
vv = linspace(min(min(vb1),min(vb2)), max(max(vb1),max(vb2)), 201);
[cb1,sb1] = avgstrainfit(vb1,eb1,vrefb1,10,:,:,1);
[cb2,sb2] = avgstrainfit(vb2,eb2,vrefb2,10,:,:,1);

# Figures will not appear on screen:
set(gcf(),"visible","off");

# plot the E(V) curves:
plot(vb1,eb1,'or;;',vb1,sb1.Efit,'-r;B1 phase;'\
    ,vb2,eb2,'og;;',vb2,sb2.Efit,'-g;B2 phase;')
xlabel('V (bohr^3)'); ylabel('E (hartree)'); grid('on');
print('test07-ev.eps', '-FHelvetica:24', '-depsc');

# get the enthalpies and plot H(p) for both phases
hb1 = sb1.Efit + sb1.pmean .* vb1;
hb2 = sb2.Efit + sb2.pmean .* vb2;
pb1 = sb1.pmean*hybohr3togpa;
pb2 = sb2.pmean*hybohr3togpa;
plot(pb1,hb1,'-r;B1 phase;', pb2,hb2,'-g;B2 phase;');
xlabel('p (GPa)'); ylabel('H (hartree)'); grid('on');
print('test07-hp.eps', '-FHelvetica:24', '-depsc');

# It is more effective to print the enthalpy difference between both
```

```
# phases, but we must be sure that both are defined in the same grid
# of pressures. We use interpolation in the {pb1,hb1} and {pb2,hb2} sets.
pp = linspace(max([min(pb1),min(pb2),0]), min([max(pb1),max(pb2)]), 101)';
hhb1 = interp1(pb1, hb1, pp);
hhb2 = interp1(pb2, hb2, pp);
fileplot = 'test07-dhp.eps';
plot(pp, hhb2-hhb1, '-b;B2 phase;');
xlabel('p (GPa)'); ylabel('H_{B2}-H_{B1} (hartree)'); grid('on');
print(fileplot, '-FHelvetica:24', '-depsc');
printf('See the DifH(p) curve in the file: %s\n', fileplot);

# Reactivate the viewing on screen of the plots:
set(gcf(),"visible","on");
```

3.8 3.8 Test08: A dataset with noise (outliers)

For decades, solid-state physicists have strived for accuracy in their calculations. We are now going to take the opposite direction. In our quest for the detection of problems we needed to produce trouble under some control and that was the origin of the "noisify()" function. In this test we introduce a number of outliers, randomly distributed, in a conventional smooth data set. A bootstrap resampling of the dataset will produce, eventually, a number of clean samples that will stand out due to their better fitting errors. The probability of suceeding depends on the actual number of outliers and on the sampling size. For 10 outliers and a sample of 1000 there is a probability of 62.36% of success, and this increases up to 85.83% and 94.67% by doubling and tripling the sample, respectively. See refs. [fit1] and [fit2] for a more datailed discussion.

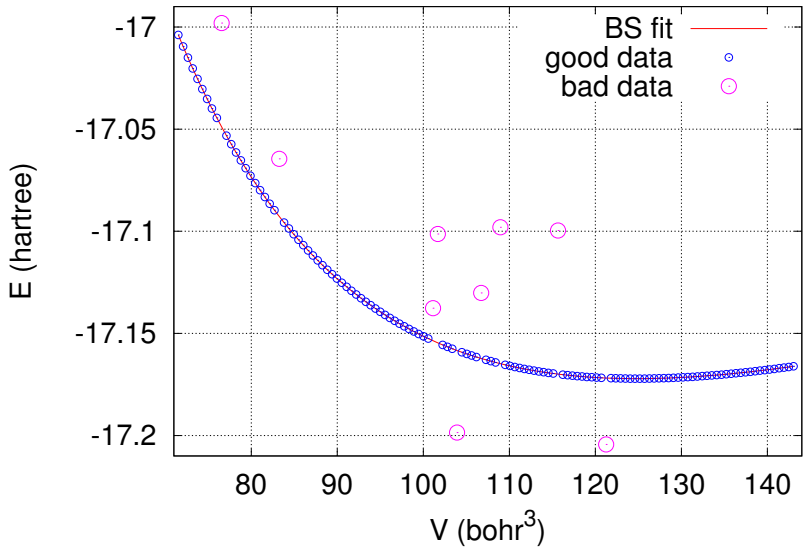


Figure 3: Outliers in an otherwise smooth data set can be detected with a bootstrap sampling plus average of polynomials fitting.

```
# octave
% Test08: Bootstrap fit to a data set with noise (disperse outlier points)
addpath("../src/");
global nelectrons
hybohr3togpa = 2*14710.50498740275538944426;

# read in a smooth data set
[v,e] = readEVdata('mgo-sety1.dat',1);
vref = median(v);

# fit to the smooth set
[c,s] = avgstrainfit(v,e,vref,10,:,:,1);
```

```
# now we add artificially some noise, only to the energy
noutliers = 10;
[vn,en] = noisify(v, e, noutliers, 0, 0.004, 1);

# lets use the bootstrap sampling and see if all outliers are detected
sample = 1000;
chance = (1-(1-0.5^noutliers)^sample);
printf("\n");
printf("Number of outliers (artificially added): %d\n", noutliers);
printf("Bootstrap sample size: %d\n", sample);
printf("Chance of finding all outliers is %.6e\n", chance);
[cb,sb] = strainbootstrap(vn, en, vref, :, sample, :, 1);

bad = sb.outliers;
good = 1:length(vn); good(bad) = [];
fileplot = 'test08-ev.eps';
plot(vn,sb.Efit,'-r;BS fit;', vn(good),en(good),'ob;good data;' \
    , vn(bad),en(bad),'om;bad data;', 'markersize', 12);
xlabel('V (bohr^3)'); ylabel('E (hartree)'); grid('on');
print(fileplot, '-FHelvetica:24', '-depsc');
printf('See the E(V) curve in the file: %s\n', fileplot);
```

3.9 3.9 Test09: A dataset with jumps

Our last test will examine the capability of AsturFit for detecting and removing jumps in the E(V) data. The jumps are created artificially in our test so we can verify the degree of success of the detection routines. Notice that the position of the jump can only be determined to occur in between two points of the original data set grid.

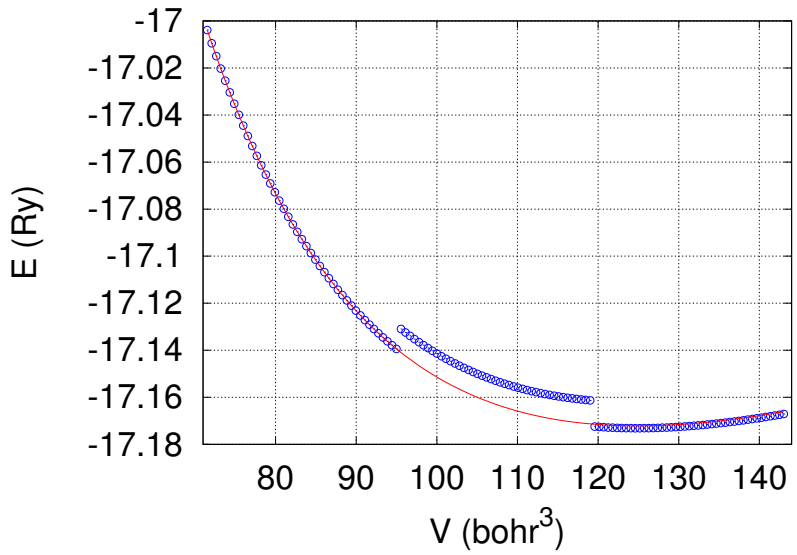


Figure 4: The best jump detection technique is based on interpolation.

```
# octave
% Test09: Detecting jumps in a dataset
addpath("../src/");

# read in a smooth data set
[v,e] = readEVdata('mgo-sety1.dat',1);
vref = median(v);

# now we add artificially some jumps
# (keep the original data for further comparison)
```

```
vmin = min(v); vmax = max(v); vrange = vmax-vmin;
vj = v; ej = e;
jumps.v = [vmin+vrange/3, vmin+2*vrange/3];
jumps.e = [0.010, -0.011];
for i = 1 : length(jumps.v)
   printf('Adding a jump of %.6f for V>%.6f\n', jumps.e(i), jumps.v(i));
   rk = find(vj > jumps.v(i));
   ej(rk) += jumps.e(i);
endfor

# let's do the automatic detection
[st,ecorr] = guessjumps3(vj,ej,:,1);

# compare the original and the corrected energies
SSerr = sum((e-ecorr).^2);
SStot = sum((e-mean(e)).^2);
R2 = 1 - SSerr / SStot;
printf('\n');
printf('Global error of the corrected data:\n');
printf('Determination coef. (R2, 1 for a exact correction): %.9e\n', R2);
printf('1-R2: %.2e\n', SSerr / SStot);
```

4 4 Alphabetic list of routines and routine documentation

4.1 4.1 allfits.m

```
allfits file1 file2 ...
```

Perform a complete set of fittings to a set of data files. The fitted models include several non linear EOS like Vinet, Holzapfel AP2, Murnaghan, Birch-Murnaghan and Poirier-Tarantola of several orders. It also includes several strain polynomials of different orders and averages of strain polynomials. The main purpose of this routine is coparing the performance of different EOS on the same dataset.

4.2 4.2 asturfit

```
asturfit file1 file2 ...
```

The standard fitting task. An average of eulerian strain polynomials is fitted to each dataset, and a representative collection of plots is produced. The output provides the equilibrium properties, including representative error bars. A file with the post important properties tabulated is also provided for further processing.

4.3 4.3 avgstrainfit.m

```
# octave
function [cavg,savg] = avgstrainfit (V, E, V0, nmax=16, MODE=1, \
                       strain='eulerian', LOG=0)
```

Fit to an average of strain polynomials.

Required input variables:

- V: cell volume.
- E: cell energy.
- V0: volume reference. Usually the volume at the minimum of energy.

Optional input variables (all have default values):

- {nmax = -1}: maximum degree of the polynomial.
- {MODE = 1}: polynomial weighting scheme.
- {strain = 'eulerian'}: strain form.
- **{LOG = 0}: print internal information about the fitting if LOG>0.** Some special modes: * LOG > 1 produces an statistical analysis of the spinodal
  point, if it is present in the average polynomial.
  - LOG > 2 produces an statistical analysis of p, B, B1p, etc

Minimum output:

- cavg: coefficients of the average fitting polynomial.

Additional (optional) output:

- savg: data structure containing:
  - **savg.eqmean[V, E, B, B1p, B2p, B3p]: mean value of the equilibrium** properties. If the average of polynomials has no suitable minimum within the input volume range the "eq" properties will correspond to the reference point.
  - savg.eqstd[V, E, B, B1p, B2p, B3p]: eq. props. standard deviation.
  - savg.R2: determination coefficient (1 for a exact fitting).
  - savg.Efit: vector contained the predicted energies.
  - **savg.pmean, savg.pstd: mean and standard error of the pressure at** all the input volumes.
  - **savg.Bmean, savg.Bstd: mean and standard error of the bulk modulus at** all the input volumes.
  - **savg.B1pmean, savg.B1pstd, savg.B2pmean, savg.B2pstd, savg.B3pmean,** savg.B3pstd, mean and standard error of the derivatives of the bulk modulus.

4.4 4.4 checknoise.m

Use as an independent script:

```
checknoise file1 file2 ...
```

It can also be used as an standard octave routine:

```
# octave
function status = checknoise (filein)
```

Check the noise of a suspicious datafile. An average strain polynomial will be used first to determine the error bars in the predicted equilibrium position (or in the reference point if there is no equilibrium within the valume range of the input dataset). If the error bars show some sign of noise the bootstrap method will be called to detect outliers and the jump detection routine will be invoked. A complete inform of the results will be send to the standard output, and the `checkfit` routine will return a final status indicative of the diagnostic.

Required input variables:

- filein: name of the input data file.

Output (if any):

- **status: code of the type of problem found in the dataset.**
  - 0 ---> data appears to be smooth.
  - 10 ---> the error bars indicate some noise.
  - 20 ---> very large error bars.
  - +1 ---> outliers have been found.
  - +2 ---> jumps have been found.
  - **These codes are added out to form the final status. A status of** 23, for instance, would mean very large error bars, with outliers and jumps detected.

4.5 4.5 errformatf.m

```
# octave
function s = errformatf(x, dx, nd=6)
```

Compact x(dx) printing of a quantity and an estimated error. This is best explained with an example. If the value is x=123.456789 and the estimated standard deviation is dx=0.0567 the routine will produce 123.457(57), that should be interpreted as 123.457(+/-)0.057.

Required input variables:

- x: quantity to print.
- dx: estimated error.

Optional input variables (all have default values):

- {nd=6}: maximum number of decimals to use.

Output:

- s: string containing the printed result.

Examples:

```
# octave
errformatf(-123.4567890123456,0.99734)          -> -123.5(10)
errformatf(-123.4567890123456,0.099734)         -> -123.46(10)
errformatf(-123.4567890123456,0.09734)          -> -123.457(97)
errformatf(-123.4567890123456,0.00009734)       -> -123.456789(97)
errformatf(-123.4567890123456,0.000009734)      -> -123.456789(10)
errformatf(-123.4567890123456,0.0000009734)     -> -123.456789(1)
errformatf(-123.4567890123456,0.00000009734)    -> -123.456789(0)
errformatf(-123.4567890123456,0.00000009734,8) -> -123.45678901(10)
```

4.6 4.6 figures.m

```
# octave
function ur = figures(x)
```

Determines the actual precision of the values contained in the vector x.

Required input variables:

- x: data vector.

Output:

- ur: resolution of x.

Examples:

```
# octave
figures(1.1234000000)  -> 1e-4
figures(1.1234000001)  -> 1e-10
figures(-1.1234000001) -> 1e-10
figures(-11234000.001) -> 1e-3
figures(-11234000.001) -> 1e-3
figures(1.23e7)        -> 1e5
figures(1.23e-7)       -> 1e-9
```

4.7 4.7 guessjumps3.m

```
# octave
function [steps,Ecorr] = guessjumps3(V,E, deltaE=0, LOG=0)
```

Detect jumps in the assumed continuous curve inherent to the (V,E) input dataset.

Required input variables:

- (V,E): vectors containing points of a E(V) curve.

Optional input variables (all have default values):

- **{deltaE = 0}: neglect the steps smaller than this threshold. The default** is accepting all jumps, no matter how small.
- {LOG = 0}: print internal information about the fitting if (LOG>0).

Minimum output:

- steps: cell array structure containing the detected jumps.
  - for k = 1 : nsteps
  - steps{k}.V ..... position of the jump.
  - steps{k}.E ..... estimated value of the step.

Additional (optional) output:

- Ecorr: vector containing the corrected values for E.

4.8 4.8 miefit.m

```
#octave
function res = miefit(p, V, T, Vref, Tref, mode1="none",
                      mode2="none", pin1=[], pin2=[], fit=1, LOG=1)
```

Miefit performs the fitting and evaluation of a Mie-Gruneisen p(V,T) EOS. The p(V,T) is made by adding up two different parts:

p(V,T) = p_T0(V) + p_th(V,T)

The p_T0(V) or "cold" isotherm EOS can be represented by many forms, like the Birch-Murnaghan, or Poirier-Tarantola, for instance. The "thermal" part is usually described in terms of a modified Debye form. This routine is designed to let the user combine many different forms for both EOS components.

Required input variables:

- (p,V,T): column vectors containing the points of the p(V,T) surface.
- Vref: reference volume used in many cold EOS forms.
- Tref: reference temperature used in the thermal EOS part. Notice that a subset of the (p,V,T) data (the points such that T==Tref) will be used to fit the cold EOS.

Required global variables:

- natoms: the number of atoms in the cell consistent with the volume v. Required by Tange thermal pressure expressions.
- nelectrons: the number of electrons, for Holzapfel's AP2 equation of state.

Optional input variables (all have default values):

- {mode1 = "none"}: select the cold EOS. The default value only provides an error message. Implemented modes:
  - "bm" or "eulerian": average of Birch-Murnaghan polynomials.
  - "bm3", "bm4", ...: Birch-Murnaghan family with a fixed order.
  - "pt" or "natural": average of Poirier-Tarantola polynomials.
  - "pt3", "pt4", ...: Poirier-Tarantola family with a fixed order.

  Notice that no input parameters are required to fit the linear forms (bm, pt).
- {pin1 = []}: column vector with the values (evaluation) or starting values (fitting) of the cold EOS parameters. In the case of fitting (fit = 1), if no values are entered, the routine will try to determine some reasonable guess. If evaluating (fit = 0), pin1 is mandatory.
- **{mode2 = "none"}: select the thermal EOS. The default value only provides an** error message. Implemented modes:
  + "tange1", "tange2", "tange3": hierarchical Debye models by Tange et

    al [J. Geophys. Res. 114 (2009) B03208]. The parameters are: - "tange1": [theta0, gamma0]. - "tange2": [theta0, gamma0, q0]. - "tange3": [theta0, gamma0, a0, b0].
  - "jackson4": [a0, a1, a2, a3].
- {pin2 = []}: column vector with the start value for the cold EOS parameters. If no values are entered, the routine will try to determine some starting point.
- {fit = 1}: in default mode the analytical EOS will be fitted to the (V,E) data. In the fit=0 mode, the pressure will be evaluated with the provided pin parameters (required in this mode) at the volumes contained in V and the temperatures in T.
- {LOG = 1}: print internal information about the fitting if (LOG>0).

Output:

- res.pout1 ... cold isotherm fitting parameters (equiv. to pin1)
- res.pout2 ... thermal pressure fitting parameters (equiv. to pin2)
- res.konv .... 1 if the fitting converged and 0 if not.
- res.pfit .... column vector with the values of p predicted by the fit.
- res.R2 ..... determination coefficient (1 for a exact fit).

4.9 4.9 nlf.m

```
# octave
function res = nlf(V, E, mode="none", pin=[], fit=1, LOG=1)
```

Non-linear fitting to a traditional E(V) EOS. It can also be used to evaluate the EOS for a fixed set of parameters.

Required input variables:

- (V,E): column vectors containing the points of the E(V) curve.

Optional input variables (all have default values):

- **{mode = "none"}: select the EOS. The default value will only provide an** error message. Implemented modes:
  - Birch-Murnaghan family: 'bm3', 'bm4', 'bm5';
  - Poirier-Tarantola family: 'pt3', 'pt4', 'pt5';
  - Murnaghan EOS: 'murn';
  - Holzapfel AP2: 'ap2';
- {pin = []}: column vector with the start value for the EOS parameters. If no values are entered, the routine will try to determine some starting point. If mode='ap2', pin(1) *must* contain the value for Z, even though this is not a true parameter and will not be used for the fitting.

- {fit = 1}: in default mode the analytical EOS will be fitted to the (V,E) data. In the fit=0 mode, the EOS will be evaluated with the provided pin parameters (required in this mode) at the volumes contained in V.
- {LOG = 1}: print internal information about the fitting if (LOG>0).

Output:

- res.pout .... column vector with the best estimate of the parameters.
- res.konv .... 1 if the fitting converged and 0 if not.
- res.Efit .... column vector with the values for E predicted by the fit.
- res.R2 ..... determination coefficient (1 for a exact fitting).

4.10 4.10 noisify.m

```
# octave
function [xn,yn] = noisify (x, y, points=-0.1, xfac=0.1, yfac=0.1, LOG=0)
```

Add noise to the (x,y) data.

Required input variables:

- x: vector with values of the independent variable.
- y: vector with values of the dependent variable.

Optional input variables (all have default values):

- {points = -0.1}: points to which some noise will be added. A positive integer represents the number of points to be modified. A negative number represents (after changing the sign) the fraction of input points that will be modified. The default, -0.1, means that ten percent of the input data will be changed.
- {xfac = 0.1}: noise factor for the x. An input value, x(k), will be converted into x(k) + x(k)*xfac*random, where the random number is uniformly distributed in [0,1). Use xfac=0 to prevent changes to the x values. **Caution** no test will be done internally to prevent absurd values for (xfac,yfac).
- {yfac = 0.1}: noise factor for the y.
- {LOG = 0}: print an internal information report.

Output:

- [xn,yn]: vectors containing the [x,y] data with added noise.

4.11 4.11 pvavgstrainfit.m

```
# octave
function [cavg,savg] = pvavgstrainfit (p, V, Vref, nmax=16, MODE=1, strain='eulerian',
```

Fit to an average strain polynomials using pressure-volume data.

Required input variables:

- p: pressure.
- V: cell volume.
- Vref: volume reference. Usually the volume at the minimum of energy.

Optional input variables (all have default values):

- {nmax = -1}: maximum degree of the polynomial.
- {MODE = 1}: polynomial weighting scheme.
- {strain = 'eulerian'}: strain form.
- {LOG = 0}: print internal information about the fitting if LOG>0.

Minimum output:

- cavg: coefficients of the average fitting polynomial.

Additional (optional) output:

- savg: data structure containing: + savg.eqmean[V, B, B1p, B2p, B3p]: mean value of the equilibrium

  properties. If the average of polynomials has no suitable minimum within the input volume range the "eq" properties will correspond to the reference point.

  - savg.eqstd[V, B, B1p, B2p, B3p]: eq. props. standard deviation.
  - savg.R2: determination coefficient (1 for a exact fitting).
  - savg.pmean, savg.pstd: mean and standard error of the pressure at all the input volumes.
  - savg.Bmean, savg.Bstd: mean and standard error of the bulk modulus at all the input volumes.
  - savg.B1pmean, savg.B1pstd, savg.B2pmean, savg.B2pstd, savg.B3pmean, savg.B3pstd, mean and standard error of the derivatives of the bulk modulus.

4.12 4.12 pvstraineval.m

```
# octave
function [s] = pvstraineval (c, Vref, V, strain='eulerian')
```

Evaluation of properties derived from a polynomial p(V) function based on a given strain form.

Required input variables:

- c: coefficients of the "energy versus f" polynomial. Usually this is determined in a previous call to the strainfit() routine.
- Vref: reference volume.
- V: vector containing the volumes at which the polynomial properties must be calculated.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- s: data structure containing: + s.E1v: first derivative of E versus V. + s.E2v: second derivative of E versus V. + s.E3v: third derivative of E versus V. + s.E4v: fourth derivative of E versus V. + s.p: vector with pressures at the V points. + s.B: vector with bulk moduli at the V points. + s.B1p: first derivative of the bulk modulus versus p. + s.B2p: second derivative of the bulk modulus versus p. + s.B3p: third derivative of the bulk modulus versus p.

4.13 4.13 pvstrainevalp.m

```
# octave
function p = pvstrainevalp (c, Vref, V, strain='eulerian')
```

Evaluation of the pressure from a polynomial p(V) function based on a given strain form.

Required input variables:

- **c: coefficients of the "energy versus f" polynomial. Usually this is** determined in a previous call to the strainfit() routine.
- Vref: reference volume.
- **V: vector containing the volumes at which the polynomial properties must** be calculated.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- p: pressures evaluated at the V points.

4.14 4.14 pvstrainfit.m

```
# octave
function [c,s] = pvstrainfit (p,V,Vref, ndeg=4, strain='eulerian', LOG=1)
```

Polynomial fitting of the pressure to one of several forms of crystal strain. Currently available strain forms are: eulerian, natural, lagrangian, infinitesimal, quotient, and x.

Required input variables:

- p: pressure.
- V: cell volume.
- Vref: volume reference used in thhe strain.

Optional input variables:

- {ndeg = 4}: degree of the polynomial.
- {strain = 'eulerian'}: strain form.
- {LOG = 1}: print internal information about the fitting.

Minimum output:

- c: coefficients of the fitting polynomial.

Additional (optional) output:

- s: data structure containing: + s.V0: recalculated volume at the minimum of energy. + s.B0[1:3]: bulk modulus and its pressure derivatives. + s.R2: determination coefficient (1 for a exact fitting). + s.S2: square sum of residuals. + s.Efit: vector containing the predicted energies. + s.p: vector containing the predicted pressure. + s.B: vector containing the predicted bulk modulus. + s.f: vector containing the strain values. + s.err: 0 if no error.

4.15 4.15 pvstrainmin.m

```
# octave
function [s] = pvstrainmin(c, Vref, Vrange, strain='eulerian')
```

Find the equilibrium point of the strain polynomial in the volume range indicated.

Required input variables:

- c: coefficients of the strain polynomial.
- Vref: reference volume used in the definition of the strain.
- **Vrange: vector containing the range of volumes in which the minimum of** energy will be determined.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- s: data structure containing: + s.Vmin: volume at the minimum of energy. + s.fmin: strain at the minimum. + s.pmin: pressure at the minimum (it should be 0). + s.err: error code. 0 means no problem.

4.16 4.16 readEVdata.m

```
# octave
function [V,E] = readEVdata (filename, LOG=1)
```

Read in the volume and energy of a crystal phase.

Required input variables:

- filename: name of the data file.

Optional input variables (all have default values):

- **{LOG = 1}: print information about the data read in if LOG>0.** LOG = 0 no output. LOG = 1 number of points read in, volume and energy range. LOG = 2 like 1 plus a complete list of the points read in.

4.17 4.17 readPVTdata.m

```
# octave
function [p, V, T] = readPVTdata (filename, LOG=1)
```

Read in the pressure, volume and temperature of a crystal phase.

Required input variables:

- filename: name of the data file.

Optional input variables (all have default values):

- {LOG = 1}: print information about the data read in if LOG>0. + LOG = 0 no output. + LOG = 1 number of points read in, volume and energy range. + LOG = 2 like 1 plus a complete list of the points read in.

4.18 4.18 skimPVT.m

```
# octave
function [p,v,t] = skimPVT(p,v,t,mode='square',a1=0,a2=0)
```

The p, v and t arrays must be 1D and have the same size. There are two discarding modes: 'square' (default), that discards points that are both low pressure (below a1 GPa) and high temperature (above a2 K). a1 and a2 are required parameters for the 'square' mode.

The 'debye' mode estimates the Debye temperature from the p(V) isotherm at temperature a1 (default min(t)) assuming poisson ratio 0.25. Then, for each pressure, points above the corresponding Debye temperature times a factor a2 are discarded. a2 defaults to 1.5.

Using the 'debye' mode requires two global variables: natoms (the number of atoms associated to the volume v) and mm (the molecular mass of the formula).

4.19 4.19 strainbootstrap.m

```
# octave
function [cb,sb] = strainbootstrap(V, E, V0, ndeg=12, nsample=100, \
strain='eulerian', LOG=0)
```

Bootstrap statistical method applied to the strain polynomials. A number of random samples are created from the initial {V,E} data. A collection of strain polynomials of different degree are fitted to each sample, and the equilibrium properties are averaged over the samples and polynomials. This is a form of Monte carlo statistics.

Required input variables:

- V: cell volume.
- E: cell energy.
- V0: volume reference. Usually the volume at the minimum of energy.

Optional input variables (all have default values):

- **{ndeg = 12}: maximum degree of the polynomial.** Polynomials up to this degree wil be used and averaged.
- {nsample = 100}: number of random samples to create and average.
- {strain = 'eulerian'}: strain form.
- {LOG = 0}: print internal information about the fitting.
  - 0 ..... no print
  - 1 ..... print basic information
  - 2 ..... time the run

Minimum output:

- cb: coefficients of the average fitting polynomial.

Additional (optional) output: REVISAR

- sb: data structure containing:
  - sb.mean(1:5): mean values of equilibrium {V,E,B,B',B",B"'}.
  - sb.std(1:5): std dev of equilibrium {V,E,B,B',B",B"'} values.
  - sb.outliers: list of the points declared as outliers.
  - sb.Efit: fitted values for the energy.
  - sb.R2: determination coefficient (1 for a exact fitting).

4.20 4.20 strain2volume.m

```
# octave
function V = strain2volume(f, V0, strain='eulerian')
```

Convert a vector of strain into a vector of volume values.

Required input variables:

- f: vector of strains.
- V0: reference volume used in the definition of the strain.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- V: vector containing the volumes.

4.21 4.21 strainevalE.m

```
# octave
function E = strainevalE (c, V0, V, strain='eulerian')
```

Evaluation of the energy derived from a polynomial function based on a given strain form. Use the more general straineval() for more properties.

Required input variables:

- **c: coefficients of the "energy versus f" polynomial. Usually this is** determined in a previous call to the strainfit() routine.
- V0: reference volume.
- **V: vector containing the volumes at which the polynomial properties must** be calculated.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- E: vector with energies at the V points.

4.22 4.22 straineval.m

```
# octave
function [s] = straineval (c, V0, V, strain='eulerian')
```

Evaluation of properties derived from a polynomial energy function based on a given strain form.

Required input variables:

- **c: coefficients of the "energy versus f" polynomial. Usually this is** determined in a previous call to the strainfit() routine.
- V0: reference volume.

- **V: vector containing the volumes at which the polynomial properties must** be calculated.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- s: data structure containing:
  - s.E: vector with energies at the V points.
  - s.E1v: first derivative of E versus V.
  - s.E2v: second derivative of E versus V.
  - s.E3v: third derivative of E versus V.
  - s.E4v: fourth derivative of E versus V.
  - s.p: vector with pressures at the V points.
  - s.B: vector with bulk moduli at the V points.
  - s.B1p: first derivative of the bulk modulus versus p.
  - s.B2p: second derivative of the bulk modulus versus p.
  - s.B3p: third derivative of the bulk modulus versus p.

4.23 4.23 strainfit.m

```
# octave
function [c,s] = strainfit (V,E,V0, ndeg=4, strain='eulerian', LOG=0)
```

Polynomial fitting of the energy to one of several forms of crystal strain. Currently available strain forms are: eulerian, natural, lagrangian, infinitesimal, quotient, and x.

Required input variables:

- V: cell volume.
- E: cell energy.
- V0: volume reference. Usually the volume at the minimum of energy.

Optional input variables (all have default values):

- {ndeg = 4}: degree of the polynomial.
- {strain = 'eulerian'}: strain form.
- {LOG = 0}: print internal information about the fitting.

Minimum output:

- c: coefficients of the fitting polynomial.

Additional (optional) output:

- s: data structure containing:
  - s.V0: recalculated volume at the minimum of energy.
  - s.E0: energy at the minimum.
  - s.B0[1:3]: bulk modulus and its pressure derivatives.
  - s.R2: determination coefficient (1 for a exact fitting).
  - s.S2: square sum of residuals.
  - s.Efit: vector contained the predicted energies.
  - s.p: vector contained the predicted pressure.
  - s.B: vector contained the predicted bulk modulus.
  - s.f: vector containing the strain values.
  - s.err: 0 if no error.

4.24 4.24 strainjumpfit.m

```
# octave
function [c, stepout, Ecorr] = strainjumpfit (V, E, V0, stepin\
   , ndeg=14 ,strain='eulerian', conv=1e-4, MAXSTEP=20, LOG=0)
```

Least squares fitting of a strain polynomial plus a set of step functions. Notice that a good guess of the step functions is very important, so calling this routine should be the second part after calling guessjump3() or a similar task.

Required input variables:

- (V,E): column vectors containing the volume and energy of the E(V) points.
- V0: reference volume for the strain.
- **stepin: position and value of the Heaviside step functions. The value** will be optimized in this routine, but the number and position will be maintained. The structure of this datum is:

  for k = 1 (**length(stepin)**) stepin{k}.V stepin{k}.E

  endfor

Optional input variables (all have default values):

- **{ndeg = 14}: degree for the strain polynomial. If negative, an average** of strain poynomials up to degree abs(ndeg) will be used.
- {strain = 'eulerian'}: strain form.

Output:

- c: coefficients of the fitting polynomial.
- **stepout: optimized step functions. The structure is identical to that of** stepin, so the output step o f a run can be used as the input step for a second round.

Additional (optional) output:

- Ecorr: energy values with the known jumps taken out.

4.25 4.25 strainmin.m

```
# octave
function [s] = strainmin(c, V0, Vrange, strain='eulerian')
```

Find the minimum of the strain polynomial in the volume range indicated.

Required input variables:

- c: coefficients of the strain polynomial.
- V0: reference volume used in the definition of the strain.
- **Vrange: vector containing the range of volumes in which the minimum of** energy will be determined.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- s: data structure containing:
  - s.Vmin: volume at the minimum of energy.
  - s.fmin: strain at the minimum.
  - s.Emin: energy at the minimum.
  - s.err: error code. 0 means no problem.

4.26 4.26 strainplot.m

```
# octave
function strainplot(V,E,V0,c,strain='eulerian',fplot='plot.eps',type='v')
```

Plot of the E(V) or E(f) data and the fitting polynomial. This is a basically obsolete routine. We have keeped it in the set following bad Felipe's advise (*everything serves for something*) rather than virtuous Mafalda answer (*but nothing serves for everything*). Feel free to transform this small and dumb routine into the splendid and wonderful masterpiece of artificial intelligence that it could have been.

Required input variables:

- V: cell volume.
- E: cell energy.
- V0: volume reference. Usually the volume at the minimum of energy.
- c: polynomial coefficients.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.
- {fplot = 'plot.eps'}: plot file name.
- {type = 'v'}: select between E(V) (defaulf) or E(f) (option 'f') plots.

4.27 4.27 strainspinodal.m

```
# octave
function [s] = strainspinodal(c, V0, Vrange, strain='eulerian')
```

Find the spinodal point of the strain polynomial in the volume range indicated. The spinodal point is characterized because the second derivative of the energy versus the volume becomes zero, and so does the bulk modulus. In addition, the pressure has its most negative value.

Required input variables:

- c: coefficients of the strain polynomial.
- V0: reference volume used in the definition of the strain.
- **Vrange: vector containing the range of volumes in which the search must** be performed.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- s: data structure containing:
  - s.Vsp: spinodal volume.
  - s.psp: pressure at the spinodal point.
  - s.Bsp: bulk modulus at the spinodal point (it should be zero).
  - s.err: 0 if no error happened, !=0 otherwise.

4.28 4.28 strainstepevalE.m

```
# octave
function E = strainstepevalE (c, V0, step, V, strain = 'eulerian')
```

Evaluate a strain polynomial augmented with Heaviside step functions.

Required input variables:

- c: coefficients of the strain polynomial.
- V0: reference volume for the strain.
- **step: position and value of the Heaviside step functions.** The structure of this datum is: for k = 1 : length(step)
  step{k}.V step{k}.E
  endfor
- V: volume of the points to be evaluated.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- E: energy values.

4.29 4.29 volume2strain.m

```
# octave
function f = volume2strain(V, V0, strain='eulerian')
```

Convert a vector of volumes into a vector of strain values.

Required input variables:

- V: vector containing the volumes to convert.
- V0: reference volume used in the definition of the strain.

Optional input variables (all have default values):

- {strain = 'eulerian'}: strain form.

Output:

- f: vector of strains.

5 5 Copyright notice

References

[fit1] A. Otero-de-la-Roza and V. Luaña, "Gibbs2: A new version of the quasi-harmonic model. I. Robust treatment of the static data", Comput. Phys. Commun. (submitted, 2010).

[fit2] A. Otero-de-la-Roza and V. Luaña, "Equations of State in Solids. Fitting theoretical data, possibly including noise and jumps", Comput. Theor. Chem. (submitted, 2010).

[fit3] M. A. Blanco, "Métodos cuánticos locales para la simulación de materiales iónicos. Fundamentos, algoritmos y aplicaciones". Tesis doctoral, Universidad de Oviedo, Julio de 1997. <http://www.unioviedo.es/qcg/mab/tesis.pdf>

[fit4] M. A. Blanco, E. Francisco, and V. Luaña "GIBBS: isothermal-isobaric thermodynamics of solids from energy curves using a quasi-harmonic Debye model", Comput. Phys. Commun. **158** (2004) 57--72.

[fit5] A. Otero-de-la-Roza, xxx, and V. Luaña, "Gibbs2: A new version of the quasi-harmonic model. II. The thermal treatment and the code", Comput. Phys. Commun. (submitted, 2011).