ImageShift

API Documentation

December 15, 2005

Contents

1 Package multireg

1.1 Modules

2 Module multireg.atrous

2.1 Functions

atrous2d(*arr*, *maxscale*=1, *kernel*='linear')

```
Compute a' trous wavelet transforms of 2-d numarray
object up to the 'scale' specified by the user.

This method supports 'linear', 'spline', and 'edge' kernels.

Syntax:
    waveplanes,wimage = atrous2d(arr,maxscale=1,kernel='linear')

Input:
    arr        - numarray array for input image
    maxscale   - number wavelet transformations to apply
    kernel     - kernel for wavelet transformation:
                     'linear'(default),'spline','leftedge','rightedge'
Output:
    waveplanes - stack of wavelet difference images
    wimage     - wavelet transformed image for scale 'scale'
```

atrous_diff(*wavelet_planes*, *scaled_arr*, *scale*=0)

Converts the final scaled atrous array and the stack of differences back into the original input image. If a scale is provided, it will return the wavelet transformed image corresponding to that scale. The higher the scale value, the lower the resolution, with a scale=0 corresponding to full resolution.

atrous_restore(*wavelet_planes*, *scaled_arr*, *scale*=0)

Converts the final scaled atrous array and the stack of differences back into the original input image. If a scale is provided, it will return the wavelet transformed image corresponding to that scale. The higher the scale value, the lower the resolution, with a scale=0 corresponding to full resolution.

atrousmed(*arr*, *scale*=1, *kernel*='linear', *median*=1)

```
Compute a' trous wavelet transforms of 2-d numarray
object up to the 'scale' specified by the user, applying
the median filter at each scale to remove artifacts.

This method supports 'linear', 'spline', and 'edge' kernels.

Syntax:
    waveplanes,wimage = atrous2d(arr,scale=1,kernel='linear')

Input:
    arr        - numarray array for input image
    scale      - scale for final wavelet transformation
    kernel     - kernel for wavelet transformation:
                    'linear'(default),'spline','leftedge','rightedge'
Output:
    waveplanes - stack of wavelet difference images
    wimage     - wavelet transformed image for scale 'scale'
```

multimed(*arr*, *maxscale*=2, *median*=2)

```
Compute multiscale median transforms of 2-d numarray
object up to the 'maxscale' specified by the user, applying
the median filter at each scale to remove artifacts.

Syntax:
    waveplanes,wimage = multimed(arr,maxscale=1)

Input:
    arr        - numarray array for input image
    maxscale   - number wavelet transformations to apply

Output:
    waveplanes - stack of wavelet difference images
    wimage     - wavelet transformed image for scale 'scale'
```

2.2 Variables

Name	Description
KERNELS	**Value:** {'edge': array([0. , 0.0625, 0.25 , 0.3-75 , 0.25 , 0.0625]), 'line... *(type=dict)*

3 Module multireg.chainMoments

3.1 Functions

compute_binary_moment_pq(...)

compute_moment_pq(xpos,ypos,p,q)

compute_moment_pq(*img*, *xcen*, *ycen*, *p*, *q*)

compute_moment_pq(img,xcen,ycen,p,q)

computeChainMatch(*a*, *b*, *order*, *kernel*)

computeChainMatch(a, b, order, kernel)

computeChainMatchCoeff(*a*, *b*)

computeChainMatchCoeff(a, b)

getChainCode(*image*, *npix*)

getChainCode(image, npix)

getFirstMoment(*image*, *xcen*, *ycen*)

getFirstMoment(image,xcen,ycen)

getMomentMatrix(*moments1*, *moments2*)

getMomentMatrix(moments1, moments2)

getMoments(*image*, *xcen*, *ycen*)

getMoments(image,xcen,ycen)

getSecondMoment(*image*, *xcen*, *ycen*)

getSecondMoment(image,xcen,ycen)

4 Module multireg.chipwavelets

4.1 Functions

compute_ccode_matrix(*img_ccode*, *ref_ccode*, *Lccode*=`0.10000000000000001`, *Tccode*=`0.80000000000000004`)

convert_1d(*index*, *shape*)

find_cog(*array*, *xpos*, *ypos*, *limit*)

perform_ImageMatch(*imgobs*, *refobs*, *scale*, *T_ccode*=`0.80000000000000004`, *T_moment*=`0.10000000000000001`)

Implement the Steps 2-6 of the ImageMatch algorithm on an input Observation object, img_obs, and a reference Observation, ref_obs.
T_ccode: threshold for chain-code matching T_moment: threshold for invariant moment matching

4.2 Variables

Name	Description
__version__	**Value:** `'0.3.0 (27-Sept-2005)'` *(type=str)*

4.3 Class Chip

```
This class keeps track of all the wavelet transformations
for a chip, and performs the object finding on those transformed
images.

Input:
    imagename   - full image name complete with extension
                    such as 'test_flt.fits[sci,1]'.
    imagearray  - numarray object containing the science data for image
    offset      - zero-point offset of chip relative to final output frame
    pyasn       - PyDrizzle object relating image to output frame
                    if None, perform no distortion correction in positions
    scale       - Number of wavelet transformations to apply to image
    form        - form of wavelet interpolation: spline or linear (default)
    photzpt     - photometric zero-point appropriate for this chip
    photflam    - photometric conversion factor to convert counts to flux
Methods:
    getPositions(scale=0)
        - returns list of undistorted positions for objects
            identified at specified wavelet transformation scale
    getRawPositions(scale=0)
        - returns list of original positions for objects identified
            at specified wavelet transformation scale
```

4.3.1 Methods

__init__(*self*, *exposure*, *keep_wavelets*=`False`, *scale*=`2`, *form*=`'linear'`, *median*=`1`)

addDelta(*self*, *delta*)

cleanWavelets(*self*)

computeRange(*self*)

getFluxes(*self*, *scale*=`0`, *units*=`'mag'`)

Returns fluxes for all objects. If units='mag', fluxes will be returned as magnitudes rather than electrons/counts/ADUs based on photometric keywords read in for this chip.

getMask(*self*)

Return expanded version of mask for chip in output frame.

outputPositions(*self*, *output*, *scale*=`0`, *clean*=`True`)

Writes extracted undistorted positions to output ASCII file.

setDelta(*self*, *delta*)

4.4 Class Observation

Known Subclasses: ReferenceObs

4.4.1 Methods

__init__(*self*, *name*)

addChip(*self*, *chip*)

computeFeatureMatrices(*self*, *refobs*, *scale*=`0`, *order*=`3`)

Return the feature matrices of the input Observation relative to the reference Observation; specifically, the invariant-moment distance matrix, the chain-code matching matrix, and a center-of-gravity matrix for each.

computeRange(*self*)

compute range of pixels spanned by entire observation in output frame.
This NEEDS to be run once all member chips have been added to this object.

createMask(*self*)

Creates mask of entire observation in output field.

getChainCodes(*self*, *scale*=0)

getCoords(*self*, *scale*=0)

Returns positions, weight, max, and type for all Objects detected at the specified scale.
USED ONLY BY .writeCoords() method of WaveShifts object. The type/list of values returned could still be modified.

getFluxes(*self*, *scale*=0)

getMoments(*self*, *scale*=0)

getPositions(*self*, *scale*=0)

getScales(*self*)

getSlices(*self*, *scale*=0)

setDelta(*self*, *delta*)

Set global delta for entire observation.

4.5 Class ReferenceObs

multireg.chipwavelets.Observation
└ **ReferenceObs**

Class used as reference observation for iterating the fit.

4.5.1 Methods

__init__(*self*, *wcs*)
Overrides: multireg.chipwavelets.Observation.__init__

addChip(*self*, *obs*)
Overrides: multireg.chipwavelets.Observation.addChip

checkOverlap(*self*, *obs*)

Determine whether this observation overlaps the current reference mask.

overlapMask(*self*, *obs*)

Updates mask of entire observation in output field, IF it is found to overlap observations already in reference mask.
It returns a flag denoting whether the observation overlapped or not and, therefore, whether the mask was updated or not.
It works on entire observations, rather than just chip-by-chip.

writeShifts(*self*, *filename*)

Inherited from Observation: computeFeatureMatrices, computeRange, createMask, getChainCodes, getCoords, getFluxes, getMoments, getPositions, getScales, getSlices, setDelta

5 Module multireg.edge_detect

5.1 Functions

canny_edge(*image*, *alpha*=0.10000000000000001, *thin*=False)

Canny edge detector algorithm implementation.

compute_edge_strength(*array*, *zero_cross*)

Compute the edge strength map for a LoG of an image by considering the slopes along both the x and y directions.

compute_LoG_image(*image*, *k_d*, *k_sigma*, *gsigma*, *gauss_sigma*)

compute_max_neighbor(*array*)

Compute max[N(i,j)*signum(-1*array).

dgauss(*x*, *sigma*)

Compute first order derivative of gaussian function.

find_edge_points(*array*)

```
Detects the edge points from a LoG image
using the criteria:
- for each zero-crossing pixel, it is an
edge point iff
LoG(i,j) <= max[N(i,i)*signum(-LoG(i,j))]

where N(i,j) is the 8-neighborhood values for i,j and
      Log(i,j) is the LoG value of the pixel i,j
Ref: Dai & Khorram (1999), IEEE Trans. GeoSci. and
     Remote Sensing, Vol 37, No 5, 2351-2362.
```

find_index(*array*, *values*)

Return the index into array that corresponds to each value.

find_LoG_zeros(*image*, *esigma*=3)

gauss(*x*, *sigma*)

Compute gaussian.

gauss_edge_kernel(*nx*, *ny*, *sigma_x*, *sigma_y*, *theta*)

Computes the 2D edge detector (first order derivative of 2D Gaussian function) with size n1*n2. Theta is the angle the detector rotated counter clockwise, while sigma_x and sigma_y are the stddev of the Gaussian functions.

gauss_kernel(*nx*, *ny*=`None`, *sigma_x*=`1.0`, *sigma_y*=`None`)

Computes the 2D Gaussian with size n1*n2. Sigma_x and sigma_y are the stddev of the Gaussian functions. The kernel will be normalized to a sum of 1.

interp2(*xk*, *yk*, *zk*, *xint*, *yint*)

Compute the 2D interpolated value.

interp_bilinear1d(*x*, *xi*)

Compute the bilinear interpolated value(s) of xi within x.

LoG_mask(*sigma*=`1.0`, *cutoff*=`3.8999999999999999`, *peak*=`1.0`)

Returns a Laplacian-of-Gaussian kernel [Ref. 1] with a mean of 0 and a peak specified by the user. If no width is provided by the user, the size of the kernel will be automatically determined based on C_log/sigma = 3.9 to include 99.83% of the energy. References: (1)Chen, et al, IEEE PAMI, PAMI-9, No. 4, July 1987, pg 584-590.

signum(*array*)

```
Return the signum values for the input array,
where the signum has the standard definition:
    signum(x = array(i,j)) = -1 for x < 0

                             0 for x == 0
                             1 for x > 0
```

6 Module multireg.expandArray

6.1 Functions

collapseMask(*image*)

collapseMask(image)

expandArrayF32(*image*, *factor*, *order*, *outnx*, *outny*)

expandArrayF32(image, factor, order, outnx, outny)

inflateMask(*mask*)

inflateMask(mask)

resample1DF32(*image*, *factor*, *order*, *outnx*)

resample1DF32(image, factor, order, outnx)

7 Module multireg.findobjects

7.1 Functions

build_LOGKernel(*size*, *sigma*)

Build an LoG kernel of arbitrary size

center1d(*region*)

Compute the center of gravity of a 1-d array. Based on 'mpc_getcenter' from IRAF imutil task 'center' in the cl.proto package.

DEGTORAD(*deg*)

discriminate_source(*array*, *labels*, *idnum*)

For source number 'idnum' from labels, determine whether it is a positive feature or the edge of a larger source. It will return 0 for an edge and 1 for a source.

find_center(*region*)

```
Compute the center of a star using MPC algorithm.
Based on 'mpc_cntr' from IRAF imutil task 'center'
in the cl.proto package.

Syntax:
    center = find_center(region)
Input:
    region - slice of array around target star
Output:
    center - array position of center as (y,x)
                relative to region origin
```

get_positions(*input*, *sigma*=2.0, *size*=None, *offset*=None, *region*=None, *thin*=False)

```
Process the input array to return the list of positions that correspond
to all objects.

This function relies on Numarray nd_image module for its operations.
Syntax:
    poslist,object,raw = get_positions(input,offset=(0.,0.),region=None)

Input:
    input   :  numarray array of the (wavelet scaled?) science data
                this array should correspond to slice specified in 'region'
    sigma   : sigma for gaussian for source/edge detection in image
    size    :  detection limit for objects, typically the size of the kernel
                used to filter input image. If None, no minimum size
                will be imposed. Default: None
    offset  : pixel offset of chip relative to final output frame
    region  : slice from full frame array to be used for object detection
    thin    : use homotropic thinning algorithm to extract line segments
                corresponding to detected edges, rather than border of
                identified edge region. This algorithm currently runs
                MUCH SLOWER than border extraction.

Output:
    poslist,objects,rawlist
    where
        poslist: list of positions for identified objects
        objects: list of slices corresponding to identified objects
        rawlist: list of positions for identified objects from array

    poslist and rawlist are of the form:
        [id, position, mag]
    where:
        id      :  target number from chip (integer)
        position:  list of position(s) given as x,y pairs (list of lists)
        counts  :  photometry for position(s)(float):
                    sum of masked region for sources, mean value for edges
    The positions reported in these lists correspond either to a single
    position for a positive source (star,small galaxy,...) or a list of
    x,y pixels which correspond to edge features in image.
```

LOG_function(*x*, *y*, *s*)

Return Laplacian-of-Gaussian for a given position with a width of sigma s.

RADTODEG(*rad*)

8 Module multireg.imageshift

This module implements a multi-scale transform based method for determining the offset between 2 images.

The original algorithm was implemented by ESO for their automated image registration in the ESI Imaging Survey image processing pipeline, as described by B. Vandame (2002, SPIE 4847, p. 123). The developers are grateful for the cooperation of the ESO developers in providing a copy of some of the EIS software for review during the development of this package.

Version 0.1 (Initial version) - WJH (3-Dec-2004) Version 0.2 - WJH (15-Dec-2005)

8.1 Variables

Name	Description
__version__	**Value:** `'0.2 (15 December 2005)'` *(type=str)*

8.2 Class ImageShift

```
The ImageShift class serves as the primary interface for
computing offsets between images using the multi-scale
wavelet transform.

=================================
DEVELOPMENT NOTE:
This code may eventually need to support fitting to a reference
image or coordinate list from a reference frame.
=================================

Syntax:
    ImageShift(input,output='shifts',reference=None, coeffs='header',
               scale=2,form='linear')

Inputs:
    input       - list of input filenames
    output      - name of output shiftfile
                    if nothing is specified, defaults to 'shifts'
    reference   - user-specified reference image
                    if None (default), first image from
                    input will be used
    coeffs      - parameter used to specify source (if any) of
                    distortion model to be applied to input images.
                    This corresponds directly to MultiDrizzle
                    'coeffs' and PyDrizzle 'idckey' parameters.
    scale       - number of wavelet transforms to apply: 4 (default)
    form        - form of interpolation kernel to use with wavelet
                    transforms: spline or linear (default)
Methods:
    .run(verbose=False,min_match=10):
        --> Computes shifts and writes them to output.
        radius  - object matching threshold in pixels
        min_match   - only compute shift if image has
                        at least 'min_match' objects
```

```
        verbose - print computed shifts interactively

    .writeShiftFile(shift_list = None, output=None):
        --> Writes out shifts to output file.
        shift_list  - list of computed shifts for each image in form of:
                (filename,((xshift,yshift),rotation,(scale,xscale,yscale))
                    If no shift_list is provided, it writes out values
                    in 'self.shifts'.
        output      - name of output shift file
                    If none is provided, defaults to name specified
                    when class was initialized.
```

8.2.1 Methods

__init__(*self*, *input*, *output*='shifts', *reference*=None, *coeffs*='header', *scale*=2, *form*='linear', *median*=1)

---

getPositionArrays(*self*, *objlist*)

Return detected positions as numarray arrays

---

run(*self*, *verbose*=False, *min_match*=10, *overwrite*=True)

```
Perform the matching between the position lists, then perform
a generalized linear fit to find the shifts.  These shifts
then get written out to a shiftfile.

The 'verbose' parameter turns on/off output of compute shifts to
STDOUT, with the default being turned off (quiet mode).

=================================
DEVELOPMENT NOTE:
    This needs to be expanded to support iteration over all
    resolution scales, and for all input images.
=================================

ImageMatch Algorithm:
Xiaolong Dai, Siamak Khorram, "A Feature-based Image Registration
Algorithm Using Improved Chain-code Representation Combined with
Invariant Moments", IEEE Trans. Geo. and Remote Sensing,
Vol 37, No. 5, 2351-2362, September 1999.
```

---

writeCoordFile(*self*, *imagename*, *scale*, *objlist*)

Write out object list to the coordinate file.

---

writeCoords(*self*, *scale*=0)

Write out coordinate files for all input observations.

writeShiftFile(*self*, *shift_list*=`None`, *output*=`None`)

Write out a shiftfile.

9 Module multireg.linearfit

9.1 Functions

apply_fit(*xy*, *coeffs*)

Apply the coefficients from a linear fit to an array of x,y positions.
The coeffs come from the 'coeffs' member of the 'fit_arrays()' output.

apply_old_coeffs(*xy*, *coeffs*)

Apply the offset/shift/rot values from a linear fit to an array of x,y positions.

DEGTORAD(*deg*)

fit_arrays(*xy*, *uv*)

```
Performs a generalized fit between matched lists of positions
given by the 2 column arrays xy and uv.

This function fits for translation, rotation, and scale changes
between 'xy' and 'uv', allowing for different scales and
orientations for X and Y axes.

Output:
   (Xo,Yo),Rot,(Scale,Sx,Sy)
   where
        Xo,Yo:  offset,
        Rot:    rotation,
        Scale:  average scale change, and
        Sx,Sy:  scale changes in X and Y separately.

Algorithm and nomenclature provided by: Colin Cox (11 Nov 2004)
```

RADTODEG(*rad*)

10 Module multireg.morph

10.1 Functions

closehole(*f*)

```
- Purpose
    Close holes of binary and gray-scale images.
- Synopsis
    y = closehole(f, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) or binary image.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity).
- Output
    y: (same datatype of f ).
- Description
    mmclohole creates the image y by closing the holes of the image
    f , according with the connectivity defined by the structuring
    element Bc .The images can be either binary or gray-scale.
- Examples
    #
    #   example 1
    #
    a = mmreadgray('pcb1bin.tif')
    b = closehole(a)
    mmshow(a)
    mmshow(b)
    #
    #   example 2
    #
    a = mmreadgray('boxdrill-B.tif')
    b = closehole(a)
    mmshow(a)
    mmshow(b)
```

closing_by_recon(*image*, *size*=3)

Implements closing by reconstruction of size n of image as defined on p. 211, P. Soille, 2002. closing = recon_by_erosion (dilation(image,size),image)

gauss(*sigma*, *dist*)

geodesic_dilation(*marker*, *mask*)

Perform geodesic dilation based on algorithm on p. 185 of Soille (2002).

geodesic_erosion(*marker*, *mask*)

Perform Geodesic erosion based on algorithm on p. 168, P. Soille, 2002.

geodesic_erosion1d(*marker*, *mask*)

Perform Geodesic erosion (in 1D) based on algorithm on p. 168, P. Soille, 2002.

grey_chm_transform(*array*, *fg*, *bg*, *origin*=0)

```
Perform Grey-scale constrained-hit-or-miss transform
   based on algorithm from pg. 145, Soille (2002).
   Example based on Fig 5.4 (pg 144, Soille 2002).

Input:
   array: gray-scale array (1-D or 2-D)
   fg   : foreground mask array (UInt8 only)
   bg   : background mask array (UInt8 only)

 Example:
   f = N.array([3, 3, 3, 0, 1, 6, 2, 1, 7, 5, 1, 0, 4, 6, 7, 3, 3, 3],
               type=Int8)
   B = N.array([0,0,1,1],N.UInt8)
   Bc = N.array([1,1,0,0,1,1],N.UInt8)
   chm = morph.grey_chm_transform(f,B,Bc)
   print chm
   array([0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0], type=Int8)
```

grey_uhm_transform(*array*, *fg*, *bg*)

```
Perform grey-scale unconstrained hit-or-miss transform
based on the algorithm from p 143 of Soille 2002.

Input:
   array: gray-scale array (1-D or 2-D)
   fg   : foreground mask array (UInt8 only)
   bg   : background mask array (UInt8 only)

 Example:
   f = N.array([3, 3, 3, 0, 1, 6, 2, 1, 7, 5, 1, 0, 4, 6, 7, 3, 3, 3],
               type=Int8)
   B = N.array([0,0,1,1],N.UInt8)
   Bc = N.array([1,1,0,0,1,1],N.UInt8)
   chm = morph.grey_uchm_transform(f,B,Bc)
   print chm
   array([0, 0, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 2, 0, 0, 0, 0], type=Int8)
```

limits(*f*)

```
- Purpose
    Get the possible minimum and maximum of an image.
- Synopsis
    y = limits(f)
- Input
    f: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Output
    y: Vector, the first element is the infimum, the second, the
       supremum.
- Description
    The possible minimum and the possible maximum of an image depend
    on its data type. These values are important to compute many
    morphological operators (for instance, negate of an image). The
    output is a vector, where the first element is the possible
    minimum and the second, the possible maximum.
- Examples
    #
    print limits(mmbinary([0, 1, 0]))
    print limits(uint8([0, 1, 2]))
```

makegauss(*shape*, *sigma*, *norm*)

numneg(*f*)

```
- Purpose
    Negate an image.
- Synopsis
    y = numneg(f)
- Input
    f: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Output
    y: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Description
    mmneg returns an image y that is the negation (i.e., inverse or
    involution) of the image f . In the binary case, y is the
    complement of f .
- Examples
    #
    #   example 1
    #
    f=uint8([255, 255, 0, 10, 20, 10, 0, 255, 255])
    print numneg(f)
    print numneg(uint8([0, 1]))
    print numneg(int32([0, 1]))
    #
    #   example 2
    #
    a = mmreadgray('gear.tif')
    b = nummneg(a)
    mmshow(a)
    mmshow(b)
    #
    #   example 3
    #
    c = mmreadgray('astablet.tif')
    d = nummneg(c)
    mmshow(c)
    mmshow(d)
```

opening_by_recon(*image*, *size*=3)

Implements opening by reconstruction of size n of image as defined on p. 210, P. Soille, 2002.
opening = recon_by_dilation (erosion(image,size),image)

point_maximum(*array*, *mask*)

Return a point-wise maximum array based on the mask image.

point_minimum(*array*, *mask*)

Return a point-wise minimum array based on the mask image.

recon_by_dilation(*marker*, *mask*, *max_iter*=10)

```
Iterate over geodesic dilation operations
until dilation(j+1) = dilation(j).
Perform Geodesic dilation based on
algorithm on p. 190-191, P. Soille, 2002.
This can be referenced as:
  R^delta_mask(marker)
```

recon_by_erosion(*marker*, *mask*, *max_iter*=10)

```
Iterate over geodesic erosion operations
until erosion(j+1) = erosion(j).
Perform Geodesic erosion based on
algorithm on p. 191-192, P. Soille, 2002.
This can be referenced as:
  R^epsilon_mask(marker)
```

transform_concave(*marker*, *interval*=10)

```
Perform h-concave transformation using algorithm
specified on pg. 203 of Soille (2002).
The algorithm can be stated as:

    HMIN_h(f) = recon_by_erosion(f+h,f)
    HCONCAVE_h(f) = HMIN_h(f) - f

where f = marker, h = interval.
```

transform_convex(*marker*, *interval*=10)

```
Perform h-convex transformation using algorithm
specified on pg. 203 of Soille (2002).
The algorithm can be stated as:

    HMAX_h(f) = recon_by_dilation(f+h,f)
    HCONVEX_h(f) = f - HMAX_h(f)

where f = marker, h = interval.
```

11 Module multireg.num_pymorph

```
Module morph -- SDC Morphology Toolbox
    Partially translated to numarray and nd_image

-------------------------------------------------------------------
The pymorph Morphology Toolbox for Python is a powerful collection of latest
state-of-the-art gray-scale morphological tools that can be applied to image
segmentation, non-linear filtering, pattern recognition and image analysis.
-------------------------------------------------------------------
**********************
*
* Translated to work with numarray
*
**********************
mmadd4dil()      -- Addition for dilation
mmbinary()       -- Convert a gray-scale image into a binary image
mmcwatershed()   -- Detection of watershed from markers.
mmdatatype()     -- Return the image datatype string
mmdil()          -- Dilate an image by a structuring element.
mmdist()         -- Distance transform.
mmero()          -- Erode an image by a structuring element.
mmgray()         -- Convert a binary image into a gray-scale image.
mmhomothin()     -- Interval for homotopic thinning.
mmimg2se()       -- Create a structuring element from a pair of images.
mminterot()      -- Rotate an interval
mmintersec()     -- Intersection of images.
mmis()           -- Verify if a relationship among images is true or false.
mmisbinary()     -- Check for binary image
mmisequal()      -- Verify if two images are equal
mmislesseq()     -- Verify if one image is less or equal another (is
                    beneath)
mmlimits()       -- Get the possible minimum and maximum of an image.
mmmat2set()      -- Converts image representation from matrix to set
mmmaxleveltype() -- Returns the maximum value associated to an image
                    datatype
mmneg()          -- Negate an image.
mmse2hmt()       -- Create a Hit-or-Miss Template (or interval) from a pair
                    of structuring elements.
mmsecross()      -- Diamond structuring element and elementary 3x3 cross.
mmseline()       -- Create a line structuring element.
mmsereflect()    -- Reflect a structuring element
mmserot()        -- Rotate a structuring element.
mmsesum()        -- N-1 iterative Minkowski additions
mmset2mat()      -- Converts image representation from set to matrix
mmsetrans()      -- Translate a structuring element
mmseunion()      -- Union of structuring elements
mmsubm()         -- Subtraction of two images, with saturation.
mmsupgen()       -- Sup-generating (hit-miss).
mmunion()        -- Union of images.

int32()          -- Convert an image to an int32 image.
uint8()          -- Convert an image to an uint8 image.
```

```
uint16()         -- Convert an image to a uint16 image.


**********************
*
* Original form based on Numeric
*
**********************
mmaddm()         -- Addition of two images, with saturation.
mmareaclose()    -- Area closing
mmareaopen()     -- Area opening
mmasf()          -- Alternating Sequential Filtering
mmasfrec()       -- Reconstructive Alternating Sequential Filtering
mmbench()        -- benchmarking main functions of the toolbox.
mmblob()         -- Blob measurements from a labeled image.
mmbshow()        -- Generate a graphical representation of overlaid binary
                    images.
mmcbisector()    -- N-Conditional bisector.
mmcdil()         -- Dilate an image conditionally.
mmcenter()       -- Center filter.
mmcero()         -- Erode an image conditionally.
mmclohole()      -- Close holes of binary and gray-scale images.
mmclose()        -- Morphological closing.
mmcloserec()     -- Closing by reconstruction.
mmcloserecth()   -- Close-by-Reconstruction Top-Hat.
mmcloseth()      -- Closing Top Hat.
mmcmp()          -- Compare two images pixelwisely.
mmconcat()       -- Concatenate two or more images along width, height or
                    depth.
mmcthick()       -- Image transformation by conditional thickening.
mmcthin()        -- Image transformation by conditional thinning.
mmdrawv()        -- Superpose points, rectangles and lines on an image.
mmdtshow()       -- Display a distance transform image with an iso-line
                    color table.
mmedgeoff()      -- Eliminate the objects that hit the image frame.
mmendpoints()    -- Interval to detect end-points.
mmflood()        -- Flooding filter- h,v,a-basin and dynamics (depth, area,
                    volume)
mmframe()        -- Create a frame image.
mmfreedom()      -- Control automatic data type conversion.
mmgdist()        -- Geodesic Distance Transform.
mmgdtshow()      -- Apply an iso-line color table to a gray-scale image.
mmglblshow()     -- Apply a random color table to a gray-scale image.
mmgradm()        -- Morphological gradient.
mmgrain()        -- Gray-scale statistics for each labeled region.
mmgshow()        -- Apply binary overlays as color layers on a binary or
                    gray-scale image
mmhistogram()    -- Find the histogram of the image f.
mmhmax()         -- Remove peaks with contrast less than h.
mmhmin()         -- Remove basins with contrast less than h.
mmhomothick()    -- Interval for homotopic thickening.
mminfcanon()     -- Intersection of inf-generating operators.
mminfgen()       -- Inf-generating.
```

```
mminfrec()       -- Inf-reconstruction.
mminpos()        -- Minima imposition.
mminstall()      -- Verify if the Morphology Toolbox is registered.
mmintershow()    -- Visualize an interval.
mmlabel()        -- Label a binary image.
mmlabelflat()    -- Label the flat zones of gray-scale images.
mmlastero()      -- Last erosion.
mmlblshow()      -- Display a labeled image assigning a random color for
                    each label.
mmopen()         -- Morphological opening.
mmopenrec()      -- Opening by reconstruction.
mmopenrecth()    -- Open-by-Reconstruction Top-Hat.
mmopenth()       -- Opening Top Hat.
mmopentransf()   -- Open transform.
mmpad4n()        -- mmpad4n
mmpatspec()      -- Pattern spectrum (also known as granulometric size
                    density).
mmplot()         -- Plot a function.
mmreadgray()     -- Read an image from a commercial file format and stores
                    it as a gray-scale image.
mmregister()     -- Register the SDC Morphology Toolbox.
mmregmax()       -- Regional Maximum.
mmregmin()       -- Regional Minimum (with generalized dynamics).
mmse2interval()  -- Create an interval from a pair of structuring elements.
mmsebox()        -- Create a box structuring element.
mmsedil()        -- Dilate one structuring element by another
mmsedisk()       -- Create a disk or a semi-sphere structuring element.
mmseshow()       -- Display a structuring element as an image.
mmshow()         -- Display binary or gray-scale images and optionally
                    overlay it with binary images.
mmskelm()        -- Morphological skeleton (Medial Axis Transform).
mmskelmrec()     -- Morphological skeleton reconstruction (Inverse Medial
                    Axis Transform).
mmskiz()         -- Skeleton of Influence Zone - also know as Generalized
                    Voronoi Diagram
mmstats()        -- Find global image statistics.
mmsupcanon()     -- Union of sup-generating or hit-miss operators.
mmsuprec()       -- Sup-reconstruction.
mmswatershed()   -- Detection of similarity-based watershed from markers.
mmsymdif()       -- Symmetric difference between two images
mmtext()         -- Create a binary image of a text.
mmthick()        -- Image transformation by thickening.
mmthin()         -- Image transformation by thinning.
mmthreshad()     -- Threshold (adaptive)
mmtoggle()       -- Image contrast enhancement or classification by the
                    toggle operator.
mmvdome()        -- Obsolete, use mmvmax.
mmversion()      -- SDC Morphology Toolbox version.
mmvmax()         -- Remove domes with volume less than v.
mmwatershed()    -- Watershed detection.

---
```

11.1 Functions

int32(*f*)

- Purpose Convert an image to an int32 image.
- Synopsis img = int32(f)
- Input f: Any image
- Output img: The converted image
- Description int32 clips the input image between the values -2147483647 and 2147483647 and converts it to the signed 32-bit datatype.

mmadd4dil(*f*, *c*)

- Purpose Addition for dilation
- Synopsis a = mmadd4dil(f, c)
- Input f: Gray-scale (uint8 or uint16) or binary image. Image c: Gray-scale (uint8 or uint16) or binary image. Constant
- Output a: Image f + c

mmaddm(*f1*, *f2*)

```
- Purpose
    Addition of two images, with saturation.
- Synopsis
    y = mmaddm(f1, f2)
- Input
    f1: Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image.
    f2: Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image. Or constant.
- Output
    y: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Description
    mmaddm creates the image y by pixelwise addition of images f1
    and f2 . When the addition of the values of two pixels saturates
    the image data type considered, the greatest value of this type
    is taken as the result of the addition.
- Examples
    #
    #   example 1
    #
    f = uint8([255,   255,    0,   10,    0,   255,   250])
    g = uint8([ 0,    40,   80,   140,  250,    10,    30])
    y1 = mmaddm(f,g)
    print y1
    y2 = mmaddm(g, 100)
    print y2
    #
    #   example 2
    #
    a = mmreadgray('keyb.tif')
    b = mmaddm(a,128)
    mmshow(a)
    mmshow(b)
```

mmareaclose(*f*, *a*, *Bc*=None)

```
- Purpose
    Area closing
- Synopsis
    y = mmareaclose(f, a, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) or binary image.
    a:  Double non negative integer.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity).
- Output
    y: Same type of f
- Description
    mmareaclose removes any pore (i.e., background connected
    component) with area less than a of a binary image f . The
    connectivity is given by the structuring element Bc . This
    operator is generalized to gray-scale images by applying the
    binary operator successively on slices of f taken from higher
    threshold levels to lower threshold levels.
- Examples
    #
    #   example 1
    #
    a=mmreadgray('form-1.tif')
    b=mmareaclose(a,400)
    mmshow(a)
    mmshow(b)
    #
    #   example 2
    #
    a=mmreadgray('n2538.tif')
    b=mmareaclose(a,400)
    mmshow(a)
    mmshow(b)
```

mmareaopen(*f*, *a*, *Bc*=None)

```
- Purpose
    Area opening
- Synopsis
    y = mmareaopen(f, a, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) or binary image.
    a:  Double non negative integer.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity).
- Output
    y: Same type of f
- Description
    mmareaopen removes any grain (i.e., connected component) with
    area less than a of a binary image f . The connectivity is given
    by the structuring element Bc . This operator is generalized to
    gray-scale images by applying the binary operator successively
    on slices of f taken from higher threshold levels to lower
    threshold levels.
- Examples
    #
    #   example 1
    #
    f=mmbinary(uint8([
     [1, 1, 0, 0, 0, 0, 1],
     [1, 0, 1, 1, 1, 0, 1],
     [0, 0, 0, 0, 1, 0, 0]]))
    y=mmareaopen(f,4,mmsecross())
    print y
    #
    #   example 2
    #
    f=uint8([
       [10,   11,   0,    0,   0,   0,  20],
       [10,    0,   5,    8,   9,   0,  15],
       [10,    0,   0,    0,  10,   0,   0]])
    y=mmareaopen(f,4,mmsecross())
    print y
    #
    #   example 3
    #
    a=mmreadgray('form-1.tif');
    b=mmareaopen(a,500);
    mmshow(a);
    mmshow(b);
    #
    #   example 4
    #
    a=mmreadgray('bloodcells.tif');
    b=mmareaopen(a,500);
    mmshow(a);
    mmshow(b);
```

mmasf(*f*, *SEQ*=`'OC'`, *b*=`None`, *n*=`1`)

- Purpose Alternating Sequential Filtering
- Synopsis y = mmasf(f, SEQ="OC", b=None, n=1)
- Input f: Gray-scale (uint8 or uint16) or binary image. SEQ: String Default: "OC". 'OC', 'CO', 'OCO', 'COC'. b: Structuring Element Default: None (3x3 elementary cross). n: Non-negative integer. Default: 1. (number of iterations).
- Output y: Image
- Description mmasf creates the image y by filtering the image f by n iterations of the close and open alternating sequential filter characterized by the structuring element b . The sequence of opening and closing is controlled by the parameter SEQ . 'OC' performs opening after closing, 'CO' performs closing after opening, 'OCO' performs opening after closing after opening, and 'COC' performs closing after opening after closing.
- Examples # # example 1 # f=mmreadgray('gear.tif') g=mmasf(f,'oc',mmsecross(),2) mmshow(f) mmshow(g) # # example 2 # f=mmreadgray('fabric.tif') g=mmasf(f,'oc',mmsecross(),3) mmshow(f) mmshow(g)

mmasfrec(*f*, *SEQ*=`'OC'`, *b*=`None`, *bc*=`None`, *n*=`1`)

- Purpose Reconstructive Alternating Sequential Filtering
- Synopsis y = mmasfrec(f, SEQ="OC", b=None, bc=None, n=1)
- Input f: Gray-scale (uint8 or uint16) or binary image. SEQ: String Default: "OC". Values: "OC" or "CO". b: Structuring Element Default: None (3x3 elementary cross). bc: Structuring Element Default: None (3x3 elementary cross). n: Non-negative integer. Default: 1. (number of iterations).
- Output y: Same type of f
- Description mmasf creates the image y by filtering the image f by n iterations of the close by reconstruction and open by reconstruction alternating sequential filter characterized by the structuring element b . The structure element bc is used in the reconstruction. The sequence of opening and closing is controlled by the parameter SEQ . 'OC' performs opening after closing, and 'CO' performs closing after opening.
- Examples # f=mmreadgray('fabric.tif') g=mmasfrec(f,'oc',mmsecross(),mmsecross(),3) mmshow(f) mmshow(g)

mmbench(*count*=10)

```
- Purpose
    benchmarking main functions of the toolbox.
- Synopsis
    mmbench(count=10)
- Input
    count: Double Default: 10. Number of repetitions of each
           function.

- Description
    mmbench measures the speed of many of SDC Morphology Toolbox
    functions in seconds. An illustrative example of the output of
    mmbench is, for a MS-Windows 2000 Pentium 4, 2.4GHz, 533MHz
    system bus, machine: SDC Morphology Toolbox V1.2 27Sep02
    Benchmark Made on Wed Jul 16 15:33:17 2003 computer= win32 image
    filename= csample.jpg width= 640 , height= 480 Function time
    (sec.) 1. Union bin 0.00939999818802 2. Union gray-scale
    0.00319999456406 3. Dilation bin, mmsecross 0.0110000014305 4.
    Dilation gray, mmsecross 0.00780000686646 5. Dilation gray,
    non-flat 3x3 SE 0.0125 6. Open bin, mmsecross 0.0125 7. Open
    gray-scale, mmsecross 0.0141000032425 8. Open gray, non-flat 3x3
    SE 0.0235000014305 9. Distance mmsecross 0.021899998188 10.
    Distance Euclidean 0.0264999985695 11. Geodesic distance
    mmsecross 0.028100001812 12. Geodesic distance Euclidean
    0.303100001812 13. Area open bin 0.0639999985695 14. Area open
    gray-scale 0.148500001431 15. Label mmsecross 0.071899998188 16.
    Regional maximum, mmsecross 0.043700003624 17. Open by rec,
    gray, mmsecross 0.0515000104904 18. ASF by rec, oc, mmsecross, 1
    0.090600001812 19. Gradient, gray-scale, mmsecross
    0.0171999931335 20. Thinning 0.0984999895096 21. Watershed
    0.268799996376 Average 0.0632523809161
```

mmbinary(*f*, *k1*=1)

```
- Purpose
    Convert a gray-scale image into a binary image
- Synopsis
    y = mmbinary(f, k1=1)
- Input
    f:  Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image.
    k1: Double Default: 1. Threshold value.
- Output
    y: Binary image.
- Description
    mmbinary converts a gray-scale image f into a binary image y by
    a threshold rule. A pixel in y has the value 1 if and only if
    the corresponding pixel in f has a value greater or equal k1 .
- Examples
    #
    #   example 1
    #
    a = array([0, 1, 2, 3, 4])
    b=mmbinary(a)
    print b
    #
    #   example 2
    #
    a=mmreadgray('mm3.tif')
    b=mmbinary(a,82)
    mmshow(a)
    mmshow(b)
```

mmblob(*fr*, *measurement*, *option*='image')

```
- Purpose
    Blob measurements from a labeled image.
- Synopsis
    y = mmblob(fr, measurement, option="image")
- Input
    fr:          Gray-scale (uint8 or uint16) image. Labeled image.
    measurement: String Default: "". Choice from 'AREA', 'CENTROID',
                 or 'BOUNDINGBOX'.
    option:      String Default: "image". Output format: 'image':
                 results as a binary image; 'data': results a column
                 vector of measurements (double).
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    Take measurements from the labeled image fr . The measurements
    are: area, centroid, or bounding rectangle. The parameter option
    controls the output format: 'IMAGE': the result is an image;
    'DATA': the result is a double column vector with the
    measurement for each blob. The region with label zero is not
    measured as it is normally the background. The measurement of
    region with label 1 appears at the first row of the output.
- Examples
    #
    #   example 1
    #
    fr=uint8([
       [1,1,1,0,0,0],
       [1,1,1,0,0,2],
       [1,1,1,0,2,2]])
    f_area=mmblob(fr,'area')
    print f_area
    f_cent=mmblob(fr,'centroid')
    print f_cent
    f_bb=mmblob(fr,'boundingbox')
    print f_bb
    d_area=mmblob(fr,'area','data')
    print d_area
    d_cent=mmblob(fr,'centroid','data')
    print d_cent
    d_bb=mmblob(fr,'boundingbox','data')
    print d_bb
    #
    #   example 2
    #
    f=mmreadgray('blob3.tif')
    fr=mmlabel(f)
    g=mmblob(fr,'area')
    mmshow(f)
    mmshow(g)
    #
    #   example 3
    #
    f=mmreadgray('blob3.tif')
    fr=mmlabel(f)
    centr=mmblob(fr,'centroid')
    mmshow(f,mmdil(centr))
    #
    #   example 4
    #
```

mmbshow(*f1*, *f2*=None, *f3*=None, *factor*=17)

```
- Purpose
    Generate a graphical representation of overlaid binary images.
- Synopsis
    y = mmbshow(f1, f2=None, f3=None, factor=17)
- Input
    f1:     Binary image.
    f2:     Binary image. Default: None.
    f3:     Binary image. Default: None.
    factor: Double Default: 17. Expansion factor for the output
            image. Use odd values above 9.
- Output
    y: Binary image. shaded image.
- Description
    Generate an expanded binary image as a graphical representation
    of up to three binary input images. The 1-pixels of the first
    image are represented by square contours, the pixels of the
    optional second image are represented by circles and for the
    third image they are represented by shaded squares. This
    function is useful to create graphical illustration of small
    images.
- Examples
    #
    f1=mmtext('b')
    f2=mmtext('w')
    g2=mmbshow(f1,f2)
    mmshow(g2)
    f3=mmtext('x')
    g3=mmbshow(f1,f2,f3)
    mmshow(g3);
```

mmcbisector(*f*, *B*, *n*)

- Purpose N-Conditional bisector.
- Synopsis y = mmcbisector(f, B, n)
- Input f: Binary image. B: Structuring Element n: positive integer (filtering rate)
- Output y: Binary image.
- Description mmcbisector creates the binary image y by performing a filtering of the morphological skeleton of the binary image f , relative to the structuring element B . The strength of this filtering is controlled by the parameter n. Particularly, if n=0 , y is the morphological skeleton of f itself.
- Examples # a=mmreadgray('blob2.tif') b=mmcbisector(a,mmsebox(),1) c=mmcbisector(a,mmsebox(),3) d=mmcbisector(a,mmsebox(),10) mmshow(a,b) mmshow(a,c) mmshow(a,d)

mmcdil(*f*, *g*, *b*=None, *n*=1)

```
- Purpose
    Dilate an image conditionally.
- Synopsis
    y = mmcdil(f, g, b=None, n=1)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    g: Gray-scale (uint8 or uint16) or binary image. Conditioning
       image.
    b: Structuring Element Default: None (3x3 elementary cross).
    n: Non-negative integer. Default: 1. (number of iterations).
- Output
    y: Image
- Description
    mmcdil creates the image y by dilating the image f by the
    structuring element b conditionally to the image g . This
    operator may be applied recursively n times.
- Examples
    #
    #   example 1
    #
    f = mmbinary(uint8([[1, 0, 0, 0, 0, 0, 0],                [0, 0, 0, 0, 0, 0, 0],                [0,
    g = mmbinary(uint8([[1, 1, 1, 0, 0, 1, 1],                [1, 0, 1, 1, 1, 0, 0],                [0,
    y1=mmcdil(f,g,mmsecross())
    y2=mmcdil(f,g,mmsecross(),3)
    #
    #   example 2
    #
    f = uint8([                [   0,    0,   0,   80,   0,   0],                [   0,    0,   0,    0
    g = uint8([                [   0,    1,   2,   50,   4,   5],                [   2,    3,   4,    0
    y1=mmcdil(f,g,mmsecross())
    y2=mmcdil(f,g,mmsecross(),3)
    #
    #   example 3
    #
    g=mmreadgray('pcb1bin.tif')
    f=mmframe(g,5,5)
    y5=mmcdil(f,g,mmsecross(),5)
    y25=mmcdil(f,g,mmsecross(),25)
    mmshow(g)
    mmshow(g,f)
    mmshow(g,y5)
    mmshow(g,y25)
    #
    #   example 4
    #
    g=mmneg(mmreadgray('n2538.tif'))
    f=mmintersec(g,0)
    f=mmdraw(f,'LINE:40,30,60,30:END')
    y1=mmcdil(f,g,mmsebox())
    y30=mmcdil(f,g,mmsebox(),30)
    mmshow(g)
    mmshow(f)
    mmshow(y1)
    mmshow(y30)
```

mmcenter(*f*, *b*=`None`)

- Purpose Center filter.
- Synopsis y = mmcenter(f, b=None)
- Input f: Gray-scale (uint8 or uint16) or binary image. b: Structuring Element Default: None (3x3 elementary cross).
- Output y: Image
- Description mmcenter creates the image y by computing recursively the morphological center, relative to the structuring element b , of the image f .
- Examples # f=mmreadgray('gear.tif') g=mmcenter(f,mmsedisk(2)) mmshow(f) mmshow(g)

mmcero(*f*, *g*, *b*=`None`, *n*=`1`)

```
- Purpose
    Erode an image conditionally.
- Synopsis
    y = mmcero(f, g, b=None, n=1)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    g: Gray-scale (uint8 or uint16) or binary image. Conditioning
       image.
    b: Structuring Element Default: None (3x3 elementary cross).
    n: Non-negative integer. Default: 1. (number of iterations).
- Output
    y: Image
- Description
    mmcero creates the image y by eroding the image f by the
    structuring element b conditionally to g . This operator may be
    applied recursively n times.
- Examples
    #
    f = mmneg(mmtext('hello'))
    mmshow(f)
    g = mmdil(f,mmseline(7,90))
    mmshow(g)
    a1=mmcero(g,f,mmsebox())
    mmshow(a1)
    a13=mmcero(a1,f,mmsebox(),13)
    mmshow(a13)
```

mmclohole(*f*, *Bc*=`None`)

```
- Purpose
    Close holes of binary and gray-scale images.
- Synopsis
    y = mmclohole(f, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) or binary image.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity).
- Output
    y: (same datatype of f ).
- Description
    mmclohole creates the image y by closing the holes of the image
    f , according with the connectivity defined by the structuring
    element Bc .The images can be either binary or gray-scale.
- Examples
    #
    #   example 1
    #
    a = mmreadgray('pcb1bin.tif')
    b = mmclohole(a)
    mmshow(a)
    mmshow(b)
    #
    #   example 2
    #
    a = mmreadgray('boxdrill-B.tif')
    b = mmclohole(a)
    mmshow(a)
    mmshow(b)
```

mmclose(*f*, *b*=`None`)

- Purpose Morphological closing.
- Synopsis y = mmclose(f, b=None)
- Input f: Gray-scale (uint8 or uint16) or binary image. b: Structuring Element Default: None (3x3 elementary cross).
- Output y: Image
- Description mmclose creates the image y by the morphological closing of the image f by the structuring element b . In the binary case, the closing by a structuring element B may be interpreted as the intersection of all the binary images that contain the image f and have a hole equal to a translation of B . In the gray-scale case, there is a similar interpretation taking the functions umbra.
- Examples # # example 1 # f=mmreadgray('blob.tif') bimg=mmreadgray('blob1.tif') b=mmimg2se(bimg) mmshow(f) mmshow(mmclose(f,b)) mmshow(mmclose(f,b),mmgradm(f)) # # example 2 # f = mmreadgray('form-1.tif') mmshow(f) y = mmclose(f,mmsedisk(4)) mmshow(y) # # example 3 # f = mmreadgray('n2538.tif') mmshow(f) y = mmclose(f,mmsedisk(3)) mmshow(y)

mmclose_old(*f*, *b*=`None`)

- Purpose Morphological closing.
- Synopsis y = mmclose(f, b=None)
- Input f: Gray-scale (uint8 or uint16) or binary image. b: Structuring Element Default: None (3x3 elementary cross).
- Output y: Image
- Description mmclose creates the image y by the morphological closing of the image f by the structuring element b . In the binary case, the closing by a structuring element B may be interpreted as the intersection of all the binary images that contain the image f and have a hole equal to a translation of B . In the gray-scale case, there is a similar interpretation taking the functions umbra.
- Examples # # example 1 # f=mmreadgray('blob.tif') bimg=mmreadgray('blob1.tif') b=mmimg2se(bimg) mmshow(f) mmshow(mmclose(f,b)) mmshow(mmclose(f,b),mmgradm(f)) # # example 2 # f = mmreadgray('form-1.tif') mmshow(f) y = mmclose(f,mmsedisk(4)) mmshow(y) # # example 3 # f = mmreadgray('n2538.tif') mmshow(f) y = mmclose(f,mmsedisk(3)) mmshow(y)

mmcloserec(*f*, *bdil*=`None`, *bc*=`None`)

```
- Purpose
    Closing by reconstruction.
- Synopsis
    y = mmcloserec(f, bdil=None, bc=None)
- Input
    f:    Gray-scale (uint8 or uint16) or binary image.
    bdil: Structuring Element Default: None (3x3 elementary cross).
          (dilation).
    bc:   Structuring Element Default: None (3x3 elementary cross).
          ( connectivity).
- Output
    y: Same type of f .
- Description
    mmcloserec creates the image y by a sup-reconstruction ( with
    the connectivity defined by the structuring element bc ) of the
    image f from its dilation by bdil .
- Examples
    #
    a = mmreadgray('danaus.tif')
    mmshow(a)
    b = mmcloserec(a,mmsebox(4))
    mmshow(b)
```

mmcloserecth(*f*, *bdil*=None, *bc*=None)

```
- Purpose
    Close-by-Reconstruction Top-Hat.
- Synopsis
    y = mmcloserecth(f, bdil=None, bc=None)
- Input
    f:    Gray-scale (uint8 or uint16) or binary image.
    bdil: Structuring Element Default: None (3x3 elementary cross).
          (dilation)
    bc:   Structuring Element Default: None (3x3 elementary cross).
          ( connectivity)
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmcloserecth creates the image y by subtracting the image f of
    its closing by reconstruction, defined by the structuring
    elements bc and bdil .
- Examples
    #
    a = mmreadgray('danaus.tif')
    mmshow(a)
    b = mmcloserecth(a,mmsebox(4))
    mmshow(b)
```

mmcloseth(*f*, *b*=None)

```
- Purpose
    Closing Top Hat.
- Synopsis
    y = mmcloseth(f, b=None)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    b: Structuring Element Default: None (3x3 elementary cross).
- Output
    y: Gray-scale (uint8 or uint16) or binary image. (Same type of f
       ).
- Description
    mmcloseth creates the image y by subtracting the image f of its
    morphological closing by the structuring element b .
- Examples
    #
    a = mmreadgray('danaus.tif')
    mmshow(a)
    b = mmcloseth(a,mmsebox(5))
    mmshow(b)
```

mmcmp(*f1*, *oper*, *f2*, *oper1*=None, *f3*=None)

```
- Purpose
    Compare two images pixelwisely.
- Synopsis
    y = mmcmp(f1, oper, f2, oper1=None, f3=None)
- Input
    f1:    Gray-scale (uint8 or uint16) or binary image.
    oper:  String Default: "". relationship from: '==', '~=',
           '<','<=', '>', '>='.
    f2:    Gray-scale (uint8 or uint16) or binary image.
    oper1: String Default: None. relationship from: '==', '~=',
           '<','<=', '>', '>='.
    f3:    Gray-scale (uint8 or uint16) or binary image. Default:
           None.
- Output
    y: Binary image.
- Description
    Apply the relation oper to each pixel of images f1 and f2 , the
    result is a binary image with the same size. Optionally, it is
    possible to make the comparison among three image. It is
    possible to use a constant value in place of any image, in this
    case the constant is treated as an image of the same size as the
    others with all pixels with the value of the constant.
- Examples
    #
    #   example 1
    #
    print mmcmp(uint8([1, 2, 3]),'<', uint8(2))
    print mmcmp(uint8([1, 2, 3]),'<', uint8([0, 2, 4]))
    print mmcmp(uint8([1, 2, 3]),'==', uint8([1, 1, 3]))
    #
    #   example 2
    #
    f=mmreadgray('keyb.tif')
    fbin=mmcmp(uint8(10), '<', f, '<', uint8(50))
    mmshow(f)
    mmshow(fbin)
```

mmconcat(*DIM*, *X1*, *X2*, *X3*=None, *X4*=None)

```
- Purpose
    Concatenate two or more images along width, height or depth.
- Synopsis
    Y = mmconcat(DIM, X1, X2, X3=None, X4=None)
- Input
    DIM: String Dimension to concatenate. 'WIDTH' or 'W', 'HEIGHT'
         or 'H', or ' DEPTH' or 'D'.
    X1:  Gray-scale (uint8 or uint16) or binary image.
    X2:  Gray-scale (uint8 or uint16) or binary image.
    X3:  Gray-scale (uint8 or uint16) or binary image. Default:
         None.
    X4:  Gray-scale (uint8 or uint16) or binary image. Default:
         None.
- Output
    Y: Gray-scale (uint8 or uint16) or binary image.
- Description
    Concatenate two or more images in any of the dimensions: width,
    height or depth. If the images do not match the dimension, a
    larger image is create with zero pixels to accommodate them. The
    images must have the same datatype.
- Examples
    #
    f1=mmreadgray('cameraman.tif')
    f2=mmreadgray('blob.tif')
    g=mmconcat('W',f1,mmgray(mmneg(f2)))
    mmshow(g);
```

mmcthick(*f*, *g*, *Iab*=None, *n*=-1, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Image transformation by conditional thickening.
- Synopsis
    y = mmcthick(f, g, Iab=None, n=-1, theta=45,
    DIRECTION="CLOCKWISE")
- Input
    f:         Binary image.
    g:         Binary image.
    Iab:       Interval Default: None (mmhomothick).
    n:         Non-negative integer. Default: -1. Number of
               iterations.
    theta:     Double Default: 45. Degrees of rotation: 45, 90, or
               180.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or
               'ANTI-CLOCKWISE'.
- Output
    y: Binary image.
- Description
    mmcthick creates the binary image y by performing a thickening
    of the binary image f conditioned to the binary image g . The
    number of iterations of the conditional thickening is n and in
    each iteration the thickening is characterized by rotations of
    theta of the interval Iab .
- Examples
    #
    #   example 1
    #
    f=mmreadgray('blob2.tif')
    mmshow(f)
    t=mmse2hmt(mmbinary([[0,0,0],[0,0,1],[1,1,1]]),
                              mmbinary([[0,0,0],[0,1,0],[0,0,0]]))
    print mmintershow(t)
    f1=mmthick(f,t,40); # The thickening makes the image border grow
    mmshow(f1)
    #
    #   example 2
    #
    f2=mmcthick(f,mmneg(mmframe(f)),t,40) # conditioning to inner pixels
    fn=mmcthick(f,mmneg(mmframe(f)),t) #pseudo convex hull
    mmshow(f2)
    mmshow(fn,f)
```

mmcthin(*f*, *g*, *Iab*=None, *n*=-1, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Image transformation by conditional thinning.
- Synopsis
    y = mmcthin(f, g, Iab=None, n=-1, theta=45,
    DIRECTION="CLOCKWISE")
- Input
    f:         Binary image.
    g:         Binary image.
    Iab:       Interval Default: None (mmhomothin).
    n:         Non-negative integer. Default: -1. Number of
               iterations.
    theta:     Double Default: 45. Degrees of rotations: 45, 90, or
               180.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
               ANTI-CLOCKWISE'.
- Output
    y: Binary image.
- Description
    mmcthin creates the binary image y by performing a thinning of
    the binary image f conditioned to the binary image g . The
    number of iterations of the conditional thinning is n and in
    each iteration the thinning is characterized by rotations of
    theta of the interval Iab .
```

mmcwatershed(*f*, *g*, *Bc*=None, *LINEREG*='LINES')

```
- Purpose
    Detection of watershed from markers.
- Synopsis
    Y = mmcwatershed(f, g, Bc=None, LINEREG="LINES")
- Input
    f:       Gray-scale (uint8 or uint16) image.
    g:       Gray-scale (uint8 or uint16) or binary image. marker
             image: binary or labeled.
    Bc:      Structuring Element Default: None (3x3 elementary
             cross). (watershed connectivity)
    LINEREG: String Default: "LINES". 'LINES' or ' REGIONS'.
- Output
    Y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmcwatershed creates the image y by detecting the domain of the
    catchment basins of f indicated by the marker image g ,
    according to the connectivity defined by Bc . According to the
    flag LINEREG y will be a labeled image of the catchment basins
    domain or just a binary image that presents the watershed lines.
    To know more about watershed and watershed from markers, see
    BeucMeye:93 . The implementation of this function is based on
    LotuFalc:00 . WARNING: There is a common mistake related to the
    marker image g . If this image contains only zeros and ones, but
    it is not a binary image, the result will be an image with all
    ones. If the marker image is binary, you have to set this
    explicitly using the logical function.
- Examples
    #
    #   example 1
    #
    a = uint8([                [10,   10,   10,   10,   10,   10,   10],                [10,    9,    6
    b = mmcmp(a,'==',uint8(6))
    print mmcwatershed(a,b)
    print mmcwatershed(a,b,mmsecross(),'REGIONS')
    #
    #   example 2
    #
    f=mmreadgray('astablet.tif')
    grad=mmgradm(f)
    mark=mmregmin(mmhmin(grad,17))
    w=mmcwatershed(grad,mark)
    mmshow(grad)
    mmshow(mark)
    mmshow(w)
```

mmdatatype(f)

```
- Purpose
    Return the image datatype string
- Synopsis
    type = mmdatatype(f)
- Input
    f: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image. Any image
- Output
    type: String String representation of image type: 'binary',
          'uint8', 'uint16' or 'int32'
- Description
    mmdatatype returns a string that identifies the pixel datatype
    of the image f .
```

mmdil(*f*, *b*=None)

```
- Purpose
    Dilate an image by a structuring element.
- Synopsis
    y = mmdil(f, b=None)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    b: Structuring Element Default: None (3x3 elementary cross).
- Output
    y: Image
- Description
    mmdil performs the dilation of image f by the structuring
    element b . Dilation is a neighbourhood operator that compares
    locally b with f , according to an intersection rule. Since
    Dilation is a fundamental operator to the construction of all
    other morphological operators, it is also called an elementary
    operator of Mathematical Morphology. When f is a gray-scale
    image, b may be a flat or non-flat structuring element.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [0, 0, 0, 0, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0]])
    b=mmbinary([1, 1, 0])
    mmdil(f,b)
    f=uint8([
       [ 0,   1,  2, 50,  4,  5],
       [ 2,   3,  4,  0,  0,  0],
       [12, 255, 14, 15, 16, 17]])
    mmdil(f,b)
    #
    #   example 2
    #
    f=mmbinary(mmreadgray('blob.tif'))
    bimg=mmbinary(mmreadgray('blob1.tif'))
    b=mmimg2se(bimg)
    mmshow(f)
    mmshow(mmdil(f,b))
    mmshow(mmdil(f,b),mmgradm(f))
    #
    #   example 3
    #
    f=mmreadgray('pcb_gray.tif')
    b=mmsedisk(5)
    mmshow(f)
    mmshow(mmdil(f,b))
```

mmdil_old(*f*, *b*=None)

```
- Purpose
    Dilate an image by a structuring element.
- Synopsis
    y = mmdil(f, b=None)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    b: Structuring Element Default: None (3x3 elementary cross).
- Output
    y: Image
- Description
    mmdil performs the dilation of image f by the structuring
    element b . Dilation is a neighbourhood operator that compares
    locally b with f , according to an intersection rule. Since
    Dilation is a fundamental operator to the construction of all
    other morphological operators, it is also called an elementary
    operator of Mathematical Morphology. When f is a gray-scale
    image, b may be a flat or non-flat structuring element.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [0, 0, 0, 0, 0, 0, 1],
       [0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0]])
    b=mmbinary([1, 1, 0])
    mmdil(f,b)
    f=uint8([
       [ 0,   1,  2, 50,  4,  5],
       [ 2,   3,  4,  0,  0,  0],
       [12, 255, 14, 15, 16, 17]])
    mmdil(f,b)
    #
    #   example 2
    #
    f=mmbinary(mmreadgray('blob.tif'))
    bimg=mmbinary(mmreadgray('blob1.tif'))
    b=mmimg2se(bimg)
    mmshow(f)
    mmshow(mmdil(f,b))
    mmshow(mmdil(f,b),mmgradm(f))
    #
    #   example 3
    #
    f=mmreadgray('pcb_gray.tif')
    b=mmsedisk(5)
    mmshow(f)
    mmshow(mmdil(f,b))
```

mmdist(*f*, *Bc*=None, *METRIC*=None)

```
- Purpose
    Distance transform.
- Synopsis
    y = mmdist(f, Bc=None, METRIC=None)
- Input
    f:      Binary image.
    Bc:     Structuring Element Default: None (3x3 elementary
            cross). (connectivity)
    METRIC: String Default: None. 'EUCLIDEAN', or 'EUC2' for squared
            Euclidean.
- Output
    y: distance image in uint16, or in int32 datatype with EUC2
       option.
- Description
    mmdist creates the distance image y of the binary image f . The
    value of y at the pixel x is the distance of x to the complement
    of f , that is, the distance of x to nearest point in the
    complement of f . The distances available are based on the
    Euclidean metrics and on metrics generated by a a regular graph,
    that is characterized by a connectivity rule defined by the
    structuring element Bc . The implementation of the Euclidean
    algorithm is based on LotuZamp:01 .
- Examples
    #
    #   example 1
    #
    a = mmframe(mmbinary(ones((5,9))),2,4)
    f4=mmdist(a)
    f8=mmdist(a,mmsebox())
    fe=mmdist(a,mmsebox(),'EUCLIDEAN')
    #
    #   example 2
    #
    f = mmreadgray('gear.tif')
    f = mmneg(mmgradm(f))
    d4=mmdist(f)
    d8=mmdist(f,mmsebox())
    de=mmdist(f,mmsebox(),'EUCLIDEAN')
    mmshow(f)
    mmshow(d4%8)
    mmshow(d8%8)
    mmshow(de%8)
```

mmdrawv(*f, data, value, GEOM*)

```
- Purpose
    Superpose points, rectangles and lines on an image.
- Synopsis
    y = mmdrawv(f, data, value, GEOM)
- Input
    f:     Gray-scale (uint8 or uint16) or binary image.
    data:  Gray-scale (uint8 or uint16) or binary image. vector of
           points. Each row gives information regarding a
           geometrical primitive. The interpretation of this data is
           dependent on the parameter GEOM. The line drawing
           algorithm is not invariant to image transposition.
    value: Gray-scale (uint8 or uint16) or binary image. pixel
           gray-scale value associated to each point in parameter
           data. It can be a column vector of values or a single
           value.
    GEOM:  String Default: "". geometrical figure. One of
           'point','line', 'rect', or 'frect' for drawing points,
           lines, rectangles or filled rectangles respectively.
- Output
    y: Gray-scale (uint8 or uint16) or binary image. y has the same
       type of f .
- Description
    mmdrawv creates the image y by a superposition of points,
    rectangles and lines of gray-level k1 on the image f . The
    parameters for each geometrical primitive are defined by each
    line in the 'data' parameter. For points , they are represented
    by a matrix where each row gives the point's row and column, in
    this order. For lines , they are drawn with the same convention
    used by points, with a straight line connecting them in the
    order given by the data matrix. For rectangles and filled
    rectangles , each row in the data matrix gives the two points of
    the diagonal of the rectangle, where the points use the same
    row, column convention.
- Examples
    #
    #   example 1
    #
    f=uint8(zeros((3,5)))
    pcoords=uint16([[0,2,4],
                    [0,0,2]])
    pvalue=uint16([1,2,3])
    print mmdrawv(f,pcoords,pvalue,'point')
    print mmdrawv(f,pcoords,pvalue,'line')
    rectcoords=uint16([[0],
                       [0],
                       [3],
                       [2]])
    print mmdrawv(f,rectcoords, uint16(5), 'rect')
    #
    #   example 2
    #
    f=mmreadgray('blob3.tif')
    pc=mmblob(mmlabel(f),'centroid','data')
    lines=mmdrawv(mmintersec(f,0),transpose(pc),uint8(1),'line')
    mmshow(f,lines)
```

mmdtshow(*f*, *n*=10)

```
- Purpose
    Display a distance transform image with an iso-line color table.
- Synopsis
    y = mmdtshow(f, n=10)
- Input
    f: Gray-scale (uint8 or uint16) image. Distance transform.
    n: Boolean Default: 10. Number of iso-contours.
- Output
    y: Gray-scale (uint8 or uint16) or binary image. Optionally
       return RGB uint8 image
- Description
    Displays the distance transform image f (uint8 or uint16) with a
    special gray-scale color table with n pseudo-color equaly
    spaced. The final appearance of this display is similar to an
    iso-contour image display. The infinity value, which is the
    maximum level allowed in the image, is displayed as black. The
    image is displayed in the MATLAB figure only if no output
    parameter is given.
- Examples
    #
    f=mmreadgray('blob.tif')
    fd=mmdist(f)
    mmshow(fd)
    mmdtshow(fd)
```

mmedgeoff(*f*, *Bc*=None)

```
- Purpose
    Eliminate the objects that hit the image frame.
- Synopsis
    y = mmedgeoff(f, Bc=None)
- Input
    f:  Binary image.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity)
- Output
    y: Binary image.
- Description
    mmedgeoff creates the binary image y by eliminating the objects
    (connected components) of the binary image f that hit the image
    frame, according to the connectivity defined by the structuring
    element Bc .
- Examples
    #
    a=mmreadgray('form-1.tif')
    b=mmedgeoff(a)
    mmshow(a)
    mmshow(b)
```

mmendpoints(*OPTION*='LOOP')

- Purpose Interval to detect end-points.
- Synopsis Iab = mmendpoints(OPTION="LOOP")
- Input OPTION: String Default: "LOOP". 'LOOP' or 'HOMOTOPIC'
- Output Iab: Interval
- Description mmendpoints creates an interval that is useful to detect end-points of curves (i.e., one pixel thick connected components) in binary images. It can be used to prune skeletons and to mark objects transforming them in a single pixel or closed loops if they have holes. There are two options available: LOOP, deletes all points but preserves loops if used in mmthin ; HOMOTOPIC, deletes all points but preserves the last single point or loops.
- Examples # # example 1 # print mmintershow(mmendpoints()) # # example 2 # print mmintershow(mmendpoints('HOMOTOPIC')) # # example 3 # f = mmreadgray('pcbholes.tif') mmshow(f) f1 = mmthin(f) mmshow(f1) f2 = mmthin(f1,mmendpoints(),20) mmshow(f2) # # example 4 # fn = mmthin(f1,mmendpoints('HOMOTOPIC')) mmshow(mmdil(fn))

mmero(*f*, *b*=None)

```
- Purpose
    Erode an image by a structuring element.
- Synopsis
    y = mmero(f, b=None)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    b: Structuring Element Default: None (3x3 elementary cross).
- Output
    y: Image
- Description
    mmero performs the erosion of the image f by the structuring
    element b . Erosion is a neighbourhood operator that compairs
    locally b with f , according to an inclusion rule. Since erosion
    is a fundamental operator to the construction of all other
    morphological operators, it is also called an elementary
    operator of Mathematical Morphology. When f is a gray-scale
    image , b may be a flat or non-flat structuring element.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [1, 1, 1, 0, 0, 1, 1],
       [1, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 0, 0]])
    b=mmbinary([1, 1, 0])
    mmero(f,b)
    f=uint8([
       [ 0,   1,  2, 50,  4,  5],
       [ 2,   3,  4,  0,  0,  0],
       [12, 255, 14, 15, 16, 17]])
    mmero(f,b)
    #
    #   example 2
    #
    f=mmbinary(mmreadgray('blob.tif'))
    bimg=mmbinary(mmreadgray('blob1.tif'))
    b=mmimg2se(bimg)
    g=mmero(f,b)
    mmshow(f)
    mmshow(g)
    mmshow(g,mmgradm(f))
    #
    #   example 3
    #
    f=mmreadgray('pcb_gray.tif')
    b=mmsedisk(3)
    mmshow(f)
    mmshow(mmero(f,b))
```

mmero_old(*f*, *b*=None)

```
- Purpose
    Erode an image by a structuring element.
- Synopsis
    y = mmero(f, b=None)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    b: Structuring Element Default: None (3x3 elementary cross).
- Output
    y: Image
- Description
    mmero performs the erosion of the image f by the structuring
    element b . Erosion is a neighbourhood operator that compairs
    locally b with f , according to an inclusion rule. Since erosion
    is a fundamental operator to the construction of all other
    morphological operators, it is also called an elementary
    operator of Mathematical Morphology. When f is a gray-scale
    image , b may be a flat or non-flat structuring element.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [1, 1, 1, 0, 0, 1, 1],
       [1, 0, 1, 1, 1, 0, 0],
       [0, 0, 0, 0, 1, 0, 0]])
    b=mmbinary([1, 1, 0])
    mmero(f,b)
    f=uint8([
       [ 0,   1,  2, 50,  4,  5],
       [ 2,   3,  4,  0,  0,  0],
       [12, 255, 14, 15, 16, 17]])
    mmero(f,b)
    #
    #   example 2
    #
    f=mmbinary(mmreadgray('blob.tif'))
    bimg=mmbinary(mmreadgray('blob1.tif'))
    b=mmimg2se(bimg)
    g=mmero(f,b)
    mmshow(f)
    mmshow(g)
    mmshow(g,mmgradm(f))
    #
    #   example 3
    #
    f=mmreadgray('pcb_gray.tif')
    b=mmsedisk(3)
    mmshow(f)
    mmshow(mmero(f,b))
```

mmflood(*fin*, *T*, *option*, *Bc*=None)

```
- Purpose
    Flooding filter- h,v,a-basin and dynamics (depth, area, volume)
- Synopsis
    y = mmflood(fin, T, option, Bc=None)
- Input
    fin:    Gray-scale (uint8 or uint16) image.
    T:      Criterion value. If T==-1, then the dynamics is
            determined, not the flooding at this criterion. This was
            selected just to use the same algoritm to compute two
            completely distinct functions.
    option: String Default: "". criterion: 'AREA', 'VOLUME', 'H'.
    Bc:     Structuring Element Default: None (3x3 elementary
            cross). Connectivity.
- Output
    y: Gray-scale (uint8 or uint16) image.
- Description
    This is a flooding algorithm. It is the basis to implement many
    topological functions. It is a connected filter that floods an
    image following some topological criteria: area, volume, depth.
    These filters are equivalent to area-close, volume-basin or
    h-basin, respectively. This code may be difficult to understand
    because of its many options. Basically, when t is negative, the
    generalized dynamics: area, volume, h is computed. When the
    flooding is computed, every time a new level in the flooding
    happens, a test is made to verify if the criterion has reached.
    This is used to set the value to that height. This value image
    will be used later for sup-reconstruction (flooding) at that
    particular level. This test happens in the raising of the water
    and in the merging of basins.
```

mmframe(*f*, *WT*=1, *HT*=1, *DT*=0, *k1*=None, *k2*=None)

```
- Purpose
    Create a frame image.
- Synopsis
    y = mmframe(f, WT=1, HT=1, DT=0, k1=None, k2=None)
- Input
    f:  Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image.
    WT: Double Default: 1. Positive integer ( width thickness).
    HT: Double Default: 1. Positive integer ( height thickness).
    DT: Double Default: 0. Positive integer ( depth thickness).
    k1: Non-negative integer. Default: None (Maximum pixel value
        allowed in f). Frame gray-level.
    k2: Non-negative integer. Default: None (Minimum pixel value
        allowed in f). Background gray level.
- Output
    y: image of same type as f .
- Description
    mmframe creates an image y , with the same dimensions (W,H,D)
    and same pixel type of the image f , such that the value of the
    pixels in the image frame is k1 and the value of the other
    pixels is k2 . The thickness of the image frame is DT.
```

mmfreedom(*L*=5)

```
- Purpose
    Control automatic data type conversion.
- Synopsis
    Y = mmfreedom(L=5)
- Input
    L: Double Default: 5. level of FREEDOM: 0, 1 or 2. If the input
       parameter is omitted, the current level is returned.
- Output
    Y: Double current FREEDOM level
- Description
    mmfreedom controls the automatic data type conversion. There are
    3 possible levels, called FREEDOM levels, for automatic
    conversion: 0 - image type conversion is not allowed; 1- image
    type conversion is allowed, but a warning is sent for each
    conversion; 2- image type conversion is allowed without warning.
    The FREEDOM levels are set or inquired by mmfreedom . If an
    image is not in the required datatype, than it should be
    converted to the maximum and nearest pymorph Morphology Toolbox
    datatype. For example, if an image is in int32 and a
    morphological gray-scale processing that accepts only binary,
    uint8 or uint16 images, is required, it will be converted to
    uint16. Another example, if a binary image should be added to a
    uint8 image, the binary image will be converted to uint8. In
    cases of operators that have as parameters an image and a
    constant, the type of the image should be kept as reference,
    while the type of the constant should be converted, if
    necessary.
- Examples
    #
    #   example 1
    #
    a=mmsubm([4., 2., 1.],uint8([3, 2, 0]))
    print a
    print mmdatatype(a)
    #
    #   example 2
    #
    a=mmsubm([4., 2., 1], mmbinary([3, 2, 0]))
    print a
    print mmdatatype(a)
    #
    #   example 3
    #
    a=mmsubm(uint8([4, 3, 2, 1]), 1)
    print a
    print mmdatatype(a)
```

mmgdist(*f*, *g*, *Bc*=None, *METRIC*=None)

```
- Purpose
    Geodesic Distance Transform.
- Synopsis
    y = mmgdist(f, g, Bc=None, METRIC=None)
- Input
    f:      Binary image.
    g:      Binary image. Marker image
    Bc:     Structuring Element Default: None (3x3 elementary
            cross). (metric for distance).
    METRIC: String Default: None. 'EUCLIDEAN' if specified.
- Output
    y: uint16 (distance image).
- Description
    mmgdist creates the geodesic distance image y of the binary
    image f relative to the binary image g . The value of y at the
    pixel x is the length of the smallest path between x and f . The
    distances available are based on the Euclidean metrics and on
    metrics generated by a neighbourhood graph, that is
    characterized by a connectivity rule defined by the structuring
    element Bc . The connectivity for defining the paths is
    consistent with the metrics adopted to measure their length. In
    the case of the Euclidean distance, the space is considered
    continuos and, in the other cases, the connectivity is the one
    defined by Bc .
- Examples
    #
    #   example 1
    #
    f=mmbinary([
     [1,1,1,1,1,1],
     [1,1,1,0,0,1],
     [1,0,1,0,0,1],
     [1,0,1,1,0,0],
     [0,0,1,1,1,1],
     [0,0,0,1,1,1]])
    g=mmbinary([
     [0,0,0,0,0,0],
     [1,1,0,0,0,0],
     [0,0,0,0,0,0],
     [0,0,0,0,0,0],
     [0,0,0,0,0,0],
     [0,0,0,0,0,1]])
    y=mmgdist(f,g,mmsecross())
    print y
    #
    #   example 2
    #
    f=mmreadgray('maze_bw.tif')
    g=mmintersec(f,0)
    g=mmdrawv(g,uint16([[2],[2],[6],[6]]),uint16(1),'frect')
    y=mmgdist(f,g,mmsebox(),'EUCLIDEAN')
    mmshow(f,g)
    mmdtshow(y,200)
```

mmgdtshow(*X*, *N*=10)

- Purpose Apply an iso-line color table to a gray-scale image.
- Synopsis Y = mmgdtshow(X, N=10)
- Input X: Gray-scale (uint8 or uint16) image. Distance transform image. N: Default: 10. Number of iso-contours.
- Output Y: Gray-scale (uint8 or uint16) or binary image.

mmglblshow(*X*, *border*=0.0)

- Purpose Apply a random color table to a gray-scale image.
- Synopsis Y = mmglblshow(X, border=0.0)
- Input X: Gray-scale (uint8 or uint16) image. Labeled image. border: Boolean Default: 0.0. Labeled image.
- Output Y: Gray-scale (uint8 or uint16) or binary image.

mmgradm(*f*, *Bdil*=None, *Bero*=None)

```
- Purpose
    Morphological gradient.
- Synopsis
    y = mmgradm(f, Bdil=None, Bero=None)
- Input
    f:    Gray-scale (uint8 or uint16) or binary image.
    Bdil: Structuring Element Default: None (3x3 elementary cross).
          for the dilation.
    Bero: Structuring Element Default: None (3x3 elementary cross).
          for the erosion.
- Output
    y: Gray-scale (uint8 or uint16) or binary image. (same type of f
       ).
- Description
    mmgradm creates the image y by the subtraction of the erosion of
    the image f by Bero of the dilation of f by Bdil .
- Examples
    #
    #   example 1
    #
    a = mmreadgray('small_bw.tif')
    b = mmgradm(a)
    mmshow(a)
    mmshow(b)
    #
    #   example 2
    #
    c=mmgradm(a,mmsecross(0),mmsecross())
    d=mmgradm(a,mmsecross(),mmsecross(0))
    mmshow(a,c)
    mmshow(a,d)
    #
    #   example 3
    #
    a = mmreadgray('bloodcells.tif')
    b = mmgradm(a)
    mmshow(a)
    mmshow(b)
```

mmgrain(*fr*, *f*, *measurement*, *option*='image')

```
- Purpose
    Gray-scale statistics for each labeled region.
- Synopsis
    y = mmgrain(fr, f, measurement, option="image")
- Input
    fr:          Gray-scale (uint8 or uint16) image. Labeled image,
                 to define the regions. Label 0 is the background
                 region.
    f:           Gray-scale (uint8 or uint16) image. To extract the
                 measuremens.
    measurement: String Default: "". Choose the measure to compute:
                 'max', 'min', 'median', 'mean', 'sum', 'std',
                 'std1'.
    option:      String Default: "image". Output format: 'image':
                 results as a gray-scale mosaic image (uint16);
                 'data': results a column vector of measurements
                 (double).
- Output
    y: Gray-scale (uint8 or uint16) image. Or a column vector
       (double) with gray-scale statistics per region.
- Description
    Computes gray-scale statistics of each grain in the image. The
    grains regions are specified by the labeled image fr and the
    gray-scale information is specified by the image f . The
    statistics to compute is specified by the parameter measurement
    , which has the same options as in function mmstats . The
    parameter option defines: ('image') if the output is an uint16
    image where each label value is changed to the measurement
    value, or ('data') a double column vector. In this case, the
    first element (index 1) is the measurement of region 1. The
    region with label zero is not measure as it is normally the
    background.
- Examples
    #
    #   example 1
    #
    f=uint8([range(6),range(6),range(6)])
    fr=mmlabelflat(f)
    mmgrain(fr,f,'sum','data')
    mmgrain(fr,f,'sum')
    #
    #   example 2
    #
    f=mmreadgray('astablet.tif')
    g=mmgradm(f)
    marker=mmregmin(mmclose(g))
    ws=mmcwatershed(g,marker,mmsebox(),'regions')
    g=mmgrain(ws,f,'mean')
    mmshow(f)
    mmshow(g)
```

mmgray(*f*, *TYPE*='uint8', *k1*=None)

```
- Purpose
    Convert a binary image into a gray-scale image.
- Synopsis
    y = mmgray(f, TYPE="uint8", k1=None)
- Input
    f:    Binary image.
    TYPE: String Default: "uint8". 'uint8', 'uint16', or 'int32'.
    k1:   Non-negative integer. Default: None (Maximum pixel level
          in pixel type).
- Output
    y: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Description
    mmgray converts a binary image into a gray-scale image of a
    specified data type. The value k1 is assigned to the 1 pixels of
    f , while the 0 pixels are assigned to the minimum value
    associated to the specified data type.
- Examples
    #
    b=mmbinary([0, 1, 0, 1])
    print b
    c=mmgray(b)
    print c
    d=mmgray(b,'uint8',100)
    print d
    e=mmgray(b,'uint16')
    print e
    f=mmgray(b,'int32',0)
    print f
```

mmgshow(*X*, *X1*=None, *X2*=None, *X3*=None, *X4*=None, *X5*=None, *X6*=None)

- Purpose Apply binary overlays as color layers on a binary or gray-scale image
- Synopsis Y = mmgshow(X, X1=None, X2=None, X3=None, X4=None, X5=None, X6=None)
- Input X: Gray-scale (uint8 or uint16) or binary image. X1: Binary image. Default: None. Red overlay. X2: Binary image. Default: None. Green overlay. X3: Binary image. Default: None. Blue overlay. X4: Binary image. Default: None. Magenta overlay. X5: Binary image. Default: None. Yellow overlay. X6: Binary image. Default: None. Cyan overlay.
- Output Y: Gray-scale (uint8 or uint16) or binary image.

mmhistogram(*f*, *option*='uint16')

```
- Purpose
    Find the histogram of the image f.
- Synopsis
    h = mmhistogram(f, option="uint16")
- Input
    f:      Gray-scale (uint8 or uint16) or binary image.
    option: String Default: "uint16". Values: "uint16" or "int32".
- Output
    h: Gray-scale (uint8 or uint16) image. Histogram in a uint16 or
       an int32 vector.
- Description
    Finds the histogram of the image f and returns the result in the
    vector h . For binary image the vector size is 2, for gray-scale
    uint8 and uint16 images, the vector size is the maximum pixel
    value plus one. h[0] gives the number of pixels with value 0.
- Examples
    #
    #   example 1
    #
    f=uint8([0, 1, 1, 2, 2, 2, 5, 3, 5])
    h=mmhistogram(f)
    print h
    #
    #   example 2
    #
    f=mmreadgray('lenina.tif')
    mmshow(f)
    h=mmhistogram(f)
    mmplot([[h]],[['style', 'impulses']])
```

mmhmax(*f*, *h*=1, *Bc*=None)

```
- Purpose
    Remove peaks with contrast less than h.
- Synopsis
    y = mmhmax(f, h=1, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) image.
    h:  Default: 1. Contrast parameter.
    Bc: Structuring Element Default: None (3x3 elementary cross).
        Structuring element ( connectivity).
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmhmax inf-reconstructs the gray-scale image f from the marker
    created by the subtraction of the positive integer value h from
    f , using connectivity Bc . This operator removes connected
    peaks with contrast less than h .
- Examples
    #
    #   example 1
    #
    a = uint8([
        [4,   3,   6,  1,  3,  5,  2],
        [2,   9,   6,  1,  6,  7,  3],
        [8,   9,   3,  2,  4,  9,  4],
        [3,   1,   2,  1,  2,  4,  2]])
    print mmhmax(a,2,mmsebox())
    #
    #   example 2
    #
    f = mmreadgray('r4x2_256.tif')
    mmshow(f)
    fb = mmhmax(f,50)
    mmshow(fb)
    mmshow(mmregmax(fb))
```

mmhmin(*f*, *h*=1, *Bc*=None)

```
- Purpose
    Remove basins with contrast less than h.
- Synopsis
    y = mmhmin(f, h=1, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) image.
    h:  Default: 1. Contrast parameter.
    Bc: Structuring Element Default: None (3x3 elementary cross).
        Structuring element (connectivity).
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmhmin sup-reconstructs the gray-scale image f from the marker
    created by the addition of the positive integer value h to f ,
    using the connectivity Bc . This operator removes connected
    basins with contrast less than h . This function is very userful
    for simplifying the basins of the image.
- Examples
    #
    #   example 1
    #
    a = uint8([
        [10,   3,   6,  18,  16,  15,  10],
        [10,   9,   6,  18,   6,   5,  10],
        [10,   9,   9,  15,   4,   9,  10],
        [10,  10,  10,  10,  10,  10,  10]])
    print mmhmin(a,1,mmsebox())
    #
    #   example 2
    #
    f = mmreadgray('r4x2_256.tif')
    mmshow(f)
    fb = mmhmin(f,70)
    mmshow(fb)
    mmshow(mmregmin(fb))
```

mmhomothick()

- Purpose Interval for homotopic thickening.
- Synopsis Iab = mmhomothick()
- Output Iab: Interval
- Description mmhomothick creates an interval that is useful for the homotopic (i.e., that conserves the relation between objects and holes) thickening of binary images.
- Examples # print mmintershow(mmhomothick())

mmhomothin()

- Purpose Interval for homotopic thinning.
- Synopsis Iab = mmhomothin()
- Output Iab: Interval
- Description mmhomothin creates an interval that is useful for the homotopic (i.e., that conserves the relation between objects and holes) thinning of binary images.

mmimg2se(*fd*, *FLAT*='FLAT', *f*=None)

```
- Purpose
    Create a structuring element from a pair of images.
- Synopsis
    B = mmimg2se(fd, FLAT="FLAT", f=None)
- Input
    fd:   Binary image. The image is in the matrix format where the
          origin (0,0) is at the matrix center.
    FLAT: String Default: "FLAT". 'FLAT' or 'NON-FLAT'.
    f:    Unsigned gray-scale (uint8 or uint16), signed (int32) or
          binary image. Default: None.
- Output
    B: Structuring Element
- Description
    mmimg2se creates a flat structuring element B from the binary
    image fd or creates a non-flat structuring element b from the
    binary image fd and the gray-scale image f . fd represents the
    domain of b and f represents the image of the points in fd .
- Examples
    #
    #   example 1
    #
    a = mmimg2se(mmbinary([
      [0,1,0],
      [1,1,1],
      [0,1,0]]))
    print mmseshow(a)
    #
    #   example 2
    #
    b = mmbinary([
      [0,1,1,1],
      [1,1,1,0]])
    b1 = mmimg2se(b)
    print mmseshow(b1)
    #
    #   example 3
    #
    c = mmbinary([
      [0,1,0],
      [1,1,1],
      [0,1,0]])
    d = int32([
      [0,0,0],
      [0,1,0],
      [0,0,0]])
    e = mmimg2se(c,'NON-FLAT',d)
    print mmseshow(e)
```

mminfcanon(*f*, *Iab*, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Intersection of inf-generating operators.
- Synopsis
    y = mminfcanon(f, Iab, theta=45, DIRECTION="CLOCKWISE")
- Input
    f:         Binary image.
    Iab:       Interval
    theta:     Double Default: 45. Degrees of rotation: 45, 90, or
               180.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
               ANTI-CLOCKWISE'
- Output
    y: Binary image.
- Description
    mminfcanon creates the image y by computing intersections of
    transformations of the image f by inf-generating (i.e., dual of
    the hit-or-miss) operators. These inf-generating operators are
    characterized by rotations (in the clockwise or anti-clockwise
    direction) of theta degrees of the interval Iab .
```

mminfgen(*f*, *Iab*)

- Purpose Inf-generating.
- Synopsis y = mminfgen(f, Iab)
- Input f: Binary image. Iab: Interval
- Output y: Binary image.
- Description mminfgen creates the image y by computing the transformation of the image f by the inf-generating operator (or dual of the hit-or-miss) characterized by the interval Iab .

mminfrec(*f*, *g*, *bc*=None)

```
- Purpose
    Inf-reconstruction.
- Synopsis
    y = mminfrec(f, g, bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) or binary image. Marker image.
    g:  Gray-scale (uint8 or uint16) or binary image. Conditioning
        image.
    bc: Structuring Element Default: None (3x3 elementary cross).
        Structuring element ( connectivity).
- Output
    y: Image
- Description
    mminfrec creates the image y by an infinite number of recursive
    iterations (iterations until stability) of the dilation of f by
    bc conditioned to g . We say the y is the inf-reconstruction of
    g from the marker f . For algorithms and applications, see
    Vinc:93b .
- Examples
    #
    #   example 1
    #
    g=mmreadgray('text_128.tif')
    f=mmero(g,mmseline(9,90))
    y=mminfrec(f,g,mmsebox())
    mmshow(g)
    mmshow(f)
    mmshow(y)
    #
    #   example 2
    #
    g=mmneg(mmreadgray('n2538.tif'))
    f=mmintersec(g,0)
    f=mmdraw(f,'LINE:40,30,60,30:END')
    y30=mmcdil(f,g,mmsebox(),30)
    y=mminfrec(f,g,mmsebox())
    mmshow(g)
    mmshow(f)
    mmshow(y30)
    mmshow(y)
```

mminpos(*f*, *g*, *bc*=None)

```
- Purpose
    Minima imposition.
- Synopsis
    y = mminpos(f, g, bc=None)
- Input
    f:  Binary image. Marker image.
    g:  Gray-scale (uint8 or uint16) image. input image.
    bc: Structuring Element Default: None (3x3 elementary cross).
        (connectivity).
- Output
    y: Gray-scale (uint8 or uint16) image.
- Description
    Minima imposition on g based on the marker f . mminpos creates
    an image y by filing the valleys of g that does not cover the
    connect components of f . A remarkable property of y is that its
    regional minima are exactly the connect components of g .
```

mminstall(*code*=None)

- Purpose Verify if the Morphology Toolbox is registered.
- Synopsis mminstall(code=None)
- Input code: String Default: None. Authorization code.
- Description mminstall verifies if the toolbox is registered or not. If not, it identifies the internal code that must be used to get the authorization code from the software manufacturer.

mminterot(*Iab*, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Rotate an interval
- Synopsis
    Irot = mminterot(Iab, theta=45, DIRECTION="CLOCKWISE")
- Input
    Iab:       Interval
    theta:     Double Default: 45. Degrees of rotation. Available
               values are multiple of 45 degrees.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
               ANTI-CLOCKWISE'.
- Output
    Irot: Interval
- Description
    mminterot rotates the interval Iab by an angle theta .
- Examples
    #
    b1 = mmendpoints()
    b2 = mminterot(b1)
    print mmintershow(b1)
    print mmintershow(b2)
```

mmintersec(*f1*, *f2*, *f3*=None, *f4*=None, *f5*=None)

```
- Purpose
    Intersection of images.
- Synopsis
    y = mmintersec(f1, f2, f3=None, f4=None, f5=None)
- Input
    f1: Gray-scale (uint8 or uint16) or binary image.
    f2: Gray-scale (uint8 or uint16) or binary image. Or constant.
    f3: Gray-scale (uint8 or uint16) or binary image. Default: None.
        Or constant.
    f4: Gray-scale (uint8 or uint16) or binary image. Default: None.
        Or constant.
    f5: Gray-scale (uint8 or uint16) or binary image. Default: None.
        Or constant.
- Output
    y: Image
- Description
    mmintersec creates the image y by taking the pixelwise minimum
    between the images f1, f2, f3, f4, and f5 . When f1, f2, f3, f4,
    and f5 are binary images, y is the intersection of them.
- Examples
    #
    #   example 1
    #
    f=uint8([255,  255,    0,   10,    0,   255,   250])
    g=uint8([ 0,    40,   80,   140,  250,    10,    30])
    print mmintersec(f, g)
    print mmintersec(f, 0)
    #
    #   example 2
    #
    a = mmreadgray('form-ok.tif')
    b = mmreadgray('form-1.tif')
    c = mmintersec(a,b)
    mmshow(a)
    mmshow(b)
    mmshow(c)
    #
    #   example 3
    #
    d = mmreadgray('tplayer1.tif')
    e = mmreadgray('tplayer2.tif')
    f = mmreadgray('tplayer3.tif')
    g = mmintersec(d,e,f)
    mmshow(d)
    mmshow(e)
    mmshow(f)
    mmshow(g)
```

mmintershow(*Iab*)

- Purpose Visualize an interval.
- Synopsis s = mmintershow(Iab)
- Input Iab: Interval
- Output s: String (representation of the interval).
- Description mmintershow creates a representation for an interval using 0, 1 and . (don't care).
- Examples # print mmintershow(mmhomothin())

mmis(*f1*, *oper*, *f2*=None, *oper1*=None, *f3*=None)

```
- Purpose
    Verify if a relationship among images is true or false.
- Synopsis
    y = mmis(f1, oper, f2=None, oper1=None, f3=None)
- Input
    f1:    Gray-scale (uint8 or uint16) or binary image.
    oper:  String relationship from: '==', '~=', '<','<=', '>',
           '>=', 'binary', 'gray'.
    f2:    Gray-scale (uint8 or uint16) or binary image. Default:
           None.
    oper1: String Default: None. relationship from: '==', '~=',
           '<','<=', '>', '>='.
    f3:    Gray-scale (uint8 or uint16) or binary image. Default:
           None.
- Output
    y: Bool value: 0 or 1
- Description
    Verify if the property or relatioship between images is true or
    false. The result is true if the relationship is true for all
    the pixels in the image, and false otherwise. (Obs: This
    function replaces mmis equal, mmis lesseq, mmis binary ).
- Examples
    #
    fbin=mmbinary([0, 1])
    f1=uint8([1, 2, 3])
    f2=uint8([2, 2, 3])
    f3=uint8([2, 3, 4])
    mmis(fbin,'binary')
    mmis(f1,'gray')
    mmis(f1,'==',f2)
    mmis(f1,'<',f3)
    mmis(f1,'<=',f2)
    mmis(f1,'<=',f2,'<=',f3)
```

mmisbinary(*f*)

- Purpose Check for binary image
- Synopsis bool = mmisbinary(f)
- Input f:
- Output bool: Boolean
- Description mmisbinary returns TRUE(1) if the datatype of the input image is binary. A binary image has just the values 0 and 1.
- Examples # a=uint8([0, 1, 0, 1]) print mmisbinary(a) b=(a) print mmisbinary(b)

mmisequal(*f1*, *f2*, *MSG*=`None`)

```
- Purpose
    Verify if two images are equal
- Synopsis
    bool = mmisequal(f1, f2)
- Input
    f1:  Unsigned gray-scale (uint8 or uint16), signed (int32) or
         binary image.
    f2:  Unsigned gray-scale (uint8 or uint16), signed (int32) or
         binary image.
- Output
    bool: Boolean
- Description
    mmisequal compares the images f1 and f2 and returns true (1), if
    f1(x)=f2(x) , for all pixel x , and false (0), otherwise.
- Examples
    #
    f1 = uint8(arrayrange(4))
    print f1
    f2 = uint8([9, 5, 3, 3])
    print f2
    f3 = f1
    mmisequal(f1,f2)
    mmisequal(f1,f3)
```

mmislesseq(*f1*, *f2*, *MSG*=`None`)

- Purpose Verify if one image is less or equal another (is beneath)
- Synopsis bool = mmislesseq(f1, f2)
- Input f1: Gray-scale (uint8 or uint16) or binary image. f2: Gray-scale (uint8 or uint16) or binary image.
- Output bool: Boolean
- Description mmislesseq compares the images f1 and f2 and returns true (1), if f1(x) <= f2(x) , for every pixel x, and false (0), otherwise.
- Examples # f1 = uint8([0, 1, 2, 3]) f2 = uint8([9, 5, 3, 3]) print mmislesseq(f1,f2) print mmislesseq(f2,f1) print mmislesseq(f1,f1)

mmlabel(*f*, *Bc*=None)

```
- Purpose
    Label a binary image.
- Synopsis
    y = mmlabel(f, Bc=None)
- Input
    f:  Binary image.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity).
- Output
    y: Image If number of labels is less than 65535, the data type
       is uint16, otherwise it is int32.
- Description
    mmlabel creates the image y by labeling the connect components
    of a binary image f , according to the connectivity defined by
    the structuring element Bc . The background pixels (with value
    0) are not labeled. The maximum label value in the output image
    gives the number of its connected components.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [0,1,0,1,1],
       [1,0,0,1,0]])
    g=mmlabel(f)
    print g
    #
    #   example 2
    #
    f = mmreadgray('blob3.tif')
    g=mmlabel(f)
    nblobs=mmstats(g,'max')
    print nblobs
    mmshow(f)
    mmlblshow(g)
```

mmlabelflat(*f*, *Bc*=None, *_lambda*=0)

```
- Purpose
    Label the flat zones of gray-scale images.
- Synopsis
    y = mmlabelflat(f, Bc=None, _lambda=0)
- Input
    f:       Gray-scale (uint8 or uint16) or binary image.
    Bc:      Structuring Element Default: None (3x3 elementary
             cross). ( connectivity).
    _lambda: Default: 0. Connectivity given by |f(q)-f(p)|<=_lambda.
- Output
    y: Image If number of labels is less than 65535, the data type
       is uint16, otherwise it is int32.
- Description
    mmlabelflat creates the image y by labeling the flat zones of f
    , according to the connectivity defined by the structuring
    element Bc . A flat zone is a connected region of the image
    domain in which all the pixels have the same gray-level
    (lambda=0 ). When lambda is different than zero, a quasi-flat
    zone is detected where two neighboring pixels belong to the same
    region if their difference gray-levels is smaller or equal
    lambda . The minimum label of the output image is 1 and the
    maximum is the number of flat-zones in the image.
- Examples
    #
    #   example 1
    #
    f=uint8([
       [5,5,8,3,0],
       [5,8,8,0,2]])
    g=mmlabelflat(f)
    print g
    g1=mmlabelflat(f,mmsecross(),2)
    print g1
    #
    #   example 2
    #
    f=mmreadgray('blob.tif')
    d=mmdist(f,mmsebox(),'euclidean')
    g= d /8
    mmshow(g)
    fz=mmlabelflat(g,mmsebox());
    mmlblshow(fz)
    print mmstats(fz,'max')
    #
    #   example 3
    #
    f=mmreadgray('pcb_gray.tif')
    g=mmlabelflat(f,mmsebox(),3)
    mmshow(f)
    mmlblshow(g)
```

mmlastero(*f*, *B*=None)

- Purpose Last erosion.
- Synopsis y = mmlastero(f, B=None)
- Input f: Binary image. B: Structuring Element Default: None (3x3 elementary cross).
- Output y: Binary image.
- Description mmlastero creates the image y by computing the last erosion by the structuring element B of the image f . The objects found in y are the objects of the erosion by nB that can not be reconstructed from the erosion by (n+1)B , where n is a generic non negative integer. The image y is a proper subset of the morphological skeleton by B of f .

mmlblshow(*f*, *option*='noborder')

```
- Purpose
    Display a labeled image assigning a random color for each label.
- Synopsis
    y = mmlblshow(f, option='noborder')
- Input
    f:      Gray-scale (uint8 or uint16) image. Labeled image.
    option: String Default: 'noborder'. BORDER or NOBORDER: includes
            or not a white border around each labeled region
- Output
    y: Gray-scale (uint8 or uint16) or binary image. Optionally
       return RGB uint8 image
- Description
    Displays the labeled image f (uint8 or uint16) with a pseudo
    color where each label appears with a random color. The image is
    displayed in the MATLAB figure only if no output parameter is
    given.
- Examples
    #
    f=mmreadgray('blob3.tif')
    f1=mmlabel(f,mmsebox())
    mmshow(f1)
    mmlblshow(f1)
    mmlblshow(f1,'border')
```

mmlimits(*f*)

```
- Purpose
    Get the possible minimum and maximum of an image.
- Synopsis
    y = mmlimits(f)
- Input
    f: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Output
    y: Vector, the first element is the infimum, the second, the
       supremum.
- Description
    The possible minimum and the possible maximum of an image depend
    on its data type. These values are important to compute many
    morphological operators (for instance, negate of an image). The
    output is a vector, where the first element is the possible
    minimum and the second, the possible maximum.
- Examples
    #
    print mmlimits(mmbinary([0, 1, 0]))
    print mmlimits(uint8([0, 1, 2]))
```

mmmat2set(*A*)

```
- Purpose
    Converts image representation from matrix to set
- Synopsis
    CV = mmmat2set(A)
- Input
    A: Image in matrix format, where the origin (0,0) is at the
       center of the matrix.
- Output
    CV: Image Tuple with array of pixel coordinates and array of
        corresponding pixel values
- Description
    Return tuple with array of pixel coordinates and array of
    corresponding pixel values. The input image is in the matrix
    format, like the structuring element, where the origin (0,0) is
    at the center of the matrix.
- Examples
    #
    #   example 1
    #
    f=uint8([[1,2,3],[4,5,6],[7,8,9]])
    i,v=mmmat2set(f)
    print i
    print v
    #
    #   example 2
    #
    f=uint8([[1,2,3,4],[5,6,7,8]])
    i,v=mmmat2set(f)
    print i
    print v
```

mmmaxleveltype(*TYPE*='uint8')

```
- Purpose
    Returns the maximum value associated to an image datatype
- Synopsis
    max = mmmaxleveltype(TYPE='uint8')
- Input
    TYPE: String Default: 'uint8'. One of the strings 'uint8',
          'uint16' or 'int32', specifying the image type
- Output
    max: the maximum level value of type TYPE
```

mmneg(*f*)

```
- Purpose
    Negate an image.
- Synopsis
    y = mmneg(f)
- Input
    f: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Output
    y: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Description
    mmneg returns an image y that is the negation (i.e., inverse or
    involution) of the image f . In the binary case, y is the
    complement of f .
- Examples
    #
    #   example 1
    #
    f=uint8([255, 255, 0, 10, 20, 10, 0, 255, 255])
    print mmneg(f)
    print mmneg(uint8([0, 1]))
    print mmneg(int32([0, 1]))
    #
    #   example 2
    #
    a = mmreadgray('gear.tif')
    b = mmneg(a)
    mmshow(a)
    mmshow(b)
    #
    #   example 3
    #
    c = mmreadgray('astablet.tif')
    d = mmneg(c)
    mmshow(c)
    mmshow(d)
```

mmopen(*f*, *b*=None)

- Purpose Morphological opening.
- Synopsis y = mmopen(f, b=None)
- Input f: Gray-scale (uint8 or uint16) or binary image. b: Structuring Element Default: None (3x3 elementary cross).
- Output y: Image
- Description mmopen creates the image y by the morphological opening of the image f by the structuring element b . In the binary case, the opening by the structuring element B may be interpreted as the union of translations of B included in f . In the gray-scale case, there is a similar interpretation taking the functions umbra.
- Examples # # example 1 # f=mmbinary(mmreadgray('blob.tif')) bimg=mmbinary(mmreadgray('blob1.tif')) b=mmimg2se(bimg) mmshow(f) mmshow(mmopen(f,b)) mmshow(mmopen(f,b),mmgradm(f)) # # example 2 # a=mmbinary(mmreadgray('pcb1bin.tif')) b=mmopen(a,mmsebox(2)) c=mmopen(a,mmsebox(4)) mmshow(a) mmshow(b) mmshow(c) # # example 3 # a=mmreadgray('astablet.tif') b=mmopen(a,mmsedisk(18)) mmshow(a) mmshow(b)

mmopenrec(*f*, *bero*=None, *bc*=None)

```
- Purpose
    Opening by reconstruction.
- Synopsis
    y = mmopenrec(f, bero=None, bc=None)
- Input
    f:    Gray-scale (uint8 or uint16) or binary image.
    bero: Structuring Element Default: None (3x3 elementary cross).
          (erosion).
    bc:   Structuring Element Default: None (3x3 elementary cross).
          (connectivity).
- Output
    y: Image (same type of f ).
- Description
    mmopenrec creates the image y by an inf-reconstruction of the
    image f from its erosion by bero , using the connectivity
    defined by Bc .
```

mmopenrecth(*f*, *bero*=None, *bc*=None)

```
- Purpose
    Open-by-Reconstruction Top-Hat.
- Synopsis
    y = mmopenrecth(f, bero=None, bc=None)
- Input
    f:    Gray-scale (uint8 or uint16) or binary image.
    bero: Structuring Element Default: None (3x3 elementary cross).
          (erosion)
    bc:   Structuring Element Default: None (3x3 elementary cross).
          ( connectivity)
- Output
    y: Gray-scale (uint8 or uint16) or binary image. (same type of f
       ).
- Description
    mmopenrecth creates the image y by subtracting the open by
    reconstruction of f , defined by the structuring elements bero e
    bc , of f itself.
```

mmopenth(*f*, *b*=None)

```
- Purpose
    Opening Top Hat.
- Synopsis
    y = mmopenth(f, b=None)
- Input
    f: Gray-scale (uint8 or uint16) or binary image.
    b: Structuring Element Default: None (3x3 elementary cross).
       structuring element
- Output
    y: Gray-scale (uint8 or uint16) or binary image. (same type of f
       ).
- Description
    mmopenth creates the image y by subtracting the morphological
    opening of f by the structuring element b of f itself.
- Examples
    #
    a = mmreadgray('keyb.tif')
    mmshow(a)
    b = mmopenth(a,mmsebox(3))
    mmshow(b)
```

mmopentransf(*f*, *type*=’OCTAGON’, *n*=65535, *Bc*=None, *Buser*=None)

```
- Purpose
    Open transform.
- Synopsis
    y = mmopentransf(f, type=’OCTAGON’, n=65535, Bc=None,
    Buser=None)
- Input
    f:     Binary image.
    type:  String Default: ’OCTAGON’. Disk family: ’OCTAGON’,
           ’CHESSBOARD’, ’CITY-BLOCK’, ’LINEAR-V’, ’LINEAR-H’,
           ’LINEAR-45R’, ’LINEAR-45L’, ’USER’.
    n:     Default: 65535. Maximum disk radii.
    Bc:    Structuring Element Default: None (3x3 elementary cross).
           Connectivity for the reconstructive opening. Used if
           ’-REC’ suffix is appended in the ’type’ string.
    Buser: Structuring Element Default: None (3x3 elementary cross).
           User disk, used if ’type’ is ’USER’.
- Output
    y: Gray-scale (uint8 or uint16) image.
- Description
    Compute the open transform of a binary image. The value of the
    pixels in the open transform gives the largest radii of the disk
    plus 1, where the open by it is not empty at that pixel. The
    disk sequence must satisfy the following: if r > s, rB is
    sB-open, i.e. rB open by sB is equal rB. Note that the Euclidean
    disk does not satisfy this property in the discrete grid. This
    function also computes the reconstructive open transform by
    adding the suffix ’-REC’ in the ’type’ parameter.
- Examples
    #
    #   example 1
    #
    f = mmbinary([
                  [0,0,0,0,0,0,0,0],
                  [0,0,1,1,1,1,0,0],
                  [0,0,1,1,1,1,1,0],
                  [0,1,0,1,1,1,0,0],
                  [1,1,0,0,0,0,0,0]])
    print mmopentransf( f, ’city-block’)
    print mmopentransf( f, ’linear-h’)
    print mmopentransf( f, ’linear-45r’)
    print mmopentransf( f, ’user’,10,mmsecross(),mmbinary([0,1,1]))
    print mmopentransf( f, ’city-block-rec’)
    #
    #   example 2
    #
    f=mmreadgray(’numbers.tif’)
    mmshow(f)
    g=mmopentransf(f,’OCTAGON’)
    mmshow(g)
    #
    #   example 3
    #
    b=mmsedisk(3,’2D’,’OCTAGON’)
    g1=mmopen(f,b)
    mmshow(g1)
    g2=mmcmp(g,’>’,3)
    print mmis(g1,’==’,g2)
```

mmpad4n(*f*, *Bc*, *value*, *scale*=1)

- Purpose mmpad4n
- Synopsis y = mmpad4n(f, Bc, value, scale=1)
- Input f: Image Bc: Structuring Element (connectivity). value: scale: Default: 1.
- Output y: The converted image

mmpatspec(*f*, *type*='OCTAGON', *n*=65535, *Bc*=None, *Buser*=None)

```
- Purpose
    Pattern spectrum (also known as granulometric size density).
- Synopsis
    h = mmpatspec(f, type='OCTAGON', n=65535, Bc=None, Buser=None)
- Input
    f:     Binary image.
    type:  String Default: 'OCTAGON'. Disk family: 'OCTAGON',
           'CHESSBOARD', 'CITY-BLOCK', 'LINEAR-V', 'LINEAR-H',
           'LINEAR-45R', 'LINEAR-45L', 'USER'.
    n:     Default: 65535. Maximum disk radii.
    Bc:    Structuring Element Default: None (3x3 elementary cross).
           Connectivity for the reconstructive granulometry. Used if
           '-REC' suffix is appended in the 'type' string.
    Buser: Structuring Element Default: None (3x3 elementary cross).
           User disk, used if 'type' is 'USER'.
- Output
    h: Gray-scale (uint8 or uint16) or binary image. a uint16
       vector.
- Description
    Compute the Pattern Spectrum of a binary image. See Mara:89b .
    The pattern spectrum is the histogram of the open transform, not
    taking the zero values.
```

mmplot(*plotitems*=[], *options*=[], *outfig*=-1, *filename*=None)

```
- Purpose
    Plot a function.
- Synopsis
    fig = mmplot(plotitems=[], options=[], outfig=-1, filename=None)
- Input
    plotitems: Default: []. List of plotitems.
    options:   Default: []. List of options.
    outfig:    Default: -1. Integer. Figure number. 0 creates a new
               figure.
    filename:  Default: None. String. Name of the PNG output file.
- Output
    fig: Figure number.

- Examples
    #
    import Numeric
    #
    x = Numeric.arange(0, 2*Numeric.pi, 0.1)
    mmplot([[x]])
    y1 = Numeric.sin(x)
    y2 = Numeric.cos(x)
    opts = [['title', 'Example Plot'],                    ['grid'],                    ['style', 'lines
    y1_plt = [x, y1, None,    'sin(X)']
    y2_plt = [x, y2, 'lines', 'cos(X)']
    #
    # plotting two graphs using one step
    fig1 = mmplot([y1_plt, y2_plt], opts, 0)
    #
    # plotting the same graphs using two steps
    fig2 = mmplot([y1_plt], opts, 0)
    fig2 = mmplot([y2_plt], opts, fig2)
    #
    # first function has been lost, lets recover it
    opts.append(['replot'])
    fig2 = mmplot([y1_plt], opts, fig2)
```

mmreadgray(*filename*)

- Purpose Read an image from a commercial file format and stores it as a gray-scale image.
- Synopsis y = mmreadgray(filename)
- Input filename: String Name of file to read.
- Output y: Gray-scale (uint8 or uint16) or binary image.
- Description mmreadgray reads the image in filename and stores it in y , an uint8 gray-scale image (without colormap). If the input file is a color RGB image, it is converted to gray-scale using the equation: y = 0.2989 R + 0.587 G + 0.114 B. This functions uses de PIL module.
- Examples # a=mmreadgray('cookies.tif') mmshow(a)

mmregister(*code*=None, *file_name*=None)

```
- Purpose
    Register the SDC Morphology Toolbox.
- Synopsis
    s = mmregister(code=None, file_name=None)
- Input
    code:      String Default: None. Authorization code.
    file_name: String Default: None. Filename of the license file to
               be created.
- Output
    s: String Message of the status of the license.
- Description
    mmregister licenses the copy of the SDC Morphology Toolbox by
    entering the license code and the toolbox license file. If
    mmregister is called without parameters, it returns the internal
    code that must be sent for registration.
```

mmregmax(*f*, *Bc*=None)

```
- Purpose
    Regional Maximum.
- Synopsis
    y = mmregmax(f, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) image.
    Bc: Structuring Element Default: None (3x3 elementary cross).
        (connectivity).
- Output
    y: Binary image.
- Description
    mmregmax creates a binary image y by computing the regional
    maxima of f , according to the connectivity defined by the
    structuring element Bc . A regional maximum is a flat zone not
    surrounded by flat zones of higher gray values.
```

mmregmin(*f*, *Bc*=None, *option*='binary')

```
- Purpose
    Regional Minimum (with generalized dynamics).
- Synopsis
    y = mmregmin(f, Bc=None, option="binary")
- Input
    f:      Gray-scale (uint8 or uint16) image.
    Bc:     Structuring Element Default: None (3x3 elementary
            cross). (connectivity).
    option: String Default: "binary". Choose one of: BINARY: output
            a binary image; VALUE: output a grayscale image with
            points at the regional minimum with the pixel values of
            the input image; DYNAMICS: output a grayscale image with
            points at the regional minimum with its dynamics;
            AREA-DYN: int32 image with the area-dynamics;
            VOLUME-DYN: int32 image with the volume-dynamics.
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmregmin creates a binary image f by computing the regional
    minima of f , according to the connectivity defined by the
    structuring element Bc . A regional minimum is a flat zone not
    surrounded by flat zones of lower gray values. A flat zone is a
    maximal connected component of a gray-scale image with same
    pixel values. There are three output options: binary image;
    valued image; and generalized dynamics. The dynamics of a
    regional minima is the minimum height a pixel has to climb in a
    walk to reach another regional minima with a higher dynamics.
    The area-dyn is the minimum area a catchment basin has to raise
    to reach another regional minima with higher area-dynamics. The
    volume-dyn is the minimum volume a catchment basin has to raise
    to reach another regional minima with a higher volume dynamics.
    The dynamics concept was first introduced in Grimaud:92 and it
    is the basic notion for the hierarchical or multiscale watershed
    transform.
- Examples
    #
    #   example 1
    #
    a = uint8([
        [10,  10,  10,  10,  10,  10,  10],
        [10,   9,   6,  18,   6,   5,  10],
        [10,   9,   6,  18,   6,   5,  10],
        [10,   9,   9,  15,   4,   9,  10],
        [10,   9,   9,  15,  12,  10,  10],
        [10,  10,  10,  10,  10,  10,  10]])
    print mmregmin(a)
    print mmregmin(a,mmsecross(),'value')
    print mmregmin(a,mmsecross(),'dynamics')
    #
    #   example 2
    #
    f1=mmreadgray('bloodcells.tif')
    m1=mmregmin(f1,mmsebox())
    mmshow(f1,m1)
    f2=mmhmin(f1,70)
    mmshow(f2)
    m2=mmregmin(f2,mmsebox())
    mmshow(f2,m2)
    #
```

mmse2hmt(*A*, *Bc*)

- Purpose Create a Hit-or-Miss Template (or interval) from a pair of structuring elements.
- Synopsis Iab = mmse2hmt(A, Bc)
- Input A: Structuring Element Left extremity. Bc: Structuring Element Complement of the right extremity.
- Output Iab: Interval
- Description mmse2hmt creates the Hit-or-Miss Template (HMT), also called interval [A,Bc] from the structuring elements A and Bc such that A is included in the complement of Bc . The only difference between this function and mmse2interval is that here the second structuring element is the complement of the one used in the other function. The advantage of this function over mmse2interval is that this one is more flexible in the use of the structuring elements as they are not required to have the same size.

mmse2interval(*a*, *b*)

- Purpose Create an interval from a pair of structuring elements.
- Synopsis Iab = mmse2interval(a, b)
- Input a: Structuring Element Left extremity. b: Structuring Element Right extremity.
- Output Iab: Interval
- Description mmse2interval creates the interval [a,b] from the structuring elements a and b such that a is less or equal b .

mmsebox(*r*=1)

- Purpose Create a box structuring element.
- Synopsis B = mmsebox(r=1)
- Input r: Non-negative integer. Default: 1. Radius.
- Output B: Structuring Element
- Description mmsebox creates the structuring element B formed by r successive Minkowski additions of the elementary square (i.e., the 3x3 square centered at the origin) with itself. If R=0, B is the unitary set that contains the origin. If R=1, B is the elementary square itself.
- Examples # b1 = mmsebox() mmseshow(b1) b2 = mmsebox(2) mmseshow(b2)

mmsecross(*r*=1)

- Purpose Diamond structuring element and elementary 3x3 cross.
- Synopsis B = mmsecross(r=1)
- Input r: Double Default: 1. (radius).
- Output B: Structuring Element
- Description mmsecross creates the structuring element B formed by r successive Minkowski additions of the elementary cross (i.e., the 3x3 cross centered at the origin) with itself. If r=0, B is the unitary set that contains the origin. If r=1 , B is the elementary cross itself.
- Examples # b1 = mmsecross() print mmseshow(b1) b2 = mmsecross(2) print mmseshow(b2)

mmsedil(*B1*, *B2*)

- Purpose Dilate one structuring element by another
- Synopsis Bo = mmsedil(B1, B2)
- Input B1: Structuring Element B2: Structuring Element
- Output Bo: Structuring Element
- Description mmsedil dilates an structuring element by another. The main difference between this dilation and mmdil is that the dilation between structuring elements are not bounded, returning another structuring element usually larger than anyone of them. This gives the composition of the two structuring elements by Minkowski addition.
- Examples # b1 = mmseline(5) mmseshow(b1) b2 = mmsedisk(2) mmseshow(b2) b3 = mmsedil(b1,b2) mmseshow(b3)

mmsedisk(*r*=3, *DIM*='2D', *METRIC*='EUCLIDEAN', *FLAT*='FLAT', *h*=0)

```
- Purpose
    Create a disk or a semi-sphere structuring element.
- Synopsis
    B = mmsedisk(r=3, DIM="2D", METRIC="EUCLIDEAN", FLAT="FLAT",
    h=0)
- Input
    r:      Non-negative integer. Default: 3. Disk radius.
    DIM:    String Default: "2D". '1D', '2D, or '3D'.
    METRIC: String Default: "EUCLIDEAN". 'EUCLIDEAN', ' CITY-BLOCK',
            'OCTAGON', or ' CHESSBOARD'.
    FLAT:   String Default: "FLAT". 'FLAT' or 'NON-FLAT'.
    h:      Double Default: 0. Elevation of the center of the
            semi-sphere.
- Output
    B: Structuring Element
- Description
    mmsedisk creates a flat structuring element B that is disk under
    the metric METRIC , centered at the origin and with radius r or
    a non-flat structuring element that is a semi-sphere under the
    metric METRIC, centered at (0, h) and with radius r . This
    structuring element can be created on the 1D, 2D or 3D space.
- Examples
    #
    #   example 1
    #
    a=mmseshow(mmsedisk(10,'2D','CITY-BLOCK'))
    b=mmseshow(mmsedisk(10,'2D','EUCLIDEAN'))
    c=mmseshow(mmsedisk(10,'2D','OCTAGON'))
    mmshow(a)
    mmshow(b)
    mmshow(c)
    #
    #   example 2
    #
    d=mmseshow(mmsedisk(10,'2D','CITY-BLOCK','NON-FLAT'))
    e=mmseshow(mmsedisk(10,'2D','EUCLIDEAN','NON-FLAT'))
    f=mmseshow(mmsedisk(10,'2D','OCTAGON','NON-FLAT'))
    mmshow(d)
    mmshow(e)
    mmshow(f)
    #
    #   example 3
    #
    g=mmsedisk(3,'2D','EUCLIDEAN','NON-FLAT')
    mmseshow(g)
    h=mmsedisk(3,'2D','EUCLIDEAN','NON-FLAT',5)
    mmseshow(h)
```

mmseline(*l*=3, *theta*=0)

- Purpose Create a line structuring element.
- Synopsis B = mmseline(l=3, theta=0)
- Input l: Non-negative integer. Default: 3. theta: Double Default: 0. (degrees, clockwise)
- Output B: Structuring Element
- Description mmseline creates a structuring element B that is a line segment that has an extremity at the origin, length l and angle theta (0 degrees is east direction, clockwise). If l=0 , it generates the origin.
- Examples # mmseshow(mmseline()) b1 = mmseline(4,45) mmseshow(b1) b2 = mmseline(4,-180) mmseshow(b2) a=mmtext('Line') b=mmdil(a,b1) mmshow(a) mmshow(b)

mmsereflect(*Bi*)

- Purpose Reflect a structuring element
- Synopsis Bo = mmsereflect(Bi)
- Input Bi: Structuring Element
- Output Bo: Structuring Element
- Description mmsereflect reflects a structuring element by rotating it 180 degrees.
- Examples # b1 = mmseline(5,30) print mmseshow(b1) b2 = mmsereflect(b1) print mmseshow(b2)

mmserot(*B*, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Rotate a structuring element.
- Synopsis
    BROT = mmserot(B, theta=45, DIRECTION="CLOCKWISE")
- Input
    B:         Structuring Element
    theta:     Double Default: 45. Degrees of rotation. Available
               values are multiple of 45 degrees.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
               ANTI-CLOCKWISE'.
- Output
    BROT: Structuring Element
- Description
    mmserot rotates a structuring element B of an angle theta .
- Examples
    #
    b = mmimg2se(mmbinary([[0, 0, 0], [0, 1, 1], [0, 0, 0]]));
    mmseshow(b)
    mmseshow(mmserot(b))
    mmseshow(mmserot(b,45,'ANTI-CLOCKWISE'))
```

mmseshow(*B*, *option*=`'NORMAL'`)

```
- Purpose
    Display a structuring element as an image.
- Synopsis
    y = mmseshow(B, option="NORMAL")
- Input
    B:      Structuring Element
    option: String Default: "NORMAL". 'NORMAL', ' EXPAND' or '
            NON-FLAT'
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmseshow used with the option EXPAND generates an image y that
    is a suitable graphical representation of the structuring
    element B . This function is useful to convert a structuring
    element to an image. The origin of the structuring element is at
    the center of the image. If B is flat, y is binary, otherwise, y
    is signed int32 image. When using the option NON-FLAT, the
    output y is always a signed int32 image.
- Examples
    #
    #   example 1
    #
    b=mmsecross(3);
    print mmseshow(b)
    a = mmseshow(b,'EXPAND')
    mmshow(a)
    print mmseshow(b,'NON-FLAT')
    #
    #   example 2
    #
    b=mmsedisk(2,'2D','EUCLIDEAN','NON-FLAT')
    print mmseshow(b)
```

mmsesum(*B*=`None`, *N*=1)

- Purpose N-1 iterative Minkowski additions
- Synopsis NB = mmsesum(B=None, N=1)
- Input B: Structuring Element Default: None (3x3 elementary cross). N: Non-negative integer. Default: 1.
- Output NB: Structuring Element
- Description mmsesum creates the structuring element NB from N - 1 iterative Minkowski additions with the structuring element B .
- Examplesdef mmsesum # # example 1 # b = mmimg2se(mmbinary([[1, 1, 1], [1, 1, 1], [0, 1, 0]])) mmseshow(b) b3 = mmsesum(b,3) mmseshow(b3) # # example 2 # b = mmsedisk(1,'2D','CITY-BLOCK','NON-FLAT'); mmseshow(b) mmseshow(mmsesum(b,2))

mmset2mat(*A*)

```
- Purpose
    Converts image representation from set to matrix
- Synopsis
    M = mmset2mat(A)
- Input
    A: Tuple with array of pixel coordinates and optional array of
       corresponding pixel values
- Output
    M: Image in matrix format, origin (0,0) at the matrix center
- Description
    Return an image in the matrix format built from a tuple of an
    array of pixel coordinates and a corresponding array of pixel
    values
- Examples
    #
    coord=int32([
      [ 0,0],
      [-1,0],
      [ 1,1]])
    A=mmset2mat((coord,))
    print A
    print mmdatatype(A)
    vu = uint8([1,2,3])
    f=mmset2mat((coord,vu))
    print f
    print mmdatatype(f)
    vi = int32([1,2,3])
    g=mmset2mat((coord,vi))
    print g
    print mmdatatype(g)
```

mmsetrans(*Bi*, *t*)

- Purpose Translate a structuring element
- Synopsis Bo = mmsetrans(Bi, t)
- Input Bi: Structuring Element t:
- Output Bo: Structuring Element
- Description mmsetrans translates a structuring element by a specific value.
- Examples # b1 = mmseline(5) mmseshow(b1) b2 = mmsetrans(b1,[2,-2]) mmseshow(b2)

mmseunion(*B1*, *B2*)

- Purpose Union of structuring elements
- Synopsis B = mmseunion(B1, B2)
- Input B1: Structuring Element B2: Structuring Element
- Output B: Structuring Element
- Description mmseunion creates a structuring element from the union of two structuring elements.
- Examples # b1 = mmseline(5) mmseshow(b1) b2 = mmsedisk(3) mmseshow(b2) b3 = mmseunion(b1,b2) mmseshow(b3)

mmshow(*f*, *f1*=`None`, *f2*=`None`, *f3*=`None`, *f4*=`None`, *f5*=`None`, *f6*=`None`)

- Purpose Display binary or gray-scale images and optionally overlay it with binary images.
- Synopsis mmshow(f, f1=None, f2=None, f3=None, f4=None, f5=None, f6=None)
- Input f: Gray-scale (uint8 or uint16) or binary image. f1: Binary image. Default: None. Red overlay. f2: Binary image. Default: None. Green overlay. f3: Binary image. Default: None. Blue overlay. f4: Binary image. Default: None. Magenta overlay. f5: Binary image. Default: None. Yellow overlay. f6: Binary image. Default: None. Cyan overlay.
- Description Displays the binary or gray-scale (uint8 or uint16) image f , and optionally overlay it with up to six binary images f1 to f6 in the following colors: f1 as red, f2 as green, f3 as blue, f4 as yellow, f5 as magenta, and f6 as cian. The image is displayed in the MATLAB figure only if no output parameter is given.
- Examples # f=mmreadgray('mribrain.tif'); f150=mmthreshad(f,150); f200=mmthreshad(f,200); mmshow(f); mmshow(f150); mmshow(f,f150,f200);

mmskelm(*f*, *B*=None, *option*='binary')

```
- Purpose
    Morphological skeleton (Medial Axis Transform).
- Synopsis
    y = mmskelm(f, B=None, option="binary")
- Input
    f:      Binary image.
    B:      Structuring Element Default: None (3x3 elementary
            cross).
    option: String Default: "binary". Choose one of: binary: output
            a binary image (medial axis); value: output a grayscale
            image with values of the radius of the disk to
            reconstruct the original image (medial axis transform).
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmskelm creates the image y by computing the morphological
    skeleton by B of the image f , when option is BINARY. In this
    case, the pixels of value 1 in y are center of maximal balls
    (generated from B ) included in f . This is also called Medial
    Axis. If option is VALUE, the non zeros pixels in y are the
    radius plus 1 of the maximal balls. This is called Medial Axis
    Transform or valued morphological skeleton.
- Examples
    #
    #   example 1
    #
    from Numeric import ones
    a=mmneg(mmframe(mmbinary(ones((7,9)))))
    print a
    print mmskelm(a)
    print mmskelm(a,mmsebox())
    #
    #   example 2
    #
    a=mmreadgray('pcbholes.tif')
    b=mmskelm(a)
    mmshow(a)
    mmshow(b)
    #
    #   example 3
    #
    c=mmskelm(a,mmsecross(),'value')
    mmshow(c)
```

mmskelmrec(*f*, *B*=`None`)

- Purpose Morphological skeleton reconstruction (Inverse Medial Axis Transform).
- Synopsis y = mmskelmrec(f, B=None)
- Input f: Gray-scale (uint8 or uint16) or binary image. B: Structuring Element Default: None (3x3 elementary cross).
- Output y: Binary image.
- Description mmskelmrec reconstructs the valued morphological skeleton to recover the original image.
- Examples # from Numeric import ones a=mmneg(mmframe(mmbinary(ones((7,9))))) print a b=mmskelm(a,mmsecross(),'value') print b c=mmskelmrec(b,mmsecross()) print c

mmskiz(*f*, *Bc*=`None`, *LINEREG*=`'LINES'`, *METRIC*=`None`)

```
- Purpose
    Skeleton of Influence Zone - also know as Generalized Voronoi
    Diagram
- Synopsis
    y = mmskiz(f, Bc=None, LINEREG="LINES", METRIC=None)
- Input
    f:       Binary image.
    Bc:      Structuring Element Default: None (3x3 elementary
             cross). Connectivity for the distance measurement.
    LINEREG: String Default: "LINES". 'LINES' or 'REGIONS'.
    METRIC:  String Default: None. 'EUCLIDEAN' if specified.
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmskiz creates the image y by detecting the lines which are
    equidistant to two or more connected components of f , according
    to the connectivity defined by Bc . Depending on with the flag
    LINEREG, y will be a binary image with the skiz lines or a
    labeled image representing the zone of influence regions. When
    the connected objects of f are single points, the skiz is the
    Voronoi diagram.
- Examples
    #
    #   example 1
    #
    f=mmreadgray('blob2.tif')
    y=mmskiz(f,mmsebox(),'LINES','EUCLIDEAN')
    mmshow(f,y)
    #
    #   example 2
    #
    from Numeric import zeros
    f=mmbinary(zeros((100,100)))
    f[30,25],f[20,75],f[50,50],f[70,30],f[80,70] = 1,1,1,1,1
    y = mmskiz(f,mmsebox(),'LINES','EUCLIDEAN')
    mmshow(f,y)
```

mmstats(*f*, *measurement*)

```
- Purpose
    Find global image statistics.
- Synopsis
    y = mmstats(f, measurement)
- Input
    f:
    measurement: String Default: "". Choose the measure to compute:
                 'max', 'min', 'median', 'mean', 'sum', 'std',
                 'std1'.
- Output
    y:
- Description
    Compute global image statistics: 'max' - maximum gray-scale
    value in image; 'min' - minimum gray-scale value in image; 'sum'
    - sum of all pixel values; 'median' - median value of all pixels
    in image; 'mean' - mean value of all pixels in image; 'std' -
    standard deviation of all pixels (normalized by N-1); 'std1' -
    idem, normalized by N.
```

mmsubm(*f1, f2*)

```
- Purpose
    Subtraction of two images, with saturation.
- Synopsis
    y = mmsubm(f1, f2)
- Input
    f1: Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image.
    f2: Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image. Or constant.
- Output
    y: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Description
    mmsubm creates the image y by pixelwise subtraction of the image
    f2 from the image f1 . When the subtraction of the values of two
    pixels is negative, 0 is taken as the result of the subtraction.
    When f1 and f2 are binary images, y represents the set
    subtraction of f2 from f1 .
- Examples
    #
    #   example 1
    #
    f = uint8([255,   255,    0,   10,   20,   10,    0,   255,  255])
    g = uint8([10,     20,   30,   40,   50,   40,   30,    20,    10])
    print mmsubm(f, g)
    print mmsubm(f, 100)
    print mmsubm(100, f)
    #
    #   example 2
    #
    a = mmreadgray('boxdrill-C.tif')
    b = mmreadgray('boxdrill-B.tif')
    c = mmsubm(a,b)
    mmshow(a)
    mmshow(b)
    mmshow(c)
```

mmsubm_old(*f1*, *f2*)

```
- Purpose
    Subtraction of two images, with saturation.
- Synopsis
    y = mmsubm(f1, f2)
- Input
    f1: Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image.
    f2: Unsigned gray-scale (uint8 or uint16), signed (int32) or
        binary image. Or constant.
- Output
    y: Unsigned gray-scale (uint8 or uint16), signed (int32) or
       binary image.
- Description
    mmsubm creates the image y by pixelwise subtraction of the image
    f2 from the image f1 . When the subtraction of the values of two
    pixels is negative, 0 is taken as the result of the subtraction.
    When f1 and f2 are binary images, y represents the set
    subtraction of f2 from f1 .
- Examples
    #
    #   example 1
    #
    f = uint8([255,   255,    0,   10,   20,   10,    0,   255,  255])
    g = uint8([10,     20,   30,   40,   50,   40,   30,    20,    10])
    print mmsubm(f, g)
    print mmsubm(f, 100)
    print mmsubm(100, f)
    #
    #   example 2
    #
    a = mmreadgray('boxdrill-C.tif')
    b = mmreadgray('boxdrill-B.tif')
    c = mmsubm(a,b)
    mmshow(a)
    mmshow(b)
    mmshow(c)
```

mmsupcanon(*f*, *Iab*, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Union of sup-generating or hit-miss operators.
- Synopsis
    y = mmsupcanon(f, Iab, theta=45, DIRECTION="CLOCKWISE")
- Input
    f:          Binary image.
    Iab:        Interval
    theta:      Double Default: 45. Degrees of rotation: 45, 90, or
                180.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
                ANTI-CLOCKWISE'
- Output
    y: Binary image.
- Description
    mmsupcanon creates the image y by computing the union of
    transformations of the image f by sup-generating operators.
    These hit-miss operators are characterized by rotations (in the
    clockwise or anti-clockwise direction) of theta degrees of the
    interval Iab .
```

mmsupgen(*f*, *INTER*)

```
- Purpose
    Sup-generating (hit-miss).
- Synopsis
    y = mmsupgen(f, INTER)
- Input
    f:     Binary image.
    INTER: Interval
- Output
    y: Binary image.
- Description
    mmsupgen creates the binary image y by computing the
    transformation of the image f by the sup-generating operator
    characterized by the interval Iab . The sup-generating operator
    is just a relaxed template matching, where the criterion to keep
    a shape is that it be inside the interval Iab . Note that we
    have the classical template matching when a=b . Note yet that
    the sup-generating operator is equivalent to the classical
    hit-miss operator.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [0,0,1,0,0,1,1],
       [0,1,0,0,1,0,0],
       [0,0,0,1,1,0,0]])
    i=mmendpoints()
    print mmintershow(i)
    g=mmsupgen(f,i)
    print g
    #
    #   example 2
    #
    a=mmreadgray('gear.tif')
    b=mmsupgen(a,mmendpoints())
    mmshow(a)
    mmshow(mmdil(b))
```

mmsupgen_old(*f*, *INTER*)

```
- Purpose
    Sup-generating (hit-miss).
- Synopsis
    y = mmsupgen(f, INTER)
- Input
    f:     Binary image.
    INTER: Interval
- Output
    y: Binary image.
- Description
    mmsupgen creates the binary image y by computing the
    transformation of the image f by the sup-generating operator
    characterized by the interval Iab . The sup-generating operator
    is just a relaxed template matching, where the criterion to keep
    a shape is that it be inside the interval Iab . Note that we
    have the classical template matching when a=b . Note yet that
    the sup-generating operator is equivalent to the classical
    hit-miss operator.
- Examples
    #
    #   example 1
    #
    f=mmbinary([
       [0,0,1,0,0,1,1],
       [0,1,0,0,1,0,0],
       [0,0,0,1,1,0,0]])
    i=mmendpoints()
    print mmintershow(i)
    g=mmsupgen(f,i)
    print g
    #
    #   example 2
    #
    a=mmreadgray('gear.tif')
    b=mmsupgen(a,mmendpoints())
    mmshow(a)
    mmshow(mmdil(b))
```

mmsuprec(*f*, *g*, *Bc*=None)

```
- Purpose
    Sup-reconstruction.
- Synopsis
    y = mmsuprec(f, g, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) or binary image. Marker image.
    g:  Gray-scale (uint8 or uint16) or binary image. Conditioning
        image.
    Bc: Structuring Element Default: None (3x3 elementary cross). (
        connectivity).
- Output
    y: Image
- Description
    mmsuprec creates the image y by an infinite number of recursive
    iterations (iterations until stability) of the erosion of f by
    Bc conditioned to g . We say that y is the sup-reconstruction of
    g from the marker f .
```

mmswatershed(*f*, *g*, *B*=None, *LINEREG*='LINES')

```
- Purpose
    Detection of similarity-based watershed from markers.
- Synopsis
    y = mmswatershed(f, g, B=None, LINEREG="LINES")
- Input
    f:       Gray-scale (uint8 or uint16) image.
    g:       Gray-scale (uint8 or uint16) or binary image. Marker
             image. If binary, each connected component is an object
             marker. If gray, it is assumed it is a labeled image.
    B:       Structuring Element Default: None (3x3 elementary
             cross). (watershed connectivity)
    LINEREG: String Default: "LINES". 'LINES' or ' REGIONS'.
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmswatershed creates the image y by detecting the domain of the
    catchment basins of f indicated by g , according with the
    connectivity defined by B . This watershed is a modified version
    where each basin is defined by a similarity criterion between
    pixels. The original watershed is normally applied to the
    gradient of the image. In this case, the gradient is taken
    internally. According to the flag LINEREG y will be a labeled
    image of the catchment basins domain or just a binary image that
    presents the watershed lines. The implementation of this
    function is based on LotuFalc:00 .
- Examples
    #
    f = uint8([
        [0,  0,  0,  0,  0,  0,  0],
        [0,  1,  0,  0,  0,  1,  0],
        [0,  1,  0,  0,  0,  1,  0],
        [0,  1,  1,  1,  1,  1,  0],
        [0,  1,  0,  0,  0,  0,  0],
        [0,  0,  0,  0,  0,  0,  0]])
    m = uint8([
        [0,  0,  0,  0,  0,  0,  0],
        [0,  1,  0,  0,  0,  0,  0],
        [0,  0,  0,  0,  0,  0,  0],
        [0,  0,  0,  0,  0,  0,  0],
        [0,  0,  0,  0,  0,  0,  0],
        [0,  0,  0,  2,  0,  0,  0]])
    print mmswatershed(f,m,mmsecross(),'REGIONS')
```

mmsymdif(*f1*, *f2*)

- Purpose Symmetric difference between two images
- Synopsis y = mmsymdif(f1, f2)
- Input f1: Gray-scale (uint8 or uint16) or binary image. f2: Gray-scale (uint8 or uint16) or binary image.
- Output y: Image i
- Description mmsymdif creates the image y by taken the union of the subtractions of f1 from f2 and f2 from f1 . When f1 and f2 are binary images, y represents the set of points that are in f1 and not in f2 or that are in f2 and not in f1 .
- Examples # # example 1 # a = uint8([1, 2, 3, 4, 5]) b = uint8([5, 4, 3, 2, 1]) print mmsymdif(a,b) # # example 2 # c = mmreadgray('tplayer1.tif') d = mmreadgray('tplayer2.tif') e = mmsymdif(c,d) mmshow(c) mmshow(d) mmshow(e)

mmtext(*txt*)

- Purpose Create a binary image of a text.
- Synopsis y = mmtext(txt)
- Input txt: String Default: "". Text to be written.
- Output y: Binary image.
- Description mmtext creates the binary image y of the text txt . The background of y is 0, while its foreground is 1. The text should be composed only by lower and upper case letters.

mmthick(*f*, *Iab*=`None`, *n*=`-1`, *theta*=`45`, *DIRECTION*=`'CLOCKWISE'`)

```
- Purpose
    Image transformation by thickening.
- Synopsis
    y = mmthick(f, Iab=None, n=-1, theta=45, DIRECTION="CLOCKWISE")
- Input
    f:         Binary image.
    Iab:       Interval Default: None (mmhomothick).
    n:         Non-negative integer. Default: -1. Number of
               iterations.
    theta:     Double Default: 45. Degrees of rotation: 45, 90, or
               180.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
               ANTI-CLOCKWISE'
- Output
    y: Binary image.
- Description
    mmthick creates the binary image y by performing a thickening of
    the binary image f . The number of iterations of the thickening
    is n and each iteration is performed by union of f with the
    points that are detected in f by the hit-miss operators
    characterized by rotations of theta degrees of the interval Iab
    .
```

mmthin(*f*, *Iab*=None, *n*=-1, *theta*=45, *DIRECTION*='CLOCKWISE')

```
- Purpose
    Image transformation by thinning.
- Synopsis
    y = mmthin(f, Iab=None, n=-1, theta=45, DIRECTION="CLOCKWISE")
- Input
    f:         Binary image.
    Iab:       Interval Default: None (mmhomothin).
    n:         Non-negative integer. Default: -1. Number of
               iterations.
    theta:     Double Default: 45. Degrees of rotation: 45, 90, or
               180.
    DIRECTION: String Default: "CLOCKWISE". 'CLOCKWISE' or '
               ANTI-CLOCKWISE'
- Output
    y: Binary image.
- Description
    mmthin creates the binary image y by performing a thinning of
    the binary image f . The number of iterations of the thinning is
    n and each iteration is performed by subtracting the points that
    are detect in f by hit-miss operators characterized by rotations
    of theta of the interval Iab . When n is infinite and the
    interval is mmhomothin (default conditions), mmthin gives the
    skeleton by thinning.
- Examples
    #
    f=mmreadgray('scissors.tif')
    f1=mmthin(f)
    mmshow(f,f1) # skeleton
    f2=mmthin(f1,mmendpoints(),15) # prunning 15 pixels
    mmshow(f,f2) # prunned skeleton
```

mmthreshad(*f*, *f1*, *f2*=`None`)

```
- Purpose
    Threshold (adaptive)
- Synopsis
    y = mmthreshad(f, f1, f2=None)
- Input
    f:  Gray-scale (uint8 or uint16) image.
    f1: Gray-scale (uint8 or uint16) image. lower value
    f2: Gray-scale (uint8 or uint16) image. Default: None. upper
        value
- Output
    y: Binary image.
- Description
    mmthreshad creates the image y as the threshold of the image f
    by the images f1 and f2 . A pixel in y has the value 1 when the
    value of the corresponding pixel in f is between the values of
    the corresponding pixels in f1 and f2 .
- Examples
    #
    a = mmreadgray('keyb.tif')
    mmshow(a)
    b = mmthreshad(a,uint8(10), uint8(50))
    mmshow(b)
    c = mmthreshad(a,238)
    mmshow(c)
```

mmtoggle(*f*, *f1*, *f2*, *OPTION*=`'GRAY'`)

- Purpose Image contrast enhancement or classification by the toggle operator.
- Synopsis y = mmtoggle(f, f1, f2, OPTION="GRAY")
- Input f: Gray-scale (uint8 or uint16) image. f1: Gray-scale (uint8 or uint16) image. f2: Gray-scale (uint8 or uint16) image. OPTION: String Default: "GRAY". Values: 'BINARY' or 'GRAY'.
- Output y: Image binary image if option is 'BINARY' or same type as f
- Description mmtoggle creates the image y that is an enhancement or classification of the image f by the toggle operator, with parameters f1 and f2 . If the OPTION is 'GRAY', it performs an enhancement and, if the OPTION is 'BINARY', it performs a binary classification. In the enhancement, a pixel takes the value of the corresponding pixel in f1 or f2 , according to a minimum distance criterion from f to f1 or f to f2 . In the classification, the pixels in f nearest to f1 receive the value 0 , while the ones nearest to f2 receive the value 1.
- Examples # # example 1 # f = uint8([0,1,2,3,4,5,6]) print f f1 = uint8([0,0,0,0,0,0,0]) print f1 f2 = uint8([6,6,6,6,6,6,6]) print f2 print mmtoggle(f,f1,f2) # # example 2 # a = mmreadgray('angiogr.tif') b = mmero(a,mmsedisk(2)) c = mmdil(a,mmsedisk(2)) d = mmtoggle(a,b,c) mmshow(a) mmshow(d) # # example 3 # e = mmreadgray('lenina.tif') f = mmero(e,mmsedisk(2)) g = mmdil(e,mmsedisk(2)) h = mmtoggle(e,f,g,'BINARY') mmshow(e) mmshow(h)

mmunion(*f1*, *f2*, *f3*=None, *f4*=None, *f5*=None)

```
- Purpose
    Union of images.
- Synopsis
    y = mmunion(f1, f2, f3=None, f4=None, f5=None)
- Input
    f1: Gray-scale (uint8 or uint16) or binary image.
    f2: Gray-scale (uint8 or uint16) or binary image. Or constant
    f3: Gray-scale (uint8 or uint16) or binary image. Default: None.
        Or constant.
    f4: Gray-scale (uint8 or uint16) or binary image. Default: None.
        Or constant.
    f5: Gray-scale (uint8 or uint16) or binary image. Default: None.
        Or constant.
- Output
    y: Image
- Description
    mmunion creates the image y by taking the pixelwise maximum
    between the images f1, f2, f3, f4, and f5 . When f1, f2, f3, f4,
    and f5 are binary images, y represents the union of them.
- Examples
    #
    #   example 1
    #
    f=uint8([255, 255,  0,  10,   0, 255, 250])
    print 'f=',f
    g=uint8([  0,  40, 80, 140, 250,  10,  30])
    print 'g=',g
    print mmunion(f, g)
    print mmunion(f, 255)
    #
    #   example 2
    #
    a = mmreadgray('form-ok.tif')
    b = mmreadgray('form-1.tif')
    c = mmunion(a,b)
    mmshow(a)
    mmshow(b)
    mmshow(c)
    #
    #   example 3
    #
    d = mmreadgray('danaus.tif')
    e = mmcmp(d,'<',80)
    f = mmunion(d,mmgray(e))
    mmshow(d)
    mmshow(e)
    mmshow(f)
    #
    #   example 4
    #
    g = mmreadgray('tplayer1.tif')
    h = mmreadgray('tplayer2.tif')
    i = mmreadgray('tplayer3.tif')
    j = mmunion(g,h,i)
    mmshow(g)
    mmshow(h)
    mmshow(i)
    mmshow(j)
```

mmvdome(*f*, *v*=1, *Bc*=None)

```
- Purpose
    Obsolete, use mmvmax.
- Synopsis
    y = mmvdome(f, v=1, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) image.
    v:  Default: 1. Volume parameter.
    Bc: Structuring Element Default: None (3x3 elementary cross).
        Structuring element (connectivity).
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    The correct name for this operator mmvdome is mmvmax.
```

mmversion()

- Purpose SDC Morphology Toolbox version.
- Synopsis S = mmversion()
- Output S: String (description of the version).
- Description mmversion gives the SDC Morphology Toolbox version.
- Examples # print mmversion()

mmvmax(*f*, *v*=1, *Bc*=None)

```
- Purpose
    Remove domes with volume less than v.
- Synopsis
    y = mmvmax(f, v=1, Bc=None)
- Input
    f:  Gray-scale (uint8 or uint16) image.
    v:  Default: 1. Volume parameter.
    Bc: Structuring Element Default: None (3x3 elementary cross).
        Structuring element (connectivity).
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmvmax This operator removes connected domes with volume less
    than v . This function is very similar to mmhmax , but instead
    of using a gray scale criterion (contrast) for the dome, it uses
    a volume criterion.
- Examples
    #
    #   example 1
    #
    a = uint8([
        [4,  3,  6,  1,  3,  5,  2],
        [2,  9,  6,  1,  6,  7,  3],
        [8,  9,  3,  2,  4,  9,  4],
        [3,  1,  2,  1,  2,  4,  2]])
    print mmvmax(a,10,mmsebox())
    #
    #   example 2
    #
    f = mmreadgray('astablet.tif')
    mmshow(f)
    fb = mmvmax(f,80000)
    mmshow(fb)
    mmshow(mmregmax(fb))
```

mmwatershed(*f*, *Bc*=None, *LINEREG*='LINES')

```
- Purpose
    Watershed detection.
- Synopsis
    y = mmwatershed(f, Bc=None, LINEREG="LINES")
- Input
    f:       Gray-scale (uint8 or uint16) or binary image.
    Bc:      Structuring Element Default: None (3x3 elementary
             cross). ( connectivity)
    LINEREG: String Default: "LINES". 'LINES' or ' REGIONS'.
- Output
    y: Gray-scale (uint8 or uint16) or binary image.
- Description
    mmwatershed creates the image y by detecting the domain of the
    catchment basins of f , according to the connectivity defined by
    Bc . According to the flag LINEREG y will be a labeled image of
    the catchment basins domain or just a binary image that presents
    the watershed lines. The implementation of this function is
    based on VincSoil:91 .
- Examples
    #
    f=mmreadgray('astablet.tif')
    grad=mmgradm(f)
    w1=mmwatershed(grad,mmsebox())
    w2=mmwatershed(grad,mmsebox(),'REGIONS')
    mmshow(grad)
    mmshow(w1)
    mmlblshow(w2)
```

sign(*val*)

- Purpose Determine the sign of a numeric value.
- Synopsis B = sign(val)
- Input val: scalar numeric value
- Output B: -1,0,1 depending on sign of numeric input
- Description sign determines the numeric sign of the input value, assigning a value of -1 for negative values, 1 for positive values and 0 for 0.
- Examples # b1 = sign(-4) print b1 b2 = sign(0.1) print b2 b3 = sign(0) print b3

uint16(*f*)

- Purpose Convert an image to a uint16 image.
- Synopsis img = uint16(f)
- Input f: Any image
- Output img: The converted image
- Description uint16 clips the input image between the values 0 and 65535 and converts it to the unsigned 16-bit datatype.
- Examples # a = int32([-3,0,8,100000]) print uint16(a)

uint8(*f*)

- Purpose Convert an image to an uint8 image.
- Synopsis img = uint8(f)
- Input f: Any image
- Output img: Gray-scale uint8 image. The converted image
- Description uint8 clips the input image between the values 0 and 255 and converts it to the unsigned 8-bit datatype.
- Examples # a = int32([-3,0,8,600]) print uint8(a)

11.2 Variables

Name	Description
__build_date__	**Value:** `'04aug2003 12:07'` *(type=`str`)*
__figs__	**Value:** `[None]` *(type=`list`)*
__version__	**Value:** `'0.8.1 numbase'` *(type=`str`)*
__version_string__	**Value:** `'SDC Morphology Toolbox V0.8.1 01Feb05 (numarray version)'` *(type=`str`)*
mydir	**Value:** `'/data/chulak1/dev/Multidrizzle/multireg'` *(type=`str`)*

12 Module multireg.objectlist

12.1 Functions

center1d(*region*)

Compute the center of gravity of a 1-d array. Based on 'mpc_getcenter' from IRAF imutil task 'center' in the cl.proto package.

find_center(*region*)

```
Compute the center of a star using MPC algorithm.
Based on 'mpc_cntr' from IRAF imutil task 'center'
in the cl.proto package.

Syntax:
    center = find_center(region)
Input:
    region - slice of array around target star
Output:
    center - array position of center as (y,x)
                relative to region origin
```

sumSlices(*a*, *b*)

Updates slice 'a' to be consistent with the indexing used for slice 'b'. For example, if slice 'a' was derived from slice 'b', the sum would return slice 'a' in the frame of 'b'.

12.2 Variables

Name	Description
__version__	**Value:** `'0.1.0 (30-November-2004)'` *(type=`str`)*
Chain_kernel	**Value:** `[0.10000000000000001, 0.20000000000000001, 0.40-000000000000002, 0.20000000000...` *(type=`list`)*

12.3 Class Object

12.3.1 Methods

__init__(*self*, *image*, *edges*, *region*, *index*, *binary*=`False`)

computeBinaryMoments(*self*, *edge*)

For this contour/object, compute the 7 invariant moments

computeChainCode(*self*, *cpix*)

Compute modified chain-code for each contour

computeMoments(*self*, *image*)

For this contour/object, compute the 7 invariant moments

12.4 Class ObjectList

```
This class manages the properties of detected objects
from multi-scale wavelet transformed image stacks.
Methods include:
    getObjects(scale=0)
    getPositions(scale=0)
    getRawPositions(scale=0)
    getSlices(scale=0)
```

12.4.1 Methods

__init__(*self*, *image*, *scale*=`0`, *offset*=`(0.0, 0.0)`)

addObjects(*self*, *image*, *scale*, *slices*=`None`)

buildObjectlist(*self*, *image*, *slices*=`None`)

Adds members to objectlist.

getChainCodes(*self*, *scale*=`0`)

Returns extracted chain codes for the scale specified.
This method ALWAYS returns a list, even if it only has 1 member.

getFluxes(*self*, *scale*=`0`)

Returns fluxes/total counts for each object at the scale specified.
This method ALWAYS returns a list, even if it only has 1 member.

getMoments(*self*, *scale*=`0`)

Returns computed invariant moments for the scale specified.
This method ALWAYS returns a list, even if it only has 1 member.

getObjects(*self*, *scale*=`0`)

Return FULL list of object instances for a given scale.

getPositions(*self*, *scale*=`0`)

Returns positions (center-of-gravity) for the scale specified.
This method ALWAYS returns a list, even if it only has 1 member.

getScales(*self*)

getSlices(*self*, *scale*=`0`)

Returns list of slices for member objects at the given scale.
This method ALWAYS returns a list, even if it only has 1 member.

verifyScale(*self*, *scale*)

Index