Environment for Modular Image Reconstruction Algorithms and Data Analysis (EMIRA+*da*)

If you use the system we would ask you to cite: M. Radermacher, *A New Environment for Modular Image Reconstruction and Data Analysis*, Microscopy and Microanalysis 19, S2, 2013, 762 - 763

Table of Content:

Introduction

EMIRA was designed to provide a user interface for a system of image reconstruction / processing modules, with imposing virtually no constraints on the programs that carry out the actual work. The system consists of management modules and a set of wrapper modules that provide an environment for easy wrapping of executable programs. The only requirement is that the binaries can be run on the command line (Programs that request multiple user input from standard input – like SPIDER – can be wrapped, however, the syntax of EMIRA will apply only to a limited extend). There is no requirement as to the language the source of the executables is written in. Even simple programs that only take input from standard input (unit 5 in FORTRAN) and write to standard output (unit 6 in FORTRAN) can be used. A main objective of the package is providing an environment where anybody can add a program, which then becomes usable interactively and within the scripting language of the package.

For installation see the separate INSTALLATION file in the distribution.

The usage syntax of the system has intentionally been kept simple, so that it is easy to learn. In addition, parts of the syntax were designed such that they can almost directly be interpreted by Python which keeps the code small.

The user language is inspired by the syntax of SPIDER, based on our experience that it has been very easy to teach this syntax even to inexperienced users.
The goals of further development are additions to the user interface without invalidating any current syntax (backwards compatibility), and to add further wrapper functions, again without invalidating currently existing ones. (The backwards compatibility of the wrapper functions may only be compromised if and when PYTHON 2.7 becomes obsolete. To keep any disruptions minimal, wherever possible, the code was written PYTHON 3 compatible.)

Currently the system is in the pre-alpha version. Error reporting is still in its infancy and the number of operations is still limited.

It is very easy to add your own operations to the system (see developer section). If you would like to have your programs distributed as modules of the EMIRA distribution, you can provide a tar file which includes the source code, the Makefile or command file to compile it, the binary for X86/64 (in case we we not have the compiler), the wrapper and a manual file that follows the template of the current manual chapters. We will create a directory “contributed” within the EMIRA distribution where these programs will be located and distribute them with the system. Please include your email in the manual chapter in case other users need to contact you.

User guide

Usage syntax:

Example of a small session (*user input in italics*):

emira
Project/data code: *prj/dat*
operation: *four*
input file: *image001*
output file: *fourier001*

operation: *end*

EMIRA uses a ***project*** and ***data code*** to uniquely identify a project.
By default the data code is used as the extension to all data files. The project code is attached to all script (batch) files results files and log files. Three characters are suggested for the data code, but longer project/data codes are possible. Operation (here “four”) is any operation defined either through a plugin, created as described below, or a built in operation. The name of operations can be split and is case insensitive. For example “Fourier Filter” is the same as “fourierfilter”. Current bug: “readdoc” and “writedoc” cannot be split.

Files used by EMIRA:

Image file names: Image file names are specified by a sequence of alphanumeric characters, ending in a multi-digit number and a “.”-separated extension which is the data code. Special symbols are not allowed (except “_”). If a file extension is specified it will not be replaced by the data code. Default image file format is the spider single image format. However, the image format in the end depends on the program that is run. Nothing prevents wrapping programs that for example use the MRC format. As a converter between image formats *bconvert* from the BSOFT package (http://lsbr.niams.nih.gov/bsoft/) has been implemented. A possible image name is *image1003file001.spi*

Document files: Document files are files that contain columns of numbers. The first column is a key; the second column specifies how many numbers follow in a line, then the specified number of numbers follows. The columns are separated by at least one space. The format is compatible with the 2012 format of SPIDER document files.

Results files: Each job creates a file named Results.<project code>.<version number>.
The version number is counted up with each new job in the same directory. The name of the Results file starts with a capital letter so that they can be differentiated from results files that spider may create.

Log files: Log files record every command that is run and can be used as input “batch” files to rerun. Analogous to results files Log files have the name
Log.<project code>.<version number> (again capital first letter). The version number is counted up with each run.

Batch files: Batch files are files that contain a script to be run (more later). The name of a batch file can be any **LOWER-CASE** alphanumerical sequence, followed by the project code extension.
example: batchfile.prj

Usage of EMIRA:

Example (from above)**:**
Start EMIRA by typing

emira	1
Project/data code: *prj/dat*	2
operation: *four*	3
input file: *image001*	4
output file: *fourier001*	5
operation: *end*	6

Line 1: The command “emira” starts emira
Line2: Enter the project and data code separated by a /. Suggested is to use three letters for each, but more are allowed.
Line 3: Enter an operation (see manual of operations).
Line 4: The operation fourier asks for an input file. Specify a name. The extension is automatically attached to all files if it is not explicitly specified.
Line 5: Enter the name of the output file.
Line 6: “end” ends the run of EMIRA.

Variables: EMIRA allows the usage of variables. Variables must start with “v.” followed by a character followed by any alphanumeric string. Examples: v.x, v.filter. (This convention was chosen, since it allows directly accessing a python function called “v” that evaluates all expressions. See module adm_expr.py)

Usage of variables:

Assignments and expressions:

operation: v.x=10
operation: v.other=v.x*100 etc.
All standard expressions are allowed. Expressions are directly interpreted by Python, therefore all assignments allowed in python are allowed here.

CAUTION: THE FOLLOWING VARIABLES CANNOT BE USED SINCE THEY ARE RESERVED NAMES IN PYTHON: The most tempting ones are **bold.**

`v.and`	`v.del`	**`v.from`**	`v.not`	`v.while`
`v.as`	`v.elif`	`v.global`	`v.or`	`v.with`
`v.assert`	`v.else`	`v.if`	`v.pass`	`v.yield`
`v.break`	`v.except`	`v.import`	`v.print`	
`v.class`	`v.exec`	`v.in`	`v.raise`	
`v.continue`	`v.finally`	`v.is`	`v.return`	
`v.def`	`v.for`	**`v.lambda`**	**`v.try`**	

Variable as numbers in file names:

Instead of specifying the file number explicitly it can be specified by variables. Example:
v.i=1
image[3] v.i creates the file name image001.prj
v.k=10
image[5]file[3] v.i v.k creates: image00001file010.prj
Numbers at any location in a file name can be substituted by variables. The number between [ ] specifies the number of digits used padded with 0s. The file name is followed by variables, separated by spaces. For each pair of [ ] a variable must be supplied.

Loops:

example:

for v.i from 1 to 100 step 2 do

```
fourier
image[3] v.i
fourier[3] v.i
enddo
```

The format of the loop statement is:

```
for <loop counter> from <start value> to <end value> [step <stepsize>] do
<statements to be executed within the loop>
enddo
```

The specification of a step size is optional. For readability the word “do” and “enddo” may be extended without spaces. These extensions are not interpreted by EMIRA and only serve as a reading help.

Example:

```
v.start=1
v.end=10
for v.k from v.start to v.end do1
for v.i from 1 to 100 step 2 do2
fourier
image[2]file[3] v.k v.i
fourier[2]file[3] v.k v.i
enddo2
enddo1
```

<u>if statements and logical flow control:</u>

Example:

```
if (v.x > 10) then
four
image[3] v.x
fourier[3] v.x
endif
```

Format of the if-statement:

```
if <logical expression> then
<statements to be executed within the if-block>
endif
```

<u>*Symbols within the logical expression:*</u>
== equal, > greater than, >= greater or equal, < smaller than, <= smaller or equal,
!= not equal . The expression after “if” is interpreted by Python and the complete
Python syntax is available.
At this time only the simple if-statement is available. “else” may be added in the future. At this time a negated logical expression needs to be used instead. Both, “then” and “endif” can be extended with identifiers that are not interpreted by EMIRA, analogous to do and enddo.

Comment text:

Comments start with the symbol “#” and can be entered after any input, or by itself. When a comment is entered in answer to “operation” the next request from EMIRA is again “operation”. A comment can also be entered in answer to a request for any other input. The input request then will be repeated. Restriction: when a comment is entered
instead of an answer to a program it must have the # sign in the first position (no spaces before it).

<u>Operations:</u>

Operations have the name of the module that defines them. For example the module for the operation four is em_four.py. em_four.py contains the wrapper for a binary that carries out the Fourier transform. Operations can be added by simply writing a python module with the name em_<operation> (<operation> must be the python function name in the def-statement.). EMIRA checks all directories specified in PYHONPATH. Operations may be split into several space separated words. For example the operation in the wrapper with name
em_radon2d.py may be called as “radon 2d”. Current bug: “readdoc” and “writedoc” cannot be split.

<u>Running batch files:</u>

Example:

operation: *@batchfile*

This will execute the script inside the file “batchfile.<current project code>. Batch file names must be LOWER CASE, while the calling of the batch file is case insensitive (i.e. internally all is converted for lower case).

New in version 0.1.3:

User created modules:

Now users can write their own operation/wrapper. For more detailed instructions see below. User defined operations need to be placed in directory:
<home directory>/emira_plugins.
To define a new command, create a file named: em_<command name>.py
that will contain your module and starts with: def em_<command name>():
For the rest see instructions below. **User defined operations have preference of system provided operations.** Thus, if you need to you can even replace an existing module with your own.

If, in addition you would like to create additional depositories, for example for a set of operations used only within your group, you can edit the file: emira, which is the one that is called first. This file contains the python path dedinitions. Note: the last specified path is searched first.

If you like to contribute you operation to the general distribution, please email the wrapper, the source code of the program that you are wrapping and the compilation commands (e.g. makefile). Code for redistribution must carry the GNU public license or compatible. If the files are too large, we can make arrangements for other file transfer methods.

Automatic display mode:
Now, by default output images of many operations are displayed using geeqie. The behavior is controlled by the operation “mode”. Options are no display, display of output images (default) and display of both, input and output images. In batch mode, the default is no auto display.
This behavior is controlled by the wrapper function adm_showpict. Since not all operations have images that make sense to display, image output must be specified in the wrapper.

In Development and not fully tested yet:

Procedure mechanism:
Variable assignments in the beginning of a batch file can be replaced by:

askfilename,v.file, please enter file name:
will pose the question to the calling process, if interactive to the user, and assign the answer to the variable v.file

asknumber, v.var, please enter value:
will pose the question to the calling process, if interactive to the user, and assign the answer to the variable v.var.

asktext, v.string, please enter something:
will pose the question to the calling process, if interactive to the user, and assign the answer to the variable v.string

If the procedure is called from a batchfile, the procedure answers must be preceeded with ‘<’.

More details to come.

For developers

Wrappers:

EMIRA comes with a set of python functions that should make it simple to write wrappers. The functions get input from the user, assemble the command line and run the program. Functions are available for:

- ask user for file name
- ask user for number(s)
- ask user for a character string
- assemble the command line string
- run the command line program
- auto display image

In addition, a wrapper needs to import a minimum number of libraries. The typical imports are:

```
import sys                                   1.
import adm_env                               2.
import adm_winput                            3.
from adm_run import adm_run                  4.
from adm_run1 import adm_run1                5
from adm_makeargs import adm_makeargs        6.
import pdb                                   7.
```

1. import the system system library
2. adm_env is the module that stores the EMIRA environment variables, like the project and data code, modes, the current (maybe redirected) input and output.
3. adm_winput is the module that contains many input functions.
4. adm_run is the module that runs the program
5. adm_run1 is a module that runs a program and returns values to variables
6. adm_makeargs assembles the command line
7. pdb is the python debugger. By importing it and placing a line “pdb.set_trace()” at the place in your wrapper that you want to debug, you can examine all statements and variables.

optional additional imports:

adm_printout is a module that prints to the user input device.
adm_expr is a module that stores all variables and evaluates expressions containing variables

Usage of wrapper functions:

Variables with in the code of the python modules of EMIRA all (almost) start with “adm_”
All wrapper functions also start with the prefix “em_”. This is done to avoid any conflict with added variables of modules. Currently all wrapper function contain two dummy arguments that must be provided as empty lists: “[ ]”. These will be used at a later stage but are part of the call now to ensure later backwards compatibility.

Get a file name

adm_**getfilename**(question,<optionals>):

Example:
adm_error, adm_filename = adm_getfilename(‘enter input file: ’,credir=’create’)

This call will echo to the user the text: ‘enter input file: ‘. Note that the space behind the “:” is part of the question. Please always use this extra space since it will look nicer in the Results file. The function will return a string with the filename and takes care of all number substitutions and the file extension. adm_error will be either ‘none’ or contain an error message if one occurred. At this time the program checks that the number of substitutions requested in the file name string matches the number of variables provided. If there is a mismatch the returned error string is ‘variable mismatch’ and the filename returned is an empty string.
optionals: If credir=’create’ is specified, the filename will be analyzed for a directory specification and, if the directory does not exist, the directory is created. Useful for output files.

Get one or more numbers:

adm_**getnumbers**(question,[defaults],<optionals>):

Example:
```
adm_error, adm_numbers=adm_winput.adm_getnumbers(\
          'p-dimension of Radon Transform, Mask Radius: ',\
          [64,30], type=’int’)
```

The question to the user will be:
'p-dimension of Radon Transform, Mask Radius: ‘
[defaults] is a list that must contain default values should the user hit return instead of answering or does not provide all the values. If three numbers are requested and only two numbers are entered, then the third number will receive the default value specified third in the list.
The return values are adm_error = ‘none’ if no error is detected. (At this time no errors are detected.)
adm_numbers will contain a list with the numbers answered (if a variable was given as an answer the value of the variable is returned.) If only one number is returned, adm_number is still a list and must be used as adm_numbers[0] (counting like in the C-languages, sorry).
optionals: type=’float’ or type=’int’ The type= specification converts the input to the specified type (integer or floating point), independent of which type was provided by the user.

Get a character-string answer:

adm_**getstring**(question,case,maxlength,’default’):

Example:

```
adm_error, adm_thresh=adm_winput.adm_getstring(\
'Threshold: Average threshold, Threshold, Lift, None (A/T/L/N): ','upper',0,'N')
```

The question to the user will be: 'Threshold: Average threshold, Threshold, Lift, None (A/T/L/N): ‘
case = ‘upper’ will return a string in all capital letters
case = ‘lower’ will return a string in all lower case letters
case = ‘none’ will return the unchanged user input.

max_length is the maximum length of the answer returned to the wrapper. This allows the user to answer with a longer string of which only the max-length number of letters are used. If 0 is given, any length will be returned. ‘default’ is a default answer string.
Return values are, using the names in the example:
adm_error = ‘none’ or an error message if one was detected. At this time no errors are detected.
adm_thresh = <answered string>

Get string of numbers (e.g. 1,4-10)

adm_getnumstring(question,default,optionals)

Get a string containing numbers. An example of strings are: 1-100,110,20-40 or
v.num1-v.num1,20-40. This query was mainly created to serve the getnum subroutine
in SPIDER. For example, the averaging operation asks for the list of files numbers of
the images to be averaged. Optionals are type=’float’ or ‘int’.

Built the argument part of the command line:

adm_**makeargs**(adm_list)

This creates the argument list for the binary that is run. All inputs are provided as a comma separated list.

Example:

```
adm_error, adm_argstring = adm_makeargs(\
      ['-inf',filename1,'-outf',filename2,\
      '-inc',adm_inc[0],'-xdim',adm_dims[0],'-rmask',adm_dims[1],\
      '-shx',adm_cshift[0],'-shy',adm_cshift[1],'-th',adm_thresh,\
      '-upper',adm_tvalues[0],'-lower',adm_tvalues[1]])
```

This may create the string (depending on the user’s answer):
adm_argstring=’-inf image001.dat –outf radon001.dat –dim 124.0 –rmask 45.0 –shx 0.0 –shy 0.0 –th N –upper 0.1 –lower 0.05’

where image001.dat and radon001.dat are answers received by adm_getfilename
the values 124, 45, 0.0, 0.1 and 0.05 are received by adm_getnumbers. The answers to shx and shy were received by the same call to adm_getnumbers, therefore shx is in list position [0] and shy in position [1]. The letter N was received by adm_getstring. The arguments to makeargs are provided in a comma separated list. If the variable debug is set to True, the assembled argument string is echoed to the terminal.

Run the program:

adm_error = adm_run(‘program’,adm_argstring)

This call runs the program “program” with adm_string as command-line argument.

alternative:

adm_error = adm_run1(‘program’,adm_argstring, outtuple, returnvalues, returnnames) outtuple will contain the values retrieved from the program output.

returnvalues must be specified as a list of stings that precede the values in the standard output of the program
example: adm_returnvalues=['FMIN =', 'FMAX =', 'AV =' , 'SIG =']

returnnames must be specified as the list of names under which the values are stored in the return tuple.
example: adm_returnvalues=['min',,max','avg','sig'].

<u>Auto-display image:</u>

adm_showpict(filename,inext='spi',inout='out')
arguments: filename - image name,
inext - extension that specifies file type,
inout - specify if image is input or output image.

<u>Example of usage:</u>

```
emira mix/dr1
files opened: Results.mix.24 log.mix.24
operation: filestats                          #operation , find file statistics
Input file: vismap045                         #input image
Enter variable to receive output :v.vismap    #tuple name that receives the output
```

Remarks:
this is the actual command line program that runs:

```
filestatistics  -inf1 vismap045.dr1 -mode N
```

this is the output of the program that does the calculation:

```
vismap045.dr1
(R )  1000  1000 CREATED 29-MAY-2013 AT 16:44:55  O HEADER BYTES:   4000
FMIN = -0.486294     FMAX =  0.679143     AV =  0.101817E-01 SIG = 0.120829
```

the values are retrieved according to adm_returnvalues=['FMIN =', 'FMAX =', 'AV =' , 'SIG =']
EMIRA continues with:

```
operation: print v.vismap
filestatistics(min=-0.486294, max=0.679143, avg=0.0101817, sig=0.120829),
```

the values are stored according to: adm_returnvalues=['min','max','avg','sig'].
and can be used as below:

```
v.maximum=v.vismap.max

operation: end
end
```

The following problem should be fixed by now but did not undergo sufficient testing yet:
Caution: for retrieving values, the identifiers must be unique. If in the example above "fmin=" is found multiple times in the program output, a mismatch between the number of retrieved values and the places provided in the return tuple may occur will cause the program to crash.

Wrapper example:

Example of a very simple wrapper for operation 'copy' that wraps the linux command cp:

```
def em_copy():
        import adm_winput
import sys
import adm_env
import adm_winput
import pdb
from adm_run import adm_run
from adm_makeargs import adm_makeargs

adm_error, filename1 = adm_winput.adm_getfilename('Input file: ')
adm_error, filename2 = adm_winput.adm_getfilename('Output file: ',\
        credir='create')

adm_args=adm_makeargs([filename1,filename2])
adm_run('cp',adm_args)
```

New Developments

Help with further development is appreciated. Developments need to be in agreement with the basic design philosophy of EMRIA. These are: Keep backwards compatibility of wrappers and of user scripts.

Keeping wrappers compatible should not be too difficult, using the feature of Python that allows for the predefinition of optional variables.

User interface backwards compatibility is a design issue and must be explicitly taken care of.

Expansions to the system will be distributed after review for stability and compatibility with the system's design principles.

Currently planned developments are:

Expand the if-statement to if-else. No further expansion of the if-statement is planned. If-else should allow for all needed logical constructs. More elaborate if-constructs would unnecessarily increase the complexity of the user syntax. Keeping the user interface syntax simple is one of the design principles.

Create a procedure mechanism similar to the one existing in SPIDER.

Other

Appendix

Example batch file

Example of a batch file for 2D alignment using 2D Radon transforms. (equivalent to the spider procedure radali.sys

```
v.inputimage='../imcb/imb[5]'       #input image string
v.firstimg=1                        #first image number
v.lastimg=362                       #last image number
v.selfile='sel001'                  #selection document file (0 and 1)
v.reference='rfreerefshiftmlp001'   #reference image
v.documentfile='radalidoc001'       #output alignment doc file
v.outputimage='../imcc/imc[5]'      #output image
# the peakfile needs to be specific,
# number is calculated inside the binary.
v.peakfile='../radalipeak/peak00001'#name of first cross-correlation
v.radondim=160                      #Radon transform dimension
v.radonrefmask=52                   #mask radius of ref.Radon transform
v.radonmask=65                      #mask radius for images
v.radincrement=1                    #Radon transform increment
v.lowpass=.123                      #low-pass filter
v.highpass=.02                      #high pass filter
v.shiftrange=15                     #maximum shift
v.raddirectory='../radalirad/'      #directory for Radon transforms
v.skip=0                            #which part of batchfile to skip
#---------------------------------------------------------
# define intermediate files, this is PYTHON string syntax:
v.imageradon=v.raddirectory+'rad[5]'
v.imageradfour=v.raddirectory+'radf[5]'

# radon transform of reference:
if(v.skip <1 ) then
rad2d
v.reference             #Input file:
scr_refrad001           #Output file:
160                     #p-dimension of Radon Transform:
60,1.                   #Mask radius, Angular Increment:
0,0                     #Center Offset in x,y:
n     #Threshold: Average threshold, Threshold, Lift, None (A/T/L/N):

radfour
scr_refrad001           #Input file:
scr_refradfour001       #Fourier Output file or *:
*                       #Filtered Radon transform or *:
n                       #Pad to larger dimension? (P) :
y                       #Fourier Filter? (Y/N) :
1                       #Number of filters (max 3) :
8                       #Filter type (1-6):
# Preserve Average (P),
# Divide average by number of rows (D)
#
a                       # Apply filter also to average (A) :
n                       #Amplitude normalization? (Y/N) :
n                       #Sigma normalization? (Y/N) :
```

```
#Radon transform the image series:
for v.i from v.firstimg to v.lastimg do
readdoc
v.selfile
v.i,v.flag
if (v.flag > 0.5) then

rad2d
v.inputimage v.i              #Input file:
v.imageradon v.i              #Output file:
v.radondim                    #p-dimension of Radon Transform:
v.radonmask,v.radincrement    #Mask radius, Angular Increment:
0,0                           #Center Offset in x,y:
n     #Threshold: Average threshold, Threshold, Lift, None (A/T/L/N):

radfour
v.imageradon v.i              #Input file:
v.imageradfour v.i            #Fourier Output file or *:
*                             #Filtered Radon transform or *:
n                             #Pad to larger dimension? (P) :
y                             #Fourier Filter? (Y/N) :
3                             #Number of filters (max 3) :
8                             #Filter type (1-6):
5                             #Filter type (1-6):
v.lowpass                     #Radius:
.02                           #Temperature:
6                             #Filter type (1-6):
v.highpass                    #Radius:
.01                           #Temperature:
# Preserve Average (P),
# Divide average by number of rows (D)
#
a                             # Apply filter also to average (A) :
n                             #Amplitude normalization? (Y/N) :
n                             #Sigma normalization? (Y/N) :
endif
enddo
endif #v.skip <0

if(v.skip <2) then

radalign2d
scr_refradfour001             #Input reference Radon Fourier file:
v.imageradfour    1           #Input Radon Fourier file example:
v.documentfile                #Output document file:
v.peakfile                    #Output 3D ccf or *:
v.firstimg-v.lastimg          #file numbers:
*                 #Optional single ccf output file (* if not wanted):
-180,179,1                    #Search angles: from, to, increment:
v.shiftrange                  #Maximum shift in pixels:
v.radali                      #Enter variable to receive output :
print v.radali

endif #v.skip <1

if(v.skip < 3) then
for v.i from v.firstimg to v.lastimg do
```

```
readdoc
sel001
v.i, v.sel
if v.sel > 0 then
readdoc
radalidoc001
v.i, v.max, v.ang, v.x, v.y, v.cang, v.cx, v.cy
shift
v.inputimage v.i
scratch001
-v.cx, -v.cy
F
rot
scratch001
v.outputimage v.i
-v.cang
endif
enddo

endif # v.skip<2
end
```

(incomplete) List of operations:

For a complete list see the current manual chapters in the “manual” directory

add	add 2 images
appenddoc	append and ascii file to another
ask	Ask for input values in a procedure
average	calculate averages of an image series
averagehistogram	calculate histogram for all point in an average
backproject	calculate a simple backprojection
bconvert	Convert image formats
bin	Bin down image
bin3d	Bin down volume
bindensity	density" Bin down image and convert to optical densities
calcproshifts	shift projections based on 3d volume shift
calcslope	calculate the tunning slope of a curve, averaged over a
calczinplane	Calculate a z-value in a plane based on the x,y coordinates
chim2euler	Convert UCSF Chimera Matrix to EMIRA document file
classididay	diday Classification with moving centers.
combineeuler	euler Combines 2 Euler rotations into a single one
convertmarkers	convert marker selected in IMOD for tomography
copy	Copy a file (any format)
copyspider	copy spider files (optional stacks)
createdir	create a new directory
crossco	Cross-correlate two images, not normalized.
crossconorm	Cross-correlate two images, normalized.
delete	delete a file (any format)
display	display a 2D image
divide	Divide one image (volume) by another
exposurelog2doc	Convert the exposure log file from a tecnai microscope
filestats	Get file statistice (Min,MAx,AV,SIG)
filestatsmask	Get file statistice (Min,MAx,AV,SIG)
findfourdimensions	Find next higher/lower dimension f. Fourier transf.
fitplane	Fit a plane to a set of x,y,z coordinates
fittomoexposure	tomo exposure Fit a curve to the average densities of a file series.
four	Calculate the Fourier transform of an image of volume
fourfilter	Apply a Fourier Filter to the Fourier transform
getfilenumbers	extract the file numbers from a file series and put
getheader	Get header information from SPIDER file

getprocenters	Program to get projection centers in a tilt series
hsearchlorentz	lorentz Search for helical pitch
imcfromppcaem	Convert the ppcaem output coordinate file to
imodcoordinates2doc	Convert 3D coordinate file from IMOD to SPIDER-style
mask	Apply a round mask to an image or volume
model	Create model image (2D)
montage	montage many images into one large one
msa	Run multivariate statistical analysis
msaimc2doc	imc2doc Convert image coordinate file created by MSA to document file
msamap	Print 2D maps from PCA or coran coordinates.
multiply	Multiply two images
multiplyconjugate	Multiply two Fourier transforms
new2olddoc	Convert new format document file to old format document file,
pba3	Peak file averaging for projection based 3D alignment
pickstickturn	Program to box out the projection series of an
plothisto	plot a histogram from a document file to a postscript file
plotlines	lines Plot lines from a document file.
plotscatter3d	Create a 3D scatter plot
ppcaem	PROBABILISTIC PRINCIPLE COMPONENT ANALYSIS WITH EXP. MAX.
project	project a 3D volume onto 2D projection
qvol	vector quantization of volume (SITUS program)
rad2d	Calculate a 2D Redon transform
rad3d	Calculate a 3D Radon transform
rad3dinv	Invert 3D Radon transform
radalign2d	2D image alignment with Fourier-Radon transforms
radalignto3d	2D Radon transform alignment to 3D Radon transform
radfour	Calculate the 1D Fourier transform of a 2D or 3D Radon transform
radmake3d	Create an empty 3D Radon transform (or Fourier-Radon
radsigstat	Calculate histogram of sigmas in Fourier Radon transforms
radsum3d	Average 2D Radon transforms into 3D Radon transforms.
readdoc	read values from a document file
readdocclose	Close document file to free up memory
removeoutliers	outliers Remove outliers from document file columns
rot	rotate an image
scalevalues	Scale density values in an image
setangle	Put Euler angles in the header of an image.
shift	shift an image or volume

table2doc	convert whitespace seprated table to SPIDER document file.
threshold	Theshold an image
variables	their use in EMIRA
vs2doc	convert image coordinates from an XMIPP neural net to
window	Box out small image or volume from large image or volume
writedoc	write numbers into a document file
writedocclose	Close document file to make is useful for reading.
zeiss2spider	convert Zeiss/SCAI scanner tif to spider format

CHEAT SHEET OF RESTRICTIONS and troubleshooting hints:

- readdoc and writedoc operations cannot have spaces in the call (will be corrected in the future)

- THE FOLLOWING VARIABLES CANNOT BE USED SINCE THEY ARE RESERVED NAMES IN PYTHON: The most tempting ones are bold.

v.and	v.del	v.from	v.not	v.while
v.as	v.elif	v.global	v.or	v.with
v.assert	v.else	v.if	v.pass	v.yield
v.break	v.except	v.import	v.print	
v.class	v.exec	v.in	v.raise	
v.continue	v.finally	v.is	v.return	
v.def	v.for	v.lambda	v.try	

- Variable name must not start with a digit: example: v.3fold is invalid, v.fold3 is valid.

- All pseudo batch file names must be lower case.

- All wrapper file names must be lower case.

Some trouble shooting hints:

FREQUENTLY OCCURRING ERROR:

example of error from shift with v.xshift
AttributeError: 'function' object has not attribute xshift
means that v.xshift was not defined, i.e. no value had been assigned to it. Often caused by a misspelling in the assignment.

If a program seems not to do what it should do, check the Results file. For each operation it lists the command line of the program called. Paste this line into a terminal window to run by itself. Sometimes you get more informative error messages, which often solve the problem. You can also use the command line to debug the program.