`Technical Report No.2`

MEM Package for Image Restoration in IRAF

Nailong Wu

SCARS/STSDAS, STScI; CCS, NOAO
Beijing Observatory, Chinese Academy of Sciences

October 1992

The package `mem0` for image restoration by the Maximum Entropy Method (MEM) in IRAF has been updated to its version b. The package `mem0 v.b` consists of five tasks. They are:

irme0b	–	Perform 2-D deconvolution for an image by MEM
immakeb	–	Generate an image of Gaussian type
imconvb	–	Convolve an image with a point spread function
irfftesb	–	Test the 2-D FFT procedure
foolfactor	–	Factorize a natural number

Their usage *etc.* are described in this report.

1. irme0b

USAGE

irmeb0 degrade psf prior icf restore vc0 vc1 vc2 tp nppsf

PARAMETERS

degrade
: Input degraded image to be deconvolved.

psf
: Input data file of the point spread function.

prior
: Input prior estimated image. Enter " ", *i.e.*, two double quotation marks, on the command line for none.

icf
: Input data file of the Intrinsic Correlation Function. Enter " " on the command line for none.

restore
: Output restored image.

vc0

Uniform noise variance.

vc1

Coefficient for calculating Poisson noise variance.

vc2

Coefficient for calculating other noise variance.

tp

The total power (flux) of the image. Enter 0 for automatic calculation.

nppsf

The number of pixels contained in the p.s.f., *i.e.*, its volume. Enter 0 for automatic calculation from psf.

sigma=0.0,0.0

Sigmas of an elliptic Gaussian function used as ICF if no icf file is provided. Enter 0 for using another parameter fwhm (full width at half maximum).

fwhm=0.0,0.0

FWHMs of the Gaussian function (ICF).

a_sp=10

Speed factor for renewing the Lagrange multiplier alpha for the data fit constraint.

a_rate=0.5

Reduction rate for a_sp, used when a_sp is too large (alpha increases too fast) so that an ME solution has not been found after 7 or 15 or 23 ... iterations for fixed alpha and beta.

b_sp=3

Speed factor for renewing the Lagrange multiplier beta for the total power.

maxiter=30

Prescribed maximum number of iterations.

tol=0.05,0.05,0.05

Convergence tolerances for ME solution, data fit, and total power.

opt=yes

The maximum value of the objective function is searched along the direction determined by the Newton–Raphson method. If opt=yes, an optimal step is determined by one–dimensional search, otherwise step=1.

hidden=no

The restored hidden image may be convolved with ICF to get an apparent image. Set hidden=yes to output the hidden image, or use hidden=no to output the apparent image.

message=1

For the detailedness of the output messages when running the task, 1 (lest) – 3 (most).
message=1: basic iteration summary only.
message=2: plus input summary.
message=3: plus additional iteration summary.
The meanings of messages will be explained later.

DESCRIPTION

This is a task for 2-D image restoration by MEM. The Newton–Raphson method for optimization is used. (See Cornwell, J. and Evans, K., *Astron. Astrophys.*, **143**, pp.77–83 (1985)). Generally speaking, MEM is very good in resolution enhancement and noise suppression. However, because of its nonlinearity, the processing is time consuming, and it is hard to use the results for photometry. The difficulty in choosing proper parameters may be another problem.

In image restoration by MEM, the entropy of the image (in the 1-D notation)

$$H2 = -\sum_j b_j \log(b_j/em_j) \tag{1}$$

is maximized subject to the data (mis)fit constraint in the image domain

$$E = \chi^2 = \sum_j |\sum_k p_{jk} b_k - d_j|^2/\mathrm{var}(j) \le E_\mathrm{c} \tag{2}$$

and the total power constraint

$$F = \sum_j b_j = F_\mathrm{ob} \tag{3}$$

where b_j represent the image to be restored, m_j the prior estimate of the image, d_j the degraded image, var(j) the noise variance; p_{jk} is the p.s.f.; the subscript c stands for critical value, and ob for observational value.

Form the objective function

$$J = H2 - \alpha E - \beta F \tag{4}$$

then use the Newton–Raphson method to find an ME solution, *i.e.*, maximize J for particular values of the Lagrange multipliers α and β. The desired values of E and F are achieved by choosing appropriate α and β.

The Newton–Raphson method is efficient in optimization. However, its exact implementation needs inversion of matrices of large size in image restoration. This cannot be done in practice. So some kind of approximation is inevitable. In the underlining algorithm for this task, the solution is to simply ignore the non-diagonal elements of the p.s.f. (p_{jk}) and increase the diagonal ones for compensation so that the matrix $\nabla\nabla J$ becomes a diagonal one. In this way the inversion of the matrix becomes a simple operation. This zeroth-order approximation is the basis of the level 0 package `mem0`.

Choosing appropriate parameters is always a rather difficult matter in running an MEM program. Much effort in writing the program has been made for the user's comfort. The following is some suggestions and comments.

Image sizes

The input degraded image, p.s.f., prior image and ICF may have different sizes. They need not be a power of two. The actual sizes of the arrays holding the images in the program are equal to the maximum of the degraded image and p.s.f. sizes. The read-in areas of the prior image and ICF, if provided by the user, will not exceed this maximum. The output deconvolved image will have the same size as the input degraded image. As the general guideline, keep the p.s.f., prior image and ICF in the smallest sizes possible. Perform deconvolution only for the degraded image's area of interest. 6 real and 2 complex arrays are needed in the deconvolution procedure. So, for example, to process a 512×512 image, the required core memory is somewhat more than $1.0 \times 10 = 10$MB.

psf

The peak of p.s.f. need not be centered or normalized in the input file. The program takes care of this.

prior

In the first run of the task, the prior image may be a filtered degraded image. If a flat prior is to be used, simply enter a space. In the subsequent runs, the previous ME solutions may be used as the priors.

vc0, vc1 and *vc2*

The formula for calculating the total noise variance from these coefficients is:

$$\mathrm{var}(j) = vc0 + vc1 \cdot |d_j| + vc2 \cdot d_j^2$$

where the first term may be interpreted as the uniform (signal and position independent) noise variance, the second term as Poisson noise variance, and the third term as other noise variance. For instance, with the WF/PC of the Hubble Space Telescope, from the instrument handbook, $vc0$ is the constant noise in the data, equal to $(13/7.5)^2 = 3.0$; $vc1$ is the reciprocal of edu (analogue-to-digital unit) or gain, equal to $1/7.5 = 0.1333$; $vc2$ may be set to zero. (The noise level may be higher if estimated from the degraded image.) With FOC, $vc0 = vc2 = 0$, $vc1 = 1$.

These coefficients can be estimated from the degraded image. In the case of zero-mean Gaussian noise, for example, the noise rms may be estimated from the noisy background, or simply set to, say, 1% of the image maximum. Then $vc0 = rms^2$. Don't forget the square operation. If you have more than one independent source for any type of noise, just sum up the individual coefficients to get vc for that noise.

tp

The pixel number or volume of the p.s.f. is actually normalized to one before the deconvolution procedure is started. Therefore, the total power of the image is conserved after deconvolution.

A non-zero *tp* provided by the user will be used for the constraint. If its value is zero, the program will look for the keyword `ME_TP` in the prior image (if provided by the user). The existence of this keyword indicates that the prior is an ME image from the previous run of the task, and its value will be taken as *tp*. In this way, a constant *tp* will be used in a step-by-step deconvolution of an image. If the user proveded *tp* is zero and no `ME_TP` is found, the total power of the degraded image will be taken as *tp*. In optical image deconvolution, normally the user doesn't have to take care about *tp*.

nppsf

Before the deconvolution is started, the p.s.f. data from the psf file is normalized twice. First, its peak is normalized to one. Then the actual number of pixels, *i.e.*, volume, of the p.s.f. is calculated simply by summing up all the positive pixel values. This value is output as `NPPSF`, and assigned to *nppsf* if a zero-valued *nppsf* has been entered by the user. An ideal p.s.f. is a delta function, so `NPPSF`=1. A large `NPPSF` indicates that the p.s.f. deviates far from the delta function. Consequently, the deconvolution will be more difficult. Tests so far have shown that the task works satisfactorily for `NPPSF` up to about 20.

Second, the p.s.f. is FFTed and combined with the FFT of the volume-normalized ICF. The DC term of this combination (*i.e.*, the volume of p.s.f.∗ICF) is normalized by `NPPSF` to one for the purpose of total power conservation in deconvolution. *nppsf* is normalized in the same way. Therefore, the *nppsf* used in the iteration is actually $nppsf/$`NPPSF`. This value will be greater or less than one depending on whether the user entered non-zero *nppsf* is greater or less than `NPPSF`. In this way the user can have control over the damping in the approximation. Normally, a zero *nppsf* should be entered by the user. The program will take care of everything. $nppsf/$`NPPSF` will be equal to unity (default damping).

icf, sigma, fwhm and *hidden*

The image formation is modeled like this:

hidden image ∗ ICF = apparent image,
apparent image ∗ p.s.f. = blurred image,
blurred image + noise = degraded image

where ∗ denotes 2-D convolution; hidden image, for which the entropy is defined, has no correlations between its pixels; apparent image is what we really see. The above steps may be combined:

hidden image ∗ ICF ∗ p.s.f. + noise = degraded image.

The correlations between pixels in the apparent image are introduced by convolving the hidden image with ICF.

ICF may be input from a data file. If no file is provided, an elliptic Gaussian function is generated as ICF. Its sigma, or FWHM in each dimension may be entered, $sigma = fwhm/\sqrt{8\ln 2}$. In the first dimension, for example, if *sigma[1]*=0, then *fwhm[1]* will be read in and *sigma[1]* will be calculated from *fwhm[1]*. Using a very small number, say 1.0E–4, will virtually set *sigma[1]* to zero and ignore *fwhm[1]*. *sigma[2]* and *fwhm[2]* are treated in the same way. By default, *sigma[1]-[2]* and *fwhm[1]-[2]* are zero. Then the hidden image is identical to the apparent image.

The result from deconvolution is a hidden image, which may be output if *hidden* = `yes`, or convolved with ICF to get the apparent image before output if *hidden* = `no`.

a_sp, a_rate and *b_sp*

The Lagrange multiplier *alpha* increases gradually, starting with zero, in the iteration to redunce E. Its increase speed, and consequently the reduction speed of E, is controlled by the parameter *a_sp*, which may be empirically set to $5 \sim 20$ for reducing E at reasonable speed. If *alpha* increases too fast (*a_sp* is too large), then for a particular *alpha* a large number of iterations may be needed to find an ME solution. (This is number, `inner_iter`, may be output if *message* = 3.) If this happens, the current *alpha* and *a_sp* will be reduced when mod(`inner_iter`,8)=0. The rate of reduction depends on the parameter *a_rate*: $a\_sp = a\_sp \times a\_rate$. *a_rate*=0.5 by default.

The parameter *b_sp* plays a similar role as *a_sp*. However, the constraint on the total power is not as crucial as the one on data fit, so *b_sp* is set to a smaller value, being 3 by default. No parameter like *a_rate* is needed for *beta* and *b_sp*.

maxiter

The maximum number of iterations is prescribed so that the task may not run forever. After *maxiter* iterations, if an ME solution for the final *alpha* and *beta* has not been found, maximally extra 5 iterations are allowed.

tol

This array contains the convergence tolerances for ME solution, data fit and the total power, respectively. 0.05 is a reasonable default value for them.

opt

In searching for the maximum value of the objective function, a full step(=1) is first taken in the direction determined by the Newton–Raphson method. This step may not be optimal. So if *opt* = `yes`, then an optimal step is calculated and taken by one–dimensional search. This "optimization" is rather rudimental. Therefore, step is limited to be not greater than 4.

message

Most output messages are self-explanatory or can be understood on the basis of the above description. Additional information is given in the following.

`|gradJ|/|1|`: ratio between the magnitudes of the gradient of the objective function J and unit vector, used to indicate the degree of approximation to the ME solution. (This value is zero for an exact ME solution.) The tolerance *tol[1]* is displayed in parentheses.

`test`: the value $1 - \cos < \nabla H2 - \beta\nabla F, \nabla E >$. This is an indication of the parallelism between the two vectors shown in the above. This value is zero for an exact ME solution.

`An ME image obtained`: An ME solution has been obtained for the final *alpha* and *beta*, that it, the objective function has been maximized, `|gradJ|/|1|`$\leq$*tol[1]*.

`Congratulations for convergence !!`: The iteration has converged, *i.e.*, the entropy has been maximized under the data fit and total power constraints.

Some parameters and statistics about the restored image are written into the output image header file. It is a pity that there is no simple way to write comments into the user defined header cards. All the cards written by the task have a prefix `ME_`. The meanings of the keywords are as follows.

`ME_VC0, ME_VC1, ME_VC2`: coefficients *vc0-2* for calculating the noise variance used for deconvolution.

`ME_TP, ME_SIGM1, ME_SIGM2, ME_FWHM1, ME_FWHM2`: parameters *tp, sigma* and *fwhm* used for deconvolution.

`ME_HIDDN`: a hidden (or apparent) image?

`ME_MEIMG`: an ME image?

`ME_CONVG`: a converged image?

`ME_NITER`: the number of iterations.

`ME_MAX, ME_MIN`: the maximum and minimum values of the hidden image (not necessarily equal to those of the output image).

TIMINGS

Two FFTs is needed in one iteration. In the case where the image size is a power of two, the other math operations is equivalent roughly to 0.5 FFT in CPU time; otherwise the CPU time for the former is negligible. Thus, the CPU time is estimated to be

CPU time for an FFT $\times$ (2~2.5) $\times$ the number of iterations.

As an example, for processing a 256$\times$256 image on RA (Sun 4/370) at STScI, the CPU time for an FFT is about 2 seconds, so 30 iterations requires approximately 150 seconds or 2.5 minutes. On SCIVAX (VAX–8800), the required CPU time is roughly 330 seconds or 5.5 minutes (4.4 seconds for a 256$\times$256 FFT).

EXAMPLES

1. Perform deconvolution on the degraded image `mdeg` with p.s.f. `mpsf`, using a flat prior image and a generated ICF with $sigma = 0.5, 1.0$, outputting a hidden image named `mhidden`, having only uniform Gaussian noise of variance 4.0, with *tp* and *nppsf* automatically calculated, displaying all messages. (*sigma* can be changed only by `>epar irme0b`.)

```
me>irme0b mdeg mpsf "" "" mhidden 4.0 0 0 0 0 hidden=yes message=3
```

2. Use the ME image from 1. as the prior image to do deconvolution once again, outputting an apparent image.

```
me>irme0b mdeg mpsf mhidden "" mapparent 4.0 0 0 0 0 message=3
```

3. Perform deconvolution on a section of the degraded image `r136` with p.s.f. `psfr136`,

using a flat prior image and no ICF, outputting an image named `mer136`, using the default *vc0-3* for WF/PC of the HST, with *tp* and *nppsf* automatically calculated, displaying all messages.

me>`irme0b r136[11:80,41:110] psfr136 "" "" mer136 3 .1333 0 0 0 message=3`

IMPORTANT NOTICE

(1) This task works with input images of arbitrary size. However, this does not mean that you need not use your brain in choosing images' sizes. The general guideline was presented before. Now from the point of view of FFT speed, the array size determined by the maximum of the degraded image and p.s.f. sizes should be a power of two if possible. This is may well be impractical. Then, it must be avoided to use an array size (usually equal to the input degraded image' size) having a large prime number factor. As a good example, on RA (Sun 4/370) a 128×128 FFT takes 0.42 second, while a 127×127 FFT takes 6.9 seconds. (A 512×512 FFT takes 8.7 seconds.)

To assist the user in choosing the array size, two tasks are provided: `foolfactor` used to factorize a natural number, and `irfftesb` used to determine the FFT speed. They are described later.

(2) Because quite a few parameters are needed for running the task, some of which are hidden, so the advice is to edit the parameters before running the task. Then enter only `irme0b` on the command line and respond to each prompt carefully. Keep your fingers crossed. Wait patiently to the end, or do something else. There is no way to intervene the execution of the task once it begins to run, except using CTRL/C to zap it. If you are confident enough, you may run the task as a background job.

The task may be run again using as a prior image the previous output image for which *hidden* = `yes`, or $sigma = 0, 0$ and $fwhm = 0, 0$. Perhaps some parameters need to be changed, and the prior image's background should be raised a bit. In this way the restored image can be "refined" step by step. To get the apparent image, set *hidden* = `no` in the last run, or convolve the output hidden image with a Gaussian function (ICF) by running the IRAF task `gauss` or the task `imconvb` (with the p.s.f. file generated by running the task `immakeb`). A script task can be written to accomplish this procedure, with intermediate images displayed.

BUGS

May be a lot. To be found out and fixed.

SEE ALSO

`irme0c, ..., irme1, ...`, which may be available in x months.

2. `immakeb`

USAGE

immakeb outim n1 n2 pos1 pos2 amp sigma1 sigma2 fwhm1 fwhm2 rms

PARAMETERS

outim
: Output image name.

n1, n2
: Image sizes in the first (x) and second (y) dimensions.

pos1, pos2
: Gaussian function's central positions in the first and second dimensions.

amp
: Amplitude of the Gaussian function.

sigma1, sigma2
: Gaussian function's sigmas in the first and second dimensions.

fwhm1, fwhm2
: Gaussian function's full widths at half maximum in the first and second dimensions.

rms
: Rms value of Gaussian noise.

seed=347951
: Seed for generating the noise.

DESCRIPTION

This is a task for generating a 2-D image having an object of Gaussian type. Zero–mean Gaussian white noise may be added. Its usage is quite simple. However, a brief explanation is still needed.

If *sigma1* is zero, then *fwhm1* will be used and *sigma1* will be ignored. Otherwise *sigma1* will be used and *fwhm1* will be ignored. They are related by $sigma1 = fwhm1/\sqrt{8\ln 2}$. Enter a small value, say 1.0E–4, for *sigma1* to virtually set it to zero, but *sigma1* is used. This rule is also applicable to *sigma2* and *fwhm2*.

This task generating an elliptic Gaussian function, together with other IRAF tasks, may be used to make images having simple patterns.

TIMINGS

EXAMPLES

1. Generate a noise–free point spread function (p.s.f.) of Gaussian type, which is centrally located and normalized so that its maximum is one, and has FWHMs equal to 2 in the both dimensions.

me>**immakeb outim 128 128 65 65 1 0 0 2 2 0**

2. Generate a delta function at the center.

me>`immakeb outim 128 128 65 65 1 0 0 0 0 0`

3. Generate a noise–free function with zero values everywhere except along a line segment (1-D Gaussian).

me>`immakeb outim 128 128 65 65 1 3 0 0 0 0`

4. Generate a noise-only image with rms=2.

me>`immakeb outim 128 128 65 65 0 1 1 1 1 2`

BUGS

SEE ALSO

3. `imconvb`

USAGE

imconvb inimage psf outimage

PARAMETERS

inimage
: Input image to be convolved.

psf
: Input p.s.f. for convolution.

outimage
: Output image after convolution.

center=yes
: Move the p.s.f. peak to the DFT center. Set center=no for no move.

norm="volume"
: Normalize the p.s.f.'s volume to one. Set norm="peak" to normalize the peak value to one, or set norm="no" for no any normalization.

DESCRIPTION

This is a task particularly for convolving an image with a p.s.f.. By default, the p.s.f.'s peak will be moved to the DFT center, and its volume will be normalized to one. So the convolution will be correct and the image's total power will be conserved after convolution. For the general use of this task, set *center* = `no` and *norm* = "`no`".

Note that the input image and p.s.f. may have arbitrary sizes. The output image will have the same size as the input image. Beware of the aliasing problem because convolution is accomplished by the FFT technique.

TIMINGS

This task will take the CPU time for 3 FFTs.

EXAMPLES

1. Convolve two images without centering and normalizing the second one.

me>`imconvb image1 image2 outimage center=no norm="no"`

BUGS

SEE ALSO

4. `irfftesb`

USAGE

irfftesb outm

PARAMETERS

outm
: The common part of the output data file names.

n1=128, n2=128
: Array size in each dimension.

DESCRIPTION

This is a task for testing the 2-D FFT procedure `FFT_NCAR`. The test data are generated by a separate procedure `FFTDATA` (a Gaussian function normally).

The output data file name before FFT is outm//"r", after a forward FFT is outm//"c", and after double (forward and then inverse) FFT is outm//"d". The output data after a forward FFT are their magnitudes.

The FFT size is arbitrary, not necessarily a power of two. The FFT will be fastest if the size is a power of two, and will be slow if it has a large prime factor. Use the task `foolfactor` to see the factors.

TIMINGS

CPU time in seconds.

n1xn2	Sun 4/370	VAX-8800	Factors	n1xn2	Sun 4/370	VAX-8800	Factors
53x63	0.14	0.30	63=3x3x7	511x511	40	74	511=7x73
63x65	0.16	0.35	65=5x13	511x513	26	50	513=3x3x3x19
64x64	0.10	0.25		512x512	8.7	20	
65x65	0.16	0.33		513x513	13	28	
127x127	6.9	14	127=127	800x800	23	62	800= 2x2x2x2x2x5x5
127x129	4.3	8.6	129=3x43	1023x1023	83		1023=3x11x31
128x128	0.42	1.0		1024x1024	37		
129x129	1.7	3.4					
255x255	3.2	7.1	255=3x5x17				
255x257	29	54	257=257				
256x256	1.9	4.4					
257x257	55	110					

CPU times will be displayed when the task is running. Some results are presented in the table.

EXAMPLES

1. Generate a 127×129 real array, FFT and then IFFT it.

me>`irfftesb fftfile n1=127 n2=129`

BUGS

SEE ALSO

APPENDIX

FFT procedures

There are 4 interface procedures for using subroutines in the NCARFFT library for 2-D FFT.

(1) fft_b_ma (pt_carray, n1, n2, work)
(2) fft_b_mf (pt_carray, work)
(3) ffft_b (rarray, carray, n1, n2, work)
(4) ifft_b (carray, rarray, n1, n2, work)

To use them, link to libpkg.a or libpkg.olb.

In the NCARFFT library, FFT is performed for an array of arbitrary size, complex to complex without transposition and scaling.

`fft_b_ma` is for initialization. `fft_b_mf` is for the dynamic memory deallocation. `ffft_b` is for the forward FFT, from real to complex without scaling, while `ifft_b` is for the inverse FFT, from complex to real with scaling.

A program segment for calling FFT and then IFFT is as follows.

```
......
int     n1, n2          # Array size
int     narr            # Total number of points in the array
pointer pt_rarray       # Pointer of the real array
pointer pt_carray       # Pointer of the complex array
pointer work            # Pointer of fft working space structure

begin
        # Memory allocation for the real array
        narr = n1 * n2
        call malloc (pt_r, narr, TY_REAL)

        # FFT initialization, including dynamic memory allocation for the
        # complex array holding input/output for FFT, calculate
        # trigonometrical function tables, and allocate some working space
        # for FFT.
        call fft_b_ma (pt_carray, n1, n2, work)

        # Calling forward FFT
        call ffft_b (Memr[pt_rarray], Memx[pt_carray], n1, n2, work)

        # Calling inverse FFT
        call ifft_b (Memx[pt_carray], Memr[pt_rarray], n1, n2, work)

        # Finish up
        call fft_b_mf (pt_carray, work)
        call mfree (pt_rarray)
        ......
end
```

5. foolfactor

USAGE

foolfactor number

PARAMETERS

number

A natural number to be factorized.

DESCRIPTION

This is a task written in a foolish way to factorize a natural number no greater than 1223.

TIMINGS

EXAMPLES

1. Factorize 272.

me>`foolfactor 272`

BUGS

The maximum number is 1223, limited by the prime number list in the program.

SEE ALSO

stsdas.fourier.factor