IMCOM: IMage COMbination
Documentation for Version 0.2 (*alpha*)

Barnaby Rowe (2011)

1 Introduction

This is some (very) light documentation describing how to use this early, prototype version (v0.2) of the IMCOM image combination software described in Rowe, Hirata & Rhodes (2011). For theoretical details and discussion of the precise implementation we follow, please see that paper: the purpose of this documentation is to make it easier for you to get IMCOM running.

2 Compiling IMCOM v0.2

IMCOM is written in Fortran 95, and is well-tested using GNU Fortran (`gfortran`) which is available as part of the GNU Compiler Collection (GCC) at
`http://gcc.gnu.org/`
Version 4.3 or later of GCC allows the use of OpenMP-based multi-threading to speed up many of the IMCOM calculations on multi-cored systems. The OpenMP directives can be ignored for earlier versions of GCC by omitting the OpenMP compiler flag `-fopenmp` in the `Makefile`. This `Makefile` for the IMCOM package can be found in the directory `./src/` (all paths given as navigated from the directory containing this document).

The IMCOM package has certain additional library dependencies, all of which are open source and available free of charge. These are:

- The BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra Pack) libraries, available and documented at
  `http://www.netlib.org/blas/` and
  `http://www.netlib.org/lapack/`.
  Both libraries depend for speed on the implementation of the BLAS. Quoting from the LAPACK site: *Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. For details of known vendor- or ISV-provided BLAS, consult the BLAS FAQ. Alternatively, the user can download ATLAS to automatically generate an optimized BLAS library for the architecture. A Fortran 77 reference implementation of the BLAS is available from netlib; however, its use is discouraged as it will not perform as well as a specifically tuned implementation.*
- The CFITSIO/FITSIO library (Version 3.25 is tested as compatible with IMCOM) of C and Fortran subroutines for reading and writing data files in FITS (Flexible Image Transport System) data format, available to download free from NASA at
  `http://heasarc.gsfc.nasa.gov/fitsio/fitsio.html`.
- The FFTW library of C subroutines with Fortran wrappers for computing the discrete Fourier transform (DFT) in one or more dimensions (Version 3.2.2 is tested as compatible with IMCOM), available and documented at
  `http://www.fftw.org/`

These libraries should be installed on your system and the paths to these libraries and their include files should be changed in the `Makefile` to match their respective locations in your system.

Once this is done, typing `make` from the terminal in this `./src/` directory should compile the code. This will not take long as IMCOM is not a large piece of software.

3 Running IMCOM v0.2 from the command line

The IMCOM package is called using the minimum-possible set of command line arguments as follows:

```
./imcom <config_file> <U/S> <U/S_max> <U/S_tol>
```

where

- `<config_file>` gives the path and filename of the important IMCOM config file, described in detail in Section 4 below.
- `<U/S>` is either one of the single character strings `U` or `S` – other values should produce an error message. This command line input switches IMCOM behaviour between solving κ_α while requiring $U_\alpha^{\rm max} - \Delta U_\alpha^{\rm max} \leq U_\alpha < U_\alpha^{\rm max}$ (for input `U`) or requiring $\Sigma_{\alpha\alpha}^{\rm max} - \Delta\Sigma_{\alpha\alpha}^{\rm max} \leq \Sigma_{\alpha\alpha} < \Sigma_{\alpha\alpha}^{\rm max}$ (for input `S`; see Rowe et al. 2011 for details of what these relations mean).
- `<U/S_max>` should be a double precision floating point number, typically given in exponential, e.g. `1.d-8`), which specifies the numerical value of either $U_\alpha^{\rm max}$ (in units of C_α) or $\Sigma_{\alpha\alpha}^{\rm max}$ as specified by `<U/S>`.
- `<U/S_tol>` should be another double precision floating point number, typically given in exponential format, e.g. `1.d-12`, which specifies the numerical value of either $\Delta U_\alpha^{\rm max}$ (in units of C_α) or $\Delta\Sigma_{\alpha\alpha}^{\rm max}$ as specified by `<U/S>`.

We now give a simple example of this usage. The user has a set of images which are described in the file `config_example`, along with an output PSF and sampling locations. This user wishes to attempt to maintain $9.99 \times 10^{-4} \leq \Sigma_{\alpha\alpha} < 1 \times 10^{-3}$ in their final output image, and therefore calls IMCOM using the command:

```
./imcom config_example S 1.d-3 1.d-6
```

This is an exercise you yourself can try using the images and `config_example` configuration file available in the directory `./example/`.

3.1 Optional parameters

If you call IMCOM with too few (or far too many) command line arguments, something like the following message should be printed:

```
IMCOM: IMage COMbination v0.2 (B. Rowe & C. Hirata 2010-2011)
IMCOM: [usage]
./imcom <config_file> <U/S> <U/S_max> <U/S_tol> [...<forceT> <forceSys>
<saturation>]

<config_file> [string] :  Global config file containing image locations etc.
<U/S> [string] :  U/u or S/s to specify minimization of U or S in output image
<U/S_max> [dbl] :  Required maximum U or S
```

```
<U/S_tol> [dbl] :  Absolute tolerance on U or S for interval bisection
...<forceT> [int] :  1 = force build T matrix , 0 = not
...<forceSys> [int] :  1 = force build system matrices A, B etc., 0 = not
...<saturation> [dbl] :  Image saturation or bad pixel value
```

These latter three optional input parameters, which must come in that order, fulfill the following purposes:

- `<forceT>` should be set to integer value `1` to force IMCOM to recalculate the $T_{\alpha i}$ matrix rather than read it from the filename specified in the configuration files (see Section 4 below). This is important as the default behaviour is to read in $T_{\alpha i}$ if it exists, but if other parameters (e.g. $U_\alpha^{\rm max}$) have changed since the previous call then the output image will be created using this old $T_{\alpha i}$ matrix unless specifed otherwise using `<forceT>` = 1.
- `<forceSys>` is similar to `<forceT>` and should be set to integer value `1` to force IMCOM to recalculate the $A_{\alpha ij}$, $B_{\alpha i}$ etc. system matrices (including the eigendecomposition and projection matrices) rather than read them in from the filenames specified in the configuration files (see Section 4 below).
- `<saturation>` is means of specifying saturated, cosmic ray-corrupted or otherwise bad pixels in the input images. Input pixels for which $I_i \geq$ `<saturation>` are then explicitly removed from I_i and $\mathbf{r}_i$, so that this loss of information will be faithfully reflected in the output U_α and $\Sigma_{\alpha\alpha}$ maps.

Following on from the previous example, if the user wished to re-run IMCOM as before but rebuild all the matrices (perhaps following some change in the input conditions), they would then use the command:

```
./imcom config_example S 1.d-3 1.d-6 1 1
```

One further point worth mentioning is that if the $T_{\alpha i}$ matrix specified in the configuration files already exists at run time, and if `<forceT>` or `<forceSys>` are not set, then the IMCOM code will simply read in the pre-existing $T_{\alpha i}$ and use it to calculate H_α from I_i. No other functions will be performed (e.g. calculating $A_{\alpha ij}$, U_α etc.). This is done to allow the same transformation to be applied to many small, nearby patches with maximum efficiency for those cases where $T_{\alpha i}$ is unchanged.

4 The IMCOM v0.2 config files

The IMCOM config file contains filenames and paths to all the input images and essential information about what these contain, as well as defining filenames for the outputs of the IMCOM algorithm. The best way to elucidate the configuration file is using the example given in the `./example/userxy1/` directory:

```
PSFXSCALE   0.1
PSFYSCALE   0.1
NEXP        4
USERXY      1
INCONFIG1   config_example.input1
INCONFIG2   config_example.input2
INCONFIG3   config_example.input3
INCONFIG4   config_example.input4
OUTCONFIG   config_example.output
AFILE       A.example.fits
BFILE       B.example.fits
QFILE       Q.example.fits
```

```
LFILE       L.example.fits
PFILE       P.example.fits
```

Order and spacing matters for the simple interface used by IMCOM, although IDL scripts are provided in the directory `./idl/` to write this file automatically supplying the relevant parameters as keywords (see Section 4.3). We now give a brief description of these parameters (see also Rowe et al. 2011):

- `PSFXSCALE` & `PSFYSCALE` give the physical dimensions along x and y of the pixels in the input PSF images $G_i(\mathbf{r})$ and $\Gamma(\mathbf{r})$.
- `NEXP` is the number of exposures in the input images I_i. In this implementation each of these exposures is supplied as a separate `FITS` image.
- `USERXY` switches between the two supported input formats for the user to supply $\mathbf{r}_i$ and $\mathbf{R}_\alpha$. This is discussed in detail in Sections 4.1 & 4.2.
- `INCONFIG`*N* gives the filename of the configuration file for the *N*th exposure where *N* $= 1, \ldots,$ `NEXP`. See Sections 4.1 & 4.2.
- `OUTCONFIG` gives the filename of the configuration file for the output properties, see Sections 4.1 & 4.2.
- `AFILE`, `BFILE`, `QFILE`, `LFILE` & `PFILE` give the filename of the `FITS` files used to store the $A_{\alpha ij}$, $B_{\alpha i}$, Q_{ij}, λ_i & $P_{\alpha i}$ system matrix/vector objects, respectively. If the file does not exist, these objects will be calculated and saved to the corresponding *X*`FILE`; if the file does exist it will be read in and used in subsequent calculations unless `forceSys` = 1 is specified at the command line.

So much for the primary configuration file. We now go on to describe its offshoots, the input and output configuration files, which differ in their content depending on the value of `USERXY`.

4.1 Input & output config files in the general case USERXY = 1

The most general and powerful way for the user to supply the input images and pixel locations is by giving `FITS` images of x_i and y_i corresponding to the input images for I_i, where $\mathbf{r}_i = (x_i, y_i)$. This choice is specified by setting `USERXY=1` in the primary configuration file.

Each of the `NEXP` input configuration files then takes a format illustrated by the following example `config_example.input1` (i.e. the input config file corresponding to the first exposure) from the `./example/userxy1/` directory:

```
PSFFILE     psf.example.fits
GIMFILE     test1.example.fits
ROTANGDEG   0.0000000
NOISE       1.0000000
GIMXFILE    x1.example.fits
GIMYFILE    y1.example.fits
```

- `PSFFILE` is the filename for the image of the PSF $G_i(\mathbf{r})$, with x and y pixel dimensions given as `PSFXSCALE` and `PSFYSCALE` in the primary config file. Note therefore that these units are fixed and may not vary from exposure to exposure.
- `GIMFILE` is the 'galaxy' image file for this exposure, making up part of I_i.

- ROTANGDEG is the angle at which the pixel grid for this input exposure is rotated, in degrees, relative to the output pixel grid. This needs to be known as it leads to a relative rotation of the PSF. As in USERXY = 1 the user specifies x_i and y_i as image files (see directly below) this would allow, in principle, this value to be determined from the input data without it needing to be specified. However, in this current version of IMCOM the author has been too lazy to do this! Positive angles denote an anti-clockwise rotation.
- NOISE gives *all* the diagonal values of the assumed stationary, diagonal input noise covariance N_{ij} relating to this exposure. Values of this parameter can vary between exposures, but not yet within them.
- GIMXFILE & GIMYFILE give the filenames of the FITS files where the input x_i and y_i have been stored by the user. These must be in the same physical units as PSFXSCALE and PSFYSCALE.

The output config file similarly contains the location of the files needed to specify $\mathbf{R}_\alpha$, as well as the desired storage locations of the output image and objective functions U_α and $\Sigma_{\alpha\alpha}$. Yet again, its format can be illustrated using one of files in the `./example/userxy1/` directory, `config_example.output`:

```
GAMFILE     psf.example.fits
OUTFILE     H.example.fits
KFILE       K.example.fits
TFILE       T.example.fits
SFILE       S.example.fits
UFILE       U.example.fits
OUTXFILE    xout.example.fits
OUTYFILE    yout.example.fits
```

- GAMFILE is the filename for the image of the desired output PSF $\Gamma(\mathbf{r})$, with x and y pixel dimensions given as PSFXSCALE and PSFYSCALE in the primary config file.
- OUTFILE is the filename for the output image H_α.
- KFILE is the filename for the output κ_α solution map.
- TFILE is the filename for the output $T_{\alpha i}$ transformation matrix. If the file does not exist, $T_{\alpha i}$ will be built and saved to the corresponding TFILE; if the file does exist it will be read in and used to generate H_α, U_α and $\Sigma_{\alpha\alpha}$ unless forceT = 1 is specified at the command line.
- SFILE is the filename for the output noise variance $\Sigma_{\alpha\alpha}$.
- UFILE is the filename for the output leakage objective U_α.
- OUTXFILE & OUTYFILE are the filenames for the FITS files where the desired output sampling locations $(X_\alpha, Y_\alpha) = \mathbf{R}_\alpha$ have been stored by the user. These must be in the same physical units as PSFXSCALE and PSFYSCALE.

4.2 Input & output config files in the case USERXY = 0

The values of $\mathbf{r}_i$ and $\mathbf{R}_\alpha$ can also be specified in a less general manner by setting USERXY = 0. This calling procedure is illustrated by the example configuration files in the `./example/userxy0/` directory. An example input file is as follows:

```
PSFFILE     psf.example.fits
GIMFILE     test1.example.fits
ROTANGDEG   0.0000000
```

```
NOISE       1.0000000
GIMXSCALE   0.4000000
GIMYSCALE   0.4000000
DITHER      0.0000000 0.0000000
```
The first four parameters are used exactly as before in Section 4.2. The three different parameters are:

- `GIMXSCALE` & `GIMYSCALE` specify the (constant) x and y dimensions of each pixel in the `GIMFILE` image. These must be in the same physical units as `PSFXSCALE` and `PSFYSCALE`.
- `DITHER` gives the position, in the same physical units as `PSFXSCALE` and `PSFYSCALE`, of the *exact center of the lower-leftmost pixel.*

In the same directory can be found an example output configuration file:

```
GAMFILE     psf.example.fits
OUTFILE     H.example.fits
KFILE       K.example.fits
TFILE       T.example.fits
SFILE       S.example.fits
UFILE       U.example.fits
OUTXSCALE   0.200000000
OUTYSCALE   0.200000000
OUTPOS      0.0000000 0.0000000
NOUT        81 81
```

The first six parameters are those described in Section 4.2. The remaining parameters are

- `OUTXSCALE` & `OUTYSCALE` specify the desired, constant x and y dimensions of each pixel in the output `HFILE` image. These must be in the same physical units as `PSFXSCALE` and `PSFYSCALE`.
- `OUTPOS` is analagous to `DITHER` and gives the desired position, in the same physical units as `PSFXSCALE` and `PSFYSCALE`, of the *exact center of the lower-leftmost pixel* of H_α.
- `NOUT` supplies as two arguments the overall x and y dimensions of the output H_α in numbers of pixels, 81 pixels $\times$ 81 pixels in the example above.

This concludes the description of the somewhat lengthy configuration file interface for IMCOM v0.2.

4.3 The IDL `imcom_write_config.pro` script

To help alleviate the process of creating these long configuration files, I have created simple IDL scripts to generate them automatically with correct spacing. These scripts can be found in the `./idl/` directory.

The main script is called `imcom_write_config.pro` and should be called from within your program unit as described in the example IDL script `example_write_config.pro` which can also be found in the `./example/` directory. This script generates the config files shown above.

There are plans for Python scripts to perform the same task, but as yet these are not written. Since IMCOM communicates solely via ASCII text and `FITS` images it should be simple to script in a variety of high-level languages.

5 The example files

The example files that have been described in the previous Sections demonstrate a very simple 2×2 dither pattern, with an output at double the resolution of the input. The input PSF is an Airy disc with a first minimum at around 1.22 in the physical units chosen convolved with the input pixel resonse (assumed to be boxcar). There is also an 'ideal' output, made using unfair knowledge of the sky profile, to allow you to compare with H_α. This is a rather facile case, but should produce predictable results: good luck!