Astronomical Data Analysis Software and Systems VI
ASP Conference Series, Vol. 125, 1997
Gareth Hunt and H. E. Payne, eds.

Difmap: An Interactive Program for Synthesis Imaging

M. C. Shepherd

Owens Valley Radio Observatory, California Institute of Technology, Pasadena, CA 91125

Abstract. Difmap is a stand-alone program used by the radio-astronomy community to produce images from radio interferometers. It reads and writes the standard UV FITS file format produced by packages such as AIPS, and provides convenient ways to inspect, edit, and self-calibrate visibility data while incrementally building up a model of the sky. It has proven to be a popular alternative to larger packages that do much more—so part of this paper will focus on why this is so, and on whether the lessons learned could benefit future projects. Some time will also be spent describing the general structure of Difmap, particularly how being contained within a single-process is exploited to achieve speed, portability, and a degree of interactivity that is hard to match in larger systems.

1. Introduction

In June of 1992 I was asked to write a new program for the Caltech VLBI package. The new program would implement an iterative mapping technique called "difference mapping," but would rely on existing programs for data inspection, data editing, and image display. I didn't want users to have to continually switch between programs, so I decided instead to write an integrated difference mapping environment in which all of the functionality of the Caltech VLBI package would be incorporated within a single program. Three months later I had written the minimum quorum of commands needed to implement difference mapping, packaged under a scripting interface that I had written previously. The resulting program was called Difmap.

Difmap quickly gained users from the Caltech VLBI group, where it was initially used to augment the Caltech VLBI package. Meanwhile Difmap continued to subsume the functionality of the Caltech VLBI package, and in June of 1993—one year after the project started—I made the first external release of Difmap. Difmap quickly spread throughout the VLBI community, and later, after Difmap had been upgraded to support multi-dimensional AIPS UV-FITS files, it became a popular alternative to more general packages, such as AIPS.

No significant development has taken place since the release of the FITS version of Difmap in April 1995, owing to a reduction in NSF funding. However, Difmap has retained a strong following and has required very little maintenance.

2. Interferometry in a Nutshell

What follows is a brief introduction to radio interferometry. Many of the terms that are introduced here will be used throughout the rest of the text.

The essential steps needed to construct an image from signals received by a radio interferometer are:

1. Point an array of telescopes at a radio source.
2. Measure the complex wavefronts that arrive at each telescope with radio receivers.
3. Insert artificial delays in the signal paths of each telescope so that the telescopes all appear to be at the same effective distance from the source.
4. Interfere signals from pairs of telescopes and record the resulting complex fringe visibilities as a function of time.
5. Note that the visibilities from a given pair of telescopes sample spatial frequencies on the sky in proportion to the projected separation of the telescopes. Also note that this separation changes as the Earth turns, so each pair of telescopes samples a locus of spatial frequencies as a function of time. This is called *Earth rotation synthesis*.
6. Use the observations of a bright source to calibrate the visibilities.
7. Interpolate the visibilities onto a 2-D spatial-frequency plane called the *UV plane*.
8. Calculate the FFT of the UV plane. The resulting image is called a *dirty map*, and constitutes an image of the source corrupted by the point-spread function of the interferometer.
9. Form a model of the source through CLEAN deconvolution of the dirty map, or by fitting a model to the visibilities.
10. Convolve the model by an elliptical Gaussian approximation of the point spread function of the interferometer, and add the result to the un-modeled residual noise in the dirty map. This is called a *clean map*.
11. Publish the map.

3. What Does Difmap Do?

3.1. Data Inspection and Editing

A significant fraction of Difmap is devoted to providing users with convenient tools for interactive data inspection and editing. A family of graphical commands displays observed and model visibilities from a variety of perspectives, and allows the user to flag or unflag visibilities directly with the mouse. Simple key-bindings are combined with mouse positioning to quickly navigate through telescopes, baselines, bands, and sub-arrays, to edit visibilities collectively or individually, to zoom in on parts of the data, and to perform many other command-specific operations. These facilities are important because:

- They allow users quickly to identify and excise corrupt data.
- They provide a direct comparison between the model and the data.
- They help to familiarize users with the peculiarities of their datasets.
- They provide beginners with visualization tools to learn how given visibility profiles produce given maps.
- They can be used to inspect dynamically changes made to the visibilities and the model during processing.

3.2. Difference Mapping

Difmap was named after a mapping technique called *Difference Mapping.* This technique was originally pioneered in the Olaf package written at the Nuffield Radio Astronomy Laboratories.

When a model of a source is subtracted from a dirty map, what remains is known as a *difference* or *residual* map. This is a key component in Difference Mapping.

The source model that is subtracted from the dirty map is built up in an iterative fashion, usually through CLEAN deconvolution of successive versions of the residual map. Alternatively a model can be fitted to the observed visibilities.

When CLEAN is used to build up the source model, each iteration of CLEAN not only adds a delta-function to the model, but also subtracts that delta-function and its PSF from the residual map. Thus CLEAN automatically keeps the residual map up to date with the changes that it makes to the model.

Conversely, when model fitting is used to build up the model, or when a user edits either the visibilities or the model between successive iterations of CLEAN, then the residual map becomes out of date. When this happens, Difmap automatically re-calculates the residual map by taking the 2-D FFT of the *difference* between the observed and the model visibilities.

At this point, if visibilities were edited by the user, then the current model may retain minor features that were deconvolved from those visibilities. With the offending visibilities removed from the data, these artifacts only remain in the model, so when the model is subtracted from the data, the revised residual map contains an inverted version of the artifacts. Subsequent iterations of CLEAN thus erase them from the model. This typically involves both positive and negative CLEAN components.

The ability to transparently continue deconvolution after modifying the model or the observed visibilities highlights a fundamental difference between difference mapping and traditional techniques. The traditional approach would require the user to restart deconvolution from scratch, whereas the difference mapping approach encourages the user to edit or re-calibrate the data on-the-fly as ever weaker artifacts appear in successive versions of the residual map. It also enables users to incrementally add clean windows. This is especially useful when features that were initially obscured by the point spread functions of brighter features, are revealed in the residual map.

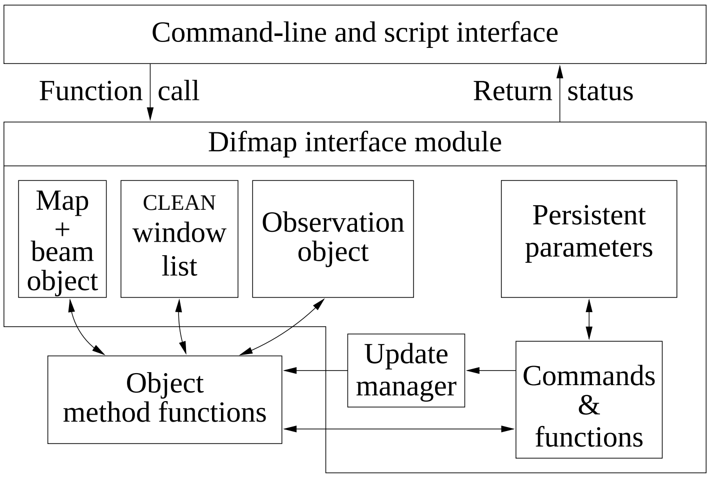

Figure 1. An overview of Difmap's architecture.

4. Difmap Architecture

An overview of Difmap's internal architecture is depicted in Figure 1. At the top is the command-line scripting interface with which users interact. This performs expression evaluation, parameter checking, etc., then delegates work to functions in the Difmap interface module. To eliminate dependencies on the user interface, Difmap functions never call back to the scripting interface.

Functions within the Difmap interface module maintain a number of objects that collectively contain the state of Difmap. Operations on those objects are delegated to method functions.

4.1. The Observation Object

The most significant object in Difmap is the Observation object. This encapsulates the observed visibilities (contained mainly in paging files), the observation parameters, model components and their associated visibilities, self-calibration corrections, and the pseudo-continuum channel that is the unit of in-core processing in Difmap.

The constructor for an `Observation` object reads a UV FITS file and records all but its visibilities in memory. The visibilities are recorded in a scratch file. The `select` command is then used to select or change which polarization and range of spectral-line channels to process next. This results in the construction of one pseudo-continuum channel per band. The corresponding visibilities are stored in a paging file, ordered such that Difmap processing commands can sequentially page into memory one band at a time, for processing.

Modifications to the visibilities, such as self-calibration corrections, phase-center shifts, UV coordinate frequency scaling, and weight scales are recorded in memory, then applied on-the-fly whenever a new band is paged into memory. They are also applied on-the-fly if an observation is written to disk as a new UV FITS file. The only modifications that are made to the scratch files are editing operations. Even these are temporarily buffered in memory to reduce file I/O to a minimum. As a result, un-doing most visibility corrections is simply a matter of clearing them from memory.

Model visibilities for each band are also recorded in a paging file. The associated image-plane model components are stored as lists in memory, partitioned into tentative and established models. The tentative model records components that have not yet been established as model visibilities, and is automatically established in the UV plane whenever a method function needs the model visibilities, or when the `keep` command is called. Until then, it can be discarded with the `clrmod` command. Thus it is possible to call `clean`, look at the resulting residual map, and then decide to discard the latest additions to the model. Note that to reduce the number of components that have to be transformed to the UV plane, identically positioned delta-components are combined into single components.

The model may be further split into normal and continuum models. Continuum subtraction in Difmap exploits difference mapping techniques to perform the model subtraction on-the-fly, without changing the recorded observed visibilities. This is important, not only because it gives the user the ability to modify or discard the continuum model, but also because self-calibration can then continue to operate on the original observed visibilities. The only practical difference in the treatment of the continuum and the normal models is that the continuum model is not added back to the residual map when the clean map is constructed.

4.2. The Update Manager

The update manager acts a bit like the UNIX `make` facility. It has the responsibility of keeping the objects of the Difmap interface module up-to-date with each other. It does this in two stages:

1. When a user command modifies an object in such a way as to invalidate the contents of another object, then the latter object is marked as out-of-date.
2. Later, when a user command requires a given object, it asks the update manager for it. If the object has been marked as invalid, then the update manager causes the object to be updated before passing it back to the requesting command.

For example, when the `shift` command is called to move the phase center of the observed and model visibilities, the map object is marked as out of date by the `shift` command. If the `clean` command is then invoked, it asks the update manager for the map object, and the update manager interpolates a call to the `invert` command to re-compute the residual map.

The update manager thus manages relationships between otherwise unrelated objects. This frees users from the tedious bookkeeping that traditional packages require, and this makes it safer for users to experiment.

The update manager enforces consistency among its objects by interpolating calls to user commands (hereafter referred to as *command interpolation*). On the rare occasions when the default running parameters of these commands are not appropriate, users can call them explicitly.

While it is easiest to describe the update manager as a single entity, in reality, it is a collection of facilities operating at different levels to implement the rules described above.

5. What Distinguishes Difmap from its Peers?

5.1. Difmap is a Single Program

Whereas most large packages split tasks into separate programs, Difmap implements them as functions within a single program. The advantages of keeping everything within a single process are hard to ignore. In particular:

- The use of memory objects and function calls instead of intermediate files, interprocess communications, and multiple processes: (*i*) reduces overhead dramatically, (*ii*) removes a major source of unportability, (*iii*) eliminates a significant amount of code, and (*iv*) simplifies the user interface.
- Since a lot of state information resides permanently in memory, sanity checking can trivially be extended to cross command boundaries. This makes command-interpolation possible.
- The low overhead of invoking command functions means that: (*i*) minor commands are justifiable on their own, rather than having to be redundantly tacked on to other commands, and (*ii*) redundancy is reduced because commands can trivially delegate work to other commands via function calls.
- Bugs in a single process package often manifest themselves quickly through cumulative effects. This means that most bugs are caught during pre-release tests.

5.2. Dynamic Resource Allocation

Difmap doesn't employ hard-wired limits. All resources are dynamically allocated to take advantage of whatever resources a given computer has available. This includes memory for objects and paging files for visibilities.

5.3. Scripting Facilities

Most packages provide some kind of scripting language, and Difmap is no exception in this respect. However, Difmap exploits its scripting interface in a couple of novel ways.

Log files: When Difmap is started, a log file is created, and everything that the user types, along with the resulting output messages, are recorded in this file. By recording command-lines verbatim and output messages as comments, a log file doubles as a script. This script can then be used to re-play a Difmap session.

Recording log files as scripts is not only useful to users, but really comes into its own when a user encounters a bug. When this happens, the script can be used directly to re-produce the bug.

Save files: The Difmap `save` command saves the state of Difmap by writing a script that contains the user-commands needed to restore the running parameters. In addition, the `save` command saves the visibility data, model components, etc., in files, and adds to the script the commands needed to re-load them.

Saving parameters via scripts eliminates the need for special parameter-file formats.

5.4. Difmap Doesn't Obfuscate the Environment

Traditional packages redundantly implement their own file-systems, internal file formats, printer spooling facilities, and tape I/O facilities. Difmap doesn't. It lets users use the facilities that they are familiar with, and, in the process, simplifies package management and the amount of work that users have to do to get data into the program.

In particular, Difmap directly reads UV FITS files, so that users can start inspecting and processing new observations within seconds of starting the program. In traditional packages, these files would first have to be converted to an internal file format and then written to a package-specific file-system. Difmap doesn't need an efficient internal format because its single process design removes the need to re-read files each time that a new command is invoked.

6. Conclusions

Difmap is a popular alternative to more capable packages such as AIPS because:

- It is tightly focused on a particular processing scheme.
- It is highly interactive.
- It is fast for small to moderate sized data sets.
- UV FITS files can be processed directly without the overhead of conversion to an internal format and without requiring users to learn new file management utilities.
- It is easy to install and manage.
- It encourages learning through experimentation via its use of command-interpolation to correct for missed steps, and the ability to undo certain operations.
- Its small set of simple commands is easier to use and understand than the complex tasks that traditional packages provide.

From a programming standpoint, experience with Difmap has shown that there is no need for packages to partition tasks into separate programs. Object based techniques provide sufficient partitioning on their own. It is even more feasible to do this now than it was when Difmap was written, because:

- The concurrency gained by spawning tasks as separate programs can now be achieved much more efficiently in a single program by using POSIX threads.
- To allow multiple programmers to work on a single program, and to keep the size of the resulting executable small, commands or modules can now be written as dynamically loadable entities, which can then be loaded on-the-fly into the running kernel of the program. At this time, there is no single standard for dynamic loading that works on all machines, but the dynamic loading facilities of popular environments such as Tcl and Perl have shown that this is not a big obstacle.

Acknowledgments. As a graduate student at Jodrell Bank, in England, I used both Jodrell Bank's Olaf package and NRAO's AIPS package for data reduction. The interactive and difference mapping aspects of Olaf, together with the painful lack of the same features in AIPS, inspired me to design Difmap in the manner that I did.

Algorithm development for some of the core commands in Difmap would have been much more difficult if the code of the Caltech VLBI package hadn't been at hand for comparison. Similarly, the algorithm underlying the `selfcal` command was derived from SDE code that Tim Cornwell supplied.

Difmap owes its graphical capabilities to the PGPLOT library written by Timothy Pearson, and exploits the algorithms in Patrick Wallace's SLALIB library for precessing source coordinates.

Most importantly, Difmap would not have been as useful as it came to be were it not for the encouragement and suggestions of Difmap users, both at Caltech and externally. In particular I would like to thank Greg Taylor both for writing the Difmap cookbook, and for some invaluable discussions. I would also like to thank Anthony Readhead and Timothy Pearson for giving me the freedom to write Difmap.