

# IDA Research Topics and Development Plan: An informal white paper.

Matt Sottile

October 3, 2006

## **Abstract**

This white paper discusses the current (October 2006) state of the DDMA Image and Data Analysis (IDA) toolkit. Although IDA is maturing rapidly, there are many areas of work that are both improvements on existing implementations and research areas that explore new features and capabilities. This document enumerates many of the items that have been identified as areas of improvement and research. It is intended to communicate to users the areas to expect improvements (bug fixes, improved functionality). It is also intended to communicate the view of the IDA developers to researchers who will use IDA to implement new and cutting edge capabilities that are currently unavailable in the current IDA framework. This is a living document, and will be updated as IDA matures with issues being addressed and new ones arising.

## **1 Introduction**

This document serves to outline research topics and known areas of improvement to existing features in the IDA toolkit. The items are prioritized with two values: priority to users, and complexity of implementation and research. This is intended to give a sense for what is most important to be done soon, while also attempting to lay out the relative difficulty for each item. See Table 1 for information on the priority labeling.

Following the discussion of research topics and software improvements, constraints are laid out for IDA development under which research and contributed code must obey. Due to the diverse installation platform, usability and ease-of-installation requirements for IDA, these constraints are strict.

## **2 Infrastructure Improvements**

This section covers areas of work that are improvements on existing features that lie in the IDA infrastructure level.

|   | <b>Importance</b>                   |
|---|-------------------------------------|
| 1 | Very important                      |
| 2 | Important                           |
| 3 | Useful                              |
|   | <b>Difficulty</b>                   |
| A | Large effort (1 FTE, 3 months)      |
| B | Significant effort (1 FTE, 1 month) |
| C | Small effort (1 FTE, 0.5 month)     |

Table 1: Table of importance and difficulty estimates used to label projects in this document. Note that these are vague estimates, and should not be taken as final nor accurate.

## 2.1 Data input

Currently, IDA uses a crude data input function (IDAinput) that reads all data files from a given directory that match the desired extension. It has a few flaws.

1. Files are read in lexicographic order, not numerical order.
2. Sets of files are treated atomically by IDA.

### 2.1.1 File ordering

**Rating:** [1C]

This is usually noticed if one has files such as `file1.txt`, `file10.txt`, `file2.txt`. Often this leads to confusion when examining the data and expecting the ordering to be numerical (1,2,10) versus lexical (1,10,2). The user should be able to specify the file naming pattern to IDA input, possibly following the C-style `printf` formatting. For example, one could specify the files above as `file%d.txt`. This would have two benefits.

- Numerical ordering would be followed.
- IDA could automatically extract the numerical value out and use it to identify the data from the file in addition to the filename. This is often useful, as in some cases the numerical value corresponds to a time stamp.

Regular expressions could also be considered, although this is likely a poor choice for expressing these patterns due to the lack of familiarity with the notation to most IDA users.

### 2.1.2 Granularity

**Rating:** [1A]

If a directory contains  $n$  files, IDA input reads all  $n$  into memory, and passes them into the processing pipeline. IDA internally iterates over this set of inputs, passing them one by one into the processing routines (or pairwise for metrics). This has proven to be functional, but has severe limitations.

- Easily exceed the RAM capacity of a machine by forcing bulk data into memory, when RAM would not be exceeded for individual data inputs.
- The addition of 1 file to a directory of  $n$  files that have already been processed requires  $O(n + 1)$  work to be performed if the IDA input file is executed again.
- This limits parallelism potential, or makes parallelism more complex to implement. To execute  $p$  instances of IDA on  $n$  files would require each instance to work on  $\frac{n}{p}$ . Unfortunately, there is no way to tell IDA input to work on a subset of size  $\frac{n}{p}$  without manually breaking up the directories.

The best path forward should be to stop having IDA pass bulk sets of files around, and be able to function on individual files from sets. This will require work at the IDAinput stage, IDA internals, and the database. Due to the tie in with data storage and possible parallel execution, this item should be carefully designed with the other aspects of the system kept in mind, hence the "A" rating for difficulty.

## 2.2 Automatic type conversion and awareness

**Rating:** [2A]

Types were invented for a reason. Loosely typed languages do **not** have the advantage of being good in that they “keep programmers on their toes”. All they do is allow people to be lazy, and periodically prove that even the most careful programmer is still human and makes mistakes occasionally. The problem with loosely typed languages (like Matlab) is that they do not complain until it’s too late if you pass data of the wrong type into a subroutine. The subroutine will happily start executing, and somewhere in the middle it will experience a type error. We have observed cases where errors may not even occur (such as the IDAview interaction with the database cleaning function) when types are compatible (character versus integer) but are cast in unexpected ways.

IDA deals with data that is typed from both a syntactic and semantic point of view. Semantically we have matrices and sequences of images and tensors, all of which are syntactically just arrays. A three dimensional array may be a sequence of 2D images, or a single 3D tensor. Syntactically it doesn’t matter, but semantically it does. In this section, I speak of syntactic type checking. Semantic type checking via metadata is discussed in section 2.3.3.

IDA should force programmers to specify the data type that they expect into their routines, possibly in the .FIT files. It already forces them to state some amount of metadata about the parameters that is necessary for the parameter

study functionality (ie, if a function specifies a scalar input and gets a vector, IDA iterates over the vector passing scalars in to the function in sequence). This should be expanded to capture the different data types that we expect to deal with, such as *double*, *integer*, *boolean* (or logical), and *string* (assume a character is a length 1 string). This would allow IDA to check the types being passed between routines, make the proper type casting operations to convert one type to a valid target, or emit warnings or errors should incompatible types be passed around.

Note that this is of particular importance when passing data across language boundaries. Matlab is content to deal with ambiguous types internally. It is not clear how well it would be handled should data be passed from Matlab into C or Fortran. This is going to be of special interest should we support parallel execution of non-Matlab compute slaves and have to consider how data is marshaled between the client and compute node.

## 2.3 Database

The current database is functional, but has limitations both with respect to performance and user interface.

- Knowledge of input data files.
- Representation problem when we move to a per-file model for processing and storage versus the current whole-directory approach.
- Metadata and experiment setup.
- Input deck storage and recovery.
- IDA code versioning.

### 2.3.1 Knowledge of input data files

**Rating:** [2C]

Currently one has to perform a no-op on input files (such as `makegray`) in order to get them into the database to be viewed by `IDAview`. It would be nice if there was a way to make IDA aware of them without simply pulling them into the database. Data files may change in the filesystem for good reason (user discovers corruption, wrong files, etc...) and this may be difficult to deal with if IDA has them in the database and fails to look at the filesystem.

### 2.3.2 Representation of data sets versus data files

**Rating:** [1A]

When we move to a model where IDA functions on individual data items, IDA will be significantly more suited to parallelism and incremental analysis of

large datasets. IDA currently looks at data simply as output from `IDAinput`, and has no notion of the separation of an abstract data collection from individual data items. This must be dealt with, especially with respect to how users query IDA for results. It would be awkward to force users to iterate over the items in a data set themselves, and similarly awkward (or impossible due to RAM constraints) to force them to retrieve entire sets of data items on queries.

### 2.3.3 Metadata and experimental setup

**Rating:** [2A]

Currently IDA has a very rudimentary knowledge of metadata (ie, semantic type information) that is restricted to the `.FIT` files and is used for the parameter study capability. There is no way currently for the user to attach metadata to input data. Furthermore, there is no way to carry semantic data along with data. For example, it would be nice if one could associate units with data. This could be time units on timestamps (see section 2.1.1), spatial units on pixel dimensions, etc... Users are currently required to carry this metadata in their heads and properly scale parameters to reflect spatial units properly, and recognize quirks in data where wrong parameters can result in output being orders of magnitude off from what was expected, requiring manual correction or interpretation.

The concept of experimental setup is to provide users some mechanism where they can input metadata about a particular data set and analysis pipeline before executing actual analysis routines. There are parameters to routines, such as the `pixel.width` argument to `trigpoly` that could be specified as metadata to the analysis experiment, or attached to the data inputs themselves. The parameters could then be implicitly set by IDA, removing the need of the user to ensure that the proper units are applied to the parameters being handed to the analysis routines.

### 2.3.4 Input deck storage

**Rating:** [2C]

Currently there is no easy way to recover the input deck for a run that is in the database. The version of the input that is stored in the database is somewhat mangled, and there is no function that exists to automatically retrieve the information from the database and reconstruct the `.IDA` file as it originally existed should the user require it. This is important for recovery of accidentally deleted files, or to use things like `diff` to compare inputs and figure out what differs between runs. It would be nice if there was an IDA command that would let a user take two runs, and ask to see the `diff` output of the `.IDA` files to see what changed between them.

### 2.3.5 Versioning

**Rating:** [3C]

Currently IDA relies on a version indicator that accompanies each component, and it is manually updated by the author of the code when the version increases. This is functionally fine, but is prone to problems. Recall that the purpose of the version is to force IDA to re-run analysis on data should the version of an analysis package increase. This is to ensure that users will see bug fixes automatically instead of having to hope that they hear about new versions and re-run their analysis themselves.

- Tie versioning in with source control system version, removing need to have user manually update the version.
- Allow the user to specify the version of code that they want to use in their input deck if they do not want to use the default. This is useful to force IDA to work with the version the user is most comfortable with, not necessarily the bleeding-edge version.

## 2.4 Component composition

**Rating:** [2B]

IDA is essentially a rough, domain-specific component framework for creating data-flow analysis pipelines. The simplistic way it treats pipelines, although good for most users, leaves much to be desired with respect to flexibility in how one creates pipelines through composition of components. Adopting a component model similar to that of the Common Component Architecture group, one can treat components as having input ports and output ports. Composition is the act of gluing components together by matching compatible outputs with inputs where compatibility is defined primarily by ordering.

IDA does not support components with multiple outputs and inputs well at the moment, and requires the users to pay attention to ordering to make things work. Should a new version of a component be released, the user must manually ensure that their ordering of inputs and outputs matches the new version. IDA also introduces a notion of component “type” that overcomes the limitations in the way IDA composition is expressed and implemented. Composition done right will remove the need for component types.

One path to explore with this is to give inputs and outputs of IDA components names, and allow users to say that “output A from component X goes to input M of component Z” (see Figure 1 for an illustration). This would remove the ordering assumptions that currently exist in IDA, and allow components to shift ordering of inputs or outputs and add additional ones without perturbing existing user input decks. It would also provide a way for us to distribute compositions of components that perform tasks built out of individual components that can themselves be treated as a component.

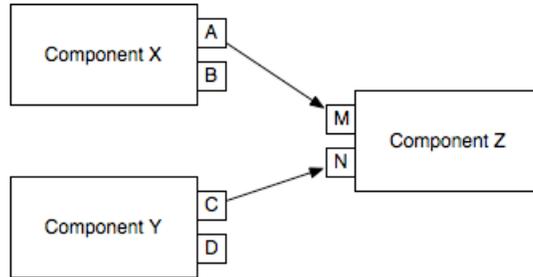


Figure 1: Component composition via named port matching.

## 2.5 Custom code integration

**Rating:** [1C]

Provide a way for users to maintain their own library of personal, custom subroutines that are callable from IDA. This has been discussed in the past, and would likely require users to tell IDA via a configuration file where they put their code so that IDA can include it in the path of valid routines for use. This is quite easy, but just hasn't been done yet. This would be preferable to the current method of users placing code within the IDA directory hierarchy itself.

## 3 Algorithmic Improvements

This section lists improvements that are necessary in the algorithms that IDA contains for performing analysis.

### 3.1 Segmentation improvements

IDA contains many segmentation algorithms, two of which require more work to bring them up to the same level of performance and flexibility as the others. These are Matt's `ccseg` algorithm and Selim's `etseg` method.

#### 3.1.1 `ccseg`

**Rating:** [2C]

The connected-components segmenter (`ccseg`) has recently been returned to functionality after debugging the C code to eliminate some issues with array bounds handling. This algorithm can easily produce a huge number of components from a simple image, requiring a post-processing step to merge those that

are similar together to reach the desired segment count. The `ccseg_refine` algorithm was created to do this, but it has some severe performance problems.

Some level of effort should be spent in improving the performance of the refinement routine. Furthermore, it is often useful to request large segment counts from other segmenters and then later merge some subset of them together to reduce the segment count. The refinement routine should be made more general so as to be applied to more than just `ccseg` output.

### 3.1.2 etseg

**Rating:** [2C]

The Esedoglu-Tsai (`etseg`) segmentation algorithm is promising for many applications, but is currently constrained in its implementation to square images. It should be an easy task to make it more general.

## 3.2 Conservative remappers

**Rating:** [2B]

Techniques such as warping algorithms require that warped data be remapped to a common grid with the data that is being matched in order to compute things like the  $L^2$  norm to evaluate the quality of the warp. Most warp methods that we have examined use interpolation schemes to perform this remapping during the warp computation and refinement stages. Unfortunately, often one desires this remapping operator to obey conservation laws. Consider an image ( $I$ ) composed of pixels that represent densities over spatial regions. If we have a warp operator  $W$  that transforms  $I$ , then we would like to remap  $W(I)$  to the same grid as  $I$  so that we can measure the quality of  $W$  via  $|I - W(I)|$ . We must ensure that the remapped  $W(I)$ ,  $R(W(I))$  has the same total mass as  $I$ :

$$\sum_i \sum_j p_{ij} \in I = \sum_i \sum_j p'_{ij} \in R(W(I)).$$

As we proceed in developing IDA and adding new methods, we are going to be adding warp-based metrics, and possibly others, where such conservative grid remapping will be necessary. Therefore we should ensure that proper remappers are available for use within IDA.

The CORE library from T-7 provides a general purpose C++ class library for conservative grid remapping (CORE = COnservative REmapper). We have a rudimentary wrapper for this library written, and this can be refined into a higher quality such that we can accept it into the IDA distribution.

For performance reasons, we have explored a new algorithm beyond what CORE provides for remapping two dimensional image-based grid data. By restricting the domain of grids that it can accept, we are able to have an algorithm that is tuned to perform well on this narrower set of grids by removing the extra

code necessary for a general purpose algorithm. This algorithm has been prototyped in Matlab and has been shown to work, but due to the non-vectorized nature of the algorithm, we are in the process of porting it to C.

Note that performance is key for a remapper due to the frequent use of remapping algorithms in the core loop of a nested optimization procedure.

### 3.3 Boundary parameterization

**Rating:** [2A]

IDA currently ships with a single functional boundary parameterizer based on a Fourier representation of shape boundaries. Although functional and stable for most inputs, the `trigpoly` routine shows limitations on some data sets. For robustness purposes, IDA should provide multiple boundary parameterizers to better cover the space of shapes that we will be processing. Three ideas for proceeding are (in prioritized order):

1. Bill Allard (Duke) has spent a year or so working on a very promising boundary parameterization scheme based on a heuristic walk of a triangulated image grid. Recently it has appeared to be functional, and a Matlab interface is nearly complete. We should encourage this routine to be included in IDA in the near future.
2. Fix the spline-based parameterizer to remove the dependency on the Matlab Spline toolbox. This could be accomplished by porting the octave-forge code currently used to pure Matlab, or Matlab + C. This should be possible in a reasonable amount of time, and would remove the need to purchase the Spline toolbox for all IDA users.
3. Explore novel interpolation schemes such as ENO (Essentially Non-Oscillatory) interpolation. The RIPS 2006 team at UCLA developed a C++ ENO interpolation routine for their visibility problem, and we may be able to base a more sophisticated boundary parameterization method on their work.

### 3.4 Warping algorithms

**Rating:** [1A]

One of the most promising whole-image metrics on the horizon for our team is the development of warp-based methods. IDA currently includes a few of these, such as the MK-warp and curvature warping. We would like to include both fluid warping, elastic warping, and particle warping also. Implementations of these vary from mature but poor with respect to performance, to rough prototypes that are not suited to IDA inclusion. A significant amount of effort should be expended in the near term to bring these algorithms to a higher, production-grade quality. The problems most in need of attention for warp methods are:

- Performance. Elastic warping has great promise, but current routines that use optimization techniques to derive the warp are very time consuming.
- Lack of conservative remapper in IDA. This is covered above in section 3.2.
- Parameter selection. Little guidance is provided currently for how one should select parameters, or even begin to explore the parameter space for their specific problems. Given the time consuming nature of such parameter studies it is necessary to perform research on warp parameters so guidance can be given to users.

### 3.5 Sequence similarity

One of the most common methods for distilling a metric from our routines is to compare two sequences or vectors using algorithms that give a quantitative measure of their similarity. We have many options available for this:

- Histogram comparison techniques
- $L^p$  norms
- DTW algorithm
- 1D elastic warping

The DTW algorithm is often appropriate when the most natural way to compare two sequences is to measure how difficult it is to transform one into the other. This is essentially a warp operation. In the past we have used a very basic default quantity that comes out of DTW that is not appropriate for all problems. One can construct custom distance functions based on DTW by computing the cost directly from the warp path and the two sequences being compared. This is currently a manual process, and we would like to provide a general purpose method for expressing these custom distance functions without requiring users to manually code up the whole thing and understand the details of how DTW warp paths work.

Another promising approach is the 1D elastic warping algorithm. Like it's two or three dimensional counterpart, the largest limitation of this algorithm currently is it's poor performance.

## 4 Research Topics

This section discusses topics that are more research oriented because they require an investigation into tools and methods that may have been developed already. These are areas where we can push IDA ahead into being more sophisticated than simply a carefully constructed set of Matlab scripts. Building these capabilities can turn IDA into a vehicle on which we can do things that are difficult to do manually or with existing tools. In other words, these are things that even a skilled Matlab hacker will have trouble implementing well themselves.

## 4.1 Integration with LANL visualization capabilities

Metrics are diverse and plentiful, yet determining which metric is appropriate for a given problem is difficult. It would be useful if users could explore the way different metrics provide different views of the relationships between a single family of data sets. One method for performing such comparisons would be to use the relative distances provided by metrics to visually present relative relationships within the family of data sets. For example, one could compute all pairwise distances, and create a graph that has edge weights that correspond to these distances. Applying graph layout algorithms (such as those found in the Bell Laboratories `graphviz` package) would allow one to explore how different metrics change the relative presentation of the datasets.

A first step towards leveraging high-end visualization capabilities is to provide a clean method for exchanging data between IDA/Matlab and the visualization tool. Tools such as Ensign and Paraview have their own data representation formats, and there is a significant learning curve required to move what appears to be simple data (such as a two dimensional vector field) into the tool format. We should provide methods in IDA to hide the details of data format from the user.

Once we can demonstrate a rudimentary connection between IDA and a visualization tool, we can then proceed forward and explore how to use the metrics provided by IDA to aid in the presentation and visual data mining process that the viz tools enable.

I would consider the following visualization tools, with a priority towards the free tools (VisIT and Paraview). The IDA users should be provided the most usable, lowest cost-of-entry tool for them as possible - *not* the most politically correct. The priority in IDA development is always the IDA user, and our choices in tools and third party add-ons should keep that in mind.

- VisIT (LLNL)
- ParaView (LANL/Kitware)
- Ensign

## 4.2 Distributed, workgroup database

The DDMA workshop and team model is one in which the members are tasked with developing methods and performing analysis on data that is handed to us by clients. The current model is to either e-mail data sets around, or place them on a server requiring users to manually download them. Furthermore, people are required to read documents to acquire metadata information such as time or spatial units of the data sets. Should we tackle the metadata problems above and agree on representations, we can remove the metadata requirement via a properly designed database schema.

This still leaves the problem of how to get data to users, and leaves open the question of how one can eliminate the redundant computations performed by

the team. For example, it is possible that each team member will execute the same pre-processing step. It is currently impossible for any one member to take advantage of computations that another member has performed. Similarly, if at some point after project initiation the data files are updated, each user must be responsible for manually updating their local data on which they work.

We would like to move towards the model of making IDA able to connect to a workgroup database server that has the same structure as the local server, and use it for replicating their local computational results and acquiring the raw data for analysis. Doing so will require extending the current direct interface to SQLite with one that can go remote to access these databases.

#### 4.2.1 “Disconnected” database

Most DDMA users perform analysis on both their workstations and their laptops. As such, it is frequently the case that they are not connected to a network where they can access a workgroup server. To properly support the concept of a workgroup server, we would need a way to allow individuals to work independently of a workgroup server when necessary and provide mechanisms to reconcile differences that occur on their private database with those on the workgroup server at some later time. This is a highly nontrivial problem, and has been explored to some degree by folks working on traditional filesystems, such as the CODA and Intermezzo projects in the Linux world. This has also been looked at by database folks who deal with distributed databases. The difficulty in these schemes tends to lie in how one reconciles databases that have been disconnected for some period of time and have both independently changed since the most recent synchronization point.

### 4.3 IDA Language

The current IDA language for input decks is very simplistic, and allows users to create data flow pipelines with very rudimentary branching. Conditionals are not possible, and iteration is controlled by IDA - not the user. Variables and reuse of partial pipelines is accomplished by a form of macro expansion. Language techniques for expressing data flow models or function composition has been addressed by computer scientists in the past. For example, existing IDA pipelines are trivial to express in LISP or Scheme. Should we adopt a language that is equivalent to a subset of LISP or Scheme (possibly without so many parenthesis for the squeamish), we can take advantage of the control flow constructs in those languages to extend the IDA language. This would likely turn the IDA core code into a rudimentary interpreter of these languages or the  $\lambda$ -calculus directly. This would be a big change internally to IDA, and would require us to modify the language. It would have the benefit of making IDA input decks more “programmable”, allowing users to write control logic in the input deck as opposed to pushing it down into a custom `.m` file. Furthermore, it would make automatic parallelization easier.

It must be stressed that a formalized IDA language must be carefully designed. We have explicitly forbidden the use of the full Matlab language from the IDA pipeline specification in order to provide an abstraction layer that decouples the expression of pipelines and iterators from their actual implementation. This is intentional, and can allow the IDA engine to hide some degree of parallelism and data reuse from the user. We must ensure that a more formalized IDA language continue to preserve this abstraction. Therefore naive embedding of complete languages such as Scheme, LISP, Python, or others is strictly forbidden.

### 4.3.1 Visual programming

It has also come up periodically that the data flow nature of IDA input decks is naturally suited to visual representation. Tools such as LabView have implemented this in the past, and Matt’s work on the ViNE project (UOregon) visual programming editor can likely be leveraged in building one for IDA.

## 4.4 Explicit iterators

IDA functions by iterating over data inputs, performing analysis processes on them via specified analysis pipelines. Not all processes that users want to do can be modeled simply as “ $\forall I \in \text{INPUTS}, F(I)$ ”. For example, one might have two analysis pipelines that produce equal numbers of outputs that must be combined in a pairwise manner to produce single outputs. This can be called “zipper” iteration. Similarly, one might have an analysis pipeline that produces outputs that must be combined individually with each output of another pipeline. This can be represented using the “fold” operator of functional languages such as Scheme or Haskell.

It should be noted that there has been a great success recently in the move away from explicit, imperative loops (such as the for loop) in order to take advantage of high scalability and transparent use of parallel and distributed computing. One case that everyone is familiar with is Google. Recent papers have discussed the “map-reduce” model of computation used at Google to express algorithms that can run on their tremendous parallel and distributed server farms. Map and reduce are operators that are familiar to functional language programmers and are quite powerful in both their computational power and their high level of abstraction. There is a natural fit for models such as this in the IDA package.

These iteration operations are currently implicitly represented in IDA – the user does not specify them, but relies on choosing the correct function type in their pipelines to make it happen. I would argue that iterators should be made explicit for many reasons. First is to ensure that IDA does not violate the law of least surprise. Multiple function types that take two inputs yet iterate them differently may be confusing to users, and naive assumptions can cause the wrong one to be chosen. Second, and more importantly, is the fact that

much of the parallelism that can be extracted from an IDA pipeline is in the iteration pattern that it performs.

This has been recognized in the parallel computing community, and has led folks to use C++ iterators to automatically identify parallel execution opportunities (ie, POOMA). New languages such as Chapel (Cray) and FORTRESS (Sun Microsystems) make iterators an important part of how they represent parallelism. IDA should take advantage of this.

## 4.5 Regression testing

IDA should provide some mechanism for regression testing to verify that new version behave as expected on a set of problems of interest for users and IDA developers. IDA currently is validated against a simple test suite (ie, whatever Tom and Matt have available), but there is no way for a user to define a test suite of their own on which they can test new IDA versions to ensure that the new version behaves properly on their specific problems.

## 4.6 Tools for porting Matlab to C/F95

The lifetime of an analysis tool in Matlab, especially a successful one, requires IDA to support the evolution from raw Matlab prototype code to compiled, possibly parallel implementations in C or Fortran 95. No real tools exist for performing such porting operations - the Mathworks compiler is very limited with respect to performance, and tools for source-to-source translation tend to be immature and intended for student theses in language theory instead of handling general purpose production code.

Much of the functionality in Matlab can be acquired through third party libraries callable from compiled languages <sup>1</sup>. High performance solvers and arithmetic libraries are available for both sequential and parallel systems (ATLAS, ScaLAPACK, SuperLU), as are sophisticated optimization routines (DAKOTA), functions such as FFTs (FFTW) and novel algorithmic libraries (Qhull). Larger packages such as PetSc could also be used to aid in fulfilling the standard library needs of codes being ported out of Matlab.

Work in building these tools should take two paths. First, identifying the set of third party libraries that are considered valid candidates for ported code to use to replace routines provided by Matlab. Second, constructing parser and compiler-like tools for processing Matlab code for source-to-source translation purposes.

This porting process is more important than many users would acknowledge, and requires looking out beyond the immediate needs of specific users of IDA. It is highly unlikely that a true parallel matlab will be available to run at very large scale, especially on novel processor architectures (such as the IBM Cell or other acceleration technologies). The licensing costs in addition to technological limitations of Matlab make this highly unlikely (and attempting to force

---

<sup>1</sup>In fact, Matlab is built out of these itself!

Matlab to do so would be unproductive). Furthermore, compiled code that is independent from a tool such as Matlab has a more reliable lifetime should one wish to preserve an algorithm for the indefinite future.

## 5 Constraints

IDA has very strict constraints on the software requirements of any feature. Additions to the set of allowed dependencies are carefully avoided unless they are largely optional for the average user, and even then, should be avoided if possible. Additions must be discussed with respect to portability and **more importantly**, the impact they would have on the average, modestly computer-savvy user. Keep in mind that many machines where IDA is installed do not have the luxury of “just grabbing something off of the web”. Furthermore, due to security constraints, it is not possible in some cases to allow IDA to carry a large set of .tar files along with it to fulfill dependencies (in other words, the “sumo” distribution concept is *banned* in IDA).

- **Matlab:** IDA must function on Matlab version 7.1. Although the IDA developers are likely to be trying new versions of Matlab as they come out, it may be difficult to keep the machines where the work is actually done up to date. As such, we cannot rely on any new versions being present for the near future. This version constraint will get kicked up at some point, but it will be rare and NOT at the rate that Mathworks emits new versions.
- **Compiled languages:** GCC 3.3, G++ 3.3. These are dictated unfortunately by the compiled version of Matlab 7.1. Java is also valid, and should conform to the Sun JDK version 1.4.2. Fortran is likely constrained to G77, as GFortran (with support for F90+) was only commonly available with the version 4.0 release of the GNU compilers.
- **Auxiliary tools:** Autoconf 2.59 or later (2.13 is NOT supported), GNU make, md5sum<sup>2</sup>.
- **Operating systems:** IDA should compile out of the box, given the above compiler and Matlab versions on the following : RedHat Enterprise Linux, Ubuntu 5.10 and 6.06, MacOSX 10.3 and later. RHEL is the “official” LANL distribution, and Ubuntu is commonly used by people who care about not using the heap of garbage that is RHEL. MacOSX is used by the kids who know that you don’t have to use Linux to have a nice UNIX box.

---

<sup>2</sup>This is part of most default distributions, and was used to avoid requiring IDA to carry it’s own MD5 implementation with it. Note that IDA originally used the built in Java MD5 hash algorithm, but this failed on sufficiently large files, while the command line tool does not explode on big files.

The most likely near future change in these requirements will be when Mathworks releases a Matlab that is compatible with compiled code that uses the GNU GCC 4.0 compiler suite. **When that occurs, we will kick the versions of GCC and G++ up to 4.0, and make GFortran an officially supported compiler.** This will allow us to write analysis algorithms in Fortran 95, which is arguably more appropriate for use in porting Matlab code to compiled versions as the languages are similar. C, although fast, is not the best choice for numerical algorithms. It persists simply due to easy access on most systems, not because it's the right choice.

## 5.1 Do it right, or don't do it at all

This may sound a bit preachy, but acceptance of new features or ways of doing things is strictly controlled with IDA. Many of the items that are listed in this document to be added to IDA are actually quite easy to do as a hack. For example, making IDA aware of the input data for use in IDAview could be accomplished by simply shoving them into the database. Unfortunately, this can have unintended consequences. For example, what if you find a data file was corrupted, and you update the file in the filesystem but fail to remove it from the database. Will IDA notice and replace the old data with a new one? Will it suggest that you re-run previous analysis? These are the questions that are part of "doing it right". Doing it badly is doing the hack of just loading the data into the database and asking questions later. Should you be inclined to do that, ask yourself "do I really want to be known as a really bad programmer?". Of course you don't - so think before acting. Good software engineering is dominated by time spent designing and thinking, a small portion of time implementing, and an indefinite and long period of debugging and refining.

## 5.2 Notes

- Additional complex functionality (such as using PostgreSQL or MySQL in place of SQLite) should be easy to implement without violating the above. Provide a C or Java interface to ODBC/JDBC, and put an abstract interface between IDA and the database (including the currently sqlite database). A user by default will not have additional requirements. Those that wish to use the bigger database can do so without making life more difficult for users who do not.
- Octave is not an option. Matlab and Octave are not compatible beyond the simplest scripts, and we've gone down the hybrid Octave/Matlab route once already and it was not a pleasant experience. The UOregon Octave-based tool looks promising, but it is not clear how to take advantage of that technology to handle Matlab code that is not compatible with Octave.