Distributed Matlab v0.6

Authors: Dan Hill (dhill@biomail.ucsd.edu), Samar Mehta (samar@cs.stanford.edu); 10/11/05

Distibuted Matlab helps to run computationally-intensive Matlab programs by utilizing the resources of a local network. To use Distributed Matlab, you will need a secure private network of computers running Windows 2000/XP where each computer is capable of running Matlab 6.5 or later. Earlier versions of Windows or Matlab may work but have not been tested.

The functionality of this software is similar in some ways to The Mathworks' Distributed Computing Toolbox. This software was, however, developed previous to and independently of the release of the commercial toolbox.

Typesetting note: All *italicized* words are defined in the glossary as well as in the text of the manual. These words appear in ***bold face*** the first time they are used in the text. Computer commands, computer names and directory names appear in a `fixed width` font.

Introduction

Distributed Matlab is intended for a small, private laboratory or office network with computers that are typically idle at night or on weekends. For certain computation-intensive Matlab programs, Distributed Matlab can utilize otherwise unused computer hours to increase the rate of processing. The main requirement for a problem to benefit from this approach is that it must be easily broken into pieces that do not depend on one another; this is typically the case in problems where the same Matlab code is run with many different inputs. For example, a Matlab script that repeats the same analysis on several data files or a function that repeats a simulation for many different parameters would both be candidates.

In this document, the Matlab code that will be run repeatedly is referred to as the ***user code*** and the files that contain the data needed to run one piece of the original problem are called an ***input data set*** (e.g., in the examples above, a single data file or a file listing one set of parameters would be an *input data set*). The result of running the *user code* on an *input data set* produces a corresponding ***output data set***.

A Distributed Matlab session proceeds as follows. One computer serves as the ***manager*** and runs the Distributed Matlab GUI. Other computers on the same local Windows network can be designated ***workstations***. The *manager* computer copies the *user code* and one *input data set* to each available *workstation*. It then starts Matlab on each *workstation* and runs the *user code*. When the *user code* on a *workstation* finishes, the *manager* retrieves the resulting *output data set* and sends a new *input data set* to the *workstation*. This process continues until all *input data sets* have been run.

Setting up a Network

To run Distributed Matlab, the local network must be configured to allow the *manager* to copy files to and run programs on each *workstation* and a Matlab path must be set for all *workstation* computers. These steps must be completed the first time Distributed Matlab is run on a particular network.

Note that Distributed Matlab operations are not secure. Passwords are unencrypted and malicious changes to the Distributed Matlab code could damage computers on the network. This program should thus only be run when the local network is known to be safe from security threats and the Distributed Matlab program has been obtained directly from the authors.

Workstations

Network information is provided in a ***network file*** that has one line for each computer that can be used as a *workstation*. This file can be edited by pressing the `Network File` button in the Distributed Matlab GUI. Each line should contain the following information separated by tabs:

1. Computer Name — Windows computer name as visible on the network
2. Username — Valid Windows user on the *workstation*
3. Password — Windows password for the above username
4. Scratch Directory — Share name of a directory on the *workstation* that can be used as a temporary directory. This directory must be accessible from the *manager* computer. If there are any spaces in the directory name, the entire name must be enclosed in double quotes.

As an example, suppose two *workstations* named `blue` and `green` are available. The computer named `blue` has a shared drive with share name `C` and a user `blueuser` with password `blue1`. The computer named `green` has a shared folder with share name `public stuff`. However, the main user of `green` does not mind if the machine is used as a *workstation*, but does not want to share her user name and password. The first step in this case would be to create a new user on `green` for Distributed Matlab; for example, user `dmatlab` with password `matlab1`. The *network file* should then read:

```
blue       blueuser   blue1      c\scratch
green      dmatlab    matlab1    "public stuff\scratch"
```

To check if the settings for `green` are correct, run the following line from a command prompt on the *manager* computer*:

```
net use "\\green\public matlab1" /USER:dmatlab
```

*Each Scratch Directory must be shared so that the listed user has write permission to that directory, and the user must have sufficient security rights to remotely execute applications. If the network is completely secure, the simplest solution is to make the user listed in the *network file* a member of the Administrator group. Less risky options are possible, but the process of making these security settings is not described here.

Matlab Path

Distributed Matlab starts Matlab and executes the *source code* functions on each of the *workstations*. To ensure that the code will work correctly on all of these computers, the Matlab version and path should be compatible on each *workstation*.

The Matlab version may be an issue when *workstation* computers have different versions of Matlab installed, since the *user code* will run on different Matlab versions. For any single computer with multiple versions of Matlab, type `Matlab` at a command prompt on that computer (while logged on as the user from the *network file*) to determine the default version, which is what Distributed Matlab will use. Be sure to test the *user code* on any version of Matlab that may result from running this default version on all computers in the *network file*. In particular, if the *manager* computer is running an earlier version of Matlab than any of the *workstations*, the *manager* will not be able read `.mat` files generated by the later version of Matlab unless care is taken to enforce compatibility (for example, see the example in Appendix B).

The Matlab path must also be set to ensure that the *user code* sees the same Matlab environment on each *workstation*. This is accomplished by a path file that explicitly lists all Matlab directories that will be accessible by the *user code*; by default, this file places all subdirectories in the `MATLABROOT\toolbox` directory on the Matlab path. This default behavior should be acceptable in most circumstances, but the path file can be edited when necessary by pressing the `Path File` button in the Distributed Matlab GUI.

Parellelizing a Matlab Program

To prepare a Matlab program to be run by Distributed Matlab, the program is split conceptually into 3 parts: the code common to all of the computation (*user code*), the data needed to specify a particular piece of the computation (*input data set*) and the results desired from running *user code* on a particular *input data set* (*output data set*). All of this information is gathered in a directory called the ***Project Directory*** and must be organized as follows:

1. The *user code* is collected in a subdirectory called `source`.
2. Each *input data set* is collected in a subdirectory whose name starts with `input`. A simple approach is to use {`input1`, `input2`, … , `inputN` }.
3. The directories for the *output data sets* will be created by Distributed Matlab with names matching the corresponding *input data set* (e.g., for the numbered input example above, the *output data set* directories will be {`output1`, `output2`, … , `outputN` }. After running Distributed Matlab, these output directories will be collected in the ***Session Directory*** (defined below).

When the *user code* is copied to a *workstation*, an accompanying *input data set* will be copied to a subdirectory named `input`. Any results which are to be saved as part of the *output data set* must be saved in the subdirectory named `output` before the *user code* completes. Thus, for an existing Matlab program, the following steps are needed:

1. Identify the Matlab code common to all computations. Save all of these files in a directory called `source`.
2. Break the program into pieces that can be run in parallel. Create one directory for each of these pieces and put any files necessary to run each piece into the corresponding directory. Name these directories in an intuitive way.
3. Modify the code for the program so that it processes only one of the pieces and uses directories named `input` and `output`. For example, if the original code loads a series of files in a for loop, remove the for loop and load only one file from a subdirectory called `input`. At the end of the code, save any results into the `output` subdirectory.
4. To recreate the results of the original program, it may be necessary to write a script to collect the results from a group of `output` directories (named to match the `input` directories from (1) above). **This script can assume that the variable `session_name` exists and contains the name of the *session directory*. **

This organization is depicted graphically in Appendix A and an example is given in Appendix B.

In dividing up a problem, it is important to keep in mind the overhead incurred by Distributed Matlab. The average overhead time, S, can be estimated from the time needed to copy one *input data set* and one *output data set* over the local network plus a fixed cost of ~30 seconds for starting and stopping Matlab. As a rule of thumb, for a network with N *workstations* and a program with N *input data sets* that takes T seconds on a single computer, Distributed Matlab will provide an advantage when S is less than (N-1)/N * T/N. For example, consider a program that takes 60 minutes to run on a single computer. If the network has 10 *workstations* and the program is divided into 10 jobs, Distributed Matlab does not provide an advantage unless S is at least less than ~5 ½ minutes. Note that it is best to choose the number of *input data sets* N to be equal to the number of *workstations* when possible, to minimize overhead.

Running a Distributed Matlab Session

To run Distributed Matlab on a Matlab program that has been prepared as described above, identify one computer as a *manager*. If each *input data set* contains a large amount of data and all of this data is currently on one computer, it is most efficient to use that computer as the *manager*. Otherwise, the identity of the *manager* computer is unimportant – it is even possible to use a single computer both as a *manager* and as a *workstation*, if that computer has sufficient resources to run two simultaneous instances of Matlab.

Start Matlab on the *manager* computer. Confirm that the folder is on the Matlab path. Type `distributed_matlab` to start the GUI. The GUI asks for the following information before starting a session (items 1-4 are required):

1. *Start Time*: The time at which the session should be started: DD-Mon-YYYY, HH:MM:SS
2. *Termination Time*: The time at which the session should be stopped: DD-Mon-YYYY, HH:MM:SS
3. *Project Directory:* The directory containing the `source` and `input` directories described above
4. *Remote Command*: The *user code* Matlab command that should be run on each *workstation*
5. *Initialize Command*: A Matlab command that should be run on the *manager* before starting
6. *Wrap-up Command*: A Matlab command that should be run on the *manager* after finishing

Once this information has been entered, press the `Start` button to run the session. The time at which the start button is pressed and the name of the *manager* computer are used to create a ***session name*** (e.g., `HOST_blue__START_20050926T190000`) which is displayed on the GUI. The start time is in the date/time ISO 8601 format yyyymmddThhmmss). A directory with the same name will be created in the *Project Directory*; this is called the *Session Directory*.

After creating the *Session Directory*, the GUI will go through the following steps (see Appendix A for a diagram):

1. Wait until the system clock on the *manager* computer reaches *Start Time*.
2. Create a list of *workstations* from the file *Network File*.
3. Create a list of *input data sets* from all subdirectories of *Project Directory* whose names start with `input`.
4. The *Initialize Command* (if any) is run on the *manager*.

For each *workstation*, steps 5-8 are repeated.

5. The `source` directory from *Project Directory* is copied to a scratch directory on the *workstation*. Subdirectories named `input` and `output` are created in this directory.
6. A new *input data set* is chosen and the corresponding `input` directory is copied to the `input` directory on the *workstation*.
7. Matlab is started on the *workstation* in the scratch directory created in step 4. An 'opt-out' window appears on the *workstation*'s screen to allow a user to force Distributed Matlab to stop using that *workstation*.
8. *Remote Command* is run on the *workstation* as if it were typed at the Matlab prompt.

9. When all *workstations* have been assigned work, the *manager* checks at regular intervals to determine if any *workstation* has completed its processing. When a *workstation* is done, the *manager* copies the `output` subdirectory from the *workstation* to a new subdirectory (in the *Session Directory*) whose name starts with `output`. Steps 5-8 are then repeated to assign a new job to the *workstation*.
10. This cycle stops when either (1) all *input data sets* have been attempted, (2) the *Termination Time* is reached, or (3) the `Stop` button is pressed in the Distributed Matlab GUI. If any *workstations* are still busy when (2) or (3) occurs, these *workstations* are interrupted.
11. The *Wrap-up Command* (if any) is run on the *manager*.

Note that each *workstation* that starts an *input data set* can terminate in one of three ways: (1) forced to quit by Distributed Matlab, (2) Matlab error encountered in *user code*, or (3) successful termination of *user code*. In all cases, an `output` subdirectory is created in the *Sesssion Directory*. This subdirectory will contain all files that were in the *workstation's* `output` directory when the *workstation* terminated.

Information about a Distributed Matlab Session

Information about the progress of a Distributed Matlab session can be useful both while the session is running and after it completes. Uses include checking on progress, determining if any *input data sets* did not complete successfully, and trouble-shooting *workstations* or *input data sets* that cause errors. This information is available in the following forms: The *manager* display, *manager* and *workstation* log files, and a Matlab log data structure.

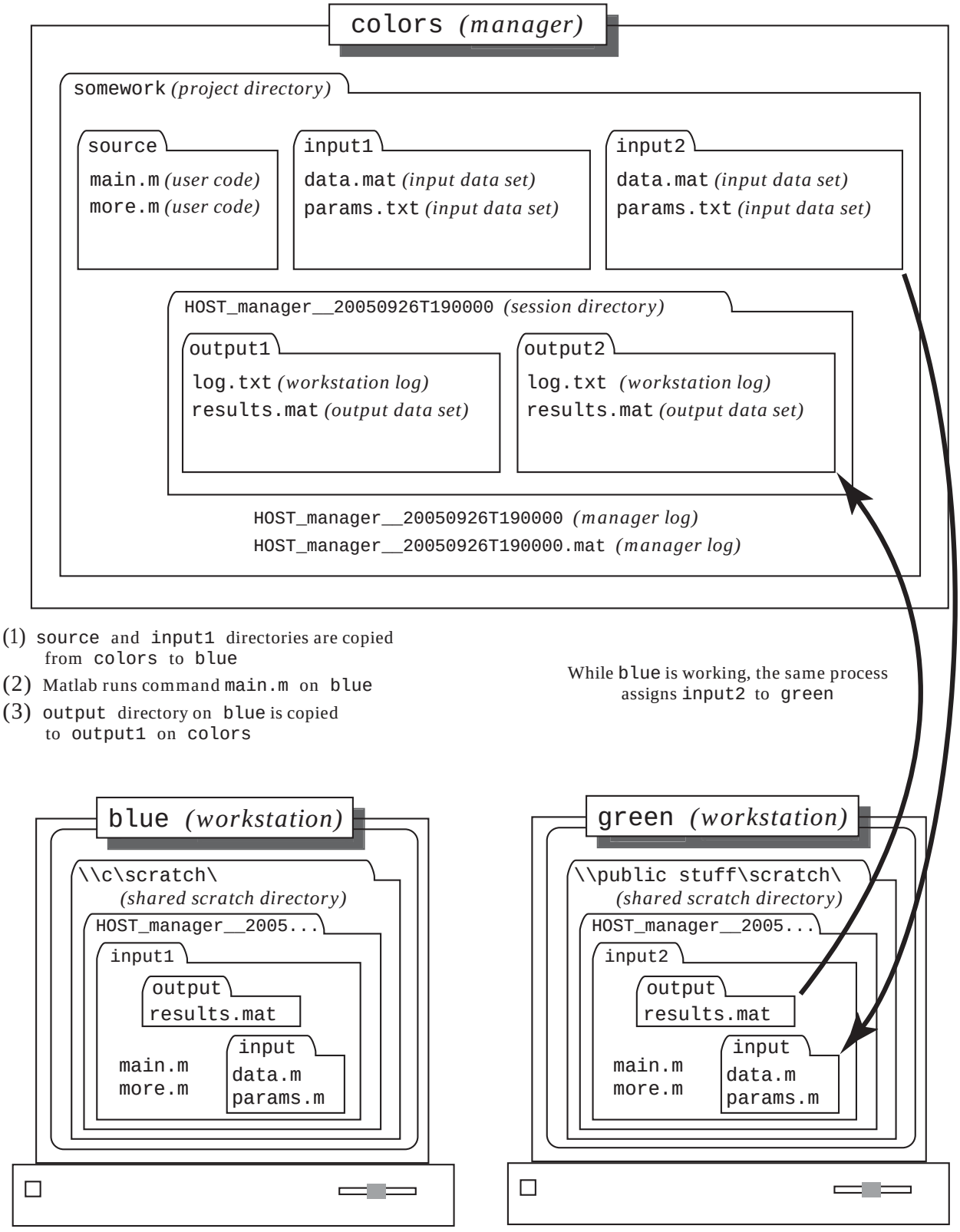
Manager display: While a Distributed Matlab session is running, a series of messages will be printed in the command window of the Matlab instance that is running Distributed Matlab. These messages include notification of *workstation* log-ons, creation and assignment of *input data sets*, *workstation* termination messages, etc. When the session completes, a summary of the entire session is printed in this display.

Manager log files: All messages in the *manager* display are saved in a *manager* log. While a Distributed Matlab session is running, pressing the `Log File` button will open a text file with the same messages displayed in the *manager* display. After a session has completed, all of these messages are saved to a file, in the *Project Directory*, named after the *session name*. This file can be opened with any text editor to review the progress of a session.

Matlab Log data structure: The information in the *manager* log file, along with several internal Distributed Matlab parameters, is saved as a Matlab data structure in a file (located in the *Project Directory*) named after the *session name* and ending in `.mat`. This is mainly used for trouble-shooting purposes, and this file can be safely deleted after a session has ended.

Workstation log files: If a *workstation* encounters an error while processing an *input data set*, messages about the error may be recoverable from the log file for that *input data set*. While an *input data set* is being processed, the `Busy Station`, `Busy Job`, and `Get Remote Log` controls in the GUI can be used to view a *workstation* log file. After an *input data set* has completed, a corresponding *workstation* log file, named `log.txt`, is saved in the appropriate `output` directory. Note that when an error occurs on a particular *input data set*, that *input data set* is tried again (up to three times). The *workstation* log files for both failed and successful attempts are saved in the *Session Directory* to aid in determining the cause of an error.

Appendix A: Organization Diagram



Appendix B: An Example

We consider an example of an analysis using Distributed Matlab to demonstrate the process of parallelizing code. We assume that the network has already been prepared as described in 'Setting up the Network' above and focus on the Matlab code.

Original program

Suppose that we have measured N two-dimensional data points in an experiment and repeated this experiment M times. The resulting data is stored in two N x M matrices, `xpts` and `ypts`. We simulate this data by calling

```
N = 500;   % number of points
M = 50;    % number of experiments
xpts = randn(N,M);    % generate data
ypts = randn(N,M);
```

We are interested in calculating the mean distance over all pairs of points in a given experiment and then repeating this for all experiments. This can be done with the following short program:

```
mean_distances = zeros(1,M);
for xprmt = 1:M   % for each experiment, …
  for j = 1:N     % … loop through all pairs of points
    for k = (j+1):N
      distance = norm([xpts(j,xprmt),ypts(j,xprmt))] – …
                          [xpts(k,xprmt),ypts(k,xprmt)]);
      mean_distances(xprmt) = mean_distances(xprmt) + distance;
    end
  end
end
mean_distances = mean_distances ./ ((N*N-1)/2);  % normalize
```

We can then look at the distribution of mean distances using

```
figure;  plot(mean_distances);
```

This code can be found in the file `meandistance.m` in the `distributed_matlab_example` subdirectory of the Distributed Matlab directory.

Parallelized program

To parallelize the `meandistance` program, we break the above code into three functions: generate *input data sets*, process a single *input data set* and collect *output data sets*. The code for two of these functions (`generate_data` and `collect_results`) can be found in the `distributed_matlab_example` subdirectory of the Distributed Matlab directory. The code to process a single *input data set* (`parallel_meandistance.m`) can be found in the `distributed_matlab_example\source` subdirectory of the Distributed Matlab directory

The function `generate_data` below creates input directories to split the data into three pieces:

```
%% Generate Data
N = 500;      % number of points
M = 50;       % number of experiments
xpts = randn(N,M);    % generate data
ypts = randn(N,M);

%% Create input data directories
projectdir = fileparts(mfilename('fullpath'));
subsets = {[1:20],[21:40],[41:55]};  % split up experiments
for c = 1:length(subsets)
  % select a piece of data
  xdata = xpts(:,subsets{c});
  ydata = ypts(:,subsets{c});

  % make an input directory
  inputdir = ['\input' num2str(c)];
  mkdir(projectdir, inputdir);
  % save subset of xpts/ypts to the input directory
  save([projectdir inputdir '\data.mat'], 'xdata', 'ydata');
end
```

Here, subsets of the data are chosen in advance and a directory is created for each. The data file (`data.mat`) and data variable names are the same ('`xdata`', '`ydata`') in each `input` directory.

The analysis code is untouched except for loading data from and saving results to the appropriate directories. The function `parallel_meandistance` includes these steps:

```
%% Load data from input directory
load('input\data.mat');
xpts = xdata;  ypts = ydata; % rename data to match original name
N = size(xpts,1);  M = size(xpts,2);  % get data parameters

%% Calculate Distances
mean_distances = zeros(1,M);
for xprmt = 1:M   % for each experiment, ...
  for j = 1:N   % ... loop through all pairs of points
    for k = (j+1):N
      distance = norm([xpts(j,xprmt),ypts(j,xprmt))] – …
                                [xpts(k,xprmt),ypts(k,xprmt)]);
       mean_distances(xprmt) = mean_distances(xprmt) + distance;
    end
  end
end
mean_distances = mean_distances ./ ((N*N-1)/2);  % normalize

matlabversion = ver('matlab');
if (str2num(matlabversion.Version(1)) > 6)
     save('output\result', 'mean_distances', '-v6');
else
     save('output\result', 'mean_distances');
end
```

The closing if statement is needed when the *manager* is running Matlab 6.5 and at least one *workstation* is running Matlab 7.0. This code forces Matlab 7.0 to save data in a format that can be read by Matlab 6.5.

Finally, the results of these calculations need to be assembled before they are plotted. This is done in `collect_results`:

```
%% Collect Data
sessiondir = [fileparts(mfilename('fullpath')) '\' session_name];
results = [];
for c = 1:length(subsets)
  % get results from output directory
  outputdir = ['\output' num2str(c)];
  load([sessiondir outputdir '\result.mat']);

  % collect output data set
  results = [results mean_distances];
end
mean_distances = results;  % rename to original name

%% Display Results
figure;  plot(mean_distances);
```

As in the `generate_data` function above, the main issue here is to get the output directory names correct. The data are then assembled and give the same plot as the original code.

In the example above, 50 experiments were split into 3 pieces as a demonstration. However, with these parameters, the computation per *input data set* is small enough that the ratio of Distributed Matlab overhead to computation time is large. Thus `meandistance.m` will likely run faster on one computer than `parallel_meandistance.m` on three *workstations*. The same example with a larger amount of data and more computers (e.g., 400 experiments, 2000 data points, 5 *workstations*) can be tried to better demonstrate the computational advantage of parallelizing the code.

Appendix C: Glossary

input data set	The set of input files that distinguishes one subproblem from another. See *user code*.
manager	The computer that runs the Distributed Matlab GUI and assigns *input data sets* to *workstations.*
network file	A file containing login information for all computers to be used by Distributed Matlab.
output data set	The set of output files that are the result of computing one subproblem. See *user code*.
project directory	The directory containing the *user code* and *input data sets*. This directory will also contain the *session directory* and session log files after Distributed Matlab starts.
session directory	The directory that will contain *output data sets* as they are completed.
session name	The name used to identify a particular Distributed Matlab session, in the format: HOST_hostname__START_YYYYMMDDTHHMMSS
user code	Matlab code that processes an *input data set*. The *user code* should be able to run on any of the *input data sets* to produce an *output data set*.
workstation	A computer used for Distributed Matlab computation. These computers are assigned *input data sets* by the *manager*.