# Remote access and processing of PATSTAT data

Mark Huberty*

April 17, 2012

## 1 Introduction

This document provides background on how to access the PATSTAT database as hosted at `durkheim.berkeley.edu`, use R and python to query and process data, and move results back to your local machine for analysis and use. It assumes that any access to `durkheim` will occur via the command line, or via implicit remote connections through a locally-invoked version of R, python, or MATLAB.

All code for this session is hosted at `github`. See `https://github.com/markhuberty/durkheim_mysql_remote` for more detail.

## 2 Tools

Remote access will require several tools. For the purposes of this document, we assume you are using the following:

- Remote access
  - Windows

*Travers Department of Political Science, University of California, Berkeley. Contact: markhuberty@berkeley.edu.

Windows users need to install separate programs for remote access. PuTTY[1] is the default client and works well for most people. You should have both the SSH (remote shell access) and SCP (remote copy) pieces installed.

  - OS X

    OS X users have access to the SSH and SCP clients via the Terminal application found under `Applications/Utilities`.

  - Linux

    Linux users should have the SSH client installed by default and available inside their distribution's terminal emulator.

- Python

  Python access to the database should occur using the `MySQLdb` module[2]. That is installed on the server and can be called via `import MySQLdb` or a suitable variant.

- R

  R access to the database is most easily had through the `RMySQL` interface[3], available through CRAN.

- MATLAB

  Matlab access to MySQL and other databases is supported via the `database` routines[4].

[1] http://www.chiark.greenend.org.uk/~sgtatham/putty/
[2] http://mysql-python.sourceforge.net/
[3] http://cran.r-project.org/web/packages/RMySQL/index.html
[4] http://www.mathworks.com/help/toolbox/database/ug/database.html

## 3 Remote or local sessions?

Database access can occur through either a remote or a local session. "Remote" here means that the R or python process is running on `durkheim` itself, so that the connection to the database occurs from a process on the server to the server database. "Local" means that the R, python, or MATLAB process is running on a user's local machine, which accesses `durkheim` solely for the purpose of connecting to the database, executing the database query, and pulling the data back into the user's local session.

"**Local**" access has clear advantages for ease of use. However, users who want to do this should recognize that there are also downsides:

- If the network connection is unstable, it might die mid-query, taking the query and any output with it. This is particularly problematic for querying large databases like PATSTAT, where execution times might be minutes or tens of minutes
- If the output is large, there will be a long delay as the output is pulled back through the network connection. So, for instance, a 1 Mb/s connection (a FAST connection) will take 1.5 minutes to pull back 100mb of output.

We recommend that users wishing to run local sessions connect to the database via an SSH tunnel, which is more secure and easier for the server administrator to manage. More information on tunnelling is provided in each section below.

"**Remote**" connections can work in one of three ways:

1. Log into the server and run an interactive session in R or python, pushing code stored on the server to an R or python session running on the server
2. Log into the server but run local code on the remote server. Note that this is distinct from the "local" option above: though the code is local, the R or python process is

on the remote box, so there's no need to suck large amounts of data over a network connection

3. Log into the server and run code as a batch process, in which a script executes beginning-to-end without user interaction.

Note that option (2) really only works for R, via either R Studio's remote server protocols or Emacs' `ess-remote` mode.[5]

# 4 Example R sessions

## 4.1 Interactive remote R session, remote code

Emacs and ESS provide a full-featured remote suite for writing and executing R code on the server. The steps are:

1. Log into `durkheim` via ssh
2. Open the desired R code file in Emacs (`emacs mycode.R`)
3. Split the buffer (`C-x 3`)
4. Start the R session in the new buffer (`M-x R`)

A sample result is shown in figure 2. You will now be able to send code from the R code file to the R shell using standard Emacs codebindings. All saved output (figures and data) are written to the remote filesystem, not the local computer. The syntax for issuing database queries is shown in section 4.5

[5] ipython html notebook might be an answer here...

### 4.2 Interactive remote R session, local code with `ess-remote`

For running an interactive remote session from code local to your personal machine, Emacs provides the `ess-remote` function. The steps to run this go like this:

1. Open the desired code file in emacs on your local machine
2. Split the buffer as desired (e.g, `C-x 3` for side-by-side buffers)
3. In the new buffer, invoke an emacs-shell (`M-x shell`)
4. From the shell, log into the server and start R from the comamnd line, as in:[6]

```
$ ssh username@durkheim.berkeley.edu
Welcome to Linux durkheim 2.6.32-37-generic
#81-Ubuntu SMP Fri Dec 2 20:32:42 UTC 2011 x86_64 GNU/Linux
Ubuntu 10.04.1 LTS

Welcome to Ubuntu!
 * Documentation:  https://help.ubuntu.com/

77 packages can be updated.
76 updates are security updates.

Last login: Mon Apr 16 14:26:33 2012
from airbears-136-152-132-25.airbears.berkeley.edu
username@durkheim:~/ R
R version 2.13.0 (2011-04-13)
```

[6] http://www.r-bloggers.com/run-a-remote-r-session-in-emacs-emacs-ess-r-ssh/

```
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-unknown-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

5. Attach the remote R session to your local `ess` process (`M-x ess-remote`)

You should now be able to push code from your local file to the remote R process with the standard ESS keybindings. Figure 2 shows that this looks no different from running in a local R session. Note, though, that any *saved* output (figures, data, etc) will be written to the filesystem on the remote server. The syntax for issuing database queries is shown in section 4.5.

### 4.3 Interactive local R session, remote database connection

Users who don't want to use Emacs and still want an interactive R session may use whichever R interface they desire, and connect to `durkheim` solely for interacting with the database. This should be done through a tunnelled SSH connection. This occurs in two steps: creating the tunnel, and running the db connection in R.

To create the tunnel, log into `durkheim.berkeley.edu` using an SSH client as follows:

```
ssh -L 3306:localhost:3306 username@durkheim.berkeley.edu
```

This is doing three things:

1. Establishing an open SSH connection to the server
2. Forwarding port 3306 (which mysql listens on) through that connection
3. Mapping that port to localhost, the implicit address of the local machine

Users can then issue calls to the database from within a local R session as follows:

```
library(DBI)
username = "markhuberty"
password = "***"
db = "patstatOct2011"

my.query = "SELECT count(*) FROM tls206_person
            WHERE person_ctry_code=="US""

## Establish the connection
```

```
conn = dbConnect("MySQL",
                 username=username,
                 password=password,
                 dbname=db,
                 host="127.0.0.1",
                 port=3306
                 )
## Pass the query to the db
query = dbSendQuery(conn, my.query)

## Note that -1 fetches all records. Set it to a
## 0 < N < infinity to fetch N rows.
output = fetch(query, -1)

dbDisconnect(conn)
```

### 4.4 Running R code in batch

You can also run R code on the server in batch mode. Here, you provide a complete R script that runs and saves output to one or more files, and run the entire script at once. Given that scripts may take some time to execute, it's important to do this such that you can log out (or the connection can die) without stopping the batch process.

The steps to do so go as follows:

1. Log into `durkheim` with `ssh`
2. Execute your script with this syntax:

```
nohup R CMD BATCH mycode.R &
```

3. Wait for the script to finish and write output to the files you indicated

Here, `nohup` provides a "no hangup" signal to the shell, so that it does not stop the batch process when you log out. `R CMD BATCH mycode.R` executes the code in batch mode. Finally, `&` pushes the process to the background and returns control of the shell to the user. Any status, print, or error messages are written to a `nohup.out` file.

### 4.5 Database querying in R

Querying a database in R is straightforward when using the `DBI` package, available from CRAN. The code segment below shows how this can be done easily:

```
library(DBI)
username = "markhuberty"
password = "***"
db = "patstatOct2011"

my.query = "SELECT count(*) FROM tls206_person
            WHERE person_ctry_code=="US""

## Establish the connection
conn = dbConnect("MySQL",
                 username=username,
                 password=password,
                 dbname=db
                 )
```

```
## Pass the query to the db
query = dbSendQuery(conn, my.query)

## Note that -1 fetches all records. Set it to a
## 0 < N < infinity to fetch N rows.
output = fetch(query, -1)

dbDisconnect(conn)
```

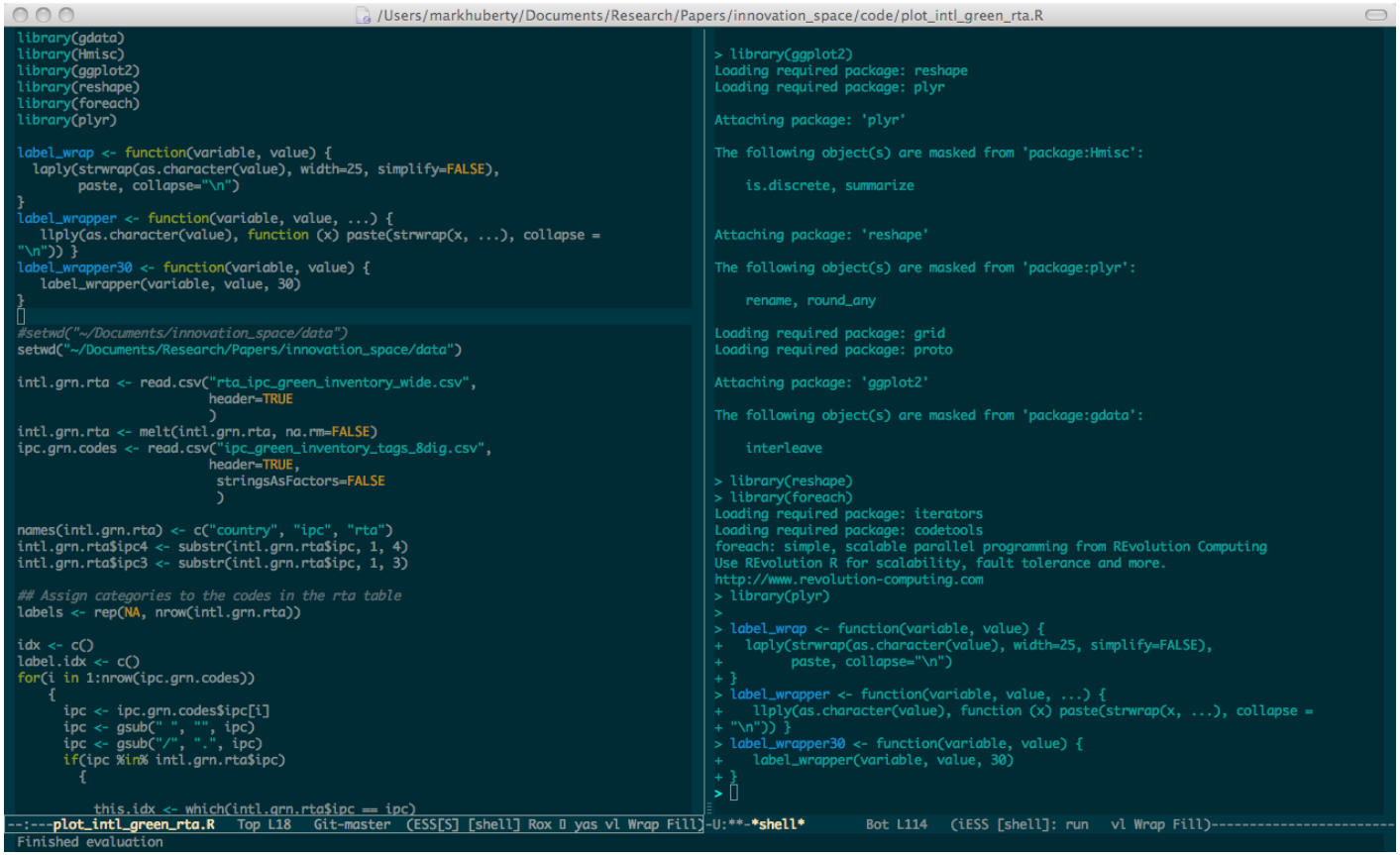


Figure 1: A remote R session running from local code via `ess-remote`. The user's code is on their local machine, but the R session is running inside an ssh connection to `durkheim`, via ess-remote. Code can be sent from the code buffer to the ESS shell via standard Emacs ess-mode keybindings.

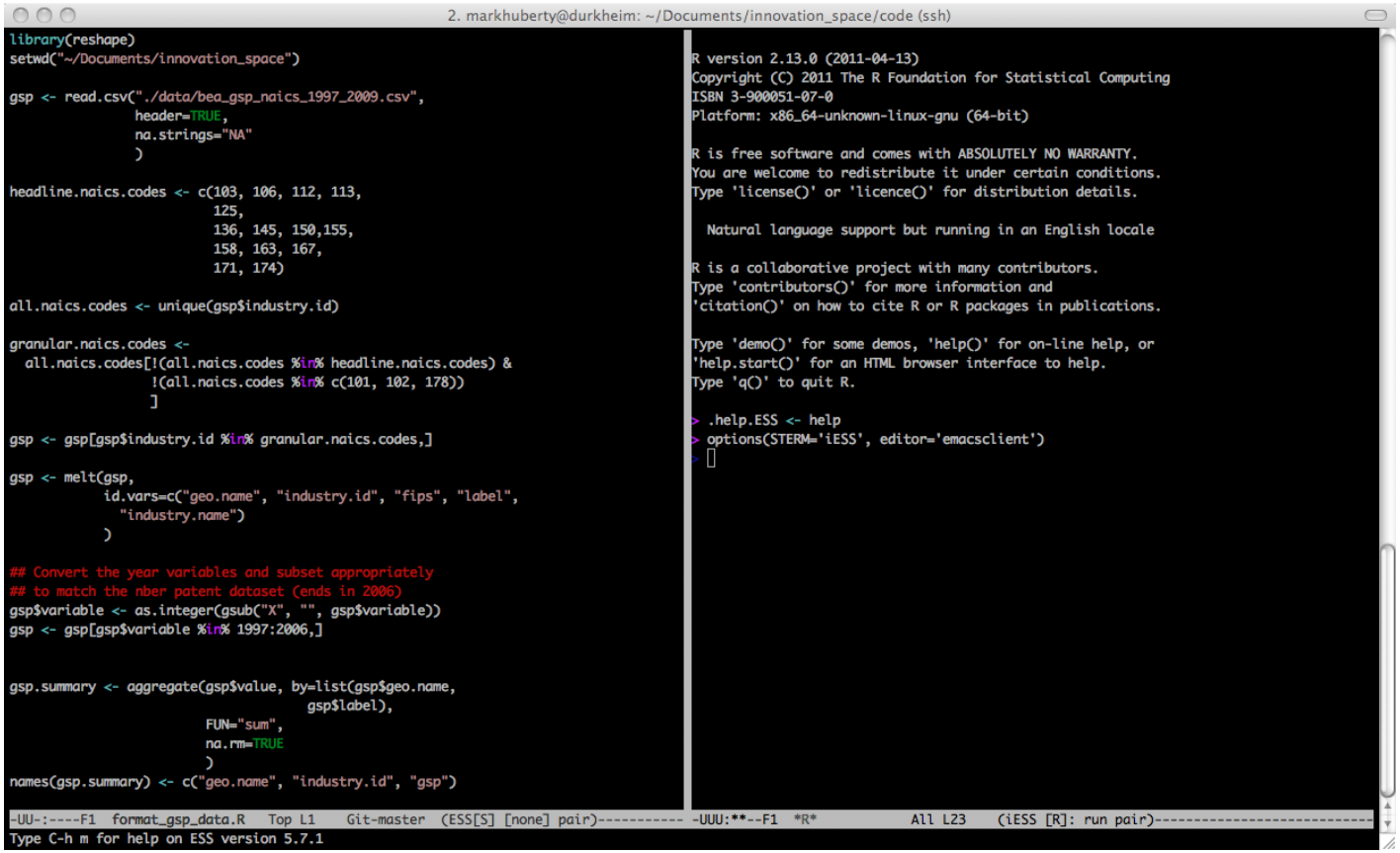


Figure 2: A remote R session running with remote code and a remote shell.. The user is logged in to `durkheim`. The left-hand buffer is an R code file open in Emacs. The right-hand buffer is an interactice ESS shell. Code can be sent from the code buffer to the ESS shell via standard Emacs ess-mode keybindings.

# 5 A sample Python session

## 5.1 Local interactive sessions in python

You can run a local python session and only connect to the server for the purposes of issuing database queries and receiving output. The easiest way to do this is via a `ssh` tunnel. This is easily done via any SSH client, including PuTTY.

From the client window, log into `durkheim.berkeley.edu` as follows:

```
ssh -L 3306:localhost:3306 username@durkheim.berkeley.edu
```

This is doing three things:

1. Establishing an open SSH connection to the server
2. Forwarding port 3306 (which mysql listens on) through that connection
3. Mapping that port to localhost, the implicit address of the local machine

Basically, this means that when python goes looking for mysql, it looks for it over the server connection rather than on the local machine. For more information on port forwarding for MySQL on Windows, see http://www.codemastershawn.com/library/tutorial/ssh.tunnel.php.

Once this is done, fire up python and import `MySQLdb`. The syntax for issuing a query looks like this:

```
import MySQLdb

my_query = "SELECT count(*) FROM tls206_person
            WHERE person_ctry_code=="US""
```

```
## Establish the db connection
## Note here: if you are running remote code in a remote
## python session, host="localhost" is correct. Otherwise,
## host="" is required so that the code goes looking for the right
## remote connection
conn = MySQLdb.connect(host="127.0.0.1",
                       user="markhuberty" ,
                       passwd="***",
                       db="patstatOct2011"
                       )

conn_cursor = conn.cursor()

## Execute the query
conn_cursor.execute(my_query)

## Collect the output
output = conn_cursor.fetchall()
## Shut down the
conn_cursor.close()
conn.close()
```

### 5.2 Remote interactive sessions in python

Unlike R, python lacks a simple mechanism for sending local code to a python session on a remote server. This leaves only the remote/remote option. The iPython HTML Note-

book functionality may provide a way around this, but is less useful for data-intensive operations

#### 5.2.1 Remote code and remote python session with emacs

Emacs and iPython provide a powerful interactive data analysis interface and make working remotely much easier. The steps are:

1. Log into `durkheim` via ssh
2. Open the desired Python code file in Emacs
3. Split the buffer (`C-x 3`)
4. Start the ipython session (`C-c !`)

A sample result is shown in figure 3. You will now be able to send code from the python file to the iPython session directly using the standard iPython keybindings. Again, all saved output (figures and data) are written to the remote filesystem, not the local computer.

#### 5.2.2 Local code and remote python session with ipython-notebook

(Still working on this problem; ipython html notebook might be the best way to proceed.))

#### 5.2.3 Running python code in batch

Like R, python scripts can be run start-to-finish without an interactive session. Also like R, the code must be structured to run without user input, and to write any desired output to files.

The process for running python code in batch is much the same as for R::

1. Push your script to your server account via either `scp` or `git`
2. Log into `durkheim` with `ssh`
3. Execute your script with this syntax:

```
nohup python mycode.py mycode.R &
```

4. Wait for the script to finish and write output to the files you indicated

Note that, unlike R, python will only write status and error messages to `nohup.out` when the entire process is complete.

### 5.3 Querying the database from Python

Python provides easy access to MySQL databases via the `MySQLdb` module, which permits querying and extraction of data from the database into standard Python objects. `MySQLdb` provides for both local and remote database connections, via a consistent syntax. The code below provides a sample syntax for doing so with PATSTAT.

```
import MySQLdb

my.query = "SELECT count(*) FROM tls206_person
            WHERE person_ctry_code=="US""

## Establish the db connection
## Note here: if you are running remote code in a remote
## python session, host="localhost" is correct. Otherwise,
## host="" is required so that the code goes looking for the right
```

```
## remote connection
conn = MySQLdb.connect(host="localhost",
                       user="markhuberty" ,
                       passwd="***",
                       db="patstatOct2011"
                       )

conn_cursor = conn.cursor()

## Execute the query
conn_cursor.execute(my.query)

## Collect the output
output = conn_cursor.fetchall()
## Shut down the
conn_cursor.close()
conn.close()
```

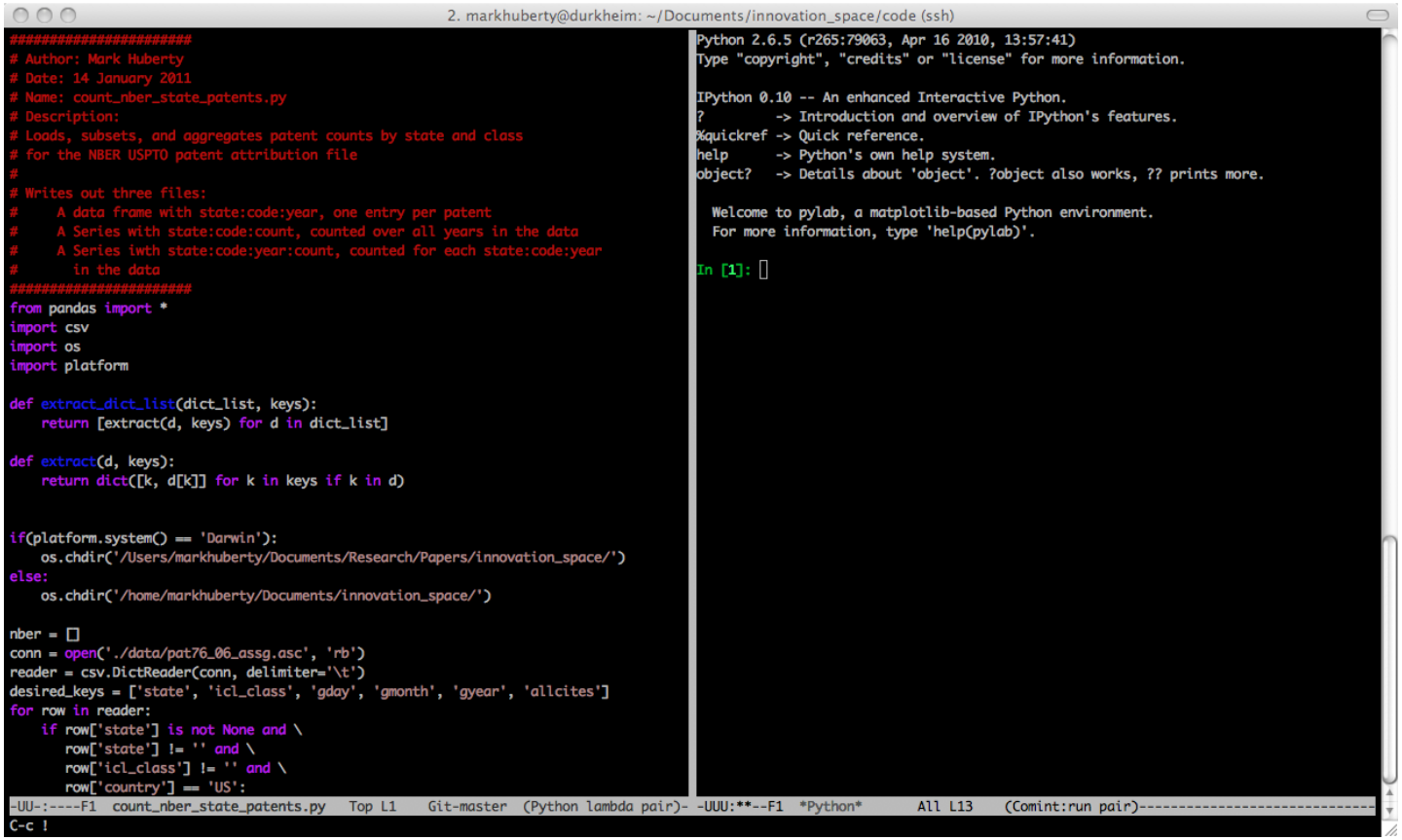


Figure 3: A remote python session. The user is logged in to `durkheim`. The left-hand buffer is a python code file open in Emacs. The right-hand buffer is a `ipython` interactive shell. Code can be sent from the code buffer to the python shell via standard Emacs python-mode keybindings.

## 6 A sample MATLAB session

NOTE: the author is not a MATLAB user and has no MATLAB install to test this against.

Caveat emptor, RTFM, etc.

```
conn = database('test_db','markhuberty','***','Vendor','MySQL',...
          'Server','durkheim.berkeley.edu')

query = exec(conn, my.query)
output = fetch(query, -1)
```