Restructuring Option Chain Data Sets Using Matlab

Alison M. Baker

Worcester Polytechnic Institute
Restructuring Option Chain Data Sets Using Matlab

A Directed Research Project

Submitted to the Faculty of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Professional Degree of Master of Science

in

Financial Mathematics

by

Alison Baker

May 2010

Approved:

Professor Marcel Blais, Advisor

Professor Bogdan Vernescu, Head of Department
Abstract

Large data sets are required to store all of the information contained in option chains. The data set we work with includes all U.S. exchange traded put and call options. This data set is part of a larger data set commonly referred to as the National Best Bid Offer (NBBO) data set. The national bid best offer is a Securities and Exchange Commission (SEC) term for the best available ask price and bid price. Brokers must guarantee investors these prices on their trades.

We have acquired data for the 5 year period from 2005 to 2009 for all U.S. traded options. Each year of data is approximately 6 gigabytes. The company, (DeltaNeutral - Options Data And End Of Day Downloads, 2010), from which we acquired the data, also has a software package, OptimalTrader, to process the data. For this data to be used in research projects, the data must be accessible by specific underlying security for selected date ranges. This type of data is more useful to the financial mathematics student than the output given by the software provided by DeltaNeutral.

The software used in this data manipulation is Matlab. Each individual file of original data was parsed, and new files were written with some reformatting in which the original data was largely reorganized. The new organization will make searching for information from one stock or any specific group of stocks easier to achieve. We have created 3 m-files in Matlab which deal with reformatting the data, error handling, and searching through the original or reformatted data. The result is that new datasets can be created for further studying and manipulation. Future students working with this data should find this method, toolset, and the newly constructed datasets to be useful tools in working with options data and examining option chains.
Acknowledgements

I would like to thank my advisor, Professor Marcel Blais, for his guidance, encouragement, support, and most of all for his patience throughout this process. Thanks to Adriana Hera, for her generous Matlab insight. And last but not least, thanks to my friends and family for their emotional support over the past 2 years.
Table of Contents

Contents
1. Introduction .................................................................................................................. 5
2. Data ........................................................................................................................... 7
   2.1 Structure ............................................................................................................... 7
   2.2 Variable Explanation ......................................................................................... 9
3. Our Approach ............................................................................................................. 11
   3.1 Matlab Scripts .................................................................................................. 11
   3.2 Creating New Data Sets ................................................................................... 11
   3.3 Dataset Parsing ................................................................................................. 13
   3.4 Writing New Files .............................................................................................. 14
4. Error Handling .......................................................................................................... 15
5. Data Processing Challenges ...................................................................................... 17
6. How To Use ................................................................................................................ 18
7. Conclusion ................................................................................................................ 25
A. Appendix .................................................................................................................. 27
   A.1 formatDateToTicker.m ......................................................................................... 27
   A.2 deleteAday.m ..................................................................................................... 29
   A.3 Option Extension Codes .................................................................................... 30
   A.4 Exchange Variable ............................................................................................ 32
B. Bibliography .............................................................................................................. 33
1. Introduction

We have acquired 5 years of options data from DeltaNeutral (DeltaNeutral - Options Data And End Of Day Downloads, 2010). The data is organized by date, where each file includes one full day of all put and call options trading on exchanges in the U.S. This data is referred to as the National Best Bid Offer (NBBO) data set. The national bid best offer is a Securities and Exchange Commission (SEC) term for the best available ask price and bid price. Brokers must guarantee investors these prices on their trades. DeltaNeutral also uses the Black-Scholes model to calculate the Greeks, (beta, gamma, theta, and vega), and the implied volatility, and includes them in this data set. The Fed Funds Rate is used as the risk free rate in these calculations. Dividends are not considered. In all, the data set from DeltaNeutral is approximately 30 gigabytes of data.

This project’s first aim is to create a new data set which will be formatted and organized in a way which is more suitable for financial mathematics students to use for options analysis and mathematical modeling. Along with creating a new data set, we have also created a template for searching the data and creating specialized file sets.

The main change in the data from its original format to the new format is separation between organization in files by date to organization in files by stock symbol (ticker), and within these files, by date. Also, the strings that are the expiry and data day need to be changed to numbers to allow for calculations involving time change.

The first step is to successfully read through each original file by date, in chronological order, one file at a time. The data is systematically written to new files arranged by their tickers. Data from each date thereafter is appended to rest of the data in each symbol file.
The try / catch function is used in order to find any problems in writing the new files. One problem found was that the symbol included a '/' character which cannot be used within a filename to save to the same folder. The '/' is now automatically changed to a ‘1’, which will cause no problems with writing files.

Since we are parsing 30 gigabytes of data, a failsafe was created to erase a whole data day from all of the newly created files. This uses a variable created while the files are being written, which stores the latest data day being processed. This data day is then used in the deleteAday script which will read the symbol files, delete any data from that data day in the output files and rewrite the output files without that information. The original script can then be restarted from that day without causing a repeat in any information already written to the new files.

The software used in this data manipulation is Matlab. Each individual file of original data was parsed, and new files were written with some reformatting in which the original data was largely reorganized. The new organization will make searching for information from one stock or any specific group of stocks easier to achieve.
2. Data

2.1 Structure

The original data set, provided by DeltaNeutral (DeltaNeutral - Options Data And End Of Day Downloads, 2010), is comprised of 5 years of options data, from 2005 through 2009, consisting of all U.S. traded options, using NBBO data. A list of exchanges can be obtained from the SEC’s website. (Exchanges, 2010)

Some of the options exchanges are:

- The Chicago Board Options Exchange (CBOE)
- BATS Options, New York Stock Exchange (including NYSE Arca, NYSE Alternext US),
- Philadelphia Stock Exchange (PHLX), now known as NASDAQ OMX PHLX
- International Securities Exchange Holdings, Inc
- The Chicago Mercantile Exchange (CME)
- The Chicago Board of Trade (CBOT)
- Boston Options Exchange (BOX)

Each of the 5 years of data is stored in a separate folder. Each year has one file of data per trading day. All of this data is stored in one folder labeled byDate. Each day of data (each original file) is structured as follows:

- Data is in comma separated values, csv, format. This means that each line, or row, of the file has information in it, each piece of information separated by a comma.
- Each row is an observation of one available option.
Within each line of data (row in the file), the information contained looks like this:

A,16.24,*,A,AR,call,01/16/2009,01/02/2009 04:00:00 PM,7.5,0,8.65,8.85,0,0,2.3849,0.9706,0.8822,-1.8602,0.2128

DeltaNeutral (DeltaNeutral - Options Data And End Of Day Downloads, 2010) describes the data per line with these names:

UnderlyingSymbol, UnderlyingPrice, Exchange, OptionRoot, OptionExt, Type, Expiration, DataDate, Strike, Last, Bid, Ask, Volume, OpenInterest, IV, Delta, Gamma, Theta, Vega
## 2.2 Variable Explanation

<table>
<thead>
<tr>
<th>DeltaNeutral Variable Name</th>
<th>Description</th>
<th>Example</th>
<th>Matlab variable name</th>
<th>Matlab Variable Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnderlyingSymbol</td>
<td>This is the stock symbol or ticker.</td>
<td>A</td>
<td>tickers (used for file names) uS (unique symbol) used for file contents</td>
<td>string</td>
</tr>
<tr>
<td>UnderlyingPrice</td>
<td>Price of the underlying on day of observation</td>
<td>16.24</td>
<td>price</td>
<td>double</td>
</tr>
<tr>
<td>Exchange</td>
<td>Data is from all US exchanges</td>
<td>*</td>
<td>exchange</td>
<td>string</td>
</tr>
<tr>
<td>OptionRoot</td>
<td>The option’s symbol</td>
<td>A</td>
<td>optroot</td>
<td>string</td>
</tr>
<tr>
<td>OptionExt</td>
<td>Contains information about expiry, strike, and type of the option (described further in appendix)</td>
<td>MT</td>
<td>optext</td>
<td>string</td>
</tr>
<tr>
<td>Type</td>
<td>put or call</td>
<td>put</td>
<td>optype</td>
<td>string</td>
</tr>
<tr>
<td>Expiration</td>
<td>expiration day</td>
<td>01/16/2009</td>
<td>expiry</td>
<td>string</td>
</tr>
<tr>
<td>DataDate</td>
<td>observation day</td>
<td>01/02/2009 04:00:00 PM</td>
<td>dataday</td>
<td>string</td>
</tr>
<tr>
<td>Strike</td>
<td>strike price</td>
<td>17.5</td>
<td>strike</td>
<td>double</td>
</tr>
<tr>
<td>Last</td>
<td>price at which option was last traded at</td>
<td>1.47</td>
<td>oplast</td>
<td>double</td>
</tr>
<tr>
<td>Bid</td>
<td>price buyer is willing to pay</td>
<td>1.39</td>
<td>bid</td>
<td>double</td>
</tr>
<tr>
<td>Ask</td>
<td>price seller is willing to accept</td>
<td>1.43</td>
<td>ask</td>
<td>double</td>
</tr>
<tr>
<td>Volume</td>
<td>number of transactions filed for that day</td>
<td>10</td>
<td>volume</td>
<td>double</td>
</tr>
<tr>
<td>OpenInterest</td>
<td>number of open stock options positions</td>
<td>1451</td>
<td>openinterest</td>
<td>double</td>
</tr>
<tr>
<td>IV</td>
<td>implied volatility (calculated using Black-Scholes algorithm and Fed Funds Rate)</td>
<td>0.4709</td>
<td>iv</td>
<td>double</td>
</tr>
<tr>
<td>Delta</td>
<td>[ \Delta = \frac{\partial V}{\partial S} ]</td>
<td>-0.7745</td>
<td>delta</td>
<td>double</td>
</tr>
<tr>
<td>Gamma</td>
<td>[ \Gamma = \frac{\partial \Delta}{\partial S} = \frac{\partial^2 V}{\partial S^2} ]</td>
<td>20.0497</td>
<td>gamma</td>
<td>double</td>
</tr>
<tr>
<td>Theta</td>
<td>[ \theta = \frac{\partial V}{\partial T} ]</td>
<td>-1.5105</td>
<td>theta</td>
<td>double</td>
</tr>
<tr>
<td>Vega</td>
<td>[ \nu = \frac{\partial V}{\partial \sigma} ]</td>
<td>0.9551</td>
<td>vega</td>
<td>double</td>
</tr>
</tbody>
</table>
Where (in the Greeks):

$V$ is value of the option.

$S$ is stock price.

$T$ is time to expiry.

$\sigma$ is volatility of the underlying.

Tables for explanation of option symbols, roots, and extensions are included in the appendix. There has been a change in the meaning of the “exchange” variable in DeltaNeutral’s data sets. The new meaning of the data in this variable is explained in a file from DeltaNeutral which is included in Appendix A.4.
3. Our Approach

3.1 Matlab Scripts

The use of Matlab scripts makes the type of calculation (dataset manipulation) easy to handle for several reasons.

The first is that there are no inputs or outputs. We do not need to enter any information and we do not want any returned. The purpose of this project is to create software for manipulation of datasets, formatting, deleting information, and searching within files for specific lines of data thus there is no need for output. Within the scripts are functions for creating new folders and writing files within these folders. This makes a convenient, compact set of information for the user to then perform data analysis Matlab or another programming language.

3.2 Creating New Data Sets

The first script, `formatDateToTicker`, is not intended to be changed (or even used) by the user. This file is used only to create an entire new data set from the original, producing a new data set with a different format.

The structure of the data set is largely similar to the original set, but there is reformatting and significant rearrangement of the data. The first step in this script is to successfully read through each original file, one by one, in chronological order.

The original files are named as ‘options_yearmonthday.csv’, where year is a 4 digit year (e.g. 2005), month is 2 digits (e.g. 02), and day is 2 digits (e.g. 15). The first file in chronological order is options_20050102.csv. The new files that are being created from the
original data are files containing only information about one underlying symbol. The data in these new files should be arranged in chronological order, therefore the original files need to be read in chronological order and data from each sequential file appended to the proper previous date’s information.

Example of data from an original file:

A 23.88 * A AA call 1/21/2005 1/3/2005 16:00 …
AA 30.99 * AA AA call 1/21/2005 1/3/2005 16:00 …
AAI 10.66 * AAI AA call 1/21/2005 1/3/2005 16:00 …
AAI 10.66 * AAI NC put 2/18/2005 1/3/2005 16:00 …

Example of data from a new file:

A 2.39E+01 * OAE MG put 733061 732315.7 …
A 2.39E+01 * OAE AH call 733061 732315.7 …
A 2.39E+01 * OAE MH put 733061 732315.7 …

The original file’s data is saved as a variable, C, which is then parsed, reformatted, and allocated to new files. In all, there are several hundred thousand lines in each date file, including data for approximately 3,000 unique underlying symbols.

The underlying symbols included in date files change through time. New companies may be formed or go public and options will begin to be sold on the underlying for that company. Others may dissolve, and options will no longer be traded on their stock. In the end there are approximately 3200 new files, one for each underlying symbol throughout the 5 years of data between the years 2005 and 2009.
3.3 Dataset Parsing

As mentioned above, each date file is read and saved as a variable. Since these data sets are comma delimited and contain a combination of numbers and strings (words or symbols), we used the Matlab functions `fopen`, `textscan`, and `fclose` to “read” the files.

`fopen` and `fclose` (file open and file close) are self-explanatory as regarding their purpose. We use `fopen`, then `textscan` to read the data, then `fclose`. `textscan` stores the data as a variable, so once `fclose` is used, the original file is closed and is left unchanged. We set $C = textscan(file, variable types, delimiter)$. Since the input file is a comma delimited file with numbers (type `double`) and strings, we define those when using the `textscan` function.

We have created a variable, `C`, which contains all of the data from the original file. This variable is a nested cell array with each of our variables stored as a separate array. It is nested because columns 1 and 3-8, which are read as strings, are cell arrays containing individual cells in which each string is stored separately. This is noted by the double indexing. For example, for a variable such as price in the second column which is type `double`, to get the price from the first row, you would index into `C` with $C{2}(1)$. To index into the third column to get the exchange which is type `cell`, you would need another layer of cell indexing, such as `exchange = C{3}{1}`.

The function `unique` is applied to its first column, `C{1}`, which contains the stock symbols. This gives us a list of each symbol for which we have data (there may be hundreds of lines of data for each symbol), and by using the option 'first', `unique` gives an array of the first index for each symbol. This is used to both name the new files and to find which lines of the original files should be written to each of the new files. For instance, if the first two symbol names returned by `unique` are A and AA, and the first two indices are 1 and 178, then we know that lines 1 to 177 should be written in the file `A.csv`. 
After all data from one symbol is written to its file, a new file is created for the next symbol until that entire file for that data date is complete. Each data date thereafter is appended to the last data in each symbol file, unless that file doesn't already exist, in which case a new file is created for the new symbol.

### 3.4 Writing New Files

New files are created using the `fopen` command with the permission set to ‘a+’ which means append to the information already in the file, if it exists. If the file doesn’t exist already, this will create the file. These new file names are appended to a folder name, ‘byTicker\’, therefore all of the new files are saved in the folder `byTicker`. After a file is created and/or opened, the data lines from the date file are written to the ticker file one line at a time. Each variable is indexed into and written here. The date variables are changed into numbers, typically referred to as Julian day, before they are written into the line in the new file. These date numbers allow for easy calculation of time between dates. They can also be changed back to easily read dates in original format.
4. Error Handling

A few exceptions may occur while processing this data. One exception handling technique is being used within the `formatDateToTicker` script. A `try/catch` loop is added around the `fprintf` command which writes each individual line to the new file. This `try/catch` loop will attempt to write the line of data into the new file. If there is an error in writing, then instead of Matlab terminating execution of the script, the `catch` statement will execute, and then the next iteration in the loop will be executed. In this case, the `catch` statement will display a sentence describing which line there was an error in writing, including the date and ticker that are currently being written. By using this `try/catch` pair, we were able to find some “trouble-maker” data such as the presence of a ‘/’ within a ticker symbol. This ‘/’ within a ticker symbol is then used in a file name and attempted to be saved as such. This creates errors with indexing into folders. With this find, we added a check for any ‘/’ within a ticker symbol and replaced it with a ‘1’ using the function `strrep` (string replace).

Another exception handling technique is the script `deleteAday`. `deleteAday` is used when an error occurs while running `formatDateToTicker`. In this case, the output files may include a partial set of data from one date file. If this occurs, the user should first figure out what caused the error, and edit the `formatDateToTicker` to handle this exception. Any data anomalies causing errors should be caught in the `try/catch` loop, and as a result, the user will see which file and line of data that caused the error. After the error is resolved, data will need to be deleted which was written from the file that caused the error. After this data is deleted, the script `formatDateToTicker` can be used again beginning with the day which was just deleted from the new files, since the error should have been fixed.
deleteAday uses a saved .mat file with variables containing information about what file formatDateToTicker is currently reading from and writing to. This information allows us to find what information has already been written to the new files.

deleteAday uses the daytodelete variable.

deleteAday works as follows:

1. First, it opens each of the new files one at a time, reads them with textscan, and closes them.

2. deleteAday then finds the line in the file where the daytodelete was being written.

3. A new file is then opened with the same name, but with permission ‘w’ which will discard any information already in the file.

4. deleteAday then begins writing the file again, writing up to but not including the daytodelete.

This procedure creates the same file set with data up to and not including the date during which the error occurred. The formatDateToTicker script can be altered to begin at the current date and move on again, appending data from there.
5. Data Processing Challenges

There are 3 substantial challenges that must be addressed when working with this data. The first issue is the large size of the files. One individual date file is usually between 30 and 40 megabytes in size. For example, the files in January, 2009 average around 37 megabytes in size, containing approximately 300,000 lines of data. This file size is also a major hindrance when trying to look through the data “by hand”, meaning opening a file in excel and looking through a file to find something.

The second issue is the mixture of strings and numbers in each line of data. This forces us to use the Matlab function `textscan` to read the original files. Given that the files are so large, the use of `textscan` on a date file takes up roughly 75% of the running time for parsing and rewriting that one file. The input files are closed after the information contained in them is stored as a variable. This variable is overwritten after each iteration, so there is always only one large variable saved at a time.

The third issue is runtime. Due to the size of these files and the functions needed to parse the data, using a dual core CPU with 3 GB RAM, processing one year of data can take 26 hours or more. This brings up issue of writing efficient code. The use of double-precision numbers was maintained even though downgrading to single precision would have cut about 7% of running time. Due to the organization of the data, the use of loops cannot be avoided. Since the variables change size for every iteration of every loop (for instance, the file sizes themselves are all different, therefore the variable C will be a different length for each file), memory cannot be pre-allocated for these. Parallel processing can greatly cut down on run time of this code. WPI has a few “super computers” which will allow for dividing the run time.
6. How To Use

The user file, *getData*, is shown below. This is a Matlab script with instructions included which describe general use of the script to obtain different data sets from the files that were created using the script *formatDateToTicker*. Note that the spacing is slightly different than in Matlab, and therefore there are the occasional lines that wrap here which would not in a Matlab editor window, and therefore would not be treated as a new line.

If a user wants to compile a file set from those created with *formatDateToTicker*, this script is what they will use. The other scripts are for initially setting up the whole file set, and generally will not be used unless it is desired to create a whole new file set with the data reformatted again, in which case the scripts *formatDateToTicker* and *deleteAday* would have to be altered.

This script includes instructions within its 5 sections for how and when to execute them. In the first section, a new directory is created. The name of the directory can be changed, but should be changed before it is created. The default new directory name is *workingFiles*. Section 2 is for specifying what date range the user would like the new data set to include. Section 3 should be executed without editing. This section creates the appropriate variables from the dates entered to be used in future sections. The dates entered are also checked for validity in this section. A valid date range is one where the first day occurs before the last day, and both are within the span between January 1, 2005, and December 31, 2009. If the date range entered is not a valid date range, an error is displayed.

Section 4 allows the user to enter a list of underlying symbols or use all symbols, as well as whether to use all dates or only the specified date range. These choices are executed by simply uncommenting specified lines within the sections. These lines have comments above
them describing what they do and why they should or should not be commented out. Nothing should be changed within this section except for entering the list of symbols.

Section 5 is similar to section 4 in that there is only 1 input from the user, and the rest is executed as-is with choices for search parameters (date range) to be commented or uncommented. This section allows a user to input a number of ‘largest’ files to write. ‘Largest’ is defined to be the files with the largest size or number of bytes. The largest files in the set will be the underlying assets for which the whole date range (January 2005 to December 2009) is included, and for each date a large number of options are listed.

This script is also intended to be written in such a way that the user may copy the file and create new types of searches for file sets based off of how these searches are built. For instance, section 4 could be altered to search for files which include date sets that begin in one date range and end during another specified date range.

% Objective:
% Get information from ticker or date files and write that information to
% new files saved to a new directory.

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Section 1
% First:
% Create a folder to store your new files in:

myDirectory = 'workingFiles';

% You can change the folder name from workingFiles, but be sure to keep the
% single quotes.
% Use the next line to create the directory for you:

mkdir(myDirectory);

% Now:
% What information do you want?
% You can fill that in next.

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Section 2
% TO GET ONLY A SPECIFIC DATE SET
% Beginning date:
bmonth = 1;     %Enter a month number between 1 and 12
bday = 7;       %Enter a day number between 1 and 31
byear = 2009;   %Enter a year between 2005 and 2009

% End date:
emonth = 1;     %Enter a month number between 1 and 12
eday = 8;       %Enter a day number between 1 and 31
eyear = 2009;   %Enter a year between 2005 and 2009

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Section 3
% Don't change anything in this section

month = num2str(bmonth);
day = num2str(bday);
year = num2str(byear);
if month < 10
    month = ['0',month];
end
if day < 10
    day = ['0',day];
end

firstday = datenum([month,'/',day,'/',year]);

month = num2str(emonth);
day = num2str(eday);
year = num2str(eyear);
if month < 10
    month = ['0',month];
end
if day < 10
    day = ['0',day];
end

lastday = datenum([month,'/',day,'/',year]);

if firstday>lastday || firstday<datenum(2005,01,01) || ... 
    lastday>datenum(2009,01,31);
    disp('Enter a valid date range between 01/01/2005 and 12/31/2009.');
end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Section 4

% 1. TO GET A SPECIFIC STOCK:
% Change the symbols in the cell array below. Enter underlying symbol in % all caps, with single quotes, eg. 'AAPL'.
% If you want option data for several different underlying symbols, % enter them all in the same array, separated by commas.
% The {} are needed here, since the elements are strings and of % different sizes.
symbols = {'A', 'BB', 'RTC'};
% 2. TO GET THE SPECIFIED DATE RANGE FOR ALL STOCKS:
% Uncomment the next 2 lines.
% files = dir('byTicker/*.csv');
% symbols = {files.name};

for ii = 1:length(symbols)
    currentfile = ['byTicker\', symbols{ii}];
    fid = fopen(currentfile);
    C = textscan(fid, '%s%f%s%s%f%f%f%f%f%f%f%f%f%f%f','Delimiter','
    fclose(fid);
    % Opens and reads one file at a time from your list. The file that is
    % currently being used is stored as a variable, C.
    alldaysinfile = C(8);
    [listofdays, dayindices] = unique(C(8), 'first');
    dayindices = [dayindices; length(C(8)) + 1];
    % listofdays is an array of numbers representing the days from which
    % this file has data.
    % dayindices is a list of the rows in the file where each day of data
    % begins.
    startfilehere = find(floor(listofdays) >= firstday, 1, 'first');
    if isempty(startfilehere);
        disp(['The day you want to start with is beyond the data dates '...
             'in the file ', currentfile, '. This file ',...
             'begins with ', datestr(floor(C(8)(1))), ' and ends with ',...
             datestr(floor(C(8)(end))), '. It will not be ',...
             'included in your data set.']);
        continue
    end
    endfilehere = find(floor(listofdays) <= lastday, 1, 'last') + 1;
    % this finds the INDEX of where the NEXT day would start
    % (the day after where you want to end the file)
    if isempty(endfilehere);
        disp(['The day you want to end with is before the data dates '...
             'in the file ', currentfile, '. This file ',...
             'begins with ', datestr(floor(C(8)(1))), ' and ends with ',...
             datestr(floor(C(8)(end))), '. It will not be ',...
             'included in your data set.']);
        continue
    end
    for cells = 1:19
        C(cells) = C(cells)(dayindices(startfilehere):dayindices(endfilehere)
        - 1);
    end
    if (floor(C(8)(end)) - floor(C(8)(1))) < (lastday - firstday);
        disp(['The file ', currentfile, ' does not include ',...
             'the whole requested set of dates. It begins with ',...
             datestr(floor(C(8)(1))), ' and ends with ',...
             datestr(floor(C(8)(end))), '. The date range ',...
             '...
'you requested is ', datestr(firstday), ' to ', ... datestr(lastday),'.']

end

fid=fopen([myDirectory,'\',symbols{ii}],'w'); % the new file
for jj=1:length(C{1})
  uS=C{1}{jj};
  price=C{2}{jj};
  exchange = C{3}{jj};
  optroot = C{4}{jj};
  optext = C{5}{jj};
  optype = C{6}{jj};
  expiry = C{7}{jj};
  dataday = C{8}{jj};
  % ** If you want only the whole day part of dataday (take out 04:00 % PM) then uncomment the next line:
  % dataday = floor(dataday);
  strike = C{9}{jj};
  oplast = C{10}{jj};
  bid = C{11}{jj};
  ask = C{12}{jj};
  volume = C{13}{jj};
  openinterest = C{14}{jj};
  iv = C{15}{jj};
  delta = C{16}{jj};
  gamma = C{17}{jj};
  theta = C{18}{jj};
  vega = C{19}{jj};

  fprintf(fid,'%s,%f,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s
   \n',...
    uS, price, exchange, optroot, optext, optype, ...
    expiry, dataday, strike, oplast, bid, ask, ...
    volume, openinterest, iv, delta, gamma, theta, vega);
end

fclose(fid);
end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
% Section 5
% TO GET X NUMBER OF LARGEST FILES:
% These are the ones with the most lines of data, meaning
% there is a longer time period of data for these underlyings, and that
% there are many options available on each day.

numberOfFiles = 3; % change this to the number of files you want.

files = dir('byTicker/*.csv');
symbols = {files.name};
if numberOfFiles < length(symbols)
  sizes = {files.bytes};
  filesizes = [sizes{::}];
}
orderedsizes = sort(filesizes,'descend');
smallest = orderedsizes(numberOfFiles);
largestNfiles = find(filesizes>=smallest);

for ii = 1:numberOfFiles
  currentfile = ['byTicker\', symbols{ii}];
  fid = fopen(currentfile);
  C = textscan(fid,'%s%f%s%s%f%f%f%f%f%f%f%f%f%f%f%f%f','Delimiter','
  fclose(fid);

  % If you want only the dates specified in Section 2, uncomment this next
  % section, keep in mind that if the specified date range is not in the
  % selected files, your chosen files will not be written, but that file
  % name will be printed to the screen and you can choose to simply copy
  % that file with whatever time period it includes.
  %~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
  %%% Section 5.b
  % alldaysinfile = C{8};
  % [listofdays, dayindices] = unique(C{8}, 'first');
  % dayindices = [dayindices; length(C{8}) + 1];
  % startfilehere = find(floor(listofdays)>=firstday,1,'first');
  % if isempty(startfilehere);
  %   disp(['The day you want to start with is beyond the data dates '...
  %   'in the file ', currentfile, '. This file ',...
  %   'begins with ', datestr(floor(C{8}(1))), ' and ends with ',...
  %   datestr(floor(C{8}(end))), '. It will not be '...
  %   'included in your data set.']);
  %   continue
  % end
  %
  % endfilehere = find(floor(listofdays)<=lastday,1,'last')+1;
  % this finds the INDEX of where the NEXT day would start
  % (the day after where you want to end the file)
  % if isempty(endfilehere);
  %   disp(['The day you want to end with is before the data dates '...
  %   'in the file ', currentfile, '. This file ',...
  %   'begins with ', datestr(floor(C{8}(1))), ' and ends with ',...
  %   datestr(floor(C{8}(end))), '. It will not be '...
  %   'included in your data set.']);
  %   continue
  % end
  % for cells = 1:19
  %   C{cells} =
  % end

  % if (floor(C{8}(end)) - floor(C{8}(1))) < (lastday - firstday);
  % disp(['The file ', currentfile,' does not include '...
  % 'the whole requested set of dates. ',
  %   currentfile,'....
  % ' begins with ', datestr(floor(C{8}(1))), ' and ends with ','...
fid=fopen([myDirectory, '\', symbols{ii}],'w'); % the new file
for jj=1:length(C{1})
    uS=C{1}{jj};
    price=C{2}{jj};
    exchange = C{3}{jj};
    optroot = C{4}{jj};
    optext = C{5}{jj};
    optype = C{6}{jj};
    expiry = C{7}{jj};
    dataday = C{8}{jj};
    % ** If you want only the whole day part of dataday (take out 04:00
    % PM) then uncomment the next line:
    % dataday = floor(dattday);
    strike = C{9}{jj};
    oplast = C{10}{jj};
    bid = C{11}{jj};
    ask = C{12}{jj};
    volume = C{13}{jj};
    openinterest = C{14}{jj};
    iv = C{15}{jj};
    delta = C{16}{jj};
    gamma = C{17}{jj};
    theta = C{18}{jj};
    vega = C{19}{jj};

    fprintf(fid,'%s,%f,%s,%s,%s,%s,%s,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
\r\n',...
    uS, price, exchange, optroot, optext, optype, ...
    expiry, dataday, strike, oplast, bid, ask, ...
    volume, openinterest, iv, delta, gamma, theta, vega);
end
fclose(fid);

else
    disp(['Enter a smaller number of files to retrieve. There are only
',...
     num2str(length(symbols)), ' files in this folder.']);
end

%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~%
7. Conclusion

We have created 3 m-files which parse large file sets, reformat the data, and create new files within new directories. The first m-file is a script, `formatDateToTicker`, which reads information from files which are arranged by date, in a folder called `byDate`, reformats the data and writes it in new files arranged by underlying symbol, saved in a new folder, `byTicker`. The second m-file is a script whose main purpose is to recover a definite starting point (file) from which to resume reading data after an error is encountered. The third m-file is written for a user. This m-file creates a new directory to be named by the user, default named `workingFiles`. After this new directory is created, all new files that are created through searches by the user are saved into this directory. In the end, there are 3 folders, `byDate`, `byTicker`, and `workingFiles`. The only files that should ever change after the initial file set creation into `byTicker`, are the files that are saved into the `workingFiles` directory.

The original files which are stored in `byDate` are arranged by date. Each file is one day of data for all tickers. The data in these files is ordered by ticker alphabetically, then by option information such as strike price. The new set of files which are stored in `byTicker` is arranged by underlying symbol, or ticker. There is one file per ticker which includes all data for that ticker from all files in `byDate`. Each ticker file is sorted by data date first, then by option information. Any new files in `workingFiles` are arranged in the same manner as the files in `byTicker`, but may include less data days depending on the search criteria used.

These three folders and the three m-files are all contained in one folder, `optionsData`. In addition, we have run this for the entire data set in `byDate` and created the complete data set stored in `byTicker`. 
Option chain data sets produced by this software will be in a more accessible format for use in mathematical modeling and financial mathematics research.


A. Appendix

A.1 formatDateToTicker.m

%% COMMENTS:

clear; clc; close all;

for year = 2005:2009
    year = num2str(year);
    for month = 1:1
        if month<10
            month=strcat(num2str(0),num2str(month));
            % turn single digit to '0'
        else month = num2str(month);
        end
        for day = 1:3
            if day<10
                day=strcat(num2str(0),num2str(day));
            else day = num2str(day);
            end
            currentdate = [month,'/', day,'/', year];
            % Creating a variable currentdate for use later in the
            % deleteAday script in case this script hangs up on a day
            fid = fopen(strcat('byDate\options_',year,month,day,'.csv'));
            if fid == -1;
                % if the file doesn't exist, then move on to next date
                continue;
            end
            C =
            textscan(fid,'%s%f%s%s%s%s%s%f%f%f%f%f%f%f%f%f%f%f%f','Delimiter',',');
            fclose(fid);
            tickers=C{1};
            % This file is a nested cell array.
            % The first cell (column) contains ticker symbols
            [unigtik, utikindices, other] = unique(tickers, 'first');
            % unigtik is unique tickers
            % utikindices is the 'first' index where each new ticker occurs
            utikindices=[utikindices; size(C{1},1)+1];
            % We need the last index plus one so we can copy
            % them all later. If there were another ticker
            % symbol added on, it would start at index
            % size(C{1},1)+1, so to get all of the rows for the
            % last ticker we have, we need to know where the
            % next one would start.
for ii=1:length(uniqtik)
    newfile = strcat(uniqtik{ii,:}, '.csv');
    % creating a new file name
    newfile = strrep(newfile, '/', '1');
    % Change filenames that include / to include 1 instead
    % because saving a filename with / will not work properly.
    newfileext = strcat('byTicker\',newfile);
    % in the folder one level up, called "byTicker"
    currentwritingfile = newfileext;
    % This is saved in the file dayinfo.mat with currentdate
    % for use in deleteAday script.
    fid = fopen(newfileext,'a+');
    % 'a+' means append to what's already there. You can
    % open, read, and write by adding to the file. This
    % doesn't replace the file that's already there by
    % that name. If the file doesn't already exist it
    % is created here.
    for jj=utikindices(ii):utikindices(ii+1)-1
        uS=C{1}{jj};
        price=C{2}{jj};
        exchange = C{3}{jj};
        optroot = C{4}{jj};
        optext = C{5}{jj};
        optype = C{6}{jj};
        expiry = datenum(C{7}(jj),'mm/dd/yyyy');
        dataday = datenum(C{8}(jj),'mm/dd/yyyy HH:MM:SS PM');
        strike = C{9}(jj);
        oplast = C{10}(jj);
        bid = C{11}(jj);
        ask = C{12}(jj);
        volume = C{13}(jj);
        openinterest = C{14}(jj);
        iv = C{15}(jj);
        delta = C{16}(jj);
        gamma = C{17}(jj);
        theta = C{18}(jj);
        vega = C{19}(jj);
        try
            fprintf(fid,'%s,%f,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s
\r\n',...
                uS, price, exchange, optroot, optext, optype, ...
                expiry, dataday, strike, oplast, bid, ask, ...
                volume, openinterest, iv, delta, gamma, theta,
                vega);
        catch ME
            fprintf(['Unable to print line ', num2str(jj), ' in file ', ...'
                'options_',year,month,day,...'
                '.csv to file ', newfileext]);
            % using [] here instead of strcat allows for
            % obligatory character in case filename...
        end
    end
end
% whitespace between items.
% strcat is used in other places in this script to
% avoid unwanted whitespace in file names.
end
end
fclose(fid);

daytodelete = dataday;
save currentdayandticker daytodelete currentwritingfile
% This saves as a .mat file and can easily be used in
% deleteAday script. Double clicking this mat file will
% create % last dataday and currentwritingfile variables in the
% workspace.
end
end
end

A.2 deleteAday.m

% deletes all lines of data (within ticker files) which were written from
% the last date file

files = dir('byTicker\*.csv')
filenames = {files.name}
for ii = 1:length(filenames)
    currentfile = strcat('byTicker,', filenames{ii});
    fid = fopen(currentfile);
    C = textscan(fid,'%s%f%s%s%f%f%f%f%f%f%f%f%f%f%f,f','Delimiter',',');
    fclose(fid);
    alldaysinfile = C{8};
    endfilehere = find(abs(alldaysinfile-daytodelete)<.1,1,'first')-1;
    for cells = 1:19
        C{cells} = C{cells}(1:endfilehere);
    end

    fid = fopen(currentfile,'w');
    % Open or create new file for writing. Discard existing contents, if any.
    for jj=1:length(C{1})
        uS=C{1}{jj};
        price=C{2}{jj};
        exchange = C{3}{jj};
        optroot = C{4}{jj};
        optext = C{5}{jj};
        optype = C{6}{jj};
        expiry = C{7}{jj};
        dataday = C{8}{jj};
strike = C{9}(jj);
oplast = C{10}(jj);
bid = C{11}(jj);
ask = C{12}(jj);
volume = C{13}(jj);
openinterest = C{14}(jj);
iv = C{15}(jj);
delta = C{16}(jj);
gamma = C{17}(jj);
theta = C{18}(jj);
vega = C{19}(jj);

fprintf(fid,'%s,%f,%s,%s,%s,%s,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
\r\n',...
    uS, price, exchange, optroot, optext, optype, ...
    expiry, dataday, strike, oplast, bid, ask, ...
    volume, openinterest, iv, delta, gamma, theta, vega);
end
fclose(fid);
end

A.3 Option Extension Codes

Since options have underlying symbols, option symbols, option roots, extensions, exchanges, etc, the name can be confusing to interpret. Below is a chart which describes how to read an option symbol. This was directly copied from (How do you Find an Option Ticker Symbol? | Wall Street Survivor University:, 2009).

Option symbols are typically the ticker symbol for the stock (3 letters or less), a space, a letter code for the Expiration Month, and then a letter code indicating the Strike Price. The Stock Options names are written in the following manner:

SYMBOL MP
Symbol = The Option Root Symbol
M = Expiration Month
P = Strike Price

<table>
<thead>
<tr>
<th>Expiration Month</th>
<th>Call Code</th>
<th>Put Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
<td>A</td>
<td>M</td>
</tr>
<tr>
<td>February</td>
<td>B</td>
<td>N</td>
</tr>
<tr>
<td>March</td>
<td>C</td>
<td>O</td>
</tr>
<tr>
<td>April</td>
<td>D</td>
<td>P</td>
</tr>
<tr>
<td>May</td>
<td>E</td>
<td>Q</td>
</tr>
<tr>
<td>June</td>
<td>F</td>
<td>R</td>
</tr>
<tr>
<td>Code</td>
<td>Strike Prices</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>5  105 205 305 405</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>10 110 210 310 410</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>15 115 215 315 415</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>20 120 220 320 420</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>25 125 225 325 425</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>30 130 230 330 430</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>35 135 235 335 435</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>40 140 240 340 440</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>45 145 245 345 445</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>50 150 250 350 450</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>55 155 255 355 455</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>60 160 260 360 460</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>65 165 265 365 465</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>70 170 270 370 470</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>75 175 275 375 475</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>80 180 280 380 480</td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td>85 185 285 385 485</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>90 190 290 390 490</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>95 195 295 395 495</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>100 200 300 400 500</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>12.5</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>17.5</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>22.5</td>
<td></td>
</tr>
</tbody>
</table>
A.4 Exchange Variable

What are the values in the exchange column?
In the beginning of collecting and distributing historical option data, we included a column called “exchange”. The initial intend was that we might want to list quotes from individual exchanges. This never happened and the column was always marked with an asterisk (*), or sometimes letter G (for good).

Then within our first year we decided that we needed to do about bad options, and non-standard options.

Non-standard options are option contracts that are NOT 100 shares of stock, but also include other amounts of stock, other stocks, or cash dividends. A formal listing of these is maintained by the Options Clearing Corporation, and they call them the Equity Special Settlement reports. Non-standard options are marked in the exchange column as letter N.

Bad options prices are determined by a simple formula and only for in the money options. If a call has a strike of 50 and the stock is trading at 71, then the ask price of the call option must be at least 21.00. If it is not, then it becomes an arbitrage value, which basically means “free money”, because you could buy the option and immediately make a profit. The same goes for the puts. If the strike of the put is 80.00, then the ask price must be at least 9.00. If it is lower than this, then again, it is an arbitrage situation.

The bad option price could either mean that the underlying stock price is wrong, or the option price is wrong. Bad option prices are marked in the exchange column as a question mark (?). This arbitrage test is only done for in the money options, and out of the money prices are not tested.

Summary

(*) and (G) – should be good if they are in the money. Asterisk means it is a composite price, including all option exchanges.

(N) – this is a non-standard, or special settlement option and should only be used if you are looking for this exact option symbol, and not used in back testing.

(?) – this denotes that it is an in the money option and that the price is suspect.
B. Bibliography

