Final Project Write-up

MBA 614 – Spreadsheet Automation and Modeling

Jason S. Merrill

10 Apr 2010

Executive Summary

Description of the Business

Business school students and business professionals are expected, depending on their position, to keep abreast of financial market values and how those values change over time. The paper version of the Wall Street Journal has a few values of interest at the top of the front page, but they are not comprehensive. Many people rely on the Wall Street Journal website and email alerts and can't be bothered to hunt down values buried within the paper version. Additionally, there are many websites where one can find financial market values, but the user must typically open a browser, navigate to the appropriate website, and then click through many different screens in order to find the values in which he or she may be interested. I have come up with a solution that will aggregate the financial market values of most interest, graph the values that change the most, and then send an email containing the data and graphs to the user based on the user's content and frequency preferences.

Overview of the System

The system establishes connections with various pages on the Bloomberg.com website. Once these connections are established, updated data are downloaded from the websites to Excel. This data is then copied over to a historical values database contained within the Excel workbook. Several graphs are produced that are based on the historical data. A list of users, including email addresses and system preferences, is also included in the Excel workbook. Depending on the day of the week, emails are then sent to the users. Each email is custom tailored to only include the values and graphs that the user desires. Additionally, the user can specify the frequency with which the emails are sent. The two options are 'daily' or 'weekly'. If the 'daily' option is specified, the user will receive an email on each weekday of the week. If the 'weekly' option is specified, the user will receive one email on Monday of each week. Users of the service will not interact directly with the system. As such, next steps include the following:

- Putting the system on a server
- Automating the execution of the program
- Developing a web interface so that users can sign up for the service automatically, without the developer having to manually add each user

Implementation Documentation

The program can be run from the first worksheet of the workbook. The first worksheet includes a button to run the program, the financial market values from the most recent download, the number of users in the email list, and the number of web query used to get each value. The nineteen values deemed the most important to track are listed in following snapshot of the first worksheet:

Market Values

Jason Merrill

	Value	Web Query #
Dow	10,997.35	1
S&P 500	1,194.37	1
NASDAQ	2,454.05	1
Gold	1,161.90	5
Oil	84.92	2
Fed Funds Rate	0.20%	3
Fed Reserve Target Rate	0.25%	3
US Unemployment Rate	9.7%	3
1-Month LIBOR	0.25%	3
3-Month LIBOR	0.30%	3
3-Month US Treasury	0.15%	4
6-Month US Treasury	0.23%	4
12-Month US Treasury	0.44%	4
2-Year US Treasury	1.06%	4
3-Year US Treasury	1.69%	4
5-Year US Treasury	2.62%	4
7-Year US Treasury	3.34%	4
10-Year US Treasury	3.88%	4
30-Year US Treasury	4.74%	4

Get Market Values

Number of Users: 2

The 'Get Market Values' button is linked to the getMarketValues( ) subroutine. Once the button is clicked, five data connections are established with the Bloomberg.com website. The five data connections are from four websites, with one of the websites having two data connections. The four websites are the following:

- http://www.bloomberg.com/markets/stocks/wei.html
- http://www.bloomberg.com/markets/commodities/cfutures.html
- http://www.bloomberg.com/markets/rates/keyrates.html
- http://www.bloomberg.com/markets/rates/index.html

Testing revealed that the tables on their homepage do rotate around, which would have required using the 'find' function in order to find the correct data. However, the value references were typically duplicated many times on the homepage, so I decided to use the other websites referenced above. On these pages, the table numbers remained static, which permitted the data connection to reference specific tables, without having to use the 'find' function on a download of the entire webpage. The following snapshots show what the downloaded data look like for each data connection:

webQuery1

INDEX	VALUE	CHANGE	%CHANGE	TIME
DOW JONES INDUS. AVG	10,997.35	70.28	0.64%	9-Apr
S&P 500 INDEX	1,194.37	7.93	0.67%	9-Apr
NASDAQ COMPOSITE INDEX	2,454.05	17.24	0.71%	9-Apr
S&P/TSX COMPOSITE INDEX	12,176.84	63.31	0.52%	9-Apr
MEXICO BOLSA INDEX	33,840.85	273.5	0.81%	9-Apr
BRAZIL BOVESPA INDEX	71,417.27	-367.51	-0.51%	9-Apr

webQuery2

	PRICE	CHANGE	%CHANGE	TIME
BRENT CRUDE FUTR (USD/bbl.)	84.83	0.02	0.02	9-Apr
GAS OIL FUT (ICE) (USD/MT)	707.5	0.5	0.07	9-Apr
GASOLINE RBOB FUT (USd/gal.)	228.93	-0.9	-0.39	9-Apr
HEATING OIL FUTR (USd/gal.)	222.6	-0.22	-0.1	9-Apr
NATURAL GAS FUTR (USD/MMBtu)	4.07	0	0	17:25
WTI CRUDE FUTURE (USD/bbl.)	84.92	-0.47	-0.55	9-Apr

webQuery3

	CURRENT	1 MONTH PRIOR	3 MONTH PRIOR	6 MONTH PRIOR	1 YEAR PRIOR
Fed Funds Rate	0.2	0.15	0.11	0.13	0.15
Fed Reserve Target Rate	0.25	0.25	0.25	0.25	0.25
Prime Rate	3.25	3.25	3.25	3.25	3.25
US Unemployment Rate	9.7	9.7	10	9.8	8.6
1-Month Libor	0.25	0.23	0.23	0.24	0.45
3-Month Libor	0.3	0.26	0.25	0.28	1.13

webQuery4

	COUPON	MATURITY DATE	CURRENT PRICE/YIELD	PRICE/YIELD CHANGE	TIME
3-Month	0	7/8/2010	0.14 / .15	1	9-Apr
6-Month	0	10/7/2010	0.22 / .23	1	9-Apr
12-Month	0	4/7/2011	0.43 / .44	1	9-Apr
2-Year	1	3/31/2012	99-28½ / 1.06	0-00+ / -.008	9-Apr
3-Year	1.75	4/15/2013	100-06 / 1.69	0-00 / .000	9-Apr
5-Year	2.5	3/31/2015	99-13½ / 2.62	0-01½ / -.010	9-Apr
7-Year	3.25	3/31/2017	99-15 / 3.34	0-02+ / -.013	9-Apr
10-Year	3.625	2/15/2020	97-28+ / 3.88	0-02 / -.008	9-Apr
30-Year	4.625	2/15/2040	98-05½ / 4.74	0-06+ / -.013	9-Apr

Inflation Indexed Treasury

	COUPON	MATURITY DATE	CURRENT PRICE/YIELD	PRICE/YIELD CHANGE	TIME
5-Year	1.25	4/15/2014	103-18 / .35	0-00 / -.009	9-Apr
10-Year	1.375	1/15/2020	98-15 / 1.54	0-00 / -.000	9-Apr
20-Year	2.5	1/15/2029	106-19 / 2.07	-333.3333333	9-Apr
30-Year	2.125	2/15/2040	100-27 / 2.09	0-09 / -.014	9-Apr

webQuery5

	PRICE	CHANGE	%CHANGE	TIME
GOLD 100 OZ FUTR (USD/t oz.)	1161.9	9	0.78	9-Apr
SILVER FUTURE (USD/t oz.)	18.351	0.224	1.24	9-Apr

The treasury yields in the fourth web query had to be parsed, since the values are listed with the price, separated by a slash. This parsing was performed while the web query data was being copied over to the formatted value list located on the first worksheet.

After all of the relevant values are copied to the first worksheet, the values are copied over, with a timestamp, to the historical values database. The following is a snapshot of the historical values database:

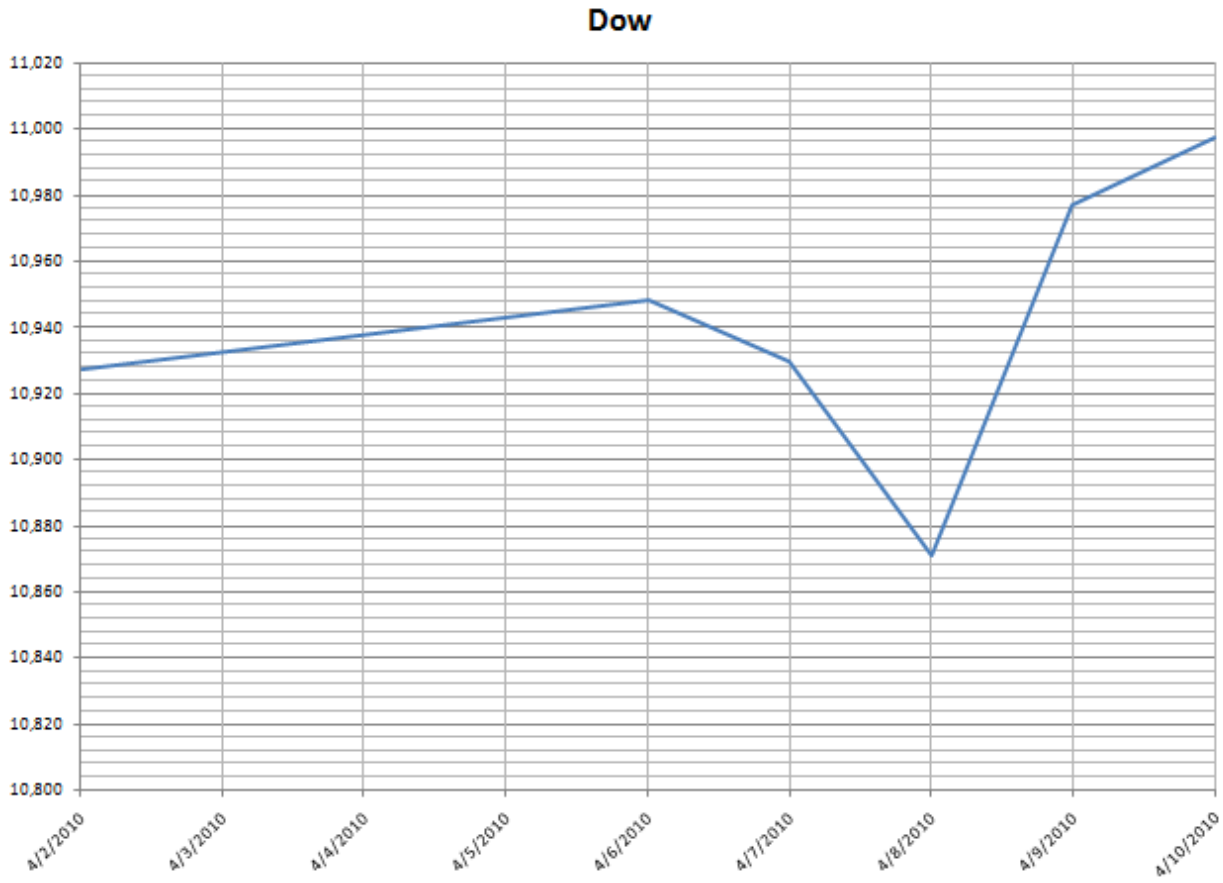
Date Stamp	Dow	S&P 500	NASDAQ	Gold	Oil	Fed Fun
4/10/2010 1:17 PM	10,997.35	1,194.37	2,454.05	1,161.90	84.92	0.2
4/9/2010 9:59 AM	10,976.95	1,191.10	2,443.93	1,163.40	84.85	0.2
4/8/2010 9:36 AM	10,870.69	1,179.44	2,423.10	1,151.60	85.05	0.2
4/7/2010 10:45 AM	10,929.33	1,185.78	2,435.13	1,151.10	86.46	0.2
4/6/2010 8:57 AM	10,948.08	1,185.93	2,424.36	1,137.20	86.87	0.2
4/2/2010 1:35 PM	10,927.07	1,178.10	2,402.58	1,126.10	84.87	0.2

The program finds the end of the table and adds the new entry. It then re-sorts the table so that the most recent entry is at the top.

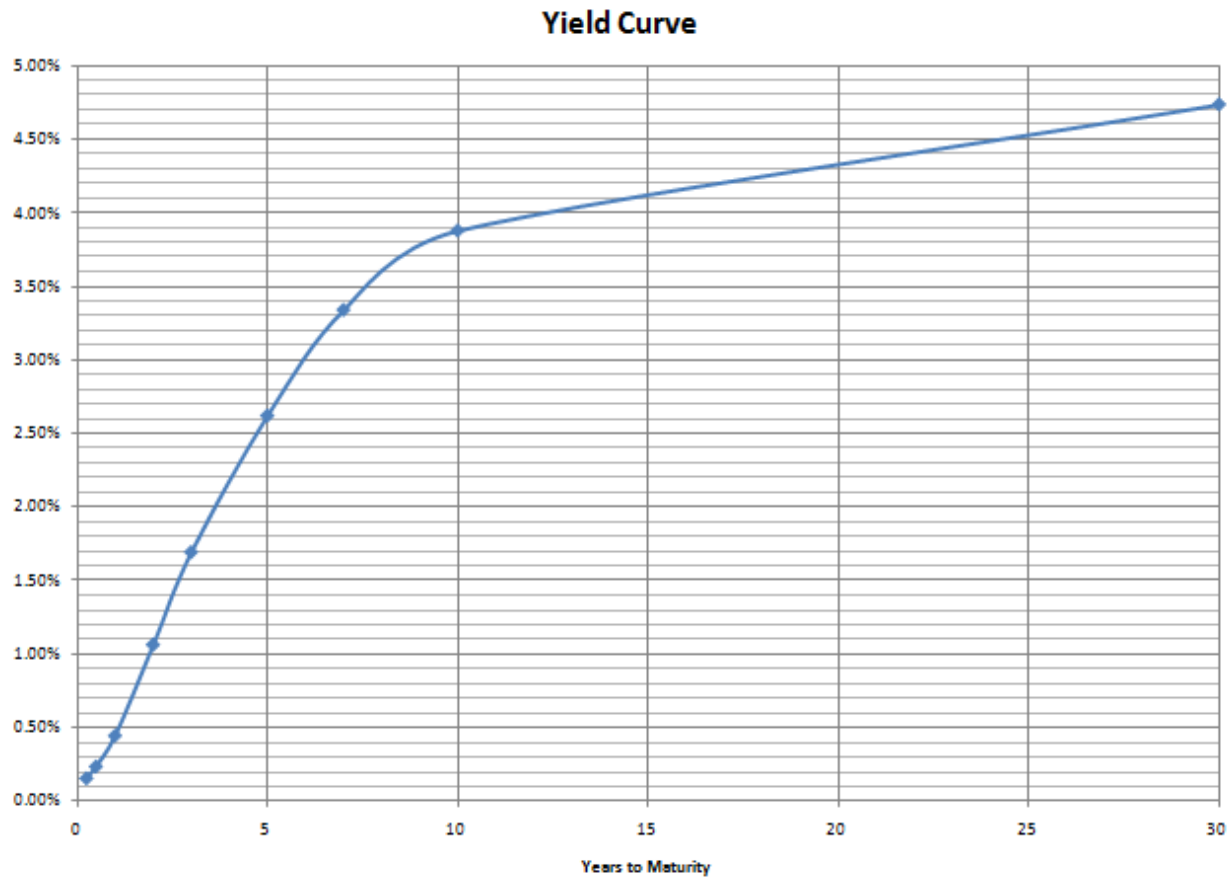
The primary use of this historical values database is in producing graphs of changes in the values over time. Graphs are produced for the following financial market values:

- Dow
- S&P 500
- NASDAQ
- Gold
- Oil
- 3-Month US Treasury
- 6-Month US Treasury
- 12-Month US Treasury
- 2-Year US Treasury
- 3-Year US Treasury
- 5-Year US Treasury
- 7-Year US Treasury
- 10-Year US Treasury
- 30-Year US Treasury

A graph depicting the yield curve, which is based on all the US treasury values, is also produced. All the graphs for the values listed above pretty much look the same, so I will only include one of them as an example as follows:



I will also include a snapshot of the yield curve graph below, since it looks a bit different than all the other graphs:



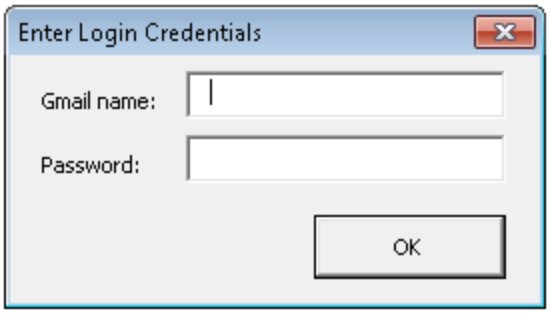
Note that graphs for the following financial market values are not included:

- Fed Funds Rate
- Fed Reserve Target Rate
- US Unemployment Rate
- 1-Month LIBOR
- 3-Month LIBOR

Graphs for these values were not included because they tend to vary much less than the others over time.

The program then exports all of these graphs to GIF picture files. Each picture file is named according to what is graphed, and the paths to all of the picture files are stored in an array for use in attaching the picture files to emails later in the program. At this point, the sendMessages( ) subroutine is called.

A user form is then presented to the developer so that a username and password can be entered that correspond to the Gmail account from which the emails will be sent. A snapshot of this user form is included below:

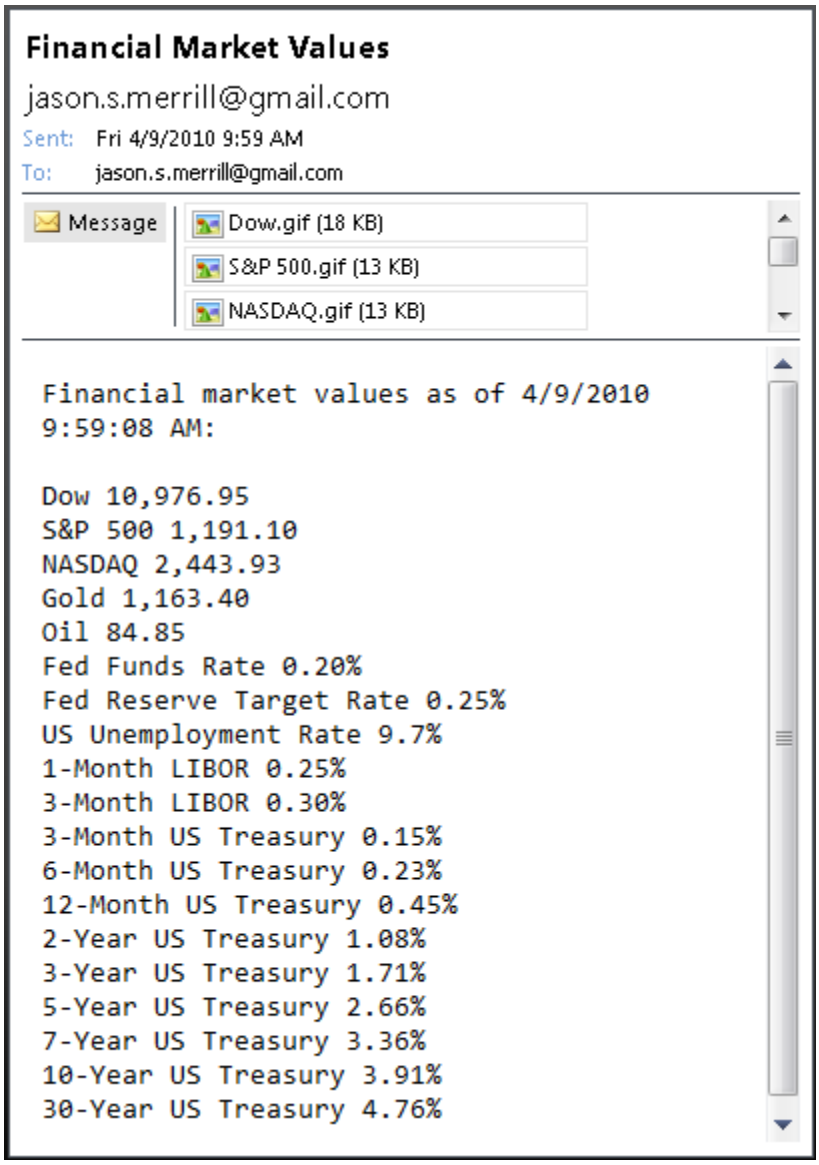


At this point, the text of the email for each user is crafted based on the user's preferences. The user can decide exactly which values should be included in the email. This process is accomplished through the use of a string array, where each string in the array is the body of an email for one user. After all of these strings are created, the sendGmail( ) function is called based on the email frequency preferences of the user. A snapshot of the email list and user preferences is below:

Email Address	Dow	S&P 500	NASDAQ	Gold
jason.s.merrill@gmail.com	TRUE	TRUE	TRUE	TRUE
jason.s.merrill.00@alum.dartmouth.org	TRUE	FALSE	TRUE	FALSE

The two options for email frequency are daily and weekly. If the user wishes to only receive a weekly email, then the email will only be sent on Mondays. If the user wishes to receive daily emails, then an email will be sent every weekday, Monday through Friday (U.S. financial markets are closed on Saturdays and Sundays).

After the sendGmail( ) function is called, the graph picture files are attached to the email according to the user's preferences. Only the graphs which the user specifies will be attached to the email. After all of the desired graphs are attached to the email, the email is sent. An example email from an Outlook email preview can be seen below:



This subroutine is performed for all of the users in the email list, although as mentioned before the users will only get emails according to their desired email frequency.

As a final note, the program was completed successfully and runs well. The program consists of 615 lines of code.

Learning and Conceptual Difficulties

This project provided a rich learning experience for me. The biggest learning points came from the topics I struggled with, which are the following:

- When defining a string array, I tried to dimension it using a number value from my spreadsheet. While doing this I kept on getting an error that said "Compile Error: Constant Expression Required." I was getting super frustrated with this and researched the book and the web for a solution. The solution I ended up using, which may not be the

best, was one that I found on the web. Basically, the solution was to dim the string array without sizing it, and then redim it with the appropriate size, using the following code:

```
Dim m() As String
ReDim m(numUsers - 1) As String
```

- I actually had some message boxes that would pop up and tell me if an email was sent successfully or not. These really helped during the development of the program, but in the end I decided to get rid of them. My reasoning was that I intend to put the program on a server and have it run automatically, so there wouldn't be a user staring at the program while it's running to see if an email was sent successfully or not. Additionally, as the number of users in the email list gets large, it doesn't make sense to have a message box popping up for each individual email.
- Another problem I came across was during testing of the program. When I was testing the functionality that determines the user's email frequency preferences, I was checking the various inboxes and not seeing the messages when I knew they should be there. After much code tweaking and head scratching, I thought of checking my spam folder, and there they were. Gmail had decided to start classifying the messages as spam, perhaps due to the similarity and frequency of the test emails. This served as a good reminder to me that I should remind the users to check their spam folder if they feel that they're not receiving the emails like they should.
- Another learning point for me was the difference in the code if you are creating a data connection for the first time, or if you are simply refreshing an existing data connection. At first I was creating a new data connection each time I ran the program. This caused several problems, which were solved when I just created the data connections once offline, and then adding the code to refresh these connections to the actual program.