Documentation for the `fredclass` Python Module

Brian C. Jenkins[*]
Department of Economics
University of California, Irvine

October 13, 2014

Abstract

`fredclass` is a Python module for retrieving and working with data from Federal Reserve Economic Data (FRED). The module makes it easy to download specific data series and provides a set of tools for transforming the data in order to construct plots and perform statistical analysis. The `fredclass` module is useful for anyone doing empirical research using the data available from FRED and for anyone, e.g. economics teachers and journalists, that will benefit from having an efficient and flexible way to access FRED with Python.

[*]Email: bcjenkin@uci.edu This project is ongoing and I welcome all feedback. I have no affiliation with the Federal Reserve Bank of St. Louis or with any other component of the Federal Reserve System.

Contents

1 Introduction

Federal Reserve Economic Data or FRED is a rich database maintained but the Federal Reserve Bank of St. Louis.[1] `fredclass` is a Python module that simplifies the process of downloading and manipulating data from FRED. The module offers a streamlined way of retrieving data directly from the FRED website and provides a number of tools to assist with management of the series obtained. This module is particularly useful for creating Python programs that integrate data retrieval with statistical analysis. The module is also well-suited for use with programs that will update figures and tables as new data are released.

2 Preliminaries

Currently, a `fredclass` installation is unavailable. To use the module, download `fredclass.py` from either my Github repository and place it in the same directory as your Python code. To use `fredclass`, you must have the following Python modules installed:

- `matplotlib`
- `numpy`
- `scipy`
- `statsmodels`

2.1 Initialization

To create a `fred` instance, you must have the unique Series ID for the data that you wish to retrieve. Generally, you will find this by searching the FRED website for the data series by name. For example, use the FRED site to find that GDPC96 is the Series ID for real GDP of the US (quarterly frequency, seasonally adjusted). Create a `fred` instance based on this data:

```
>>>from fredclass import fred
>>>gdp = fred('GDPC96')
```

Now the `fred` instance `gdp` is available for manipulation.

Each `fred` instance is initialized with a set of attributes that describe the characteristics of the series retrieved from FRED. These attributes are:

`title`: Title of the data.

`source`: Original source of the data.

`season`: Indicates whether the data has been seasonally adjusted.

`freq`: Equals 365, 52, 12, 4, or 1 to indicate daily, monthly, quarterly, or annual frequency.

`units`: Units of the data.

`daterange`: Date range of data.

`updated`: Date on which data was last updated.

`id`: The unique FRED identification code for the series.

[1]Site: http://research.stlouisfed.org/fred2/

`data`: A list containing the data.

`dates`: A list containing date strings in yyy-mm-dd format.

`datenums`: A list containing date numbers to use with the `date time` module.

In terms of the GDP example, find the precise title of data series used by FRED:

```
>>> gdp.title
'Real Gross Domestic Product, 3 Decimal'
```

Or learn that the original source for the data series was the BEA:

```
>>> gdp.source
'U.S. Department of Commerce: Bureau of Economic Analysis'
```

Or find out that quarterly real GDP data goes back to January 1947 and that, as of the time this document was written, the most recent observation was for the quarter beginning April 2014:

```
>>> gdp.daterange
'Range: 1947-01-01 to 2014-04-01'
```

Several of the methods described in the next section will alter the values of the attributes created upon initialization. Furthermore, some methods – the filtering methods in particular – add a couple of new attributes as necessary.

2.2 Methods

Each `fred` instance has the following methods:

`pc(log=True)`

Replaces the `data` attribute with the percentage change of the data series from the pervious time period. If `log=True`, then percentage changeis computed as the difference between the log of the current value and the log of the one period lag value:

$$100 \times \left[\log(X_t) - \log(X_{t-1})\right]. \tag{1}$$

If `log=False`, then the percentage change is computed in the standard way:

$$100 \times \left[(X_t - X_{t-1})/X_{t-1}\right]. \tag{2}$$

Also, method drops the first elements of the `date` and `datenums` attributes to account for the shorter observation window. Other attributes modified: `units` and `title`.

`apc(log=True)`

Replaces the `data` attribute with the *annual* percentage change of the data series from the pervious time period. If `log=True`, then percentage change is computed as the difference between the log of the current value and the log of the one period lag value:

$$100 \times \left[\log(X_t) - \log(X_{t-k})\right], \tag{3}$$

where $k = 1, 4, 12$, or 365 is the annual frequency of the data series. If `log=False`, then the percentage change is computed in the standard way:

$$100 \times \left[(X_t - X_{t-k})/X_{t-k}\right]. \tag{4}$$

Also, method drops the first k elements of the `date` and `datenums` attributes to account for the shorter observation window. Other attributes modified: `units` and `title`.

`ma(length)`

Replaces the `data` attribute with the two-sided moving average with window of size `length`. If `length` is not even, the window size is rounded down to the nearest even number. The `date` and `datenums` attributes are adjusted from both ends to account for the reduced observation window. Other attributes modified: `daterange` and `title`.

`replace(new)`

Replaces the `data` attribute with a the list `new`. Prints a warning if the new series has a different length from the original series.

`recent(T=10)`

Constrains data series to the most recent `T` years. The `date` and `datenums` attributes are adjusted from both ends to account for the reduced observation window. Other attributes modified: `daterange`

`window(win)`

Constrains data series to a specified window. `win` is a list specifying the minimum and maximum dates to be included. Dates must be in yyyy-mm-dd format. For example, to restrict observations to those between January 1, 1980 to December 31, 1999:

`win=['1980-01-01','1999-12-31'].` (5)

Other attributes modified: `daterange`

`log()`

Replaces the `data` attribute with the log of the data series. Other attributes modified: `title`

`bpfilter()`

Computes the bandpass (Baxter-King) filter of the series using the `statsmodels` module. Instead of replacing the modifying the original attributes, this method creates three new attributes:

`bpcycle`: cyclical component of series

`bpdates`: dates corresponding to bp filtered data

`bpdatenums`: date numbers corresponding to bp filtered data

Method displays a warning if the original series is not quarterly.

`hpfilter()`

Computes the Hodrick-Prescott filter of the series using the `statsmodels` module. Instead of replacing the modifying the original `data` attribute, this method creates two new attributes:

`hpcycle`: cyclical component of series

`hptrend`: trend component of series

Method displays a warning if the original series is not quarterly.

`cffilter()`

Computes the Christiano-Fitzgerald filter of the series using the `statsmodels` module. Instead of replacing the modifying the original `data` attribute, this method creates two new attributes:

`cfcycle`: cyclical component of series

`cftrend`: trend component of series

Method displays a warning if the original series is not quarterly.

`lintrend()`

Computes the linear trend of the series. Instead of replacing the modifying the original `data` attribute, this method creates two new attributes:

`lincycle`: cyclical component of series

`lintrend`: trend component of series

Method displays a warning if the original series is not quarterly.

`firstdiff()`

Computes the first-difference of the series. Instead of replacing the modifying the original `data` attribute, this method creates four new attributes:

`diffcycle`: cyclical component of series

`difftrend`: trend component of series

`diffdates`: shorter date sequence

`diffdatenums`: shorter date numbers

`diffdata`: shorter data series

`monthtoquarter(method='AVG')`

Converts series with monthly frequency to quarterly frequency. Modifies `data`, `dates`, `datenums`, `t` attributes. If `method='AVG'`, then the value for a quarter is the average over each month of the corresponding three month period. If `method='SUM'`, then the value for a quarter is the sum over the months of the corresponding three month period. And if `method='END'`, then the value for a quarter is value of the last month of the corresponding three month period.

`quartertoannual(method='AVG')`

Converts series with quarterly frequency to annual frequency. Modifies `data`, `dates`, `datenums`, `t` attributes. If `method='AVG'`, then the value for a year is the average over each quarter of the corresponding four quarter period. If `method='SUM'`, then the value for a year is the sum over the quarters of the corresponding four quarter period. And if `method='END'`, then the value for a year is value of the last quarter of the corresponding four quarter period.

`monthtoannual(method='AVG')`

Converts series with monthly frequency to annual frequency. Modifies `data`, `dates`, `datenums`, `t` attributes. If `method='AVG'`, then the value for a year is the average over each month of the corresponding twelve month period. If `method='SUM'`, then the value for a year is the sum over the months of the corresponding twelve month period. And if `method='END'`, then the value for a year is value of the last month of the corresponding twelve month period.

`percapita(pop_type=1)`

Replaces the `data` attribute with the values divided by the US population using one of two definitions of population:

`pop_type=1`: total population US population.

`pop_type=2`: civilian noninstitutional population is defined as persons 16 years of age and older.

Other attributes modified: `title` and `units`

`recessions()`

Creates gray recession bars for plots. Should be used after a plot has been made but before either (1) a new plot is created or (2) a show command is issued.

2.3 Other Functions

The `fredclass` module is equipped with two functions that are occasionally useful for working with `fred` objects:

`quickplot(x,year_mult=10,show=True,recess=False,save=False,name='file',width=2)`

This function produces a plot of the data for a given `fred` instance `x`. `year_mult` specifies the multiple in which years on the horizontal axis should be formed. `show=True` means that a new plot window should be created. `recess=False` means that no recession bars should be plotted and `recess=True` means the opposite. If `save=True`, the plot is saved in .png format. `width` specifies the width of the plotted line.

`window_equalize(fred_list)`

Equalizes the data windows for a list of `fred` instances by finding the largest date range for which there is data available from each series in `fred_list`.

3 Examples

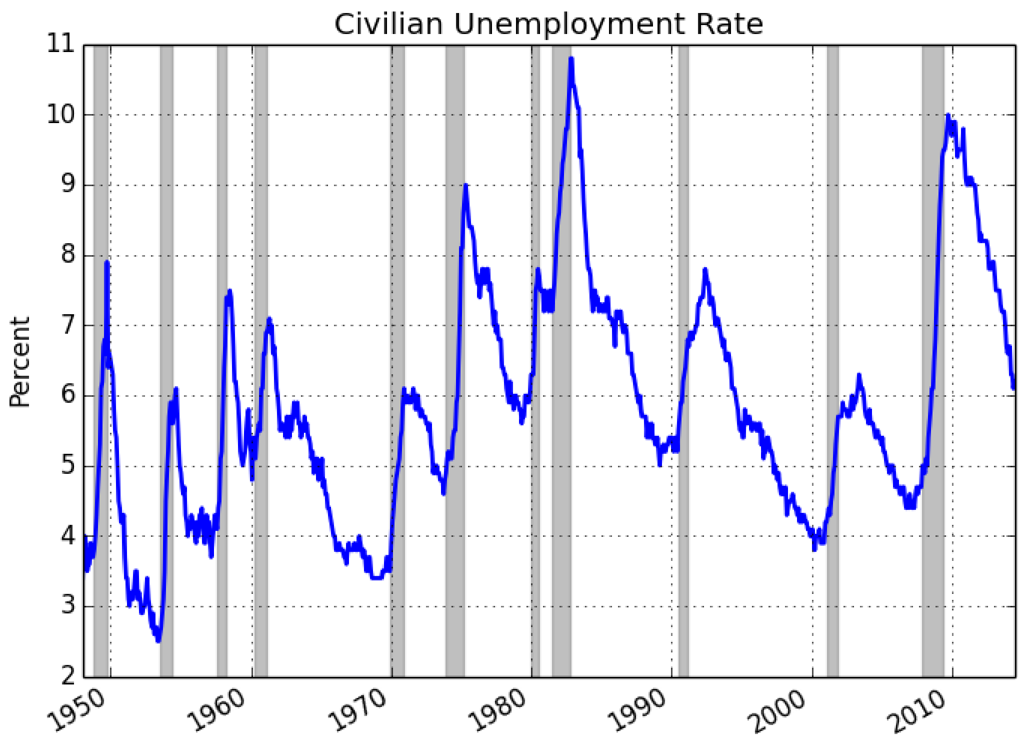
3.1 Example 1: Plotting with the `quickplot` function

To begin, let's construct a plot of all currently available data for the US unemployment rate. To do this, we run a script that contains the following:

```
from fredclass import fred, quickplot
unemp = fred('UNRATE')
quickplot(unemp,recess=True,save=True,name='fredclass_example1')
```

This program will display a plot and will save the plot to a file called `fredclass_example1.png` in the working directory. The output is depicted in figure 1. The quick plot function has the advantage of being able to generate plots on the fly, but most of the time you will want to mange plotting directly.

Figure 1: A plot of the US unemployment rate using the `quickplot` function.



3.2 Example 2: Plotting Multiple Series

Next we will construct a more elaborate plot using the module. The objective is to create a four-panel plot containing the rate of real GDP growth, CPI inflation, the unemployment rate, and the 3-month T-bill rate. Since the available ranges for these series are different, we will make use of the `window_equalize` function to make sure that we plot all four series over the same date range. First, the import statements:

```
from __future__ import division
import matplotlib.pyplot as plt
from fredclass import fred, window_equalize
import matplotlib.dates as dts
from datetime import date
```

Next, download data from FRED:

```
gdp = fred('GDPC96')
cpi = fred('CPIAUCSL')
unemp=fred('UNRATE')
tbill=fred('TB3MS')
```

Then replace divide gdp data by 1,000 to convert units from billions of dollars to trillions of dollars.

```
gdp.replace([y/1000 for y in gdp.data ])
```

Use the `apc()` method to compute annual growth in real GDP and the CPI. Note that this method computes percentage change *from one year previous.*

```
gdp.apc()
cpi.apc()
```

Now use the `window_equalize` function to set the date ranges to the smallest range common to all four series.

```
series = [gdp,cpi,unemp,tbill]
window_equalize(series)
```

Finally, plot the data. Take note of the plot syntax. Use the function `plot_date()` to plot the data of each `fred`object against its date numbers.

```
fig = plt.figure()
years10 = dts.YearLocator(10)

# plot real gdp growth
ax1 = fig.add_subplot(221)
ax1.plot_date(gdp.datenums,gdp.data,'b-',lw = 3)
ax1.xaxis.set_major_locator(years10)
ax1.yaxis.set_major_formatter(y_format)
ax1.set_ylabel('Trillions of 2009 $')
fig.autofmt_xdate()
ax1.grid(True)
gdp.recessions()
ax1.set_title('Real GDP')

# plot cpi inflation
ax2 = fig.add_subplot(222)
ax2.plot_date(cpi.datenums,cpi.data,'b-',lw = 3)
ax2.xaxis.set_major_locator(years10)
ax2.set_ylabel('Percent')
fig.autofmt_xdate()
ax2.grid(True)
cpi.recessions()
ax2.set_title('CPI Inflation')
```

```
# plot unemployment rate
ax3 = fig.add_subplot(223)
ax3.plot_date(unemp.datenums,unemp.data,'b-',lw = 3)
ax3.xaxis.set_major_locator(years10)
ax3.set_ylabel('Percent')
fig.autofmt_xdate()
ax3.grid(True)
unemp.recessions()
ax3.set_title('Unemployment Rate')

# plot 3-mo T-bill rate
ax4 = fig.add_subplot(224)
ax4.plot_date(tbill.datenums,tbill.data,'b-',lw = 3)
ax4.xaxis.set_major_locator(years10)
ax4.set_ylabel('Percent')
fig.autofmt_xdate()
ax4.grid(True)
tbill.recessions()
ax4.set_title('3-mo T-bill Rate')

plt.savefig('fredclass_example2.png',bbox_inches='tight')
plt.show()
```

Figure 2 contains the output of this program.

3.3 Example 3: Filtering

In the final example, we will use the Hodrick-Prescott (HP) and the Bandpass (BP) filters to isolate the business cycle components of real US GDP. Specifically, we will filter log per capita real GDP so that the resulting series may be interpreted as the log-deviations from trend.

```
from __future__ import division   import matplotlib.pyplot as plt   from fredclass
import fred, window_equalize import matplotlib.dates as dts
```

Then download real GDP data from FRED and convert into units of log thousands of dollars per person.

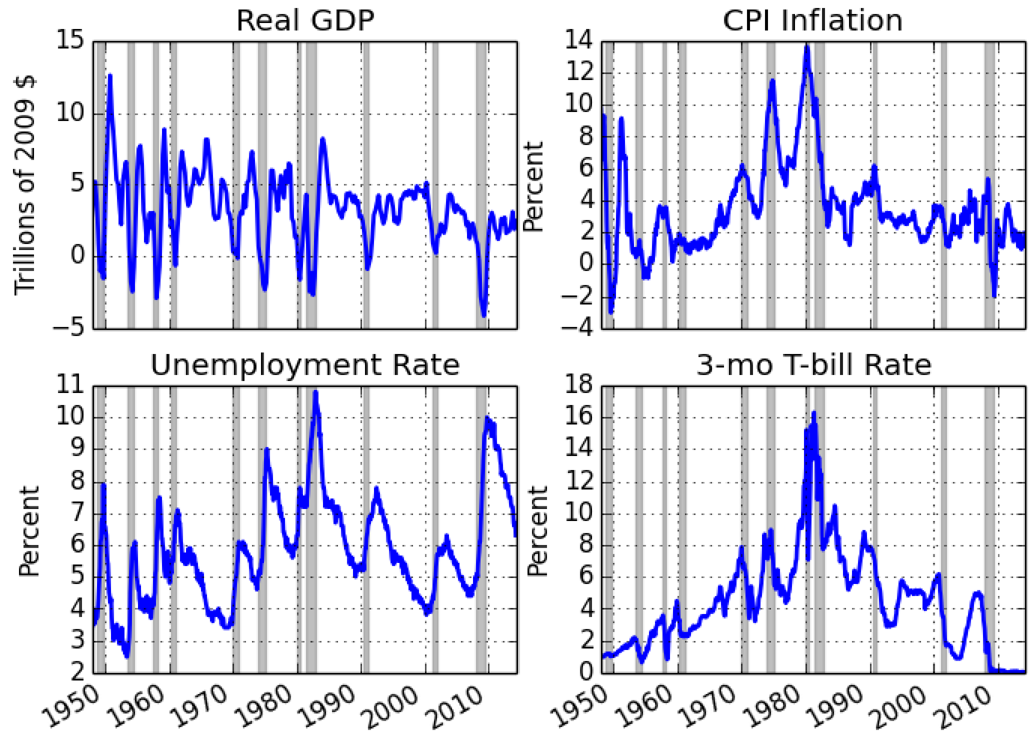
```
gdp = fred('GDPC96')
gdp.percapita()
gdp.replace([y*1000 for y in gdp.data ])
gdp.log()
```

Apply the filters. Note that the data frequency must be quarterly. Currently, the module is not capable of properly filtering data that has a different frequency.

```
gdp.bpfilter()
gdp.hpfilter()
```

Finally, plot the data.

Figure 2: Plots of real GDP growth, CPI inflation, the unemployment rate and the 3-moth T-bill rate.



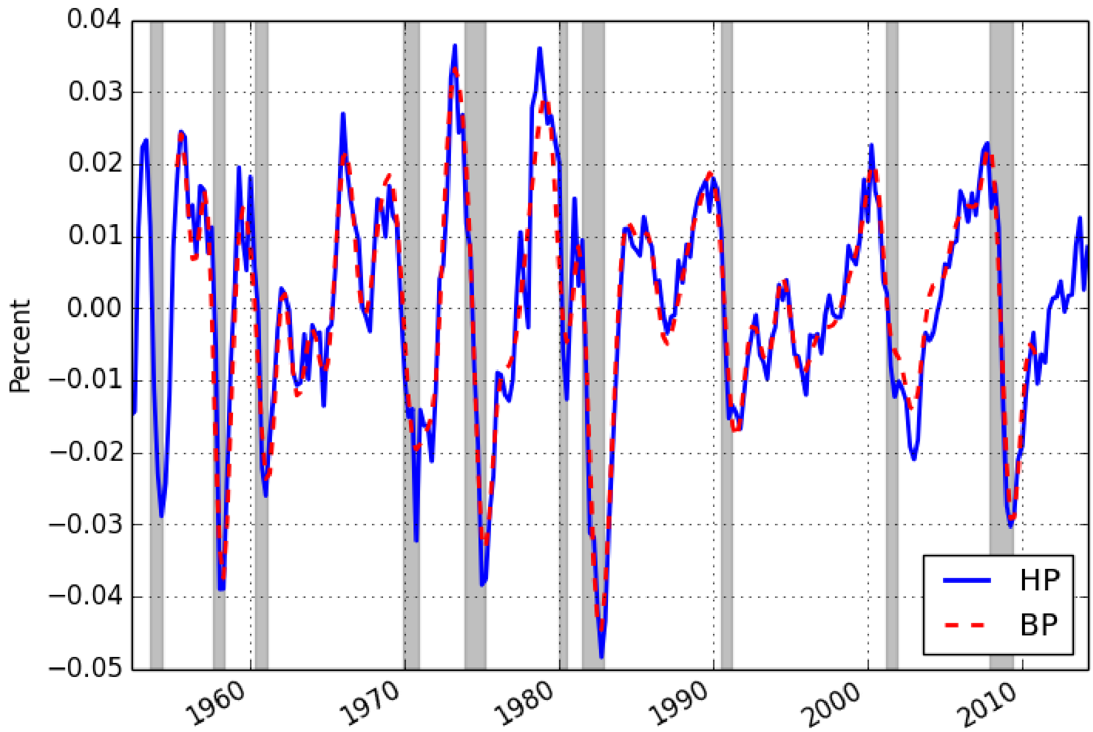

```
fig = plt.figure()
years10 = dts.YearLocator(10)

{ax1 = fig.add_subplot(111)
ax1.plot_date(gdp.datenums,gdp.hpcycle,'b-',lw = 2)
ax1.plot_date(gdp.bpdatenums,gdp.bpcycle,'r--',lw = 2)
ax1.xaxis.set_major_locator(years10)
ax1.set_ylabel('Percent')
fig.autofmt_xdate()
ax1.grid(True)
gdp.recessions()
ax1.legend(['HP','BP'],loc='lower right')

plt.savefig('fredclass_example3.png',bbox_inches='tight')
plt.show()
```

The output of the program is contained in Figure 3.

Figure 3: HP and BP filtered log real GDP per capita.



4 Comments

The examples above emphasize the ease with which the `fredclass` module is used for downloading data and making plots. For another example in this spirit, check out my animation of the US Treasury yield curve. Like all other `fredclass`-related materials, the code for this animation is also available from my Github repository or from my personal website.

The `fredclass` module has uses beyond simple data visualization. A researcher doing statistical analysis using series available from FRED can use the module to create a single Python program – or set of programs – that retrieves data from FRED, performs the analysis, and exports to LaTeX tables. Such functionality will reduce the researchers time spent managing data files and will allow the researcher to easily update results as new data become available.