fredpy Documentation

Release 2.0.2

Brian C. Jenkins

Oct 03, 2016

CONTENTS

1 Installation **2**

2 Contents: **3**

2.1 `fredpy.series` class 3

2.2 Additional `fredpy` Functions 7

2.3 `fredpy` Examples 8

3 Indices and tables **23**

Index **24**

`fredpy` is a Python package for retrieving and working with data from Federal Reserve Economic Data (FRED). The package makes it easy to download specific data series and provides a set of tools for transforming the data in order to construct plots and do statistical analysis. The `fredpy` package is useful for anyone doing empirical research using data from FRED and for anyone, e.g. economics teachers, students, and journalists, that will benefit from having an efficient and flexible way to access FRED with Python. `fredpy` is compatible with Python 2 and 3.

CHAPTER ONE

INSTALLATION

Install `fredpy` from PyPI with the shell command:

```
pip install fredpy
```

Or download the source here: https://github.com/letsgoexploring/fredpy-package/raw/gh-pages/dist/fredpy-2.0.2.tar.gz

CHAPTER

TWO

CONTENTS:

2.1 `fredpy.series` class

class `fredpy.series`(*series_id=None*)

Creates an instance of a *fredpy.series* instance that stores information about the specified data series from FRED with the unique series ID code given by `series_id`.

Parameters **series_id** (*string*) – unique FRED series ID. If `series_id` equals None, an empy *fredpy.series* instance is created.

Attributes:

data(numpy ndarray) – data values.

daterange(string) – specifies the dates of the first and last observations.

dates(list) – list of date strings in YYYY-MM-DD format.

datetimes(numpy ndarray) – array containing observation dates formatted as `datetime.datetime` instances.

freq(string) – data frequency. 'Daily', 'Weekly', 'Monthly', 'Quarterly', or 'Annual'.

idCode(string) – unique FRED series ID code.

season(string) – specifies whether the data has been seasonally adjusted.

source(string) – original source of the data.

t(integer) – number corresponding to frequency: 365 for daily, 52 for weekly, 12 for monthly, 4 for quarterly, and 1 for annual.

title(string) – title of the data series.

units(string) – units of the data series.

updated(string) – date series was last updated.

Methods:

apc(*log=True*, *method='backward'*)

Computes the percentage change in the data over one year.

Parameters

- **log** (*bool*) – If True, computes the percentage change as $100 \cdot \log(x_t/x_{t-1})$. If False, compute the percentage change as $100 \cdot (x_t/x_{t-1} - 1)$.
- **method** (*string*) – If 'backward', compute percentage change from the previous period. If 'forward', compute percentage change from current to subsequent period.

Returns *fredpy.series*

bpfilter(*low=6*, *high=32*, *K=12*)

Computes the bandpass (Baxter-King) filter of the data. Returns a list of two `fredpy.series` instances containing the cyclical and trend components of the data:

[new_series_cycle, new_series_trend]

Parameters

- **low** (*integer*) – Minimum period for oscillations. Select 24 for monthly data, 6 for quarterly data (default), and 3 for annual data.
- **high** (*integer*) – Maximum period for oscillations. Select 84 for monthly data, 32 for quarterly data (default), and 8 for annual data.
- **K** (*integer*) – Lead-lag length of the filter. Select, 84 for monthly data, 12 for for quarterly data (default), and 1.5 for annual data.

Returns`list` of two `fredpy.series` instances

Note: In computing the bandpass filter, K observations are lost from each end of the original series so the attributes *dates*, *datetimes*, and *data* are 2K elements shorter than their counterparts in the original series.

cffilter(*low=6*, *high=32*)

Computes the Christiano-Fitzgerald filter of the data. Returns a list of two `fredpy.series` instances containing the cyclical and trend components of the data:

[new_series_cycle, new_series_trend]

Parameters

- **low** (*integer*) – Minimum period for oscillations. Select 6 for quarterly data (default) and 1.5 for annual data.
- **high** (*integer*) – Maximum period for oscillations. Select 32 for quarterly data (default) and 8 for annual data.

Returns`list` of two `fredpy.series` instances

copy()

Returns a copy of the `fredpy.series` instance.

ParametersNone

Returns`fredpy.series`

divide(*series2*)

Divides the data from the current fredpy series by the data from `series2`.

Parameters**series2** (`fredpy.series`) – A `fredpy.series` instance.

Returns`fredpy.series`

firstdiff()

Computes the first difference filter of original series. Returns a list of two `fredpy.series` instances containing the cyclical and trend components of the data:

[new_series_cycle, new_series_trend]

Parameters

Returns`list` of two `fredpy.series` instances

Note: In computing the first difference filter, the first observation from the original series is lost so the attributes *dates*, *datetimes*, and *data* are 1 element shorter than their counterparts in the original series.

hpfilter(*lamb=1600*)

Computes the Hodrick-Prescott filter of the data. Returns a list of two `fredpy.series` instances containing the cyclical and trend components of the data:

[new_series_cycle, new_series_trend]

Parameters **lamb** (*integer*) – The Hodrick-Prescott smoothing parameter. Select 129600 for monthly data, 1600 for quarterly data (default), and 6.25 for annual data.
Returns `list` of two *fredpy.series* instances

lintrend()
Computes a simple linear filter of the data using OLS. Returns a list of two *fredpy.series* instances containing the cyclical and trend components of the data:
[new_series_cycle, new_series_trend]

Parameters
Returns `list` of two *fredpy.series* instances

log()
Computes the natural log of the data.
Parameters
Returns *fredpy.series*

ma1side(*length*)
Computes a one-sided moving average with window equal to `length`.
Parameters **length** (*integer*) – `length` of the one-sided moving average.
Returns *fredpy.series*

ma2side(*length*)
Computes a two-sided moving average with window equal to 2 times `length`.
Parameters **length** (*integer*) – half of `length` of the two-sided moving average. For example, if `length = 12`, then the moving average will contain 24 the 12 periods before and the 12 periods after each observation.
Returns *fredpy.series*

minus(*series2*)
Subtracts the data from `series2` from the data from the current fredpy series.
Parameters **series2** (`fredpy.series`) – A `fredpy.series` instance.
Returns *fredpy.series*

monthtoannual(*method='average'*)
Converts monthly data to annual data.
Parameters **method** (*string*) – If 'average', use the average values over each twelve month interval (default), if 'sum,' use the sum of the values over each twelve month interval, and if 'end' use the values at the end of each twelve month interval.
Returns *fredpy.series*

monthtoquarter(*method='average'*)
Converts monthly data to quarterly data.
Parameters **method** (*string*) – If 'average', use the average values over each three month interval (default), if 'sum,' use the sum of the values over each three month interval, and if 'end' use the values at the end of each three month interval.
Returns *fredpy.series*

pc(*log=True*, *method='backward'*, *annualized=False*)
Computes the percentage change in the data from the preceding period.
Parameters
- **log** (*bool*) – If True, computes the percentage change as $100 \cdot \log(x_t/x_{t-1})$. If False, compute the percentage change as $100 \cdot (x_t/x_{t-1} - 1)$.
- **method** (*string*) – If 'backward', compute percentage change from the previous period. If 'forward', compute percentage change from current to subsequent period.

- **annualized** (*bool*) – If True, percentage change is annualized by multipying the simple percentage change by the number of data observations per year. E.g., if the data are monthly, then the annualized percentage change is $4 \cdot 100 \cdot \log(x_t/x_{t-1})$.

Returns *fredpy.series*

percapita (*total_pop=True*)

Transforms the data into per capita terms (US) by dividing by one of two measures of the total population.

Parameters total_pop (*string*) – If `total_pop == True`, then use the toal population (Default). Else, use Civilian noninstitutional population defined as persons 16 years of age and older.

Returns *fredpy.series*

plus (*series2*)

Adds the data from the current fredpy series to the data from `series2`.

Parameters series2 (`fredpy.series`) – A `fredpy.series` instance.

Returns *fredpy.series*

quartertoannual (*method='average'*)

Converts quarterly data to annual data.

Parameters method (*string*) – If 'average', use the average values over each four quarter interval (default), if 'sum,' use the sum of the values over each four quarter interval, and if 'end' use the values at the end of each four quarter interval.

Returns *fredpy.series*

recent (*N*)

Restrict the data to the most recent N observations.

Parameters N (*integer*) – Number of periods to include in the data window.

Returns *fredpy.series*

recessions (*color='0.5', alpha = 0.5*)

Creates 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.

Parameters

- **color** (*string*) – Color of the bars. Default: '0.5'.
- **alpha** (*float*) – Transparency of the recession bars. Must be between 0 and 1. Default: 0.5.

Returns

times (*series2*)

Multiplies the data from the current fredpy series with the data from `series2`.

Parameters series2 (`fredpy.series`) – A `fredpy.series` instance.

Returns *fredpy.series*

window (*win*)

Restricts the data to the most recent N observations.

Parameters win (*list*) – is an ordered pair: `win = [win_min, win_max]` where `win_min` is the date of the minimum date desired and `win_max` is the date of the maximum date. Date strings must be entered in either 'yyyy-mm-dd' or 'mm-dd-yyyy' format.

Returns *fredpy.series*

2.2 Additional `fredpy` Functions

`fredpy.date_times`(*date_strings*)

Converts a list of date strings in yyyy-mm-dd format to datetime.

Parameters **date_strings** (*list*) – a list of date strings formated as: 'yyyy-mm-dd'.

Returns `numpy.ndarray`

`fredpy.divide`(*series1*, *series2*)

Divides the data from `series1` by the data from `series2`.

Parameters

- **series1** (`fredpy.series`) – A `fredpy.series` object.
- **series2** (`fredpy.series`) – A `fredpy.series` object.

Returns *fredpy.series*

`fredpy.plus`(*series1*, *series2*)

Adds the data from `series1` to the data from `series2`.

Parameters

- **series1** (`fredpy.series`) – A `fredpy.series` object.
- **series2** (`fredpy.series`) – A `fredpy.series` object.

Returns *fredpy.series*

`fredpy.quickplot`(*fred_series*, *year_mult=10*, *show=True*, *recess=False*, *save=False*, *filename='file'*, *linewidth=2*, *alpha = 0.75*)

Create a plot of a FRED data series

Parameters

- **fred_series** (`fredpy.series`) – A `fredpy.series` object.
- **year_mult** (*integer*) – Interval between year ticks on the x-axis. Default: 10.
- **show** (*bool*) – Show the plot? Default: True.
- **recess** (*bool*) – Show recession bars in plot? Default: False.
- **save** (*bool*) – Save the image to file? Default: False.
- **filename** (*string*) – Name of file to which image is saved *without an extension*. Default: `'file'`.
- **linewidth** (*float*) – Width of plotted line. Default: 2.
- **alpha** (*float*) – Transparency of the recession bars. Must be between 0 and 1. Default: 0.7.

Returns

`fredpy.minus`(*series1*, *series2*)

Subtracts the data from `series2` from the data from `series1`.

Parameters

- **series1** (`fredpy.series`) – A `fredpy.series` object.
- **series2** (`fredpy.series`) – A `fredpy.series` object.

Returns *fredpy.series*

`fredpy.times` (*series1*, *series2*)

Multiplies the data from `series1` with the data from `series2`.

Parameters

- **series1** (`fredpy.series`) – A `fredpy.series` object.
- **series2** (`fredpy.series`) – A `fredpy.series` object.

Returns *fredpy.series*

`fredpy.window_equalize` (*series_list*)

Adjusts the date windows for a collection of fredpy.series objects to the smallest common window.

Parameters **series_list** (*list*) – A list of *fredpy.series* objects

Returns

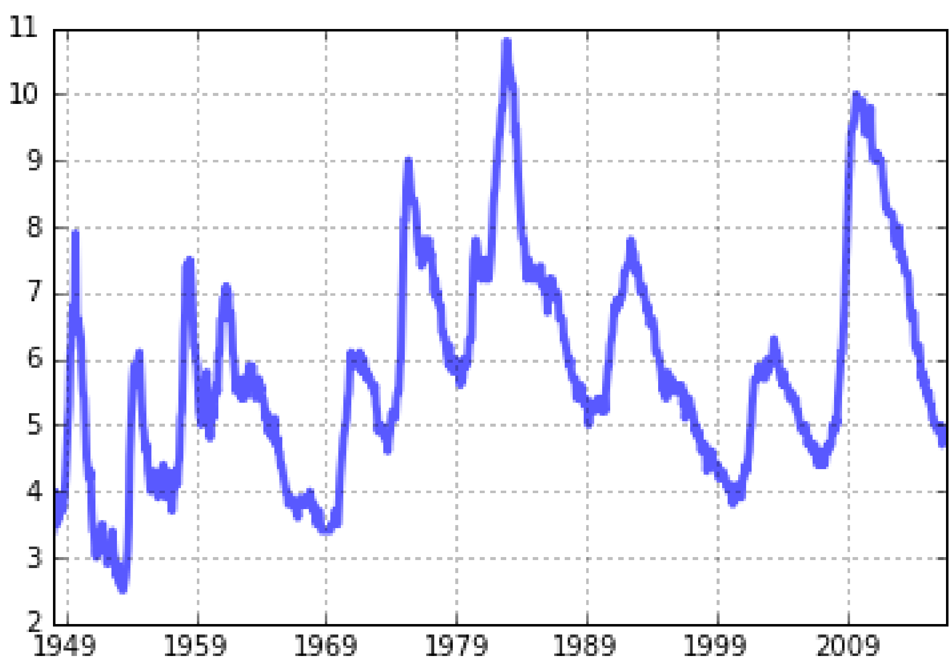
2.3 `fredpy` Examples

```
In [1]: import pandas as pd
        import numpy as np
        from fredpy import series
        import matplotlib.pyplot as plt
        %matplotlib inline
```

2.3.1 Preliminary example

Downloading and plotting unemployment rate data for the US is easy with `fredpy`:

```
In [2]: u = series('UNRATE')
        plt.plot_date(u.datetimes,u.data,'-',lw=3,alpha = 0.65)
        plt.grid()
```



2.3.2 A closer look at `fredpy` using real GDP data

Use `fredpy` to download real GDP data. The FRED page for real GDP: https://fred.stlouisfed.org/series/GDPC1. Note that the series ID - `GDPC1` - is in the URL and is visible in several places on the page.

The data in text format is located at: https://fred.stlouisfed.org/data/gdpc1.txt. When supplied with the series ID `GDPC1`, `fredpy` visits the the URL for the text-formatted data, reads the information on the page, and stores the data as attributes of a *`fredpy.series`* instance.

```
In [3]: # Download quarterly real GDP data using `fredpy`. Save the data in a variable cal
        gdp = series('gdpc1')

        # Note that gdp is an instance of the `fredpy.series` class
        print(type(gdp))
```

```
<class 'fredpy.series'>
```

Attributes

A *`fredpy.series`* instance stores information about a FRED series in 12 attribues:

- **data:** (numpy ndarray) – data values.
- **daterange:** (string) – specifies the dates of the first and last observations.
- **dates:** (list) – list of date strings in YYYY-MM-DD format.
- **datetimes:** (numpy ndarray) – array containing observation dates formatted as datetime.datetime instances.
- **freq:** (string) – data frequency. 'Daily', 'Weekly', 'Monthly', 'Quarterly', or 'Annual'.
- **idCode:** (string) – unique FRED series ID code.
- **season:** (string) – specifies whether the data has been seasonally adjusted.
- **source:** (string) – original source of the data.
- **t:** (integer) – number corresponding to frequency: 365 for daily, 52 for weekly, 12 for monthly, 4 for quarterly, and 1 for annual.
- **title:** (string) – title of the data series.
- **units:** (string) – units of the data series.
- **updated:** (string) – date series was last updated.

```
In [4]: # Print the title, the units, the frequency, the date range, and the source of the
        print(gdp.title)
        print(gdp.units)
        print(gdp.freq)
        print(gdp.daterange)
        print(gdp.source)
```

```
Real Gross Domestic Product
Billions of Chained 2009 Dollars
Quarterly
Range: 1947-01-01 to 2016-04-01
US. Bureau of Economic Analysis
```

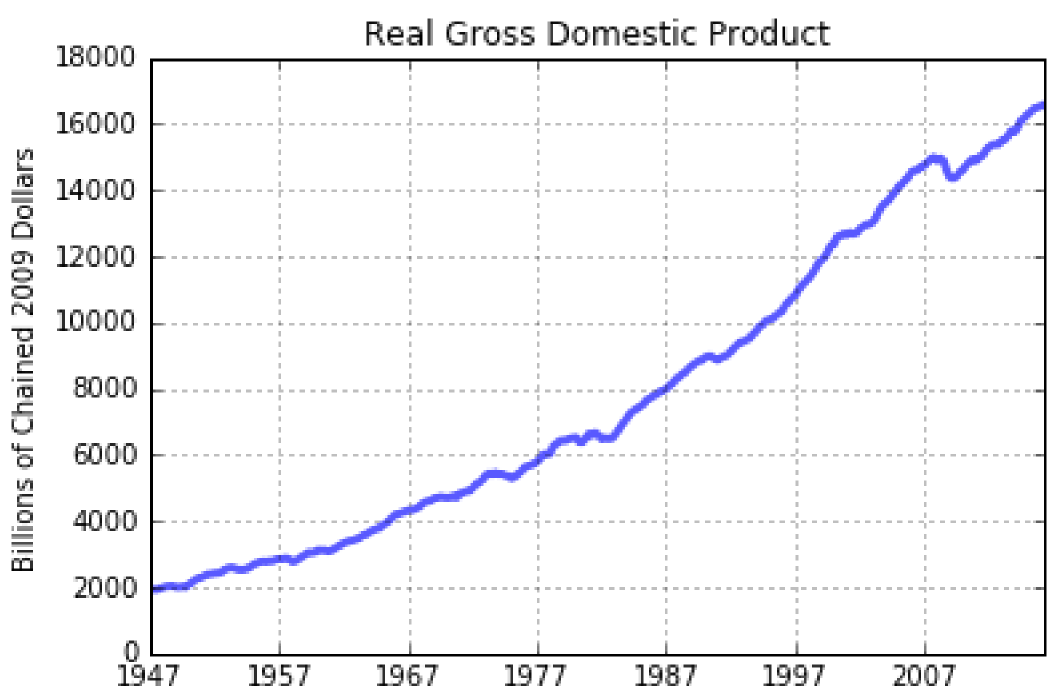
```
In [5]: # Print the last 4 values of the gdp data
        print(gdp.data[-4:],'\n')
```

```
        # Print the last 4 values of the gdp series dates
        print(gdp.dates[-4:],'\n')

        # Print the last 4 values of the gdp series datetimes
        print(gdp.datetimes[-4:])
```

```
[ 16454.9  16490.7  16525.   16570.2]

['2015-07-01', '2015-10-01', '2016-01-01', '2016-04-01']

[datetime.datetime(2015, 7, 1, 0, 0) datetime.datetime(2015, 10, 1, 0, 0)
 datetime.datetime(2016, 1, 1, 0, 0) datetime.datetime(2016, 4, 1, 0, 0)]
```

```
In [6]: # Plot real GDP data
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.plot_date(gdp.datetimes,gdp.data,'-',lw=3,alpha = 0.65)
        ax.grid()
        ax.set_title(gdp.title)
        ax.set_ylabel(gdp.units)
```

```
Out[6]: <matplotlib.text.Text at 0x11aa812e8>
```



Methods

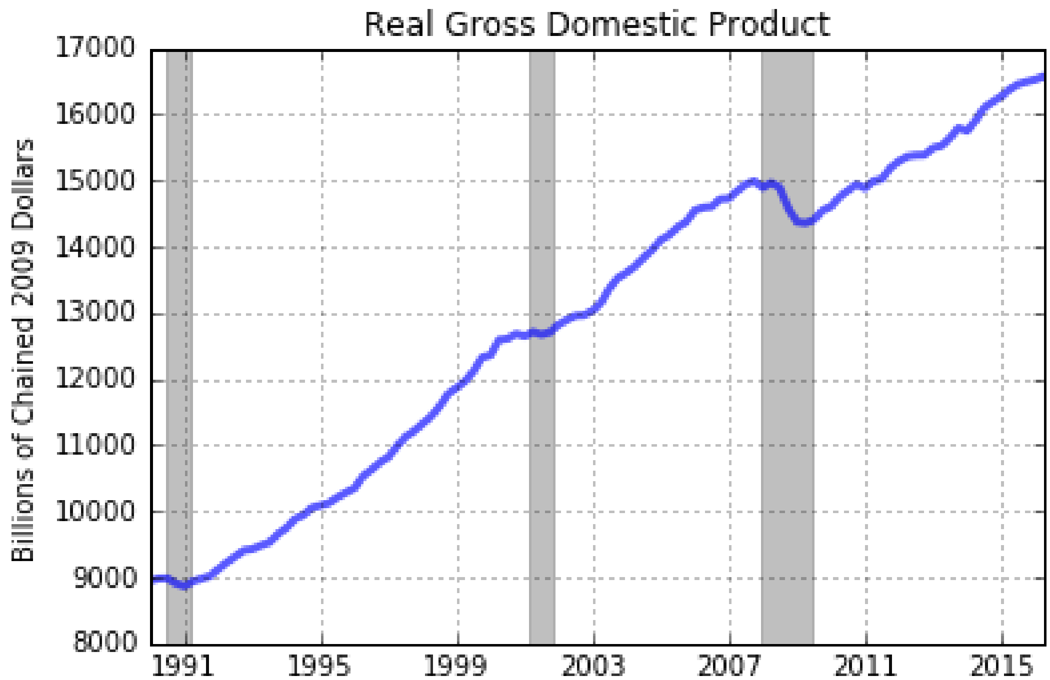
A *fredpy.series* instance has 22 methods:

- **apc**(log=True, method='backward')
- **bpfilter**(low=6, high=32, K=12)
- **cffilter**(low=6, high=32)
- **copy**()

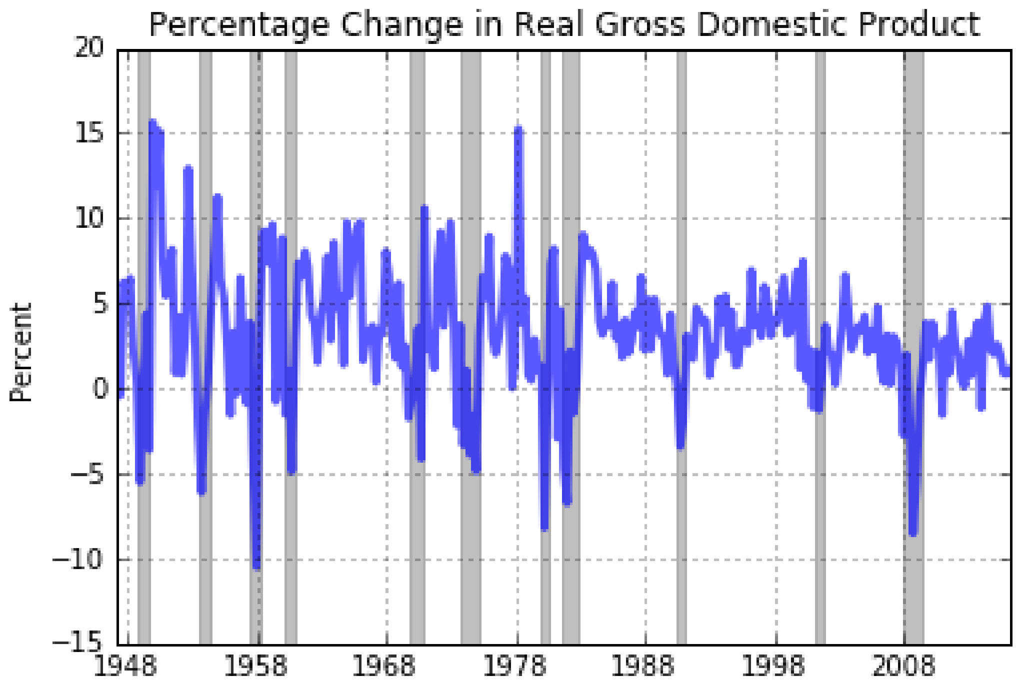
- **divide**(series2)
- **firstdiff**()
- **hpfilter**(lamb=1600)
- **lintrend**()
- **log**()
- **ma1side**(length)
- **ma2side**(length)
- **minus**(series2)
- **monthtoannual**(method='average')
- **monthtoquarter**(method='average')
- **pc**(log=True, method='backward', annualized=False)
- **percapita**(total_pop=True)
- **plus**(series2)
- **quartertoannual**(method='average')
- **recent**(N)
- **recessions**(color='0.5', alpha = 0.5)
- **times**(series2)
- **window**(win)

The fredpy documentation has detailed explanations of the use of these methods: http://www.briancjenkins.com/fredpy-package/documentation/build/html/series_class.html.

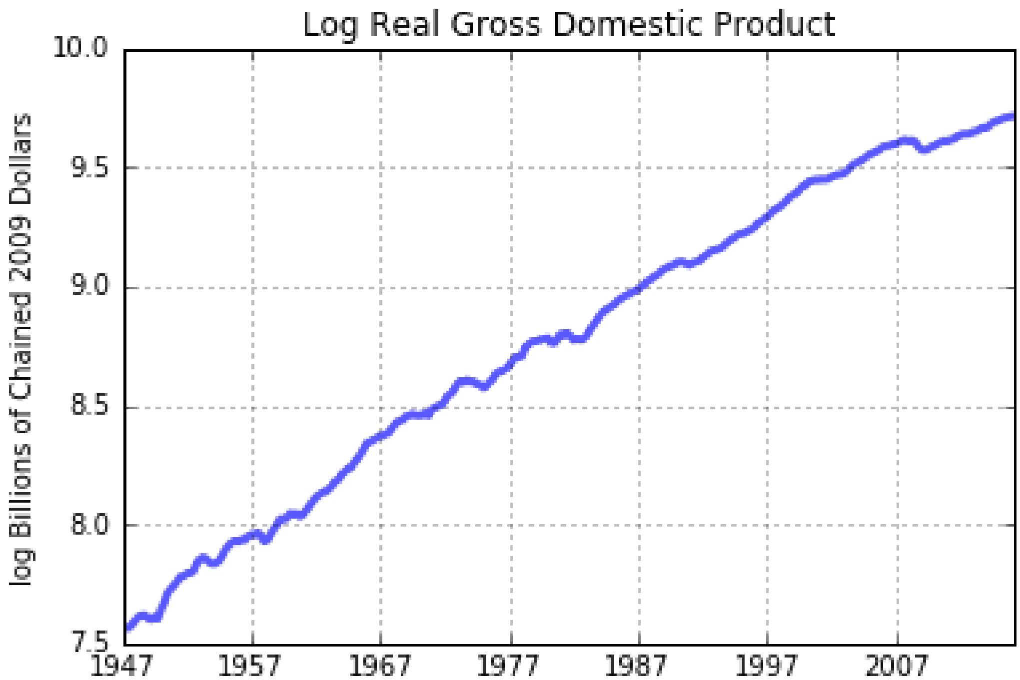
```
In [7]: # Restrict GDP to observations from January 1, 1990 to present
        win = ['01-01-1990','01-01-2200']
        gdp_win = gdp.window(win)

        # Plot
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.plot_date(gdp_win.datetimes,gdp_win.data,'-',lw=3,alpha = 0.65)
        ax.grid()
        ax.set_title(gdp_win.title)
        ax.set_ylabel(gdp_win.units)

        # Plot recession bars
        gdp_win.recessions()
```




```
In [8]: # Compute and plot the (annualized) quarterly growth rate of real GDP
        gdp_pc = gdp.pc(annualized=True)

        # Plot
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.plot_date(gdp_pc.datetimes,gdp_pc.data,'-',lw=3,alpha = 0.65)
        ax.grid()
        ax.set_title(gdp_pc.title)
        ax.set_ylabel(gdp_pc.units)

        # Plot recession bars
        gdp_pc.recessions()
```




```
In [9]: # Compute and plot the log of real GDP
        gdp_log = gdp.log()

        # Plot
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1)
        ax.plot_date(gdp_log.datetimes,gdp_log.data,'-',lw=3,alpha = 0.65)
        ax.set_title(gdp_log.title)
        ax.set_ylabel(gdp_log.units)
        ax.grid()
```



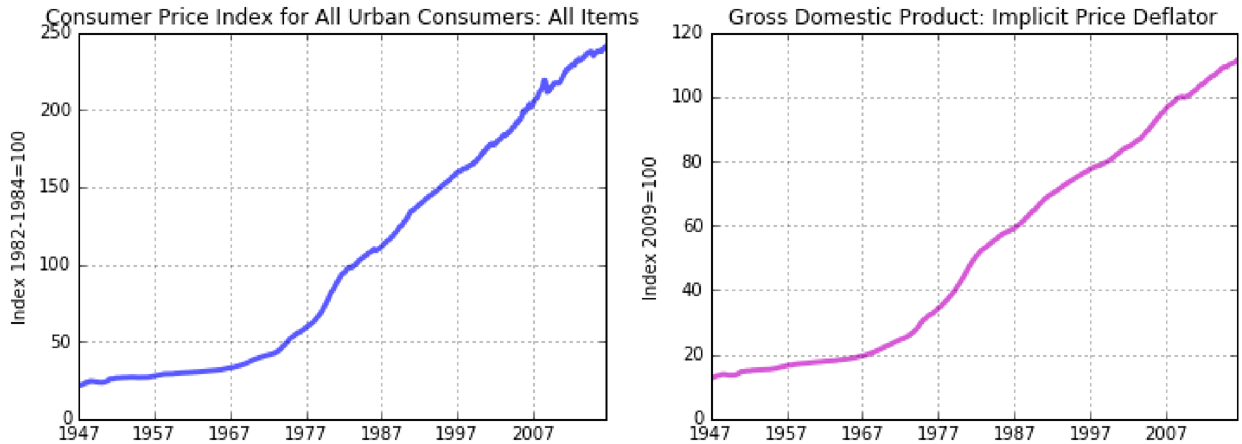
2.3.3 More examples

The following examples demonstrate some additional `fredpy` functionality.

Comparison of CPI and GDP deflator inflation

CPI data are released monthly by the BLS while GDP deflator data are released quarterly by the BEA. Here we'll first convert the monthly CPI data to monthly frequency compute inflation as the percentage change in the respective index since on year prior.

```
In [10]: # Download CPI and GDP deflator data
         cpi = series('CPIAUCSL')
         deflator = series('GDPDEF')

         fig = plt.figure(figsize=(12,4))
         ax = fig.add_subplot(1,2,1)
         ax.plot_date(cpi.datetimes,cpi.data,'-',lw=3,alpha = 0.65)
         ax.grid()
         ax.set_title(cpi.title)
         ax.set_ylabel(cpi.units)

         ax = fig.add_subplot(1,2,2)
         ax.plot_date(deflator.datetimes,deflator.data,'-m',lw=3,alpha = 0.65)
         ax.grid()
         ax.set_title(deflator.title)
         ax.set_ylabel(deflator.units)

Out[10]: <matplotlib.text.Text at 0x11afd5b70>
```



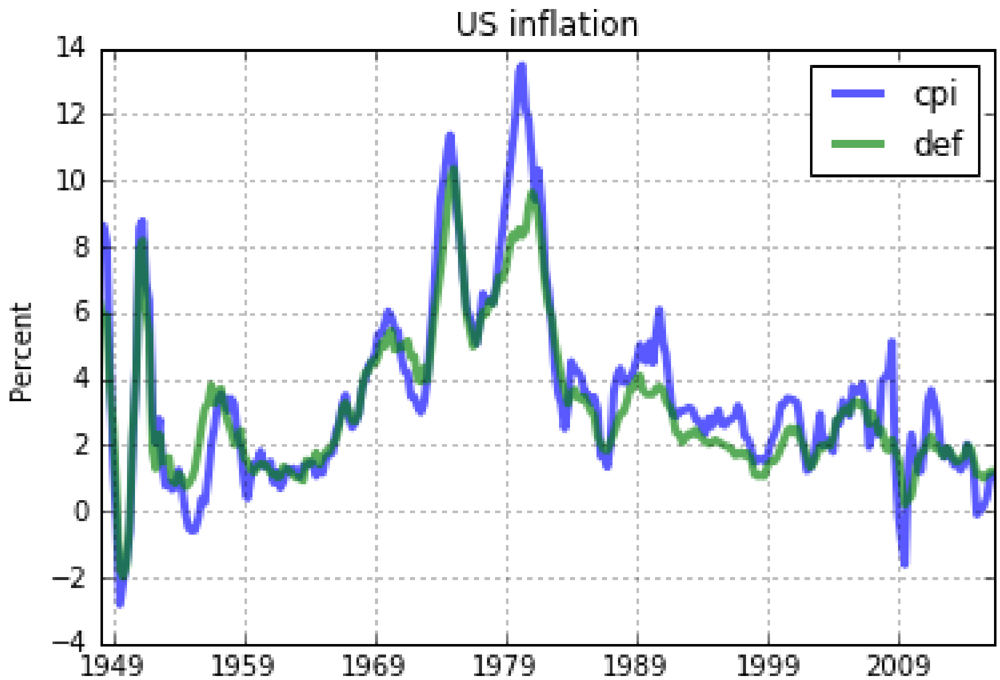

```
In [11]: # The CPI data are produced at a monthly frequency
         print(cpi.freq)

         # Convert CPI data to quarterly frequency to conform with the GDP deflator
         cpi2 = cpi.monthtoquarter()
         print(cpi2.freq)
```

```
Monthly
Quarterly
```

```
In [12]: # Compute the inflation rate based on each index
         cpi_pi = cpi2.apc()
         def_pi = deflator.apc()

         # Print date ranges for new inflation series
         print(cpi_pi.daterange)
         print(def_pi.daterange)
```

```
Range: 1948-01-01 to 2016-04-01
Range: 1948-01-01 to 2016-04-01
```

```
In [13]: fig = plt.figure(figsize=(6,4))
         ax = fig.add_subplot(1,1,1)
         ax.plot_date(cpi_pi.datetimes,cpi_pi.data,'-',lw=3,alpha = 0.65,label='cpi')
         ax.plot_date(def_pi.datetimes,def_pi.data,'-',lw=3,alpha = 0.65,label='def')
         ax.legend(loc='upper right')
         ax.set_title('US inflation')
         ax.set_ylabel('Percent')
         ax.grid()
```



Even though the CPI inflation rate is on average about .3% higher the GDP deflator inflation rate, the CPI and the GDP deflator produce comparable measures of US inflation.

Equalizing date ranges of different series

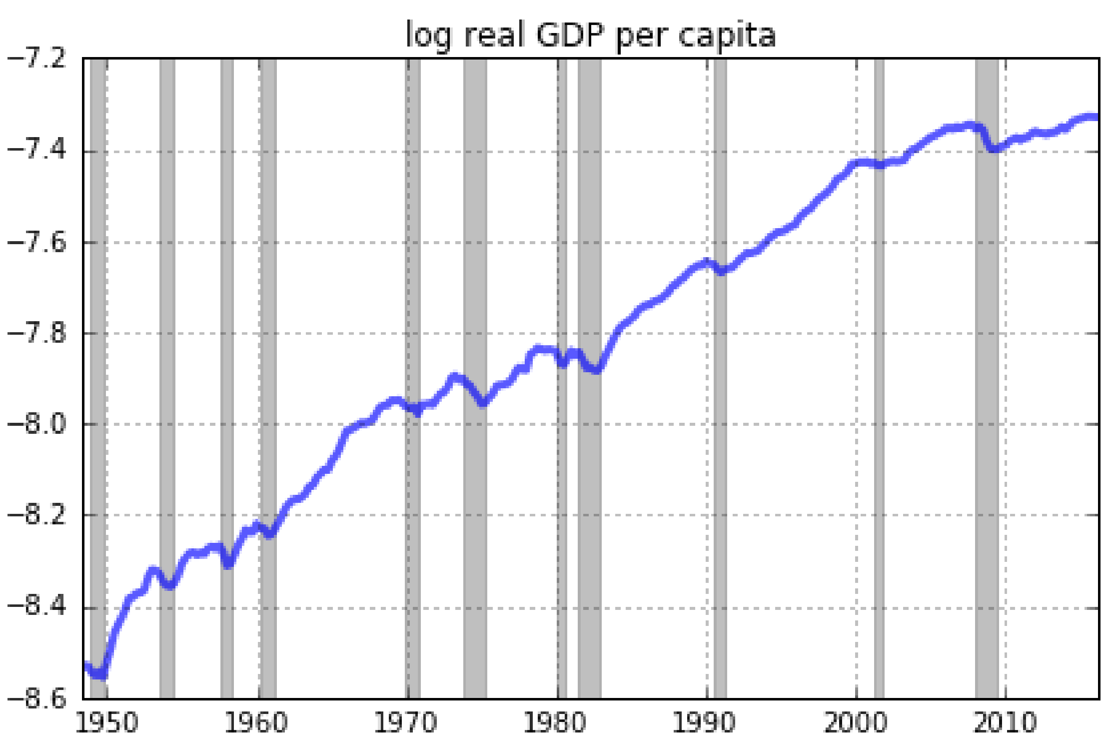
Often data series have different observation ranges. The *fredpy.window_equalize()* function provides a quick way to set the date ranges for multiple series to the same interval.

```
In [14]: from fredpy import window_equalize

         # Download unemployment and 3 month T-bill data
         unemp = series('UNRATE')
         tbill_3m = series('TB3MS')

         # Print date ranges for series
         print(unemp.daterange)
         print(tbill_3m.daterange)

         # Equalize the date ranges
         unemp, tbill_3m = window_equalize([unemp, tbill_3m])

         # Print the new date ranges for series
         print()
         print(unemp.daterange)
         print(tbill_3m.daterange)
```

```
Range: 1948-01-01 to 2016-08-01
Range: 1934-01-01 to 2016-08-01

Range: 1948-01-01 to 2016-08-01
Range: 1948-01-01 to 2016-08-01
```

Filtering 1: Extracting business cycle components from quarterly data with the HP filter

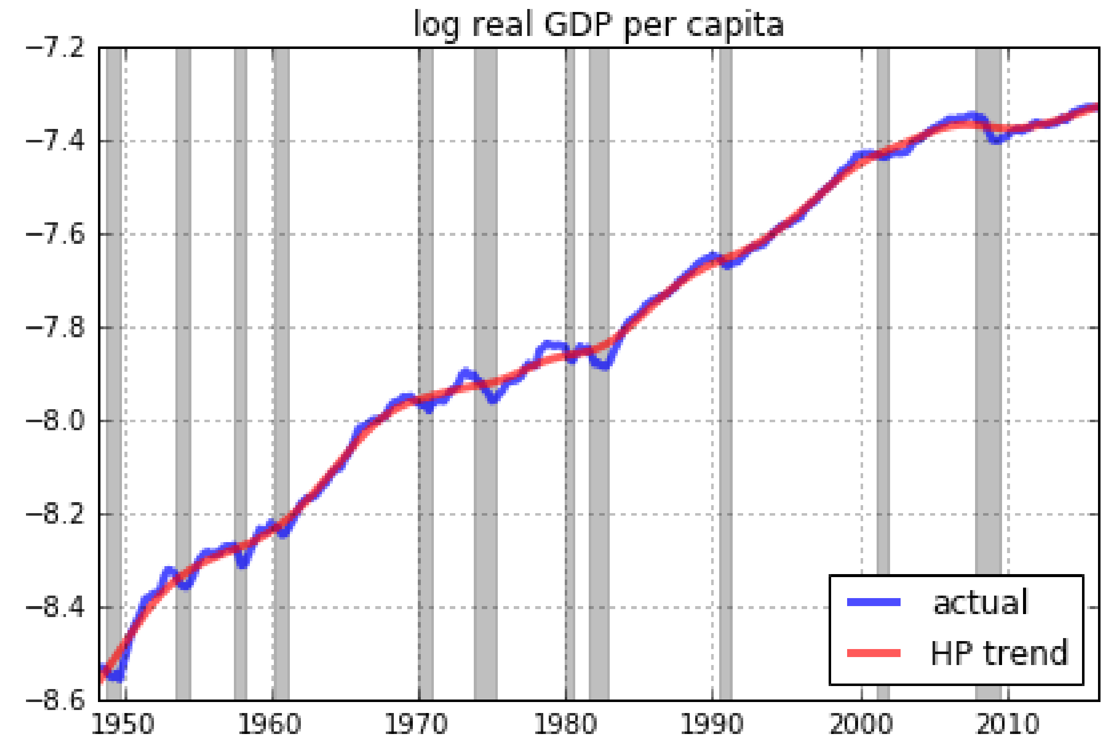
```
In [15]: # Download nominal GDP, the GDP deflator

         gdp = series('GDP')
         defl = series('GDPDEF')

         # Make sure that all series have the same window of observation
         gdp,defl = window_equalize([gdp,defl])

         # Deflate GDP series
         gdp = gdp.divide(defl)

         # Convert GDP to per capita terms
         gdp = gdp.percapita()

         # Take log of GDP
         gdp = gdp.log()

In [16]: # Plot log data
         fig = plt.figure(figsize=(6,4))

         ax1 = fig.add_subplot(1,1,1)
         ax1.plot_date(gdp.datetimes,gdp.data,'-',lw=3,alpha = 0.65)
         ax1.grid()
         ax1.set_title('log real GDP per capita')
         # ax1.set_ylabel(gdp.units)
         gdp.recessions()

         fig.tight_layout()
```

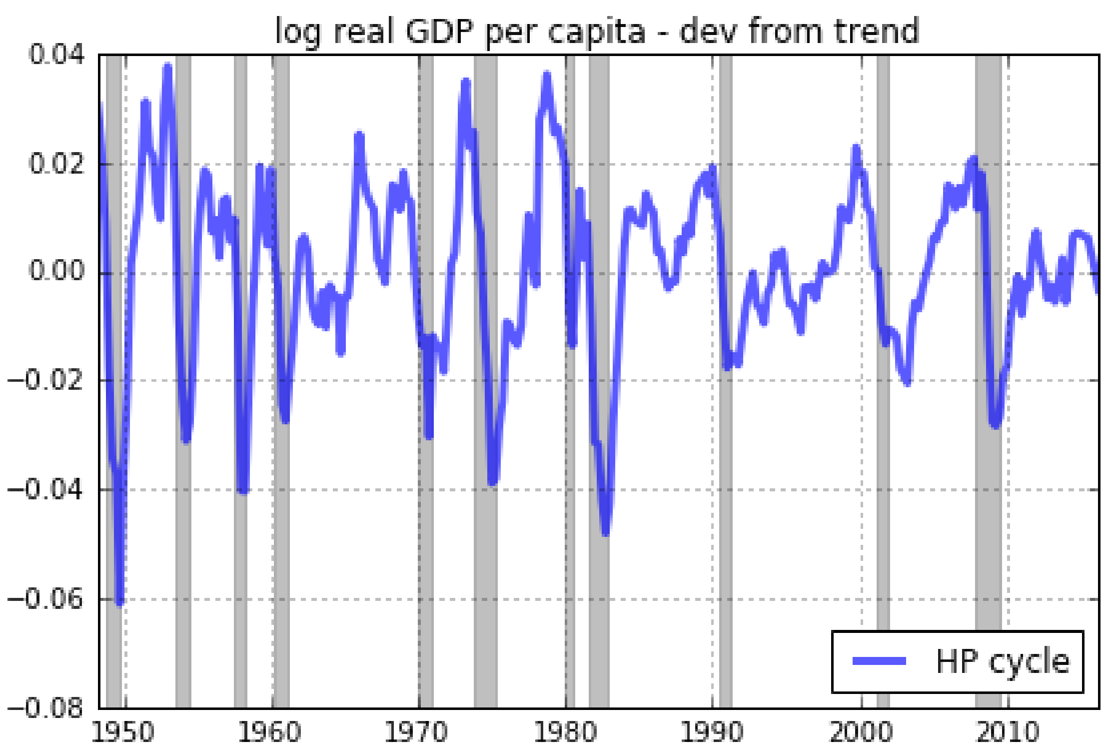


The post-Great Recession slowdown in US real GDP growth is apparent in the figure.

```
In [17]: # Compute the hpfilter
         gdp_cycle, gdp_trend = gdp.hpfilter()
```

```
In [18]: # Plot log data
         fig = plt.figure(figsize=(6,8))

         ax1 = fig.add_subplot(2,1,1)
         ax1.plot_date(gdp.datetimes,gdp.data,'-',lw=3,alpha = 0.7,label='actual')
         ax1.plot_date(gdp_trend.datetimes,gdp_trend.data,'r-',lw=3,alpha = 0.65,label='HP
         ax1.grid()
         ax1.set_title('log real GDP per capita')
         gdp.recessions()
         ax1.legend(loc='lower right')
         fig.tight_layout()

         ax1 = fig.add_subplot(2,1,2)
         ax1.plot_date(gdp_cycle.datetimes,gdp_cycle.data,'b-',lw=3,alpha = 0.65,label='HP
         ax1.grid()
         ax1.set_title('log real GDP per capita - dev from trend')
         gdp.recessions()
         ax1.legend(loc='lower right')
         fig.tight_layout()
```

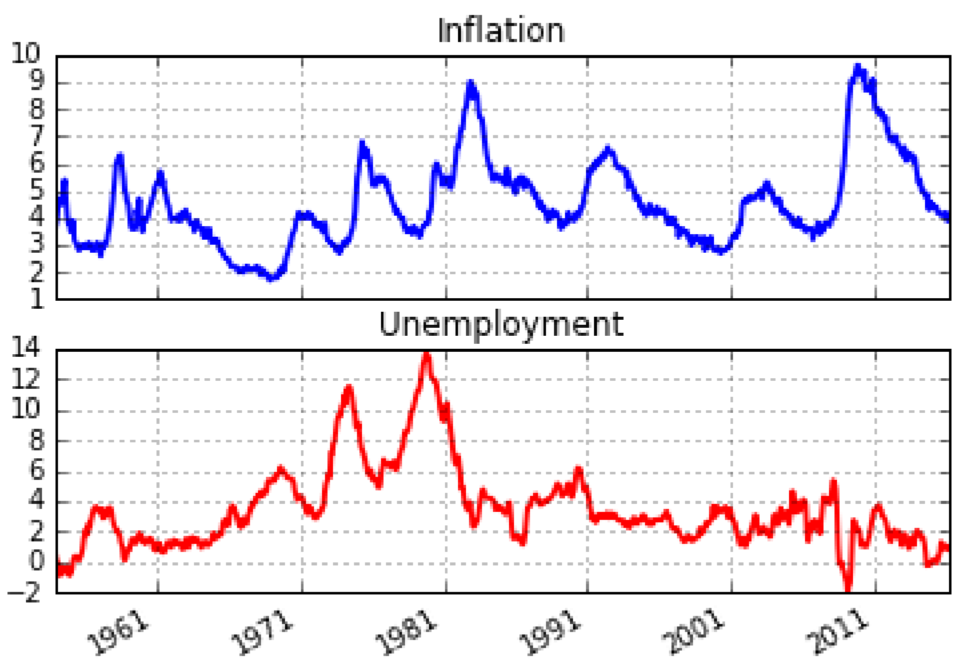




Filtering 2: Extracting business cycle components from monthly data

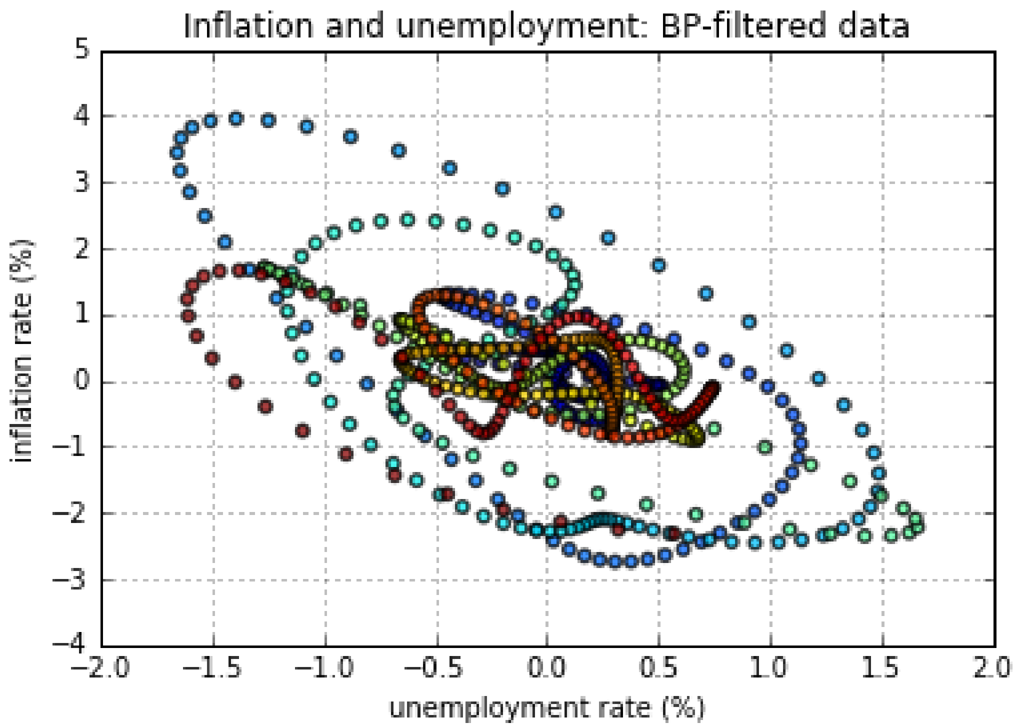
In Figure 1.5 from *The Conquest of American Inflation*, Thomas Sargent compares the business cycle componenets (BP filtered) of monthly inflation and unemployment data for the US from 1960-1982. Here we replicate Figure 1.5 to include the most recently available data and we also consturct the figure using HP filtered data.

```
In [19]: u = series('LNS14000028')
         p = series('CPIAUCSL')

         # Construct the inflation series
         p = p.pc(annualized=True)
         p = p.ma2side(length=6)

         # Make sure that the data inflation and unemployment series cver the same time in

         p,u = window_equalize([p,u])

         # Data

         fig = plt.figure()
         ax = fig.add_subplot(2,1,1)
         ax.plot_date(u.datetimes,u.data,'b-',lw=2)
         ax.grid(True)
         ax.set_title('Inflation')

         ax = fig.add_subplot(2,1,2)
         ax.plot_date(p.datetimes,p.data,'r-',lw=2)
         ax.grid(True)
         ax.set_title('Unemployment')

         fig.autofmt_xdate()
```

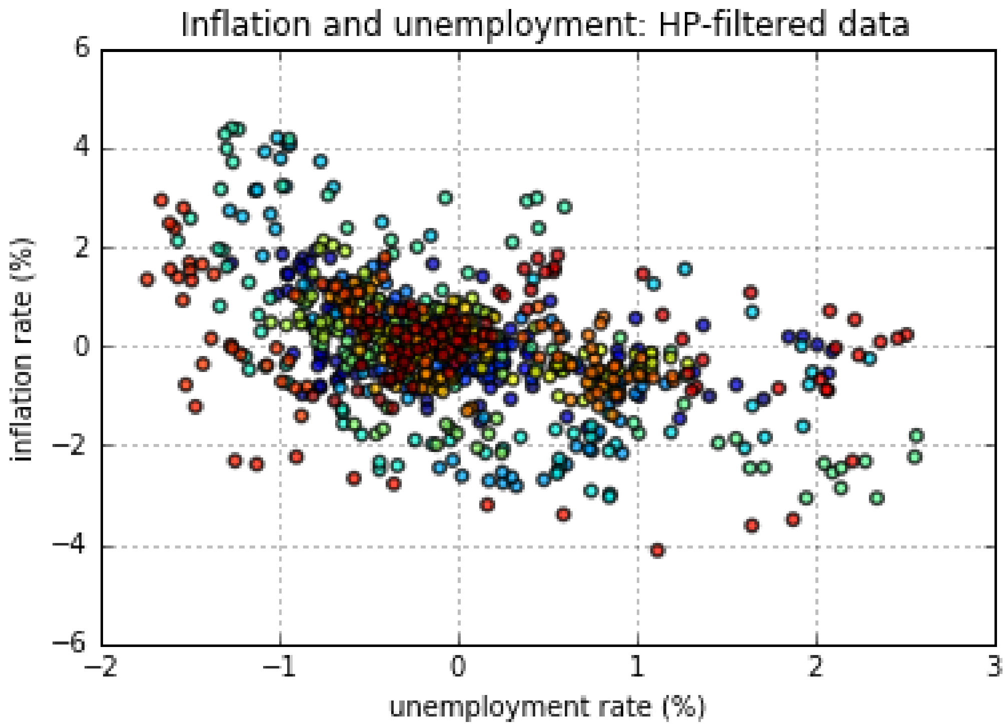



```
In [20]: # Filter the data
         p_bpcycle,p_bptrend = p.bpfilter(low=24,high=84,K=84)
         u_bpcycle,u_bptrend = u.bpfilter(low=24,high=84,K=84)

         # Scatter plot of BP-filtered inflation and unemployment data (Sargent's Figure 1
         fig = plt.figure()
```

```
ax = fig.add_subplot(1,1,1)
t = np.arange(len(u_bpcycle.data))
ax.scatter(u_bpcycle.data,p_bpcycle.data,facecolors='none',alpha=0.75,s=20,c=t, li
ax.set_xlabel('unemployment rate (%)')
ax.set_ylabel('inflation rate (%)')
ax.set_title('Inflation and unemployment: BP-filtered data')
ax.grid(True)
```




```
In [21]: # HP filter
         p_hpcycle,p_hptrend = p.hpfilter(lamb=129600)
         u_hpcycle,u_hptrend = u.hpfilter(lamb=129600)

         # Scatter plot of BP-filtered inflation and unemployment data (Sargent's Figure 1
         fig = plt.figure()
         ax = fig.add_subplot(1,1,1)
         t = np.arange(len(u_hpcycle.data))
         ax.scatter(u_hpcycle.data,p_hpcycle.data,facecolors='none',alpha=0.75,s=20,c=t, li
         ax.set_xlabel('unemployment rate (%)')
         ax.set_ylabel('inflation rate (%)')
         ax.set_title('Inflation and unemployment: HP-filtered data')
         ax.grid(True)
```



The choice of filterning method appears to strongly influence the results. While both filtering methods

Exporting data sets

Exporting data inported with `fredpy` to csv files is easy with `Pandas`.

```
In [22]: # create a Pandas DataFrame
         df = pd.DataFrame({'inflation':p.data,
                            'unemployment':u.data})

         # Set the index of the DataFrame
         df = df.set_index(pd.to_datetime(p.dates))

         print(df.head())

         # Export to csv
         df.to_csv('data.csv')
```

```
               inflation  unemployment
1954-01-01  6.330313e-01           3.6
1954-02-01  2.609508e-01           3.8
1954-03-01 -1.411834e-14           4.1
1954-04-01 -2.979518e-01           4.7
1954-05-01 -8.570949e-01           4.6
```

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search

F

fredpy.date_times() (built-in function), 7
fredpy.divide() (built-in function), 7
fredpy.minus() (built-in function), 7
fredpy.plus() (built-in function), 7
fredpy.quickplot() (built-in function), 7
fredpy.series (built-in class), 3
fredpy.series.apc() (built-in function), 3
fredpy.series.bpfilter() (built-in function), 4
fredpy.series.cffilter() (built-in function), 4
fredpy.series.copy() (built-in function), 4
fredpy.series.divide() (built-in function), 4
fredpy.series.firstdiff() (built-in function), 4
fredpy.series.hpfilter() (built-in function), 4
fredpy.series.lintrend() (built-in function), 5
fredpy.series.log() (built-in function), 5
fredpy.series.ma1side() (built-in function), 5
fredpy.series.ma2side() (built-in function), 5
fredpy.series.minus() (built-in function), 5
fredpy.series.monthtoannual() (built-in function), 5
fredpy.series.monthtoquarter() (built-in function), 5
fredpy.series.pc() (built-in function), 5
fredpy.series.percapita() (built-in function), 6
fredpy.series.plus() (built-in function), 6
fredpy.series.quartertoannual() (built-in function), 6
fredpy.series.recent() (built-in function), 6
fredpy.series.recessions() (built-in function), 6
fredpy.series.times() (built-in function), 6
fredpy.series.window() (built-in function), 6
fredpy.times() (built-in function), 7
fredpy.window_equalize() (built-in function), 8