Market Risk Calculation

Christian Carmona

May 8, 2014

In [1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib
import zipfile

from lxml import etree
from scipy.interpolate import interp1d
from datetime import datetime, timedelta
from matplotlib.backends.backend_pdf import PdfPages
from IPython.display import Image
```

Part I

Introduction

Given a portfolio comprised with bonds and spot position in different currencies, what are the possible looses that my portfolio could face for variations in market prices?

The purpose of this work is to implement a methodology for calculating Market risk indicators for investment portfolios containing a variety of fixed-income securities, spot position in several currencies and gold.

The main indicator to be calculated is the Value at Risk (VaR), which represents the 95% quantile of the profit and losses distribution.

In order to satisfy the above-mentioned goal, It is necessary to implement one of the existing methodologies in a computational language (Python). There are different approaches for calculating market risk, and one of the most used in practice is JPMorgan's RiskMetrics.

The processes for calculating this measures include the following steps:

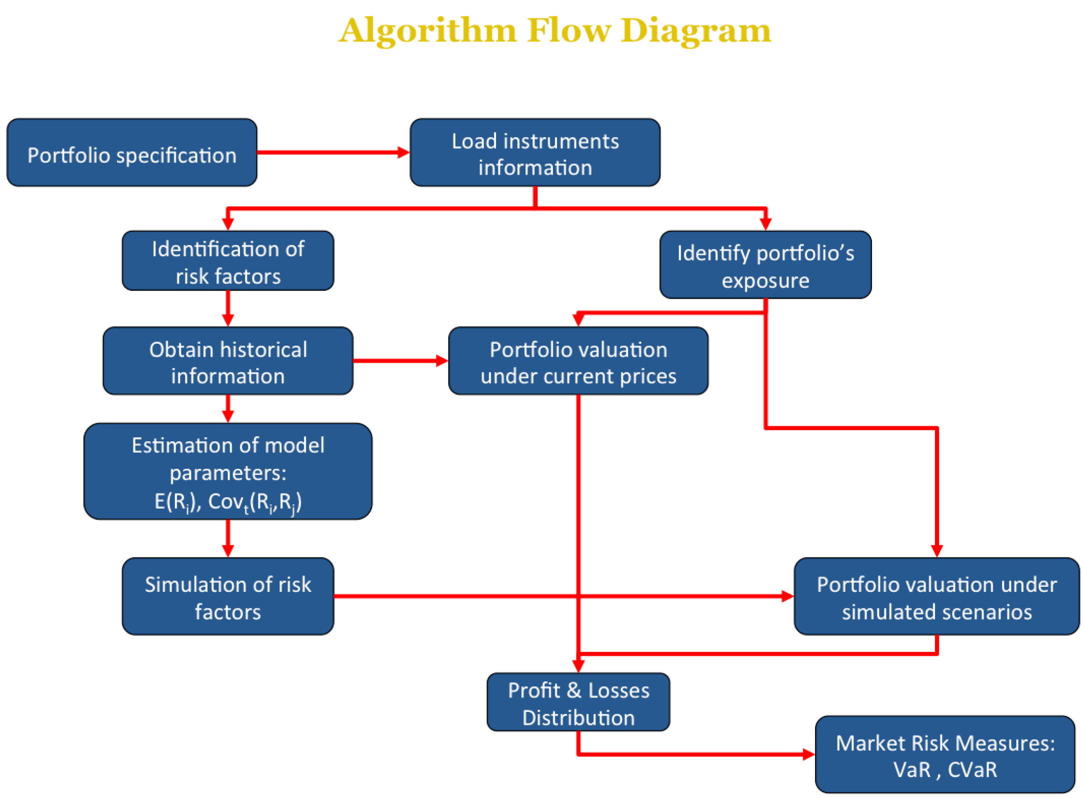
1. Define a portfolio
2. Identify market variables that affect the price of assets in the portfolio, namely risk factors
3. Obtain historical data for the risk factors
4. Calculate de covariance matrix of risk factors
5. Simulate market scenarios
6. Evaluate portfolio market value under simulated scenarios

7. Estimate the probability distribution of Profit and Losses in USD

The output of the process is a simulated distribution of profit and losses for each position in the portfolio. This enable the user to aggregate instruments by type, maturity, currency, etc. and perform risk attribution analysis.

In [7]:
```
Image(filename=data_dir+'algorithm.png')
```

Out [7]:



1 GitHub project repository

I developed this project using Github for version control and for storing all the files required to run this algorithm as well as additional documentation regarding market risk.

This repository can be found in the following address:

https://github.com/christianu7/stat_222_chris_carmona

Part II

Parameter definition

```
In [5]: # FOR READING and SAVING THE DATASETS IN THE SPECIFIED DIRECTORY #
        data_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_c

        # FOR SAVING THE RESULTS IN THE SPECIFIED DIRECTORY #
        out_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_ca

        # Portfolio file #
        port_file = 'port_2013-12.csv'

        # Date of Calculation #
        calc_date = '2013-12-30'
        calc_date = datetime.strptime(calc_date,'%Y-%m-%d')
```

Part III

Definition of Investment Portfolio

The portfolio is loaded from a .csv file containing two columns:

1. ID of each instrument in position
2. Position: depending on the instrument it can be position in foreign currency or face value of bond holding

```
In [3]: # Portfolio #
        #port = pd.read_csv( data_dir + port_file , na_values=['','NA','na','NaN','NULL'] )
        port = pd.read_csv( data_dir + port_file , na_values=['','NA','na','NaN','NULL'] )

        port = pd.Series(port.position.values,index=port.id_instr)
        port
```

```
Out [3]:
id_instr
USD               1000000
CAD                900000
EUR               7000000
JPY              99000000
US912796AQ20        85001
US912796AR03        83002
US912796AW97        50001
US912796BA68        50001
US912796BE80        53002
US912796BJ77        53001
US912796BP38        25000
US912796BS76        50002
US912796BT59        25001
US912796BU23        60001
US912796BV06        60001
...
CA135087ZC17      9000.0
CA135087ZF48     11341.7
CA135087ZL16      9900.0
CA135087ZN71      7991.7
CA135087ZQ03     10500.0
CA135087ZR85     10816.2
CA135087ZW70      8732.3
```

```
CA135087ZX53    15600.0
CA135087ZY37     7973.7
CA135087A388    15300.0
CA135087A537     8639.2
CA135087A792    14700.0
CA135087A958     9900.0
CA135087B295     8100.0
CA135087B527     9900.0
Length: 383, dtype: float64
```

Part IV

Risk Factors historical information

2 Historical Bond's Yield-to-Maturity Rates

2.1 Downloading data

The historical Data of yield to maturity for US government bonds is downloaded from the Department of treasury website.

The data is available in xml format

In [4]:
```
for year in range(2005,2015):
    url = 'http://www.treasury.gov/resource-center/data-chart-center/interest-rates/pa
    urllib.urlretrieve(url, data_dir+"yields_"+str(year)+".xml")
```

2.2 Reading data

For reading the data from the xml files I used the modile 'lxml' and 'etree'

In this case, the data was stored in the 'text' property of a deeply nested label.

This process dive into the levels of the xml, reads the data and finally creates a csv file with the cleaned data.

In [5]:
```
nodes_names = ['GOVT_USD_USA_1m','GOVT_USD_USA_3m','GOVT_USD_USA_6m',
               'GOVT_USD_USA_1y','GOVT_USD_USA_2y','GOVT_USD_USA_3y',
               'GOVT_USD_USA_5y','GOVT_USD_USA_7y','GOVT_USD_USA_10y',
               'GOVT_USD_USA_20y','GOVT_USD_USA_30y']
nodes_names_xml = ['BC_1MONTH','BC_3MONTH','BC_6MONTH',
                   'BC_1YEAR','BC_2YEAR','BC_3YEAR',
                   'BC_5YEAR','BC_7YEAR','BC_10YEAR',
                   'BC_20YEAR','BC_30YEAR']

##### Loading xml file #####

def read_yields_xml(xml_file,nodes_names,nodes_names_xml):
    doc = etree.parse(xml_file)
    # root element: 254 elements
    root = doc.getroot()

    rates_data = {}
    for entries in root.findall('{http://www.w3.org/2005/Atom}entry'):
        for content_i in entries.find('{http://www.w3.org/2005/Atom}content'):
```

```
            row_i = {}
            for property_i in content_i.getchildren():
                if property_i.tag.replace('{http://schemas.microsoft.com/ado/2007/08/d
                    date_i = property_i.text.replace("T00:00:00","")
                else:
                    row_i[ property_i.tag.replace('{http://schemas.microsoft.com/ado/2
            rates_data[date_i]=row_i

    #DATE=pd.DataFrame(rates_data).NEW_DATE.astype(np.string_)
    #DATE=pd.to_datetime( DATE )
    #rates_data = pd.DataFrame(rates_data,index=DATE)
    #rates_data[:3]

    rates_data = pd.DataFrame(rates_data)
    rates_data = rates_data.T
    rates_data = rates_data.drop(['BC_30YEARDISPLAY','Id'],axis=1)
    rates_data.rename( columns=dict( zip(nodes_names_xml,nodes_names) ),
                       inplace=True)

    rates_data = rates_data.convert_objects(convert_numeric=True)
    rates_data = rates_data.divide(100)
    rates_data = rates_data[nodes_names]

    rates_data = rates_data.reindex(index=pd.to_datetime(rates_data.index))
    return rates_data

yields_data = pd.DataFrame()
for year in range(2005,2015):
    xml_file = data_dir + "yields_" + str(year) + ".xml"
    yields_data = pd.concat([yields_data,read_yields_xml(xml_file,nodes_names,nodes_na

yields_data = yields_data[nodes_names]
yields_data = yields_data.convert_objects(convert_numeric=True)

yields_data.to_csv(data_dir+'ytm_data.csv')

yields_data
```

Out [5]:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2327 entries, 2005-01-03 00:00:00 to 2014-04-16
00:00:00
Data columns (total 11 columns):
GOVT_USD_USA_1m     2326  non-null values
GOVT_USD_USA_3m     2323  non-null values
GOVT_USD_USA_6m     2326  non-null values
GOVT_USD_USA_1y     2326  non-null values
GOVT_USD_USA_2y     2326  non-null values
GOVT_USD_USA_3y     2326  non-null values
GOVT_USD_USA_5y     2326  non-null values
GOVT_USD_USA_7y     2326  non-null values
GOVT_USD_USA_10y    2326  non-null values
GOVT_USD_USA_20y    2326  non-null values
GOVT_USD_USA_30y    2050  non-null values
dtypes: float64(11)
```

In [6]:

```
# If you want to get a pdf with the YTM rates
# Change "if False:" by "if True:"
if False:
    pp = PdfPages(out_dir+'yield_curves.pdf')
    for i in range(len(yields_data.columns)):
        plt.plot(yields_data.index, yields_data[yields_data.columns[i]])
```

```
        plt.legend([yields_data.columns[i]])
        pp.savefig()
        plt.close()
    pp.close()
```

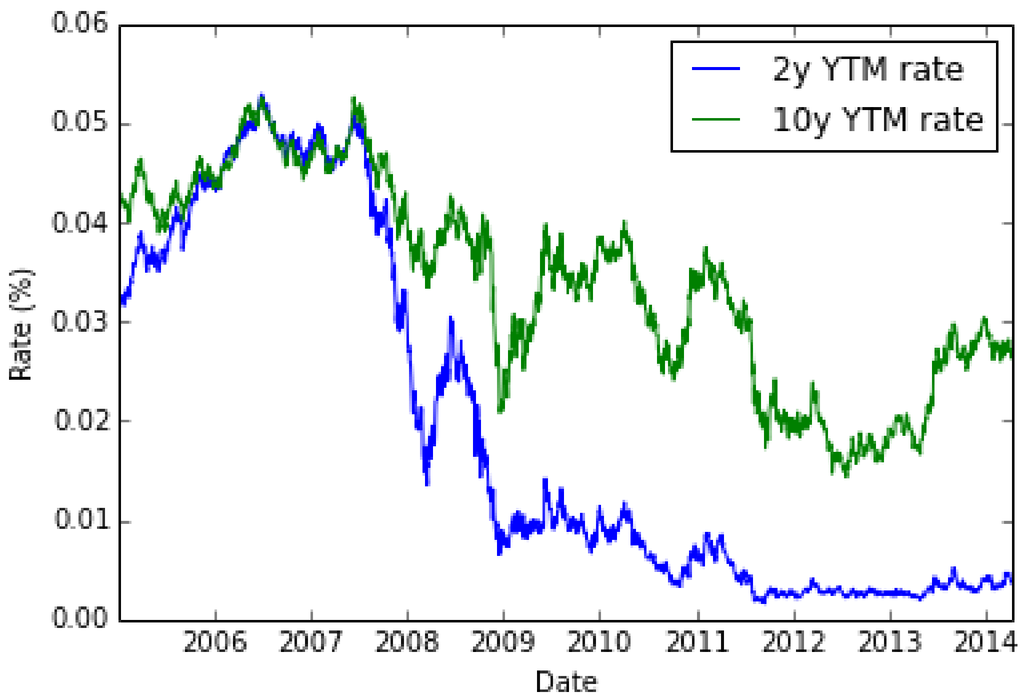
The following graphic shows the historical vlue of two important interest rates: the 2 year and 10 year Yield-to-maturity.

This two rates are often compared as a benchmark for analizing the fixed income market.

In [7]:
```
# Historical data: 2 and 10 year YTM rate
plt.plot(yields_data.index, yields_data['GOVT_USD_USA_2y'])
plt.plot(yields_data.index, yields_data['GOVT_USD_USA_10y'])
plt.ylabel('Rate (%)')
plt.xlabel('Date')
plt.legend(['2y YTM rate','10y YTM rate'])
```

Out [7]:

```
<matplotlib.legend.Legend at 0x109c447d0>
```



2.3 Transform yield rates to zero-coupon rates (Bootstrap method)

This method is an iterative procedure for getting spot rates from given prices of coupon bonds.

Suppose we have the following prices: $B_{0.5}, B_1, P_{1.5}, P_2, P_{2.5}, P_3, \ldots$.

It is easy to get spot rates from zero-coupon bonds by:

$z_t = -t * log(B_t)$

for $t = 0.5$ and $t = 1$, we can directly obtain $z_{0.5}$ and z_1.

For $t = 1.5$, we can use $P_{1.5}$ and get $B_{1.5}$ from the expression:

$$P_{1.5} = c * (B_{0.5} + B_1 + B_{1.5}) + B_{1.5}$$

then,

$$B_{1.5} = \frac{1}{1+c} * (P_{1.5} - c * (B_{0.5} + B_1))$$

which is a zero-coupon bond, from which we can get $z_{1.5}$

this algorithm is repeated for t={ 1.5, 2, 2.5, 3, … }

In [8]:
```
#ytm_curve = rates_data.ix[0,].values

nodes = np.array([1,3,6],dtype=np.float64)
nodes = nodes/12
nodes = np.append(nodes, np.array([1,2,3,5,7,10,20,30],dtype=np.float64) )

def zero_from_yield_bootstrap( ytm_curve , nodes ):

    nodes_old = nodes.copy()
    nodes = np.append(0,nodes)
    ytm_curve = np.append(0,ytm_curve)

    nodes_new = np.arange(0,max(nodes)+0.5,0.5)
    nodes_new = np.append(nodes,nodes_new)
    nodes_new = np.sort(nodes_new)
    nodes_new = np.unique(nodes_new)

    f = interp1d(nodes, ytm_curve, kind='linear')
    ytm_new = f(nodes_new)
    ytm_new[0]=0

    ytm_new = pd.Series(ytm_new,index=nodes_new)

    zero_new = np.zeros_like(ytm_new)

    nodes_coupon = np.in1d(nodes_new,np.arange(0,max(nodes),0.5)+0.5)

    for node_i in nodes_new[nodes_coupon==False]:
        zero_new[node_i] = (1+ytm_new[node_i]*node_i) ** (1/node_i)-1
    zero_new[0] = 0


    for node_i in nodes_new[nodes_coupon]:
        cpn_i = ytm_new[node_i]/2
        zero_new[node_i] = - np.log( (1 - cpn_i * np.exp(-nodes_new[ nodes_new<node_i

    return zero_new[np.in1d(nodes_new,nodes_old)].values
```

The following process obtain the zero-coupon rate for all days in the downloaded data

In [9]:
```
zero_rates_data = yields_data.copy()

#yields_data.apply(zero_from_yield_bootstrap,axis=1,nodes=nodes)

for i in range(yields_data.shape[0]):
#    print i
    zero_rates_data.ix[i,:] = zero_from_yield_bootstrap( yields_data.ix[i,].values,nod



zero_rates_data = zero_rates_data.dropna()
```

```
zero_rates_data.to_csv(data_dir+'zero_rates_data.csv')

zero_rates_data
```

Out [9]:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2047 entries, 2006-02-09 00:00:00 to 2014-04-16
00:00:00
Data columns (total 11 columns):
GOVT_USD_USA_1m     2047  non-null values
GOVT_USD_USA_3m     2047  non-null values
GOVT_USD_USA_6m     2047  non-null values
GOVT_USD_USA_1y     2047  non-null values
GOVT_USD_USA_2y     2047  non-null values
GOVT_USD_USA_3y     2047  non-null values
GOVT_USD_USA_5y     2047  non-null values
GOVT_USD_USA_7y     2047  non-null values
GOVT_USD_USA_10y    2047  non-null values
GOVT_USD_USA_20y    2047  non-null values
GOVT_USD_USA_30y    2047  non-null values
dtypes: float64(11)
```

In [10]:

```
# If you want to get a pdf with the historical value of zero-rates
# Change "if False:" for "if True:"
if False:
    pp = PdfPages(out_dir+'ytm_to_zero.pdf')
    for i in range(len(zero_rates_data.index)-5,len(zero_rates_data.index)):
        plt.plot(nodes,zero_rates_data.ix[i,],nodes,yields_data.ix[i,])
        plt.legend(['zero-coupon continuos','ytm semiannual'])
        pp.savefig()
        plt.close()
    pp.close()
```

The following graph shows the difference between the two curves:

1. YTM composed semiannualy
2. zero-coupon continuous rate

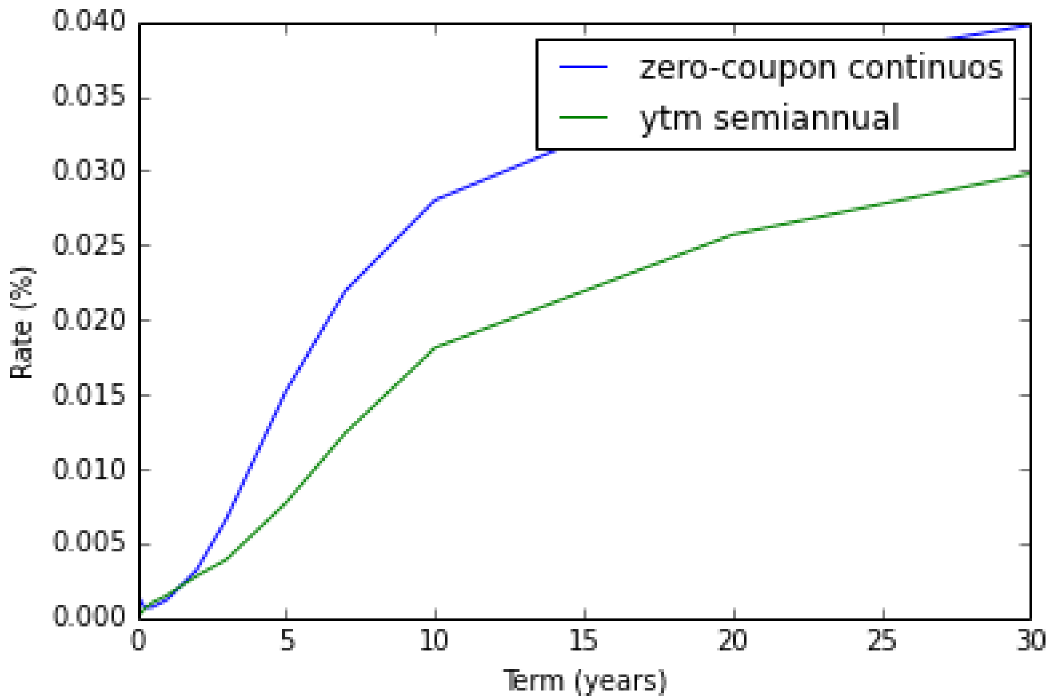
Both rates are equivalent, but the quotation is different

In [11]:

```
# Comparison of a zero-coupon yield curve with YTM curve
i = len(zero_rates_data.index)-50
plt.plot(nodes,zero_rates_data.ix[i,],nodes,yields_data.ix[i,])
plt.legend(['zero-coupon continuos','ytm semiannual'])
plt.ylabel('Rate (%)')
plt.xlabel('Term (years)')
```

Out [11]:

```
<matplotlib.text.Text at 0x109c8a750>
```



3 Historical Foreign Exchange Rates

3.1 Downloading data

The historical Data of Currency rates is downloaded from the Federal Resevre website. The data is available in xml format.

In [12]:
```
url_zip = 'http://www.federalreserve.gov/datadownload/Output.aspx?rel=H10&filetype=zip

##### Download zip file with data #####

#import urllib
urllib.urlretrieve(url_zip, data_dir+"ccy.zip")

#import zipfile
zip_file = open(data_dir+'ccy.zip', 'rb')
z = zipfile.ZipFile(zip_file)
#print z.namelist()
z.extract('H10_data.xml', data_dir)
```

Out [12]:
```
'/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT
222/stat_222_chris_carmona/data/H10_data.xml'
```

3.2 Reading data

Equivalent to the historical data for YTM, the process for obtaining the information consists in going deep into de different levels of the xml format and find the important data.

In this case, the data was embeded as an attribute for the labels 'Obs'

In [13]:
```
##### Loading xml file #####

doc = etree.parse(data_dir+'H10_data.xml')
# root element: 254 elements
root = doc.getroot()

data_set = root.find('{http://www.federalreserve.gov/structure/compact/common}DataSet'
series = data_set.findall('{http://www.federalreserve.gov/structure/compact/H10_H10}Se

currencies = ['AUD', 'CAD', 'CHF', 'CLP', 'CNY', 'EUR', 'GBP', 'JPY', 'NOK', 'NZD', 'S

ccy_data = {}
for serie in series:
#    print serie.attrib['FX'] + ' ' + serie.attrib['CURRENCY'] + ' ' + serie.attrib['U
    if serie.attrib['FX'] in currencies and serie.attrib['FREQ']=='9':
#        print serie.attrib['FX'] + ' ' + serie.attrib['CURRENCY'] + ' ' + serie.attri
        row_i = {}
        for obs in serie.findall('{http://www.federalreserve.gov/structure/compact/com
            row_i[ obs.attrib['TIME_PERIOD'] ] = obs.attrib['OBS_VALUE']
        ccy_data[serie.attrib['FX']]=row_i

ccy_data = pd.DataFrame(ccy_data)
ccy_data = ccy_data.convert_objects(convert_numeric=True)
ccy_data[ccy_data==-9999] = float('NaN')
ccy_data = ccy_data.reindex(index=pd.to_datetime(ccy_data.index))

ccy_data.to_csv(data_dir+'currencies_data.csv')

ccy_data
```

Out [13]:
```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 11290 entries, 1971-01-04 00:00:00 to 2014-04-11
00:00:00
Data columns (total 11 columns):
AUD    10856  non-null values
CAD    10869  non-null values
CHF    10863  non-null values
CNY    8303  non-null values
EUR    3842  non-null values
GBP    10863  non-null values
JPY    10857  non-null values
NOK    10862  non-null values
NZD    10847  non-null values
SEK    10862  non-null values
SGD    8362  non-null values
dtypes: float64(11)
```

In [14]:
```
# If you want to get a pdf with the historical value of currencies
# Change "if False:" for "if True:"
if False:
    pp = PdfPages(out_dir+'currencies.pdf')
    for i in range(len(ccy_data.columns)):
        plt.plot(ccy_data.index, ccy_data[ccy_data.columns[i]])
        plt.legend([ccy_data.columns[i]])
        pp.savefig()
```

```
        plt.close()
    pp.close()
ccy_data.columns
```

Out [14]:

```
Index([u'AUD', u'CAD', u'CHF', u'CNY', u'EUR', u'GBP', u'JPY', u'NOK',
u'NZD', u'SEK', u'SGD'], dtype=object)
```

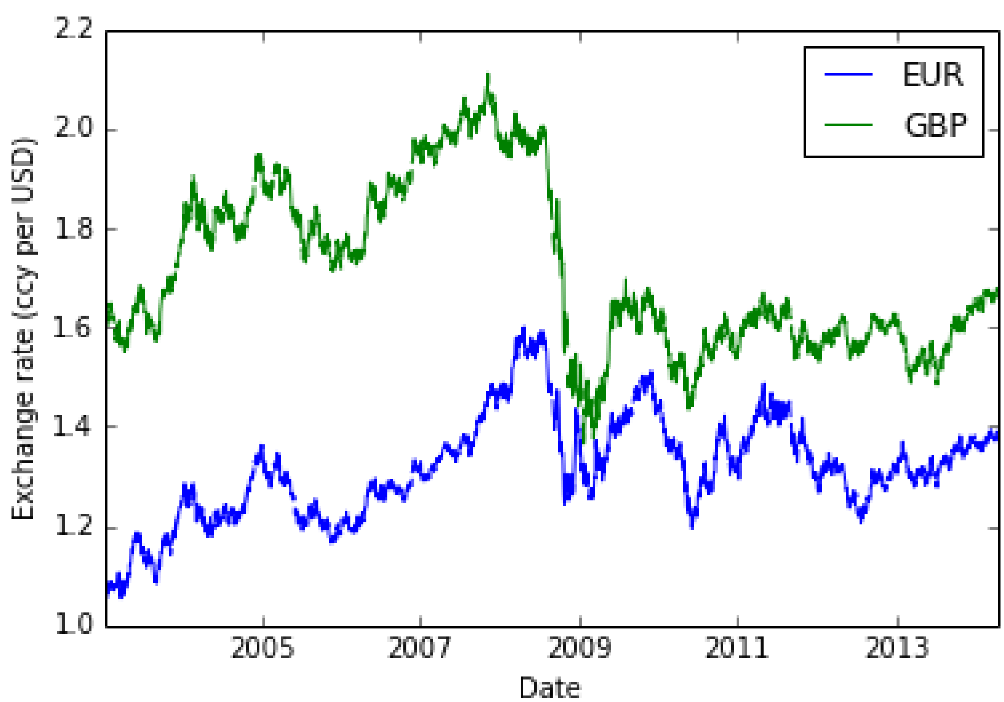
The following graphic shows the historical levels for a couple of foreign currencies.

In [18]:

```
# Historical data: Some currencies rates
idx = (ccy_data.index > datetime.strptime('2003-01-01','%Y-%m-%d'))
plt.plot(ccy_data.index[idx], ccy_data['EUR'][idx])
plt.plot(ccy_data.index[idx], ccy_data['GBP'][idx])
plt.ylabel('Exchange rate (ccy per USD)')
plt.xlabel('Date')
plt.legend(['EUR','GBP'])
```

Out [18]:

```
<matplotlib.legend.Legend at 0x109cd3b50>
```



4 Full Risk factors Information (Merging yield rates and currencies)

The following process merges the two data sources: rates and currencies

In [19]:

```
risk_factors_hist = pd.merge( ccy_data, zero_rates_data, right_index=True, left_index=

risk_factors_hist = risk_factors_hist.dropna()

risk_factors_hist.to_csv(data_dir+'factor_hist.csv')
```

```
risk_factors_hist
```

Out [19]:

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2043 entries, 2006-02-09 00:00:00 to 2014-04-11
00:00:00
Data columns (total 22 columns):
AUD                  2043  non-null values
CAD                  2043  non-null values
CHF                  2043  non-null values
CNY                  2043  non-null values
EUR                  2043  non-null values
GBP                  2043  non-null values
JPY                  2043  non-null values
NOK                  2043  non-null values
NZD                  2043  non-null values
SEK                  2043  non-null values
SGD                  2043  non-null values
GOVT_USD_USA_1m      2043  non-null values
GOVT_USD_USA_3m      2043  non-null values
GOVT_USD_USA_6m      2043  non-null values
GOVT_USD_USA_1y      2043  non-null values
GOVT_USD_USA_2y      2043  non-null values
GOVT_USD_USA_3y      2043  non-null values
GOVT_USD_USA_5y      2043  non-null values
GOVT_USD_USA_7y      2043  non-null values
GOVT_USD_USA_10y     2043  non-null values
GOVT_USD_USA_20y     2043  non-null values
GOVT_USD_USA_30y     2043  non-null values
dtypes: float64(22)
```

Part V

Descriptive Data bases

The following databases specify additional information for each instrument according to their ID.

instr_info_file

Table containing the description of each instrument. i.e. for a given ID, specifies if the instrument is a currency position or a bond. If it is a bond, what type of bond it is AGENCY or GOVT, and additional information.

cshf_info_file

Table containing all cashflows for the bonds that can be part of the portfolio. It consists of three columns: ID, date of cashflow, and cashflow amount in USD. With one row for each cashflow date an instrument combination.

curves_info_file

Table containing the description of the risk factors. Remember that the risk factors are the random processes that will be stressed to valuate the portfolio under simulated scenarios.

In [20]:
```
# Global databases #

# instruments description #
instr_info_file = 'instr_description.csv'
instr_info = pd.read_csv( data_dir + instr_info_file , na_values=['','NA','na','NaN','

# Risk factors description #
curves_info_file = 'curve_nodes_description.csv'
curves_info = pd.read_csv( data_dir + curves_info_file , na_values=['','NA','na','NaN'

# fixed-income instruments cashflows #
cshf_info_file = 'instr_cashflows.csv'
cshf_info = pd.read_csv( data_dir + cshf_info_file , na_values=['','NA','na','NaN','NU
cshf_info['Date'] = pd.to_datetime(cshf_info['Date'])
cshf_info = cshf_info.groupby(['id_instr','Date'])['value'].sum()
#cshf_info
```

Part VI

Risk factors scenario simulation

In the following section, we perform the simulation of risk factors. This task requires two basic steps:

1. Estimation of the covariance matrix of risk factors using historical values and EWMA estimator
2. Simulation of returns of risk factors using the EWMA estimation and our geometric stochastic model.

RiskMetrics model of risk factor returns: A modified random walk RiskMetrics methodology assumes that logarithmic returns on the risk factors follow a multivariate normal distribution conditional on dynamic volatility and covariances. Suppose that we have n risk factors Then, the process generating the changes for the i-th risk factor can be written as:

$$\frac{dP_t^{(i)}}{P_t^{(i)}} = \mu_i dt + \sigma_i dW_t$$

where

$$Var(dW^{(i)}) = dt$$
$$Cov(dW^{(i)}, dW^{(j)}) = \rho_{i,j} dt$$

It follows that the return on each asset from time t to time t+T can be written as:

$$r_{t,T}^{(i)} = (\mu_i - \frac{1}{2}\sigma_i^2)(T-t) + \sigma_i \epsilon_i \sqrt{T-t}$$

where

$$\epsilon_i \sim N(0,1)$$
$$Cov(\epsilon_i, \epsilon_j) = \rho_{i,j}$$

If we incorporate the zero mean assumption we get that:

$$r_{t,T}^{(i)} = \sigma_i \epsilon_i \sqrt{T-t}$$

Finally, the EWMA estimation of the covariance matrix is an estimation that gives more importance to the more recent observations and the importance decrease exponentially with new observations.

The parameter to control the diference in importance given to newest observations is called 'Decay factor'

The i,j entrance in the matrix is given by:

$$\Sigma_{i,j} = \frac{1-\lambda}{1-\lambda^{m+1}} \sum_{k=0}^{m} \lambda^k r_{t-k}^{(i)} r_{t-k}^{(j)}$$

where:

m is the number of observations considered for calculation

λ is the decay factor

r_t risk factor return at time t

5 Calculation of Log-returns for Risk-factors

In [21]:
```
# Flip all currencies to dollars per curency
cur_usd = ['AUD', 'EUR', 'GBP', 'NZD']
cur_flip = list(set(currencies).difference(set(cur_usd)))

# Log-returns calculation #
def get_logrtn(risk_factors_hist,calc_date,cur_flip):
    # only consider 1 year and half of history
    rtn_data = risk_factors_hist.copy()
    rtn_data = rtn_data[ (rtn_data.index >= calc_date - timedelta(days=365*1.5+1)) & (

    # Get rid of negative and zero values
    rtn_data[rtn_data<=0] = float('NaN')
    rtn_data = rtn_data.dropna()


    # Flip all currencies to dollars per curency
    for cur in cur_flip:
        if cur in rtn_data.columns:
            #print cur
            rtn_data[cur] = 1/rtn_data[cur]

    # Calculate log-returns
    rtn_data = rtn_data.apply(np.log)
    rtn_data = rtn_data - rtn_data.shift(1)
    rtn_data = rtn_data.dropna()
    rtn_data.to_csv(data_dir+'logrtn_data.csv')
    return rtn_data
```

6 Estimation of Risk factors Covariance matrix

In [ ]:
```
# Covariance estimation:
# EWMA function
def ewma( X, decay = 0.96):
    x_i = np.matrix(X[0,:])
    sigma = x_i.T * x_i

    for i in range(1,X.shape[0]) :
        x_i = np.matrix(X[i,:])
        sigma = decay * sigma + (1-decay) * x_i.T * x_i

    return(sigma)
```

7 Function for simulation of risk factors

The function will estimate the covariance matrix for a specific day, simulates returns according to the Multivariate geometrical brownian model defined above, apply those returns to the quotes of risk factors in the date, and that will provide us with simulated new quotes.

The User has to specify:

1. Number of simulations
2. Simulation date
3. Historical risk factors (a data frame with the historical information)
4. Currencies whose quote standard is not US dollars per currency
5. Decay factor for the EWMA covariance estimation

In [ ]:
```
# SIMULATE RISK FACTORS #

def simulate_risk_factors( n_sim, calc_date, risk_factors_hist, cur_flip, decay=0.96,
#def get_logrtn_sim( n_sim, calc_date, risk_factors_hist, decay=0.96, seed=0 ):

    # Calculate log returns
    rtn_data = get_logrtn(risk_factors_hist,calc_date,cur_flip)

    # Covariance matrix estimation
    sigma = ewma( rtn_data.values, decay=decay )
    # sigma

    mean = np.zeros(sigma.shape[0])

    # Generate simulated log returns
    # Simulation of log-returns (Multivariate Normal)
    np.random.seed(seed=seed)
    rtn_data_sim = np.random.multivariate_normal( mean, sigma, n_sim )
    rtn_data_sim = pd.DataFrame( rtn_data_sim )
    rtn_data_sim.columns = rtn_data.columns

    #####     Baseline scenario     #####
    temp = risk_factors_hist.ix[calc_date,]
    risk_factors_base = temp.copy()

    # Flip all currencies to dollars per curency
    for cur in cur_flip:
        if cur in risk_factors_base.index :
            risk_factors_base[cur] = 1/risk_factors_base[cur]

    #####     Simulated scenarios     #####
    risk_factors_sim = risk_factors_base * np.exp(rtn_data_sim)

    # Flip all currencies to their original exchange standard
    for cur in cur_flip:
        if cur in risk_factors_sim.columns :
            risk_factors_sim[cur] = 1/risk_factors_sim[cur]
            risk_factors_base[cur] = 1/risk_factors_base[cur]

    risk_factors_sim.to_csv(data_dir+'simulated_scenarios.csv')
    risk_factors_sim

    return risk_factors_sim
```

8 Obtain Simulated scenarios

In [22]:
```
risk_factors_sim = simulate_risk_factors( n_sim=1000,
                                          calc_date=calc_date,
                                          risk_factors_hist=risk_factors_hist,
                                          cur_flip=cur_flip, decay=0.96, seed=0 )
risk_factors_sim
```

Out [22]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Data columns (total 22 columns):
AUD                    1000  non-null values
CAD                    1000  non-null values
CHF                    1000  non-null values
CNY                    1000  non-null values
EUR                    1000  non-null values
GBP                    1000  non-null values
JPY                    1000  non-null values
NOK                    1000  non-null values
NZD                    1000  non-null values
SEK                    1000  non-null values
SGD                    1000  non-null values
GOVT_USD_USA_1m        1000  non-null values
GOVT_USD_USA_3m        1000  non-null values
GOVT_USD_USA_6m        1000  non-null values
GOVT_USD_USA_1y        1000  non-null values
GOVT_USD_USA_2y        1000  non-null values
GOVT_USD_USA_3y        1000  non-null values
GOVT_USD_USA_5y        1000  non-null values
GOVT_USD_USA_7y        1000  non-null values
GOVT_USD_USA_10y       1000  non-null values
GOVT_USD_USA_20y       1000  non-null values
GOVT_USD_USA_30y       1000  non-null values
dtypes: float64(22)
```

Part VII

Portfolio valuation

The following section valuate the portfolio using the cashflow of each instrument and obtaining the Financial present value.

9 Function for portfolio valuation

For a given set of

In [23]:
```
def port_valuation_slow( risk_factors , port, instr_info, cshf_info):

    port_mtm_value = np.zeros_like( port )
    port_mtm_value.index = port.index
```

```
    # Discount factors calculation
    discount = pd.Series( np.array(risk_factors[nodes_names].values) ,
                          index=[ calc_date + pd.DateOffset(days=x*365) for x in nodes
    discount = np.exp(-discount * nodes)
    discount = discount.set_value(calc_date, 1)

    for instr in port.index:
        if instr == 'USD':
            port_mtm_value[instr] = port[instr]
        if instr_info.ix[instr_info.id_instr==instr,"instr_type_01"] == 'CURRENCY' and
            if instr in cur_flip:
                port_mtm_value[instr] = port[instr] / risk_factors[instr]
            else:
                port_mtm_value[instr] = port[instr] * risk_factors[instr]
        if instr_info.ix[instr_info.id_instr==instr,"instr_type_01"] == 'BOND':
            #print instr
            instr_cshf_dates = cshf_info[instr].index
            instr_cshf_dates = instr_cshf_dates[instr_cshf_dates>=calc_date]

            discount_1 = discount.reindex(index=discount.index.append(instr_cshf_dates
            discount_1 = discount_1.sort_index()
            discount_1 = discount_1.interpolate(method="time").dropna()

            #print cshf_info[instr]
            #print discount_1
            #print (cshf_info[instr] * discount_1)

            port_mtm_value[instr] = sum( ( port[instr] * (cshf_info[instr]/1000000) *
            #print port_mtm_value[instr]
            if instr_info.ix[instr_info.id_instr==instr,"currency"] != 'USD':
                instr_ccy = instr_info.ix[instr_info.id_instr==instr,"currency"]
                if instr_ccy in cur_flip:
                    port_mtm_value[instr] = port_mtm_value[instr] / risk_factors[instr
                else:
                    port_mtm_value[instr] = port_mtm_value[instr] * risk_factors[instr

    return port_mtm_value
```

In [24]:

```
#risk_factors = risk_factors_base.copy()
#port
#instr_info
#cshf_info

def port_valuation( risk_factors , port, instr_info, cshf_info,calc_date):
    # Cash flows for bonds
    bonds_cshf = cshf_info.ix[ port.index.values ].unstack('id_instr')
    bonds_cshf = bonds_cshf[bonds_cshf.index>=calc_date]
    bonds_cshf = bonds_cshf/1000000
    bonds_cshf = bonds_cshf * port[bonds_cshf.columns]

    # Cash flows for currencies
    ccy_cshf = port[currencies].dropna()
    ccy_cshf = pd.DataFrame( ccy_cshf.values, index=ccy_cshf.index, columns=[calc_date
    if 'USD' in port.index:
        ccy_cshf['USD'] = port['USD']

    # cashflows for all the portfolio
    port_cshf = pd.merge( ccy_cshf, bonds_cshf, left_index=True, right_index=True, how
    port_cshf.dropna( how='all' )

    # Discount factors calculation
    discount = pd.Series( np.array(risk_factors[nodes_names].values) ,
                          index=[ calc_date + pd.DateOffset(days=x*365) for x in nodes
    discount = np.exp(-discount * nodes)
    discount = discount.set_value(calc_date, 1)
    discount = discount.reindex( index= discount.index.append(port_cshf.index).unique(
    discount = discount.sort_index()
```

```
    discount = discount.interpolate(method="time")
    discount = discount.reindex( index=port_cshf.index)

    # present value
    port_cshf_pv = discount * port_cshf

    # present value in USD
    ccy_instr = instr_info.ix[ instr_info.id_instr.isin(port.index), ['id_instr','curr
    ccy_instr = ccy_instr.set_index('id_instr').currency

    #print port_cshf_pv.ix[:1,"JPY"]

    for ccy in ccy_instr[ ccy_instr != 'USD' ].unique():
        instr_ccy = ccy_instr[ccy_instr == ccy].index.values
        if ccy in cur_flip:
            port_cshf_pv[instr_ccy] = port_cshf_pv[instr_ccy] / risk_factors[ccy]
        else:
            port_cshf_pv[instr_ccy] = port_cshf_pv[instr_ccy] * risk_factors[ccy]

    #print port_cshf_pv.ix[:1,"JPY"]
    port_mtm_value = port_cshf_pv.sum(axis=0)[port.index]
    return port_mtm_value
```

10 Valuation with base scenario

In [29]:

```
#print port_valuation_slow( risk_factors_base ,port, instr_info, cshf_info)
#print port_valuation( risk_factors_base ,port, instr_info, cshf_info)
port_mtm_base = port_valuation(risk_factors=risk_factors_hist.ix[calc_date,],
                               port=port,
                               instr_info=instr_info,
                               cshf_info=cshf_info,
                               calc_date=calc_date)

port_mtm_base

risk_factors_hist.ix[calc_date,]
```

Out [29]:

```
AUD                   0.892200
CAD                   1.064000
CHF                   0.886600
CNY                   6.061700
EUR                   1.381600
GBP                   1.652200
JPY                 105.000000
NOK                   6.065700
NZD                   0.821900
SEK                   6.412700
SGD                   1.267100
GOVT_USD_USA_1m       0.000100
GOVT_USD_USA_3m       0.000700
GOVT_USD_USA_6m       0.001000
GOVT_USD_USA_1y       0.001300
GOVT_USD_USA_2y       0.003902
GOVT_USD_USA_3y       0.007724
GOVT_USD_USA_5y       0.017348
GOVT_USD_USA_7y       0.024643
GOVT_USD_USA_10y      0.031104
```

```
GOVT_USD_USA_20y      0.039064
GOVT_USD_USA_30y      0.042263
Name: 2013-12-30 00:00:00, dtype: float64
```

11 Valuation with simulated scenarios

In [30]:
```
port_mtm_sim = risk_factors_sim.apply(port_valuation, axis=1,
                                      port=port,
                                      instr_info=instr_info,
                                      cshf_info=cshf_info,
                                      calc_date=calc_date)

port_mtm_sim.to_csv(data_dir+'port_mtm_sim.csv')
port_mtm_sim
```

Out [30]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Columns: 383 entries, USD to CA135087B527
dtypes: float64(383)
```

Part VIII

Profit and losses distribution

Once we have the valuation for the portfolio in both, the base scenario, and the simulated scenarios, we can obtain the simulated distribution of profit and losses by obtaining the difference of them.

In [31]:
```
port_pl = port_mtm_base - port_mtm_sim
port_pl
```

Out [31]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000 entries, 0 to 999
Columns: 383 entries, USD to CA135087B527
dtypes: float64(383)
```

Part IX

Value-at-Risk (VaR) estimation

Finally, The Value at Risk represents the p-th percentile for the distribution of profit and losses.

This is a well-know and broadly used measure for assesing the risk in portfolios.

In [32]:
```
# Value-at-Risk Calculation
var_levels = np.array( [90,95,97.5,99] )
port_var = pd.DataFrame( 0., columns=(['portfolio']+port_pl.columns.values.tolist()),

for i in var_levels:
    port_var.ix[i,'portfolio'] = -np.percentile( port_pl.sum(axis=1), 100-i )
    port_var.ix[i,1:] = -np.percentile( port_pl.values, 100-i , axis=0 )

print 'Portfolio Value-at-Risk'
print port_var.ix[:,'portfolio']
print '\nAll instruments 95% Value-at-Risk'
print port_var.ix[95.,:]

port_var
```

```
Portfolio Value-at-Risk
90.0     60336.944782
95.0     74573.273333
97.5     92676.724189
99.0    106956.432642
Name: portfolio, dtype: float64

All instruments 95% Value-at-Risk
portfolio       74573.273333
USD                -0.000000
CAD              5366.173027
EUR             64284.022342
JPY              7702.179255
US912796AQ20        0.130946
US912796AR03        0.628322
US912796AW97        1.205395
US912796BA68        1.989758
US912796BE80        2.740046
US912796BJ77        3.590347
US912796BP38        2.190616
US912796BS76        0.023109
US912796BT59        2.184744
US912796BU23        0.157135
...
CA135087ZC17    55.293438
CA135087ZF48    70.429791
CA135087ZL16    60.409990
CA135087ZN71    48.177326
CA135087ZQ03    65.855110
CA135087ZR85    66.071877
CA135087ZW70    52.341666
CA135087ZX53    94.671512
CA135087ZY37    47.623946
CA135087A388    92.661270
CA135087A537    51.913526
CA135087A792    88.527981
CA135087A958    59.672396
CA135087B295    48.390606
CA135087B527    59.142763
Name: 95.0, Length: 384, dtype: float64
```

Out [32]:

```
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, 90.0 to 99.0
Columns: 384 entries, portfolio to CA135087B527
dtypes: float64(384)
```

In [33]:

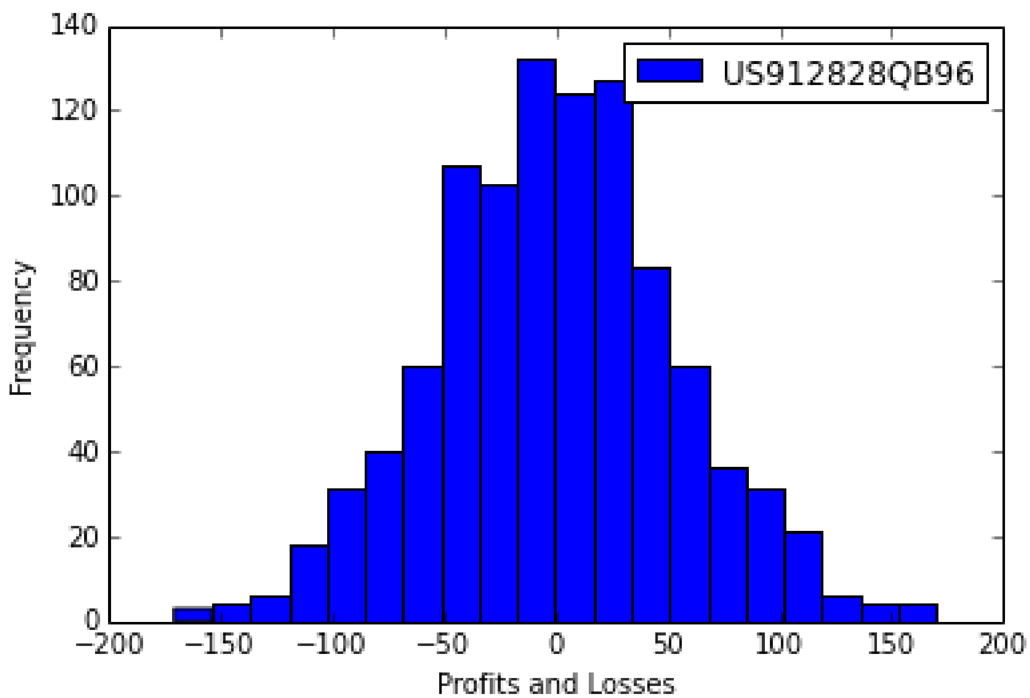
```
# If you want to get a pdf with the Value-at-Risk for selected instruments
# Change "if False:" by "if True:"
if False:
    pp = PdfPages(out_dir+'pl_instr.pdf')
    for i in range(1,382):
        plt.hist(port_pl.ix[:,port_pl.columns.values[i]].values, 20)
        plt.legend([port_pl.columns.values[i]])
        pp.savefig()
        plt.close()
    pp.close()
```

In [35]:

```
plt.hist(port_pl.ix[:,port_pl.columns.values[i]].values, 20)
plt.ylabel('Frequency')
plt.xlabel('Profits and Losses')
plt.legend([port_pl.columns.values[i]])
```

Out [35]:

```
<matplotlib.legend.Legend at 0x11a672710>
```

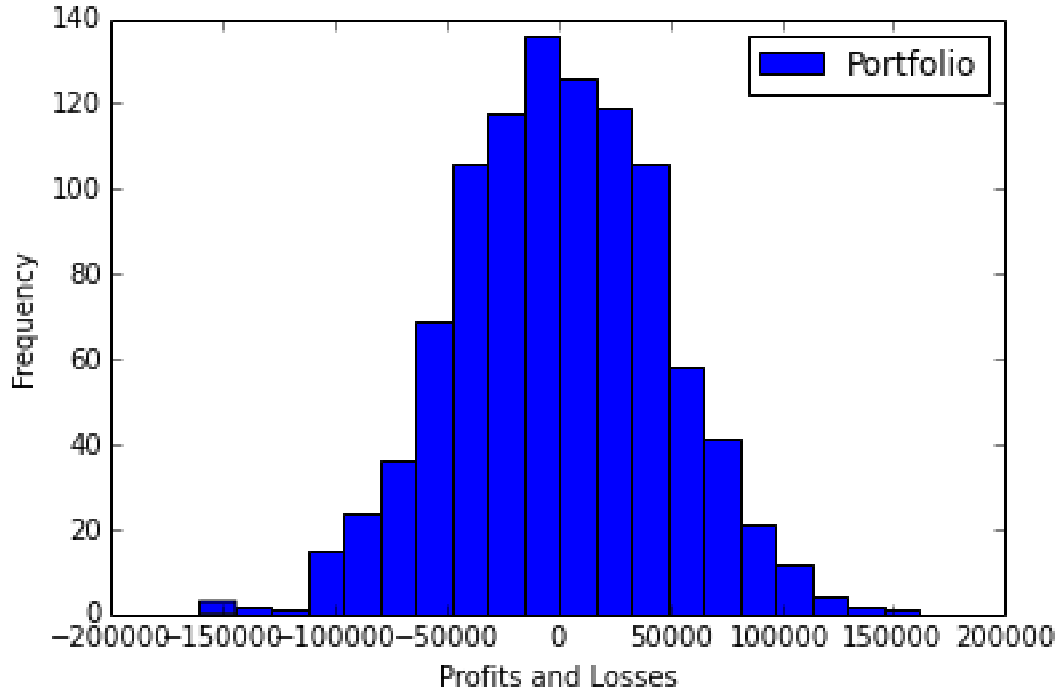


In [36]:

```
# Value-at-Risk for the Portfolio
plt.hist(port_pl.sum(axis=1), 20)
plt.ylabel('Frequency')
plt.xlabel('Profits and Losses')
plt.legend(['Portfolio'])
```

Out [36]:

```
<matplotlib.legend.Legend at 0x11a672690>
```



Part X

Back testing

12 For 2013

In order to measure the performance of this software, I did the following analysis:

1. Created a Portfolio incluiding fixed-income securities and currency deposits according to the following table:

Instrument type	Number of securities	Weight in portfolio
US Treasury Bills	30	10.04
US Treasury Notes 1-5 years	150	38.65
US Treasury Notes 5-10 years	130	21.12
US Agencies 1-3 years	60	2.63
CA Govt. Notes 0-3 years	20	1.79
Euro (EUR)	1	11.54
Sterling Pound (GBP)	1	14.22

2. Rebalancing the portfolio once each month
3. Calculate the daily market value for the portfolio and obtain the real profit and losses with observed market prices (full valuation, no factor-based valuation).

4. Calculate the daily Value at Risk for the aggregated portfolio and for each separated subportfolio specified in the table above.

5. Calculation from Jan 01, 2013 to Dec 31, 2013

CAUTION!!! This code take several hours to be finished!!!

In [19]:

```
start = datetime(2013, 12, 4)
end = datetime(2013, 12, 31)

var_lever = 95

port_var_backtest = {}
port_mtm_backtest = {}

for calc_date in pd.bdate_range(start,end):
    if calc_date in risk_factors_hist.index:
        port_file = 'port_' + str(calc_date.year) + '-' + str(calc_date.month).zfill(2
        port = pd.read_csv( data_dir + port_file , na_values=['','NA','na','NaN','NULL
        port = pd.Series(port.position.values,index=port.id_instr)

        print calc_date
        risk_factors_sim = simulate_risk_factors( n_sim=1000,
                                                  calc_date=calc_date,
                                                  risk_factors_hist=risk_factors_hist,
                                                  cur_flip=cur_flip, decay=0.96, seed=0
        port_mtm_base = port_valuation(risk_factors=risk_factors_hist.ix[calc_date,],
                                       port=port,
                                       instr_info=instr_info,
                                       cshf_info=cshf_info )
        port_mtm_sim = risk_factors_sim.apply(port_valuation, axis=1,
                                              port=port,
                                              instr_info=instr_info,
                                              cshf_info=cshf_info )
        port_pl_sim = port_mtm_base - port_mtm_sim
        port_mtm_backtest[calc_date] = port_mtm_base.sum()
        port_var_backtest[calc_date] = np.percentile( port_pl_sim.sum(axis=1), 100-var
```

```
2013-12-04 00:00:00
2013-12-05 00:00:00
2013-12-06 00:00:00
2013-12-09 00:00:00
2013-12-10 00:00:00
2013-12-11 00:00:00
2013-12-12 00:00:00
2013-12-13 00:00:00
2013-12-16 00:00:00
2013-12-17 00:00:00
2013-12-18 00:00:00
2013-12-19 00:00:00
2013-12-20 00:00:00
2013-12-23 00:00:00
2013-12-24 00:00:00
2013-12-26 00:00:00
2013-12-27 00:00:00
2013-12-30 00:00:00
2013-12-31 00:00:00
```

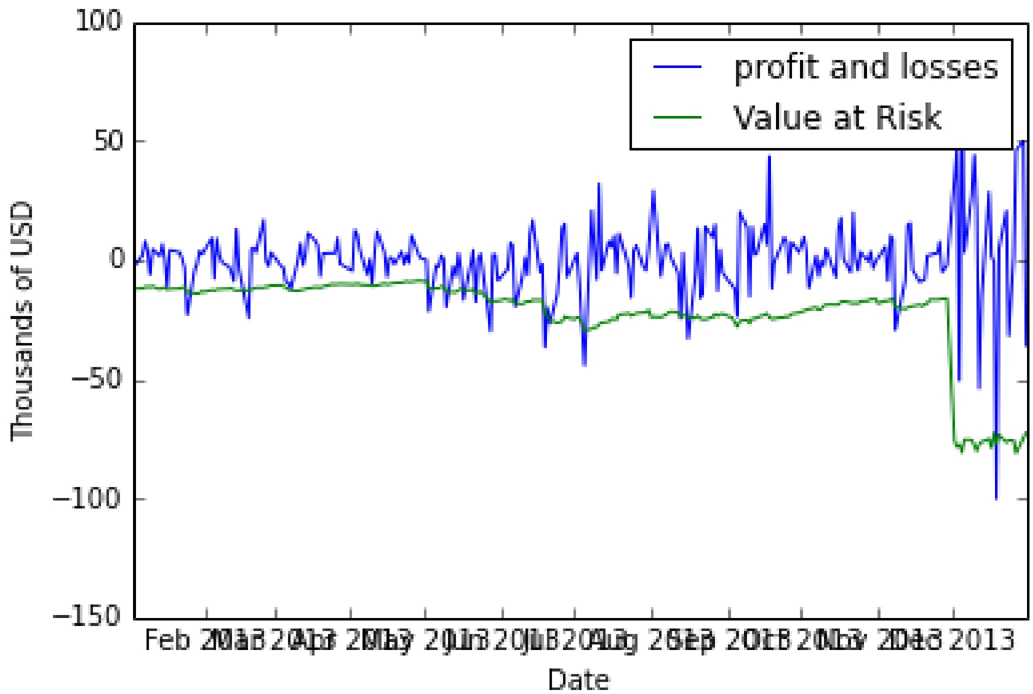
In [29]:
```
port_mtm_backtest = pd.Series(port_mtm_backtest)
port_var_backtest = pd.Series(port_var_backtest)

port_mtm_backtest.to_csv(out_dir+'backtesting_mtm.csv')
port_var_backtest.to_csv(out_dir+'backtesting_var.csv')
```

In [38]:
```
port_mtm_backtest = pd.read_csv( out_dir+'backtesting_mtm.csv' , na_values=['','NA','n
port_var_backtest = pd.read_csv( out_dir+'backtesting_var.csv' , na_values=['','NA','n

port_mtm_backtest['Date'] = pd.to_datetime(port_mtm_backtest['Date'])
port_var_backtest['Date'] = pd.to_datetime(port_var_backtest['Date'])

port_mtm_backtest = port_mtm_backtest.set_index('Date')['mtm']
port_var_backtest = port_var_backtest.set_index('Date')['var']

# Calculate observed profit and losses
port_pl_backtest = port_mtm_backtest - port_mtm_backtest.shift(1)

# Eliminate profit and losses from first day in month
# Rebalancing portfolio effect

port_pl_backtest = port_pl_backtest.drop( pd.to_datetime( [ '2013-02-01', '2013-03-01'
                                                           '2013-04-01', '2013-05-01',
                                                           '2013-07-01', '2013-08-01',
                                                           '2013-10-01', '2013-11-01',
port_pl_backtest = port_pl_backtest.dropna()
# print port_var_backtest
```

In [43]:
```
plt.plot(port_pl_backtest.index, port_pl_backtest.values/1000,
         port_var_backtest.index, port_var_backtest.values/1000 )
plt.ylabel('Thousands of USD')
plt.xlabel('Date')
plt.legend(['profit and losses','Value at Risk'])
```

Out [43]:

```
<matplotlib.legend.Legend at 0x11a86bc10>
```



The results from this Backtesting exercise for evaluating the historical performance of the 95% confidence Value-at-Risk shows that only about 6% of the observations fall below the estimated VaR.

This is a positive result, because it means that the metodology for estimating the maximum loss in 95% of the observed market scenarios is being acurate.

Part XI

Testing Suite

I realized that the fundamental parts in my code that are subject to errors and have to be tested in order to ensure a proper output are the following:

1. All the instruments defined in the portfolio are contained in the cashflows databases
2. Zero-coupon rates are being correctly calculated from yield to maturity values.
3. Valuation of a given instrument with a known set of rates is correct.

13 Test 1: Portfolio cashflows are known

In [10]:
```
%%file test_01_instr_cshf.py

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib
import zipfile
```

```
from lxml import etree
from scipy.interpolate import interp1d
from datetime import datetime, timedelta
from matplotlib.backends.backend_pdf import PdfPages

# Data directory #
data_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_c
out_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_ca

# Portfolio file #
port_file = 'port_2013-12.csv'

# Portfolio #
port = pd.read_csv( data_dir + port_file , na_values=['','NA','na','NaN','NULL'] )

port = pd.Series(port.position.values,index=port.id_instr)

# fixed-income instruments cashflows #
cshf_info_file = 'instr_cashflows.csv'
cshf_info = pd.read_csv( data_dir + cshf_info_file , na_values=['','NA','na','NaN','NU
cshf_info['Date'] = pd.to_datetime(cshf_info['Date'])
cshf_info = cshf_info.groupby(['id_instr','Date'])['value'].sum()

def test_1():

    instr_name_len = np.array( [ len(i) for i in  port.index.values ] )
    instr_port = port.index.values[ instr_name_len == 12 ]

    instr_port_defined = [ (i in cshf_info.unstack().index.values) for i in instr_port

    result = all(instr_port_defined)
    print 'Checking all instruments cashflows defined:', result
    assert result == True

test_1()
```

```
Writing test_01_instr_cshf.py
```

14 Test 2: Zero-coupon calculation

In [11]:

```
%%file test_02_rates.py

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib
import zipfile

from lxml import etree
from scipy.interpolate import interp1d
from datetime import datetime, timedelta
from matplotlib.backends.backend_pdf import PdfPages

# Data directory #
data_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_c
out_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_ca

# Portfolio file #
port_file = 'port_2013-12.csv'

# Portfolio #
port = pd.read_csv( data_dir + port_file , na_values=['','NA','na','NaN','NULL'] )
```

```
port = pd.Series(port.position.values,index=port.id_instr)

# fixed-income instruments cashflows #
cshf_info_file = 'instr_cashflows.csv'
cshf_info = pd.read_csv( data_dir + cshf_info_file , na_values=['','NA','na','NaN','NU
cshf_info['Date'] = pd.to_datetime(cshf_info['Date'])
cshf_info = cshf_info.groupby(['id_instr','Date'])['value'].sum()

def zero_from_yield_bootstrap( ytm_curve , nodes ):

    nodes_old = nodes.copy()
    nodes = np.append(0,nodes)
    ytm_curve = np.append(0,ytm_curve)

    nodes_new = np.arange(0,max(nodes)+0.5,0.5)
    nodes_new = np.append(nodes,nodes_new)
    nodes_new = np.sort(nodes_new)
    nodes_new = np.unique(nodes_new)

    f = interp1d(nodes, ytm_curve, kind='linear')
    ytm_new = f(nodes_new)
    ytm_new[0]=0

    ytm_new = pd.Series(ytm_new,index=nodes_new)
    zero_new = np.zeros_like(ytm_new)

    nodes_coupon = np.in1d(nodes_new,np.arange(0,max(nodes),0.5)+0.5)

    for node_i in nodes_new[nodes_coupon==False]:
        zero_new[node_i] = (1+ytm_new[node_i]*node_i) ** (1/node_i)-1
    zero_new[0] = 0


    for node_i in nodes_new[nodes_coupon]:
        cpn_i = ytm_new[node_i]/2
        zero_new[node_i] = - np.log( (1 - cpn_i * np.exp(-nodes_new[ nodes_new<node_i


    return zero_new[np.in1d(nodes_new,nodes_old)].values

def test_2():
    nodes = np.array(range(1,31),dtype=np.float64)
    ytm_curve_test = pd.Series( np.zeros_like(nodes)+0.05 , index=nodes)
    zero_curve_test = zero_from_yield_bootstrap( ytm_curve=ytm_curve_test.values , nod
    zero_curve_test = np.array( [ round(rate, 2) for rate in zero_curve_test ],dtype=n
    result = all( ytm_curve_test == zero_curve_test )

    print 'Checking zero rates:', result
    assert result == True

test_2()
```

```
Writing test_02_rates.py
```

15 Test 3-4: Instrument valuation

In [12]:

```
%%file test_03_valuation.py

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import urllib
```

```python
import zipfile

from lxml import etree
from scipy.interpolate import interp1d
from datetime import datetime, timedelta
from matplotlib.backends.backend_pdf import PdfPages

# Data directory #
data_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_c
out_dir = '/Users/Chris/Documents/26 UC Berkeley/03 Courses/STAT 222/stat_222_chris_ca

# Portfolio file #
port_file = 'port_2013-12.csv'

# Portfolio #
port = pd.read_csv( data_dir + port_file , na_values=['','NA','na','NaN','NULL'] )

port = pd.Series(port.position.values,index=port.id_instr)

# fixed-income instruments cashflows #
cshf_info_file = 'instr_cashflows.csv'
cshf_info = pd.read_csv( data_dir + cshf_info_file , na_values=['','NA','na','NaN','NU
cshf_info['Date'] = pd.to_datetime(cshf_info['Date'])
cshf_info = cshf_info.groupby(['id_instr','Date'])['value'].sum()

# instruments description #
instr_info_file = 'instr_description.csv'
instr_info = pd.read_csv( data_dir + instr_info_file , na_values=['','NA','na','NaN','

currencies = ['AUD', 'CAD', 'CHF', 'CLP', 'EUR', 'GBP', 'JPY', 'NOK', 'NZD', 'SEK', 'S
# Flip all currencies to dollars per curency
cur_usd = ['AUD', 'EUR', 'GBP', 'NZD']
cur_flip = list(set(currencies).difference(set(cur_usd)))


nodes = np.array([1,3,6],dtype=np.float64)
nodes = nodes/12
nodes = np.append(nodes, np.array([1,2,3,5,7,10,20,30],dtype=np.float64) )
nodes_names = ['GOVT_USD_USA_1m','GOVT_USD_USA_3m','GOVT_USD_USA_6m',
               'GOVT_USD_USA_1y','GOVT_USD_USA_2y','GOVT_USD_USA_3y',
               'GOVT_USD_USA_5y','GOVT_USD_USA_7y','GOVT_USD_USA_10y',
               'GOVT_USD_USA_20y','GOVT_USD_USA_30y']

def port_valuation( port, calc_date, risk_factors, instr_info, cshf_info):
    # Cash flows for bonds
    bonds_cshf = cshf_info.ix[ port.index.values ].unstack('id_instr')
    bonds_cshf = bonds_cshf[bonds_cshf.index>=calc_date]
    bonds_cshf = bonds_cshf/1000000
    bonds_cshf = bonds_cshf * port[bonds_cshf.columns]

    # Cash flows for currencies
    ccy_cshf = port[currencies].dropna()
    ccy_cshf = pd.DataFrame( ccy_cshf.values, index=ccy_cshf.index, columns=[calc_date
    if 'USD' in port.index:
        ccy_cshf['USD'] = port['USD']

    # cashflows for all the portfolio
    port_cshf = pd.merge( ccy_cshf, bonds_cshf, left_index=True, right_index=True, how
    port_cshf.dropna( how='all' )

    # Discount factors calculation
    discount = pd.Series( np.array(risk_factors[nodes_names].values) ,
                          index=[ calc_date + pd.DateOffset(days=x*365) for x in nodes
    discount = np.exp(-discount * nodes)
    discount = discount.set_value(calc_date, 1)
    discount = discount.reindex( index= discount.index.append(port_cshf.index).unique(
    discount = discount.sort_index()
```

```
    discount = discount.interpolate(method="time")
    discount = discount.reindex( index=port_cshf.index)

    # present value
    port_cshf_pv = discount * port_cshf

    # present value in USD
    ccy_instr = instr_info.ix[ instr_info.id_instr.isin(port.index), ['id_instr','curr
    ccy_instr = ccy_instr.set_index('id_instr').currency

    #print port_cshf_pv.ix[:1,"JPY"]

    for ccy in ccy_instr[ ccy_instr != 'USD' ].unique():
        instr_ccy = ccy_instr[ccy_instr == ccy].index.values
        if ccy in cur_flip:
            port_cshf_pv[instr_ccy] = port_cshf_pv[instr_ccy] / risk_factors[ccy]
        else:
            port_cshf_pv[instr_ccy] = port_cshf_pv[instr_ccy] * risk_factors[ccy]

    #print port_cshf_pv.ix[:1,"JPY"]
    port_mtm_value = port_cshf_pv.sum(axis=0)[port.index]
    return port_mtm_value

def test_3():
    risk_factors_test = pd.Series( [1.5]*len(currencies) ,
                              index=currencies)
    port_test = pd.Series( [1000]*len(currencies) ,
                              index=currencies)

    port_mtm_base = port_valuation(port=port_test,
                               calc_date=datetime.strptime('2013-12-30','%Y-%m-%d'
                               risk_factors=risk_factors_test,
                               instr_info=instr_info,
                               cshf_info=cshf_info )
    result = all(port_mtm_base[cur_flip] == 1000/1.5) and all(port_mtm_base[cur_usd] =

    print 'Checking currency valuation:', result
    assert result == True


def test_4():
    risk_factors_test = pd.Series( nodes * 0.05 ,
                              index=nodes_names)
    port_test = pd.Series( [1000] ,
                              index=['US912796BU23','US912796BV06'])

    port_mtm_base = port_valuation(port=port_test,
                               calc_date=datetime.strptime('2013-12-30','%Y-%m-%d'
                               risk_factors=risk_factors_test,
                               instr_info=instr_info,
                               cshf_info=cshf_info )

    result = all( np.array( [round(i,4) for i in port_mtm_base] ) ==  np.array( [999.8
    print 'Checking Bonds valuation:', result
    assert result == True
```

```
Writing test_03_valuation.py
```

16 Running the tests

In [13]:
```
!nosetests -v test_01_instr_cshf.py
```

```
test_01_instr_cshf.test_1 ... ok

----------------------------------------------------------------------
Ran 1 test in 4.465s

OK
```

In [14]:
```
!nosetests -v test_02_rates.py
```

```
test_02_rates.test_2 ... ok

----------------------------------------------------------------------
Ran 1 test in 0.016s

OK
```

In [15]:
```
!nosetests -v test_03_valuation.py
```

```
test_03_valuation.test_3 ... //anaconda/lib/python2.7/site-
packages/pandas/core/frame.py:3879: FutureWarning: TimeSeries
broadcasting along DataFrame index by default is deprecated. Please
use DataFrame.<op> to explicitly broadcast arithmetic operations along
the index
  FutureWarning)
ok
test_03_valuation.test_4 ... ok

----------------------------------------------------------------------
Ran 2 tests in 0.060s

OK
```

In [ ]:
```

```