Portfolio Risk Measurement of Crude Oil Futures

- by Chee-Foong on 17 Aug 2020

Summary

This analysis demonstrates how portfolio VaR can be calculated by historical simulation method. Each position in the portfolio is revalued based on historical scenarios on current market prices and rates. VaR is calculated on the simulated P&L of each position. Portfolio VaR is calculated on aggregated simulated P&L of all positions.

Historical simulation approach performs full revaluation of all positions on scenario rates. Consistent scenario rates generation process is of utmost importance here in order to maintain correlation across all factors/parameters passed in as inputs to valuation models.

Specifically, for future revaluation, prices of future contracts are not continuous. Number of days to maturity of each future contract reduces every other day. Scenario rates generation needs consistent continual time series. To convert them into consistent continual time series, we usually construct **Nearbys** or **Constant Maturity** price series for future prices.

Below demostrated the generation and creation of such time series and follow by Portfolio VaR calculaton using the Constant Maturity price series.

Data

Historical Data gathered from :

1. Crude Oil Future Prices - TurtleTrader.com (https://www.turtletrader.com/hpd/)
2. Crude Oil Spot Prices - datahub.io (https://datahub.io/core/oil-prices)

Reference

1. https://www.value-at-risk.net/futures-prices/ (https://www.value-at-risk.net/futures-prices/)
2. http://blog.smaga.ch/expected-shortfall-closed-form-for-normal-distribution/#:~:text=For%20those%20of%20you%20who,shortfall%20can%20be%20empirically%20estimated (http://blog.smaga.ch/expected-shortfall-closed-form-for-normal-distribution/#:~:text=For%20those%20of%20you%20who,shortfall%20can%20be%20empirically%20estimated).
3. https://medium.com/quaintitative/expected-shortfall-in-python-d049914e1e85 (https://medium.com/quaintitative/expected-shortfall-in-python-d049914e1e85)

Import Libraries

In [1]:
```
import time
import re
import random
import numpy as np
import pandas as pd

import seaborn as sns; sns.set()
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
%matplotlib inline

## Changing the default settings
pd.options.display.max_columns = 50
plt.rcParams.update({'figure.figsize':(15,6), 'figure.dpi':60})
plt.style.use('fivethirtyeight')

import warnings
warnings.filterwarnings('ignore')
```

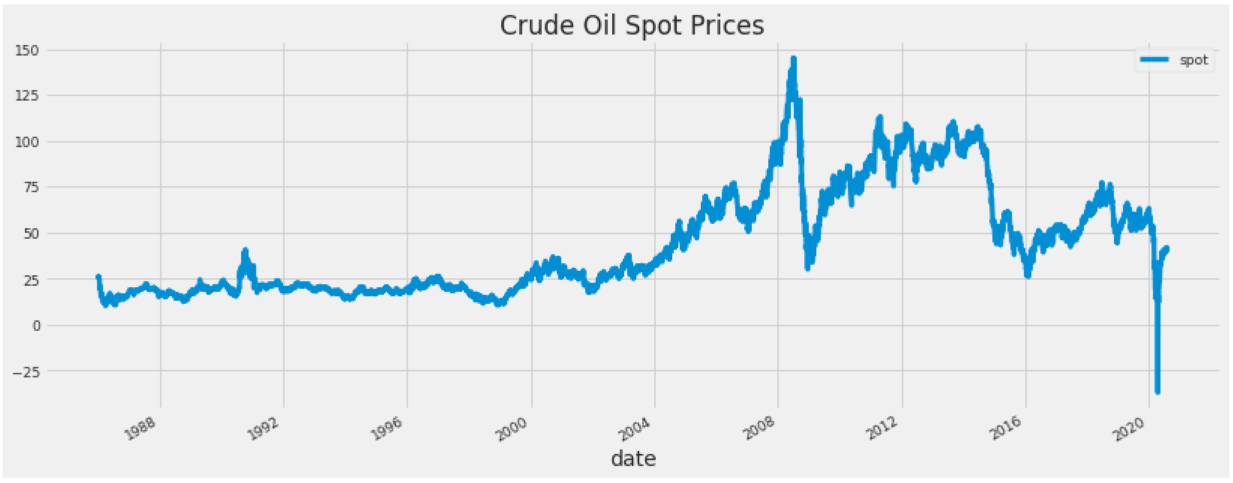
Loading Raw Prices

Process spot prices

In [2]:
```
PRICE_FOLDER = '../data/crude-oil/'
prices_spot = pd.read_csv(PRICE_FOLDER + 'SPOT.csv')
prices_spot.columns = ['date','spot']
prices_spot.date = pd.to_datetime(prices_spot.date)
prices_spot.set_index('date', inplace=True)
```

In [3]:
```
# prices_spot.head()
# prices_spot.tail()
```

In [4]:
```
prices_spot.plot()
plt.title("Crude Oil Spot Prices")
plt.show()
```



Process Future Prices

In [5]:
```
import os
from datetime import date
from dateutil.relativedelta import relativedelta
from tqdm.notebook import tqdm

filelist = []

for dirname, _, filenames in os.walk(PRICE_FOLDER):
    for filename in filenames:
        filelist.append(os.path.join(dirname, filename))
```

In [6]:
```
prices_full = []
filelist2 = []
filelist3 = []

print('Number of files to process: {}'.format(len(filelist)))

for file in tqdm(filelist):
    try:
        prices = pd.read_csv(file, header=None, dtype={0:'str'})
        prices.columns = ['date', 'open', 'high', 'low', 'close', 'volume', 'open_interest']
        prices.date = pd.to_datetime(prices.date, format='%y%m%d')
        maturity_date = max(prices.date) + relativedelta(days=1)
        prices['maturity_date'] = maturity_date
        prices['days_to_maturity'] = prices.maturity_date - prices.date
        prices['days_to_maturity'] = prices.days_to_maturity.dt.days
        prices_full.append(prices)
    except:
        filelist2.append(file)

for file in tqdm(filelist2):
    try:
        prices = pd.read_csv(file, dtype={0:'str'})
        prices.columns = ['date', 'open', 'high', 'low', 'close', 'volume', 'open_interest']
        prices.date = pd.to_datetime(prices.date, format='%m/%d/%Y')
        maturity_date = max(prices.date) + relativedelta(days=1)
        prices['maturity_date'] = maturity_date
        prices['days_to_maturity'] = prices.maturity_date - prices.date
        prices['days_to_maturity'] = prices.days_to_maturity.dt.days
        prices_full.append(prices)
    except:
        filelist3.append(file)

print('Number of files processed successfully: {}'.format(len(prices_full)))
prices_full = pd.concat(prices_full)
```

```
Number of files to process: 236


Number of files processed successfully: 235
```

Creating Nearby Future Prices

At any point in time, there will be contracts trading for several maturities. A first nearby is a time series comprising the price, at each point in time, of the nearest-to-maturity contract. The second nearby comprises the price, at each point in time, of the second nearest-to-maturity contract, etc.

In this analysis, we created three nearbys and compare them with the spot prices. Expect the time series to be correlated.

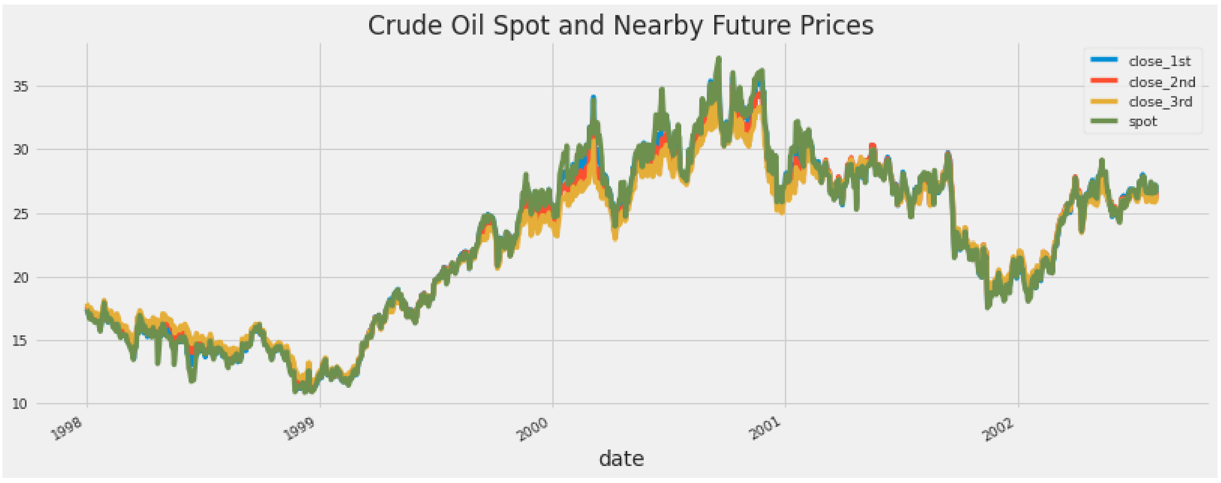
In [7]:
```
prices_full.sort_values(['date','maturity_date'], inplace=True)
```

In [8]:
```
def generate_nearby(prices_full, N):

    prices_nearby = []

    for dt in prices_full.date.unique():
        px = prices_full[(prices_full.date == dt) & (prices_full.days_to_maturity >= 14)]
        if px.shape[0] > N:
            prices_nearby.append(px.iloc[[N-1]])

    prices_nearby = pd.concat(prices_nearby)
    prices_nearby.set_index('date', inplace=True)

    return prices_nearby[['close']]
```

In [9]:
```
prices_nearby_first = generate_nearby(prices_full, 1)
prices_nearby_second = generate_nearby(prices_full, 2)
prices_nearby_third = generate_nearby(prices_full, 3)
```

In [10]:
```
prices_nearby = prices_nearby_first.merge(prices_nearby_second, left_index=True,
                                          right_index=True)
prices_nearby = prices_nearby.merge(prices_nearby_third, left_index=True,
                                          right_index=True)
prices_nearby.columns = ['close_1st','close_2nd','close_3rd']
```

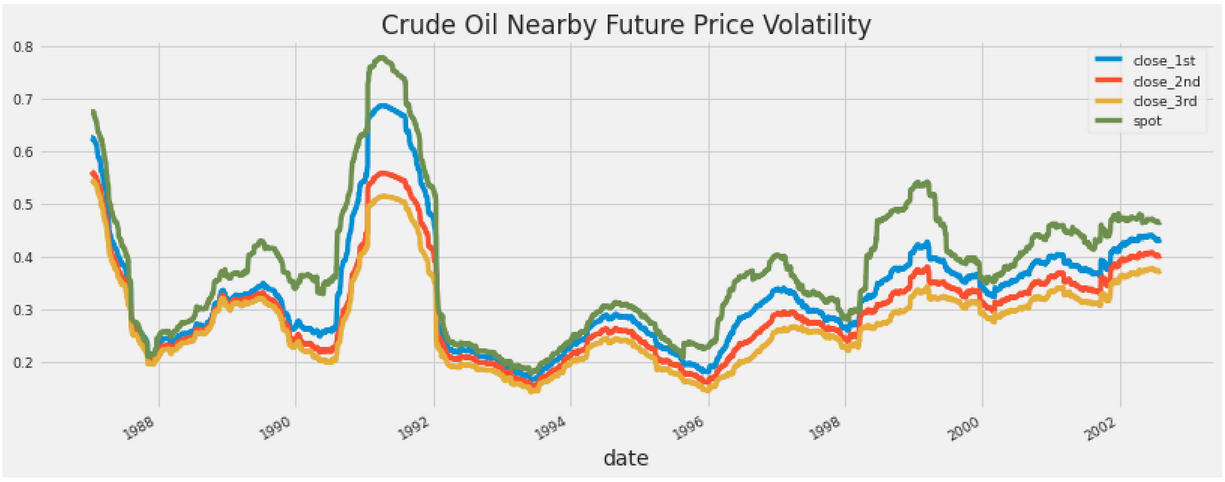
In [11]:
```
prices_nearby = prices_nearby.join(prices_spot, how='inner')
prices_nearby['1998':'2002'].plot()
plt.title('Crude Oil Spot and Nearby Future Prices')
plt.show()
```



Once the nearby time series are created, the price volaility of each series can be calculated. We observe lower volatility for future contracts with longer time to maturity.

In [12]:
```
price_vol = prices_nearby.pct_change().dropna().rolling(252).std().dropna()*np.sqrt(252)
```

In [13]:
```
price_vol.plot()
plt.title('Crude Oil Nearby Future Price Volatility')
plt.show()
```



Generating Constant Maturity Price Curve

As an alternative to nearbys, futures price data can be merged into continual time series as constant-maturity futures prices. A constant-maturity price series indicates, for each time t, an interpolated price reflecting a specific time-to-expiration that is constant over time.

Defining the list to constant maturity period we wish to calculate a price as follows:

Period	Number of Days	Period	Number of Days
0 day	Spot	1 day	1
3 months	90	1 week	5
6 months	180	1 month	30
1 year	360	2 months	60
2 years	720		

Linear interpolation is performed to impute a price at these constant maturity periods.

In [14]:
```
prices_mean = pd.DataFrame(prices_full.groupby(['date','days_to_maturity'])['close'].mean())
prices_mean.sort_index(level=['date', 'days_to_maturity'], inplace=True)
```

In [15]:
```
prices_spot['days_to_maturity'] = 0
prices_spot.rename(columns={'spot':'close'}, inplace=True)
prices_spot.reset_index(inplace=True)
prices_spot.set_index(['date', 'days_to_maturity'], inplace=True)
```

In [16]:
```
prices_mean = pd.concat([prices_mean, prices_spot])
prices_mean.sort_index(level=['date', 'days_to_maturity'], inplace=True)
```

In [17]:
```python
from scipy import interpolate

idx = pd.IndexSlice

cm_points = [0,1,5,30,60,90,180,360]
cm = pd.DataFrame(cm_points, columns=['days_to_maturity'])
cm['close'] = np.nan

const_mat = []

for dt in tqdm(prices_mean.index.levels[0]):
    sample = prices_mean.loc[idx[dt,:],:]
    sample.reset_index(inplace=True)

    if sample.shape[0] > 5:
        f = interpolate.interp1d(sample.days_to_maturity, sample.close, fill_value="extrapolate")
#         f = interpolate.CubicSpline(sample.days_to_maturity, sample.close, extrapolate=True)
#         f = interpolate.InterpolatedUnivariateSpline(sample.days_to_maturity, sample.close, k=3, ext=
3)

        new_sample = pd.concat([sample[['days_to_maturity','close']], \
                                cm[~cm.days_to_maturity.isin(sample.days_to_maturity)]]) \
                                .sort_values('days_to_maturity') \
                                .reset_index(drop=True)

        new_sample.close = f(new_sample.days_to_maturity)
        new_sample = new_sample[new_sample.days_to_maturity.isin(cm_points)]
        new_sample['date'] = dt
        new_sample['num_of_contracts'] = sample.shape[0]
        const_mat.append(new_sample)
#     elif sample.shape[0] == 1:
#         print('Warning! Only one record in sample: {}'.format(dt))

#         new_sample = cm.copy()
#         new_sample['close'] = sample.iloc[0].close
#         new_sample['date'] = dt
#         new_sample['num_of_contracts'] = sample.shape[0]
#         const_mat.append(new_sample)
    else:
        pass

const_mat = pd.concat(const_mat)
```

In [18]:
```python
# const_mat[const_mat.close < 0].days_to_maturity.unique().tolist()
```

In [19]:
```python
days_to_maturity_todrop = const_mat[const_mat.close < 0].days_to_maturity.unique().tolist()
const_mat = const_mat[~const_mat.days_to_maturity.isin(days_to_maturity_todrop)]

const_mat.drop('num_of_contracts', axis=1, inplace=True)

const_mat.set_index(['date', 'days_to_maturity'], inplace=True)
const_mat = const_mat.unstack('days_to_maturity')
```

Generating Constant Maturity Volatility Curve

All in a single line of code:

1. Calculating the daily returns of each period
2. Calculating the rolling volatilily
3. Annualising the volatility

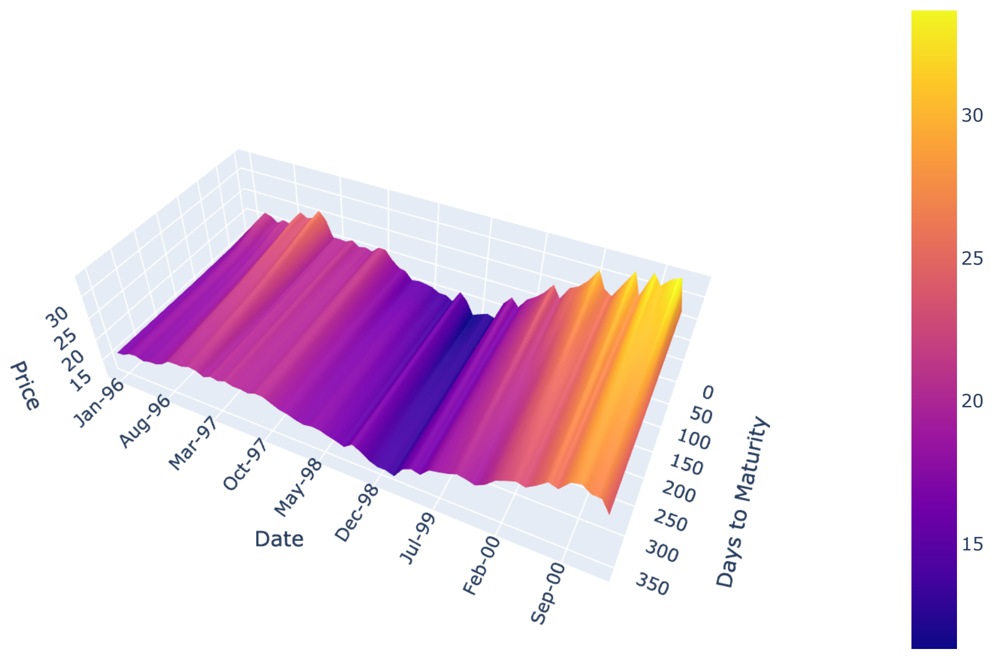
In [20]:
```python
const_mat_vol = const_mat.pct_change().dropna().rolling(252).std().dropna()*np.sqrt(252)
```

3D Constant Maturity Price and Volaility Plots

Given that the raw data between 1996 and 2000 are more complete, we choose to plot the surfaces between 1996 and 2000.

```
In [21]: import plotly.graph_objects as go

         plotdata = const_mat['1996':'2000'].resample('M').fillna('ffill')

         x = plotdata.columns.levels[1]
         y = plotdata.index.strftime('%b-%y').tolist()
         z = plotdata.values

         fig = go.Figure(data=[go.Surface(x=x,y=y,z=z)])

         fig.update_layout(title='Crude Oil Futures Price Curve', autosize=True,
                           width=800, height=600,
                           scene=dict(
                               xaxis = dict(
                                   title = 'Days to Maturity',
                               ),
                               yaxis = dict(
                                   title = 'Date',
                               ),
                               zaxis = dict(
                                   title = 'Price',
                               ),
                               camera=dict(
                                   eye=dict(
                                       x= 4,
                                       y= 2,
                                       z= 4
                                   )
                               ),
                               aspectmode='manual',
                               aspectratio=dict(x=2,y=4,z=1),
                            ),
                           margin=dict(l=65, r=50, b=65, t=90))

         fig.show()
```

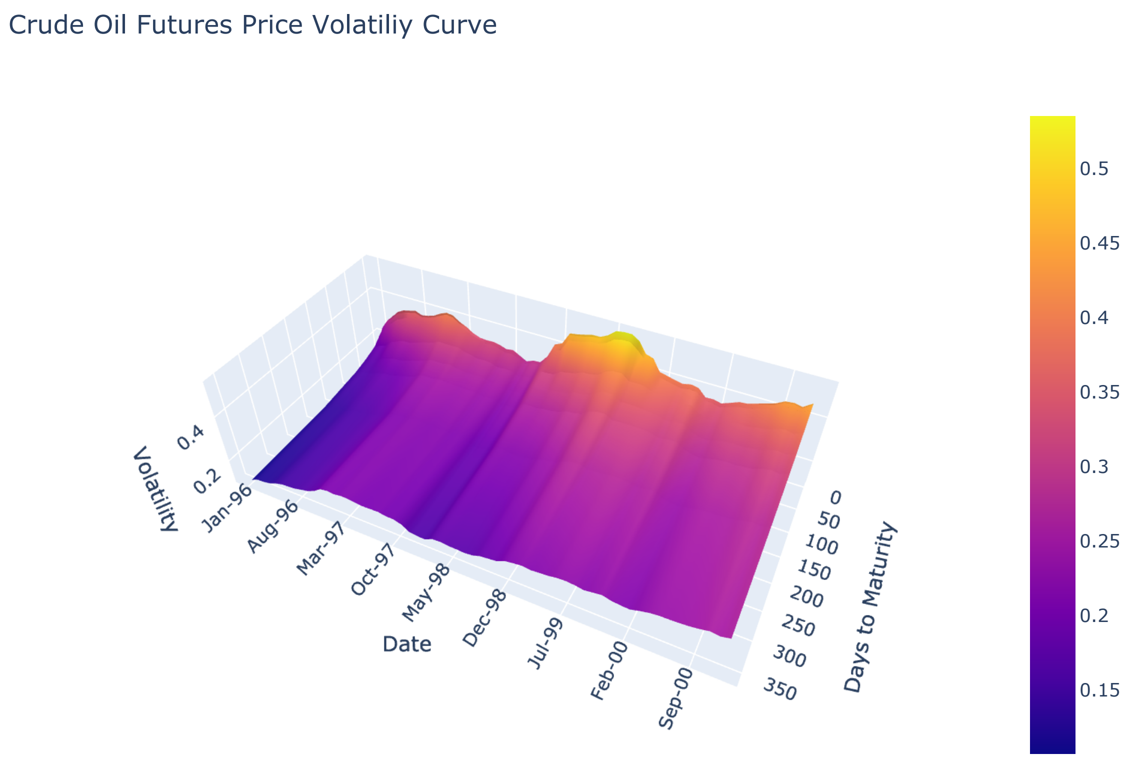
Crude Oil Futures Price Curve



Observations:

1. Backwardation between Jan-Mar 1997 and Sep 1999 to Dec 2000
2. Contango between Dec 1998 to Feb 1999
3. Price bottomed about Feb 1999 and have been increasing to a high in Nov 2000

In [22]:
```
plotdata = const_mat_vol['1996':'2000'].resample('M').fillna('ffill')

x = plotdata.columns.levels[1]
y = plotdata.index.strftime('%b-%y').tolist()
z = plotdata.values

fig = go.Figure(data=[go.Surface(x=x,y=y,z=z)])

fig.update_layout(title='Crude Oil Futures Price Volatiliy Curve', autosize=True,
                  width=800, height=600,
                  scene=dict(
                      xaxis = dict(
                          title = 'Days to Maturity',
                      ),
                      yaxis = dict(
                          title = 'Date',
                      ),
                      zaxis = dict(
                          title = 'Volatility',
                      ),
                      camera=dict(
                          eye=dict(
                              x= 4,
                              y= 2,
                              z= 4
                          )
                      ),
                      aspectmode='manual',
                      aspectratio=dict(x=2,y=4,z=1),
                   ),
                  margin=dict(l=65, r=50, b=65, t=90))

# fig.show()
```



Observations:

1. Volatility tends to be higher for contracts nearing maturity maybe due to higher trading volume
2. Higher volatility early 1999 due to lower oil prices

In [23]:
```
# new_sample.merge(sample[['days_to_maturity','close']], how='left', on=['days_to_maturity', 'days_to_ma
turity'])
# sns.lineplot(new_sample.days_to_maturity, new_sample.close)
# plt.show()
```

Portfolio VaR Computation

Below explore that various ways to compute VaR at position and portfolio levels. Positions of the portfolio are fabricated to demostrate how to calculate VaR. Positions in the portfolios are all crude oil futures long and short positions of different maturities and different quantities.

Each position is revalued and PnL calculated based on 1-day historical simulation. The VaR and Expected Shortfall (CVaR) of each position is then calculated based on the following methods:

1. Parametric
2. Cutoff

Portfolio VaR is calculated based sum of positions PnL at each historical scenario. Other risk analytics like risk decomposition, VaR of various horizon periods are also calculated.

In [24]:
```
position = pd.read_csv('../data/futures.csv')
```

In [25]:
```
position
```

Out[25]:

	id	position_id	child_id	position_date	position_desc	security	contract	asset_type	sub_asset_type	txn_price	quantity	contract_s
0	Pos_1_0	1	0	20-May-99	Long 98 CLN99 contracts	CLN99	CL	FUT	NaN	15.5	98	1
1	Pos_2_0	2	0	20-May-99	Short 100 CLQ99 contracts	CLQ99	CL	FUT	NaN	17.0	100	1
2	Pos_4_0	4	0	20-May-99	Short 65 CLU99 contracts	CLU99	CL	FUT	NaN	17.8	65	1
3	Pos_5_0	5	0	20-May-99	Short 60 CLV99 contracts	CLV99	CL	FUT	NaN	18.7	60	1
4	Pos_6_0	6	0	20-May-99	Long 58 CLZ99 contracts	CLZ99	CL	FUT	NaN	15.7	58	1
5	Pos_7_0	7	0	20-May-99	Long 76 CLF00 contracts	CLF00	CL	FUT	NaN	16.8	76	1
6	Pos_8_0	8	0	20-May-99	Long 75 CLJ00 contracts	CLJ00	CL	FUT	NaN	16.3	75	1
7	Pos_9_0	9	0	20-May-99	Short 73 CLN99 contracts	CLN99	CL	FUT	NaN	16.9	73	1

In [26]:
```
futures = position[position.asset_type == 'FUT']
futures = futures.dropna(axis=1)
futures.maturity_date = pd.to_datetime(futures.maturity_date)
futures.position_date = pd.to_datetime(futures.position_date)
futures['days_to_maturity'] = futures['maturity_date'] - futures['position_date']
futures.days_to_maturity = futures.days_to_maturity.dt.days
```

In [27]:
```
## Extract the current future price based on position and maturity date of the future contract
def getCBR(pos_date, mat_date=None):
    px = prices_full.reset_index()
    if not mat_date:
        out = px[(px.date == pos_date)].close
    else:
        out = px[(px.date == pos_date) & (px.maturity_date == mat_date)].close
    return out.iloc[0]

## Calculate the base exposure of the future contract
def calcBaseExposure_Futures(position, longshort=False):
    pos_date = position.position_date
    mat_date = position.maturity_date
    contract_size = position.contract_size
    quantity = position.quantity
    long = bool(position.long)

    cbr = getCBR(pos_date, mat_date)

    notionalbase = cbr * contract_size * quantity

    if longshort:
        if not long:
            notionalbase *= -1
    return notionalbase
```

In [28]:
```python
## Generate 1-year daily simulated PnLs based on historical simulation
def generatePnL_Futures(position):
    days_to_maturity = position.days_to_maturity
    pos_date = position.position_date
    mat_date = position.maturity_date
    contract_size = position.contract_size
    quantity = position.quantity
    long = bool(position.long)

    cbr = getCBR(pos_date, mat_date)

    cm = const_mat.stack('days_to_maturity')
    end_date = pos_date + relativedelta(days=-1)
    start_date = end_date + relativedelta(years=-1)
    cm = cm.loc[idx[start_date:end_date,:],:].reset_index().set_index(['date','days_to_maturity'])

    past_prices = []

    for dt in cm.index.levels[0]:
        sample = cm.loc[idx[dt,:],:]
        sample.reset_index(inplace=True)

        f = interpolate.interp1d(sample.days_to_maturity, sample.close, fill_value="extrapolate")
#         f = interpolate.CubicSpline(sample.days_to_maturity, sample.close, extrapolate=True)

        past_prices.append({'date':dt, 'close':f(days_to_maturity)})

    past_prices = pd.DataFrame(past_prices)
    past_prices.set_index('date', inplace=True)

    scenario_rates = past_prices.pct_change().dropna()
    simulated_prices = (1 + scenario_rates) * cbr

    simulated_pnl = (simulated_prices - cbr) * quantity * contract_size
    if not long: simulated_pnl *= -1

    return simulated_pnl

## Calculate risk decomposition of each position to the portfolio
def calcRiskDecomp(pnl, base='Portfolio'):
    risk_decomp_all = []
    base_pnl = pnl[base]

    for index, pos in pnl.iteritems():
        risk_decomp = pos.cov(base_pnl) / base_pnl.cov(base_pnl)
        risk_decomp_all.append({'pos':index, 'risk_decomp':risk_decomp})

    risk_decomp_all = pd.DataFrame(risk_decomp_all)
    risk_decomp_all.set_index('pos', inplace=True)
    return risk_decomp_all
```

In [29]:
```python
from scipy.stats import norm

# norm.cdf(1.645) # Cumulative Density Function
# norm.ppf(1-0.05) # Inverse Cumulative Density Function
# norm.pdf(1.645) # Probability Density Function

## Calculate VaR and Expected Shortfall - Parametric Method
def calcVaR_Parametric(pnl, confidence=0.95):
    mu, sigma = pnl.mean(), pnl.std()

    Z = abs(norm.ppf(confidence))
    VaR = mu - Z * sigma
    ES = mu - (1-confidence)**-1 * norm.pdf(norm.ppf(confidence)) * sigma
    return -VaR, -ES

## Calculate VaR and Expected Shortfall - Cutoff Method
def calcVaR_Cutoff(pnl, confidence=0.95):
    pnl_series = pd.DataFrame(pnl.sort_values().values, columns=['pnl'])
    pnl_series['prob'] = 1/pnl_series.shape[0]
    pnl_series['cumprob'] = pnl_series['prob'].cumsum()

    f = interpolate.interp1d(pnl_series.cumprob, pnl_series.pnl, fill_value="extrapolate")

    varCutOff = f(1-confidence)
    esCutoff = pnl_series.loc[pnl_series.pnl < varCutOff].pnl.mean()
    return -varCutOff, -esCutoff
```

Revaluation via Historial Simulation and Generate Simulated PnLs

In [30]:
```
pnls = []
for index, pos in futures.iterrows():
    pnl = generatePnL_Futures(pos)
    pnl.columns = [pos.id]
    pnl.index.name = 'scenario'
    pnls.append(pnl)

pnls = pd.concat(pnls, axis=1)
pnls = pnls.astype('float')
pnls['Portfolio'] = pnls.sum(axis=1)
```

Calculate 1-day VaR - Parametric Method

In [31]:
```
parametric = pd.DataFrame(list(pnls.apply(lambda col: calcVaR_Parametric(col), axis=0)),
                          columns=['var95_Parametric','es95_Parametric'])
parametric.index = pnls.columns
```

Calculate 1-day VaR - Cutoff Method

In [32]:
```
cutOff = pd.DataFrame(list(pnls.apply(lambda col: calcVaR_Cutoff(col), axis=0)), columns=['var95_CutOff'
,'es95_CutOff'])
cutOff.index = pnls.columns
```

Calculate Risk Decomposition of each position

In [33]:
```
riskdecomp = calcRiskDecomp(pnls)
```

Calculate Base Exposure of each position

In [34]:
```
baseexposure = []
for index, pos in futures.iterrows():
    exposure = calcBaseExposure_Futures(pos)
    baseexposure.append({'pos':pos.id, 'base_exposure':exposure})

baseexposure = pd.DataFrame(baseexposure).set_index('pos')
```

Calculate VaR (assuming mean zero) and across various horizon period - Parametric Method

In [35]:
```
CONFIDENCE = 0.95

Z = abs(norm.ppf(1-CONFIDENCE))

# var95 = abs(pnls.mean() - Z * pnls.std())
var95 = Z * pnls.std()
es95 = -pnls[pnls < -var95].mean()

var95 = pd.concat([var95, es95], axis=1)
var95.columns = ['var95_1d','es95_1d']

var95['var95_5d'] = var95.var95_1d * np.sqrt(5)
var95['var95_1m'] = var95.var95_1d * np.sqrt(252/12)
var95['var95_3m'] = var95.var95_1d * np.sqrt(252/4)
var95['var95_1y'] = var95.var95_1d * np.sqrt(252)
```

Preparing position description

In [36]:
```
position_desc = futures[['id','position_desc']]
position_desc.set_index('id', inplace=True)
```

Results

Analytics	Calculation Method	Description
id	-	Position Identifier
position_desc	-	Position Description
base_exposure	Current Price x Quantity x Contract Size	Base Exposure
var95_CutOff	CutOff	1 day 95% VaR
es95_CutOff	CutOff	1 day 95% Expected Shortfall
var95_Parametric	Parametric	1 day 95% VaR
es95_Parametric	Parametric	1 day 95% Expected Shortfall
var95_1d	Parametric	1 day 95% VaR (assuming zero expected return)
es95_1d	Parametric	1 day 95% Expected Shortfall (assuming zero expected return)
var95_5d	Parametric	5 days 95% VaR (assuming zero expected return)
var95_1m	Parametric	1 month 95% VaR (assuming zero expected return)
var95_3m	Parametric	3 months 95% VaR (assuming zero expected return)
var95_1y	Parametric	Annualised 95% VaR (assuming zero expected return)
risk_decomp	Covariance(Position PnL, Portfolio PnL) / Variance(Portfolio PnL)	Risk decomposition or risk contribution of the position to the total portfolio.

Observations:

1. Portfolio VaR is relatively lower than individual portfolio because the portfolio comprise a balanced mix of long and short future contracts.
2. VaR calculations are consistent across various computation methodologies.
3. Expected shortfalls are more conservative than standard VaR measures.
4. From the results, positions **Pos_1_0** and **Pos_7_0** are two highest contributors to total portfolio risk.

In [37]:
```
results = pd.concat([position_desc, baseexposure, cutOff, parametric, var95, riskdecomp], axis=1)
results.index.name = 'id'
# results['var_percent'] = results.var95 / results.baseexposure
results
```

Out[37]:

	position_desc	base_exposure	var95_CutOff	es95_CutOff	var95_Parametric	es95_Parametric	var95_1d	es95_1d	v
id									
Pos_1_0	Long 98 CLN99 contracts	1672860.0	70385.681426	86671.445979	65443.905245	82415.275214	66805.806229	84155.900290	149382
Pos_2_0	Short 100 CLQ99 contracts	1703000.0	60867.735581	80950.626549	62481.358964	78113.888343	61535.617364	80950.626549	137597
Pos_4_0	Short 65 CLU99 contracts	1101750.0	35353.364630	47920.638557	37004.923542	46294.354544	36566.754990	47920.638557	81765
Pos_5_0	Short 60 CLV99 contracts	1011600.0	29684.576009	41017.805596	31847.463039	39862.017132	31548.351652	44431.920972	70544
Pos_6_0	Long 58 CLZ99 contracts	969180.0	27131.506424	36851.168399	26966.276090	33847.346450	27086.525953	36851.168399	60567
Pos_7_0	Long 76 CLF00 contracts	1266160.0	34812.138094	46639.623534	34023.308584	42693.369357	34128.676769	45712.138780	76314
Pos_8_0	Long 75 CLJ00 contracts	1239000.0	30395.205499	41780.773294	30117.964398	37758.802075	30077.260834	40893.179093	67254
Pos_9_0	Short 73 CLN99 contracts	1198660.0	23226.258622	33565.264305	26255.778788	32968.673835	26424.523569	36397.178498	59087
Portfolio	NaN	NaN	5803.712384	7601.374211	6239.770671	7833.190302	6272.309382	8419.154559	14025

Concluding Notes

1. Above analysis can be extended to portfolios with non-linear positions like options by including option valuation models and scenario generation of other risk factors like interest rates. One can include foreign exchange risk also if the reporting currency is different from the trading currency.
2. Hypothetical and stress scenarios can also be included to calculate VaR on extreme events.
3. Senstivity measures like delta, gamma, theta, rho, vega of each position can be derived by model revaluation.
4. Additional analytics like implied volatility can be also be calculated for option pricing and market making.
5. Marginal and Incremental VaR can be quickly calculated by making relevant adjustments to the portfolio positions like adding new positions, scaling down position size, etc.

```
In [ ]:
```