FBDQA-课程编号-01510313

iCenter
清华大学iCenter

衍生品估值库DX__Library

期权估值应用

智能系统实验室

清华大学iCenter

导学


衍生品估值库DX__Library及其应用
资产定价基本定理
风险中性测度
衍生品分析库
市场环境配置
风险中立折现
三大基本模型
几何布朗运动（GBM）
跳跃扩散（JD）
平方根扩散（SRD）
期权估值方面的应用
欧式期权
美式期权
投资组合

目录

- 参考文献：[德] 伊夫·希尔皮斯科（Yves Hilpisch） 著，姚军 译，Python金融大数据分析，人民邮电出版社，2015.（第15-18章）
- notebook： 15_DX_Library_ipynb.ipynb

资产定价基本定理

- 中心思想是鞅(Martingale)测度——也就是从折现后风险因素(随机过程)中消去漂移的概率测度。
- 换句话说，在鞅测度下，所有风险因素随无风险短期利率漂移——而不随包含某种无风险短期利率之上风险溢价的任何其他市场利率漂移。

简单示例

- 考虑只有今天和明天两个交易日期的一个简单经济环境，其中包含一种高风险资产——股票和一种无风险资产——债券。

- 今天的债券价格为10 美元，明天清偿价格为10 美元(0 利率)。股票今天价格为10 美元，明日售出价格为20 美元和0 美元的概率分别为60% 和40% 。

现实世界的概率测度

- 债券的风险收益为0 。股票的预期收益率为（0.6 *20+0.4 *0）/10-1 = 0.2 。这就是补偿股票风险特性的风险溢价。

- 现在考虑行权价为15 美元的看涨期权。它有60%的概率取得5 元收益，在其他情况下收益为0 ，其公允价值是多大呢?

- 我们可以取得期望值，并将结果值折现(这里使用的0 利率)。这种方法得到的价值是0.6*5=3 美元

风险中性的概率测度

- 通过可交易证券的组合复制期权收益。
- 很容易验证，购买0.25 份股票就可以完全复制期权的收益(在60% 的情况下.我们的收益为0.25*20=5 美元)。四分之一的股票成本仅为2.5 美元而非3 美元，在现实世界概率测度下的预期会过高估计期权的价值。
- 现实世界的测度隐含着20%的股票风险溢价，是因为股票中隐含的风险(盈利100%或者损失100%) 无法依靠多样化或者对冲来消除。另一方面，有某种投资组合可以没有任何风险地复制期权的收益。这也意味着，卖出这种期权可以完全对冲任何风险 。这种由期权和对冲头寸组成的完全对冲投资组合必须得到等于无风险利率的收益，以避免套利机会。

风险中性的概率测度

- “只”需要改变概率，使风险资产(股票)随无风险利率(0)漂移。很显然，为两种情形同样设定50% 的(鞍)测度可以实现这一点;计算式为（0.5*20+0.5*0）/10-1 =0 。
- 现在，在新的鞍测度下计算期权收益，可以得到正确的(无套利)公允价值: 0.5*5+0.5*0=2.5美元。

$$\frac{20\times p+0\times(1-p)}{10}-1=0$$

解得：p=0. 5，即为风险中性概率测度

风险中性的概率测度

- 根据资产定价基本定理，下面三个陈述等价:

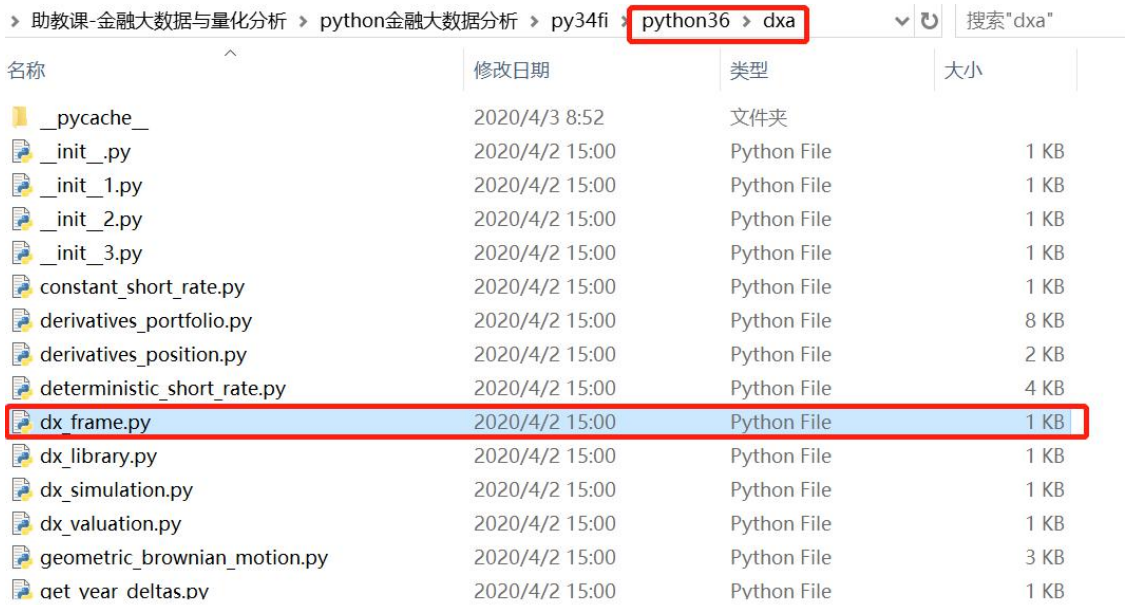
- 市场模型中没有套利机会（唯一价格）
- 鞅测度（风险中性测度）集合不为空
- 一致线性定价系统集合不为空

目录

- 参考文献：[德] 伊夫·希尔皮斯科（Yves Hilpisch） 著，姚军 译，Python金融大数据分析，人民邮电出版社，2015.（第15-18章）
- notebook： 15_DX_Library_ipynb.ipynb

衍生品分析库

- 为市场环境配置和风险中立折现开发了一个python模块：dx_frame.py
- from dx_frame import *
- 导入基本的numpy,pandas,datetime等库。
- 风险中立估值是整个DX 分析库的首要目标。




```
#DX library

from pylab import plt
plt.style.use('seaborn')
import matplotlib as mpl
mpl.rcParams['font.family'] = 'serif'

import numpy as np
import pandas as pd
import datetime as dt
import sys
sys.path.append('../python36/dxa')
#建立衍生品分析库

from dx_frame import *
```

风险中立折现

```
#Risk-Neutral Discounting

dates = [dt.datetime(2015, 1, 1), dt.datetime(2015, 7, 1), dt.datetime(2016, 1, 1)]
deltas = [0.0, 0.5, 1.0]
csr = constant_short_rate('csr', 0.05)
#短期无风险利率固定为5%
csr.get_discount_factors(dates)
#获取折现因子
array([[datetime.datetime(2015, 1, 1, 0, 0), 0.951229424500714],
       [datetime.datetime(2015, 7, 1, 0, 0), 0.975510338765722 8],
       [datetime.datetime(2016, 1, 1, 0, 0), 1.0]], dtype=object)

deltas = get_year_deltas(dates)
deltas
array([0.        , 0.49589041, 1.        ])
csr.get_discount_factors(deltas, dtobjects=False)
```

配置市场环境

```
#Market Environment

me_gbm = market_environment('me_gbm', dt.datetime(2015, 1, 1))

me_gbm.add_constant('initial_value', 36.)
me_gbm.add_constant('volatility', 0.2)
me_gbm.add_constant('final_date', dt.datetime(2015, 12, 31))
me_gbm.add_constant('currency', 'EUR')
me_gbm.add_constant('frequency', 'M')
me_gbm.add_constant('paths', 10000)
#初始值设置
me_gbm.add_curve('discount_curve', csr)
```

目录

- 参考文献：[德] 伊夫·希尔皮斯科（Yves Hilpisch） 著，姚军 译，Python金融大数据分析，人民邮电出版社，2015.（第15-18章）
- notebook： 15_DX_Library_ipynb.ipynb

几何布朗运动

- Black 和Scholes (1973)的开创性工作引入期权定价文献
- 在金融现实中得到越来越多的证实，是期权和衍生品估值的一个基准过程。
- 它的随机微分方程：

$$dS_t = rS_t dt + \sigma S_t dZ_t$$

也可以看成：$\frac{dS_t}{S_t} = rdt + \sigma dZ_t$

分别是drift项和diffusion项

```
#Geometric Brownian Motion

from dx_frame import *
me_gbm = market_environment('me_gbm', dt.datetime(2015, 1, 1))

#市场环境设置
me_gbm.add_constant('initial_value', 36.)
me_gbm.add_constant('volatility', 0.2)
me_gbm.add_constant('final_date', dt.datetime(2015, 12, 31))
me_gbm.add_constant('currency', 'EUR')
me_gbm.add_constant('frequency', 'M')

# monthly frequency (respcective month end)

me_gbm.add_constant('paths', 10000)
csr = constant_short_rate('csr', 0.06)
me_gbm.add_curve('discount_curve', csr)
#初始值设置
```

几何布朗运动

- 生成和检查time_grid 。注意到，在time__grid数组对象中有13 个datetime 对象(对应年的所有月末pricing_date):

```
#Geometric Brownian Motion

from geometric_brownian_motion import geometric_brownian_motion
gbm = geometric_brownian_motion('gbm', me_gbm)
gbm.generate_time_grid()
gbm.time_grid

array([datetime.datetime(2015, 1, 1, 0, 0),
       datetime.datetime(2015, 1, 31, 0, 0),
       datetime.datetime(2015, 2, 28, 0, 0),
       datetime.datetime(2015, 3, 31, 0, 0),
       datetime.datetime(2015, 4, 30, 0, 0),
       datetime.datetime(2015, 5, 31, 0, 0),
       datetime.datetime(2015, 6, 30, 0, 0),
       datetime.datetime(2015, 7, 31, 0, 0),
       datetime.datetime(2015, 8, 31, 0, 0),
       datetime.datetime(2015, 9, 30, 0, 0),
       datetime.datetime(2015, 10, 31, 0, 0),
       datetime.datetime(2015, 11, 30, 0, 0),
       datetime.datetime(2015, 12, 31, 0, 0)], dtype=object)
```

几何布朗运动

- 接下来，我们可以求得模拟金融工具价值:
- 以及还要生成更高波动率下的金融工具价值:

```
#Geometric Brownian Motion

%time paths_1 = gbm.get_instrument_values()
#paths_1 volatility = 0.2

gbm.update(volatility=0.5)
%time paths_2 = gbm.get_instrument_values()
import matplotlib.pyplot as plt
%matplotlib inline
plt.figure(figsize=(10, 6))
p1 = plt.plot(gbm.time_grid, paths_1[:, :10], 'b')
p2 = plt.plot(gbm.time_grid, paths_2[:, :10], 'r-.')
plt.grid(True)
l1 = plt.legend([p1[0], p2[0]],
                ['low volatility', 'high volatility'], loc=2)
plt.gca().add_artist(l1)
plt.xticks(rotation=30);
#画出高低波动率的几何布朗运动的图
```

几何布朗运动


low volatility
high volatility
60
50
40
30
20
10
2015-01
2015-03
2015-05
2015-07
2015-09
2015-11
2016-01

跳跃扩散

- 用geometric_brownian_motion类，实现Merton (1976)描述的跳跃扩散模型就很简单了
- Merton 跳跃扩散模型的随机微分方程：

$$dS_t = (r - r_J)S_t dt + \sigma S_t dZ_t + J_t S_t dZN_t$$

```
#Jump Diffusion

me_jd = market_environment('me_jd', dt.datetime(2015, 1, 1))
me_jd.add_constant('lambda', 0.3)
#跳跃密度（概率）
me_jd.add_constant('mu', -0.75)
#预期跳跃规模
me_jd.add_constant('delta', 0.1)
#跳跃规模的标准差
me_jd.add_environment(me_gbm)
#添加GBM 模拟类的完整环境
from jump_diffusion import jump_diffusion
jd = jump_diffusion('jd', me_jd)
#导入跳跃扩散的库，设置基本参数
```

跳跃扩散

- 我们的目标仍然是比较两组不同的路径，例如，改变跳跃概率:

```
#Jump Diffusion

%time paths_3 = jd.get_instrument_values()
jd.update(lamb=0.9)
#改变跳跃概率
%time paths_4 = jd.get_instrument_values()
plt.figure(figsize=(10, 6))
p1 = plt.plot(gbm.time_grid, paths_3[:, :10], 'b')
p2 = plt.plot(gbm.time_grid, paths_4[:, :10], 'r-.')
plt.grid(True)
l1 = plt.legend([p1[0], p2[0]],
                ['low intensity', 'high intensity'], loc=3)
plt.gca().add_artist(l1)
plt.xticks(rotation=30);
#改变参数，画出跳跃扩散的图
```

跳跃扩散

- 在图中可以看到，低密度的情况下跳跃较少，而高密度的情况下有多处跳跃:
- 用于模拟市场发生较大外部冲击时候，股票价格出现跳跃的情形。例如：00年互联网泡沫破灭，08年次贷危机，20年新冠疫情（美股三次熔断）

平方根扩散

- 模拟的第三种随机过程：Cox、Ingersoll 和Ross (1985) 用于建立随机短期利率模型的平方根扩散。

- 平方根扩散模型的随机微分方程：

$$dx_t = k(\theta - x_t)dt + \sigma\sqrt{x_t}dZ_t$$

```
#Square-Root Diffusion

me_srd = market_environment('me_srd', dt.datetime(2015, 1, 1))
me_srd.add_constant('initial_value', .25)
me_srd.add_constant('volatility', 0.05)
me_srd.add_constant('final_date', dt.datetime(2015, 12, 31))
me_srd.add_constant('currency', 'EUR')
me_srd.add_constant('frequency', 'W')
me_srd.add_constant('paths', 10000)

me_srd.add_constant('kappa', 4.0)
#设置均值回归因子
me_srd.add_constant('theta', 0.2)
#设置过程长期均值
me_srd.add_curve('discount_curve', constant_short_rate('r', 0.0))
from square_root_diffusion import square_root_diffusion
srd = square_root_diffusion('srd', me_srd)
srd_paths = srd.get_instrument_values()[:, :10]
#导入单位根扩散的库，设置基本参数
```

平方根扩散

```
#Square-Root Diffusion

plt.figure(figsize=(10, 6))
plt.plot(srd.time_grid, srd.get_instrument_values()[:, :10])
plt.axhline(me_srd.get_constant('theta'), color='r', ls='--', lw=2.0)
plt.grid(True)
plt.xticks(rotation=30);
#画出单位根扩散的图
```

平方根扩散

- 下图显示单一模拟路径平均回归到长期均值theta (虚线), 说明了均值回归特性:

目录

- 参考文献：[德] 伊夫·希尔皮斯科（Yves Hilpisch） 著，姚军 译，Python金融大数据分析，人民邮电出版社，2015.（第15-18章）
- notebook： 15_DX_Library_ipynb.ipynb

欧式期权

- 期权和衍生品估值很久以来都属于华尔街的“火箭科学家”的领域。然而，通过蒙特卡洛模拟等数值方法，这些模型的应用通常不像理论模型本身那么复杂。
- 对于欧式期权来说，一般采用风险中立的估值方法。
- 对于美式期权来说则稍微复杂一些，往往采用最小二乘蒙特卡洛模拟(LSM)算法。

```
#European Options

from dx_simulation import *
me_gbm = market_environment('me_gbm', dt.datetime(2015, 1, 1))
me_gbm.add_constant('initial_value', 36.)
#设置初始值36
me_gbm.add_constant('volatility', 0.2)
me_gbm.add_constant('final_date', dt.datetime(2015, 12, 31))
me_gbm.add_constant('currency', 'EUR')
me_gbm.add_constant('frequency', 'M')
me_gbm.add_constant('paths', 10000)
csr = constant_short_rate('csr', 0.06)
me_gbm.add_curve('discount_curve', csr)
gbm = geometric_brownian_motion('gbm', me_gbm)

#配置市场环境
```

(注：火箭科学家是指拥有物理学或者与数学相关的学科博士学位的人们)

欧式期权

```
#European Options

me_call = market_environment('me_call', me_gbm.pricing_date)
me_call.add_constant('strike', 40.)
#设置行权价
me_call.add_constant('maturity', dt.datetime(2015, 12, 31))
#设置到期日
me_call.add_constant('currency', 'EUR')
payoff_func = 'np.maximum(maturity_value - strike, 0)'
#内在价值
from valuation_mcs_european import valuation_mcs_european
eur_call = valuation_mcs_european('eur_call', underlying=gbm,
                    mar_env=me_call, payoff_func=payoff_func)
%time eur_call.present_value()
2.140776
%time eur_call.delta()
0.5148
%time eur_call.vega()
14.2782
#计算期权价值
```

欧式期权

```
#European Options

%%time
s_list = np.arange(34., 46.1, 2.)
p_list = []; d_list = []; v_list = []
for s in s_list:
    eur_call.update(initial_value=s)
    p_list.append(eur_call.present_value(fixed_seed=True))
    d_list.append(eur_call.delta())
    v_list.append(eur_call.vega())
from plot_option_stats import plot_option_stats
%matplotlib inline
plot_option_stats(s_list, p_list, d_list, v_list)

#底层标的的初试价格不同时，期权价值的变化
```

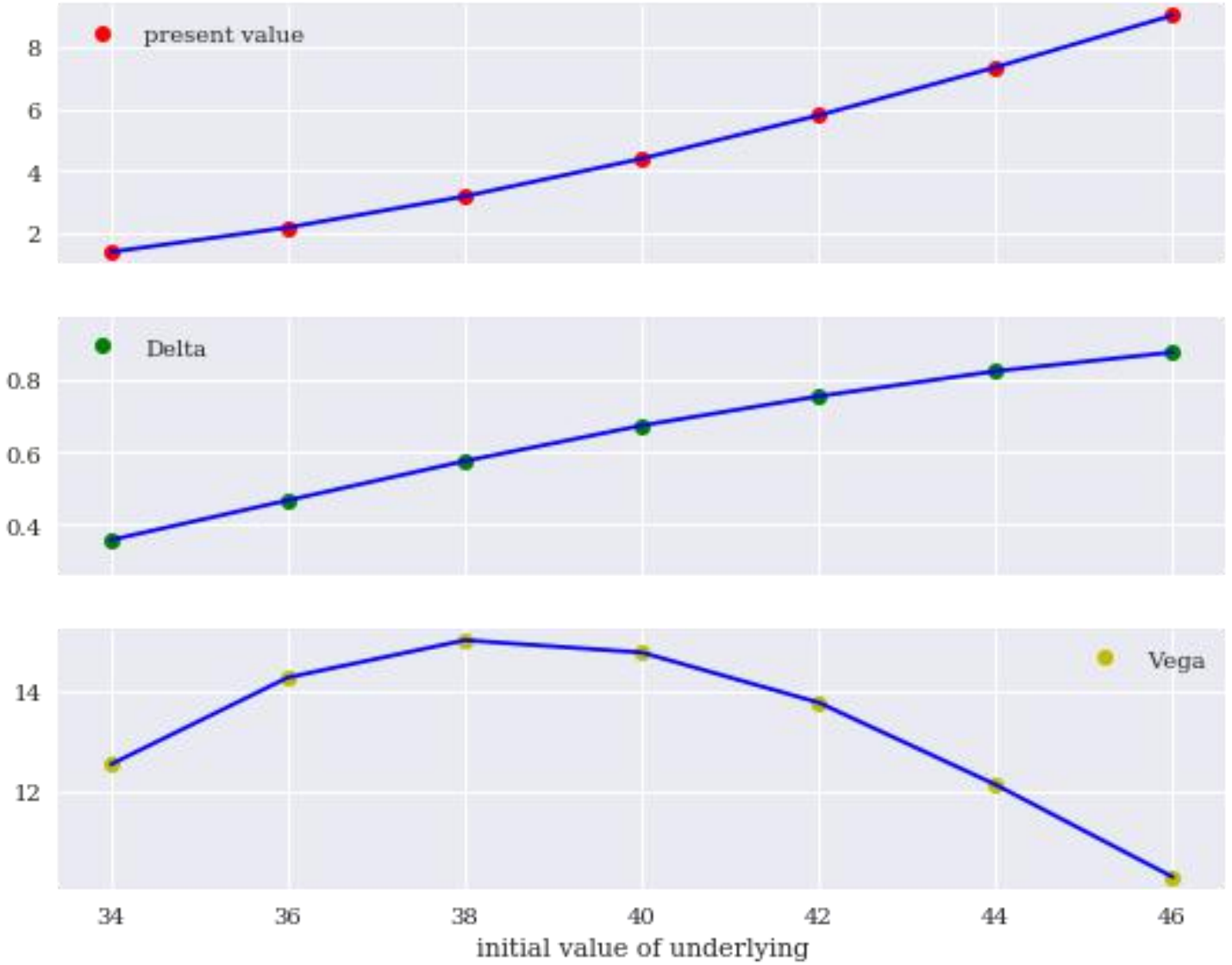
欧式期权

- 期权的Delta 值定义为期权现值对标的当前价值S_0的一阶偏微分：

$$\Delta=\frac{\partial V(\bullet)}{\partial S_0}$$

- 期权的Vega 值定义为在当前(即期)波动率σ_0下现值的一阶偏微分：

$$\Delta=\frac{\partial V(\bullet)}{\partial \sigma_0}$$



美式期权

- 美式期权的估值比欧式期权要复杂得多。
- 美式期权的估值通常利用倒推法实现——从到期日开始估算美式期权的持续价值并倒推出现值。
- 一般采用最小二乘蒙特卡洛模拟的方法（LSM）

```
#American Options

from dx_simulation import *
me_gbm = market_environment('me_gbm', dt.datetime(2015, 1, 1))
me_gbm.add_constant('initial_value', 36.)
#设置初始价值为36
me_gbm.add_constant('volatility', 0.2)
me_gbm.add_constant('final_date', dt.datetime(2015, 12, 31))
me_gbm.add_constant('currency', 'EUR')
me_gbm.add_constant('frequency', 'W')
me_gbm.add_constant('paths', 50000)
csr = constant_short_rate('csr', 0.06)
me_gbm.add_curve('discount_curve', csr)
gbm = geometric_brownian_motion('gbm', me_gbm)

#配置市场环境
```

美式期权

- 期权类型是一种有收益的美式看跌期权。
- 行权期为一年，行权价始终为40。
- 美式看跌期权估值花费的时间比欧式期权长得多。我们不仅增加了路径的数量和估值的频率，由于向后推导和每个推导步骤的回归，该算法的计算要求也更高。

```
#American Options

payoff_func = 'np.maximum(strike - instrument_values, 0)'
me_am_put = market_environment('me_am_put', dt.datetime(2015, 1, 1))
me_am_put.add_constant('maturity', dt.datetime(2015, 12, 31))
me_am_put.add_constant('strike', 40.)
me_am_put.add_constant('currency', 'EUR')
from valuation_mcs_american import valuation_mcs_American
am_put = valuation_mcs_american('am_put', underlying=gbm,
                    mar_env=me_am_put, payoff_func=payoff_func)

#标的资产仍然是根据GBM模型模拟的股票
%time am_put.present_value(fixed_seed=True, bf=5)

#构建看跌美式期权
```

美式期权

- 计算不同初始价格，不同波动率下美式看跌期权的现值，并以表格的形式呈现出来。

```
#American Options

%%time
ls_table = []
for initial_value in (36., 38., 40., 42., 44.):
    for volatility in (0.2, 0.4):
        for maturity in (dt.datetime(2015, 12, 31),
                         dt.datetime(2016, 12, 31)):
            am_put.update(initial_value=initial_value,
                          volatility=volatility,
                          maturity=maturity)
            ls_table.append([initial_value,
                             volatility,
                             maturity,
                             am_put.present_value(bf=5)])
#底层标的的不同的初始价格
```

美式期权

```
#American Options

print("S0  | Vola | T | Value")
print(22 * "-")
for r in ls_table:
    print("%d  | %3.1f  | %d | %5.3f" %
          (r[0], r[1], r[2].year - 2014, r[3]))

#计算看跌美式期权价值
am_put.update(initial_value=36.)
am_put.delta()
am_put.vega()
```

S0	Vola	T	Value
36	0.2	1	4.444
36	0.2	2	4.769
36	0.4	1	7.000
36	0.4	2	8.378
38	0.2	1	3.210
38	0.2	2	3.645
38	0.4	1	6.066
38	0.4	2	7.535
40	0.2	1	2.267
40	0.2	2	2.778
40	0.4	1	5.203
40	0.4	2	6.753
42	0.2	1	1.554
42	0.2	2	2.099
42	0.4	1	4.459
42	0.4	2	6.046
44	0.2	1	1.056
44	0.2	2	1.618
44	0.4	1	3.846
44	0.4	2	5.494

投资组合

- 期权的投资组合组合估值需要加入以下因素：非冗余性、相关性、头寸

```
#Portfolios

#这是欧式看涨期权和美式看跌期权的投资组合
portfolio.get_statistics(fixed_seed=False)
portfolio.get_statistics(fixed_seed=False)[['pos_value', 'pos_delta', 'pos_vega']].sum()
  # aggregate over all positions
pos_value    27.489253
pos_delta     1.271000
pos_vega     73.308100
```

	name	quant.	value	curr.	pos_value	pos_delta	pos_vega
0	am_put_pos	3	4.460280	EUR	13.38084	-2.0406	30.5181
1	eur_call_pos	5	2.814638	EUR	14.07319	3.3605	42.7900

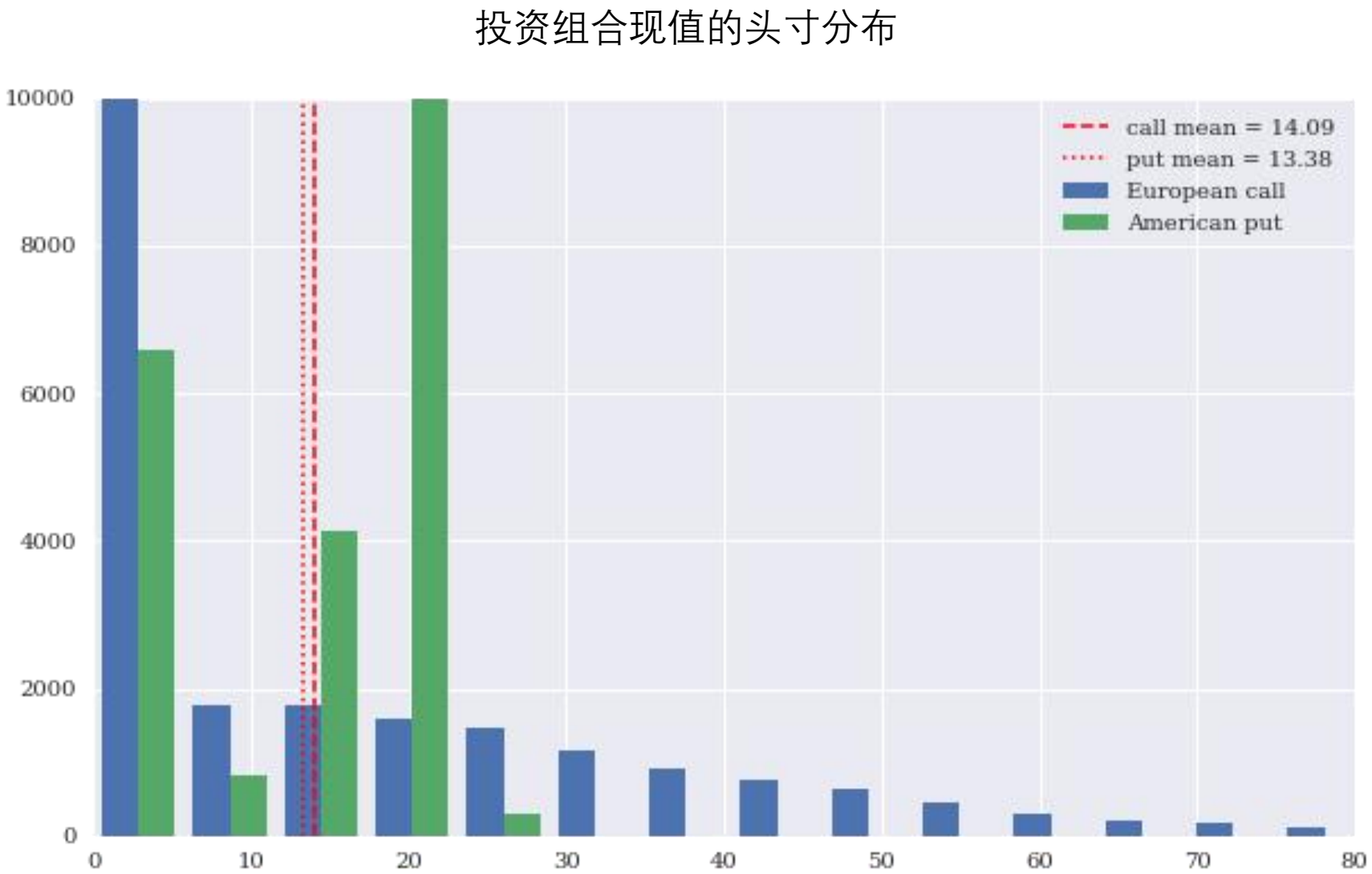
投资组合

```
#Portfolios

plt.figure(figsize=(10, 6))
plt.hist([pv1, pv2], bins=25,
         label=['European call', 'American put']);
plt.axvline(pv1.mean(), color='r', ls='dashed',
            lw=1.5, label='call mean = %4.2f' % pv1.mean())
plt.axvline(pv2.mean(), color='r', ls='dotted',
            lw=1.5, label='put mean = %4.2f' % pv2.mean())
plt.xlim(0, 80); plt.ylim(0, 10000)
plt.legend();

#期权投资组合的各自仓位价值
```

投资组合


投资组合现值的头寸分布
call mean = 14.09
put mean = 13.38
European call
American put
10000
8000
6000
4000
2000
0
0
10
20
30
40
50
60
70
80

投资组合

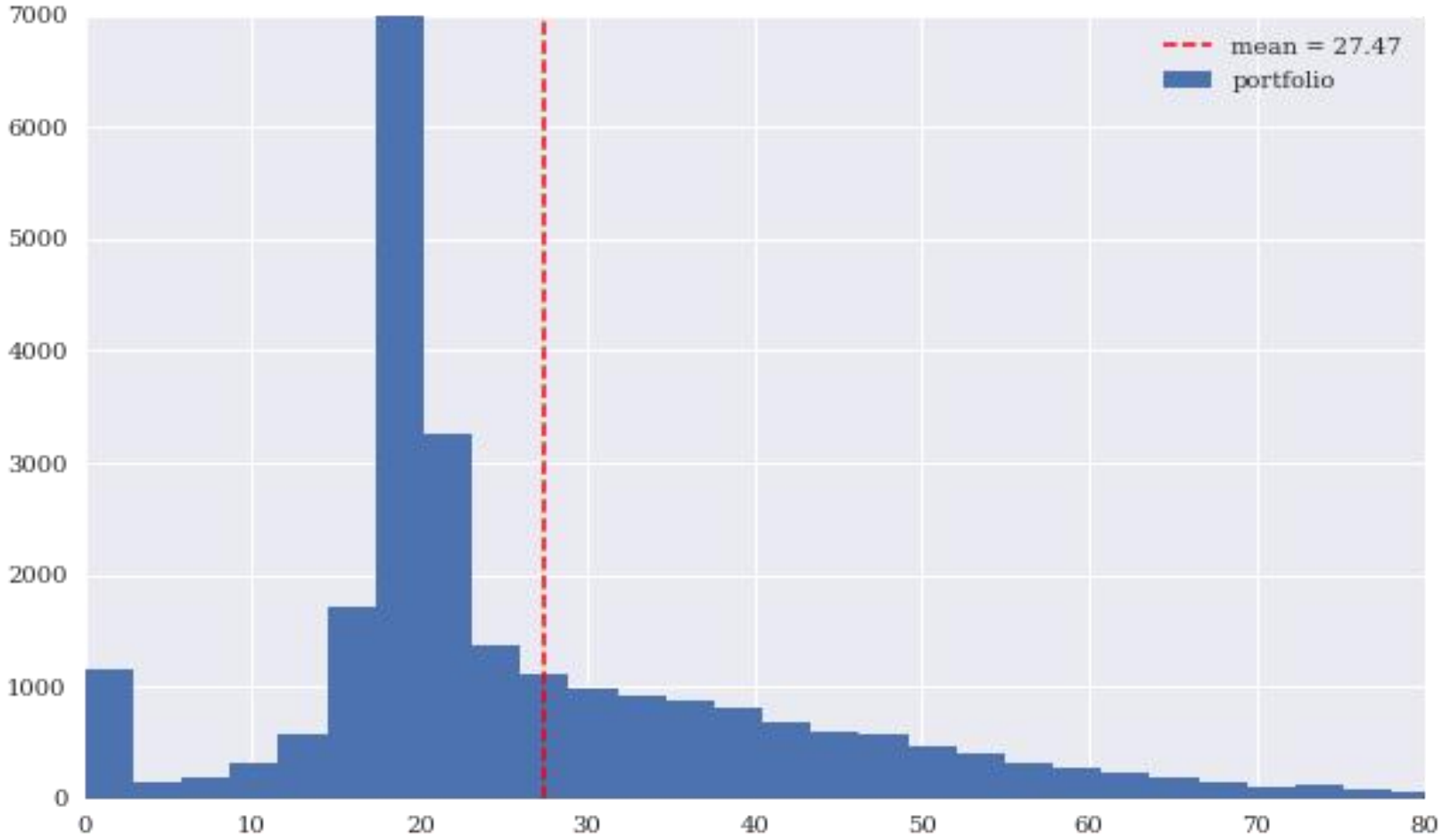
```
#Portfolios

pvs = pv1 + pv2
plt.figure(figsize=(10, 6))
plt.hist(pvs, bins=50, label='portfolio');
plt.axvline(pvs.mean(), color='r', ls='dashed',
            lw=1.5, label='mean = %4.2f' % pvs.mean())
plt.xlim(0, 80); plt.ylim(0, 7000)
plt.legend();

#期权投资组合的总价值
```

投资组合

- 右图是投资组合现值的全部频率分布。
- 在图中可以清晰地看到组合看涨和看跌期权的多样化对冲效应。
- 投资组合现值的平均值为27.47。



谢谢指正！