Problems1

April 8, 2016

0.1 General utils

In [1]:
```
import sys
print (sys.version)
if sys.version_info >= (3,4):
    print( "with enums" )
    from enum import Enum
    class CallPut(Enum):
        call = 1
        put = 2
else:
    print( "no enums" )
    class CallPut(object):
        __values__ = ['call', 'put']

        def __init__(self, t, s):
            self.value = t
            self.str_value = s


        def __eq__(self,y):
            if type(y) is not self.__class__:
                return False
            return self.value == y.value

        def __str__(self):
            return self.str_value

        @classmethod
        def __enum_init__(cls):
            for i in range(len(cls.__values__)):
                setattr(cls, cls.__values__[i], CallPut(i+1, cls.__values__[i]))


    CallPut.__enum_init__()

t = CallPut.call
print( "Call: %s; Booleans: %r, %r " %(t, CallPut.call == t, CallPut.put == t) )
print( "Wrong type: %r" % (t == 1) )
```

```
2.7.10 (default, Dec 28 2015, 04:30:58)
[GCC 5.2.1 20151010]
no enums
```

```
Call: call; Booleans: True, False
Wrong type: False
```

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline

import math as m
from time import time
from scipy import stats
import scipy as sp
```

In [3]:
```
def N(x):
    return stats.norm.cdf(x, 0.0, 1.0)

def NPrime(x):
    return stats.norm.pdf(x, 0.0, 1.0)

def bsm_d1(S, K, T, r, q, sigma):
    S = float(S)
    return (m.log(S / K) + (r - q + 0.5 * sigma ** 2) * T) / (sigma * m.sqrt(T))

def bsm_d2(S, K, T, r, q, sigma):
    S = float(S)
    return (m.log(S / K) + (r - q - 0.5 * sigma ** 2) * T) / (sigma * m.sqrt(T))

def bsm_pv(callPutType, S, K, T, r, q, sigma):
    d1 = bsm_d1(S, K, T, r, q, sigma)
    d2 = bsm_d2(S, K, T, r, q, sigma)
    if CallPut.call == callPutType:
        return S * N(d1) * m.exp(-q * T) - K * m.exp(-r * T) * N(d2)
    else:
        return K * N(-d2) * m.exp(-r * T)  - S * m.exp(-q * T) * N(-d1)

def bsm_delta(callPutType, S, K, T, r, q, sigma):
    d1 = bsm_d1(S, K, T, r, q, sigma)
    if CallPut.call == callPutType:
        return N(d1) * m.exp(-q * T)
    else:
        return -N(-d1) * m.exp(-q * T)

def bsm_gamma(callPutType, S, K, T, r, q, sigma):
    d2 = bsm_d1(S, K, T, r, q, sigma)
    return NPrime(d2) * m.exp(-q * T) / (S * sigma * m.sqrt(T))

def bsm_vega(callPutType, S, K, T, r, q, sigma):
    d1 = bsm_d1(S, K, T, r, q, sigma)
    return S * NPrime(d1) * m.sqrt(T)
```

In [4]:
```
K=90; T=0.5
S=100; sigma=0.2; q=0; r=0.05
print("Call: pv=%.5f, delta %.5f, vega: %.5f, gamma: %.5f" % (
```

```
            bsm_pv(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
            bsm_delta(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
            bsm_vega(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
            bsm_gamma(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
        ))
    print("Put: pv=%.5f, delta %.5f, vega: %.5f, gamma: %.5f" % (
            bsm_pv(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
            bsm_delta(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
            bsm_vega(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
            bsm_gamma(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
        ))
```

```
Call: pv=13.49852, delta 0.83952, vega: 17.23826, gamma: 0.01724
Put: pv=1.27641, delta -0.16048, vega: 17.23826, gamma: 0.01724
```

```
In [5]: K=100; T=1.
        S=100; sigma=0.2; q=0; r=0.05
        print("Call: pv=%.5f, delta %.5f, vega: %.5f, gamma: %.5f" % (
                bsm_pv(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
                bsm_delta(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
                bsm_vega(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
                bsm_gamma(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
            ))
        print("Put: pv=%.5f, delta %.5f, vega: %.5f, gamma: %.5f" % (
                bsm_pv(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
                bsm_delta(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
                bsm_vega(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma),
                bsm_gamma(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
            ))
```

```
Call: pv=10.45058, delta 0.63683, vega: 37.52403, gamma: 0.01876
Put: pv=5.57353, delta -0.36317, vega: 37.52403, gamma: 0.01876
```

0.2 Find spot value for given value of delta

0.2.1 delta neutral position

```
In [6]: def find_strike_for_delta(callPutType, S, T, r, q, sigma, target_delta):
            y = lambda x: bsm_delta(callPutType, S=S, K=x, T=T, r=r, q=q, sigma=sigma) - target_delta
            K_ = sp.optimize.brentq(y, 1e-10, S*10.)
            print( "Strike value %.8f" % K_ )
            print( "Call delta: %.8f, gamma %.8f" % (
                    bsm_delta(CallPut.call, S=S, K=K_, T=T, r=r, q=q, sigma=sigma),
                    bsm_gamma(CallPut.call, S=S, K=K_, T=T, r=r, q=q, sigma=sigma)
                ) )
            print( "Put delta: %.8f, gamma %.8f" % (
                    bsm_delta(CallPut.put, S=S, K=K_, T=T, r=r, q=q, sigma=sigma),
                    bsm_gamma(CallPut.put, S=S, K=K_, T=T, r=r, q=q, sigma=sigma)
                ) )
```

```
In [7]: T=9./12.; S=100.; sigma=0.2; q=0; r=0.05

        find_strike_for_delta(CallPut.call, S=S, T=T, r=r, q=q, sigma=sigma, target_delta=0.5)
```

```
Strike value 105.39025621
Call delta: 0.50000000, gamma 0.02303294
Put delta: -0.50000000, gamma 0.02303294
```

```
In [8]: T=9./12.; S=100.; sigma=0.2; q=0.02; r=0.05

        find_strike_for_delta(CallPut.call, S=S, T=T, r=r, q=q, sigma=sigma, target_delta=0.5)
```

```
Strike value 103.48112615
Call delta: 0.50000000, gamma 0.02268596
Put delta: -0.48511194, gamma 0.02268596
```

0.2.2 delta .25 for call

```
In [9]: find_strike_for_delta(CallPut.call, S=S, T=T, r=r, q=q, sigma=sigma, target_delta=0.25)
```

```
Strike value 116.44790994
Call delta: 0.25000000, gamma 0.01821736
Put delta: -0.73511194, gamma 0.01821736
```

0.2.3 delta .25 for put

```
In [10]: find_strike_for_delta(CallPut.put, S=S, T=T, r=r, q=q, sigma=sigma, target_delta=-0.25)
```

```
Strike value 92.56363223
Call delta: 0.73511194, gamma 0.01821736
Put delta: -0.25000000, gamma 0.01821736
```

0.2.4 NB

the Newton method desn't work here

```
In [11]: target_delta = 0.5
         t = CallPut.call

         y = lambda x: bsm_delta(callPutType=t, S=S, K=x, T=T, r=r, q=q, sigma=sigma) - target_delta
         dy = lambda x: bsm_gamma(callPutType=t, S=S, K=x, T=T, r=r, q=q, sigma=sigma)
         sp.optimize.newton(y, S, fprime = dy)


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-11-d00901f0280d> in <module>()
          4 y = lambda x: bsm_delta(callPutType=t, S=S, K=x, T=T, r=r, q=q, sigma=sigma) - target_delta
          5 dy = lambda x: bsm_gamma(callPutType=t, S=S, K=x, T=T, r=r, q=q, sigma=sigma)
    ----> 6 sp.optimize.newton(y, S, fprime = dy)


        /projects/sage/sage-6.10/local/lib/python2.7/site-packages/scipy/optimize/zeros.pyc in newton(fu
        128         for iter in range(maxiter):
        129             myargs = (p0,) + args
    --> 130             fder = fprime(*myargs)
        131             if fder == 0:
        132                 msg = "derivative was zero."
```

```
    <ipython-input-11-d00901f0280d> in <lambda>(x)
      3
      4 y = lambda x: bsm_delta(callPutType=t, S=S, K=x, T=T, r=r, q=q, sigma=sigma) - target_delta
----> 5 dy = lambda x: bsm_gamma(callPutType=t, S=S, K=x, T=T, r=r, q=q, sigma=sigma)
      6 sp.optimize.newton(y, S, fprime = dy)


    <ipython-input-3-1d339f6e0483> in bsm_gamma(callPutType, S, K, T, r, q, sigma)
     29
     30 def bsm_gamma(callPutType, S, K, T, r, q, sigma):
---> 31     d2 = bsm_d1(S, K, T, r, q, sigma)
     32     return NPrime(d2) * m.exp(-q * T) / (S * sigma * m.sqrt(T))
     33


    <ipython-input-3-1d339f6e0483> in bsm_d1(S, K, T, r, q, sigma)
      7 def bsm_d1(S, K, T, r, q, sigma):
      8     S = float(S)
----> 9     return (m.log(S / K) + (r - q + 0.5 * sigma ** 2) * T) / (sigma * m.sqrt(T))
     10
     11 def bsm_d2(S, K, T, r, q, sigma):


    ValueError: math domain error
```

1 Monte Carlo

1.1 for European

Call / Put, $q \neq 0$

```
In [12]: def mc_euro_pv(callPutType, S0, K, T, r, q, sigma, M, I):
             # Simulating I paths with M time steps
             S = np.zeros((M + 1, I))
             S[0] = S0
             dt = float(T) / M
             for t in range(1, M + 1):
                 z = np.random.standard_normal(I)
                 mean = np.mean(z)
                 std = np.std(z)
                 z = (z - mean)/std
                 S[t] = S[t - 1] * np.exp(( (r - q) - 0.5 * sigma ** 2) * dt + sigma * m.sqrt(dt) * z)

             # Calculating the Monte Carlo estimator
             if CallPut.call == callPutType:
                 C = m.exp(-r * T) * np.sum(np.maximum(S[-1] - K, 0)) / I
             else:
                 C = m.exp(-r * T) * np.sum(np.maximum(K - S[-1], 0)) / I
             return C, S
```

```
        # Parameters
        S = 80.; K = 85.; T = 1.0; r = 0.05; q = 0.02; sigma = 0.2
        M = 360; I = 50000

        np.random.seed(12345)
        t0 = time()
        C, SPaths = mc_euro_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma, M=M, I=I)
        calcTime = time() - t0

        ref_pv = bsm_pv(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
        ref_delta = bsm_delta(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
        print( "ref_pv: %.6f, ref_delta: %.6f " % (ref_pv, ref_delta) )

        print( "PV: %.5f, abs diff: %.5f, rel diff:  %.5f" % (C, ref_pv - C, (ref_pv - C)/C) )
        print( "Calculation time   %.5f" % calcTime )
```

```
ref_pv: 5.198058, ref_delta: 0.469336
PV: 5.18811, abs diff: 0.00995, rel diff:  0.00192
Calculation time   1.56068
```

```
In [13]: np.random.seed(12345)
         t0 = time()
         C, SPaths = mc_euro_pv(CallPut.put, S0=S, K=K, T=T, r=r, q=q, sigma=sigma, M=M, I=I)
         calcTime = time() - t0

         ref_pv = bsm_pv(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
         ref_delta = bsm_delta(CallPut.put, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
         print( "ref_pv: %.6f, ref_delta: %.6f " % (ref_pv, ref_delta) )

         print( "PV: %.5f, abs diff: %.5f, rel diff:  %.5f" % (C, ref_pv - C, (ref_pv - C)/C) )
         print( "Calculation time   %.5f" % calcTime )
```

```
ref_pv: 7.636666, ref_delta: -0.510863
PV: 7.62543, abs diff: 0.01123, rel diff:  0.00147
Calculation time   1.71575
```

1.2 for Barrier

```
In [14]: SPaths[0:2]
```

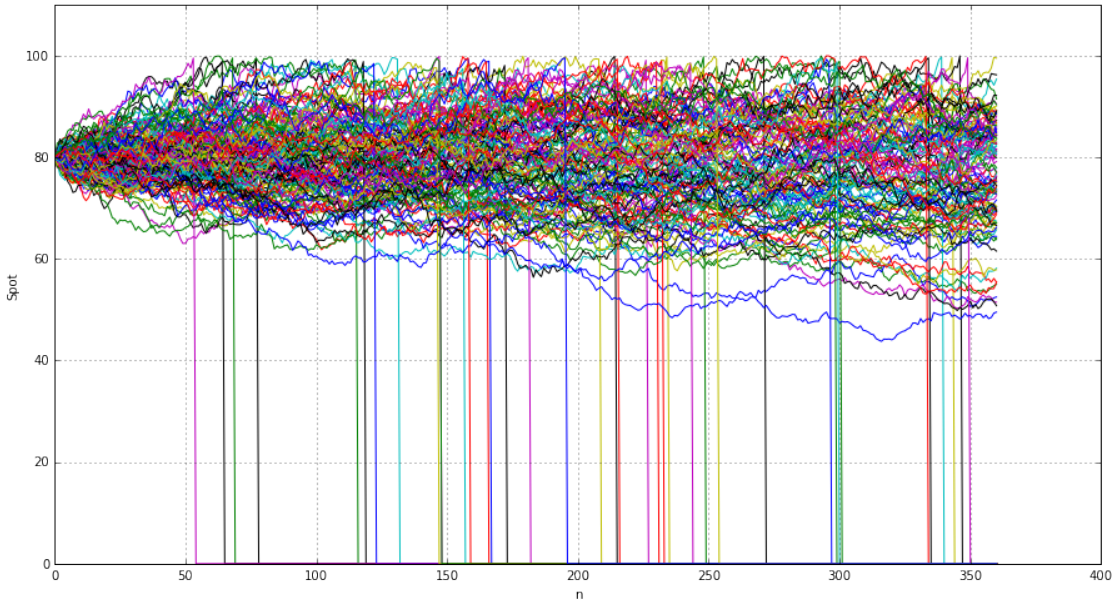
```
Out[14]: array([[ 80.        ,  80.        ,  80.        , ...,  80.        ,
                  80.        ,  80.        ],
                [ 79.83177552,  80.40754091,  79.56809924, ...,  78.34882432,
                  78.2206164 ,  79.71675684]])
```

```
In [15]: def mc_up_bar_pv(callPutType, S0, K, T, r, q, sigma, bar, M, I):
             # Simulating I paths with M time steps
             S = np.zeros((M + 1, I))
             S[0] = S0
             dt = float(T) / M
             for t in range(1, M + 1):
                 z = np.random.standard_normal(I)
                 mean = np.mean(z)
                 std = np.std(z)
                 z = (z - mean)/std
```

```
            Snext = S[t - 1] * np.exp(( (r - q) - 0.5 * sigma ** 2) * dt + sigma * m.sqrt(dt) * z)
            Snext[ Snext > bar ] = 0.
            S[t] = Snext


        # Expected payoff
        E = np.sum(np.maximum(S[-1] - K, 0)) / I
        # PV
        C = m.exp(-r * T) * E

        return C, S


    # Parameters
    S = 80.; K = 85.; T = 1.0; r = 0.05; q = 0.02; sigma = 0.2;
    bar = 100
    M = 360; I = 50000

    np.random.seed(12345)
    t0 = time()
    C, SPaths = mc_up_bar_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma, bar=bar, M=M, I=I
    calcTime = time() - t0

    print( "PV: %.5f" % (C) )
    print( "Calculation time   %.5f" % calcTime )
```

```
PV: 0.77395
Calculation time   1.50975
```

```
In [16]: plt.figure(figsize=(15,8))
         plt.plot(SPaths[:, :150])
         plt.grid(True)
         plt.xlabel('n')
         plt.ylabel('Spot')
         plt.ylim([0, bar+10])
         plt.show()
```



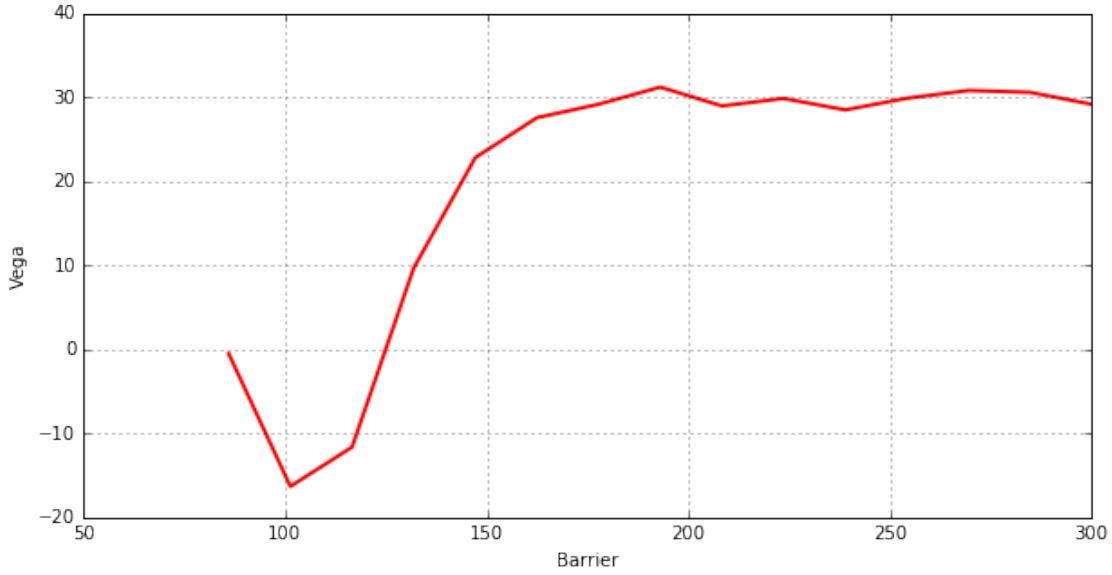

```
In [20]: ref_pv = bsm_pv(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
         ref_vega = bsm_vega(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
         print( "ref_pv: %.6f, ref_vega: %.6f " % (ref_pv, ref_vega) )


         np.random.seed(12345)
         t0 = time()
         C, SPaths = mc_up_bar_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma, bar=S*100, M=M, I
         calcTime = time() - t0

         print( "PV: %.5f, abs diff: %.5f, rel diff:  %.5f" % (C, ref_pv - C, (ref_pv - C)/C) )
         print( "Calculation time   %.5f" % calcTime )
```

```
ref_pv: 5.198058, ref_vega: 31.870381
PV: 5.18811, abs diff: 0.00995, rel diff:  0.00192
Calculation time   1.57727
```

```
In [49]: dsigma = sigma / 10.
         S = 80.; K = 80.; T = 1.0; r = 0.05; q = 0.02; sigma = 0.2;
         bar = 150.

         np.random.seed(12345)
         t0 = time()
         C_1, _ = mc_up_bar_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma-dsigma, bar=bar, M=M,
         C1, _ = mc_up_bar_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma+dsigma, bar=bar, M=M,
         vega = (C1 - C_1) / (2. * dsigma)
         calcTime = time() - t0

         print( "Vega: %.5f" % (vega) )
         print( "Calculation time   %.5f" % calcTime )
```

```
Vega: 23.00726
Calculation time   3.44547
```

```
In [52]: def mc_up_bar_vega(x):
             C_1, _ = mc_up_bar_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma-dsigma, bar=x, M=
             C1, _ = mc_up_bar_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma+dsigma, bar=x, M=M
             vega = (C1 - C_1) / (2. * dsigma)
             return vega

         x = np.linspace(86, 300, num=15)
         y = [mc_up_bar_vega(i) for i in x ]

         lw = 2.0

In [53]: plt.figure(figsize=(10,5))
         plt.plot(x, y, 'r', linewidth=lw)
         axes = plt.gca()
         axes.set_xlabel('Barrier')
         axes.set_ylabel('Vega')
         plt.grid(True)
         plt.show()
```



1.2.1 for Asian option

```
In [55]: def mc_asian_pv(callPutType, S0, K, T, r, q, sigma, M, I):
             # Simulating I paths with M time steps
             S = np.zeros((M + 1, I))
             S[0] = S0
             dt = float(T) / M
             for t in range(1, M + 1):
                 z = np.random.standard_normal(I)
                 mean = np.mean(z)
                 std = np.std(z)
                 z = (z - mean)/std
                 S[t] = S[t - 1] * np.exp(( (r - q) - 0.5 * sigma ** 2) * dt + sigma * m.sqrt(dt) * z)
```

```
            # Expected payoff
            E = np.sum( np.maximum( np.apply_along_axis(np.sum, 0, S) / M - K, 0 ) ) / I #np.sum(np.ma
            # PV
            C = m.exp(-r * T) * E

            return C, S


        # Parameters
        S = 80.; K = 80.; T = 2./12.; r = 0.05; q = 0.02; sigma = 0.2;
        M = 360; I = 50000

        np.random.seed(12345)
        t0 = time()
        C, SPaths = mc_asian_pv(CallPut.call, S0=S, K=K, T=T, r=r, q=q, sigma=sigma, M=M, I=I)
        calcTime = time() - t0

        print( "PV: %.5f" % (C) )
        print( "Calculation time   %.5f" % calcTime )
```

```
PV: 1.72025
Calculation time   1.97063
```

```
In [56]: ref_pv = bsm_pv(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
         ref_vega = bsm_delta(CallPut.call, S=S, K=K, T=T, r=r, q=q, sigma=sigma)
         print( "ref_pv: %.6f, ref_delta: %.6f " % (ref_pv, ref_vega) )

         print( "PV: %.5f, abs diff: %.5f, rel diff:  %.5f" % (C, ref_pv - C, (ref_pv - C)/C) )
```

```
ref_pv: 2.793701, ref_delta: 0.538847
PV: 1.72025, abs diff: 1.07345, rel diff:  0.62401
```

```
In [ ]:
```