

Derivatives Pricing Homework 3

Pan Yiming, Tai Lo Yeung, Qiwei Liu

2021/02/28

1 Exercise 1: Delta-hedged portfolio

We set $K = 100, K = 100, r = 0.02, T = 1/12, t = 0.0, \sigma = 0.20$.

Assume the stock moves by x units immediately after Delta-hedge. And then we plot the exact value of the profit and loss portfolio as a function of x together with its second order Tylor approximation. We use the formulas like following:

The real loss of our delta-hedging portfolio is computed as follows:

$$V_0 = C'_0 - C_0 - \Delta_0 x$$

Where the C_0 is the call price when stock price is S_t at time t , C'_0 is the call price after stock price change to $S_t + x$.

The loss and profit approximated by Tylor's formula is computed by:

$$V_0 = \frac{1}{2} \Gamma_0 x^2$$

The plot is like following. As we can see from the plot that as the the change of stock price gets bigger, the error of Tylor approximation will get bigger, too. But when the change is small, the Tylor's formula tracks the real loss and profits very well.

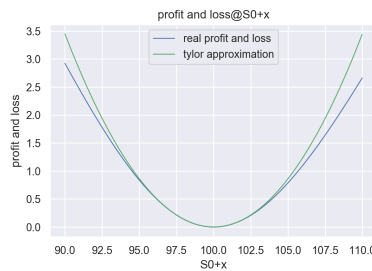


Figure 1: stock price

2 Exercise 2: Pricing Barrier options

2.1 Stock price paths

We generate several random paths and choose two of them, one of which hits the barrier and the other does not. We set $S_0=100, T = 1/2, N = 1000, r=0.05, K = 105, \sigma = 0.20, H = 95$.



Figure 2: stock paths

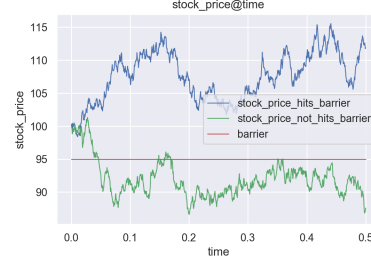


Figure 3: stock paths

2.2 Pricing barrier option

Because we set $H < K$, so the formulas to compute the value of options are as follows:

For the down-and-out option:

$$C_t^{d/o} = S_t N(d_{1,t}) - \left(\frac{H}{S_t}\right)^{1+2r\sigma^{-2}} S_t N(h_{1,t}) - K e^{-r(T-t)} N(d_{2,t}) + K e^{-r(T-t)} \left(\frac{H}{S_t}\right)^{-1+2\sigma^{-2}} N(h_{2,t})$$

For the down-and-in option:

$$C_t^{d/i} = \left(\frac{H}{S_t}\right)^{1+2r\sigma^{-2}} S_t N(h_{1,t}) - K e^{-r(T-t)} \left(\frac{H}{S_t}\right)^{-1+2\sigma^{-2}} N(h_{2,t})$$

where:

$$d_j = \frac{\log(\frac{S_t}{K}) + (r + (-1)^{j-1} \frac{1}{2} \sigma^2)(T-t)}{\sigma \sqrt{T-t}}$$

and:

$$h_j = \frac{\log(\frac{H^2}{S_t K}) + (r + (-1)^{j-1} \frac{1}{2} \sigma^2)(T-t)}{\sigma \sqrt{T-t}}$$

2.2.1 Pricing down-and-out barrier option: $S_{min} > H$

For the stock whose price does not hit the barrier, we get the value of the down-and-out call and the difference between it and plain vanilla call, which is shown as follows. From the plot, we can see that the down-and-out call value is less than plain vanilla call. And when $S_{min} > H$, as time goes forward, the difference between them is getting smaller.

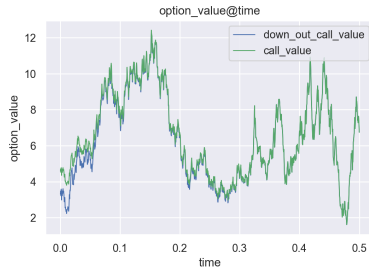


Figure 4: stock paths

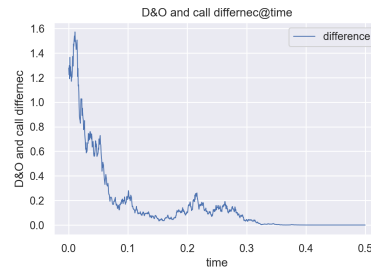


Figure 5: stock paths

2.2.2 Pricing down-and-out barrier option: $S_{min} \leq H$

For the stock whose price hits the barrier, we get the value of the down-and-out call and the difference between it and plain vanilla call, which is shown as follows. From the plot, we can see that the down-and-out call value is less than plain vanilla call. After the stock price hits the barrier first time, the price of down-and-out call gets 0.

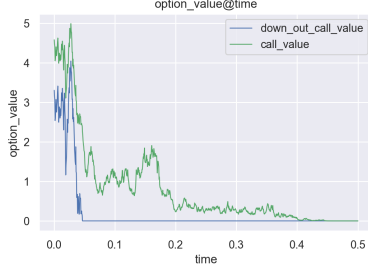


Figure 6: stock paths

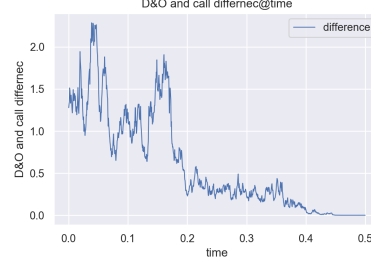


Figure 7: stock paths

2.2.3 Pricing down-and-in barrier option: $S_{min} > H$

For the stock whose price does not hit the barrier, we get the value of the down-and-in call as follows. From the plot, we can see that the down-and-in call value is less than plain vanilla call.

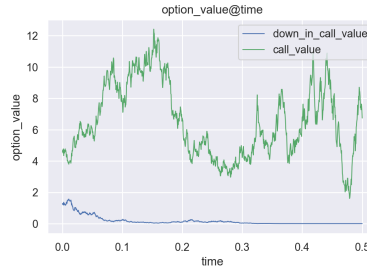


Figure 8: call paths

2.2.4 Pricing down-and-in barrier option: $S_{min} \leq H$

For the stock whose price hits the barrier, we get the value of the down-and-in call as follows. From the plot, we can see that the down-and-in call value is less than plain vanilla call when the stock price does not hit the barrier. After the stock price hits the Barrier, the down-and-in call price becomes the same as plain vanilla call.

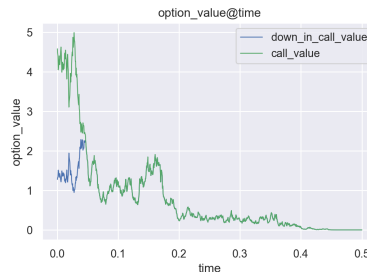


Figure 9: call paths

2.3 Plot the 3 options as a function of stock price

2.3.1 Call option price

We set $K=105$. As the S_0 increases, the call price increases, too. For the same S_0, K , the call price increases as the time to maturity increase.



Figure 10: call paths

2.3.2 Down-and-out call option price

We set $K=105$ and $H=90$. If S_0 is below $H=90$, the down-and-out call's value is 0. If the S_0 higher $H=90$, the down-and-out call's value becomes higher than 0, which increases as S_0 increases. In addition, for the same K, S_0, H , as the time to maturity increases, the value of down-and-out call goes up.



Figure 11: call paths

2.3.3 Down-and-in call option price

We set $K=105$ and $H=90$. If the S_0 is below $H=90$, the down-and-in call's value is the same as the plain vanilla call, which increases as S_0 increases. If S_0 is higher than the Barrier, the down-and-in call's value decreases as S_0 increases. In addition, for the same K, S_0, H , as the time to maturity increases, the value of down-and-in call goes up.

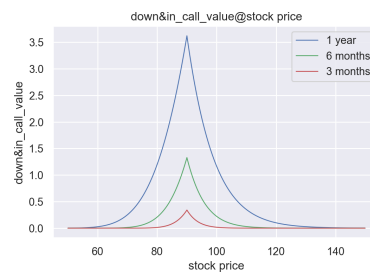


Figure 12: call paths

3 Appendix: code

```
1 import math
2 import random
3 import numpy as np
4 import seaborn as sns
5 import matplotlib.pyplot as plt
6 import matplotlib._color_data as mcd
7 import matplotlib.patches as mpatch
8 from scipy.stats import norm
9
10
11 def d1_v(St, K, sigma, expir_t, r):
12     d1 = (math.log(St/K) + (r+0.5*sigma**2)*expir_t)/(sigma*expir_t**0.5)
13     N1 = norm.cdf(d1)
14     return N1
15
16 def d2_v(St, K, sigma, expir_t, r):
17     d2 = (math.log(St/K) + (r-0.5*sigma**2)*expir_t)/(sigma*expir_t**0.5)
18     N2 = norm.cdf(d2)
19     return N2
20
21 def h1_v(St, H, K, r, sigma, et):
22     h1 = (math.log(H**2/(St*K))+(r+0.5*sigma**2)*et)/(sigma*et**0.5)
23     NH1 = norm.cdf(h1)
24     return NH1
25
26 def h2_v(St, H, K, r, sigma, et):
27     h2 = (math.log(H**2/(St*K))+(r-0.5*sigma**2)*et)/(sigma*et**0.5)
28     NH2 = norm.cdf(h2)
29     return NH2
30
31 def DO_v(St, H, K, sigma, r, et):
32     NH1 = h1_v(St, H, K, r, sigma, et)
33     NH2 = h2_v(St, H, K, r, sigma, et)
34     N1 = d1_v(St, K, sigma, et, r)
35     N2 = d2_v(St, K, sigma, et, r)
36     HS1 = (H/St)**(1+2*r*sigma**(-2))
37     HS2 = (H/St)**(-1+2*r*sigma**(-2))
38     dis_f = K * math.exp(-r*et)
39     CDO = St*N1 - HS1*St*NH1 - dis_f*N2 + dis_f*HS2*NH2
40     return CDO
41
42 def DI_v(St, H, K, sigma, r, et):
43     NH1 = h1_v(St, H, K, r, sigma, et)
44     NH2 = h2_v(St, H, K, r, sigma, et)
45     HS1 = (H/St)**(1+2*r*sigma**(-2))
46     HS2 = (H/St)**(-1+2*r*sigma**(-2))
47     dis_f = K * math.exp(-r*et)
48     CDI = HS1*St*NH1 - dis_f*HS2*NH2
49     return CDI
50
51 def gama_v(St, K, sigma, expir_t, r):
52     d1 = (math.log(St/K) + (r+0.5*sigma**2)*expir_t)/(sigma*expir_t**0.5)
53     N11 = norm.pdf(d1)
54     gama = 1/(St*sigma*expir_t**0.5) * N11
55     return gama
56
57 def DO_value(path, H, K, sigma, T, N):
58     CDO_path = []
59     dt = T/N
60     paths_gone = []
61     for i in range(N):
62         St = path[i]
63         paths_gone.append(St)
64         et = T - i*dt
65         if min(paths_gone) > H:
66             DO_vt = DO_v(St, H, K, sigma, r, et)
67         else:
68             DO_vt = 0
```

```

69         CDO_path.append(DO_vt)
70     return CDO_path
71
72 def DI_value(path, H, K, sigma, T, N):
73     CDI_path = []
74     dt = T/N
75     paths_gone = []
76     for i in range(N):
77         St = path[i]
78         et = T - i*dt
79         paths_gone.append(St)
80         if min(paths_gone)>H:
81             DI_vt = DI_v(St, H, K, sigma, r, et)
82         else:
83             DI_vt = Ct_v(St, K, sigma, et, r)
84         CDI_path.append(DI_vt)
85     return CDI_path
86
87 def Ct_v(St, K, sigma, expir_t, r):
88     N1 = d1_v(St, K, sigma, expir_t, r)
89     N2 = d2_v(St, K, sigma, expir_t, r)
90     Ct = St*N1 - K*math.exp(-r*expir_t)*N2
91     return Ct
92
93 def Ct_value(path, K, T, N, sigma):
94     dt = T/N
95     Ct_path = []
96     Deltas = []
97     Deltas_dis = []
98     for i in range(N):
99         St = path[i]
100         Ti = T - i*dt
101         Ct = Ct_v(St, K, sigma, Ti, r)
102         Ct_path.append(Ct)
103     return Ct_path
104
105 def St_simulation_SDE(S0, T, N, r, sig):
106     dt = T/N
107     dBs = np.random.normal(0, dt**0.5, N)
108     path = []
109     St = S0
110     path.append(St)
111     for i, dB_i in enumerate(dBs):
112         St = St + sig*St*dB_i+r*St*dt
113         path.append(St)
114     return path
115
116
117 def plot_path_price(paths, names, T, xname, yname):
118     colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
119     num = len(names)
120     plt.figure(figsize=(6, 4))
121     plt.rcParams['savefig.dpi'] = 200
122     plt.rcParams['figure.dpi'] = 200
123     for i in range(num):
124         path_i = paths[i]
125         num_ps = len(path_i)
126         index = [i*T/(num_ps-1) for i in range(num_ps)]
127         plt.plot(index, path_i, color=colors[i],
128                 linestyle="-", linewidth=0.9, label=names[i])
129
130     plt.xlabel(xname)
131     plt.ylabel(yname)
132     plt.title('{}@{}'.format(yname, xname))
133     plt.legend()
134     sns.set()
135     plt.show()
136
137
138 def plot_path(paths, idx, names, T, xname, yname):

```

```

139     colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']
140     num = len(names)
141     plt.figure(figsize=(6, 4))
142     plt.rcParams['savefig.dpi'] = 200
143     plt.rcParams['figure.dpi'] = 200
144     for i in range(num):
145         path_i = paths[i]
146         num_ps = len(path_i)
147         #index = [i*T/(num_ps-1) for i in range(num_ps)]
148         plt.plot(idx, path_i, color=colors[i],
149                 linestyle="-", linewidth=0.9, label=names[i])
150
151     plt.xlabel(xname)
152     plt.ylabel(yname)
153     plt.title('{}@{}'.format(yname, xname))
154     plt.legend()
155     sns.set()
156     plt.show()
157
158
159 # ----1 Tylor expansion approximation of profits and loss---
160 S0 = 100
161 K = 100
162 r = 0.02
163 T = 1/12
164 t = 0.0
165 sigma = 0.20
166
167 et = T-t
168 PL_real = []
169 PL_tylor = []
170 idx = []
171
172 C0 = Ct_v(S0, K, sigma, et, r)
173 gama = gama_v(S0, K, sigma, et, r)
174 delta = d1_v(S0, K, sigma, et, r)
175
176 for i in range(-1000,1000, 1):
177     x = i * 0.01
178     S_i = S0+x
179     idx.append(S_i)
180     Ct = Ct_v(S_i, K, sigma, et, r)
181     pl_real = Ct - C0 - delta*x
182     PL_real.append(pl_real)
183     pl_tylor = 0.5*gama*x**2
184     PL_tylor.append(pl_tylor)
185
186
187 paths = [PL_real, PL_tylor]
188 names = ['real profit and loss', 'tylor approximation']
189
190 xname = 'S0+x'
191 yname = 'profit and loss'
192 plot_path(paths, idx, names, T, xname, yname)
193
194 #----- stock price paths-----#
195 S0=100
196 T = 1/2
197 N = 1000
198 r=0.05
199 K = 105
200 Sig = 0.20
201 Barrier = 120
202 paths = []
203 names = []
204 # simulate stock price with
205 for i in range(6):
206     Sts = St_simulation_SDE(S0, T, N, r, Sig)
207     paths.append(Sts)
208     names.append('stock_price{}'.format(i))

```

```

209
210 xname = 'time'
211 yname = 'stock_price'
212 plot_path_price(paths, names, T, xname, yname)
213
214 S0=100
215 T = 1/2
216 N = 1000
217 r=0.05
218 K = 105
219 Sig = 0.20
220 H = 95
221 paths = []
222 names = []
223 Barriers = [H for i in range(N)]
224
225 # simulate stock path that does not hit the barrier:
226 min_st = -100
227 while min_st < H:
228     Sts_do = St_simulation_SDE(S0, T, N, r, Sig)
229     min_st = min(Sts_do)
230
231
232 # simulate stock path that hits the barrier:
233 min_st = 1000
234 while min_st > H:
235     Sts_di = St_simulation_SDE(S0, T, N, r, Sig)
236     min_st = min(Sts_di)
237
238 paths.append(Sts_do)
239 paths.append(Sts_di)
240 names.append('stock_price_hits_barrier')
241 names.append('stock_price_not_hits_barrier')
242 paths.append(Barriers)
243 names.append('barrier')
244 xname = 'time'
245 yname = 'stock_price'
246 plot_path_price(paths, names, T, xname, yname)
247
248 #---pricing down-and-out call option-----
249 N=1000
250 DO_path = DO_value(Sts_do, H, K, sigma, T, N)
251 COt_path = Ct_value(Sts_do, K, T, N, sigma)
252
253 paths = [DO_path, COt_path]
254 names = ['down_out_call_value', 'call_value']
255 xname = 'time'
256 yname = 'option_value'
257 plot_path_price(paths, names, T, xname, yname)
258
259
260 diff = (np.array(COt_path) - np.array(DO_path)).tolist()
261 names = ['difference']
262 xname = 'time'
263 yname = 'D&O and call differnec'
264 paths = [diff]
265 plot_path_price(paths, names, T, xname, yname)
266
267 N=1000
268 DO_path = DO_value(Sts_di, H, K, sigma, T, N)
269 COt_path = Ct_value(Sts_di, K, T, N, sigma)
270 paths = [DO_path, COt_path]
271 names = ['down_out_call_value', 'call_value']
272 xname = 'time'
273 yname = 'option_value'
274 plot_path_price(paths, names, T, xname, yname)
275
276 diff = (np.array(COt_path) - np.array(DO_path)).tolist()
277 names = ['difference']
278 xname = 'time'

```



```

279 yname = 'D&O and call differnec'
280 paths = [diff]
281 plot_path_price(paths, names, T, xname, yname)
282
283 #---pircing down-and-in call option-----
284 DI_path = DI_value(Sts_di, H, K, sigma, T, N)
285 CIt_path = Ct_value(Sts_di, K, T, N, sigma)
286 paths = [DI_path, CIt_path]
287 names = ['down_in_call_value', 'call_value']
288 xname = 'time'
289 yname = 'option_value'
290 plot_path_price(paths, names, T, xname, yname)
291
292 DI_path = DI_value(Sts_do, H, K, sigma, T, N)
293 CIt_path = Ct_value(Sts_do, K, T, N, sigma)
294 paths = [DI_path, CIt_path]
295 names = ['down_in_call_value', 'call_value']
296 xname = 'time'
297 yname = 'option_value'
298 plot_path_price(paths, names, T, xname, yname)
299
300 # call option value with different expirations and stock price
301 Et1 = 1
302 Et2 = 1/2
303 Et3 = 1/4
304 N1 = 800
305 r=0.05
306 K = 105
307 Sig = 0.20
308 sigma = 0.2
309
310 Sts = [i*0.1+60 for i in range(N1)]
311
312 def get_cts(Sts, H, K, sigma, Et):
313     C_vts = []
314     for S_i in Sts:
315         C_vt = Ct_v(S_i, K, sigma, Et, r)
316         C_vts.append(C_vt)
317     return C_vts
318
319 Ct_path1 = get_cts(Sts, H, K, sigma, Et1)
320 Ct_path2 = get_cts(Sts, H, K, sigma, Et2)
321 Ct_path3 = get_cts(Sts, H, K, sigma, Et3)
322
323 paths = [Ct_path1, Ct_path2, Ct_path3]
324 names = ['1 year', '6 months', '3 months']
325 xname = 'stock price'
326 yname = 'call_value'
327 plot_path(paths, Sts, names, T, xname, yname)
328
329 # down-and-out call option value with different expirations and stock price
330 r=0.05
331 K = 105
332 H = 80
333 sigma = 0.20
334
335 Et1 = 1
336 Et2 = 1/2
337 Et3 = 1/4
338 N1 = 800
339 Sts = [i*0.1+60 for i in range(N1)]
340
341
342 def get_ps(Sts, H, K, sigma, Et1):
343     DO_vts = []
344     for S_i in Sts:
345         if S_i > H:
346             DO_vt = DO_v(S_i, H, K, sigma, r, Et1)
347         else:
348             DO_vt = 0

```

```

349     DO_vts.append(DO_vt)
350     return DO_vts
351
352 Do_path1 = get_ps(Sts, H, K, sigma, Et1)
353 Do_path2 = get_ps(Sts, H, K, sigma, Et2)
354 Do_path3 = get_ps(Sts, H, K, sigma, Et3)
355
356 paths = [Do_path1, Do_path2, Do_path3]
357 names = ['1 year', '6 months', '3 months']
358 xname = 'stock price'
359 yname = 'down&out_call_value'
360 plot_path(paths, Sts, names, T, xname, yname)
361
362 # down-and-in call option value with different expirations and stock price
363 Et1 = 1
364 Et2 = 1/2
365 Et3 = 1/4
366 N1 = 1000
367 r=0.05
368 K = 105
369 H = 100
370 sigma = 0.20
371
372 Sts = [i*0.1+50 for i in range(N1)]
373
374
375 def get_dis(Sts, H, K, sigma, Et1):
376     DI_vts = []
377     CS = []
378     for S_i in Sts:
379         if S_i < H:
380             DI_vt = Ct_v(S_i, K, sigma, Et1, r)
381         else:
382             DI_vt = DI_v(S_i, H, K, sigma, r, Et1)
383         DI_vts.append(DI_vt)
384     return DI_vts
385
386
387
388 DI_path1 = get_dis(Sts, H, K, sigma, Et1)
389 DI_path2 = get_dis(Sts, H, K, sigma, Et2)
390 DI_path3 = get_dis(Sts, H, K, sigma, Et3)
391
392
393 paths = [DI_path1, DI_path2, DI_path3]
394 names = ['1 year', '6 months', '3 months']
395 xname = 'stock price'
396 yname = 'down&in_call_value'
397 plot_path(paths, Sts, names, T, xname, yname)

```