

1 Functions

1.1 Vanilla Euro

In [1]:

```

1 # =====
2 # This file is especially for visualization of option price/Greeks w.r.t its parameters
3 # Mainly based on Quantlib
4 # =====
5 import QuantLib as ql
6
7 def vanilla_euro_call(S,K,r,q,tau,sigma,**kwargs):
8     # specify timeframe
9     tau_day=int(round(tau*365))
10    start=ql.Date(1,1,2000)
11    ql.Settings.instance().evaluationDate=start
12    maturity=start+tau_day
13    day_count = ql.Actual365Fixed()
14    calendar = ql.UnitedStates()
15    # specify option type
16    option_type = ql.Option.Call
17    payoff = ql.PlainVanillaPayoff(option_type, K)
18    exercise = ql.EuropeanExercise(maturity)
19    option=ql.VanillaOption(payoff, exercise)
20    # specify pricing method
21    spot_handle = ql.QuoteHandle(
22        ql.SimpleQuote(S)
23    )
24    flat_ts = ql.YieldTermStructureHandle(
25        ql.FlatForward(start, r, day_count)
26    )
27    dividend_yield = ql.YieldTermStructureHandle(
28        ql.FlatForward(start, q, day_count)
29    )
30    flat_vol_ts = ql.BlackVolTermStructureHandle(
31        ql.BlackConstantVol(start, calendar, sigma, day_count)
32    )
33    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
34                                                dividend_yield,
35                                                flat_ts,
36                                                flat_vol_ts)
37    option.setPricingEngine(ql.AnalyticEuropeanEngine(bsm_process))
38    #setup is finished
39    # return option.NPV(),option.delta(),option.gamma(),option.vega(),option.theta(),option.rho()
40    return {
41        'PV':option.NPV(),
42        'delta':option.delta(),
43        'gamma':option.gamma(),
44        'vega':option.vega(),
45        'theta':option.theta(),
46        'rho':option.rho(),
47    }
48
49 def vanilla_euro_put(S,K,r,q,tau,sigma,**kwargs):
50     # specify timeframe
51     tau_day=int(round(tau*365))
52     start=ql.Date(1,1,2000)
53     ql.Settings.instance().evaluationDate=start
54     maturity=start+tau_day
55     day_count = ql.Actual365Fixed()
56     calendar = ql.UnitedStates()
57     # specify option type
58     option_type = ql.Option.Put
59     payoff = ql.PlainVanillaPayoff(option_type, K)

```

```
60     exercise = ql.EuropeanExercise(maturity)
61     option=ql.VanillaOption(payoff, exercise)
62     # specify pricing method
63     spot_handle = ql.QuoteHandle(
64         ql.SimpleQuote(S)
65     )
66     flat_ts = ql.YieldTermStructureHandle(
67         ql.FlatForward(start, r, day_count)
68     )
69     dividend_yield = ql.YieldTermStructureHandle(
70         ql.FlatForward(start, q, day_count)
71     )
72     flat_vol_ts = ql.BlackVolTermStructureHandle(
73         ql.BlackConstantVol(start, calendar, sigma, day_count)
74     )
75     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
76                                                 dividend_yield,
77                                                 flat_ts,
78                                                 flat_vol_ts)
79     option.setPricingEngine(ql.AnalyticEuropeanEngine(bsm_process))
80     #setup is finished
81     # return option.NPV(),option.delta(),option.gamma(),option.vega(),option.theta(),option.rho()
82     return {
83         'PV':option.NPV(),
84         'delta':option.delta(),
85         'gamma':option.gamma(),
86         'vega':option.vega(),
87         'theta':option.theta(),
88         'rho':option.rho(),
89     }
```

1.2 Vanilla Ame

In [2]:

```

1 def _vanilla_ame_call(S,K,r,q,tau,sigma):
2     # specify timeframe
3     tau_day=int(round(tau*365))
4     start=ql.Date(1,1,2000)
5     ql.Settings.instance().evaluationDate=start
6     maturity=start+tau_day
7     day_count = ql.Actual365Fixed()
8     calendar = ql.UnitedStates()
9     # specify option type
10    option_type = ql.Option.Call
11    payoff = ql.PlainVanillaPayoff(option_type, K)
12    exercise = ql.AmericanExercise(start,maturity)
13    option=ql.VanillaOption(payoff, exercise)
14    # specify pricing method
15    spot_handle = ql.QuoteHandle(
16        ql.SimpleQuote(S)
17    )
18    flat_ts = ql.YieldTermStructureHandle(
19        ql.FlatForward(start, r, day_count)
20    )
21    dividend_yield = ql.YieldTermStructureHandle(
22        ql.FlatForward(start, q, day_count)
23    )
24    flat_vol_ts = ql.BlackVolTermStructureHandle(
25        ql.BlackConstantVol(start, calendar, sigma, day_count)
26    )
27    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
28                                                dividend_yield,
29                                                flat_ts,
30                                                flat_vol_ts)
31    steps=tau_day*2
32    option.setPricingEngine(ql.BinomialVanillaEngine(bsm_process, 'crr', steps))
33    #setup is finished
34    return option
35
36 def vanilla_ame_call(S,K,r,q,tau,sigma,**kwargs):
37     option=_vanilla_ame_call(S,K,r,q,tau,sigma)
38     epsilon=1e-8
39     option_sigam_upper=_vanilla_ame_call(S,K,r,q,tau,sigma+epsilon)
40     option_sigam_lower=_vanilla_ame_call(S,K,r,q,tau,sigma-epsilon)
41     option_r_upper=_vanilla_ame_call(S,K,r+epsilon,q,tau,sigma)
42     option_r_lower=_vanilla_ame_call(S,K,r-epsilon,q,tau,sigma)
43     return {
44         'PV':option.NPV(),
45         'delta':option.delta(),
46         'gamma':option.gamma(),
47         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
48         'theta':option.theta(),
49         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
50     }
51
52 def _vanilla_ame_put(S,K,r,q,tau,sigma):
53     # specify timeframe
54     tau_day=int(round(tau*365))
55     start=ql.Date(1,1,2000)
56     ql.Settings.instance().evaluationDate=start
57     maturity=start+tau_day
58     day_count = ql.Actual365Fixed()
59     calendar = ql.UnitedStates()

```

```

60 # specify option type
61 option_type = ql.Option.Put
62 payoff = ql.PlainVanillaPayoff(option_type, K)
63 exercise = ql.AmericanExercise(start,maturity)
64 option=ql.VanillaOption(payoff, exercise)
65 # specify pricing method
66 spot_handle = ql.QuoteHandle(
67 ql.SimpleQuote(S)
68 )
69 flat_ts = ql.YieldTermStructureHandle(
70     ql.FlatForward(start, r, day_count)
71 )
72 dividend_yield = ql.YieldTermStructureHandle(
73     ql.FlatForward(start, q, day_count)
74 )
75 flat_vol_ts = ql.BlackVolTermStructureHandle(
76     ql.BlackConstantVol(start, calendar, sigma, day_count)
77 )
78 bsm_process = ql.BlackScholesMertonProcess(spot_handle,
79                                             dividend_yield,
80                                             flat_ts,
81                                             flat_vol_ts)
82 steps=tau_day*2
83 option.setPricingEngine(ql.BinomialVanillaEngine(bsm_process, 'crr', steps))
84 #setup is finished
85 return option
86 def vanilla_ame_put(S,K,r,q,tau,sigma,**kwargs):
87     option=_vanilla_ame_put(S,K,r,q,tau,sigma)
88     epsilon=1e-8
89     option_sigam_upper=_vanilla_ame_put(S,K,r,q,tau,sigma+epsilon)
90     option_sigam_lower=_vanilla_ame_put(S,K,r,q,tau,sigma-epsilon)
91     option_r_upper=_vanilla_ame_put(S,K,r+epsilon,q,tau,sigma)
92     option_r_lower=_vanilla_ame_put(S,K,r-epsilon,q,tau,sigma)
93     return {
94         'PV':option.NPV(),
95         'delta':option.delta(),
96         'gamma':option.gamma(),
97         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
98         'theta':option.theta(),
99         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
100    }

```

1.3 Barrier

In [3]:

```

1 def _barrier_upin_call(S,K,X,r,q,tau,sigma):
2     # specify timeframe
3     tau_day=int(round(tau*365))
4     start=ql.Date(1,1,2000)
5     ql.Settings.instance().evaluationDate=start
6     maturity=start+tau_day
7     day_count = ql.Actual365Fixed()
8     calendar = ql.UnitedStates()
9     # specify option type
10    option_type = ql.Option.Call
11    payoff = ql.PlainVanillaPayoff(option_type, K)
12    exercise = ql.EuropeanExercise(maturity)
13    option=ql.BarrierOption(ql.Barrier.UpIn, X, 0, payoff, exercise)
14    # specify pricing method
15    spot_handle = ql.QuoteHandle(
16        ql.SimpleQuote(S)
17    )
18    flat_ts = ql.YieldTermStructureHandle(
19        ql.FlatForward(start, r, day_count)
20    )
21    dividend_yield = ql.YieldTermStructureHandle(
22        ql.FlatForward(start, q, day_count)
23    )
24    flat_vol_ts = ql.BlackVolTermStructureHandle(
25        ql.BlackConstantVol(start, calendar, sigma, day_count)
26    )
27    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
28                                                dividend_yield,
29                                                flat_ts,
30                                                flat_vol_ts)
31    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
32    return option
33
34 def barrier_upin_call(S,K,X,r,q,tau,sigma,**kwargs):
35     option=_barrier_upin_call(S,K,X,r,q,tau,sigma)
36     epsilon=1e-4
37     option_S_upper=_barrier_upin_call(S+epsilon,K,X,r,q,tau,sigma)
38     option_S_lower=_barrier_upin_call(S-epsilon,K,X,r,q,tau,sigma)
39     option_sigam_upper=_barrier_upin_call(S,K,X,r,q,tau,sigma+epsilon)
40     option_sigam_lower=_barrier_upin_call(S,K,X,r,q,tau,sigma-epsilon)
41     option_tau_upper=_barrier_upin_call(S,K,X,r,q,tau+100*epsilon,sigma)
42     option_tau_lower=_barrier_upin_call(S,K,X,r,q,tau-100*epsilon,sigma)
43     option_r_upper=_barrier_upin_call(S,K,X,r+epsilon,q,tau,sigma)
44     option_r_lower=_barrier_upin_call(S,K,X,r-epsilon,q,tau,sigma)
45     return {
46         'PV':option.NPV(),
47         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
48         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
49         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
50         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
51         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
52     }
53
54 def _barrier_upin_put(S,K,X,r,q,tau,sigma):
55     # specify timeframe
56     tau_day=int(round(tau*365))
57     start=ql.Date(1,1,2000)
58     ql.Settings.instance().evaluationDate=start
59     maturity=start+tau_day

```

```

60     day_count = ql.Actual365Fixed()
61     calendar = ql.UnitedStates()
62     # specify option type
63     option_type = ql.Option.Put
64     payoff = ql.PlainVanillaPayoff(option_type, K)
65     exercise = ql.EuropeanExercise(maturity)
66     option=ql.BarrierOption(ql.Barrier.UpIn, X, 0, payoff, exercise)
67     # specify pricing method
68     spot_handle = ql.QuoteHandle(
69         ql.SimpleQuote(S)
70     )
71     flat_ts = ql.YieldTermStructureHandle(
72         ql.FlatForward(start, r, day_count)
73     )
74     dividend_yield = ql.YieldTermStructureHandle(
75         ql.FlatForward(start, q, day_count)
76     )
77     flat_vol_ts = ql.BlackVolTermStructureHandle(
78         ql.BlackConstantVol(start, calendar, sigma, day_count)
79     )
80     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
81                                                 dividend_yield,
82                                                 flat_ts,
83                                                 flat_vol_ts)
84     option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
85     return option
86
87 def barrier_upin_put(S,K,X,r,q,tau,sigma,**kwargs):
88     option=_barrier_upin_put(S,K,X,r,q,tau,sigma)
89     epsilon=1e-4
90     option_S_upper=_barrier_upin_put(S+epsilon,K,X,r,q,tau,sigma)
91     option_S_lower=_barrier_upin_put(S-epsilon,K,X,r,q,tau,sigma)
92     option_sigam_upper=_barrier_upin_put(S,K,X,r,q,tau,sigma+epsilon)
93     option_sigam_lower=_barrier_upin_put(S,K,X,r,q,tau,sigma-epsilon)
94     option_tau_upper=_barrier_upin_put(S,K,X,r,q,tau+100*epsilon,sigma)
95     option_tau_lower=_barrier_upin_put(S,K,X,r,q,tau-100*epsilon,sigma)
96     option_r_upper=_barrier_upin_put(S,K,X,r+epsilon,q,tau,sigma)
97     option_r_lower=_barrier_upin_put(S,K,X,r-epsilon,q,tau,sigma)
98     return {
99         'PV':option.NPV(),
100        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
101        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
102        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
103        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
104        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
105    }
106
107 def _barrier_upout_call(S,K,X,r,q,tau,sigma):
108     # specify timeframe
109     tau_day=int(round(tau*365))
110     start=ql.Date(1,1,2000)
111     ql.Settings.instance().evaluationDate=start
112     maturity=start+tau_day
113     day_count = ql.Actual365Fixed()
114     calendar = ql.UnitedStates()
115     # specify option type
116     option_type = ql.Option.Call
117     payoff = ql.PlainVanillaPayoff(option_type, K)
118     exercise = ql.EuropeanExercise(maturity)
119     option=ql.BarrierOption(ql.Barrier.UpOut, X, 0, payoff, exercise)
120     # specify pricing method

```

```

121     spot_handle = ql.QuoteHandle(
122         ql.SimpleQuote(S)
123     )
124     flat_ts = ql.YieldTermStructureHandle(
125         ql.FlatForward(start, r, day_count)
126     )
127     dividend_yield = ql.YieldTermStructureHandle(
128         ql.FlatForward(start, q, day_count)
129     )
130     flat_vol_ts = ql.BlackVolTermStructureHandle(
131         ql.BlackConstantVol(start, calendar, sigma, day_count)
132     )
133     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
134                                                 dividend_yield,
135                                                 flat_ts,
136                                                 flat_vol_ts)
137     option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
138     return option
139
140 def barrier_upout_call(S,K,X,r,q,tau,sigma,**kwargs):
141     option=_barrier_upout_call(S,K,X,r,q,tau,sigma)
142     epsilon=1e-4
143     option_S_upper=_barrier_upout_call(S+epsilon,K,X,r,q,tau,sigma)
144     option_S_lower=_barrier_upout_call(S-epsilon,K,X,r,q,tau,sigma)
145     option_sigam_upper=_barrier_upout_call(S,K,X,r,q,tau,sigma+epsilon)
146     option_sigam_lower=_barrier_upout_call(S,K,X,r,q,tau,sigma-epsilon)
147     option_tau_upper=_barrier_upout_call(S,K,X,r,q,tau+100*epsilon,sigma)
148     option_tau_lower=_barrier_upout_call(S,K,X,r,q,tau-100*epsilon,sigma)
149     option_r_upper=_barrier_upout_call(S,K,X,r+epsilon,q,tau,sigma)
150     option_r_lower=_barrier_upout_call(S,K,X,r-epsilon,q,tau,sigma)
151     return {
152         'PV':option.NPV(),
153         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
154         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
155         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
156         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
157         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
158     }
159
160 def _barrier_upout_put(S,K,X,r,q,tau,sigma):
161     # specify timeframe
162     tau_day=int(round(tau*365))
163     start=ql.Date(1,1,2000)
164     ql.Settings.instance().evaluationDate=start
165     maturity=start+tau_day
166     day_count = ql.Actual365Fixed()
167     calendar = ql.UnitedStates()
168     # specify option type
169     option_type = ql.Option.Put
170     payoff = ql.PlainVanillaPayoff(option_type, K)
171     exercise = ql.EuropeanExercise(maturity)
172     option=ql.BarrierOption(ql.Barrier.UpOut, X, 0, payoff, exercise)
173     # specify pricing method
174     spot_handle = ql.QuoteHandle(
175         ql.SimpleQuote(S)
176     )
177     flat_ts = ql.YieldTermStructureHandle(
178         ql.FlatForward(start, r, day_count)
179     )
180     dividend_yield = ql.YieldTermStructureHandle(
181         ql.FlatForward(start, q, day_count)

```

```

182     )
183     flat_vol_ts = ql.BlackVolTermStructureHandle(
184         ql.BlackConstantVol(start, calendar, sigma, day_count)
185     )
186     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
187                                                 dividend_yield,
188                                                 flat_ts,
189                                                 flat_vol_ts)
190     option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
191     return option
192
193 def barrier_upout_put(S,K,X,r,q,tau,sigma,**kwargs):
194     option=_barrier_upout_put(S,K,X,r,q,tau,sigma)
195     epsilon=1e-4
196     option_S_upper=_barrier_upout_put(S+epsilon,K,X,r,q,tau,sigma)
197     option_S_lower=_barrier_upout_put(S-epsilon,K,X,r,q,tau,sigma)
198     option_sigam_upper=_barrier_upout_put(S,K,X,r,q,tau,sigma+epsilon)
199     option_sigam_lower=_barrier_upout_put(S,K,X,r,q,tau,sigma-epsilon)
200     option_tau_upper=_barrier_upout_put(S,K,X,r,q,tau+100*epsilon,sigma)
201     option_tau_lower=_barrier_upout_put(S,K,X,r,q,tau-100*epsilon,sigma)
202     option_r_upper=_barrier_upout_put(S,K,X,r+epsilon,q,tau,sigma)
203     option_r_lower=_barrier_upout_put(S,K,X,r-epsilon,q,tau,sigma)
204     return {
205         'PV':option.NPV(),
206         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
207         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
208         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
209         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
210         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
211     }
212
213 def _barrier_downin_call(S,K,X,r,q,tau,sigma):
214     # specify timeframe
215     tau_day=int(round(tau*365))
216     start=ql.Date(1,1,2000)
217     ql.Settings.instance().evaluationDate=start
218     maturity=start+tau_day
219     day_count = ql.Actual365Fixed()
220     calendar = ql.UnitedStates()
221     # specify option type
222     option_type = ql.Option.Call
223     payoff = ql.PlainVanillaPayoff(option_type, K)
224     exercise = ql.EuropeanExercise(maturity)
225     option=ql.BarrierOption(ql.Barrier.DownIn, X, 0, payoff, exercise)
226     # specify pricing method
227     spot_handle = ql.QuoteHandle(
228         ql.SimpleQuote(S)
229     )
230     flat_ts = ql.YieldTermStructureHandle(
231         ql.FlatForward(start, r, day_count)
232     )
233     dividend_yield = ql.YieldTermStructureHandle(
234         ql.FlatForward(start, q, day_count)
235     )
236     flat_vol_ts = ql.BlackVolTermStructureHandle(
237         ql.BlackConstantVol(start, calendar, sigma, day_count)
238     )
239     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
240                                                 dividend_yield,
241                                                 flat_ts,
242                                                 flat_vol_ts)

```

```

243     option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
244     return option
245
246 def barrier_downin_call(S,K,X,r,q,tau,sigma,**kwargs):
247     option=_barrier_downin_call(S,K,X,r,q,tau,sigma)
248     epsilon=1e-4
249     option_S_upper=_barrier_downin_call(S+epsilon,K,X,r,q,tau,sigma)
250     option_S_lower=_barrier_downin_call(S-epsilon,K,X,r,q,tau,sigma)
251     option_sigam_upper=_barrier_downin_call(S,K,X,r,q,tau,sigma+epsilon)
252     option_sigam_lower=_barrier_downin_call(S,K,X,r,q,tau,sigma-epsilon)
253     option_tau_upper=_barrier_downin_call(S,K,X,r,q,tau+100*epsilon,sigma)
254     option_tau_lower=_barrier_downin_call(S,K,X,r,q,tau-100*epsilon,sigma)
255     option_r_upper=_barrier_downin_call(S,K,X,r+epsilon,q,tau,sigma)
256     option_r_lower=_barrier_downin_call(S,K,X,r-epsilon,q,tau,sigma)
257     return {
258         'PV':option.NPV(),
259         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
260         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
261         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
262         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
263         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
264     }
265
266 def _barrier_downin_put(S,K,X,r,q,tau,sigma):
267     # specify timeframe
268     tau_day=int(round(tau*365))
269     start=ql.Date(1,1,2000)
270     ql.Settings.instance().evaluationDate=start
271     maturity=start+tau_day
272     day_count = ql.Actual365Fixed()
273     calendar = ql.UnitedStates()
274     # specify option type
275     option_type = ql.Option.Put
276     payoff = ql.PlainVanillaPayoff(option_type, K)
277     exercise = ql.EuropeanExercise(maturity)
278     option=ql.BarrierOption(ql.Barrier.DownIn, X, 0, payoff, exercise)
279     # specify pricing method
280     spot_handle = ql.QuoteHandle(
281         ql.SimpleQuote(S)
282     )
283     flat_ts = ql.YieldTermStructureHandle(
284         ql.FlatForward(start, r, day_count)
285     )
286     dividend_yield = ql.YieldTermStructureHandle(
287         ql.FlatForward(start, q, day_count)
288     )
289     flat_vol_ts = ql.BlackVolTermStructureHandle(
290         ql.BlackConstantVol(start, calendar, sigma, day_count)
291     )
292     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
293                                                 dividend_yield,
294                                                 flat_ts,
295                                                 flat_vol_ts)
296     option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
297     return option
298
299 def barrier_downin_put(S,K,X,r,q,tau,sigma,**kwargs):
300     option=_barrier_downin_put(S,K,X,r,q,tau,sigma)
301     epsilon=1e-4
302     option_S_upper=_barrier_downin_put(S+epsilon,K,X,r,q,tau,sigma)
303     option_S_lower=_barrier_downin_put(S-epsilon,K,X,r,q,tau,sigma)

```

```

304     option_sigam_upper=_barrier_downin_put(S,K,X,r,q,tau,sigma+epsilon)
305     option_sigam_lower=_barrier_downin_put(S,K,X,r,q,tau,sigma-epsilon)
306     option_tau_upper=_barrier_downin_put(S,K,X,r,q,tau+100*epsilon,sigma)
307     option_tau_lower=_barrier_downin_put(S,K,X,r,q,tau-100*epsilon,sigma)
308     option_r_upper=_barrier_downin_put(S,K,X,r+epsilon,q,tau,sigma)
309     option_r_lower=_barrier_downin_put(S,K,X,r-epsilon,q,tau,sigma)
310     return {
311         'PV':option.NPV(),
312         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
313         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2)
314         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
315         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
316         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
317     }
318
319 def _barrier_downout_call(S,K,X,r,q,tau,sigma):
320     # specify timeframe
321     tau_day=int(round(tau*365))
322     start=ql.Date(1,1,2000)
323     ql.Settings.instance().evaluationDate=start
324     maturity=start+tau_day
325     day_count = ql.Actual365Fixed()
326     calendar = ql.UnitedStates()
327     # specify option type
328     option_type = ql.Option.Call
329     payoff = ql.PlainVanillaPayoff(option_type, K)
330     exercise = ql.EuropeanExercise(maturity)
331     option=ql.BarrierOption(ql.Barrier.DownOut, X, 0, payoff, exercise)
332     # specify pricing method
333     spot_handle = ql.QuoteHandle(
334         ql.SimpleQuote(S)
335     )
336     flat_ts = ql.YieldTermStructureHandle(
337         ql.FlatForward(start, r, day_count)
338     )
339     dividend_yield = ql.YieldTermStructureHandle(
340         ql.FlatForward(start, q, day_count)
341     )
342     flat_vol_ts = ql.BlackVolTermStructureHandle(
343         ql.BlackConstantVol(start, calendar, sigma, day_count)
344     )
345     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
346                                                 dividend_yield,
347                                                 flat_ts,
348                                                 flat_vol_ts)
349     option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
350     return option
351
352 def barrier_downout_call(S,K,X,r,q,tau,sigma,**kwargs):
353     option=_barrier_downout_call(S,K,X,r,q,tau,sigma)
354     epsilon=1e-4
355     option_S_upper=_barrier_downout_call(S+epsilon,K,X,r,q,tau,sigma)
356     option_S_lower=_barrier_downout_call(S-epsilon,K,X,r,q,tau,sigma)
357     option_sigam_upper=_barrier_downout_call(S,K,X,r,q,tau,sigma+epsilon)
358     option_sigam_lower=_barrier_downout_call(S,K,X,r,q,tau,sigma-epsilon)
359     option_tau_upper=_barrier_downout_call(S,K,X,r,q,tau+100*epsilon,sigma)
360     option_tau_lower=_barrier_downout_call(S,K,X,r,q,tau-100*epsilon,sigma)
361     option_r_upper=_barrier_downout_call(S,K,X,r+epsilon,q,tau,sigma)
362     option_r_lower=_barrier_downout_call(S,K,X,r-epsilon,q,tau,sigma)
363     return {
364         'PV':option.NPV(),

```

```

365     'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
366     'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
367     'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
368     'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
369     'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
370   }
371
372 def _barrier_downout_put(S,K,X,r,q,tau,sigma):
373   # specify timeframe
374   tau_day=int(round(tau*365))
375   start=ql.Date(1,1,2000)
376   ql.Settings.instance().evaluationDate=start
377   maturity=start+tau_day
378   day_count = ql.Actual365Fixed()
379   calendar = ql.UnitedStates()
380   # specify option type
381   option_type = ql.Option.Put
382   payoff = ql.PlainVanillaPayoff(option_type, K)
383   exercise = ql.EuropeanExercise(maturity)
384   option=ql.BarrierOption(ql.Barrier.DownOut, X, 0, payoff, exercise)
385   # specify pricing method
386   spot_handle = ql.QuoteHandle(
387     ql.SimpleQuote(S)
388   )
389   flat_ts = ql.YieldTermStructureHandle(
390     ql.FlatForward(start, r, day_count)
391   )
392   dividend_yield = ql.YieldTermStructureHandle(
393     ql.FlatForward(start, q, day_count)
394   )
395   flat_vol_ts = ql.BlackVolTermStructureHandle(
396     ql.BlackConstantVol(start, calendar, sigma, day_count)
397   )
398   bsm_process = ql.BlackScholesMertonProcess(spot_handle,
399                                             dividend_yield,
400                                             flat_ts,
401                                             flat_vol_ts)
402   option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
403   return option
404
405 def barrier_downout_put(S,K,X,r,q,tau,sigma,**kwargs):
406   option=_barrier_downout_put(S,K,X,r,q,tau,sigma)
407   epsilon=1e-4
408   option_S_upper=_barrier_downout_put(S+epsilon,K,X,r,q,tau,sigma)
409   option_S_lower=_barrier_downout_put(S-epsilon,K,X,r,q,tau,sigma)
410   option_sigam_upper=_barrier_downout_put(S,K,X,r,q,tau,sigma+epsilon)
411   option_sigam_lower=_barrier_downout_put(S,K,X,r,q,tau,sigma-epsilon)
412   option_tau_upper=_barrier_downout_put(S,K,X,r,q,tau+100*epsilon,sigma)
413   option_tau_lower=_barrier_downout_put(S,K,X,r,q,tau-100*epsilon,sigma)
414   option_r_upper=_barrier_downout_put(S,K,X,r+epsilon,q,tau,sigma)
415   option_r_lower=_barrier_downout_put(S,K,X,r-epsilon,q,tau,sigma)
416   return {
417     'PV':option.NPV(),
418     'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
419     'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
420     'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
421     'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
422     'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
423   }

```

1.4 Asian

In [4]:

```

1 def asian_geometric_call(S,K,r,q,tau,sigma,freq='1M',**kwargs):
2     # specify timeframe
3     tau_day=int(round(tau*365))
4     start=ql.Date(1,1,2000)
5     ql.Settings.instance().evaluationDate=start
6     maturity=start+tau_day
7     day_count = ql.Actual365Fixed()
8     calendar = ql.UnitedStates()
9     period=ql.Period(freq)
10    asianFutureFixingDates = [start+period]
11    while asianFutureFixingDates[-1]<=maturity:
12        asianFutureFixingDates.append(asianFutureFixingDates[-1]+period)
13    asianFutureFixingDates.pop(-1)
14    # specify option type
15    option_type = ql.Option.Call
16    payoff = ql.PlainVanillaPayoff(option_type, K)
17    exercise = ql.EuropeanExercise(maturity)
18    option = ql.DiscreteAveragingAsianOption(ql.Average().Geometric,
19                                              1, 0,
20                                              asianFutureFixingDates,
21                                              payoff,
22                                              exercise)
23    # specify pricing method
24    spot_handle = ql.QuoteHandle(
25        ql.SimpleQuote(S)
26    )
27    flat_ts = ql.YieldTermStructureHandle(
28        ql.FlatForward(start, r, day_count)
29    )
30    dividend_yield = ql.YieldTermStructureHandle(
31        ql.FlatForward(start, q, day_count)
32    )
33    flat_vol_ts = ql.BlackVolTermStructureHandle(
34        ql.BlackConstantVol(start, calendar, sigma, day_count)
35    )
36    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
37                                                dividend_yield,
38                                                flat_ts,
39                                                flat_vol_ts)
40    option.setPricingEngine(ql.AnalyticDiscreteGeometricAveragePriceAsianEngine(bsm_pr
41    return {
42        'PV':option.NPV(),
43        'delta':option.delta(),
44        'gamma':option.gamma(),
45        'vega':option.vega(),
46        'theta':option.theta(),
47        'rho':option.rho(),
48    }
49
50 def asian_geometric_put(S,K,r,q,tau,sigma,freq='1M',**kwargs):
51     # specify timeframe
52     tau_day=int(round(tau*365))
53     start=ql.Date(1,1,2000)
54     ql.Settings.instance().evaluationDate=start
55     maturity=start+tau_day
56     day_count = ql.Actual365Fixed()
57     calendar = ql.UnitedStates()
58     period=ql.Period(freq)
59     asianFutureFixingDates = [start+period]
```

```

60     while asianFutureFixingDates[-1] <= maturity:
61         asianFutureFixingDates.append(asianFutureFixingDates[-1] + period)
62     asianFutureFixingDates.pop(-1)
63     # specify option type
64     option_type = ql.Option.Put
65     payoff = ql.PlainVanillaPayoff(option_type, K)
66     exercise = ql.EuropeanExercise(maturity)
67     option = ql.DiscreteAveragingAsianOption(ql.Average().Geometric,
68                                              1, 0,
69                                              asianFutureFixingDates,
70                                              payoff,
71                                              exercise)
72     # specify pricing method
73     spot_handle = ql.QuoteHandle(
74         ql.SimpleQuote(S)
75     )
76     flat_ts = ql.YieldTermStructureHandle(
77         ql.FlatForward(start, r, day_count)
78     )
79     dividend_yield = ql.YieldTermStructureHandle(
80         ql.FlatForward(start, q, day_count)
81     )
82     flat_vol_ts = ql.BlackVolTermStructureHandle(
83         ql.BlackConstantVol(start, calendar, sigma, day_count)
84     )
85     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
86                                                 dividend_yield,
87                                                 flat_ts,
88                                                 flat_vol_ts)
89     option.setPricingEngine(ql.AnalyticDiscreteGeometricAveragePriceAsianEngine(bsm_pr
90     return {
91         'PV':option.NPV(),
92         'delta':option.delta(),
93         'gamma':option.gamma(),
94         'vega':option.vega(),
95         'theta':option.theta(),
96         'rho':option.rho(),
97     }
98
99 def _asian_arithmetic_call(S,K,r,q,tau,sigma,freq='1M'):
100     # specify timeframe
101     tau_day=int(round(tau*365))
102     start=ql.Date(1,1,2000)
103     ql.Settings.instance().evaluationDate=start
104     maturity=start+tau_day
105     day_count = ql.Actual365Fixed()
106     calendar = ql.UnitedStates()
107     period=ql.Period(freq)
108     asianFutureFixingDates = [start+period]
109     while asianFutureFixingDates[-1] <= maturity:
110         asianFutureFixingDates.append(asianFutureFixingDates[-1] + period)
111     asianFutureFixingDates.pop(-1)
112     # specify option type
113     option_type = ql.Option.Call
114     payoff = ql.PlainVanillaPayoff(option_type, K)
115     exercise = ql.EuropeanExercise(maturity)
116     option = ql.DiscreteAveragingAsianOption(ql.Average().Arithmetic,
117                                              0, 0,
118                                              asianFutureFixingDates,
119                                              payoff,
120                                              exercise)

```

```

121 # specify pricing method
122 spot_handle = ql.QuoteHandle(
123     ql.SimpleQuote(S)
124 )
125 flat_ts = ql.YieldTermStructureHandle(
126     ql.FlatForward(start, r, day_count)
127 )
128 dividend_yield = ql.YieldTermStructureHandle(
129     ql.FlatForward(start, q, day_count)
130 )
131 flat_vol_ts = ql.BlackVolTermStructureHandle(
132     ql.BlackConstantVol(start, calendar, sigma, day_count)
133 )
134 bsm_process = ql.BlackScholesMertonProcess(spot_handle,
135                                             dividend_yield,
136                                             flat_ts,
137                                             flat_vol_ts)
138 numPaths=10000
139 option.setPricingEngine(ql.MCDiscreteArithmeticAPEngine(bsm_process, 'lowdiscrepancy',
140                                                       antitheticVariate=True,
141                                                       controlVariate=True,
142                                                       requiredSamples=numPaths, s
143 )
144
145 def asian_arithmetic_call(S,K,r,q,tau,sigma,freq='1M',**kwargs):
146     option=_asian_arithmetic_call(S,K,r,q,tau,sigma,freq)
147     epsilon=1e-4
148     option_S_upper=_asian_arithmetic_call(S+epsilon,K,r,q,tau,sigma,freq)
149     option_S_lower=_asian_arithmetic_call(S-epsilon,K,r,q,tau,sigma,freq)
150     option_sigam_upper=_asian_arithmetic_call(S,K,r,q,tau,sigma+epsilon,freq)
151     option_sigam_lower=_asian_arithmetic_call(S,K,r,q,tau,sigma-epsilon,freq)
152     option_tau_upper=_asian_arithmetic_call(S,K,r,q,tau+100*epsilon,sigma,freq)
153     option_tau_lower=_asian_arithmetic_call(S,K,r,q,tau-100*epsilon,sigma,freq)
154     option_r_upper=_asian_arithmetic_call(S,K,r+epsilon,q,tau,sigma,freq)
155     option_r_lower=_asian_arithmetic_call(S,K,r-epsilon,q,tau,sigma,freq)
156     return {
157         'PV':option.NPV(),
158         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
159         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
160         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
161         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
162         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
163     }
164
165 def _asian_arithmetic_put(S,K,r,q,tau,sigma,freq='1M'):
166     # specify timeframe
167     tau_day=int(round(tau*365))
168     start=ql.Date(1,1,2000)
169     ql.Settings.instance().evaluationDate=start
170     maturity=start+tau_day
171     day_count = ql.Actual365Fixed()
172     calendar = ql.UnitedStates()
173     period=ql.Period(freq)
174     asianFutureFixingDates = [start+period]
175     while asianFutureFixingDates[-1]<=maturity:
176         asianFutureFixingDates.append(asianFutureFixingDates[-1]+period)
177     asianFutureFixingDates.pop(-1)
178     # specify option type
179     option_type = ql.Option.Put
180     payoff = ql.PlainVanillaPayoff(option_type, K)
181     exercise = ql.EuropeanExercise(maturity)

```

```

182     option = ql.DiscreteAveragingAsianOption(ql.Average().Arithmetic,
183                                         0, 0,
184                                         asianFutureFixingDates,
185                                         payoff,
186                                         exercise)
187     # specify pricing method
188     spot_handle = ql.QuoteHandle(
189         ql.SimpleQuote(S)
190     )
191     flat_ts = ql.YieldTermStructureHandle(
192         ql.FlatForward(start, r, day_count)
193     )
194     dividend_yield = ql.YieldTermStructureHandle(
195         ql.FlatForward(start, q, day_count)
196     )
197     flat_vol_ts = ql.BlackVolTermStructureHandle(
198         ql.BlackConstantVol(start, calendar, sigma, day_count)
199     )
200     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
201                                                 dividend_yield,
202                                                 flat_ts,
203                                                 flat_vol_ts)
204     numPaths=10000
205     option.setPricingEngine(ql.MCDiscreteArithmeticAPEngine(bsm_process, 'lowdiscrepancy',
206                                                               antitheticVariate=True,
207                                                               controlVariate=True,
208                                                               requiredSamples=numPaths,
209     )
210
211 def asian_arithmetic_put(S,K,r,q,tau,sigma,freq='1M',**kwargs):
212     option=_asian_arithmetic_put(S,K,r,q,tau,sigma,freq)
213     epsilon=1e-4
214     option_S_upper=_asian_arithmetic_put(S+epsilon,K,r,q,tau,sigma,freq)
215     option_S_lower=_asian_arithmetic_put(S-epsilon,K,r,q,tau,sigma,freq)
216     option_sigam_upper=_asian_arithmetic_put(S,K,r,q,tau,sigma+epsilon,freq)
217     option_sigam_lower=_asian_arithmetic_put(S,K,r,q,tau,sigma-epsilon,freq)
218     option_tau_upper=_asian_arithmetic_put(S,K,r,q,tau+100*epsilon,sigma,freq)
219     option_tau_lower=_asian_arithmetic_put(S,K,r,q,tau-100*epsilon,sigma,freq)
220     option_r_upper=_asian_arithmetic_put(S,K,r+epsilon,q,tau,sigma,freq)
221     option_r_lower=_asian_arithmetic_put(S,K,r-epsilon,q,tau,sigma,freq)
222     return {
223         'PV':option.NPV(),
224         'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
225         'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2),
226         'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
227         'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
228         'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
229     }
230
231 def asian_continuous_geometric_call(S,K,r,q,tau,sigma,**kwargs):
232     # specify timeframe
233     tau_day=int(round(tau*365))
234     start=ql.Date(1,1,2000)
235     ql.Settings.instance().evaluationDate=start
236     maturity=start+tau_day
237     day_count = ql.Actual365Fixed()
238     calendar = ql.UnitedStates()
239     # specify option type
240     option_type = ql.Option.Call
241     payoff = ql.PlainVanillaPayoff(option_type, K)
242     exercise = ql.EuropeanExercise(maturity)

```

```

243     option = ql.ContinuousAveragingAsianOption(ql.Average().Geometric,
244                                         payoff,
245                                         exercise)
246     # specify pricing method
247     spot_handle = ql.QuoteHandle(
248         ql.SimpleQuote(S)
249     )
250     flat_ts = ql.YieldTermStructureHandle(
251         ql.FlatForward(start, r, day_count)
252     )
253     dividend_yield = ql.YieldTermStructureHandle(
254         ql.FlatForward(start, q, day_count)
255     )
256     flat_vol_ts = ql.BlackVolTermStructureHandle(
257         ql.BlackConstantVol(start, calendar, sigma, day_count)
258     )
259     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
260                                              dividend_yield,
261                                              flat_ts,
262                                              flat_vol_ts)
263     option.setPricingEngine(ql.AnalyticContinuousGeometricAveragePriceAsianEngine(bsm_
264     return {
265         'PV':option.NPV(),
266         'delta':option.delta(),
267         'gamma':option.gamma(),
268         'vega':option.vega(),
269         'theta':option.theta(),
270         'rho':option.rho(),
271     }
272
273 def asian_continuous_geometric_put(S,K,r,q,tau,sigma,**kwargs):
274     # specify timeframe
275     tau_day=int(round(tau*365))
276     start=ql.Date(1,1,2000)
277     ql.Settings.instance().evaluationDate=start
278     maturity=start+tau_day
279     day_count = ql.Actual365Fixed()
280     calendar = ql.UnitedStates()
281     # specify option type
282     option_type = ql.Option.Put
283     payoff = ql.PlainVanillaPayoff(option_type, K)
284     exercise = ql.EuropeanExercise(maturity)
285     option = ql.ContinuousAveragingAsianOption(ql.Average().Geometric,
286                                                 payoff,
287                                                 exercise)
288     # specify pricing method
289     spot_handle = ql.QuoteHandle(
290         ql.SimpleQuote(S)
291     )
292     flat_ts = ql.YieldTermStructureHandle(
293         ql.FlatForward(start, r, day_count)
294     )
295     dividend_yield = ql.YieldTermStructureHandle(
296         ql.FlatForward(start, q, day_count)
297     )
298     flat_vol_ts = ql.BlackVolTermStructureHandle(
299         ql.BlackConstantVol(start, calendar, sigma, day_count)
300     )
301     bsm_process = ql.BlackScholesMertonProcess(spot_handle,
302                                               dividend_yield,
303                                               flat_ts,

```

```

304                                         flat_vol_ts)
305     option.setPricingEngine(ql.AnalyticContinuousGeometricAveragePriceAsianEngine(bsm_
306     return {
307         'PV':option.NPV(),
308         'delta':option.delta(),
309         'gamma':option.gamma(),
310         'vega':option.vega(),
311         'theta':option.theta(),
312         'rho':option.rho(),
313     }

```

1.5 Visualization

In [5]:

```

1 # =====
2 # Visualization
3 # =====
4 def option_plot(option_fcn,changing_param_name,changing_param_bound,other_params,num=100):
5     import numpy as np
6     import pandas as pd
7     import numba as nb
8     import plotly.express as px
9     from math import ceil
10    from plotly.subplots import make_subplots
11    import plotly.graph_objects as go
12    params=other_params.copy()
13    def f(x):
14        params[changing_param_name]=x
15        return option_fcn(**params)
16    f=np.vectorize(f)
17    x=np.linspace(changing_param_bound[0],changing_param_bound[1],num)
18    y=f(x)
19    df=pd.DataFrame(np.array(list(map(lambda x:list(x.values()),y))),columns=y[0].keys())
20    df.index.name=changing_param_name
21    # for i,(name,col) in enumerate(df.iteritems()):
22    #     fig=px.line(col,y=name,color_discrete_sequence=[px.colors.qualitative.D3[i%10]]*10)
23    #     fig.show()
24    col_num=3
25    fig = make_subplots(rows=ceil(len(df.columns)/col_num), cols=col_num)
26    for i,(name,col) in enumerate(df.iteritems()):
27        fig.add_trace(go.Scatter(x=df.index,y=col),row=i//col_num+1,col=i%col_num+1)
28        fig.update_xaxes(title_text=changing_param_name, row=i//col_num+1, col=i%col_num+1)
29        fig.update_yaxes(title_text=name, row=i//col_num+1, col=i%col_num+1)
30    fig.update_layout(showlegend=False,height=ceil(len(df.columns)/col_num)*300, width=1000)
31    fig.show()
32    return df

```

In [6]:

```

1 def interact_plot(changing_param_name,other_params,num=100,**kwargs):
2     from functools import partial
3     import numpy as np
4     from ipywidgets import interact
5     from ipywidgets import FloatSlider, Dropdown, FloatRangeSlider
6     min_dict={
7         'S':0,
8         'K':0,
9         'X':0,
10        'r':-0.05,
11        'q':0,
12        'tau':0,
13        'sigma':0
14    }
15    max_dict={
16        'S':200,
17        'K':200,
18        'X':200,
19        'r':0.2,
20        'q':0.5,
21        'tau':5,
22        'sigma':1
23    }
24    option_list=[
25        ('European Call',vanilla_euro_call),
26        ('European Put',vanilla_euro_put),
27        ('American Call',vanilla_ame_call),
28        ('American Put',vanilla_ame_put),
29        ('Barrier UpIn Call',barrier_upin_call),
30        ('Barrier UpIn Put',barrier_upin_put),
31        ('Barrier UpOut Call',barrier_upout_call),
32        ('Barrier UpOut Put',barrier_upout_put),
33        ('Barrier DownIn Call',barrier_downin_call),
34        ('Barrier DownIn Put',barrier_downin_put),
35        ('Barrier DownOut Call',barrier_downout_call),
36        ('Barrier DownOut Put',barrier_downout_put),
37        ('Asian Geometric Call',asian_geometric_call),
38        ('Asian Geometric Put',asian_geometric_put),
39        ('Asian Arithmetic Call',asian_arithmetic_call),
40        ('Asian Arithmetic Put',asian_arithmetic_put),
41        ('Asian Continuous Geometric Call',asian_continuous_geometric_call),
42        ('Asian Continuous Geometric Put',asian_continuous_geometric_put)
43    ]
44    widgets_dict={'option_fcn': Dropdown(
45        options=option_list,
46        description='Option Type:'
47    ),
48        'changing_param_bound':FloatRangeSlider(
49            value=[min_dict[changing_param_name]+(
50                max_dict[changing_param_name]-min_dict[changing_param_name])*(
51                    min_dict[changing_param_name]+(
52                        max_dict[changing_param_name]-min_dict[changing_param_name])
53                min=min_dict[changing_param_name],
54                max=max_dict[changing_param_name],
55                step=10**((round(np.log10(max_dict[changing_param_name]-min_dict[changing_param_name]))+1)),
56                description='x range:',
57                disabled=False,
58                continuous_update=False,
59                orientation='horizontal',

```

```
60         readout=True,
61         readout_format='.{2f}' if np.log10(max_dict[changing_param_name])-min_dict[changing_param_name]>1 else '.{1f}' ),
62     )
63     for key,value in other_params.items():
64         widgets_dict[key]=FloatSlider(
65             value=value,
66             min=min_dict[key],
67             max=max_dict[key],
68             step=10**round(np.log10(max_dict[key]-min_dict[key]))-2),
69             description=key,
70             disabled=False,
71             continuous_update=False,
72             orientation='horizontal',
73             readout=True,
74             readout_format='.{3f}' ,
75         )
76     def f(option_fcn,changing_param_bound,**other_params):
77         return option_plot(option_fcn,changing_param_name,changing_param_bound,other_params)
78     interact(f,**widgets_dict)
```

2 Interact Plot

In [7]:

```

1 changing_param_name='S'
2 other_params={
3     'S':100,
4     'K':100,
5     'X':120,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 interact_plot(changing_param_name,other_params)

```

Option Type: European Call

x range: 50 – 150

S 100.000

K 100.000

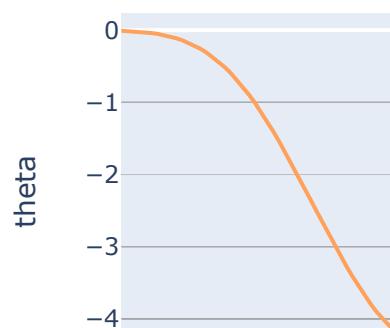
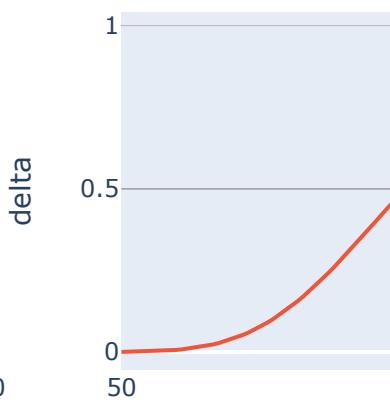
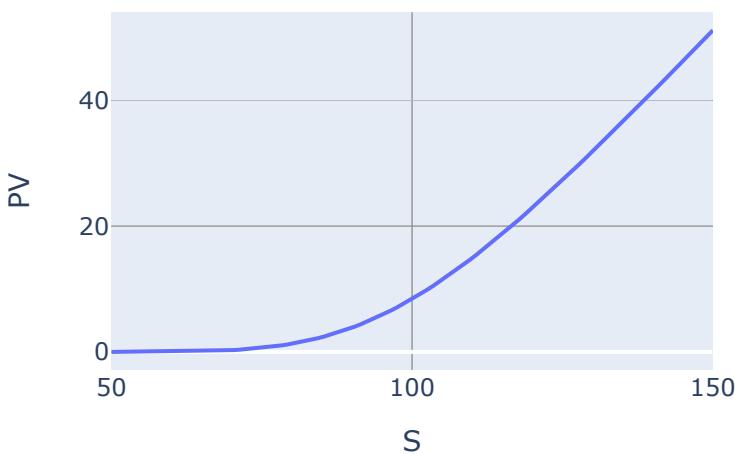
X 120.000

r 0.010

q 0.000

tau 1.000

sigma 0.200





S	PV	delta	gamma	vega	theta	rho
50.000000	0.001142	0.000457	0.000164	0.081760	-0.008393	0.021709
51.010101	0.001696	0.000651	0.000222	0.115627	-0.011878	0.031489
52.020202	0.002479	0.000911	0.000297	0.160847	-0.016534	0.044931
53.030303	0.003566	0.001257	0.000392	0.220269	-0.022658	0.063119
54.040404	0.005055	0.001710	0.000509	0.297176	-0.030591	0.087365
...
145.959596	47.192566	0.979365	0.001703	7.256971	-1.683248	95.755089
146.969697	48.182670	0.981020	0.001576	6.806594	-1.640634	95.997506
147.979798	49.174382	0.982550	0.001457	6.379459	-1.600178	96.223239
148.989899	50.167582	0.983965	0.001346	5.974824	-1.561815	96.433292
150.000000	51.162155	0.985272	0.001243	5.591925	-1.525479	96.628624

100 rows × 6 columns

3 Vanilla European

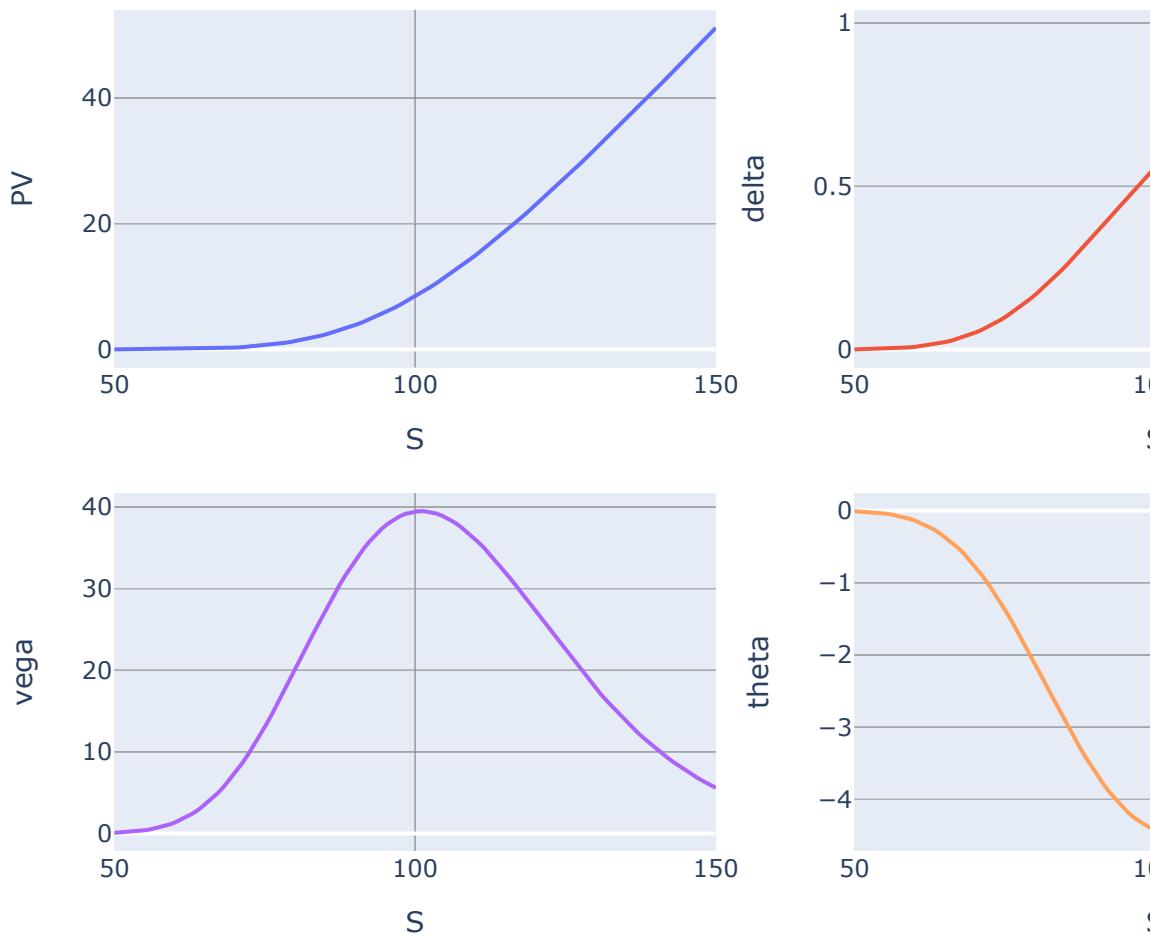
3.1 ...Vs S, call

In [8]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)

```



Out[8]:

S	PV	delta	gamma	vega	theta	rho
50.000000	0.001142	0.000457	0.000164	0.081760	-0.008393	0.021709
51.010101	0.001696	0.000651	0.000222	0.115627	-0.011878	0.031489
52.020202	0.002479	0.000911	0.000297	0.160847	-0.016534	0.044931

S	PV	delta	gamma	vega	theta	rho
53.030303	0.003566	0.001257	0.000392	0.220269	-0.022658	0.063119
54.040404	0.005055	0.001710	0.000509	0.297176	-0.030591	0.087365
...
145.959596	47.192566	0.979365	0.001703	7.256971	-1.683248	95.755089
146.969697	48.182670	0.981020	0.001576	6.806594	-1.640634	95.997506
147.979798	49.174382	0.982550	0.001457	6.379459	-1.600178	96.223239
148.989899	50.167582	0.983965	0.001346	5.974824	-1.561815	96.433292
150.000000	51.162155	0.985272	0.001243	5.591925	-1.525479	96.628624

100 rows × 6 columns

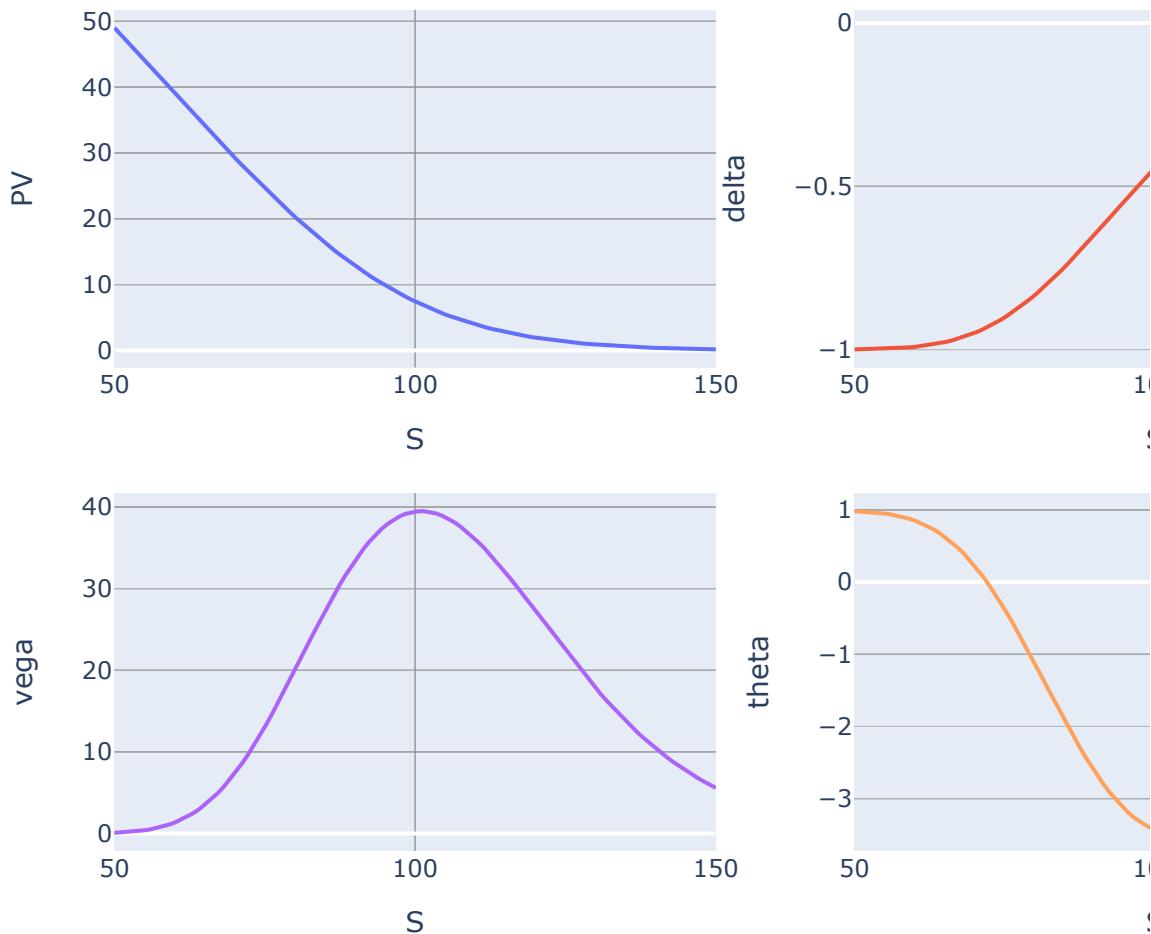
3.2 ...Vs S, put

In [9]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)

```



Out[9]:

S	PV	delta	gamma	vega	theta	rho
50.000000	49.006125	-0.999543	0.000164	0.081760	0.981657	-98.983275
51.010101	47.996579	-0.999349	0.000222	0.115627	0.978172	-98.973495
52.020202	46.987260	-0.999089	0.000297	0.160847	0.973516	-98.960052

S	PV	delta	gamma	vega	theta	rho
53.030303	45.978246	-0.998743	0.000392	0.220269	0.967392	-98.941864
54.040404	44.969634	-0.998290	0.000509	0.297176	0.959459	-98.917619
...
145.959596	0.237954	-0.020635	0.001703	7.256971	-0.693198	-3.249894
146.969697	0.217957	-0.018980	0.001576	6.806594	-0.650585	-3.007477
147.979798	0.199568	-0.017450	0.001457	6.379459	-0.610128	-2.781744
148.989899	0.182666	-0.016035	0.001346	5.974824	-0.571765	-2.571692

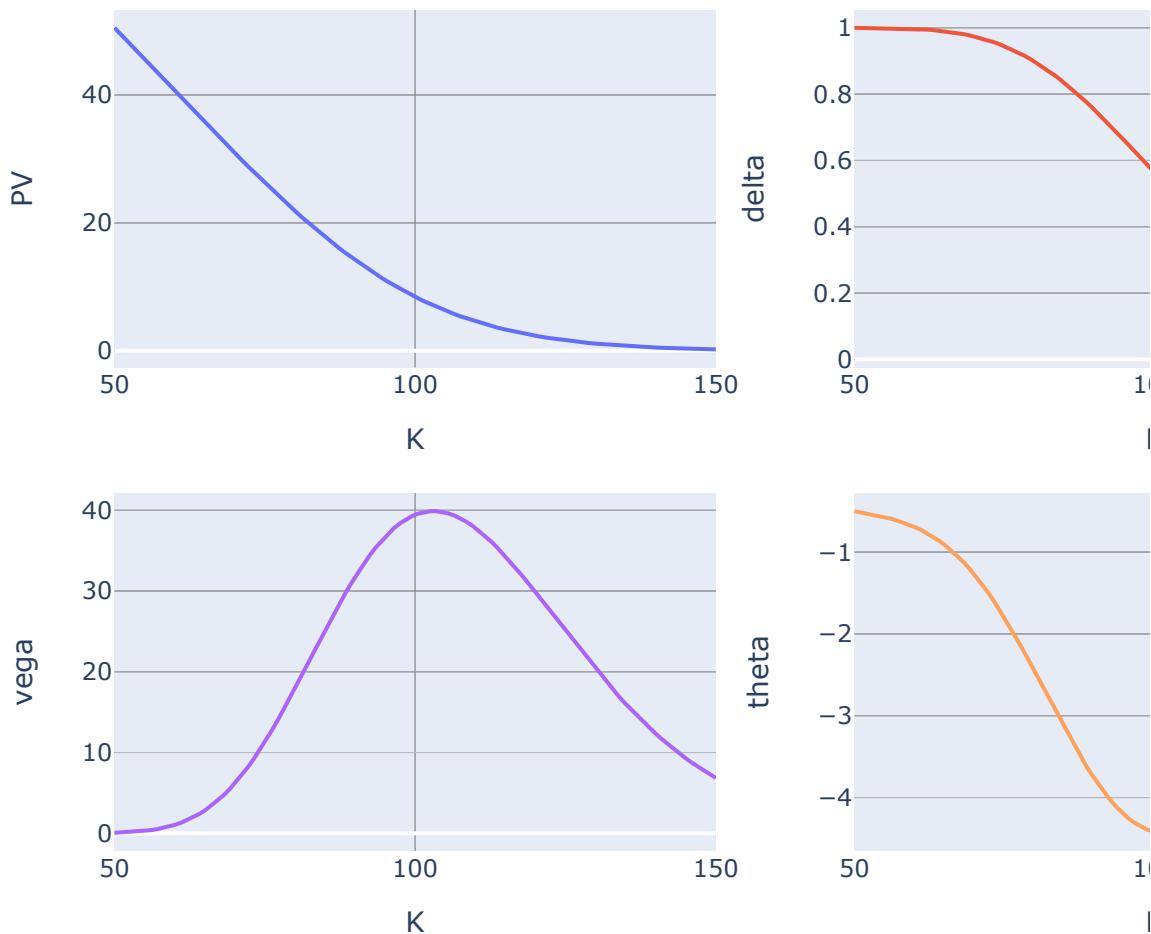
3.3 ...Vs K, call

In [10]:

```

1 changing_param_name='K'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)

```



Out[10]:

K	PV	delta	gamma	vega	theta	rho
50.000000	50.498278	0.999850	0.000029	0.057813	-0.500649	49.486747
51.010101	49.498611	0.999781	0.000041	0.082583	-0.513053	50.479462
52.020202	48.499108	0.999684	0.000058	0.116011	-0.526294	51.469315

K	PV	delta	gamma	vega	theta	rho
53.030303	47.499824	0.999552	0.000080	0.160404	-0.540595	52.455413
54.040404	46.500833	0.999375	0.000109	0.218461	-0.556213	53.436661
...
145.959596	0.308328	0.040859	0.004384	8.767418	-0.914518	3.777619
146.969697	0.283210	0.037926	0.004126	8.251706	-0.860264	3.509364
147.979798	0.260041	0.035185	0.003880	7.760417	-0.808626	3.258410
148.989899	0.238681	0.032625	0.003646	7.292955	-0.759534	3.023817

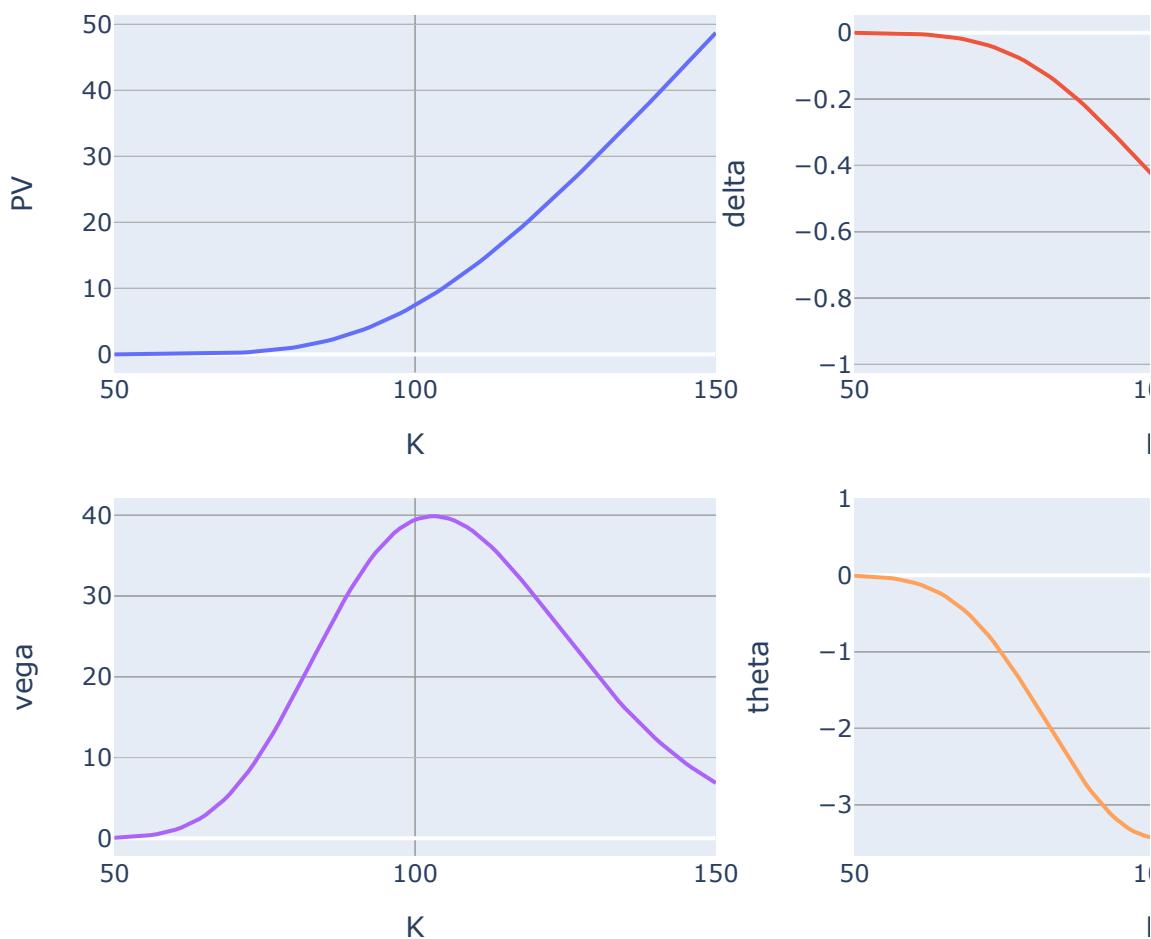
3.4 ...Vs K, put

In [11]:

```

1 changing_param_name='K'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)

```



Out[11]:

K	PV	delta	gamma	vega	theta	rho
50.000000	0.000769	-0.000150	0.000029	0.057813	-0.005624	-0.015744
51.010101	0.001153	-0.000219	0.000041	0.082583	-0.008027	-0.023080
52.020202	0.001700	-0.000316	0.000058	0.116011	-0.011268	-0.033278

K	PV	delta	gamma	vega	theta	rho
53.030303	0.002467	-0.000448	0.000080	0.160404	-0.015568	-0.047229
54.040404	0.003526	-0.000625	0.000109	0.218461	-0.021186	-0.066032
...
145.959596	44.815602	-0.959141	0.004384	8.767418	0.530555	-140.729655
146.969697	45.790534	-0.962074	0.004126	8.251706	0.594809	-141.997960
147.979798	46.767416	-0.964815	0.003880	7.760417	0.656448	-143.248964
148.989899	47.746106	-0.967375	0.003646	7.292955	0.715541	-144.483607

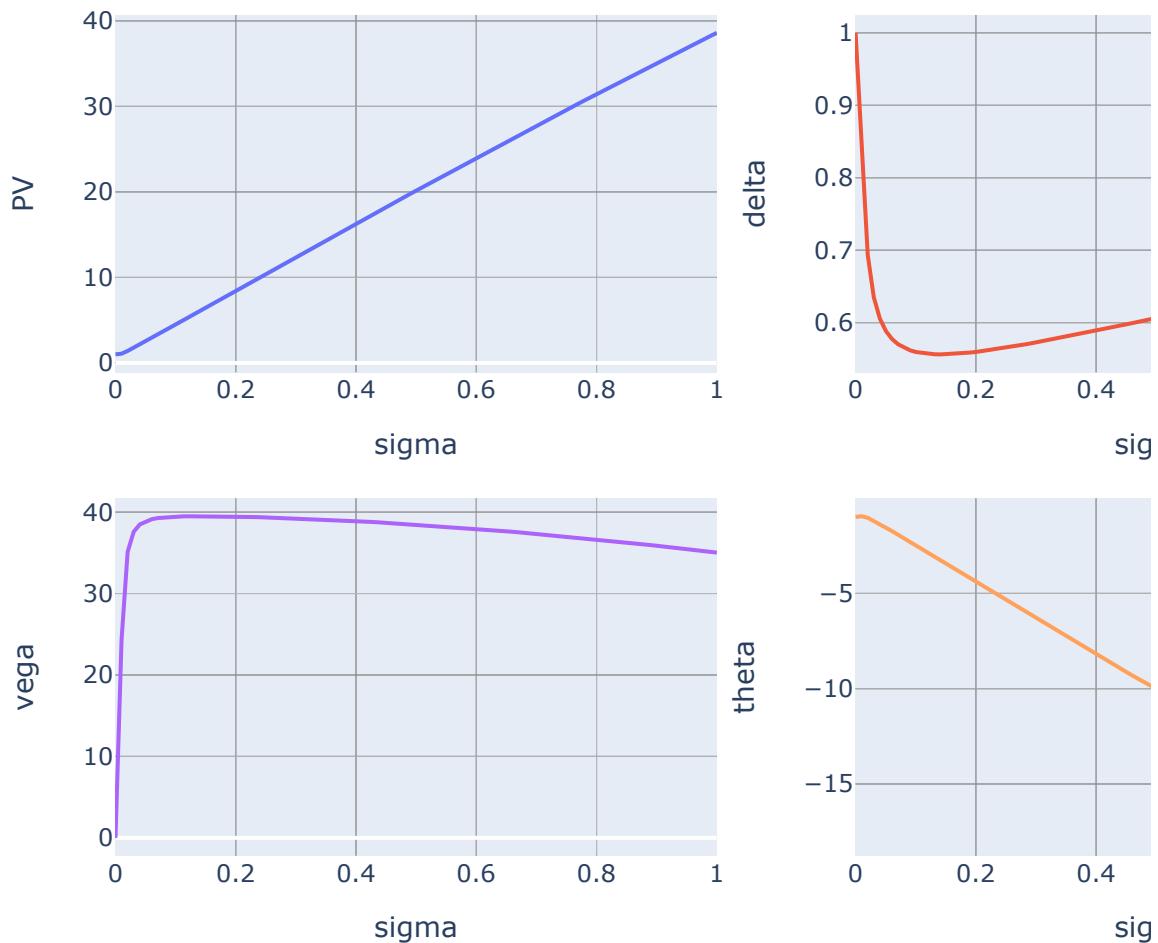
3.5 ...Vs σ , call

In [12]:

```

1 changing_param_name='sigma'
2 changing_param_bound=(1e-8,1)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)

```



Out[12]:

sigma	PV	delta	gamma	vega	theta	rho
1.000000e-08	0.995017	1.000000	0.000000	0.000000	-0.990050	99.004983
1.010102e-02	1.080360	0.840144	0.240737	24.316858	-0.952153	82.934031
2.020203e-02	1.395712	0.693256	0.173826	35.116400	-1.034010	67.929885

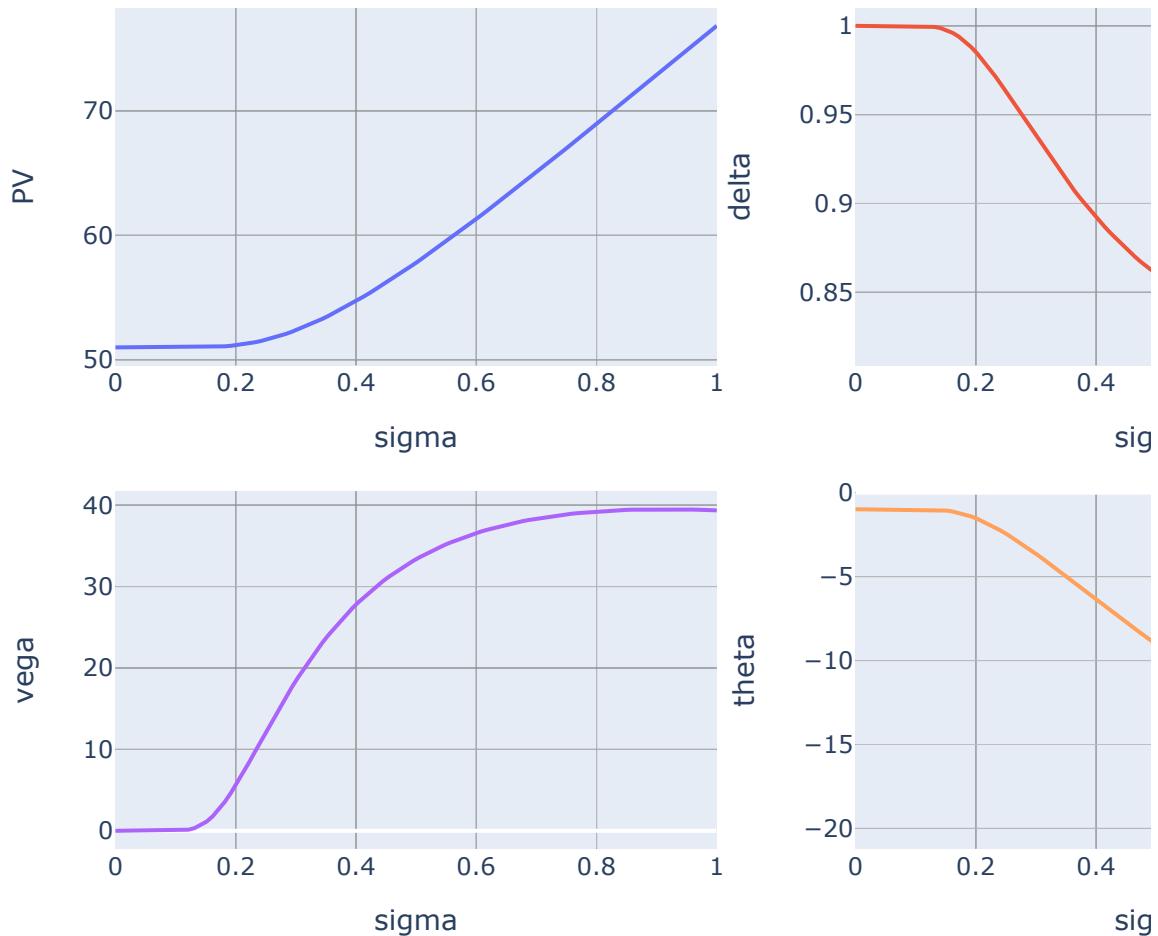
sigma	PV	delta	gamma	vega	theta	rho
3.030304e-02	1.765266	0.635010	0.124038	37.587325	-1.186862	61.735707
4.040405e-02	2.150132	0.605536	0.095263	38.490039	-1.361611	58.403431
...
9.595960e-01	37.178851	0.688011	0.003687	35.377439	-17.290246	31.622203
9.696970e-01	37.535765	0.689757	0.003639	35.291405	-17.425383	31.439911
9.797980e-01	37.891806	0.691499	0.003593	35.204681	-17.559319	31.258143
9.898990e-01	38.246968	0.693239	0.003548	35.117273	-17.692046	31.076900

In [13]:

```

1 changing_param_name='sigma'
2 changing_param_bound=(1e-8,1)
3 other_params={
4     'S':150,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)

```



Out[13]:

	PV	delta	gamma	vega	theta	rho
sigma						
1.000000e-08	50.995017	1.000000	0.000000e+00	0.000000e+00	-0.990050	99.004983
1.010102e-02	50.995017	1.000000	0.000000e+00	0.000000e+00	-0.990050	99.004983
2.020203e-02	50.995017	1.000000	1.544760e-93	7.021639e-91	-0.990050	99.004983

sigma	PV	delta	gamma	vega	theta	rho
3.030304e-02	50.995017	1.000000	1.084253e-42	7.392636e-40	-0.990050	99.004983
4.040405e-02	50.995017	1.000000	5.861327e-25	5.328480e-22	-0.990050	99.004983
...
9.595960e-01	75.244066	0.819315	1.827342e-03	3.945397e+01	-19.406466	47.653131
9.696970e-01	75.642496	0.819457	1.807415e-03	3.943450e+01	-19.592520	47.276022
9.797980e-01	76.040712	0.819623	1.787747e-03	3.941169e+01	-19.776773	46.902781
9.898990e-01	76.438681	0.819813	1.768334e-03	3.938562e+01	-19.959226	46.533323

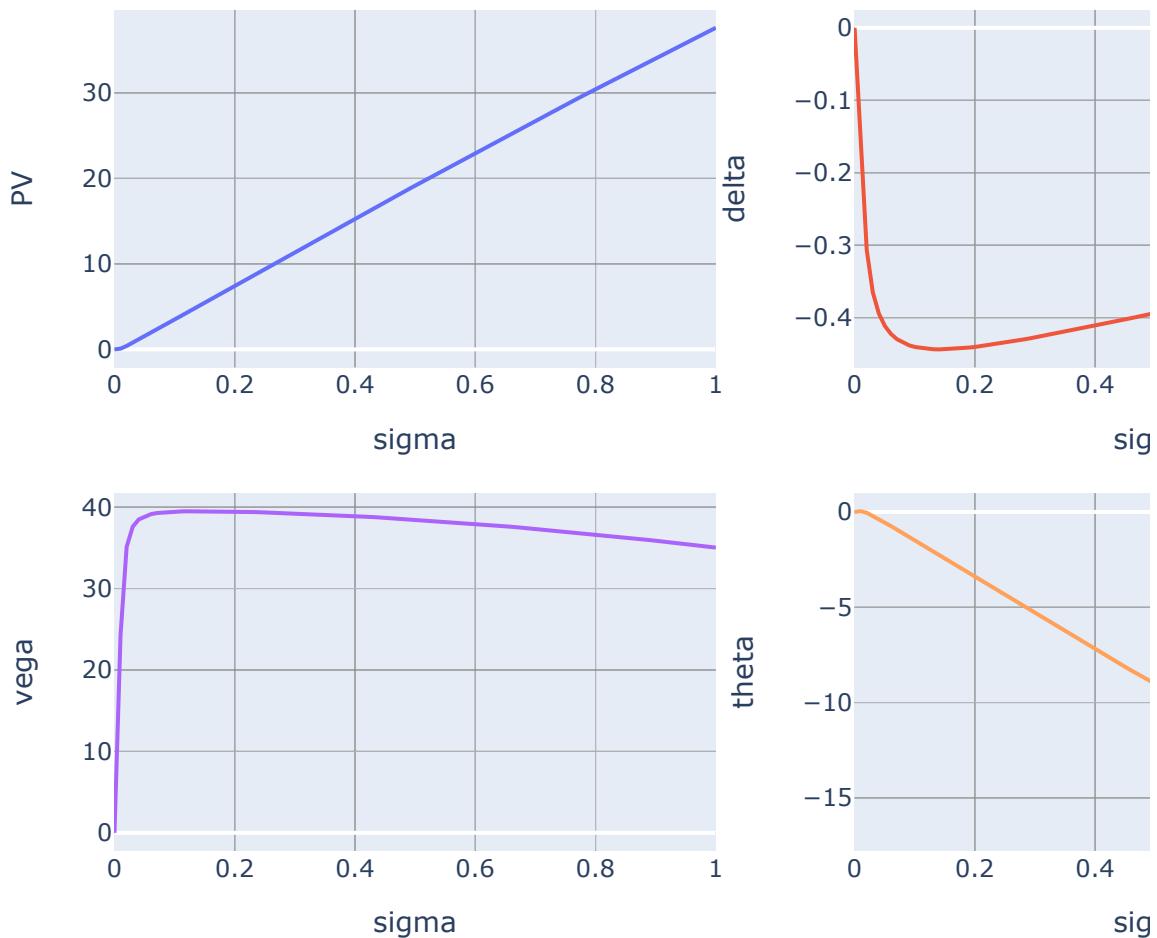
3.6 ...Vs σ , put

In [14]:

```

1 changing_param_name='sigma'
2 changing_param_bound=(1e-8,1)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)

```



◀ ▶ ⌂

Out[14]:

sigma	PV	delta	gamma	vega	theta	rho
1.000000e-08	-1.104690e-16	-1.110223e-16	0.000000	0.000000	1.099176e-16	-1.099176e-14
1.010102e-02	8.534364e-02	-1.598561e-01	0.240737	24.316858	3.789699e-02	-1.607095e+01

PV	delta	gamma	vega	theta	rho	
sigma						
2.020203e-02	4.006958e-01	-3.067440e-01	0.173826	35.116400	-4.396030e-02	-3.107510e+01
3.030304e-02	7.702495e-01	-3.649903e-01	0.124038	37.587325	-1.968123e-01	-3.726928e+01
4.040405e-02	1.155115e+00	-3.944644e-01	0.095263	38.490039	-3.715612e-01	-4.060155e+01
...
9.595960e-01	3.618383e+01	-3.119895e-01	0.003687	35.377439	-1.630020e+01	-6.738278e+01
9.696970e-01	3.654075e+01	-3.102432e-01	0.003639	35.291405	-1.643533e+01	-6.756507e+01
9.797980e-01	3.689679e+01	-3.085005e-01	0.003593	35.204681	-1.656927e+01	-6.774684e+01
9.898990e-01	3.725195e+01	-3.067613e-01	0.003548	35.117273	-1.670200e+01	-6.792808e+01
1.000000e+00	3.760623e+01	-3.050257e-01	0.003503	35.029188	-1.683351e+01	-6.810880e+01

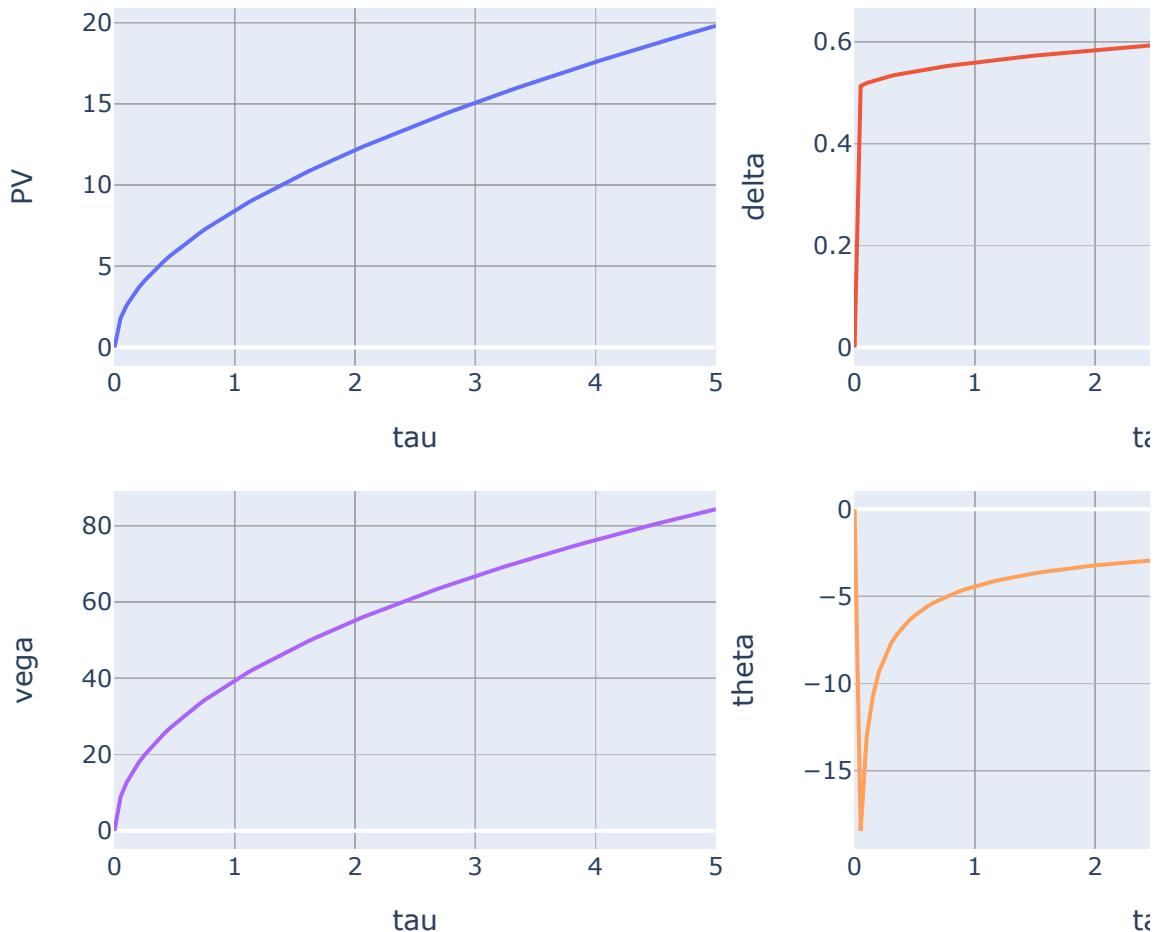
3.7 ...Vs τ , call

In [15]:

```

1 changing_param_name='tau'
2 changing_param_bound=(1e-8,5)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)

```



◀ ▶

Out[15]:

tau	PV	delta	gamma	vega	theta	rho
1.000000e-08	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5.050506e-02	1.796040	0.513287	0.089774	8.854396	-18.450074	2.442704
1.010101e-01	2.589620	0.519045	0.062579	12.687300	-13.009000	4.999047

	PV	delta	gamma	vega	theta	rho
tau						
1.515152e-01	3.170000	0.523216	0.051299	15.459974	-10.751317	7.406408
2.020202e-01	3.689930	0.526924	0.044200	17.922102	-9.329980	9.934749
...
4.797980e+00	19.372820	0.628748	0.008629	82.788152	-2.160758	208.690414
4.848485e+00	19.485018	0.629420	0.008577	83.187375	-2.150016	210.736565
4.898990e+00	19.590798	0.630053	0.008529	83.562919	-2.139987	212.671406
4.949495e+00	19.701922	0.630717	0.008479	83.956551	-2.129555	214.709933
5.000000e+00	19.806701	0.631342	0.008433	84.326874	-2.119813	216.637577

100 rows × 6 columns

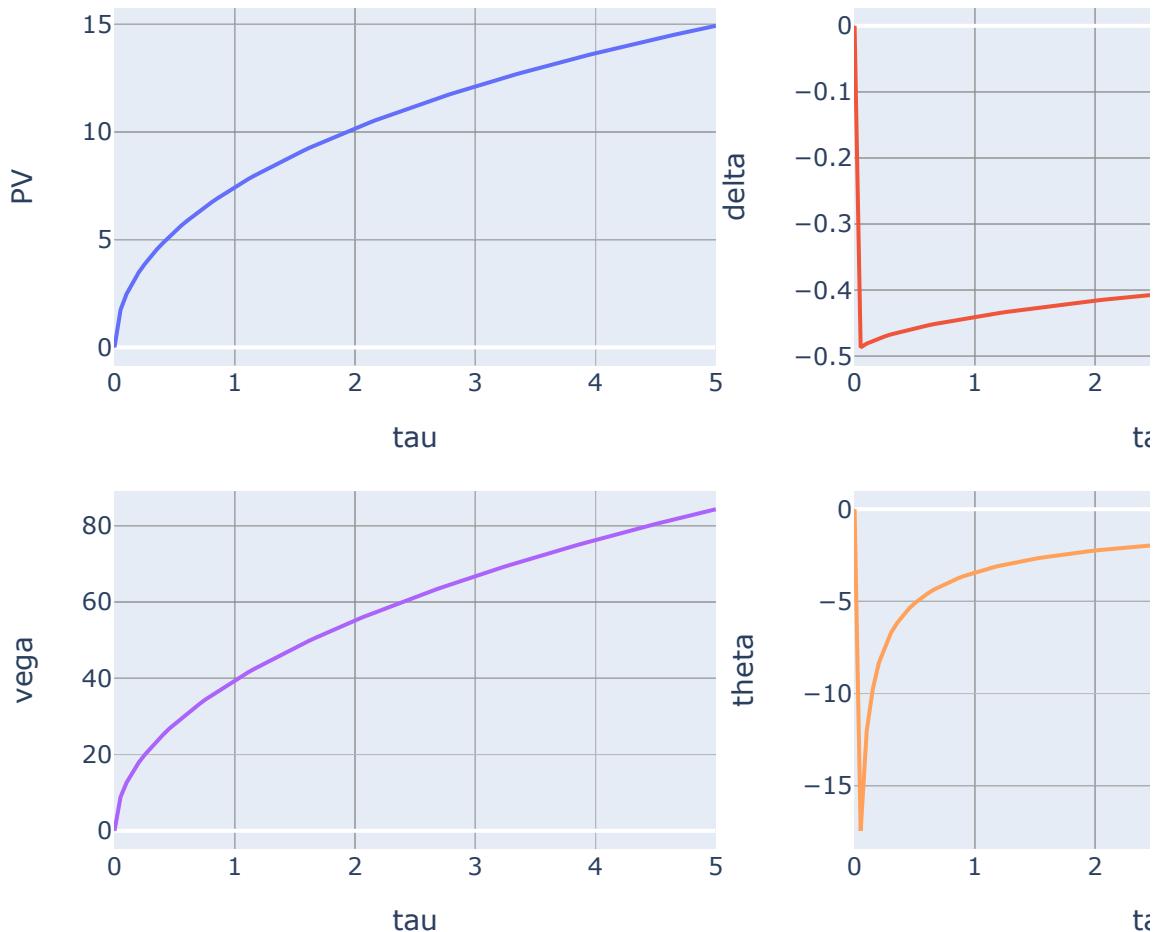
3.8 ...Vs τ , put

In [16]:

```

1 changing_param_name='tau'
2 changing_param_bound=(1e-8,5)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)

```



◀ ▶ ⌂

Out[16]:

	PV	delta	gamma	vega	theta	rho
tau						
1.000000e-08	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5.050506e-02	1.746737	-0.486713	0.089774	8.854396	-17.450567	-2.486371
1.010101e-01	2.488302	-0.480955	0.062579	12.687300	-12.010013	-5.127669

	PV	delta	gamma	vega	theta	rho	
tau							
1.515152e-01	3.019429	-0.476784	0.051299	15.459974	-9.752823	-7.639396	
2.020202e-01	3.487396	-0.473076	0.044200	17.922102	-8.332006	-10.298162	
...
4.797980e+00	14.688810	-0.371252	0.008629	82.788152	-1.207598	-248.565199	
4.848485e+00	14.751404	-0.370580	0.008577	83.187375	-1.197352	-251.240158	
4.898990e+00	14.810215	-0.369947	0.008529	83.562919	-1.187793	-253.773301	
4.949495e+00	14.871786	-0.369283	0.008479	83.956551	-1.177856	-256.446078	

4 Vanilla American

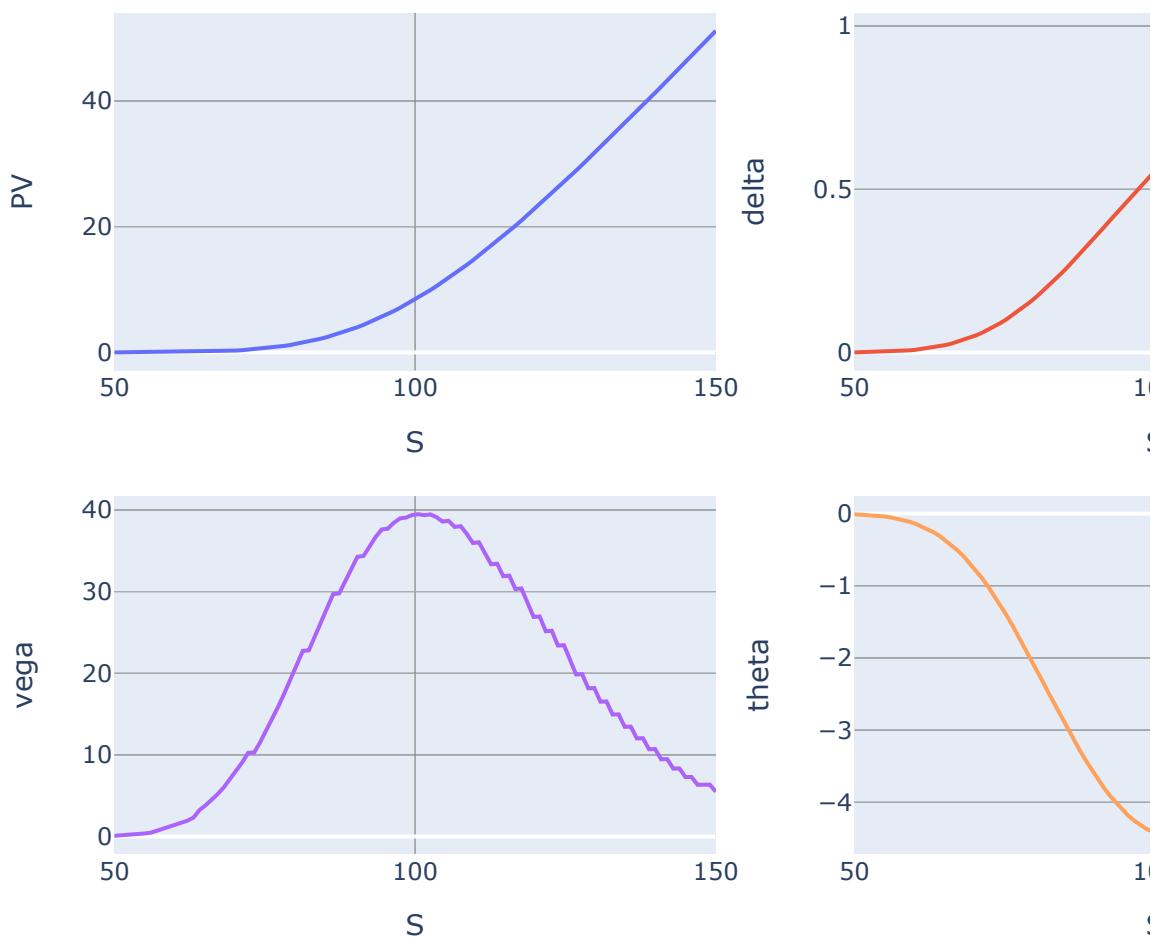
4.1 ...Vs S, call

In [17]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)

```



Out[17]:

S	PV	delta	gamma	vega	theta	rho
50.000000	0.001120	0.000448	0.000160	0.087330	-0.008233	0.021282
51.010101	0.001673	0.000641	0.000219	0.113079	-0.011704	0.031025
52.020202	0.002438	0.000896	0.000292	0.145595	-0.016265	0.044164

S	PV	delta	gamma	vega	theta	rho
53.030303	0.003515	0.001239	0.000386	0.236188	-0.022337	0.062185
54.040404	0.005008	0.001693	0.000504	0.299018	-0.030280	0.086462
...
145.959596	47.192486	0.979389	0.001701	7.292678	-1.682550	95.754873
146.969697	48.181677	0.981101	0.001571	6.333714	-1.638708	96.006608
147.979798	49.174270	0.982575	0.001455	6.344536	-1.599432	96.223049
148.989899	50.166864	0.984028	0.001342	6.355357	-1.560157	96.439501
150.000000	51.161979	0.985299	0.001241	5.489084	-1.524635	96.628850

100 rows × 6 columns

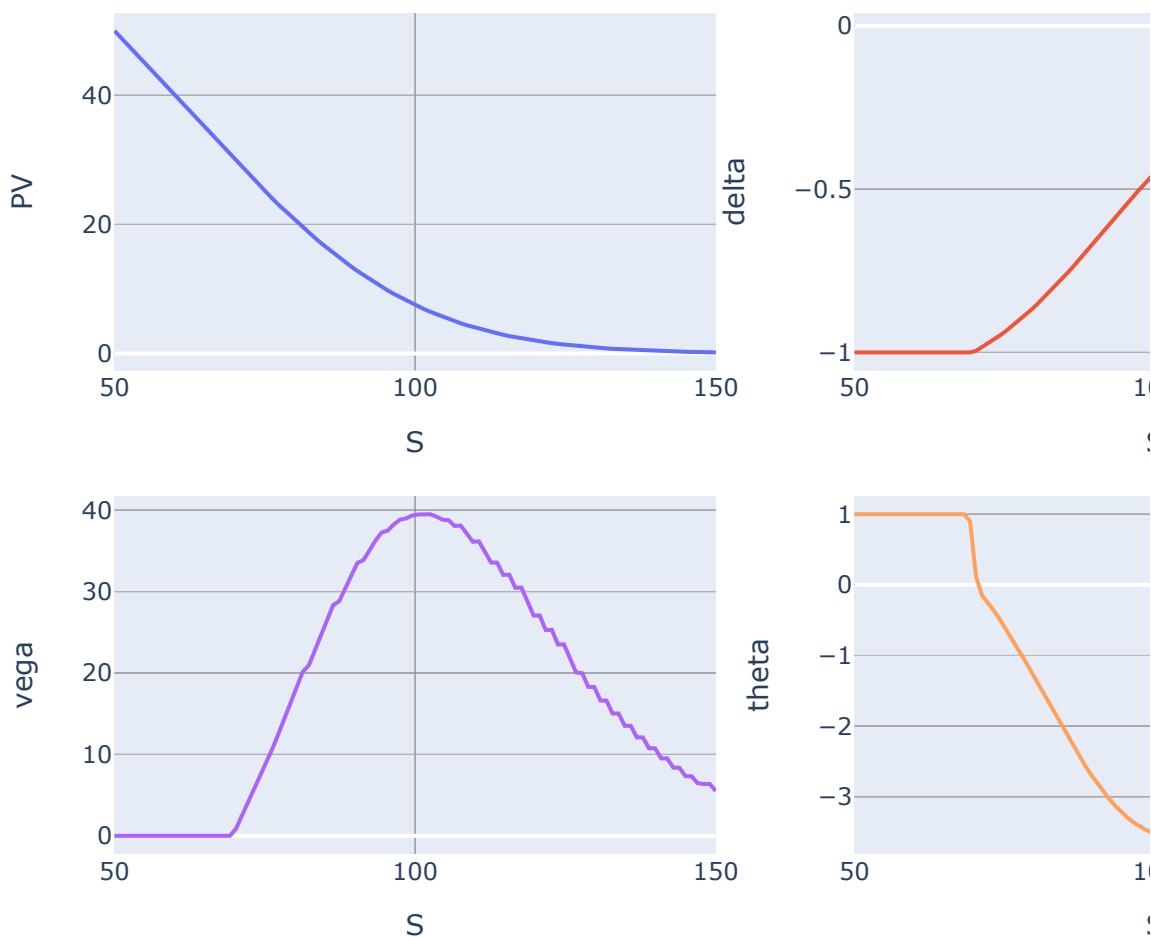
4.2 ...Vs S, put

In [18]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)

```



Out[18]:

	PV	delta	gamma	vega	theta	rho
S						
50.000000	50.000000	-1.000000	0.000000	0.000000	1.000000	0.000000
51.010101	48.989899	-1.000000	0.000000	0.000000	1.000000	0.000000
52.020202	47.979798	-1.000000	0.000000	0.000000	1.000000	0.000000

S	PV	delta	gamma	vega	theta	rho
53.030303	46.969697	-1.000000	0.000000	0.000000	1.000000	0.000000
54.040404	45.959596	-1.000000	0.000000	0.000000	1.000000	0.000000
...
145.959596	0.239001	-0.020721	0.001712	7.317031	-0.696821	-3.124263
146.969697	0.218004	-0.019000	0.001581	6.461240	-0.652696	-2.881830
147.979798	0.200379	-0.017515	0.001464	6.367541	-0.613053	-2.678137
148.989899	0.182810	-0.016055	0.001350	6.370930	-0.573549	-2.469471

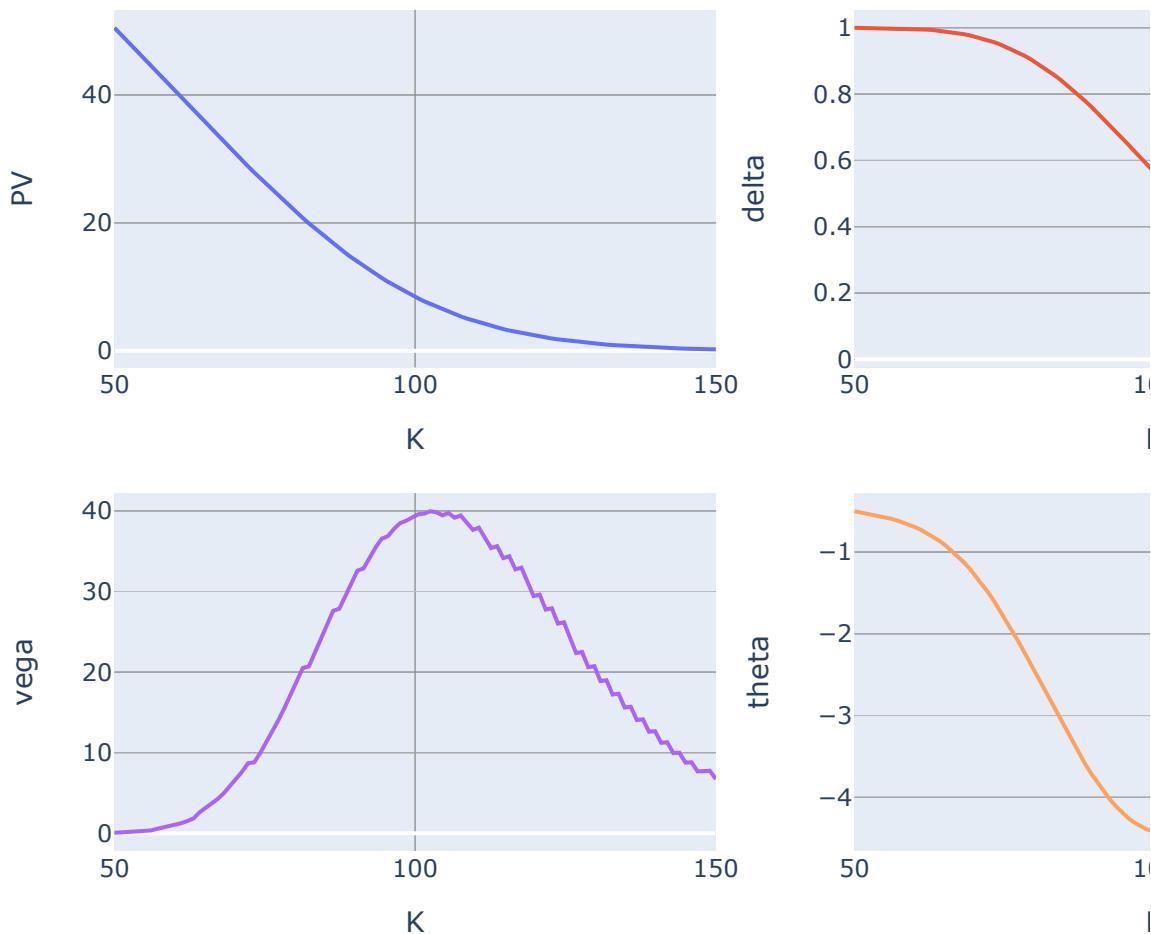
4.3 ...Vs K, call

In [19]:

```

1 changing_param_name='K'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)

```



◀ ▶

Out[19]:

K	PV	delta	gamma	vega	theta	rho
50.000000	50.498257	0.999853	0.000028	0.061561	-0.500543	49.484427
51.010101	49.498589	0.999784	0.000041	0.080572	-0.512939	50.477157
52.020202	48.499073	0.999689	0.000057	0.104824	-0.526114	51.467239

K	PV	delta	gamma	vega	theta	rho
53.030303	47.499783	0.999559	0.000079	0.171813	-0.540379	52.453465
54.040404	46.500795	0.999381	0.000108	0.219631	-0.556006	53.434673
...
145.959596	0.308167	0.040803	0.004378	8.810819	-0.913350	3.772004
146.969697	0.281945	0.037762	0.004113	7.678700	-0.857612	3.494124
147.979798	0.259847	0.035128	0.003874	7.718209	-0.807411	3.252833
148.989899	0.237750	0.032494	0.003635	7.757718	-0.757211	3.011543

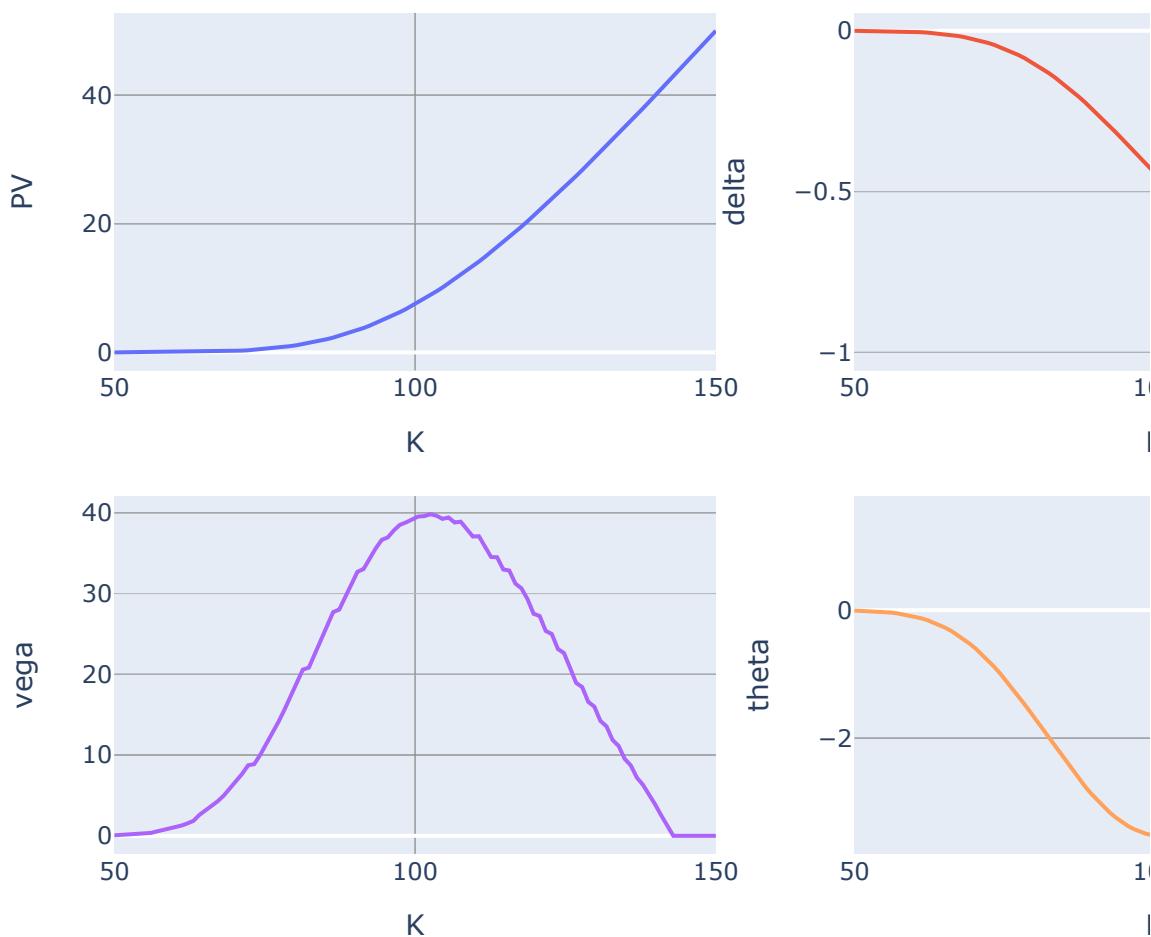
4.4 ...Vs K, put

In [20]:

```

1 changing_param_name='K'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)

```



Out[20]:

K	PV	delta	gamma	vega	theta	rho
50.000000	0.000757	-0.000147	0.000028	0.061833	-0.005538	-0.015209
51.010101	0.001142	-0.000217	0.000041	0.080931	-0.007942	-0.022408
52.020202	0.001677	-0.000311	0.000057	0.105394	-0.011127	-0.032199

K	PV	delta	gamma	vega	theta	rho
53.030303	0.002441	-0.000443	0.000079	0.172256	-0.015411	-0.045726
54.040404	0.003507	-0.000621	0.000109	0.220254	-0.021060	-0.064214
...
145.959596	45.959596	-1.000000	0.000000	0.000000	1.459596	0.000000
146.969697	46.969697	-1.000000	0.000000	0.000000	1.469697	0.000000
147.979798	47.979798	-1.000000	0.000000	0.000000	1.479798	0.000000
148.989899	48.989899	-1.000000	0.000000	0.000000	1.489899	0.000000

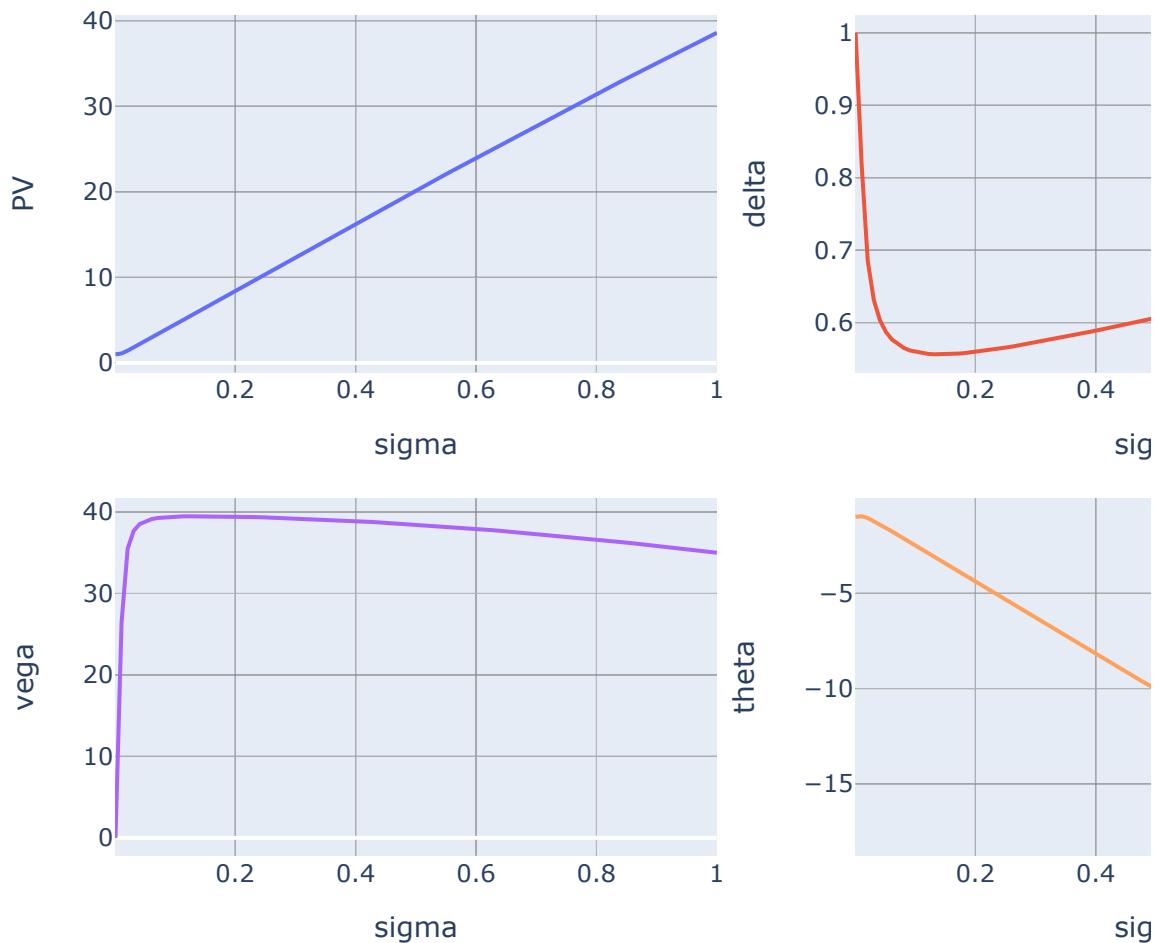
4.5 ...Vs σ , call

In [21]:

```

1 changing_param_name='sigma'
2 changing_param_bound=(0.001,1)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)

```



Out[21]:

	PV	delta	gamma	vega	theta	rho
sigma						
0.001000	0.995010	1.000000	8.324049e-12	4.996004e-07	-0.990050	99.001302
0.011091	1.105327	0.817828	2.388222e-01	2.642145e+01	-0.953660	80.674429
0.021182	1.429995	0.685296	1.678538e-01	3.549448e+01	-1.047551	67.097100

sigma	PV	delta	gamma	vega	theta	rho
0.031273	1.801326	0.631300	1.207336e-01	3.769919e+01	-1.203664	61.326327
0.041364	2.186511	0.603475	9.328717e-02	3.853054e+01	-1.379659	58.158827
...
0.959636	37.164572	0.687905	3.688355e-03	3.535075e+01	-17.299324	31.637947
0.969727	37.520859	0.689647	3.641063e-03	3.526440e+01	-17.434188	31.456037
0.979818	37.876269	0.691385	3.594675e-03	3.517721e+01	-17.567847	31.274804
0.989909	38.230798	0.693120	3.549163e-03	3.508938e+01	-17.700292	31.094248

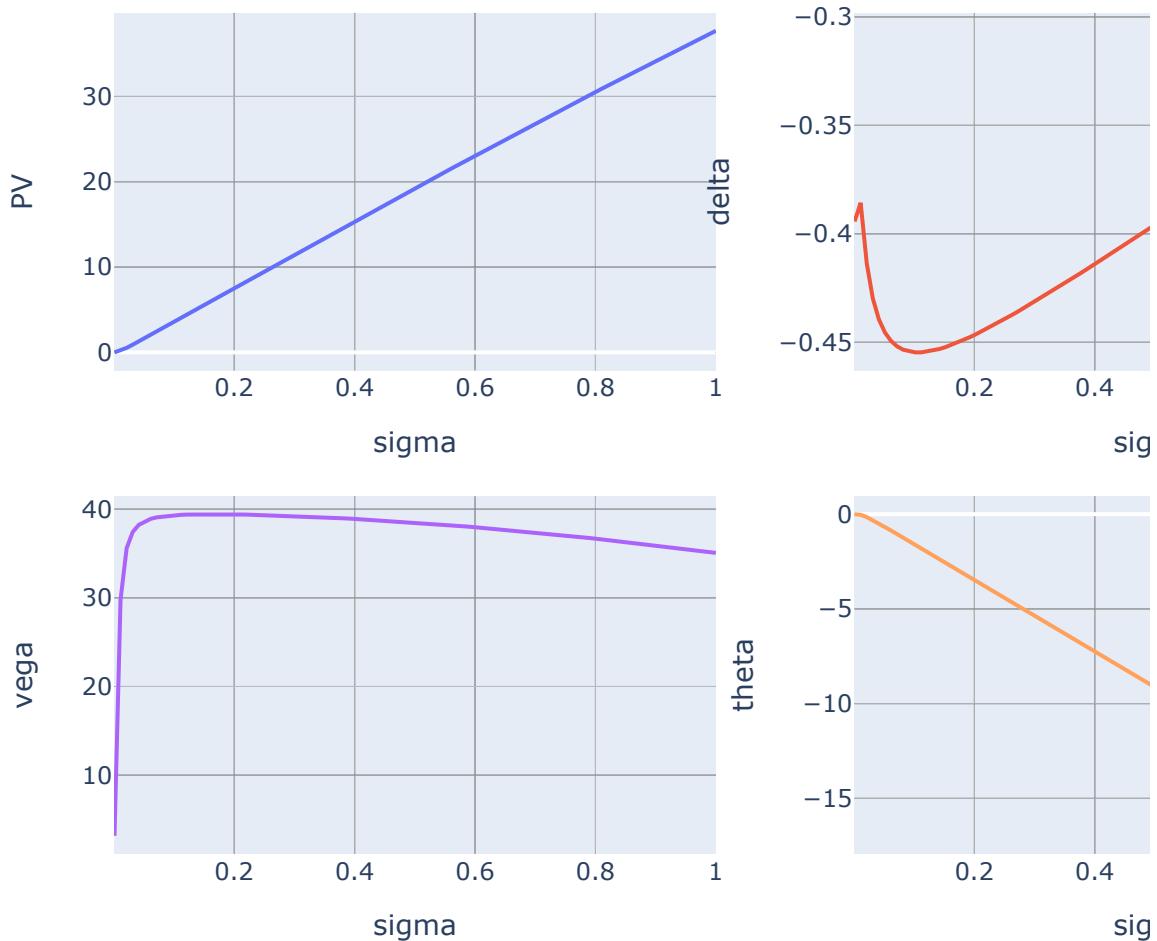
4.6 ...Vs σ , put

In [22]:

```

1 changing_param_name='sigma'
2 changing_param_bound=(0.001,1)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)

```



◀ ▶

Out[22]:

sigma	PV	delta	gamma	vega	theta	rho
0.001000	0.001702	-0.394316	79.550129	3.160911	-0.003418	-0.145938
0.011091	0.197148	-0.385738	0.682846	29.803719	-0.032269	-13.370055
0.021182	0.533548	-0.413564	0.256234	35.593787	-0.155923	-22.160783

sigma	PV	delta	gamma	vega	theta	rho
0.031273	0.903340	-0.429767	0.154444	37.419664	-0.316417	-26.929764
0.041364	1.285497	-0.439440	0.110364	38.228869	-0.491846	-29.960064
...
0.959636	36.268768	-0.313421	0.003717	35.414729	-16.439993	-56.304143
0.969727	36.625705	-0.311666	0.003670	35.329278	-16.575722	-56.431415
0.979818	36.981775	-0.309916	0.003623	35.243020	-16.710250	-56.558737
0.989909	37.336972	-0.308169	0.003577	35.156163	-16.843571	-56.690794

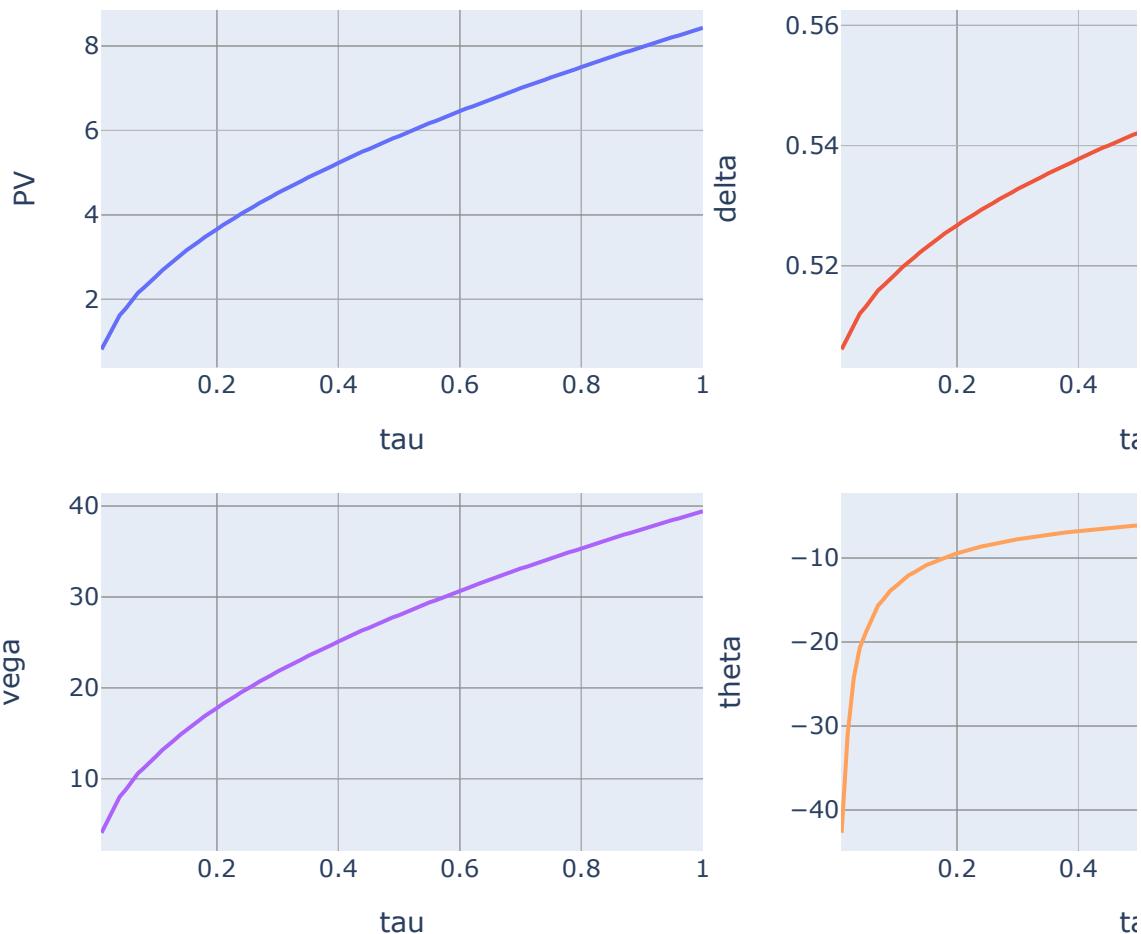
4.7 ...Vs τ , call

In [23]:

```

1 changing_param_name='tau'
2 changing_param_bound=(0.01,1)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)

```



Out[23]:

	PV	delta	gamma	vega	theta	rho
tau						
0.01	0.815061	0.506072	0.211055	4.047651	-42.708901	0.545652
0.02	1.094886	0.508140	0.152341	5.425880	-30.965425	0.953490
0.03	1.384325	0.510270	0.118974	6.845090	-24.291133	1.496038

	PV	delta	gamma	vega	theta	rho
tau						
0.04	1.624247	0.512029	0.100898	8.016594	-20.675379	2.037419
0.05	1.783783	0.513195	0.091699	8.793126	-18.835205	2.442785
...
0.96	8.247141	0.558368	0.020173	38.632855	-4.510499	45.632447
0.97	8.296411	0.558698	0.020056	38.848354	-4.486934	46.138221
0.98	8.345425	0.559027	0.019941	39.062557	-4.463762	46.643703
0.99	8.382021	0.559272	0.019856	39.222372	-4.446633	47.022623

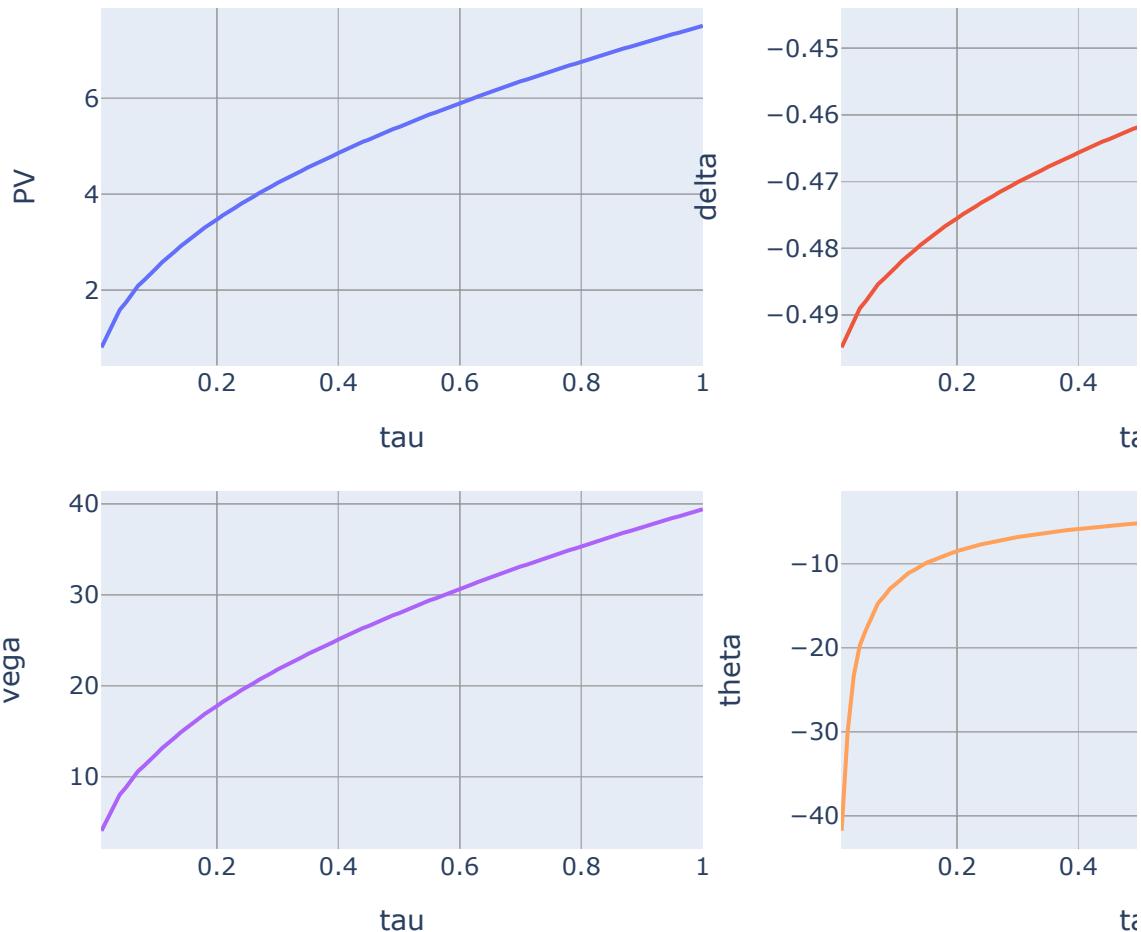
4.8 ...Vs τ , put

In [24]:

```

1 changing_param_name='tau'
2 changing_param_bound=(0.01,1)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)

```



Out[24]:

	PV	delta	gamma	vega	theta	rho
tau						
0.01	0.805608	-0.494855	0.211454	4.047762	-41.787896	-0.400291
0.02	1.077732	-0.492788	0.152632	5.426092	-30.022808	-0.763107
0.03	1.356782	-0.490691	0.119233	6.845317	-23.342264	-1.257515

	PV	delta	gamma	vega	theta	rho	
tau							
0.04	1.586310	-0.489007	0.101161	8.016612	-19.727368	-1.748479	
0.05	1.738059	-0.487905	0.091968	8.792889	-17.888266	-2.114098	
...
0.96	7.365121	-0.447704	0.020646	38.606954	-3.607944	-40.571062	
0.97	7.404501	-0.447420	0.020531	38.822345	-3.584699	-41.022453	
0.98	7.443630	-0.447139	0.020417	39.036446	-3.561844	-41.473633	
0.99	7.472814	-0.446928	0.020333	39.196187	-3.544953	-41.811880	

5 Barrier

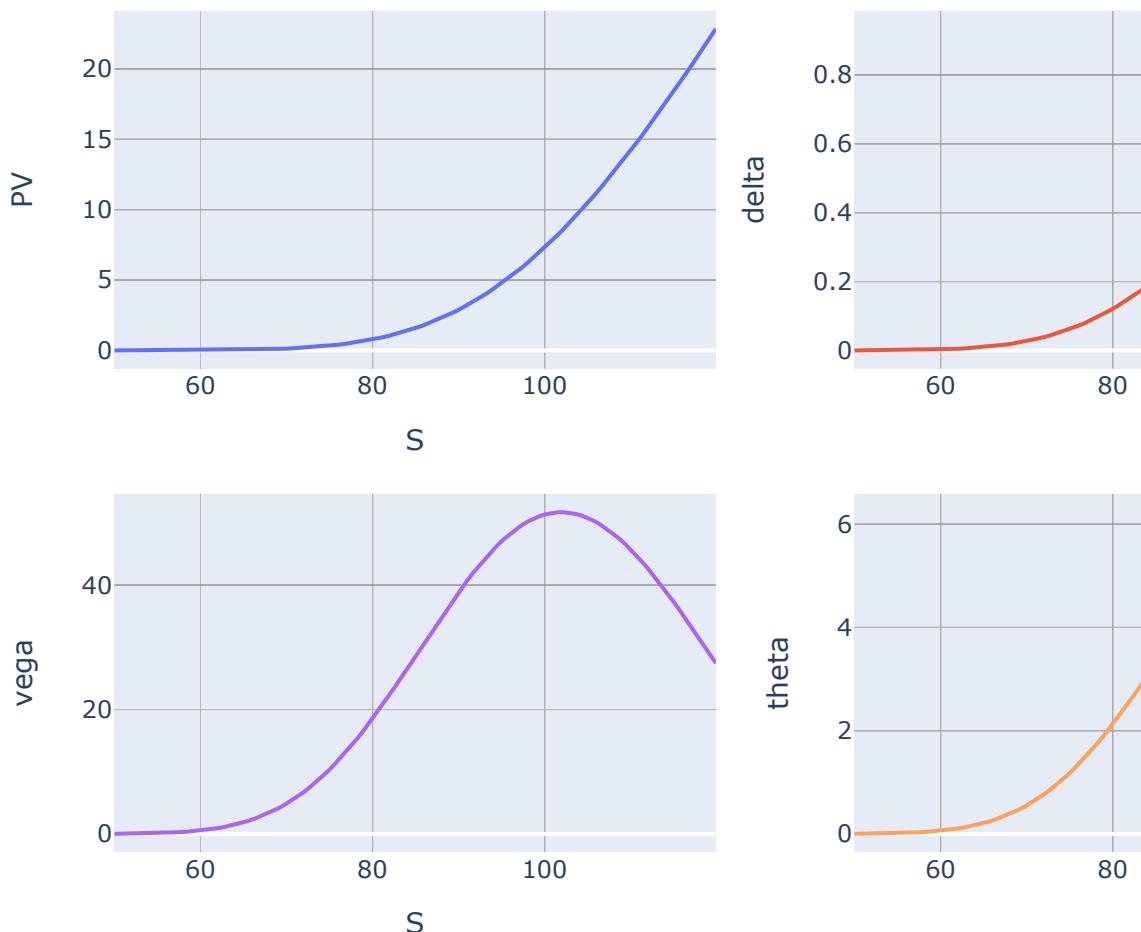
5.1 ...Vs S, UpIn call

In [25]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,119.9)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':120,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_upin_call,changing_param_name,changing_param_bound,other_params)

```



Out[25]:

S	PV	delta	gamma	vega	theta	rho
50.000000	0.000192	0.000089	0.000036	0.018869	0.002115	0.004151
50.706061	0.000265	0.000120	0.000049	0.025261	0.002833	0.005643

S	PV	delta	gamma	vega	theta	rho
51.412121	0.000363	0.000159	0.000063	0.033513	0.003759	0.007603
52.118182	0.000493	0.000210	0.000079	0.044070	0.004944	0.010154
52.824242	0.000663	0.000275	0.000102	0.057462	0.006449	0.013446
...
117.075758	20.251285	0.904173	0.011798	33.227705	4.482524	76.748744
117.781818	20.892571	0.912268	0.011136	31.799081	4.334631	77.541125
118.487879	21.539409	0.919901	0.010488	30.357672	4.184864	78.290491
119.193939	22.191475	0.927077	0.009841	28.908282	4.033758	78.997379
119.900000	22.848450	0.933808	0.009221	27.455588	3.881831	79.662441

100 rows × 6 columns

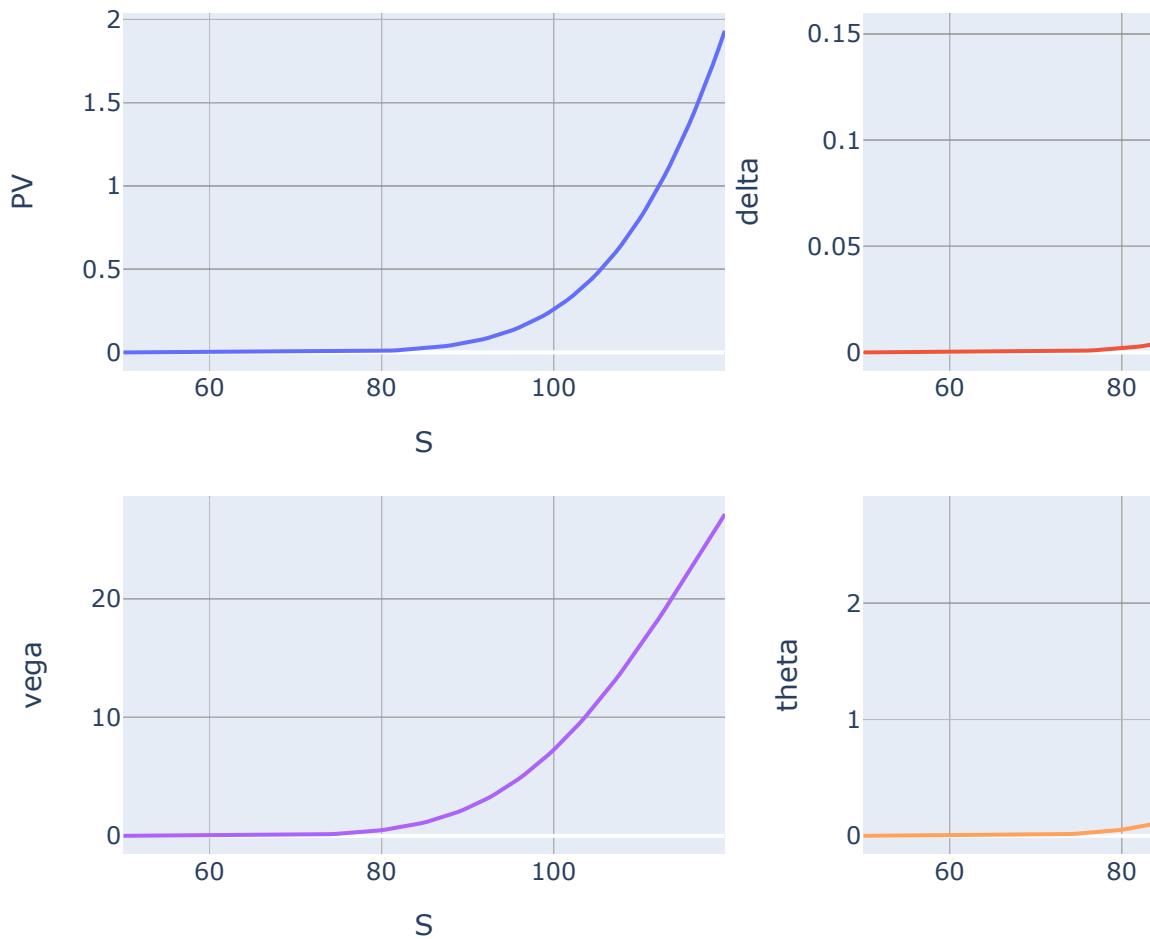
5.2 ...Vs S, UpIn put

In [26]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,119.9)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':120,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_upin_put,changing_param_name,changing_param_bound,other_params)

```



Out[26]:

S	PV	delta	gamma	vega	theta	rho
50.000000	1.785085e-07	1.014156e-07	-5.350880e-07	0.000027	0.000003	0.000003
50.706061	2.652457e-07	1.468714e-07	5.591032e-07	0.000039	0.000004	0.000004

S	PV	delta	gamma	vega	theta	rho
51.412121	3.901974e-07	2.105822e-07	4.102011e-08	0.000057	0.000006	0.000006
52.118182	5.684796e-07	2.991748e-07	-1.132728e-06	0.000080	0.000009	0.000008
52.824242	8.205045e-07	4.211079e-07	4.953140e-07	0.000113	0.000013	0.000011
...
117.075758	1.535885e+00	1.290250e-01	7.688072e-03	23.829008	2.458175	-13.979092
117.781818	1.628913e+00	1.345025e-01	7.827339e-03	24.651635	2.535641	-15.136812
118.487879	1.725842e+00	1.400766e-01	7.960566e-03	25.477110	2.612636	-16.366023
119.193939	1.826739e+00	1.457429e-01	8.088108e-03	26.304416	2.689019	-17.669445
119.900000	1.931669e+00	1.514970e-01	8.209611e-03	27.132542	2.764648	-19.049808

100 rows × 6 columns

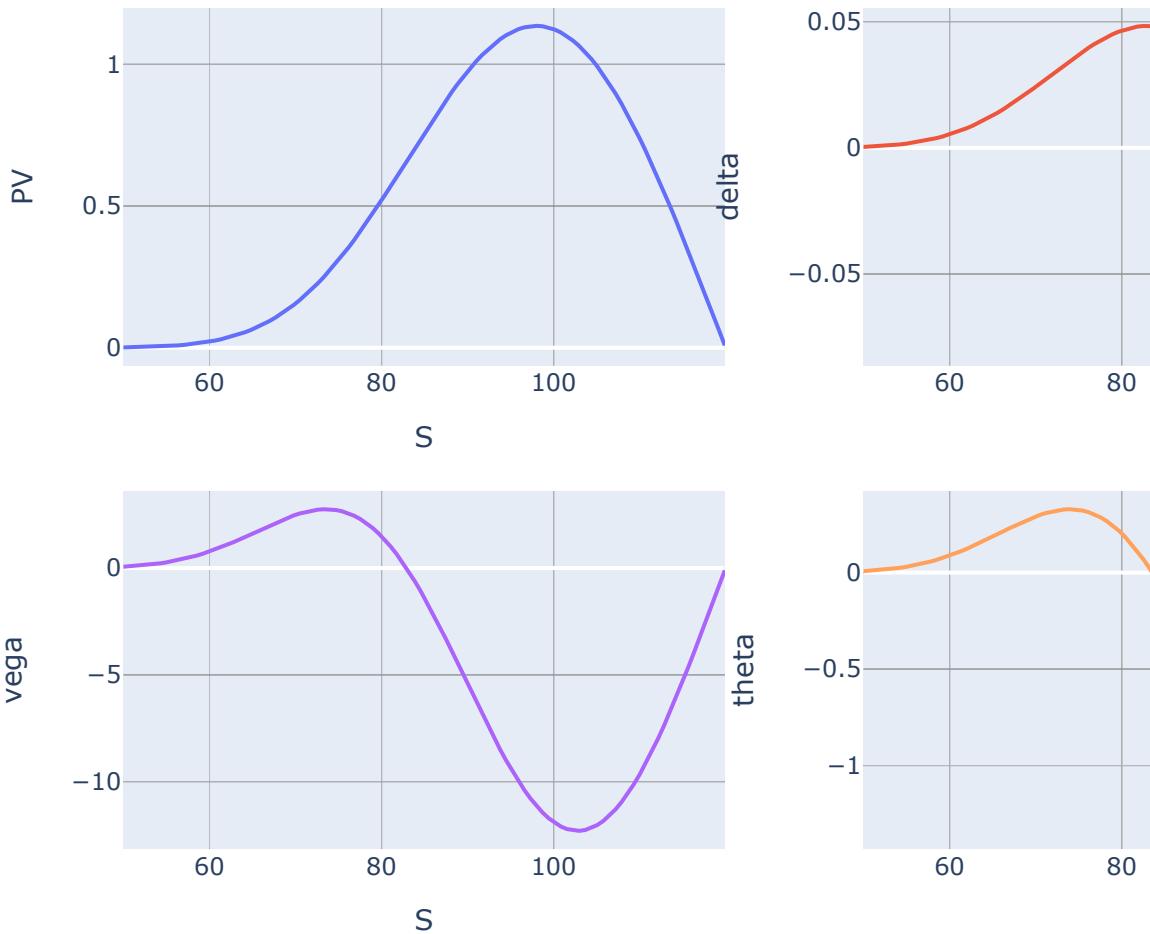
5.3 ...Vs S, UpOut call

In [27]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,119.9)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':120,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_upout_call,changing_param_name,changing_param_bound,other_params)

```



Out[27]:

S	PV	delta	gamma	vega	theta	rho
50.000000	0.000950	0.000368	0.000127	0.062891	0.007086	0.017558
50.706061	0.001243	0.000466	0.000153	0.079097	0.008917	0.022561

	PV	delta	gamma	vega	theta	rho
S						
51.412121	0.001613	0.000586	0.000188	0.098603	0.011122	0.028742
52.118182	0.002077	0.000731	0.000226	0.121870	0.013755	0.036314
52.824242	0.002652	0.000904	0.000269	0.149379	0.016870	0.045517
...
117.075758	0.230613	-0.078229	-0.000823	-3.151895	-0.351303	-0.532698
117.781818	0.175197	-0.078711	-0.000539	-2.396725	-0.267418	-0.430991
118.487879	0.119511	-0.078991	-0.000257	-1.635538	-0.182682	-0.311843
119.193939	0.063698	-0.079074	0.000028	-0.871558	-0.097453	-0.175664
119.900000	0.007895	-0.078963	0.000293	-0.107943	-0.012083	-0.022937

100 rows × 6 columns

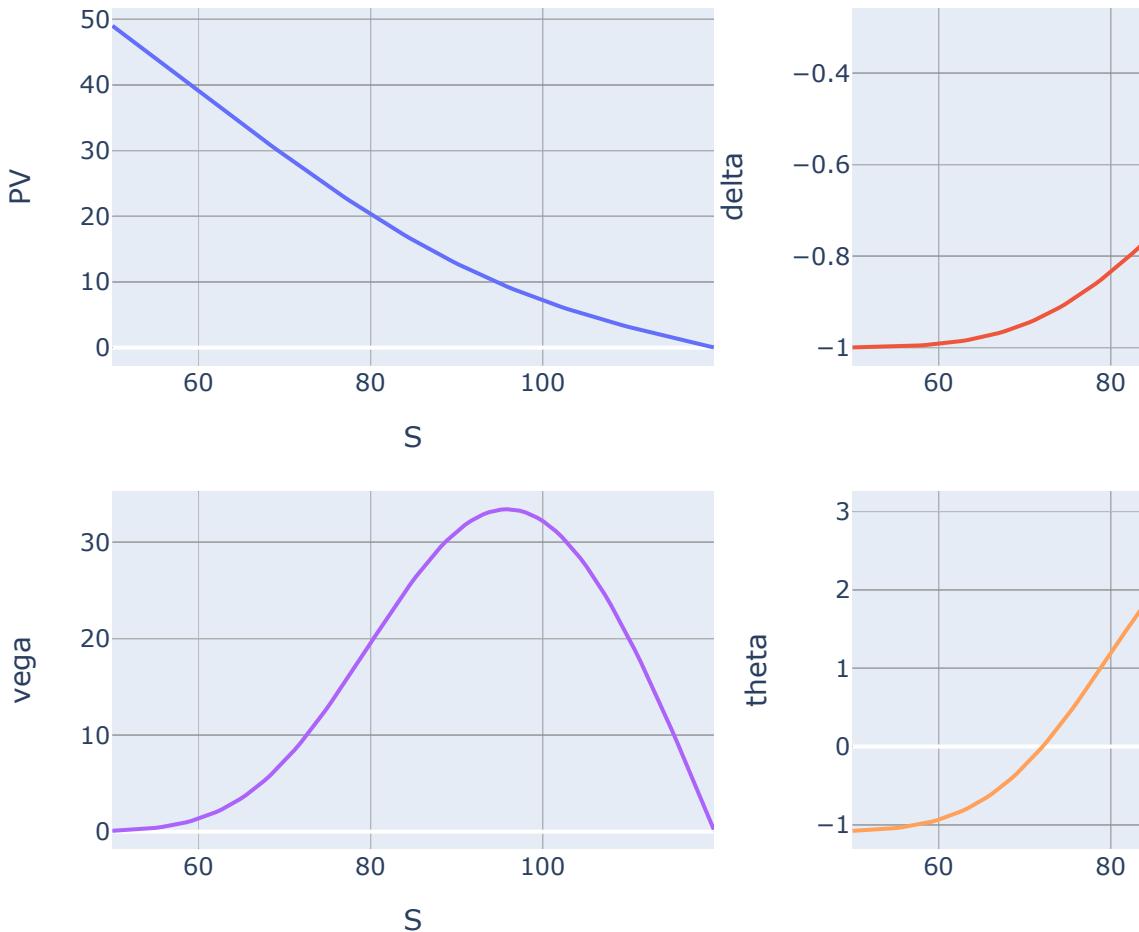
5.4 ...Vs S, UpOut put

In [28]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,119.9)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':120,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_upout_put,changing_param_name,changing_param_bound,other_params)

```



Out[28]:

S	PV	delta	gamma	vega	theta	rho
50.000000	49.006125	-0.999543	0.000162	0.081733	-1.075788	-98.983278
50.706061	48.300431	-0.999414	0.000204	0.104319	-1.073241	-98.976784

S	PV	delta	gamma	vega	theta	rho
51.412121	47.594838	-0.999255	0.000249	0.132059	-1.070111	-98.968645
52.118182	46.889370	-0.999059	0.000307	0.165860	-1.066296	-98.958523
52.824242	46.184055	-0.998821	0.000371	0.206728	-1.061680	-98.946031
...
117.075758	0.875239	-0.303082	0.003283	6.246803	0.588060	-8.809846
117.781818	0.662020	-0.300945	0.002772	4.750722	0.446586	-6.758038
118.487879	0.450183	-0.299167	0.002268	3.245024	0.304560	-4.660313
119.193939	0.239477	-0.297739	0.001778	1.732308	0.162300	-2.513824
119.900000	0.029659	-0.296653	0.001303	0.215104	0.020114	-0.315672

100 rows × 6 columns

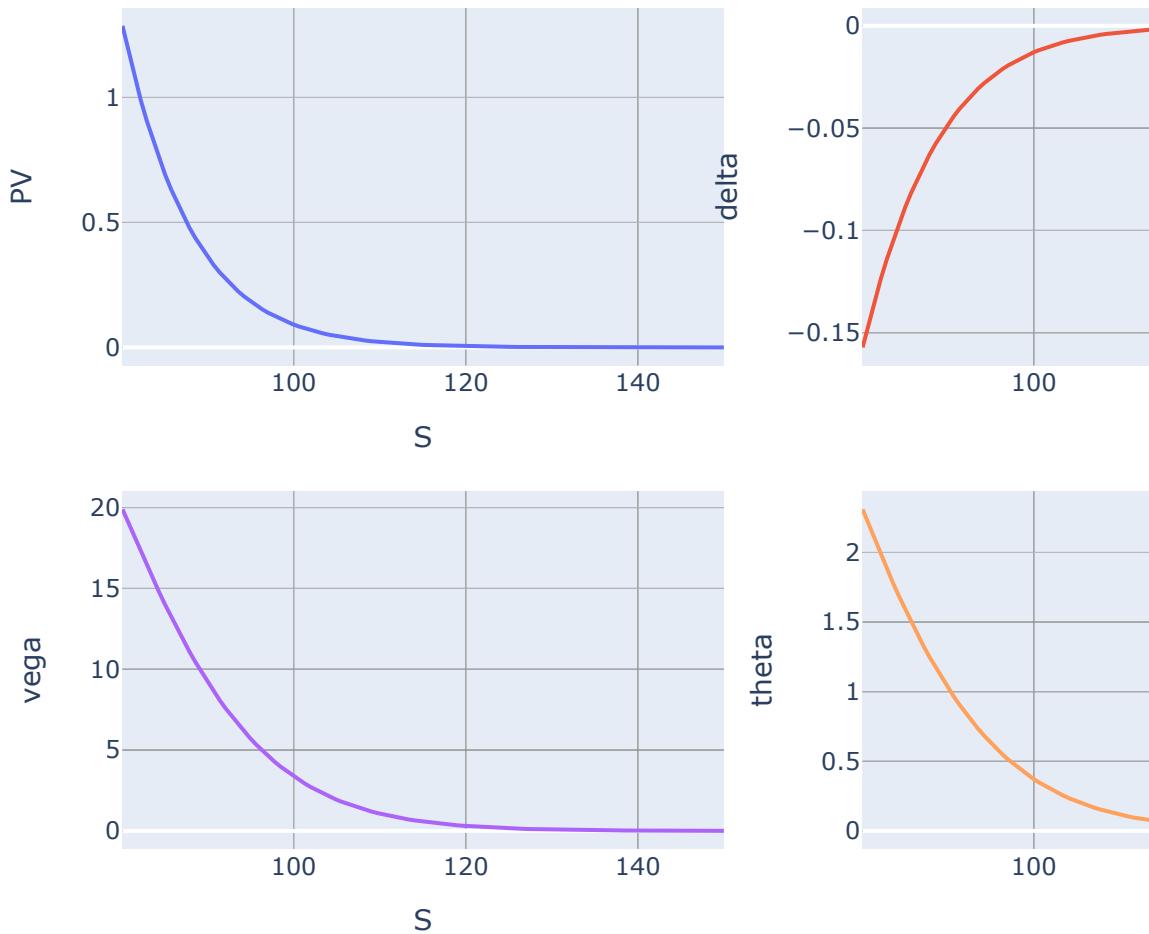
5.5 ...Vs S, DownIn call

In [29]:

```

1 changing_param_name='S'
2 changing_param_bound=(80.1,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':80,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_downin_call,changing_param_name,changing_param_bound,other_params)

```



Out[29]:

S	PV	delta	gamma	vega	theta	rho
80.100000	1.286438	-0.157194	1.751044e-02	19.895836	2.310388	11.867683
80.806061	1.179717	-0.145241	1.635740e-02	19.014849	2.199455	10.555127

S	PV	delta	gamma	vega	theta	rho
81.512121	1.081154	-0.134082	1.526157e-02	18.141762	2.090856	9.376459
82.218182	0.990200	-0.123676	1.422240e-02	17.280235	1.984855	8.319234
82.924242	0.906340	-0.113985	1.323777e-02	16.433475	1.881678	7.372025
...
147.175758	0.000076	-0.000012	1.855417e-06	0.007703	0.000838	-0.000631
147.881818	0.000068	-0.000011	1.131300e-06	0.006978	0.000759	-0.000574
148.587879	0.000061	-0.000009	2.110942e-06	0.006321	0.000688	-0.000523
149.293939	0.000055	-0.000008	2.503033e-06	0.005726	0.000623	-0.000476
150.000000	0.000049	-0.000008	2.713541e-07	0.005185	0.000564	-0.000433

100 rows × 6 columns

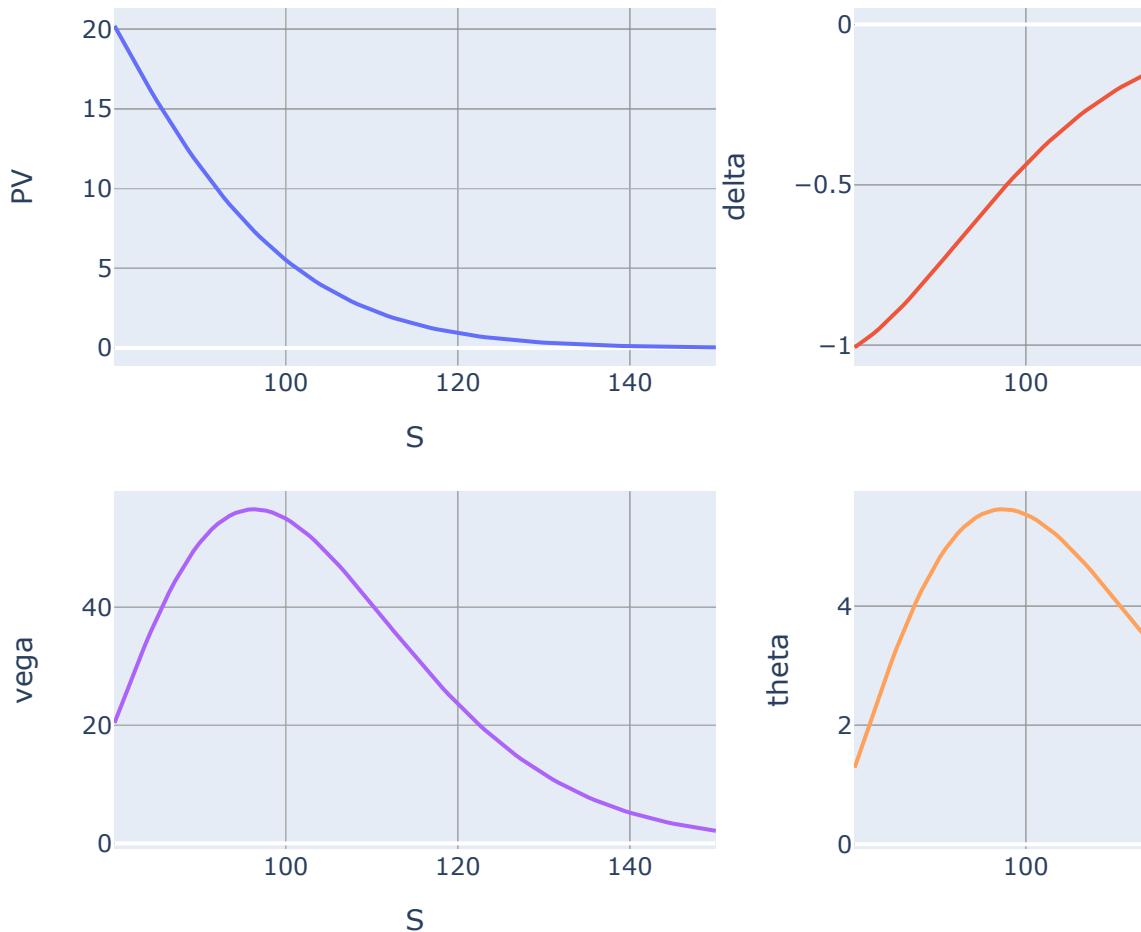
5.6 ...Vs S, DownIn put

In [30]:

```

1 changing_param_name='S'
2 changing_param_bound=(80.1,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':80,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_downin_put,changing_param_name,changing_param_bound,other_params)

```



Out[30]:

S	PV	delta	gamma	vega	theta	rho
80.100000	20.206446	-1.006986	0.016994	20.406057	1.284503	-86.846901
80.806061	19.499836	-0.994362	0.018752	23.115791	1.589259	-86.137708

S	PV	delta	gamma	vega	theta	rho
81.512121	18.802571	-0.980528	0.020423	25.797733	1.891983	-85.336060
82.218182	18.115482	-0.965546	0.021997	28.435370	2.190847	-84.443509
82.924242	17.439356	-0.949487	0.023475	31.013118	2.484120	-83.462242
...
147.175758	0.052831	-0.005881	0.000639	2.759689	0.292827	-0.877465
147.881818	0.048833	-0.005447	0.000592	2.587550	0.274631	-0.816587
148.587879	0.045131	-0.005044	0.000549	2.425376	0.257482	-0.759760
149.293939	0.041704	-0.004670	0.000511	2.272660	0.241328	-0.706732
150.000000	0.038531	-0.004322	0.000472	2.128911	0.226117	-0.657262

100 rows × 6 columns

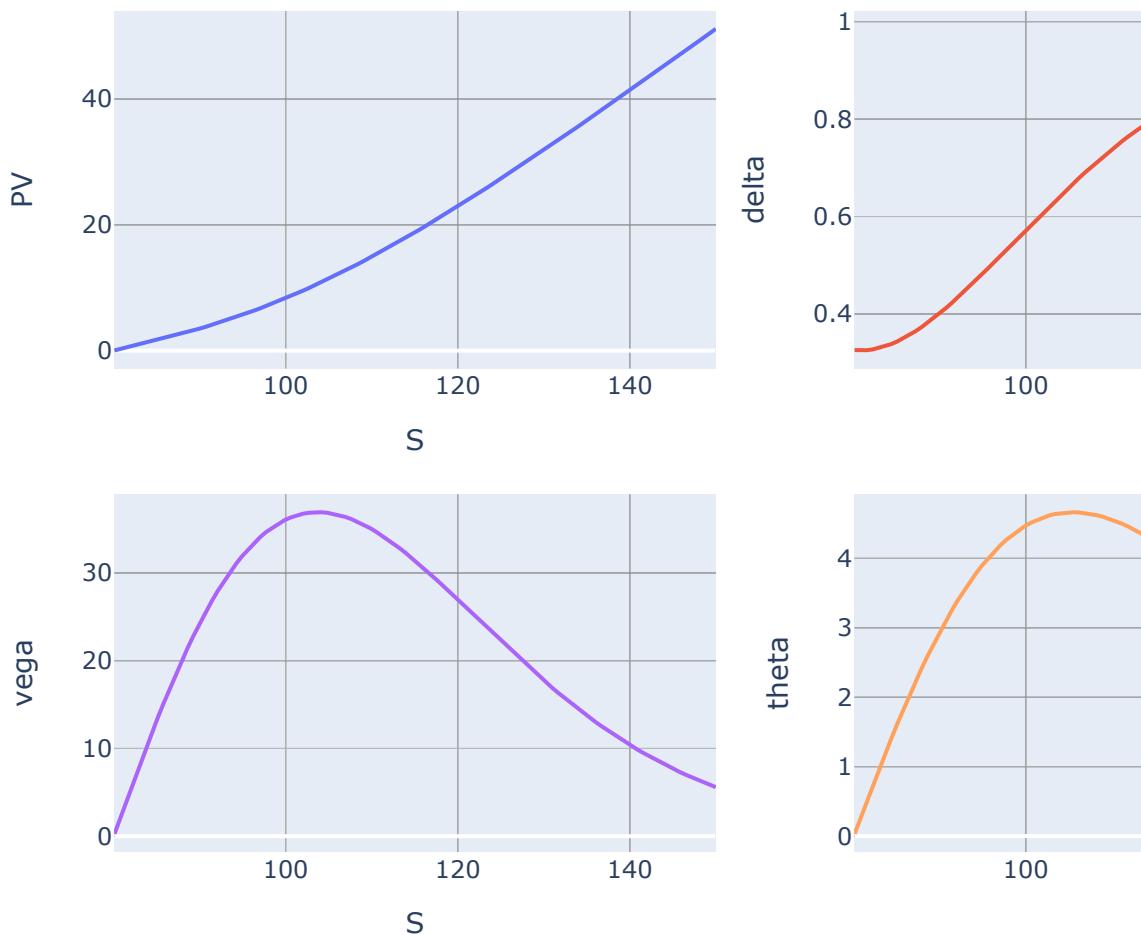
5.7 ...Vs S, DownOut call

In [31]:

```

1 changing_param_name='S'
2 changing_param_bound=(80.1,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':80,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_downout_call,changing_param_name,changing_param_bound,other_params)

```



Out[31]:

S	PV	delta	gamma	vega	theta	rho
80.100000	0.032595	0.325855	-0.001794	0.271097	0.033250	0.322999
80.806061	0.262361	0.325182	-0.000125	2.184275	0.267247	2.543095

S	PV	delta	gamma	vega	theta	rho
81.512121	0.492062	0.325661	0.001469	4.090415	0.499412	4.666339
82.218182	0.722492	0.327237	0.002984	5.982283	0.729080	6.704498
82.924242	0.954406	0.329856	0.004420	7.853079	0.955623	8.668182
...
147.175758	48.384777	0.981354	0.001550	6.709893	1.787862	96.045507
147.881818	49.078051	0.982418	0.001472	6.412916	1.757040	96.202619
148.587879	49.772056	0.983425	0.001386	6.126901	1.727335	96.352029
149.293939	50.466754	0.984378	0.001308	5.851596	1.698722	96.494068
150.000000	51.162105	0.985279	0.001246	5.586740	1.671178	96.629057

100 rows × 6 columns

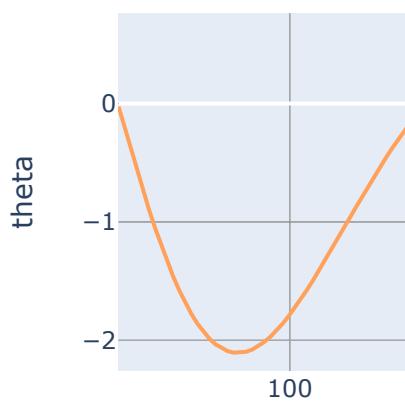
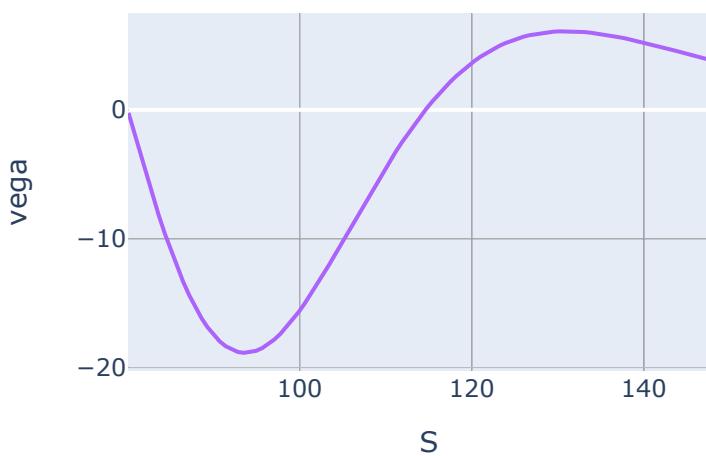
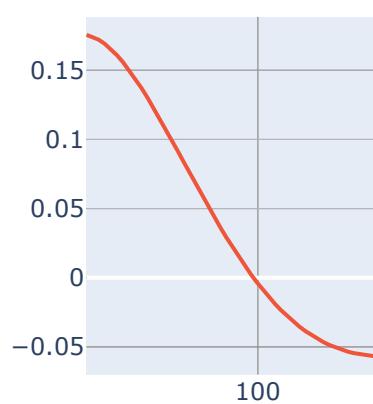
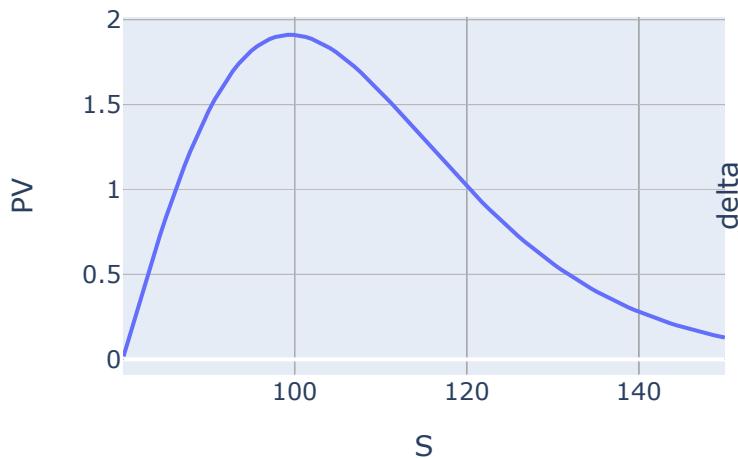
5.8 ...Vs S, DownOut put

In [32]:

```

1 changing_param_name='S'
2 changing_param_bound=(80.1,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'X':80,
7     'r':0.01,
8     'q':0,
9     'tau':1,
10    'sigma':0.2
11 }
12 option_plot(barrier_downout_put,changing_param_name,changing_param_bound,other_params)

```



Out[32]:

S	PV	delta	gamma	vega	theta	rho
80.100000	0.017571	0.175646	-0.001277	-0.239123	-0.025852	0.032599
80.806061	0.141165	0.174303	-0.002517	-1.916667	-0.207543	0.230946

	PV	delta	gamma	vega	theta	rho
S						
81.512121	0.263507	0.172107	-0.003696	-3.565555	-0.386701	0.373875
82.218182	0.384011	0.169107	-0.004792	-5.172852	-0.561899	0.462257
82.924242	0.502130	0.165358	-0.005815	-6.726564	-0.731805	0.497466
...
147.175758	0.161248	-0.012777	0.000911	3.957907	0.410887	-2.082642
147.881818	0.152451	-0.012146	0.000878	3.832345	0.398182	-1.986352
148.587879	0.144091	-0.011540	0.000840	3.707846	0.385554	-1.893717
149.293939	0.136149	-0.010961	0.000803	3.584661	0.373031	-1.804659
150.000000	0.128607	-0.010406	0.000772	3.463015	0.360638	-1.719098

100 rows × 6 columns

6 Asian

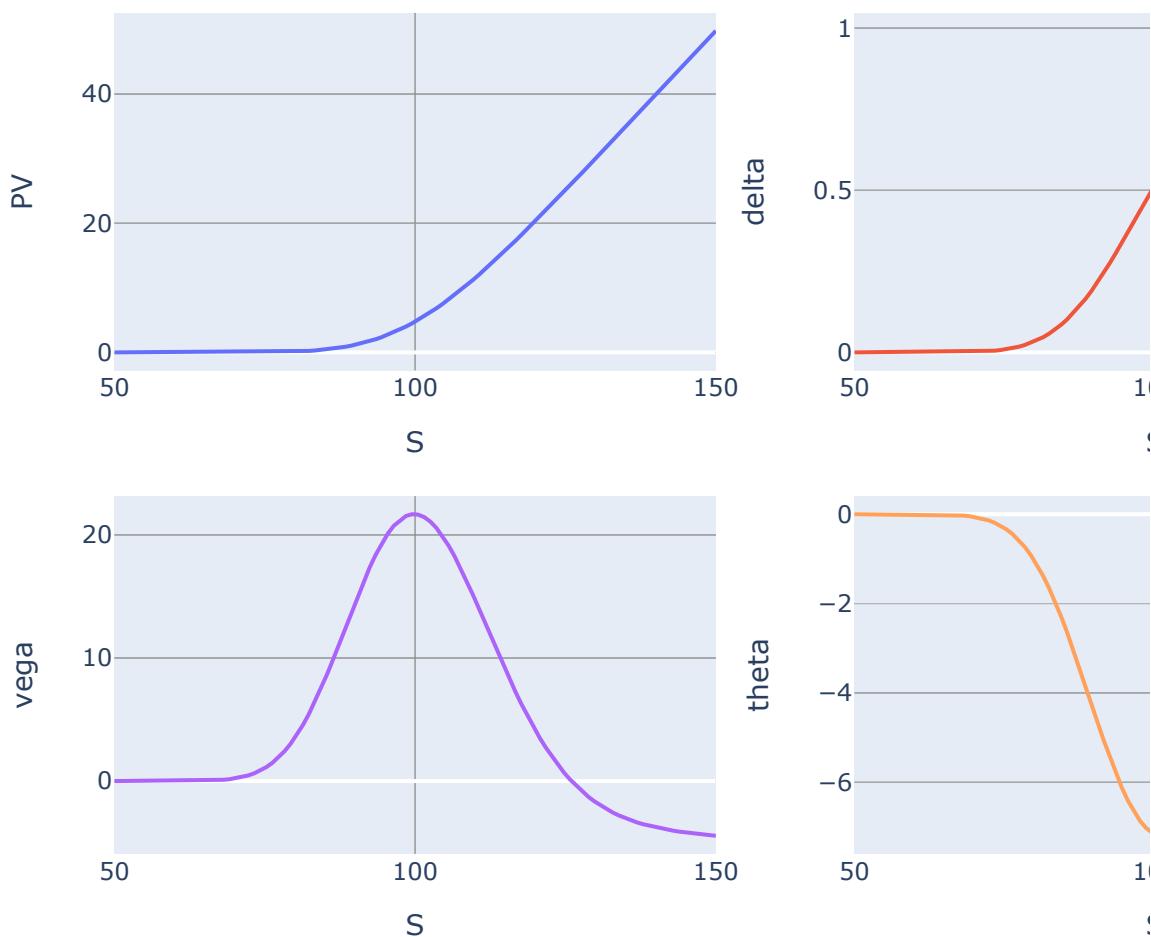
6.1 ...Vs S, Geometric call

In [33]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(asian_geometric_call,changing_param_name,changing_param_bound,other_params)

```



Out[33]:

S	PV	delta	gamma	vega	theta	rho
50.000000	3.149653e-09	3.326922e-09	3.361398e-09	5.800775e-07	-1.697019e-07	8.004411e-07
51.010101	8.934602e-09	9.011064e-09	8.677070e-09	1.558130e-06	-4.560672e-07	2.209503e-07

S	PV	delta	gamma	vega	theta	rho
52.020202	2.418190e-08	2.329366e-08	2.138157e-08	3.991997e-06	-1.169089e-06	5.818394e-01
53.030303	6.259536e-08	5.760339e-08	5.041113e-08	9.778347e-06	-2.865258e-06	1.465148e-01
54.040404	1.553083e-07	1.365721e-07	1.139660e-07	2.295002e-05	-6.728696e-06	3.535816e-01
...
145.959596	4.578847e+01	9.914793e-01	1.059034e-04	-4.241558e+00	-1.034398e+00	2.658751e+0
146.969697	4.679001e+01	9.915763e-01	8.668033e-05	-4.299162e+00	-1.026863e+00	2.609397e+0
147.979798	4.779165e+01	9.916556e-01	7.080416e-05	-4.352371e+00	-1.020543e+00	2.559912e+0
148.989899	4.879335e+01	9.917203e-01	5.772269e-05	-4.401850e+00	-1.015256e+00	2.510320e+0
150.000000	4.979512e+01	9.917730e-01	4.696833e-05	-4.448173e+00	-1.010844e+00	2.460638e+0

100 rows × 6 columns

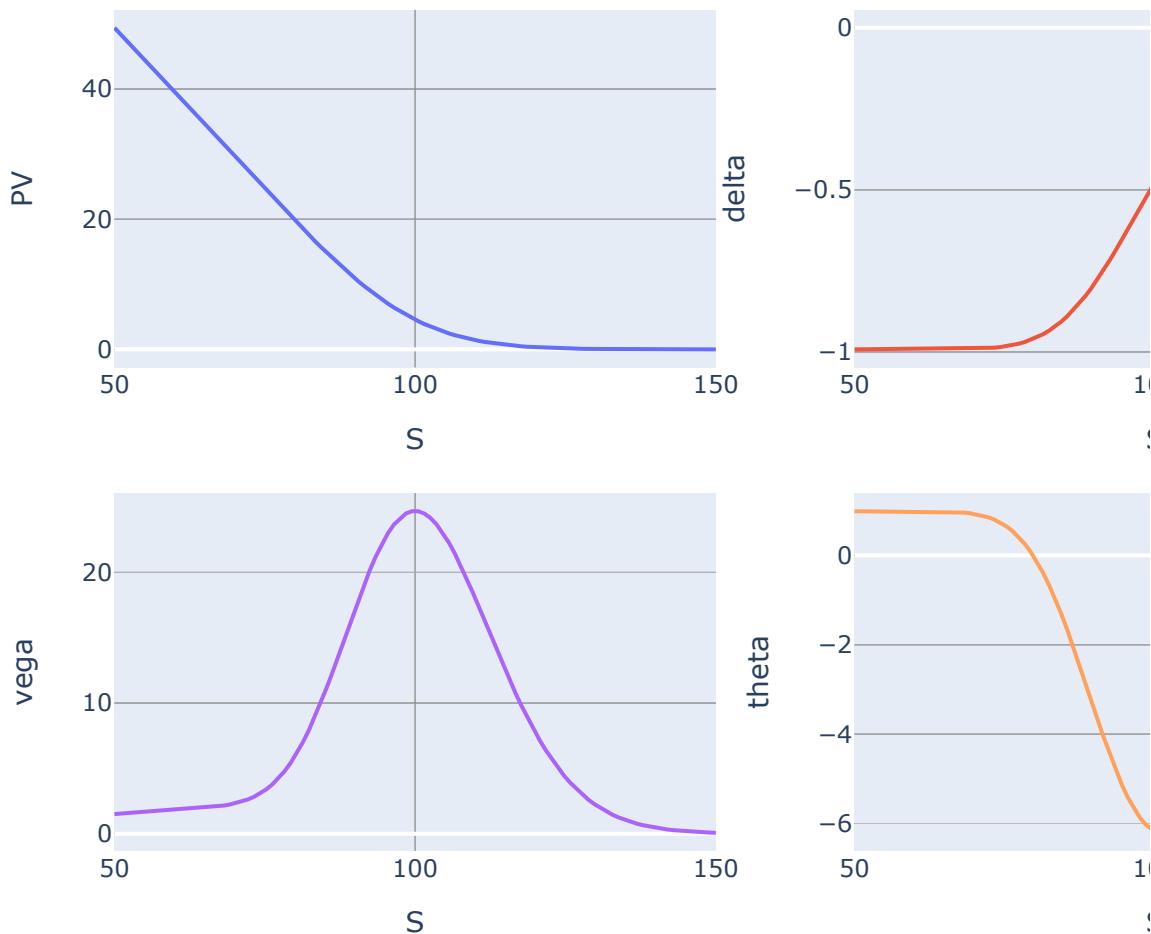
6.2 ...Vs S, Geometric put

In [34]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(asian_geometric_put,changing_param_name,changing_param_bound,other_params)

```



Out[34]:

S	PV	delta	gamma	vega	theta	rho
50.000000	49.405286	-0.991994	3.361398e-09	1.507589	0.990050	-74.211311
51.010101	48.403272	-0.991994	8.677070e-09	1.538046	0.990049	-73.710429
52.020202	47.401257	-0.991994	2.138157e-08	1.568505	0.990049	-73.209546

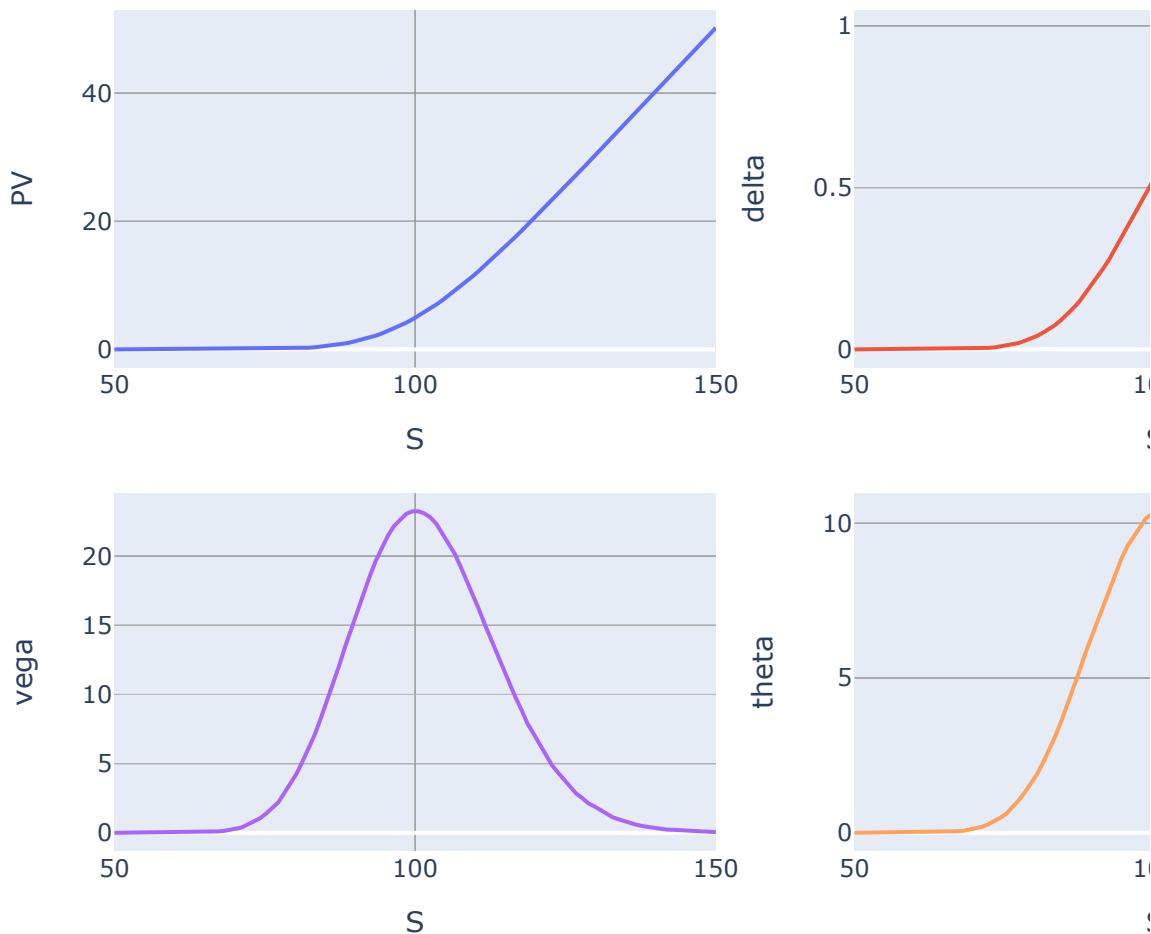
S	PV	delta	gamma	vega	theta	rho
53.030303	46.399243	-0.991994	5.041113e-08	1.598967	0.990047	-72.708663
54.040404	45.397229	-0.991994	1.139660e-07	1.629436	0.990043	-72.207779
...
145.959596	0.002416	-0.000515	1.059034e-04	0.159381	-0.044348	-0.039982
146.969697	0.001947	-0.000418	8.668033e-05	0.132234	-0.036813	-0.032646
147.979798	0.001566	-0.000338	7.080416e-05	0.109481	-0.030493	-0.026609
148.989899	0.001258	-0.000274	5.772269e-05	0.090458	-0.025206	-0.021651
150.000000	0.001009	-0.000221	4.696833e-05	0.074591	-0.020794	-0.017587

100 rows × 6 columns

6.3 ...Vs S, Arithmetic call

In [35]:

```
1 changing_param_name='S'  
2 changing_param_bound=(50,150)  
3 other_params={  
4     'S':100,  
5     'K':100,  
6     'r':0.01,  
7     'q':0,  
8     'tau':1,  
9     'sigma':0.2  
10    }  
11 option_plot(asian_arithmetic_call,changing_param_name,changing_param_bound,other_params)
```



Out[35]:

	PV	delta	gamma	vega	theta	rho
S						
50.000000	3.149653e-09	3.326933e-09	3.361443e-09	5.801049e-07	4.701107e-07	8.004447e-08
51.010101	8.934602e-09	9.011101e-09	8.677492e-09	1.558197e-06	1.208688e-06	2.209514e-07

S	PV	delta	gamma	vega	theta	rho
52.020202	2.418190e-08	2.328417e-08	3.525487e-07	3.992143e-06	2.971561e-06	5.818235e-07
53.030303	6.259536e-08	5.761359e-08	-5.338580e-08	9.778628e-06	7.001084e-06	1.465127e-06
54.040404	1.553083e-07	1.365798e-07	4.296271e-08	2.295061e-05	1.583974e-05	3.535815e-06
...
145.959596	4.618896e+01	9.937675e-01	1.101341e-04	1.019216e-01	4.038530e+00	2.268892e+01
146.969697	4.719280e+01	9.938643e-01	1.371347e-04	7.506023e-02	4.072494e+00	2.211212e+01
147.979798	4.819675e+01	9.939770e-01	3.481659e-05	4.252253e-02	4.101276e+00	2.153630e+01
148.989899	4.920079e+01	9.940084e-01	3.410605e-05	3.361285e-02	4.131634e+00	2.095502e+01
150.000000	5.020481e+01	9.939616e-01	1.350031e-05	4.844707e-02	4.170231e+00	2.036792e+01

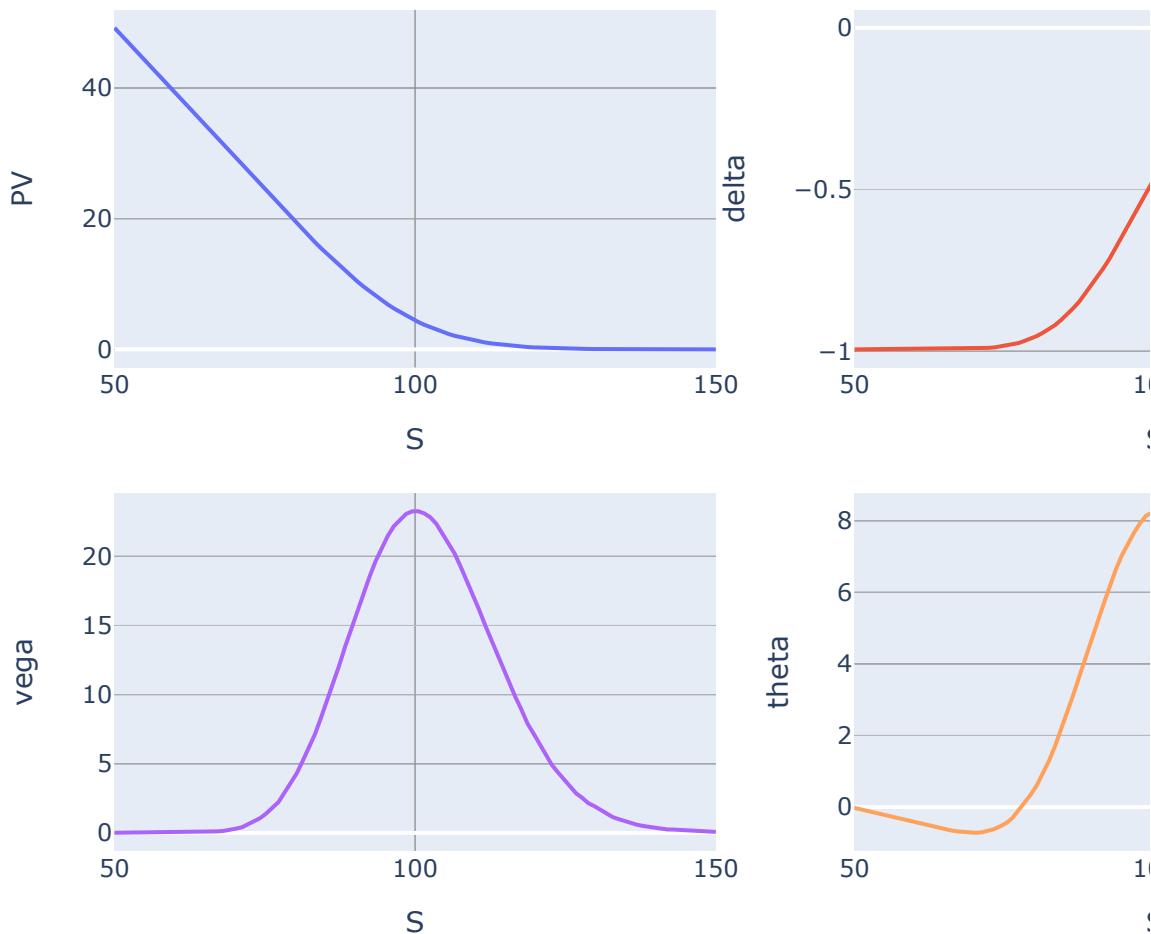
6.4 ...Vs S, Arithmetic put

In [36]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(asian_arithmetic_put,changing_param_name,changing_param_bound,other_params)

```



Out[36]:

S	PV	delta	gamma	vega	theta	rho
50.000000	49.214368	-0.994184	-0.000009	0.008044	-0.025308	-78.173398
51.010101	48.210141	-0.994184	0.000023	0.008207	-0.066950	-77.589675
52.020202	47.205915	-0.994184	-0.000007	0.008372	-0.108591	-77.005952

	PV	delta	gamma	vega	theta	rho
S						
53.030303	46.201689	-0.994184	-0.000001	0.008541	-0.150230	-76.422229
54.040404	45.197462	-0.994184	0.000023	0.008716	-0.191865	-75.838504
...
145.959596	0.001815	-0.000417	0.000108	0.125402	0.057137	-0.030836
146.969697	0.001433	-0.000320	0.000087	0.098703	0.049458	-0.023913
147.979798	0.001155	-0.000207	0.000073	0.066328	0.036597	-0.016012
148.989899	0.000971	-0.000176	0.000056	0.057580	0.025312	-0.013572
150.000000	0.000766	-0.000223	0.000045	0.072577	0.022266	-0.016947

100 rows × 6 columns

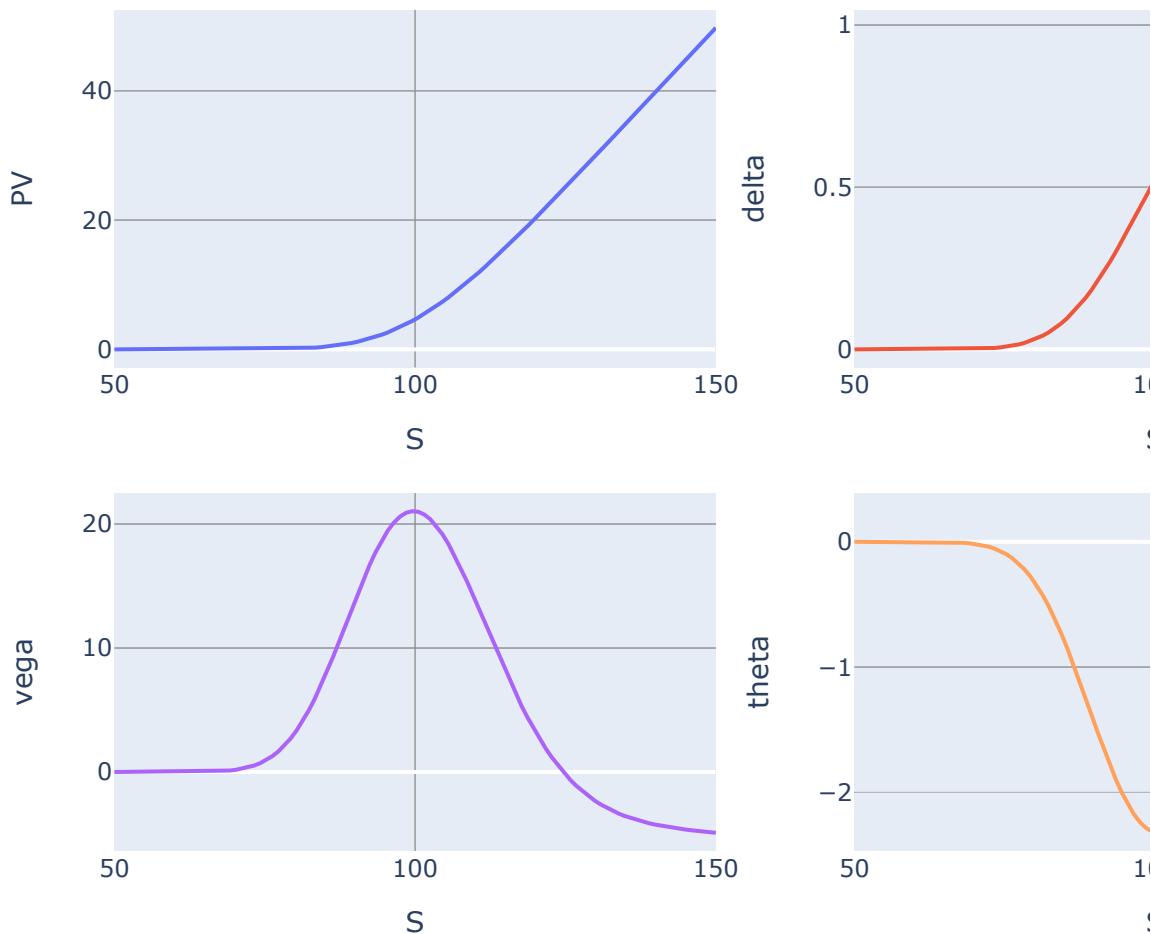
6.5 ...Vs S, Continuous Geometric call

In [37]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(asian_continuous_geometric_call,changing_param_name,changing_param_bound,other_params)

```



◀ ▶ ⌂

Out[37]:

S	PV	delta	gamma	vega	theta	rho
50.000000	1.358844e-09	1.496060e-09	1.578287e-09	2.605543e-07	-2.641586e-08	3.604267e-08
51.010101	4.028282e-09	4.234244e-09	4.257121e-09	7.312776e-07	-7.416743e-08	1.039663e-07

S	PV	delta	gamma	vega	theta	rho
52.020202	1.137004e-08	1.141352e-08	1.093823e-08	1.953540e-06	-1.982089e-07	2.854967e-07
53.030303	3.063235e-08	2.937286e-08	2.683685e-08	4.979474e-06	-5.054293e-07	7.481936e-07
54.040404	7.895456e-08	7.233579e-08	6.301635e-08	1.213842e-05	-1.232597e-06	1.875573e-06
...
145.959596	4.574517e+01	9.912964e-01	8.666254e-05	-4.699889e+00	2.039944e-01	2.659945e+01
146.969697	4.674652e+01	9.913754e-01	7.034633e-05	-4.755439e+00	2.144983e-01	2.610456e+01
147.979798	4.774794e+01	9.914395e-01	5.698212e-05	-4.807248e+00	2.246391e-01	2.560857e+01
148.989899	4.874942e+01	9.914914e-01	4.606231e-05	-4.855907e+00	2.344739e-01	2.511168e+01
150.000000	4.975095e+01	9.915333e-01	3.716080e-05	-4.901925e+00	2.440521e-01	2.461405e+01

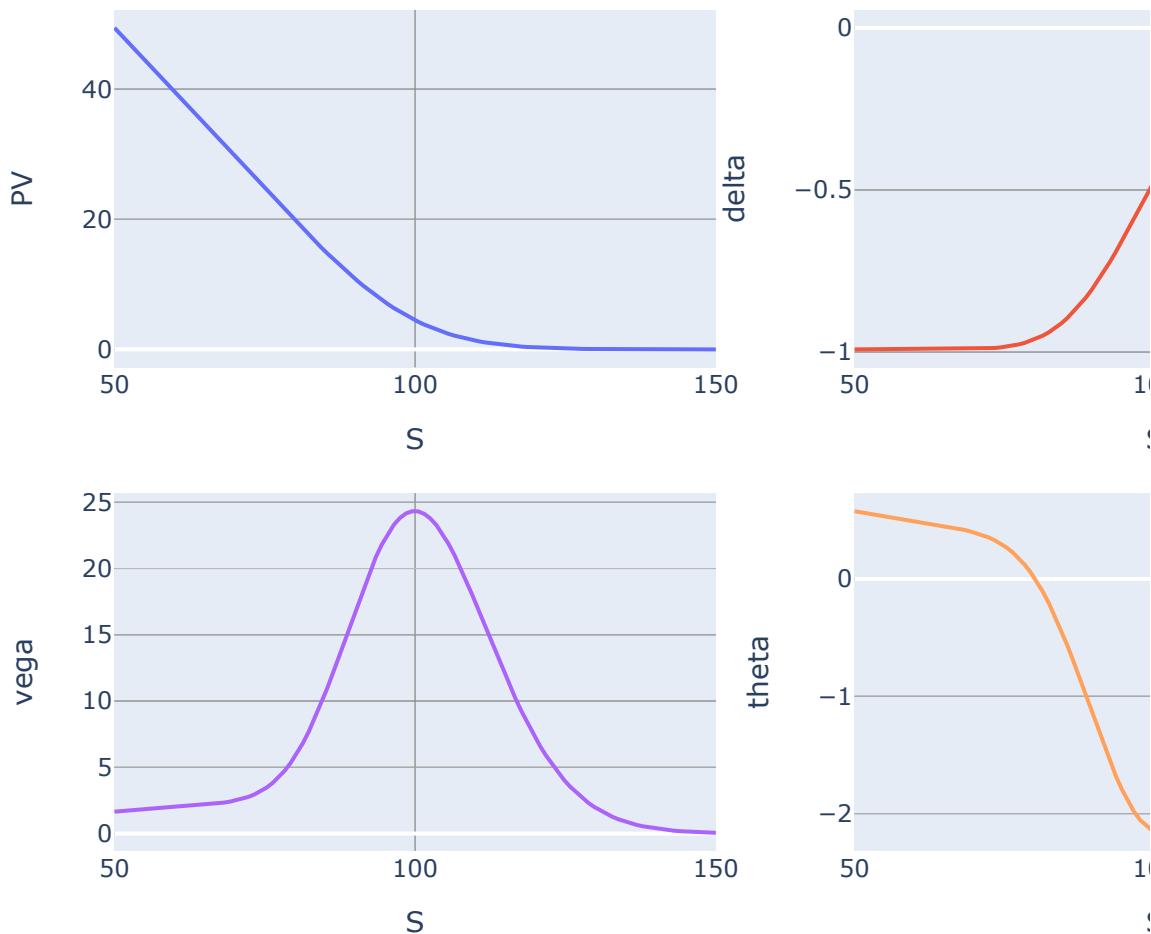
6.6 ...Vs S, Continuous Geometric put

In [38]:

```

1 changing_param_name='S'
2 changing_param_bound=(50,150)
3 other_params={
4     'S':100,
5     'K':100,
6     'r':0.01,
7     'q':0,
8     'tau':1,
9     'sigma':0.2
10    }
11 option_plot(asian_continuous_geometric_put,changing_param_name,changing_param_bound,oth

```



Out[38]:

S	PV	delta	gamma	vega	theta	rho
50.000000	49.419919	-0.991701	1.578287e-09	1.652836	0.576841	-74.212451
51.010101	48.418200	-0.991701	4.257121e-09	1.686227	0.568493	-73.711592
52.020202	47.416482	-0.991701	1.093823e-08	1.719619	0.560145	-73.210732

S	PV	delta	gamma	vega	theta	rho
53.030303	46.414763	-0.991701	2.683685e-08	1.753012	0.551797	-72.709873
54.040404	45.413045	-0.991701	6.301635e-08	1.786410	0.543449	-72.209012
...
145.959596	0.001829	-0.000405	8.666254e-05	0.125055	-0.012192	-0.031377
146.969697	0.001461	-0.000326	7.034633e-05	0.102896	-0.010035	-0.025407
147.979798	0.001165	-0.000262	5.698212e-05	0.084478	-0.008242	-0.020534
148.989899	0.000928	-0.000210	4.606231e-05	0.069209	-0.006755	-0.016566
150.000000	0.000738	-0.000168	3.716080e-05	0.056581	-0.005525	-0.013341

100 rows × 6 columns

In []:

1	
---	--