1 Functions

1.1 Vanilla Euro

In [1]:

```
# ========================================================================
# This file is especially for visualization of option price/Greeks w.r.t its parameters
# Mainly based on Quantlib
# ========================================================================
import QuantLib as ql

def vanilla_euro_call(S,K,r,q,tau,sigma,**kwargs):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.VanillaOption(payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticEuropeanEngine(bsm_process))
    #setup is finished
    # return option.NPV(),option.delta(),option.gamma(),option.vega(),option.theta(),op
    return {
        'PV':option.NPV(),
        'delta':option.delta(),
        'gamma':option.gamma(),
        'vega':option.vega(),
        'theta':option.theta(),
        'rho':option.rho(),
        }

def vanilla_euro_put(S,K,r,q,tau,sigma,**kwargs):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
```

```
    exercise = ql.EuropeanExercise(maturity)
    option=ql.VanillaOption(payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticEuropeanEngine(bsm_process))
    #setup is finished
    # return option.NPV(),option.delta(),option.gamma(),option.vega(),option.theta(),op
    return {
        'PV':option.NPV(),
        'delta':option.delta(),
        'gamma':option.gamma(),
        'vega':option.vega(),
        'theta':option.theta(),
        'rho':option.rho(),
        }
```

1.2 Vanilla Ame

In [2]:

```
def _vanilla_ame_call(S,K,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.AmericanExercise(start,maturity)
    option=ql.VanillaOption(payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    steps=tau_day*2
    option.setPricingEngine(ql.BinomialVanillaEngine(bsm_process,'crr',steps))
    #setup is finished
    return option

def vanilla_ame_call(S,K,r,q,tau,sigma,**kwargs):
    option=_vanilla_ame_call(S,K,r,q,tau,sigma)
    epsilon=1e-8
    option_sigam_upper=_vanilla_ame_call(S,K,r,q,tau,sigma+epsilon)
    option_sigam_lower=_vanilla_ame_call(S,K,r,q,tau,sigma-epsilon)
    option_r_upper=_vanilla_ame_call(S,K,r+epsilon,q,tau,sigma)
    option_r_lower=_vanilla_ame_call(S,K,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':option.delta(),
        'gamma':option.gamma(),
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':option.theta(),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _vanilla_ame_put(S,K,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
```

```
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.AmericanExercise(start,maturity)
    option=ql.VanillaOption(payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    steps=tau_day*2
    option.setPricingEngine(ql.BinomialVanillaEngine(bsm_process,'crr',steps))
    #setup is finished
    return option
def vanilla_ame_put(S,K,r,q,tau,sigma,**kwargs):
    option=_vanilla_ame_put(S,K,r,q,tau,sigma)
    epsilon=1e-8
    option_sigam_upper=_vanilla_ame_put(S,K,r,q,tau,sigma+epsilon)
    option_sigam_lower=_vanilla_ame_put(S,K,r,q,tau,sigma-epsilon)
    option_r_upper=_vanilla_ame_put(S,K,r+epsilon,q,tau,sigma)
    option_r_lower=_vanilla_ame_put(S,K,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':option.delta(),
        'gamma':option.gamma(),
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':option.theta(),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }
```

1.3 Barrier

In [3]:

```
def _barrier_upin_call(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.UpIn, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_upin_call(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_upin_call(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_upin_call(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_upin_call(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_upin_call(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_upin_call(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_upin_call(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_upin_call(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_upin_call(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_upin_call(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_upin_put(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
```

```
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.UpIn, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_upin_put(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_upin_put(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_upin_put(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_upin_put(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_upin_put(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_upin_put(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_upin_put(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_upin_put(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_upin_put(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_upin_put(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_upout_call(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.UpOut, X, 0, payoff, exercise)
    # specify pricing method
```

```
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_upout_call(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_upout_call(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_upout_call(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_upout_call(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_upout_call(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_upout_call(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_upout_call(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_upout_call(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_upout_call(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_upout_call(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_upout_put(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.UpOut, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
```

```
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_upout_put(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_upout_put(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_upout_put(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_upout_put(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_upout_put(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_upout_put(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_upout_put(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_upout_put(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_upout_put(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_upout_put(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_downin_call(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.DownIn, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
```

```
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_downin_call(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_downin_call(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_downin_call(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_downin_call(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_downin_call(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_downin_call(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_downin_call(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_downin_call(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_downin_call(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_downin_call(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_downin_put(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.DownIn, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_downin_put(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_downin_put(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_downin_put(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_downin_put(S-epsilon,K,X,r,q,tau,sigma)
```

```
    option_sigam_upper=_barrier_downin_put(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_downin_put(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_downin_put(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_downin_put(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_downin_put(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_downin_put(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_downout_call(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.DownOut, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_downout_call(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_downout_call(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_downout_call(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_downout_call(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_downout_call(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_downout_call(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_downout_call(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_downout_call(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_downout_call(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_downout_call(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
```

```
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _barrier_downout_put(S,K,X,r,q,tau,sigma):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option=ql.BarrierOption(ql.Barrier.DownOut, X, 0, payoff, exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticBarrierEngine(bsm_process))
    return option

def barrier_downout_put(S,K,X,r,q,tau,sigma,**kwargs):
    option=_barrier_downout_put(S,K,X,r,q,tau,sigma)
    epsilon=1e-4
    option_S_upper=_barrier_downout_put(S+epsilon,K,X,r,q,tau,sigma)
    option_S_lower=_barrier_downout_put(S-epsilon,K,X,r,q,tau,sigma)
    option_sigam_upper=_barrier_downout_put(S,K,X,r,q,tau,sigma+epsilon)
    option_sigam_lower=_barrier_downout_put(S,K,X,r,q,tau,sigma-epsilon)
    option_tau_upper=_barrier_downout_put(S,K,X,r,q,tau+100*epsilon,sigma)
    option_tau_lower=_barrier_downout_put(S,K,X,r,q,tau-100*epsilon,sigma)
    option_r_upper=_barrier_downout_put(S,K,X,r+epsilon,q,tau,sigma)
    option_r_lower=_barrier_downout_put(S,K,X,r-epsilon,q,tau,sigma)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }
```

1.4 Asian

In [4]:

```
def asian_geometric_call(S,K,r,q,tau,sigma,freq='1M',**kwargs):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    period=ql.Period(freq)
    asianFutureFixingDates = [start+period]
    while asianFutureFixingDates[-1]<=maturity:
        asianFutureFixingDates.append(asianFutureFixingDates[-1]+period)
    asianFutureFixingDates.pop(-1)
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option = ql.DiscreteAveragingAsianOption(ql.Average().Geometric,
                                             1, 0,
                                             asianFutureFixingDates,
                                             payoff,
                                             exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticDiscreteGeometricAveragePriceAsianEngine(bsm_pr
    return {
    'PV':option.NPV(),
    'delta':option.delta(),
    'gamma':option.gamma(),
    'vega':option.vega(),
    'theta':option.theta(),
    'rho':option.rho(),
    }

def asian_geometric_put(S,K,r,q,tau,sigma,freq='1M',**kwargs):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    period=ql.Period(freq)
    asianFutureFixingDates = [start+period]
```

```
    while asianFutureFixingDates[-1]<=maturity:
        asianFutureFixingDates.append(asianFutureFixingDates[-1]+period)
    asianFutureFixingDates.pop(-1)
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option = ql.DiscreteAveragingAsianOption(ql.Average().Geometric,
                                             1, 0,
                                             asianFutureFixingDates,
                                             payoff,
                                             exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticDiscreteGeometricAveragePriceAsianEngine(bsm_pr
    return {
    'PV':option.NPV(),
    'delta':option.delta(),
    'gamma':option.gamma(),
    'vega':option.vega(),
    'theta':option.theta(),
    'rho':option.rho(),
    }

def _asian_arithmetic_call(S,K,r,q,tau,sigma,freq='1M'):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    period=ql.Period(freq)
    asianFutureFixingDates = [start+period]
    while asianFutureFixingDates[-1]<=maturity:
        asianFutureFixingDates.append(asianFutureFixingDates[-1]+period)
    asianFutureFixingDates.pop(-1)
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option = ql.DiscreteAveragingAsianOption(ql.Average().Arithmetic,
                                             0, 0,
                                             asianFutureFixingDates,
                                             payoff,
                                             exercise)
```

```
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    numPaths=10000
    option.setPricingEngine(ql.MCDiscreteArithmeticAPEngine(bsm_process,'lowdiscrepanc
                                                            antitheticVariate=True,
                                                            controlVariate=True,
                                                            requiredSamples=numPaths,s
    return option

def asian_arithmetic_call(S,K,r,q,tau,sigma,freq='1M',**kwargs):
    option=_asian_arithmetic_call(S,K,r,q,tau,sigma,freq)
    epsilon=1e-4
    option_S_upper=_asian_arithmetic_call(S+epsilon,K,r,q,tau,sigma,freq)
    option_S_lower=_asian_arithmetic_call(S-epsilon,K,r,q,tau,sigma,freq)
    option_sigam_upper=_asian_arithmetic_call(S,K,r,q,tau,sigma+epsilon,freq)
    option_sigam_lower=_asian_arithmetic_call(S,K,r,q,tau,sigma-epsilon,freq)
    option_tau_upper=_asian_arithmetic_call(S,K,r,q,tau+100*epsilon,sigma,freq)
    option_tau_lower=_asian_arithmetic_call(S,K,r,q,tau-100*epsilon,sigma,freq)
    option_r_upper=_asian_arithmetic_call(S,K,r+epsilon,q,tau,sigma,freq)
    option_r_lower=_asian_arithmetic_call(S,K,r-epsilon,q,tau,sigma,freq)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def _asian_arithmetic_put(S,K,r,q,tau,sigma,freq='1M'):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    period=ql.Period(freq)
    asianFutureFixingDates = [start+period]
    while asianFutureFixingDates[-1]<=maturity:
        asianFutureFixingDates.append(asianFutureFixingDates[-1]+period)
    asianFutureFixingDates.pop(-1)
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
```

```
    option = ql.DiscreteAveragingAsianOption(ql.Average().Arithmetic,
                                             0, 0,
                                             asianFutureFixingDates,
                                             payoff,
                                             exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    numPaths=10000
    option.setPricingEngine(ql.MCDiscreteArithmeticAPEngine(bsm_process,'lowdiscrepanc
                                                            antitheticVariate=True,
                                                            controlVariate=True,
                                                            requiredSamples=numPaths,s
    return option

def asian_arithmetic_put(S,K,r,q,tau,sigma,freq='1M',**kwargs):
    option=_asian_arithmetic_put(S,K,r,q,tau,sigma,freq)
    epsilon=1e-4
    option_S_upper=_asian_arithmetic_put(S+epsilon,K,r,q,tau,sigma,freq)
    option_S_lower=_asian_arithmetic_put(S-epsilon,K,r,q,tau,sigma,freq)
    option_sigam_upper=_asian_arithmetic_put(S,K,r,q,tau,sigma+epsilon,freq)
    option_sigam_lower=_asian_arithmetic_put(S,K,r,q,tau,sigma-epsilon,freq)
    option_tau_upper=_asian_arithmetic_put(S,K,r,q,tau+100*epsilon,sigma,freq)
    option_tau_lower=_asian_arithmetic_put(S,K,r,q,tau-100*epsilon,sigma,freq)
    option_r_upper=_asian_arithmetic_put(S,K,r+epsilon,q,tau,sigma,freq)
    option_r_lower=_asian_arithmetic_put(S,K,r-epsilon,q,tau,sigma,freq)
    return {
        'PV':option.NPV(),
        'delta':(option_S_upper.NPV()-option_S_lower.NPV())/(2*epsilon),
        'gamma':(option_S_upper.NPV()-2*option.NPV()+option_S_lower.NPV())/(epsilon**2
        'vega':(option_sigam_upper.NPV()-option_sigam_lower.NPV())/(2*epsilon),
        'theta':(option_tau_upper.NPV()-option_tau_lower.NPV())/(2*100*epsilon),
        'rho':(option_r_upper.NPV()-option_r_lower.NPV())/(2*epsilon),
        }

def asian_continuous_geometric_call(S,K,r,q,tau,sigma,**kwargs):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Call
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
```

```
    option = ql.ContinuousAveragingAsianOption(ql.Average().Geometric,
                                               payoff,
                                               exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
                                           flat_vol_ts)
    option.setPricingEngine(ql.AnalyticContinuousGeometricAveragePriceAsianEngine(bsm_
    return {
    'PV':option.NPV(),
    'delta':option.delta(),
    'gamma':option.gamma(),
    'vega':option.vega(),
    'theta':option.theta(),
    'rho':option.rho(),
    }

def asian_continuous_geometric_put(S,K,r,q,tau,sigma,**kwargs):
    # specify timeframe
    tau_day=int(round(tau*365))
    start=ql.Date(1,1,2000)
    ql.Settings.instance().evaluationDate=start
    maturity=start+tau_day
    day_count = ql.Actual365Fixed()
    calendar = ql.UnitedStates()
    # specify option type
    option_type = ql.Option.Put
    payoff = ql.PlainVanillaPayoff(option_type, K)
    exercise = ql.EuropeanExercise(maturity)
    option = ql.ContinuousAveragingAsianOption(ql.Average().Geometric,
                                               payoff,
                                               exercise)
    # specify pricing method
    spot_handle = ql.QuoteHandle(
    ql.SimpleQuote(S)
    )
    flat_ts = ql.YieldTermStructureHandle(
        ql.FlatForward(start, r, day_count)
    )
    dividend_yield = ql.YieldTermStructureHandle(
        ql.FlatForward(start, q, day_count)
    )
    flat_vol_ts = ql.BlackVolTermStructureHandle(
        ql.BlackConstantVol(start, calendar, sigma, day_count)
    )
    bsm_process = ql.BlackScholesMertonProcess(spot_handle,
                                           dividend_yield,
                                           flat_ts,
```

```
                                                flat_vol_ts)
    option.setPricingEngine(ql.AnalyticContinuousGeometricAveragePriceAsianEngine(bsm_
    return {
    'PV':option.NPV(),
    'delta':option.delta(),
    'gamma':option.gamma(),
    'vega':option.vega(),
    'theta':option.theta(),
    'rho':option.rho(),
    }
```

1.5 Visualization

In [5]:

```
# ========================================================================
# Visualization
# ========================================================================
def option_plot(option_fcn,changing_param_name,changing_param_bound,other_params,num=10
    import numpy as np
    import pandas as pd
    import numba as nb
    import plotly.express as px
    from math import ceil
    from plotly.subplots import make_subplots
    import plotly.graph_objects as go
    params=other_params.copy()
    def f(x):
        params[changing_param_name]=x
        return option_fcn(**params)
    f=np.vectorize(f)
    x=np.linspace(changing_param_bound[0],changing_param_bound[1],num)
    y=f(x)
    df=pd.DataFrame(np.array(list(map(lambda x:list(x.values()),y))),columns=y[0].keys(
    df.index.name=changing_param_name
#     for i,(name,col) in enumerate(df.iteritems()):
#         fig=px.line(col,y=name,color_discrete_sequence=[px.colors.qualitative.D3[i%10
#         fig.show()
    col_num=3
    fig = make_subplots(rows=ceil(len(df.columns)/col_num), cols=col_num)
    for i,(name,col) in enumerate(df.iteritems()):
        fig.add_trace(go.Scatter(x=df.index,y=col),row=i//col_num+1,col=i%col_num+1)
        fig.update_xaxes(title_text=changing_param_name, row=i//col_num+1, col=i%col_nu
        fig.update_yaxes(title_text=name, row=i//col_num+1, col=i%col_num+1)
    fig.update_layout(showlegend=False,height=ceil(len(df.columns)/col_num)*300, width=
    fig.show()
    return df
```

In [6]:

```
def interact_plot(changing_param_name,other_params,num=100,**kwargs):
    from functools import partial
    import numpy as np
    from ipywidgets import interact
    from ipywidgets import FloatSlider, Dropdown, FloatRangeSlider
    min_dict={
        'S':0,
        'K':0,
        'X':0,
        'r':-0.05,
        'q':0,
        'tau':0,
        'sigma':0
    }
    max_dict={
        'S':200,
        'K':200,
        'X':200,
        'r':0.2,
        'q':0.5,
        'tau':5,
        'sigma':1
    }
    option_list=[
        ('European Call',vanilla_euro_call),
        ('European Put',vanilla_euro_put),
        ('American Call',vanilla_ame_call),
        ('American Put',vanilla_ame_put),
        ('Barrier UpIn Call',barrier_upin_call),
        ('Barrier UpIn Put',barrier_upin_put),
        ('Barrier UpOut Call',barrier_upout_call),
        ('Barrier UpOut Put',barrier_upout_put),
        ('Barrier DownIn Call',barrier_downin_call),
        ('Barrier DownIn Put',barrier_downin_put),
        ('Barrier DownOut Call',barrier_downout_call),
        ('Barrier DownOut Put',barrier_downout_put),
        ('Asian Geometric Call',asian_geometric_call),
        ('Asian Geometric Put',asian_geometric_put),
        ('Asian Arithmetic Call',asian_arithmetic_call),
        ('Asian Arithmetic Put',asian_arithmetic_put),
        ('Asian Continuous Geometric Call',asian_continuous_geometric_call),
        ('Asian Continuous Geometric Put',asian_continuous_geometric_put)
    ]
    widgets_dict={'option_fcn': Dropdown(
        options=option_list,
        description='Option Type:'
    ),
                  'changing_param_bound':FloatRangeSlider(
                     value=[min_dict[changing_param_name]+(
                         max_dict[changing_param_name]-min_dict[changing_param_name])*0
                           min_dict[changing_param_name]+(
                               max_dict[changing_param_name]-min_dict[changing_param_na
                      min=min_dict[changing_param_name],
                      max=max_dict[changing_param_name],
                      step=10**(round(np.log10(max_dict[changing_param_name]-min_dict[c
                      description='x range:',
                      disabled=False,
                      continuous_update=False,
                      orientation='horizontal',
```

```
                    readout=True,
                    readout_format='.2f' if np.log10(max_dict[changing_param_name]-mi
                )}
    for key,value in other_params.items():
        widgets_dict[key]=FloatSlider(
                value=value,
                min=min_dict[key],
                max=max_dict[key],
                step=10**(round(np.log10(max_dict[key]-min_dict[key]))-2),
                description=key,
                disabled=False,
                continuous_update=False,
                orientation='horizontal',
                readout=True,
                readout_format='.3f',
        )
    def f(option_fcn,changing_param_bound,**other_params):
        return option_plot(option_fcn,changing_param_name,changing_param_bound,other_pa
    interact(f,**widgets_dict)
```

2 Interact Plot

In [7]:

```
changing_param_name='S'
other_params={
    'S':100,
    'K':100,
    'X':120,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
interact_plot(changing_param_name,other_params)
```

Option Type: European Call

x range: 50 – 150

S 100.000

K 100.000

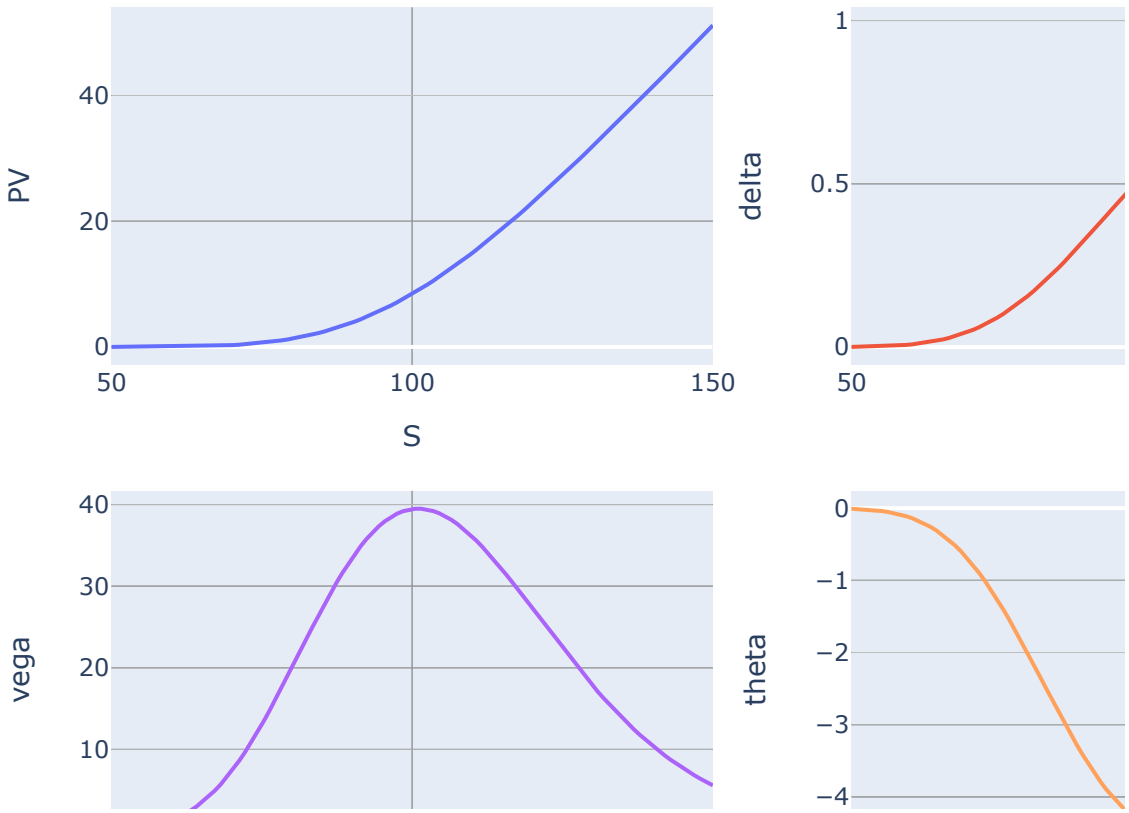
X 120.000

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

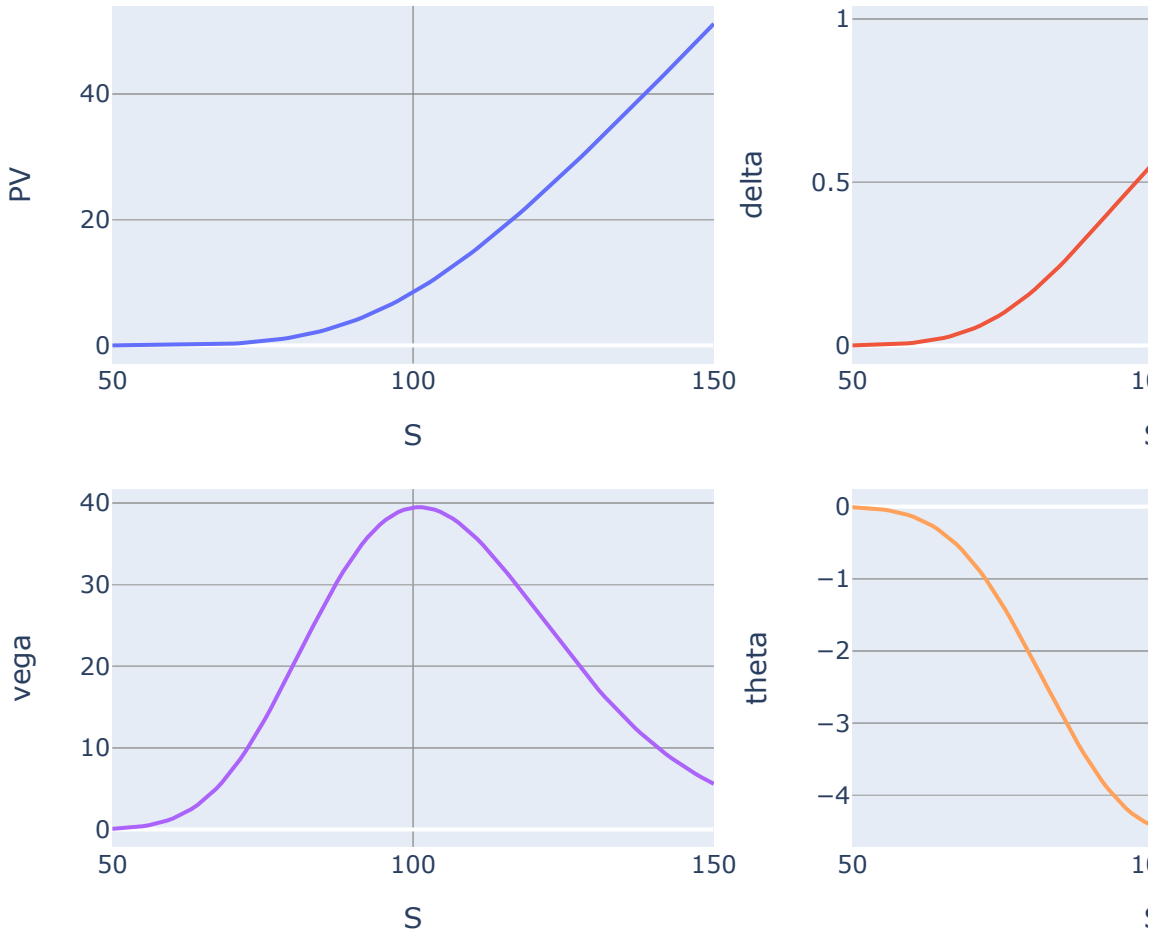
100 rows × 6 columns

3 Vanilla European

3.1 ...Vs S, call

In [8]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

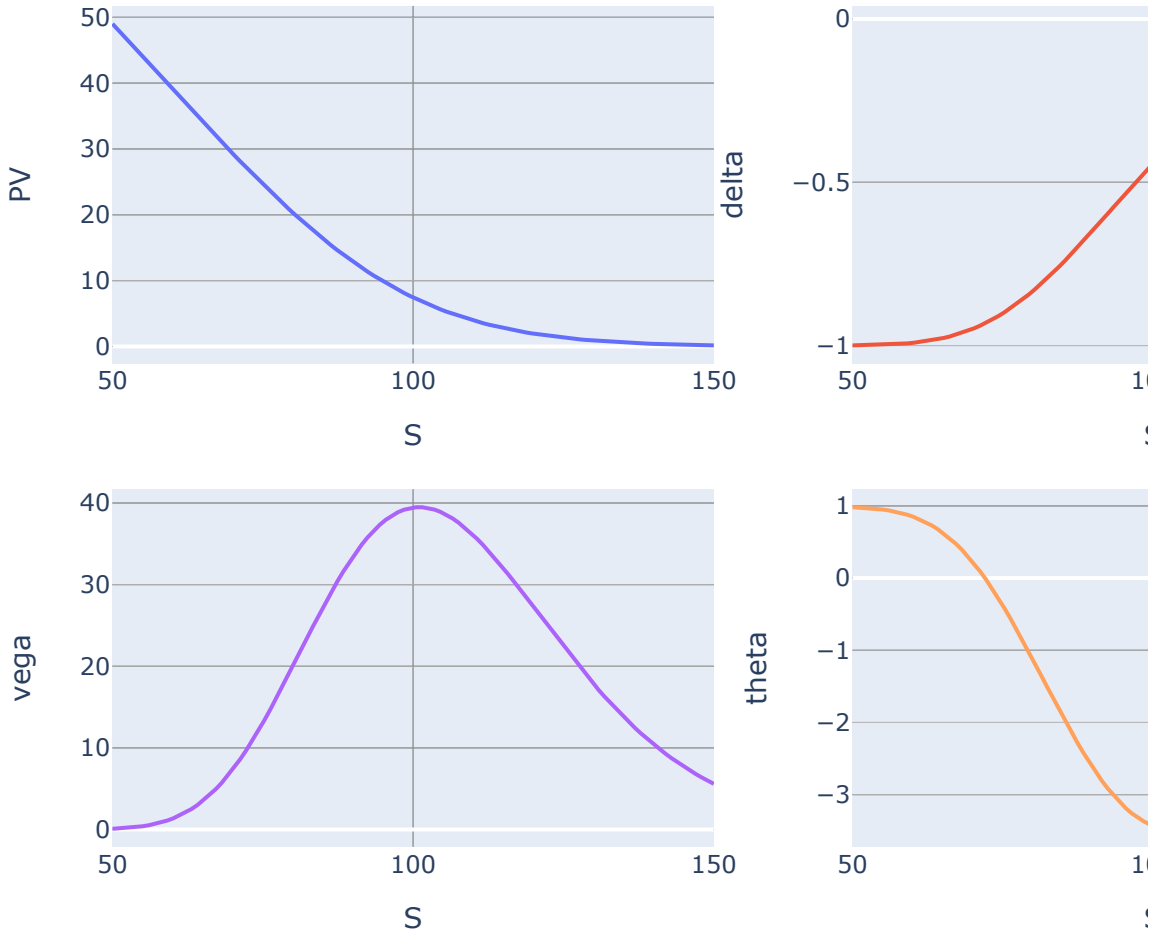
S	PV	delta	gamma	vega	theta	rho
53.030303	0.003566	0.001257	0.000392	0.220269	-0.022658	0.063119
54.040404	0.005055	0.001710	0.000509	0.297176	-0.030591	0.087365
...	...	...	...	...	...	...
145.959596	47.192566	0.979365	0.001703	7.256971	-1.683248	95.755089
146.969697	48.182670	0.981020	0.001576	6.806594	-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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)
```



Out[9]:

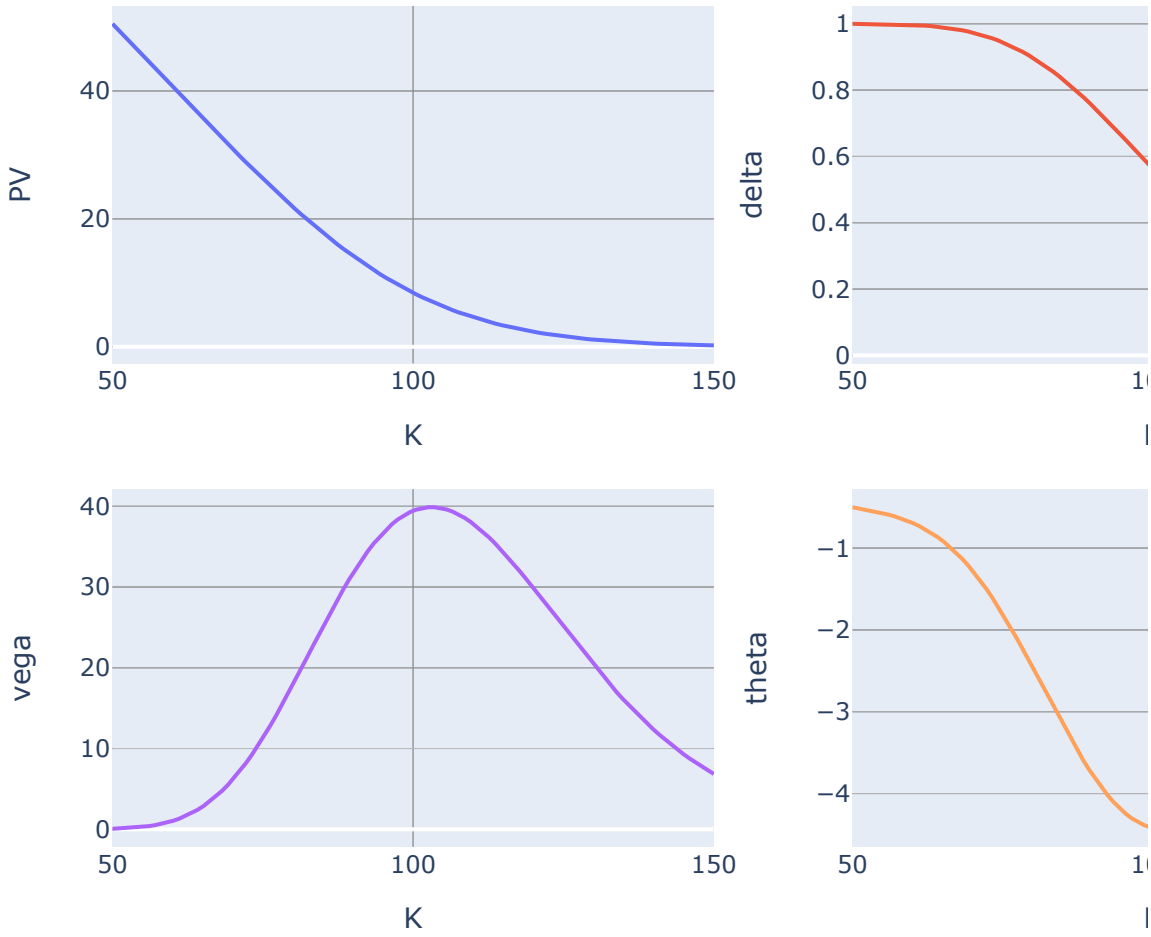
S	PV	delta	gamma	vega	theta	rho
50.000000	49.006125	-0.999543	0.000164	0.081760	0.981657	-98.983275
51.010101	47.996579	-0.999349	0.000222	0.115627	0.978172	-98.973495
52.020202	46.987260	-0.999089	0.000297	0.160847	0.973516	-98.960052

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]:

```
changing_param_name='K'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)
```



Out[10]:

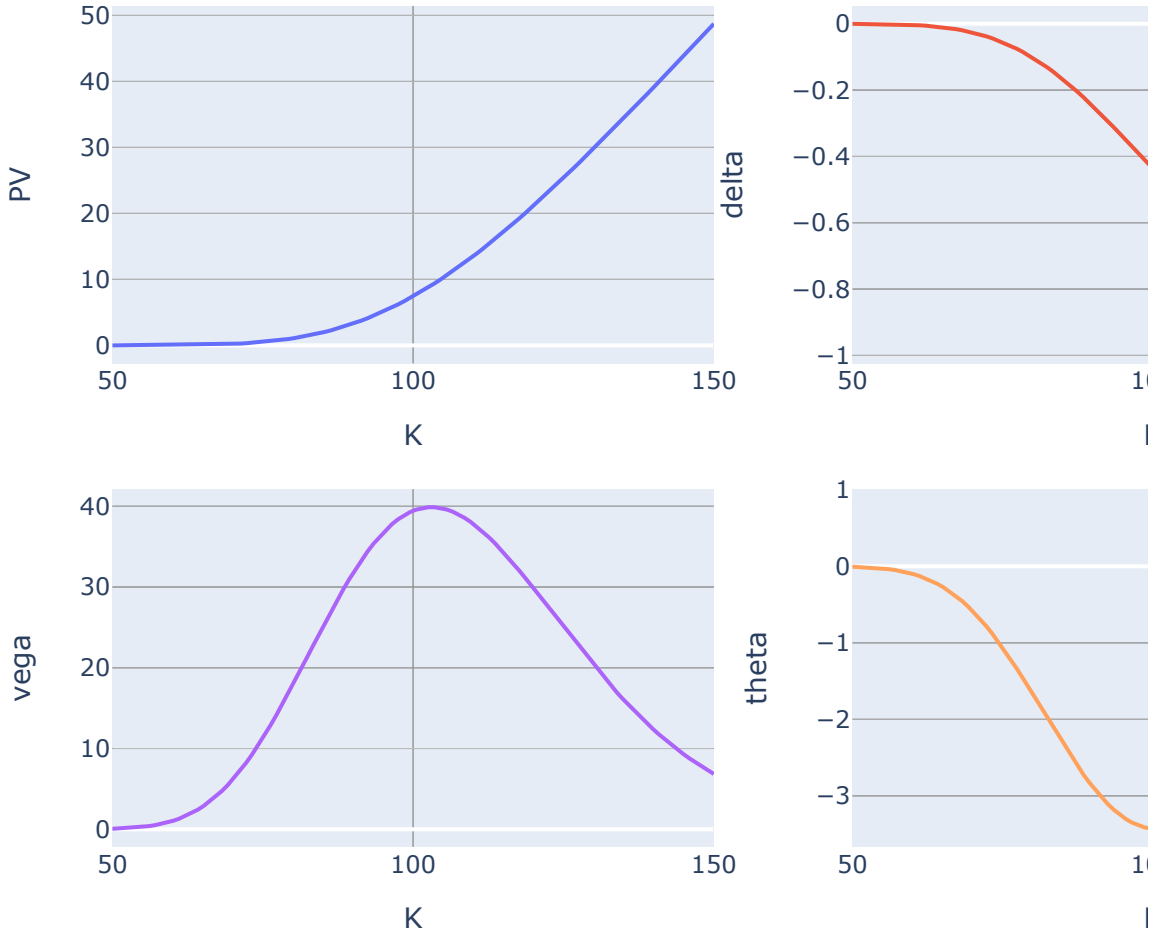
K	PV	delta	gamma	vega	theta	rho
50.000000	50.498278	0.999850	0.000029	0.057813	-0.500649	49.486747
51.010101	49.498611	0.999781	0.000041	0.082583	-0.513053	50.479462
52.020202	48.499108	0.999684	0.000058	0.116011	-0.526294	51.469315

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]:

```
changing_param_name='K'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)
```



Out[11]:

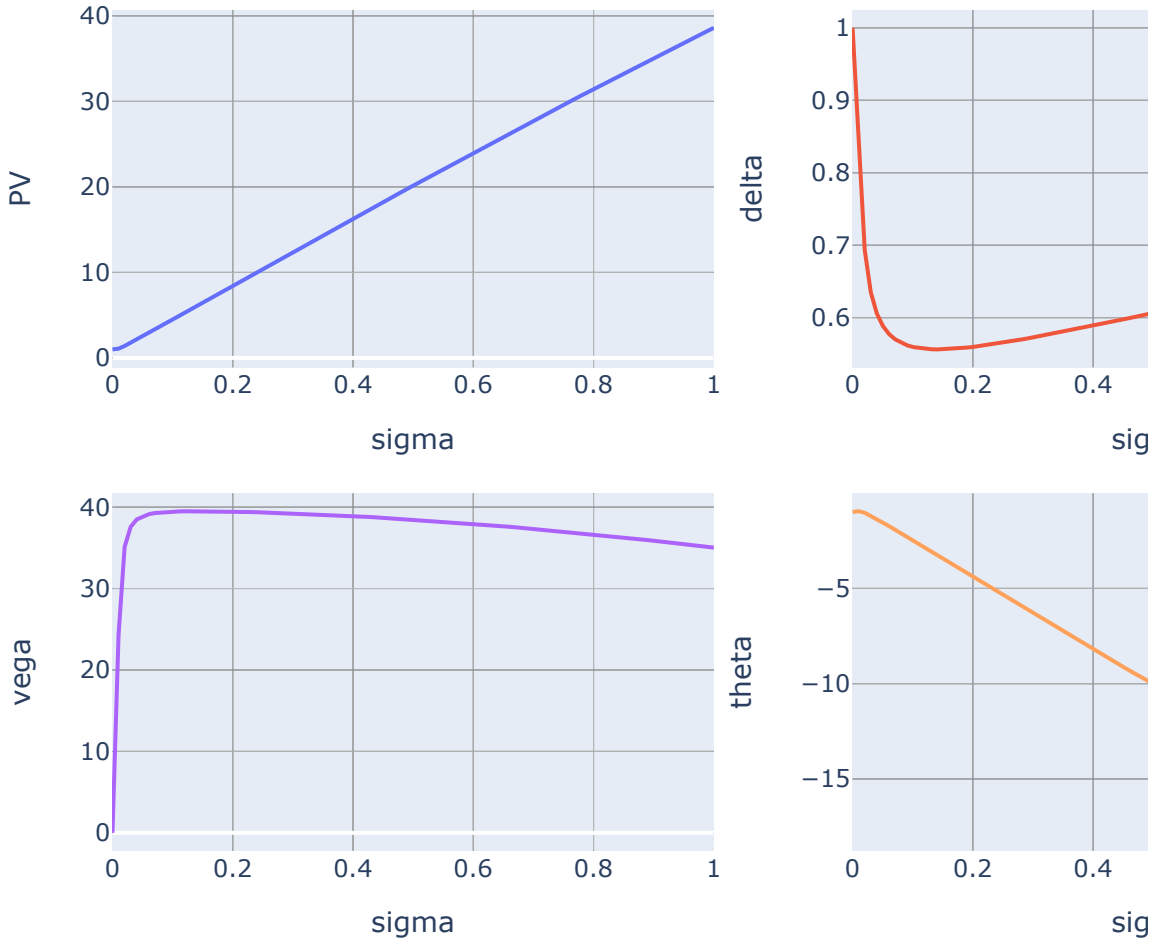
K	PV	delta	gamma	vega	theta	rho
50.000000	0.000769	-0.000150	0.000029	0.057813	-0.005624	-0.015744
51.010101	0.001153	-0.000219	0.000041	0.082583	-0.008027	-0.023080
52.020202	0.001700	-0.000316	0.000058	0.116011	-0.011268	-0.033278

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]:

```
changing_param_name='sigma'
changing_param_bound=(1e-8,1)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)
```



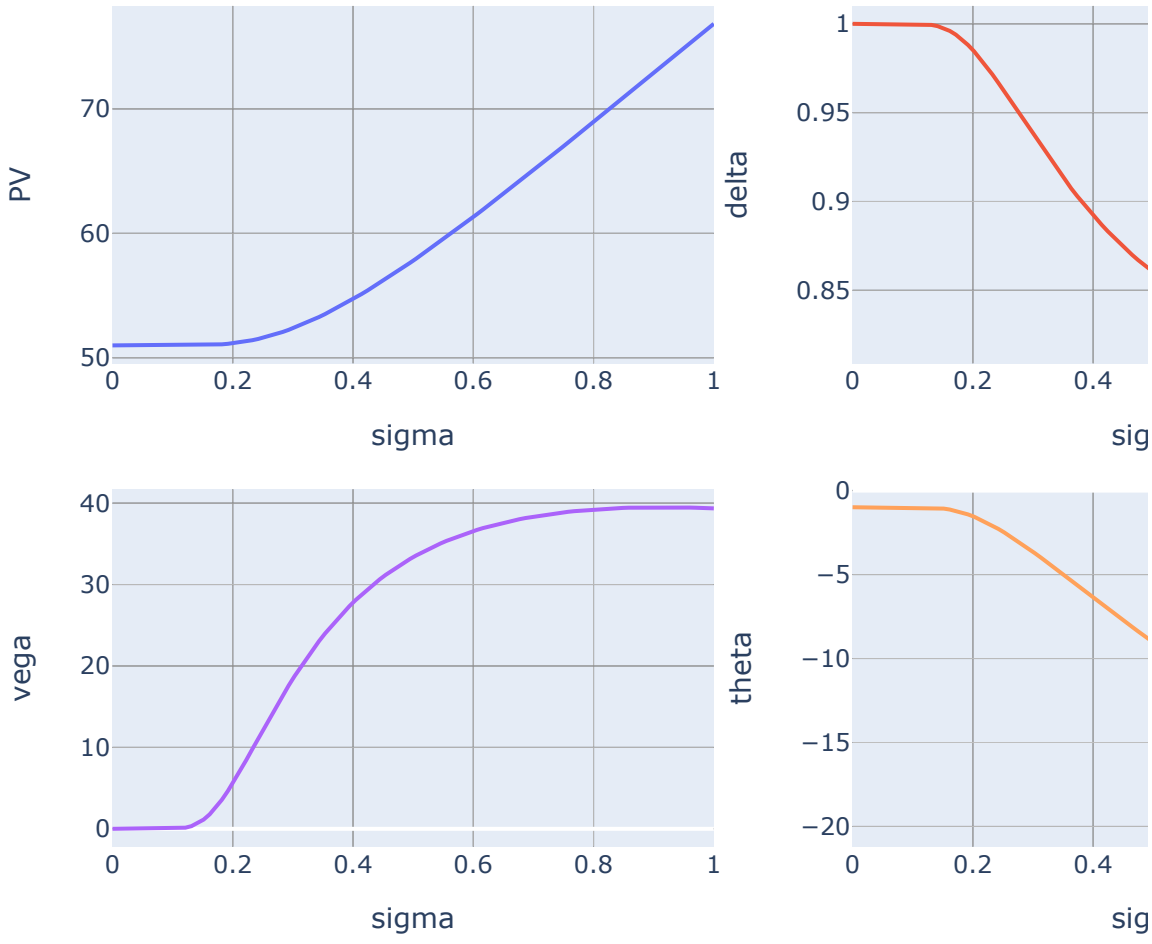
Out[12]:

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]:

```
changing_param_name='sigma'
changing_param_bound=(1e-8,1)
other_params={
    'S':150,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_call,changing_param_name,changing_param_bound,other_params)
```



Out[13]:

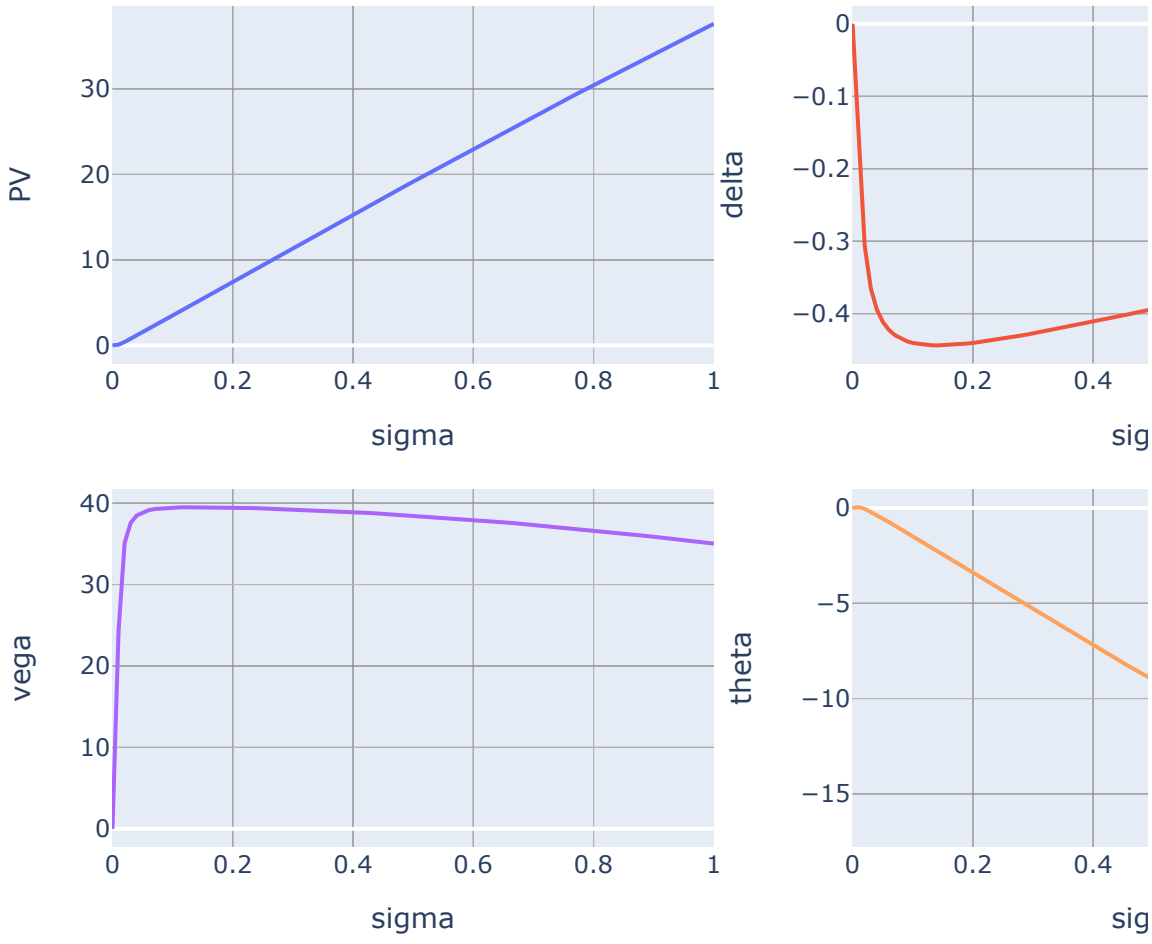
sigma	PV	delta	gamma	vega	theta	rho
1.000000e-08	50.995017	1.000000	0.000000e+00	0.000000e+00	-0.990050	99.004983
1.010102e-02	50.995017	1.000000	0.000000e+00	0.000000e+00	-0.990050	99.004983
2.020203e-02	50.995017	1.000000	1.544760e-93	7.021639e-91	-0.990050	99.004983

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]:

```
changing_param_name='sigma'
changing_param_bound=(1e-8,1)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)
```



Out[14]:

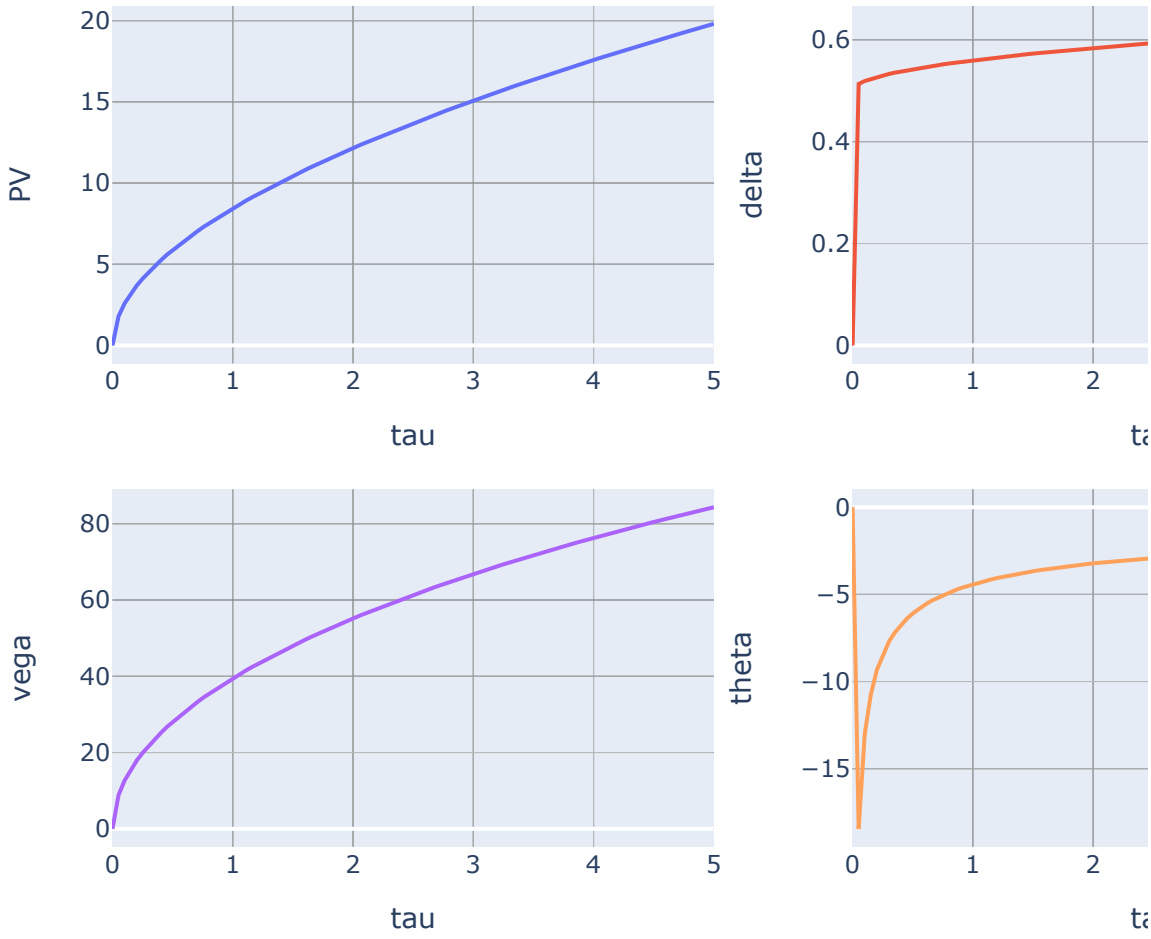
sigma	PV	delta	gamma	vega	theta	rho
1.000000e-08	-1.104690e-16	-1.110223e-16	0.000000	0.000000	1.099176e-16	-1.099176e-14
1.010102e-02	8.534364e-02	-1.598561e-01	0.240737	24.316858	3.789699e-02	-1.607095e+01

sigma	PV	delta	gamma	vega	theta	rho
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]:

```
changing_param_name='tau'
changing_param_bound=(1e-8,5)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

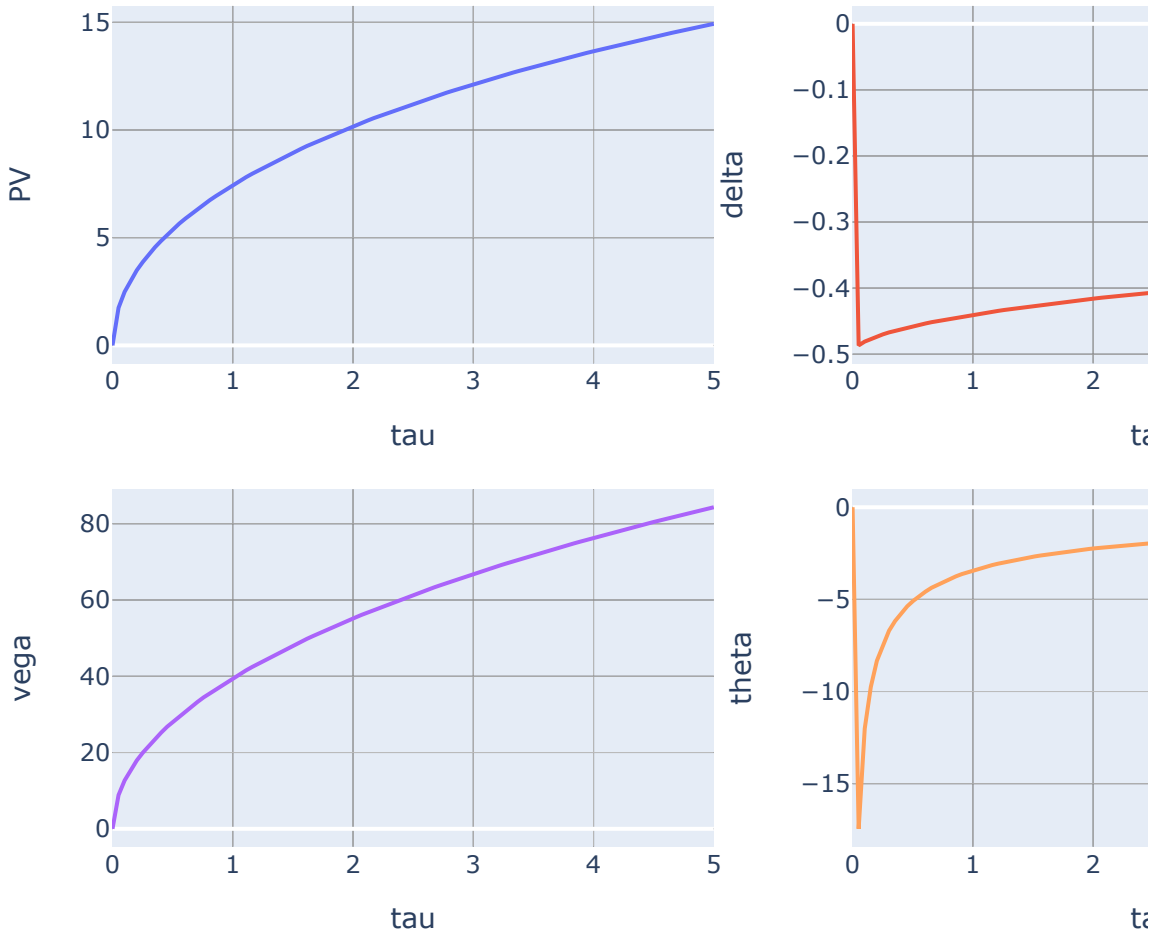
tau	PV	delta	gamma	vega	theta	rho
1.515152e-01	3.170000	0.523216	0.051299	15.459974	-10.751317	7.406408
2.020202e-01	3.689930	0.526924	0.044200	17.922102	-9.329980	9.934749
...	...	...	...	...	...	...
4.797980e+00	19.372820	0.628748	0.008629	82.788152	-2.160758	208.690414
4.848485e+00	19.485018	0.629420	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]:

```
changing_param_name='tau'
changing_param_bound=(1e-8,5)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_euro_put,changing_param_name,changing_param_bound,other_params)
```



Out[16]:

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	1.746737	-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

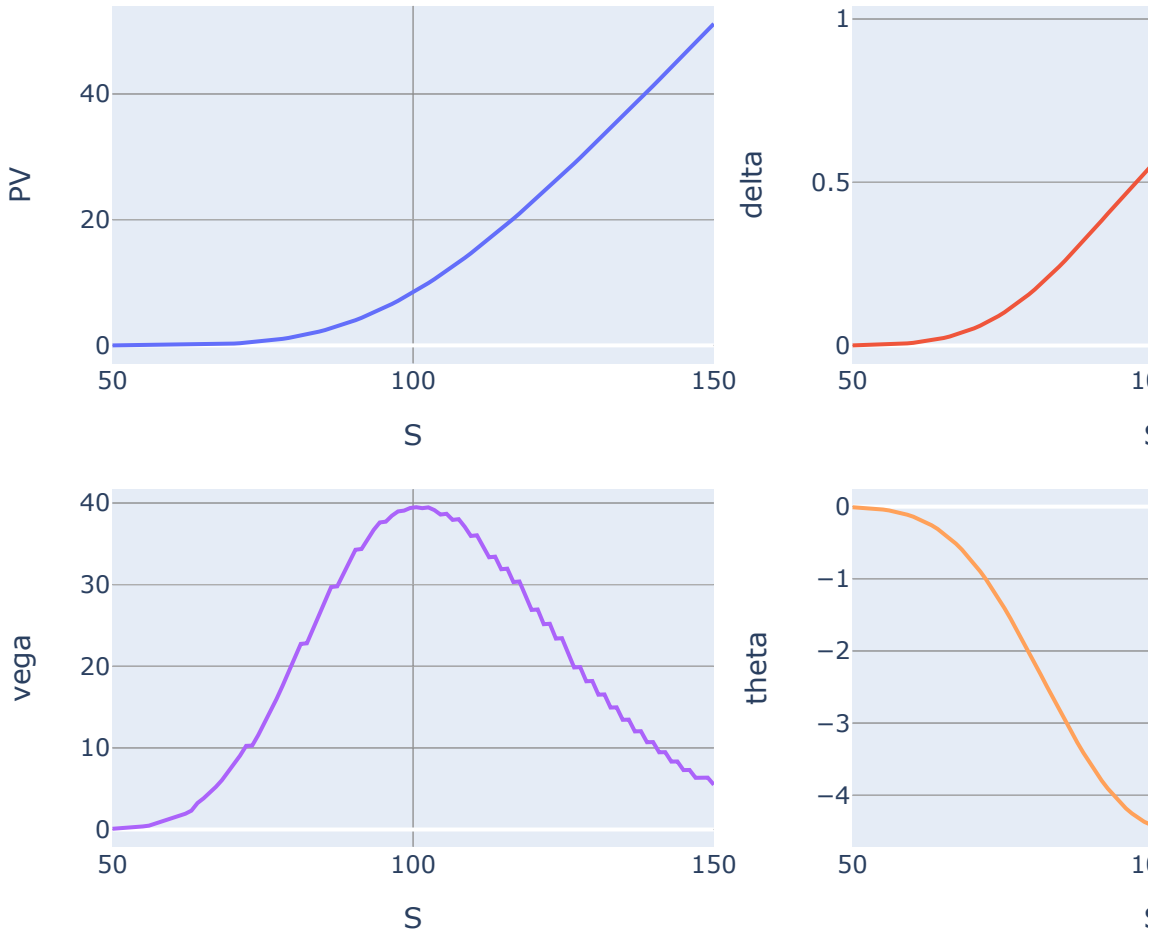
tau	PV	delta	gamma	vega	theta	rho
1.515152e-01	3.019429	-0.476784	0.051299	15.459974	-9.752823	-7.639396
2.020202e-01	3.487396	-0.473076	0.044200	17.922102	-8.332006	-10.298162
...	...	...	...	...	...	...
4.797980e+00	14.688810	-0.371252	0.008629	82.788152	-1.207598	-248.565199
4.848485e+00	14.751404	-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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

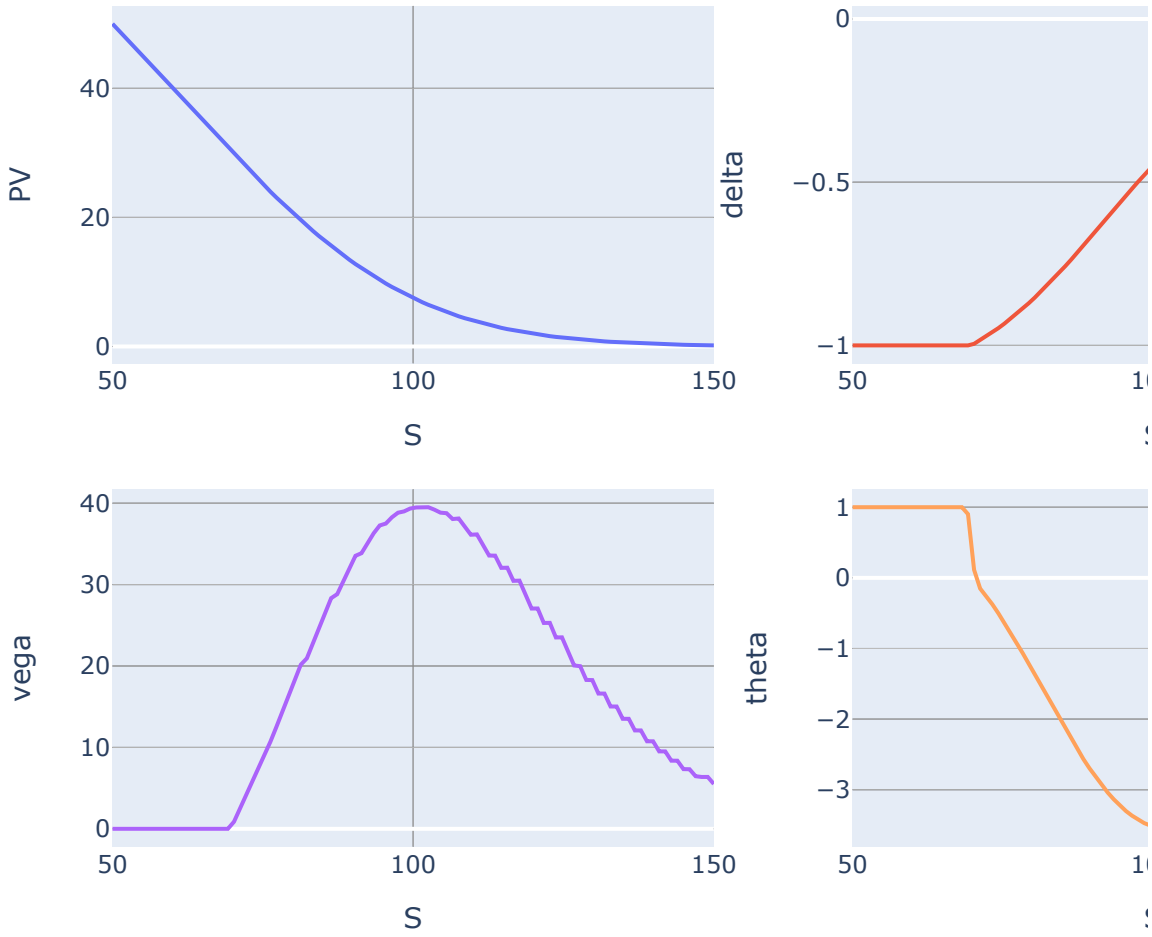
S	PV	delta	gamma	vega	theta	rho
53.030303	0.003515	0.001239	0.000386	0.236188	-0.022337	0.062185
54.040404	0.005008	0.001693	0.000504	0.299018	-0.030280	0.086462
...	...	...	...	...	...	...
145.959596	47.192486	0.979389	0.001701	7.292678	-1.682550	95.754873
146.969697	48.181677	0.981101	0.001571	6.333714	-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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)
```



Out[18]:

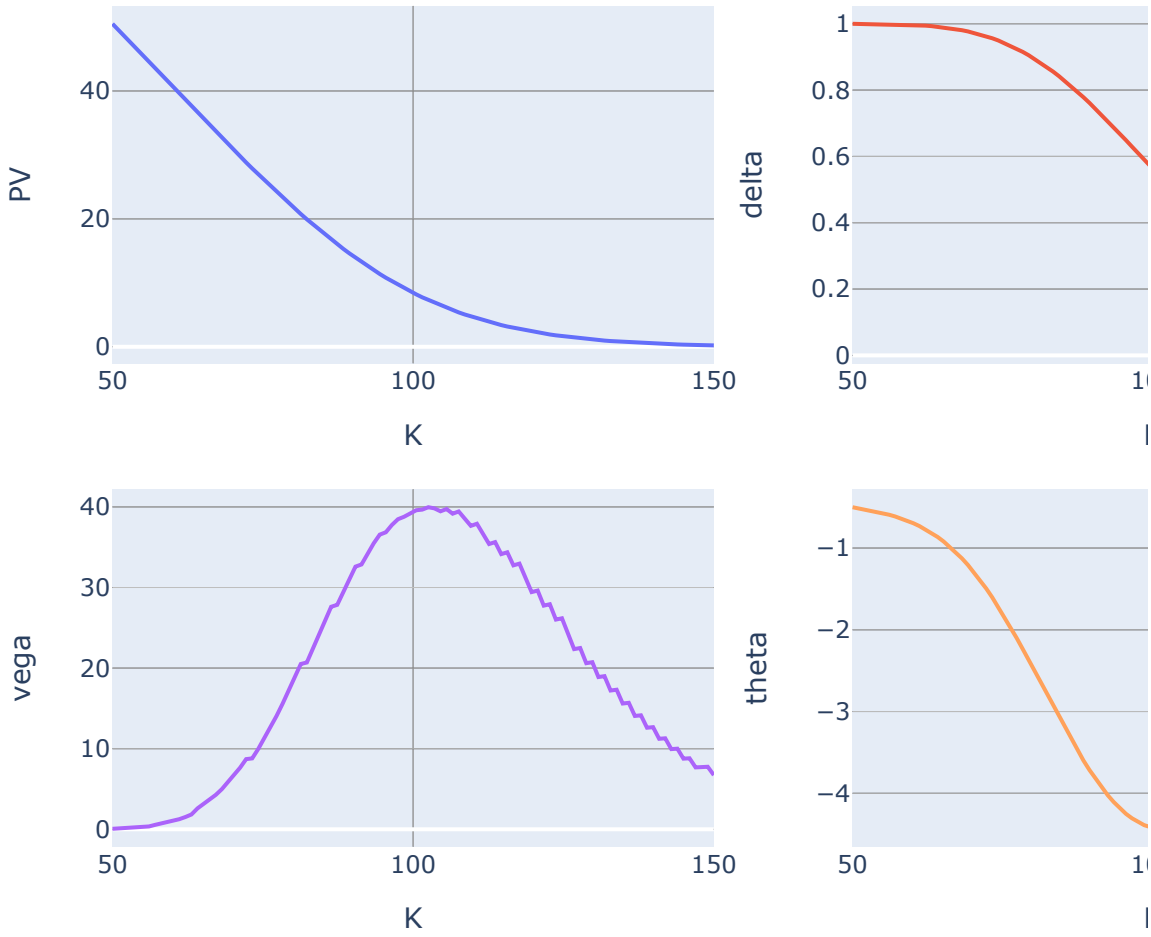
S	PV	delta	gamma	vega	theta	rho
50.000000	50.000000	-1.000000	0.000000	0.000000	1.000000	0.000000
51.010101	48.989899	-1.000000	0.000000	0.000000	1.000000	0.000000
52.020202	47.979798	-1.000000	0.000000	0.000000	1.000000	0.000000

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]:

```
changing_param_name='K'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)
```



Out[19]:

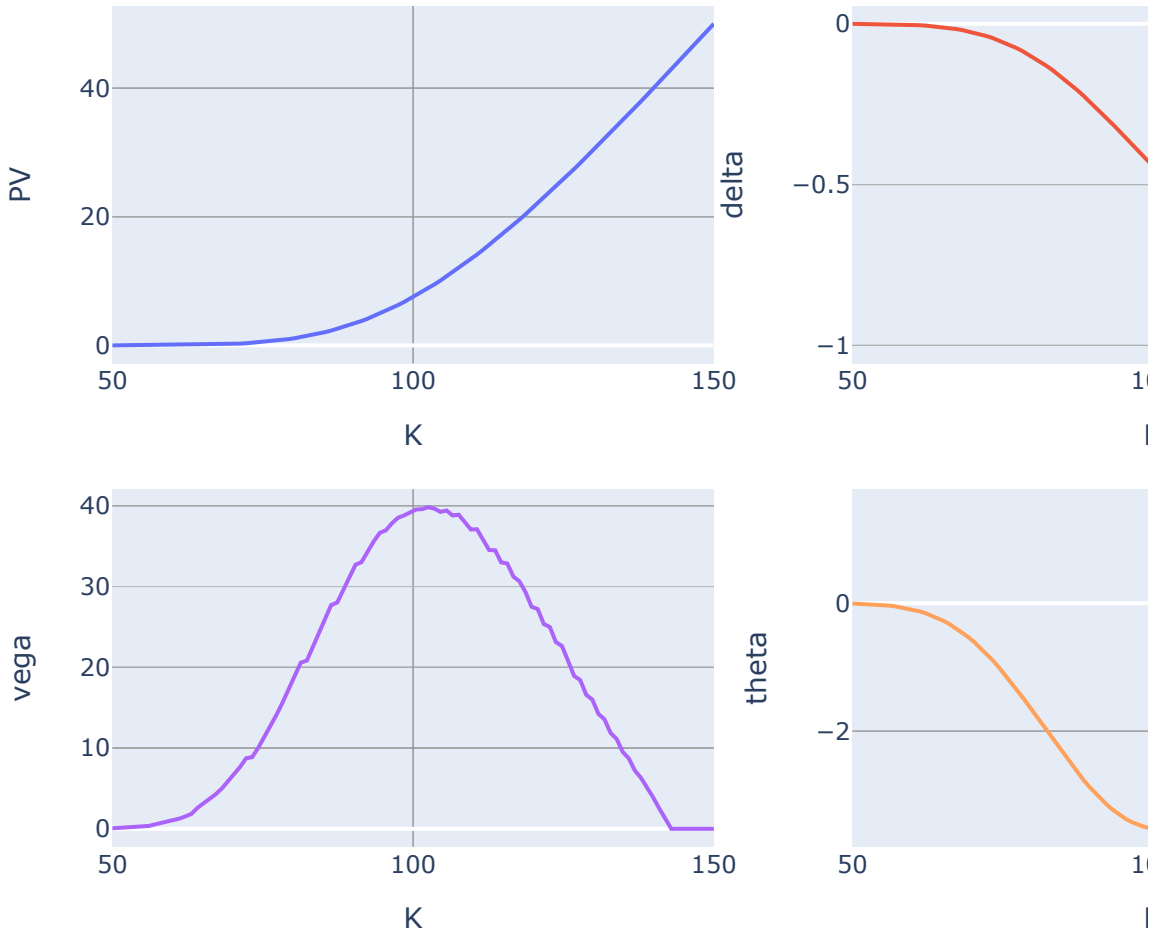
K	PV	delta	gamma	vega	theta	rho
50.000000	50.498257	0.999853	0.000028	0.061561	-0.500543	49.484427
51.010101	49.498589	0.999784	0.000041	0.080572	-0.512939	50.477157
52.020202	48.499073	0.999689	0.000057	0.104824	-0.526114	51.467239

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]:

```
changing_param_name='K'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)
```



Out[20]:

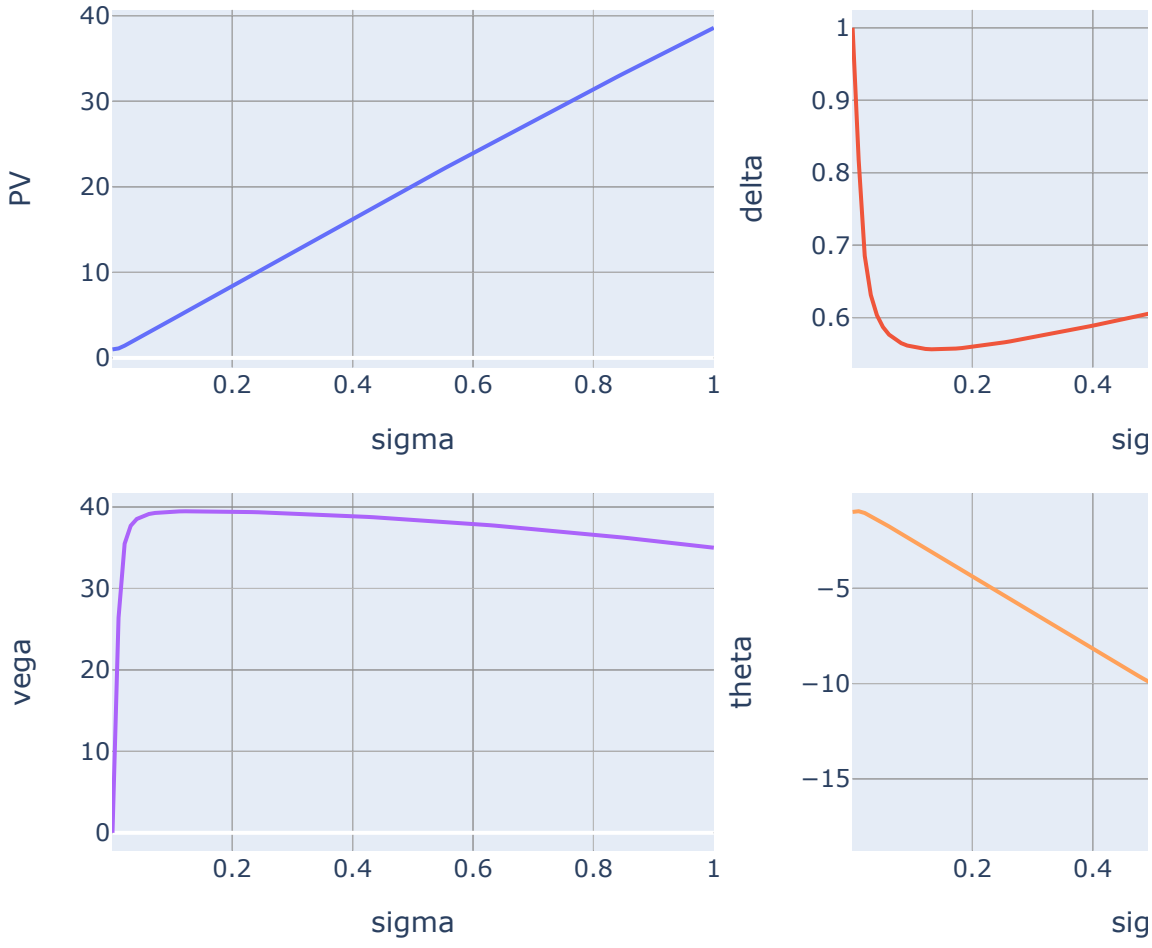
K	PV	delta	gamma	vega	theta	rho
50.000000	0.000757	-0.000147	0.000028	0.061833	-0.005538	-0.015209
51.010101	0.001142	-0.000217	0.000041	0.080931	-0.007942	-0.022408
52.020202	0.001677	-0.000311	0.000057	0.105394	-0.011127	-0.032199

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]:

```
changing_param_name='sigma'
changing_param_bound=(0.001,1)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)
```



Out[21]:

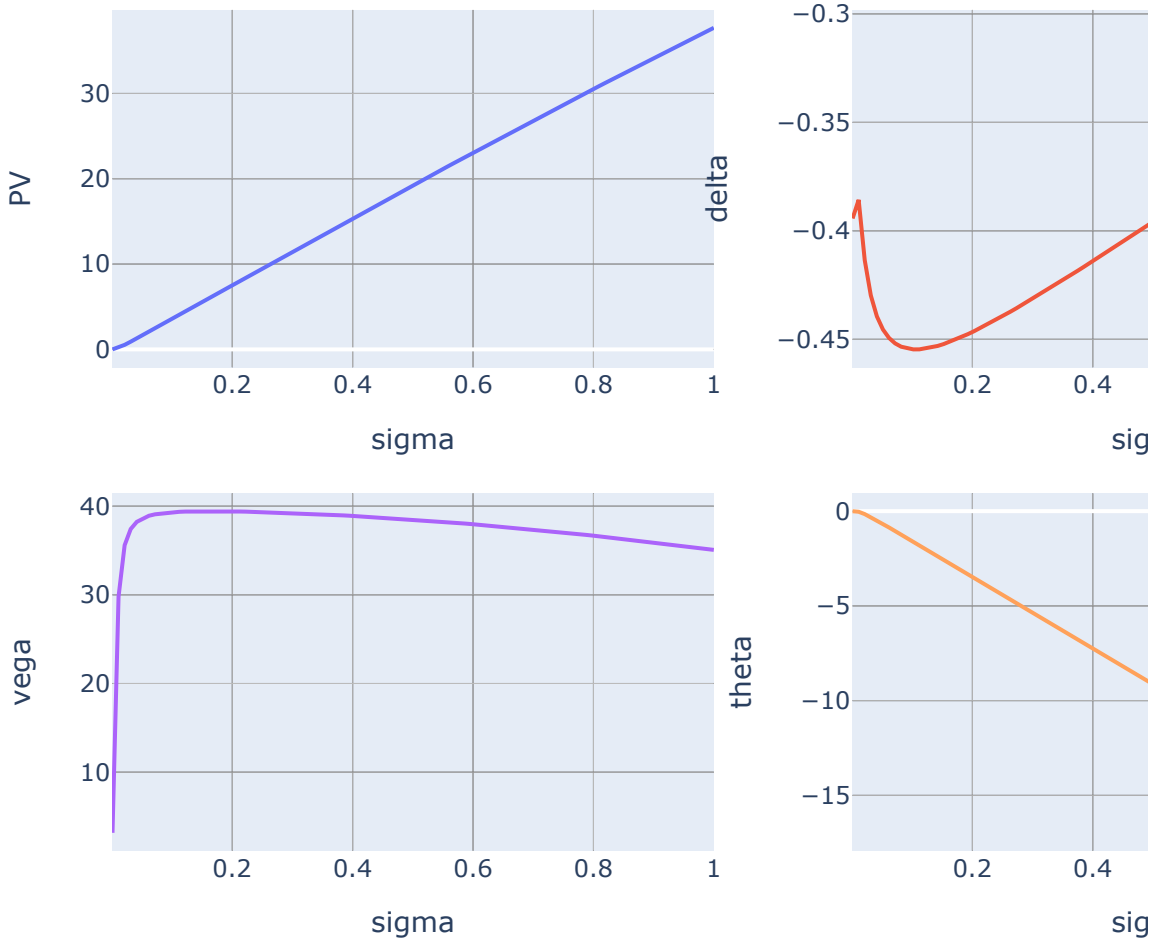
sigma	PV	delta	gamma	vega	theta	rho
0.001000	0.995010	1.000000	8.324049e-12	4.996004e-07	-0.990050	99.001302
0.011091	1.105327	0.817828	2.388222e-01	2.642145e+01	-0.953660	80.674429
0.021182	1.429995	0.685296	1.678538e-01	3.549448e+01	-1.047551	67.097100

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]:

```
changing_param_name='sigma'
changing_param_bound=(0.001,1)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_ame_put,changing_param_name,changing_param_bound,other_params)
```



Out[22]:

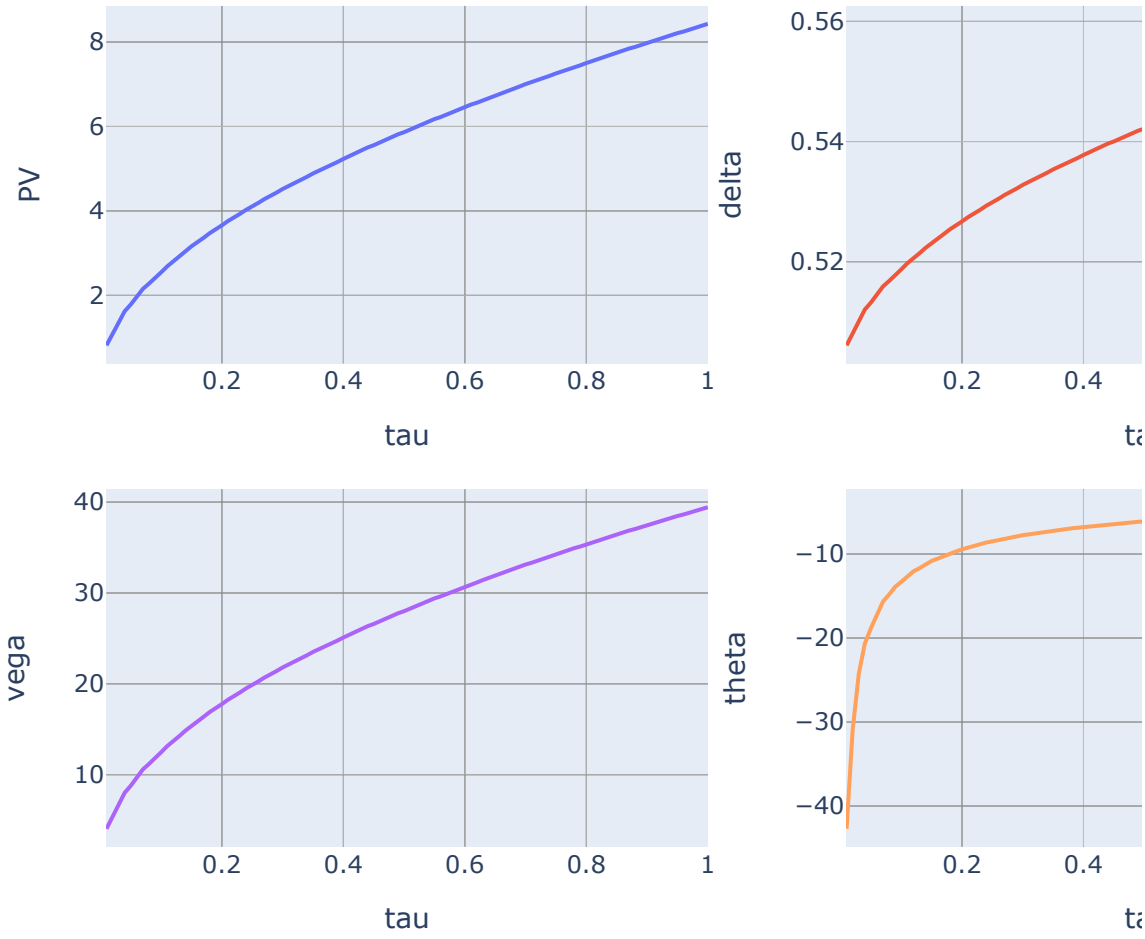
sigma	PV	delta	gamma	vega	theta	rho
0.001000	0.001702	-0.394316	79.550129	3.160911	-0.003418	-0.145938
0.011091	0.197148	-0.385738	0.682846	29.803719	-0.032269	-13.370055
0.021182	0.533548	-0.413564	0.256234	35.593787	-0.155923	-22.160783

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]:

```
changing_param_name='tau'
changing_param_bound=(0.01,1)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(vanilla_ame_call,changing_param_name,changing_param_bound,other_params)
```



Out[23]:

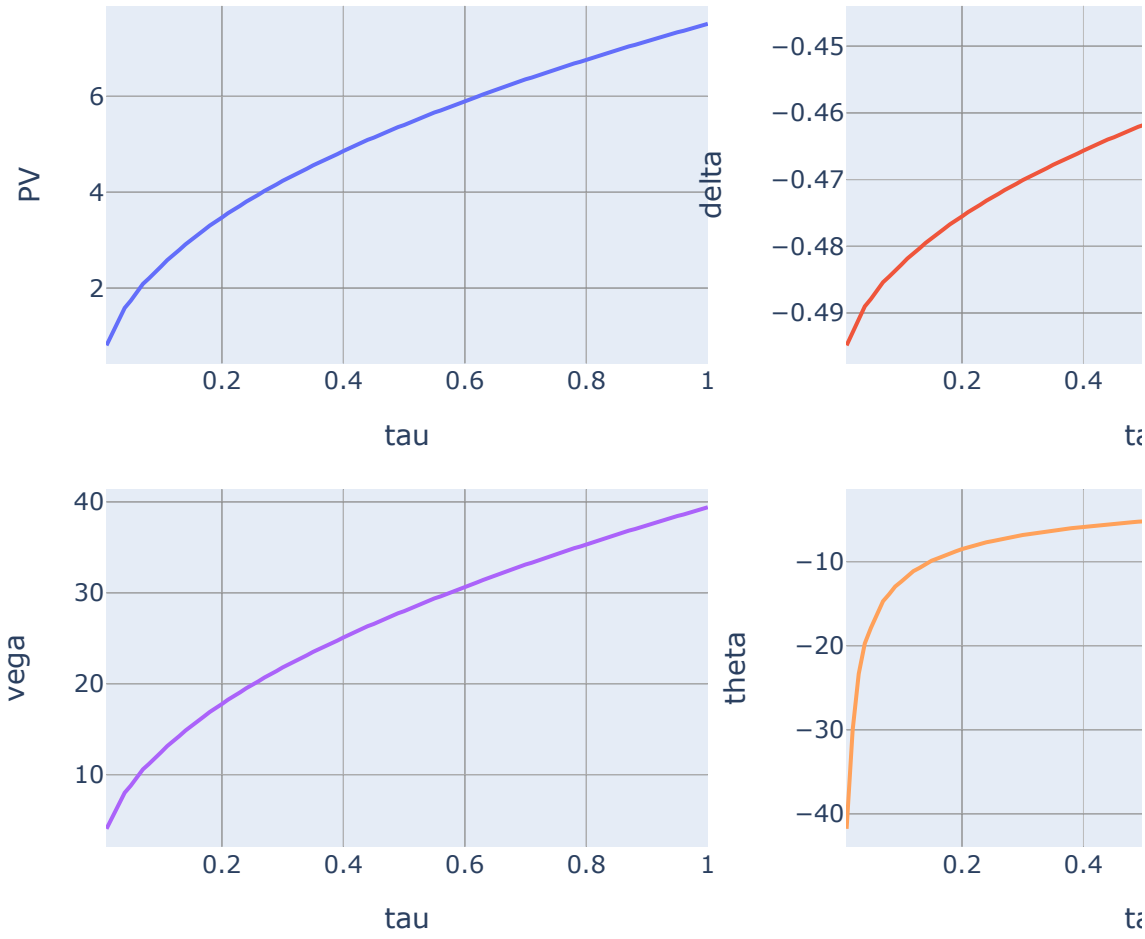
tau	PV	delta	gamma	vega	theta	rho
0.01	0.815061	0.506072	0.211055	4.047651	-42.708901	0.545652
0.02	1.094886	0.508140	0.152341	5.425880	-30.965425	0.953490
0.03	1.384325	0.510270	0.118974	6.845090	-24.291133	1.496038

tau	PV	delta	gamma	vega	theta	rho
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]:

```
changing_param_name='tau'
changing_param_bound=(0.01,1)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

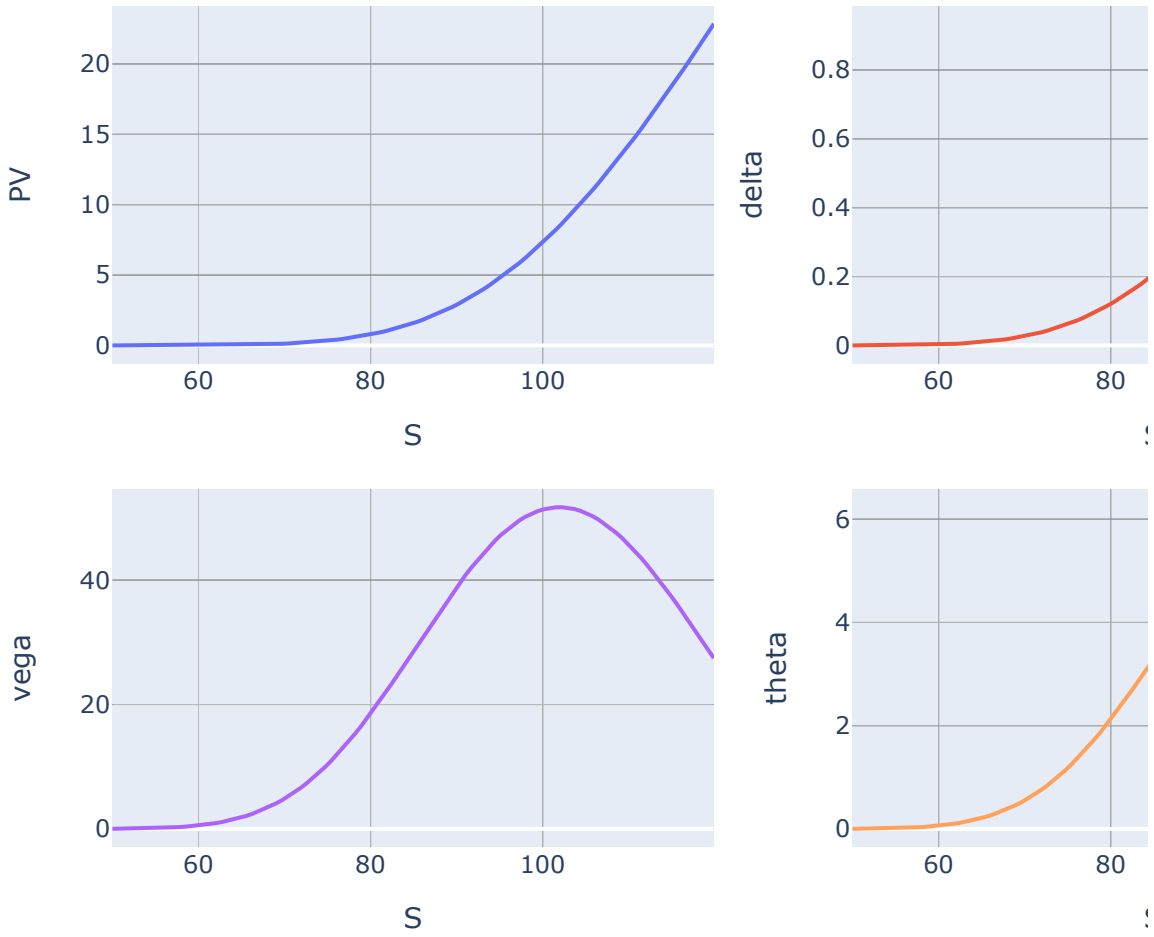
tau	PV	delta	gamma	vega	theta	rho
0.04	1.586310	-0.489007	0.101161	8.016612	-19.727368	-1.748479
0.05	1.738059	-0.487905	0.091968	8.792889	-17.888266	-2.114098
...	...	...	...	...	...	...
0.96	7.365121	-0.447704	0.020646	38.606954	-3.607944	-40.571062
0.97	7.404501	-0.447420	0.020531	38.822345	-3.584699	-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]:

```
changing_param_name='S'
changing_param_bound=(50,119.9)
other_params={
    'S':100,
    'K':100,
    'X':120,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(barrier_upin_call,changing_param_name,changing_param_bound,other_params)
```



Out[25]:

	PV	delta	gamma	vega	theta	rho
S						
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

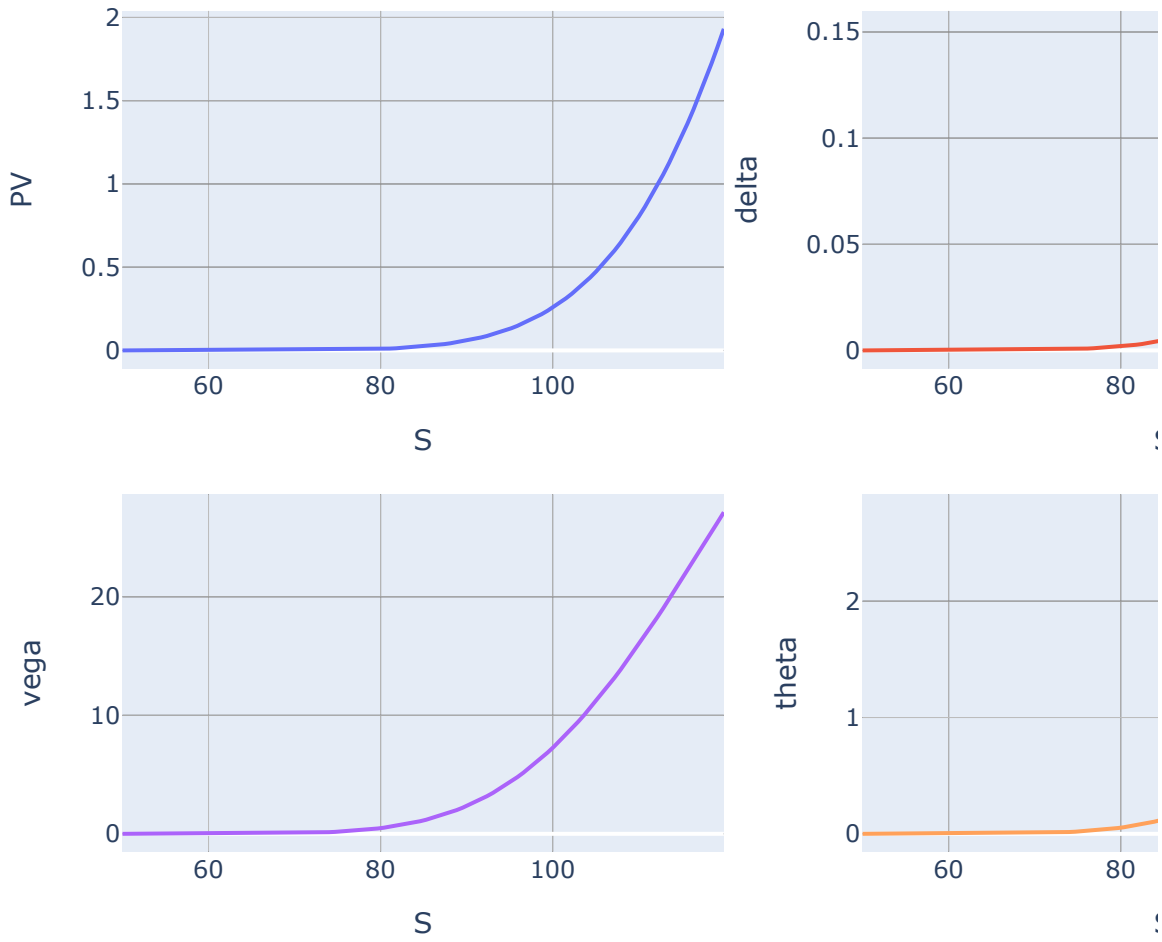
S	PV	delta	gamma	vega	theta	rho
51.412121	0.000363	0.000159	0.000063	0.033513	0.003759	0.007603
52.118182	0.000493	0.000210	0.000079	0.044070	0.004944	0.010154
52.824242	0.000663	0.000275	0.000102	0.057462	0.006449	0.013446
...	...	...	...	...	...	...
117.075758	20.251285	0.904173	0.011798	33.227705	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]:

```
changing_param_name='S'
changing_param_bound=(50,119.9)
other_params={
    'S':100,
    'K':100,
    'X':120,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

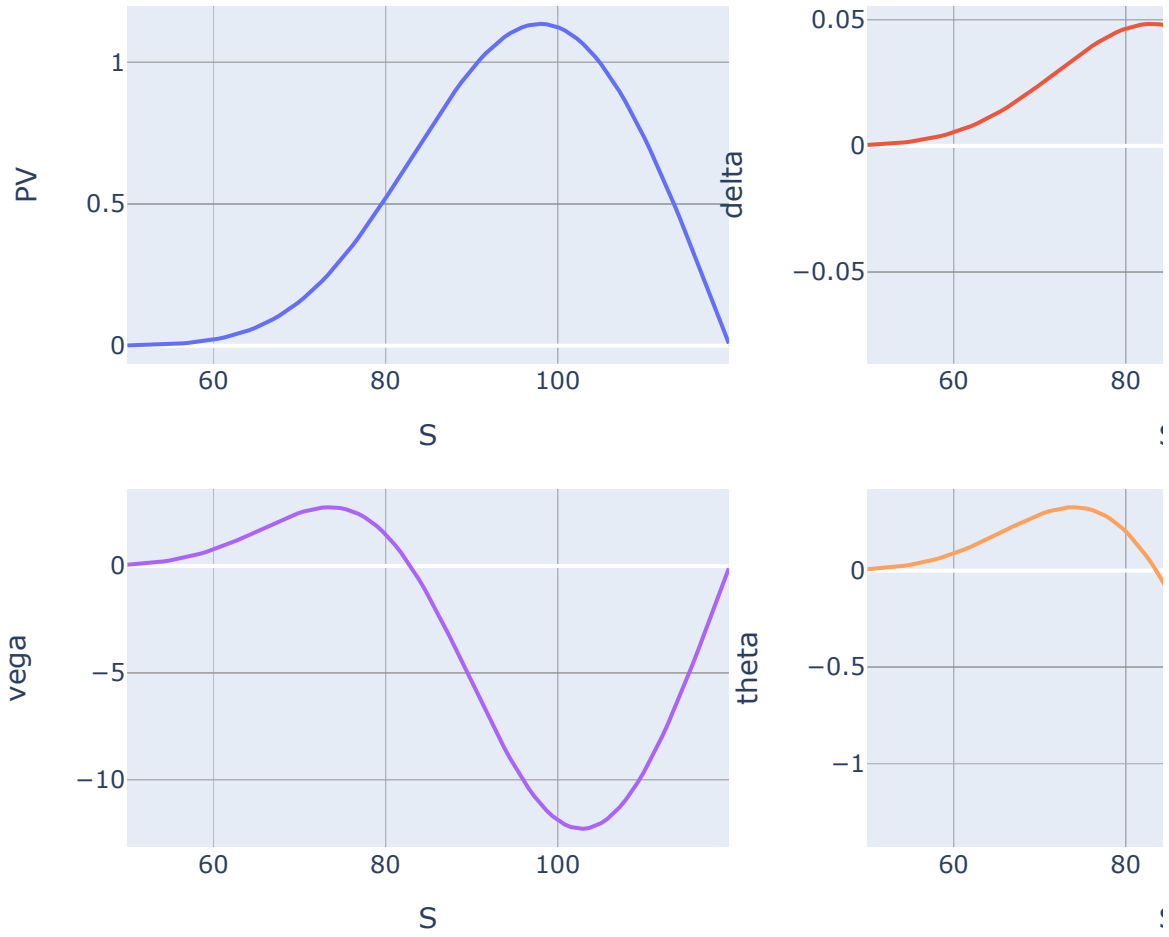
S	PV	delta	gamma	vega	theta	rho
51.412121	3.901974e-07	2.105822e-07	4.102011e-08	0.000057	0.000006	0.000006
52.118182	5.684796e-07	2.991748e-07	-1.132728e-06	0.000080	0.000009	0.000008
52.824242	8.205045e-07	4.211079e-07	4.953140e-07	0.000113	0.000013	0.000011
...	...	...	...	...	...	...
117.075758	1.535885e+00	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]:

```
changing_param_name='S'
changing_param_bound=(50,119.9)
other_params={
    'S':100,
    'K':100,
    'X':120,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(barrier_upout_call,changing_param_name,changing_param_bound,other_params)
```



Out[27]:

	PV	delta	gamma	vega	theta	rho
S						
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

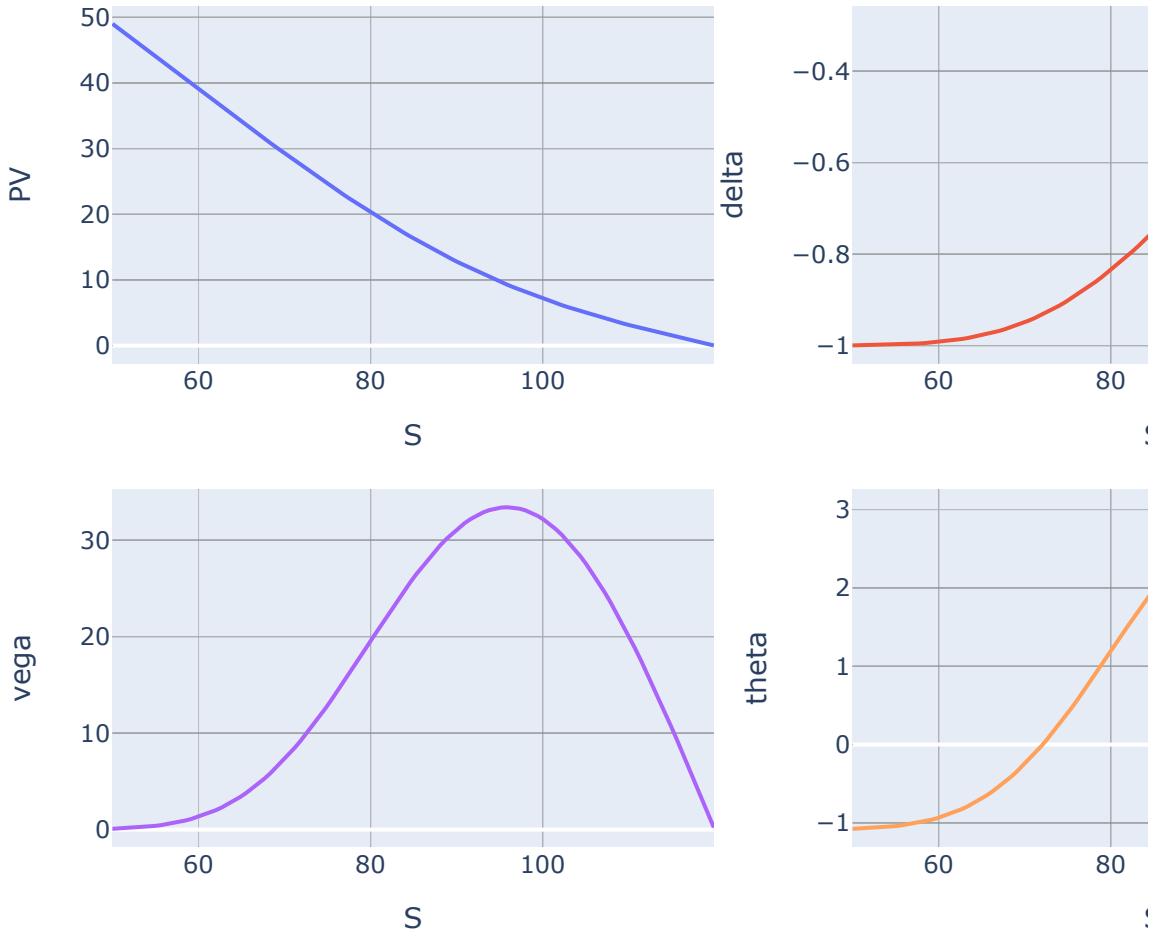
S	PV	delta	gamma	vega	theta	rho
51.412121	0.001613	0.000586	0.000188	0.098603	0.011122	0.028742
52.118182	0.002077	0.000731	0.000226	0.121870	0.013755	0.036314
52.824242	0.002652	0.000904	0.000269	0.149379	0.016870	0.045517
...	...	...	...	...	...	...
117.075758	0.230613	-0.078229	-0.000823	-3.151895	-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]:

```
changing_param_name='S'
changing_param_bound=(50,119.9)
other_params={
    'S':100,
    'K':100,
    'X':120,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(barrier_upout_put,changing_param_name,changing_param_bound,other_params)
```



Out[28]:

	PV	delta	gamma	vega	theta	rho
S						
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

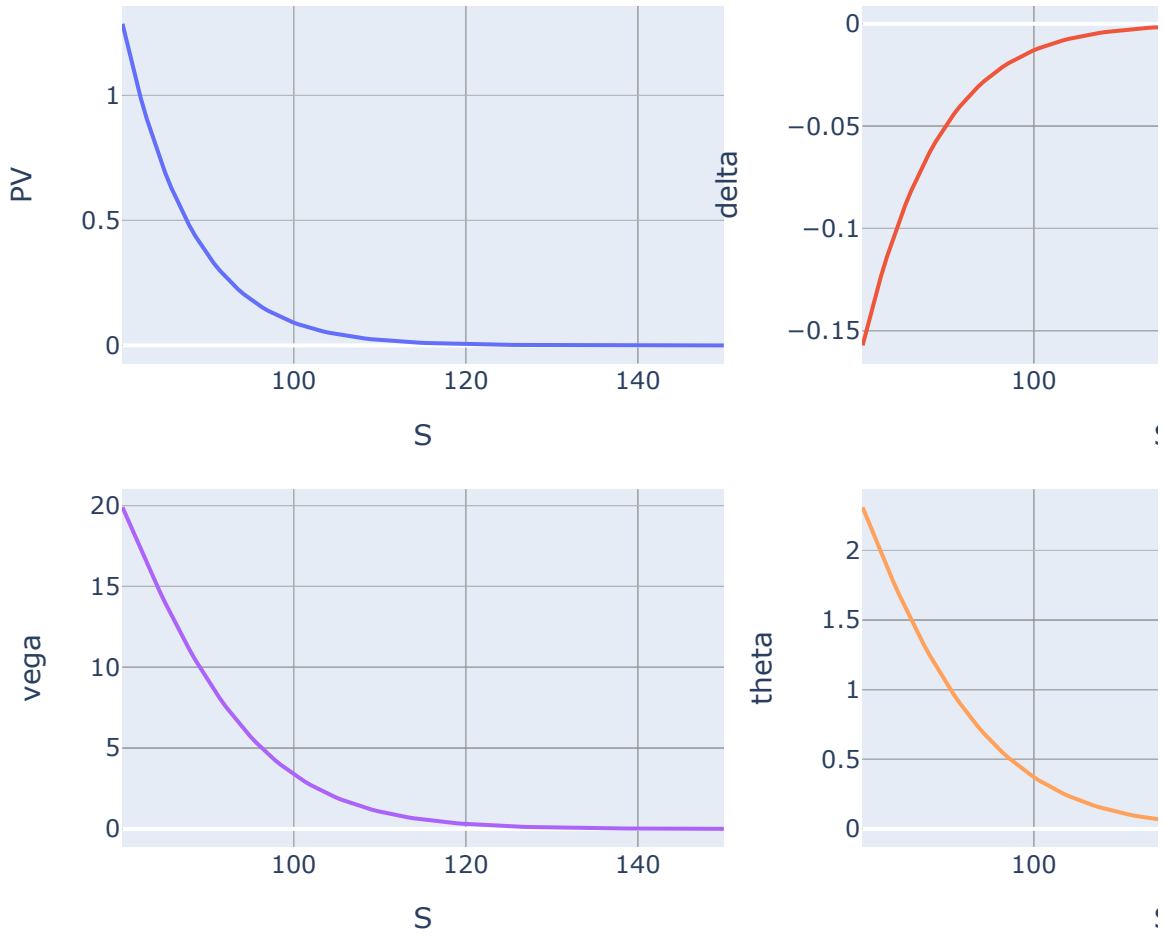
S	PV	delta	gamma	vega	theta	rho
51.412121	47.594838	-0.999255	0.000249	0.132059	-1.070111	-98.968645
52.118182	46.889370	-0.999059	0.000307	0.165860	-1.066296	-98.958523
52.824242	46.184055	-0.998821	0.000371	0.206728	-1.061680	-98.946031
...	...	...	...	...	...	...
117.075758	0.875239	-0.303082	0.003283	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]:

```
changing_param_name='S'
changing_param_bound=(80.1,150)
other_params={
    'S':100,
    'K':100,
    'X':80,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

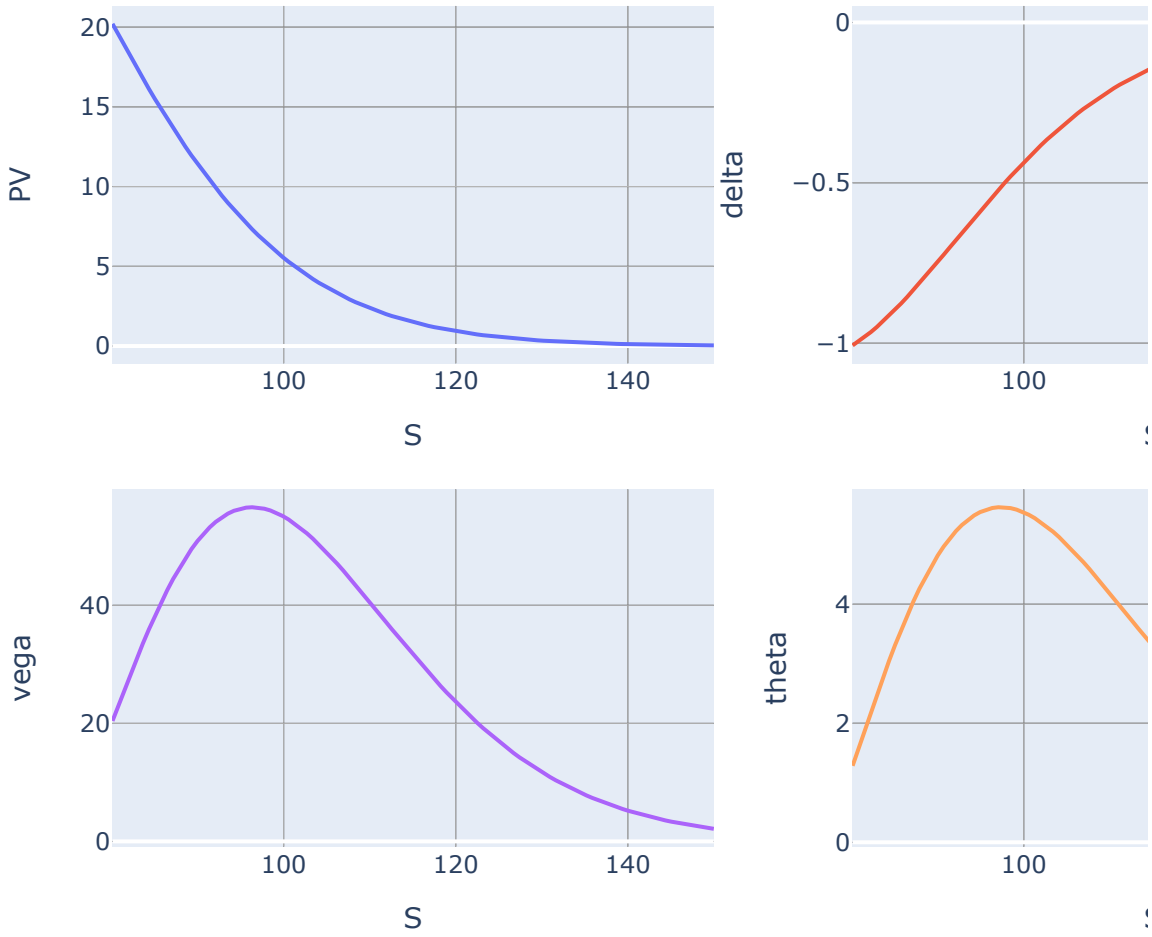
S	PV	delta	gamma	vega	theta	rho
81.512121	1.081154	-0.134082	1.526157e-02	18.141762	2.090856	9.376459
82.218182	0.990200	-0.123676	1.422240e-02	17.280235	1.984855	8.319234
82.924242	0.906340	-0.113985	1.323777e-02	16.433475	1.881678	7.372025
...	...	...	...	...	...	...
147.175758	0.000076	-0.000012	1.855417e-06	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]:

```
changing_param_name='S'
changing_param_bound=(80.1,150)
other_params={
    'S':100,
    'K':100,
    'X':80,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(barrier_downin_put,changing_param_name,changing_param_bound,other_params)
```



Out[30]:

	PV	delta	gamma	vega	theta	rho
S						
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

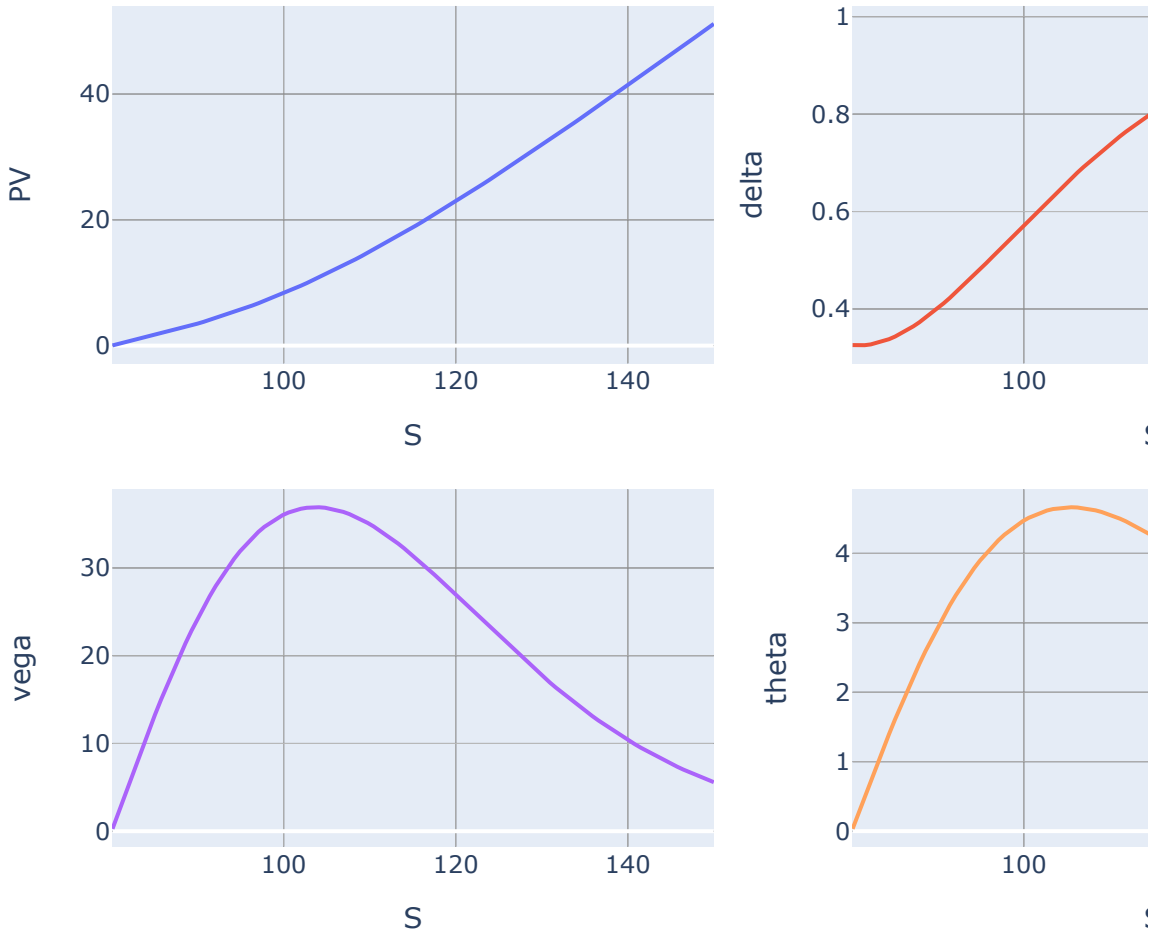
S	PV	delta	gamma	vega	theta	rho
81.512121	18.802571	-0.980528	0.020423	25.797733	1.891983	-85.336060
82.218182	18.115482	-0.965546	0.021997	28.435370	2.190847	-84.443509
82.924242	17.439356	-0.949487	0.023475	31.013118	2.484120	-83.462242
...	...	...	...	...	...	...
147.175758	0.052831	-0.005881	0.000639	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]:

```
changing_param_name='S'
changing_param_bound=(80.1,150)
other_params={
    'S':100,
    'K':100,
    'X':80,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(barrier_downout_call,changing_param_name,changing_param_bound,other_params)
```



Out[31]:

	PV	delta	gamma	vega	theta	rho
S						
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

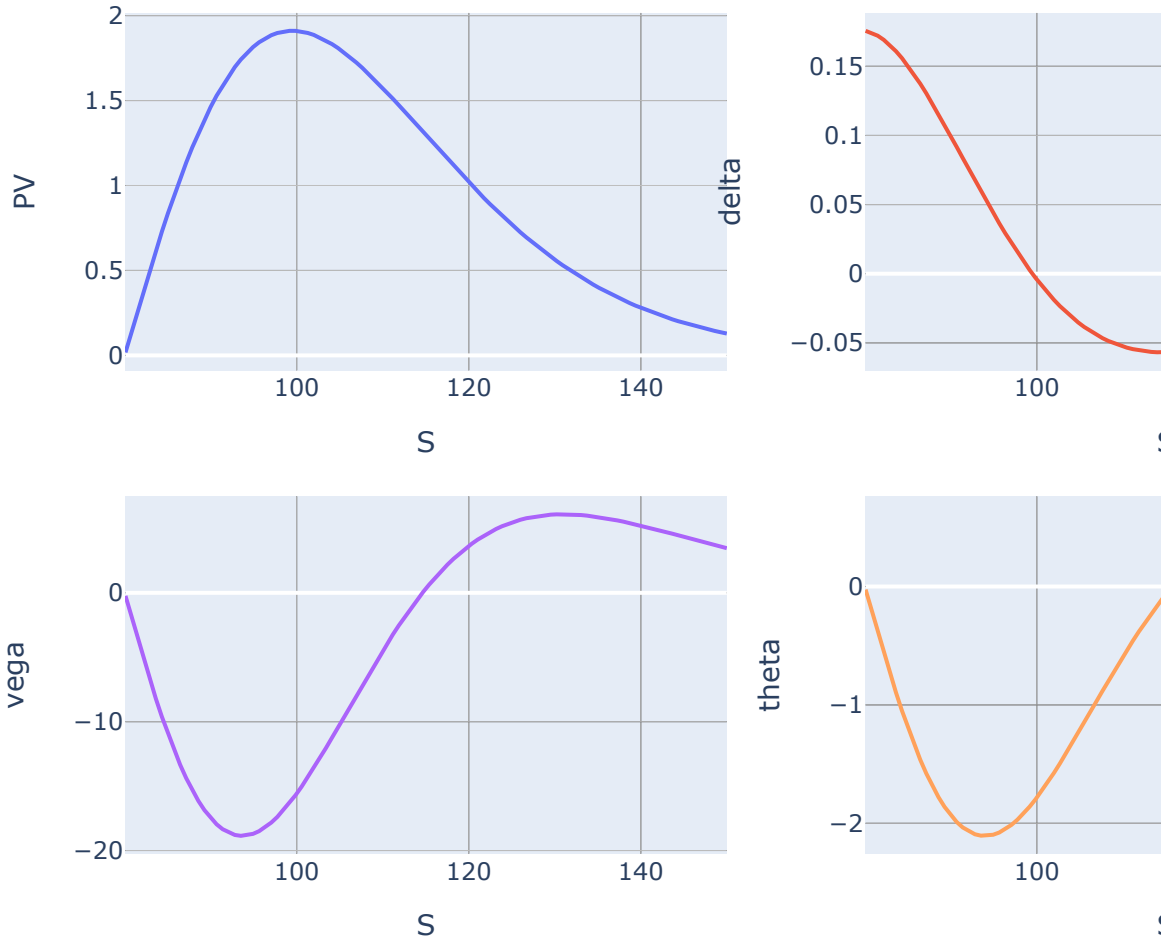
S	PV	delta	gamma	vega	theta	rho
81.512121	0.492062	0.325661	0.001469	4.090415	0.499412	4.666339
82.218182	0.722492	0.327237	0.002984	5.982283	0.729080	6.704498
82.924242	0.954406	0.329856	0.004420	7.853079	0.955623	8.668182
...	...	...	...	...	...	...
147.175758	48.384777	0.981354	0.001550	6.709893	1.787862	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]:

```
changing_param_name='S'
changing_param_bound=(80.1,150)
other_params={
    'S':100,
    'K':100,
    'X':80,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

S	PV	delta	gamma	vega	theta	rho
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

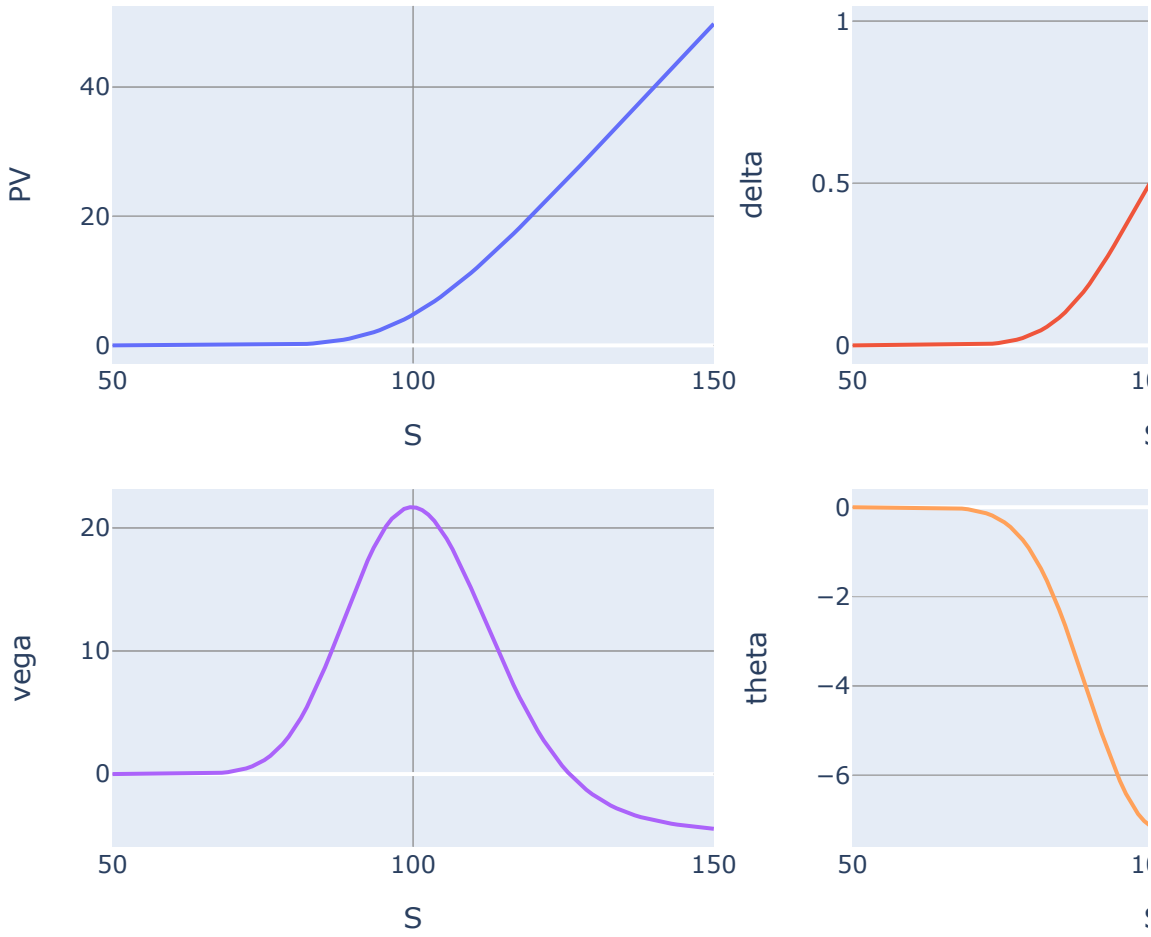
100 rows × 6 columns

6 Asian

6.1 ...Vs S, Geometric call

In [33]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(asian_geometric_call,changing_param_name,changing_param_bound,other_params)
```



Out[33]:

S	PV	delta	gamma	vega	theta	rh
50.000000	3.149653e-09	3.326922e-09	3.361398e-09	5.800775e-07	-1.697019e-07	8.004411e-0
51.010101	8.934602e-09	9.011064e-09	8.677070e-09	1.558130e-06	-4.560672e-07	2.209503e-0

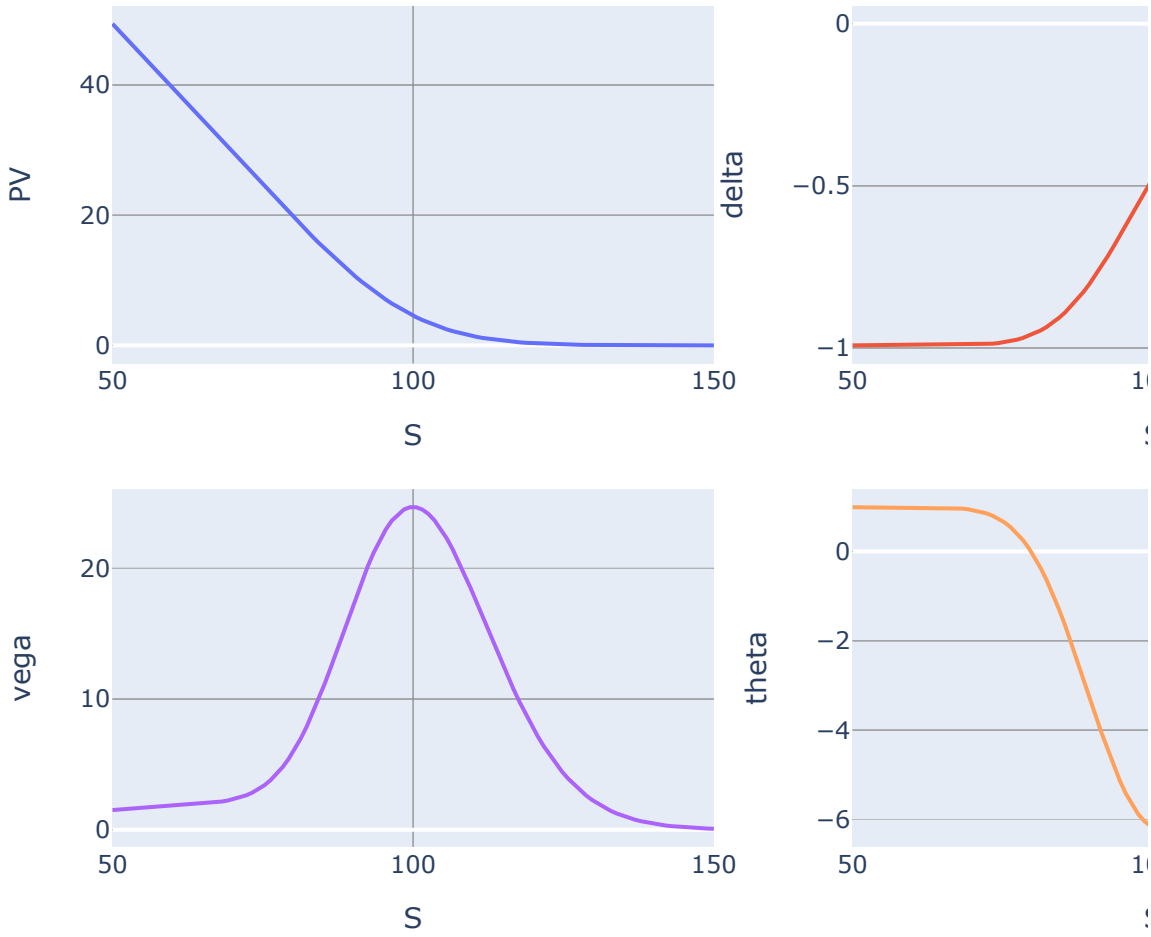
S	PV	delta	gamma	vega	theta	rh
52.020202	2.418190e-08	2.329366e-08	2.138157e-08	3.991997e-06	-1.169089e-06	5.818394e-0
53.030303	6.259536e-08	5.760339e-08	5.041113e-08	9.778347e-06	-2.865258e-06	1.465148e-0
54.040404	1.553083e-07	1.365721e-07	1.139660e-07	2.295002e-05	-6.728696e-06	3.535816e-0
...	...	...	...	...	...	...
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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

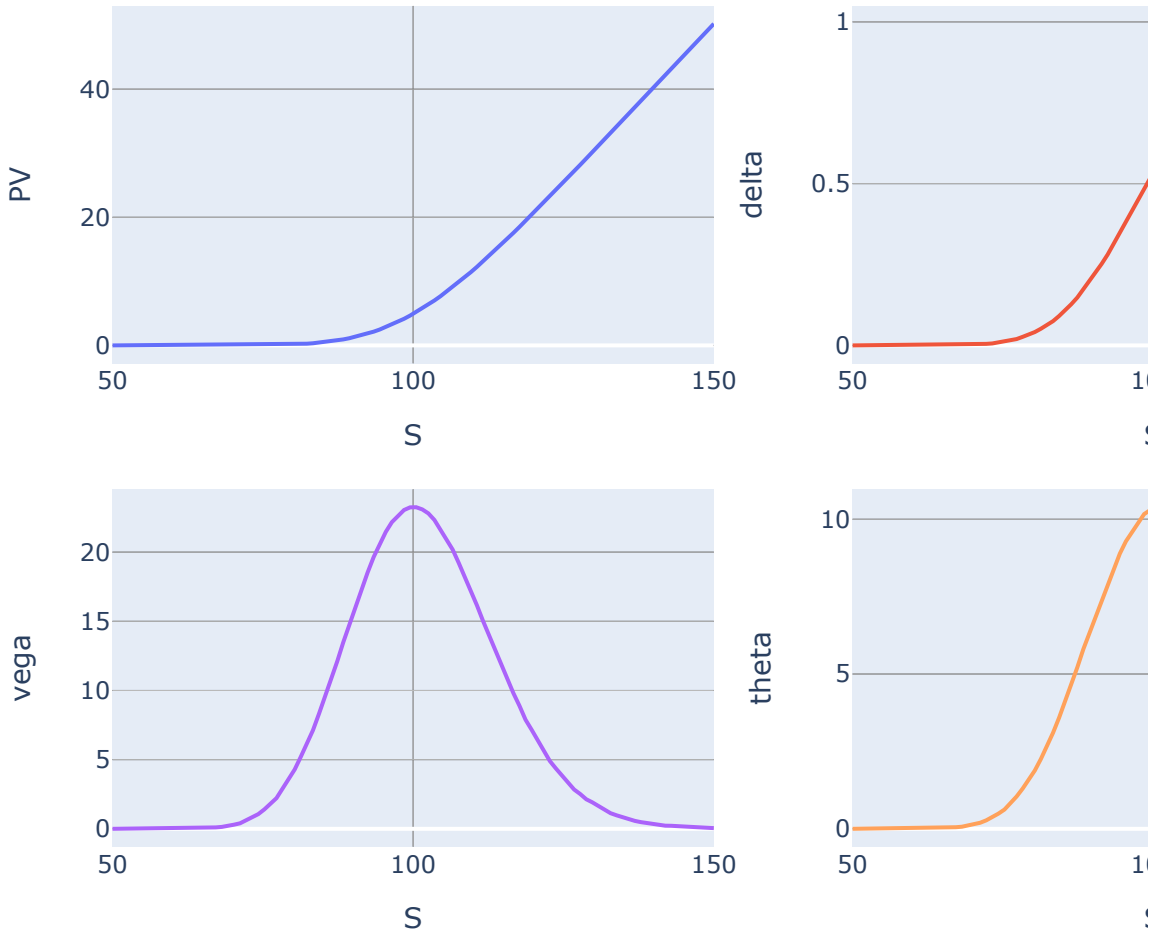
S	PV	delta	gamma	vega	theta	rho
53.030303	46.399243	-0.991994	5.041113e-08	1.598967	0.990047	-72.708663
54.040404	45.397229	-0.991994	1.139660e-07	1.629436	0.990043	-72.207779
...	...	...	...	...	...	...
145.959596	0.002416	-0.000515	1.059034e-04	0.159381	-0.044348	-0.039982
146.969697	0.001947	-0.000418	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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(asian_arithmetic_call,changing_param_name,changing_param_bound,other_params
```



Out[35]:

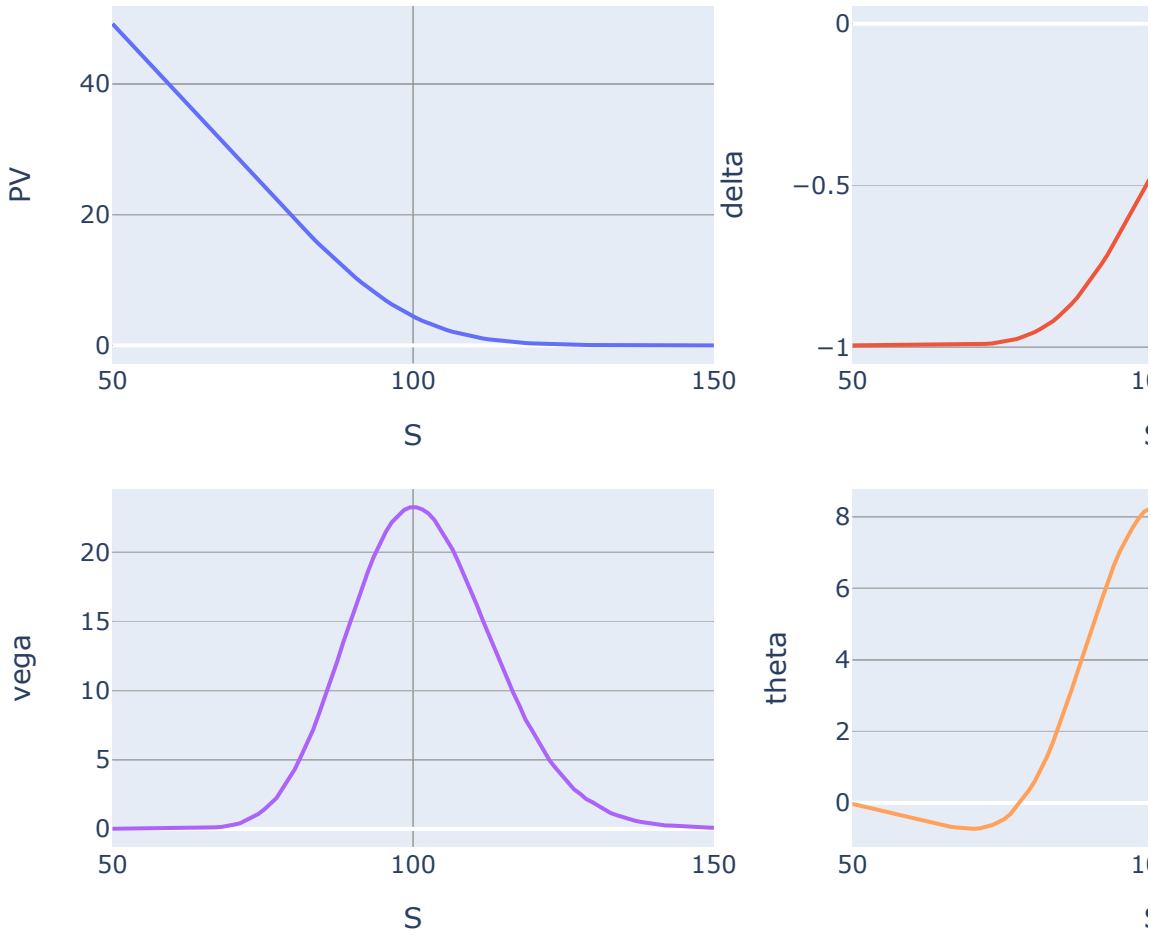
S	PV	delta	gamma	vega	theta	rho
50.000000	3.149653e-09	3.326933e-09	3.361443e-09	5.801049e-07	4.701107e-07	8.004447e-08
51.010101	8.934602e-09	9.011101e-09	8.677492e-09	1.558197e-06	1.208688e-06	2.209514e-07

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-	1.350031e-	4.844707e-	4.170231e+00	2.036792e+01

6.4 ...Vs S, Arithmetic put

In [36]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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

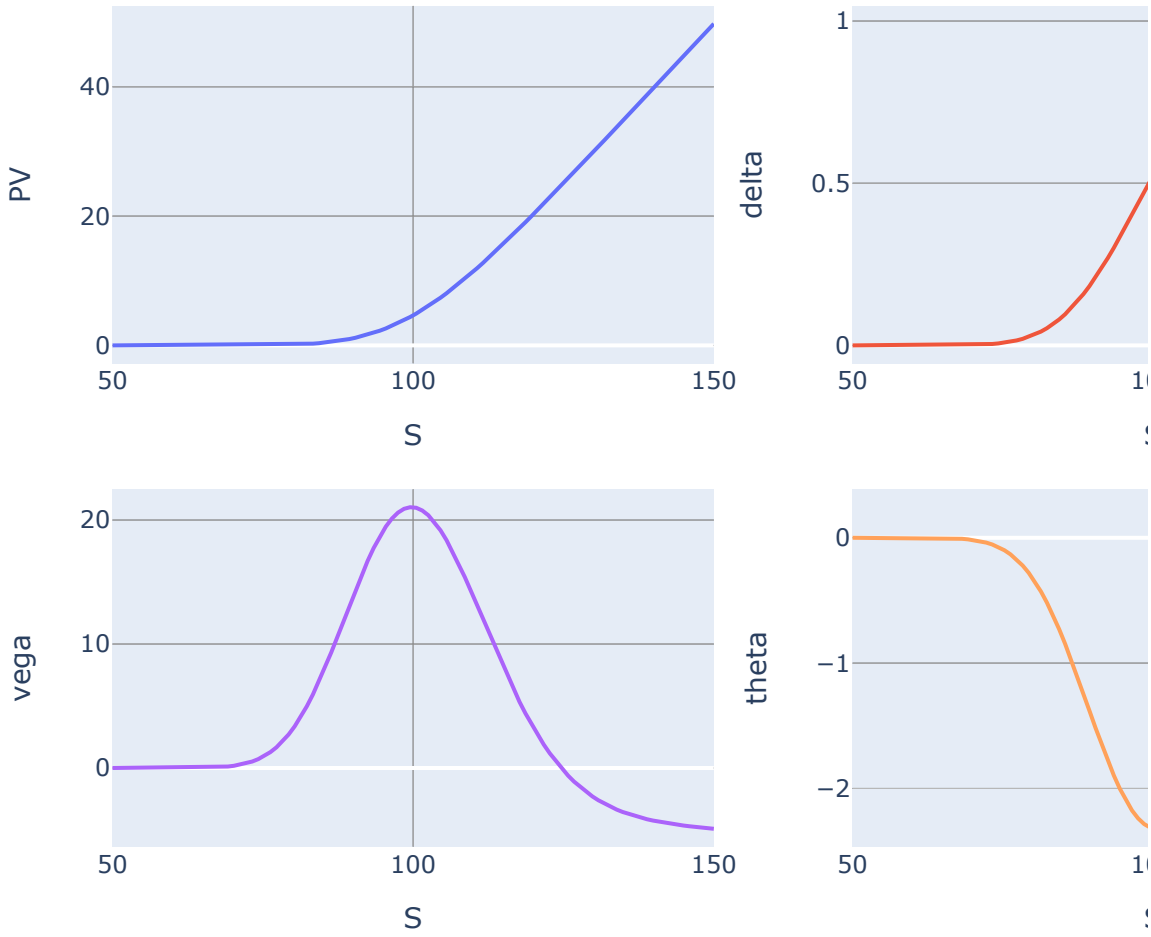
S	PV	delta	gamma	vega	theta	rho
53.030303	46.201689	-0.994184	-0.000001	0.008541	-0.150230	-76.422229
54.040404	45.197462	-0.994184	0.000023	0.008716	-0.191865	-75.838504
...	...	...	...	...	...	...
145.959596	0.001815	-0.000417	0.000108	0.125402	0.057137	-0.030836
146.969697	0.001433	-0.000320	0.000087	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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
option_plot(asian_continuous_geometric_call,changing_param_name,changing_param_bound,ot
```



Out[37]:

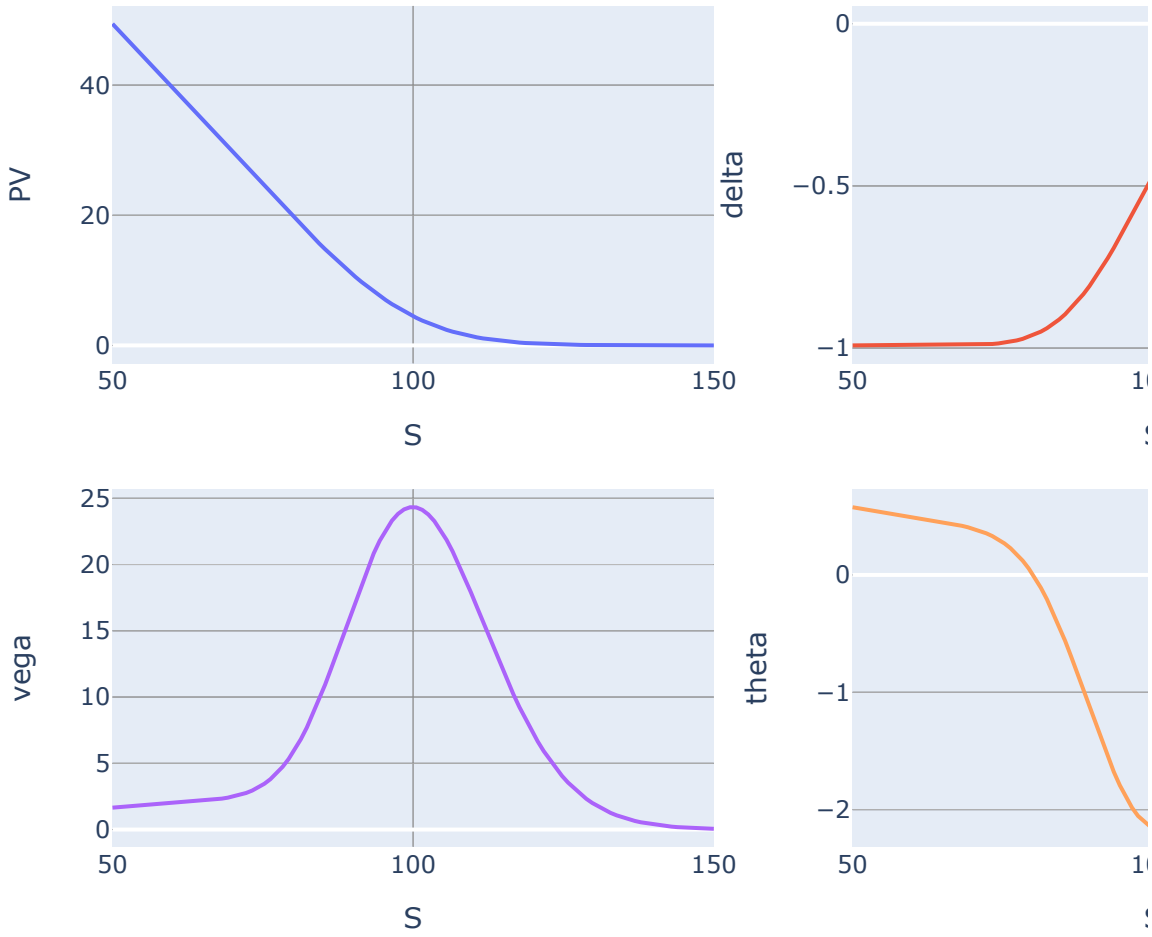
S	PV	delta	gamma	vega	theta	rho
50.000000	1.358844e-09	1.496060e-09	1.578287e-09	2.605543e-07	-2.641586e-08	3.604267e-08
51.010101	4.028282e-09	4.234244e-09	4.257121e-09	7.312776e-07	-7.416743e-08	1.039663e-07

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]:

```
changing_param_name='S'
changing_param_bound=(50,150)
other_params={
    'S':100,
    'K':100,
    'r':0.01,
    'q':0,
    'tau':1,
    'sigma':0.2
    }
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
```