Case Study 3: Computational Methods in Finance

Charles Laferte - cl4249

In [1]:
```
import modulesForCalibration as mfc

import warnings
warnings.filterwarnings("ignore")

import math
import numpy as np
import scipy.integrate as integrate
import pandas as pd

from scipy.optimize import fmin, fmin_bfgs

import cmath
import math

from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
%matplotlib inline

from datetime import datetime
from tqdm import tqdm
from matplotlib import cm
```

Import data

In [2]:
```
'''Standard & Poors 500 Index,Last: 4104.8301,Change:  -4.1099
Date: 12 avril 2023 √† 15:44 UTC,àí4,Bid: 4104.1299,Ask: 4105.5498,Size: 1*1
```

Out[2]:
```
'Standard & Poors 500 Index,Last: 4104.8301,Change:  -4.1099\nDate: 12 avri
l 2023 √† 15:44 UTC,àí4,Bid: 4104.1299,Ask: 4105.5498,Size: 1*1,Volume: 0'
```

Start_date

In [3]:
```
# Set the start date to compute the maturities
date_str = "2023-04-12"
# create a datetime object from the date string
start_date = datetime.strptime(date_str, "%Y-%m-%d")
```

Spot Price

In [4]:
```
S0 = 4104.8301
```

Rates

In [5]:
```
r = 0.0485
```

In [6]:
```
q = 0.0331026
```

Import market data

In [7]:
```
df_price = pd.read_csv("spx_4.csv", index_col=0)
```

In [8]:
```
#Filtrer les SPX... qui ne sont pas des SPXW...
mask = df_price['Calls'].str.contains('SPXW', case=False, na=False)
df_price = df_price[mask]
```

In [9]:
```
df_price.columns
```

Out[9]:
```
Index(['Calls', 'Last Sale', 'Net', 'Bid', 'Ask', 'Volume', 'IV', 'Delta',
       'Gamma', 'Open Interest', 'Strike', 'Puts', 'Last Sale.1', 'Net.1',
       'Bid.1', 'Ask.1', 'Volume.1', 'IV.1', 'Delta.1', 'Gamma.1',
       'Open Interest.1'],
      dtype='object')
```

In [10]:
```
df_price[['Strike']]
```

Out[10]:

	Strike
Expiration Date	
Wed Apr 12 2023	3450
Wed Apr 12 2023	3600
Wed Apr 12 2023	3625
Wed Apr 12 2023	3650
Wed Apr 12 2023	3675
...	...
Fri Jun 30 2023	4700
Fri Jun 30 2023	4750
Fri Jun 30 2023	4800
Fri Jun 30 2023	5000
Fri Jun 30 2023	5025

3248 rows × 1 columns

In [11]:
```
callPrices = df_price[['Strike']] #['Last Sale','Strike']]
# Compute the mid-price
callPrices['Price'] = np.abs(df_price['Bid'].array + df_price['Ask'].array)/
# Convert index to datetime
callPrices.index = pd.to_datetime(callPrices.index)

# Getting the weights inversely proportional to bid-ask spread
callPrices['w'] = np.abs(1/(df_price['Bid'].array - df_price['Ask'].array))

# define a function to compute the difference in days between two dates
def date_diff(date):
    diff = (date - start_date)
    return diff.days

# create a new column in the DataFrame that contains the difference in days
```

```
callPrices['Maturity'] = callPrices.index.to_series().apply(date_diff)
callPrices['Maturity'] = callPrices['Maturity']/252# trading days.../365.25
#callPrices['Strike'] = np.log(callPrices['Strike'].array)

callPrices = callPrices[callPrices['Strike']>=S0]

# drop today
callPrices = callPrices[callPrices['Maturity']!=0]
#579 avec zero mat included
```

In [12]:
```
putPrices = df_price[['Strike']] #['Last Sale','Strike']]
# Compute the mid-price
putPrices['Price'] = np.abs(df_price['Bid.1'].array + df_price['Ask.1'].arra
# Convert index to datetime
putPrices.index = pd.to_datetime(putPrices.index)

# Getting the weights inversely proportional to bid-ask spread
putPrices['w'] = np.abs(1/(df_price['Bid.1'].array - df_price['Ask.1'].array

# define a function to compute the difference in days between two dates
def date_diff(date):
    diff = (date - start_date)
    return diff.days

# create a new column in the DataFrame that contains the difference in days
putPrices['Maturity'] = putPrices.index.to_series().apply(date_diff)
putPrices['Maturity'] = putPrices['Maturity']/365.25
#callPrices['Strike'] = np.log(callPrices['Strike'].array)

putPrices = putPrices[putPrices['Strike']<=S0]

# drop today
putPrices = putPrices[putPrices['Maturity']!=0]
#579 avec zero mat included
```

Visualize market data

In [13]:
```
callPrices
```

Out[13]:

Expiration Date	Strike	Price	w	Maturity
2023-04-13	4105	16.350	10.000000	0.002738
2023-04-13	4110	13.750	10.000000	0.002738
2023-04-13	4115	11.400	5.000000	0.002738
2023-04-13	4120	9.300	5.000000	0.002738
2023-04-13	4125	7.450	10.000000	0.002738
...	...	...	...	...
2023-06-30	4700	1.775	6.666667	0.216290
2023-06-30	4750	1.175	6.666667	0.216290
2023-06-30	4800	0.825	6.666667	0.216290
2023-06-30	5000	0.275	6.666667	0.216290
2023-06-30	5025	0.225	6.666667	0.216290

1069 rows × 4 columns

In [14]:
```
putPrices
```

Out[14]:

Expiration Date	Strike	Price	w	Maturity
2023-04-13	3450	0.025	20.000000	0.002738
2023-04-13	3500	0.025	20.000000	0.002738
2023-04-13	3575	0.025	20.000000	0.002738
2023-04-13	3600	0.025	20.000000	0.002738
2023-04-13	3625	0.025	20.000000	0.002738
...	...	...	...	...
2023-06-30	4075	87.050	1.428571	0.216290
2023-06-30	4080	88.400	1.666667	0.216290
2023-06-30	4085	89.800	1.666667	0.216290
2023-06-30	4090	91.250	1.428571	0.216290
2023-06-30	4100	94.150	1.428571	0.216290

2077 rows × 4 columns

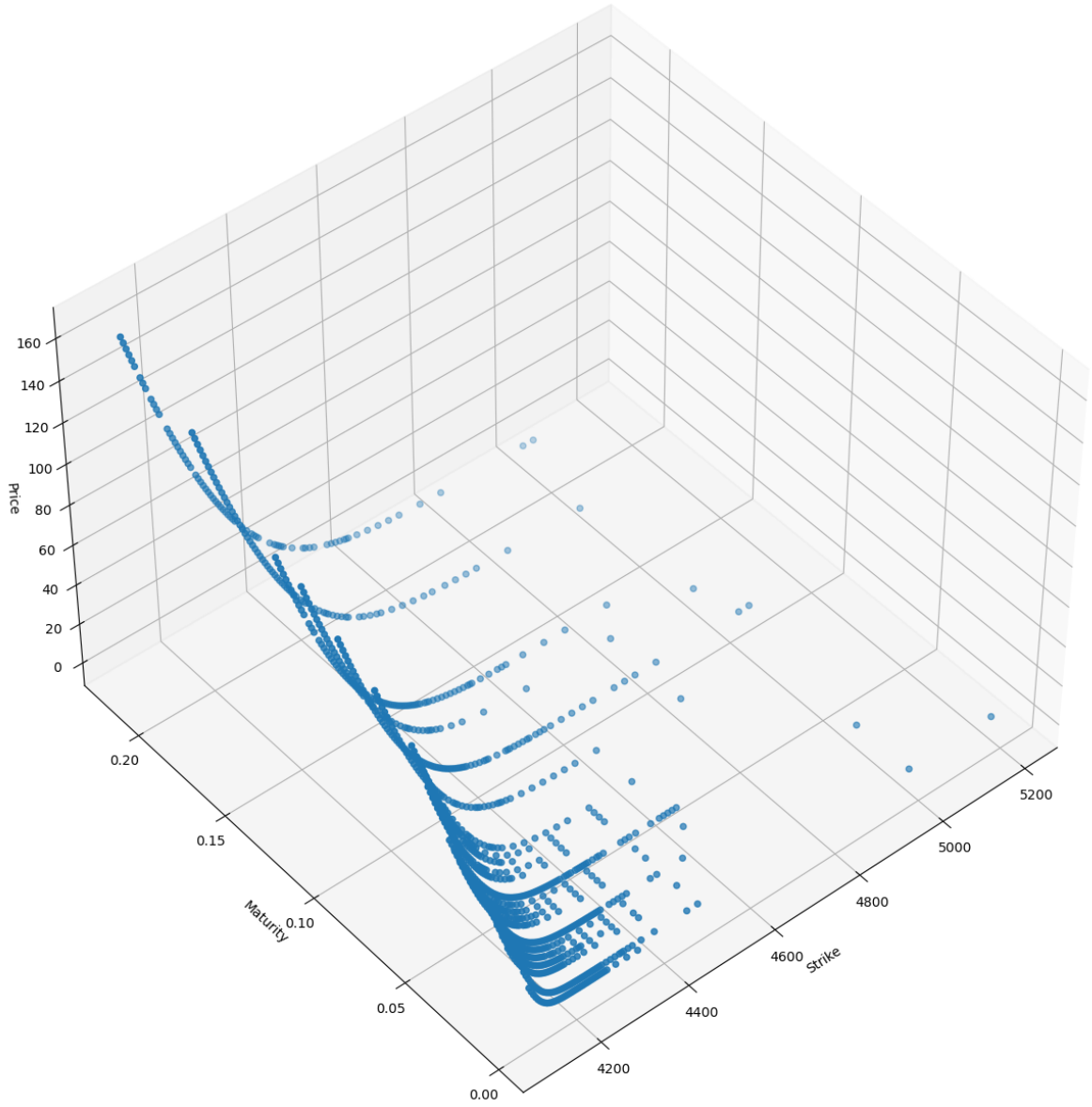
In [15]:
```
callPrices.describe()
```

Out[15]:

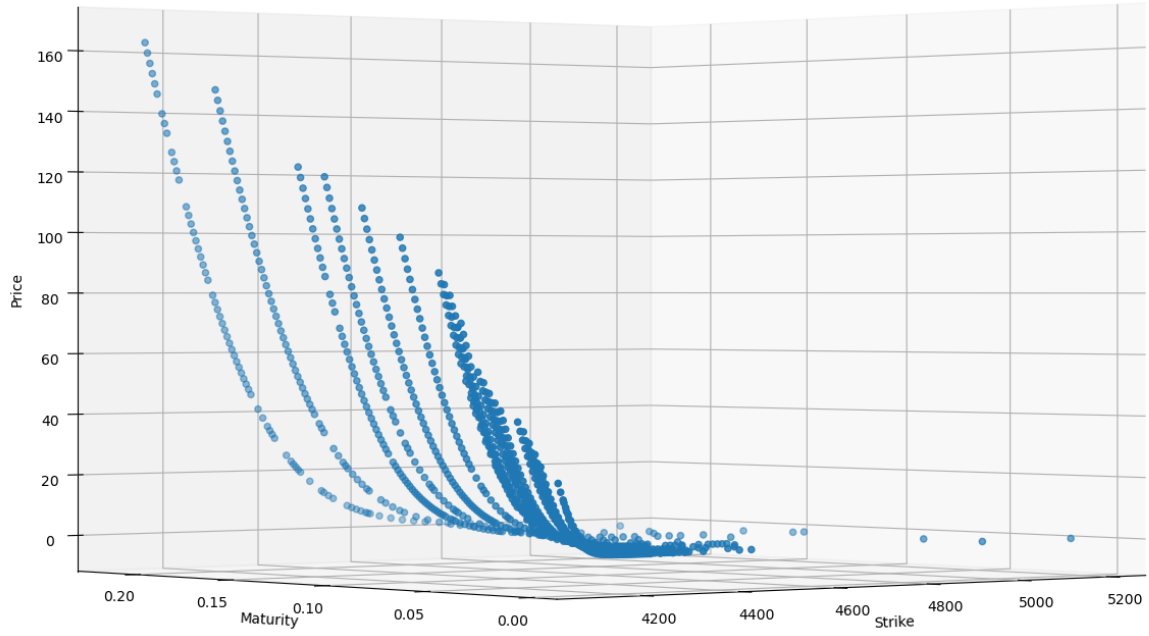
	Strike	Price	w	Maturity
count	1069.000000	1069.000000	1069.000000	1069.000000
mean	4267.970065	26.481805	5.603212	0.075489
std	152.757105	32.132976	4.964135	0.061322
min	4105.000000	0.025000	1.250000	0.002738
25%	4160.000000	1.950000	2.500000	0.024641
50%	4225.000000	13.550000	3.333333	0.054757
75%	4325.000000	40.550000	6.666667	0.120465
max	5200.000000	162.600000	20.000000	0.216290

In [16]:

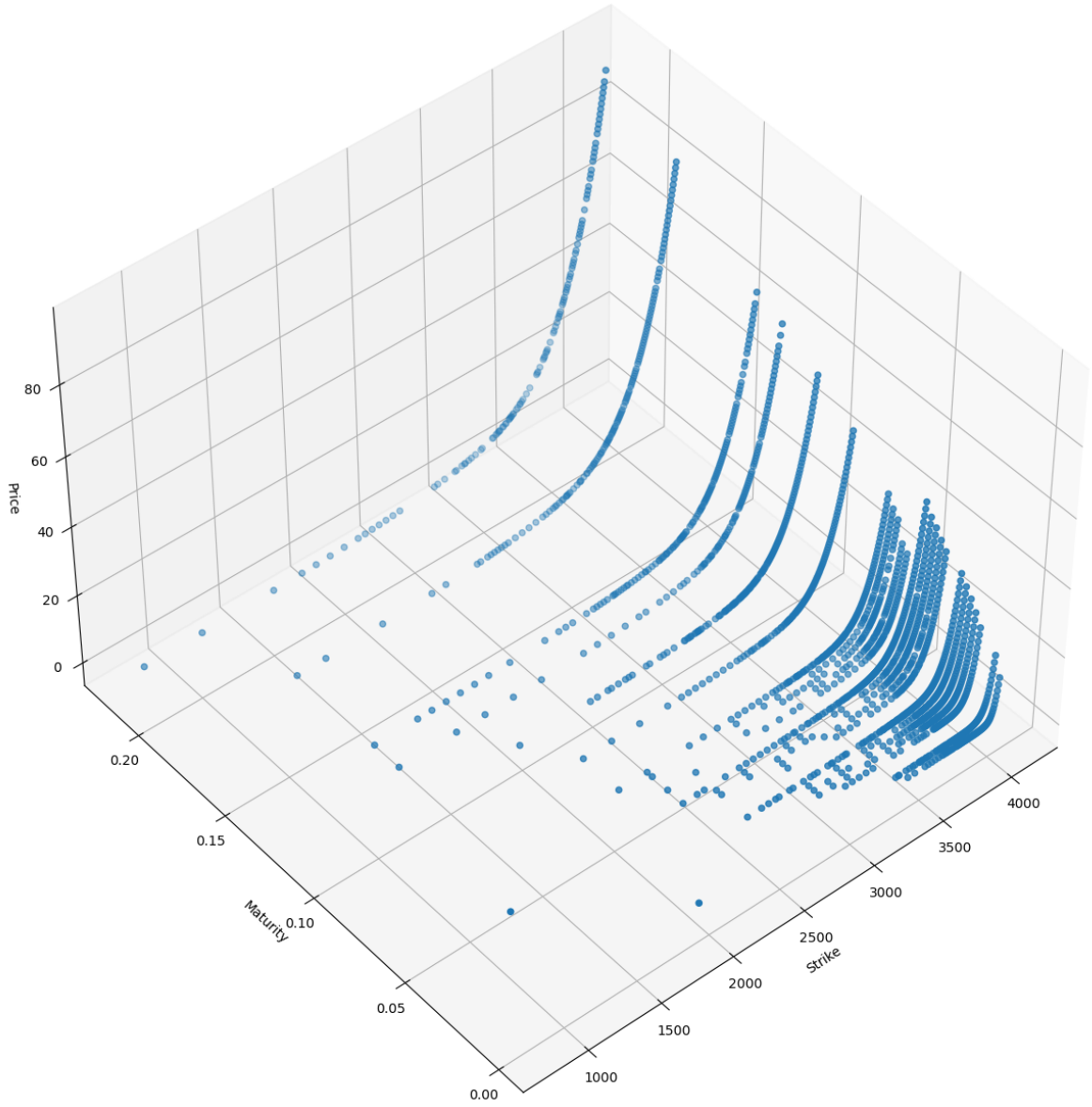
```
# create 3D scatter plot
fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(callPrices['Strike'], callPrices['Maturity'], callPrices['Price']
#ax.plot_surface(callPrices['Strike'], callPrices['Maturity'], callPrices['P
ax.view_init(elev=45, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')

plt.show()
```



In [17]:
```
# create 3D scatter plot
fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(callPrices['Strike'], callPrices['Maturity'], callPrices['Price']
#ax.plot_surface(callPrices['Strike'], callPrices['Maturity'], callPrices['P
ax.view_init(elev=0, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')

plt.show()
```



In [18]:
```
# create 3D scatter plot
fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(putPrices['Strike'], putPrices['Maturity'], putPrices['Price'])#,
#ax.plot_surface(callPrices['Strike'], callPrices['Maturity'], callPrices['P
ax.view_init(elev=45, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')

plt.show()
```



Building the marketprice matrix

In [19]:
```
strikes = pd.Series(callPrices['Strike'].unique()).sort_values().to_list()
maturities = pd.Series(callPrices['Maturity'].unique()).sort_values().to_lis
lenK = len(strikes)
lenT = len(maturities)

strike_m = []

#for i in strikes:
for j in maturities:
        #print(i, j)
        strike_m_temp = callPrices[(callPrices['Maturity']==j)]['Strike'].to
        #marketPrices_temp = callPrices[(callPrices['Maturity']==j)]['Price'

        #marketPrices.append(marketPrices_temp)
        strike_m.append(strike_m_temp)

strikes = set(strike_m[0])
for i in range(1,len(strike_m)):
    strikes = strikes.intersection(set(strike_m[i]))

strikes = list(strikes)
strikes = sorted(strikes)
```

```
print(strikes)

marketPrices = np.zeros((len(strikes), len(maturities)))
w = np.zeros((len(strikes), len(maturities)))
for j in range(len(maturities)):
    for i in range(len(strikes)):
        #print(maturities[j])
        #print(strikes[i])
        #print(callPrices[(callPrices['Maturity']== maturities[j]) & (callPr
        marketPrices[i,j] = callPrices[(callPrices['Maturity']== maturities[
        w[i,j] = callPrices[(callPrices['Maturity']== maturities[j]) & (call

marketPrices = marketPrices.T
w = w.T
```

```
[4105, 4110, 4120, 4125, 4130, 4140, 4150, 4170, 4190, 4200, 4210, 4220, 42
25, 4230, 4250, 4275, 4300]
```

In [20]:

```
strikes_p = pd.Series(putPrices['Strike'].unique()).sort_values().to_list()
maturities_p = pd.Series(putPrices['Maturity'].unique()).sort_values().to_li
lenK_p = len(strikes_p)
lenT_p = len(maturities_p)

strike_m_p = []

#for i in strikes:
for j in maturities_p:
        #print(i, j)
        strike_m_temp_p = putPrices[(putPrices['Maturity']==j)]['Strike'].to
        #marketPrices_temp = callPrices[(callPrices['Maturity']==j)]['Price'

        #marketPrices.append(marketPrices_temp)
        strike_m_p.append(strike_m_temp_p)

strikes_p = set(strike_m_p[0])
for i in range(1,len(strike_m_p)):
    strikes_p = strikes_p.intersection(set(strike_m_p[i]))

strikes_p = list(strikes_p)
strikes_p = sorted(strikes_p)
#print(strikes_p)
#print(maturities_p)
marketPrices_p = np.zeros((len(strikes_p), len(maturities_p)))
w_p = np.zeros((len(strikes_p), len(maturities_p)))
for j in range(len(maturities_p)):
    for i in range(len(strikes_p)):
        #print(maturities_p[j])
        #print(strikes_p[i])
        #print(putPrices[(putPrices['Maturity']== maturities_p[j]) & (putPri
        marketPrices_p[i,j] = putPrices[(putPrices['Maturity']== maturities_
        w_p[i,j] = putPrices[(putPrices['Maturity']== maturities_p[j]) & (pu

marketPrices_p = marketPrices_p.T
w_p = w_p.T
```

1. Heston and VGSA parameters via calibration

In [21]:
```
#Generating Option Price Surface Under A model from the list
def generic_CF(u, params, T, model):

    if (model == 'GBM'):

        sig = params[0];
        mu = np.log(S0) + (r-q-sig**2/2)*T;
        a = sig*np.sqrt(T);
        phi = np.exp(1j*mu*u-(a*u)**2/2);

    elif(model == 'Heston'):

        kappa  = params[0];
        theta  = params[1];
        sigma  = params[2];
        rho    = params[3];
        v0     = params[4];

        tmp = (kappa-1j*rho*sigma*u);
        g = np.sqrt((sigma**2)*(u**2+1j*u)+tmp**2);

        pow1 = 2*kappa*theta/(sigma**2);

        numer1 = (kappa*theta*T*tmp)/(sigma**2) + 1j*u*T*r + 1j*u*math.log(S
        log_denum1 = pow1 * np.log(np.cosh(g*T/2)+(tmp/g)*np.sinh(g*T/2));
        tmp2 = ((u*u+1j*u)*v0)/(g/np.tanh(g*T/2)+tmp);
        log_phi = numer1 - log_denum1 - tmp2;
        phi = np.exp(log_phi);

    elif (model == 'VG'):

        sigma  = params[0];
        nu     = params[1];
        theta  = params[2];

        if (nu == 0):
            mu = math.log(S0) + (r-q - theta -0.5*sigma**2)*T;
            phi  = math.exp(1j*u*mu) * math.exp((1j*theta*u-0.5*sigma**2*u**
        else:
            mu  = math.log(S0) + (r-q + math.log(1-theta*nu-0.5*sigma**2*nu)
            phi = cmath.exp(1j*u*mu)*((1-1j*nu*theta*u+0.5*nu*sigma**2*u**2)

    elif (model == 'VGSA'):

        sigma  = params[0];
        nu     = params[1];
        theta  = params[2];
        kappa  = params[3];
        eta    = params[4];
        lbda   = params[5];

        #cf if mu = 0 ce qu'on fait
        #if (nu == 0):
        psi_VG_i = - np.log(1-theta*nu-sigma**2*nu*0.5)/nu
        psi_VG_u = - np.log(1-1j*u*theta*nu+sigma**2*nu*u**2*0.5)/nu

        tp = 1j*u * (np.log(S0) + (r-q)*T)

        def A(v):
            ga = np.sqrt(kappa**2 - 2*lbda**2*1j*v)
            A_t_u = np.exp(kappa**2*eta*T/(lbda**2))/((np.cosh(ga*T/2)+(kapp
```

```
            return(A_t)
        def B(v):
            ga = np.sqrt(kappa**2 - 2*lbda**2*1j*v)
            B_t = 2*1j*v / (kappa+ ga/np.tanh(ga*T/2))
            return(B_t)

        A_t_u = np.exp(kappa**2*eta*T/(lbda**2))/((np.cosh(ga*T/2)+(kappa/ga
        B_t_u = 2*1j*u / (kappa+ ga/np.tanh(ga*T/2))

        phi_up = A(-1j*psi_VG_u)*np.exp(B(-j*psi_VG_u)/nu)
        phi_down = A(-1j*psi_VG_i)*np.exp(B(-j*psi_VG_i)/nu)

        phi = np.exp(tp)*phi_up/phi_down

    return phi
```

In [22]:

```
def genericFFT(params, T, model):

    # forming vector x and strikes km for m=1,...,N
    km = []
    xX = []

    # discount factor
    df = math.exp(-r*T)

    for j in range(N):

        nuJ=j*eta
        km.append(beta+j*lda)

        psi_nuJ = df*generic_CF(nuJ-(alpha+1)*1j, params, T, model)/((alpha
        if j == 0:
            wJ = (eta/2)
        else:
            wJ = eta

        xX.append(cmath.exp(-1j*beta*nuJ)*psi_nuJ*wJ)

    yY = np.fft.fft(xX)

    cT_km = []
    for i in range(N):
        multiplier = math.exp(-alpha*km[i])/math.pi
        cT_km.append(multiplier*np.real(yY[i]))

    return km, cT_km
```

In [23]:

```
# function for the search:
def myRange(start, finish, increment):
    while (start <= finish):
        yield start
        start += increment

def objFunc(v, x0, x1, x2):
    # Paraboloid centered on (x, y), with scale factors (10, 20) and minimum
    return 10.0*(v[0]-x0)**2 + 20.0*(v[1]-x1)**2 + 30.0*(v[2]-x2)**2 + 40.0
```

Here, for calibration, we will use Gradient-Based routines e.g. the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm.

1.1 Heston model:

Parameters

κ is the mean reversion speed, θ is the long run variance, σ is the volatility of the volatility.

1.1.a Heston with equal weights on Call Prices

In [24]:
```
params = [2.3,     0.046,   0.0825, -0.53,    0.054]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

model ='Heston'

def callbackF(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValue(xi, *arg)))
    num_iter += 1

arg = (marketPrices, maturities, strikes, r, q, S0, alpha, eta, n, model)

num_iter = 1
[xopt, fopt, gopt, Bopt, func_calls, grad_calls, warnflg] = fmin_bfgs(
        mfc.eValue,
        params,
        args=arg,
        fprime=None,
        callback=callbackF,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
26.917603936385817
26.91760393238787
26.917605278349026
26.91760387195445
26.917603944359737
26.917614794097368
12.991818243251787
12.991818272468665
12.991816692672232
12.991818172188792
12.991818252744512
12.991830028417139
50.96669946226243
50.96669953341327
50.96670062854896
50.96669940556747
50.966699471221276
50.96669071849897
106.9613422166266
106.96134225676293
106.96134143365141
106.96134216789483
106.96134222398355
106.96134829815324
60.34533238532081
60.345332400430465
60.34533132890937
60.345332330967906
60.34533239266174
60.345340605117244
36.43331597979236
36.433315994014805
36.43331469956929
36.433315919157266
36.43331598773015
36.433305994419854
12.23468722374694
12.234687172132482
12.23468899417724
12.234687264182828
12.234687219365162
12.234671105503251
6.432774378930434
6.432774403368332
6.432773297313222
6.432774317443811
6.432774387857382
6.4327813583364675

i = 1
x_i = [ 2.30037256 -0.07905618  0.08850429 -0.53074308 -0.95781907]
6.432774378930434
f_i = 6.432774378930434
29.16790021733975
29.16790020467359
29.167901518304856
29.167900152261982
29.167900225954124
29.16788958876774
20.666023633036072
20.666023585072157
```

```
20.66602628964367
20.66602369149957
20.66602362651305
20.666044357417167
5.842614095436365
5.842614111395298
5.842613371002047
5.842614043820542
5.842614103298167
5.842618178222392

i = 2
x_i = [ 2.30034348 -0.07708348  0.08853363 -0.53074815 -0.95906305]
5.842614095436365
f_i = 5.842614095436365
4.775845468717899
4.775845466356098
4.775845627374657
4.775845450257072
4.775845495058575
4.775845478939367

i = 3
x_i = [ 2.29106615 -0.06280154  0.43161502 -0.58817238 -0.95639569]
4.775845468717899
f_i = 4.775845468717899
7.566418741250666
7.56641876571366
7.5664174566304165
7.566418646735742
7.566418736052455
7.566428342876404
4.651739370176704
4.651739362977551
4.651739799142365
4.651739373929344
4.6517393916149326
4.651737679319135

i = 4
x_i = [ 2.29057713 -0.06346969  0.49040055 -0.63313168 -0.95628481]
4.651739370176704
f_i = 4.651739370176704
4.444298924079344
4.444298912466058
4.444299522345551
4.444298947252625
4.444298935457264
4.444295693772086

i = 5
x_i = [ 2.21963045 -0.06728171  0.51036523 -0.76271843 -0.95633924]
4.444298924079344
f_i = 4.444298924079344
4.162435266477409
4.162435257081724
4.162435733137313
4.162435283520728
4.1624352586804125
4.16243308311904
```

```
i = 6
x_i = [ 2.12910914 -0.06818527  0.49309146 -1.05511755 -0.95540201]
4.162435266477409
f_i = 4.162435266477409
4.624874686078102
4.624874718490613
4.624873871253818
4.6248747721712435
4.624874710648465
4.624877774214598
4.108262360871668
4.108262345535541
4.1082628936305055
4.108262414611254
4.108262371134748
4.108258121115195

i = 7
x_i = [ 2.09709667 -0.08300488  0.57574379 -1.03355828 -0.95643369]
4.108262360871668
f_i = 4.108262360871668
4.048197967866347
4.048197961089184
4.048198187242386
4.048197985057176
4.048197965922176
4.048195381404116

i = 8
x_i = [ 2.1253625  -0.08750559  0.57932186 -1.07817874 -0.95677954]
4.048197967866347
f_i = 4.048197967866347
4.046848445988844
4.046848454991841
4.046848132690163
4.046848492097218
4.046848421587724
4.0468502296515
4.01701928472929
4.0170192789828985
4.017019473890098
4.0170192985537225
4.017019281488889
4.017017140689667

i = 9
x_i = [ 2.13425011 -0.08698202  0.59308412 -1.0855581  -0.95646309]
4.01701928472929
f_i = 4.01701928472929
4.005786659528933
4.005786666113015
4.005786427525798
4.0057866940865265
4.005786640380554
4.00578764536261
3.9917270874995534
3.9917270842196855
3.9917271962009258
3.9917270826272286
3.991727081371994
3.9917256068518334
```

```
3.982947570333757
3.982947570988497
3.982947545783287
3.9829475810944466
3.982947558901603
3.9829470552998405

i = 10
x_i = [ 2.16210216 -0.08644165  0.61125804 -1.08505631 -0.9561084 ]
3.982947570333757
f_i = 3.982947570333757
3.942349368779897
3.9423493648979115
3.9423495185221418
3.942349360385283
3.9423493652575505
3.942348390687107

i = 11
x_i = [ 2.21339145 -0.08095323  0.60297347 -1.04386448 -0.95540097]
3.942349368779897
f_i = 3.942349368779897
4.0328420250513215
4.032842032210913
4.032841701763238
4.032842074949518
4.032842047918506
4.03284488930862
3.9264655321297823
3.9264655305637937
3.9264655955509356
3.926465533120142
3.926465526713368
3.9264653337814948

i = 12
x_i = [ 2.24868965 -0.07924007  0.61001085 -1.02006977 -0.9550754 ]
3.9264655321297823
f_i = 3.9264655321297823
3.9491741044585624
3.9491741079745806
3.949173950797592
3.9491741308144532
3.9491741191053484
3.9491753078033636
3.919420169326068
3.9194201679861624
3.919420223004575
3.919420169432114
3.9194201653442717
3.919420015340762

i = 13
x_i = [ 2.28234929 -0.07871365  0.61073604 -1.002988   -0.95503488]
3.919420169326068
f_i = 3.919420169326068
3.9432275557896963
3.943227559189695
3.9432274079759546
3.9432275805105066
3.9432275693663486
```

```
3.943228627039514
3.9187101562100697
3.918710155294222
3.918710192333203
3.918710158161037
3.9187101606883954
3.918710102444895

i = 14
x_i = [ 2.29362188 -0.07861307  0.61110368 -0.99762384 -0.95503307]
3.9187101562100697
f_i = 3.9187101562100697
3.917544727441579
3.9175447266627583
3.917544756579051
3.9175447293170014
3.9175447316328085
3.91754462515195

i = 15
x_i = [ 2.29517747 -0.07913206  0.61087371 -0.9987426  -0.95511229]
3.917544727441579
f_i = 3.917544727441579
3.9175892756402995
3.9175892753150987
3.9175892845539924
3.917589278759959
3.917589271301672
3.917589162095357
3.917008736214747
3.91700873554039
3.917008760151607
3.9170087380550584
3.9170087322193803
3.917008604959326

i = 16
x_i = [ 2.29565147 -0.07950794  0.61049741 -1.00055713 -0.95516313]
3.917008736214747
f_i = 3.917008736214747
3.9165647241942447
3.9165647236795613
3.9165647407959403
3.9165647261826266
3.9165647280313465
3.9165645776135825

i = 17
x_i = [ 2.30487867 -0.07974328  0.60935638 -0.99963877 -0.95521067]
3.9165647241942447
f_i = 3.9165647241942447
3.9161230945668724
3.916123094106447
3.916123108356522
3.9161230962716433
3.9161230981165325
3.9161229278253047

i = 18
x_i = [ 2.31404675 -0.07986113  0.60765162 -0.99989564 -0.95523477]
3.9161230945668724
```

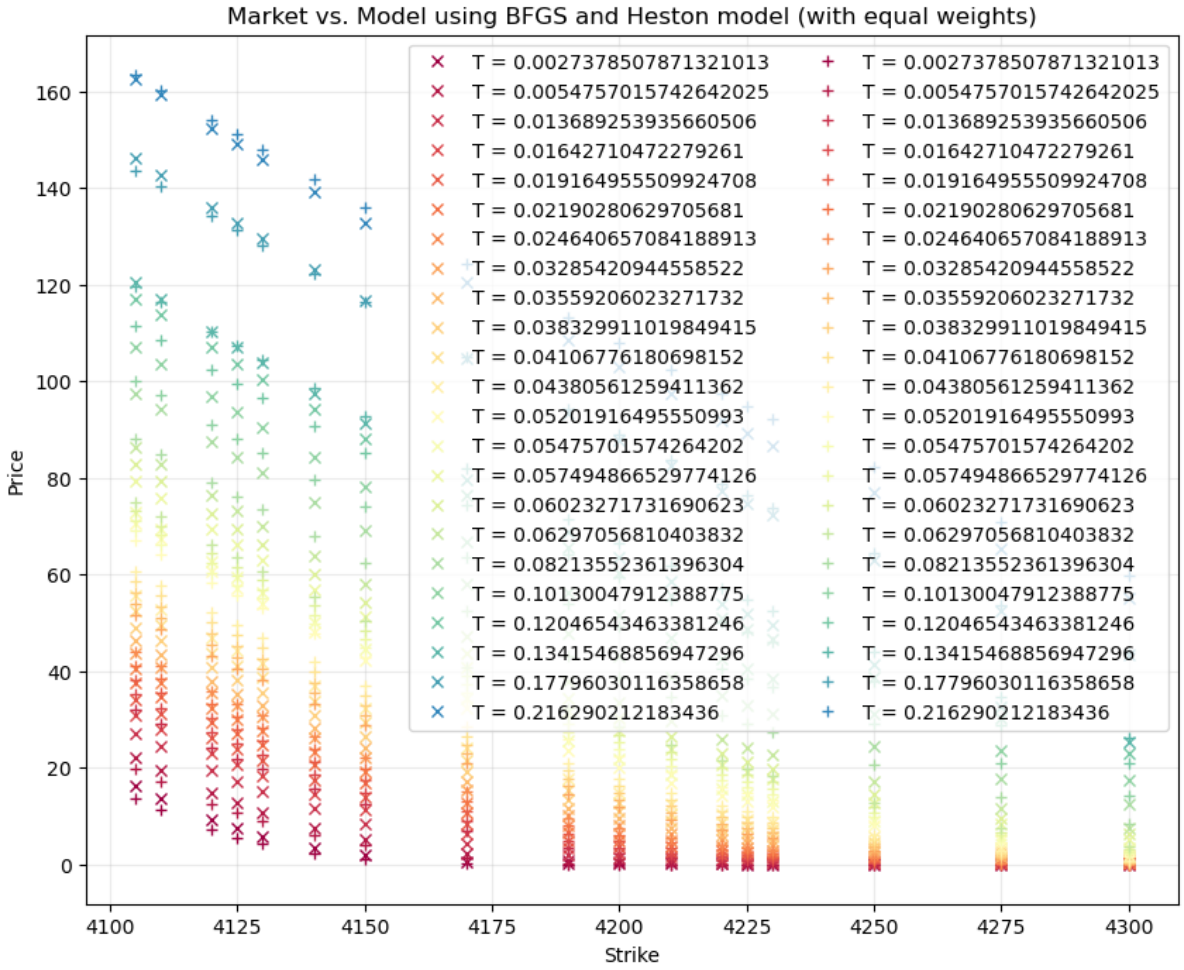
```
f_i = 3.9161230945668724
3.915476474408625
3.9154764740192336
3.915476484756873
3.915476475840516
3.9154764711684162
3.9154763055220396

i = 19
x_i = [ 2.33000744 -0.07983527  0.6046012  -1.00038865 -0.95523867]
3.915476474408625
f_i = 3.915476474408625
3.914488226632618
3.9144882266070264
3.914488221057401
3.914488224238086
3.914488223389572
3.914488139105679

i = 20
x_i = [ 2.35948156 -0.07959283  0.59916615 -1.00043828 -0.95522165]
3.914488226632618
f_i = 3.914488226632618
Warning: Maximum number of iterations has been exceeded.
         Current function value: 3.914488
         Iterations: 20
         Function evaluations: 228
         Gradient evaluations: 38
```

In [25]:
```
xopt
```

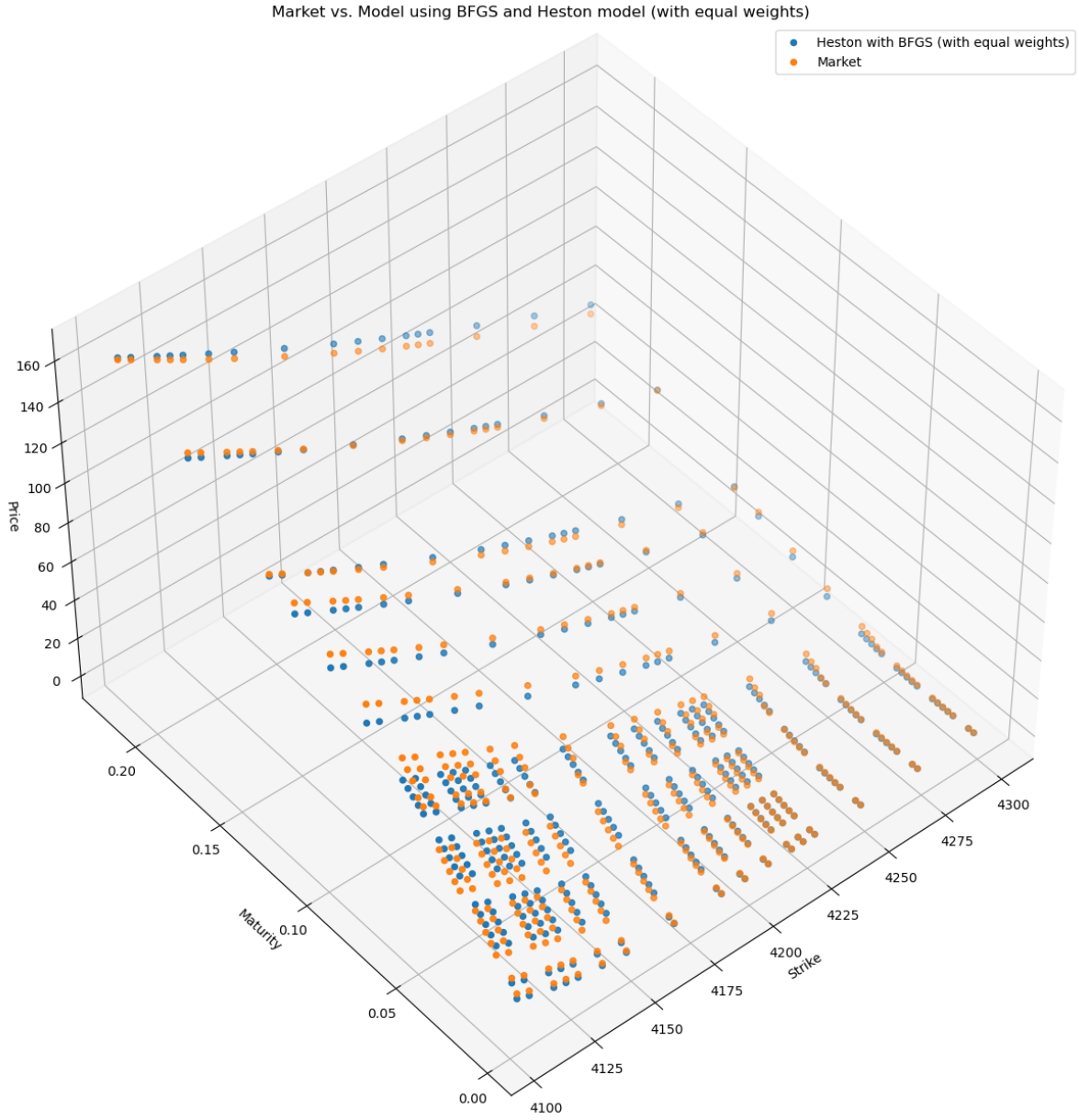
Out[25]:
```
array([ 2.35948156, -0.07959283,  0.59916615, -1.00043828, -0.95522165])
```

In [26]:
```
params2 = xopt
lenT = len(maturities)
lenK = len(strikes)
modelPrices = np.zeros((lenT, lenK))

for i in range(lenT):
    for j in range(lenK):
        T = maturities[i]
        K = strikes[j]
        [km, cT_km] = mfc.genericFFT(params2, S0, K, r, q, T, alpha, eta, n,
        modelPrices[i,j] = cT_km[0]


# plot
fig = plt.figure(figsize=(10,8))
labels = []
colormap = cm.Spectral
# create a list of colors to cycle through
colors = [colormap(i) for i in np.linspace(0, 0.9, len(maturities))]

#plt.gca().set_color_cycle([colormap(i) for i in np.linspace(0, 0.9, len(mat
plt.gca().set_prop_cycle(color=colors)
for i in range(len(maturities)):
    plt.plot(strikes, marketPrices[i,:], 'x')
    labels.append('T = ' + str(maturities[i]))

for i in range(len(maturities)):
    plt.plot(strikes, modelPrices[i,:], '+')
```

```
    labels.append('T = ' + str(maturities[i]))
plt.legend(labels, loc='upper right', ncol=2)
plt.grid(alpha=0.25)
plt.xlabel('Strike')
plt.ylabel('Price')
plt.title('Market vs. Model using BFGS and Heston model (with equal weights)
plt.savefig('MarketvsModel_BFGS.png')
plt.show()
```



In [27]:
```
# create a meshgrid of the maturities and strikes
maturities_, strikes_ = np.meshgrid(maturities, strikes)

maturities_flat = maturities_.flatten()
strikes_flat =  strikes_.flatten()
prices_flat =  marketPrices.flatten()


fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(strikes_flat, maturities_flat, modelPrices.T, label='Heston with
ax.scatter(strikes_, maturities_, marketPrices.T, label='Market')
ax.view_init(elev=45, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')
ax.set_title('Market vs. Model using BFGS and Heston model (with equal weigh
plt.legend()

plt.show()
```



In [ ]:

1.1.b Heston with weights inversely proportional to bid-ask spread on Call prices

In [28]:
```
params = [2.3,     0.046,   0.0825, -0.53,    0.054]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

model ='Heston'

def callbackF_w(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValue_w(xi, *arg)))
    num_iter += 1
```

```
arg = (marketPrices, maturities, strikes, r, q, S0, alpha, eta, n, model, w)

num_iter = 1
[xopt_w, fopt_w, gopt_w, Bopt_w, func_calls_w, grad_calls_w, warnflg_w] = fm
        mfc.eValue_w,
        params,
        args=arg,
        fprime=None,
        callback=callbackF_w,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
20.546802346241865
20.546802343831924
20.546803146890287
20.546802296350254
20.546802352816727
20.54681037116938
6.876877062307984
6.8768770706721964
6.876876298266712
6.876877008109876
6.876877070311694
6.876885285293197
47.40311192366545
47.403111990058726
47.40311256921619
47.40311188245921
47.40311193079521
47.403105982508144
84.52840451960896
84.52840452603287
84.52840406316486
84.52840448190877
84.52840452562069
84.52840902888315
50.095964967886324
50.09596496404388
50.095964365207166
50.09596492574039
50.095964973963135
50.09597088771282
22.26013035452369
22.26013035725458
22.260129551782597
22.26013030495832
22.26013036147372
22.26012247660815
10.780663977683824
10.780663949118116
10.780665438074763
10.780664016429032
10.780663973091626
10.780651161065522
4.031851711046701
4.031851710957969
4.031851726950896
4.0318516807707185
4.031851716428797
4.031853421906533

i = 1
x_i = [ 2.30030394 -0.05497772  0.08879233 -0.53082922 -0.95810324]
4.031851711046701
f_i = 4.031851711046701
89.16112195633876
89.16112198144945
89.16112151416806
89.16112190270229
89.16112197749418
89.1611263026273
13.06760207270274
13.067602064367103
```

```
13.06760024131812
13.06760212338054
13.067602066618225
13.067588522169105
4.018509497078486
4.018509495519067
4.018509624928808
4.018509470816846
4.01850950197544
4.018510357614958

i = 2
x_i = [ 2.30029936 -0.05330676  0.08900891 -0.53087301 -0.95834557]
4.018509497078486
f_i = 4.018509497078486
3.992207102038953
3.992207100501226
3.9922072223866425
3.992207075962471
3.992207107100609
3.9922079056201456
3.900352902273768
3.90035290089691
3.9003529926554554
3.900352877024467
3.900352907940782
3.900353458582342

i = 3
x_i = [ 2.30116308 -0.05991043  0.10689887 -0.53446545 -0.95912994]
3.900352902273768
f_i = 3.900352902273768
3.6666560452630996
3.6666560430530835
3.666656182735026
3.666656025482446
3.666656056644799
3.6666561701489995
3.189665141725024
3.1896651411292916
3.189665181795048
3.1896651634217683
3.189665160461524
3.189665848537285

i = 4
x_i = [ 2.32479002 -0.07178125  0.75757404 -0.73521633 -0.95753116]
3.189665141725024
f_i = 3.189665141725024
4.423179230455004
4.423179251554657
4.423178536384955
4.423179261855304
4.42317920315972
4.423183679867638
2.977694774525486
2.9776947722739657
2.9776948753521917
2.9776947858609324
2.9776947838365095
2.9776941646677493
```

```
i = 5
x_i = [ 2.33632343 -0.07704151  0.7690758  -0.94204659 -0.95807733]
2.977694774525486
f_i = 2.977694774525486
4.611447217436139
4.61144723732958
4.611446386187853
4.611447277414433
4.611447194395415
4.611453566710259
2.96410589226653
2.964105890838401
2.9641059566232415
2.964105906831706
2.9641059017925797
2.9641054957536888

i = 6
x_i = [ 2.34182321 -0.07742333  0.77630266 -0.96760207 -0.95805022]
2.96410589226653
f_i = 2.96410589226653
2.941992644798059
2.941992643158502
2.941992717634146
2.941992657128469
2.9419926365304625
2.9419921584458777

i = 7
x_i = [ 2.34973934 -0.07746277  0.77433504 -1.00332098 -0.95798398]
2.941992644798059
f_i = 2.941992644798059
2.914195620634691
2.914195617654922
2.9141957481876304
2.914195621623542
2.9141956251758976
2.914194553060686

i = 8
x_i = [ 2.40878122 -0.07862463  0.73874494 -0.99357151 -0.95805647]
2.914195620634691
f_i = 2.914195620634691
2.8745924080147764
2.8745924050694547
2.874592532295626
2.874592400518627
2.8745924081126546
2.8745912117198027

i = 9
x_i = [ 2.53758641 -0.07987171  0.68412134 -1.00148737 -0.95786551]
2.8745924080147764
f_i = 2.8745924080147764
2.8416827199282437
2.841682719586443
2.8416827253069297
2.841682718568589
2.8416827202612702
2.8416824120620072
```

```
i = 10
x_i = [ 2.69875181 -0.07958358  0.63361324 -1.00207988 -0.95723085]
2.8416827199282437
f_i = 2.8416827199282437
2.8378446238234214
2.837844623181276
2.837844646541011
2.8378446210324055
2.837844624907233
2.837844504709243

i = 11
x_i = [ 2.74889727 -0.07754139  0.61729626 -1.00252653 -0.95678855]
2.8378446238234214
f_i = 2.8378446238234214
2.8373715128522745
2.837371512824238
2.837371504381784
2.8373715132125112
2.837371512779025
2.8373715641587482

i = 12
x_i = [ 2.76378636 -0.07757421  0.61834979 -1.00340388 -0.95678515]
2.8373715128522745
f_i = 2.8373715128522745
2.8371866025272894
2.837186602473496
2.8371865953512128
2.8371866027913812
2.8371866024734733
2.837186648289471
2.8364663042761564
2.8364663040878924
2.8364663042575615
2.836466303996629
2.836466304351185
2.836466318984772
2.835349590443806
2.835349589282173
2.835349653069673
2.835349585498715
2.835349591855295
2.8353493424029073

i = 13
x_i = [ 3.05598159 -0.07196391  0.61359165 -1.00587337 -0.95679914]
2.835349590443806
f_i = 2.835349590443806
2.8336457152523677
2.8336457147067295
2.833645738275407
2.8336457132619834
2.8336457157642823
2.833645608058919

i = 14
x_i = [ 3.07896972 -0.07231748  0.61324454 -1.00644719 -0.95684847]
2.8336457152523677
f_i = 2.8336457152523677
```

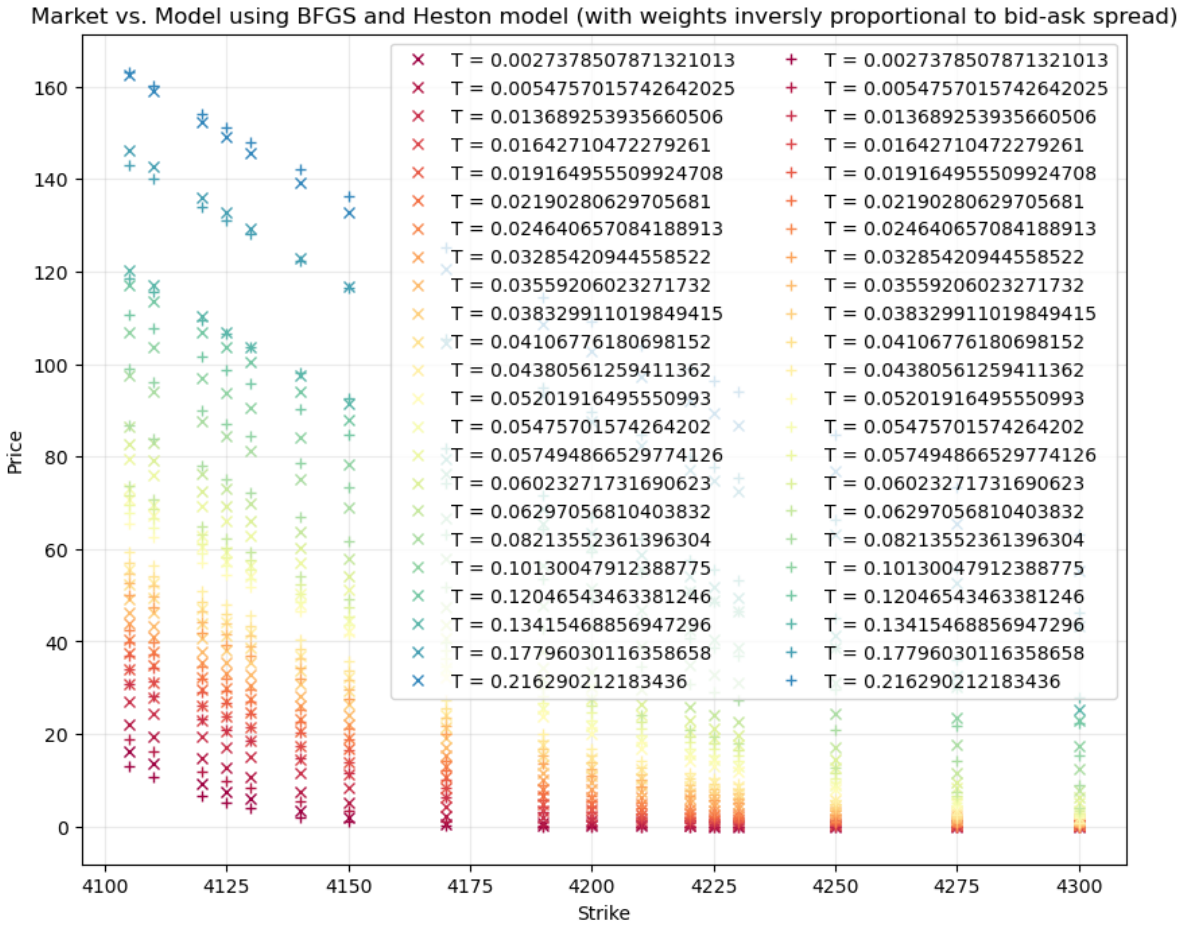
```
2.830755161732768
2.8307551611195025
2.8307551931825805
2.8307551590298323
2.830755162514395
2.8307549850049507

i = 15
x_i = [ 3.34201947 -0.06914397  0.60711131 -1.00958489 -0.95692849]
2.830755161732768
f_i = 2.830755161732768
2.825499868046222
2.8254998678005223
2.8254998798484574
2.825499866481402
2.825499866150773
2.825499877729877

i = 16
x_i = [ 3.87069705 -0.06305475  0.59095929 -1.01662683 -0.95697124]
2.825499868046222
f_i = 2.825499868046222
2.8327221547968837
2.8327221529678797
2.832722362435219
2.8327221639173428
2.832722156323386
2.832721455232906
2.825212496169349
2.825212495689227
2.825212532507558
2.8252124960390885
2.8252124947114527
2.825212406428481

i = 17
x_i = [ 3.93803218 -0.06230056  0.59071898 -1.01689846 -0.95700855]
2.825212496169349
f_i = 2.825212496169349
2.824725422147117
2.8247254217104154
2.8247254560063877
2.824725421470491
2.8247254203345866
2.82472533702968

i = 18
x_i = [ 4.02210027 -0.06154468  0.58867961 -1.01699276 -0.95705137]
2.824725422147117
f_i = 2.824725422147117
2.8238731763856366
2.8238731759900193
2.8238732054933013
2.8238731757566513
2.8238731746998678
2.823873058286859

i = 19
x_i = [ 4.08984552 -0.06122278  0.58836259 -1.01556704 -0.95712648]
2.8238731763856366
f_i = 2.8238731763856366
```

```
2.8223846405665913
2.8223846401736203
2.8223846709081175
2.8223846401155024
2.8223846389906537
2.8223844729802416

i = 20
x_i = [ 4.26196148 -0.05999494  0.58630809 -1.01179205 -0.95722616]
2.8223846405665913
f_i = 2.8223846405665913
Warning: Maximum number of iterations has been exceeded.
         Current function value: 2.822385
         Iterations: 20
         Function evaluations: 216
         Gradient evaluations: 36
```

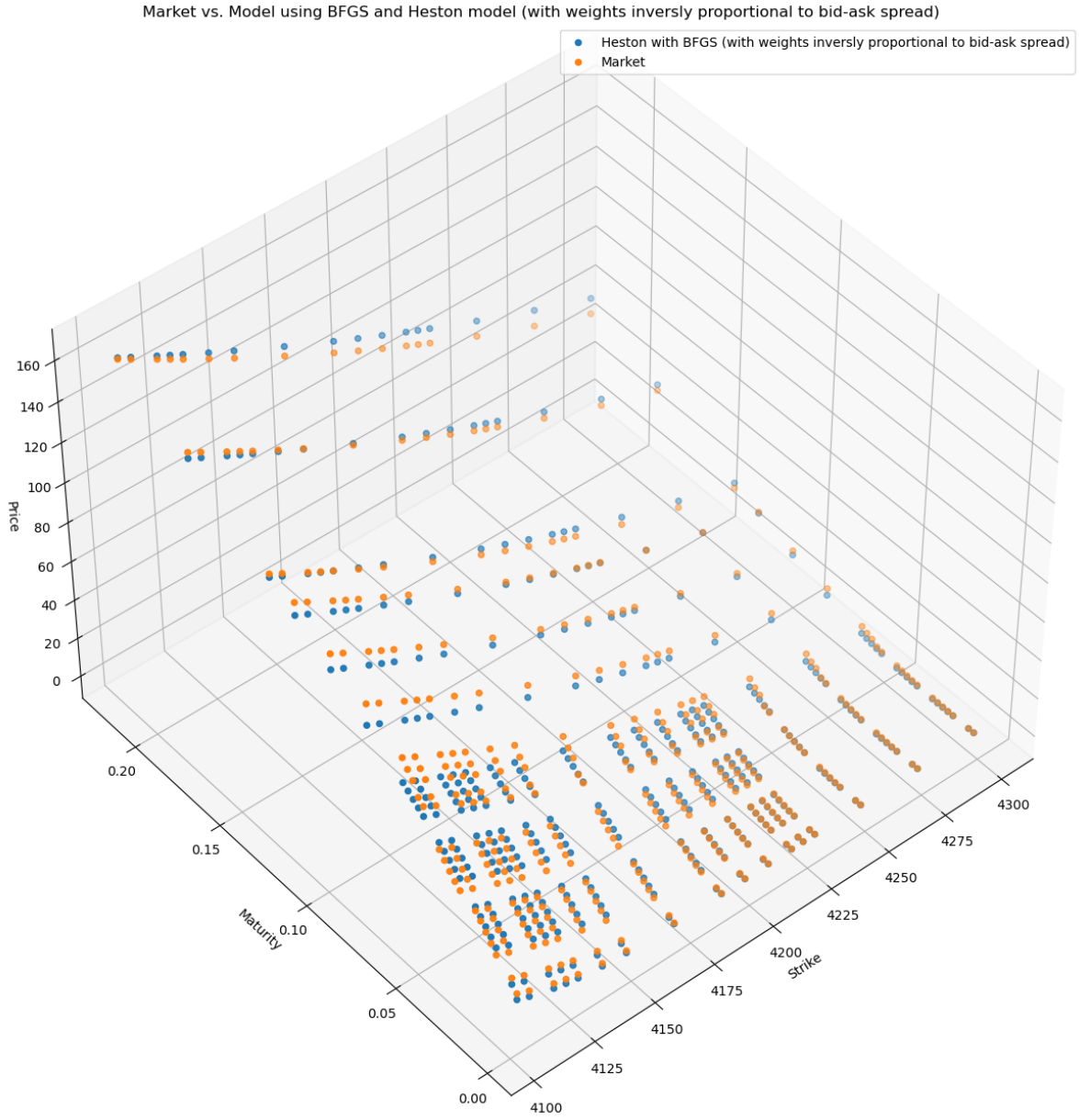
In [29]:
```
xopt_w
```

Out[29]:
```
array([ 4.26196148, -0.05999494,  0.58630809, -1.01179205, -0.95722616])
```

In [30]:
```
params2 = xopt_w
lenT = len(maturities)
lenK = len(strikes)
modelPrices_w = np.zeros((lenT, lenK))

for i in range(lenT):
    for j in range(lenK):
        T = maturities[i]
        K = strikes[j]
        [km, cT_km] = mfc.genericFFT(params2, S0, K, r, q, T, alpha, eta, n,
        modelPrices_w[i,j] = cT_km[0]

# plot
fig = plt.figure(figsize=(10,8))
labels = []
colormap = cm.Spectral
# create a list of colors to cycle through
colors = [colormap(i) for i in np.linspace(0, 0.9, len(maturities))]

#plt.gca().set_color_cycle([colormap(i) for i in np.linspace(0, 0.9, len(mat
plt.gca().set_prop_cycle(color=colors)
for i in range(len(maturities)):
    plt.plot(strikes, marketPrices[i,:], 'x')
    labels.append('T = ' + str(maturities[i]))

for i in range(len(maturities)):
    plt.plot(strikes, modelPrices_w[i,:], '+')
    labels.append('T = ' + str(maturities[i]))
plt.legend(labels, loc='upper right', ncol=2)
plt.grid(alpha=0.25)
plt.xlabel('Strike')
plt.ylabel('Price')
plt.title('Market vs. Model using BFGS and Heston model (with weights invers
plt.savefig('MarketvsModel_BFGS.png')
plt.show()
```



In [31]:
```
# create a meshgrid of the maturities and strikes
maturities_, strikes_ = np.meshgrid(maturities, strikes)

maturities_flat = maturities_.flatten()
strikes_flat =  strikes_.flatten()
prices_flat =  marketPrices.flatten()


fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(strikes_flat, maturities_flat, modelPrices_w.T, label='Heston wit
ax.scatter(strikes_, maturities_, marketPrices.T, label='Market')
ax.view_init(elev=45, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')
ax.set_title('Market vs. Model using BFGS and Heston model (with weights inv

plt.legend()

plt.show()
```



1.1.c Heston with equal weights on Call & Put prices

In [32]:
```
params = [ 2.34548424, 0.06375792, 0.59865706, -0.43626748, -0.95937438]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

r = 0.0485
q = 0.03310264838967327

model ='Heston'

def callbackF_full(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValuefull(xi, *arg)))
    num_iter += 1

arg = (marketPrices,marketPrices_p,maturities,maturities_p,strikes,strikes_p
```

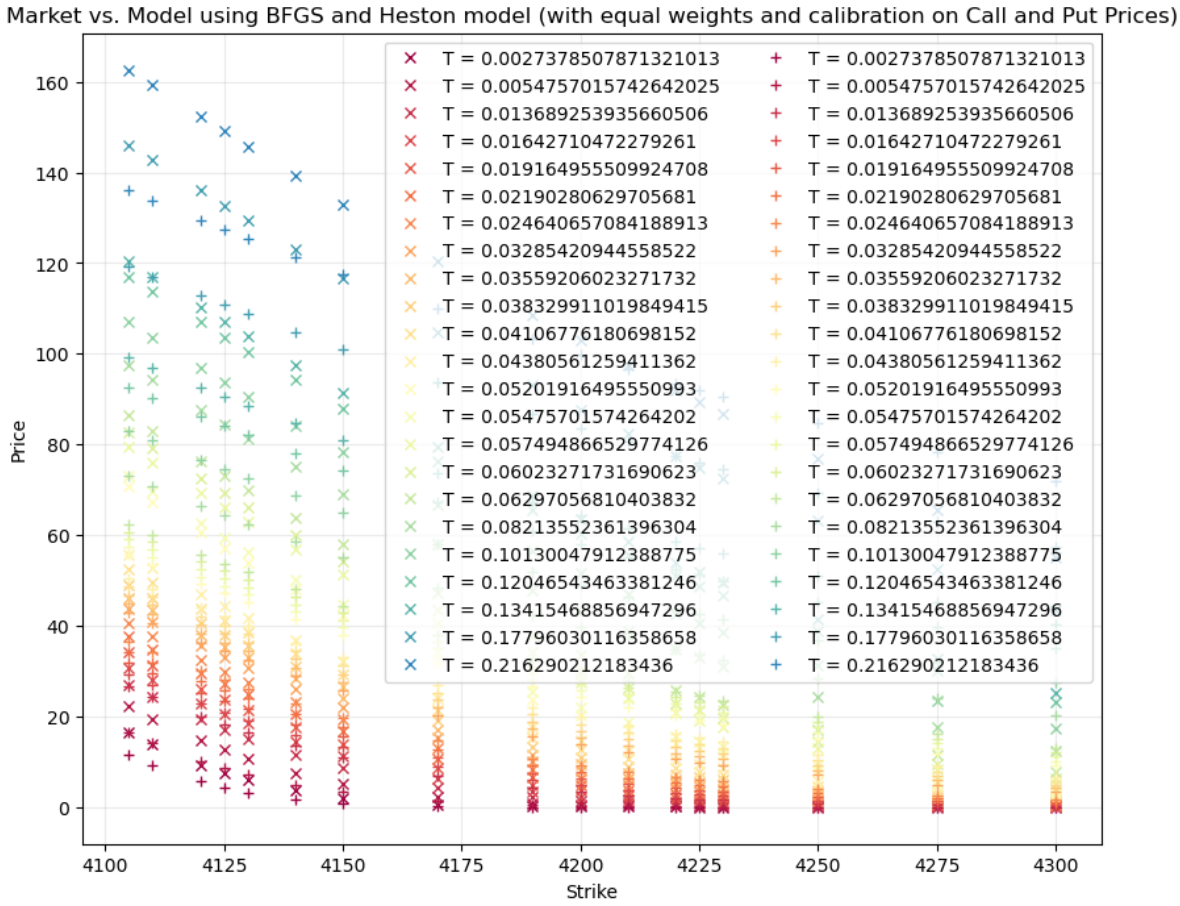
```
num_iter = 1
[xopt_pc, fopt_pc, gopt_pc, Bopt_pc, func_calls_pc, grad_calls_pc, warnflg_p
        mfc.eValuefull,
        params,
        args=arg,
        fprime=None,
        callback=callbackF_full,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
i = 1
x_i = [ 2.34547637  0.06327202  0.59865338 -0.43623731 -0.96281965]
f_i = 10.664129184177813

i = 2
x_i = [ 2.34561038  0.05224145  0.58476893 -0.41892556 -0.96145466]
f_i = 10.581207242590827

i = 3
x_i = [ 2.34860803  0.04854858  0.46347785 -0.27171281 -0.96200312]
f_i = 9.856611114317921

i = 4
x_i = [ 2.30236594  0.04025805  0.20175872  0.11680907 -0.9624521 ]
f_i = 9.01451520755

i = 5
x_i = [ 2.18171845  0.06120302  0.60523637  0.88626161 -0.96215728]
f_i = 8.219633646350761

i = 6
x_i = [ 2.18295688  0.06308904  0.63660412  0.91057553 -0.96222963]
f_i = 8.196553166805044

i = 7
x_i = [ 2.18309035  0.06448111  0.63468301  0.86719437 -0.96255072]
f_i = 8.160828737178342

i = 8
x_i = [ 2.17446657  0.06709596  0.6463773   0.83203293 -0.96309211]
f_i = 8.142135808656153

i = 9
x_i = [ 2.16935444  0.06628563  0.65413944  0.81862544 -0.96302117]
f_i = 8.140135440610125

i = 10
x_i = [ 2.15342294  0.06605197  0.66724578  0.81997676 -0.96306048]
f_i = 8.138684389350624

i = 11
x_i = [ 2.11686215  0.06532224  0.69431199  0.83361468 -0.96307657]
f_i = 8.136692819201063

i = 12
x_i = [ 2.08696977  0.06485728  0.71193002  0.85096872 -0.96306508]
f_i = 8.135705693605894

i = 13
x_i = [ 2.06744259  0.06476803  0.71928517  0.86490251 -0.96304512]
f_i = 8.135304405023856

i = 14
x_i = [ 2.05842306  0.06489913  0.72076533  0.8689683  -0.96304095]
f_i = 8.135253782007444

i = 15
x_i = [ 2.04669833  0.06512269  0.72165631  0.87102911 -0.96303918]
f_i = 8.135212312631193
```

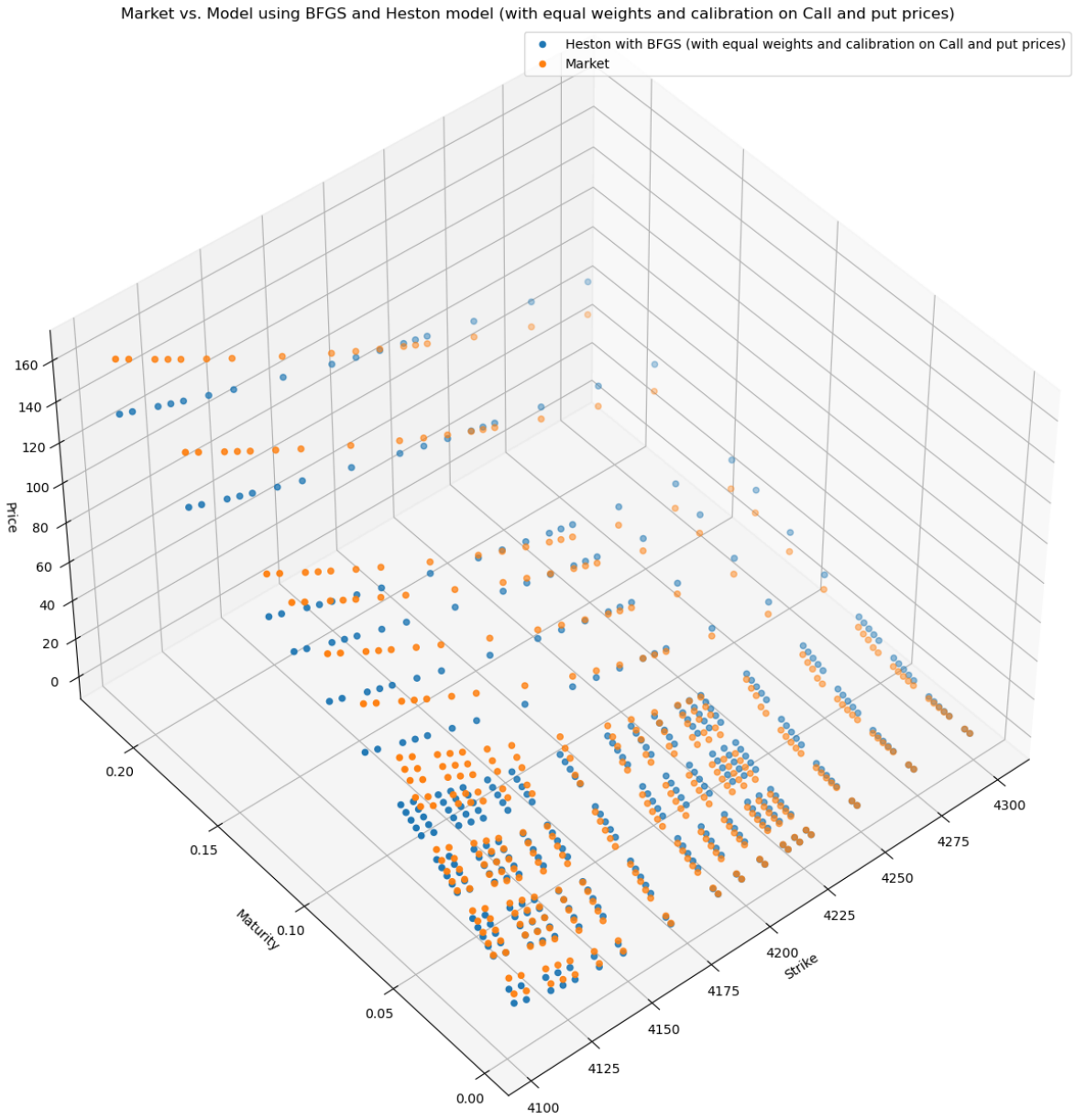
```
i = 16
x_i = [ 2.02499771  0.06555717  0.72261676  0.87292428 -0.96303649]
f_i = 8.135135772114497

i = 17
x_i = [ 1.98471225  0.06638767  0.72380659  0.87487082 -0.96303154]
f_i = 8.134993041314885

i = 18
x_i = [ 1.90898467  0.06799825  0.72543007  0.8768807  -0.96302122]
f_i = 8.134742776709741

i = 19
x_i = [ 1.76930512  0.07111464  0.72758606  0.87815828 -0.96299505]
f_i = 8.134507114352253

i = 20
x_i = [ 1.76660337  0.07138229  0.72646606  0.87494188 -0.96297739]
f_i = 8.134155767250494
Warning: Maximum number of iterations has been exceeded.
         Current function value: 8.134156
         Iterations: 20
         Function evaluations: 174
         Gradient evaluations: 29
```

In [33]:

```
params2 = xopt_pc
lenT = len(maturities)
lenK = len(strikes)
modelPrices_pc = np.zeros((lenT, lenK))

for i in range(lenT):
    for j in range(lenK):
        T = maturities[i]
        K = strikes[j]
        [km, cT_km] = mfc.genericFFT(params2, S0, K, r, q, T, alpha, eta, n,
        modelPrices_pc[i,j] = cT_km[0]


# plot
fig = plt.figure(figsize=(10,8))
labels = []
colormap = cm.Spectral
# create a list of colors to cycle through
colors = [colormap(i) for i in np.linspace(0, 0.9, len(maturities))]

#plt.gca().set_color_cycle([colormap(i) for i in np.linspace(0, 0.9, len(mat
plt.gca().set_prop_cycle(color=colors)
for i in range(len(maturities)):
    plt.plot(strikes, marketPrices[i,:], 'x')
    labels.append('T = ' + str(maturities[i]))

for i in range(len(maturities)):
    plt.plot(strikes, modelPrices_pc[i,:], '+')
    labels.append('T = ' + str(maturities[i]))
plt.legend(labels, loc='upper right', ncol=2)
plt.grid(alpha=0.25)
plt.xlabel('Strike')
plt.ylabel('Price')
plt.title('Market vs. Model using BFGS and Heston model (with equal weights
#plt.savefig('MarketvsModel_BFGS.png')
plt.show()
```



In [34]:
```
# create a meshgrid of the maturities and strikes
maturities_, strikes_ = np.meshgrid(maturities, strikes)

maturities_flat = maturities_.flatten()
strikes_flat =  strikes_.flatten()
prices_flat =  marketPrices.flatten()


fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(strikes_flat, maturities_flat, modelPrices_pc.T, label='Heston wi
ax.scatter(strikes_, maturities_, marketPrices.T, label='Market')
ax.view_init(elev=45, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')
ax.set_title('Market vs. Model using BFGS and Heston model (with equal weigh

plt.legend()

plt.show()
```



1.1.d Heston with weights inversely proportional to bid-ask spread on Call & Put prices

In [35]:
```
params = [ 2.34548424, 0.06375792, 0.59865706, -0.43626748, -0.95937438]

# Parameters
alpha = 1.5
eta = 0.2
n = 12

model ='Heston'

def callbackF_full_w(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValuefull_w(xi, *arg)))
    num_iter += 1

arg = (marketPrices,marketPrices_p,maturities,maturities_p,strikes,strikes_p
```

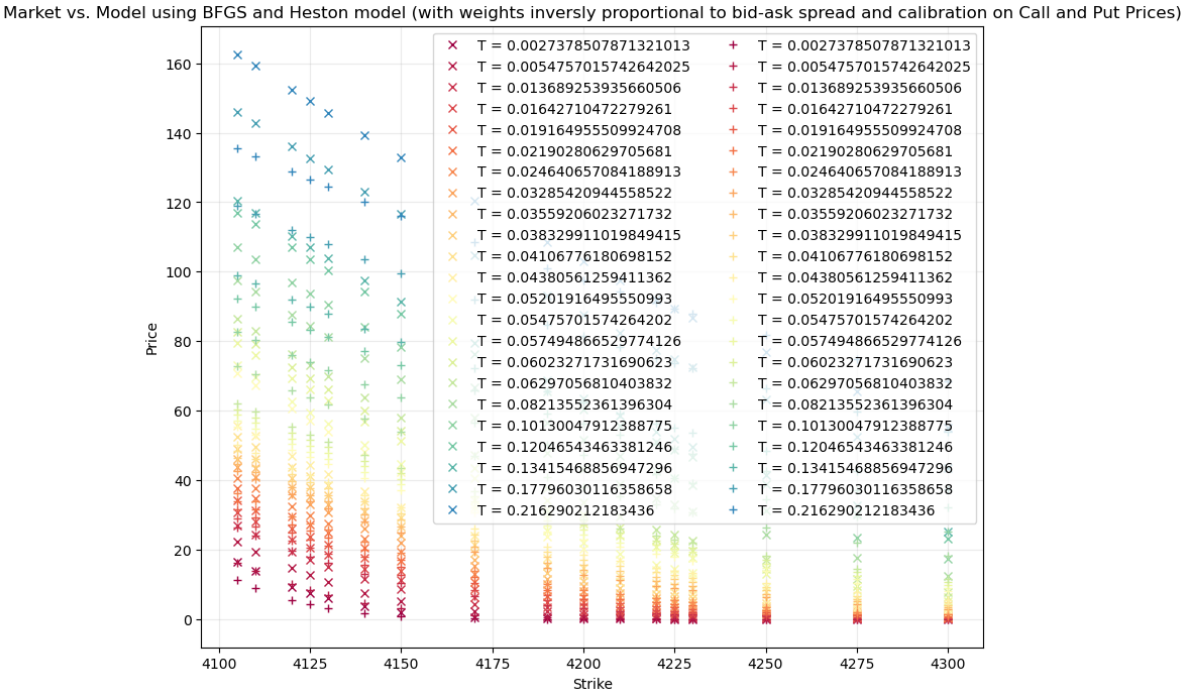
```
num_iter = 1
[xopt_pc_w, fopt_pc_w, gopt_pc_w, Bopt_pc_w, func_calls_pc_w, grad_calls_pc_
        mfc.eValuefull_w,
        params,
        args=arg,
        fprime=None,
        callback=callbackF_full_w,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
i = 1
x_i = [ 2.34543232  0.06057074  0.59862431 -0.43606929 -0.98452827]
f_i = 16.153223290752752

i = 2
x_i = [ 2.3455017   0.06418362  0.59832765 -0.43587775 -0.98572011]
f_i = 15.917158054363158

i = 3
x_i = [ 2.35042628  0.05412734  0.38767425 -0.18154877 -0.9860408 ]
f_i = 14.074332148846993

i = 4
x_i = [ 2.34129286  0.04574274  0.30573994  0.28106768 -0.98690756]
f_i = 12.887163145046701

i = 5
x_i = [ 2.32665223  0.05007535  0.4962001   0.78039261 -0.98736054]
f_i = 12.258277777300565

i = 6
x_i = [ 2.22673468  0.05398794  0.41317925  0.67312783 -0.98715418]
f_i = 12.22529565238518

i = 7
x_i = [ 2.19887251  0.05629963  0.43045331  0.73395742 -0.98698714]
f_i = 12.17233138841139

i = 8
x_i = [ 2.12577327  0.06048217  0.43851084  0.82609758 -0.9865544 ]
f_i = 12.14653444237217

i = 9
x_i = [ 2.11270107  0.06040255  0.42104421  0.82667624 -0.98652819]
f_i = 12.140839270277452

i = 10
x_i = [ 2.05462792  0.06097409  0.38778002  0.8582367  -0.98647346]
f_i = 12.135107175773179

i = 11
x_i = [ 1.98458594  0.06152094  0.36179671  0.89600116 -0.98645878]
f_i = 12.131644267746054

i = 12
x_i = [ 1.8756527   0.06239994  0.33491794  0.95045464 -0.98650283]
f_i = 12.127829436972767

i = 13
x_i = [ 1.76830954  0.06346705  0.31671603  0.99718825 -0.98656356]
f_i = 12.125828921789664

i = 14
x_i = [ 1.75977142  0.06356129  0.31511514  1.00065612 -0.98656875]
f_i = 12.125752184391267

i = 15
x_i = [ 1.74757139  0.06375523  0.31548581  1.0020283  -0.98658817]
f_i = 12.125610778149847
```

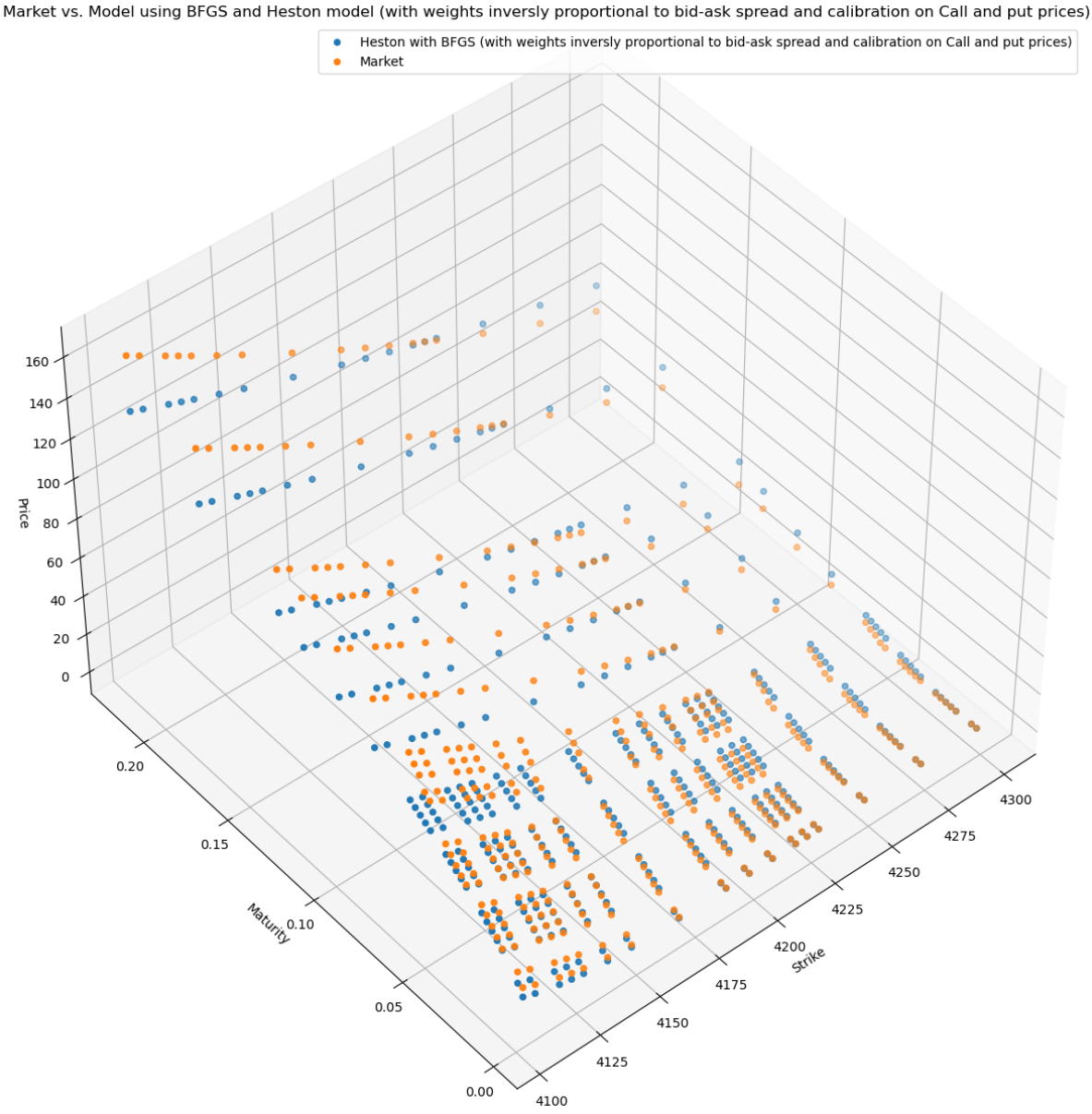
```
i = 16
x_i = [ 1.71949352  0.06427923  0.31609549  1.00520453 -0.98660997]
f_i = 12.125341927788961

i = 17
x_i = [ 1.66386737  0.06545377  0.31799663  1.01123898 -0.98664152]
f_i = 12.124856022147128

i = 18
x_i = [ 1.54436863  0.06813131  0.32066467  1.02727443 -0.98669624]
f_i = 12.124287532823017

i = 19
x_i = [ 1.50472195  0.06953021  0.32572423  1.03105944 -0.98671219]
f_i = 12.123942530920903

i = 20
x_i = [ 1.47772675  0.07080994  0.32848725  1.03740732 -0.98671367]
f_i = 12.123339169296703
Warning: Maximum number of iterations has been exceeded.
         Current function value: 12.123339
         Iterations: 20
         Function evaluations: 174
         Gradient evaluations: 29
```

In [36]:

```
params2 = xopt_pc_w
lenT = len(maturities)
lenK = len(strikes)
modelPrices_pc_w = np.zeros((lenT, lenK))

for i in range(lenT):
    for j in range(lenK):
        T = maturities[i]
        K = strikes[j]
        [km, cT_km] = mfc.genericFFT(params2, S0, K, r, q, T, alpha, eta, n,
        modelPrices_pc_w[i,j] = cT_km[0]


# plot
fig = plt.figure(figsize=(10,8))
labels = []
colormap = cm.Spectral
# create a list of colors to cycle through
colors = [colormap(i) for i in np.linspace(0, 0.9, len(maturities))]

#plt.gca().set_color_cycle([colormap(i) for i in np.linspace(0, 0.9, len(mat
plt.gca().set_prop_cycle(color=colors)
for i in range(len(maturities)):
    plt.plot(strikes, marketPrices[i,:], 'x')
    labels.append('T = ' + str(maturities[i]))

for i in range(len(maturities)):
    plt.plot(strikes, modelPrices_pc_w[i,:], '+')
    labels.append('T = ' + str(maturities[i]))
plt.legend(labels, loc='upper right', ncol=2)
plt.grid(alpha=0.25)
plt.xlabel('Strike')
plt.ylabel('Price')
plt.title('Market vs. Model using BFGS and Heston model (with weights invers
#plt.savefig('MarketvsModel_BFGS.png')
plt.show()
```



In [37]:
```
# create a meshgrid of the maturities and strikes
maturities_, strikes_ = np.meshgrid(maturities, strikes)

maturities_flat = maturities_.flatten()
strikes_flat =  strikes_.flatten()
prices_flat =  marketPrices.flatten()


fig = plt.figure(figsize= [15,15])
ax = fig.add_subplot(111, projection='3d')
ax.scatter(strikes_flat, maturities_flat, modelPrices_pc_w.T, label='Heston
ax.scatter(strikes_, maturities_, marketPrices.T, label='Market')
ax.view_init(elev=45, azim=230)
ax.set_xlabel('Strike')
ax.set_ylabel('Maturity')
ax.set_zlabel('Price')
ax.set_title('Market vs. Model using BFGS and Heston model (with weights inv

plt.legend()

plt.show()
```



1.2 VGSA model:

ΘVGSA = {σ, ν, θ, κ, η, λ}

1.2.a VGSA with equal weights on Call Prices

In [38]:
```
params = [0.0825, 0.8, 0.046, 2.3, 0.8 , 0.9 ]
params = [0.02351992, 0.03838838, 0.00533235, 2.38917246, 1.89090463, 0.8400
#params_v = [0.0825, ?, 0.046, 2.3, ? , v0 ]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

model_v = 'VGSA'

def callbackF(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
```

```
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValue(xi, *arg)))
    num_iter += 1

arg = (marketPrices, maturities, strikes, r, q, S0, alpha, eta, n, model_v)

num_iter = 1
[xopt_v, fopt_v, gopt_v, Bopt_v, func_calls_v, grad_calls_v, warnflg_v] = fm
        mfc.eValue,
        params,
        args=arg,
        fprime=None,
        callback=callbackF,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
6.314782183908005
6.314761177908677
6.314794915280337
6.314735959397373
6.314782211893761
6.314782180769975
6.314782183959068
51.61254295958128
51.61254295958897
51.6125429598005
51.61254295958098
51.61254295958114
51.61254295958124
51.61254295958132
123902531433.16956
117766962574.62431
109473250589.9518
132163467346.19774
130030318032.12843
124423565396.27554
108041754426.755
1554.3200194250874
1554.3200304117847
1554.3186900301894
1554.3201699558315
1554.3200186739193
1554.3200194492003
1554.3200194305243
49.45614808366735
49.45617156382418
49.45610657233308
49.456229489125036
49.456147988812845
49.45614809365504
49.45614808395662
3.8114633660966515
3.8114733727745236
3.8114563969000717
3.811485757113267
3.811463345702105
3.8114633684329293
3.8114633661315898

i = 1
x_i = [0.02412953 0.03801891 0.00667382 2.38917165 1.89090472 0.84004647]
3.8114633660966515
f_i = 3.8114633660966515
11.45820713261508
11.458218883012199
11.45820316332773
11.458219604436422
11.45820710356856
11.458207152389075
11.458207132430026
3.697456259446731
3.697461351218871
3.6974535001750537
3.6974653109938065
3.6974562462172123
3.6974562617587883
3.6974562595004103
```

```
i = 2
x_i = [0.02913407 0.05527216 0.00936538 2.38932342 1.89088659 0.84004513]
3.697456259446731
f_i = 3.697456259446731
4.267914842835607
4.267912426311919
4.267914525197936
4.267919571149809
4.267914829521385
4.267914845592156
4.267914842888928
3.6831457492268522
3.6831497433832054
3.6831433802216726
3.683154172465795
3.6831457358896555
3.6831457516114225
3.6831457492832413

i = 3
x_i = [0.02885127 0.05610095 0.00973116 2.38933454 1.89088478 0.84004503]
3.6831457492268522
f_i = 3.6831457492268522
3.6552034736173424
3.65520707710686
3.655201338805894
3.6552110732165684
3.6552034612094677
3.6552034758575656
3.655203473673052
3.5625328596324466
3.5625348595130055
3.5625317029698547
3.5625370314289957
3.5625328510067398
3.56253286125872
3.562532859685118

i = 4
x_i = [0.02830089 0.05416262 0.00945117 2.54722199 1.85603775 0.83879377]
3.5625328596324466
f_i = 3.5625328596324466
3.408905369290925
3.408903364373613
3.408906766264336
3.408900783807213
3.4089053701546366
3.408905369221316
3.4089053693313707

i = 5
x_i = [0.02748172 0.05199039 0.0094647  3.05554239 1.74761475 0.83299369]
3.408905369290925
f_i = 3.408905369290925
3.3250893564407344
3.3250866421645995
3.3250913491967404
3.3250827677374666
3.3250893598974205
3.3250893557605883
```

```
3.3250893564685566

i = 6
x_i = [0.0266923  0.04978441 0.00951384 3.45709033 1.6889403  0.82716997]
3.3250893564407344
f_i = 3.3250893564407344
3.1665426176994744
3.1665428682503505
3.166542885329333
3.1665410972012
3.1665426180801903
3.1665426176734592
3.1665426177088665

i = 7
x_i = [0.02465117 0.04349004 0.00920883 4.15592893 1.60799899 0.81679032]
3.1665426176994744
f_i = 3.1665426176994744
6.415182934229975
6.4152023138861995
6.415158528590675
6.415243152297902
6.415182894240722
6.415182944609781
6.415182934308679
3.143022943920842
3.1430243967914144
3.1430222338536966
3.14302444348264
3.143022942133782
3.1430229444261557
3.143022943926121

i = 8
x_i = [0.02406407 0.04212554 0.00923432 4.34301794 1.58317934 0.8141652 ]
3.143022943920842
f_i = 3.143022943920842
3.119630154200514
3.1196299500181293
3.119630196197422
3.119630287124216
3.119630153562273
3.119630154462203
3.1196301541880627

i = 9
x_i = [0.02361808 0.04288007 0.00991309 4.63247779 1.53416675 0.81049096]
3.119630154200514
f_i = 3.119630154200514
3.1171029881003287
3.1171032385394235
3.117102741424554
3.117103820779901
3.1171029872401617
3.117102988407415
3.117102988086165

i = 10
x_i = [0.02348812 0.04241256 0.00991319 4.73223279 1.50811437 0.81034776]
3.1171029881003287
f_i = 3.1171029881003287
```

```
3.116388318411422
3.11638833614059
3.116388322702283
3.1163883114417543
3.116388318458743
3.116388318453826
3.116388318395094

i = 11
x_i = [0.02354817 0.0425382  0.00995691 4.77615547 1.48749931 0.81149089]
3.116388318411422
f_i = 3.116388318411422
3.116344010316701
3.116344078011849
3.1163439722002617
3.116344130336316
3.1163440102492377
3.1163440103918143
3.1163440103006628

i = 12
x_i = [0.02352924 0.04247257 0.00994245 4.77246557 1.48451257 0.81250161]
3.116344010316701
f_i = 3.116344010316701
3.1162667869042755
3.1162669019155564
3.116266707659116
3.1162670311289387
3.1162667867238105
3.1162667870111935
3.1162667868883247

i = 13
x_i = [0.02349713 0.04236086 0.00991678 4.76575029 1.47459338 0.81529525]
3.1162667869042755
f_i = 3.1162667869042755
3.1161282862080943
3.1161284542680043
3.1161281606717437
3.1161286698204314
3.1161282859003467
3.1161282863498916
3.1161282861923327

i = 14
x_i = [0.02344294 0.04216822 0.00987088 4.75353563 1.45345306 0.82104832]
3.1161282862080943
f_i = 3.1161282862080943
3.1158826157869304
3.115882852196412
3.1158824302128316
3.1158831793955817
3.1158826153147907
3.1158826159720125
3.115882615771657

i = 15
x_i = [0.02334797 0.0418269  0.00978787 4.73116329 1.41168379 0.83226659]
3.1158826157869304
f_i = 3.1158826157869304
3.11547509302537
```

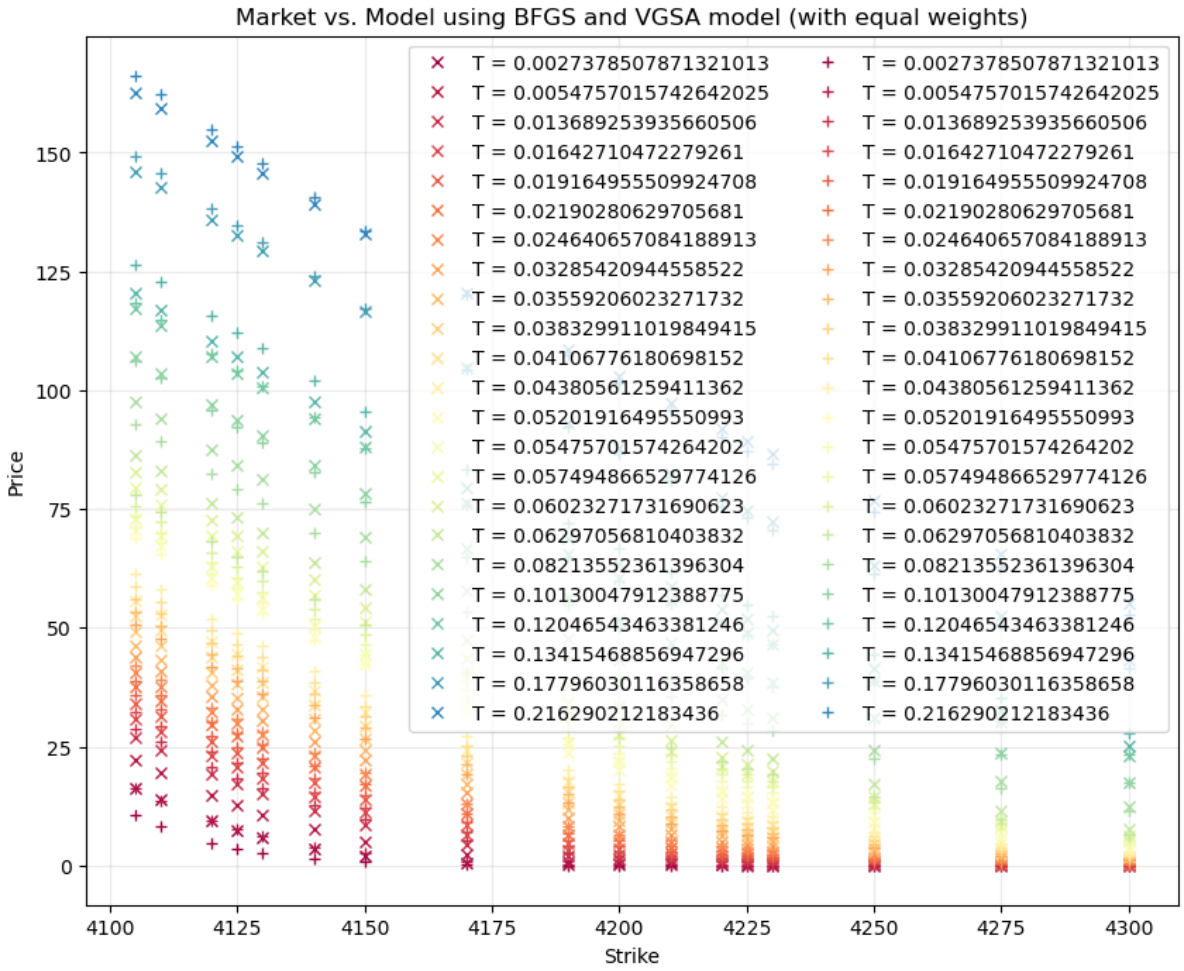
```
3.115475429451407
3.115474819012293
3.1154759198408164
3.115475092311191
3.1154750932700903
3.1154750930103763

i = 16
x_i = [0.0231796  0.04121674 0.00963654 4.68938444 1.33103696 0.85382213]
3.11547509302537
f_i = 3.11547509302537
3.114911865869051
3.1149123539522208
3.114911456096826
3.1149130929559643
3.1149118647831786
3.1149118661961666
3.1149118658547894

i = 17
x_i = [0.02289427 0.04017794 0.00937368 4.61455246 1.18219044 0.89365681]
3.114911865869051
f_i = 3.114911865869051
3.11421802555623
3.1142185384876906
3.114217586590334
3.1142193413655934
3.114218024355946
3.1142180258974452
3.114218025542317

i = 18
x_i = [0.02266232 0.03933039 0.00915024 4.54637598 1.04057151 0.9318654 ]
3.11421802555623
f_i = 3.11421802555623
3.1130444678263287
3.1130448926775323
3.113044099928276
3.1130455726620925
3.113044466778138
3.1130444681166294
3.1130444678119957

i = 19
x_i = [0.02241489 0.03843875 0.00891737 4.47903002 0.86724616 0.9783936 ]
3.1130444678263287
f_i = 3.1130444678263287
3.111184405163533
3.111184713427916
3.111184141834628
3.111185182516415
3.111184404430715
3.1111844053643667
3.111184405147594

i = 20
x_i = [0.02201809 0.03707185 0.00858907 4.40381212 0.57567413 1.05553105]
3.111184405163533
f_i = 3.111184405163533
Warning: Maximum number of iterations has been exceeded.
         Current function value: 3.111184
```

```
         Iterations: 20
         Function evaluations: 203
         Gradient evaluations: 29
```

In [39]:
```
xopt_v
```

Out[39]:
```
array([0.02201809, 0.03707185, 0.00858907, 4.40381212, 0.57567413,
       1.05553105])
```

In [40]:
```
params2 = xopt_v
lenT = len(maturities)
lenK = len(strikes)
modelPrices = np.zeros((lenT, lenK))

for i in range(lenT):
    for j in range(lenK):
        T = maturities[i]
        K = strikes[j]
        [km, cT_km] = mfc.genericFFT(params2, S0, K, r, q, T, alpha, eta, n,
        modelPrices[i,j] = cT_km[0]


# plot
fig = plt.figure(figsize=(10,8))
labels = []
colormap = cm.Spectral
# create a list of colors to cycle through
colors = [colormap(i) for i in np.linspace(0, 0.9, len(maturities))]

#plt.gca().set_color_cycle([colormap(i) for i in np.linspace(0, 0.9, len(mat
plt.gca().set_prop_cycle(color=colors)
for i in range(len(maturities)):
    plt.plot(strikes, marketPrices[i,:], 'x')
    labels.append('T = ' + str(maturities[i]))

for i in range(len(maturities)):
    plt.plot(strikes, modelPrices[i,:], '+')
    labels.append('T = ' + str(maturities[i]))
plt.legend(labels, loc='upper right', ncol=2)
plt.grid(alpha=0.25)
plt.xlabel('Strike')
plt.ylabel('Price')
plt.title('Market vs. Model using BFGS and VGSA model (with equal weights)')
plt.savefig('MarketvsModel_BFGS.png')
plt.show()
```



In [ ]:

1.2.b VGSA with weights inversely proportional to bid-ask spread on Call prices

In [41]:
```
params = [0.0825, 0.8, 0.046, 2.3, 0.8 , 0.9 ]
params = [0.02351992, 0.03838838, 0.00533235, 2.38917246, 1.89090463, 0.8400
#params_v = [0.0825, ?, 0.046, 2.3, ? , v0 ]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

model_v = 'VGSA'

def callbackF_w(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValue_w(xi, *arg)))
    num_iter += 1

arg = (marketPrices, maturities, strikes, r, q, S0, alpha, eta, n, model_v,

num_iter = 1
[xopt_v_w, fopt_v_w, gopt_v_w, Bopt_v_w, func_calls_v_w, grad_calls_v_w, war
        mfc.eValue_w,
```

```
        params,
        args=arg,
        fprime=None,
        callback=callbackF_w,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
4.194365299468084
4.194352241425854
4.1943731098703045
4.194337418243462
4.19436531612836
4.194365297606574
4.194365299498152
33.48909991477731
33.48909991479321
33.489099915207625
33.48909991477664
33.489099914777064
33.489099914777235
33.48909991477738
10310677212.206112
9148226736.87956
14602478428.863794
12121802476.381157
8018623743.787641
10302343813.680382
9249060107.309277
1124.292653452149
1124.2926633243271
1124.2915911755422
1124.2927715001122
1124.2926529659771
1124.2926534672463
1124.2926534553071
31.35395711959494
31.353974451659212
31.353930154346646
31.35400895738603
31.35395706285284
31.353957125536542
31.353957119770506
2.7586532445134
2.758659675405733
2.758649423203719
2.7586639999808855
2.758653235524795
2.758653245534402
2.75865324453698

i = 1
x_i = [0.02411301 0.03803363 0.00659871 2.3891717  1.89090471 0.84004647]
2.7586532445134
f_i = 2.7586532445134
34.18604091441784
34.186040370183676
34.186021040292424
34.18608129575542
34.18604084002865
34.18604092734706
34.1860409146971
2.7444994180568973
2.7445048550696343
2.74449556064697
2.7445118792870757
2.7444994067511272
2.7444994193665
2.7444994180830053
```

```
i = 2
x_i = [0.02337586 0.03845699 0.00709799 2.38917319 1.89090454 0.84004644]
2.7444994180568973
f_i = 2.7444994180568973
2.719711194843781
2.719715915395189
2.7197078766487865
2.7197220709491368
2.7197111843774975
2.7197111961045115
2.719711194870227

i = 3
x_i = [0.02371362 0.03976892 0.00732255 2.38918422 1.89090318 0.84004632]
2.719711194843781
f_i = 2.719711194843781
2.6731698334195415
2.673173756656139
2.67316708525627
2.6731788771818734
2.6731698243596465
2.6731698345415156
2.673169833444797

i = 4
x_i = [0.02366942 0.03972349 0.00731086 2.44170519 1.88190334 0.83963392]
2.6731698334195415
f_i = 2.6731698334195415
2.594128316138295
2.5941306304079266
2.5941267051076475
2.594133656417323
2.5941283100208237
2.5941283169462004
2.594128316160303

i = 5
x_i = [0.02340406 0.03907367 0.00722596 2.59046661 1.85507572 0.83835636]
2.594128316138295
f_i = 2.594128316138295
2.507676833675361
2.507675764336925
2.507677671834593
2.5076742913414276
2.507676833738137
2.5076768337088553
2.5076768336879076

i = 6
x_i = [0.02263973 0.03725107 0.00709732 3.0300721  1.7750448  0.83440868]
2.507676833675361
f_i = 2.507676833675361
2.4745296049579455
2.4745286784717577
2.4745304202876075
2.4745270518221116
2.47452960529651
2.47452960494901
2.4745296049668752
2.6820369606221326
```

```
2.6820464904266608
2.6820282406715035
2.6820599839501864
2.682036949239097
2.6820369622351805
2.6820369606352354
2.437283319916808
2.437283821634198
2.437283131178706
2.437283581947591
2.437283318962415
2.4372833200901716
2.4372833199230572

i = 7
x_i = [0.0207424  0.03172876 0.00654456 3.60000946 1.69507022 0.82940738]
2.437283319916808
f_i = 2.437283319916808
2.9974050228271008
2.9974167006311863
2.9973921329248228
2.9974391101894735
2.9974050063941036
2.997405025304199
2.9974050228453586
2.4266468672530603
2.426648243079897
2.426645886239695
2.426649494467158
2.426646865152791
2.4266468675971096
2.4266468672578325

i = 8
x_i = [0.02016142 0.03034886 0.00647612 3.7975075  1.66495436 0.82772714]
2.4266468672530603
f_i = 2.4266468672530603
2.412237607767465
2.412237938386477
2.412237293113824
2.4122386497343196
2.4122376067824023
2.4122376079619383
2.4122376077662704

i = 9
x_i = [0.01990941 0.03049288 0.00676458 4.07062636 1.61737501 0.82542513]
2.412237607767465
f_i = 2.412237607767465
2.4086671273298954
2.408667259403773
2.408667032684568
2.408667417386768
2.40866712721065
2.4086671273767517
2.408667127324843

i = 10
x_i = [0.01969026 0.03007785 0.00684449 4.31588362 1.57052448 0.8236714 ]
2.4086671273298954
f_i = 2.4086671273298954
```

```
2.4086077362459286
2.4086078191162787
2.4086076880630616
2.4086078907999338
2.408607736193914
2.4086077362813954
2.4086077362409157

i = 11
x_i = [0.01969724 0.03007966 0.00683369 4.30512551 1.56982598 0.82406235]
2.4086077362459286
f_i = 2.4086077362459286
2.4084961173923607
2.408496186068952
2.408496085287081
2.4084962210458274
2.408496117359083
2.4084961174246127
2.408496117387385

i = 12
x_i = [0.01967351 0.02998433 0.0068018  4.29287233 1.56644577 0.82484422]
2.4084961173923607
f_i = 2.4084961173923607
2.4082829125990624
2.408282962819568
2.408282900203645
2.408282954810202
2.4082829125895575
2.40828291262718
2.4082829125942964
2.4076312237322934
2.4076312022365687
2.4076312940547164
2.407630999697012
2.407631223827301
2.4076312237430786
2.4076312237281283

i = 13
x_i = [0.01936422 0.02891616 0.00650183 4.2148982  1.52563255 0.83226554]
2.4076312237322934
f_i = 2.4076312237322934
2.406327740019726
2.4063277623409647
2.406327826485107
2.406327370295371
2.4063277401406853
2.406327740025727
2.4063277400167675

i = 14
x_i = [0.0186673  0.02667708 0.00593395 4.11207999 1.43715557 0.84637482]
2.406327740019726
f_i = 2.406327740019726
2.405656791939093
2.40565759570207
2.405656238451682
2.4056579969124767
2.4056567913328393
2.4056567920287972
```

```
2.4056567919381675
2.405395866768705
2.4053960642495826
2.4053958184364768
2.4053958210361817
2.4053958667160247
2.405395866796857
2.4053958667664137

i = 15
x_i = [0.01816426 0.02517871 0.00558328 4.0694341  1.37280545 0.85568094]
2.405395866768705
f_i = 2.405395866768705
2.4043985878008938
2.404399244610652
2.4043980937449874
2.4043997678984708
2.4043985871843203
2.4043985878931076
2.404398587799311

i = 16
x_i = [0.01740667 0.02308154 0.00512124 4.03263656 1.27199918 0.86942315]
2.4043985878008938
f_i = 2.4043985878008938
2.40313818361174
2.4031384745498525
2.403137895912976
2.403139043233052
2.403138183084185
2.403138183695631
2.403138183609082

i = 17
x_i = [0.01727145 0.02295293 0.00513251 4.05757339 1.24684933 0.8720406 ]
2.40313818361174
f_i = 2.40313818361174
2.4020796037166123
2.402079940817757
2.402079241396169
2.4020806935125303
2.402079603160386
2.4020796037983927
2.4020796037136414

i = 18
x_i = [0.01675863 0.02165757 0.00486571 4.07323179 1.15946803 0.88286053]
2.4020796037166123
f_i = 2.4020796037166123
2.4007208149926735
2.400720883938901
2.400720771431156
2.4007209246656727
2.400720814972007
2.4007208150111277
2.4007208149890182

i = 19
x_i = [0.01621236 0.02026727 0.00456162 4.09307472 1.04973466 0.89672412]
2.4007208149926735
f_i = 2.4007208149926735
```

```
2.4193498007356733
2.4193534587755186
2.4193459730249565
2.4193594999154677
2.4193497978222473
2.4193498009792123
2.4193498007363154
2.4004375497943107
2.4004378873999435
2.4004372530479037
2.4004382981254073
2.4004375495501695
2.400437549833817
2.4004375497911052

i = 20
x_i = [0.01590109 0.01945312 0.00437731 4.08278376 0.99640352 0.90398703]
2.4004375497943107
f_i = 2.4004375497943107
Warning: Maximum number of iterations has been exceeded.
         Current function value: 2.400438
         Iterations: 20
         Function evaluations: 224
         Gradient evaluations: 32
```

1.2.c VGSA with equal weights on Call & Put prices

In [42]:

```
params = [0.0825, 0.8, 0.046, 2.3, 0.8 , 0.9 ]
params = [0.02351992, 0.03838838, 0.00533235, 2.38917246, 1.89090463, 0.8400
#params_v = [0.0825, ?, 0.046, 2.3, ? , v0 ]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

model_v = 'VGSA'

def callbackF_full(xi):
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValuefull(xi, *arg)))
    num_iter += 1

arg = (marketPrices,marketPrices_p,maturities,maturities_p,strikes,strikes_p

num_iter = 1
[xopt_v_pc, fopt_v_pc, gopt_v_pc, Bopt_v_pc, func_calls_v_pc, grad_calls_v_p
        mfc.eValuefull,
        params,
        args=arg,
        fprime=None,
        callback=callbackF_full,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
i = 1
x_i = [2.30923148e-02 3.89963337e-02 1.85247147e-03 2.38917374e+00
 1.89090449e+00 8.40046472e-01]
f_i = 13.464623041787542

i = 2
x_i = [ 3.65464860e-02  3.82657748e-02 -2.03828354e-03  2.38917001e+00
  1.89090496e+00  8.40046394e-01]
f_i = 11.013114563988871

i = 3
x_i = [ 4.00954142e-02  4.95569032e-02 -8.76928901e-04  2.38916188e+00
  1.89090614e+00  8.40046125e-01]
f_i = 10.530972464266782

i = 4
x_i = [ 3.74577268e-02  5.01925741e-02 -4.93994558e-05  2.27812215e+00
  1.90435743e+00  8.39616980e-01]
f_i = 9.818753550438256

i = 5
x_i = [3.52338495e-02 5.12963266e-02 6.32092302e-04 1.83775311e+00
 1.96927486e+00 8.33886161e-01]
f_i = 9.539568460694458

i = 6
x_i = [3.49656490e-02 5.27058615e-02 7.35587871e-04 1.32329156e+00
 2.03470895e+00 8.26895062e-01]
f_i = 9.396308983117484

i = 7
x_i = [3.49567079e-02 5.53358863e-02 7.30587466e-04 3.10987928e-01
 2.14590085e+00 8.13066222e-01]
f_i = 9.131506600489436

i = 8
x_i = [ 3.52052941e-02  6.00656456e-02  4.48195110e-04 -1.79482842e+00
  2.34357191e+00  7.84055021e-01]
f_i = 8.720286428356854

i = 9
x_i = [ 3.50899610e-02  6.28572594e-02 -4.33398080e-04 -4.38426646e+00
  2.59389635e+00  7.48076491e-01]
f_i = 8.500053981854586

i = 10
x_i = [ 3.51384374e-02  6.12585264e-02 -6.87131693e-04 -4.19060125e+00
  2.60349435e+00  7.51010723e-01]
f_i = 8.478682591584333

i = 11
x_i = [ 3.57238179e-02  6.17316385e-02 -7.11109102e-04 -4.03552637e+00
  2.57795563e+00  7.53425358e-01]
f_i = 8.46921166687018

i = 12
x_i = [ 3.56919077e-02  6.17860676e-02 -7.35226648e-04 -4.11846120e+00
  2.58491595e+00  7.52552561e-01]
f_i = 8.468936744746841
```

```
i = 13
x_i = [ 3.56883524e-02  6.17746531e-02 -7.35415221e-04 -4.11769113e+00
  2.58393075e+00  7.52883222e-01]
f_i = 8.468934976061067

i = 14
x_i = [ 3.56823551e-02  6.17550376e-02 -7.35554066e-04 -4.11663606e+00
  2.58144258e+00  7.53681641e-01]
f_i = 8.468931766243582

i = 15
x_i = [ 3.56716326e-02  6.17199755e-02 -7.35544769e-04 -4.11485632e+00
  2.57632535e+00  7.55308783e-01]
f_i = 8.468925777082585

i = 16
x_i = [ 3.56518274e-02  6.16550066e-02 -7.35230130e-04 -4.11153554e+00
  2.56617630e+00  7.58516299e-01]
f_i = 8.468914584156007

i = 17
x_i = [ 3.56143038e-02  6.15327213e-02 -7.34433897e-04 -4.10581129e+00
  2.54623349e+00  7.64807127e-01]
f_i = 8.468893776575706

i = 18
x_i = [ 3.55425757e-02  6.12988216e-02 -7.32435584e-04 -4.09485621e+00
  2.50704265e+00  7.77150939e-01]
f_i = 8.468855926897685

i = 19
x_i = [ 3.54045123e-02  6.08500987e-02 -7.28090517e-04 -4.07463329e+00
  2.43024904e+00  8.01332452e-01]
f_i = 8.468789477605563

i = 20
x_i = [ 3.51409268e-02  5.99958322e-02 -7.18759803e-04 -4.03698394e+00
  2.28151121e+00  8.48164337e-01]
f_i = 8.468680311119828
Warning: Maximum number of iterations has been exceeded.
         Current function value: 8.468680
         Iterations: 20
         Function evaluations: 182
         Gradient evaluations: 26
```

1.2.d VGSA with weights inversely proportional to bid-ask spread on Call & Put prices

In [43]:

```
params = [0.0825, 0.8, 0.046, 2.3, 0.8 , 0.9 ]
params = [0.02351992, 0.03838838, 0.00533235, 2.38917246, 1.89090463, 0.8400
#params_v = [0.0825, ?, 0.046, 2.3, ? , v0 ]
# Parameters
alpha = 1.5
eta = 0.2
n = 12

model_v = 'VGSA'

def callbackF_full_w(xi):
```

```
    global num_iter
    global arg
    print(' ')
    print('i = ' + str(num_iter))
    print('x_i = ' + str(xi))
    print('f_i = ' + str(mfc.eValuefull_w(xi, *arg)))
    num_iter += 1

arg = (marketPrices,marketPrices_p,maturities,maturities_p,strikes,strikes_p

num_iter = 1
[xopt_v_w_pc, fopt_v_w_pc, gopt_v_w_pc, Bopt_v_w_pc, func_calls_v_w_pc, grad
        mfc.eValuefull_w,
        params,
        args=arg,
        fprime=None,
        callback=callbackF_full_w,
        maxiter=20,
        full_output=True,
        retall=False)
```

```
i = 1
x_i = [2.31229849e-02 3.89918330e-02 1.82117049e-03 2.38917359e+00
 1.89090450e+00 8.40046472e-01]
f_i = 19.23825496897768

i = 2
x_i = [ 3.01081185e-02  3.82351913e-02 -1.83871436e-04  2.38917081e+00
  1.89090484e+00  8.40046433e-01]
f_i = 15.153752333195028

i = 3
x_i = [3.38786779e-02 4.80785156e-02 7.35069930e-04 2.38916725e+00
 1.89090535e+00 8.40046111e-01]
f_i = 14.292109748758524

i = 4
x_i = [ 3.30681797e-02  5.10611761e-02  4.23025329e-04 -7.69739113e-02
  2.25140704e+00  8.08912041e-01]
f_i = 13.189833559206718

i = 5
x_i = [ 3.35872179e-02  5.58043870e-02 -3.08540831e-05 -2.86949243e+00
  2.57586847e+00  7.72623996e-01]
f_i = 12.501968529098587

i = 6
x_i = [ 3.30121907e-02  5.57932380e-02 -3.11349714e-04 -3.93286845e+00
  2.69496083e+00  7.58859329e-01]
f_i = 12.457770386136666

i = 7
x_i = [ 3.32443396e-02  5.62911375e-02 -2.21129888e-04 -3.77919923e+00
  2.66959345e+00  7.61087130e-01]
f_i = 12.452851384242976

i = 8
x_i = [ 3.31785674e-02  5.59003057e-02 -2.30306747e-04 -3.68420697e+00
  2.66184871e+00  7.62491045e-01]
f_i = 12.451886009054762

i = 9
x_i = [ 3.31626227e-02  5.59352152e-02 -2.37305922e-04 -3.73472482e+00
  2.66496817e+00  7.62009775e-01]
f_i = 12.451690079981457

i = 10
x_i = [ 3.31483635e-02  5.58974721e-02 -2.36822775e-04 -3.73884642e+00
  2.66265893e+00  7.62235955e-01]
f_i = 12.451623811402046

i = 11
x_i = [ 3.29897478e-02  5.54250981e-02 -2.33110064e-04 -3.75557409e+00
  2.63718933e+00  7.64876833e-01]
f_i = 12.45104377931212

i = 12
x_i = [ 3.26556340e-02  5.43909091e-02 -2.26261644e-04 -3.76878354e+00
  2.58357579e+00  7.70560060e-01]
f_i = 12.450026834668968
```

```
i = 13
x_i = [ 3.19914122e-02  5.23026367e-02 -2.14269880e-04 -3.77423017e+00
  2.47785292e+00  7.81898409e-01]
f_i = 12.448354208767325

i = 14
x_i = [ 3.07323581e-02  4.83463227e-02 -1.96313828e-04 -3.76612416e+00
  2.28302639e+00  8.02963256e-01]
f_i = 12.446376667980585

i = 15
x_i = [ 3.02058923e-02  4.67727866e-02 -1.96497214e-04 -3.76145650e+00
  2.21379869e+00  8.10650161e-01]
f_i = 12.444083698665077

i = 16
x_i = [ 2.92948986e-02  4.40151627e-02 -1.80142644e-04 -3.69146303e+00
  2.08784438e+00  8.24717060e-01]
f_i = 12.440230153640615

i = 17
x_i = [ 2.77197284e-02  3.93380121e-02 -1.56560633e-04 -3.55111360e+00
  1.88127206e+00  8.48074781e-01]
f_i = 12.435032929126146

i = 18
x_i = [ 2.63274679e-02  3.54924675e-02 -1.24376127e-04 -3.48026105e+00
  1.71473400e+00  8.67181107e-01]
f_i = 12.433163493172176

i = 19
x_i = [ 2.55093056e-02  3.34764420e-02 -1.05703883e-04 -3.43228245e+00
  1.64002182e+00  8.76407472e-01]
f_i = 12.431032144326393

i = 20
x_i = [ 2.48372470e-02  3.16473038e-02 -1.08263283e-04 -3.42577940e+00
  1.56808419e+00  8.84855828e-01]
f_i = 12.4299852609262
Warning: Maximum number of iterations has been exceeded.
         Current function value: 12.429985
         Iterations: 20
         Function evaluations: 217
         Gradient evaluations: 31
```

1.3 Results of calibration for the different algorihtms:

In [44]:
```
df_param = pd.DataFrame({'Model':['Heston', 'Heston']*2,'Method':['BFGS with

list_arg_heston = ['kappa','theta', 'sigma','rho', 'v0']
i=0
for para in list_arg_heston:
    df_param[para] = [xopt[i],xopt_w[i], xopt_pc[i], xopt_pc_w[i]]
    i+=1

df_param
```

Out[44]:

	Model	Method	Prices	kappa	theta	sigma	rho	v0
0	Heston	BFGS with equal weights	Call	2.359482	-0.079593	0.599166	-1.000438	-0.955222
1	Heston	BFGS with weights inversely proportional to bi...	Call	4.261961	-0.059995	0.586308	-1.011792	-0.957226
2	Heston	BFGS with equal weights	Call & Put	1.766603	0.071382	0.726466	0.874942	-0.962977
3	Heston	BFGS with weights inversely proportional to bi...	Call & Put	1.477727	0.070810	0.328487	1.037407	-0.986714

In [45]:
```
df_param_v = pd.DataFrame({'Model':['VGSA', 'VGSA']*2,'Method':['BFGS with e

list_arg_vgsa = ['sigma','nu','theta','kappa', 'eta', 'lambda']
i=0
for para in list_arg_heston:
    df_param_v[para] = [xopt_v[i],xopt_v_w[i],xopt_v_pc[i],xopt_v_w_pc[i]]
    i+=1

df_param_v
```

Out[45]:

	Model	Method	Prices	kappa	theta	sigma	rho	v0
0	VGSA	BFGS with equal weights	Call	0.022018	0.037072	0.008589	4.403812	0.575674
1	VGSA	BFGS with weights inversely proportional to bi...	Call	0.015901	0.019453	0.004377	4.082784	0.996404
2	VGSA	BFGS with equal weights	Call & Put	0.035141	0.059996	-0.000719	-4.036984	2.281511
3	VGSA	BFGS with weights inversely proportional to bi...	Call & Put	0.024837	0.031647	-0.000108	-3.425779	1.568084

2. Local volatility surface

In [82]:
```
def vol_surface(params, strikes, maturities, modelPrices, model, title_):
    Prices = modelPrices.T
    params2 = params
    lenT = 3 * len(maturities)
    lenK = 3 * len(strikes)
    h_strikes = (strikes[-1] - strikes[0]) / lenK
    h_maturities = (maturities[-1] - maturities[0]) / lenT
    strikes_l = np.arange(strikes[0], strikes[-1] + h_strikes, h_strikes)
    maturities_l = np.arange(maturities[0], maturities[-1] + h_maturities, h
    lenT = len(maturities_l)
    lenK = len(strikes_l)
```

```
    Prices = np.zeros((lenK, lenT))

    print('Computing the prices according to the model and the opt-parameter
    for i in tqdm(range(lenK)):
        for j in range(lenT):
            T = maturities_l[j]
            K = strikes_l[i]
            [km, cT_km] = mfc.genericFFT(params2, S0, K, r, q, T, alpha, eta
            Prices[i, j] = cT_km[0]

    sig_loc = np.zeros((lenK, lenT))
    print('Computing the local volatility surface')
    for i in tqdm(range(1, lenK - 1)):
        for j in range(1, lenT - 1):
            dC_dt = (Prices[i, j + 1] - Prices[i, j - 1]) / (maturities_l[j
            dC_dK = (Prices[i + 1, j] - Prices[i - 1, j]) / (strikes_l[i + 1
            dC2_dK2 = (Prices[i + 1, j] - 2 * Prices[i, j] + Prices[i - 1, j
            sig_loc[i, j] = np.sqrt(2 * (dC_dt + q * Prices[i, j] + (r - q)

    print(60 * '=')
    # create a meshgrid of the maturities and strikes
    maturities_, strikes_ = np.meshgrid(maturities_l[1:-1], strikes_l[1:-1])

    sig_loc_ = sig_loc[1:-1, 1:-1]
    print(np.shape(maturities_))
    print(np.shape(sig_loc_))

    # show plot
    fig = plt.figure(figsize=[15, 15])
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(strikes_, maturities_, sig_loc_, cmap='rainbow')#cm.cool
    #ax.scatter(strikes_, maturities_, sig_loc_, label='Local Volatility Sur
    #ax.plot_wireframe(strikes_, maturities_, sig_loc_, label='Local Volatil
    ax.view_init(elev=22, azim=160)
    ax.set_xlabel('Strike')
    ax.set_ylabel('Maturity')
    ax.set_zlabel('Local Volatility')
    ax.set_title(title_)

    plt.show()
```

''' As for the models, the results seem slightly better using with weights inversely proportional to bid-ask spread instead of equal weights, we will plot the local volatility surfaces for both VGSA and Heston model using the parameters obtained with with weights inversely proportional to bid-ask spread.'''

2.1 Local Volatility Surface for Heston model

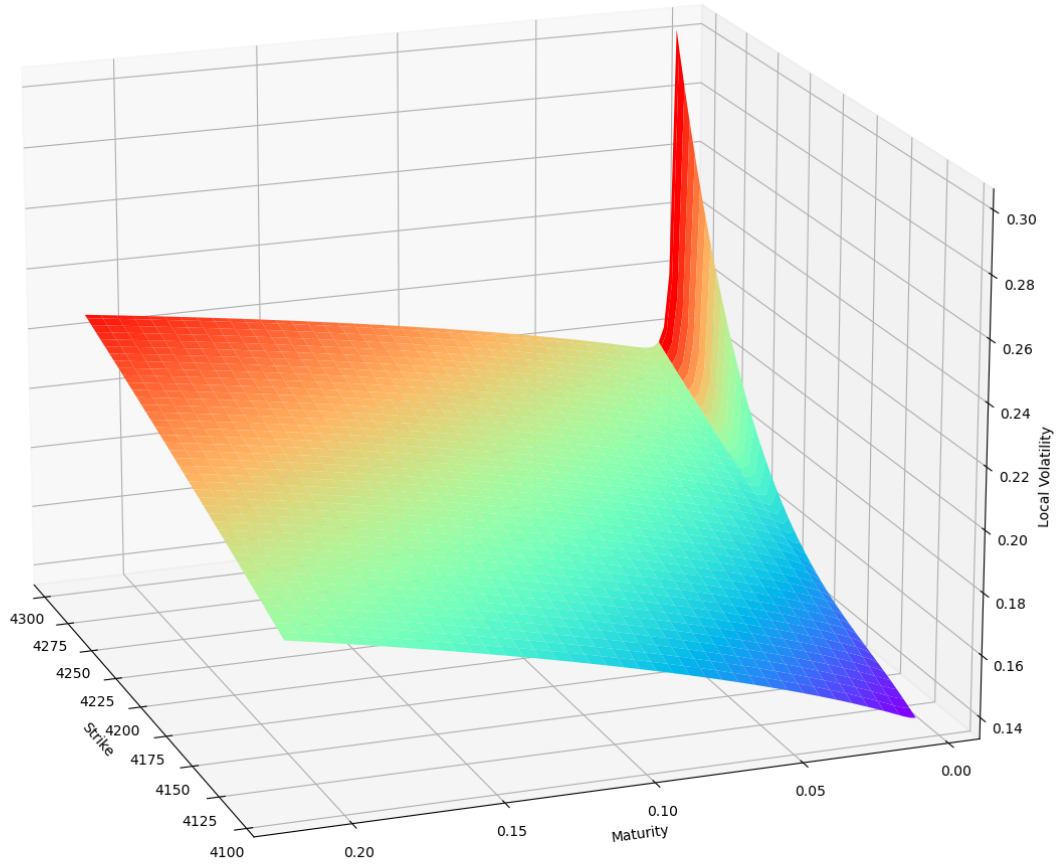
In [73]:
```
params_ = xopt_pc
model_ = 'Heston'
title_ = f'Local Volatility Surface for Heston model (with the parameters xo
vol_surface(params_,strikes,maturities,modelPrices,model_, title_)
```

```
Computing the prices according to the model and the opt-parameters
100%|████████████████████████████████████████████████████████████
████████████████████████████████████████████████████████████
██████████| 86/86 [03:09<00:00,  2.21s/it]
Computing the local volatility surface
```

```
100%|██████████| 84/84 [00:00<00:00, 522.52it/s]
```

```
============================================================
(84, 115)
(84, 115)
```

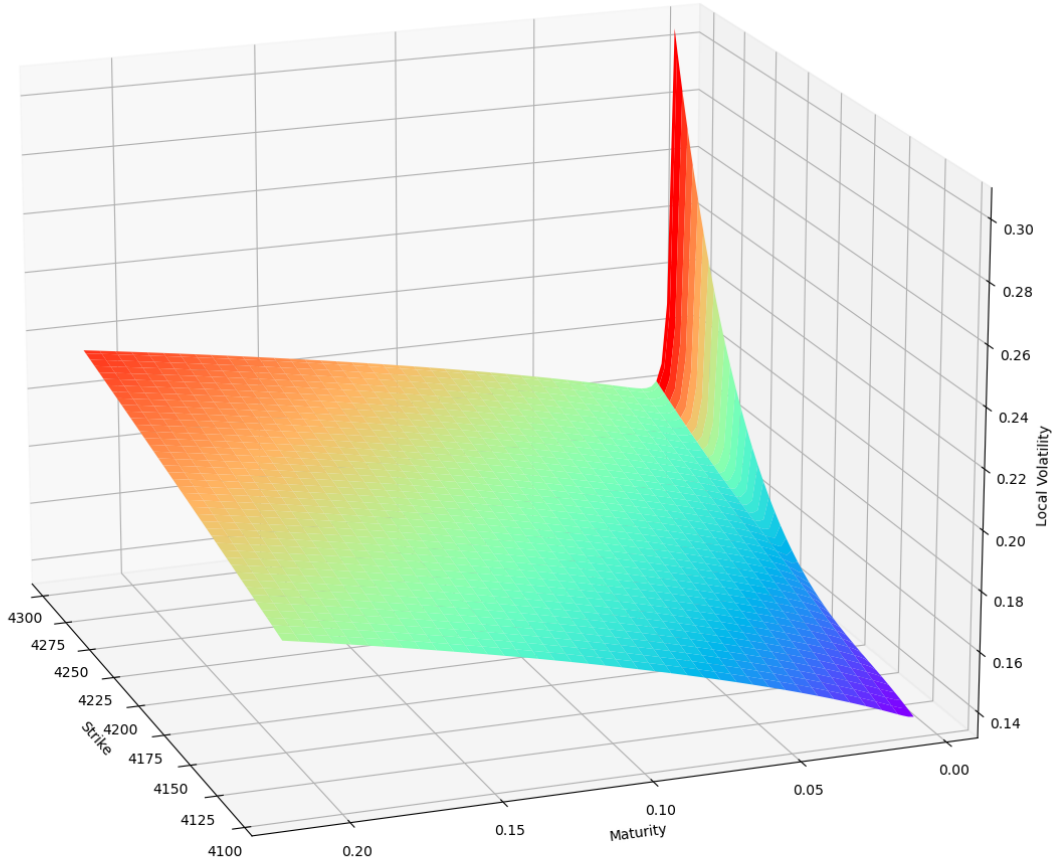
Local Volatility Surface for Heston model (with the parameters xopt_w=[2.35948156 -0.07959283 0.59916615 -1.00043828 -0.95522165])



In [74]:
```
params_ = xopt_pc_w
model_ = 'Heston'
title_ = f'Local Volatility Surface for Heston model (with the parameters xo
vol_surface(params_,strikes,maturities,modelPrices,model_, title_)
```

```
Computing the prices according to the model and the opt-parameters
100%|██████████| 86/86 [03:18<00:00,  2.31s/it]
Computing the local volatility surface
100%|██████████| 84/84 [00:00<00:00, 965.83it/s]
============================================================
(84, 115)
(84, 115)
```

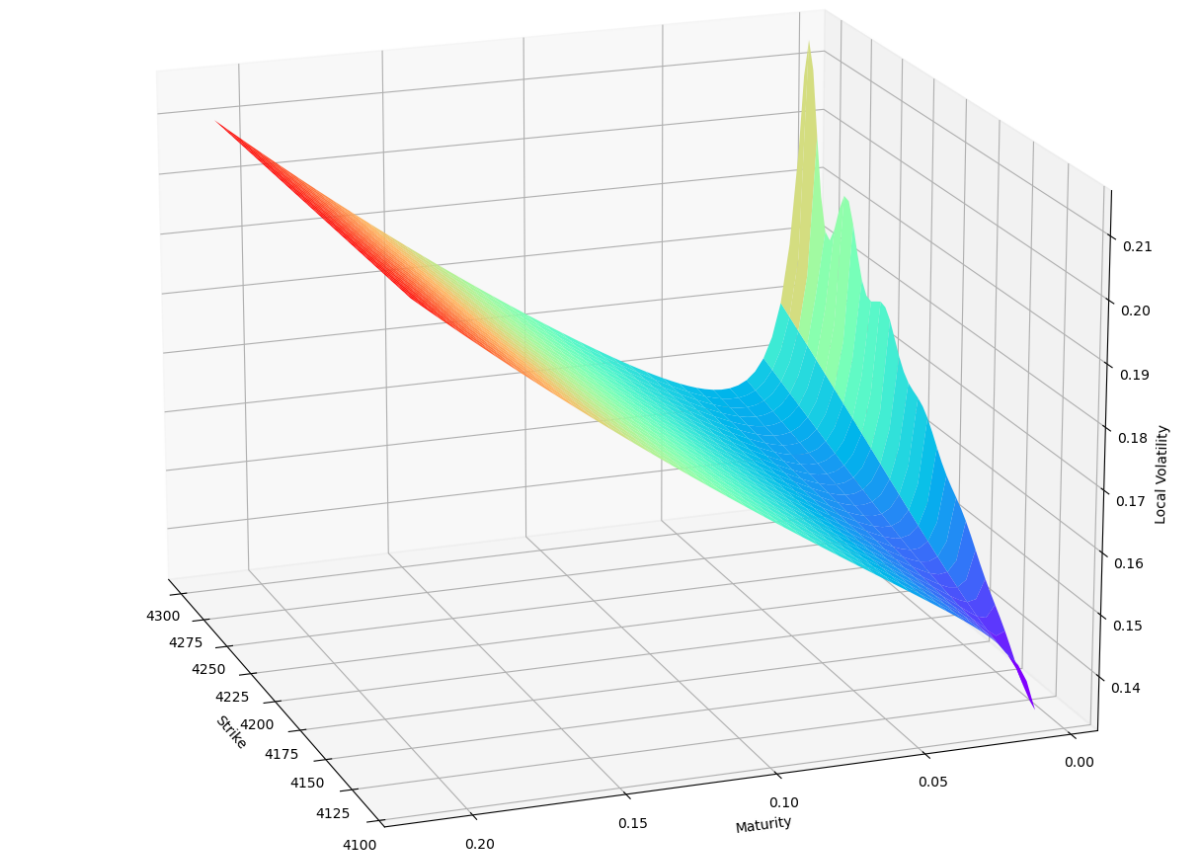
Local Volatility Surface for Heston model (with the parameters xopt_w=[2.35948156 -0.07959283 0.59916615 -1.00043828 -0.95522165])



2.2 Local Volatility Surface for VGSA model

In [81]:
```
params_ = xopt_v_pc
model_ = 'VGSA'
title_ = f'Local Volatility Surface for {model_} model (with the parameters
vol_surface(params_,strikes,maturities,modelPrices,model_, title_)
```

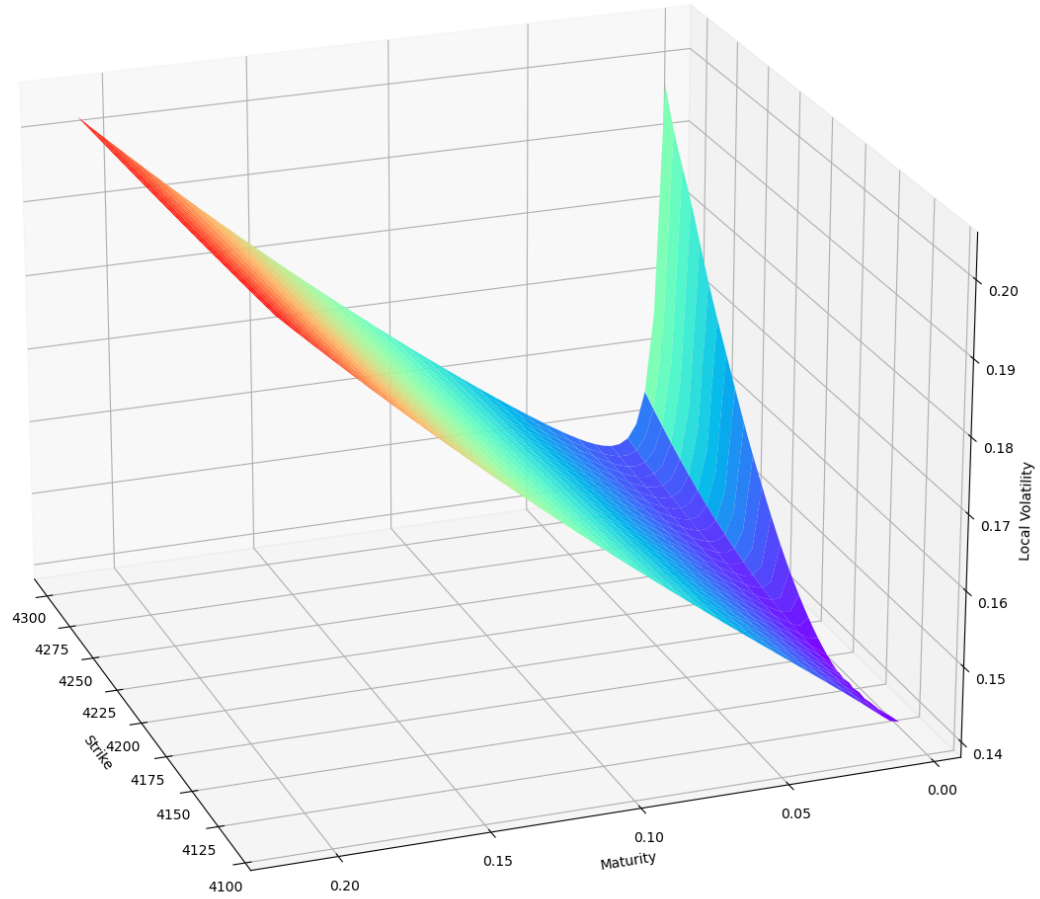
```
Computing the prices according to the model and the opt-parameters
100%|██████████| 53/53 [01:07<00:00,  1.27s/it]
Computing the local volatility surface
100%|██████████| 51/51 [00:00<00:00, 1948.88it/s]
============================================================
(51, 68)
(51, 68)
```



In [83]:
```
params_ = xopt_v_w_pc
model_ = 'VGSA'
title_ = f'Local Volatility Surface for {model_} model (with the parameters
vol_surface(params_,strikes,maturities,modelPrices,model_, title_)
```

```
Computing the prices according to the model and the opt-parameters
100%|██████████| 53/53 [01:07<00:00,  1.27s/it]
Computing the local volatility surface
100%|██████████| 51/51 [00:00<00:00, 2214.20it/s]
============================================================
(51, 68)
(51, 68)
```

Local Volatility Surface for VGSA model (with the parameters xopt_w=[2.48372470e-02 3.16473038e-02 -1.08263283e-04 -3.42577940e+00
 1.56808419e+00 8.84855828e-01])



3. Findings/Observations

The Heston model assumes a constant correlation between the underlying asset and volatility, while the VGSA model allows for a varying correlation. This results in differences in the local volatility surface between the two models.The VGSA model is more flexible than the Heston model in terms of fitting market data. It can capture different shapes and behaviors of the implied volatility surface more accurately. We can see the surface for VGSA is indeed slightly more complex as can be seen for low maturity near the spot price.

We can see the surfaces obtained for the 2 models the convexity of the surface are different. The slightly concave form of the surface at higher maturity for Heston is an element that makes this model more robust in a certain way here whereas VSGA is less optimist at longer term than Heston according its local vol surface. Moreover, the VGSA model may have produced a smoother local volatility surface compared to the Heston model, which can exhibit more volatility and fluctuations.

Additionnal Notes :

Here high local vol values at very low maturity for call options indicate that investors perceive a high probability of significant SP500 stock price movements in the very near future, which translates into high implied volatility for these options. This may be due to impending events such as major corporate announcements, quarterly earnings, macroeconomic events, which are happening in the beginning of the year or other events.