

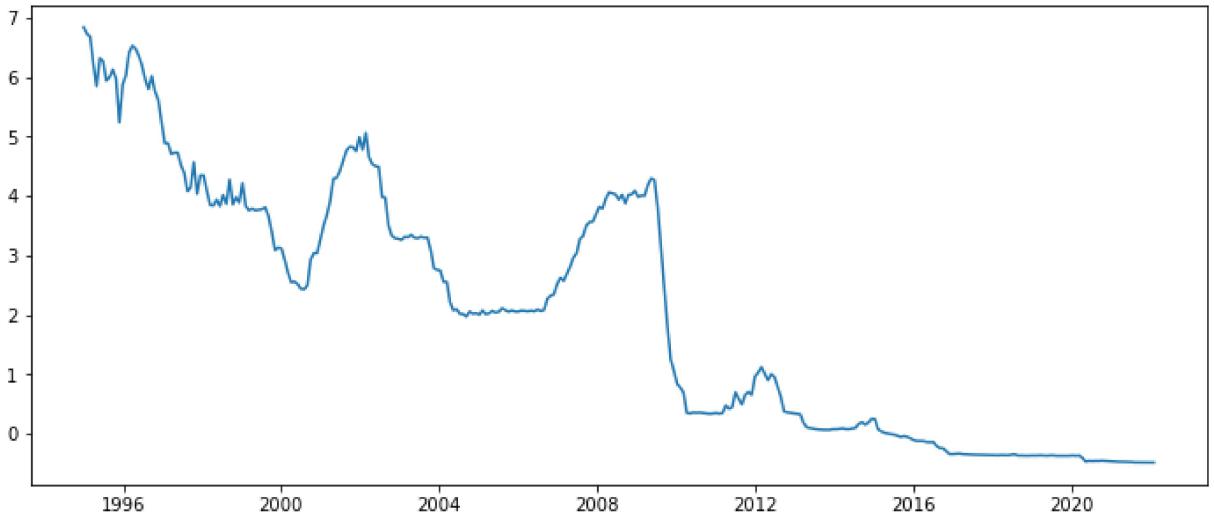
```
In [1]:  
import matplotlib.pyplot as plt  
from mpl_toolkits.mplot3d import Axes3D  
import numpy as np  
import pandas as pd  
import time  
import datetime  
import math  
import QuantLib as ql  
from scipy.optimize import minimize  
  
from initialize import *  
from plotting import *  
from SABR import *  
from Heston import *  
from mixedSABR import *  
from computing_time import *
```

```
In [2]: # Spot rates table and chart (EONIA)  
  
rates = [-0.575, -0.557, -0.549, -0.529, -0.494]  
tenors = [.25, 1, 3, 6, 12]  
  
spot_rates = pd.DataFrame({"Tenors": tenors, "Spot Rate": rates})  
spot_rates.set_index('Tenors')  
  
display(spot_rates)  
  
fig = plt.figure(figsize=(12,5))  
eonia_dates = [datetime.date(1994, 12, 31) + datetime.timedelta(days=30*n) for n in  
plt.plot(eonia_dates, eonia_rates['value'])  
plt.title("EONIA Rates (%) 1996-2021")
```

	Tenors	Spot Rate
0	0.25	-0.575
1	1.00	-0.557
2	3.00	-0.549
3	6.00	-0.529
4	12.00	-0.494

```
Out[2]: Text(0.5, 1.0, 'EONIA Rates (%) 1996-2021')
```

EONIA Rates (%) 1996-2021

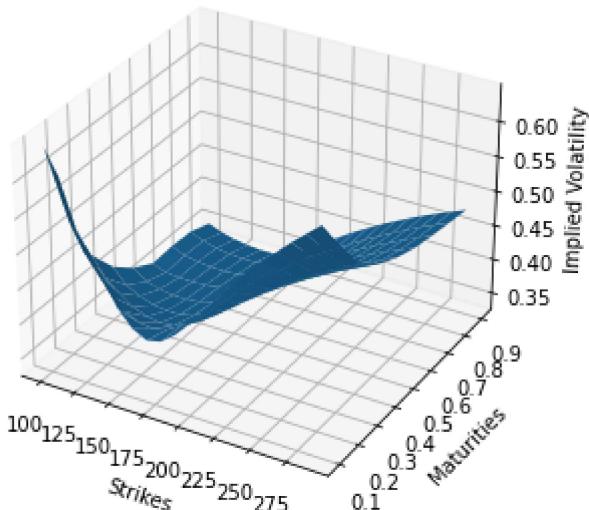


In [3]:

```
# BLACK VOLATILITY SURFACE

title = "Black-Scholes Implied Volatility Surface on {} \n {} to {}".format(data, tod
plot_vol_surface(vol_surface=black_var_surface, plot_strikes=strikes, funct='blackVo
```

Black-Scholes Implied Volatility Surface on COFFEE
August 31st, 2021 to August 12th, 2022



In [4]:

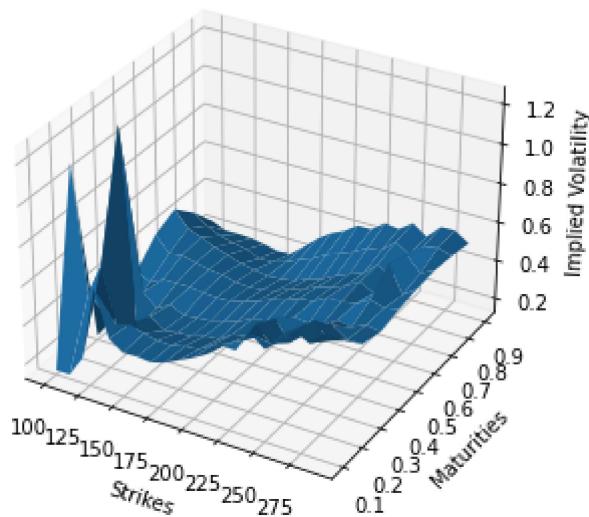
```
# DUPIRE LOCAL VOLATILITY SURFACE (NOT PLOTTABLE)

black_var_surface.setInterpolation("bicubic")
local_vol_handle = ql.BlackVolTermStructureHandle(black_var_surface)

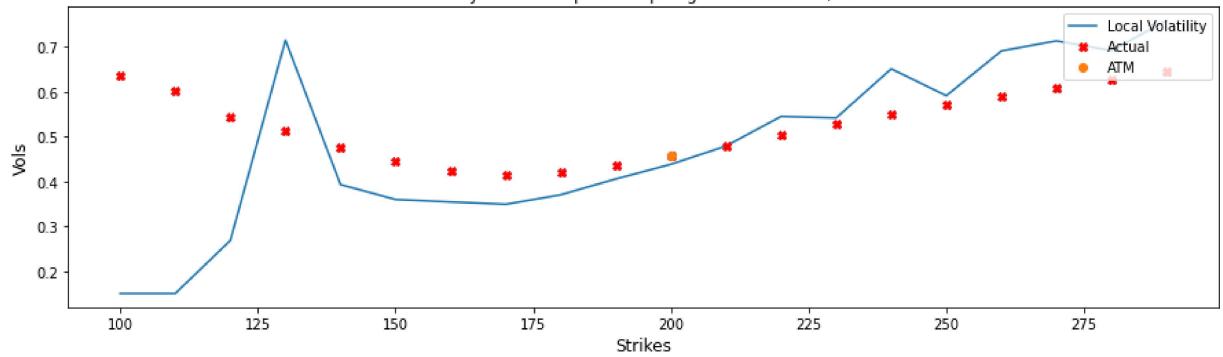
# Local_vol_surface = ql.LocalVolSurface(Local_vol_handle, flat_ts, dividend_ts, spot)
local_vol_surface = ql.NoExceptLocalVolSurface(local_vol_handle, flat_ts, dividend_t

# Plot the Dupire surface ...
local_vol_surface.enableExtrapolation()
plot_vol_surface(local_vol_surface, funct='localVol', title="Dupire Local Volatility
smiles_comparison(local_models=[local_vol_surface], black_volatility=False)
```

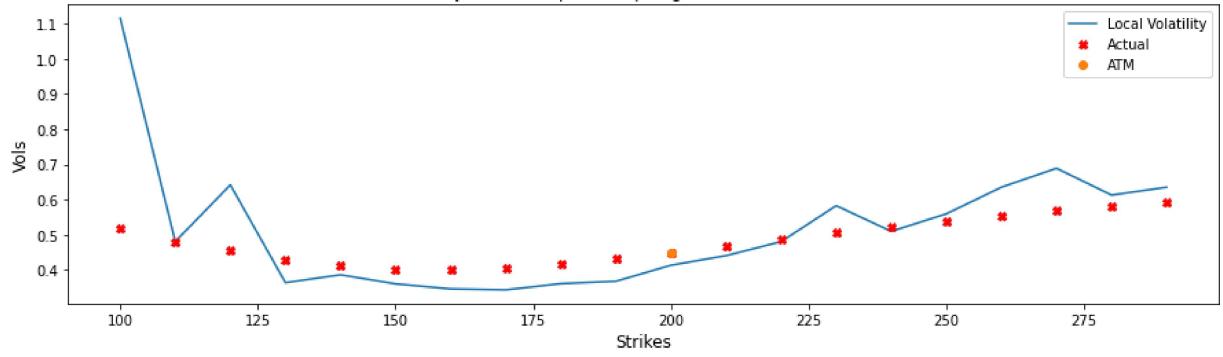
Dupire Local Volatility Surface on Coffee



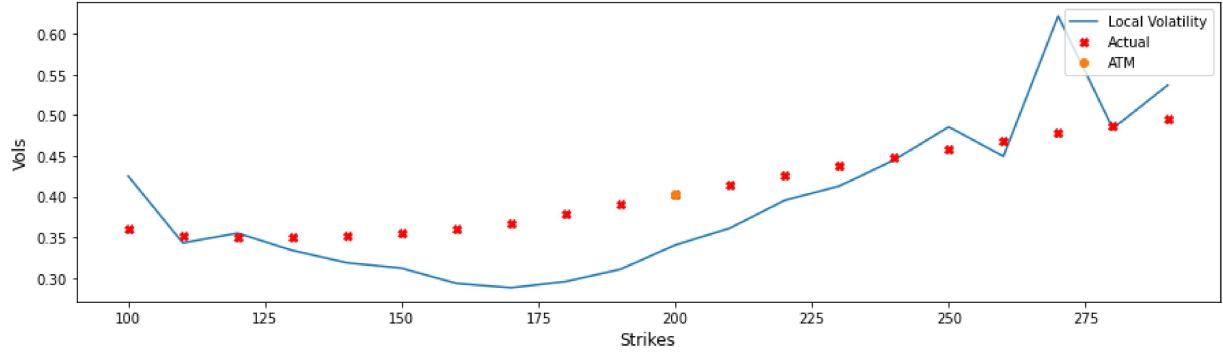
Volatility Smile for options expiring on October 8th, 2021

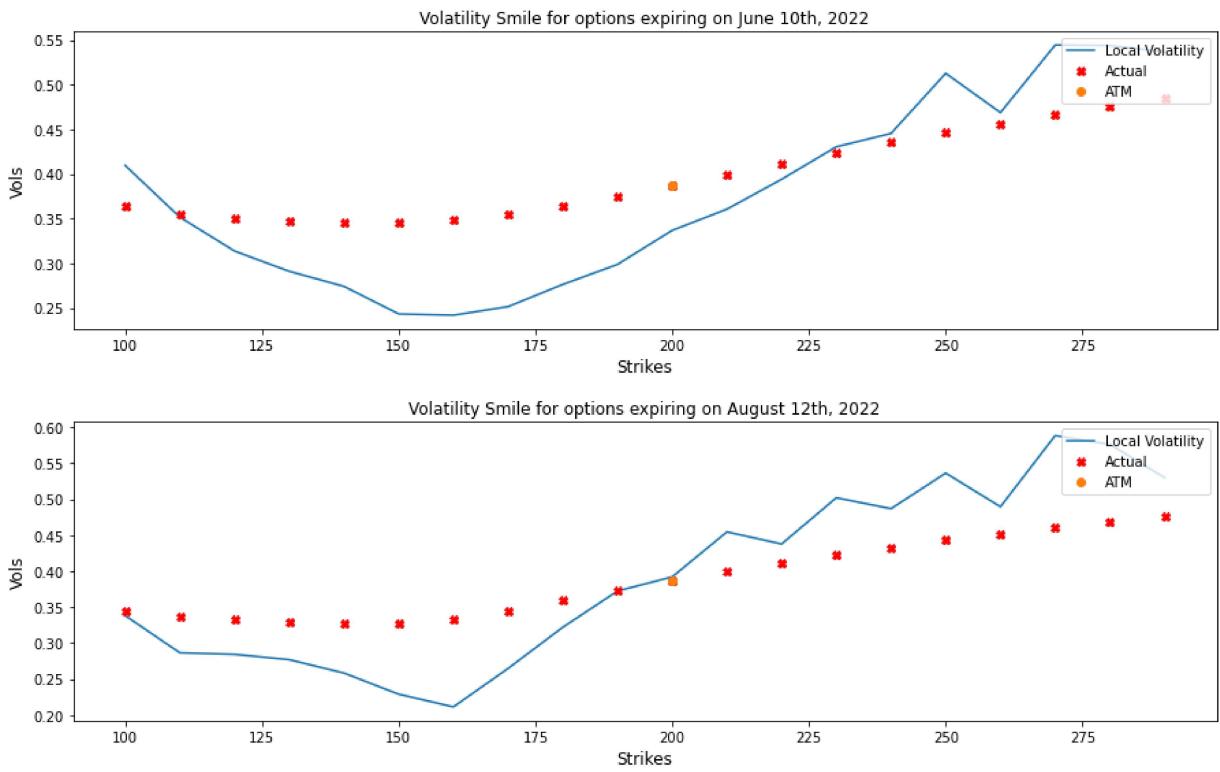


Volatility Smile for options expiring on November 12th, 2021



Volatility Smile for options expiring on April 8th, 2022





In [5]:

```
#HESTON MODEL SURFACE PLOTTING (Levenberg-Marquardt Method)

m1_params, m2_params = (None, None)
if data == "COFFEE":
    m1_params = (0.01, 0.1, 0.3, 0.1, 0.02)
    m2_params = (0.2, 0.9, 0.9, 0.9, -0.19)
elif data == "OIL":
    m2_params = (0.023, 0.009, 1.00, 0.95, 0.2)
    m1_params = (0.15, 0.5, 0.2, 0.7, 0.01)
elif data == "GOLD":
    m1_params = (0.03, 0.3, 0.5, 0.3, 0.04)
    m2_params = (0.01, 0.5, 0.5, 0.1, 0.03)
elif data == "SILVER":
    m2_params = (0.5, 0.5, 1.25, 0.3, 0.00)
    m1_params = (0.01, 0.5, 0.5, 0.1, 0.03)

hestonModel11 = hestonModelSurface(m1_params, label="Heston Model 1, {}".format(data))
hestonModel12 = hestonModelSurface(m2_params, label="Heston Model 2, {}".format(data))

# Use to Calibrate first time the Heston Model
def calibrateHeston():
    def f(params):
        return hestonModelSurface(params).avgError
        # v0, kappa, theta, sigma, rho
    cons = (
        {'type': 'ineq', 'fun': lambda x: x[0] - 0.001},
        {'type': 'ineq', 'fun': lambda x: 2. - x[0]},
        {'type': 'ineq', 'fun': lambda x: x[1] - 0.001},
        {'type': 'ineq', 'fun': lambda x: 2. - x[1]},
        {'type': 'ineq', 'fun': lambda x: x[2] - 0.001},
        {'type': 'ineq', 'fun': lambda x: 2. - x[2]},
        {'type': 'ineq', 'fun': lambda x: x[3] - 0.001},
        {'type': 'ineq', 'fun': lambda x: .999 - x[3]},
        {'type': 'ineq', 'fun': lambda x: .99 - x[4]**2}
    )
    result = minimize(f, m1_params, constraints=cons, method="SLSQP", bounds=((1e-8,
```

```

print(result["x"])

plot_vol_surface(hestonModel0.heston_vol_surface, title="{} Volatility Surface".
plot_vol_surface(hestonModel1.heston_vol_surface, title="{} Volatility Surface".

init_conditions = pd.DataFrame({"theta": [m1_params[0], m2_params[0]], "kappa": [m1_
    "sigma": [m1_params[2], m2_params[2]], "rho": [m1_pa
    "v0": [m1_params[4], m2_params[4]]}, index = ["Model
display(init_conditions.style.set_caption("Heston Model Initial Conditions on ({})".

```

Heston Model Initial Conditions on (Coffee)

	theta	kappa	sigma	rho	v0
Model1	0.010000	0.100000	0.300000	0.100000	0.020000
Model2	0.200000	0.900000	0.900000	0.900000	-0.190000

In [6]:

```

# HESTON Surface Plotting (Model1, Model2)

for model in (hestonModel1, hestonModel2):
    plot_vol_surface(model.heston_vol_surface, title="Heston Volatility Surface for
display(model.errors_data.style.set_caption("{} calibration results".format(mode

fig1 = plt.figure(figsize=plot_size)
plt.plot(model.strks, model.marketValue, label="Market Value")
plt.plot(model.strks, model.modelValue, label="Model Value")
plt.title('Model1: Heston surface Market vs Model Value'); plt.xlabel='strikes';
plt.legend()
fig2 = plt.figure(figsize=plot_size)
plt.plot(model.strks, model.relativeError)
plt.title('Model1: Heston surface Relative Error (%)'); plt.xlabel='strikes'; pl
plt.legend()

```

Heston Model 1, Coffee calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	100.000000	0.336740	0.333489	-0.965443
1	110.000000	0.655519	0.659828	0.657296
2	120.000000	1.218025	1.218920	0.073469
3	130.000000	2.131744	2.125694	-0.283803
4	140.000000	3.511885	3.528806	0.481833
5	150.000000	5.528623	5.607725	1.430781
6	160.000000	8.461670	8.548821	1.029953
7	170.000000	12.562362	12.483658	-0.626503
8	180.000000	17.671092	17.417277	-1.436324
9	190.000000	23.555020	23.228484	-1.386272
10	200.000000	30.054349	29.746388	-1.024680
11	210.000000	27.026128	26.770173	-0.947064
12	220.000000	24.402363	24.226489	-0.720728
13	230.000000	22.138513	22.028049	-0.498968
14	240.000000	20.156662	20.110653	-0.228258

	Strikes	Market Value	Model Value	Relative Error (%)
15	250.000000	18.421029	18.425992	0.026945
16	260.000000	16.889495	16.936752	0.279800
17	270.000000	15.548853	15.613419	0.415246
18	280.000000	14.348527	14.432189	0.583068
19	290.000000	13.272169	13.373561	0.763947

Heston Model 1,
Coffee parameters
output

	Value
v0	1.230783
kappa	0.000000
theta	0.623021
sigma	0.618250
rho	0.162907
avgError	0.693019

No handles with labels found to put in legend.

Heston Model 2, Coffee calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	100.000000	0.336740	0.333489	-0.965550
1	110.000000	0.655519	0.659827	0.657200
2	120.000000	1.218025	1.218919	0.073379
3	130.000000	2.131744	2.125693	-0.283892
4	140.000000	3.511885	3.528803	0.481740
5	150.000000	5.528623	5.607719	1.430680
6	160.000000	8.461670	8.548812	1.029845
7	170.000000	12.562362	12.483644	-0.626612
8	180.000000	17.671092	17.417259	-1.436429
9	190.000000	23.555020	23.228460	-1.386370
10	200.000000	30.054349	29.746362	-1.024769
11	210.000000	27.026128	26.770144	-0.947174
12	220.000000	24.402363	24.226457	-0.720858
13	230.000000	22.138513	22.028016	-0.499119
14	240.000000	20.156662	20.110619	-0.228430
15	250.000000	18.421029	18.425957	0.026752
16	260.000000	16.889495	16.936716	0.279587
17	270.000000	15.548853	15.613382	0.415013
18	280.000000	14.348527	14.432153	0.582815

Strikes	Market Value	Model Value	Relative Error (%)
---------	--------------	-------------	--------------------

19 290.000000 13.272169 13.373525 0.763674

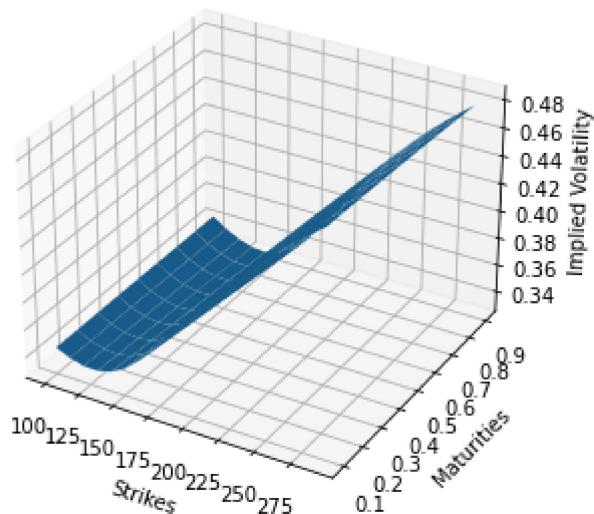
Heston Model 2,
Coffee parameters
output

Value

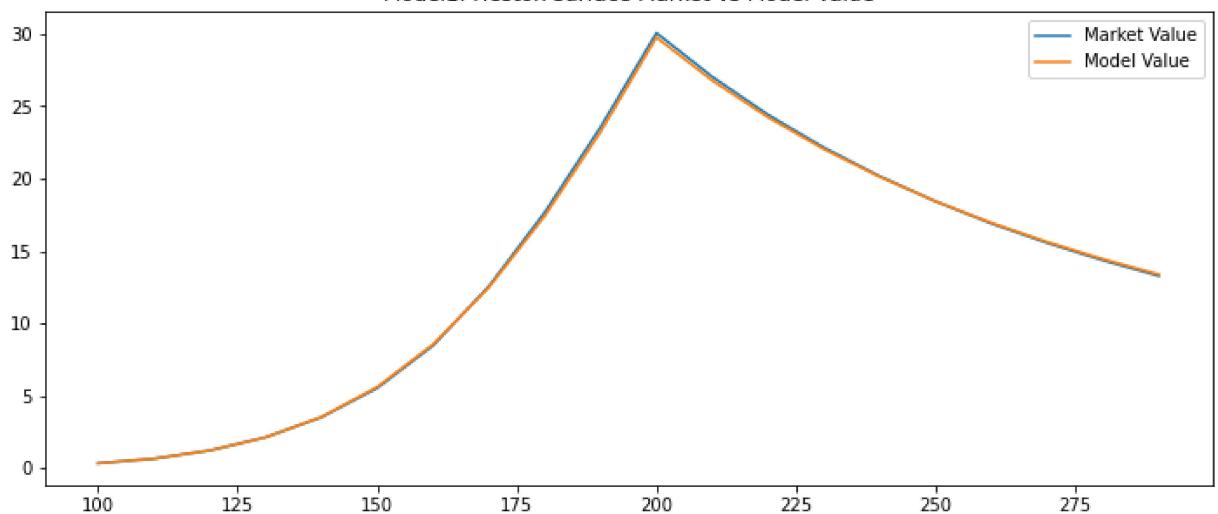
v0	0.939937
kappa	0.000000
theta	0.623019
sigma	0.618249
rho	0.162907
avgError	0.692994

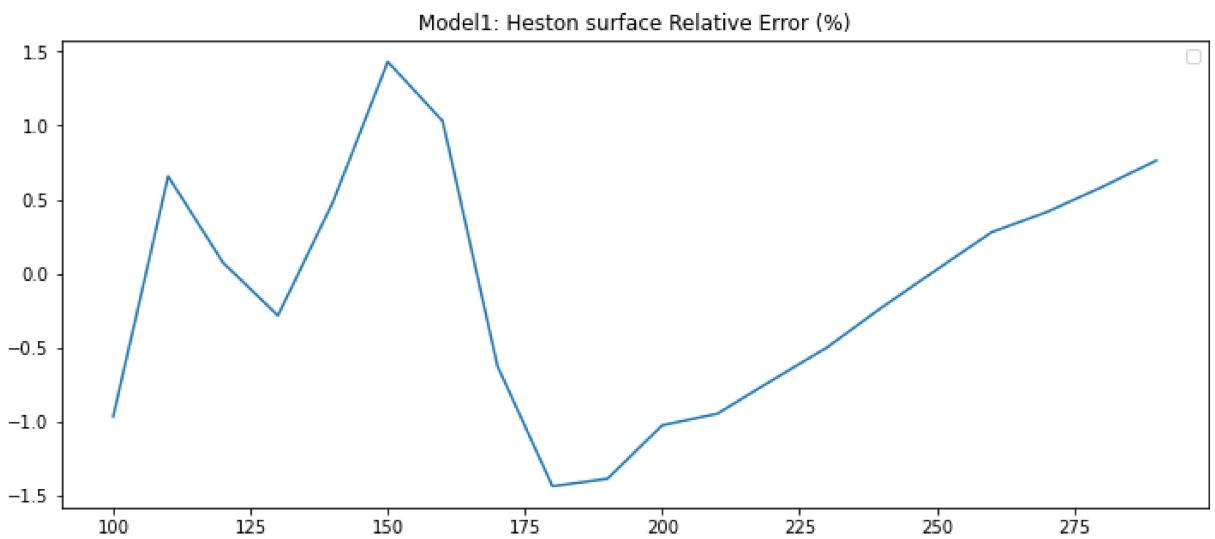
No handles with labels found to put in legend.

Heston Volatility Surface for Heston Model 1, Coffee

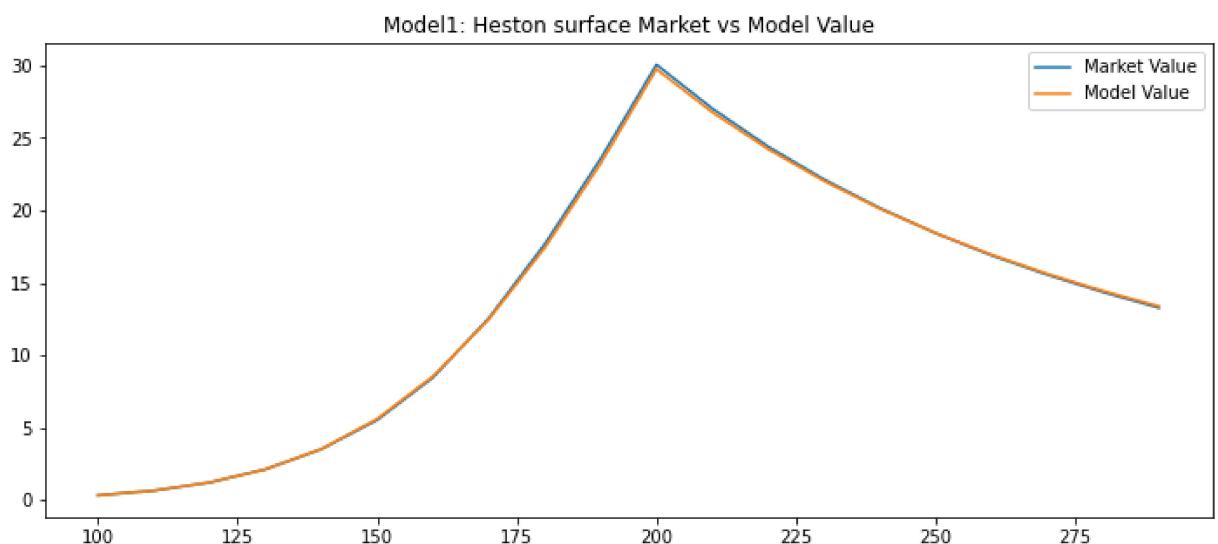
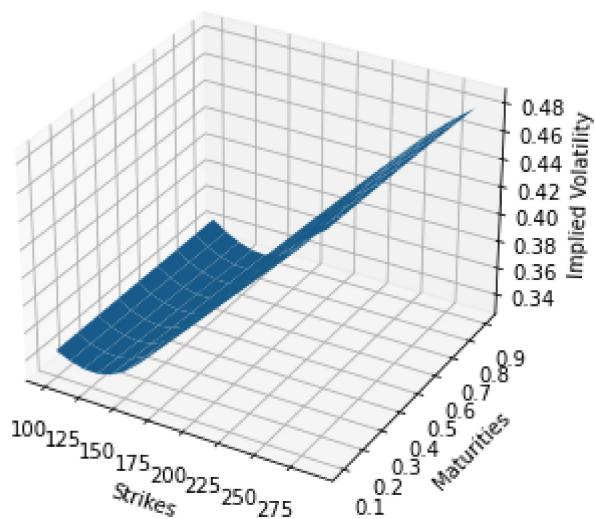


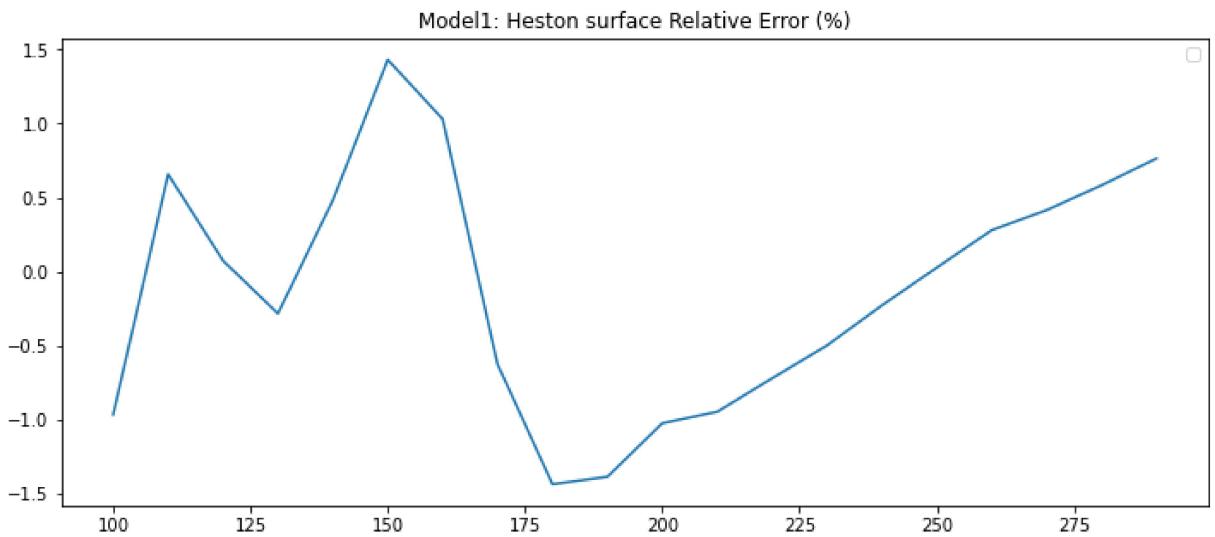
Model1: Heston surface Market vs Model Value





Heston Volatility Surface for Heston Model 2, Coffee

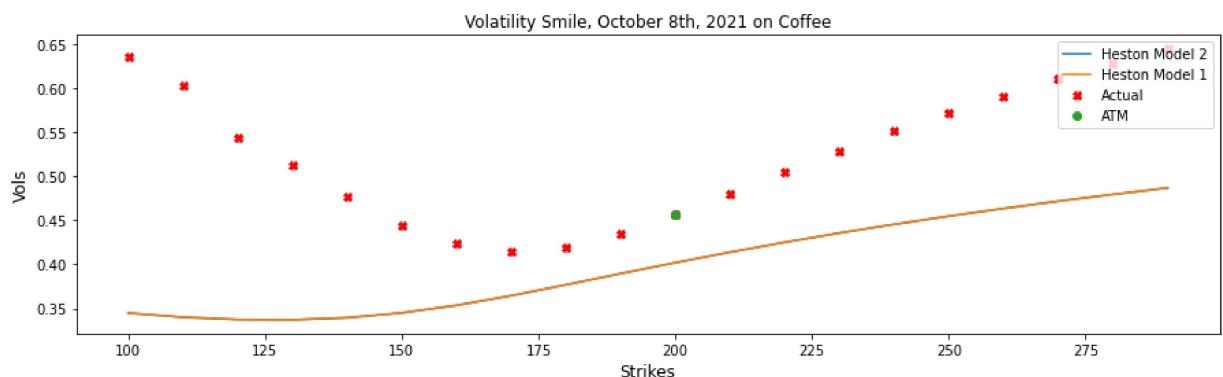
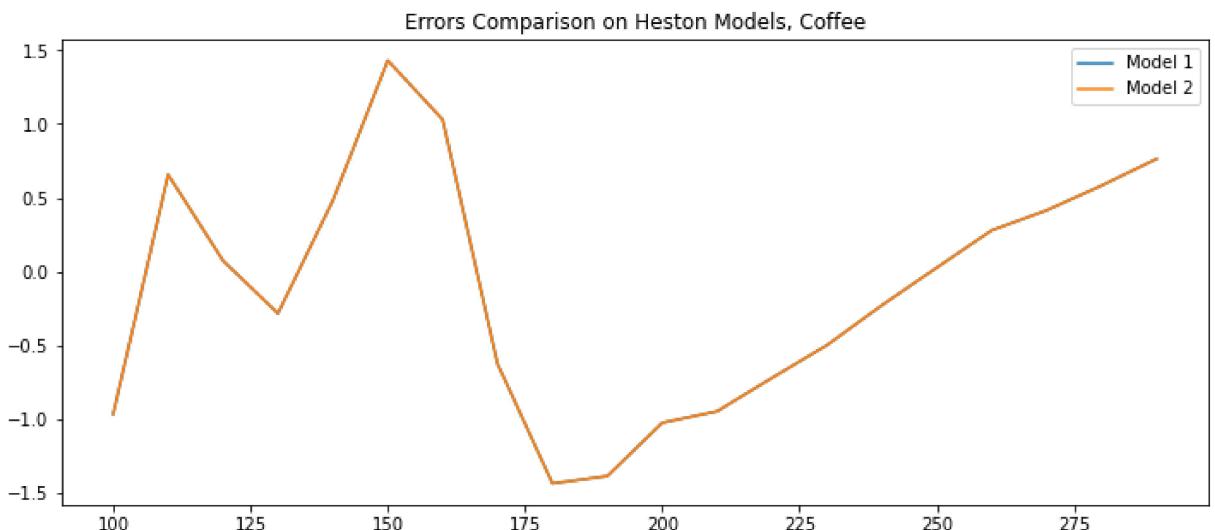


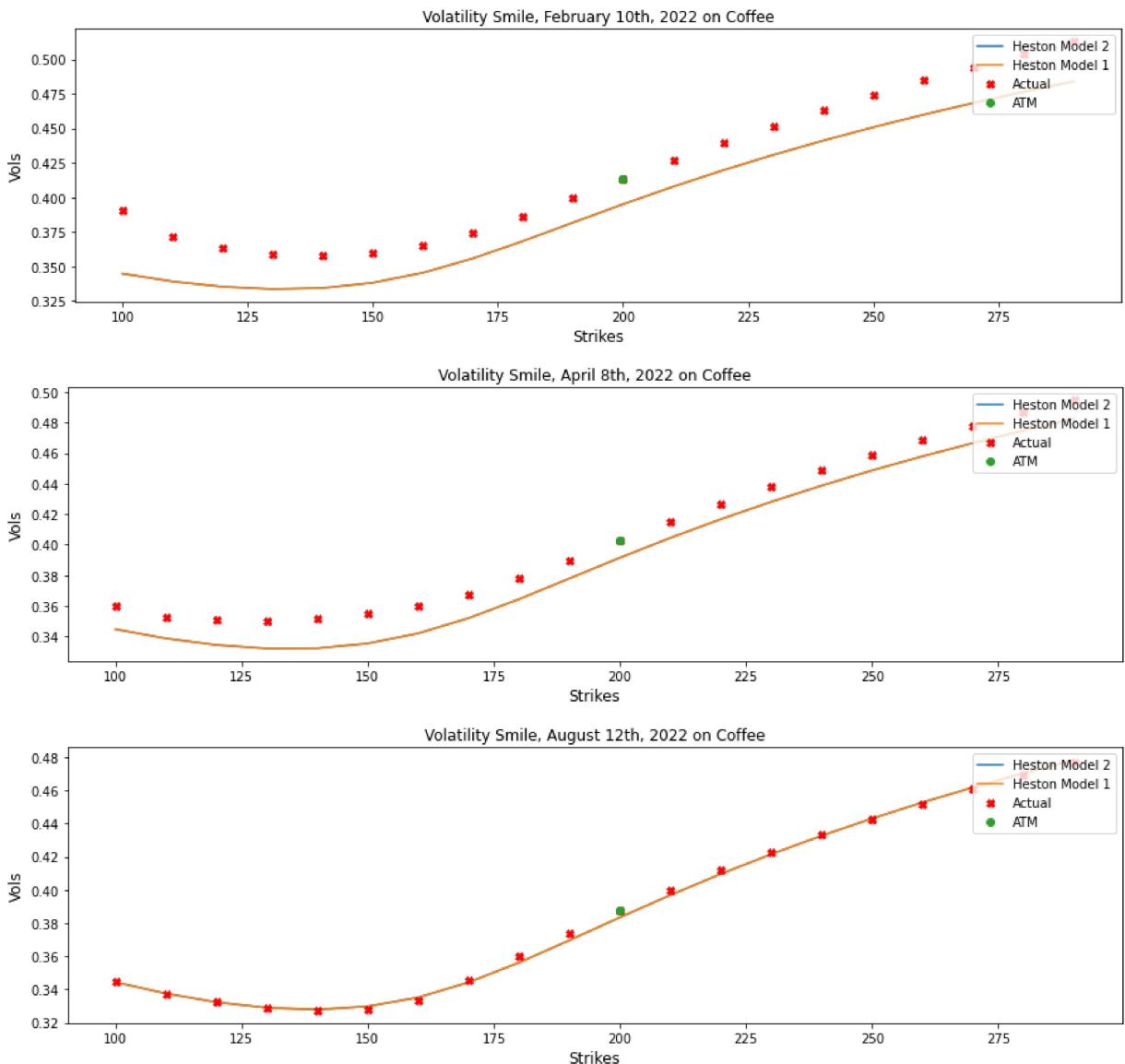


In [7]:

```
# Relative error comparison
plt.figure(figsize=plot_size)
plt.plot(hestonModel1.strks, hestonModel1.relativeError, label="Model 1")
plt.plot(hestonModel2.strks, hestonModel2.relativeError, label="Model 2")
plt.title("Errors Comparison on Heston Models, {}".format(data_label));
plt.legend()

# Volatility smiles comparison
tenors = [dates[round((len(dates)-1) * x)] for x in (.2, .5, .75, 1)]
for tenor in tenors:
    l = [
        ([hestonModel2.hestон_vol_surface.blackVol(tenor, s) for s in strikes], "Hes"
         ([hestonModel1.hestон_vol_surface.blackVol(tenor, s) for s in strikes], "Hes"
          ])
    plot_smile(tenor, l, market=True)
```





In [8]:

```

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import time
import datetime
import math
import QuantLib as ql
from scipy.optimize import minimize

from initialize import *
from plotting import *

# CALIBRATE SABR VOLATILITY SURFACE

volMatrix = ql.Matrix(len(strikes), len(dates))

for i in range(len(vols)):
    for j in range(len(vols[i])):
        volMatrix[j][i] = vols[i][j]

black_var_surface = ql.BlackVarianceSurface(
    today, calendar, dates, strikes, volMatrix, day_count)
black_var_surface.enableExtrapolation()

class SABRSmile:

```

```

def __init__(self, date, marketVols, shift=0, beta=1, method="normal", strikes=s
    self.date = date
    self.expiryTime = round((self.date - today)/365, 6)
    self.marketVols = marketVols
    self.shift = shift
    self.strikes=strikes;
    self.fwd = fwd
    self.forward_price = self.fwd * \
        math.exp(rate.value() * self.expiryTime)
    self.zero_rho = zero_rho
    self.alpha, self.beta, self.nu, self.rho = (
        .1, beta, 0., 0. if self.zero_rho else .1)
    self.method = method
    self.newVols = None
    self.error = None

def initialize(self):
    # alpha, beta, nu, rho
    cons = (
        {'type': 'ineq', 'fun': lambda x: x[0] - 0.001},
        {'type': 'eq', 'fun': lambda x: x[1] - self.beta},
        {'type': 'ineq', 'fun': lambda x: x[2] - .001},
        {'type': 'ineq', 'fun': lambda x: .99 - x[3]**2},
    )

    x = self.set_init_conds()

    result = minimize(self.f, x, constraints=cons, method="SLSQP", bounds=((
        (1e-8, None), (0, 1), (1e-8, None), (-.999, .999)))
    self.error = result['fun']
    [self.alpha, self.beta, self.nu, self.rho] = result['x']

    self.newVols = [self.vols_by_method(
        strike, self.alpha, self.beta, self.nu, self.rho) for strike in self.str

def set_init_conds(self):
    return [self.alpha, self.beta, self.nu, self.rho]

def vols_by_method(self, strike, alpha, beta, nu, rho):
    if self.method == "floc'h-kennedy":
        return ql.sabrFlochKennedyVolatility(strike, self.forward_price, self.ex
    elif self.shift != 0:
        return ql.shiftedSabrVolatility(strike, self.forward_price, self.expiryT
    else:
        return ql.sabrVolatility(strike, self.forward_price, self.expiryTime, al

def f(self, params):

    alpha, beta, nu, rho = params

    vols = np.array([self.vols_by_method(
        strike, alpha, beta, nu, rho) for strike in self.strikes])

    self.error = ((vols - np.array(self.marketVols))**2).mean() ** .5

    return self.error

class SABRVolatilitySurface:
    def __init__(self, strks=strikes, method="normal", beta=1, shift=0, fwd=current_
        self.method = method
        self._beta = beta
        self.shift = shift
        self.strikes = strks
        self.fwd = fwd

```

```

        self.label = label
        self.zero_rho = zero_rho

        self.initialize()

    def initialize(self):
        self.vol_surface_vector, self.errors, self.smiles, self.alpha, self.beta, se
        ], [], [], [], [], []
        self.SABRVolMatrix, self.SABRVolDiffMatrix = (
            ql.Matrix(len(self.strikes), len(dates)), ql.Matrix(len(self.strikes), 1

    for i, d in enumerate(dates):

        v = []
        for j in range(len(self.strikes)):
            if self.strikes[i] in strikes:
                v.append(vols[i][j])

        volSABR = SABRSmile(date=d, beta=self._beta, strikes=self.strikes, marke
                           method=self.method, fwd=self.fwd, zero_rho=self.zero_rho)
        volSABR.initialize()

        self.alpha.append(volSABR.alpha)
        self.beta.append(volSABR.beta)
        self.nu.append(volSABR.nu)
        self.rho.append(volSABR.rho)

        self.errors.append(volSABR.error)

        smile = volSABR.newVols

        self.vol_surface_vector.extend(smile)
        self.smiles.append(volSABR)

        # constructing the SABRVolatilityMatrix
        for j in range(len(smile)):
            self.SABRVolMatrix[j][i] = smile[j]
            self.SABRVolDiffMatrix[j][i] = (
                smile[j] - v[j]) / v[j]

        self.vol_surface = ql.BlackVarianceSurface(
            today, calendar, dates, self.strikes, self.SABRVolMatrix, day_count)
        self.vol_surface.enableExtrapolation()

    def to_data(self):
        d = {'alpha': self.alpha, 'beta': self.beta,
              'nu': self.nu, 'rho': self.rho}
        return pd.DataFrame(data=d, index=dates)

# Backbone modelling for SABR
def SABR_backbone_plot(beta=1, bounds=None, shift=0, strikes=strikes, fixes=(.95, 1,
1 = []
for i in fixes:
    vol_surface = SABRVolatilitySurface(
        method="normal", shift=current_price*shift, beta=beta, fwd=current_price
    SABR_vol_surface = ql.BlackVarianceSurface(
        today, calendar, dates, strikes, vol_surface.SABRVolMatrix, day_count)
    SABR_vol_surface.enableExtrapolation()

    l.append(([SABR_vol_surface.blackVol(tenor, s)
              for s in strikes], "fwd = {}".format(current_price * i)))

plot_smile(tenor, l, bounds=bounds, market=False,
           title="backbone, beta = {}, {}".format(vol_surface.beta[0], tenor))

```

```

def SABRComparison(methods, title="", display=False):
    fig, axs = plt.subplots(2, 2, figsize=plot_size)
    plt.subplots_adjust(left=None, bottom=None, right=None,
                        top=1.5, wspace=None, hspace=None)

    for method in methods:
        lbl = "beta={}".format(method.beta[1])
        axs[0, 0].plot(maturities, method.alpha, label=lbl)
        axs[0, 0].set_title('{}: Alpha'.format(title))
        axs[0, 0].set(xlabel='maturities', ylabel='value')
        axs[0, 0].legend()
        axs[1, 0].plot(maturities, method.nu, label=lbl)
        axs[1, 0].set_title('{}: Nu'.format(title))
        axs[1, 0].set(xlabel='maturities', ylabel='value')
        axs[1, 0].legend()
        axs[0, 1].plot(maturities, method.rho, label=lbl)
        axs[0, 1].set_title('{}: Rho'.format(title))
        axs[0, 1].set(xlabel='maturities', ylabel='value')
        axs[0, 1].legend()
        axs[1, 1].plot(maturities, method.errors, label=lbl)
        axs[1, 1].set_title('{}: MSE'.format(title))
        axs[1, 1].set(xlabel='maturities', ylabel='value')
        axs[1, 1].legend()

    if display:
        method_df = method.to_data()
        display(method_df.style.set_caption("SABR, {}".format(lbl)))

    plot_vol_surface(method.vol_surface, title="{}".format(method.label))

smiles_comparison(methods)

```

In [9]:

```

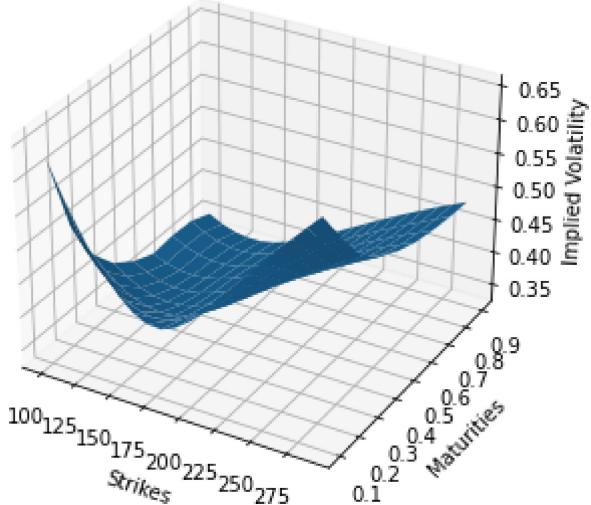
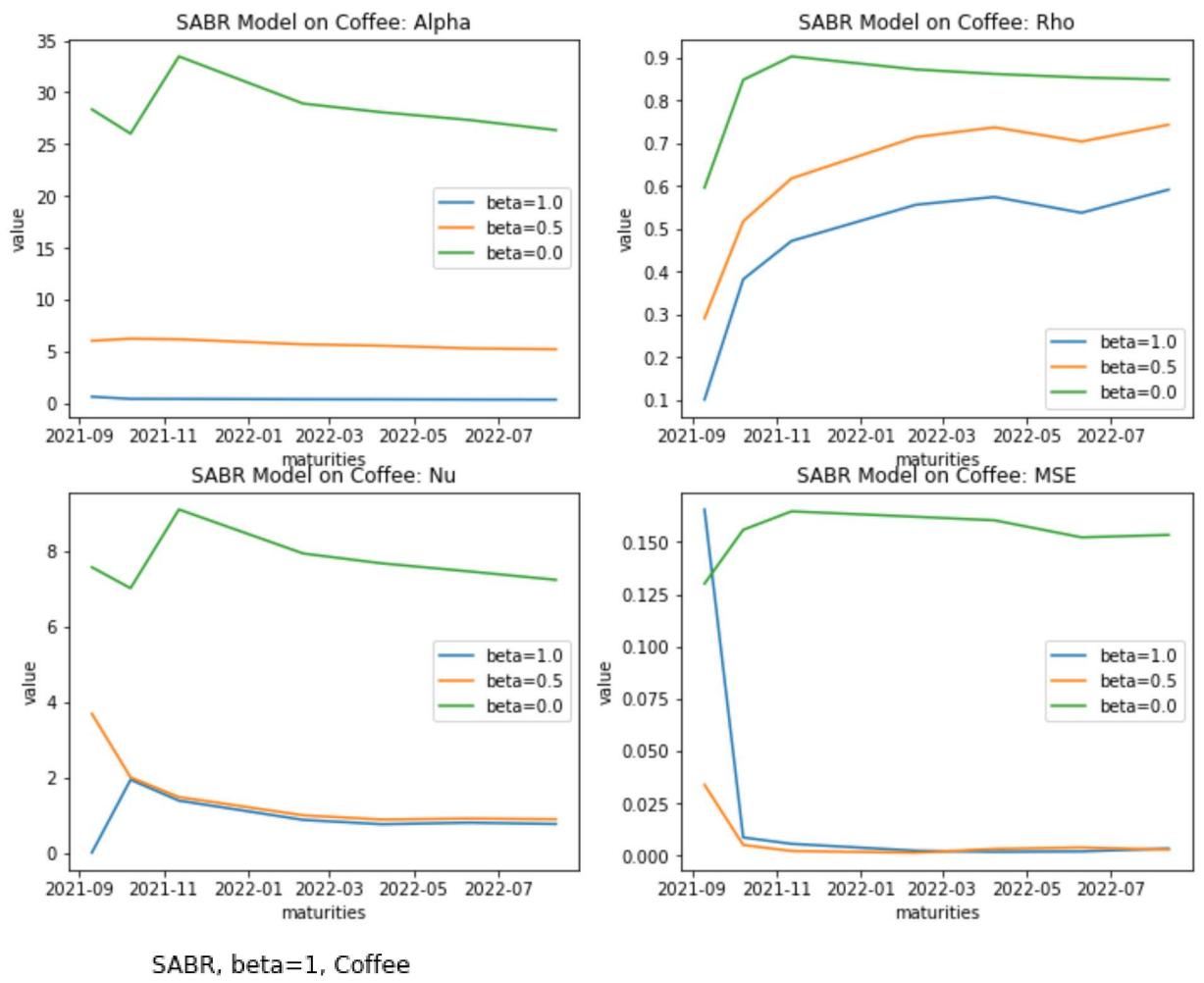
# SABR Volatility model

strks = strikes
SABR_beta1 = SABRVolatilitySurface(beta=1, shift=0, strks=strks, label="SABR, beta=1")
SABR_beta5 = SABRVolatilitySurface(beta=.5, shift=0, strks=strks, label="SABR, beta=.5")
SABR_beta0 = SABRVolatilitySurface(beta=.0, shift=0, strks=strks, label="Normal SABR")

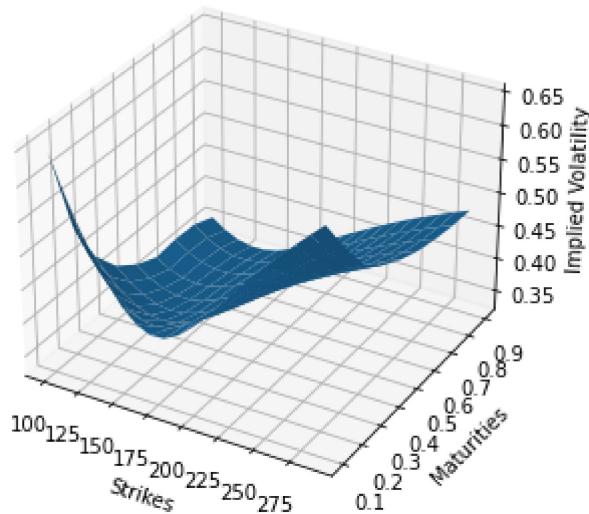
SABRComparison([SABR_beta1, SABR_beta5, SABR_beta0], title="SABR Model on {}".format

```

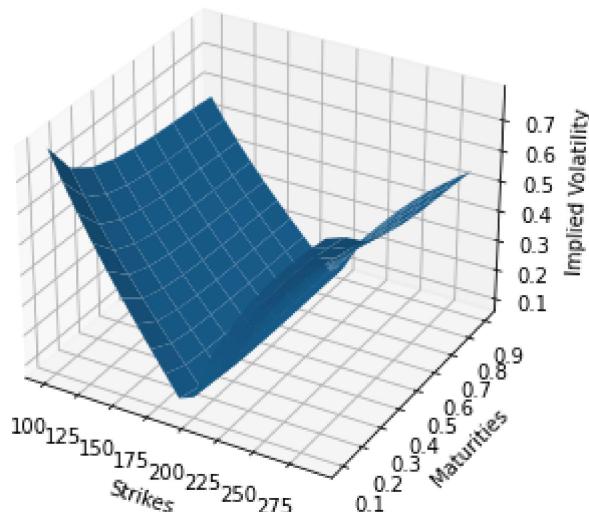
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWarning: Values in x were outside bounds during a minimize step, clipping to bounds
 warnings.warn("Values in x were outside bounds during a "



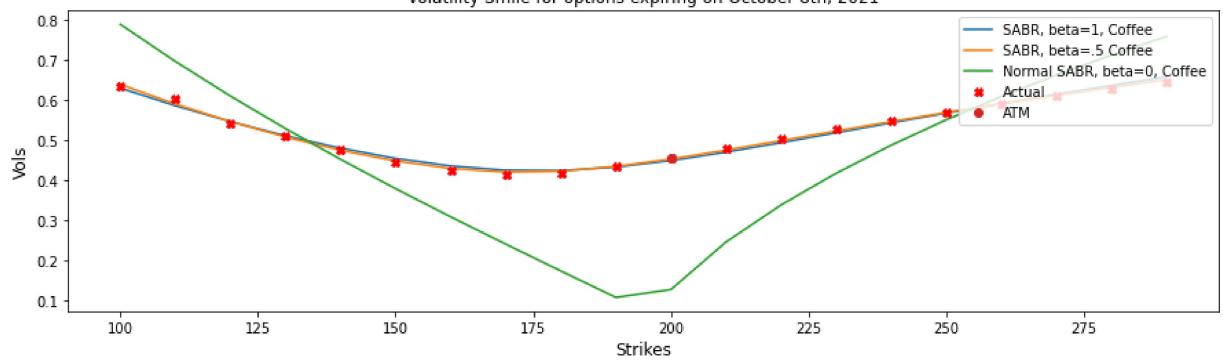
SABR, beta=.5 Coffee



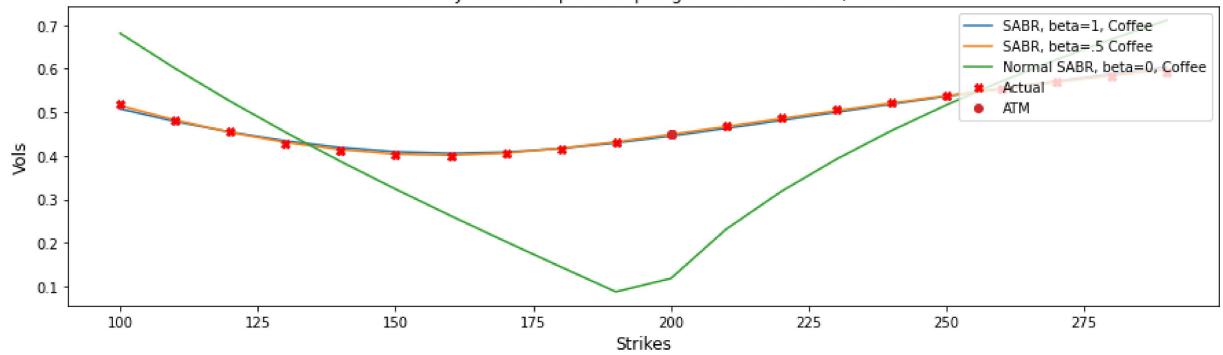
Normal SABR, beta=0, Coffee

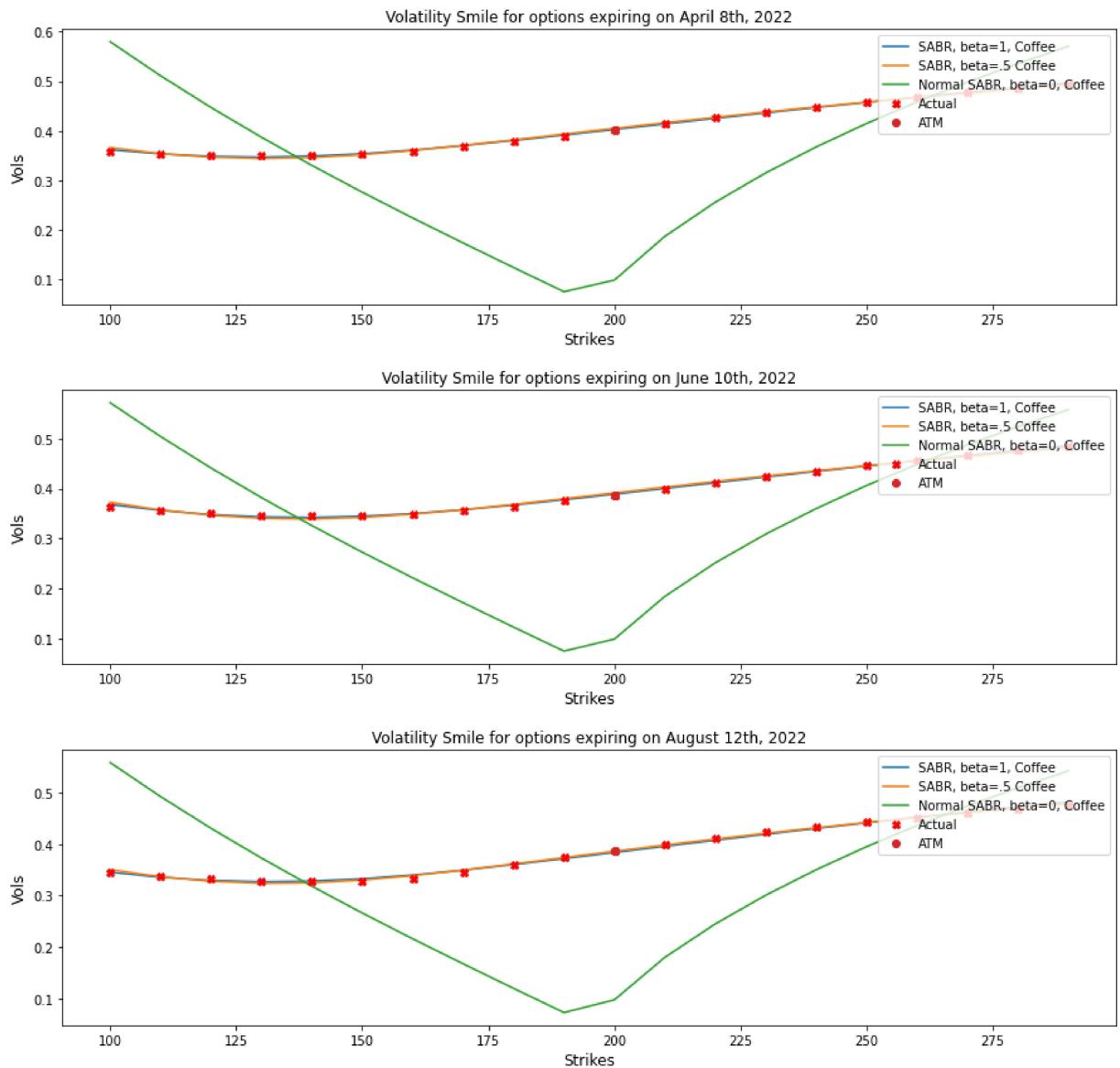


Volatility Smile for options expiring on October 8th, 2021



Volatility Smile for options expiring on November 12th, 2021





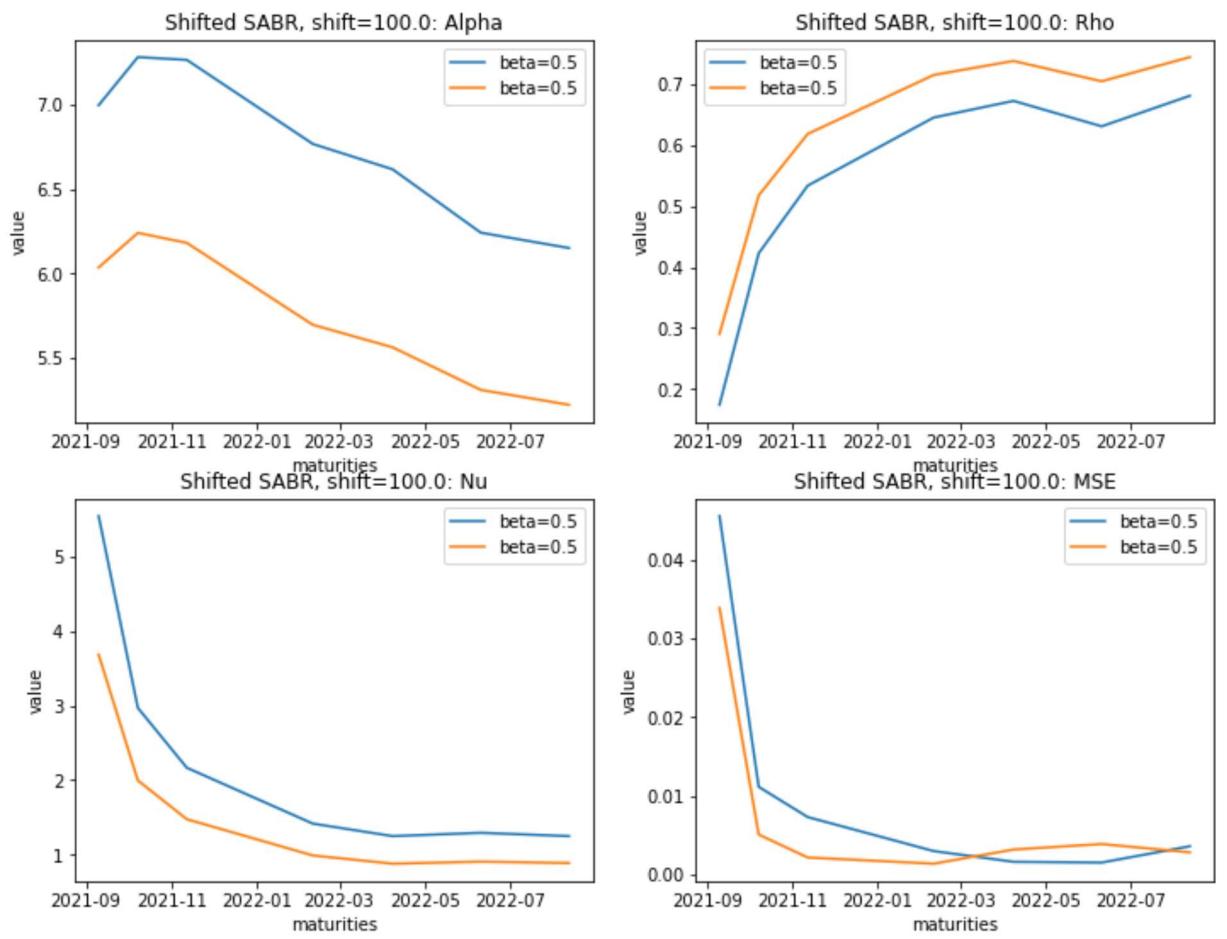
In [10]:

```
# Shifted SABR Volatility model

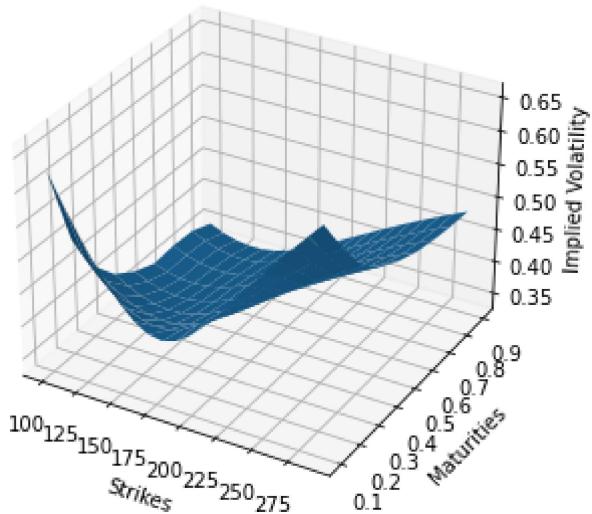
shft = .50 * current_price
shiftedSABR_beta1 = SABRVolatilitySurface(beta=1, shift=shft, strks=strikes, label="SABR, beta=1, Coffee")
shiftedSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=shft, strks=strikes, label="SABR, beta=.5 Coffee")
shiftedSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=shft, strks=strikes, label="Normal SABR, beta=0, Coffee")

SABRComparison([shiftedSABR_beta5, SABR_beta5], title="Shifted SABR, shift={}.format(shft)")
SABRComparison([shiftedSABR_beta0, SABR_beta0], title="Shifted SABR, shift={}.format(shft))
```

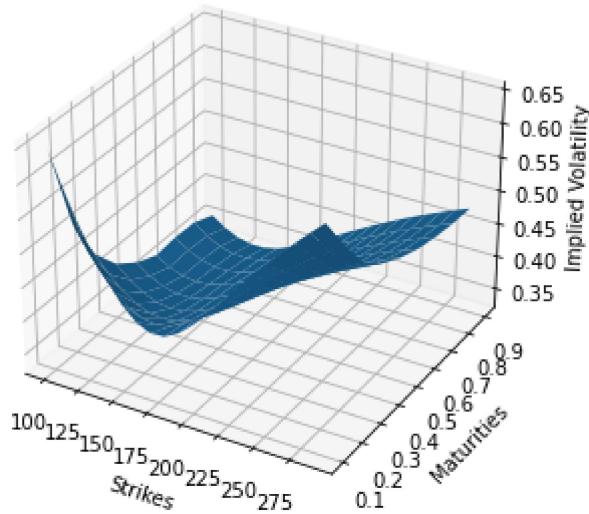
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWarning: Values in x were outside bounds during a minimize step, clipping to bounds
 warnings.warn("Values in x were outside bounds during a "



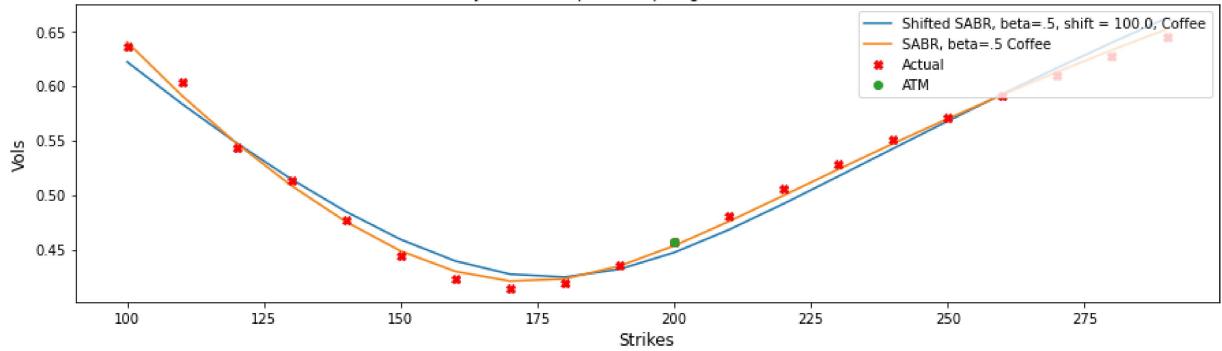
Shifted SABR, beta=.5, shift = 100.0, Coffee



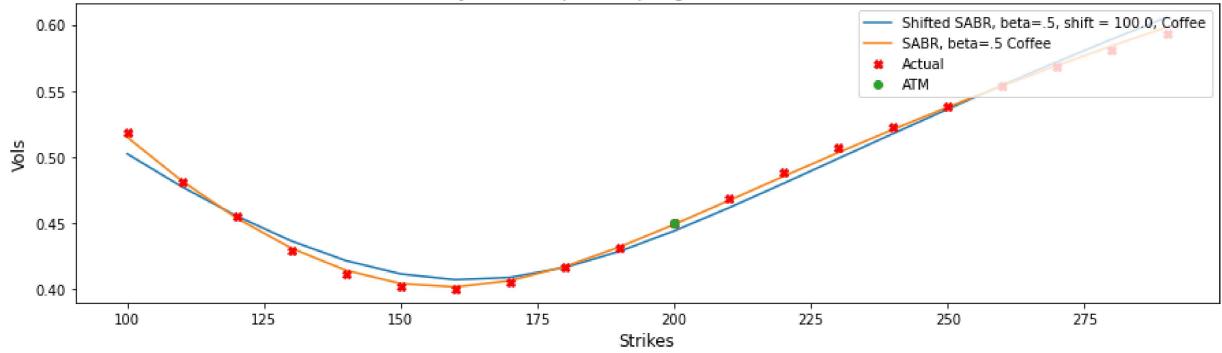
SABR, beta=.5 Coffee



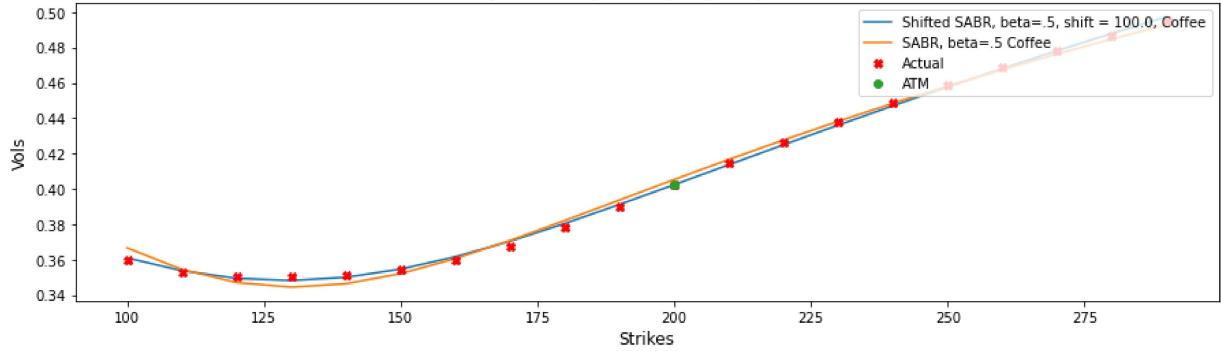
Volatility Smile for options expiring on October 8th, 2021

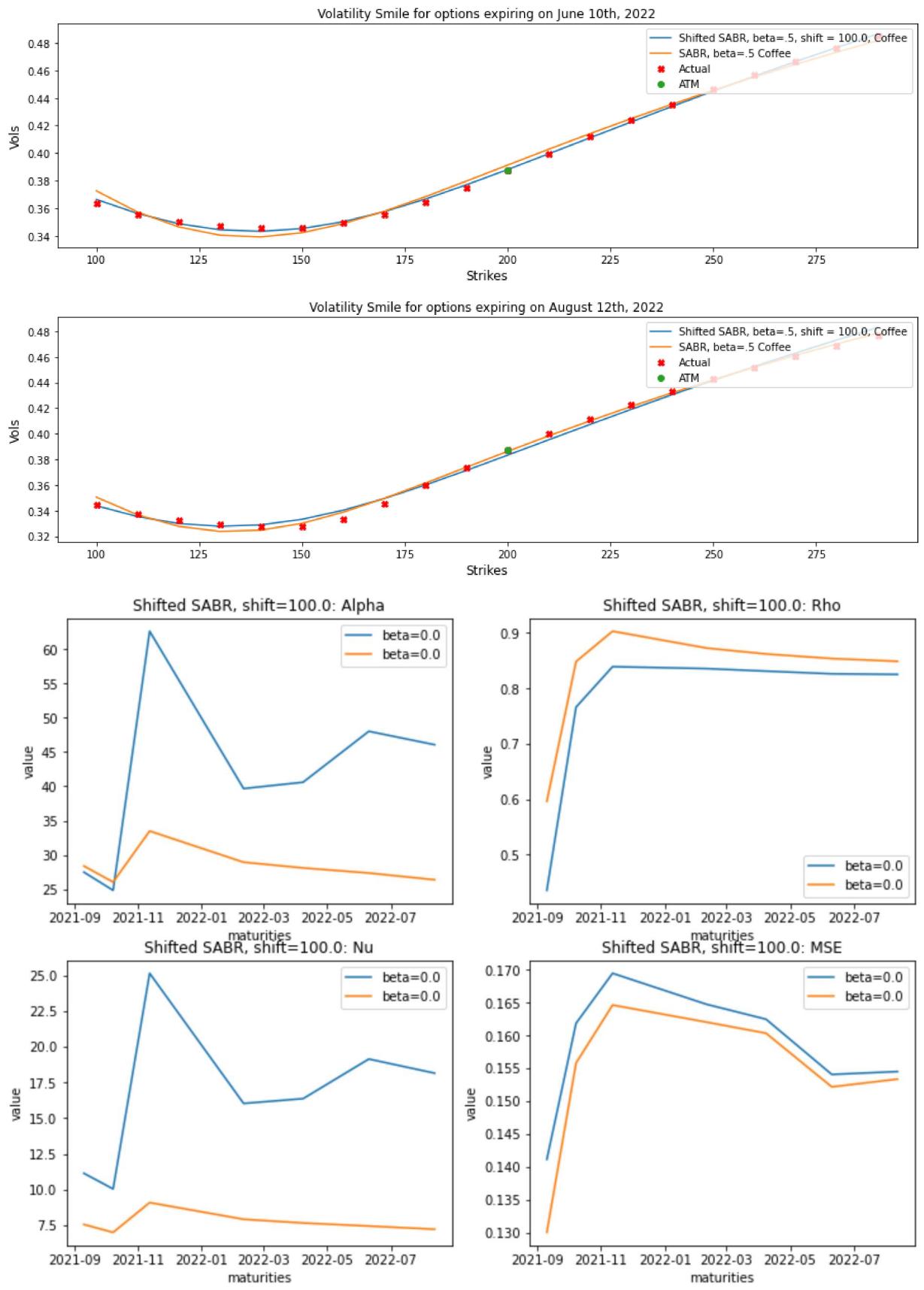


Volatility Smile for options expiring on November 12th, 2021

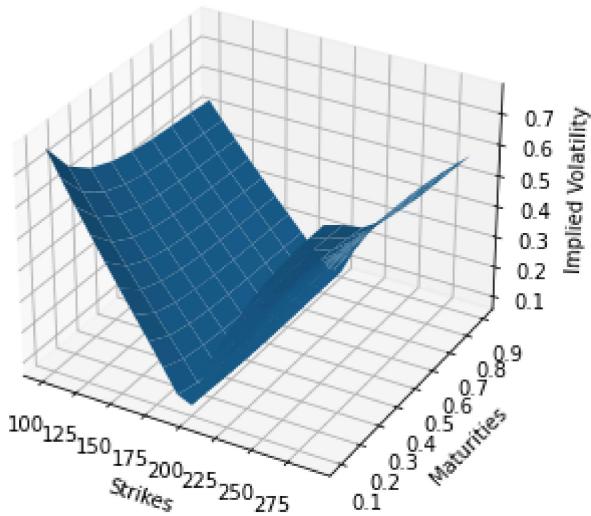


Volatility Smile for options expiring on April 8th, 2022

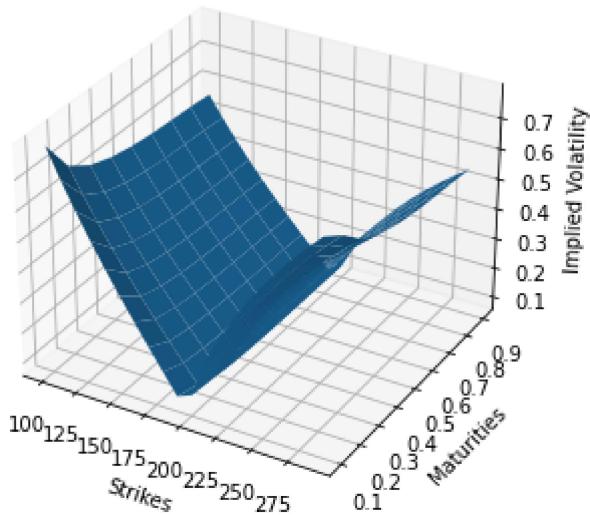




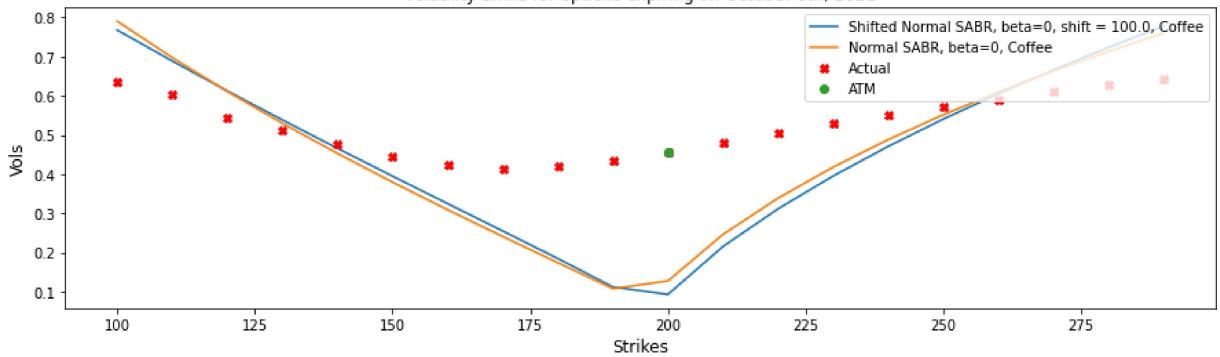
Shifted Normal SABR, beta=0, shift = 100.0, Coffee



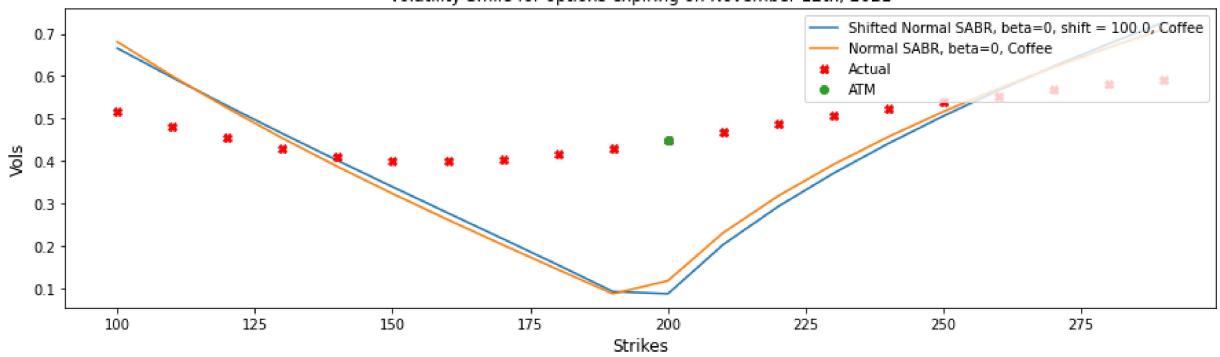
Normal SABR, beta=0, Coffee

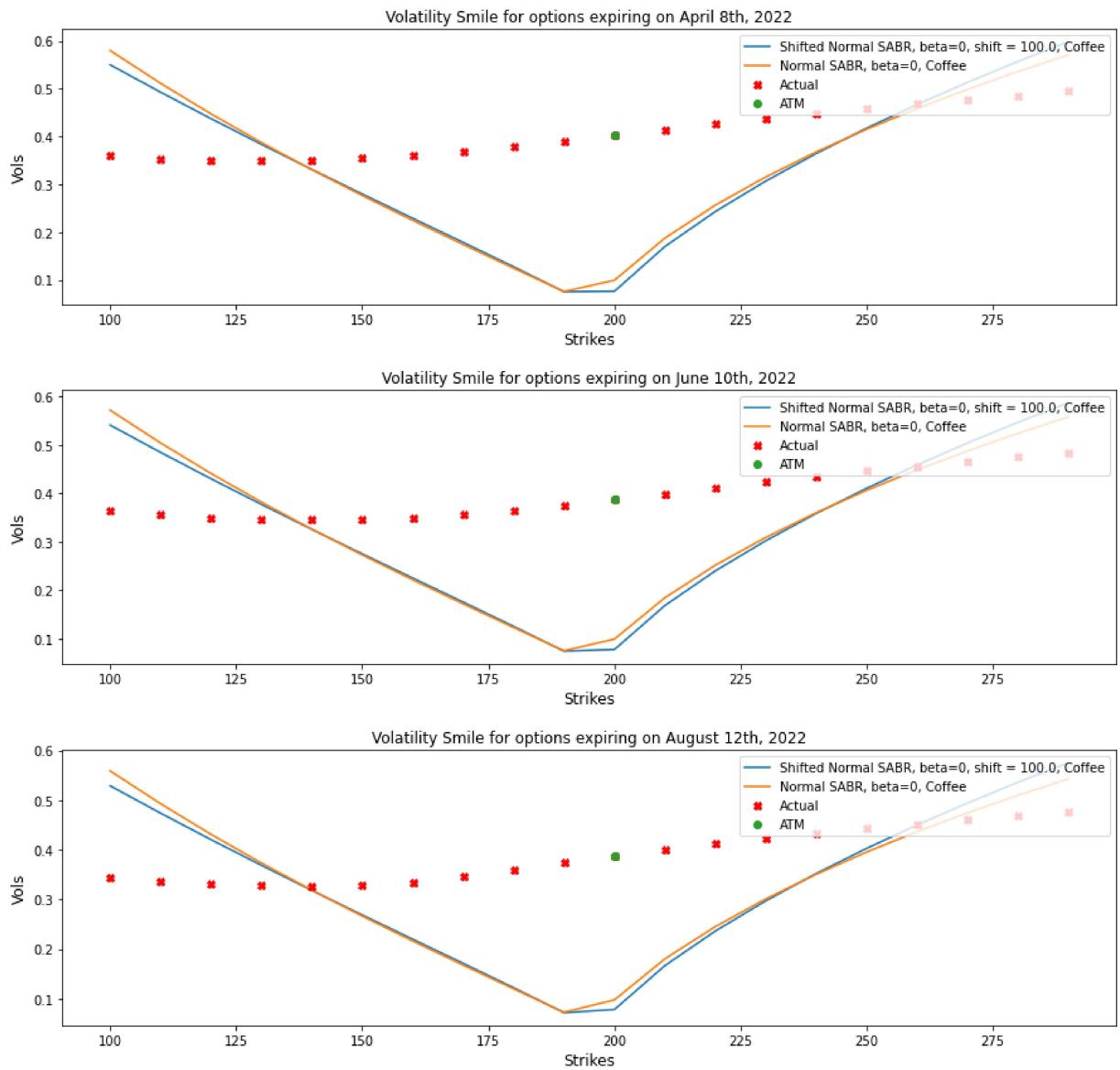


Volatility Smile for options expiring on October 8th, 2021



Volatility Smile for options expiring on November 12th, 2021

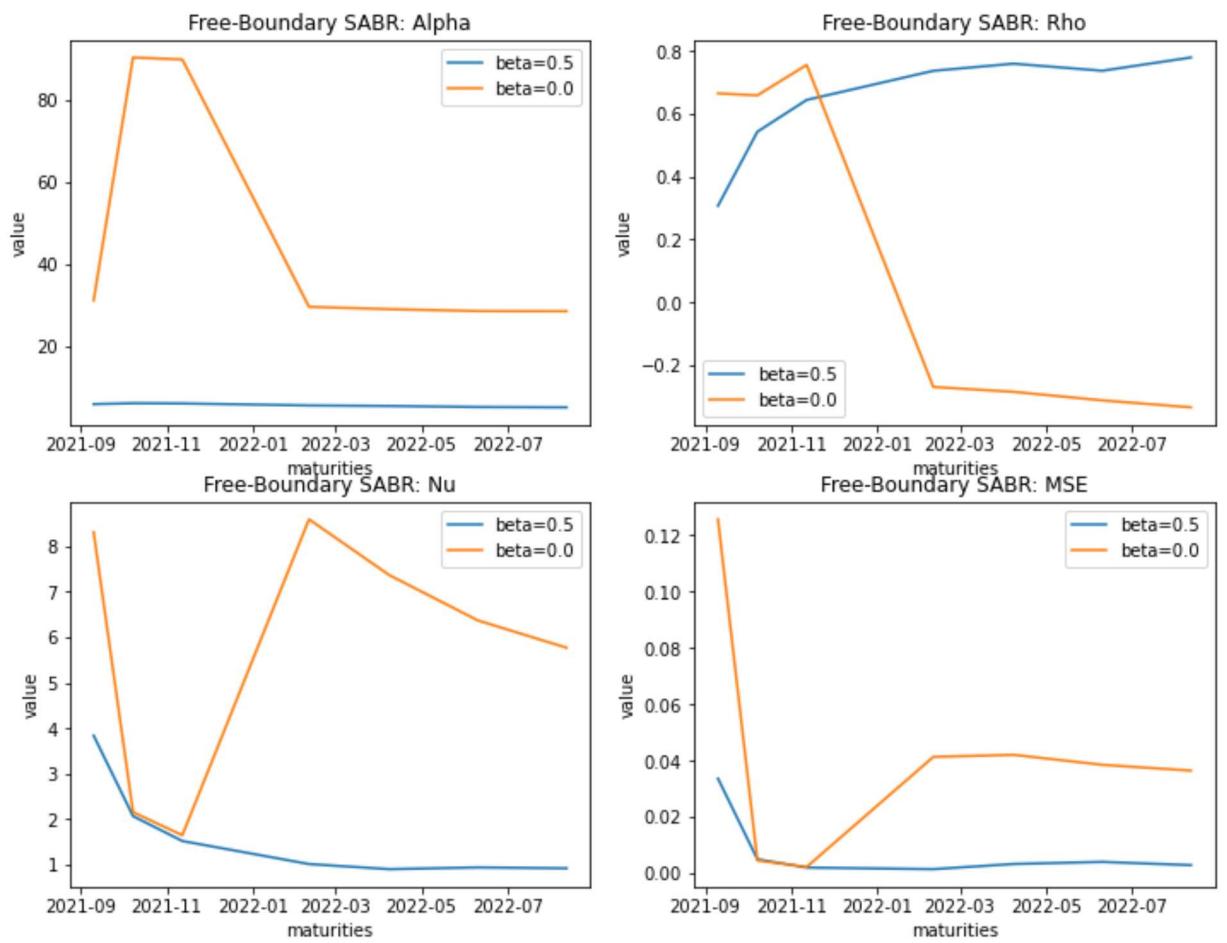




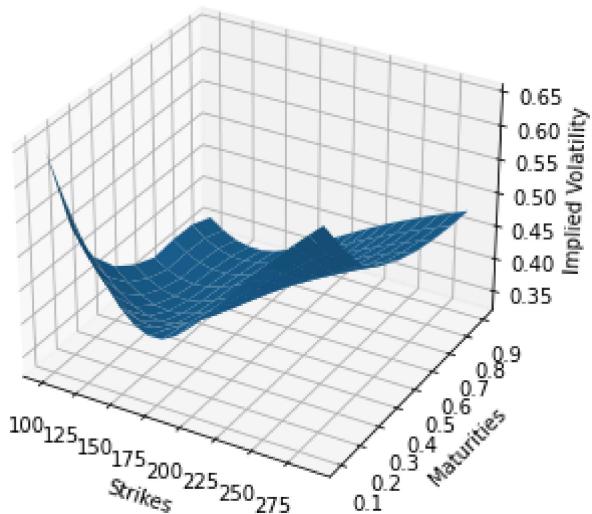
In [11]:

```
# Free-Boundary SABR Volatility model
freeSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=0, method="floch-kennedy", str
freeSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=0, method="floch-kennedy", str
SABRComparison([freeSABR_beta5, freeSABR_beta0], title="Free-Boundary SABR")
```

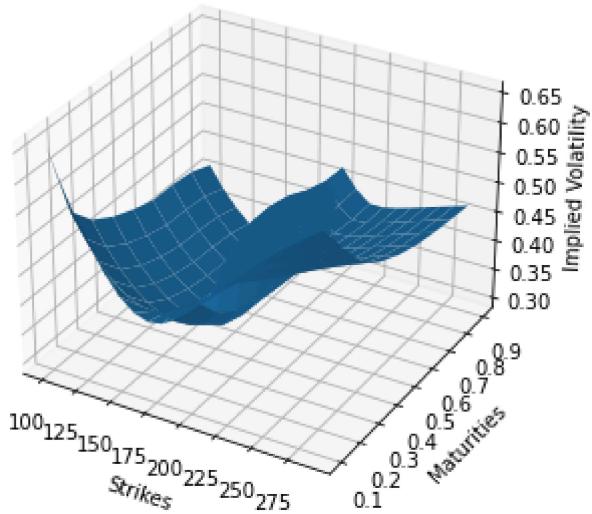
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWarning: Values in x were outside bounds during a minimize step, clipping to bounds
 warnings.warn("Values in x were outside bounds during a "



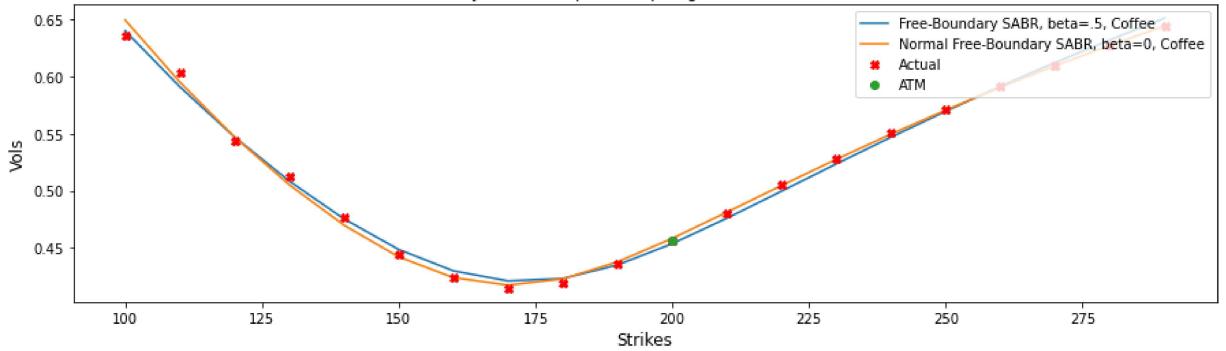
Free-Boundary SABR, beta=.5, Coffee



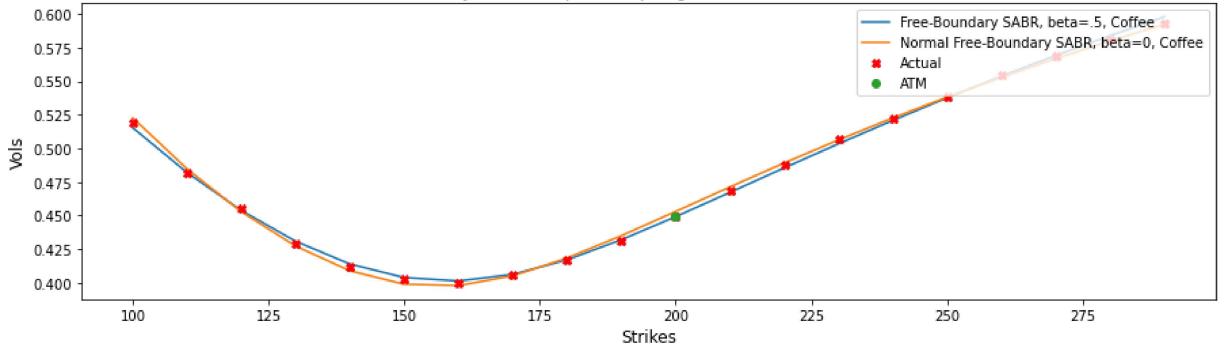
Normal Free-Boundary SABR, beta=0, Coffee



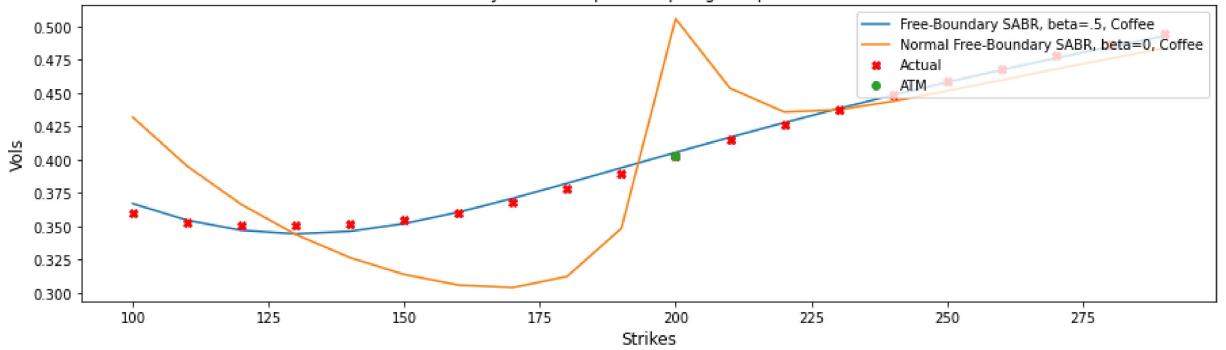
Volatility Smile for options expiring on October 8th, 2021

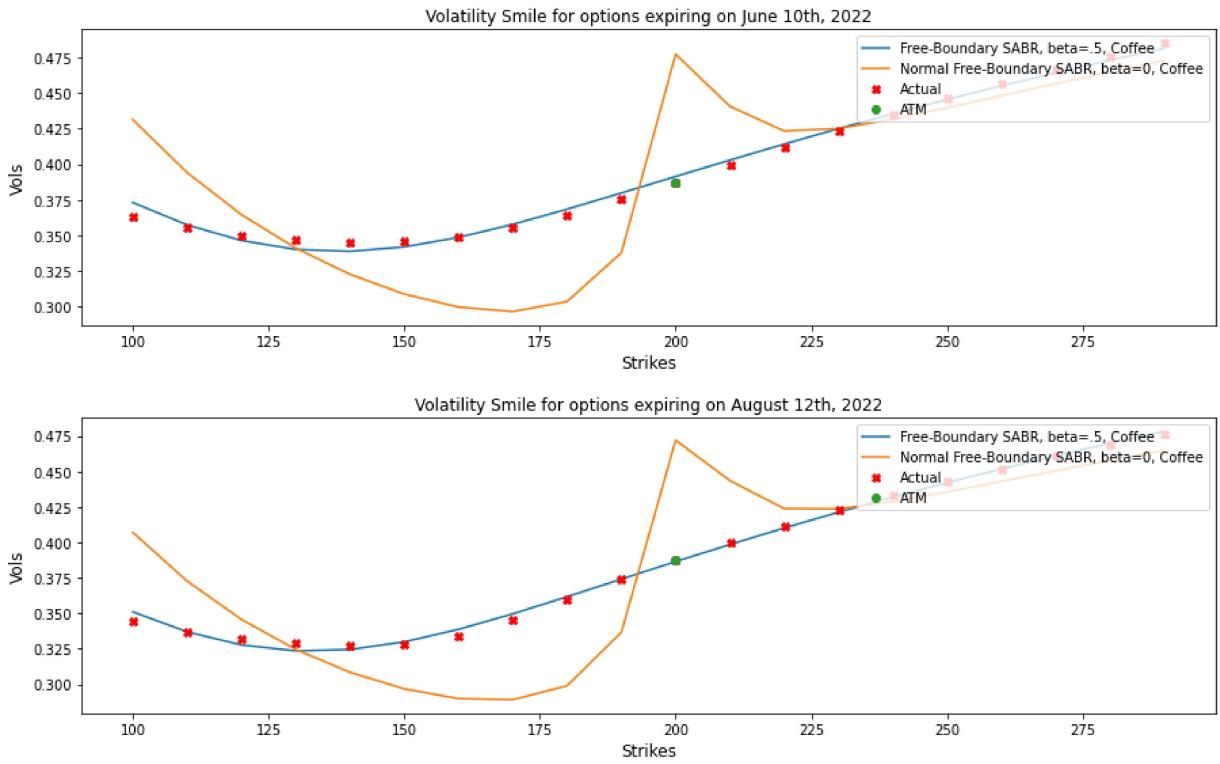


Volatility Smile for options expiring on November 12th, 2021



Volatility Smile for options expiring on April 8th, 2022





In [12]:

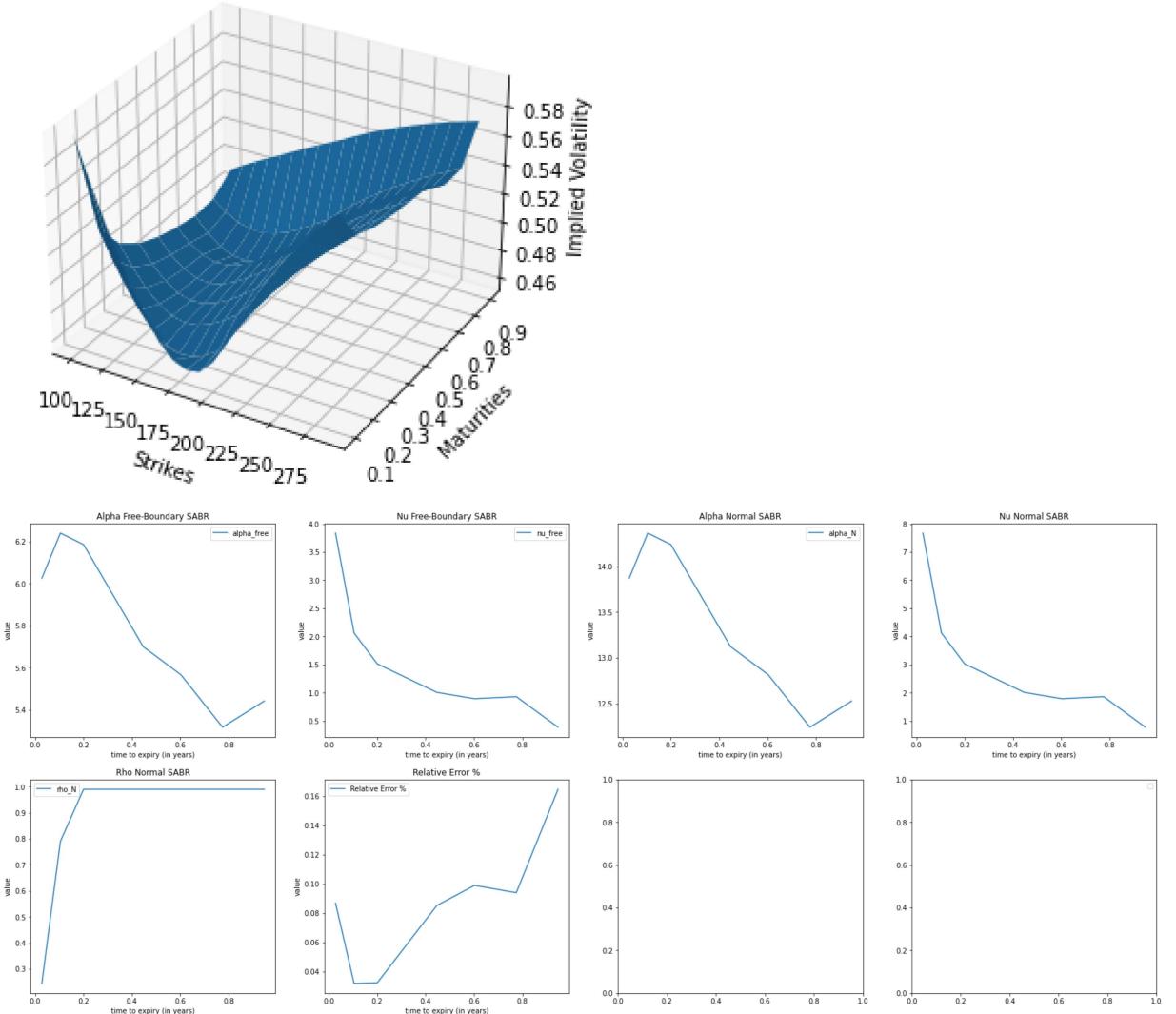
```
# Mixed SABR
```

```
mixtureSABR = MixtureSABRVolatilitySurface(dates=dates)
display(mixtureSABR.to_data())
plot_vol_surface([mixtureSABR.vol_surface], title=mixtureSABR.label)
mixtureSABR.plot()
```

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSI
September 10th, 2021	6.025622		0.5	3.837102	0.0	6.025622	0.0	7.674203	0.244219
October 8th, 2021	6.240184		0.5	2.063050	0.0	6.240184	0.0	4.126099	0.790108
November 12th, 2021	6.185562		0.5	1.515953	0.0	6.185562	0.0	3.031906	0.990000
February 10th, 2022	5.700999		0.5	1.008944	0.0	5.700999	0.0	2.017888	0.990000
April 8th, 2022	5.567359		0.5	0.896325	0.0	5.567359	0.0	1.792650	0.990000
June 10th, 2022	5.318200		0.5	0.933453	0.0	5.318200	0.0	1.866906	0.990000
August 12th, 2022	5.442629		0.5	0.390795	0.0	5.442629	0.0	0.781591	0.990000

No handles with labels found to put in legend.

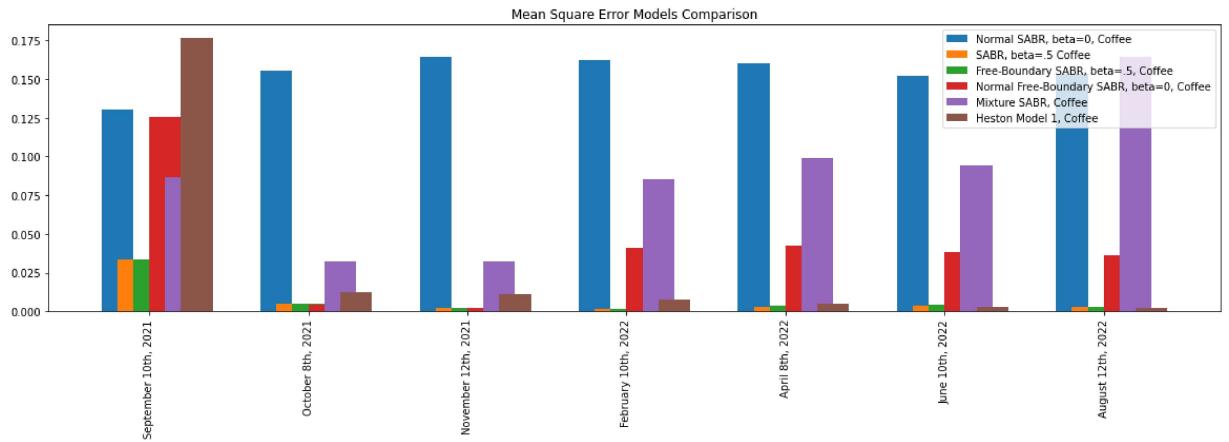
Mixture SABR, Coffee



```
In [13]: # VOLATILITY SMILES ERRORS COMPARISON
```

```
fig = plt.figure(figsize=(20, 5), )
width = .20
plt.bar(np.arange(len(maturities)) - 2*width/2, SABR_beta0.errors, label=SABR_beta0.
plt.bar(np.arange(len(maturities)) - width/2, SABR_beta5.errors, label=SABR_beta5.la
plt.bar(np.arange(len(maturities)), freeSABR_beta5.errors, label=freeSABR_beta5.lab
plt.bar(np.arange(len(maturities)) + width/2, freeSABR_beta0.errors, label=freeSABR_
plt.bar(np.arange(len(maturities)) + 2*width/2, mixtureSABR.errors, label=mixtureSAB
plt.bar(np.arange(len(maturities)) + 3*width/2, hestonModel1.errors, label=hestonMod
plt.xticks(np.arange(len(maturities)), dates, rotation='vertical')
plt.title("Mean Square Error Models Comparison")
plt.legend()
```

```
Out[13]: <matplotlib.legend.Legend at 0x1f301c3e670>
```



In [14]:

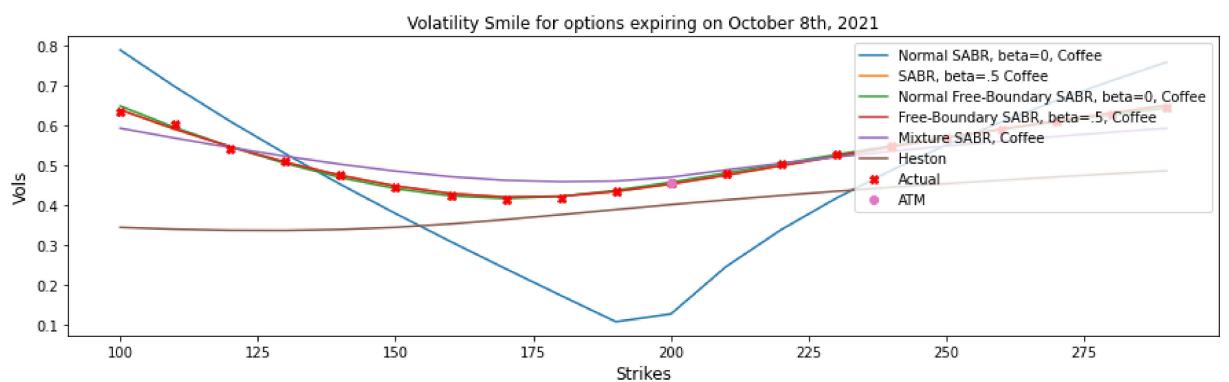
```
# Volatility Smiles Comparisons (Final)
```

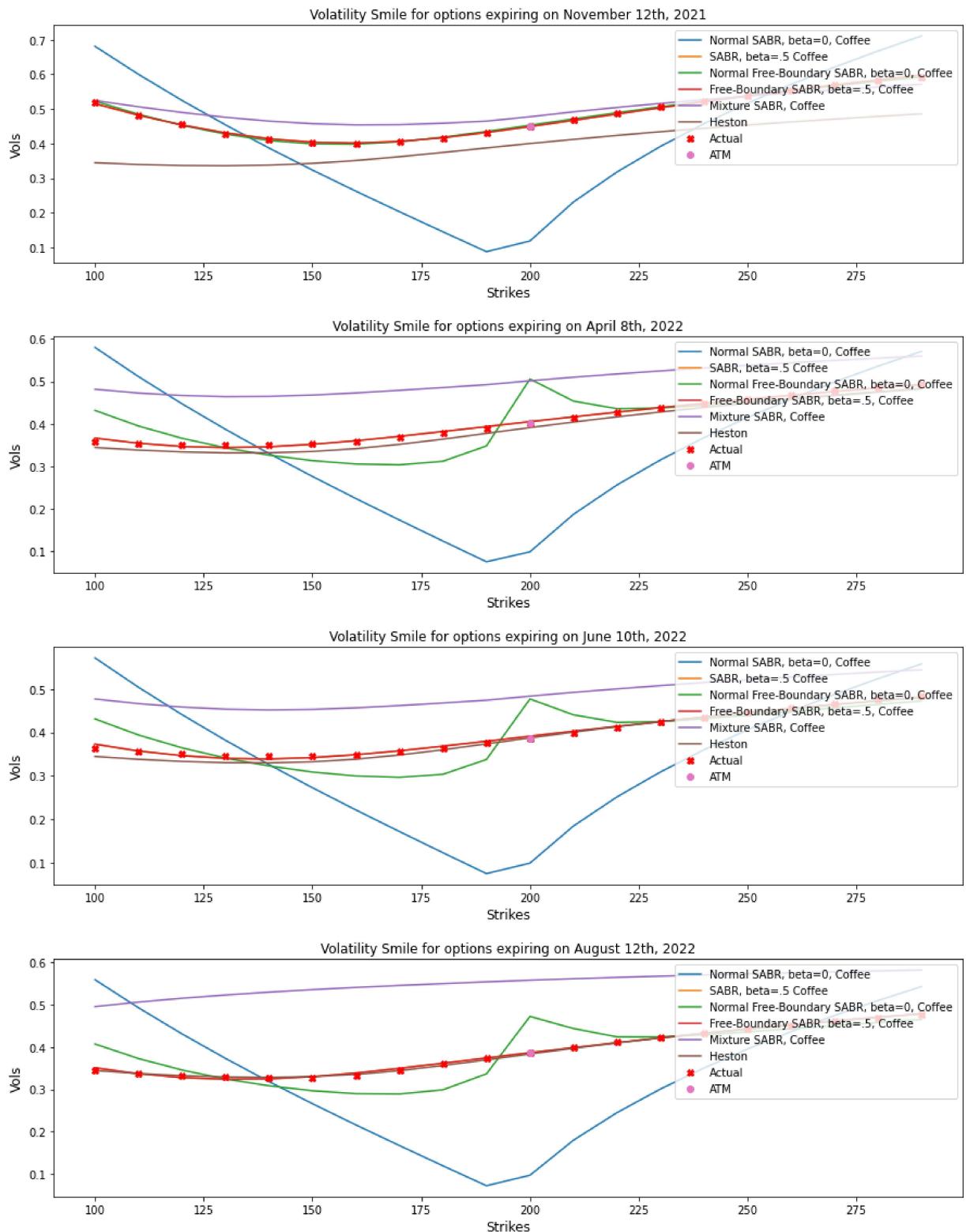
```
models = (
    SABR_beta0,
    SABR_beta5,
    freeSABR_beta0,
    freeSABR_beta5,
    mixtureSABR
)

errors_data = pd.DataFrame([np.mean(m.errors) for m in models], index=[m.label for m in models])
display(errors_data)

smiles_comparison(models, heston_models=[hestonModel1])
```

Error	
Normal SABR, beta=0, Coffee	0.154055
SABR, beta=.5 Coffee	0.007454
Normal Free-Boundary SABR, beta=0, Coffee	0.041553
Free-Boundary SABR, beta=.5, Coffee	0.007456
Mixture SABR, Coffee	0.084790





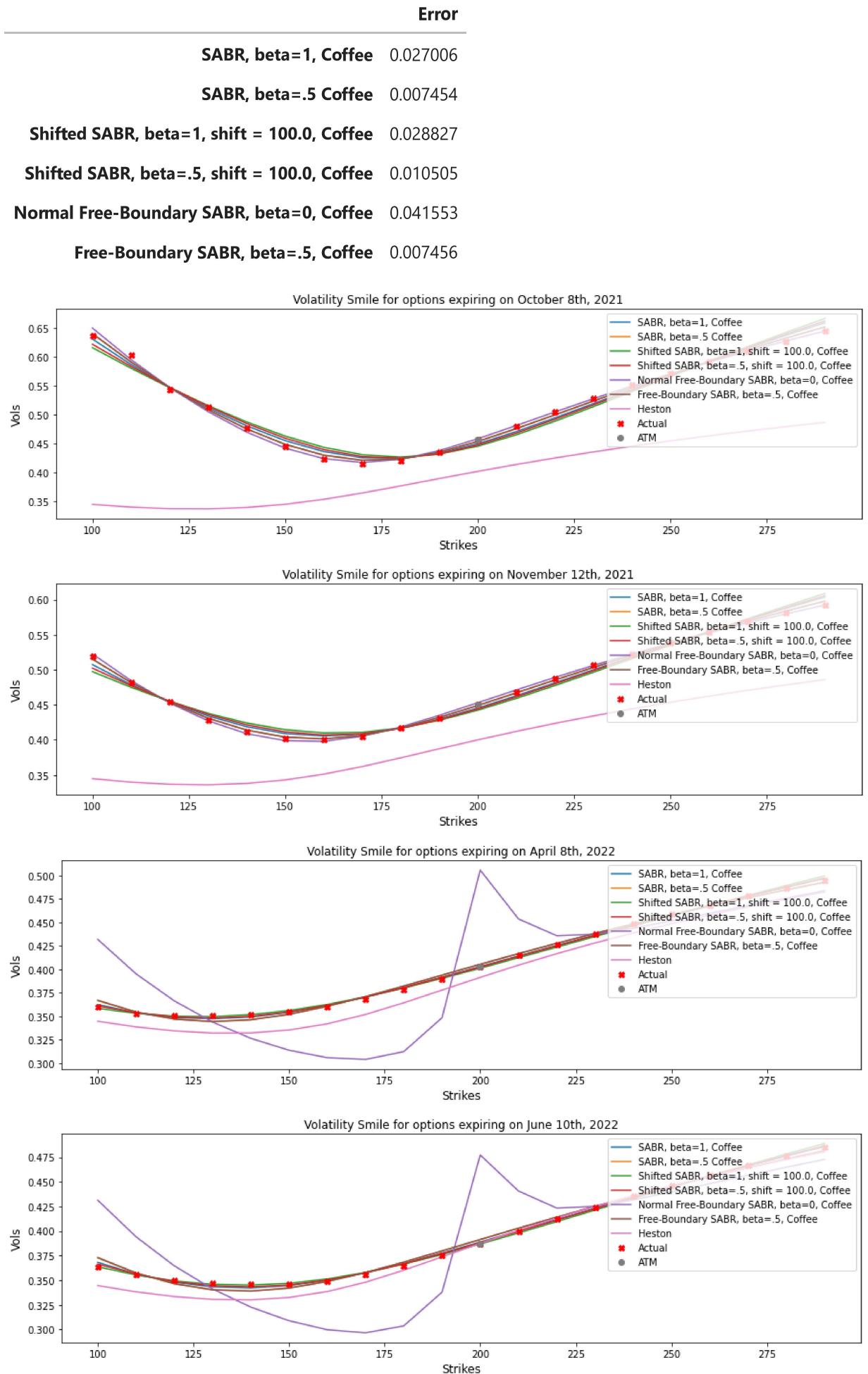
In [15]:

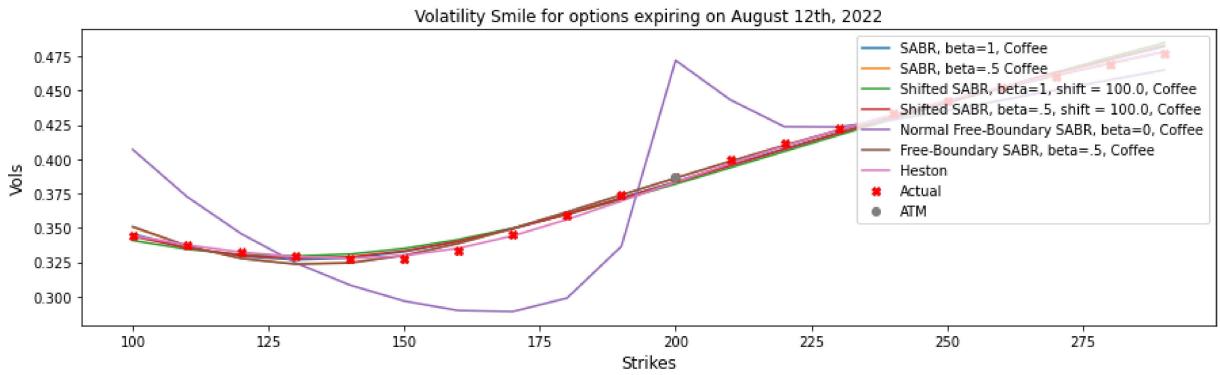
```
# Volatility Smiles Comparisons 2 (Final)
```

```
models = (
    SABR_beta1,
    SABR_beta5,
    shiftedSABR_beta1,
    shiftedSABR_beta5,
    freeSABR_beta0,
    freeSABR_beta5,
)
```

```
errors_data = pd.DataFrame([np.mean(m.errors) for m in models], index=[m.label for m
display(errors_data)
```

```
smiles_comparison(models, heston_models=[hestonModel1])
```





In [16]: *# Volatility Surfaces plots comparison*

```

title = "normal SABR Volatility Surface on {} VS IV surface\n{} to {}\nBeta = 1".format(data, today, plot_vol_surface([SABR_beta0.vol_surface, black_var_surface], title=title))

title = "normal SABR Volatility Surface on {} VS IV surface\n{} to {}\nBeta = 0.5".format(data, today, plot_vol_surface([SABR_beta5.vol_surface, black_var_surface], title=title))

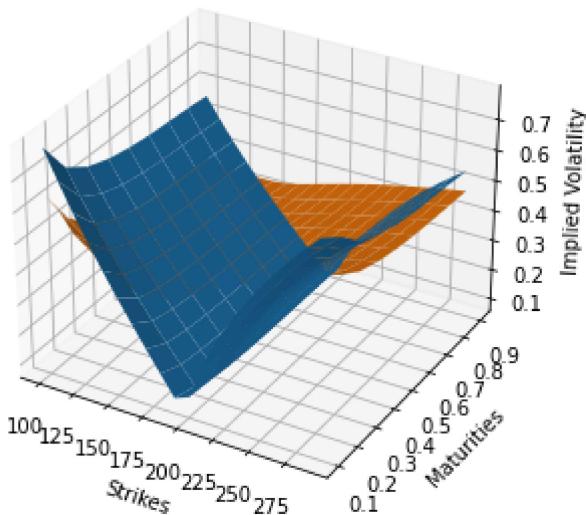
title = "normal SABR Volatility Surface on {} VS IV surface\n{} to {}\nBeta = 0".format(data, today, plot_vol_surface([SABR_beta0.vol_surface, black_var_surface], title=title))

title = "SABR Volatility Surface on {} VS Heston surface\n{} to {}\nBeta = 1".format(data, today, plot_vol_surface([SABR_beta1.vol_surface, hestonModel1.hestон_vol_surface], title=title))

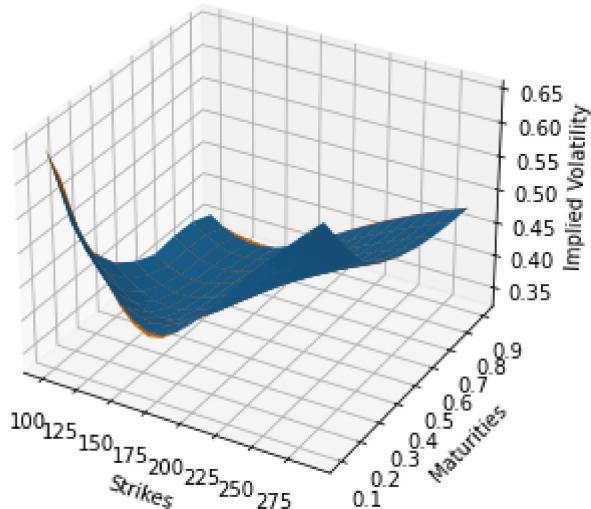
title = "IV Surface on {} vs Heston Surface\n{} to {}\nBeta = 1".format(data, today, plot_vol_surface([black_var_surface, hestonModel1.hestон_vol_surface], title=title))

```

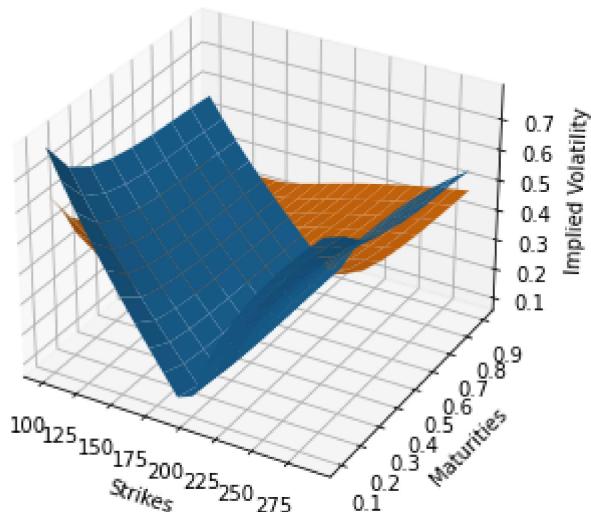
normal SABR Volatility Surface on COFFEE VS IV surface
August 31st, 2021 to August 12th, 2022
Beta = 1



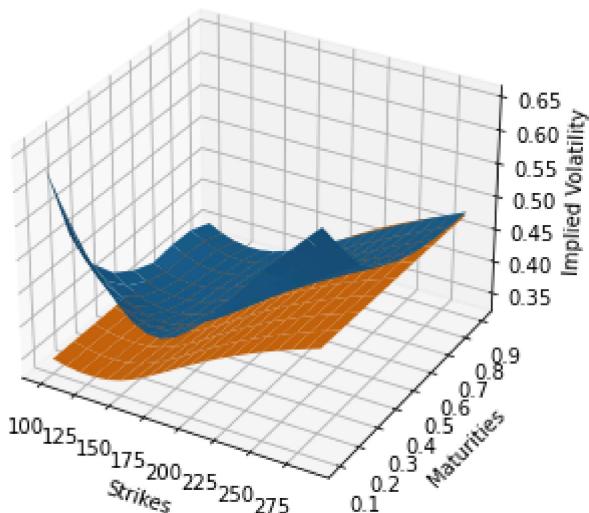
normal SABR Volatility Surface on COFFEE VS IV surface
August 31st, 2021 to August 12th, 2022
Beta = 0.5



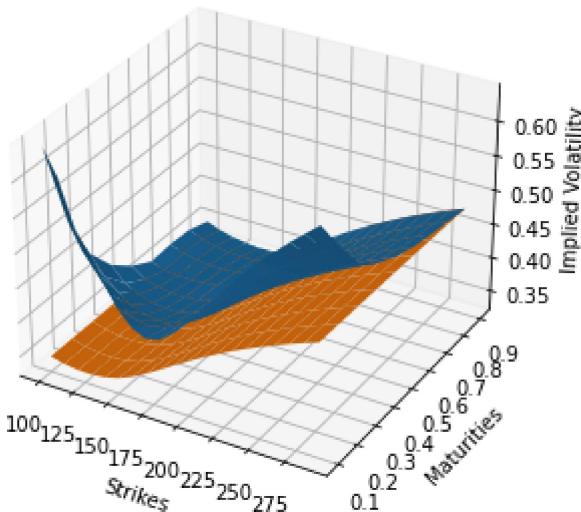
normal SABR Volatility Surface on COFFEE VS IV surface
August 31st, 2021 to August 12th, 2022
Beta = 0



SABR Volatility Surface on COFFEE VS Heston surface
August 31st, 2021 to August 12th, 2022
Beta = 1

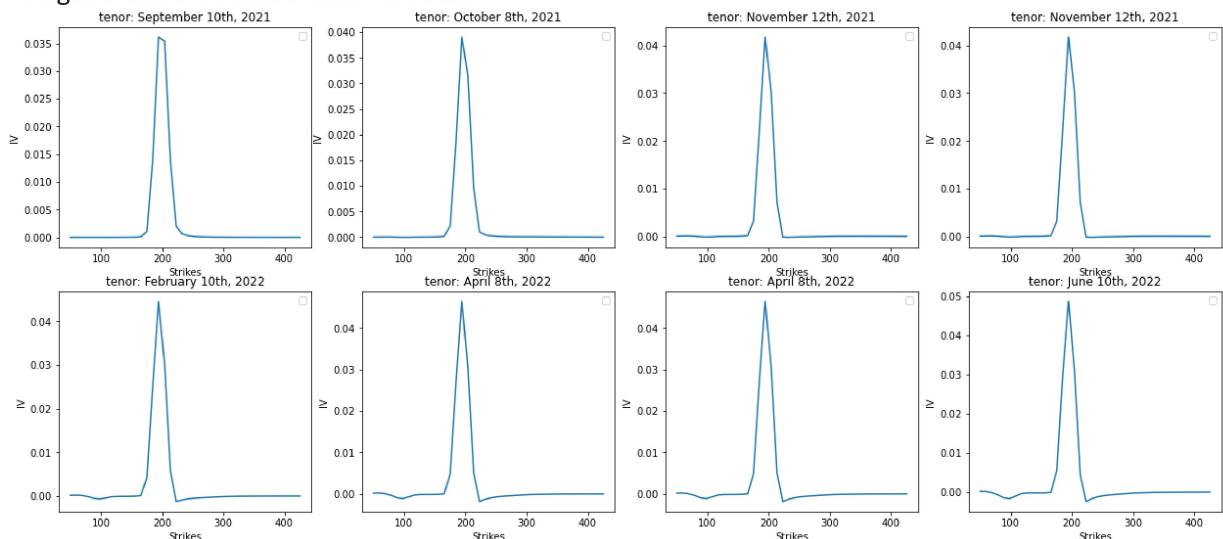


IV Surface on COFFEE vs Heston Surface
August 31st, 2021 to August 12th, 2022
Beta = 1



```
In [17]: # Compare Implied Volatility distributions accross time
plot_densityIV(SABR_beta0)
```

No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
No handles with labels found to put in legend.
<Figure size 864x360 with 0 Axes>



```
In [18]: # Compute time for each model
time_data = compute_time()

time_df = pd.DataFrame(time_data.values(), index=time_data.keys(), columns=["Computation Time"])
display(time_df)

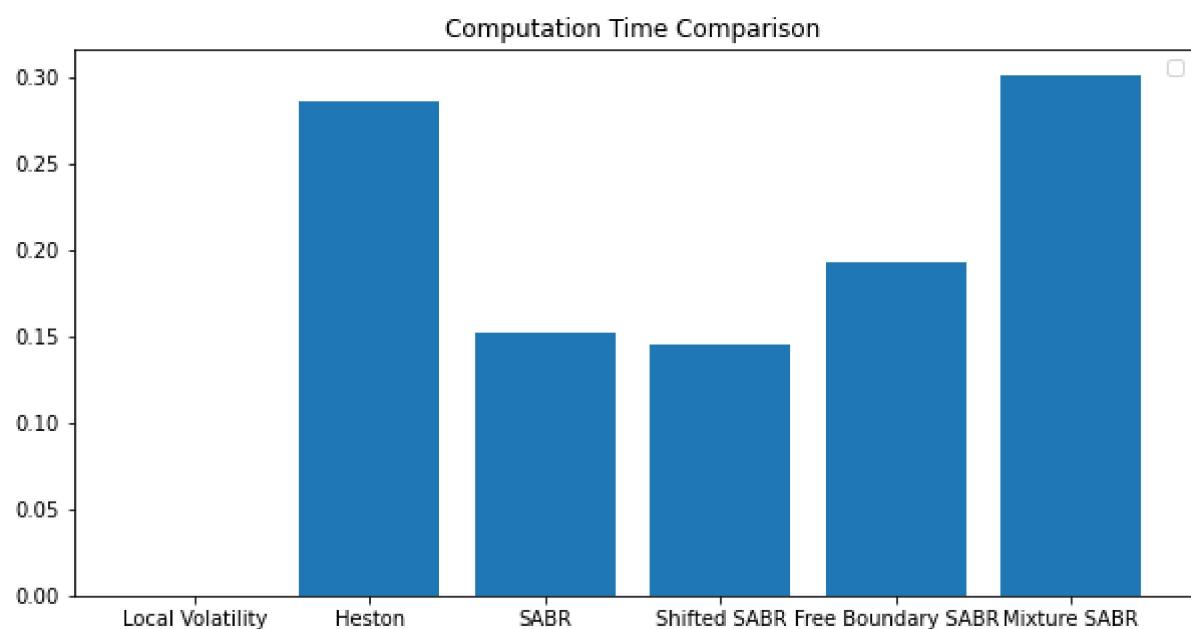
fig = plt.figure(figsize=(10, 5))
plt.bar(x=time_data.keys(), height=time_data.values(), )
plt.title("Computation Time Comparison")
plt.legend()
```

```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWarning: Values in x were outside bounds during a minimize step, clipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```

Computation time	
Local Volatility	0.000000
Heston	0.285760
SABR	0.152148
Shifted SABR	0.145122
Free Boundary SABR	0.193186
Mixture SABR	0.301274

No handles with labels found to put in legend.

```
Out[18]: <matplotlib.legend.Legend at 0x1f301a7f8e0>
```



```
In [ ]:
```