In [1]:

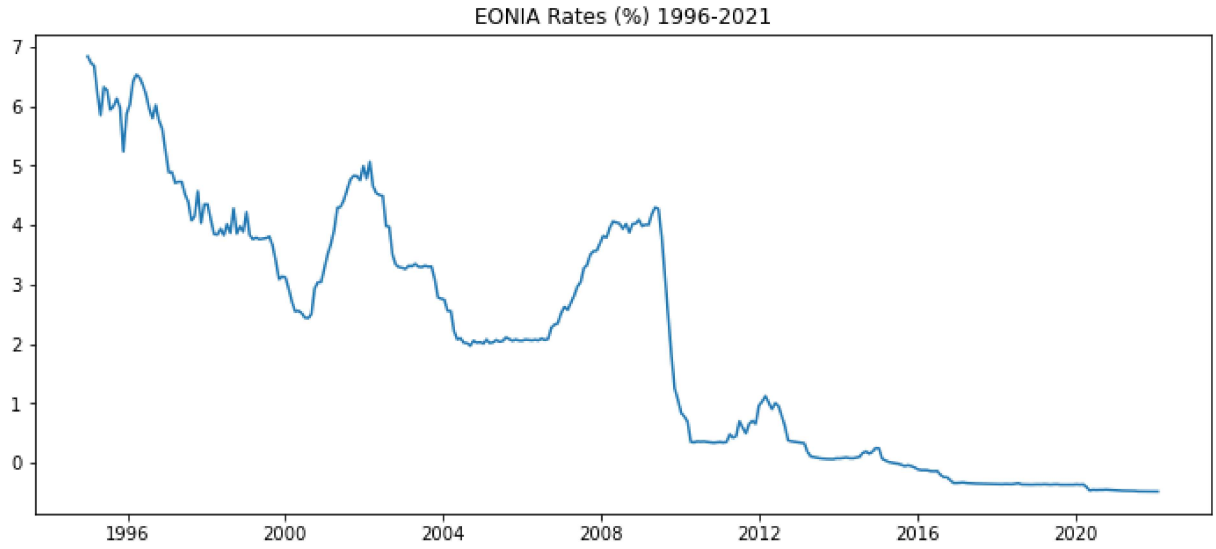
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import time
import datetime
import math
import QuantLib as ql
from scipy.optimize import minimize

from initialize import *
from plotting import *
from SABR import *
from Heston import *
from mixedSABR import *
```

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")
```

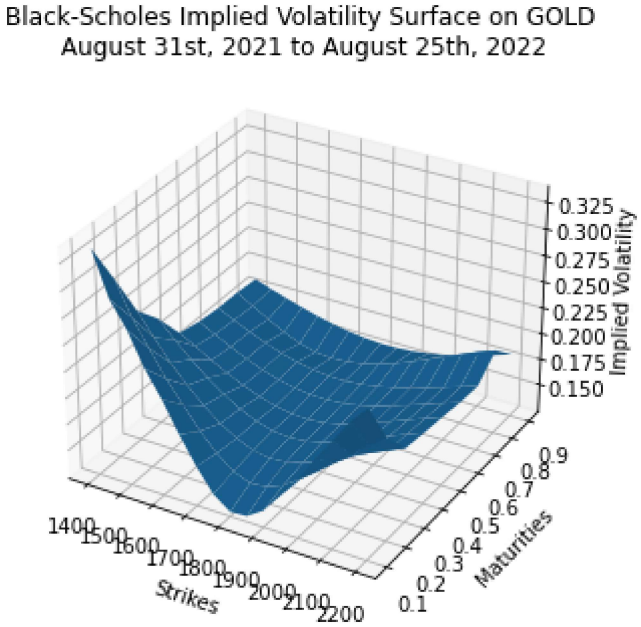
	Tenors	Spot Rate
0	0.25	-0.575
1	1.00	-0.557
2	3.00	-0.549
3	6.00	-0.529
4	12.00	-0.494

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



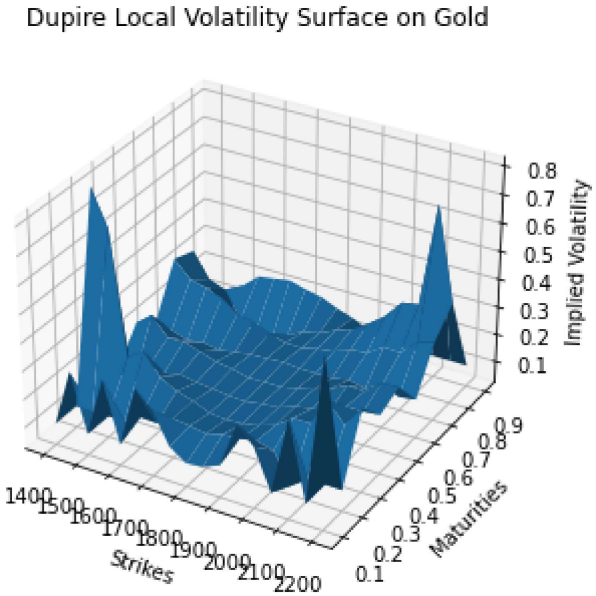
In [3]:

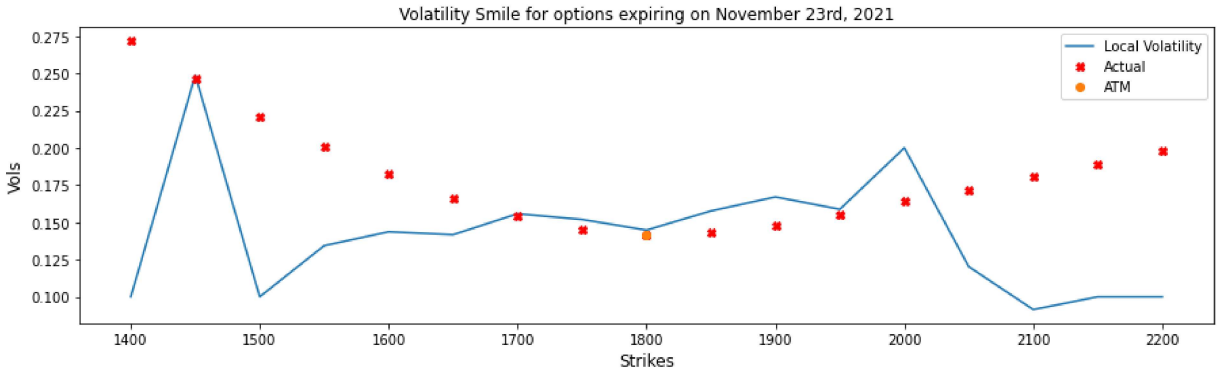
```
# BLACK VOLATILITY SURFACE

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
```

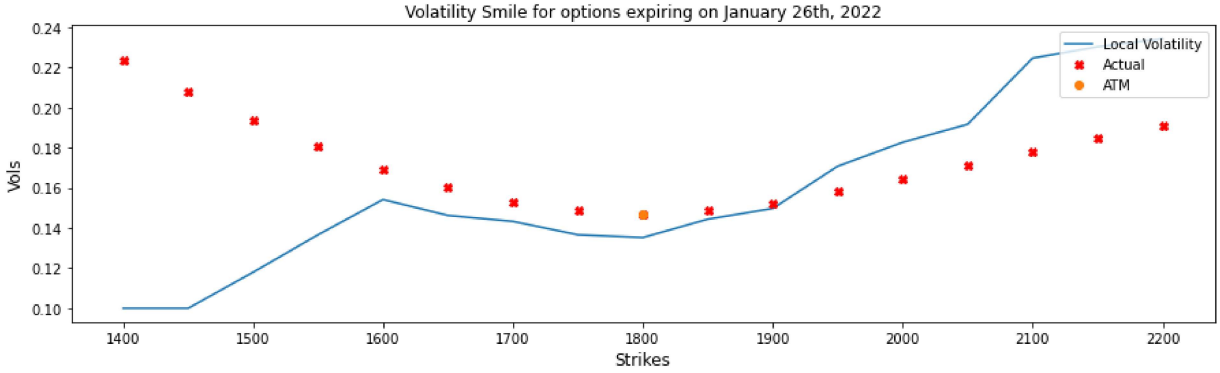


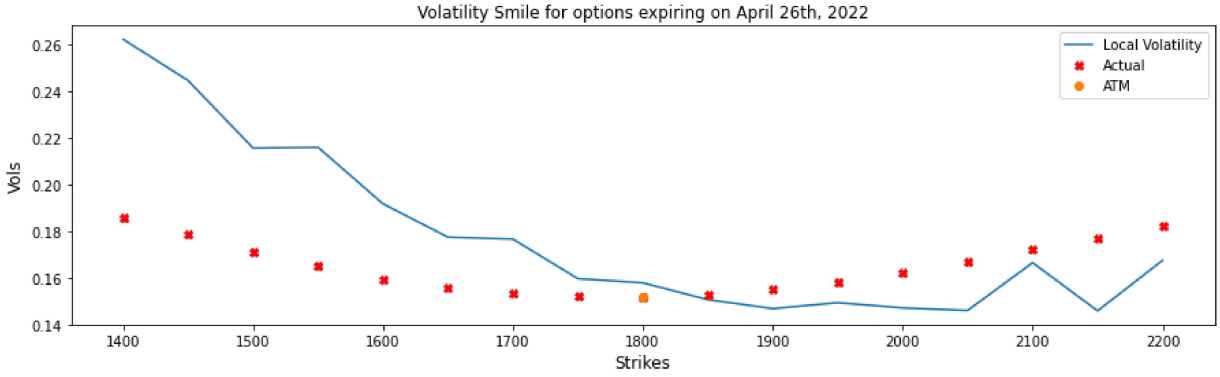
In [4]:

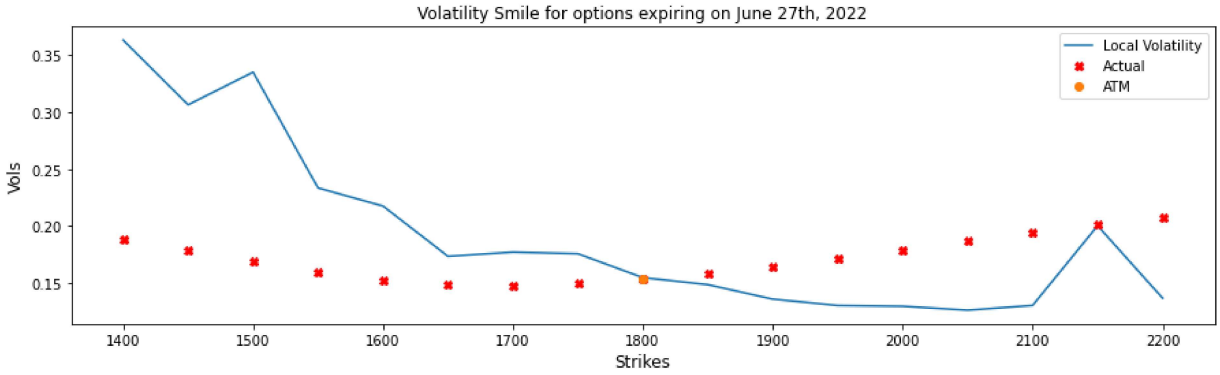
```
#DUPIRE LOCAL VOLATILITY SURFACE (NOT PLOTTABLE)

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, spo
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)
```

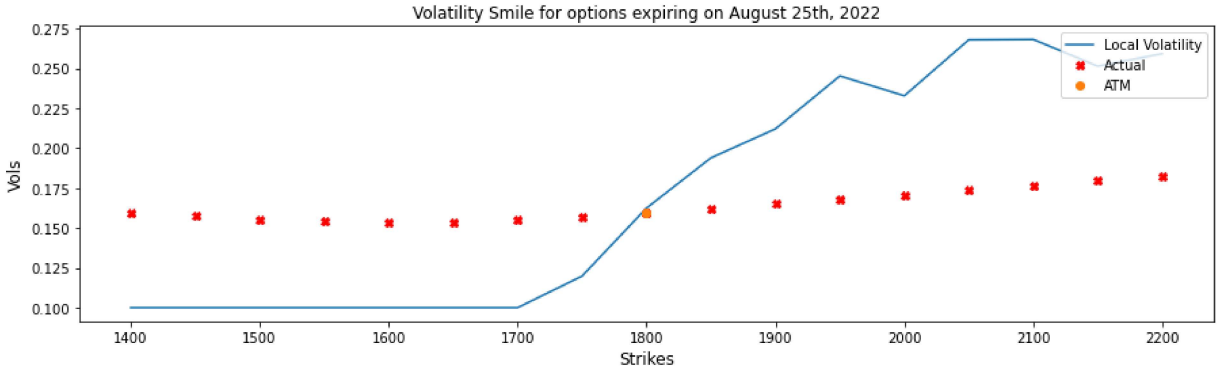












In [5]:

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

```
m1_params, m2_params = (None, None)

if data == "SPX":
    m1_params = (0.05,0.2,0.5,0.1,0.09)
    m2_params = (0.15,0.5,0.2,0.7,0.01)
elif 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)
else:
    m1_params = (0.03,0.3,0.5,0.3,0.04)
    m2_params = (0.01,0.5,0.5,0.1,0.03)

hestonModel1 = hestonModelSurface(m1_params, label="Heston Model 1, {}".format(data_
hestonModel2 = 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,
    hestonModel0 = hestonModelSurface(result["x"], label="Heston Model 0, {}".format


    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 (Gold)

	theta	kappa	sigma	rho	v0
Model1	0.030000	0.300000	0.500000	0.300000	0.040000
Model2	0.010000	0.500000	0.500000	0.100000	0.030000

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, Gold calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	1400.000000	5.148632	5.112091	-0.709715
1	1450.000000	8.254253	8.239515	-0.178553
2	1500.000000	12.849969	12.901971	0.404683
3	1550.000000	19.492403	19.639502	0.754649
4	1600.000000	28.798716	29.067573	0.933574
5	1650.000000	41.573900	41.817053	0.584869
6	1700.000000	58.585336	58.429318	-0.266309
7	1750.000000	79.768612	79.228944	-0.676541
8	1800.000000	105.019940	104.240402	-0.742276
9	1850.000000	103.874121	103.065787	-0.778187
10	1900.000000	85.994838	85.321891	-0.782544
11	1950.000000	71.103345	70.612004	-0.691023
12	2000.000000	58.676208	58.459215	-0.369814
13	2050.000000	48.535785	48.436423	-0.204719
14	2100.000000	40.145960	40.176177	0.075266
15	2150.000000	33.216956	33.368876	0.457357
16	2200.000000	27.591312	27.756651	0.599242

Heston Model 1, Gold parameters output

	Value
v0	0.238487
kappa	0.000000
theta	0.183581
sigma	0.313453
rho	0.027788
avgError	0.541725

```
No handles with labels found to put in legend.
```

Heston Model 2, Gold calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	1400.000000	5.148632	26.339113	411.574996
1	1450.000000	8.254253	34.638642	319.645996

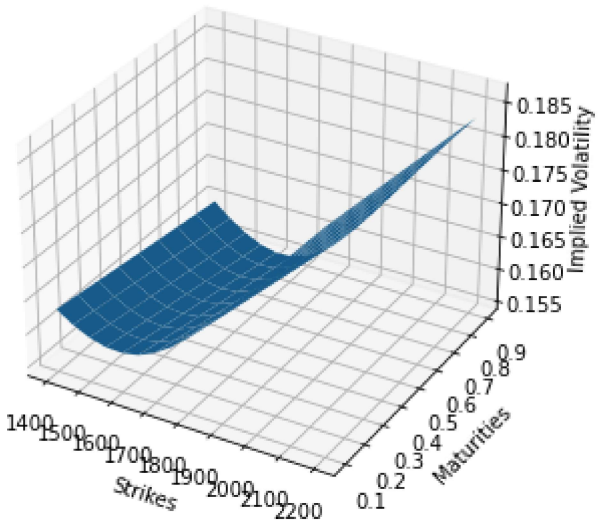
	Strikes	Market Value	Model Value	Relative Error (%)
2	1500.000000	12.849969	45.308477	252.596003
3	1550.000000	19.492403	58.847844	201.901439
4	1600.000000	28.798716	75.694679	162.840465
5	1650.000000	41.573900	96.118017	131.197982
6	1700.000000	58.585336	120.144660	105.076335
7	1750.000000	79.768612	147.569081	84.996426
8	1800.000000	105.019940	178.034164	69.524154
9	1850.000000	103.874121	180.984029	74.233992
10	1900.000000	85.994838	166.064879	93.110286
11	1950.000000	71.103345	153.016007	115.202263
12	2000.000000	58.676208	141.536740	141.216574
13	2050.000000	48.535785	131.378794	170.684393
14	2100.000000	40.145960	122.339063	204.735674
15	2150.000000	33.216956	114.251632	243.955758
16	2200.000000	27.591312	106.980577	287.732838

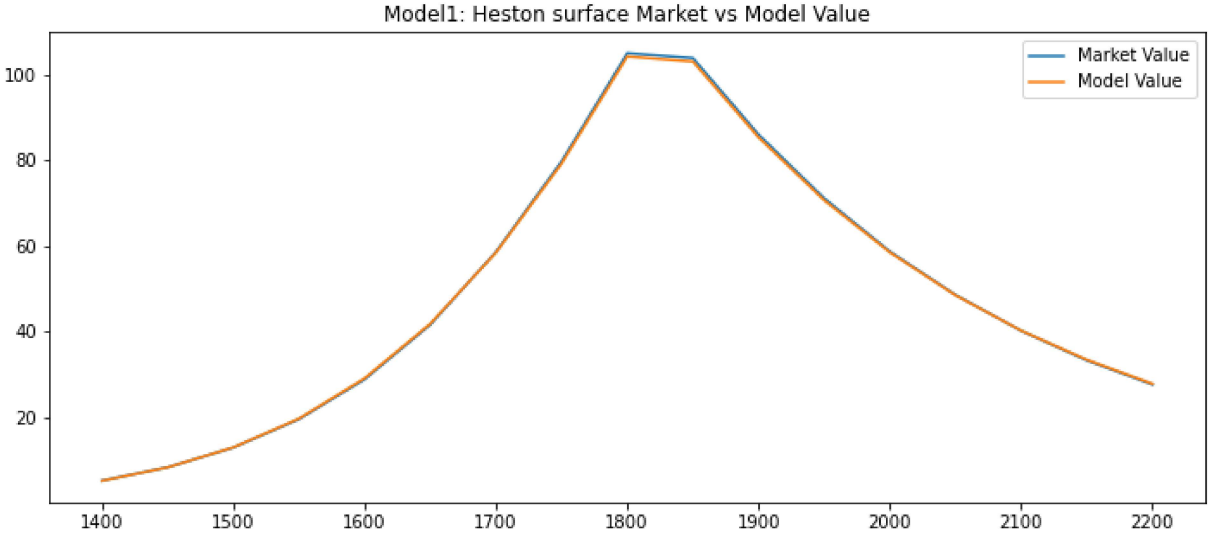
Heston Model 2, Gold parameters output

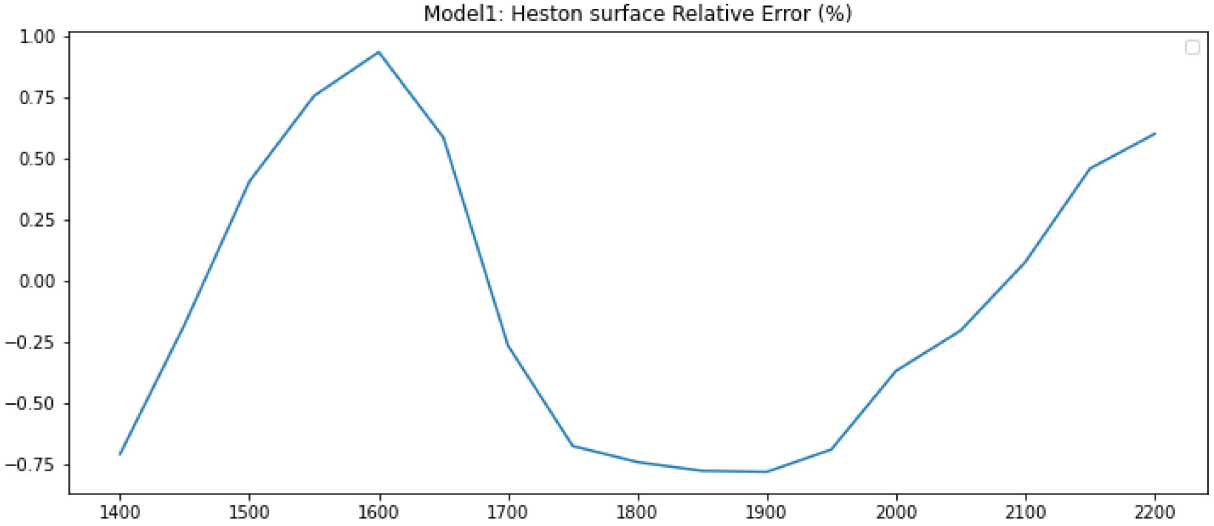
	Value
v0	0.461751
kappa	0.452693
theta	1.028407
sigma	0.551760
rho	0.000000
avgError	180.601504

```
No handles with labels found to put in legend.
```

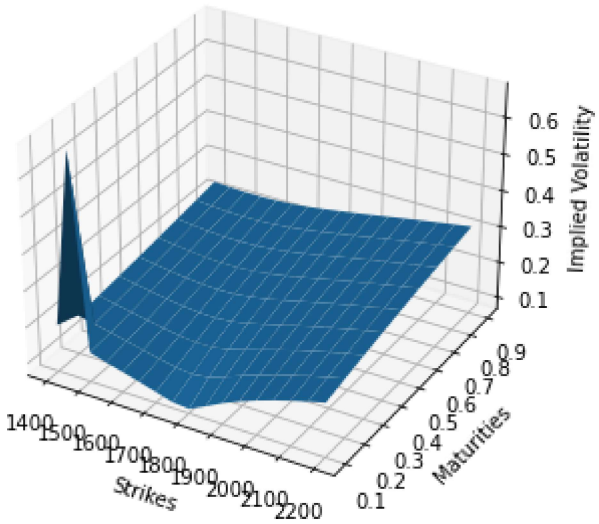
Heston Volatility Surface for Heston Model 1, Gold

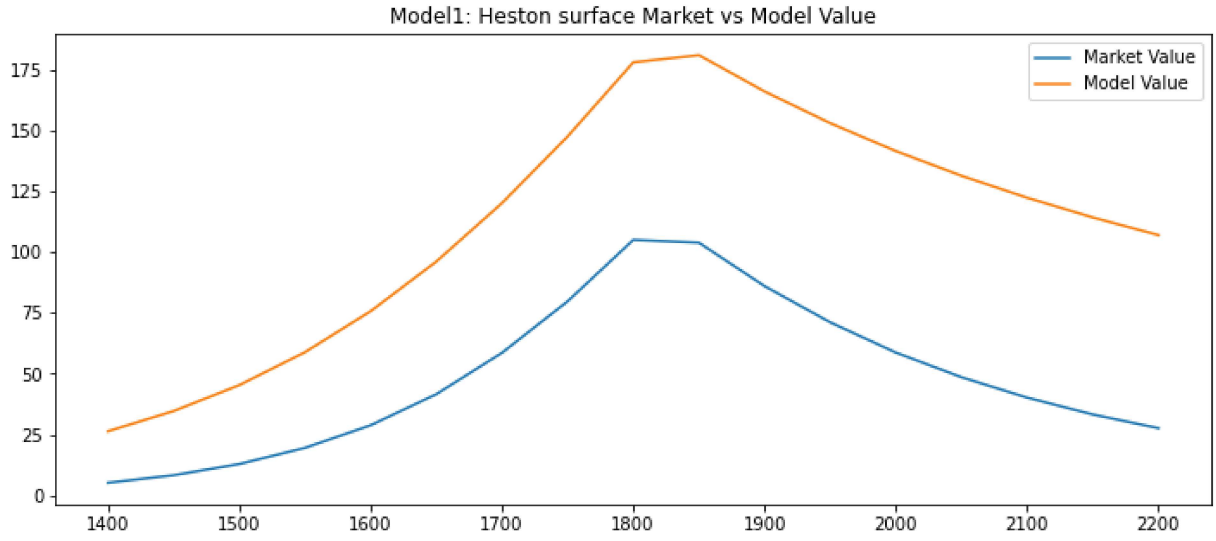


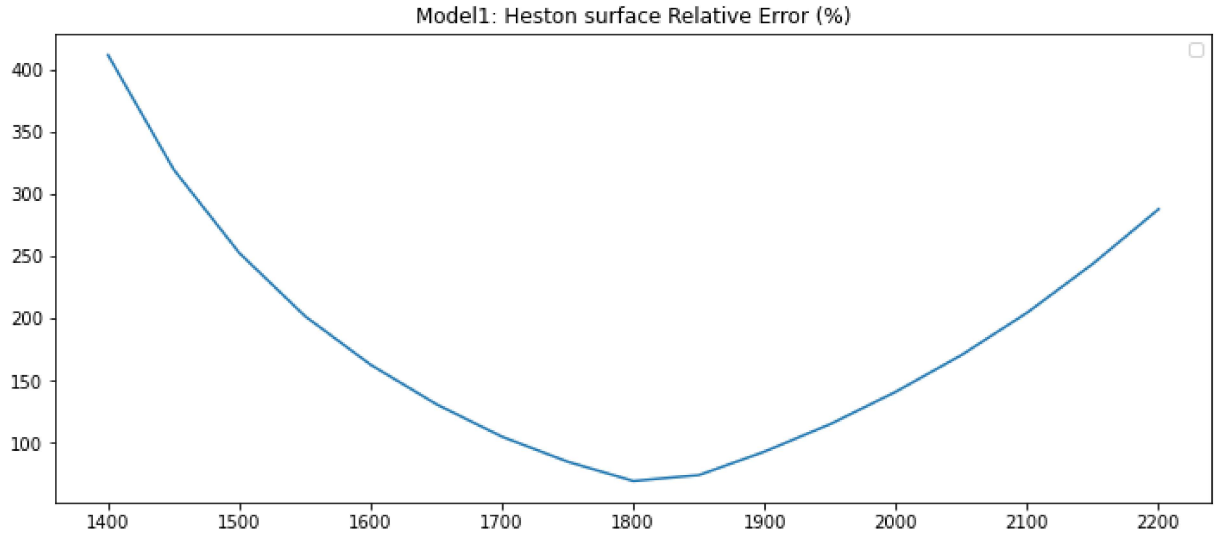




Heston Volatility Surface for Heston Model 2, Gold

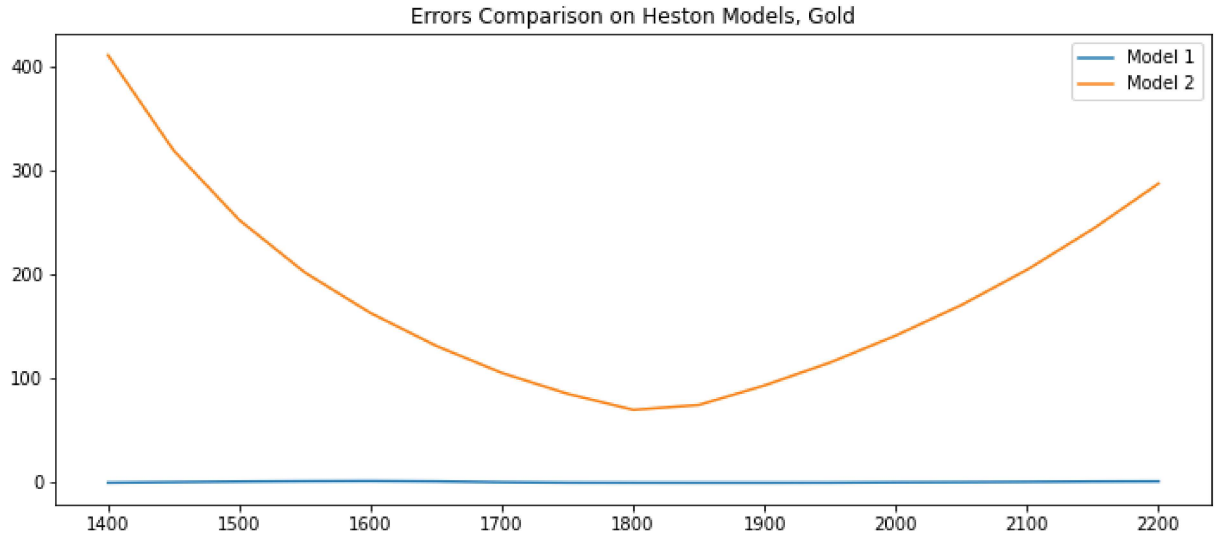


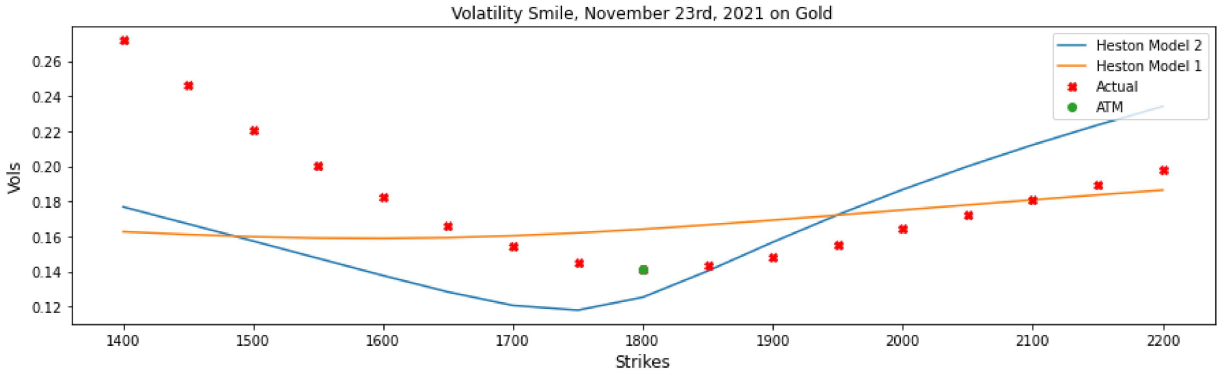


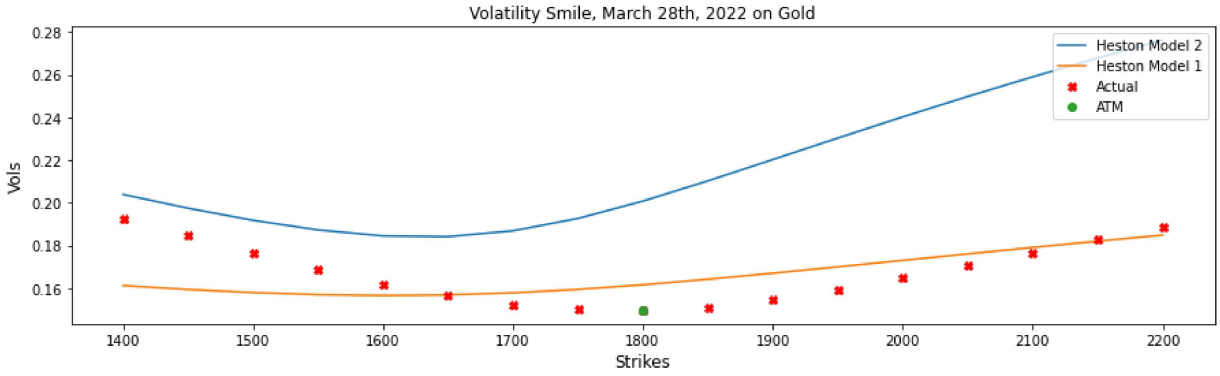


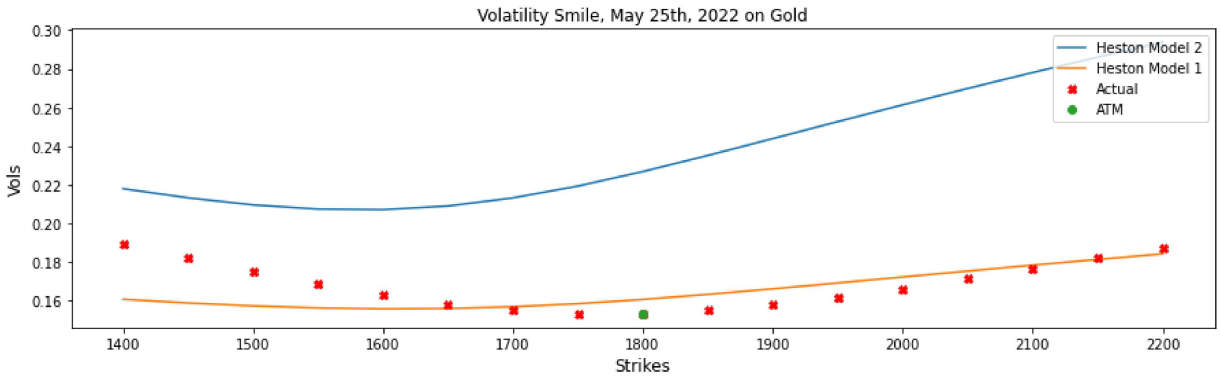
In [7]:

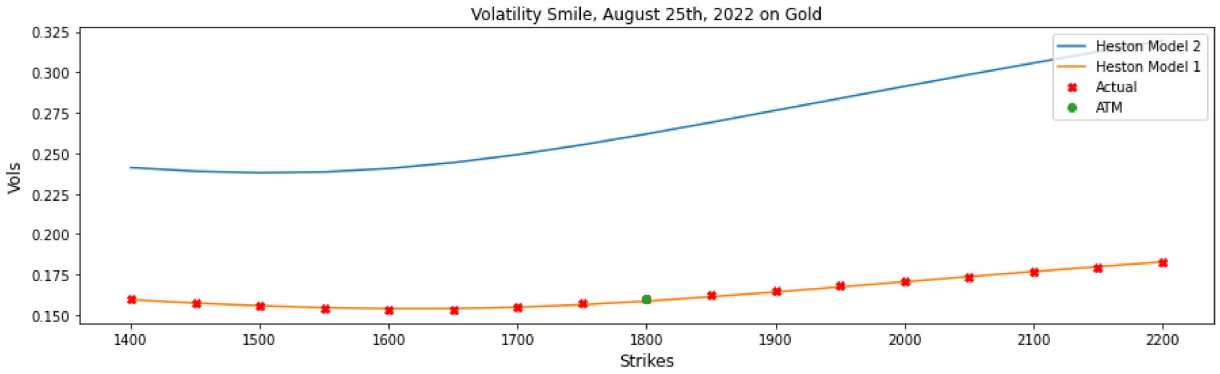
```
# Relative error comparison
plt.figure(figsize=plot_size)
plt.plot(hestonModel1.strks, hestonModel1.relativeError, label="Model 1")
plt.plot(hestonModel2.strks, hestonModel2.relativeError, label="Model 2")
plt.title("Errors Comparison on Heston Models, {}".format(data_label));
plt.legend()

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


Errors Comparison on Heston Models, Gold
Model 1
Model 2
400
300
200
100
0
1400
1500
1600
1700
1800
1900
2000
2100
2200


Volatility Smile, November 23rd, 2021 on Gold
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes


Volatility Smile, March 28th, 2022 on Gold
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes


Volatility Smile, May 25th, 2022 on Gold
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes



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()

def plot_vol_surface(vol_surface, plot_years=np.arange(0.1, (dates[-1] - today) / 36
    if type(vol_surface) != list:
        surfaces = [vol_surface]
    else:
        surfaces = vol_surface

    fig = plt.figure(figsize=plot_size)
    ax = fig.add_subplot(projection='3d')
    ax.set_xlabel('Strikes')
    ax.set_ylabel('Maturities')
    ax.set_zlabel('Implied Volatility')
    ax.set_title(title)
    X, Y = np.meshgrid(plot_strikes, plot_years)

    for surface in surfaces:
        method_to_call = getattr(surface, funct)

        Z = np.array([method_to_call(float(y), float(x))
                      for xr, yr in zip(X, Y)
                      for x, y in zip(xr, yr)]
                     ).reshape(len(X), len(X[0]))

        surf = ax.plot_surface(X, Y, Z, rstride=1, cstride=1, linewidth=0.3)
```

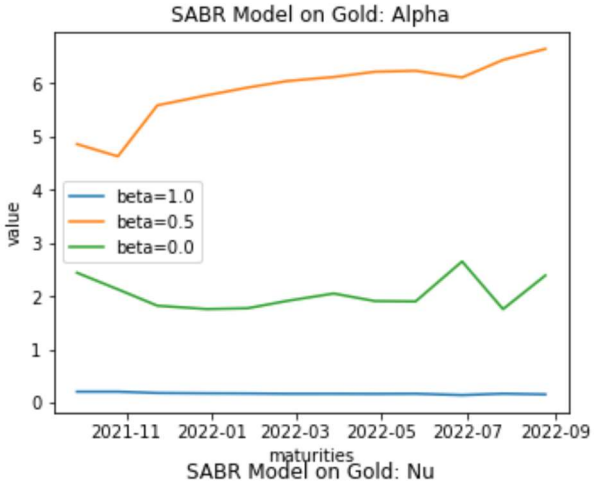
```
class SABRSmile:
    def __init__(self, date, shift=0, beta=1, method="normal", fwd=current_price, ze
        self.date = date
        self.expiryTime = round((self.date - today)/365, 6)
        self.marketVols = vols[dates.index(self.date)]
        self.shift = shift
        self.fwd = fwd
        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), (-1, 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 strikes]

    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 == "floch-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

        # beta = self.beta
        # alpha = max(alpha, 1e-8) # Avoid alpha going negative
        # nu = max(nu, 1e-8) # Avoid nu going negative
        # rho = max(rho, -0.999) if self.zero_rho==False else 0.0 # Avoid rhp going
        # rho = min(rho, 0.999) # Avoid rho going > 1.0

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

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

        return self.error


class SABRVolatilitySurface:
```

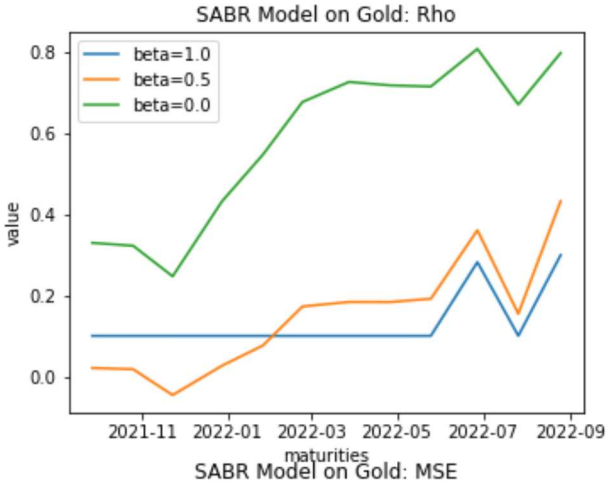
```
    def __init__(self, method="normal", beta=1, shift=0, fwd=current_price, label=""
        self.method = method
        self._beta = beta
        self.shift = shift
        self.fwd = fwd
        self.label = label
        self.zero_rho = zero_rho

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

        for i, d in enumerate(dates):
            volSABR = SABRSmile(date=d, beta=self._beta, shift=self.shift,
                                method=self.method, fwd=self.fwd, zero_rho=self.zero
            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] - vols[i][j]) / vols[i][j]

            self.vol_surface = ql.BlackVarianceSurface(
                today, calendar, dates, 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, fixes=(.95, 1, 1.14, 1.24), ten
    l = []
    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))
```

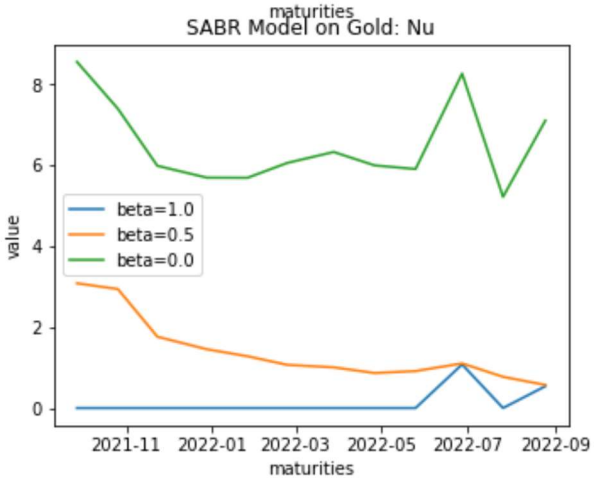
```
def SABRComparison(methods, title="", display=False):
    fig, axs = plt.subplots(2, 2, figsize=plot_size)
    plt.subplots_adjust(left=None, bottom=None, right=None,
                        top=1.5, wspace=None, hspace=None)

    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)
```

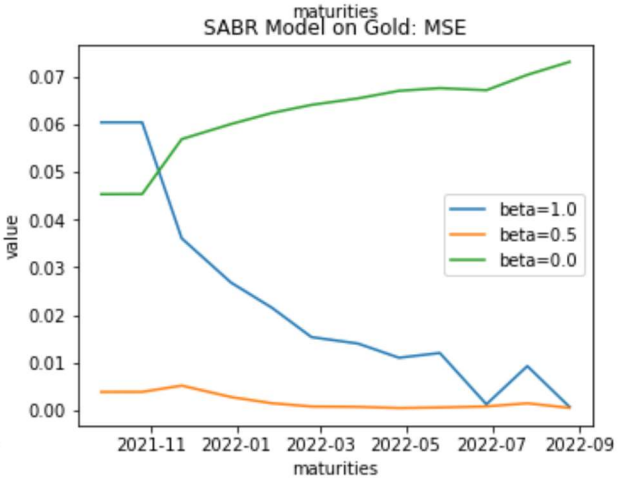
In [9]:

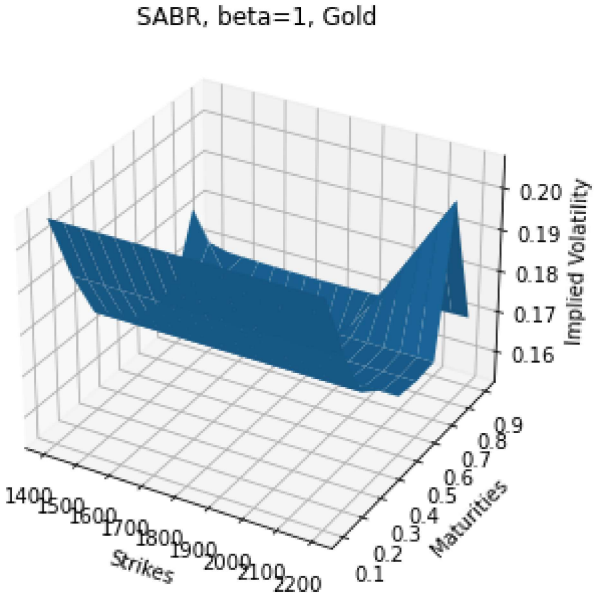
```
# SABR Volatility model

SABR_beta1 = SABRVolatilitySurface(beta=1, shift=0, label="SABR, beta=1, {}".format(
SABR_beta5 = SABRVolatilitySurface(beta=.5, shift=0, label="SABR, beta=.5 {}".format
SABR_beta0 = SABRVolatilitySurface(beta=.0, shift=0, label="Normal SABR, beta=0, {}"

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


SABR Model on Gold: Alpha
value
maturities
beta=1.0
beta=0.5
beta=0.0
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09

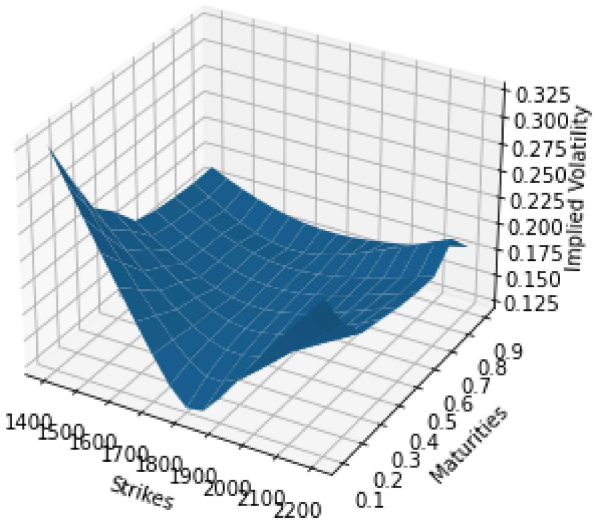

SABR Model on Gold: Rho
value
maturities
beta=1.0
beta=0.5
beta=0.0
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09


SABR Model on Gold: Nu
value
maturities
beta=1.0
beta=0.5
beta=0.0
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09

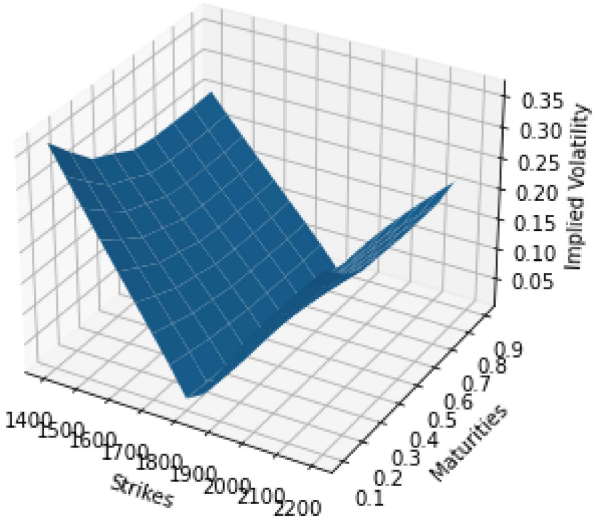

SABR Model on Gold: MSE
value
maturities
beta=1.0
beta=0.5
beta=0.0
2021-11
2022-01
2022-03
2022-05
2022-07
2022-09

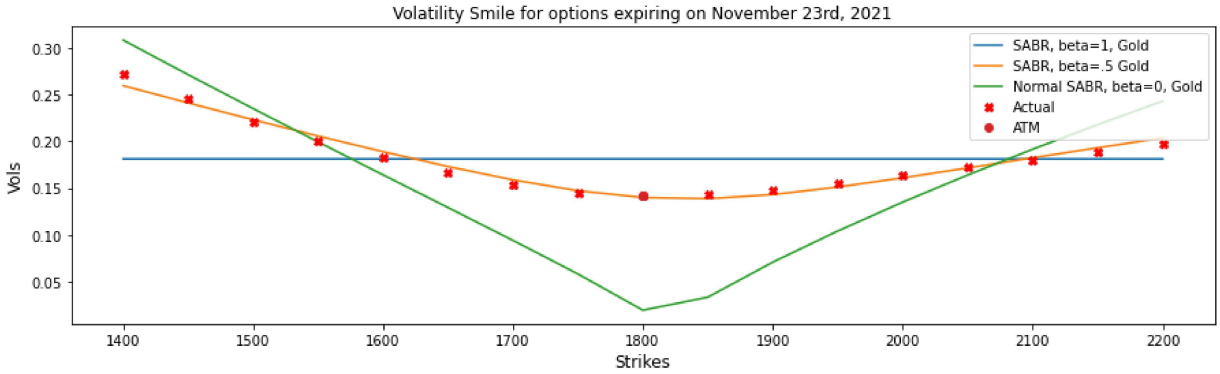

SABR, beta=1, Gold
Implied Volatility
Strikes
Maturities

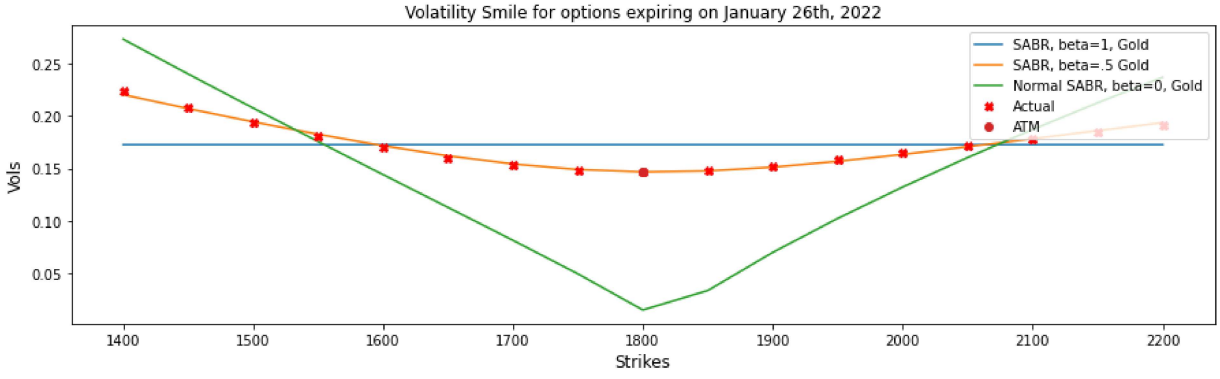
SABR, beta=.5 Gold

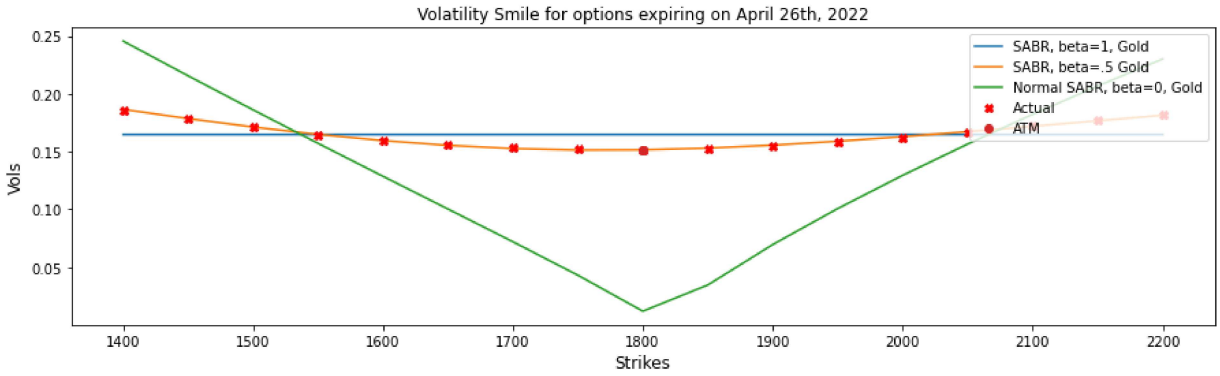


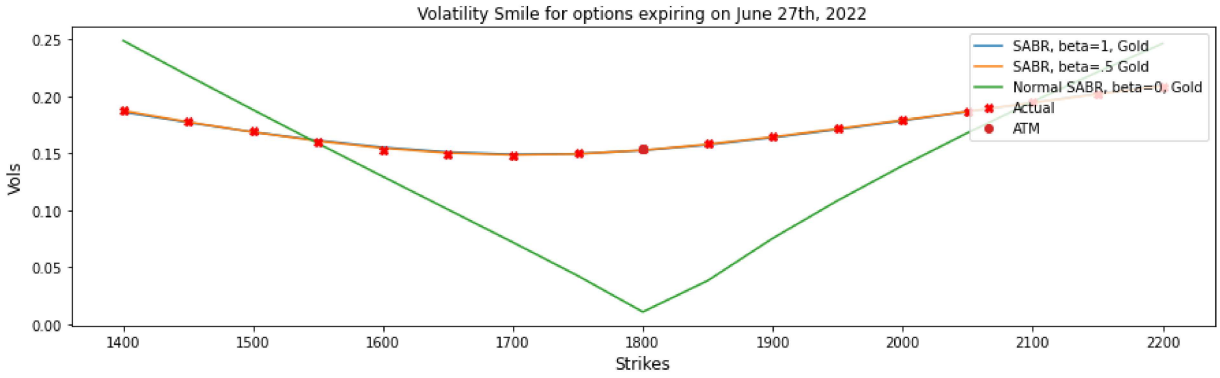
Normal SABR, beta=0, Gold

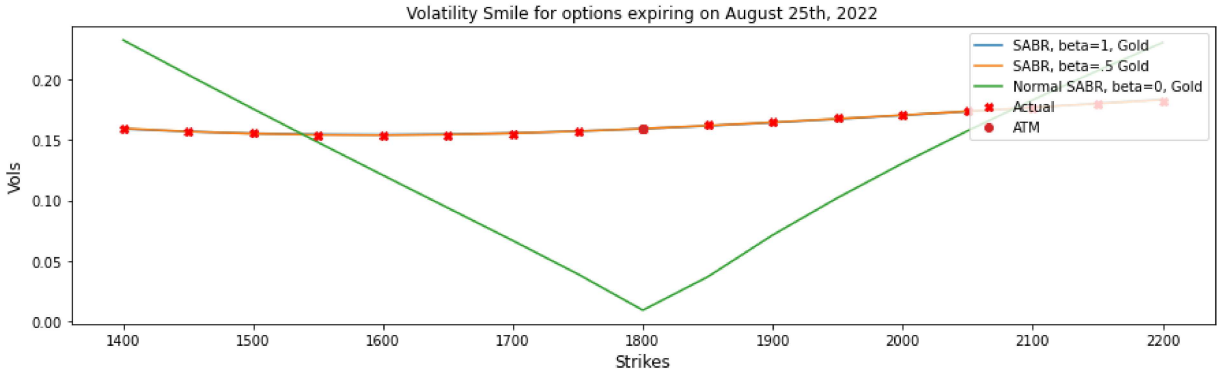








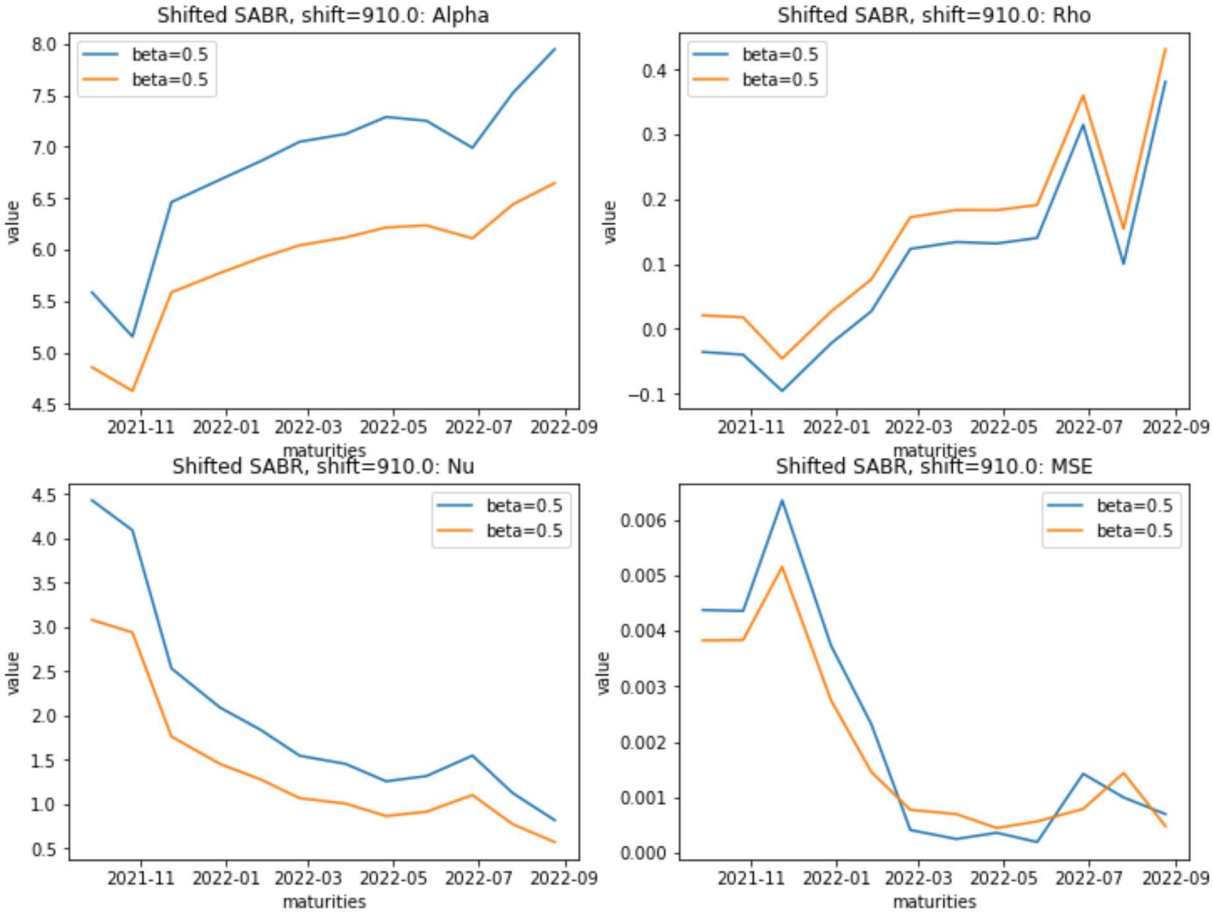




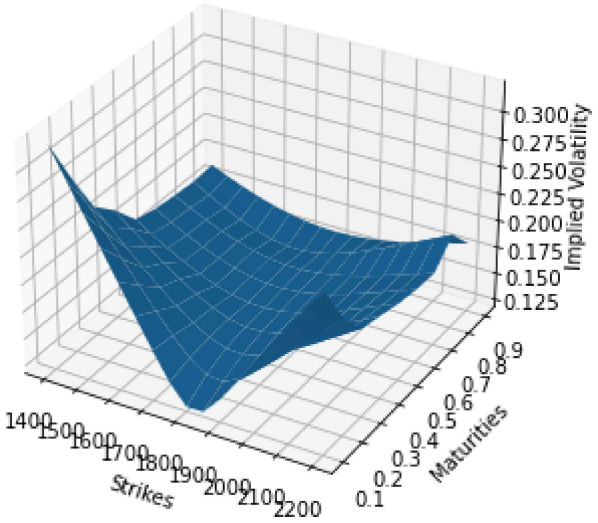
In [10]:

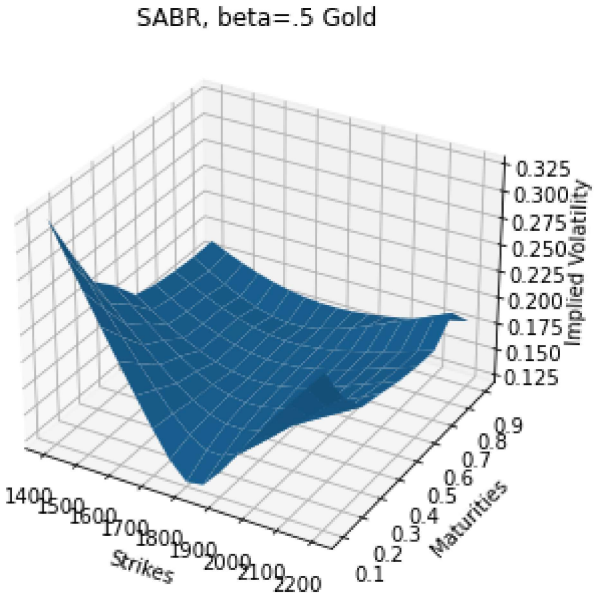
```
# Shifted SABR Volatility model
shft = .50 * current_price
shiftedSABR_beta1 = SABRVolatilitySurface(beta=1, shift=shft, label="Shifted SABR, b
shiftedSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=shft, label="Shifted SABR,
shiftedSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=shft, label="Shifted Normal

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

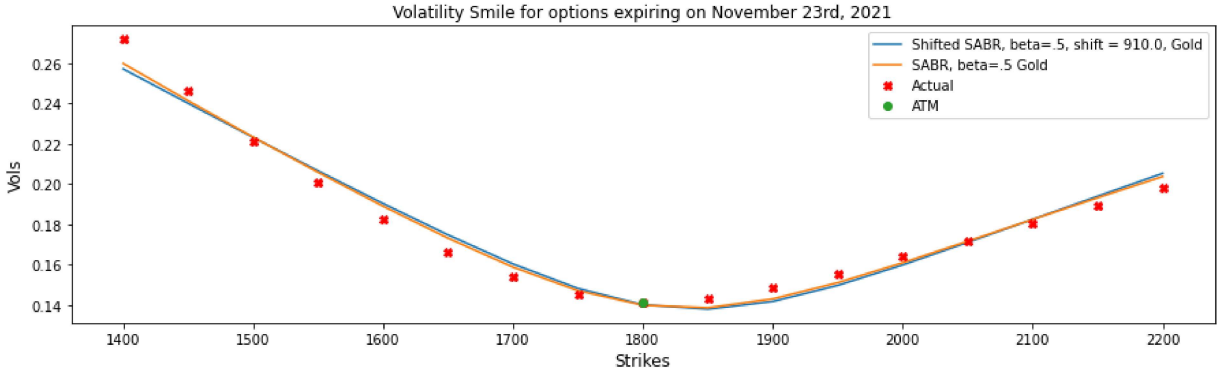
```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWa
rning: Values in x were outside bounds during a minimize step, clipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```

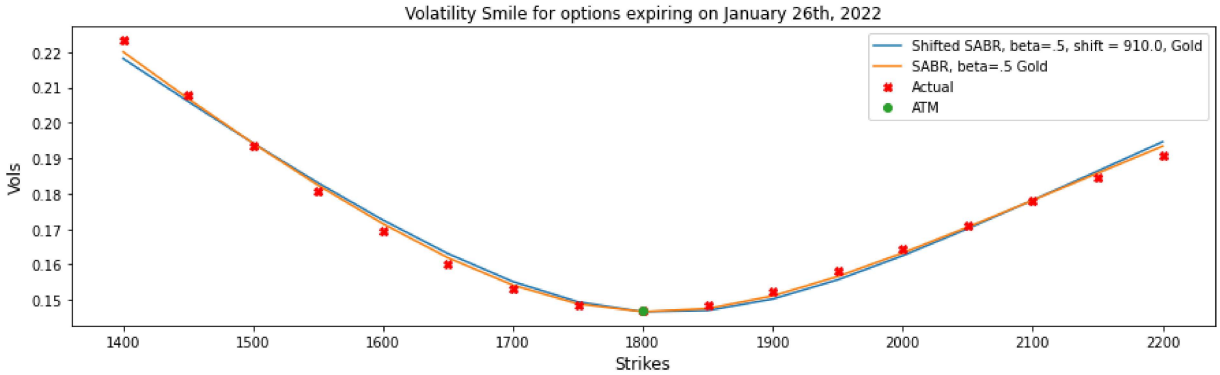


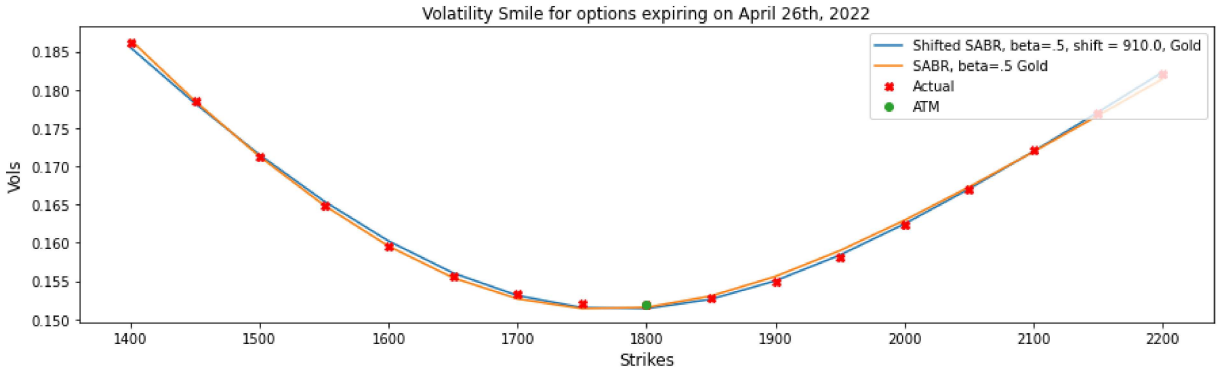
Shifted SABR, beta=.5, shift = 910.0, Gold

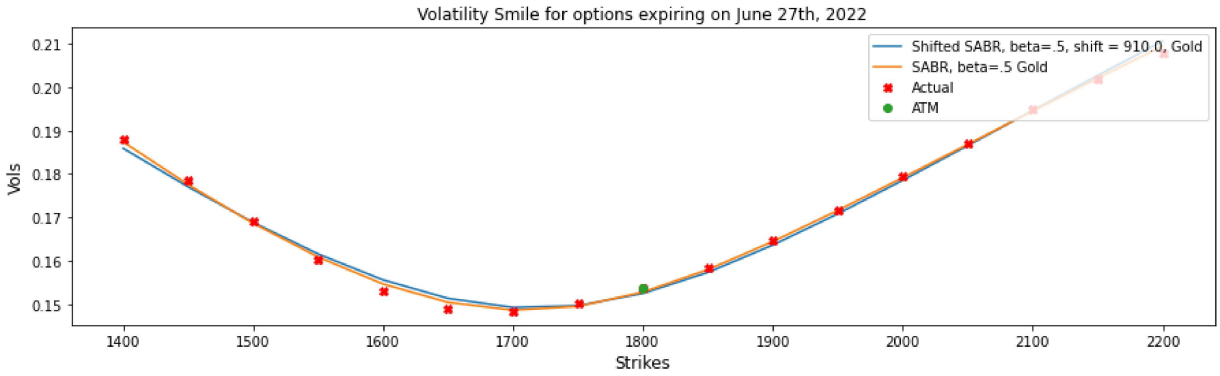


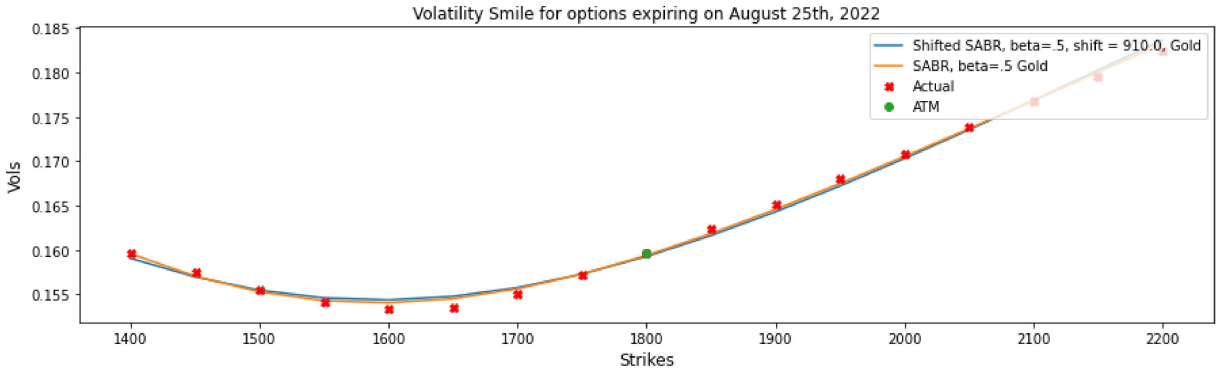

SABR, beta=.5 Gold
Implied Volatility
0.325
0.300
0.275
0.250
0.225
0.200
0.175
0.150
0.125
Strikes
1400
1500
1600
1700
1800
1900
2000
2100
2200
Maturities
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

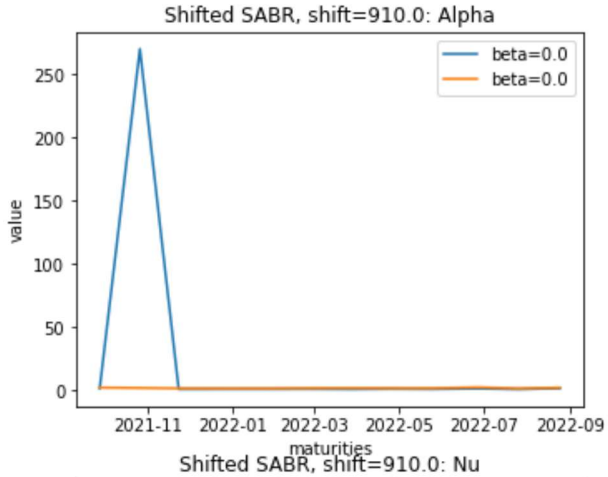

Volatility Smile for options expiring on November 23rd, 2021
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

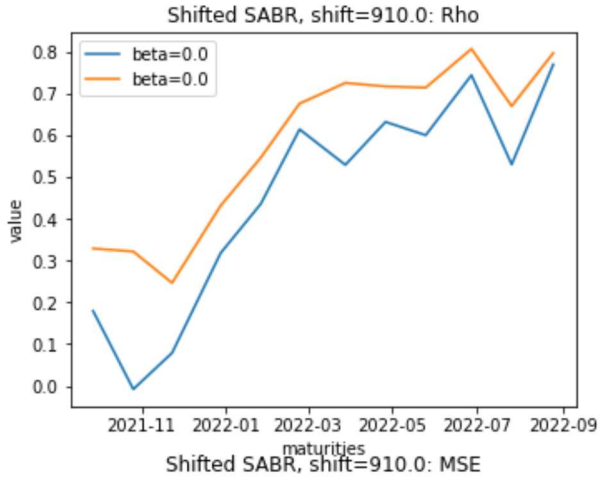

Volatility Smile for options expiring on January 26th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

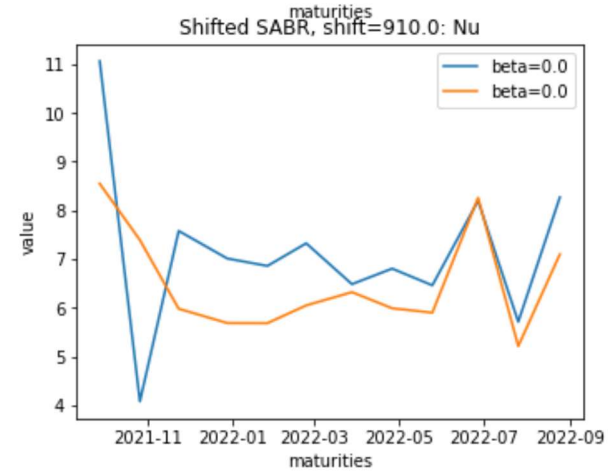

Volatility Smile for options expiring on April 26th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

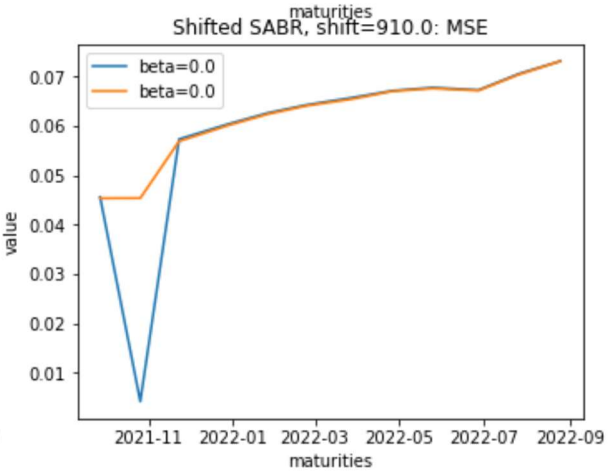

Volatility Smile for options expiring on June 27th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes


Volatility Smile for options expiring on August 25th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

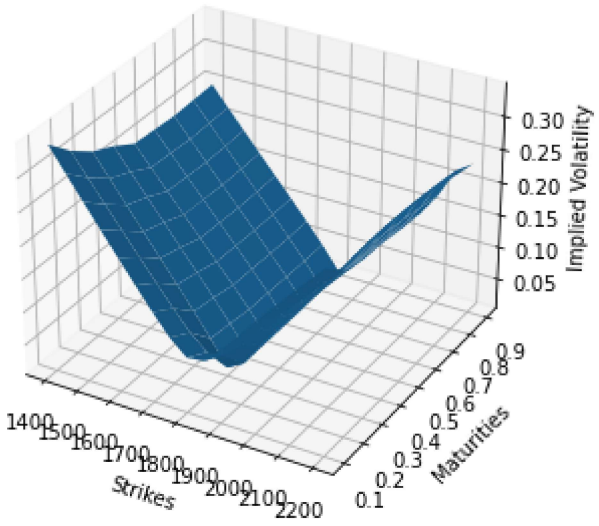

Shifted SABR, shift=910.0: Alpha
beta=0.0
beta=0.0
value
maturities


Shifted SABR, shift=910.0: Rho
beta=0.0
beta=0.0
value
maturities

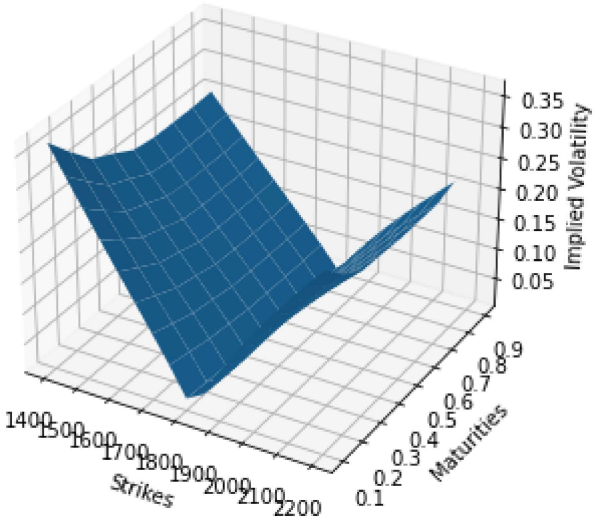

Shifted SABR, shift=910.0: Nu
beta=0.0
beta=0.0
value
maturities

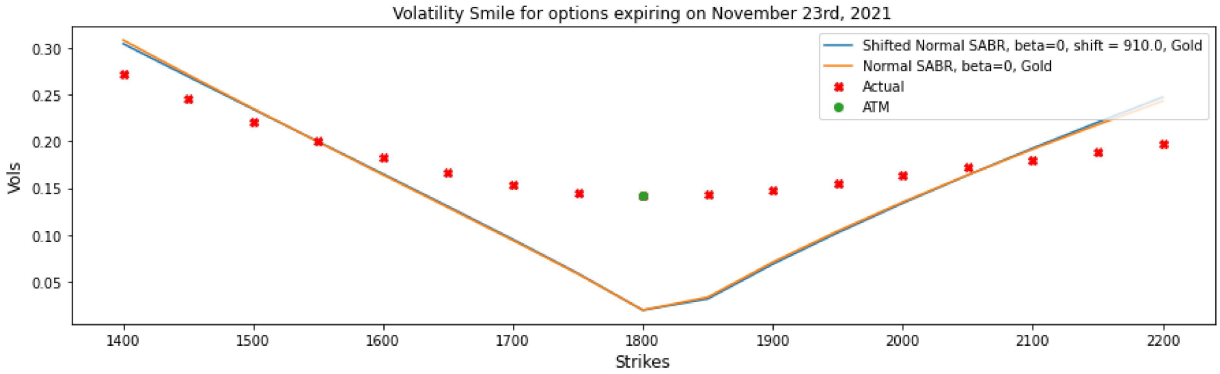

Shifted SABR, shift=910.0: MSE
beta=0.0
beta=0.0
value
maturities

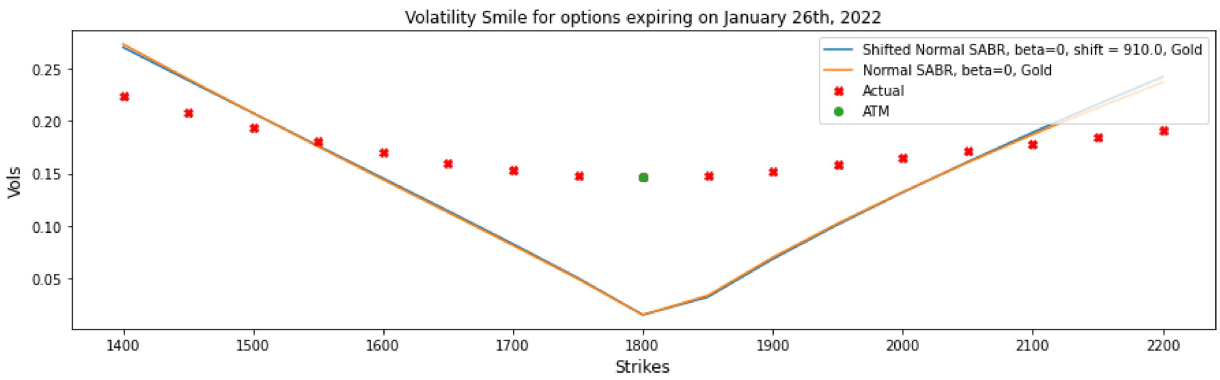
Shifted Normal SABR, beta=0, shift = 910.0, Gold

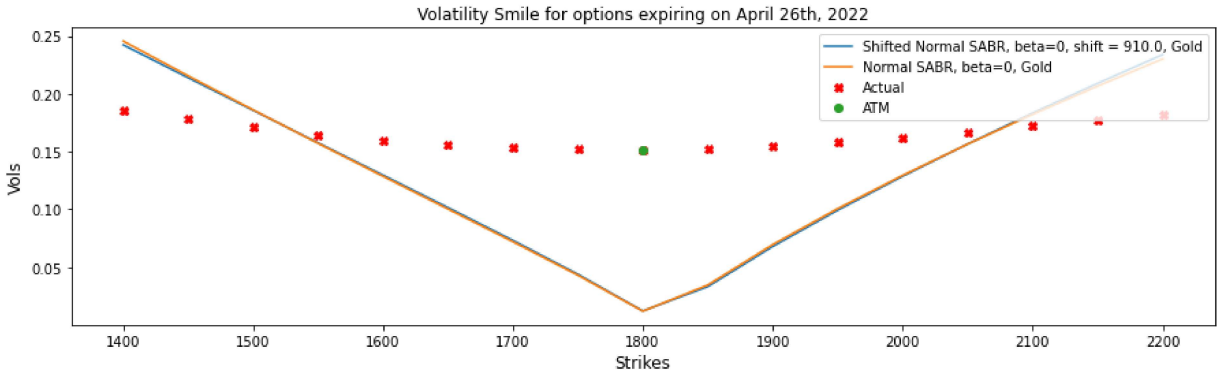


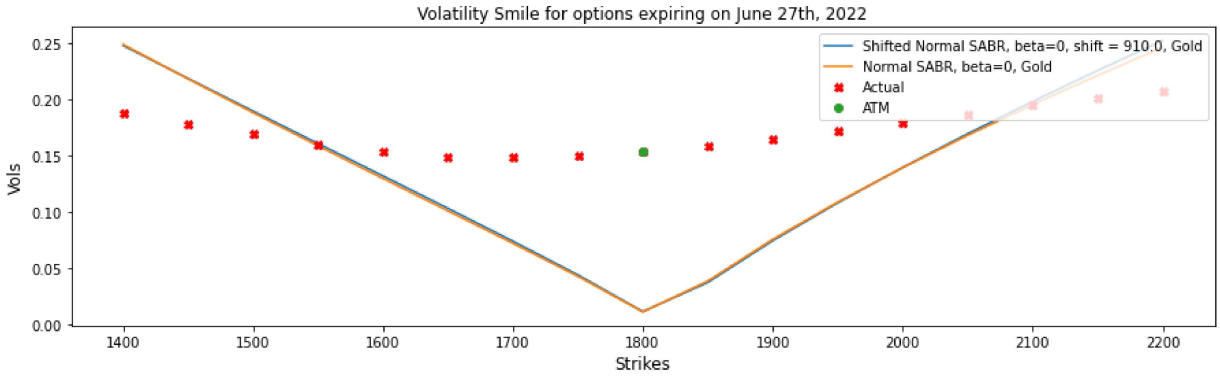
Normal SABR, beta=0, Gold

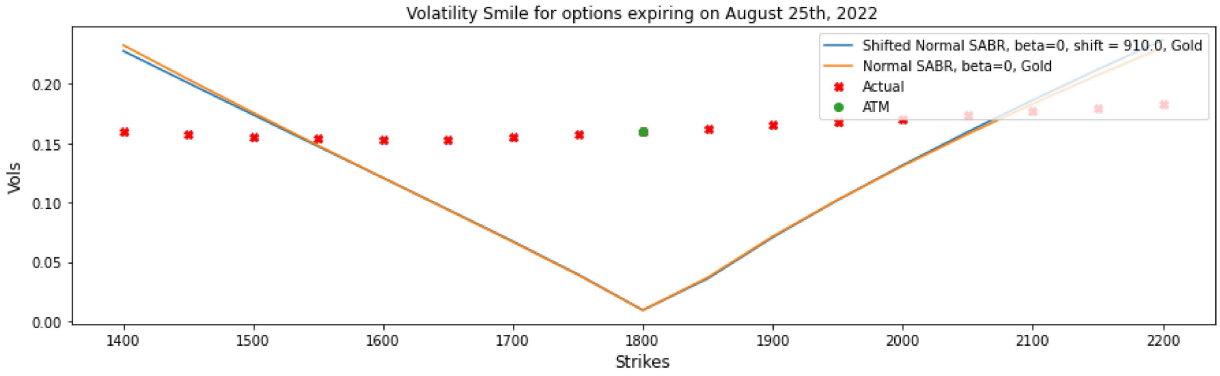








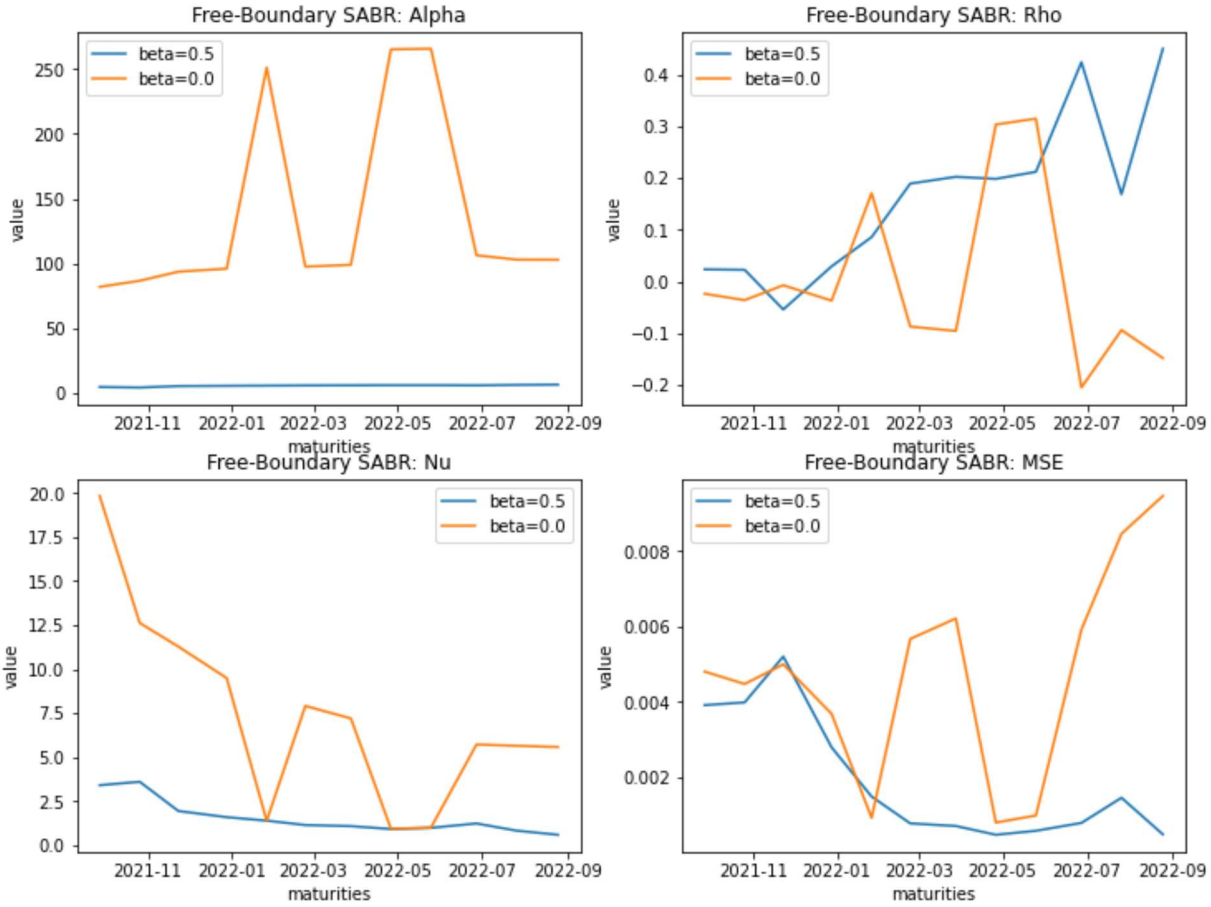




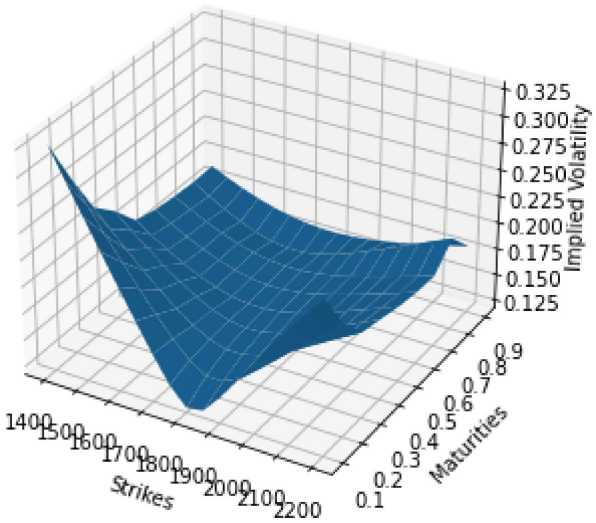
In [11]:

```
# Free-Boundary SABR Volatility model

freeSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=0, method="floch-kennedy", lab
freeSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=0, method="floch-kennedy", lab

SABRComparison([freeSABR_beta5, freeSABR_beta0], title="Free-Boundary SABR")
```

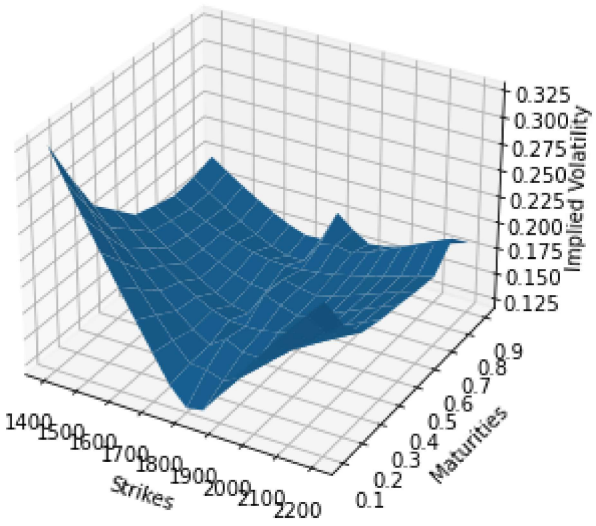
```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWa
rning: Values in x were outside bounds during a minimize step, clipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```

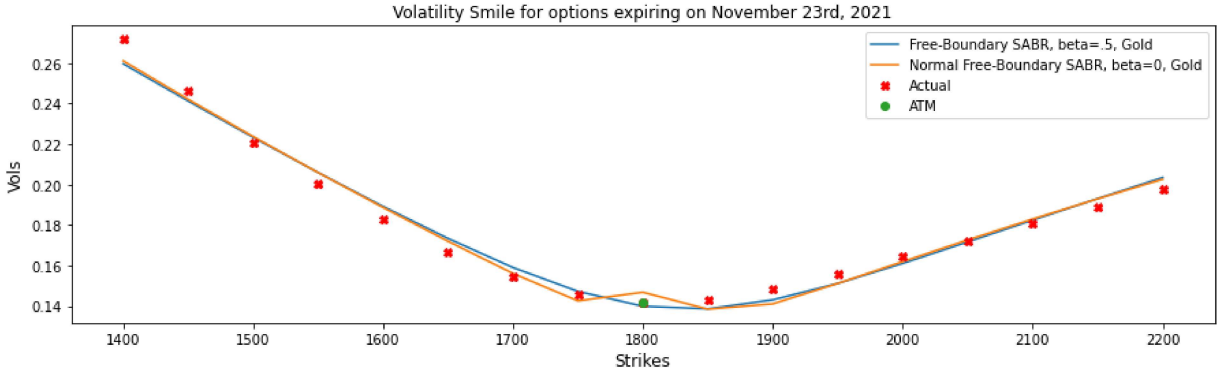


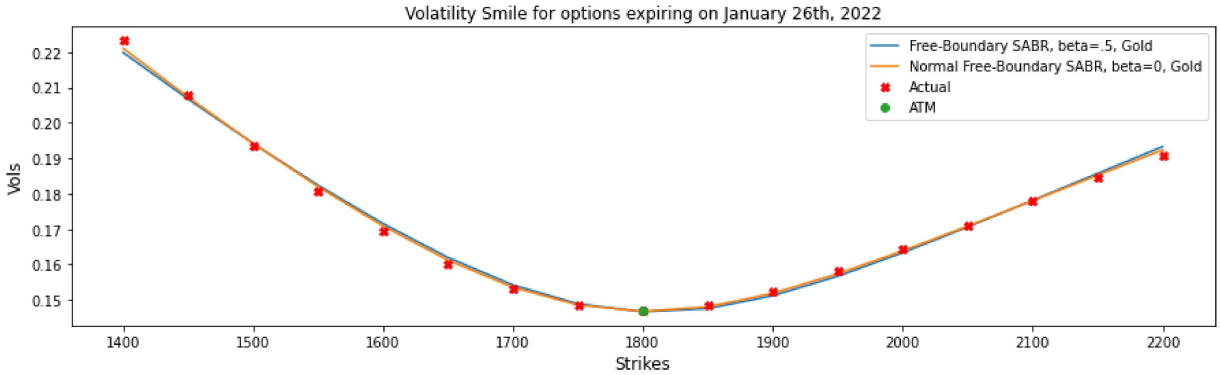
Free-Boundary SABR, beta=.5, Gold

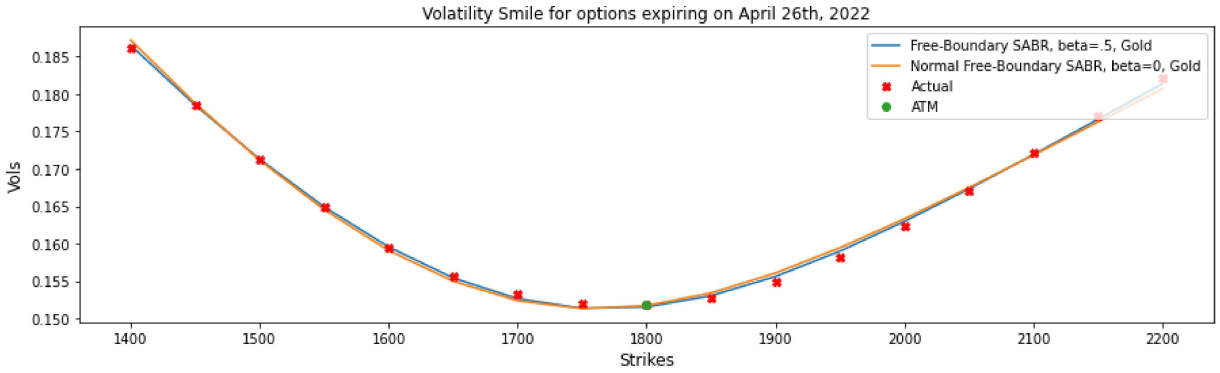


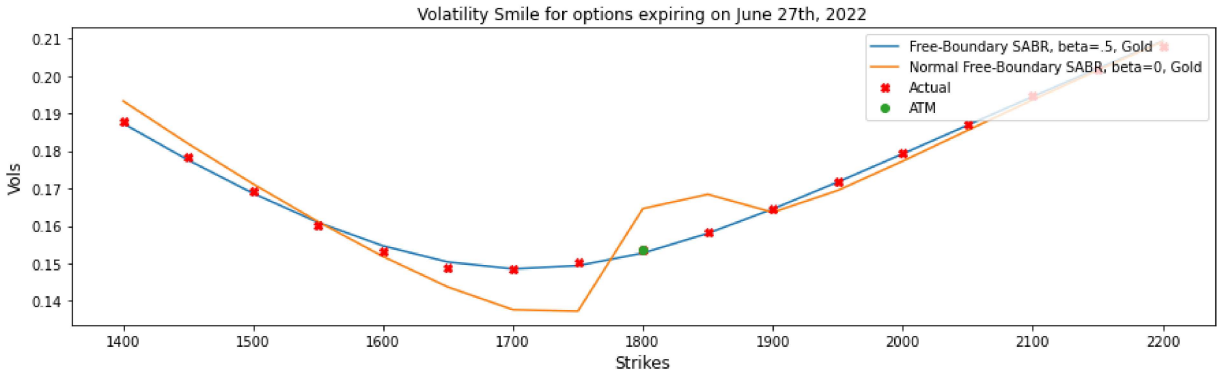
Normal Free-Boundary SABR, beta=0, Gold

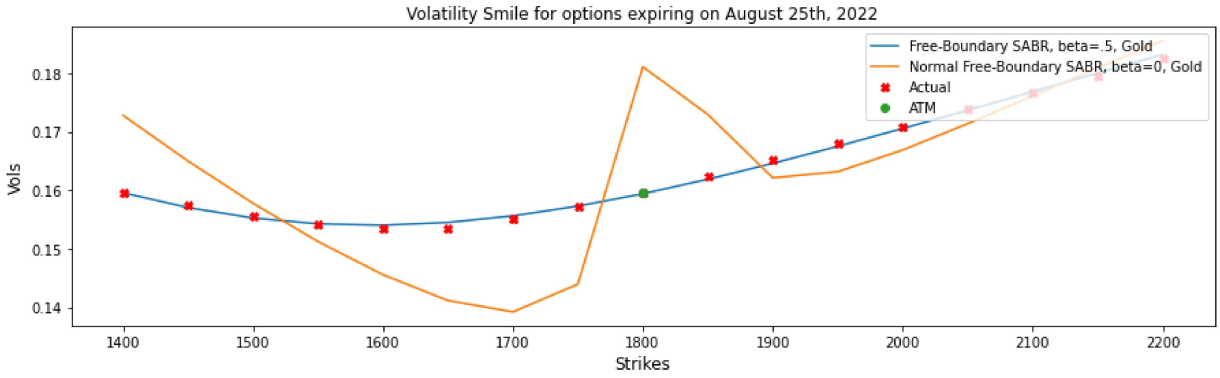












In [12]:

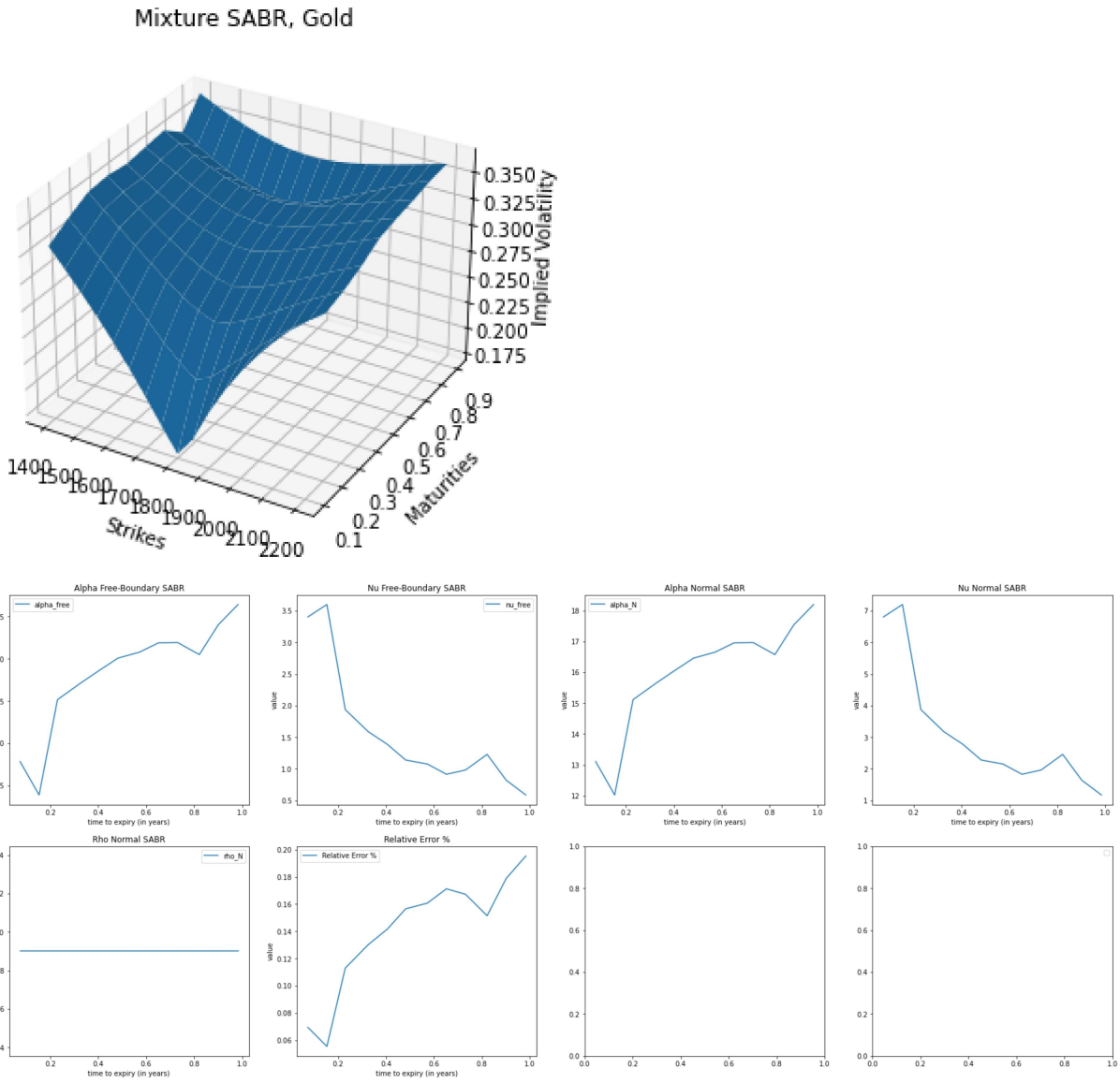
```
# Mixed SABR

tenors = dates[6:] if data == "SPX" else dates
mixtureSABR = MixtureSABRVolatilitySurface(dates=tenors)
display(mixtureSABR.to_data())
plot_vol_surface([mixtureSABR.vol_surface], title=mixtureSABR.label)
mixtureSABR.plot()
```

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

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSE
September 27th, 2021	4.785646	0.5	3.402979	0.0	4.785646	0.0	6.805958	0.99	0.069476
October 26th, 2021	4.389699	0.5	3.599256	0.0	4.389699	0.0	7.198511	0.99	0.055437
November 23rd, 2021	5.516388	0.5	1.938113	0.0	5.516388	0.0	3.876227	0.99	0.113028
December 28th, 2021	5.711042	0.5	1.588608	0.0	5.711042	0.0	3.177216	0.99	0.130247
January 26th, 2022	5.863852	0.5	1.389392	0.0	5.863852	0.0	2.778784	0.99	0.141623
February 23rd, 2022	6.006846	0.5	1.139989	0.0	6.006846	0.0	2.279978	0.99	0.156587
March 28th, 2022	6.078232	0.5	1.077586	0.0	6.078232	0.0	2.155172	0.99	0.160735
April 26th, 2022	6.188313	0.5	0.915616	0.0	6.188313	0.0	1.831232	0.99	0.171255
May 25th, 2022	6.191132	0.5	0.982120	0.0	6.191132	0.0	1.964240	0.99	0.167289

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSE
June 27th, 2022	6.048971	0.5	1.229845	0.0	6.048971	0.0	2.459691	0.99	0.151487
July 26th, 2022	6.402704	0.5	0.822404	0.0	6.402704	0.0	1.644807	0.99	0.178853
August 25th, 2022	6.640873	0.5	0.587827	0.0	6.640873	0.0	1.175653	0.99	0.195413

```
No handles with labels found to put in legend.
```

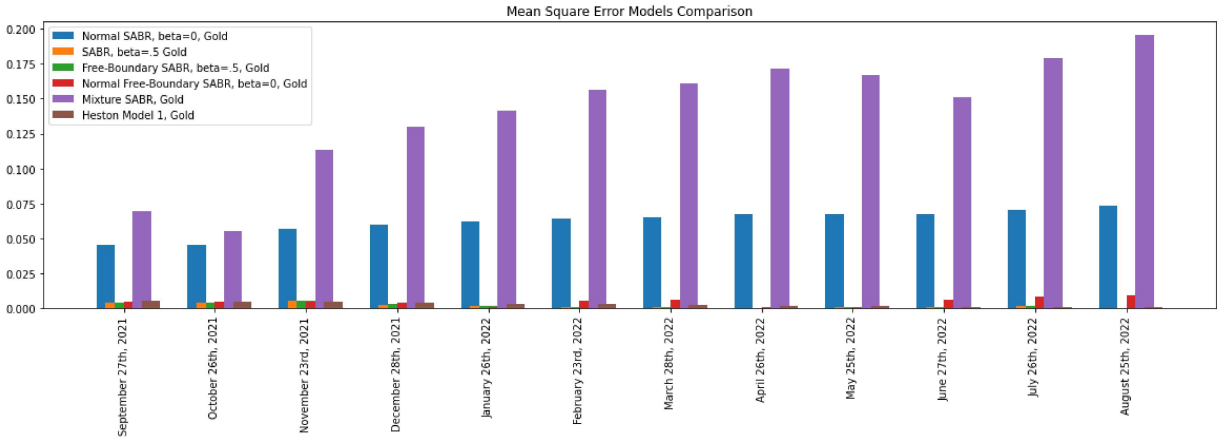


In [13]:

```
# VOLATILITY SMILES ERRORS COMPARISON


# fix Mixture SABR errors due to missing dates
mixtureSABR_errors = None
if data == "SPX":
    np.concatenate((np.zeros(6), mixtureSABR.errors))
else:
     mixtureSABR_errors = mixtureSABR.errors

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()
```

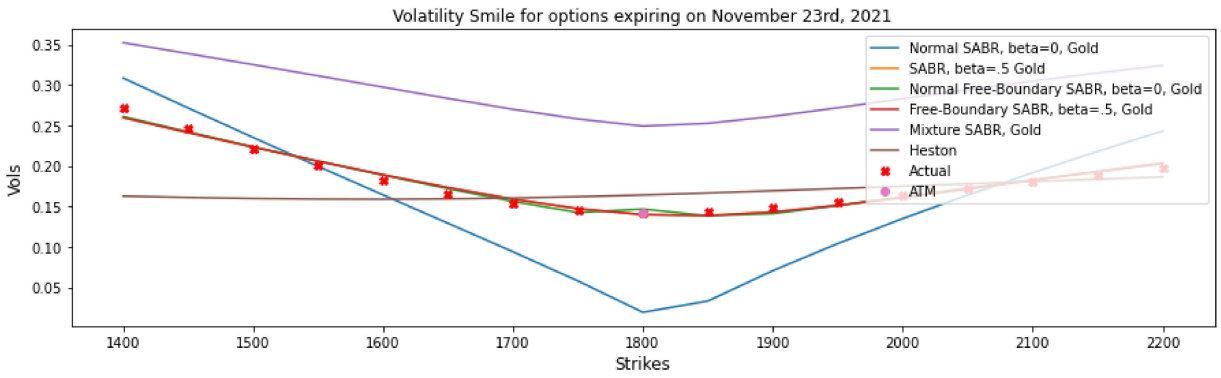
Out[13]: <matplotlib.legend.Legend at 0x2ae51b37100>

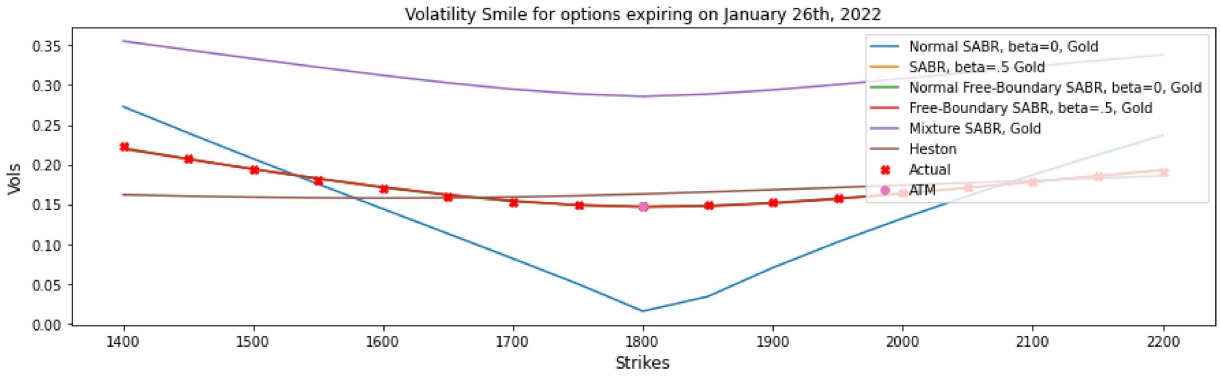


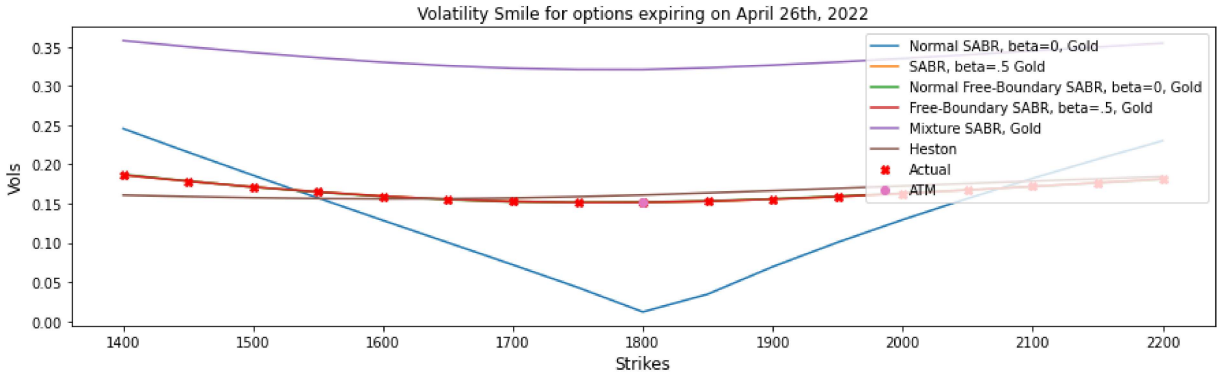
In [14]:

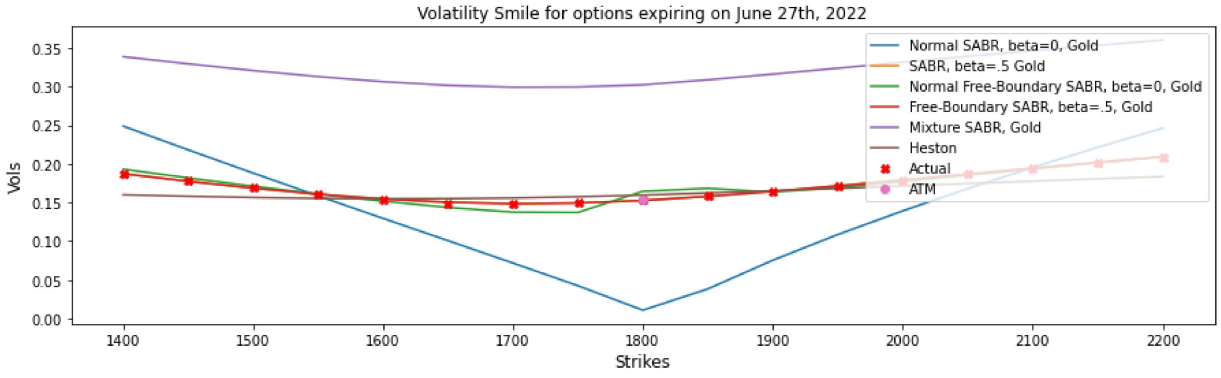
```
# Volatility Smiles Comparisons (Final)

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
display(errors_data)

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

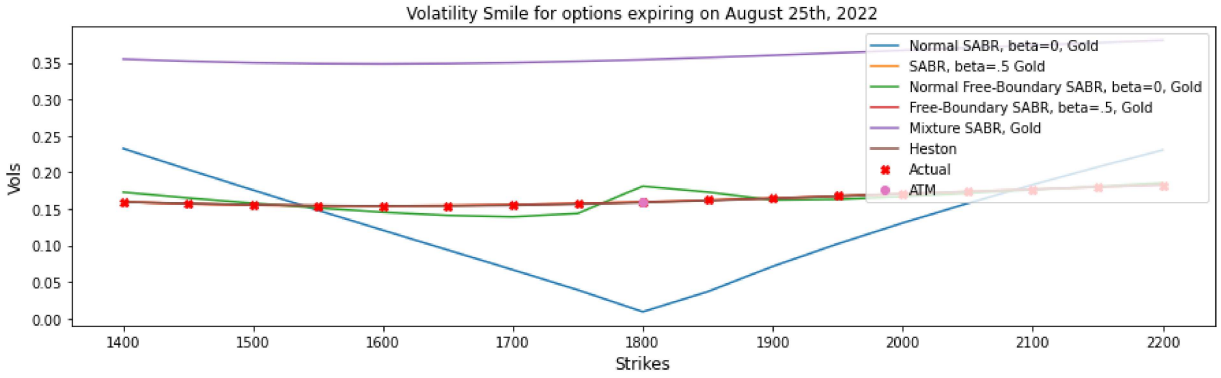
	Error
Normal SABR, beta=0, Gold	0.062036
SABR, beta=.5 Gold	0.001851
Normal Free-Boundary SABR, beta=0, Gold	0.004694
Free-Boundary SABR, beta=.5, Gold	0.001877
Mixture SABR, Gold	0.140952







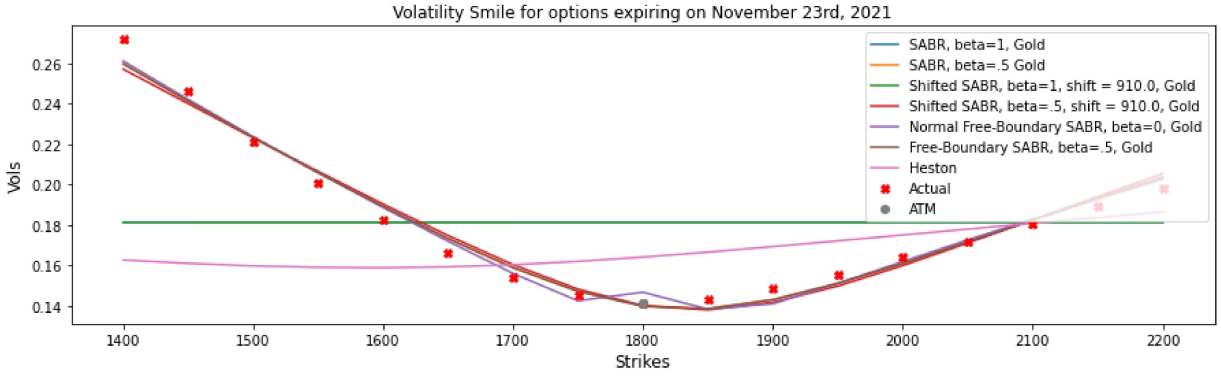


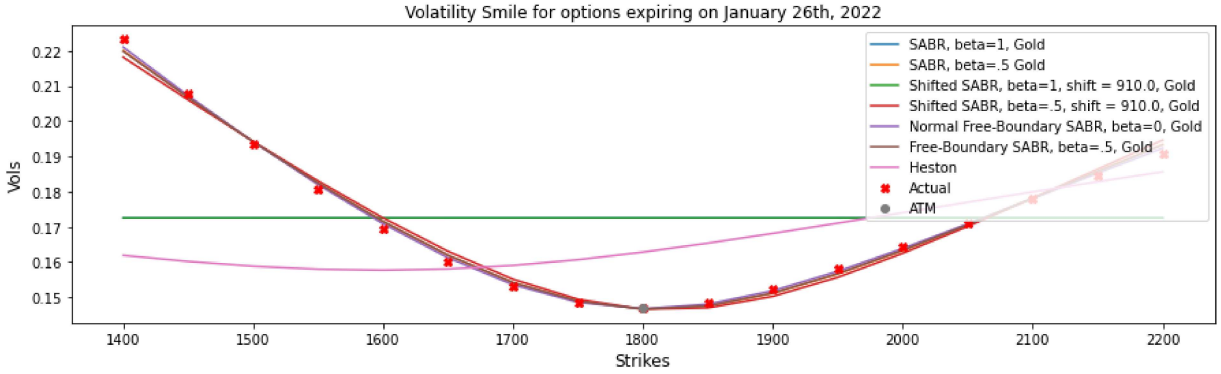


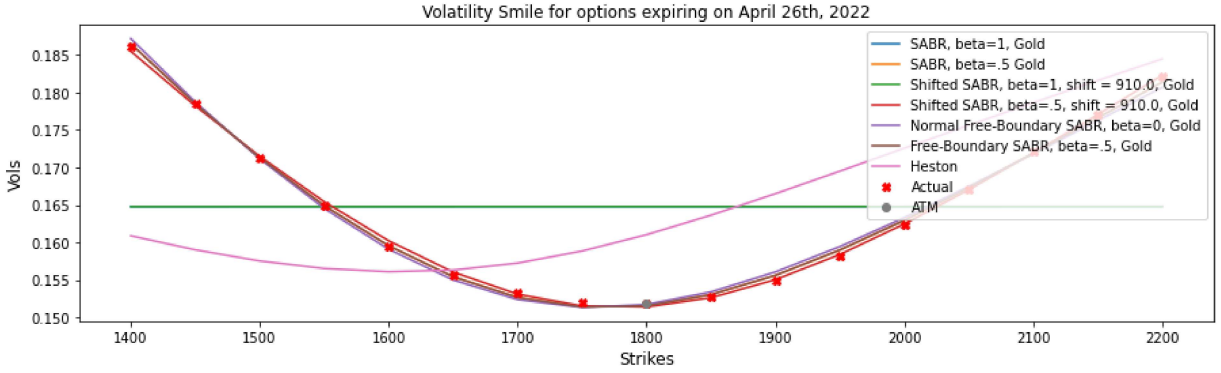
In [15]:

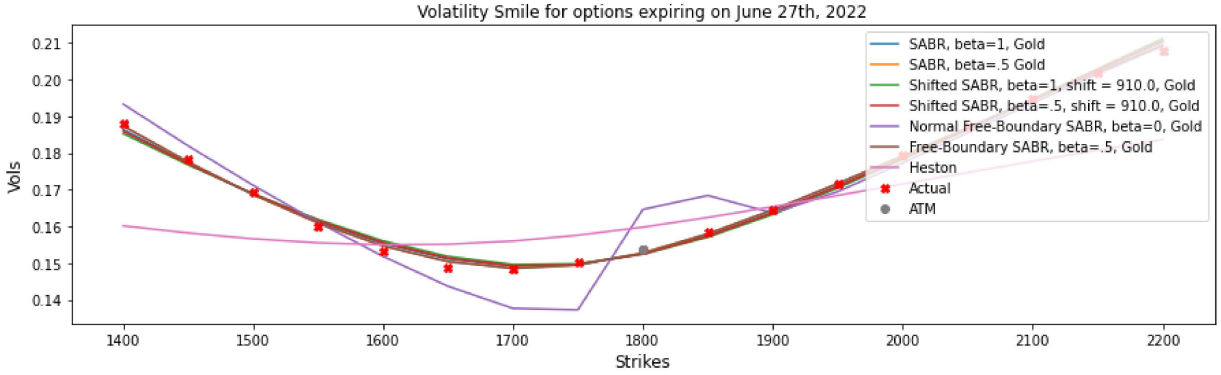
```
# Volatility Smiles Comparisons 2 (Final)

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])
```

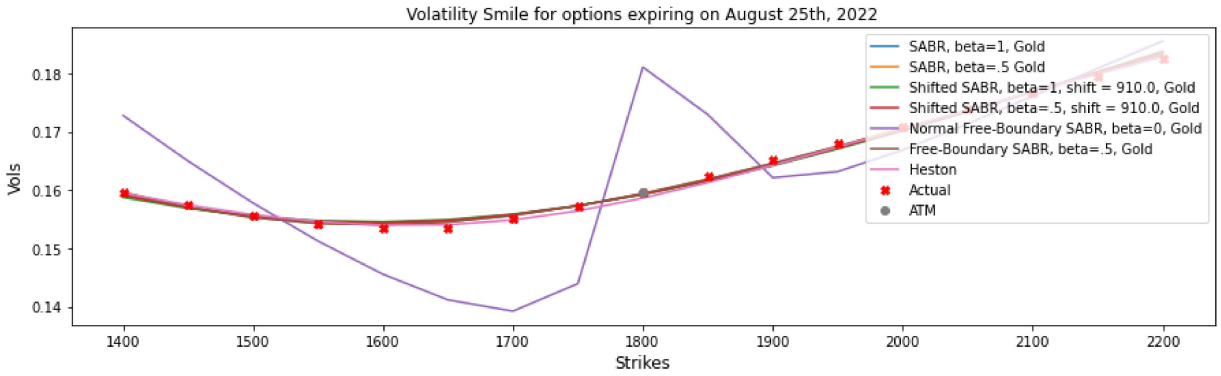
	Error
SABR, beta=1, Gold	0.022375
SABR, beta=.5 Gold	0.001851
Shifted SABR, beta=1, shift = 910.0, Gold	0.022431
Shifted SABR, beta=.5, shift = 910.0, Gold	0.002122
Normal Free-Boundary SABR, beta=0, Gold	0.004694
Free-Boundary SABR, beta=.5, Gold	0.001877





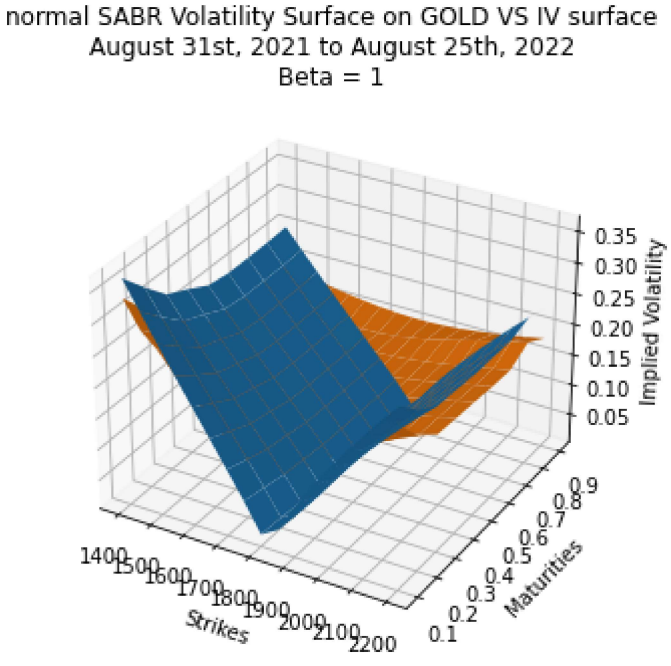




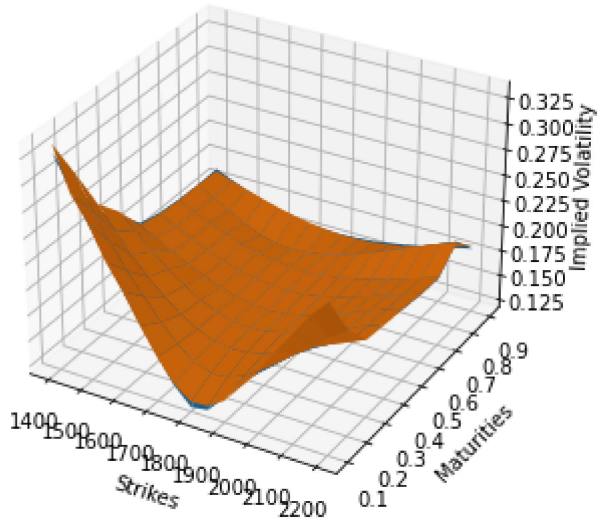


In [16]:

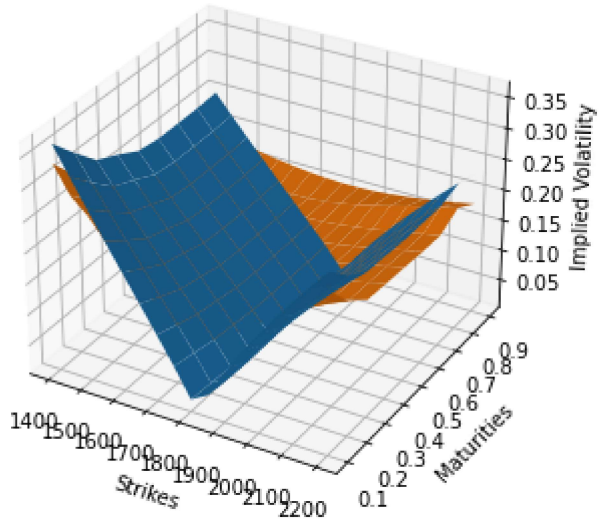
```
# Volatility Surfaces plots comparison

title = "normal SABR Volatility Surface on {} VS IV surface\n{} to {}\nBeta = 1".for
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".f
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".for
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
plot_vol_surface([SABR_beta1.vol_surface, hestonModel1.heston_vol_surface], title=ti

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



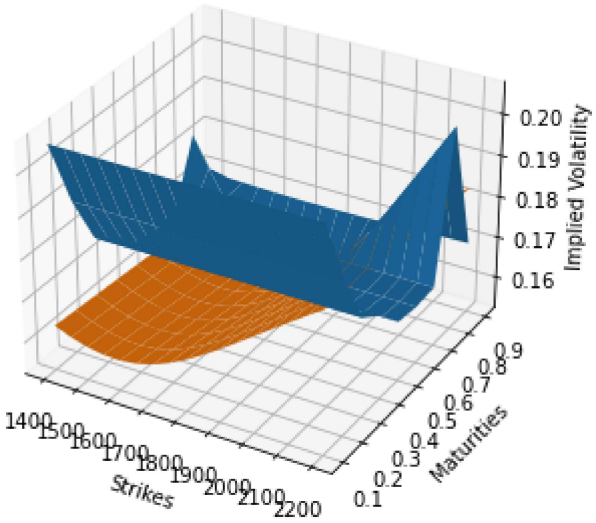
normal SABR Volatility Surface on GOLD VS IV surface
August 31st, 2021 to August 25th, 2022
Beta = 0.5

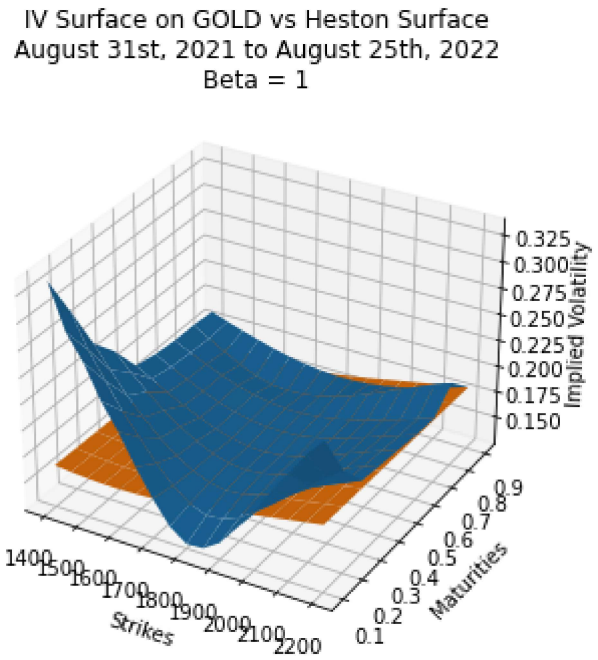


normal SABR Volatility Surface on GOLD VS IV surface
August 31st, 2021 to August 25th, 2022
Beta = 0



SABR Volatility Surface on GOLD VS Heston surface
August 31st, 2021 to August 25th, 2022
Beta = 1

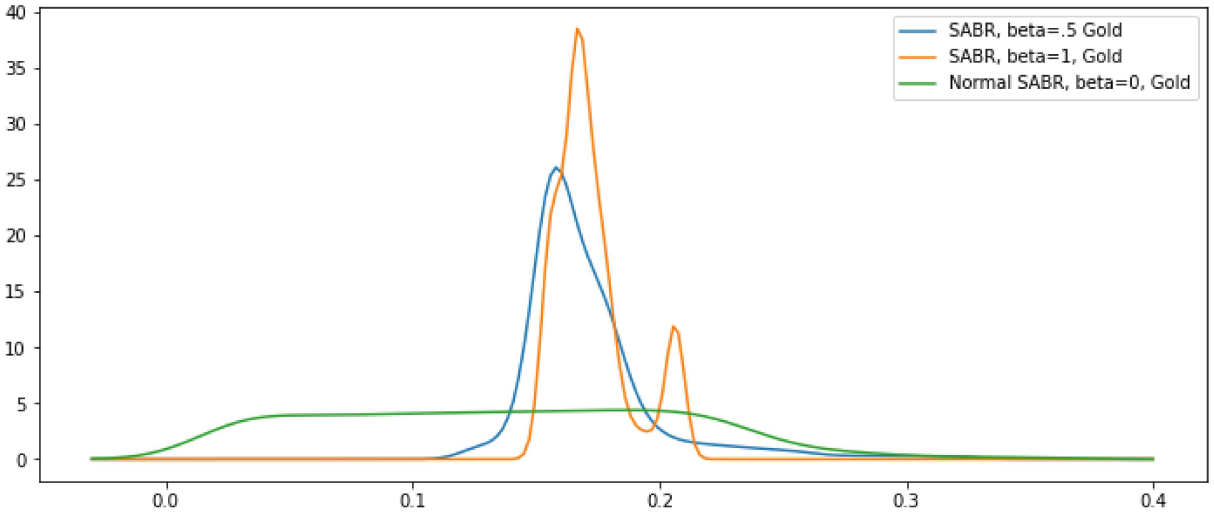




In [17]:

```
from scipy.stats import gaussian_kde
import seaborn as sns
plt.figure(figsize=plot_size)
def density(model):
    rn = []
    for i in np.arange(0, 1.5, .01):
        for j in np.arange(model.vol_surface.minStrike(), model.vol_surface.maxStrik
            rn.append(model.vol_surface.blackVol(i,j))

    density = gaussian_kde(rn)
    xs = np.linspace(-.03,.4,200)
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs,density(xs), label=model.label)

    plt.legend()



density(SABR_beta5), density(SABR_beta1), density(SABR_beta0),
```

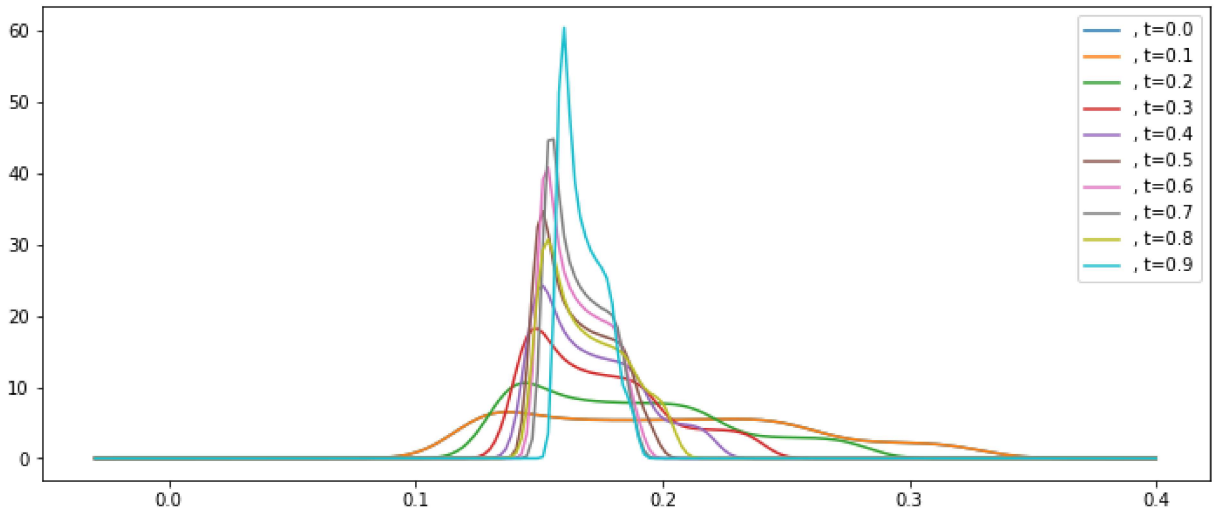
Out[17]: (None, None, None)

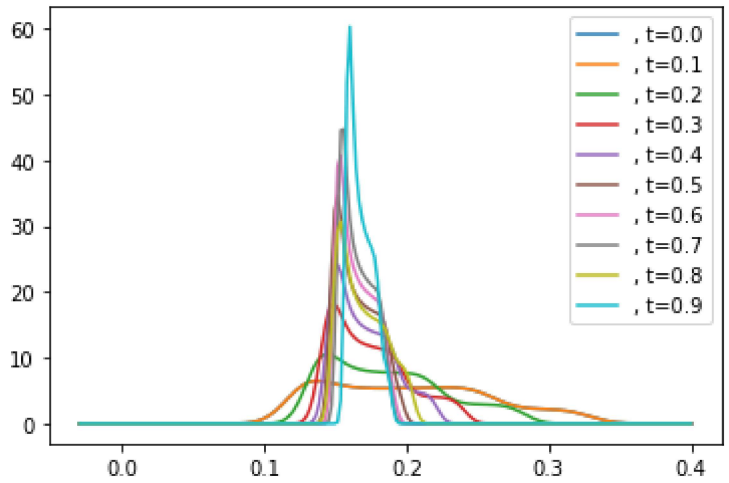


In [18]:

```
from scipy.stats import gaussian_kde
```

```
plt.figure(figsize=plot_size)
def density(model, t):
    rn = []

    for j in np.arange(model.vol_surface.minStrike(), model.vol_surface.maxStrike(),
        rn.append(model.vol_surface.blackVol(t,j))

    density = gaussian_kde(rn)
    xs = np.linspace(-.03,.4,200)
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs,density(xs), label=model.label+", t="+str(round(t, 2)))

    plt.legend()


for i in np.arange(0,1,.1):
    density(SABRVolatilitySurface(beta=.5, zero_rho=True), i)
plt.show()
for i in np.arange(0,1,.1):
    density(SABRVolatilitySurface(beta=.5, zero_rho=False), i)
```





In [ ]: