In [18]:

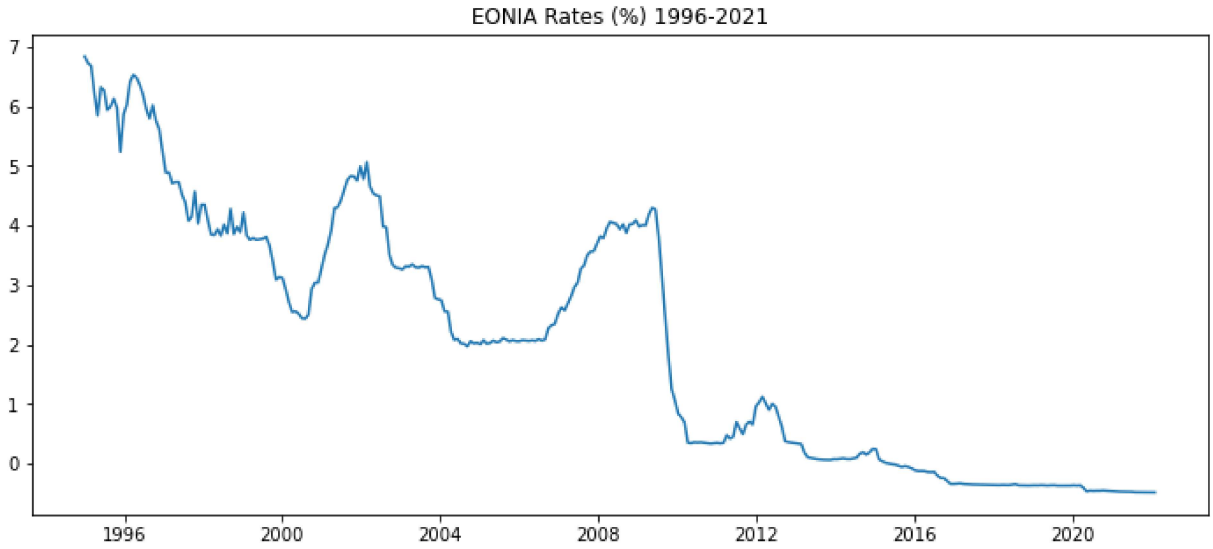
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import time
import datetime
import math
import QuantLib as ql
from scipy.optimize import minimize

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

In [19]:

```
# 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")
```

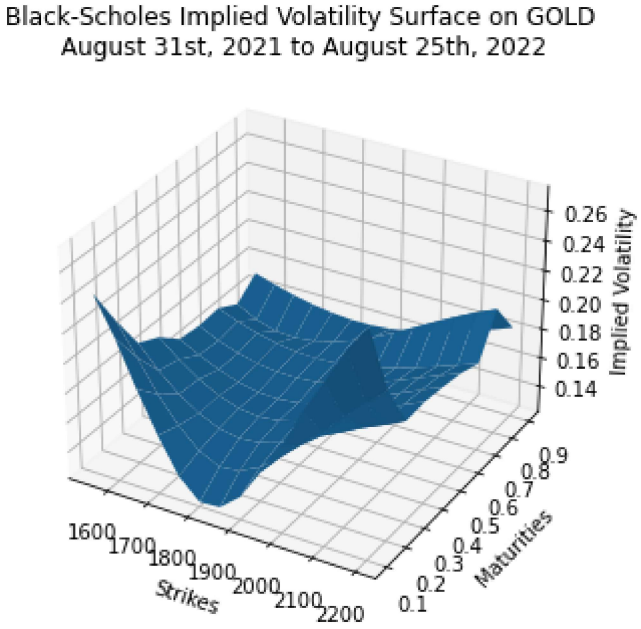
	Tenors	Spot Rate
0	0.25	-0.575
1	1.00	-0.557
2	3.00	-0.549
3	6.00	-0.529
4	12.00	-0.494

Out[19]: Text(0.5, 1.0, 'EONIA Rates (%) 1996-2021')



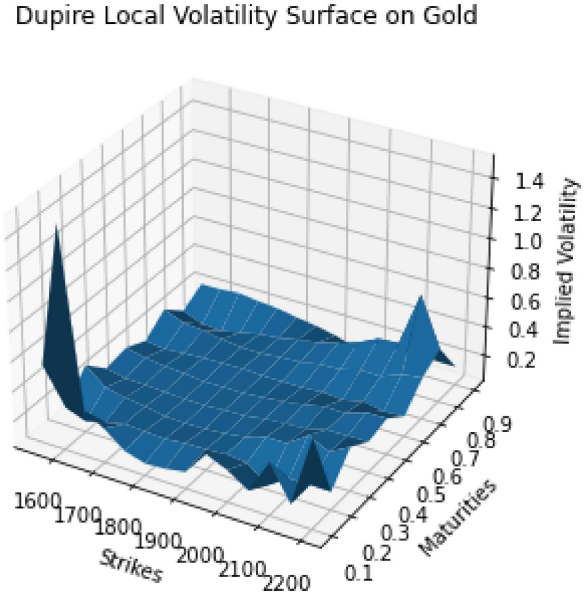
In [20]:

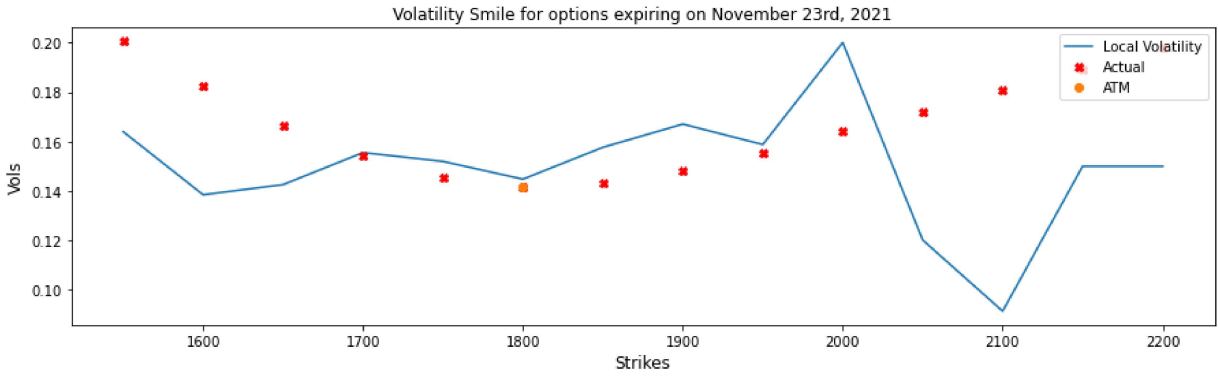
```
# BLACK VOLATILITY SURFACE

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
```

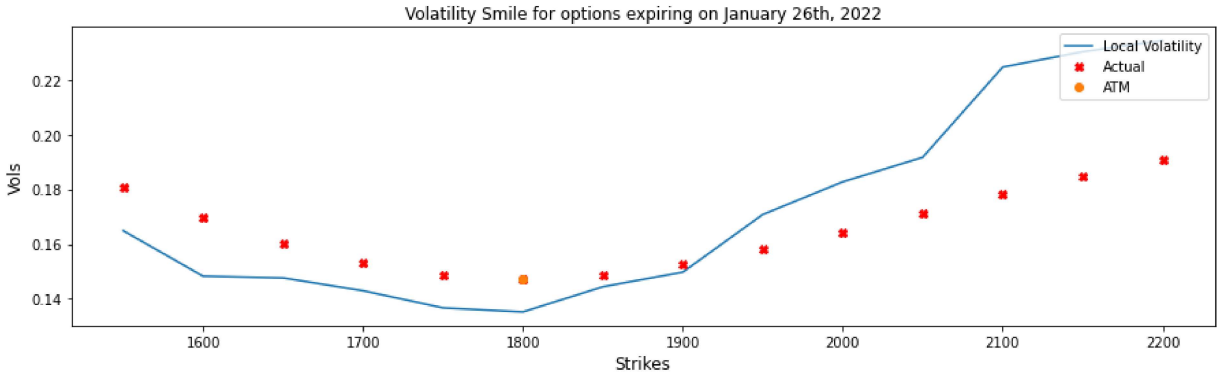


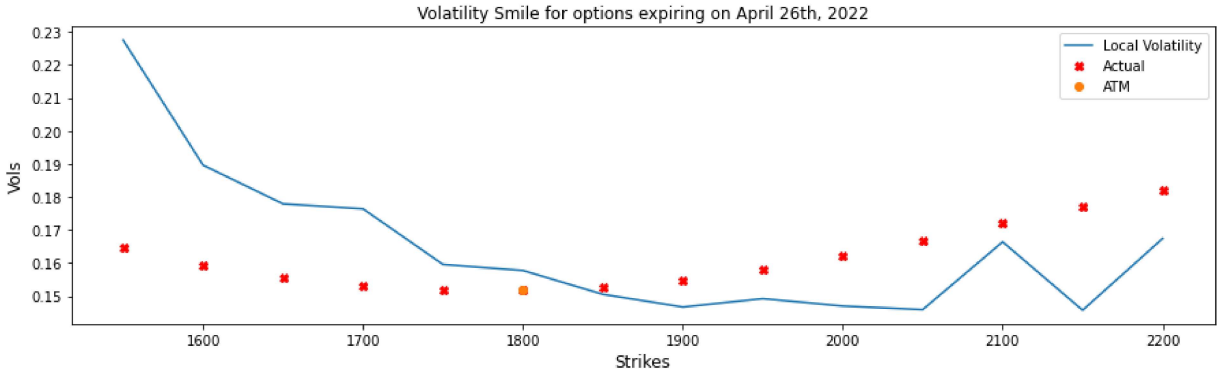
In [21]:

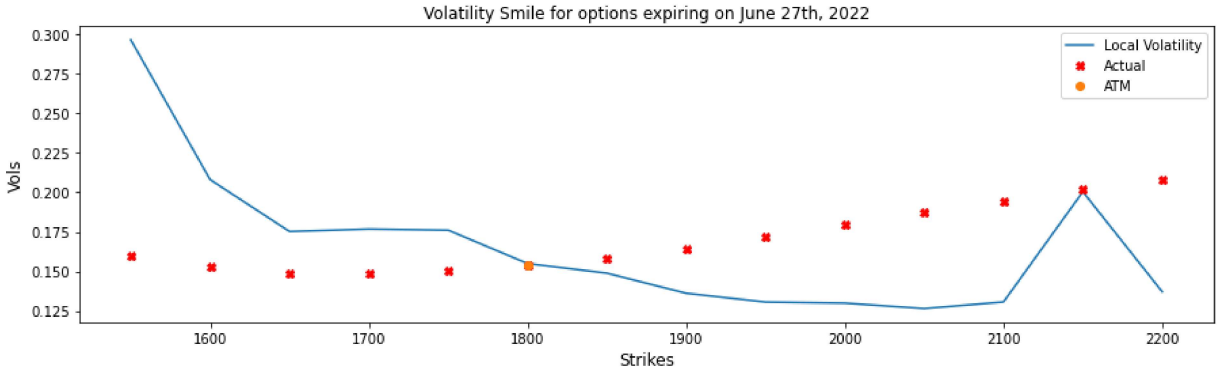
```
#DUPIRE LOCAL VOLATILITY SURFACE (NOT PLOTTABLE)

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)
```

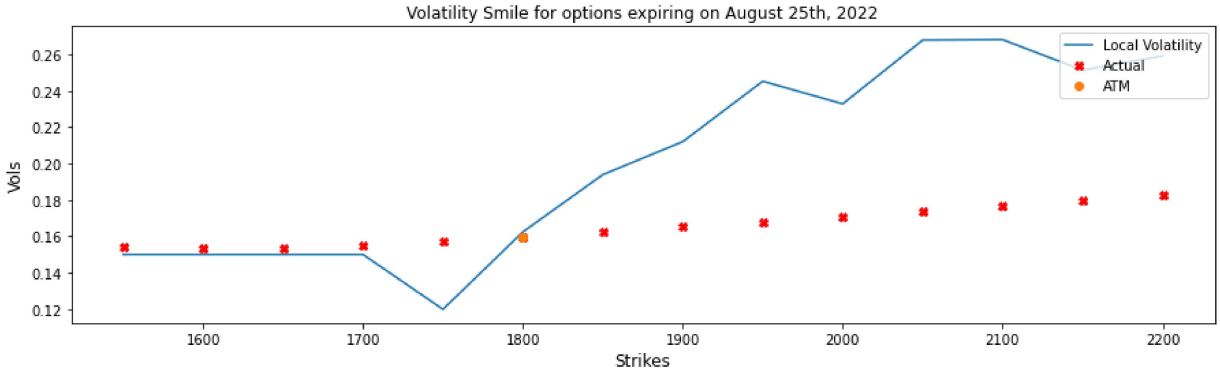












In [22]:

```
#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)


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

    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 (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 [23]:

```
# 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	1550.000000	19.492403	19.375897	-0.597696
1	1600.000000	28.798716	28.934788	0.472492
2	1650.000000	41.573900	41.866857	0.704666
3	1700.000000	58.585336	58.672316	0.148467
4	1750.000000	79.768612	79.623471	-0.181952
5	1800.000000	105.019940	104.708932	-0.296141
6	1850.000000	103.874121	103.527089	-0.334090
7	1900.000000	85.994838	85.715468	-0.324869
8	1950.000000	71.103345	70.905193	-0.278682
9	2000.000000	58.676208	58.642578	-0.057314
10	2050.000000	48.535785	48.515652	-0.041479
11	2100.000000	40.145960	40.164975	0.047363
12	2150.000000	33.216956	33.284023	0.201907
13	2200.000000	27.591312	27.615042	0.086006

Heston Model 1, Gold parameters output

	Value
v0	0.252117
kappa	0.000000
theta	0.172547
sigma	0.334091
rho	0.027691
avgError	0.269509

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

Heston Model 2, Gold calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	1550.000000	19.492403	83.867143	330.255541
1	1600.000000	28.798716	97.568235	238.793698
2	1650.000000	41.573900	113.217348	172.327946
3	1700.000000	58.585336	131.037464	123.669391
4	1750.000000	79.768612	151.243329	89.602559
5	1800.000000	105.019940	174.019636	65.701519

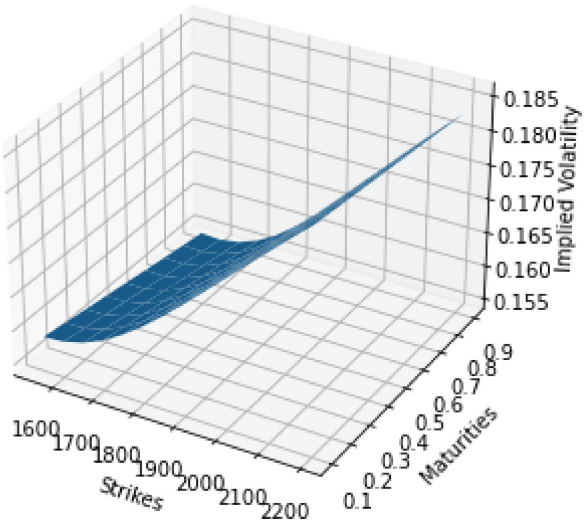
	Strikes	Market Value	Model Value	Relative Error (%)
6	1850.000000	103.874121	169.351989	63.035785
7	1900.000000	85.994838	147.334642	71.329634
8	1950.000000	71.103345	128.013438	80.038560
9	2000.000000	58.676208	111.246076	89.593158
10	2050.000000	48.535785	96.817246	99.476008
11	2100.000000	40.145960	84.468619	110.403782
12	2150.000000	33.216956	73.928840	122.563561
13	2200.000000	27.591312	64.936544	135.351417

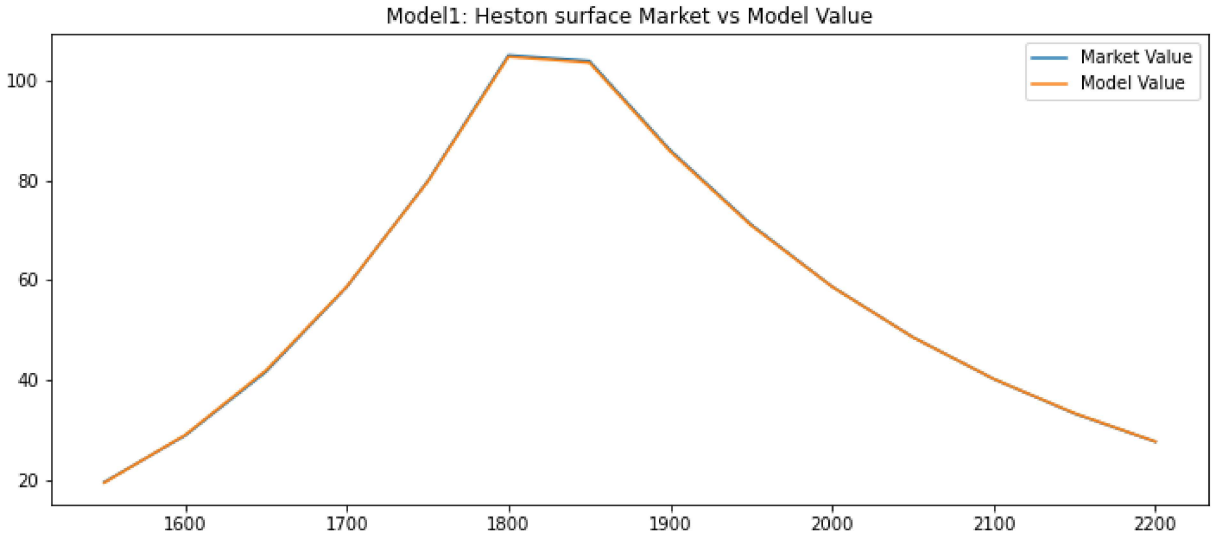
Heston Model 2, Gold parameters output

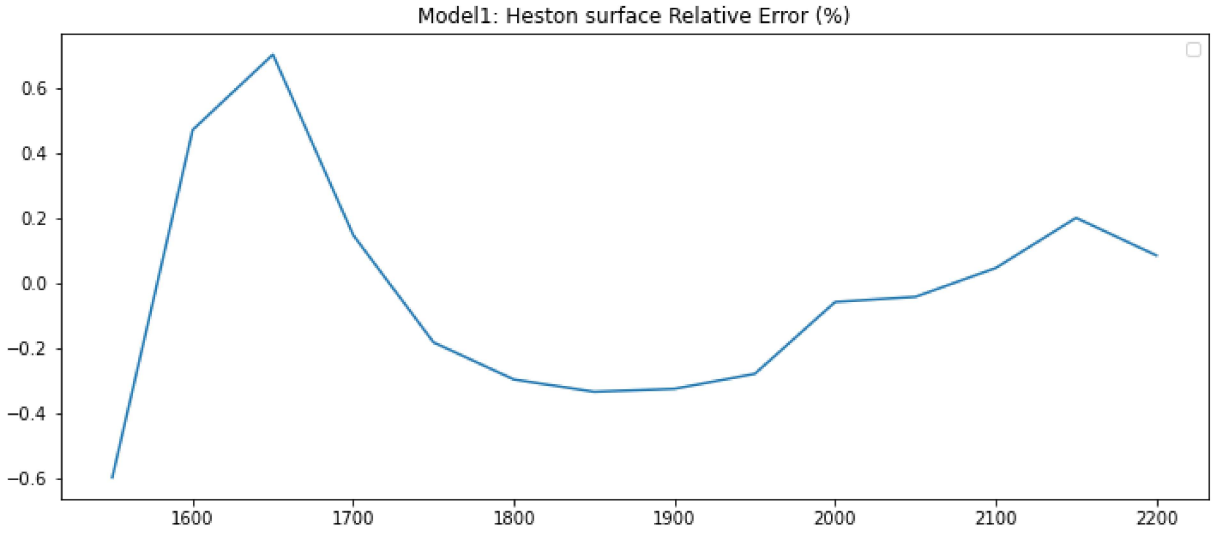
	Value
v0	0.461423
kappa	0.453074
theta	0.986932
sigma	-0.282696
rho	0.000000
avgError	128.010183

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

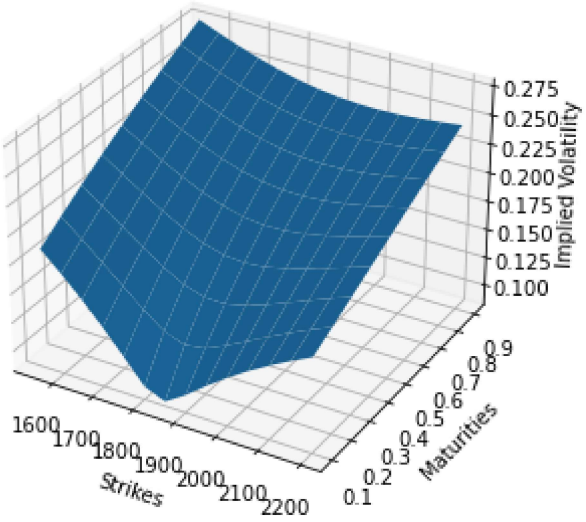
Heston Volatility Surface for Heston Model 1, Gold

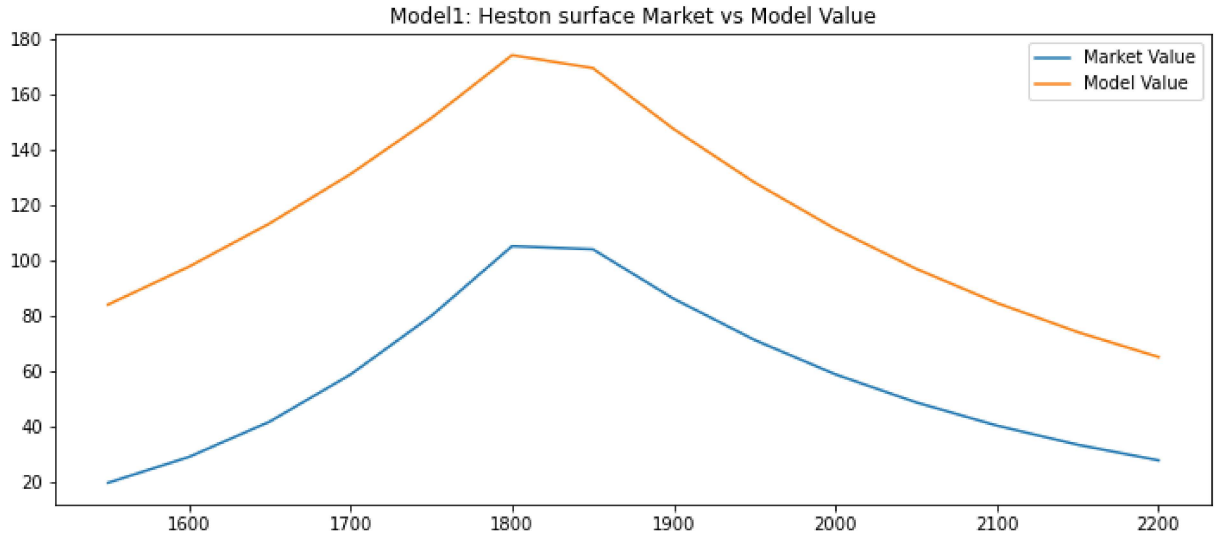


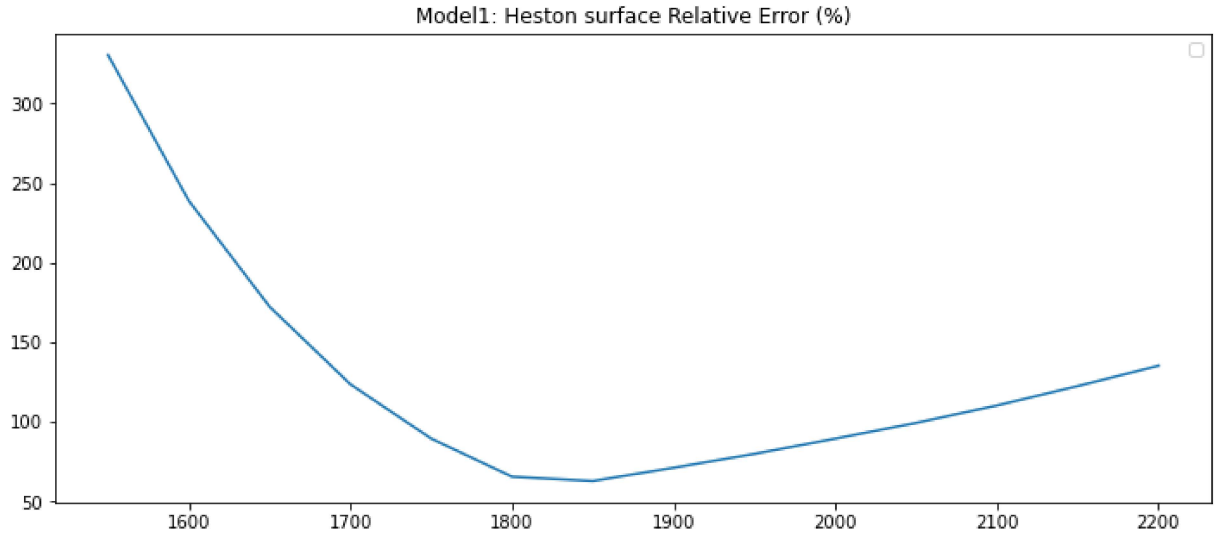




Heston Volatility Surface for Heston Model 2, Gold

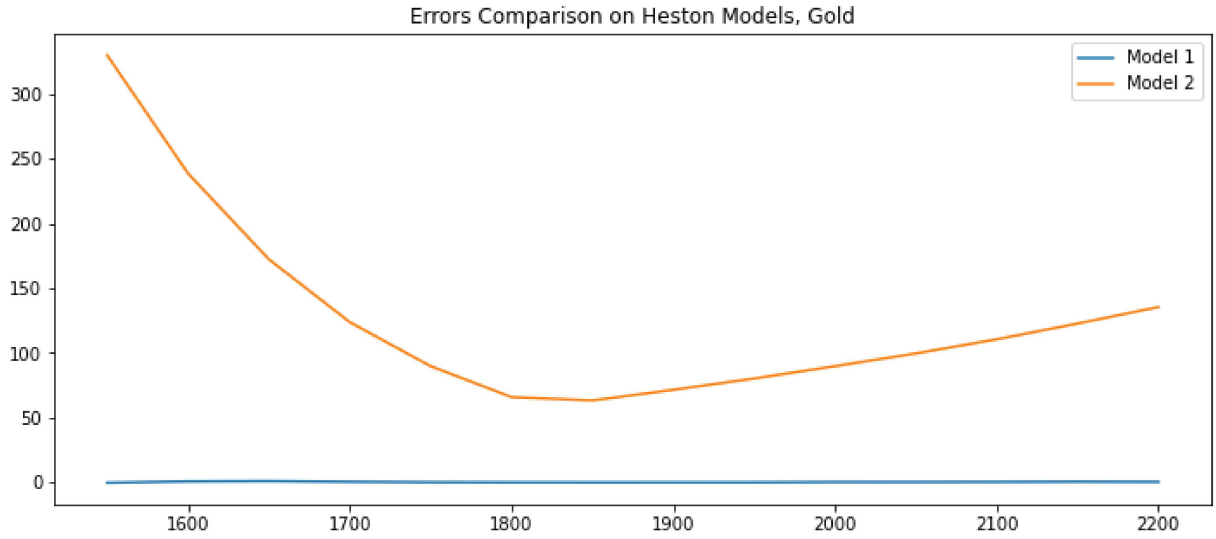


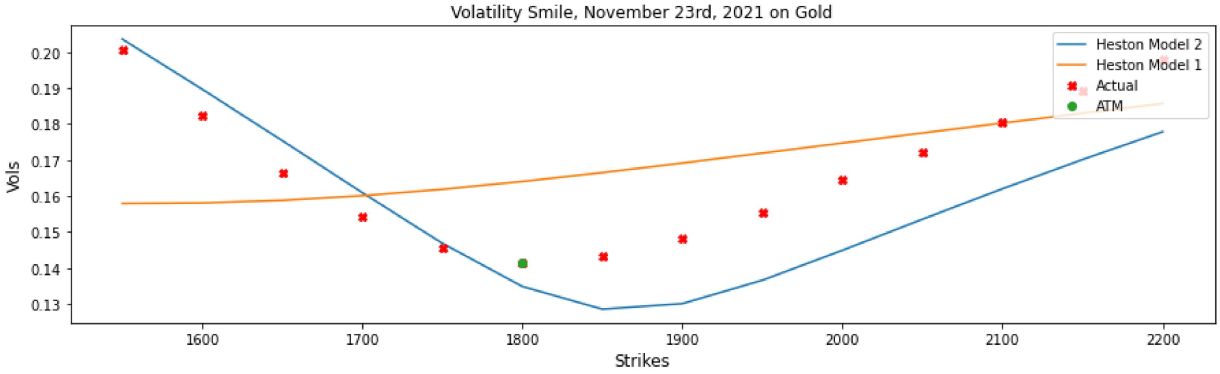


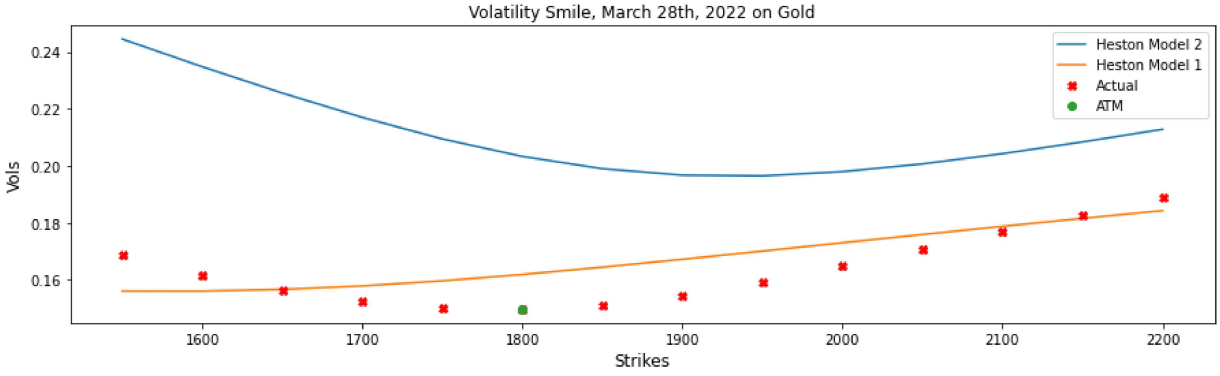


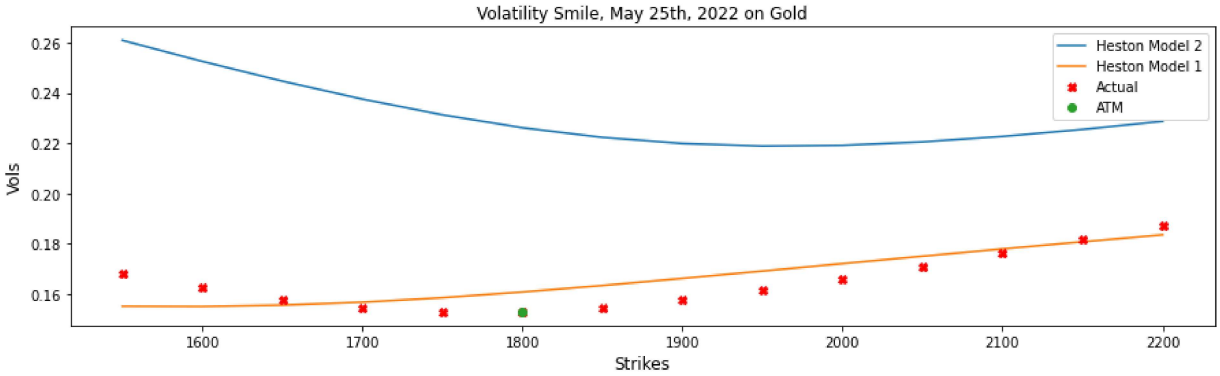
In [24]:

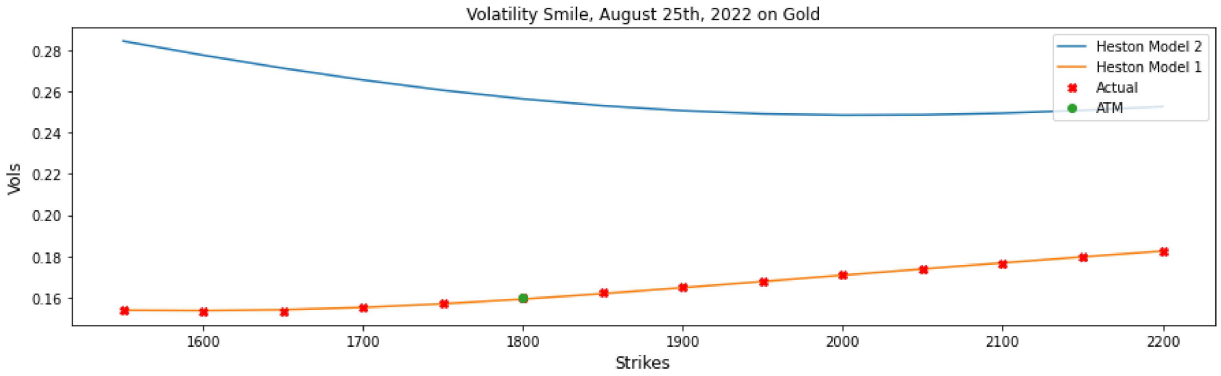
```
# Relative error comparison
plt.figure(figsize=plot_size)
plt.plot(hestonModel1.strks, hestonModel1.relativeError, label="Model 1")
plt.plot(hestonModel2.strks, hestonModel2.relativeError, label="Model 2")
plt.title("Errors Comparison on Heston Models, {}".format(data_label));
plt.legend()

# 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
300
250
200
150
100
50
0
1600
1700
1800
1900
2000
2100
2200


Volatility Smile, November 23rd, 2021 on Gold
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes
0.20
0.19
0.18
0.17
0.16
0.15
0.14
0.13
1600
1700
1800
1900
2000
2100
2200


Volatility Smile, March 28th, 2022 on Gold
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes
0.24
0.22
0.20
0.18
0.16
1600
1700
1800
1900
2000
2100
2200


Volatility Smile, May 25th, 2022 on Gold
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes
0.26
0.24
0.22
0.20
0.18
0.16
1600
1700
1800
1900
2000
2100
2200



In [25]:

```
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 == "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

        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), l

        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
            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,
    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')
```

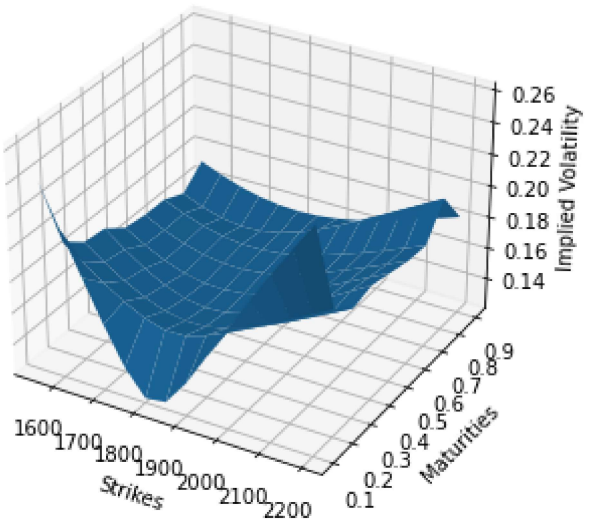
```
        axs[1, 1].legend()

        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 [26]:

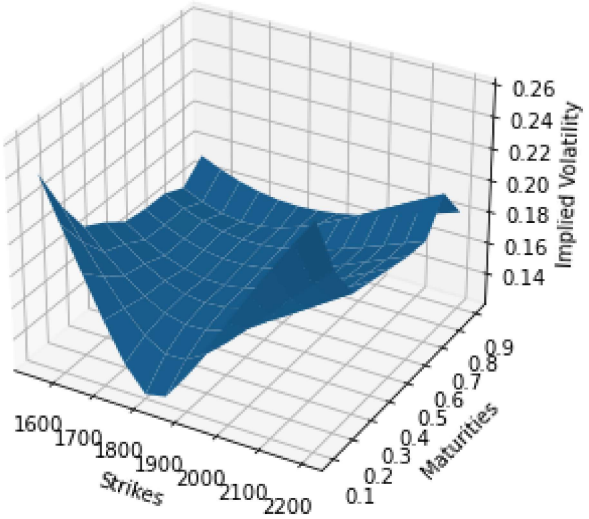
```
# SABR Volatility model

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=
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: 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 "
```

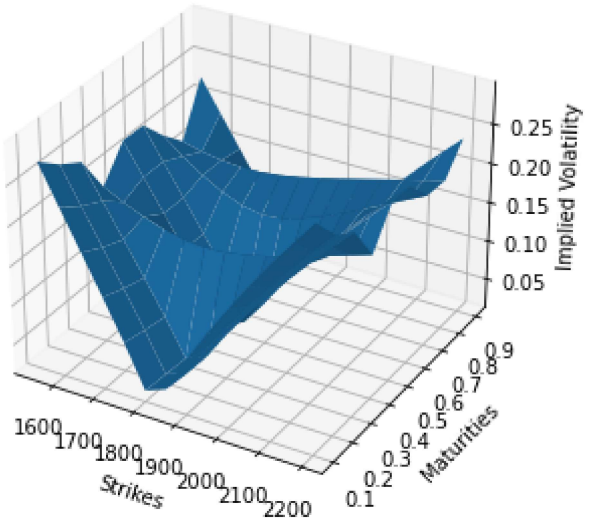
SABR, beta=1, Gold

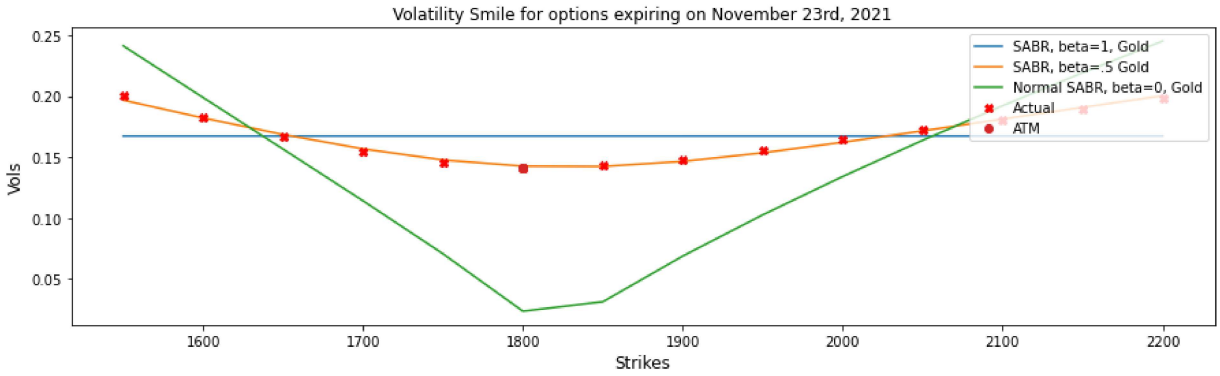


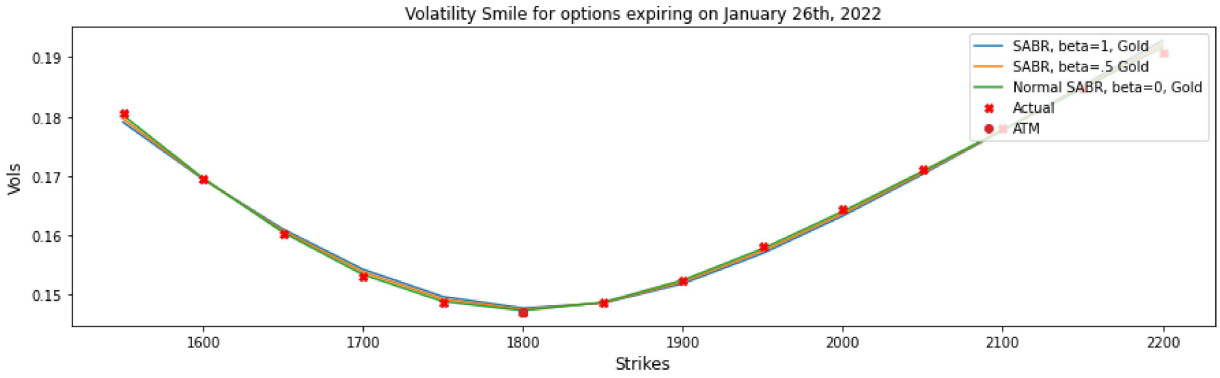
SABR, beta=.5 Gold

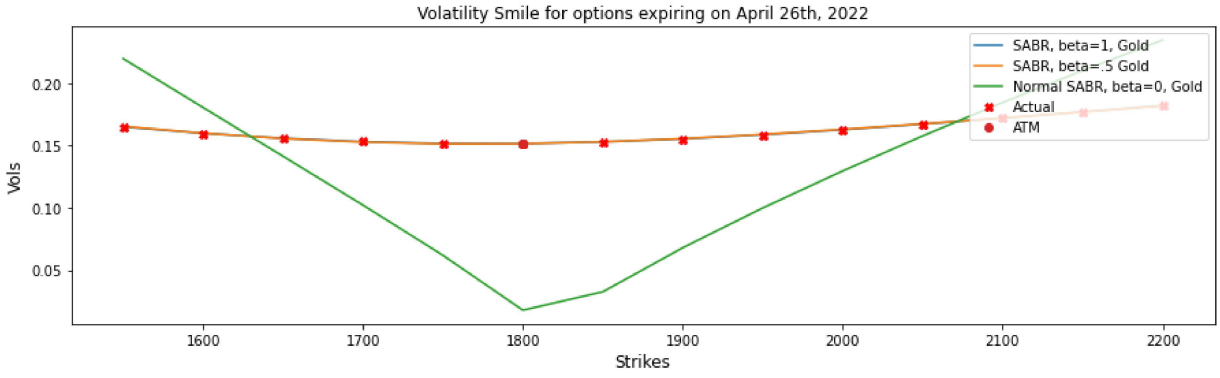


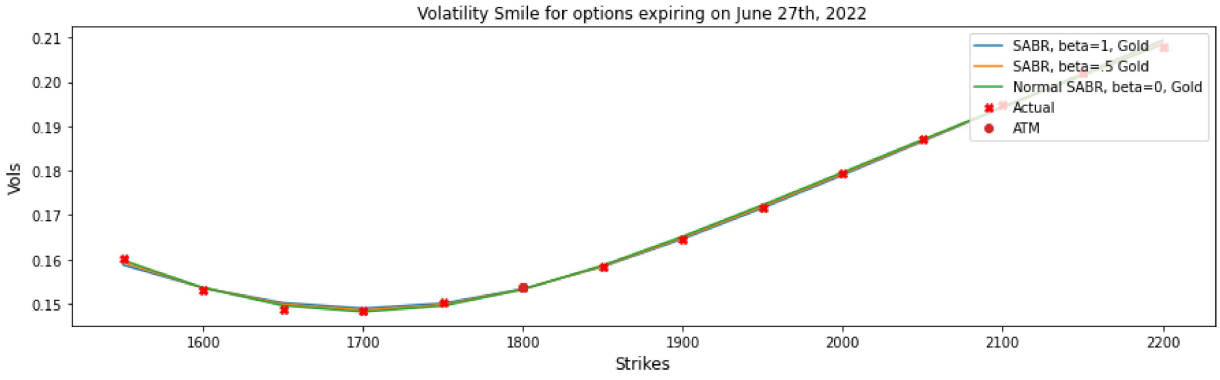
Normal SABR, beta=0, Gold

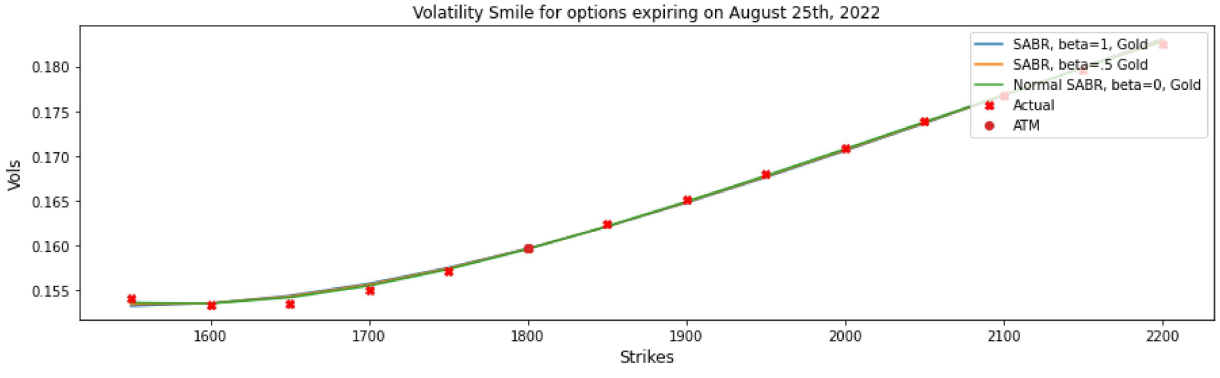












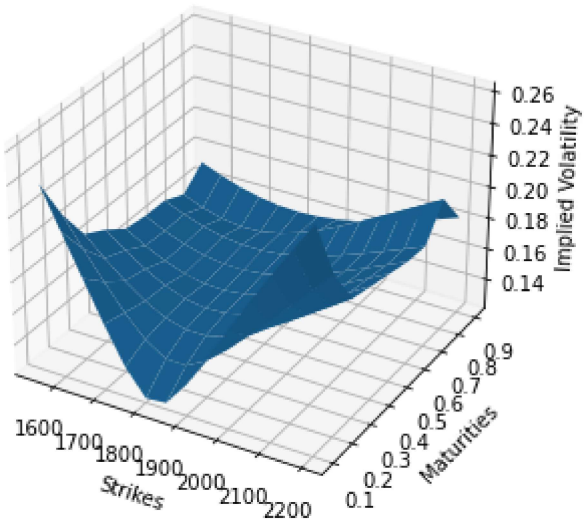
In [27]:

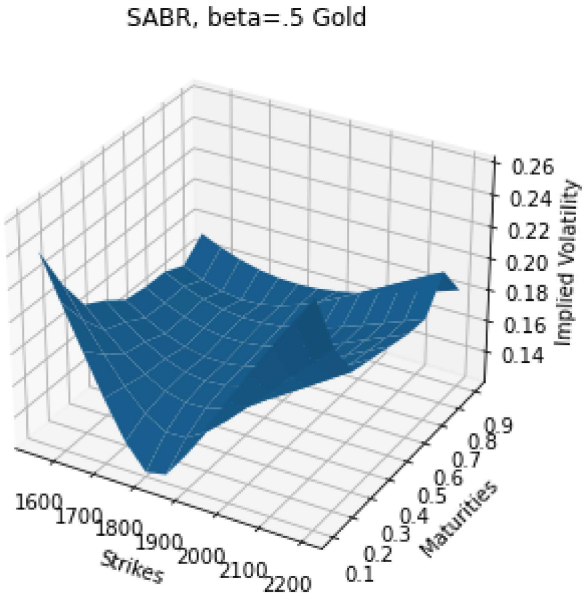
```
# Shifted SABR Volatility model
```

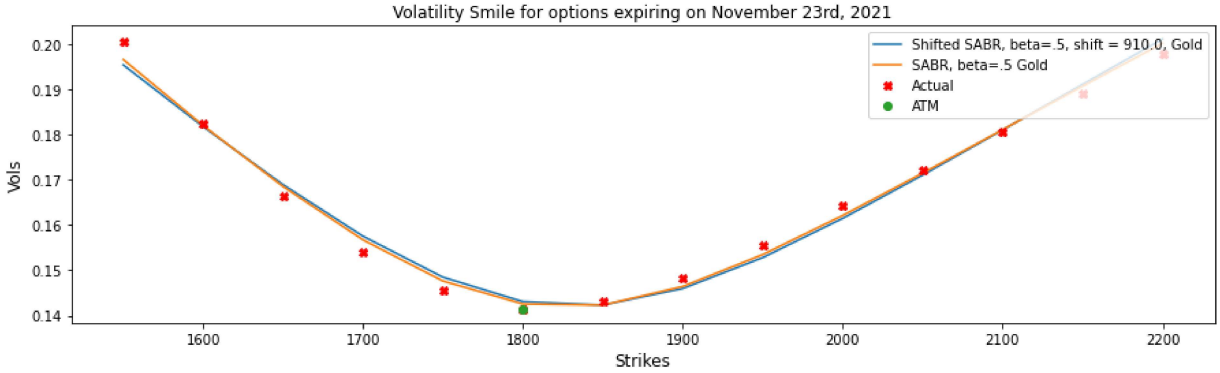
```
shft = .50 * current_price
shiftedSABR_beta1 = SABRVolatilitySurface(beta=1, shift=shft,strks=strikes, label="S
shiftedSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=shft, strks=strikes,label="
shiftedSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=shft, strks=strikes,label="

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

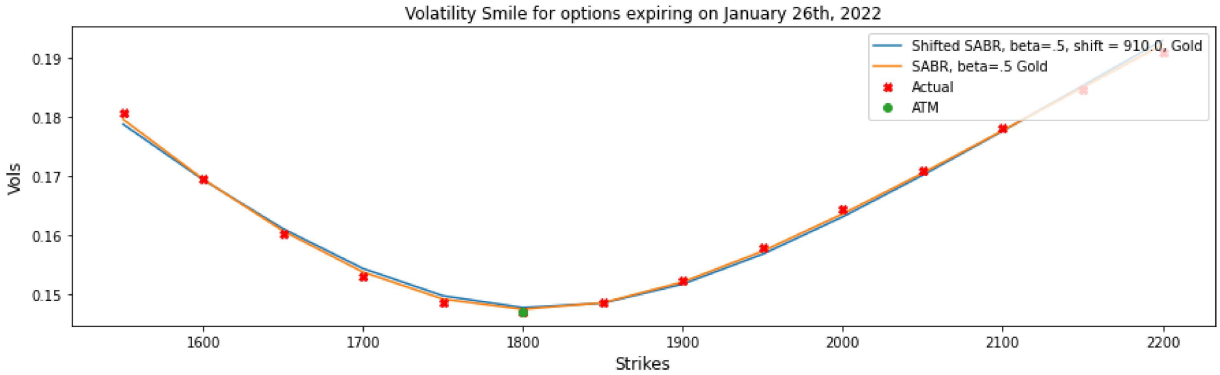
```
C:\ProgramData\Anaconda3\lib\site-packages\scipy\optimize\optimize.py:282: RuntimeWa
rning: Values in x were outside bounds during a minimize step, clipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```

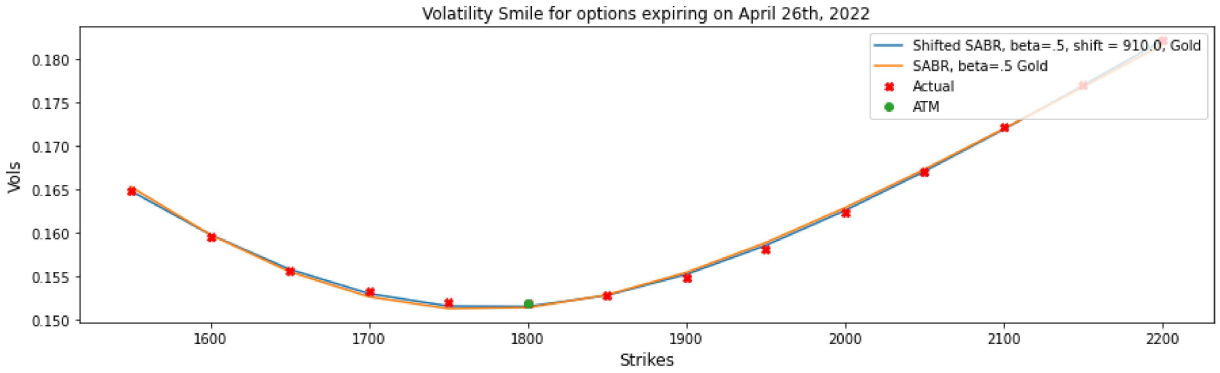
Shifted SABR, beta=.5, shift = 910.0, Gold

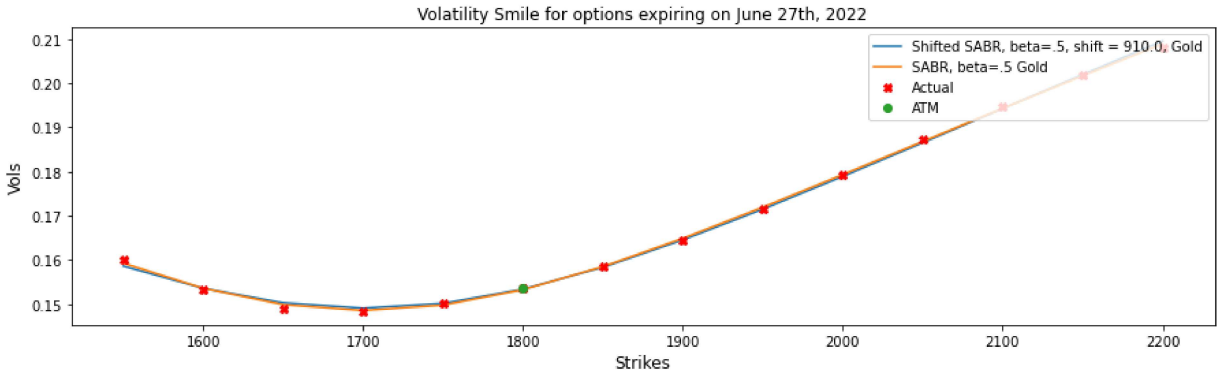


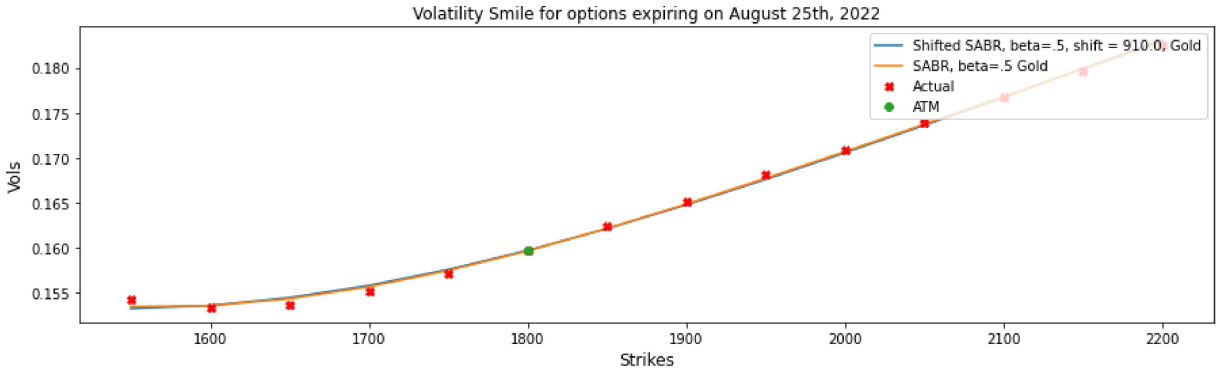

SABR, beta=.5 Gold
Implied Volatility
0.26
0.24
0.22
0.20
0.18
0.16
0.14
Strikes
1600
1700
1800
1900
2000
2100
2200
Maturities
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9

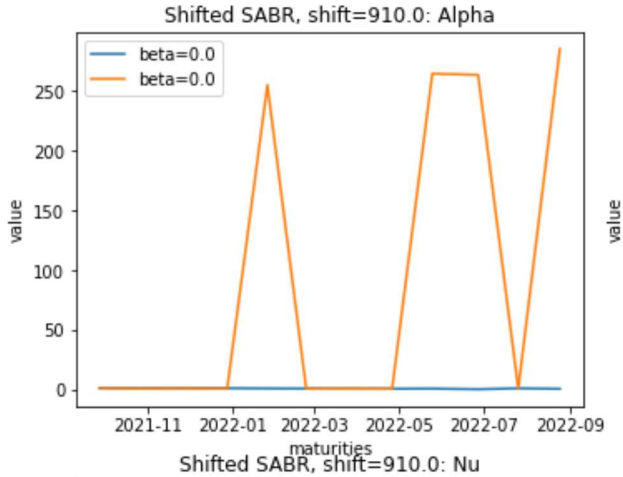

Volatility Smile for options expiring on November 23rd, 2021
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

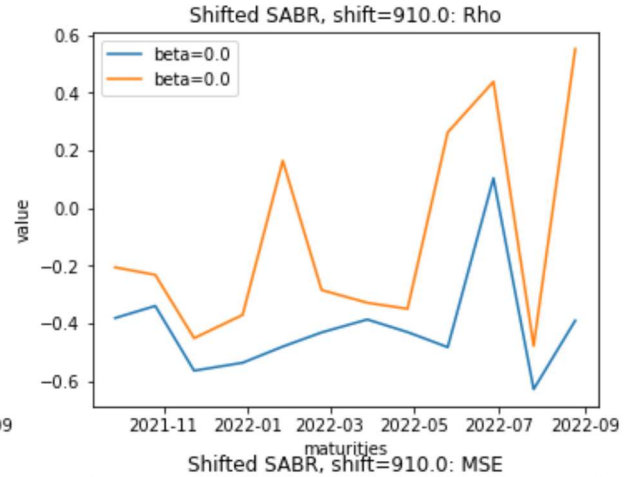

Volatility Smile for options expiring on January 26th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

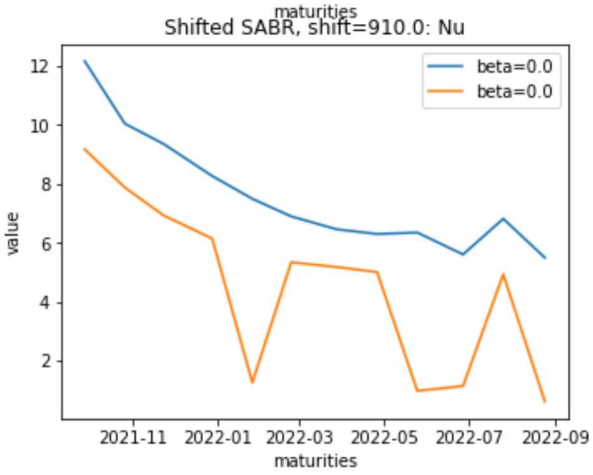

Volatility Smile for options expiring on April 26th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes

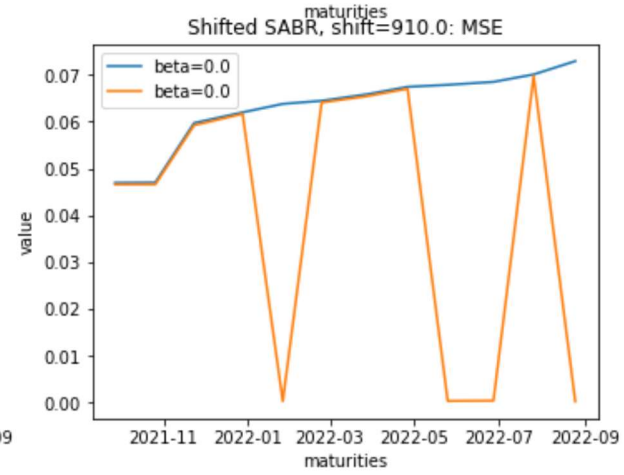

Volatility Smile for options expiring on June 27th, 2022
Shifted SABR, beta=.5, shift = 910.0, Gold
SABR, beta=.5 Gold
Actual
ATM
Vols
Strikes


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

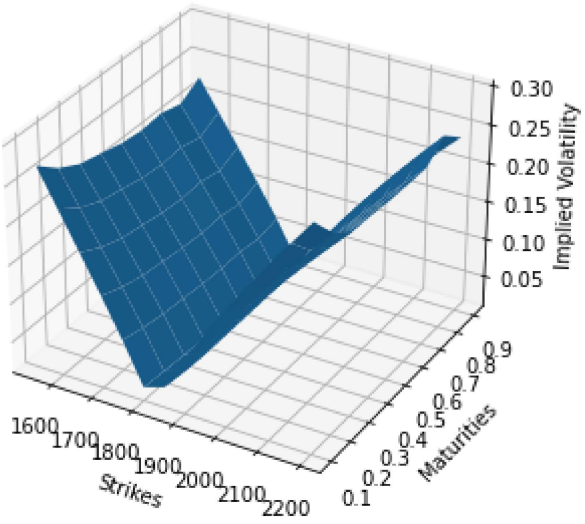

Shifted SABR, shift=910.0: Alpha
beta=0.0
beta=0.0
value
maturities


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

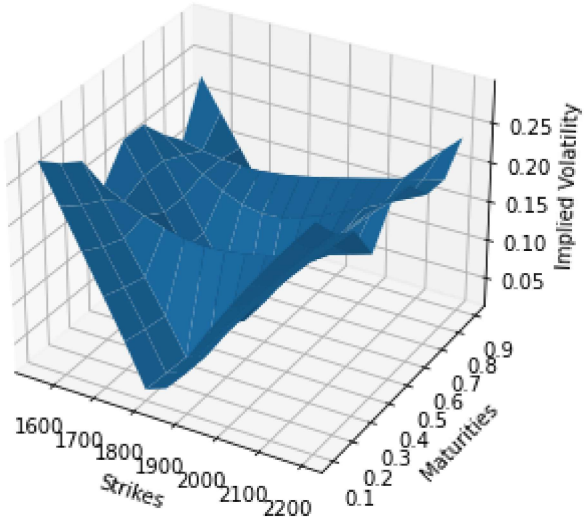

Shifted SABR, shift=910.0: Nu
beta=0.0
beta=0.0
value
maturities

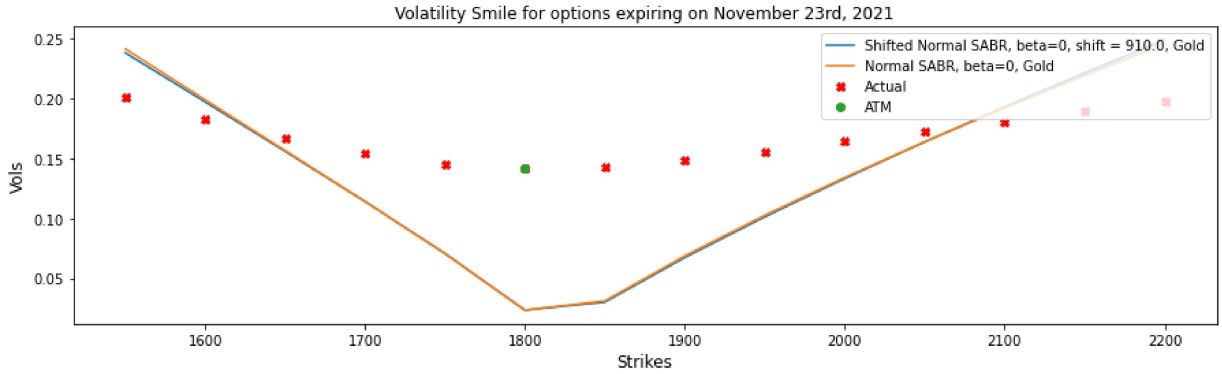

Shifted SABR, shift=910.0: MSE
beta=0.0
beta=0.0
value
maturities

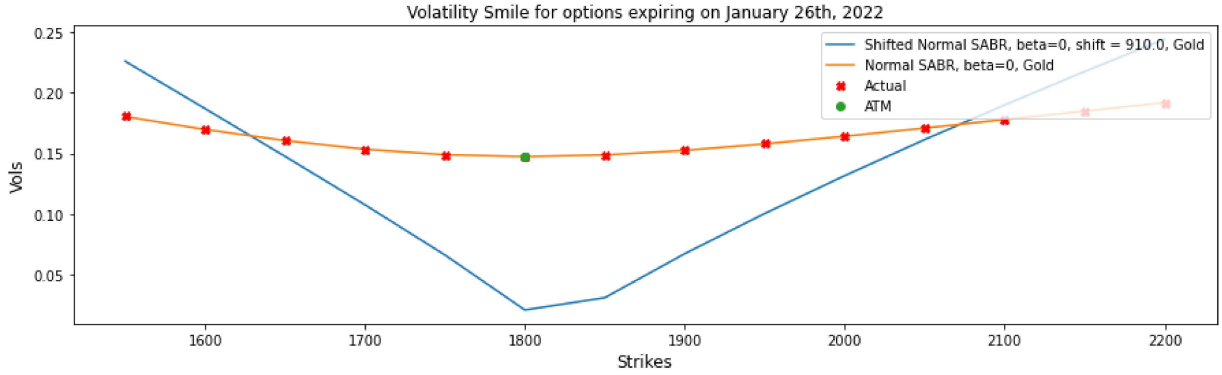
Shifted Normal SABR, beta=0, shift = 910.0, Gold

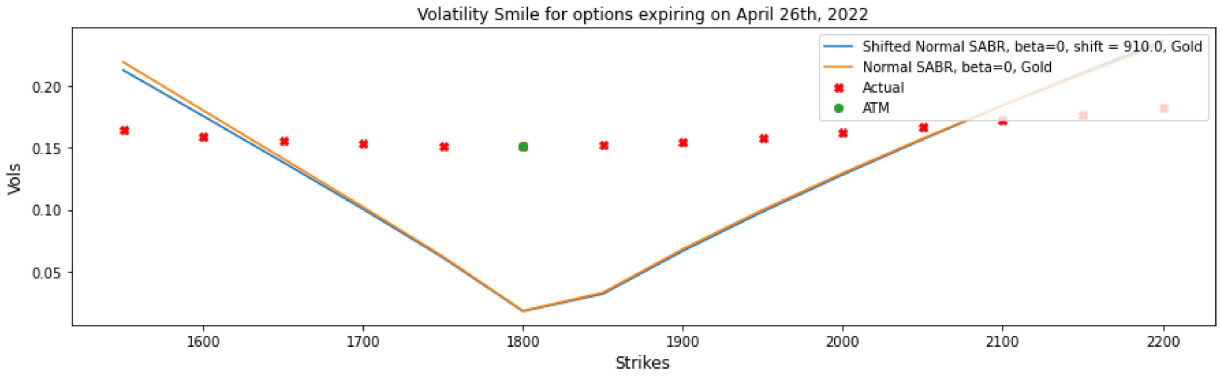


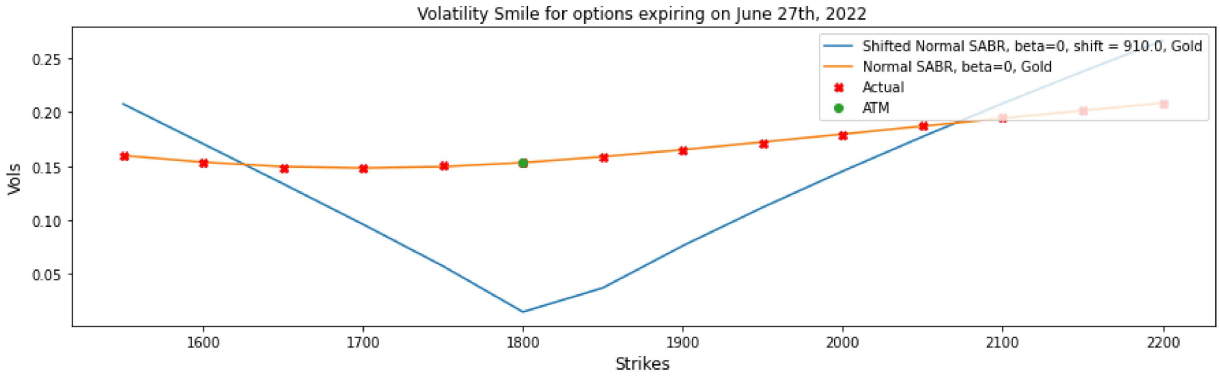
Normal SABR, beta=0, Gold

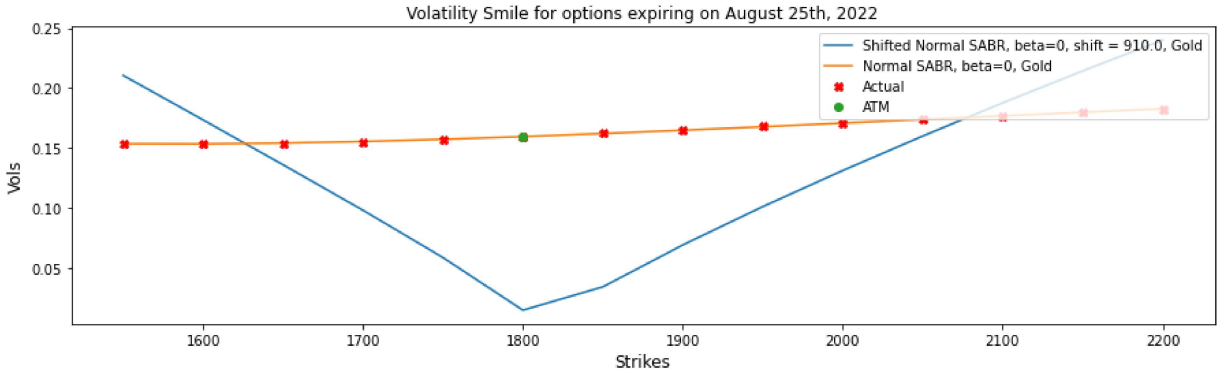






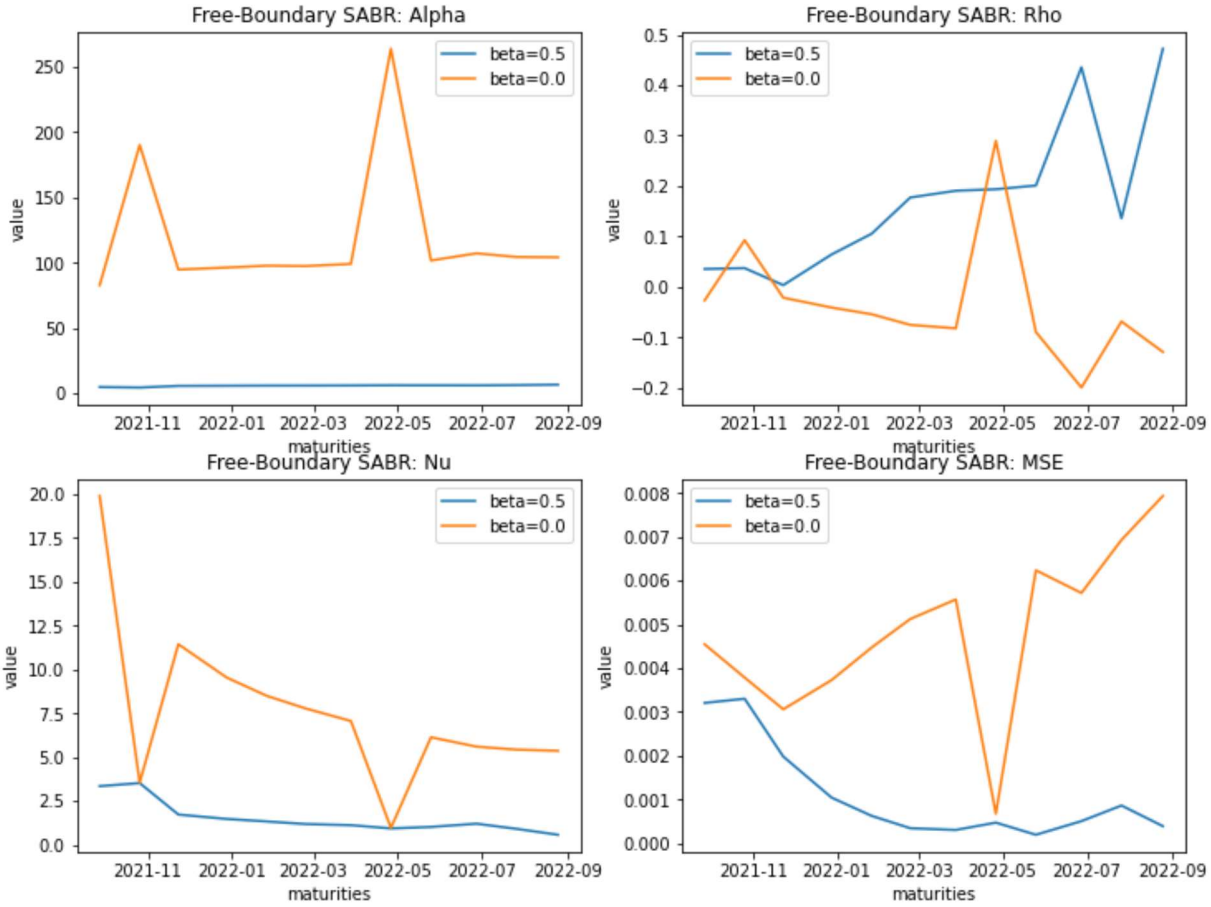




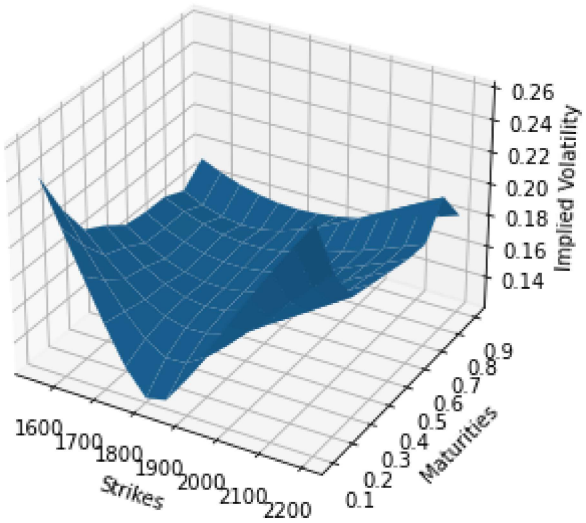


In [28]:

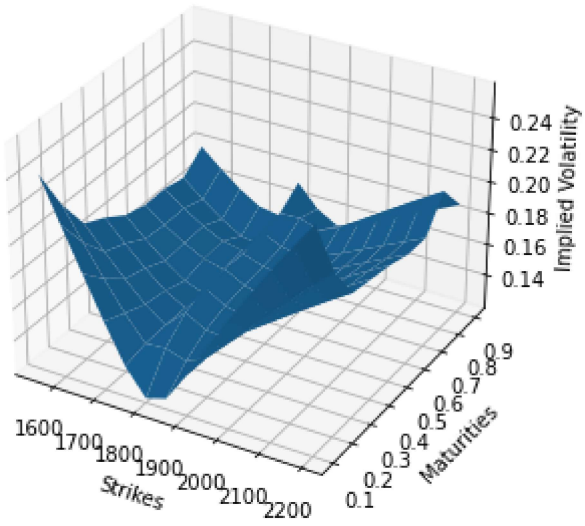
```
# Free-Boundary SABR Volatility model

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")
```

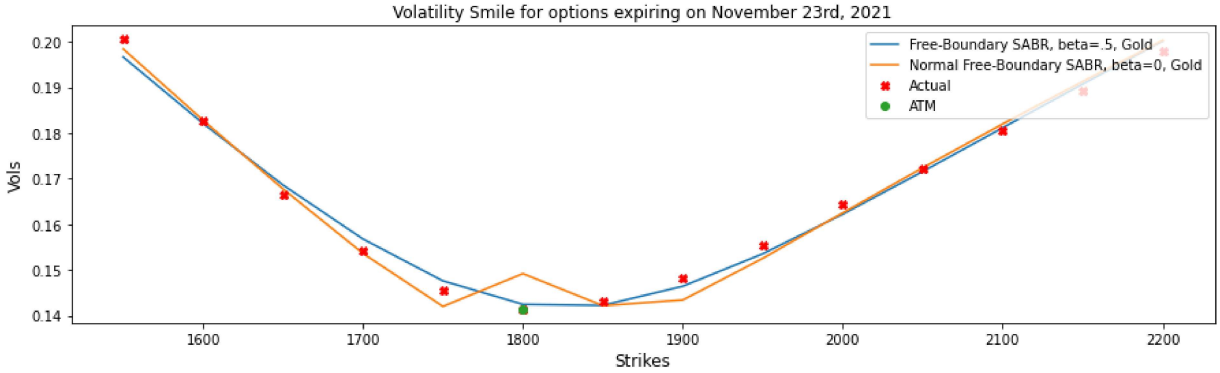


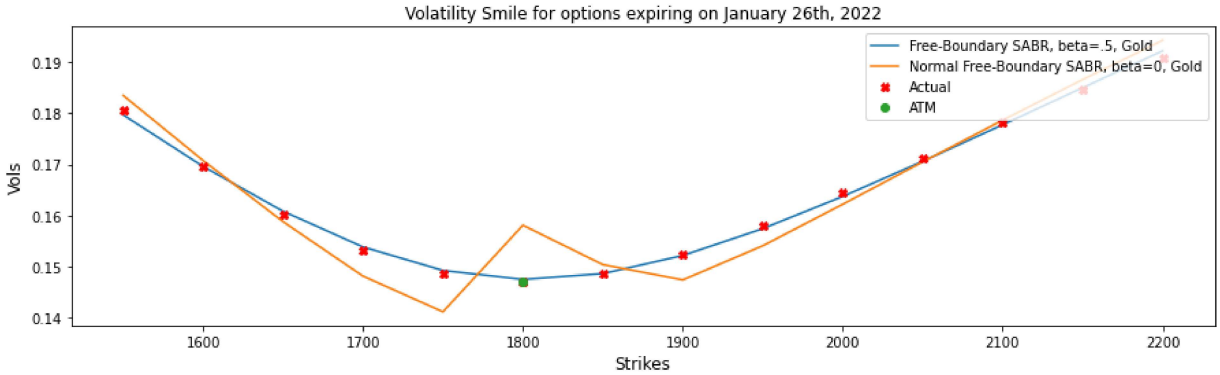
Free-Boundary SABR, beta=.5, Gold

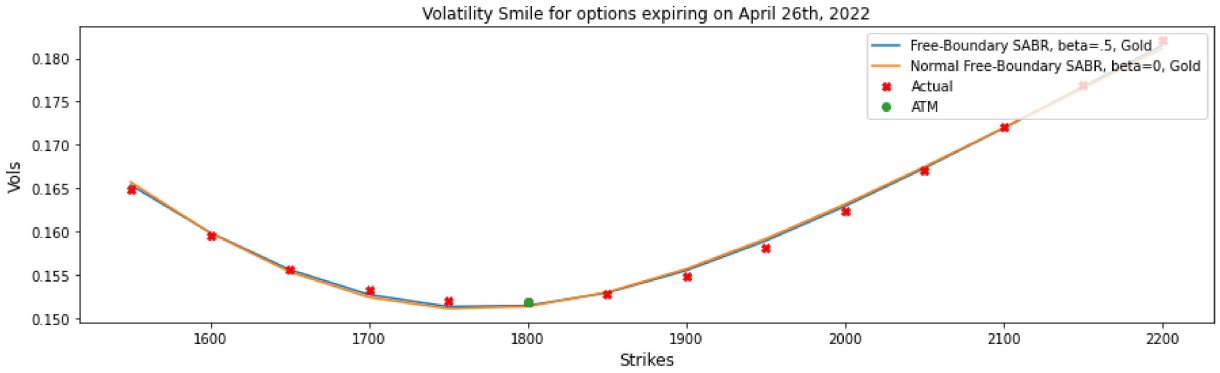


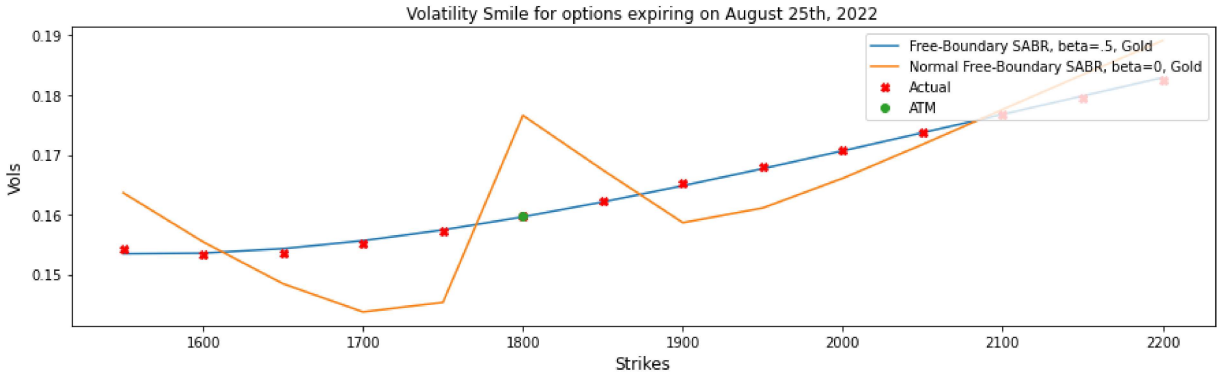
Normal Free-Boundary SABR, beta=0, Gold











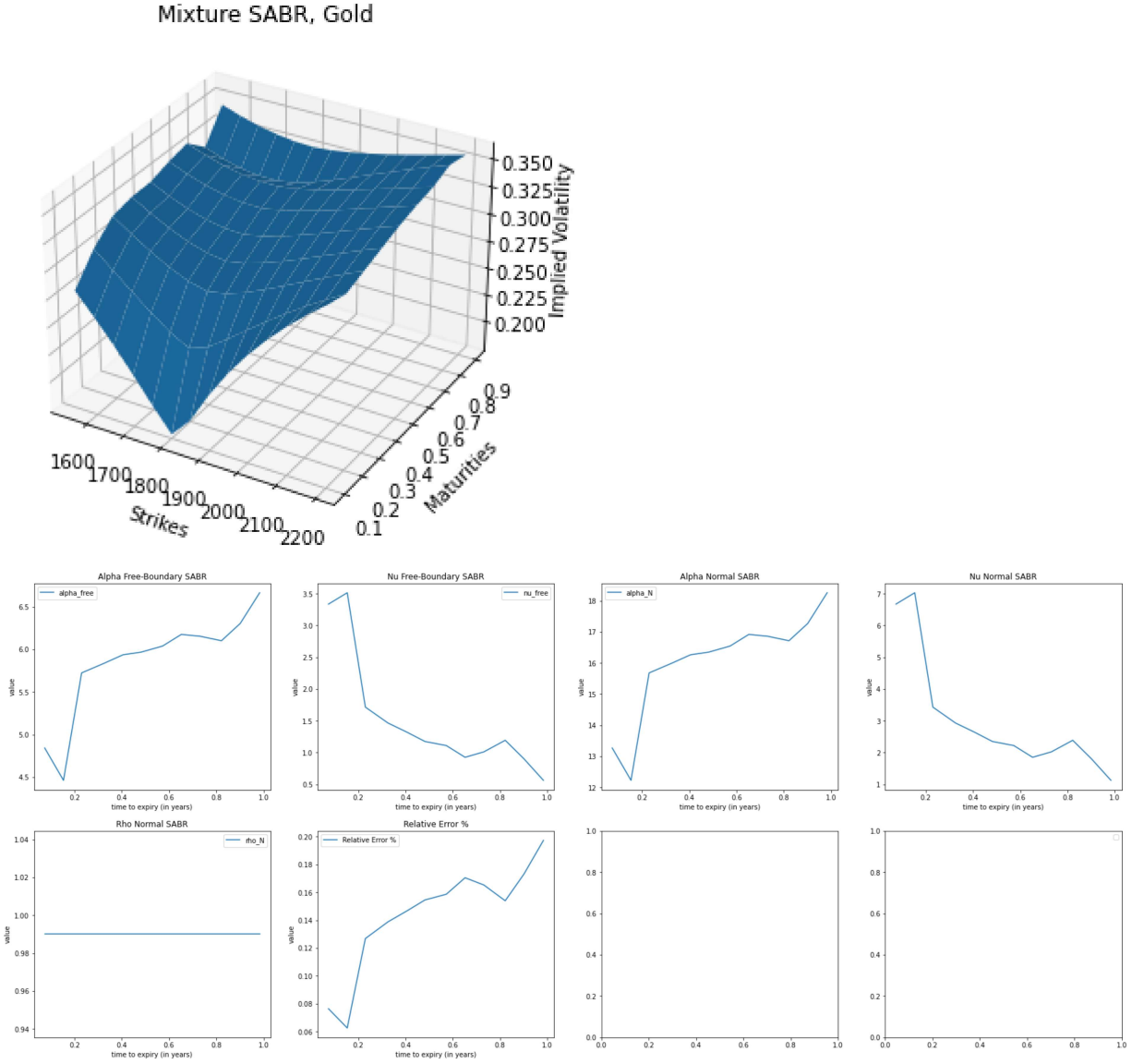
In [29]:

```
# 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	MSE
September 27th, 2021	4.842674	0.5	3.336643	0.0	4.842674	0.0	6.673287	0.99	0.076402
October 26th, 2021	4.462455	0.5	3.515373	0.0	4.462455	0.0	7.030745	0.99	0.062545
November 23rd, 2021	5.724018	0.5	1.715591	0.0	5.724018	0.0	3.431182	0.99	0.126814
December 28th, 2021	5.837908	0.5	1.464982	0.0	5.837908	0.0	2.929965	0.99	0.138828
January 26th, 2022	5.937087	0.5	1.322429	0.0	5.937087	0.0	2.644858	0.99	0.146612
February 23rd, 2022	5.967954	0.5	1.174825	0.0	5.967954	0.0	2.349650	0.99	0.154522
March 28th, 2022	6.039150	0.5	1.110556	0.0	6.039150	0.0	2.221112	0.99	0.158635
April 26th, 2022	6.176050	0.5	0.925985	0.0	6.176050	0.0	1.851970	0.99	0.170492
May 25th, 2022	6.154438	0.5	1.011414	0.0	6.154438	0.0	2.022828	0.99	0.165279
June 27th, 2022	6.101873	0.5	1.193507	0.0	6.101873	0.0	2.387014	0.99	0.153949
July 26th, 2022	6.304776	0.5	0.901238	0.0	6.304776	0.0	1.802476	0.99	0.173185

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSE
August 25th, 2022	6.665379	0.5	0.563103	0.0	6.665379	0.0	1.126207	0.99	0.197323

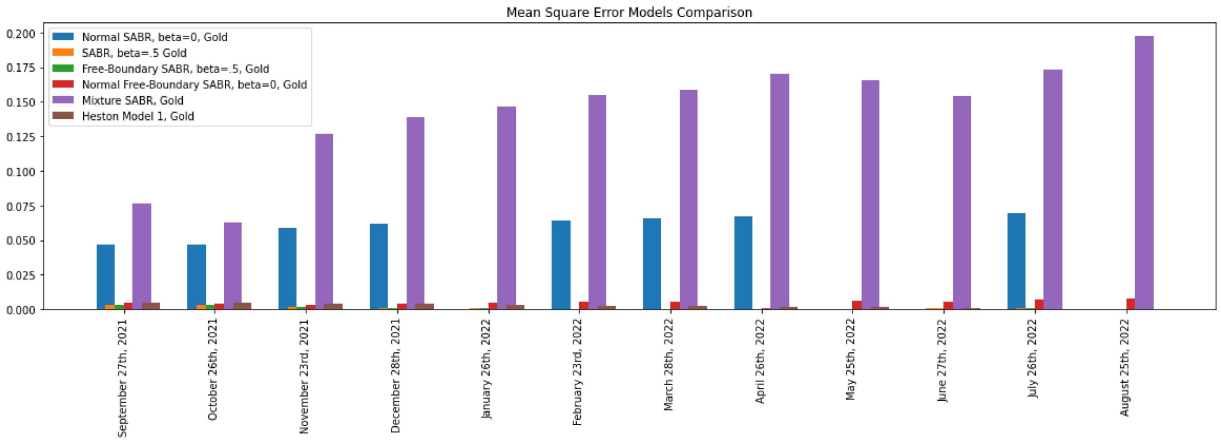
```
No handles with labels found to put in legend.
```



In [30]:

```
# 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()
```

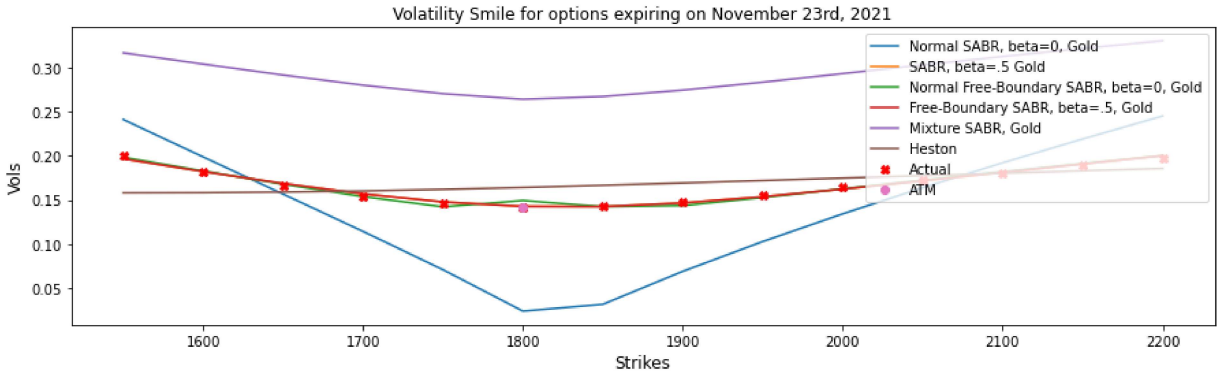
Out[30]: <matplotlib.legend.Legend at 0x1e4c8ff0b50>

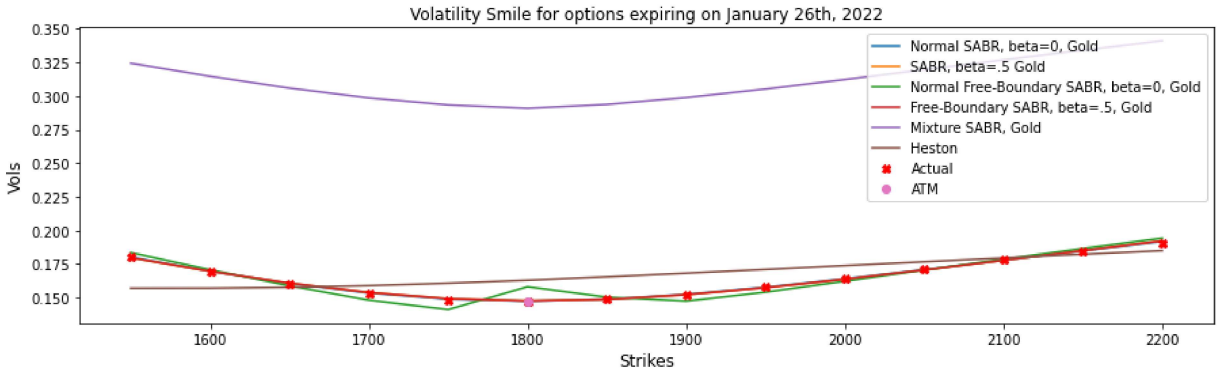


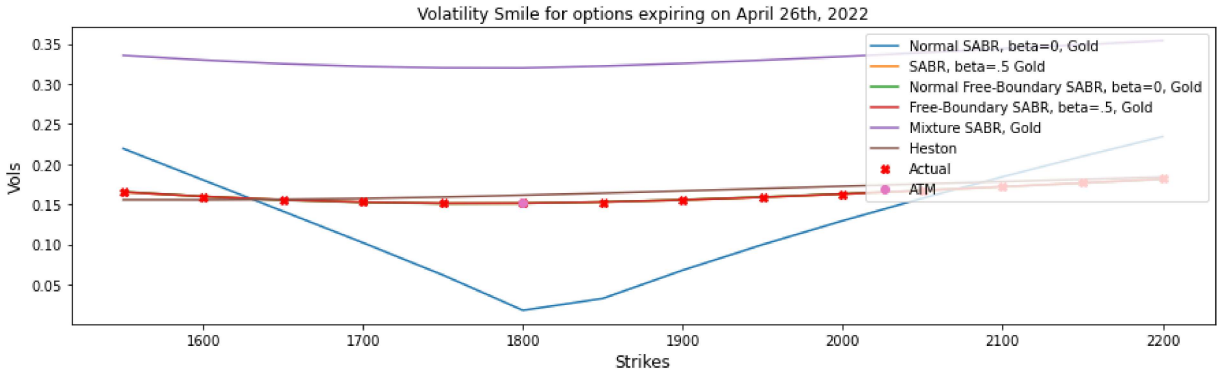
In [31]:

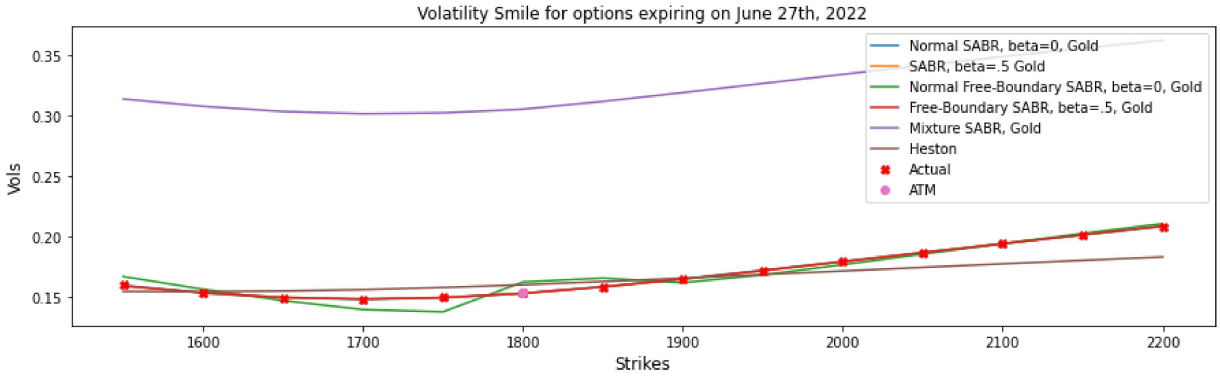
```
# Volatility Smiles Comparisons (Final)

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])
```

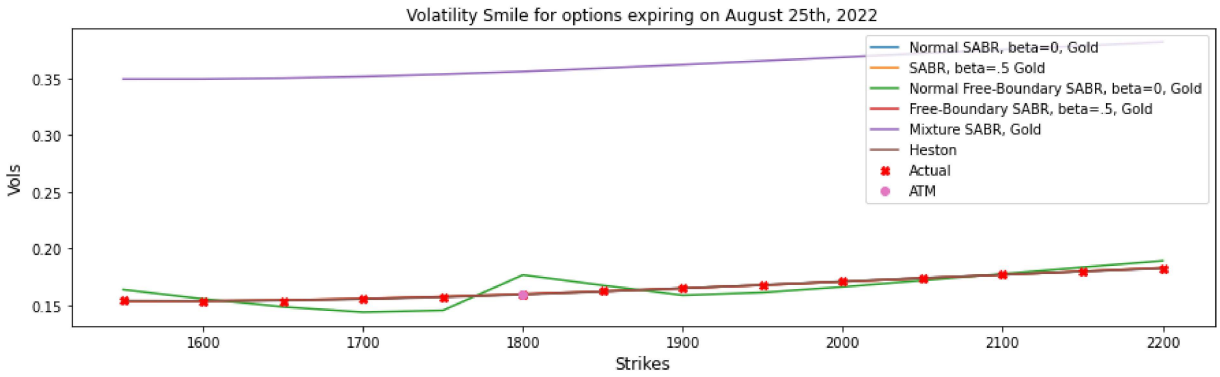
	Error
Normal SABR, beta=0, Gold	0.040140
SABR, beta=.5 Gold	0.001084
Normal Free-Boundary SABR, beta=0, Gold	0.004813
Free-Boundary SABR, beta=.5, Gold	0.001103
Mixture SABR, Gold	0.143716







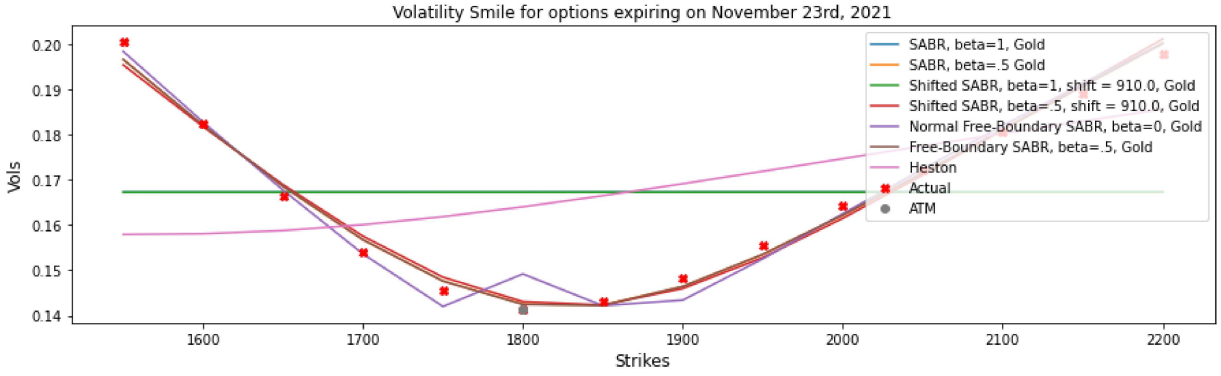


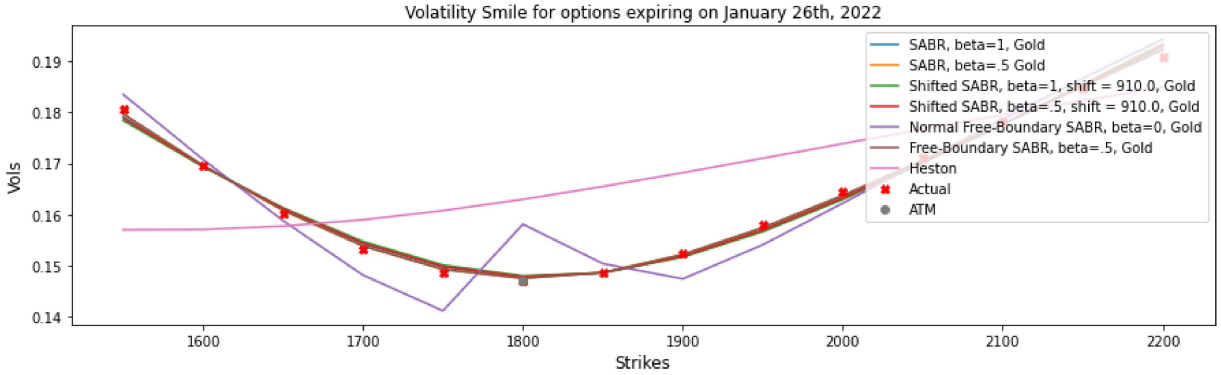


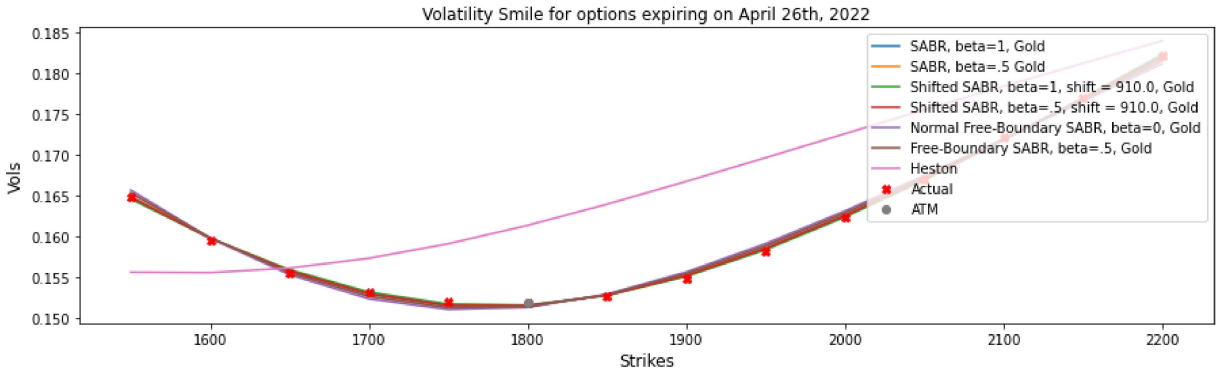
In [32]:

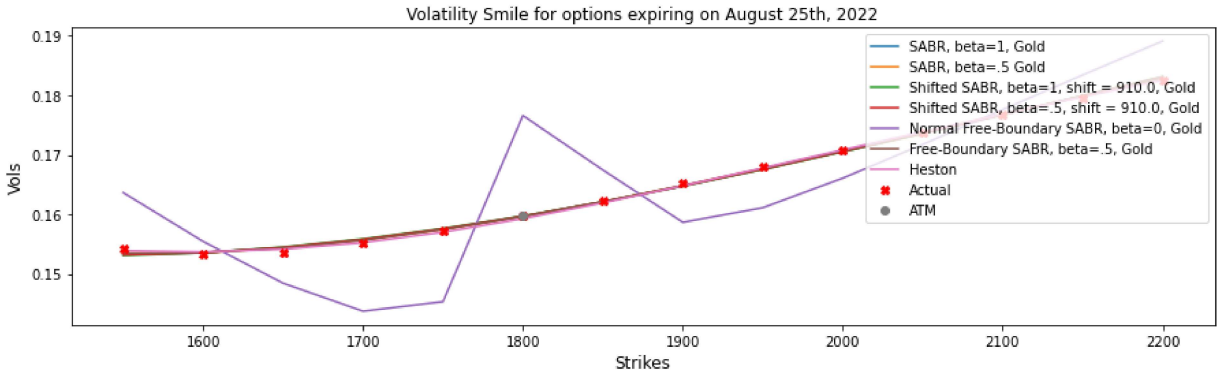
```
# Volatility Smiles Comparisons 2 (Final)

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	0.002516
SABR, beta=.5 Gold	0.001084
Shifted SABR, beta=1, shift = 910.0, Gold	0.002566
Shifted SABR, beta=.5, shift = 910.0, Gold	0.001099
Normal Free-Boundary SABR, beta=0, Gold	0.004813
Free-Boundary SABR, beta=.5, Gold	0.001103



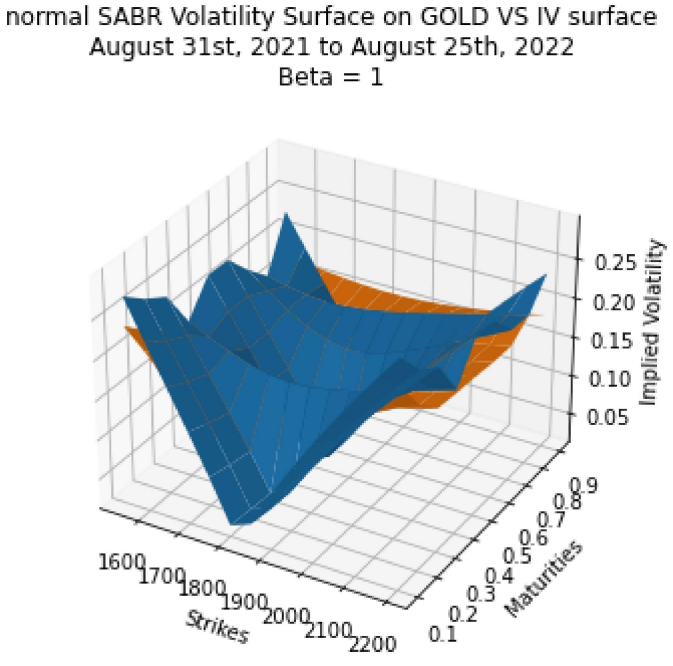




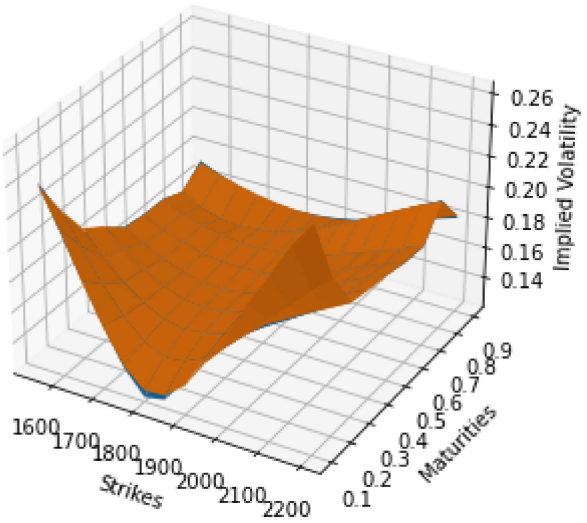


In [33]:

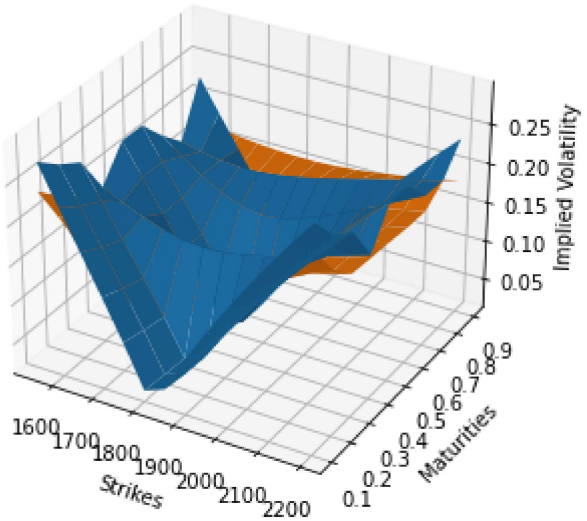
```
# Volatility Surfaces plots comparison

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)
```



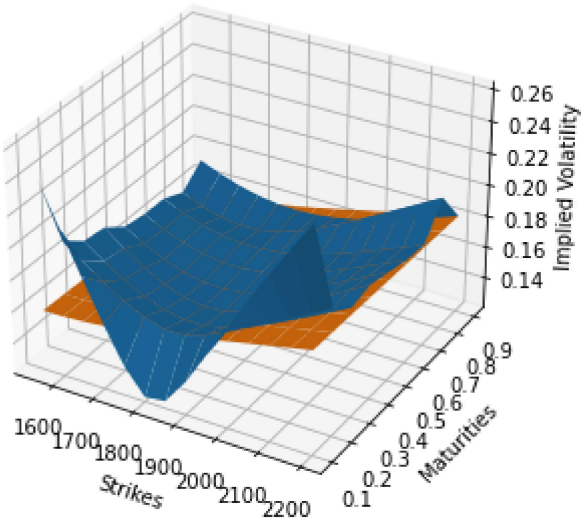
normal SABR Volatility Surface on GOLD VS IV surface
August 31st, 2021 to August 25th, 2022
Beta = 0.5

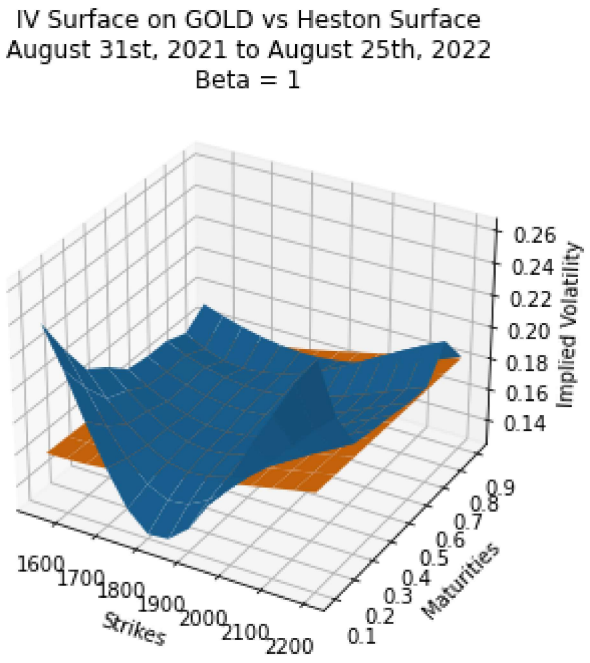


normal SABR Volatility Surface on GOLD VS IV surface
August 31st, 2021 to August 25th, 2022
Beta = 0



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

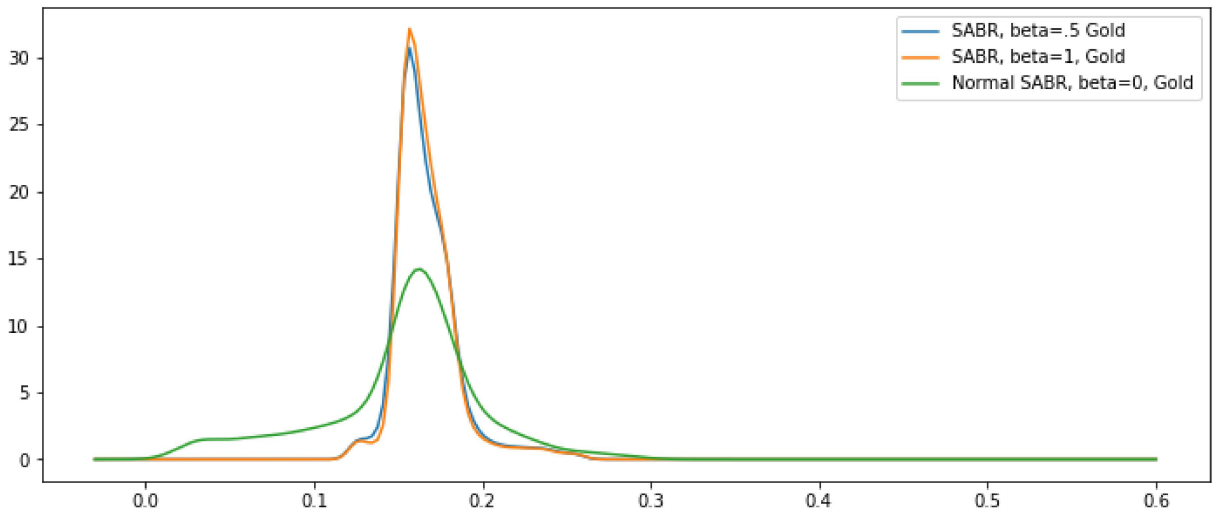




In [34]:

```
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,.6,200)
    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[34]: (None, None, None)

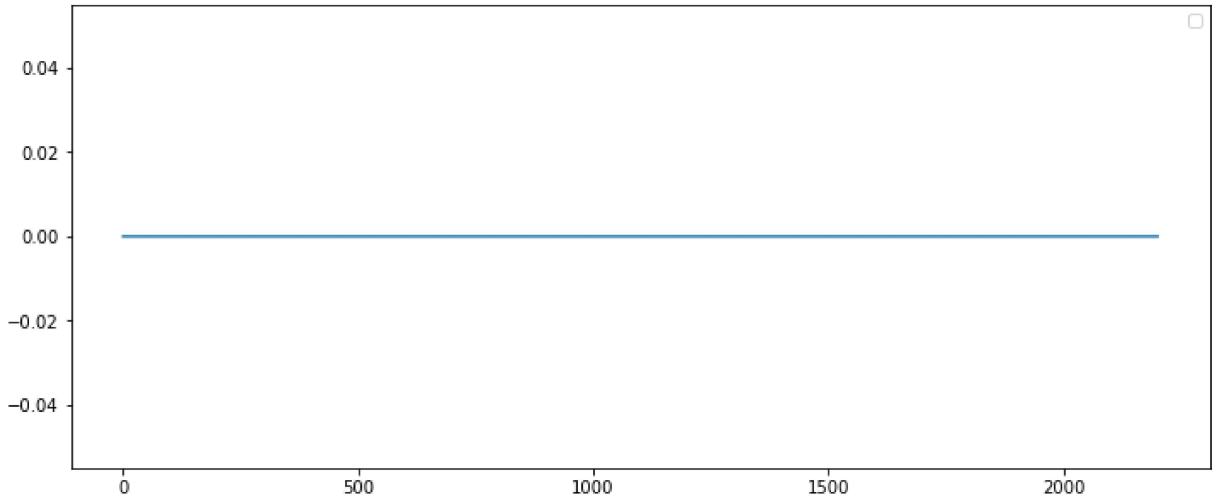


In [35]:

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

```
plt.figure(figsize=plot_size)
def distribution(model):
    rn = []
    t=1
    for i in np.arange(0, (model.vol_surface.maxDate()-today)/365, .1):
        for j in np.arange(model.vol_surface.minStrike(), model.vol_surface.maxStrik
            rn.append(model.vol_surface.blackVol(j, i))

    density = gaussian_kde(rn)
    xs = np.linspace(0,model.vol_surface.maxStrike(),len(rn))
    density.covariance_factor = lambda : .25
    density._compute_covariance()
    plt.plot(xs, density(xs))
    plt.legend()


distribution(SABR_beta5)
```

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

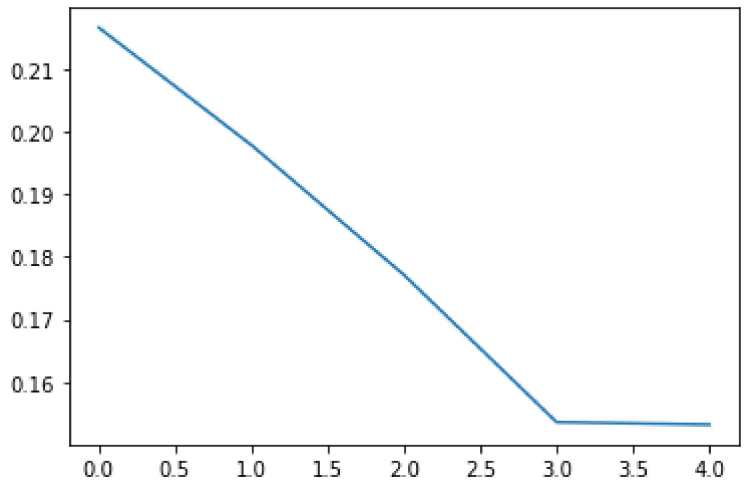


In [36]:

```
rn = [SABR_beta0.vol_surface.blackForwardVol(today, dates[-3],i) for i in [1000, 120

plt.plot(rn)
```

Out[36]:

```
[<matplotlib.lines.Line2D at 0x1e4c8eb7190>]
```



In [ ]: