In [1]:

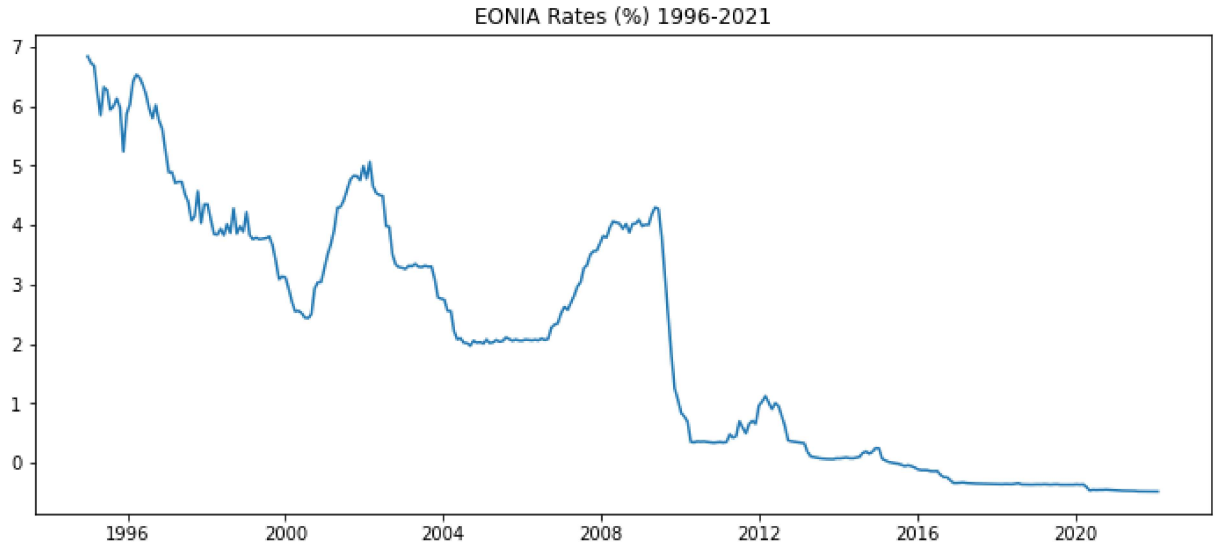
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import time
import datetime
import math
import QuantLib as ql
from scipy.optimize import minimize

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")
```

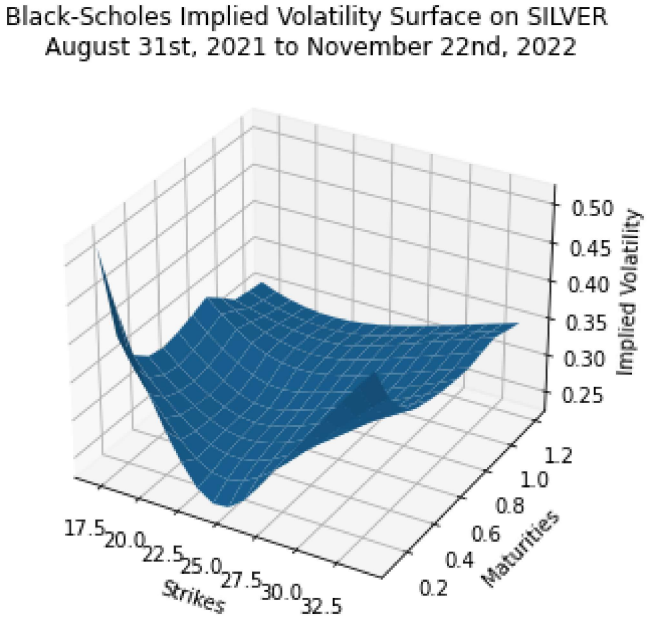
	Tenors	Spot Rate
0	0.25	-0.575
1	1.00	-0.557
2	3.00	-0.549
3	6.00	-0.529
4	12.00	-0.494

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



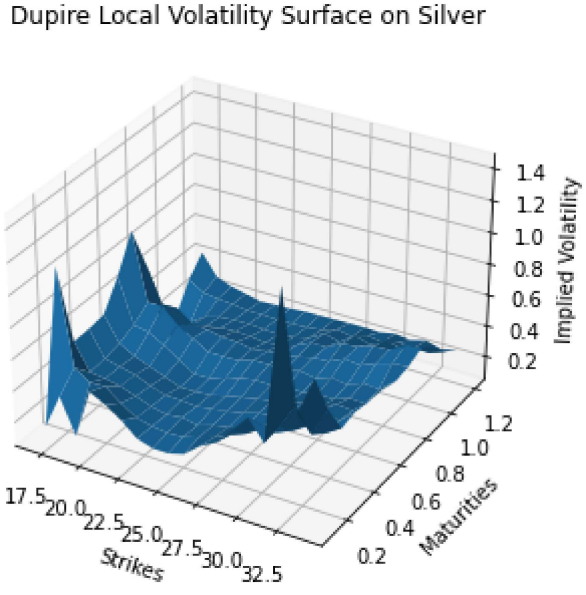
In [3]:

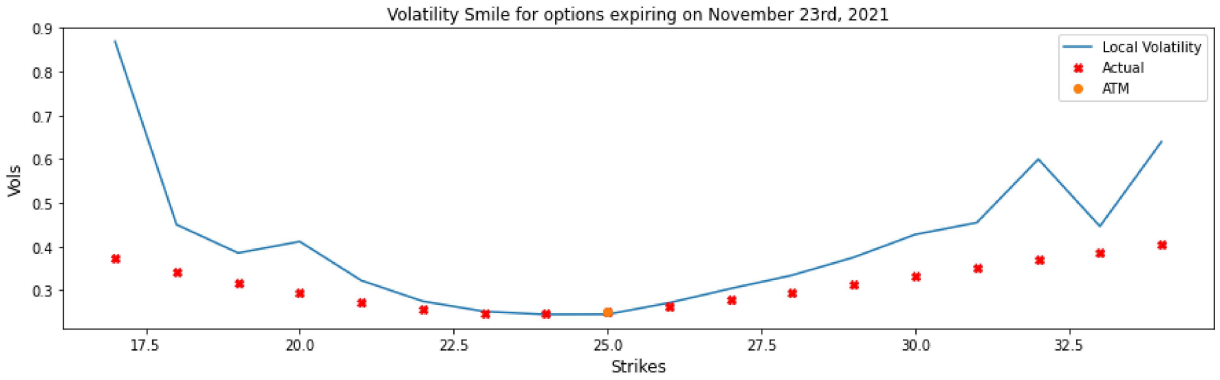
```
# BLACK VOLATILITY SURFACE

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
```

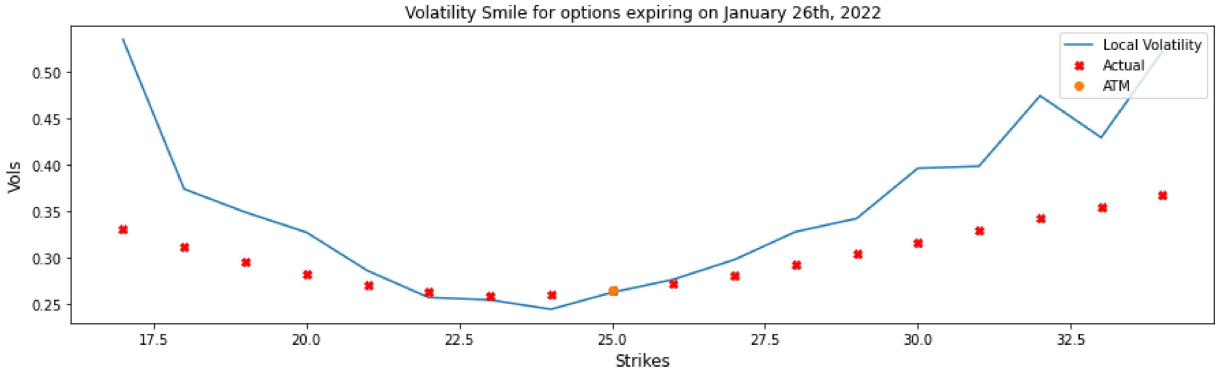


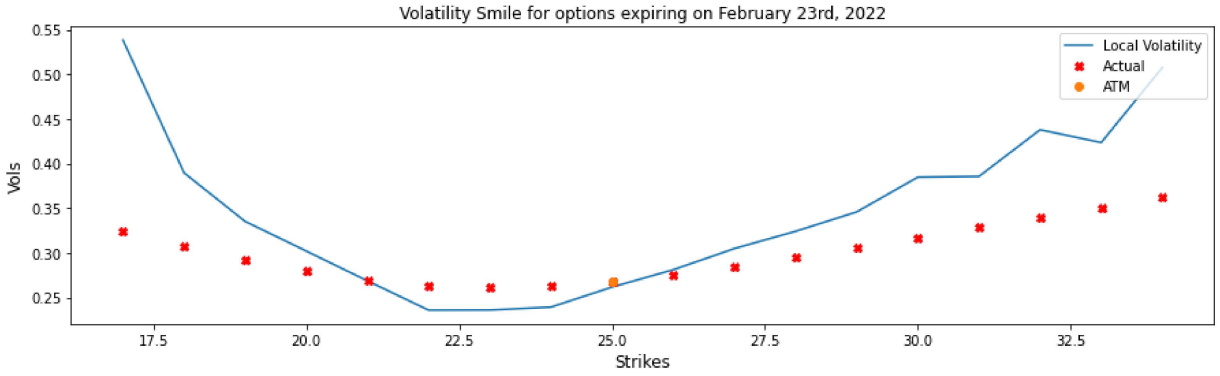
In [4]:

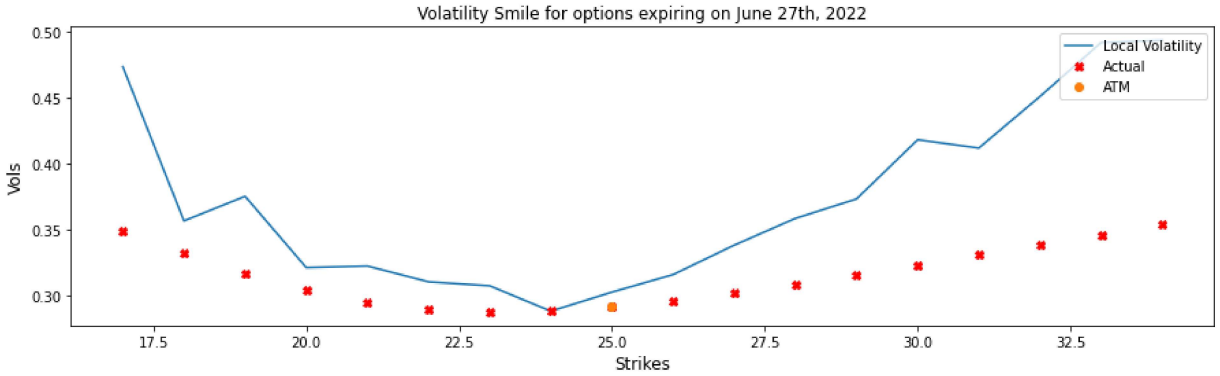
```
#DUPIRE LOCAL VOLATILITY SURFACE (NOT PLOTTABLE)

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)
```

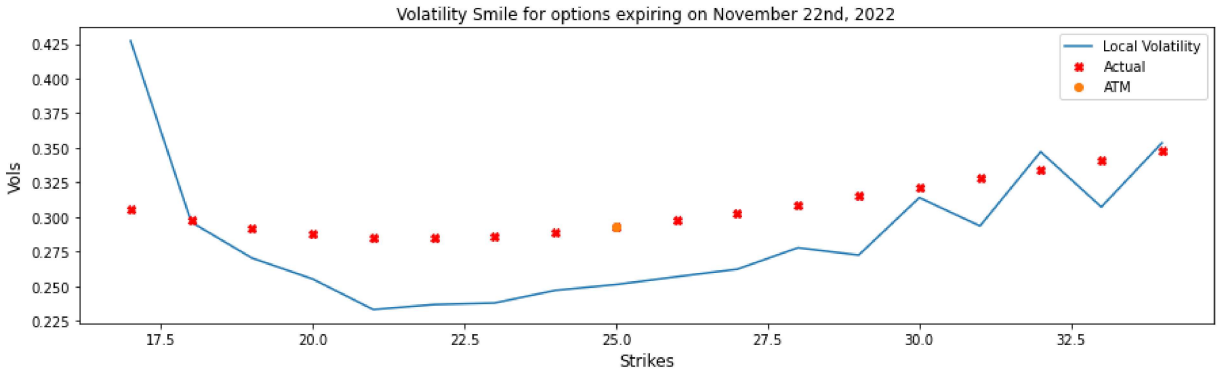












In [38]:

```
#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 ({})".
```

```
UsageError: Line magic function `%%time` not found.
```

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, Silver calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	17.000000	0.441963	0.459567	3.983277
1	18.000000	0.584625	0.598604	2.391130
2	19.000000	0.772837	0.775236	0.310407
3	20.000000	1.016130	0.999160	-1.670087
4	21.000000	1.320553	1.281587	-2.950709
5	22.000000	1.694830	1.633884	-3.595998
6	23.000000	2.140845	2.064862	-3.549233
7	24.000000	2.657936	2.577651	-3.020570
8	25.000000	3.241375	3.168390	-2.251649
9	26.000000	2.879332	2.821984	-1.991708
10	27.000000	2.570582	2.533089	-1.458539
11	28.000000	2.307485	2.290853	-0.720777
12	29.000000	2.082518	2.086032	0.168738
13	30.000000	1.890759	1.911202	1.081251
14	31.000000	1.724693	1.760562	2.079688
15	32.000000	1.581650	1.629603	3.031865
16	33.000000	1.456465	1.514822	4.006769
17	34.000000	1.348394	1.413472	4.826319

Heston Model 1, Silver parameters output

	Value
v0	0.473522
kappa	0.468403
theta	1.267293
sigma	0.303634
rho	0.000000
avgError	2.393818

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

Heston Model 2, Silver calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	17.000000	0.441963	0.494731	11.939443
1	18.000000	0.584625	0.626302	7.128863
2	19.000000	0.772837	0.776131	0.426138
3	20.000000	1.016130	0.950184	-6.489953
4	21.000000	1.320553	1.157844	-12.321257

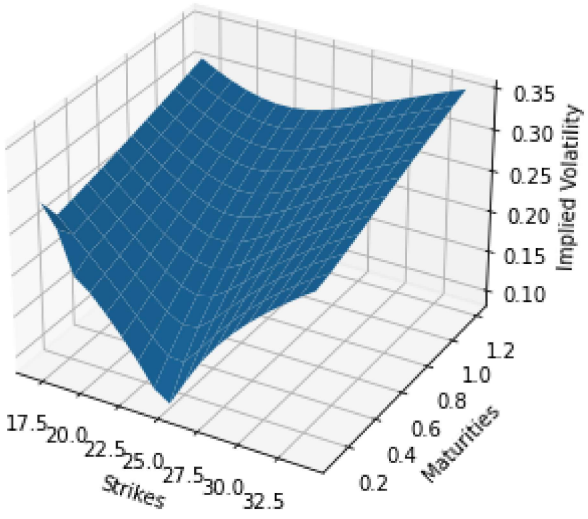
	Strikes	Market Value	Model Value	Relative Error (%)
5	22.000000	1.694830	1.414538	-16.538056
6	23.000000	2.140845	1.745868	-18.449578
7	24.000000	2.657936	2.188864	-17.647980
8	25.000000	3.241375	2.772408	-14.468139
9	26.000000	2.879332	2.477830	-13.944281
10	27.000000	2.570582	2.271424	-11.637754
11	28.000000	2.307485	2.120275	-8.113146
12	29.000000	2.082518	2.004393	-3.751439
13	30.000000	1.890759	1.912151	1.131395
14	31.000000	1.724693	1.836541	6.485091
15	32.000000	1.581650	1.773126	12.106125
16	33.000000	1.456465	1.718957	18.022530
17	34.000000	1.348394	1.671993	23.998798

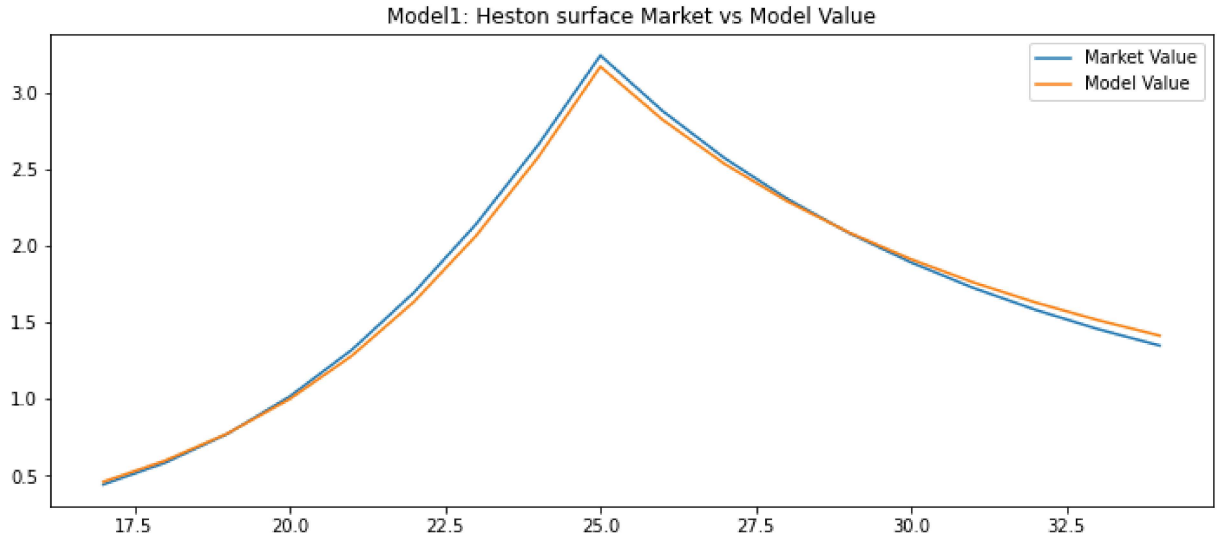
Heston Model 2, Silver parameters output

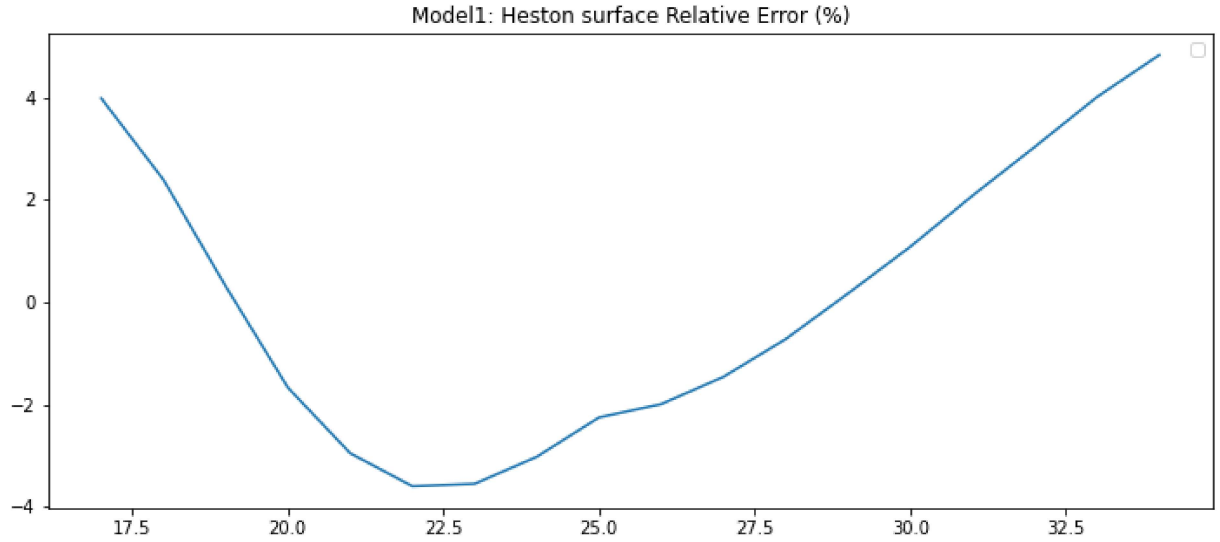
	Value
v0	1.498890
kappa	0.000000
theta	26.829691
sigma	0.268779
rho	2.496803
avgError	11.366665

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

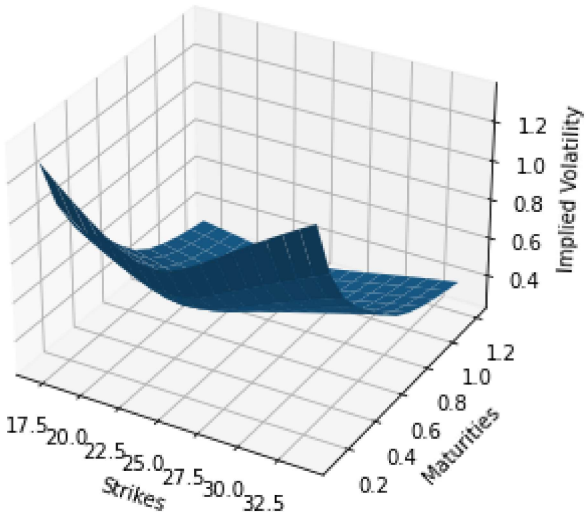
Heston Volatility Surface for Heston Model 1, Silver

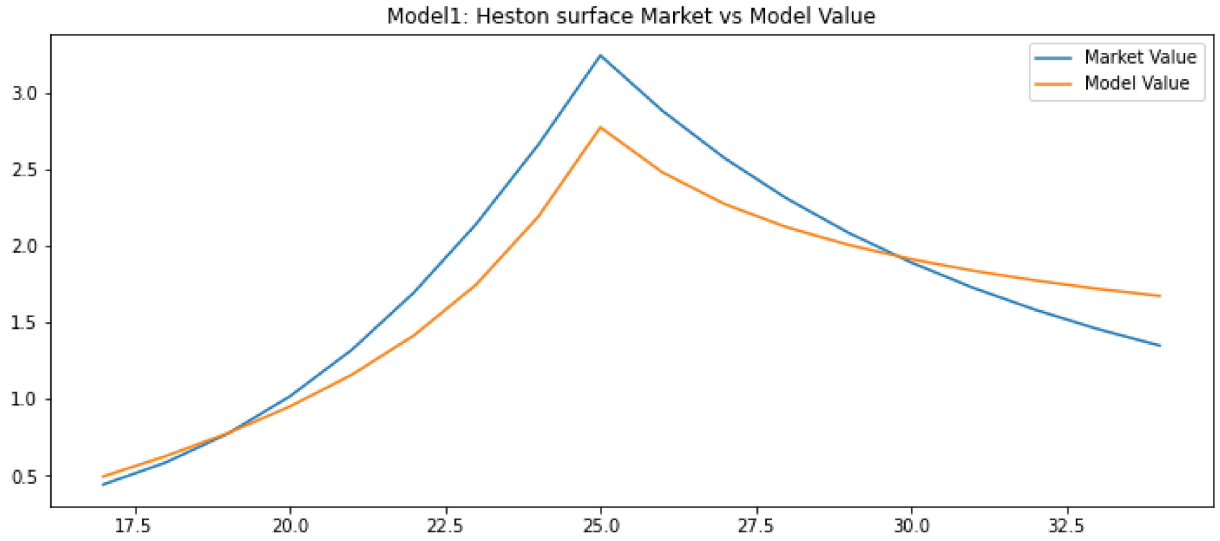


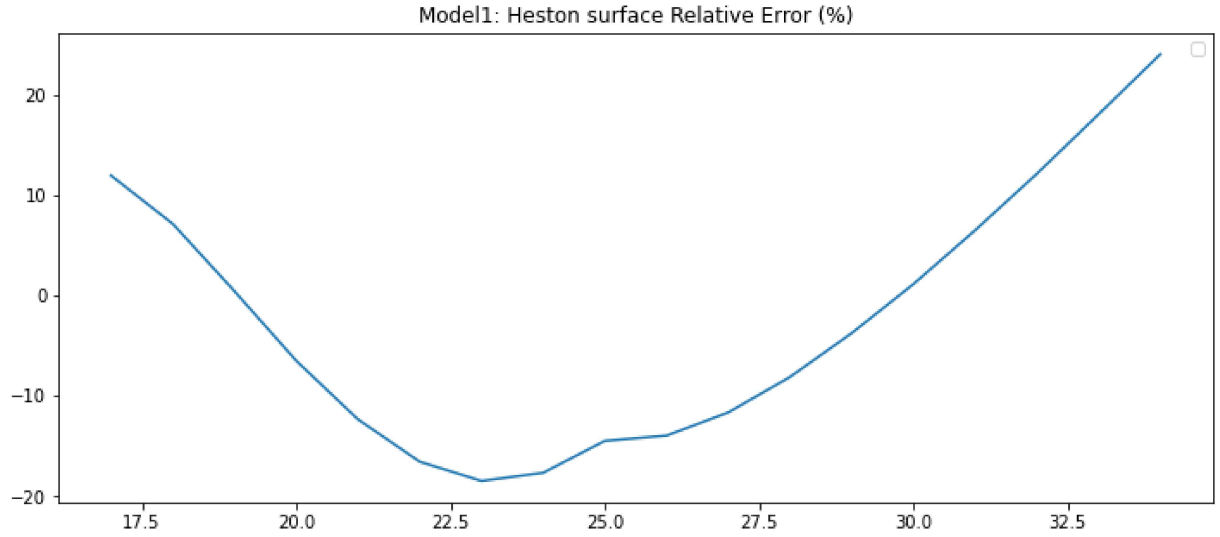




Heston Volatility Surface for Heston Model 2, Silver

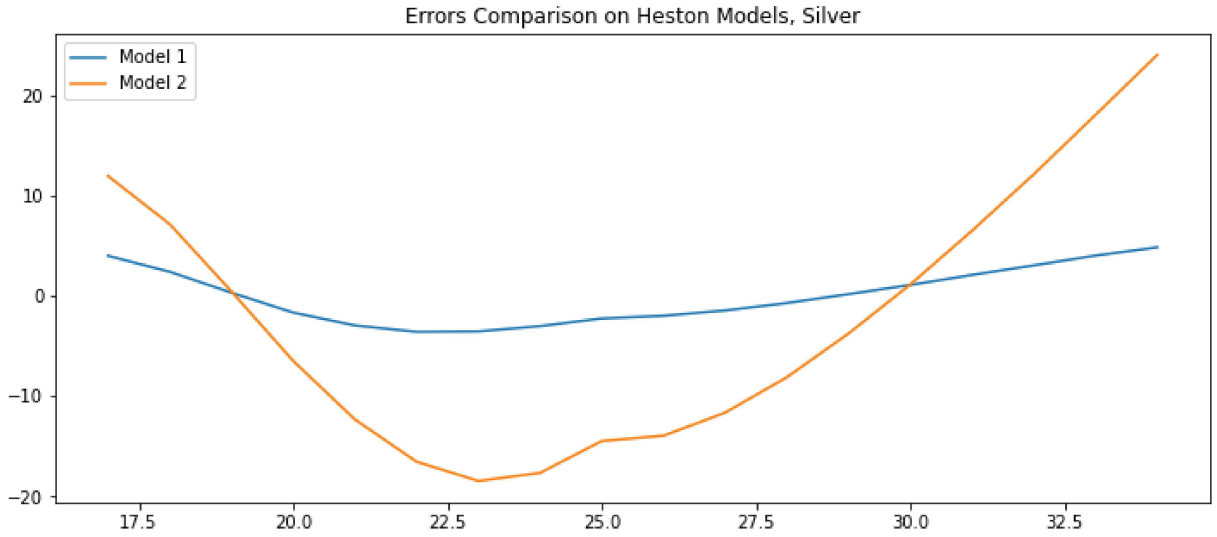


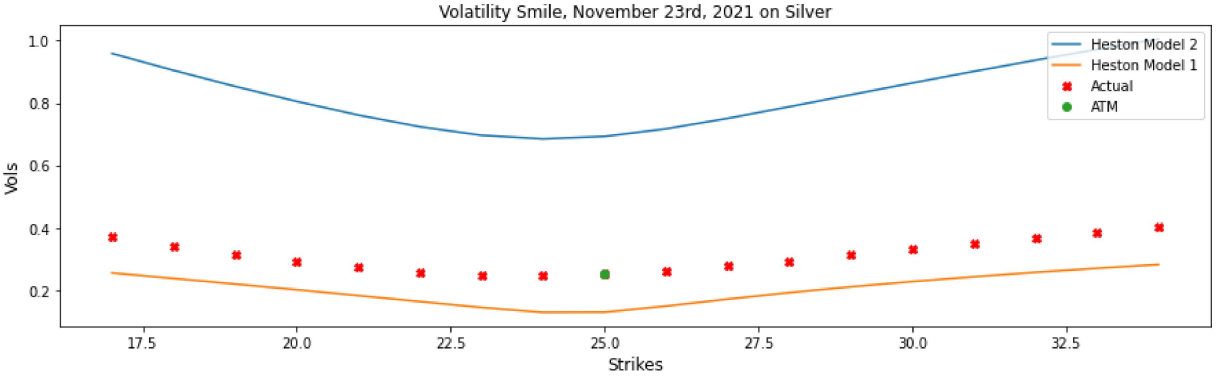


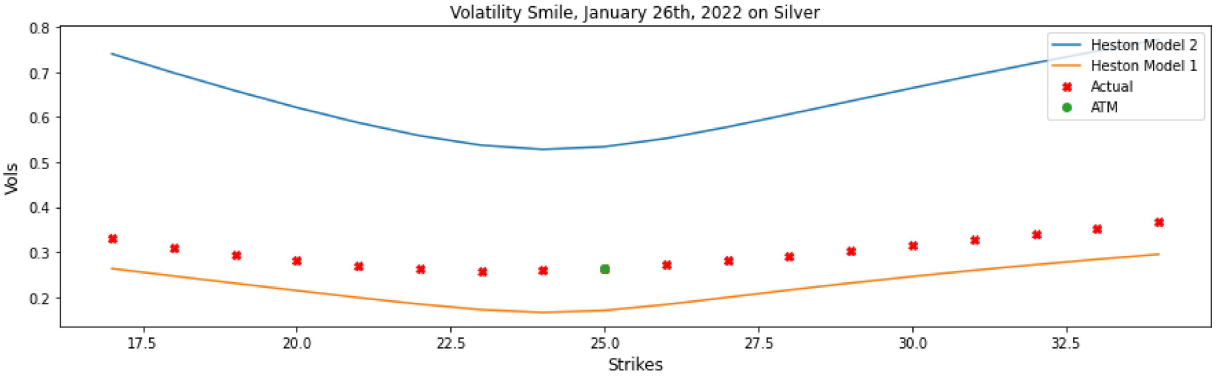


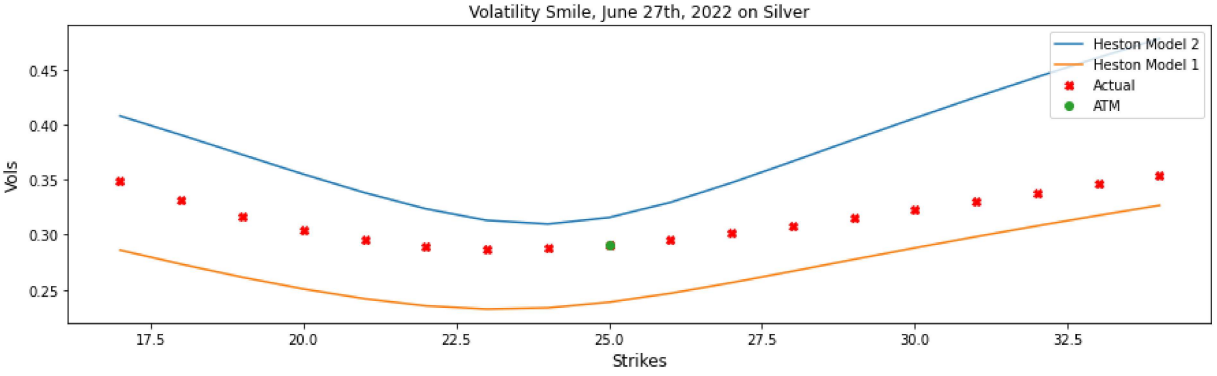
In [7]:

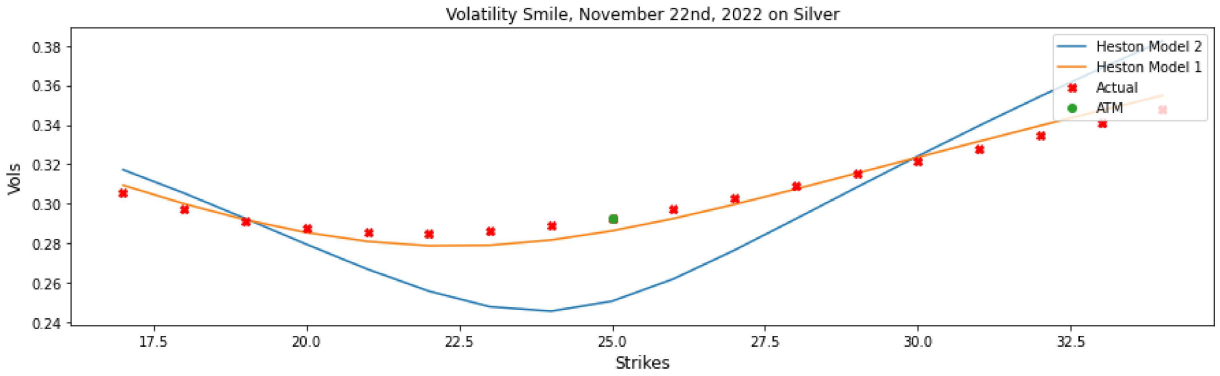
```
# Relative error comparison
plt.figure(figsize=plot_size)
plt.plot(hestonModel1.strks, hestonModel1.relativeError, label="Model 1")
plt.plot(hestonModel2.strks, hestonModel2.relativeError, label="Model 2")
plt.title("Errors Comparison on Heston Models, {}".format(data_label));
plt.legend()

# 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, Silver
Model 1
Model 2
20
10
0
−10
−20
17.5
20.0
22.5
25.0
27.5
30.0
32.5


Volatility Smile, November 23rd, 2021 on Silver
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes
1.0
0.8
0.6
0.4
0.2
17.5
20.0
22.5
25.0
27.5
30.0
32.5


Volatility Smile, January 26th, 2022 on Silver
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes
0.8
0.7
0.6
0.5
0.4
0.3
0.2
17.5
20.0
22.5
25.0
27.5
30.0
32.5


Volatility Smile, June 27th, 2022 on Silver
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes
0.45
0.40
0.35
0.30
0.25
17.5
20.0
22.5
25.0
27.5
30.0
32.5



In [8]:

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

from initialize import *
from plotting import *

# CALIBRATE SABR VOLATILITY SURFACE

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

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

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


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

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

```
        x = self.set_init_conds()

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

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

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

    def vols_by_method(self, strike, alpha, beta, nu, rho):
        if self.method == "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)
```

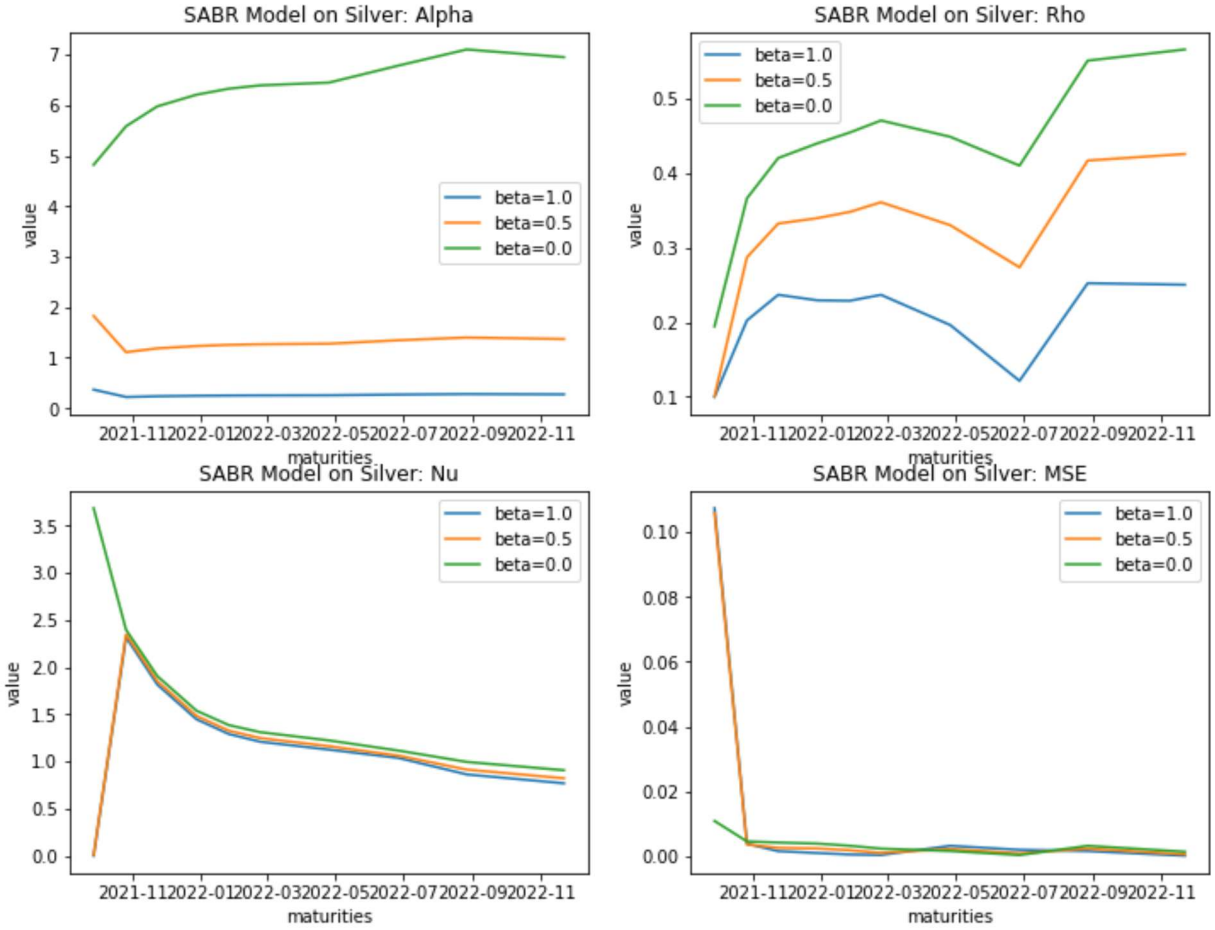
```
            self.nu.append(volSABR.nu)
            self.rho.append(volSABR.rho)

            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)
```

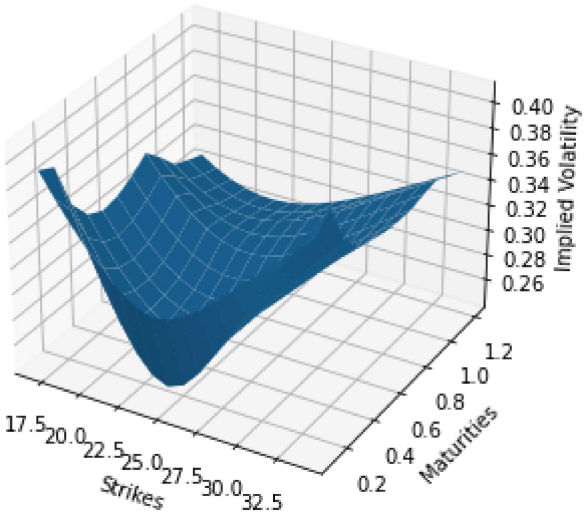
In [9]:

```
# SABR Volatility model
%%time

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
```

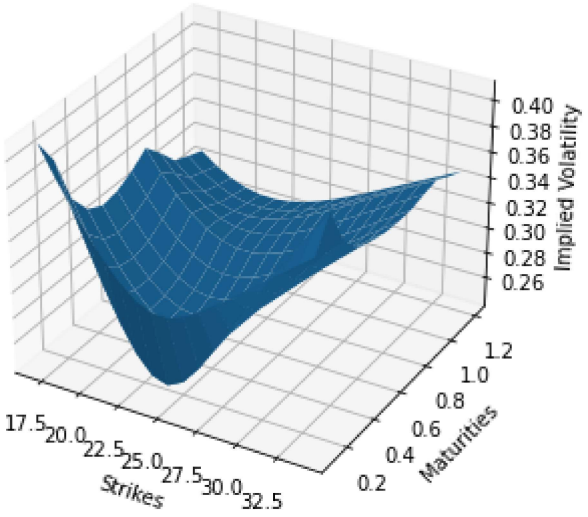
```
C:\Users\somig\AppData\Roaming\Python\Python39\site-packages\scipy\optimize\optimiz
e.py:282: RuntimeWarning: Values in x were outside bounds during a minimize step, cl
ipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```



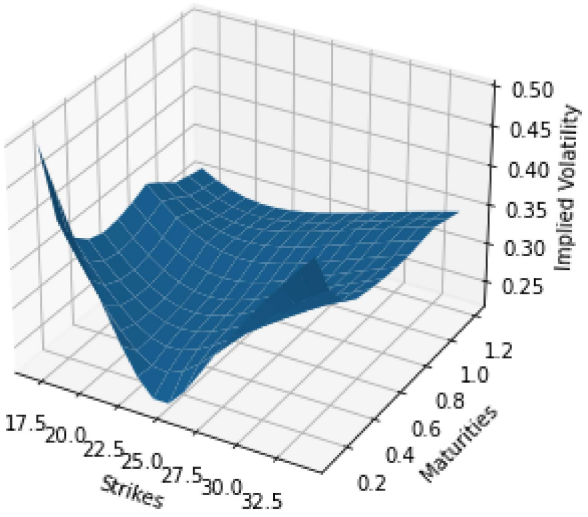
SABR, beta=1, Silver

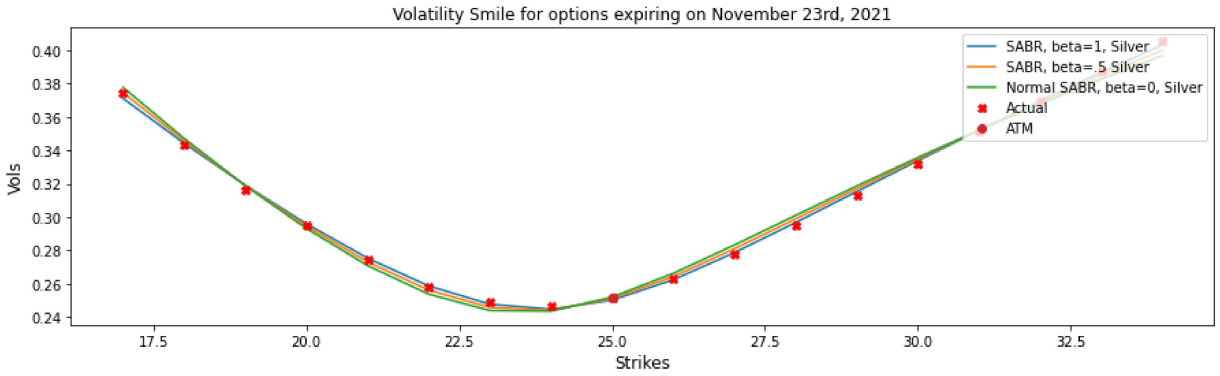


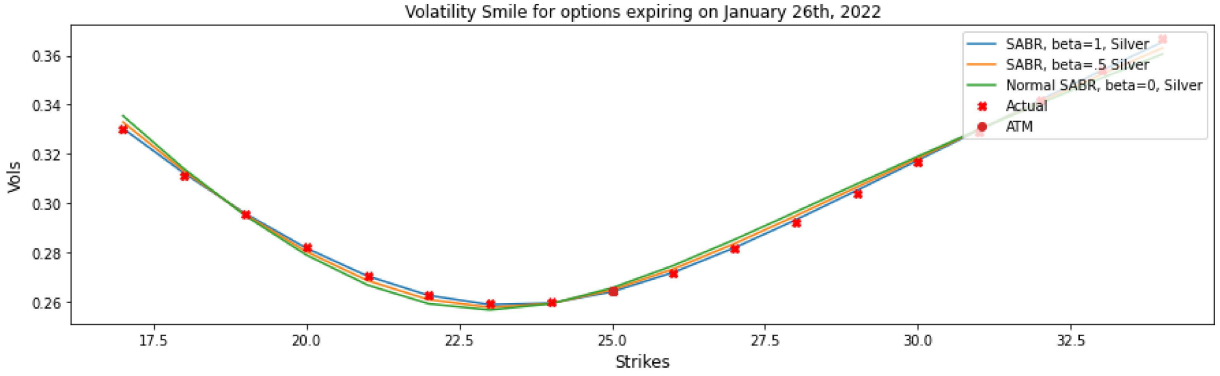
SABR, beta=.5 Silver

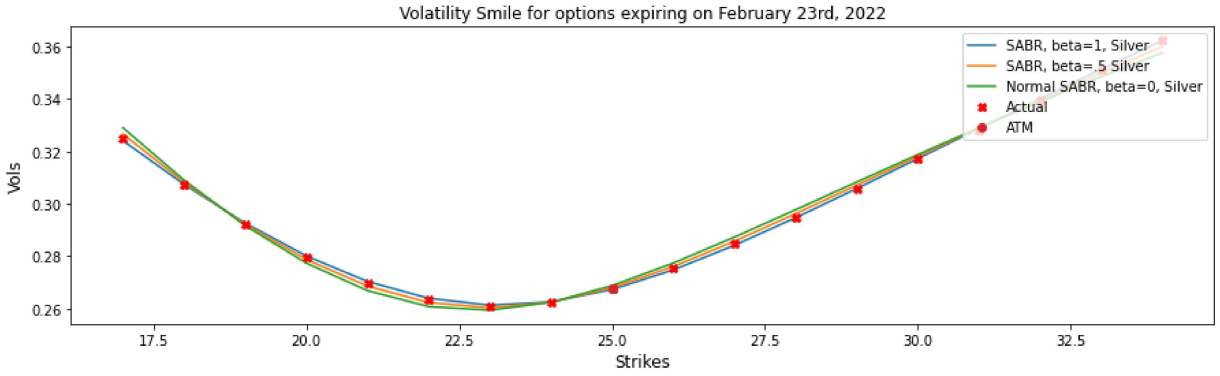


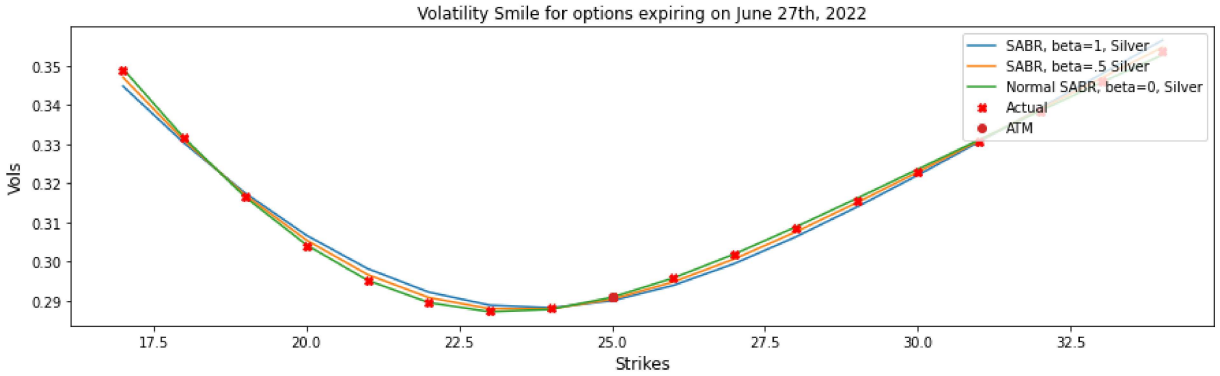
Normal SABR, beta=0, Silver

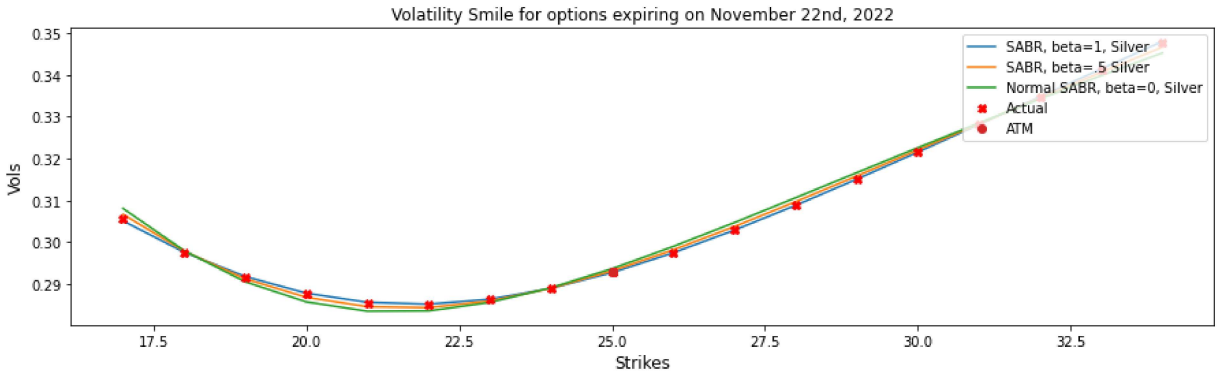












In [37]:
```
# Shifted SABR Volatility model
```

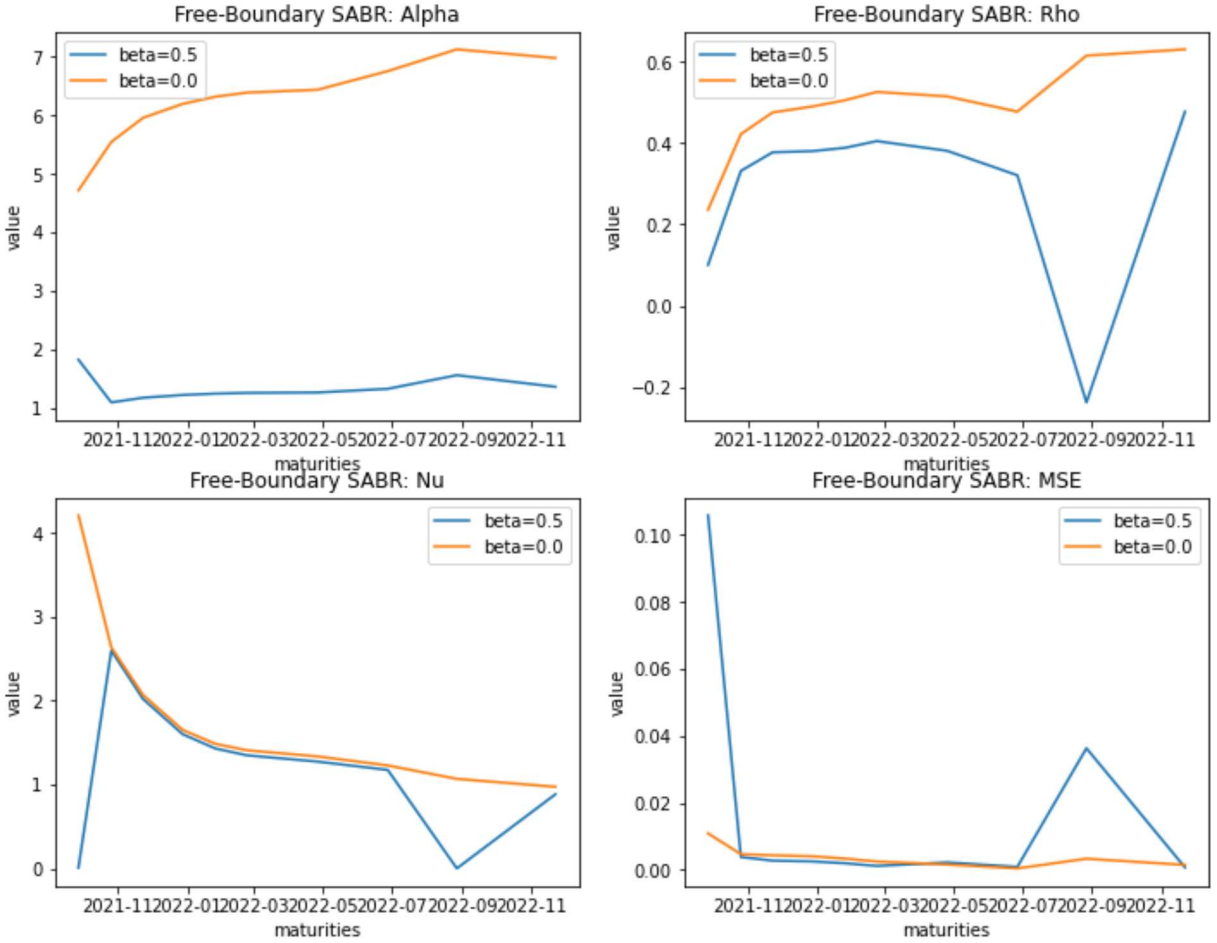
```
shft = .50 * current_price
shiftedSABR_beta1 = SABRVolatilitySurface(beta=1, shift=shft,strks=strikes, label="S
shiftedSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=shft, strks=strikes,label="
shiftedSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=shft, strks=strikes,label="

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

```
UsageError: Line magic function `%%time` not found.
```

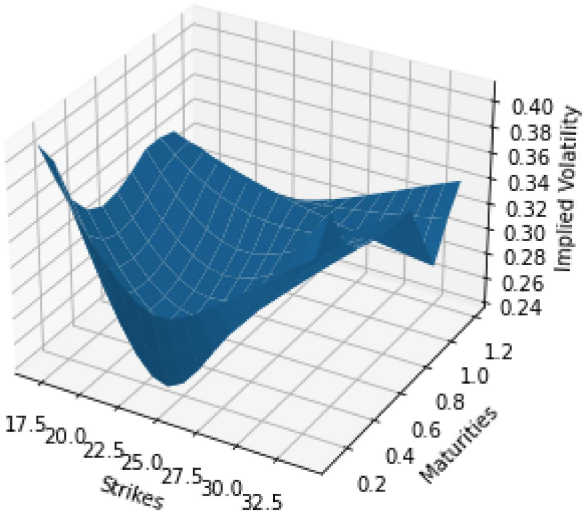
In [11]:

```
# Free-Boundary SABR Volatility model

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

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

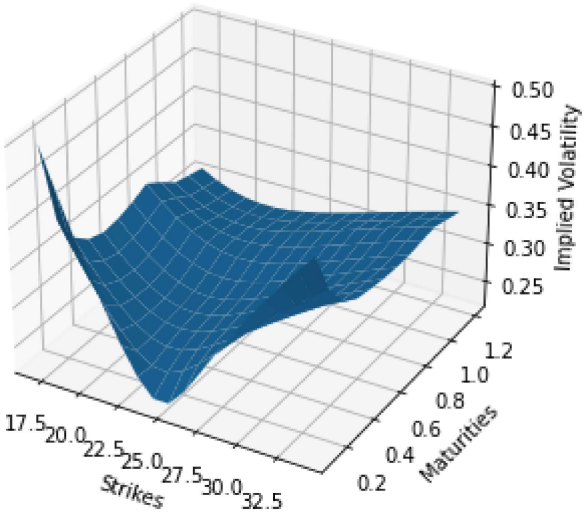
```
C:\Users\somig\AppData\Roaming\Python\Python39\site-packages\scipy\optimize\optimiz
e.py:282: RuntimeWarning: Values in x were outside bounds during a minimize step, cl
ipping to bounds
  warnings.warn("Values in x were outside bounds during a "
```

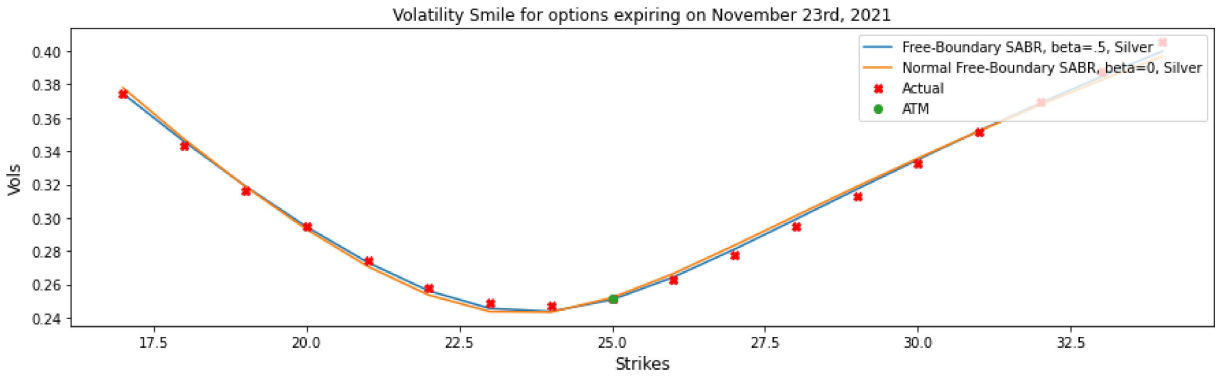


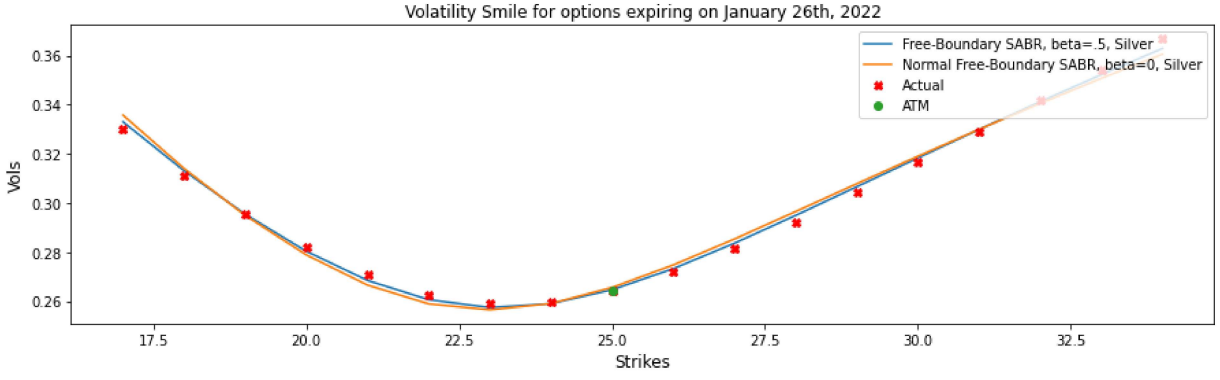
Free-Boundary SABR, beta=.5, Silver

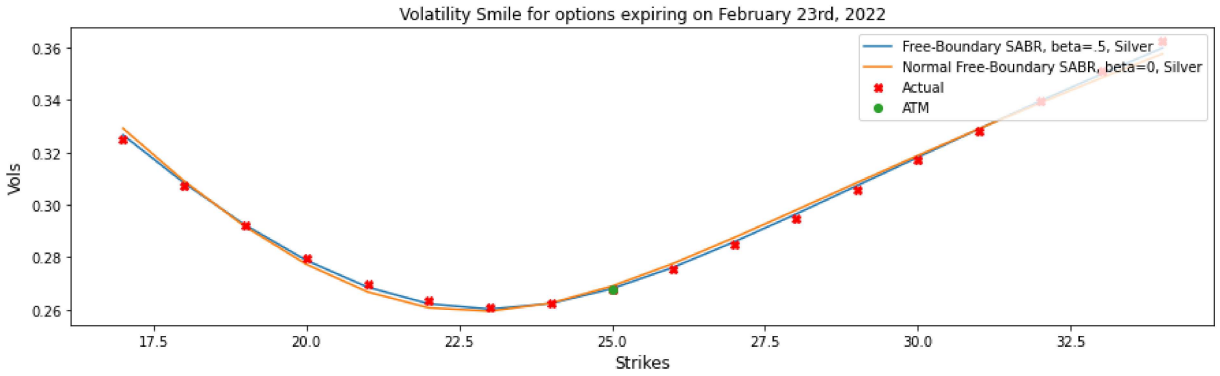


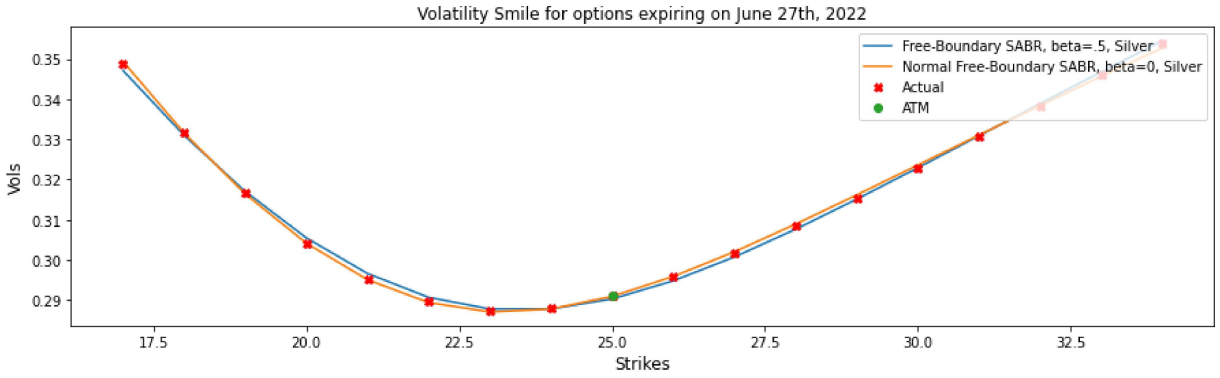
Normal Free-Boundary SABR, beta=0, Silver

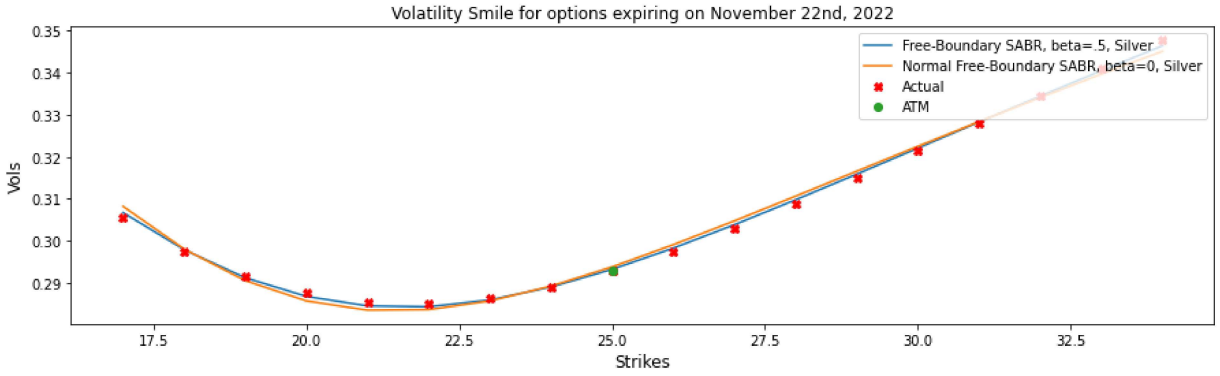












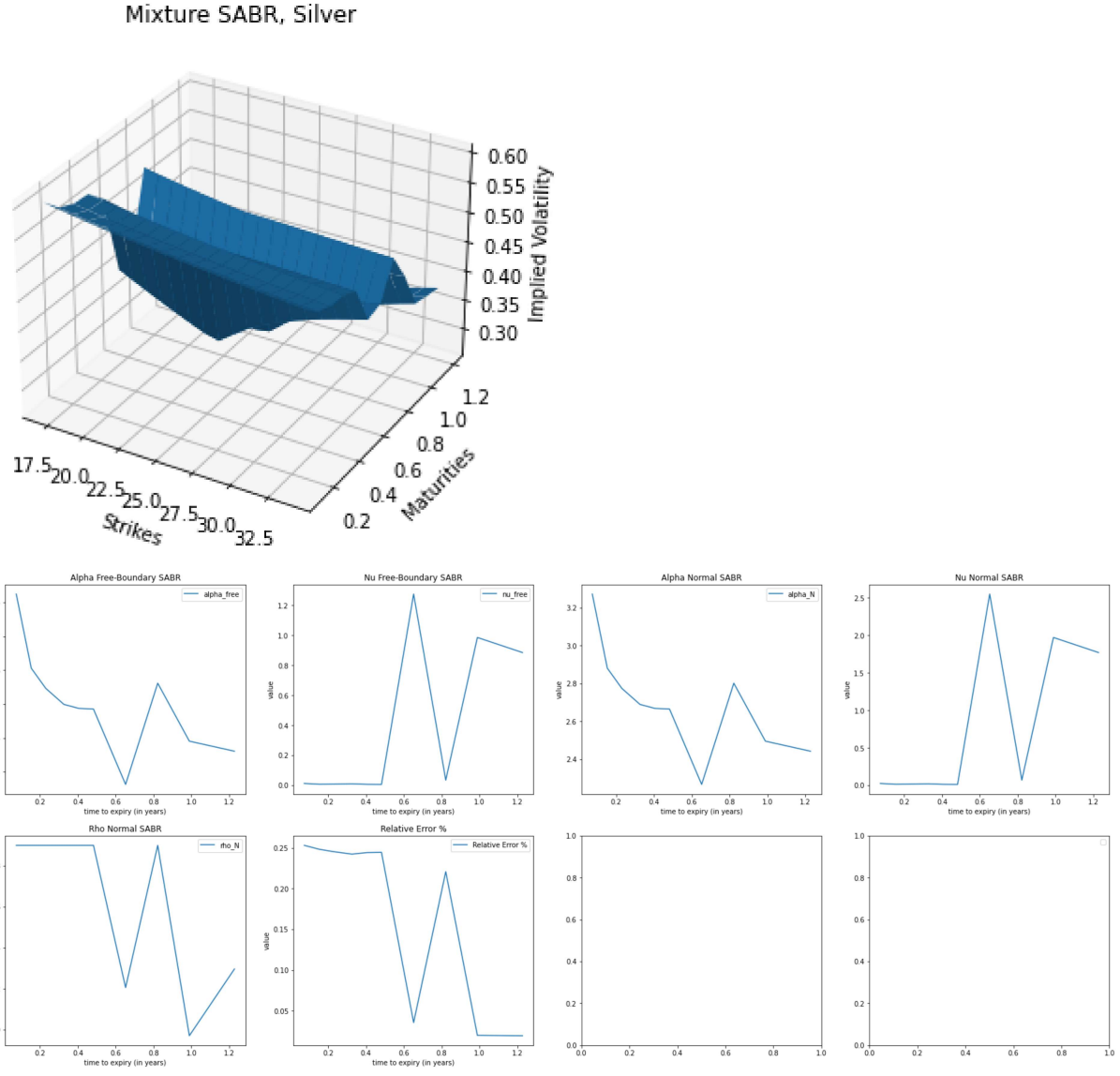
In [12]:

```
# Mixed SABR

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

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSI
September 27th, 2021	1.824276	0.5	0.012676	0.0	1.824276	0.0	0.025352	0.990000	0.25299
October 26th, 2021	1.605741	0.5	0.007406	0.0	1.605741	0.0	0.014812	0.990000	0.24815
November 23rd, 2021	1.546282	0.5	0.008135	0.0	1.546282	0.0	0.016270	0.990000	0.24519
December 28th, 2021	1.499043	0.5	0.010128	0.0	1.499043	0.0	0.020257	0.990000	0.24225
January 26th, 2022	1.487262	0.5	0.006905	0.0	1.487262	0.0	0.013810	0.990000	0.24412
February 23rd, 2022	1.485328	0.5	0.006224	0.0	1.485328	0.0	0.012448	0.990000	0.24441

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSI
April 26th, 2022	1.263259	0.5	1.275493	0.0	1.263259	0.0	2.550987	0.920582	0.03550
June 27th, 2022	1.561563	0.5	0.034805	0.0	1.561563	0.0	0.069611	0.990000	0.22069
August 27th, 2022	1.390720	0.5	0.985957	0.0	1.390720	0.0	1.971915	0.897071	0.01983
November 22nd, 2022	1.361176	0.5	0.885531	0.0	1.361176	0.0	1.771062	0.929645	0.01940

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

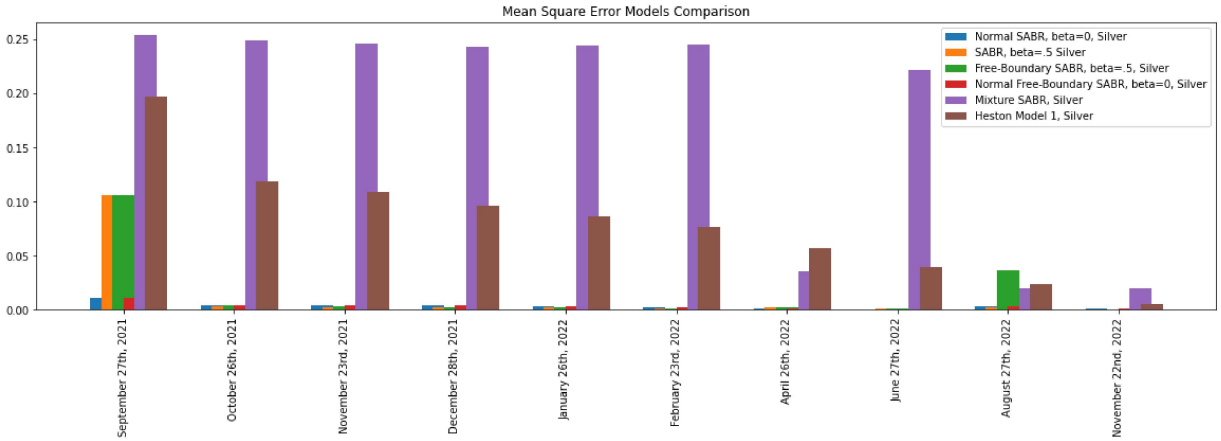


In [13]:

```
# VOLATILITY SMILES ERRORS COMPARISON

fig = plt.figure(figsize=(20, 5), )
width = .20
plt.bar(np.arange(len(maturities)) - 2*width/2, SABR_beta0.errors, label=SABR_beta0.
plt.bar(np.arange(len(maturities)) - width/2, SABR_beta5.errors, label=SABR_beta5.la
plt.bar(np.arange(len(maturities)) , freeSABR_beta5.errors, label=freeSABR_beta5.lab
plt.bar(np.arange(len(maturities)) + width/2, freeSABR_beta0.errors, label=freeSABR_
plt.bar(np.arange(len(maturities)) + 2*width/2, mixtureSABR.errors, label=mixtureSAB
plt.bar(np.arange(len(maturities)) + 3*width/2, hestonModel1.errors, label=hestonMod
```

```
plt.xticks(np.arange(len(maturities)), dates, rotation='vertical')
plt.title("Mean Square Error Models Comparison")
plt.legend()
```

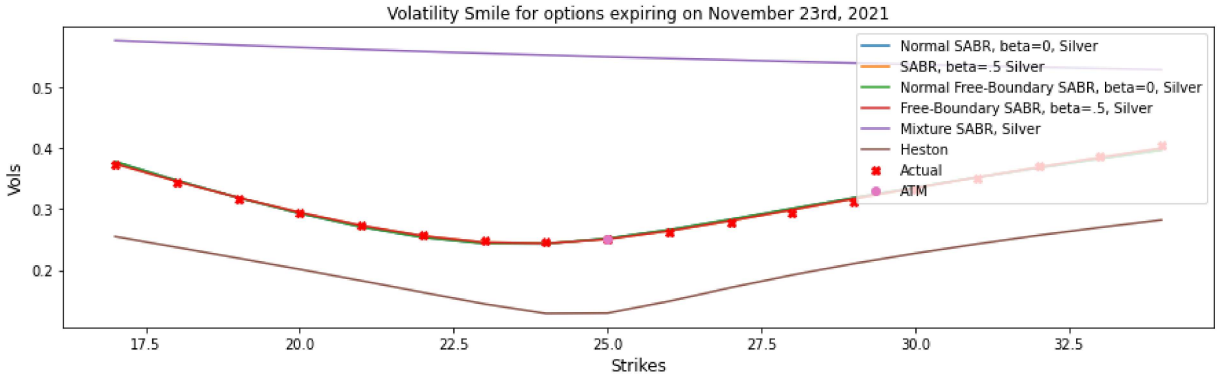
Out[13]: <matplotlib.legend.Legend at 0x207f61c7790>

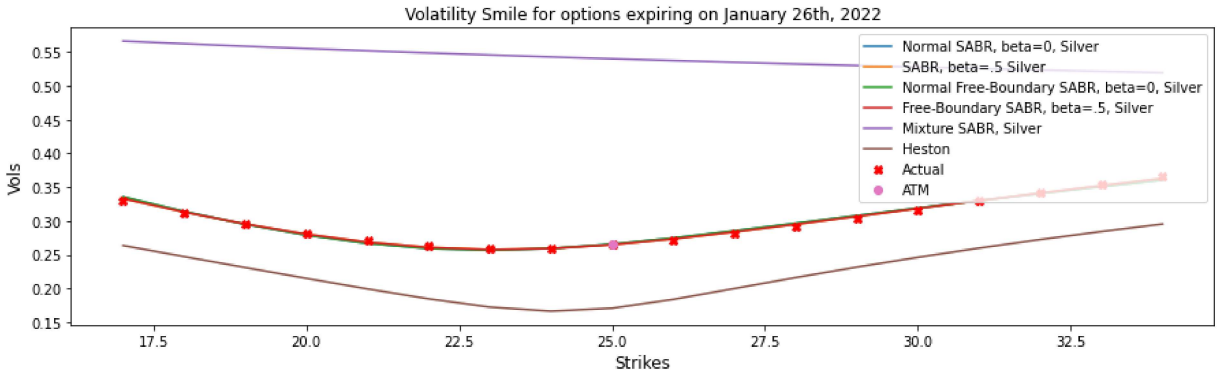


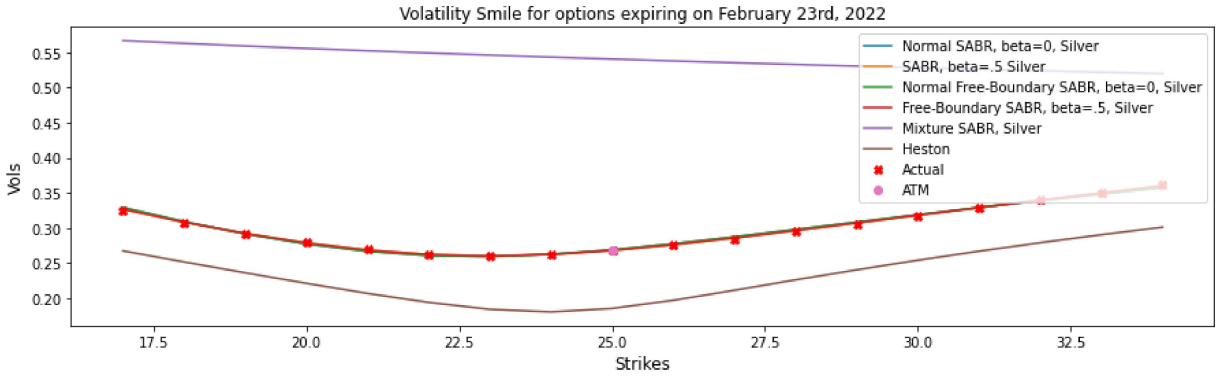
In [14]:

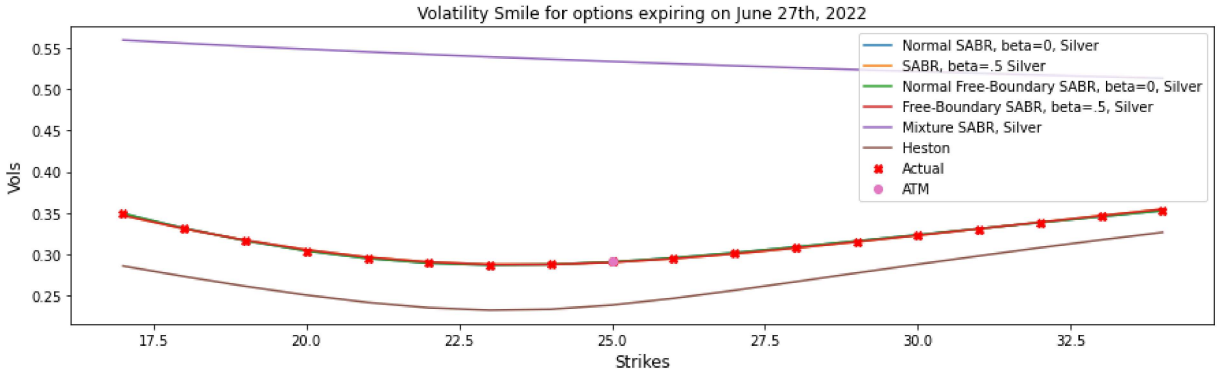
```
# Volatility Smiles Comparisons (Final)

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])
```

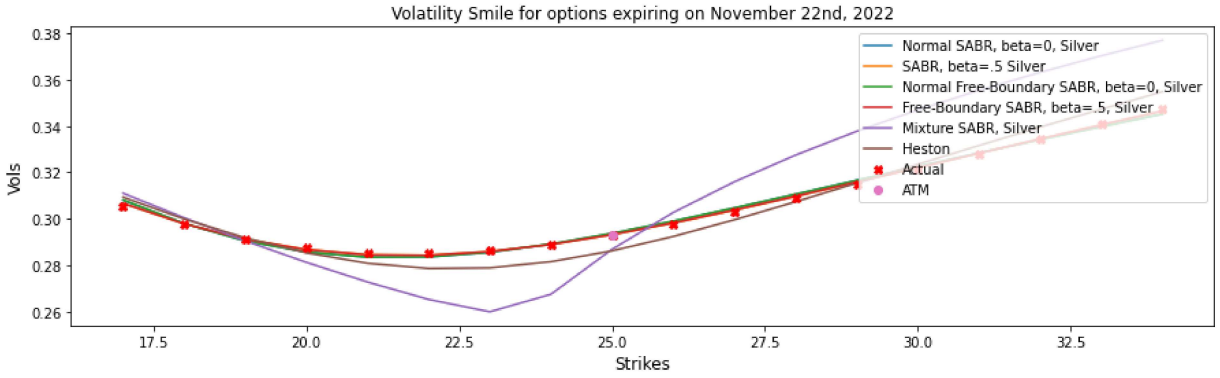
	Error
Normal SABR, beta=0, Silver	0.003617
SABR, beta=.5 Silver	0.012385
Normal Free-Boundary SABR, beta=0, Silver	0.003667
Free-Boundary SABR, beta=.5, Silver	0.015829
Mixture SABR, Silver	0.177258







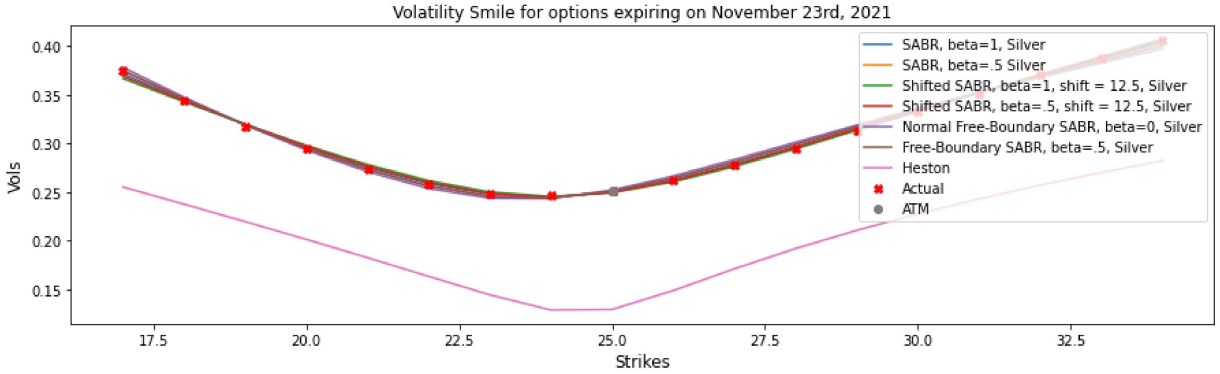


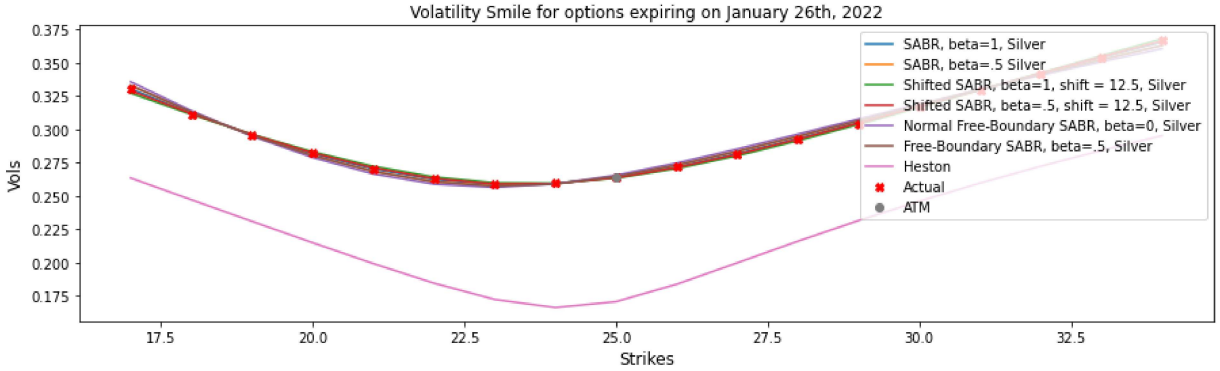


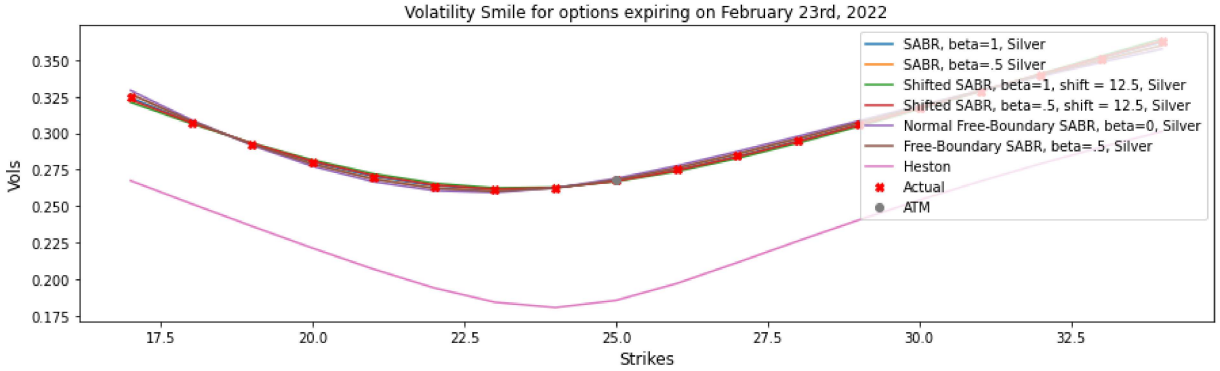
In [15]:

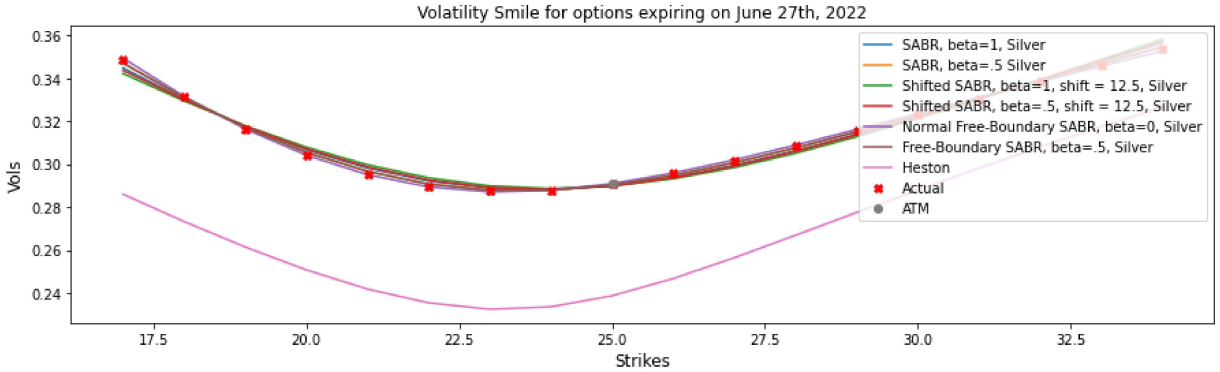
```
# Volatility Smiles Comparisons 2 (Final)

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])
```

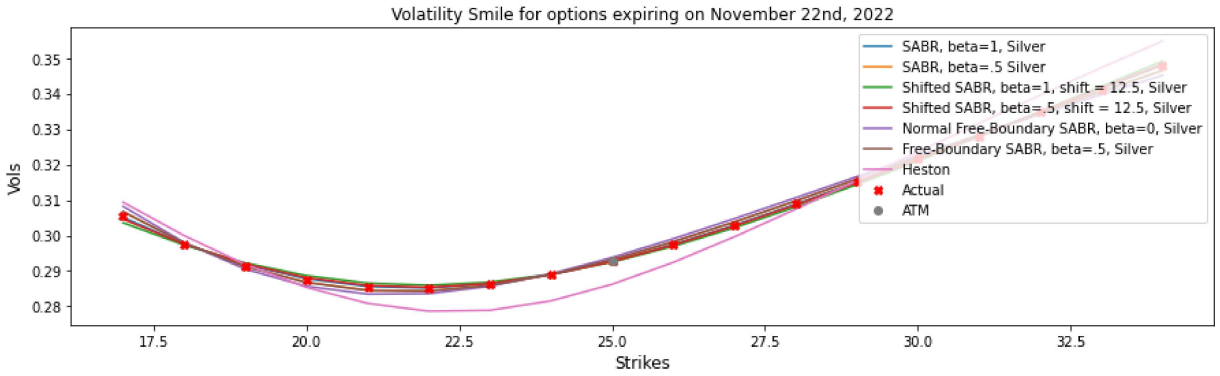
	Error
SABR, beta=1, Silver	0.012200
SABR, beta=.5 Silver	0.012385
Shifted SABR, beta=1, shift = 12.5, Silver	0.012953
Shifted SABR, beta=.5, shift = 12.5, Silver	0.012346
Normal Free-Boundary SABR, beta=0, Silver	0.003667
Free-Boundary SABR, beta=.5, Silver	0.015829





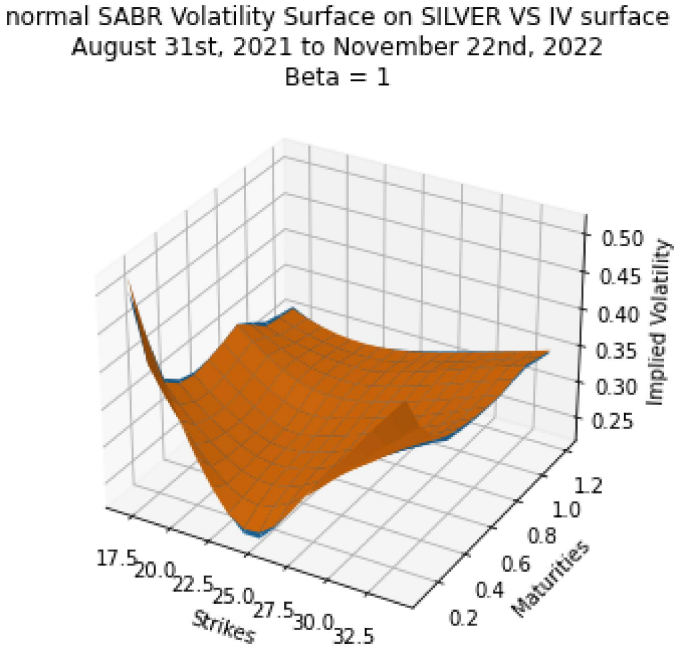




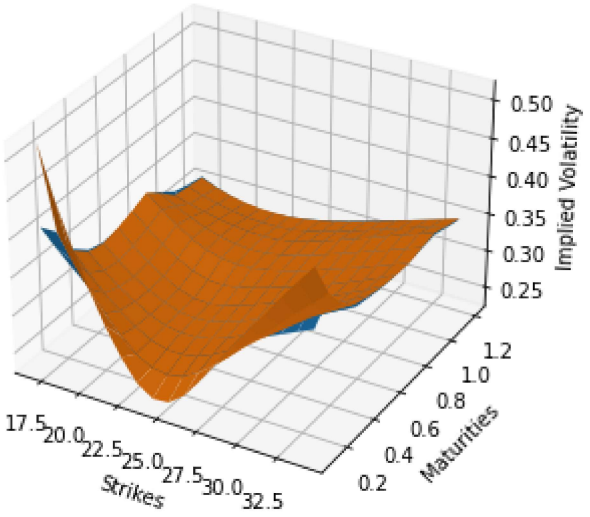


In [16]:

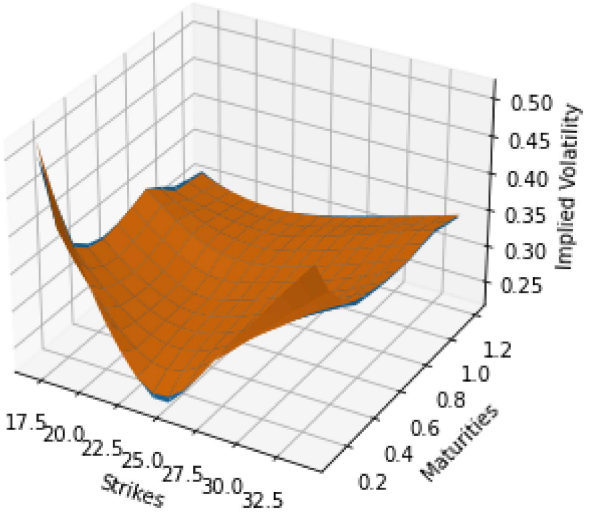
```
# Volatility Surfaces plots comparison

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)
```



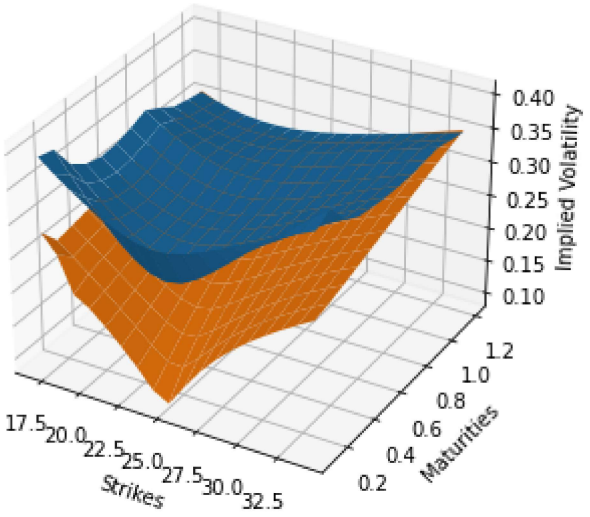
normal SABR Volatility Surface on SILVER VS IV surface
August 31st, 2021 to November 22nd, 2022
Beta = 0.5

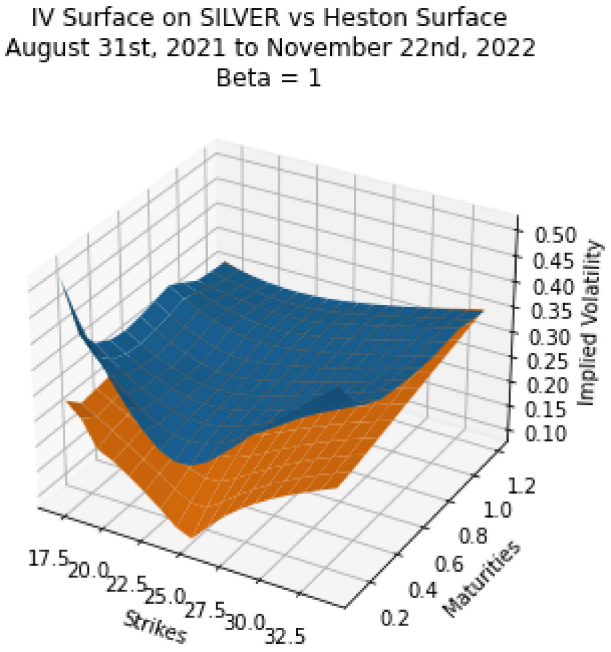


normal SABR Volatility Surface on SILVER VS IV surface
August 31st, 2021 to November 22nd, 2022
Beta = 0



SABR Volatility Surface on SILVER VS Heston surface
August 31st, 2021 to November 22nd, 2022
Beta = 1





In [83]:

```
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[83]: (None, None, None)