

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

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

```
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 Volatility Surface': 1.6279830932617188, 'SABR': 0.09308457374572754, 'Shifte
d SABR': 0.09809017181396484, 'Free Boundary SABR': 0.17015409469604492, 'Mixture SA
BR': 0.7897202968597412}
```

```
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]

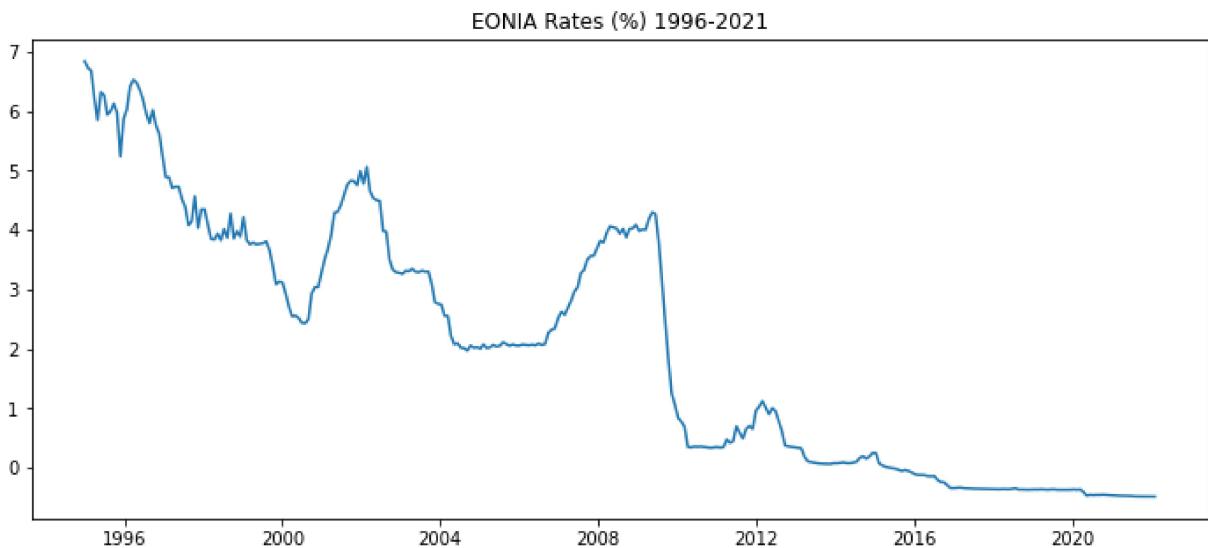
spot_rates = pd.DataFrame({"Tenors": tenors, "Spot Rate": rates})
spot_rates.set_index('Tenors')

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')
```

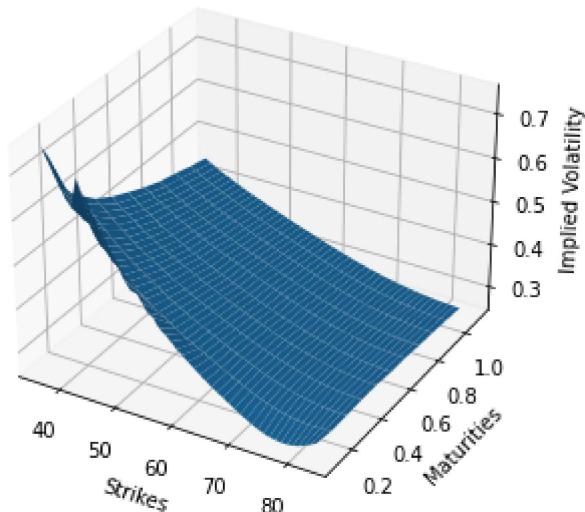


In [3]:

```
# 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
```

Black-Scholes Implied Volatility Surface on OIL
August 30th, 2021 to October 26th, 2022



In [4]:

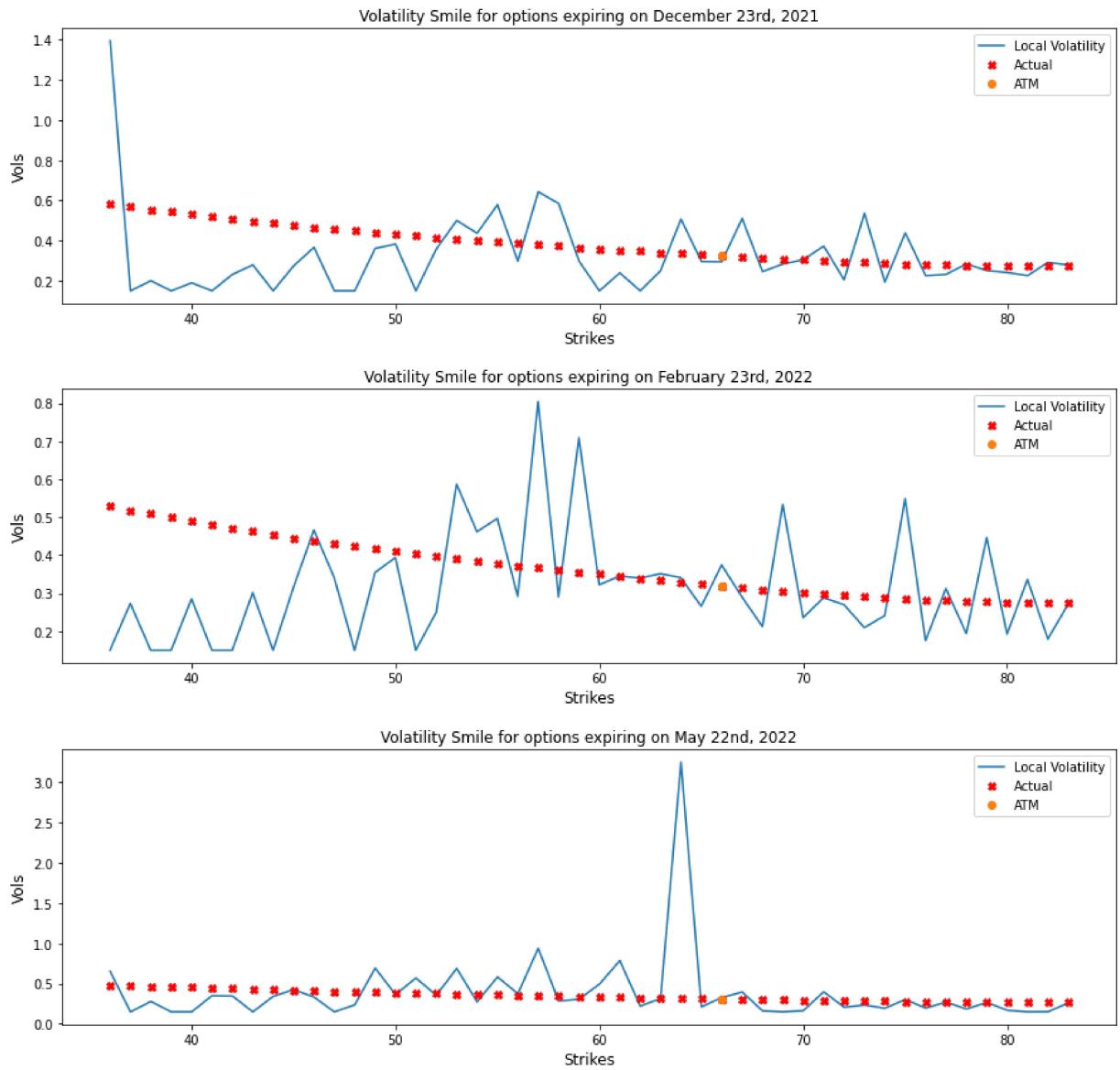
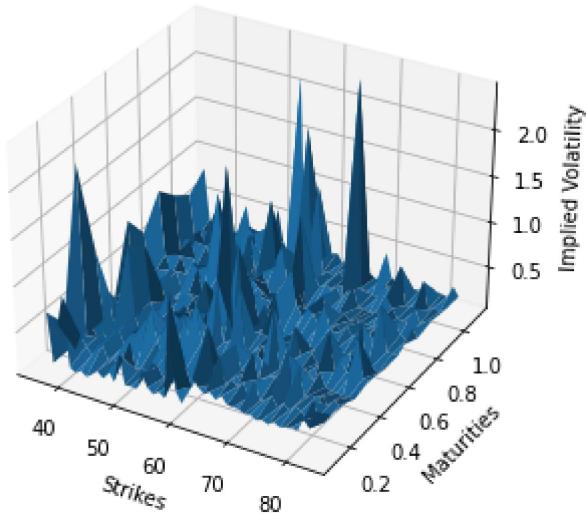
```
# DUPIRE LOCAL VOLATILITY SURFACE (NOT PLOTTABLE)

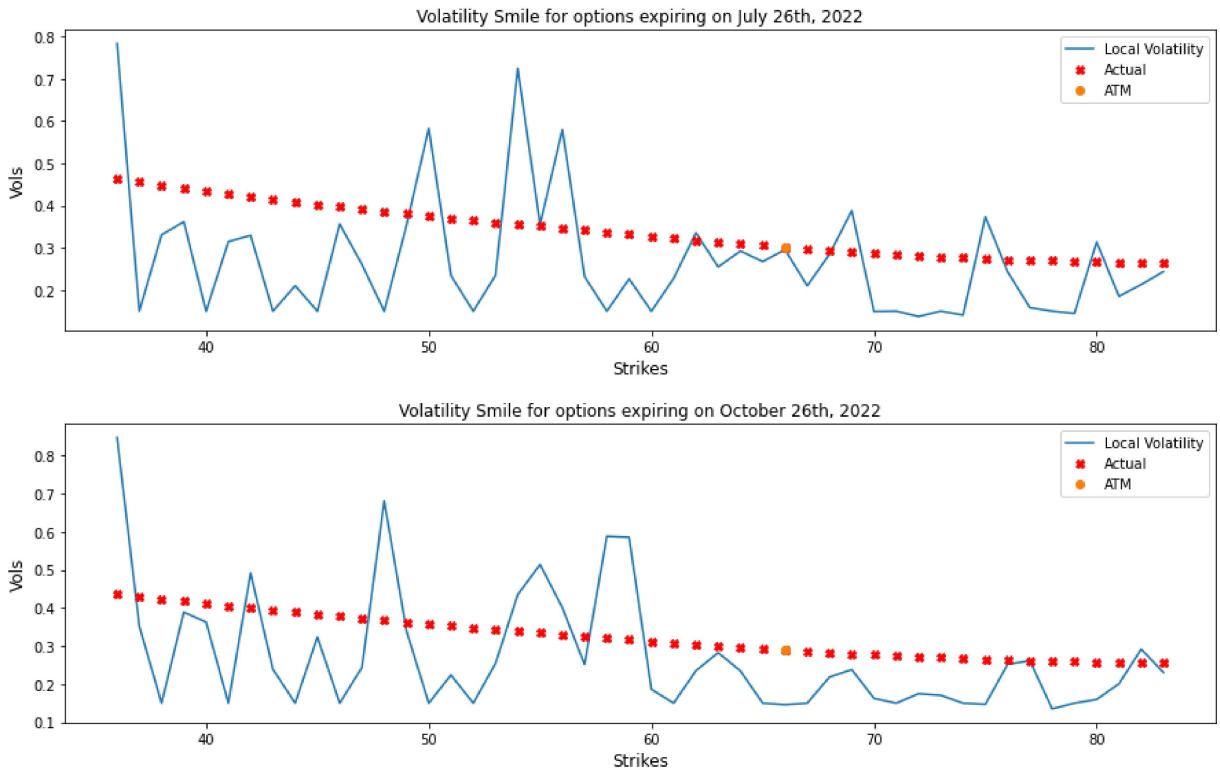
black_var_surface.setInterpolation("bicubic")
local_vol_handle = ql.BlackVolTermStructureHandle(black_var_surface)

# Local_vol_surface = ql.LocalVolSurface(local_vol_handle, flat_ts, dividend_ts, 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)
```

Dupire Local Volatility Surface on WTI Crude Oil





In [5]:

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

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

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(data))
```

```

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 (WTI Crude Oil)

	theta	kappa	sigma	rho	v0
Model1	0.150000	0.500000	0.200000	0.700000	0.010000
Model2	0.023000	0.009000	1.000000	0.950000	0.200000

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, WTI Crude Oil calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	36.000000	0.958086	1.199101	25.155870
1	37.000000	1.037192	1.312911	26.583265
2	38.000000	1.125773	1.434219	27.398577
3	39.000000	1.214808	1.563324	28.688957
4	40.000000	1.313419	1.700528	29.473371
5	41.000000	1.422697	1.846141	29.763453
6	42.000000	1.531639	2.000477	30.610180
7	43.000000	1.649683	2.163855	31.167928
8	44.000000	1.788489	2.336599	30.646502
9	45.000000	1.927791	2.519038	30.669656
10	46.000000	2.075759	2.711506	30.627235
11	47.000000	2.224970	2.914342	30.983444
12	48.000000	2.392998	3.127887	30.709959
13	49.000000	2.571337	3.352486	30.379097
14	50.000000	2.758713	3.588488	30.078368

	Strikes	Market Value	Model Value	Relative Error (%)
15	51.000000	2.946000	3.836244	30.218741
16	52.000000	3.153999	4.096106	29.870250
17	53.000000	3.372193	4.368427	29.542604
18	54.000000	3.609428	4.653560	28.927910
19	55.000000	3.858208	4.951856	28.346005
20	56.000000	4.114372	5.263664	27.933599
21	57.000000	4.382562	5.589330	27.535677
22	58.000000	4.670428	5.929191	26.951766
23	59.000000	4.966610	6.283581	26.516499
24	60.000000	5.273682	6.652822	26.151370
25	61.000000	5.610139	7.037226	25.437650
26	62.000000	5.956183	7.437090	24.863361
27	63.000000	6.322759	7.852698	24.197326
28	64.000000	6.710584	8.284311	23.451412
29	65.000000	7.117514	8.732174	22.685731
30	66.000000	7.532672	9.196506	22.088221
31	67.000000	7.987682	9.677498	21.155268
32	68.000000	8.062401	9.783134	21.342698
33	69.000000	7.544439	9.292317	23.167766
34	70.000000	7.053301	8.818551	25.027283
35	71.000000	6.595024	8.361895	26.790971
36	72.000000	6.146337	7.922370	28.895785
37	73.000000	5.727596	7.499953	30.944179
38	74.000000	5.338542	7.094581	32.893623
39	75.000000	4.958786	6.706147	35.237674
40	76.000000	4.610897	6.334496	37.380986
41	77.000000	4.282964	5.979433	39.609701
42	78.000000	3.974510	5.640718	41.922347
43	79.000000	3.698365	5.318067	43.795067
44	80.000000	3.429545	5.011157	46.117240
45	81.000000	3.183419	4.719625	48.256511
46	82.000000	2.956429	4.443076	50.285215
47	83.000000	2.740365	4.181080	52.573819

Heston Model 1, WTI

Crude Oil parameters
output

Value

	Value
v0	0.403305
kappa	0.910291
theta	0.992253
sigma	-0.562644
rho	0.000000
avgError	30.688544

No handles with labels found to put in legend.

Heston Model 2, WTI Crude Oil calibration results

	Strikes	Market Value	Model Value	Relative Error (%)
0	36.000000	0.958086	1.026927	7.185248
1	37.000000	1.037192	1.120166	7.999892
2	38.000000	1.125773	1.219387	8.315483
3	39.000000	1.214808	1.324851	9.058454
4	40.000000	1.313419	1.436829	9.396061
5	41.000000	1.422697	1.555603	9.341805
6	42.000000	1.531639	1.681467	9.782146
7	43.000000	1.649683	1.814726	10.004535
8	44.000000	1.788489	1.955700	9.349272
9	45.000000	1.927791	2.104722	9.177934
10	46.000000	2.075759	2.262142	8.979025
11	47.000000	2.224970	2.428322	9.139572
12	48.000000	2.392998	2.603646	8.802682
13	49.000000	2.571337	2.788512	8.445993
14	50.000000	2.758713	2.983339	8.142434
15	51.000000	2.946000	3.188565	8.233719
16	52.000000	3.153999	3.404651	7.947132
17	53.000000	3.372193	3.632078	7.706693
18	54.000000	3.609428	3.871350	7.256612
19	55.000000	3.858208	4.122996	6.862959
20	56.000000	4.114372	4.387566	6.639987
21	57.000000	4.382562	4.665638	6.459139
22	58.000000	4.670428	4.957810	6.153208
23	59.000000	4.966610	5.264705	6.001981
24	60.000000	5.273682	5.586968	5.940558
25	61.000000	5.610139	5.925263	5.617047
26	62.000000	5.956183	6.280268	5.441144

	Strikes	Market Value	Model Value	Relative Error (%)
27	63.000000	6.322759	6.652674	5.217892
28	64.000000	6.710584	7.043177	4.956237
29	65.000000	7.117514	7.452471	4.706088
30	66.000000	7.532672	7.881236	4.627360
31	67.000000	7.987682	8.330131	4.287214
32	68.000000	8.062401	8.407597	4.281554
33	69.000000	7.544439	7.892971	4.619718
34	70.000000	7.053301	7.400161	4.917696
35	71.000000	6.595024	6.929579	5.072842
36	72.000000	6.146337	6.481532	5.453577
37	73.000000	5.727596	6.056207	5.737340
38	74.000000	5.338542	5.653655	5.902615
39	75.000000	4.958786	5.273783	6.352299
40	76.000000	4.610897	4.916345	6.624483
41	77.000000	4.282964	4.580948	6.957428
42	78.000000	3.974510	4.267052	7.360451
43	79.000000	3.698365	3.973984	7.452447
44	80.000000	3.429545	3.700956	7.913910
45	81.000000	3.183419	3.447084	8.282467
46	82.000000	2.956429	3.211411	8.624644
47	83.000000	2.740365	2.992929	9.216412

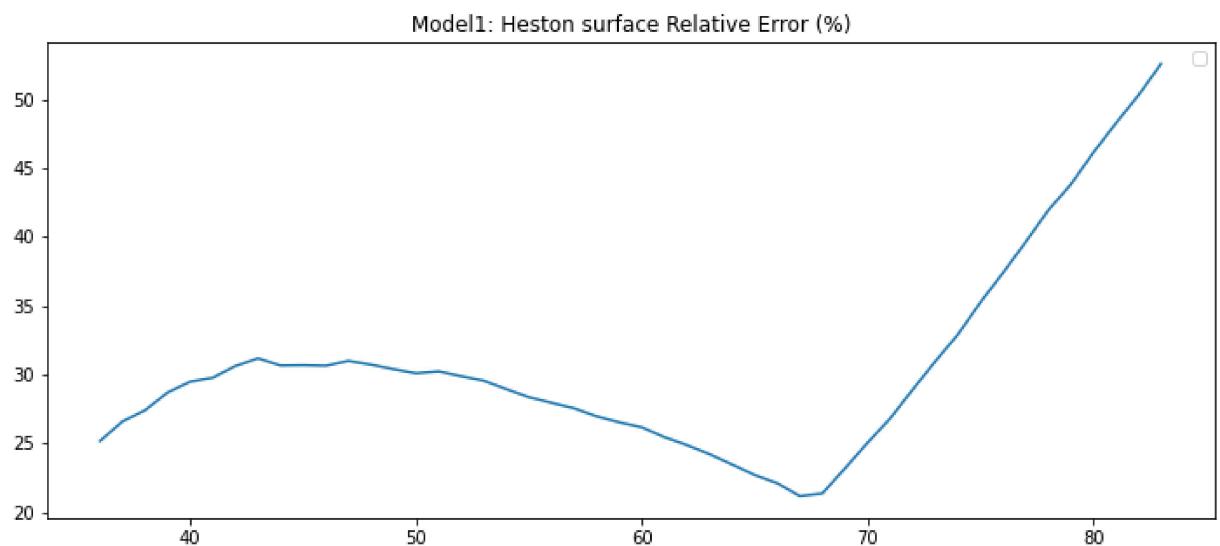
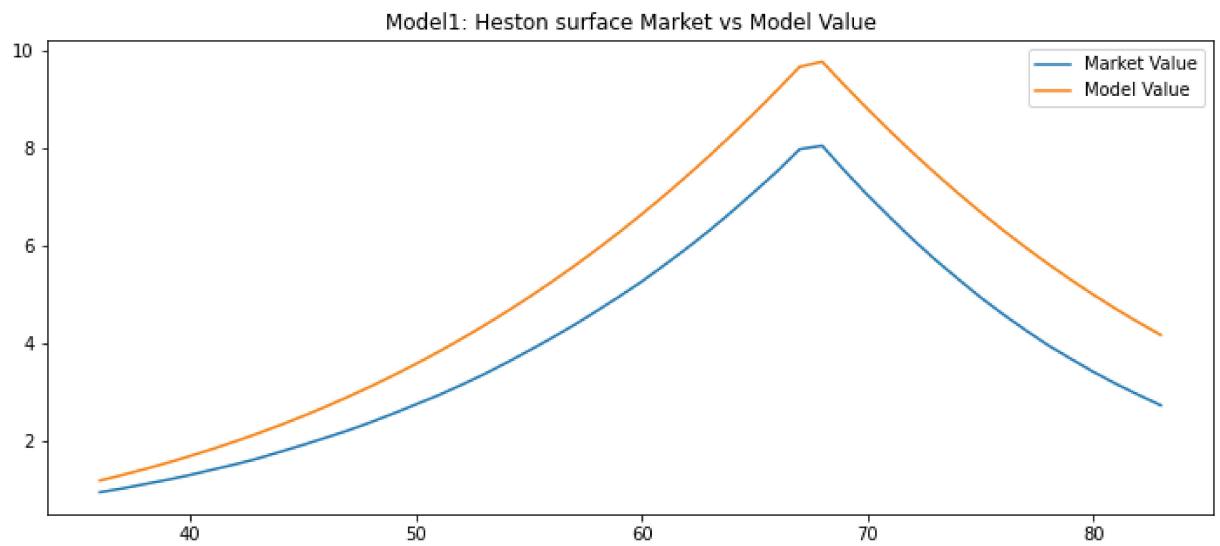
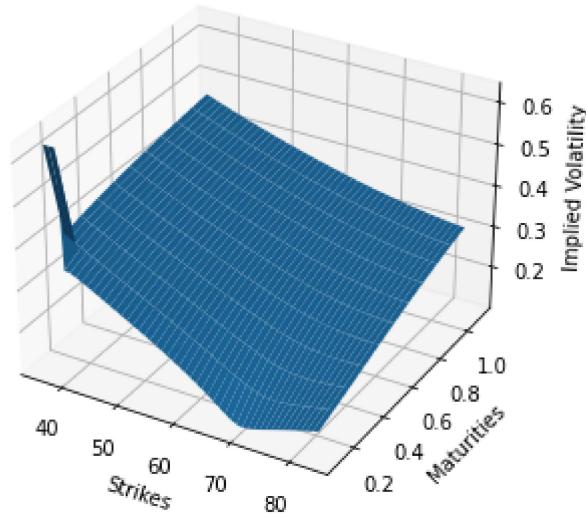
Heston Model 2, WTI

Crude Oil parameters
output

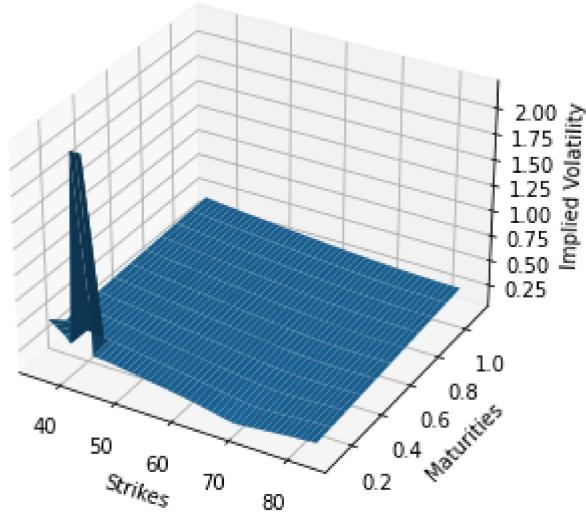
	Value
v0	4.924425
kappa	0.046090
theta	0.980092
sigma	-0.549255
rho	0.000000
avgError	7.123862

No handles with labels found to put in legend.

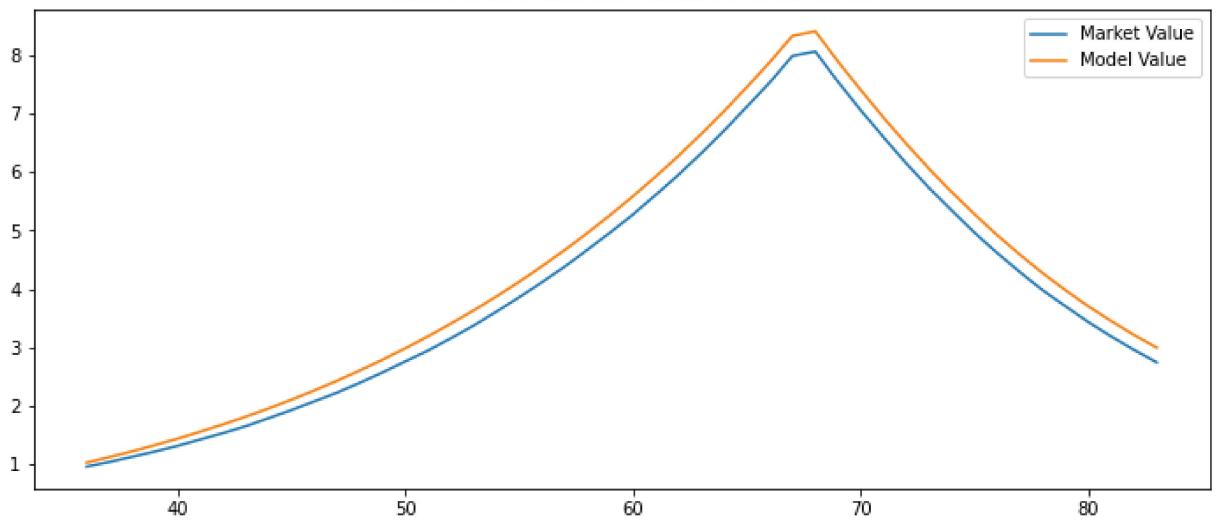
Heston Volatility Surface for Heston Model 1, WTI Crude Oil



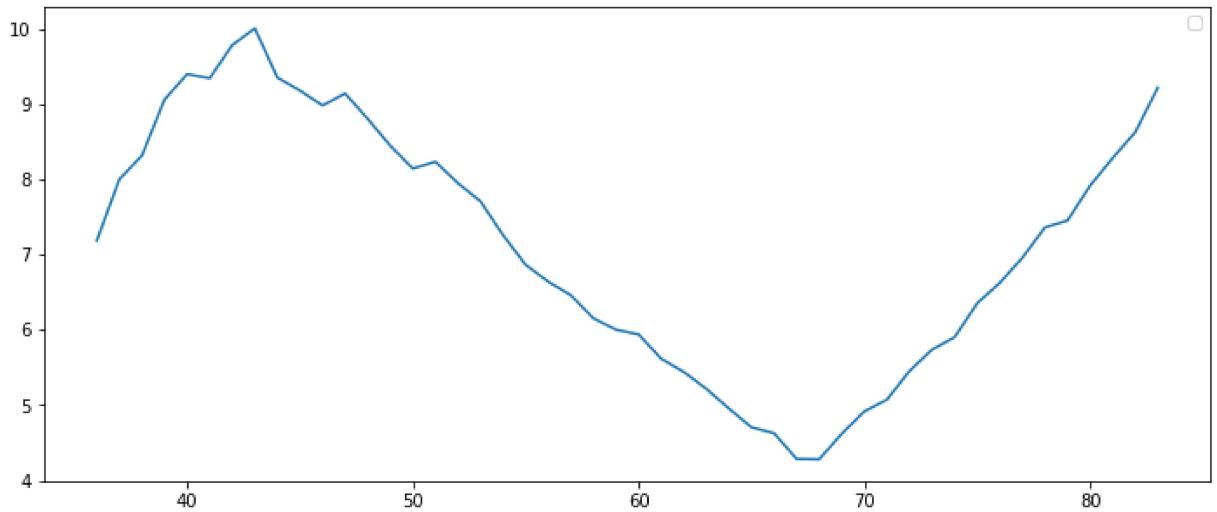
Heston Volatility Surface for Heston Model 2, WTI Crude Oil



Model1: Heston surface Market vs Model Value



Model1: Heston surface Relative Error (%)



In [7]:

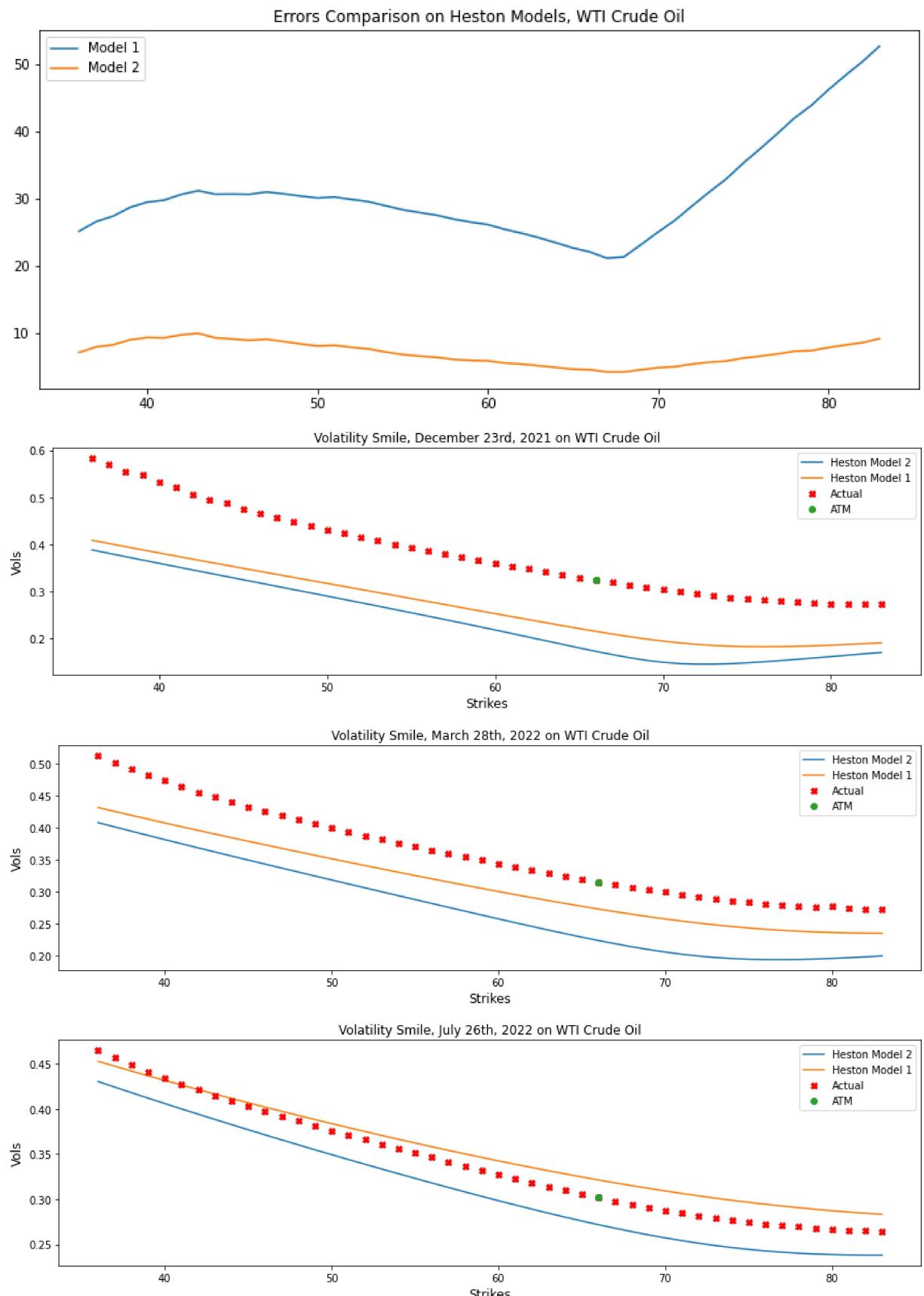
```
# Relative error comparison
plt.figure(figsize=plot_size)
plt.plot(hestonModel1.strks, hestonModel1.relativeError, label="Model 1")
plt.plot(hestonModel2.strks, hestonModel2.relativeError, label="Model 2")
plt.title("Errors Comparison on Heston Models, {}".format(data_label));
plt.legend()

# Volatility smiles comparison
tenors = [dates[round((len(dates)-1) * x)] for x in (.2, .5, .75, 1)]
```

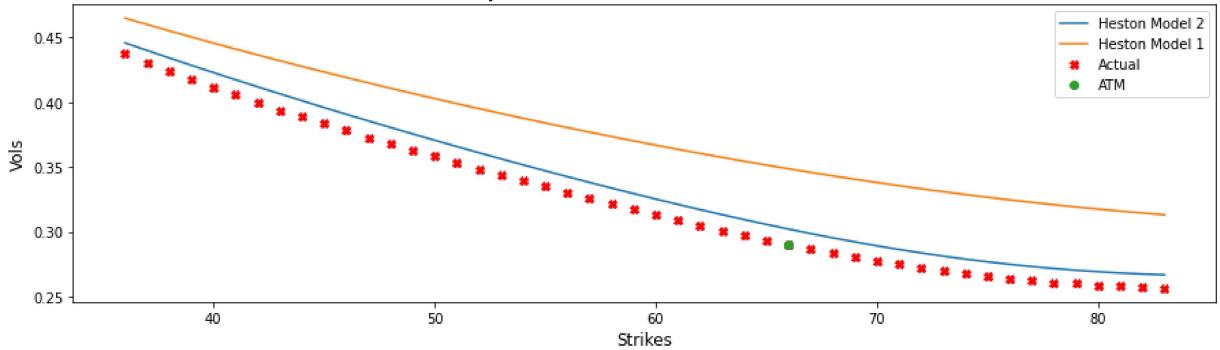
```

for tenor in tenors:
    l = [
        ([hestonModel2.heston_vol_surface.blackVol(tenor, s) for s in strikes], "Hes2"),
        ([hestonModel1.heston_vol_surface.blackVol(tenor, s) for s in strikes], "Hes1")
    ]
    plot_smile(tenor, l, market=True)

```



Volatility Smile, October 26th, 2022 on WTI Crude Oil



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.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.expiryTime)
    elif self.shift != 0:
        return ql.shiftedSabrVolatility(strike, self.forward_price, self.expiryTime, self.shift)
    else:
        return ql.sabrVolatility(strike, self.forward_price, self.expiryTime, alpha, beta, nu, rho)

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_forward):
        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, self.nu, self.rho, self.vol_matrix, self.vol_diff_matrix = (
            ql.Matrix(len(self.strikes)), len(dates)), ql.Matrix(len(self.strikes)), len(self.strikes), len(dates))

        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, marketVols=v, method=self.method, fwd=self.fwd, zero_rho=self.zero_rho)
            volSABR.initialize()

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

```

```

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

        self.errors.append(volSABR.error)

        smile = volSABR.newVols

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

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

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

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

# Backbone modelling for SABR
def SABR_backbone_plot(beta=1, bounds=None, shift=0, strikes=strikes, fixes=(.95, 1,
    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

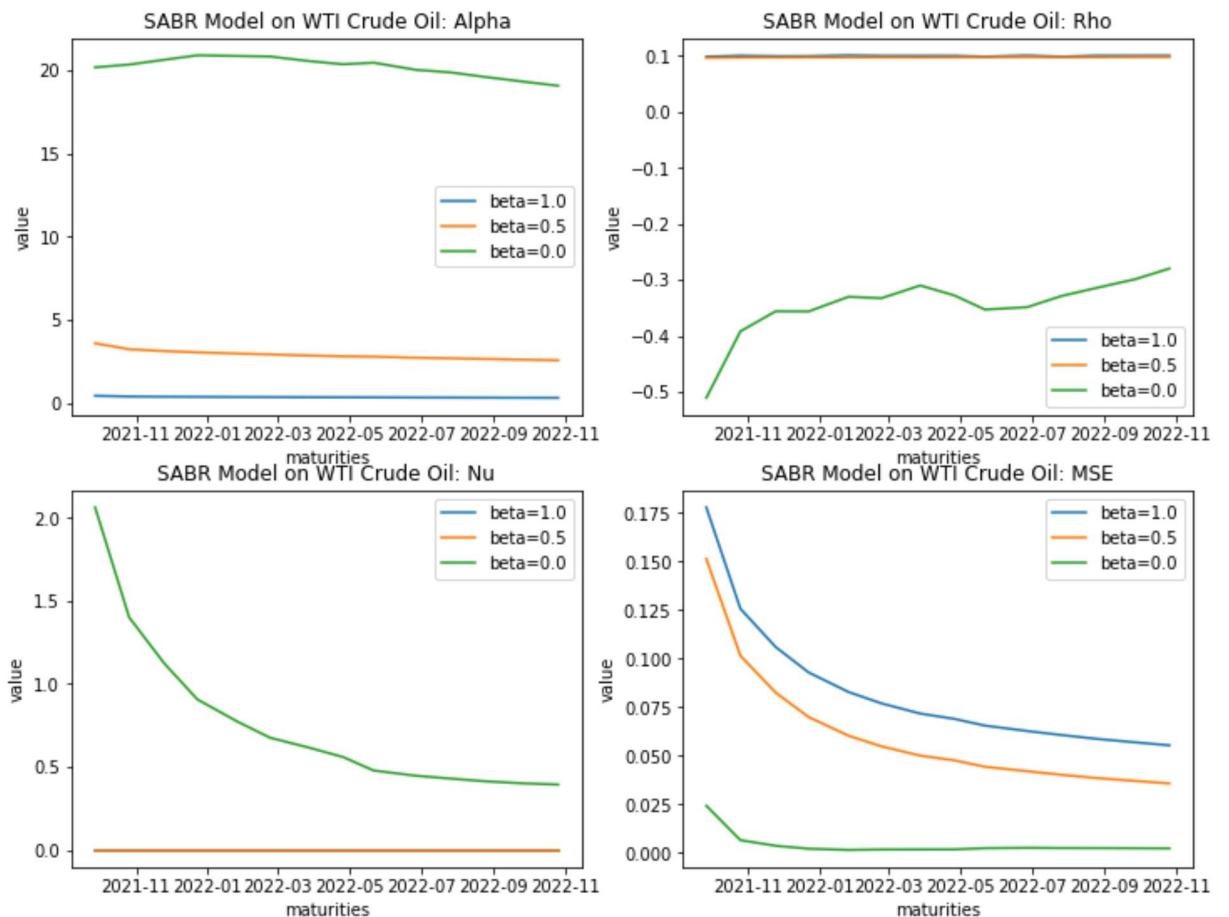
strks = strikes
SABR_beta1 = SABRVolatilitySurface(beta=1, shift=0, strks=strks, label="SABR, beta=1")
SABR_beta5 = SABRVolatilitySurface(beta=.5, shift=0, strks=strks, label="SABR, beta=.5")
SABR_beta0 = SABRVolatilitySurface(beta=.0, shift=0, strks=strks, label="Normal SABR")

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

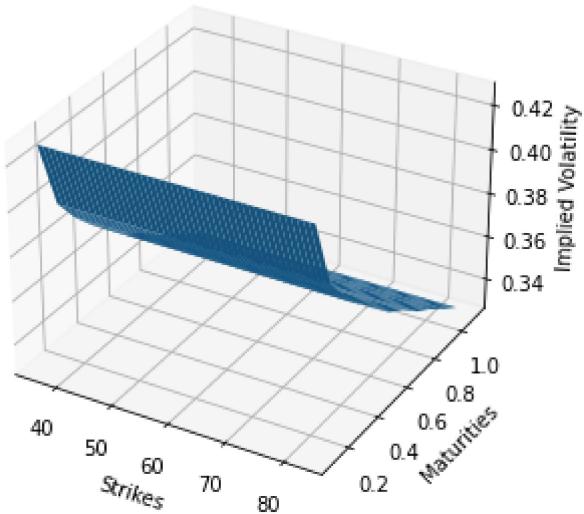
```

C:\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

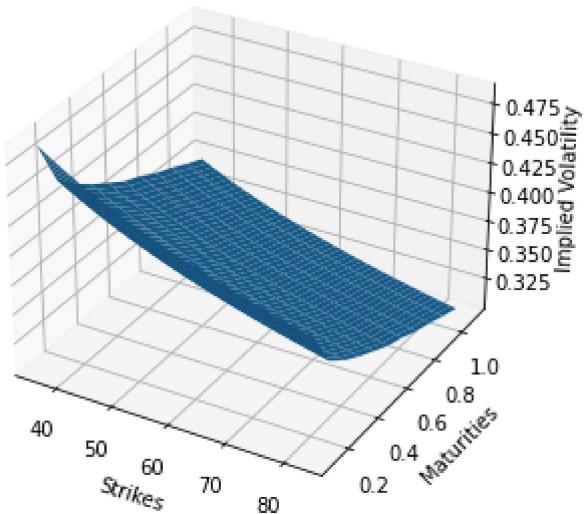
warnings.warn("Values in x were outside bounds during a "



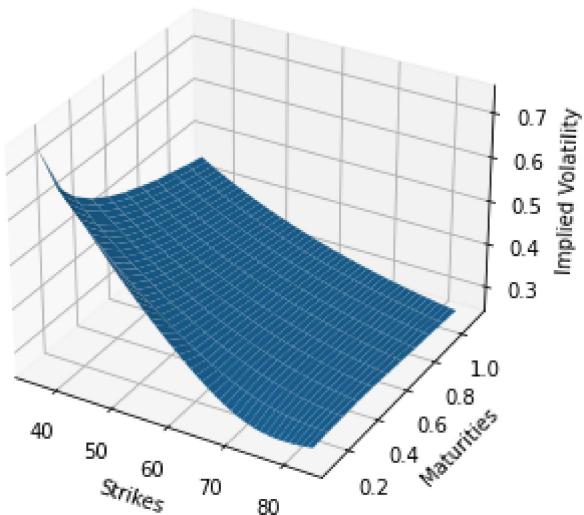
SABR, beta=1, WTI Crude Oil

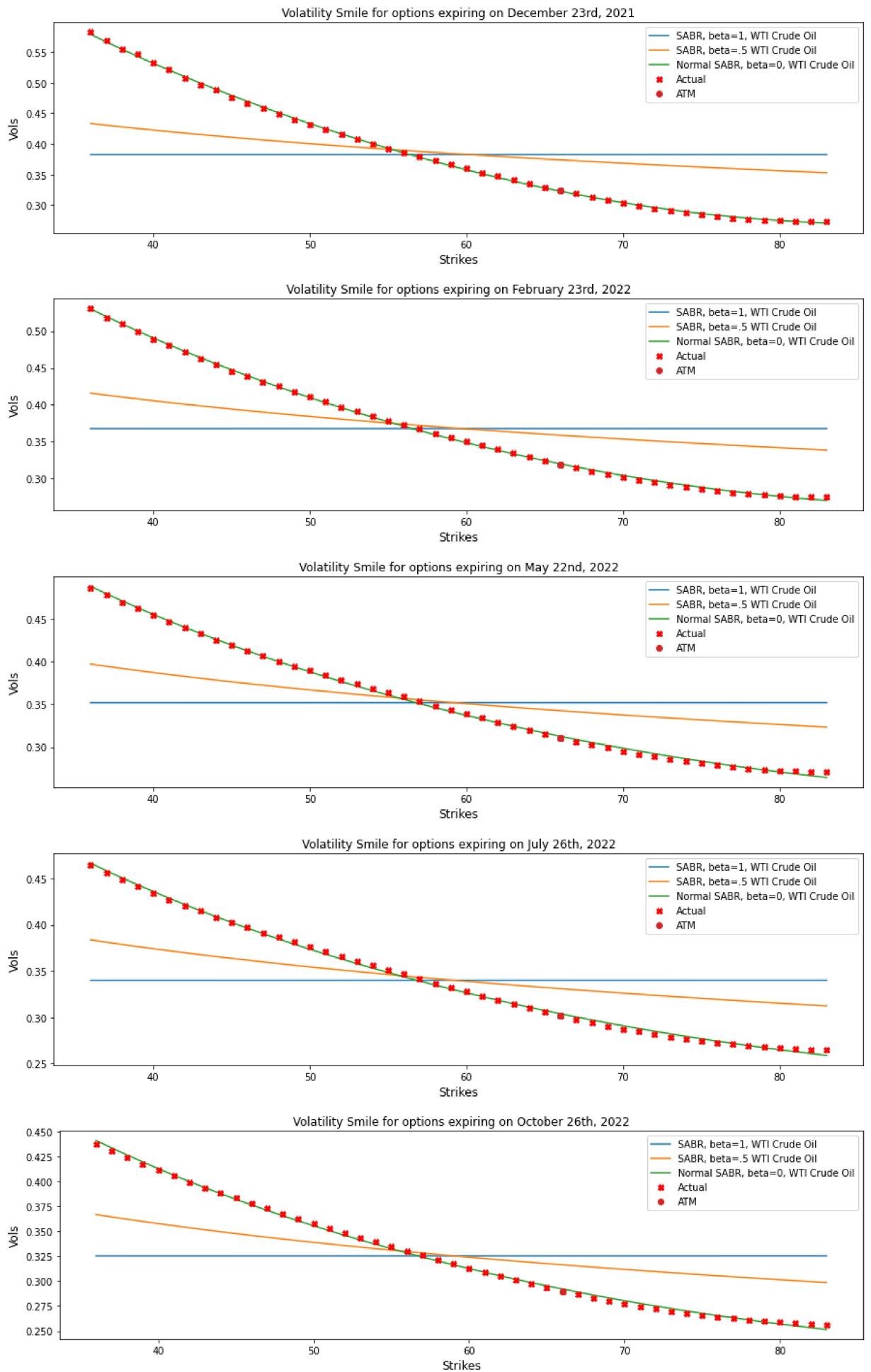


SABR, beta=.5 WTI Crude Oil



Normal SABR, beta=0, WTI Crude Oil





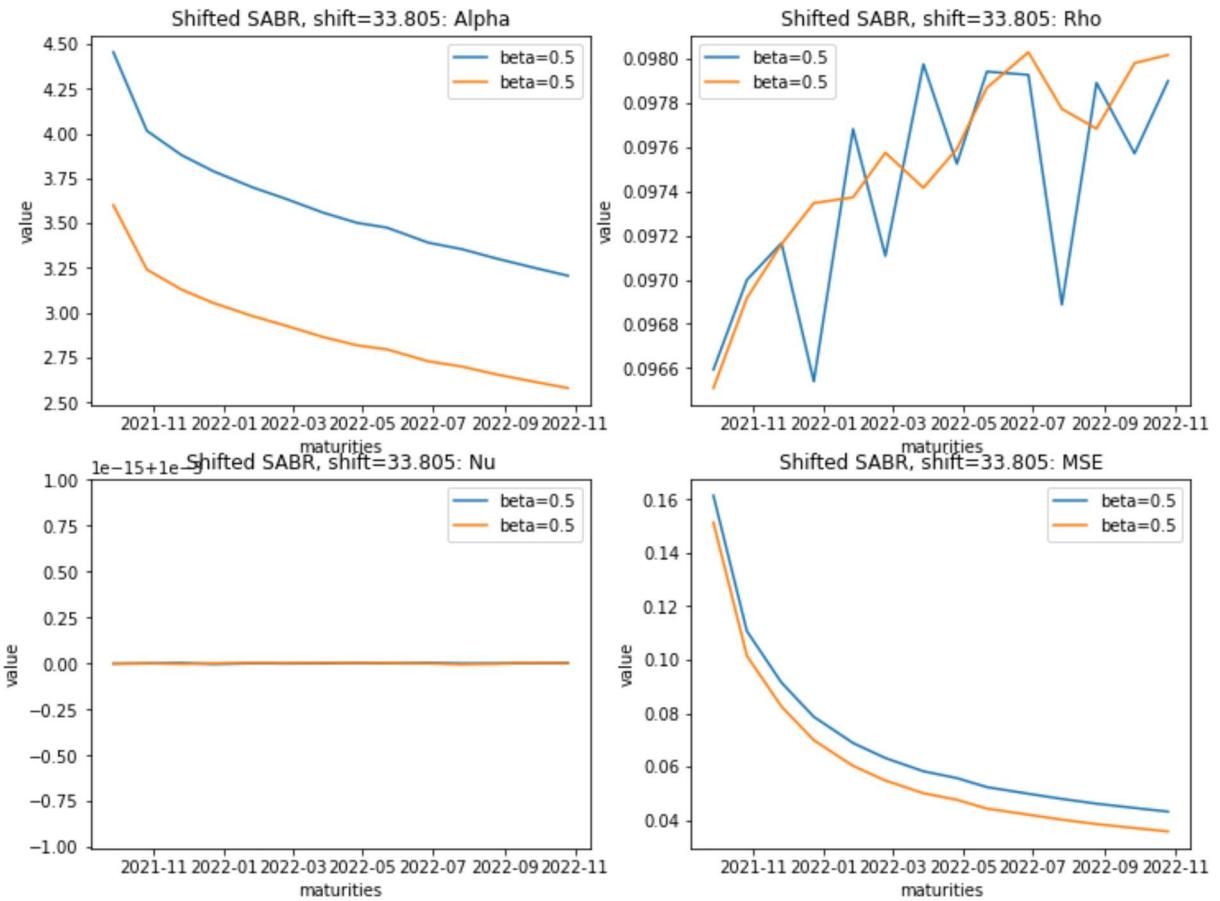
In [10]: `# Shifted SABR Volatility model`

```

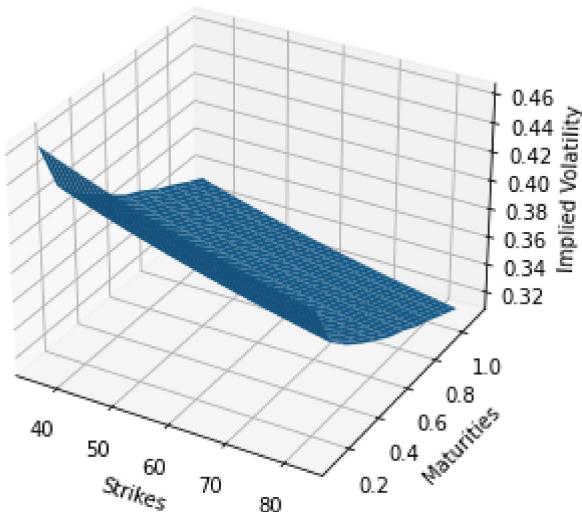
shft = .50 * current_price
shiftedSABR_beta1 = SABRVolatilitySurface(beta=1, shift=shft, strks=strikes, label="S")
shiftedSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=shft, strks=strikes, label="S")
shiftedSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=shft, strks=strikes, label="S")

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

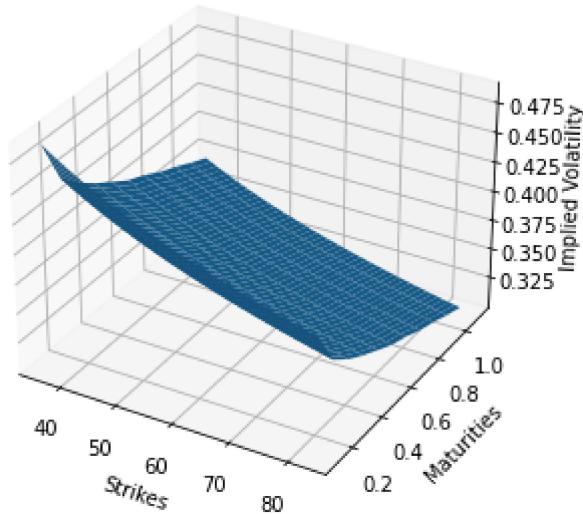
```



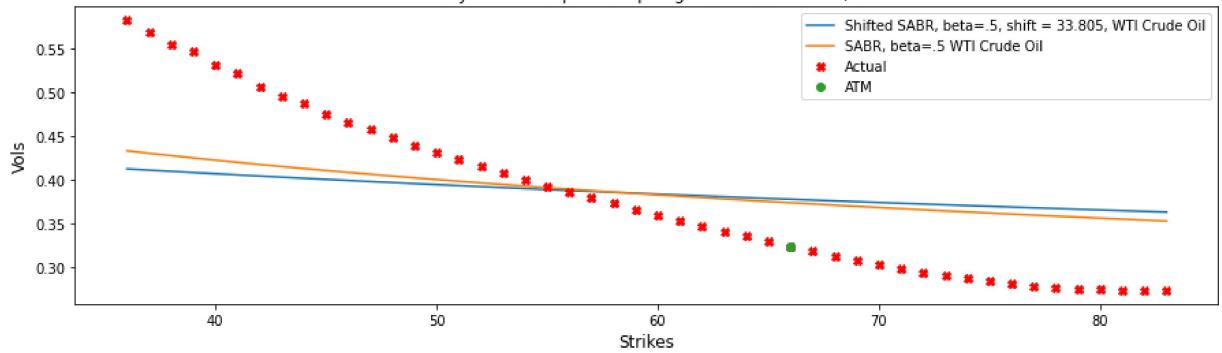
Shifted SABR, beta=.5, shift = 33.805, WTI Crude Oil



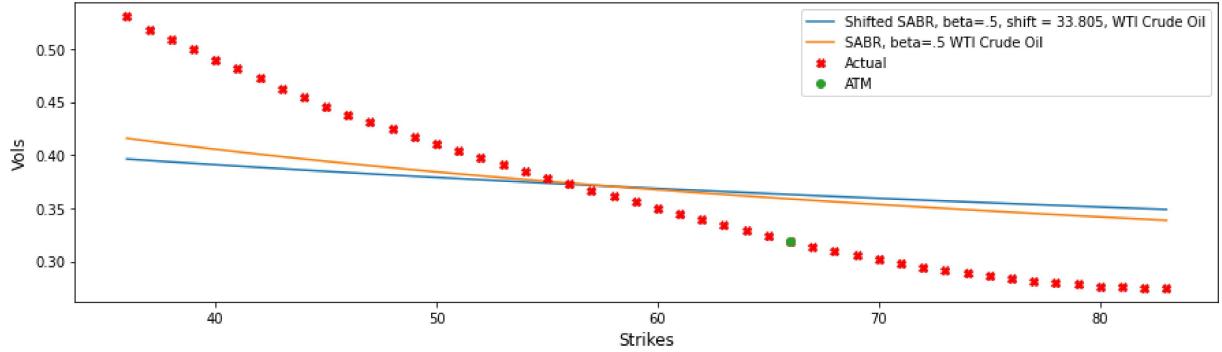
SABR, beta=.5 WTI Crude Oil



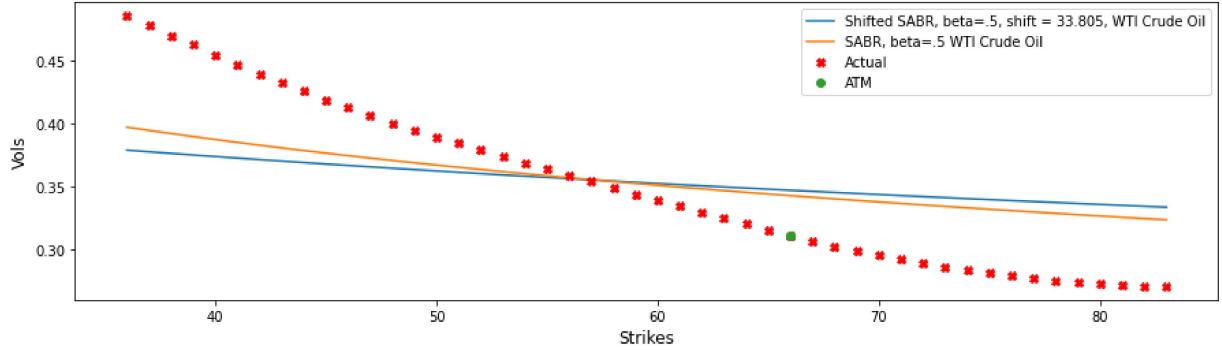
Volatility Smile for options expiring on December 23rd, 2021



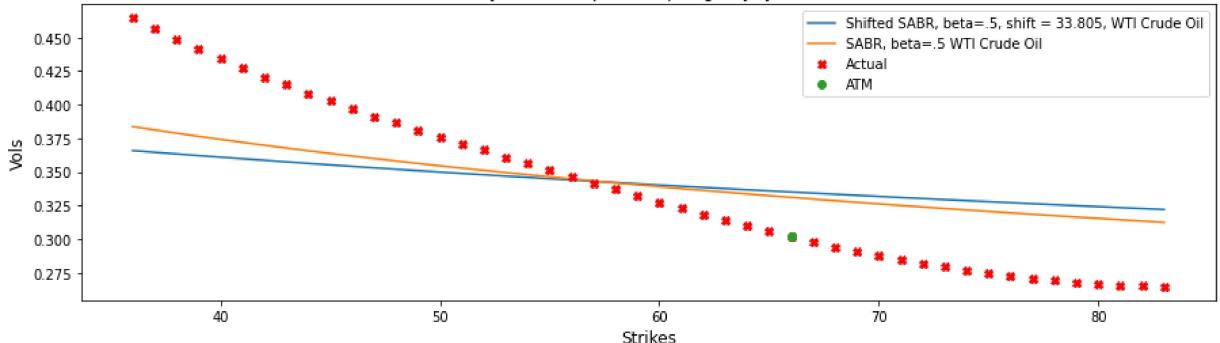
Volatility Smile for options expiring on February 23rd, 2022



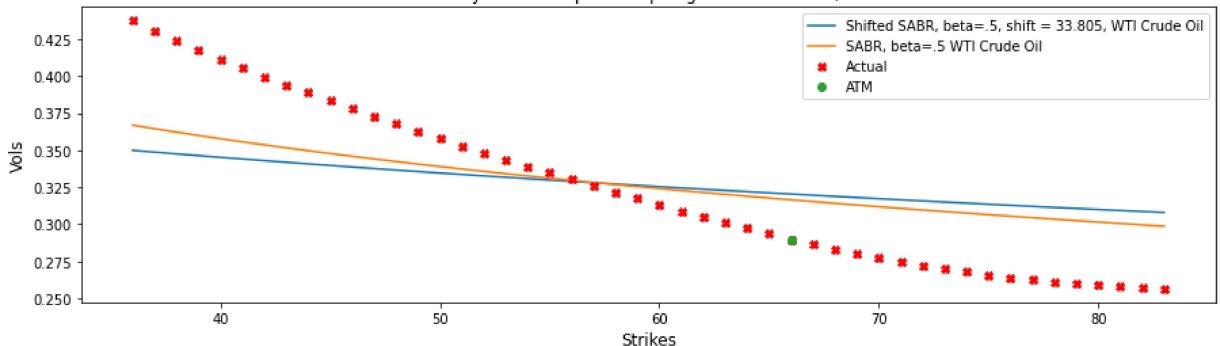
Volatility Smile for options expiring on May 22nd, 2022



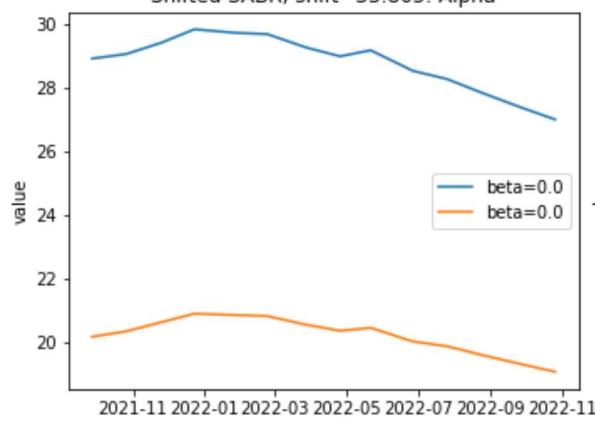
Volatility Smile for options expiring on July 26th, 2022



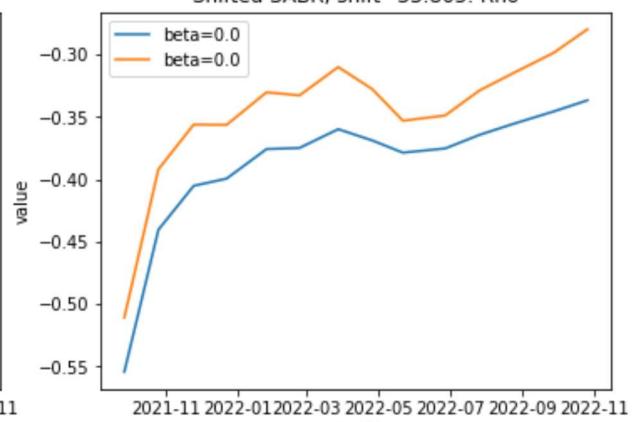
Volatility Smile for options expiring on October 26th, 2022



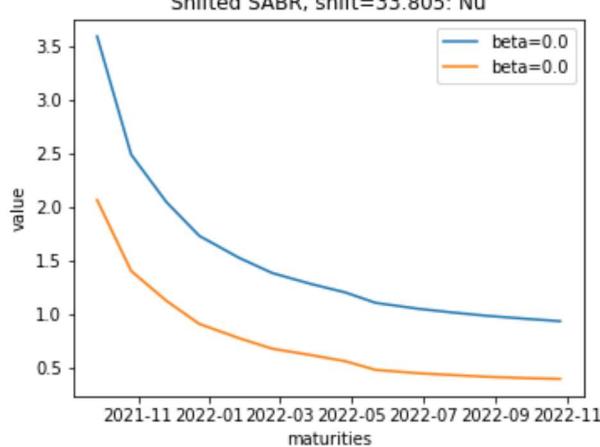
Shifted SABR, shift=33.805: Alpha



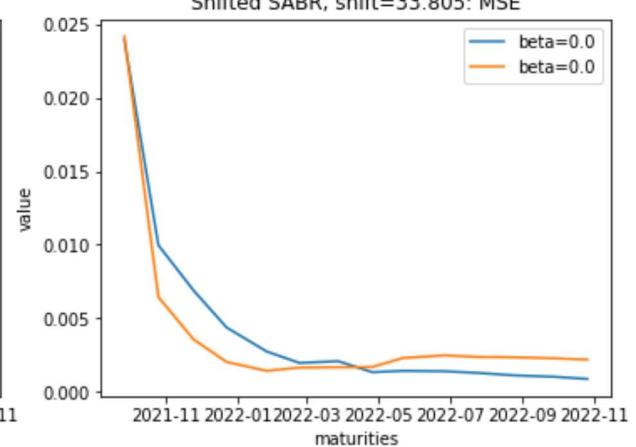
Shifted SABR, shift=33.805: Rho



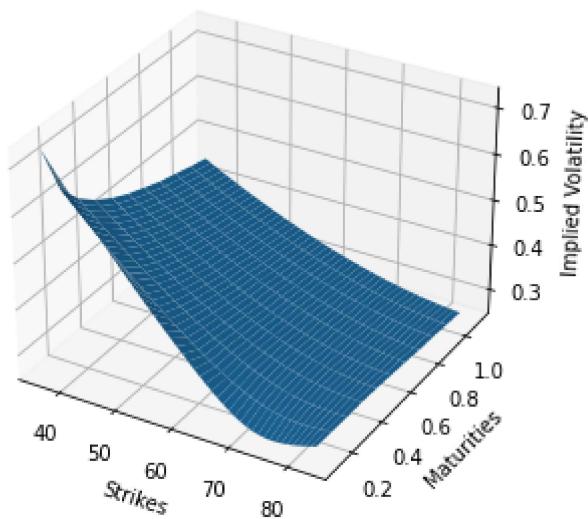
Shifted SABR, shift=33.805: Nu



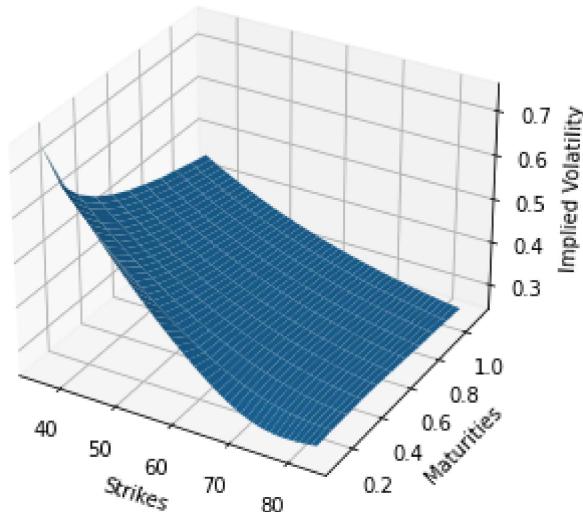
Shifted SABR, shift=33.805: MSE



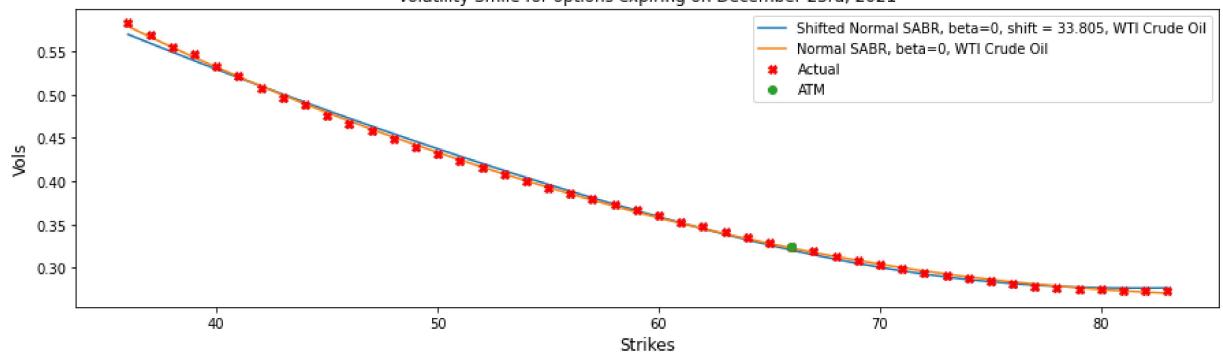
Shifted Normal SABR, beta=0, shift = 33.805, WTI Crude Oil



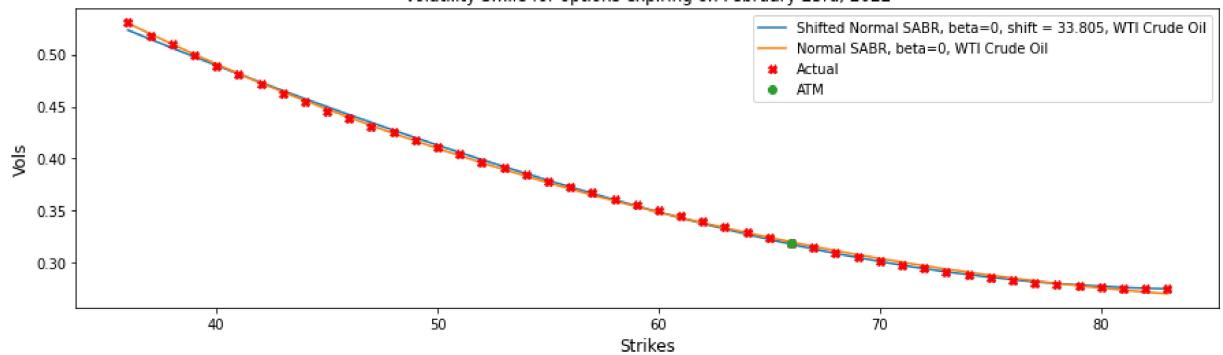
Normal SABR, beta=0, WTI Crude Oil

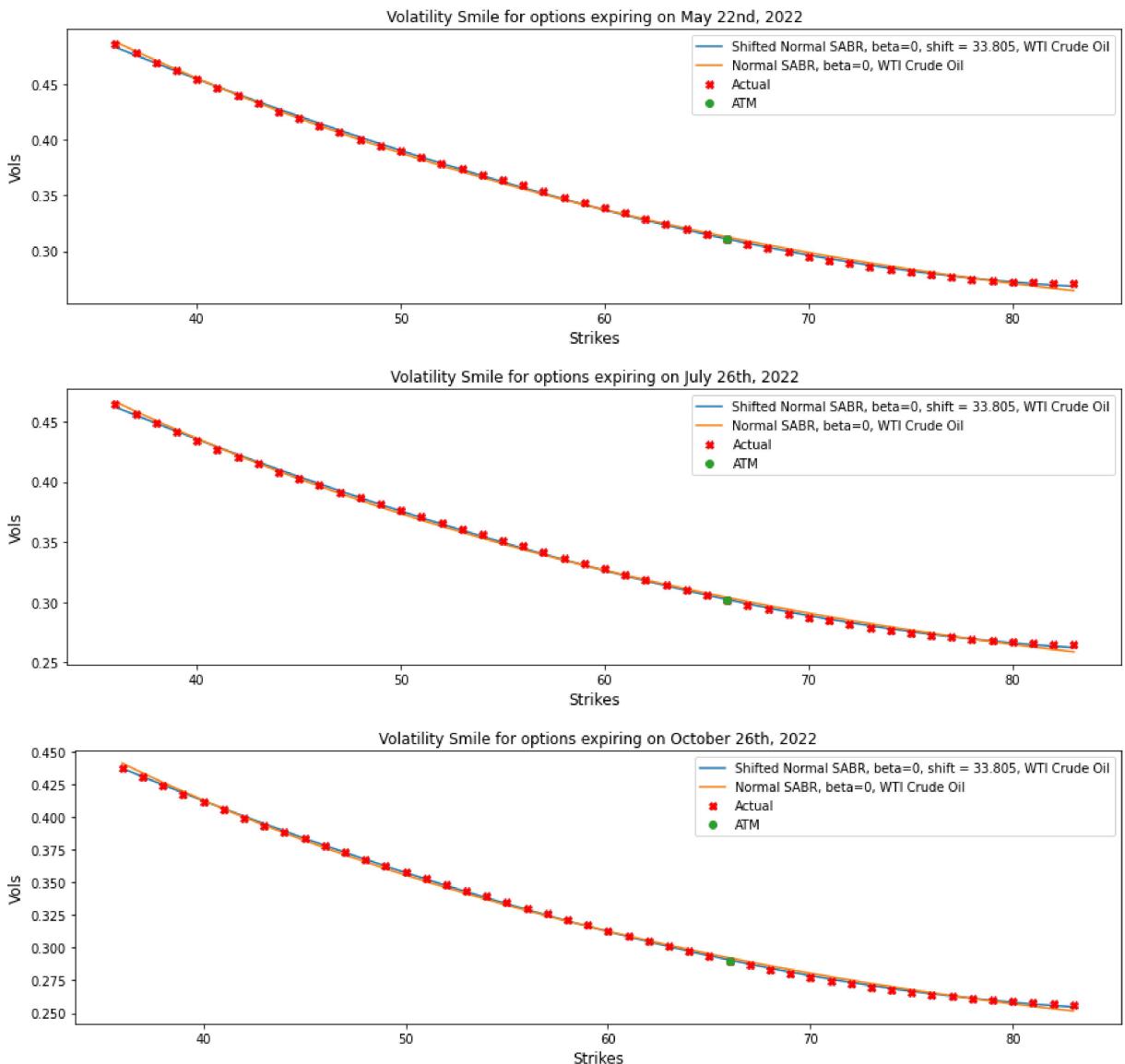


Volatility Smile for options expiring on December 23rd, 2021



Volatility Smile for options expiring on February 23rd, 2022



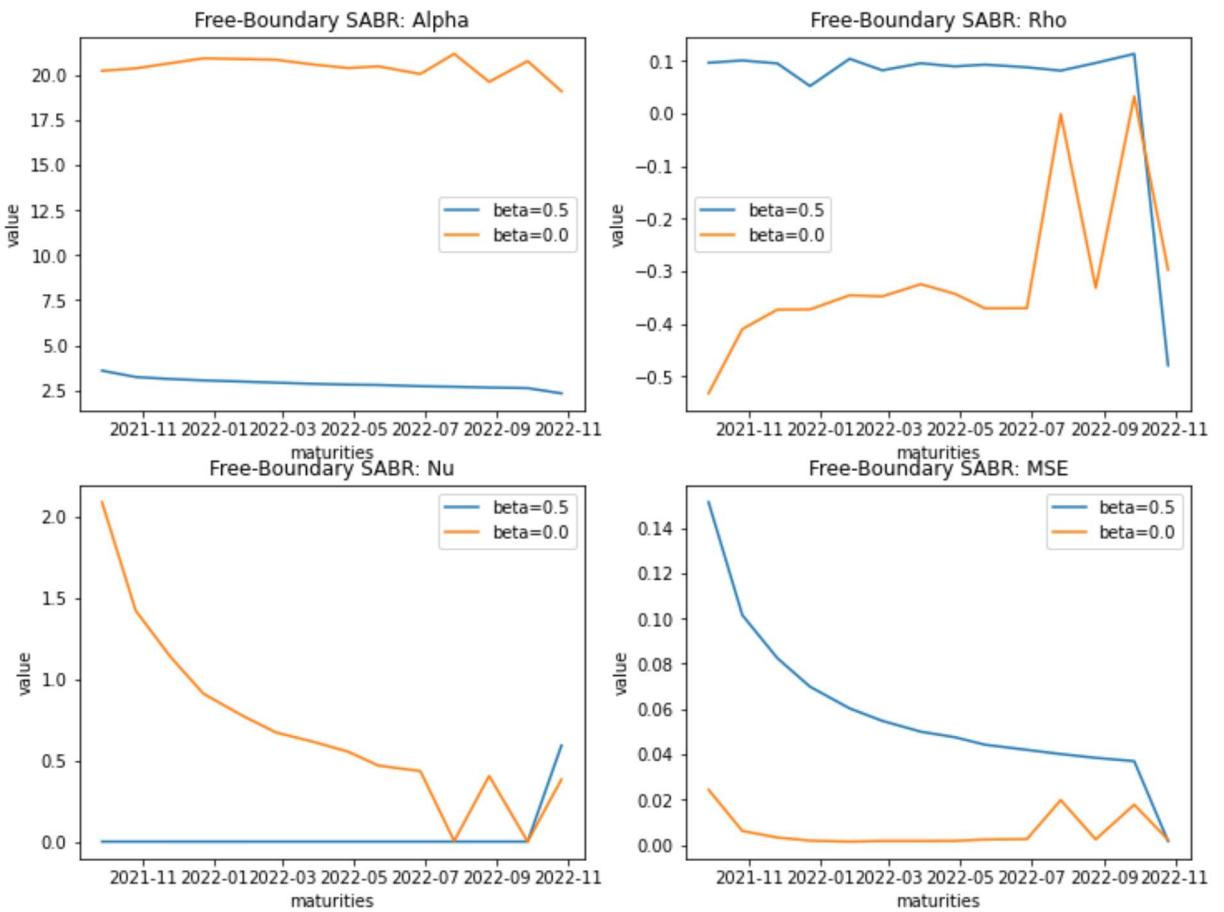


In [11]:

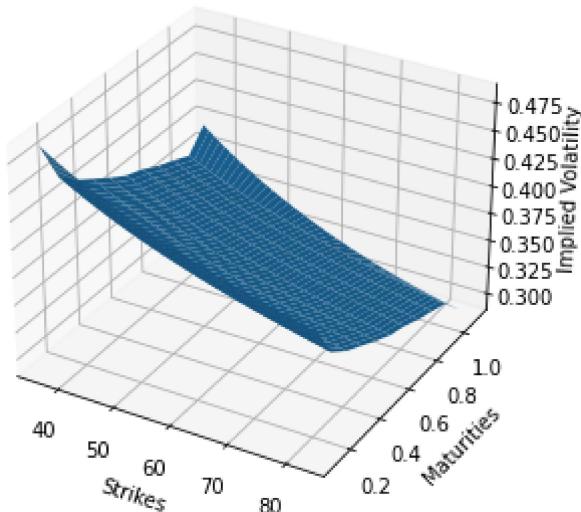
```
# Free-Boundary SABR Volatility model

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

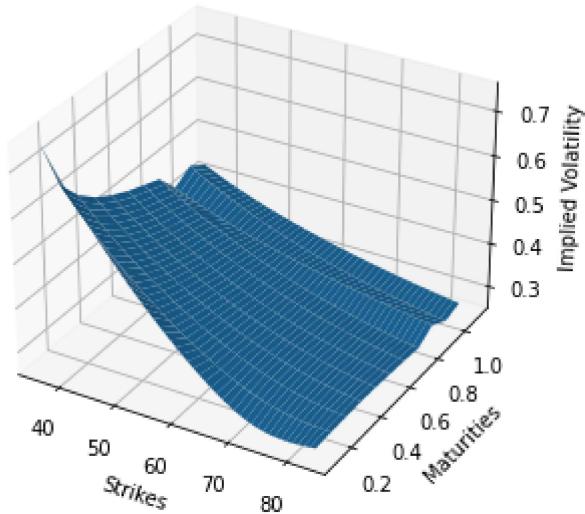
SABRComparison([freeSABR_beta5, freeSABR_beta0], title="Free-Boundary SABR")
```



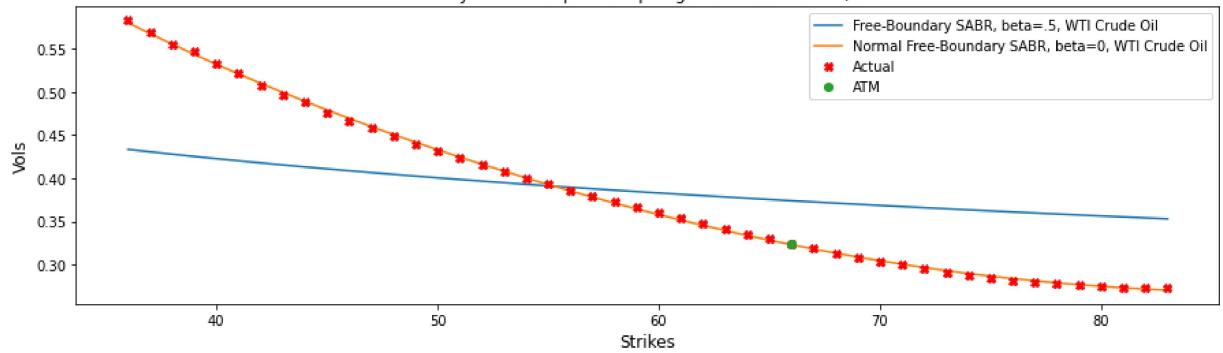
Free-Boundary SABR, beta=.5, WTI Crude Oil



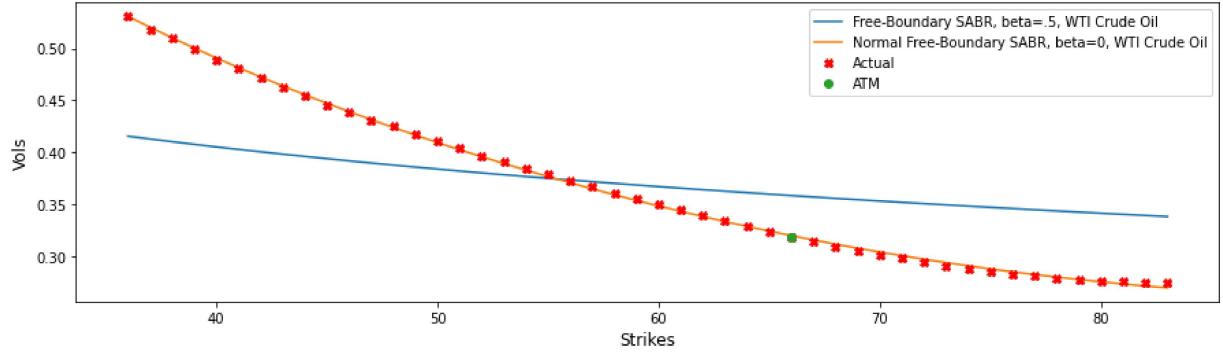
Normal Free-Boundary SABR, beta=0, WTI Crude Oil



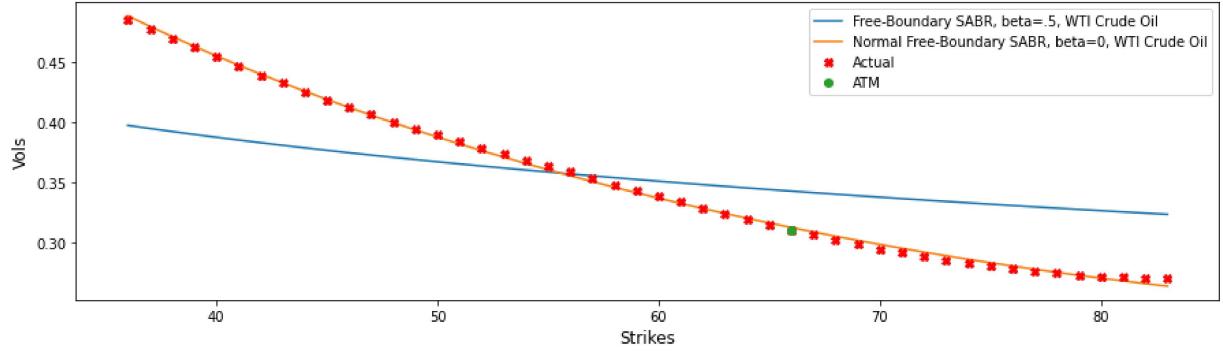
Volatility Smile for options expiring on December 23rd, 2021

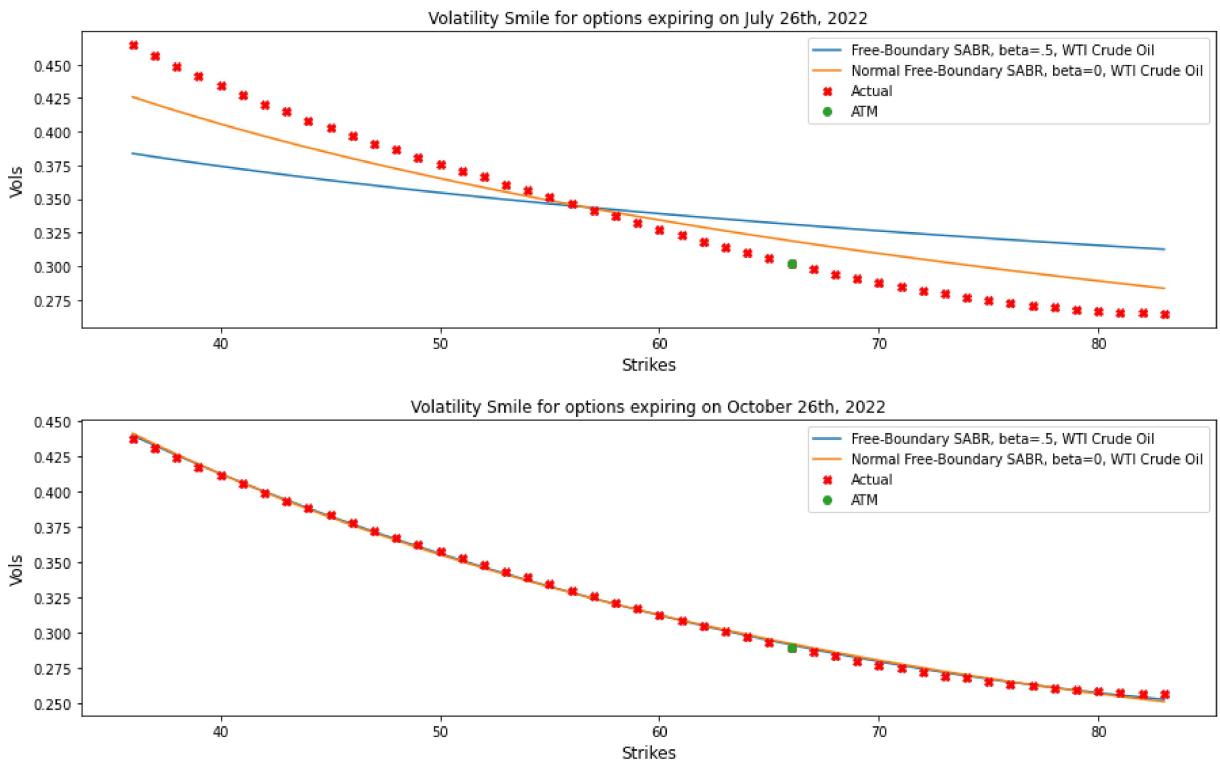


Volatility Smile for options expiring on February 23rd, 2022



Volatility Smile for options expiring on May 22nd, 2022





In [12]:

```
# Mixed SABR

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

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 "

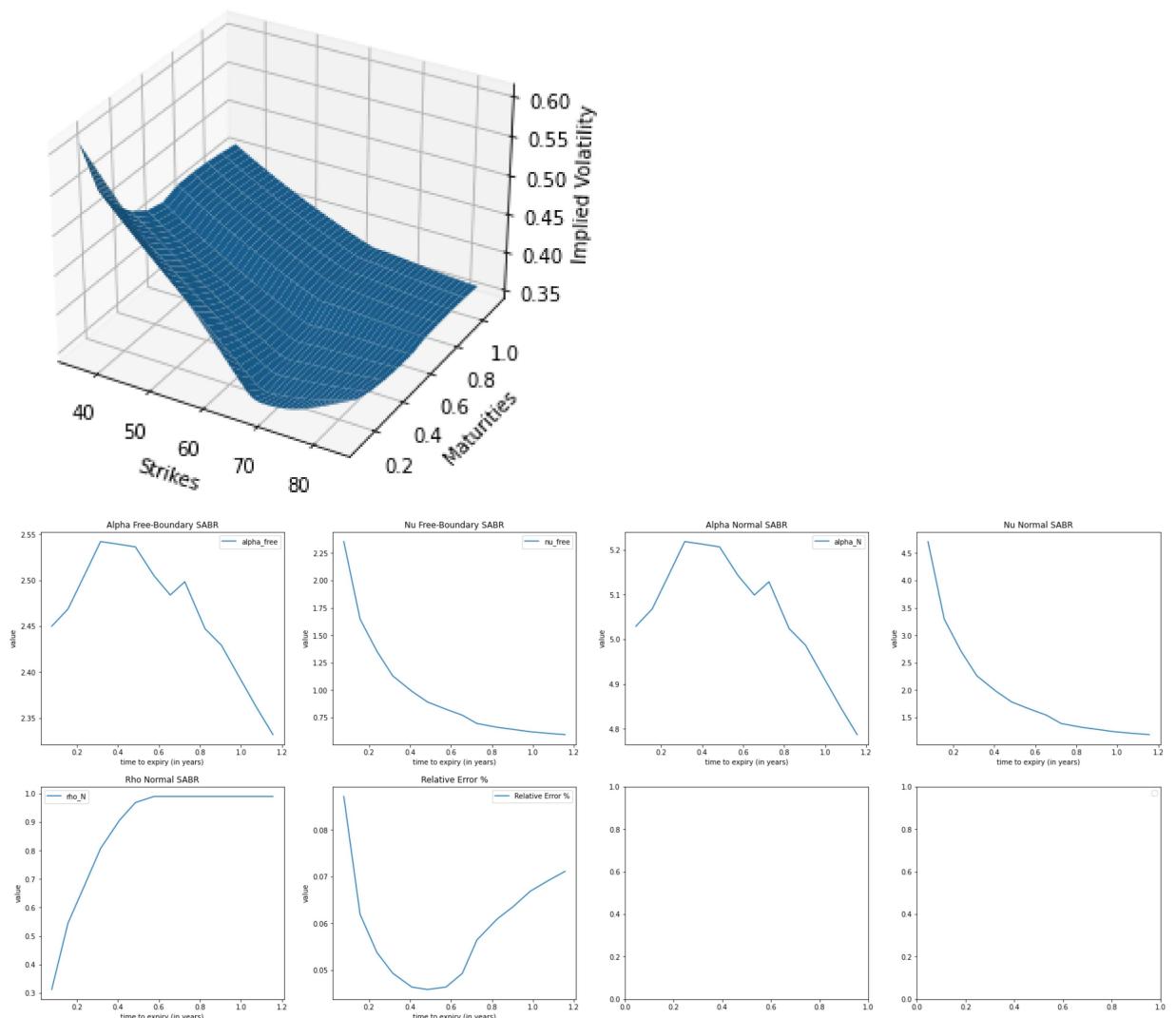
	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSI
September 27th, 2021	2.449958	0.5	2.354421		0.0	2.449958	0.0	4.708842	0.312404
October 26th, 2021	2.468698	0.5	1.646774		0.0	2.468698	0.0	3.293547	0.545531
November 25th, 2021	2.506417	0.5	1.353955		0.0	2.506417	0.0	2.707910	0.680170
December 23rd, 2021	2.542062	0.5	1.130503		0.0	2.542062	0.0	2.261007	0.807568
January 26th, 2022	2.538988	0.5	0.989837		0.0	2.538988	0.0	1.979675	0.907097
February 23rd, 2022	2.536233	0.5	0.892065		0.0	2.536233	0.0	1.784129	0.968444
March 28th, 2022	2.505012	0.5	0.825410		0.0	2.505012	0.0	1.650820	0.990000
April 26th, 2022	2.483907	0.5	0.771357		0.0	2.483907	0.0	1.542715	0.990000
May 22nd, 2022	2.498308	0.5	0.695785		0.0	2.498308	0.0	1.391571	0.990000

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSI
June 27th, 2022	2.447246	0.5	0.661259	0.0	2.447246	0.0	1.322519	0.990000	0.060931
July 26th, 2022	2.429392	0.5	0.641539	0.0	2.429392	0.0	1.283077	0.990000	0.063634
August 25th, 2022	2.396771	0.5	0.620344	0.0	2.396771	0.0	1.240688	0.990000	0.066878
September 27th, 2022	2.361294	0.5	0.605586	0.0	2.361294	0.0	1.211172	0.990000	0.069220
October 26th, 2022	2.332039	0.5	0.593914	0.0	2.332039	0.0	1.187828	0.990000	0.071131

◀ ▶

No handles with labels found to put in legend.

Mixture SABR, WTI Crude Oil



In [13]:

```
# VOLATILITY SMILES ERRORS COMPARISON
```

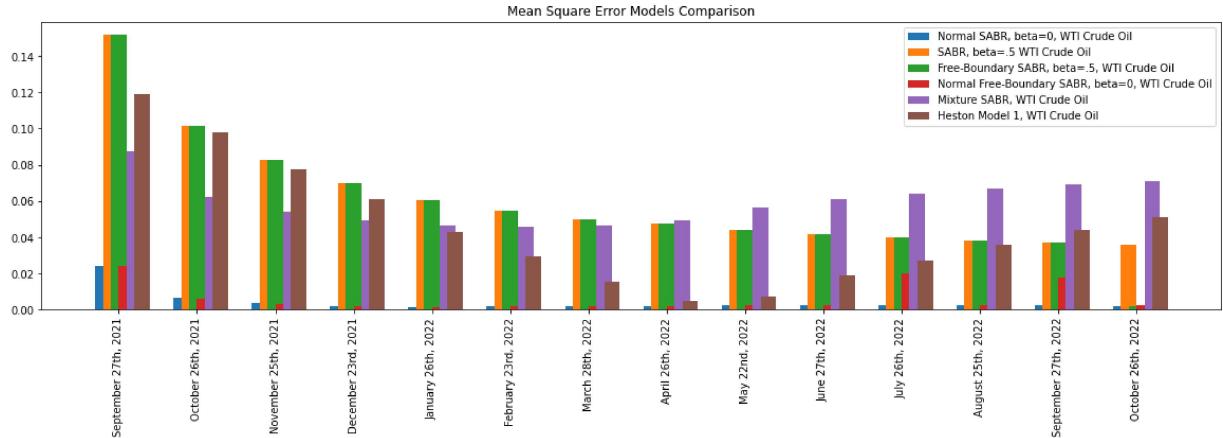
```
fig = plt.figure(figsize=(20, 5), )
width = .20
plt.bar(np.arange(len(maturities)) - 2*width/2, SABR_beta0.errors, label=SABR_beta0.
plt.bar(np.arange(len(maturities)) - width/2, SABR_beta5.errors, label=SABR_beta5.la
plt.bar(np.arange(len(maturities)), freeSABR_beta5.errors, label=freeSABR_beta5.lab
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 0x203c66c3070>



In [14]:

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

```

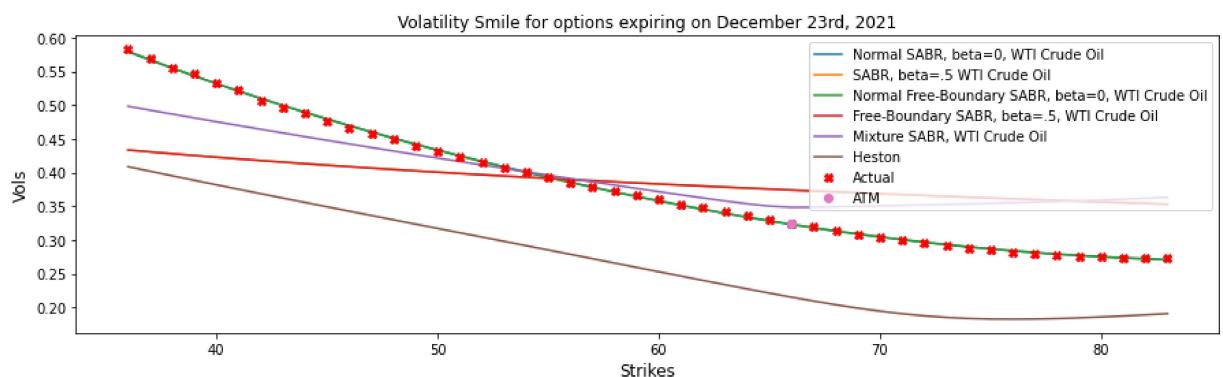
models = (
    SABR_beta0,
    SABR_beta5,
    freeSABR_beta0,
    freeSABR_beta5,
    mixtureSABR
)

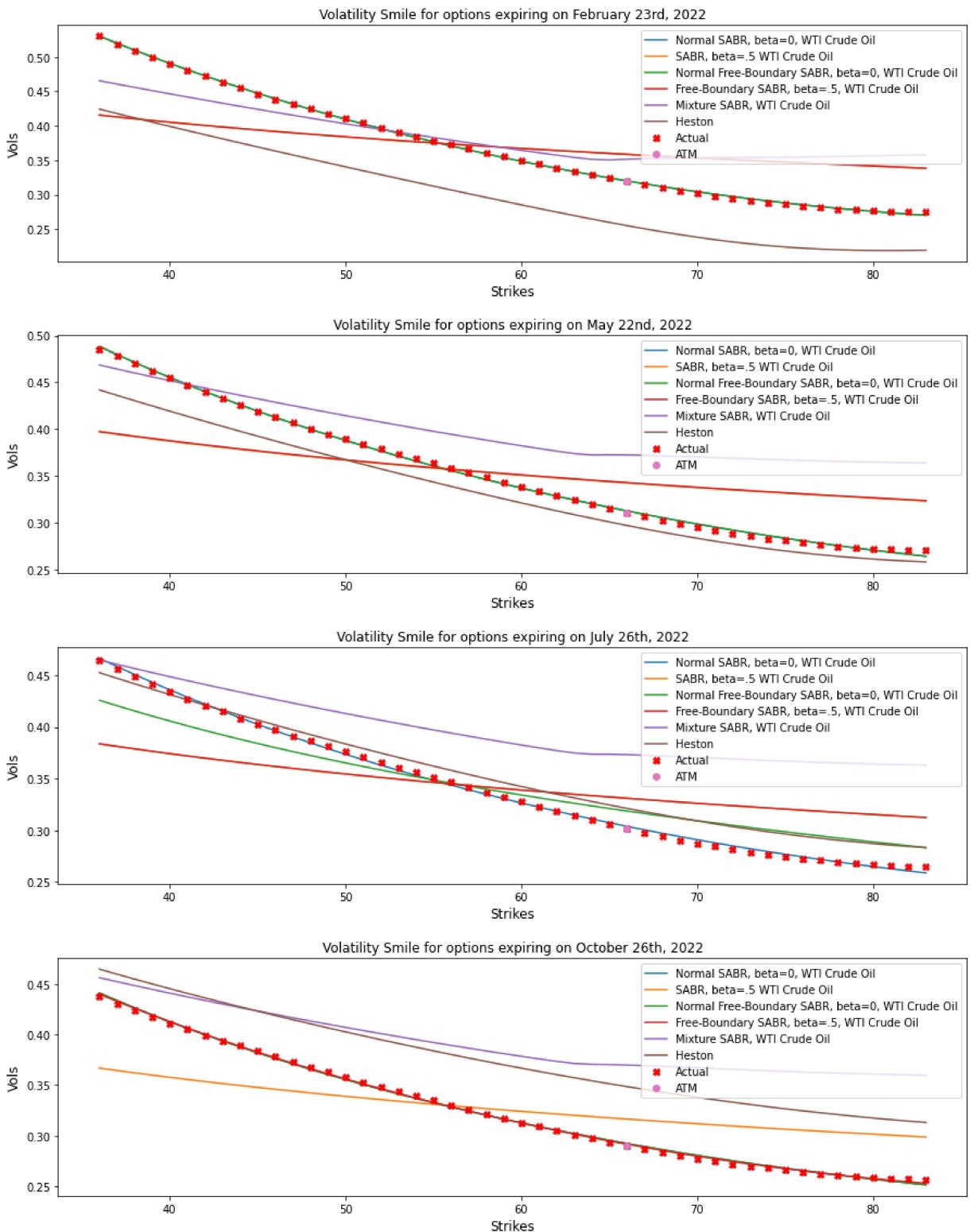
errors_data = pd.DataFrame([np.mean(m.errors) for m in models], index=[m.label for m in models])
display(errors_data)

smiles_comparison(models, heston_models=[hestonModel1])

```

Error	
Normal SABR, beta=0, WTI Crude Oil	0.004042
SABR, beta=.5 WTI Crude Oil	0.061090
Normal Free-Boundary SABR, beta=0, WTI Crude Oil	0.006413
Free-Boundary SABR, beta=.5, WTI Crude Oil	0.058661
Mixture SABR, WTI Crude Oil	0.059175





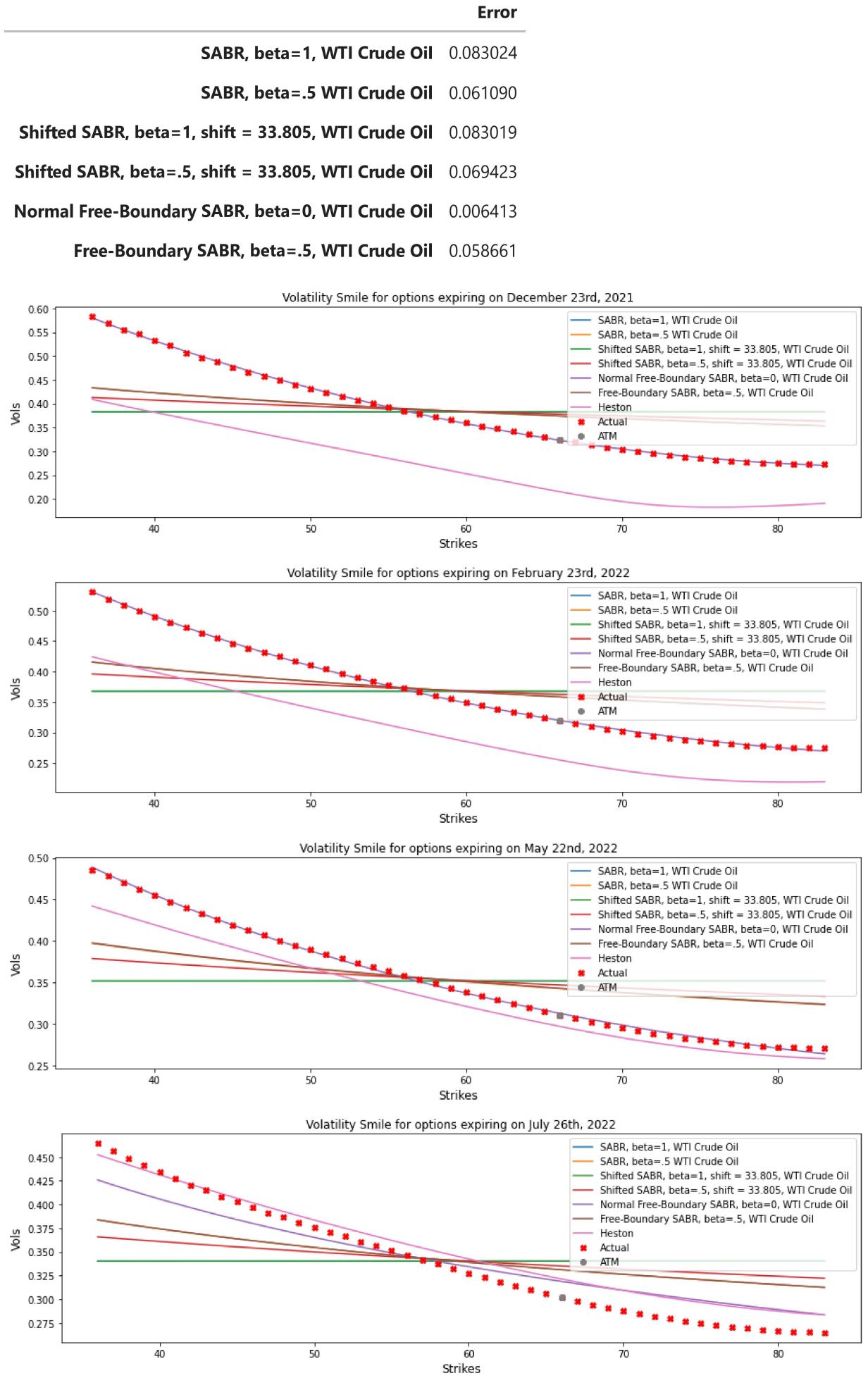
In [15]:

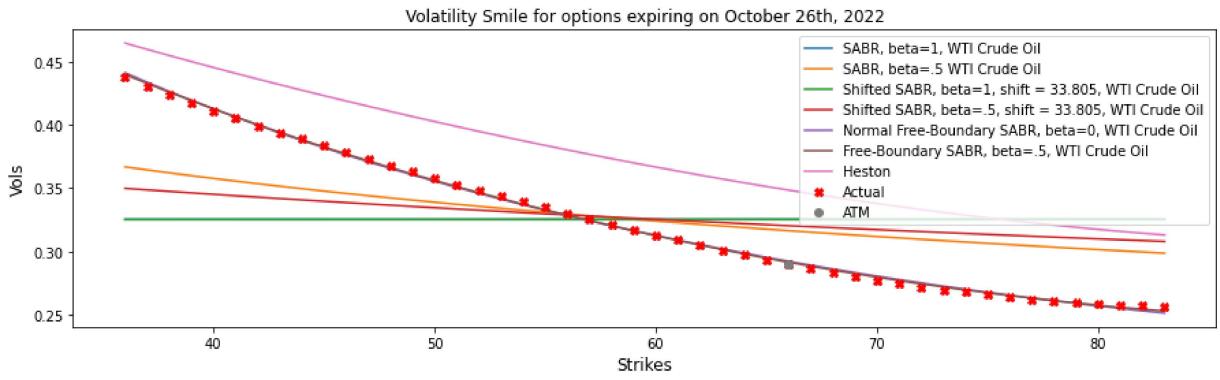
```
# Volatility Smiles Comparisons 2 (Final)
```

```
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 in models])
display(errors_data)
```

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





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

```

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

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

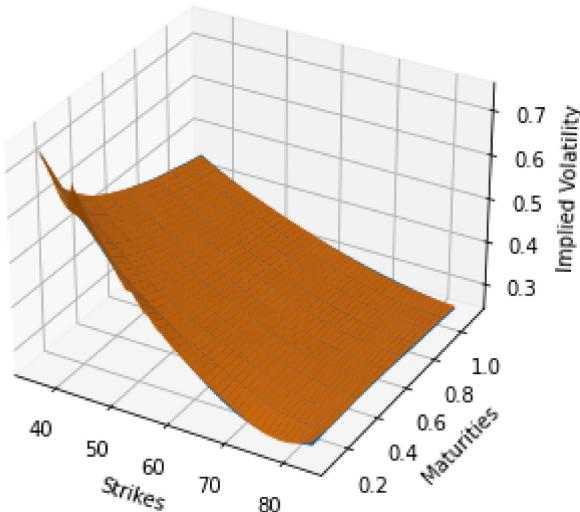
title = "normal SABR Volatility Surface on {} VS IV surface\n{} to {}\\nBeta = 0".format(data, today, plot_vol_surface([SABR_beta0.vol_surface, black_var_surface], title=title)

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

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

```

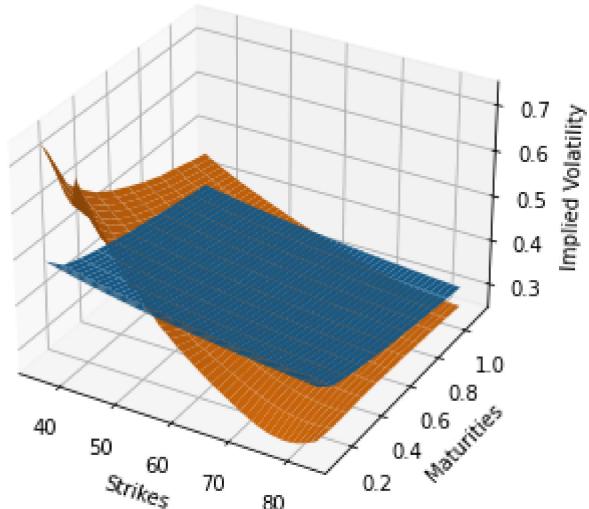
normal SABR Volatility Surface on OIL VS IV surface
August 30th, 2021 to October 26th, 2022
Beta = 1



normal SABR Volatility Surface on OIL VS IV surface

August 30th, 2021 to October 26th, 2022

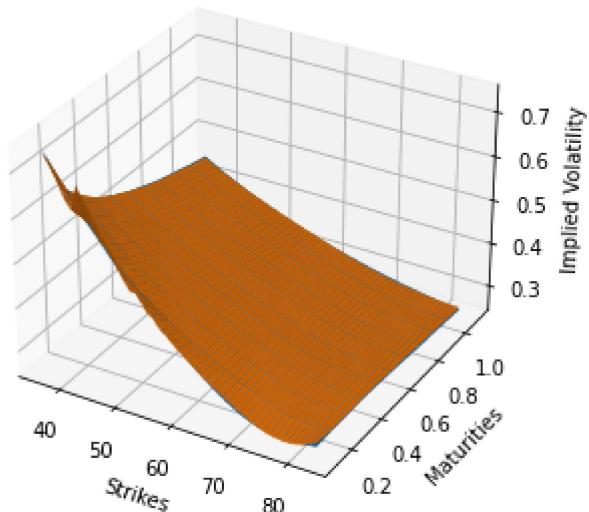
Beta = 0.5



normal SABR Volatility Surface on OIL VS IV surface

August 30th, 2021 to October 26th, 2022

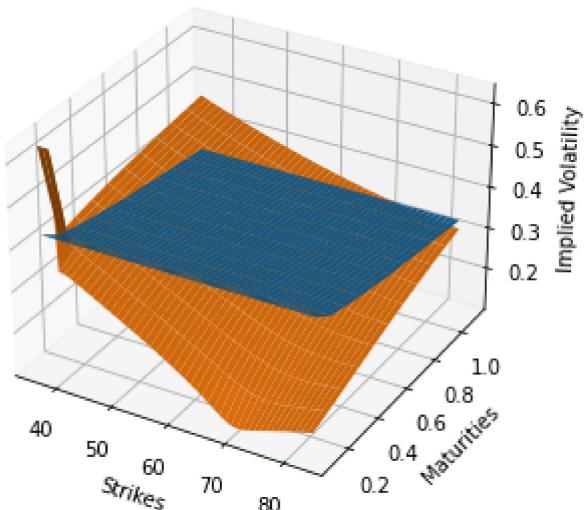
Beta = 0



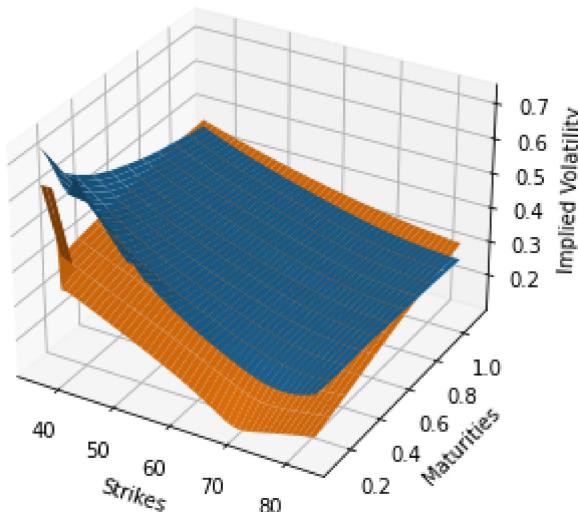
SABR Volatility Surface on OIL VS Heston surface

August 30th, 2021 to October 26th, 2022

Beta = 1

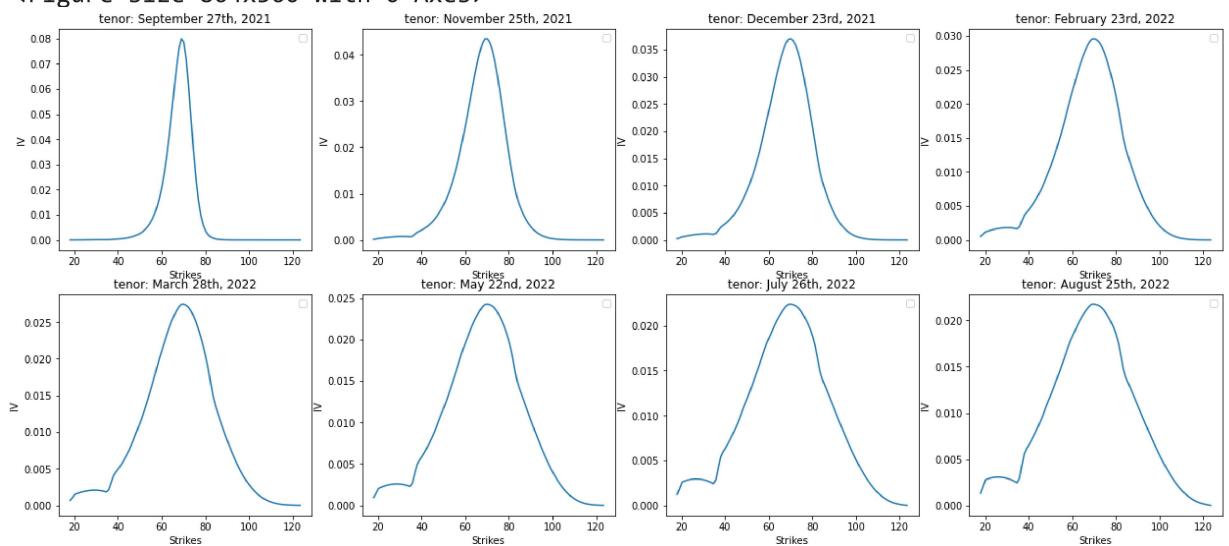


IV Surface on OIL vs Heston Surface
 August 30th, 2021 to October 26th, 2022
 Beta = 1



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

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

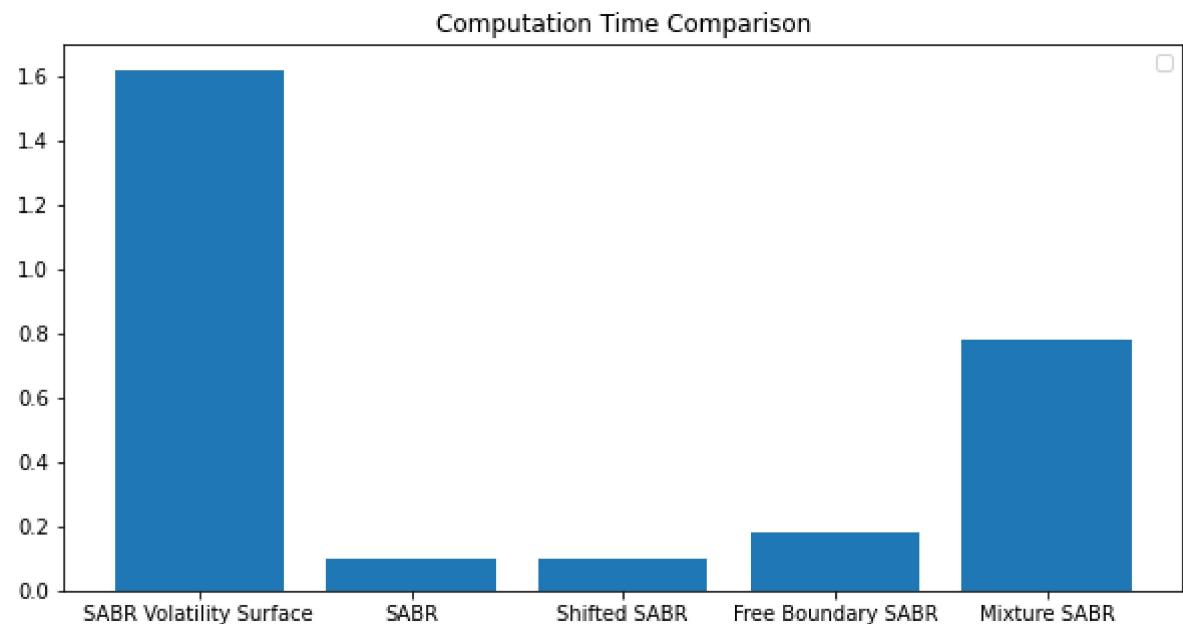


```
In [23]: # Compute time for each model
time_data = compute_time()

time_df = pd.DataFrame(time_data.values(), index=time_data.keys(), columns=[ "Computa
fig = plt.figure(figsize=(10, 5), )
plt.bar(x=time_data.keys(), height=time_data.values(), )
plt.title("Computation Time Comparison")
plt.legend()
```

No handles with labels found to put in legend.

Out[23]: <matplotlib.legend.Legend at 0x203c6142190>



In []: