In [2]:

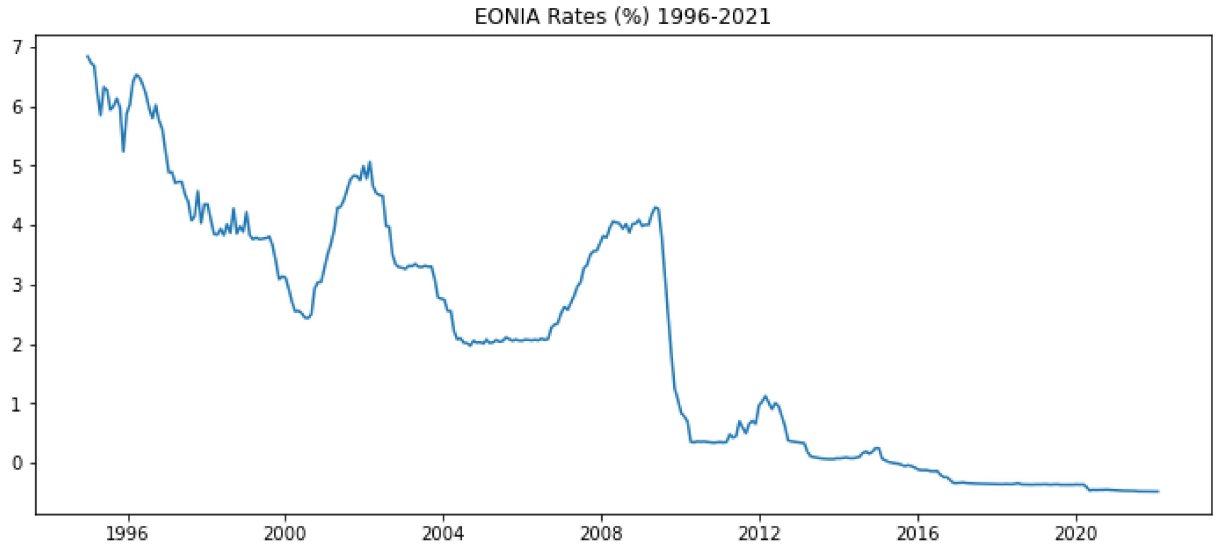
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import pandas as pd
import time
import datetime
import math
import QuantLib as ql
from scipy.optimize import minimize

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

In [3]:

```
# 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

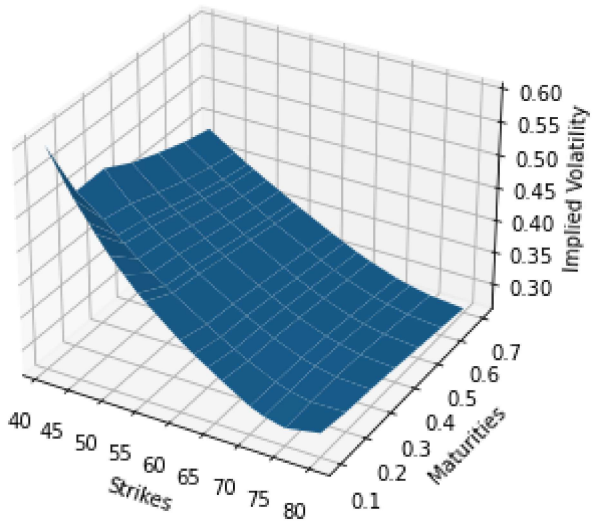
Out[3]: Text(0.5, 1.0, 'EONIA Rates (%) 1996-2021')



In [4]:

```
# 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
```

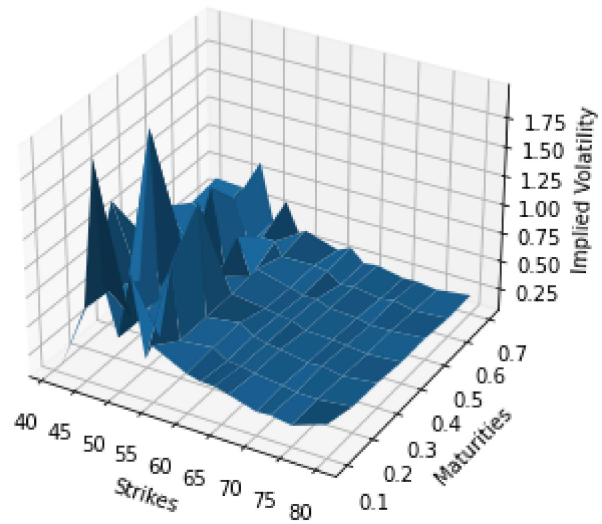
Black-Scholes Implied Volatility Surface on OIL
August 30th, 2021 to May 17th, 2022

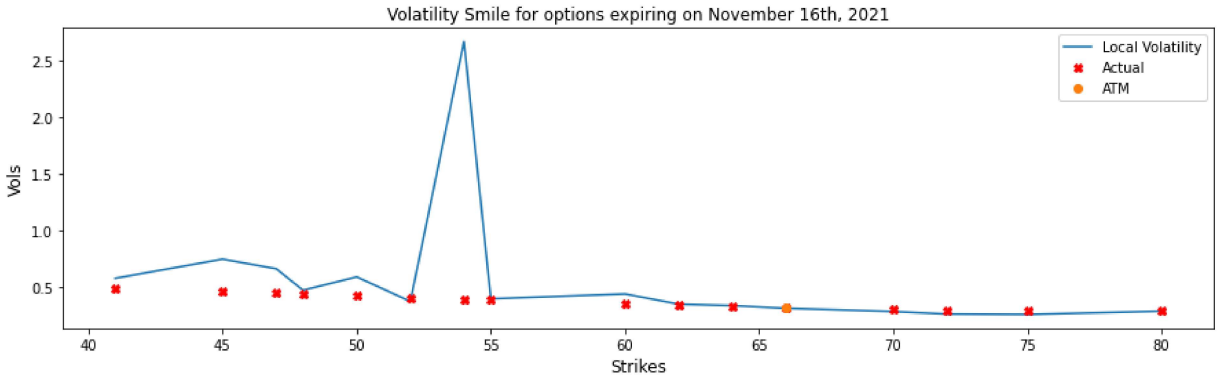


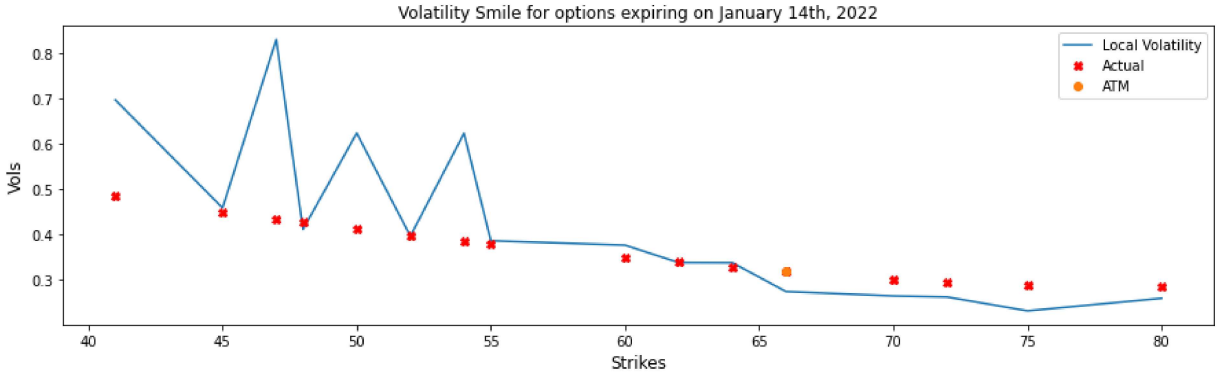
In [5]:

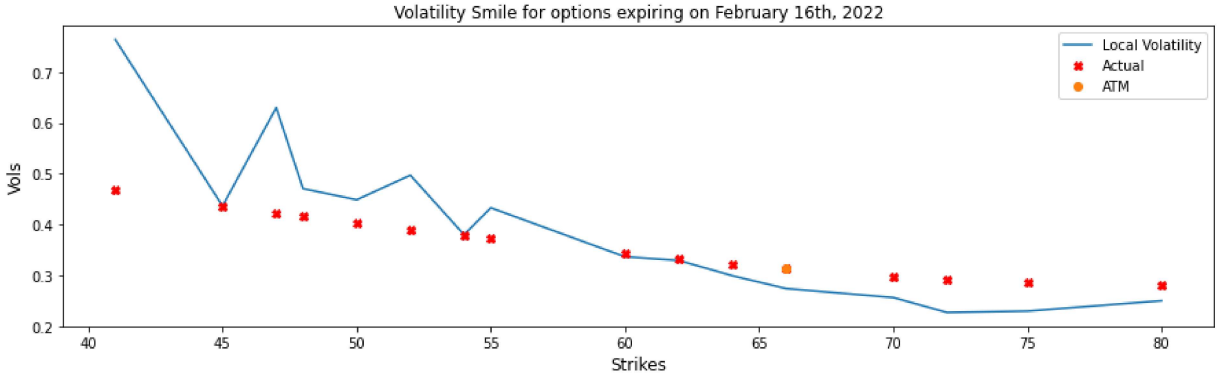
```
#DUPIRE LOCAL VOLATILITY SURFACE (NOT PLOTTABLE)

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)
```

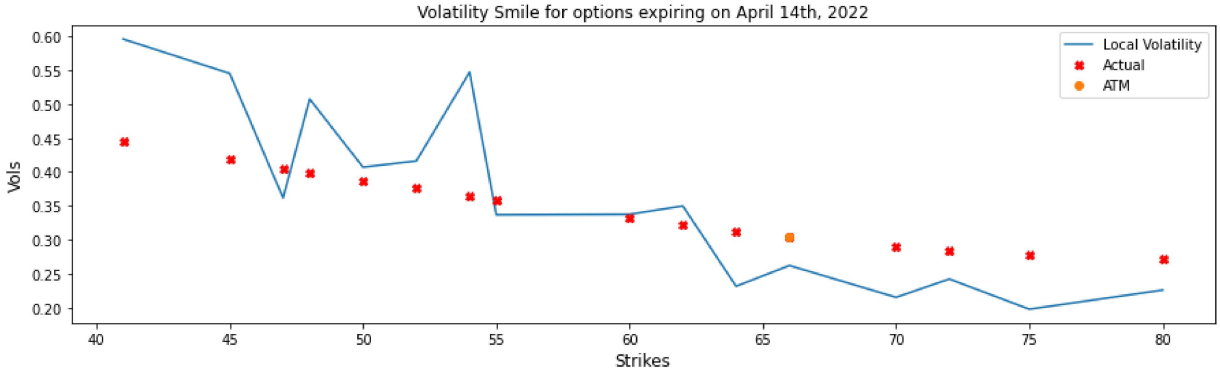
Dupire Local Volatility Surface on WTI Crude Oil

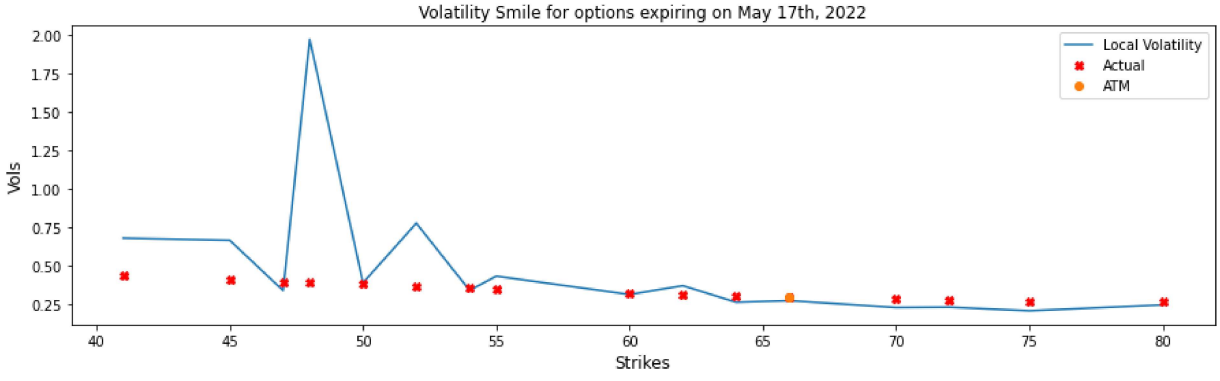












In [6]:

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

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

if data == "SPX":
    m1_params = (0.05,0.2,0.5,0.1,0.09)
    m2_params = (0.15,0.5,0.2,0.7,0.01)
elif data == "COFFEE":
    m1_params = (0.01,0.1,0.3,0.1,0.02)
    m2_params = (0.2,0.9,0.9,0.9,-0.19)
elif data == "OIL":
    m2_params = (0.023,0.009,1.00,0.95,0.2)
    m1_params = (0.15,0.5,0.2,0.7,0.01)
elif data == "GOLD":
    m1_params = (0.03,0.3,0.5,0.3,0.04)
    m2_params = (0.01,0.5,0.5,0.1,0.03)
else:
    m1_params = (0.03,0.3,0.5,0.3,0.04)
    m2_params = (0.01,0.5,0.5,0.1,0.03)

hestonModel1 = hestonModelSurface(m1_params, label="Heston Model 1, {}".format(data_
hestonModel2 = hestonModelSurface(m2_params, label="Heston Model 2, {}".format(data_

# Use to Calibrate first time the Heston Model
def calibrateHeston():
    def f(params):
        return hestonModelSurface(params).avgError
        # v0, kappa, theta, sigma, rho
    cons = (
            {'type': 'ineq', 'fun': lambda x: x[0] - 0.001},
            {'type': 'ineq', 'fun': lambda x: 2. - x[0]},
            {'type': 'ineq', 'fun': lambda x: x[1] - 0.001},
            {'type': 'ineq', 'fun': lambda x: 2. - x[1]},
            {'type': 'ineq', 'fun': lambda x: x[2] - 0.001},
            {'type': 'ineq', 'fun': lambda x: 2. - x[2]},
            {'type': 'ineq', 'fun': lambda x: x[3] - 0.001},
            {'type': 'ineq', 'fun': lambda x: .999 - x[3]},
            {'type': 'ineq', 'fun': lambda x: .99 - x[4]**2}
            )
    result = minimize(f, m1_params, constraints=cons, method="SLSQP", bounds=((1e-8,
    hestonModel0 = hestonModelSurface(result["x"], label="Heston Model 0, {}".format


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

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

Heston Model Initial Conditions on (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 [7]:

```
# 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	41.000000	0.777746	0.760591	-2.205772
1	45.000000	1.126362	1.124302	-0.182865
2	47.000000	1.340478	1.350918	0.778809
3	48.000000	1.469466	1.477097	0.519270
4	50.000000	1.745112	1.758073	0.742698
5	52.000000	2.065527	2.081707	0.783340
6	54.000000	2.420076	2.454429	1.419468
7	55.000000	2.619973	2.661603	1.588934
8	60.000000	3.843719	3.956029	2.921889
9	62.000000	4.462640	4.624389	3.624511
10	64.000000	5.155103	5.401357	4.776882
11	66.000000	5.951370	6.301558	5.884171
12	70.000000	5.469599	6.103811	11.595221
13	72.000000	4.596370	5.389752	17.261047
14	75.000000	3.498663	4.519952	29.190815
15	80.000000	2.198774	3.467044	57.680767

Heston Model 1, WTI Crude Oil parameters output

	Value
v0	0.166211
kappa	0.000000
theta	0.973239
sigma	-0.077452
rho	0.141798
avgError	8.822279

```
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	41.000000	0.777746	0.772297	-0.700644
1	45.000000	1.126362	1.127930	0.139202

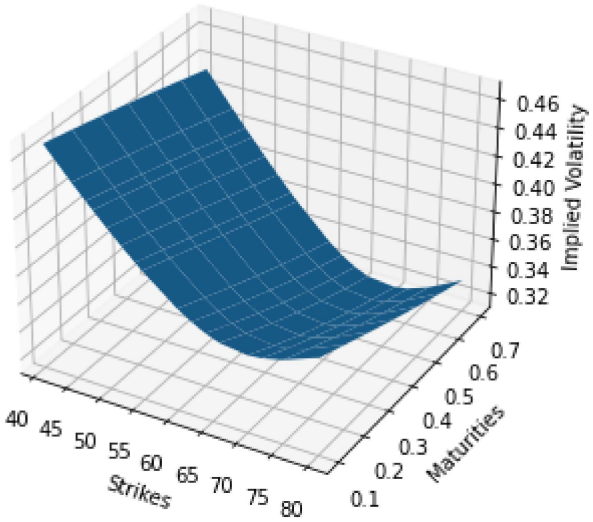
	Strikes	Market Value	Model Value	Relative Error (%)
2	47.000000	1.340478	1.349282	0.656769
3	48.000000	1.469466	1.472456	0.203441
4	50.000000	1.745112	1.746436	0.075911
5	52.000000	2.065527	2.061186	-0.210139
6	54.000000	2.420076	2.421979	0.078613
7	55.000000	2.619973	2.621487	0.057755
8	60.000000	3.843719	3.844906	0.030866
9	62.000000	4.462640	4.458570	-0.091200
10	64.000000	5.155103	5.157036	0.037484
11	66.000000	5.951370	5.950287	-0.018199
12	70.000000	5.469599	5.460680	-0.163067
13	72.000000	4.596370	4.583127	-0.288114
14	75.000000	3.498663	3.490033	-0.246680
15	80.000000	2.198774	2.204387	0.255269

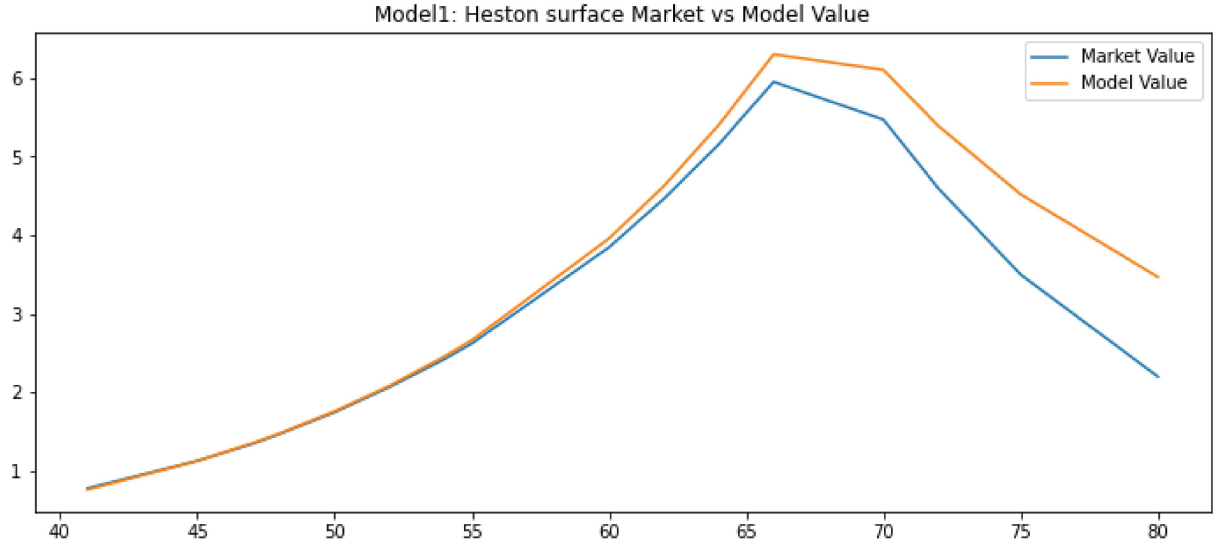
Heston Model 2, WTI Crude Oil parameters output

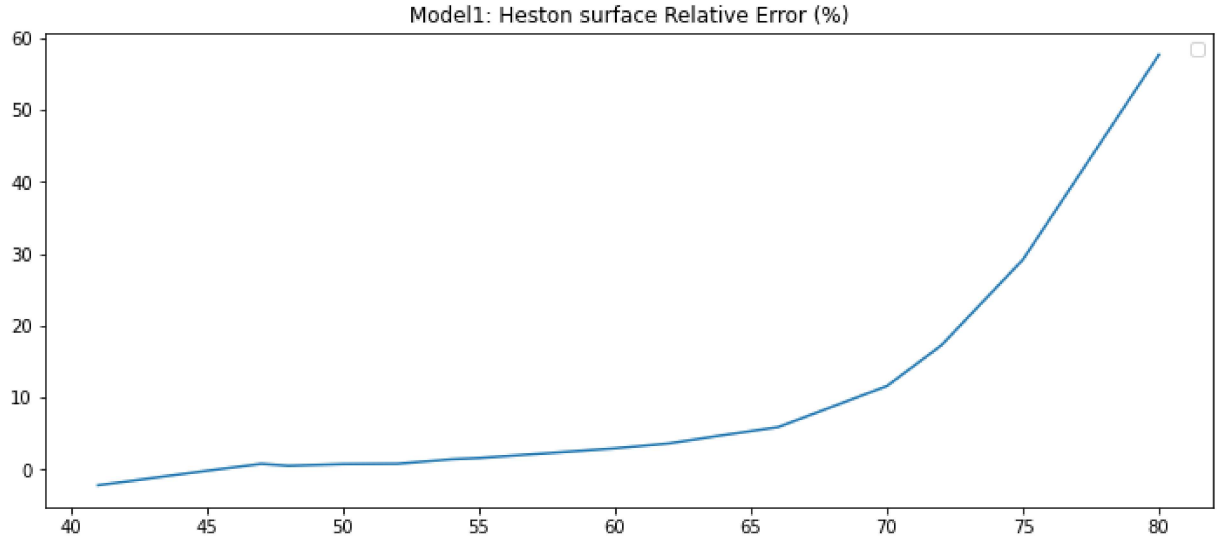
	Value
v0	3.966787
kappa	0.086569
theta	1.185124
sigma	-0.500578
rho	0.000272
avgError	0.203335

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

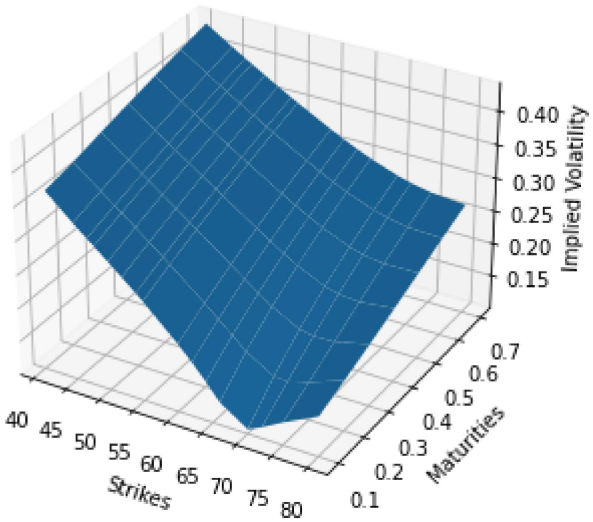
Heston Volatility Surface for Heston Model 1, WTI Crude Oil

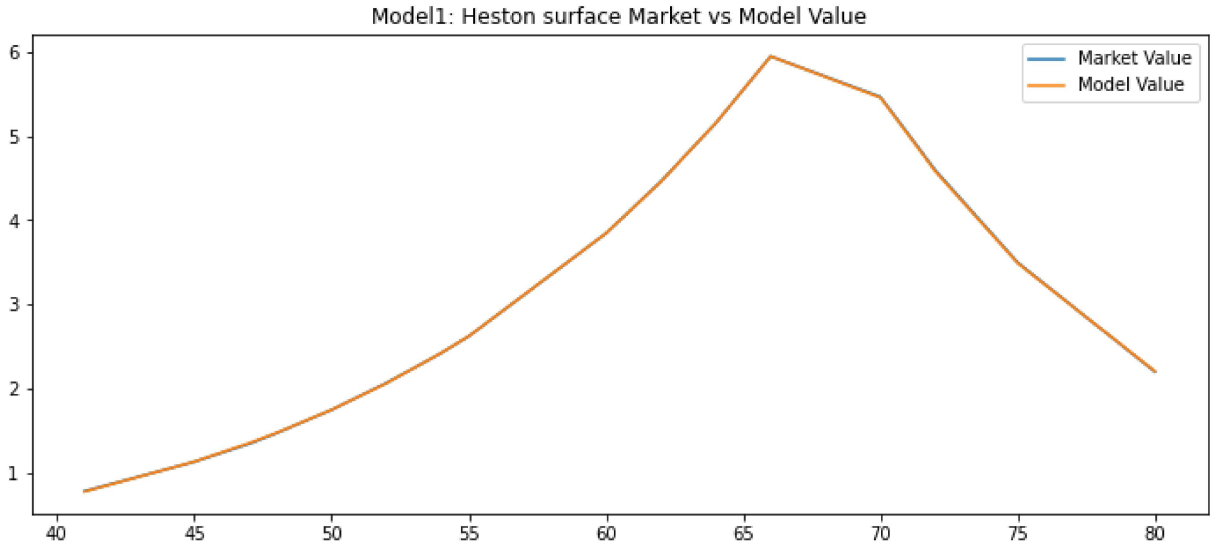


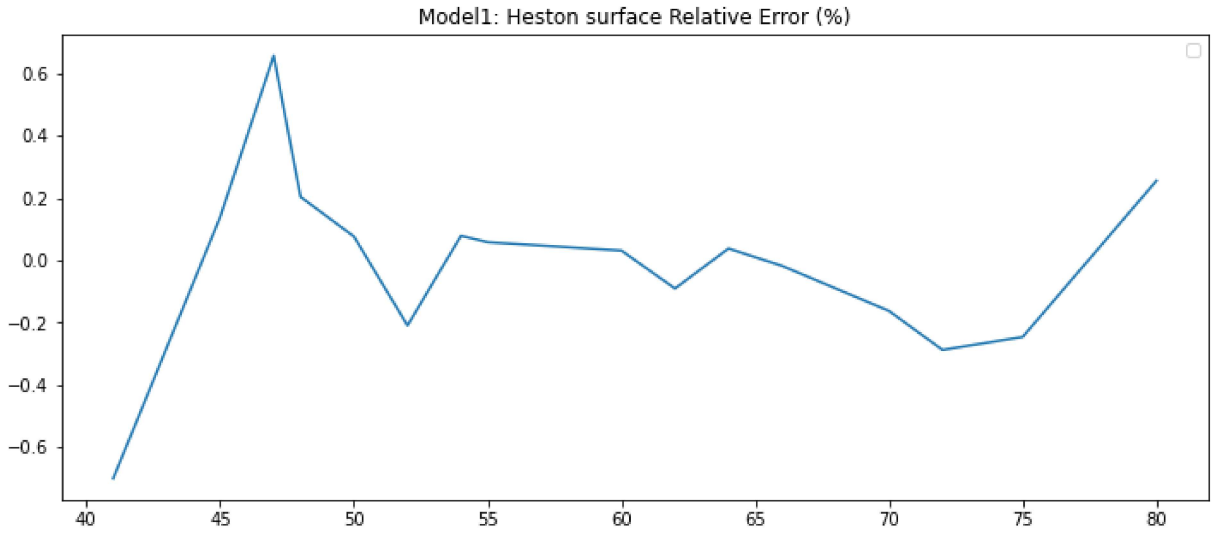




Heston Volatility Surface for Heston Model 2, WTI Crude Oil

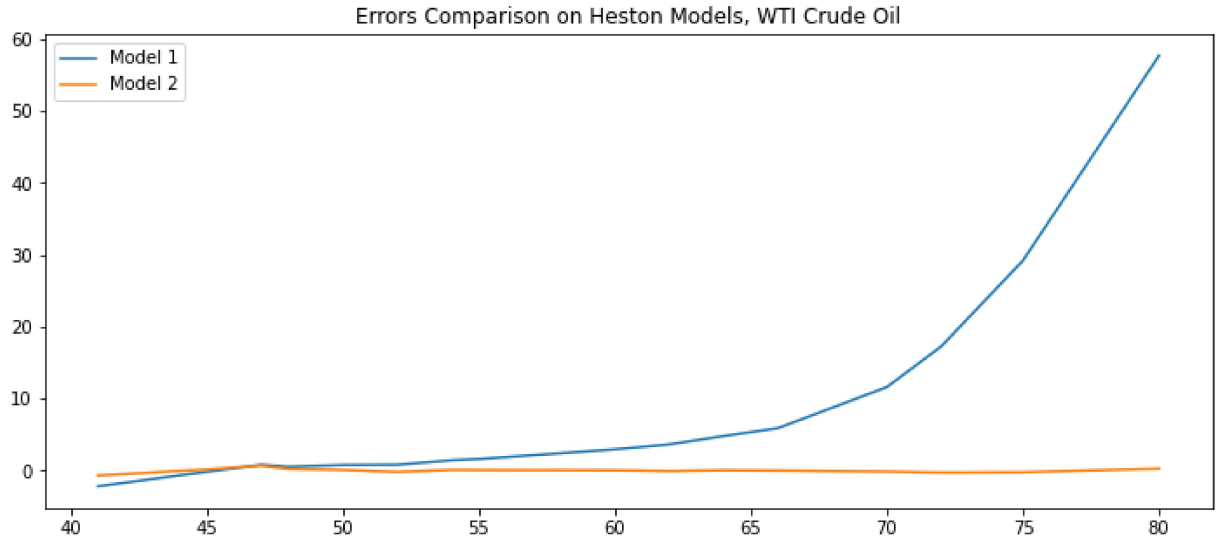


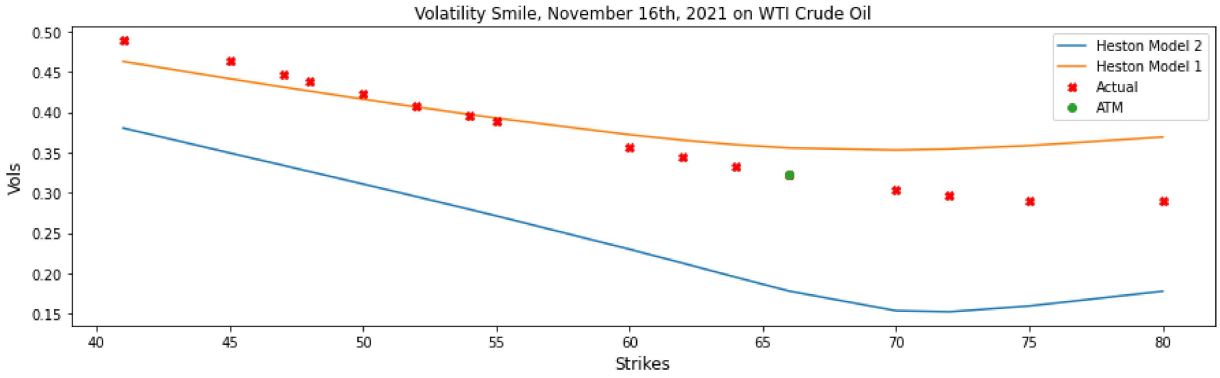


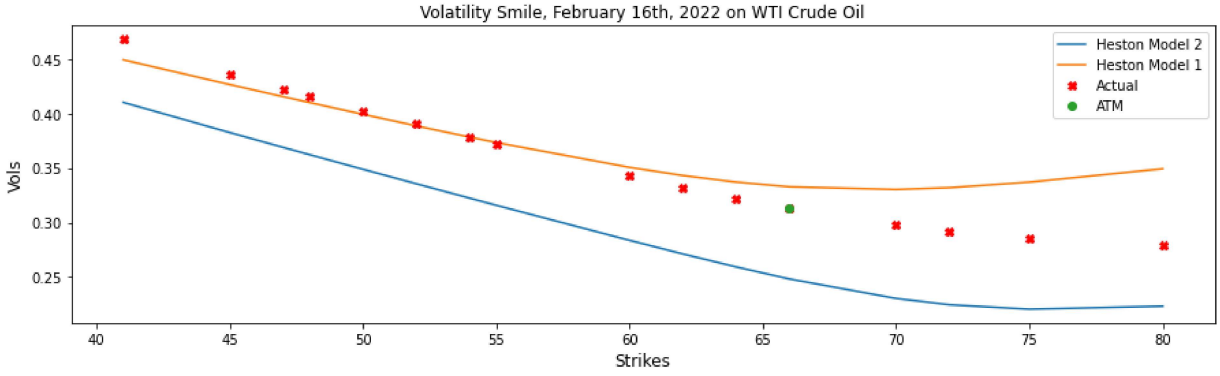


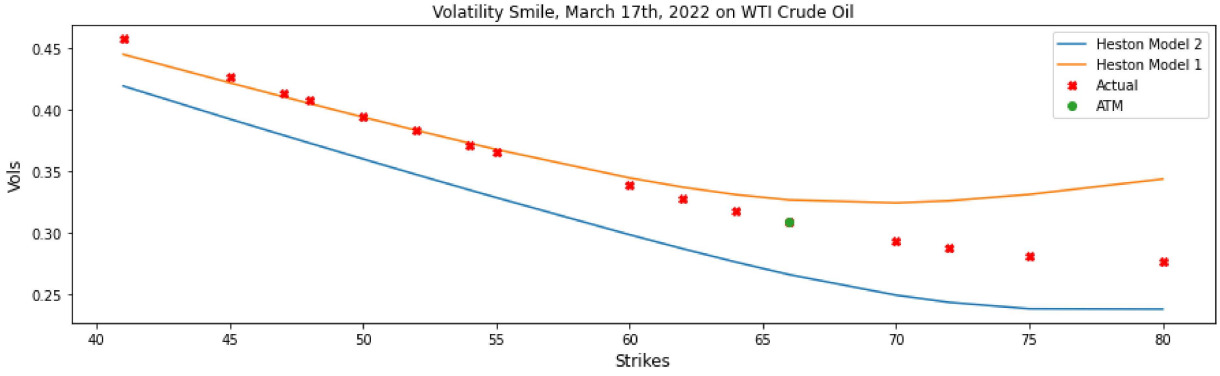
In [8]:

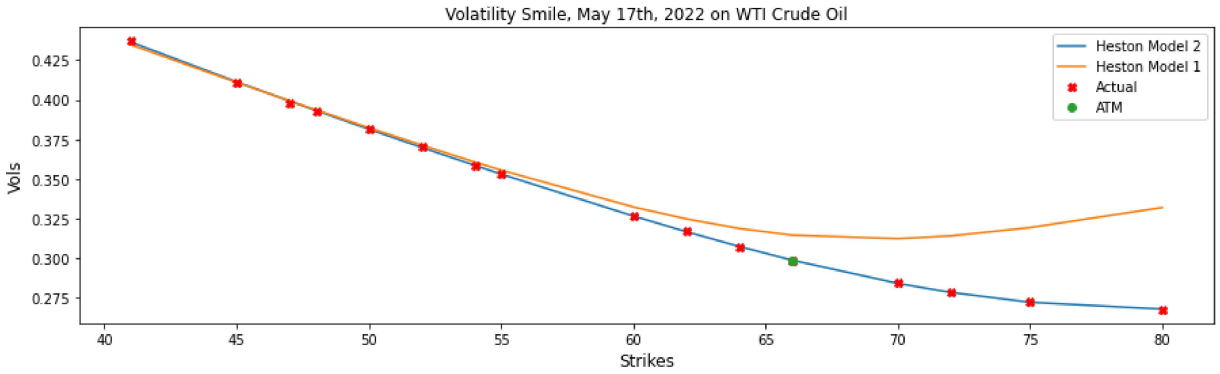
```
# Relative error comparison
plt.figure(figsize=plot_size)
plt.plot(hestonModel1.strks, hestonModel1.relativeError, label="Model 1")
plt.plot(hestonModel2.strks, hestonModel2.relativeError, label="Model 2")
plt.title("Errors Comparison on Heston Models, {}".format(data_label));
plt.legend()

# 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, WTI Crude Oil
Model 1
Model 2


Volatility Smile, November 16th, 2021 on WTI Crude Oil
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes


Volatility Smile, February 16th, 2022 on WTI Crude Oil
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes


Volatility Smile, March 17th, 2022 on WTI Crude Oil
Heston Model 2
Heston Model 1
Actual
ATM
Vols
Strikes



In [12]:

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

from initialize import *
from plotting import *

# CALIBRATE SABR VOLATILITY SURFACE

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

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

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

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

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

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

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

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

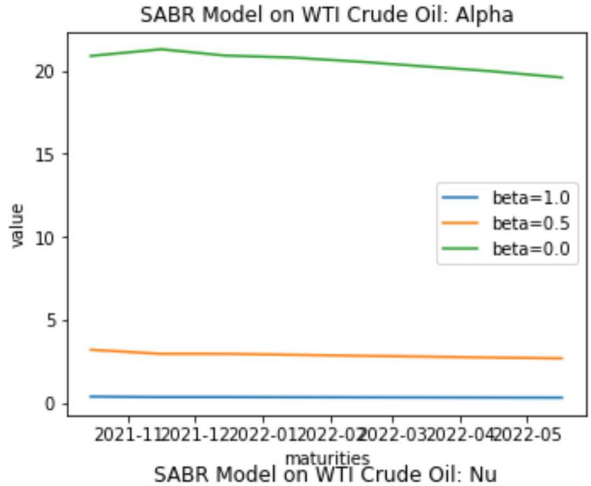
```python
class SABRSmile:
    def __init__(self, date, shift=0, beta=1, method="normal", fwd=current_price, ze
        self.date = date
        self.expiryTime = round((self.date - today)/365, 6)
        self.marketVols = vols[dates.index(self.date)]
        self.shift = shift
        self.fwd = fwd
        self.forward_price = self.fwd * \
            math.exp(rate.value() * self.expiryTime)
        self.zero_rho = zero_rho
        self.alpha, self.beta, self.nu, self.rho = (
            .1, beta, 0., 0. if self.zero_rho else .1)
        self.method = method
        self.newVols = None
        self.error = None

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

        x = self.set_init_conds()

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

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

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

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

    def f(self, params):

        alpha, beta, nu, rho = params

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

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

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

        return self.error


class SABRVolatilitySurface:
```

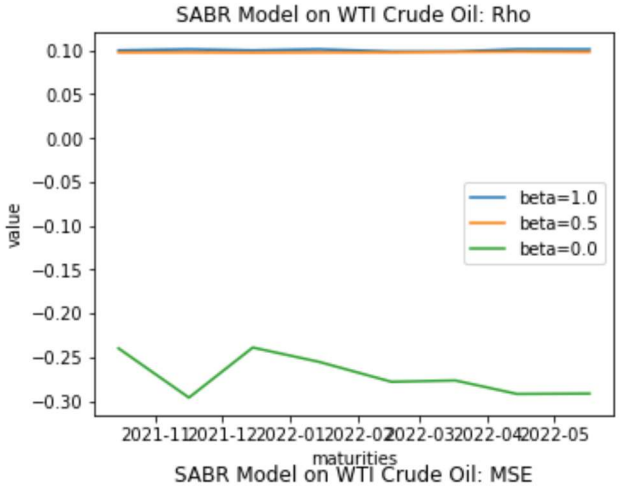
```
    def __init__(self, method="normal", beta=1, shift=0, fwd=current_price, label=""
        self.method = method
        self._beta = beta
        self.shift = shift
        self.fwd = fwd
        self.label = label
        self.zero_rho = zero_rho

        self.initialize()

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

        for i, d in enumerate(dates):
            volSABR = SABRSmile(date=d, beta=self._beta, shift=self.shift,
                                method=self.method, fwd=self.fwd, zero_rho=self.zero
            volSABR.initialize()

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

            self.errors.append(volSABR.error)

            smile = volSABR.newVols

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

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

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

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


# Backbone modelling for SABR
def SABR_backbone_plot(beta=1, bounds=None, shift=0, fixes=(.95, 1, 1.14, 1.24), ten
    l = []
    for i in fixes:
        vol_surface = SABRVolatilitySurface(
            method="normal", shift=current_price*shift, beta=beta, fwd=current_price
        SABR_vol_surface = ql.BlackVarianceSurface(
            today, calendar, dates, strikes, vol_surface.SABRVolMatrix, day_count)
        SABR_vol_surface.enableExtrapolation()

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

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

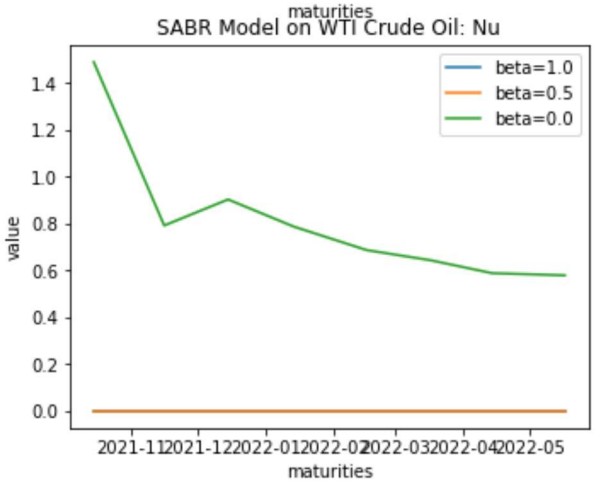
```
def SABRComparison(methods, title="", display=False):
    fig, axs = plt.subplots(2, 2, figsize=plot_size)
    plt.subplots_adjust(left=None, bottom=None, right=None,
                        top=1.5, wspace=None, hspace=None)

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

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

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

    smiles_comparison(methods)
```

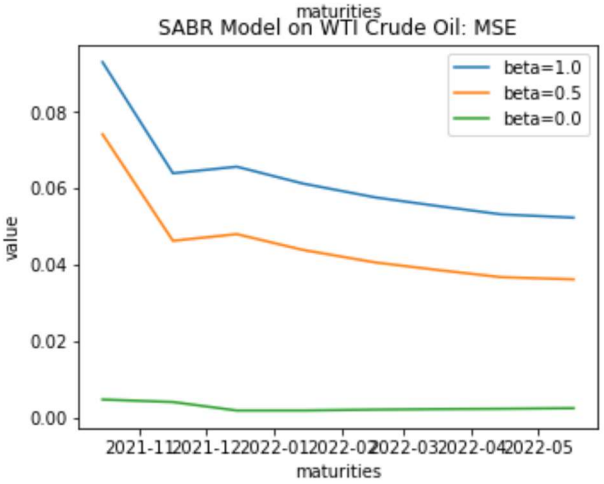
In [13]:

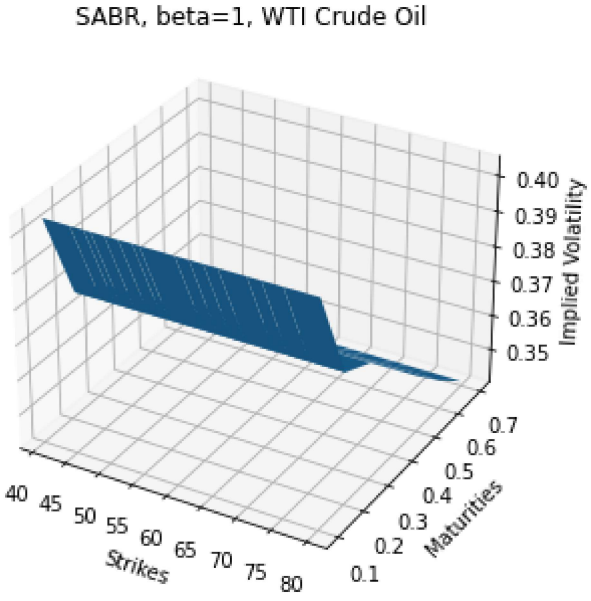
```
# SABR Volatility model

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

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


SABR Model on WTI Crude Oil: Alpha
value
maturities
beta=1.0
beta=0.5
beta=0.0

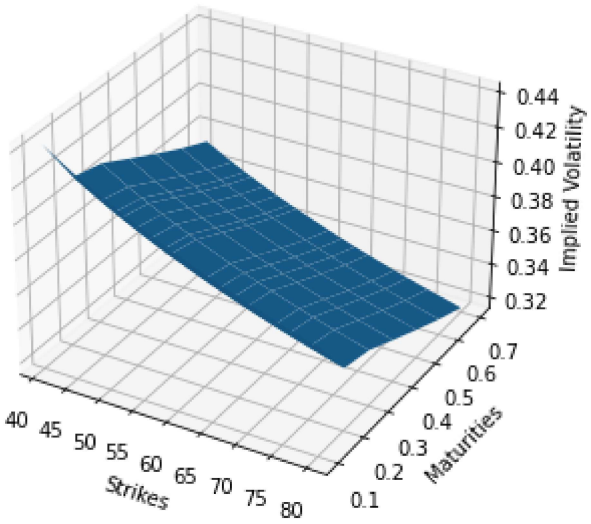

SABR Model on WTI Crude Oil: Rho
value
maturities
beta=1.0
beta=0.5
beta=0.0


SABR Model on WTI Crude Oil: Nu
value
maturities
beta=1.0
beta=0.5
beta=0.0

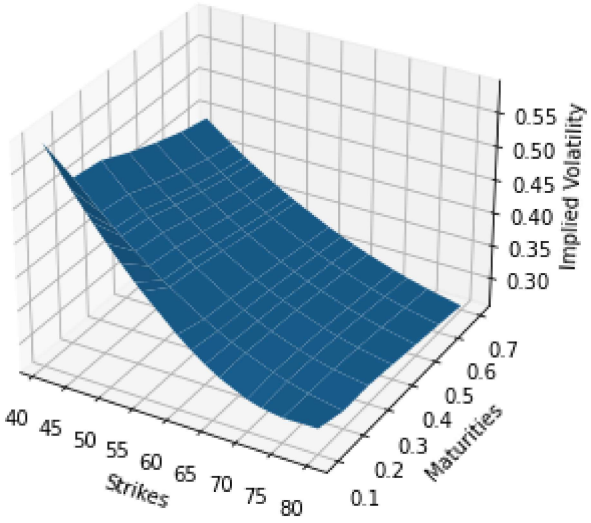

SABR Model on WTI Crude Oil: MSE
value
maturities
beta=1.0
beta=0.5
beta=0.0

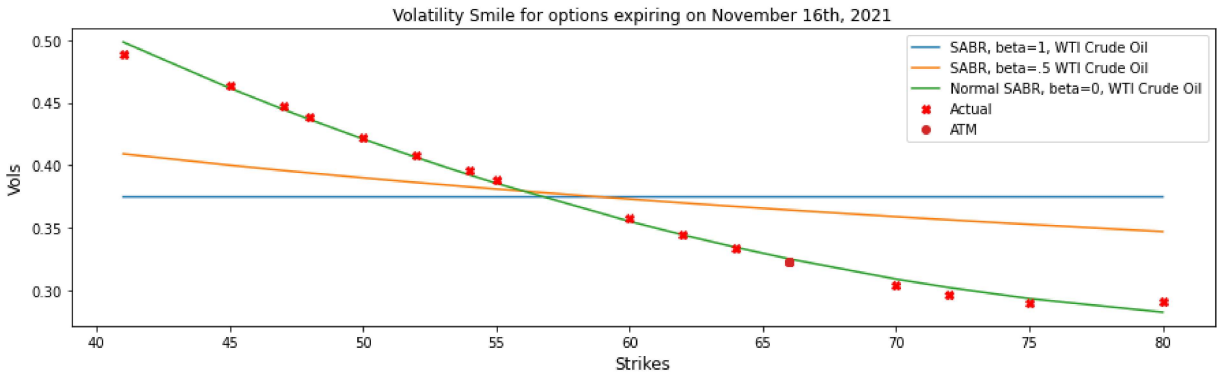

SABR, beta=1, WTI Crude Oil
Implied Volatility
Strikes
Maturities

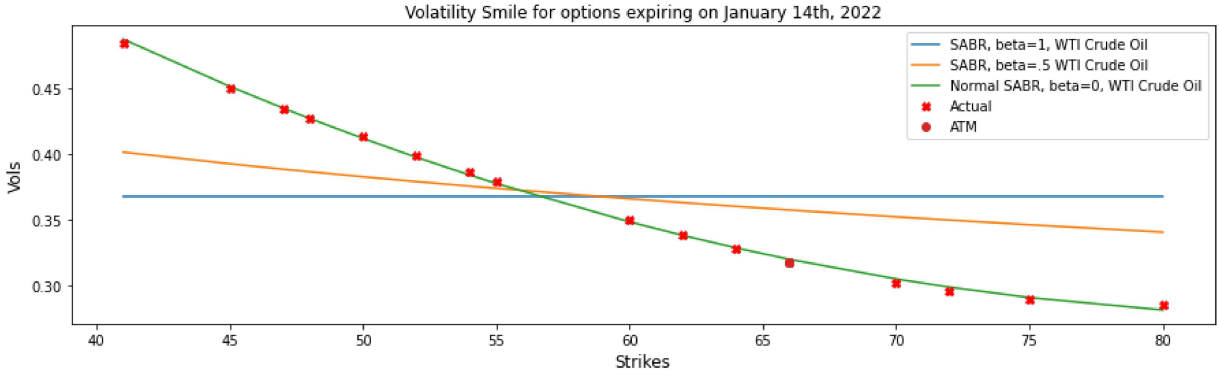
SABR, beta=.5 WTI Crude Oil

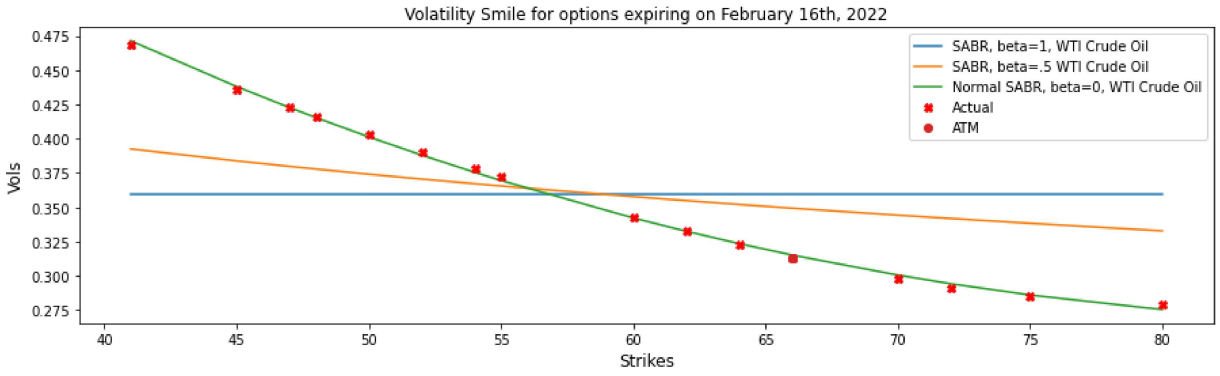


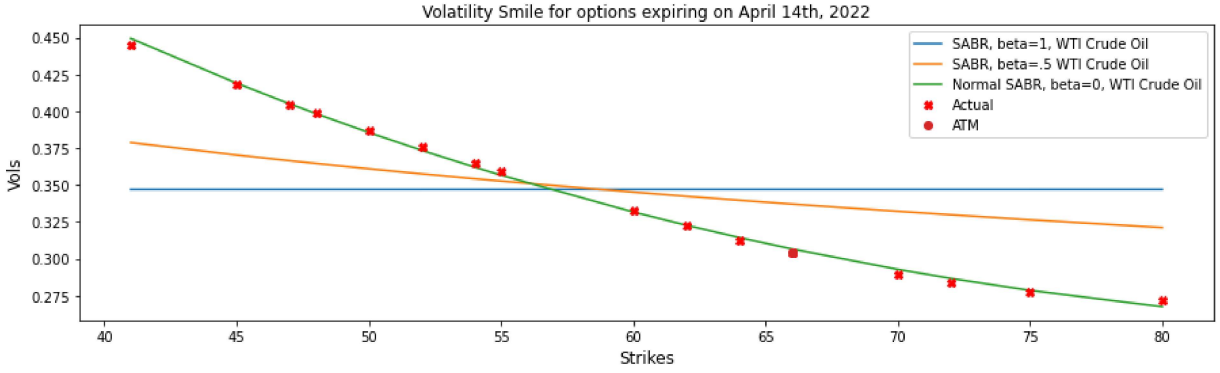
Normal SABR, beta=0, WTI Crude Oil

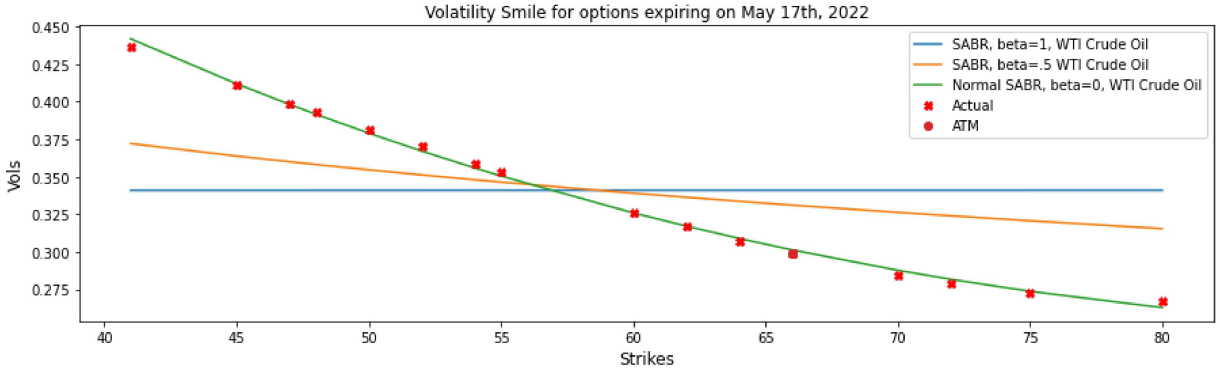








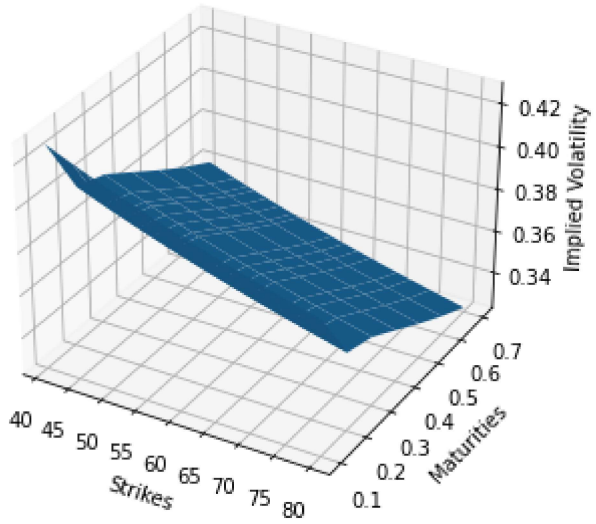




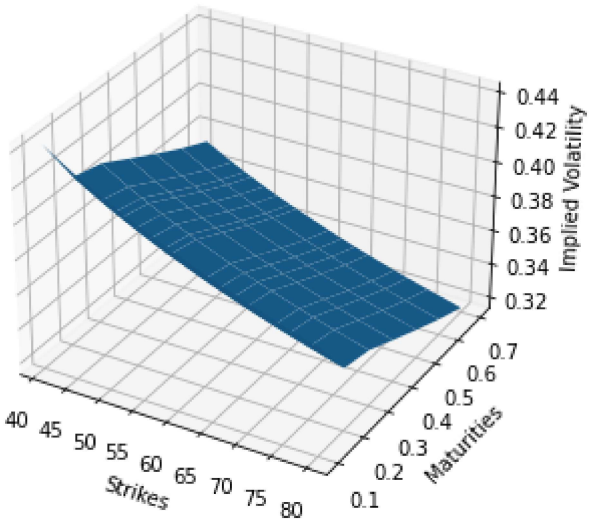
In [14]:

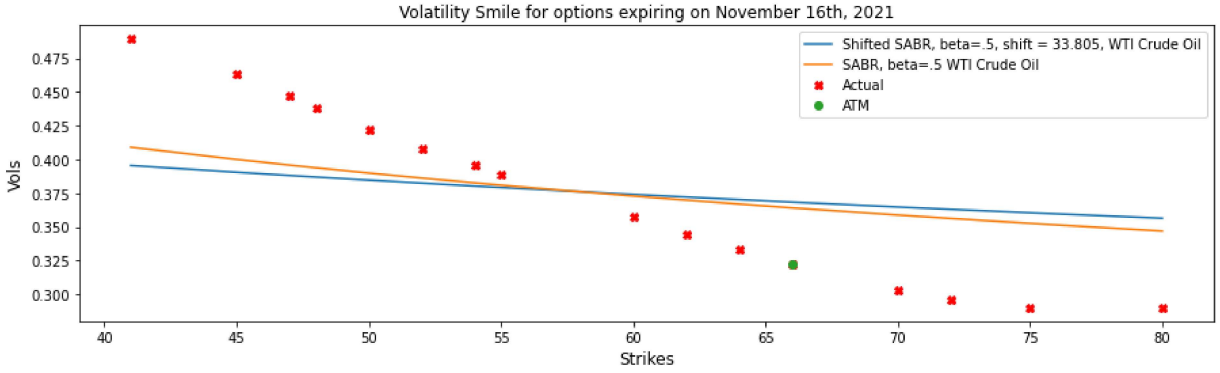
```
# Shifted SABR Volatility model
shft = .50 * current_price
shiftedSABR_beta1 = SABRVolatilitySurface(beta=1, shift=shft, label="Shifted SABR, b
shiftedSABR_beta5 = SABRVolatilitySurface(beta=.5, shift=shft, label="Shifted SABR,
shiftedSABR_beta0 = SABRVolatilitySurface(beta=.0, shift=shft, label="Shifted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

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

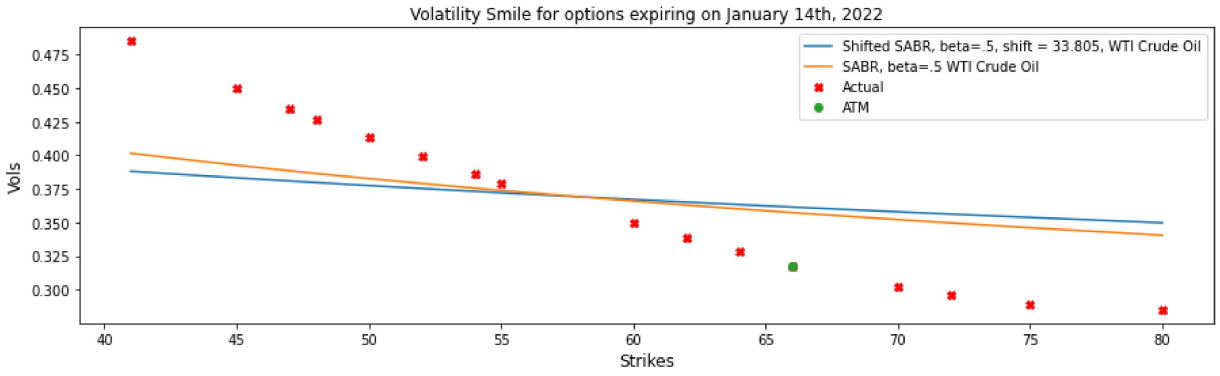
Shifted SABR, beta=.5, shift = 33.805, WTI Crude Oil

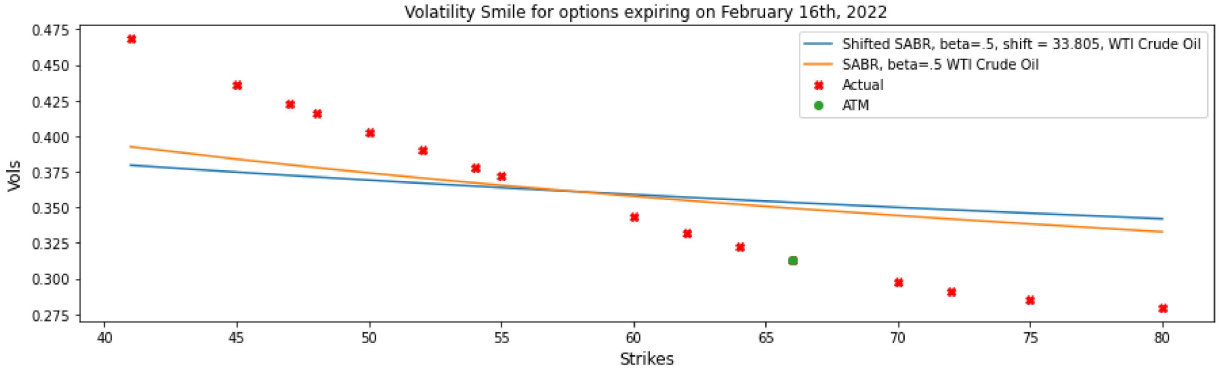


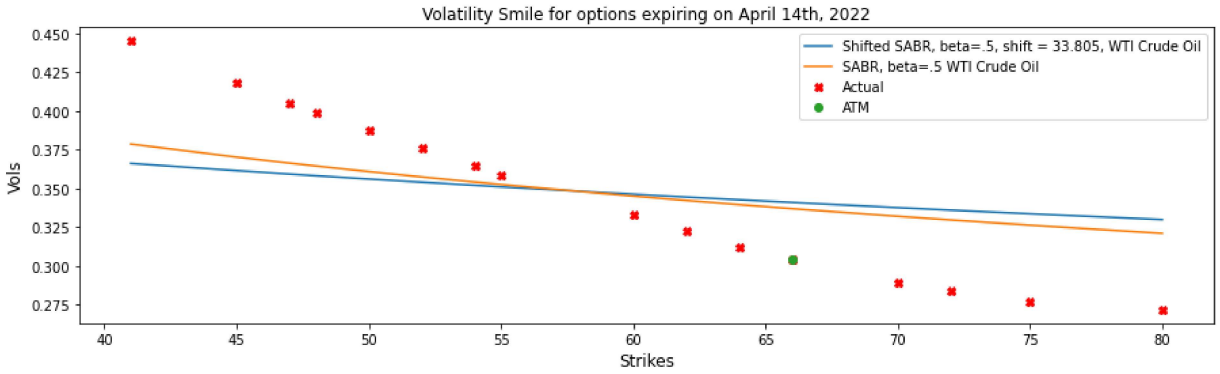
SABR, beta=.5 WTI Crude Oil

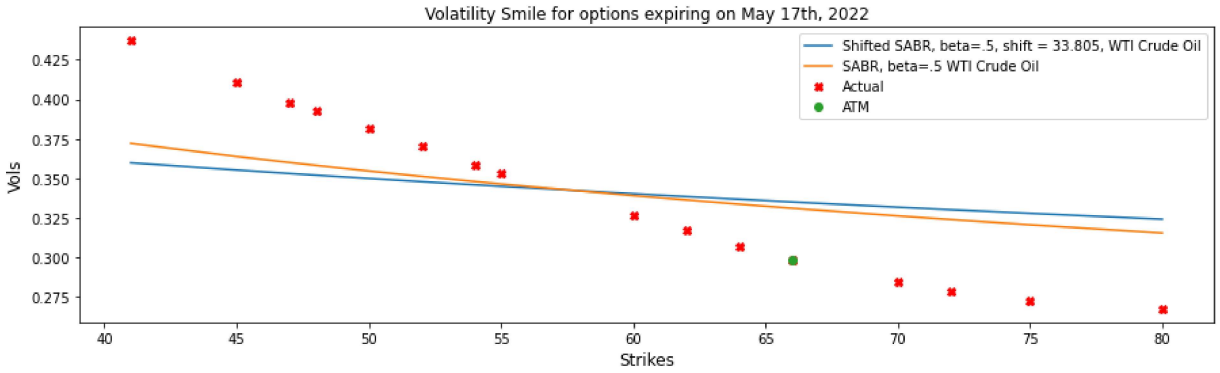


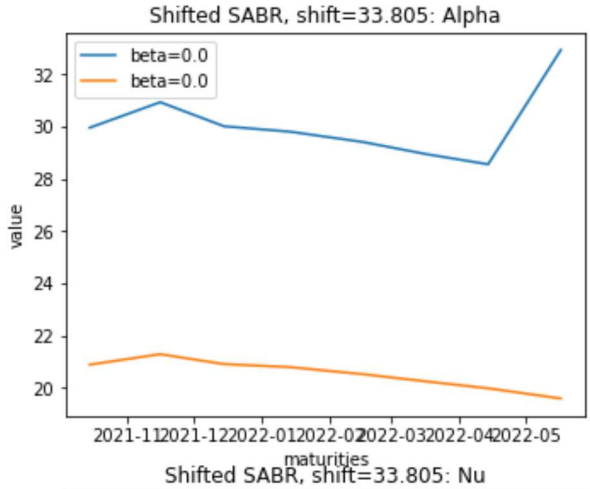


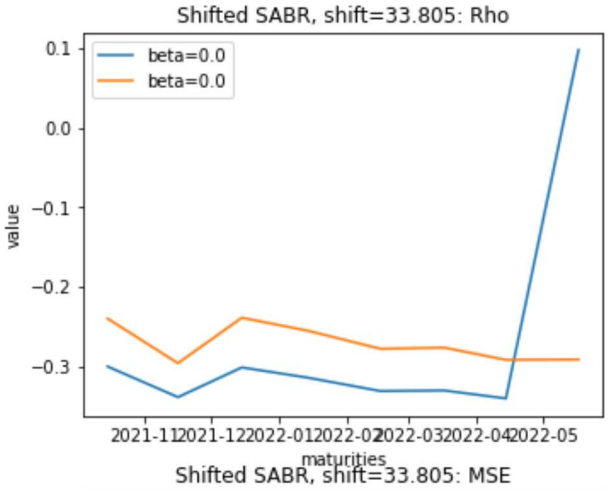


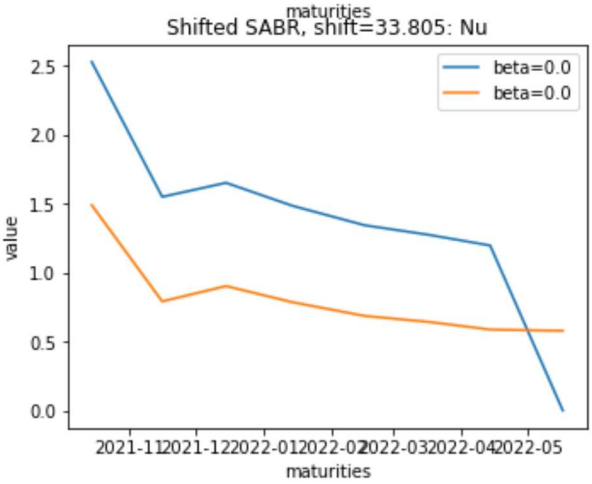


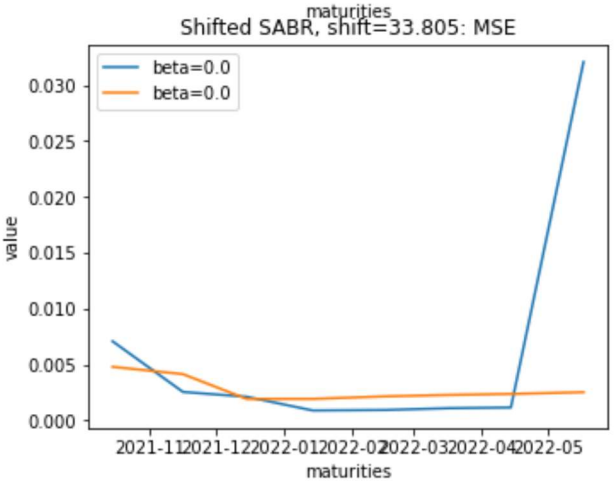

Volatility Smile for options expiring on April 14th, 2022
Shifted SABR, beta=.5, shift = 33.805, WTI Crude Oil
SABR, beta=.5 WTI Crude Oil
Actual
ATM
Vols
Strikes


Volatility Smile for options expiring on May 17th, 2022
Shifted SABR, beta=.5, shift = 33.805, WTI Crude Oil
SABR, beta=.5 WTI Crude Oil
Actual
ATM
Vols
Strikes

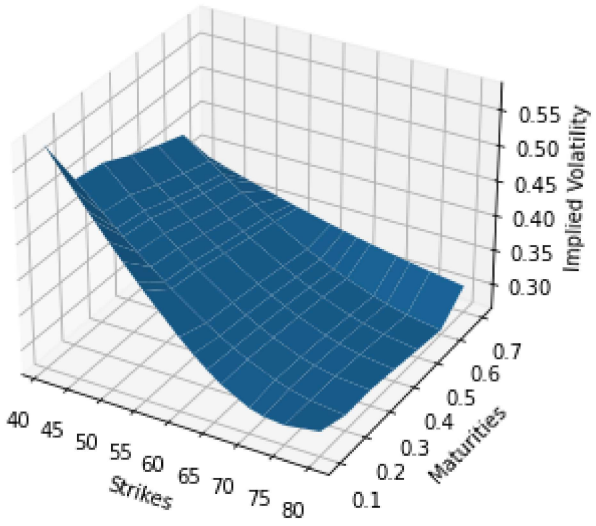

Shifted SABR, shift=33.805: Alpha
beta=0.0
beta=0.0
value
maturities


Shifted SABR, shift=33.805: Rho
beta=0.0
beta=0.0
value
maturities

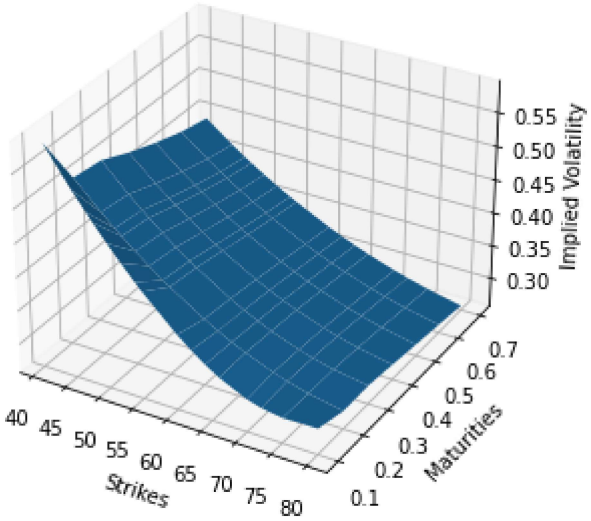

Shifted SABR, shift=33.805: Nu
beta=0.0
beta=0.0
value
maturities

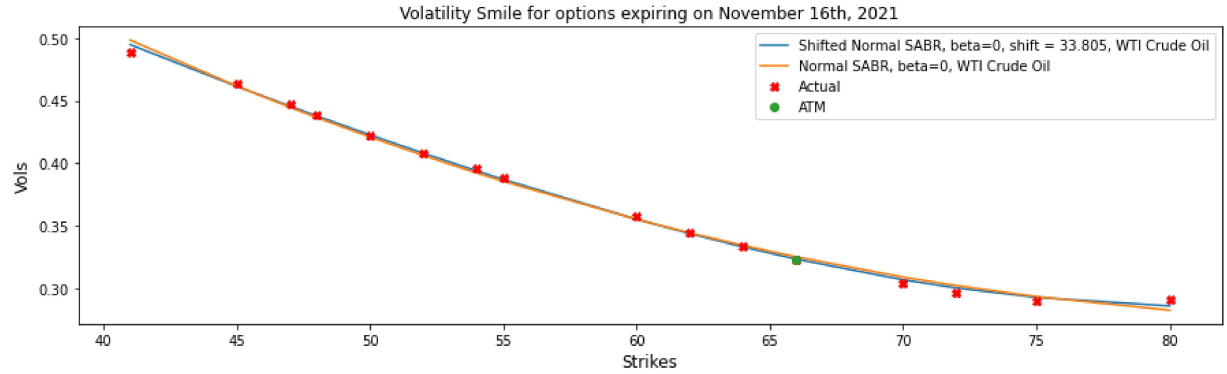

Shifted SABR, shift=33.805: MSE
beta=0.0
beta=0.0
value
maturities

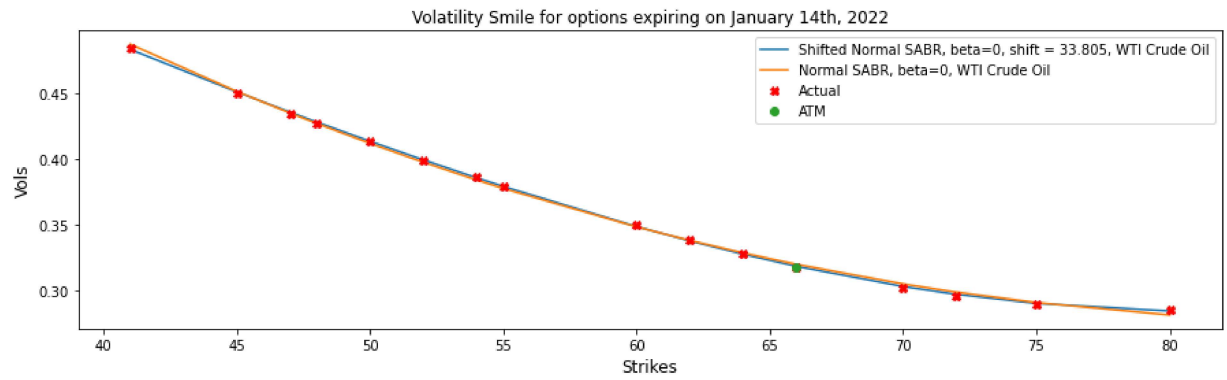
Shifted Normal SABR, beta=0, shift = 33.805, WTI Crude Oil

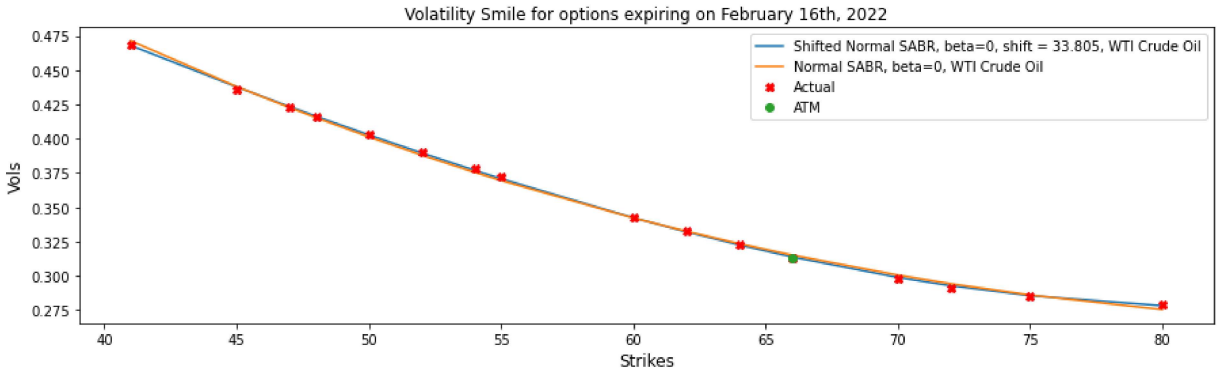


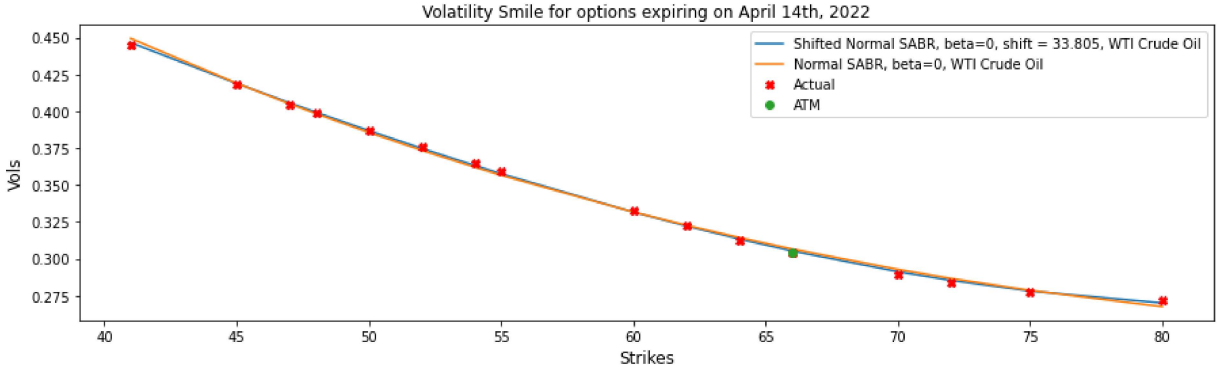
Normal SABR, beta=0, WTI Crude Oil

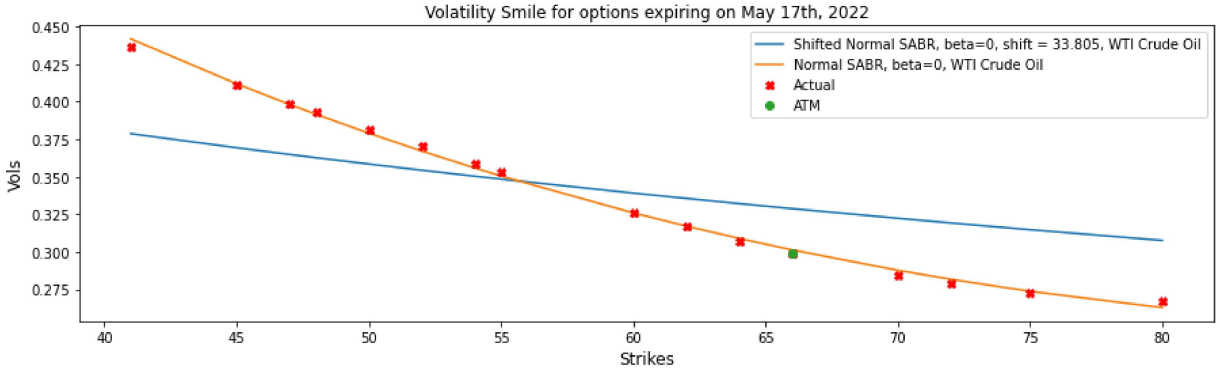






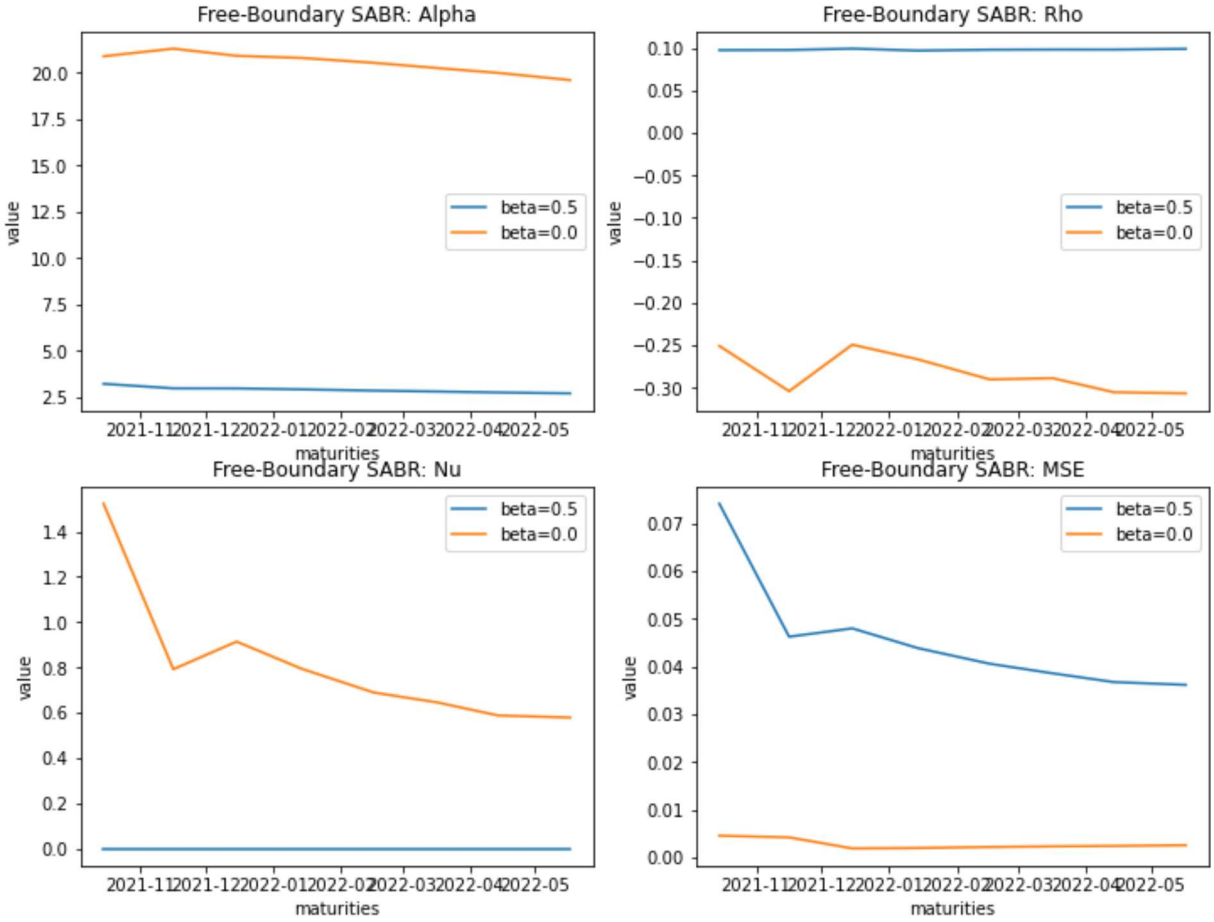




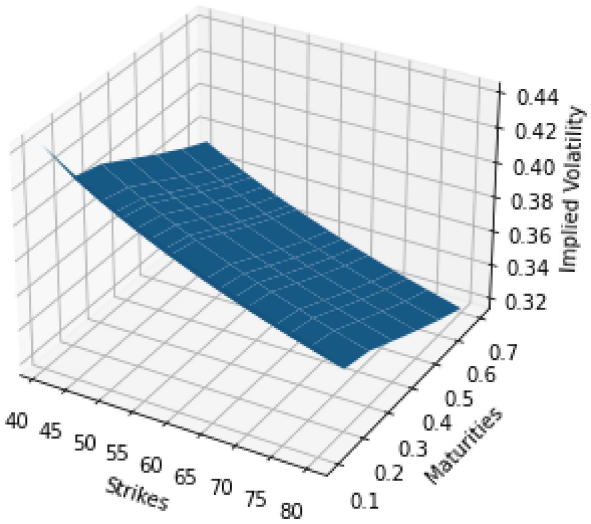


In [15]:

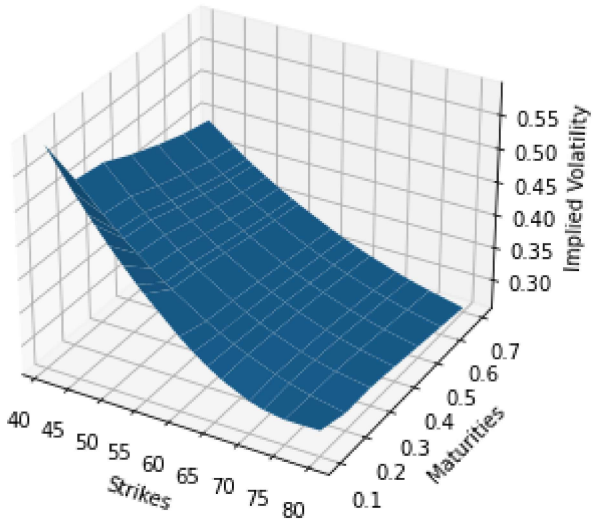
```
# Free-Boundary SABR Volatility model

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

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

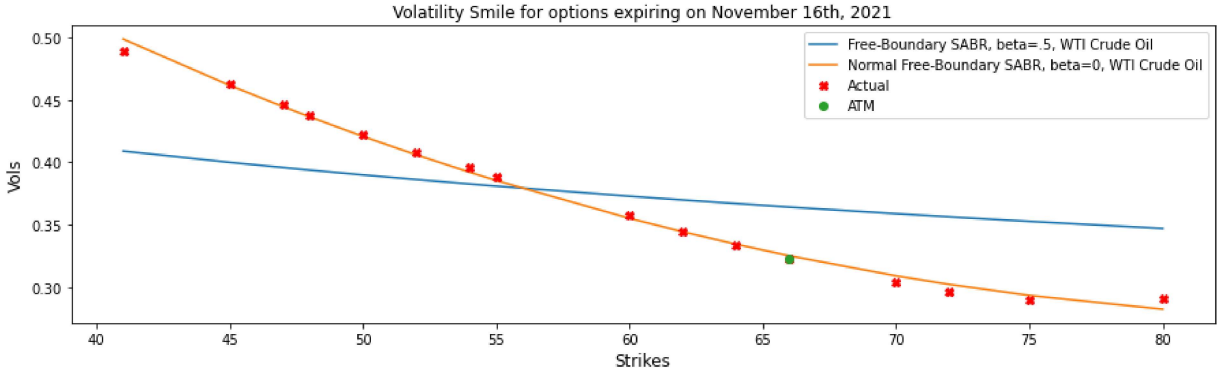


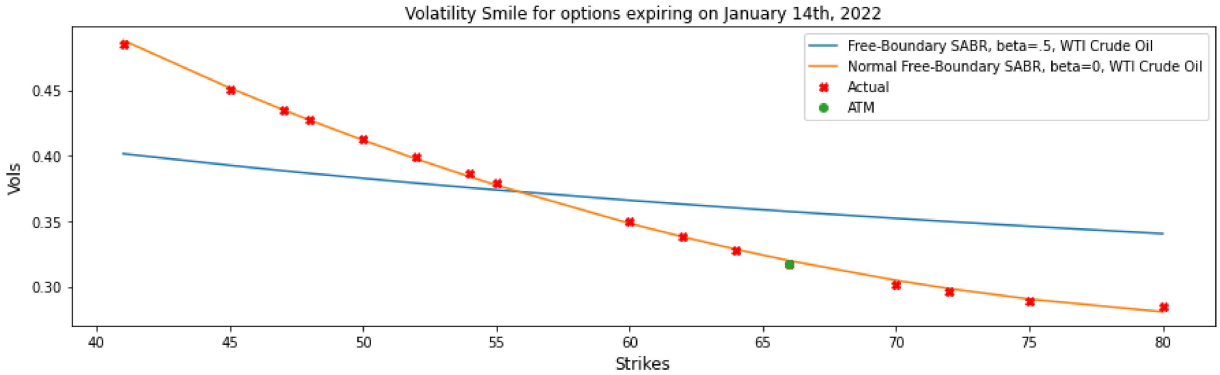
Free-Boundary SABR, beta=.5, WTI Crude Oil

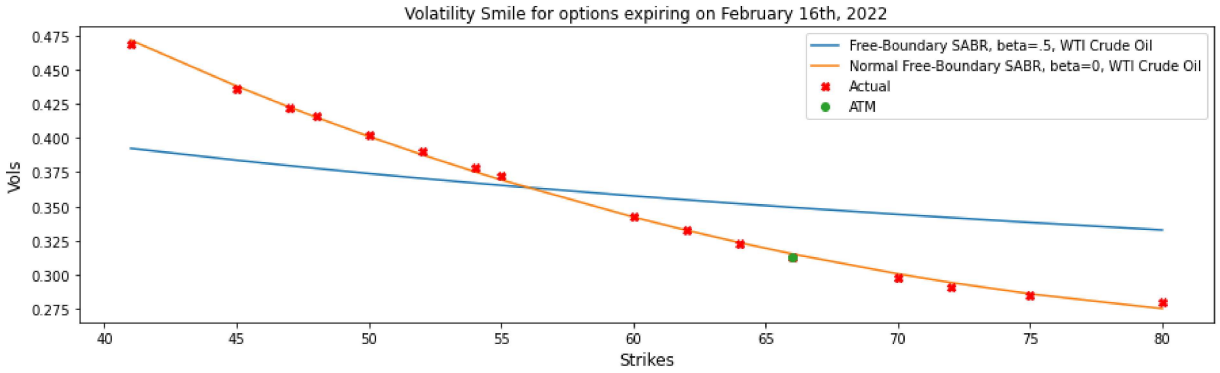


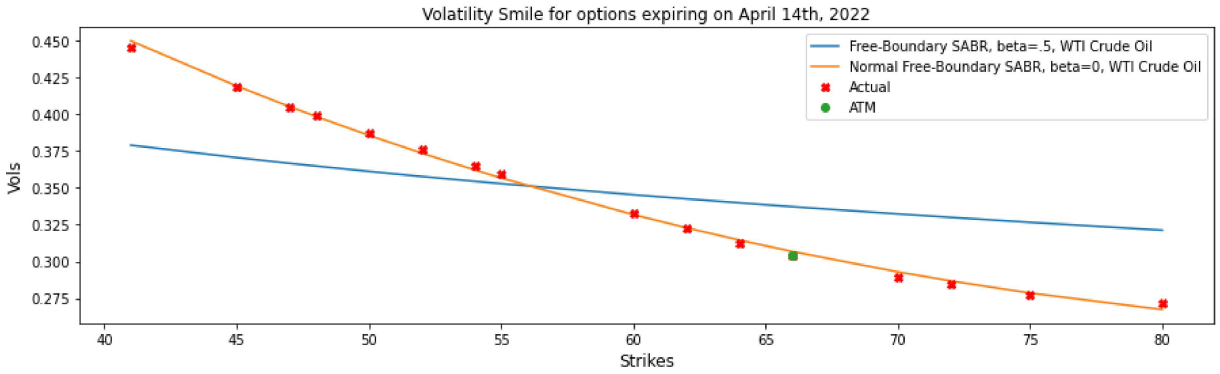
Normal Free-Boundary SABR, beta=0, WTI Crude Oil

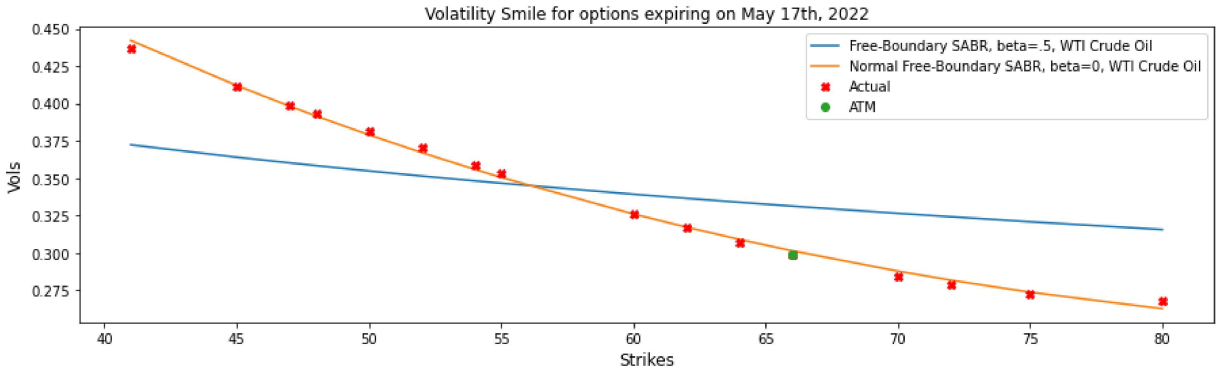












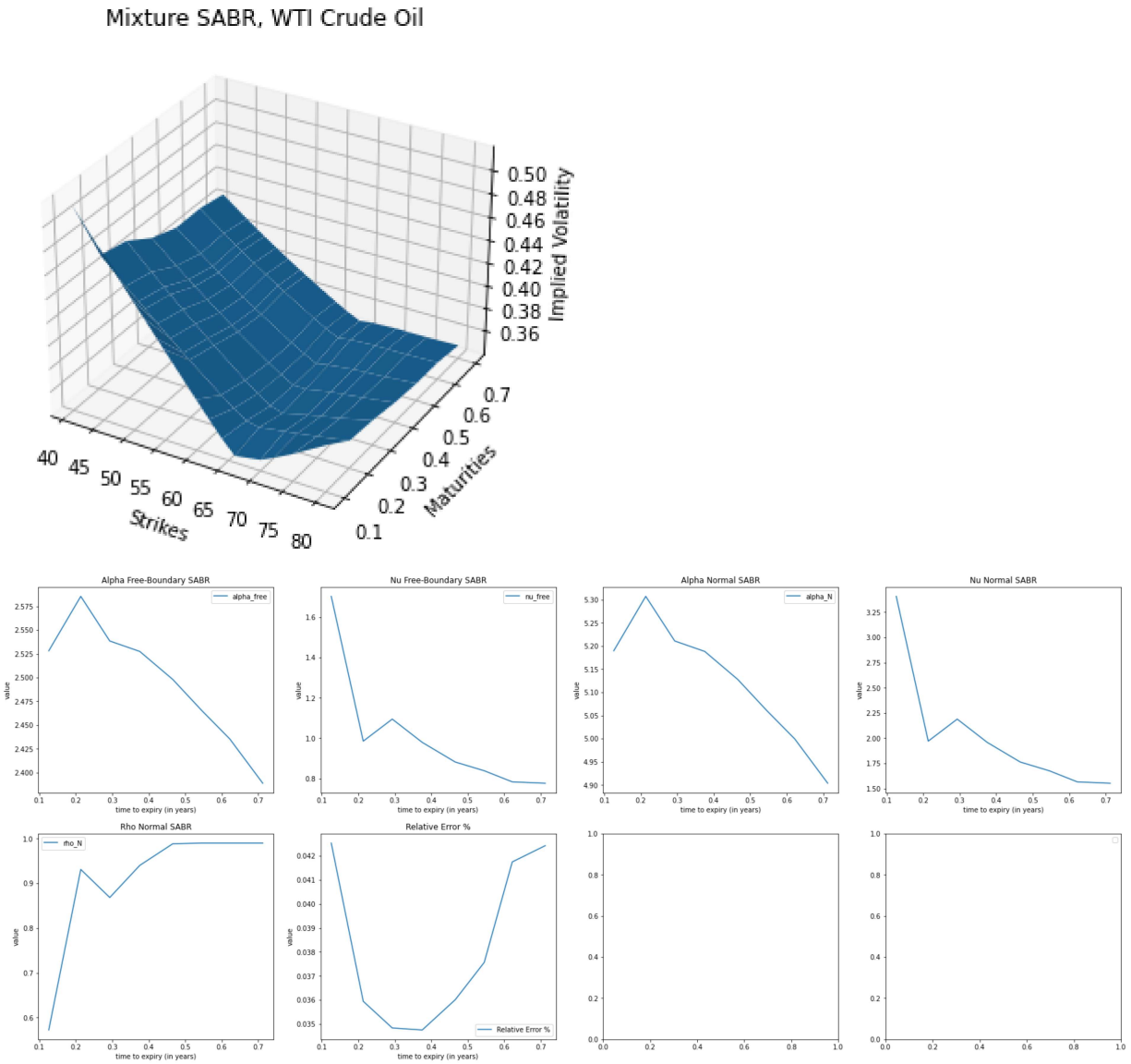
In [16]:

```
# Mixed SABR

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

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

	alpha_free	beta_free	nu_free	rho_free	alpha_N	beta_N	nu_N	rho_N	MSE
October 15th, 2021	2.528157	0.5	1.703207	0.0	2.528157	0.0	3.406414	0.572433	0.042534
November 16th, 2021	2.585460	0.5	0.985183	0.0	2.585460	0.0	1.970367	0.931097	0.035941
December 15th, 2021	2.538470	0.5	1.094539	0.0	2.538470	0.0	2.189079	0.868384	0.034831
January 14th, 2022	2.527511	0.5	0.979080	0.0	2.527511	0.0	1.958159	0.940111	0.034747
February 16th, 2022	2.498050	0.5	0.881706	0.0	2.498050	0.0	1.763412	0.988438	0.036008
March 17th, 2022	2.465342	0.5	0.838396	0.0	2.465342	0.0	1.676792	0.990000	0.037564
April 14th, 2022	2.435463	0.5	0.783394	0.0	2.435463	0.0	1.566788	0.990000	0.041749
May 17th, 2022	2.388834	0.5	0.776446	0.0	2.388834	0.0	1.552892	0.990000	0.042426

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



In [17]:

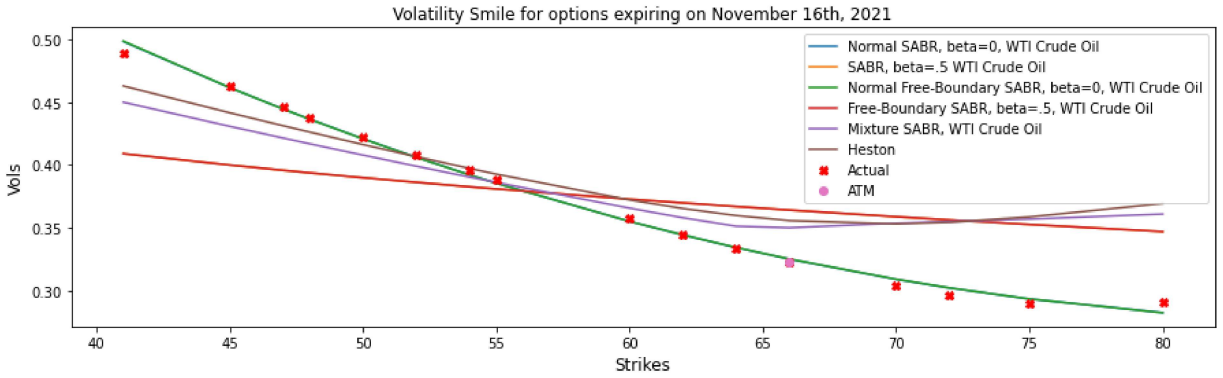
```
# VOLATILITY SMILES ERRORS COMPARISON



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

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

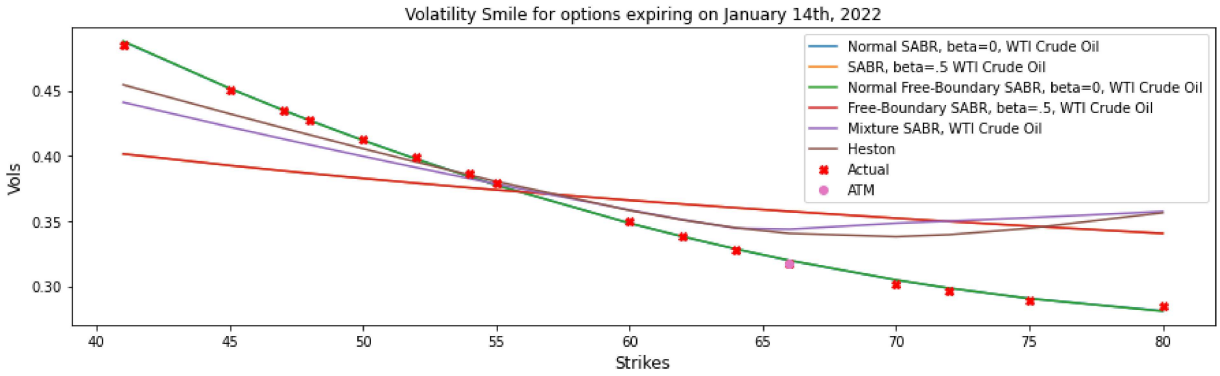
Out[17]: <matplotlib.legend.Legend at 0x187de5861f0>

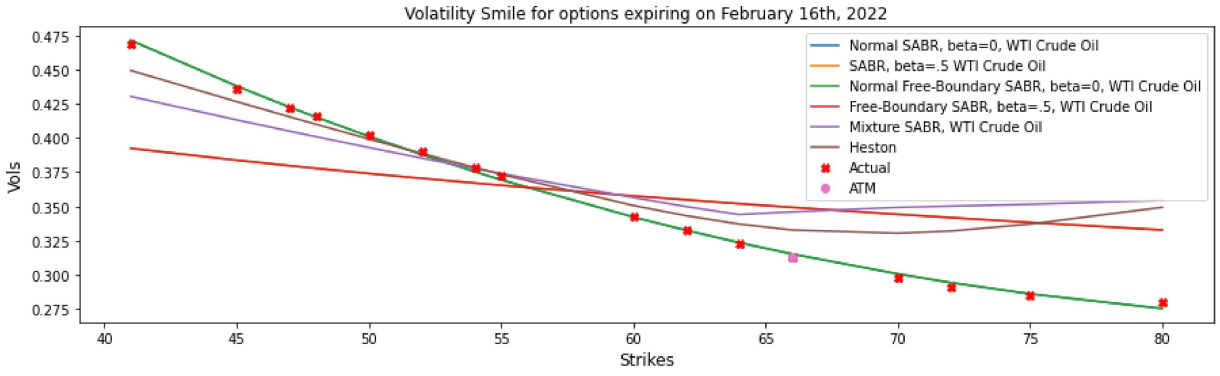
In [18]:

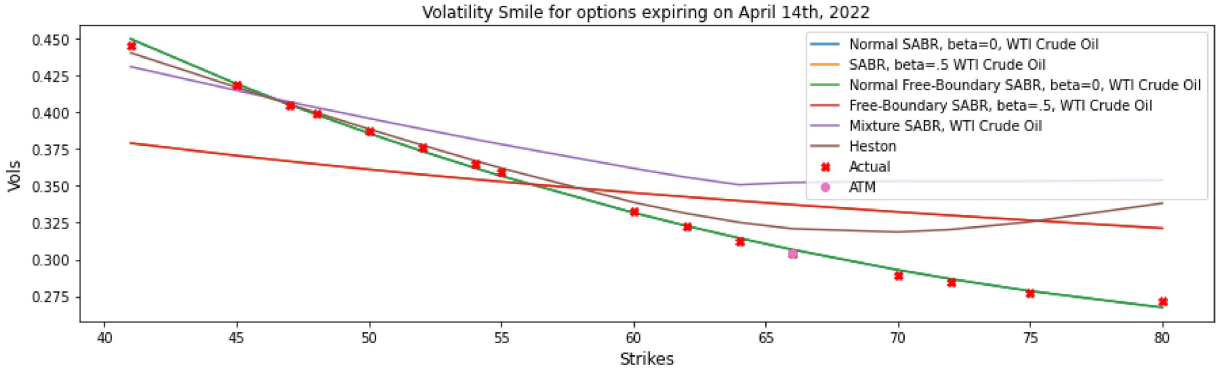
```
# Volatility Smiles Comparisons (Final)

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])
```

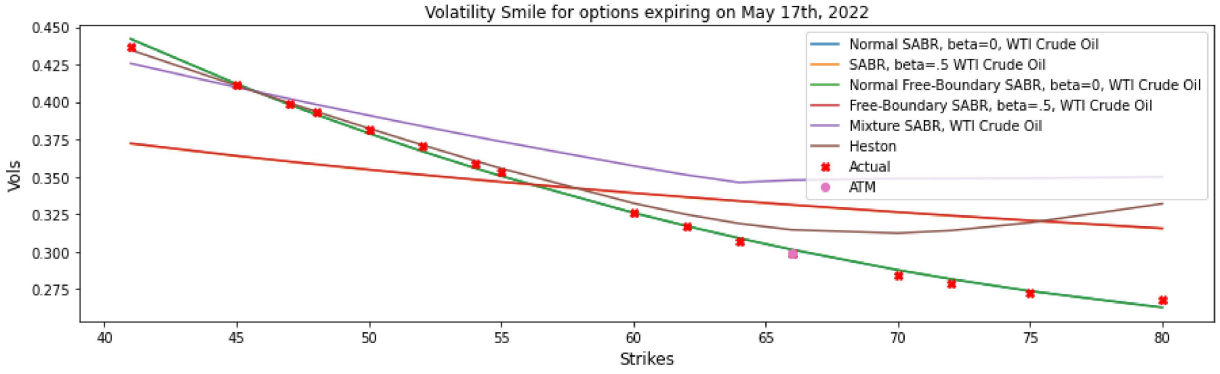
	Error
Normal SABR, beta=0, WTI Crude Oil	0.002759
SABR, beta=.5 WTI Crude Oil	0.045556
Normal Free-Boundary SABR, beta=0, WTI Crude Oil	0.002808
Free-Boundary SABR, beta=.5, WTI Crude Oil	0.045557
Mixture SABR, WTI Crude Oil	0.038225







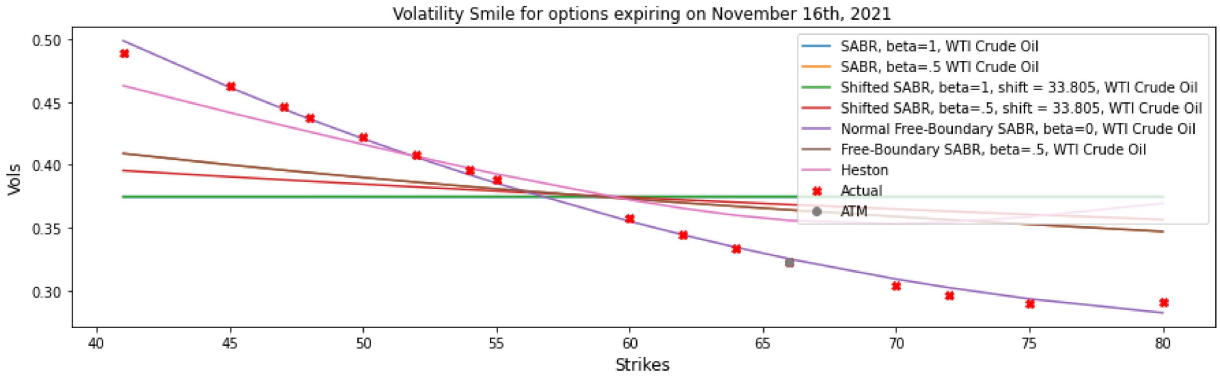


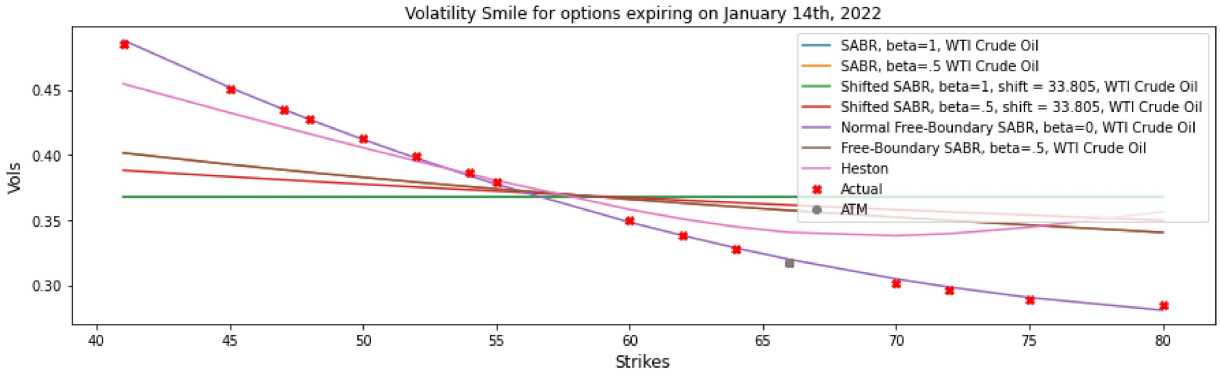


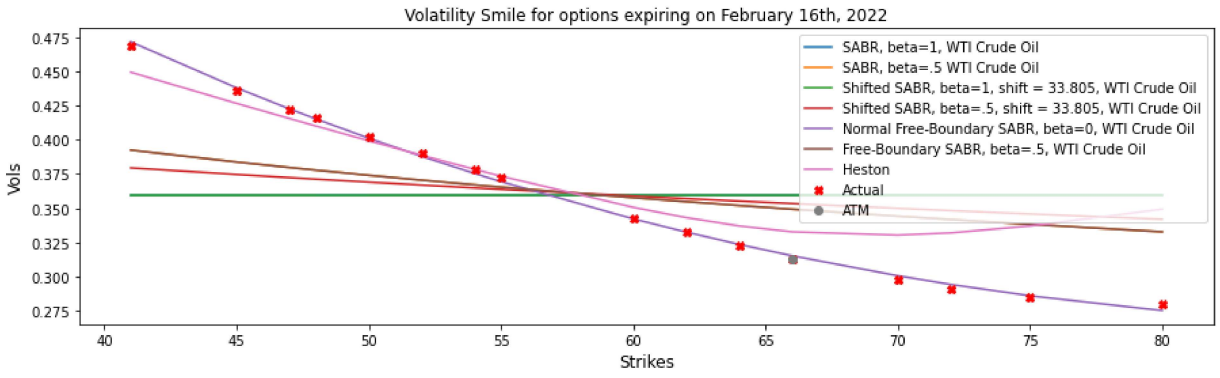
In [19]:

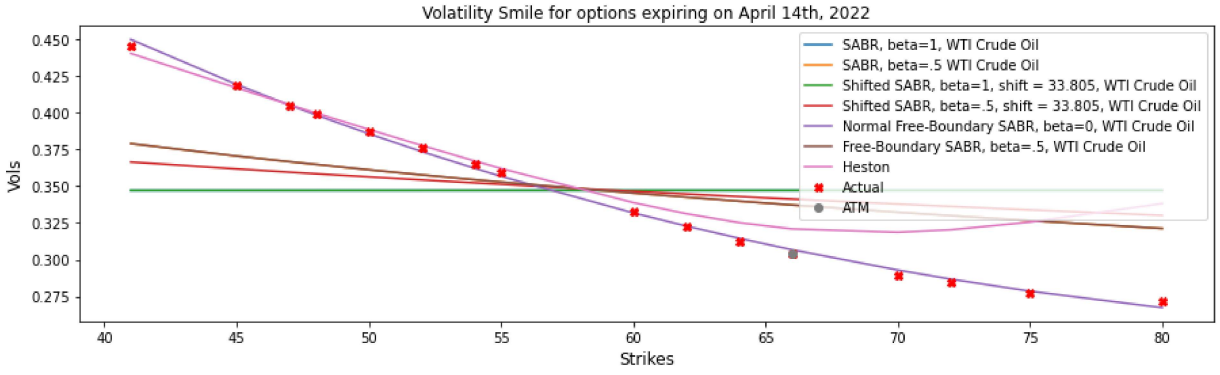
```
# Volatility Smiles Comparisons 2 (Final)

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])
```

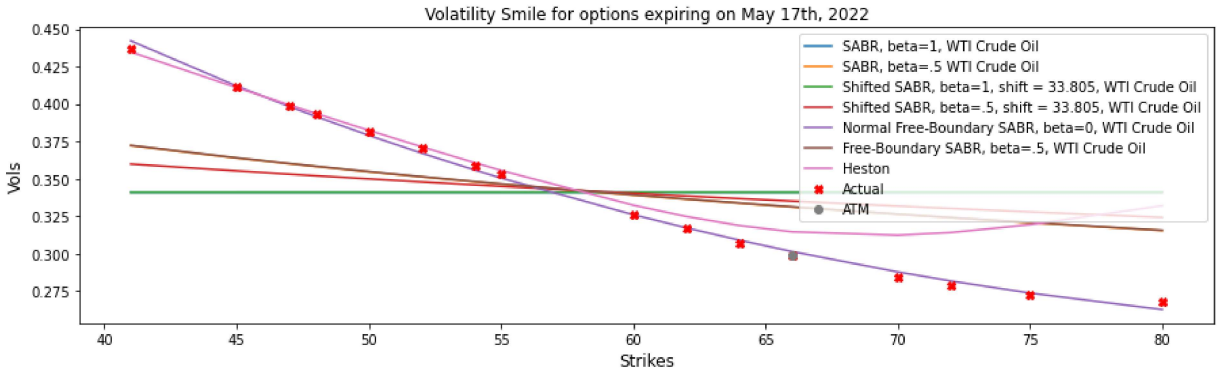
	Error
SABR, beta=1, WTI Crude Oil	0.062773
SABR, beta=.5 WTI Crude Oil	0.045556
Shifted SABR, beta=1, shift = 33.805, WTI Crude Oil	0.062769
Shifted SABR, beta=.5, shift = 33.805, WTI Crude Oil	0.051923
Normal Free-Boundary SABR, beta=0, WTI Crude Oil	0.002808
Free-Boundary SABR, beta=.5, WTI Crude Oil	0.045557





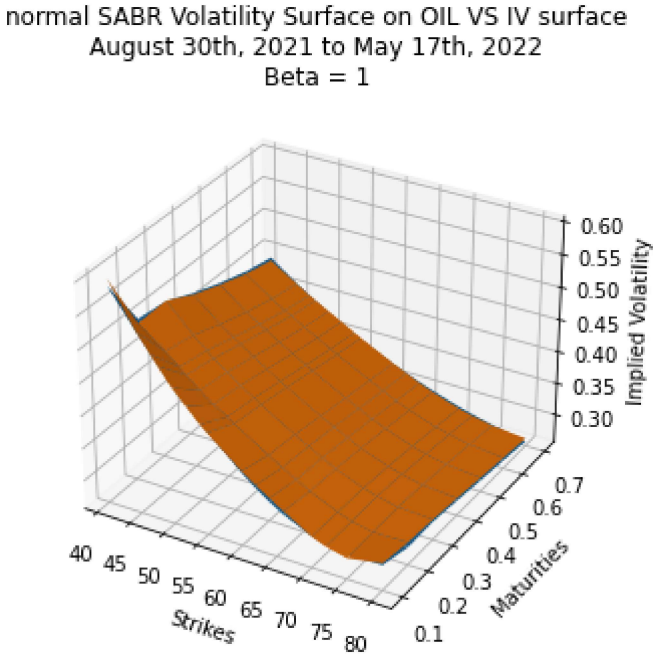




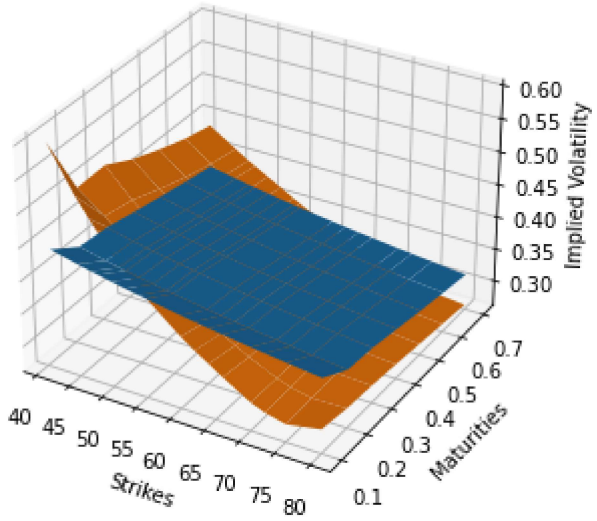


In [20]:

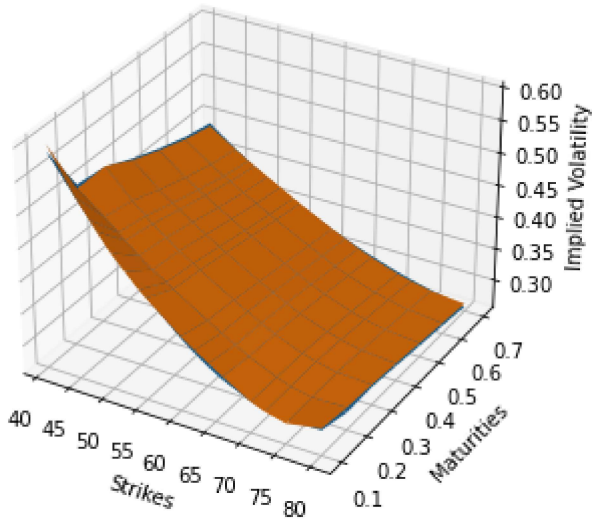
```
# Volatility Surfaces plots comparison

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)
```



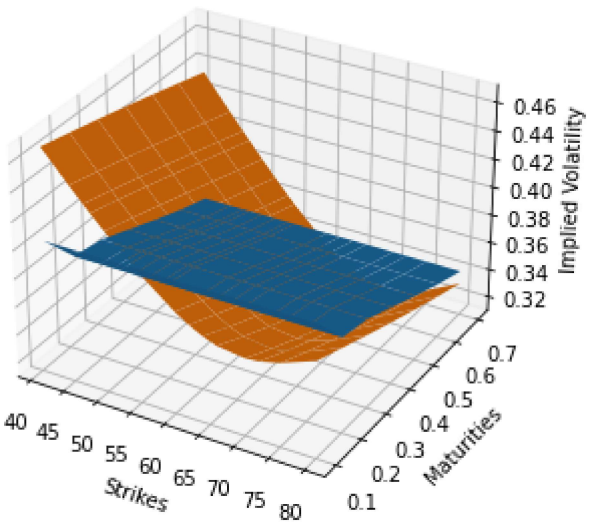
normal SABR Volatility Surface on OIL VS IV surface
August 30th, 2021 to May 17th, 2022
Beta = 0.5

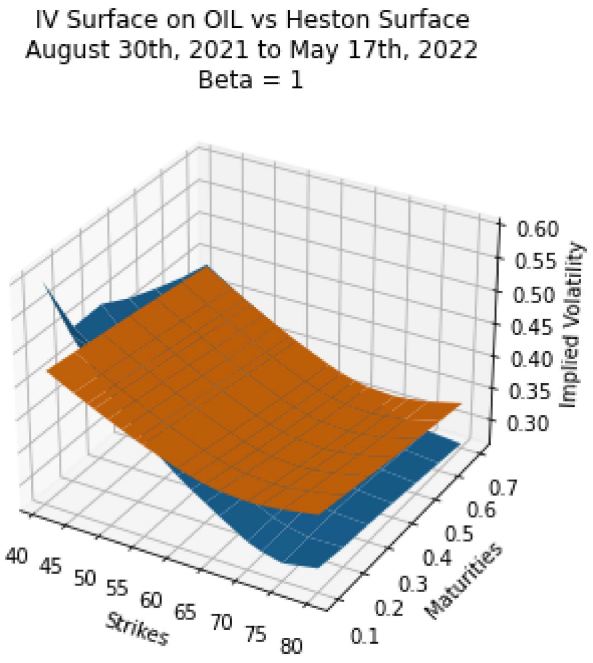


normal SABR Volatility Surface on OIL VS IV surface
August 30th, 2021 to May 17th, 2022
Beta = 0



SABR Volatility Surface on OIL VS Heston surface
August 30th, 2021 to May 17th, 2022
Beta = 1

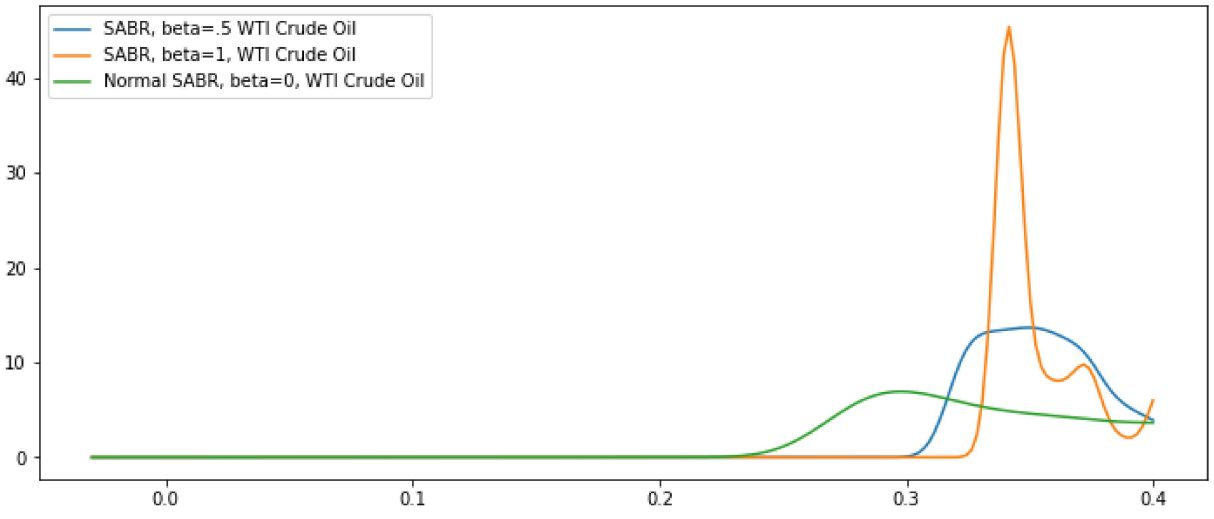




In [21]:

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

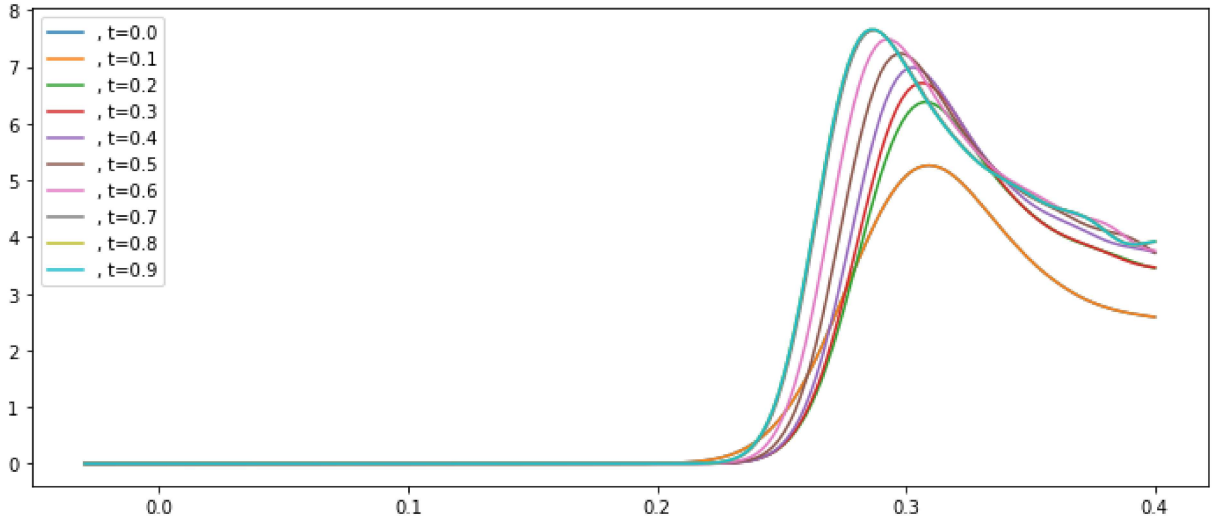


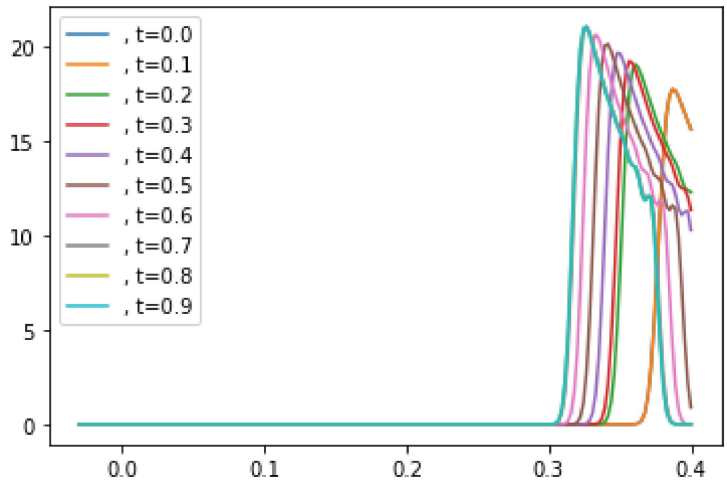
In [22]:

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

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

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

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

    plt.legend()


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

```
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 "
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 "
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 "
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 "
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 "
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 "
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 "
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 "
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 "
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 "
```


, t=0.0
, t=0.1
, t=0.2
, t=0.3
, t=0.4
, t=0.5
, t=0.6
, t=0.7
, t=0.8
, t=0.9
0
1
2
3
4
5
6
7
8
0.0
0.1
0.2
0.3
0.4


, t=0.0
, t=0.1
, t=0.2
, t=0.3
, t=0.4
, t=0.5
, t=0.6
, t=0.7
, t=0.8
, t=0.9
0
5
10
15
20
0.0
0.1
0.2
0.3
0.4

In [ ]: