Using the Kalotay-Williams-Foboozi model to forecast next month's 3 year and 5 year treasury yield rates

In [1]:

```
# importing required libraries

import pandas as pd
import pandas_datareader as pdr
import yfinance as yf
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
```

In [2]:

```
# importing the dataset and naming it to 'interest'
interest = pd.read_csv('yield-curve-rates-1990-2021.csv')
```

In [3]:

```
# converting date column to datetime
interest['Date']= pd.to_datetime(interest['Date'])
```

In [4]:

```
# setting the Date column as index
interest.set_index('Date', inplace=True)
interest.head()
```

Out[4]:

Date	3 year	5 year
2021-12-31	0.97	1.26
2021-12-30	0.98	1.27
2021-12-29	0.99	1.29
2021-12-28	0.99	1.27
2021-12-27	0.98	1.26

In [5]:

```
# since we have to estimate the interest rate path for the next month, we extract monthly data fro
# for this purpose, we use the last value for every month as the interest rate for that particular
interest = interest.resample('M').last()
```

In [6]:

```
# select the 3-year and 5-year Treasury rates
r_3 = interest['3 year']
r_5 = interest['5 year']
```

In [7]:

```
# for the minimization function, we need an initial value of theta and sigma.
# I generated random initial guess and chose the value which has the highest log likelihood

import numpy as np
from scipy.optimize import minimize

# Define log-likelihood function for the KWF model
def log_likelihood(params, rates, delta_t):
    theta, sigma = params
    t = np.arange(1, len(rates) + 1) / 12
    r_t = rates[:-1]
    r_t_plus_delta_t = rates[1:]
    expected_rates = r_t + theta * delta_t
    error = r_t_plus_delta_t - expected_rates
    likelihood = -np.sum(error ** 2) / (2 * sigma ** 2)
    return likelihood

# Example usage for 3-year rates with delta_t=1/12
r_3 = interest['3 year']
delta_t = 1 / 12

# Generate random initial guesses for theta and sigma
np.random.seed(12345)
n_guesses = 1000
theta_guesses = np.random.uniform(0, 0.1, size=n_guesses)
sigma_guesses = np.random.uniform(0.01, 0.05, size=n_guesses)
params_guesses = np.vstack([theta_guesses, sigma_guesses]).T

# Evaluate log-likelihood for each initial guess
log_likelihoods = []
for params in params_guesses:
    log_likelihoods.append(log_likelihood(params, r_3, delta_t))

# Choose the initial guess with the highest log-likelihood
best_guess_idx = np.argmax(log_likelihoods)
best_guess = params_guesses[best_guess_idx]

# Print the best initial guess
print("Best initial guess for 3-year rates: theta = {:.6f}, sigma = {:.6f}".format(*best_guess))
```

```
Best initial guess for 3-year rates: theta = 0.000011, sigma = 0.039194
```

In [8]:

```
import numpy as np
from scipy.optimize import minimize

# Define log-likelihood function for the KWF model
def log_likelihood(params, rates, delta_t):
    theta, sigma = params
    t = np.arange(1, len(rates) + 1) / 12
    r_t = rates[:-1]
    r_t_plus_delta_t = rates[1:]
    expected_rates = r_t + theta * delta_t
    error = r_t_plus_delta_t - expected_rates
    likelihood = -np.sum(error ** 2) / (2 * sigma ** 2)
    return likelihood

# Generate random initial guesses for theta and sigma
np.random.seed(12345)
n_guesses = 1000
theta_guesses = np.random.uniform(0, 0.1, size=n_guesses)
sigma_guesses = np.random.uniform(0.01, 0.05, size=n_guesses)
params_guesses = np.vstack([theta_guesses, sigma_guesses]).T

# Evaluate log-likelihood for each initial guess
log_likelihoods = []
for params in params_guesses:
    log_likelihoods.append(log_likelihood(params, r_5, delta_t))

# Choose the initial guess with the highest log-likelihood
best_guess_idx = np.argmax(log_likelihoods)
best_guess = params_guesses[best_guess_idx]

# Print the best initial guess
print("Best initial guess for 5-year rates: theta = {:.6f}, sigma = {:.6f}".format(*best_guess))
```

```
Best initial guess for 5-year rates: theta = 0.000011, sigma = 0.039194
```

In [9]:

```
import numpy as np
from scipy.optimize import minimize
import matplotlib.pyplot as plt
import math

def expected_rate(r_t, t, delta_t, theta, sigma):
    """Expected spot rate at time t+delta_t given the KWF model."""
    drift = theta * delta_t
    volatility = sigma * np.random.normal(0,1) * math.sqrt(delta_t)
    return r_t * np.exp(drift + volatility)

def log_likelihood(params, rates, delta_t):
    """Log-likelihood function for the KWF model."""
    theta, sigma = params
    t = np.arange(1, len(rates) + 1) / 12
    r_t = rates[:-1]
    r_t_plus_delta_t = rates[1:]
    expected_rates = expected_rate(r_t, t, delta_t, theta, sigma)
    return -np.sum((r_t_plus_delta_t - expected_rates)**2)

# Example usage for 3-year rates with delta_t=1/12
r_3 = interest['3 year']
delta_t = 1/12
params_3 = minimize(log_likelihood, [0.000011,  0.039194], args=(r_3, delta_t))['x']
theta_3, sigma_3 = params_3
print("Optimized values for 3-year rates: theta = {:.6f}, sigma = {:.6f}".format(theta_3, sigma_3))

# Example usage for 5-year rates with delta_t=1/12
r_5 = interest['5 year']
delta_t = 1/12
params_5 = minimize(log_likelihood, [0.000011,  0.039194], args=(r_5, delta_t))['x']
theta_5, sigma_5 = params_5
print("Optimized values for 5-year rates: theta = {:.6f}, sigma = {:.6f}".format(theta_5, sigma_5))
```

```
Optimized values for 3-year rates: theta = 0.000017, sigma = 0.039195
Optimized values for 5-year rates: theta = 0.000036, sigma = 0.039200
```

In [ ]:

```
# the model thus gives us the optmizied value of theta which is the expected logarithmic change in
# and sigma which is the expected standard deviation for 3 year and 5 year treasury rates
```

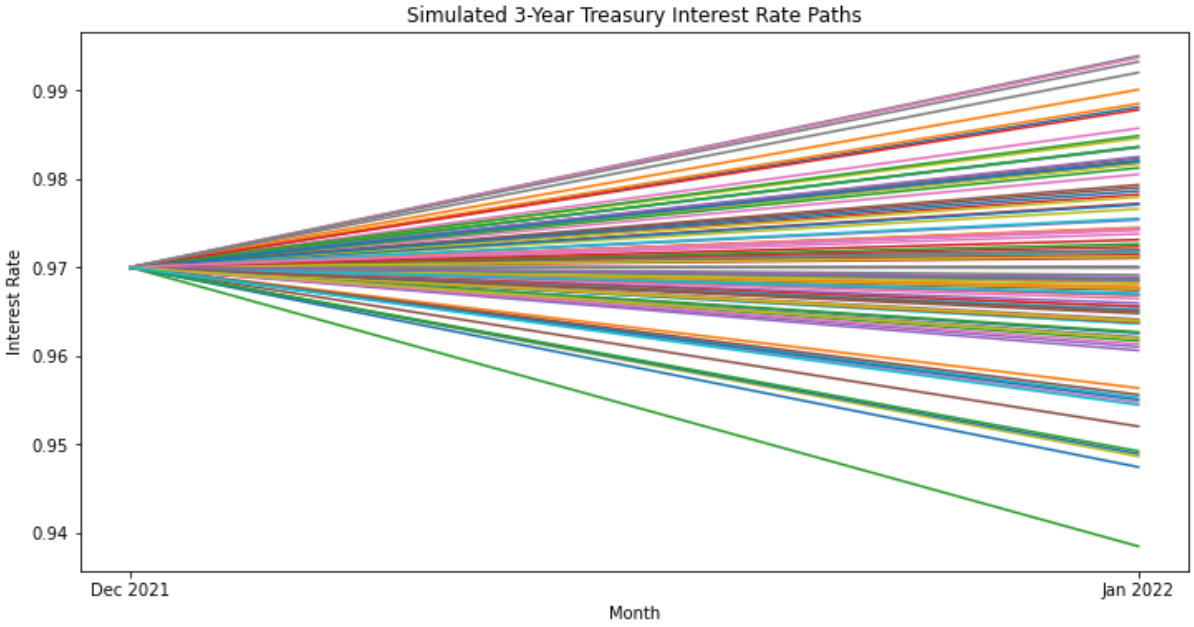
In [10]:

```
# 3 year treasury

import calendar
simulated_3yr_rates = []
for i in range(100):
    sim_rates = [r_3.iloc[-1]]
    for j in range(1, 2):
        t = j * delta_t
        sim_rates.append(expected_rate(sim_rates[-1], t, delta_t, theta_3, sigma_3))
        simulated_3yr_rates.append(sim_rates)

# create x-axis labels with month names
month_names = ['Dec 2021','Jan 2022']

# plot the simulated rates on the same graph
plt.figure(figsize=(12,6))
for sim_rates in simulated_3yr_rates:
    plt.plot(month_names, sim_rates)
plt.xlabel('Month')
plt.ylabel('Interest Rate')
plt.title('Simulated 3-Year Treasury Interest Rate Paths')
plt.show()
```



In [11]:

```
# 5 year treasury

simulated_5yr_rates = []
for i in range(100):
    sim_rates = [r_5.iloc[-1]]
    for j in range(1, 2):
        t = j * delta_t
        sim_rates.append(expected_rate(sim_rates[-1], t, delta_t, theta_5, sigma_5))
        simulated_5yr_rates.append(sim_rates)

# create x-axis labels with month names
month_names = ['Dec 2021','Jan 2022']

# plot the simulated rates on the same graph
plt.figure(figsize=(12,6))
for sim_rates in simulated_5yr_rates:
    plt.plot(month_names, sim_rates)
plt.xlabel('Month')
plt.ylabel('Interest Rate')
plt.title('Simulated 5-Year Treasury Interest Rate Paths')
plt.show()
```