HULL-WHITE AND EIOPA RISK-FREE CURVE TERM STRUCTURE COVERAGE AND VARIANCE CHECK

The Hull-White model (HW) is a very popular choice when modeling interest rates. For example economic scenario generators can use the HW model to simulated risk-free curves. This notebook is devided into 4 sections. The first section uses the EIOPA Risk-Free-Rate (RFR) calibration to produce a yield curve. In the second section, this yield curve is used to produce a number of stochastic scenarios using a simple implementation of the HW model. The third section checks the goodness of fit of the simulated paths compared to the input term structure. The final section calculates the closed form variance and compares it to the simulated variance.

HW model is presented in two different forms in literature. This script starts from the folowing diferential equation:

$$dr(t) = \big(\theta(t) - ar(t)\big)dt + \sigma dW(t)$$

Where:

- t is time.
- r(t) is the short rate at time t.
- $\theta(t)$ is the time-dependent parameter theta.
- a is the mean reversion speed parameter.
- σ is the volatility parameter.
- $W(t)$ is a standard Brownian motion.

Summary

The goal of this script is to answer the following questions:

- Is the HW simulation correctly implemented
- Is the company is using a sufficienty large number of stochastic scenarios to aquaretly cover the term structure.
- Are the parameters used in the simulation correctly implemented.

Table of Contents

Note on Smith & Wilson algorithm

The validation of RFR rate is performed in another script that checks the correct calibration of the EIOPA's RFR output, also available on OSM Github as a Jupyter notebook.

This example uses a modified Smith&Wilson implementation (The original implementation is availible on GitHub:

- Python
- Matlab
- JavaScript

Limitations of the implementation

This script generates a synthetic dataset of ESG scenarios. It also simplifies the day-count convention with the assumption that each month represents 1/12-th of a year. The discretisation and numeric integration uses the simplest Euler scheme and the rectangular rule respectively.

Only two checks are performed to verify the correctnes and the properties of the result.

Only a single EIOPA curve is used (No volatility adjustment, Euro curve).

Success criteria

The following success criteria is defined:

- Maximum difference between the average simulated yield curve and the one provided as input is less than 0.1 bps.
- Average difference between the average simulated curve and the one provided as input is less than 0.05 bps.
- Maximum difference between the empirical volatility of the simulated yield curve and the theoretical volatility is less than 0.1 bps.
- Average difference between the empirical volatility of the simulated yield curve and the theoretical volatility is less than 0.05 bps.

In [3]:
```
statistics_mean_max_diff_in_bps = 500
statistics_mean_average_diff_in_bps = 100
statistics_vol_max_diff_in_bps = 2
statistics_vol_average_diff_in_bps = 1
```

The success function that is called at the end of every check is the following (With the threasholds that are defined above).

In [4]:
```
def SuccessTest(TestStatistics, threshold_max, threshold_mean):
    out1 = False
    out2 = False
    if max(TestStatistics)<threshold_max:
        print("Test passed")
        out1 = True
    else:
        print("Test failed")

    if np.mean(TestStatistics)<threshold_mean:
        print("Test passed")
        out2 = True
    else:
        print("Test failed")
    return [out1, out2]
```

This implementation looks at two kinds of test statistics. The average deviation and the maximum deviation.

The average deviation is defined as:

$$S_{AVERAGE} = \frac{1}{N}\sum_{t=0}^{T}|x_{THEORETICAL}(t) - x_{EST}(t)|$$

The maximum deviation is defined as:

$$S_{MAX} = \max_{t}|x_{THEORETICAL}(t) - x_{EST}(t)|$$

Where `N` is the number of time increments and `T` is the maximum maturity

Data requirements

This script contains the EIOPA RFR published for March 2023. The publication can be found on

the EIOPA RFR website.
The observed maturities `m_obs` and the calibrated vector `Qb` can be found in the published Excel workbook *EIOPA_RFR_20230331_Qb_SW.xlsx*.

The Euro curve without the volatility adjustment (VA) is used. It can be found in the sheet *SW_Qb_no_VA*. This example is focused on the EUR curve, but can be easily extended to other curves.

Target maturities (`t_obs`), the additional parameters (`UFR` and `alpha`), and the given curve can be found in the Excel *EIOPA_RFR_20230331_Term_Structures.xlsx*. Sheet *RFR_spot_no_VA* looks at the curve without the Volatility Adjustment.

Hull-White Parameters

This example does not have a specific dataset in mind therefore, a set of dummy parameters is used for the Hull-White model:

In [5]:
```
NoOfPaths = 20000 # Number of stochastic scenarios
NoOfSteps = 600  # Number of equidistand discrete modelling points (50*12 = 600)
T = 50.0         # Time horizon in years (A time horizon of 50 years; T=50)
a = 0.02         # Hull-White mean reversion parameter a
sigma = 0.02     # Hull-White volatility parameter sigma
epsilon = 0.01   # Incremental distance used to calculate for numerical approximatio
                 # of for example the instantaneous spot rate (Ex. 0.01 will use an
                 # of 0.01 as a discreete approximation for a derivative)
```

External dependencies

This script uses a set of well known Python packages that are commonly used in finance. Numpy for the mathematical operation and matrix multiplication, Pandas for its table manipulation functionality and Matplotlib for charts.

In [6]:
```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [43]:
```
print(f"The Numpy version is {np.__version__}.")
```

```
The Numpy version is 1.20.3.
```

In [44]:
```
print(f"The Pandas version is {pd.__version__}.")
```

```
The Pandas version is 1.3.4.
```

In [45]:
```
print(f"The Matplotlib version is {mpl.__version__}.")
```

```
The Matplotlib version is 3.4.3.
```

Importing data

This section specifies the names of external files with the information about the term structure. In particular, the Smith-Wilson calibration parameters and the term structure that is used to generate the stochastic paths.

In [10]:
```
selected_param_file = 'Param_no_VA.csv'
selected_curves_file = 'Curves_no_VA.csv'
```

In [11]:
```
param_raw = pd.read_csv(selected_param_file, sep=',', index_col=0)
```

EIOPA provides curves for multiple countries. In this script, Slovenia is used as an example, but any other country can be substituted into the country variable.

In [12]:
```
country = "Slovenia"
```

In [13]:
```
maturities_country_raw = param_raw.loc[:,country+"_Maturities"].iloc[6:]
param_country_raw = param_raw.loc[:,country + "_Values"].iloc[6:]
extra_param = param_raw.loc[:,country + "_Values"].iloc[:6]
```

In [14]:
```
relevant_positions = pd.notna(maturities_country_raw.values)
```

In [15]:
```
maturities_country = maturities_country_raw.iloc[relevant_positions]
```

In [16]:
```
Qb = param_country_raw.iloc[relevant_positions]
```

In [17]:
```
curve_raw = pd.read_csv(selected_curves_file, sep=',',index_col=0)
```

In [18]:
```
curve_country = curve_raw.loc[:,country]
```

Smith & Wilson

In this section, the EIOPA parameters and the EIOPA RFR curve are processed into the correct form that will be used for the rest of the script

Smith & Wilson parameters

In [19]:
```
# Maturity of observations:
m_obs = np.transpose(np.array(maturities_country.values))

# Ultimate froward rate ufr represents the rate to which the rate curve will
# converge as time increases:
ufr = extra_param.iloc[3]/100

# Convergence speed parameter alpha controls the speed at which the curve
# converges towards the ufr from the last liquid point:
alpha = extra_param.iloc[4]

# For which maturities do we want the SW algorithm to calculate the rates.
# In this case, for every year up to 150:
m_target = np.transpose(np.arange(1,151))

# Qb calibration vector published by EIOPA for the curve calibration:
Qb = np.transpose(np.array(Qb.values))
```

Smith & Wilson calculation functions

An independent version of the Smith&Wilson algorithm is implemented. To facilitate this, two functions are taken from the publicly available repository that already contains the complete implementation. It is then modified to accept the product of `Q*b` instead of the calibration vector `b`.

In [20]:

```
def sw_extrapolate(m_target, m_obs, Qb, ufr, alpha, epsilon = 0.00001):
    """
    Interpolate or extrapolate rates for targeted maturities using a
    Smith-Wilson algorithm.
       sw_extrapolate(m_target, m_obs, Qb, ufr, alpha, epsilon) calculates the rates
           maturities specified in M_Target using the calibration vector b.
    Args:
        m_target (ndarray): k x 1 array of targeted bond maturities.
        m_obs (ndarray): n x 1 array of observed bond maturities.
        Qb (ndarray): n x 1 array. Calibration vector.
        ufr (float): Ultimate forward rate.
        alpha (float): Convergence speed parameter.
        epsilon (float): Increment to calculate the instantaneous spot rate.

    Returns:
        ndarray: k x 1 array of targeted rates for zero-coupon bonds with
            maturity specified in m_target.

    For more information see
    https://www.eiopa.europa.eu/sites/default/files/risk_free_interest_rate
        /12092019-technical_documentation.pdf
    """

    def sw_heart(u, v, alpha):
        """
        Calculate the heart of the Wilson function. sw_heart(u, v, alpha)
        calculates the matrix H (Heart of the Wilson function) for maturities
        specified by vectors u and v. The formula is taken from the EIOPA technical
        specifications paragraph 132.

        Args:
            u (ndarray): n_1 x 1 vector of maturities.
            v (ndarray): n_2 x 1 vector of maturities.
            alpha (float): Convergence speed parameter.

        Returns:
            ndarray: n_1 x n_2 matrix representing the Heart of the Wilson function.
        """
        u_mat = np.tile(u, [v.size, 1]).transpose()
        v_mat = np.tile(v, [u.size, 1])
        return 0.5 * (alpha * (u_mat + v_mat) + np.exp(-alpha * (u_mat + v_mat))
                      - alpha * np.absolute(u_mat - v_mat) -
                      np.exp(-alpha * np.absolute(u_mat - v_mat)))

    # Heart of the Wilson function from paragraph 132
    h = sw_heart(m_target, m_obs, alpha)

    # Discount pricing function for targeted maturities from paragraph 147
    p = np.exp(-np.log(1 + ufr) * m_target) + np.diag(np.exp(-np.log(1 + ufr)
                                                             * m_target)) @ h @ Qb

    # If the first element of m_target is zero, replace it with time "epsilon"
    # to avoid division by zero error.
    m_target[0] = epsilon if m_target[0] == 0 else m_target[0]

    return p ** (-1 / m_target) - 1
```

Hull-White interest rate simulation functions

In this step, the Hull-White simulation is implemented. The calculation of a discounted ZCB bond, calculation of a forward rate, the parameter θ and finally the generation of multiple sample paths.

The function P0t_f() calculates the price of a zero-coupon bond (ZCB) based on the given yield and maturity.

$$P0t = e^{-y0t*t}$$

Where:

- `t` is the time of interest expressed as a year fraction (Ex. for 18 months, `t` = 1.5).
- `y0t` is the yield for single payoff at time `t`.
- `P0t` is the price of a zero-coupon bond issued in time `0` with a notional amount of 1 and maturity at time `t`.

In [21]:

```
def P0t_f(t, m_obs, Qb, ufr, alpha):
    """
    Calculates the price of a zero-coupon bond issued at time 0,
    for a given maturity 't', using the Smith-Wilson extrapolation technique.
    P0t_f(t, m_obs, Qb, ufr, alpha)

    Args:
        t (float or ndarray): vector (or a single number) of maturities represented
            as time fraction (Ex. for 18 months; t=1.5).
        m_obs (ndarray): n x 1 array of observed bond maturities used for
            calibration.
        Qb (ndarray): n x 1 calibration vector of the Smith-Wilson algorithm
           calculated on observed bonds.
        ufr (float): Ultimate forward rate parameter for the Smith-Wilson algorithm.
        alpha (float): Convergence speed parameter for the Smith-Wilson algorithm.

    Returns:
        ndarray: n x 1 the price of zero-coupon bonds issued at time 0 with a notiona
            and maturity t.

    Example of use
        m_obs = np.array([1, 2, 3, 5, 7, 10, 15, 20, 30])
        Qb = np.array([0.02474805, 0.02763133, 0.02926931, 0.0302894, 0.03061605,
           0.03068016, 0.03038397, 0.02999401, 0.02926168])
        ufr = 0.042
        alpha = 0.05

        # For a single maturity
        t = 5
        price = P0t_f(t, m_obs, Qb, ufr, alpha)
        print(f"Price of zero-coupon bond with maturity {t} years is: {price}")

        # For multiple maturities
        t = [1, 3, 5, 10]
        prices = P0t_f(t, m_obs, Qb, ufr, alpha)
        print("Prices of zero-coupon bonds with maturities", t, "years are:")
        print(prices)

    Implemented by Gregor Fabjan from Open-Source Modelling on 29/07/2023
    """

    if isinstance(t, np.ndarray): # If the input is a numpy array
        y0t = sw_extrapolate(np.transpose(t), m_obs, Qb, ufr, alpha)
        out = np.exp(-y0t*np.transpose(t))
    else:# If the input is a single maturity given as a number
        y0t = sw_extrapolate(np.transpose([t]), m_obs, Qb, ufr, alpha)
        out = np.exp(-y0t*[t])
    return out
```

The function f0t() calculates the instantaneous forward rate at time `t` using a numerical approximation with a step size given by the parameter `epsilon`. The calculation of the instantaneous forward rate relies on the difference between two zero-coupon bond prices. One maturing at time $t + \epsilon$ and another at time $t - \epsilon$.

The centered finite difference method estimates the instantaneous forward rate using the following approximation:

$$f(0,t) \approx -\frac{\log(P(0,t+\epsilon)) - \log(P(0,t-\epsilon))}{2\epsilon}$$

In [22]:

```
def f0t(t, P0t, epsilon):
    """
    Calculates the instantaneous forward rate for time t given the zero-coupon
    bond price function P0t, using the centered finite difference method.
    f0t(t, P0t, epsilon)

    Args:
        t (float): Time at which the instantaneous forward rate is calculated.
        P0t (function): Function that takes a float argument `t` and
            returns the price of a zero-coupon bondissued at time 0 with maturity `t`
            notional amount 1.
        epsilon (float): Step size for the centered finite difference method.

    Returns:
        float: The instantaneous forward rate at time t, calculated using the
            centered finite difference method.
    """

    p_plus = P0t(t + epsilon)
    p_minus = P0t(t - epsilon)
    return -(np.log(p_plus) - np.log(p_minus)) / (2 * epsilon)
```

The function HW_theta calculates the time depened parameter θ. In this implementation, this parameter is calibrated to the term stucture obtained using the EIOPA RFR. The term structure is specified by the P0t function that is passed as argument.

The parameter `theta` is calibrated using the following relation:

$$\theta(t) = \frac{\partial f(0,t)}{\partial t} + af(0,t) + \frac{\sigma^2}{2a}\left(1 - e^{-2at}\right)$$

Where:

- `t` is the time at which we wish to calibrate θ to the term structure. (Ex. 18 months means t = 1.5).
- a is the a parameter in the Hull-White model.
- σ is the volatility parameter σ of the Hull-White model.
- $f(0,t)$ is the instantaneous forward rate at time `t` estimated at time 0.
- $\theta(t)$ is the time dependet parameter θ of the Hull-White model.

In [23]:

```
def HW_theta(a, sigma, P0t, eps):
    """
    Calculates the theta value for the Hull-White model
    using a numeric approximation of the instantaneous forward rate
    and the spot rate.

    Args:
        a (float): Mean reversion rate parameter a.
        sigma (float): Volatility parameter sigma.
        P0t (function handle): Function that calculates the price of a
            zero-coupon bond as a function of time.
        eps (float): Increment of time used in the numeric calculation of the
            derivative of the instantaneous forward rate.

    Returns:
        theta (function): Function that returns the parameter theta of
            Hull-White model at the time t implied by the calibration.
    """

    def theta(t):
        insta_forward_term = (f0t(t+eps, P0t, eps)
                                        -f0t(t-eps,P0t,eps))/(2.0*eps)

        forward_term = a*f0t(t, P0t, eps)
        variance_term = sigma**2/(2.0*a)*(1.0-np.exp(-2.0*a*t))
        return insta_forward_term + forward_term + variance_term
    return theta
```

In [24]:

```
def Paths(NoOfPaths, NoOfSteps, T, P0t, a, sigma, epsilon):
    """
    Simulates a series of stochastic interest rate paths using the Hull-White model.

    Args:
        NoOfPaths (int): number of paths to simulate.
        NoOfSteps (int): number of time steps per path.
        T (float): end of the modelling window (in years).
            (Ex. a modelling window of 50 years means T=50).
        P0t (function): function that calculates the price of a
            zero coupon bond issued at time 0 that matures at time t, with a
            notional amount 1 and discounted using the assumed term structure.
        a (float): mean reversion speed parameter a of
            the Hull-White model.
        sigma (float): volatility parameter sigma of the Hull-White model.
        epsilon (float): size of the increment used for finite
            difference approximation.

    Returns:
        dict: A dictionary containing arrays with time steps, interest rate paths,
            and bond prices.
            time (array): array of time steps.
            R (array): array of interest rate paths with
              shape (NoOfPaths, NoOfSteps+1).
            M (array): array of bond prices with
              shape (NoOfPaths, NoOfSteps+1).

    Implemented by Gregor Fabjan from Open-Source Modelling on 29/07/2023.

    Original inspiration: https://www.youtube.com/watch?v=BIZdwUDbnDo
    """

    # Initial instantaneous forward rate at time t-> 0 (also spot rate at time 0).
    # r(0) = f(0,0) = - partial derivative of log(P_mkt(0, epsilon) w.r.t epsilon)
    r0 = f0t(epsilon, P0t, epsilon)

    # Calculation of theta = 1/a * partial derivative of f(0,t) w.r.t. t
    # + f(0,t) + sigma^2/(2 a^2)* (1-exp(-2*a*t)).
    theta = HW_theta(a, sigma, P0t, epsilon)

    # Generate the single source of random noise.
    Z = np.random.normal(0.0, 1.0, [NoOfPaths, NoOfSteps])

    # Initialize arrays

    # Vector of time moments.
    time = np.linspace(0, T, NoOfSteps+1)

    W = np.zeros([NoOfPaths, NoOfSteps+1])

    # Initialize array with interest rate increments
    R = np.zeros([NoOfPaths, NoOfSteps+1])

    # First interest rate equals the instantaneous forward (spot)
    # rate at time 0.
    R[:, 0] = r0
    dt = T/float(NoOfSteps) # Size of increments between two steps
```

```
    for iTime in range(1, NoOfSteps+1): # For each time increment
        # Making sure the samples from the normal distribution have a mean of 0
        # and variance 1
        if NoOfPaths > 1:
            Z[:, iTime-1] = (Z[:, iTime-1]-np.mean(Z[:, iTime-1]))/np.std(Z[:, iTime

        # Apply the Euler-Maruyama discretisation scheme for the Hull-White model
        # at each time increment.
        W[:, iTime] = W[:, iTime-1] + np.power(dt, 0.5)*Z[:, iTime-1]
        noise_term = sigma* (W[:, iTime]-W[:, iTime-1])
        rate_term = (theta(time[iTime-1])-a*R[:, iTime-1])*dt
        R[:, iTime] = R[:, iTime-1] + rate_term + noise_term

    # Vectorized numeric integration using the Euler integration method .
    M = np.exp(-0.5 * (R[:, :-1] + R[:, 1:]) * dt)
    M = np.insert(M, 0, 1, axis=1).cumprod(axis=1)

    # Output is a dataframe with time moment, the interest rate path and the price
    # of a zero coupon bond issued at time 0 that matures at the selected time
    # moment with a notional value of 1.
    paths = {"time":time, "R":R, "M":M}
    return paths
```

Test 1; Comparison between ESG output and assumed term structure

In this final section, the ESG output is generated and visualy compared to the assumed term structure. The Hull-White model belongs to the family of modells known as HJM. A well known property of this family of models is that the average rate over all simulations should be equal to the input term structure by construction. However in practice, this is often not exactly the case especially in the tails of the curve and if the number of paths is small. The visual check shows the average simulation compared to the input term structure (interpolated/extrapolated by the Smith-Willson algorithm).

In [25]:
```
def mainCalculation(NoOfPaths, NoOfSteps, T, a, sigma, P0t, epsilon):
    """
    Calculates and plots the prices of zero-coupon bonds (ZCB) calculated
    using the Hull-White model`s analytical formula and the Monte Carlo simulation.

    Args:
        NoOfPaths (int): number of Monte Carlo simulation paths.
        NoOfSteps (int): number of time steps per path.
        T (float): length in years of the modelling window (Ex. 50 years means t=50)
        a (float): mean reversion rate parameter a of the Hull-White model.
        sigma (float): volatility parameter sigma of the Hull-White model.
        P0t (function): function that calculates the price of a zero coupon bond issu
           at time 0 that matures at time t, with a notional amount 1 and discounted
           the assumed term structure.
        epsilon (float): the size of the increment  used for finite difference approx

    Returns:
        t XXX: time increments.
        P XXX: average of the sumulated paths.
        implied_term_structure XXX: term structure provided as input into the HW simu
    """

    paths = Paths(NoOfPaths, NoOfSteps, T, P0t, a, sigma, epsilon)
    M = paths["M"]
    t = paths["time"]
    implied_term_structure = P0t(t)
    # Compare the price of an option on a ZCB from Monte Carlo and the analytical exp
    P = np.zeros([NoOfSteps+1])
    for i in range(0, NoOfSteps+1):
        P[i] = np.mean(M[:, i])

    return [t, P, implied_term_structure]
```

In [26]:
```
# Zero coupon bond prices calculated using the assumed term structure
P0t = lambda t: P0t_f(t, m_obs, Qb, ufr, alpha)
```

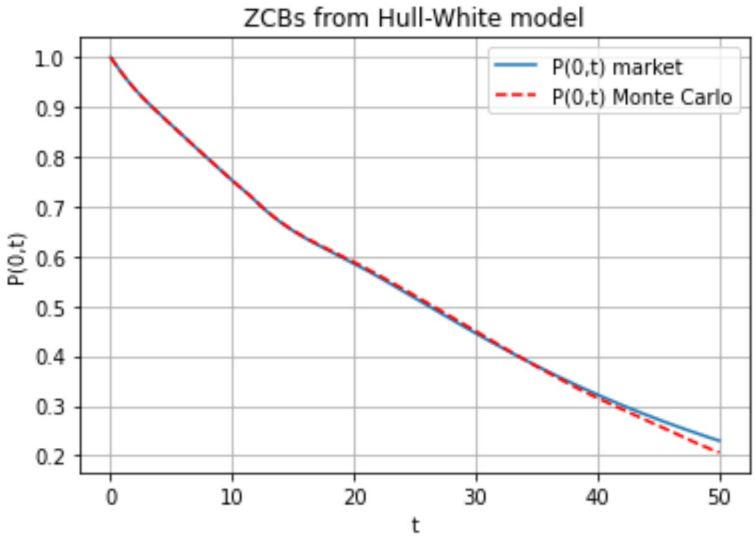
In [27]:
```
# Final comparison
[t, P, implied_term_structure] = mainCalculation(NoOfPaths, NoOfSteps, T, a, sigma, P
```

Implied and simulated price of a ZCB

Visual comparison

In [28]:
```
plt.figure(1)
plt.grid()
plt.xlabel("t")
plt.ylabel("P(0,t)")
plt.plot(t, implied_term_structure)
plt.plot(t, P,"--r")
plt.legend(["P(0,t) market", "P(0,t) Monte Carlo"])
plt.title("ZCBs from Hull-White model")
```

Out[28]:
```
Text(0.5, 1.0, 'ZCBs from Hull-White model')
```



In [29]:
```
test_statistics_bdp_1 = pd.DataFrame(abs(P- implied_term_structure)*10000, columns=['
```

Implied and simulated term structure

Absolute difference in bps

In [30]:
```
test_statistics_bdp_1.head()
```

Out[30]:

	Abs diff in bps
0	2.220446e-12
1	3.006300e-03
2	1.528834e-02
3	3.678171e-02
4	6.747583e-02

Test 1; Success criteria

The successful application of the success criteria marks the completion/failiure of the test.

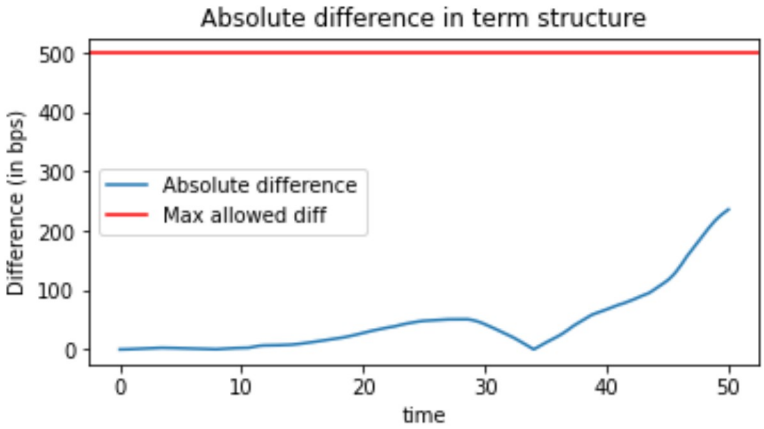
In [31]:
```
result1 = SuccessTest(test_statistics_bdp_1.values, statistics_mean_max_diff_in_bps,
```

```
Test passed
Test passed
```

In [32]:
```
fig, ax1 = plt.subplots(1,1)
ax1.plot(t, test_statistics_bdp_1, label= "Absolute difference")
ax1.axhline(y = statistics_mean_max_diff_in_bps, color = 'r', linestyle = '-',label='

ax1.set_xlabel("time")
ax1.set_ylabel("Difference (in bps)")
ax1.set_title('Absolute difference in term structure')
ax1.legend()
fig.set_figwidth(6)
fig.set_figheight(3)

plt.show()
```



Test 2; Comparison between simulated an calculated volatility

The Hull-White model is one of the few widely used short rate models that has a closed-form formula for the calculation of the volatility. The formula is:

$$V[r(t) \mid F_0] = \frac{\sigma^2}{2a}\left(1 - e^{-2at}\right)$$

This fact makes it possible to compare the volatility calculated from the parameters with the simulated volatility.

In [33]:
```
paths = Paths(NoOfPaths, NoOfSteps, T, P0t, a, sigma, epsilon)
R = paths["R"]
t = paths["time"]
```

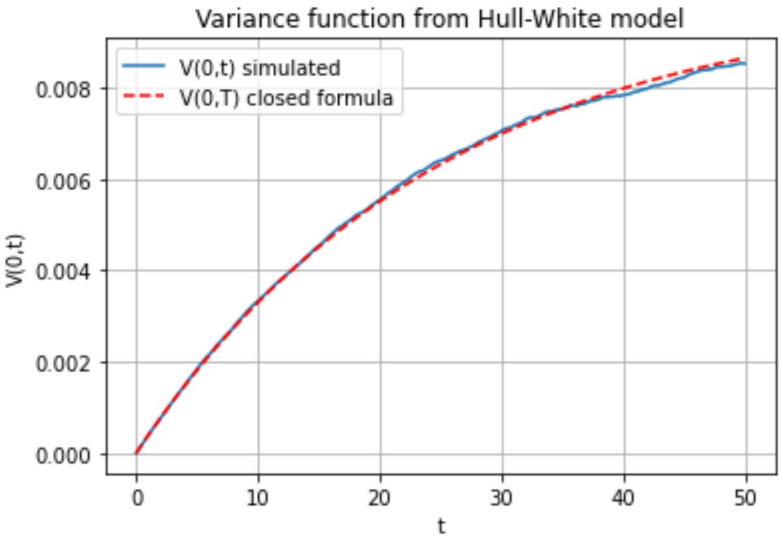
In [34]:
```
time = np.linspace(start=0,stop=T,num=NoOfSteps+1, endpoint=True)
```

In [35]:
```
var_series = np.var(R, axis=0)
```

In [36]:
```
vol = sigma**2/(2*a)*(1-np.exp(-2*a*time))
```

In [37]:
```
plt.figure(1)
plt.grid()
plt.xlabel("t")
plt.ylabel("V(0,t)")
plt.plot(t,var_series)
plt.plot(t,vol,"--r")
plt.legend(["V(0,t) simulated", "V(0,T) closed formula"])
plt.title("Variance function from Hull-White model")
```

Out[37]:
```
Text(0.5, 1.0, 'Variance function from Hull-White model')
```



In [38]:
```
test_statistics_bdp_2 = pd.DataFrame(abs(var_series-vol)*10000, columns=["Abs diff ir
```

Theoretical and simulated volatility

Absolute difference in bps

In [39]:
```
test_statistics_bdp_2.head()
```

Out[39]:

	Abs diff in bps
0	1.302904e-24
1	5.549388e-04
2	8.896989e-03
3	1.328121e-02
4	1.487104e-02

Test 2; Success criteria

The successful application of the success criteria marks the completion/failiure of the test.

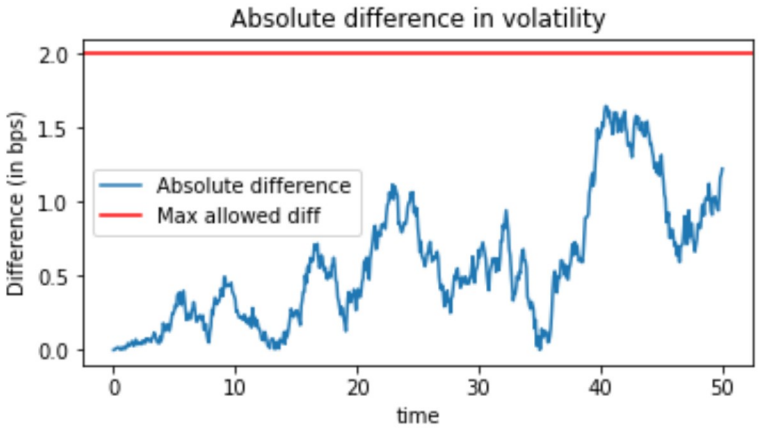
In [40]:
```
result2 = SuccessTest(test_statistics_bdp_2.values, statistics_vol_max_diff_in_bps, s
```

```
Test passed
Test passed
```

In [41]:
```
fig, ax1 = plt.subplots(1,1)
ax1.plot(t, test_statistics_bdp_2, label= "Absolute difference")
ax1.axhline(y = statistics_vol_max_diff_in_bps, color = 'r', linestyle = '-',label="M

ax1.set_xlabel("time")
ax1.set_ylabel("Difference (in bps)")
ax1.set_title('Absolute difference in volatility')
ax1.legend()
fig.set_figwidth(6)
fig.set_figheight(3)

plt.show()
```



Conclusion

The tests are successfull if the Hull-White curve generator calibrated using the March 2023 EIOPA curve, passes the success criteria. Based on the preformed tests, if all the tests pass, it is likely that the implementation was generated using a correct methodology and ran using the correct parameters.

In [42]:
```
pd.DataFrame(data = [result1, result2], columns = ["Mean test","Max test"],  \
             index= ["Average term structure","Theoretical vs empirical volatility"])
```

Out[42]:

	Mean test	Max test
Average term structure	True	True

Mean test **Max test**