Corporate Bond Pricer

Belinda AYDOGMUS

Myriam Magda BENOUDINA

Sihame JDID

Faculté des sciences économiques et de gestion
Université de Strasbourg

Introduction

I. Mise en place de l'environnement

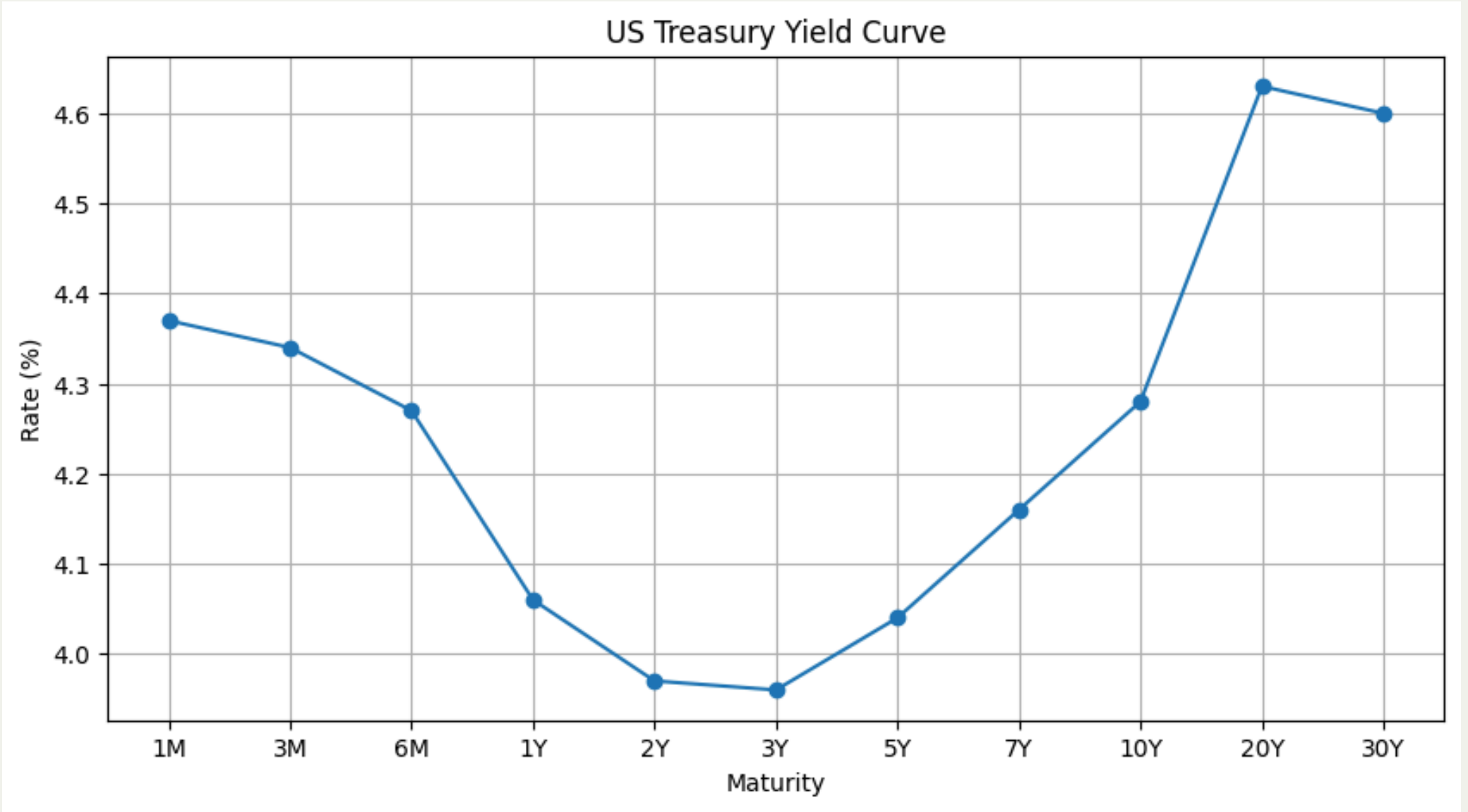
```
##### MISE EN PLACE DE L'ENVIRONNEMENT #####

# Installation des modules #
import os
import pandas as pd
import numpy as np
import tkinter as tk
from tkinter import ttk, messagebox
import matplotlib.pyplot as plt
from fredapi import Fred
from scipy import interpolate
from bs4 import BeautifulSoup
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import time

# Forcage du bon dossier
```

II. Base de données

A. US Yield Curve

```
##### DATA BASE #####

## US YIELD CURVE ##
# FRED API key : ce6755615bc789593ab6aed48597ea81

fred = Fred(api_key='ce6755615bc789593ab6aed48597ea81')

# Dictionnaire des maturités & codes FRED correspondants
series_ids = {
    '1M': 'GS1M',
    '3M': 'GS3M',
    '6M': 'GS6M',
    '1Y': 'GS1',
    '2Y': 'GS2',
    '3Y': 'GS3',
    '5Y': 'GS5',
    '7Y': 'GS7',
    '10Y': 'GS10',
```

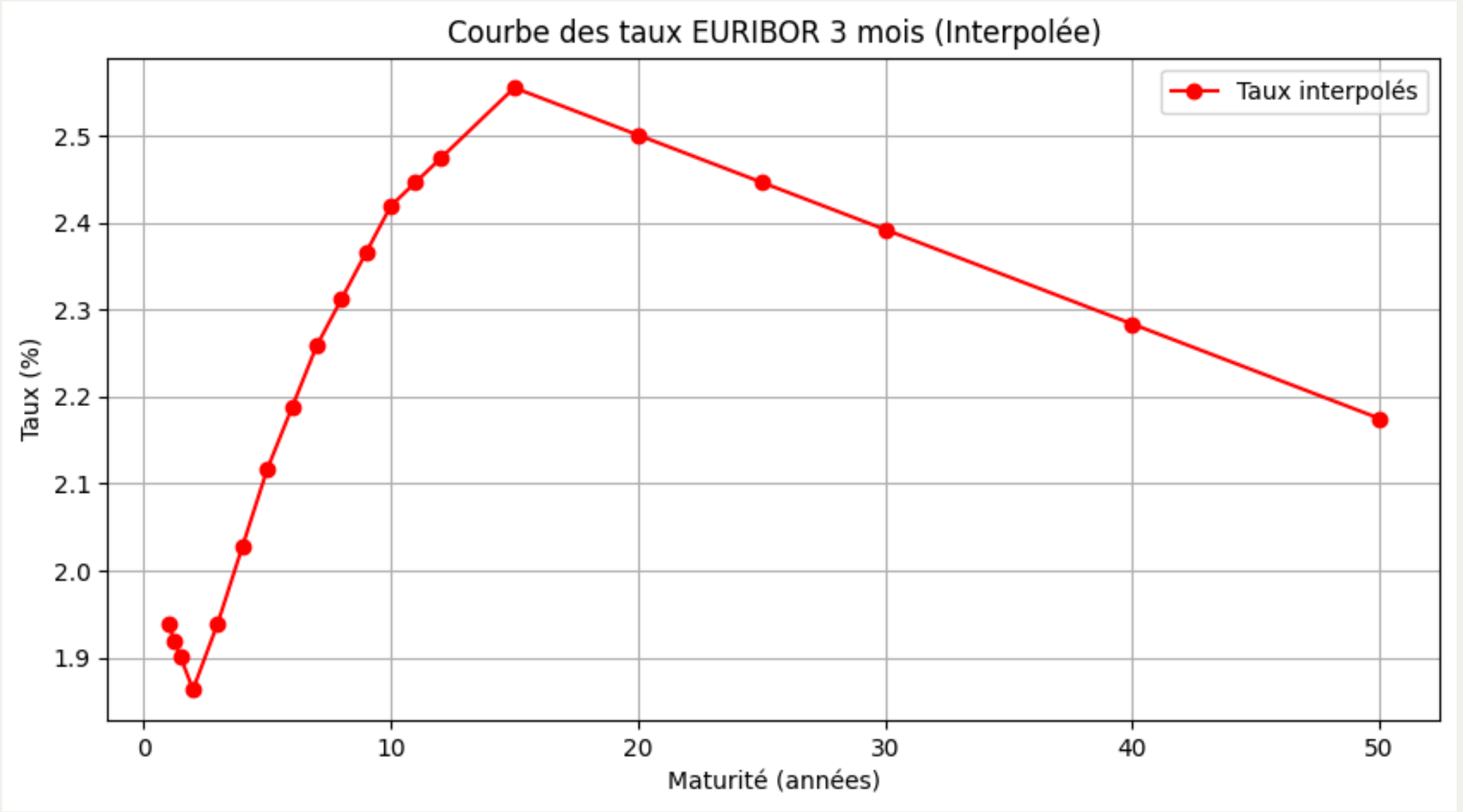
Courbe des taux sans risque US



B. EURIBOR 3M Curve

```
## EURIBOR 3M CURVE ##

# Vérifier si le fichier existe déjà
csv_file = "euribor_rates_interpolated.csv"
if not os.path.exists(csv_file):  # Si le fichier n'existe pas, on exécute le code
    print("Le fichier n'existe pas. Début de l'extraction et de l'interpolation de

    # Lancer le driver Selenium
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    # URL de la page à scraper
    url = 'https://www.chathamfinancial.com/technology/european-market-rates'

    # Ouverture de la page dans un navigateur avec Selenium
    driver.get(url)

    # Attente de 5 secondes pour s'assurer que la page est bien chargée
    time.sleep(5)
```

Courbe EURIBOR 3M



C. CDX IG Prices

```
## CDX IG PRICES ##
# Les données proviennent de Bloomberg /!\

# Nom du fichier
filename = "CDX_IG_Prices.csv"

# Vérifier si le fichier existe déjà
if not os.path.exists(filename):
    # Données
    data = {
        "Name": [
            "21st Century Fox America Inc", "Allstate Corp/The", "Apache Corp", "A
            "Carnival Corp", "Comcast Corp", "HP Inc", "MetLife Inc", "Xerox Corp"
        ],
        "CDS Ticker": [
            "CNCP1U5 CBIN Curncy", "CALL1U5 CBIN Curncy", "CAPA1U5 CBIN Curncy", "
            "CHRB1U5 CBIN Curncy", "CCCL1U5 CBIN Curncy", "CT363688 CBIN Curncy",
            "CMET1U5 CBIN Curncy", "CXRX1U5 CBIN Curncy"
```

III. Code

A. Interpolation linéaire

- Interpolation linéaire : $y = y_1 + \frac{(x - x_1)}{(x_2 - x_1)} \times (y_2 - y_1)$

```
##### CODE #####

def convert_maturity_to_float(maturity):
# ------------------------------------------------------------------------
# Objectif : Convertir une maturité exprimée en chaîne de caractères ("1Y", "6M")
#            en un nombre flottant représentant le nombre d'années.
#            Exemple : "6M" devient 0.5 et "5Y" devient 5.0.
# Entrée : maturity (str ou float) - La maturité sous forme de chaîne ou de nombre
# Sortie : float - La maturité exprimée en années (nombre flottant).
# ------------------------------------------------------------------------
    if isinstance(maturity, str):
        if 'Y' in maturity:
            return float(maturity.replace('Y', ''))
        elif 'M' in maturity:
            return float(maturity.replace('M', '')) / 12
    return float(maturity)

```

B. Prix, duration et calendrier des flux de trésorerie

Prix de l'obligation :

$$P_0 = \sum_{t=1}^{T} \frac{c}{(1+r_{f_t}+s_t)^t} + \frac{1}{(1+r_{f_T}+s_T)^T}$$

$$\text{Duration : } D_0 = \frac{\sum_{t=1}^{T} \frac{t \times c}{(1+r_{f_t}+s_t)^t} + \frac{T}{(1+r_{f_T}+s_T)^T}}{P_0}$$

- r_t : taux sans risque à la date t issu de la courbe souveraine interpolée
- s_t : spread de crédit spécifique à l'émetteur à la date t
- c : coupon payé périodiquement
- T : maturité

B. Prix, duration et calendrier des flux de trésorerie

- Cashflow :

$$\mathrm{CF}(t) = \begin{cases} \frac{c}{f} + 1, & \text{si } t = T \\ \frac{c}{f}, & \text{sinon} \end{cases}$$

- Discounted Factor : $\mathrm{DF}(t) = \frac{1}{(1+r_{f_t}+s_t)^t}$
- Discounted Cash Flow : $\mathrm{DCF}(t) = \mathrm{CF}(t) \times \mathrm{DF}(t)$

B. Prix, duration et calendrier des flux de trésorerie

```
class cMod_Bond:
# ------------------------------------------------------------------------
# Objectif : Représente une obligation d'entreprise.
#            Cette classe permet de calculer :
#            - Le prix de l'obligation (en actualisant les flux futurs)
#            - La duration (sensibilité du prix à une variation des taux)
#            - Le calendrier des flux de trésorerie ("schedule")
#
# Attributs :
#     - issuer : Nom de l'émetteur
#     - coupon_rate : Taux facial du coupon
#     - margin : Marge pour les coupons variables
#     - coupon_type : "Fixed" ou "Variable"
#     - coupon_frequency : "Annual", "Semi-Annual" ou "Quarterly"
#     - maturity : Maturité de l'obligation (en années)
#     - spread, obj_RFRate, obj_Spread, obj_LiborCurve : courbes de taux utilisées
#
# Méthodes principales :
```

C. Notre outil : le pricer

```
def load_data():
# ------------------------------------------------------------------------
# Objectif : Charger les données de taux depuis les fichiers CSV :
#             - Libor 3M interpolé
#             - US Yield Curve
#             - CDX IG Prices (spreads)
# Retourne :
#     - dict_LIBOR3M : Dictionnaire des taux Libor 3M
#     - dict_US_YIELD_CURVE : Dictionnaire des taux US Risk-Free
#     - cdx_ig_prices : Table complète des spreads par entreprise
# ------------------------------------------------------------------------
    libor_curve = pd.read_csv('euribor_rates_interpolated.csv')
    us_yield_curve = pd.read_csv('US_Yield_Curve.csv')
    cdx_ig_prices = pd.read_csv('CDX_IG_Prices.csv')

    dict_LIBOR3M = {convert_maturity_to_float(maturity): rate / 100 for maturity,
    dict_US_YIELD_CURVE = {convert_maturity_to_float(maturity): rate / 100 for mat

```

D. L’interface du pricer

```
def run_interface():
# ------------------------------------------------------------------------
# Fonction : run_interface()
# Objectif : Lancer une interface graphique Tkinter pour permettre à l'utilisateur
#            - Choisir les paramètres de l'obligation (émetteur, taux, maturité,
#            - Calculer le prix et la duration
#            - Afficher dynamiquement le calendrier de flux
# Résultat : L'interface Tkinter est lancée et interactive
# ------------------------------------------------------------------------
    def calculate():
    # ------------------------------------------------------------------------
    # Fonction interne : calculate()
    # Objectif : (Appelée par le bouton Tkinter)
    #            - Récupérer les inputs utilisateur
    #            - Lancer le pricer
    #            - Afficher le prix, la duration et le calendrier dans l'interfac
    # ------------------------------------------------------------------------
        try:
```

Conclusion