ENGIN 604 Introducción a Python para las Finanzas — Otoño 2021

Examen

Entrega: 10:50am, Jueves, Abril 22
Enviar a: engin604assignments@gmail.com
Límite máximo de páginas: ∞ páginas

Instrucciones

- El examen es **INDIVIDUAL**.
- Usted dispone desde las 9:30 am hasta las 10:50 am (1 hora y 20 minutos) del día jueves 22 de Abril del 2021 para desarrollar y enviar el examen.
- Debe desarrollar el examen en **UN SOLO SCRIPT** de Python con extensión **.py** y enviarlo al correo `engin604assignments@gmail.com` antes de las 10:50 am del día jueves 22 de Abril del 2021. Cualquier correo que sea enviado después del horario límite será evaluado con nota 1.0.
- El nombre del archivo debe ser `Ex_RUT sin puntos ni guión.py`. Ejemplo: si su rut es `12.345.678-9`, el nombre del archivo deberá ser `Ex_123456789.py`.
- El asunto del correo debe ser `Examen + RUT sin puntos ni guión`. Ejemplo: si su rut es `12.345.678-9`, el asunto del correo deberá ser `Examen 123456789`.
- Toda pregunta que involucre una respuesta escrita deberá ser respondida mediante un comentario en el archivo **.py**.
- En caso que tenga alguna duda respecto al desarrollo de su examen, puede indicar su supuesto como comentario en el archivo **.py**, cuya pertinencia será analizada al momento de corregir. Si presenta algún problema en el desarrollo de su examen, puede ingresar a la siguiente sala de Webex: `https://fenuchile.webex.com/meet/gcabrerag`.
- **SOLO** puede utilizar los enunciados y pautas de las **GUÍAS**, **TAREAS** y **AYUDANTÍAS** como apoyo durante el desarrollo del examen.
- Cualquier indicio de copia será evaluado con nota 1.0 y se tomarán las acciones que estipula el reglamento de **TOLERANCIA CERO A LA COPIA** de la Escuela de Postgrado.

Preguntas

1. (10 puntos) Un número primo es aquel que solo es divisible por si mismo y el número 1. Por ejemplo, el número 11 es un número primo porque solo es divisible por 11 y por 1. En una lista muestre los números primos existentes entre 1 y 100 (no considere el 1 como número primo).

```
# lista de números del 1 al 100
num1 = list(range(1,101))

# lista vacia
num_primos = []

# loop
for i in num1:
    if i > 1:
        num2 = []
        for j in range(2,i+1):
            if i % j == 0:
                num2.append(j)
        if len(num2) == 1:
            num_primos.append(i)
```

2. Un bono bullet es aquel en donde el emisor pagará al tenedor del bono cupones (pago de interés) correspondiente a cada periodo y al vencimiento (último periodo) recibirá el cupón más el principal (valor nominal del bono). La formula matemática sería:

$$P_B = \sum_{t=1}^{T-1} \frac{C}{(1+r)^t} + \frac{\text{C + Valor Nominal}}{(1+r)^T}$$

Donde P_B es el precio del bono, C el pago de intereses o cupones, T número de periodos y r la tasa de descuento.

   a. (10 puntos) Genere un *array* que contenga una secuencia de números desde 0.02 hasta 0.4 (40 elementos) donde la distancia entre cada número sea 0.01.

```
# se carga numpy
import numpy as np
# se crea el array
num3 = np.array(list(range(2,41))) / 100
```

   b. (10 puntos) Utilizando como tasa de descuento (r) cada elemento del *array* generado en (a), calcule y almacene en un lista el precio que tendría el bono bullet (P_B). A excepción de la tasa de descuento (r) mantenga constante los siguientes parámetros:

Cuadro 1: Parámetros Bono Bullet

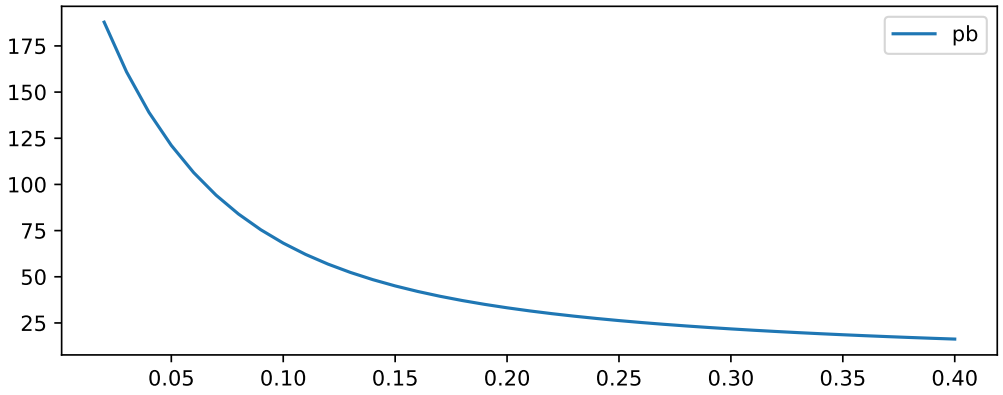
Parámetro	Valor
T	25 años
C	0.065
Valor Nominal	100

El pago de los cupones son anuales.

```
def precio_bono_bullet(vn, t, tc, r):
    """
    Parameters
    ----------
    vn : Valor nominal del bono bullet (float).
    t : Número de periodos (float).
    tc : Interes del cupón (float).
    r : Tasa de descuento (float).
    Returns
    -------
    Precio de un bono bullet.
    """
    pb = 0

    for i in range(1,t+1):
        if i == t:
            pb += (vn * tc + vn) / (1 + r) ** i
        else:
            pb += (vn * tc) / (1 + r) ** i

    return(pb)

# lista vacia
pb_k_list = []

# en un loop se calcula y guarda el precio del bono
for k in num3:
    pb_k = precio_bono_bullet(100, 25, 0.065, k)
    pb_k_list.append(pb_k)
```

c. (10 puntos) Grafique la relación entre el precio del bono bullet (P_B) y la tasa libre de riesgo (r). ¿Qué se observar?.

```
# se carga pandas
import pandas as pd

# se crea el DataFrame
df1 = pd.DataFrame(data = {'pb':pb_k_list}, index = num3)

# se grafica
df1.plot(figsize=(8,3))
```



3. La volatilidad realizada mensual del cobre para el mes m se construye como la raíz de la suma de los retornos logarítmicos diarios al cuadrado:

$$RV_t = \sqrt{\sum_{j=1}^{M_t} r_{j,t}^2}$$

Donde M_t es el número de días hábiles totales del mes m.

a. (5 puntos) El archivo `cobre_diario.xlsx` contiene dos hojas:

i. `copper`: Precio *spot* diario del cobre (`icopper`) desde 2000-01-03 hasta 2019-12-31 (`date`).

ii. `recession`: Ciclo económico mensual de China (`chrec`) desde 2000-01 hasta 2019-12 (`year` y `month`). 1 implica recesión y 0 expansión.

Cargue los datos a su espacio de trabajo y calcule la volatilidad realizada mensual para el cobre. Recuerde eliminar los `NAs` que se originan al crear los retornos logarítmicos.

```
# carga la hoja con el precio diario del cobre en dolares
cobre_diario = pd.read_excel('cobre_diario.xlsx', sheet_name='copper')
```

```
# se agrega la fecha al índice
cobre_diario.set_index('date', inplace=True)

# se calcula el retorno logarítmico
cobre_diario['logret'] = cobre_diario.apply(lambda x: np.log(x / x.shift(1)))
# se elimna el na
cobre_diario.dropna(inplace=True)

# se crea la columna año
cobre_diario['year'] = cobre_diario.index.year
# se creala la columna mes
cobre_diario['month'] = cobre_diario.index.month

# se elimina la variable contiene el precio del cobre
cobre_diario.drop(columns=['icopper'], axis=1, inplace=True)

# se agrupa por año y mes, se calcula el retorno al cuadrado y
# se suma creando una observación por mes
```

```
cobre_diario = cobre_diario.groupby(['year', 'month'])['logret'].\
                            apply(lambda x: np.sqrt(np.sum(x ** 2)))

# se transforma a DataFrame y luego el índice (año y fecha) pasa a variable
rv_copper = cobre_diario.to_frame().reset_index()

# renombre de la variable
rv_copper.rename(columns={'logret':'log_rv_copper'}, inplace=True)
```

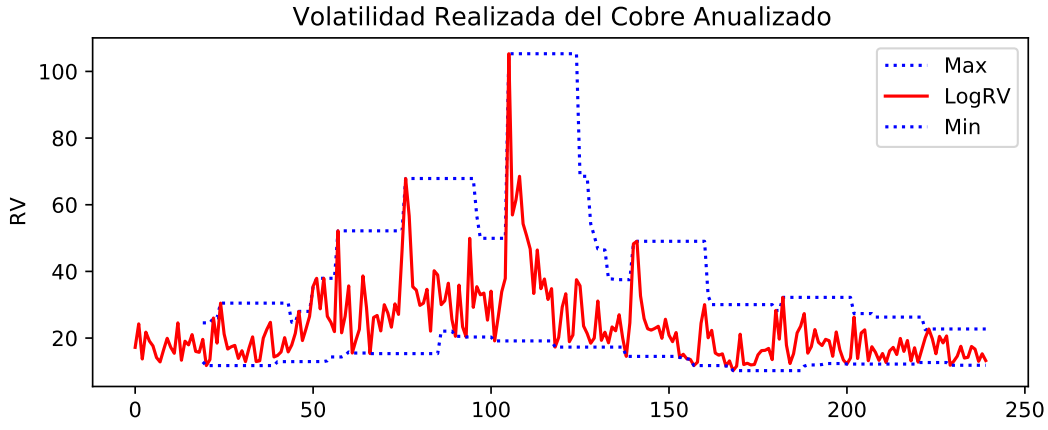
b. (5 puntos) Junte la volatilidad realizada mensual calculada en (a) con la variable `rec` de la hoja `recession`.

```
# carga la hoja con las recesiones
rec = pd.read_excel('cobre_diario.xlsx', sheet_name='recession')
```

```
# mediante un merge se une la recesión con el precio
rv_copper_with_rec = rv_copper.merge(rec, on=['year', 'month'])
```

c. (10 puntos) Anualice[1] la volatilidad realizada mensual creada en (a) y luego grafíquela agregando el mínimo y máximo movil a 20 días (también de la volatilidad realizada anualizada). El resultado esperado es el siguiente:

```
# se importa matplotlib
import matplotlib.pyplot as plt

# se crea variables
rv_copper_with_rec2 = rv_copper_with_rec.copy()
rv_copper_with_rec2['LogRV'] = rv_copper_with_rec2['log_rv_copper'].\
                               apply(lambda x: x * np.sqrt(12) * 100)
rv_copper_with_rec2['Min'] = rv_copper_with_rec2['LogRV'].rolling(20).min()
rv_copper_with_rec2['Max'] = rv_copper_with_rec2['LogRV'].rolling(20).max()

# se grafica la volatilidad realizada anualizada
rv_copper_with_rec2.loc[:,['Max','LogRV','Min']].plot(figsize=(8,3),
                                                      style=['b:','r-','b:'])
plt.title("Volatilidad Realizada del Cobre Anualizado")
plt.ylabel("RV")
```



[1] Para anualizar se debe multiplicar por $\sqrt{12}$.

d. (10 puntos) Calcule la estadística descriptiva[2] de la volatilidad realizada mensual para los periodos donde:

i. China estuvo en recesión (`rec` = 1)

```
# se filtra por chrec == 0 y se calcula la estadística descriptiva
# de la volatilidad realizada
rv_copper_with_rec[rv_copper_with_rec['chrec']==0].loc[:,'log_rv_copper'].\
                                                    describe()[['mean','std',
                                                                'min','max']]
```

ii. China estuvo en expansión (`rec` = 0)

```
# se filtra por chrec == 1 y se calcula la estadística descriptiva
# de la volatilidad realizada
rv_copper_with_rec[rv_copper_with_rec['chrec']==1].loc[:,'log_rv_copper'].\
                                                    describe()[['mean','std',
                                                                'min','max']]
```

[2]Debe solo incluir promedio, desviación estandar, valor mínimo y valor máximo.