7

Métodos Numéricos

Trabajo práctico N°3

Ecuaciones no lineales

Grupo 11: Martin Amagliani
Olivia De Vincenti
Nicolás Fernandez Pelayo

Profesor: Pablo Ignacio Fierens

Fecha de entrega: 1 de octubre de 2020

Buenos Aires, Argentina

Consigna

Grupo impar

1. Escriba una función que calcule:

$$y = e^x,$$

donde $x \in [0, 1)$. La función debe tener el nombre `expi` y recibir x. No puede usar funciones trascendentales y sólo puede realizar sumas, restas, productos y cocientes.

2. Escriba una función que calcule

$$y = e^x,$$

donde $x \in \mathbb{R}$. La función debe tener el nombre `expc` y recibir x. No puede usar funciones trascendentales y sólo puede realizar sumas, restas, productos, cocientes y usar `expi`. Puede utilizar una tabla de valores.

3. Escriba una función que calcule

$$y = LambertW(x),$$

donde x es un número no-negativo y $LambertW()$ es la función W de Lambert. Esta función debe dar la respuesta resolviendo una ecuación no-lineal y sólo puede realizar sumas, resta, productos, cocientes y usar la función `expc`. La función debe tener el nombre `lambertww` y recibir x como parámetro.

4. Escriba una función que pruebe el correcto funcionamiento de todas la demás funciones implementadas. La función debe llamarse `test`, sin argumentos.

Todas las funciones deben estar en un archivo denominado `lamberto.py`.

Funciones

Índice de Códigos

1. Ejercicio 1

¿Dónde se explica esto?

1.1. expi

Recibe un número en el intervalo $[0, 1)$ y, utilizando su polinomio de Taylor de orden 17, no tiene sentido aumentar el orden porque este basta para tener un error menor que el de máquina. Calcula:

$$y = e^x \tag{1}$$

Si no recibe un número en ese intervalo devuelve NaN.

Código 1: Definición de expi

```
def expi(x):
    if not checkn(x) or x >= 1 or x < 0:     # Compruebo rango y que x sea un número
        print("Por favor ingrese un número entre 0 y 1")
        return np.nan

    ITERACIONES = 17                  # Constante que define el orden del Polinomio de Taylor
    y = 0
    for i in range(ITERACIONES):       # Polinomio de taylor de exponencial
        y = y + (pot(x, i))/math.factorial(i)
    return y
```

Esto tiene varios problemas. Por ej., al calcular pot(x,4) se hace el producto x*x*x*x. Pero al calcular pot(x,3) ya se hizo el producto x*x*x. Algo parecido sucede con factorial(). Y dicho sea de paso: si usan su propia implementación de la potencia, ¿por qué usan factorial() de numpy?

1.2. checkn

Otro problema viene por el lado de los errores. Es mayor el error de redondeo al hacer x^k/k! que si la cuenta se hiciera de manera distinta (ver los ejercicios de la guía).

Recibe un dato y revisa si es un número revisando si es un int o float.
Si es un número devuelve True, sino devuelve False.

Código 2: Definición de checkn

```
def checkn(x):
    if isinstance(x, int) or isinstance(x, float):     # Verifica si es un número a través del tipo de dato
        return True                  # Es número
    return False                     # No es número
```

1.3. pot

Recibe un número $x \in \mathbb{R}_0^+$ y un exponente $n \in \mathbb{N}_0$. Calcula la n-ésima potencia de x.
En el contexto de este tp, siempre se cumplirán los requisitos.

Código 3: Definición de pot

```
def pot(x, n):
    y = 1
    for i in range(n):               # Multiplico a un número n veces por si mismo
        y = y * x
    return y
```

2. Ejercicio 2

2.1. expc

Recibe un número $x \in \mathbb{R}$ y calcula (1) utilizando la siguiente propiedad de la potencia:

$$a^{x+y} = a^x.a^y \tag{2}$$

Así, separamos x en parte entera y parte racional para luego calcular (1) usando `pot` y `expi`

Código 4: Definición de expc

```
def expc(x):
    if not checkn(x):                          # Compruebo que x es un número
        return np.nan
    y = x
    if x < 0:                                 # Si x es negativo tomo su módulo
        y = -x
    pe = int(y)                                 # Separo en parte entera y parte decimal
    pd = y - int(y)

    ex = (pot(math.e, pe))*(expi(pd))                 # Calculo e^y usando propiedades
    if x < 0:                                 # Si x era negativo invierto el resultado
        ex = 1/ex

    return ex
```

3. Ejercicio 3

3.1. lambertww

Recibe un número $x \in \mathbb{R}_0^+$ y calcula la función W de Lambert:

$$x = W(x)e^{W(x)} \tag{3}$$

Esta ecuación no lineal se resuelve por el método de Newton-Raphson, haciendo la siguiente iteración:

$$w_k = w_{k-1} - \frac{w_{k-1}e^{w_{k-1}-x}}{w_{k-1}e^{w_{k-1}} + e^{w_{k-1}}} \tag{4}$$

Es importante para lograr mejor eficiencia elegir un buen w inicial, nosotros tomamos distintos casos según qué tan grande es el número.

Código 5: Definición de lambertww

```
def lambertww(x):
    if not checkn(x) or x < 0:  # Compruebo que x sea un numero y mayor o igual a 0
        print("Por favor ingrese un número real no negativo")
        return np.nan
    w = 0.5                     # w inicial 0.5 para numeros menores a 1000
    n = 50                      # Número de iteraciones para numeros menores a 100
    if x > 100:
        n = int(x/2)            # Número de iteraciones para numeros menores a 1000 y mayores a 100
        if x > 1000:
            n = 500             # Número de iteraciones para numeros mayores a 1000
            w = 2*(digit(x)-2)  # w inicial para numeros mayores a 1000
    for i in range(n):
        w = w - ((w * expc(w) - x)/(w * expc(w) + expc(w)))      # Newton-Raphson
    return w
```

¿Por qué????

Usar un número fijo de iteraciones (y tan grande) no es razonable.

¿Por qué??

3.2. digit

Recibe un número y calcula su cantidad de dígitos. Si recibe un número decimal, calculará sólo los dígitos de la parte entera. La función está diseñada para recibir sólo números $n \in \mathbb{R}_0^+$ ya que es utilizada para aproximar el valor inicial usado en `lambertww`.

Código 6: Definición de digit

```
def digit(x):
    i = 0
    while(x >= 1):
        x = x/10                    # Divido por 10 hasta que el modulo sea menor a 1
        i = i + 1                   # Aumento el contador
    return i
```

4. Ejercicio 4

4.1. test

Función de prueba. Se evalúan `expi`, `expc` y `lambertww`. Los valores se comparan con los resultados de `math.exp` y de `scipy.special.lambertw`, los cuales consideramos que funcionan bien. Primero se verifican los casos que deberían dar error porque no cumplen con los requisitos de la función. Luego se chequea cada una para dsitintos valores aleatorios. En el caso de `expi` estos variarán entre 0 y 1, en `expc` entre -700 y 700 y en `lambertww` entre 0 y 10^{15}.
Para facilitar la lectura y prolijidad, las funciones se evalúan de a una, por lo que el usuario debe apretar enter para analizar la siguiente.

Código 7: Definición de test

```
def test():
##############################################################
# EXPI
##############################################################
   print("EXPI:\n")
   print("e^", 1, "=", expi(1), "\n")                    # Caso x=1: fuera del rango de la funcion
   print("e^hola =", expi("hola"), "\n")                 # Caso x no es numero
   print("e^", -1, "=", expi(-1), "\n")                  # Caso x negativo: fuera del rango de la
    ↪ funcion
   print("e^", 0, "=", expi(0), "\n")                    # Caso x = 0: numero conocido valido
   for i in range (10):
      x = random.uniform(0,1)
      print("\nsegun expi()")                            # Casos aleatorios
      print("e^", x, " =", expi(x))
      print("segun math.exp()")                         # Se comparan con la librería math
      print("e^", x, " =", math.exp(x))

   print("Presione enter para continuar")
   input()

##############################################################
# EXPC
##############################################################
   print("EXPC:\n")
   print("e^", 1, " =", expc(1), "\n")                  # Caso x=1: compruebo que devuelva e
   print("e^hola =", expc("hola"), "\n")                # Caso x no es numero
   print("e^", -1, " =", expc(-1), "\n")                # Caso x negativo
   print("e^", 0.5, " =", expc(0.5), "\n")              # Caso x entre 0 y 1: fuciona


   print("e^", 699.5872, " =", expc(699.5872), "\n")     # Caso numero grande
   print("e^", 4200.69, " =", expc(4200.69), "\n")       # Caso numero grande que debe dar infinito
   print("e^", -750.32674654, " =", expc(-750.32674654), "\n")     # Caso numero grande negativo
    ↪ que debe dar 0
   for i in range(10):
      x=random.uniform(-700, 700)                       # Casos aleatorios
```

```
        print("\nsegun expc()")
        print("e^", x, " =", expc(x))
        print("segun math.exp()")                    # Se comparan con la librería math
        print("e^", x, " =", math.exp(x))
    print("\n\nPresione enter para continuar")
    input()

##########################################################################
# LAMBERTWW
##########################################################################
    print("LAMBERTWW:\n")


    print("\nSegun lambertww()")
    print("W(", 0, ") =", lambertww(0))                 # Caso x=0
    print("Segun scipy.special.lambertw()")
    print("W(", 0, ")=", scipy.special.lambertw(0))

    print("\nSegun lambertww()")
    print("W(", 1.56, ") =", lambertww(1.56))           # Caso x= 1.56
    print("Segun scipy.special.lambertw()")
    print("W(", 1.56, ")=", scipy.special.lambertw(1.56))

    print("\nSegun lambertww()")
    print("W(hola) =", lambertww("hola"))               # Caso x no es un numero

    print("\nSegun lambertww()")
    print("W(", -1, ") =", lambertww(-1))               # Caso x negativo: no es valido

    print("\nSegun lambertww()")
    print("W(", 0.0000000152, ") =", lambertww(0.0000000152))   # Caso x numero pequeño
    print("Segun scipy.special.lambertw()")
    print("W(", 0.0000000152, ")=", scipy.special.lambertw(0.0000000152))

    x = random.uniform(0, 1)                         # Casos aleatorios:
    print("\nSegun lambertww()")                     # Excepción: el primer caso es entre 0 y 1
    print("W(", x, ")=", lambertww(x))
    print("Segun scipy.special.lambertw()")
    print("W(", x, ")=", scipy.special.lambertw(x))
    for i in range(15):                              # C/u está entre una potencia de 10
        x=random.uniform(pot(10, i), pot(10, i+1))   # y la siguiente potencia
        print("\nSegun lambertww()")
        print("W(", x, ")=", lambertww(x))
        print("Segun scipy.special.lambertw()")
        print("W(", x, ")=", scipy.special.lambertw(x))     # Se comparan con la librería scipy
    print("\n\nPresione enter para continuar")
    input()
    print("W(1e26) =", lambertww(1e26))              # Caso extremo
    print("scipy no soluciona esto")
                                                     Sí lo hace.
    return
```