Curso de Optimización (DEMAT)

Tarea 1

Descripción:	Fechas
Fecha de publicación del documento:	**Febrero 3, 2022**
Fecha límite de entrega de la tarea:	**Febrero 13, 2022**

Indicaciones

El propósito de esta tarea es poner en práctica lo que hemos revisado sobre Python, por lo que los ejercicios son de programación.

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- **Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.**
- **Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimirlo y puede anexar sólo el notebook en la tarea del Classroom.**
- **Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. No lo incluya dentro del ZIP , porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.**

En la descripción de los ejercicios se nombran algunas variables para el algoritmo, pero sólo es para facilitar la descripción. En la implementación pueden nombrar sus variables como gusten.

En los algoritmos se describen las entradas de las funciones. La intención es que tomen en cuenta lo que requiere el algoritmo y que tiene que haber parámetros que permitan controlar el comportamiento del algoritmo, evitando que dejen fijo un valor y que no se puede modificar para hacer diferentes pruebas. Si quieren dar esta información usando un tipo de dato que contenga todos los valores o usar variables por separado, etc., lo pueden hacer y no usen variables globales si no es necesario.

Lo mismo para los valores que devuelve una función. Pueden codificar como gusten la manera en que regresa los cálculos. El punto es que podamos tener acceso a los resultados, sin usar variables globales, y que la función no sólo imprima los valores que después no los podamos usar.

Ejercicio 1 (6 puntos)

Programar y probar el método de la iteración de Halley para el cálculo de raíces de una función de una variable.

Descripción del método

El método de Halley usa una aproximación de la función $f(x)$ de segundo orden del desarrollo de Taylor de $f(x)$.

$$f(x_{k+1}) \approx f(x_k) + f'(x_k)\Delta x + \frac{1}{2}f''(x_k)(\Delta x)^2$$

Si igualamos a cero la aproximación tenemos que

$$\Delta x = -\frac{f(x_k)}{}$$

$$f'(x_k) + \frac{1}{2}f''(x_k)\Delta x$$

El valor Δx en el lado izquierdo de la igualdad corresponde a $\Delta x = x_{k+1} - x_k$, mientras que el que está en el denominador se aproxima por el paso de Newton-Raphson:

$$\Delta x = -\frac{f(x_k)}{f'(x_k)},$$

de modo que

$$x_{k+1} - x_k = -\frac{f(x_k)}{f'(x_k)}, \\ -\frac{1}{2}f''(x_k)f(x_k) \\ /f'(x_k)$$

es decir, el método de Halley propone generar la secuencia de puntos mediante la siguiente regla:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k) - \frac{f''(x_k)f(x_k)}{2f'(x_k)}}.$$

1. **Escriba la función que aplique el método de Halley. Debe recibir como argumentos un punto inicial x_0, las función $f(x)$, sus derivadas $f'(x)$ y $f''(x)$, el número máximo de iteraciones y un tolerancia $\tau > 0$, similar a la función `NewtonRaphson()` vista en el ejemplo de la clase, de modo que se detenga cuando se cumpla que $|f(x_k)| < \tau$. Defina la variable `res` que indique el resultado obtenido (`res=0` se acabaron las iteraciones y no se encontró un punto que satisfaga el criterio de convergencia, `res=1` el algoritmo converge, `res=-1` hay un problema al evaluar la expresión. La función debe devolver el último punto x_k, $f(x_k)$, el número de iteraciones realizadas y la variable `res`.**
2. **Pruebe el algoritmo de Halley con las siguientes funciones y puntos inciales:**

$$f_1(x) = x^3 - 2x + 1, \\ x_0 = -1000, 1000.$$

$$f_2(x) = 1 + x - \frac{3}{2}x^2 \\ + \frac{1}{6}x^3 + \frac{1}{4}x^4, x_0 = \\ -1000, 1000.$$

En cada caso imprima x_0, $f(x_0)$, x_k, $f(x_k)$, el número de iteraciones k realizadas y el valor de la variable res.

1. **Repita las pruebas anteriores con el método de Newton-Raphson y escriba un comentario sobre los resultados.**

Solución:

In [15]:

```
# Implementación del método de Newton-Raphson.
# Pueden modificar la función si lo desean.
def NewtonRaphson(x0, fnc, derf, iterMax, tol):
    xk  = x0
    res = 0
    for i in range(iterMax):
        fk = fnc(xk)
        if fk<tol and fk>-tol:
            res = 1
            break
        else:
            dfx = derf(xk)
            if dfx!=0:
                xk = xk - fk/dfx
```

```
            else:
                res = -1
                break
    return xk,fk,i,res
```

In [4]:

```
# En esta celda puede poner el código de la función pedida
# o poner la instrucción para importar la función de un archivo .py

def Halley(x0, fnc, derf, derf2, iterMax, tol):
    xk = x0
    res = 0
    for i in range(iterMax):
        fk = fnc(xk)
        if fk<tol and fk>-tol:
            res = 1
            break
        else:
            derfk = derf(xk)
            derfk2 = derf2(xk)
            if derfk!=0 or derfk - derfk2*fk/(2*derfk)!=0:
                xk = xk - fk/(derfk - derfk2*fk/(2*derfk))
            else:
                res = -1
                break
    return xk,fk,i,res
```

In [8]:

```
# Esta celda o en otras adicionales pueden poner las pruebas realizadas.
def f1(x):
    return x**3 - 2*x + 1

def f2(x):
    return 1+x - (3/2)*x**2 + (1/6)*x**3+(1/4)*x**4

def derf1(x):
    return 3*x**2 - 2

def derf2(x):
    return 1 - 3*x + (1/2)*x**2 + x**3

def der2f1(x):
    return 6*x

def der2f2(x):
    return -3 + x + 3*x**2
```

In [13]:

```
# Implementación de los métodos: Halley f1
xk,fk,i,res = Halley(-1000, f1, derf1, der2f1, 1000, 10e-6)
print("Evaluando la función 1 con el método de Halley:")
print("xk:" ,xk , ", f(xk):", fk, ", iteraciones:", i, ", res", res)
```

```
Evaluando la función 1 con el método de Halley:
xk: -1.6180339887504553 , f(xk): -3.2809310823722626e-12 , it3eraciones: 12 , res 1
```

In [16]:

```
# Implementación de los métodos: Newton-Raphson f1
xk,fk,i,res = NewtonRaphson(-1000, f1, derf1, 1000, 10e-6)
print("Evaluando la función 1 con el método de Newton-Raphson:")
print("xk:" ,xk , ", f(xk):", fk, ", iteraciones:", i, ", res", res)
```

```
Evaluando la función 1 con el método de Newton-Raphson:
xk: -1.6180339888222295 , f(xk): -4.234541606251696e-10 , iteraciones: 20 , res 1
```

In [17]:

```
# Implementación de los métodos: Halley f2
```

```
# Implementación de los métodos: Halley f2
xk,fk,i,res = Halley(-1000, f2, derf2, der2f2, 1000, 10e-6)
print("Evaluando la función 2 con el método de Halley:")
print("xk:" ,xk , ", f(xk):", fk, ", iteraciones:", i, ", res", res)
```

```
Evaluando la función 2 con el método de Halley:
xk: -2.9796542579895906 , f(xk): 8.718689059605822e-07 , iteraciones: 14 , res 1
```

In [18]:

```
# Implementación de los métodos: Newton-Raphson f2
xk,fk,i,res = NewtonRaphson(-1000, f2, derf2, 1000, 10e-6)
print("Evaluando la función 2 con el método de Newton-Raphson:")
print("xk:" ,xk , ", f(xk):", fk, ", iteraciones:", i, ", res", res)
```

```
Evaluando la función 2 con el método de Newton-Raphson:
xk: -2.9796541859258454 , f(xk): 1.6091874499579717e-09 , iteraciones: 25 , res 1
```

Esta celda es para el comentario:

Podemos ver que con el método de Halley la función converge más rápido. Esto quiza se deba a que se utiliza la segunda derivada y esto nos da más información para encontrar la concavidad de la función.

Ejercicio 2 (4 puntos)

Una manera de aproximar la función $\cos(x)$ es mediante la función

$$C(x;n) = \sum_{i=0}^{n} c_i$$

donde n es un parámetro que indica la cantidad de términos en la suma y

$$c_i = -c_{i-1}\frac{x^2}{2i(2i-1)} \quad \text{y} \quad c_0 = 1.$$

1. **Programe la función $C(x;n)$.**
2. **Imprima el valor del error $C(x;n) - 1$ para $x \in \{2\pi, 8\pi, 12\pi\}$ y $n = 10, 50, 100, 200$.**
3. **Imprima el valor del error $C(x;n) + 1$ para $x \in \{\pi, 9\pi, 13\pi\}$ y $n = 10, 50, 100, 200$.**
4. **Comente sobre el comportamiento de los errores obtenidos y cuál sería una manera apropiada de usar esta función.**

Solución:

In [2]:

```
# En esta celda puede poner el código de la función
# o poner la instrucción para importar la función de un archivo .py
import numpy as np

def cos(x,n):
    sum = 1
    prev = 1
    for i in range(1,n+1):
        cur = -prev* x**2 / (2* i *(2*i-1))
        sum = sum + cur
        prev = cur

    return sum
```

In [9]:

```
# En esta celda o en otra adicionales puede poner las
```

```
# pruebas realizadas



#Ejercicios 2

for x in [2*np.pi, 8*np.pi,12*np.pi]:
    for i in [10,50,100,200]:
        print("x:", x, "n:", i, "Error:", cos(x,i)-1)
```

```
x: 6.283185307179586 n: 10 Error: 0.0003012240418271972
x: 6.283185307179586 n: 50 Error: -4.6629367034256575e-15
x: 6.283185307179586 n: 100 Error: -4.6629367034256575e-15
x: 6.283185307179586 n: 200 Error: -4.6629367034256575e-15
x: 25.132741228718345 n: 10 Error: 2515268877.313313
x: 25.132741228718345 n: 50 Error: 4.869289242925845e-07
x: 25.132741228718345 n: 100 Error: 4.869289242925845e-07
x: 25.132741228718345 n: 200 Error: 4.869289242925845e-07
x: 37.69911184307752 n: 10 Error: 10801559649420.639
x: 37.69911184307752 n: 50 Error: 0.19176691520961198
x: 37.69911184307752 n: 100 Error: 0.13572456351592055
x: 37.69911184307752 n: 200 Error: 0.13572456351592055
```

In [10]:

```
#Ejercicios 3

for x in [np.pi, 9*np.pi,13*np.pi]:
    for i in [10,50,100,200]:
        print("x:", x, "n:", i, "Error:", cos(x,i)+1)
```

```
x: 3.141592653589793 n: 10 Error: 7.565070792026063e-11
x: 3.141592653589793 n: 50 Error: -2.220446049250313e-16
x: 3.141592653589793 n: 100 Error: -2.220446049250313e-16
x: 3.141592653589793 n: 200 Error: -2.220446049250313e-16
x: 28.274333882308138 n: 10 Error: 29037871175.43674
x: 28.274333882308138 n: 50 Error: -1.0977908454945506e-05
x: 28.274333882308138 n: 100 Error: -1.0977908465603647e-05
x: 28.274333882308138 n: 200 Error: -1.0977908465603647e-05
x: 40.840704496667314 n: 10 Error: 55393063446067.77
x: 40.840704496667314 n: 50 Error: 194.4088792156606
x: 40.840704496667314 n: 100 Error: 1.4636083162256588
x: 40.840704496667314 n: 200 Error: 1.4636083162256588
```

Esta celda es para el comentario:

Podemos notar que comparando, mientras el valor de la x sea mayor, mayor será el error. Vemos que se puede hacer más pequeño el error si el valor de la n es mayor.

La manera más apropiada de usar la función, es que si tenemos una x pequeña, es mejor usar una n también pequeña para que no itere tantas veces, ya que es innecesario.

Si la x es grande, lo mejor es usar una n grande para poder minimizar el error lo más posible.