parcial1

October 22, 2020

1 Examen 1

Notebook correspondiente al primer parcial de Metodos Numericos

[20]:
```
from helper import *

import sys
import scipy

import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from math import *
from scipy.linalg import solve_triangular

NOTEBOOK = True
```

1.1 Ejercicio 1

Usando los códigos de la **Tarea2**, escriba un programa que trate de obtener todas las raíces reales del polinomio mediante lo que se conoce como el método de Newton-Horner.

[2]:
```
def metodo_newton_horner(n, a, x0, N, t=None, dtype=np.float64):

    c = np.copy(a)
    r = np.zeros(n, dtype=dtype)
    retm = 0

    for m in range(n):
        x = x0
        res = False
        for k in range(N):
            px, b = metodo_horner(n-m+1, c, x,
                                  dtype=dtype)
            if abs(px) < t:
                r[m-1] = x
                c = np.copy(b)
```

```
                res = True
                break
            else:
                dpx, b = metodo_horner(n-m, b, x,
                                       dtype=dtype)
                x = x - px/dpx

        if res == False:
            break
        retm += 1
    print(b)
    return r, retm
```

[3]:
```
""" Entonces el m altera el movimiento en el bucle????  """
def Algoritmo2(n, a, x0, t, N, dtype=np.float64):

    c = np.copy(a)
    r = np.zeros(n)

    m=0
    while m < (n):
        x = x0
        res = False
        for k in range(N):
            px, b = metodo_horner(n-m+1, c, x,
                                  dtype=dtype)
            if abs(px) < t:
                r[m-1] = x
                c = np.copy(b)
                res = True
                break
            else:
                dpx, b = metodo_horner(n-m, b, x,
                                       dtype=dtype)
                x = x- px/dpx

        if res == False:
            m = m-1
            break
        m +=1
        print(k)

    return r, m
```

[4]:
```
def Ejercicio1():
    dtype = np.float64
```

```
    coef = [1/5, 3, 101/20, -129/2, -483/4, 585/2]
    coef = np.array(coef, dtype=dtype, copy=True)

    x0 = 0
    T = (np.finfo(dtype).eps)**(1/2)
    N = 500

    ## Probado con Algoritmo2 y con metodo_newton_horner
    r, m = Algoritmo2(5, coef, x0, T, N, dtype=dtype)

    if True:
        print(f'Encontramos {m} raices, las que son ', end='')
        print(show1D(r))
        print(r)

        mi = min(r); ma = max(r);
        intervalo = np.linspace(mi-0.5, ma+0.5,
                                endpoint=True)
Ejercicio1()
```

```
299
0
0
0
0
Encontramos 5 raices, las que son
[ 0.  0.  0.  0. 10.]
```

1.1.1 No funciona y no se porque

1.2 Ejercicio 2

Me embobe en el anterior porque pense que estaba muy simple, pra cuando me di cuenta el profesor ya habian pasado 2:45 hrs. Por eso no acabe este.

[ ]:

[5]:
```
# Parte 3
# Solucion de minimos cuadrados

def minimosCuadrados(A, b,/, dtype=np.float64):
    """ Funcion que calcula la solucion de minimos cuadrados.

    Funcion que calcula la solucion de minimos cuadrado. Para
    esto se basa de funciones ya implementadas para calcular
    A^t@A y A^tb, para luego calcular la solucion del sitema
    A^t@Ax = A^tb con fact de Cholesky.
```

```
    La funcion devuelve la solucion del sistema x si se encontro
    y None en caso de que no se haya encontrado.

    A pesar de que la funcion tambien pide que se pasen las
    dimensiones de las matrices, la forma pythonica no lo
    requiere; por lo que seran obtenidas dentro del metodo.

    Input:
        A := apuntador a matriz A
        b := apuntador a vector b

    Output:
        x := si existe sus valors; None en otro caso
    """
    x = np.zeros((A.shape[0],1))

    At = A.transpose()@A
    yt = A.transpose()*b

    return solChol(At, At.shape[0], yt)
```

[18]:
```
def Ejercicio1(d, n_tabla, n,/,path='datos/', dtype=np.float64, plot=True,␣
↪prnt=True, ask=True):
    # Cargar datos
    tabla = np.load(path+n_tabla, allow_pickle=False)

    info = {'minx': min(tabla[0,:]),
            'maxx': max(tabla[1,:])}
    # Valores obtenidos
    x = np.ravel(tabla[0,:])
    y = np.ravel(tabla[1,:])
    rng = np.linspace(info['minx'], info['maxx'], num=d)

    T = ceil(info['maxx'])
    nf = T-1
    def fk(k):
        return k/T


    if plot and ask:
        plot_ej1_1(x,y)
        try:
            print(f"Seguro que quieres usar grado {n} para aproximar?")
            inp = input("[S para mantener]: ").lower().strip()

            if not inp.startswith('s'):
                n = int(inp)
```

```
        finally:
            if prnt: print(f'n={n}')



    sz = len(x)
    if n < sz:
        # Crear matriz a
        A = np.ones((sz, 2*n+1),
                    dtype=dtype)
        for i in range(1,n):
            A[0,i] = 1 # Por el c0
            for k in range(sz):
                """ Debemos asignar a los elementos
                    A[k,i] la expresion de sale del desarrollo
                """
                x = [k,i]
                A[k,2*i-1] = cos(2*pi*fk(i))*x
                A[k,2*i]   = sin(2*pi*fk(i))*x
        # vector y
        b = np.matrix(tabla[1,:]).transpose()

        coef = minimosCuadrados(A, b)
        p = f_polinomio(coef)
        write2Dvec(path+'resp-'+n_tabla, x, [p(xi) for xi in x])

        if plot:
            plot_ej1_2(x, y, p, rng)
        if prnt:
            print(f'Se encontraron los coeficientes')
            print('\t'+show1D(coef, show=False))
            print(f'Error = {error_ej1(p, x, y)}')
    else:
        raise Exception("Sistema indeterminado")
```

[23]:
```
"""
No corre, pero asi se ejecutaria en caso de qe si

# Parte 5
if NOTEBOOK:
    Ejercicio1(50, 'caso1.npy', 2)"""
None
```

2 Comentario

Mejor dejelo como examen tarea, esto es damasiada presion xD

Entrego porque se acabo el tiempo.

[ ]:

[ ]: