



UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Taller 4: Polinomios de Chebyshev

Néstor Heli Aponte Ávila

Métodos Numéricos

Cód. 20182167052

2022 - I

July 7, 2022

1 Ejercicios

1. Realice las modificaciones pertinentes al programa socializado para que compile correctamente en cualquier intervalo $[a, b]$ con $a < b$.

Construire el programa desde cero basandome en las siguientes fórmulas:

Nodos. los cuales provienen de la sustitución $x = \left(\frac{b-a}{2}\right)t + \frac{a+b}{2}$ y también $t = 2\frac{x-a}{b-a} - 1$

$$x_k = \frac{b-a}{2} \cos\left(\frac{(2k+1)\pi}{2N+2}\right) + \frac{a+b}{2} \quad k = 0, 1, \dots, N \quad (1)$$

Aproximación de Chebyshev.

$$P_N(x) = \sum_{j=0}^N c_j T_j\left(2\frac{x-a}{b-a} - 1\right) \quad (2)$$

donde

$$c_0 = \frac{1}{N+1} \sum_{k=0}^N f(x_k) \quad y \quad c_j = \frac{2}{N+1} \sum_{k=0}^N f(x_k) T_j(x_k) \quad j = 1, 2, \dots, N \quad (3)$$

```
1 from cmath import cos, e, exp, pi, sin # Libreria de la que se sacan cositas
   matematicas como funciones, numeros especiales, etc.
2 import numpy as np # Esencial para trabajar matematicas
3 import sympy as sym # Tambien trae muchas funciones matematicas
4 import matplotlib.pyplot as plt # Libreria para construir las graficas
5 from lagrangeinter import interpol
6
7 # PREVIOUSLY
8 x = sym.Symbol('x') # Declara 'x' como variable simbolica
9
10 # INPUT
```

```

11 fun = 1 / (1+x**2) # Funcion que se desea aproximar
12 n = 20 # Grado del polinomio de aproximacion
13 a = -5 # Extremo inferior del intervalo
14 b = 5 # Extremo superior del intervalo
15 evalf = sym.lambdify(x, fun, 'numpy') # evalf() evalua la funcion fun
16
17 # OBTENCION DE LOS NODOS DE Chebyshev
18 X = [] # Vector que recoge los nodos
19 for i in range(n+1): # Recorrido
20     X.insert(0, (b-a)/2*cos(((2*i+1)*pi)/(2*n+2))+(a+b)/2) # Calculo con la
        sustitucion a bordo y ordenados de una vez porque .sort() no me sirvio
21
22 # Evaluacion de la funcion en los nodos
23 Y = evalf(np.array(X))
24
25 # OBTENCION DE LOS COEFICIENTES c_j DE Chebyshev
26 C = [(1/(n+1))*sum(Y)] # c_0 primera iteracion
27 for j in range(n):
28     c = 0 # Variable auxiliar para recoger las sumatorias
29     for k in range(n+1):
30         c += (2/(n+1))*evalf(X[k])*cos((j+1)*np.arccos(2*((X[k]-a)/(b-a))-1)) #
        Formula, usando la sustitucion y tambien la construccion alternativa de T_j(x) =
        cos(j*arccos(x))
31     C.append(c) # guarda lo calculado
32
33 # CALCULO DE LOS POLINOMIOS T_j(x)
34 T = [1, x] # Vector que recoge los polinomios T_j(x) con la construccion natural
35 for i in range(n-1): # Recorrido hasta el polinomio que necesito
36     T.append(2*x*T[-1]-T[-2]) # Formulilla
37
38 # OBTENCION DEL POLINOMIO DE Chebyshev
39 P = [] # Vector recogedor
40 for i in range(n+1): # Recorrido
41     P.append(0) # Anado 0's para poder acceder a esas posiciones en la iteracion
42     if i != 0: # T(0) = 1 (No lo puedo evaluar en x)
43         T[i] = T[i].subs(x, 2*((x-a)/(b-a))-1) # Calculo los polinomios con la
        sustitucion
44     else:
45         P[i] = C[i]
46     P[i] = C[i]*T[i] # Formulilla
47
48 polinomio = sum(P).expand() # .sum() suma todo en el vector P y .expand() simplifica
        mi polinomio
49
50 # OUTPUT
51 print(f"Nodos: \n{X} \nImagen en esos nodos: \n{Y} \nPolinomio Interpolador: \n{
        polinomio}")

```

Listing 1: Aproximación de Chebyshev

2. Grafique la función propuesta en clase en el intervalo $[-5, 5]$ y sobre esta el polinomio de grado 20 evaluado en los nodos. Adjunte la salida del programa. Agregue nombres a los ejes y título a la gráfica.

$$f(x) = \frac{1}{1+x^2}$$

```

1
2 # INPUT
3 fun = 1 / (1+x**2) # Funcion que se desea aproximar
4 n = 20 # Grado del polinomio de aproximacion
5 a = -5 # Extremo inferior del intervalo
6 b = 5 # Extremo superior del intervalo
7
8 #GRAPHICS
9 evalp = sym.lambdify(x, polinomio, 'numpy') # evalp() evalua en el polinomio
        interpolador
10 dom = np.linspace(start = a, stop = b, num = 100) # linspace() me ayuda a crear el
        dominio para los graficos
11 plt.plot(dom, evalf(dom), X, evalp(np.array(X)), "o") # Graficos de la funcion y el
        polinomio de aproximacion evaluado en los nodos
12 plt.legend(["f(x)", "P(x_k)"]) # Convenciones
13 plt.xlabel("Coordenada x") # Indicacion eje horizontal

```

```

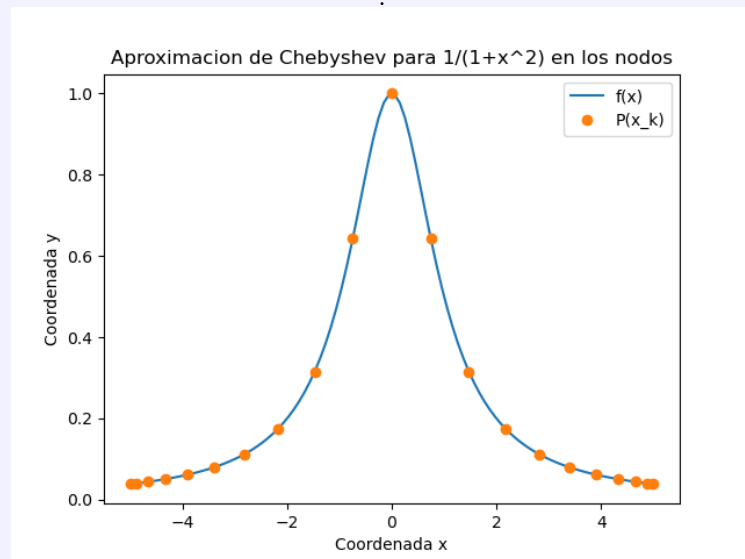
14 plt.ylabel("Coordenada y") # Indicacion eje vertical
15 plt.title("Aproximacion de Chebyshev para 1/(1+x^2) en los nodos") # Titulo
16 plt.show() # Lanza la interfaz para la visualizacion del grafico
17

```

```

[yoriichinara@MANJARO codes]$ /bin/python chebyaprox.py
Nodos:
[(-4.9860189859059005+0j), (-4.874639560909118+0j), (-4.6543687432210215+0j), (-4.330127018922193+0j), (-3.909157412340149+0j), (-3.40
08636888545962+0j), (-2.81660029031811+0j), (-2.1694186955877903+0j), (-1.473775872054521+0j), (-0.7452113308808715+0j), (3.0616169978
68383e-16+0j), (0.7452113308808732+0j), (1.4737758720545204+0j), (2.1694186955877908+0j), (2.8166002903181098+0j), (3.4008636888545976
+0j), (3.909157412340149+0j), (4.330127018922194+0j), (4.6543687432210215+0j), (4.874639560909118+0j), (4.9860189859059005+0j)]
Imagen en esos nodos:
[[0.03866918-0.j 0.04038428-0.j 0.0441245 -0.j 0.05063291-0.j
 0.06141936-0.j 0.07958062-0.j 0.1119415 -0.j 0.17524253-0.j
 0.31525699-0.j 0.64294627-0.j 1. -0.j 0.64294627-0.j
 0.31525699-0.j 0.17524253-0.j 0.1119415 -0.j 0.07958062-0.j
 0.06141936-0.j 0.05063291-0.j 0.0441245 -0.j 0.04038428-0.j
 0.03866918-0.j]]
Polinomio Interpolador:
6.78067022005649e-11*x**20 + 1.93566083908081e-24*x**19 - 8.96743636602473e-9*x**18 - 2.7053989470005e-22*x**17 + 5.09571604956136e-7*
x**16 + 1.58089678734541e-20*x**15 - 1.62693324679782e-5*x**14 - 5.03431874676608e-19*x**13 + 0.000320455893243221*x**12 + 0.527118436
38099e-18*x**11 - 0.0040277296026915*x**10 - 1.09384572510862e-16*x**9 + 0.0323471815498659*x**8 + 7.43946060310918e-16*x**7 - 0.16238
54813073*x**6 - 2.77863843045623e-15*x**5 + 0.490607151368132*x**4 + 4.79448078460898e-15*x**3 - 0.870493343568169*x**2 - 2.3789347228
2424e-15*x + 1.0
/usr/lib/python3.10/site-packages/matplotlib/cbook/_init_.py:1298: ComplexWarning: Casting complex values to real discards the imagi
nary part
  return np.asarray(x, float)
[yoriichinara@MANJARO codes]$

```



3. Con la misma función y misma cantidad de nodos pero tomados de manera que sean equidistantes aproxime la función con el polinomio P de Lagrange.

```

1
2 from lagrangeinter import interpol
3
4 # Usando la funcion con lagrange
5 X1 = np.linspace(start=-5, stop=5, num=21) # Crea el vector de nodos equiespaciados
6      (21 porque necesito que el polinomio sea de grado 20)
7 lagpolinomio = interpol(X1, evalf(np.array(X1))) # Usa la funcion
8 evalplag = sym.lambdify(x, lagpolinomio, 'numpy') # evalplag() para evaluar en el
9      polinomio de Lagrange
10 print("Polinomio de Lagrange:\n", lagpolinomio)
11
12 # Grafica para visualizar
13 plt.plot(dom, evalf(dom), dom, evalplag(dom), X1, evalplag(np.array(X1)), "o") #
14      Graficos de la funcion y el polinomio de lagrange en dom
15 plt.legend(["f(x)", "P_L(x)", "(x_k,y_k)"]) # Convenciones
16 plt.xlabel("Coordenada x") # Indicacion eje horizontal

```

```

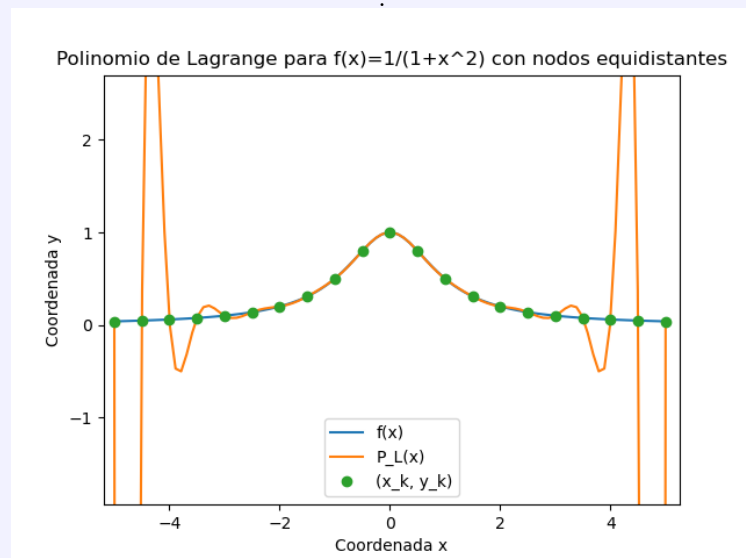
14 plt.ylabel("Coordenada y") # Indicacion eje vertical
15 plt.title("Interpolacion de Lagrange para  $f(x)=1/(1+x^2)$  con nodos equidistantes")
16 plt.show() # Lanza la interfaz para la visualizacion del grafico
17

```

```

[yoriichinara@MANJARO codes]$ /bin/python chebyaprox.py
Polinomio de Lagrange:
2.72817068149799e-9*x**20 - 4.54173855079897e-23*x**19 - 2.65314598775676e-7*x**18 - 6.38649607
339942e-20*x**17 + 1.07425130797328e-5*x**16 + 5.45325105398239e-18*x**15 - 0.000236412102809865
*x**14 - 1.08589623838001e-16*x**13 + 0.00310184793200539*x**12 - 4.45315713557687e-15*x**11 - 0
.0251135266738683*x**10 - 1.78681939036474e-15*x**9 + 0.126252909857137*x**8 - 2.24466712578364e
-14*x**7 - 0.391630076762948*x**6 + 4.11996825544492e-18*x**5 + 0.753353962815142*x**4 - 8.79743
326798188e-15*x**3 - 0.965739184991246*x**2 - 7.99057001121817e-17*x + 1.0
[yoriichinara@MANJARO codes]$

```



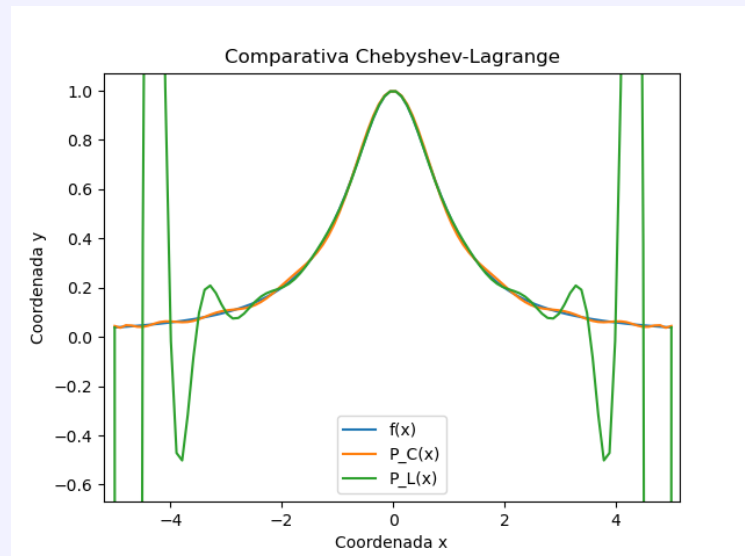
4. Ahora use los nodos de Chebyshev y halle el polinomio de grado 20 que aproxime la función. Escriba una conclusión de los puntos 3 y 4.

```

1
2 plt.plot(dom, evalf(dom), dom, evalp(dom), dom, evalplag(dom)) # Graficos de la
  funcion y el polinomio en dominio dom
3 plt.legend(["f(x)", "P_C(x)", "P_L(x)"]) # Convenciones
4 plt.xlabel("Coordenada x") # Indicacion eje horizontal
5 plt.ylabel("Coordenada y") # Indicacion eje vertical
6 plt.title("Comparativa Chebyshev-Lagrange") # Titulo
7 plt.show() # Lanza la interfaz para la visualizacion del grafico
8

```

Ya lo tenía así que construí el gráfico para comparar.



Reflexión. El error usando Chebyshev es prácticamente imperceptible, mientras que Lagrange lo dispara en los extremos. Esto deja ver que cuando de funciones específicas se trata, donde tengo cierta autonomía para escoger los puntos muestra, es mejor invocar al amigo Chebyshev.

References

- [1] Jhon H. Mathews - Kurtis D. Fink, *Métodos Numéricos con MATLAB*, Prentice Hall, (2000)