INF-155: Algoritmos y complejidad
Tarea #2
Interpolation in a nutshell

Anghelo Carvajal
201473062-4

16 de octubre de 2018

1. Pregunta 1

Implementar en Python 3 (dispone de toda la biblioteca científica) funciones para los métodos de interpolación vistos en clases, *Matriz de Vandermonde*, *Lagrange* y *Diferencias divididas de Newton*. Estas funciones deben recibir dos argumentos: 1. la función a interpolar (*callable*) y 2. los puntos de interpolación (lista).

Deberán retornar un *callable* que corresponda a la función interpolada. Deberá nombrar las funciones de la siguiente manera

- `my_vandermonde`
- `my_lagranje`
- `my_divided_differences`

1.1. Respuesta 1

Código 1: `my_vandermonde.py`

```python
import numpy as np

def my_vandermonde(f, A):
    matrix = []
    for j in np.arange(len(A)):
        aux = []
        for i in np.arange(len(A)):
            aux.append(np.power(A[j], i, dtype=np.float64))
        matrix.append(np.array(aux))
    matrix = np.array(matrix)
    y_list = np.array([f(x) for x in A])
    coef = np.linalg.solve(matrix, y_list)
    def thing_to_be_called(x):
        return sum([coef[i] * np.power(x, i) for i in range(len(coef))])
    return thing_to_be_called
```

Código 2: `my_lagrange.py`

```python
import numpy as np

def my_lagrange(f, A):
    y_list = [f(x) for x in A]
    range_y_list = range(len(y_list))
    def thing_to_be_called(x):
        total = 0
        for k in range_y_list:
            multiplicatoria = 1
            for j in range_y_list:
                if(j != k):
                    multiplicatoria *= np.true_divide(x - A[j], A[k] - A[j])
            total += y_list[k]*multiplicatoria
        return total
    return thing_to_be_called
```

Código 3: `my_divided_differences.py`

```python
import numpy as np

def my_divided_differences(f, A):
    aux = [f(x) for x in A]
    len_ = len(A)
    for j in range(1, len_):
        for i in range(len_-1, j-1, -1):
            aux[i] = np.true_divide(aux[i]-aux[i-1], A[i]-A[i-j], dtype=np.float64)

    def thing_to_be_called(xx):
        result = aux[0]
        olds = []
        for i in range(1, len(aux)):
            substraction = (xx - A[i-1])
```

```
            actualMultiplication = aux[i]*substraction
            for j in olds:
                actualMultiplication *= j
            result += actualMultiplication
            olds.append(substraction)
        return result

    return thing_to_be_called
```

2. Pregunta 2

Sea g la siguiente función:

$$g(x) = xcos(8x) + xsin(8x) x \in [0, 20]$$

Interpolar usando sus métodos anteriores y el método `interp1d` de **SciPy** la función $g(x)$ ya definida para:

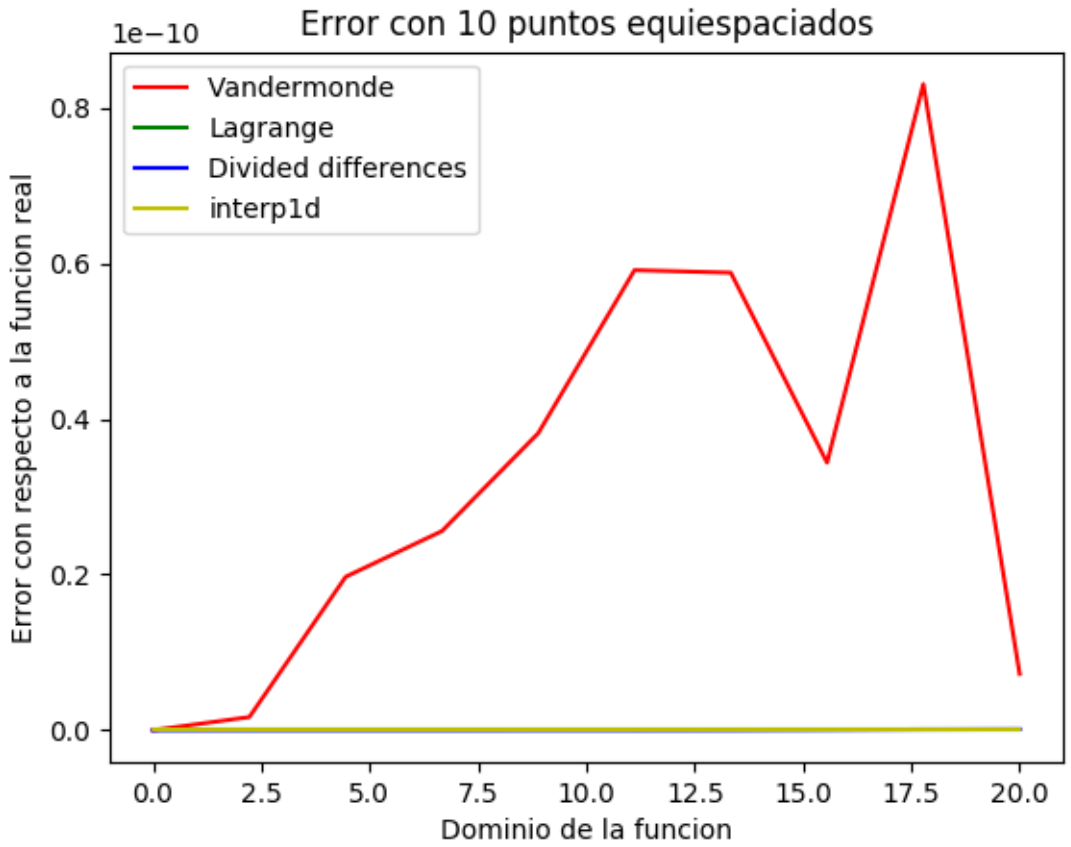
1. 10, 150 y 300 puntos equiespaciados entre 0 y 20.
2. 10, 150 y 300 puntos de *chebyshev* entre 0 y 20.

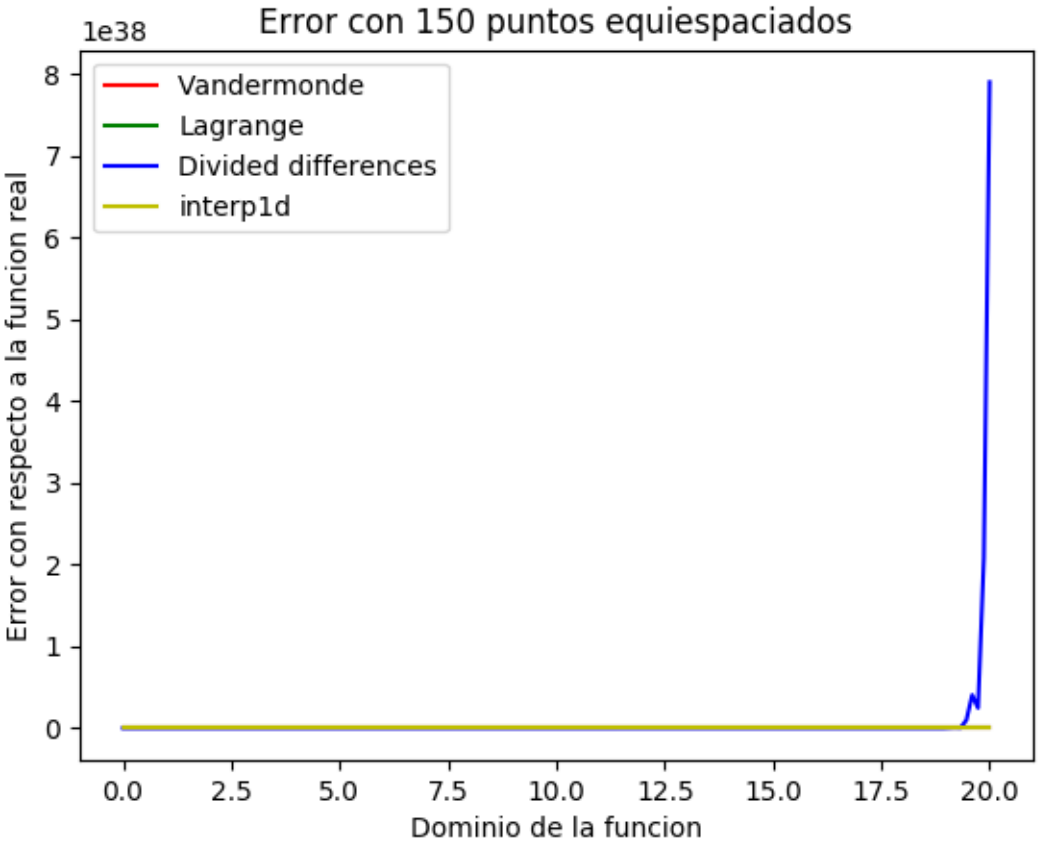
Graficar para cada cantidad de puntos un plot del error usando las 4 versiones de interpolación. Comente:

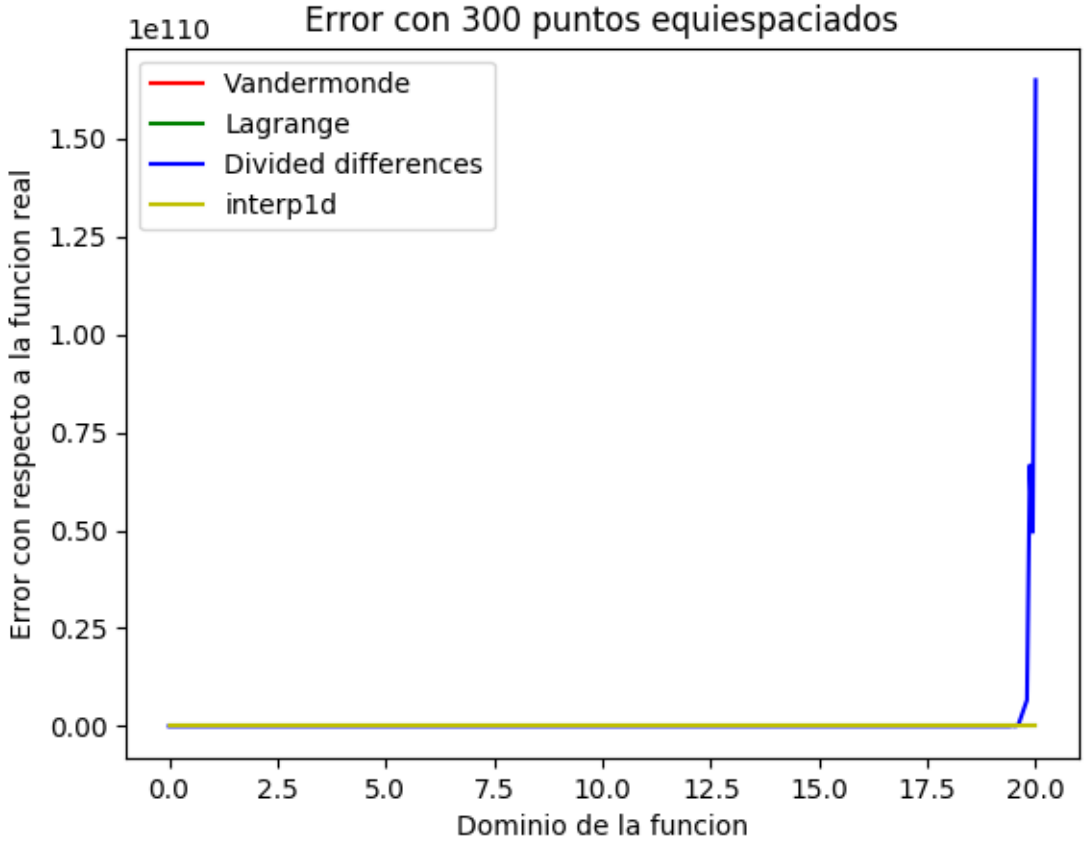
- Como cambia el error a medida que varía la cantidad de puntos a interpolar.
- Como el método escogido para definir los puntos (equiespaciados o *Chebyshev*) influye en el error.

2.1. Respuesta 2

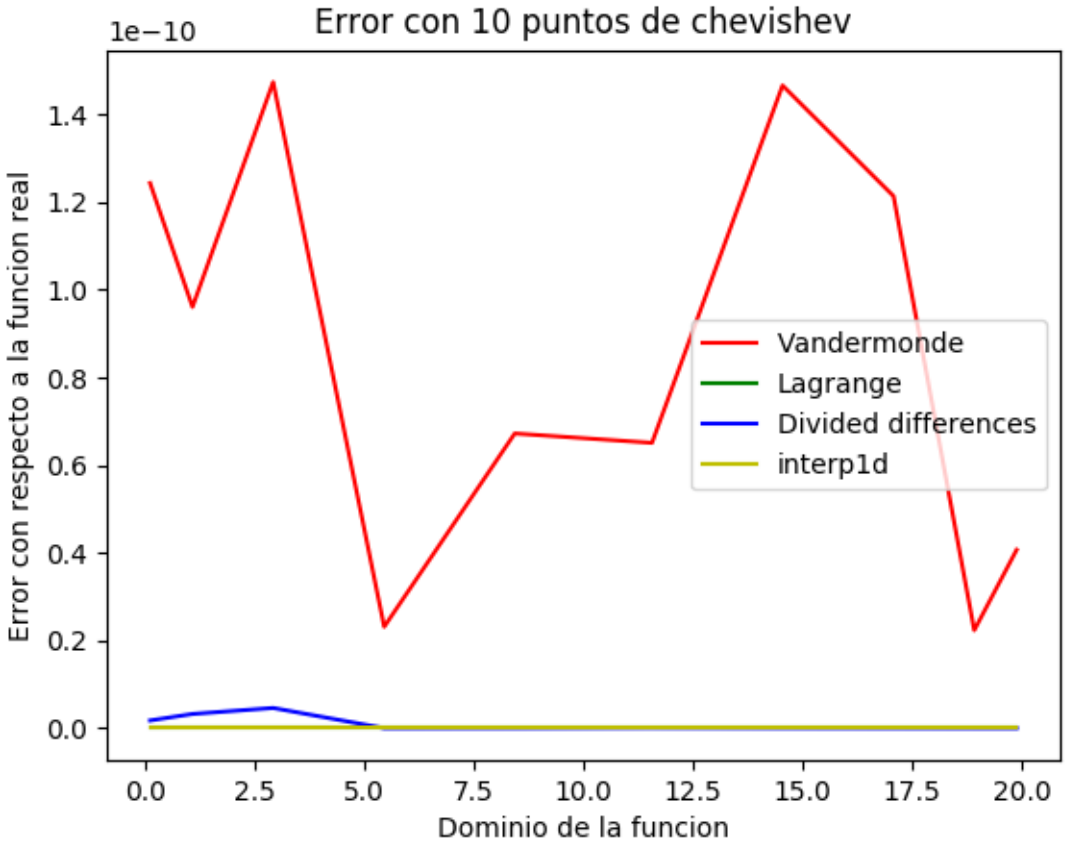
1. Puntos equiespaciados.

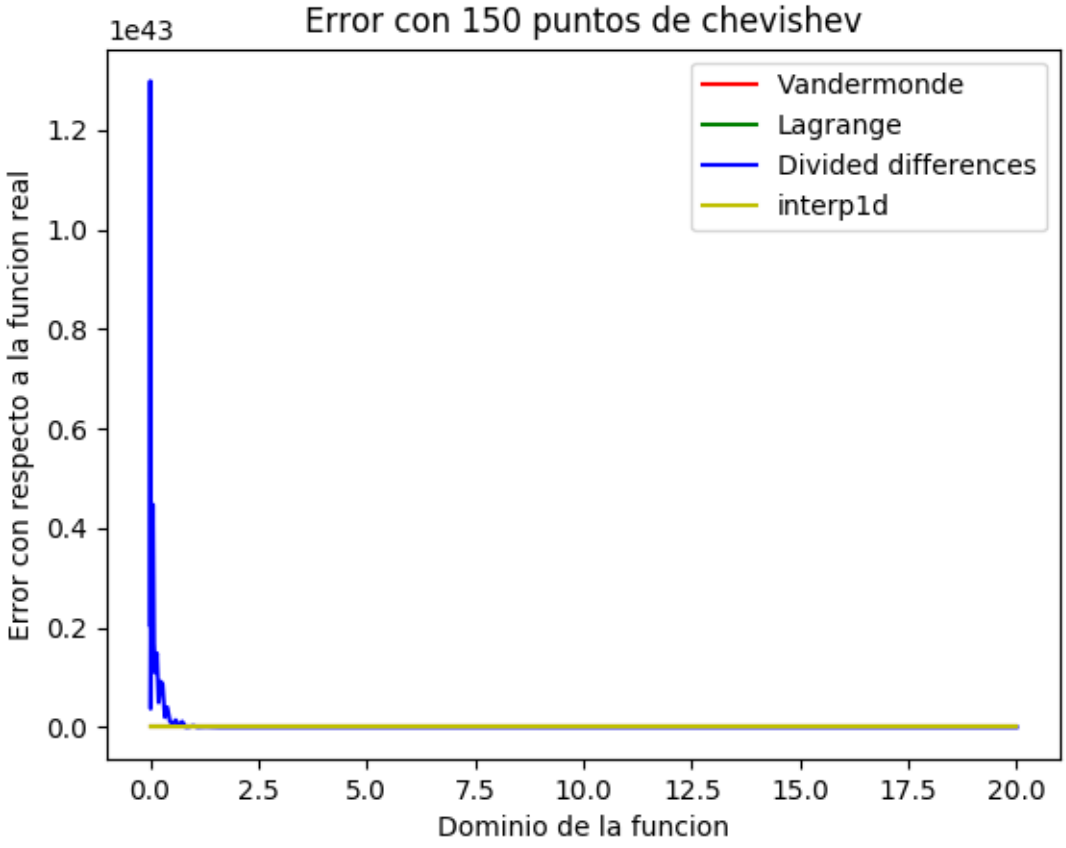


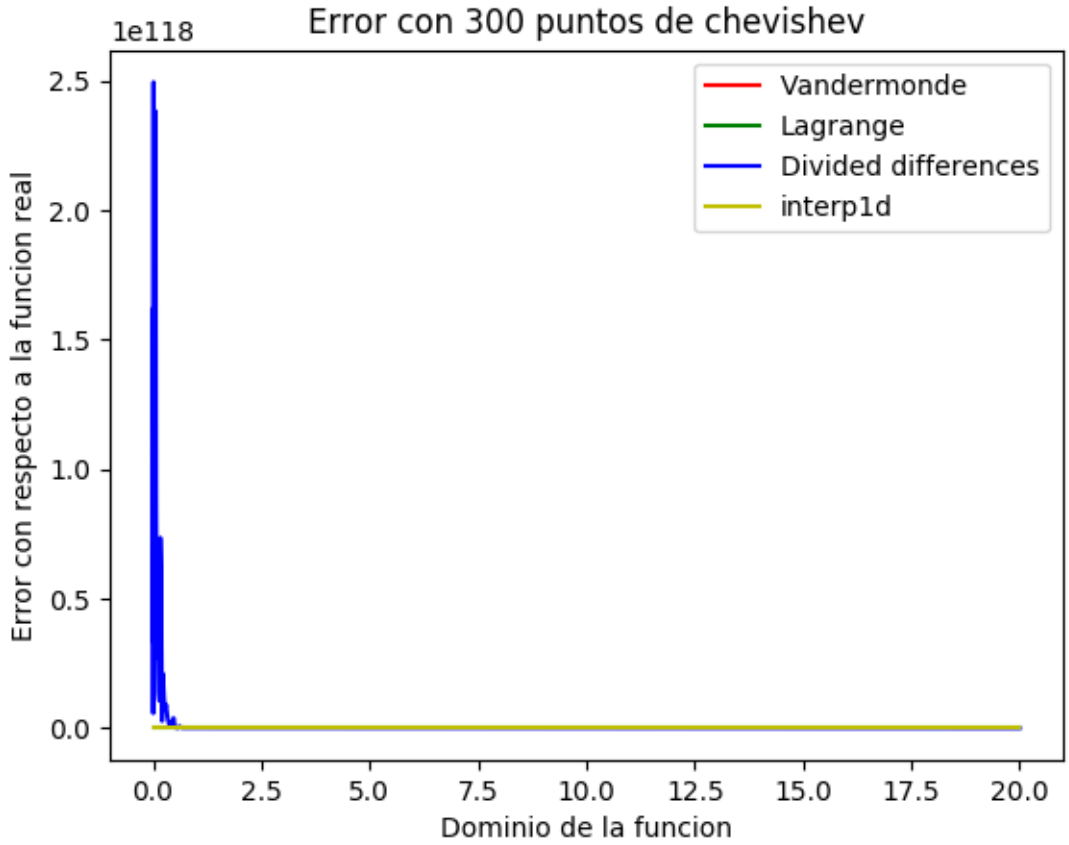




2. Puntos de *Chevishev*.


Error con 10 puntos de chevishev
1e−10
Error con respecto a la funcion real
Dominio de la funcion
Vandermonde
Lagrange
Divided differences
interp1d
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0


Error con 150 puntos de chevishev
1e43
Error con respecto a la funcion real
Dominio de la funcion
Vandermonde
Lagrange
Divided differences
interp1d
0.0
0.2
0.4
0.6
0.8
1.0
1.2
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0



No se que concluir de esto.

En base a los gráficos, mientras mayor sea la cantidad de puntos, mayor es el error resultante entre la función interpolada y la función real.

La diferencia entre usar puntos equiespaciados y *Chevyshev*, en base a estos gráficos, solo mueve el error de un limite del dominio al otro.

Quiero rescatar que, en base a un análisis un poco mas exhaustivo a cada función, tanto *interp1d* como *Lagrange* tienen un error de 0 con respecto a la función real. Esto probablemente se deba a que se están evaluando los mismos puntos con los que se interpolo. Esto es esperado en el caso de *Lagrange* debido a su naturaleza.