Tarea6
Métodos Numéricos

Benjamin Rivera

6 de octubre de 2020

Índice

1. Tarea 6

Tarea 6 de *Benjamín Rivera* para el curso de **Métodos Numéricos** impartido por *Joaquín Peña Acevedo*. Fecha limite de entrega **11 de Octubre de 2020**.

[72]:
```
import sys
import seaborn as sns
import scipy

import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve_triangular # Para backward y forward␣
 ↪substitution

from local.utils import *
from local.Tarea4 import solChol

NOTEBOOK = True
```

1.1. Ejercicio 1

Estas funciones no son utilizadas en el resto del programa pero fue solicitada en los enunciados de la tarea. Por lo que las programe pero después use las que tiene implementada la librería de python `numpy`.

[]:
```
# Parte 1
# Matriz traspuesta

def traspuesta(A,/, dtype=np.float32):
    """ Funcion que regresa la matriz traspuesta. """
    sz = A.shape
    ret = np.zeros((sz[1], sz[0]),
                   dtype=dtype)

    for i in range(sz[0]):
        for j in range(sz[1]):
            ret[j,i] = A[i,j]

     return ret

# Parte 2
# Producto de matrices

def prodMat(A, B,/, dtype=np.float32):
    szA = A.shape
    szB = B.shape

    if szA[1] == szB[0]:
        ret = np.zeros((szA[0], szB[1]), dtype=dtype)

        for i in range(szA[0]):
            for j in range(szB[1]):
                ret[i,j] = sum(A[i,k]*B[k,j] for k in range(szA[1]))
        return ret
    else:
        raise Exception("No coinciden las dimensiones")
```

[4]:
```python
# 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 existen 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)
```

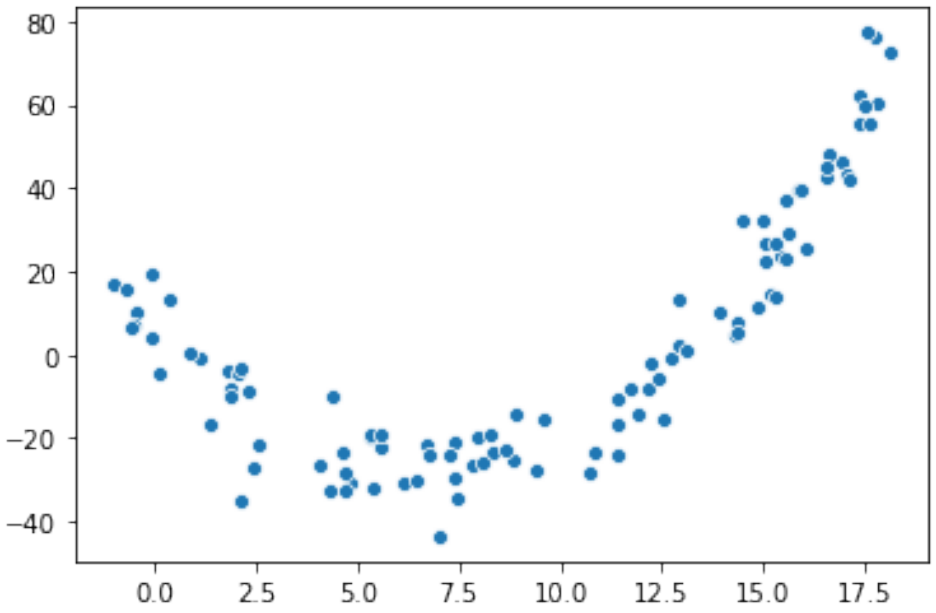
[23]:
```python
# Parte 4

def get2Dvec(path,/, dtype=np.float64, info=True):
    """ Funcion para cargar vector 2D.

    Esta funcion tratara de cargar un vector 2D de unarchivo
    de texto que tenga dos columnas (correspondientes a dos
    vectores y separada por un espacio) con k filas (donde k
    es el tamanio de los vectores que estan separados por \n)

    Los datos los guardaremos en una instancia de np.matrix

    Input:
        path := direccion del archivo para cargar los
            vectores

        dtype := tipo de dato para usar
        info := Indica si queremos extraer la informacion
    Output:
        (ret, info)
        ret := np.matrix de (2,k)
        info := Para evitar tener que hacer otro recorrido
            sobre el arreglo se puede extraer informacion
            en este recorrido
            minx := El minimo valor de x
            maxx := El maximo valor de x
    """

    try:
        ...
        if info:
            return np.matrix(ret, dtype=dtype), ret_info
        else:
            return np.matrix(Ret, dtype=dtype)
    except:
        raise Exception("Error al cargar el archivo")


def plot_ej1_1(x, y):
    """ Funcion 1 para graficar resultados.

    Esta funcion buscca graficar los datos recibidos para poder
    tomar la mejor decision respecto al grado a utilizar en la
    aproximacion a polinomios.

    Input:
        x := Valores de cordenadas x
        y := Valores de cordenadas y
    """
```

```python
def plot_ej1_2(x, y, f, rng):
    """ Funcion 2 para graficar resultados

    Esta funcion busca graficar los datos recibidos para mostrar
    la posible eproximacion obtenida por el metodo.
    Input:
        x := Valores de cordenadas x
        y := Valores de cordenadas y
        f := Funcion polinomica obtenida
        rng := particion del rango para graficar
    """

def error_ej1(p, x, y):
    """ Funcion para calcular error del polinomio. """
    return sum((p(x[i]) - y[i])**2 for i in range(len(x)))


def Ejercicio1(d, n_tabla, n,/,path='datos/', dtype=np.float64,␣
 ↪plot=True, prnt=True, ask=True):
    # Cargar datos
    tabla, info = get2Dvec(path+n_tabla,
                           dtype=dtype)
    # Valores obtenidos
    x = np.ravel(tabla[0,:])
    y = np.ravel(tabla[1,:])
    rng = np.linspace(info['minx'], info['maxx'], num=d)

    if plot and ask:
        plot_ej1_1(x,y)

    sz = len(x)
    if n < sz:
        # Crear matriz a
        A = np.ones((sz, n+1),
                    dtype=dtype)
        for i in range(n):
            A[:,i] = np.power(tabla[0,:], n-i)
        # 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])

    else:
        raise Exception("Sistema indeterminado")
```

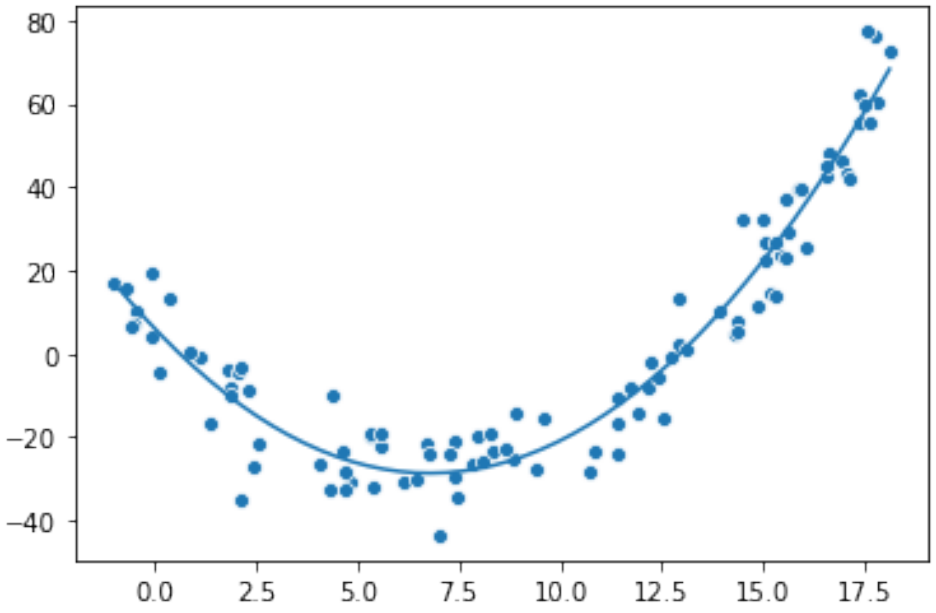
1.1.1. Conjunto 1

Claramente se puede apreciar una parabola, por eso mi primer intento fue tratar de conseguir un polinomio de grado 2. Esto es dificil de mejorar, dado la dispercion de los puntos, ademas con n's muy grandes empeiza a sufrir de *overfitting* y no mejora el error; esto se puede apreciar en el segundo intento con $n = 11$

[12]:
```
# Parte 5
if NOTEBOOK:
    Ejercicio1(50, 'puntos2D_conjunto1.txt', 2)
```

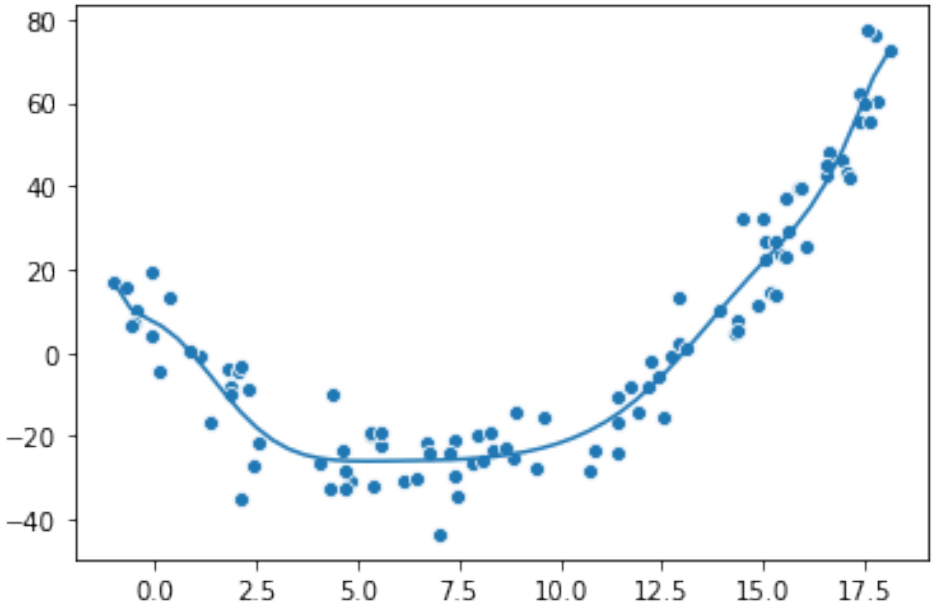



```
Seguro que quieres usar grado 2 para aproximar?
[S para mantener]: 2
n=2
```



```
Se encontraron los coeficientes
        0.7541817962086255, -10.231492940619027, 6.078939265117226
Error = 5650.301069137684
```

[32]:
```
if NOTEBOOK:
    Ejercicio1(50, 'puntos2D_conjunto1.txt', 11, ask=False)
```



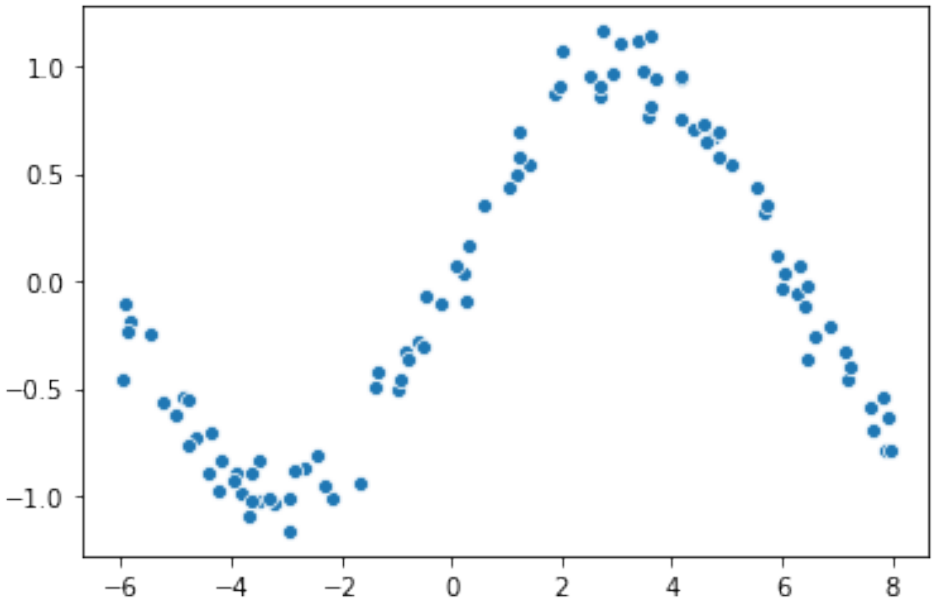

```
Se encontraron los coeficientes
        -1.3588485818714197e-08, 1.2843215253923994e-06,
-5.2305115646995115e-05, 0.0012003310105050808, ..., -2.
 ↪941809213167029,
-2.1452912757616205, -5.178739868339335, 7.361058552062617
Error = 5248.19300240789
```

1.1.2. Conjunto 2

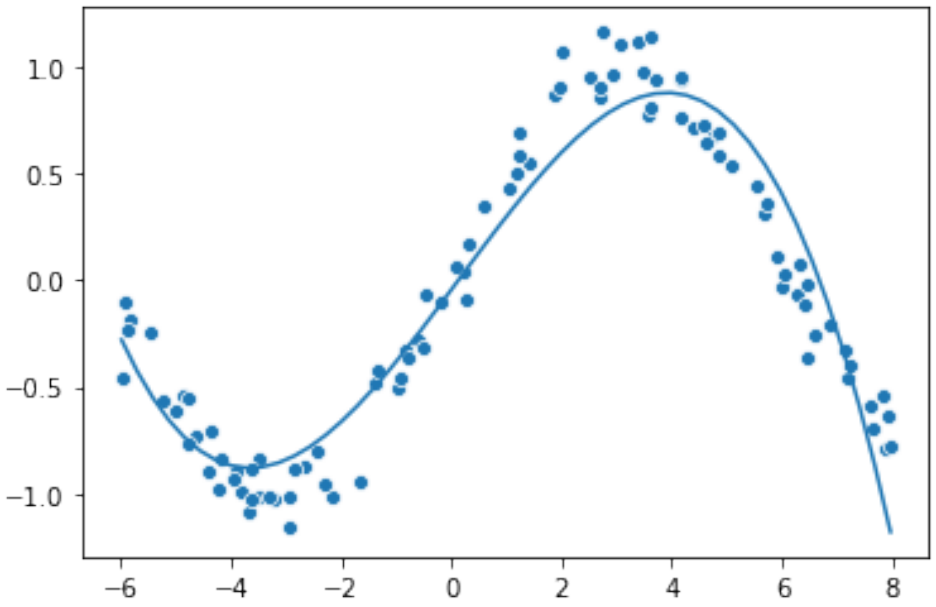
Respecto al conjunto dos lo primero que pense es que correspondia a una funcion de grado 3, sin embargo el error queda bastante grande. En el segundo intento pense que probablemente un polinomio de grado 5 se ajustaria mejor, lo que mejoro considerablemente, ya que ahora el valor es ~ 1. En otros intentos[1] vi que para bajar el error de 0 es necesario tomar un polinomio de grado 12, que ya me parece que recolecta demasiada informacion que podria no ser relevante.

[16]:
```
if NOTEBOOK:
    Ejercicio1(50, 'puntos2D_conjunto2.txt', 3)
```

[1] Que no agregare a este reporte por ser muchos y muy a prueba y error

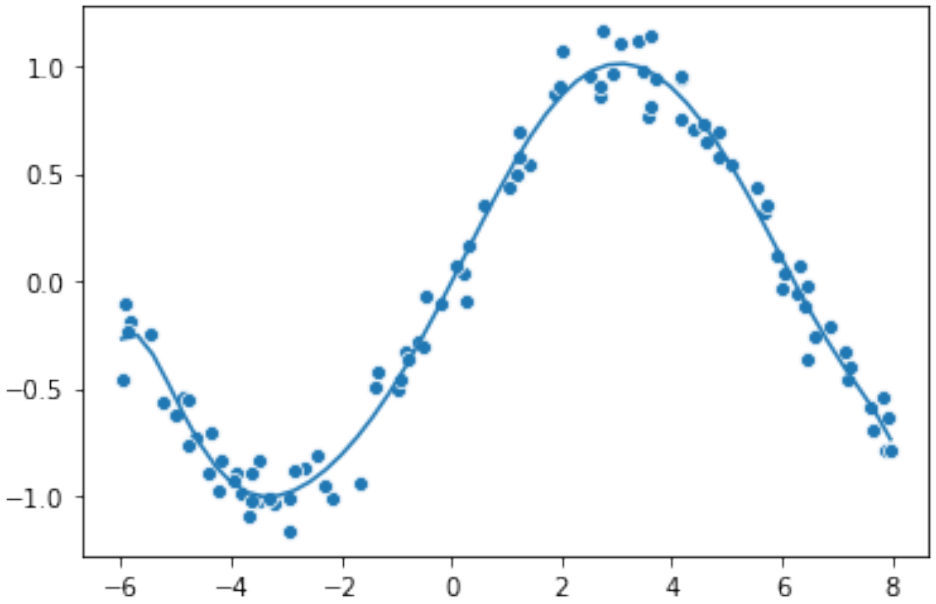



```
Seguro que quieres usar grado 3 para aproximar?
[S para mantener]: s
n=3
```




```
Se encontraron los coeficientes
        -0.008095864120276854, 0.0030857409969524373, 0.
↪347724224152898,
-0.04518101609383535
Error = 4.2437023557985745
```

[42]:
```
if NOTEBOOK:
    Ejercicio1(50, 'puntos2D_conjunto2.txt', 11, ask=False)
```

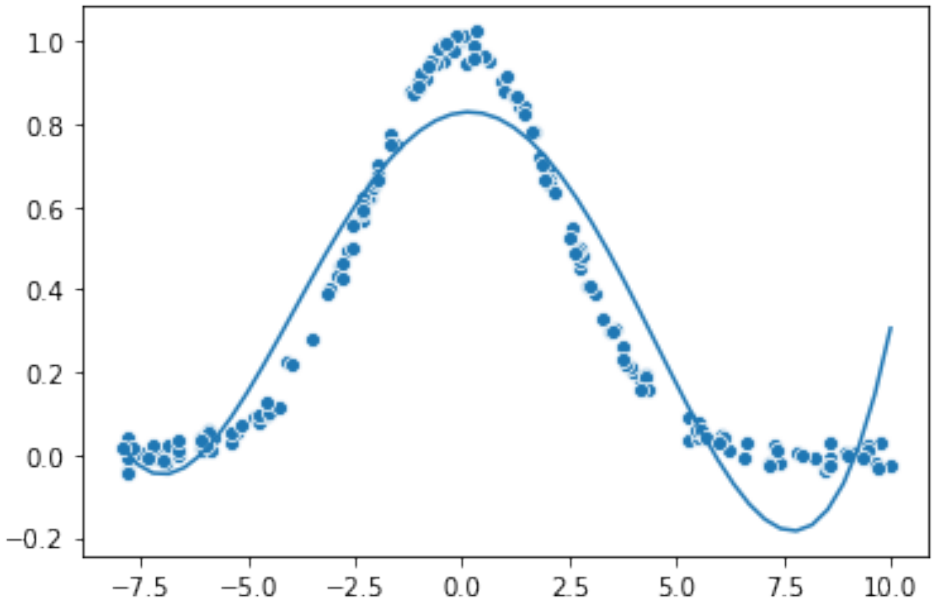



```
Se encontraron los coeficientes
        -1.5639735180840897e-09, -3.647711279013421e-09, 4.
 ↪450736533289704e-07,
-1.7100043270209563e-06, ..., -0.022473411213090114, 0.
 ↪01713626219890958,
0.49928235722555336, -0.0070410241304996795
Error = 1.0080196832546382
```

1.1.3. Conjunto 3

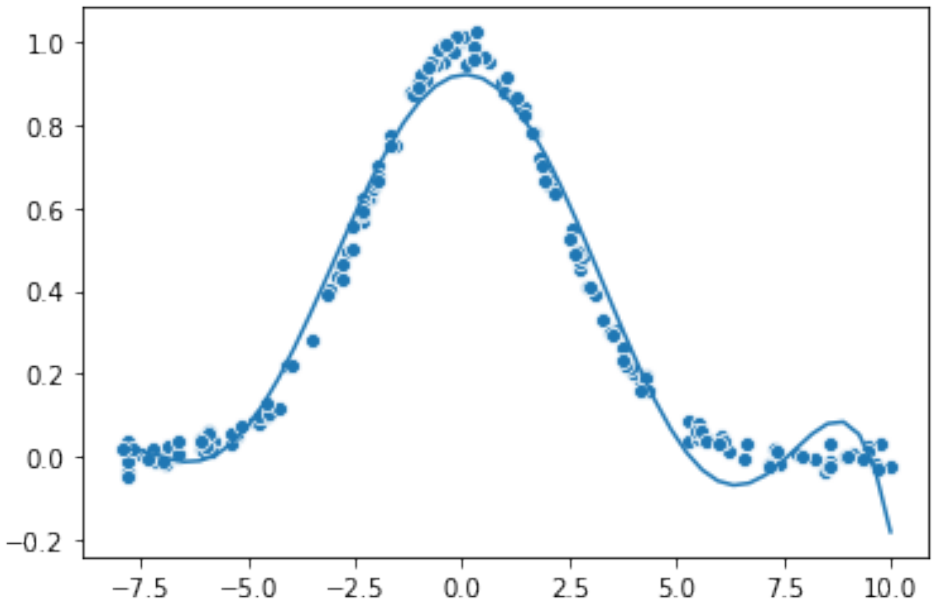
Claramente podemos ver una distribucion normal en los puntos dispersos que se muestran. El primer polinomio que tenia una apariencia similar a esta fue el de grado 4, el cual daba un error ~ 2. Despues de eso la siguiente mejora considerable fue la del polinomio de grado 6, que daba un error $\sim 0{,}5$

[50]:
```
if NOTEBOOK:
    Ejercicio1(50, 'puntos2D_conjunto3.txt', 4, ask=False)
```




```
Se encontraron los coeficientes
        0.00032206033629311026, -0.0003855417565318806, -0.
 ↪034690166313100616,
0.011550168869917873, 0.827944199634344
Error = 2.2603307586775965
```

[66]:
```
if NOTEBOOK:
    Ejercicio1(50, 'puntos2D_conjunto3.txt', 6, ask=False)
```




```
Se encontraron los coeficientes
```

```
        -7.065981612069293e-06, 9.045049435863039e-06, 0.
↪0011564541100981907,
-0.000697767703063815, -0.059029633270661845, 0.008978629606807174,
0.9215749569687338
Error = 0.578133274637197
```

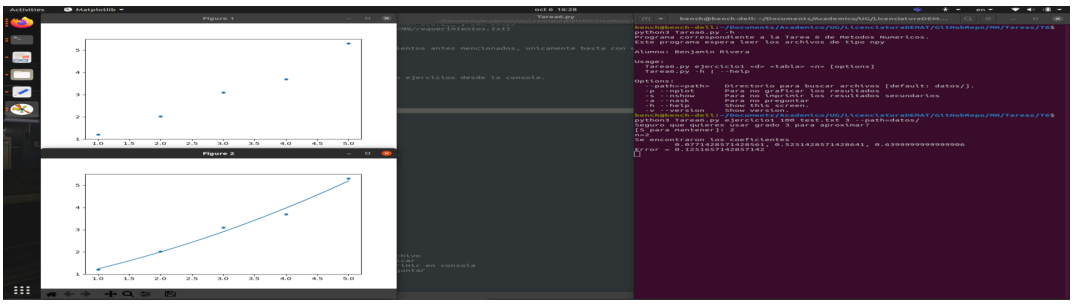
1.2. Como ejecutar

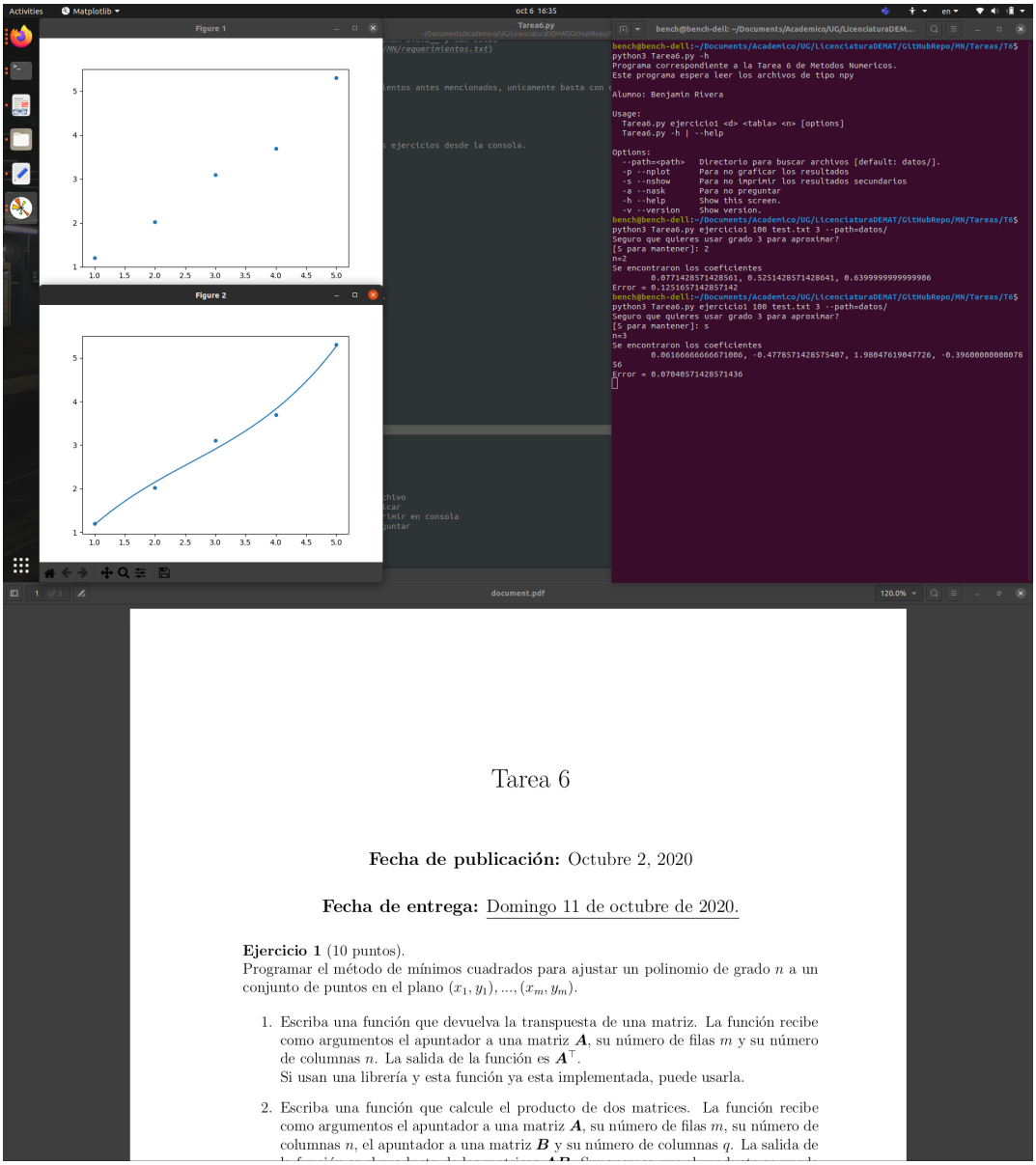
Requerimientos Este programa se ejecuto en mi computadora con la version de **Python 3.8.2** y con estos requerimientos

Jupyter En caso de tener acceso a un *servidor jupyter* ,con los requerimientos antes mencionados, unicamente basta con ejecutar todas las celdas de este *notebook*. Probablemente no todas las celdas de *markdown* produzcan el mismo resultado por las *Nbextensions*.

Consola Habrá archivos e instrucciones para poder ejecutar cada uno de los ejercicios desde la consola.

[71]:
```
usage = """
Programa correspondiente a la Tarea 6 de Metodos Numericos.
Este programa espera leer los archivos de tipo npy

Alumno: Benjamin Rivera

Usage:
  Tarea6.py ejercicio1 <d> <tabla> <n> [options]
  Tarea6.py -h | --help

Options:
  --path=<path>   Directorio para buscar archivos [default: datos/].
  -p --nplot      Para no graficar los resultados
  -s --nshow      Para no imprimir los resultados secundarios
  -a --nask       Para no preguntar
  -h --help       Show this screen.
  -v --version    Show version.
"""
```





Tarea 6

Fecha de publicación: Octubre 2, 2020

Fecha de entrega: Domingo 11 de octubre de 2020.

Ejercicio 1 (10 puntos).
Programar el método de mínimos cuadrados para ajustar un polinomio de grado n a un conjunto de puntos en el plano $(x_1, y_1), ..., (x_m, y_m)$.

1. Escriba una función que devuelva la transpuesta de una matriz. La función recibe como argumentos el apuntador a una matriz $\boldsymbol{A}$, su número de filas m y su número de columnas n. La salida de la función es $\boldsymbol{A}^\top$.
   Si usan una librería y esta función ya esta implementada, puede usarla.

2. Escriba una función que calcule el producto de dos matrices. La función recibe como argumentos el apuntador a una matriz $\boldsymbol{A}$, su número de filas m, su número de columnas n, el apuntador a una matriz $\boldsymbol{B}$ y su número de columnas q. La salida de