Métodos Numéricos

Trabajo práctico N°2

Cuadrados mínimos

Grupo 11: Martin Amagliani
Olivia De Vincenti
Nicolás Fernandez Pelayo

Profesor: Pablo Ignacio Fierens

Fecha de entrega: 18 de septiembre de 2020

Buenos Aires, Argentina

Consigna

Grupo impar

1. Escriba una función que resuelva, usando descomposición QR, el problema de cuadrados mínimos lineal:

$$\vec{x}^* = argmin\left\|\mathbf{A}\vec{x} - \vec{b}\right\| \tag{1}$$

El nombre de la función debe ser `leastsq` y debe recibir como parámetros $\mathbf{A}$ y $\vec{b}$, en ese orden, ambos como arreglos de `Numpy`. El vector $\vec{b}$ debe ser un arreglo con 1 columna (no 0 columnas). La función debe estar dentro de un archivo de nombre `leastqr.py`.

La función no puede usar una función ya hecha (de alguna biblioteca) que realice la descomposición QR. Tampoco puede utilizar funciones de alguna biblioteca que resuelvan sistemas triangulares.

2. Escriba una función que pruebe el correcto funcionamiento de la implementada en el ítem anterior. La función debe llamarse `test`, sin argumentos, y se la debe colocar en el mismo archivo que la anterior.

Funciones

Índice de Códigos

1. Ejercicio 1

1.1. leastsq

Recibe la matriz $\mathbf{A} \in \mathbb{R}^{m \times n}$ y el vector $\vec{b} \in \mathbb{R}^m$ del problema de cuadrados mínimos (1) y lo resuelve utilizando la descomposición QR, devuelve el vector $\vec{x}$, solución al problema. Se pide que $\mathbf{A}$ sea de rango completo y con cantidad de filas mayor o igual a la de columnas, también es importante que las dimensiones de $\mathbf{A}$ y $\vec{b}$ sean compatibles.

En caso de que lo recibido no sea correcto, imprime un string en pantalla indicando lo ocurrido y devuelve -1.

Código 1: Definición de leastsq

```
def leastsq(A, b):
    # Verificación
    if not verify(A, b):                    # Verifico que se cumplan los requisitos
        return -1

    # Factorización QR
    QR = desc_qr(A)                         # Realizo la descomposición QR
    Q1 = QR[0]                              # Recupero Q1 de m*n y ortonormal de A
    R1 = QR[1]                              # Recupero R1 triangular superior de n*n

    # Resuelvo para encontrar x mínima      # R1.x = Q^T.b
    C = np.dot(np.transpose(Q1), b)         # C = Q1^T.b
    x = rev_subs(R1, C)                     # R1.x = C

    return x
```

1.2. desc_qr

Recibe una matriz $\mathbf{A}$ y devuelve $\mathbf{Q_1}$ ortogonal a $\mathbf{A}$ y $\mathbf{R_1}$ triangular superior tal que:

$$\mathbf{A} = \mathbf{Q_1}.\mathbf{R_1} \tag{2}$$

Para encontrar $\mathbf{Q_1}$ y $\mathbf{R_1}$ usamos el algoritmo de ortogonalización de Gram-Schmidt:

Sean $\vec{p}_i$ *las columnas de* $\mathbf{A}$, $\vec{q}_i$ *las columnas de* $\mathbf{Q_1}$ *y* r_{ij} *las componentes de* $\mathbf{R_1}$:

$$\textit{Se hace} \begin{cases} \vec{u}_1 = \vec{p}_1 \\ \vec{u}_i = \vec{p}_i - \sum_{k=1}^{i-1} \frac{\langle \vec{p}_i, \vec{q}_k \rangle}{\|\vec{q}_k\|} \vec{q}_k \quad \textit{con } i > 1 \end{cases} \tag{3}$$

$$\textit{Entonces} \qquad \vec{q}_i = \frac{\vec{u}_i}{\|u_i\|} \qquad r_{ij} = \begin{cases} \|\vec{u}_i\| & \textit{si } i = j \\ \langle \vec{p}_j, \vec{q}_i \rangle & \textit{si } i \neq j \end{cases}$$

Código 2: Definición de desc_qr

```
def desc_qr(A):
    m = get_dim(A)                                  # Defino las dimensiones m y n
    n = get_dim(A[0])

    Q1 = np.zeros(shape=(m, n))                     # Creo las matrices base
    R1 = np.zeros(shape=(n, n))
    for i in range(n):
        aux = A[:, i]
        for k in range(i):                  # Calculo las proyecciones (Q1[:, k] nunca va a ser nula)
            aux = aux - (np.dot(A[:, i], Q1[:, k])/norma2(Q1[:, k])) * Q1[:, k]
        Q1[:, i] = normalize(aux)                   # Normalizo la columna
        R1[i, i] = norma2(aux)                      # Calculo la diagonal de R1
        j = i + 1
        while(j < n):
            R1[i, j] = np.dot(A[:, j], Q1[:, i])    # Calculo la esquina superior de R1
            j += 1

    return Q1, R1
```

1.3. verify

Revisa que se cumplan los requisitos de **leastsq**:

- La cantidad de filas de $\mathbf{A}$ es mayor o igual a la de columnas
- La matriz $\mathbf{A}$ es de rango completo
- El vector $\vec{b}$ tiene tantas filas como $\mathbf{A}$

Devuelve True si se cumplen todos, sino imprime un mensaje indicando el problema y devuelve False.

Código 3: Definición de verify

```
def verify(A, b):
    r = True
    m = get_dim(A)                                  # Defino las dimensiones m y n
    n = get_dim(A[0])
    if m < n:                                       # Verifico que m >= n
        print("POR FAVOR INGRESE UNA MATRIZ CON MAYOR O IGUAL CANTIDAD FILAS
     ↪ QUE COLUMNAS")
        r = False
        if np.linalg.matrix_rank(A) != m:           # Verifico que sea de rango completo (con m)
            print("POR FAVOR INGRESE UNA MATRIZ DE RANGO COMPLETO")
    elif np.linalg.matrix_rank(A) != n:             # Verifico que sea de rango completo (con n)
            print("POR FAVOR INGRESE UNA MATRIZ DE RANGO COMPLETO")
            r = False
    if len(b) != m:                                 # Verifico que la dimensión de b tenga sentido
        print("POR FAVOR CHEQUEE LAS DIMENSIONES DE b CON RESPECTO A LAS DE A")
        r = False

    return r
```

1.4. rev_subs

Recibe la matriz $\mathbf{A}$ triangular superior y el vector $\mathbf{y}$. Resuelve la ecuación lineal $\mathbf{A}.\vec{x} = \vec{y}$ utilizando el método de sustitución hacia atrás, ya que $\mathbf{A}$ es triangular superior. Devuelve $\vec{x}$.

Código 4: Definición de rev_subs

```
def rev_subs(A, y):
    m = get_dim(A)                                          # Defino m
    x = np.zeros(shape=(m, 1))                              # Crea el vector base
    for i in range(m):
        d = 0
        for j in range(i):
            d += A[m - i - 1, m - j - 1] * x[m - j - 1]     # Calcula los términos conocidos
        x[m - i - 1] = (1/A[m - i - 1, m - i - 1])*(y[m - i - 1] - d)    # Calcula una de las componente

    return x
```

1.5. get_dim

Calcula la dimensión de un arreglo usando `len`. Si esta es 1 y no utiliza el formato de `len`, se atrapa la excepción y se setea n en 1. Esto resulta necesario cuando la cantidad de columnas de la matriz es 1 y fue ingresada en forma de vector.
En caso de que lo ingresado sea otra cosa, devuelve -1.

Código 5: Definición de get_dim

```
def get_dim(M):
    try:
        n = len(M)                      # Calculo la dimensión del arreglo
    except TypeError:                   # Si tira excepción, me fijo si es un número
        if isinstance(M, int) or isinstance(M, float):
            n = 1                       # Si es, devuelve 1
        else:
            n = -1                      # Si no, devulve -1 (no va a ocurrir el contexto de este TP)

    return n
```

1.6. norma2

Devuelve la norma 2 del vector recibido.

Código 6: Definición de norma2

```
def norma2(v):
    n = 0
    for i in range(len(v)):
        n = n + (v[i])**2               # Elevo cada componente al cuadrado y las sumo
    return np.sqrt(n)                   # Devuelvo la raíz de la suma de los cuadrados
```

1.7. normalize

Devuelve la versión normalizada del vector recibido.

Código 7: Definición de normalize

```
def normalize(v):
    v = v/norma2(v)

    return v
```

2. Ejercicio 2

2.1. test

Función de prueba. Llama a leastsq con los distintos valores de $\mathbf{A}$ y $\vec{(b)}$ contenidos en el banco de pruebas, intentando probar los casos más representativos. Imprime con 15 cifras significativas el resultado obtenido junto con el de la función `numpy.linalg.lstsq`, la cual consideramos que funciona correctamente. Para facilitar la lectura y prolijidad, los casos se revisan de a uno, por lo que el usuario debe apretar enter para analizar el siguiente caso. Se consideró incluir en el testeo:

- Matriz $\mathbf{A} \in \mathbb{R}^{4\times2}$ (cumple con los 3 requisitos)
- Matriz $\mathbf{A} \in \mathbb{R}^{3\times2}$ (cumple con los 3 requisitos)
- Matriz $\mathbf{A} \in \mathbb{R}^{3\times1}$ (cumple con los 3 requisitos)
- Matriz $\mathbf{A} \in \mathbb{R}^{3\times3}$ (cumple con los 3 requisitos)
- Matriz $\mathbf{A} \in \mathbb{R}^{5\times4}$ (cumple con los 3 requisitos)
- Matriz $\mathbf{A} \in \mathbb{R}^{5\times4}$ con $\vec{b} \in \mathbb{R}^3$ (error de dimensiones)
- Matriz $\mathbf{A} \in \mathbb{R}^{4\times2}$ de rango incompleto
- Matriz $\mathbf{A} \in \mathbb{R}^{3\times4}$ (más columnas que filas)
- Matriz $\mathbf{A} \in \mathbb{R}^{3\times4}$ de rango incompleto con $\vec{b} \in \mathbb{R}^4$ (todos los errores)
- Matriz $\mathbf{A} \in \mathbb{R}^{10\times8}$ (cumple con los 3 requisitos)

Código 8: Definición de test

```
def test():

    # Banco de pruebas:
    #              A                                                                b
    test_bank = (
    (np.array([[1.02, 1], [1.01, 1], [0.94, 1], [0.99, 1]]),                  np.array([2.05, 1.99, 2.02, 1.93])),
    (np.array([[-1, -1], [1, 0], [-1,  1]]),                                   np.array([4, 2, 3])),
    (np.array([[-1], [1], [-1]]),                                              np.array([1, 7, 3])),
    (np.array([[2, 1, 1], [1, 2, 1], [1, 1, 2]]),                              np.array([1, 3, -1])),
    (np.array([[2, 1], [1, 2], [1, 1]]),                                       np.array([7, 5, -1])),
    (np.array([[1, 1, 1, 1], [1, 5, 6, 4], [1, 4, 8, 6], [1, 2, 1, 3], [1, 3, 2, 8]]), np.array([1, 8, 6, -5, 3])),
    (np.array([[1, 1, 1, 1], [1, 5, 6, 4], [1, 4, 8, 6], [1, 2, 1, 3], [1, 3, 2, 8]]), np.array([1, 8, 6, -5])),
    (np.array([[-1, -1], [-1, -1], [-1,  -1], [-1, -1]]),                       np.array([1, 2, 3, 4])),
    (np.array([[-1, 1, 0, 3], [-1, 2, -1, -5], [7, -0, -1, 4]]),               np.array([4, 2, 0])),
    (np.array([[-1, 1, 0, 3], [1, 2, -1, -5], [-1, 4, -1, 1]]),                np.array([1, 2, 3, 4])),
    (np.array([[1, 1, 1, 1, 1, 1, -1, -2], [1, 5, 6, 4, 1, 1, -1, 0], [1, 4, 8, 6, 1, 1, 1, 1], [1, 2, 1, 3, 1, 1, 1, 4],
              [1, 3, 2, 8, 1, 1, 1, -1], [2, 4, 5, 1, 1, -1, 1, 3], [1, 2, 3, 4, 1, 1, 1, 4], [7, 6, 5, 4, 1, -1, 1, 8],
              [0, -2, 1, 3, 1, -1, 0, 1], [7, 6, 5, 4, 2, -1, 3, 0]]),      np.array([1, 8, 6, -5, 1, 2, 3, 4, -6, 1])))

```

```
20    print("\n PROGRAMA DE PRUEBA DE leastsq")
21
22    # Testeo los casos del banco de prueba
23    for i in range(len(test_bank)):
24        print("_____________________________________________________________________________")
25        A = test_bank[i][0]                                    # Recupero A para el caso i
26        b = test_bank[i][1]                                    # Recupero b para el caso i
27
28        print("   " * get_dim(A[0]), "A", "   "*(get_dim(A[0]) + 3), "b")     # Imprimo A y b
29        for k in range(get_dim(A)):
30            print("[ " + ' '.join(['{:4}'.format(float(item)) for item in A[k]]), "]", end=" ")
31            if k < get_dim(b):                                 # Se considera el caso que b no tenga las
32                print("\t " * 2, "[", float(b[k]), "]")        # dimensiones correctas
33        for j in range(k+1, get_dim(b)):
34            print("\t\t"*(get_dim(A[0])), " [", float(b[j]), "]")  # Si b tiene más filas que A, las imprime
35        print("\n")
36
37        x = leastsq(A, b)    # Resuelvo el problema de los cuadrados mínimos con descomposición QR
38
39        if not isinstance(x, type(-1)):                        # Verifico que no hubo error
40
41            y = np.linalg.lstsq(A, b, None)                    # Resuelvo el problema con numpy
42            print("  x según leastqr" + "\t "*3 + "   x según numpy")     # Imprimo los resultados
43            for c in range(get_dim(x)):
44                print("[", '{:{width}.12f}'.format(x[c][0], width=15), "]", "\t"*2, "[", '{:{width}.12f}'.
      ↪ format(y[0][c], width=15), "]")
45
46        else:
47            print("ERROR: Los parámetros ingresados no cumplen con alguno de los requisitos")
48
49        print("_____________________________________________________________________________")
50        print("\nPULSA ENTER PARA CONTINUAR")          # Pulsar enter para evaluar el
51        input()                                        # caso que sigue
52
53    print("FIN DEL PROGRAMA DE PRUEBA")
54
55    return
```