Taller # 1: Conceptos básicos

Elaborado por Denis Perez / Heyzel Moncada

25 de Noviembre de 2019

Pregunta 1.a

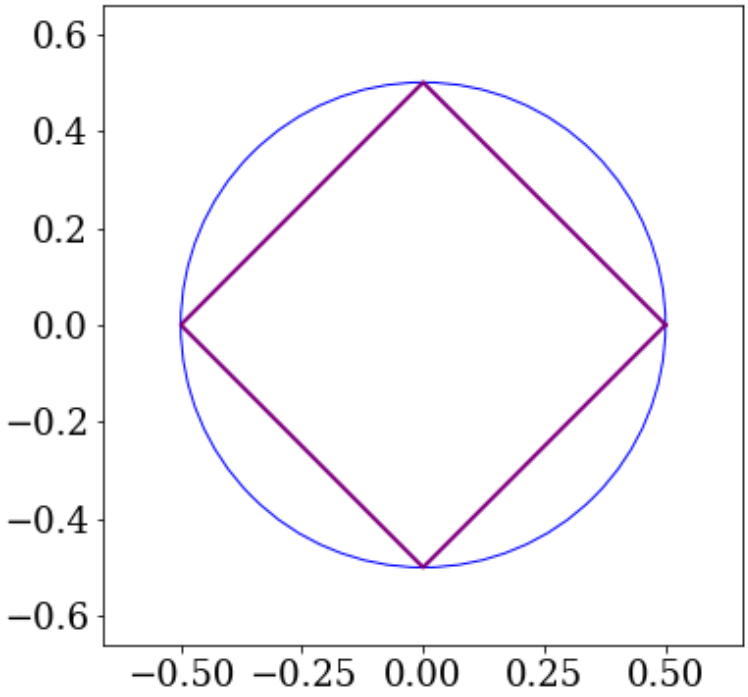
Genere una animación a través de una secuencia de gráficas, que permita visualizar el circulo de radio 1, junto con los sucesivos polígonos que se forman al aumentar el valor de $n \geq 4$ hasta N, con N suficientemente grande.

```
def init():
    line.set_data([], [])
    return line,

def animate(i):
    coseno = lambda i,k : r * (s.cos(((2*np.pi)*(k))/i))        #(1)
    seno = lambda i,k : r * (s.sin(((2*np.pi)*(k))/i))          #(2)
    xvalues = []
    yvalues = []

    for j in range(i+5):
        xvalues.append(coseno(i+4,j))
        yvalues.append(seno(i+4,j))
    line.set_color('purple')
    line.set_data(xvalues, yvalues)

    return line

anim = animation.FuncAnimation(fig, animate, init_func = init,
                               frames = n, interval = 150, blit = True)
```

Códigos

El archivo que da solución a la pregunta se llama **ejercicio1.py**

La rutina init crea el frame inicial de la animación, que en un principio va vacío. La rutina animate es quien hace como tal la animación, para ello definimos dos funciones lambda (**(1)** y **(2)**) que se definen en el modelo matemático, luego calculamos el valor de los puntos del polígono y dicho polígono es el que se muestra en el frame. El rango del ciclo **for** está hasta el valor **i+5** , esto se debe a que la rutina **FuncAnimation** llama a animate con el valor **i = 0** y el primer polígono que debemos hacer es el de lado 4. Dando como resultado la el gif pedido en la pregunta



.

Pregunta 1.b

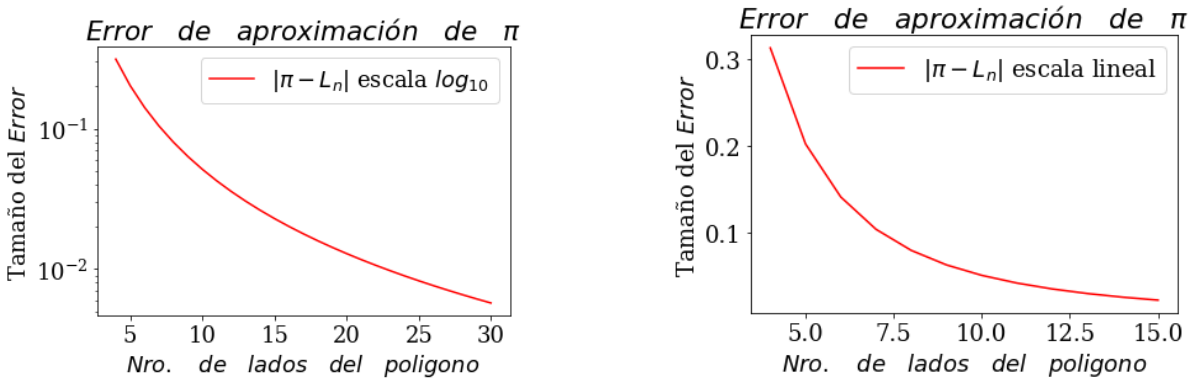
Genere una gráfica adicional, que muestre el *error de la aproximación*, es decir, que muestre $|\pi - L_n|$ para $n = 4, 5, ..., N$ Comente lo observado, teniendo en mente la siguiente pregunta: ¿Qué espera que ocurra con el valor de L_n cuando $n \to \infty$?

```
def polygon(n):

    coseno = lambda n,k : r * (s.cos(((2*np.pi)*(k))/n))
    seno = lambda n,k : r * (s.sin(((2*np.pi)*(k))/n))
    xvalues = []
    yvalues = []

    for i in range(n+1):
        xvalues.append(coseno(n,i))
        yvalues.append(seno(n,i))
        #ax.plot(xvalues,yvalues, 'r')


    x = (xvalues[1]-xvalues[0])**2
    y = (yvalues[1] - yvalues[0])**2

    L = 0
    L = (n)*np.sqrt(x + y)
    pi_aproximations.append(L)
```

```
for i in range(4,n+1):
    polygon(i)
result = list(map(lambda y: np.abs(np.pi-y), pi_aproximations))     (1)
```

Códigos

El archivo que da solución a la pregunta se llama **ejercicio1.py**



(a) Polígono de 30 lados

(b) Polígono de 15 lados

Fig. 1: Errores de aproximación de π

Definimos la rutina **polygon** la cual se encarga de hacer el calculo de la longitud de los polígonos como se define en el modelo matemático, luego en (1) realizamos el calculo del error para cada longitud calculada en las llamadas a **polygon**. En las figuras 1a y 1b se muestran las gráficas del *error de la aproximación de* π para polígonos de 30 y 15 lados respectivamente.

Cuando $n \to \infty$ se espera que el error absoluto dado por: $|\pi - L_n| \to 0$.

Pregunta 2.a

¿Si Usted necesita programar una rutina para el producto matriz-vector, la forma en cómo un arreglo bidimensional es almacenado en la memoria del computador, podrıa influir en su forma de programación? Justifique sus respuesta.

Si. El almacenamiento en memoria de los arreglos bidimensionales podría afectar significativamente la eficiencia del programa, sin embargo, por la forma en que se almacena la memoria, el acceso a dichos arreglos de manera lineal debería ser mas eficiente.

Pregunta 2.b

Compare mediante una gráfica el tiempo de CPU que requiere **MatVecFila** y **MatVecCol** para diferentes valores n. Inicie con $n = 100$ hasta un valor de n de su elección pero suficientemente grande. (Use A y x aleatorios). Comente lo observado en la gráfica.

Códigos

El archivo que da solución a la pregunta se llama **ejercicio2.py**

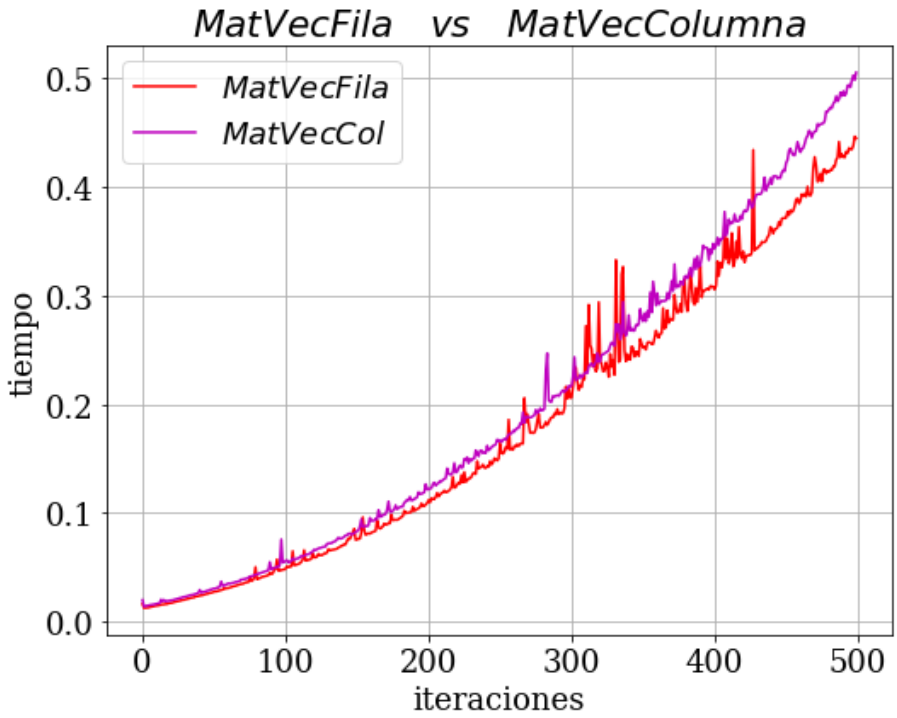


Fig. 2: Orientación por filas vs orientación por columnas

En la gráfica 2 se observa que el tiempo para el producto matriz-vector orientado por filas es mucho menor que el tiempo orientado por columnas (probando con n=600)

Pregunta 2.c

¿La gráfica le sirve de soporte para su respuesta del ítem **2.a**?

En efecto, la gráfica permite visualizar de manera clara lo que se explicaba en el ítem **1.a**

Pregunta 3.a

Describa detalladamente la forma kji. Para dicha descripcion tenga en cuenta como se recorren las matrices A y B y como se va construyendo la matriz C.

Recorrer una matriz de la forma kji recorre la matriz A por columnas y la matriz B por filas. La matriz C se va ir construyendo por columnas, de tal forma que la suma de la multiplicación de cada una de las columnas de A por las filas de B generaran la matriz C.

Algoritmo 1 MatMatFormakji

1: A$\in \mathbb{R}^{mxp}$ y B$\in \mathbb{R}^{pxn}$
2: $C := zeros(m, n)$
3: **for** k **= 1,2** $\cdots$, p **do**
4: **for** j **= 1,2** $\cdots$, n **do**
5: **for** i **= 1,2** $\cdots$, m **do**
6: $c_{ij} = c_{ij} + a_{ik}b_{kj}$
7: **end for**
8: **end for**
9: **end for**

Tabla 1: Algoritmo Pseudo formal forma kji

Pregunta 3.b

Realice el esquema kji.

$$C = \sum_{k=1}^{p} \begin{bmatrix} | \\ \hat{a_k} \\ | \end{bmatrix} \begin{bmatrix} - & \hat{b_k} & - \end{bmatrix}$$

Fig. 3: Orientación por filas vs orientación por columnas

La fig: 3 muestra el esquema de la forma kji.

Pregunta 3.c

Escriba el algoritmo pseudo formal de la forma kji.

0.1 Algoritmo pseudo formal de la multiplicacion kji Tabla:1.

Pregunta 3.d

Programe el Algoritmo anterior en una rutina llamada MatMatFormakji. Asegúrese que su rutina funcione.

```
def MatMatFormakji(A, x):
    C = [[0 for i in range(n)] for j in range(m)]
    start_time = time()
    for k in range(len(x)):
        for j in range(len(x[0])):
            for i in range(len(A)):
                C[i][j] = C[i][j] + A[i][k]*x[k][j]
    time_kji.append(time()-start_time)
```

Códigos

El archivo que da solución a la pregunta se llama **ejercicio3.py**

El algoritmo programado multiplica dos matrices de la forma kji, indexando primeramente las columnas de A y las filas de B para generar una matriz C que sera rellenada por columnas. Del mismo modo se generaron matrices aleatorias de tamaño cuadrado para comprobar el tiempo que se tarda para realizar el computo.

Pregunta 3.d

¿En que contexto, forma kji presenta ventajas en relacion a la forma ijk?

El caso en que i es muy grande con respecto a k, la forma kji presenta un incremento de computo con respecto a la forma ijk ya que los accesos a memoria son mas rápidos porque esta accediendo a posiciones de memoria contigua.

Pregunta 4.a

Demuestre que T define una transformación lineal de $\mathbb{R}^n$ en $\mathbb{R}^m$. Nota: En este caso sencillamente se dice que la matriz A define una transformación lineal de $\mathbb{R}^n$ en $\mathbb{R}^m$

Primero demostraremos que $T : \mathbb{R}^n \to \mathbb{R}^m$. Sea $A \in \mathbb{R}^{mxn}$ y sea $\vec{x} \in \mathbb{R}^n$ definidos como

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}_{mxn} \quad \text{y } \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}_{nx1} \quad \text{Tenemos que } T(\vec{x}) = A\vec{x} \text{ es}$$

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ a_{21}x_2 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{pmatrix}_{mx1}$$

Así pues, el resultado de operar $A\vec{x}$ es un vector de $\mathbb{R}^m$.

Ahora bien, sabemos que para que una transformación sea lineal ésta debe cumplir que

$$T(\alpha x + y) = \alpha T(\vec{x}) + T(\vec{y}) \tag{1}$$

. Entonces, sean $A \in \mathbb{R}^{mxn}$, $\vec{x}, \vec{y} \in \mathbb{R}^n$ y $\alpha \in \mathbb{R}$. Tenemos que

$T(\alpha\vec{x} + \vec{y})$

$= A(\alpha\vec{x} + \vec{y})$ **Definición de** T

$= A\alpha\vec{x} + A\vec{y}$ **Distributividad de matrices**

$= \alpha A\vec{x} + A\vec{y}$ **Conmutatividad**

$= \alpha T(\vec{x}) + T(\vec{y})$ **Defínción de** T

De esta manera, queda demostrado que T define una transformación lineal de $\mathbb{R}^n$ en $\mathbb{R}^m$

Pregunta 4.b

¿Es $T(\vec{x}) = A\vec{x} + b$ con $b \in \mathbb{R}^m$ una transformación lineal?, ¿Qué es una transformación afín?, ¿Es $T(\vec{x})$ una transformación afín?

En general, $T(\vec{x}) = A\vec{x}+\vec{b}$ no es una transformación lineal[1] ya que no cumple con la propiedad (1). Una transformación afín viene dada por una transformación lineal aplicada a un vector $\vec{x}$ más un vector $\vec{b}$[2], es decir $T(\vec{x}) = A\vec{x}+\vec{b}$. Si, en efecto lo es, ya que cumple con lo anteriormente dicho.

Pregunta 4.c

Programe una rituna bajo el nombre de `transformacion(`n`)` que grafique los puntos $p_i \in \mathbb{R}^2$ con $i = 1, 2, 3, ..., n$ definidos como:

$$p_i = \begin{cases} A_1x + b_1 & si \quad 0 \leqslant r_i < 0.85 \\ A_2x + b_2 & si \quad 0.85 \leqslant r_i < 0.92 \\ A_3x + b_3 & si \quad 0.92 \leqslant r_i < 0.99 \\ A_4x & si \quad 0.99 \leqslant r_i \leqslant 1 \end{cases}$$

Pruebe su rutina con $n = 500, 1000, 3000, 10000$ ¿Qué observa? ¿Qué son los puntos p_i en relación al punto x?

```
def transformacion(n):
    for i in range(n):
        r = random.uniform(0,1)
        if(r < 0.85):
            xvalues.append(0.445) # A1x + b1
            yvalues.append(2.005)
        elif(r < 0.92):
            xvalues.append(-0.03) # A2x + b2
            yvalues.append(0.665)
        elif(r < 0.99):
            xvalues.append(0.065) # A3x + b3
            yvalues.append(0.665)
        elif(r <= 1):
            xvalues.append(0)     # A4x
            yvalues.append(0.08)
```

Códigos

El archivo que da solución a la pregunta se llama **ejercicio4.py**

[1] Solo en el caso en que $\vec{b} = \vec{0}$, T es una transformación lineal

[2] Dicho vector $\vec{b}$ en una transformación afín es llamado traslación

Se observa que sin importar el valor de n los puntos graficados siempre serán los mismos. Los p_i son puntos obtenidos al aplicar una transformación afín al punto x.