UNIVERSIDAD DE BUENOS AIRES
FACULTAD DE INGENIERÍA
<75.12> ANÁLISIS NUMÉRICO

DATOS DEL TRABAJO PRÁCTICO

1	2020	Resolución de Sistemas de
	AÑO	Ecuaciones Lineales
	2do	"Conducción de calor 2D en un anillo"
TP NRO	CUAT	TEMA

INTEGRANTES DEL GRUPO

	Puglisi, Agustín	104245
	APELLIDO Y NOMBRE	PADRÓN
	Apud Benavente, María	103951
GRUPO	APELLIDO Y NOMBRE	PADRÓN

Objetivos

I. Utilizar métodos iterativos para la resolución de ecuaciones no lineales.
II. Verificar experimentalmente los resultados teóricos respecto a la convergencia de los distintos métodos para los diferentes sistemas.

Desarrollo

Aclaración: Usamos python, se nos dificultó la traducción de matlab a python entonces utilizamos los archivos de entrada con el N de padrón 99999.

Además, tuvimos problemas con la resolución de la matriz más grande, debido a que luego de pasar más de 12 horas no lo lograba resolver, y no encontramos forma de optimizar el código, para reducir el tiempo de procesamiento.

a. Función x=solver_SOR(A,b) presentada en anexo 1

b. Para el w óptimo usamos la matriz mediana de dimensión 3600. Decidimos utilizar una tolerancia alta y para ello elegimos un valor de 0.1 y nos dio como resultado lo siguiente:

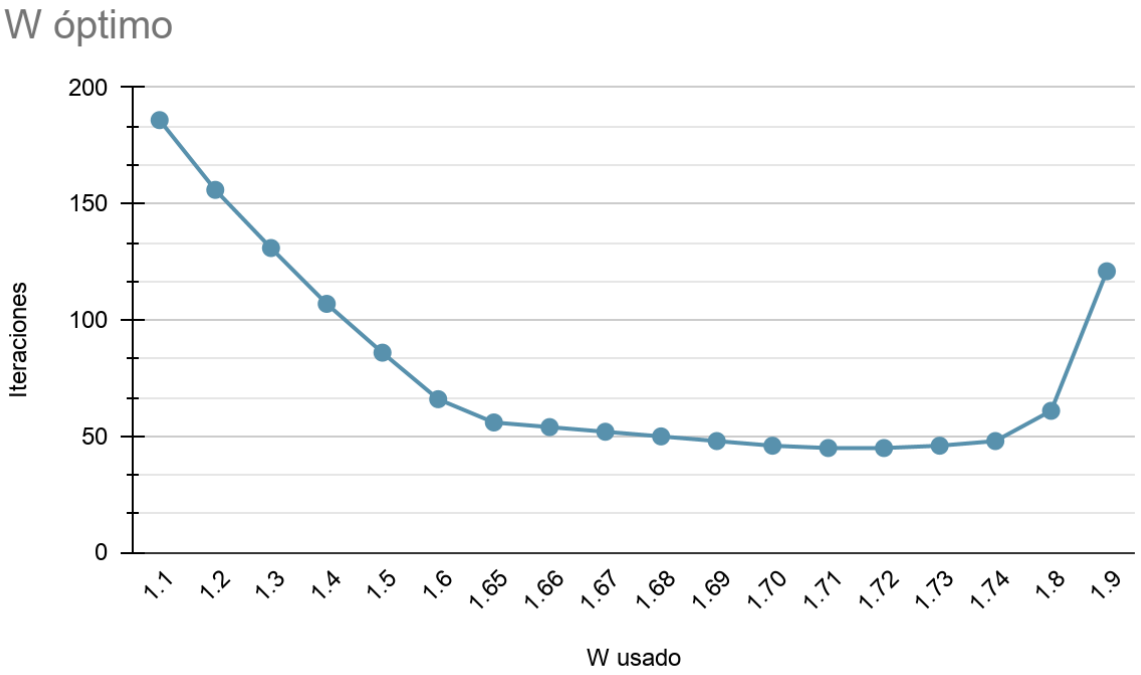
w usado	Iteraciones
1.1	186
1.2	156
1.3	131
1.4	107
1.5	86
1.6	66

1.7	46
1.8	61
1.9	121

De acuerdo al cuadro inferimos que el w óptimo se encuentra entre 1.6 y 1.8, seguimos refinando y llegamos a lo siguiente:

w usado	Iteraciones
1.65	56
1.66	54
1.67	52
1.68	50
1.69	48
1.71	45
1.72	45
1.73	46
1.74	48

Con estos resultados determinamos que el w óptimo para la matriz, con las condiciones impuestas, debe estar entre 1.71 y 1.72. Elegimos quedarnos con 1.72 porque el tiempo de ejecución es menor aunque por un margen mínimo con respecto al w = 1.71.



c. Con una tolerancia de 0.00001 y un w = 1.72, el algoritmo tardó 16 minutos y realizó 87 iteraciones. Para este punto trabajamos con la matriz mediana (de dimensión 3600) por lo aclarado anteriormente.

Primeros 20 valores hallados de X

1.210300000000468344e+03
1.175882038716221814e+03
1.142985899945547317e+03
1.112931158906246765e+03
1.086691910756760535e+03
1.065066625200939370e+03
1.048538560921879707e+03
1.037452370010165396e+03
1.032003393155624281e+03
1.032274239591654805e+03
1.038183101977807382e+03

1.049649323424188424e+03
1.066583133572511997e+03
1.088679159483172043e+03
1.115593806397695744e+03
1.146789842151995344e+03
1.181668229667278865e+03
1.219477554634769831e+03
1.259277278841904945e+03
1.300000000000502951e+03

Últimos 20 valores hallados de X:

1.210300000000468344e+03
1.171482564927738622e+03
1.134597872927739672e+03
1.101206904595487686e+03
1.072357515643787792e+03
1.048802103987826740e+03
1.030919442192630640e+03
1.018936019037066217e+03
1.012945674636038802e+03
1.012962071976164225e+03
1.018875138793999440e+03
1.030614593116080414e+03
1.048136736339115487e+03
1.071214181266431069e+03
1.099596895156024402e+03
1.132839028272590213e+03

1.170400014923995968e+03
1.211520259417216948e+03
1.255150755933893151e+03
1.300000000000502951e+03

d. CUADRO:

Cálculos hechos con 0.00001 de tolerancia y w = 1.72

Valores	Tiempo de procesamiento
ni = 90 nj = 10	00:00:31
ni = 180 nj = 20	00:16:11
ni = 360 nj = 50	

e.

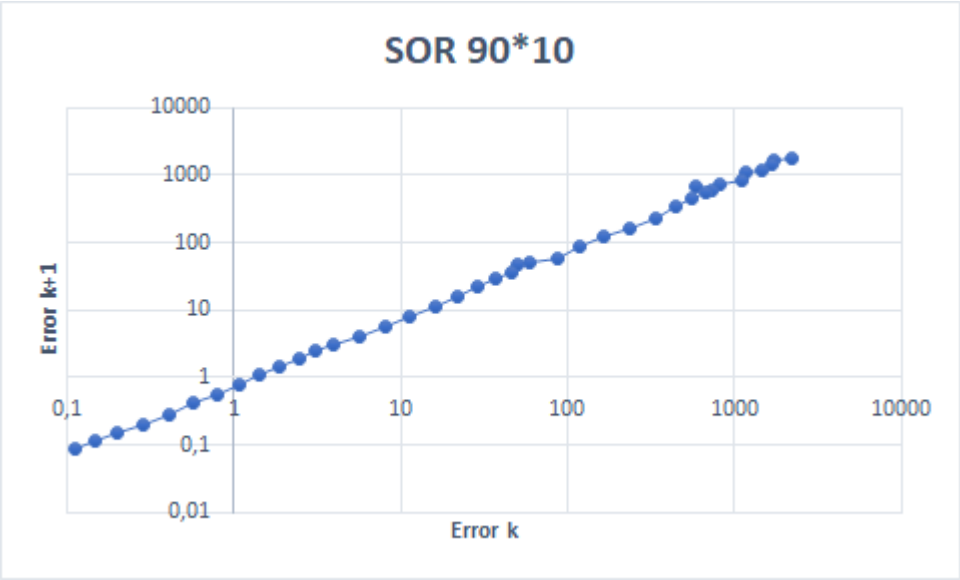
g. CUADRO CON GAUSS SEIDEL:

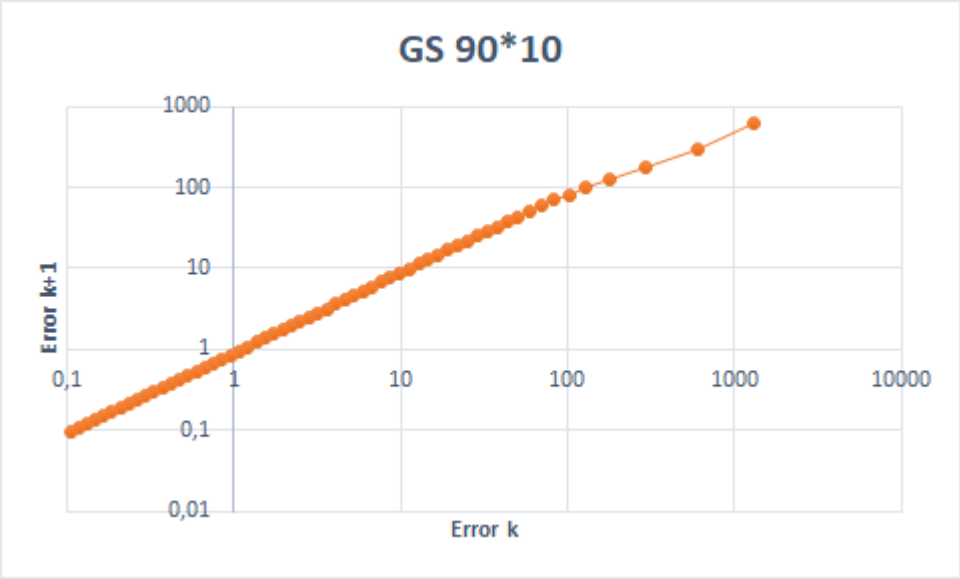
La tolerancia utilizada, al igual que para completar el cuadro de SOR, fue de 0.00001.

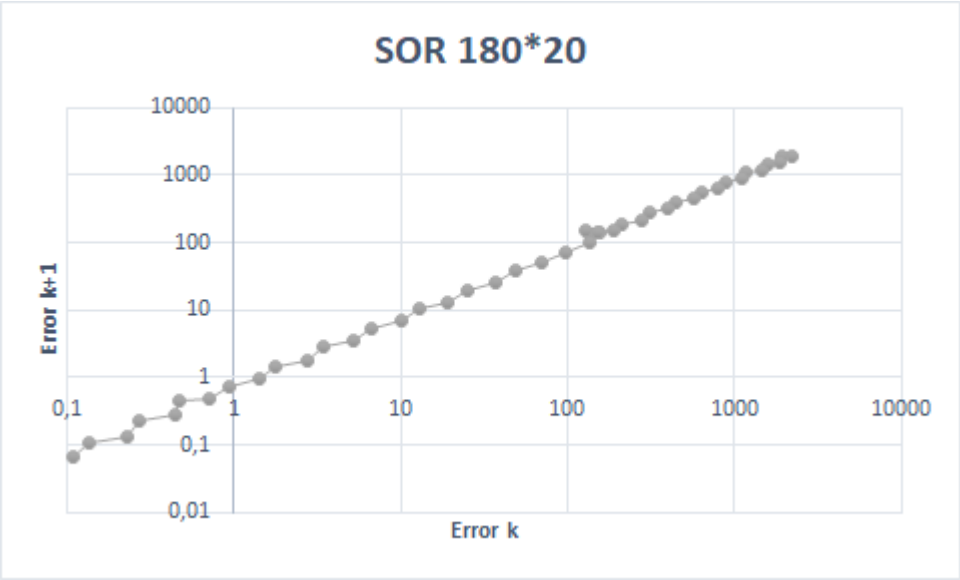
Valores	Tiempo de procesamiento
$ni = 90\ nj = 10$	00:01:35
$ni = 180\ nj = 20$	01:19:50
$ni = 360\ nj = 50$	

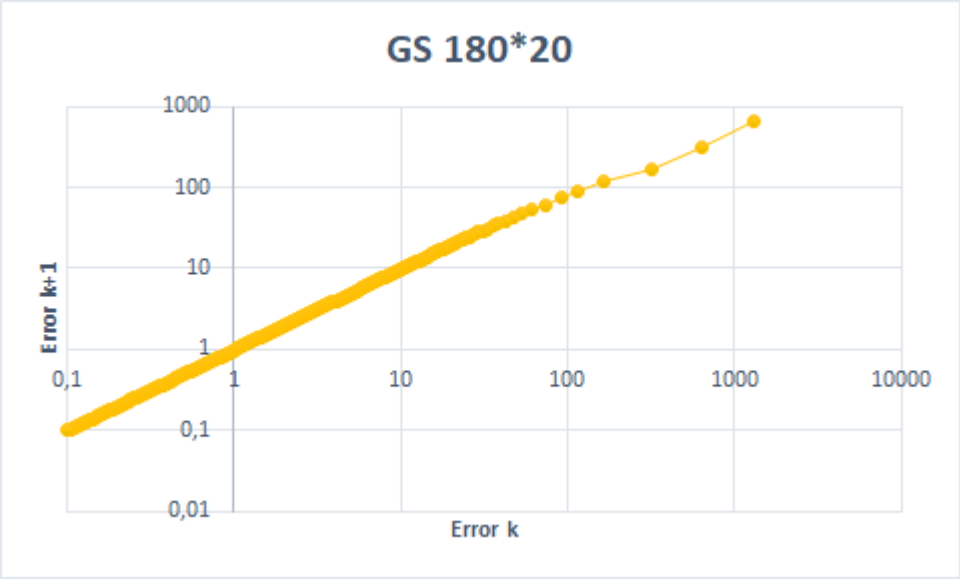
Se adjunta en el anexo 2 la función Gauss Seidel

h.









P= orden de convergencia	SOR	Gauss-Seidel
90*10	1,093	0,999
180*20	2,142	1

λ= cte asintótica del error (*)	SOR	Gauss-Seidel
90*10	0,830	0,889
180*20	7,907	0,974

(*) Por lo visto en clase podemos aproximar el valor de la constante asintótica del error con el radio

espectral de la matriz, ya que ambos se relacionan con la velocidad de convergencia.

Si bien, el valor de 7,907 en la matriz mediana de SOR se ve muy grande, podemos observar una notable diferencia entre el tiempo de procesamiento de esta matriz entre SOR y Gauss-Seidel (para el primero 16 minutos y para el segundo 80). Eso mismo verifica la diferencia entre sus ordenes de convergencia.

Cómo es esperable teóricamente, SOR tiene un mayor orden de convergencia que Gauss-Seidel.

Con respecto de los gráficos, podemos mencionar que se observa la linealidad entre los errores, lo cual me indica que converge, y su pendiente me indica el orden de convergencia

i.

Para este tipo de problemas donde el sistema de ecuaciones tiene una matriz A que no es densa si no que por el contrario tiene muchos coeficientes nulos, es decir las llamadas matrices ralas, lo más práctico a la hora de trabajar sería utilizar métodos iterativos. La razón de esto es que los métodos de esta clase tienen en cuenta la existencia de este tipo de matrices, y pueden reducir el número de operaciones aritméticas y evitar los problemas que surgen al realizar muchas operaciones con coeficientes nulos o transformar coeficientes nulos en otros no nulos, incorporando errores antes inexistentes.

Conclusiones

Los resultados obtenidos en este trabajo corresponden a lo previsto por la teoría. El método de SOR efectivamente converge significativamente más rápido que Gauss-Seidel, y aunque el w no es fácil

de obtener de forma analítica, para este problema en particular resulta sencillo de conseguir de forma experimental, y además vale la pena hacerlo por todo lo que se gana en el aspecto computacional. Como observación se podría recalcar que el error de truncamiento se hace presente en ambos métodos y es el que mayor relevancia tiene, porque el método teóricamente se extiende hasta el infinito hasta alcanzar la solución exacta, pero obviamente esto es imposible de hacer en el mundo real por lo que debo definir una condición de corte. Por lo tanto el proceso se trunca cuando los números obtenidos alcanzan la precisión deseada, en este caso esta precisión viene dada por el valor de la tolerancia. Además a esto se suma los errores inherentes al sistema como pueden ser los errores de redondeo o la limitación del hardware utilizado.

Anexo 1

```
import numpy as np

import csv

import time

import pandas as pd



df = pd.read_csv('A_090_010.csv', header=None, index_col=None,
delimiter=',', dtype=float)

A = df.values



df2 = pd.read_csv('b_090_010.csv', header=None, index_col=None,
delimiter=',', dtype=float)

b = df2.values



tol = 0.01
```

```
n = len(b)

def solver_SOR(A, b):

      w = 1.72
      error = 2  * tol
      x = np.ones(n, dtype = float)
      diferencia = np.zeros(n, dtype = float)
      iteraciones = 0

      while not error <= tol:
            for i in range(0, n):
                  suma = 0.0
                  suma2 = 0.0
                  for j in range(0, i):
                        if j != i and A[i][j] != 0:
                              suma = suma + A[i][j] * x[j]
                  for j in range(i, n):
                        if j != i and A[i][j] != 0:
                              suma2 = suma2 + A[i][j] * x[j]
                  R = b[i] - suma - suma2
                  nuevo = x[i] * (1 - w) + (w / A[i][i]) * R
                  diferencia[i] = np.abs(nuevo - x[i])
                  x[i] = nuevo

            error = np.max(diferencia)
            iteraciones = iteraciones + 1 #cuento iteraciones
```

Anexo 2

```
import numpy as np

import pandas as pd

import csv

import time


tolerancia = 0.1


df = pd.read_csv('A_360_050.csv', header=None, index_col=None,
delimiter=',', dtype=float)

A = df.values


df2 = pd.read_csv('b_360_050.csv', header=None, index_col=None,
delimiter=',', dtype=float)

b = df2.values


n = len(A)


def gauss_seidel(A, b):

      x = np.zeros(n, dtype = float)

      diferencia = np.ones(n, dtype = float)

      errado = 2  * tolerancia

      iteracion = 0

      i = 0

      while not (errado <= tolerancia):

            for i in range(0, n):

                  suma = 0.0

                  for j in range(0, n):

                        if j != i and A[i][j] != 0:
```

```
                suma = suma + A[i][j] * x[j]
        nuevo = (b[i] - suma) / A[i][i]
        diferencia[i] = np.abs(nuevo - x[i])
        x[i] = nuevo
errado = np.max(diferencia)
iteracion += 1
```