FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1 — Conducción de calor 2D en un anillo

[75.12] Análisis Numérico
Segundo cuatrimestre 2022

Padron	Alumno
105226	Franco Gentile
100589	Verónica Beatriz Leguizamón

Índice

1. Introducción

El presente informe reúne la documentación de la solución del primer trabajo práctico de la materia Análisis Numérico que consiste en estudiar la distribución de temperatura generada por la soldadura durante el proceso de fabricación. Para la resolución de este problema, hicimos uso de un modelo simplificado, en donde dispondremos de información para tres casos en los que varían las coordenadas polares r y tita, estos estarán representados por ni y nj respectivamente.

El eje de este trabajo practico será la resolución de los distintos sistemas a partir los métodos estudiados en la materia.

El objetivo será analizar los resultados obtenidos, para poder decidir cual es la manera de resolver nuestro modelo de la manera más eficiente y analizar cuales son sus ventajas y desventajas con respecto a los otros métodos.

2. Generalidades

2.1. Obtención del valor Thot.

Para la obtención del valor Thot, usamos como dato el padrón 100589.

Como $Thot = \frac{Padrn}{100} + 300$ $Thot = 1305,89$

3. Desarrollo

Realizamos varias versiones del código, primero iterando las filas y las columnas de las matrices y luego iterando solamente las filas y realizando un producto escalar con el vector x que guarda tanto los valores de $k+1$ iteración como también k iteración tal y como lo expresa el método utilizado. Al cabo de varias pruebas notamos que la velocidad de ejecución del código no era óptima y se podía realizar el método SOR y Gauss Seidel de forma personalizada para esta matriz en particular.

Analizando los datos pudimos observar que posee muchos ceros, que es diagonal dominante y que, el estar realizando muchos cálculos con todos esos ceros ralentizaba nuestro programa. Notamos que los coeficientes de la diagonal seguían un patrón que se repetía según los nodos de la coordenada radial nj, comenzaban y terminaban con un uno. A su vez, estos estaban acompañados por dos valores más a su izquierda y a su derecha en la posición consiguiente, todos excepto los que tenían un uno en su diagonal que poseían toda la fila con ceros. Los valores más distantes hicieron que separemos el comportamiento del programa en tres, (1) las nj primeras filas, (2) las nj últimas filas y (3) las que estaban en el medio.

(3) Tiene valores distintos a ceros nj posiciones a su izquierda, como a su derecha.
(1) Tiene valores nj posiciones hacia su derecha y, también, la franja que en (3) está a la izquierda, pero esta vez a la derecha como si esta primera fuese su continuación.
(2) Ocurre lo mismo que en el punto anterior, solamente que del lado derecho.

Este analisis nos llevó a realizar un método SOR y Gauss Seidel personalizado para estas matrices que, las tres, cumplen el mismo patrón. De esta forma pudimos resolver los puntos con más rapidéz.

3.1. Implementación de la función del método SOR.

Inicialmente, resolvimos el sistema haciendo uso del método SOR o de sobrerrelajaciones sucesivas, este es un método iterativo, para el cual tuvimos que fijar una cantidad de iteraciones máximas, un w y un error máximo, de modo que si nuestra solución cumplía con los requisitos establecidos, habríamos dado con la solución buscada. Llevándonos de la ecuación siguiente, pensamos un código que resuelva el problema para la matriz específica que tenemos y que, como explicamos anteriormente, cumplen un patrón.

$$x_i^{(n+1)} = (1-w)x^n + w\frac{b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{n+1} - \sum_{j=i+1}^{n} a_{ij}x_j^{(n)}}{a_{ii}}$$

Se verificó que los valores a la derecha de la diagonal se multiplicaran por las soluciones ya obtenidas en los pasos siguientes para cumplir con lo que el método pide, mientras que los de la izquierda sean por los valores anteriores de x. Esto para acelerar la convergencia del método. Luego, se lo pondera con w.
El código se encuentra en el Anexo I.

3.2. Obtención del w_{ptimo} de forma experimental.

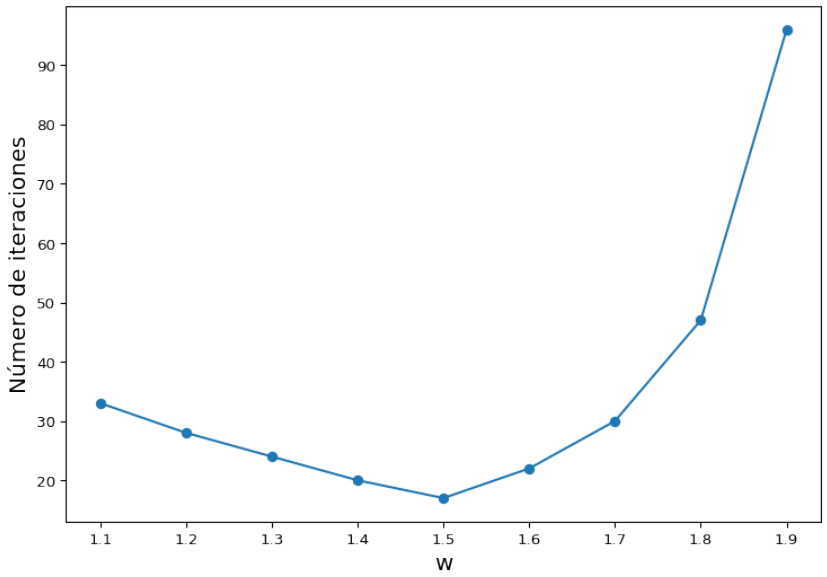
Anteriormente, para la implementación del método SOR, nosotros podíamos calcular la solución para cualquier valor del coeficiente w síendo este un factor de ponderación que nos indicara la velocidad de convergencia del método, de manera que existirá un valor que hace máxima la velocidad de convergencia.

La siguiente explicación se basa en la hallar el valor w que hace máxima a la velocidad de convergencia.

Con el método SOR implementado, fuimos variando el valor de w desde 1.1 a 1.9, sabiendo que en el método SOR los valores de w deben cumplir que $1 < w < 2$, con un paso de 0.1 y calculamos el número de iteraciones necesitadas para llegar a un resultado con una tolerancia de 0.001.

- Para la matriz $ni = 90$ y $nj = 10$ obtuvimos los siguientes resultados.

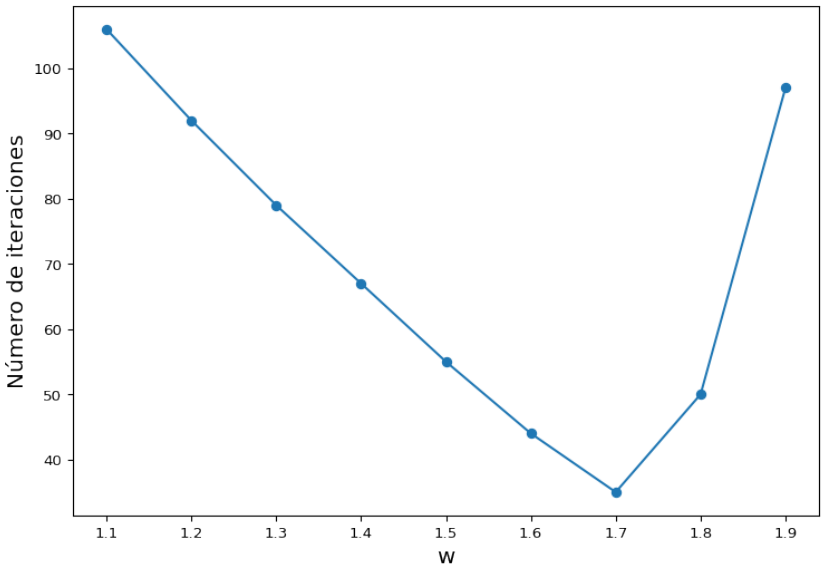
w	Número de iteraciones
1.1	33
1.2	28
1.3	24
1.4	20
1.5	17
1.6	22
1.7	30
1.8	47
1.9	96



De modo que, como se puede ver en el grafico, nuestro w_{optimo} para la matriz con ni=90 y nj=10 es 1.5, ya que para hallar la solución solo hicieron falta 17 iteraciones.

- Para la matriz $ni = 180$ y $nj = 20$ obtuvimos los siguientes resultados.

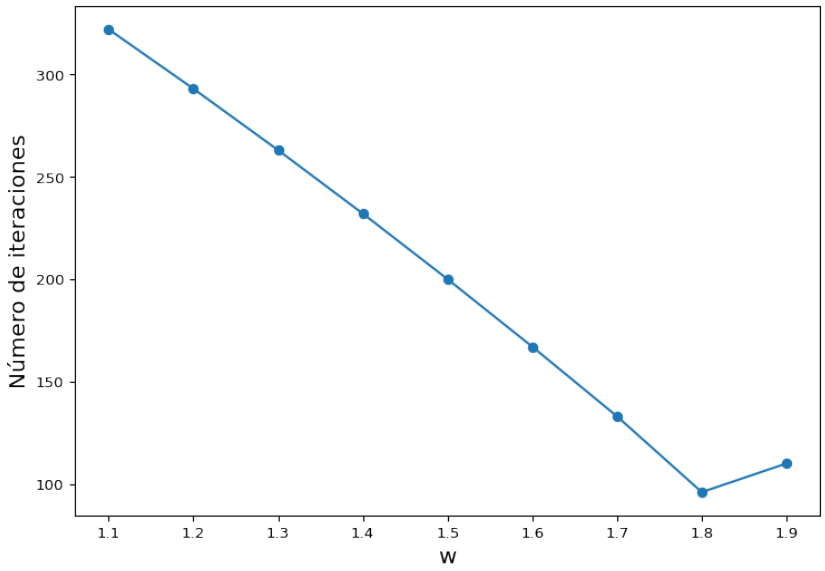
w	Número de iteraciones
1.1	106
1.2	92
1.3	79
1.4	67
1.5	55
1.6	44
1.7	35
1.8	50
1.9	97



De que, como se puede ver en el grafico, nuestro w_{optimo} para la matriz con ni=180 y nj=20 es 1.7, ya que para hallar la solución solo hicieron falta 35 iteraciones.

- Para la matriz $ni = 360$ y $nj = 50$ obtuvimos los siguientes resultados.

w	Número de iteraciones
1.1	322
1.2	293
1.3	263
1.4	232
1.5	200
1.6	167
1.7	133
1.8	96
1.9	110



De modo que, como se puede ver en el gráfico , nuestro w_{optimo} para la matriz con ni=360 y nj=50 es 1.8, ya que para hallar la solución solo hicieron falta 96 iteraciones.

3.3. Resolución del sistema para la matriz con ni=360 y nj=50

Aquí nos centramos en resolver el mencionado sistema, utilizando el resultado obtenido anteriormente, el w_{optimo}, y forzando al algoritmo a que nos brinde un resultado de manera tal que el error relativo sea menor a 0.00001. Para realizarlo, hicieron falta 216 iteraciones, a continuación mostraremos una tabla con los primeros y últimos 10 resultados obtenidos, solo los cinco primeros valores. Al costado derecho se encuentra su error.

```
0 [0. 0. 0. 0. 0.]
1 [2178.54  1781.591 1457.653 1193.174  977.141] 1.0
2 [435.708 452.352 478.974 506.496 529.625] 0.9957571747171038
3 [1829.974 1589.479 1386.327 1210.324 1055.457] 0.9956413780171809
4 [714.561 721.853 721.994 713.721 697.847] 0.9918566265906468
5 [1606.891 1434.175 1277.582 1135.979 1008.809] 0.9891282921164795
6 [893.027 871.978 845.441 815.429 783.314] 0.9836251719234306
7 [1464.118 1327.965 1204.406 1092.858  992.338] 0.9580015642585985
8 [1007.246  970.637  932.553  893.429  853.456] 0.9212024822523264
9 [1372.743 1264.178 1164.457 1072.686  988.121] 0.8445665362028589
10 [1080.346 1035.94   990.731  945.078  899.341] 0.8024167264601204
206 [1210.298 1197.836 1185.487 1173.344 1161.452] 1.2307711242691447e-05
207 [1210.302 1197.841 1185.493 1173.35  1161.459] 1.2307673372868964e-05
208 [1210.298 1197.839 1185.493 1173.353 1161.464] 1.2307711242691447e-05
209 [1210.302 1197.844 1185.499 1173.359 1161.47 ] 1.1538443787086515e-05
210 [1210.298 1197.842 1185.499 1173.361 1161.474] 1.0769247337311292e-05
211 [1210.302 1197.847 1185.504 1173.366 1161.48 ] 1.0769214201216617e-05
212 [1210.298 1197.844 1185.503 1173.367 1161.481] 1.0769247337398743e-05
213 [1210.302 1197.849 1185.508 1173.372 1161.488] 1.0769214201304068e-05
214 [1210.298 1197.846 1185.507 1173.374 1161.491] 1.000001538475239e-05
215 [1210.302 1197.851 1185.513 1173.379 1161.496] 1.0769214201129166e-05
216 [1210.298 1197.849 1185.511 1173.379 1161.496] 8.461551479459684e-06
```

3.4. Tiempos de procesamiento método SOR

Los tiempos de procesamiento presentados son producto de usar una tolerancia de 0.00001.

SOR		
Matriz	w	Tiempo de procesamiento
ni=90 nj=10	1.5	0.29 s
ni=180 nj=20	1.7	1.67 s
ni=360 nj=50	1.8	13.58 s

3.5. Implementación de la función del método Gauss-Seidel

En el inicio de este documento describimos algunas características de la utilización del método SOR, cabe señalar que este es un método eficiente, pero no es el único que existe, por lo que en este inciso realizaremos nuevamente los cálculos utilizando el método de Gauss-Seidel.

$$x_i^{(n+1)} = \frac{b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{n+1} - \sum_{j=i+1}^{n} a_{ij} x_j^{(n)}}{a_{ii}}$$

Se verificó que los valores a la derecha de la diagonal se multiplicaran por las soluciones ya obtenidas en los pasos siguientes para cumplir con lo que el método pide, mientras que los de la izquierda sean por los valores anteriores de x. La única diferencia con el otro método estudiado, SOR, es que no se pondera con w. Este método, cabe destacar, que es un caso especial del método SOR.
El código se encuentra en el Anexo II.

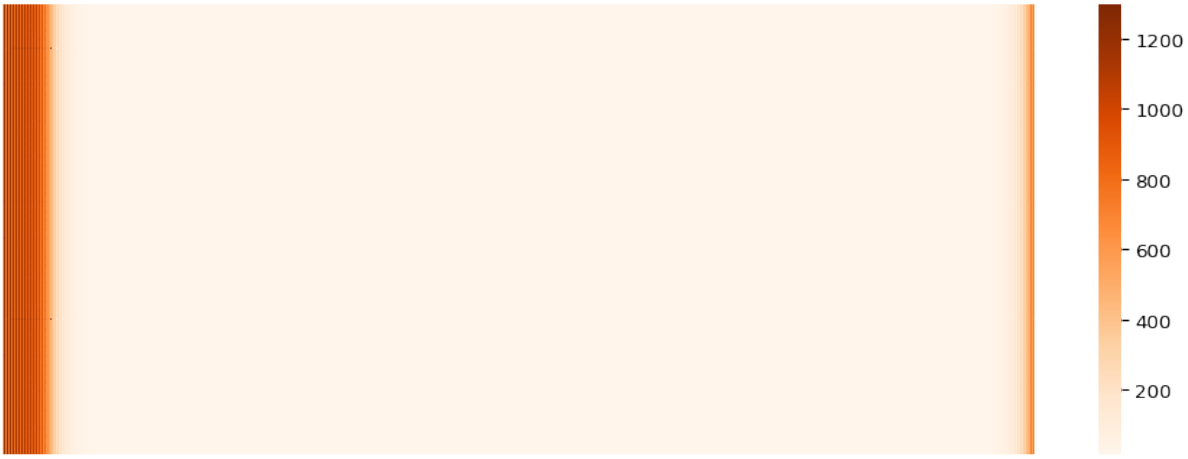
3.6. Tiempos de procesamiento método G-S

Los tiempos de procesamiento presentados son producto de usar una tolerancia de 0.00001.

Gauss-Seidel	
Matriz	Tiempo de procesamiento
ni=90 nj=10	1.578 s
ni=180 nj=20	22.82 s
ni=360 nj=50	170.6 s

3.7. Gráfico de temperatura

En este inciso, nos centramos en hacer un gráfico para ver como se distribuía el calor en la mitad del anillo.



En los extremos se concentra muchísimo más el calor que en la parte media del mismo.

3.8. Velocidad de convergencia.

Para estudiar la velocidad de convergencia se nos propuso calcular el orden de convergencia y el radio espectral de la matriz T de cada una de las matrices y de cada método.

$$T_{SOR} = (D - w * L))^{-1} * (1 - w) * D + w * U$$
$$T_{GS} = (D - L)^{-1} + U$$

Radio espectral/Velocidad de convergencia		
Matriz	$\rho(T_{GS})$	$\rho(T_{SOR})$
ni=90 nj=10	0.89	0.65
ni=180 nj=20	0.97	0.84

Comenzamos a analizar y armar la matriz T, pudimos realizarlo con dos de las matrices propuestas en el presente trabajo práctico, pero no para la última por las limitaciones computacionales al trabajar con una matriz con muchos datos. Ya que poseíamos las matrices decidimos también hacer un análisis de convergencia con las mismas.

Para la matriz más pequeña de ni = 90 y nj = 180 obtuvimos los siguientes resultados.

Gauss-Seidel		
Matriz	$\|T_{GS}\|_\infty$	$\|T_{GS}\|_\infty$
ni=90 nj=10	1.04	0.99

SOR		
Matriz	$\|T_{SOR}\|_\infty$	$\|T_{SOR}\|_\infty$
ni=90 nj=10	2.23	1.79

Las normas, al ser mayores a uno no nos aseguraban la convergencia del método SOR, pero sí de Gauss Seidel. Por otra parte, el radio espectral de las matrices de ambos métodos sí nos aseguraban que convergían y, como se puede apreciar por el radio espectral, SOR converge más rápido que Gauss Seidel.

Analizando la matriz T para ni = 180 y nj = 90:

Gauss-Seidel		
Matriz	$\|T_{GS}\|_1$	$\|T_{GS}\|_\infty$
ni=180 nj=20	1.04	1.00

SOR		
Matriz	$\|T_{SOR}\|_1$	$\|T_{SOR}\|_\infty$
ni=180 nj=20	5.63	2.49

Esta vez, ninguna norma de la matriz T nos aseguró la convergencia del método pero, al calcular el radio espectral obtuvimos un resultado que nos aseguró su convergencia y, por el valor, que Gauss Seidel converge más rápido. Por otro lado, procedimos a calcular los órdenes de convergencia de forma práctica de cada método con la siguiente fórmula:

$$p = \frac{ln(\Delta x^{k+1}/\Delta x^k)}{ln(\Delta x^k/\Delta x^{k-1})}$$

Obtuvimos los siguientes resultados:

SOR	
Matriz	Orden de convergencia
ni=90 nj=10	0.95248
ni=180 nj=20	0.99991
ni=360 nj=50	0.99983

Gauss-Seidel	
Matriz	Orden de convergencia
ni=90 nj=10	0.99930
ni=180 nj=20	0.99947
ni=360 nj=50	0.99980

3.9. Grafico del error k+1 en función del error k.

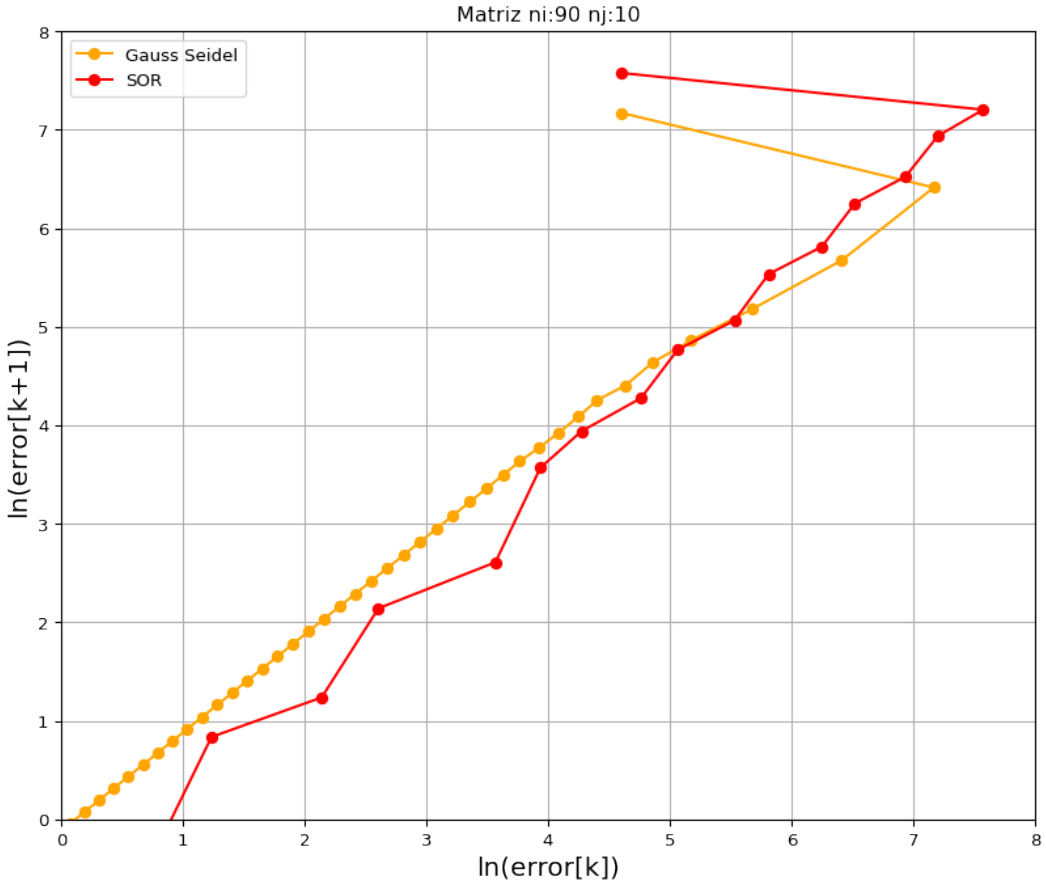


Figura 1: Caption

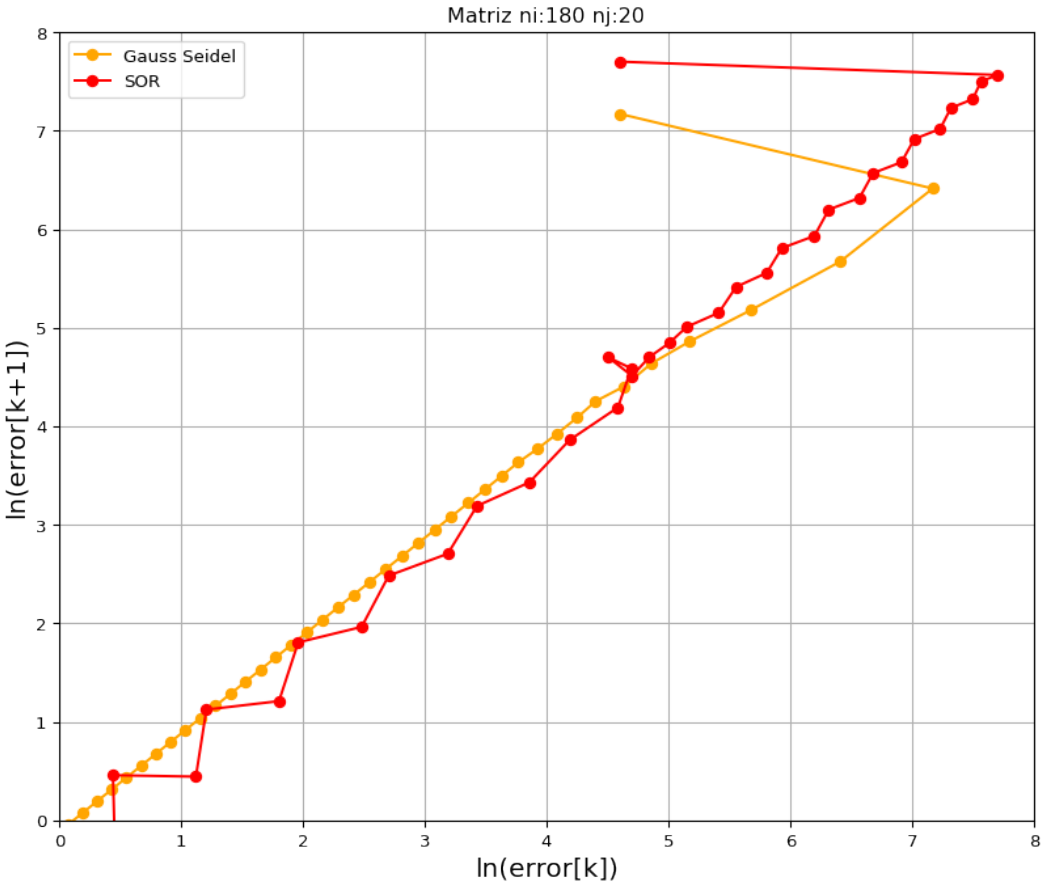


Figura 2: Caption

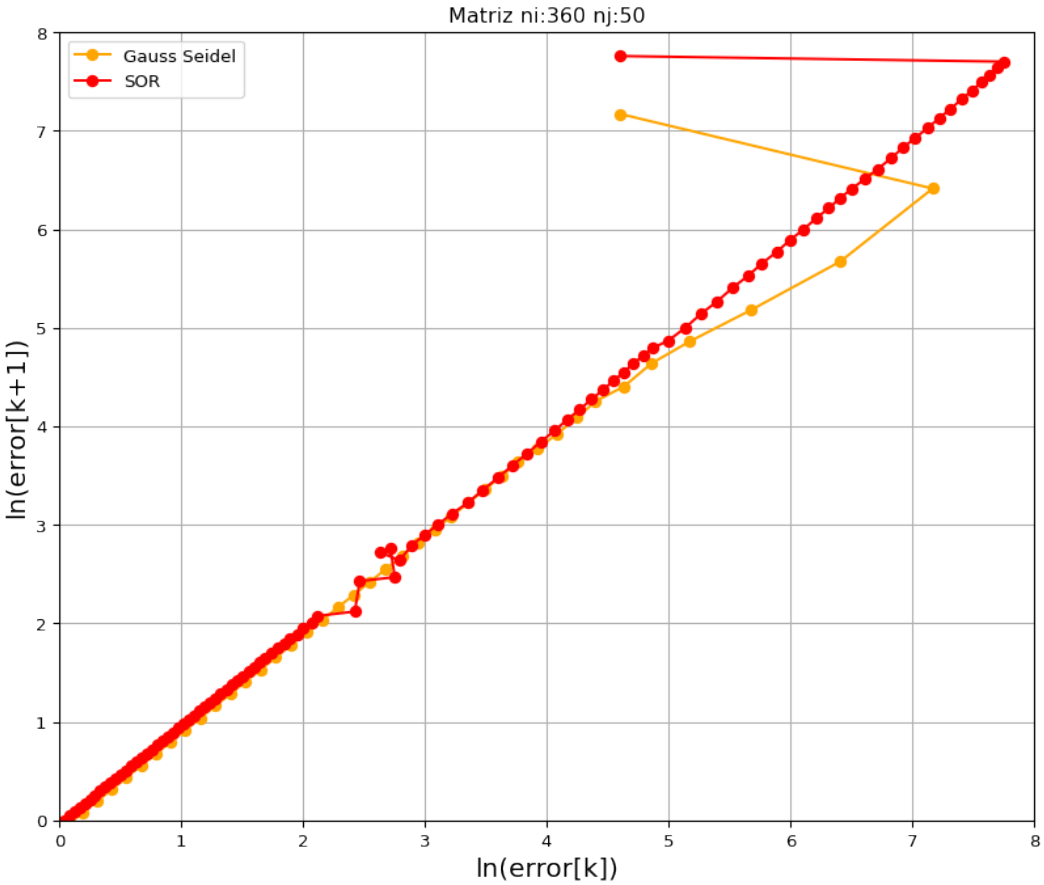


Figura 3: Caption

Como se puede observar en los ítems anteriores, el método de SOR converge más rápido de forma práctica que Gauss Seidel, resolviendo el mismo problema con un número menor de iteraciones. El orden de convergencia para la mayor parte de las matrices resultan mayor para el método SOR y, como se puede observar en los gráficos de error, disminuye el mismo mucho más rápido.

3.10. Métodos numéricos.

Para resolver estas matrices que tienen una gran cantidad de datos donde la mayoría de ellos son ceros, es mucho más factible utilizar un método iterativo. Si se quisiera realizar por medio de uno directo, induciríamos mucho error de redondeo al hacer que valores que ya son ceros, tengan otro diferente. Además de que resulta poco práctico programar el método con sus respectivos pivoteos, con un método iterativo muchos de estos ceros no se toman en cuenta y, de esta forma, el cálculo resulta más sencillo y rápido de ejecutar.

4. Conclusiones

Tiempo de procesamiento		
Matriz	Método SOR	Método G-S
ni=90 nj=10	0.29 s	1.578 s
ni=180 nj=20	1.67 s	22.82 s
ni=360 nj=50	13.58 s	170.6 s

Cuadro 1: Comparando tiempos de procesamiento.

Orden de convergencía		
Matriz	Método SOR	Método G-S
ni=90 nj=10	0.95248	0.99930
ni=180 nj=20	0.99991	0.99947
ni=360 nj=50	0.99983	0.99980

Cuadro 2: Comparando ordenes de convergencia.

Para comenzar analizando los resultados obtenidos, lo que más nos llamo la atención fue que al calcular el orden de convergencia para la matriz más pequeña resulto ser que este valor era menor para el método SOR que para el método G-S, esperábamos que este resultado sea al revés, tal como paso en los cálculos realizados en las otras dos matrices.
Como sabemos que SOR converge más rápido que Gauss-Seidel siempre que w>1, creemos que este resultado es debido a errores de truncamiento, efectos de cancelación de términos, o bien, porque a pesar de que el valor utilizado disminuya el número de iteraciones y converja a una solución más rápido que Gauss Seidel, no es el más óptimo. Por esto consideramos de vital importancia la elección del w.
Aun así, en el cuadro donde comparamos los tiempos de procesamiento, se puede apreciar como estos se reducen notablemente al utilizar el método SOR, esto era de esperarse ya que este es más eficiente.

Con respecto a la convergencía:
Con los resultados obtenidos pudimos comprobar que estas convergían de forma práctica y teórica. De forma práctica porque gráficamente pudimos apreciar que este disminuía con el paso de las iteraciones hasta llegar a la tolerancia pedida y el valor obtenido fue coherente para la solución del problema planteado. De forma teórica, pudimos evaluar para dos de las matrices sus normas y su radio espectral que nos aseguró la convergencia de ambas. Pero, si no hubiésemos podido calcular esto último, igualmente podíamos asegurar que convergía de forma práctica por los resultados obtenidos.

Errores involucrados:
A diferencia de los métodos directos que poseen error de redondeo, los métodos iterativos solamente presentan error de truncamiento al estar realizando cálculos con decimales infinitos en computadoras con espacio finito y de redondeo a la hora de presentar el último resultado. También poseemos errores de cancelación de términos al calcular soluciones intermedias muy parecidas entre sí a la hora de calcular el error en cada una de las iteraciones.

Para finalizar, además, observamos que la semilla utilizada difería totalmente del resultado final. Por ese motivo se apreciaron errores grandes en todos los gráficos en las primeras iteraciones. Esto se debe a que no teníamos una aproximación inicial sobre los valores que x debería tener, sino que, haciendo la prueba suponiendo que las matrices debían converger, por lo tanto cualquier valor que usemos de semilla funcionaría. Si hubiésemos utilizado un método de arranque para te-

ner una idea de por qué valores estaría x seguramente los métodos serían más rápidos, con menos iteraciones y tal vez mejoraba la performance de nuestros códigos.

5. Anexo I: código implementado SOR

```
def calculadorSOR(w,radio,semilla,diagonal, franja_cerca_derecha, franja_cerca_izquierda,
                  franja_lejos_derecha,franja_lejos_izquierda,tamanio):
  fila = 0
  sum = 0
  x = semilla
  sum_total = 0

  while(fila < tamanio):
    i=0
    while (i<radio and fila < tamanio):
      sum=0
      if(i==0): sum_total = b[fila]
      if(i==radio-1): sum_total = b[fila]

      # los coeficientes de al lado de la diagional
      if(i>0 and i<radio-1):
        sum = -((franja_cerca_derecha[i-1] * x[fila+1]) +
                (franja_cerca_izquierda[i-1] * x[fila-1]))

        #el coeficiente más lejos a la derecha de la diagonal
        if(fila < radio): sum = sum - ((franja_lejos_derecha[i-1] * x[fila+radio]) +
                                       (franja_lejos_derecha[i-1] * x[tamanio-radio+i]))

        # coeficientes más lejos de la diagonal.
        if(fila > radio and fila < tamanio - radio):
          sum = sum - (franja_lejos_izquierda[i-1]*x[fila-radio]
                       + franja_lejos_derecha[i-1] * x[fila+radio])

        if(fila > tamanio - radio and fila<tamanio-1):
          sum = sum - (franja_lejos_izquierda[i-1]*x[fila-radio] +
                       franja_lejos_izquierda[i-1]*x[i])
        sum_total = sum/diagonal[i-1]

      total = (sum_total * w) + ((1-w)*x[fila])
      x[fila] = total

      fila +=1
      i +=1
  return x.copy()

  def solver_SOR(A,b):

  x_anterior = np.zeros(len(A))
  x_siguiente = np.zeros(len(A))

  r = 50 # nj de cada matriz

  # armado de listas.
  franja_lejos_derecha = []
  for i in range(1,r-1):
    franja_lejos_derecha.append(A[i,i+r])
```

```
  franja_lejos_izquierda = []
  for i in range(1,r-1):
    franja_lejos_izquierda.append(A[r+i,i])

  diagonal = []
  for i in range(1,r-1):
    diagonal.append(A[i,i])

  franja_cerca_derecha = []
  for i in range(1,r-1):
    franja_cerca_derecha.append(A[i,i+1])

  franja_cerca_izquierda = []
  for i in range(1,r-1):
    franja_cerca_izquierda.append(A[i,i-1])

  tamanio = len(A)
  iteraciones = 0
  error = 1000
  tolerancia = 0.00001
  w = 1.8

  while(iteraciones < 500 and error > tolerancia):

    calculadorSOR(w,r,x_anterior.copy(), diagonal.copy(), franja_cerca_derecha.copy(),
                              franja_cerca_izquierda.copy(), franja_lejos_derecha.copy(),
                              franja_lejos_izquierda.copy(),tamanio)

    error=(np.max(abs(x_siguiente-x_anterior)) / np.max(abs(x_siguiente)))

    iteraciones += 1
    x_anterior = x_siguiente.copy()
  return x_siguiente.copy()
```

calculador es quien se encarga de realizar el despeje de la ecuación según el análisis presentado de la matriz. Guarda en la lista x los valores que ya han sido calculados (los que se encuentran en el lado derecho de la diagonal) como también los que todavía no (en el costado izquierdo de la diagonal) con la finalidad de poder realizar el método de Gauss Seidel con los valores ya calculados.

Este método sigue siendo poco óptimo al tener que cargar la matriz A en una variable. La solución más efectiva sería leer línea por línea desde el excel y obtener los datos para colocarlos en las respectivas listas. En nuestro caso no fue implementado al no poseer suficiente tiempo para hacerlo, pero es la mejor solución que pensamos para el problema.

6. Anexo II: código implementado GS

```
def calculadorGS(radio,semilla,diagonal, franja_cerca_derecha, franja_cerca_izquierda,
                  franja_lejos_derecha,franja_lejos_izquierda,tamanio):
  fila = 0
  sum = 0
  x = semilla
  sum_total = 0
```

```
  while(fila < tamanio):
    i=0
    while (i<radio and fila < tamanio):
      sum=0
      if(i==0): sum_total = b[fila]
      if(i==radio-1): sum_total = b[fila]

      # los coeficientes de al lado de la diagional
      if(i>0 and i<radio-1):
        sum = -((franja_cerca_derecha[i-1] * x[fila+1]) +
                (franja_cerca_izquierda[i-1] * x[fila-1]))

        #el coeficiente más lejos a la derecha de la diagonal
        if(fila < radio): sum = sum - ((franja_lejos_derecha[i-1] * x[fila+radio]) +
                                       (franja_lejos_derecha[i-1] * x[tamanio-radio+i]))

        # coeficientes más lejos de la diagonal.
        if(fila > radio and fila < tamanio - radio):
          sum = sum - (franja_lejos_izquierda[i-1]*x[fila-radio]
                       + franja_lejos_derecha[i-1] * x[fila+radio])

        if(fila > tamanio - radio and fila<tamanio-1):
          sum = sum - (franja_lejos_izquierda[i-1]*x[fila-radio] +
                       franja_lejos_izquierda[i-1]*x[i])
        sum_total = sum/diagonal[i-1]

      x[fila] = sum_total

      fila +=1
      i +=1
  return x.copy()


def solver_GS(A,b):

  x_anterior = np.zeros(len(A))
  x_siguiente = np.zeros(len(A))

  r = 50 # nj de las matrices

  # armado de listas.
  franja_lejos_derecha = []
  for i in range(1,r-1):
    franja_lejos_derecha.append(A[i,i+r])

  franja_lejos_izquierda = []
  for i in range(1,r-1):
    franja_lejos_izquierda.append(A[r+i,i])

  diagonal = []
  for i in range(1,r-1):
    diagonal.append(A[i,i])

  franja_cerca_derecha = []
```

```
for i in range(1,r-1):
  franja_cerca_derecha.append(A[i,i+1])

franja_cerca_izquierda = []
for i in range(1,r-1):
  franja_cerca_izquierda.append(A[i,i-1])

tamanio = len(A)
iteraciones = 0
error = 1000
tolerancia = 0.00001

while(iteraciones < 500 and error > tolerancia):

      x_siguiente = calculadorGS(r,x_anterior.copy(), diagonal.copy(), franja_cerca_derecha.cop
                              franja_cerca_izquierda.copy(), franja_lejos_derecha.copy(),
                              franja_lejos_izquierda.copy(),tamanio)

  error=(np.max(abs(x_siguiente-x_anterior)) / np.max(abs(x_siguiente)))

  iteraciones += 1
  x_anterior = x_siguiente.copy()
return x_siguiente.copy()
```