“ANÁLISIS NUMÉRICO I”

“MÉTODOS MATEMÁTICOS Y NUMÉRICOS”

75.12

DATOS DEL TRABAJO PRÁCTICO

1	2023	TRABAJO PRÁCTICO N°1
	AÑO	MÉTODOS ITERATIVOS Y COMPORTAMIENTO DEL MÉTODO SOR
	2°	
TP NRO	CUAT	TEMA

INTEGRANTES DEL GRUPO

	[redacted] LUCAS GABRIEL	97819
	APELLIDO Y NOMBRE	PADRÓN
GRUPO	APELLIDO Y NOMBRE	PADRÓN

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Introducción

Se propone, que una solución aproximada para la ecuación de Laplace es una matriz tal que satisface la ecuación:

$$\nabla^2 T = 0$$

Donde, esta misma, puede ser expandida de la forma:

$$4T_{ij} - T_{i-1\ j} - T_{i+1\ j} - T_{i\ j-1} - T_{ij+1} = 0 \qquad \textbf{(1)}$$

Donde los valores (i,j) corresponden a cada nodo en la grilla.

En este informe, se resuelve este sistema de ecuaciones implementando metodologías iterativas, y se programa una solución del método SOR en Python.

Objetivo

Para la resolución de este sistema de ecuaciones **(1)**, se propone crear el Método SOR iterativamente, pre-estableciendo una tolerancia para cada ejercicio, utilizando en un principio, un ω que es calculado con una fórmula teórica, y luego, verificando con un rango de valores ($1 < \omega < 2$) viendo cual es el w óptimo donde se hace la menor cantidad de iteraciones posibles para la resolución de la matriz.

Una vez resuelto los puntos pedidos, se realiza un análisis de los datos experimentados, verificando sí coinciden con los valores teóricos.

Desarrollo

Parte 1

Punto A

La matriz A (rala) que se obtiene por **(1)** está generada por:

- Los valores de la diagonal, que tienen coeficiente 4 dado que siempre se ubican en la posición (i, j)
- Los elementos que no están en la diagonal, por los coeficientes -1 y 0, donde, si el coeficiente es -1, significa que el nodo es "vecino" del nodo que se encuentra en la posición diagonal, y si es un 0, significa que no lo es.

Un ejemplo práctico es, se supone una matriz con enumeración:

$$\begin{matrix} 1 & 3 \\ 2 & 4 \end{matrix}$$

Genera el sistema de ecuación:

$$\begin{cases} 4T_1 - T_3 - T_4 = 0 \\ 4T_2 - T_1 - T_4 = 0 \\ 4T_3 - T_4 - T_1 = 0 \\ 4T_4 - T_2 - T_3 = 0 \end{cases}$$

Por lo tanto, la matriz A queda generada por :

$$A = \begin{bmatrix} 4 & 0 & -1 & -1 \\ -1 & 4 & 0 & -1 \\ -1 & 0 & 4 & -1 \\ 0 & -1 & -1 & 4 \end{bmatrix}$$

Ahora, denotar que si se cambia la enumeración de los nodos de la matriz, por ejemplo, se propone la siguiente enumeración:

$$\begin{matrix} 4 & 3 \\ 2 & 1 \end{matrix}$$

Se llega a un sistema de ecuaciones completamente diferente al obtenido con la numeración mostrada previamente::

$$\begin{cases} 4T_4 - T_3 - T_2 = 0 \\ 4T_3 - T_4 - T_1 = 0 \\ 4T_2 - T_4 - T_1 = 0 \\ 4T_1 - T_3 - T_2 = 0 \end{cases}$$

Que, se denota que la última ecuación generada es diferente a la ecuación generada en la primera enumeración, por ende genera una matriz A diferente.

Punto B

Se encuentra tanto el código de la creación de la matriz y el método de resolución de SOR en el Anexo 1.

Punto C

Se realizaron corridas, se creó una tabla de valores, donde se utilizan w fijos, y se analizan la cantidad de iteraciones necesarias para completar la tolerancia dada por el enunciado.

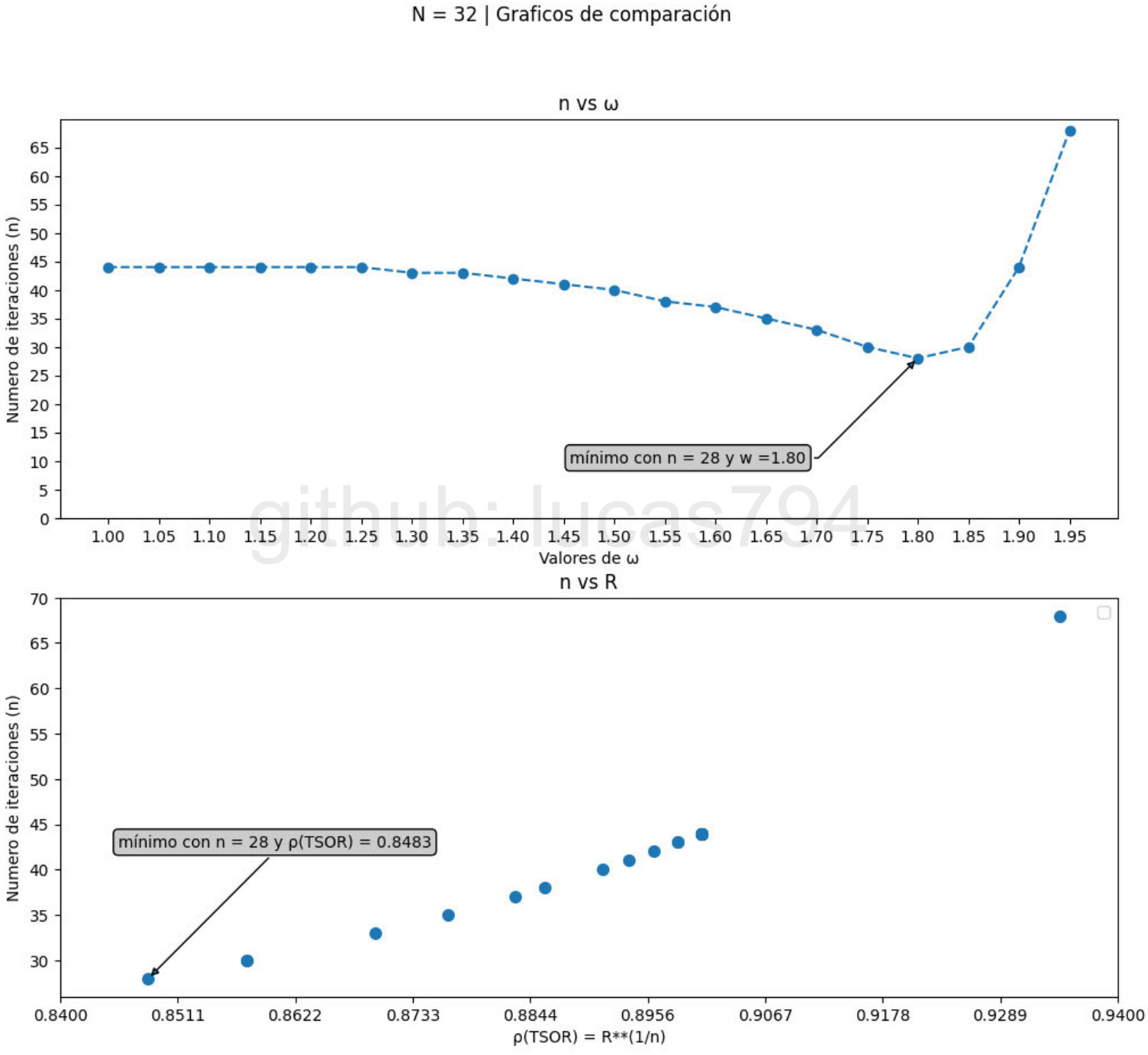
Para mayor claridad, solo se muestran las corridas relevantes (Se agrega en el anexo las corridas completas)

Valor ω	Iteraciones realizadas
1.00	8
1.05	7
1.20	6
1.30	7
1.40	8
1.45	8
1.50	9
1.70	15
1.95	46
1.95	91

Punto D

```
w = 1.7
[[0.         1.         1.         ... 1.         1.         0.        ]
 [1.         0.99580157 0.99176005 ... 0.99444905 0.99729596 1.        ]
 [1.         0.99176005 0.98382402 ... 0.98928594 0.99477616 1.        ]
 ...
 [1.         0.99444905 0.98928594 ... 0.9952806  0.99770062 1.        ]
 [1.         0.99729596 0.99477616 ... 0.99770062 0.99887964 1.        ]
 [0.         1.         1.         ... 1.         1.         0.        ]]
Se logró la convergencia luego de 30 iteraciones. | última tolerancia : 0.009810328047172393 |
w = 1.75
[[0.         1.         1.         ... 1.         1.         0.        ]
 [1.         0.99614705 0.99244854 ... 0.995055   0.99757333 1.        ]
 [1.         0.99244854 0.98519453 ... 0.9907957  0.99572506 1.        ]
 ...
 [1.         0.995055   0.9907957  ... 0.99713039 0.99861316 1.        ]
 [1.         0.99757333 0.99572506 ... 0.99861316 0.99932976 1.        ]
 [0.         1.         1.         ... 1.         1.         0.        ]]
Se logró la convergencia luego de 28 iteraciones. | última tolerancia : 0.009934669995474377 |
w = 1.7999999999999998
[[0.         1.         1.         ... 1.         1.         0.        ]
 [1.         0.99667365 0.99348705 ... 0.99625521 0.99817063 1.        ]
 [1.         0.99348705 0.98724281 ... 0.99272115 0.99644346 1.        ]
 ...
 [1.         0.99625521 0.99272115 ... 0.99891696 0.99948601 1.        ]
 [1.         0.99817063 0.99644346 ... 0.99948601 0.99975625 1.        ]
 [0.         1.         1.         ... 1.         1.         0.        ]]
Se logró la convergencia luego de 30 iteraciones. | última tolerancia : 0.009399040986168892 |
w = 1.85
[[0.         1.         1.         ... 1.         1.         0.        ]
 [1.         0.99766107 0.99541231 ... 0.99810938 0.99908194 1.        ]
 [1.         0.99541231 0.9909986  ... 1.00426141 1.00661894 1.        ]
 ...
 [1.         0.99810938 1.00426141 ... 1.00018825 1.00009711 1.        ]
 [1.         0.99908194 1.00661894 ... 1.00009711 1.0000504  1.        ]
 [0.         1.         1.         ... 1.         1.         0.        ]]
Se logró la convergencia luego de 44 iteraciones. | última tolerancia : 0.009899670352999792 |
w = 1.9
[[0.         1.         1.         ... 1.         1.         0.        ]
 [1.         1.00036894 1.00098723 ... 1.00031468 1.00003712 1.        ]
 [1.         1.00098723 1.00183529 ... 1.00009644 1.00024791 1.        ]
 ...
 [1.         1.00031468 1.00009644 ... 0.99997472 1.000089   1.        ]
 [1.         1.00003712 1.00024791 ... 1.000089   1.00003894 1.        ]
 [0.         1.         1.         ... 1.         1.         0.        ]]
```

Punto E y F



FACULTAD DE INGENIERIA
Universidad de Buenos Aires

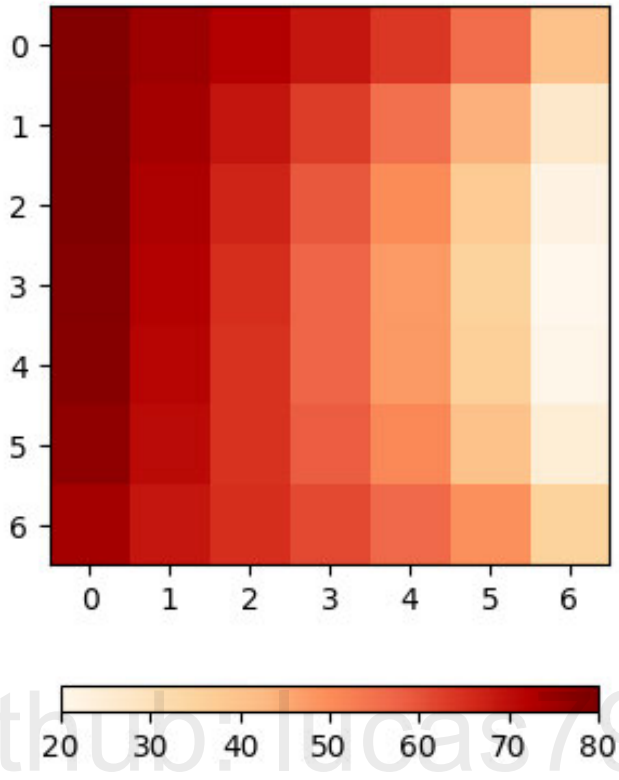
Parte 2

Matriz solución
para la condición de borde
establecida en el trabajo práctico
con padrón 97819

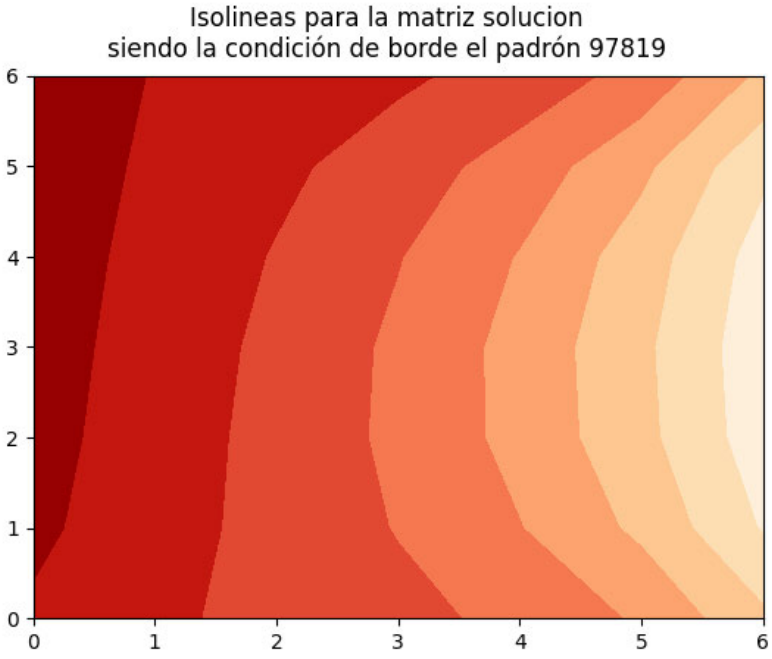
	70	70	70	70	70	70	70
90	78.81	73.37	69.78	66.29	61.86	54.90	41.56
90	81.64	75.09	69.45	63.56	56.28	46.19	31.35
90	82.72	75.97	69.41	62.23	53.50	42.23	27.64
90	83.29	76.73	70.01	62.47	53.27	41.59	26.96
90	83.74	77.67	71.44	64.35	55.51	43.89	28.63
90	83.99	78.78	73.73	68.00	60.53	49.82	33.67
90	83.43	79.73	76.71	73.37	68.78	61.20	46.22
	80	80	80	80	80	80	80

Los valores representados están con 4 dígitos significativos
$\Delta = 0.5x10^{-2}$
Resultados con mayor precisión en el anexo

Mapa de calor en 2D

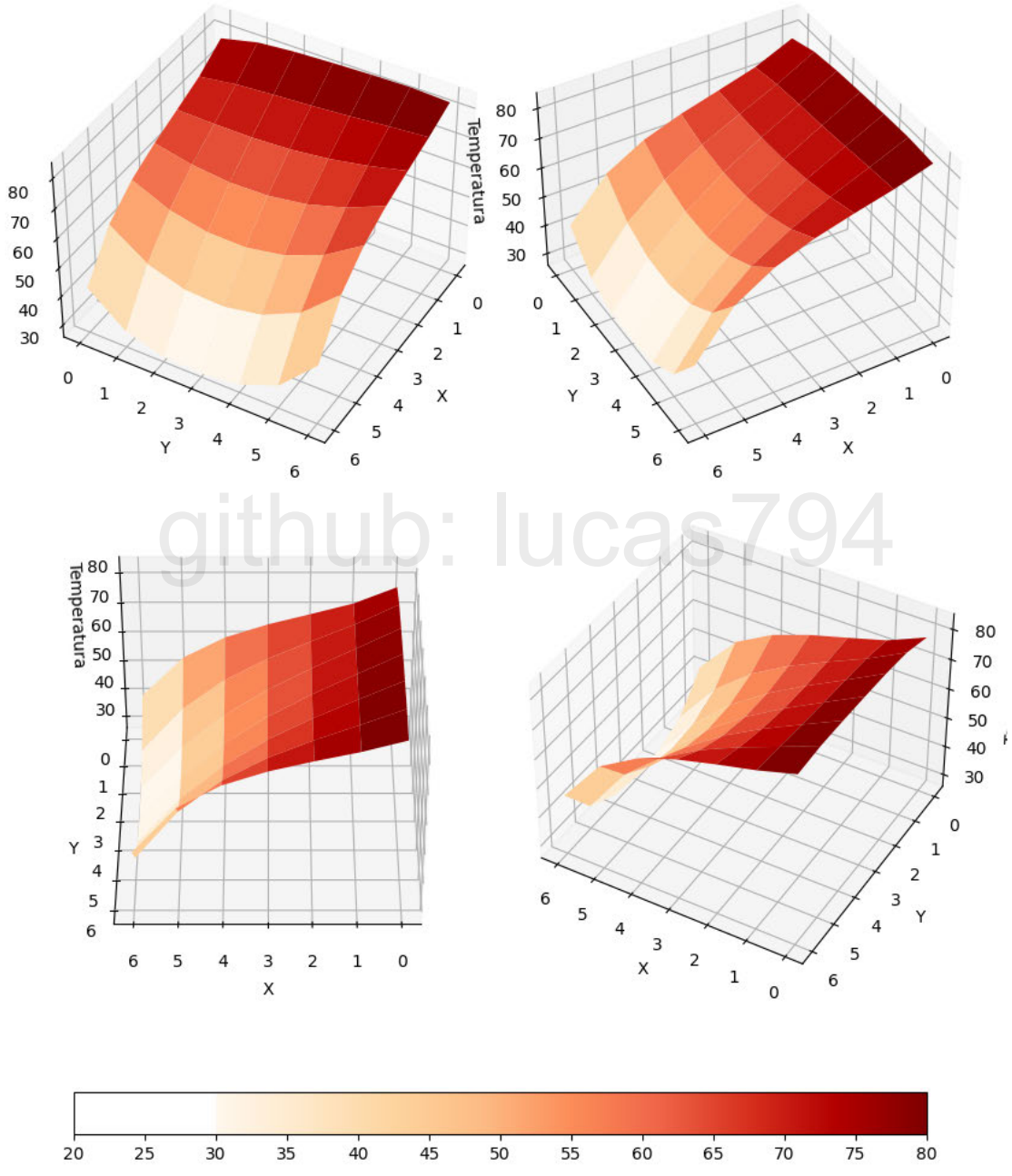


Isolíneas



FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Grafico de superficie
con distintas rotaciones de ángulo



Conclusiones

Se menciona que el radio espectral de la matriz de iteración por Gauss-Seidel se calcula:

$$\rho(T_{gs}) = cos^2(\frac{\pi}{N})$$

y, para el radio espectral de la matriz de iteración por SOR

$$\rho(T_{SOR}) = \omega_{optimo} - 1 < \rho(T_{gs})$$

$$\omega_{optimo} = \frac{2}{1 + sen(\frac{\pi}{N})}$$

Si utilizamos N = 32, y resolviendo estas cuentas de forma teórica, se llega a los valores

$$\rho(T_{gs}) = 0.99039264$$

$$\omega_{optimo} = 1.821465191$$

y, se logra verificar que:

$$\rho(T_{SOR}) = \omega_{optimo} - 1 < \rho(T_{gs})$$

$$1.821465191 - 1 < 0.99039264$$

$$0.821465191 < 0.99039264 \ \checkmark$$

Ahora, si utilizamos los valores que otorga el ejercicio práctico D, vemos que del gráfico

$$\omega_{sor_{optimo}} \cong 1.80$$

$$\rho(T_{gs}) \cong max(\lambda_i) \ ; \lambda_i = \frac{\Delta x^{k+1}}{\Delta x^k}$$

$$max(\lambda) \cong 0.9773$$

$$\omega_{sor_{optimo}} - 1 < 0.9773$$

$$0.80 < 0.9773 \ \checkmark$$

(Aclaración, el lambda está calculado en las iteraciones, se muestra en anexo el valor obtenido)

y se denota, que el valor de $\rho(T_{SOR})$ calculado teóricamente coincide aproximadamente con el generado prácticamente en el gráfico del punto E.

De este último gráfico mencionado, se vé que el $\rho(T_{SOR})$ más chico indica la mejor velocidad con la cual converge el método, y que este valor tiene que ser menor a 1 (Esto si bien se aclara en la teórica, se realizó un ejercicio extra práctico, donde realice el método SOR a esta solución siendo una matriz con N=55 y tolerancia 0.001) y el valor de $\rho(T_{SOR})$ más grande implica que es el el peor ω elegido, generando la mayor cantidad de iteraciones posibles.

También, se denota que, si se logra encontrar el valor ω (en este caso de forma teórica), el método logra converger con la menor cantidad de iteraciones posibles.

Una manera, de encontrar el ω óptimo de forma práctica es utilizar distintos ω entre los valores [1,2], y, en caso de que haya una 'meseta' de iteraciones con distintos ω, se puede establecer una tolerancia más exigente, y ahí concluir cual es el ω optimo con el cual se genera menor iteraciones.

Anexo 1:

El proyecto fue realizado sobre Google Colab, se puede acceder haciendo click aquí. (Se recomienda su visualización allí ya que están los registros de iteración completos, los gráficos presentados en este informe y la manera de cómo se graficó)

No obstante, se agrega la parte fundamental del TP, que es la creación de la matriz, y la resolución del método SOR para una matriz.

```
def calcular_tolerancia(T_sig, T_ant): # Dada 2 matrices calculamos las normas
  numerador = np.linalg.norm(T_sig[1:-1, 1:-1]- T_ant[1:-1, 1:-1]) # solo los valores de los
nodos
  denominador = np.linalg.norm(T_sig[1:-1, 1:-1])

  return numerador / denominador
```

```
def generar_lista_w():
  return np.linspace(1., 1.95, 20)

def generar_w_formula(i):
  return [2 / ( 1+ np.sin( np.pi / (i - 1) ) )]


def crear_solucion_sor(T, tolerancia, generar_w_teorico=True,
**w_personalizado_y_output_datos):
  # Dada una matriz T, tolerancia, resuelve por medio de TSOR la matriz proporcionada.

  #Si no se envia ninguna informacion en generar_w_teorico, se calcula respecto al pedido
de cátedra
  #Si no se genera un arreglo de 1 a 1.95 saltando de a 0.05

  total_nodos = len(T[0, :]) # Necesitamos saber la cantidad de nodos, matriz cuadrada por
lo tanto no importa cual fila agarremos

  if( generar_w_teorico ): #Si se pasa una lista de w, hay que utilizarla
    w_a_iterar = generar_w_formula(total_nodos)
  else:
    w_a_iterar = generar_lista_w() #Generamos los i iterables

  lambda_mas_grande = 0
  for w in w_a_iterar:
    T_iterativa = T.copy() # copia de la ultima T para cada w
    n_iteracion = 0
    ultima_tolerancia = 1
    while True:
      T_anterior = T_iterativa.copy()  # copia de la última T

      for i in range(1, total_nodos - 1):
        for j in range(1, total_nodos - 1):
          # https://drive.google.com/file/d/1xz66KwFax61omaQwcZxJjawLqvfwy21M
          T_iterativa[i, j] = ( (w / 4) * (T_iterativa[i - 1, j] + T_iterativa[i + 1, j] + T_iterativa[i, j
- 1] + T_iterativa[i, j + 1]) ) + ( (1 - w) * T_iterativa[i, j] )

      tolerancia_calculada = calcular_tolerancia(T_iterativa, T_anterior)
      if( tolerancia_calculada < tolerancia ):
        print(f"Se logró la convergencia luego de {n_iteracion + 1} iteraciones. | última
tolerancia : {tolerancia_calculada} | Δ = {np.abs(tolerancia_calculada - tolerancia)}")
        print(f"w = {w}")

        if( 'informacion' in w_personalizado_y_output_datos.keys() ): # se quiere guardar
informacion
          w_personalizado_y_output_datos['informacion'][w] = [ n_iteracion + 1,
tolerancia_calculada ] # guardo [ iteraciones, tolerancia calculada ] auque finalmente no lo
```

```
uso.

            print(T_iterativa)
            break

        ultima_tolerancia = tolerancia_calculada
        n_iteracion += 1

    if( lambda_mas_grande < (tolerancia_calculada / ultima_tolerancia) ):
      lambda_mas_grande = tolerancia_calculada / ultima_tolerancia


  if ( generar_w_teorico ): # Si se decidió solucionar con un solo w, devolvemos la matriz solución
    return T_iterativa[1:-1, 1:-1]
  else:
    print(f"λ mas grande encontrado en las iteraciones = {lambda_mas_grande}")
```

Creación de la matriz de temperaturas


```
def crear_matriz_temperaturas(N, semilla=0, valores_bordes=[1,1,1,1]):

  # armamos una matriz N+1xN+1 llena de ceros.
  # Se le puede pasar una semilla a cada nodo, por defecto en 0.
  # Los valores bordes, establecidos por el TP, por defecto serán [1,1,1,1] donde corresponde [NORTE, SUR, ESTE, OESTE]

  matriz = np.zeros((N+1,N+1)) # Matriz principal, llena de 0

  if( semilla ): # Se establece una semilla por parámetro
    for i in range(1,N):
      for j in range(1, N):
        matriz[i, j] = semilla

  # Ponemos en los bordes algún valor, defecto [1,1,1,1]
  matriz[0, 1:-1] = valores_bordes[0] # Primera fila, desde la primera columna hasta la anteúltima
  matriz[-1, 1:-1] = valores_bordes[1] # Última fila, desde la primera columna hasta la anteúltima
  matriz[1:-1, -1] = valores_bordes[2] # Última columna: desde la primera fila hasta la anteúltima
  matriz[1:-1, 0] = valores_bordes[3] # Primera columna: desde la primera fila hasta la anteúltima

  # La matriz generada queda del estilo =>
```

```
    '''
    0 1 1 1 0
    1 | | | 1
    1 | | | 1
    1 | | | 1
    0 1 1 1 0
    '''

    print(f"Se creó una matriz con {len(matriz[:, 0]) - 2}x{len(matriz[0, :]) - 2} nodos (N-1)x(N-1)")
    print(f"La matriz final tiene dimensión {len(matriz[:, 0])}x{len(matriz[0, :])}")

    return matriz
```

Anexo 2:

Solución parte 2 con condición de bordes establecidos por los últimos 4 dígitos del padrón:


```
Se creó una matriz con 7x7 nodos ( (N-1)x(N-1)
La matriz final tiene dimensión 9x9
Se logró la convergencia luego de 13 iteraciones. | última tolerancia : 0.0005285489309536774 | Δ = 0.0004714510690463226
w = 1.4464626921716894
[[ 0.         70.         70.         70.         70.         70.
  70.         70.          0.        ]
 [90.         78.81310465 73.37117991 69.78052721 66.2945612  61.86369504
  54.90234328 41.56264778 10.        ]
 [90.         81.63990337 75.08852326 69.45177122 63.56003977 56.27680979
  46.18876733 31.34758335 10.        ]
 [90.         82.71704039 75.9733257  69.41206373 62.23260959 53.50144492
  42.22945931 27.63618096 10.        ]
 [90.         83.29474156 76.72925591 70.00998639 62.46598919 53.26711397
  41.58821352 26.96428738 10.        ]
 [90.         83.73722793 77.67004869 71.44153723 64.35436926 55.51010639
  43.88847407 28.6305435  10.        ]
 [90.         83.98681039 78.77949553 73.73167095 67.99643631 60.52689552
  49.82240596 33.66804226 10.        ]
 [90.         83.42995227 79.72912418 76.70794848 73.37058025 68.77593384
  61.2044148  46.2182449  10.        ]
 [ 0.         80.         80.         80.         80.         80.
  80.         80.          0.        ]]
```

Resolución ejercicio C, se muestran las últimas 3 iteraciones:

```
Se logró la convergencia luego de 30 iteraciones. | última tolerancia : 0.00969839263451827 | Δ = 0.00030160736548173056
w = 1.85
[[0.         1.         1.         1.         0.        ]
 [1.         0.99888902 1.00620644 0.9997463  1.        ]
 [1.         1.00620644 1.00712337 1.00840167 1.        ]
 [1.         0.9997463  1.00840167 1.00082719 1.        ]
 [0.         1.         1.         1.         0.        ]]
Se logró la convergencia luego de 46 iteraciones. | última tolerancia : 0.007958316408537657 | Δ = 0.0020416835914623434
w = 1.9
[[0.         1.         1.         1.         0.        ]
 [1.         1.00074097 1.00303549 0.99989104 1.        ]
 [1.         1.00303549 1.00763725 1.00729017 1.        ]
 [1.         0.99989104 1.00729017 0.99493837 1.        ]
 [0.         1.         1.         1.         0.        ]]
Se logró la convergencia luego de 91 iteraciones. | última tolerancia : 0.00995697032894683 | Δ = 4.302967105316961e-05
w = 1.95
[[0.         1.         1.         1.         0.        ]
 [1.         1.00413837 1.00088281 1.00528024 1.        ]
 [1.         1.00088281 1.00116654 0.99122866 1.        ]
 [1.         1.00528024 0.99122866 0.99635855 1.        ]
 [0.         1.         1.         1.         0.        ]]
w = 1.99 | iteraciones necesarias = 8
```

Solución parte 2 con mayor precisión

FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Matriz solución
para la condición de borde
establecida en el trabajo práctico
con padrón 97819

	70	70	70	70	70	70	70	
90	78.81310	73.37118	69.78053	66.29456	61.86370	54.90234	41.56265	10
90	81.63990	75.08852	69.45177	63.56004	56.27681	46.18877	31.34758	10
90	82.71704	75.97333	69.41206	62.23261	53.50144	42.22946	27.63618	10
90	83.29474	76.72926	70.00999	62.46599	53.26711	41.58821	26.96429	10
90	83.73723	77.67005	71.44154	64.35437	55.51011	43.88847	28.63054	10
90	83.98681	78.77950	73.73167	67.99644	60.52690	49.82241	33.66804	10
90	83.42995	79.72912	76.70795	73.37058	68.77593	61.20441	46.21824	10
	80	80	80	80	80	80	80	

$\Delta = 0.5x10^{-5}$

Corridas completas del ejercicio C para distintos ω

```
w = 1.00 | iteraciones necesarias = 8
w = 1.05 | iteraciones necesarias = 7
w = 1.10 | iteraciones necesarias = 6
w = 1.15 | iteraciones necesarias = 6
w = 1.20 | iteraciones necesarias = 6
w = 1.25 | iteraciones necesarias = 6
w = 1.30 | iteraciones necesarias = 7
w = 1.35 | iteraciones necesarias = 7
w = 1.40 | iteraciones necesarias = 8
w = 1.45 | iteraciones necesarias = 8
w = 1.50 | iteraciones necesarias = 9
w = 1.55 | iteraciones necesarias = 9
w = 1.60 | iteraciones necesarias = 10
w = 1.65 | iteraciones necesarias = 12
w = 1.70 | iteraciones necesarias = 15
w = 1.75 | iteraciones necesarias = 18
w = 1.80 | iteraciones necesarias = 23
w = 1.85 | iteraciones necesarias = 30
w = 1.90 | iteraciones necesarias = 46
w = 1.95 | iteraciones necesarias = 91
```


Obtención del valor lambda al finalizar las iteraciones en el ejercicio D:

```
 [0.         1.         1.         ... 1.         1.         0.        ]]
Se logró la convergencia luego de 68 iteraciones. | última tolerancia : 0.008860612644302783 | Δ = 0.0011393873556972169
w = 1.95
[[0.         1.         1.         ... 1.         1.         0.        ]
 [1.         0.99994329 1.00676978 ... 0.99446921 0.9990417  1.        ]
 [1.         1.00676978 1.00858962 ... 0.99677591 0.99716567 1.        ]
 ...
 [1.         0.99446921 0.99677591 ... 0.98156691 0.99331045 1.        ]
 [1.         0.9990417  0.99716567 ... 0.99331045 0.99618052 1.        ]
 [0.         1.         1.         ... 1.         1.         0.        ]]
λ mas grande encontrado en las iteraciones = 0.9773994870620455
```