FACULTAD DE INGENIERIA
Universidad de Buenos Aires

“ANÁLISIS NUMÉRICO I”

“MÉTODOS MATEMÁTICOS Y NUMÉRICOS”

<75.12> <95.04> <95.13>

DATOS DEL TRABAJO PRÁCTICO

1	2 0 2 2	Resolución de Sistemas de Ecuaciones Lineales
	AÑO	Conducción de calor 2D en un anillo”
	2	
TP NRO	CUAT	TEMA

INTEGRANTES DEL GRUPO

	DE LA PLATA FACUNDO	100558
	APELLIDO Y NOMBRE	PADRÓN
GRUPO	APELLIDO Y NOMBRE	PADRÓN

Objetivos

El objetivo del trabajo es analizar la problemática de la transmisión de calor en el tubo durante la soldadura.

Se resolverá el sistema de ecuaciones AX=b obtenido luego de la aplicación del método de las diferencias finitas.

Para esto, se utilizarán los métodos iterativos Gauss-Seidel y SOR. Luego se hará un análisis y comparación entre ambos métodos sobre su convergencia.

Desarrollo

a) Implementación función x=solver_SOR(A,b)

Ver ANEXO I

Se implementó la función utilizando el cálculo de los índices, en base a:

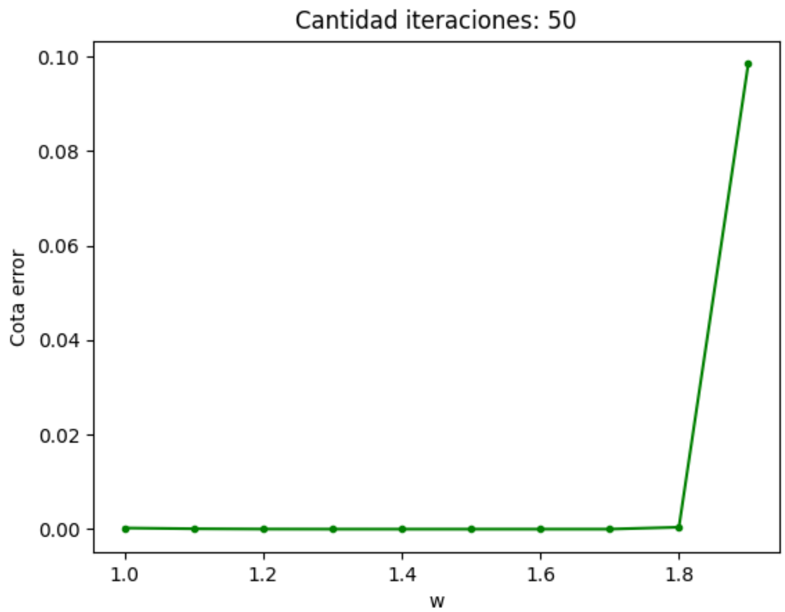
$$x_i^{(k)} = (1-\omega)x_i^{(k-1)} + \frac{\omega}{a_{ii}}\left[b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right]$$

Además, se hizo un análisis de las 3 matrices A, las cuales poseen valores en una cierta cantidad de diagonales. Así, se optimizó el código, de forma que sólo se hagan los cálculos sobre dichas posiciones y evitar hacer cálculos innecesarios con los valores nulos.

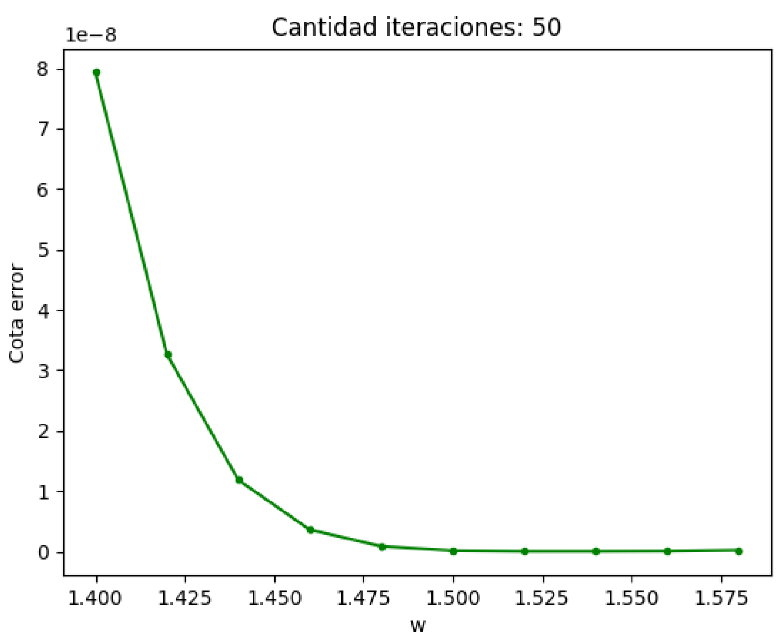
b) Determinación experimental de $w_{óptimo}$

Matriz 090_020

Se calculó la cota del error, fijando la cantidad de iteraciones en 50 y variando en cada caso el valor de w. Inicialmente entre 1 y 2:

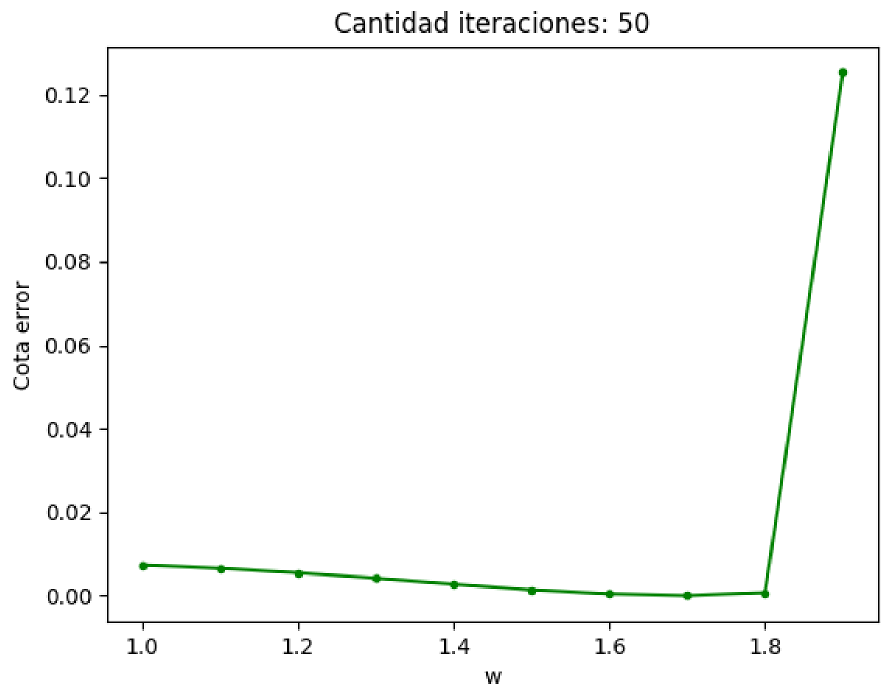


Donde w=1.5 obtuvo el menor error: 1.57e-10. Luego se volvió a calcular otra serie de w entre 1.4 y 1.6, donde se determinó que el $w_{óptimo} = 1{,}52$

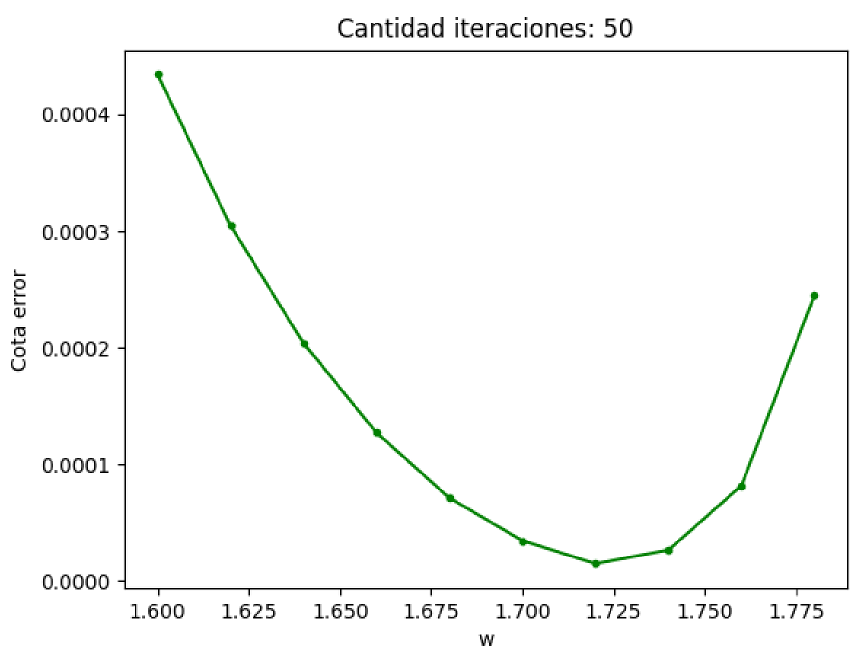


Matriz 180_020

Se calculó la cota del error, fijando la cantidad de iteraciones en 50 y variando en cada caso el valor de w. Inicialmente entre 1 y 2:

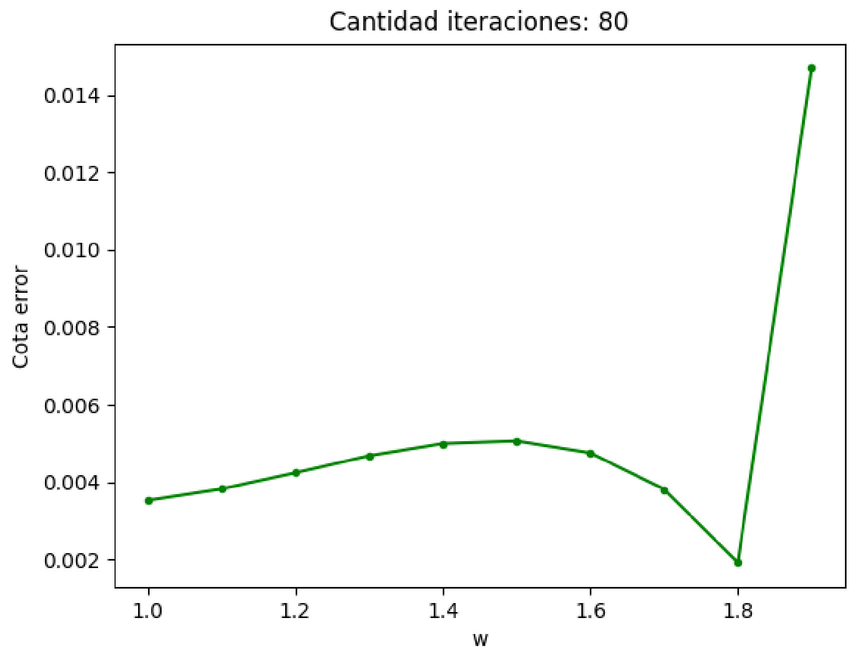


Donde w=1,7 obtuvo el menor error: 3.48e-05. Luego se volvió a calcular otra serie de w entre 1.6 y 1.8, donde se determinó que $w_{óptimo} = 1{,}72$

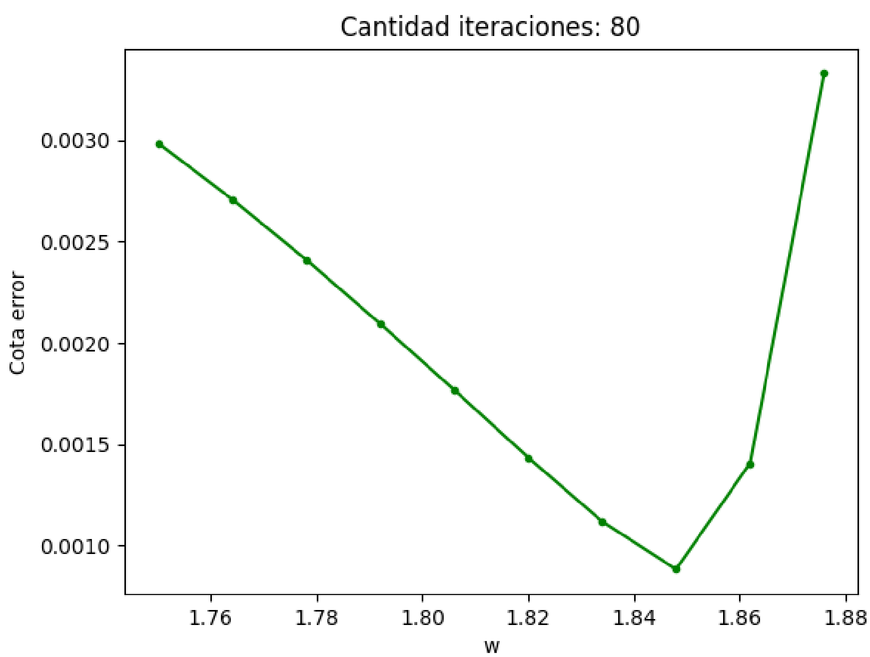


Matriz 360_050

Se calculó la cota del error, fijando la cantidad de iteraciones en 80 y variando en cada caso el valor de w. Inicialmente entre 1 y 1.9:

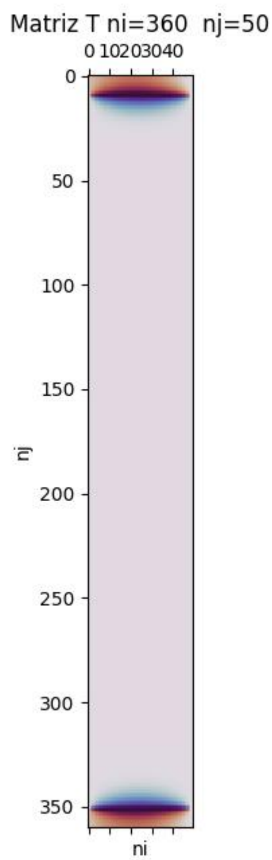


Donde w=1.8 obtuvo el menor error: 0,0054. Luego se volvió a calcular otra serie de w entre más próxima a 1.8, donde se determinó que $w_{óptimo} = 1{,}85$



Valores	$w_{óptimo}$
ni = 90 nj = 10	1,52
ni = 180 nj = 20	1,72
ni = 360 nj = 50	1,85

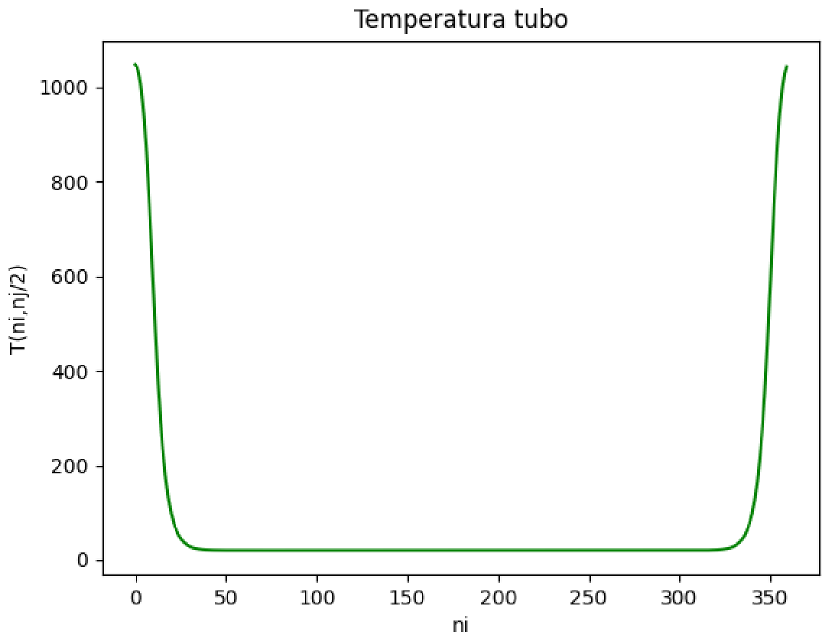
c)



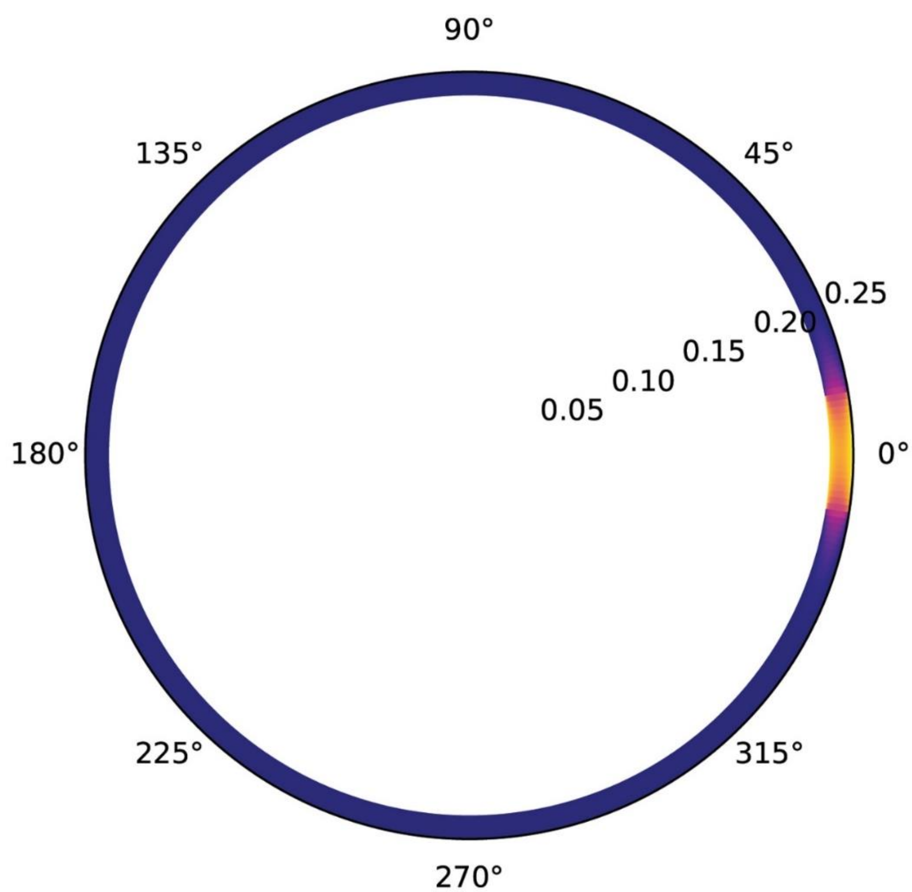
d) Utilizando SOR con el w óptimo en cada caso:

Valores	Tiempo de procesamiento
ni = 90 nj = 10	0,35s
ni = 180 nj = 20	5,2s
ni = 360 nj = 50	126s

e) Temperatura en la mitad de espesor del tubo:



f)



g) Tiempo de procesamiento utilizando Gauss-Seidel

Valores	Tiempo de procesamiento
ni = 90 nj = 10	0,6s
ni = 180 nj = 20	7,6s
ni = 360 nj = 50	212s

h)

Análisis de convergencia

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

$$\boldsymbol{\Delta}^{(k+1)} = |\,\overline{\boldsymbol{x}}^{(k+1)} - \overline{\boldsymbol{x}}^{(k)}|$$

Se calculó $\boldsymbol{p}$ en cada caso, utilizando la diferencia en el error de las últimas 3 iteraciones al llegar a la tolerancia.

Radio espectral (ρ)
Se calculó en cada caso, obteniendo la matriz T_{SOR}

$$T_{SOR} = (D - wL)^{-1}\ .\ [(1-w)D\ +\ wU]$$

Utilizando la descomposición $A = D - L - U$, con D diagonal, L triangular superior y U triangular inferior.

$$\rho(T_{SOR}) = máximo\ autovalor\ de\ T_{SOR}$$

Notar que para analizar el método Gauss-Seidel alcanza con utilizar $w = 1$

Método SOR

Valores	Orden de convergencia $\boldsymbol{p}$	Radio espectral
ni = 90 nj = 10	1,024	0,62
ni = 180 nj = 20	0,988	0,82
ni = 360 nj = 50	1	0,95

Método Gauss-Seidel

Valores	Orden de convergencia $\boldsymbol{p}$	Radio espectral
ni = 90 nj = 10	1	0,90
ni = 180 nj = 20	1	0,98
ni = 360 nj = 50	1	0,996

Se observa en los datos de las tablas, que para todos los casos $\rho(T) < 1$. Esto indica que los métodos convergen.

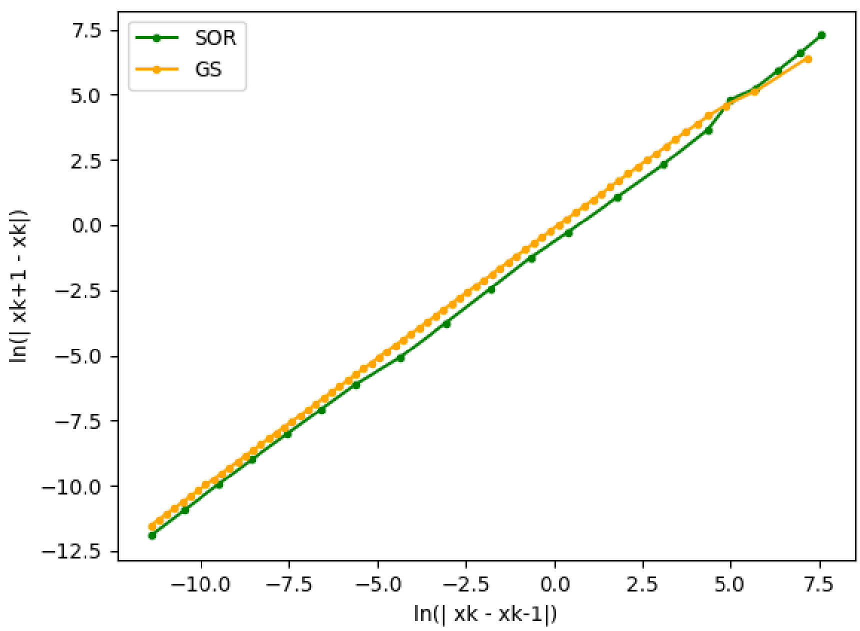
Además, el radio espectral es parámetro de la velocidad de convergencia. Esto se puede notar observando que los valores hallados para Gauss-Seidel son mayores a los de SOR.

También se comprobó que el orden de convergencia de los métodos analizados es lineal de

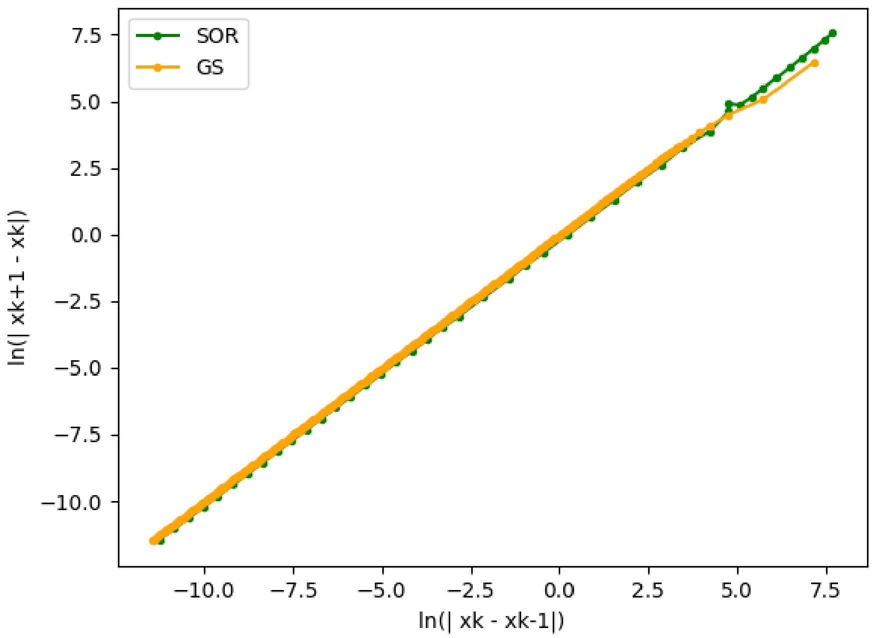
$$\boldsymbol{p = 1}$$

Gráficos Comparación métodos, Error K+1 sobre Error K

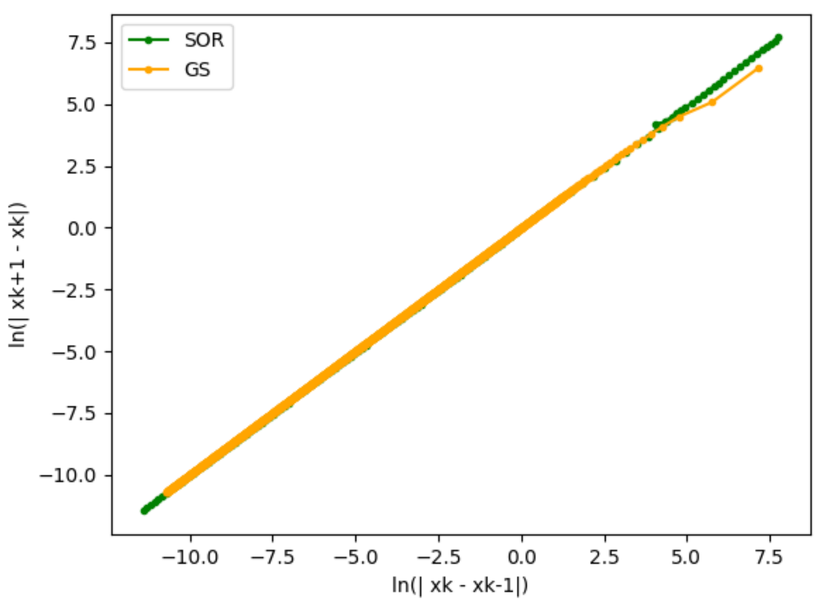
Matriz 090_010



Matriz 180_020



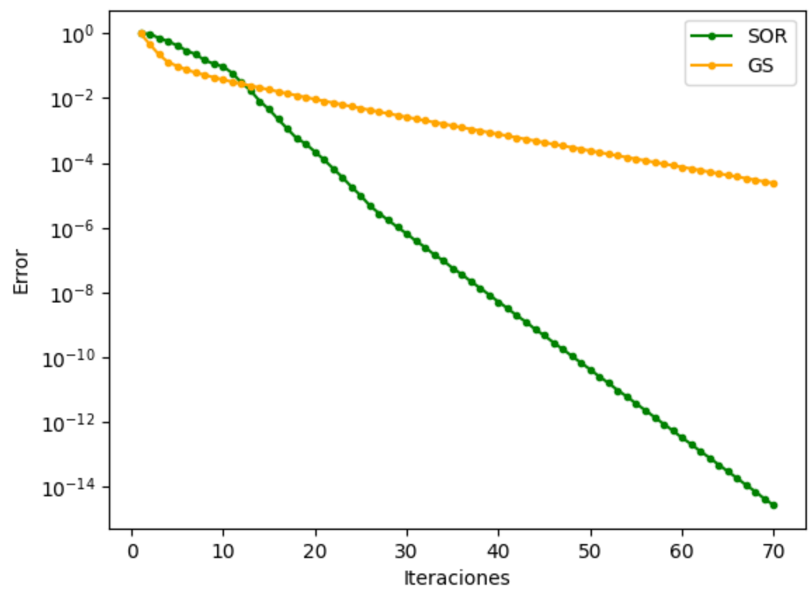
Matriz 360_050



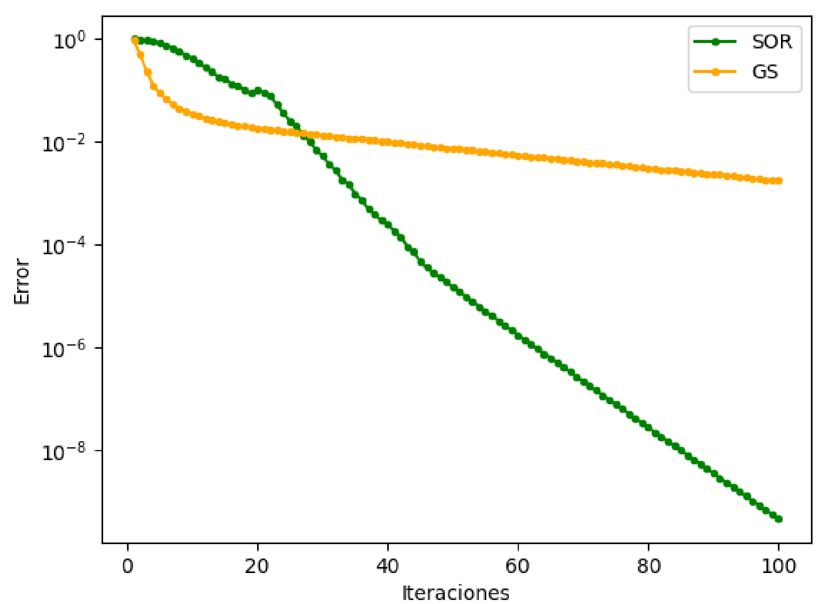
Lo observado en los gráficos se corresponde con lo calculado anteriormente, se halló que $p \approx 1$ para ambos métodos y para todas las matrices. Esto se condice con la pendiente observada en los gráficos.

<u>Gráficos comparación del error relativo en cada iteración</u>

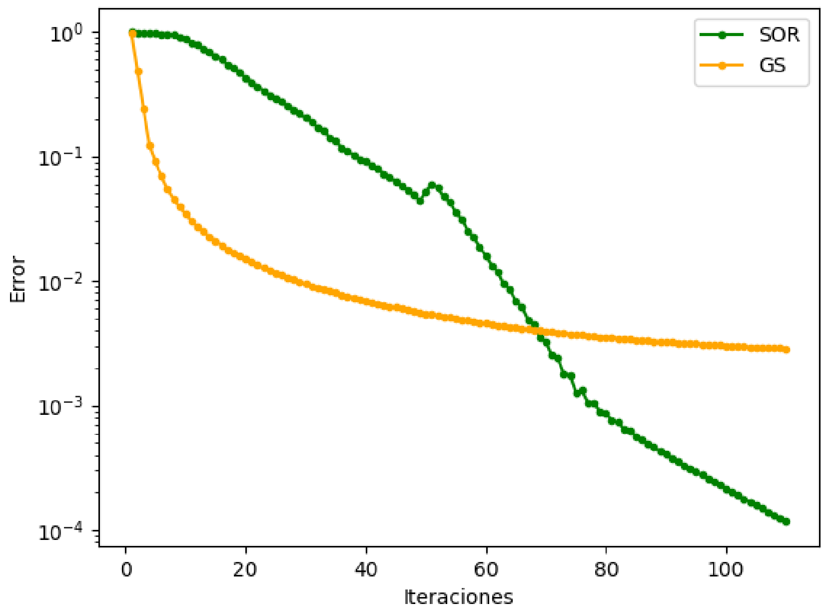
Matriz 090_010



Matriz 180_020



Matriz 360_050



A partir de los gráficos, se observa que inicialmente en unas pocas iteraciones, el método Gauss-Seidel es capaz de reducir el error relativo "más rápido". Sin embargo, a partir de cierta iteración, el método SOR consolida su mayor reducción del error.

i) Para resolver este tipo de sistemas lineales con gran cantidad de ecuaciones, el método SOR es apropiado, ya que al optimizar su w se logra llegar a la solución con un menor error y en menor cantidad de iteraciones.
En la medida de lo posible, es mejor evitar utilizar resoluciones que utilicen matrices y evitar también cálculos que impliquen el cálculo de una matriz inversa, ya que son operaciones con un gran costo computacional.

CONCLUSIONES

Se pudo comprobar experimentalmente que los métodos de resolución iterativos son efectivos. En particular, se probó que el método SOR con su w óptimo da como resultado cantidad de iteraciones y tiempos de cómputo menores.

Se verificó tal como indica la teoría, que ambos métodos iterativos son de orden 1 y que el método SOR tiene una mayor velocidad de convergencia. Esto se ve contrastado en el menor espectro radial y gráficamente en la pendiente más pronunciada observada al graficar el error.

ANEXO I

```
def solver_SOR(A,b):
    tolerancia = 0.00001
    max_iteraciones = 2000
    posicion_valores_090 = [-889, -9, 0, 2, 11, 891]
    posicion_valores_180 = [-3579, -19, 0, 2, 21, 3581]  # No incluyo 1 porque es la
diagonal i=j
    posicion_valores_360 = [-17949, -49, 0, 2, 51, 17951]
    return resolver_SOR(A, b, w, tolerancia, max_iteraciones,posicion_valores)

def calcularR(X,XAnterior):
    diferencia = [0] * len(X)
    for i in range(len(X)):
        diferencia[i] = X[i] - XAnterior[i]
    maxDif = max([abs(valor) for valor in diferencia])
    return maxDif

def calcularRRelativo(X, XAnterior):
    maxActual = max([abs(valor) for valor in X])
    R = calcularR(X,XAnterior)
    return R / maxActual


def calcularPosicionValoresFila(posicion_valores, i):
    posicion = [x + (i - 1) for x in posicion_valores]
    posicion[:] = [x for x in posicion if (x >= 0 and x < tam_matriz)]
    return posicion


def resolver_SOR(A, b, w, tolerancia, max_iteraciones, posicion_valores):
    start = time.time()
    tam_matriz = len(b)
    X = [20] * tam_matriz  # semilla Tamb

    X[0] = b[0]  # La primer fila viene resuelta
    X[tam_matriz - 1] = b[tam_matriz - 1]  # La ultima fila viene resuelta

    for iteracion in range(max_iteraciones):
        XAnterior = X.copy()
        for i in range(1, tam_matriz - 1):
            sum = 0
            posicion_valores_fila = calcularPosicionValoresFila(posicion_valores, i)
```

```
            for j in posicion_valores_fila:
                sum = sum + A[i][j] * X[j]
            X[i] = (1 - w) * X[i] + (b[i] - sum) * (w / A[i][i])

        R = calcularRRelativo(X, XAnterior)
        print("R = " + str(R))
        if R <= tolerancia:
            print("Se llegó a la tolerancia: " + str(tolerancia))
            print("Cantidad de iteraciones: ", iteracion + 1)
            print("R = " + str(R))
            break
    end = time.time()
    print("Tiempo calculo: ", end - start)
    return X


def resolver_GS(A, b, tol, max_iteraciones):
    resolver_SOR(A, b, 1, tol, max_iteraciones)


def leerCSV(nombreArchivo):
    matriz = list()
    with open(nombreArchivo) as csv_file:
        csv_reader = csv.reader(csv_file, delimiter=',')

        for fila in csv_reader:
            filaFloat = [float(i) for i in fila]
            filaNumeros = [int(i) for i in filaFloat]
            if len(fila) == 1:
                matriz.append(int(float(fila[0])))
            else:
                matriz.append(filaNumeros)
    return matriz


def cantidadIteracionesSOR(A, b, w, tolerancia, max_iteraciones):
    tam_matriz = len(b)
    X = [20] * tam_matriz  # semilla

    cantidadIteraciones = 0
    for iteracion in range(max_iteraciones):
        XAnterior = X.copy()
```

```
        for i in range(tam_matriz):
            sum = 0
            for j in range(tam_matriz):
                if j == i:
                    continue
                sum = sum + A[i][j] * X[j]
            X[i] = (1 - w) * X[i] + (b[i] - sum) * (w / A[i][i])

        R = calcularRRelativo(X, XAnterior)
        if R <= tolerancia:
            cantidadIteraciones = iteracion + 1
            break
    return cantidadIteraciones


def graficarResultadoIteraciones(listaW, resultadosError, iteraciones):
    plt.title("Cantidad iteraciones: " + str(iteraciones))
    plt.xlabel("w")
    plt.ylabel("Cota error")
    plt.plot(listaW, resultadosError, color="green", marker='.')
    plt.show()


def hallarWOptimoIteraciones(A, b, wDesde, wHasta, iteraciones, posicion_valores):
    start = time.time()
    resultadosError = list()
    paso = (wHasta - wDesde) / 10
    listaW = np.arange(wDesde, wHasta, paso)  # start (included), stop (excluded),
step
    for w in listaW:
        tam_matriz = len(b)
        X = [20] * tam_matriz  # semilla
        XAnterior = X.copy()

        X[0] = b[0]  # La primer fila viene resuelta
        X[tam_matriz - 1] = b[tam_matriz - 1]  # La ultima fila viene resuelta

        for iteracion in range(iteraciones):
            XAnterior = X.copy()
            for i in range(1, tam_matriz - 1):
                sum = 0
                posicion_valores_fila = calcularPosicionValoresFila(posicion_valores, i)
```

```
            for j in posicion_valores_fila:
                sum = sum + A[i][j] * X[j]
            X[i] = (1 - w) * X[i] + (b[i] - sum) * (w / A[i][i])
        R = calcularRRelativo(X, XAnterior)
        print("w: ", w, " R: ", R)
        resultadosError.append(R)

    end = time.time()
    print("Tiempo calculo: ", end - start)

    graficarResultadoIteraciones(listaW, resultadosError, iteraciones)


def obtenerT(X, numero_filas, numero_columnas):
    T = [[None] * numero_columnas for i in range(numero_filas)]  # Defino la matriz
del tamaño requerido

    for i in range(1, ni + 1):
        for j in range(1, nj + 1):
            kx = j + nj * (i - 1)  # fila-columna
            T[i - 1][j - 1] = X[kx - 1]
    return T


def graficarCentroTubo(T, ni, nj):
    plt.title("Temperatura tubo")
    plt.xlabel("ni")
    plt.ylabel("T(ni,nj/2)")

    temperatura = list()
    for i in range(ni):
        temperatura.append(T[i][int(nj / 2)])
    plt.plot(range(ni), temperatura, color="green")
    # plt.savefig("tempTubo.eps", dpi=1200)
    plt.show()


def graficarTuboPolar(T, rext, wt, ni, nj, dr):
    theta = np.linspace(0, 2 * np.pi, ni)
    r = np.linspace(rext - wt, rext, nj)

    R, THETA = np.meshgrid(r, theta)
```

```
    Z = np.sin(THETA) * R
    plt.subplot(111, projection='polar')
    plt.pcolormesh(THETA, R, T, cmap='plasma')
    plt.gca().set_rmin(0.0)

    plt.show()


def graficarMatrizT(T, ni, nj):
    plt.rcParams["figure.figsize"] = [1, 4]
    plt.rcParams["figure.autolayout"] = True

    fig, ax = plt.subplots()
    ax.set_title("Matriz T ni=" + str(ni) + "  nj=" + str(nj))
    ax.set_xlabel("ni")
    ax.set_ylabel("nj")
    matrix = T
    ax.matshow(matrix, cmap='twilight')
    # plt.savefig("matrizT.eps", dpi=1200)
    plt.show()


def calcularRadioEspectral(A):
    matriz = np.matrix(A)
    autovalores = linalg.eigvals(A)
    a = list()

    return numpy.abs(autovalores).max()


def hallarTSOR(A1, w):
    A = np.matrix(A1,dtype=np.int32)
    # Descomposicion A = d-l-u
    u = -np.triu(A, 1)  # Separa la parte diagonal superior e invierte los signos
    l = -np.tril(A, -1)  # Separa la parte diagonal inferior e invierte los signos
    d = np.tril(np.triu(A))  # Separa la diagonal
    invertida = numpy.linalg.inv(d - w * l)

    tSOR = np.matmul(invertida,((1 - w) * d + w * u))
    return tSOR

def obtenerErroresSOR(A,B,w,cantidadIteraciones,posicion_valores):
```

```
    start = time.time()
    tam_matriz = len(b)
    X = [20] * tam_matriz  # semilla Tamb
    listaErrores = list()
    X[0] = b[0]  # La primer fila viene resuelta
    X[tam_matriz - 1] = b[tam_matriz - 1]  # La ultima fila viene resuelta

    for iteracion in range(cantidadIteraciones):
        XAnterior = X.copy()
        for i in range(1, tam_matriz - 1):
            sum = 0
            posicion_valores_fila = calcularPosicionValoresFila(posicion_valores, i)
            for j in posicion_valores_fila:
                sum = sum + A[i][j] * X[j]
            X[i] = (1 - w) * X[i] + (b[i] - sum) * (w / A[i][i])

        R = calcularRRelativo(X, XAnterior)
        listaErrores.append(R)
        print("R = " + str(R))
    end = time.time()
    print("Tiempo calculo: ", end - start)
    return listaErrores

def obtenerErroresGS(A,b,cantidadIteraciones,posicion_valores):
    return obtenerErroresSOR(A,b,1,cantidadIteraciones,posicion_valores)

def graficarErrorRelativoIteraciones(A, b, w, cantidadIteraciones, posicion_valores):
    resultadosErrorSOR =
obtenerErroresSOR(A,b,w,cantidadIteraciones,posicion_valores)
    resultadosErrorGS = obtenerErroresGS(A,b,cantidadIteraciones,posicion_valores)
    listaIteraciones = range(1,cantidadIteraciones+1)
    print(listaIteraciones)
    print(len(listaIteraciones))
    print(len(resultadosErrorSOR))
    plt.xlabel("Iteraciones")
    plt.ylabel("Error")
    plt.plot(listaIteraciones, resultadosErrorSOR, color="green", marker='.',label=
"SOR")
    plt.plot(listaIteraciones, resultadosErrorGS, color="orange", marker='.',
label="GS")
    plt.yscale('log')
    plt.legend()
```

```
    plt.show()


def obtenerListasErroresSOR(A,b,w,tolerancia,max_iteraciones,posicion_valores):
    tam_matriz = len(b)
    X = [20] * tam_matriz  # semilla Tamb

    X[0] = b[0]  # La primer fila viene resuelta
    X[tam_matriz - 1] = b[tam_matriz - 1]  # La ultima fila viene resuelta
    listaErrores = list()
    for iteracion in range(max_iteraciones):
        XAnterior = X.copy()
        for i in range(1, tam_matriz - 1):
            sum = 0
            posicion_valores_fila = calcularPosicionValoresFila(posicion_valores, i)
            for j in posicion_valores_fila:
                sum = sum + A[i][j] * X[j]
            X[i] = (1 - w) * X[i] + (b[i] - sum) * (w / A[i][i])

        R = calcularR(X,XAnterior)
        listaErrores.append(R)
        print("R = " + str(R))
        if R <= tolerancia:
            print("Se llegó a la tolerancia: " + str(tolerancia))
            print("Cantidad de iteraciones: ", iteracion + 1)
            print("R = " + str(R))
            break
    return listaErrores

def obtenerListasErroresGrafico(listaErrores):
    if len(listaErrores) % 2 != 0:
        listaErrores.pop()
    listaImparEjeY = listaErrores[1::2]  # Elements from list1 starting from 1 iterating
by 2
    listaParEjeX = listaErrores[::2]  # Elements from list1 starting from 0 iterating by 2

    listaImparEjeY[:] = [numpy.log(x) for x in listaImparEjeY]
    listaParEjeX[:] = [numpy.log(x) for x in listaParEjeX]
    return listaParEjeX, listaImparEjeY

def obtenerListasErroresGS(A, b, tolerancia, max_iteraciones, posicion_valores):
    w = 1
```

```
    return
obtenerListasErroresSOR(A,b,w,tolerancia,max_iteraciones,posicion_valores)

def graficarErrorIteracion(A,b,w,tolerancia,posicion_valores):
    listaErrores = obtenerListasErroresSOR(A,b,w,tolerancia,3000,posicion_valores)
    listaEjeXSOR, listaEjeYSOR = obtenerListasErroresGrafico(listaErrores)

    listaErrores = obtenerListasErroresGS(A, b, tolerancia, 3000, posicion_valores)
    listaEjeXGS, listaEjeYGS = obtenerListasErroresGrafico(listaErrores)

    plt.xlabel("ln(| xk - xk-1|)")
    plt.ylabel("ln(| xk+1 - xk|)")
    plt.plot(listaEjeXSOR, listaEjeYSOR, color="green", marker='.',label= "SOR")
    plt.plot(listaEjeXGS, listaEjeYGS, color="orange", marker='.', label="GS")
    plt.legend()
    plt.show()

def calcularOrdenConvergencia(A, b, w, tolerancia, max_iteraciones,
posicion_valores):
    listaErrores = obtenerListasErroresSOR(A, b, w, tolerancia, max_iteraciones,
posicion_valores)
    ultimoError = listaErrores.pop()
    anteUltimoError = listaErrores.pop()
    antePenultimoError = listaErrores.pop()
    p = numpy.log(ultimoError/anteUltimoError) /
numpy.log(anteUltimoError/antePenultimoError)
    return p

def hallarRadioEspectral(A,w):
    tSOR = hallarTSOR(A, w)

    max_ava = calcularRadioEspectral(tSOR)
    print("Radio Espectral: ", max_ava)

if __name__ == '__main__':
    start = time.time()
    padron = 100558

    tHot = padron / 100 + 300  # °C
    tAmb = 20
    ni = 180  # 360  # nodos coordenada angular
    nj = 20  # 50  # nodos coordenada radial
```

```
    n = ni * nj  # nodos totales

    rExt = 0.250  # radio externo del tubo en metros
    wt = 0.015  # espesor de la pared del tubo en metros
    rInt = rExt - wt  # radio interno del tubo en metros
    dr = wt / (nj - 1)  # delta r de la malla en metros

    #A = leerCSV("A_090_010.csv")
    #b = leerCSV("b_090_010.csv")

    A = leerCSV("A_180_020.csv")
    b = leerCSV("b_180_020.csv")

#   A = leerCSV("A_360_050.csv")
#   b = leerCSV("b_360_050.csv")

    posicion_valores_090 = [-889, -9, 0, 2, 11, 891]
    posicion_valores_180 = [-3579, -19, 0, 2, 21, 3581]  # No incluyo 1 porque es la
diagonal i=j
    posicion_valores_360 = [-17949, -49, 0, 2, 51, 17951]

    posicion_valores = posicion_valores_180

    tam_matriz = len(b)
    X = [0] * tam_matriz  # semilla
    max_iteraciones = 20000

    tolerancia = 0.00001

    # Resuelvo el SEL A*x=b por el método Gauss-Seidel
    X = resolver_SOR(A, b, w, tolerancia, max_iteraciones, posicion_valores)
```