Facultad de Ingeniería | Universidad de Buenos Aires

2C Cuatrimestre | 2024

75.12 | Análisis Numérico I
95.10 | Modelación Numérica
95.13 | Métodos Matemáticos y Numéricos

Trabajo Práctico #1

Ecuación de Poisson: resolución numérica del sistema de ecuaciones lineales

Grupo Nº	**11**
Cardoso Juan	**110845**
Balbuena Victoria	**111031**
Gallo Genis Juan Bautista	**111547**

Fecha	Correcciones	Docente

1 Introducción

Para resolver la ecuación unidimensional de Poisson ($\Delta\Phi = \rho$) de forma numérica, se necesita encontrar la solución de un sistema de ecuaciones. Esta ecuación es de utilidad en física y otros campos.
El sistema de ecuaciones lineales a resolver (Ax=b) incluye a una matriz tridiagonal y a ϕj (vector incógnita) y ρj (el vector independiente).

$$\begin{vmatrix} 1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{vmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_n \end{bmatrix}$$

La matriz A es cuadrada, con n filas y n columnas. Sólo tiene un valor (1) en la primera y última fila, mientras que en las filas internas presenta los valores “-1”, “2” y “-1”, dándole una estructura tridiagonal. En el problema particular que se está trabajando, se establece que tanto $\phi 1$ como Φn son iguales a cero. La definición de los restantes coeficientes del vector independiente

$$\varrho j = \hbar^{2} \cdot (-xj \cdot (xj + 3) \cdot e\, xj\,)\; j = 2,3, \dots, n$$

siendo h=1/(n-1) y xj = h * (j-1).
La solución exacta del problema es $uj = xj \cdot (xj - 1) \cdot e\, xj$ (con j=1,2, ….,n).

Dicho problema, en el presente informe, es llevado a cabo en el programa “Octave”. Se emplean dos métodos indirectos, Gauss-Seidel y Jacobi, los cuales se describirán posteriormente.

2. Metodología

Los métodos de iteración indirectos que se llevaron a cabo para resolver sistemas de ecuaciones lineales, fueron los de Gauss-Seidel y Jacobi. El método de

Jacobi actualiza cada variable utilizando los valores de las iteraciones anteriores, sin cambiar los valores actuales hasta que se complete la iteración. En cambio, el método de Gauss-Seidel actualiza cada variable tan pronto como se calcula en la iteración actual, utilizando los valores más recientes de las demás variables. Ambos métodos son iterativos y buscan acercarse a la solución exacta a través de múltiples repeticiones.

3. Resolución

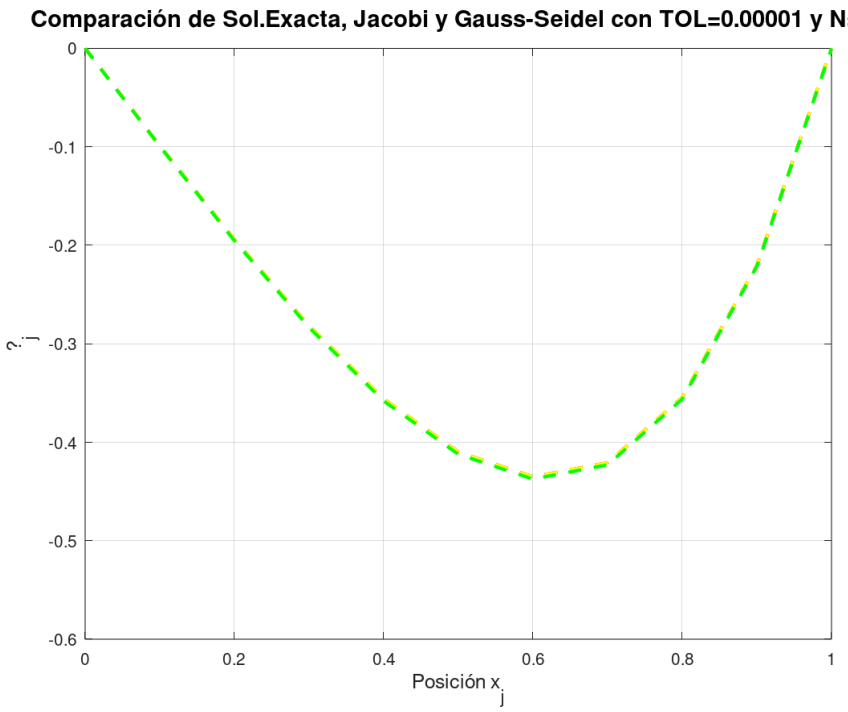
Los métodos iterativos pueden ser un tanto tediosos y repetitivos para resolver manualmente, es por eso que se implementaron algoritmos que permitan agilizar la resolución de los mismos, sobre todo para casos donde las dimensiones de las matrices son muy grandes. Dichos algoritmos siguen la definición de estos métodos, repitiendo los pasos mediante ciclos iterativos(con condición de corte dictado por una constante).

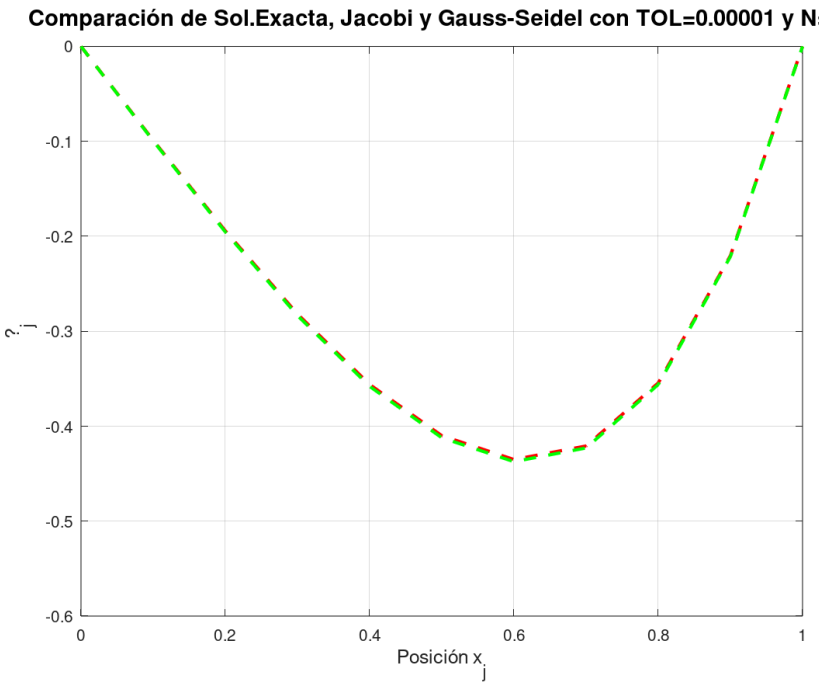
Así mismo, también se definieron funciones que crean las matrices con los valores base que da el enunciado. Dichas funciones crean la matriz A y el vector ρ (a la que por conveniencia llamaremos P) que da el enunciado.

Por último, se definió una función Tp1 que sirve como *main* o función global. En esta se establecen los valores constantes (el valor de N, el valor de la tolerancia permitida etc), se invocan a las funciones previamente definidas y se establecen los gráficos pedidos.

En el anexo de código se puede encontrar la implementación realizada y de la que se obtuvieron los gráficos.

A continuación, se presentan los gráficos pedidos:
En el **ítem c)** se pide una comparación de los valores de las soluciones obtenidas por ambos métodos. El valor del eje X representaría el valor de N normalizado (para que este contenido entre 0 y 1) y en el eje Y el valor solución. Es de notar que ambos gráficos (gauss-seidel y jacobi) están prácticamente superpuestos(De hecho, Jacobi es una gráfica de color rojo, pero está superpuesta por la amarilla de Gauss-Seidel) debido a que el criterio de tolerancia es muy bajo y los valores están muy próximos a la convergencia. La gráfica en verde representa la solución exacta que es el valor al que ambos métodos buscan acercarse. Notamos una diferencia notable en los valores arrojados por los métodos en comparación a los valores exactos.

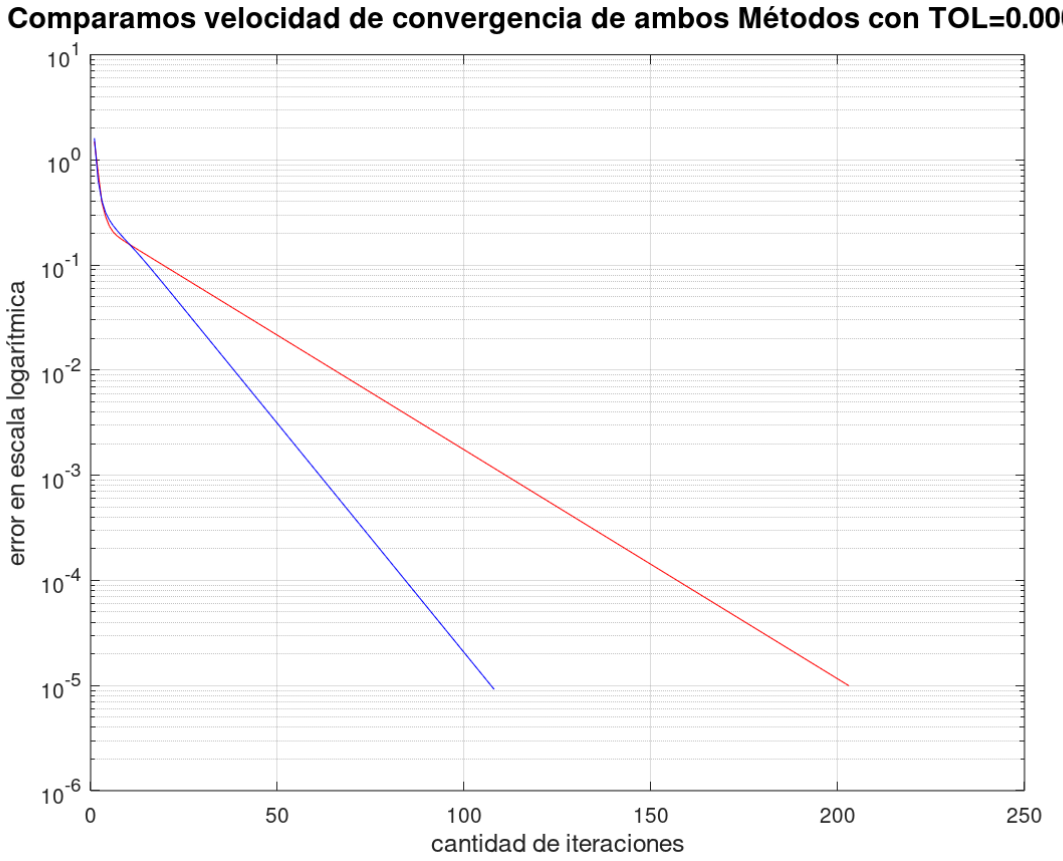

Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.00001 y N=11
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.2
0.4
0.6
0.8
1
Posición x
j


Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.00001 y N=11
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.2
0.4
0.6
0.8
1
Posición x
j

Consideramos que en matrices de mayor tamaño se apreciará más que la diferencia con la solución exacta(color verde) no es tan grande como esta matriz 11x11 puede hacernos ver:

$$A= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

En el **item d)** se pide comparar la velocidad de convergencia en función de la cantidad de iteraciones. En el gráfico se observa que Gauss-Seidel logra una convergencia más rápida que Jacobi, alcanzando la solución en menos iteraciones efectivas, lo cual lo hace más eficiente para sistemas adecuados. Esto se debe a que Gauss-Seidel utiliza el valor actualizado de cada variable tan pronto como se calcula permitiendo que la aproximación se acerque más rápido a la solución exacta. Gauss-Seidel requiere menos iteraciones totales en comparación con Jacobi, que necesita más pasos para reducir el error, al usar siempre los valores de la iteración anterior en todos los cálculos. Adicionalmente, al observar el gráfico, es importante destacar que la escala logarítmica utilizada para el arrastre de error facilita la visualización de las diferencias en la convergencia entre los métodos. En particular, se puede ver que la curva de Gauss-Seidel desciende de manera más pronunciada en las primeras iteraciones, lo que indica una tasa de convergencia más rápida en comparación con Jacobi. Es decir, se llega a errores similares en un menor número de iteraciones, ya que se trabaja con la misma tolerancia.



El **ítem e)** es similar a los ítems anteriores, pero cambiando las dimensiones de la matriz A(nxn) tanto para n=11, n=51 y n=101 (nótese que los dos gráficos anteriores corresponden a una matriz de 11x11). El tiempo que le tomó a cada algoritmo conseguir la solución (con criterio de tolerancia 0.00001) fueron:

-Para Jacobi:

Matriz	Tiempo (seg.)	Iteraciones
11x11	0.209451	202
51x51	80.1753	3827
101x101	1447.12	13520

-Para Gauss-Seidel:

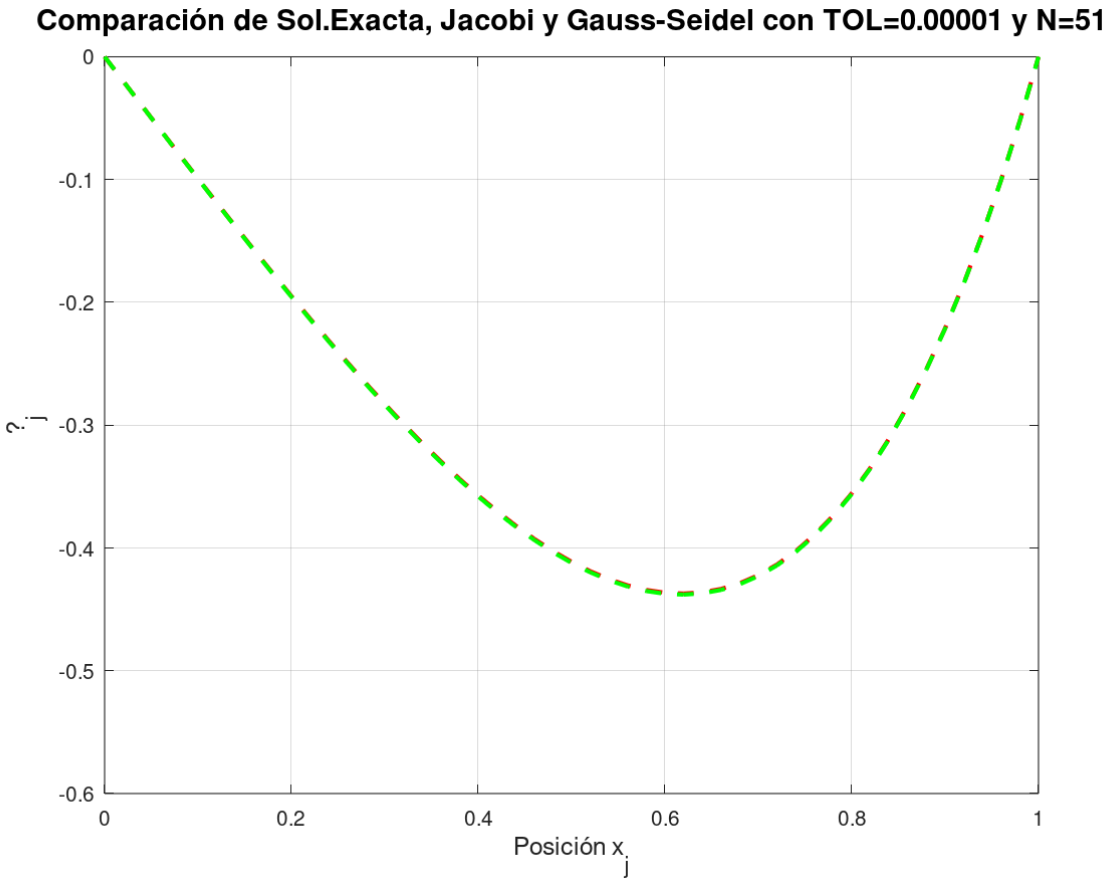
Matriz	Tiempo (seg.)	Iteraciones
11x11	0.120589	117
51x51	46.2087	2105

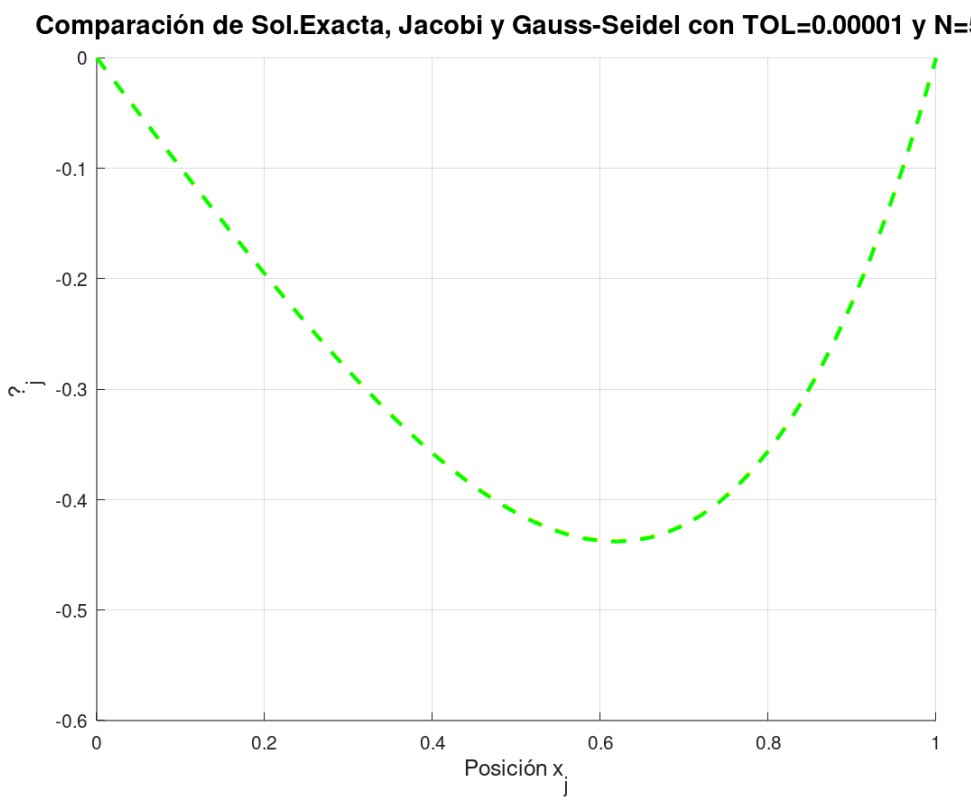
101x101	796.045	5687

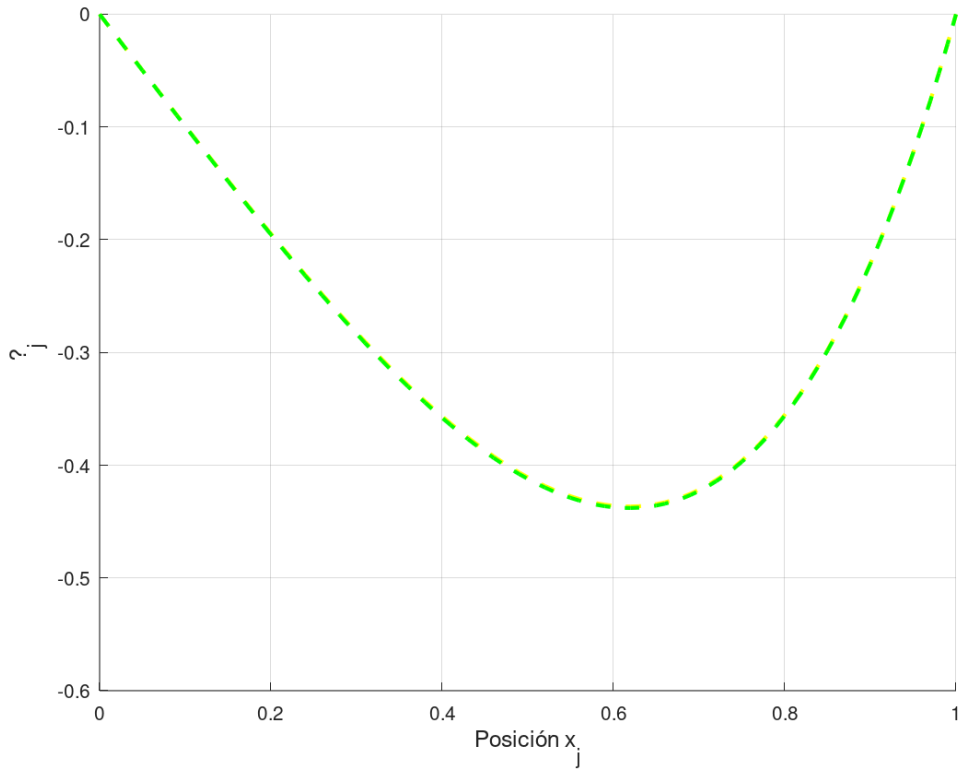
Se demuestra una vez más que Gauss-Seidel es casi el doble de eficiente pues le toman muchas menos iteraciones necesarias para llegar a una solución que satisfaga el criterio de tolerancia. Así mismo, los tiempos de ejecución tienen casi la mitad de la diferencia. Podemos concluir, por ahora que para cualquier tamaño de matriz, se recomendaría ir por gauss seidel pues su diferencia de resultados es objetivamente despreciable pero ejecuta en tiempos mucho menores.

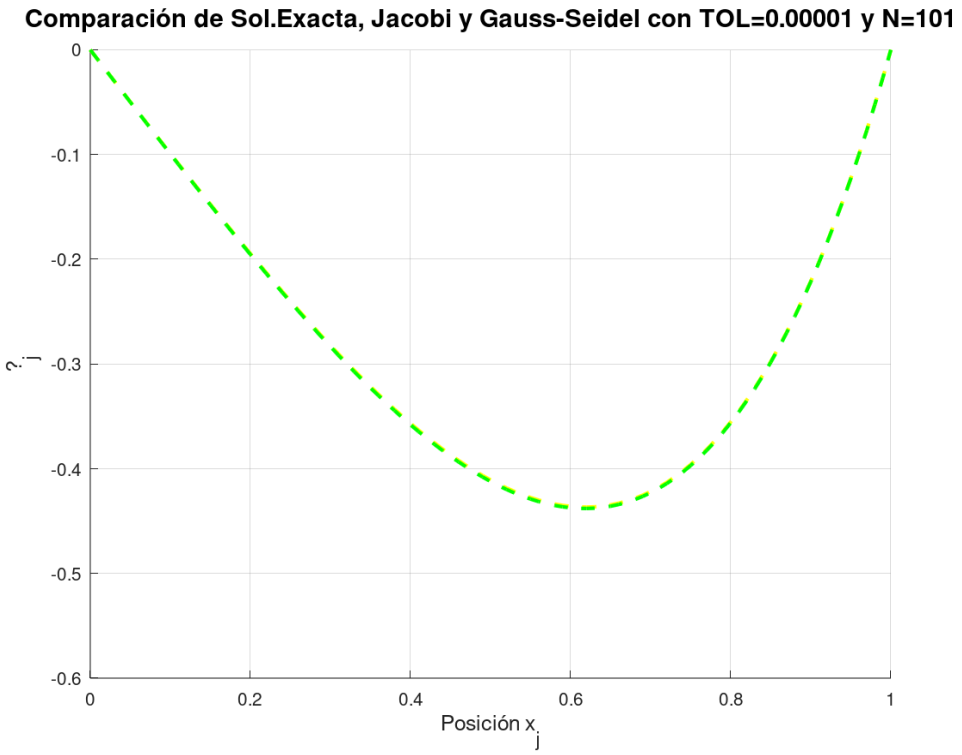
Se muestran los gráficos de las soluciones obtenidas mediante ambos métodos para las distintas discretizaciones (11, 51 y 101 nodos).

Para estos últimos gráficos las dos soluciones arrojan valores muy cercanos, es por eso que se presenta una imagen ampliada a una porción del gráfico para mostrar que estas no se superponen.



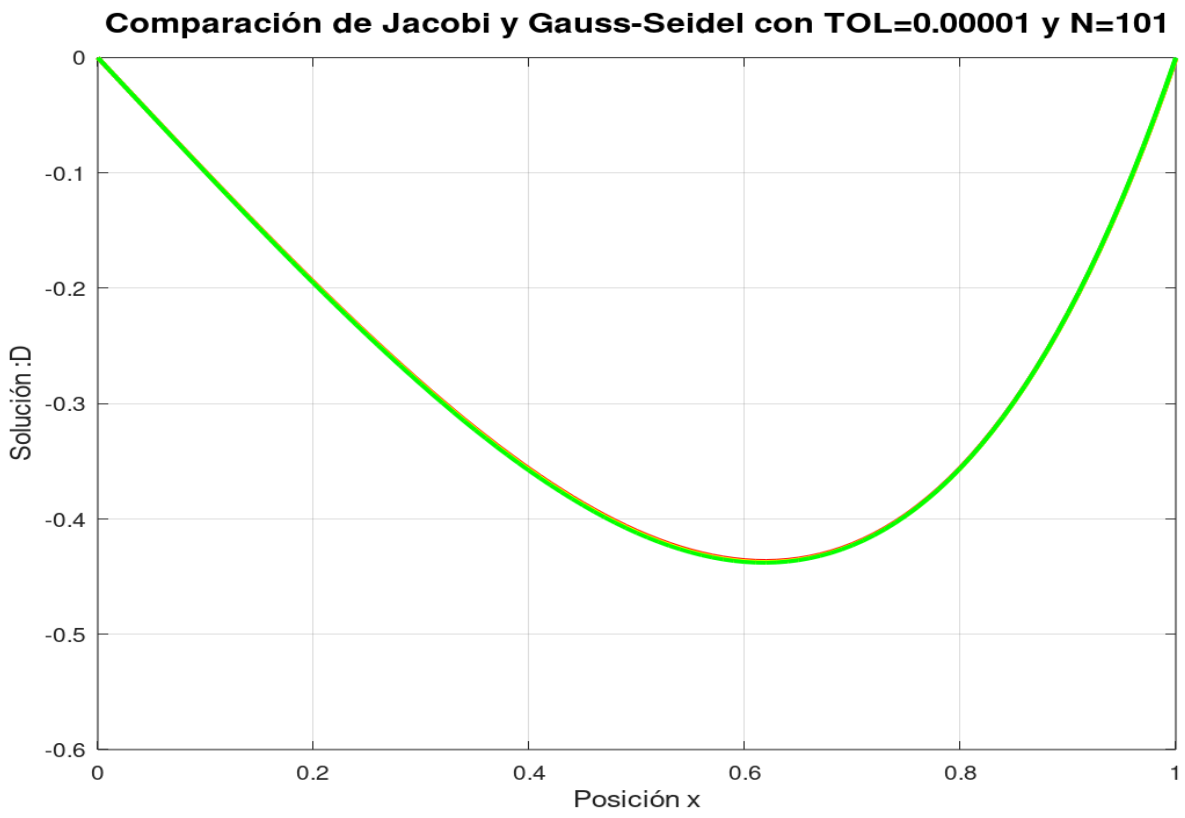

Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.00001 y N=51
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.2
0.4
0.6
0.8
1
Posición x
j
?
j

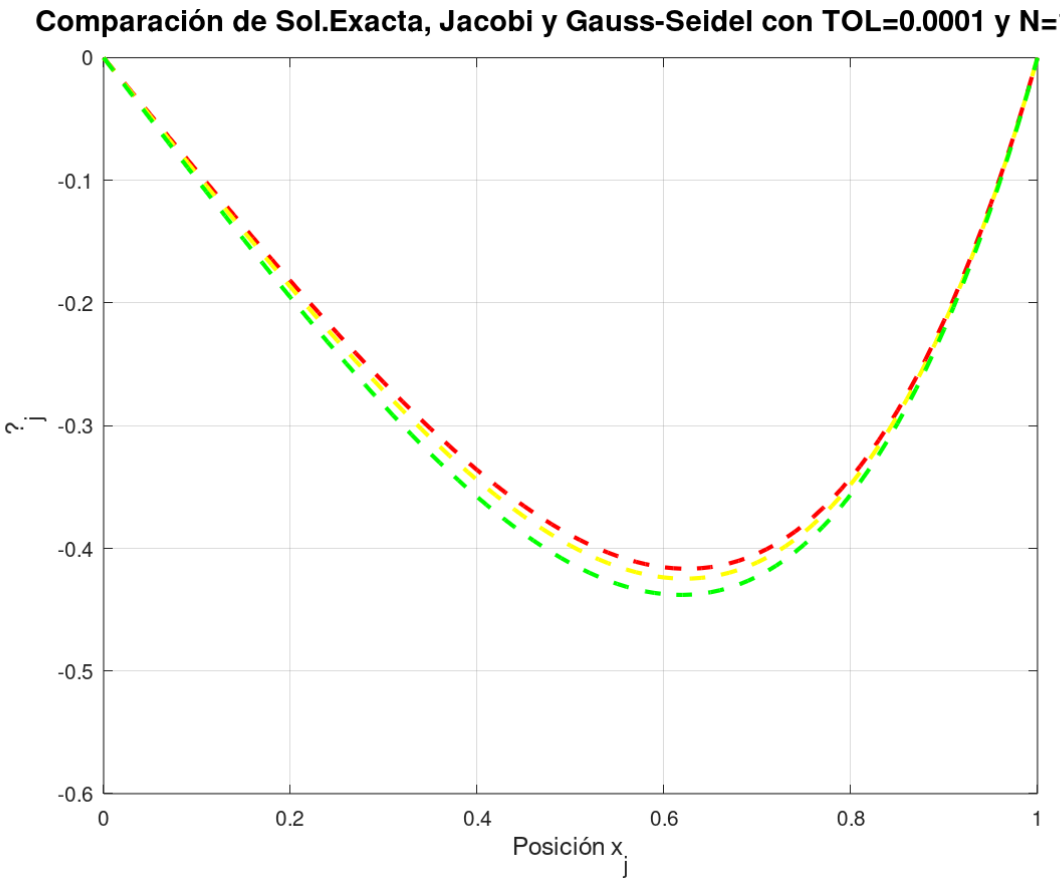

Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.00001 y N=101
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.2
0.4
0.6
0.8
1
Posición x
j
?
j

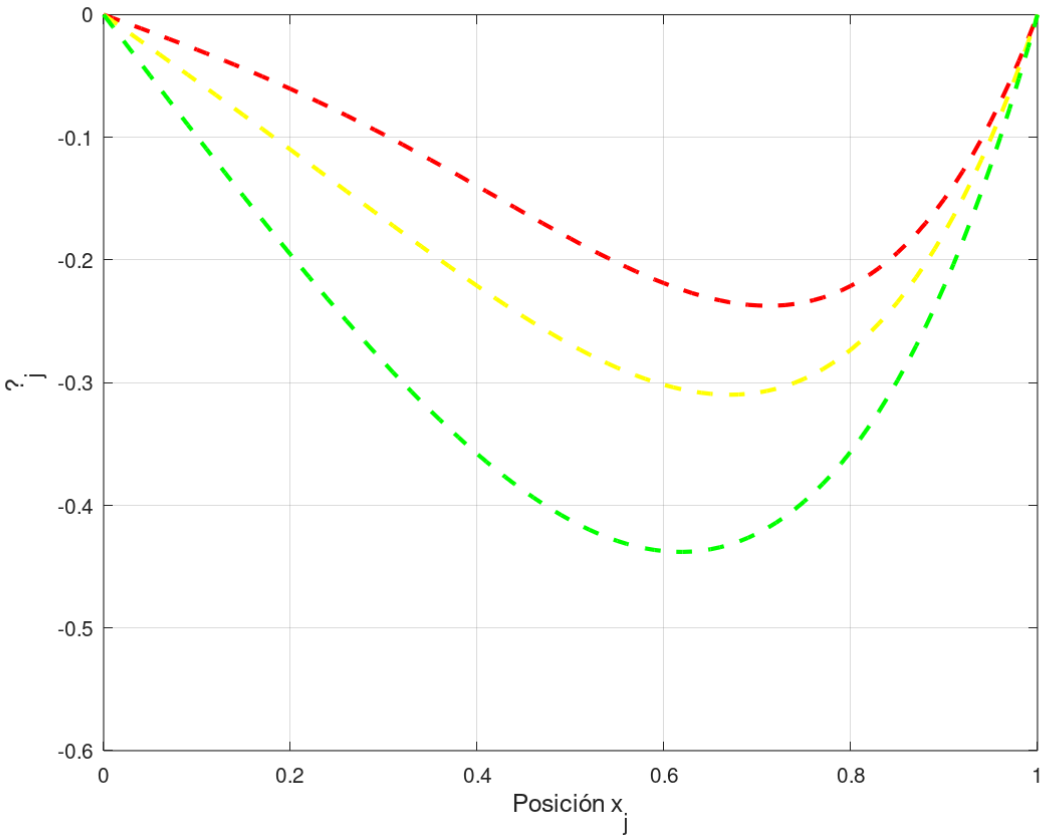


Como se puede apreciar, a medida que aumentamos el tamaño de la matriz, hay mayor acercamiento por parte de los valores de GS y Jacobi con los valores exactos calculados en nuestra solución exacta. Resaltamos también que esta cercanía de valores se debe a que tenemos un criterio de tolerancia razonablemente bajo que permite a los métodos poder ajustarse lo mejor posible. Ya en el próximo ítem veremos como el valor del criterio de tolerancia juega un papel crucial en la aproximación con la solución exacta. Demostramos también que el tamaño de la matriz no termina siendo crucial para ninguno de los métodos. Es decir, deberíamos tener soluciones objetivamente buenas para cualquier N.

Por último, para el **ítem f)** lo que queda es resolver para una matriz 101x101 pero con distintos criterios de tolerancia. Esto debería empujar a que las dos soluciones no converjan tanto pues al haber mayor condición de corte, se puede observar mejor la diferencia entre ambos métodos.
Repetimos el gráfico del ítem anterior para la N=101 y TOL=0.00001


Comparación de Jacobi y Gauss-Seidel con TOL=0.00001 y N=101
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
Solución :D
0
0.2
0.4
0.6
0.8
1
Posición x


Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.0001 y N=101
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.2
0.4
0.6
0.8
1
Posición x
j


Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.001 y N=101
0
-0.1
-0.2
-0.3
-0.4
-0.5
-0.6
0
0.2
0.4
0.6
0.8
1
Posición x
j

-Para Jacobi, con N=101:

TOL	Iteraciones	Tiempo (en seg.)
0,00001	13520	1447,12
0,0001	8677	737
0,001	4010	353

-Para Gauss Seidel, con N=101:

TOL	Iteraciones	Tiempo (en seg.)
0,00001	5687	796,045
0,0001	5227	412
0,001	2241	192

A diferencia del ítem anterior. Lo que pudimos observar es que a menor tolerancia, los métodos se aproximan más a la solución exacta. Sin embargo, si elevamos la tolerancia, notamos como los resultados obtenidos por ambos métodos difieren considerablemente de la solución exacta.
Podemos entonces elegir entre trabajar con criterios de tolerancia bajos para acercarnos lo mejor posible a la solución exacta, pero requiriendo mayor tiempo de ejecución para resolverlos o también podemos elegir trabajar con tolerancias más altas permitiendo ejecuciones rápidas pero con resultados más alejados de los exactos. Cuanto mayor sea la tolerancia, mayor será la divergencia de la solución exacta. Por ende, también disminuye la calidad de las soluciones.
Nótese también que al aumentar el margen de tolerancia, se necesitan menos iteraciones para llegar a la condición de corte. Esto es, en cualquier método utilizado se llegará a la solución con menor cantidad de iteraciones si se aumenta la tolerancia.

4. Conclusiones

A partir de los gráficos obtenidos, que abarcan diferentes tamaños de matrices y criterios de tolerancia, se observa que el método de Gauss-Seidel es preferible para abordar esta problemática, ya que demuestra una mayor eficiencia en tiempos de ejecución en comparación con el método de Jacobi. En base a las tres pruebas realizadas (con tamaños **n=11,51,101**), se observa que el método de Gauss-Seidel converge aproximadamente en la mitad del tiempo de Jacobi. Esto se debe a que Gauss-Seidel requiere menos iteraciones para alcanzar la solución deseada, lo que reduce significativamente los tiempos de ejecución, dado que cada iteración implica un conjunto de cálculos repetidos.

Con respecto a la variación en la tolerancia TOL, se concluye que valores de tolerancia más grandes (por ejemplo, **TOL=0.001**) permiten que ambos métodos convergen en menos iteraciones, acortando los tiempos de ejecución. Sin embargo, esta reducción en la tolerancia puede afectar la precisión de las soluciones numéricas obtenidas. Para valores menores de tolerancia (por ejemplo, **TOL=0.00001**), ambos métodos requieren más iteraciones para alcanzar una solución, aumentando el tiempo de ejecución.

Comparando los resultados obtenidos con la solución exacta, ambos métodos demostraron ser adecuados para resolver la ecuación de Poisson, aunque destacamos Gauss-Seidel por su velocidad sin sacrificar la precisión. La precisión de las soluciones fue consistente con la exacta en todos los casos de tolerancia, aunque se observa que valores más pequeños de n y tolerancias más grandes tienden a producir soluciones menos ajustadas a la exacta en relación a casos con mayor nnn y menor tolerancia, donde la solución obtenida se aproxima mejor a la exacta.

Referencias

Apuntes tomados de la materia “Modelación numérica”, cátedra a cargo del Ing. Re.

ANEXO: Códigos

Algoritmo:

```
1.    function resultado = Tp1()
2.    #numeros magicos
3.    e = exp(1);
4.    n = 101;
5.    h = 1 / (n - 1);
6.    TOL = 0.001;
7.    x=linspace(0,1,n);
8.    #comparamos siempre con la solución exacta
9.    sol_ex=sol_exacta(x);
10.   max_iteraciones = 11111111;
11.   A = getA(n);
12.   P = getP(n, h, e);
13.   tic;
14.   [titon_bobi,arr_err_bobi,iter_bobi] = metodo_bobi(A, P, n,
    max_iteraciones, TOL);
15.   toc;
16.   tic;
17.   [titon_seidel,arr_err_gs,iter_gs] = metodo_gauss_seidell(A, P, n,
    max_iteraciones, TOL);
18.   toc;
19.   fprintf("iteraciones de jacobi",iter_bobi);
20.   fprintf("iteraciones de gs",iter_gs);
21.
22.   #item c, piden gráfico y comparación de la solución para los dos
23.   #métodos
24.   figure(1,'Position', [100, 100, 800, 600]);
25.   plot(x, titon_bobi, '--r', 'LineWidth', 2);
26.   hold on;
27.   plot(x, titon_seidel, '--y', 'LineWidth', 2);
28.   #agrego gráfico de sol exacta
29.   plot(x,sol_ex,"--g",'LineWidth', 2);
30.   hold off;
31.
32.
33.   ylabel("?_j","Fontsize",12);
34.   xlabel("Posición x_j","Fontsize",12);
35.   title("Comparación de Sol.Exacta, Jacobi y Gauss-Seidel con TOL=0.001 y
    N=101","Fontsize",14);
36.
37.   grid on;
38.
39.
40.
41.   #item d, mostrar la cantidad de iteraciones de ambos métodos
42.   #mostrar también el arrastre que va llevando cada método
43.   figure(2,'Position', [100, 100, 800, 600]);
```

```
  semilogy(1:length(arr_err_bobi),arr_err_bobi,"--r");
  hold on;
  semilogy(1:length(arr_err_gs),arr_err_gs,"--b");
  hold off;
  ylabel("error en escala logarítmica","Fontsize",12);
  xlabel("cantidad de iteraciones","Fontsize",12);
  title("Comparamos velocidad de convergencia de ambos Métodos con
TOL=0.00001","Fontsize",14);

  grid on;


  #item e: re utilizo mis funciones pero cambiando el valor de N
  #item f: idem, pero cambiando el valor de la constante TOL


end


#crear A
function A = getA(n)
  A = zeros(n, n);
  A(1, 1) = 1;
  A(n, n) = 1;

  for fila = 2:(n-1)
    A(fila, fila-1) = -1;
    A(fila, fila) = 2;
    A(fila, fila+1) = -1;
  endfor
end

#vreo P
function P = getP(n, h, e)
  P = zeros(n, 1);
  #P(1,1)=0;
  #P(n,1)=1;

  for j = 2:(n-1)
    xj = h * (j - 1);
    pj = (h^2) * ((-xj) * (xj + 3) * (e^xj));
    P(j, 1) = pj;
  endfor
end

# metodo del bobi
function [Titon_bobi,arr_err,iteraciones] = metodo_bobi(A, P, n,
max_iteraciones, TOL)
  #la solucion base es una matriz nx1 con valores 1
  Titon_bobi = ones(n, 1);
  Titon_bobi(1)=0;
  Titon_bobi(n)=0;
  #almacena el arrastre de errores para cada iter
  arr_err=[]
  iteraciones=0;
  for iteraciones = 1:max_iteraciones
    X = zeros(n, 1);
    for i = 1:n
        contador = 0;
```

```
        for j = 1:n
          if j ~= i
            contador = contador + A(i, j) * Titon_bobi(j);
          endif
        endfor
        X(i, 1) = (P(i, 1) - contador) / A(i, i);
      endfor

      error_act=norm(Titon_bobi-X);
      #appendeo el error actual al arreglo
      arr_err(end+1)=error_act;
      #si la diff entre la iteracion actual y la anterior va por
      #debajo de la tolerancia, cortamos ciclo
      if error_act < TOL
        break;
      endif
      Titon_bobi = X;
    endfor
  end

  #metodo seidell
  function [Titon_seidel,arr_err,iteraciones] = metodo_gauss_seidell(A, P,
   n, max_iteraciones, TOL)
    Titon_seidel = ones(n, 1);
    Titon_seidel(1)=0;
    Titon_seidel(n)=0;
    arr_err=[]
    iteraciones=0;

    for iteraciones = 1:max_iteraciones
      X = Titon_seidel;
      for i = 1:n
        var1 = 0;
        var2 = 0;
        for j = 1:n
          if j < i
            var1 = var1 + (A(i, j) * X(j));
          endif
          if j > i
            var2 = var2 + (A(i, j) * Titon_seidel(j));
          endif
        endfor
        X(i, 1) = (P(i) - var1 - var2) / A(i, i);
      endfor
      arr_err(end+1)=norm(Titon_seidel - X);
      if norm(Titon_seidel - X) < TOL
        break;
      endif
      Titon_seidel = X;
    endfor
  end

  function sol_ex = sol_exacta(x)
    #voy a calcular mi solucion exacta dada la formula del enunciado
    sol_ex=x.*(x-1).*(e.^x)
  end

```