Métodos Iterativos con Julia y Python

Cómputo de Álto Rendimiento

Edwin Enrique Pérez Rodríguez

Profesor:
Dr. Oscar A. Esquivel Flores

UNAM
IIMAS

30 de abril de 2022

Introducción

Sea A una matriz no singular de tamaño $n \times n$ y considere un sistema lineal de ecuaciones

$$Ax = b$$

con la solución exacta $x := A^{-1}b$.
Un método iterativo para resolver el sistema $Ax = b$ consiste en empezar con un vector inicial x^0 y con éste construir una sucesión $\{x^{(k)}\}_{k=0}^{\infty}$ de vectores en $\mathbb{R}^n$ que aproximan a la solución exacta $x = A^{-1}b$.
Consideramos métodos iterativos de la forma

$$x^{(i+1)} = \Phi(x^{(i)}), \ \ i = 0, 1, 2, , \ldots .$$

Con la ayuda de una matriz B no singular de tamaño $n \times n$ arbitraria el algoritmo del método iterativo se puede derivar de la siguiente ecuación

$$Bx + (A - B)x = b,$$

y expresándolo como

$$Bx^{(i+1)} + (A - B)x^{(i)} = b,$$

o resolviendo para $x^{(i+1)}$,

$$x^{(i+1)} = x^{(i)} - B^{-1}(Ax^{(i)} - b) = (I - B^{-1}A)x^{(i)} + B^{-1}b.$$

Los métodos iterativos fueron primeramente considerados por Wittmeyer (1936).

Cada elección de una matriz no singular B, conduce a un potencial método iterativo. Cuanto mejor satisfaga B las siguientes condiciones, más útil será el método:

1. el sistema de ecuaciones $Ax = b$ se resuelve fácilmente por x^{i+1},
2. los valores propios de $I - B^{-1}A$ tienen módulos que son tan pequeños como sea posible.

Algunos métodos iterativos clásicos son obetenidos mediante la elección de una B diferente. Consideremos la siguiente descomposición estándar de A :

$$A = D + L + U,$$

$$\text{con } D = \begin{bmatrix} a_{11} & & 0 \\ & \ddots & \\ 0 & & a_n \end{bmatrix},$$

$$L = \begin{bmatrix} 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{21} & \ddots & & & & \vdots \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & \cdots & \cdots & a_{n(n-1)} & 0 \end{bmatrix}, \; U = \begin{bmatrix} 0 & a_{21} & \cdots & \cdots & \cdots & a_{n1} \\ \vdots & \ddots & \ddots & & & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & a_{n(n-1)} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 \end{bmatrix}$$

asumiendo que $a_{ii} \neq 0 \; \forall i = 1, 2, 3, \ldots, n$.

Método de Jacobi

El método iterativo de Jacobi (descrito por C. G, Jacobi en 1845) y también llamado «Proceso de paso total», es una forma de iteración de punto fijo. El sistema de ecuaciones $Ax = b$ puede reordenarse en una iteración de punto fijo de la forma $(D + L + U)x = b$ y se puede obtener la forma matricial del método de Jacobi:

$$Dx = b - (L + U)x$$

y reordenando la expresión anterior se obtiene el método de Jacobi:

x_0: es el vector inicial

$$x_{k+1} = D^{-1}(b - (L + U)x_k), \ \forall k = 0, 1, 2 \ldots .$$

Método de Gauss-Seidel (GS

El método de Gauss-Seidel es mencionado por Gauss in 1826. El segundo nombre tiene su origen en la contribución de Seidel. También es conocido por el nombre de «Método de Desplazamiento Sucesivo» o «Procesos de un solo paso». Gauss-Seidel puede escribirse en forma matricial y se identifica como una iteración de punto fijo en la que se aísla la expresión $(L + D + U)x = b$ como

$$(L + D)x_{k+1} = -Ux_k + b,$$

y reordenando lo anterior obtenemos el método de Gauss-Seidel:

x_0: es el vector inicial

$$x_{k+1} = D^{-1}(b - Ux_k - Lx_{k+1}),\ \forall k = 0, 1, 2 \ldots.$$

Método de Sobre-Relajación Sucesiva (SOR)

La única diferencia entre Gauss-Seidel y Jacobi es que en el primero, los valores más recientemente actualizados de las incógnitas se utilizan en cada paso, incluso si la actualización se produce en el paso actual.

El método llamado Sobre-Relajación Sucesiva (SOR, por sus siglas en inglés) toma la dirección de Gauss-Seidel hacia la solución y lo «rebasa» para tratar de acelerar la convergencia. Sea ω un número real y defina cada componente de la nueva estimación x_{k+1} como una media ponderada de ω veces la fórmula de Gauss-Seidel y $1-\omega$ veces la estimación actual x_k. El número ω se conoce como el **parámetro de relajación**, y $\omega > 1$ se conoce como **sobre-relajación**.

El problema de $Ax = b$ puede escribirse como $(D + L + U)x = b$ y, tras la multiplicación por ω y el reordenamiento, obtenemos la forma matricial del método de SOR,

$$(\omega L + D)x = \omega b - \omega U x + (1 - \omega)Dx,$$

y reacomodando la expresión anterior obtenemos el método de SOR:

x_0: es el vector inicial

$$x_{k+1} = (\omega L + D)^{-1}[(1 - \omega)Dx_k - \omega U x_k] + \omega(D + \omega L^{-1}b), \ \forall k = 0, 1, 2 \ldots.$$

Métodos del Gradiente Conjugado (CGM)

El método del Gradiente Conjugado (Hestenes y Steifel, 1952) marcó el comienzo de una nueva era en los métodos iterativos para resolver los problemas de matrices dispersas. Éste método llega a la solución x del sistema simétrico definido positivo $Ax = b$ con el siguiente ciclo finito:

x_0: vector inicial
$d_0 = r_0 = b - Ax_0$
for $k = 0, 1, 2, \ldots, n-1$
    if r_k es lo suficientemente pequeño
        detenerse
    fin
    $\alpha_k = \dfrac{r_k^T r_k}{d_k^T A d_k}$
    $x_{k+1} = x_k + \alpha_k d_k$
    $r_{k+1} = r_k - \alpha_k A d_k$
    $\beta_k = \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
    $d_{k+1} = r_{k+1} + \beta_k d_k$
end

La iteración del gradiente conjugado actualiza tres vectores diferentes en cada paso. El vector x_k es la solución aproximada en el paso k. El vector r_k representa el residual de la solución aproximada x_k. Esto es claro para r_0 por definición, y durante la iteración, observe que

$$\begin{aligned} Ax_{k+1} + r_{k+1} &= A(x_k + \alpha_k d_k) + r_k - \alpha_k d_k \\ &= Ax_k + r_k, \end{aligned}$$

y así por inducción $r_k = b - Ax_k$ para todo k. Finalmente, el vector d_k representa la nueva dirección de búsqueda utilizada para actualizar la aproximación x_k a la versión mejorada x_{k+1}.

Los métodos de Jacobi, Gauss-Seidel y SOR fueron implementados en los lenguajes de programación de Alto Nivel Julia y Python. Se obtuvieron los resultados numéricos que se muestran en la tablas de las Figuras 1 y 2. Para el método de SOR se usó el parámetro $\omega = 1.13$. Se usa como criterio de paro una tolerancia de $1e^{-8}$. Para realizar el análisis se usa la herramienta "BenchmarkTools"de Julia; en cada análisis se llevan a cabo 100 muestras. Se usa como vector inicial el origen euclidiano.

Para los experimentos numéricos se utiliza la siguiente matriz

$$A = \begin{bmatrix} 3 & -1 & 0 & 0 & 0 & 1/2 \\ -1 & 3 & -1 & 0 & 1/2 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & 1/2 & 0 & -1 & 3 & -1 \\ 1/2 & 0 & 0 & 0 & -1 & 3 \end{bmatrix}$$

y el vector:

$$b = \left[2.5, 1.5, \ldots, 1.5, 1.0, 1.0, 1.5, \ldots, 1.5, 2.5\right]$$

expandibles a los tamaños $n \in \{20.40, 60, 80, 100\}$.

	Jacobi		Gauss-Seidel		SOR		CGM	
Tamaño Matriz	Iteración de convergencia	Error	Iteración de convergencia	Error	Iteración de convergencia	Error	Iteración convergencia	Error
20	74	8.56E-09	41	9.95E-09	32	5.99E-09	11	1.33E-18
40	77	9.84E-09	47	9.74E-09	37	6.49E-09	19	3.27E-09
60	77	8.70E-09	49	7.57E-09	38	6.84E-09	19	3.50E-09
80	77	8.57E-09	49	8.60E-09	38	7.98E-09	19	8.31E-09
100	77	8.56E-09	49	8.85E-09	38	8.30E-09	19	8.20E-09

Figura 1: Análisis de convergencia de los cuatro métodos utilizando Julia. Para el método de SOR se usó el parámetro $\omega = 1.13$. Se usa como criterio de paro una tolerancia de $1e^{-8}$.

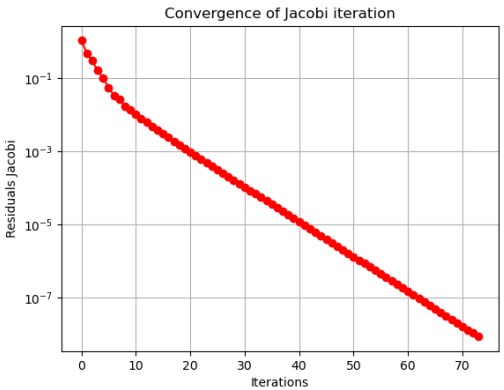
En la Figura 1 podemos observar que el método de Jacobi se estabiliza en cuanto al número de iteraciones (77 iteraciones) en que alcanza la convergencia, con el criterio de paro establecido, a partir de considerar matrices mayor o igual a 40×40. Algo similar sucede con los demás métodos iterativos. Se puede notar que tuvieron mejor desempeño en cuanto a convergencia el método de SOR y CGM. Los cuatro métodos arrojan errores similares cuando se consideran matrices de tamaño 80×80 y 100×100.

En la Figura 2 podemos observar un aumento en los tiempos de ejecución de los cuatro métodos cuando se aumenta el tamaño de la matriz del sistema. OR y CGM. Los cuatro métodos arrojan errores similares cuando se consideran matrices de tamaño 80×80 y 100×100. Notemos que Python tiene un rendimiento menor a Julia en cualquiera de los cuatro métodos. El método de CGM tanto en Julia como Python tiene los mejores tiempo de ejecución.

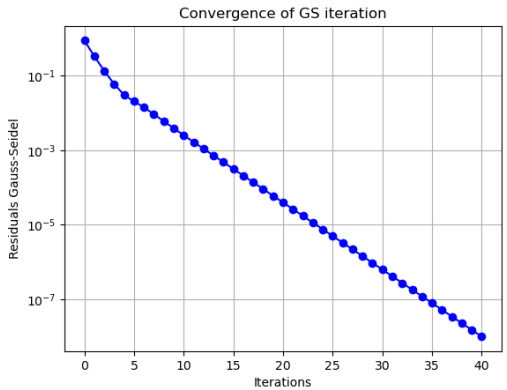
	Jacobi		Gauss-Seidel		SOR		CGM	
Tamaño Matriz	Tiempo Julia	Tiempo Python	Tiempo Julia	Tiempo Python	Tiempo Julia	Tiempo Python	Tiempo Julia	Tiempo Python
20	160.7 µs	2.872 ms	398 µs	1.621 ms	661.3 µs	2.074 ms	11.5 µs	372.9 µs
40	307.7 µs	4.704 ms	992 µs	2.947 ms	1.510 ms	3.895 ms	39.6 µs	630.1 µs
60	623.8 µs	6.051 ms	2.031 ms	3.849 ms	3.171 ms	5.540 ms	97.2 µs	638 µs
80	1.026 ms	8.322 ms	3.183 ms	5.135 ms	5.078 ms	8.036 ms	149.7 µs	704.4 µs
100	1.488 ms	11.93 ms	4.841 ms	7.342 ms	7.658 ms	10.926 ms	221.2 µs	725.7 µs

Figura 2: Análisis de rendimiento (tiempos de ejecución) de los cuatro métodos utilizando Julia y Python. Para el método de SOR se usó el parámetro $\omega = 1.13$. Se usa como criterio de paro una tolerancia de $1e^{-8}$. Para realizar el análisis se usa la herramienta "BenchmarkTools"de Julia; en cada análisis se llevan a cabo 100 muestras.

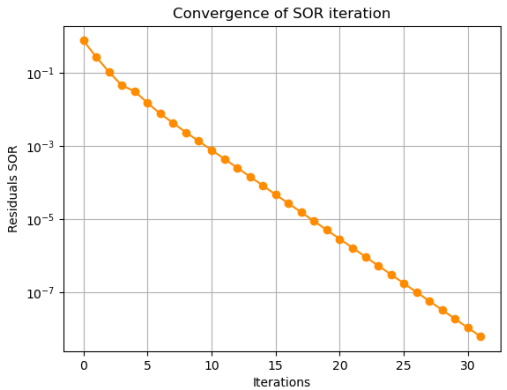
En las Figuras 3, 4, 5, 6 y 7, se muestran las gráficas obtenidas con diferentes tamaños de la matriz A. En todas podemos observar que conforme se aumenta el tamaño de la matriz se van estabilizando los residuos para cada uno de los métodos.



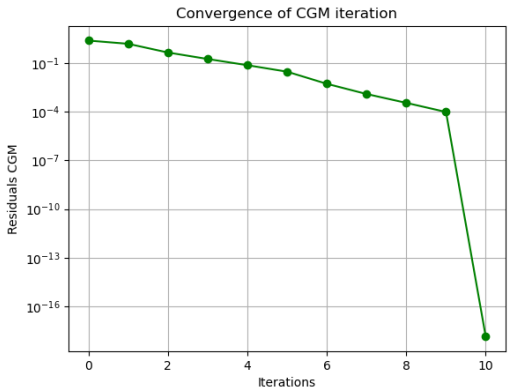
(a) Jacobi



(b) Gauss-Seidel.

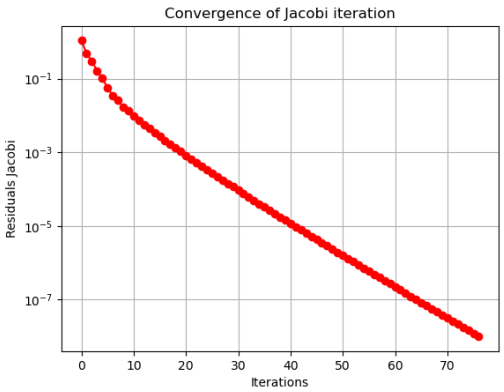


(c) SOR.

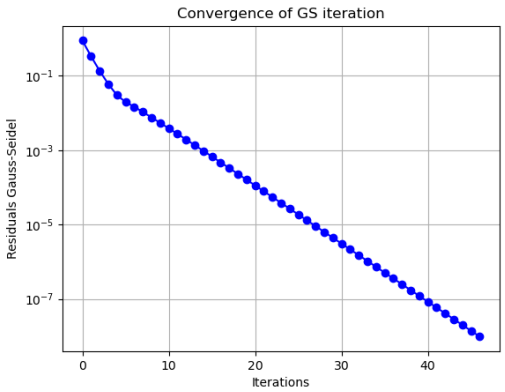


(d) CGM.

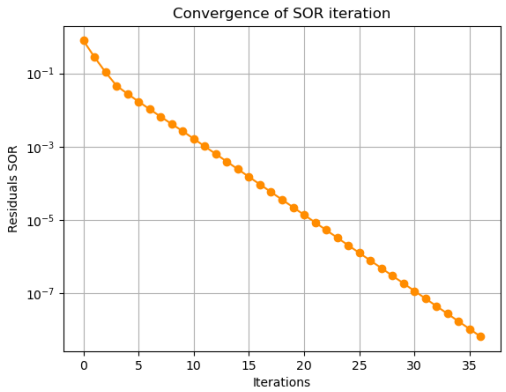
Figura 3: Gráficas obtenidas con la matriz A de tamaño 20×20. Se gráfica los residuos que se obtienen en cada iteración.



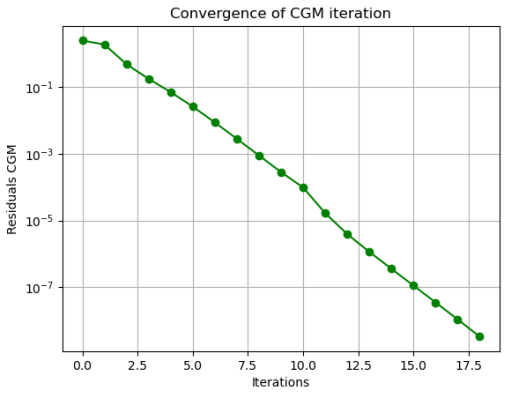
(a) Jacobi



(b) Gauss-Seidel.

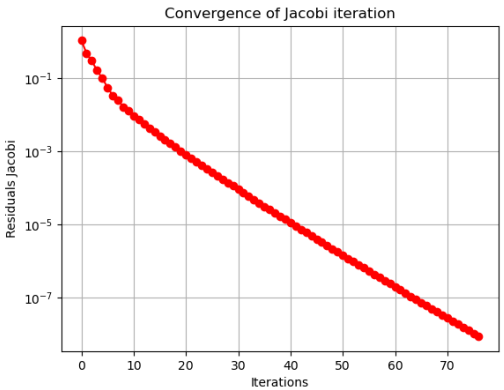


(c) SOR.

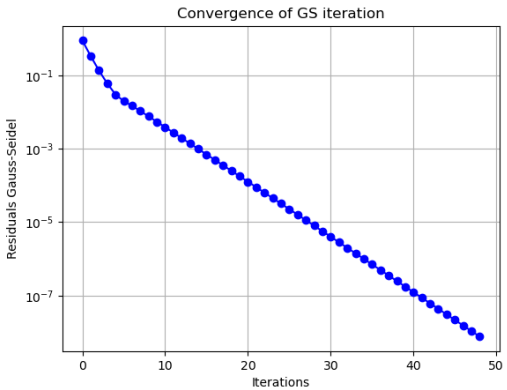


(d) CGM.

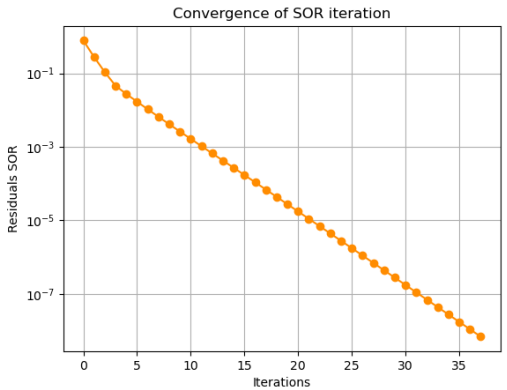
Figura 4: Gráficas obtenidas con la matriz A de tamaño 40×40. Se gráfica los residuos que se obtienen en cada iteración.



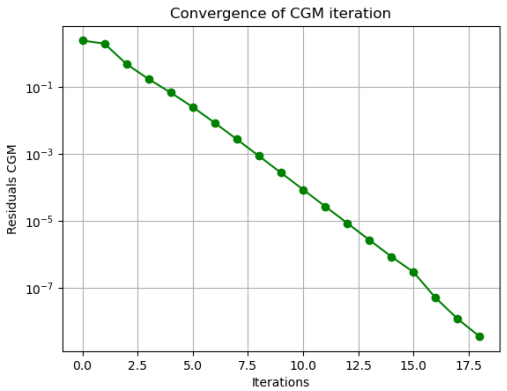
(a) Jacobi



(b) Gauss-Seidel.

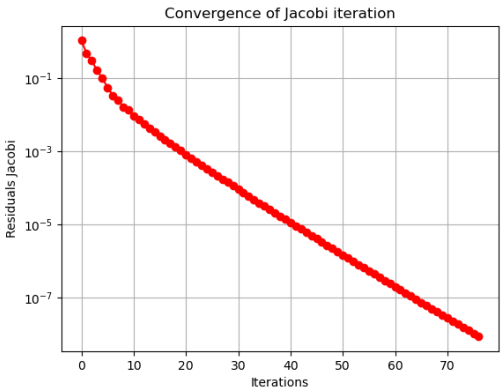


(c) SOR.

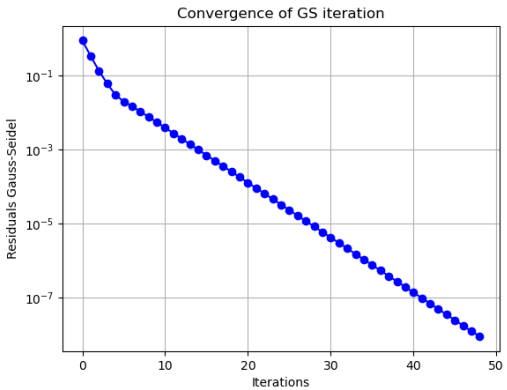


(d) CGM.

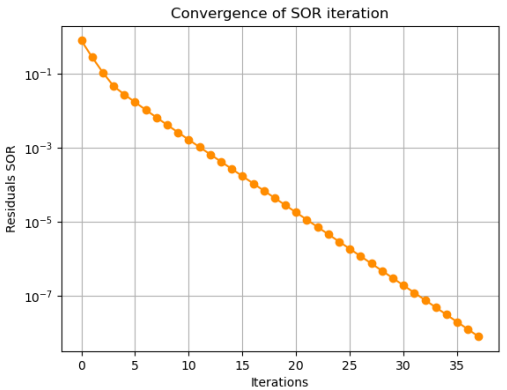
Figura 5: Gráficas obtenidas con la matriz A de tamaño 60×60. Se gráfica los residuos que se obtienen en cada iteración.



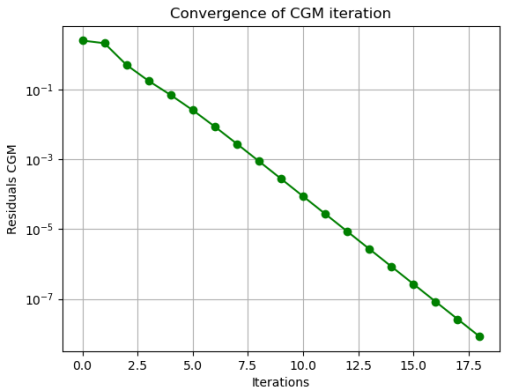
(a) Jacobi



(b) Gauss-Seidel.

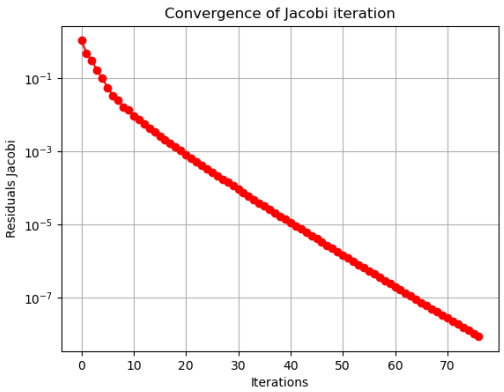


(c) SOR.

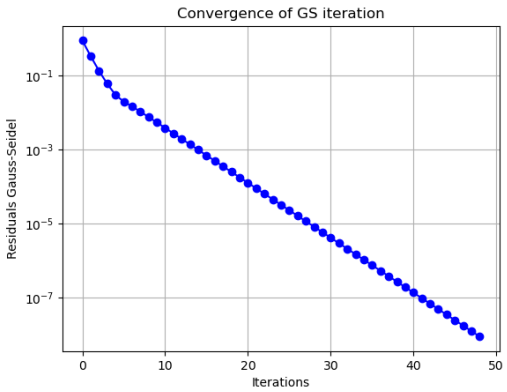


(d) CGM.

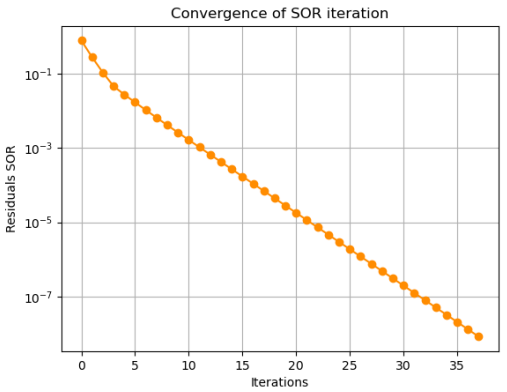
Figura 6: Gráficas obtenidas con la matriz A de tamaño 80×80. Se gráfica los residuos que se obtienen en cada iteración.



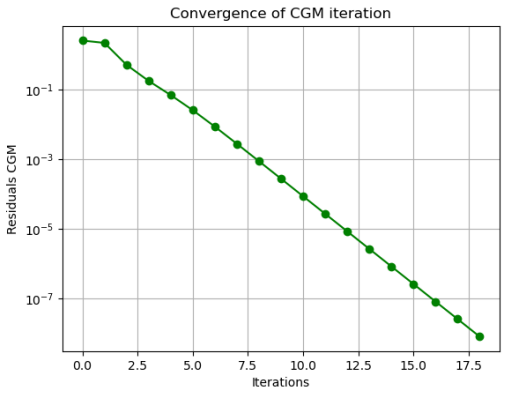
(a) Jacobi



(b) Gauss-Seidel.



(c) SOR.



(d) CGM.

Figura 7: Gráficas obtenidas con la matriz A de tamaño 100×100. Se gráfica los residuos que se obtienen en cada iteración.

Conclusiones

De este microproyecto queda claro la superioridad en cuanto a rendimiento que tiene Julia sobre Python. Ciertamente este último tiene mucha más popularidad que Julia. En mi propia expriencia, el lenguaje de programación de Julia está mejor integrado que Python al no tener que usar librerías externas como "numba"para mejorar su rendimiento.

Tanto el método de SOR y CGM son los que tienen mejores resultados en el análisis de convergencia, pero es CGM quien obtuvo la mejor convergencia con el menor número de iteraciones.

Referencias I

[1] Johnson, S. (20 de mayo de 2020). *PyCall*. Julia https://www.juliapackages.com/p/pycall

[2] Hackbusch, W. (2016). *Iterative Solution of Large Sparse Systems of Equations*. Leipzig: Springer.

[3] Sauer, T. (2012). *Numerical Analysis. 2nd ed.*. Boston: PEARSON.

[4] Stoer, J. and Bulirsch, R. (2002). *Introduction to Numerical Analysis*. New York: Springer-verlag.