XXI ENMC
ENCONTRO NACIONAL DE MODELAGEM COMPUTACIONAL
IX ECTM
ENCONTRO DE CIÊNCIA E TECNOLOGIA DE MATERIAIS

08 a 11 de Outubro de 2018
Instituto Federal Fluminense
Búzios - RJ

UM ESTUDO COMPARATIVO ENTRE MÉTODOS ITERATIVOS PARA RESOLUÇÃO DE SISTEMAS LINEARES

Rafaela Correia Brum[1] - rafinhacbrum@gmail.com
Maria Clicia Stelling de Castro[1] - clicia@ime.uerj.br
Cristiane Oliveira de Faria[1] - cofaria@ime.uerj.br
[1]Programa de Pós-Graduação em Ciências Computacionais, Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro - Rio de Janeiro, RJ, Brazil

Abstract. *Um estudo comparativo entre métodos iterativos para resolução de sistemas lineares de grande porte já estabelecidos na literatura e alguns propostos recentemente é apresentado neste trabalho. Os métodos escolhidos foram: Jacobi-Richardson (JR), Gauss-Seidel (GS), sobre-relaxação sucessiva (SOR), Jacobi-Richardson com relaxação (JOR) e os métodos modificados de Gauss-Seidel para execução distribuída (DGS) e JOR com relaxação atrasada (DOR). Em todos os experimentos foram utilizados matrizes simétricas diagonalmente dominantes geradas aleatoriamente. Além disso, implementamos o DGS com a biblioteca MPI (Message Passing Interface) para troca de mensagens, diferente da versão original baseada em PVM (Parallel Virtual Machine). Neste estudo, foi observado que o método GS é o método que converge com mais rapidez. Com relação ao método DGS, se concluiu que o ganho de desempenho obtido com a versão implementada é um pouco menor do que o ganho obtido com a versão original.*

Keywords: *Sistemas lineares, métodos iterativos, algoritmos modificados*

1. INTRODUÇÃO

Resolução de sistemas lineares é uma etapa fundamental no processo de obtenção de soluções numéricas em simulações computacionais de problemas reais, não importa qual foi o método numérico escolhido na etapa de discretização do modelo matemático. Um sistema linear é descrito por m equações com n incógnitas x_i como em: $a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{in}x_n = b_i$, e é usualmente representado na forma matricial $Ax = b$, que facilita a visualização computacional.

É bem estabelecido que sistemas lineares podem ser resolvidos seguindo duas metodologias, chamadas de métodos diretos ou iterativos. Os métodos diretos solucionam o sistema com um número finito de operações na matriz de coeficientes e em sua maioria estão associados com processos de decomposição. Já os métodos iterativos, partindo de um valor inicial, calculam sucessivas aproximações até atingir a convergência. A convergência da solução do método de-

pende de alguns fatores como o condicionamento da matriz de coeficientes (Cristina & Cunha, 2000).

Neste trabalho, o foco está nos métodos iterativos sendo realizado um estudo comparativo entre os seguintes métodos: Jacobi-Richardson (JR) (Cristina & Cunha, 2000), Gauss-Seidel (GS) (Franco, 2006), método de sobrerelaxação (SOR) (Cristina & Cunha, 2000) e o Jacobi com sobrerelaxação (JOR) (Saad, 2003). Além desses, outros dois métodos mais recentes foram comparados. O método proposto por Shang (2009), que é uma versão paralelizada do método Gauss-Seidel (DGS) e o método JOR modificado (DOR) de Antuono & Colicchio (2016).

Este trabalho está organizado da seguinte forma. A Seção 2. apresenta uma breve descrição dos métodos aqui comparados. Na Seção 3. os detalhes das implementações destes algoritmos são abordados e os resultados numéricos obtidos são discutidos na Seção 4.. A Seção 5. conclui este trabalho com algumas considerações e sugestões de trabalhos futuros.

2. MÉTODOS ITERATIVOS

Seja a matriz dos coeficientes A decomposta da seguinte forma:

$$A = L + D + U, \tag{1}$$

onde nesta decomposição, as matrizes L e U são matrizes triangulares inferiores e superiores, respectivamente, e a matriz D é uma matriz diagonal. Saad (2003) utiliza a decomposição mostrada em Eq. (2), considerando que $E = -L$ e $F = -U$ e assim tem-se que

$$A = D - E - F. \tag{2}$$

Esta decomposição gera maior rapidez computacional em relação à Eq. (1) pois os esquemas finais dos métodos iterativos realizam apenas operações de adição em vez de subtrações. Esta rapidez decorre do modo como a operação de subtração é implementada nos processadores. Os valores são representados em complemento a dois e realizadas apenas operações de adição (Patterson & Hennessy, 2014).

Como ponto de partida para proposta dos métodos iterativos para obtenção da solução de um sistema linear ($Ax = b$), com b sendo o vetor do lado direito, x o vetor de incógnitas e a matriz A decomposta como em Eq.(2), tem-se que

$$x = D^{-1}(b + Ex + Fx). \tag{3}$$

No método de Jacobi-Richardson (JR) utiliza-se o vetor $x^{(i-1)}$ da iteração anterior para calcular o vetor $x^{(i)}$ da iteração atual, como pode ser visto na Eq. (4).

$$x^{(i)} = D^{-1}(b + Ex^{(i-1)} + Fx^{(i-1)}) \tag{4}$$

O método de Gauss-Seidel (GS), Eq. (5), foi proposto a partir de uma pequena modificação no método JR. Esta modificação é baseada na computação sequencial realizada pelos computadores onde cada elemento do vetor x é calculado sequencialmente, começando com x_1 até o x_n. Por isso, no método GS, os elementos anteriores da iteração atual são utilizados para calcular o elemento atual.

$$x^{(i)} = D^{-1}(b + Ex^{(i)} + Fx^{(i-1)}) \tag{5}$$

Em Shang (2009) uma versão distribuída do método de Gauss-Seidel (DGS) foi proposta, onde os dados são divididos em blocos de linhas. Blocos com g linhas são atribuídos a cada processo e a quantidade g de linhas é calculada a partir de uma equação que considera o poder computacional de cada nó do *cluster* utilizado. Na versão DGS o processamento de cada iteração do vetor x é dividido em duas partes. Primeiro, todos os processos calculam paralelamente $z^{(i)} = b + Fx^{(i-1)}$ sobre a sua fatia de dados. Para calcular o restante da equação ($x^{(i)} = D^{-1}(z^{(i)} + Ex^{(i)})$), cada processo deve esperar pelos valores do vetor x da iteração atual calculados nos processos que possuem os índices anteriores do vetor x. O último processo envia o vetor inteiro para o processo inicial caso a solução final seja encontrada. Caso contrário, ele envia o vetor para todos os outros processos para iniciar a próxima iteração.

Os dois próximos métodos, SOR e JOR, utilizam a metodologia de sobre-relaxação. Essa sobre-relaxação é feita através de um balanceamento entre o valor do vetor x da iteração anterior e o calculado na iteração atual. O balanceamento utiliza o parâmetro ω considerando $0 < \omega < 2$. Mais detalhes sobre a convergência do método SOR podem ser visto em Cristina & Cunha (2000). O método SOR é baseado no método GS para calcular o vetor x na iteração atual, como pode ser visto na Eq. (6). Já o método JOR é baseado no método JR para calcular o vetor x, como mostra a Eq. (7).

$$x^{(i)} = (1 - \omega)x^{i-1}_{SOR} + \omega x^{i}_{GS} \tag{6}$$

$$x^{(i)} = (1 - \omega)x^{i-1}_{JOR} + \omega x^{i}_{JR} \tag{7}$$

Em Antuono & Colicchio (2016) foi proposto uma modificação do método JOR atrasando a relaxação, e foi denominado por DOR (*Delayed Over Relaxation*). Neste método o vetor solução x de duas iterações anteriores a iteração atual foi usado para encontrar a solução atual. Assim,

$$x^{(i)} = (1 - \omega)x^{i-2}_{DOR} + \omega x^{i}_{JR}. \tag{8}$$

3. ALGORITMOS IMPLEMENTADOS

Neste trabalho, todos os métodos (JR, GS, DGS, SOR, JOR e DOR) foram implementados na linguagem C que facilita a manipulação de memória, em particular, na alocação dinâmica (Ascencio & de Campos, 2008). Em todos os métodos, uma matriz quadrada cheia (A), de dimensão $n \times n$, vetor b (valores independentes) e x (incógnitas) de de dimensão n, com números de pontos flutuantes com precisão dupla foram considerados.

O algoritmo do método JR pode ser visto no Algoritmo 1. No algoritmo JR, o teste de convergência considera a diferença de cada elemento do vetor `x_prox` em relação ao vetor `x`. Quando a diferença é menor do que o valor de tolerância pré-determinado, a solução é atingida e o processamento é interrompido. A diferença do algoritmo JR para o GS está no segundo *loop* (linha 9). No GS, o *loop* usa o vetor `x_prox` em vez do vetor `x`, sendo então $soma \leftarrow soma + (A[i * tam + j] * x\_prox[j])$.

Os métodos SOR e JOR tem duas fases de processamento a cada iteração. Na primeira fase é calculado o vetor `x_gs` ou `x_jr`, respectivamente, no SOR e no JOR, com o método GS ou o método JR. A segunda fase é responsável por calcular o vetor `x_prox` usando o vetor `x` e o vetor `x_gs` (ou `x_jr`) com o parâmetro ω. O método SOR pode ser visto no Algoritmo 2. Neste algoritmo a variável `w` armazena o valor determinado do parâmetro ω.

Algoritmo 1 Algoritmo Jacobi-Richardson (JR).

para $k \leftarrow 1$ até it_max **faça**
    para $i \leftarrow 0$ até tam **faça**
        x_prox[i] ← B[i]
        soma ← 0
        para $j \leftarrow i + 1$ até tam **faça**
            soma ← soma + (A[i*tam + j] * x[j])
        x_prox[i] ← x_prox[i] + soma
        soma ← 0
        para $j \leftarrow 0$ até i **faça**
            soma ← soma + (A[i*tam + j] * x[j])
        x_prox[i] ← x_prox[i] + soma
        x_prox[i] ← $\frac{x\_prox[i]}{A[i][i]}$
        /* testar convergencia */
    x ← x_prox
    // se convergir - break

Algoritmo 2 Algoritmo SOR.

para $k \leftarrow 1$ até it_max **faça**
    // calcular x_gs
    para $i \leftarrow 0$ até tam **faça**
        x_prox[i] ← (1-w) * x[i] + w *x_gs[i]
    /* testar convergencia */
    x ← x_prox
    // se convergir - break

O método DGS (Shang, 2009) utiliza o paradigma de troca de mensagens para comunicação entre os processos. O autor utilizou a biblioteca de comunicação paralela PVM (*Parallel Virtual Machine*) que foi criada pelo Laboratório Nacional de *Oak Ridge* (Geist et al, 1994). Em nosso trabalho o DGS foi implementado utilizando a biblioteca de troca de mensagens MPI (*Message Passing Interface*) que é considerada padrão, portátil e largamente utilizada por pesquisadores e pela indústria (Gropp et al, 1999).

O método DOR (Antuono & Colicchio (2016)) utiliza duas iterações anteriores do vetor `x` em seu esquema. A primeira iteração deste método usa somente o método JR. A partir da segunda iteração, o vetor `x_prox` é calculado em duas partes, como mostrado no método JOR. O Algoritmo 3 mostra a implementação do método DOR.

Algoritmo 3 DOR

```
/*calcular x_prox com o metodo JR*/
x_ant ← x
x ← x_prox
para k ← 2 até it_max faça
    para i ← 0 até tam faça
        x_jr[i] ← B[i]
        soma ← 0
        para j ← i + 1 até tam faça
            soma ← soma + (A[i*tam + j] * x[j])
        x_jr[i] ← x_jr[i] + soma
        soma ← 0
        para j ← 0 até i faça
            soma ← soma + (A[i*tam + j] * x[j])
        x_jr[i] ← x_jr[i] + soma
        x_jr[i] ← x_jr[i] / A[i][i]
    para i ← 0 até tam faça
        x_prox[i] ← (1-w) * x_ant[i] + w * x_gs[i]
    /* testar convergencia */
    x_ant ← x
    x ← x_prox
    //se convergiu dar break
```

4. TESTES NUMÉRICOS E ANÁLISE DOS RESULTADOS

Em todos os experimentos, foram consideradas matrizes simétricas diagonalmente estritamente dominates geradas aleatoriamente. Para satisfazer essa condição, as matrizes (A) foram geradas a partir da Eq. (9), onde C é uma matriz com números aleatórios de 0 a 999, n é a ordem da matriz A e I é a matriz identidade. Em todos os testes, o sistema linear possui solução trivial. Então, o vetor solução b foi encontrado a partir da multiplicação da matriz A calculada e o vetor $\vec{x} = (1, 1, \ldots, 1)$. Foram geradas matrizes de dimensão 5000, 10000, 15000 e 20000.

$$A = C \times C^T + 0.75 \times n \times I \tag{9}$$

O ambiente de execução utilizado nos experimentos possui um processador Intel Core i7 de 2,80 GHz, com 4 núcleos, 8 GB de memória principal, 8 MB de memória *cache* e sistema operacional Ubuntu versão 14.04. Cada método foi compilado com gcc versão 4.8.4.

4.1 Influência do parâmetro ω no método DOR

Com o intuito de avaliar a influência do parâmetro ω para o método DOR, testes computacionais variando o valor de ω e o tamanho das matrizes são mostrados na Tabela 1.

Table 1- Variação dos valores de ω.

ω	Matriz de ordem 5000	Matriz de ordem 20000
0,85	$k = 6329$	$k = 6381$
0,95	$k = 5167$	$k = 5209$
1,05	$k = 4226$	$k = 4260$
1,50	$k = 1527$	$k = 1537$
1,80	$k = 380$	$k = 383$
1,90	$k = 391$	$k = 393$
1,95	$k = 781$	$k = 796$
2,05	$-$	$-$

Para o nosso sistema modelo foi encontrado resultados que convergiram considerando $\omega < 1$, o que não é previsto na teoria, embora necessite uma quantidade maior de iterações. Porém, não converge para valores maiores que 2 (ver Antuono & Colicchio (2016)). Além disso, foi observado que os melhores resultados foram encontrados com ω entre 1,80 e 1,90. Para se obter um valor experimental mais preciso, um segundo teste foi realizado (Tabela 2) variando ω na segunda casa decimal.

Table 2- Variação dos valores de ω entre 1,80 e 1,90.

ω	Matriz de ordem 5000	Matriz de ordem 20000
1,81	$k = 328$	$k = 332$
1,82	$k = 257$	$k = 258$
1,83	$k = 233$	$k = 238$
1,84	$k = 238$	$k = 241$
1,85	$k = 253$	$k = 296$
1,86	$k = 311$	$k = 313$
1,87	$k = 291$	$k = 293$
1,88	$k = 350$	$k = 352$
1,89	$k = 370$	$k = 372$

O melhor valor encontrado foi $\omega = 1,83$. Um terceira rodada de testes considerando mais uma casa decimal foi realizado e o melhor resultado é $\omega = 1,831$, onde a convergência foi obtida com $k = 225$ para a matriz de dimensão 5000 e $k = 238$ para 20000. O acréscimo de mais casas decimais não altera a quantidade de iterações. Portanto, foi escolhido o valor $\omega = 1,831$ para ser usado nos estudos comparativos descritos a seguir.

4.2 Comparação dos métodos sequenciais

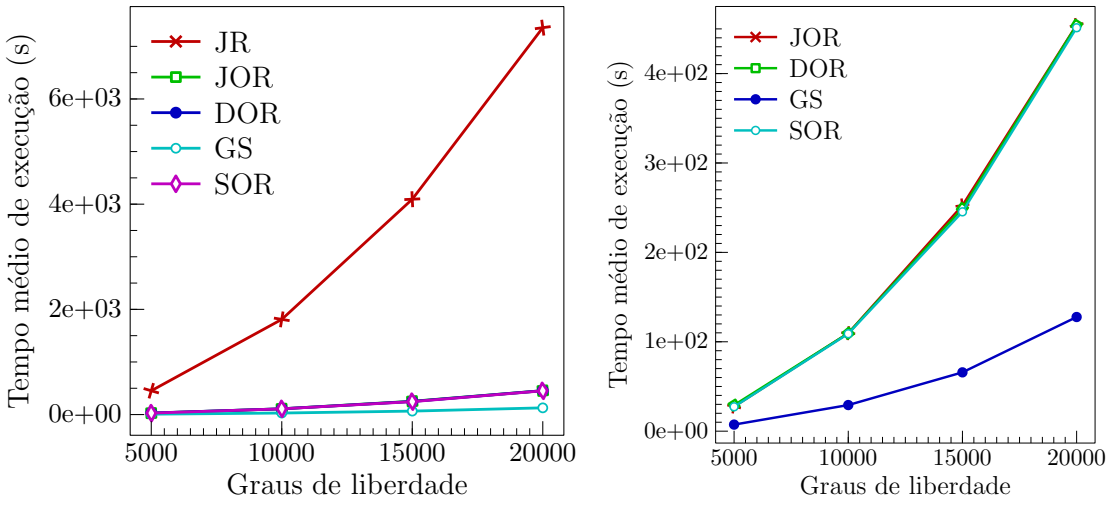
Neste teste comparativo foi utilizado apenas os métodos em suas versões sequenciais. Os resultados das quantidade de iterações (k) necessárias para obtenção da solução podem ser vistas na Tabela 3.

Table 3- Quantidade de iterações para convergência.

ω	$n = 5000$	$n = 10000$	$n = 15000$	$n = 20000$
JR	$k = 4673$	$k = 4676$	$k = 4699$	$k = 4711$
GS	$k = 17$	$k = 17$	$k = 17$	$k = 17$
SOR com $\omega = 0,95$	$k = 215$	$k = 215$	$k = 215$	$k = 215$
JOR com $\omega = 0,95$	$k = 215$	$k = 215$	$k = 215$	$k = 215$
DOR com $\omega = 1,831$	$k = 225$	$k = 226$	$k = 229$	$k = 230$

A Tabela 3 mostra que o método GS converge com o menor número de iterações. Os métodos com relaxação convergem com menor número de iterações se comparados ao método de JR. Além disso, percebe-se que o método DOR aumenta a quantidade de iterações a medida que aumenta a dimensão da matriz, enquanto os métodos JOR e SOR continuam com a mesma quantidade de iterações independentemente da dimensão das matrizes.

Os próximos experimentos foram realizados para obter os tempos de execução de cada método. Todos os métodos foram executados 10 vezes, obtendo-se a média aritmética e o desvio padrão de cada um. As médias podem ser vistas na Figura 1(a) e o valor do desvio padrão é considerado desprezível (inferior a 0,15%).



(a) Todos os métodos.

(b) Sem o método JR.

Figure 1- Estudo comparativo dos tempos de execução dos métodos.

A Figura 1(a) mostra que o método JR tem um tempo superior aos demais métodos. Assim, geramos a Figura 1(b) sem o método JR e com outra escala para observar melhor os outros métodos. O método GS foi o método com o menor tempo de execução. Esse tempo aumenta proporcionalmente ao grau de liberdade do sistema. Outro fato observado é que os métodos com relaxação (JOR, SOR e o mais recente DOR) tem um tempo de execução muito similar

entre eles. Assim, as iterações a mais que o método DOR possui em relação ao JOR não são suficientes para deixar o método mais lento.

4.3 Comparação dos métodos GS e DGS

O último experimento reproduz os resultados apresentados em (Shang, 2009) com a biblioteca de troca de mensagens MPI e o valor do parâmetro g sendo a divisão entre o total de linhas pelos *cores* usados. A versão original é baseada na biblioteca PVM. A versão distribuída foi executada em 4 *cores*.

A Figura 2 mostra os tempos de execução das versões sequencial e distribuída do método Gauss-Seidel. Para a comparação ser a mais precisa possível, o algoritmo da versão sequencial do GS apresentado na seção anterior (o Algoritmo 1 com a modificação na linha 9) teve a computação separada em duas partes. Primeiro o GS calcula $x^{(i)} = b + Fx^{(i-1)}$ para todas as posições de `x_prox` e depois calcula $x^{(i)} = D^{-1}(z^{(i)} + Ex^{(i)})$.

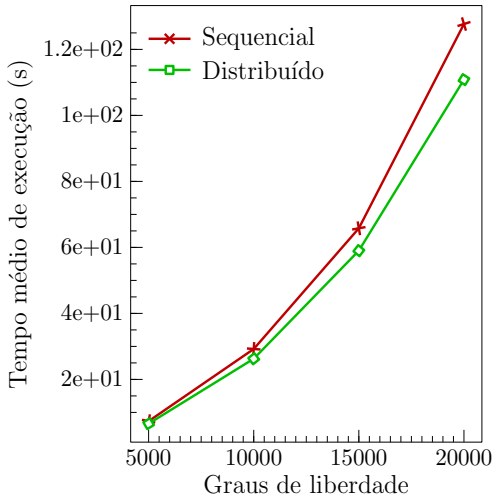


Figure 2- Tempos de execução dos métodos GS (sequencial) e DGS (distribuído).

A versão distribuída do método GS obteve um tempo de execução menor em relação ao sequencial embora o *speedup*, ($s = \frac{t_{seq}}{t_{dis}}$), tenha sido pequeno. Para matrizes de dimensão 5000 a 15000 o *speedup* foi de 1,11 e para a matriz de 20000 o *speedup* foi de 1,15. Como usamos 4 processos (1 para cada núcleo), o *speedup* máximo ideal seria igual a 4. O método GS está distante deste *speedup* porque possui muita comunicação entre os processos. Esse *overhead* prejudica o possível ganho de desempenho.

No artigo de Shang (2009), o *speedup* encontrado foi de 2,08, usando 3 processadores. Shang (2009) divide os dados de entrada em mais do que três blocos, enviando mais de um bloco por processador ciclicamente. Como dito anteriormente, essa divisão em mais de 3 blocos é feita através do parâmetro g e considera o poder de processamento de cada nó do *cluster* usado (ou de cada processador). Como os experimentos neste trabalho foram feitos em uma máquina homogênea, o melhor cenário de divisão dos dados de entrada é a divisão igualitária, para não aumentar *overhead* do tempo de comunicação.

5. CONCLUSÕES

Neste trabalho foi feito um estudo comparativo com diversos métodos iterativos de resolução de sistemas lineares tradicionais e recentes considerando matrizes simétricas diagonalmente dominantes mas geradas aleatoriamente. Implementou-se o método DOR, apresentado em (Antuono & Colicchio, 2016). Provou-se a convergência do método, através de experimentos computacionais, para valores do parâmetro ω entre 0 e 2. Além disso, o tempo de execução do método DOR é similar ao dos métodos JOR e SOR para as matrizes testadas. Mas os melhores resultados foram obtidos com o método GS.

Este trabalho também reproduziu os resultados do artigo (Shang, 2009), do método DGS, com uma biblioteca de troca de mensagens diferente (MPI). Observou-se um ganho de desempenho (*speedup*) de 1,11 com esta versão. Esse ganho é pequeno comparado com o máximo de ganho possível. Porém, este é um método com muita comunicação entre os processos e isso aumenta o *overhead* de tempo da paralelização. Com isso, o método de GS é um método paralelizável, apesar da comunicação excessiva.

Para trabalhos futuros, está planejado a execução de todos os métodos com matrizes maiores e a execução do DGS em um *cluster*.

Acknowledgements

Agradecimentos à Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela suporte financeiro concedido.

REFERENCES

Adsuara, J.E. e Cordero-Carrión, I. e Cerdá-Durán, P. e Mewes, V. e Aloy, M.A. (2017), On the equivalence between the Scheduled Relaxation Jacobi method and Richardson's non-stationary method. Journal of Computational Physics, 332, 446-460.

Antuono, M. e Colicchio, G. (2016), Delayed Over-Relaxation for iterative methods. Journal of Computational Physics, 321, 892-907.

Ascencio, A. F. G. e de Campos, E. A. V. (2008), *"Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ e Java"*, Pearson Prentice Hall, São Paulo.

Cristina, M. e Cunha, C. (2000), *"Métodos Numéricos"*, 1º ed., Editora da UNICAMP, Campinas.

Franco, N. B. (2006), *"Cálculo Numérico"*, 1º ed., Pearson, São Paulo.

Geist, A. e Beguelin, A. e Dongarra, J. e Jiang, W. e Manchek, R. and Sunderam, V. (1994), *"PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing"*, MIT press, Massachusetts.

Gropp, W. D. e Gropp, W. e Lusk, E. e Skjellum, A. (1999), *"Using MPI: portable parallel programming with the message-passing interface"*, 2º ed., MIT press, Massachusetts.

Patterson, D. A. e Hennessy, J. L. (2014), *"Organização e projeto de computadores: interface hardware/software"*, 4º ed., Elsevier Brasil.

Saad, Y. (2003), *"Iterative methods for sparse linear systems"*, 2º ed., Society for Industrial and Applied Mathematics.

Shang, Y. (2009), A distributed memory parallel Gauss-Seidel algorithm for linear algebraic systems. Computers & Mathematics with Applications, 8, 1369-1376.

COMPARATIVE STUDY OF ITERATIVE METHODS OF LINEAR SYSTEMS RESOLUTION

Abstract. *This paper presents a comparative study of some well-known iterative methods for big linear systems resolution and other recent ones. The studied methods are Jacobi-Richardson (JR), Gauss-Seidel (GS), Successive Over-Relaxation (SOR), Jacobi-Richardson Over Relaxation (JOR) and the recent ones, distributed Gauss-Seidel (DGS) and Delayed Over Relaxation for Jacobi-Richardson method (DOR). Randomly Symmetric diagonally dominant matrices were used in all experiments and the DGS method was implemented with the MPI (Message Passing Interface) library for message exchange against the original version implemented with PVM (Parallel Virtual Machine). In this study were observed that for this kind of matrix, the GS method obtained the best results and the speedup obtained with the MPI version of DGS is a bit smaller than the speedup obtained in the original version.*

Keywords: *Linear systems, iterative methods, comparative study*