Resolução de Sistemas Lineares Esparsos

Fernando Barbosa Neto Jeferson de Oliveira Batista

2015

Resumo

Este relatório consiste nos detalhes da implementação em linguagem C do algoritmo de eliminação de Gauss e do algoritmo SOR, ambos utilizando técnicas de armazenamento de matrizes esparsas. A abordagem visa a eficiência geral da resolução do sistema ao tratar a grande quantidade de elementos matriciais nulos.

Palavras-chave: sistema linear esparso. eliminação de Gauss. SOR.

Introdução

A resolução de sistemas lineares é recorrente em diversas áreas do conhecimento. Há várias formas de resolvê-los, através da utilização de métodos diretos, como a eliminação de Gauss, e iterativos, como o SOR (*Sucessive Over Relaxation*). Porém, para sistemas lineares esparsos de alta ordem, pode-se utilizar técnicas que se aproveitam do fato de que a quantidade de elementos não nulos é pequena quando comparada ao sistema como um todo.

O objetivo deste trabalho é implementar, em linguagem C, o algoritmo de eliminação de Gauss e o algoritmo SOR, ambos utilizando técnicas de armazenamento de matrizes esparsas. Logo o foco é armazenar apenas os elementos indispensáveis, ou seja, não-nulos, para a resolução do sistema linear, desprezando assim uma infinidade de coeficientes iguais a zero.

Este relatório explanará nas futuras seções, através do referencial teórico utilizado, a implementação e os experimentos numéricos empregados a fim de obter o melhor resultado, especialmente no quesito performance, dos algoritmos.

1 Referencial Teórico

Este trabalho trata da obtenção de um algoritmo desenvolvido pela linguagem de programação C para a resolução de Sistemas Lineares Esparsos. Visto que os sistemas lineares trabalhados possuem ordem matricial entre 968 e 65025, logo a utilização de uma técnica inocente para a resolução deste sistema não seria adequado dada o possível gasto excessivo de memória e queda de performance.

As formas de resolução abordadas são a Eliminação de Gauss e SOR. A primeira é considerado um método direto que consiste no, ao tratar matricialmente o sistema linear,

escalonamento e, em seguida, substituições sucessivas e possui complexidade O(n^3). A segunda é denominado como um método iterativo que, com o auxílio de um parâmetro de relaxação que deve possuir valores em [0,2[, contribuindo para alcançar uma resposta mais rapidamente ou divergir do resultado esperado, são realizadas iterações até atingir um erro menor que o desejado e possui complexidade O(n^2). Essa base teórica engloba principalmente as aulas ministradas pelo Professor Leonardo Muniz de Lima da disciplina de Algoritmos Numéricos I no período 2015/2 na Universidade Federal do Espírito Santo entre 4 de Agosto de 2015 e 1º de Setembro de 2015. Também pode ser utilizado como material de apoio o livro Algoritmos Numéricos - Frederico Ferreira Campos, LTC, 2002.

Referindo-se à técnica de manipulação da matriz oriunda do sistema linear esparso, há conhecidas formas de compactação como o *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC) ou *Skyline*, porém não foi buscado nenhuma dessas soluções a fim de desenvolver uma técnica própria que vise a melhor perfomance para as matrizes e os métodos de resolução a serem trabalhados. Dessa forma, em primeira instância, o foco utilizado para a compactação é o agrupamento dos elementos pelas suas respectivas colunas, ignorando os elementos nulos. Entretanto, durante a confecção do algoritmo, percebeu-se que o agrupamento por colunas era eficiente para a eliminação Gaussiana, enquanto para o SOR o mais adequado é o agrupamento de linhas. A implementação em termos de programação será descrita em detalhes na seção sobre Implementação.

2 Implementação

A implementação pode ser dividida em cinco partes: armazenamento e compactação matricial, entrada de dados, eliminação de Gauss, SOR e saída de dados. Cada uma dessas partes serão imediatamente explanadas nas subseções conseguintes.

2.1 Armazenamento e compactação matricial

O tipo abstrato de dado (TAD) desenvolvido para o armazenamento e compactação da matriz pode ser encontrada nos arquivos listas.c e listas.h. A ideia inicial é um vetor de ponteiros, onde cada ponteiro é responsável por apontar na memória uma lista. Essas listas são dinamicamente alocadas e representam, cada uma delas, as colunas matriciais, compostas apenas dos elementos diferentes de zero.

O vetor de ponteiros foi declarado como uma *struct*, chamando-a de coluna e definindo seu tipo posteriormente como Coluna. Cada tipo Coluna possui um ponteiro que aponta para o primeiro elemento. Os elementos são constituídos de *struct elemento*, também atribuída definição de Elemento. Cada Elemento possui um inteiro l, representando a linha em que o elemento se encontra na matriz original, uma variável *double* v, esta possuindo o valor do elemento na matriz original, e ponteiros do tipo Elemento que apontam tanto para o elemento anterior como para o próximo. Logo a matriz é, resumidamente, armazenada em um vetor de listas duplamente encadeadas. Oberserve que o próximo elemento do último de uma lista possui valor *NULL* definido em stdlib.h, assim como o anterior do primeiro elemento.

listas.c compreende três funções de manipulação e criação da TAD da matriz. Uma delas possui assinatura *Coluna* cria_matriz(int tam);* que tem por objetivo a inicialização da matriz, cujos ponteiros do tipo Elemento apontam para *NULL*. Outra dessas funções é *void adiciona_elemento(int l, int c, double valor, Coluna *matriz, int tam);*, cujo fim é para adicionar elementos à matriz, mas observando que esta função é tanto utilizada

no processo de criação da matriz quanto durante os passos de resolução da mesma. A outra função, isto é, *void imprime_matriz(Coluna* matriz, int tam);*, auxilia no processo de *debug* ao imprimir a matriz e não possui finalidade prática na resolução dos sistemas lineares diretamente. Outro detalhe a ser notado é que o vetor independente é considerada como uma coluna da matriz, logo as listas 0 a n de Coluna* matriz é a própria matriz focada enquanto a (n+1)-ésima lista refere-se ao vetor de termos independentes.

Apesar do foco inicial ser a implementação de um vetor de ponteiros para as colunas matriciais, ao desenvolver o método SOR percebeu-se que era necessário uma tática mais eficiente para o mesmo. Portanto, ao analisar o algoritmo deste método, foi buscada a implementação de vetor para as linhas matriciais, porém utilizada somente para este método. Para fins práticos, ao invés de criar um novo tipo abstrato de dado, ao adicionar o elemento no vetor de linhas bastava trocar o número da linha pelo número da coluna ao utilizar a função adiciona_elemento, a qual foi explicada anteriormente. Observe que o último ponteiro, isto é, na posição n-ésima, diferentemente das posições 0 a n-1, armazenam o vetor independente, assim como na implementação anterior. Para mais detalhes sobre o desenvolvimento do método SOR favor consultar a subseção 2.4 SOR.

2.2 Entrada de dados

A entrada de dados é tratada nos arquivos main.c, leitura.c e leitura.h.

Ao rodar no terminal o comando da forma *./main algoritmo [tolerancia] [relaxacao] arquivo* conforme determinado na especificação, o arquivo principal (main.c) filtrará e direcionará os argumentos passados pela linha de comando para utilizar as funções correspondentes desejadas.

A leitura da matriz se dá no módulo que corresponde aos arquivos leitura.c e leitura.h através da função de assinatura *Coluna* le_matriz(char* arquivo, int *tam, int sor);*. Tal função receberá como parâmetros o nome do arquivo, a variável que será armazenada a ordem da matriz a ser lida e um inteiro que determina se o método a ser trabalhado é o SOR, a fim de construir corretamente a matriz para o respectivo caso. no caso do parâmetro sor, espera-se que seja setado valor 0 no uso de Gauss e 1 para o SOR. A função le_matriz utiliza subrotinas declaradas no módulo de listas.c e listas.h para a inicialização da matriz e a inserção de seus respectivos elementos. Por fim, le_matriz retorna a matriz lida e pronta para a ser entregue à função indicada de resolução do sistema linear.

2.3 Eliminação de Gauss

O módulo responsável pela solução do sistema linear através da eliminação gaussiana encontra-se nos arquivos gauss.c e gauss.h. Tal solução é obtida com o uso de uma única função, cuja assinatura é *void resolve_gauss(Coluna *matriz, int n);*, onde *matriz* é a matriz armazenada e compactada conforme explicada em 2.1 e *n* é a ordem da matriz original.

O algoritmo se assemelha, estruturalmente falando, a um algoritmo comum de eliminação de Gauss, porém adequada ao tipo de dados em que está sendo trabalhado. *Elemento *atual[n+1]* representa a linha atual que irá escalonar as linhas posteriores, sendo estas controladas por *Elemento *subatual[n+1]*, sendo que ambos ponteiros de índice n trabalham no vetor independente devido à estrutura do TAD criado. Para percorrer as colunas a fim de encontrar um elemento de determinadas linhas, são utilizados *while*s para

iterar entre os elementos a fim de identificar o elemento da linha desejada. Um detalhe incomum decorrente da implementação ocorre na seguinte situação: se a x-ésima linha está escalonando a y-ésima linha e o elemento da posição (a,x) é diferente de zero enquanto o da posição (a,y) é nulo, logo o elemento em (a,y) não estará na matriz compactada. Para resolver este impasse, é invocada a função adiciona_elemento declarada em listas.c, criando um elemento com valor zero (0), prosseguindo em seguida no escalonamento da linha referida.

Após o escalonamento, é utilizado o algoritmo de substituição sucessivas, o qual é deveras semelhante a um algoritmo de substituição sucessivas comuns, porém adequado ao tipo de dado que se está trabalhando. A resposta é armazenada no vetor *double resp[n]*, o qual será direcionado para a função saída_de_dados, declarada em saida.c, escrevendo a respostas no respectivo arquivo de saída.

O tempo é marcado logo antes do escalonamento na eliminação de Gauss e termina logo após as substituições através da função clock_gettime da biblioteca time.h. Para mais informações sobre a marcação do tempo e resultados, ver seção 3 Experimentos Numéricos.

2.4 SOR

O módulo responsável pela solução do sistema linear através do método SOR encontra-se nos arquivos sor.c e sor.h.

Primeiramente, como adiantado nas seções e subseções anteriores, foi utilizado como estratégia inicial a mesma do método de eliminação de Gauss: vetor de listas, as quais representam as colunas da matriz. O algoritmo conforme esta implementação pode ser encontrado na função de assinatura *void resolve_sor(Coluna *matriz, int n, double omega, double toler);*. Os parâmetros são matriz, que é a matriz oriunda do sistema linear, n, a ordem da matriz, omega, que é o parâmetro de relaxação, e toler, este a tolerância que determina o critério de parada consequente do erro normal utilizado. Todavia esta tática possuía uma perfomance decepcionante ao comparada com o método direto. Portanto, ao analisar o algoritmo, percebeu-se que seria mais prático e eficiente utilizar ainda um vetor de listas, porém agora estas representando as linhas da matriz. De qualquer forma, foi mantida essa função derivada de tal implementação no código para fins de curiosidade do leitor.

Para a segunda implementação, foi criada a função *void resolve_sor2(Coluna *matriz, int n, double omega, double toler);*, cujos parâmetros possuem mesmo propósito explanados na implementação anterior. Ambas as implementações utilizam um algoritmo simples para o SOR, portanto apenas traduzido de um pseudocódigo para a linguagem C e, logicamente, com os devidos ajustes para os casos de matrizes esparsas e respectivas implementações. Uma ressalva ao comparar o algoritmo em C com um pseudocódigo é que foi utilizado o aproveitamento de variável, evitando que várias sejam declaradas para um breve fim. Lembre-se que esta agora é a correta implementação para este método.

Em ambas implementações do SOR foi utilizada para critério de parada, a fim de se comparar com a tolerência providenciada, a norma-∞. Como não é definido um número máximo de iterações, pois isso exigiria mais processamento para verificação de mais um critério de parada, é necessário verificar manualmente, por exemplo, inserindo *printf* para ser analisada a variação dos erros em tempo de execução.

O tempo é calculado a partir do início do algoritmo de fato e a contagem é terminada logo após o erro normal ser menor que a tolerância. Para este objetivo, foi utilizada a

função clock_gettime da biblioteca time.h. Para mais informações sobre a marcação do tempo e resultados, ver seção 3 Experimentos Numéricos.

2.5 Saída de dados

A saída dos dados compreende os módulos saida.c e saida.h. É o módulo mais simples e possui apenas a função de assinatura *void saida_de_dados(int n, double resp[]);*, a qual recebe como parâmetros uma variável do tipo inteiro que representa a quantidade de elementos do segundo parâmetro, o qual é um vetor de *double* que armazena as respostas do método utilizado. Os resultados contidos nesse vetor são escritos no arquivo saida_exemplo.txt. Observe que a primeira linha deste arquivo indica o tamanho do vetor-resposta, enquanto as demais linhas corresponde à solução propriamente dita, da variável x(0) até x(n-1).

3 Experimentos Numéricos

A fim de analisar e comparar os diversos modos de resolução, foram utilizadas as seguintes matrizes de teste: rdb968 de ordem 968 com 5632 elementos diferentes de zero, rail_5177 de ordem 5177 com 35185 elementos diferentes de zero, aft01 de ordem 8205 com 125567 elementos diferentes de zero, FEM_3D_thermal1 de ordem 17880 com 430740 elementos diferentes de zero e Dubcova2 de ordem 65025 com 1030225 elementos diferentes de zero.

Para medição e comparação dos tempos do processamentos dos algoritmos de Eliminação de Gauss e SOR, foram utilizados os recursos providos da biblioteca de C chamada time.h, enviando a seguinte mensagem para stdout: *O tempo de execucao do trecho foi t segundos*, onde t é o valor do tempo calculado em segundos. As características em questão de hardware e software do equipamento utilizado para medição temporal é: Notebook Dell Vostro 3460, com Sistema Operacional Ubuntu 15.04 64-bit, memória de 5.7 GiB, processador Intel® Core^{TM} i5-3230M CPU @ 2.60GHz x 4 e placa de vídeo GeForce GT 630M/PCIe/SSE2.

Na Tabela 1 encontram-se os resultados das matrizes-testes para os métodos utilizados. Perceba que n refere à ordem da matriz, enquanto nnz à quantidade de elementos não-nulos e ω o parâmetro de relaxação. Os tempos são padronizados em segundos. Deve-se manter em mente que, para o método SOR, foi utilizada tolerância de 10^{-5}.

Tabela 1 – Resultados

			Eliminação de Gauss	Método Sor		
Matriz	n	nnz	Tempo	Iterações	ω	Tempo
rdb968	968	5632	0.384736	2156	0.000005	0.173919
rail_5177	5177	35185	2615.636530	14418	0.00003	5.742975
aft01	8205	125567	66.859452	1646	1.99	2.642259
FEM_3D_thermal1	17880	430740	140738.037651	29	1.4	0.189317
Dubcova2	65025	1030225	*	402	1.9	16.555322

* = Procedimento foi abortado após 96 horas de processamento contínuo.

A diferença entre tempos nos experimentos da eliminação de Gauss, devido à complexidade, deve-se ao fato de que, para uma dada entrada de tamanho N, ao multiplicar o tamanho de uma entrada por uma constante c, isto é, uma matriz com cN elementos, o

tempo de processamento aumenta por aproximadamente um fator multiplicativo c^3, pois o algoritmo é $O(n^3)$. Entretanto, pode-se perceber que isso não aconteceu ao comparar os casos rail_5177 e aft01, aliás parece que ocorreu o oposto, devido à natureza dessas matrizes: os elementos da aft01 estão muito próximos da diagonal principal, enquanto a rail_5177 possui, além dos elementos da diagonal principal, a primeira linha e a primeira coluna praticamente todas preenchidas com números diferentes de zero. Destarte, para a rail_5177 o escalonamento produzirá diversas chamadas à função adiciona_elemento, declarada em listas.c, causando significativo aumento no tempo de processamento, ao contrário da aft01.

Para o método SOR foram buscados os parâmetros de relaxação que oferecessem, para a tolerância 10^{-5}, aproximadamente o menor tempo de resolução, visto que geralmente há um ω ótimo no quesito de tempo de resolução para uma tolerância dada. Entretanto é possível inferir que, para os casos com baixo parâmetro de relaxação, os resultados tem a sua precisão prejudicada, principalmente quando $\omega << 1$. Pode-se perceber que os casos das duas menores matrizes possuem um fator de relaxação procurado muito pequeno devido à forte inclinação à divergência destes casos, ao contrário dos demais, estes sendo provavelmente diagonal dominante, garantindo a convergência para relaxação próxima de 2. Por mais que a complexidade do algoritmo do SOR seja $O(n^2)$, nota-se que o tempo depende muito do parâmetro de relaxação, da tolerância utilizada e da própria natureza da matriz. Um fato importante a ser observado é que a razão Tempo/Iterações cresce juntamente com a ordem da matriz, como esperado.

Pode-se observar que, apesar do melhor desempenho aparente da Eliminação de Gauss no caso de rdb968 e da pequena matriz armazenada no arquivo exemplo.txt frente ao SOR, é notória a maior rapidez de resolução no caso do SOR do que no do Gauss, mesmo para um erro tolerável relativamente pequeno.

4 Conclusão

Diversas conclusões puderam ser tiradas durante e após a confecção dos algoritmos e suas respectivas implementações. Uma delas foi fundamental para a mudança do rumo do projeto, no caso da implementação por linhas ao invés de colunas para o SOR.

Um ponto cuidadoso que remete a vários possíveis problemas na resolução de um sistema linear esparso é a análise do arranjo dos elementos matriciais. Os dados empíricos dispostos na Tabela 1 da seção imediatamente anterior demonstram isso claramente. Elementos concentrados na diagonal principal, por exemplo, facilitam o processamento e seu tempo tende a diminuir no método de Gauss. Para o SOR, a natureza da matriz, o parâmetro de relaxação e a tolerância são fortes fatores que determinam não somente o tempo de resolução, mas a precisão da mesma e a quantidade de iterações.

De maneira geral, pode-se inferir que, para sistemas lineares sparsos de alta ordem, o SOR produz resultado mais rápido, desde que ajeitados os parâmetros de forma que se adequem à exigência do usuário. Por outro lado, a implementação para resolução por eliminação gaussiana produziu melhores porém pouco diferentes resultados, especialmente em questão de desempenho, ao compará-lo com o SOR.

Sistemas lineares esparsos necessitam de implementações especiais para suas resoluções, mas também é fortemente recomendado a análise da matriz a ser manipulada a fim de se determinar o melhor método e melhores parâmetros. O método Gauss mostrou-se eficiente em vários quesitos mesmo com complexidade $O(n^3)$, porém o SOR pode ser ainda mais

útil dependendo da necessidade, possuindo uma complexidade temporal aproximadamente quadrática.

Resolution of Sparse Linear Systems

Fernando Barbosa Neto Jeferson de Oliveira Batista

2015

Abstract

This essay describes the Gauss elimination and SOR algorithms implementation, that were made in C programming language, using sparse matrix storage techniques. This approach aims the efficiency of solving linear systems whose matrix elements are, most of them, null, ie, zeroed.

Keywords: sparse linear system. Gauss elimination. SOR.