Exercício Programa 2
Métodos iterativos para sistemas lineares: Gradientes Conjugados

Lucas Magno
7994983

Introdução

Este EP consiste em implementar a resolução de sistemas lineares na forma

$$\mathbf{Ax} = \mathbf{b}$$

onde

$$\mathbf{A} \in \mathbf{R}^{n \times n}, \text{ simétrica, definida positiva e esparsa}$$
$$\mathbf{x}, \mathbf{b} \in \mathbf{R}^{n}, \text{ densos}$$

através do método de Gradientes Conjugados.

Motivação

Por que utilizar matrizes esparsas? A necessidade de matrizes esparsas fica mais clara quando se lida com matrizes gigantes, de forma que armanezar a matriz toda na memória não é uma possibilidade. No entanto, em diversas aplicações (diferenças finitas para resolução de equações diferenciais, por exemplo) surgem matrizes bem estruturadas, com apenas alguns elementos não-nulos. Assim, guardando apenas estes elementos, se torna possível manipular matrizes ordens de grandezas maiores do que caberia na memória de um computador.

Por outro lado, embora economizem muita memória dependendo da densidade de elementos não nulos, geralmente o acesso aos elementos de uma matriz esparsa é muito mais caro do que aos de uma densa (que é $O(1)$), já que estes são armazenados utilizando estruturas que não permitem o acesso direto a um elemento (como uma lista ligada). Este acesso pode ser feito barato, a custo da eficiência de se iterar sobre as linhas ou colunas da matriz, sendo difícil conciliar o desempenho dessas duas operações.

Por esses motivos, os métodos padrões de resolução de sistemas lineares (LU, Cholesky) não são eficazes para matrizes esparsas, pois além de acessarem diretamente elementos da matriz, também podem estragar sua esparsidade ao decompô-la em outras matrizes.

É aí que entra o método de Gradientes Conjugados, uma vez que ele depende somente do produto matriz-vetor entre a matriz esparsa e vetores densos, o que pode ser feito eficaz facilmente, escolhendo uma estrutura adequada. Além disso, como não altera a matriz original, nem a decompõe em novas, não altera sua esparsidade, mantendo a eficiência de seu uso.

Matrizes Esparsas

Há diversas implementações comuns de matrizes esparsas, com suas vantagens e desvantagens, mas aqui vamos focar nas duas utilizadas neste EP.

Coordinate List (COO)

É uma forma simples de matriz esparsa, composta por uma cadeia ordenada de tuplas na forma (linha, coluna, valor). Neste trabalho foi implementada utilizando uma lista ligada, tornando simples a inserção de elementos novos mantendo a ordenação escolhida (por colunas e então por linhas). Entretanto, obter uma coluna específica de uma matriz neste formato não é muito eficiente, então ele foi utilizado somente para construção da matriz esparsa e então traduzido para um formato mais adequado.

Compressed Sparse Column (CSC)

Mais complexo do que o formato COO, o CSC é composto principalmente por três vetores

- **nzval**: contém os valores de todos os elementos não-nulos, ordenados por coluna e então por linha
- **rowval**: contém os valores dos índices de linha de cada elemento correspondente em `nzval`
- **colptr**: contém os índices de `nzval` em que se encontram os primeiros elementos de cada coluna da matriz. Por exemplo, `nzval(colptr(j))` representa o valor do primeiro elemento da coluna `j` na matriz.

Vale notar que `nzval` e `rowval` são obtidos diretamente a partir de uma matriz no formato COO, sendo necessário apenas o cálculo de `colptr`. A vantagem de se utilizar este formato está na facilidade de se obter uma coluna da matriz (a coluna `j` são os elementos em `nzval` entre `colptr(j)` e `colptr(j+1)-1`), o que torna barato o produto matriz-vetor orientado a colunas, mais especificamente, da ordem do número do elementos não nulos da matriz, já que basta iterar sobre `nzval`.

Gradientes Conjugados

Como $\mathbf{A} \in \mathbf{R}^{n \times n}$ é uma matriz definida positiva, vale que

$$\mathbf{x^T A x} \geq \mathbf{0}$$

onde a igualdade vale se e somente se $\mathbf{x} = \mathbf{0}$. Assim, $\mathbf{A}$ define uma norma e portanto um produto interno. Podemos então definir que dois vetores $\mathbf{u}$ e $\mathbf{v}$ são ortogonais se e somente se

$$\mathbf{u^T A v} = \mathbf{0}$$

Tendo isso, podemos pegar um conjunto P de n vetores $\mathbf{p_i}$ ortogonais entre si, que então formam uma base do $\mathbf{R}^n$ e podemos escrever qualquer vetor $\mathbf{x} \in \mathbf{R}^n$ como uma combinação linear dos $\mathbf{p_i}$

$$\mathbf{x} = \sum_i \alpha_i \mathbf{p_i}$$

onde os α_i são as projeções de $\mathbf{x}$ em $\mathbf{p_i}$:

$$\alpha_i = \frac{\mathbf{p_i A x}}{\mathbf{p_i^T A p_i}} = \frac{\mathbf{p_i b}}{\mathbf{p_i^T A p_i}}$$

Então bastaria escolher os $\mathbf{p_i}$ e calcular as projeções. No entanto, mesmo isso pode ser custoso, pois estamos lidando com n muito grande. Na verdade, podemos escolher os $\mathbf{p_i}$ de forma que apenas alguns deles sejam necessários para uma boa aproximação de $\mathbf{x}$, então é possível construir um algoritmo iterativo, que se aproxima da solução verdadeira e a alcançaria em n passos, se fosse utilizada precisão infitita, uma vez que isso representa a soma de todas as projeções, que é a identidade.

Definindo $\mathbf{x}_\star$ como a solução de $\mathbf{A}\mathbf{x}_\star = \mathbf{b}$, queremos então minimizar o erro definido por

$$\mathbf{e} = \mathbf{x}_\star - \mathbf{x}$$

mas não temos como calcular $\mathbf{e}$, pois não possuímos $\mathbf{x}_\star$. Entretanto, vale que

$$\|\mathbf{e}\|_A \propto \frac{1}{2}\mathbf{x^T A x} - \mathbf{x^T b}$$

Como queremos minimizar essa função, poderíamos escolher as projeções (que podem ser vistas como direções de passo) no sentido contrário de seu gradiente (que é $\mathbf{Ax} - \mathbf{b}$), mas também queremos manter os $\mathbf{p_i}$ ortogonais entre si. Logo, se definirmos $\mathbf{r_i} = \mathbf{b} - \mathbf{Ax_i}$ (o resíduo no passo i), podemos definir

$$\mathbf{p_i} = \mathbf{r_i} - \sum_{k<i} \frac{\mathbf{p_k^T A r_i}}{\mathbf{p_k^T A p_k}} \mathbf{p_k}$$

Explorando as ortogonalidades entre $\mathbf{p}$, $\mathbf{r}$ e $\mathbf{x}$, pode-se mostrar que

$$\begin{aligned}
\mathbf{x_{i+1}} &= \mathbf{x_i} + \alpha_i \mathbf{p_i} \\
\mathbf{r_{i+1}} &= \mathbf{r_i} - \alpha_i \mathbf{A p_i} \\
\mathbf{p_{i+1}} &= \mathbf{r_i} + \beta_i \mathbf{p_i}
\end{aligned}$$

onde

$$\begin{aligned}
\alpha_i &= \frac{\mathbf{r_i^T r_i}}{\mathbf{p_i^T A p_i}} \\
\beta_i &= \frac{\mathbf{r_{i+1}^T r_{i+1}}}{\mathbf{r_i^T r_i}}
\end{aligned}$$

Portanto, o algoritmo fica

Algorithm 1 Gradientes Conjugados

$\mathbf{r_0} \leftarrow \mathbf{b} - \mathbf{Ax_0}$
$\mathbf{p_0} \leftarrow \mathbf{r_0}$
$i \leftarrow 0$
loop
  $\alpha_i \leftarrow \frac{\mathbf{r_i \cdot r_i}}{\mathbf{p_i \cdot A p_i}}$
  $\mathbf{x_{i+1}} \leftarrow \mathbf{x_i} + \alpha_i \mathbf{p_i}$
  $\mathbf{r_{i+1}} \leftarrow \mathbf{r_i} - \alpha_i \mathbf{A p_i}$
  if $\sqrt{\mathbf{r_{i+1} \cdot r_{i+1}}} < \epsilon$ **then**
    exit
  end if
  $\beta_i \leftarrow \frac{\mathbf{r_{i+1} \cdot r_{i+1}}}{\mathbf{r_i \cdot r_i}}$
  $\mathbf{p_{i+1}} \leftarrow \mathbf{r_{i+1}} + \beta_i \mathbf{p_i}$
  $k \leftarrow k + 1$
end loop
  return $\mathbf{x_{i+1}}$

em que a matriz $\mathbf{A}$ só aparece em um produto matriz-vetor.

O Programa

Para testar o algoritmo, foram geradas matrizes esparsas aleatórias utilizando o método descrito em [1]

Resultados

O programa foi executado

Referências

[1] Lloyd N. Trefethen David Bau, *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, 1997.