Lista de Fevereiro

Natan Ledur

31 de março de 2023

Visite o diretório no GitHub para ter acesso aos arquivos usados neste presente trabalho.

1 Exercício

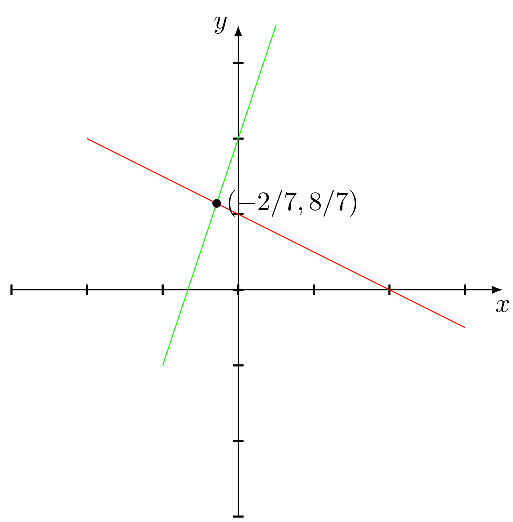
$$A = \begin{cases} x + 2y = 2 \\ -3x + y = 2 \end{cases} \quad B = \begin{cases} 5x - y = 10 \\ -\frac{1}{5} + \frac{y}{25} = -\frac{2}{5} \\ 3x + 4y = 2 \end{cases} \quad C = \begin{cases} x + y - z = 2 \\ -x - y + z = 0 \\ x + 2y - 3z = 4 \end{cases}$$

- Para o sistema A temos;

$$A = \begin{pmatrix} 1 & 2 & 2 \\ -3 & 1 & 2 \end{pmatrix} \xrightarrow{L_2 = L_2 - (-3)L_1} \begin{pmatrix} 1 & 2 & 2 \\ 0 & 7 & 8 \end{pmatrix}$$

$$A = \begin{cases} x + 2y = 2 \\ 7y = 8 \end{cases} \tag{1}$$

Resolvendo o sistema (1) temos, $\left[\left[y = \frac{8}{7}, x = -\frac{2}{7}\right]\right]$, com isso vemos que o sistema A é consistente.



- Para o sistema B temos;
  É possível notar que no sistema B a segunda equação é a primeira equação multiplicado por $-\frac{1}{25}$, logo podemos analisar uma das duas com a ultima equação e teremos nosso resultado.

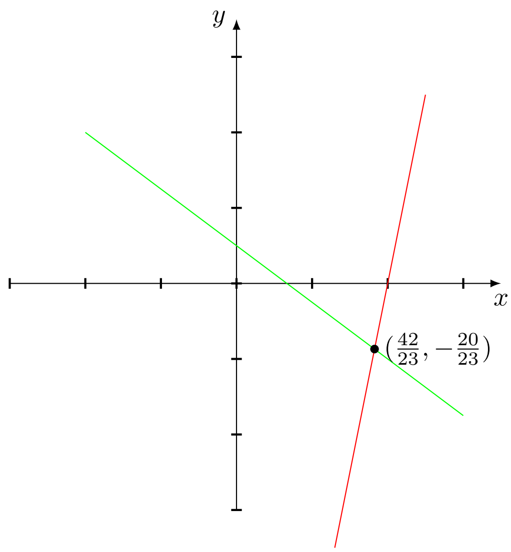
$$\begin{pmatrix} 5 & -1 & 10 \\ -\frac{1}{5} & \frac{1}{25} & -\frac{2}{5} \\ 3 & 4 & 2 \end{pmatrix} \xrightarrow{L_3 = L_3 - (\frac{3}{5})L_1} \begin{pmatrix} 5 & -1 & 10 \\ -\frac{1}{5} & \frac{1}{25} & -\frac{2}{5} \\ 0 & \frac{23}{5} & -\frac{20}{5} \end{pmatrix}$$

Obtemos assim um sistema mais simples de se resolver:

$$B = \begin{cases} 5x - y = 10 \\ -\frac{1}{5} + \frac{y}{25} = -\frac{2}{5} \\ \frac{23}{5}y = -\frac{20}{5} \end{cases} \tag{2}$$

Resolvendo o sistema (2) temos que, $[[y = -\frac{20}{23}, x = \frac{42}{23}]]$, com isso vemos que o sistema B é consistente.

Podemos analisar também através do gráfico das funções, que o ponto de intercessão é $y = -\frac{20}{23}, x = \frac{42}{23}$.



- Para o sistema C temos;

$$\begin{pmatrix} 1 & 2 & -1 & 2 \\ -1 & -1 & 1 & 0 \\ 1 & 2 & -3 & 4 \end{pmatrix} \xrightarrow{L_2 = L_2 - (-1)L_1} \begin{pmatrix} 1 & 2 & -1 & 2 \\ 0 & 0 & 0 & 2 \\ 1 & 2 & -3 & 4 \end{pmatrix}$$

Resultando na matriz

$$C = \begin{cases} x + y - z = 2 \\ 0 = 2 \\ x + 2y - 3z = 4 \end{cases} \tag{3}$$

Como $0 = 2$ é um absurdo o sistema (3) não pode ser resolvido logo o sistema C não é consistente. Uma forma mais rápida de ver que o sistema não é consistente e ver que a primeira e a segunda equação do sistema são planos paralelos, sendo assim não possuem interseção.

2 Exercício

Listing 1: Métodos de eliminação de Gauss e Gauss com pivotamento.

```
function gausspivo(A, b) {
  const n = A.length;
  const Ab = new Array(n);


  for (let i = 0; i < n; i++) {
    Ab[i] = [...A[i], b[i]];
  }

  for (let i = 0; i < n; i++) {

    let max = Math.abs(Ab[i][i]);
    let maxIndex = i;
```

```
    for (let j = i + 1; j < n; j++) {
      if (Math.abs(Ab[j][i]) > max) {
        max = Math.abs(Ab[j][i]);
        maxIndex = j;
      }
    }


    if (maxIndex !== i) {
      [Ab[i], Ab[maxIndex]] = [Ab[maxIndex], Ab[i]];
    }


    for (let j = i + 1; j < n; j++) {
      const factor = Ab[j][i] / Ab[i][i];
      for (let k = i; k <= n; k++) {
        Ab[j][k] -= factor * Ab[i][k];
      }
    }
  }


  const x = new Array(n);
  for (let i = n - 1; i >= 0; i--) {
    let sum = 0;
    for (let j = i + 1; j < n; j++) {
      sum += Ab[i][j] * x[j];
    }
    x[i] = (Ab[i][n] - sum) / Ab[i][i];
  }

  return x;
}
function gauss(A, b) {
  const n = A.length;
  const Ab = new Array(n);


  for (let i = 0; i < n; i++) {
    Ab[i] = [...A[i], b[i]];
  }


  for (let i = 0; i < n - 1; i++) {
    for (let j = i + 1; j < n; j++) {
      const factor = Ab[j][i] / Ab[i][i];
      for (let k = i; k <= n; k++) {
        Ab[j][k] -= factor * Ab[i][k];
      }
    }
  }


  const x = new Array(n);
  for (let i = n - 1; i >= 0; i--) {
    let sum = 0;
```

```
    for (let j = i + 1; j < n; j++) {
      sum += Ab[i][j] * x[j];
    }
    x[i] = (Ab[i][n] - sum) / Ab[i][i];
  }

  return x;
}
function decomposeLU(A) {
  const n = A.length;
  const L = new Array(n).fill(null).map(() => new Array(n).fill(0));
  const U = new Array(n).fill(null).map(() => new Array(n).fill(0));

  // Inicializa L com a diagonal principal igual a 1
  for (let i = 0; i < n; i++) {
    L[i][i] = 1;
  }


  for (let i = 0; i < n; i++) {

    let maxRow = i;
    for (let j = i + 1; j < n; j++) {
      if (Math.abs(A[j][i]) > Math.abs(A[maxRow][i])) {
        maxRow = j;
      }
    }


    [A[i], A[maxRow]] = [A[maxRow], A[i]];
    [L[i], L[maxRow]] = [L[maxRow], L[i]];


    for (let j = i + 1; j < n; j++) {
      const factor = A[j][i] / A[i][i];
      L[j][i] = factor;
      for (let k = i; k < n; k++) {
        A[j][k] -= factor * A[i][k];
      }
    }
  }


  for (let i = 0; i < n; i++) {
    for (let j = i; j < n; j++) {
      U[i][j] = A[i][j];
    }
  }

  return { L, U };
}

const A = [[0.4096, 0.1234, 0.3678,0.2943],
           [0.2246, 0.3872, 0.4015,0.1129],
           [0.3645, 0.1920, 0.3781,0.0643],
           [0.1784,0.4002,0.2786,0.3927]]
```

```
const b = [0.4043, 0.1550, 0.4240,0.2557]
const x = gauss(A, b);
console.log(x); // [ 2, 1, -1 ]
const a = gausspivo(A, b);
console.log(a); // [ 2, 1, -1 ]
const { L, U } = decomposeLU(A);
console.log("L = ", L);
console.log("U = ", U);
```

Listing 2: Execussão do codido

```
node main.js
[3.4605863893500874,
1.56095288560023,
-2.934233975722392,
-0.4300595137280605]

[3.4605863893500883,
1.560952885600231,
-2.9342339757223943,
-0.4300595137280602]


L =  [[ 1, 0, 0, 0 ],
     [ 0.435546875, 0, 0, 1 ],
     [ 0.54833984375, 0.9223022681839759, 0, 0 ],
     [ 0.8898925781249999, 0.23722448222558723, 0.2506079359717326, 0 ]]

U =  [[ 0.4096, 0.1234, 0.3678, 0.2943 ],
     [ 0, 0.346453515625, 0.118405859375, 0.26451855468749996 ],
     [ 0, 0, 0.09061461280091464, -0.2924424789806533 ],
     [ 0, 0, 0, -0.18705725686919206 ]]
```

- Para a matriz:

$$A = \begin{pmatrix} 1 & 1 & 2 \times 10^9 \\ 1 & -1 & 10^9 \\ 2 & 2 & 0 \end{pmatrix}, b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Temos o seguinte resultado no console

Listing 3: Resoltado

```
 [ 0.74951171875, -0.24951171875000003, 9.765625e-13 ]
 [ 0.74951171875, -0.24951171875, 9.765625e-13 ]
L =  [ [ 0, 0, 1 ], [ 0.5, 1, 0 ], [ 0.5, -0, 0 ] ]
U =  [ [ 2, 2, 0 ], [ 0, -2, 1000000000 ], [ 0, 0, 512000000000 ] ]
```

Como podemos ver, os resultados encontrados pelos dois métodos são diferentes. Além disso, esses resultados não resolvem o problema inicial, pois os valores encontrados para x têm magnitude muito grande, o que sugere uma perda significativa de precisão.

3 Execício

Considerado a matriz triangular simétrica:

$$A = \begin{pmatrix} 5 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 5 & -1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 5 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 5 \end{pmatrix}$$

- Segue a função em JavaScript para a decomposição de Cholesky

Listing 4: Deconposição de Cholesky

```
unction cholesky(A) {
  const n = A.length;
  const L = Array(n).fill().map(() => Array(n).fill(0));

  for (let i = 0; i < n; i++) {
    for (let j = 0; j <= i; j++) {
      let sum = 0;
      for (let k = 0; k < j; k++) {
        sum += L[i][k] * L[j][k];
      }

      if (i === j) {
        L[i][j] = Math.sqrt(A[i][i] - sum);
      } else {
        L[i][j] = (A[i][j] - sum) / L[j][j];
      }
    }
  }

  return L;
}
```

- Para obter a matriz G para A de ondem 4×4, 5×5 e temos:

Listing 5: Resuldado 4×4

```
[ 2.23606797749979, 0, 0, 0 ],
[ -0.4472135954999579, 2.1908902300206643, 0, 0 ],
[ 0, -0.45643546458763845, 2.188987589427283, 0 ],
[ 0, 0, -0.45683219257612856, 2.188904828407596 ]
```

Listing 6: Resuldado 5×5

```
[ 2.23606797749979, 0, 0, 0, 0 ],
[ -0.4472135954999579, 2.1908902300206643, 0, 0, 0 ],
[ 0, -0.45643546458763845, 2.188987589427283, 0, 0 ],
[ 0, 0, -0.45683219257612856, 2.188904828407596, 0 ],
[ 0, 0, 0, -0.45684946509414437, 2.188901223500776 ]
```

- Para resolver este pontos vamos criar uma função que nos possibilite a criação de uma matriz $n \times n$ de forma automática;

Listing 7: Matriz $n \times n$

```
function createTridiagonalMatrix(n) {
  // Inicializa a matriz com todos os elementos iguais a 0
  let matrix = Array(n).fill().map(() => Array(n).fill(0));

  // Adiciona os elementos da diagonal principal
  for (let i = 0; i < n; i++) {
    matrix[i][i] = 5;
  }

  // Adiciona os elementos nas diagonais superior e inferior
  for (let i = 0; i < n - 1; i++) {

    if (matrix[i][i + 1] !== -1) {
      matrix[i][i + 1] = -1;
    }
    if (matrix[i + 1][i] !== -1) {
      matrix[i + 1][i] = -1;
    }
  }

  return matrix;
}
```

A decomposição de Cholesky é uma técnica eficiente para resolver sistemas lineares simétricos e definidos positivos, mas seu custo computacional pode ser alto para matrizes muito grandes. Além disso, a complexidade do algoritmo cresce de forma quadrática com o tamanho da matriz.