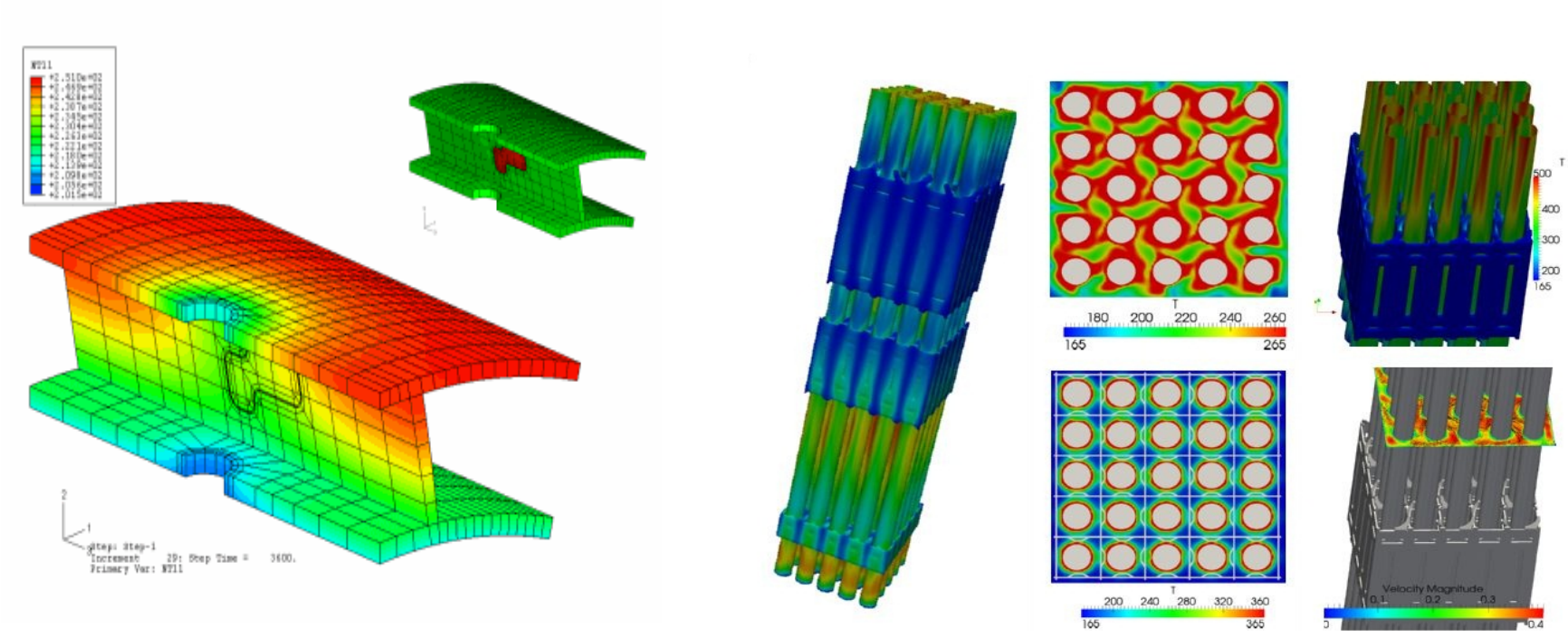
SME0305 - 2016

Roberto F. Ausas / Gustavo C. Buscaglia

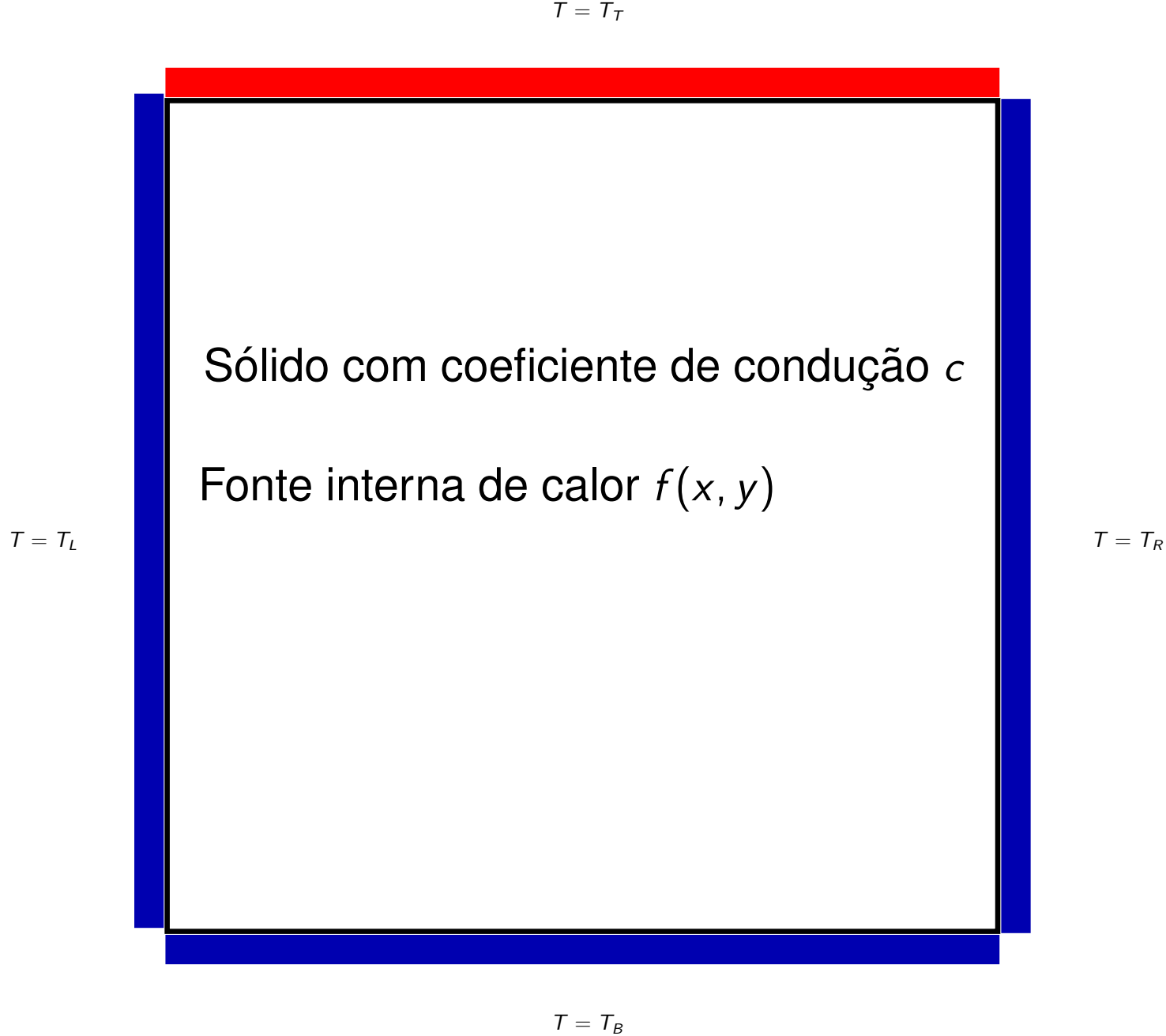
ICMC - Ramal 736628, rfausas@gmail.com
ICMC - Ramal 738176, gustavo.buscaglia@gmail.com

Transferência de Calor

Vamos estudar métodos numéricos para resolver o problema de achar a distribuição de temperaturas num sólido:



Para começar simples, vamos aprender resolver o problema num dominio de forma quadrada de lado *H*:


$T = T_T$
Sólido com coeficiente de condução c
Fonte interna de calor $f(x, y)$
$T = T_L$
$T = T_R$
$T = T_B$

Vamos supor que conhecemos:

- Um coeficiente de condução térmica, que denotaremos com a letra c.

  O coeficiente de condução é uma propriedade de cada material. Valores médios para alguns materiais são:

Material	Coeficiente c $[\ \mathrm{W} \cdot \mathrm{m}^{-1} \cdot \mathrm{K}^{-1}\ ]$
Aluminio	230
Aço	20
Concreto	1.5
Plástico isolante	0.03
Ar	0.02

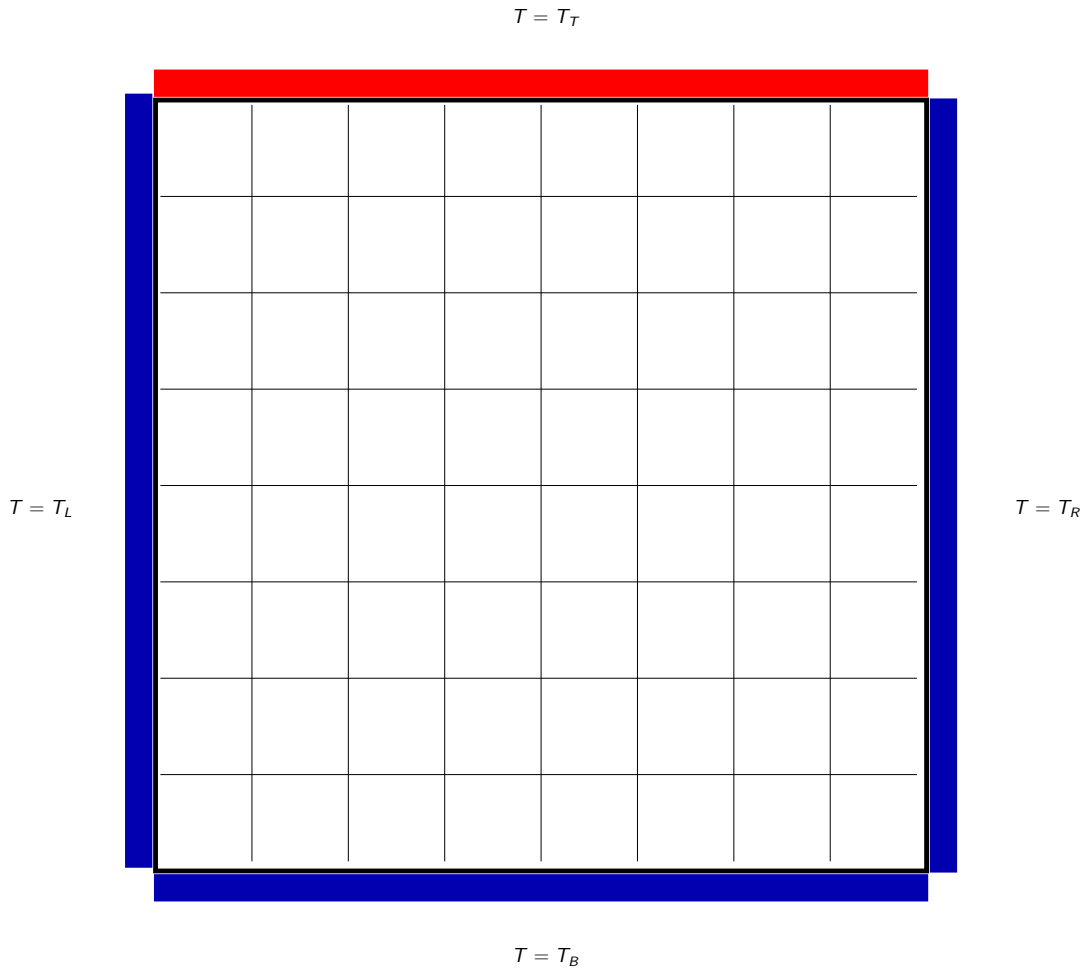
- Uma fonte de calor interna, que denotaremos com a letra f. Por exemplo, pode ser uma reação química, nu-

clear ou devido a circulação de uma corrente elétrica. As unidades da fonte em SI seriam $W \cdot m^{-3}$.

- A temperatura fixada em cada cara, que denotaremos com as letras T_B, T_T, T_L e T_R.

Agora, queremos formular o problema de condução de calor no sólido. Vamos ver que isto nos leva a um problema linear que precisa ser resolvido utilizando alguns dos métodos numéricos (diretos ou iterativos) que já estudamos.

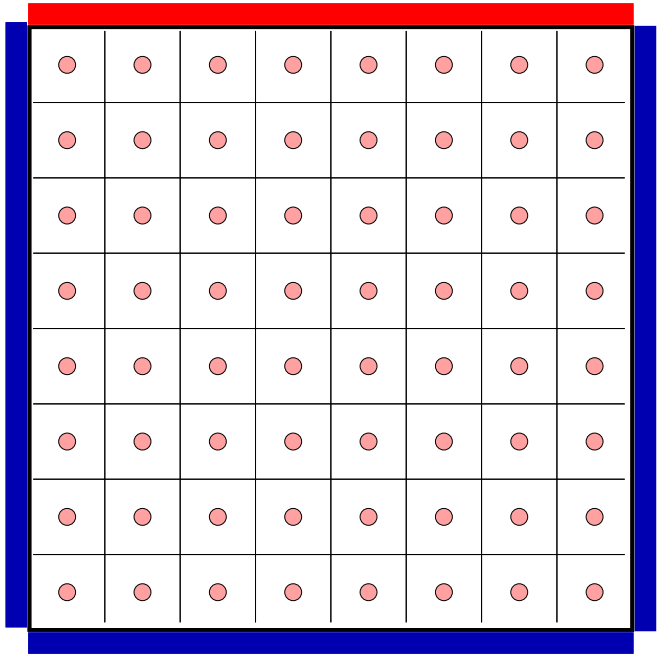
Vamos **discretizar** o dominio computacional. Isto significa, dividir ele em pedaços ou células de cálculo, que por simplicidade vamos asumir todas do mesmo tamanho:



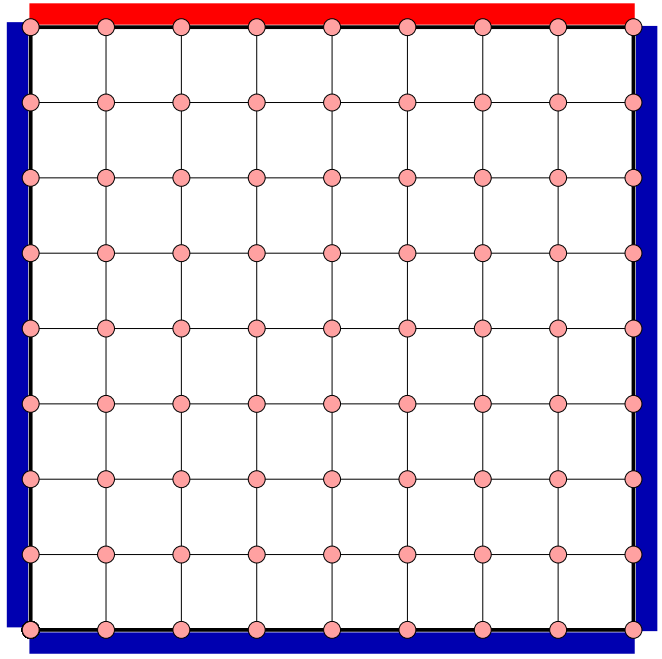
Quanto mais células colocarmos, mais precisas serão as nossas predições, porém, os cálculos serão mais costosos computacionalmente como veremos.

Neste ponto há duas formas de trabalhar

- Centrados nas célula (cell-centered);
- Centrados nos vértices (vertex-centered);



Centrado na celula

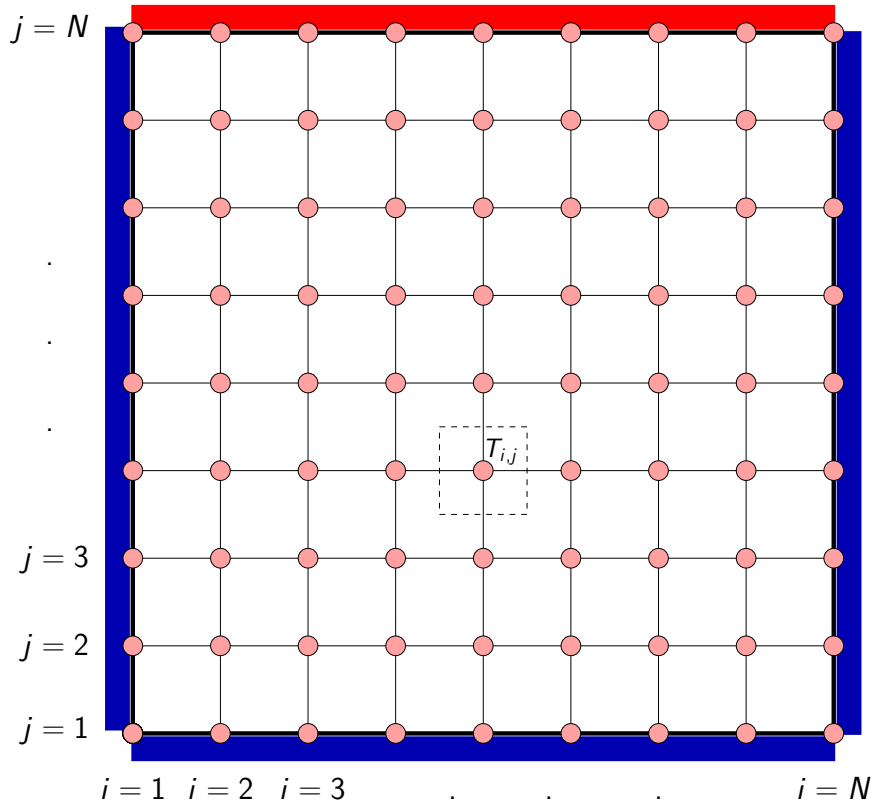


Centrado no vértice

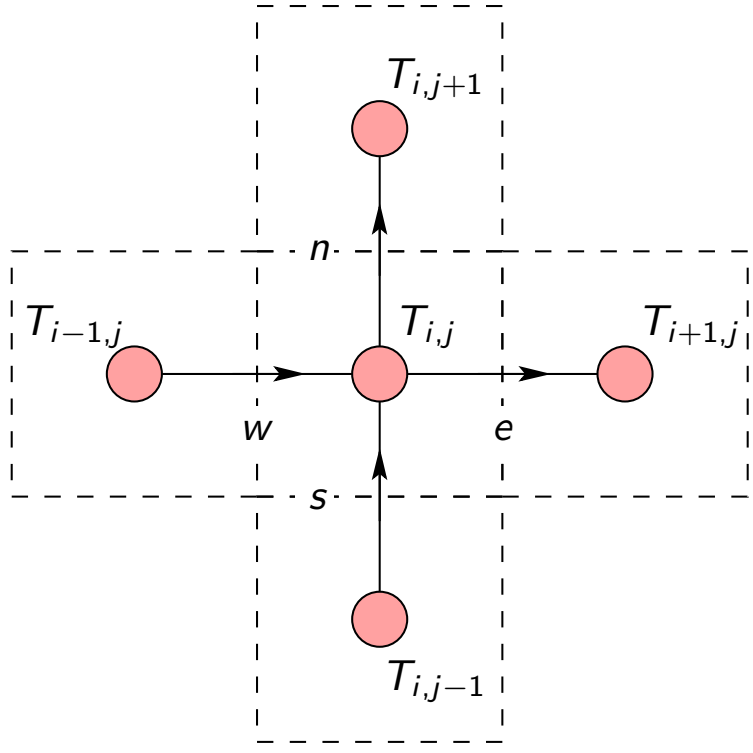
A cada pontinho rosa associamos uma temperatura T, que é representativa da temperatura no sólido ao redor desse ponto.

Vamos adotar a segunda forma de trabalho pois é mais prática para colocar as condições de temperatura fixa (as chamadas condições de contorno) nas caras do dominio.

Vamos indexar as células com um índice horizontal e um vertical:



Neste caso, se pegarmos qualquer ponto dentro do dominio, teremos:



Por enquanto, vamos supor que a geração interna de calor é nula. Vamos fazer o balanço de troca de calor entre essas células:

Balanço de Energia: Todo o calor que está entrando o saindo de uma célula tem que somar zero.

Uma aproximação muito razoável que pode ser feita, é assumir que o calor indo desde a célula (i, j) às outras células vizinhas é

$$\begin{aligned} q_e &= c\,(T_{i,j} - T_{i+1,j}) \\ q_w &= c\,(T_{i,j} - T_{i-1,j}) \\ q_n &= c\,(T_{i,j} - T_{i,j+1}) \\ q_s &= c\,(T_{i,j} - T_{i,j-1}) \end{aligned}$$

Isto significa que:

$$c\,(T_{i,j} - T_{i+1,j}) + c\,(T_{i,j} - T_{i-1,j}) + c\,(T_{i,j} - T_{i,j+1}) + c\,(T_{i,j} - T_{i,j-1}) = 0$$

ou re-acomodando os termos:

$$c\,(-T_{i+1,j} - T_{i-1,j} + 4\,T_{i,j} - T_{i,j+1} - T_{i,j-1}) = 0$$

O que significa que a temperatura num ponto é a média das temperaturas nos seus vizinhos:

$$T_{i,j} = \frac{T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1}}{4}$$

Mas, nos não conhecemos a temperatura! Isso é justamento o que queremos calcular.

Partindo da fórmula de cima, poderiamos propor um método iterativo para calcular $T_{i,j}$

Dada uma condição inicial $T_{i,j}^{(0)}, \quad i,j = 2, \ldots, N-1,$

Para $k = 1, \ldots, MAX\_IT$

$$T_{i,j}^{(k)} = \frac{T_{i+1,j}^{(k-1)} + T_{i-1,j}^{(k-1)} + T_{i,j+1}^{(k-1)} + T_{i,j-1}^{(k-1)}}{4}$$

Podemos ver que assim implementado, este é um método sem matriz ou *matrix-free*.

A implementação em Octave seria:

```
k=1;
Told = zeros(N,N);
for k=1:100000 %Loop de iteracoes
  for i=2:N-1 %Loop nos horizontais
    for j=2:N-1 %Loop nos verticais
      Tnew(i,j) = 0.25 * (...
                          Told(i-1,j) + Told(i+1,j) + ...
                          Told(i,j-1) + Told(i,j+1) ...
                          );
    end
  end
  if(norm(Tnew-Told,inf,'rows') < 1.0e-8) break ; end
  Told = Tnew;
end
```

Isto não é outra coisa que uma relaxão das que temos estudado para resolver o sistemas de massa e mola!

Mas, isto está certo? Está faltando alguma coisa?

Mas, isto está certo? Está faltando alguma coisa?

Está faltando colocar as condições de contorno: Vamos colocar todas as paredes a temperatura zero ($T_B = T_L = T_R = 0$) exceto a parede superior que vamos colocar a temperatura ambiente $T_T = 20$. Antes de iniciar o laço de iterações fazemos:

```
Told = zeros(N,N);
for j=1:N
  Told(1,j) = 0.0;
  Told(N,j) = 0.0;
end
for i=1:N
  Told(i,1) = 0.0;
  Told(i,N) = 20.0;
end
```

Na tabela da sequência podemos ver como se comporta o método:

N	# incognitas	# iterações
11	81	350
21	361	1302
41	1521	4772

(Nesta tabela consideramos o número de incógnitas internas unicamente)

Exo. 1: Implementar em Octave a forma vetorizada e portanto mais eficiente do método de Jacobi. Nesta implementação, os dois laços internos sobre `i` e `j` desaparecem. Usando os comandos `tic` e `toc`, comparar o tempo de cálculo para diferentes valores de N.

Podemos ver como se comporta o método de Gauss-Seidel neste caso, que consistiría em utilizar o último valor cálculado de temperatura na fórmula de iteração. A implementação matrix-free seria:

```
k=1;
Tnew = zeros(N,N);
for k=1:100000 %Loop de iteracoes
  for i=2:N-1 %Loop nos horizontais
    for j=2:N-1 %Loop nos verticais
      Tnew(i,j) = 0.25 * (...
                          Tnew(i-1,j) + Tnew(i+1,j) + ...
                          Tnew(i,j-1) + Tnew(i,j+1) ...
                          );
    end
  end
    if(norm(Tnew-Told,inf,'rows') < 1.0e-8) break ; end
  Told = Tnew;
  k = k + 1;
end
```

N	# incognitas	# iterações Jacobi	# iterações G-S
11	81	350	185
21	361	1302	684
41	1521	4772	2509

(Nesta tabela consideramos o número de incógnitas internas unicamente)

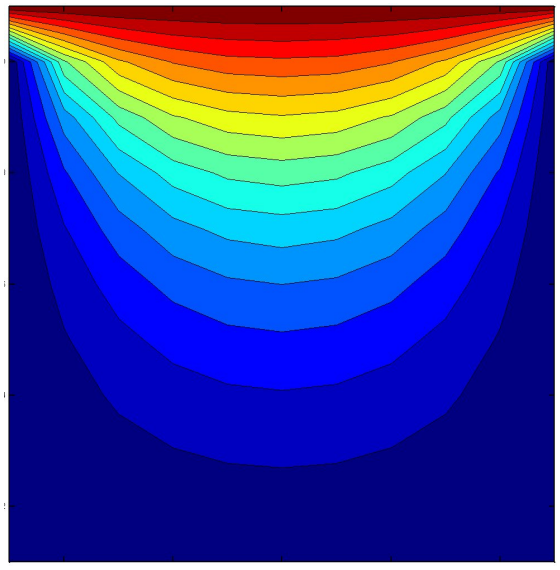
Vemos que o método de Gauss-Seidel, neste tipo de problemas, sempre precisa de menos iterações para atingir a tolerância desejada que o método de Jacobi.

Exo. 2: Baseado no código que acabamos de ver, implementar o método de sobre-relaxação

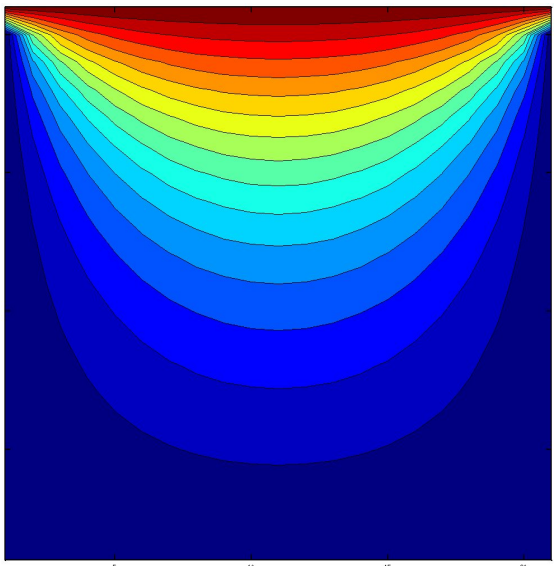
$$T^{(k)} = T^{(k-1)} + \omega \left(T^{GS} - T^{(k-1)} \right)$$

em que T^{GS} é o valor que seria predito pelo método de Gauss-Seidel. Resolver o problema para diferentes valores de N e ω medindo o numéro de iterações necessário para a convergência numa tolerância de 10^{-7}.

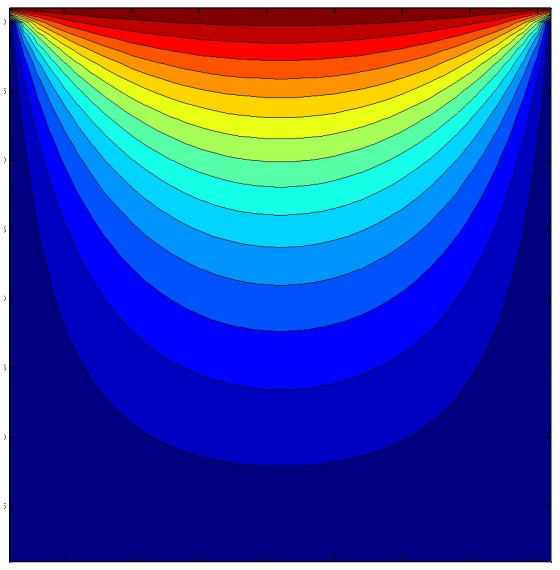
Quantas células precisamos colocar para ter resultados precisos?



10×10



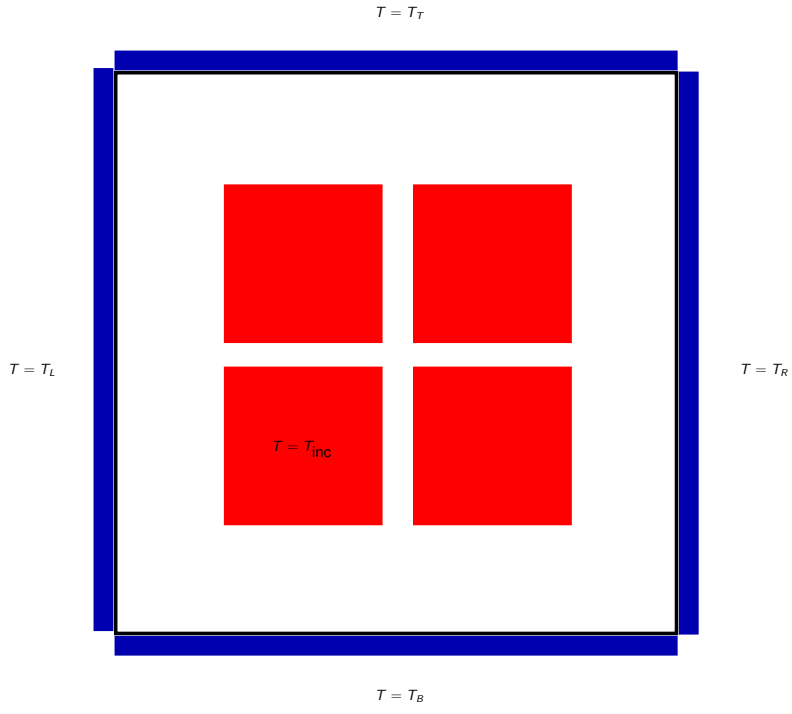
20×20



40×40

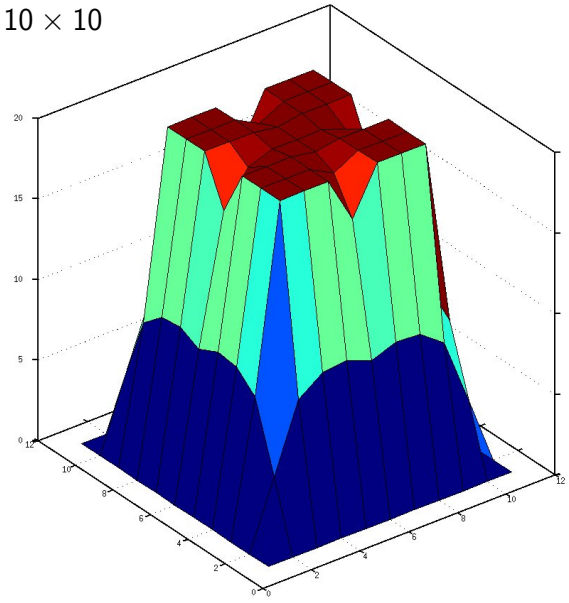
Neste problema, probabelmente seja suficiente com a discretização intermediaria. Mas, podemos tentar resolver problemas mais interessantes para melhor apreciar a necessidade de refinar os cálculos.

Vamos supor que temos regiões internas em que a temperatura é fixada.

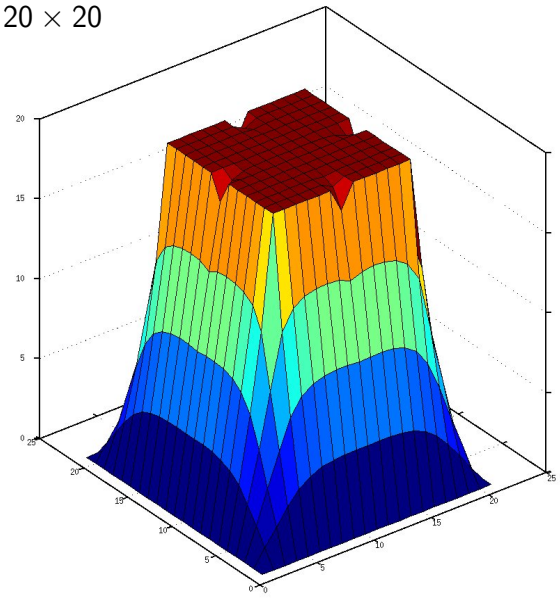


Pelos detalhes geométricos presentes no sistema, se queremos determinar com precisão a temperatura, p.e., na região central, serão necessárias muitas células de cálculo.

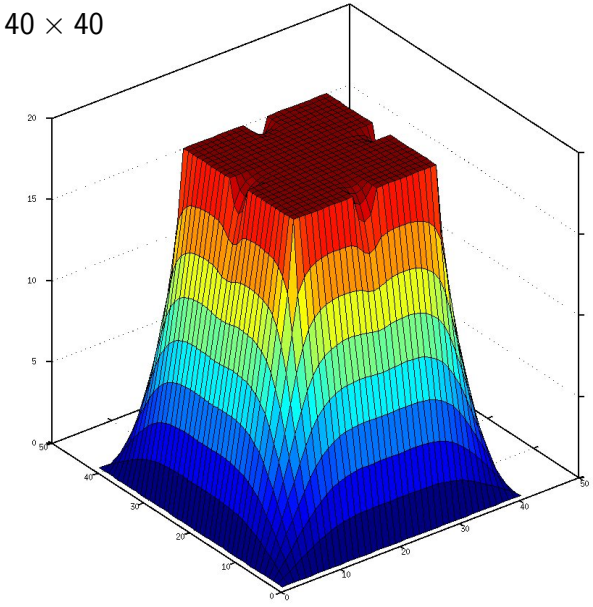
10×10



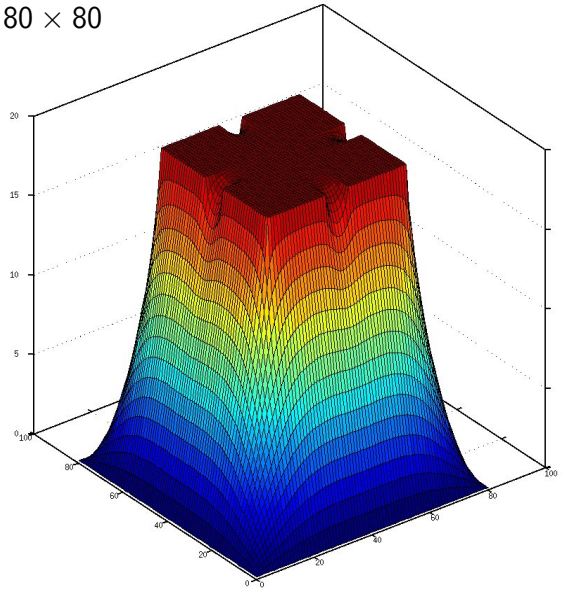
20×20



40×40



80×80



- Este exemplo mostra a necessidade de resolver grandes sistemas de equações.
- A situação piora dramaticamente quando passamos a três dimensões. Neste caso teriamos para o exemplo

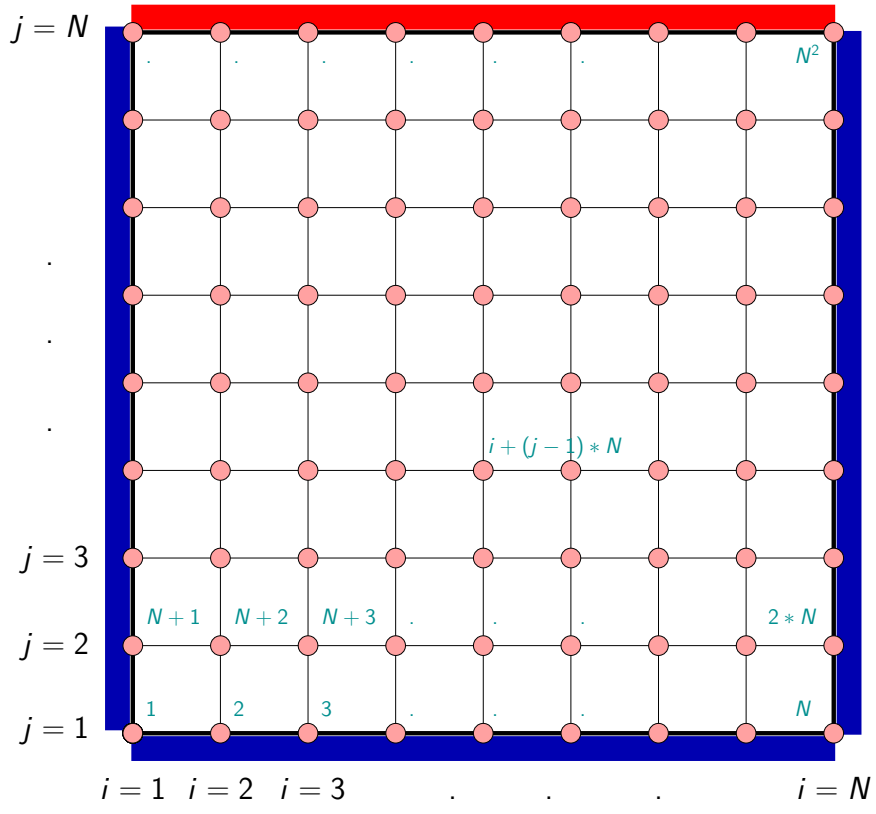
$$\#\text{ incognitas} = 80 \times 80 \times 80 = 512,000$$

  Na engenharia, é normal resolver sistemas que envolvem milhões de incógnitas!

- Precisamos métodos mais eficientes e sofisticados para atingir a precisão desejada em certos cálculos de engenharia.

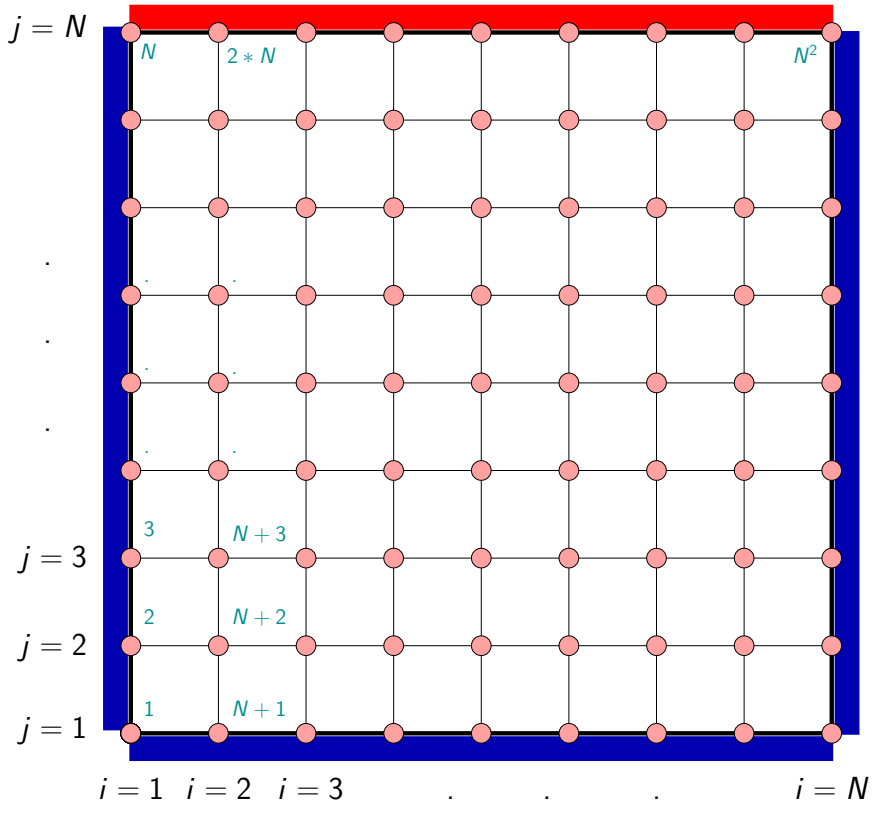
Já foram mencionados alguns, tais como o método dos Gradientes Conjugados. Para aplicar este método, primeiro vamos a escrever a forma matricial do problema.

Precisamos definir um índice global para as incógnitas, p.e.

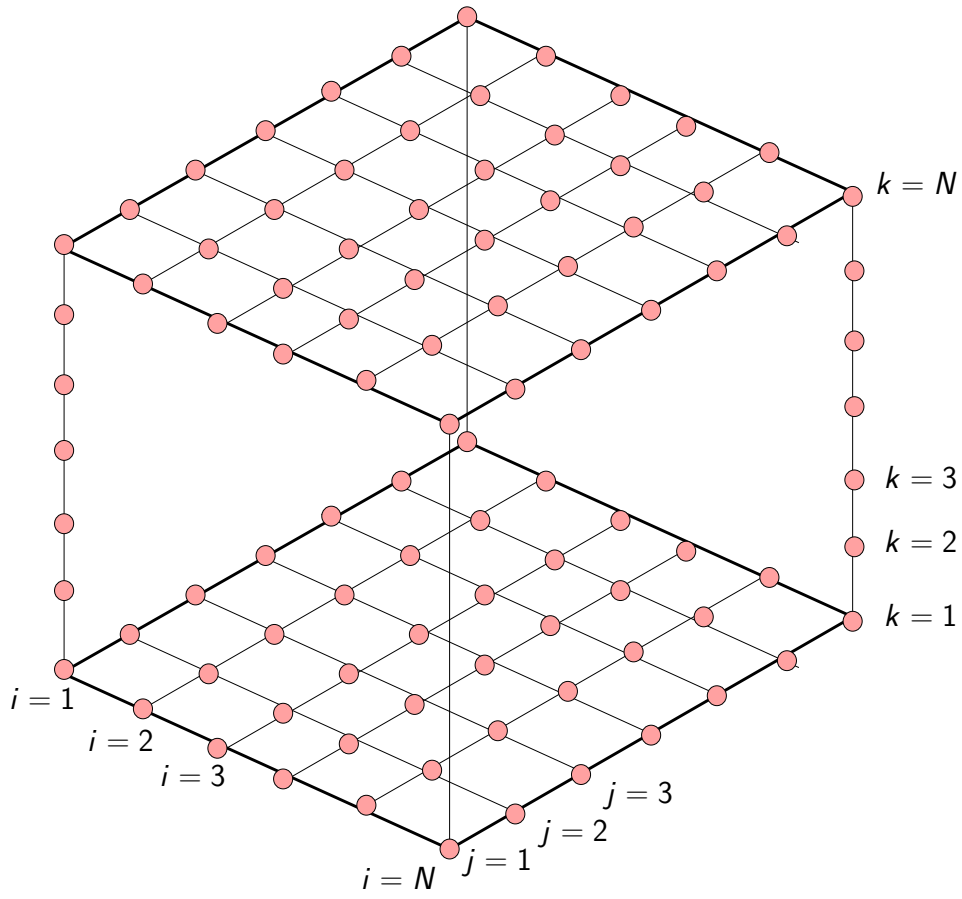



```
function IGLO = ij2n (i, j, N)
  IGLO = i + (j-1)*N;
end
```

Exo. 3 Como seria a função no caso de ter a seguinte numeração?



Exo. 4 Como seria a função para indexar as temperaturas no caso 3D, considerando N pontos em cada direção?



Montagem da matriz do problema térmico:
Lembremos que a equação para o nó i, j, no caso em que o coeficiente de conductividade c é constante:

$$-T_{i+1,j} - T_{i-1,j} + 4\,T_{i,j} - T_{i,j+1} - T_{i,j-1} = 0, \quad i, j = 2, \ldots, N-1$$

O que precisamos saber é a que índice global de temperaturas corresponde cada um dos termos na equação, i.e.

```
Ic = ij2n(i,   j,   N);
Ie = ij2n(i+1, j,   N);
Iw = ij2n(i-1, j,   N);
In = ij2n(i,   j+1, N);
Is = ij2n(i  , j-1, N);
```

Então a equação resulta:

$$-T_{Ie} - T_{Iw} + 4\,T_{Ic} - T_{In} - T_{Is} = 0$$

Finalmente, o código para montar a matriz é:

```
function A = MatAssTemp(N)
  nunk = N*N;
  A = zeros(nunk,nunk);
  for i=2:N-1
    for j=2:N-1
      Ic = ij2n(i,   j,   N);  A(Ic,Ic) =  4;
      Ie = ij2n(i+1, j,   N);  A(Ic,Ie) = -1;
      Iw = ij2n(i-1, j,   N);  A(Ic,Iw) = -1;
      In = ij2n(i,   j+1, N);  A(Ic,In) = -1;
      Is = ij2n(i,   j-1, N);  A(Ic,Is) = -1;
    end
  end
  A = sparse(A);
end
```

Mas, está faltando colocar as condições de temperatura fixa nas paredes. Para isto vamos fazer o mesmo que faziamos com as redes hidráulicas: Já que conhecemos a temperatura no nó que está sobre a parede, vamos associar com esse nó uma equação trivial. Por exemplo, se o nó i, j com indice global Ic, está na parede direita escreveriamos:

$$0\,T_1 \;+\; 0\,T_2 \;+\; \ldots \;+\; 1\,T_{Ic} \;+\; \ldots \;+\; 0\,T_{N^2} = T_R$$

Em octave:

```
Ic = ij2n(i, j, N);
A(Ic,:) = 0; %Nao necessario pois A foi iniciada a zero
A(Ic,Ic) = 1;
b(Ic) = TR
```

Para nosso problema original, o código para colocar todas as condições de contorno sería:

```
function [A b] = SetBCTemp(A, N, TL, TR, TB, TT)
  nunk = N*N;
  b = zeros(nunk,1);
  % Paredes verticais
  for j=1:N
    Ic = ij2n(1,j,N); A(Ic,Ic) = 1; b(Ic) = TL;
    Ic = ij2n(N,j,N); A(Ic,Ic) = 1; b(Ic) = TR;
  end
  % Paredes horizontais
  for i=1:N
    Ic = ij2n(i,1,N); A(Ic,Ic) = 1; b(Ic) = TB;
    Ic = ij2n(i,N,N); A(Ic,Ic) = 1; b(Ic) = TT;
  end
end
```

Esta forma de colocar as condições de contorno é muito prática, porém ela tem uma desvantagem: **A matriz perde a simetria!**. Há métodos, tais como o método dos gradientes ou dos gradientes conjugados que precisam de uma matriz simétrica!

Neste caso, precisamos impor as condições de borda sem perder a simetria. Vejamos como seria com um exemplo:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

Vamos supor que sabemos que $T_3 = T_R$, então podemos passar essa incógnita para o lado direito:

$$\begin{pmatrix} a_{11} & a_{12} & 0 & a_{14} & a_{15} \\ a_{21} & a_{22} & 0 & a_{24} & a_{25} \\ 0 & 0 & 1 & 0 & 0 \\ a_{41} & a_{42} & 0 & a_{44} & a_{45} \\ a_{51} & a_{52} & 0 & a_{54} & a_{55} \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{pmatrix} = \begin{pmatrix} b_1 - a_{13}\, T_R \\ b_2 - a_{23}\, T_R \\ T_R \\ b_4 - a_{43}\, T_R \\ b_5 - a_{53}\, T_R \end{pmatrix}$$

Como fazemos isto em Octave?

```
Ic = ij2n(i, j, N);
b(:) = b(:) - A(:,Ic)*TR;
b(Ic) = TR;
A(Ic,:) = 0;
A(:,Ic) = 0;
A(Ic,Ic) = 1;
```

Agora que já sabemos montar a matriz do sistema, queremos testar os diferentes métodos que temos estudados:

Dados $\mathbf{x}^{(0)}$, TOL, MAX_IT, $k = 0$

Enquanto $\|\mathbf{r}^{(k)}\| > TOL$ e $k < MAX\_IT$

1. Resolver $M\,\mathbf{d}^{(k+1)} = -\mathbf{r}^{(k)}$
2. Determinar o escalar β_{k+1}
3. Avançar: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \beta_{k+1}\,\mathbf{d}^{(k+1)}$
4. Calcular residual: $\mathbf{r}^{(k+1)} = A\,\mathbf{x}^{(k+1)} - \mathbf{b}$
5. Incrementar k

Fim

Para testar os métodos, considerar o código:

```
clear ;
% Dados do problema
N = 21;  % Numero de pontos em cada direcao
nunk = N*N; % Numero de incognitas
TL=TR=TB=0.0; TT = 20.0; % Temperaturas nas bordas

tic
A = MatAssTemp(N);
[A b] = SetBCTemp(A, N, TL, TR, TB, TT);
tas = toc; % Medimos tempo para montagem

tic
T0 = zeros(nunk,1);
[T, num_iter] = iterativo(A, b, T0, 10000, ...
                1.0e-8, "Jacobi");
tsol = toc; % Medimos tempo para resolver
% Plotar resultados
Taux=reshape(T,N,N);
contourf(Taux',15);
```

A função `iterativo.m` tem vários métodos implementados:

- Jacobi;
- Gauss-Seidel;
- Gradientes;
- Método baseado na fatoração LU incompleta;

Se o método requerer uma matriz simétrica, a função `SetBCTempSim.m`, que preserva a simetria da matriz, está disponível.

Exo. 5: Entender todas as linhas do código anterior e da função `iterativo.m` fornecidos.

Que é uma fatoração incompleta?

A ideia é realizar a fatoração de uma matriz, mas, sem aumentar muito os requerimentos de memória e pagar o preço de um método direito.

Por exemplo, a fatoração LU incompleta consiste em calcular fatores $\widetilde{L}$ e $\widetilde{U}$ tais que:

$$\widetilde{L}\,\widetilde{U} \approx A$$

Os fatores $\widetilde{L}$ e $\widetilde{U}$ estão construidos de forma que a **esparsidade** da matriz original A seja preservada, ou se o usuario quiser admitindo diferentes níveis de recheado. Quanto mais recheado admitimos, mais se parecerá o produto $\widetilde{L}\,\widetilde{U}$ com A.

Em Octave:

```
[l, u] = luinc(A, TOL);
```

A tolerância `TOL` indica a partir de que tamanho os coeficientes serão ignorados. Se `TOL` for muito pequeno, então,o produto dos fatores será muito parecido com `A`, em caso contrário

```
M = l*u;
```

apenas será uma aproximação da matriz `A`, e por tanto poderiamos utilizar-la no nosso método iterativo geral ou como pre-condicionador no método dos gradientes conjugados.

Exo. 6: Fazer um código de Octave que mida a norma da diferença entre a matriz `A` e a matriz `M = l*u;` para valores da tolerância `TOL` igual a: 10^{-1}, 10^{-2}, 10^{-3}, ... , 10^{-7}.

Métodos baseados em subespaços de Krylov

O método mais simples é chamado de método dos gradientes conjugados, o qual é uma variante do método dos gradientes.

- Dado $\mathbf{x}^{(0)}$ e $\mathbf{r}^{(0)} = A\,\mathbf{x}^{(0)} - \mathbf{b}$
- Calcular $\mathbf{p}^{(1)} = \mathbf{r}^{(0)}$ e $\beta_1 = \frac{{\mathbf{r}^{(0)}}^T \mathbf{r}^{(0)}}{(A\mathbf{r}^{(0)})^T \mathbf{r}^{(0)}}$
- Calcular $\mathbf{x}^{(1)} = \mathbf{x}^{(0)} + \beta_1 \mathbf{p}^{(1)}$
- Calcular $\mathbf{r}^{(1)} = \mathbf{r}^{(0)} + \beta_1\, A\mathbf{p}^{(1)}$ e definir $k = 2$
- Enquanto $\|\mathbf{r}^{(k-1)}\| > TOL_r$
  - Calcular: $\alpha_{k-1} = \frac{{\mathbf{r}^{(k-1)}}^T \mathbf{r}^{(k-1)}}{{\mathbf{r}^{(k-2)}}^T \mathbf{r}^{(k-2)}}$
  - Calcular direção: $\mathbf{p}^{(k)} = -\mathbf{r}^{(k-1)} + \alpha_{k-1}\,\mathbf{p}^{(k-1)}$

  - Calcular $\beta_k = \frac{\mathbf{r}^{(k-1)\,T}\,\mathbf{r}^{(k-1)}}{(A\,\mathbf{p}^{(k)})^T\mathbf{p}^{(k)}}$
  - Avançar: $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \beta_k \mathbf{p}^{(k)}$
  - Calcular residual: $\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} + \beta_k\, A\,\mathbf{p}^{(k)}$
  - Incrementar k

- Fim

O melhor sempre é usar implementações já disponíveis. Em Octave temos a função `pcg`:

```
[T, FLAG, RELRES, ITER] = pcg(A, b, 1.0e-8, 2000, pre);
```

em que `pre` é uma matriz de precondicionamento, que ajuda a acelerar a convergência, p.e.:

- Método direito (converge em 1 iteração):

  ```
  pre = A;
  ```

- Precondicionamento de Jacobi:

  ```
  pre = diag(diag(A));
  ```

- Precondicionamento com fatoração LU incompleta:

  ```
  [l u] = luinc(A, TOL);  pre = l*u;
  ```

O método mais utilizado atualmente é o *Generalized Minimum Residual Method* GMRES. Neste caso a matriz não precisa ser simétrica. A forma de utilizar-lo em Octave é:

```
[T, FLAG, RELRES, ITER] = gmres(A, b, [], 1.0e-8, 2000, pre);
```

Exo. 7: Testar os métodos `pcg` e `gmres` em Octave utilizando como precondicionador a fatoração LU incompleta.