Diferenças finitas e a equação de Laplace

Equações elípticas são usadas tipicamente para caracterizar problemas de contorno estacionário. Para introduzir o assunto, vamos analisar um caso simples, a equação de Laplace, deduzido a partir de um contexto físico.

A equação de Laplace pode ser usada para modelar diversos problemas envolvendo o potencial de uma variável desconhecida. Por simplicidade, vamos usar uma placa aquecida como contexto básico para a dedução e resolução dessa EDP elíptica.

A figura a seguir mostra um elemento na face de uma placa retangular fina de espessura Δz. A placa é isolada em toda parte, exceto nas bordas, em que a temperatura é constante. Assim, a transferência de calor é limitada às direções x e y.

No estado estacionário, o fluxo de calor para dentro do elemento em um período unitário Δt deve ser igual ao fluxo para fora, como em

Desejamos agora resolver numericamente um problema de condução de calor, usando a equação de Laplace

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial x^2} = 0$$

A eq. de Laplace é uma EDP elíptica usada para modelar diversos problemas de engenharia envolvendo o potencial de uma variável desconhecida.

Em razão de sua simplicidade e aplicabilidade, será utilizado como contexto fundamental para o desenvolvimento desse trabalho, uma placa aquecida.

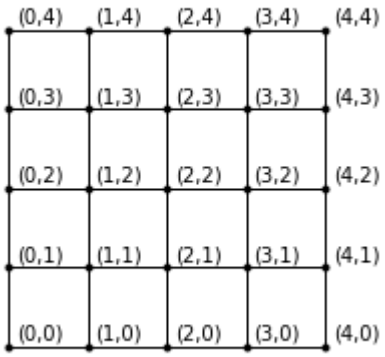
Para a solução numérica por diferenças finitas, trata-se a placa como uma grade de pontos onde as derivadas parciais da equação de Laplace são substituidas por aproximações.

In [43]:

```
import matplotlib.pyplot as plt
import numpy as np
```

In [44]:

```
# código apenas para gerar a imagem da malha
x = np.linspace(0,4,5)
y = np.linspace(0,4,5)
fig, ax = plt.subplots()

for i in range(len(x)):
    for j in range(len(y)):
        ax.plot([x[i],x[i]], [y[0],y[-1]], 'k-', lw=0.5)
        ax.plot([x[0],x[-1]], [y[i],y[i]], 'k-', lw=0.5)
        ax.plot(i, j, 'k.')
        texto = '('+str(i)+','+str(j)+')'
        ax.text(i+0.1 , j+0.1, texto)

# Remove as bordas
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)
ax.spines['bottom'].set_visible(False)
ax.spines['left'].set_visible(False)
plt.ylim(-0.5, 5)
plt.xlim(-0.5, 5)
# remove a escala
plt.xticks([])
plt.yticks([])
ax.set_aspect('equal')
plt.show()
```



Lembrando que a derivada de segunda ordem pode ser aproximada em termos da variação na derivada primeira em dois pontos. Por exemplo, considerando as expanções em série de Taylor de f em torno de x_i e calculando a junção em x_{i-1} e x_{i+1} obtemos:

$$f(x_{i-1}) = f(x_i) - hf'(x_i) + \frac{h^2 f''(x_i)}{2} - \frac{h^3 f'''(x_i)}{6} + \ldots$$

e

$$f(x_{i+1}) = f(x_i) + hf'(x_i) + \frac{h^2 f''(x_i)}{2} + \frac{h^3 f'''(x_i)}{6} + \ldots$$

Somando as duas equações acima obtemos:

$$f(x_{i-1}) + f(x_{i+1}) = 2f(x_i) + h^2 f''(x_i) + \frac{h^4 f''''(x_i)}{24} + \ldots$$

que, desprezando os termos com derivadas de 4ª ordem ou maiores, nos dá a seguinte aproximação para a derivada segunda:

$$f''(x_i) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2}$$

Assim, para cada ponto (i, j) da malha acima as aproximaçõesdadas pelas fórmulas de diferença dividida centrada, baseadas na expanção em série de Taylor e a EDP é transformada em uma equação de diferença algébrica , como descrito a seguir

$$\frac{\partial^2 T}{\partial x^2} = \frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2}$$

e

$$\frac{\partial^2 T}{\partial y^2} = \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2}$$

onde $T_{i,j}$ é a temperatura no ponto da i-ésima linha e j-ésima coluna. Substituindo essas expressões na equação de Laplace, obtem-se

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{\Delta x^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{\Delta y^2} = 0$$

Para uma grade quadrada, ou seja, em que $\Delta x = \Delta y$, a equação pode ser reescrita como

$$T_{i+1,j} + T_{i-1,j} + T_{i,j+1} + T_{i,j-1} - 4T_{i,j} = 0$$

ou

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

a qual é válida para todos pontos no interior da placa. Essa relação é conhecida como *equação de diferenças de Laplace*.

Desta maneira, é desenvolvida uma malha com $n \times m$ nós com espaçamentos e nas coordenadas x e y respectivamente e, ao aplicar as condições de contorno, é obtido um sistema linear com $(n-2) \times (m-2)$ equações e $(n-2) \times (m-2)$ incógnitas, de modo que é possível solucioná-lo numericamente utilizando-se, por exemplo, o método iterativo de Gauss-Seidel.

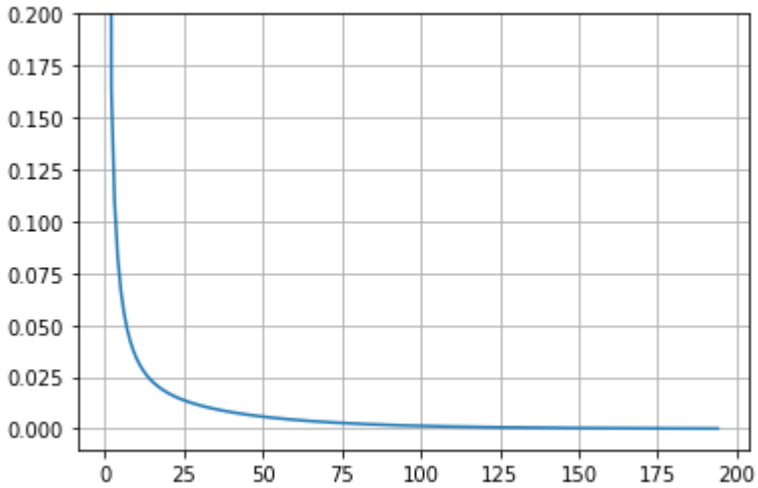
O método iterativo de Gauss-Seidel é interrompido quando a estimativa para os erros relativos

$$|Err| = \left| \frac{T_{i,j}^{novo} - T_{i,j}^{velho}}{T_{i,j}^{novo}} \right|$$

seja menor que uma tolerância pré estabelecida.

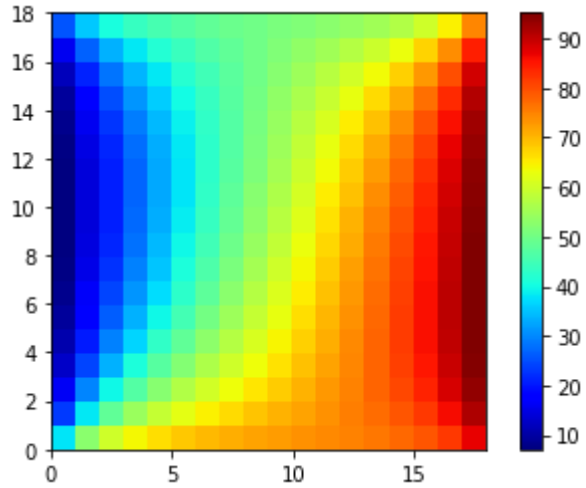
In [173]:

```
from matplotlib.pyplot import *
import matplotlib
import numpy as np

ni = 20
nj = 20

T = np.zeros([ni,nj])
T_ant = T.copy()
T[-1, :] = 50   # em cima
T[:, -1] = 100  # direita
T[0,  :] = 75   # embaixo
T[:,  0] = 0    # esquerda

err = 1000
n = 0
err_plot = []
int_plot = []

#print(T)
while err>0.0001:
    n=n+1
    int_plot.append(n)
    for i in range(1,len(T)-1):
        for j in range(1,len(T)-1):
            T[i,j] = (T[i-1,j]+T[i+1,j]+T[i,j-1]+T[i,j+1])/4
    err = np.linalg.norm(T-T_ant)/np.linalg.norm(T)
    err_plot.append(err)

    #print(err)
    T_ant = T.copy()


plt.plot(int_plot, err_plot)
plt.ylim(-0.01,0.2)
plt.grid()
```



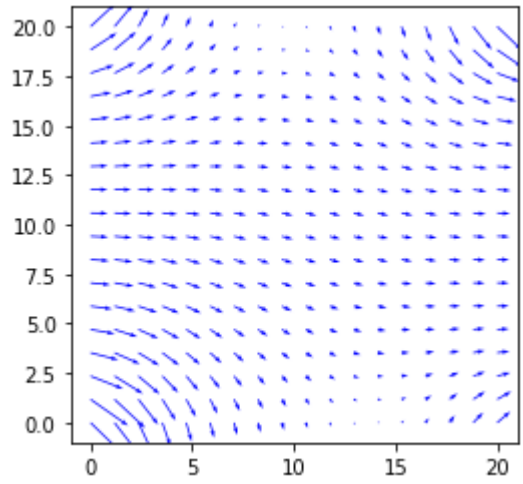
In [174]:

```
# plot
pcolor(T[1:-1,1:-1],cmap='jet')
gca().set_aspect('equal')
colorbar()
show()
```



In [175]:

```
fig = plt.figure(figsize=(4,4))
i,j = np.meshgrid(np.linspace(0,ni,ni-2),
                  np.linspace(0,nj,nj-2))

u = np.gradient(T[1:-1,1:-1])[1]
v = np.gradient(T[1:-1,1:-1])[0]
plt.quiver(i,j,u,v,
            color='b')

plt.show()
```



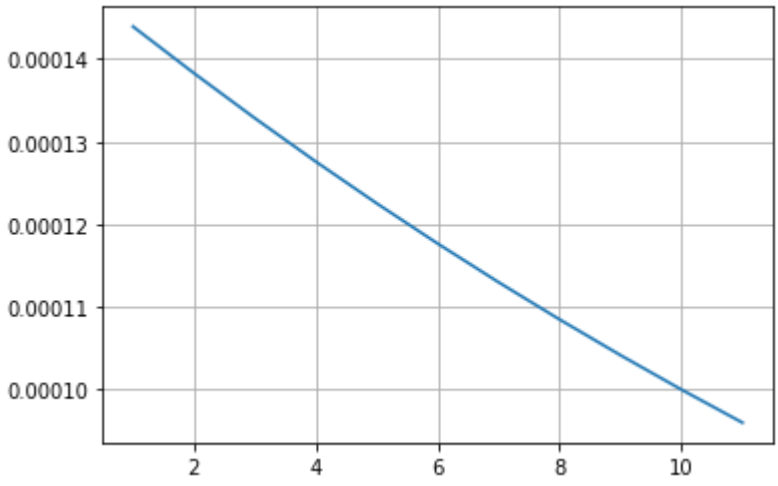
A chamada *sobrerelaxação* é usada para acelerar a taxa de convergência por meio da aplicação da seguinte fórmula, depois de cada interação:

$$T_{i,j}^{novo} = \lambda T_{i,j}^{novo} + (1 - \lambda) T_{i,j}^{velho}$$

onde λ é um fator de peso fixado entre 1 e 2.

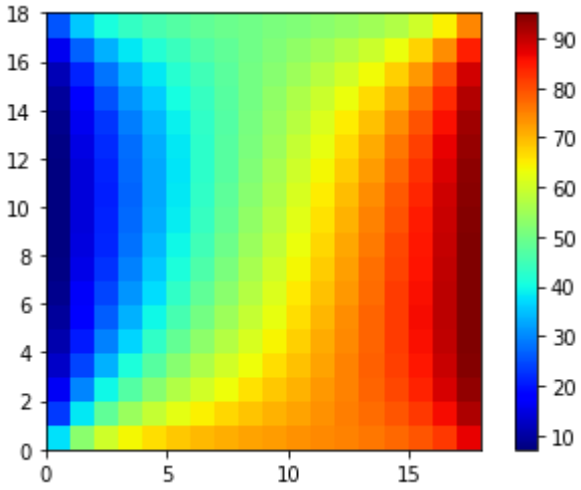
In [176]:

```
lamb = 1.2

err = 1000
n = 0
err_plot = []
int_plot = []

#print(T)
while err>0.0001:
    n=n+1
    int_plot.append(n)
    for i in range(1,len(T)-1):
        for j in range(1,len(T)-1):
            Tnovo = (T[i-1,j]+T[i+1,j]+T[i,j-1]+T[i,j+1])/4
            T[i,j]=lamb*Tnovo+(1-lamb)*T[i,j]
    err = np.linalg.norm(T-T_ant)/np.linalg.norm(T)
    err_plot.append(err)

    #print(err)
    T_ant = T.copy()


plt.plot(int_plot, err_plot)
plt.grid()
```



In [177]:

```
# plot
pcolor(T[1:-1,1:-1],cmap='jet')
gca().set_aspect('equal')
colorbar()
show()
```



In [ ]:

```

```

In [ ]:

```

```