# Modelação Numérica 2017

# Aula 12, 28/Mar

- Método de Runge-Kutta 4ª ordem (RK4)
- Advecção 2D

http://modnum.ucs.ciencias.ulisboa.pt

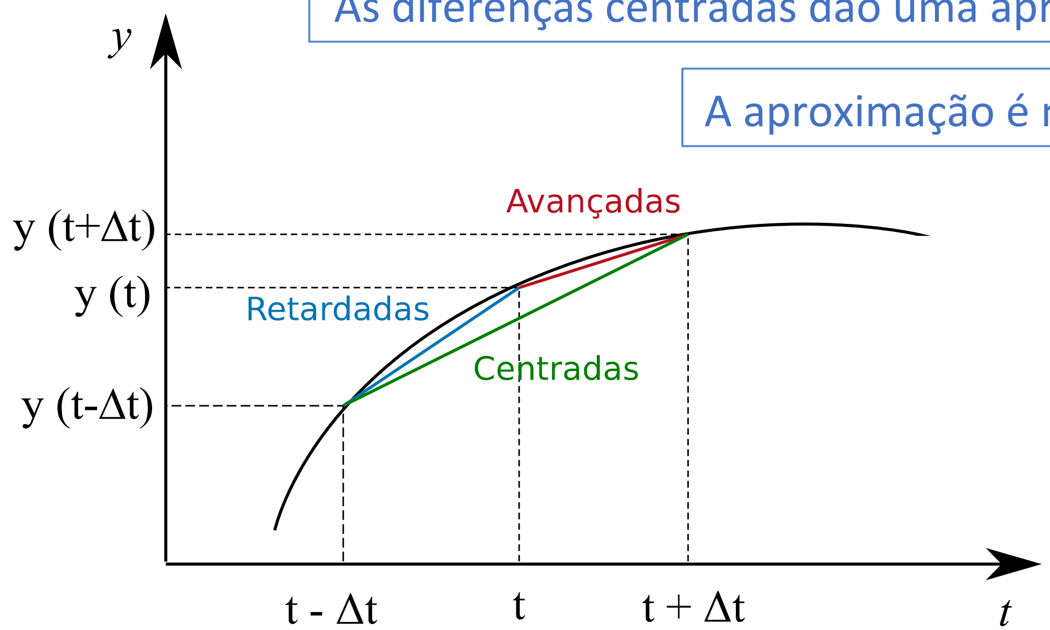
Aula passada

# Diferenças finitas

- Diferenças avançadas: $$\left(\frac{\partial f}{\partial x}\right)_{x=x_0} = \frac{f(x_0+\Delta x) - f(x_0)}{\Delta x} + \mathcal{O}(\Delta x)$$
- Diferenças retardadas: $$\left(\frac{\partial f}{\partial x}\right)_{x=x_0} = \frac{f(x_0) - f(x_0-\Delta x)}{\Delta x} + \mathcal{O}(\Delta x)$$
- Diferenças centradas: $$\left(\frac{\partial f}{\partial x}\right)_{x=x_0} = \frac{f(x_0+\Delta x) - f(x_0-\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2)$$

As diferenças centradas dão uma aproximação mais exacta

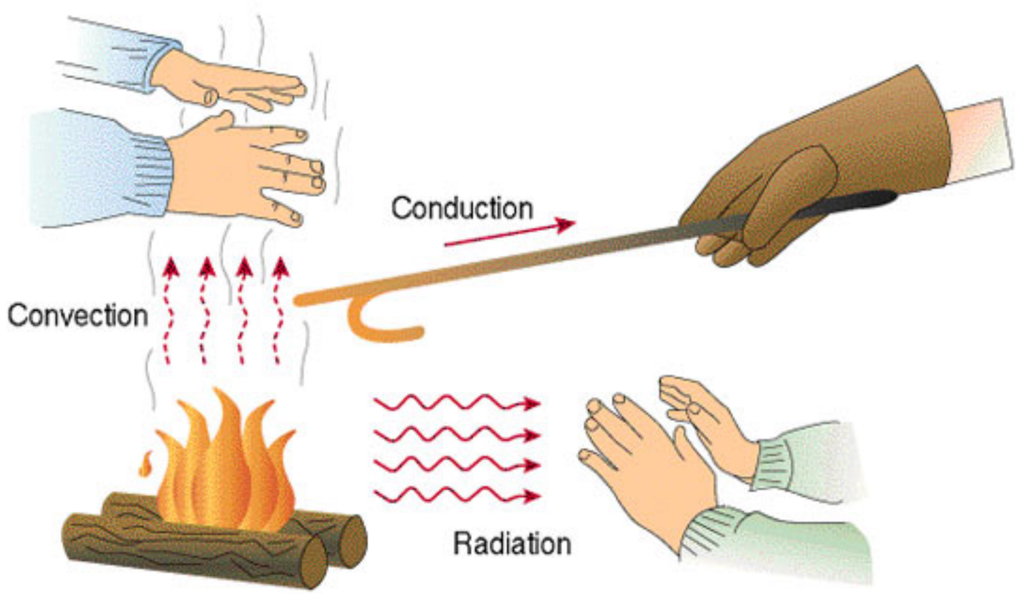
A aproximação é melhor quando Δx -> 0



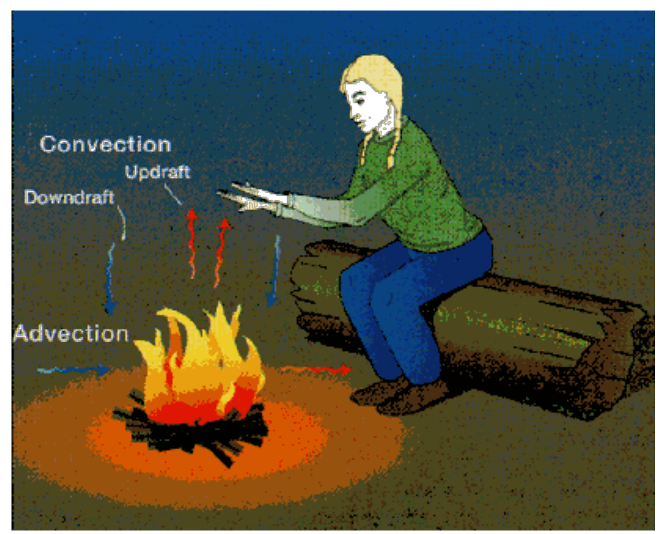
# Equação de advecção (linear, 1D)

Aula passada

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x}, \; u = const$$




https://en.wikipedia.org/wiki/Taylor_series



Aula passada

# 0. FTCS – Forward-time, central space (método instável)

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x},\; u = const$$

$$\frac{T_k^{n+1} - T_k^n}{\Delta t} = -u\frac{T_{k+1}^n - T_{k-1}^n}{2\Delta x} \Rightarrow T_k^{n+1} = T_k^n - u\Delta t\frac{T_{k+1}^n - T_{k-1}^n}{2\Delta x}$$

Diferenças avançadas no tempo

Diferenças centradas no espaço

Aula passada

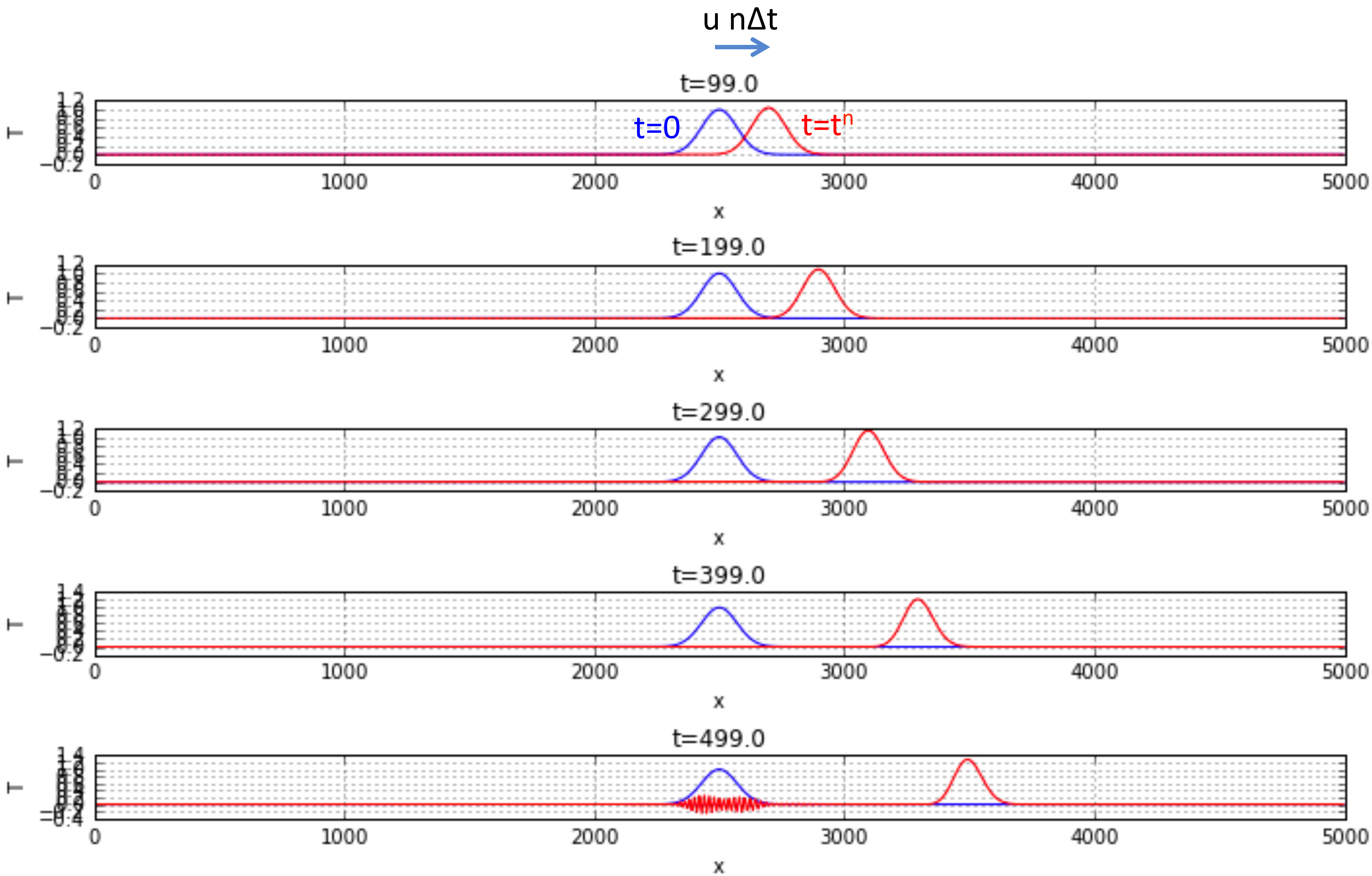
```python
import matplotlib.pyplot as plt
import numpy as np

plt.rcParams['figure.figsize'] = 10, 6

#%% Parâmetros

nx=1000; dx=5.            # número de pontos no espaço, intervalo entre pontos no espaço
nt=5000; dt=1.              # número de pontos no tempo, intervalo entre pontos no tempo
u=2.
x=np.arange(0,nx*dx,dx)     # vector de posições

#%% Condições iniciais

x0=dx*nx/2                  # ponto onde a temperatura inicial é máxima
L=100                       # largura da anomalia inicial de temperatura
Ti=np.exp(-((x-x0)/L)**2)   # vector de temperaturas iniciais
T=np.zeros(len(x))          # inicializar o vector de temperaturas presentes
Tp=np.zeros(len(x))         # inicializar o vector de temperaturas futuras (próximas)
T[:]=Ti[:]

# Evolução do sistema

isp=1
for it in range(1,nt):
    for ix in range(1,nx-1):
        Tp[ix] = T[ix] - u*dt/(2*dx)* (T[ix+1] - T[ix-1])   # próxima temperatura

    Tp[nx-1] = T[nx-1] - u*dt/(2*dx)* (T[0] - T[nx-2])      # fronteira cíclica
    Tp[0] = T[0] - u*dt/(2*dx)* (T[1] - T[nx-1])            # fronteira cíclica
    T[:]=Tp[:]

    if (it+1)%250==0 and isp<=5:
        plt.subplot(5,1,isp)
        plt.plot(x,Ti,'b', x,T,'r')
        plt.xlabel('x')
        plt.ylabel('T')
        plt.title('t='+str(it*dt))
        plt.grid()
        isp += 1

    if max(T) > 10:
        print('it=' + str(it)+ ', T=' + str(T))
        break

plt.tight_layout()
```

$$T_k^{n+1} = T_k^n - u\Delta t \frac{T_{k+1}^n - T_{k-1}^n}{2\Delta x}$$

$$x_0 = x_N,\ x_{N+1} = x_1$$

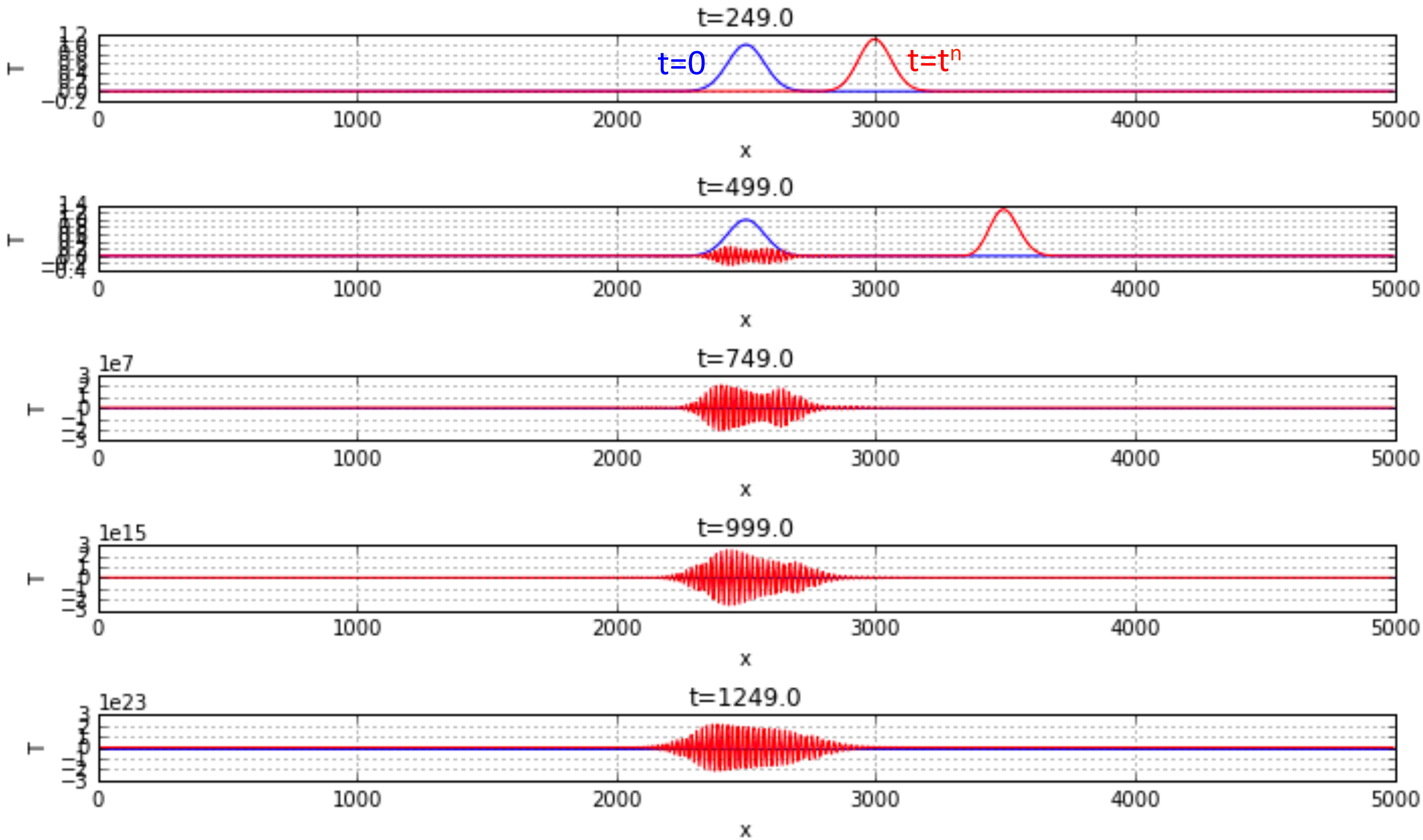
Aula passada

# 0. FTCS – Forward-time, central space (método instável)



Aula passada

# 0. FTCS – Forward-time, central space (método instável)



Aula passada

# 1. Aproximação de Lax–Friedrichs

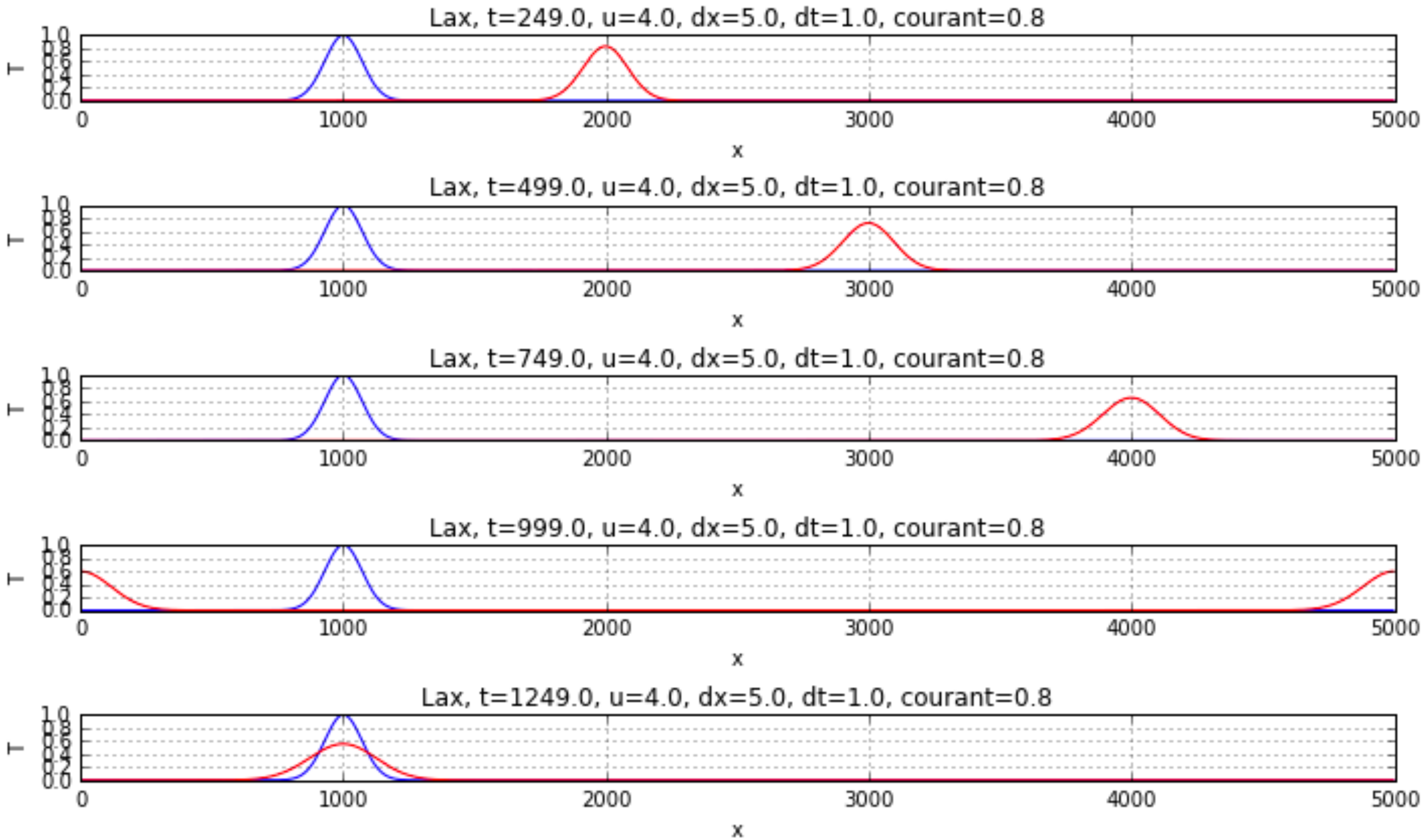
- Em vez de: $$T_k^{n+1} = \underline{T_k^n} - u\Delta t \frac{T_{k+1}^n - T_{k-1}^n}{2\Delta x}$$

- Fazemos: $$T_k^{n+1} = \underline{\frac{1}{2}(T_{k-1}^n + T_{k+1}^n)} - u\Delta t \frac{T_{k+1}^n - T_{k-1}^n}{2\Delta x}$$

- Continua a ser um método com 1 nível temporal e de primeira ordem no tempo e segunda no espaço.

Aula passada

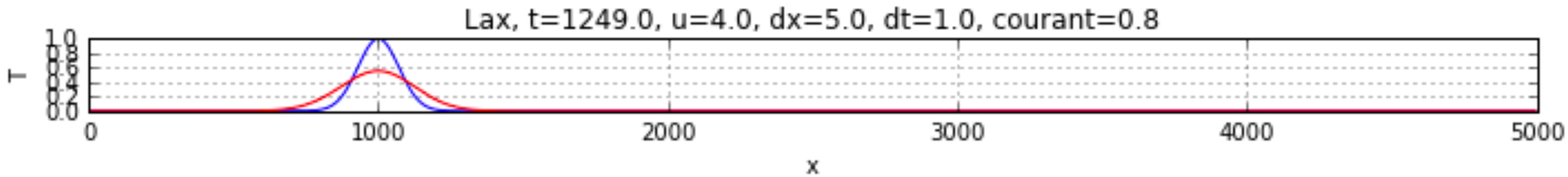
# 1. Comportamento do método Lax



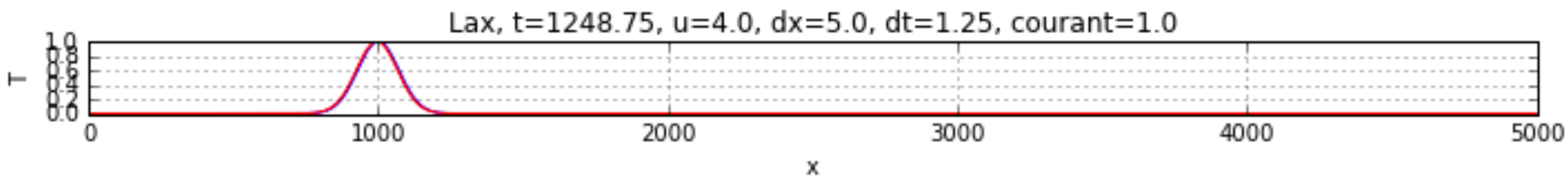
Aula passada

# 1. Comportamento do método Lax (10 voltas)

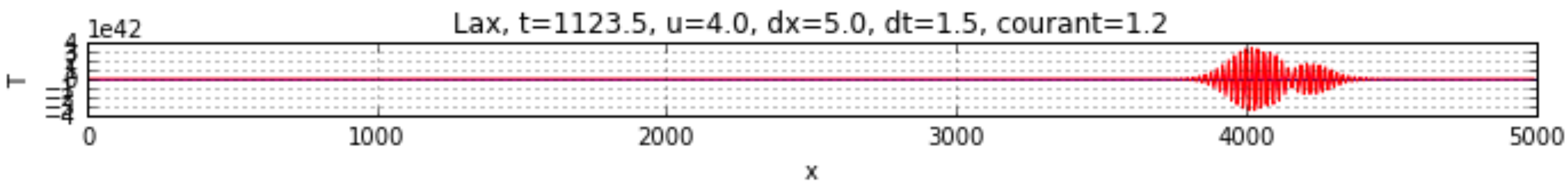
- Estável, difusivo:




- Estável (quase perfeito!):




- Instável




- Número de Courant: $\frac{u\Delta t}{\Delta x}$ $\begin{cases} \leq 1, \text{ estável} \\ > 1, \text{ instável} \end{cases}$

# 2. Upstream differencing

Aula passada

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x}, \; u = const$$

- FTCS: $T_k^{n+1} = T_k^n - u\Delta t\frac{T_{k+1}^n - T_{k-1}^n}{2\Delta x}$ Diferenças avançadas no tempo

- Se u>0: $T_k^{n+1} = T_k^n - u\Delta t\frac{T_k^n - T_{k-1}^n}{\Delta x}$ Diferenças retardadas no espaço

- Se u<0: $T_k^{n+1} = T_k^n - u\Delta t\frac{T_{k+1}^n - T_k^n}{\Delta x}$ Diferenças avançadas no espaço

Método de primeira ordem tanto no espaço como no tempo, explícito, de 1 nível.

# 3. Leapfrog (2ª ordem)

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x}, \; u = const$$

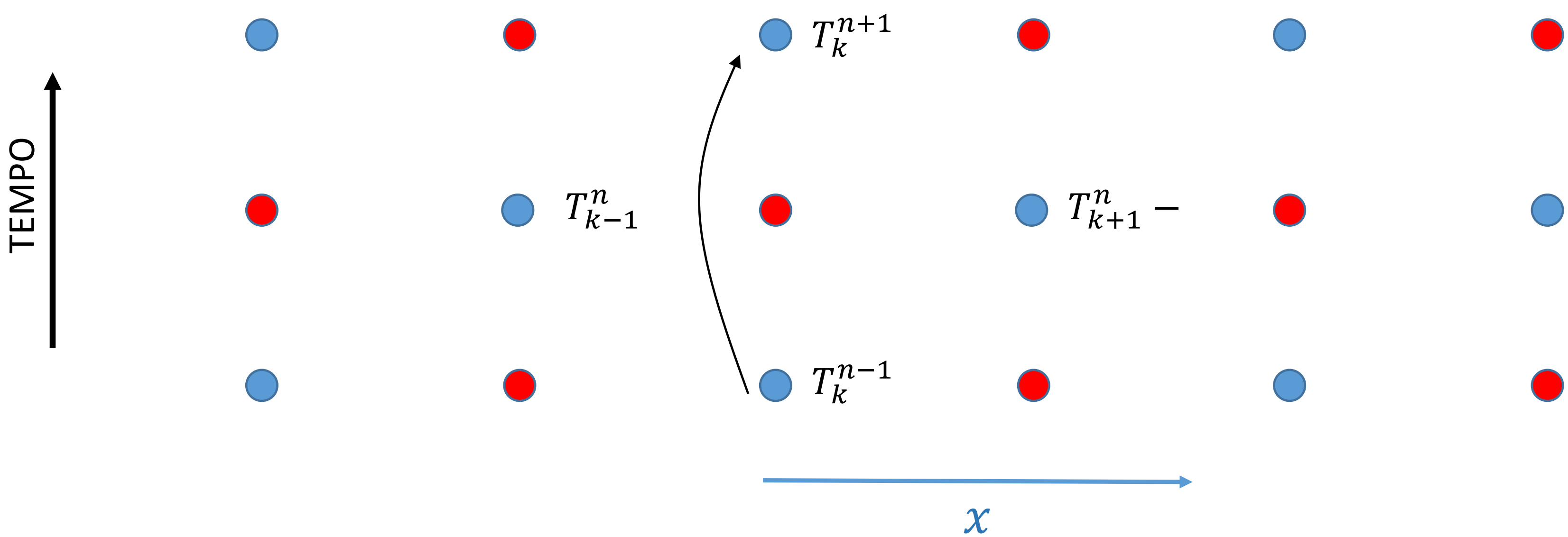
Aula passada

$$\frac{T_k^{n+1} - T_k^{n-1}}{2\Delta t} = -u\frac{T_{k+1}^{n} - T_{k-1}^{n}}{2\Delta x}$$

Diferenças centradas no tempo

Diferenças centradas no espaço

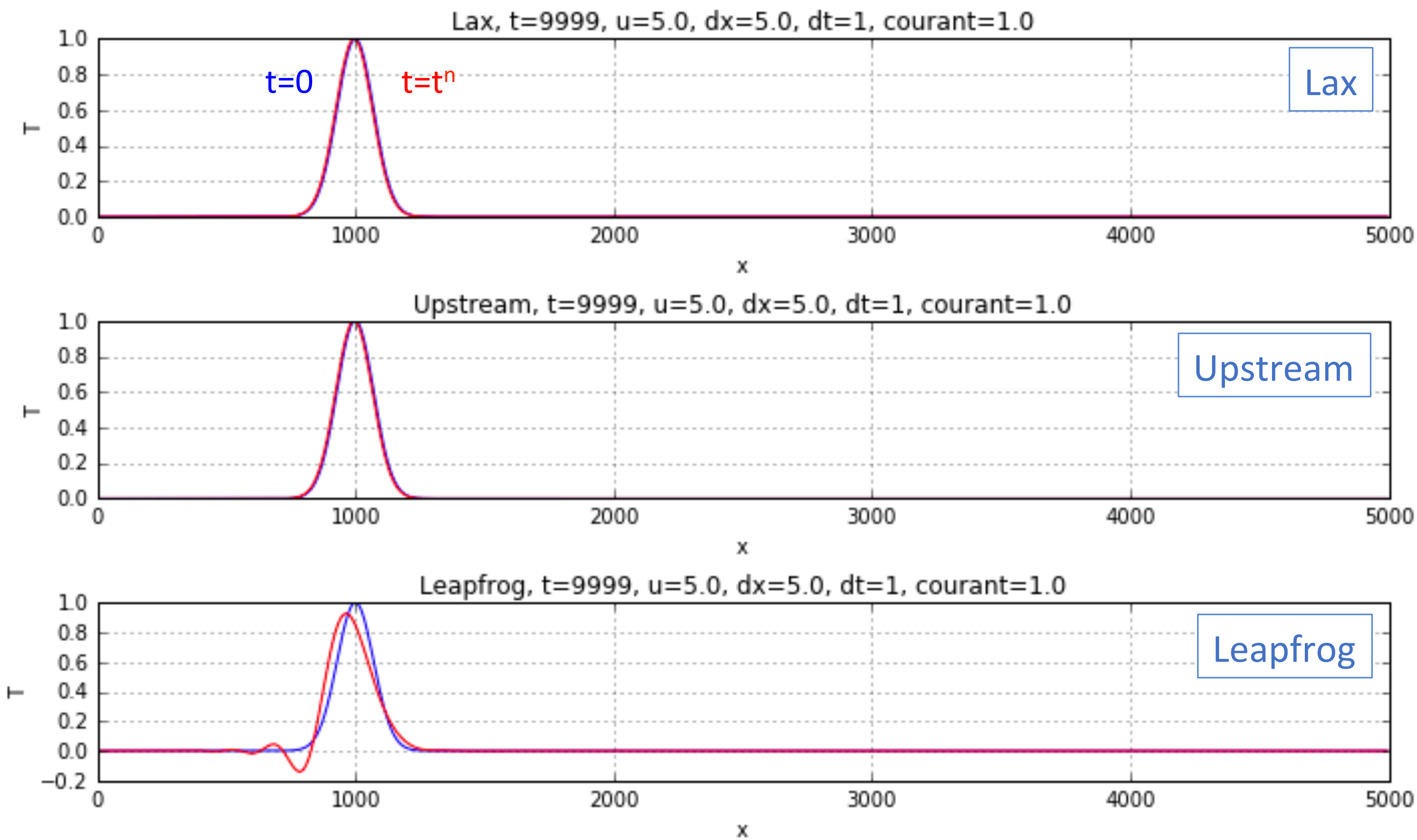
**Problema:** A malha computacional fica dividida em dois conjuntos desacoplados...



# Courant = 1.0

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x},\ u = const$$

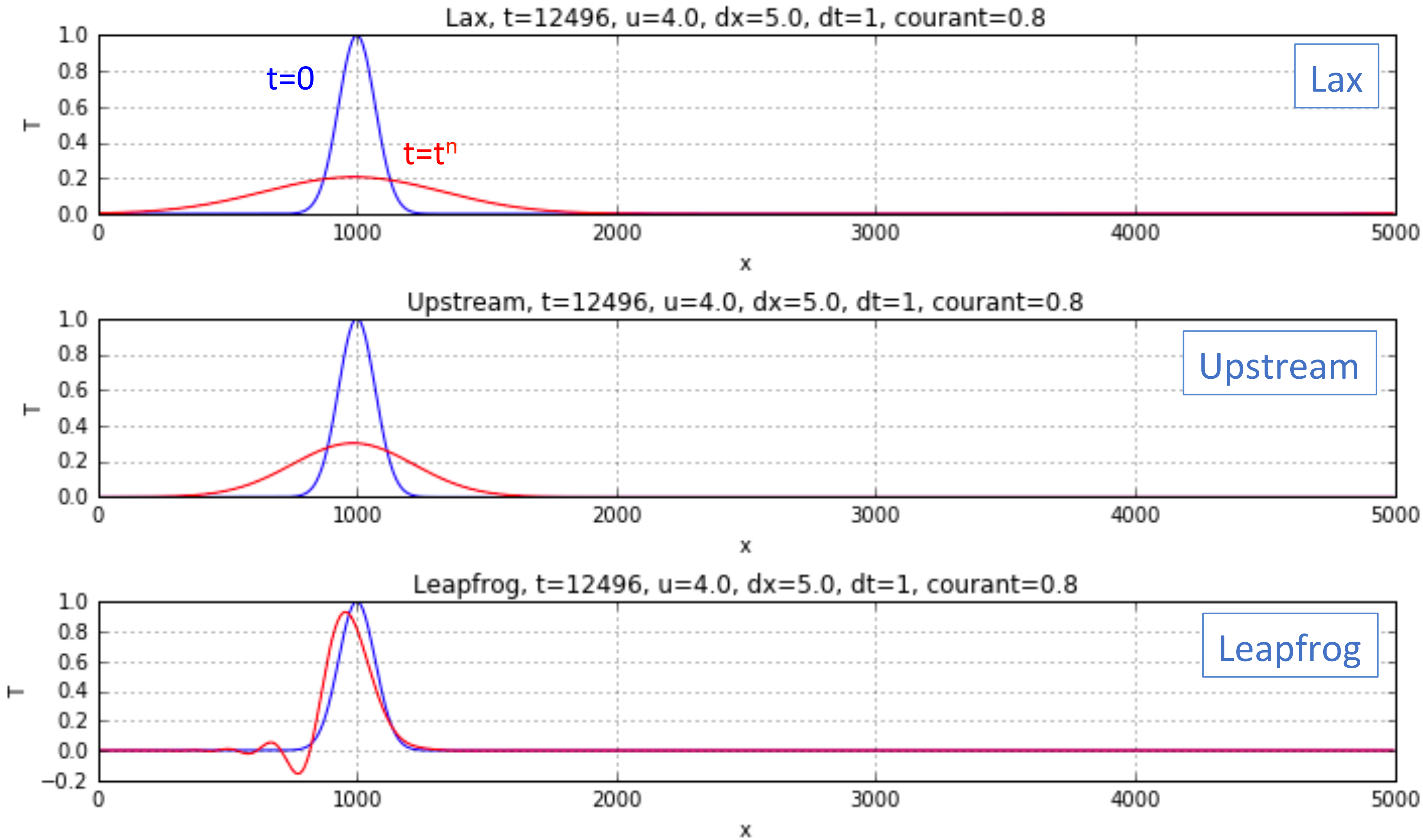
Aula passada



# Courant = 0.8

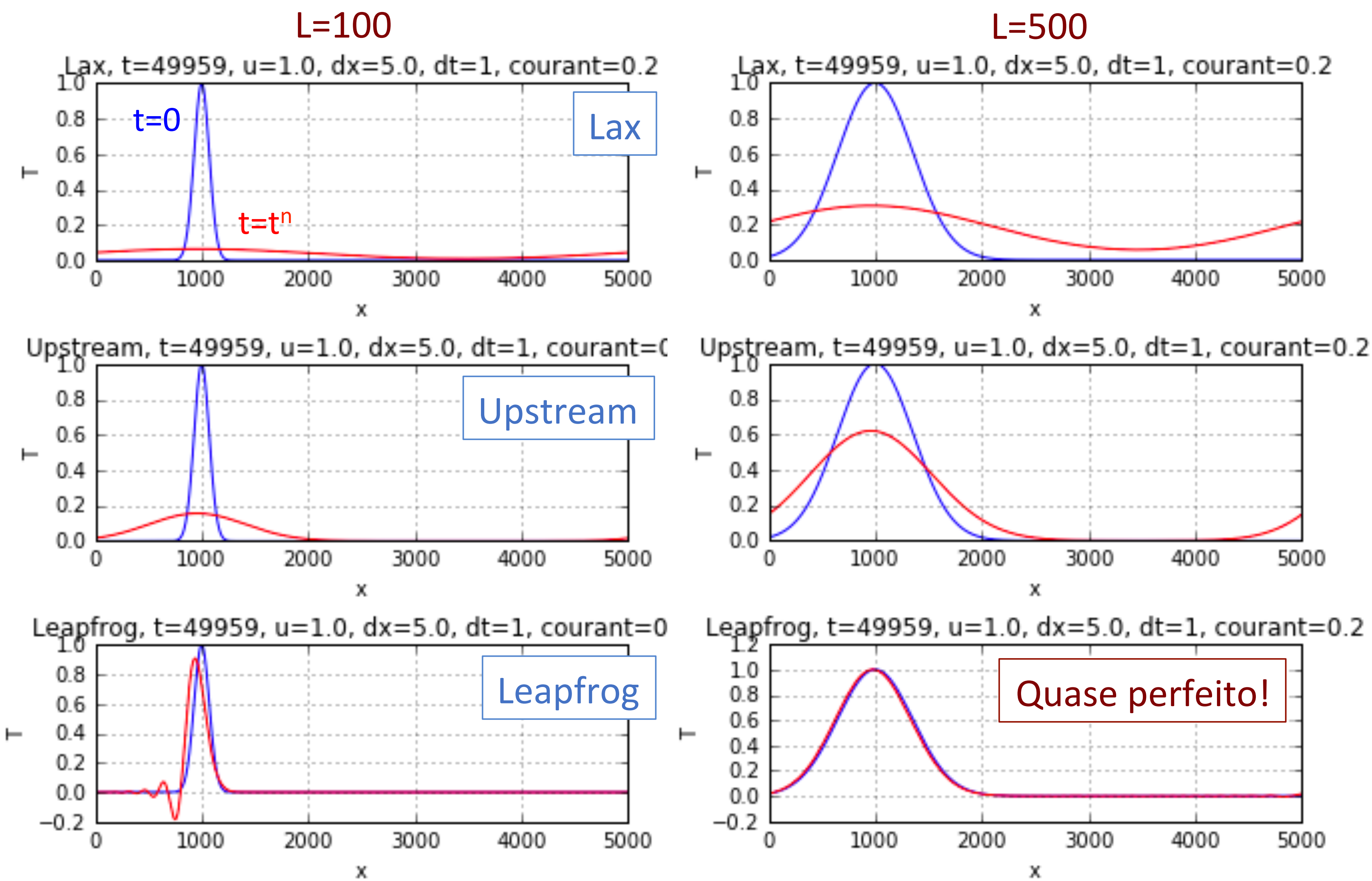
$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x}, \; u = const$$

Aula passada



Aula passada

# Impacto do espetro da perturbação a ser advectada (Courant=0.2, L=500)



# Métodos de Runge-Kutta RK4 (explícito, 4a ordem)

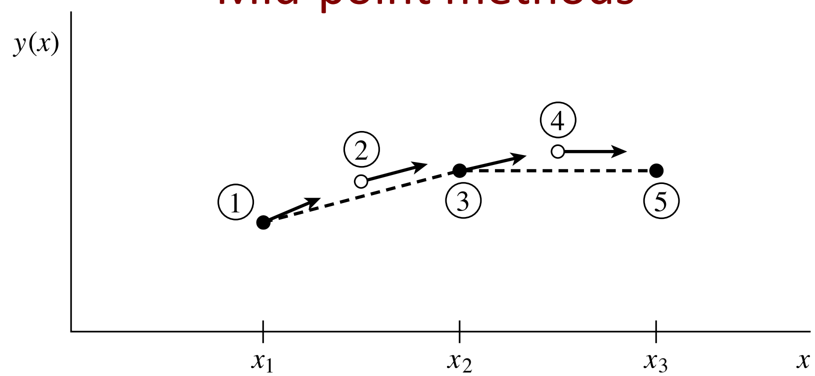
$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x},\ u = const$$

$$\dot{y} = \frac{\partial y}{dt} = f(t, y)$$

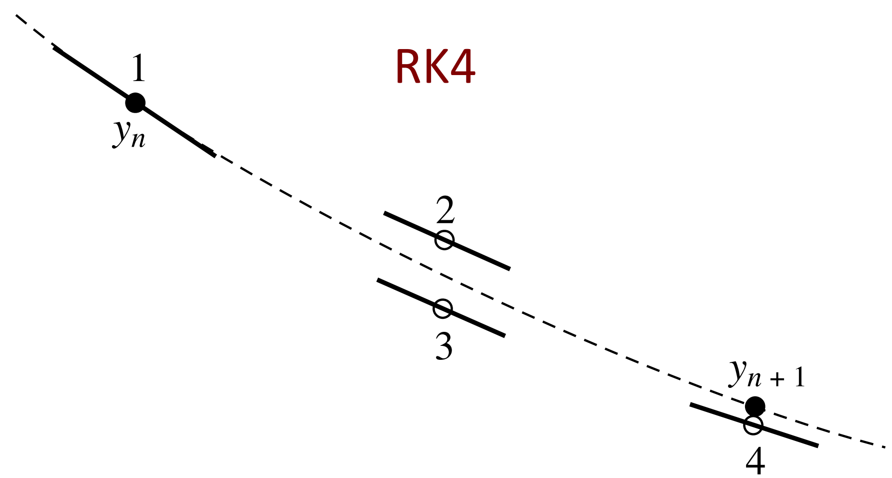
$$k_1 = f\left(t_n, y_n\right)$$
$$k_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_1\right)$$
$$k_3 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_2\right)$$
$$k_4 = f\left(t_n + \Delta t, y_n + \Delta t\, k_3\right)$$

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + \Delta t$$

Mid-point methods




RK4



# Métodos de Runge-Kutta RK4 (explícito, 4a ordem)

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x}, \; u = const$$

$$\dot{y} = \frac{\partial y}{dt} = f(t, y)$$

$$k_1 = f\left(t_n, y_n\right)$$
$$k_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} k_1\right)$$
$$k_3 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} k_2\right)$$
$$k_4 = f\left(t_n + \Delta t, y_n + \Delta t\, k_3\right)$$

$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$
$$t_{n+1} = t_n + \Delta t$$

- Solução em $t^{n+1}$ só depende do estado em $t^n$ (1 nível).
- Explícito.
- 4ª ordem resulta da utilização de passos intermédios no tempo e no espaço (é como se malha fosse refinada na vizinhança do ponto).

```python
#%% RK4
#%% Condições iniciais

T=np.zeros(len(x))          # inicializar o vector de temperaturas presentes (N)
Tn=np.zeros(len(x))         # inicializar o vector de temperaturas intermédias
Tp=np.zeros(len(x))         # inicializar o vector de temperaturas futuras
K1=np.zeros(len(x))         # inicializar o vector K1
K2=np.zeros(len(x))         # inicializar o vector K2
K3=np.zeros(len(x))         # inicializar o vector K3
K4=np.zeros(len(x))         # inicializar o vector K4
T[:]=Ti[:]

# Evolução do sistema

for it in range(1,nt):
    for ix in range(1,nx-1):
        K1[ix] = - u/(2*dx)* (T[ix+1] - T[ix-1])
    K1[nx-1] = - u/(2*dx)* (T[0] - T[nx-2])
    K1[0] = - u/(2*dx)* (T[1] - T[nx-1])

    Tn=T+.5*dt*K1
    for ix in range(1,nx-1):
        K2[ix] = - u/(2*dx)* (Tn[ix+1] - Tn[ix-1])
    K2[nx-1] = - u/(2*dx)* (Tn[0] - Tn[nx-2])
    K2[0] = - u/(2*dx)* (Tn[1] - Tn[nx-1])

    Tn=T+.5*dt*K2
    for ix in range(1,nx-1):
        K3[ix] = - u/(2*dx)* (Tn[ix+1] - Tn[ix-1])
    K3[nx-1] = - u/(2*dx)* (Tn[0] - Tn[nx-2])
    K3[0] = - u/(2*dx)* (Tn[1] - Tn[nx-1])

    Tn=T+dt*K3
    for ix in range(1,nx-1):
        K4[ix] = - u/(2*dx)* (Tn[ix+1] - Tn[ix-1])
    K4[nx-1] = - u/(2*dx)* (Tn[0] - Tn[nx-2])         # fronteira cíclica
    K4[0] = - u/(2*dx)* (Tn[1] - Tn[nx-1])            # fronteira cíclica

    Tp = T+dt/6.*(K1+2.*K2+2.*K3+K4)                  # temperatura futura (P)
    T[:]=Tp[:]
```

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x},\ u = const$$

$$k_1 = f\left(t_n, y_n\right)$$

$$k_2 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_1\right)$$

$$k_3 = f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_2\right)$$

$$k_4 = f\left(t_n + \Delta t, y_n + \Delta t\, k_3\right)$$

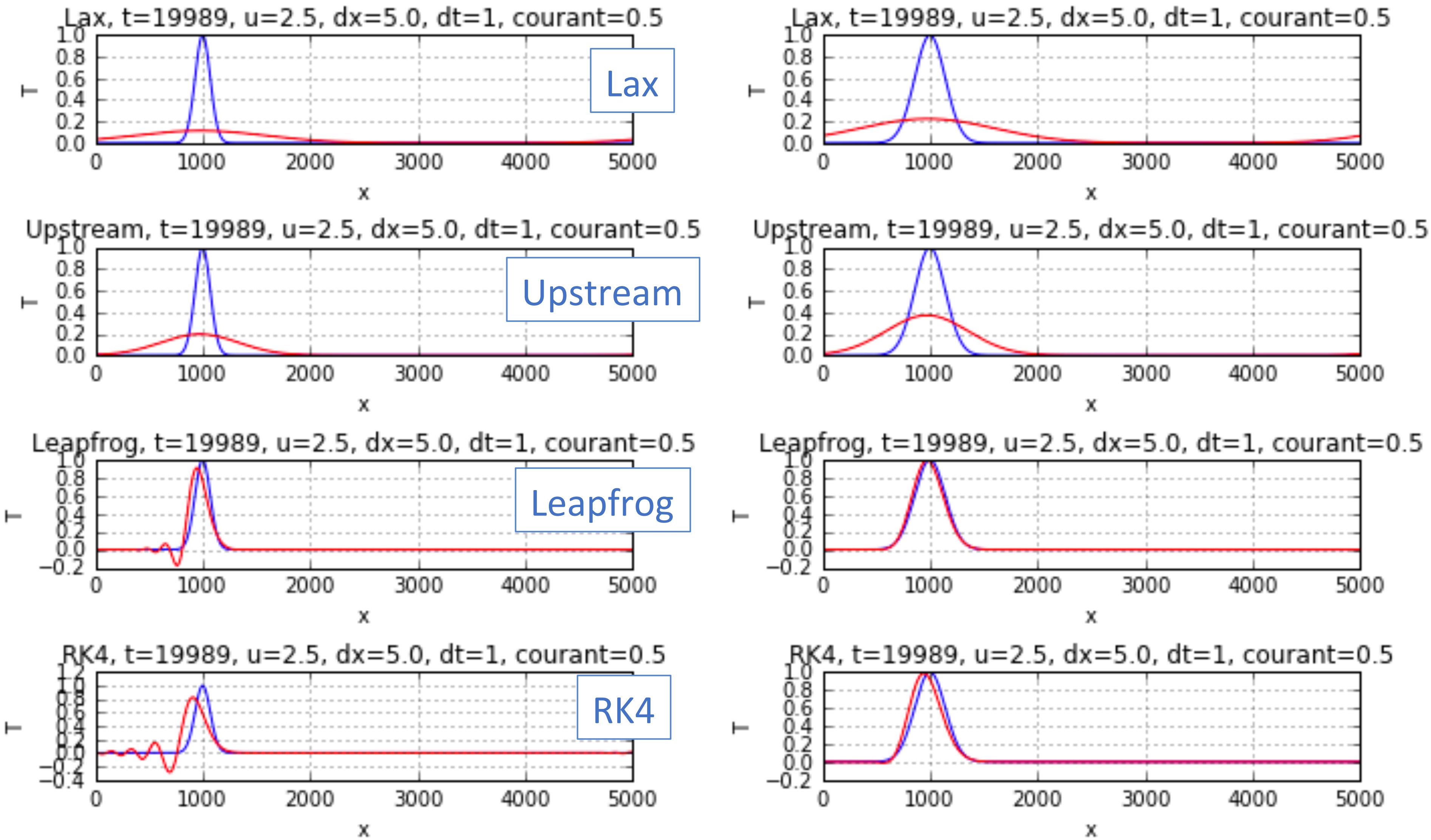
$$y_{n+1} = y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + \Delta t$$

# Resposta a diferentes escalas espaciais

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x},\ u = const$$

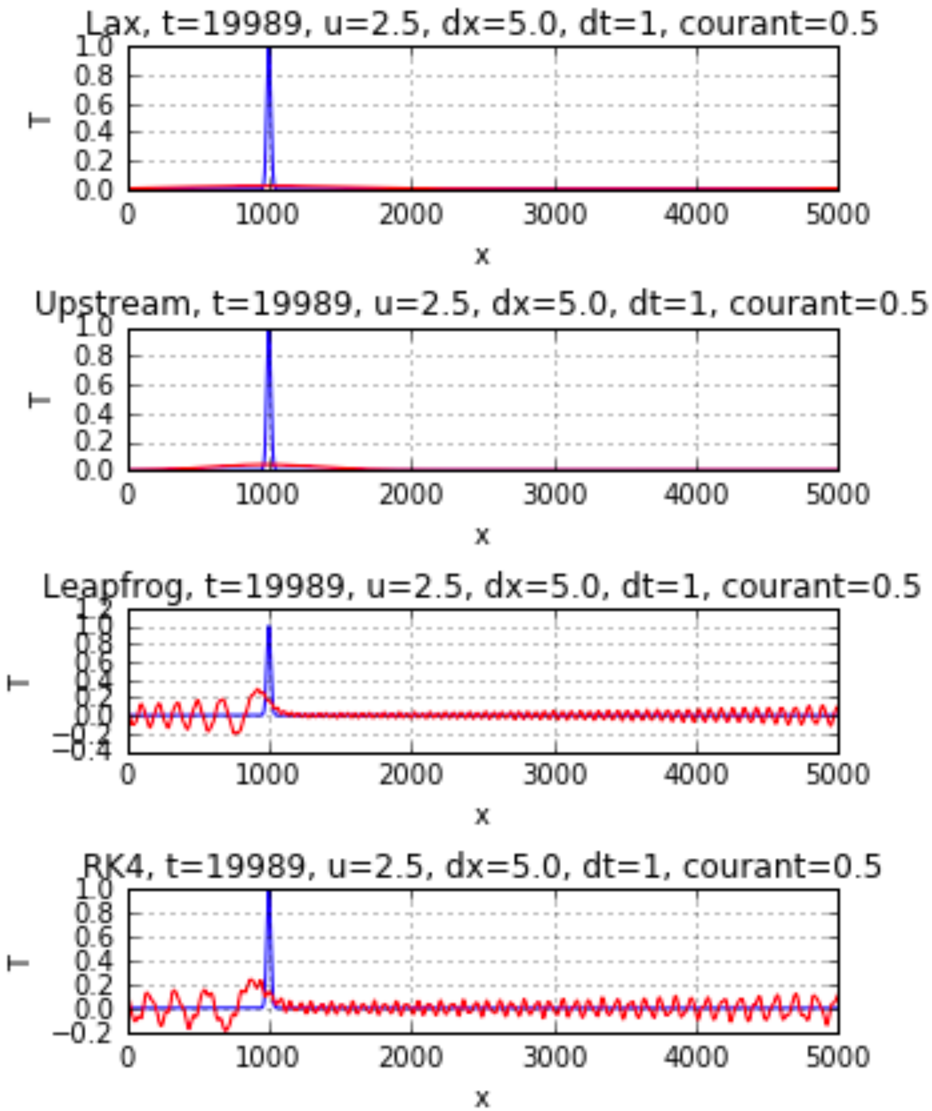
L=100 L=200



# Perturbação muito localizada

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x}, u = const$$

L=20



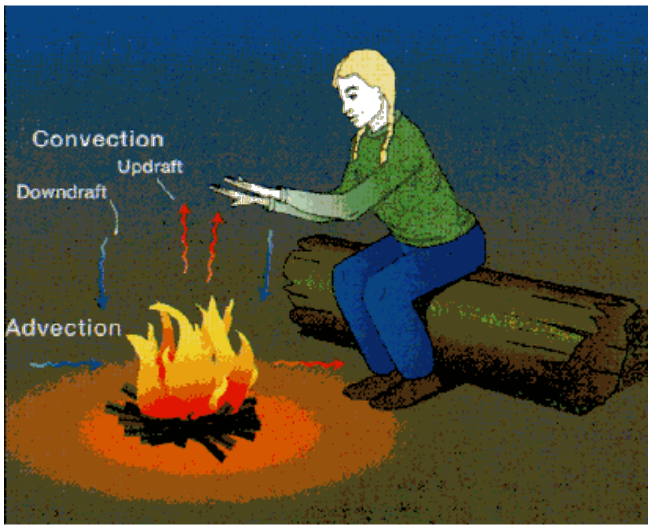

Lax

Upstream

Leapfrog

RK4

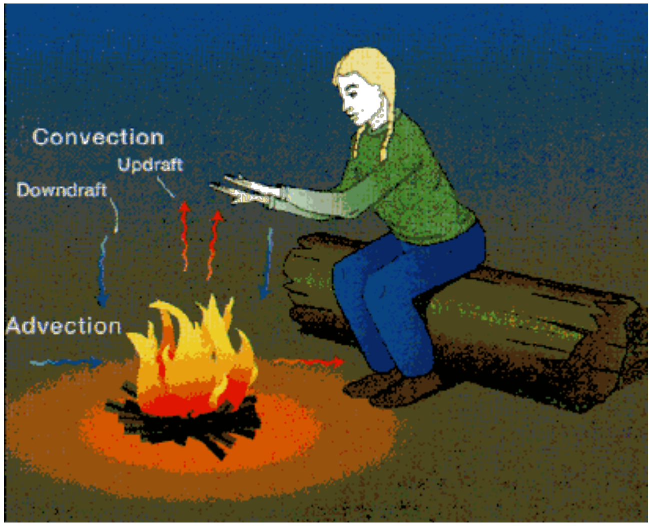
# Advecção 2D




1D: $$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x}$$

2D: $$\frac{\partial h}{\partial t} = -u_x\frac{\partial h}{\partial x} - u_y\frac{\partial h}{\partial y}$$

# Advecção 2D




1D: $$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x}$$

Outra notação:

2D: $$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y}$$

# Advecção 2D

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

$$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y} \implies \boxed{\frac{\partial h}{\partial t} = -\frac{\partial uh}{\partial x} - \frac{\partial vh}{\partial y}}$$

$$\frac{\partial h}{\partial t} = -\nabla(\vec{u}h),\ \vec{v}h \text{ é o fluxo de h}$$

Esta forma é usável em duas condições:

- u, v = constante (advecção linear)
- $\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} = 0$ (fluído incompressível)

# Advecção 2D

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

$$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y} \implies \boxed{\frac{\partial h}{\partial t} = -\frac{\partial uh}{\partial x} - \frac{\partial vh}{\partial y}}$$

Discretização no esquema de Lax:

$$\frac{h^{n+1}_{k,j} - h^n_{k,j}}{\Delta t} = -\frac{u^n_{k+1,j}h^n_{k+1,j} - u^n_{k-1,j}h^n_{k-1,j}}{2\Delta x} - \frac{v^n_{k,j+1}h^n_{k,j+1} - v^n_{k,j-1}h^n_{k,j-1}}{2\Delta y}$$

Diferença avançada no tempo

Diferença centrada no espaço em x

Diferença centrada no espaço em y

# Advecção 2D

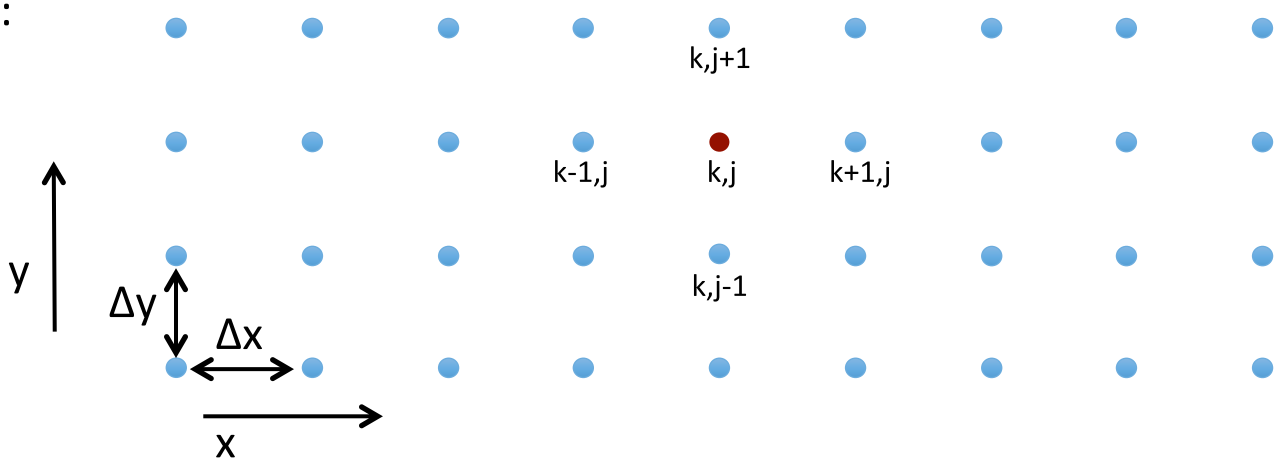
$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

$$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y} \implies \boxed{\frac{\partial h}{\partial t} = -\frac{\partial uh}{\partial x} - \frac{\partial vh}{\partial y}}$$

Discretização no esquema de Lax:

$$\frac{h^{n+1}_{k,j} - h^n_{k,j}}{\Delta t} = -\frac{u^n_{k+1,j}h^n_{k+1,j} - u^n_{k-1,j}h^n_{k-1,j}}{2\Delta x} - \frac{v^n_{k,j+1}h^n_{k,j+1} - v^n_{k,j-1}h^n_{k,j-1}}{2\Delta y}$$

t=n:



# Advecção 2D

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

$$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y} \implies \frac{\partial h}{\partial t} = -\frac{\partial uh}{\partial x} - \frac{\partial vh}{\partial y}$$

Discretização no esquema de Lax:

$$\frac{h_{k,j}^{n+1} - h_{k,j}^{n}}{\Delta t} = -\frac{u_{k+1,j}^{n}h_{k+1,j}^{n} - u_{k-1,j}^{n}h_{k-1,j}^{n}}{2\Delta x} - \frac{v_{k,j+1}^{n}h_{k,j+1}^{n} - v_{k,j-1}^{n}h_{k,j-1}^{n}}{2\Delta y}$$

$$h_{k,j}^{n+1} - h_{k,j}^{n} = -\Delta t\frac{u_{k+1,j}^{n}h_{k+1,j}^{n} - u_{k-1,j}^{n}h_{k-1,j}^{n}}{2\Delta x} - \Delta t\frac{v_{k,j+1}^{n}h_{k,j+1}^{n} - v_{k,j-1}^{n}h_{k,j-1}^{n}}{2\Delta y}$$

$$h_{k,j}^{n+1} = h_{k,j}^{n} - \Delta t\frac{u_{k+1,j}^{n}h_{k+1,j}^{n} - u_{k-1,j}^{n}h_{k-1,j}^{n}}{2\Delta x} - \Delta t\frac{v_{k,j+1}^{n}h_{k,j+1}^{n} - v_{k,j-1}^{n}h_{k,j-1}^{n}}{2\Delta y}$$

$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^{n}+h_{k+1,j}^{n}+h_{k,j-1}^{n}+h_{k,j+1}^{n}) - \Delta t\frac{u_{k+1,j}^{n}h_{k+1,j}^{n} - u_{k-1,j}^{n}h_{k-1,j}^{n}}{2\Delta x} - \Delta t\frac{v_{k,j+1}^{n}h_{k,j+1}^{n} - v_{k,j-1}^{n}h_{k,j-1}^{n}}{2\Delta y}$$

# Advecção 2D

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

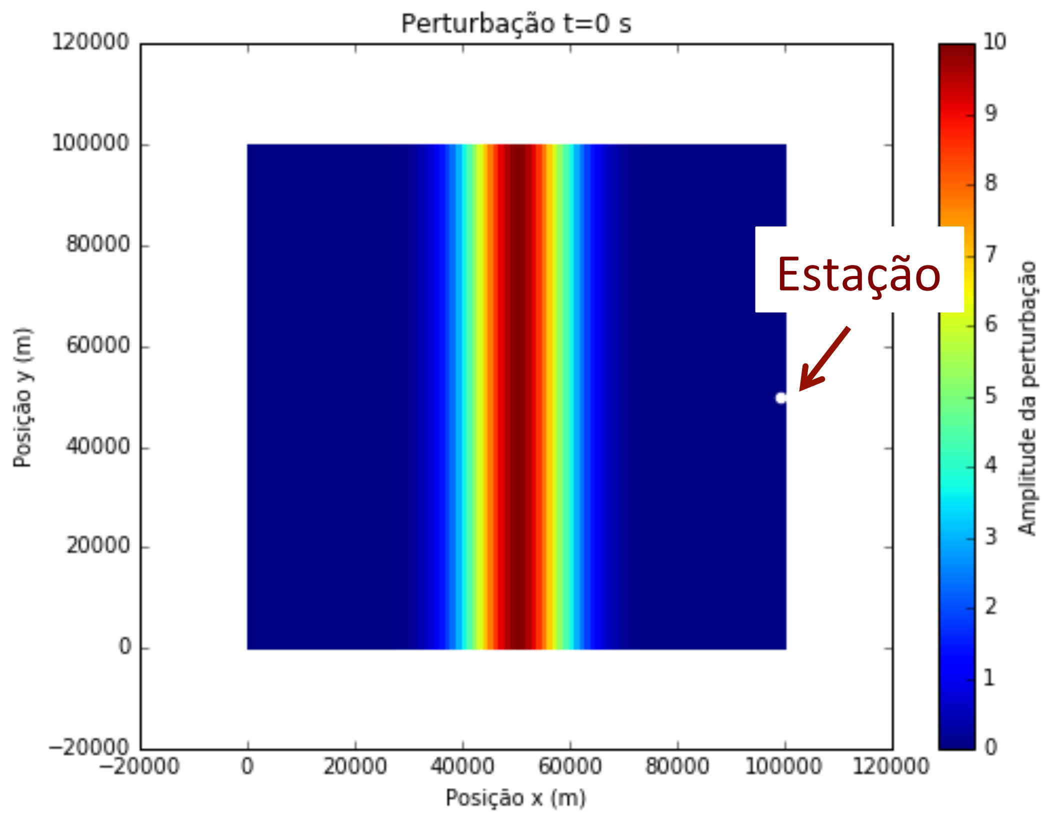
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

plt.rcParams['figure.figsize'] = 10, 6

#%% Parâmetros

# discretização
nt=2000; nx=101; ny=101           # número de pontos no tempo e espaço (x e y)
passo=10;                         # intervalo ebtre figuras
dx=1000.; dy=1000.;               # espaçamento dos pontos no espaço (x e y)
x = np.arange(0,nx)*dx            # vector de pontos no espaço (x)
y = np.arange(0,ny)*dy            # vector de pontos no espaço (y)

xmin=min(x); xmax=max(x)          # valor mínimo e máximo do vector x
ymin=min(y); ymax=max(y)          # valor mínimo e máximo do vector y

xx=np.zeros([nx,ny])              # matriz 2D de valores x (posição em x)
yy=np.zeros([nx,ny])              # matriz 2D de valores y (posição em y)
for ix in range(nx):
    for iy in range(ny):
        xx[ix,iy] = x[ix]
        yy[ix,iy] = y[iy]

# estação
ixStation = nx-2                  # posição da estação, coordenada x
iyStation = ny/2                  # posição da estação, coordenada y
hStation = np.zeros(nt)           # valores da variável h na estação
```

# Advecção 2D

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

plt.rcParams['figure.figsize'] = 10, 6

#%% Parâmetros

# discretização
nt=2000; nx=101; ny=101
passo=10;
dx=1000.; dy=1000.;
x = np.arange(0,nx)*dx
y = np.arange(0,ny)*dy

xmin=min(x); xmax=max(x)
ymin=min(y); ymax=max(y)

xx=np.zeros([nx,ny])
yy=np.zeros([nx,ny])
for ix in range(nx):
    for iy in range(ny):
        xx[ix,iy] = x[ix]
        yy[ix,iy] = y[iy]

# estação
ixStation = nx-2
iyStation = ny/2
hStation = np.zeros(nt)
```

Sinal na estação



# Advecção 2D

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

```
# velocidades
uspeed=10                                  # velocidade de propagação da direcção x
vspeed=0                                   # velocidade de propagação da direcção y
ws = np.sqrt(uspeed**2 + vspeed**2)        # velocidade de propagação total

dt = 0.68 * dx / ws                        # espaçamento entre pontos no tempo
dt2dx = dt/(2*dx)                          # dt/(2 dx)
dt2dy = dt/(2*dy)                          # dt/(2 dy)

courant = ws*dt/dx                         # número de courant

u = uspeed * np.ones([nx,ny])          # matriz de velocidades (x)
uP = np.zeros([nx,ny])                 # matriz de velocidades futuras (x)
uP[:] = u[:]                           # inicialização da matriz de velocidades futuras (x)

v = vspeed * np.ones([nx,ny])          # matriz de velocidades (y)
vP = np.zeros([nx,ny])                 # matriz de velocidades futuras (y)
vP[:] = v[:]                           # inicialização da matriz de velocidades futuras (y)
```

$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^{n}+h_{k+1,j}^{n}+h_{k,j-1}^{n}+h_{k,j+1}^{n}) - \Delta t\frac{u_{k+1,j}^{n}h_{k+1,j}^{n} - u_{k-1,j}^{n}h_{k-1,j}^{n}}{2\Delta x} - \Delta t\frac{v_{k,j+1}^{n}h_{k,j+1}^{n} - v_{k,j-1}^{n}h_{k,j-1}^{n}}{2\Delta y}$$

# Advecção 2D

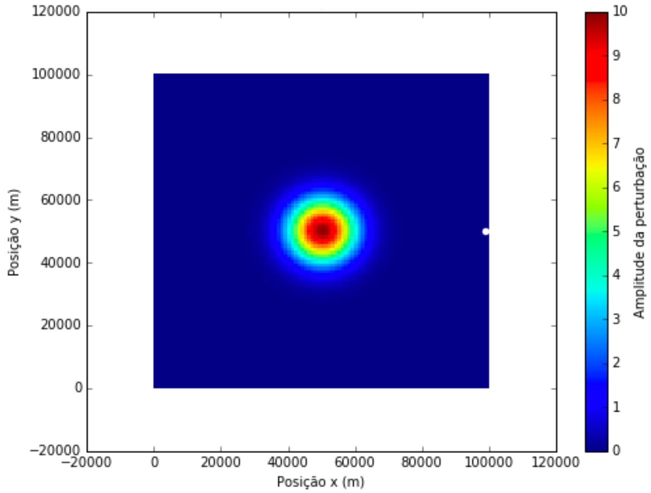
$$h = h_0 e^{-\left(\frac{x-x_0}{L_x}\right)^2 - \left(\frac{y-y_0}{L_y}\right)^2}$$

```
#%% Definição de h inicial (perturbação/sinal a propagar/advectar)

hJUMP=10                     # amplitude inicial do sinal
xJUMP=(xmax+xmin)/2.         # localização inicial (em x) do sinal (= valor médio/central do vector
LxJUMP=10.*dx                # largura inicial do sinal (x)
yJUMP=(ymax+ymin)/2.         # localização inicial (em y) do sinal (= valor médio/central do vector
LyJUMP=2000.*dy              # largura inicial do sinal (y)

# perturbação/sinal inicial:
h = hJUMP*np.exp(-((xx-xJUMP)/LxJUMP)**2 - ((yy-yJUMP)/LyJUMP)**2)
hP = np.zeros([nx,ny])                          # sinal no passo temporal seguinte
hStation[0] = h[ixStation, iyStation]           # localização da estação (x,y)

# plot
plt.close()
plt.rcParams['figure.figsize'] = 8, 6

# plt.contourf(xx,yy,h)
plt.pcolor(xx,yy,h)                  # figura 2D do sinal inicial
plt.clim(0,10)                       # limites da barra de cores
cb=plt.colorbar()
cb.set_label(u'Amplitude da perturbação')   # legenda da barra de cores

# marcar a estação com um ponto branco
plt.scatter(xx[ixStation, iyStation], yy[ixStation, iyStation], c='w', edgecolors='w')

plt.xlabel(u'Posição x (m)')
plt.ylabel(u'Posição y (m)')
plt.title(u'Perturbação t=0 s')

plt.savefig('fig/advection2d-lax-t0.png')
```

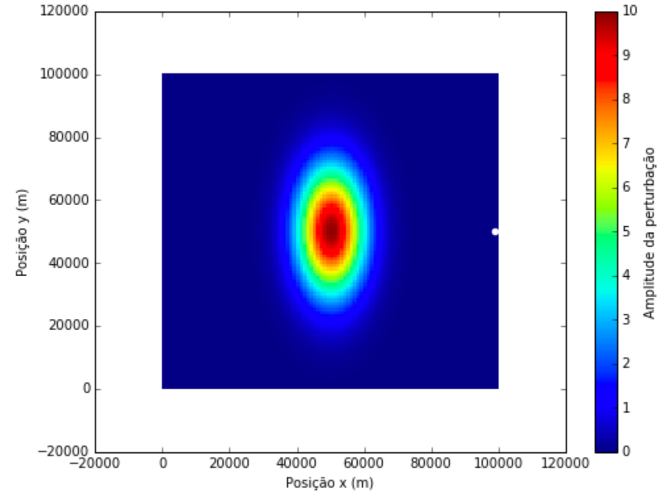
# Advecção 2D

$$h = h_0 e^{-\left(\frac{x-x_0}{L_x}\right)^2 - \left(\frac{y-y_0}{L_y}\right)^2}$$

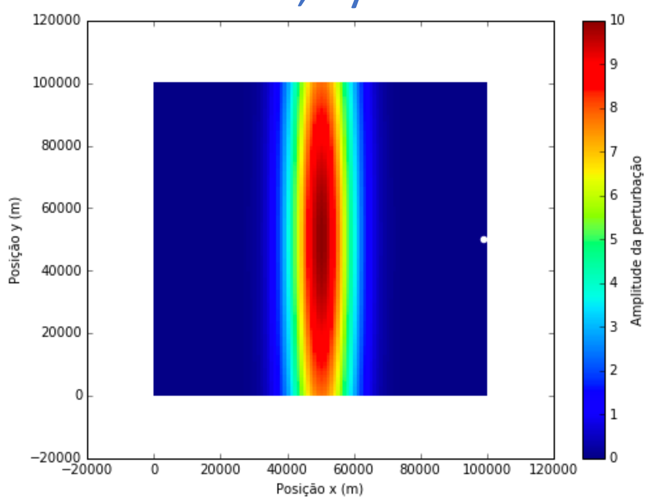
Lx=10, Ly=10



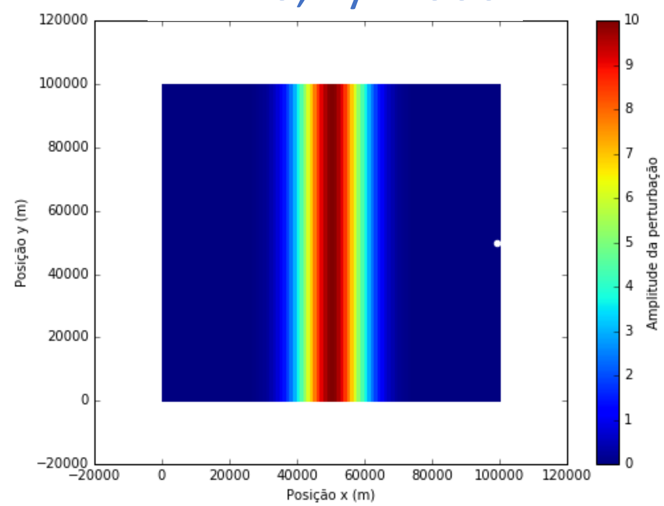

Lx=10, Ly=20




Lx=10, Ly=100

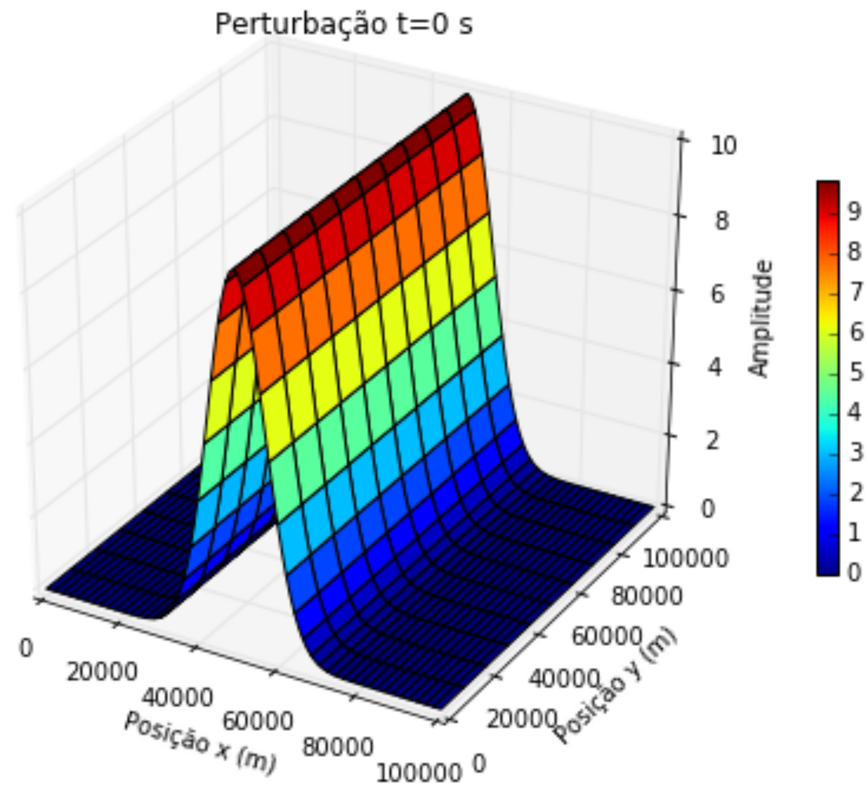



Lx=10, Ly=1000



# Advecção 2D

```
#%% plot3D

fig = plt.figure()
ax = plt.axes(projection='3d')

surf=ax.plot_surface(xx,yy,h, rstride=2, cstride=10, cmap=cm.jet)
ax.set_zlim(0,10)
fig.colorbar(surf, shrink=0.5)

ax.set_xlabel(u'Posição x (m)')
ax.set_ylabel(u'Posição y (m)')
ax.set_zlabel(u'Amplitude')
ax.set_title(u'Perturbação t=0 s')

plt.savefig('fig3d/advection2d-lax-t0.png')
```



$$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y} \implies \boxed{\frac{\partial h}{\partial t} = -\frac{\partial uh}{\partial x} - \frac{\partial vh}{\partial y}}$$

$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t \frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t \frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$

```
#%% Implementação do método LAX, propagar o sinal no tempo

for it in range(1,nt):
    hu = h*u
    hv = h*v

    # Avançar o sinal em todo o domínio excepto nas fronteiras
    for ix in range(1,nx-1):
        for iy in range(1,ny-1):
            hP[ix,iy] = (h[ix-1,iy] + h[ix+1,iy] + h[ix,iy-1] + h[ix,iy+1])/4. \
                - dt2dx * (hu[ix+1,iy] - hu[ix-1,iy]) \
                - dt2dy * (hv[ix,iy+1] - hv[ix,iy-1])
```

# Advecção 2D

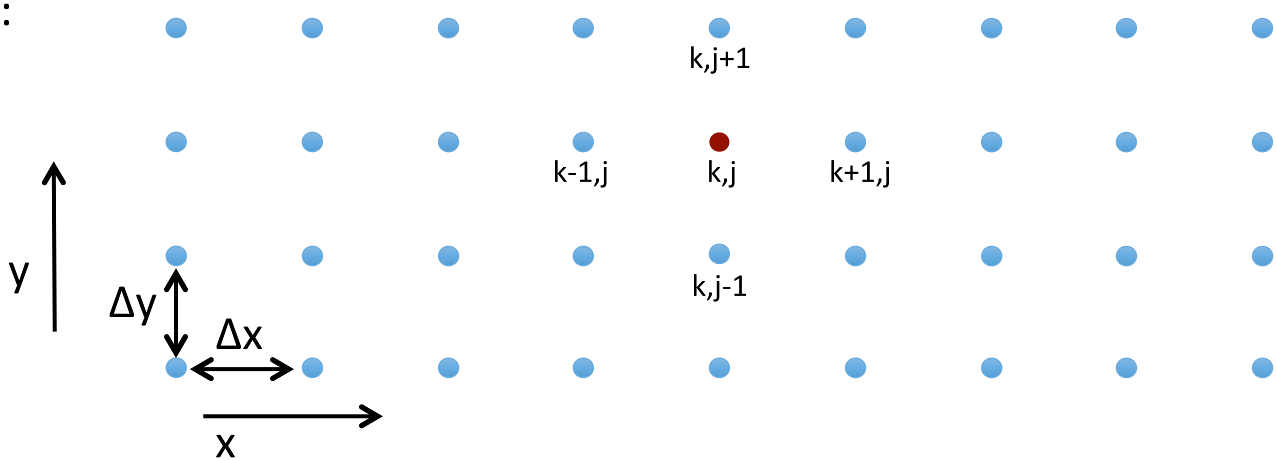
$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

$$\frac{\partial h}{\partial t} = -u\frac{\partial h}{\partial x} - v\frac{\partial h}{\partial y} \quad \Longrightarrow \quad \boxed{\frac{\partial h}{\partial t} = -\frac{\partial uh}{\partial x} - \frac{\partial vh}{\partial y}}$$

Discretização no esquema de Lax:

$$\frac{h_{k,j}^{n+1} - h_{k,j}^{n}}{\Delta t} = -\frac{u_{k+1,j}^{n}h_{k+1,j}^{n} - u_{k-1,j}^{n}h_{k-1,j}^{n}}{2\Delta x} - \frac{v_{k,j+1}^{n}h_{k,j+1}^{n} - v_{k,j-1}^{n}h_{k,j-1}^{n}}{2\Delta y}$$

t=n:



$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t \frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t \frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$

```
# Condições fronteira cíclicas em x
for iy in range(1,ny-1):
    ix=0
    hP[ix,iy] = (h[nx-1,iy] + h[ix+1,iy] + h[ix,iy-1] + h[ix,iy+1])/4. \
        - dt2dx * (hu[ix+1,iy] - hu[nx-1,iy])   \
        - dt2dy * (hv[ix,iy+1] - hv[ix,iy-1])

    ix=nx-1
    hP[ix,iy] = (h[ix-1,iy] + h[0,iy] + h[ix,iy-1] + h[ix,iy+1])/4. \
        - dt2dx * (hu[0,iy] - hu[ix-1,iy]) \
        - dt2dy * (hv[ix,iy+1] - hv[ix,iy-1])
```

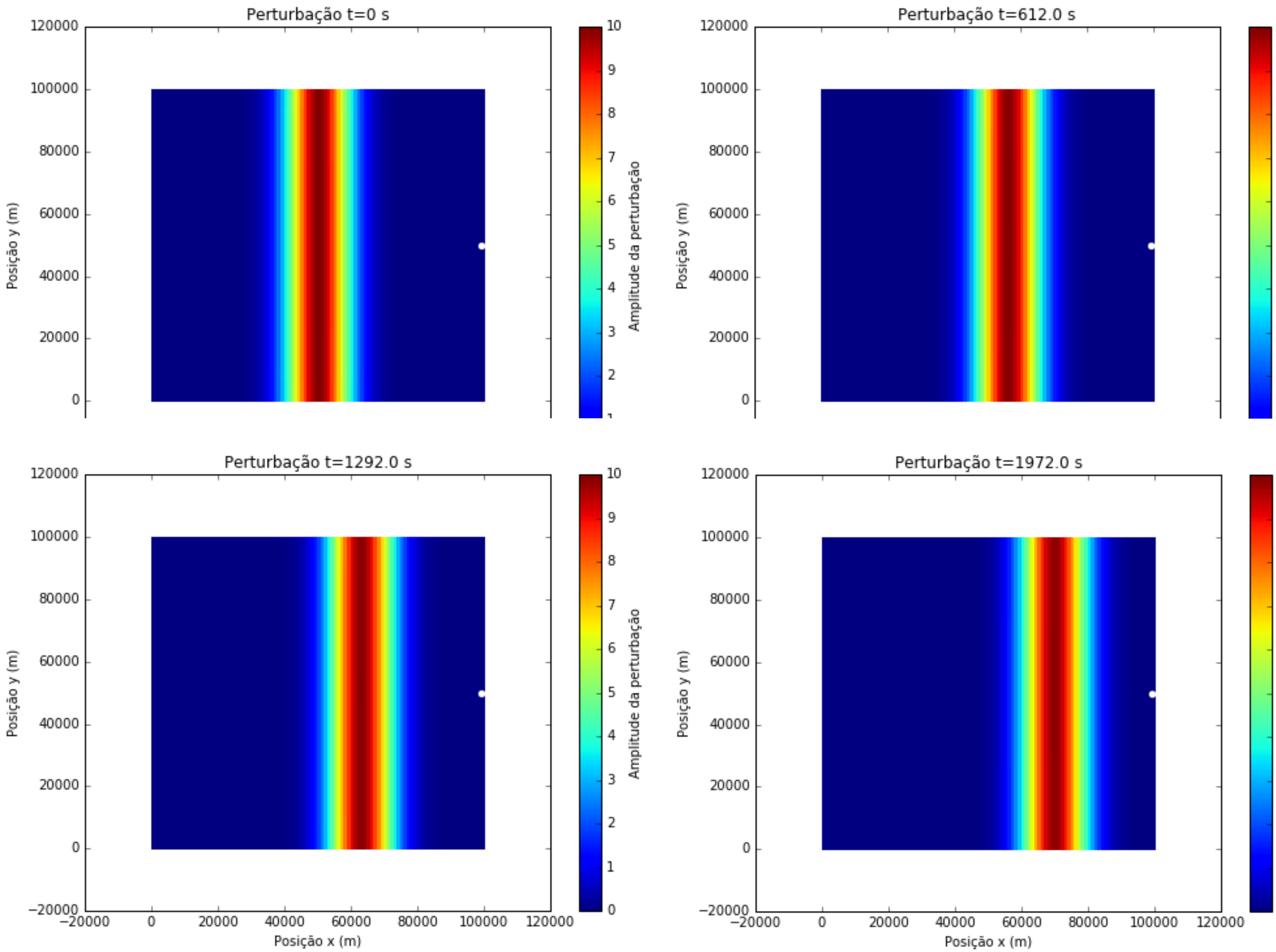
```
# Condições fronteira cíclicas em y
for ix in range(1,nx-1):
    iy=0
    hP[ix,iy] = (h[ix-1,iy] + h[ix+1,iy] + h[ix,ny-1] + h[ix,iy+1])/4. \
        - dt2dx * (hu[ix+1,iy] - hu[ix-1,iy]) \
        - dt2dy * (hv[ix,iy+1] - hv[ix,ny-1])

    iy=ny-1
    hP[ix,iy] = (h[ix-1,iy] + h[ix+1,iy] + h[ix,iy-1] + h[ix,0])/4. \
        - dt2dx * (hu[ix+1,iy] - hu[ix-1,iy]) \
        - dt2dy * (hv[ix,0] - hv[ix,iy-1])
```

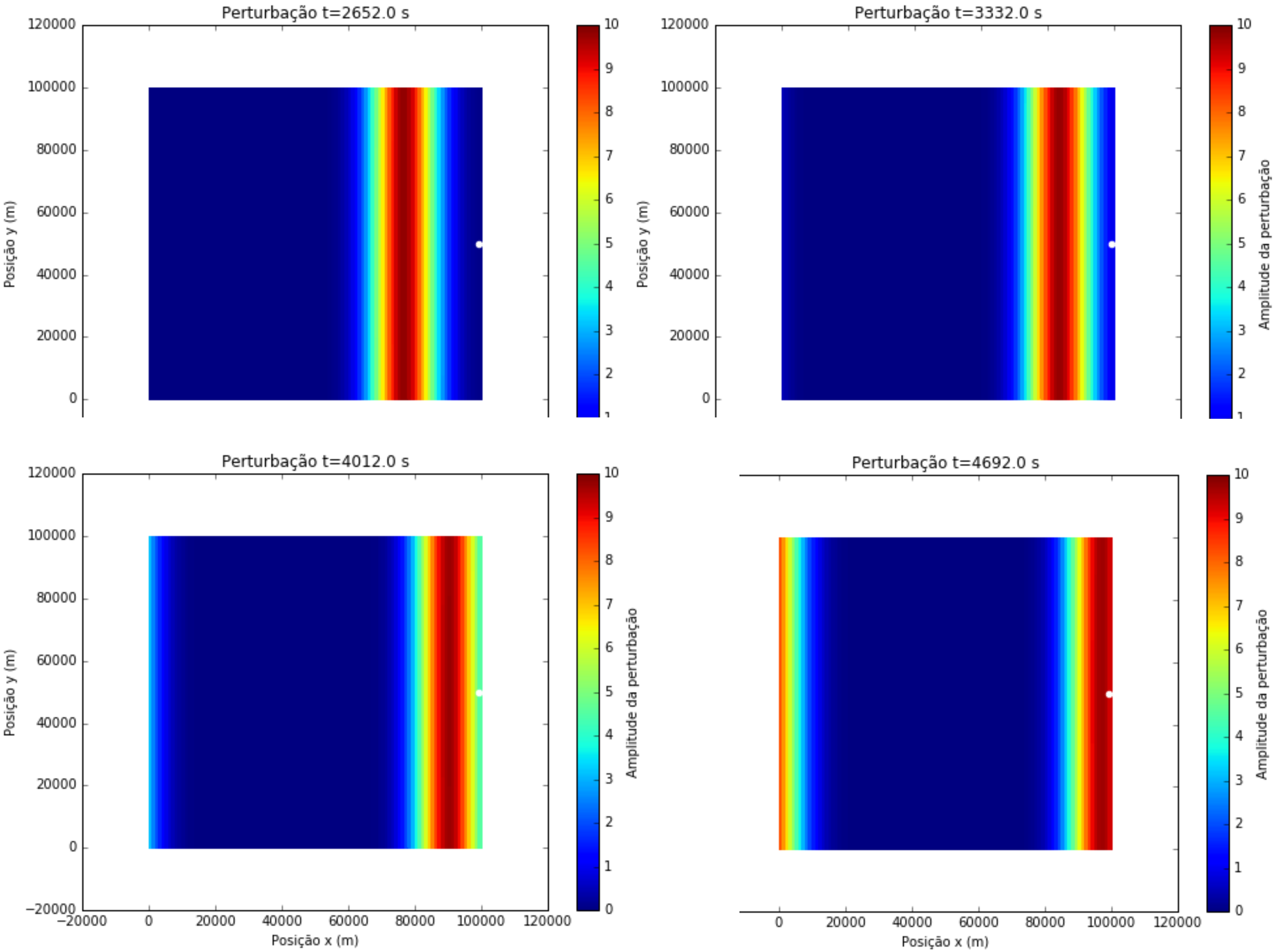
$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t \frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t \frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$

```python
# Condições fronteira nos cantos
ix=0; ixm=nx-1; ixp=1
iy=0; iym=ny-1; iyp=1
hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
    - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
    - dt2dy * (hv[ix,iyp] - hv[ix,iym])

ix=nx-1; ixm=nx-2; ixp=0
iy=0; iym=ny-1; iyp=1
hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
    - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
    - dt2dy * (hv[ix,iyp] - hv[ix,iym])

ix=0; ixm=nx-1; ixp=1
iy=ny-1; iym=ny-2; iyp=0
hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
    - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
    - dt2dy * (hv[ix,iyp] - hv[ix,iym])

ix=nx-1; ixm=nx-2; ixp=0
iy=ny-1; iym=ny-2; iyp=0
hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
    - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
    - dt2dy * (hv[ix,iyp] - hv[ix,iym])
```

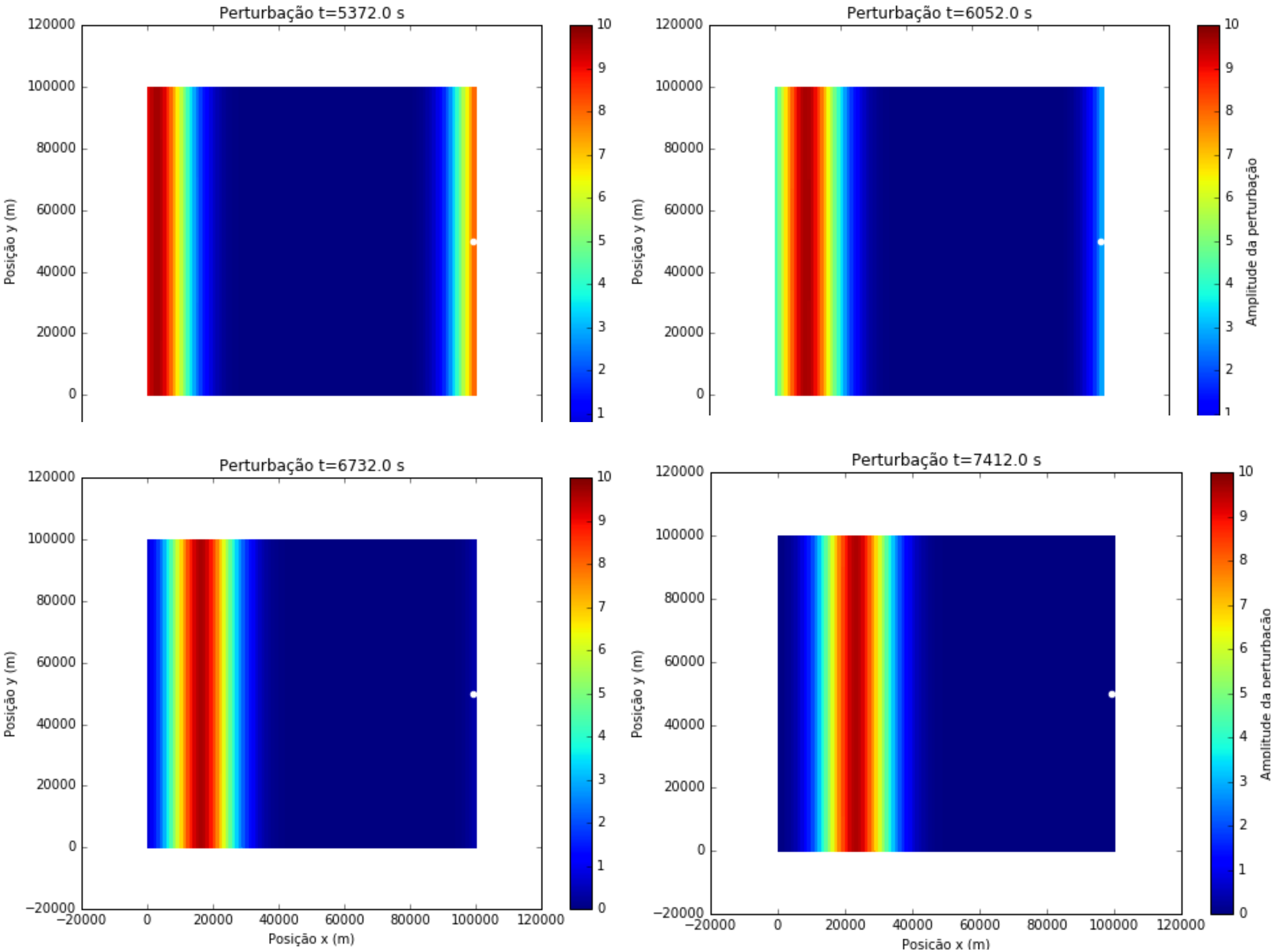
$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t \frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t \frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$

```
    # actualizar o sinal
    h[:]=hP[:]

    # Fazer figuras com as soluções, de 10 em 10 passos no tempo
    if (it+1)%passo==0:
#    if it<3*passo:
        plt.close()
        plt.rcParams['figure.figsize'] = 8, 6

#        plt.contourf(xx,yy,h)
        plt.pcolor(xx,yy,h)
        plt.clim(0,10)
        cb=plt.colorbar()
        cb.set_label(u'Amplitude da perturbação')
        plt.scatter(xx[ixStation, iyStation], yy[ixStation, iyStation], c='w', edgecolors='w')

        plt.xlabel(u'Posição x (m)')
        plt.ylabel(u'Posição y (m)')
        plt.title(u'Perturbação t='+ str(it*dt) +' s')

        plt.savefig('fig/advection2d-lax-t'+ str(it) +'.png')

    # guardar o sinal na estação
    hStation[it] = h[ixStation, iyStation]

# figura com a evolução do sinal na estação
plt.close()
plt.rcParams['figure.figsize'] = 8, 6
plt.plot(np.arange(0,nt)*dt/60., hStation)

plt.ylabel(u'Amplitude')
plt.xlabel(u'Tempo (min)')
plt.title(u'Estação ix='+ str(ixStation) +', iy='+ str(iyStation) +', courant='+ str(courant))
plt.savefig('advection2d-lax-station-courant'+ str(courant) +'.png')
```


Perturbação t=0 s
Perturbação t=612.0 s
Perturbação t=1292.0 s
Perturbação t=1972.0 s
Posição y (m)
Posição x (m)
Amplitude da perturbação
120000
100000
80000
60000
40000
20000
0
−20000
−20000
0
20000
40000
60000
80000
100000
120000
10
9
8
7
6
5
4
3
2
1
0



Perturbação t=2652.0 s
Perturbação t=3332.0 s
Perturbação t=4012.0 s
Perturbação t=4692.0 s
Posição y (m)
Posição x (m)
Amplitude da perturbação

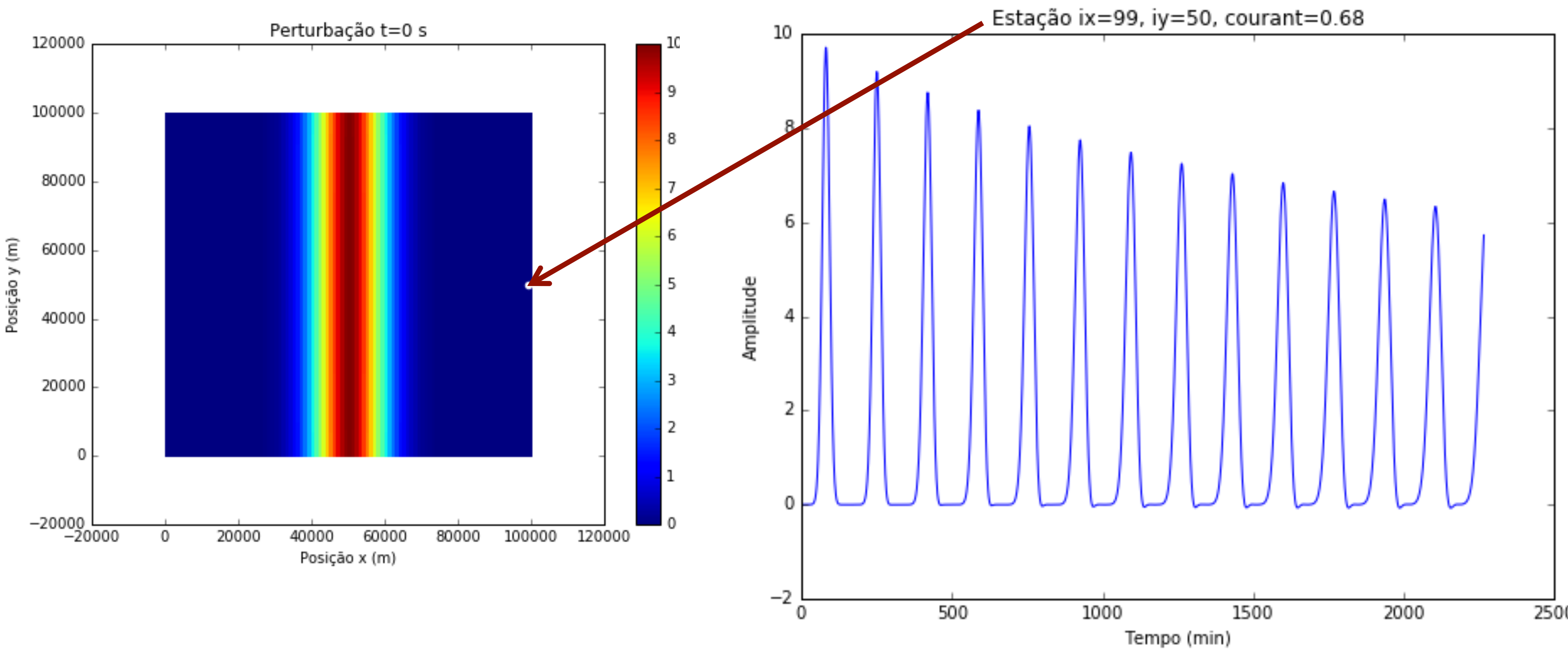


Perturbação t=5372.0 s
Perturbação t=6052.0 s
Perturbação t=6732.0 s
Perturbação t=7412.0 s
Posição y (m)
Posição x (m)
Amplitude da perturbação

# Advecção 2D, Lax

$$\frac{\partial T}{\partial t} = -u\frac{\partial T}{\partial x} - v\frac{\partial T}{\partial y}$$

$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t\frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t\frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$



$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t \frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t \frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$

```
    # Condições fronteira cíclicas em x
    for iy in range(1,ny-1):
        ix=0
        hP[ix,iy] = (h[nx-1,iy] + h[ix+1,iy] + h[ix,iy-1] + h[ix,iy+1])/4. \
            - dt2dx * (hu[ix+1,iy] - hu[nx-1,iy])   \
            - dt2dy * (hv[ix,iy+1] - hv[ix,iy-1])

        ix=nx-1
        hP[ix,iy] = (h[ix-1,iy] + h[0,iy] + h[ix,iy-1] + h[ix,iy+1])/4. \
            - dt2dx * (hu[0,iy] - hu[ix-1,iy]) \
            - dt2dy * (hv[ix,iy+1] - hv[ix,iy-1])

#        # Condições fronteira abertas em x
#        hP[0,iy]=hP[1,iy]
#        hP[nx-1,iy]=hP[nx-2,iy]

    # Condições fronteira cíclicas em y
    for ix in range(1,nx-1):
        iy=0
        hP[ix,iy] = (h[ix-1,iy] + h[ix+1,iy] + h[ix,ny-1] + h[ix,iy+1])/4. \
            - dt2dx * (hu[ix+1,iy] - hu[ix-1,iy]) \
            - dt2dy * (hv[ix,iy+1] - hv[ix,ny-1])

        iy=ny-1
        hP[ix,iy] = (h[ix-1,iy] + h[ix+1,iy] + h[ix,iy-1] + h[ix,0])/4. \
            - dt2dx * (hu[ix+1,iy] - hu[ix-1,iy]) \
            - dt2dy * (hv[ix,0] - hv[ix,iy-1])

#        # Condições fronteira abertas em y
#        hP[ix,0]=hP[ix,1]
#        hP[ix,ny-1]=hP[ix,ny-2]
```

$$h_{k,j}^{n+1} = \frac{1}{4}(h_{k-1,j}^n + h_{k+1,j}^n + h_{k,j-1}^n + h_{k,j+1}^n) - \Delta t \frac{u_{k+1,j}^n h_{k+1,j}^n - u_{k-1,j}^n h_{k-1,j}^n}{2\Delta x} - \Delta t \frac{v_{k,j+1}^n h_{k,j+1}^n - v_{k,j-1}^n h_{k,j-1}^n}{2\Delta y}$$

```
    # Condições fronteira nos cantos
    ix=0; ixm=nx-1; ixp=1
    iy=0; iym=ny-1; iyp=1
    hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
        - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
        - dt2dy * (hv[ix,iyp] - hv[ix,iym])

    ix=nx-1; ixm=nx-2; ixp=0
    iy=0; iym=ny-1; iyp=1
    hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
        - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
        - dt2dy * (hv[ix,iyp] - hv[ix,iym])

    ix=0; ixm=nx-1; ixp=1
    iy=ny-1; iym=ny-2; iyp=0
    hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
        - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
        - dt2dy * (hv[ix,iyp] - hv[ix,iym])

    ix=nx-1; ixm=nx-2; ixp=0
    iy=ny-1; iym=ny-2; iyp=0
    hP[ix,iy] = 1./4. * (h[ixm,iy] + h[ixp,iy] + h[ix,iym] + h[ix,iyp]) \
        - dt2dx * (hu[ixp,iy] - hu[ixm,iy]) \
        - dt2dy * (hv[ix,iyp] - hv[ix,iym])

#    # Condições fronteira abertas nos cantos
#    hP[0,0]=hP[1,1]
#    hP[0,ny-1]=hP[1,ny-2]
#    hP[nx-1,0]=hP[nx-2,1]
#    hP[nx-1,ny-1]=hP[nx-2,ny-2]
```