Modelação Numérica 2017

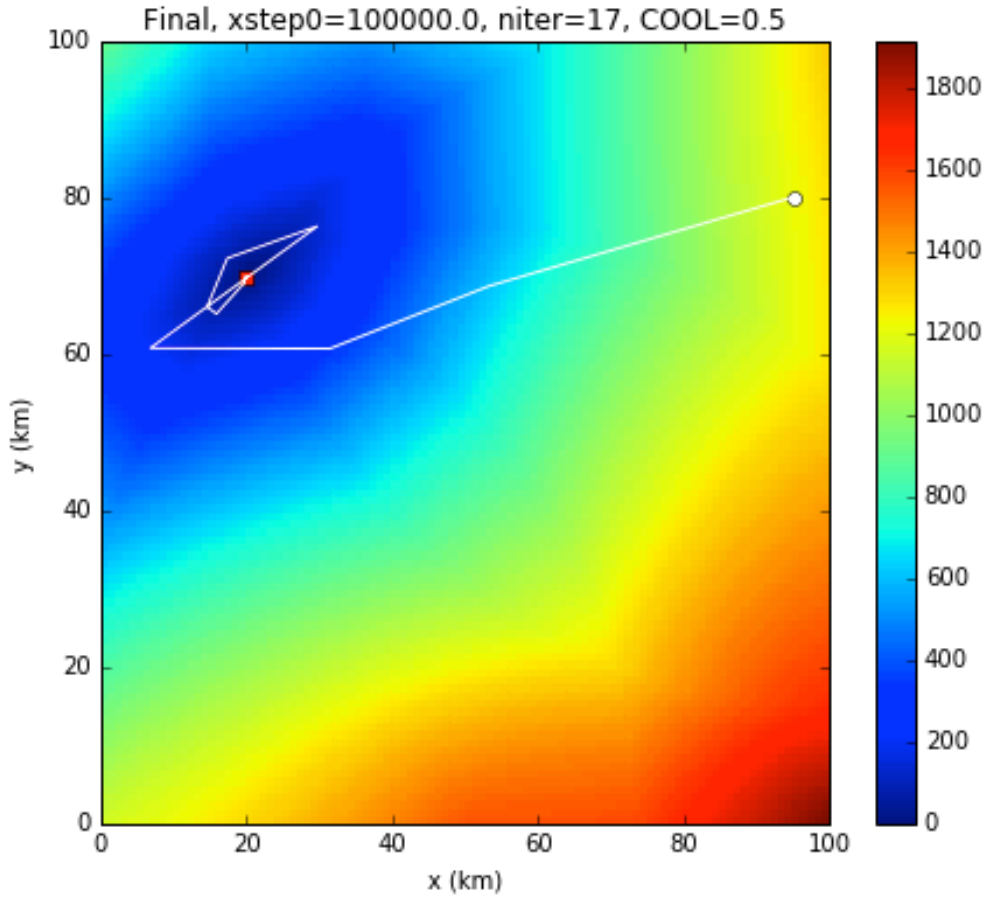
Aula 19, 3/Maio

- Estimativa de parâmetros e optimização: Simulated Annealing.

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

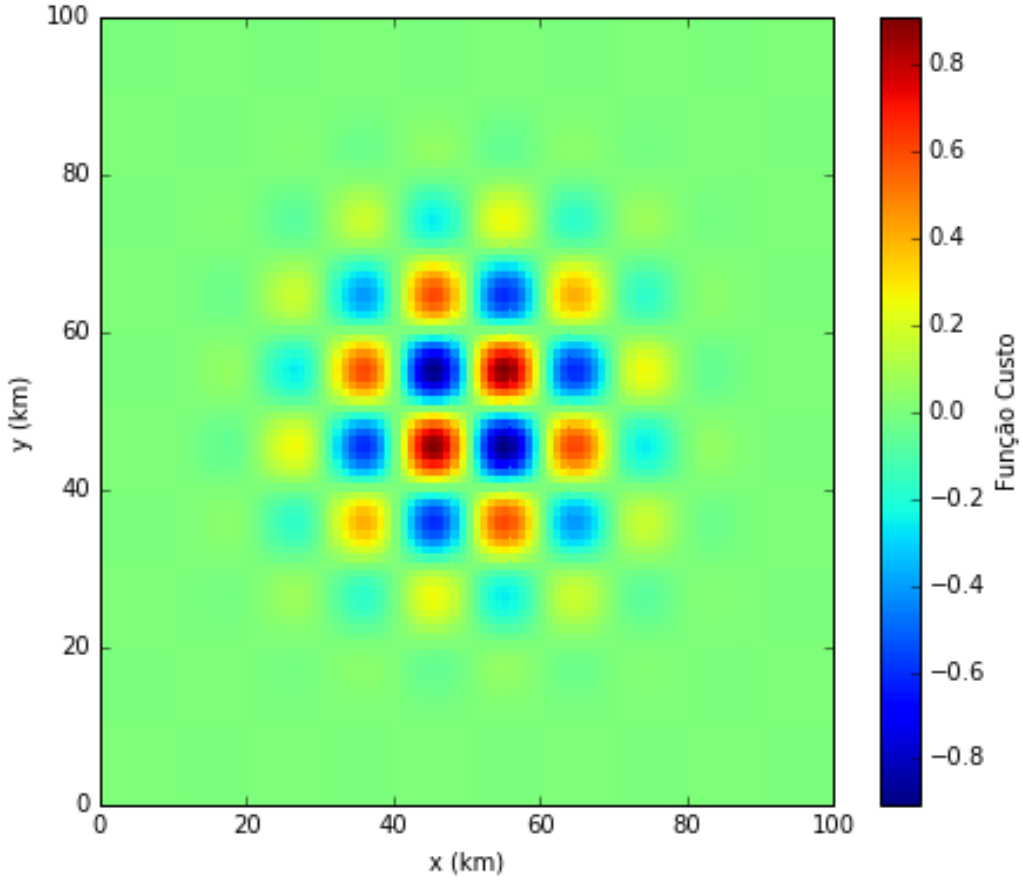
Downslope Monte Carlo

- Função de custo fácil: uma única bacia de atracção.



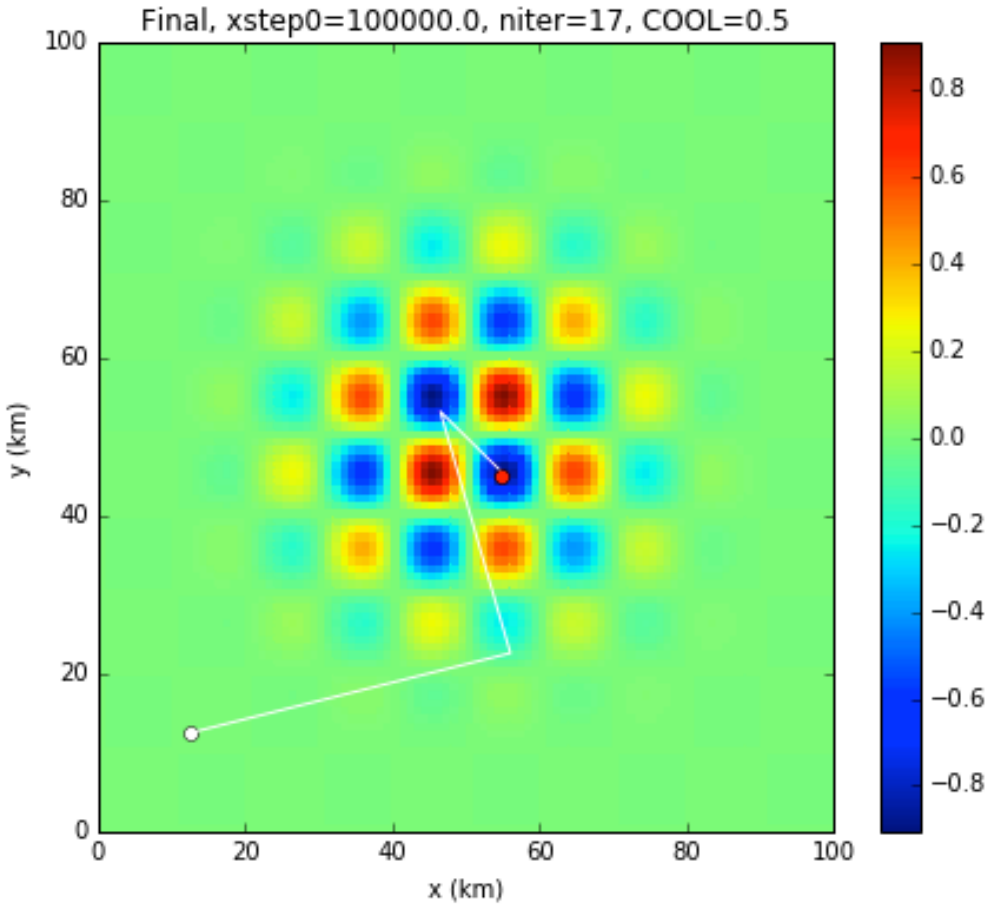
Downslope Monte Carlo

- E se a função de custo for complicada?



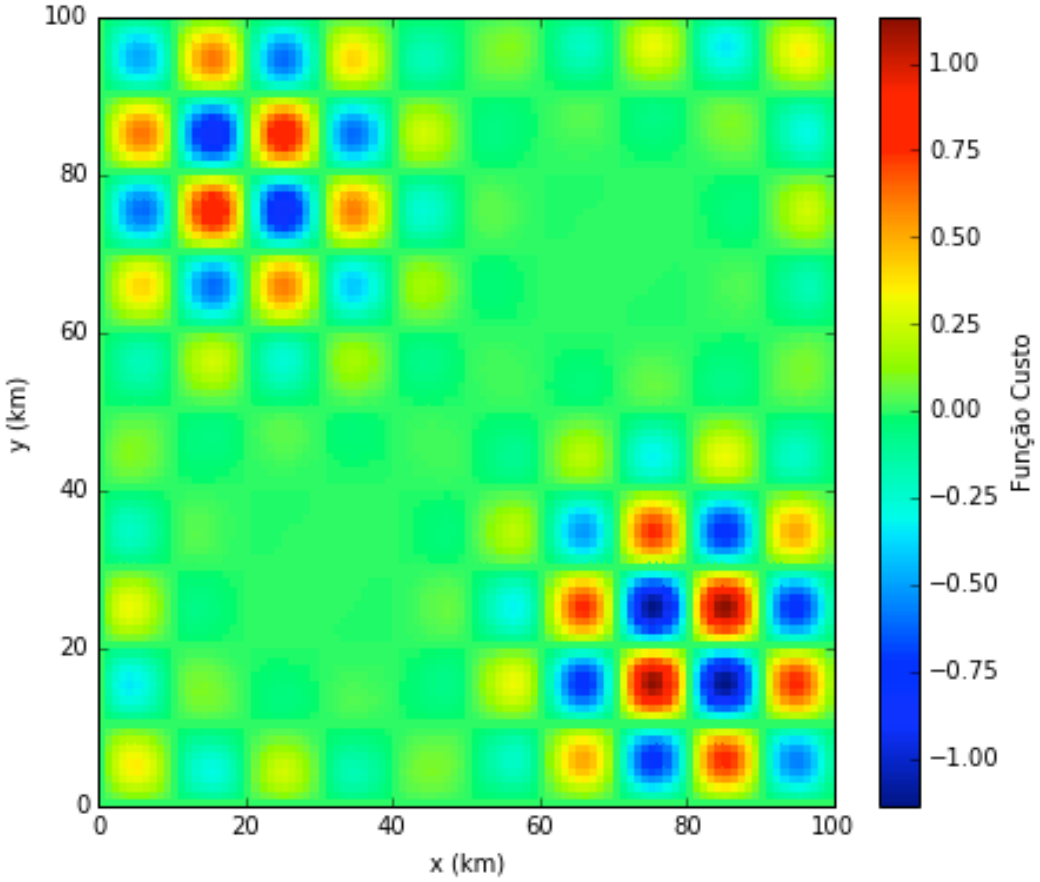
Downslope Monte Carlo

- E se a função de custo for complicada?



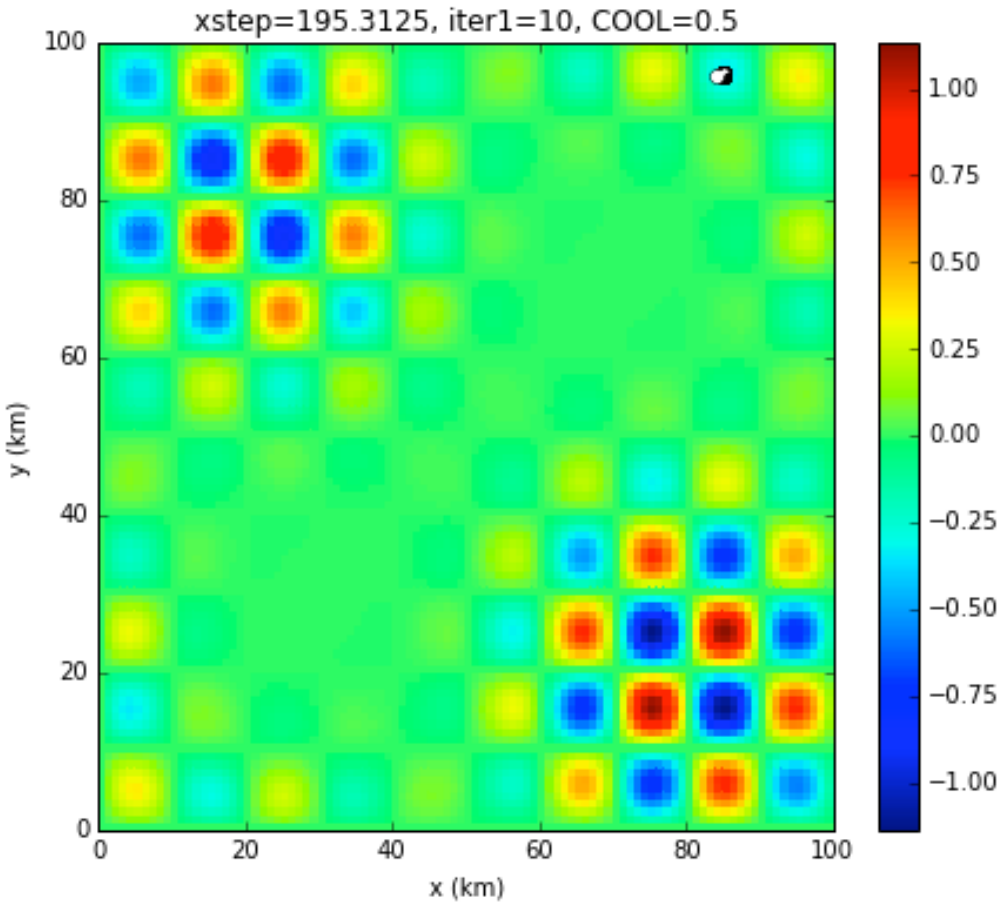
Downslope Monte Carlo

- E se a função de custo for complicada?



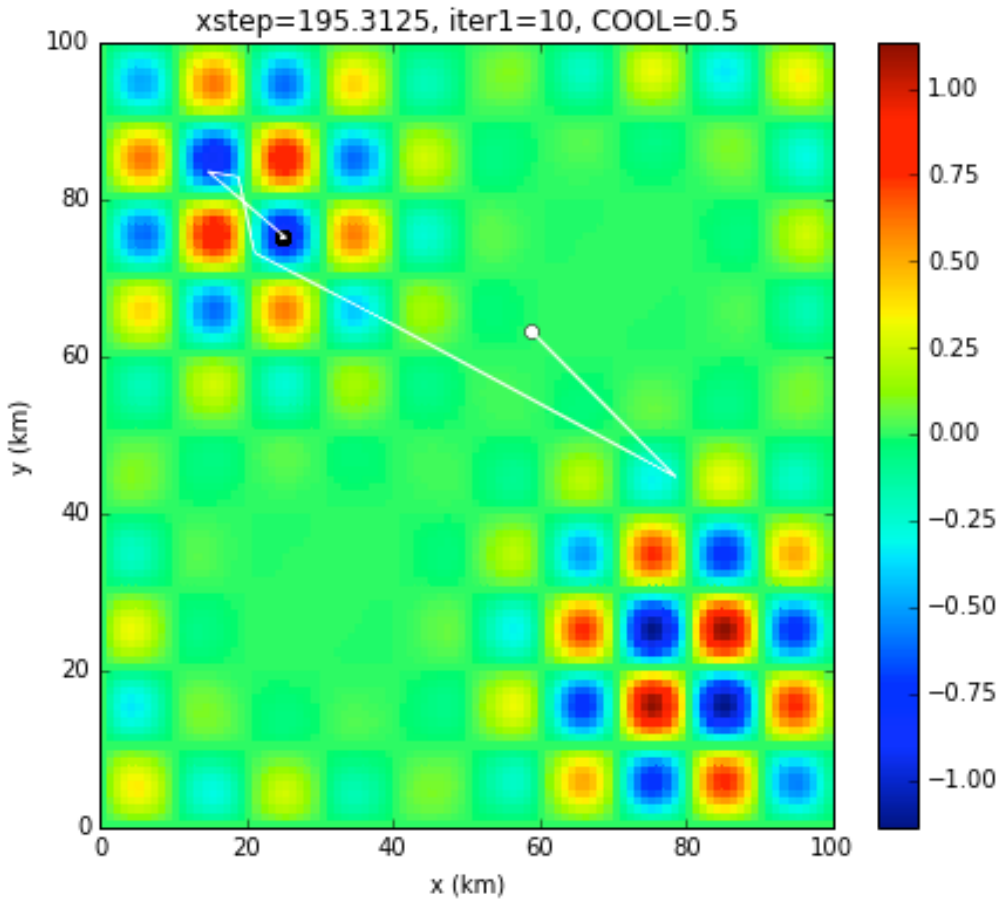
Downslope Monte Carlo

- E se a função de custo for complicada?



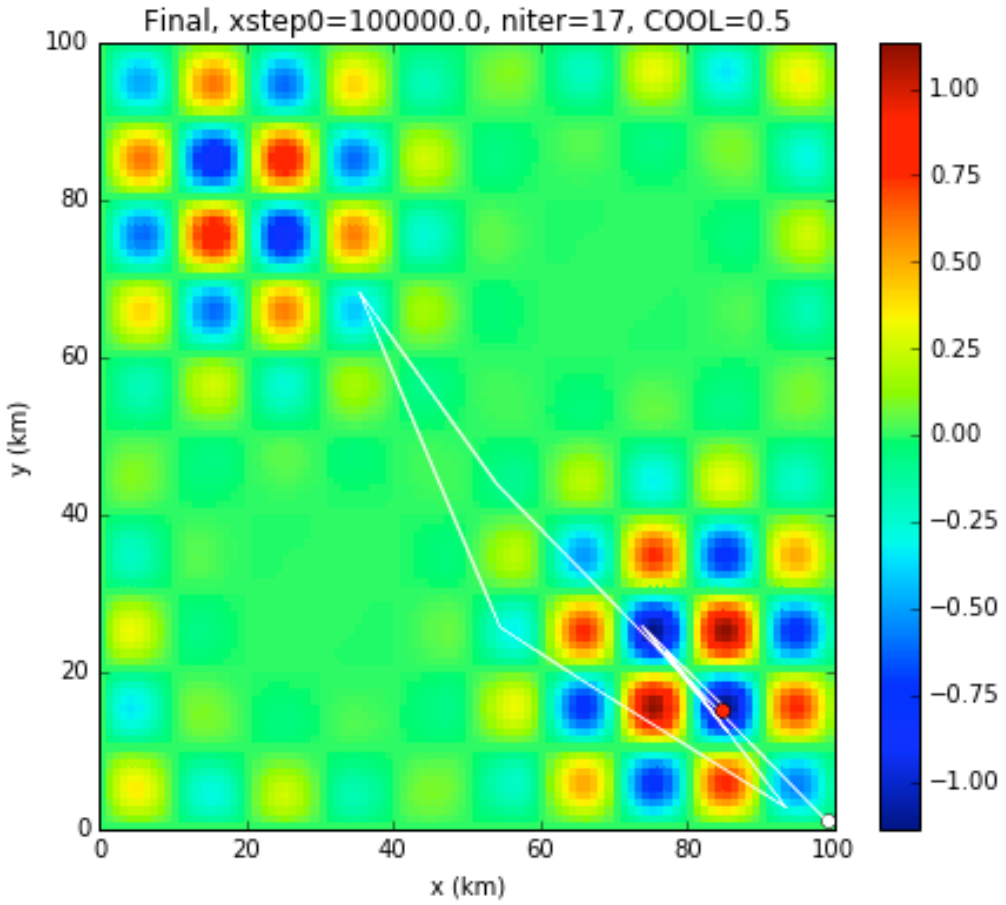
Downslope Monte Carlo

- E se a função de custo for complicada?



Downslope Monte Carlo

- E se a função de custo for complicada?



Comentários

- É possível garantir convergência com o método Downslope MC.
- Basta que:

    1. O salto da primeira iteração cubra todo o domínio.

    2. Garantir um número suficiente de tentativas (> área total/área da bacia de atração), se isso for possível e se essas áreas forem conhecidas...

- Os insucessos anteriores sugerem, no entanto, uma alteração ao método proposto que evite o bloqueio do caminho aleatório num mínimo local.

Simulated Annealing

- Trata-se de um método de caminho aleatório (tipo Monte Carlo) mas em que é permitido, dentro de certas condições, saltar para valores mais altos da função de custo.
- O nome do método baseia-se na analogia com processo termodinâmico do mesmo nome, em que um metal ajusta a sua estrutura à medida que arrefece. Assim, o método utiliza uma variável designada por temperatura, cujo valor decresce ao longo do processo iterativo, e admite saltos para "cima" com uma probabilidade que que varia monotonamente com a temperatura. Um salto em que a função de custo aumente por ΔJ, aceita o salto se:

$$e^{-\Delta J/T} > r$$

, em que $r \in [0,1]$ é um número aleatório.

Simulated Annealing

$$e^{-\Delta J/T} > r$$

- T tem as dimensões de J. No início, T deve ser maior que os valores típicos de J (ex: 10 vezes maior do que os valores médios de J).
- Tal como no algoritmo de passeio aleatório "downslope", a dimensão do salto aleatório deverá também decrescer na iteração principal.
- O método permite uma exploração mais completa do espaço da função de custo, mas não tem garantia de convergência para o mínimo absoluto, mesmo que no meio do caminho a solução passe na sua bacia de atração.

```python
import matplotlib.pyplot as plt
import numpy as np
from math import pi as pi

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

################## Inicializar

#%% Setup

# Limites do domínio espacial
xmin=0.
xmax=100e3
ymin=0.
ymax=100e3

# Espaçamento entre pontos
dx=1e3
dy=1e3

# Pontos em x e y
x=np.arange(xmin,xmax+dx,dx)
y=np.arange(ymin,ymax+dy,dy)

# Nr de pontos
nx=len(x)
ny=len(y)

# Matrizes 2D de x, y
xx=np.zeros([nx,ny])
yy=np.zeros([nx,ny])

for ix in range(nx):
    for iy in range(ny):
        xx[ix,iy]=ix*dx
        yy[ix,iy]=iy*dy

xr=np.array([xmin,xmax])/1e3      # x range
yr=np.array([ymin,ymax])/1e3      # y range
```

```
#%% ############## Função custo (complicada)

def custo(X,Y):
      custo = np.sin(pi*X/10e3) * np.sin(pi*Y/10e3) * (np.cos(pi*(X-50e3)/100e3) * np.cos(pi*(Y-50e
      return custo

cc=np.zeros([nx,ny])        # inicializar a grelha de custos

# calcular a função de custo para todos os pontos
for ix in range(nx):
    for iy in range(ny):
        cc[ix,iy]=custo(xx[ix,iy],yy[ix,iy])

## Plot
plt.rcParams['figure.figsize'] = 7,6
plt.close()

plt.pcolor(xx/1e3,yy/1e3,cc)                # função custo
cbar = plt.colorbar()
cbar.set_label(u'Função Custo')

plt.xlabel('x (km)')
plt.ylabel('y (km)')
plt.xlim(xr)
plt.ylim(yr)

plt.savefig('p1-Custo.png')
```

```
#%%  ######################## Optimização

maxITER1=10000      # nr máximo de iterações, ciclo externo (1)
maxPERT2=1e3        # nr máximo de perturbações, ciclo interno (2)
maxHIT2=10000.      # nr máximo de soluções boas encontradas dentro do ciclo interior (2)

# vectores para guardar os resultados selecionados de cada iteração (ciclo externo, 1 + ciclo in
xSEL=np.nan*np.ones([maxITER1*maxHIT2])
ySEL=np.nan*np.ones([maxITER1*maxHIT2])

# salto
xstep0=2.*xmax          # tamanho do salto inicial (x)
ystep0=2.*ymax          # tamanho do salto inicial (y)
COOL=0.9                # razão de arrefecimento
kappa=0.05              # impacto da temperatura no salto

# critérios de paragem
Jmin=-50.
minstep=100.

#%% Iniciar a optimização

# Posição e custo inicial
X=xmin+(xmax-xmin)*np.random.rand();   # Posição em x inicial (aleatória)
Y=ymin+(ymax-ymin)*np.random.rand();   # Posição em y inicial (aleatória)
J=custo(X,Y);                         # Função custo para a posição inicial

# Primeira posição seleccionada
xSEL[0]=X
ySEL[0]=Y

# Valores iniciais dos saltos
xstep=xstep0        # tamanho do salto inicial (x)
ystep=ystep0        # tamanho do salto inicial (y)

# Temperatura inicial
T=10.
```

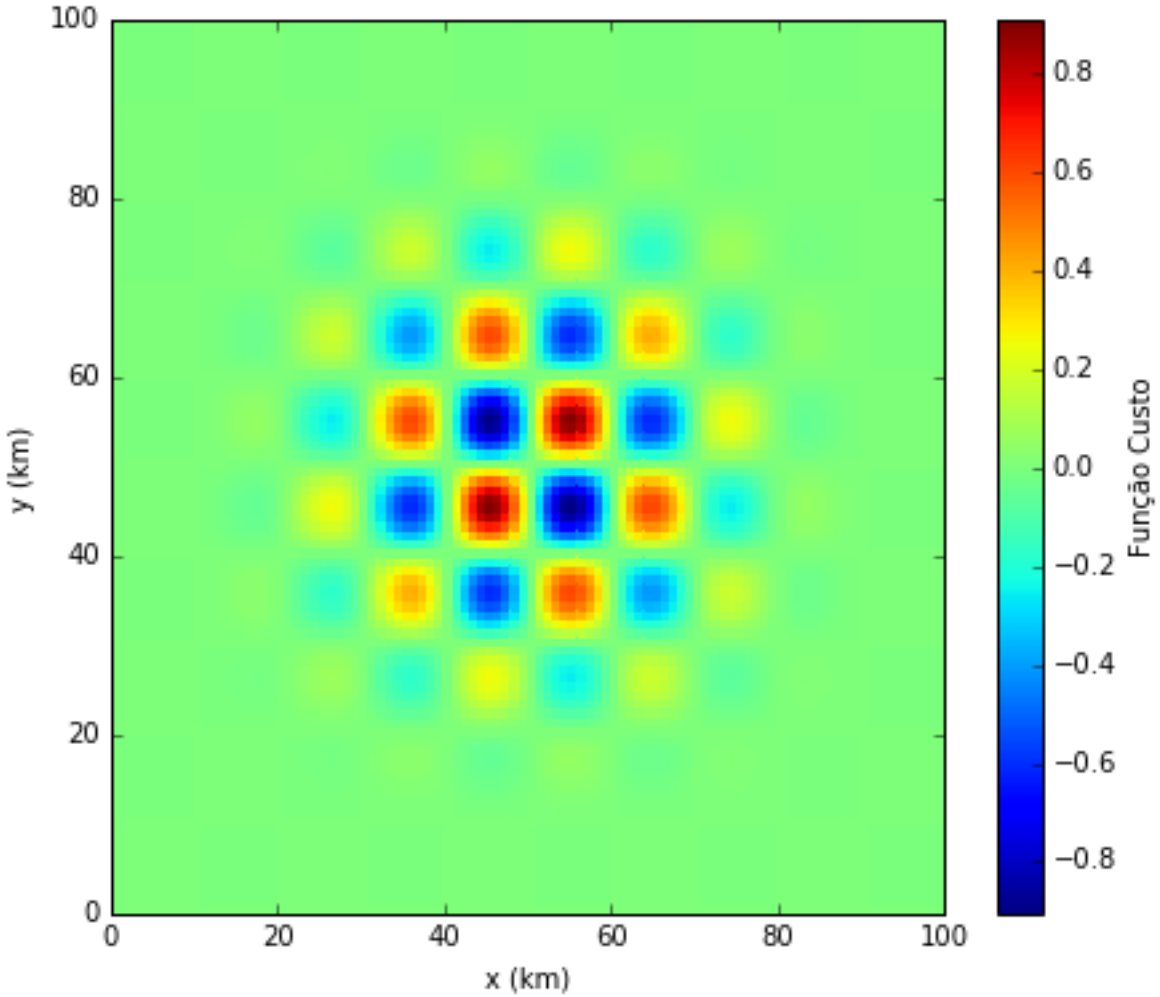
```
# Melhores valores iniciais
bestJ=J             # melhor valor da função custo
bestX=X             # melhor posição X
bestY=Y             # melhor posição Y
bestI=0             # melhor iteração
bestH=0             # melhor kHIT1

# Inicialização dos índices
iter1=0               # índice do ciclo exterior (1)
kHIT1=0               # nr total de soluções aceites (ciclo externo, 1 + ciclo interno, 2)
kpert=0               # índice da perturbação total (ciclo externo, 1 + ciclo interno, 2)

while (iter1<maxITER1 and J>Jmin and xstep>minstep):        # condições para estarmos dentro do c
    iter1+=1           # incrementar o índice do ciclo externo (1)
    iPERT2=0           # colocar a zero o índice do ciclo interno (2)
    nHIT2=0            # colocar a zero o nr de boas soluções encontradas dentro do ciclo interno
    upHIT2=0           # colocar a zero o nr de saltos para cima no ciclo interno (2)
    pertX=[]           # construir um vector para guardar as posições perturbadas, x
    pertY=[]           # construir um vector para guardar as posições perturbadas, y

    while iPERT2<maxPERT2 and nHIT2<maxHIT2:      # condições para estarmos dentro do ciclo interno
        kpert +=1

        # salto (nova posição) em x
        rr=np.random.rand()-0.5     # nr aleatório entre -.5 e .5
        XI=X+xstep*rr;              # novo x varia em torno de X por um valor aleatório, cujo valo
        while XI<xmin or XI>xmax:  # aceitar apenas valores dentro do domínio, se fora do domínio
            rr=np.random.rand()-0.5
            XI=X+xstep*rr

        # salto (nova posição) em y
        rr=np.random.rand()-0.5        # nr aleatório entre -.5 e .5
        YI=Y+ystep*rr;                 # novo y varia em torno de Y por um valor aleatório, cujo va
        while YI<ymin or YI>ymax:      # aceitar apenas valores dentro do domínio, se fora do domí
            rr=np.random.rand()-0.5
            YI=Y+ystep*rr;
```

```
pertX.append(XI)        # guardar as posições perturbadas, XI
pertY.append(YI)        # guardar as posições perturbadas, YI

JI=custo(XI,YI);    # calcular a função custo para a nova posição perturbada (aleatória)
dJ=JI-J;            # diferença entre a função custo de referiencia (J) e a calculada para

if dJ<0:            # salto pata baixo: se a nova função custo (JI) tiver um valor mais ba
    J=JI                    # actualizar a função custo de referência
    X=XI                    # actualizar a posição x de referência
    Y=YI                    # actualizar a posição y de referência
    nHIT2=nHIT2+1         # actualizar o nr de boas soluções no ciclo interior (2))
    kHIT1=kHIT1+1          # actualizar o nr de soluções aceites total (1)
    xSEL[kHIT1]=X             # guardar a posição x seleccionada
    ySEL[kHIT1]=Y             # guardar a posição y seleccionada

else:               # salto para cima, aceitar com uma certa probabilidade (nem sempre é
    rr=np.random.rand()
    if np.exp(-dJ/T) > rr:      # com uma certa probabilidade, aceitar uma solução pior
        J=JI                    # actualizar a função custo de referência
        X=XI                    # actualizar a posição x de referência
        Y=YI                    # actualizar a posição y de referência
        upHIT2=upHIT2+1        # actualizar o nr de boas soluções no ciclo interior (2))
        kHIT1=kHIT1+1          # actualizar o nr de soluções aceites total (1)
        xSEL[kHIT1]=X           # guardar a posição x seleccionada
        ySEL[kHIT1]=Y           # guardar a posição y seleccionada

if J<bestJ:         # guadar a melhor solução
    bestJ=J         # actualizar o melhor valor da função custo
    bestX=X         # actualizar a melhor posição X
    bestY=Y         # actualizar a melhor posição Y
    bestI=iter1     # actualizar a melhor iteração
    bestH=kHIT1     # actualizar o melhor kHIT1

iPERT2+=1;      # incrementar o índice do ciclo interior
```

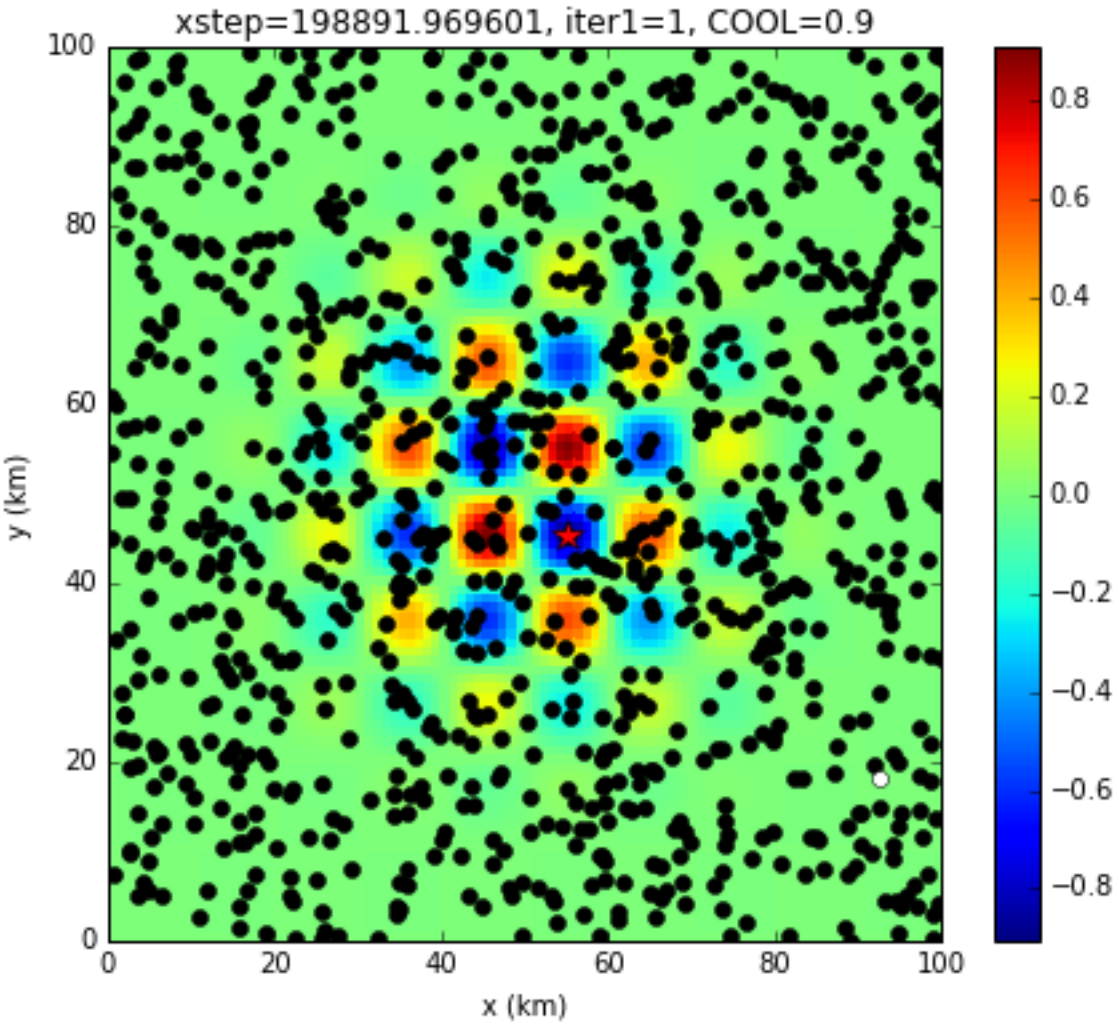
```
print('iter1='+str(iter1) + ', kHIT1='+str(kHIT1) + ', xstep='+str(xstep) + \
       ', iPERT2='+str(iPERT2) + ', nHIT2='+str(nHIT2)+ ', J='+str(J)+', DJ='+str(dJ)+ \
       ' X='+str(XI)+', Y='+str(YI) + ' \n')

# definir o salto e a temperatura da próxima iteração, ciclo externo (1)
T=T*COOL
xstep=np.max([minstep,xstep*np.exp(-kappa/T)]);    # arrefecimento
ystep=np.max([minstep,ystep*np.exp(-kappa/T)]);    # arrefecimento


# Plot
if iter1 in [1, 2, 5, 10, 20, 30, 40, 45, 100, 1000]:
    plt.rcParams['figure.figsize'] = 7,6
    plt.close()

    plt.pcolor(xx/1e3,yy/1e3,cc)
    cbar = plt.colorbar()
    cbar.set_label(u'Função Custo')

    plt.plot(np.array(pertX)/1e3, np.array(pertY)/1e3, 'o', markerfacecolor='black')
    plt.plot(xSEL[0]/1e3, ySEL[0]/1e3, 'o', markerfacecolor='white');
    plt.plot(bestX/1e3, bestY/1e3, '*', markersize=12, markerfacecolor='red');

    plt.xlabel('x (km)')
    plt.ylabel('y (km)')
    plt.xlim(xr)
    plt.ylim(yr)

    plt.title('xstep='+str(xstep) +', iter1='+str(iter1)+', COOL='+str(COOL))
    plt.savefig('p3-iter1_'+str(iter1)+'.png')
```

```
#%% Plot final

plt.rcParams['figure.figsize'] = 7,6
plt.close()

plt.pcolor(xx/1e3,yy/1e3,cc)
plt.plot(xSEL[:100]/1e3, ySEL[:100]/1e3, 'white');
cbar = plt.colorbar()
cbar.set_label(u'Função Custo')

plt.plot(xSEL[0]/1e3, ySEL[0]/1e3, 'o', markerfacecolor='white');
plt.plot(bestX/1e3, bestY/1e3, '*', markersize=12, markerfacecolor='red');

plt.xlabel('x (km)')
plt.ylabel('y (km)')
plt.xlim(xr)
plt.ylim(yr)

plt.title('xstep0='+str(xstep0) +', niter='+str(iter1)+', COOL='+str(COOL)+', kappa='+str(kappa))
plt.savefig('p2-Final.png')
```

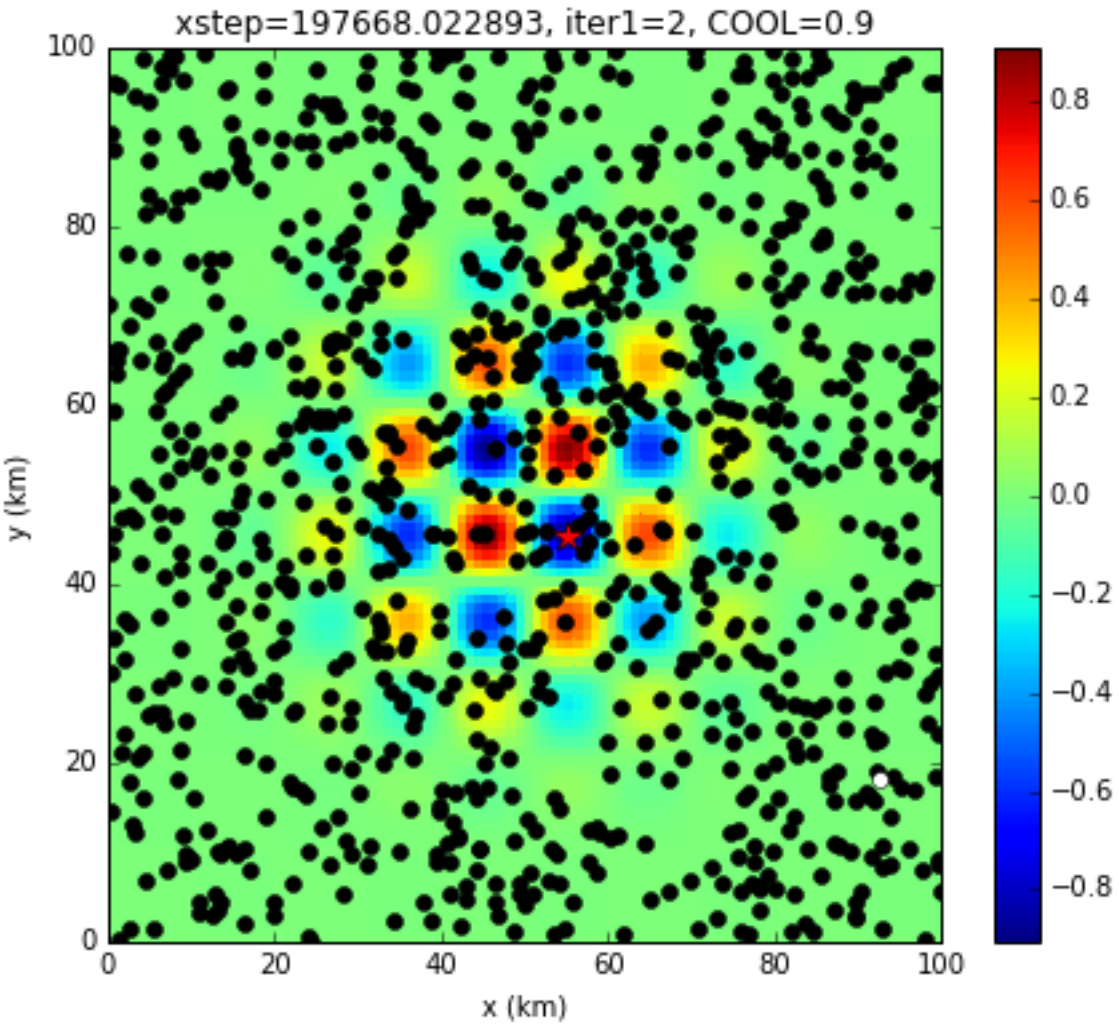
Exemplo:


100
80
60
40
20
0
y (km)
0
20
40
60
80
100
x (km)
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
Função Custo

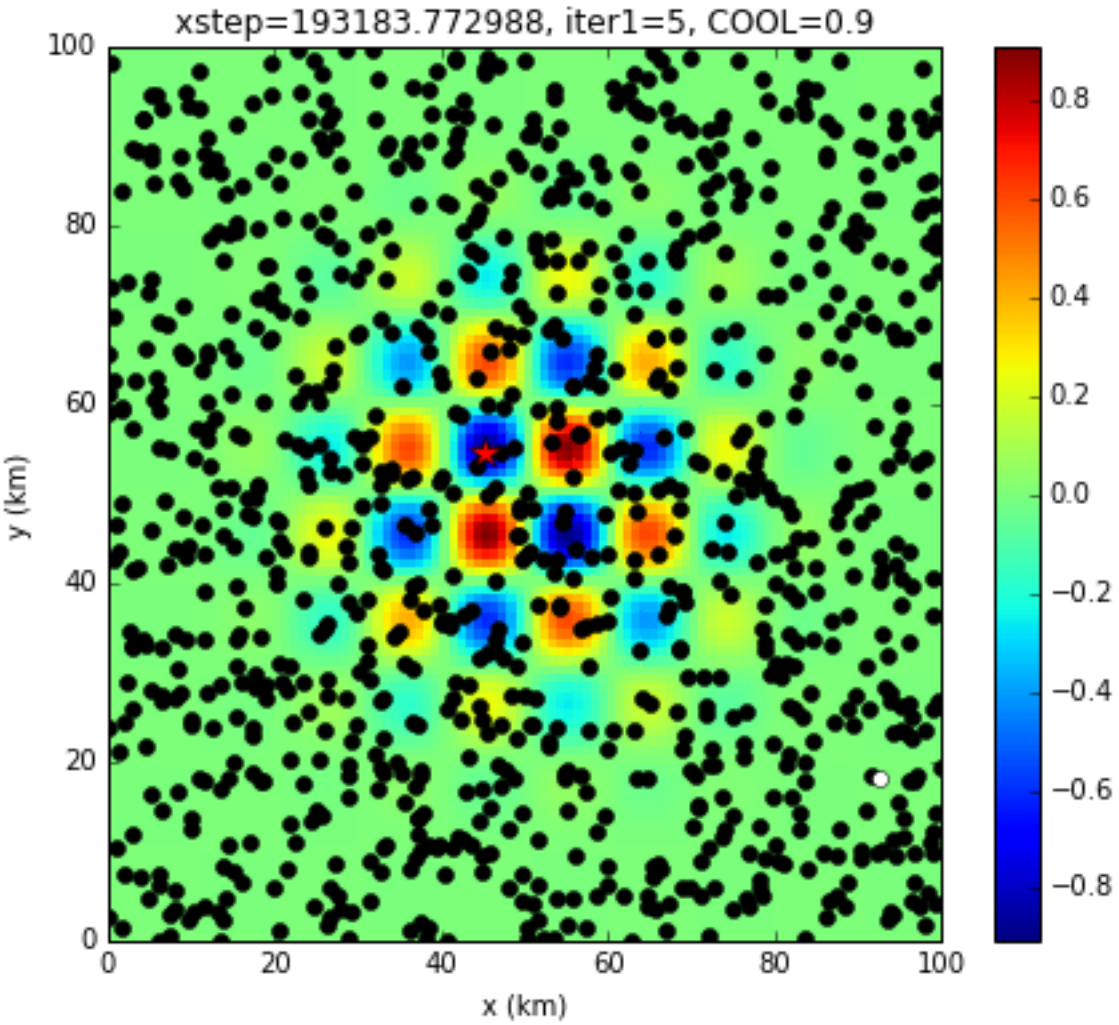
Exemplo:


xstep=198891.969601, iter1=1, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

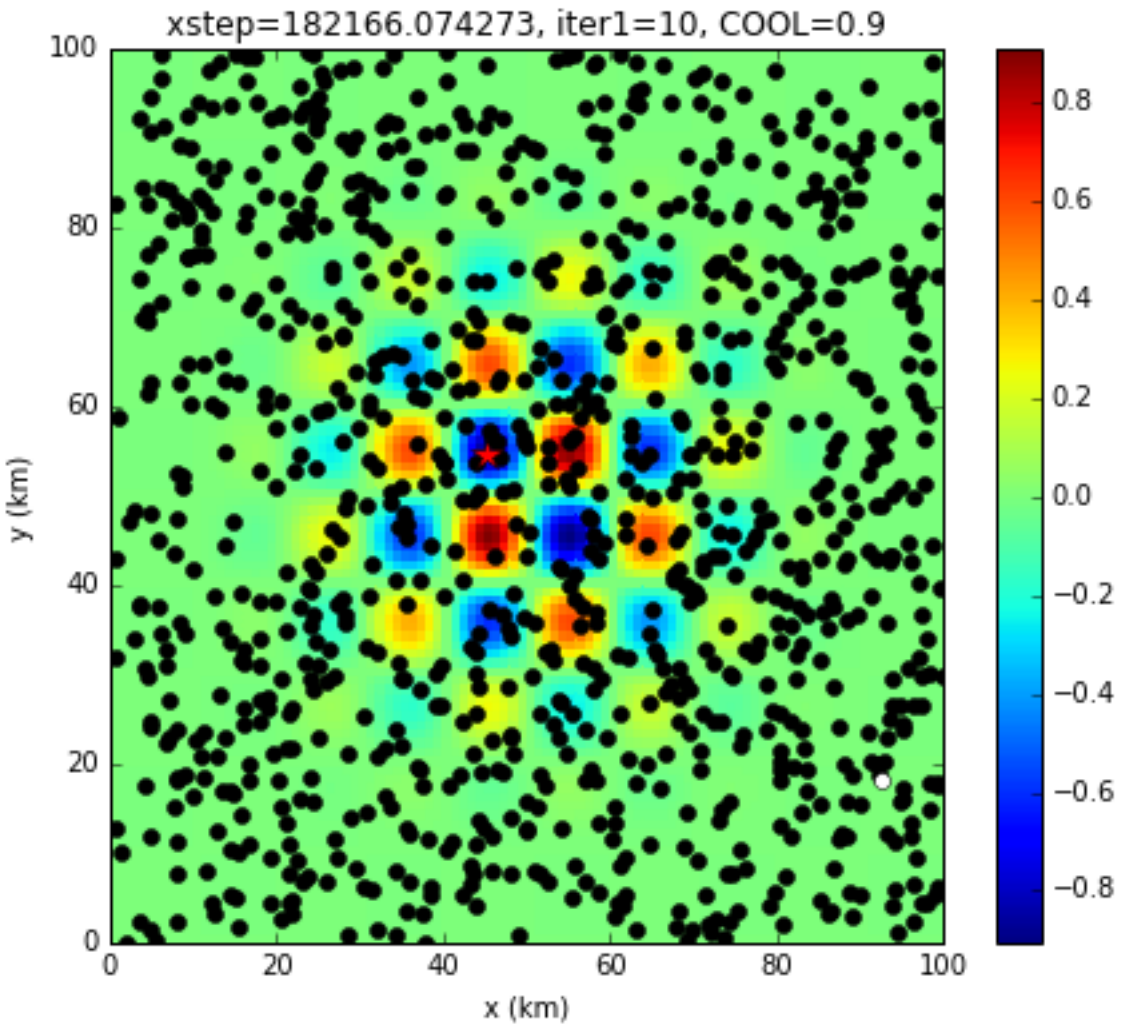
Exemplo:


xstep=197668.022893, iter1=2, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

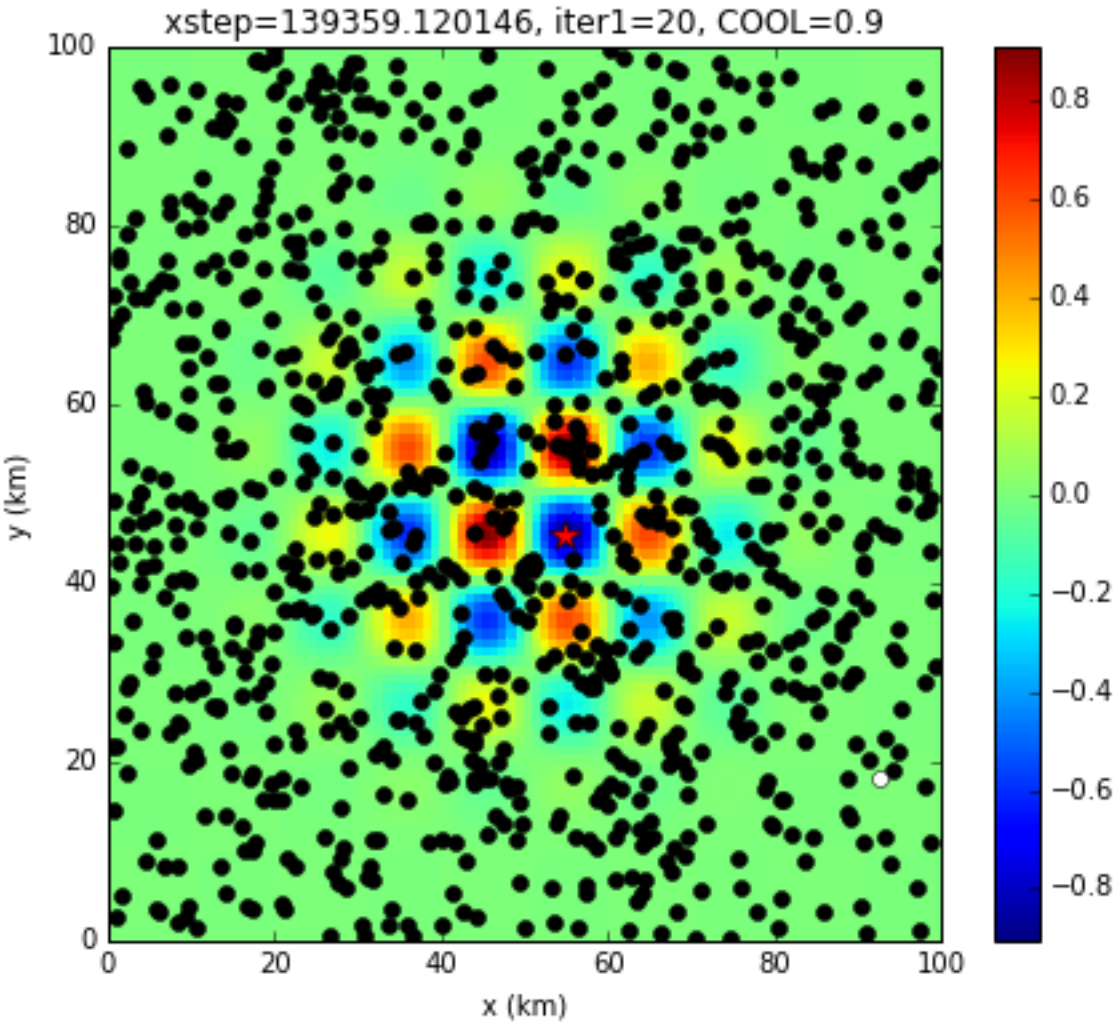
Exemplo:


xstep=193183.772988, iter1=5, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

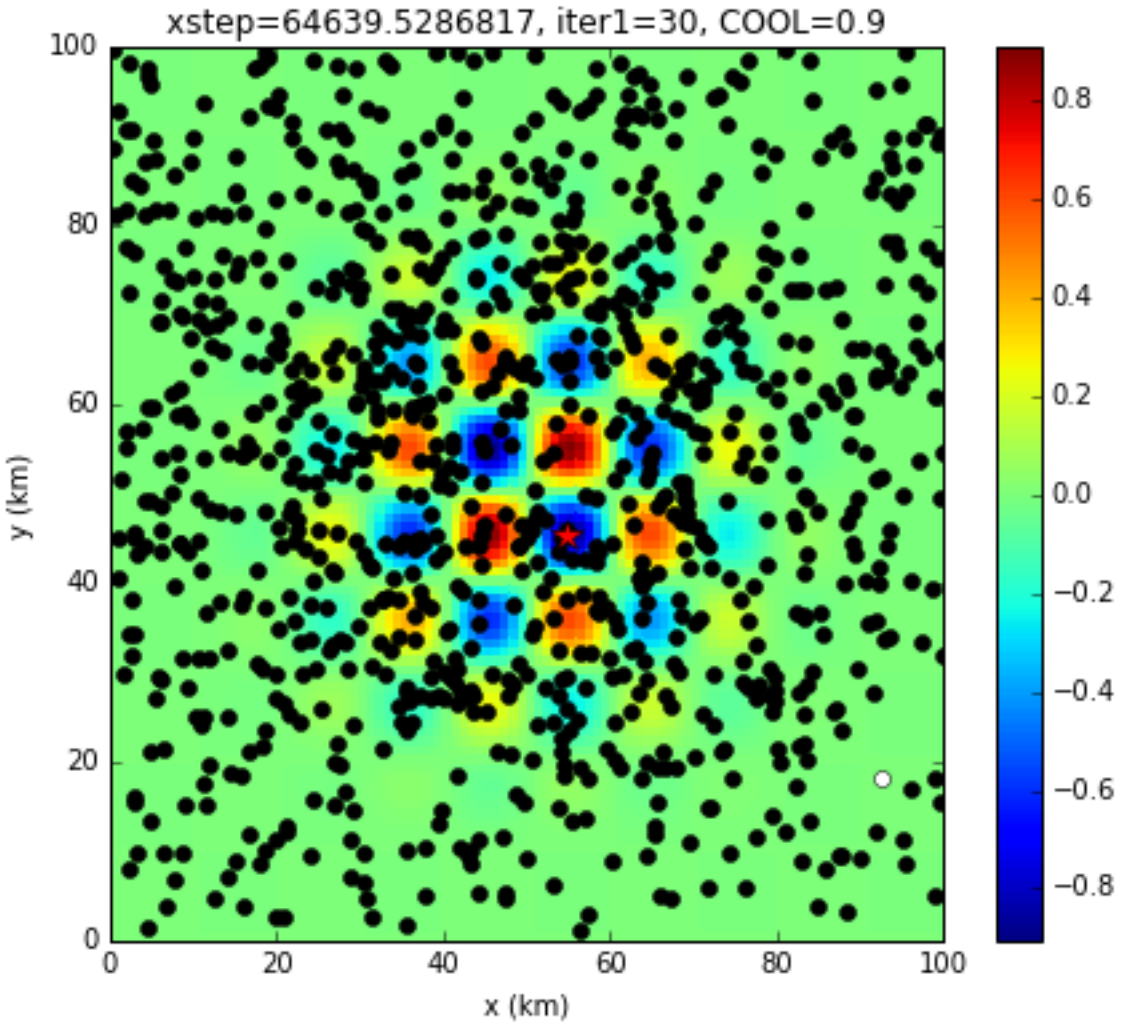
Exemplo:


xstep=182166.074273, iter1=10, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

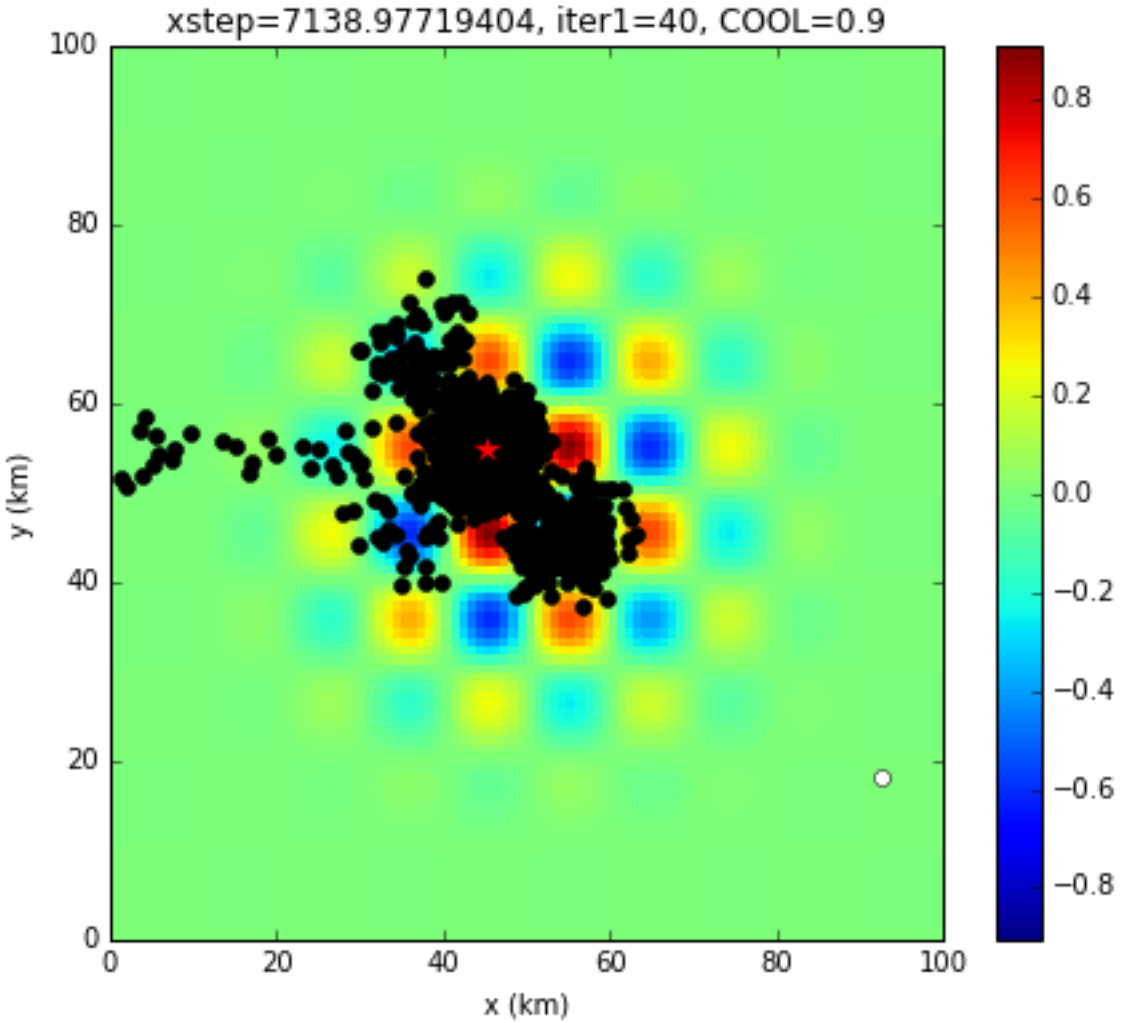
Exemplo:


xstep=139359.120146, iter1=20, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

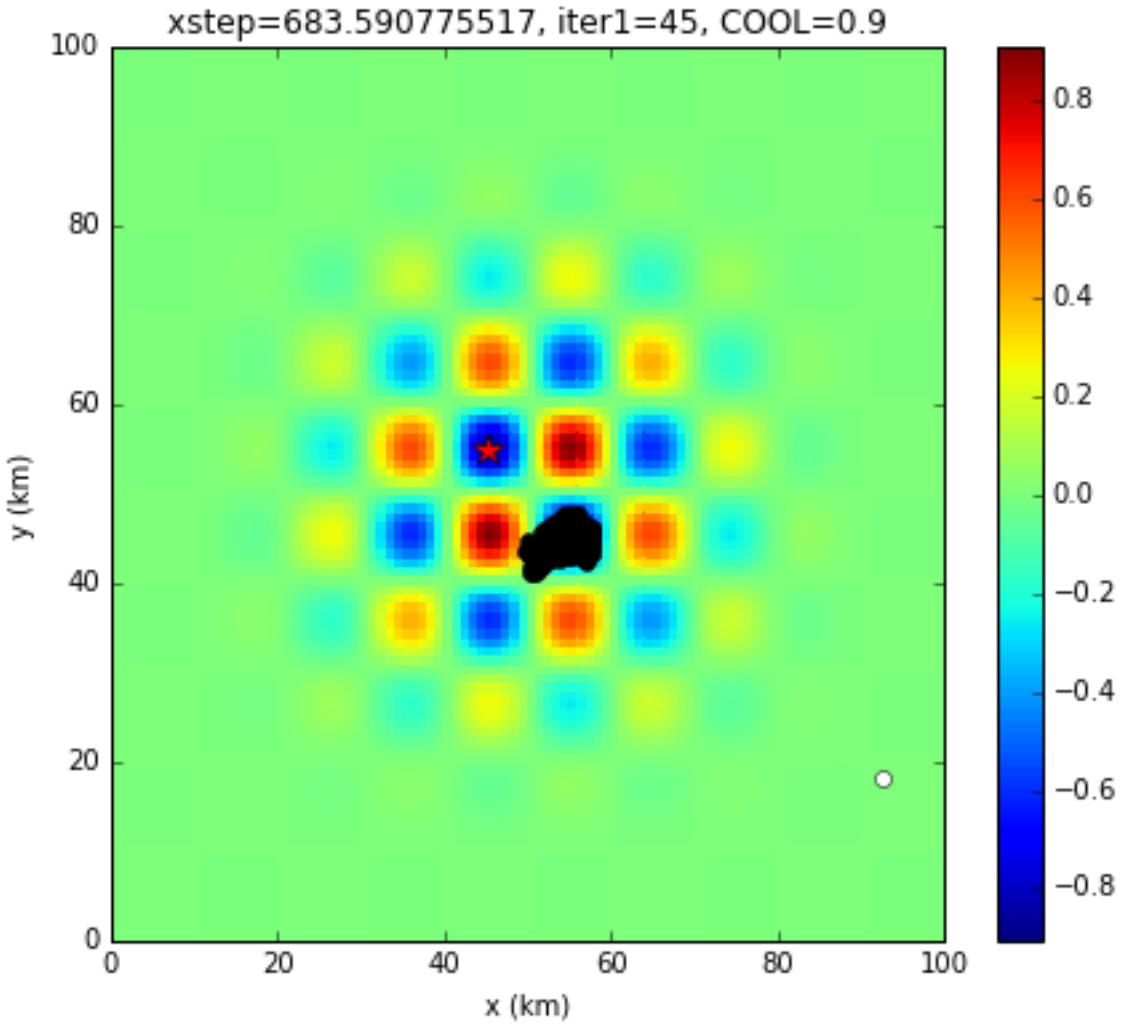
Exemplo:


xstep=64639.5286817, iter1=30, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

Exemplo:


xstep=7138.97719404, iter1=40, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

Exemplo:


xstep=683.590775517, iter1=45, COOL=0.9
y (km)
x (km)
0
20
40
60
80
100
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8

Comentários:

- A função de custo depende do problema!
- O resto do código é genérico, para um problema com 2 variáveis a calcular (X,Y), mas tem vários parâmetros ajustáveis
- T inicial
- COOL razão de arrefecimento
- kappa: efeito da temperatura no salto
- xstep: salto inicial
- maxITER1: numero máximo de iterações (ciclo externo)
- maxPERT2: numero máximo de perturbações (ciclo interno)
- Minstep: dimensão minima do salto (depende da resolução pretendida)
- Jmin: critério de paragem (J<Jmin)
- Para cada problema é preciso pensar e experimentar

Comentários:

- O método de annealing é interessante mas não é sempre mais interessante que o passeio downslope. Por outro lado, também não é muito mais complicado: o código é igual se se eliminar os saltos para “cima”.
- Se se conhecer a dimensão da bacia de atração do mínimo absoluto (definida mais estritamente como a região onde a função de custo é inferior a todas as outras regiões) pode estabelecer-se o número de perturbações (maxPERT tentativas por iteração) suficiente para que o passeio downslope tenha boa probabilidade de funcionar.
- Se se conhecer um bom first-guess pode fazer-se o passeio aleatório com um salto inicial limitado à dimensão da bacia de atração garantindo que a solução não sai dessa região.