Ciências ULisboa

Modelação Numérica
Aula 20

Exercícios de aplicação

Estimativa (astronómica) da interface núcleo-manto

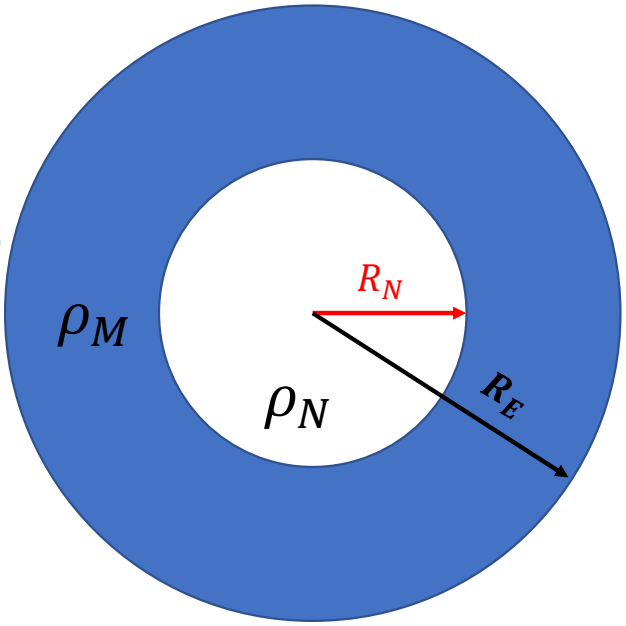
Vamos considerar um modelo simplista da Terra com duas camadas: um manto e um núcleo. Vamos admitir que são conhecidos:

O raio da Terra, $R_E \approx 6.371 \times 10^6 m$

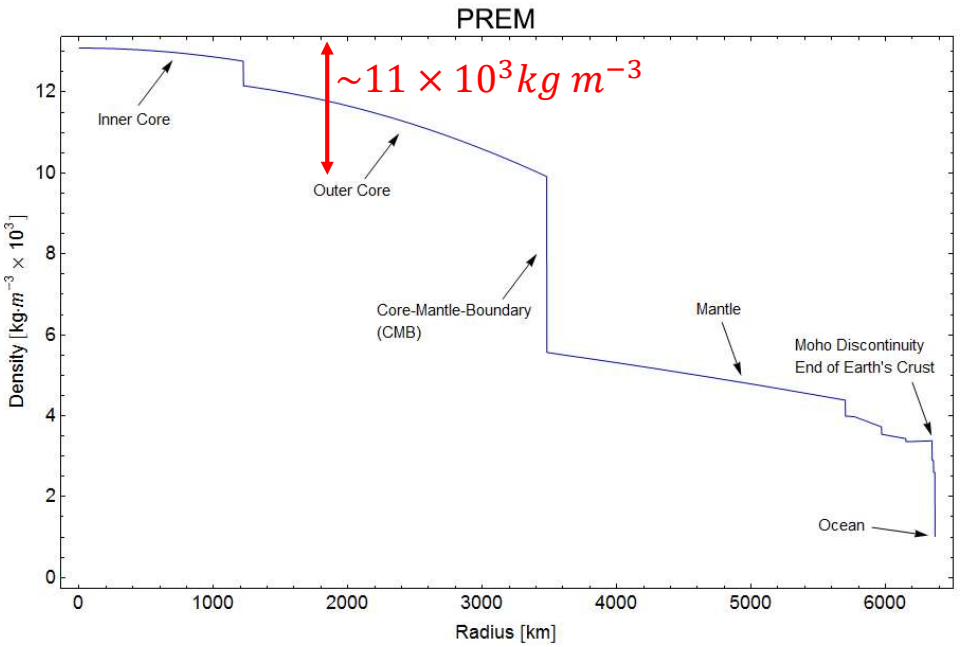
A massa da Terra, $M_E \approx 5.97 \times 10^{24} kg$

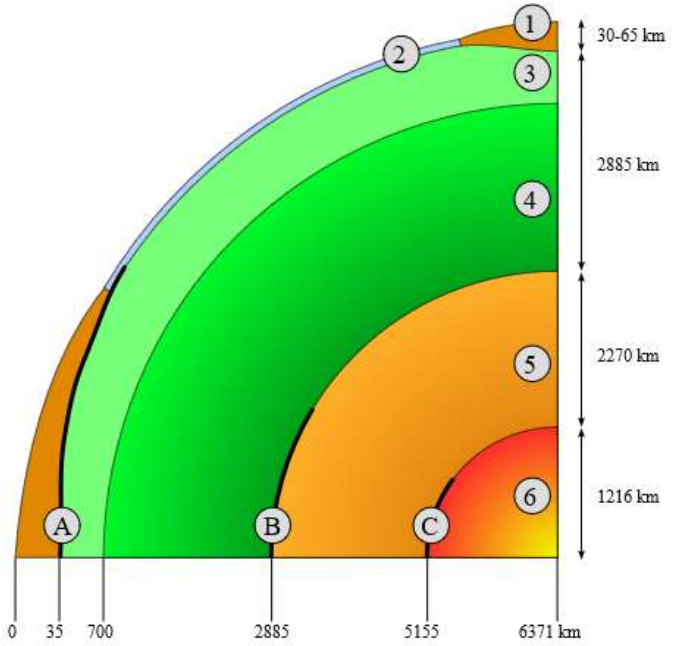
O momento de inércia da Terra, $I_E \approx 8.02 \times 10^{37}\ kg\ m^2$

Pretendemos calcular o raio do núcleo e as densidades do núcleo e do manto.



Melhor estimativa geofísica





By AllenMcC. - Own work, Paper: http://www.gps.caltech.edu/uploads/File/People/dla/DLApepi81.pdf, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=12476245

BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=809709

Análise preliminar: massa

$$M_E = \frac{4}{3}\pi R_E^3 \bar{\rho} = M_N + M_M$$

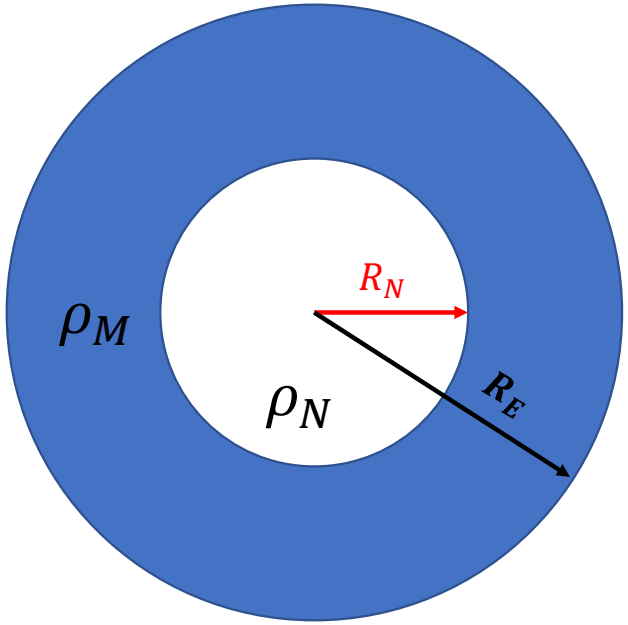
$$M_N = \frac{4}{3}\pi R_N^3 \rho_N$$

$$M_M = \frac{4}{3}\pi\left(R_E^3 - R_M^3\right)\rho_M$$

Logo

$$\rho_M = \frac{M_E - \frac{4}{3}\pi R_N^3 \rho_N}{\frac{4}{3}\pi\left(R_E^3 - R_M^3\right)} = \rho_N(R_N, \rho_N)$$

i.e. só há duas variáveis a calcular.

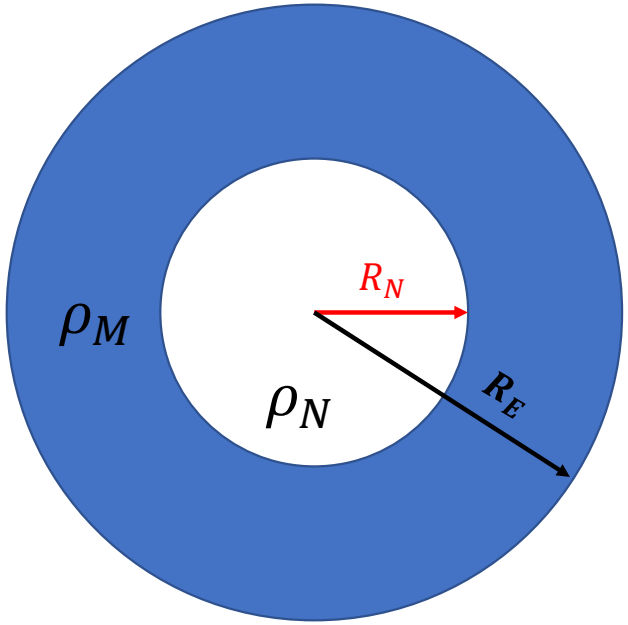


Análise preliminar: momento de inércia

$$I_E = I_N + I_M$$

$$I_E = \frac{2}{5} M_N R_N^2 + \frac{2}{5} M_M \frac{R_E^5 - R_N^5}{R_E^3 - R_N^3}$$

$$I_E = \frac{2}{5}\left[\frac{4}{3}\pi\left(\rho_N R_N^2 + \rho_M \frac{R_E^5 - R_N^5}{R_E^3 - R_N^3}\right)\right] = I_E(R_N, \rho_N)$$



Para cada valor de (R_N, ρ_N), podemos calcular uma estimativa do momento de inércia.

Solução numérica

$R_N \approx 4.47 \times 10^6 m$

$\rho_N \approx 9.03 \times 10^3 kg\, m^{-3}$

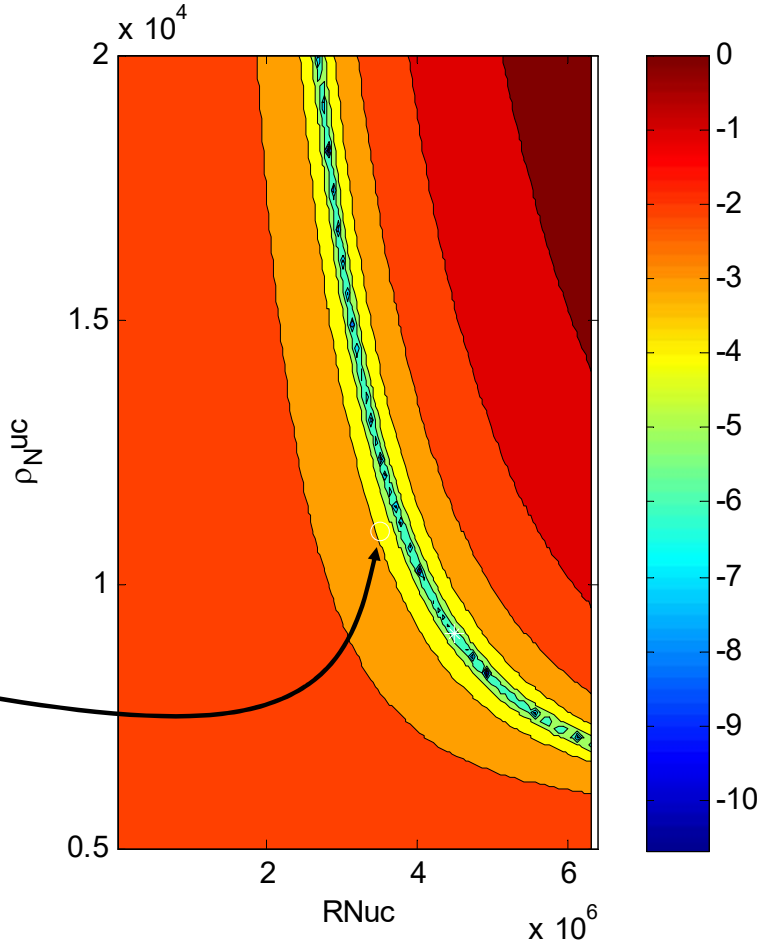
A solução está mal condicionada e o valor proposto está errado.

De facto

$R_N \approx 3486 \times 10^3 m$

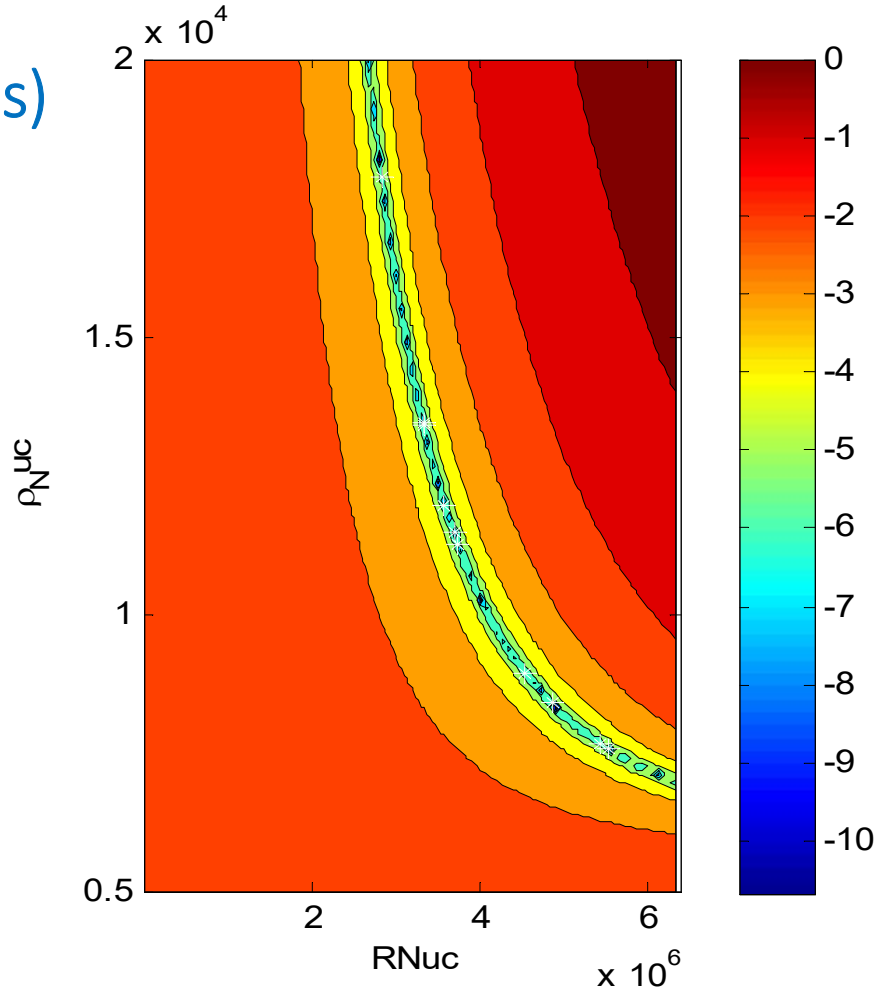
$\rho_N \approx 11 \times 10^3 kg m^{-3}$

A solução real não é satisfeita pelo modelo de 2 camadas homogéneas.



10 soluções numéricas (≠ first guess)

Os valores ao longo do vale de mínimo são igualmente prováveis…



Código

```
function custo=cost(V)
global REarth MEarth MIEarth
RNuc=V(1);rhoNuc=V(2);
MNuc=4/3*pi*RNuc^3*rhoNuc;
rhoMan=(MEarth-MNuc)/(4/3*pi*(REarth^3-RNuc^3));
MMan=4/3*pi*(REarth^3-RNuc^3)*rhoMan;
MI=2/5*MMan*((REarth^5-RNuc^5)/(REarth^3-RNuc^3))+2/5*MNuc*RNuc^2;
custo=abs((MI-MIEarth)/MIEarth);
end
```

$$-\log_{10} K_{w_k} = \frac{a}{T_k} + b \log_{10} T_k + cT_k + d$$

Mudando a notação:

$$-\log_{10} K_{w_k} = \frac{a_0}{T_k} + a_1 \log_{10} T_k + a_2 T_k + a_3 \quad [k = 0, \ldots N-1]$$

Ou de forma mais genérica:

$$y_k = \sum_{j=0}^{N_{j-1}} a_j f_j(x_k)\,, \qquad [k = 0, \ldots, N_k - 1]$$

Na presença de erro:

$$\overline{e^2} = \frac{1}{N} \sum_{k=0}^{N_{k-1}} \left(y_k - \sum_{j=0}^{N_{j-1}} a_j f_j(x_k) \right)^2$$

Condição de menor erro médio quadrático:

$$\frac{\partial \overline{e^2}}{\partial a_j} = 0 \quad [j = 0, \ldots, N_j - 1]$$

$$y_k = \sum_{j=0}^{N_{j-1}} a_j f_j(x_k)\,, \qquad [k = 0, \ldots, N_k - 1]$$

Note-se que mesmo que as funções f_j não sejam lineares, o problema de otimização, i.e., o cálculo do melhor conjunto de parâmetros a_j é um problema linear.

O caso em que as funções f_j sejam os termos de um polinómio de ordem $N_j - 1$, o problema de otimização é resolvido diretamente com a função **polyfit**.

Em geral, a expressão do erro médio quadrático pode ser desenvolvida, e a condição de mínimo aplicada, dando origem a um sistema de N_j equações lineares.

Aplicação: $-\log_{10} K_{w_k} = \frac{a_0}{T_k} + a_1 \log_{10} T_k + a_2 T_k + a_3 \quad [k = 0, \ldots N-1]$

```
def mkdata(nE=4,noise=0):
    a=np.array([0.5,2,0.01,5])
    x=273+20*np.random.random((nE))
    y=a[0]/x+a[1]*np.log(x)/np.log(10)+a[2]*x+a[3]
    ny=(np.random.random((nE))-0.5)*noise*np.mean(y)
    y=y+ny
    return x,y,a
T,mlogK,a=mkdata()
print(T)
print(mlogK)
```

[274.07875564 279.79048986 281.64724044 280.52810574]

[12.6183626 12.69335785 12.71765868 12.70301617]

Aplicação: $-\log_{10} K_{w_k} = \frac{a_0}{T_k} + a_1 \log_{10} T_k + a_2 T_k + a_3 \quad [k = 0, \ldots N-1]$

```
M=np.zeros((4,4))
b=np.zeros((4))
for i in range(4):
    M[i,0]=1/T[i]
    M[i,1]=np.log(T[i])/np.log(10)
    M[i,2]=T[i]
    M[i,3]=1
    b[i]=mlogK[i]
a1=np.linalg.solve(M,b)
print(a1)
```

[0.49999993 2. 0.01 5.] # `a=np.array([0.5,2,0.01,5])`

Aplicação: $-\log_{10} K_{w_k} = \frac{a_0}{T_k} + a_1 \log_{10} T_k + a_2 T_k + a_3 \quad [k = 0, \ldots N-1]$

```
a2=np.matmul(np.linalg.pinv(M),b)
print(a2)
```

[0.50000381 1.99999988 0.01 5.] # `a=np.array([0.5,2,0.01,5])`

Aplicação: $-\log_{10} K_{w_k} = \frac{a_0}{T_k} + a_1 \log_{10} T_k + a_2 T_k + a_3 \quad [k = 0, \ldots N-1]$

```
nE=100
T,mlogK,a=mkdata(nE=nE)
M=np.zeros((nE,4))
b=np.copy(mlogK)
for i in range(nE):
    M[i,0]=1/T[i]
    M[i,1]=np.log(T[i])/np.log(10)
    M[i,2]=T[i]
    M[i,3]=1

# inversa generalizada no caso Nj!=Nk
a3=np.matmul(np.linalg.pinv(M),b)
print(a3)
```

[0.50000001 2. 0.01 5.]

Aplicação: $-\log_{10} K_{w_k} = \frac{a_0}{T_k} + a_1 \log_{10} T_k + a_2 T_k + a_3 \quad [k = 0, \ldots N-1]$

```
#inversa generalizada no caso sobredeterminado
nE=100
T,mlogK,a=mkdata(nE=nE,noise=1e-9)
M=np.zeros((nE,4))
b=np.copy(mlogK)
for i in range(nE):
    M[i,0]=1/T[i]
    M[i,1]=np.log(T[i])/np.log(10)
    M[i,2]=T[i]
    M[i,3]=1

# inversa generalizada no caso Nj=Nk
a4=np.matmul(np.linalg.pinv(M),b)
print(a4)
```

[0.49516158 1.99992255 0.01000006 5.00019046]