B

a) Step 1 with EF

$$\frac{z(t+\Delta t) - z(t)}{\Delta t} = 0 \quad , \text{since } \dot{z}(0) = 0$$

$$\frac{z_1 - z_0}{\Delta t} = 0$$

$$z_1 = z_0$$

General formula

$$\frac{z(t+\Delta t) - 2z(t) + z(t-\Delta t)}{\Delta t^2} = - \frac{\gamma M}{(z(t) + R_E)^2}$$

$$\boxed{z_{t+1} = \frac{-\gamma M \Delta t^2}{(z(t)+R_E)^2} + 2z(t) - z(t-\Delta t)}$$

b) $$\frac{d^2 z}{dt^2} = \frac{-\gamma M}{(z+R_E)^2}$$

$$u_1 = z \qquad \frac{du_1}{dt} = u_2$$

$$u_2 = \frac{dz}{dt} \qquad \frac{du_2}{dt} = \frac{d^2 z}{dt^2} = \frac{-\gamma M}{(z+R_E)^2}$$

$$\left(\begin{pmatrix} u_{1,t+\Delta t} \\ u_{2,t+\Delta t} \end{pmatrix} = \begin{pmatrix} u_{1,t} \\ u_{2,t} \end{pmatrix} + \Delta t \begin{pmatrix} u_{2,t} \\ \dfrac{-\gamma M}{(u_{1,t}+R_E)^2} \end{pmatrix} \right)$$

$$\frac{du_1}{dt} = u_2 \qquad \frac{u_1(t+\Delta t) - u_1(t)}{\Delta t} = u_2(t)$$

$$\frac{du_2}{dt} = \frac{-\gamma M}{(u_1+R_E)^2} \qquad \frac{u_2(t+\Delta t) - u_2(t)}{\Delta t} = \frac{-\gamma M}{(u_1+R_E)^2}$$

$$\boxed{u_1(t+\Delta t) = u_1(t) + u_2(t)\,\Delta t}$$

$$\boxed{u_2(t+\Delta t) = u_2(t) - \frac{\gamma M}{(u_1(t)+R_E)^2}\,\Delta t}$$

14. $$\frac{d^2 T(x)}{dx^2} = h(x)$$

$h(x) = 0.1x \wedge 0 \leq x \leq 10$

$T(0) = 3 \wedge \left.\frac{dT}{dx}\right|_{x=10} = -2$

a) $\underline{C} = h_1 - \frac{T_0}{\Delta x^2}, h_2, \ldots$

Left side with dirichlet

$$\frac{T_{j+1} - 2T_j + T_{j-1}}{\Delta x^2} = h_j$$

$\underline{j=1}$ $\quad T_2 - 2T_1 = \left(h_1 - \frac{T_0}{\Delta x^2}\right)\Delta x^2$

$\underline{j=2}$

$\vdots \quad \ldots = h_j$

$\underline{j=N-1}$

Right side with Neumann

$\underline{j=N}$ $\quad \frac{T_{N+1} - 2T_N + T_{N-1}}{\Delta x^2} = h_N \rightarrow \frac{T_{N+1} - T_N - T_N + T_{N-1}}{\Delta x^2} = h_N$

$$\frac{dT}{dx\,\Delta x} + \frac{-T_N + T_{N-1}}{\Delta x^2} = h_N$$

$$\frac{-T_N + T_{N-1}}{\Delta x^2} = h_N - \frac{dT}{dx\,\Delta x}$$

$$\longrightarrow \underline{C} = \left(h_1 - \frac{T_0}{\Delta x^2}, h_2, \dots h_{N-1}, h_N - \frac{dT}{dx\,\Delta x} \right)$$

first → dirichlet

last row Neumann

$$\underline{\underline{M}} = \begin{pmatrix} \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \dots & & & 0 \\ \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \dots & & 0 \\ 0 & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} & 0 & \dots & 0 \\ & & \ddots & \ddots & \vdots & & \vdots \\ 0 & \dots & & & \frac{1}{\Delta x^2} & \frac{-2}{\Delta x^2} & \frac{1}{\Delta x^2} \\ 0 & \dots & & & & \frac{1}{\Delta x^2} & -\frac{1}{\Delta x^2} \end{pmatrix}$$

14.

c) Ansatz $T(x) = Ax^3 + Bx^2 + Cx + D$

$T'(x) = 3Ax^2 + 2Bx + C$

$T''(x) = 6Ax + 2B$

Sheet: $T''(x) = 0{,}1x$ Coefficients

$\rightarrow 0{,}1x = 6Ax + 2B$

no x in front of B, therefore $B \stackrel{!}{=} 0$

$0{,}1x = 6Ax \rightarrow A = \frac{0{,}1}{6}$

$T'(10) = -2$

$\rightarrow -2 = \cancel{3} \cdot \frac{\cancel{0{,}1}}{\cancel{6}_2} \cdot 10^{\cancel{2}} + C$

$C = -7$

$T(0) = 3$

$\rightarrow 3 = D$

$$\boxed{T(x) = \frac{0{,}1}{6}x^3 - 7x + 3}$$

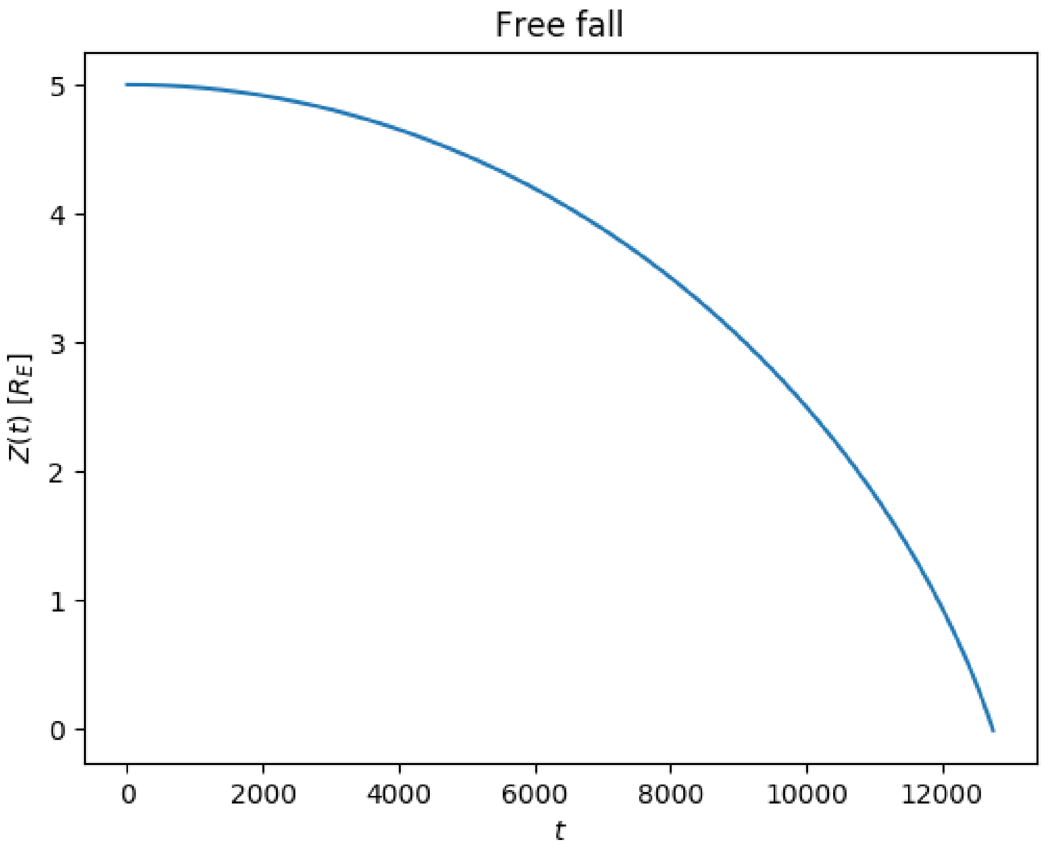
sheet9

October 31, 2017

1 13. Free fall in Earth's inhomogeneous gravitational field

1.1 a)

In [1]:
```
using PyPlot
g= 6.67408e-11 #[m^3/(kg*s^2)]
re=6371000 #mean radius [m]
m=5.97237e24 #[kg]
dt=10 #[s]

z=Float64[]
dz=Float64[]
t=Float64[]

push!(z,5*re)
push!(dz,0)
push!(t,0)

#first value
push!(z,z[1])
push!(t,t[1]+dt)
#do until z>surface (z=0)
i=2
while (z[i]>0)
    push!(z,-g*m*dt^2/(z[i]+re)^2+2*z[i]-z[i-1])
    push!(t,t[i]+dt)
    i+=1
end

title("Free fall")
plot(t,z/re)
ylabel(L"$Z(t)$ $[R_E]$")
xlabel(L"$t$")


sleep(1)
println("Time to reach the surface: ",t[length(t)],"s")
```

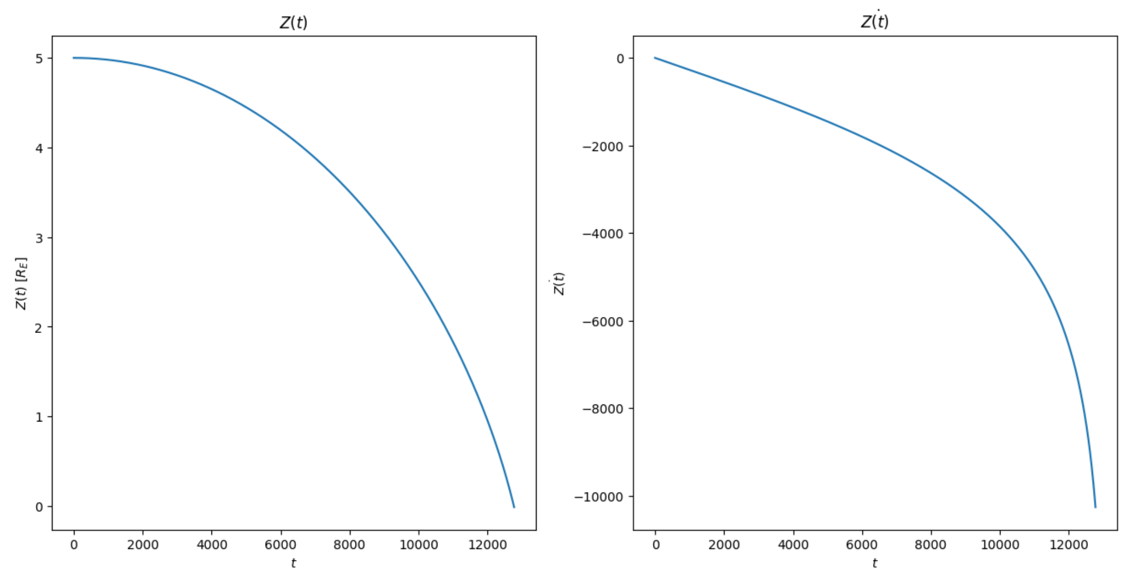



```
Time to reach the surface: 12760.0s
```

1.2 b)

```
In [2]: #initialization
        z=Float64[]   #u1
        dz=Float64[] #u2
        t=Float64[]

        push!(z,5*re)
        push!(dz,0)
        push!(t,0)

        i=1
        while (z[i]>0)
            push!(z,z[i]+dz[i]*dt)
            push!(dz,dz[i]-g*m*dt/(z[i]+re)^2)
            push!(t,t[i]+dt)
            i+=1
```

```
end

#ploting commands
figure(1,figsize=(15,7))
#2figs in line, linenumber=1, rownumber=2,number of figure=1
subplot(121)
title(L"$Z(t)$")
plot(t,z/re)
ylabel(L"$Z(t)$ $[R_E]$")
xlabel(L"$t$")
legend()
println("Time to reach the surface: ",t[length(t)],"s")
subplot(122)
title(L"$\dot{Z(t)}$")
plot(t,dz)
ylabel(L"$\dot{Z(t)}$")
xlabel(L"$t$")
legend()
```




```
Time to reach the surface: 12770.0s
```

```
/usr/local/lib/python2.7/dist-packages/matplotlib/axes/_axes.py:545: UserWarning: No labelled
  warnings.warn("No labelled objects found. "
```

There's a 10s difference between both methods for $\Delta t = 1s$. Since the method used in b) is of first order accuracy, compared to 2nd order accuracy in a), I would stick with the value from a)

2 14. Steady-state diffusion equation

2.1 b)

In [3]:
```
#algorithm from sheet 7
function thomasalgo(d,a,b,y)
    N=length(d)
    #A=a' and Y=y''
    A=Array{Float64}(N)
    Y=Array{Float64}(N)
    x=Array{Float64}(N)
    #timestep1
    A[1]=a[1]/d[1]
    Y[1]=y[1]/d[1]
    #steps 2 to N-1
    for i in 2:(N-1)
        A[i]=a[i]/(d[i]-b[i]*A[i-1])
        Y[i]=(y[i]-b[i]*Y[i-1])/(d[i]-b[i]*A[i-1])
    end
    #N-th value
    Y[N]=(y[N]-b[N]*Y[N-1])/(d[N]-b[N]*A[N-1])

    #calculating the x-Vector
    x[N]=Y[N]
    for i in (N-1):-1:1
       x[i]=Y[i]-A[i]*x[i+1]
    end
    return(x)
end
```

Out[3]:
```
thomasalgo (generic function with 1 method)
```

In [4]:
```
dx=0.05
xmax=10.
T0=3
dtx=-2



#making M
#dx:xmax
leng=floor(Int,xmax/dx)
diagonalelements=ones(leng)*(-2)/dx^2
#last element is different
diagonalelements[leng]+=1/dx^2
#offdiagonal quite easy
offdiagonala=ones(leng)*1/dx^2
offdiagonalb=ones(leng)*1/dx^2
offdiagonala[leng]=0.
offdiagonalb[1]=0
```

```
#M=
M=diagm(diagonalelements,0)+diagm(offdiagonala[1:leng-1],1)+diagm(offdiagonalb[2:leng]
```

Out[4]:
```
200E200 Array{Float64,2}:
 -800.0   400.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
  400.0  -800.0   400.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0   400.0  -800.0   400.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0   400.0  -800.0   400.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0   400.0  -800.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0   400.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0

    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0     400.0     0.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0    -800.0   400.0     0.0     0.0
    0.0     0.0     0.0     0.0     0.0     400.0  -800.0   400.0     0.0
    0.0     0.0     0.0     0.0     0.0       0.0   400.0  -800.0   400.0
    0.0     0.0     0.0     0.0     0.0       0.0     0.0   400.0  -400.0
```

In [5]:
```
#making C
h(x)=0.1*x

C=Array{Float64}(leng)
for (i,x) in enumerate(dx:dx:10)
    C[i]=h(x)
end
C[1]-=T0/dx^2
C[leng]-=dtx/dx

C
```

Out[5]:
```
200-element Array{Float64,1}:
 -1199.99
     0.01
     0.015
     0.02
```

```
   0.025
   0.03
   0.035
   0.04
   0.045
   0.05
   0.055
   0.06
   0.065
   ⋮
   0.945
   0.95
   0.955
   0.96
   0.965
   0.97
   0.975
   0.98
   0.985
   0.99
   0.995
  41.0
```

In [6]:
```
T=thomasalgo(diagonalelements,offdiagonala,offdiagonalb,C)
```

Out[6]:
```
200-element Array{Float64,1}:
   2.64875
   2.29751
   1.9463
   1.59512
   1.244
   0.892937
   0.54195
   0.19105
  -0.15975
  -0.510438
  -0.861
  -1.21143
  -1.5617
   ⋮
 -49.3215
 -49.4483
 -49.5728
 -49.6948
 -49.8145
 -49.9317
 -50.0465
 -50.1589
```

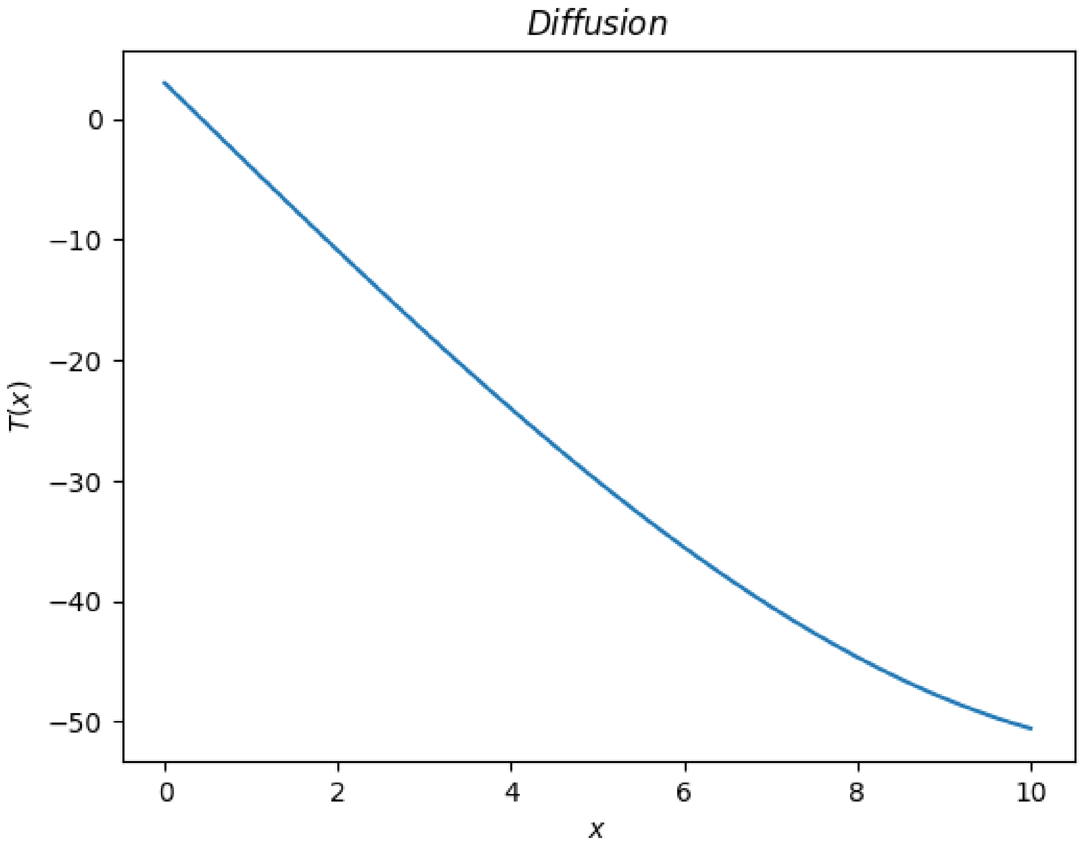
```
 -50.2688
 -50.3763
 -50.4813
 -50.5838
```

In [7]:
```
#just to test, seems like Julia has no problems to invert huge arrays
Xtest=inv(M)*C
```

Out[7]:
```
200-element Array{Float64,1}:
   2.64875
   2.29751
   1.9463
   1.59512
   1.244
   0.892937
   0.54195
   0.19105
  -0.15975
  -0.510438
  -0.861
  -1.21143
  -1.5617

 -49.3215
 -49.4483
 -49.5728
 -49.6948
 -49.8145
 -49.9317
 -50.0465
 -50.1589
 -50.2688
 -50.3763
 -50.4813
 -50.5838
```

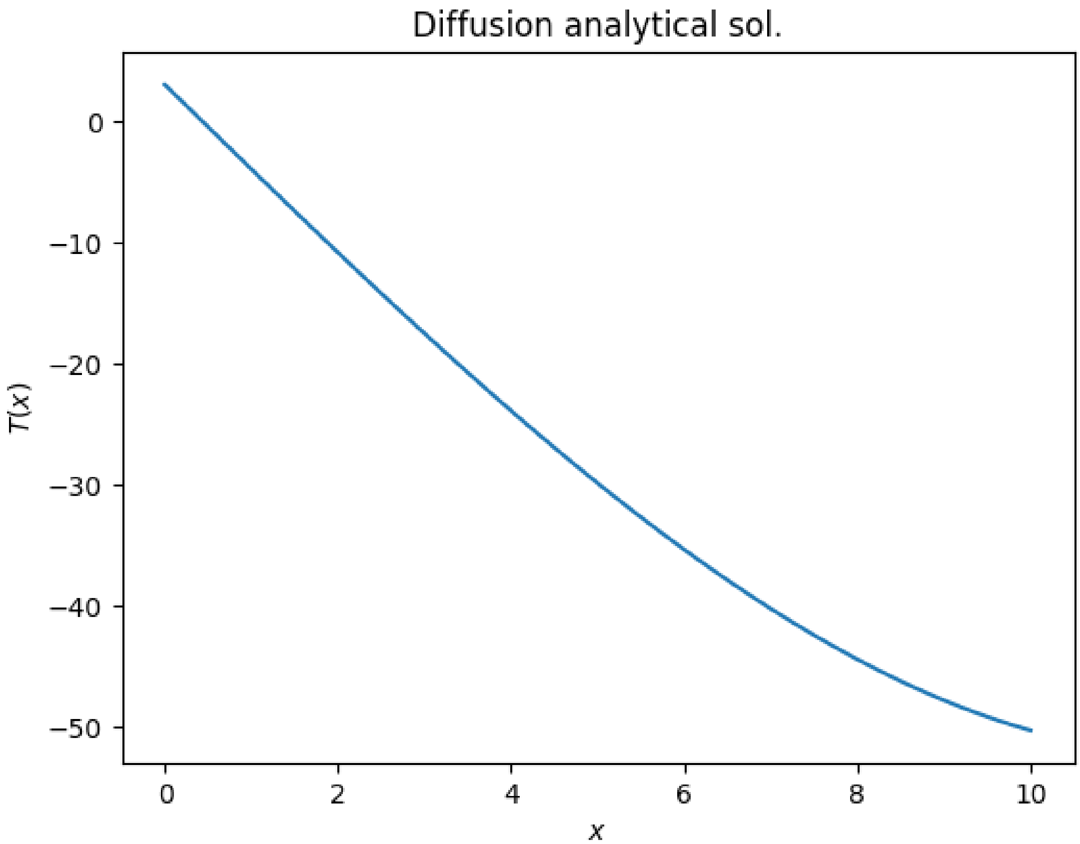
In [8]:
```
x=Array{Float64}(leng+1)
T2=Array{Float64}(leng+1)
x[1]=0
T2[1]=T0
for (i,x2) in enumerate(dx:dx:10)
    x[i+1]=x2
    T2[i+1]=T[i]
end

i=1:leng+1
plot(x[i],T2[i])
title(L"Diffusion")
```

```
ylabel(L"$T(x)$")
xlabel(L"$x$")
legend()
```

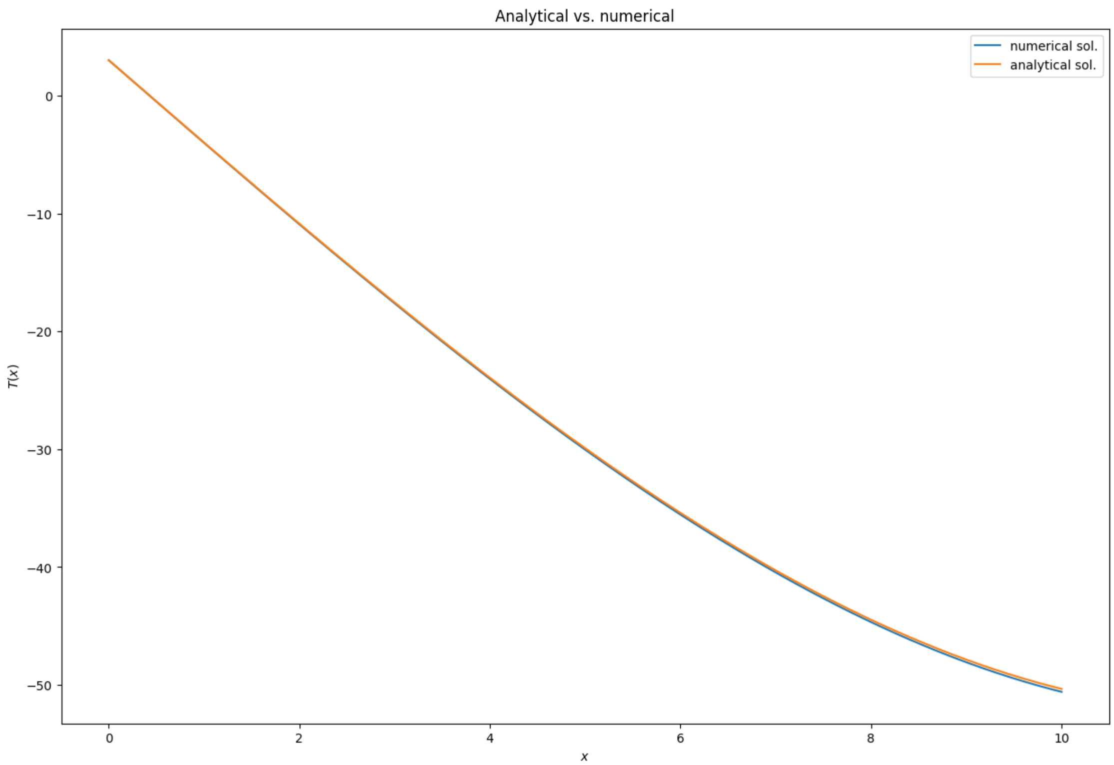


2.2 c)

```
In [9]: #analytical sol
        T3(x)=0.1/6*x.^3-7*x.+3
        x=0.:dx:10.
        plot(x,T3(x))
        title("Diffusion analytical sol.")
        ylabel(L"$T(x)$")
        xlabel(L"$x$")
        legend()
```




```
In [10]: figure(1,figsize=(15,10))
         plot(x[i],T2[i],label="numerical sol.")
         plot(x,T3(x),label="analytical sol.")
         title("Analytical vs. numerical")
         ylabel(L"$T(x)$")
         xlabel(L"$x$")
         legend()
```



Out[10]: PyObject <matplotlib.legend.Legend object at 0x7f3f6fb1dad0>

They’re almost the same, with a lower stepsize, the numerical approximation would even be better