sheet6

October 2, 2017

1 8.1 Linear system of ODEs

1.1 a)

```
In [1]: using PyPlot
```

```
In [2]: K= [0 sqrt(2) 0;
            sqrt(2) 0 sqrt(2);
            0 sqrt(2) 0]
```

```
Out[2]: 3×3 Array{Float64,2}:
         0.0      1.41421  0.0
         1.41421  0.0      1.41421
         0.0      1.41421  0.0
```

```
In [3]: Eig=eig(K)
```

```
Out[3]: ([-2.0,2.66454e-15,2.0],
        [0.5 -0.707107 0.5; -0.707107 -1.11022e-15 0.707107; 0.5 0.707107 0.5])
```

```
In [14]: println("Eigenvals+Vectors")
         for i in 0:2
            println("Eigenvalue ",i+1,"= ",Eig[1][i+1],"\nwith normalized Eigenvector ",Eig[2]
         end
```

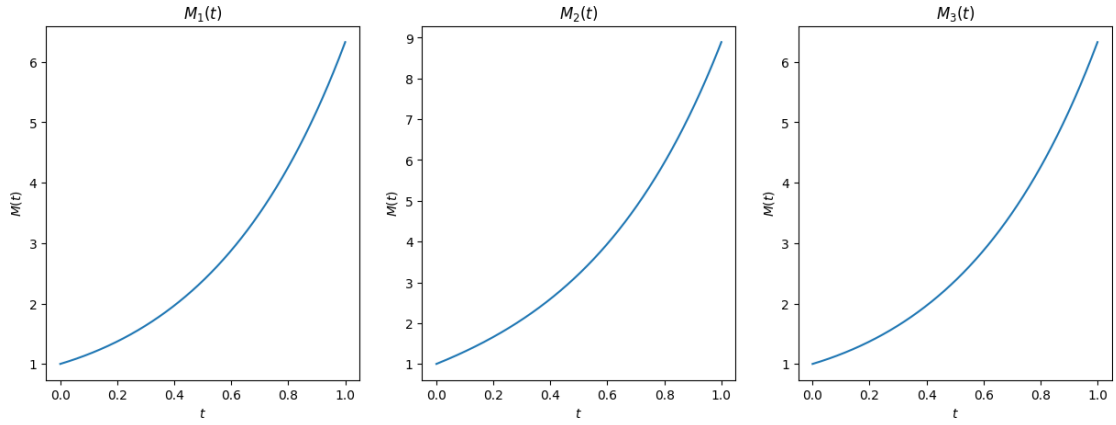
```
Eigenvals+Vectors
Eigenvalue 1= -1.999999999999997
with normalized Eigenvector [0.5,-0.707107,0.5]
Eigenvalue 2= 2.6645352591003757e-15
with normalized Eigenvector [-0.707107,-1.11022e-15,0.707107]
Eigenvalue 3= 2.0
with normalized Eigenvector [0.5,0.707107,0.5]
```

1.2 b)

```
In [5]: #Calculation for a0
        Eiginv=inv(Eig[2])
```

```
Out[5]: 3×3 Array{Float64,2}:
          0.5       -0.707107     0.5
         -0.707107  -6.66134e-16  0.707107
          0.5        0.707107     0.5

In [6]: M=[1.; 1.; 1.]

Out[6]: 3-element Array{Float64,1}:
         1.0
         1.0
         1.0

In [7]: #a0=...
        a=Eiginv*M

Out[7]: 3-element Array{Float64,1}:
         0.292893
         9.99201e-16
         1.70711

In [8]: #function for M(t)
        function m(t,a,gamma,e)
            m=[0;0;0]
            for i in 0:2
                #formula from the script
                m+=a[i+1]*exp(t*gamma[i+1])*e[i*3+1:i*3+3]
            end
            return m
        end

        m1=Float64[]
        m2=Float64[]
        m3=Float64[]
        xval=Float64[]

        for x in 0:0.001:1
            mvec=m(x,a,Eig[1],Eig[2])
            push!(m1,mvec[1])
            push!(m2,mvec[2])
            push!(m3,mvec[3])
            push!(xval,x)
        end

        figure(figsize=(15,5))
        subplot(131)
        plot(xval,m1)
        title(L"$M_1(t)$")
        ylabel(L"$M(t)$")
        xlabel(L"$t$")
```

```
legend()
subplot(132)
plot(xval,m2)
title(L"$M_2(t)$")
ylabel(L"$M(t)$")
xlabel(L"$t$")
legend()
subplot(133)
plot(xval,m3)
title(L"$M_3(t)$")
ylabel(L"$M(t)$")
xlabel(L"$t$")
legend()
```




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

1.3 c)

In [9]:
```
x=0:0.001:1
plot(x,m1(x))
```

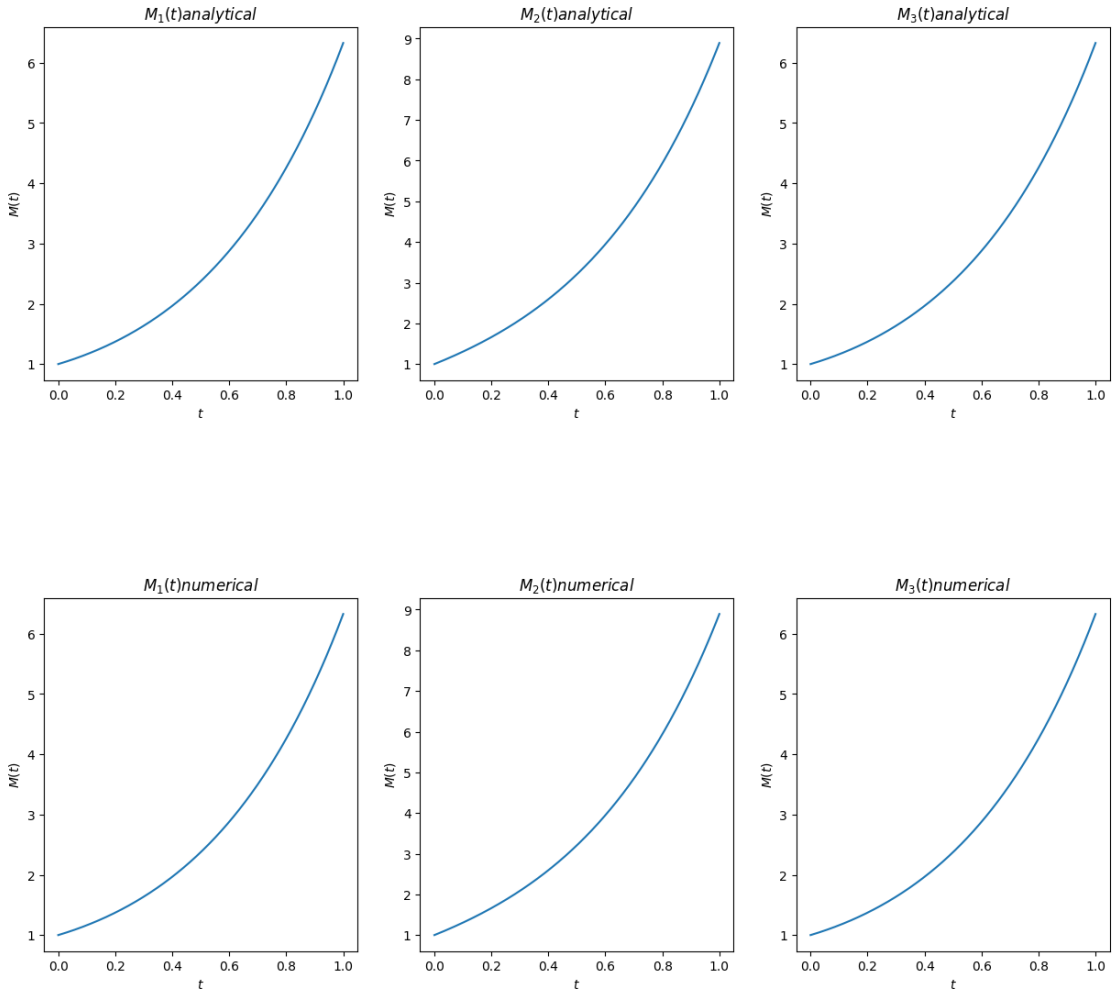
```
MethodError: objects of type Array{Float64,1} are not callable
Use square brackets [] for indexing an Array.

in include_string(::String, ::String) at ./loading.jl:441
```

In [10]:
```
#the EF-Scheme eye(3)-3x3 unit matrix
N(m,dt,M,K)=((eye(3)+K*dt)^m)*M
```

```
Out[10]: N (generic function with 1 method)
```

```
In [11]: dt=0.001
         m1num=Float64[]
         m2num=Float64[]
         m3num=Float64[]
         xval=Float64[]

         #i-iteration,x-value in the interval
         for (i,x) in enumerate(0:dt:1)
             mvec=N(i,dt,M,K)
             push!(m1num,mvec[1])
             push!(m2num,mvec[2])
             push!(m3num,mvec[3])
             push!(xval,x)
         end

         figure(figsize=(15,5))
         subplot(131)
         plot(xval,m1)
         title(L"$M_1(t) analytical$")
         ylabel(L"$M(t)$")
         xlabel(L"$t$")
         legend()
         subplot(132)
         plot(xval,m2)
         title(L"$M_2(t) analytical$")
         ylabel(L"$M(t)$")
         xlabel(L"$t$")
         legend()
         subplot(133)
         plot(xval,m3)
         title(L"$M_3(t) analytical$")
         ylabel(L"$M(t)$")
         xlabel(L"$t$")
         legend()
         figure(figsize=(15,5))
         subplot(131)
         plot(xval,m1num)
         title(L"$M_1(t) numerical$")
         ylabel(L"$M(t)$")
         xlabel(L"$t$")
         legend()
         subplot(132)
         plot(xval,m2num)
         title(L"$M_2(t) numerical$")
         ylabel(L"$M(t)$")
         xlabel(L"$t$")
```

```
legend()
subplot(133)
plot(xval,m3num)
title(L"$M_3(t) numerical$")
ylabel(L"$M(t)$")
xlabel(L"$t$")
legend()
```



I did it in 2 plots, because the difference is so small, you could not differentiate between the numerical and analytical values

1.4 d)

No, since there are only positive values >1 in the matrix, there should be at least one eigenvalue >1. Also all M(t) grow exponentialy, therefore M_{EF} has to be >1, since N(0)=const

2 9. Nuclear decay chain

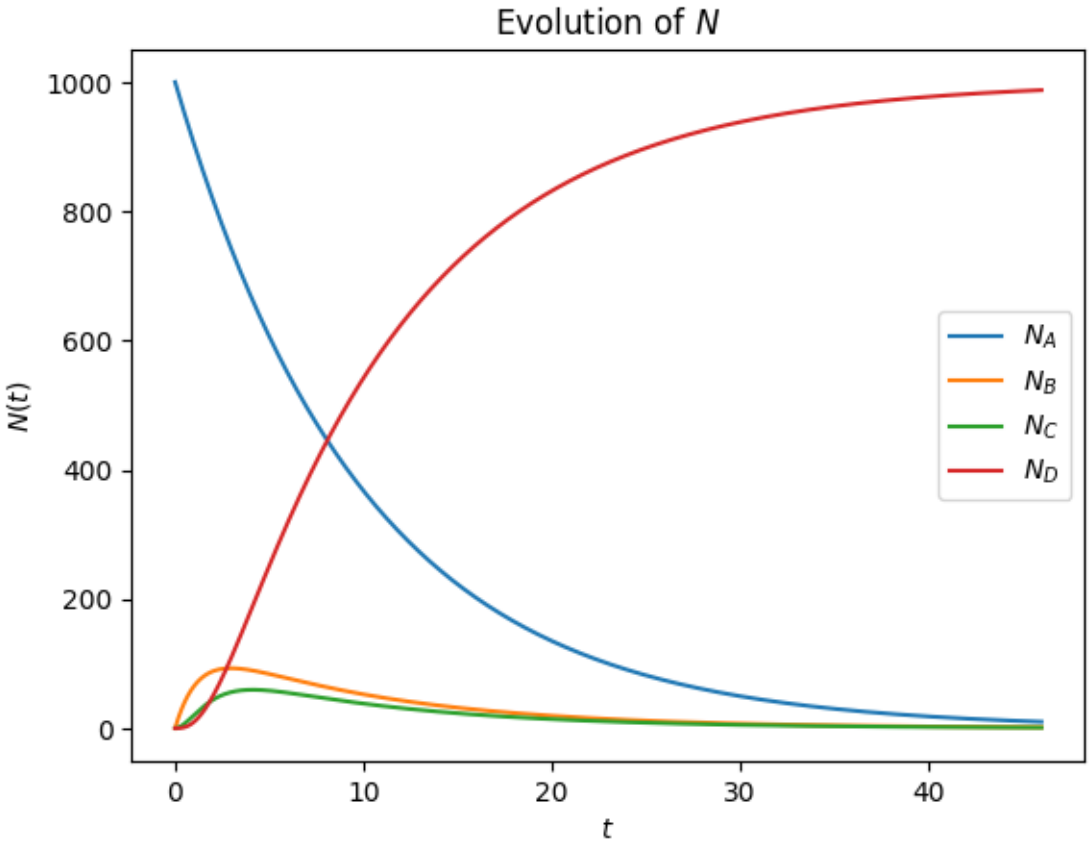
2.1 b)+d)

In [12]:
```
using PyPlot
#the EB-Scheme eye(4)-4x4 unit matrix
N2(m,dt,M,K)=(inv((eye(4)-K*dt))^m)*M
```

Out[12]:
```
N2 (generic function with 1 method)
```

In [13]:
```
a=0.1
b=0.8
c=1.2
M=[1000.; 0.; 0.;0.]
K=[-a 0 0 0;
   a -b 0 0;
   0 b -c 0;
   0 0 c 0]
dt=0.001

m1num=Float64[]
m2num=Float64[]
m3num=Float64[]
m4num=Float64[]
xval=Float64[]

i=0
mvec=N2(i,dt,M,K)
push!(m1num,mvec[1])
push!(m2num,mvec[2])
push!(m3num,mvec[3])
push!(m4num,mvec[4])
push!(xval,i*dt)

#i-iteration,x-value in the interval
#i+1 since i starts from 0, but julia arrays start from 1
while(m1num[i+1]>=10)
    i+=1
    mvec=N2(i,dt,M,K)
    push!(m1num,mvec[1])
    push!(m2num,mvec[2])
    push!(m3num,mvec[3])
    push!(m4num,mvec[4])
    push!(xval,i*dt)
end

plot(xval,m1num,label=L"$N_A$")
plot(xval,m2num,label=L"$N_B$")
plot(xval,m3num,label=L"$N_C$")
```

```
plot(xval,m4num,label=L"$N_D$")
title(L"Evolution of $N$")
ylabel(L"$N(t)$")
xlabel(L"$t$")
legend()
```



Out[13]: PyObject <matplotlib.legend.Legend object at 0x7fddc4a96510>

NA evolves with the exponent -0.1 to NB and has no ingoing rate, therefore NA decreases exponentially. Since the ingoing rates to NB and NC are lower then theyr outgoing, theyr values first increase (because the reservoirs on the left side are bigger and therefore compensate the difference between the in-and outgoing rates) and then decrease exponentially (because the value on the left side cannot compensate the ratedifference anymore). ND is the stable element -> no outgoing rate, therefore the shape of the curve is logarithmic. All isotopes evolve towards that element, but the particlenumber decreases -> the slope decreases.

In [ ]:

9.

a) $\frac{dN_A}{dt} = -\lambda_A N_A$ $\qquad \frac{dN_B}{dt} = \lambda_A N_A - \lambda_B N_B$

$\frac{dN_C}{dt} = \lambda_B N_B - \lambda_C N_C$ $\qquad \frac{dN_D}{dt} = \lambda_C N_C$

$$\frac{d\underline{N}}{dt} = \begin{pmatrix} -\lambda_A & 0 & 0 & 0 \\ \lambda_A & -\lambda_B & 0 & 0 \\ 0 & \lambda_B & -\lambda_C & 0 \\ 0 & 0 & \lambda_C & 0 \end{pmatrix}$$

b) $\underline{N(m\Delta t)} = \underline{\underline{M_{EB}}}^{m} \underline{N(0)}$ — just this formula

$\underline{\underline{M}} = \underline{\underline{1}} - \underline{\underline{K}}\Delta t$

$\rightarrow \underline{N(m\Delta t)} = \sum_i a_i \frac{1}{\gamma_i^m} \underline{e_i}$

γ_i - eigenvals of $\underline{\underline{M}} - \underline{\underline{K}}\Delta t$

$\underline{e_i}$ - eigenvectors

$a_i = \underline{\underline{E}}^{-1} \cdot \underline{M}(t=0)$

$\rightarrow$ inverse of $\underline{\underline{E}} = \begin{pmatrix} \uparrow & \uparrow & & \uparrow \\ \underline{e_1} & \underline{e_2} & \cdots & \underline{e_n} \\ \downarrow & \downarrow & & \downarrow \end{pmatrix}$