sheet3

September 14, 2017

1 4. Nuclear decay eq.

1.1 a)

The EF-scheme with user input

```
In [1]: function ef()
            print("N0: "); N0 = parse(Int,readline(STDIN))
            print("lambda: "); lambda = parse(Float64,readline(STDIN))
            print("dt: "); dt = parse(Float64,readline(STDIN))
            print("t: "); t = parse(Float64,readline(STDIN))
            #determinating the numb of timesteps
            m=t/dt
            #marching eq of EF
            return(N0*(1-lambda*dt)^m)
        end
```

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

Testing it once for $N_0 = 1$, $\lambda = 1.2$, $\Delta t = 2$ and $t = 20$

```
In [2]: print("N(t)= ",ef())
```

```
N0: STDIN> 1
lambda: STDIN> 1.2
dt: STDIN> 2
t: STDIN> 20
N(t)= 28.92546549759998
```

As we see in c), it seems to work with user input

1.2 b)

```
In [3]: using PyPlot
```

I rewrote the EF-scheme, because I prefer to set the parameters within the code.

```
In [5]: function ef2(lambda,N0,dt,t)
            #determinating the numb of timesteps
            m=t/dt
            #marching eq of EF
            return(N0*(1-lambda*dt)^m)
        end
```

```
Out[5]: ef2 (generic function with 1 method)
```

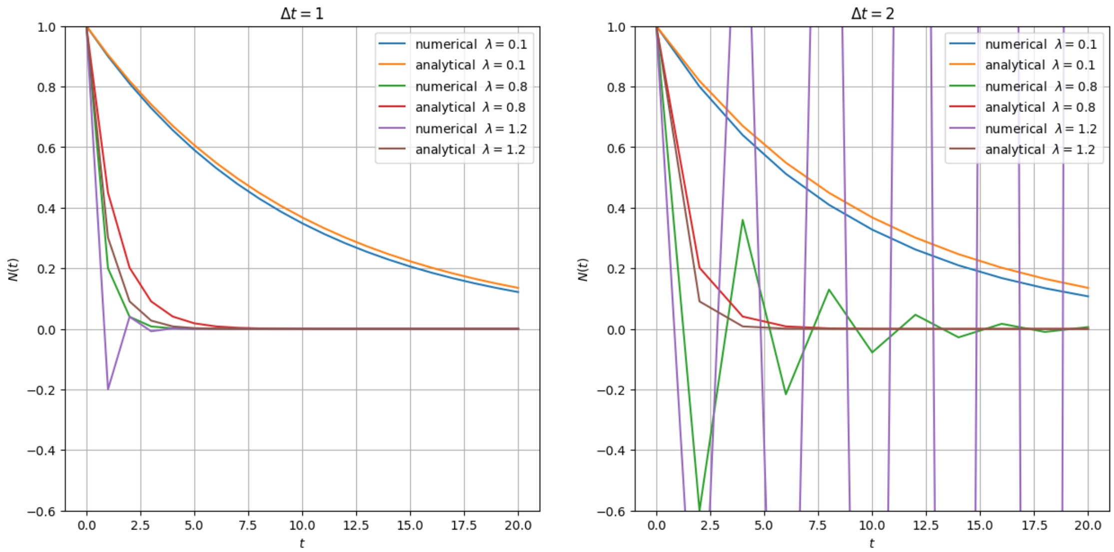
```
In [6]: #the analytical sol.
        f(t,N0,lambda)=N0*exp(-lambda*t)

        #array for num results
        valuesnum=Float64[]
        #for analytical
        valuesana=Float64[]
        #parameters
        N0=1
        lambdaarray=[0.1,0.8,1.2]

        #ploting commands
        figure(1,figsize=(15,7))
        #2figs in line, linenumber=1, rownumber=2,number of figure=0
        z= 120

        #for every set of param do
        for dt in 1:2
            for lamb in 1:length(lambdaarray)
                for t in 0:dt:20
                    #vals for num
                    push!(valuesnum,ef2(lambdaarray[lamb],N0,dt,t))
                    #analytical vals
                    push!(valuesana,f(t,N0,lambdaarray[lamb]))
                end
                #number of the figure, 1 or 2 corresponding to dt=1 and dt=2
                subplot(z+dt)
                plot(0:dt:20,valuesnum[:],label="numerical  "*L"$\lambda=$"*string(lambdaarray
                plot(0:dt:20,valuesana[:],label="analytical  "*L"$\lambda=$"*string(lambdaarray
                grid("on")
                title(L"$\Delta t= $"*string(dt))
                ylabel(L"$N(t)$")
                xlabel(L"$t$")
                legend()
                #rescalement of the plot, because of the unstable values...
                ax=gca()
                ax[:set_ylim]([-0.6,1])
                #clearing the arrays
                valuesnum=[]
```

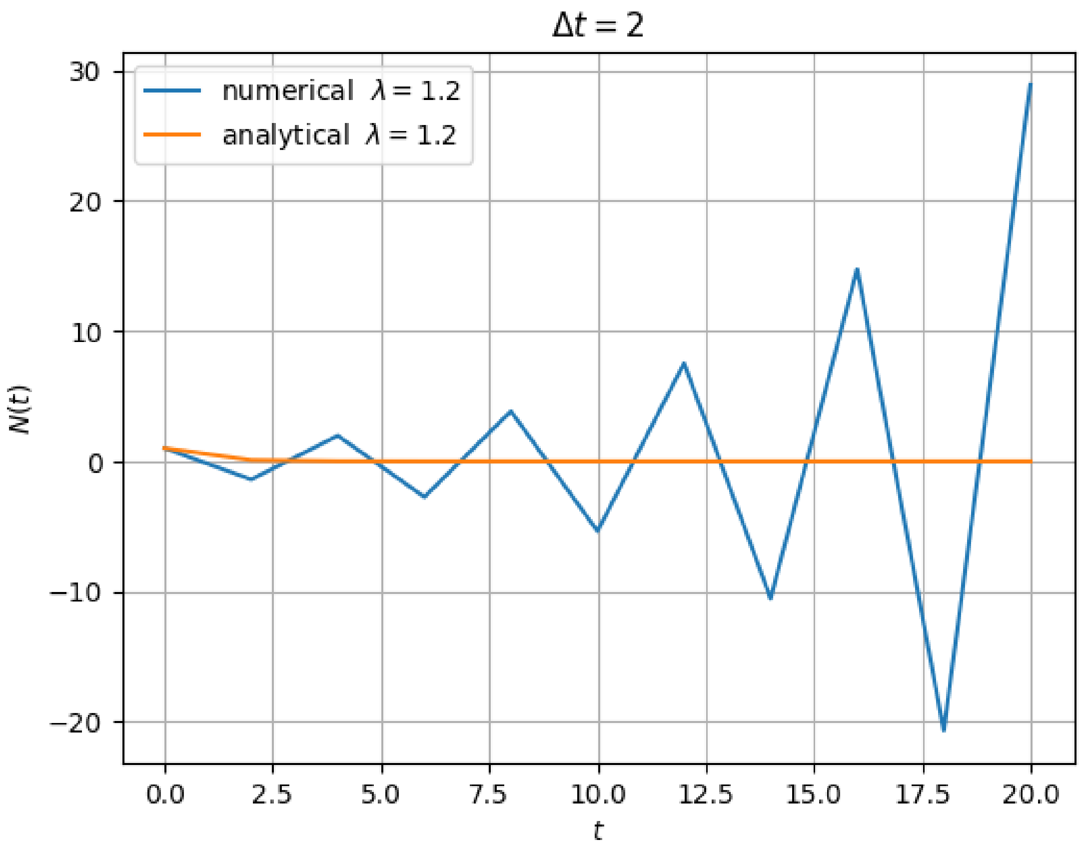
```
        valuesana=[]
    end
end
```



1.3 c)

plot the unstable $\Delta t = 2 \wedge \lambda = 1.2$ again

In [7]:
```
N0=1
lambda=1.2
dt=2
for t in 0:dt:20
   #vals for num
    push!(valuesnum,ef2(lambda,N0,dt,t))
    #analytical vals
    push!(valuesana,f(t,N0,lambda))
end
plot(0:dt:20,valuesnum[:],label="numerical  "*L"$\lambda=$"*string(lambda))
plot(0:dt:20,valuesana[:],label="analytical  "*L"$\lambda=$"*string(lambda))
grid("on")
title(L"$\Delta t= $"*string(dt))
ylabel(L"$N(t)$")
xlabel(L"$t$")
legend()
```




```
Out[7]: PyObject <matplotlib.legend.Legend object at 0x7f93494ce190>
```

As we see, the unstable case is oscillating around the analytical values. Since the marching eq. is given by $N(m\Delta t) = N(0)(1 - \lambda\Delta t)^m$, the positive values correspond to even m's and the negative values to odd m's. Due to our marching eq. the numerical error stacks with the timesteps and grows therefore

1.4 d)

```
In [8]: function eb(lambda,N0,dt,t)
            #determinating the numb of timesteps
            m=t/dt
            #marching eq of EB
            return(N0*(1+lambda*dt)^(-m))
        end
```

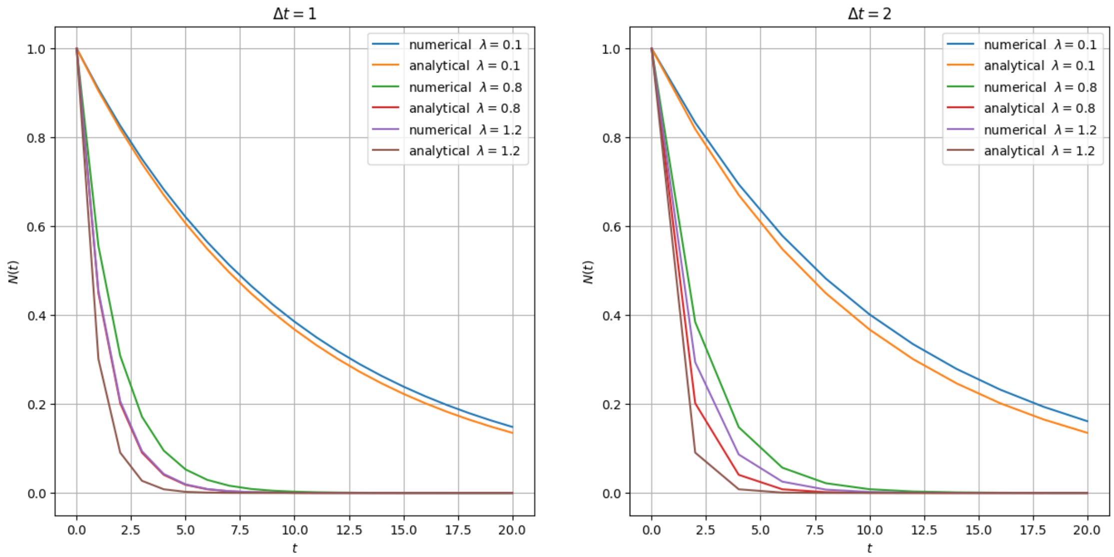
```
Out[8]: eb (generic function with 1 method)
```

```
In [9]: #the analytical sol.
        f(t,N0,lambda)=N0*exp(-lambda*t)
```

```
#array for num results
valuesnum=Float64[]
#for analytical
valuesana=Float64[]
#parameters
N0=1
lambdaarray=[0.1,0.8,1.2]

#ploting commands
figure(1,figsize=(15,7))
#2figs in line, linenumber=1, rownumber=2,number of figure=0
z= 120

#for every set of param do
for dt in 1:2
    for lamb in 1:length(lambdaarray)
        for t in 0:dt:20
            #vals for num
            push!(valuesnum,eb(lambdaarray[lamb],N0,dt,t))
            #analytical vals
            push!(valuesana,f(t,N0,lambdaarray[lamb]))
        end
        #number of the figure, 1 or 2 corresponding to dt=1 and dt=2
        subplot(z+dt)
        plot(0:dt:20,valuesnum[:],label="numerical  "*L"$\lambda=$"*string(lambdaarray
        plot(0:dt:20,valuesana[:],label="analytical  "*L"$\lambda=$"*string(lambdaarra
        grid("on")
        title(L"$\Delta t= $"*string(dt))
        ylabel(L"$N(t)$")
        xlabel(L"$t$")
        legend()
        valuesnum=[]
        valuesana=[]
    end
end
```




```
WARNING: Method definition f(Any, Any, Any) in module Main at In[6]:2 overwritten at In[9]:2.
```

No signs of numerical instability, the numerical values follow the analytical

1.5 e)

In [10]:
```
function ec(lambda,N0,dt,t)
    if (t==0)
        return(N0)
    end
    Nt0=N0
    #determing N1 value with eb, t=dt-> just one timestep
    Nt1=eb(lambda,N0,dt,dt)
    m=Int(t/dt)
    for i in 1:m-1
        Nt1=Nt0-2*lambda*dt*Nt1
        Nt0=Nt1
    end
    #marching eq of EB
    return(Nt1)
end
```

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

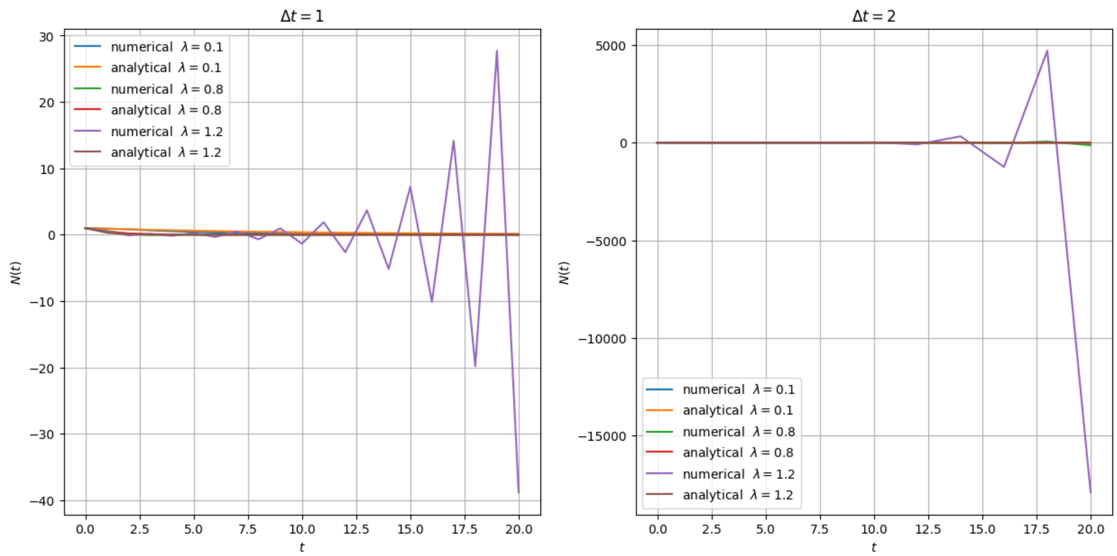
In [11]:
```
#the analytical sol.
f(t,N0,lambda)=N0*exp(-lambda*t)

#array for num results
```

```
valuesnum=Float64[]
#for analytical
valuesana=Float64[]
#parameters
N0=1
lambdaarray=[0.1,0.8,1.2]

#ploting commands
figure(1,figsize=(15,7))
#2figs in line, linenumber=1, rownumber=2,number of figure=0
z= 120

#for every set of param do
for dt in 1:2
    for lamb in 1:length(lambdaarray)
        for t in 0:dt:20
            #vals for num
            push!(valuesnum,ec(lambdaarray[lamb],N0,dt,t))
            #analytical vals
            push!(valuesana,f(t,N0,lambdaarray[lamb]))
        end
        #number of the figure, 1 or 2 corresponding to dt=1 and dt=2
        subplot(z+dt)
        plot(0:dt:20,valuesnum[:],label="numerical  "*L"$\lambda=$"*string(lambdaarray
        plot(0:dt:20,valuesana[:],label="analytical  "*L"$\lambda=$"*string(lambdaarra
        grid("on")
        title(L"$\Delta t= $"*string(dt))
        ylabel(L"$N(t)$")
        xlabel(L"$t$")
        legend()
        valuesnum=[]
        valuesana=[]
    end
end
```




```
WARNING: Method definition f(Any, Any, Any) in module Main at In[9]:2 overwritten at In[11]:2.
```

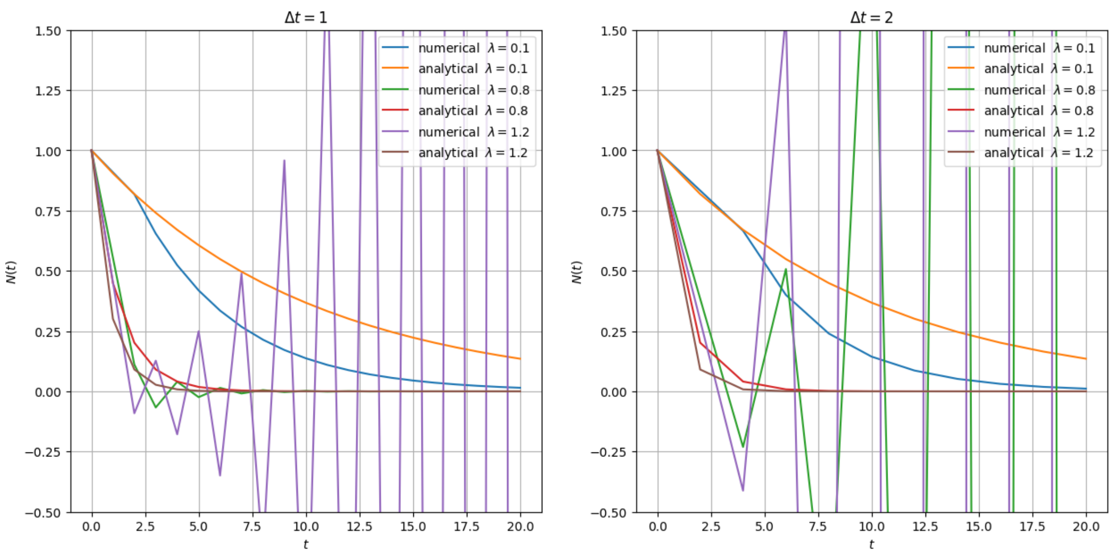
Rescale the Plot to y=[-0.5,1.5]

```
In [12]: #the analytical sol.
         f(t,N0,lambda)=N0*exp(-lambda*t)

         #array for num results
         valuesnum=Float64[]
         #for analytical
         valuesana=Float64[]
         #parameters
         N0=1
         lambdaarray=[0.1,0.8,1.2]
         t=20
         #ploting commands
         figure(1,figsize=(15,7))
         #2figs in line, linenumber=1, rownumber=2,number of figure=0
         z= 120

         #for every set of param do
         for dt in 1:2
             for lamb in 1:length(lambdaarray)
                 for t in 0:dt:t
                     #vals for num
                     push!(valuesnum,ec(lambdaarray[lamb],N0,dt,t))
                     #analytical vals
                     push!(valuesana,f(t,N0,lambdaarray[lamb]))
```

```
            end
            #number of the figure, 1 or 2 corresponding to dt=1 and dt=2
            subplot(z+dt)
            plot(0:dt:t,valuesnum[:],label="numerical  "*L"$\lambda=$"*string(lambdaarray
            plot(0:dt:t,valuesana[:],label="analytical  "*L"$\lambda=$"*string(lambdaarray
            grid("on")
            title(L"$\Delta t= $"*string(dt))
            ylabel(L"$N(t)$")
            xlabel(L"$t$")
            #rescalement of the plot, because of the unstable values...
            ax=gca()
            ax[:set_ylim]([-0.5,1.5])
            legend()
            valuesnum=[]
            valuesana=[]
        end
    end
```




```
WARNING: Method definition f(Any, Any, Any) in module Main at In[11]:2 overwritten at In[12]:2
```

1.6 e)

Regarding to our plots I would use the EB-scheme, because it's the only scheme in which the stability cond. is fullfilled (since the EF is conditionally stable and the EC unconditionally unstable)

In [ ]: