Numerical Algorithms - Assignment 3

Sebastiano Smaniotto 857744

January 2021

Introduction

In order to carry out the assignment we have used the programming language Python. To make sure that there are no compatibility problem, I have put the file `numalg.yaml` in the submission folder which contains the conda environment that we have used for the computation. In the spirit of the assignment, we have implemented from scratch all methods and functions that were required as intended.

1 Spring Equation

In this section we report and comment the results of using our implementation of Forward Euler, Backward Euler and Crank-Nicolson methods for numerical integration of ODE systems applied to a spring system with equation

$$\begin{cases} y'(t) = v(t) \\ v'(t) = -\frac{k}{m} y(t) \end{cases}$$

with $k = 1$ kg/s^2, $m = 0.1$ kg. We had to solve the CP with initial conditions $t_0 = 0$ s, $y_0 = 1$ m and $T = 15$ s.

In Figure 1 it is possible to compare the profiles of the curves obtained by numerical integration of the ODE. As expected, the Forward Euler method do not approximate well the solution because it is unstable for $h = 2^{-1} = 0.5$. Also, in Figure 2 it is possible to see that its absolute error is much higher than that of any other methods for $N = 1$.

On the right-hand side panel of Figure 1 we can compare the curves for $N = 8$. In accordance with our expectations, Forward and Backward Euler estimates are

	CPU Time Elapsed			
h	Forward Euler	Backward Euler	Crank-Nicolson	scipy.integrate.odeint()
5.0000e-01	0.0000e+00	0.0000e+00	1.5625e-02	0.0000e+00
2.5000e-01	0.0000e+00	1.5625e-02	1.5625e-02	0.0000e+00
1.2500e-01	0.0000e+00	1.5625e-02	3.1250e-02	0.0000e+00
6.2500e-02	0.0000e+00	4.6875e-02	4.6875e-02	0.0000e+00
3.1250e-02	0.0000e+00	6.2500e-02	7.8125e-02	1.5625e-02
1.5625e-02	0.0000e+00	1.0938e-01	1.5625e-01	0.0000e+00
7.8125e-03	1.5625e-02	2.3438e-01	2.9688e-01	0.0000e+00
3.9062e-03	1.5625e-02	4.6875e-01	6.2500e-01	0.0000e+00

Table 1: Spring model, $\tau = 1e{-}9$.

always larger than smaller than the analytical solution, respectively, whereas Crank-Nicolson behaves similarly as the analytical solution, and so does the higher-order library function `scipy.integrate.odeint()`.

On the left-hand side panel of Figure 2 we can see how the absolute error $e^{(N)}$ decreases as N increases. We can clearly see that Crank-Nicolson is the fastest to converge to zero, whereas FE and BE takes up to $N = 6$ to reach an error close enough to zero. The higher-order library function `scipy.integrate.odeint()` is able to perform with a very small error since the first step, at $N = 1$.

On the right-hand side panel of Figure 2 it is possible to compare the relative error $e^{(N)}/e^{(N-1)}$. Except for some extreme values ($N = 4$ for FE, $N = 5$ for BE, $N = 7$ for CN) the error ratio seems to maintain a linear trend in all cases.

The absolute errore profile for $\tau = 1e{-}9$ and $\tau = 1e{-}3$ is the same up to a couple of differences. It can be seen by comparing Figure 2 and Figure 3, respectively. The error ratio of the Crank-Nicolson method for $\tau = 1e{-}3$ has a spike at $N = 7$, and the error ratio of the Backward Euler method for $\tau = 1e{-}9$ has a spike at $N = 8$.

The CPU time for each h and each method is reported in Table 1. The average number of nonlinear interation of the nonlinear solver and the number of time steps without convergence of the same are reported in Table 2.

	Avg Nonlinear Iterations		Time Steps with no Convergence	
h	Backward Euler	Crank-Nicolson	Backward Euler	Crank-Nicolson
5.0000e-01	6.1333e+00	6.9333e+00	0.0000e+00	0.0000e+00
2.5000e-01	5.7833e+00	5.8667e+00	0.0000e+00	0.0000e+00
1.2500e-01	5.9490e+00	5.9573e+00	0.0000e+00	0.0000e+00
6.2500e-02	5.6083e+00	5.2625e+00	0.0000e+00	0.0000e+00
3.1250e-02	4.5943e+00	4.0999e+00	0.0000e+00	0.0000e+00
1.5625e-02	3.9964e+00	3.9958e+00	0.0000e+00	0.0000e+00
7.8125e-03	3.9981e+00	3.9979e+00	0.0000e+00	0.0000e+00
3.9062e-03	3.9988e+00	3.9974e+00	0.0000e+00	0.0000e+00

Table 2: Spring model, $\tau = 1e{-}9$.

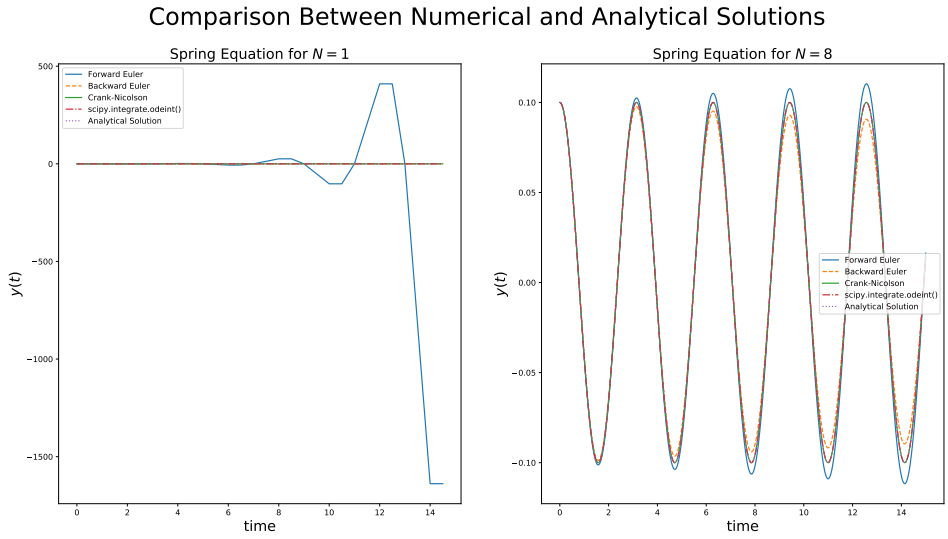

Figure 1: Spring model, $\tau = 1e{-}9$.

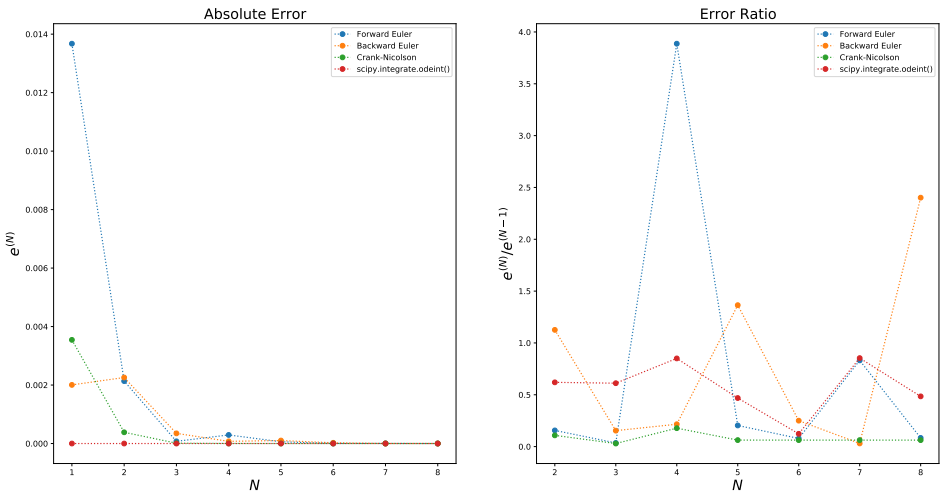

Figure 2: Spring model.

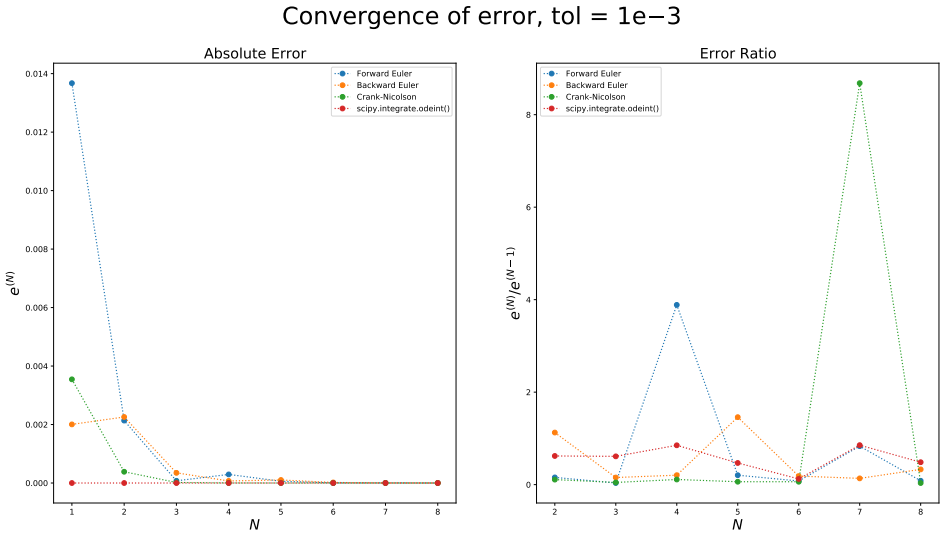

Figure 3: Spring model.

2 Epidemiological Model

We were tasked to use Crank-Nicolson to solve the following system of ODEs:

$$\begin{aligned}
\dot{S}(t) &= -\beta_0(I + \beta_A A)\frac{S}{N_p - D} \\
\dot{E}(t) &= \beta_0(I + \beta_A A)\frac{S}{N_p - D} - \delta E \\
\dot{I}(t) &= \sigma\delta E - \gamma_I I - \alpha I \\
\dot{A}(t) &= (1 - \sigma)\delta E - \gamma_A A \\
\dot{R}(t) &= \gamma_I I + \gamma_A A \\
\dot{D}(t) &= \alpha I.
\end{aligned}$$

We had to use FE, BE and CN methods to perform numerical integration and use MATLAB higher order solver `ode45()` as reference solution for the computation of errors. Since we have used Python for the assignment implementation, we have used the function `scipy.integrate.odeint()` instead. The model parameters values have been copied from the script `SEIARD_model.m` given by Prof. Pasetto for the assignment. The population size $N_p = 4459453$ is that of the number of residents in Emilia Romagna of 01/01/2019 available from the ISTAT web portal (link in the script).

The result of numerical integration for $N = 0$ is displayed in Figure 4. We can see that FE is below the reference solution of `scipy.integrate.odeint()` when $dy/dt < 0$ and above the reference when $dy/dt > 0$. The contrary holds for BE. CN curve perfectly overlap the reference curve, and in fact the error is always very close to zero as it is possible to see on the left-hand side panel of Figure 5.

As of errors, if a smaller tolerance for the nonlinear solver is used then CN performs much better, even for small values of N, as visible on the left-hand side panel of Figure 5. For $\tau = 1\text{e}{-3}$, the absolute error of each model decrease much slower with respect to N. In Both cases, CN has a sharp increase of error at $N = 4$.

To conclude, the data on the number of deaths, infected and recovered during the time period is not available. By using the data available from the previous assignment, and by defining the number of infected people $II(t) := I(t) + A(t)$, the number of estimated infection averted rounded up to integers is equal to $II(52) - i(52) = 676909$, where $i(t)$ is the number of infected people at time t.

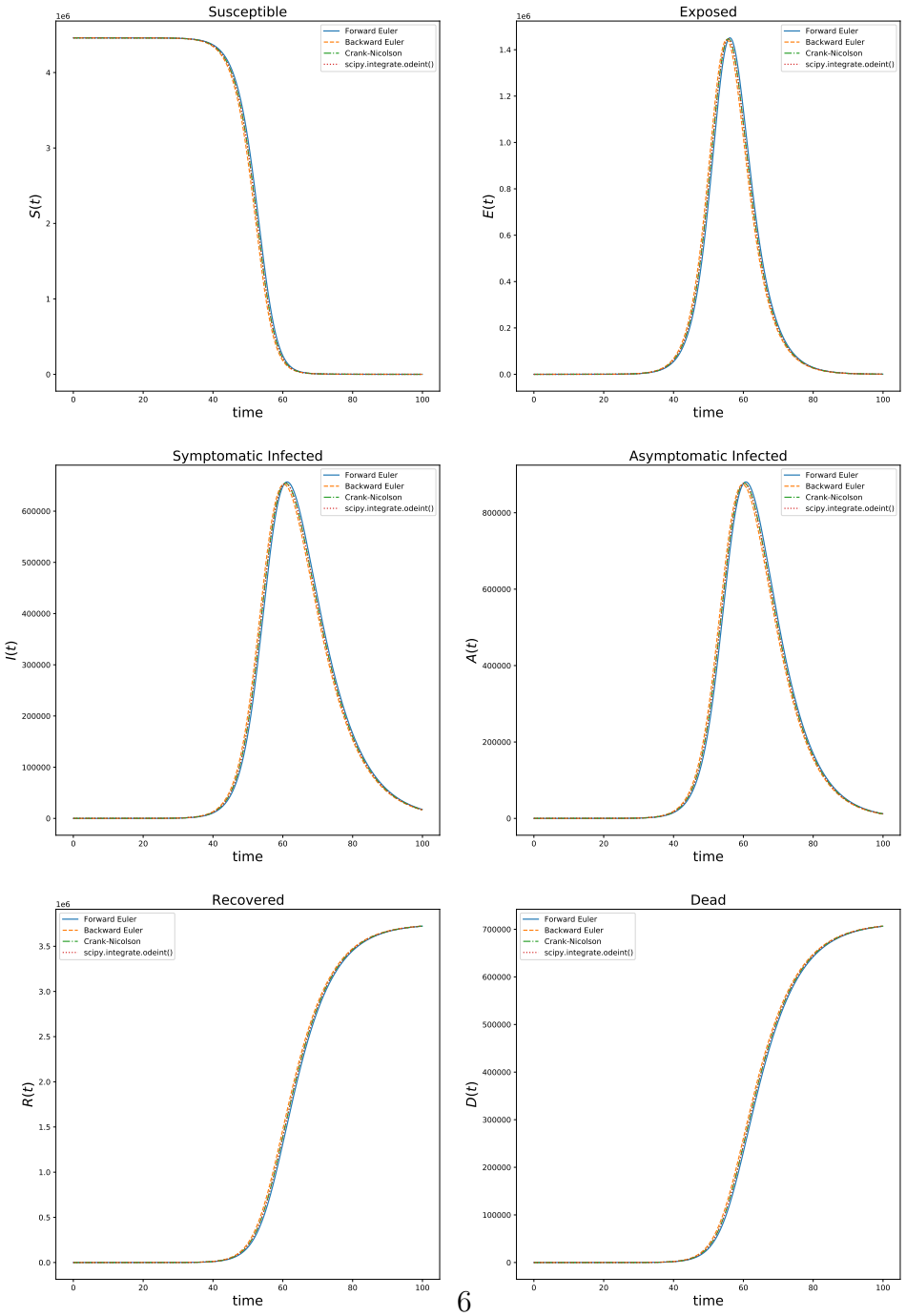

Figure 4: SEIARD model, $\tau = 1\text{e}{-9}$.

	CPU Time Elapsed			
h	Forward Euler	Backward Euler	Crank-Nicolson	scipy.integrate.odeint()
5.0000e-01	0.0000e+00	1.0938e-01	1.2500e-01	0.0000e+00
2.5000e-01	0.0000e+00	1.5625e-01	1.8750e-01	0.0000e+00
1.2500e-01	1.5625e-02	2.1875e-01	2.8125e-01	0.0000e+00
6.2500e-02	1.5625e-02	3.7500e-01	4.8438e-01	1.5625e-02

Table 3: SEIARD model, $\tau = 1e{-}9$

	Avg Nonlinear Iterations		Time Steps with no Convergence	
h	Backward Euler	Crank-Nicolson	Backward Euler	Crank-Nicolson
5.0000e-01	1.2930e+01	9.3475e+00	0.0000e+00	0.0000e+00
2.5000e-01	-9.6324e+98	6.7419e+00	0.0000e+00	0.0000e+00
1.2500e-01	6.1612e+00	5.1025e+00	0.0000e+00	0.0000e+00
6.2500e-02	4.9294e+00	4.4106e+00	0.0000e+00	0.0000e+00

Table 4: SEIARD model, $\tau = 1e{-}9$

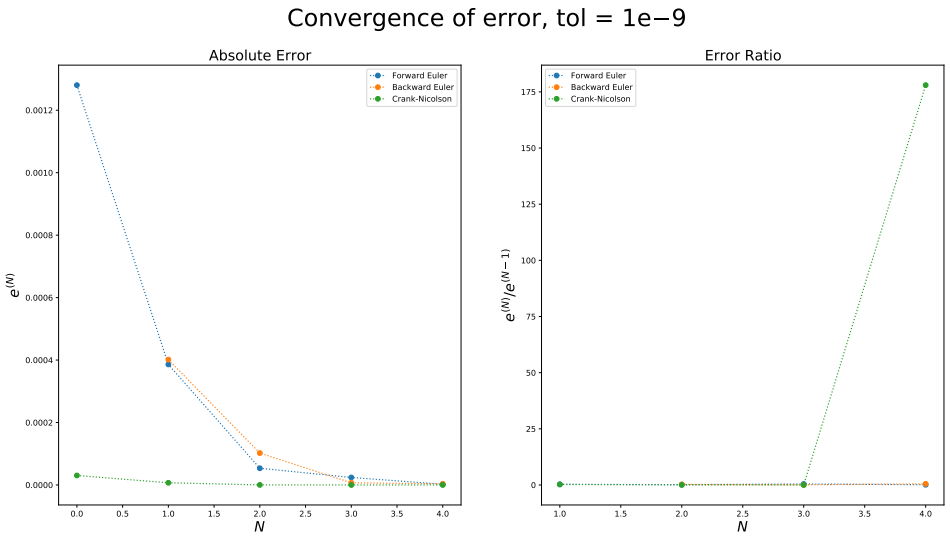

Figure 5: SEIARD model.

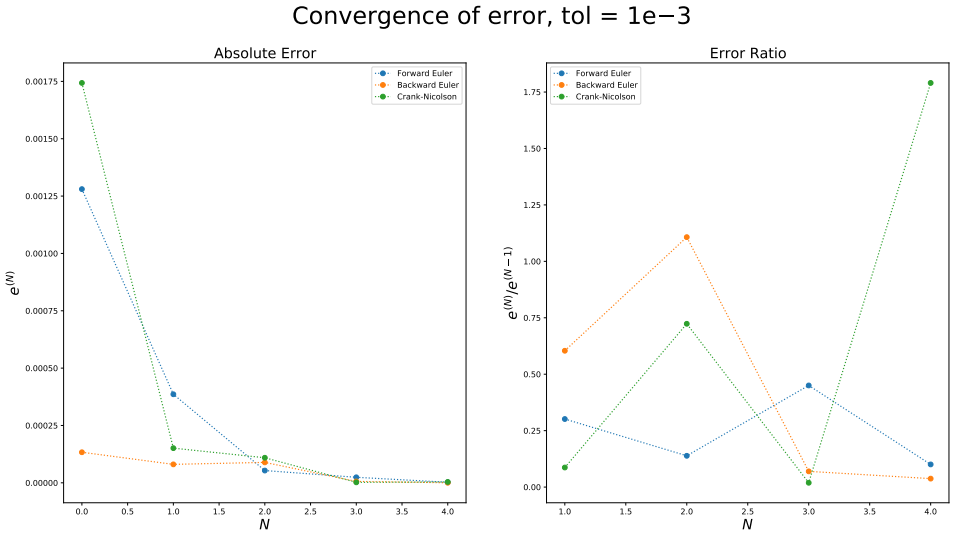

Figure 6: SEIARD model.