Numerical solution of time-dependent problems

Alberto Tibaldi

December 6, 2021

Contents

1 Introduction

The scope of this section is to discuss the numerical solution of transient (time-domain) problems. Most of this formulation is based on the thesis of Sohan Dharmaraja in MIT [1], tutored by prof. Gilbert Strang. Also, prof. Strang's webpages/books provide a lot of useful material [2], including scripts producing part of the results of the thesis, used also here.

The fundamental motivation of this section is: there are many, many numerical schemes aimed at solving time-dependent problems on the basis of a time discretization. This is because different problems may have different criticalities, be simpler or harder than others. Rather than discussing *complexity*, the sensitive word in this context is *stiffness*: a *stiff* problem is generally more difficult to be solved than a *non-stiff* one. Hence, it will require peculiar numerical schemes. This definition is quite hand-wavy, and does not give a precise idea of what *stiff* means or even implies. A first implication is that, in *stiff* problems, the time step size for the solution is more severely limited by the stability of the method than by its accuracy. Of course, this definition is kinda decepting, because it relies on other definitions such as *stability* and *accuracy*, not reported yet.

With *steps* we mean *time steps*, *i.e.,* the time samples related to the time discretization. Such discretization is needed because we have to approximate the time derivatives in view of making the problem treatable by a computer. For example, consider the equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y,t), \tag{1}$$

where y is the unknown of the problem, t is time, and f is a generic (possibly nonlinear) function. Imagine that we know the solution y at the instant t_n, corresponding to the n-th sample of our discretization. Then, our scope is to solve (1) at the successive time step, t_{n+1}, through some mathematical/numerical strategy. Let's try to integrate both members:

$$\int_{t_n}^{t_{n+1}} \frac{\mathrm{d}y}{\mathrm{d}t}\,\mathrm{d}t = \int_{t_n}^{t_{n+1}} f(y,t)\,\mathrm{d}t. \tag{2}$$

Then, the left-hand side reads

$$\int_{t_n}^{t_{n+1}} \frac{\mathrm{d}y}{\mathrm{d}t}\,\mathrm{d}t = y(t_{n+1}) - y(t_n) \triangleq y_{n+1} - y_n,$$

where the rightmost definition is used to simplify the notation. Then, by substituting this last expression into (2), we obtain

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f(y,t)\,\mathrm{d}t. \tag{3}$$

This formula allows to *step forward in time*. If this uses only one time step from the past, then we have a *single step* method. On the other hand, a possible strategy is trying to use more than one past time step to better approximate the derivatives: such methods are referred to as *multistep methods*.

From the following section, we are going to present some classes of multistep methods.

2 Time-marching schemes

Introduction to Adams methods

The first class of (possibly) multistep methods that we are going to consider is that of *Adams methods*. Then, such methods can be further sub-classified into two sub-categories: *Adams-Bashforth* methods, where the solution in the future is evaluated only on the basis of *past time samples*, and *Adams-Moulton* methods, where the solution might involve also the future itself: as usual, we start from a hand-wavy intuition, but this is going to be much clearer when we are going to focus on the individual methods. However, the peculiarity of Adams methods is clear: starting from (3), compute the integral at the right-hand side by approximating $f(y,t)$ with another function, $f_\mathrm{a}(y,t)$, tailored in such a way to enable a simple closed-form expression of the integral. Based on this common idea, the two following sections are going to present the Adams methods sub-categories.

2.1 Adams-Bashforth methods

In this section we are going to derive the Adams-Bashforth methods for some orders. The procedure is going to be always the same: approximate $f(y,t)$ with $f_\mathrm{a}(y,t)$, the latter being a linear interpolating polynomial between different time steps. We are going to find that Adams-Bashforth methods are *explicit*, and understand what it means, going through these examples.

2.1.1 First-order Adams-Bashforth: the Euler method

The first-order Adams-Bashforth method is achieve using, as approximated $f_a(y,t)$,

$$f_a(y,t) = f(y_n, t_n). \tag{4}$$

By substituting this expression in (3), we achieve

$$y_{n+1} = y_n + (t_n - t_{n-1}) f(y_n, t_n) \triangleq h f(y_n, t_n), \tag{5}$$

where

$$h \triangleq t_n - t_{n-1} \tag{6}$$

is the time step, *i.e.,* the time between two time samples. In these notes we are going to focus on even-spaced time steps, therefore h will be the same for whatever n. The time-marching scheme (5) is the first-order Adams-Bashforth method, which is also referred to as *Euler method*.

The Euler method is explicit. Indeed, in order to evaluate a future time step, we only need to evaluate the function f in past time steps, without solving any equation: it is just a matter of plugging the t_n in f, and that's it.

2.1.2 Second-order Adams-Bashforth method

Defining the approximated function for the Euler method is pretty simple, since it involves a single time step. For the second-order Adams-Bashforth method on, we need something more complicated. The idea is to define $f_a(y,t)$ as the linear polynomial interpolating f in the time samples t_n and t_{n-1}. This can be built as a Lagrange polynomial:

$$f_a(y,t) = f(y_{n-1}, t_{n-1}) \frac{t - t_n}{t_{n-1} - t_n} + f(y_n, t_n) \frac{t - t_{n-1}}{t_n - t_{n-1}}. \tag{7}$$

The polynomial is clearly linear, since it involves only the first power of t, and it is interpolating, since, if evaluated in t_{n-1} or t_n, only the first or the second term survives, and correctly interpolates f in the corresponding time samples. By using (7) in (3), we have

$$\begin{aligned}
\int_{t_n}^{t_{n+1}} f(y,t)\,\mathrm{d}t \simeq \int_{t_n}^{t_{n+1}} f_a(y,t)\,\mathrm{d}t = \\
&= \int_{t_n}^{t_{n+1}} \left[f(y_{n-1}, t_{n-1}) \frac{t - t_n}{t_{n-1} - t_n} + f(y_n, t_n) \frac{t - t_{n-1}}{t_n - t_{n-1}} \right] \mathrm{d}t = \\
&= \int_{t_n}^{t_{n+1}} \left[f(y_{n-1}, t_{n-1}) \frac{t - t_n}{-h} + f(y_n, t_n) \frac{t - t_{n-1}}{h} \right] \mathrm{d}t = \\
&= \left[\frac{1}{2h} f(y_n, t_n)(t - t_{n-1})^2 - \frac{1}{2h} f(y_{n-1}, t_{n-1})(t - t_n)^2 \right] \Bigg|_{t_n}^{t_{n-1}},
\end{aligned}$$

where we used (6) (taking care of sign). Since we are using even-spaced time samples, $t_{n+1} - t_{n-1} = 2h$, and, therefore, continuing with the derivation, we have

$$\begin{aligned}
\int_{t_n}^{t_{n+1}} f(y,t)\,\mathrm{d}t &\simeq \frac{1}{2h} f(y_n, t_n)(2h)^2 - \frac{1}{2h} f(y_{n-1}, t_{n-1})(h)^2 - \frac{1}{2h} f(y_n, t_n) h^2 + \frac{1}{2h} f(y_{n-1}, t_{n-1}) \times 0 = \\
&= \frac{h}{2} \left[3 f(y_n, t_n) - f(y_{n-1}, t_{n-1}) \right].
\end{aligned} \tag{8}$$

By plugging (8) in (3), we can find

$$y_{n+1} = y_n + \frac{h}{2} \left[3 f(y_n, t_n) - f(y_{n-1}, t_{n-1}) \right]. \tag{9}$$

This is the second-order Adams-Bashforth scheme. It is clear that, just like the Euler method, this is an explicit method.

2.1.3 Third-order Adams-Bashforth method

The last example of Adams-Bashforth method presented in these notes is the third-order scheme, which can be achieved using, as approximated function $f_\mathrm{a}(y,t)$,

$$f_\mathrm{a}(y,t) = f(y_{n-1},t_{n-1})\frac{(t-t_n)(t-t_{n-2})}{(t_{n-1}-t_n)(t_{n-1}-t_{n-2})} + \\ + f(y_n,t_n)\frac{(t-t_{n-1})(t-t_{n-2})}{(t_n-t_{n-1})(t_n-t_{n-2})} + f(y_{n-2},t_{n-2})\frac{(t-t_{n-1})(t-t_n)}{(t_{n-2}-t_{n-1})(t_{n-2}-t_n)}, \tag{10}$$

which leads, after a few manipulations, to

$$y_{n+1} = y_n + \frac{h}{12}\left[23f(y_n,t_n) - 16f(y_{n-1},t_{n-1}) + 5f(y_{n-2},t_{n-2})\right]. \tag{11}$$

2.2 Adams-Moulton methods

In general, Adams methods have a form like

$$y_{n+1} = y_n + w_0 f(y_{n+1},y_{n+1}) + \sum_{i=1}^{n} w_i f(y_{n+1-i},t_{n+1-i}). \tag{12}$$

However, all the schemes presented in the previous section had $w_0 = 0$, *i.e.,* involved only past time samples, and the function evaluated in them. The peculiarity of Adams-Moulton method is the fact that $w_0 \neq 0$, *i.e.,* they are *implicit*. In other words, the schemes contain also $f(y_{n+1},t_{n+1})$, so they involve the evaluation of the function in the *future* time step. This is why these methods are referred to as *implicit*: in order to find y_{n+1}, we need to solve an equation. And, if f is nonlinear, then we need to solve a nonlinear equation, requiring a numerical treatment such as the fixed-point method or the Newton-Raphson method.

The derivation of these schemes is quite similar to that of Adams-Bashforth methods: we have to approximate f with f_a, which can be again an interpolating polynomial (written for example in the form of Lagrangian polynomials), and compute the integral on this basis. The difference is that, in this case, the interpolant must contain also the time sample t_{n+1}, and the function evaluated in it.

2.2.1 First-order Adams-Moulton: the implicit Euler method

The simplest Adams-Moulton method is achieved using, as approximating function,

$$f_\mathrm{a}(y,t) = f(y_{n+1},t_{n+1}). \tag{13}$$

By plugging this into (3), we have

$$\int_{t_n}^{t_{n+1}} f_\mathrm{a}(y,t)\,\mathrm{d}t = \int_{t_n}^{t_{n+1}} f(y_{n+1},t_{n+1})\,\mathrm{d}t = tf(y_{n+1},t_{n+1})\Big|_{t_n}^{t_{n+1}} = \\ = hf(y_{n+1},t_{n+1}),$$

so that the first-order Adams-Moulton method reads

$$y_{n+1} = y_n + hf(y_{n+1},t_{n+1}). \tag{14}$$

This method is broadly referred to as *backward Euler method* or *implicit Euler method*, due to the similarity with (5), but involving an implicit equation rather than an explicit one due to the presence of $f(y_{n+1},t_{n+1})$.

2.2.2 Second-order Adams-Moulton: the trapezoidal rule

If we increase the order of the Lagrange polynomial, we achieve something different: we are going to find the counterpart of (9), but implicit. So, starting from

$$f_{\mathrm{a}}(y,t) = f(y_{n+1},t_{n+1})\frac{t-t_n}{t_{n+1}-t_n} + f(y_n,t_n)\frac{t-t_{n+1}}{t_n-t_{n+1}}, \tag{15}$$

by evaluating the usual integral (3), we achieve

$$\begin{aligned}\int_{t_n}^{t_{n+1}} f_{\mathrm{a}}(y,t)\,\mathrm{d}t &= \left[f(y_{n+1},t_{n+1})\frac{1}{2h}(t-t_n)^2 - f(y_n,t_n)\frac{1}{2h}(t-t_{n+1})^2\right]\Big|_{t_n}^{t_{n+1}} = \\ &= f(y_{n+1},t_{n+1})\frac{h^2}{2h} - 0 - 0 + f(y_n,t_n)\frac{h^2}{2h} = \\ &= \frac{1}{2}f(y_{n+1},t_{n+1})h + \frac{1}{2}f(y_n,t_n)h.\end{aligned}$$

By substituting this into (3), we obtain

$$y_{n+1} = y_n + \frac{1}{2}h\left[f(y_{n+1},t_{n+1}) + f(y_n,t_n)\right], \tag{16}$$

which is commonly referred to as *trapezoidal rule*, and probably it is the most popular numerical scheme among those presented so far. Indeed, it is used in several simulators, *e.g.,* in SPICE.

2.2.3 Third-order Adams-Moulton formula

It is possible to increase further the order of the Lagrange polynomial, leading to higher-order schemes, such as the third-order Adams-Moulton formula, which reads (proof not reported):

$$y_{n+1} = y_n + \frac{1}{12}h\left[5f(y_{n+1},t_{n+1}) + 8f(y_n,t_n) - f(y_{n-1},t_{n-1})\right]. \tag{17}$$

2.3 Backward difference formulae

In the previous sections we described Adams methods, which are based on *interpolating* the function f in (1). In this section we are going to present a different way, to some extent complementary: instead of approximating the function evaluated in the various time steps, we approximate directly y by means of Lagrange polynomials. The schemes derived on the basis of this principle are referred to as *backward difference formulae*, or BDF.

2.3.1 First-order backward difference formula (BDF1)

The simplest backward difference formula is the so-called BDF1. In this case, we consider

$$y(t) \simeq y_{\mathrm{a}}(t),$$

where

$$y_{\mathrm{a}}(t) = y_{n+1}\frac{t-t_n}{t_{n+1}-t_n} + y_n\frac{t-t_{n+1}}{t_n-t_{n+1}} \tag{18}$$

$$= y_{n+1}\frac{t-t_n}{h} - y_n\frac{t-t_{n+1}}{h}. \tag{19}$$

Our scope is to re-start from (1). So, we want to solve

$$\left.\frac{\mathrm{d}y_{\mathrm{a}}}{\mathrm{d}t}\right|_{t=t_{n+1}} = f(y_{n+1},t_{n+1}), \tag{20}$$

which requires to calculate *analytically* the derivative of y_{a}, and evaluate it in t_{n+1}. Applying this procedure to (19), we achieve

$$\left.\frac{\mathrm{d}y_{\mathrm{a}}}{\mathrm{d}t}\right|_{t=t_{n+1}} = \frac{y_{n+1}}{h} - \frac{y_n}{h}, \tag{21}$$

which can be plugged in (20), leading to

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}). \tag{22}$$

It is not the first time that we obtain this formula: this is perfectly coincident with (14), *i.e.,* to with the implicit Euler method. But, we derived it following a totally different path.

2.3.2 Second-order backward difference formula (BDF2)

We are now going towards the higher-order formula: the BDF2. In this case, our y_a is going to be:

$$\begin{aligned} y_a &= y_{n+1}\frac{(t-t_n)(t-t_{n-1})}{(t_{n+1}-t_n)(t_{n+1}-t_{n-1})} + y_n\frac{(t-t_{n+1})(t-t_{n-1})}{(t_n-t_{n+1})(t_n-t_{n-1})} + y_{n-1}\frac{(t-t_{n+1})(t-t_n)}{(t_{n-1}-t_{n+1})(t_{n-1}-t_n)} = \\ &= y_{n+1}\frac{(t-t_n)(t-t_{n-1})}{2h^2} + y_n\frac{(t-t_{n+1})(t-t_{n-1})}{(-h^2)} + y_{n-1}\frac{(t-t_{n+1})(t-t_n)}{2h^2}. \end{aligned} \tag{23}$$

It is quite easy to get lost in this calculation. To avoid this possibility, it is convenient to evaluate the derivatives by the chain rule, without expanding the factors. Indeed, most of them will simplify when evaluating the result in $t = t_{n+1}$. So, we have:

$$\frac{\mathrm{d}y_a}{\mathrm{d}t} = y_{n+1}\left[\frac{t-t_{n-1}}{2h^2} + \frac{t-t_n}{2h^2}\right] + y_n\left[\frac{t-t_{n-1}}{(-h^2)} + \frac{t-t_{n+1}}{(-h^2)}\right] + y_{n-1}\left[\frac{t-t_n}{2h^2} + \frac{t-t_{n+1}}{2h^2}\right], \tag{24}$$

which, evaluated in $t = t_{n+1}$, is

$$\begin{aligned} \left.\frac{\mathrm{d}y_a}{\mathrm{d}t}\right|_{t_{n+1}} &= y_{n+1}\left[\frac{t_{n+1}-t_{n-1}}{2h^2} + \frac{t_{n+1}-t_n}{2h^2}\right] + y_n\left[\frac{t_{n+1}-t_{n-1}}{(-h^2)} + \frac{t_{n+1}-t_{n+1}}{(-h^2)}\right] + y_{n-1}\left[\frac{t_{n+1}-t_n}{2h^2} + \frac{t_{n+1}-t_{n+1}}{2h^2}\right] = \\ &= y_{n+1}\left[\frac{2h}{2h^2} + \frac{h}{2h^2}\right] + y_n\left[\frac{2h}{(-h^2)} + 0\right] + y_{n-1}\left[\frac{h}{2h^2} + 0\right] = \\ &= \frac{3}{2h}y_{n+1} - \frac{2}{h}y_n + \frac{1}{2h}y_{n-1}. \end{aligned} \tag{25}$$

This can be plugged in (20), leading ultimately to

$$\frac{3}{2h}y_{n+1} - \frac{2}{h}y_n + \frac{1}{2h}y_{n-1} = f(y_{n+1}, t_{n+1}),$$

which can be re-written as

$$y_{n+1} = \frac{2}{3}hf(y_{n+1}, t_{n+1}) + \frac{4}{3}y_n - \frac{1}{3}y_{n-1}, \tag{26}$$

which is the so-called BDF2 method.

This proof has been quite cumbersome, but it could help understanding the foundations of this method. In the literature, it is possible to find smarter ways to evaluate the BDF equation coefficients. In particular, an interesting example is provided in [3, Section 2.3].

3 Stability of time-marching schemes

3.1 Introduction to A-stability

When studying the numerical discretization of some (even static) numerical problems, in particular those involving transport, it is possible to find stability issues. Right now we are doing something quite different, *i.e.,* studying the time evolution of a dynamic system. Yet, we are going to find similar issues also for this kind of problems, and their solution is going to require a careful numerical treatment.

Most of the presentations related to this concept of *stability* are based on associating, to each time-marching scheme, a *characteristic polynomial*, usually presented as $\rho(\xi)$ and $\sigma(\eta)$, as, for example, in [3, Section 2.2], or in

[1]. This theory is pretty general, and it could allow to achieve many results for whatever method with relatively simple calculations, including also estimates of the accuracy and convergence properties. In these notes, we aim at providing a different theory proposed by Dahlquist, conceptually simpler, but providing immediate understanding and to some extent more *engineering-fashionable* intuitions.

The idea we are going to exploit is based on investigating numerically the effect of discretizing a differential equation we know very well:

$$y'(t) = \lambda y(t), \tag{27}$$

where $\lambda \in \mathbb{C}$. This problem is clearly in the form of (1), where

$$f(y,t) = \lambda y, \tag{28}$$

The general integral of (27) is

$$y(t) = C \exp(\lambda t), \tag{29}$$

where C should be determined by enforcing the initial condition, at $t = 0$. By inspecting (29), it is clear that, if $\mathbb{R}\mathrm{e}\{\lambda\} < 0$, then it approaches 0 for $t \to \infty$.

Now, the big question is: if we discretize (27) using one of the schemes presented in the previous sections, does the corresponding numerical solution satisfy this convergence property? If it does, then the numerical scheme is said to be *A-stable*. If it doesn't, then such scheme does not even guarantee to reproduce the trend (hence, much less than *an accurate result*) of the *real* solution.

Clearly, (29) is a monotonic decreasing function. In this view, we are going to assess the stability properties of the methods under study by verifying whether, *a posteriori* of time discretization, the equation is still monotonically decreasing. To this aim, the idea is to define

$$z = \lambda t, \tag{30}$$

and to define an *amplification factor* $A(z)$ as

$$A(z) = \left| \frac{y_{n+1}}{y_n} \right|. \tag{31}$$

If $A(z) < 1$, then at the $(n+1)$-th iteration we are sure that the absolute value of our solution, y_{n+1} will be lower than the previous one. A-stability is verified by checking if the amplification factor is lower than 1 over the whole left complex hemi-space (corresponding to having λ with negative real part).

Before starting, in the following subsections, to study A-stability for specific methods, it is fair to remark that this idea applies to some of the schemes studied in the previous section, but its application range is not total. In this view, the formulation based on characteristic polynomials, ρ and σ, would be more general [4, 3, 1].

3.2 Stability properties of the Euler method

In Section 2 we have studied a number of methods, some explicit, some implicit. So far, there is no apparent reason to choose implicit methods. Indeed, in the very common case of nonlinear differential equations, they also require additional treatments such as the Newton-Raphson algorithm, leading to more complicated implementations and less computationally-efficient schemes due to the large number of calls of the function f to solve the implicit equation. The scope of this section is to unveil the importance of implicit schemes, demonstrating why explicit schemes are inapplicable to practical problems.

Let's start with the simplest numerical scheme presented so far: the Euler method. From (5), it reads

$$y_{n+1} = y_n + h f(y_n, t_n).$$

By recalling (28), the Euler scheme reads

$$y_{n+1} = y_n + h y_n = (1 + h\lambda) y_n = (1 + z) y_n,$$

where we are using (30). The last expression can be manipulated to achieve easily the amplification factor as

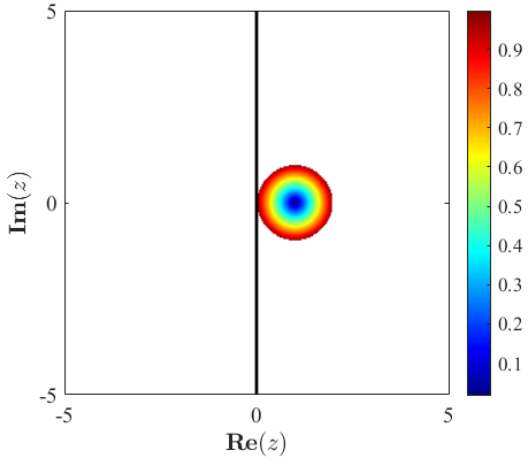

Figure 1: Amplification factor of the Euler method. The map reports only the values for which $A(z) < 1$, where the scheme is stable. White regions are indicative of instability.

$$A(z) = \left| \frac{y_{n+1}}{y_n} \right| = |1 + z| \,. \tag{32}$$

The Euler method leads to a monotonically-decreasing solution for $A(z) < 1$, *i.e.*, for the circle in the complex plane with center in $z = 1$ and radius 1. This is demonstrated in Fig. 1, which plots $A(z)$ for the Euler method in the complex plane. This plot has been performed with the following script.

Amplification factor for Euler's method (Main_Stability_Euler.m)

```
clear
close all
clc

ReZ = linspace(-5,5,301); % explored real parts of z = h*lambda
ImZ = linspace(-5,5,302); % explored imaginary parts of z = h*lambda

[REZ,IMZ] = meshgrid(ImZ,ReZ);

Z = IMZ+1i*REZ; % this is a 2D map of z = h*lambda

A = abs(1-Z); % amplification factor for Euler method

% Setting to NaN (plotted in white) the instability conditions
A(A>=1) = NaN;

figure
hold on
box on
surf(IMZ,REZ,A)
shading interp
view(2)
axis equal
axis([-5,5,-5,5])
colormap(jet)
grid off
xlabel('$\mathbf{Re}(z)$','Interpreter','LaTeX')
ylabel('$\mathbf{Im}(z)$','Interpreter','LaTeX')
plot3([0,0],[-5,5],[2,2],'k','LineWidth',2)
colorbar
set(gca,'FontName','Times New Roman','FontSize',14)
```

Here, all the values of $A(z)$ greater or equal than 1 have been set to NaN. In this view, the white regions in Fig. 1 indicate instability. The weakness of the Euler method is evident: it is stable only for a very limited range of parameters, therefore it is not reliable at all to perform time discretizations: it is very far from being A-stable.

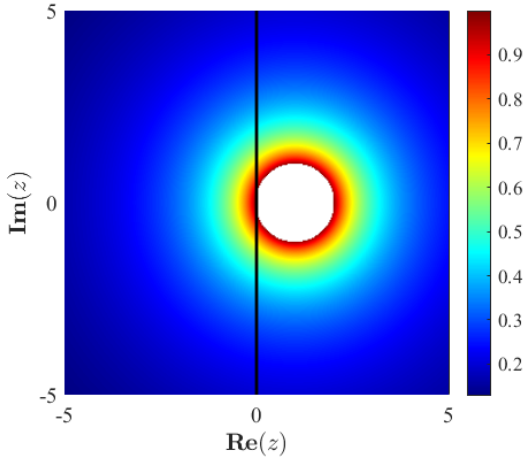

Figure 2: Amplification factor of the implicit Euler method. The map reports only the values for which $A(z) < 1$, where the scheme is stable. White regions are indicative of instability.

3.3 Stability properties of the implicit Euler method

The implicit Euler method (14) reads

$$y_{n+1} = y_n + hf(y_{n+1}, t_{n+1}).$$

By substituting (28), we have

$$f(y_{n+1}, t_{n+1}) = \lambda y_{n+1},$$

so, the method reads

$$y_{n+1} = y_n + h\lambda y_{n+1},$$

which becomes, after re-organizing everything isolating y_{n+1} and recalling (30),

$$y_{n+1} = \frac{1}{1-z} y_n,$$

and, ultimately, leading to the amplification factor

$$A(z) = \left| \frac{1}{1-z} \right|. \tag{33}$$

The stability condition for this method is, as usual, the region $A(z) < 1$, which, inverting, holds for

$$|1 - z| \geq 1.$$

In this case, as also demonstrated by Fig. 2, the stability region is out of a circle of radius 1 centered in $z = 1$. In this view, such region includes also the entire half-plane $\mathbb{R}\mathrm{e}\{z\} < 0$, and, ultimately, we can state that this method is A-stable!

3.4 Stability of the trapezoidal rule

The trapezoidal rule, from (16), reads

$$y_{n+1} = y_n + \frac{1}{2} h \left[f(y_{n+1}, t_{n+1}) + f(y_n, t_n) \right].$$

If, as usual, we substitute (28), we achieve

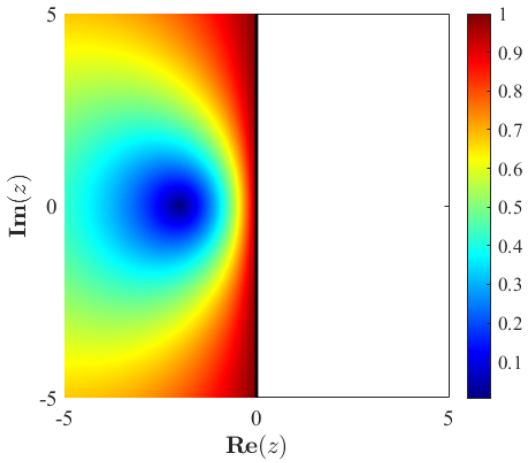

Figure 3: Amplification factor of the trapezoidal rule. The map reports only the values for which $A(z) < 1$, where the scheme is stable. White regions are indicative of instability.

$$y_{n+1}\left(1 - \frac{h\lambda}{2}\right) = y_n\left(1 + \frac{h\lambda}{2}\right),$$

and, ultimately,

$$y_{n+1} = \frac{1 + \dfrac{h\lambda}{2}}{1 - \dfrac{h\lambda}{2}} y_n,$$

and, after substituting as usual (30), the amplification factor reads

$$A(z) = \left|\frac{1 + \dfrac{h\lambda}{2}}{1 - \dfrac{h\lambda}{2}}\right| = \left|\frac{2 + z}{2 - z}\right|. \tag{34}$$

In this case, as demonstrated by Fig. 3, the stability region is exactly $\mathbb{R}\mathrm{e}\{z\} < 0$, *i.e.,* exactly as for the analytic equation. So, the trapezoidal rule is A-stable.

3.5 Concluding remarks on A-stability

As it has been anticipated in the introduction of this section, using this *simplified formalism* does not allow to prove the stability results for whatever method. We have demonstrated that the (forward) Euler method is unstable. Actually, this is quite a general property, since also higher order Adams-Bashforth (hence, explicit) methods are unstable. Instead, implicit methods, despite the more complicated implementation steps required, have the advantage to be A-stable. It could be proved that also BDF2 is stable, but, instead, BDF3 or superior order methods are not (risking also to be not convergent). Among the methods studied so far, the most stable one is the trapezoidal rule.

Before moving on with the next topic, it is worth to mention a class of methods not discussed in these slides, which is that of Runge-Kutta methods. In general, Runge-Kutta methods are less stable than the methods presented here, even if some alternative formulations allow to improve their behaviour (see, *e.g.,* [1, p. 31, footnote reference]. However, the methods introduced in these notes should be more than sufficient to study a very broad range of problems, even without resorting to tricks or modified methods.

3.6 An example of stiff problem

In the previous section, we have presented a first introduction to the stability of time-marching algorithms. We have studied the idea of A-stability, and found that some of the investigated methods, *e.g.,* the trapezoidal rule, are A-stable. Yet, we are going to find that the discussion is far from being exhaustive, if we have to deal with *stiff problems*.

The best way to understand stiff problems, is to try to address one of them. In particular, let's consider the initial value problem

$$\begin{cases} y'' + 100y' + 99y = 0 \\ y'(0) = -100 \\ y(0) = 2, \end{cases} \tag{35}$$

which has, as closed-form solution,

$$y(t) = \exp(-t) + \exp(-99t). \tag{36}$$

In general, every time we have to perform a time discretization, we have to apply it to a first-order differential equation: we don't know how to formulate a time discretization for, *e.g.,* a second-order equation such as the one of (35). Instead, there is no problem at all when studying *vectorial* problems, *i.e.,* where we have systems of equations rather than a single equation. In this view, the first step is to transform higher-degree problems into a problem involving only first derivatives, even though vectorial. So, if we define

$$w = y', \tag{37}$$

the system (35) can be re-written, equivalently, as

$$\begin{cases} y' = w \\ w' = -99y - 100w \\ w(0) = -100 \\ y(0) = 2, \end{cases} \tag{38}$$

where we have also isolated the time derivatives at the left-hand side.

If we define the state vector $\underline{y}$ as

$$\underline{y} = \begin{bmatrix} y \\ w \end{bmatrix}, \tag{39}$$

out system of equation (38) can be written in matrix form as

$$\underbrace{\begin{bmatrix} y \\ w \end{bmatrix}'}_{\underline{y}'} = \underbrace{\begin{bmatrix} 0 & 1 \\ -99 & -100 \end{bmatrix}}_{\underline{\underline{A}}} \underbrace{\begin{bmatrix} y \\ w \end{bmatrix}}_{\underline{y}}, \tag{40}$$

where the initial condition is

$$\underline{y}_0 = \begin{bmatrix} 2 \\ -100 \end{bmatrix}. \tag{41}$$

So: in some way, we managed to write an equation having the form

$$\underline{y}' = \underline{f}(\underline{y}, t) = \underline{\underline{A}}\,\underline{y},$$

which is exactly in the same form of (1). Lucky us, in this case $\underline{f}$ (which is vectorial) is simply $\underline{\underline{A}}\,\underline{y}$, because (38) is a system of linear differential equations: this will simplify the numerical treatment of this problem compared to more general cases.

In the following sections we are going to attack this problem with different numerical schemes, aiming to draw some additional conclusions with respect to our previous studies.

3.6.1 Stiff problem example: solution by the trapezoidal rule

Let's try to solve (35) by the trapezoidal rule. This can be written recalling (16) but substituting y_n with $\underline{y}_n$, and $f(y_n, t_n)$ with $\underline{\underline{A}}\,\underline{y}_n$, leading to

$$\underline{y}_{n+1} = \underline{y}_n + \frac{h}{2}\left[\underline{\underline{A}}\,\underline{y}_{n+1} + \underline{\underline{A}}\,\underline{y}_n\right].$$

By applying the identity $\underline{\underline{I}}\,\underline{y}_{n+1} = \underline{y}_{n+1}$, $\underline{\underline{I}}$ being the identity matrix of opportune dimensions (in this case, 2×2), we can re-write the last equation as

$$\left(\underline{\underline{I}} - \frac{h}{2}\underline{\underline{A}}\right)\underline{y}_{n+1} = \left(\underline{\underline{I}} + \frac{h}{2}\underline{\underline{A}}\right)\underline{y}_n,$$

which, ultimately, can be inverted to achieve

$$\underline{y}_{n+1} = \left(\underline{\underline{I}} - \frac{h}{2}\underline{\underline{A}}\right)^{-1}\left(\underline{\underline{I}} + \frac{h}{2}\underline{\underline{A}}\right)\underline{y}_n. \tag{42}$$

It can be appreciated how this expression represents the extension, to the vectorial case, of (34). This algorithm has been implemented in the following MATLAB® listing:

Implementation of trapezoidal rule for problem (40) (Main_StiffProblem_TR.m)

```
clear
close all
clc

% time step of the solver
% h = 0.4;
h = 0.004;

% time axis
tvet = 0:h:12;

% analytic solution for reference
yanal = exp(-tvet) + exp(-99*tvet);

% setting yvet to the initial condition
yvet = [2;-100];

% matrix describing the system of ordinary differential equations
Amat = [   0,     1;
         -99,  -100];

% identity matrix
Imat = eye(2);

% initializing vector of numerical solution
ynum(1,1) = yvet(1);
ynum(2,1) = yvet(2);

for indTime = 2:length(tvet)
    yvet = (Imat-h/2*Amat)\((Imat+h/2*Amat)*yvet); % trapezoidal rule
    ynum(:,indTime) = yvet;                        % saving result
end

figure
grid on
hold on
box on
plot(tvet,yanal,'b',tvet,ynum(1,:),'r--','LineWidth',1.5)
xlabel('time t')
ylabel('Output y(t)')
legend('Analytic','Trapezoidal rule')
set(gca,'FontSize',14,'FontName','Times New Roman')
```

Launching this code using as time step $h = 0.4$, it is possible to obtain the results reported in Fig. 4. It appears at a glance that the trapezoidal rule tends asymptotically to the analytic solution, but with this strange *sawtooth-like* trend, which is totally unrepresentative of the problem.

We have learned that the trapezoidal rule is A-stable. But then, what is the problem? The problem is that A-stability is not the strongest criterion, and in particular it is not strong enough for stiff problems. Inspecting the analytic solution (36), we can write it as

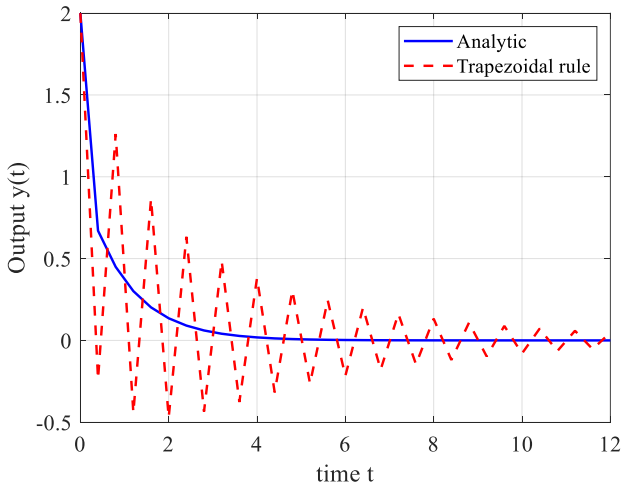

Figure 4: Solution of the problem (40) with $h = 0.4$. The solid blue and dashed red curves are obtained plotting the analytic solution, and the numerical solution achieved with the trapezoidal rule.

$$y(t) = \exp\left(-\frac{t}{\tau_1}\right) + \exp\left(-\frac{t}{\tau_2}\right),$$

where $\tau_1 = 1$, and $\tau_2 = 1/99$. This problem is stiff since it involves two very different time scales, and the numerical method must be able to *follow both of them*. An obvious solution could be using a finer time discretization. As an example, Fig. 5 reports results obtained with the very same code used to achieve (4), but using $h = 0.004$. It appears at a glance that the agreement between numerical and analytic solutions is perfect. The figure contains also an inset, which reports a magnification of the curve for initial times: there is a very steep decrease of the solution, which changes pace at about $t = 0.01$. Although the context is different, this behavior resembles the *boundary layer* that can occur in the space discretization of advection-diffusion equations: the time constant τ_2 causes a very sharp decrease in a very short time (the boundary layer), but then, later, only τ_1 is significant, since the initial transient is exhausted.

3.6.2 Stiff problem example: solution by the implicit Euler method

Before presenting criteria alternative to A-stability, let's attempt to attack (40) with another scheme: in place of the trapezoidal rule, let's try to use the implicit Euler method. Just as we did for the trapezoidal rule, the first step is to recall (14) and to write it in vector form, leading to

$$\underline{y}_{n+1} = \underline{y}_n + h\underline{\underline{A}}\,\underline{y}_{n+1},$$

which becomes, after a minor re-organization,

$$\left(\underline{\underline{I}} - h\underline{\underline{A}}\right)\underline{y}_{n+1} = \underline{y}_n,$$

and, finally,

$$\underline{y}_{n+1} = \left(\underline{\underline{I}} - h\underline{\underline{A}}\right)^{-1}\underline{y}_n. \tag{43}$$

As for the vectorial trapezoidal rule, it can be appreciated that (43) has a remarkable similarity with the amplification factor (33). Then, it can be implemented in MATLAB, which basically requires to replace, with respect to the previous code, `yvet` with

```
yvet = (Imat-h*Amat)\yvet; % BDF1
```

If we try to execute such code with $h = 0.4$, the results of Fig. 6 are obtained. Clearly, apart from some inaccuracy in $t \in [1, 5]$, the result is much better than the one obtained with the trapezoidal rule with the same step size.

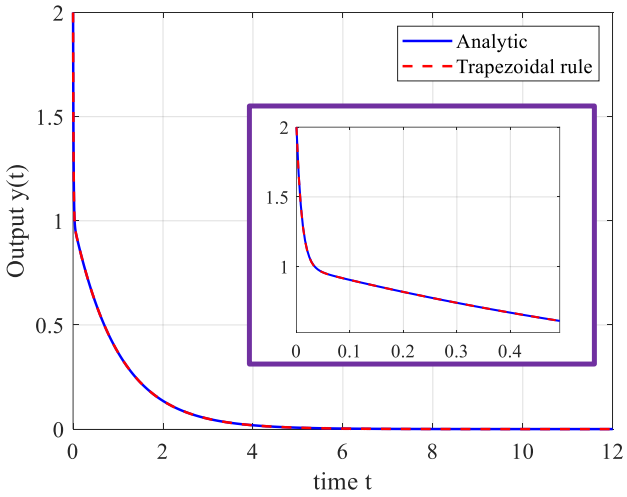

Figure 5: Solution of the problem (40) with $h = 0.004$. The solid blue and dashed red curves are obtained plotting the analytic solution, and the numerical solution achieved with the trapezoidal rule. The inset presents a magnification for small t.

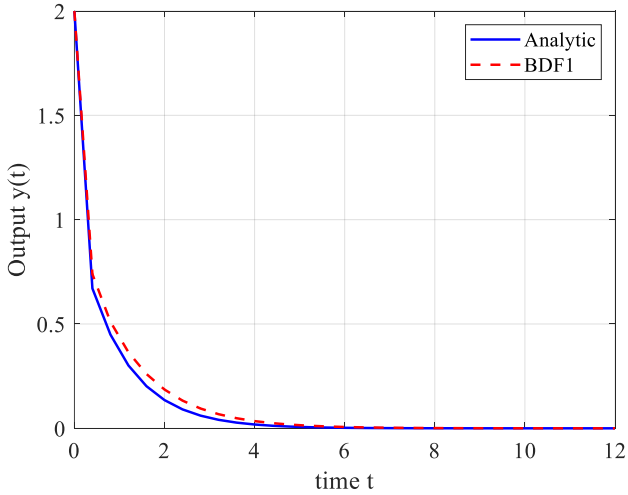

Figure 6: Solution of the problem (40) with $h = 0.4$. The solid blue and dashed red curves are obtained plotting the analytic solution, and the numerical solution achieved with the implicit Euler method (also referred to as the first order backward difference formula, BDF1).

3.7 TR vs BDF1: L-stability

Both the trapezoidal rule and the implicit Euler method are A-stable. Yet, this example unveiled that the two methods must have different stability properties. The difference between these methods is L-stability [1, 5]. L-stability requires, in addition to A-stability, that

$$\lim_{z\to-\infty} A(z) = 0. \tag{44}$$

In this view, it is clear that

- the trapezoidal rule is A-stable, but not L-stable; indeed, from (34),

$$\lim_{z\to-\infty} A(z) = \lim_{z\to-\infty} \left|\frac{2-z}{2+z}\right| = 1 \neq 0;$$

- the implicit Euler method is A-stable and L-stable, since, from (33),

$$\lim_{z\to-\infty} A(z) = \lim_{z\to-\infty} \left|\frac{1}{1-z}\right| = 0.$$

This property is particularly important when studying stiff problems.

4 Composite time-marching schemes: the TR-BDF2

We have learned that, if we want to solve stiff problems, we could use backward difference formulae, and, in particular, BDF1: it works, it might produce some inaccuracy, but certainly it is not going to provide totally wrong results such as those of Fig. 4. Now the question is: can we do something better than BDF1?

Well: something potentially better than BDF1 could be BDF2: being a 2-step method, it is expected to describe better the memory of the ordinary differential equation. If we recall the BDF2 scheme from (26), we have

$$y_{n+1} = \frac{2}{3}hf(y_{n+1}, t_{n+1}) + \frac{4}{3}y_n - \frac{1}{3}y_{n-1},$$

i.e., it clearly involves also y_{n-1}. So, bootstrapping BDF2 is not straightforward: at least for the first step we need to involve another method. And this leads us to the question: which is the best method? The first idea we could have is to use the closest scheme to BDF2, so, again, BDF1. However, this is not the best choice. Recall the disagreement between numerical and analytic solutions: it is known that this is related to *undesirable conservation properties* of backward difference schemes [6]. In this view, targeting at a composite method, the *bootstrapping scheme* should be chosen in such a way to try to counterbalance this issue. In this view, the best alternative to BDF1 as bootstrapping scheme is the trapezoidal rule! Indeed, as discussed in [5, 6], the numerical scheme based on bootstrapping with trapezoidal rule and continuing with BDF2 combines the robustness and stability of BDF2 with the energy-conserving properties of trapezoidal rule, leading to a potentially optimal integration scheme: the TR-BDF2. To be clear: the TR-BDF2 is not simply a technique where, from the $n = 0$ step, you launch TR just once, and then you only run BDF2: this would basically be a BDF2 with a first TR step, not a *composite scheme*. Moreover, it will be shown that TR-BDF2 works *only* with t_n and t_{n+1}, but not with t_{n-1}. This is not totally honest: actually, what we are going to do, is to *introduce something between* t_n and t_{n+1}.

The idea is to define a parameter $\gamma \in (0, 1)$ (to be better specified later), and to consider a *third instant*, $t_{n+\gamma}$, such that

$$t_{n+\gamma} - t_n = \gamma h, \tag{45}$$

which, obviously, means that

$$t_{n+1} - t_{n+\gamma}(1-\gamma)h, \tag{46}$$

indeed,

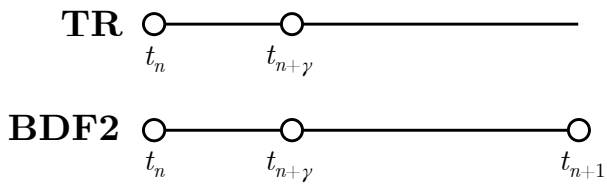

Figure 7: Pictorial view of the TR-BDF2 step: on top the TR substep evaluating the evolution from t_n to $t_{n+\gamma}$ is shown, and at bottom the BDF2 substep, involving t_n and $t_{n+\gamma}$ to evaluate t_{n+1}.

$$(1-\gamma)h + \gamma h = h.$$

Having this $t_{n+\gamma}$, the idea behind TR-BDF2 is summarized in Fig. 7, which reports a pictorial view of a TR-BDF2 step. First, the TR substep is used to evolve from t_n to $t_{n+\gamma}$. Then, the BDF2 substep, starting from t_n and $t_{n+\gamma}$, is used to evaluate t_{n+1}.

4.1 Derivation of the TR-BDF2 scheme

Unfortunately, we cannot recycle the TR and the BDF2 formulas exactly as we derived them, because they were not meant to use the *intermediate* step. Actually, for TR, this is pretty simple: because we have, instead of a step h, a step γh, and instead of y_{n+1}, $y_{n+\gamma}$. These replacements into (16) are sufficient, which lead to

$$y_{n+\gamma} - \gamma\frac{h}{2}f(y_{n+\gamma}, t_{n+\gamma}) - y_n - \gamma\frac{h}{2}f(y_n, t_n) = 0. \tag{47}$$

For the BDF2 formula, it is not this simple: we have to re-derive the whole scheme from scratch. Fortunately, we did it once, and now it is just a matter of using the correct time samples. Since BDF schemes are based on

$$\frac{\mathrm{d}y_\mathrm{a}}{\mathrm{d}t} = f(y_{n+1}, t_{n+1}),$$

where y_a is an approximation of y achieved through interpolating (Lagrange) polynomials, now, we have to write

$$y_\mathrm{a} = y_{n+1}\frac{(t-t_{n+\gamma})(t-t_n)}{(t_{n+1}-t_{n+\gamma})(t_{n+1}-t_n)} + y_{n+\gamma}\frac{(t-t_{n+1})(t-t_n)}{(t_{n+\gamma}-t_{n+1})(t_{n+\gamma}-t_n)} + y_n\frac{(t-t_{n+1})(t-t_{n+\gamma})}{(t_n-t_{n+1})(t_n-t_{n+\gamma})}, \tag{48}$$

which becomes, after substituting (45) and (46),

$$y_\mathrm{a} = y_{n+1}\frac{(t-t_{n+\gamma})(t-t_n)}{h^2(1-\gamma)} + y_{n+\gamma}\frac{(t-t_{n+1})(t-t_n)}{-h(1-\gamma)(h\gamma)} + y_n\frac{(t-t_{n+1})(t-t_{n+\gamma})}{(-h)(-h\gamma)}. \tag{49}$$

Now, we need the time derivative of (49). By applying the chain rule directly to it, we achieve

$$\frac{\mathrm{d}y_\mathrm{a}}{\mathrm{d}t} = y_{n+1}\left[\frac{t-t_n}{h^2(1-\gamma)} + \frac{t-t_{n+\gamma}}{h^2(1-\gamma)}\right] + y_{n+\gamma}\left[\frac{t-t_n}{-h^2\gamma(1-\gamma)} + \frac{t-t_{n+1}}{-h^2\gamma(1-\gamma)}\right] + y_n\left[\frac{t-t_{n+\gamma}}{\gamma h^2} + \frac{t-t_{n+1}}{\gamma h^2}\right]. \tag{50}$$

Then, we need to evaluate this expression at $t = t_{n+1}$, substituting (45) and (46) when necessary, obtaining

$$\begin{aligned}\left.\frac{\mathrm{d}y_\mathrm{a}}{\mathrm{d}t}\right|_{t_{n+1}} &= y_{n+1}\left[\frac{1+1-\gamma}{h(1-\gamma)}\right] + y_{n+\gamma}\left[\frac{-1}{\gamma(1-\gamma)h}\right] + y_n\left[\frac{1-\gamma}{\gamma h}\right] = \\ &= \frac{2-\gamma}{h(1-\gamma)}y_{n+1} - \frac{1}{h\gamma(1-\gamma)}y_{n+\gamma} + \frac{1-\gamma}{\gamma h}y_n = f(y_{n+1}, t_{n+1}).\end{aligned}$$

And, this is it: after re-arranging this last expression, we achieve the BDF2 step of the TR-BDF2 algorithm:

$$y_{n+1} = y_{n+\gamma}\frac{1}{\gamma(2-\gamma)} - y_n\frac{(1-\gamma)^2}{\gamma(2-\gamma)} + f(y_{n+1}, t_{n+1})\frac{h(1-\gamma)}{2-\gamma}. \tag{51}$$

4.2 Stability of the TR-BDF2 scheme

Now that the TR-BDF2 scheme formulation is clear, we are ready to study its stability properties. The idea is the same used in Section 3: first we study the A-stability of the method, and then, if satisfied, check for L-stability.

For the trapezoidal step, we should follow the same procedure used to derive (34), but recalling that instead of a step h we have a step $h\gamma$; so, we have:

$$y_{n+\gamma} = y_n \frac{2-\gamma z}{2+\gamma z}. \tag{52}$$

For the BDF2 part, let's re-write (51) by multiplying everything times the denominators, and isolating the $n+1$ part from the others:

$$\gamma\left[(2-\gamma)y_{n+1} - h(1-\gamma)f(y_{n+1}, t_{n+1})\right] = -(1-\gamma)^2 y_n + y_{n+\gamma}.$$

Now, we can plug $f(y_{n+1}, t_{n+1}) = \lambda y_{n+1}$ and achieve

$$\gamma\left[(2-\gamma) - z(1-\gamma)\right] y_{n+1} = -(1-\gamma)^2 y_n + y_{n+\gamma}.$$

Now, we can plug (52) in this, leading to

$$\underbrace{\gamma\left[(2-\gamma) - z(1-\gamma)\right](2-z\gamma)}_{D(z)} y_{n+1} = \underbrace{\left[-(1-\gamma)^2(2-z\gamma) + 2 + z\gamma\right]}_{N(z)} y_n.$$

Now, we can write the expressions $D(z)$ and $N(z)$ in a handier way. Starting from $D(z)$, we have

$$\begin{aligned}
D(z) &= \gamma(2-\gamma-z+z\gamma)(2-z\gamma) = \\
&= \gamma(4-2\gamma-2z+2z\gamma-2z\gamma+z\gamma^2+z^2\gamma-z^2\gamma^2) = \\
&= \gamma(2(2-\gamma)+z(-2+\gamma^2)+z^2(\gamma-\gamma^2)) = \\
&= \gamma(z^2(1-\gamma)\gamma+z(-2+\gamma^2)+2(2-\gamma)) = \\
&= -\gamma(z^2(\gamma-1)\gamma+z(2-\gamma^2)+2(\gamma-2)).
\end{aligned}$$

Then, for what concerns $N(z)$, we have:

$$\begin{aligned}
N(z) &= (z\gamma-2)(1-\gamma^2)+2+z\gamma = \\
&= (z\gamma-2)(\gamma^2-2\gamma+1)+2+z\gamma = \\
&= z\gamma^3-2\gamma^2 z+z\gamma-2\gamma^2+4\gamma-2+2+z\gamma = \\
&= z\gamma(\gamma^2-2\gamma+2)-2\gamma^2+4\gamma = \\
&= -\gamma\left[2(\gamma-2)-z(\gamma-1)^2+1)\right].
\end{aligned}$$

where, noticing that $\gamma^2-2\gamma+2 = (1-\gamma)^2+1$, we get

$$N(z) = \gamma\left[-z(1+(1-\gamma)^2)+2(2-\gamma)\right].$$

Ultimately, we have[1]

$$A(z) = \left|\frac{y_{n+1}}{y_n}\right| = \left|\frac{N(z)}{D(z)}\right| = \left|\frac{2(\gamma-2)-z(\gamma-1)^2+1)}{z^2(\gamma-1)\gamma+z(2-\gamma^2)+2(\gamma-2)}\right|. \tag{53}$$

Figure 8 shows, for different values of the parameter γ, the amplification factor (53) of the TR-BDF2 scheme. It appears at a glance that, for $\mathbb{R}\mathrm{e}\{z\} < 0$, the amplification factor is always lower than 1, demonstrating that

[1] Please mind that in the result from [1, eq. (3.13)] there is a mistake: this one should be correct...

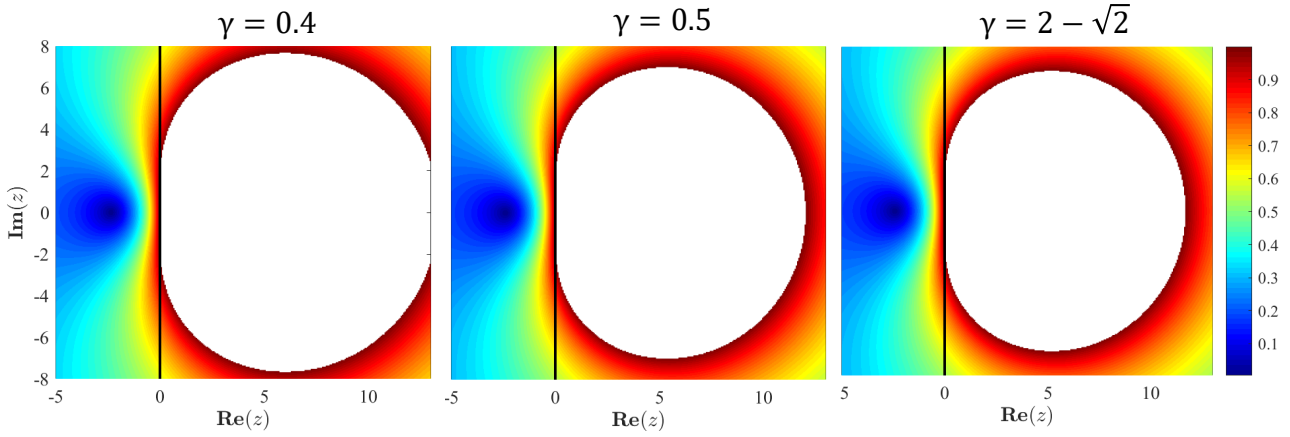

Figure 8: Amplification factor of the TR-BDF2 scheme, evaluated for three possible values of γ.

the scheme is A-stable. This is also true for $\mathbb{R}\mathrm{e}\{z\} > 0$, apart from a region whose extension depends on γ. Finally, it can be seen that

$$\lim_{z\to-\infty} A(z) = 0,$$

demonstrating that the method is also L-stable, hence suitable for stiff problems.

4.3 Solution of a linear ordinary differential equation by the TR-BDF2

The scope of this section is to formulate the TR-BDF2 for the problem (40), in order to compare its performance versus the other methods. Firstly, we have to formulate the TR step, which reads, as usual,

$$\underline{y}_{n+\gamma} = \left(\underline{\underline{I}} - \gamma\frac{h}{2}\underline{\underline{A}}\right)^{-1}\left(\underline{\underline{I}} + \gamma\frac{h}{2}\underline{\underline{A}}\right)\underline{y}_n, \tag{54}$$

which is just like (42), using γh in place of h.

The second step, based on BDF2, can be formulated starting from (51), and reads

$$\underline{y}_{n+1} = \frac{1}{\gamma(2-\gamma)}\underline{y}_{n+\gamma} - \frac{(1-\gamma)^2}{\gamma(2-\gamma)}\underline{y}_n + \frac{h(1-\gamma)}{2-\gamma}\underline{\underline{A}}\underline{y}_{n+1}.$$

By re-arranging this equation, we achieve:

$$\left[\underline{\underline{I}} - h\frac{1-\gamma}{2-\gamma}\underline{\underline{A}}\right]\underline{y}_{n+1} = \frac{1}{\gamma(2-\gamma)}\underline{y}_{n+\gamma} - \frac{(1-\gamma)^2}{\gamma(2-\gamma)}\underline{y}_n,$$

and, ultimately,

$$\underline{y}_{n+1} = \left[\underline{\underline{I}} - h\frac{1-\gamma}{2-\gamma}\underline{\underline{A}}\right]^{-1}\left[\frac{1}{\gamma(2-\gamma)}\underline{y}_{n+\gamma} - \frac{(1-\gamma)^2}{\gamma(2-\gamma)}\underline{y}_n\right] \tag{55}$$

This scheme has been implemented, and its results are compared, in Fig. 9, with the analytic solution, using $h = 0.4$ (the *coarse* time discretization). Apart from a kink at the end of the boundary layer, this method provides very accurate results compared to all the other methods.

5 Nonlinear differential equations

In this section we are not going to add anything about time discretization, but we are going to present examples of solution of nonlinear dynamic problems. We are going to focus on a reference problem: the elastic pendulum. Its governing equations are achieved through analytic mechanics [1], and they are not derived here:

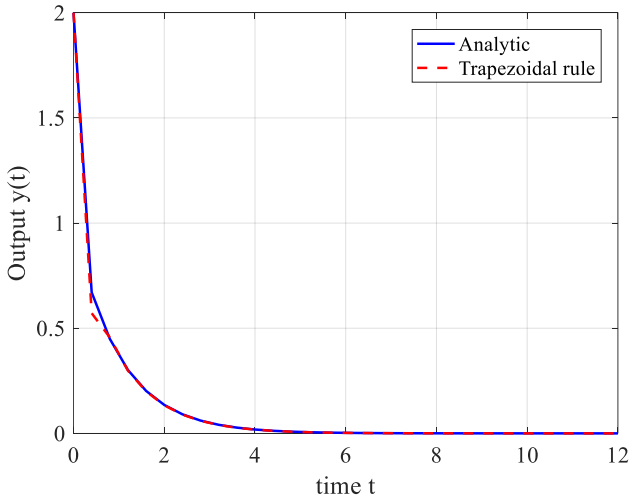

Figure 9: Solution of the problem (40) with $h = 0.4$. The solid blue and dashed red curves are obtained plotting the analytic solution, and the numerical solution achieved with the TR-BDF2 scheme using $\gamma = 2 - \sqrt{2}$.

$$\begin{cases} \ddot{r} = \mathrm{g}\cos\vartheta - \dfrac{k}{m}(r - L) + r\dot{\vartheta}^2 \\ \ddot{\vartheta} = \dfrac{2\dot{r}\dot{\vartheta} + \mathrm{g}\sin\vartheta}{-r}, \end{cases} \tag{56}$$

where r and ϑ are the two unknowns of the problem, indicating the length and the angle of the pendulum, m is the mass of the ball connected to the spring having elastic constant k, g is the gravitational acceleration ($9.81\,\mathrm{m/s^2}$), and L is the rest length. This problem contains all the possible criticalities we could think about. Indeed, it is:

- vectorial: it has two unknowns;
- dynamic: it involves time derivatives, which here are indicated with the *dots*, according to the usual notation of analytic mechanics,

$$\dot{r} = \frac{\mathrm{d}r}{\mathrm{d}t}$$
$$\dot{\vartheta} = \frac{\mathrm{d}\vartheta}{\mathrm{d}t};$$

- nonlinear: it involves products of the unknowns, of their time derivatives, and nonlinear functions;
- of second order: it involves second derivatives.

Let's start from the end: the problem should be transformed into a larger problem of first-order differential equations, introducing the supplementary unknowns

$$v \triangleq \frac{\mathrm{d}r}{\mathrm{d}t} \tag{57}$$
$$\omega = \frac{\mathrm{d}\vartheta}{\mathrm{d}t}. \tag{58}$$

By plugging these two new variables in (56), we can re-write the system as

$$\begin{cases} \dfrac{\mathrm{d}\vartheta}{\mathrm{d}t} = \omega \\ \dfrac{\mathrm{d}\omega}{\mathrm{d}t} = \dfrac{2v\omega + \mathrm{g}\sin\vartheta}{-r} \\ \dfrac{\mathrm{d}r}{\mathrm{d}t} = v \\ \dfrac{\mathrm{d}v}{\mathrm{d}t} = \mathrm{g}\cos\vartheta - \dfrac{k}{m}(r - L) + r\omega^2, \end{cases} \tag{59}$$

which can be re-written compactly, having defined the state vector $\underline{y}$

$$\underline{y} = \begin{bmatrix} \vartheta \\ \omega \\ r \\ v \end{bmatrix}, \tag{60}$$

and the function $\underline{f}$

$$\underline{f} = \begin{bmatrix} \omega \\ \dfrac{2v\omega + \mathrm{g}\sin\vartheta}{-r} \\ v \\ \mathrm{g}\cos\vartheta - \dfrac{k}{m}(r - L) + r\omega^2 \end{bmatrix}, \tag{61}$$

as

$$\frac{\mathrm{d}}{\mathrm{d}t}\underline{y} = \underline{f}(\underline{y}, t). \tag{62}$$

5.1 Solution by the trapezoidal rule and notation setting

Equation (62) is written in the usual way, which lends itself to time discretization. In this view, let's try to perform a discretization by the trapezoidal rule. This reads, for this vectorial case, just like (16), but in vector form:

$$\underline{y}_{n+1} = \underline{y}_n + \frac{h}{2}\left[\underline{f}(\underline{y}_n, t_n) + \underline{f}(\underline{y}_{n+1}, t_{n+1})\right]. \tag{63}$$

This equation can be re-written, bringing all the terms to the left-hand side, as

$$\underline{y}_{n+1} - \frac{h}{2}\underline{f}(\underline{y}_{n+1}, t_{n+1}) - \underline{y}_n - \frac{h}{2}\underline{f}(\underline{y}_n, t_n) = 0. \tag{64}$$

In this view, it is clear that our problem has the form

$$\underline{F}(\underline{y}_{n+1}) = 0,$$

where, however, this time, $\underline{f}$ cannot be written as $\underline{\underline{A}}\,\underline{y}$, otherwise it would be linear. In this view, aiming to find $\underline{y}_{n+1}$ solving (64), we need to use Newton's method. Newton's method. It consists of the following steps:

1. Start from a guess for $\underline{y}_{n+1}$;
2. Assemble the vector function $\underline{F}$ on the basis of the present approximation of the solution vector $\underline{y}_{n+1}$ (to be further refined if necessary), and its Jacobian, $\underline{\underline{J}}$ with respect to the unknowns, *i.e.,* the components of the vector $\underline{y}_{n+1}$;
3. Find the solution of the system

$$\underline{\underline{J}}\Delta\underline{y}^{(k+1)} = -\underline{F},$$

where n indicates the time step, and k the Newton iteration

4. Update the solution

$$\underline{y}_{n+1}^{(k+1)} = \underline{y}_{n+1}^{(k)} + \Delta\underline{y}^{(k+1)}$$

5. If the nonlinear problem is not solved, *i.e.,* if $\underline{F}$ is not zero or if $\underline{y}_{n+1}^{(k+1)}$ keeps changing significantly from the previous iteration, go back to (2).

Trying to fix the concepts, for this first time, we are going to write $\underline{F}$ and $\underline{\underline{J}}$ explicitly. Starting from (64), (60) and (61), we have

$$\underline{F}(\underline{y}_{n+1}) = \begin{bmatrix} F_1(\vartheta_{n+1}, \omega_{n+1}, r_{n+1}, v_{n+1}) \\ F_2(\vartheta_{n+1}, \omega_{n+1}, r_{n+1}, v_{n+1}) \\ F_3(\vartheta_{n+1}, \omega_{n+1}, r_{n+1}, v_{n+1}) \\ F_4(\vartheta_{n+1}, \omega_{n+1}, r_{n+1}, v_{n+1}) \end{bmatrix} =$$

$$= \begin{bmatrix} \vartheta_{n+1} - \dfrac{h}{2}\omega_{n+1} + \vartheta_n - \dfrac{h}{2}\omega_n \\ \omega_{n+1} - \dfrac{h}{2}\dfrac{2v_{n+1}\omega_{n+1} + \mathrm{g}\sin\vartheta_{n+1}}{-r_{n+1}} + \omega_n - \dfrac{h}{2}\dfrac{2v_n\omega_n + \mathrm{g}\sin\vartheta_n}{-r_n} \\ r_{n+1} - \dfrac{h}{2}v_{n+1} + r_n - \dfrac{h}{2}v_n \\ v_{n+1} - \dfrac{h}{2}\left(\mathrm{g}\cos\vartheta_{n+1} - \dfrac{k}{m}(r_{n+1} - L) + r_{n+1}\omega_{n+1}^2\right) + v_n - \dfrac{h}{2}\left(\mathrm{g}\cos\vartheta_n - \dfrac{k}{m}(r_n - L) + r_n\omega_n^2\right) \end{bmatrix}. \quad (65)$$

Since $\underline{F}$ is a vector, each row, F_i, corresponds to one of its equations, each one involving the four variables ϑ_{n+1}, ω_{n+1}, r_{n+1}, v_{n+1}. Notice that it also involves a number of constants such as m, k and g, but also ϑ_n, ω_n, r_n, v_n. All of them are *known*: the constants are constants, and the quantities involving the n-th time step are known. Indeed, Newton's method can be imagined as a smart way of *trying to change the unknowns until you find the ones solving the equation*: just, Newton's does not go by random tentatives, but in a sensible way, ruled by the Jacobian matrix. In this view, the definition and calculation of the Jacobian matrix must involve only the quantities that makes sense to change: the unknowns! So, in this case, the Jacobian matrix for this problem reads

$$
\underline{\underline{J}} = \begin{bmatrix}
\frac{\partial F_1}{\partial \vartheta_{n+1}} & \frac{\partial F_1}{\partial \omega_{n+1}} & \frac{\partial F_1}{\partial r_{n+1}} & \frac{\partial F_1}{\partial v_{n+1}} \\
\frac{\partial F_2}{\partial \vartheta_{n+1}} & \frac{\partial F_2}{\partial \omega_{n+1}} & \frac{\partial F_2}{\partial r_{n+1}} & \frac{\partial F_2}{\partial v_{n+1}} \\
\frac{\partial F_3}{\partial \vartheta_{n+1}} & \frac{\partial F_3}{\partial \omega_{n+1}} & \frac{\partial F_3}{\partial r_{n+1}} & \frac{\partial F_3}{\partial v_{n+1}} \\
\frac{\partial F_4}{\partial \vartheta_{n+1}} & \frac{\partial F_4}{\partial \omega_{n+1}} & \frac{\partial F_4}{\partial r_{n+1}} & \frac{\partial F_4}{\partial v_{n+1}}
\end{bmatrix} =
$$

$$
= \begin{bmatrix}
1 & -\frac{h}{2} & 0 & 0 \\
\frac{h}{2}\mathrm{g}\frac{\cos\vartheta_{n+1}}{r_{n+1}} & 1+\frac{h}{r_{n+1}}v_{n+1} & -\frac{h}{2r_{n+1}^2}(2v_{n+1}\omega_{n+1}+\mathrm{g}\sin\vartheta_{n+1}) & \frac{h\omega_{n+1}}{r_{n+1}} \\
0 & 0 & 1 & -\frac{h}{2} \\
\frac{h}{2}\mathrm{g}\sin\vartheta_{n+1} & -hr_{n+1}\omega_{n+1} & -\frac{h}{2}\left(\omega_{n+1}^2-\frac{k}{m}\right) & 1
\end{bmatrix} \tag{66}
$$

5.2 Towards a *smarter* formulation

In the previous section we have provided the $\underline{F}$ and the $\underline{\underline{J}}$ for the trapezoidal rule. Yet, we proceeded in a very straightforward, yet calculus-intensive path. There is a smarter way to formulate the methods, focusing on the structure of the methods. Take for example the trapezoidal rule $\underline{F}$ from (64),

$$
\underline{F}(\underline{y}_{n+1}) = \underline{y}_{n+1} - \frac{h}{2}\underline{f}(\underline{y}_{n+1}, t_{n+1}) - \underline{y}_n - \frac{h}{2}\underline{f}(\underline{y}_n, t_n).
$$

About the implementation of $\underline{F}$ there is no significant simplification or re-formulation we can do. However, about the Jacobian, it is convenient to identify two contributions: one coming from $\underline{y}_{n+1}$, and one coming from $\underline{f}(\underline{y}_{n+1}, t_{n+1})$. In particular, what we could do, is to write the total Jacobian matrix $\underline{\underline{J}}$ for the trapezoidal rule as

$$
\underline{\underline{J}} = \underline{\underline{J}}_1 - \frac{h}{2}\underline{\underline{J}}_0, \tag{67}
$$

where, in this case (and in most of the cases we investigate)

$$
\underline{\underline{J}}_1 = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}, \tag{68}
$$

and $\underline{\underline{J}}_0$ can be evaluated as

$$
\underline{\underline{J}}_0 = \begin{bmatrix}
\frac{\partial f_1}{\partial \vartheta_{n+1}} & \frac{\partial f_1}{\partial \omega_{n+1}} & \frac{\partial f_1}{\partial r_{n+1}} & \frac{\partial f_1}{\partial v_{n+1}} \\
\frac{\partial f_2}{\partial \vartheta_{n+1}} & \frac{\partial f_2}{\partial \omega_{n+1}} & \frac{\partial f_2}{\partial r_{n+1}} & \frac{\partial f_2}{\partial v_{n+1}} \\
\frac{\partial f_3}{\partial \vartheta_{n+1}} & \frac{\partial f_3}{\partial \omega_{n+1}} & \frac{\partial f_3}{\partial r_{n+1}} & \frac{\partial f_3}{\partial v_{n+1}} \\
\frac{\partial f_4}{\partial \vartheta_{n+1}} & \frac{\partial f_4}{\partial \omega_{n+1}} & \frac{\partial f_4}{\partial r_{n+1}} & \frac{\partial f_4}{\partial v_{n+1}}
\end{bmatrix}, \tag{69}
$$

which reads, considering for example the elastic pendulum,

$$\underline{\underline{J}}_0 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -g\dfrac{\cos\vartheta_{n+1}}{r_{n+1}} & -\dfrac{2v_{n+1}}{r_{n+1}} & \dfrac{2v_{n+1}\omega_{n+1}+g\sin\vartheta_{n+1}}{r_{n+1}^2} & -2\dfrac{\omega_{n+1}}{r_{n+1}} \\ 0 & 0 & 0 & 1 \\ -g\sin\vartheta_{n+1} & 2r_{n+1}\omega_{n+1} & \omega_{n+1}^2 - \dfrac{k}{m} & 0 \end{bmatrix}. \tag{70}$$

The great advantage of using the formulation with $\underline{\underline{J}}_0$ and $\underline{\underline{J}}_1$ is that, actually, it is disengaged from a specific discretization scheme. Indeed, in all the methods we have studied so far, *e.g.*, BDF1, BDF2, trapezoidal rule, even TR-BDF2, we always have only terms involving $\underline{y}_{n+1}$ and $\underline{f}(\underline{y}_{n+1}, t_{n+1})$. Correspondingly, $\underline{\underline{J}}_1$ and $\underline{\underline{J}}_0$ have universal application. In other words, this means that one could write once a routine evaluating these matrices, it can be used to whatever scheme we want to formulate! To demonstrate this point, the next section demonstrates its application to the formulation of a TR-BDF2 scheme.

5.3 Solution by the TR-BDF2 scheme

We are now going to attack this problem with the TR-BDF2 algorithm. The first step is the TR, which is very similar to what we have studied in the previous parts of this section. In particular, we have:

$$\underline{F}_{\mathrm{TR}}(\underline{y}_{n+\gamma}) = \underline{y}_{n+\gamma} - \gamma\frac{h}{2}\underline{f}(\underline{y}_{n+\gamma}, t_{n+\gamma}) - \underline{y}_n - \gamma\frac{h}{2}\underline{f}(\underline{y}_n, t_n), \tag{71}$$

and the corresponding Jacobian therefore reads

$$\underline{\underline{J}}_{\mathrm{TR}} = \underline{\underline{J}}_1 - \gamma\frac{h}{2}\underline{\underline{J}}_0. \tag{72}$$

So far, everything is very similar to the standard TR rule. However, for the BDF2 step, we have to restart from (51), and write it as

$$\underline{F}_{\mathrm{BDF}}(\underline{y}_{n+1}) = \underline{y}_{n+1} - h\frac{1-\gamma}{2-\gamma}\underline{f}(\underline{y}_{n+1}, t_{n+1}) - \underline{y}_{n+\gamma}\frac{1}{\gamma(2-\gamma)} + \underline{y}_n\frac{(1-\gamma)^2}{\gamma(2-\gamma)}, \tag{73}$$

and, using a very similar idea to the one presented in the previous section, the Jacobian simply reads

$$\underline{\underline{J}}_{\mathrm{BDF}} = \underline{\underline{J}}_1 - h\frac{1-\gamma}{2-\gamma}\underline{\underline{J}}_0. \tag{74}$$

Now you can appreciate the power of this formulation: $\underline{\underline{J}}_1$ and $\underline{\underline{J}}_0$ are the very same used for the simple trapezoidal rule, (67).

This is sufficient to implement the code, but, before presenting it, it could be a good time to clarify which could be the best choice for γ. A possible idea is to define γ in such a way that the two Jacobians are equal. This simplifies a bit the formulation, and maybe, in complex problems, save some calculations. If we choose

$$\underline{\underline{J}}_{\mathrm{BDF}} = \underline{\underline{J}}_{\mathrm{TR}},$$

we have

$$\underline{\underline{J}}_1 - h\frac{1-\gamma}{2-\gamma}\underline{\underline{J}}_0 = \underline{\underline{J}}_1 - \gamma\frac{h}{2}\underline{\underline{J}}_0,$$

where, after obvious simplifications, we end up with

$$\frac{1-\gamma}{2-\gamma} = \frac{\gamma}{2} \Longrightarrow 2 - 2\gamma = 2\gamma - \gamma^2,$$

or, ultimately,

$$\gamma^2 - 4\gamma + 2 = 0.$$

The solutions of this equation are:

$$\gamma = -\frac{b}{2} \pm \sqrt{\left(\frac{b}{2}\right)^2 - ac} = 2 \pm \sqrt{4-2} = 2 \pm \sqrt{2},$$

but, because we asked γ to be included from 0 to 1 for hypothesis, our solution is

$$\gamma = 2 - \sqrt{2}.$$

Now that all the details are reported, it is time to show the code. First, we report the function evaluating $\underline{f}$ and the Jacobian parts.

Matrix assembler for the elastic pendulum problem (f_AssemEquation.m)

```
function [fnew,Jmat0,Jmat1] = f_AssemEquation(ynew)

fnew = zeros(4,1);

k  = 10;          % spring constant
m  = 1;           % mass
l0 = 1;           % rest length
g  = 9.81;        % gravitational acceleration

Jmat1 = eye(4);  % Jmat1 is simply the identity

theta = ynew(1);
omega = ynew(2);
r     = ynew(3);
v     = ynew(4);

fnew(1) = omega;
fnew(2) = (2*v*omega+g*sin(theta))/(-r);
fnew(3) = v;
fnew(4) = g*cos(theta) - k/m*(r-l0)+r*omega^2;

Jmat0(1,1) = 0;
Jmat0(1,2) = 1;
Jmat0(1,3) = 0;
Jmat0(1,4) = 0;

Jmat0(2,1) = -g*cos(theta)/r;
Jmat0(2,2) = -2*v/r;
Jmat0(2,3) = (2*v*omega+g*sin(theta))/(r^2);
Jmat0(2,4) = -2*omega/r;

Jmat0(3,1) = 0;
Jmat0(3,2) = 0;
Jmat0(3,3) = 0;
Jmat0(3,4) = 1;

Jmat0(4,1) = -g*sin(theta);
Jmat0(4,2) = 2*r*omega;
Jmat0(4,3) = omega^2-k/m;
Jmat0(4,4) = 0;
```

Finally, follows the driver for the previous function, implementing the TR-BDF2 solution of the elastic pendulum problem.

TR-BDF2 solution of the elastic pendulum problem (Main_Elastic_TRBDF2.m)

```
clear
close all
clc

% time step of the solver
h = 0.05; % time step of the solver

% parameter gamma for the TR-BDF2 algorithm
gamma = 2-sqrt(2);
```

```
% parameters for the Newton-Raphson algorithm
tol     = 1e-10; % tolerance for the stopping criterion
itermax = 50;    % maximum number of iterations before exiting while loop

% parameters of the physical model
k  = 10;         % spring constant
m  = 1;          % mass
l0 = 1;          % rest length
g  = 9.81;       % gravitational acceleration

% initial conditions
theta = pi/3;
omega = 2;
r = 1;
v = 0;

% time axis
tvet = 0:h:20;

% setting the initial condition
yvet = [theta;omega;r;v];
ygamma = yvet;

% initializing yold for the first step
yold = yvet;

% initializing f_{n+1} (fnew) and f_n (fold)
fnew = zeros(4,1);
fold = zeros(4,1);

% setting f_n for the first time
fold(1) = omega;
fold(2) = (2*v*omega+g*sin(theta))/(-r);
fold(3) = v;
fold(4) = g*cos(theta) - k/m*(r-l0)+r*omega^2;

for indTime = 1:length(tvet)

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Trapezoidal rule step Newton-Raphson loop
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    count = 0;    % setting iteration counter to 0
    res   = inf;  % setting residual to infinite to enter Newton loop at least once
    while(res>=tol && count<itermax)

        count = count+1;

        [fgamma,Jmat0,Jmat1] = f_AssemEquation(ygamma);

        F_TR = ygamma - gamma*h/2*fgamma - yold - gamma*h/2*fold;
        Deltay = -(Jmat1 - h/2*Jmat0)\F_TR;

        ygamma = ygamma + Deltay;

        res = norm(Deltay);

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % BDF2 step Newton-Raphson loop
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    count = 0;    % setting iteration counter to 0
    res   = inf;  % setting residual to infinite to enter Newton loop at least once
    while(res>=tol && count<itermax)

        count = count+1;

        [fnew,Jmat0,Jmat1] = f_AssemEquation(yvet);

        F_BDF2 = yvet - h*(1-gamma)/(2-gamma)*fnew - ygamma/(gamma*(2-gamma)) + yold*(1-gamma)^2/(↩
            gamma*(2-gamma));
        Deltay = -(Jmat1 - h*(1-gamma)/(2-gamma)*Jmat0)\F_BDF2;

        yvet = yvet + Deltay;

        res = norm(Deltay);

    end

    % The results of this iteration become the inputs of the next one
    fold = fnew;
```

```
    yold = yvet;

    resvet(indTime) = res;
    thetavet(indTime) = yvet(1);
    rvet(indTime) = yvet(3);

end

figure(1)
hold on
plot(tvet,rvet)
```

References

[1] S. Dharmaraja, "An analysis of the TR-BDF2 integration scheme," M.Sc. dissertation, MIT, Massachussets, 2007.

[2] G. Strang, "Computational Science and Engineering," *Wellesley-Cambridge Press*, 2007.

[3] A. Iserles, "A First Course in the Numerical Analysis of Differential Equations," *Cambridge University Press*, 1996.

[4] J. D. Lambert, "Numerical methods for ordinary differential systems," *John Wiley & Sons*, 1991.

[5] R. E. Bank, W. M. Coughran Jr, W. Fichtner, E. H. Grosse, and D. J. Rose, "Transient simulation of silicon devices and circuits," *IEEE Trans. Comput.-Aided Des.,* vol. CAD-4, no. 4, pp. 436–451, Oct. 1985.

[6] J. D. Edwards, J. E. Morel, and D. A. Knoll, "Nonlinear variants of the TR/BDF2 method for thermal radiative diffusion," *J. Comp. Phys.,* vol. 230, pp. 1198-1214, 2011.