Hand in Assignment 4- Implicit Integrators

Fikri Farhan Witjaksono *Oliver Wallin*

25 October 2019

Task 1 . Implicit RK1(a)

Algorithm: Integration of implicit ODE

Input: $x_0, u(t_0), \ldots, u(t_{N-1})$ and Δt

for $k = 0 : N-1$ **do**

Solve

$$F\left(\frac{x_{k+1} - x_k}{\Delta t}, x_k, u(t_k)\right) = 0 \quad (2)$$

for x_{k+1}

return $x_{0,\ldots N}$

It is in fact an Explicit Euler scheme since we could notice that it has form

$$x_{k+1} = x_k + \Delta t \cdot f\big(x_k, u(t_k)\big) \quad (1)$$

However, if we analyze further, we could see that (1) is used explicitly as a function of $x_k, u(t_k)$ as shown if we move x_k, Δt and $f\big(x_k, u(t_k)\big)$ to the Right Hand Side (RHS).

$$\frac{x_{k+1} - x_k}{\Delta t} - f\big(x_k, u(t_k)\big) = 0 \ (2)$$

Therefore, we could conclude that it is Explicit Euler Scheme in disguise.

Task 1 . Implicit RK1(b)

We can change the algorithm above to have the index k+1 in place of k as shown below in (3)

$$F\left(\frac{x_{k+1} - x_k}{\Delta t} - f\big(x_{k+1}, u(t_{k+1})\big)\right) = 0 \quad (3)$$

If we simplify further it will look like (4) below

$$\frac{x_{k+1} - x_k}{\Delta t} - f\big(x_{k+1}, u(t_{k+1})\big) = 0 \ (4)$$

Which if we move the terms to RHS it will look like (5)

$$x_{k+1} = x_k + \Delta t \cdot f\big(x_{k+1}, u(t_{k+1})\big) \quad (5)$$

which is clearly an explicit euler scheme

Task 2. (a). Write the code for Implicit RK4 Scheme

$$\begin{array}{c|ccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & a_{22} & \dots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

Figure 2.1. Coefficient of the Butcher Table

$$\begin{array}{c|cc} \frac{1}{2}-\frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4}-\frac{\sqrt{3}}{6} \\ \frac{1}{2}+\frac{\sqrt{3}}{6} & \frac{1}{4}+\frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

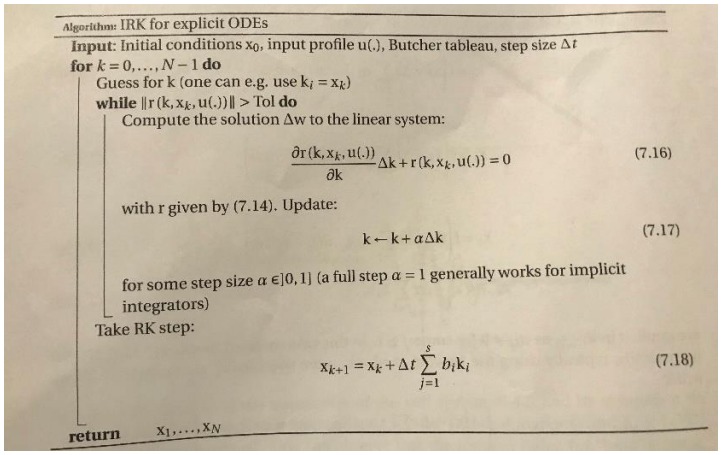
Figure 2.2. Gauss Legendre Butcher Table

In this task, we try to build the Implicit RK4 scheme by using the Newton method. In order to do this, Gauss-Legendre's Butcher table is being used and being inserted into the residual function 'r'. Butcher Table coefficients that is being used in the Gauss-Legendre is described well by figure 1 and 2. r equation is shown below.

$$x = f(x; t)$$

$$r(k, x_k, u(.)) := \begin{bmatrix} f(x_k + \Delta t \sum_{j=1}^{s} a_{1j} k_j, u(t_k + c_1 \Delta t)) - k_1 \\ f(x_k + \Delta t \sum_{j=1}^{s} a_{sj} k_j, u(t_k + c_2 \Delta t)) - k_2 \end{bmatrix} = 0$$

After finding the residual equation (r) and its jacobian derivative $\frac{\partial r(k, x_k, u(.))}{\partial k}$, then we try the algorithm for Implicit Runge-Kutta as shown in Figure 3 below. Note that r $\in \mathbb{R}^{n \cdot s}$, k $\in \mathbb{R}^{n \cdot s}$ and $\frac{\partial r(k, x_k, u(.))}{\partial k} \in \mathbb{R}^{n \cdot s \; x \; n \cdot s}$ where n is the dimension of the state and s is the number of stages.



Algorithm: IRK for explicit ODEs

Input: Initial conditions x_0, input profile u(.), Butcher tableau, step size Δt

for $k = 0, \dots, N-1$ **do**

Guess for k (one can e.g. use $k_i = x_k$)

while $\|r(k, x_k, u(.))\| >$ Tol **do**

Compute the solution Δw to the linear system:

$$\frac{\partial r(k, x_k, u(.))}{\partial k} \Delta k + r(k, x_k, u(.)) = 0 \qquad (7.16)$$

with r given by (7.14). Update:

$$k \leftarrow k + \alpha \Delta k \qquad (7.17)$$

for some step size $\alpha \in]0, 1]$ (a full step $\alpha = 1$ generally works for implicit integrators)

Take RK step:

$$x_{k+1} = x_k + \Delta t \sum_{j=1}^{s} b_i k_i \qquad (7.18)$$

return $x_1, \dots, x_N$

Figure 2.3. Implicit Runge Kutta Algorithm for explicit ODEs [1]

Task 2. (b). Compare the result between IRK4 and ERK4 for test function

In Task 2(b) , we are tasked with different function to solve, that is the test function as shown below. we set the $t_f = 2,\ \lambda = -2,\ \Delta t = 0.1.$

$$\dot{x} = \lambda x$$

First we defined the residual function in separate line (rFunction) having the input of (t,X,dt,K2) and output of r and $\frac{\partial r(k, x_k, u(.))}{\partial k}$ by using Matlab Symbolic Function. After that, we input the residual functions and its jacobian into the algorithm provided by figure 3. In this algorithm, we set the tolerance to be 10^{-6}. Then, the result of x_{k+1} and t_{k+1} is plotted where y-axis is x_{k+1} and x-axis is t_{k+1}.

After that, the plot result of this is shown in Figure 4 below.

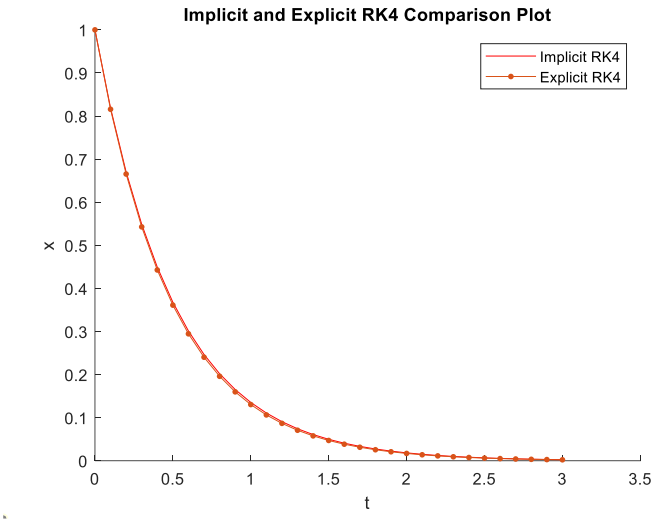


Figure 2.4. Implicit vs Explicit RK4 solution comparison

In the Figure 1, it is shown that the Implicit RK4 (IRK4) scheme solution gives the plot which is smaller in error compared to the Explicit RK4 (ERK4) scheme and it is somehow difficult to distinguish between the exact solution and IRK4 solution if we plot it together. In the matlab code, it is also shown that the Explicit RK4 scheme must have higher number of stages required as if we want to achieve order of accuracy of 4, we need 4 stages since $o = s$ for ERK4. On the other hand, we only need twice less stages for IRK4 due to the fact that $o = 2s$ for IRK4.

Task 2. (c). Compare the result between IRK4 and ERK4 for VanDerPol Dynamics

In this task, we did it with similar method to the task 2(b), however with different definition of input function to the residual function. Here, we use the

$$\dot{x} = y$$
$$\dot{y} = u\left(1 - x^2\right) y - x$$

where $t_f = 25\ ,\ u = 5,\ \Delta t = 0.01.$

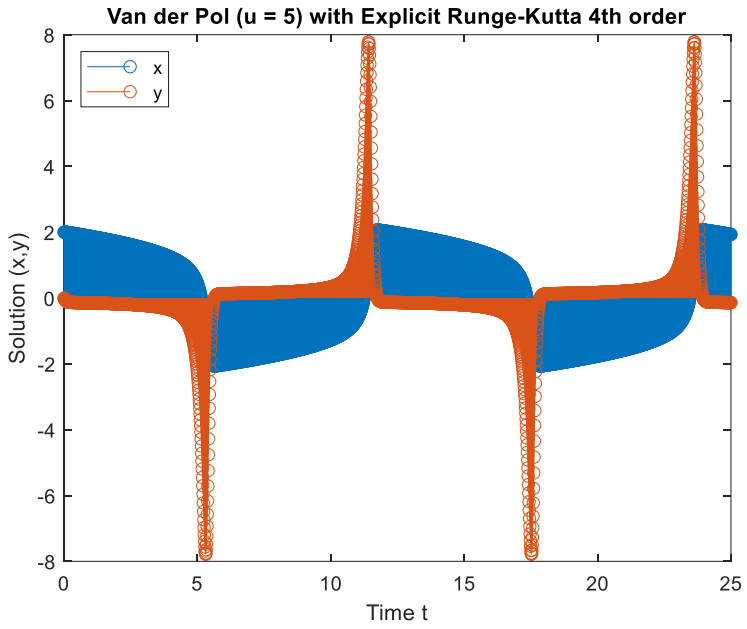


Figure 2.5. Explicit Runge Kutta 4th order Van der Pol Solution

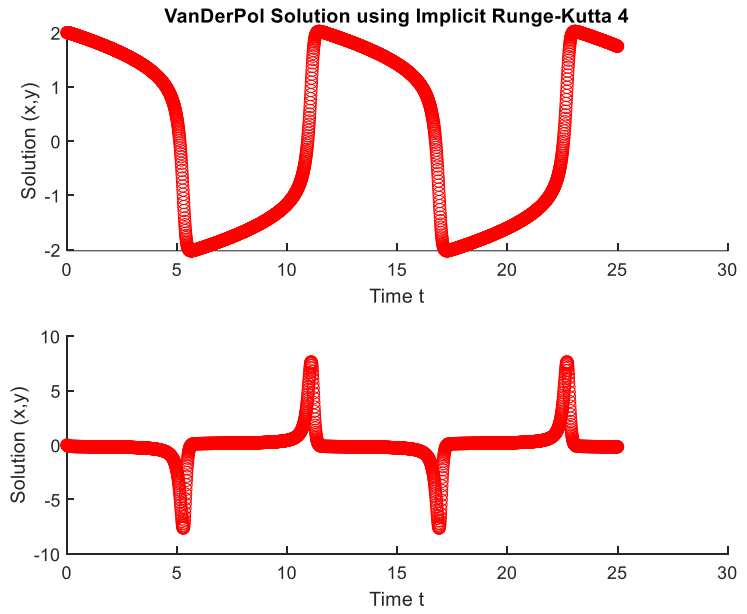


Figure 2.6. Implicit Runge Kutta 4th order Van Der Pol Solution

Table 2. Comparison of extreme point values for Δt = 0.01

	Target ODE 45	Explicit RK4	Implicit RK4
Max Point of y	7.6399	7.7809	7.635
Min Point of y	-7.6509	-7.786	-7.637

The result of the IRK4 as well as ERK4 is shown in the Figure 5, Figure 6 above. The comparison of the result can be done by comparing the extreme point values of both schemes. Assuming that the ODE45 has the best accuracy due to its adaptive integrator nature, we could compare the accuracy of both vs ODE45. This is shown clearly in Table 2 which states that Implicit RK4 result has the best accuracy compare to the Explicit RK4. However, in our simulation, if we would like to solve a more complicated problems, we will see that Implicit RK4 computational time is much more hence higher computational cost.

Task 3. (a). Fully-Implicit DAE integration using IRK4 Integrator to solve Lagrange Pendulum Problem

First, let us define the fully implicit DAE as F($\dot{x}, x, t$) $= 0$. This is shown in Matlab as below

$$\mathrm{F}(\dot{x}, x, t) = \begin{bmatrix} v - \dot{p} \\ -\mathrm{mg} \cdot [0\ 0\ 1] - \mathrm{z} \cdot \mathrm{p} - m \cdot \dot{v} \\ p^T \cdot \dot{v} + v^T \cdot v \end{bmatrix}$$

$$\text{where } \dot{v} = -\mathrm{mg} \cdot [0\ 0\ 1] - \mathrm{z} \cdot \mathrm{p} \text{ and } K = \begin{bmatrix} \dot{p} \\ \dot{v} \end{bmatrix}$$

Then, define x as 6x1 matrix with the initial condition shown below.

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \in \mathbb{R}^6$$

K is a matrix whose size is n·s (K$\in \mathbb{R}^{n \cdot s}$) and z as a matrix whose size is m·s (z$\in \mathbb{R}^{m \cdot s}$). z is defined as algebraic variable.

$$r(k, x_k, u(.)) \coloneqq \begin{bmatrix} f(x_k + \Delta t \sum_{j=1}^{s} a_{1j} k_j, z_1, u(t_k + c_1 \Delta t)) - k_1 \\ f(x_k + \Delta t \sum_{j=1}^{s} a_{sj} k_j, z_2, u(t_k + c_2 \Delta t)) - k_2 \end{bmatrix} = 0$$

Here, after that we do the procedure as it is done in task 2 with slight difference in algorithm (as shown in Appendix A) of defining the residual function (r). Since we have only 2 steps it is shown as the equation above. Then, we will have 5 input of (t, x, Δt, K, z) instead of 4 input (t, x, Δt, K) comparing to the previous question.

After that instead of doing the Jacobian derivative only on K, we define a matrix W which comprises of K and z with size of and use it in order to get $\frac{\partial r(k, x_k, u(.))}{\partial V}$ whose size is $\mathbb{R}^{(m+n) \cdot s\ x\ (m+n) \cdot s}$

$$V = \begin{bmatrix} K \\ z \end{bmatrix}, \qquad where\ V \in \mathbb{R}^{(m+n) \cdot s}$$

Then at the end of the algorithm, we record the constraint 'C' as shown below

$$C_k = \frac{1}{2} \cdot P^T \cdot P - L^2$$

The result of x_{k+1} is plotted where we will have 3 different axis since it is a 3D pendulum as shown in Figure (1) for $\Delta t = 0.001$. C_k and t_{k+1} is then plotted against each other in an x-y axis on various figure as shown below as we varied the Δt.

We could observe in Fig (2)-(5) that the Constraint vary as we vary the **Δt.** From **Δt = 0.1-Δt = 0.01,**

The constraint is oscillating between its maximum point of zero and its minimum point which is getting smaller as we decrease time step.

From **Δt = 0.001-Δt = 0.0001,**

as we decrease further the time step , the constraint is oscillating on the positive axis and its maximum point is getting smaller and smaller as we decrease **Δt. These behavior is due to instability in the simulation due to its nonlinear nature. Using a more accurate integrator (e.g. using sensitivity in simulation) might give us better results** Hence it is better to take it smaller than **0.001 as C should be zero at perfect accuracy.**

	Δt = 0.1	**Δt = 0.01**	**Δt = 0.001**	**Δt = 0.0001**
Maximum point	0	0	$2.7 \cdot 10^{-8}$	$1.2 \cdot 10^{-10}$
Minimum point	$-1 \cdot 10^{-4}$	$-1 \cdot 10^{-8}$	0	0

Table 3. Comparison of extreme point of Constraint C for different Δt

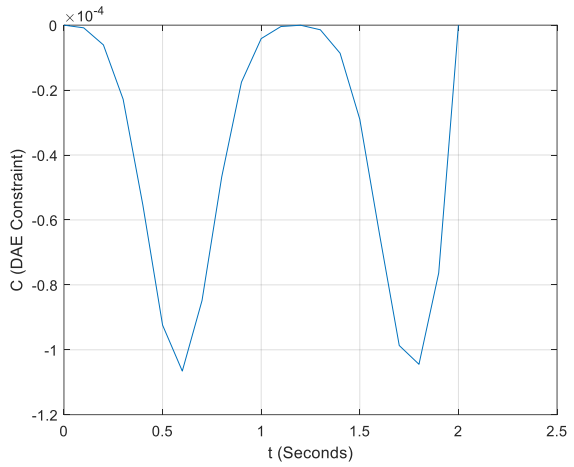


Figure 3.1 C at **Δt = 0.1**

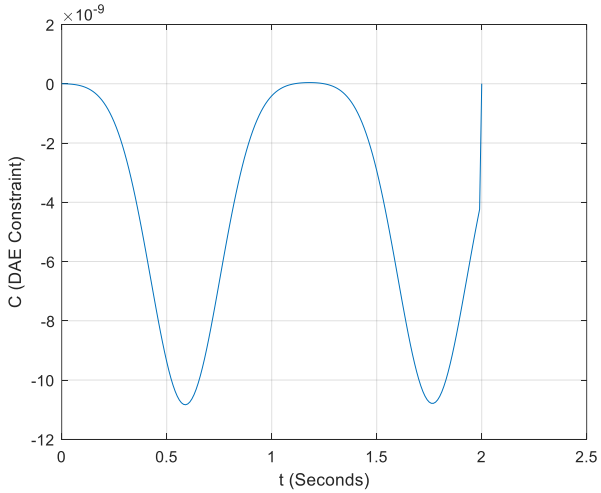


Figure 3.2 C at **Δt = 0.01**

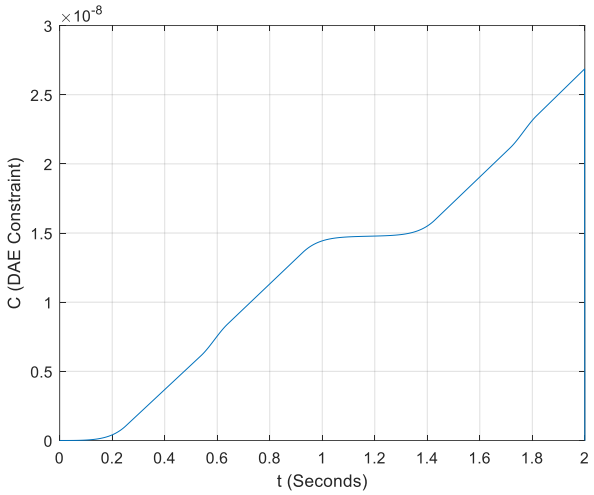


Figure 3.3. C at **Δt = 0.001**

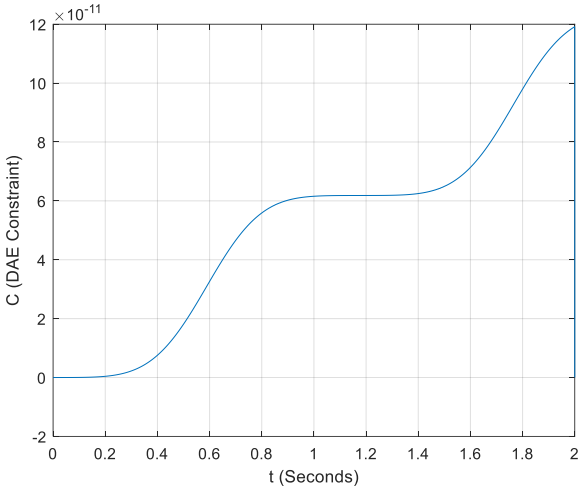


Figure 3.4 C at **Δt = 0.0001**

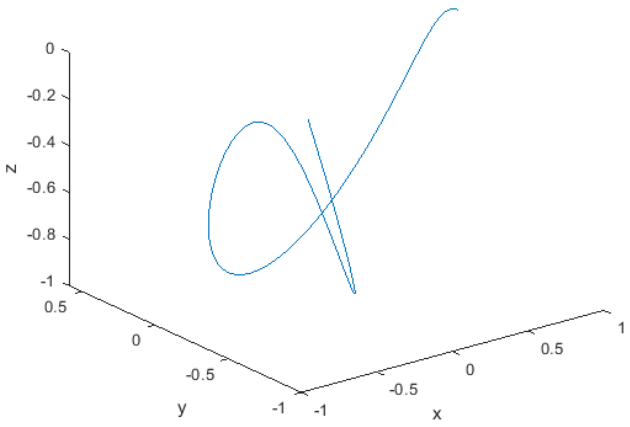


Figure 3.**5 Dynamics at Δt = 0.0001**

Task 3. (a). Fully-Implicit DAE integration using IRK4 Integrator to solve Lagrange Pendulum Problem Directly

$$\mathrm{F}(\dot{x}, x, t) = \begin{bmatrix} v - \dot{p} \\ -\mathrm{mg} \cdot [0\ 0\ 1] - \mathrm{z} \cdot \mathrm{p} - m \cdot \dot{v} \\ \frac{1}{2} \cdot p^T \cdot p - L^2 \end{bmatrix}$$

The C tend to start increasing exponentially before dropping steeply in the positive region. This is due to solving the 3D pendulum model directly form Lagrange. The C will decrease as we decrease time step.

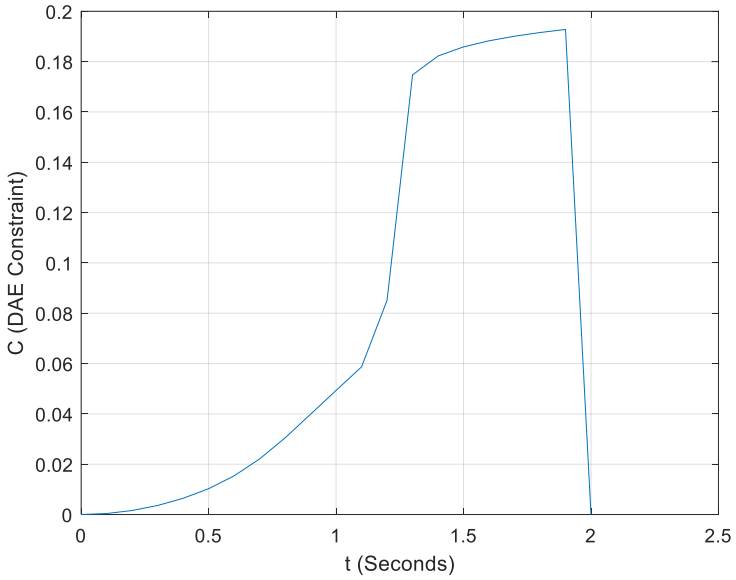


Figure 3.6 C at **Δt = 0.1**

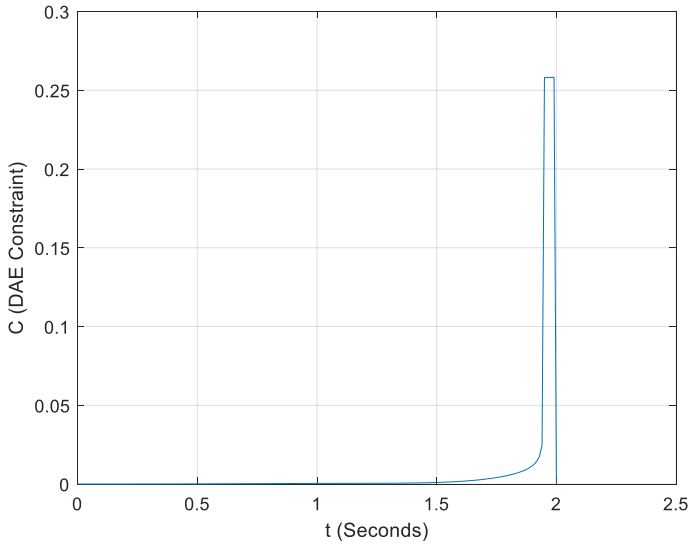


Figure 3.7 C at **Δt = 0.01**

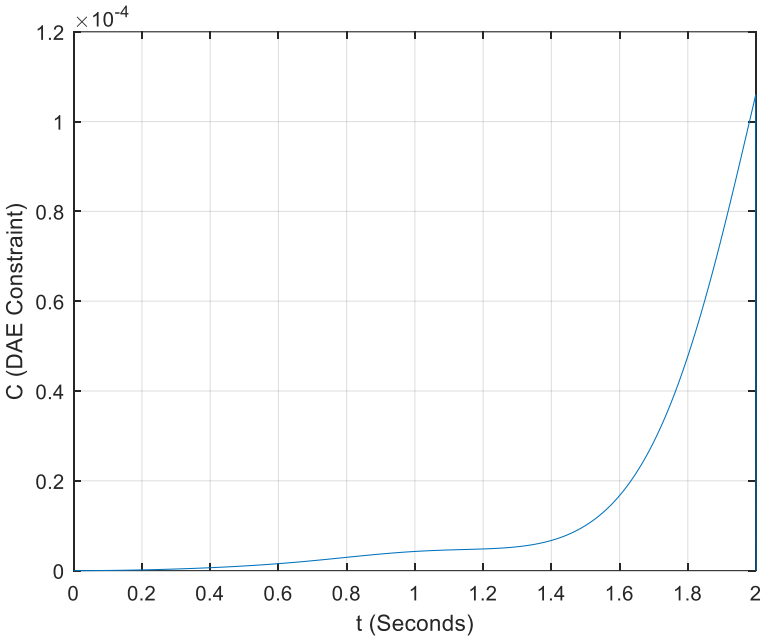


Figure 3.8 C at **Δt = 0.001**

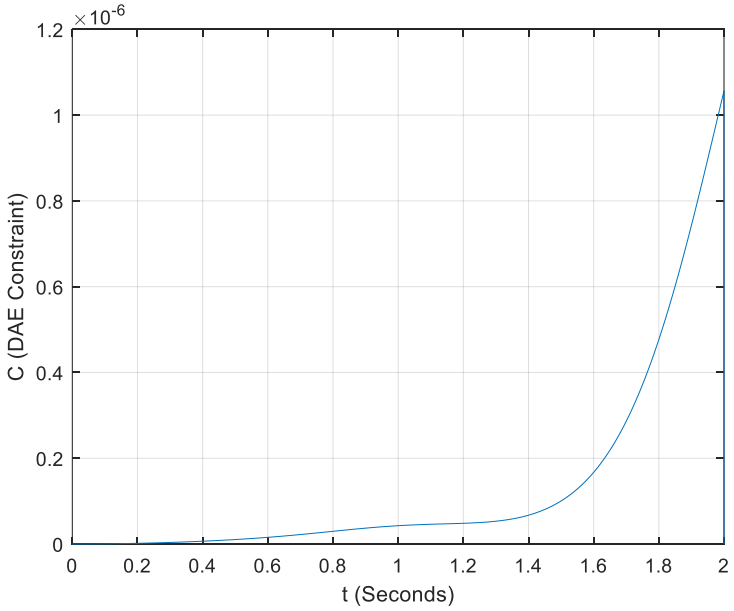
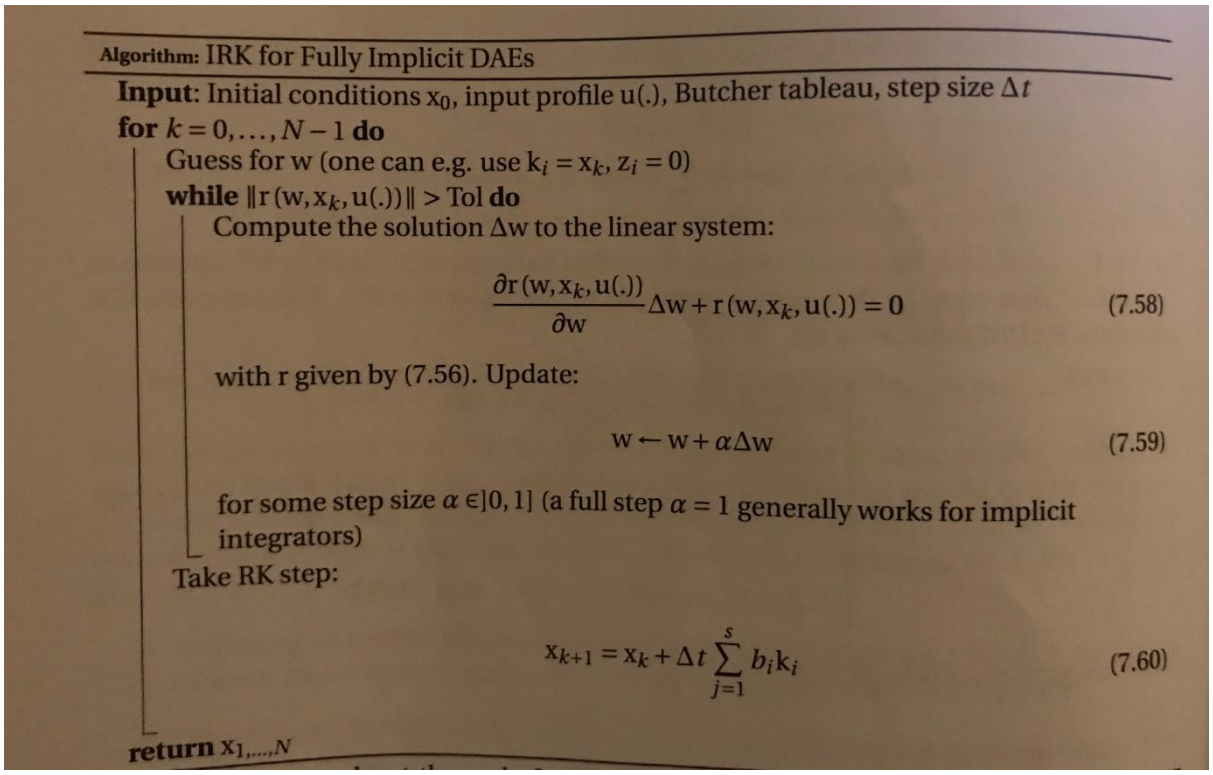


Figure 3.9 C at **Δt = 0.0001**

Appendix A. IRK FULLY IMPLICIT DAE ALGORITHM



Algorithm: IRK for Fully Implicit DAEs

Input: Initial conditions x_0, input profile u(.), Butcher tableau, step size Δt

for $k = 0, \ldots, N-1$ **do**

Guess for w (one can e.g. use $k_i = x_k$, $z_i = 0$)

while $\|r(w, x_k, u(.))\| > \text{Tol}$ **do**

Compute the solution Δw to the linear system:

$$\frac{\partial r(w, x_k, u(.))}{\partial w} \Delta w + r(w, x_k, u(.)) = 0 \tag{7.58}$$

with r given by (7.56). Update:

$$w \leftarrow w + \alpha \Delta w \tag{7.59}$$

for some step size $\alpha \in]0, 1]$ (a full step $\alpha = 1$ generally works for implicit integrators)

Take RK step:

$$x_{k+1} = x_k + \Delta t \sum_{j=1}^{s} b_i k_i \tag{7.60}$$

return $x_{1,\ldots,N}$