Technical University of Denmark

Scientific Computing for differential equations

Final Report

Author:
Victor Niaussat

March 26, 2021

Contents

Note

In the various sections, we will only display the 3D CSTR solutions and not the 1D CSTR solution because the display is essentially the same. The difference occurs at the beginning of the simulation. Where there is a slight difference.

1 Test equation for ODEs

The analytical solution to the test equation

$$\dot{x}(t) = \lambda x(t), x(0) = x_0 \tag{1}$$

is

$$x(t) = x_0 \exp \lambda t \tag{2}$$

Truncation errors in numerical integration are of two kinds:

- local truncation errors – the error caused by one iteration
- global truncation errors – the cumulative error caused by many iterations.

We can calculate that using the formulas below for the global error

$$e_{n=1} = x_{n+1} - \hat{x}(t_{n+1;t_0,x_0})$$

and local error

$$e_{n=1} = x_{n+1} - \hat{x}(t_{n+1;t_n,x_n})$$

with $\hat{x}(t; t_0, x_0) = exp(\lambda(t - t_0))$.

Then, we compute the explicit Euler method on the test equation for $n = 100$ iteration, we obtain the figures below :

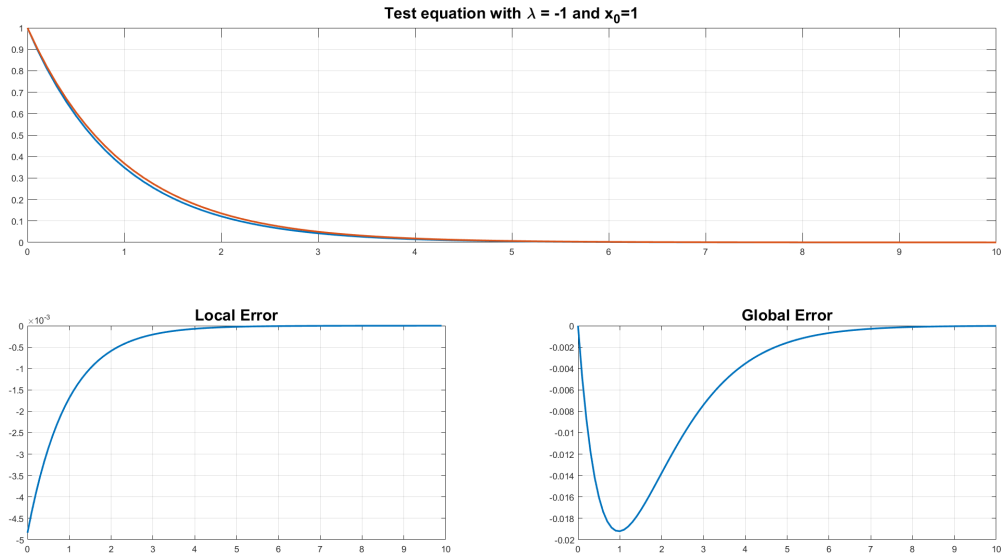

Figure 1: Explicit Euler Method

The local error and the global error are similar to those seen in John Bagterp Jørgensen's lectures, so we could expect that. Moreover, we can see that the real curve is above the curve made with the Euler Explicit method. We use the function *ExplicitEulerFixedStepSizeTest* with *@Testexp* .

We compute also the implicit Euler method on the test equation for $n = 100$ iteration, we obtain the figures below :

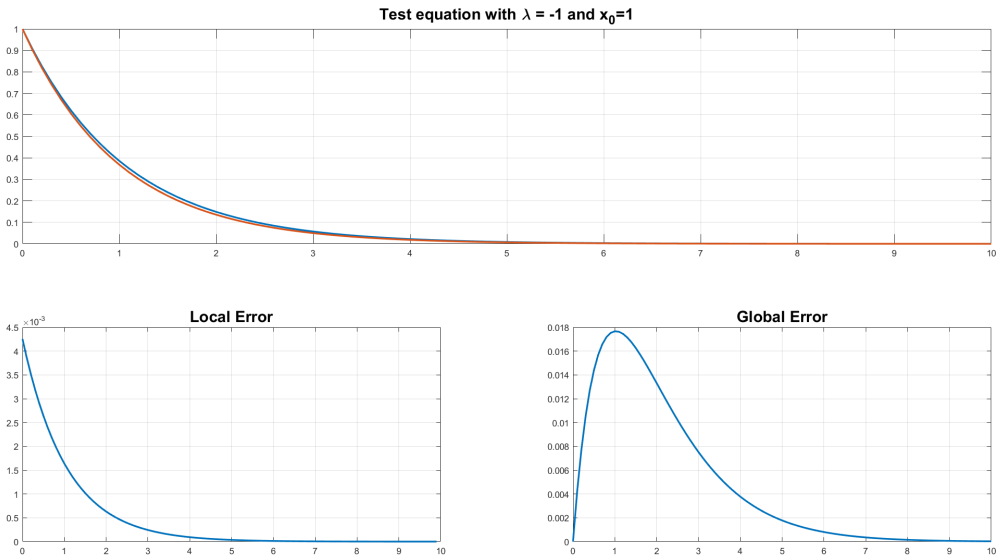

Figure 2: Implicit Euler Method

The local error and the global error are similar to those seen in John Bagterp Jørgensen's lectures. Moreover, we can see that the real curve is below the curve made with the Implicit Euler method. We use the function *ImplicitEulerFixedStepSizeTest* with *@Testexp*

Then, we need to define what it mean by order. The difference method is said to be convergent of order p if the global error e_n satisfies:

$$\forall i \in \{1, ..., n\}, e_i = O(h^p)$$

If we rewrite this equality,

$$\max |e_i| \simeq Ch^p, C \in /$$

We can compute the global error for different values of h and find an order 1 for both :

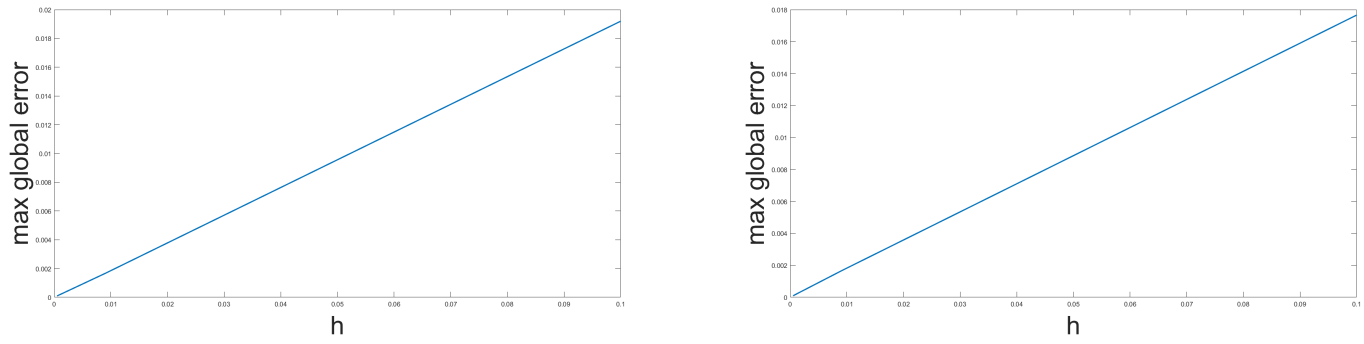

Table 1: Order of Explicit and Implicit Euler method

This is to be expected as both schemes are constructed as simple linear approximations of the differential equation, and it is straightforward to find that this is a first order method. We use the function *OrderExpandImpl*.

The behaviour of numerical methods on stiff problems can be analyzed by applying these methods to the test equation $\dot{x}(t) = \lambda x(t)$ subject to the initial condition $x(0) = x_0$ with $\lambda \in \mathbb{C}$. The solution of this equation is $x(t) = x_0 \exp \lambda t$. The solution by the Explicit Euler method is given as $x(t_n + h) = R(\lambda h)x(t_n)$ with $R(\lambda h) = 1 + \lambda h$. The stability region is

$$\mathbb{S} = \{z \in \mathbb{C} : |R(z)| < 1\} = \{z \in \mathbb{C} : |z + 1| < 1\} = \overset{\circ}{B}(-1, 1)$$

So, the stability region is the open disk of radius 1 and centre -1 in the complex plane. We can compute that and we obtain :

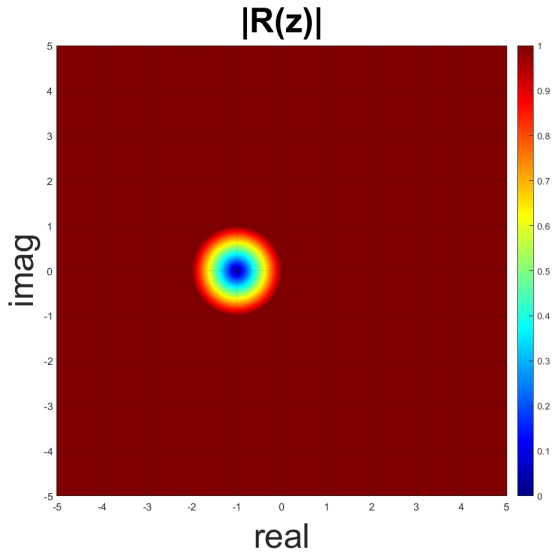

Figure 3: Stability Region of the Euler Explicit Method

The solution by the Explicit Euler method is given as $x(t_n + h) = R(\lambda h)x(t_n)$ with $R(\lambda h) = \frac{1}{1-\lambda h}$. The stability region is

$$\mathbb{S} = \{z \in \mathbb{C} : |R(z)| < 1\} = \{z \in \mathbb{C} : |z - 1| > 1\} = \mathbb{C} \backslash \overset{\circ}{B}(1, 1)$$

So, the stability region is $\mathbb{C}$ without the open disk of radius 1 and centre 1 in the complex plane. We can compute that and we obtain :

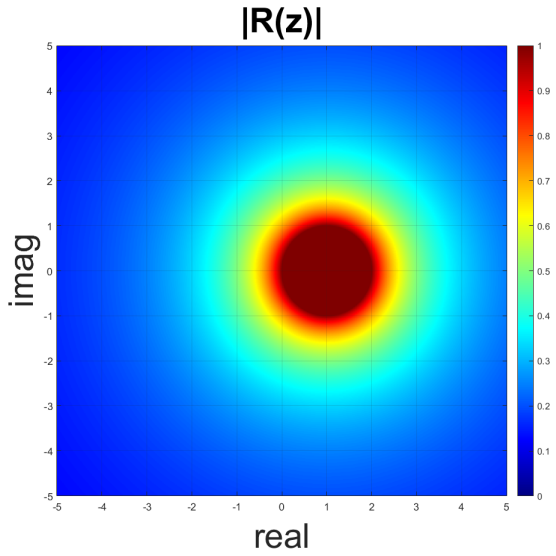

Figure 4: Stability Region of the Euler Implicit Method

We use the function *Stability*.

Then, we need to talk about A-stability. This solution approaches zero as $t \to \infty$ when $\mathrm{Re}(\lambda h) < 0$. If the numerical method also exhibits this behaviour (for a fixed step size), then the method is said to be A-stable.

The solution by a Runge-Kutta method is given as $x(t_n + h) = R(\lambda h)x(t_n)$ with $R(\lambda h) = 1 + \lambda h$. The stability region is

$$\mathbb{S} = \{z \in \mathbb{C} : |R(z)| < 1\}$$

A method is A-stable if its region of stability contains the entire left half-plane of $z = h\lambda$ So, this Euler method is not A-stable because $\exists z \in \mathbb{S} : \mathrm{Re}(z) > 0$ and this Implicit Euler method is A-stable because $\forall z \in \mathbb{S} : \mathrm{Re}(z) < 0$.

2 Explicit ODE solver

Given a differental equation $\forall t \in I, y'(t) = f(t, y(t))$ with I an interval, the purpose is to solve it using explicit Euler scheme, which is : for $i \in \{0, n-1\}$, $y_{i+1} = y_i + hf(x_i, y_i)$. So the algorithm is simply : Initialization of the step h the number of steps n, initials conditions t_0, x_0 and the function $f(t, y(t))$. For $i \in \{0, n-1\}$, $t_{i+1} = ti + h$ and $y_{i+1} = y_i + hf(t_i, y_i)$. Then, plot y datas versus t datas.

Algorithm 1: Explicit Euler method with fixed time step

Result: t and y

initialization : the step h , the number of steps n, initials conditions t_0, y_0 and the function $f(t, y(t))$;
$i = 0$
while $i < n$ **do**
    $t_{i+1} = ti + h$;
    $y_{i+1} = y_i + hf(t_i, y_i)$.;
end

We can compute that on the Vander Pol problem :

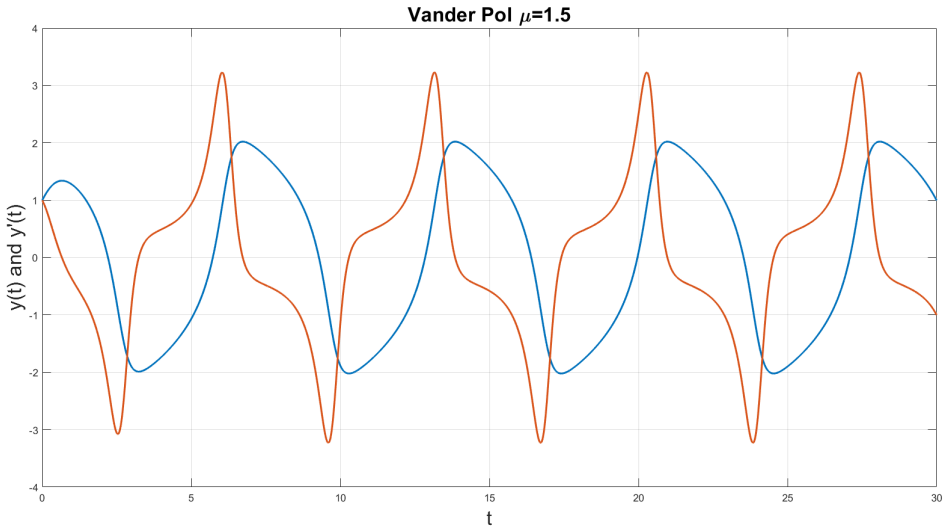

Figure 5: Explicit Euler method with fixed time step for Vander Pol Problem with $\mu = 1.5$

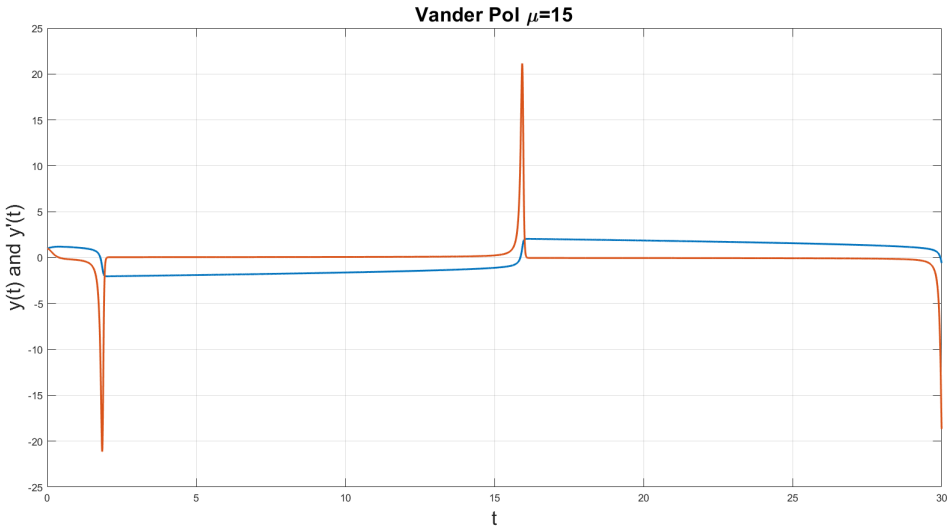

Figure 6: Explicit Euler method with fixed time step for Vander Pol Problem with $\mu = 15$

We use the function *ExplicitEulerFixedStepSize* with *@VanderPol* .
Instead of using fixed time step h, we can use the explicit Euler method with adaptive time step and error estimation using step doubling in order to control the errors of the method and to ensure stability properties such as A-stability. For the adaptive step, with use the parameter $abstol = 1e-7$ and $restol = 1e-5$

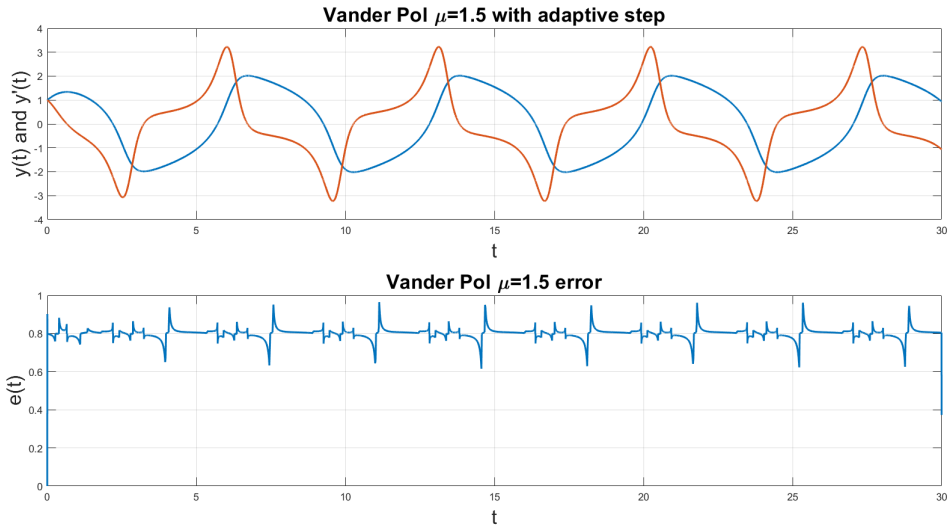

Figure 7: Explicit Euler method with adaptive step for Vander Pol Problem with $\mu = 1.5$

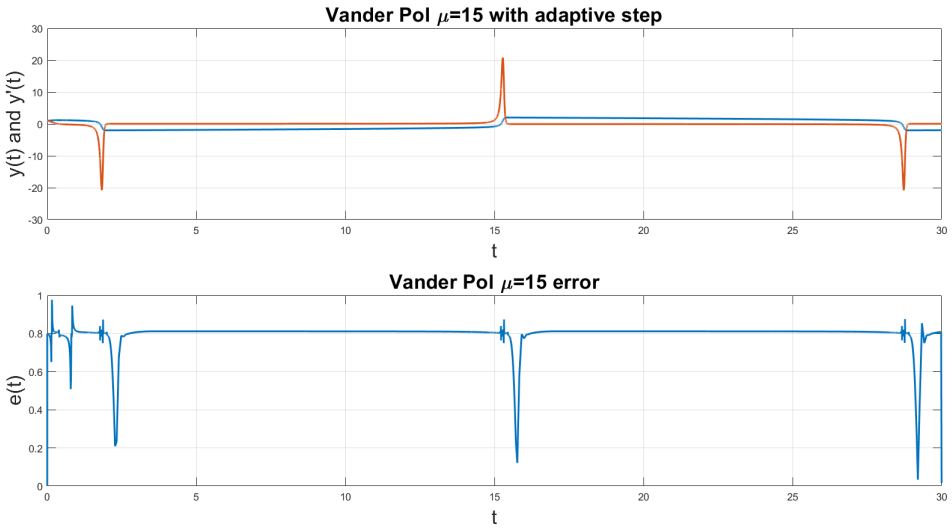

Figure 8: Explicit Euler method with adaptive step for Vander Pol Problem with $\mu = 15$

We use the function *ExplicitEulerAdaptiveStep* with *@VanderPol* Then, we can compute that on the CSTR problem in 1D and 3D. The plot are similar when we compute it for fixed time step and for adaptive time step.

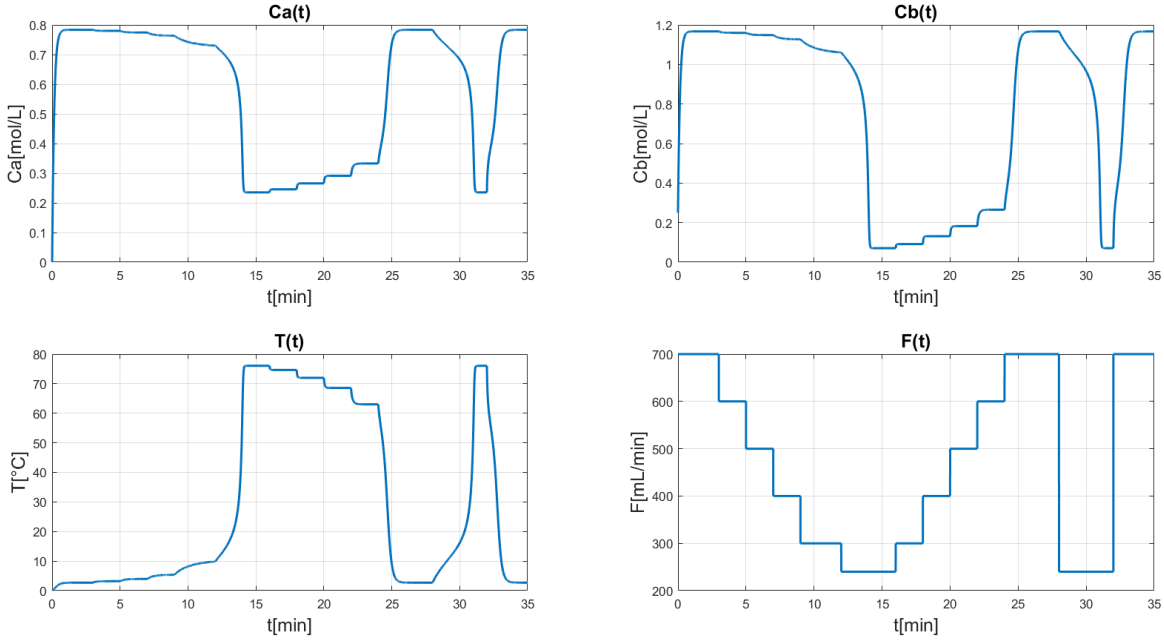

Figure 9: Explicit Euler method with fixed time step for CSTR in 3D

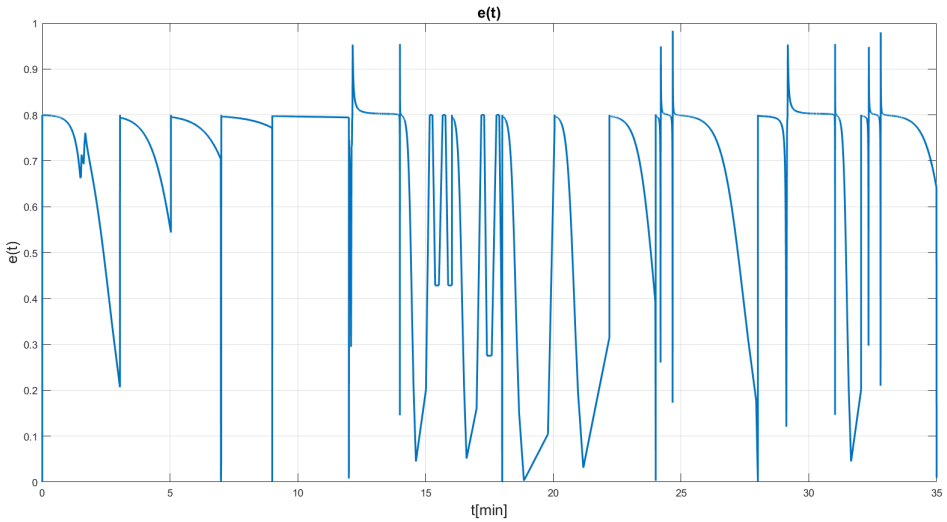

Figure 10: Error of Explicit Euler method with adaptive time step for CSTR in 3D

We use the function *ExplicitEulerAdaptiveStep* with *@CSTRR*

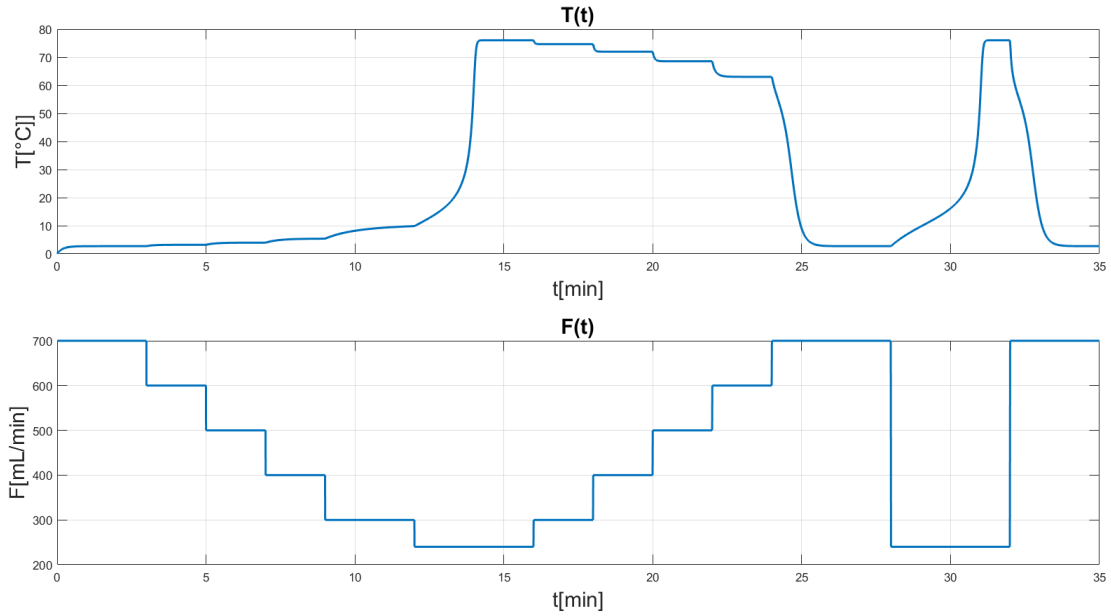

Figure 11: Explicit Euler method with fixed time step for CSTR in 1D

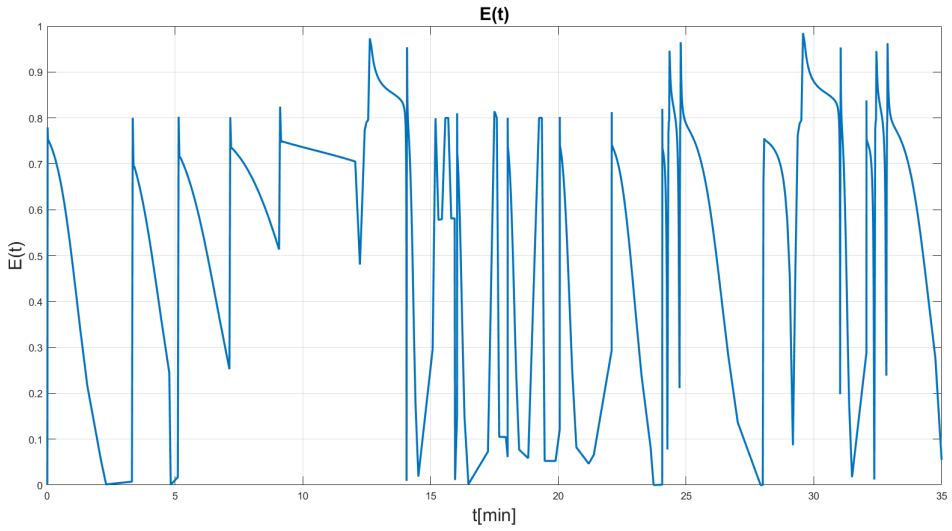

Figure 12: Error of Explicit Euler method with adaptive time step for CSTR in 1D

We use the function *ExplicitEulerAdaptiveStep* with *@CSTRR1D* Error for CSTR problem in 3D is very changing and there are many values around 0.8.

To compare the results with ode45, Explicit euler method is too uncomplicated to achieve a good approximation of the curve easily. It is necessary to take a rather large number of iterations to obtain convincing results, unlike ode45 which can give a good numerical approximation of the differential equation with little iterations.

We can compute the work precision for different tolerances and step sizes for the VanderPol problem for $\mu = 1.5$ and $\mu = 15$ for the Explicit Euler Method with adaptive step.

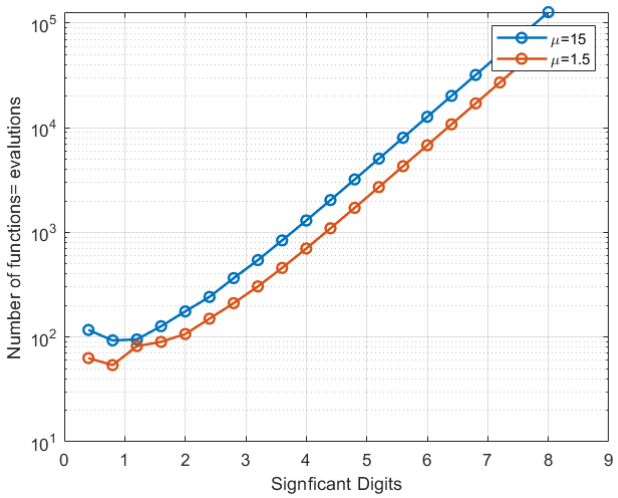

Figure 13: Work precision for the VanderPol problem for $\mu = 1.5$ and $\mu = 15$ for the Explicit Euler Method with adaptive step

It can be deduced here that as μ increases, the number of function evaluations increases. Moreover, for each Vanderpol problem, one can notice that the number of function evaluations evolves exponentially with the number of significant digits. We use the function *WorkPrecision2*.

```
function [T,X] = ExplicitEulerFixedStepSize(fun,t0,tN,N,x0,varargin)
% Compute step size and allocate memory
dt = (tN-t0)/N;
nx = size(x0,1);
X = zeros(nx,N+1);
T = zeros(1,N+1);


% Eulers Explicit Method
T(:,1) = t0;
X(:,1) = x0;
for k=1:N
    f = feval(fun,T(:,k),X(:,k),varargin{:});
    T(:,k+1) = T(:,k) + dt;
    X(:,k+1) = X(:,k) + f*dt;
end

% Form a nice table for the result
T = T';
X = X';
```

3 Implicit ODE solver

Given a differental equation $\forall t \in I, y'(t) = f(t, y(t))$ with I an interval, the purpose is to solve it using implicit Euler scheme, which is: for $i \in \{0, n-1\}$, $y_{i+1} = y_i + hf(t_{i+1}, y_{i+1})$.

So the algorithm is : Initialization of the step h the number of steps n, initials conditions t_0 and y_0. For $i \in \{0, n-1\}$, $t_{i+1} = ti + h and y_{i+1} = y_i + hf(t_{i+1}, y_{y_i i+1})$ which can be rearranged and we introduce the function R such as $R(y_{i+1}) = y_{i+1} - hf(y_{i+1}, t_{i+1}) - y_i = 0$. Then, we apply Newton's method in the implicit Euler method to find the root of R which is y_{i+1} and we itterate for each $i \in \{0, n-1\}$ Then, plot y datas versus t datas.

Algorithm 2: Implicit Euler method with fixed time step

Result: t and y

initialization : the step h , the number of steps n, initials conditions t_0 and y_0, the function $f(t, y(t))$ and the function $\frac{\partial f}{\partial y}$;

$i = 0$

while $i < n$ **do**

  $t_{i+1} = ti + h$;

  Apply the Newton's Method to find y_{i+1} on $R(y_{i+1}) = y_{i+1} - hf(y_{i+1}, t_{i+1}) - y_i$;

end

We can compute that on the Vander Pol problem with $n = 1000$ iterations ,$abstol = 1e{-}5$ and $restol = 1e{-}5$:

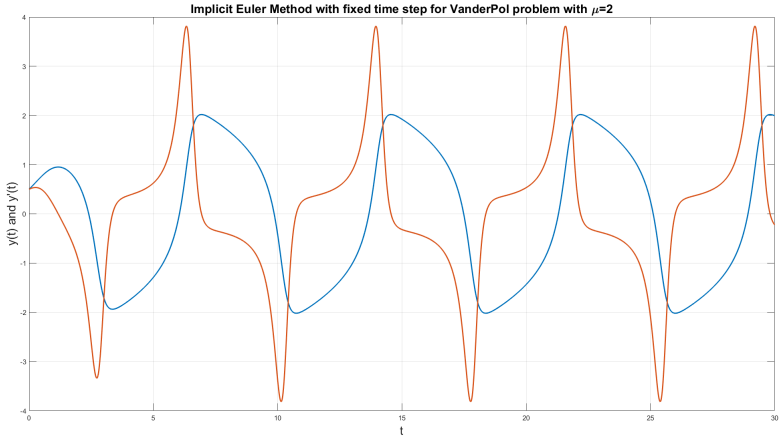

Figure 14: Implicit Euler method with fixed time step for Vander Pol Problem with $\mu = 2$

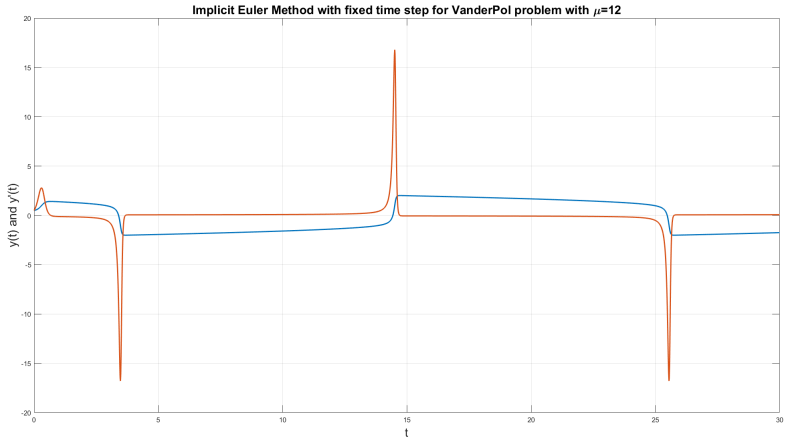

Figure 15: Implicit Euler method with fixed time step for Vander Pol Problem with $\mu = 12$

We use the function *ImplicitEulerFixedStepSizeTest* with *@VanderPolfunJac*.

We can compute that on the CSTR problem with $n = 1000$ iterations $abstol = 1e-5$ and $restol = 1e-5$:

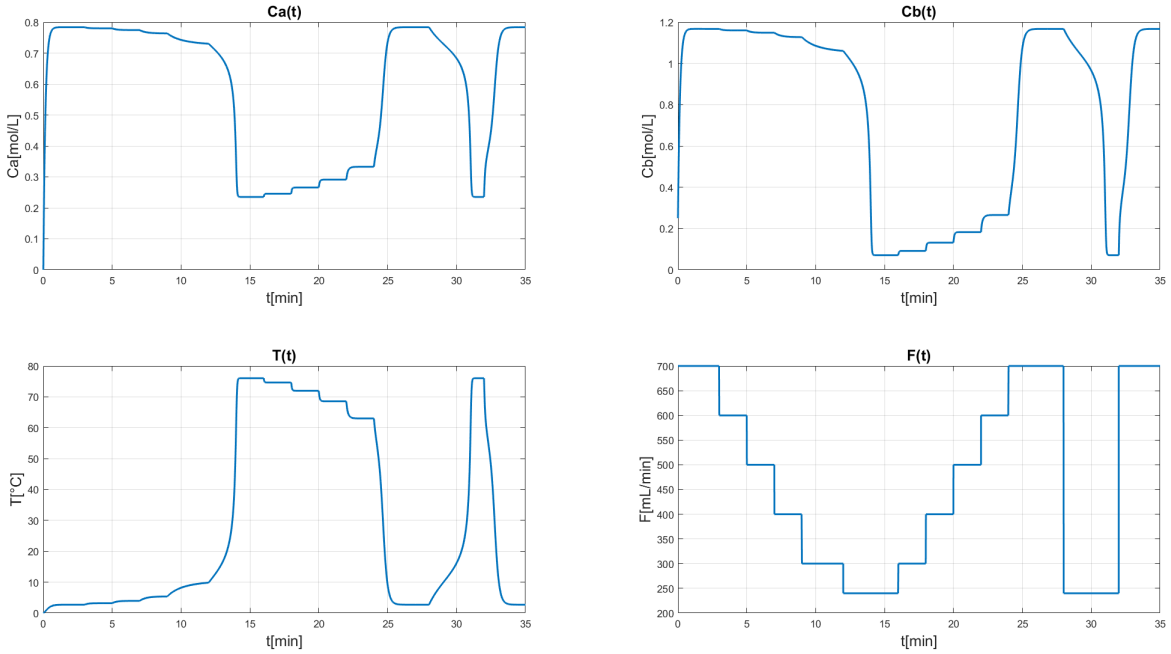

Figure 16: Implicit Euler method with fixed time step for CSTR in 3D

We use the function *ImplicitEulerFixedStepSizeTest* with *@CSTRR*

We could have displayed the implicit Euler method for the CSTR model in 1D but the picture is really the same. Indeed, one can notice that the only change between the 1D and 3D CSTR model is at the very beginning of the simulation.

Implicit methods are used because many problems arising in practice are stiff, for which the use of an explicit method requires impractically small time steps h to keep the error in the result bounded. For example, a VanderPol problem with $\mu = 1000$ is an stiff problem and is it almost impossible to get a good approximation with an explicit method.

To compare the results with ode45, Implicit euler method is too uncomplicated to achieve a good approximation of the curve easily. Nevertheless, it is a good method to give the trend of the curve. It converges more difficultly than the Euler Explicit method but achieves a better approximation.

```
function [T,X] = ImplicitEulerFixedStepSize(funJac,ta,tb,N,xa,varargin)

% Compute step size and allocate memory
dt = (tb-ta)/N;
nx = size(xa,1);
X = zeros(nx,N+1);
T = zeros(1,N+1);
tol = 1.0e-7;
maxit = 100;

% Eulers Implicit Method
T(:,1) = ta;
X(:,1) = xa;
for k=1:N
    f = feval(funJac,T(k),X(:,k));
    T(:,k+1) = T(:,k) + dt;
    xinit = X(:,k) + dt*f;
    X(:,k+1) = NewtonsMethodODE(funJac,T(:,k), X(:,k), dt, xinit, tol, maxit);

end

% Form a nice table for the result
T = T';
X = X';
```

4 Solvers for SDEs

A stochastic differential equations can be written like this :

$$dx(t) = f(t, x(t), p_f)dt + g(t, x(t), p_g)d\omega(t), \qquad d\omega(t) \sim \mathcal{N}(\mu, \sigma^2)$$

where $x \in \mathbb{R}^{n_x}$and ω is a stochastic variable with dimension n_ω .p_f and p_g are parameters for $f : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_{p_f}} \mapsto \mathbb{R}^{n_x}$ and $g : \mathbb{R} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_{p_g}} \mapsto \mathbb{R}^{n_x \times n_\omega}$. $d\omega(t)$ is called white noise.

For the VanderPol Problem, we use $g(t, x(t)) = \sigma$.

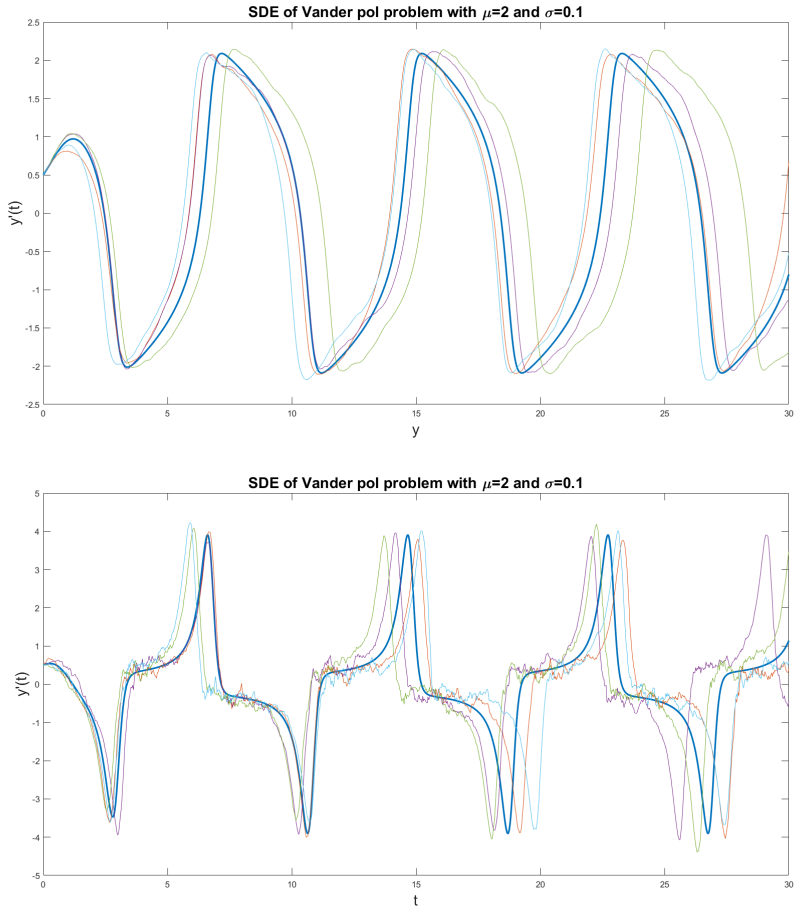

Figure 17: SDE with Explicit Euler Method for Vander Pol problem with $\mu = 2$ and $\sigma = 0.1$

We use the function *SDEVander*.

For the CSTR problem in 1D, we use $g(t, x(t)) = \frac{F}{V}\sigma_T$.

For the CSTR problem in 3D, we use $g(t, x(t)) = \begin{pmatrix} 0 \\ 0 \\ \frac{F}{V}\sigma_T \end{pmatrix}$

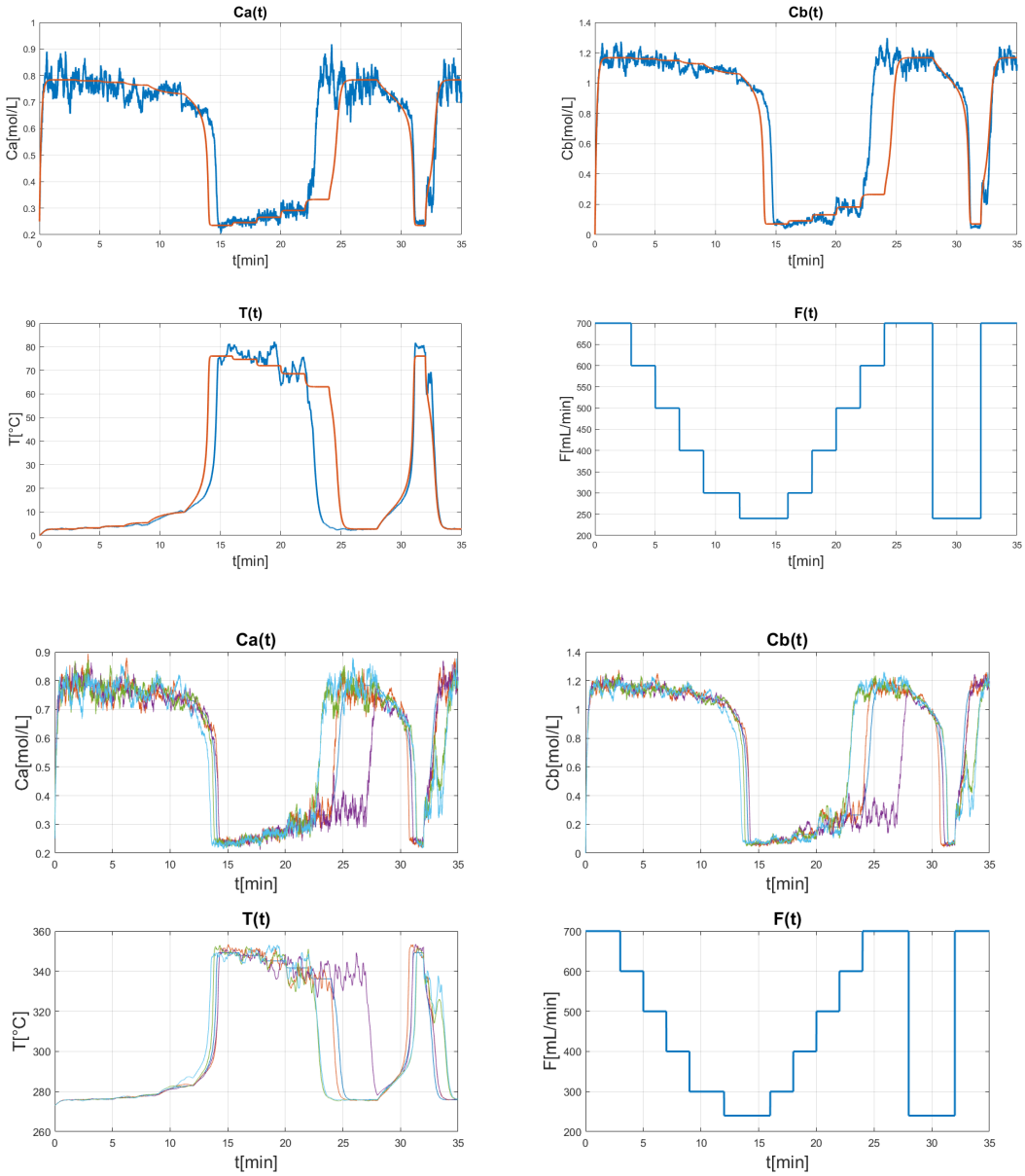

Figure 18: SDE with Explicit Euler Method for CSTR problem in 3D

We use the function *SDECSTR*

It can be observed that when simulating an EDS, time shifts can occur.

5 Classical Runge-Kutta method with fixed time step size

This method is the classical fourth order Runge-Kutta method for approximating the solution of the initial value problem $\forall t \in I, y'(t) = f(t, y(t))$ with I an interval; which evaluates the integrand, $f(t, y(t))$, four times per step.

We have so this algorithm :

Algorithm 3: Runge-Kutta RK4 Method Algorithm with fixed time step

Result: t and y

initialization : the step h , the number of steps n, initials conditions t_0, x_0 and the function $f(t, y(t))$;

$i = 0$

while $i < n$ **do**

  $t_{i+1} = ti + h$;

  $k_1 = hf(t_i, y_i)$;

  $k_2 = hf(t_i + \frac{h}{2}, y_i + \frac{k_1}{2})$ $k_3 = hf(t_i + \frac{h}{2}, y_i + \frac{k_2}{2})$;

  $k_4 = hf(t_i + h, y_i + k_3)$;

  $y_{i+1} = y_i + \frac{k_1+2k_2+2k_3+k_4}{6}$;

end

We can test it for test equation $\dot{x}(t) = \lambda x(t), x(0) = x_0$:

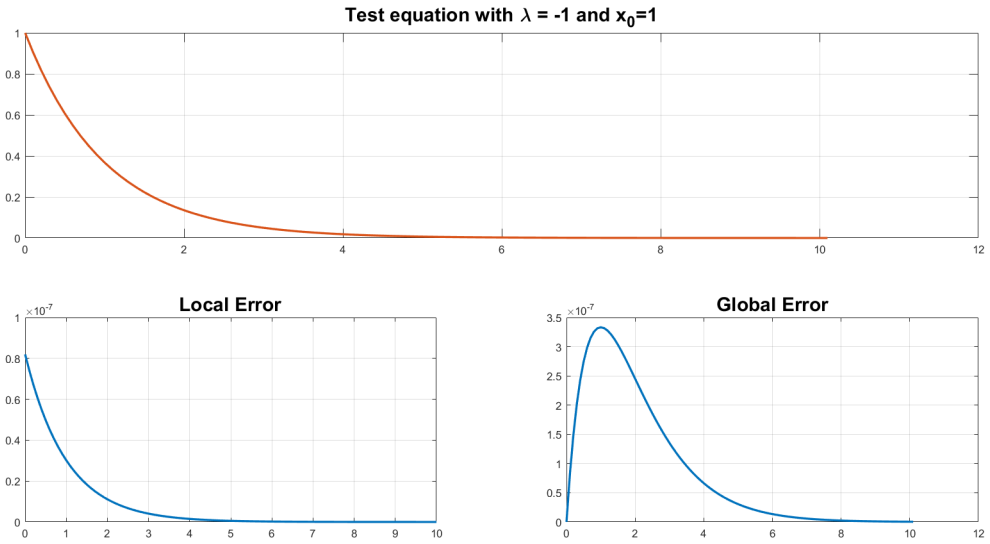

Figure 19: Runge-Kutta RK4 Method on the test equation, local error and global error

We use the function *ClassicalRungeKuttaFixedTimeTest* with *@TestExp* with =0.01.

To find the order of RK4, the global error for different values of h is computed and the order is 4for RK4 method's:

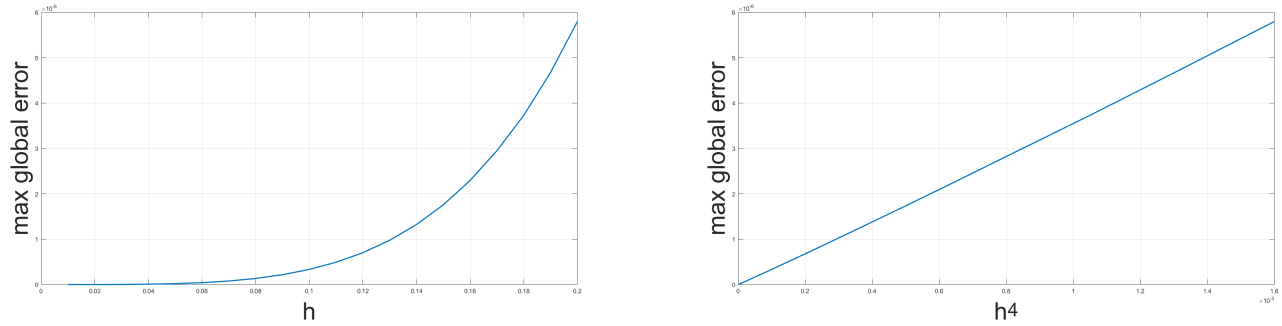

Table 2: Order of RK4 method with fixed step

We use the function *OrdreRK4* We can compute it on the VanderPol problem with $\mu = 1.5$ and $\mu = 15$ for $x_0 = [1.0; 1.0]$:

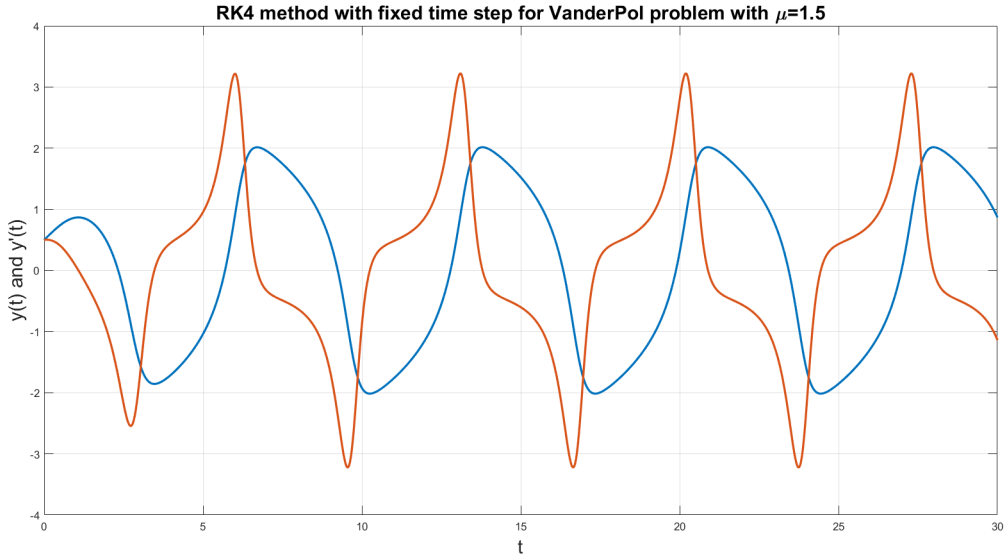

Figure 20: Runge-Kutta RK4 Method for Vander Pol problem with $\mu = 1.5$

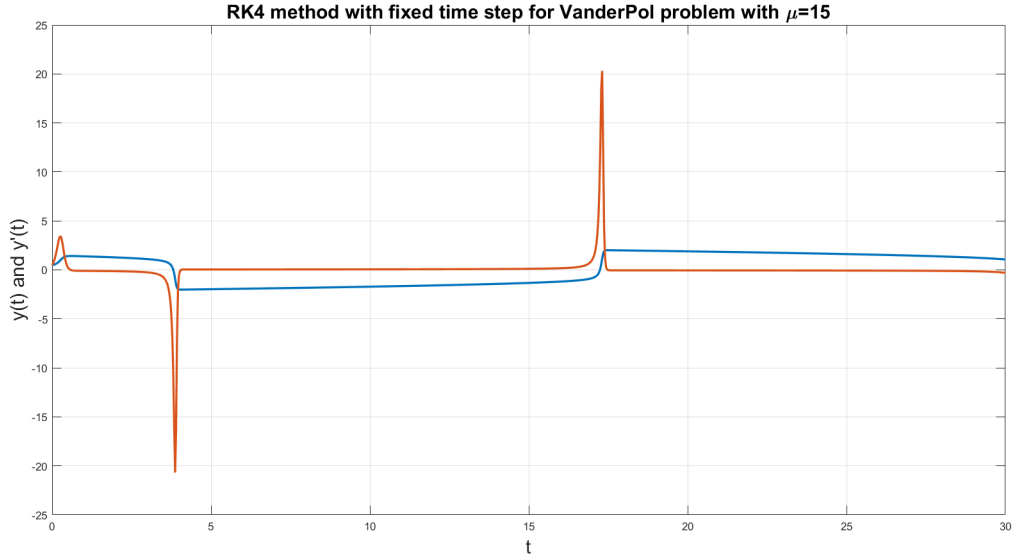

Figure 21: Runge-Kutta RK4 Method for Vander Pol problem with $\mu = 15$

We use the function *ClassicalRungeKuttaFixedTime* with *@VanderPol*with =0.01.

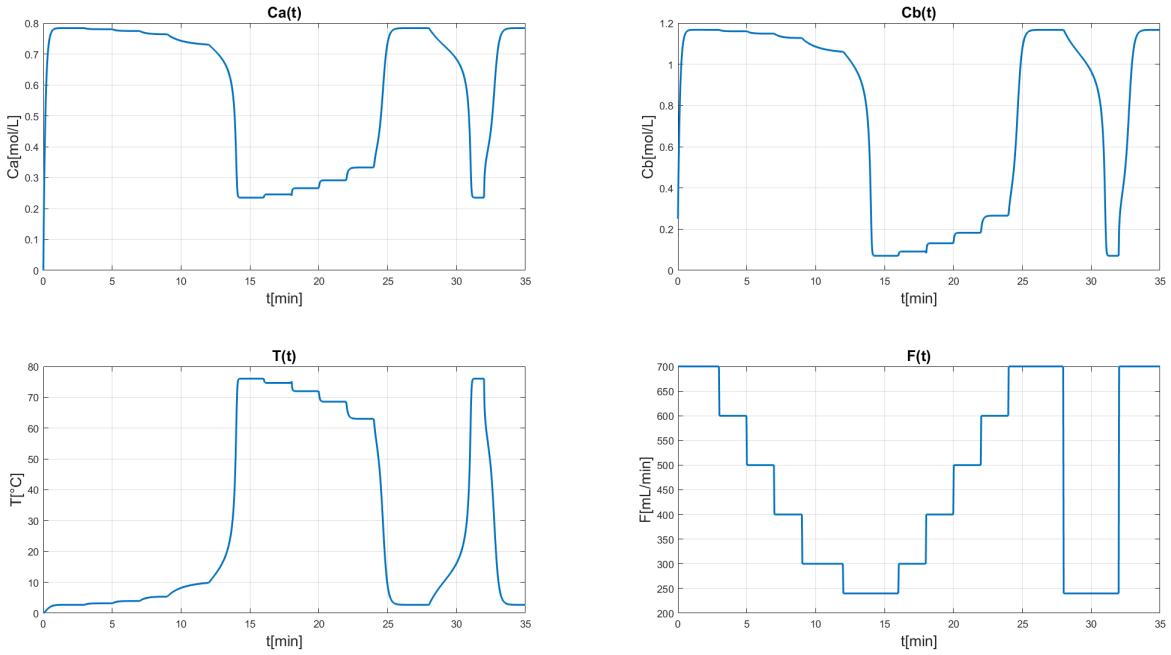

Figure 22: Runge-Kutta RK4 Method with fixed time step for CSTR in 3D

We use the function *ClassicalRungeKuttaFixedTime* with *@CSTR* with =0.01.

To compare the results with ode45, RK4 is a good method because it has the advantage of being simple to program and quite stable for the common functions of physics. In terms of numerical analysis, they have the great advantage of not requiring anything other than the knowledge of initial values. However, ode45 is still better than the classic Runge- Kutta method.

```
function [T,y]=ClassicalRungeKuttaFixedTime(fun,ti,tf,y0,h,varargin)

n=length(ti:h:tf)
T = ti;
y = y0;
for i = 1:n
    f=feval(fun,T(i),y(:,i));
    [t1,y1]=ClassicalRungeKuttaStep(fun,T(i),y(:,i),f,h,varargin{:});
    T=[T t1];
    y=[y y1];
end
```

6 Classical Runge-Kutta method with adaptive time step

This method is the fourth order Runge-Kutta method with adaptive step for approximating the solution of the initial value problem $\forall t \in I, y'(t) = f(t, y(t))$ with I an interval; which evaluates the integrand, $f(t, y(t))$, four times per step.

We have so this algorithm :

Algorithm 4: Runge-Kutta RK4 Method Algorithm with adaptive time step

Result: t and y

initialization : the step h , the number of steps n, initials conditions t_0, x_0 and the function $f(t, y(t))$;

$i = 0$

while $i < n$ **do**

  while $r \geq 1$ **do**

    Compute the error by step doubling e;

    if $r \leq 1$ **then**

      $t_{i+1} = ti + h$;

      $k_1 = hf(t_i, y_i)$;

      $k_2 = hf(t_i + \frac{h}{2}, y_i + \frac{k_1}{2})$ $k_3 = hf(t_i + \frac{h}{2}, y_i + \frac{k_2}{2})$;

      $k_4 = hf(t_i + h, y_i + k_3)$;

      $y_{i+1} = y_i + \frac{k_1+2k_2+2k_3+k_4}{6}$;

    else

      end

      r=max $\frac{|e_i|}{\max\{abstol, |x_i| reltol\}}$;

      $h = (\frac{\epsilon}{r})^{\frac{1}{5}}$

    end

  end

We can compute this classical RK4 method for the test equation with adaptive step :

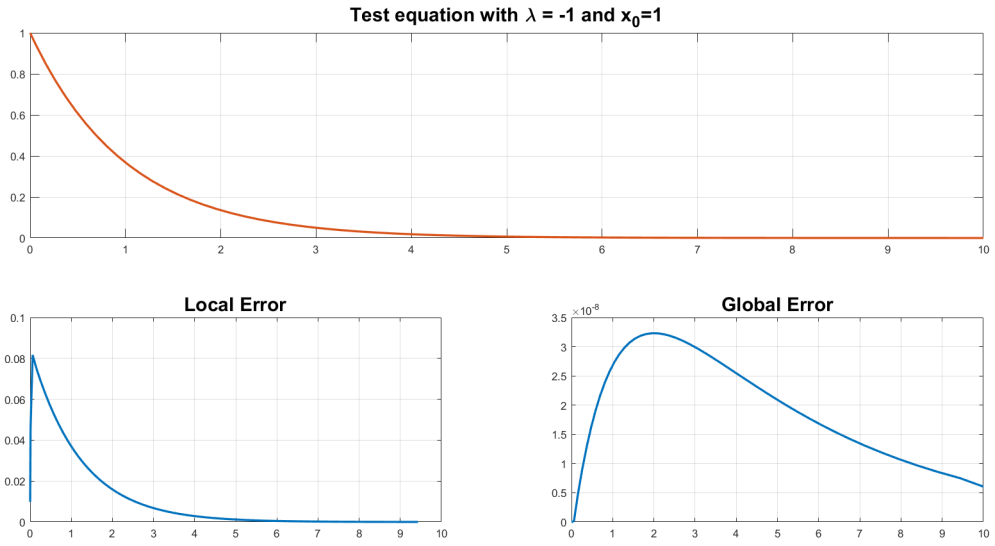

Figure 23: Runge-Kutta RK4 Method with adaptive step on the test equation, local error and global error

We use the function *ClassicalRungeKuttaAdaptiveStep* with *@Testexp* with To find the order of RK4, the

global error for different values of h is computed and the order is 5 for RK4 method's with adaptive step time:

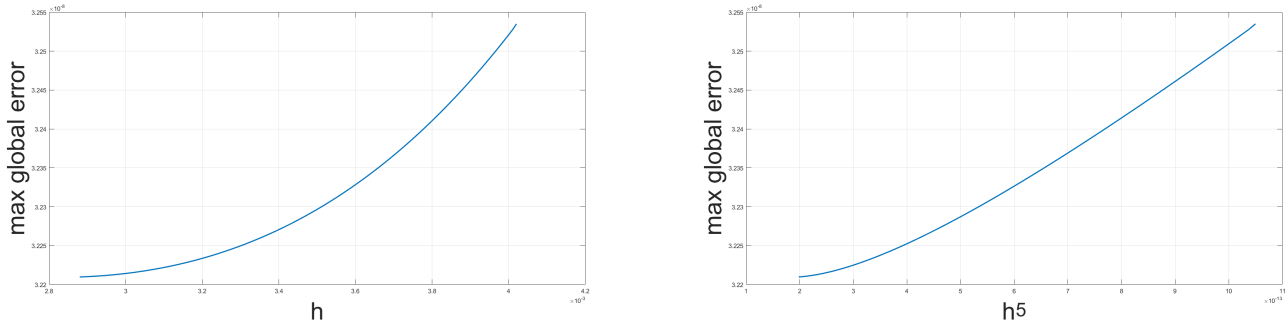

Table 3: Order of RK4 method with adaptive step

We use the function *OrdreRk4Adap*

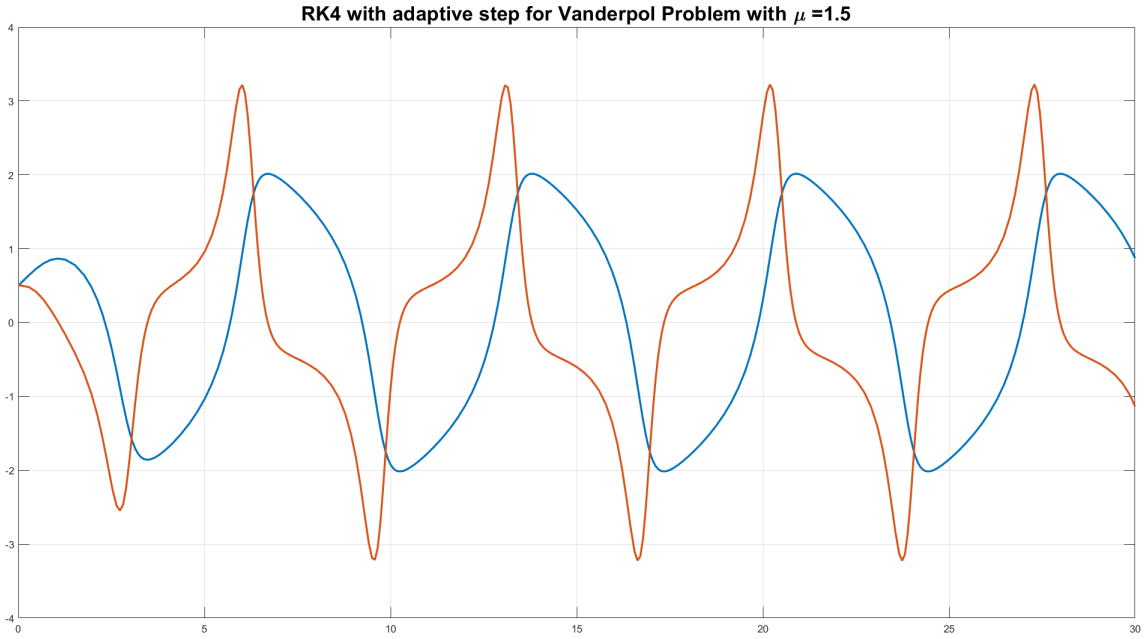

Figure 24: Runge-Kutta RK4 Method with adaptive time step for Vander Pol problem with $\mu = 1.5$

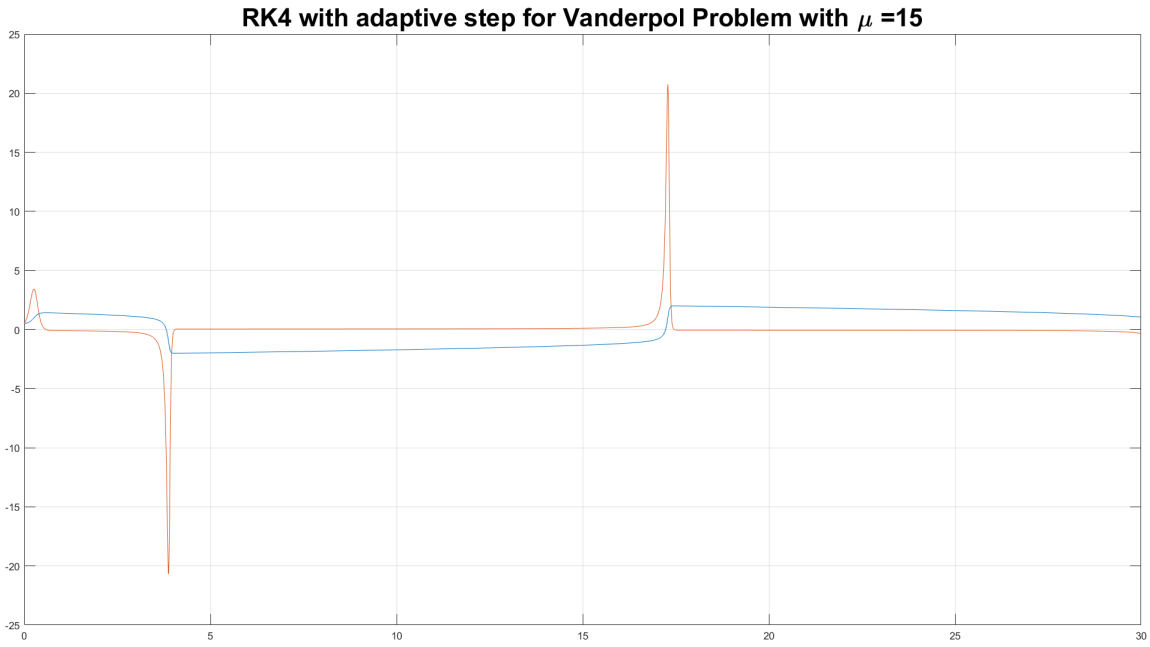

Figure 25: Runge-Kutta RK4 Method with adaptive time step for Vander Pol problem with $\mu = 15$

We use the function *ClassicalRungeKuttaAdaptiveStep* with *@VanderPol* with
With the Runge-Kutta RK4 Method with adaptive time step, I did not succeed in converging the CSTR problem. Using a lot of tolerance, the program runs to infinity and does not converge.

Nevertheless, this method works very well for the VanderPol problem. It is a more efficient method than ode45 because it converges better for this problem because it is of order 5. Moreover, it is very stable because it uses step doubling.

However, it is important to choose the toerances carefully so that they converge well. For the VanderPol problem, we take $abstol = 1e-4$ and $relstol = 1e-4$.

We use the function *ClassicalRungeKuttaAdaptiveStep* with *@CSTR*

```
function [T,X]=ClassicalRungeKuttaAdaptiveStep(fun,tspan,x0,h0,abstol,reltol,varargin)

epstol=0.8;
kpow=0.2;
facmin=0.1;
facmax=5;

t0=tspan(1);
tf=tspan(2);

t=t0;
h=h0;
x=x0;

T=t;
X=x';

while t<tf
    if(t+h>tf)
        h=tf-t
    end

    f=feval(fun,t,x,varargin{:});

    AcceptStep =false;
    while ¬AcceptStep
        [t1,x1]=ClassicalRungeKuttaStep(fun,t,x,f,h,varargin{:});

        hm=0.5*h;
        [tm,xm]=ClassicalRungeKuttaStep(fun,t,x,f,hm,varargin{:});
        fm=feval(fun,tm,xm,varargin{:});
        [t1hat,x1hat]=ClassicalRungeKuttaStep(fun,tm,xm,fm,hm,varargin{:});

        e=x1hat-x1;
        r=max(abs(e)./max(abstol,abs(x1hat).*reltol));



        AcceptStep=(r≤1.0);
        if AcceptStep
            t=t+h;
            x=x1hat;
            T=[T;t];
            X=[X;x'];
        end
        h=max(facmin,min((epstol/r)^kpow,facmax))*h;
    end
end
```

7 Dormand-Prince 5(4)

The Dormand–Prince 5(4) or DOPRI54 method is a member of the Runge-Kutta family of ODE solvers. It uses 6 functions evaluations to calculate fourth and fith order accurate solutions. Dormand–Prince is currently the default method in the ode45 solver for MATLAB. The butcher tableau is :

Table 2. Coefficients for RK5(4)7M

c_i	a_{ij}						$\hat{b}_i$	b_i
0							$\frac{35}{384}$	$\frac{5179}{57600}$
$\frac{1}{5}$	$\frac{1}{5}$						0	0
$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$					$\frac{500}{1113}$	$\frac{7571}{16695}$
$\frac{4}{5}$	$\frac{44}{45}$	$-\frac{56}{15}$	$\frac{32}{9}$				$\frac{125}{192}$	$\frac{393}{640}$
$\frac{8}{9}$	$\frac{19372}{6561}$	$-\frac{25360}{2187}$	$\frac{64448}{6561}$	$-\frac{212}{729}$			$-\frac{2187}{6784}$	$-\frac{92097}{339200}$
1	$\frac{9017}{3168}$	$-\frac{355}{33}$	$\frac{46732}{5247}$	$\frac{49}{176}$	$-\frac{5103}{18656}$		$\frac{11}{84}$	$\frac{187}{2100}$
1	$\frac{35}{384}$	0	$\frac{500}{1113}$	$\frac{125}{192}$	$-\frac{2187}{6784}$	$\frac{11}{84}$	0	$\frac{1}{40}$

Figure 26: Butcher Tableau for DOPRI 54 method

We use also the adaptive step time like Explicit Euler Method, Implicit Euler Method and RK4 method. We can compute the DOPRI54 method for the test equation with adaptative step :

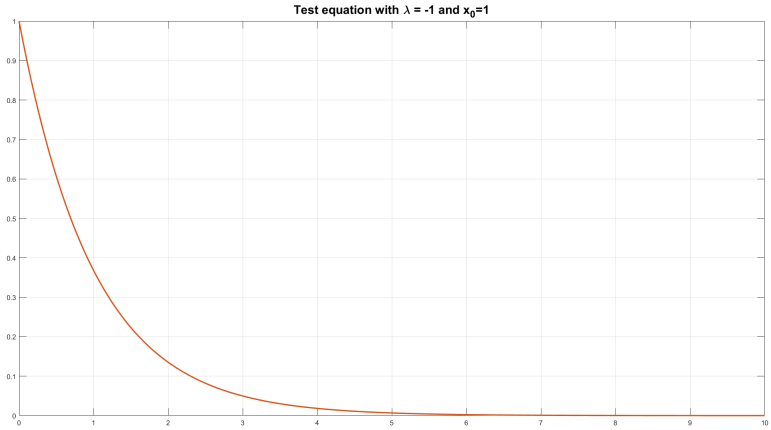

Figure 27: Runge-Kutta DOPRI54 Method with adaptative step on the test equation, local error and global error

We use the function *ERKSolverErrorEstimation* with the solver *'DOPRI54'* and with *@Testexp*

To find the order of RK4, the global error for different values of h is computed and the order is 5 for DOPRI54 method:

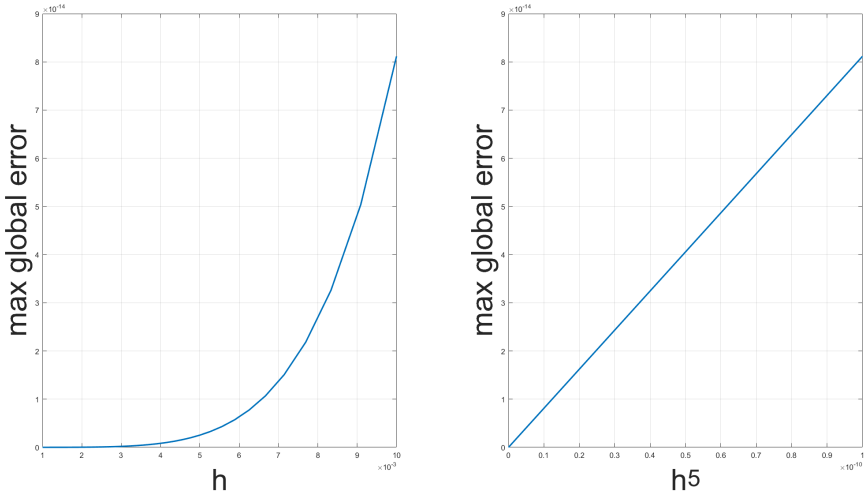

Figure 28: Order of DOPRI54 method

We use the function *OrderDOPRI54*
We can compute it for VanderPol problem with $\mu = 1.5$ and $\mu = 15$ and its error.

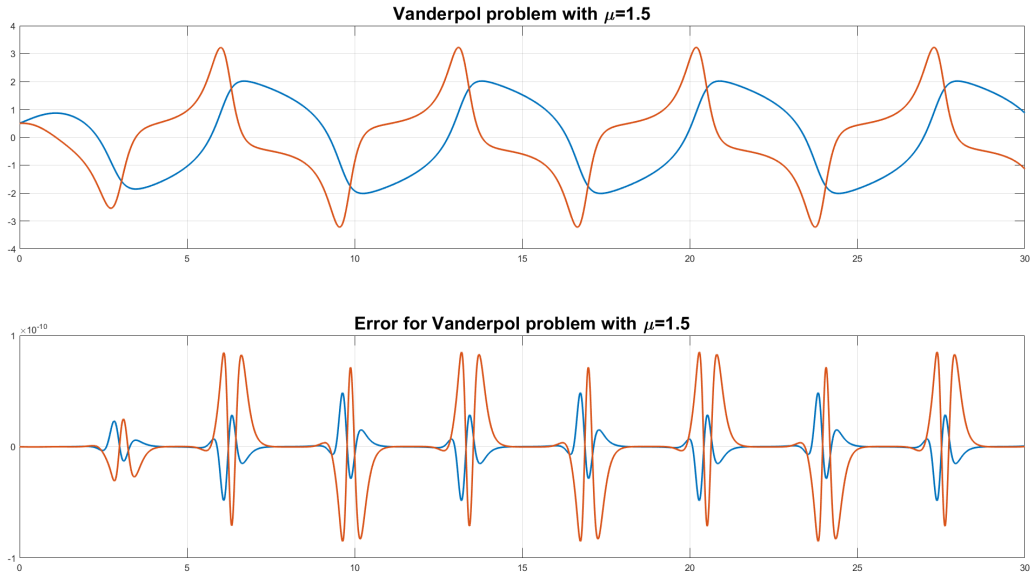

Figure 29: Runge-Kutta DOPRI54 Method VanderPol problem with $\mu = 1.5$

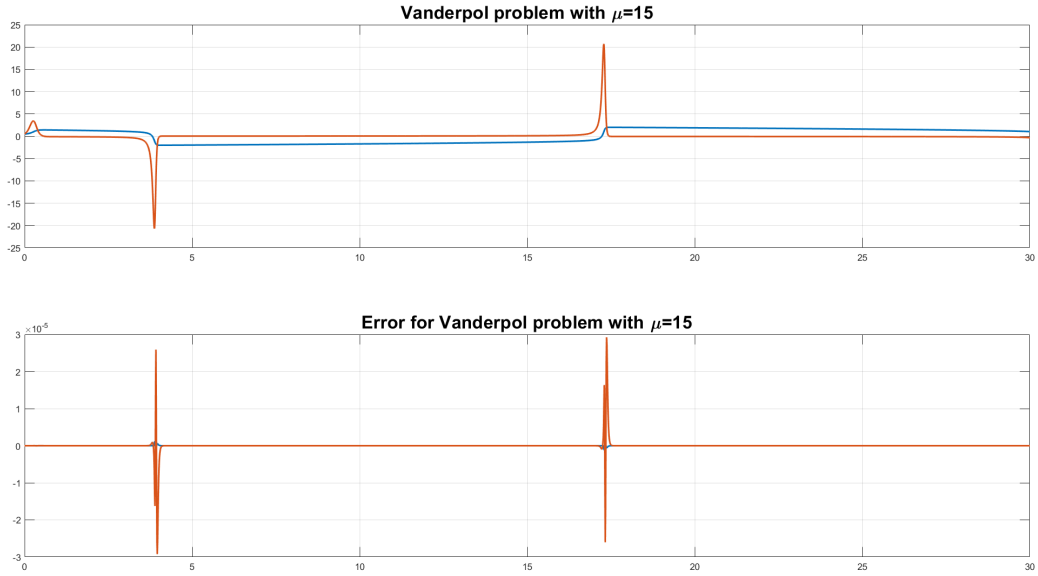

Figure 30: Runge-Kutta DOPRI54 MethodRunge-Kutta DOPRI54 Method VanderPol problem with $\mu = 15$

We use the function *ERKSolverErrorEstimation* with the solver *'DOPRI54'* and with *@VanderPol*
We can compute it for CSTR problem in 3D and its error:

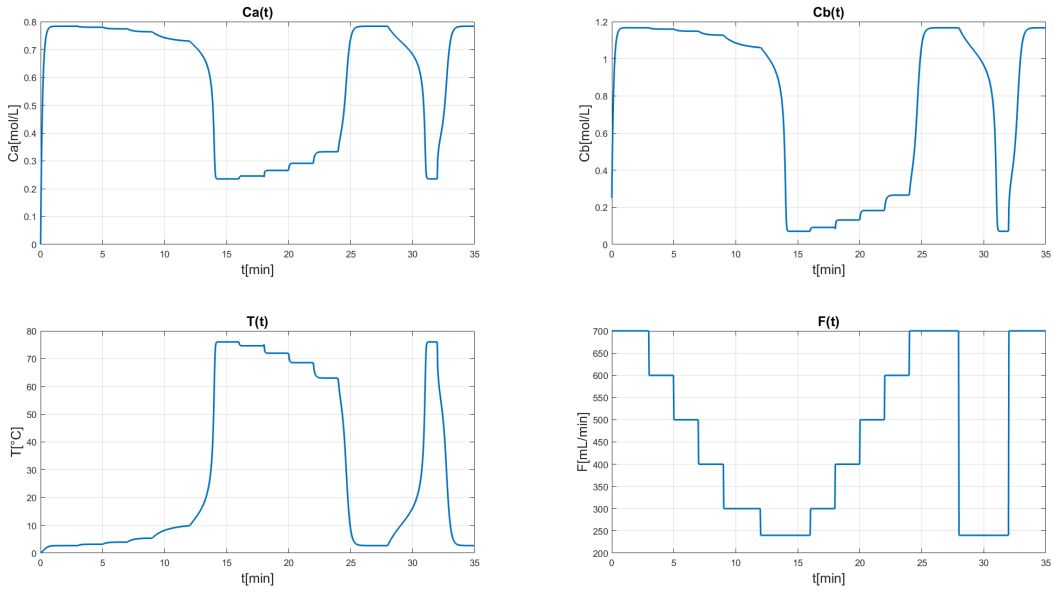

Figure 31: Runge-Kutta DOPRI54 Method for CSTR in 3D

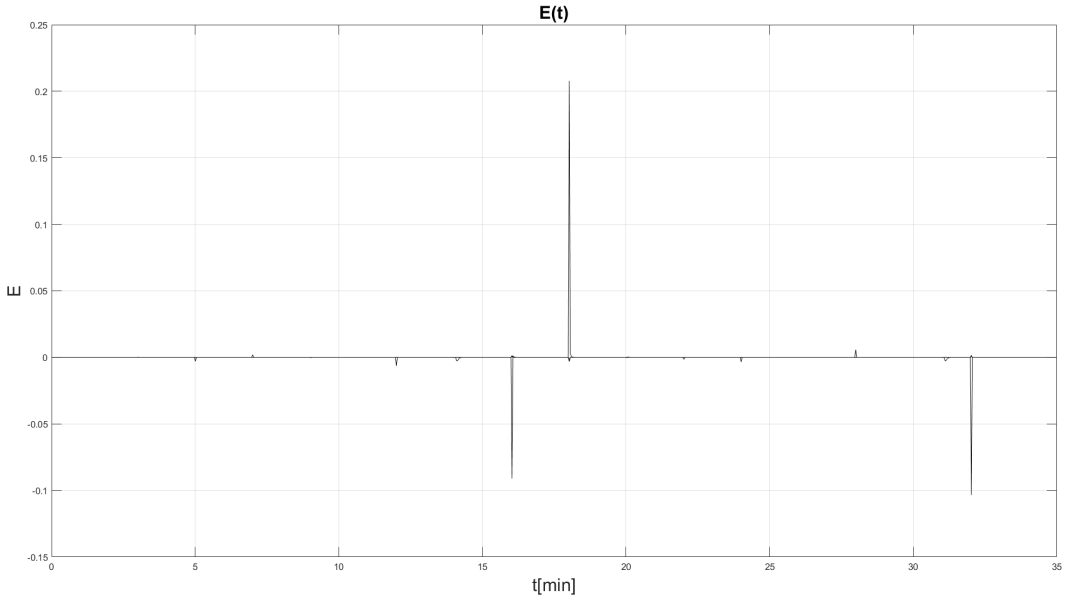

Figure 32: Error of Runge-Kutta DOPRI54 Method for CSTR in 3D

We use the function *ERKSolverErrorEstimation* with the solver *'DOPRI54'* and with *@CSTR*

With the DOPRI54,it can be noted taht ode45 is based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince pair. Indeed, we have the same solutions between ode45 solver and DOPRI54.

```
function [T,X]=RungeKuttaAdaptiveStepDOPRI54(fun,tspan,x0,h0,abstol,reltol,varargin)
solver = ERKSolverErrorEstimationParameters('DOPRI54');
epstol=0.8;
kpow=0.2;
facmin=0.1;
facmax=5;

t0=tspan(1);
tf=tspan(2);

t=t0;
h=h0;
x=x0;

T=t;
X=x';
while t<tf
    if(t+h>tf)
        h=tf-t;
    end
    AcceptStep =false;
    while ¬AcceptStep
        [t1,x1]=RungeKuttaStepDOPRI54(fun,t,x,solver,h,varargin{:});
        hm=0.5*h;
        [tm,xm]=RungeKuttaStepDOPRI54(fun,t,x,solver,hm,varargin{:});
        [t1hat,x1hat]=RungeKuttaStepDOPRI54(fun,tm,xm,solver,hm,varargin{:});
        e=x1hat-x1;
        r=max(abs(e)./max(abstol,abs(x1hat).*reltol));
        AcceptStep=(r≤1.0);
        if AcceptStep
            t=t+h;
            x=x1hat;
            T=[T;t];
            X=[X;x'];
        end
        h=max(facmin,min((epstol/r)^kpow,facmax))*h;
    end
end
```

8 ESDIRK23

ESDIRK 23 Method is an explicit singly diagonal implicit Runge-Kutta (ESDIRK) integration methods.It has basic order 3 and internal stage order of 2 The Butcher tableau is :

0	0		
2γ	γ	γ	
1	$\frac{1-\gamma}{2}$	$\frac{1-\gamma}{2}$	γ
x_{n+1}	$\frac{1-\gamma}{2}$	$\frac{1-\gamma}{2}$	γ
$\hat{x}_{n+1}$	$\frac{6\gamma-1}{12\gamma}$	$\frac{1}{12\gamma(1-2\gamma)}$	$\frac{1-3\gamma}{3(1-2\gamma)}$
e_{n+1}	$\frac{1-6\gamma^2}{12\gamma}$	$\frac{6\gamma(1-2\gamma)(1-\gamma)-1}{12\gamma(1-2\gamma)}$	$\frac{6\gamma(1-\gamma)-1}{3(1-2\gamma)}$

Figure 33: Butcher Tableau for ESDIRK23 method

Usually, $\gamma = \frac{2-\sqrt{2}}{2}$.
We can plot the stability region $\mathbb{S} = \{z \in \mathbb{C} : |R(z)| < 1\}$ for the ESDIRK 23 method :

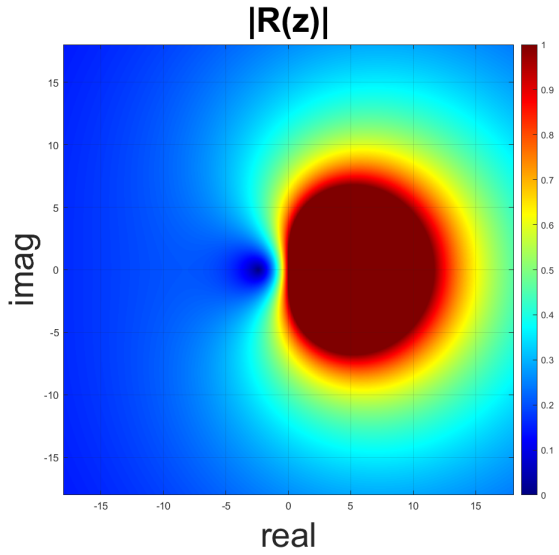

Figure 34: Stability Region of the ESDIRK23 Method

We use the function *Stability*.

Then, we need to talk about A-stability. A method is A-stable if its region of stability contains the entire left half-plane of $z = h\lambda$. This ESDIRK23 method is A-stable because $\forall z \in \mathbb{S} : \mathrm{Re}(z) < 0$. In the paper *A family of ESDIRK integration methods*, a 3-stage ESDIRK method of order 2 is A-stable if and only if $\frac{1}{4} \leq \gamma < \infty$.

Then, we need to define what is the L-stability. An integration method is said to be L-stable if its transfer function, $R(z)$, for the test equation is A-stable and in addition satisfies $\lim_{z\to-\infty} |R(z)| = 0$. When $z \to -\infty$, $|R(z)| \to 0$ and ESDIRK23 method is L-stable. The stability function of the 3-stage stiffly accurate ESDIRK integration scheme is :

$$R(z) = \frac{1 + (b_1 + b_2 - \gamma)z + (a_{21}b_2 - b_1\gamma)z^2}{(1 - \gamma z)^2}$$

In the paper *A family of ESDIRK integration methods*, a 3-stage ESDIRK method of order 2 is L-stable if and only if $\gamma = \frac{2\pm\sqrt{2}}{2}$.

Now we can compute it for Vanderpol Problem for $\mu = 1.5$ and $\mu = 15$ and $x_0 = [1.0, 1.0]$.

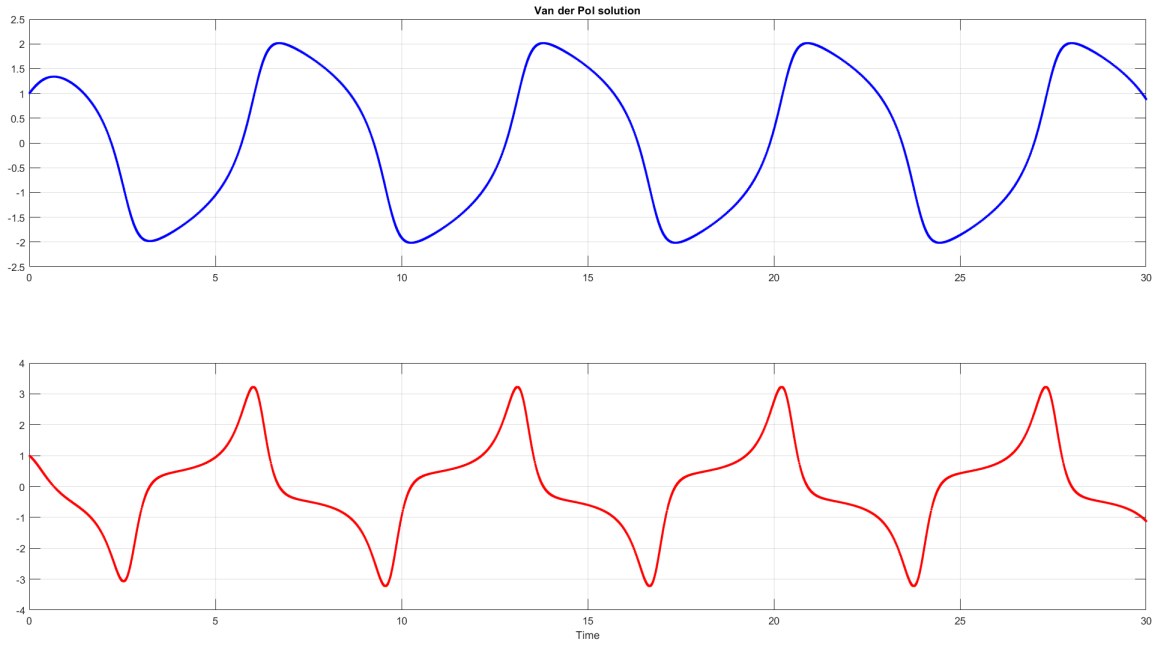

Figure 35: ESDIRK23 Method for Vanderpol problem with $\mu = 1.5$

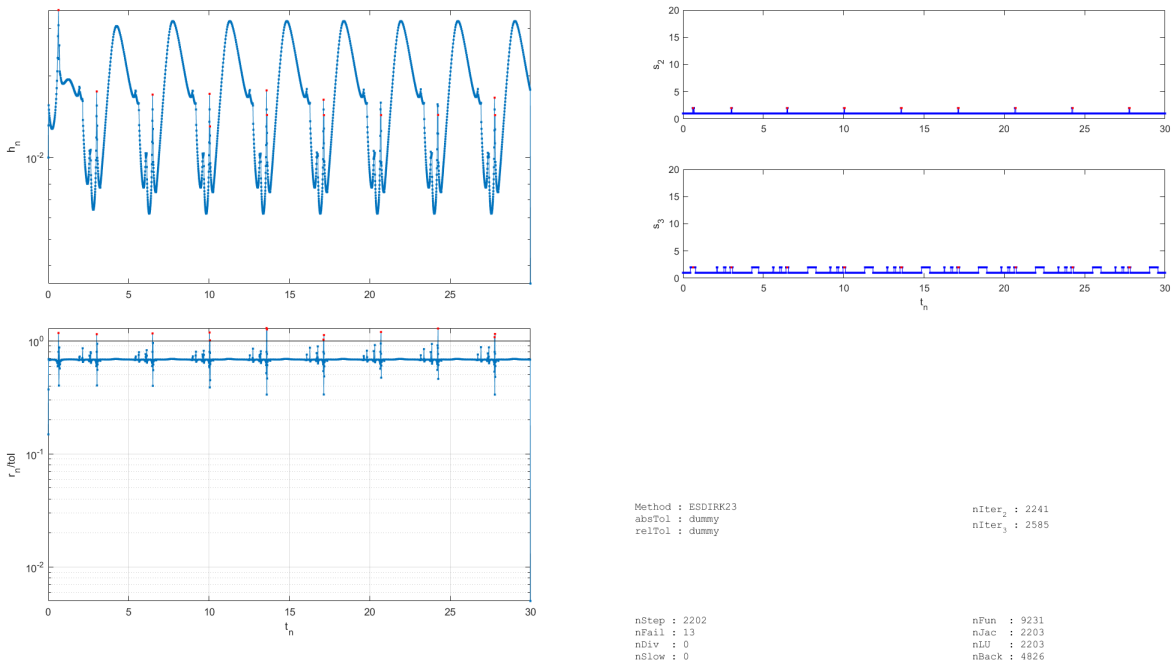

Figure 36: Error of ESDIRK23 Method for Vanderpol problem with $\mu = 1.5$

We use the function *BenchESDIRK*.

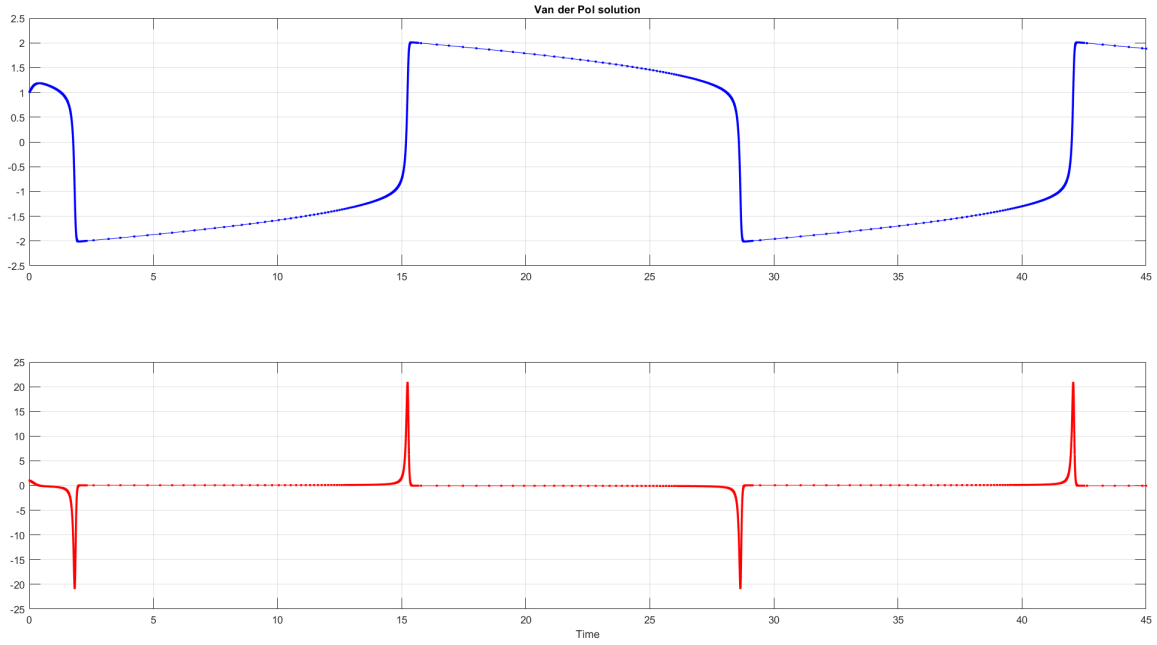

Figure 37: ESDIRK23 Method for Vanderpol problem with $\mu = 15$

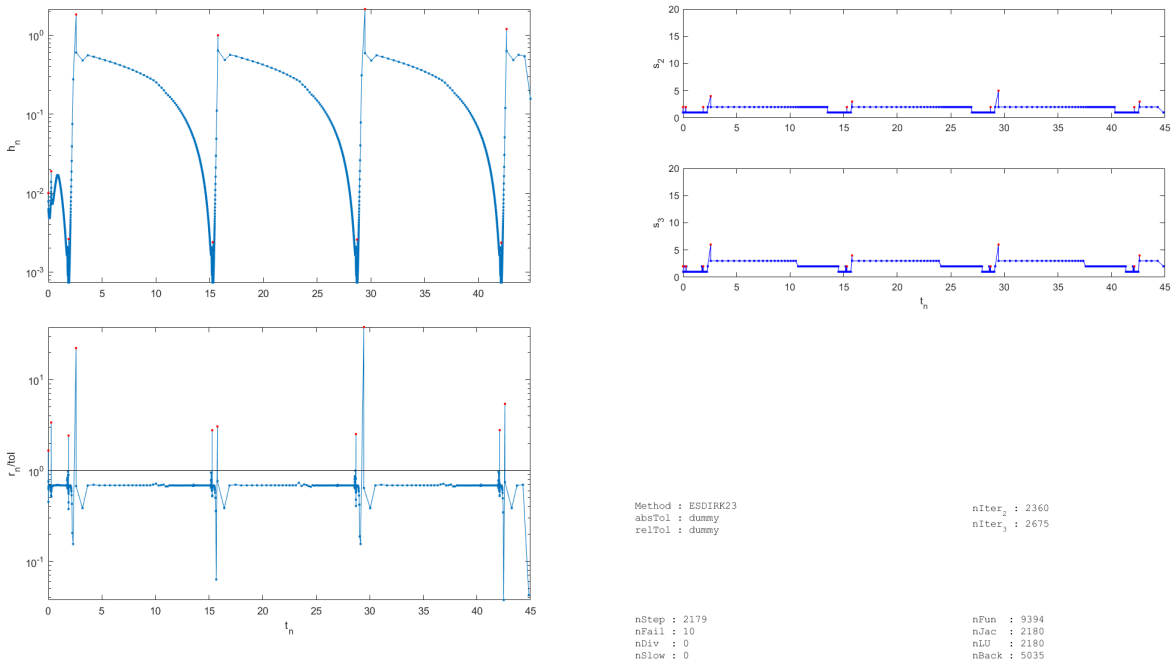

Figure 38: Error of ESDIRK23 Method for Vanderpol problem with $\mu = 15$

We use the function *BenchESDIRK*. We can compute it for the CSTR Problem in 3D.

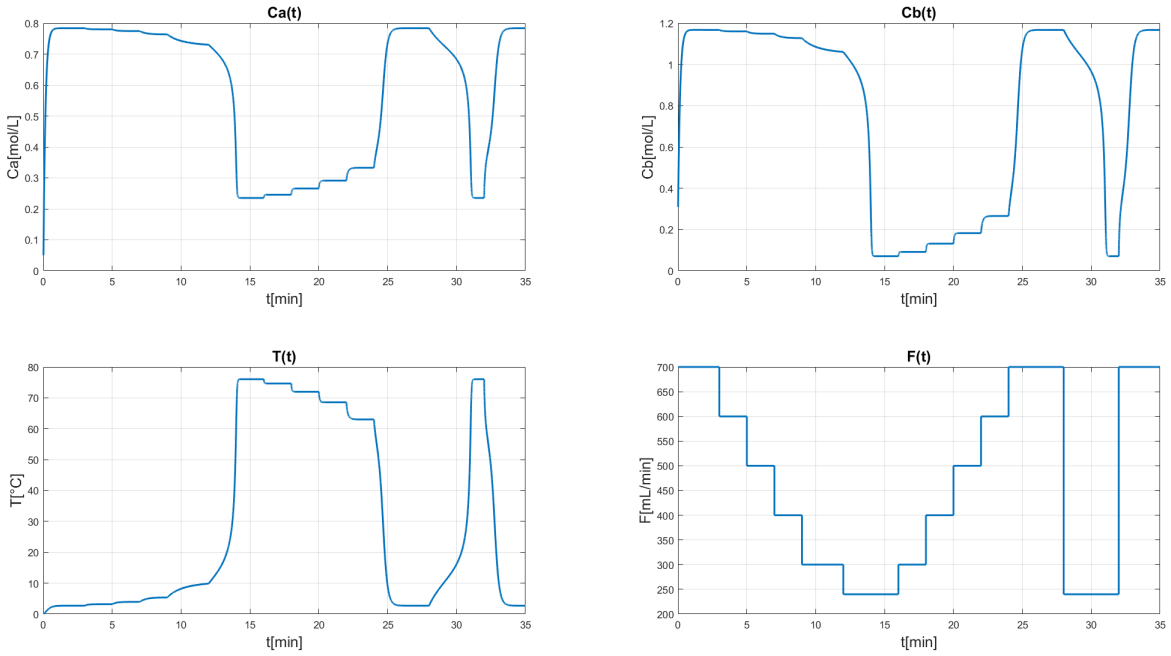

Figure 39: ESDIRK23 Method for CSTR problem in 3D

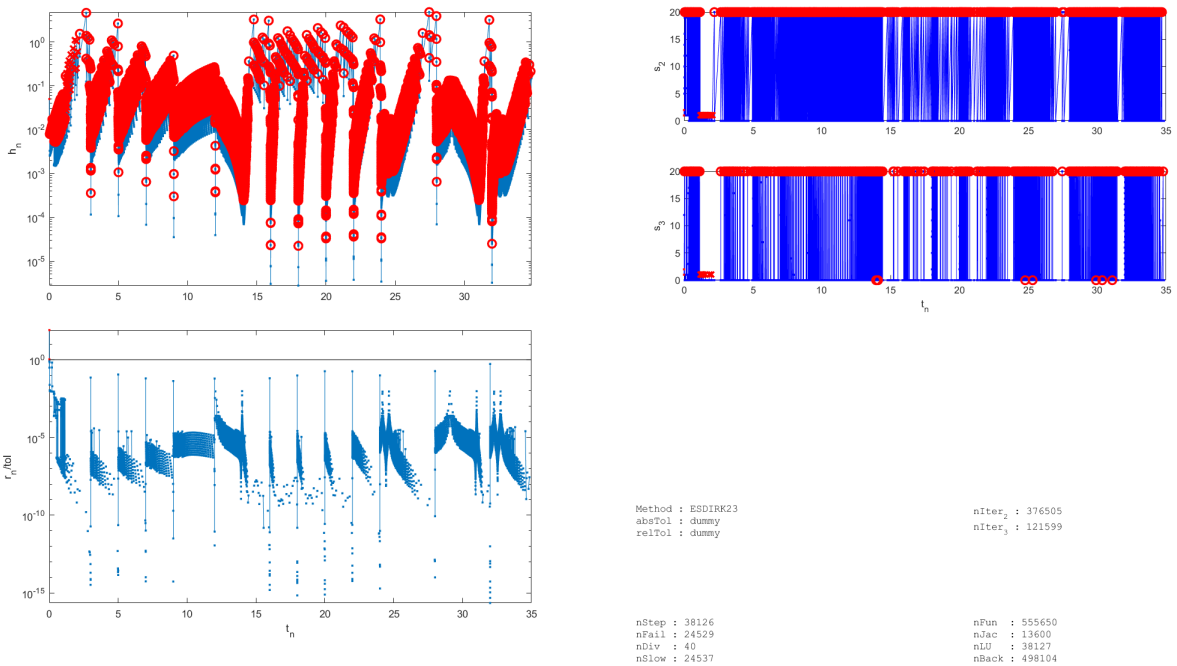

Figure 40: Error of ESDIRK23 Method for CSTR problem in 3D

We use the function *CSTRESDIRK*.

To compare ESDIRK23 with the other solvers in this exam problem, we can say that the solutions are roughly the same. Explicit Euler Method is worse than ESDIRK23 method but DOPRI54 is better than ESDIRK 23 method when we look at the errors. However, we can notice that ESDIRK method has less number of evaluation functions than the other solvers. It is therefore a very good method in terms of the number of calculations. ESDIRK23 method can therefore be used on small computers that can't do a lot of computation but can still get good results.

9 Discussion and Conclusions

In conclusion, this report has shown different methods to solve the VanderPol and CSTR problems.

Each method has an advantage and a disadvantage. Explicit methods have the particularity of being simple to code, having a good approximation of a good numerical solution but have stability problems as they are generally not A-Stable and L-Stable.

Implicit methods are a little more difficult to code because they have a method of finding the root of the equation: Newton's method. This has an advantage and a disadvantage. It can happen that the method does not converge but when it does, we have better results than the explicit methods. Moreover, it is necessary to know the Jacobian matrix to implement an implicit method.

There are methods that achieve as good or better solutions than the MATLAB optimised solver such as RK4 with adaptive step, DOPRI54 and ESDIRK23.