

TP N° 7 : Résolution des équations différentielles

Dans ce TP, nous allons présenter la fonction **ode23** comme une de la famille de fonctions **ODE solver** qui permettent de résoudre les équations différentielles ordinaires. La fonction **ode23()** est une fonction de MATLAB pour résoudre numériquement des équations différentielles ordinaires (EDO) d'ordre supérieur ou égal à 1. Elle utilise une méthode appelée méthode de Dormand-Prince qui est un algorithme de Runge-Kutta d'ordre 2-3.

Solver	Problem Type	Accuracy	When to Use
ode45	Nonstiff	Medium	Most of the time. ode45 should be the first solver you try.
ode23		Low	ode23 can be more efficient than ode45 at problems with crude tolerances, or in the presence of moderate stiffness.
ode113		Low to High	ode113 can be more efficient than ode45 at problems with stringent error tolerances, or when the ODE function is expensive to evaluate.
ode78		High	ode78 can be more efficient than ode45 at problems with smooth solutions that have high accuracy requirements.
ode89		High	ode89 can be more efficient than ode78 on very smooth problems, when integrating over long time intervals, or when tolerances are especially tight.
ode15s	Stiff	Low to Medium	Try ode15s when ode45 fails or is inefficient and you suspect that the problem is stiff. Also use ode15s when solving differential algebraic equations (DAEs).
ode23s		Low	ode23s can be more efficient than ode15s at problems with crude error tolerances. It can solve some stiff problems for which ode15s is not effective. ode23s computes the Jacobian in each step, so it is beneficial to provide the Jacobian via odeset to maximize efficiency and accuracy. If there is a mass matrix, it must be constant.
ode23t		Low	Use ode23t if the problem is only moderately stiff and you need a solution without numerical damping. ode23t can solve differential algebraic equations (DAEs).
ode23tb		Low	Like ode23s, the ode23tb solver might be more efficient than ode15s at problems with crude error tolerances.
ode15i	Fully implicit	Low	Use ode15i for fully implicit problems $f(t,y,y') = 0$ and for differential algebraic equations (DAEs) of index 1.

Exemple 1 : équation différentielle d'ordre 1

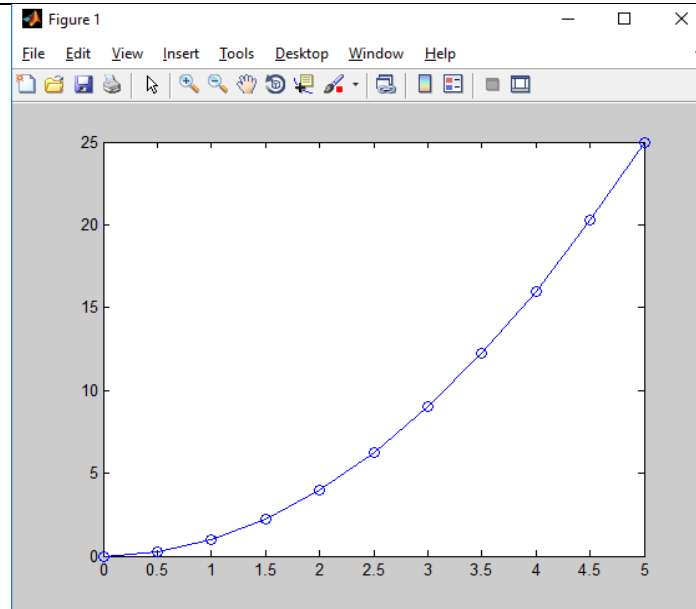
Résolution de l'équation différentielle $y' = 2t$ dans l'intervalle **[0 5]** avec une condition initiale **y(0) = 5**.

```
>> tspan = [0 5] ; % définir l'intervalle
>> y0 = 5 ; % définir la condition initiale
>> [t,y] = ode23(@(t,y) 2*t, tspan, y0) % resolution de l eq avec la fonction ode23
t =
    0
    0.5000
    1.0000
    1.5000
    2.0000
    2.5000
    3.0000
    3.5000
    4.0000
```

```

4.5000
5.0000
y =
    0
    0.2500
    1.0000
    2.2500
    4.0000
    6.2500
    9.0000
   12.2500
   16.0000
   20.2500
   25.0000
>> plot(t,y)

```



Exemple 2 : équation différentielle d'ordre 1

Résolution de l'équation différentielle :

$$y' = \frac{-ty}{\sqrt{2-y^2}} \quad ; \quad y(0) = 1 \quad ; \quad t \in [0, 5].$$

Pour cela nous allons créer une fonction matlab nommée fun1.m

```

function f=fun1(t,y)
f=-t*y/sqrt(2-y^2);
end

```

Ensuite, nous allons utiliser la fonction ode23 :

```

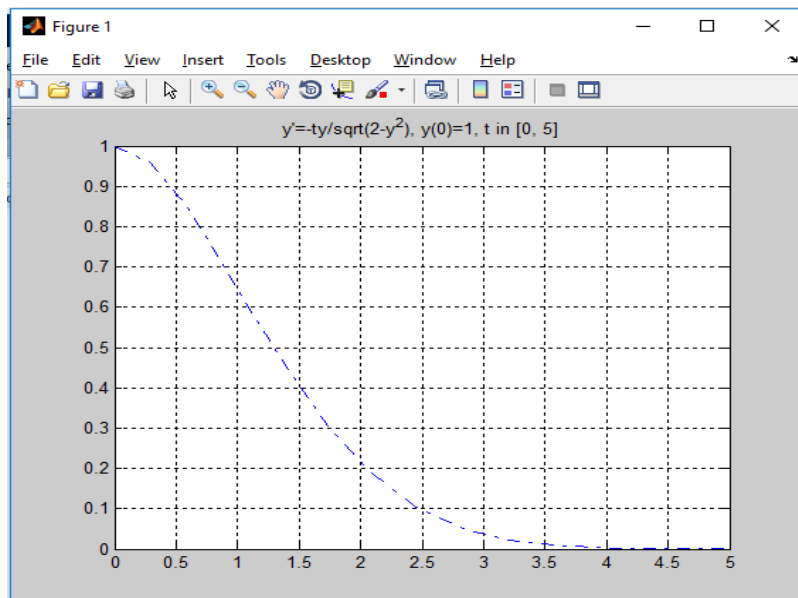
>> [t y]=ode23('fun1',[0 5],1)
t =
    0
    0.2925
    0.5849
    0.8501
    1.1251
    1.4186
    1.7311
    2.0656
    2.4351
    2.8478
    3.2066

```

```

3.4566
3.7066
3.9152
4.0867
4.2353
4.3710
4.4988
4.6234
4.7611
4.9191
5.0000
y =
1.0000
0.9590
0.8523
0.7249
0.5838
0.4398
0.3065
0.1939
0.1068
0.0483
0.0219
0.0121
0.0063
0.0036
0.0022
0.0014
0.0009
0.0006
0.0004
0.0003
0.0002
0.0001
>> plot(t,y,'-.')
>> title('y'=-ty/sqrt(2-y^2), y(0)=1, t in [0, 5]')
>> grid
>> axis([0 5 0 1])

```



Exemple 3 : Système d'équations différentielles

Résolution le système d'équations différentielles suivant :

$$\begin{cases} y_1' = 2y_1 + y_2 + 5y_3 + e^{-2t} \\ y_2' = -3y_1 - 2y_2 - 8y_3 + 2e^{-2t} - \cos(3t) \\ y_3' = 3y_1 + 3y_2 + 2y_3 + \cos(3t) \end{cases} \quad \text{avec} \quad \begin{cases} y_1(0) = 1 \\ y_2(0) = -1 \\ y_3(0) = 0 \end{cases} \quad \text{et } t \in [0, \frac{\pi}{2}].$$

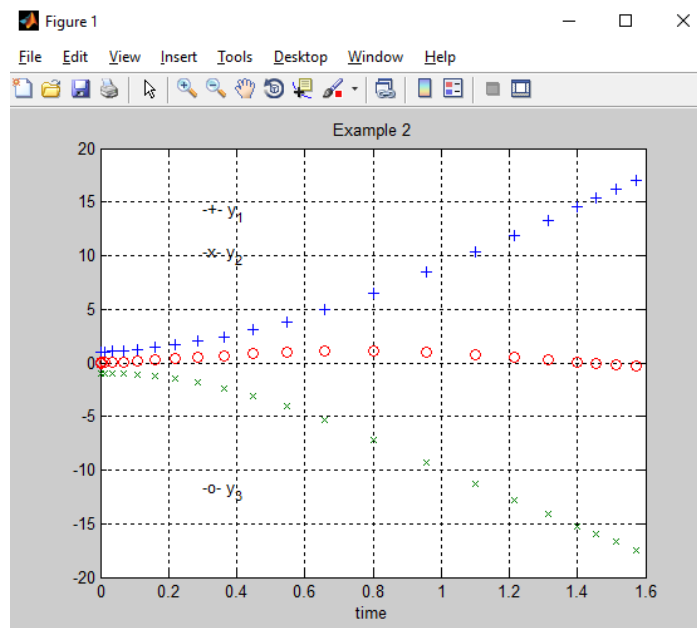
Pour cela nous allons créer une fonction matlab nommée **funsys.m**

```
function Fv=funsys(t,Y)
Fv(1,1)=2*Y(1)+Y(2)+5*Y(3)+exp(-2*t);
Fv(2,1)=-3*Y(1)-2*Y(2)-8*Y(3)+2*exp(-2*t)-
cos(3*t);
Fv(3,1)=3*Y(1)+3*Y(2)+2*Y(3)+cos(3*t);
end
```

Ensuite, nous exécutons les commandes :

```
>> [tv,Yv]=ode23('funsys',[0 pi/2],[1;-1;0]);
>> plot(tv,Yv(:,1),'+',tv,Yv(:,2),'x',tv,Yv(:,3),'o')
>> hold
Current plot held
>> grid
>> title('Example 2')
>> text(0.3,14,'-+- y_1')
>> text(0.3,10,'-x- y_2')
>> text(0.3,-12,'-o- y_3')
>> xlabel('time')
>> hold off
```

Nous obtenons le résultat suivant :



Exemple 4 : équation différentielle d'ordre 2

Résolution le système d'équations différentielles suivant :

$$y'' + 2y' + y = 2e^{-x}$$

Pour résoudre cette équation différentielle d'ordre 2, nous allons tout d'abord la réécrire sous forme d'un système de deux équations différentielles d'ordre 1 :

$$\text{Pour cela posons : } \begin{cases} y_2 = y \\ y_1 = y' \end{cases} ; \quad \text{alors, l'équation devient : } \begin{cases} y_2' = y_1 \\ y_1' = -2y_1 - y_2 + 2e^{-x} \end{cases}$$

On tombe donc dans le cas d'un système d'équations différentielles d'ordre 1, nous allons le résoudre de la même façon présentée dans l'exemple 3

```
function dy = eqdiff2(t,y)
    dy(1,:) = -2.*y(1)-y(2)+2.*exp(-t);
    dy(2,:) = y(1);
end
```

puis on tape sur la ligne de commande :

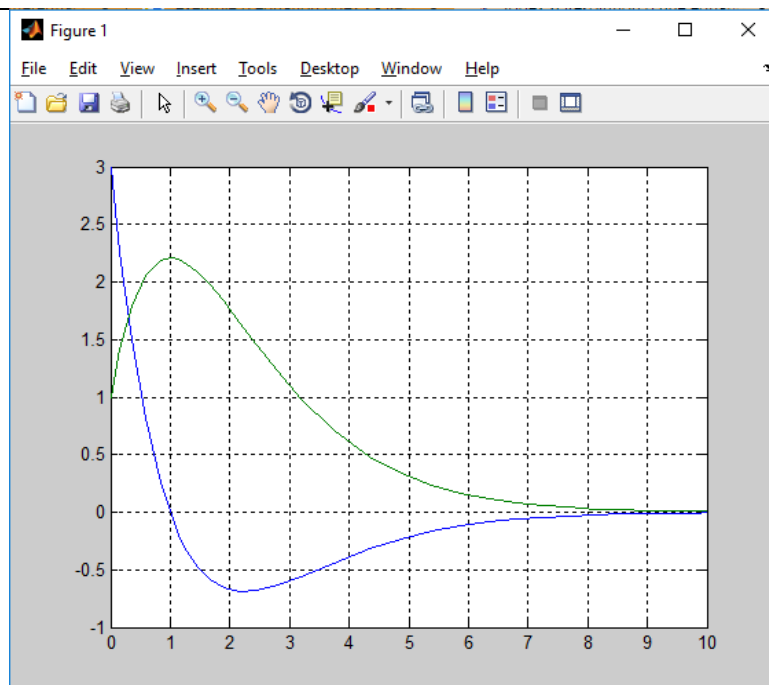
```
>> [t,y]=ode23('eqdiff2',[0 10],[3 1])
t =
    0
    0.0267
    0.1600
    0.3599
    0.5887
    0.8411
    0.9938
    1.1465
    1.2848
    1.4563
    1.6593
    1.8924
    2.1564
    2.4543
    2.7947
    3.2016
    3.7507
    4.3445
    4.8748
    5.3832
    5.8714
    6.3438
    6.8035
    7.2528
    7.6935
    8.1268
    8.5539
    8.9754
    9.3921
    9.8046
    10.0000

y =
    3.0000    1.0000
    2.8684    1.0782
    2.2619    1.4194
    1.5001    1.7930
    0.8185    2.0552
    0.2619    2.1884
    0.0079    2.2084
   -0.1942    2.1936
   -0.3387    2.1564
   -0.4750    2.0861
   -0.5854    1.9778
   -0.6590    1.8320
   -0.6912    1.6531
```

-0.6828	1.4479
-0.6372	1.2229
-0.5574	0.9801
-0.4391	0.7081
-0.3212	0.4844
-0.2344	0.3382
-0.1694	0.2363
-0.1220	0.1657
-0.0877	0.1165
-0.0630	0.0822
-0.0452	0.0580
-0.0324	0.0411
-0.0232	0.0291
-0.0167	0.0206
-0.0119	0.0147
-0.0086	0.0104
-0.0061	0.0074
-0.0052	0.0063

Pour tracer les fonctions $y(t)$ et $y'(t)$ on tape la commande :

```
>> plot(t,y)
>> grid
>>
```



NB : nous pouvons généraliser cette astuce pour résoudre les équations différentielles d'ordre supérieur.