EPFL

Programmation pour Ingénieur

Matlab IV

ME 3e semestre

rev. 2025.r1

Christophe Salzmann

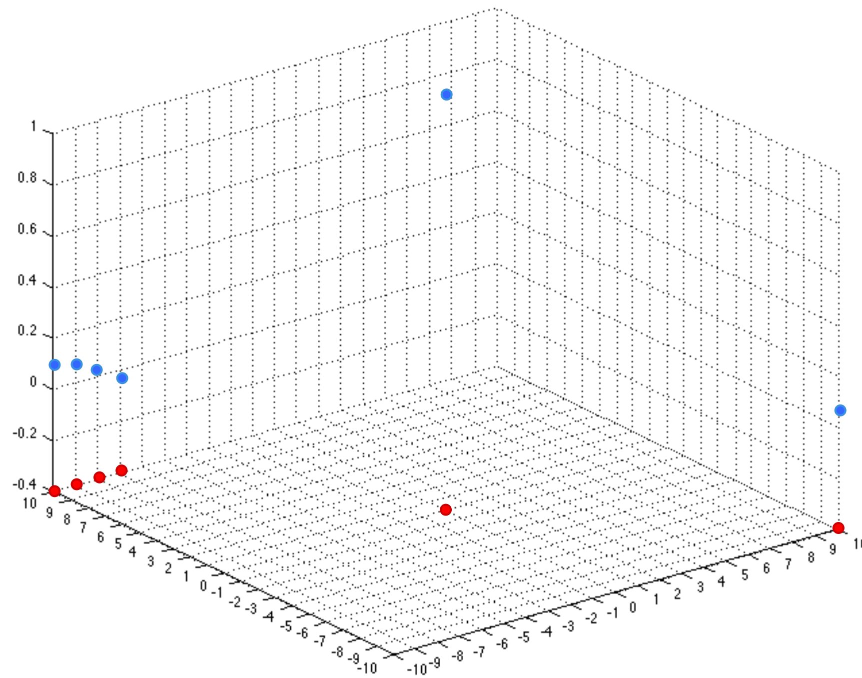
Laboratoire d'Automatique

Photo Martin Klimas

Recap

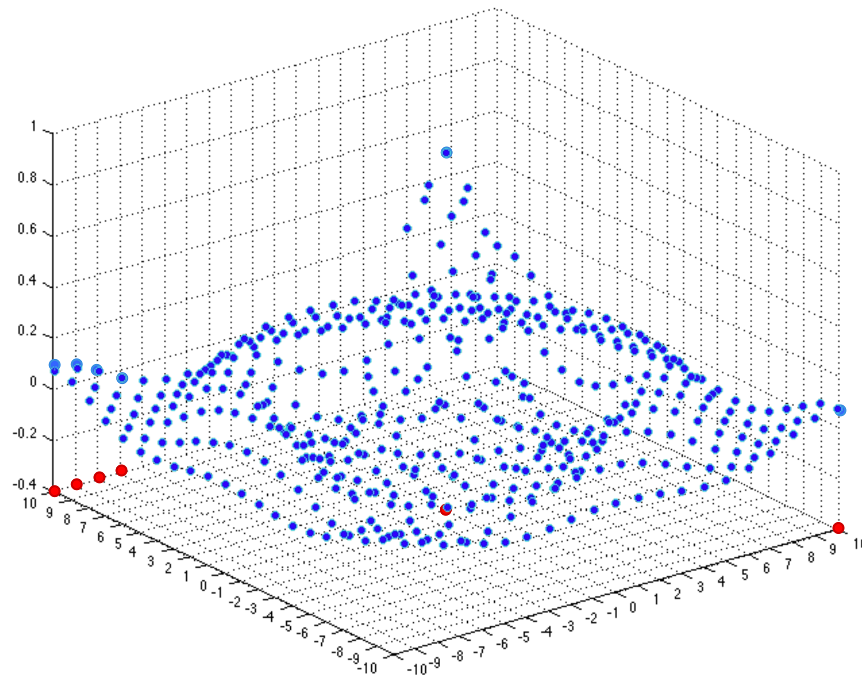
- Logical array
- Cell array
- 3D surface (meshgrid)
- 2D and 3D parametric plot
- Interpolation
- Fit

Surface 3D



```
for l=-10:10
  for c=-10:10
      R = sqrt(c^2+l^2)+eps;
      Z = sin(R)/R;
  end
end
```

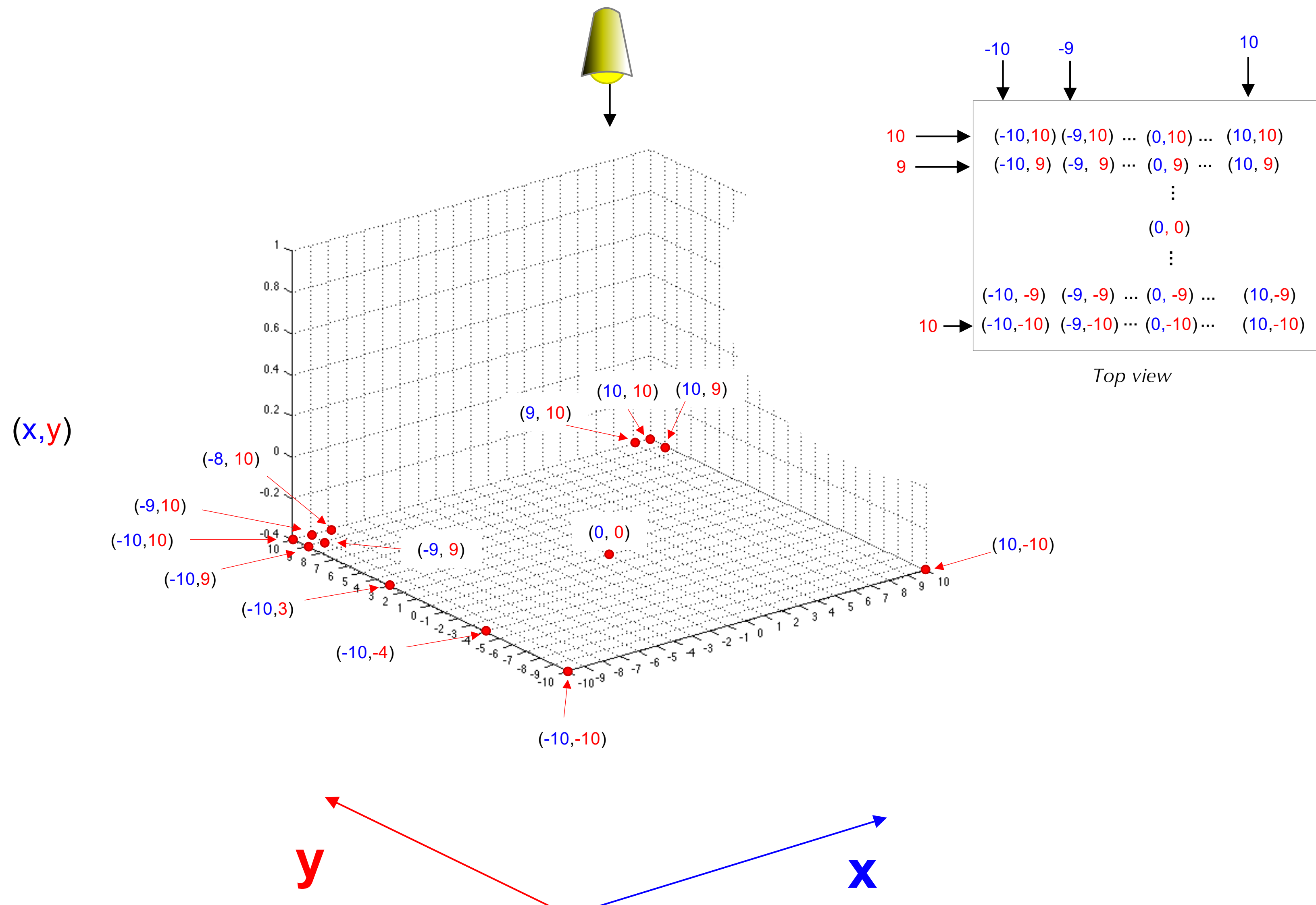
Surface 3D



2.45 [ms]

```
for l=-10:10
  for c=-10:10
      R = sqrt(c^2+l^2)+eps;
      Z = sin(R)/R; % Z scalar
  end
end
```

Surface 3D - vectorization



Surface 3D - vectorization

$$z = f\begin{pmatrix} (-10,10) & (-9,10) & \cdots & (0,10) & \cdots & (10,10) \\ (-10,9) & (-9,9) & \cdots & (0,9) & \cdots & (10,9) \\ & & & \vdots & & \\ & & & (0,0) & & \\ & & & \vdots & & \\ (-10,-9) & (-9,-9) & \cdots & (0,-9) & \cdots & (10,-9) \\ (-10,-10) & (-9,-10) & \cdots & (0,-10) & \cdots & (10,-10) \end{pmatrix}$$

Top view

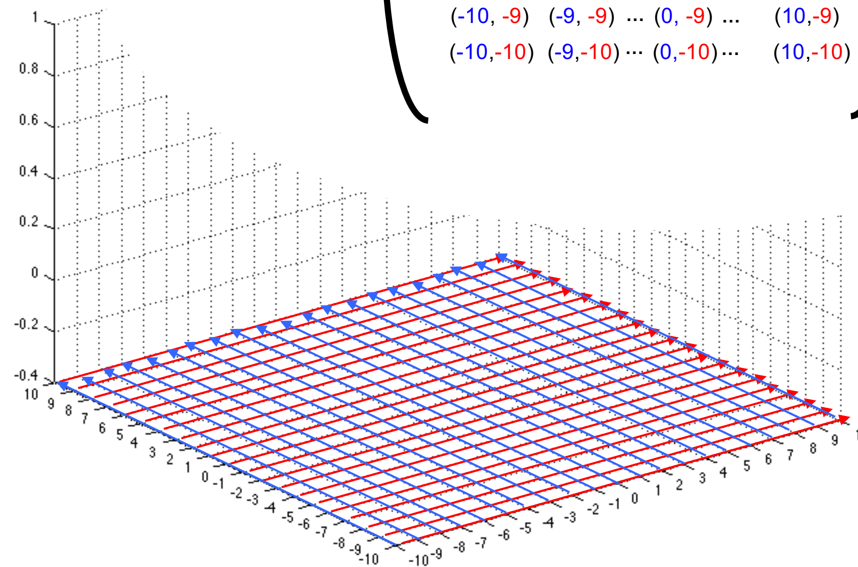
Vectorization ☺

Surface 3D - vectorization

$$z = f\begin{pmatrix} (-10,10) & (-9,10) & \cdots & (0,10) & \cdots & (10,10) \\ (-10,9) & (-9,9) & \cdots & (0,9) & \cdots & (10,9) \\ & & & \vdots & & \\ & & & (0,0) & & \\ & & & \vdots & & \\ (-10,-9) & (-9,-9) & \cdots & (0,-9) & \cdots & (10,-9) \\ (-10,-10) & (-9,-10) & \cdots & (0,-10) & \cdots & (10,-10) \end{pmatrix}$$

Vectorization ☺

```
X= [
    -10, -9 … 10 ;
    -10, -9 … 10 ;
          …
    -10, -9 … 10 ;
   ]
```




```
Y= [
     10,  10 … 10 ;
      9,   9 …  9 ;
           …
    -10, -10 … -10 ;
   ]
```

```
[X Y]= meshgrid(-10:10,-10:10)
```

Vectorization 2D with `meshgrid()`

`Meshgrid()` duplique les vecteurs vx et vy pour produire deux tableaux **X** et **Y** qui contiennent les copies des vecteur vx et vy. Le nombre de répétitions des vecteurs est défini par la taille de l'autre vecteur.

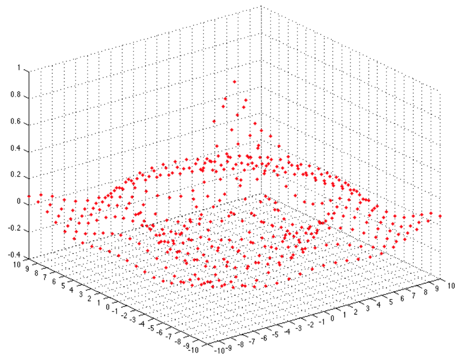
`Meshgrid()` fonctionne de la même manière en 3D

```
vx =  1: 3
vy = 10:14




% no vectorization ☹
% -> slow & inefficient !

for l=1:length(vx)
   for c=1:length(vy)
       z(l,c) =vx(l)*vy(c);
   end
end
```

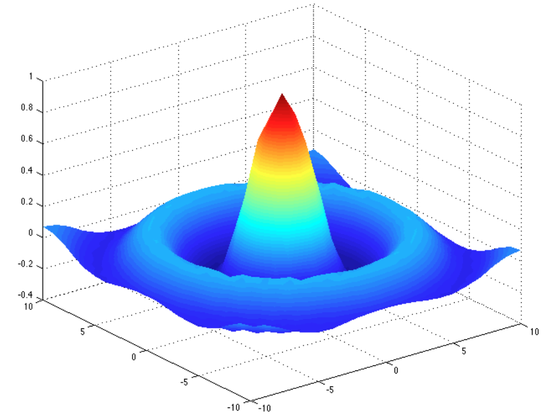
```
>>[X,Y] = meshgrid(vx,vy)
X =
     1     2     3
     1     2     3
     1     2     3
     1     2     3
     1     2     3
Y =
    10    10    10
    11    11    11
    12    12    12
    13    13    13
    14    14    14

 Z = X.*Y
```

Vectorization ☺

Surface 3D



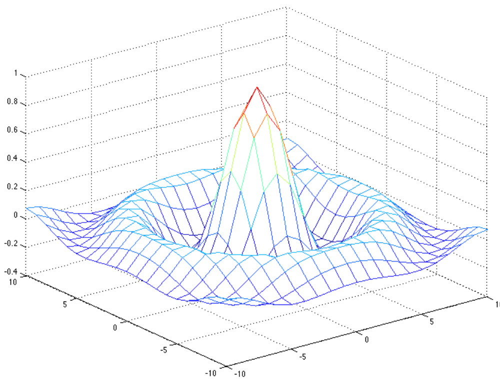
surf(X,Y,Z)

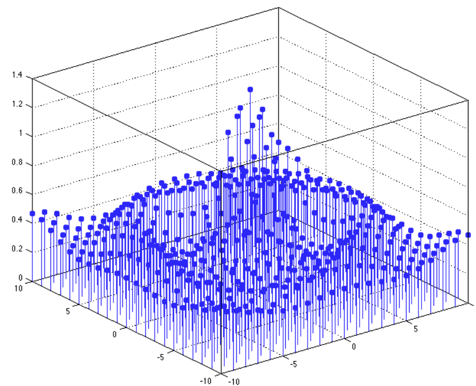


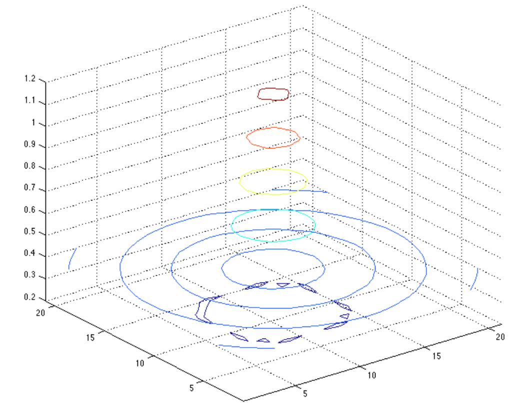
mesh(X,Y,Z)

stem3(X,Y,Z)

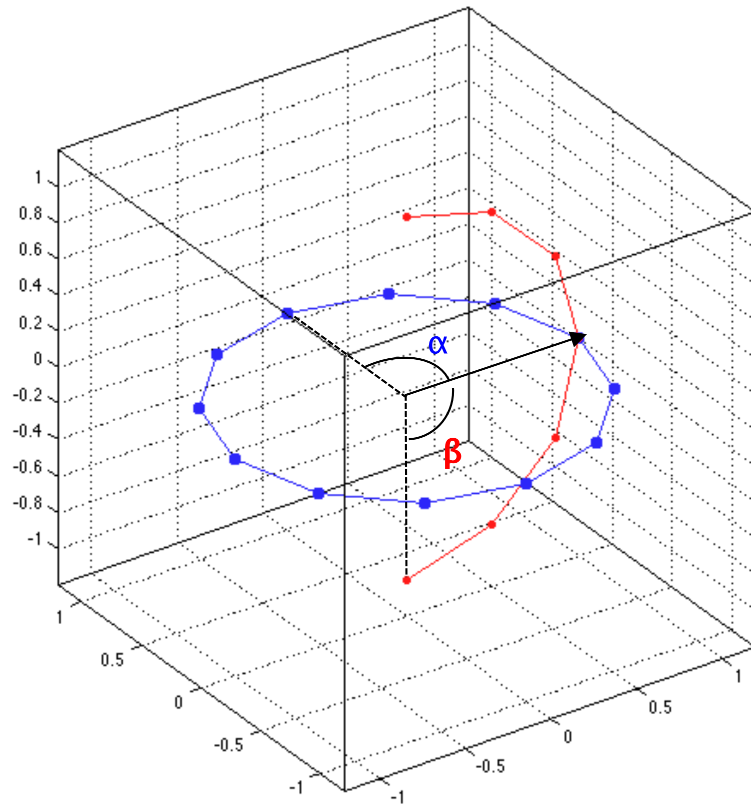
contour(X,Y,Z)







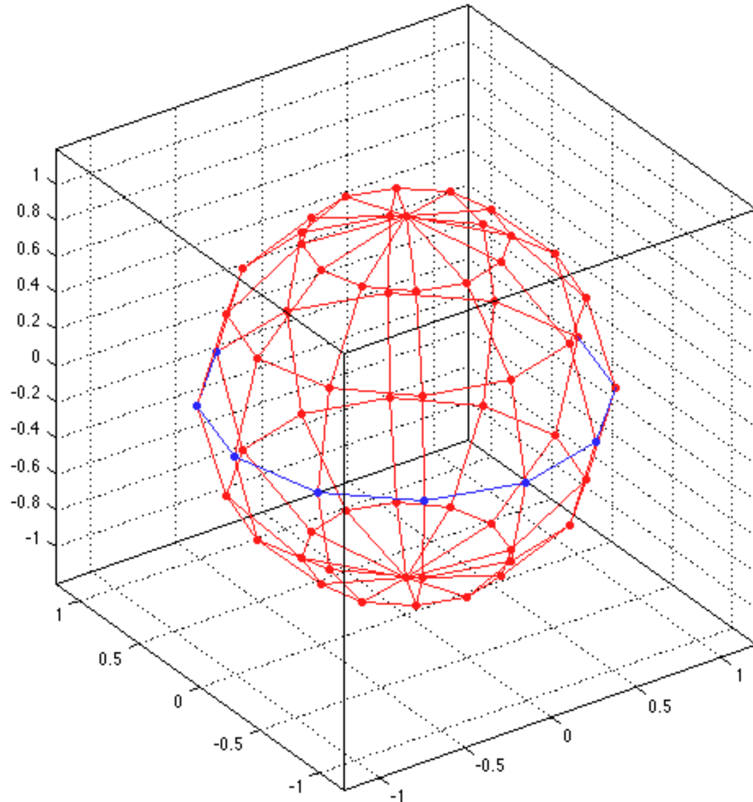
Surface paramétrique 3D



Paramètres: α, β

```
α = linspace(0,2*pi,13) % 1 tour
β = linspace(0,pi,7) % 1/2 tour
```

Surface paramétrique 3D



Paramètres: α, β

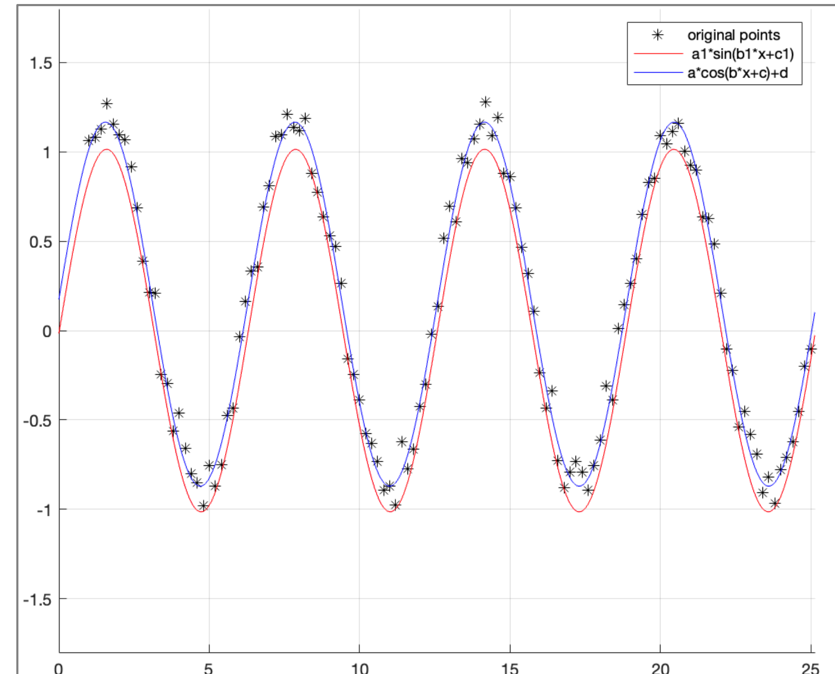
```
αv = linspace(0,2*pi,13) % 1 tour
βv = linspace(0,pi,7) % 1/2 tour

[α, β] = meshgrid(βv, αv);

X = sin(α).*cos(β);
Y = sin(α).*sin(β);
Z = cos(α);

mesh(x,y,z)
```

Vectorization

Fonction `fit()`

Exemple:

```
% fit a sinus
ft = fittype('sin1');

opts =
fitoptions('Method','NonlinearLeastSquares');

[fitResult, gof] = fit( x, y, ft, opts )

plot(fitResult);
```




```
fitResult =  General model Sin1:
     fitResult(x) =  a1*sin(b1*x+c1)
...
gof = sse: 3.4569
...
```

```
% fit a given function
ft2 = fittype( 'a*cos(b*x+c)+d');

opts2 =
fitoptions('Method','NonlinearLeastSquares');

opts2.StartPoint = [0.5 0.9 0.6 1.7];

[fitResult2, gof2] = fit( x, y, ft2, opts2 )

plot(fitResult2, '-b');
```

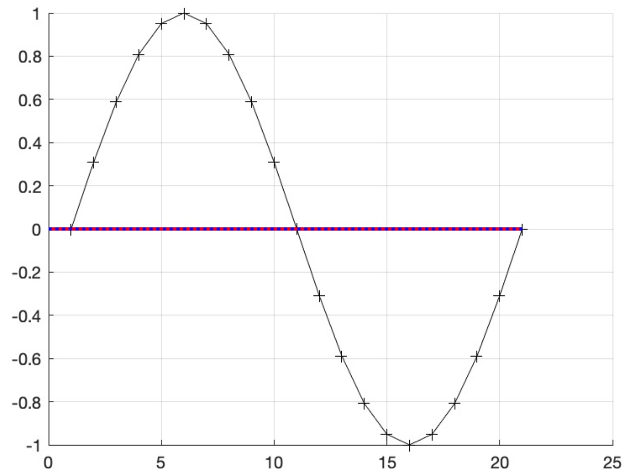
```
fitResult2 = General model:
     fitResult2(x) = a*cos(b*x+cd) + d
...
gof2 = sse: 0.7411
```

Fonction `mean() vs min/max`

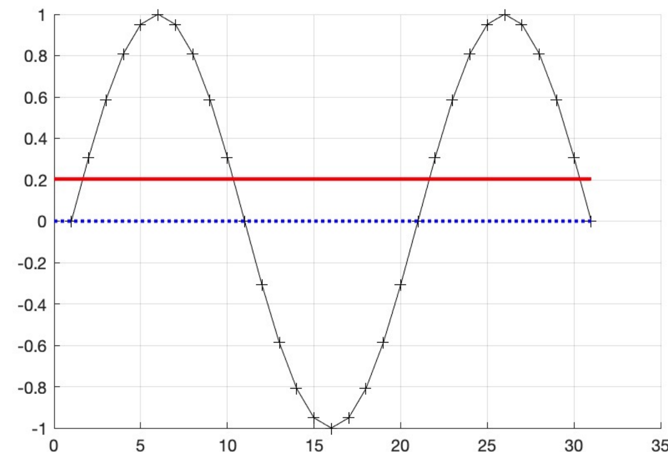
La fonction `mean(X)` calcul la valeur moyenne du vecteur X.
La valeur moyenne d'un sinus (centré) sur un nombre entier de période est 0.
Cependant si ce n'est pas un nombre entier de période, ce qui est le cas dans certain fichiers fournis, la valeur moyenne ne correspond pas à la composante continue.

```
% mean vs min/max

s  = sin(0:pi/10:3*pi);
m  = mean(s)
DC =(max(s)+ min(s))/2
```

outlier, noise ?



m = DC



m ≠ DC

Today

- Symbolic toolbox
- Dérivation (numérique + symbolique)
- Intégration (numérique + symbolique)
- Simulation (ODE + RK45)
- Exemples

Symbolic toolbox

Matlab sait faire du calcul symbolique via la *Symbolic toolbox*, à installer spécifiquement.

Il suffit de définir une variable au format symbolique via **syms** pour que les calculs avec cette variable soit fait de manière symbolique

Exemples:

```
syms x      % lors de la 1e utilisation matlab charge la toolbox, cela peut prendre un peu de temps

y=sym('y') % x et y sont des variables symboliques

f=(x+y)^2  % f est une fonction symbolique car contient des var. symb.

e= expand(f)

   e = x^2 + 2*x*y + y^2

s= simplify(e)

   s = (x + y)^2

p=sym(pi)  % p est symb.
sin(p)     % calculé symboliquement

   ans = 0

sin(pi)    % pi est une valeur numérique !

   ans = 1.2246e-16
```

Symbolic toolbox

Exemples:

```
>> sqrt(2)
  ans =  1.4142

>> sqrt(sym(2))
  ans =  2^(1/2)

% display it nicely
>> pretty(ans)
       1/2
     2

% une fraction reste une fraction
>> sym(4/12) + sym (5/9)
  ans = 8/9
```

```
>> syms x y;
>> simplify(cos(x)^2+sin(x)^2)
  ans = 1

% equivalent à :

>>
simplify(str2sym('cos(x)^2+sin(x)^2'))
   ans = 1

>> expand(str2sym('((a-b)^2)'))
  ans = a^2 - 2*a*b + b^2

>> factor(ans)
  ans = [a - b, a - b]
```

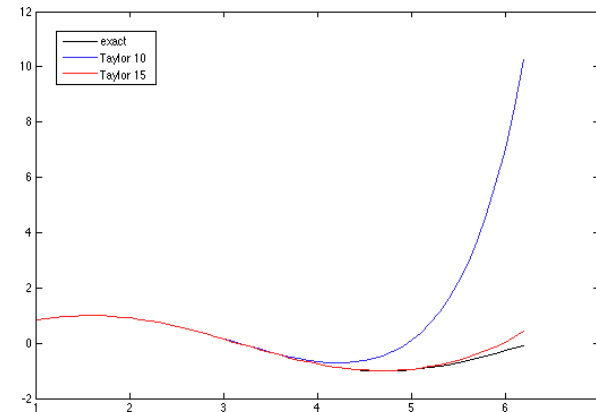
Symbolic toolbox

```
>> syms a b c x
>> f = str2sym('a*x^2 + b*x + c')

   f = a*x^2 + b*x + c

>> subs(f,a,2)  % substitue 'a' par 2 dans f()

   ans = 2*x^2 + b*x + c


% PDE
>> syms x y;
>> f = sin( x*y );
>> diff(f,x);    % dérive f() selon x

   ans = y*cos(x*y)
```

Symbolic toolbox

$$\sin(x) = \sum_{i=0}^{\infty} \frac{(-1)^i x^{(2i+1)}}{(2i+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

```
>> syms x
>> f = taylor(sin(x),'Order', 10)

 f = x^9/362880 - x^7/5040 + x^5/120 - x^3/6 + x

>> pretty(ans)

     9        7      5      3
    x        x      x      x
  ------ - ---- + --- - -- + x
  362880   5040   120    6

>> g=taylor(sin(x),'Order', 15)
>> xx = 1:0.1:2*pi;
>> plot(xx,sin(xx),xx,subs(f,x,xx),xx,subs(g,x,xx))
```



- Demos symbolic toolbox

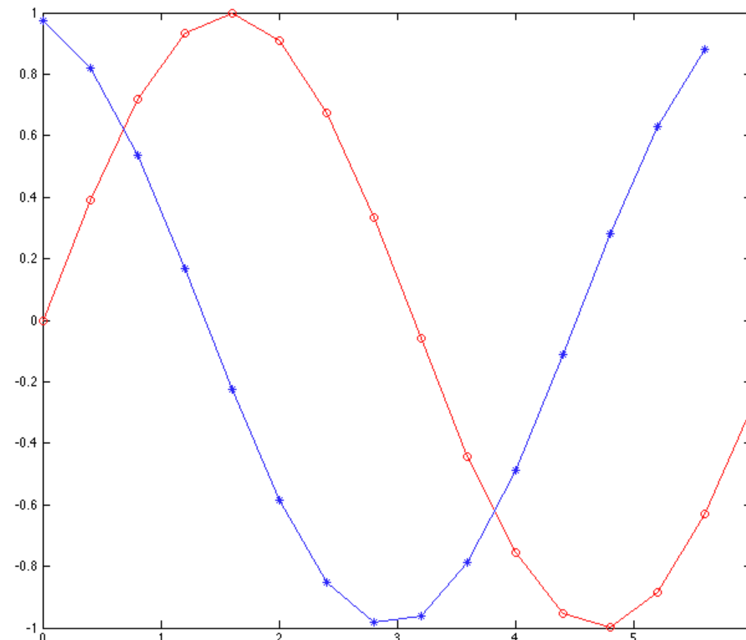
Dérivée - différentiation

La fonction **`diff()`** calcul la différence entre 2 points adjacents d'un vecteur.
Δ Le vecteur retourné à **1** élément de moins.
Pour une matrice la fonction **`diff()`** est appliquée ligne par ligne.

```
diff(X)=[X(2)-X(1) X(3)-X(2) ... X(n)-X(n-1)]
```

```
pas = 0.4;
x   = 0:pas:2*pi;
y   = sin(x);
dY  = diff(y);
dX  = diff(x);
plot(x,y,'r-o',...
     x(1:end-1),dY./dX,'-*');
```

```
Ex.

dY(1)=(y(2)-y(1))/(x(2) - x(1))
     = (0.4-0.0)/0.4    = 1
```

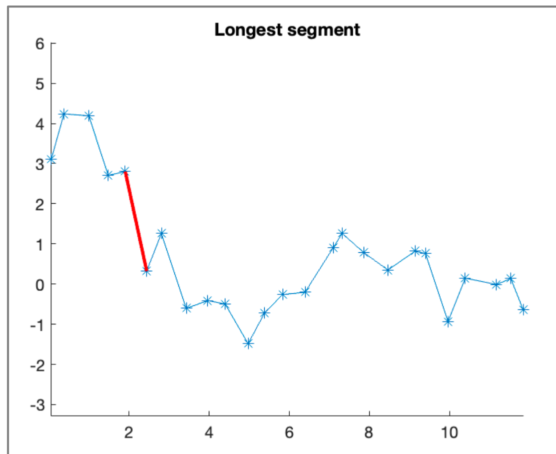


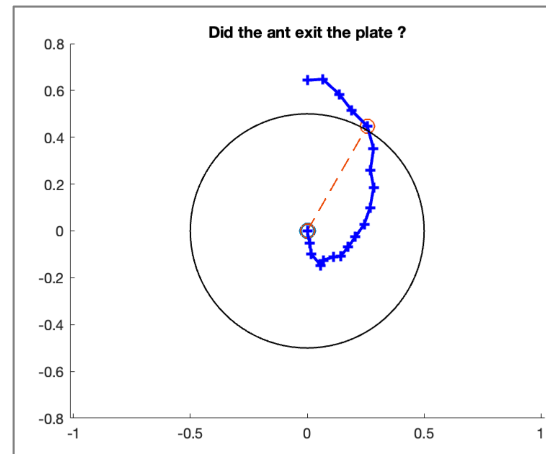
Différentiation symbolique

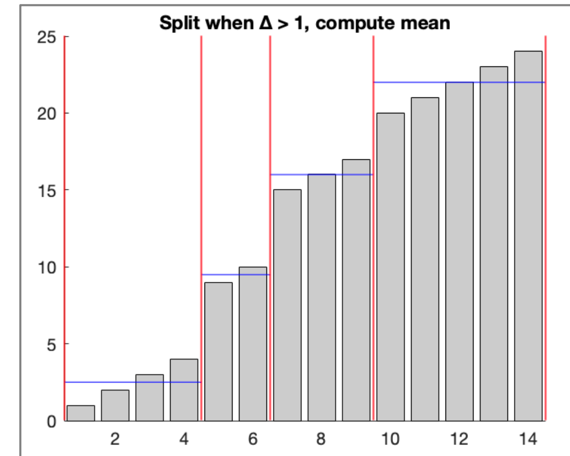
La fonction **`diff()`** permet également de calculer une dérivée de manière symbolique.

```
syms x
diff(sin(x))   %1st derivative

 cos(x)

diff(sin(x),2) %2nd derivative

 -sin(x)

diff(str2sym('cos(y)'),sym('y'))   %1st derivative

 -sin(y)

diff(str2sym('cos(y)'),sym('y'),2) %2nd derivative

 -cos(y)
```

Diff - exemples





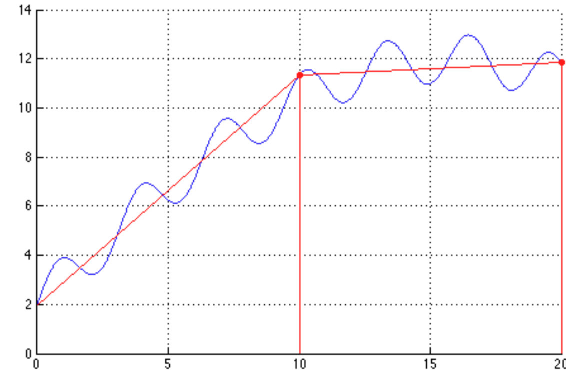


- Example longest segment
- Example ant ouf the plate
- Example split segments when Δ > seuil

- Demos dérivation

Intégration - méthode des trapèzes

Intégration numérique par la méthode des trapères
avec la fonction `trapz(X,Y)` ou `cumtrapz(X,Y)`

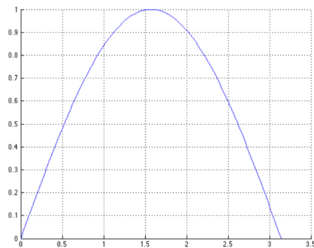


Ex. Sin(x) avec la méthode des trapèzes

$$\int_0^{\pi} \sin(x)\,dx = 2$$

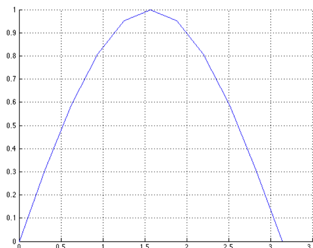
```
X = 0:pi/n:pi;
Y = sin(X);
Z = trapz(X,Y)
```

n = 100 pts



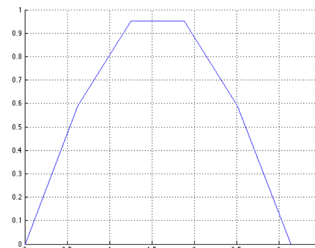
z = 1.9998

n = 10 pts



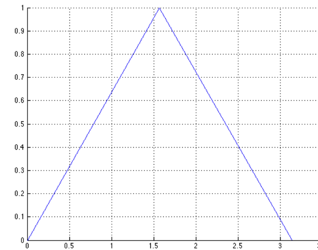
z = 1.9835

n = 6 pts



z = 1.9338

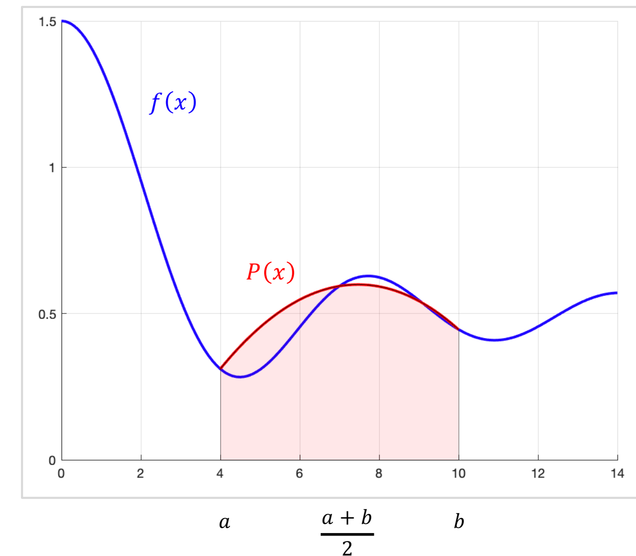
n = 3 pts



z = 1.5708

Intégration - méthode de Simpson

`f =@(x) 0.5+sin(x+eps)./(x+eps)`



Intégration numérique par la méthode de Simpson approximation de $f(x)$ par le polynôme $P(x)$ dans l'intervalle [a b]

$$\int_a^b f(x)dx \approx \int_a^b P(x)dx = \int_a^b \frac{b-a}{6}\left[f(a) + 4f\frac{a+b}{2} + f(b)\right]$$

```
q = integral(f,a,b)
```

numerically integrates function `f` from `a` to `b` using global adaptive quadrature and default error tolerances.

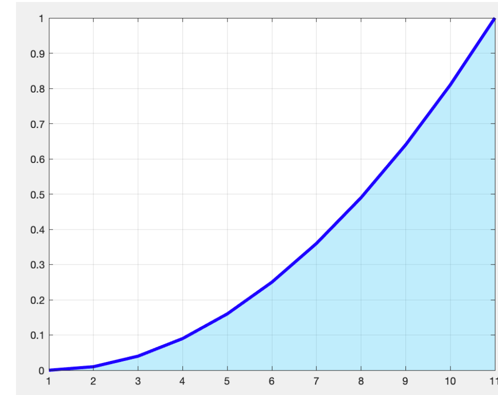
- `'AbsTol'` specify **Absolute** error tolerance
- `'RelTol'` specify **Relative** error tolerance

L'ancienne fonction `quad()` similaire est remplacée par `integral()`

Comment choisir `a` et `b`, etc. ? -> W. Gander and W. Gautschi, Adaptive Quadrature - Revisited, BIT Vol. 40, No. 1, March 2000, pp. 84-101.

Intégration numérique avec @fcn

```
mySQR= @(x) x.^2;

integral(mySQR,0,1)

   ans = 0.3333
```



Avec un paramètre dans la fonction @

```
c= 3;

mySQR2= @(x) x.^2 + c;

integral(mySQR2,0,1)

    ans = 3.3333
```

```
c= 4;

mySQR2= @(x) x.^2 + c;

integral(mySQR2,0,1)

ans = 4.3333
```

Comment faire pour définir un paramètre sans devoir réécrire la fonction anonyme `mySQR()` à chaque fois ?

@fonction avec param

Comment faire pour définir un paramètre sans devoir réécrire la fonction anonyme `mySQR()` à chaque fois ?

Idéé: *laisser matlab (langage interpreté) redéfinir la fonction à chaque appel*

```
function s= mySQRf(x,e)          % doit se trouver dans un autre fichier ou
   s=x.^2 + e;                   %   à la fin du fichier courant
end
```

```
integral( @(x) mySQRf(x,4) ,0,1)
```

```
ans =   4.3333
```

Redéfinition de la fonction anonyme

`@(x)  mySQRf(x,4)`

`fcn anonyme = mySQRf(x,4)`

```
for e =1:3
    integral( @(x) mySQRf(x,e) ,0,1)
end
```

@function redéfinie 3 x

```
ans =   1.3333
ans =   2.3333
ans =   3.3333
```

@fonction avec param

Laisser matlab (langage interpreté) redéfinir la fonction à chaque appel.

Note: matlab permet d'avoir des fonctions **imbriquées**, elles sont évaluées/définies à chaque appel de la fonction principale

```
function y = Int_mySQRf2(min, max, g)

    y = integral( @mySQRfi , min, max);  % Appel de la fonction mySQRfi()

    function s = mySQRfi(x,g)              % Définition de la fonction mySQRfi()
        s=x.^2 + g;                        % Sera réévaluée à chaque appel de Int_mySQRf2
    end

end
```

```
for e =1:3
    Int_mySQRf2 (0,1, e))
end
```

```
ans =   1.3333
ans =   2.3333
ans =   3.3333
```

Intégration symbolique

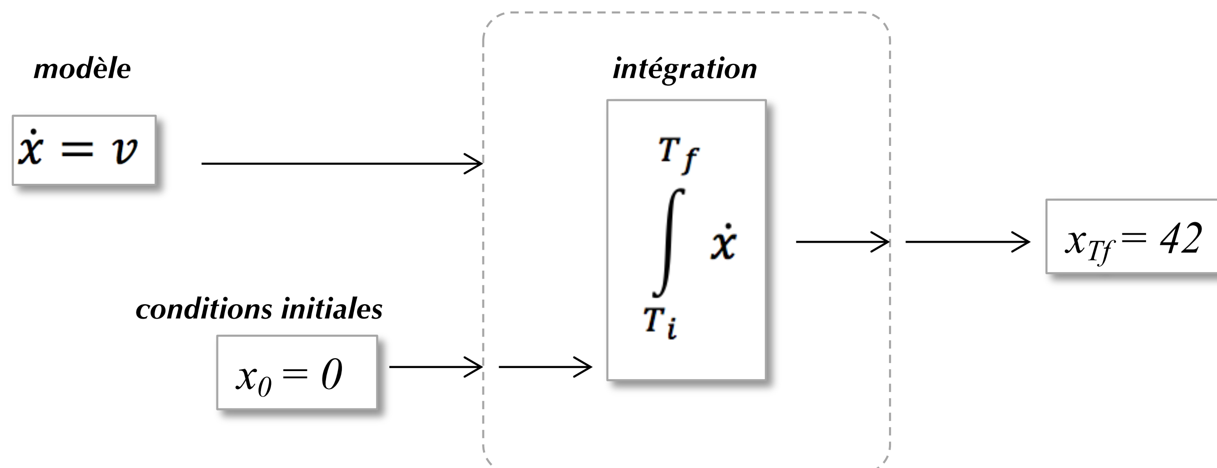
Intégration symbolique avec la fonction `int(X,Y)`.

```
syms x
z = int(sin(x))

 z = -cos(x)


z = int(sin(x),x,0,pi)

 z = 2
```

- Demos diff-trapz, Trapz, Simpson

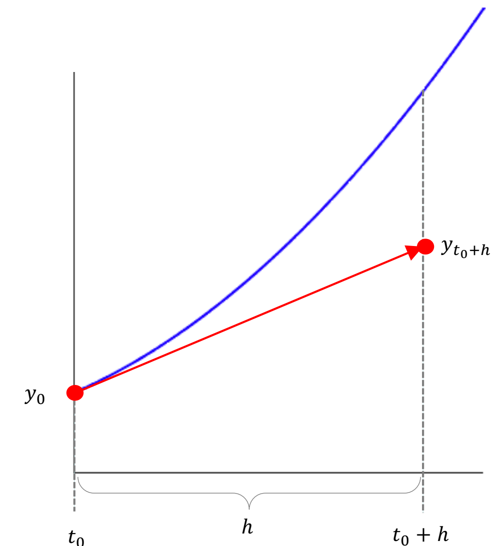
Simulation

- Un système physique (ou économique, etc.) peut être décrit à l'aide d'équations mathématiques définissant un modèle plus ou moins précis de la réalité.
- Le modèle décrivant le système physique est employé par un logiciel de simulation (ex. *toolbox* Simulink) pour représenter l'évolution du modèle au cours du temps.
- L'évolution temporelle du modèle est calculée de manière itérative en intégrant numériquement les équations sur un petit pas de temps.
- Plusieurs méthodes sont à disposition du moteur d'intégration ou *Solver* pour intégrer les équations. La plus simple est la méthode des trapèzes et la plus courante est Runge-Kutta.
- La manière de définir mathématique un modèle ainsi que les problèmes liés à l'intégration numérique (condition initiales, convergence, etc) seront abordés dans d'autres cours.

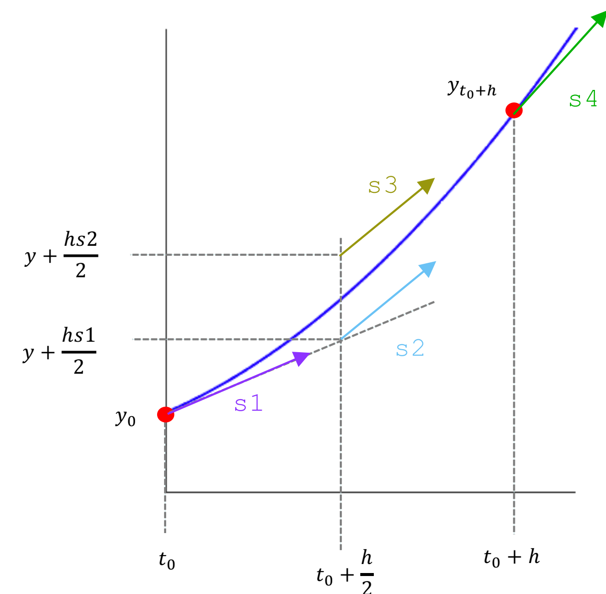


Integration avec Runge-Kutta

```
% RK1 (= Euler)

dydt = @(t,y) (2*t);

s1 = dydt(t0,y0); %slope in t0

y1 = y0 + h*s1;
```




```
% RK4

dydt = @(t,y) (2*t);

s1 = dydt(t0,      y0 );
s2 = dydt(t0+h/2, y0 + h/2 * s1);
s3 = dydt(t0+h/2, y0 + h/2 * s2);
s4 = dydt(t0+h,   y0 + h   * s3);

yf = y0 + h *(s1 + 2*s2 + 2*s3 + s4)/6;
```

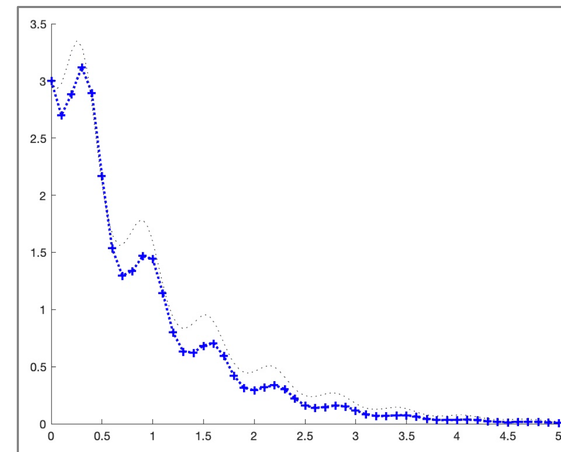


Simulation – Euler - PhM

```
%% Simple ODE solver using Euler's method
% simulate F between t0 and tf
% Func is the @function to integrate
% t0: initial time
% h: step size
% tf: final time
% y0: initial state

function y_out = myODE1(Func,t0,h,tf,y0)
  y = y0;
  y_out = y;
  for t = t0 : h : tf - h
     s = Func(t,y);
     y = y + h*s;
     y_out = [y_out; y];
  end
end

%% define dydt to simulate, PhM eq.
dydt = (2*sin(10*t)-1)*y


%% Simulate dydt between t0 and tf
t0 = 0;   h= 0.1;   tf= 5;
y0 = 3;

Y = myODE1(dydt,t0,h,tf,y0);
```

t	s	y
		3.0
0.0	-3	2.7
0.1	1.8439	2.8844
0.2	2.3612	3.1205
0.3	-2.2398	2.8965
0.4	-7.2807	2.1685
0.5	-6.3272	1.5357
…		

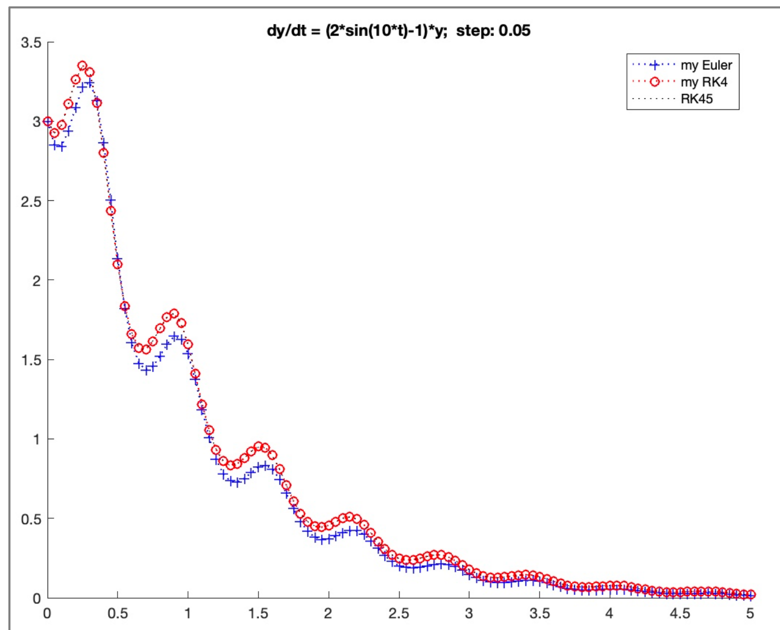
s = (2*sin(0) -1)*3, y = 3 + s*0.1
s = (2*sin(10*0.1) -1)*2.7, y = 2.7 + s*0.1
…



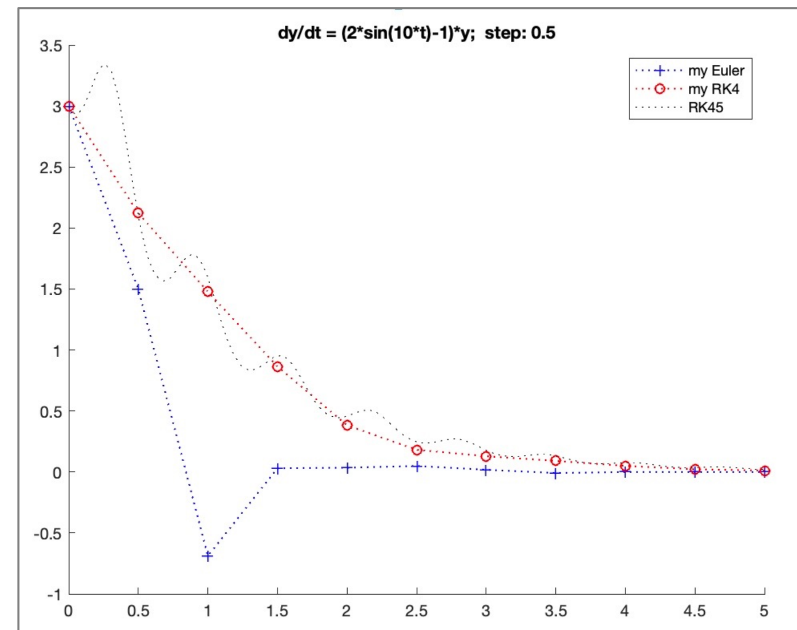

```
t = t0 : h : tfinal;
plot(t,Y,'-+b' ,'linewidth', 2)
```

>> edit ode45.m

Integration - Euler vs RK4 vs step



Step = 0.05



Step = 0.5

- Demo myODE1, myODE4 vs ode45

Matlab solver - ode45

Matlab propose différents *solvers,* ode45() est le premier qui devrait être testé.

```
[t,y] = ode45(odefun,tspan,y0),
```

`ode45()` intègre le système d'équations différentielles **odefun** $y'=f(t,y)$ de **t0** à **tf** avec les conditions initiales **y0** en utilisant un pas d'intégration variable. Chaque lignes de la solution **y** correspond au temps **t**.

`odefun` est une référence sur une fonction (@fun), **t** scalaire, **y** vecteur colonne,
`odefun(t,y)` retourne un vecteur colonne correspondant à $f(t,y)$.

`tspan` : **[t0 tfinal]** ou **[t0 t1 ... tfinal]**

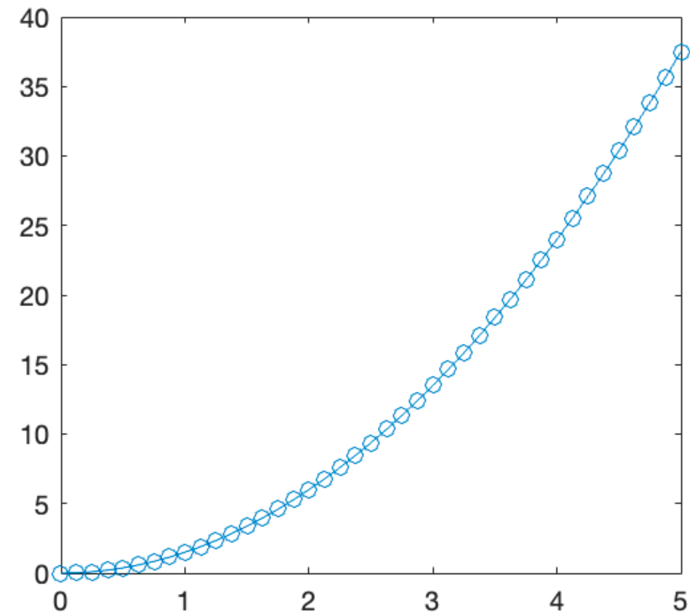
`y0` : vecteur colonne avec les conditions initiales, une par équation

Les nombreuses options de simulation peuvent être spécifiées dans `ode45(f,t,y0, opt)` via la fonction `opt = odeset(Name,Value,...)`

ODE45 exemple

Simuler l'équation $\dot{y} = 3t$ pour t entre [0..5] avec y0 = 0

```
tspan = [0 5];
y0 = 0;
odefun = @(t,y) 3*t;

[t,y] = ode45(odefun, tspan, y0);

plot(t,y,'-o')
```




```
%% symbolic solution

syms y(t)
ode =      diff(y,t) == 3*t
ySol(t) = dsolve(ode)
F = matlabFunction(ySol(t))
plot(0:0.2:5,F(0,0:0.2:5),'-*')
```

ODEfun

Matlab **ode solver** résout uniquement des équations du 1ᵉ ordre. Une équation d'ordre plus élevé doit être réécrite comme un système d'équations du 1ᵉ ordre en utilisant la substitution suivante:

$$y1 = y$$
$$y2 = \dot{y}$$
$$y3 = \ddot{y}$$
$$\dots$$
$$yn = y^{(n-1)}$$

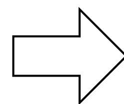
$$\Rightarrow \begin{cases} \dot{y}1 = y2 \\ \dot{y}2 = y3 \\ \dots \\ \dot{y}n = f(t, y1, y2, ..yn) \end{cases}$$

Exemple pour $m\ddot{x} + c\dot{x} + kx = 0$

$$v = \dot{x} \quad \rightarrow \quad m\dot{v} + cv + kx = 0$$
$$\dot{v} = -\frac{cv + kx}{m}$$

$$y1 = x$$
$$y2 = \dot{x} = v$$

$$\dot{y}1 = v = y2$$
$$\dot{y}2 = \dot{v} = -\frac{cv + kx}{m}$$



```
function dydt = odefun(t,y)
  dydt(1) = y(2);
  dydt(2) = (-c/m)*y(2) + (-k/m)*y(1) ;
end
```

Simulation - balle

Exemple:

Balle lâchée d'une hauteur de 10m avec une vitesse initiale nulle

- La vitesse de la balle va accélérer de manière constante, défini par *g*
- Le modèle a 2 états, la position et la vitesse de la balle

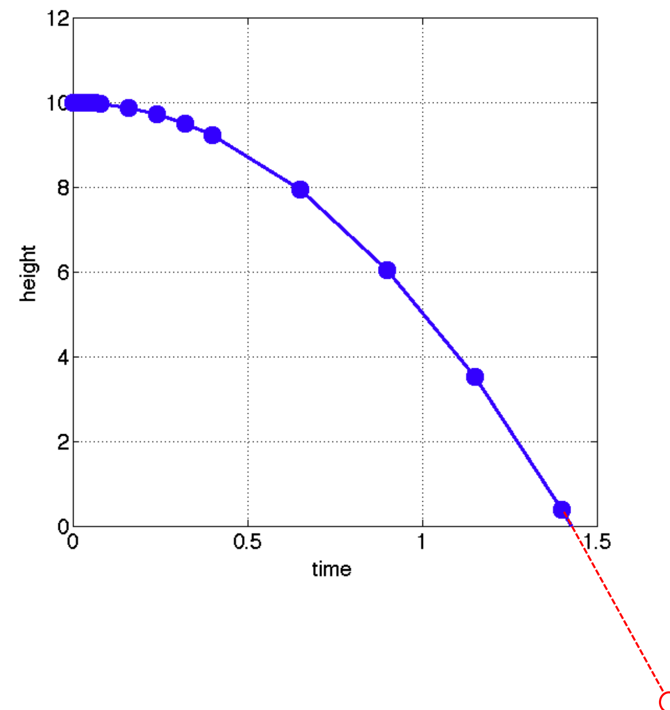
$$\frac{dx}{dt} = v \qquad \frac{dv}{dt} = -g$$

- Le *solver* ODE intègre les équations dynamiques du modèle à partir des conditions initiales, dans un lapse de temps donné en utilisant la méthode Runge-Kutha.

Simulation - balle

- Le *solver* ODE intègre la fonction `ODEball` à partir des conditions initiales `y0`, dans un lapse de temps `[0 2]` sec en utilisant la méthode Runge-Kutta **`ode45`**.
- Le modèle à intégrer a 2 états: la position *x*: `y(1)` et la vitesse *v*: `y(2)` de la balle

```
% modele a integrer
function dydt = ODEball(t,y)
   dydt = [y(2); -9.8];
end

% cond. init. h=10, v=0
y0 = [10 0];
t0 = 0;  tf = 2;
[t y]= ode45(@ODEball,[t0 tf],y0);
plot(t,y (:,1),'-bo');
```



Simulation – balle & rebonds

- La collision avec le sol (**y(1)=0**) est détectée avec un événement `events` qui stoppe le *solver*.
- La simulation est ensuite relancée avec de nouvelles conditions initiales: nouvelle vitesse = - 0.85 * ancienne vitesse; , hauteur = 0

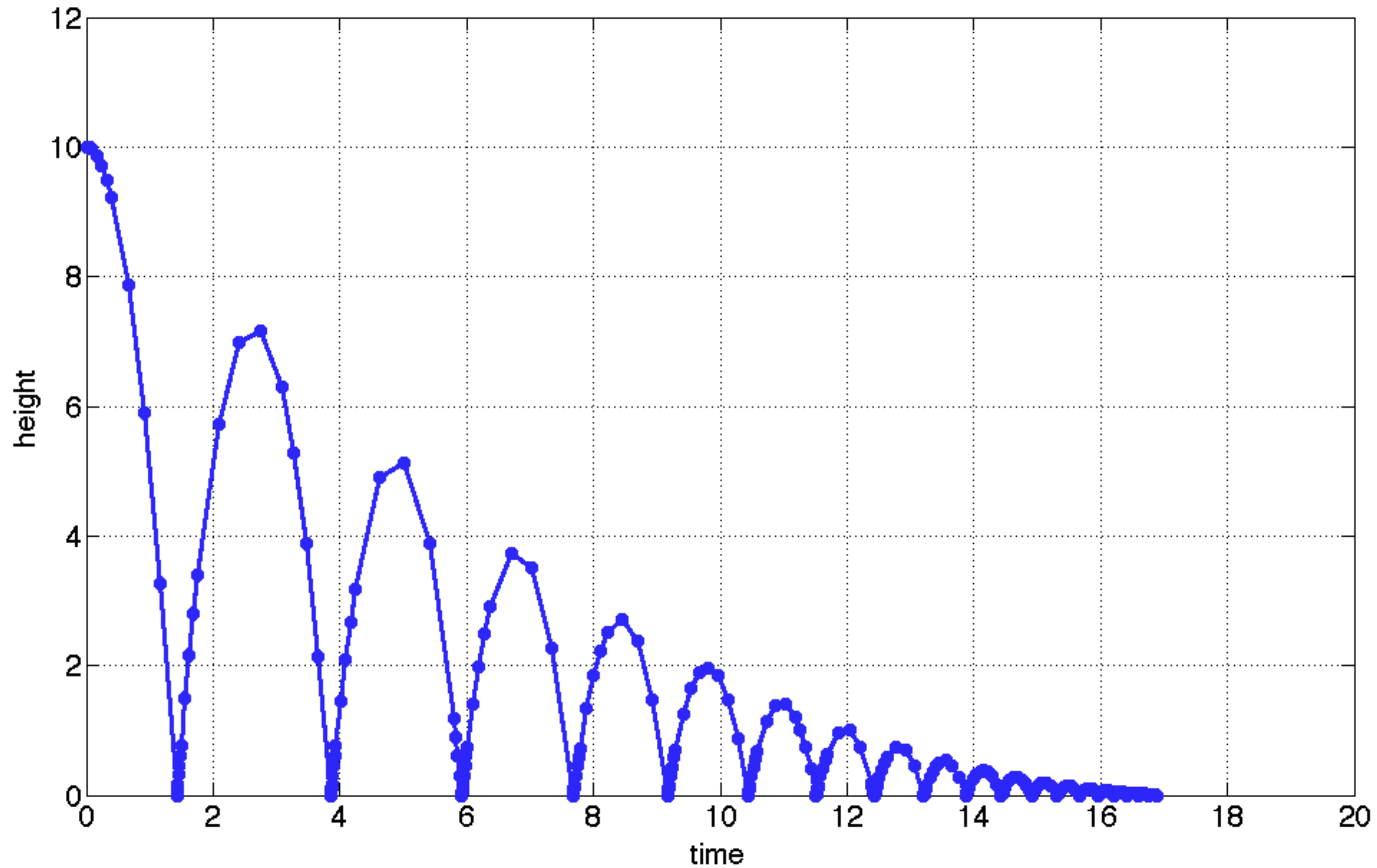
```
function [val, stop, dir] = events(t,y)
    val  = y(1);   % observe position
    stop = 1;      % stop simulation
    dir  = -1;     % only decreasing val
 end

opt = odeset('Events',@events);
```

Simulation – balle & rebonds

```
function dydt = ODEball(t,y)
   dydt = [y(2); -9.8];
end


ts = 0; tf = 30;

y0 = [10;  % 10 m
       0]; %  0 m/s


tout = ts;
yout = y0'; % init. var.
```

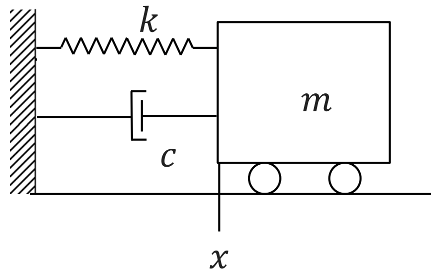
```
% 20 rebonds
for i = 1:20

    % simule 1 rebond
    [t,y] = ode45(@ODEball,[ts tf],y0,opt);

     % accumule les pts
     nt = length(t);
     tout = [tout; t(2:nt)];
     yout = [yout; y(2:nt,:)];

     % nouvelles cond. init.
     y0 = [0, -0.85*y(nt,2)];
     ts = t(nt); % t final simul current

end

 plot(t,y (:,1),'-bo');
```

Simulation – balle & rebonds



- Demos simple ode45 et ball
- (pendul)

Simulation Masse-Ressort-Amorti



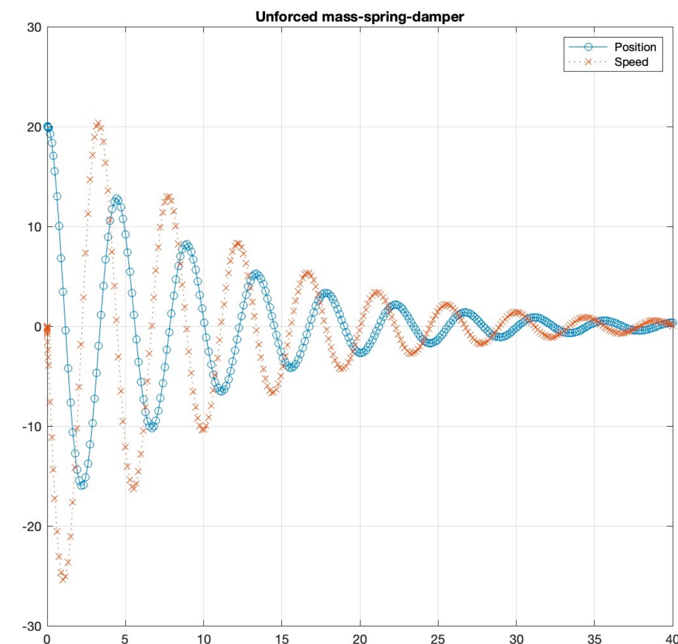
$$m\ddot{x} + c\dot{x} + kx = 0$$

$$y1 = x$$
$$y2 = \dot{x} = v$$

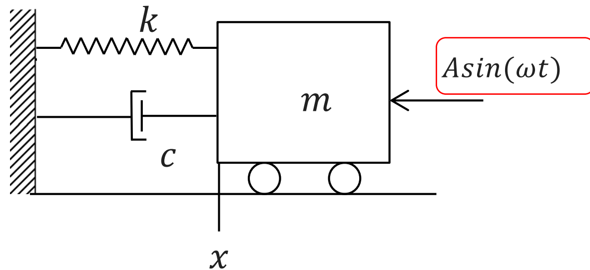
$$\dot{y}1 = v = y2$$
$$\dot{y}2 = \dot{v} = -\frac{cv + kx}{m}$$

```
function xdot = Unforced_Damped_Spring(t, x)
  m = 2;
  c = 0.4;
  k = 4;
  xdot = [ x(2); %
         -(c/m) * x(2) - (k/m) * x(1) ];
end
```

```
x0 = [20;  % Initial pos
       0]; % Initial speed

[t, X] = ode45(@Unforced_Damped_Spring, [0, 40.0],x0);

plot(t, X(:,1), 'o-',t, X(:,2),'x:')

title('Unforced mass-spring-damper')
legend(' Position', ' Speed')
```

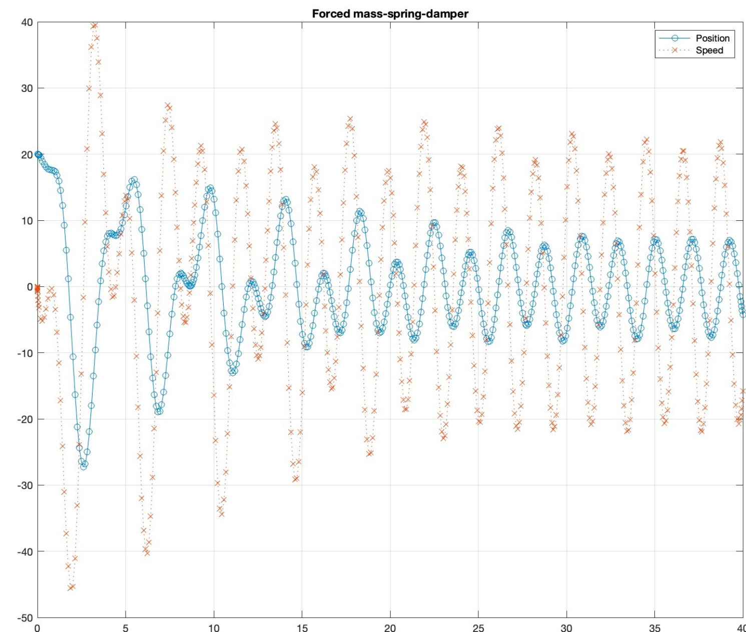


Simulation Masse-Ressort-Amorti 2



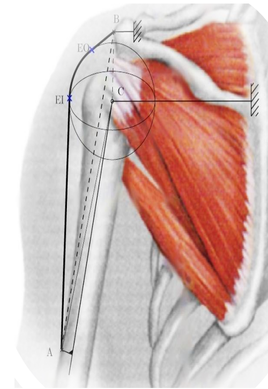
$$m\ddot{x} + c\dot{x} + kx = Asin(\omega t)$$

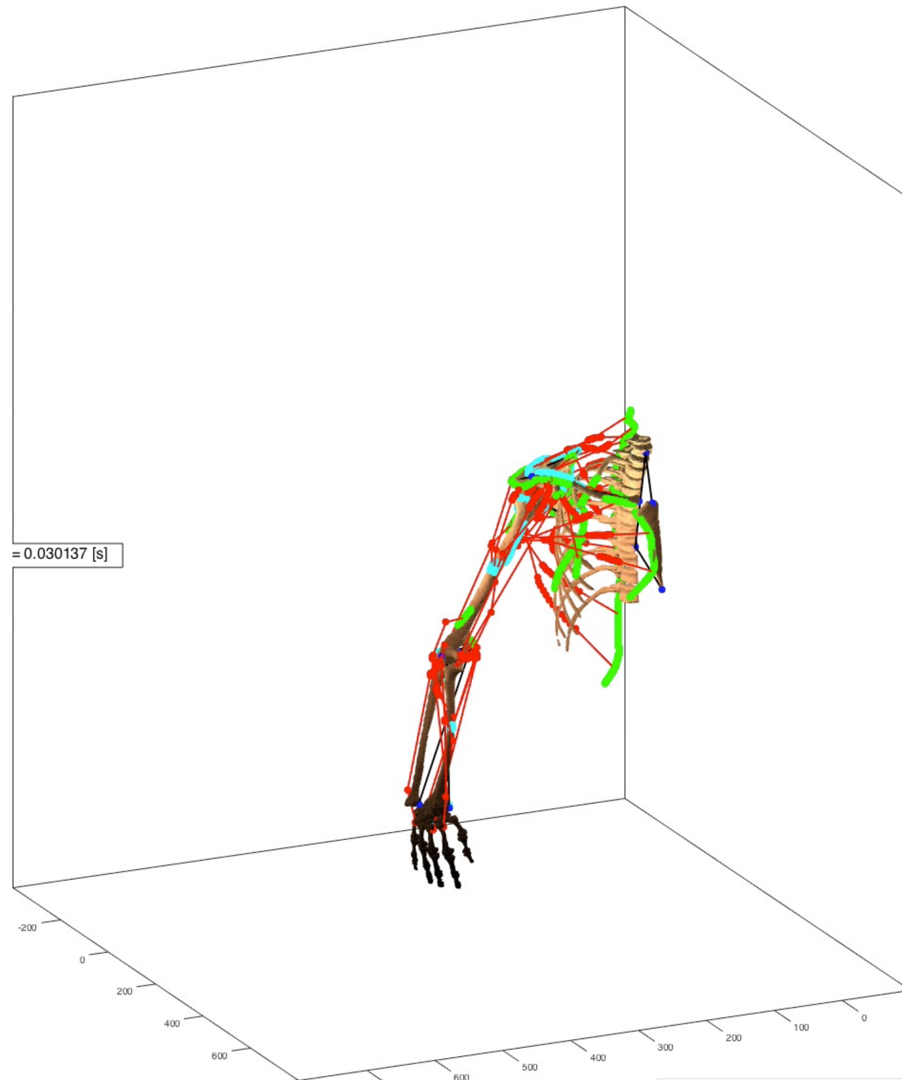
```
function xdot = Forced_Damped_Spring(t, x)
  m = 2;
  c = 0.4;
  k = 4;
  A=100;
  omega = 3;

  xdot = [ x(2); % 1st derivative,
          ((A/m)*sin(omega*t)) ... % forced excitation
             -(c/m) * x(2) - (k/m) * x(1) )]; % 2nd derivative
end


x0 = [20;  % Initial pos
       0]; % Initial speed


[t, X] = ode45(@Forced_Damped_Spring, [0, 40.0],x0);

plot(t, X(:,1), 'o-',t, X(:,2),'x:')


title(Forced mass-spring-damper')
legend(' Position', ' Speed')
```



- Demos unforced/forced mass-spring-damper + animation

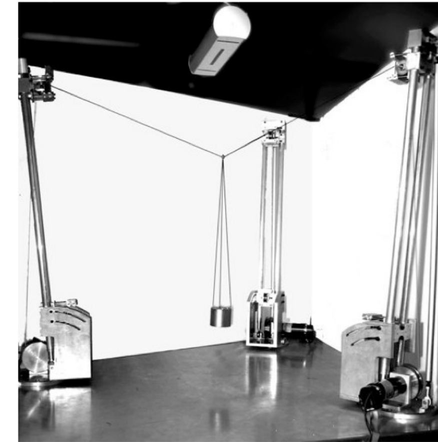
Exemple: projet épaule



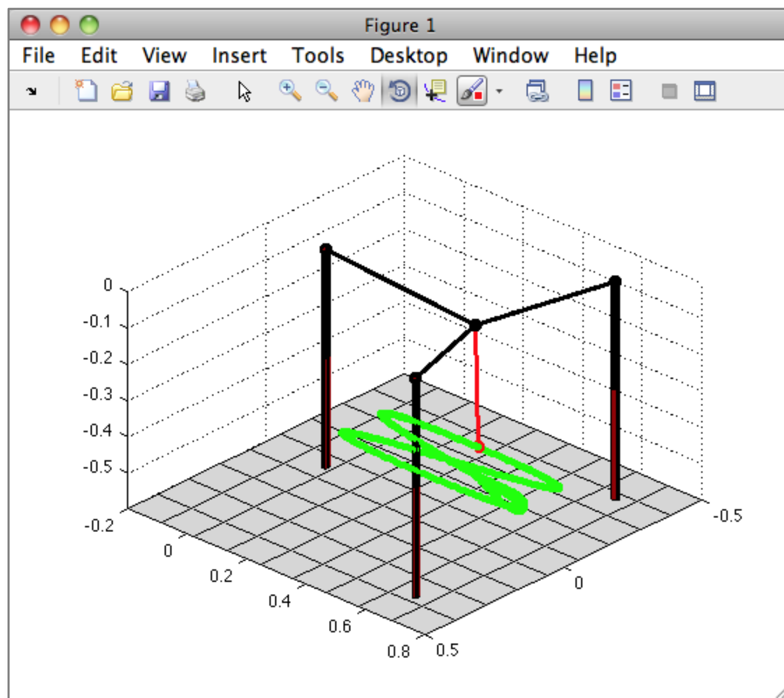


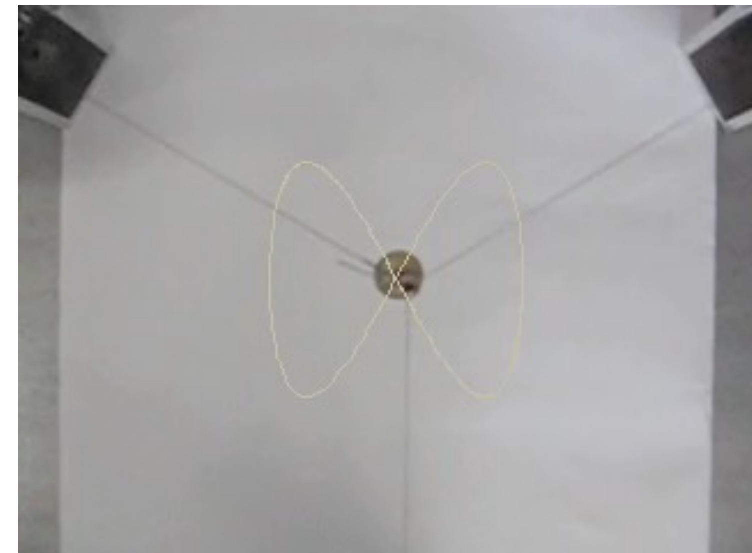
Modélisation des mouvement d'une épaule (skelet + muscles) pour étudier les forces dans les articulations afin d'étudier les causes possible de l'artrose.

Exemple: spider-crane

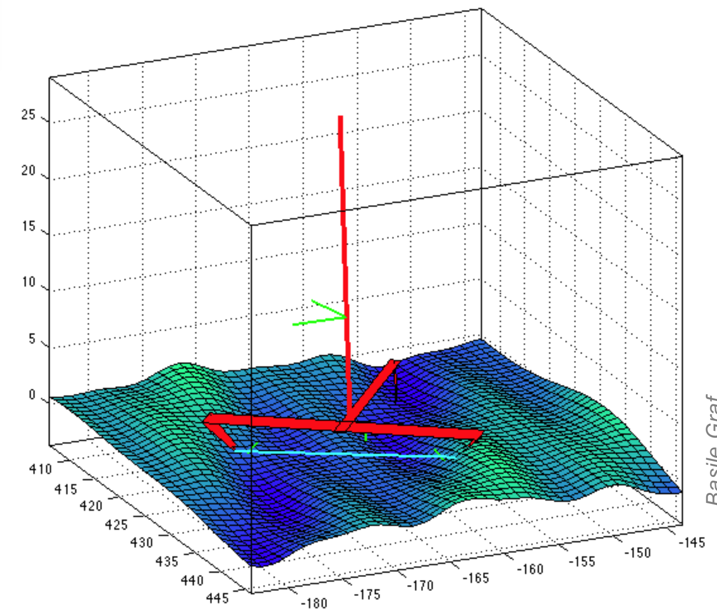


Grue sans partie mobile, permet des déplacements dynamiques rapides.





Simulation – Hydroptère



http://hydroptere.epfl.ch/

Simulation – quel solver ?

Commencer avec ode45()

Puis suivez les guidelines de matlab

https://ch.mathworks.com/help/releases/R2023b/matlab/math/choose-an-ode-solver.html

Il existe des solvers spéficiques provenant d'autres fournisseurs que Mathworks

Matlab

- Nous avons juste gratté la surface de Matlab, mais …
- **Vous n'êtes pas les premiers étudiants à employer matlab**
- **Quelqu'un a surement déjà résolut un problème similaire au votre**
- Premier réflexe est le **help/doc** de matlab, puis les forums matlab et le web qui regorge d'exemples et de réponses à un problème similaire au votre

- … vous allez continuer à employer matlab intensivement dans les cours du BA

Slides supplémentaires

eval

Il est possible d'évaluer une expression définie par une chaine de caractère à l'aide de la fonction **eval**

```
[out1, … ,outn] = eval(expression)
```

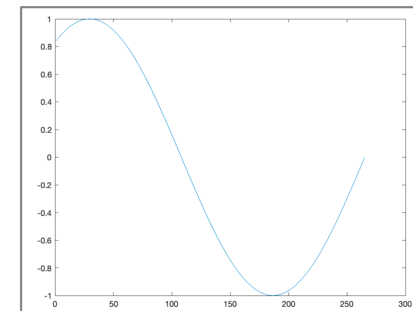
`expression` à évaluer, (sécurité slide suivant)

`out1, … ,outn` 0 ou plusieurs valeurs de sortie

la fonction **eval** est évaluée, et donc potentiellement *lente* et *dangereuse*

Exemples

```
>> s=eval('sin(1:3)')
s =
    0.8415    0.9093    0.1411

>> eval('plot(sin(1:.02:2*pi));shg')
```

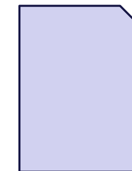


eval - sécurité

Comme l’expression à évaluer est un code quelconque qui peut être défini dans un fichier ou entré par l’utilisateur il faut être particulièrement précautionneux concernant le contenu d’expression.

Exemple

```
Password = 'VerySecretPW’

% many lines …

exp=importdata(‘Exp.txt')
eval(strjoin(exp))

% many lines …

>> Password

Password =

    '123456'
```



Exp.txt

```
Password='123456'
```

Demo solve