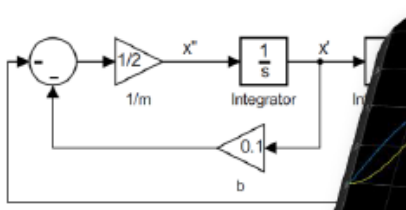
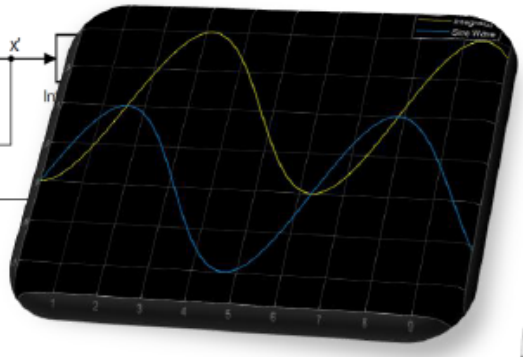
Maîtriser MATLAB et Simulink en deux heures

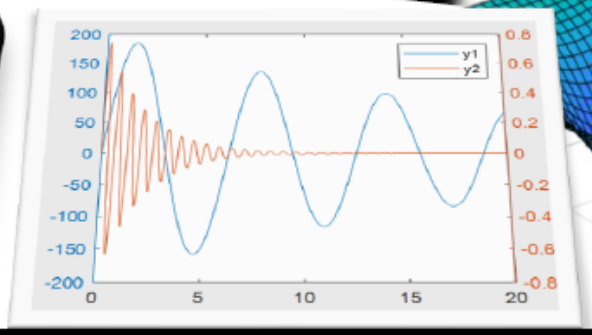
Abdelhamid BOUHELAL

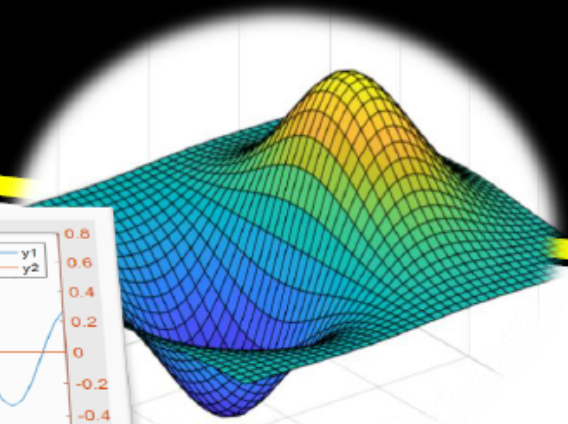
New

Maîtriser *Matlab* et *Simulink* en deux heures









Abdelhamid BOUHELAL, PhD

Plan

Introduction au Logiciel Matlab

- Matlab est un logiciel de calcul numérique commercialisé par la société MathWorks (http://www.mathworks.com).
- MATLAB est un langage **interprété**: les instructions sont interprétées et exécutées ligne par ligne (pas de compilation avant de les exécuter).
- Initialement développé à la fin des années 70 par Cleve Moler, professeur de mathématique à l'université du Nouveau-Mexique puis à Stanford, pour permettre aux étudiants de travailler à partir d'un outil de programmation de haut niveau et sans apprendre le Fortran ou le C.
- Matlab signifie **Matrix laboratory** (laboratoire des matrices).
- Matlab est un langage avec interface graphique pour le calcul scientifique, l'analyse de données, leur visualisation, le développement d'algorithmes…
- Un outil numérique puissant pour la modélisation de systèmes physiques, la simulation de modèles mathématiques, la conception et la validation (tests en simulation et expérimentation) d'applications.
- Il existe deux modes de fonctionnement Matlab: **mode interactif** (*command window*) et **mode exécutif** (*M file: Script* et *Function*)**.**

Présentation Générale de Matlab

❑ Interface de Matlab

Le logiciel propose un environnement de travail composé de multiples fenêtres. Nous pouvons distinguer quatre blocs :

- **Command window** (console d'exécution): à l'invite de commande **>>**, l'utilisateur peut entrer les instructions à exécuter. Il s'agit de la fenêtre principale de l'interface (**mode interactif**).
- **Current directory** (répertoire courant): permet de naviguer et de visualiser le contenu du répertoire courant de l'utilisateur. Les programmes de l'utilisateur doivent être situés dans ce répertoire pour être visible et donc exécutable.
- **Workspace** (espace de travail): permet de visualiser les variables définies, leur type, la taille occupée en mémoire…
- **Command history**: historique des commandes que l'utilisateur a exécutées. Il est possible de faire glisser ces commandes vers la fenêtre de commande.

❑ Interface de Matlab

❑ Toolboxes de Matlab

Le logiciel de base peut être complété par de multiples *toolboxes*, (boîtes à outils). Celles-ci sont des bibliothèques de fonctions dédiées à des domaines particuliers. Nous pouvons citer par exemple : l'Automatique, le traitement du signal, l'analyse statistique, l'optimisation…

Voici une liste non exhaustive de toolboxes, montrant la diversité des fonctionnalités de Matlab:

Control System Toolbox	Symbolic Math Toolbox	Signal Processing Toolbox
Neural Network Toolbox	Optimization Toolbox	Parallel Computing Toolbox
Statistics Toolbox	Fuzzy Logic Toolbox	Image Processing Toolbox
Aerospace Toolbox	Data Acquisition Toolbox	Bioinformatics Toolbox
MATLAB Compiler	Vehicle Network Toolbox	Model-Based Calibration Toolbox
Financial Toolbox	RF Toolbox	System Identification Toolbox

Commandes de Base en Matlab

❑ Opérations de base

On peut saisir des commandes interprétées par Matlab en mode interactif dans la *Command Window,* et le Matlab les exécute comme le ferait une calculatrice:

Exemple:

```
>> 3*5
ans =
      15
```

Ici *ans* (pour *answe*r) est une variable contient toujours le résultat de la dernière opération réalisée.

A la validation de l'instruction, l'interface affiche le résultat de cette dernière. Afin d'alléger l'affichage, un point-virgule « ; » en fin de commande empêche le renvoie du résultat dans la fenêtre (évidemment l'instruction est toujours exécutée).

```
>> 3*5;
>>
```

Le calcul a été effectué mais le résultat n'est pas affiché.

❑ Opérations de base

Les opérations de base sont résumées dans le tableau suivant:

Symbole	Description	Exemple
+	Addition des variables	>> 3+7 >> ans = 10
-	Soustraction des variables	>> 3-7 >> ans = -4
*	Multiplication des variables	>> 3*7 >> ans = 21
/	Division à droite des variables	>> 3/7 >> ans = 0.42857
\	Division à gauche des variables	>> 3\7 >> ans = 2.3333
^	La puissance	>> 3^7 >> ans = 2187

Il est possible de stoker des valeurs numériques et des opérations dans des variables. Le symbole d'affectation est le signe « = ». (g.e. x=1+2;).

❑ Constantes prédéfinies

Il existe des symboles auxquels sont associés des valeurs prédéfinies. En voici quelques uns:

Symbole	Signification	Valeur
`pi`	Nombre π	3.141592...
`i` ou `j`	Nombre complexe	$\sqrt{-1}$
`realmax`	Plus grand nombre flottant codable	1.7977e+308
`realmin`	Plus petit nombre flottant codable	2.2251e-308

Attention: ces valeurs peuvent être écrasées si le symbole est redéfini.

```
>> pi=1
pi =
     1
```

❑ Commandes clear et clc

La commande « ***clear*** » permet de supprimer une variable du *Workspace* (« ***clear all*** » les supprime toutes).
Toutes les commandes tapées dans la *Command Windows* peuvent être retrouvées et éditées grâce aux touches de direction. Appuyez sur la touche ⇧ pour remonter dans les commandes précédentes, et ⇩ pour redescendre.

Exemple:

```
>> pi=1
pi =
     1
>> clear pi
>> pi
ans =
     3.1416
```

La commande « *clc* » permet de supprimer la *Command Windows*, mais les variables restent toujours stockées dans le *Workspace*.

❑ Commandes who et whos

Les commandes « ***who*** » et « ***whos*** » permettent d'afficher les variables actives du *Workspace*.

La commande *who* affiche le nom des variables actives.
La commande *whos* donne plus d'informations: le nom, la taille du tableau (nombre de lignes et de colonnes) associé, l'espace mémoire utilisé (en Bytes) et la classe des données (principalement *double array* s'il s'agit d'un tableau de valeurs réelles ou complexes et *char* s'il s'agit d'un tableau de caractères).

Exemple :

```
>> x = 4
>> y = 2
>> x + y
```

```
>> whos
Name        Size        Bytes        Class
ans         1x1         8            double
x           1x1         8            double
y           1x1         8            double
```

❑ Commandes %, save et load

La commande «**%** » est utilisée pour écrire un commentaire (les lignes après % ne seront pas exécutées par Matlab).
La commande «***save*** » est utilisée pour sauvegarder les variables actives dans un fichier *.mat.*

Exemple:

```
>> x= (15/2)*4; % valeur de x
>> y=x/2; % valeur de y
>> save Results % Resuts.mat est un fichier de resultats
```

On peut sauvegarder une seule variable en utilisant *save* (non de fichier, nom de var).

```
>> save ('xonly', 'x')
```

Pour lire un fichier .mat, on utilise la commende «***load***»:

Exemple:

```
>> load ('Results.mat')
>> load Results
>> load Results.mat
```

❑ Formats d'affichage

- **Format short:** (automatique) 4 chiffres après le virgule.
- **Format long:** 15 chiffres après le virgule.
- **Format short (/ ou long) e:** scientifique de 4/15 chiffres après le virgule.
- **Format short (/ ou long) g:** valeur fixe de 4/15 chiffres après le virgule.
- **Format rat:** format fraction.
- **Format hex:** codage hexadécimale.

Exemples :

```
>> format short; pi

ans =

    3.1416
```

```
>> format long; pi

ans =

   3.141592653589793
```

```
>> format short e; pi

ans =

   3.1416e+00
```

```
>> format rat; pi

ans =

     355/113
```

```
>> format hex; pi

ans =

   400921fb54442d18
```

```
>> format short g;pi

ans =

        3.1416
```

❑ Types de variables

Il existe 4 types principaux des variables en Matlab:

- **Variable réel:** g.e. x = 1; y = 5.6;
- **Variable complexe:** e.g. z = 2 + 2*i; L = 1-j; m = complex (1, -2);
- **Variable caractère:** e.g. t = ‘bonjour’;
- **Variable logique:** vrai (valeur = 1); fau (valeur = 0).

Exemple:

```
>> x = 2; z = 2+i; L= complex (-2, 3); rep = 'oui'; x==1;
>> whos
  Name        Size                Bytes  Class     Attributes

  L           1x1                    16  double    complex
  ans         1x1                     1  logical
  rep         1x3                     6  char
  x           1x1                     8  double
  z           1x1                    16  double    complex
```

❑ Fonctions de base

`log(x)`	:	logarithme néperien de `x`,
`log10(x)`	:	logarithme en base 10 de `x`,
`exp(x)`	:	exponentielle de `x`,
`sqrt(x)`	:	racine carrée de `x` (s'obtient aussi par `x.^0.5`),
`abs(x)`	:	valeur absolue de `x`,
`conj(z)`	:	le conjugué de `z`,
`abs(z)`	:	le module de `z`,
`angle(z)`	:	argument de `z`,
`real(z)`	:	partie réelle de `z`,
`imag(z)`	:	partie imaginaire de `z`.
`round(x)`	:	entier le plus proche de `x`,
`floor(x)`	:	arrondi par défaut,
`ceil(x)`	:	arrondi par excès,
`fix(x)`	:	arrondi par défaut un réel

❑ Fonctions de base

Fonctions trigonométriques

sin	cos	tan	asin	acos	atan
sinh	cosh	tanh	asinh	acosh	atanh

Exemple:

```
>> x= sqrt(5)*sind(90)+exp(1)
x =
        4.9543
>> ceil(x)
ans =
     5
```

Pour les angles en dégrées on ajoute d à la fin de la fonction trigonométrique: e.g. cosd, sind …

Remarque: les commandes *help*, *lookfor* et *doc* peuvent être utilisées pour obtenir l'aide sur les fonctions utilisées.

Vecteurs et Matrices

❑ Vecteurs en Matlab

Matlab est essentiellement basé sur les matrices. Une variable scalaire est une matrice de dimension 1 x 1 et un vecteur est une matrice de dimension 1 x n (vecteur ligne) ou n x 1 (vecteur colonne).

Il existe plusieurs façons de créer un vecteur:

Exemple:

```
>> V=[1 2 3]

V =

     1     2     3
```

```
>> B=[-1,0.5,5]

B =

    -1.0000    0.5000    5.0000
```

L'ensemble des composantes est donné entre crochets et les valeurs sont séparées par un espace (ou une virgule « , »).

Nous avons ici défini un vecteur ligne. Un vecteur colonne est créé en utilisant un point-virgule « ; »:

```
>> K=[4;6;8]

K =

     4
     6
     8
```

❑ Vecteurs en Matlab

Bien que simple, cette méthode n'est pas pratique pour définir des vecteurs de taille importante. Une seconde méthode utilise l'opérateur deux-points « : ». Il permet de discrétiser un intervalle avec un pas constant:

Exemple:

```
>> V=[1:5]

V =

     1     2     3     4     5
```

En générale, la syntaxe est de la forme:

vecteur = valeur_initial : pas : valeur_finale

Par défaut, le pas est égal à 1.

Exemple:

```
>> V2=[0:0.2:0.5]

V2 =

         0    0.2000    0.4000
```

❑ Vecteurs en Matlab

On peut générer automatiquement d'un vecteur en utilisant une fonction prédéfinie: *linspace.* La syntaxe de cette fonction est donnée par:

Vecteur = linspace (début, fin, Nbr des éléments)

Exemple :

```
>> V = linspace (0, 5 , 4)

V =

         0       1.6667       3.3333            5
```

On peut accéder aux différents éléments d'un tableau en spécifiant un (ou des) indice(s) entre parenthèses.

Exemple :

```
>> v = [6 4 -1 3 7 0.3];
>> v(3)
ans =
      -1

>> v(2:4)
ans =
       4    -1     3
```

❑ Fonctions utiles sur les vecteurs

Quelques fonctions usuelles liées à l'utilisation des tableaux:

`length(v)`	renvoie la taille du tableau.
`max(v)`	renvoie la valeur maximale du tableau.
`min(v)`	renvoie la valeur minimale du tableau.
`mean(v)`	renvoie la valeur moyenne des éléments du tableau.
`sum(v)`	calcul la somme des éléments du tableau.
`prod(v)`	calcul le produit des éléments du tableau.
`sort(v)`	range les éléments du tableau dans l'ordre croissant.

Remarque: Toutes les fonctions mathématiques vues précédemment sont applicables aux variables de type vecteur. Dans ce cas, la fonction est opérée sur chacun des éléments du vecteur (e.g. cos(vecteur)).

❑ Polynômes

Sous Matlab, le polynôme de degré n: $\boldsymbol{p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0}$ est défini par un vecteur p de dimension *n+1* contenant les coefficients $\{a_i\}_{i=0,\ldots n}$ rangés dans l'ordre décroissant des indices.

C'est-à-dire que l'on a : $p(1) = a_n, \ldots, p(n+1) = a_0$.

La commande ***polyval*** permet d'évaluer le polynôme p (la fonction polynômiale) en des points donnés.

La commande ***roots*** permet d'obtenir les racines du polynôme p.

Exemple:
Soit $f(x) = x^2 - 1$

- évaluer $f(0.5)$, et
- résoudre l'équation $f(x) = 0$.

```
>> f = [1 0 -1];
>> polyval (f, 0.5)

ans =

        -0.75

>> roots (f)

ans =

    -1
     1
```

❑ Matrices en Matlab

La définition d'une matrice est délimitée par des crochets « [ ] ». Les différents éléments d'une ligne sont séparés par un espace (ou virgule) et les différentes lignes sont séparées par des points virgules « ; ». Ainsi pour définir une variable matricielle telle que M:

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

on écrira:

```
>> M = [1 2 3 ; 4 5 6 ; 7 8 9];
```

ou

```
>> M = [1,2,3 ; 4,5,6 ; 7,8,9];
```

Pour accéder à un élément *i, j* de la matrice, on écrit *M (i,j)*:

Exemple:

```
>> M(2,3)
ans =
     6
```

❑ Matrices en Matlab

Matrices particulières

- **Matrice nulle:**

```
>> Z = zeros(2,3)
Z =
     0     0     0
     0     0     0
```

- **Matrice pleine de 1:**

```
>> U = ones(4,3)
U =
     1     1     1
     1     1     1
     1     1     1
     1     1     1
```

- **Matrice identité:**

```
>> I  = eye(3)
I =
     1     0     0
     0     1     0
     0     0     1
```

- **Matrice aléatoire (éléments compris entre 0 et 1):**

```
>> R = rand(2,2)
R =
    0.9575    0.1576
    0.9649    0.9706
```

- **Matrice diagonale:**

```
>> D = diag([2,4,0,7])
D =
     2     0     0     0
     0     4     0     0
     0     0     0     0
     0     0     0     7
```

❑ Matrices en Matlab

Extraction de sous-tableaux :

Il est souvent utile d'extraire des blocs d'un tableau existant. La syntaxe générale est la suivante (pour un tableau à deux dimensions):
tableau(début:fin, début:fin)

Exemple:

```
>> M = [1 2 3 ; 4 5 6 ; 7 8 9];
>> M(1:2,2:3)
ans =
     2     3
     5     6
```

$$M = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Le caractère « : » seul, signifie toute la longueur est extraite. De cette façon, on peut isoler une ligne, ou une colonne, complète

Exemple:

```
>> M(1:2,:)
ans =
     1     2     3
     4     5     6
```

❑ Fonctions utiles sur les matrices

« `size(M)` » renvoie les dimensions de la matrice.

« `max(M)` » renvoie un vecteur-ligne contenant les valeurs maximales associées à chaque colonne.

« `min(M)` » renvoie un vecteur-ligne contenant les valeurs minimales associées à chaque colonne.

« `rank(M)` » renvoie le rang de la matrice.

« `det(M)` » renvoie le déterminant de la matrice.

« `diag(M)` » extrait la diagonale de la matrice.

« `triu(M)` » extrait la matrice-triangle supérieure de M. `tril` donne la matrice-triangle inférieure.

« `eig(M)` » renvoie un vecteur contenant les valeurs propres de la matrice.

❑ Opérations sur les matrices

+	addition
-	soustraction
*	produit
/	division à droite
\	division à gauche
^	puissance
'	transposition
inv()	inversion

Remarque:
Si l'on souhaite effectuer une opération, non pas matricielle, mais éléments par éléments, l'opérateur doit être précédé d'un point « . »: .* ./ .^ .\

❑ Exemple – Résolution d'un système d'équations linéaires

Soit le système: $\begin{cases} 2x + y = -5 \\ 4x - 3y + 2z = 0 \\ x + 2y - z = 1 \end{cases}$

Ecriture sous forme matricielle AX=B: $\begin{bmatrix} 2 & 1 & 0 \\ 4 & -3 & 2 \\ 1 & 2 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \\ 1 \end{bmatrix}$

Si A est une matrice inversible alors le système admet une solution unique qui s'exprime par: $X = A^{-1}B$

```
>> A = [2 1 0 ; 4 -3 2 ; 1 2 -1]
>> B = [-5;0;1]
>> X = inv(A)*B
X =
    1.7500
   -8.5000
  -16.2500
```

Solutions: $x = 1.75$ $y = -8.5$ $z = -16.25$

❑ Autres opérations sur les matrices

`det(A)` : renvoie le déterminant de la matrice carrée `A`.

`trace(A)` : renvoie la trace de la matrice `A`.

`expm(A)` : renvoie l'exponentielle matricielle de `A`.

`norm(A)` : renvoie la norme 2 de la matrice `A`.

`norm(A,2)` : même chose que `norm(A)`.

`norm(A,1)` : norme 1 de la matrice `A`, $\|A\|_1 = \max_{1\leq j\leq n} \sum_{1\leq i\leq n} |a_{ij}|$.

`norm(A,inf)` : norme infini de la matrice `A`, $\|A\|_\infty = \max_{1\leq i\leq n} \sum_{1\leq j\leq n} |a_{ij}|$.

Programmation sur Matlab

❑ Script et Function

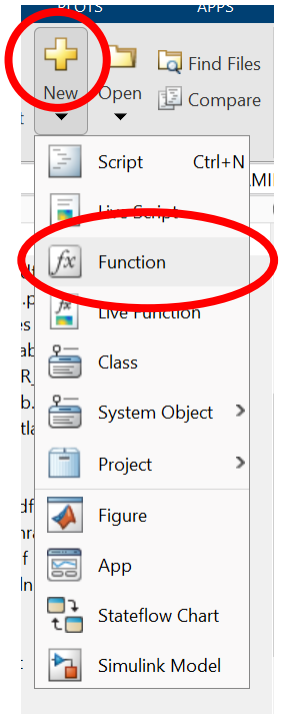
- Il est possible d'enregistrer une séquence d'instructions dans un fichier (appelé «*M-file* », extension *.m*) et de les faire exécuter par Matlab.
- On distingue deux types de fichiers *M-file*, les fichiers de ***scripts*** et les fichiers de fonctions (***Function***).
- Un *script* est un ensemble d'instructions Matlab qui joue le rôle du programme principal.
- Les fichiers de fonctions permettent à l'utilisateur de définir des fonctions qui ne figurent pas parmi les fonctions prédéfinies en Matlab et de les utiliser de la même manière que ces dernières.

La syntaxe de définition d'une fonction (nommée fonc par exemple) est:

```
function [Output1, . . . , OutputN] = fonc (input1, . . . , inputN)
séquence d'instructions
end
```

❑ Function

Exemple d'une *Function*:
Écrire une fonction pour calculer la solution d'un système d'équations linéaires (le nom de la fonction par exemple est SolSystem)




```
function [x] = SolSystem(matrix,vector)
x= inv(matrix)*vector;
end
```

On peut appeler la fonction sur *command windows :*

```
>> y = SolSystem([1 2 -1; -2 1 3; 1 1 -1],[1; 0; -1])

y =

   -11
     2
    -8
```

❑ Instructions de contrôle

- **Instruction conditionnelle IF:**

Objectif : Exécuter une instruction lorsque une condition est vérifiée.

Syntaxe :

```
if expression1
        instructions1 …
elseif expression2
        instructions2 …
else
        instructions3 …
end
```

Exemple:

```
x=input('Entrez une valeur réel:');
if x > 0
    disp('x est positif');
elseif x == 0
    disp('x est nul');
else
    disp('x est négatif');
end
```

```
Entrez une valeur réel:-11
x est négatif
```

❑ Opérateurs de comparaison et opérateurs logiques

Les opérateurs de comparaison sont résumées comme suit:

==	:	égal à (x == y)
>	:	strictement plus grand que (x >y)
<	:	strictement plus petit que (x <y)
> =	:	plus grand ou égal à (x ≥ y)
< =	:	plus petit ou égal à (x ≤ y)
~=	:	différent de (x ~= y)

Les opérateurs logiques sont résumées ci-dessous:

&	:	et (x & y)
|	:	ou (x | y)
~	:	non (~ x)

❑ Instructions de contrôle

- **Boucle FOR:**

Objectif : Répéter une instruction N fois.

Syntaxe :

for indice = borne_inf : pas :borne_sup
séquence d'instructions
end

Exemples:

```
>> x(1)=0.0;
>> for i= 1:4
x(i+1)= x(i)+0.5;
end
>> disp (x)
         0         0.5           1         1.5           2
```

```
for r = 1.1:-0.1:0.75
    disp(['r = ', num2str(r)])
end
```

```
r = 1.1
r = 1
r = 0.9
r = 0.8
```

❑ Instructions de contrôle

- **Boucle FOR:**

Exemple d'application: résolution la formule de Colebrook en utilisant la boucle *FOR*.

$$\frac{1}{\sqrt{f}} = -2.0 \log\left(\frac{2.51}{Re\sqrt{f}} + \frac{\epsilon/D}{3.7}\right)$$

```
clear all; clc;
Re=input('Entrer le nombre de Reynolds:');
e=input('Entrer la rugosité relative:');
if Re <= 2300
   error ('Formule de Colebrok est valide pour les écoulements turbulents')
end
f=1;
for k=1:100
f=1/(-2*log10((2.51/(Re*sqrt(f)))+(e/3.7)))^2;
end
disp([' ===> f = ', num2str(f) ])
```

❑ Instructions de contrôle

- **Boucle WHILE:**

Objectif : Répéter une instruction tant qu'une condition reste vérifiée.

Syntaxe : *while expression logique*
séquence d'instructions
end

Exemple :

```
n=1;
while n<100
x=n*0.05;
y(n)=5.75*cos(x);
z(n)=-3.4*sin(x);
n=n+1;
end
```

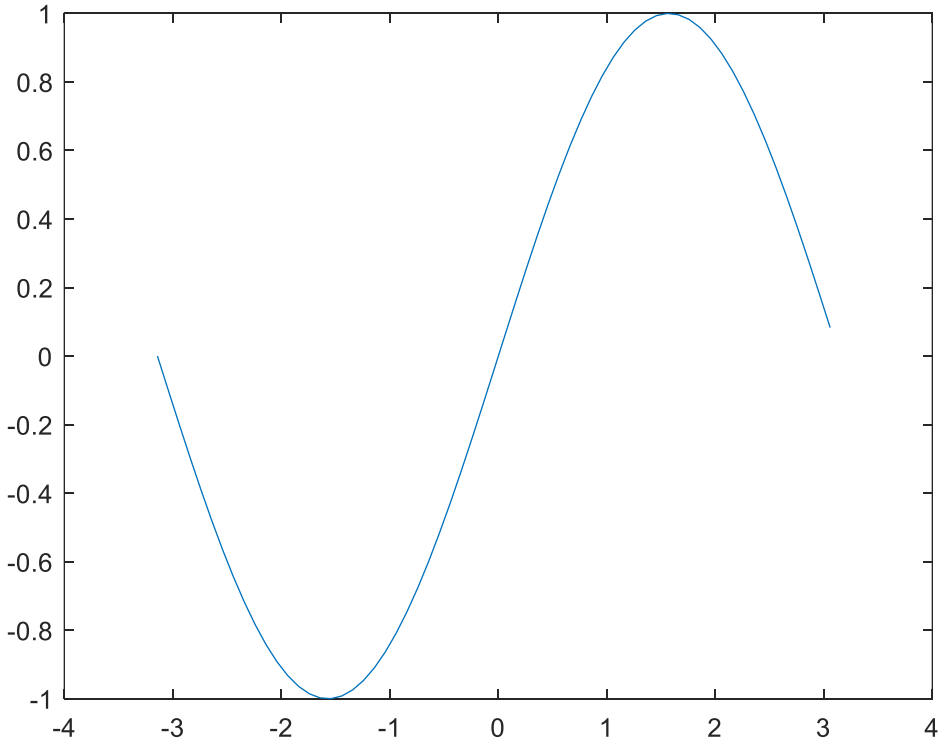
Représentation Graphique

❑ Graphiques 2D

La commande ***plot*** est utilisée pour tracer une courbe.

Exemple:

```
>> x = -pi : .01 : pi;
>> y = sin(x);
>> plot(x,y)
```

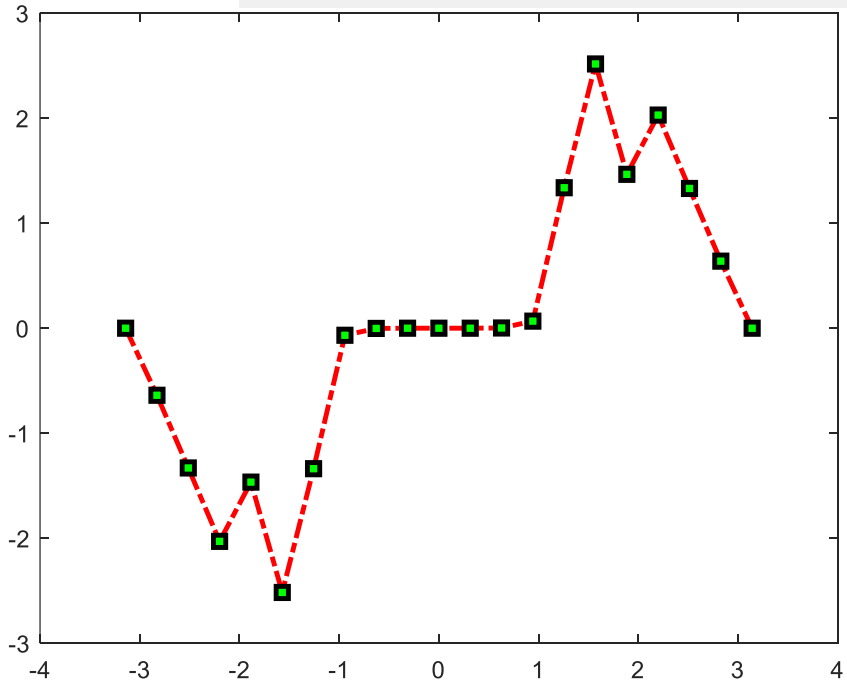


❑ Graphiques 2D

On peut changer les caractéristiques d'une courbe.

Exemple:

```
>> x = -pi:pi/10:pi;
>> y = tan(sin(x)) - sin(tan(x));
>> plot(x,y,'-.rs','LineWidth',2,...
                    'MarkerEdgeColor','k',...
                    'MarkerFaceColor','g',...
                    'MarkerSize',7)
```



❑ Graphiques 2D

Différents types de lignes, symboles de tracé et couleurs sont disponible :

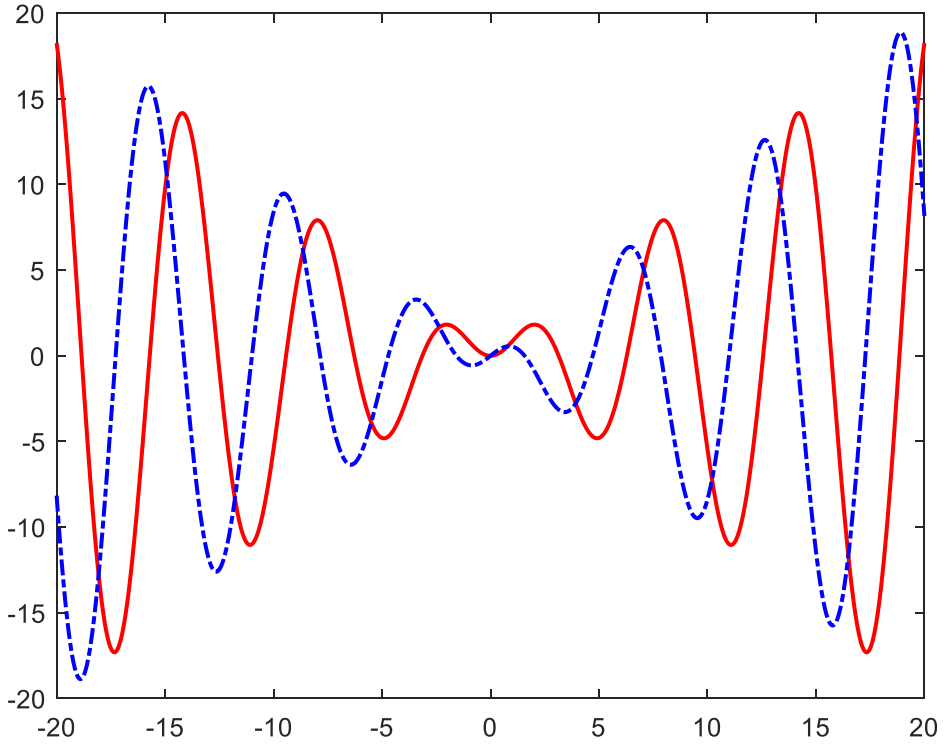
b	blue	.	point	-	solid
g	green	o	circle	:	dotted
r	red	x	x-mark	-.	dashdot
c	cyan	+	plus	--	dashed
m	magenta	*	star	(none)	no line
y	yellow	s	square		
k	black	d	diamond		
w	white	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		
		>	triangle (right)		
		p	pentagram		
		h	hexagram		

❑ Graphiques 2D

La commande ***hold on*** est utilisée pour tracer deux ou plusieurs courbes sur la même figure.
Les plot suivants se superposeront jusqu'à la désactivation par ***hold off*** ou la fermeture de la fenêtre.

Exemple:

```
x = linspace(-20,20,1000);
y1 = x.*sin(x);
plot(x,y1,'r-','LineWidth',1.5)
y2= x.*cos(x);
hold on
plot(x,y2,'b-.','LineWidth',1.5)
hold off
```

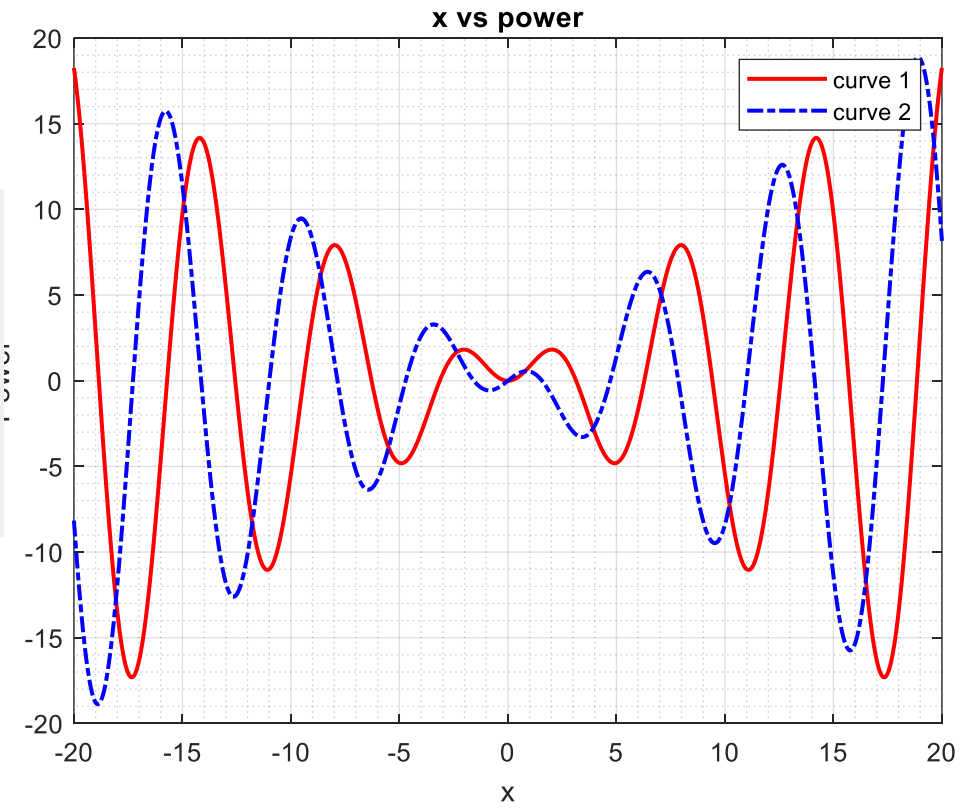


❑ Graphiques 2D

La mise en forme d'une représentation graphique (l'insertion de labels, légende, le dimensionnement des axes) peut être éditée directement en utilisant menus de l'interface de la figure (Edit et Insert), ou à partir de la *Command Window* en ligne de commande.

Exemple:

```
xlabel ('x')
ylabel('Power')
title('x vs power')
legend('curve 1','curve 2')
grid on
box on
grid minor
```

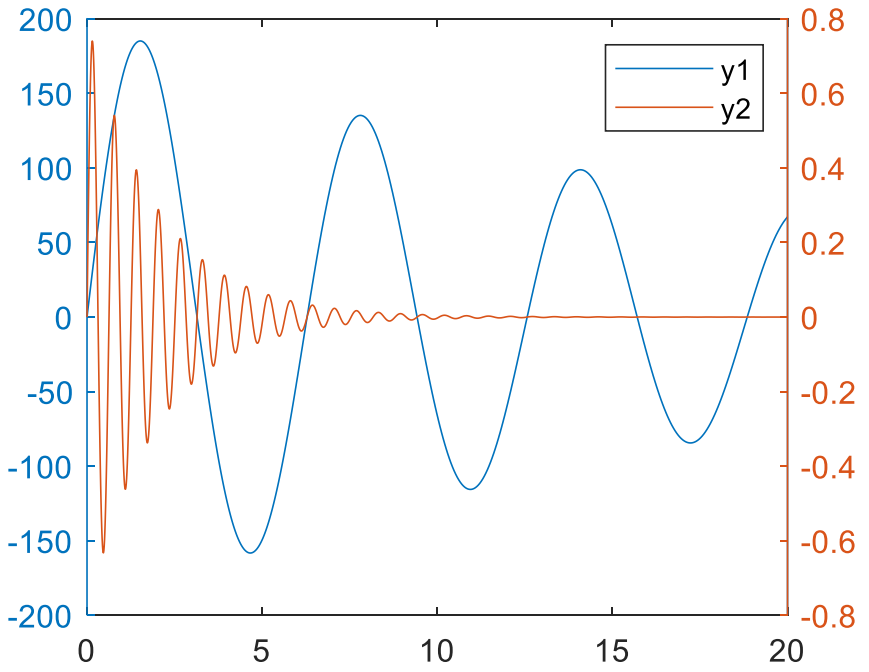


❑ Graphiques 2D

La commande ***plotyy*** est utilisée pour tracer deux ensembles de données sur la même figure avec deux axes sur y.

Exemple:

```
x = 0:0.01:20;
y1 = 200*exp(-0.05*x).*sin(x);
y2 = 0.8*exp(-0.5*x).*sin(10*x);
figure % Ouvrir nouvelle figure
plotyy(x,y1,x,y2)
legend('y1','y2')
```

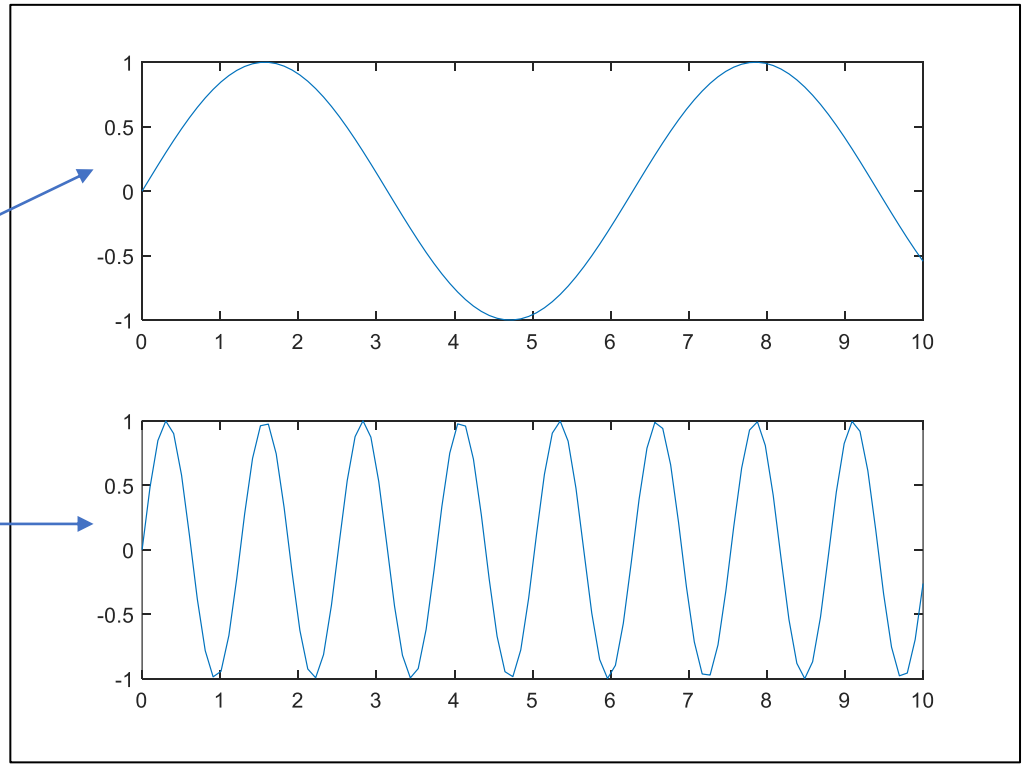


❑ Graphiques 2D

La commande ***subplot (ligne, colonne, num_figure)*** est utilisée pour tracer des figures dans la même fenêtre.

Exemple:

```
subplot(2,1,1);
x = linspace(0,10);
y1 = sin(x);
plot(x,y1)
subplot(2,1,2);
y2 = sin(5*x);
plot(x,y2)
```

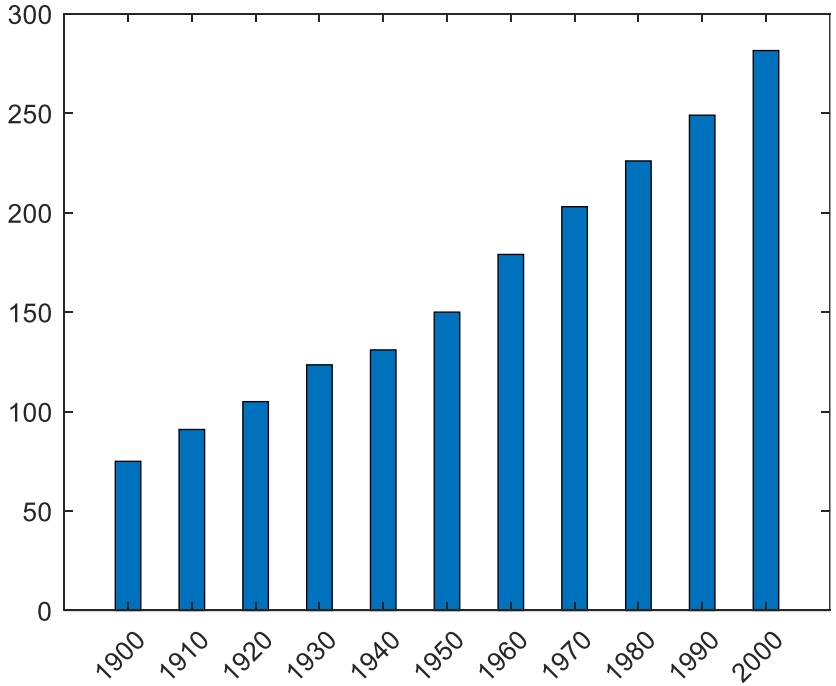


❑ Graphiques 2D

La commande ***bar*** est utilisée pour une représentation à barres (histogrammes).

Exemple:

```
x = 1900:10:2000;
y = [75 91 105 123.5 131 150 179 203 226 249 281.5];
bar(x,y,0.4)
```

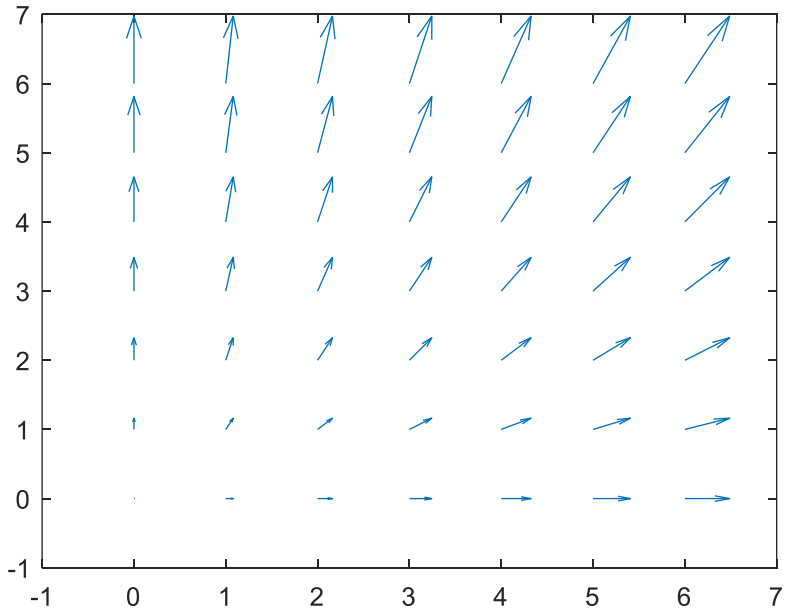


❑ Graphiques 2D

La commande ***quiver*** est utilisée pour tracer des flèches (vecteurs)

Exemple:

```
[X,Y] = meshgrid(0:6,0:6);
U = 0.25*X;
V = 0.5*Y;
quiver(X,Y,U,V)
```

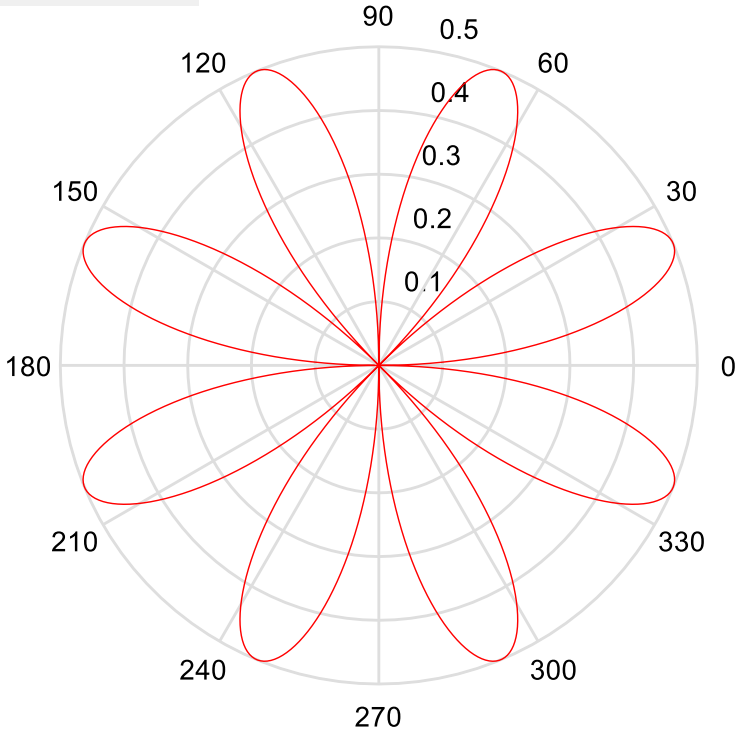


❑ Graphiques 2D

La commande ***polar (theta, rho)*** permet de tracer des graphiques polaires

Exemple:

```
figure ; % Nouvelle figure
t = 0 :.01 :2*pi ; %  Echelle du temps
polar(t,sin(2*t).*cos(2*t),'-r') % Graphique polaire
```

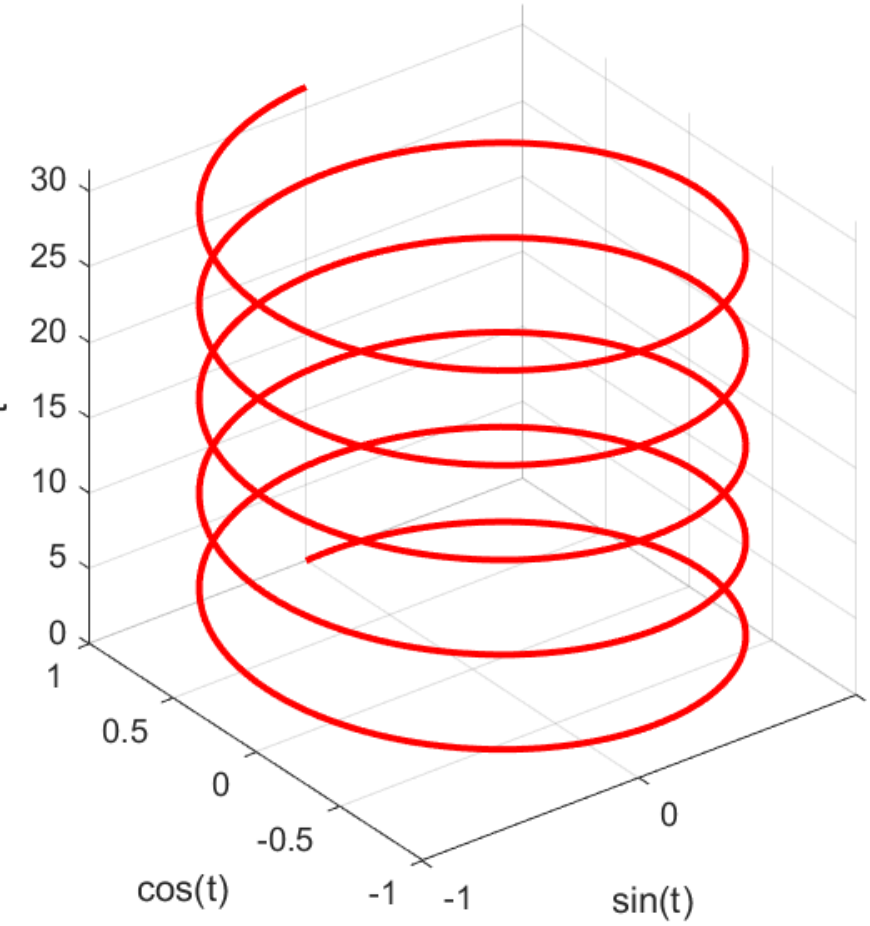


❑ Graphiques 3D

La commande ***plot3*** permet de tracer une courbe 3D.

Exemple:

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t,'r','LineWidth',2)
grid on
axis square
xlabel('sin(t)')
ylabel('cos(t)')
zlabel('t')
```

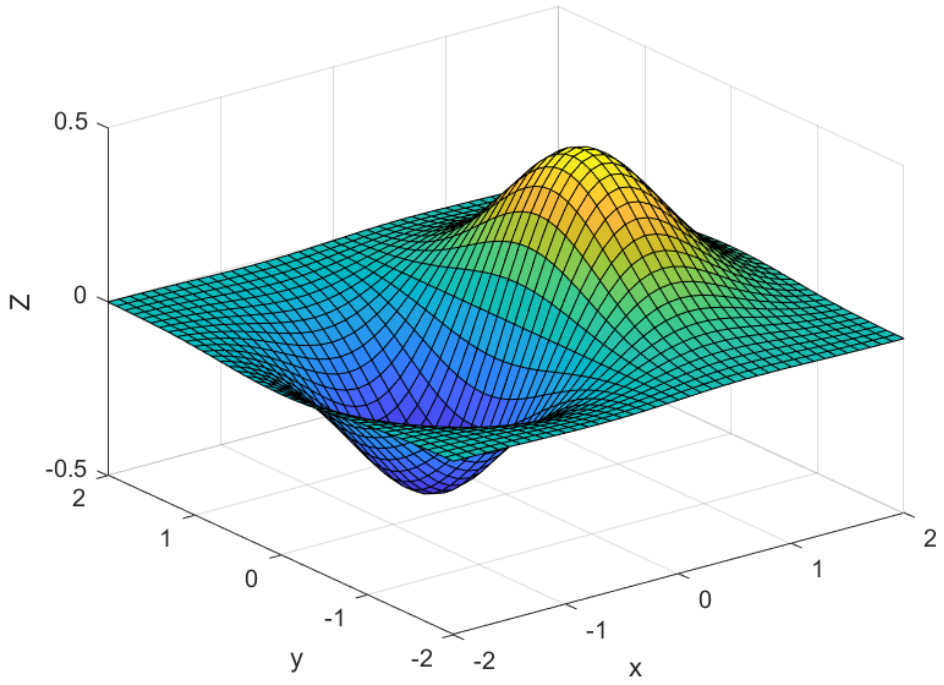


❑ Graphiques 3D

Commandes ***mesh*** et ***surf:*** Représentation par maillage dans un plan.

Exemple:

```
x=[-2:0.1:2];
y=[-2:0.1:2];
[X,Y]=meshgrid(x,y);
colormap([0 0 1]);
Z=X.*exp(-X.^2-Y.^2);
mesh(X,Y,Z)
colormap('default') ;
surf(X,Y,Z)
xlabel('x')
ylabel('y')
zlabel('Z')
```

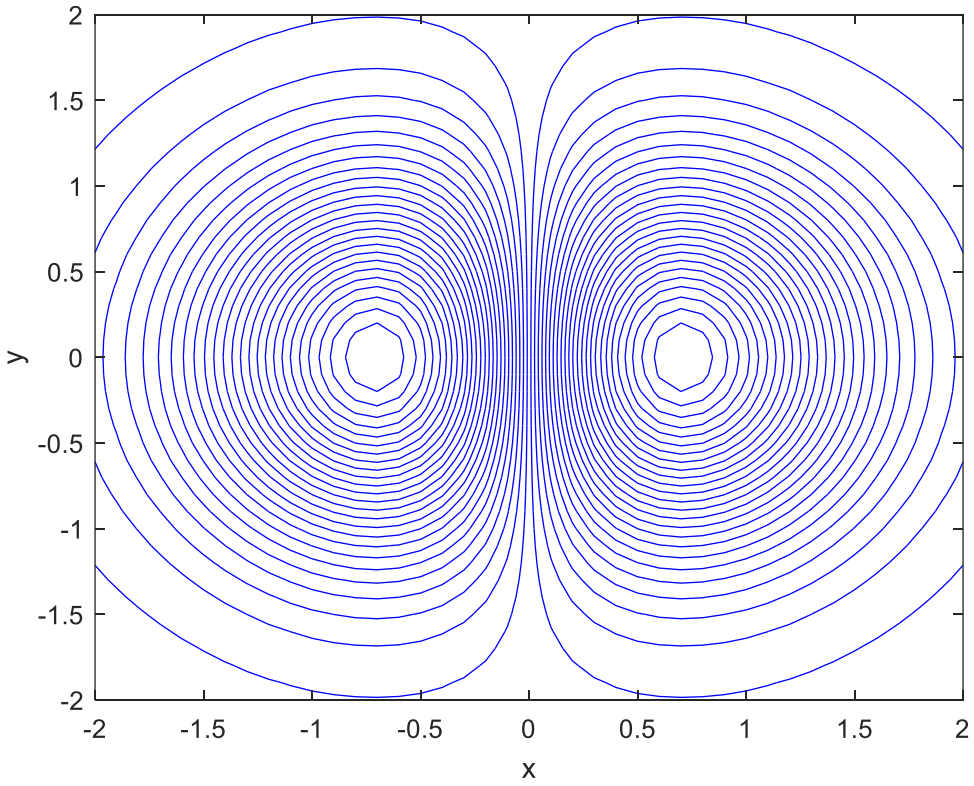


❑ Graphiques 3D

Commande ***contour****:* tracer dans le plan (x,y) les courbes z=cst d'une surface.

Exemple:

```
x=[-2:0.1:2];
y=[-2:0.1:2];
[X,Y]=meshgrid(x,y);
colormap([0 0 1]);
Z=X.*exp(-X.^2-Y.^2);
mesh(X,Y,Z)
contour(X,Y,Z,50)
xlabel('x')
ylabel('y')
zlabel('Z')
```

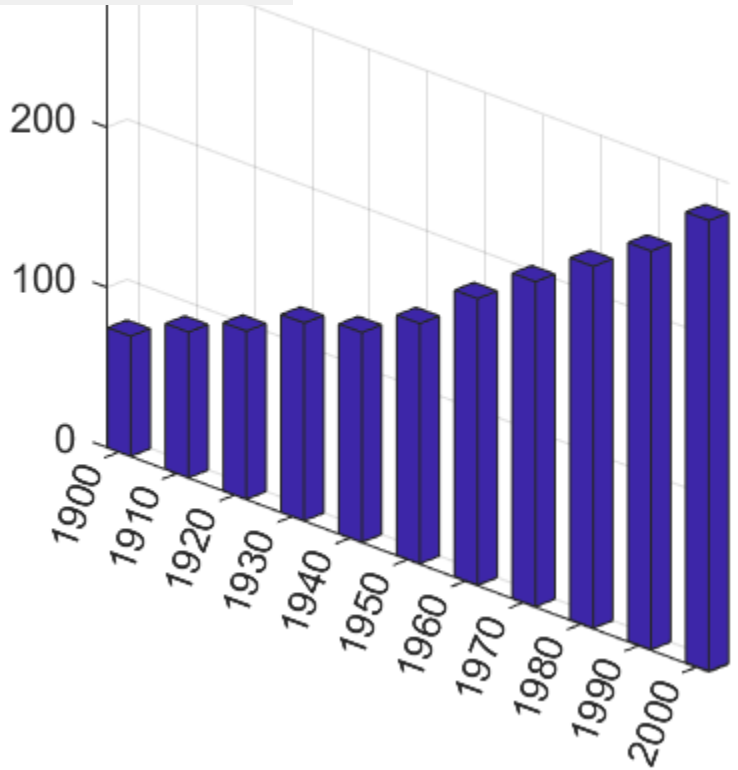


❑ Graphiques 3D

Commande ***bar3***: Représentation 3D à barres (histogrammes).

Exemple:

```
x = 1900:10:2000;
y = [75 91 105 123.5 131 150 179 203 226 249 281.5];
bar3(x,y,0.4)
```



Calcul Différentiel et Intégrale (Représentation Symbolique)

❑ Représentation symbolique

Matlab peut représenter symboliquement des équations mathématiques.

Pour définir une variable symbolique, on utilise la commande ***syms*** (i.e. *symbolic system*).

Simplification d'une équation: La commande ***simplify,*** permet de simplifier une fonction symbolique.

Exemple:

```
syms x ; % Declaration des symboles
g = cos(3*acos(x)) ; % Fonction à simplifier
simplify(g) % Simplification
```

```
ans =

4*x^3 - 3*x
```

❑ Dérivation et intégration

Les commandes ***diff*** et ***int*** sont utilisées pour déterminer la drivée et l'intégrale d'une fonction, dans la base symbolique, respectivement.

Exemple:

```
clear;clc;
syms x y
y = (1+x)/sin(x);
diff (y,x)
```

```
ans =

1/sin(x) - (cos(x)*(x + 1))/sin(x)^2
```

```
syms y ; % Declaration des symboles
f= 1/y;
int(f,y)
```

```
ans =

log(y)
```

❑ Racines d'une équation

La commande ***solve*** permet de déterminer les racines d'une équation.

Exemple #1:

```
syms x
y = solve(x^2+ x -1)
```

```
y =

- 5^(1/2)/2 - 1/2
  5^(1/2)/2 - 1/2
```

Exemple #2:

```
>> z = log (t)/t;
>> solve (z)

ans =

1
```

❑ Résolution des équations différentielles

La commande ***dsolve*** est utilisée pour résoudre une équation différentielle ou un système d'équations déférentielles.

Exemple – sans conditions initiales:

Soit l'équation différentielle suivante: $\frac{dF}{dx} = 1$

Solution sur Matlab:

```
>> syms F(x)
>> Eq = diff (F) == 1;
>> dsolve (Eq)

ans =

C1 + x
```

Solution générale : $F(x) = x + C_1$

❑ Résolution des équations différentielles

La commande ***dsolve*** est utilisée pour résoudre une équation différentielle ou un système d'équations déférentielles.

Exemple – avec conditions initiales:

Soit l'équation différentielle suivante: $\frac{dF}{dx} = 1$ Condition initiale: $F(0) = 1$

Solution sur Matlab:

```
>> syms F(x)
>> Eq = diff (F) == 1;
>> dsolve (Eq, 'F(0)=1')

ans =

x + 1
```

Solution particulière avec $F(0) = 1$ est: $F(x) = x + 1$

❑ Résolution des équations différentielles

La commande ***dsolve*** est utilisée pour résoudre une équation différentielle ou un système d'équations déférentielles.

Exemple – système d'équations différentielles:

Soit le système suivant:

$$\frac{du}{dt} = 3u + 4v,$$

$$\frac{dv}{dt} = -4u + 3v.$$

Solution sur Matlab:

```
syms u(t) v(t)
ode1 = diff(u) == 3*u + 4*v;
ode2 = diff(v) == -4*u + 3*v;
odes = [ode1; ode2];
S = dsolve(odes);
uSol(t) = S.u
vSol(t) = S.v
```

Solutions générales:

```
C2*cos(4*t)*exp(3*t) + C1*sin(4*t)*exp(3*t)
```

```
C1*cos(4*t)*exp(3*t) - C2*sin(4*t)*exp(3*t)
```

Avec les conditions initiales:

```
>> solutions = dsolve(odes,'u(0)=0','v(0)=1')
```

❑ Résolution des équations différentielles

Exemple – équation différentielle de second ordre:

Soit l'équation différentielle gouvernante la conduction de chaleur dans un mur 1D de conductivité thermique k en présence d'une source de chaleur q:

$$\frac{d^2T}{dx^2} = -\frac{q}{k}$$

Solution sur Matlab:

```
>> syms q k T(x)
>> Equation = diff(diff(T,x),x)==-q/k;
>> dsolve (Equation)

ans =

C2 + C1*x - (q*x^2)/(2*k)
```

Solution générale:

Solution particulière:

```
>> dsolve (Equation, 'T(0)=20','T(0.1)=35')

ans =

x*((0.05*q)/k + 150.0) - (q*x^2)/(2*k) + 20.0
```

Codes avec Interfaces Graphiques (GUI)

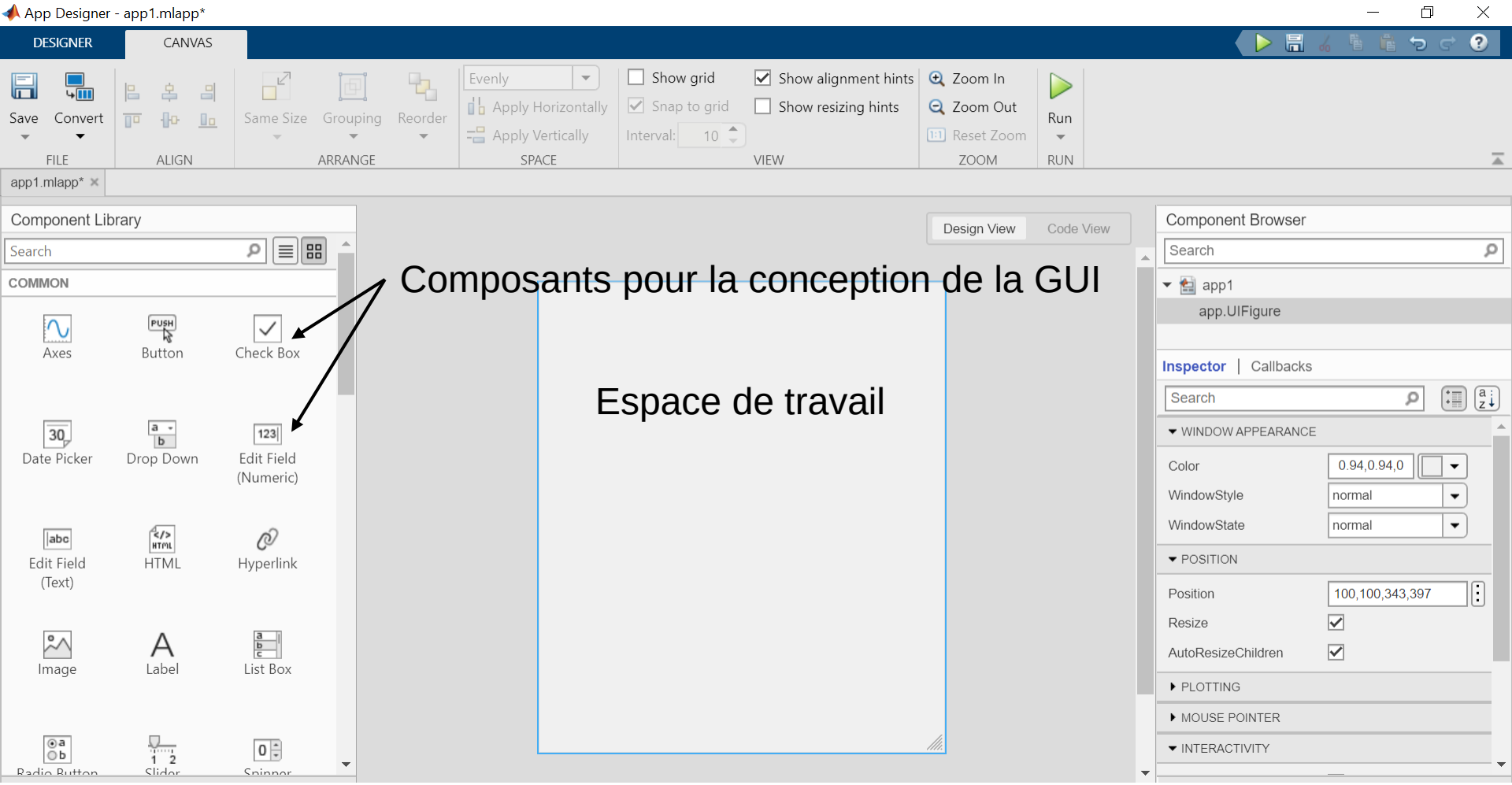
❑ Introduction

- *Matlab App Designer* permet de créer des codes avec une interface utilisateur graphique (GUI).
- L'interface graphique peut être conçue en glissant et déposant des composants visuels disponibles dans la bibliothèque des éléments.
- Le contenu du code est programmé par l'éditeur intégré.
- Le code peut être compilé et converti à un code de bureau ou à une application web en utilisant *Matlab Compiler*.
- Pour développer un code avec GUI en utilisant *App Designer*, il existe deux tâches principales:
  1. Création de la GUI : la présentation des composants visuels d'une interface utilisateur graphique, et
  2. Programmation du comportement des composant en utilisant l'éditeur intégré.

❑ Créer un code avec une GUI

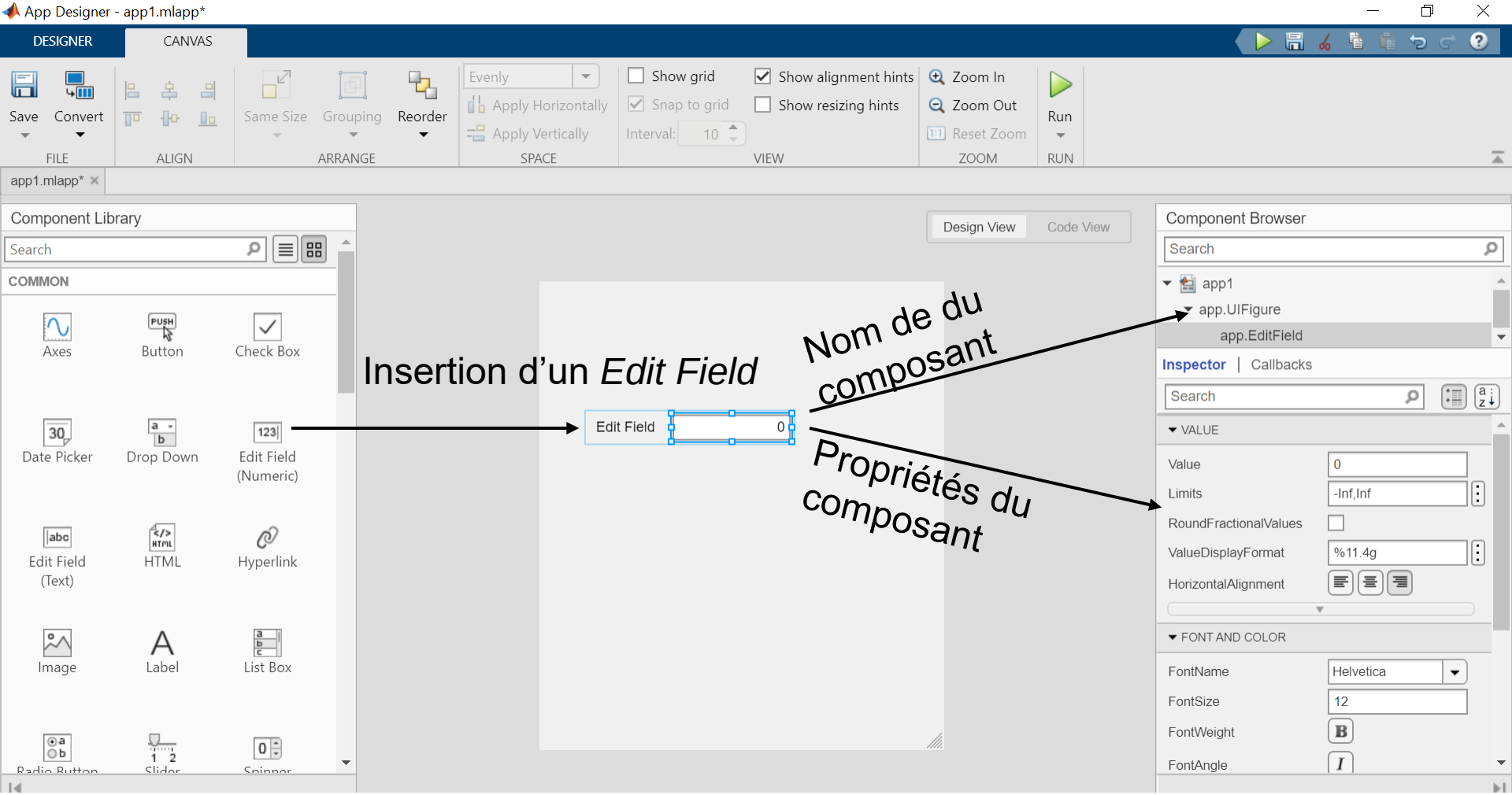
Pour ouvrir un nouveau projet *App Designer*, on clique sur *New* puis *App*.

❑ Créer un code avec une GUI



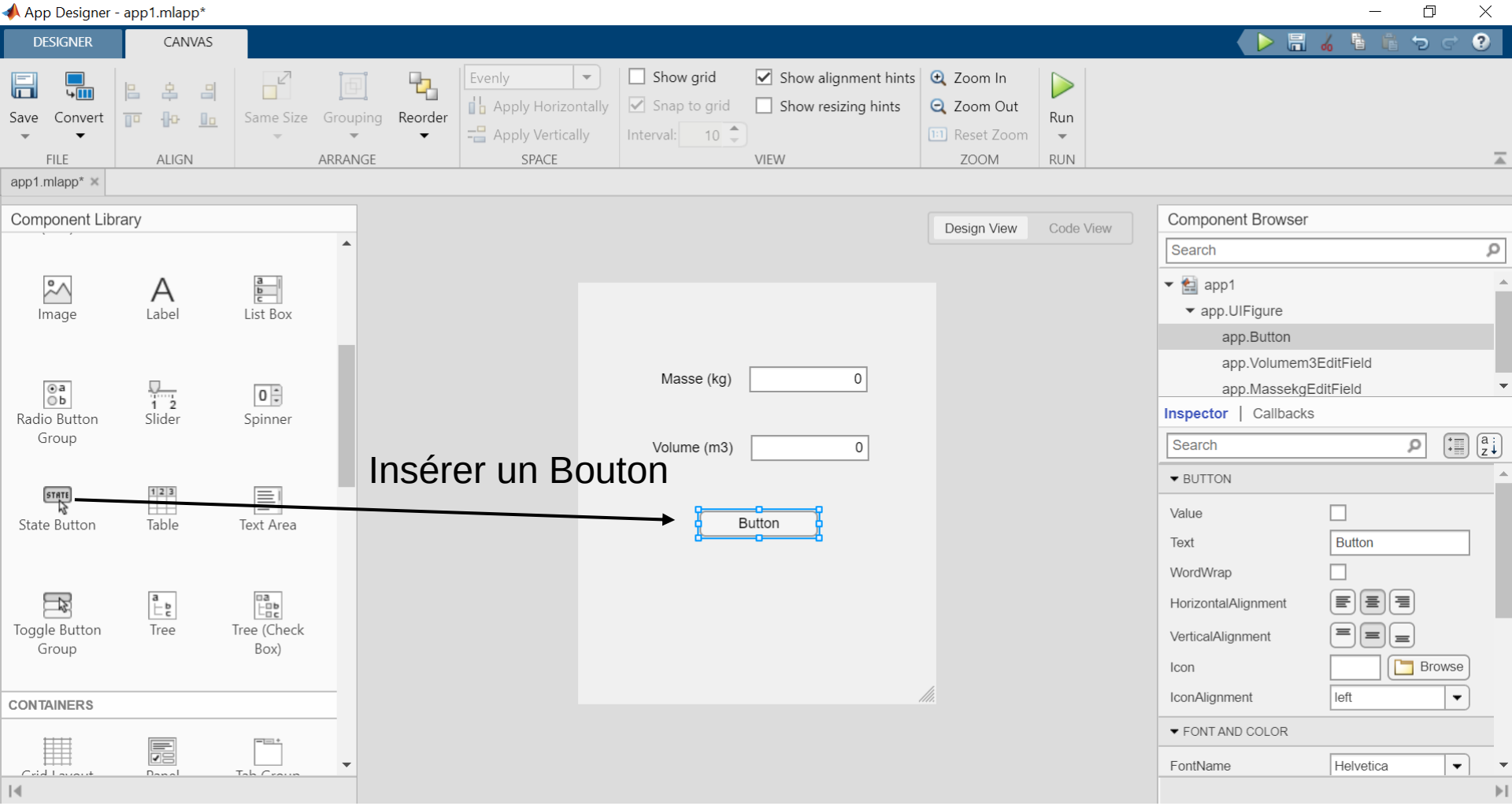
Exemple #1: une simple calculatrice de la masse volumique.

❑ Créer un code avec une GUI



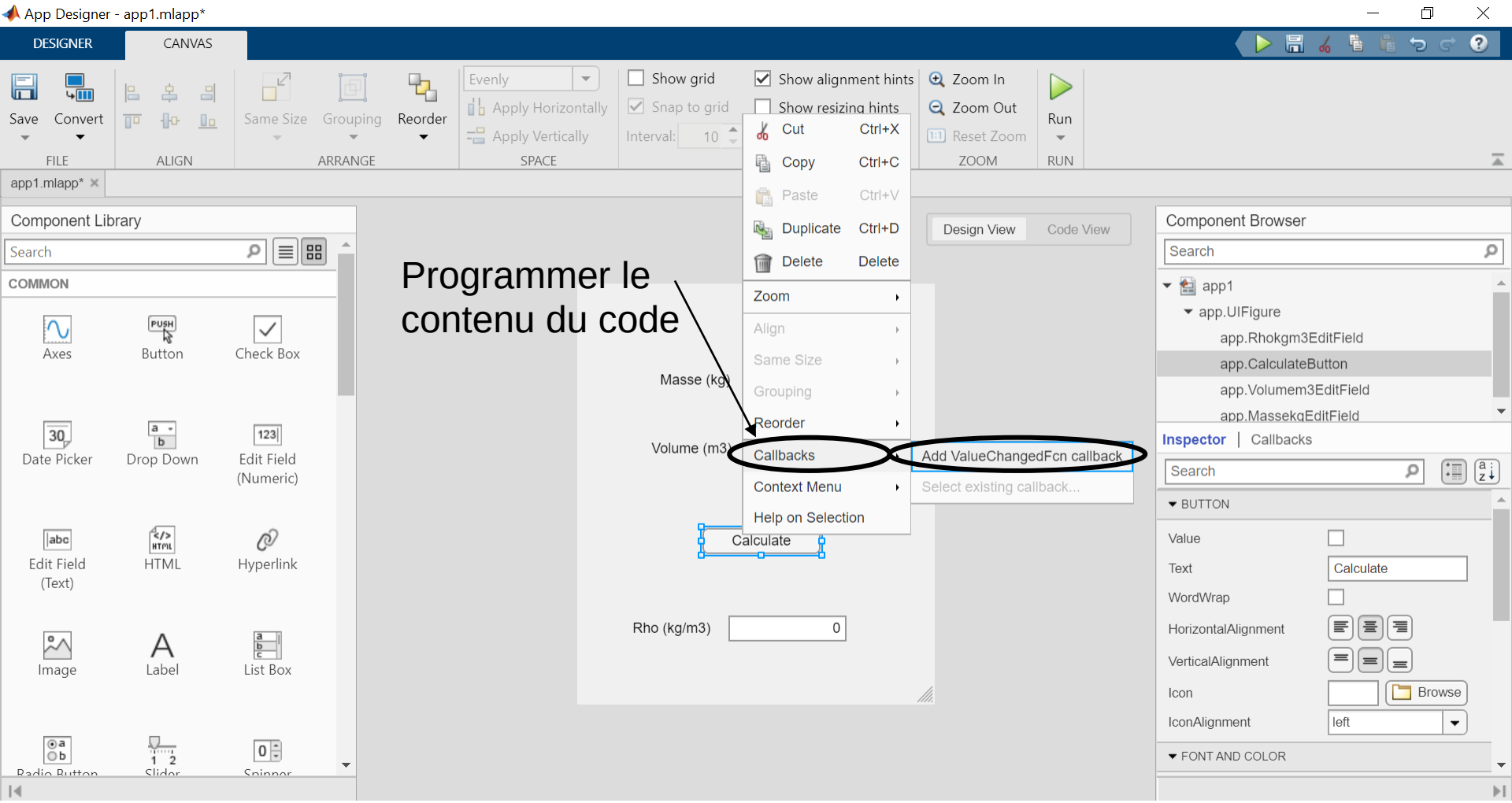
Exemple #1: une simple calculatrice de la masse volumique.

❑ Créer un code avec une GUI



Exemple #1: une simple calculatrice de la masse volumique.

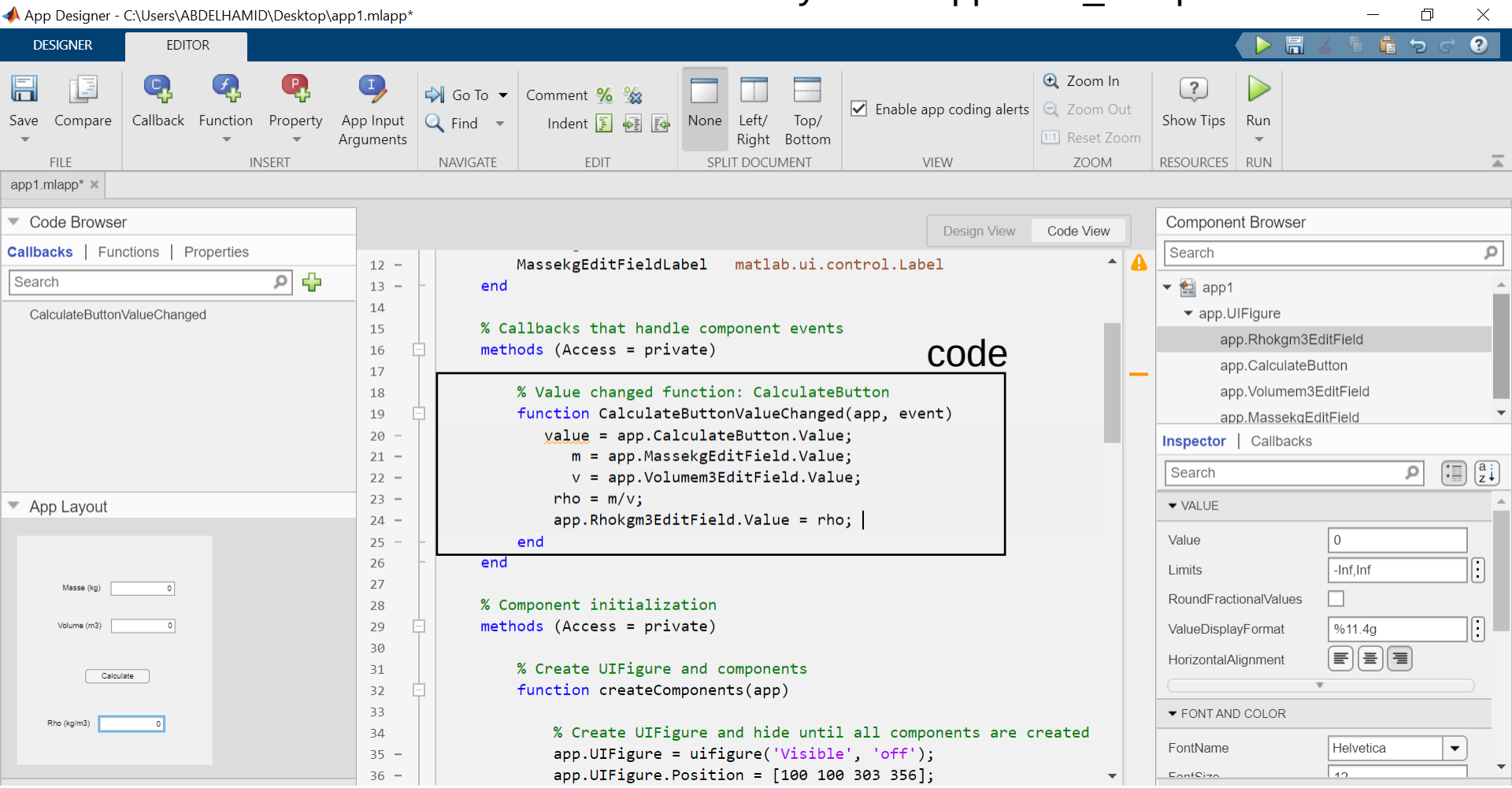
❑ Créer un code avec une GUI



Exemple #1: une simple calculatrice de la masse volumique.

❑ Créer un code avec une GUI

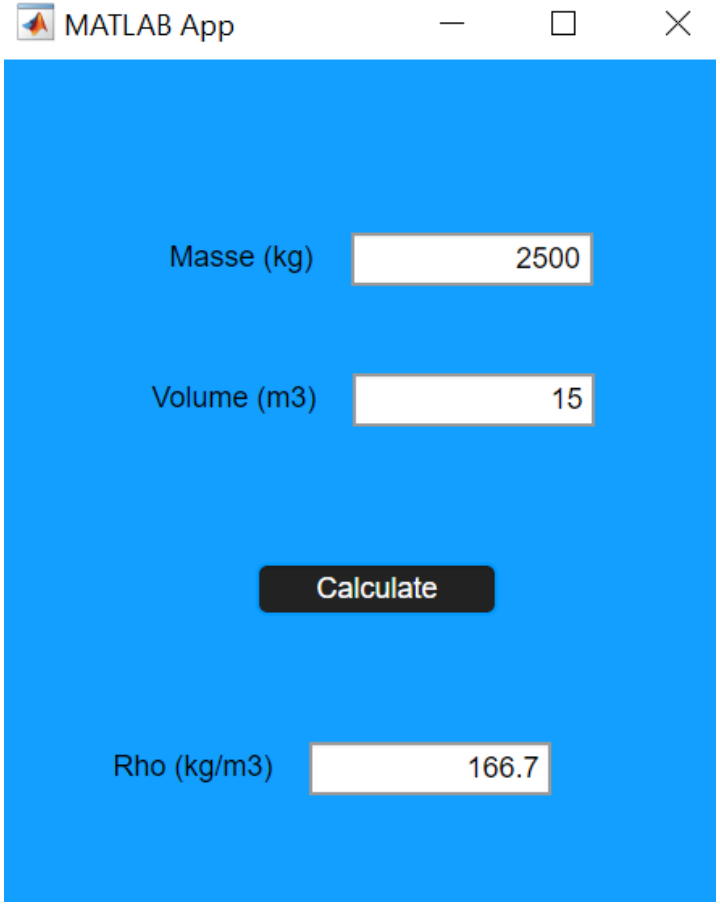
Syntaxe: app.Nom_composant.Value



Exemple #1: une simple calculatrice de la masse volumique.

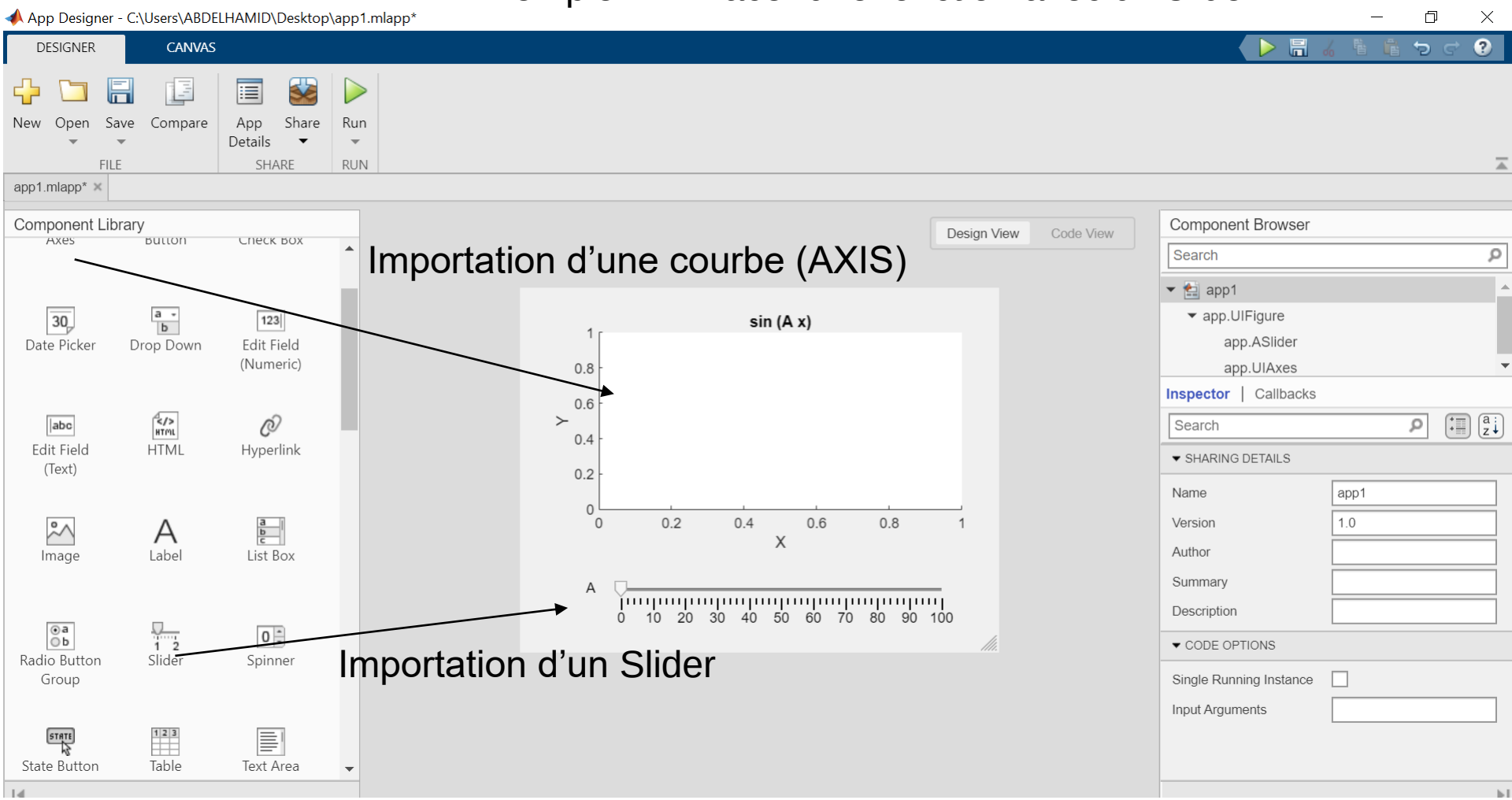
❑ Créer un code avec une GUI

Exemple #1: une simple calculatrice de la masse volumique.



❑ Créer un code avec une GUI

Exemple #2: Tracer une fonction avec un slider.



❑ Créer un code avec une GUI

Exemple #2: Tracer une fonction avec un slider.

App Designer - C:\Users\ABDELHAMID\Desktop\app1.mlapp*

DESIGNER | EDITOR

Save | Compare | Callback | Function | Property | App Input Arguments | Go To | Find | Comment | Indent | None | Left/ Right | Top/ Bottom | Enable app coding alerts | Zoom In | Zoom Out | Reset Zoom | Show Tips | Run

FILE | INSERT | NAVIGATE | EDIT | SPLIT DOCUMENT | VIEW | ZOOM | RESOURCES | RUN

app1.mlapp*

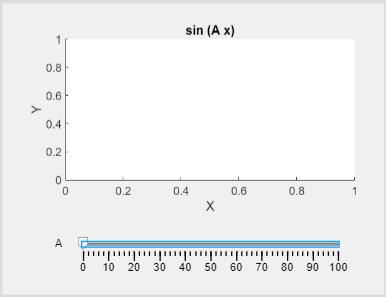
Code Browser

Callbacks | Functions | Properties

Search

ASliderValueChanged

App Layout



Design View | Code View

```
1  classdef app1 < matlab.apps.AppBase
2
3      % Properties that correspond to app components
4      properties (Access = public)
5          UIFigure      matlab.ui.Figure
6          ASlider       matlab.ui.control.Slider
7          ASliderLabel  matlab.ui.control.Label
8          UIAxes        matlab.ui.control.UIAxes
9      end
10
11     % Callbacks that handle component events
12     methods (Access = private)
13
14         % Value changed function: ASlider
15         function ASliderValueChanged(app, event)
16             value = app.ASlider.Value;
17             A= app.ASlider.Value;
18             x = -2*pi : pi/10:2* pi;
19             y = sin (A*x) ;
20             plot (app.UIAxes, x, y,'b-','LineWidth',1.5);
21         end
22     end
23
24     % Component initialization
25     methods (Access = private)
```

code

Component Browser

Search

app1
app.UIFigure
app.ASlider
app.UIAxes

Inspector | Callbacks

Search

SLIDER

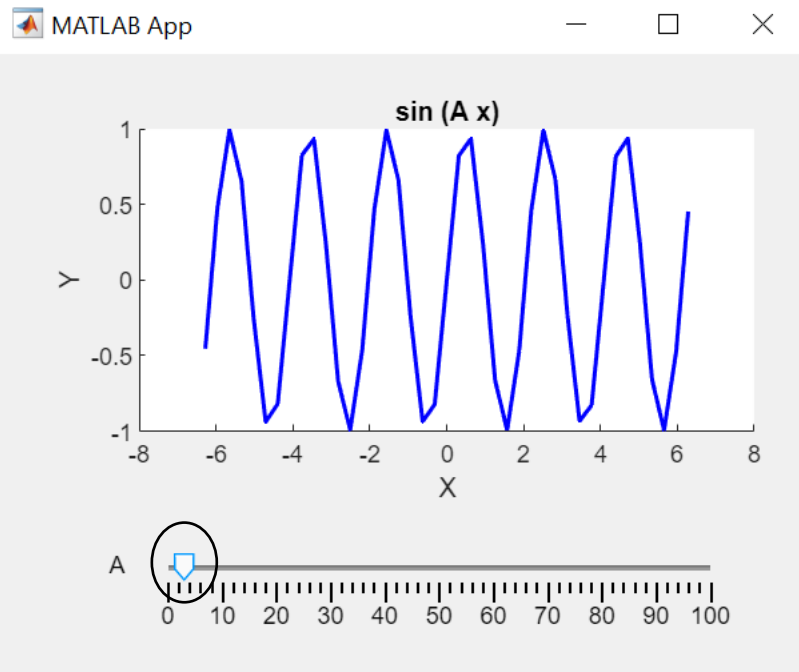
Value	0
Limits	0,100
Orientation	

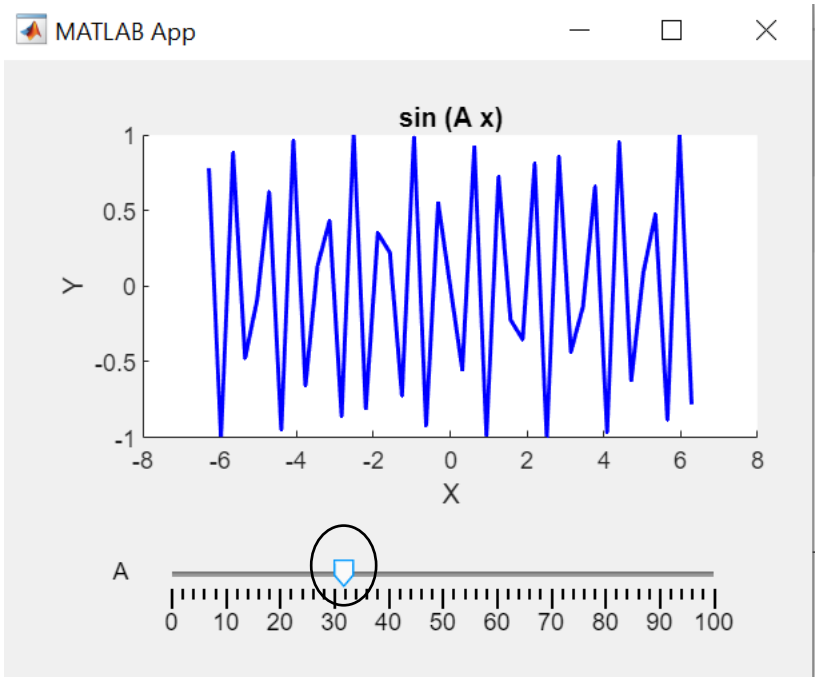
TICKS

MajorTicks	0,10,20,30,40,50,60,7(
MajorTickLabels	0,10,20,30,40,50,60,7(
MinorTicks	0,2,4,6,8,10,12,14,16,
MajorTicksMode	auto
MajorTickLabelsMode	auto
MinorTicksMode	auto

❑ Créer un code avec une GUI

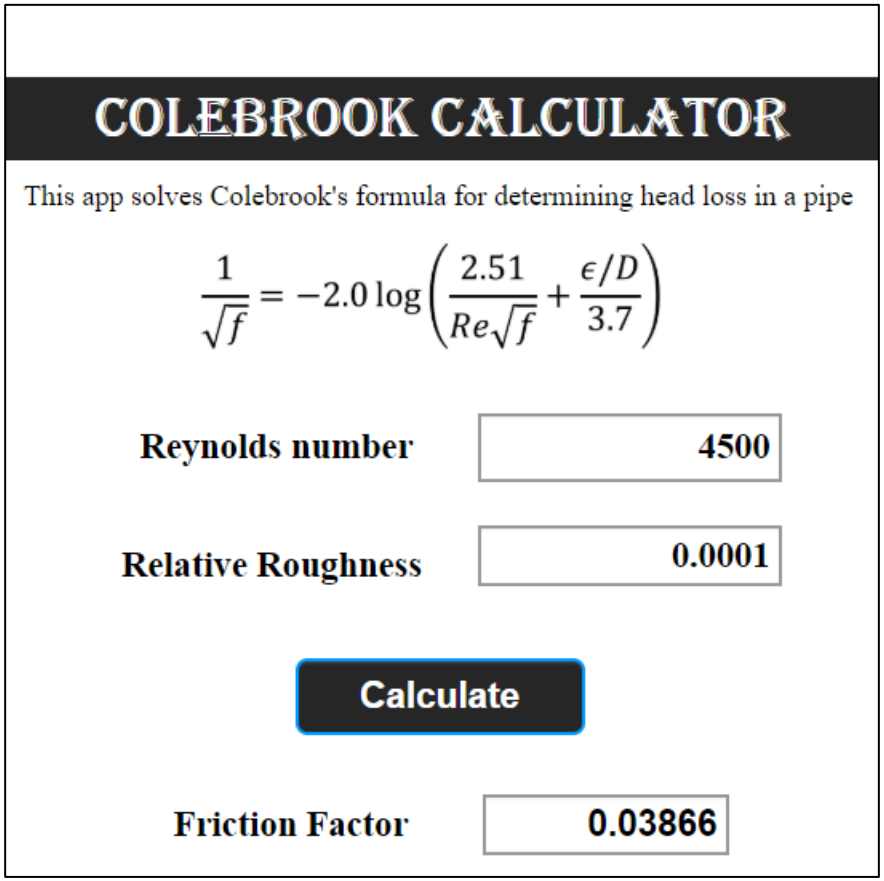
Exemple #2: Tracer une fonction avec un slider.





❑ Créer un code avec une GUI

Exercice: Construire un code avec une GUI qui permet de calculer le coefficient de Darcy (pertes de charges linéaires – écoulements turbulents en conduites).



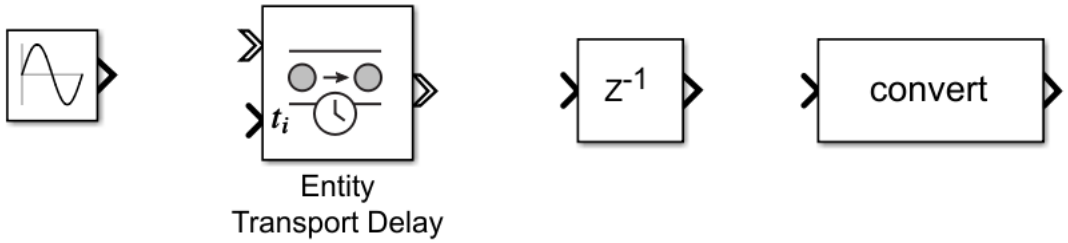
Introduction au Logiciel Simulink

❑ Introduction

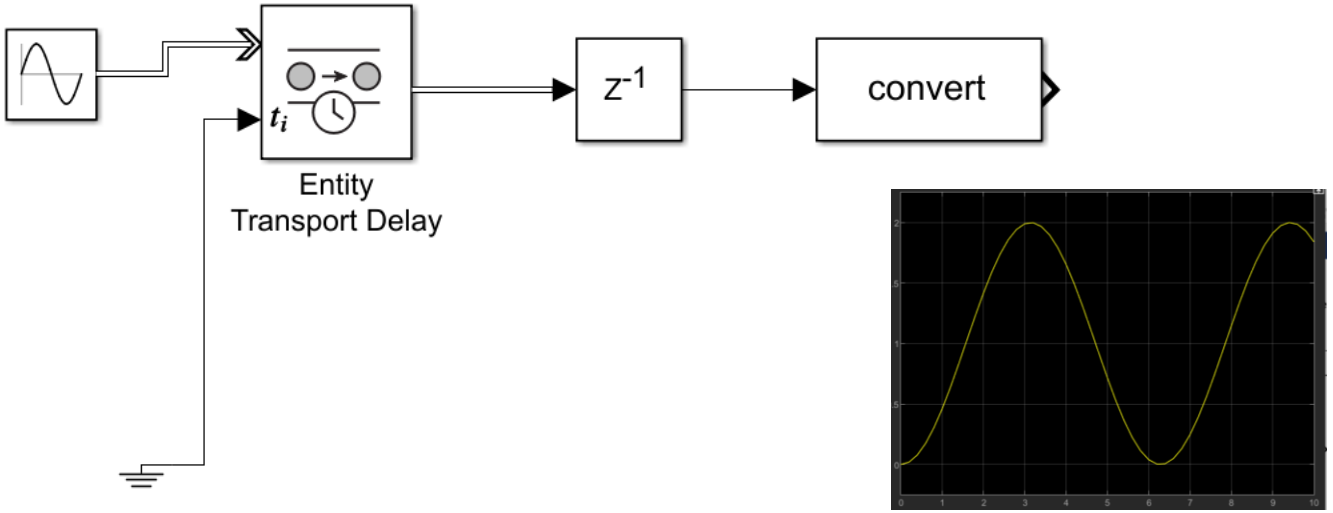
- SIMULINK est un logiciel muni d'une interface graphique, intégré à l'environnement MATLAB, permettant de représenter les fonctions mathématiques et les systèmes dynamiques se forme de schémas en blocs.
- SIMULINK permet de modéliser, simuler (en temps continus ou discrets) et analyser les systèmes dynamiques linéaires et non-linéaires.
- SIMULINK est largement utilisé dans le monde, dans différents domaines tels que:
  - ✓ Energies renouvelables
  - ✓ Aérospatial
  - ✓ Automatique
  - ✓ Communications
  - ✓ Electronique et Traitement du signal
  - ✓ Systèmes de contrôle
  - ✓ Instrumentation médicale…

❑ Introduction

Toutes représentations sous Simulink se fait au moyen de ***blocs***, caractérisés par leur fonction et leurs entrées/sorties:

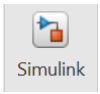


Les blocs sont reliés entres eux par des ***signaux*** temporels:

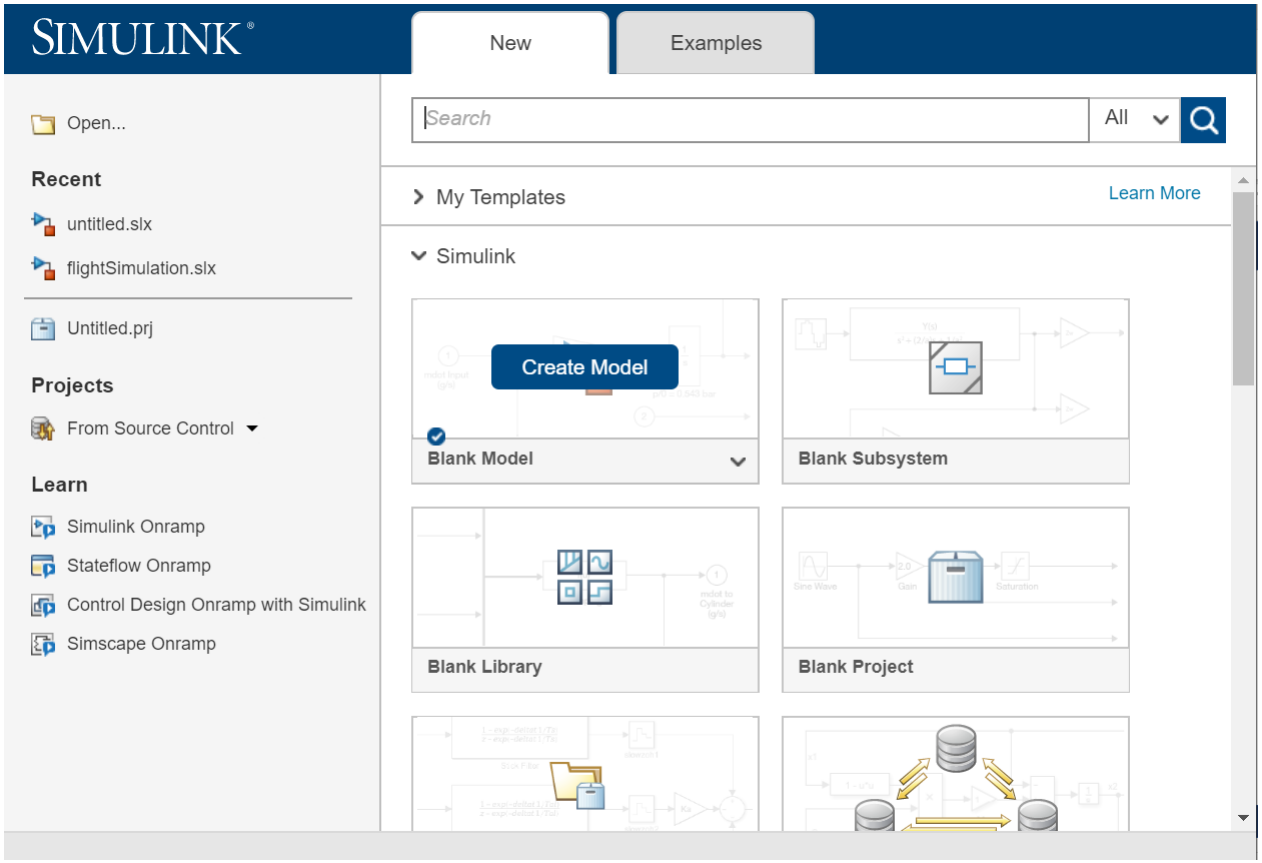


❑ Création d'un modèle Simulink

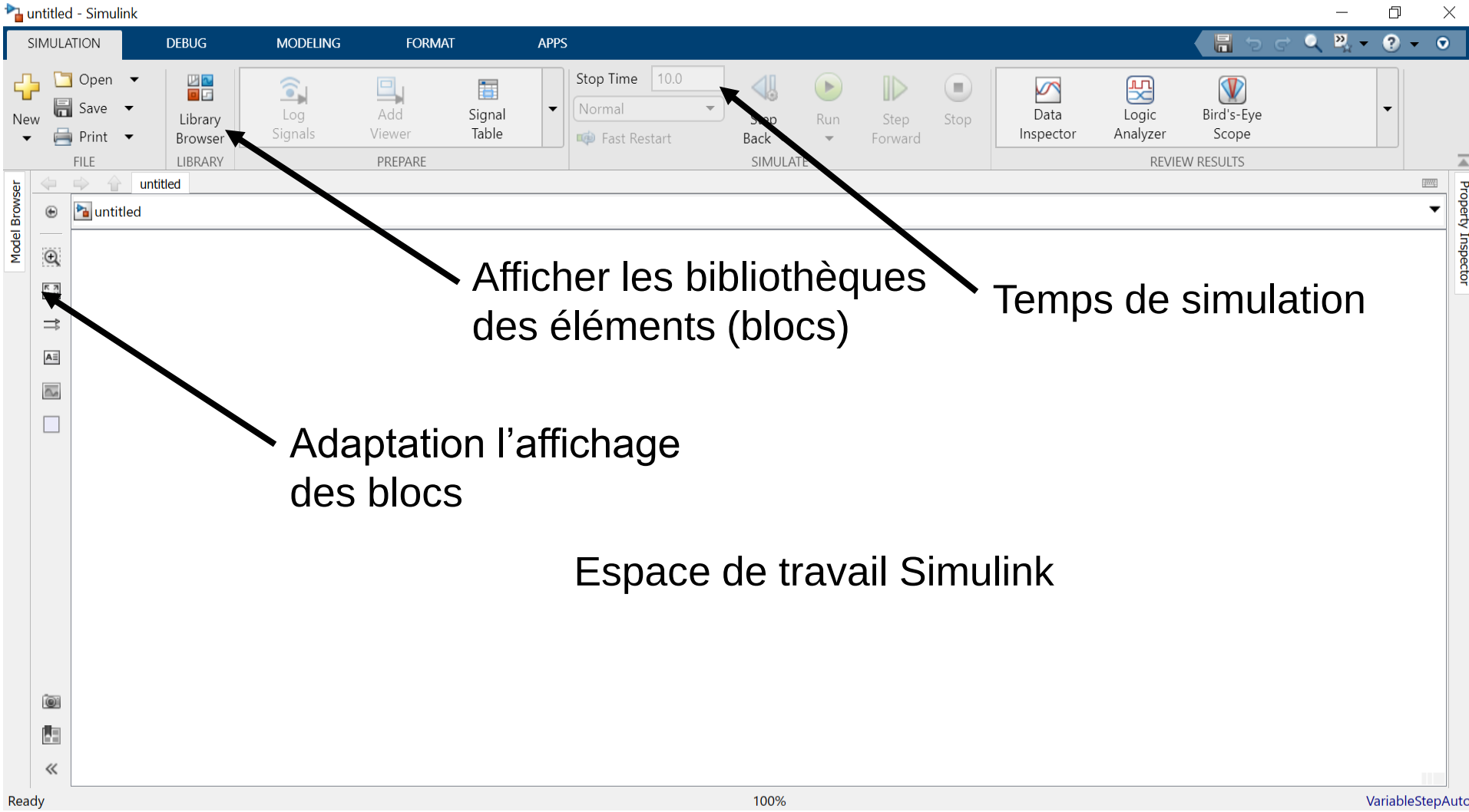
Simulink peut être lancé depuis l'environnement de MATLAB

- en cliquant, dans la barre d'outils, sur le bouton 

- ou en tapant *simulink* dans le *Command Windows.*



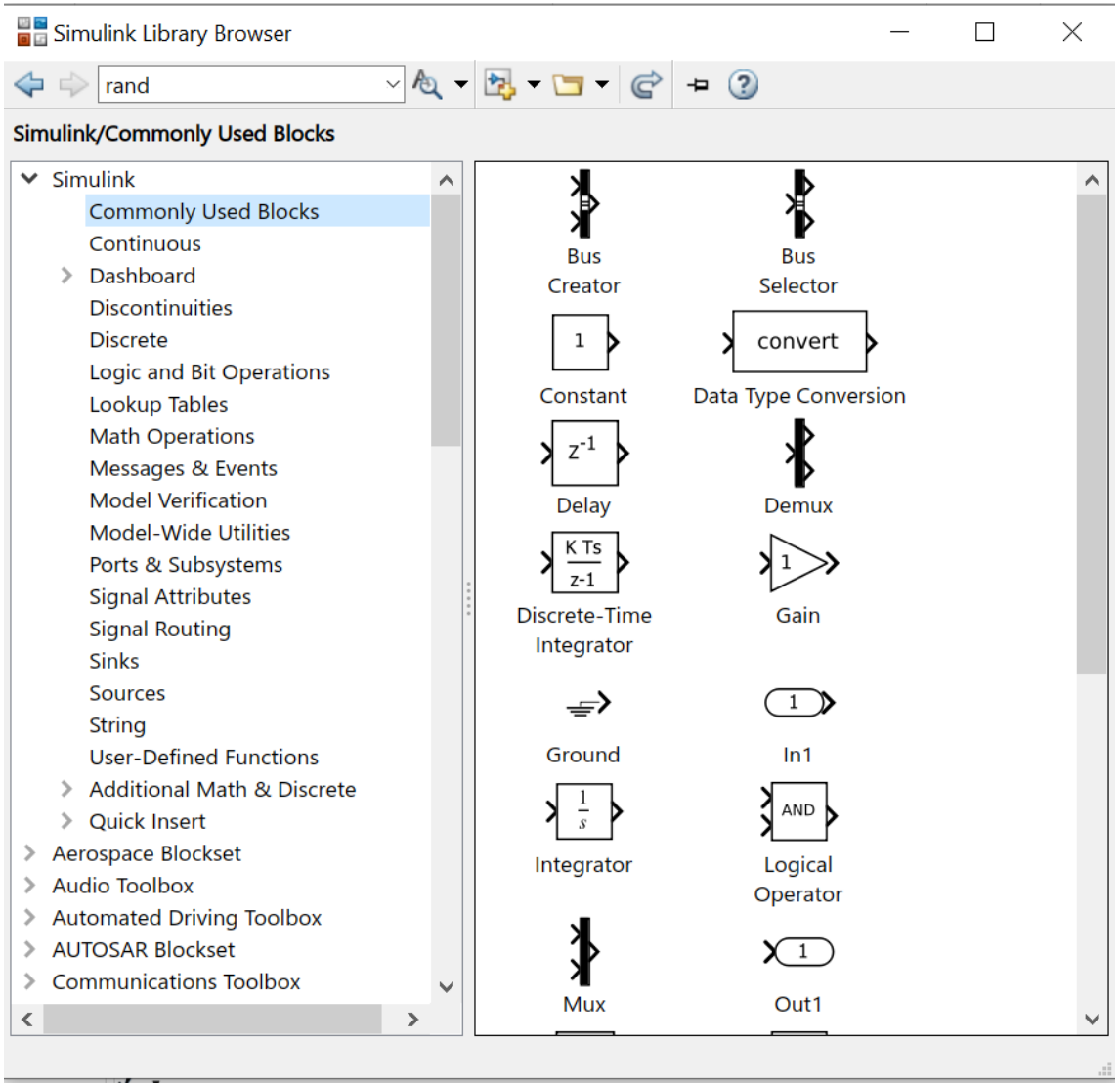
❑ Création d'un modèle Simulink



Bibliothèques et Blocs SIMULINK

- Il existe plusieurs bibliothèques dans le Simulink

- La fenêtre à gauche liste les bibliothèques des blocs disponibles.
- À la sélection d'une bibliothèque, les blocs qui la composent sont affichés dans la partie de droite.

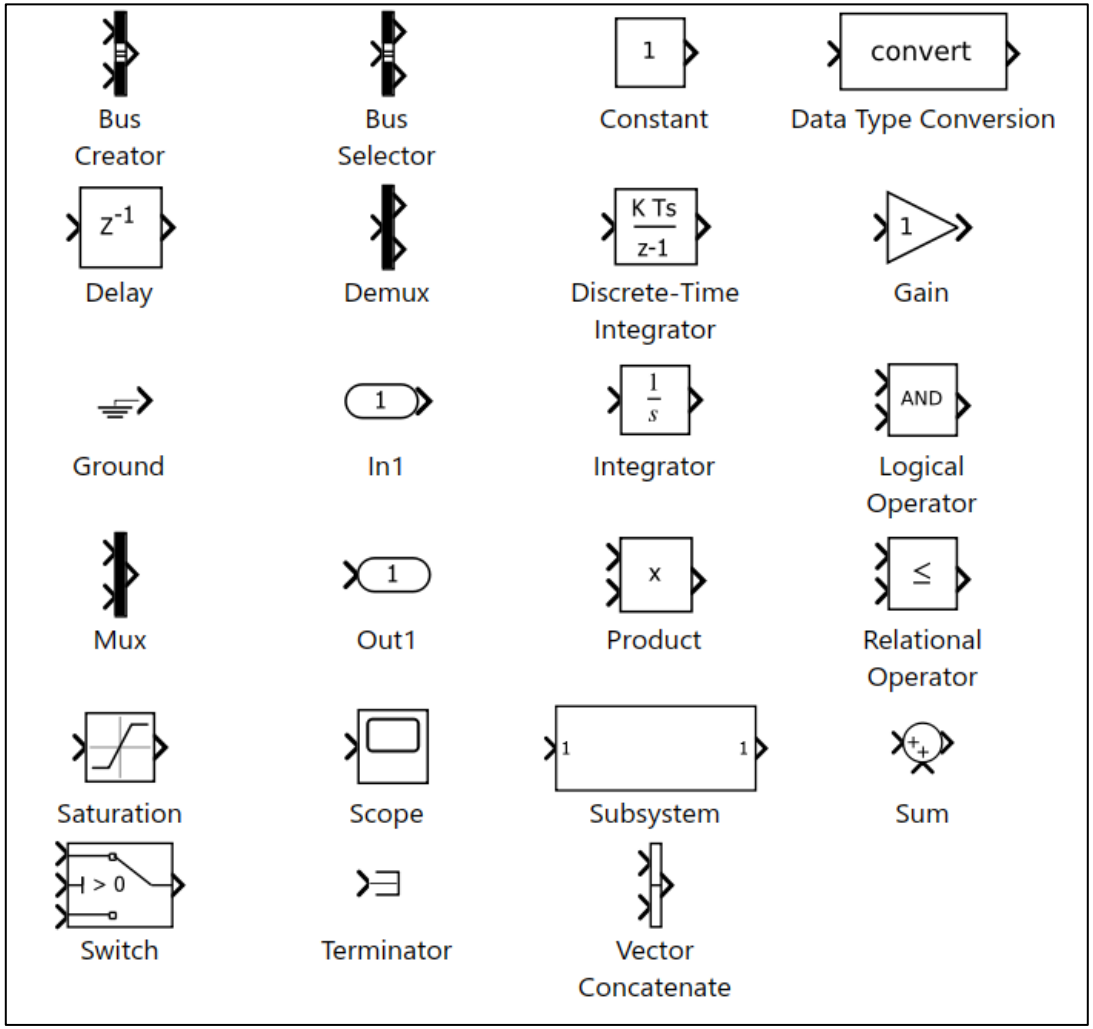


❑ *Bibliothèque «Commonly Used Blocks»:*

«Commonly Used Blocks»:

Contient les blocs les plus fréquents:

- Constant;
- Intégrateur;
- Sommateur
- Multiplicateur;
- Switch …



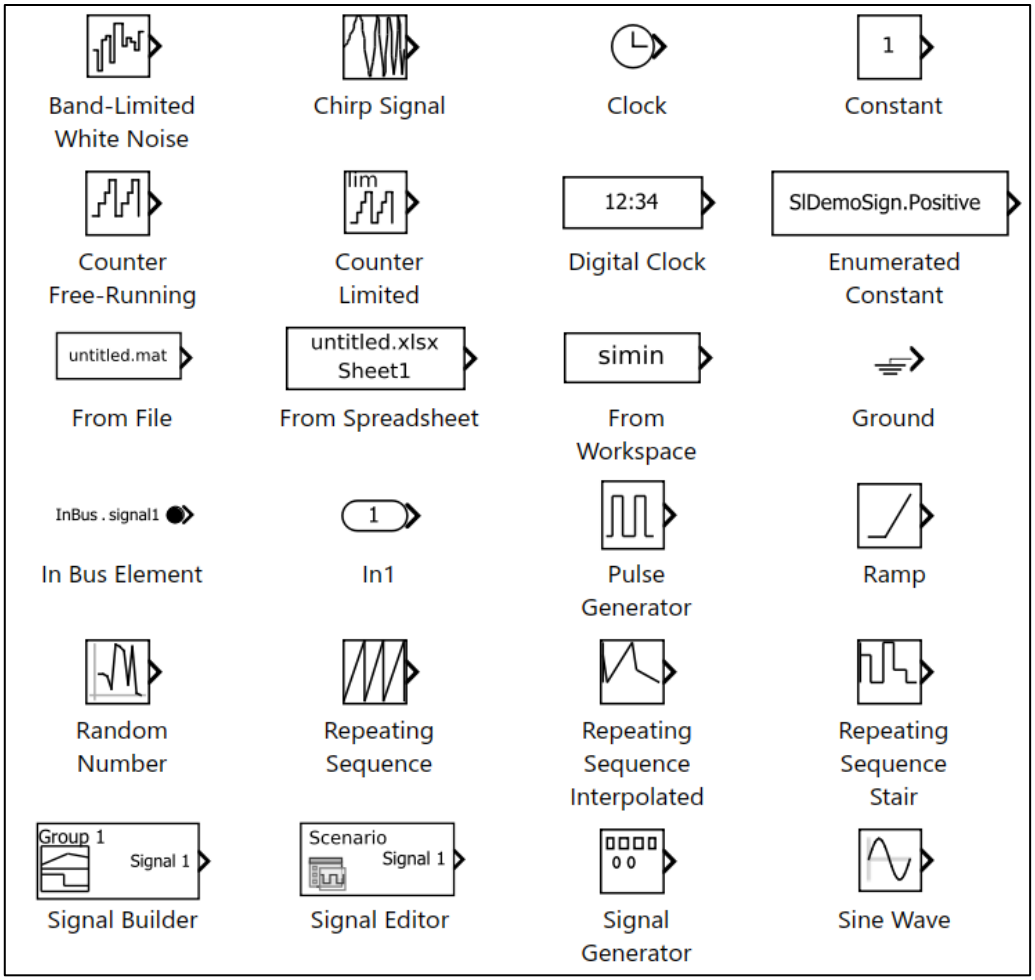
❑ *Bibliothèque «Sources»:*

«Sources»:

Contient les blocs sources:

- Onde sinusoïdale;
- Nombre aléatoire;
- Lecteur des données à partir Matlab …

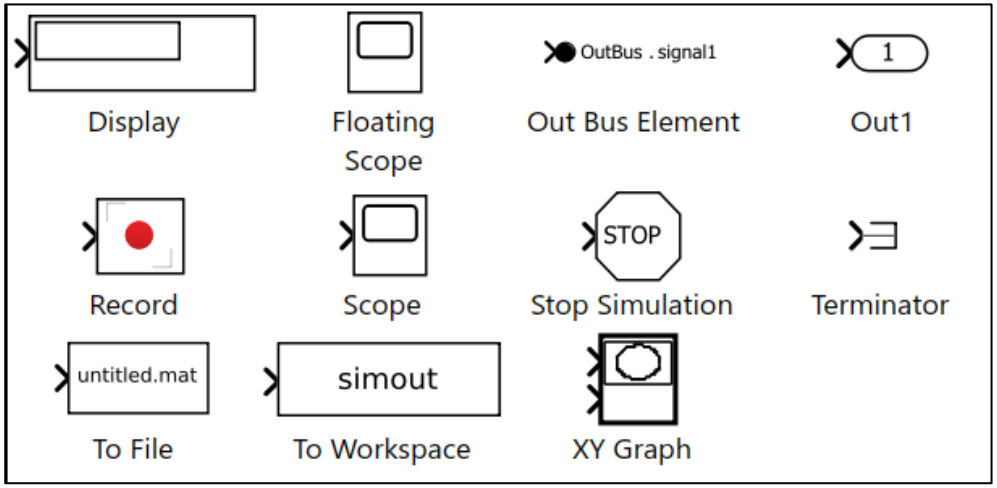
Les blocs sources sont des blocs possédant une ou plusieurs sorties et aucune entrée. Il sont utilisés pour la génération de signaux.



❑ Bibliothèque «Sinks»:

«Sinks»:

- Afficheur (Display);
- Enregistreur (Record);
- Oscilloscope (Scope);
- X Y Graph …



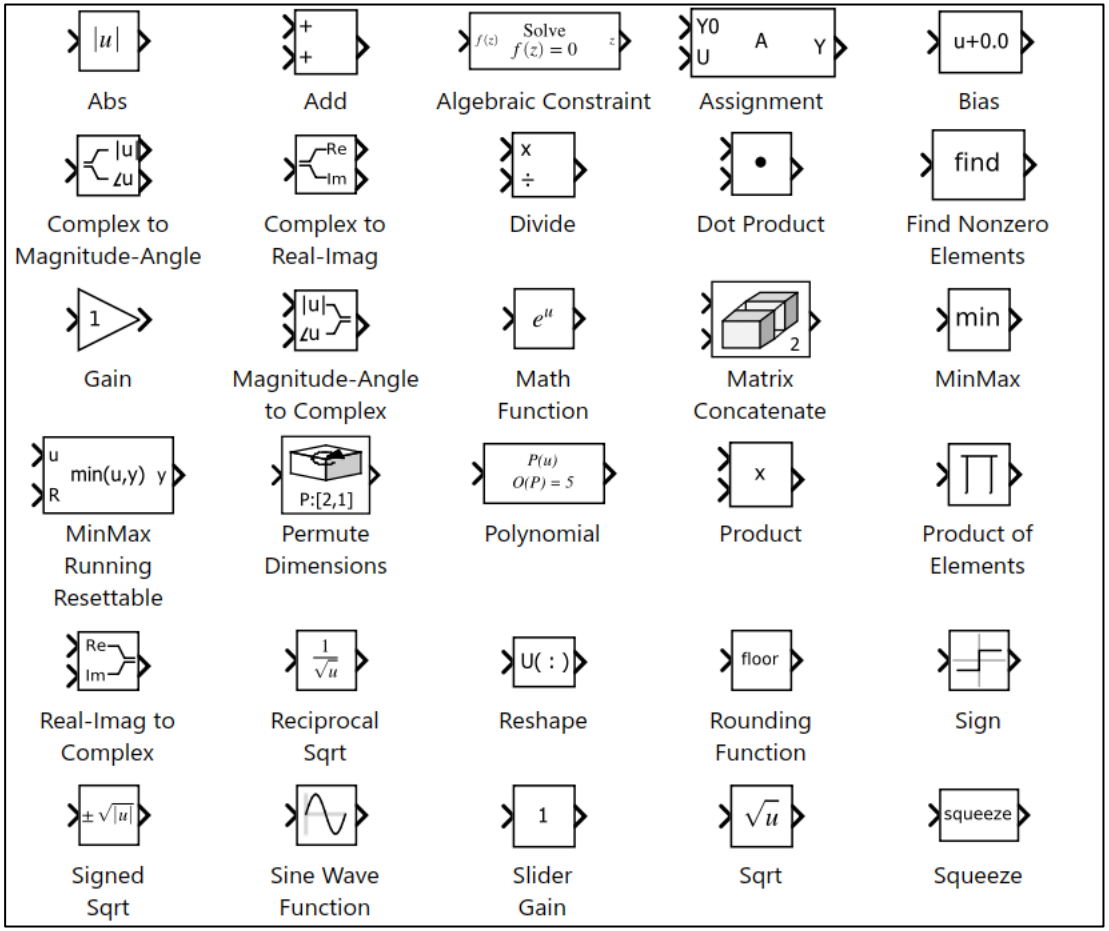
Au contraire des blocs *sources*. Les blocs *sinks* sont des blocs possédant une ou plusieurs entrées et aucune sortie. Il sont utilisés pour afficher les résultats de la simulation.

❑ Bibliothèque «Math Operations»:

«Math Operations»:

- Produit (Product);
- Diviser (Divide);
- Ajouter (Add);
- Valeur absolue (Abs);
- Multiplicateur (Gain) …

Les blocs *Math Operations* sont des blocs réalisant des opérations mathématiques pour transformer le signal entrant.

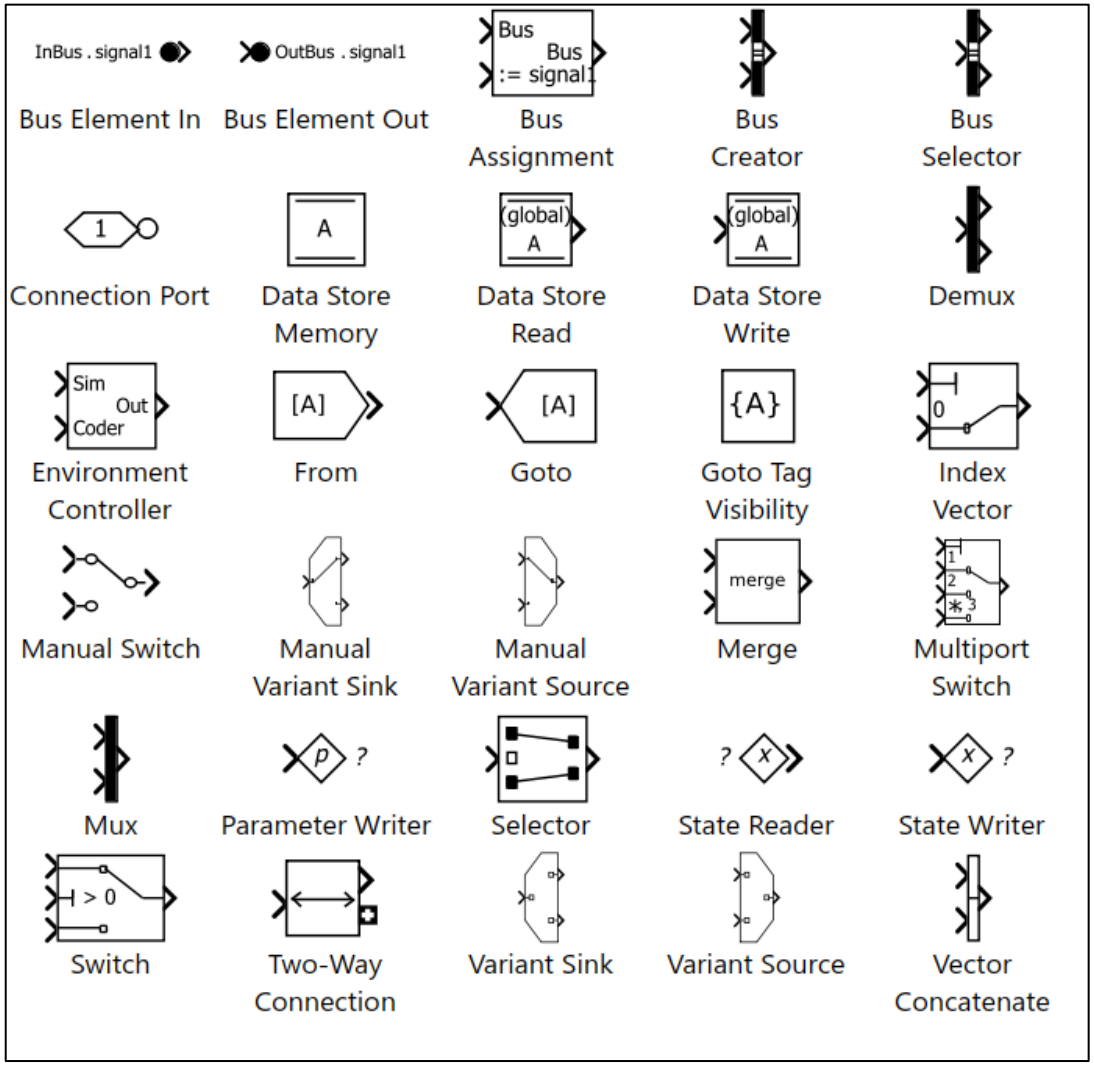


❑ Bibliothèque «Signal Routing»:

«Signal Routing»:

- Mux;
- Switch;
- Demux …

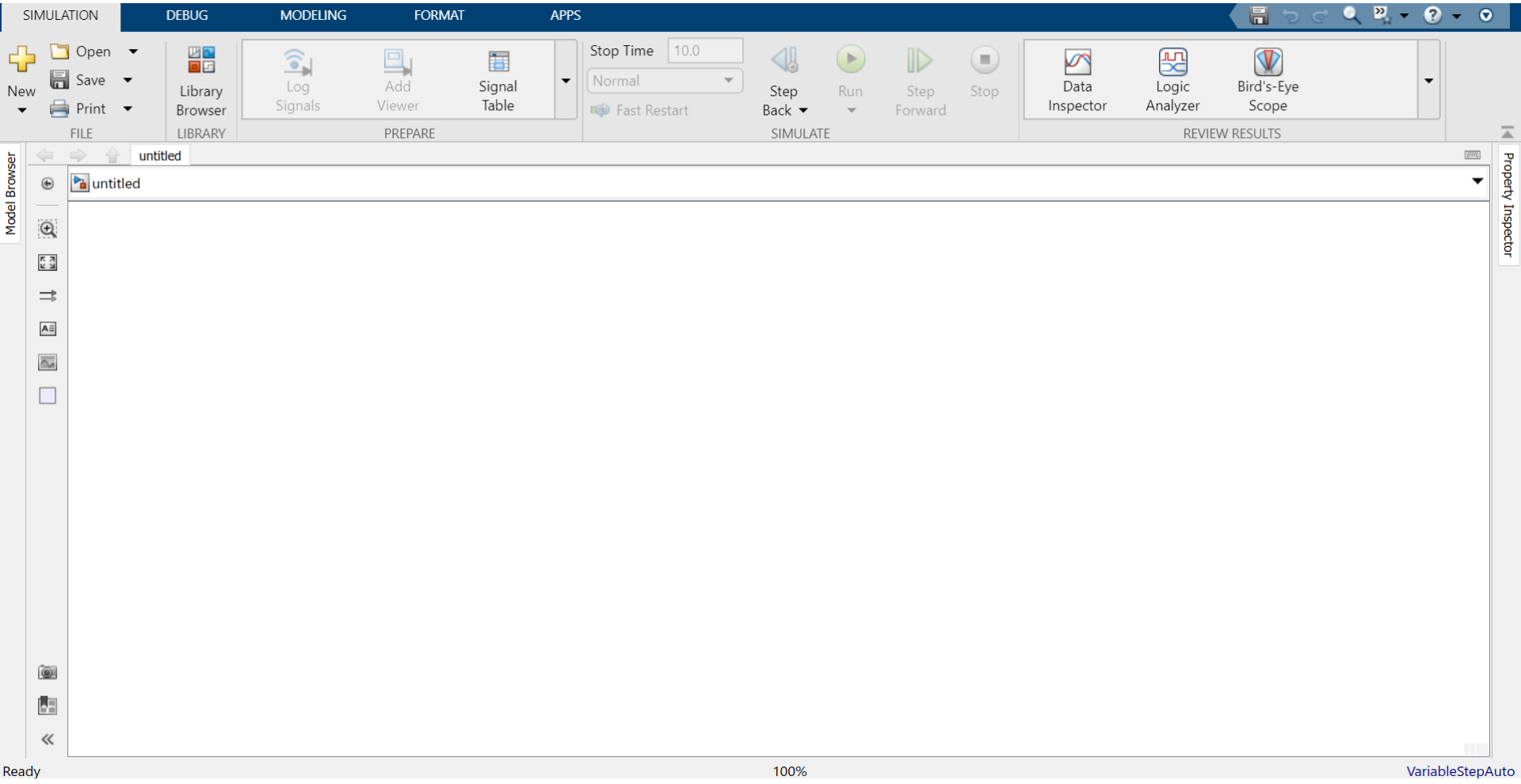
Les blocs *Signal Routing* sont des blocs utilisés pour l' aiguillage de signaux ou la connexion des blocs.



Principe de Fonctionnement de SIMULINK

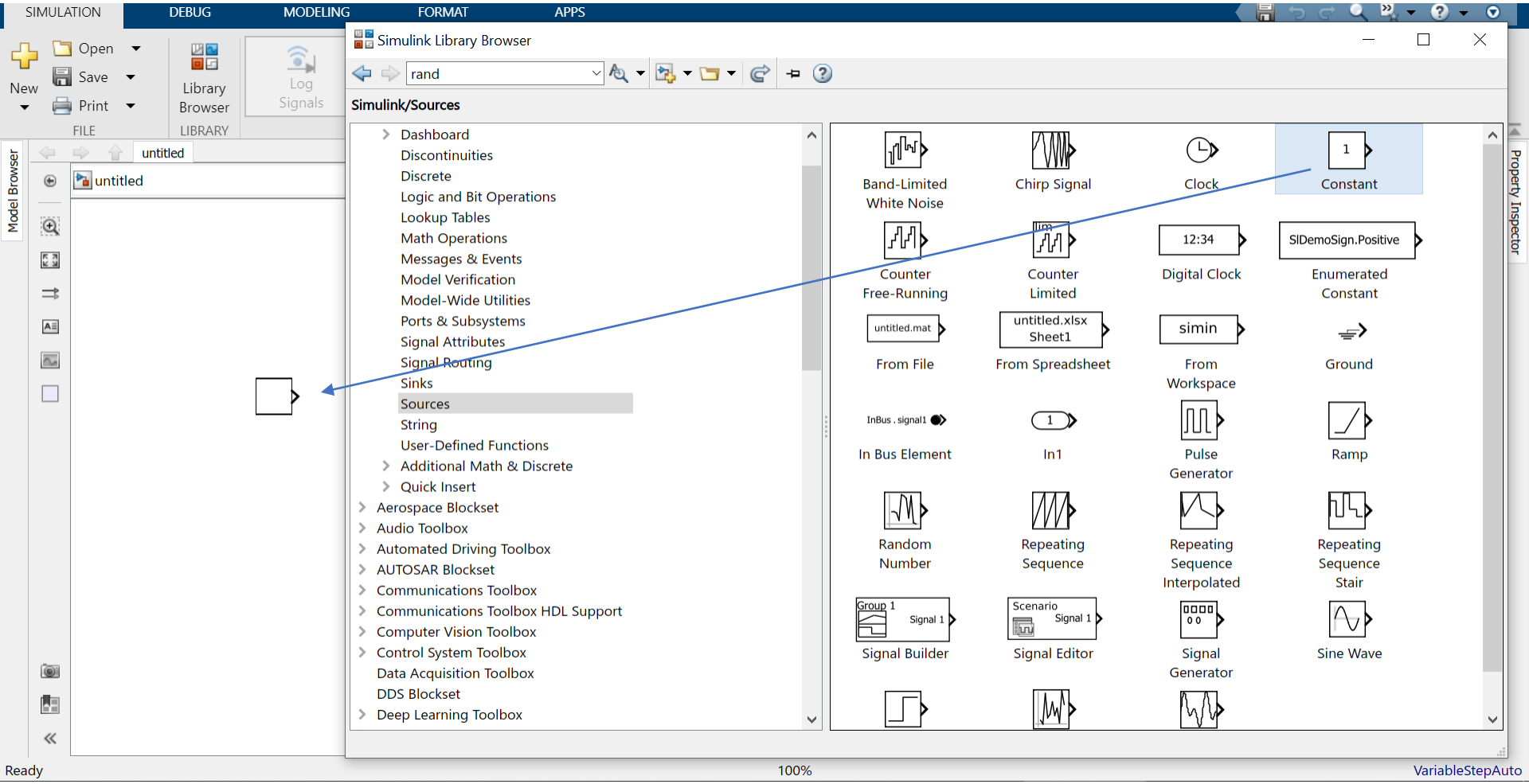
- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

Étape 1 : Création d'un nouveau modèle Simulink



- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

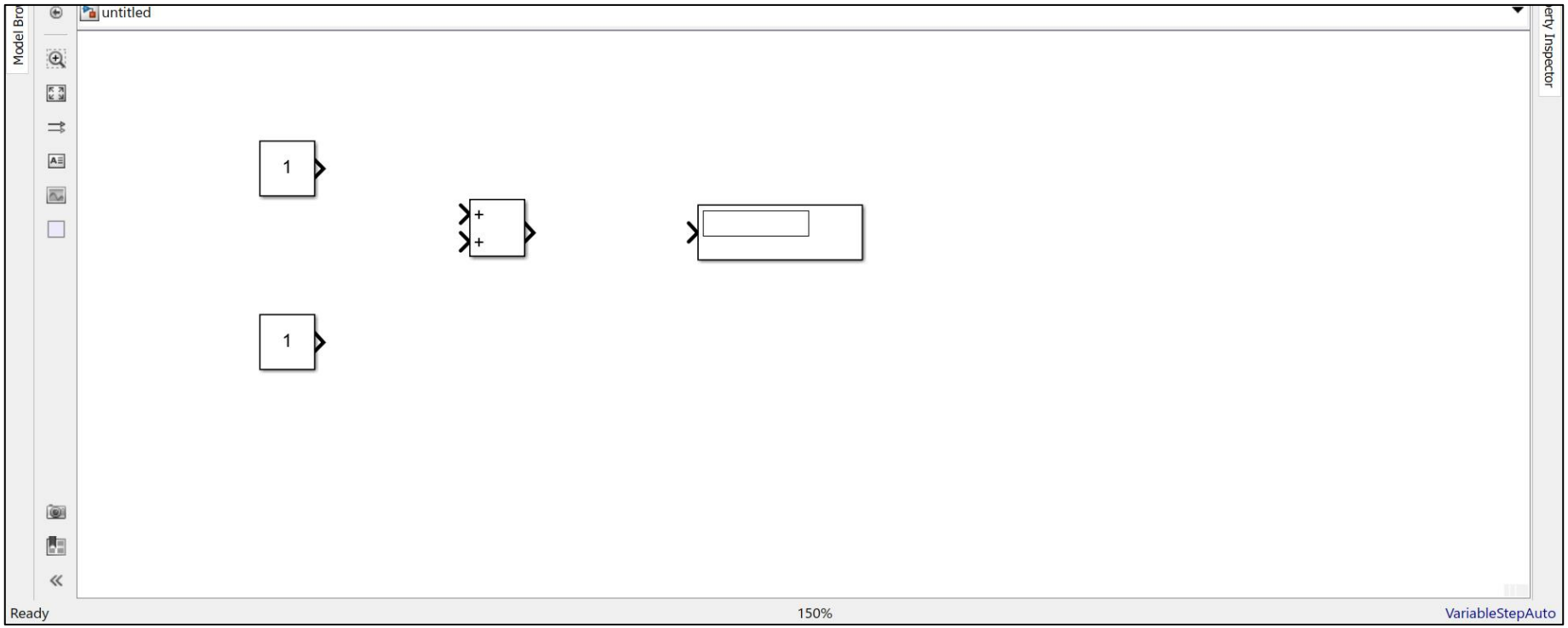
Étape 2 : Ajouter des blocs par « glisser / déposer » (drag and drop):



- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

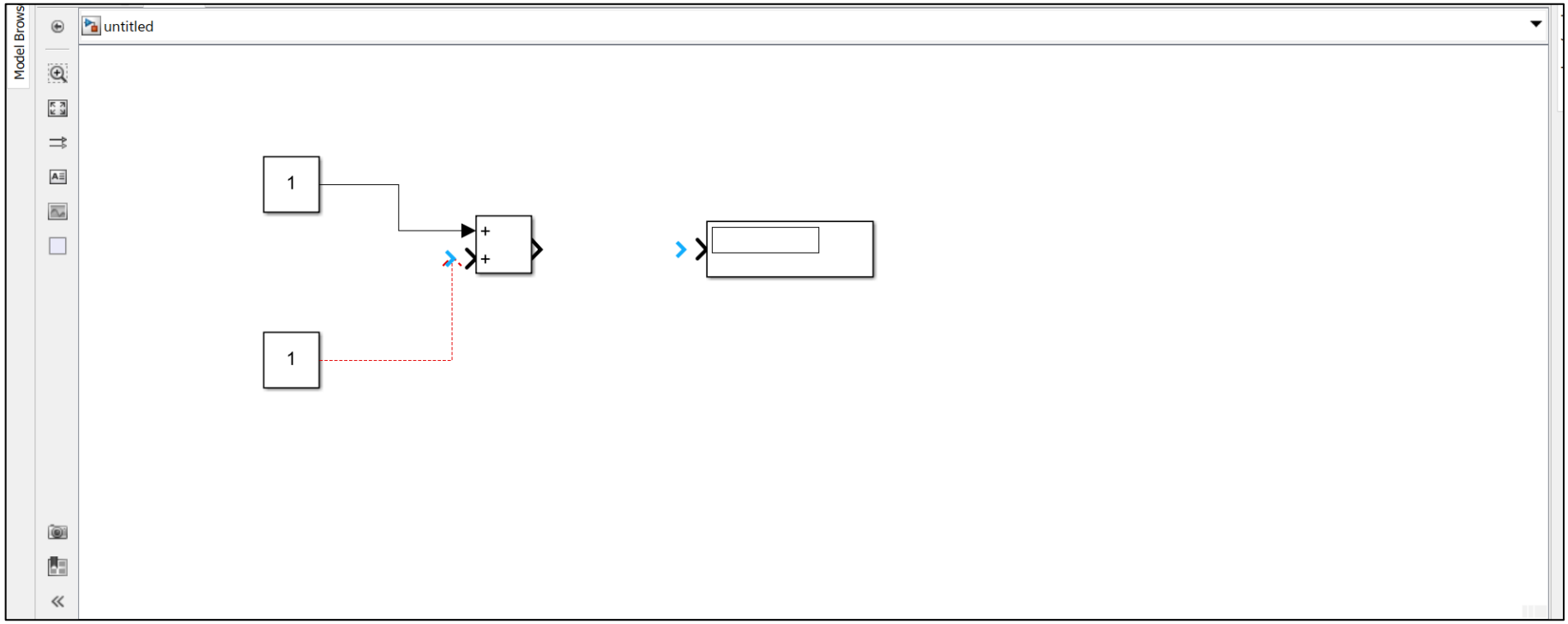
Étape 2 : Ajouter des blocs par « glisser / déposer » (drag and drop):

- 2 blocs de type *Constant* (Bib. *Sources*)
- 1 bloc de type *Add* (bib. *Math Operations*)
- 1 bloc de type Display (bib. *Sinks*)

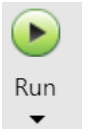


- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

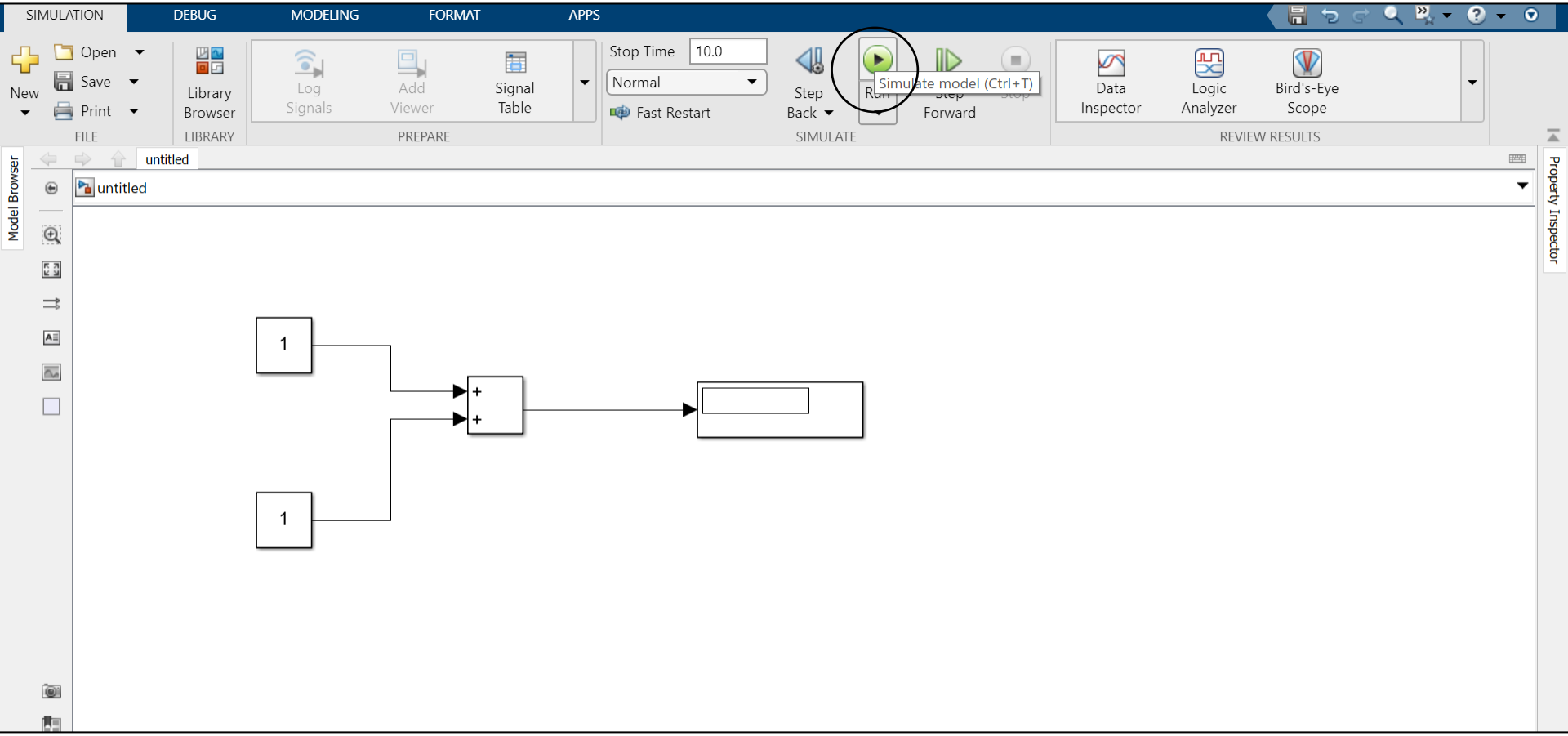
Etape 3 : Connecter les blocs en utilisant la souris



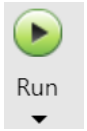
- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

Etape 4 : Lancer la simulation en appuyant sur l'icône 

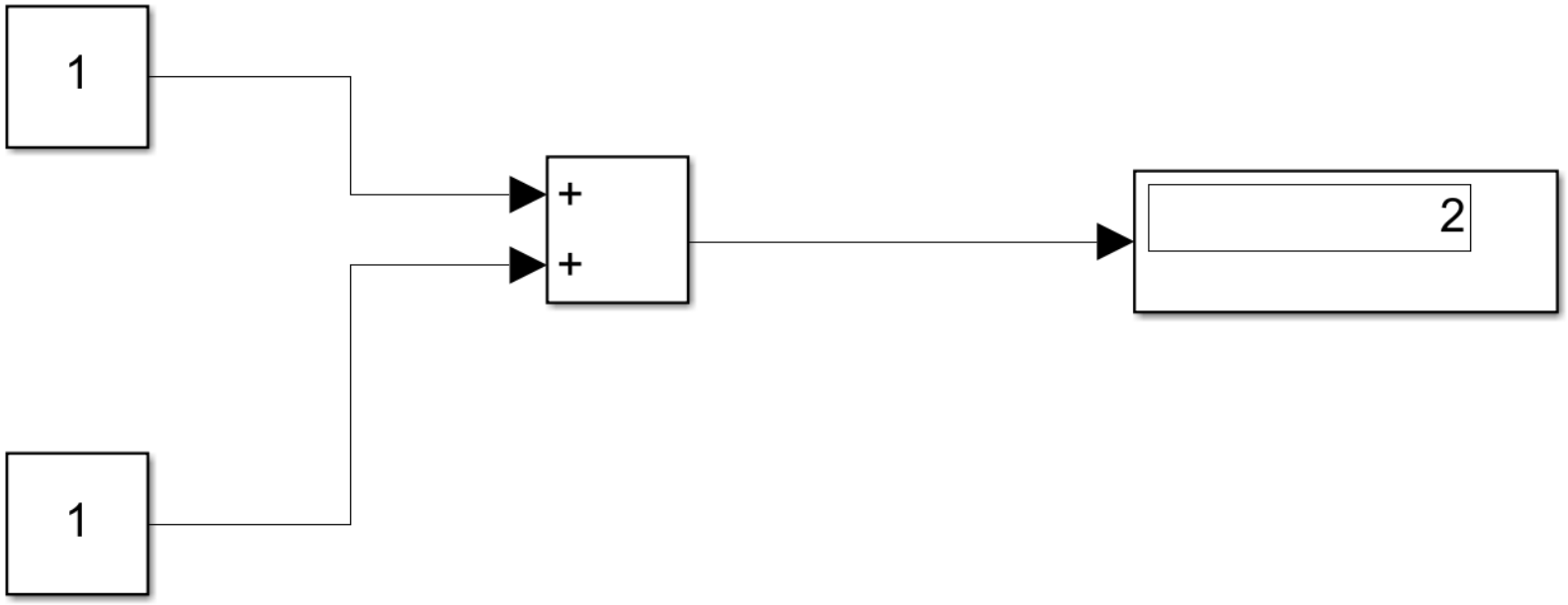
ou en tapant *sim (nom_model)* dans la *Command Windows*



- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

Etape 4 : Lancer la simulation en appuyant sur l'icône 

ou en tapant *sim (nom_model)* dans la *Command Windows:*



❑ Exercice de prise en main de Simulink: La somme de deux nombres

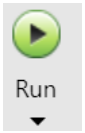
Résumé des étapes:

Étape 1 : Création d'un nouveau modèle Simulink

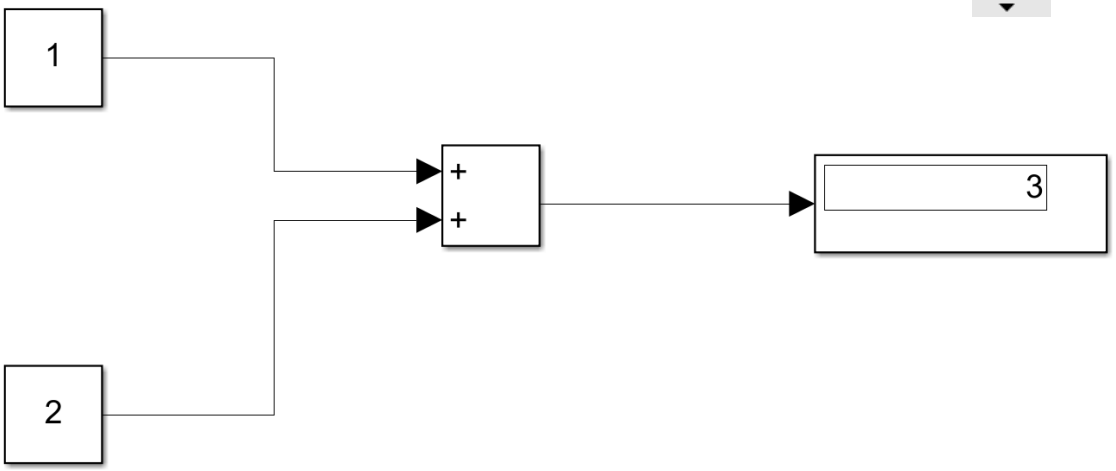
Étape 2 : Ajouter des blocs par « glisser / déposer » (drag and drop):

- 2 blocs de type *Constant* (Bib. *Sources*)
- 1 bloc de type *Add* (bib. *Math Operations*)
- 1 bloc de type Display (bib. *Sinks*)

Etape 3 : Connecter les blocs en utilisant la souris

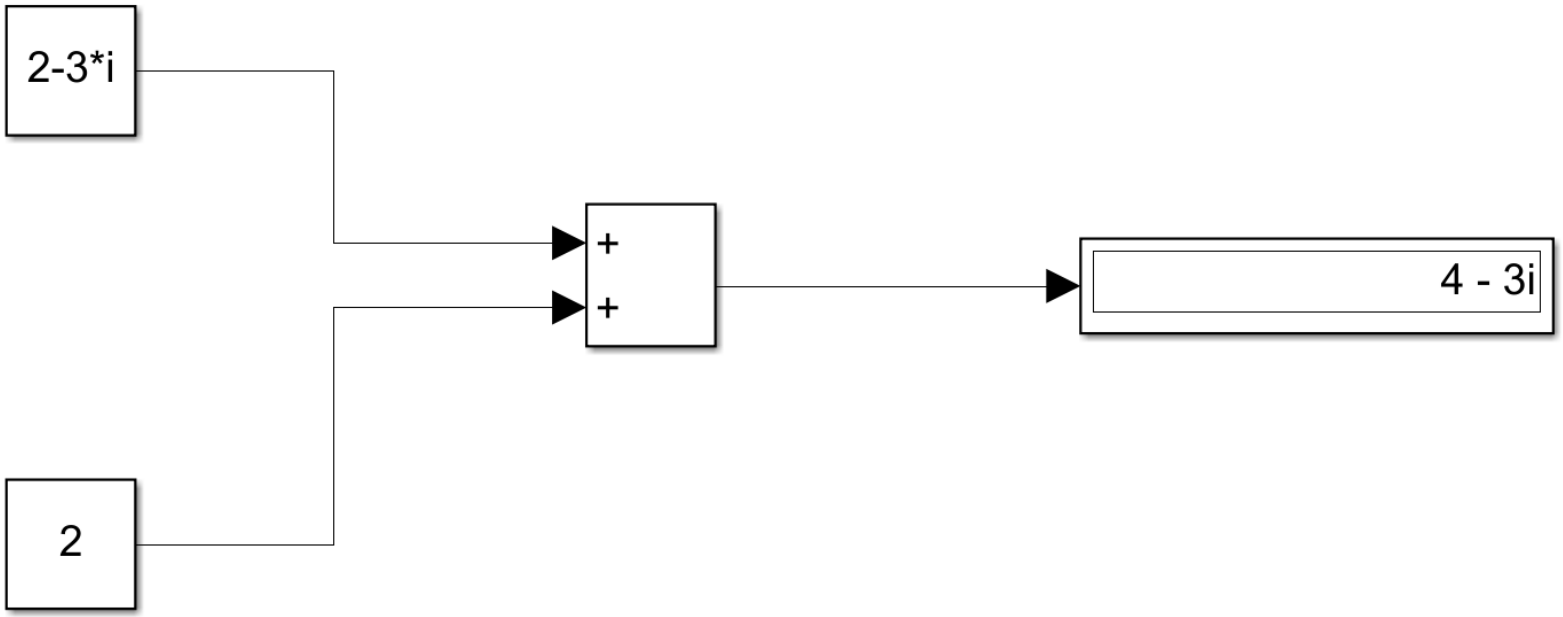
Etape 4 : Lancer la simulation en appuyant sur l'icône 

ou en tapant *sim*



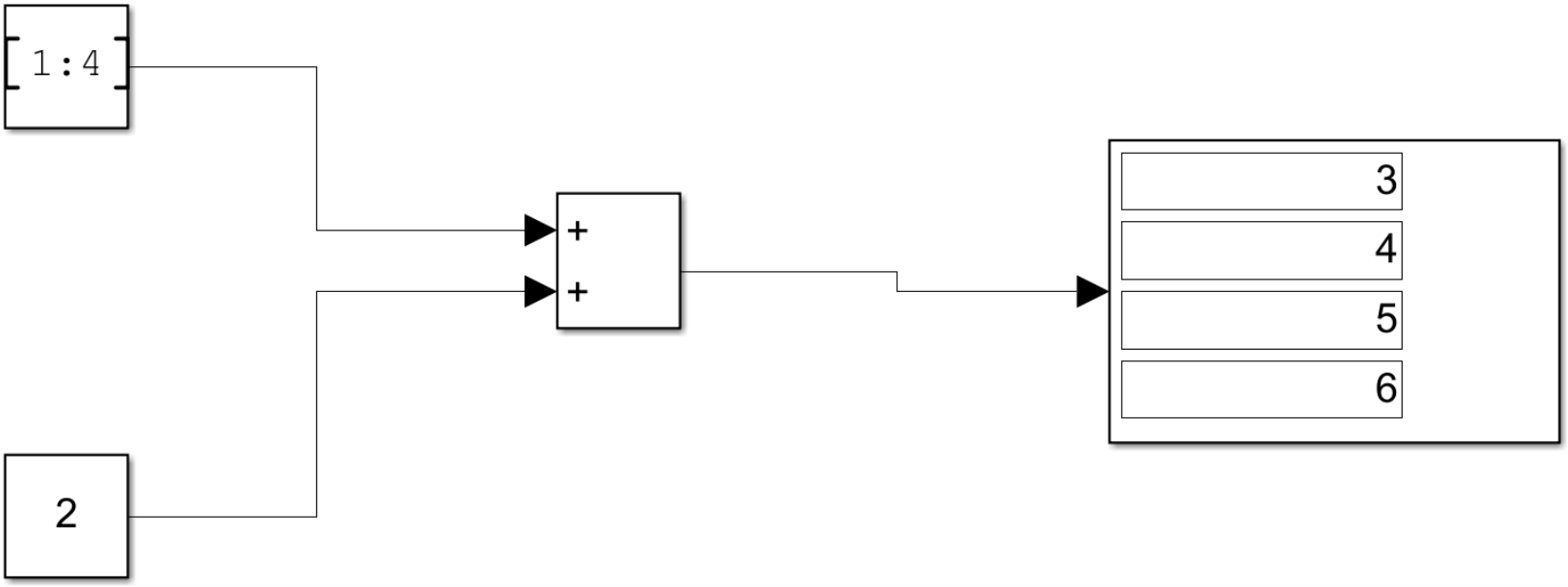
- **Exercice de prise en main de Simulink:**
  La somme de deux nombres

Le Bloc *Constant* peut être utilisé également pour représenter **les nombres complexes**.



- **Exercice de prise en main de Simulink :**
  La somme de deux nombres

Remarque: Le Bloc *Constant* peut être utilisé également pour représenter **les vecteurs**.



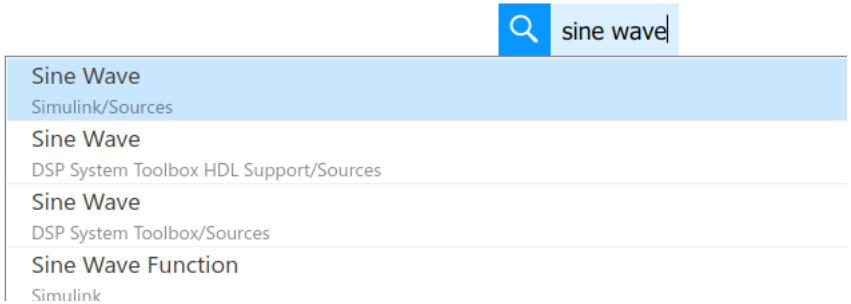
Exemples d'Applications SIMULINK

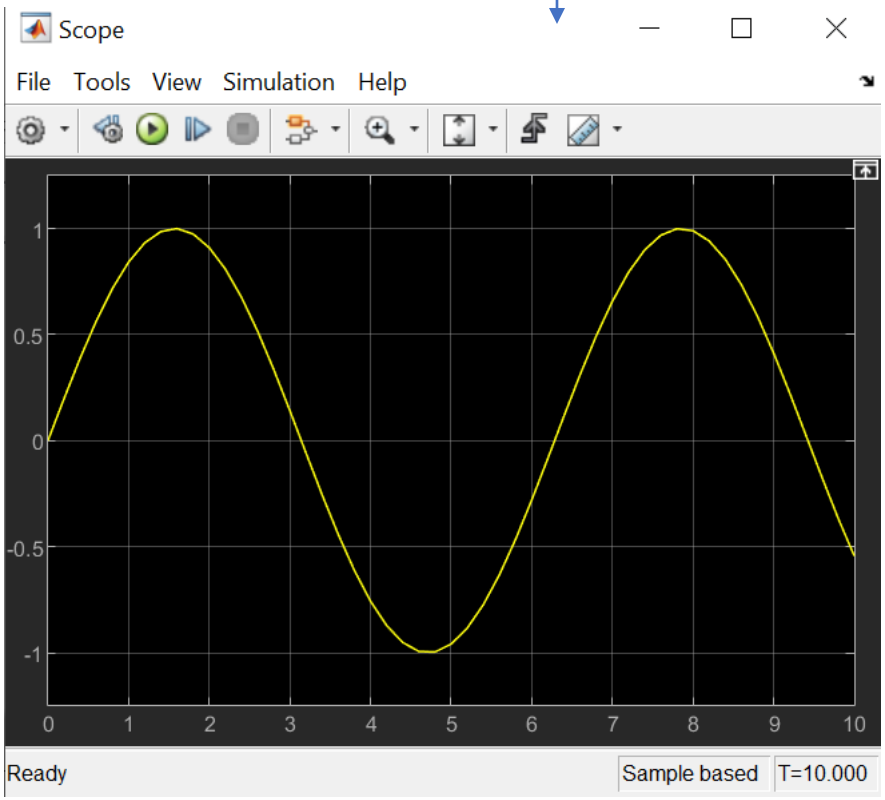
❑ Exemple #1:

- Affichage d'un signal sinusoïdal

- 1 bloc *Sine Wave (bib. Sources)*
- 1 bloc *Scope (bib. Sinks)*

- Remarque: On peut chercher les blocs d'une façon plus rapide en cliquant deux fois dans l'espace de travail Simulink et en écrivant le nom du bloc:





❑ Exemple #1:

- Paramétrage d'un *sine wave*

Output a sine wave:

$$O(t) = Amp*Sin(Freq*t+Phase) + Bias$$

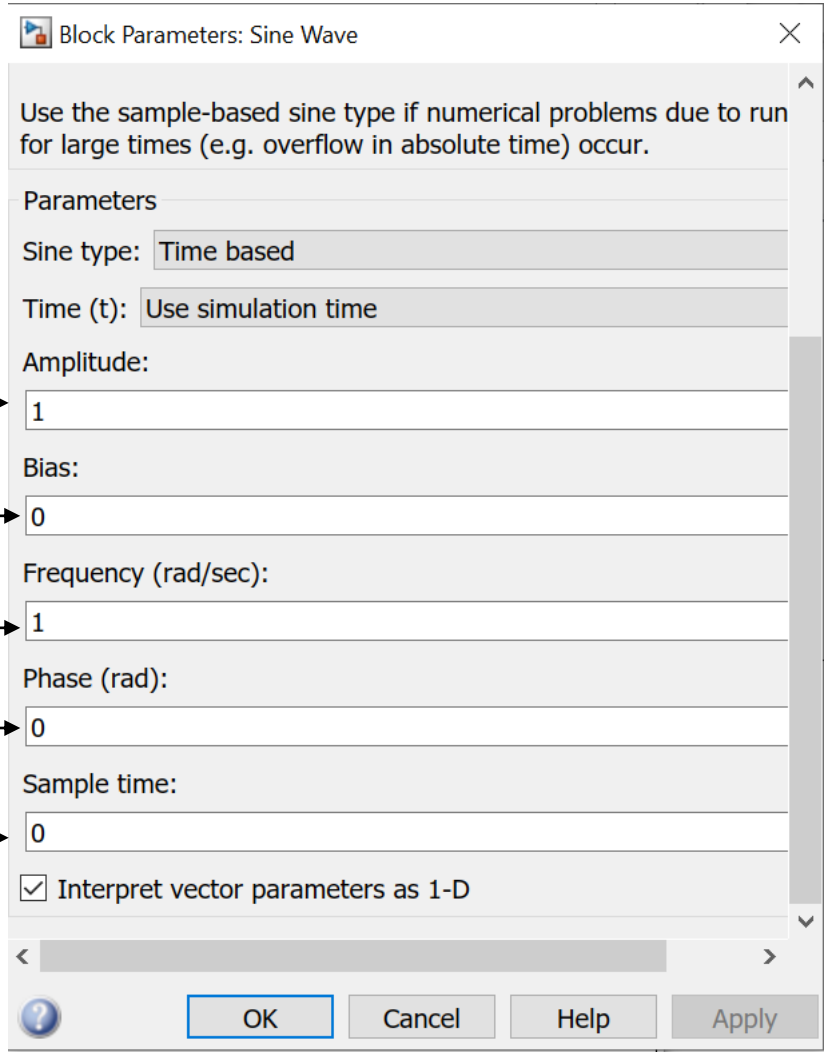
Amplitude →

Composante continue →

Fréquence →

Phase →

Période d'échantillonnage →



❑ Exemple #1:

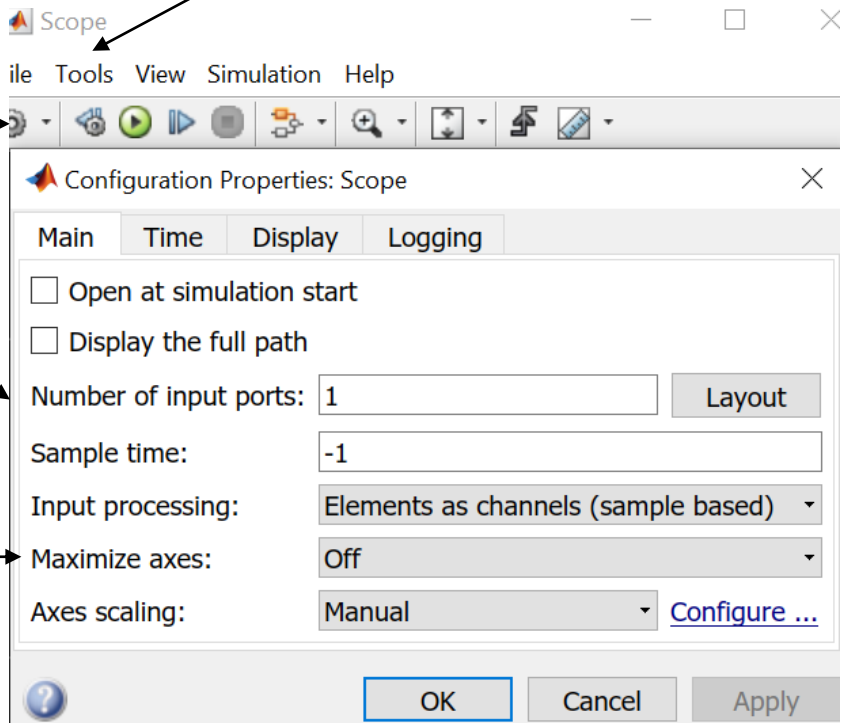
- Paramétrage d'un *scope*

Mise en forme des figures

Paramètres

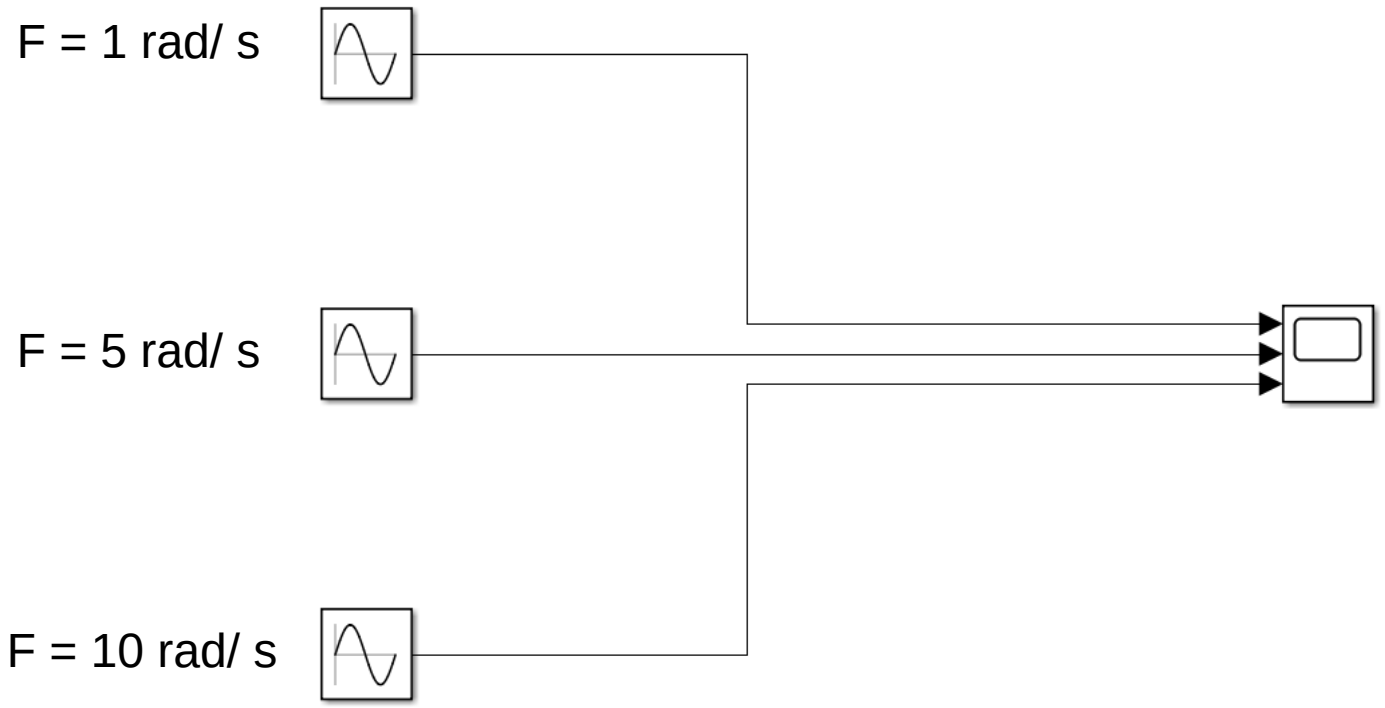
Augmenter les ports d'entrées

Mise à l'échelle des axes



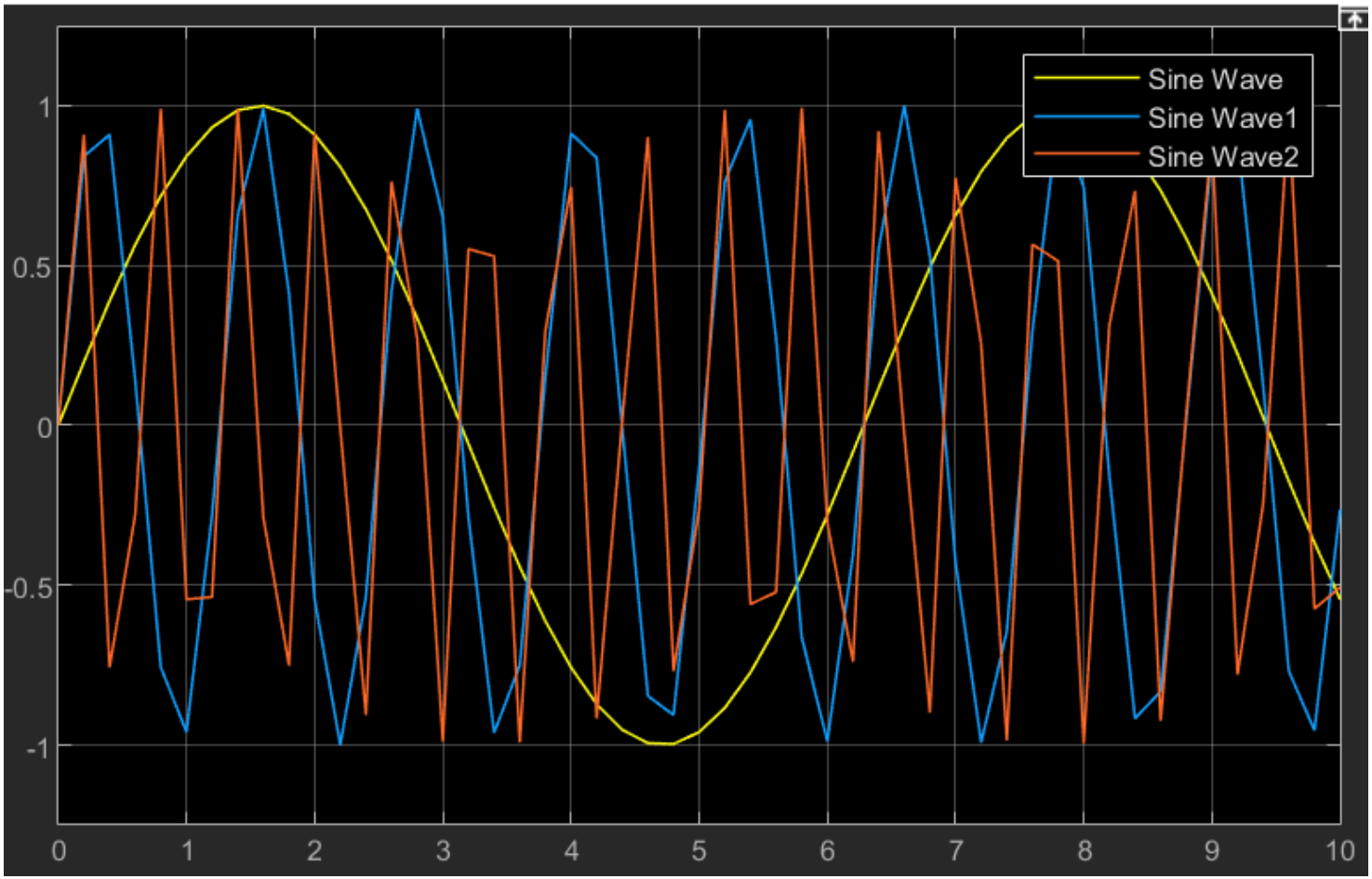
❑ Exemple #2:

- Affichage de plusieurs signaux sinusoïdaux
  - 3 bloc *Sine Wave* (bib. *Sources*)
  - 1 bloc *Scope* (bib. *Sinks*)



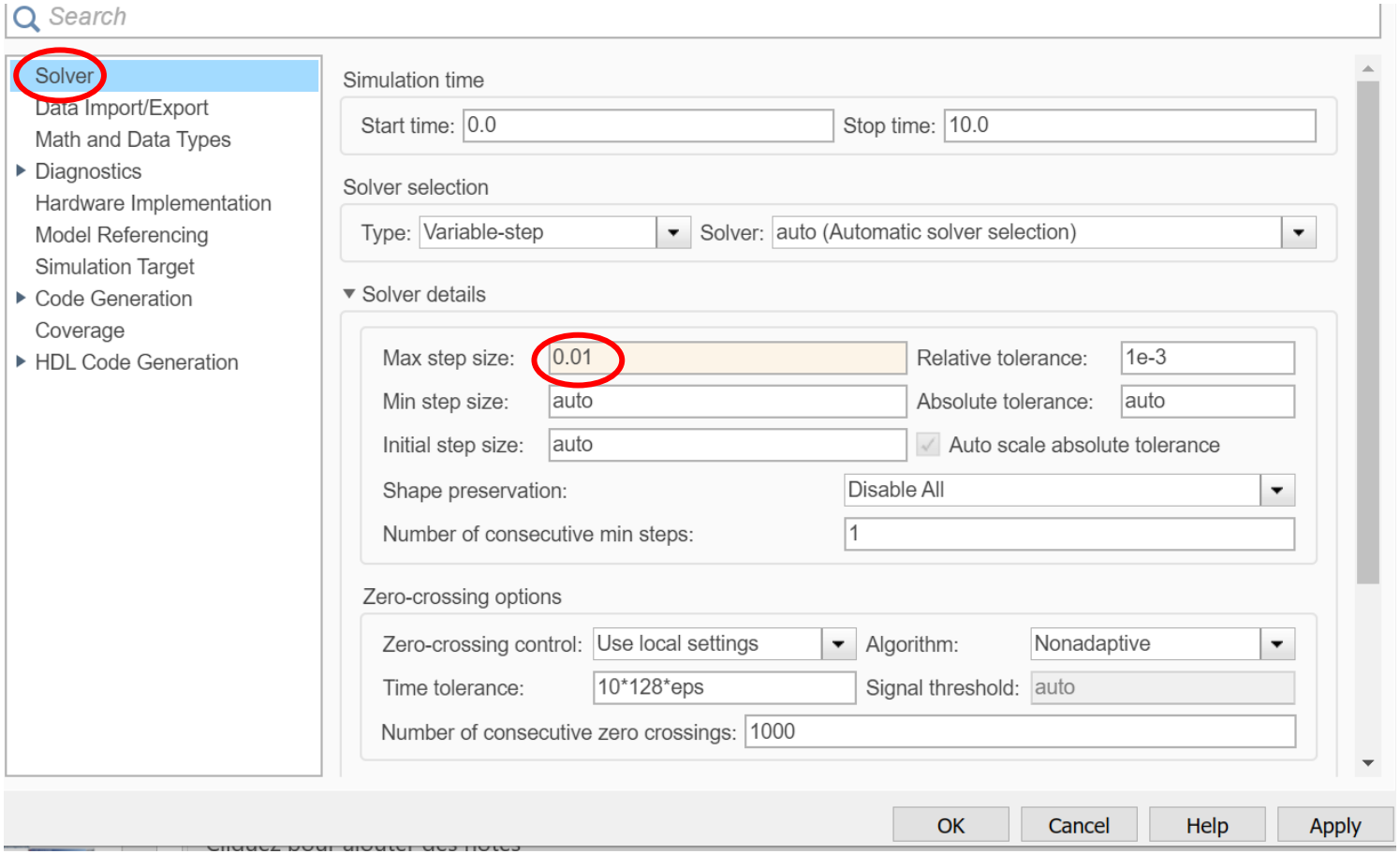
❑ Exemple #2:

- Affichage de plusieurs signaux sinusoïdaux



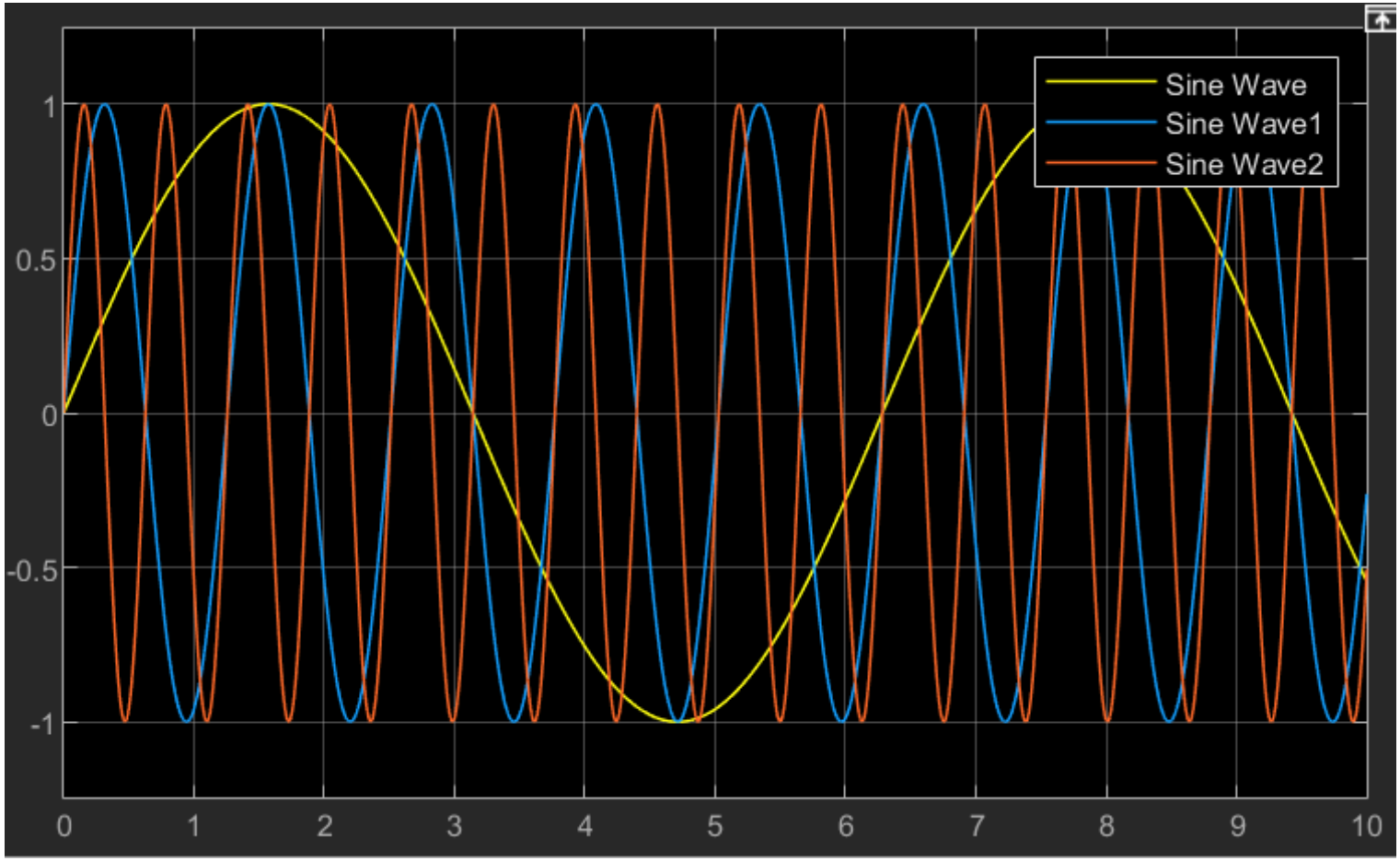
❑ Exemple #2:

- Pour augmenter la précision d'affichage, on clique sur les paramètres et en minimise *Max step size* dans le solveur:



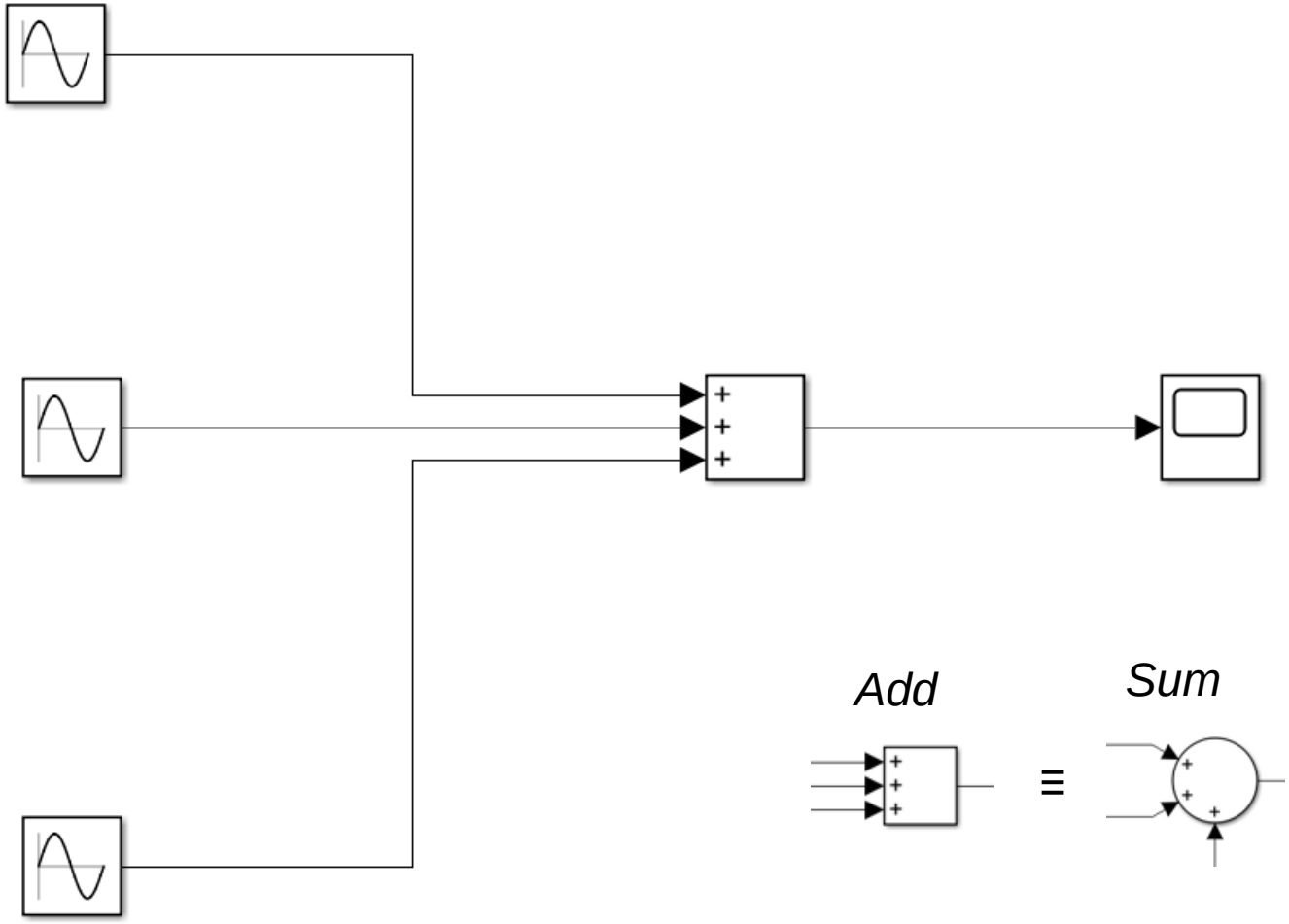
❑ Exemple #2:

- Affichage de plusieurs signaux sinusoïdaux



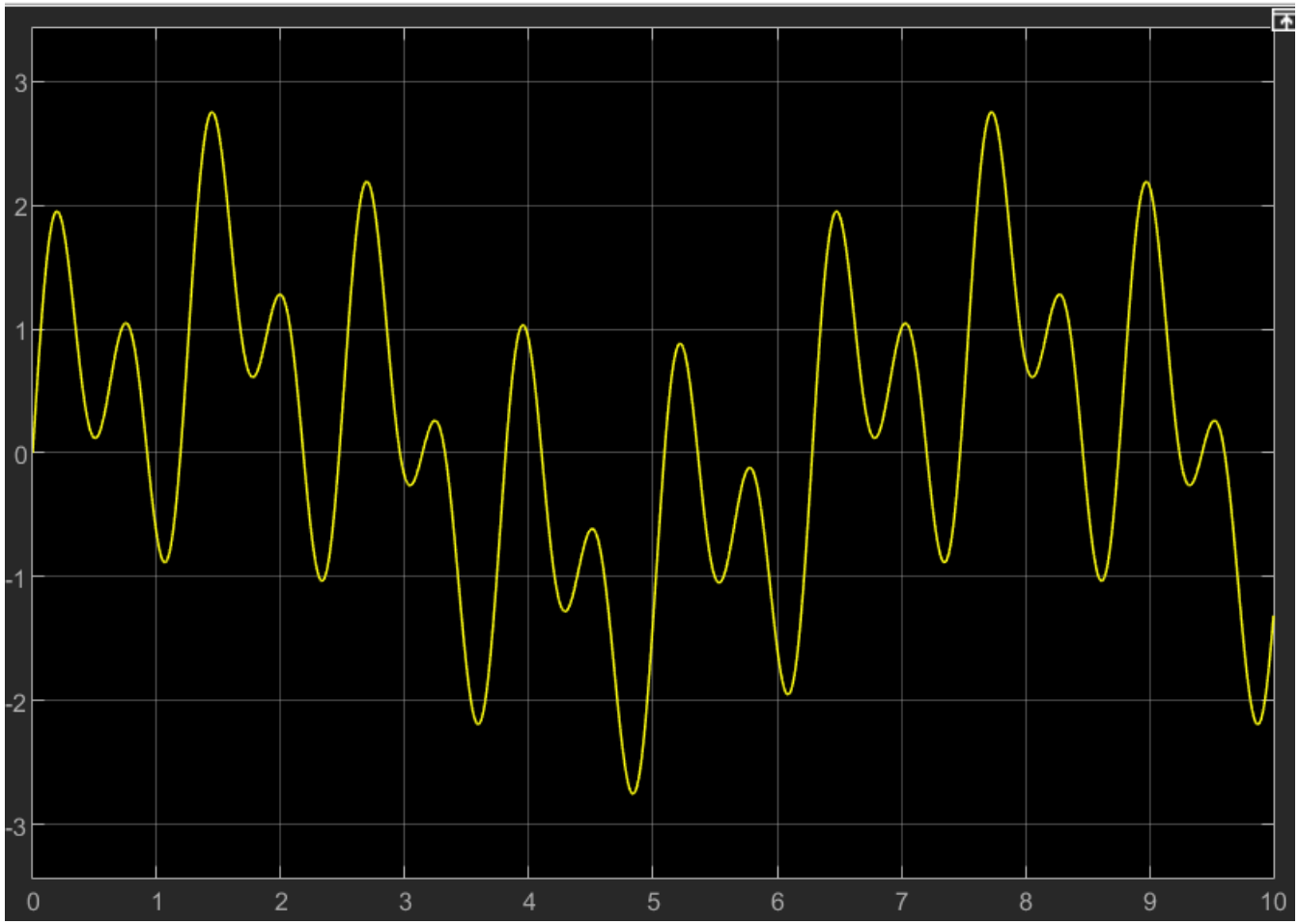
❑ Exemple #3:

- La somme de plusieurs signaux sinusoïdaux
- Exemple précèdent + 1 bloc de type *Add* ou *Sum* (bib. *Math Operations*).



❑ Exemple #3:

- La somme de plusieurs signaux sinusoïdaux



❑ Exemple #4:

- Affichage des résultats à *Workspace* Matlab
  - 1 bloc *Sine Wave* (bib. *Sources*)
  - 1 bloc *To Workspace* (bib. *Sinks*)

Workspace

Name	Value
out	1x1 Simulatio...

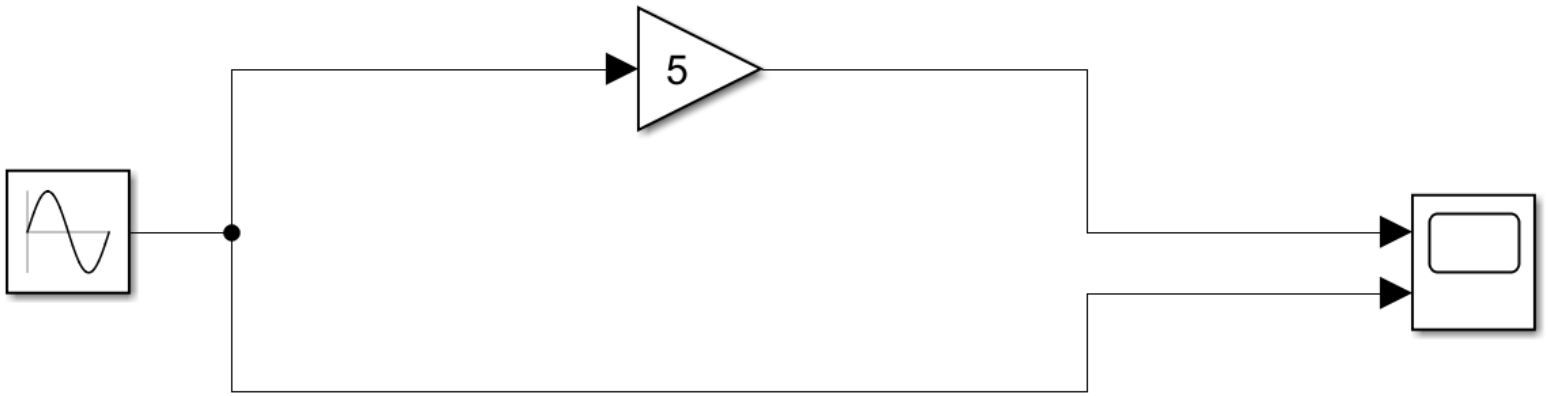
out.results

Time series name:

Time	Data:1
0	0
0.0100	0.0100
0.0200	0.0200
0.0300	0.0300
0.0400	0.0400
0.0500	0.0500
0.0600	0.0600
0.0700	0.0699
0.0800	0.0799
0.0900	0.0899
0.1000	0.0998
0.1100	0.1098
0.1200	0.1197

❑ Exemple #5:

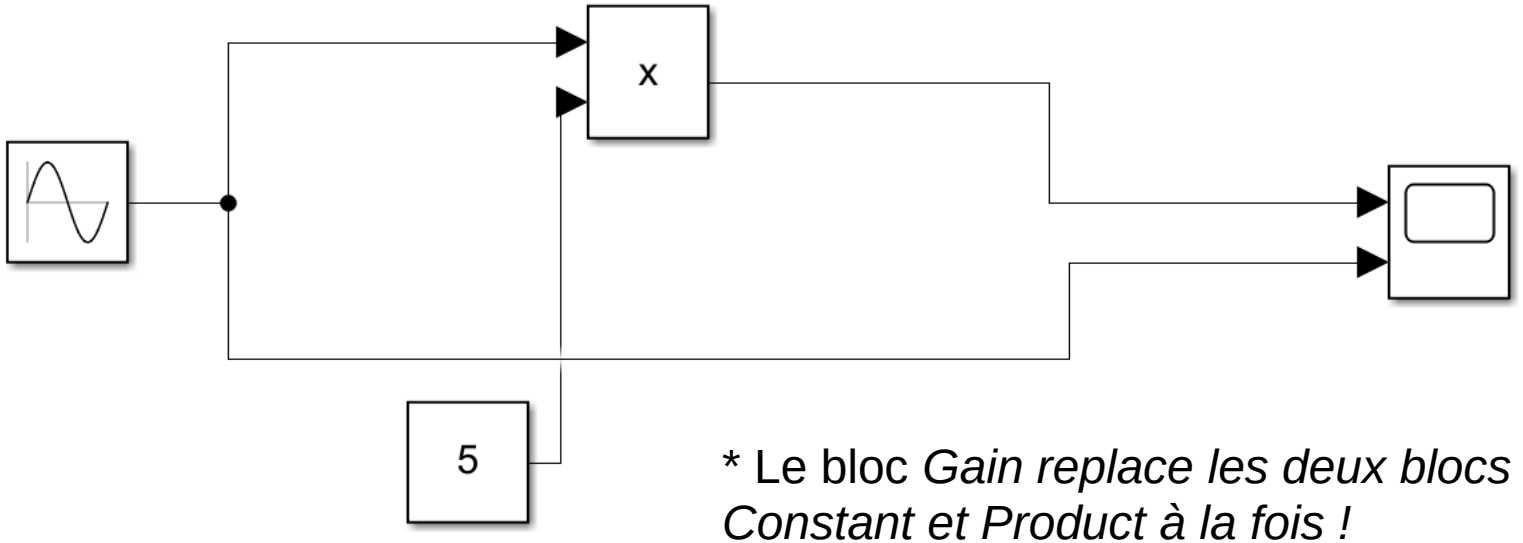
- Multiplication d'un signal par un nombre réel
  - 1 bloc *Sine Wave* (bib. *Sources*)
  - 1 bloc *Scope* (bib. *Sinks*)
  - **1 bloc *Gain* (bib. *Math Operations*)**



❑ Exemple #5:

- Multiplication d'un signal par un nombre réel
  - 1 bloc *Sine Wave* (bib. *Sources*)
  - 1 bloc *Scope* (bib. *Sinks*)
  - **1 bloc *Constant* (bib. *Sources*)**
  - **1 bloc *Product* (bib. *Math Operations*)**

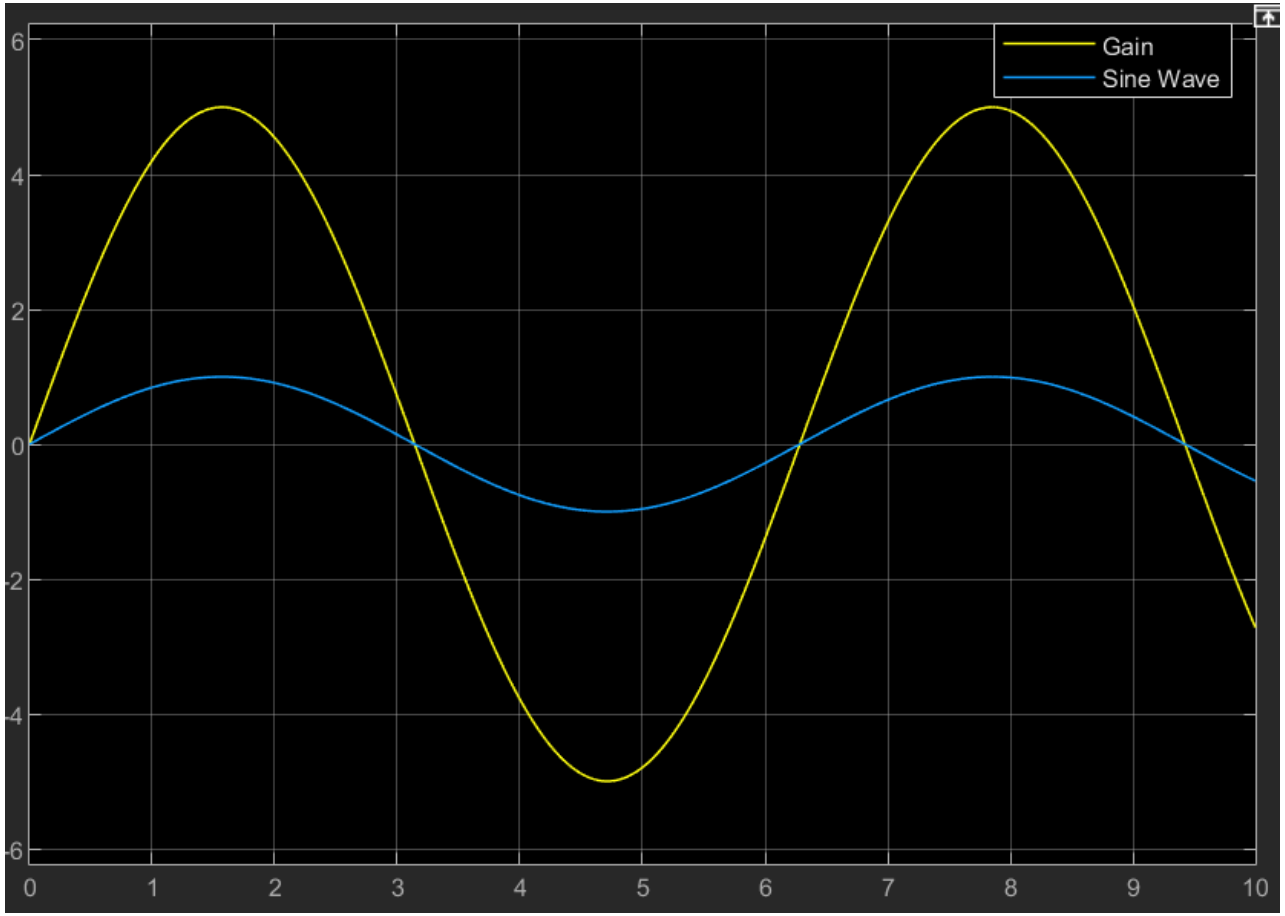
Schémas équivalent



* Le bloc *Gain replace les deux blocs Constant et Product à la fois !*

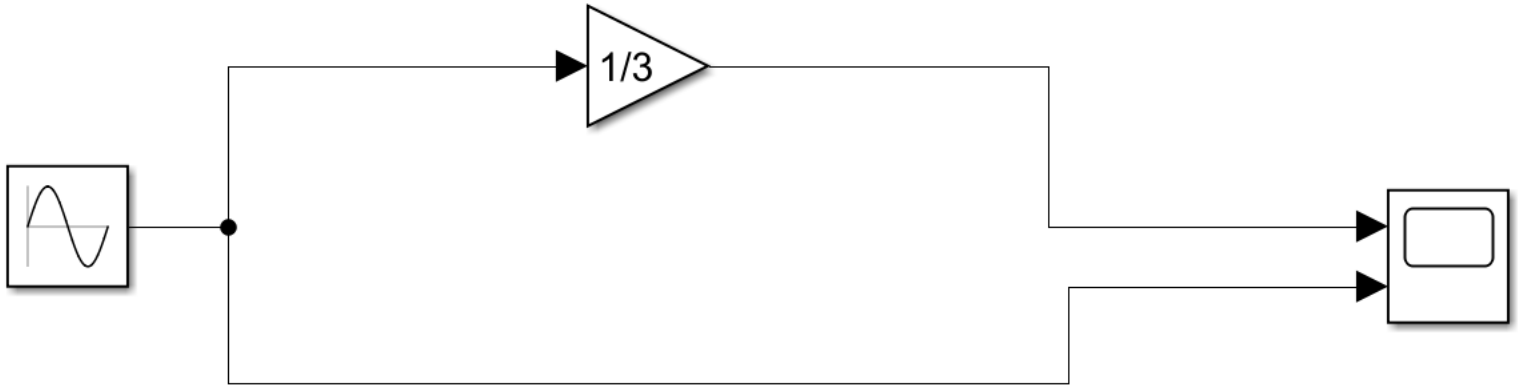
❑ Exemple #5:

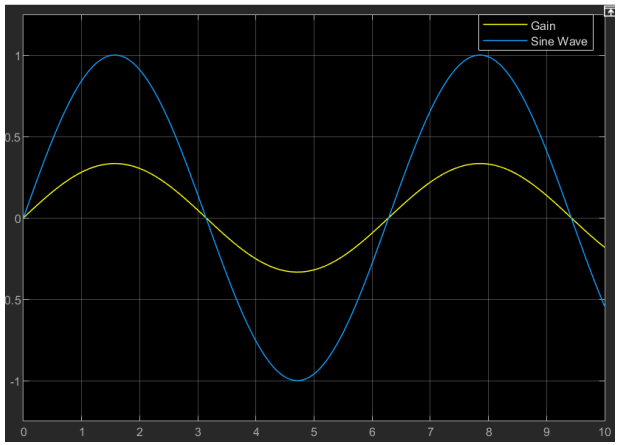
- Multiplication d'un signal par un nombre réel



❑ Exemple #5:

- Multiplication d'un signal par un nombre réel
  - Remarque: Le multiplicateur (Gain) peut être utilisé pour la multiplication ou la **division** d'un signal.



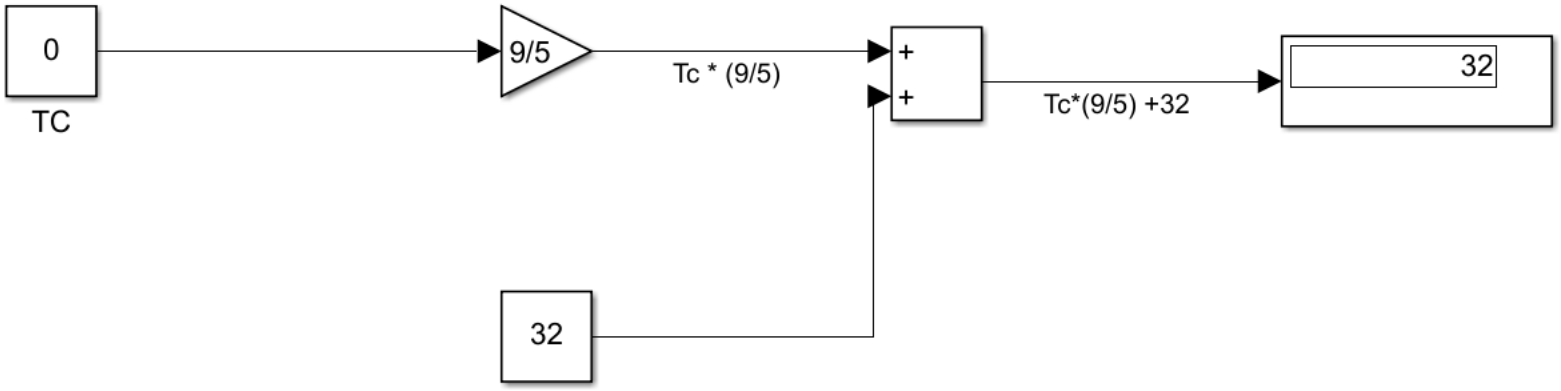


❑ Exemple #6:

- En utilisant le Simulink, réaliser un convertisseur qui transforme la température de °**C** en °**F**. sachant que :

$$\boldsymbol{T(°F) = \frac{9}{5} T(°C) + 32}$$

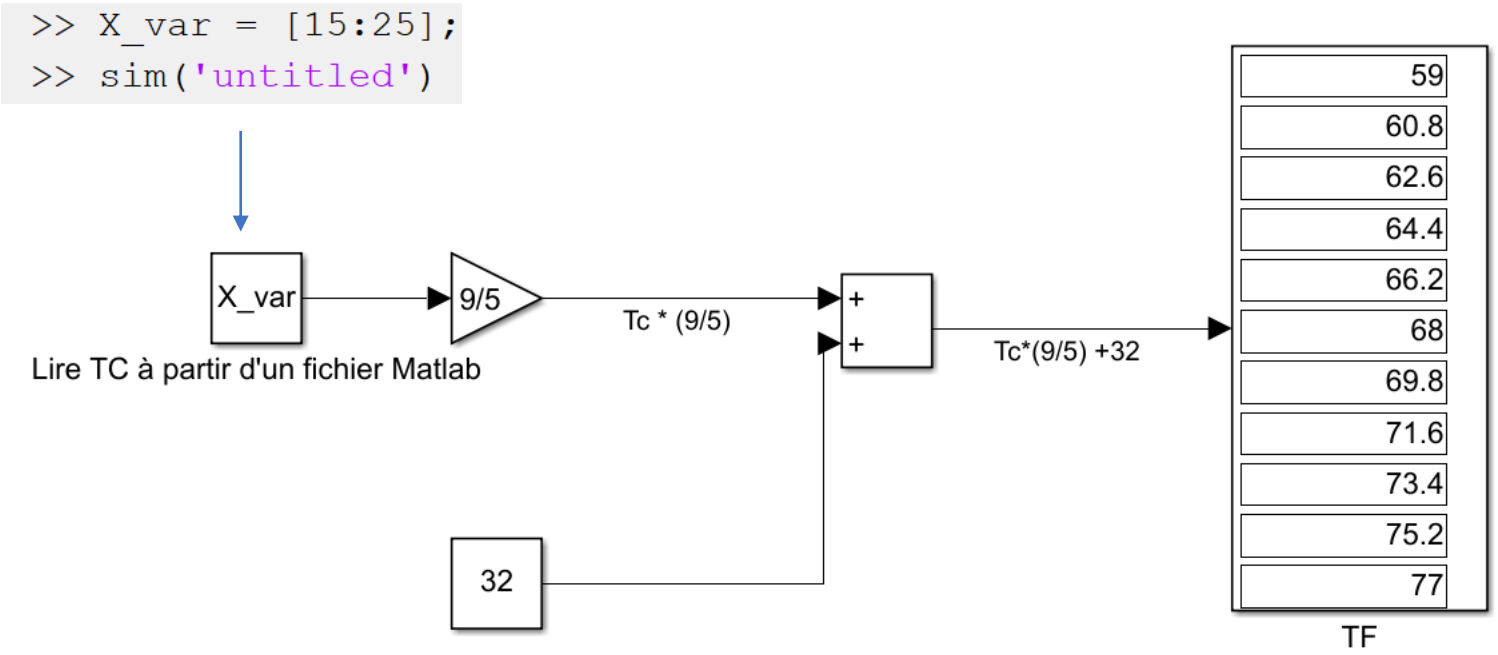
  - 2 bloc *Constant* (bib. *Sources*)
  - 1 bloc *Display* (bib. *Sinks*)
  - 1 bloc *Gain* (bib. *Math Operations*)
  - 1 bloc *Add* (bib. *Math Operations*)



❑ Exemple #6:

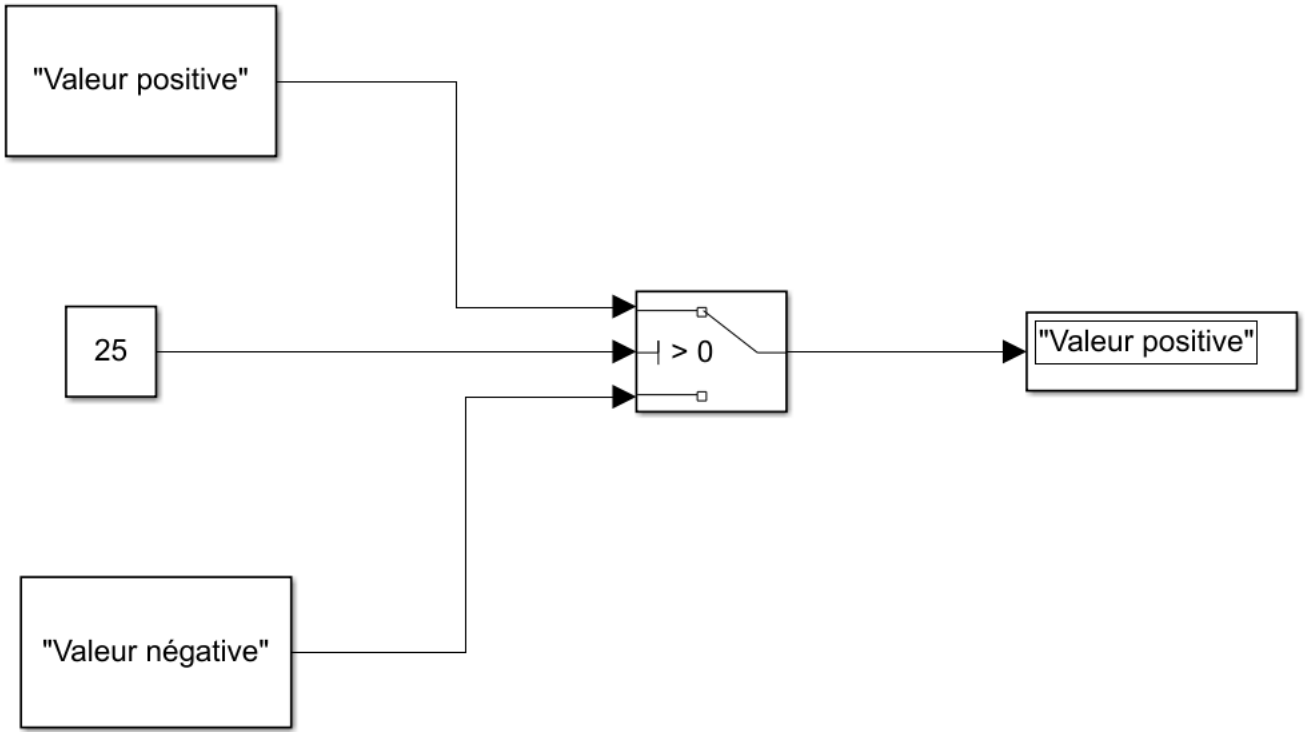
- En utilisant le Simulink, réaliser un convertisseur qui transforme la température de °$\mathbf{C}$ en °$\mathbf{F}$.

On peut lire les données $\mathbf{T}(°\mathbf{C})$ à partir d'un fichier Matlab ou Excell.



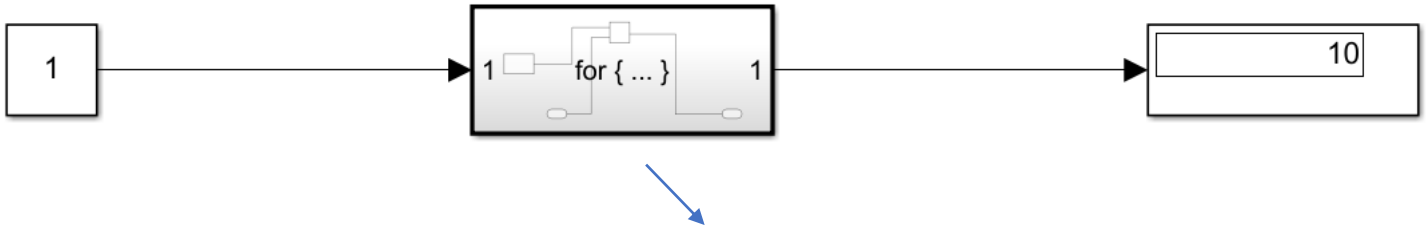
❑ Exemple #7:

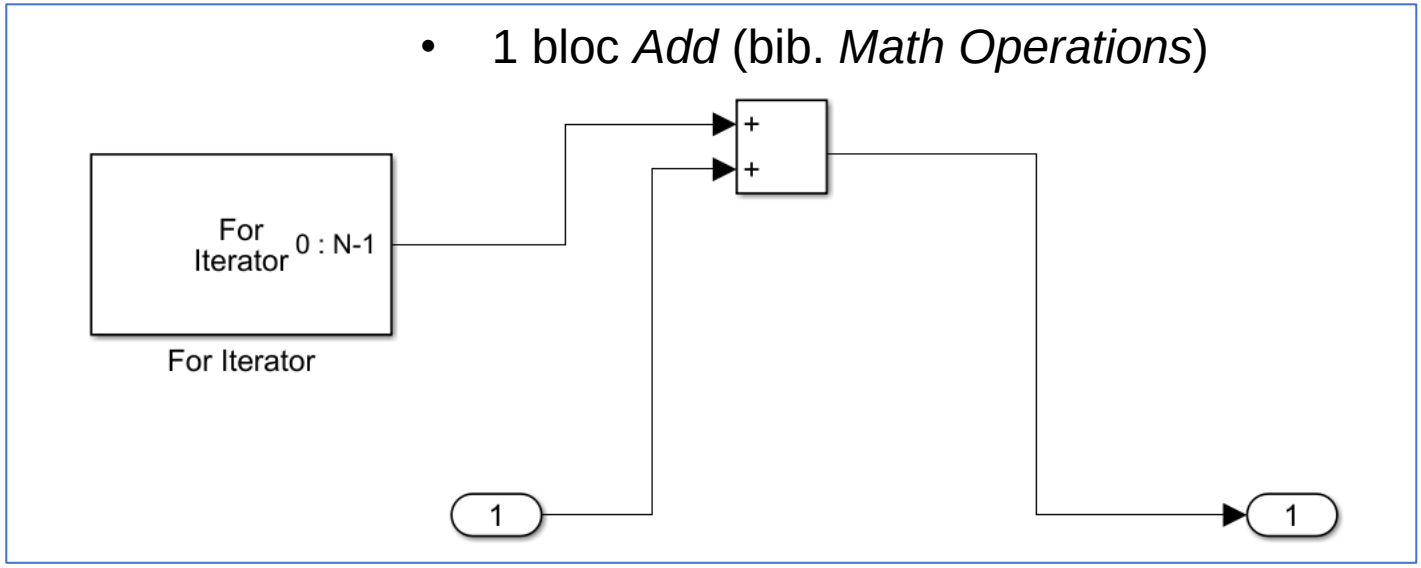
- Instruction conditionnelle (IF):
  - 2 blocs *Constant String* (bib. *String*)
  - 1 bloc *Display* (bib. *Sinks*)
  - 1 bloc *Constant* (bib. *Sources*)
  - 1 bloc *Switch* (bib. *Signal Routing*)



❑ Exemple #7:

- Instruction répétitive (For):
  - 1 bloc *Constant* (bib. *Sources*)
  - 1 bloc *Display* (bib. *Sources*)
  - 1 bloc *For Iterator Subsystem* (bib. *Ports and Subsystems*)

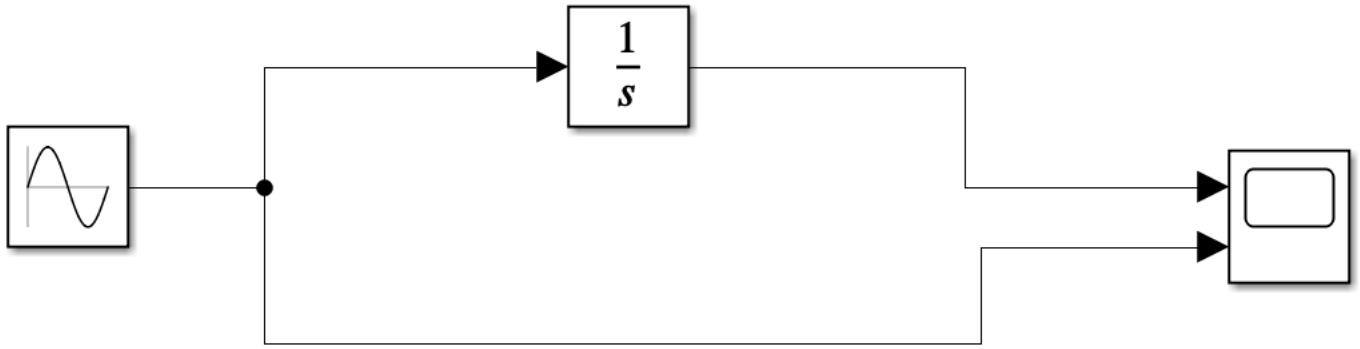




Sous-système

❑ Exemple #8:

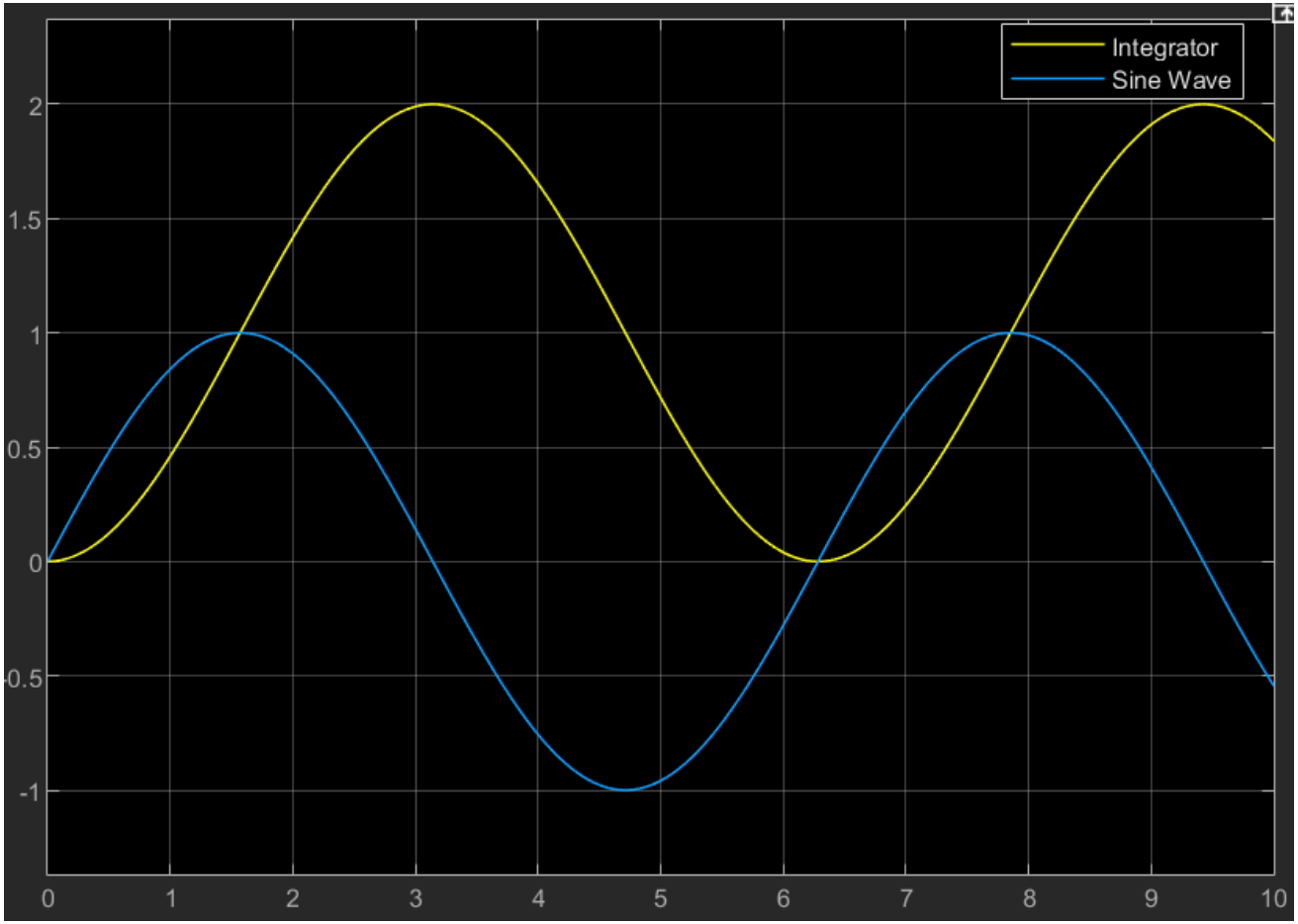
- Intégration d'un signal
  - 1 bloc *Sine Wave* (bib. *Sources*)
  - 1 bloc *Scope* (bib. *Sinks*)
  - 1 bloc *Integrator* (bib. *Commonly used*)



* L'intégrateur est utilisé pour résoudre les équations différentielles sur Simulink.

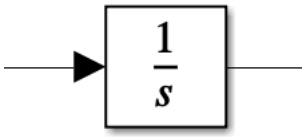
❑ Exemple #8:

- Intégration d'un signal

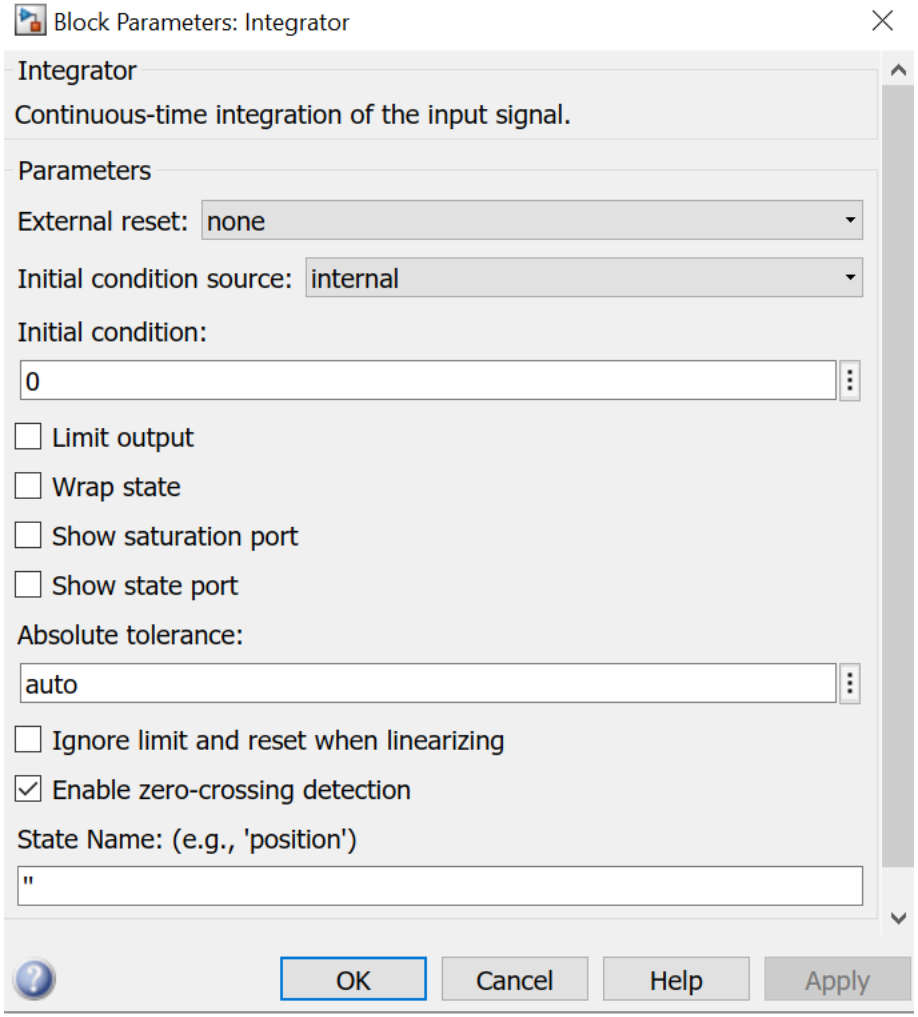


❑ Exemple #8:

- Intégration d'un signal



- On peut changer la valeur initiale dans les paramètres de l'intégrateur



Résolution des Équations Différentielles sur SIMULINK

❑ Résolution des équations différentielles sur Simulink

Exemple #1: à titre d'exemple, nous allons utiliser Simulink pour résoudre l'équation différentielle du premier ordre (ODE) suivante:

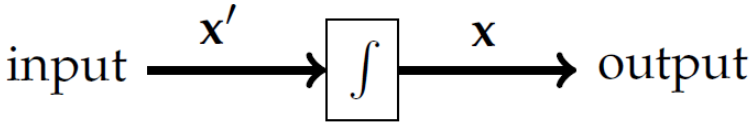
$$2\frac{dx}{dt} = 3 \quad \text{Avec : } x\,(0) = 0$$

- Les étapes de résolution du problème sont résumées comme suit:

Etape 1: Écrire le terme avec la plus haute dérivé en fonction du reste:

$$\frac{dx}{dt} = \frac{3}{2}$$

Etape 2: Insérer un intégrateur pour calculer la valeur de x, la condition initiale est par défaut x(0)= 0.



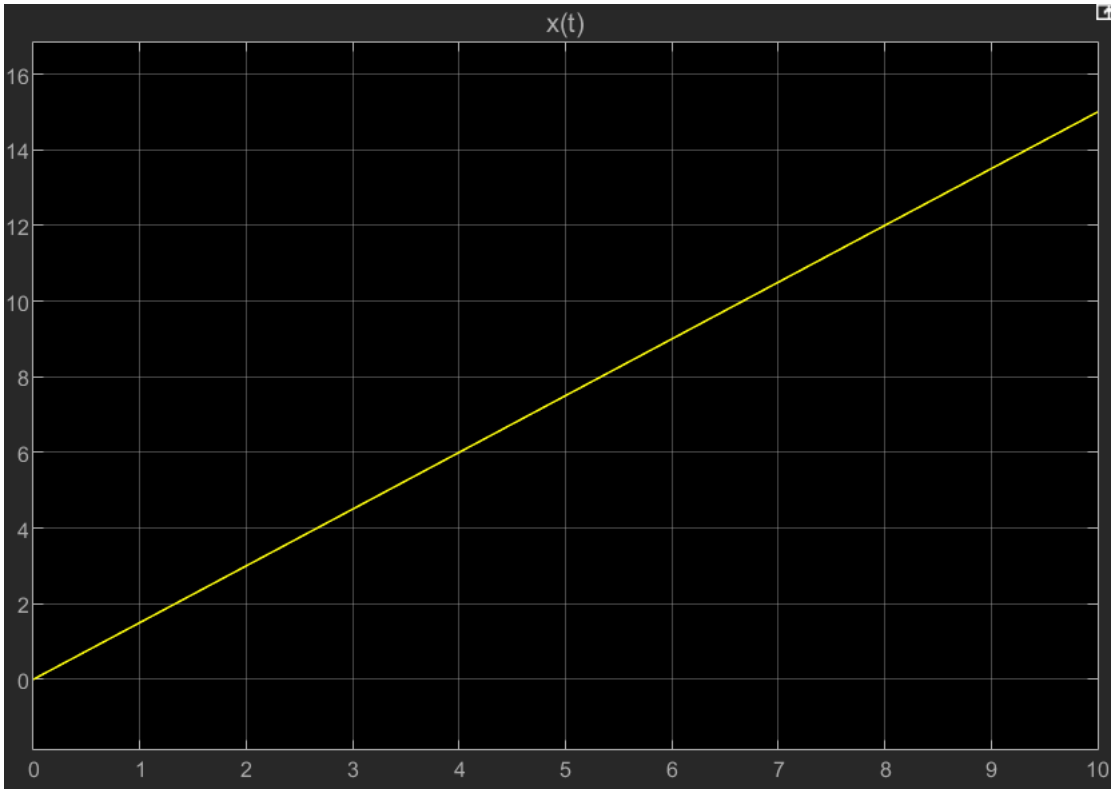
Etape 3: Insérer un *Constant* pour attribuer la valeur 3/2.

Etape 4: Connecter les blocs à un *Scope* pour afficher les résultats.

❑ Résolution des équations différentielles sur Simulink

Exemple #1:

$$x(0) = 0$$



❑ Résolution des équations différentielles sur Simulink

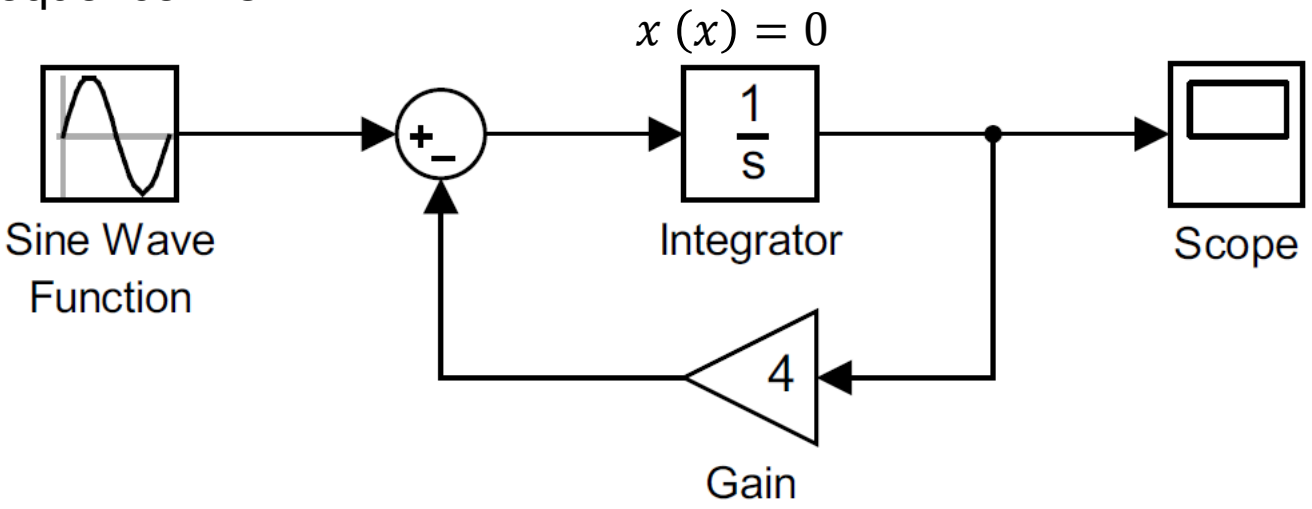
Exemple #2: Équation du 1er ordre- Mouvement sinusoïdal

On considère l'équation différentielle suivante:

$$\frac{dx}{dt} = 2\sin(3t) - 4x \qquad \text{Avec : } x\,(x) = 0$$

Schémas du système:

Amplitude = 2.
Fréquence = 3.

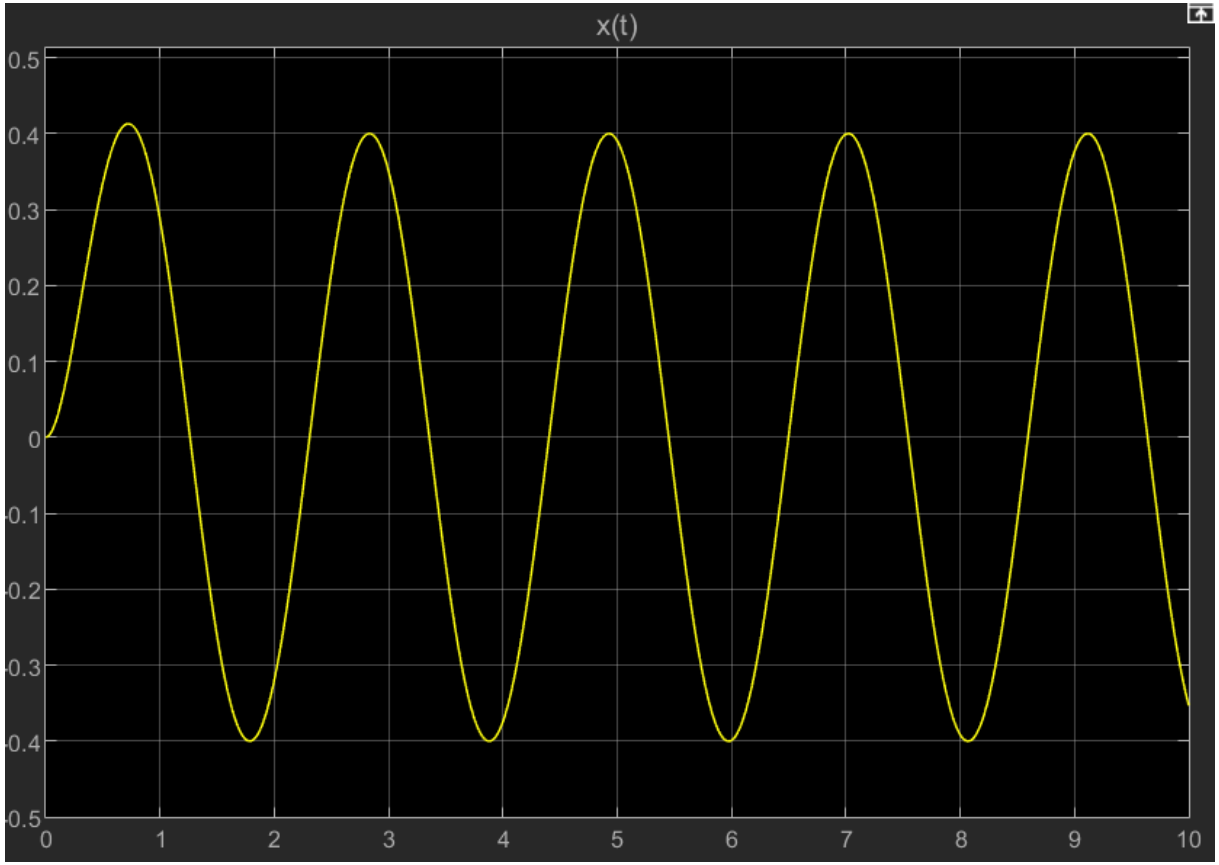


❑ Résolution des équations différentielles sur Simulink

Exemple #2: Équation du 1er ordre- Mouvement sinusoïdal

$$\frac{dx}{dt} = 2\sin(3t) - 4x$$

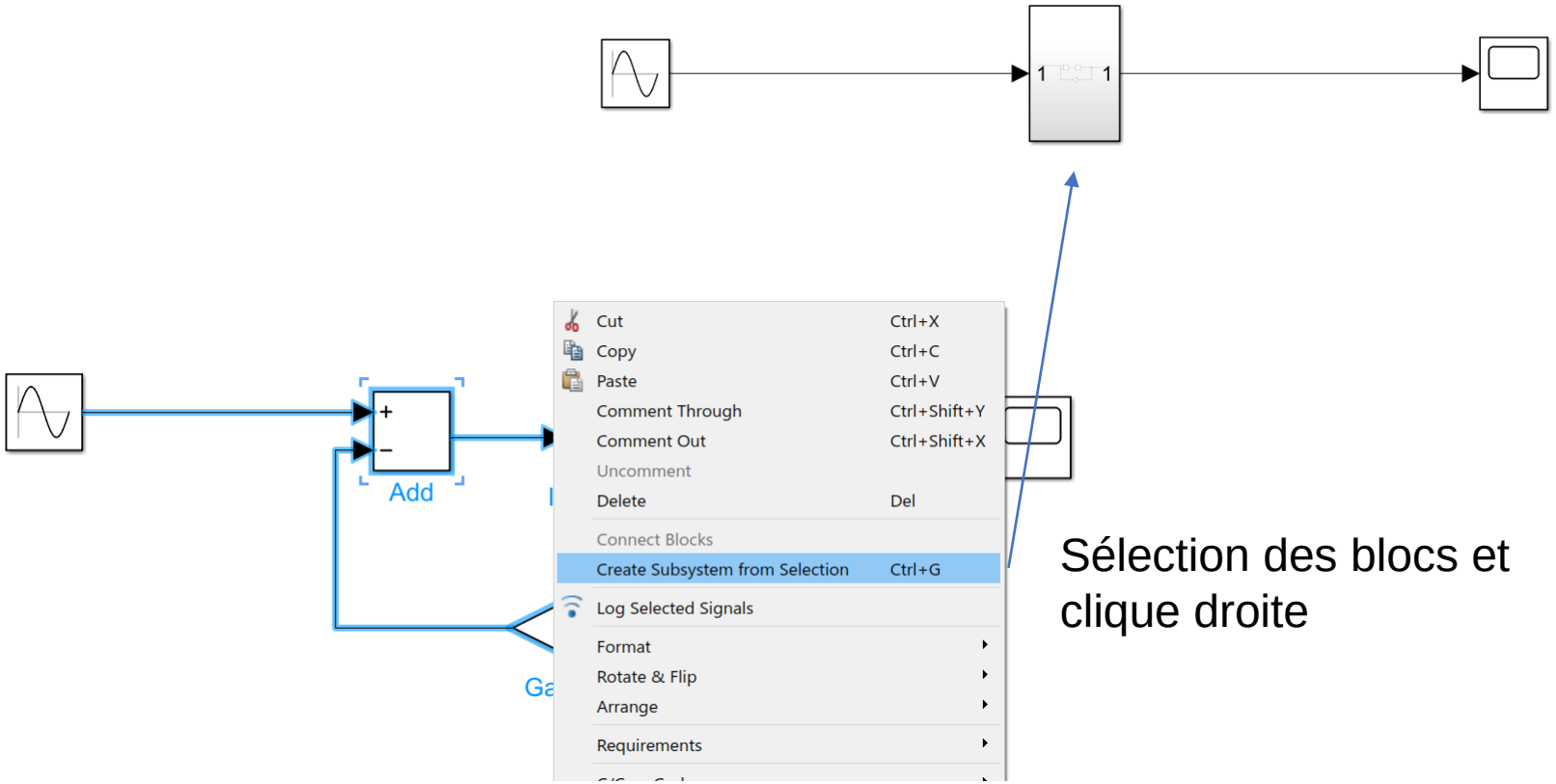
Avec : $x\,(x) = 0$



❑ Résolution des équations différentielles sur Simulink

Remarque:

On peut réduire la taille du système en créant un sous-système:



Sélection des blocs et clique droite

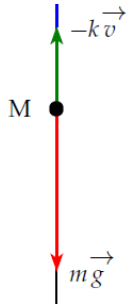
❑ Résolution des équations différentielles sur Simulink

Exemple #3: Équation du 1er ordre- Chute libre

L'équation gouvernante de chute d'un corps solide dans l'air est:

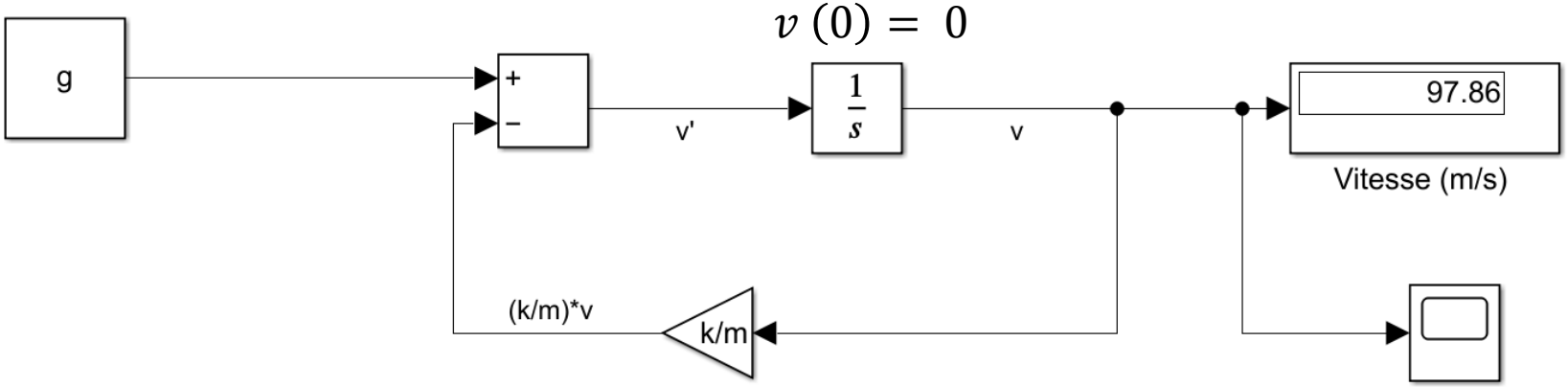
$$m\frac{dv}{dt} = -kv + mg \quad \text{Ou:} \quad \frac{dv}{dt} = -\frac{k}{m}v + g \quad \text{Avec: } v(0) = 0$$

```
>>  m = 2;
>> k = 0.2;
>> g = 9.81;
```



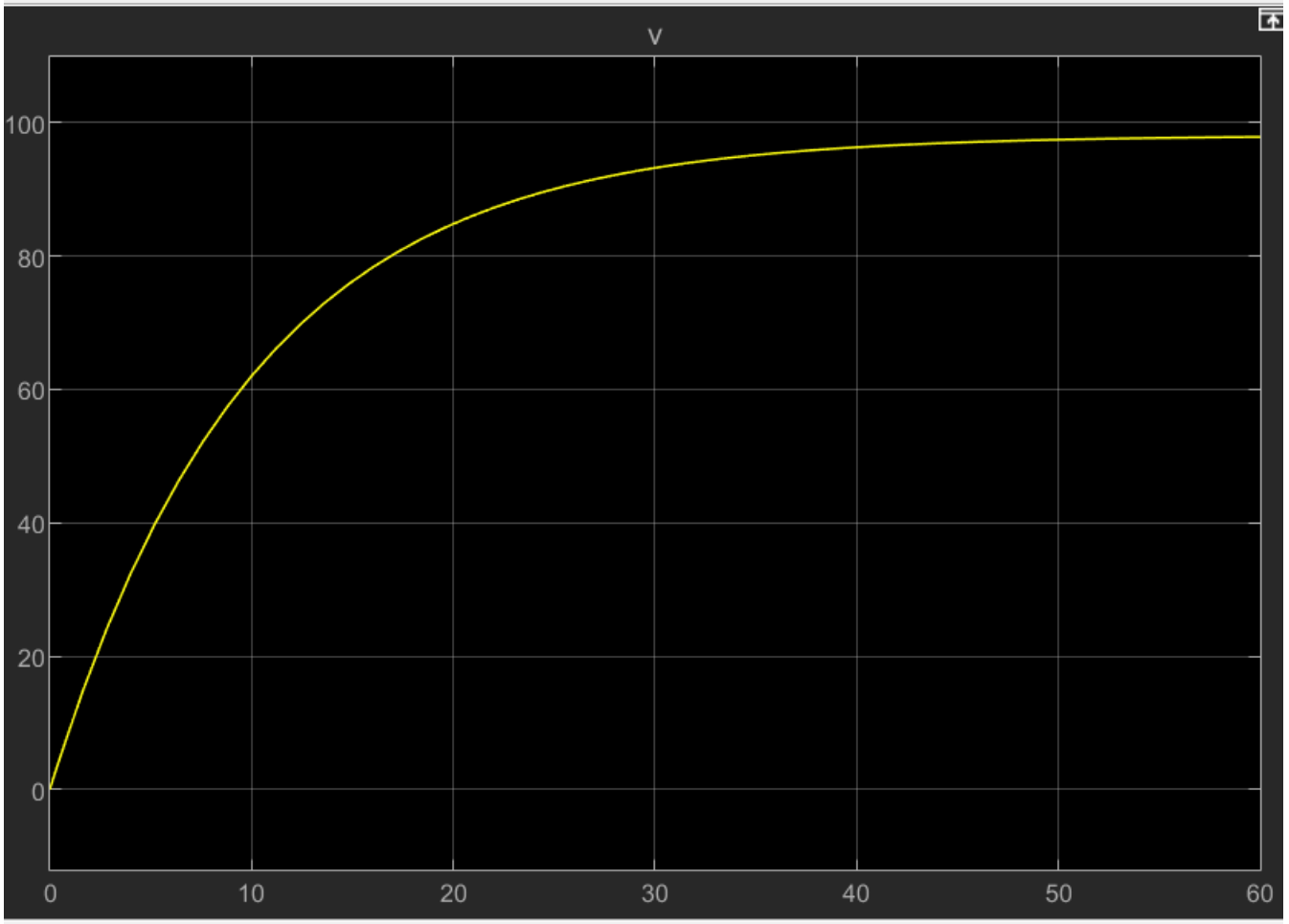
Schémas du système:

$$v(0) = 0$$



❑ Résolution des équations différentielles sur Simulink

Exemple #3: Équation du 1er ordre- Chute d'un corps dans l'air



❑ Résolution des équations différentielles sur Simulink

Exemple #4: Équation du 2nd ordre- Mouvement harmonique simple amorti

L'équation gouvernante d'un mouvement harmonique simple est:

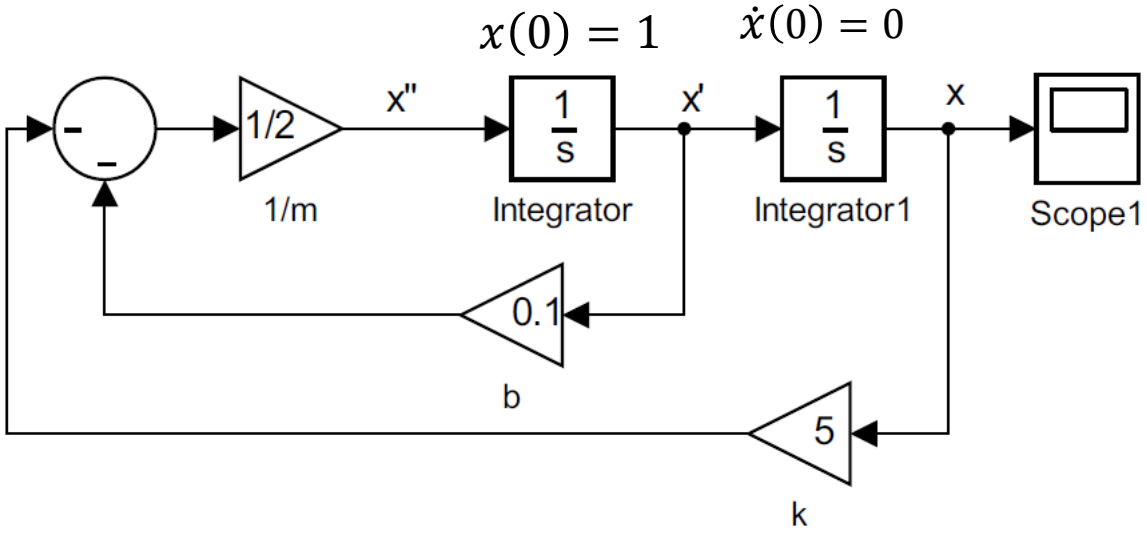
$$m\ddot{x} + b\dot{x} + kx = 0 \qquad x(0) = 1 \quad \dot{x}(0) = 0$$

b : constante d'amortissement ($b = 0.1$).

m: masse du système (i.g. $m = 2\ kg$)

k: coefficient du ressort (i.g. $k = 5$)

Schémas du système:

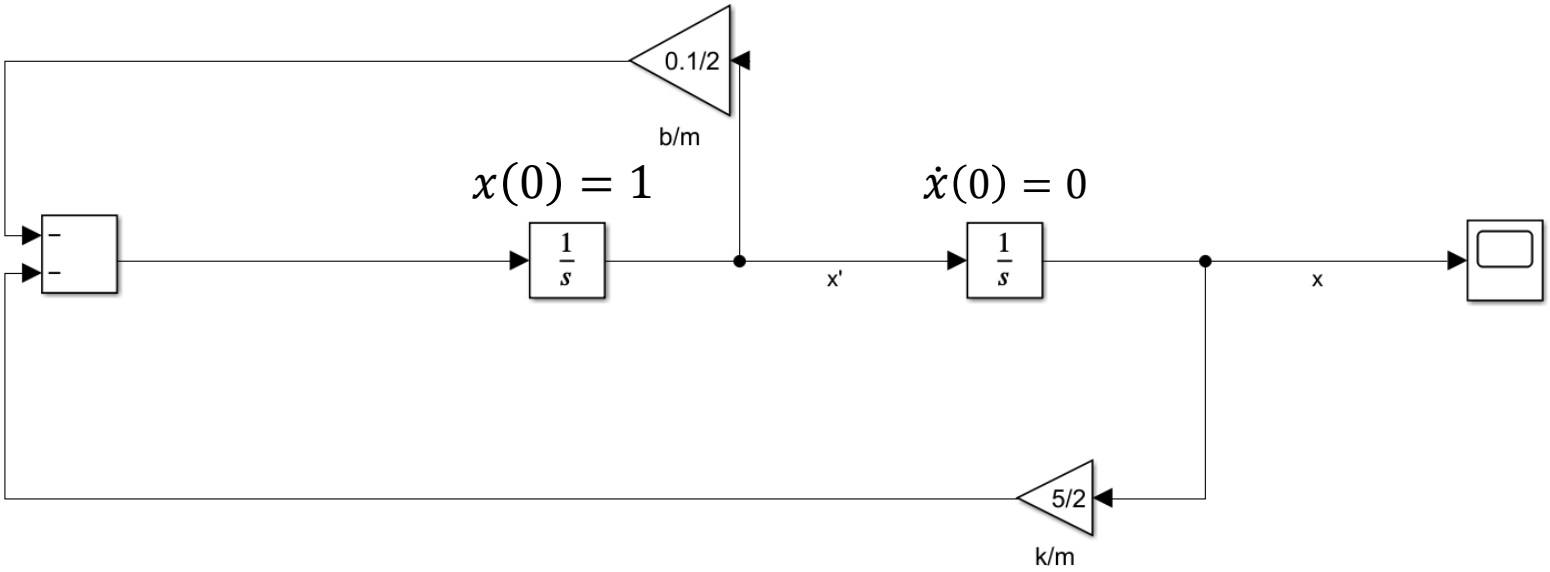


❑ Résolution des équations différentielles sur Simulink

Exemple #4: Équation du 2nd ordre- Mouvement harmonique simple amorti

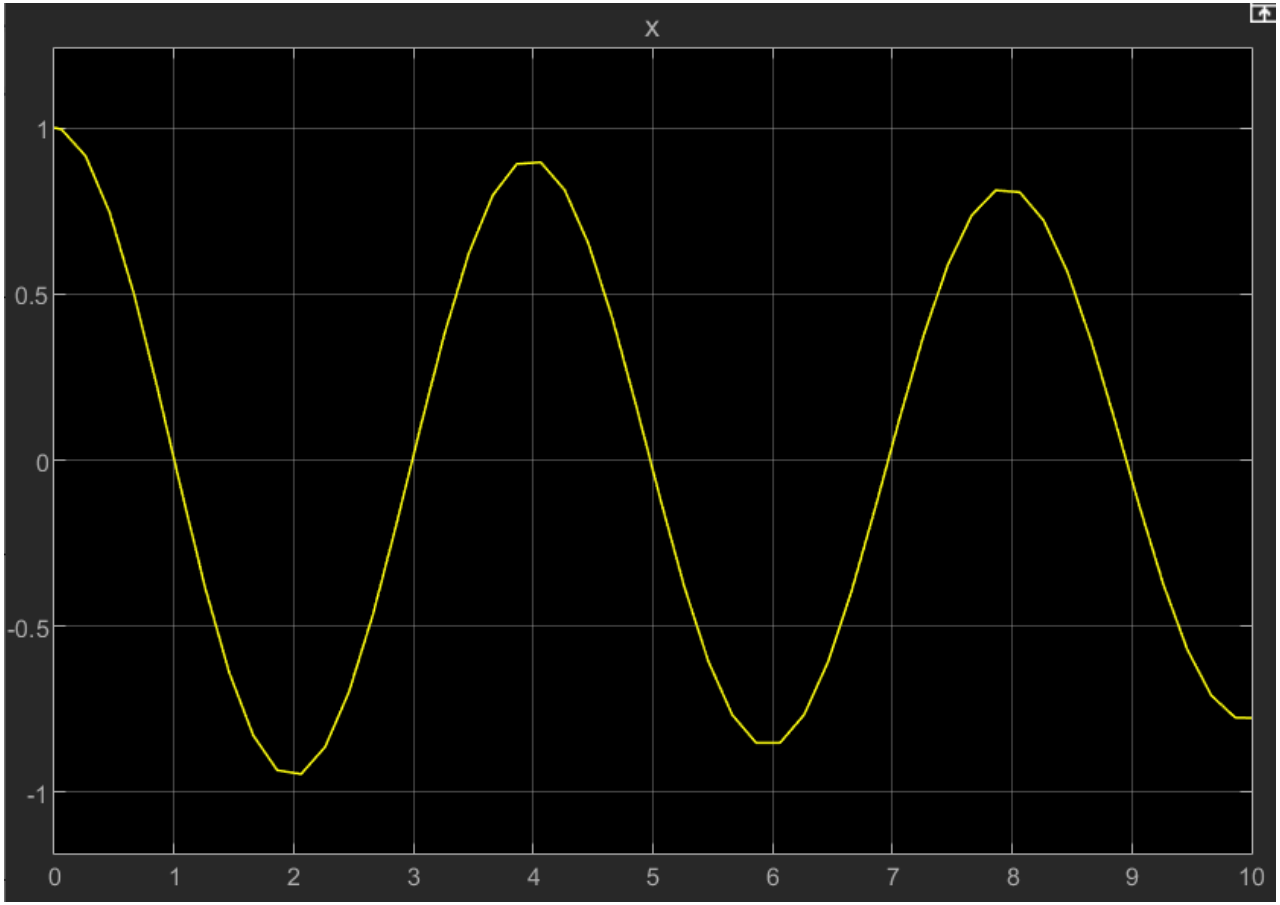
$$m\ddot{x} + b\dot{x} + kx = 0 \qquad x(0) = 1 \quad \dot{x}(0) = 0$$

Schémas équivalent du système:

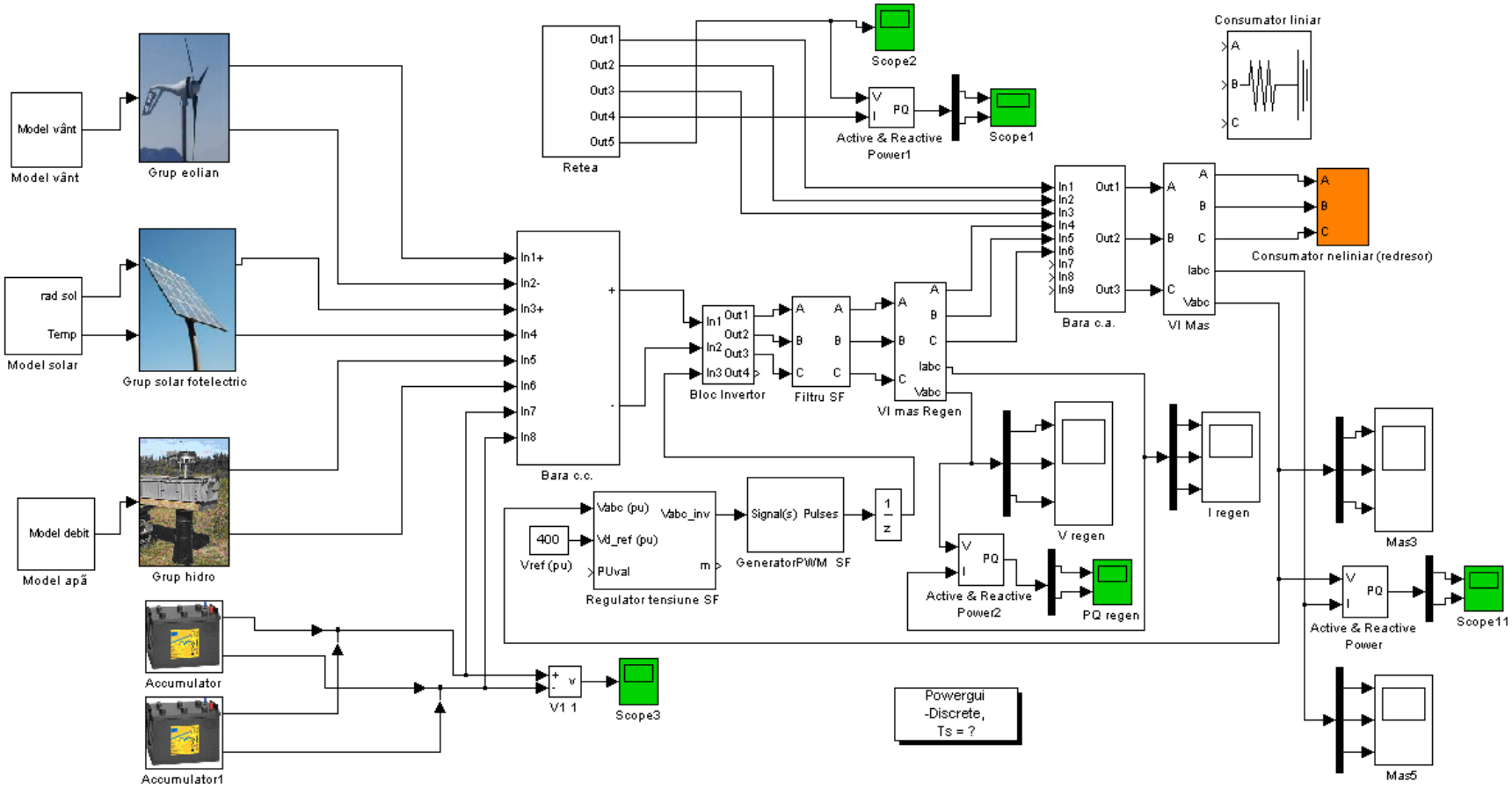


❑ Résolution des équations différentielles sur Simulink

Exemple #4: Équation du 2nd ordre- Mouvement harmonique simple amorti

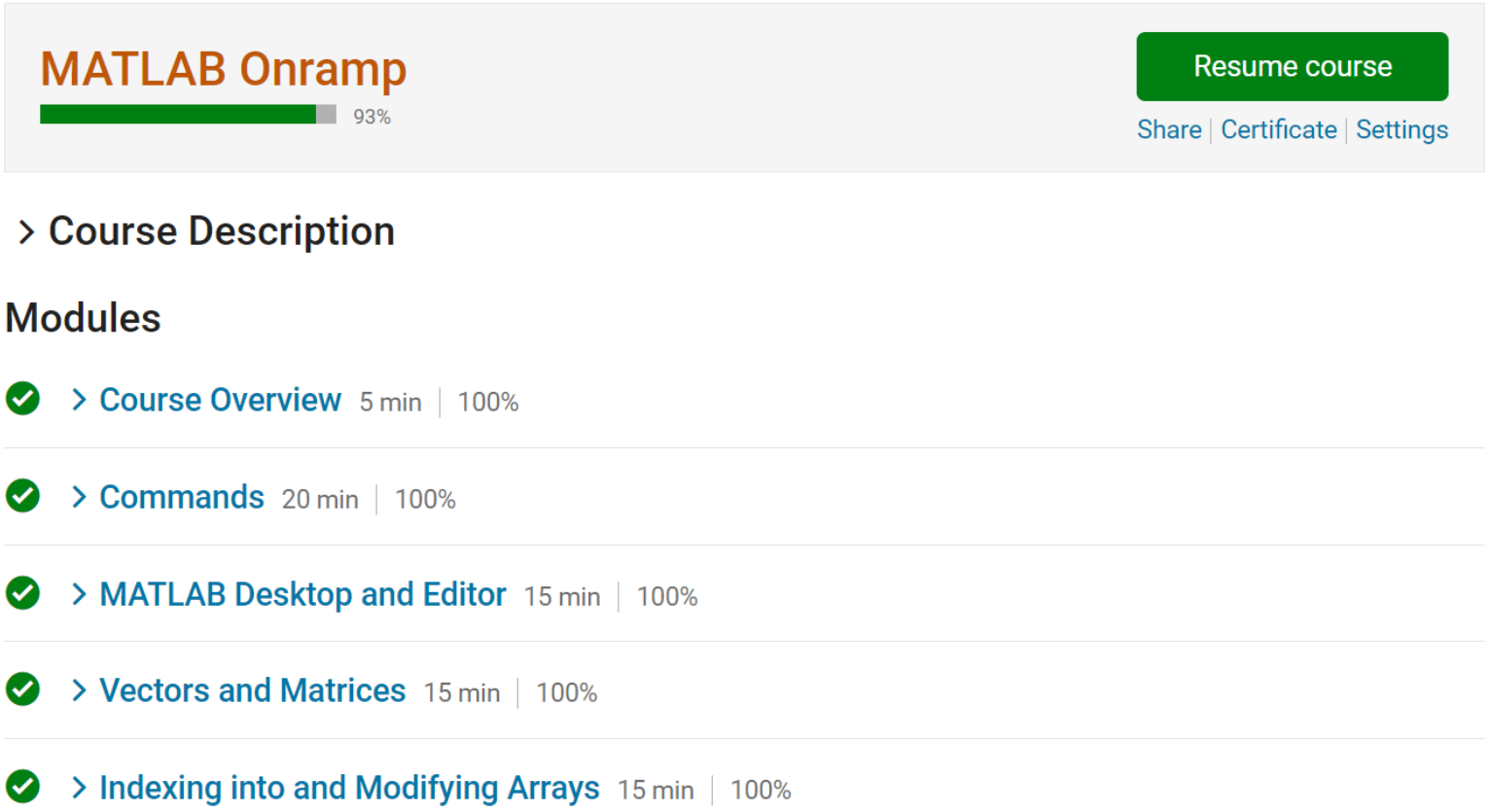


- Exemple d'un modèle Simulink d'un système hybride basé sur les énergies renouvelables

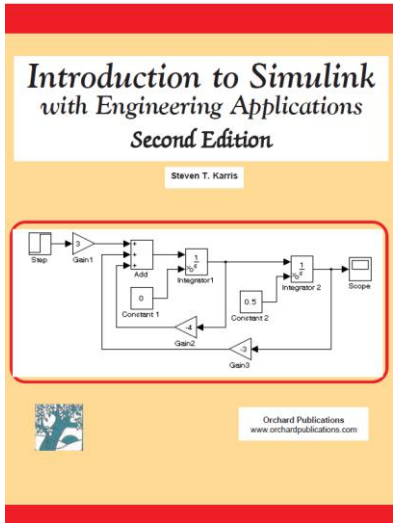


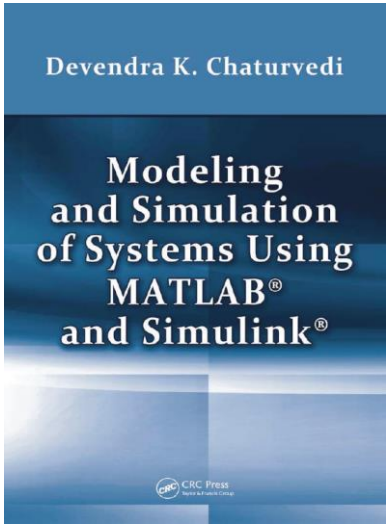
Matlab Onramp: Certificat de Matlab

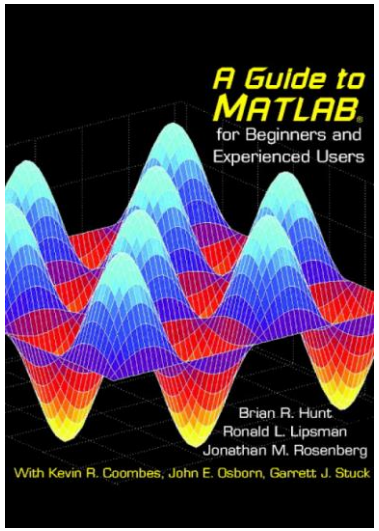
https://matlabacademy.mathworks.com/details/matlab-onramp/gettingstarted

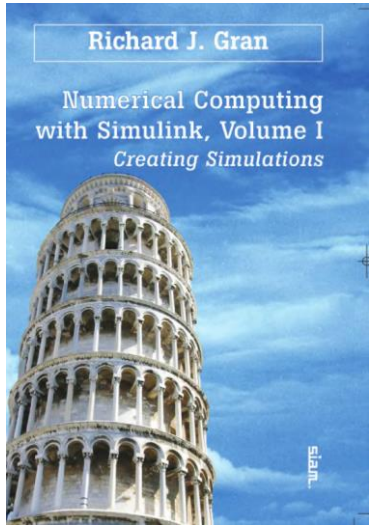


Références

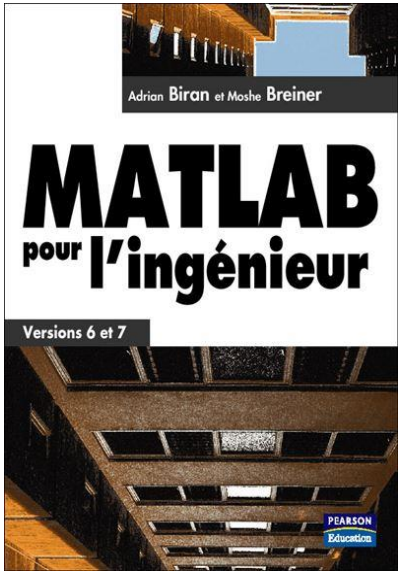

Introduction to Simulink
with Engineering Applications
Second Edition
Steven T. Karris
Orchard Publications
www.orchardpublications.com

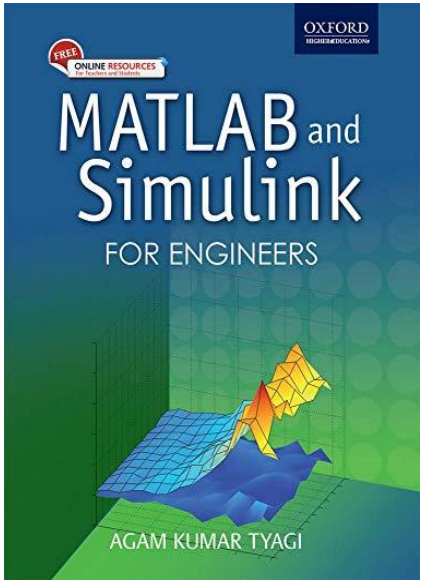

Devendra K. Chaturvedi
Modeling
and Simulation
of Systems Using
MATLAB®
and Simulink®
CRC Press

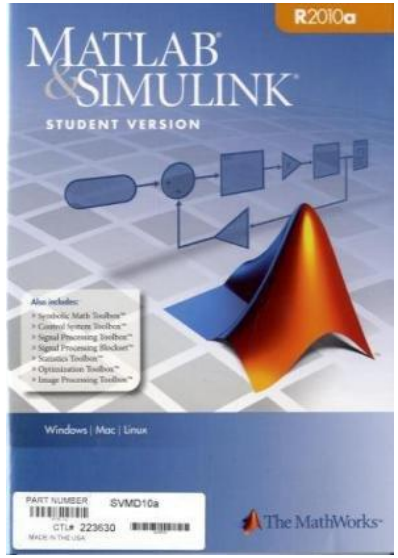

A Guide to
MATLAB
for Beginners and
Experienced Users
Brian R. Hunt
Ronald L. Lipsman
Jonathan M. Rosenberg
With Kevin R. Coombes, John E. Osborn, Garrett J. Stuck


Richard J. Gran
Numerical Computing
with Simulink, Volume I
Creating Simulations
siam


MATLAB
Notes for Professionals
100+ pages
of professional hints and tricks
GoalKicker.com
Free Programming Books


Adrian Biran et Moshe Breiner
MATLAB
pour l'ingénieur
Versions 6 et 7
PEARSON
Education


OXFORD
HIGHER EDUCATION
FREE
ONLINE RESOURCES
MATLAB and
Simulink
FOR ENGINEERS
AGAM KUMAR TYAGI


R2010a
MATLAB
&SIMULINK
STUDENT VERSION
Windows | Mac | Linux
The MathWorks

Fin