

Chapter 2

Aspects techniques de la S.S.D.

2.1 Formalisation des équations, le calcul pas à pas.

2.1.1 Systèmes d'équations différentielles. Méthodes de calcul numérique.

2.1.1.1 Equations différentielles ordinaires et système d'équation différentielles.

On cherche à calculer numériquement
un système d'équations différentielles du premier ordre.

Toute équation différentielle d'ordre n

$$y^{(n)} = F\left(t, y, y', \dots, y^{(n-1)}\right) \quad (2.1-1)$$

peut toujours se ramener à un système de n équations du premier ordre (et inversement).

On peut introduire les variables

$$x_1 = y \quad x_2 = y' = \dot{y} \quad x_3 = y'' = \ddot{y} \dots \dots x_n = y^{(n-1)} \quad (2.1-2)$$

on a le système de n équations différentielles ordinaires

$$\left| \begin{array}{ll} \dot{x}_1 = x_2 & \dot{y} \\ \dot{x}_2 = x_3 & \ddot{y} \\ \vdots & \vdots \\ \dot{x}_{n-1} = x_n & y^{(n-1)} \\ \dot{x}_n = F(t, x_1, \dots, x_n) & y^{(n)} \end{array} \right. \quad (2.1-3)$$

cas particulier du système plus général

$$\left\{ \begin{array}{l} \dot{x}_1 = F_1(t, x_1, \dots, x_n) \\ \dot{x}_2 = F_2(t, x_1, \dots, x_n) \\ \vdots \\ \dot{x}_{n-1} = F_{n-1}(t, x_1, \dots, x_n) \\ \dot{x}_n = F_n(t, x_1, \dots, x_n) \end{array} \right. \quad (2.1-4)$$

2.1.1.2 Une méthode numérique élémentaire. La méthode d'Euler ou méthode de la tangente.

Soit un problème du premier ordre aux valeurs initiales.

$$\frac{dy}{dt} = f(y, t) \quad y(t_0) = y_0$$

Théorème Il existe une solution unique $y = \varphi(t)$ du problème donné sur un certain intervalle contenant t_0

Il existe un algorithme de calcul permettant d'obtenir les valeurs approximatives

$$y_0, y_1, \dots, y_n, \dots$$

de la solution φ pour un ensemble de points

$$t_0 < t_1 < \dots < t_n < \dots$$

régulièrement espacés d'une valeur h .

$$\begin{array}{ll} y_n & \longrightarrow \text{valeur approximative} \\ \varphi(t_n) & \longrightarrow \text{valeur exacte} \end{array}$$

Méthode de la tangente (Euler $\simeq$ 1768).

$$\left. \begin{array}{l} t_0 \\ y_0 \end{array} \right\} \text{ connus} \Rightarrow \varphi'(t_0) = f(y_0, t_0) \text{ connue}$$

$$= \text{pente de la tangente à la courbe solution en } t = t_0$$

on peut construire tangente en t_0 et obtenir une valeur approchée y_1 de $\varphi(t_1)$ au point d'abscisse t_1 .

$$y_1 = y_0 + \varphi'(t_0)(t_1 - t_0) = y_0 + h \times f(y_0, t_0)$$

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.27

$$\left. \begin{array}{l} t_1 \\ y_1 \end{array} \right\} \text{ connus } \Rightarrow y'_1 = f(y_1, t_1) \text{ connue}$$

$$= \text{ pente d'une tangente approximative valable sur } (t_1, t_2)$$

d'où

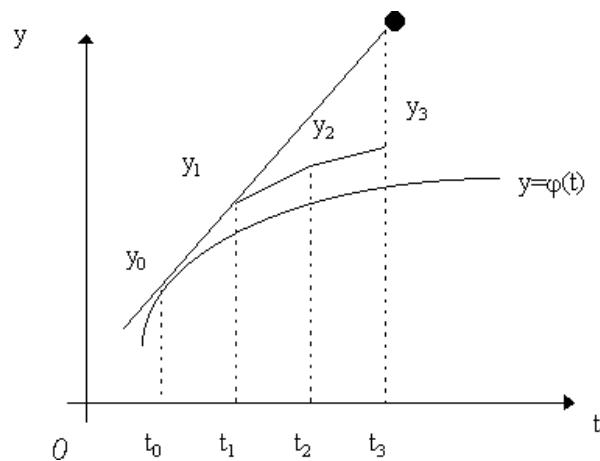
$$y_2 = y_1 + (y'_1)(t_2 - t_1) = y_1 + h \times f(y_1, t_1)$$

On poursuit le processus et la formule générale de l'approximation d'Euler s'écrit

$$y_{n+1} = y_n + h \times f(y_n, t_n) = y_n + h \times \frac{dy_n}{dt}$$

↓

Forme canonique d'une équation de niveau en SSD



TSSD01 Méthode d'intégration de la tangente.

NB.

Si l'on prend un “ pas de calcul ” trop grand, par exemple $h = t_3 - t_0$, on obtient très vite une solution numérique qui diverge fortement de la solution vraie. On notera que l'écart par rapport à la solution vraie se fait dans le sens de la concavité de la courbe analytique vraie.

2.1.1.3 Une amélioration de la méthode d'Euler. La méthode de Runge-Kutta.

Dans certains cas la méthode d'Euler n'est pas suffisamment précise.

Il existe une méthode relativement simple et aussi suffisamment précise pour être largement utilisable. Il s'agit de la méthode de Runge-Kutta*¹

$\updownarrow$
 Moyenne pondérée des valeurs de $f(t, y)$ prise en différents points
 de l'intervalle de travail $[t_n, t_{n+1}]$.

Une première formule (RK)

$$y_{n+1} = y_n + \frac{h}{6} \cdot [Q_{n1} + 4Q_{n2} + Q_{n3}]$$

où

$$\begin{aligned} Q_{n1} &= f(t_n, y_n) \\ Q_{n2} &= f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot Q_{n1}\right) \\ Q_{n3} &= f(t_n + h, y_n + h \cdot Q_{n1}) \end{aligned}$$

RK4.

$$y_{n+1} = y_n + \frac{h}{6} \cdot [K_{n1} + 2K_{n2} + 2K_{n3} + K_{n4}]$$

où

$$\begin{cases} K_{n1} = f(t_n, y_n) \\ K_{n2} = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot K_{n1}\right) \\ K_{n3} = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2} \cdot K_{n2}\right) \\ K_{n4} = f(t_n + h, y_n + h \cdot K_{n3}) \end{cases}$$

avec

K_{n1} = pente à gauche de l'intervalle
K_{n2} = pente au point médian $\rightarrow$ formule d'Euler pour aller de t_n à $t_n + h/2$
K_{n3} = seconde approximation de la pente au point médian
K_{n4} = pente en $t_n + h \rightarrow$ formule d'Euler + pente K_{n3} pour aller de t_n à $t_n + h$

Méthode RK = combinaison de simplicité et de précision.
 Une des plus largement utilisée pour la résolution numérique des
 équations différentielles.

¹* Carl David Runge (1856-1927) mathématicien et physicien allemand. Wilhelm Kutta (1867-1944) mathématicien allemand spécialiste d'aérodynamique.

2.1.2 La symbolique d'écriture des équations. Symbolique scripturale.

2.1.2.1 Les variables.

Sous Matlab

Noms de variable (ou de fonctions) peuvent être de n'importe quel longueur.

Doivent commencer par un caractère alphabétique,
ne peuvent contenir que des caractères alphabétiques,
des nombres ainsi que des "sous tiret" " _ "
Matlab ne se rappelle que des **19** premiers caractères.

Matlab est "case sensitive" il fait la distinction entre
majuscules et minuscules

A et a **ne sont pas** la même variable.

Tous les noms de fonctions sont en minuscule

$\text{inv}(A) \Rightarrow$ inverse de A

$\text{INV}(A) \Rightarrow$ fonction non définie

Exemples

Arrivées de travailleurs
ARRIVEES_DE_TRAVAILLEURS
AT

2.1.2.2 Les équations.

Si l'on traite de systèmes dynamiques simples on peut travailler directement sous Matlab. Pour des systèmes compliqués il vaut mieux utiliser la *toolbox* "SIMULINK".

On peut passer des écritures directement dans la fenêtre principale ou bien appeler des *M-files* qui sont des fichiers de programmes

Matlab est un langage d'**expression**, il interprète et évalue des expressions de la forme

$$\text{variable} = \text{expression}$$

Une expression est formée d'opérateurs, de caractères spéciaux, de fonctions et de noms de variables.

▲ Matlab travaille essentiellement avec **un seul type d'objet, une matrice rectangulaire numérique**. Dans certains cas on donne une signification spéciale aux matrices (1×1) qui sont des **scalaires** et aux matrices à une seule ligne ou une seule colonne qui sont des **vecteurs**.

Un énoncé se termine normalement par la touche **Enter**. Cependant si on termine l'énoncé par le caractère " ; " il n'y a pas d'affichage de réponse.

Exemple.

L'énoncé : » A=[1 2 3;4 5 6] donne

```
A =
  1 2 3
  4 5 6
```

Alors que l'énoncé : » A=[1 2 3;4 5 6];
donne la sortie "vide" : »

Nous avons vu qu'il existe des variables d'état (niveau) et des variables d'action (flux). Une **variable de niveau** est le résultat d'un énoncé comprenant une fonction permettant de résoudre des équations différentielles.

Sous Matlab 5.3 ces fonctions (*solver*) sont

ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb

Quel que soit le "solveur" la syntaxe de base est toujours la même

$$[T, Y] = \text{solver}('F', tspan, y0)$$

Attention on pourrait aussi écrire $[t, y] =$, mais prendre garde à l'ordre des arguments le t doit être mis **avant** le y (c'était l'inverse dans les anciennes versions de Matlab).

Les arguments de la fonction sont

→ ' F ' : c'est nécessairement une variable chaîne de caractères (*string variable*) c'est donc une chaîne entre "quotes".

C'est en fait le nom de la M-file qui permet de définir la ou les équations différentielles à intégrer. Cette définition est la première étape dans l'utilisation de n'importe quel solveur d'équations différentielles ordinaire (*ODE = Ordinary Differential Equations*) de Matlab. Dans la documentation de Matlab ce type de M-file est désigné sous le nom de " *odefile* ", mais on peut donner à la M-file n'importe quel nom désiré.

On peut utiliser la M-file pour définir un système d'équations différentielles sous la forme

$$y' = F(t, y)$$

où

t est une variable scalaire représentant le temps

y est un vecteur de variables dépendantes

F est une fonction de t et y retournant un vecteur colonne de la même longueur que y .

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.31

La M-file **doit** accepter les arguments t et y , même si elle n'a pas à les utiliser. Par défaut la M-file **doit** retourner un vecteur colonne de même longueur que y .

Exemple. On veut intégrer l'équation différentielle simple

$$\dot{x} \equiv \frac{dx}{dt} = 0.2 * x$$

On commence par appeler le MATLAB Editor/Debugger, faire File + New et choisir M-file. Dans l'éditeur on écrit les deux lignes

```
function xdot=expo(t,x)
xdot=0.2*x;
```

Attention à la syntaxe : " function " est une fonction Matlab qui permet de créer une nouvelle fonction dans le vocabulaire de Matlab (elle apparaît en bleu dans l'éditeur). Ici la nouvelle fonction s'appelle expo.

La deuxième ligne n'est autre que l'écriture de notre équation différentielle. On notera que la ligne finit par " ; ".

Dans l'éditeur on fait File + Save As et on note que le nom du fichier proposé par défaut est expo. Le fichier est sauvé dans le dossier désiré pourvu qu'on l'ait choisit en début de session à l'aide du *Path Browser*. On en a fini avec 'F' qui s'appelle ici 'expo'.

→ $tspan$: c'est un vecteur spécifiant l'intervalle d'intégration $[t_0 \ t_{final}]$.

→ y_0 : c'est un vecteur de conditions initiales.

Dans notre exemple on suppose que l'on veut intégrer sur l'intervalle $[0 \ 10]$ est que la valeur initiale de x est 1.

Sous Matlab on écrit

```
» tspan=[0 10]; + Enter
» [t,y] = ode23('expo',tspan,1); + Enter
```

Bien évidemment rien ne s'affiche puisque la deuxième expression se termine par ";". Si on veut vérifier ce qui s'est passé on peut écrire : » [t,y] , qui veut dire je demande la valeur de la matrice constituée de la colonne des t et de celle des y et Matlab nous répond

```
ans =
0 1.000000000000000
0.400000000000000 1.08328533333333
1.400000000000000 1.32305248711111
2.400000000000000 1.61588810425837
3.400000000000000 1.97353800466756
4.400000000000000 2.41034774970064
5.400000000000000 2.94383805163439
```

```

6.400000000000000 3.59540754039613
7.400000000000000 4.39119107600381
8.400000000000000 5.36310803415931
9.400000000000000 6.55014261238658
10.000000000000000 7.38520719375452

```

Noter que l'on n'a pas écrit `toto = [t,y]`, aussi Matlab crée automatiquement une variable `ans` (pour *answer*).

Remarques.

→ On a volontairement utiliser x dans la M-file `expo` et y dans l'expression de calcul sous Matlab. On aurait pu aussi écrire

```
[t,x]=ode23('expo',tspan,1);
```

cela marcherait encore.

→ Par contre ce qui marche moins bien c'est d'écrire

```
[y,t]=ode23('expo',tspan,1);
```

car alors y prend la place de t et inversement.

On peut visualiser graphiquement le résultat en écrivant les expressions

```

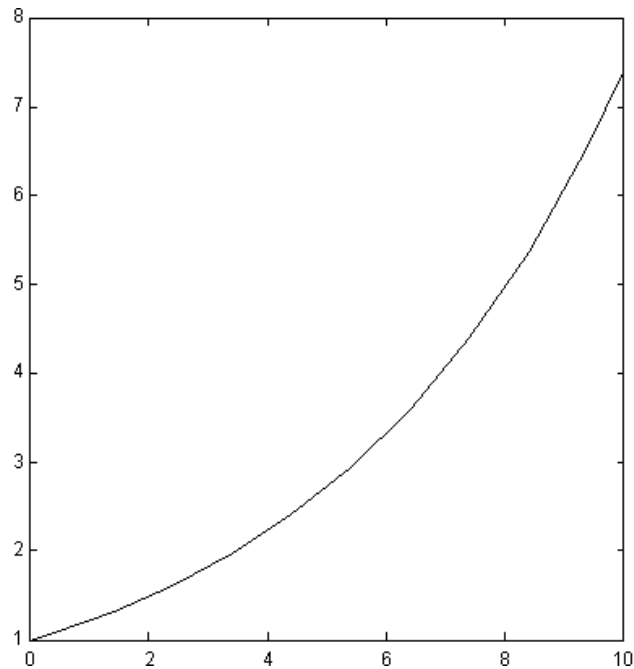
» plot(t,y)
» set(gcf,'color','w')

```

La première expression utilise la fonction "plot" qui sert à tracer le vecteur y en fonction du vecteur t . La deuxième expression permet d'obtenir une figure sur fond blanc, sinon on a par défaut un encadré grisé. (**expression à se rappeler** !)

On obtient le graphe suivant

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.33



TSSD02.

En ce qui concerne les **flux**, dans l'exemple antérieur ce n'est autre que la dérivée de la variable d'état soit

$$\dot{x} = 0.2 * x$$

Comme, par définition le taux de croissance de x est égal à $\dot{x}/x$ on a

$$\frac{\dot{x}}{x} = 0.2$$

c'est à dire que l'on a décrit l'évolution d'un phénomène croissant à taux constant.

Un **équation de flux** peut être très simple mais elle peut aussi être plus compliquée. De façon générale on peut dire que l'on a

$$Flux = f(Niveaux, constantes)$$

C'est pourquoi on peut être amené à décomposer une équation de flux en plusieurs morceaux en utilisant des **variables auxiliaires** et donc des **équations auxiliaires**.

Equation auxiliaire = subdivision algébrique d'une équation de flux ayant un sens précis et un rôle propre.

Exemple. Prenons un exemple simple en forçant le trait. On a un stock de marchandises x (niveau). On désire alimenter ce stock par un flux de commande décrit par une politique d'écart sur stock. On crée une variable auxiliaire

$$aux = sd - x \quad sd = \text{stock désiré}$$

et on a le flux de commande qui est égal à

$$flux = \frac{aux}{del}$$

ou del représente un délai d'ajustement.

Le programme correspondant sous Matlab est décrit par la M-file donnée ci-après

```
function xdot=RNSPb(t,x)
sd=50; del=5;
aux=(sd-x);
flux=aux/del;
xdot=flux;
```

On notera que l'on doit donner des valeurs aux deux paramètres (constantes) sd et del . Nous disons que nous avons forcé la trait car on aurait pu tout aussi bien écrire la M-file

```
function xdot=RNSP(t,x)
xdot=(50-x)/5;
```

On doit obtenir les mêmes résultats dans les deux cas. De fait sous Matlab on écrit

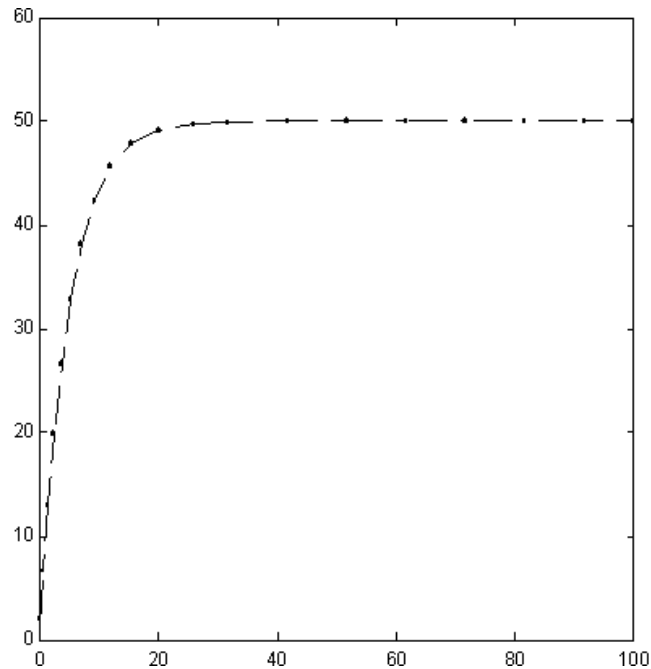
```
>> tspan=[0 100];
>> [t,y] =ode23('RNSPb',tspan,2);
>> [t,x] =ode23('RNSP',tspan,2);
>> plot(t,y,'.',t,x,'- -')
```

Commentaires.

- On suppose que la valeur initiale du niveau est égale à 2.
- La variable de niveau issue de la M-file RNSPb est stockée dans un vecteur y , celle issue de la M-file RNSP dans un vecteur x
- L'ordre plot veut dire que l'on trace y avec des points '.' et x avec des tirets'- '

On obtient le graphe qui montre bien la similitude des résultats.

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.35



TSSD03.

Pour le moment nous n'avons parlé que du **solveur** ode23 nous donnons ci-après les caractéristiques des différents solveurs selon la documentation technique de Matlab

Solveur	Type de problème	Ordre de précision	Quand l'utiliser
ode45	non difficile	moyen	La plupart du temps premier solveur à utiliser
ode23	non difficile	faible	Si erreur de tolérance grossière pour résoudre des pb modérément difficiles
ode113	non difficile	faible à élevé	Si erreur de tolérance contraignante pour résoudre des pb entraînant des calculs intensifs
ode15s	difficile	faible à moyen	Si ode45 est lent
ode23s	difficile	faible	Si erreur de tolérance grossière Pour résoudre des pb difficiles
ode23t	modérément difficile	faible	Si pb modérément difficile et si on veut une solution sans oscillations numériques
ode23tb	difficile	faible	Si erreur de tolérance grossière pour résoudre des pb difficiles

Pour ce qui est des algorithmes associés on sait que

→ **ode45** est fondé sur une formule explicite de Runge-Kutta (4,5). C'est un solveur à une étape, pour le calcul de $y(t_n)$ il ne nécessite que la solution à l'instant immédiatement précédant $y(t_{n-1})$. En général c'est la meilleure fonction à appliquer en tant que premier essai, pour la plupart des problèmes [1].

→ **ode23** est la mise en oeuvre d'une paire explicite de Runge-Kutta (2,3) selon Bogacki and Shampine. Il peut être plus efficace que ode45 pour une tolérance grossière et en présence de difficultés modérées. Comme ode45 c'est un solveur en une étape. [2]

→ **ode113** est un solveur d'ordre variable de Adams-Bashforth-Moulton. Il peut être plus efficace que ode45 pour des tolérances contraignantes et lorsque l'ODE-file est très coûteuse à évaluer. C'est un solveur multi-étapes, il nécessite normalement les solutions sur plusieurs instants précédents pour calculer la solution courante [3]

Les solveurs précédents sont censés résoudre les systèmes non difficiles. S'ils apparaissent comme étant trop lents, on peut utiliser l'un des solveurs ci-après.

→ **ode15s** est un solveur d'ordre variable fondé sur des formules de différenciation numérique (NDF). Il utilise de façon optionnelle des formules de différenciation en arrière (BDF), (connues aussi sous le nom de méthode de Gear), qui sont moins efficaces. C'est un solveur multi-étapes. Si on pense qu'un problème est difficile ou si ode45 a échoué ou a été très inefficace on peut essayer ode15s. [7]

→ **ode23s** est fondé sur les formules modifiées de Rosenbrock d'ordre 2. En tant que solveur à 1 étape il peut être plus efficace que ode15s pour des tolérances grossières. Il peut résoudre des problèmes difficiles pour lesquels ode15s n'est pas efficace. [7]

→ **ode23t** est la mise en oeuvre de la règle du trapèze. On utilise ce solveur si le problème est modérément difficile et si l'on veut une solution sans oscillations numériques.

→ **ode23tb** est la mise en oeuvre d'une formule implicite de Runge-Kutta (TR-BDF2) dont la première étape est une règle du trapèze et la seconde une formule de différenciation en arrière d'ordre 2. Comme ode23s ce solveur peut être plus efficace que ode15s pour des tolérances grossières [8, 9]

Références

[1] Dormand, J. R. and P. J. Prince, "A family of embedded Runge-Kutta formulae," J. Comp. Appl. Math., Vol. 6, 1980, pp 19-26.

[2] Bogacki, P. and L. F. Shampine, "A 3(2) pair of Runge-Kutta formulas", Appl. Math. Letters, Vol. 2, 1989, pp 1-9.

[3] Shampine, L. F. and M. K. Gordon, Computer Solution of Ordinary Differential Equations: the Initial Value Problem, W. H. Freeman, San

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.37

Francisco, 1975.

[4] Forsythe, G. , M. Malcolm, and C. Moler, Computer Methods for Mathematical Computations, Prentice-Hall, New Jersey, 1977.

[5] Shampine, L. F. , Numerical Solution of Ordinary Differential Equations, Chapman & Hall, New York, 1994.

[6] Kahaner, D. , C. Moler, and S. Nash, Numerical Methods and Software, Prentice-Hall, New Jersey, 1989.

[7] Shampine, L. F. and M. W. Reichelt, "The MATLAB ODE Suite," (to appear in SIAM Journal on Scientific Computing, Vol. 18-1, 1997).

[8] Shampine, L. F. and M. E. Hosea, "Analysis and Implementation of TR-BDF2," Applied Numerical Mathematics 20, 1996.

[9] Bank, R. E., W. C. Coughran, Jr., W. Fichtner, E. Grosse, D. Rose, and R. Smith, "Transient Simulation of Silicon Devices and Circuits," IEEE Trans. CAD, 4 (1985), pp 436-451

2.1.3 Problème du pas de calcul.

On peut appeler un solveur avec la syntaxe

$$[T, Y] = \text{solver}('F', tspan, y0, options)$$

Les options sont un argument optionnel d'intégration créé en utilisant la fonction `odeset`. Cette fonction `odeset` permet d'ajuster les paramètres d'intégration des solveurs d'EDO. Elle obéit à la syntaxe

$$options = \text{odeset}('name1', value1, 'name2', value2, \dots)$$

ou 'name' est un nom de propriété.

Il suffit de taper `odeset` sous Matlab pour voir apparaître tous les noms de propriétés avec leurs valeurs possibles. Parmi les propriétés deux nous intéressent ici, elle concernent le pas de calcul

Propriété	Valeur	Description
MaxStep	Scalaire > 0	Limite supérieure de la grandeur du pas de calcul utilisé par le solveur.
InitialStep	Scalaire > 0	Valeur suggérée du pas de calcul initial. Le solveur l'essaye en premier. Si elle est trop forte il y a une erreur, le solveur utilise un pas plus petit.

En ce qui concerne le nom de la propriété on peut se contenter de taper la premier lettre qui identifie de façon unique le nom de propriété, à ceci près qu'il y a plusieurs propriétés ayant pour initial MS.

Reprenons la M-file 'expo'. On peut écrire

```
» options = odeset('l',1,'MaxStep',1)
```

Après avoir fait **Enter** Matlab affiche tous les noms de propriétés avec leur valeur, pour les propriétés auxquelles on n'a pas fixé de valeur il retourne une matrice vide `[]`.

Si l'on veut extraire des propriétés de la structure d'options que l'on a créée avec `odeset` on utilise la fonction `odeget` qui obéit à la syntaxe

```
o = odeget(options,'name')
```

Si l'on tape ensuite

```
» tspan = [0 10]
» [t,x] = ode23('expo',tspan,1,options)
```

on constate en observant le vecteur t que les calculs se font bien de 1 en 1. Si l'on prend `ode45` les calculs se font de 0.25 en 0.25 !

Si l'on prenait

```
InitialStep = 0.5  MaxStep = 0.5
```

Les calculs avec `ode23` se font de 0.5 en 0.5 et avec `ode45` de 0.125 en 0.125.

Si l'on prend

```
InitialStep = 2  MaxStep = 2
```

Les calculs avec `ode23` se font aux instants suivants

```
0
1.484...
2.968...
4.440...
t = 5.911...
7.382...
8.854...
10...
```

Alors que avec `ode45` ils se font de 0.5 en 0.5. Il semblerait donc que au bout d'un moment (pour une valeur trop forte du pas de calcul) le solveur reprenne la main.

Cela étant revenons sur le principe du calcul pas à pas selon la méthode élémentaire d'Euler. On a

$$N = N + DT * F$$

$$N = N_0$$

$$F = a * N$$

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.39

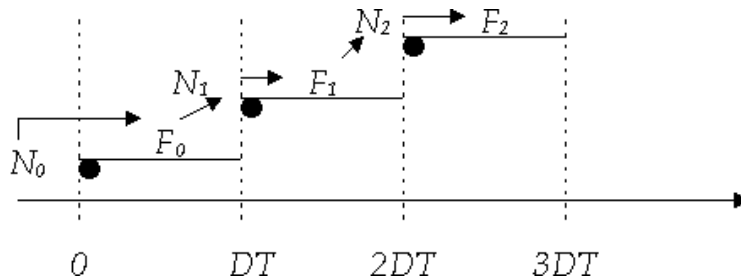
→ 1er pas de calcul $[0, DT]$

$$\begin{aligned} N_0 \text{ connu} &\rightarrow F_0 = a \cdot N_0 \\ &\Downarrow \\ N_1 &= N_0 + DT * F_0 \end{aligned}$$

→ 2ème pas de calcul $[DT, 2DT]$

$$\begin{aligned} N_1 \text{ connu} &\rightarrow F_1 = a \cdot N_1 \\ &\Downarrow \\ N_2 &= N_1 + DT * F_1 \end{aligned}$$

etc...



TSSD04 : Schéma du calcul pas à pas.

Le pas de calcul DT fait référence au processus de calcul lui même. Il obéit à une règle empirique qui peut s'énoncer comme suit

“ Le pas de calcul DT doit être compris entre $1/5$ et $1/2$ du délai de premier ordre le plus court du système modélisé. ”

Commentaires.

Notion de délai de premier ordre.

Soit la boucle de rétroaction la plus simple possible définie par

$$\begin{aligned} N &= N + DT * F \\ F &= NTC * N \\ NTC &= 0.10 \end{aligned}$$

C'est une boucle de rétroaction dite du "premier ordre" car dans cette boucle il n'entre en jeu qu'une seule variable d'état, le niveau N .

On a

$$\begin{aligned} NTC &= 0.10 = \frac{1}{10} = \frac{1}{10 \text{ "unités de temps"}} \\ &\equiv \frac{1}{\text{délai de premier ordre}} \end{aligned}$$

Si l'on travaillait à "taux constant", le niveau N doublerait en 10 ans, selon la formule

$$N = N_0(1 + g \times t) \quad g = \text{taux de croissance}$$

↓

C'est pourquoi le délai 10, est appelé " temps de doublement de N à taux constant ".

NB on aurait pu tout aussi bien écrire

$$\begin{aligned} N &= N + DT * F \\ F &= (1/DEL) * N \\ DEL &= 10 \end{aligned}$$

Que faire dans un système modélisé ?

→ D'abord rechercher tous les délais du premier ordre.

$$\begin{aligned} \text{Forme directe} &\rightarrow \equiv DEL \\ \text{Forme indirecte} &\rightarrow \equiv NTC \Rightarrow \text{inverser} \rightarrow D1O = \min(DEL) \end{aligned}$$

→ Appliquer la règle empirique qui est **impérative** !

$$(1/5) * D1O \rightarrow \text{sert à limiter le temps de calcul.}$$

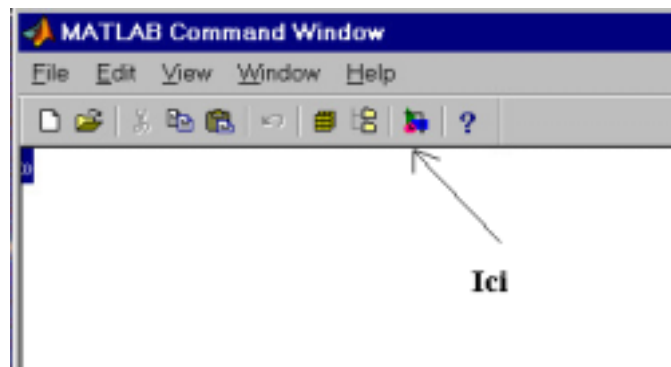
$$(1/2) * D1O \rightarrow \text{à ne pas franchir risques d'absurdités !}$$

2.1.4 Le passage à Simulink.

SIMULINK est une *Toolbox* de Matlab. C'est un logiciel qui permet de modéliser, simuler et analyser les systèmes dynamiques. Il prend en compte les systèmes linéaires et non linéaires, qu'ils soient modélisés en temps continu, en temps discret ou en temps mixte.

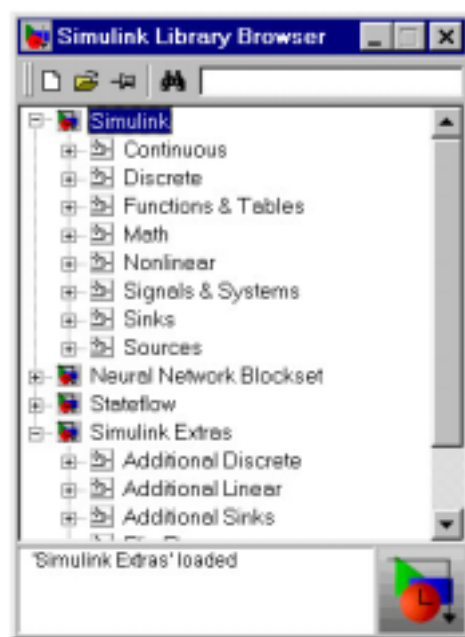
Simulink repose sur la notion de **block-diagram**, ces blocs sont copiés à partir de bibliothèques afin de former un schéma de système dynamique. Pour entrer dans Simulink, on se met d'abord sous Matlab et on clique sur l'icône **Simulink Library Browser**. Ou bien on tape `simulink` dans la fenêtre de commande de Matlab

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.41



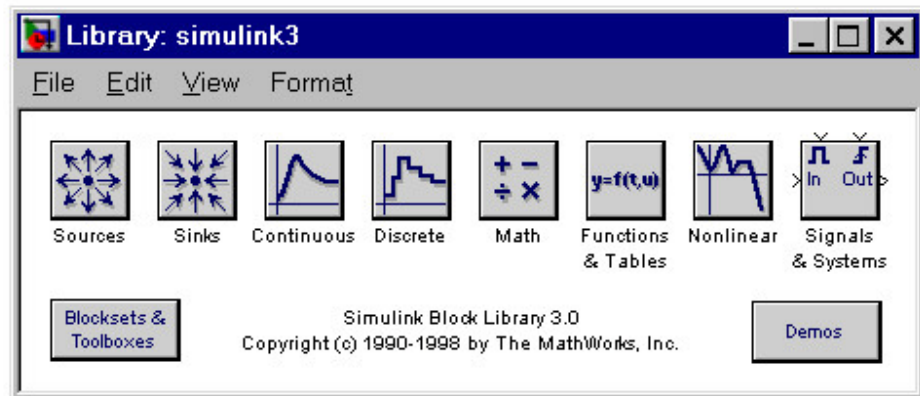
Smlk001

On fait alors apparaître l'écran suivant



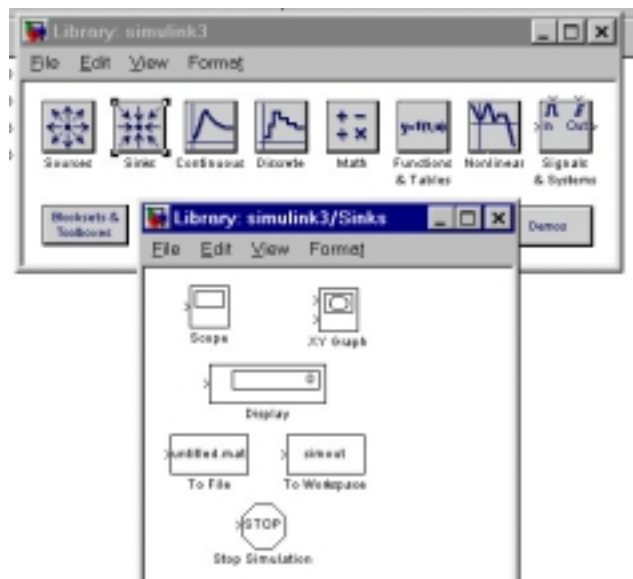
Smlk002

Ici nous avons en fait "développé" les bibliothèques '*Simulink*' et '*Simulink Extras*'. On pourrait aussi taper `simulink3` dans la fenêtre de commande et on ferait apparaître la fenêtre de la librairie de blocs de Simulink qui permet de voir toutes les catégories dont on dispose



Smlk003

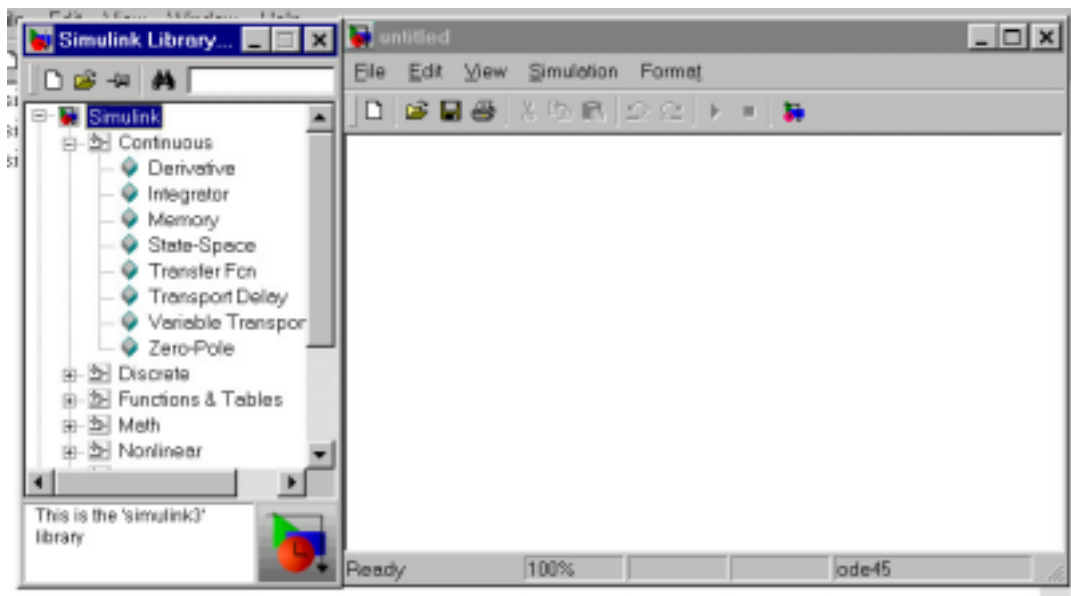
Un double clic gauche sur une des librairies permet de visualiser son contenu



Smlk004.

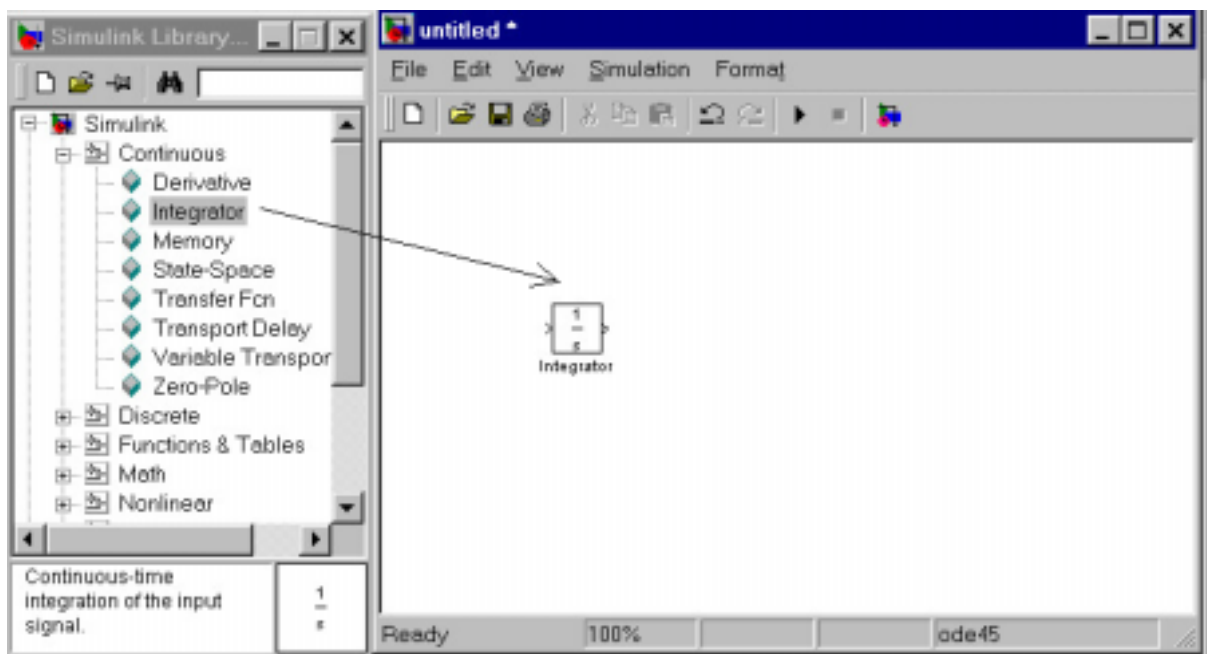
Pour créer un modèle on part du *Simulink Library Browser* (Cf. *Smlk002*). On clique sur le bouton **Nouveau Modèle** de la barre d'outil. On fait apparaître une nouvelle fenêtre. On a intérêt à déplacer cette nouvelle fenêtre vers la partie droite de l'écran de façon à voir en même temps son contenu et celui des librairies de blocs.

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.43



Smlk005.

Cela étant on va recréer le modèle RNSP. On note N notre variable de niveau (le stock), le coeur du modèle est constitué par le bloc Integrator [$\in$ *librairie* Continuous]



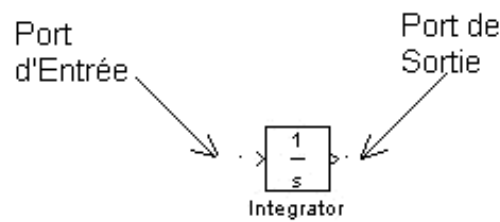
Smlk006.

Pour mettre ce bloc sur notre feuille blanche, on met le curseur de la souris sur Integrator dans la fenêtre gauche, on fait un clic gauche et en

maintenant le doigt appuyé on se déplace dans la fenêtre droite, on relâche le doigt le bloc apparaît.

Pour déplacer ce bloc sur la "feuille", il suffit de faire un clic gauche sur lui (4 petits carrés noirs apparaissent aux 4 coins) et de le déplacer. On le désélectionne en cliquant partout ailleurs sur la page vierge.

Ce bloc intégrateur possède un port d'entrée et un port de sortie



Smlk007.

Le port de sortie donne la variable de niveau (N), le port d'entrée accepte le flux (dN/dt). On sait que

$$\frac{dN}{dt} = \frac{SD - N}{Del}$$

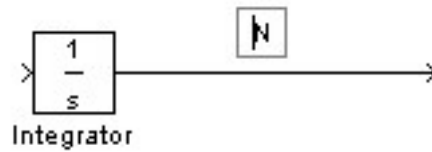
On commence par faire apparaître le "flux" de sortie. On met le curseur de la souris sur le port de sortie, **le curseur se transforme en croix (+)**. En tenant le bouton gauche enfoncé, on se déplace à droite, on relâche le doigt, une flèche est créée.



Smlk008.

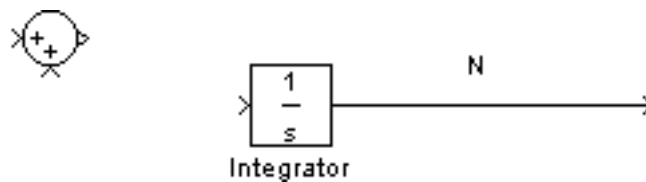
On peut mettre une annotation pour dire que la flèche de sortie représente N . Pour ce faire on fait un double clic un peu au dessus de la flèche dans une zone vierge. Une petit rectangle apparaît et le curseur se transforme en point d'insertion clignotant. On tape alors l'annotation, ici N . Pour terminer on clique sur la page vierge.

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.45



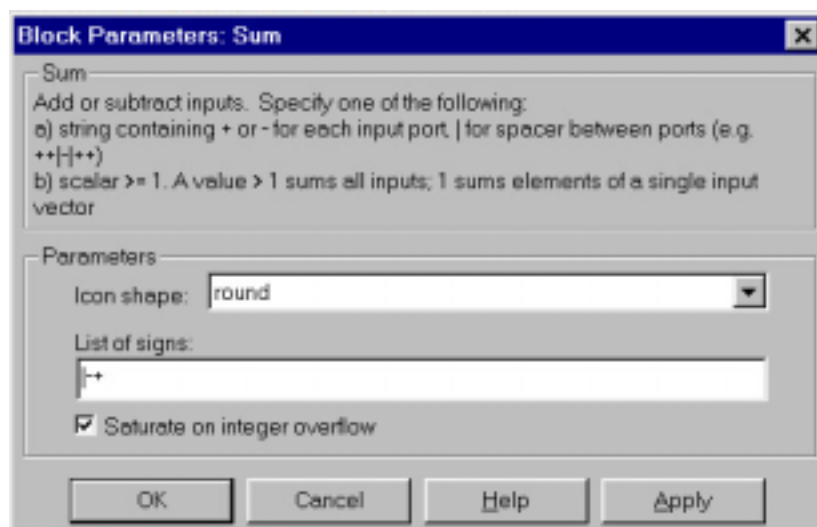
Smlk009.

On va maintenant utiliser cette valeur de N pour créer le flux de commandes. On commence par créer la différence $SD - N$. Pour ce faire on utilise un bloc Sum [$\in$ librairie Math]. On copie ce bloc sur notre feuille.



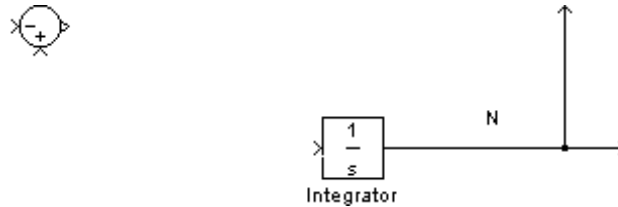
Smlk010.

Ce bloc doit comprendre une entrée positive (SD) et une négative (N). On fait un double clic gauche sur le bloc pour faire apparaître la fenêtre "Block Parameters: Sum" et on donne comme ligne des signes " $-+$ ".



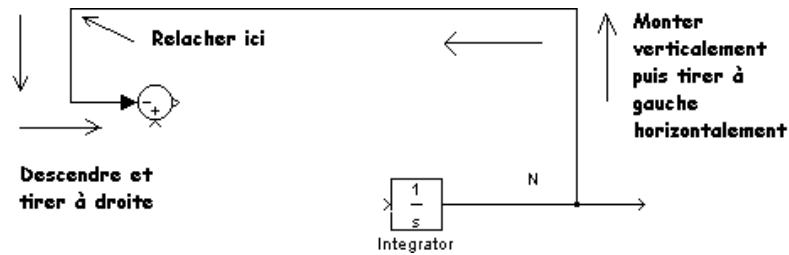
Smlk011.

Pour utiliser N il faut "dériver" le flux N existant (tracer une Branch Line). Pour ce faire on fait un clic droit sur la ligne existante, en laissant le doigt appuyé on tire une flèche, on relâche.



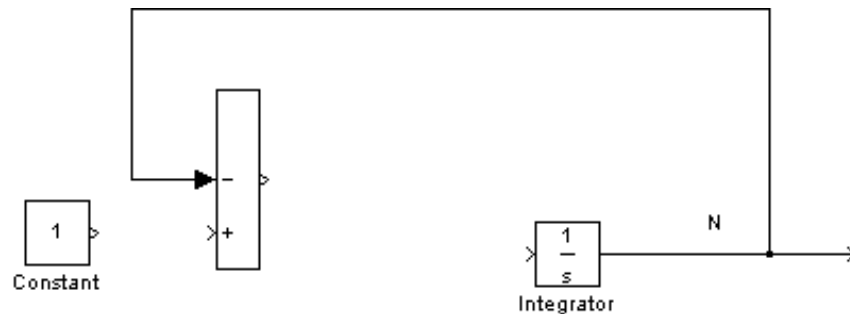
Smlk012.

Puis en se positionnant sur la flèche avec un clic gauche on déplace la ligne pour la faire entrer au port '-' du bloc Sum.



Smlk013.

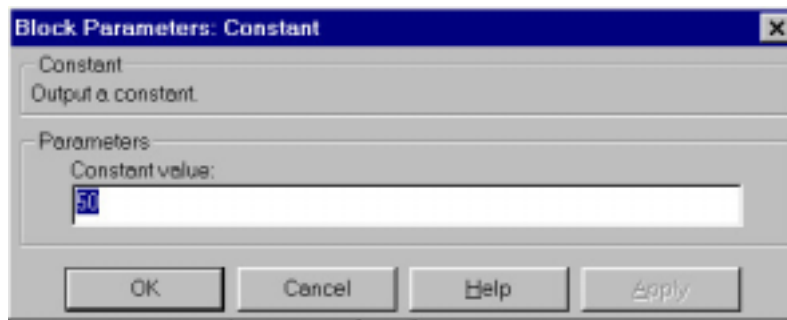
Le paramètre SD est une constante, on utilise le bloc Constant [$\in$ librairie Sources]. On copie ce bloc sur notre feuille. On en profite pour modifier la forme du bloc Sum. par un double clic gauche sur ce bloc on ouvre la fenêtre "Block Parameters: Sum" et dans "Icon Shape" on choisit "Rectangular".



Smlk014.

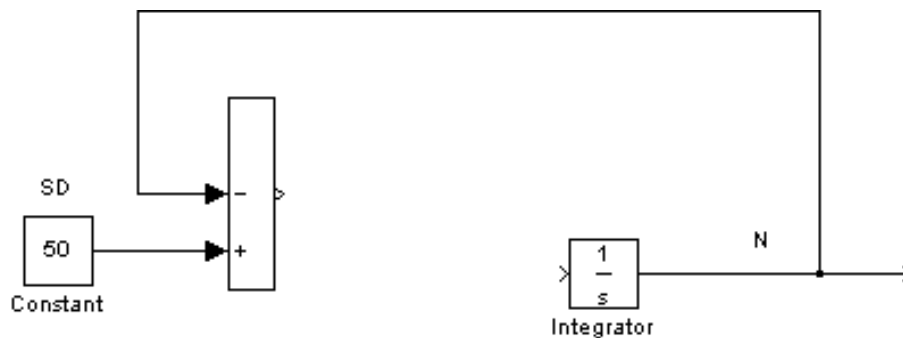
2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.47

Avec un double clic gauche sur le bloc Constant on fait apparaître la fenêtre "Block Parameters: Constant", dans Constant Value on porte la valeur 50.



Smlk015.

On met une annotation pour dire que ce bloc représente SD , enfin on tire une flèche du port de sortie du bloc Constant au port d'entrée (+) du bloc Sum.

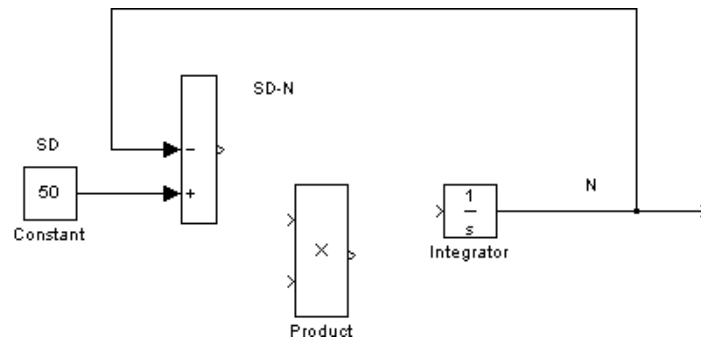


Smlk016.

Il nous faut maintenant traduire le flux

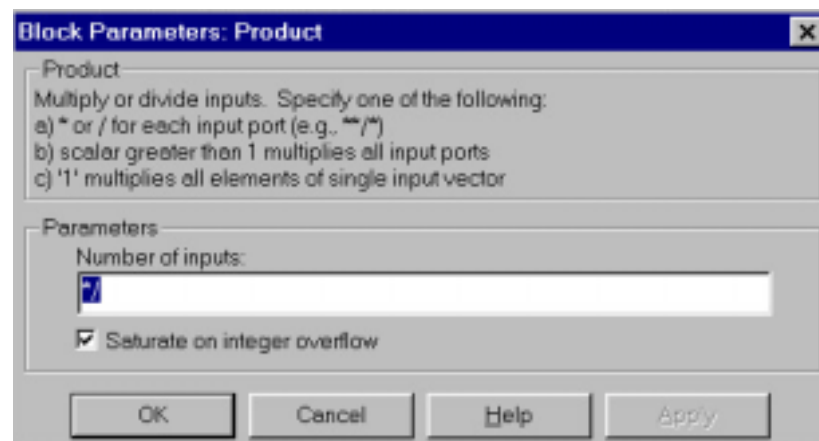
$$\frac{SD - N}{del}$$

On utilise le bloc Product [$\in$ librairie Math]. On copie ce bloc sur notre feuille, après avoir annoté le bloc Sum avec la notation $SD - N$.



Smlk017.

Tout comme le bloc Sum permet de faire des additions ou des soustractions, le bloc Product permet de faire des multiplications ou des divisions. On fait un double clic gauche sur ce bloc pour faire apparaître la fenêtre "Block Parameters: Product". Dans la fenêtre "Number of inputs" on fait apparaître la séquence (* /). Le "*" vaut pour $SD - N$ et le "/" pour del .

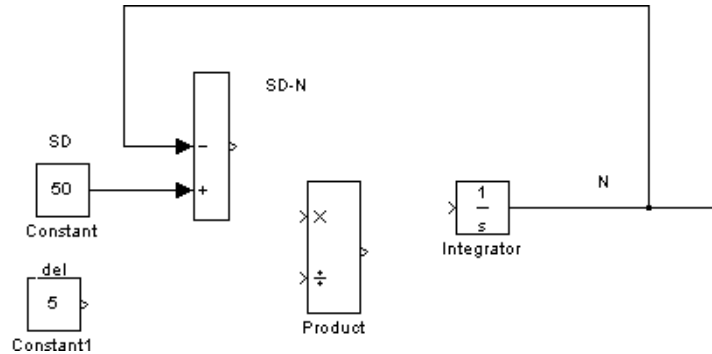


Smlk018.

Le paramètre *del* est une constante

- On copie un bloc Constant dans notre feuille.
- On lui affecte une valeur, en l'occurrence 5 (double clic gauche sur le bloc, etc...).
- On annote ce bloc avec *del*.

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.49



Smlk019.

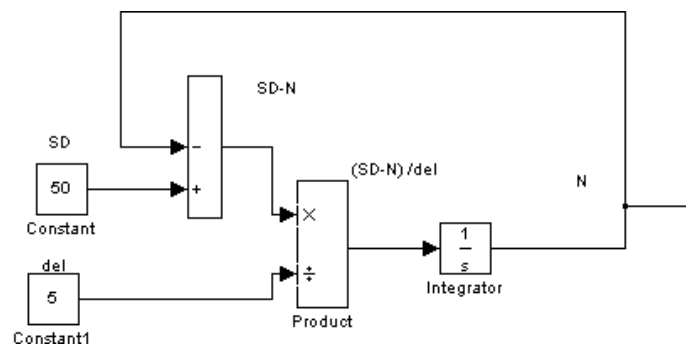
Pour en finir, ou presque

→ On joint le port de sortie du bloc $(SD - N)$ au port d'entrée ($\times$) du bloc Product.

→ On joint le port de sortie du bloc *del* au port d'entrée ($\div$) du bloc Product.

→ On annote le bloc Product avec $(SD-N)/del$.

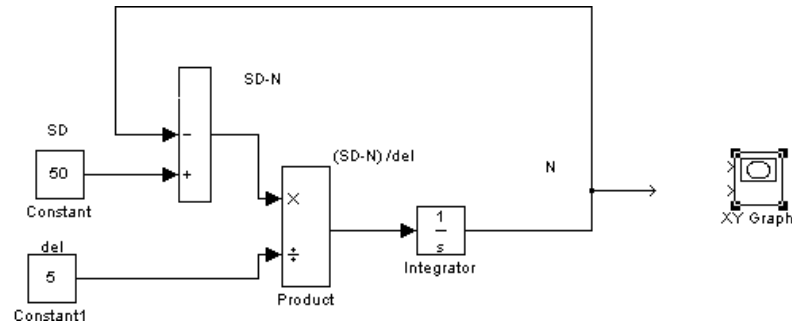
→ On joint le port de sortie de ce bloc au port d'entrée du bloc Integrator.



Smlk020.

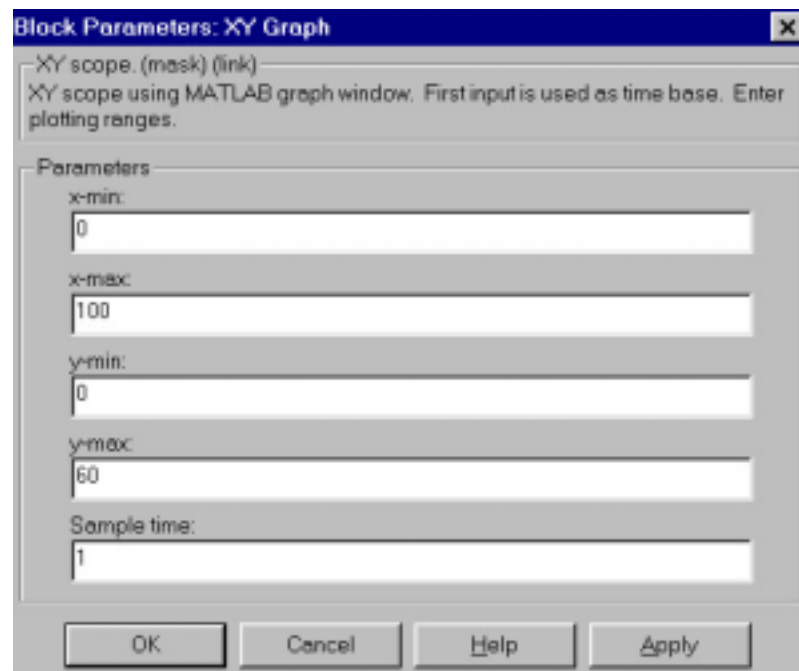
Avant d'en arriver à la partie simulation proprement dite il nous faut compléter le schéma. En effet il est nécessaire de connaître les résultats d'une façon ou d'une autre. Nous allons employer la manière la plus simple, une sortie graphique.

Dans la librairie Sinks on prend le bloc XY Graph, on le met à droite de la flèche "libre" de N .



Smlk021.

On note que ce bloc comprend deux ports d'entrée. On fait un double clic gauche sur le bloc. On voit que le premier port (l'abscisse x) est utilisé comme base de temps. On doit ensuite fixer les paramètres du tracé. On suppose que $x \in [0, 100]$ et que y (N) $\in [0, 60]$. Pour ce qui est de **Sample Time**, il s'agit de l'intervalle de temps entre les échantillonnages de valeurs. La valeur par défaut est -1 , ce qui veut dire que le temps d'échantillonnage est défini par le bloc conducteur (driving bloc).

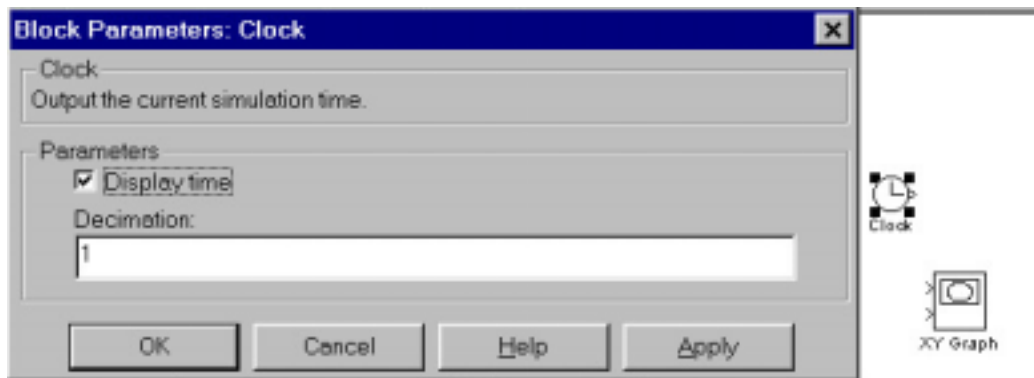


Smlk022.

Pour fixer le temps on va utiliser le bloc Clock [$\in$ librairie Sources]. On copie ce bloc dans notre feuille à côté du bloc XY Graph. On fait un double

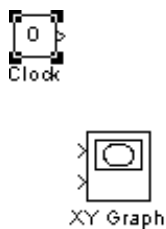
2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.51

clic gauche sur le bloc. Pour faire apparaître la fenêtre "Block Parameter : Clock". On coche la case Display time. Dans la fenêtre **Decimation** on porte la valeur 1. Le paramètre Decimation représente l'incrément auquel l'horloge se trouve mise à jour. Ce peut être n'importe quel entier positif. Si ce paramètre est à 1000, alors pour un pas d'intégration fixé à 1 milliseconde, l'horloge sera mise à jour à 1 seconde, 2 seconde, etc...



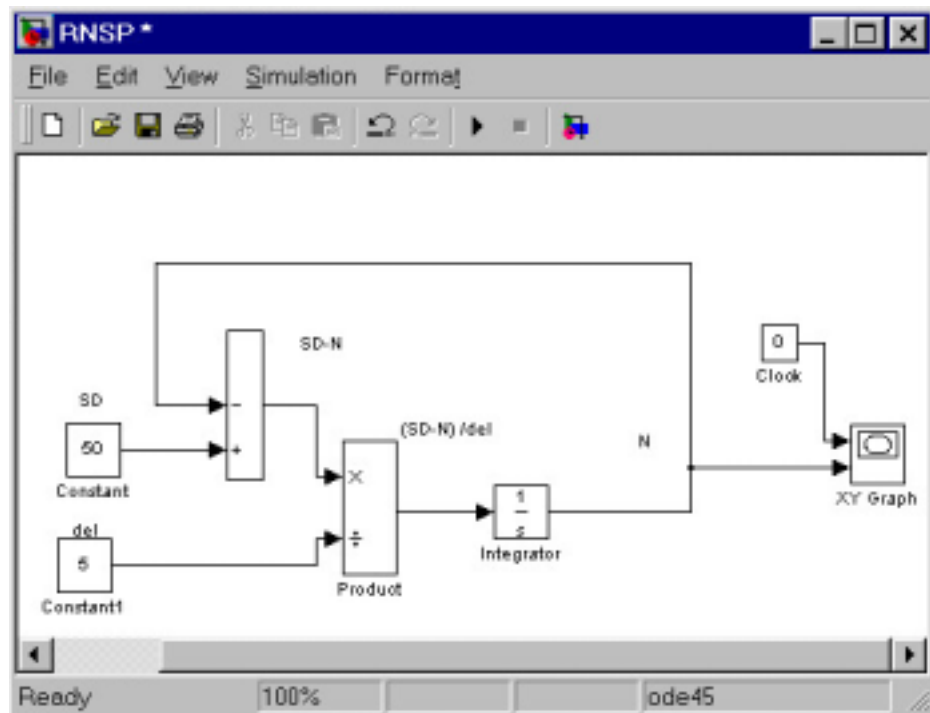
Smlk023.

Après avoir appuyer sur "OK" on revient à notre feuille. On note que l'horloge a changé de forme (passant du rond au carré) et qu'elle affiche le chiffre 0.



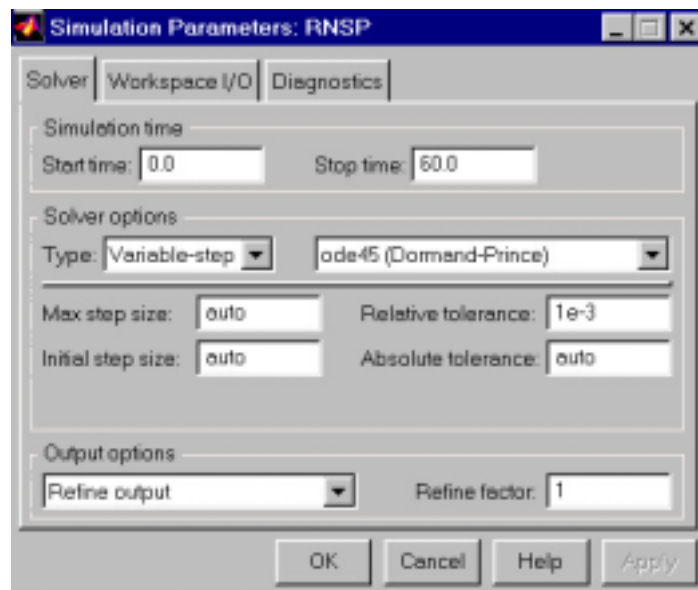
Smlk024.

On joint le port de sortie du bloc Clock au 1er port d'entrée du bloc XY Graph et N entre par le 2ème port d'entrée du bloc XY Graph. On obtient ainsi la fenêtre



Smlk025.

Pour lancer la simulation proprement dite on choisit le menu Simulation puis Parameters de façon à faire apparaître la boîte de dialogue des paramètres de simulation.



Smlk026.

Fiche technique de l'onglet Solver.

Cet onglet permet de

→ Fixer les instants de départ et d'arrêt de la simulation, ici 0 et 60.

→ Choisir le solveur et de préciser ses paramètres (par défaut le solveur est **Ode45**, comme on le voit dans la barre d'état située au bas de la fenêtre du modèle, Cf. *Smlk025*).

→ Sélectionner les options de sorties.

Le temps de simulation. C'est un temps spécifique au modèle, il n'a rien à voir avec l'horloge du micro-ordinateur. Dans la documentation de SIMULINK ce temps est exprimé en secondes, mais il peut représenter des heures, des mois ou des années selon le modèle.

Les types de solveurs. Comme nous l'avons déjà vu MATLAB et SIMULINK peuvent utiliser différents solveurs pour résoudre des systèmes d'EDO. Certains solveurs peuvent être plus efficaces que d'autres pour résoudre un type de problème particulier.

On peut choisir entre des solveurs à ***pas fixe*** et à ***pas variable***.

→ Les solveurs à pas variable peuvent modifier leur pas de calcul durant la simulation. Ils fournissent un contrôle d'erreur.

→ Les solveurs à pas fixe conservent le même pas de calcul durant toute la simulation et ne fournissent pas de contrôle d'erreur.

Les solveurs à pas variable sont

Modèles continus	ode45, ode23, ode113, ode15s ode23s, ode23t, ode23tb
Modèles discrets	discrete

Le solveur **discrete** est choisi par SIMULINK lorsqu'il détecte que le modèle élaboré ne possède pas d'états continus.

Les solveurs à pas fixe sont

ode5	version pas fixe de ode45
ode4	c'est RK4
ode3	version pas fixe de ode23
ode2	c'est la méthode d'Euler améliorée
ode1	c'est la méthode d'Euler
discrete	solveur sans intégration valable pour les modèles sans états.

Les options du solveur.

Les options par défaut fournissent des résultats précis et efficaces dans la plupart des problèmes.

Les pas de calcul.

Max step size et Initial step size sont automatiquement déterminés par défaut, comme l'indique la valeur auto. On peut modifier ces paramètres. Pour les solveurs à pas fixe on peut modifier la valeur de fixed step size, la valeur par défaut étant auto.

Max step size → Ce paramètre contrôle le plus grand pas de calcul que le solveur peut prendre. la valeur par défaut est déterminée à partir de Start time et Stop time.

$$h_{\max} = \frac{t_{\text{stop}} - t_{\text{start}}}{50}$$

En général la valeur par défaut est suffisante. *Si la durée de la simulation est trop grande, le pas par défaut peut être trop grand pour que le solveur trouve une solution. On peut en revenir à la règle sur le pas de calcul DT énoncée antérieurement.*

Si le modèle inclut des comportements périodiques ou quasi périodiques et si on connaît la période, on peut donner à Max step size une valeur fractionnaire de cette période (par ex. 1/4).

En général, si on désire plus de points de sortie, il faut changer le Refine factor (Cf. infra) plutôt que le pas maximal.

Initial step size → Le solveur sélectionne par défaut un pas de calcul initial en examinant les dérivées des variables d'état à l'origine. Le solveur essaye ce pas par défaut, mais il le réduit si des critères d'erreur ne sont pas satisfaits.

Tolérances d'erreur.

Les solveurs utilisent des techniques de contrôle d'erreur locale standards pour contrôler les erreurs à chaque pas de calcul. A chaque pas de calcul, les solveurs calculent la variable d'état à la fin du pas et déterminent l'**erreur locale**, c'est à dire l'erreur estimée de cette valeur d'état. Puis ils comparent l'erreur locale à l'**erreur acceptable**, qui est une fonction de la tolérance relative (*rtol*) et de la tolérance absolue (*atol*).

Si pour n'importe quel variable d'état l'erreur locale est supérieure à l'erreur acceptable, les solveurs réduisent la taille du pas et essayent à nouveau.

Relative tolerance → Mesure l'erreur relative à la taille de chaque variable d'état. Elle représente un pourcentage de cette valeur d'état. La valeur par défaut 1e-3 signifie que l'état calculé sera précis jusqu'à 0.1%.

Absolute tolerance → C'est une valeur d'erreur seuil. Elle représente l'erreur acceptable lorsque la valeur mesurée de la variable d'état tend vers zéro. L'erreur e_i sur la i ème variable d'état de valeur x_i doit satisfaire l'inégalité

$$e_i \leq \max [rtol \times |x_i|, atol_i]$$

2.1. FORMALISATION DES ÉQUATIONS, LE CALCUL PAS À PAS.55

Si on prend la valeur pas défaut (*auto*) SIMULINK met initialement la valeur de chaque variable d'état à $1e-6$. Au fur et à mesure que la simulation se déroule SIMULINK réévalue la tolérance absolue pour chaque variable d'état en la mettant à la valeur maximale, atteinte jusque là, par cette variable multipliée par la tolérance relative pour cette variable.

Par exemple si $rtol = 0.001$ si la variable d'état passe de 0 à 1, alors, à la fin de la simulation, on a $atol = 0.001$. Si la variable d'état passe de 0 à 1000, alors, à la fin de la simulation, on a $atol = 0.001 \times 1000 = 1$.

Output Options.

Les Options d'output de la boîte de dialogue permet de contrôler la quantité de sorties engendrées par la simulation. On a 3 options possibles (*popup options*).

- *Refine output.*
- *Produce additional output.*
- *Produce specified output only.*

Refine output → fournit des points de sortie additionnelle lorsque la sortie de simulation est trop grossière. Le *refine factor* représente un nombre entier de points de sortie entre deux pas de calcul. Un *refine factor* de 2 engendre un point de sortie à mi distance entre deux pas de calcul, ainsi qu'au pas de calcul.

Pour lisser l'extrait il vaut mieux agir sur le *refine factor* que sur la taille du pas de calcul. La modification du facteur n'entraîne pas de modification du pas de calcul.

Produce additional output → permet de spécifier directement les instants additionnels auxquels le solveur engendre des sorties. Si on choisit cette option SIMULINK affiche un champ Output Times. On peut mettre dans ce champ une expression Matlab ou un vecteur d'instant additionnels.

Produce specified output only → ce choix permet d'avoir des sorties **seulement** aux instants spécifiés. La spécification peut se faire sous forme de vecteur. Cette option modifie le pas de calcul de la simulation de sorte que les pas coïncident avec les instants spécifiés pour la production de l'extrait.

C'est un choix utile pour comparer différentes simulations en étant sûr que la simulation produit des extraits aux mêmes instants.

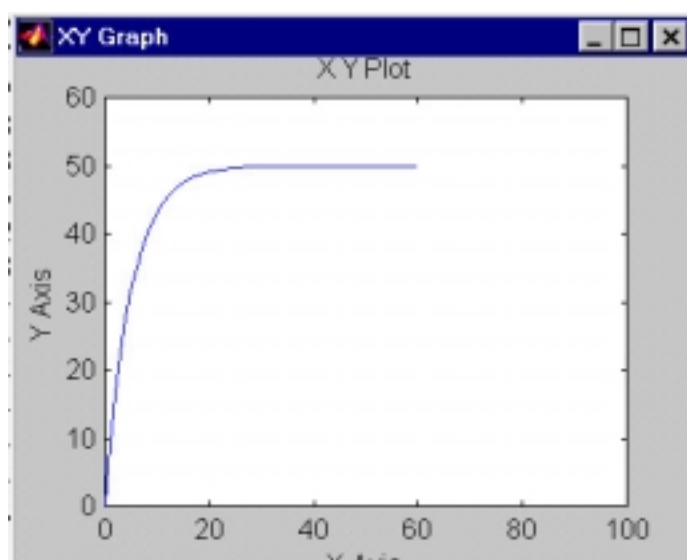
Exemple dans notre cas. On prend $t_{start} = 0$ et $t_{stop} = 5$. On démarre avec *Refine output* et un *refine factor* de 1. Dans l'espace de travail Matlab les instants de sortie sont fournies par un vecteur *tout*. Sous Matlab on fait $t1 = tout$ et on vérifie que *t1* est de format (57×1) il y a 15 instants variables entre 0 et 1 puis on va de 0.1 en 0.1 au delà de 1.

Si on passe le *refine factor* à 2 est si $t2 = tout$, *t2* est de format (113×1) , on a bien doublé les sorties entre 0 et 5.

Si on prend Produce additional output en spécifiant le vecteur $[0 : 5]$ qui veut dire j'ai les valeurs 0,1,2,3,4,5. On obtient des sorties à ces instants, mais aussi à des instants additionnels compte tenu du pas choisi par le solveur à pas variable. Si on fait $t3 = \text{tout}$, $t3$ est encore de format (113×1) .

Par contre si on prend Produce specified output only en conservant le vecteur $[0 : 5]$, si $t4 = \text{tout}$, alors $t4$ est de format (6×1) .

On prend les paramètres de simulation qui apparaissent dans la figure *Smlk026*. Après avoir fait OK pour fermer la boîte de dialogue.. On choisit le menu Simulation puis Start. On voit les chiffres défilier dans l'horloge jusqu'à 60, puis on voit apparaître le graphe suivant



Smlk027.

On retrouve bien le comportement observé antérieurement (Cf. *TSSD03*). On note en outre la cohérence avec les fixations de valeurs paramétriques, dans le bloc XY Graph $t(x)$ va de 0 à 100, pour la simulation t va de 0 à 60, d'où l'arrêt de la courbe à cet instant.

2.2 Boucles de rétroactions élémentaires.

Nous allons maintenant envisager un certain nombre de boucles de rétroaction élémentaires prises sous leur forme canonique. Nous en avons déjà vu certaines, nous en verrons d'autres. Nous envisagerons ces boucles d'un point de vue analytique et nous les simulerons en utilisant Simulink.

2.2.1 Boucle de rétroaction positive du premier ordre.

Elle correspond à l'équation différentielle

$$\frac{dY}{dt} = \dot{Y} = \frac{Y}{D} = aY$$

2.2.1.1 Etude analytique.

On a une équation différentielle du premier ordre homogène (sans second membre), à variables séparables

$$dY/Y = dt/D \Leftrightarrow \int dY/Y = \int dt/D \Leftrightarrow \ln Y = (t/D) + k$$

$$\Rightarrow Y = e^{(t/D)+k} = e^{t/D} \cdot e^k = C \cdot e^{t/D}$$

$$t = 0 \Rightarrow C = Y_0$$

$$\boxed{\boxed{Y(t) = Y_0 \cdot e^{t/D}}}$$

$D =$ constante de temps de l'exponentielle
délai de premier ordre $\subset$ boucle avec 1 niveau

$$t \rightarrow \infty \Rightarrow Y(t) \rightarrow \infty$$

Notons X le flux qui alimente Y

$$X(t) = \dot{Y}(t) = \left(\frac{Y_0}{D}\right) \cdot e^{t/D}$$

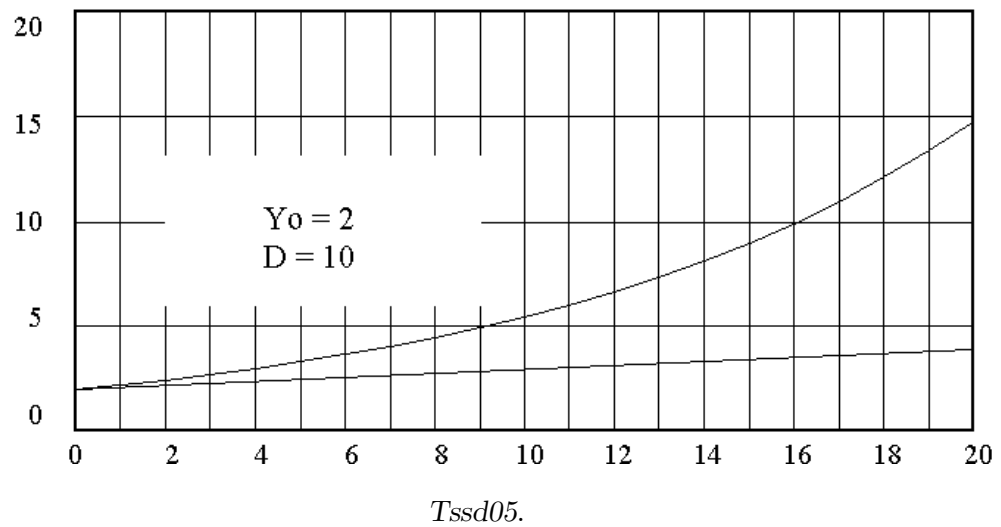
$X \equiv$ **vitesse** dans l'espace des quantités

Valeur de X = pente de la tangente en tout point de graphe de $Y(t)$

Equation de la tangente à l'origine

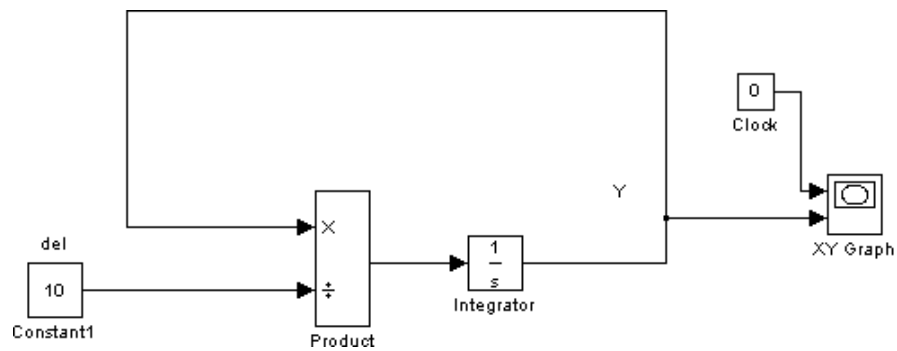
$$Y - Y_0 = \left(\frac{Y_0}{D}\right) \cdot (t - 0) \Rightarrow Y = Y_0 + \left(\frac{Y_0}{D}\right) \cdot t = Y_0 \cdot \left(1 + \frac{t}{D}\right)$$

Si l'on travaillait à **taux constant** (conservation du flux d'origine), le niveau Y doublerait de valeur dans un laps de temps égal à D . Ainsi D est le **temps de doublement de Y à taux constant**.



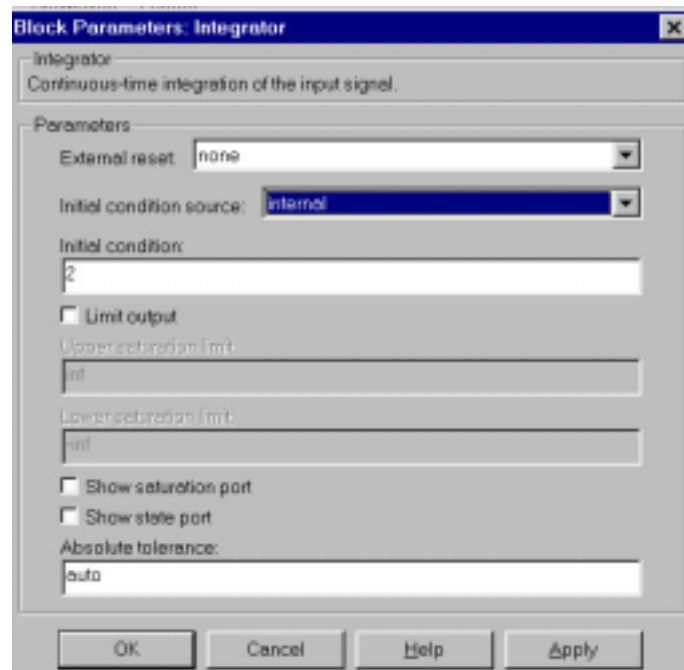
2.2.1.2 Simulation.

Le modèle s'appelle *BRP10*, sa structure est la suivante.



Smlk028.

On suppose que $del = 10$ et que $Y(0) = 2$. Pour initialiser la variable d'état on fait un double clic gauche sur le bloc integrator. On a la fenêtre suivante.

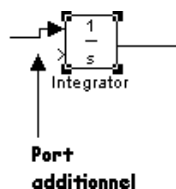


Smlk029.

On peut définir la condition initiale comme un paramètre interne au bloc ou comme provenant d'un signal externe.

Dans le premier cas on spécifie le paramètre Initial condition source comme internal. On met la valeur désirée dans le champ Initial condition (ici 2). Si on laisse le curseur de la souris (flèche) sur le bloc Integrator on voit apparaître en surimpression une fenêtre sur fond jaune donnant toutes les caractéristiques du bloc et notamment sa valeur initiale.

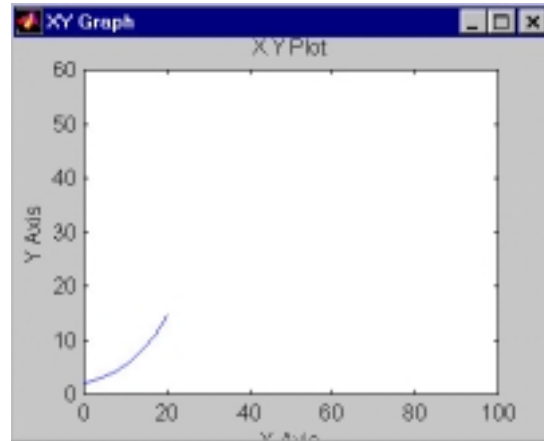
Si la valeur initiale dépend d'une source externe on spécifie le paramètre Initial condition source comme external. Un port d'entrée additionnel apparaît sur le bloc.



Smlk030.

Notons que pour établir notre modèle nous avons repris le modèle *RNSP* et supprimé les éléments indésirables. Dès lors les paramètres du bloc *XY*

Graph sont $t \in [0, 100]$ et $y \in [0, 60]$. Réalisons une première simulation en prenant $t_{start} = 0$ et $t_{stop} = 20$, pour essayer de retrouver le graphe donné dans *TSSD05*. On obtient le graphe

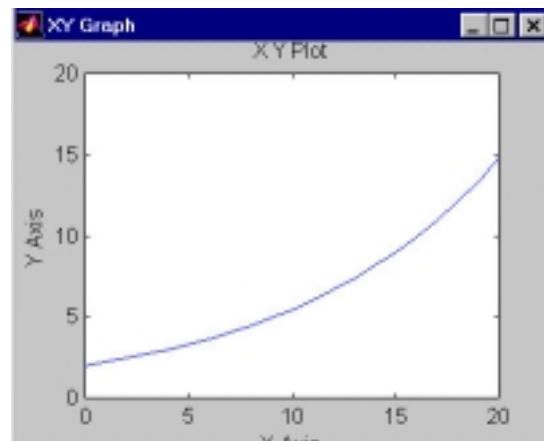


Smlk031.

Ce graphe ne correspond pas à nos attentes. Aussi il faut modifier les paramètres du bloc XY Graph pour les mettre en accord avec ceux de la simulation on prend donc

$$\begin{aligned} x\text{-min} &= 0 & x\text{-max} &= 20 \\ y\text{-min} &= 0 & y\text{-max} &= 20 \end{aligned}$$

d'où le nouveau graphe



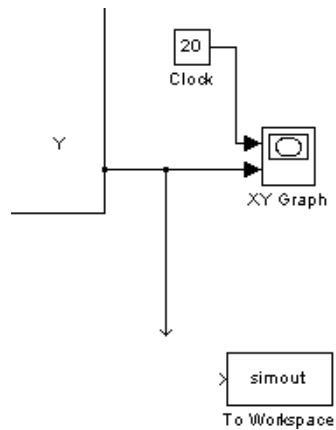
Smlk032.

On retrouve bien le comportement exponentiel d'une variable d'état croissant à taux constant. Cela étant ce type de graphe est utile pour se donner une première idée du comportement engendré.

Il possède un petit inconvénient, sa bordure grise (qui use de l'encre) et le fait qu'on ne peut le copier (ici nous faisons de la capture d'écran !).

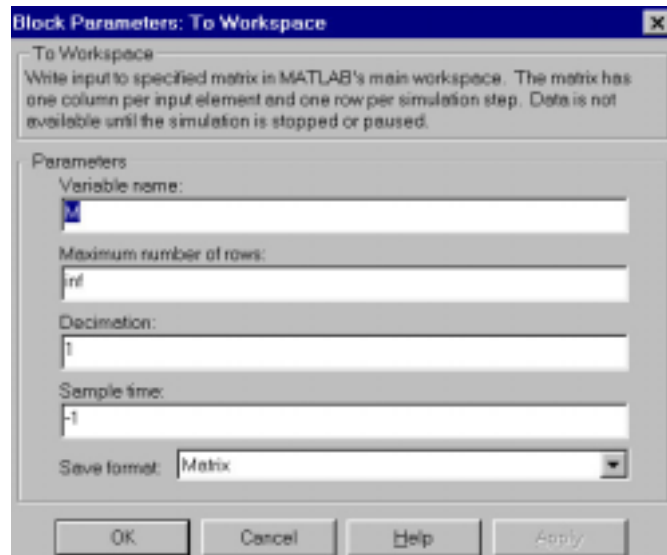
C'est pourquoi nous allons utiliser conjointement une autre solution de sortie des résultats permettant tout à la fois d'avoir des valeurs numériques et de faire un tracé graphique, le tout en ayant recours en partie à Matlab.

On commence par dériver le flux Y allant vers le bloc XY Graph. On fait une flèche vers le bas. Puis on copie le bloc To Workspace [$\in$ librairie Sinks]



Smlk033.

Par un double clic gauche sur ce bloc on fait apparaître la fenêtre de dialogue suivante



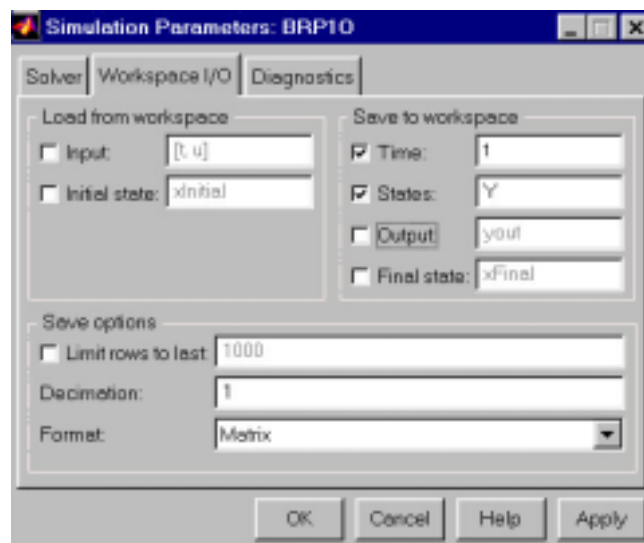
Smlk034.

On va remplir les champs de la façon suivante.

Pour Variable name il s'agit du nom de la matrice qui va contenir les données(par défaut *simout*), notons la M .

Passons au format de sauvetage, Save format. Il y a 3 possibilités, Structure, Structure with Time et Matrix. Nous utiliserons ce dernier format.

Si l'on fait un sauvetage dans le format Matrix on doit ajouter une colonne du temps de la simulation à la matrice. La solution la plus simple est la suivante. Revenons à la boîte de dialogue Simulation Parameters et prenons l'onglet **Workspace I/O**.



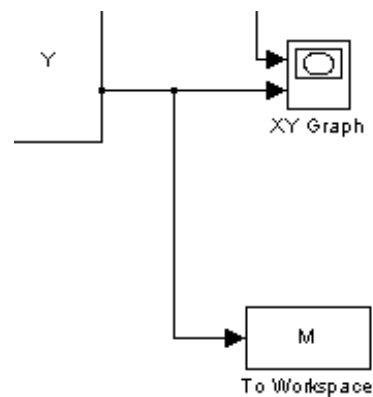
Smlk035.

On s'intéresse à la partie Save to Workspace. Par défaut deux cases sont cochées **Time** et **Output** avec les noms de variables *tout* et *yout*. Pour ce qui nous concerne nous cocherons **Time** en donnant le nom de variable t et **States** en donnant le nom de variable Y .

NB : Simulink sauve les extrants vers l'espace de travail (de Matlab) au taux d'échantillonnage de base du modèle. L'utilisation d'un bloc **To Workspace** permet le sauvetage à un taux d'échantillonnage différent.

Le facteur de décimation pris égal à 1 permet de sauver toute les valeurs. Un facteur de décimation de n donne un échantillonnage toutes les n valeurs.

Compte tenu de cette boîte de dialogue on sait que le temps est sauvé en tant que variable t . Dans notre modèle on relie tout simplement Y au port d'entrée du bloc **To Workspace** qui a pris le nom de M .



Smlk036.

On réalise notre simulation. On vérifie que le Workspace contient une matrice M (61×1) ainsi qu'un vecteur t de même format.

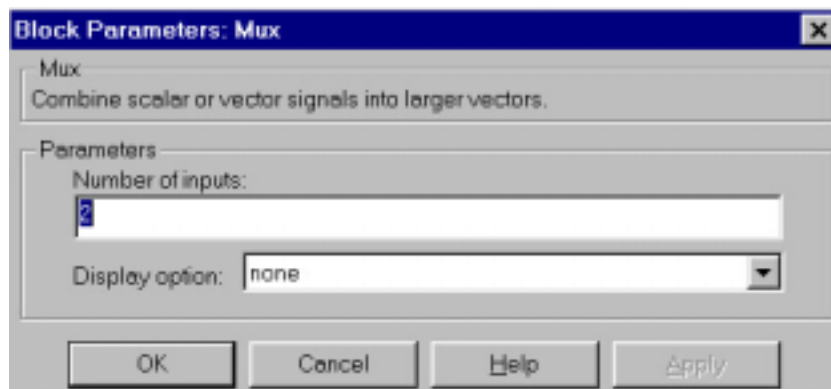
On écrit l'ordre

$$M1 = [t \ M]$$

on obtient une matrice de format (61×2)

Ici tout va bien parce que les paramètres de simulation et ceux du bloc To Workspace sont les mêmes notamment pour ce qui est de la Décimation. On peut avoir des problèmes au cas de différences entre paramètres.

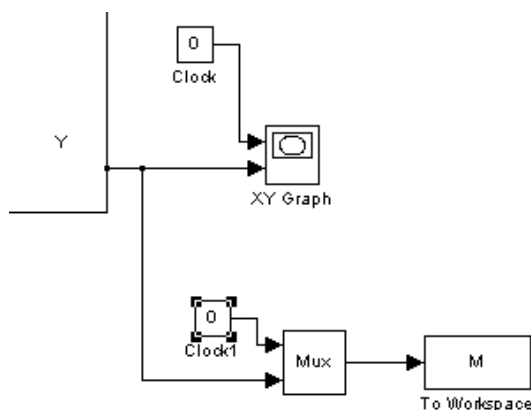
Aussi on peut employer une autre solution. On utilise un bloc multiplexeur, le bloc Mux [$\in$ librairie Signals & Systems]. On copie ce bloc, on fait un double clic gauche dessus et on prend les valeurs paramétriques suivantes



Smlk037.

On fixe le nombre de ports d'entrée à 2, et on prend comme option d'affichage none, ce qui fait que le bloc est affiché sous forme d'un rectangle, portant la mention Mux (pourvu qu'on l'agrandisse un peu).

La première entrée de ce bloc (port du haut) reçoit le signal d'un bloc Clock (sous forme de compteur) que l'on copie sur la feuille, la seconde entrée reçoit le flux Y . La sortie du bloc par le bloc To workspace. D'où le schéma partiel



Smlk038.

On peut alors régler les paramètres à partir du bloc To workspace. Notamment Decimation et Sample time.

→ **Sample time** : Ce paramètre permet de spécifier un intervalle de temps d'échantillonnage, des instants régulièrement espacés auxquels on collecte les données. C'est un paramètre utile lorsque l'on utilise un solveur à pas variable pour lequel les pas de calcul peuvent ne pas être les mêmes. La valeur par défaut -1 fait que le bloc To Workspace hérite du temps d'échantillonnage provenant du bloc conducteur lorsque l'on détermine l'écriture des données.

→ **Decimation** : le paramètre permet d'écrire des données à chaque $n^{\text{ème}}$ échantillon, n représentant le facteur de décimation. La valeur par défaut 1 engendre l'écriture des données à chaque pas de calcul.

Pour bien comprendre le jeu combiné de ces deux paramètres on va réaliser 4 simulations ayant les caractéristiques paramétriques suivantes

	Decimation	Sample time
S1	1	-1
S2	2	-1
S3	2	1
S4	2	2

A chaque simulation le bloc To Workspace envoie la matrice M dans l'espace de travail de Matlab. De façon à conserver les valeurs de ces matrices pour chaque simulation $n^{\circ}i$ on tapera l'ordre suivant dans la fenêtre de Matlab

$$M_i = M$$

Cela fait on utilise le **Workspace Browser** de Matlab pour connaître le format des différentes matrices ainsi formées. On pourrait aussi donner les ordres

`size(Mi)`

mais il faudrait passer 4 écritures alors que le **Workspace Browser** permet d'avoir une vision globale des formats des matrices dans l'espace de travail. Ces formats sont les suivants

S1	(61×2)
S2	(31×2)
S3	(11×2)
S4	(6×2)

Pour connaître les instants auxquels on a effectivement écrit les données on donnera les ordres

`ti = Mi(:,1)'`

ce qui se lit comme suit : " je veux la transposée (le ') du vecteur colonne constitué par toutes les lignes (le :) de la première colonne (le 1) de M_i ". On a ainsi

t1→

0	0.4	0.8	1.0	1.4	1.8	2.0	2.4	2.8	3.0
	3.4	3.8	4.0	4.4	4.8	5.0	5.4	5.8	6.0
	6.4	6.8	7.0	7.4	7.8	8.0	8.4	8.8	9.0
	9.4	9.8	10.0	10.4	10.8	11.0	11.4	11.8	12.0
	12.4	12.8	13.0	13.4	13.8	14.0	14.4	14.8	15.0
	15.4	15.8	16.0	16.4	16.8	17.0	17.4	17.8	18.0
	18.4	18.8	19.0	19.4	19.8	20.0			

On a là tous les instants auxquels le solveur travaille puisqu'on a pris *Sample Time* = -1 et que l'on a un facteur de décimation de 1. Si l'on passe le facteur de décimation à 2 on ne doit conserver que la "moitié" de ces instants. De fait on a

t2→

0	0.8	1.4	2.0	2.8	3.4	4.0	4.8	5.4	6.0
	6.8	7.4	8.0	8.8	9.4	10.0	10.8	11.4	12.0
	12.8	13.4	14.0	14.8	15.4	16.0	16.8	17.4	18.0
	18.8	19.4	20.0						

Il ne s'agit pas exactement d'une moitié puisque l'on passe de 61 points à

$$\frac{(61-1)}{2} + 1 = 31$$

Dans le cas suivant le temps d'échantillon est égal à l'unité et le facteur de décimation à 2 on doit donc aller de 2 en 2 unités de temps (de la simulation) de fait

$$t3 \rightarrow \quad 0 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16 \quad 18 \quad 20$$

enfin dans le dernier cas puisque le temps d'échantillon est égal à 2 et le facteur de décimation à 2 on doit aller de 4 en 4 unités de temps, de fait

$$t4 \rightarrow \quad 0 \quad 4 \quad 8 \quad 12 \quad 16 \quad 20$$

Au total tout dépend de 4 paramètres

$$\begin{array}{ll} \text{paramètres de simulation} & \left\{ \begin{array}{l} \text{Start Time} = t_{start} \\ \text{Stop Time} = t_{stop} \end{array} \right. \\ \text{paramètres du bloc TW} & \left\{ \begin{array}{l} \text{Sample Time} = ST \\ \text{Decimation} = D \end{array} \right. \end{array}$$

Dans la mesure où l'on démarre toujours à l'origine

$$t_{start} = 0$$

pour un temps d'arrêt donné

$$t_{stop} = T$$

si l'on prend un temps d'échantillonnage unité

$$ST = 1$$

on a un nombre de points égal à

$$\frac{t_{stop} - t_{start}}{ST} + 1 = \frac{T}{ST} + 1 = T + 1$$

et si l'on tient compte du paramètre de décimation ce nombre de points est égal à

$$\frac{T}{(ST \times D)} + 1 = \frac{T}{D} + 1$$

Le fait de choisir un Sample Time unité revient à faire comme si on avait un solveur à pas constant qui travaille à l'unité de temps, alors même que le solveur est à pas variable.

Le paramètre de décimation permet de diminuer le nombre de points de données lorsque l'on fait une simulation sur une longue période et que le phénomène évolue de façon relativement lissée. Si l'on a $T = 100$ on peut fort bien prendre un D égal à 2 ou 4 si l'évolution n'est pas trop heurtée.

A ce sujet on notera que le bloc To Workspace comporte un paramètre Maximum number of rows c'est le nombre de lignes de données à sauver dans la matrice. Si l'on a spécifié un maximum et si on engendre plus de lignes que ce nombre, on ne sauve que les lignes les plus récentes.

Par exemple si on fixe un maximum de 50 et si on en engendre 100, on ne conserve que les 50 dernières lignes (51 à 100). Pour prendre en compte toutes les données il suffit de garder la valeur par défaut inf (*infinity*).

Pour en revenir à notre boucle on a pris

$$\begin{aligned} \textit{Start Time} &= 0 \\ \textit{Stop Time} &= 20 \end{aligned}$$

prenons

$$\begin{aligned} \textit{Sample Time} &= 1 \\ \textit{Decimation} &= 1 \end{aligned}$$

et réalisons une simulation. N'oublions pas que dans la fenêtre Simulation parameters nous avons sauvé dans l'espace de travail le temps comme un vecteur t et la variable d'état comme un vecteur Y . On envoie par ailleurs dans cette espace de travail une matrice M . Si l'on observe les dimensions de ces 3 matrices (un vecteur est une matrice unicolonne) on a

$$\begin{aligned} t &: (61 \times 1) & Y &: (61 \times 1) \\ M &: (21 \times 1) \end{aligned}$$

La matrice M comprend les informations sur t et Y mais réduites à 21 observations au lieu de 61.

Cela étant arrivons en à ce qui nous intéresse le tracé d'un graphe, sans bordure grisé et copiable. On commence par extraire de l'information de M en écrivant

$$\begin{aligned} t1 &= M(:,1) \\ Y1 &= M(:,2) \end{aligned}$$

ce qui veut dire j'appelle $t1$ le vecteur constitué par la première colonne de M et $Y1$ celui constitué par la seconde colonne de M . Puis on réalise un tracé comparatif en écrivant

$$\begin{aligned} &\gg \text{plot}(t,Y,'- ',t1,Y1,'o') \\ &\gg \text{set(gcf,'color','w')} \end{aligned}$$

ce qui veut dire : "je trace les couples de points (t, Y) en tirets et les couples de points $(t1, Y1)$ avec un petit cercle (le o minuscule). Puis je passe le fond en blanc. On peut même changer les couleurs de tracé en écrivant

$$\text{plot}(t,Y,'b- ',t1,Y1,'ro')$$

la courbe en tirets sera tracée en bleu et cela avec des o en rouge.

On voit que le type d'un tracé est spécifié par les symboles entre quotes (' ') il existe une spécification de syntaxe de ligne (LineSpec sous Matlab). Un spécification peut comporter jusqu'à 3 spécifieurs

- Style de ligne (Line Style)
- Type de marquage (Marker Type)
- Couleur de ligne (Color)

Ces spécifieurs peuvent s'énoncer dans n'importe quel ordre. Leurs caractéristiques sont les suivantes

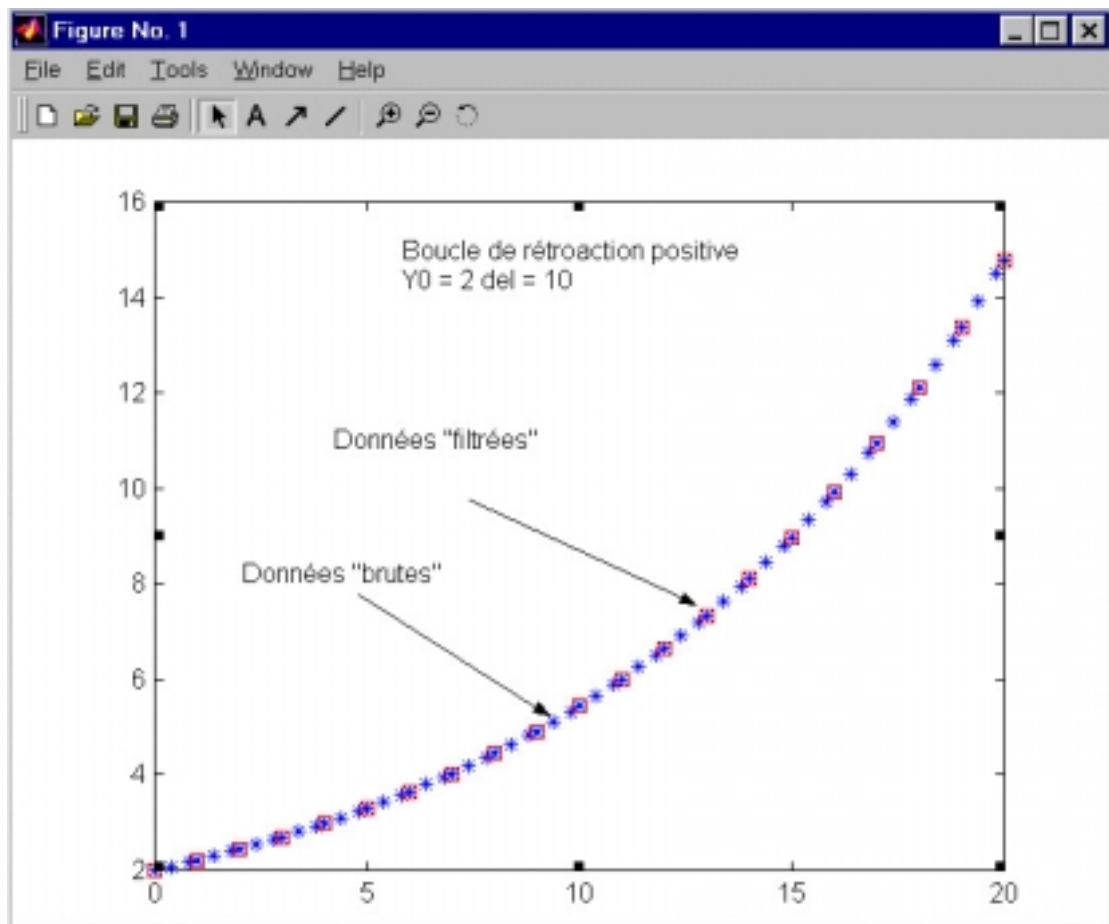
Specifieur	Line Style	Specifieur	Marker Type
-	solid line (default)	+	plus sign
- -	dashed line	o	circle
:	dotted line	*	asterisk
-.	dash-dot line	.	point
		x	cross
		s	square
		d	diamond
		^	upward pointing triangle
		v	downward pointing triangle
		>	right pointing triangle
		<	left pointing triangle
		p	five-pointed star (pentagram)
		h	six-pointed star (hexagram)

Specifieur	Color
r	red
g	green
b	blue
c	cyan
m	magenta
y	yellow
k	black
w	white

En ce qui nous concerne on peut écrire

```
plot(t,Y,'b*',t1,Y1,'rs')
```

les données brutes de la simulation sont tracées en bleu avec le symbole *, les données filtrées le sont en rouge avec un carré (□). On a ainsi le graphe



Smlk039.

On a quelque peu amélioré ce graphe en profitant des fonctionnalités offertes par la dernière version de Matlab (5.3 release 11). Les commentaires sont faits en utilisant l'icone (**A**), les flèches avec l'icone ($\nearrow$).

Si l'on désire tracer la tangente à l'origine de façon à avoir un graphe similaire à celui donné dans (Tssd05), graphe obtenu sous le logiciel Vensim, on rappelle que l'équation de cette tangente est

$$Y_0 \left(1 + \frac{t}{del} \right)$$

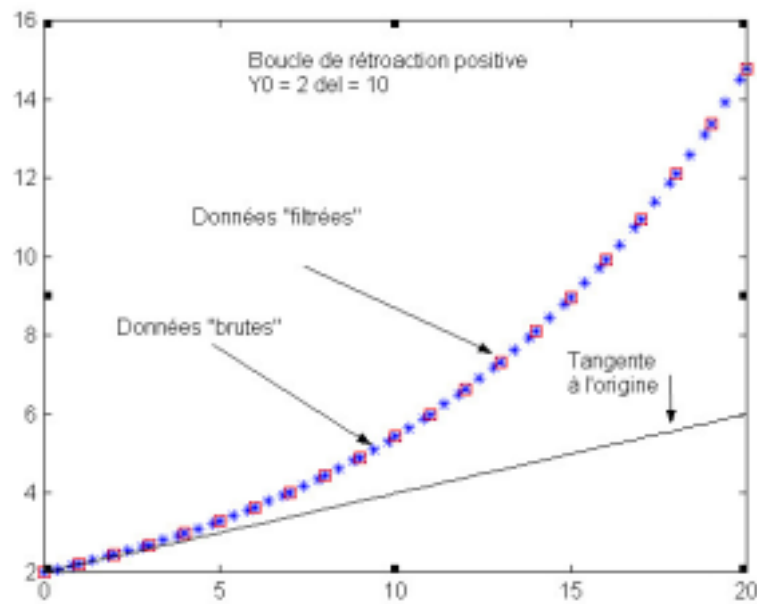
Dans le workspace de Matlab on crée un vecteur

$$\text{tangor} = 2 * (1 + t/10)$$

on écrit l'ordre

$$\text{plot}(t, Y, 'b*', t1, Y1, 'rs', t, \text{tangor}, 'k')$$

Pour ce faire inutile de tout retaper ! On remonte dans la fenêtre de Matlab à l'aide de la touche ($\uparrow$) jusqu'à retrouver l'expression `plot(t,Y,'b*',t1,Y1,'rs')` et on complète l'expression. On fait "Enter" et on a le graphe



Smlk040.

A noter que si l'on n'a pas fermé le graphe précédent le tracé de la tangente à l'origine s'y surajoute. On a simplement ajouté un commentaire et une flèche.

Autre chose, on note que par défaut l'axe des ordonnées démarre à 2. On peut le faire démarrer à l'origine. On règle l'échelle des axes à l'aide de la fonction `axis` de syntaxe

```
axis([xmin xmax ymin ymax])
```

Ici attention à ce que fait Matlab ! On suppose toujours que nous n'avons pas fermé la figure antérieure. On tape l'expression

```
axis([0 20 0 16])
```

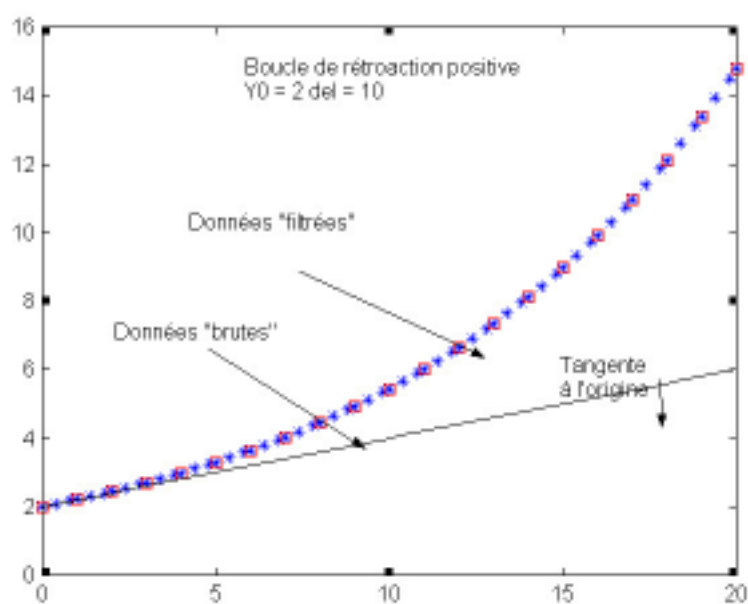
on fait Enter, oh surprise la tangente a disparu du graphe. On retape l'expression (utiliser la touche $\uparrow$ pour la retrouver)

```
plot(t,Y,'b*',t1,Y1,'rs',t,tangor,'k')
```

et on fait Enter la tangente est revenue mais les ordonnées sont redescendues à 0, on retape

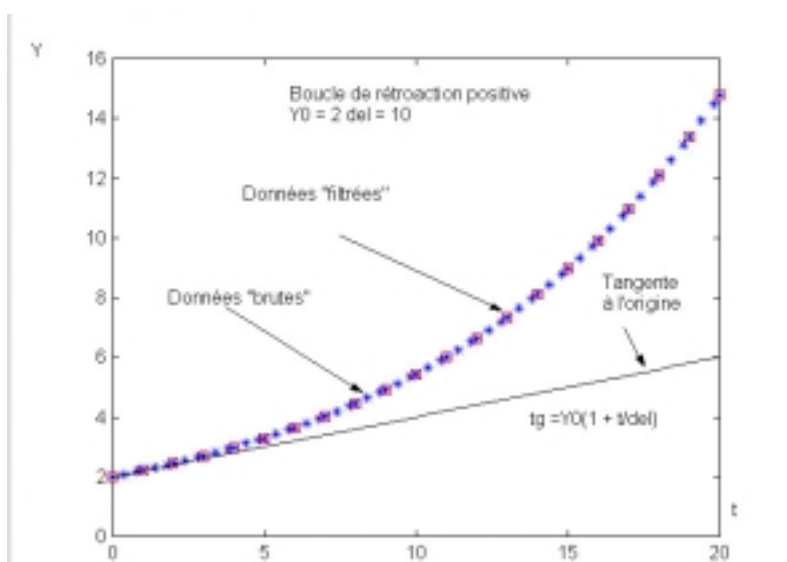
```
axis([0 20 0 16])
```

tout est à peu près dans l'ordre sauf que les annotations et les flèches ont subi une translation basse (ce sont des "images" surajoutées au graphe)



Smlk041.

Pour mettre les choses en ordre il suffit de déplacer les annotations et les flèches (clic gauche sur l'objet + déplacement à volonté). On obtient *in fine* le graphe ci-après



Smlk042.

2.2.2 Boucle de rétroaction négative du premier ordre à seuil nul.

Il s'agit en quelque sorte de l'opposée de la boucle antérieure elle correspond à l'équation différentielle

$$\dot{Y} = -\frac{Y}{D} = -aY$$

2.2.2.1 Etude analytique

Un processus d'intégration similaire au précédent conduit à la solution générale

$$\boxed{\boxed{Y(t) = Y_0 \cdot e^{-t/D}}}$$

On voit que

$$t \rightarrow +\infty \Rightarrow Y(t) \rightarrow 0^+$$

Si X est le flux alimentant Y (en fait un flux de sortie) on a

$$X(t) = \dot{Y}(t) = -\left(\frac{Y_0}{D}\right) \cdot e^{-t/D}$$

La pente de la tangente au graphe de Y est négative en tout point.

Pente est maximale à l'origine, $\dot{Y}(0) = -(Y_0/D)$

$$t \rightarrow +\infty \Rightarrow \dot{Y}(t) \rightarrow 0^-$$

Il y a décroissance à taux moins que proportionnel

Si taux constant $\Rightarrow$ on se déplace sur la tangente au graphe à l'origine.

$$\begin{aligned} y - y_0 &= -\left(\frac{y_0}{D}\right) \cdot (t - t_0) \Leftrightarrow y = y_0 \cdot \left(1 - \frac{t}{D}\right) \\ y &= 0 \Leftrightarrow t = D \end{aligned}$$

D = temps nécessaire pour qu'à taux constant
le niveau Y atteigne son seuil nul

2.2.2.2 Simulation

Pour simuler ce modèle on reprend le modèle antérieur. Comme le flux alimentant le bloc **Integrator** est négatif on insère un bloc **Gain** [$\in$ librairie Math] dans la ligne correspondante.

Ce type de bloc engendre un extrant en multipliant son intrant par une constante, une variable ou une expression.

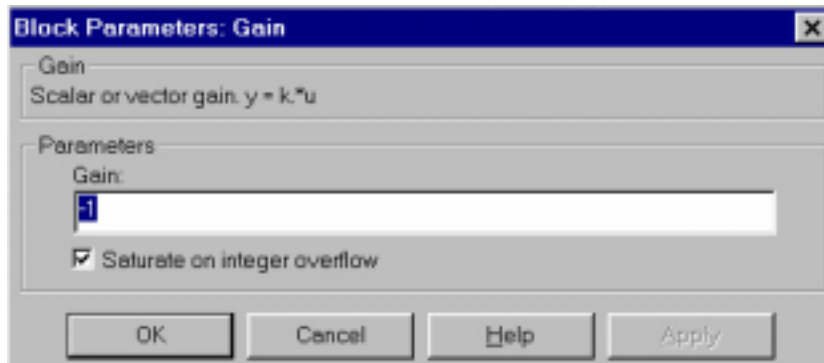
On copie le bloc sur le schéma. Par défaut les signaux entrent dans un bloc en allant de gauche à droite. Ici on a besoin d'aller de droite à gauche, il faut changer l'orientation du bloc. Pour ce faire on utilise le menu **Format** et les commandes **Flip Block** ou **Rotate Block**.

→ **Flip Block** (Ctrl + F) engendre une rotation du bloc de 180°.

→ **Rotate Block** (Ctrl + R) engendre une rotation du bloc de 90° dans le sens des aiguilles d'une montre.

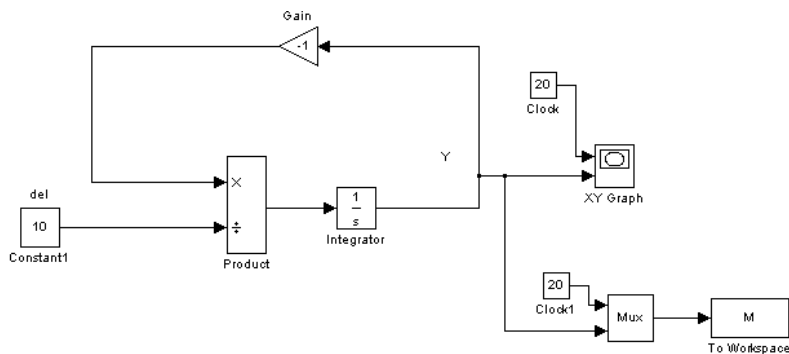
Ici on fait un clic gauche sur le bloc **Gain** (on le sélectionne, 4 petits carrés noirs apparaissent aux 4 coins) puis on fait Ctrl+F. Le bloc est bien orienté.

On fait un double clic gauche pour avoir la fenêtre de dialogue suivante



Smlk043.

On fixe la valeur du gain à -1, cette valeur est affichée dans le bloc. Il reste à déplacer le bloc en direction de la ligne supérieure de Y . Lorsque entrée et sortie correspondent avec cette ligne, elle se scinde en 2, il suffit de relâcher le doigt de la souris. On a alors le modèle BRNPOSN



Smlk044.

Pour la simulation proprement dite on initialise le bloc Integrator à 20. On effectue la simulation. **A supposer que l'on soit toujours dans la même session de travail** on doit effacer l'ancienne figure réaliser sous Matlab.

La matrice M s'étant modifiée on doit retaper les expressions

```
t1=M(:,1);
Y1=M(:,2);
```

sinon on conserverait les anciennes valeurs. Puis on tape les expressions

```
» plot(t1,Y1,'r-s')
» set(gcf,'color','w')
» axis([0 20 0 20])
```

On peut vouloir ajouter le graphe de la tangente à l'origine au graphe d'équation

$$Y_0 \cdot \left(1 - \frac{t}{D}\right)$$

on réécrit tangor sous la forme

```
tangor=20*(1 - t/10);
```

si l'on passe l'expression

```
plot(t,tangor,'-k')
```

et que l'on fait Enter on a la désagréable surprise de n'avoir plus qu'une droite sur la graphe, il faut reprendre les expressions comme suit

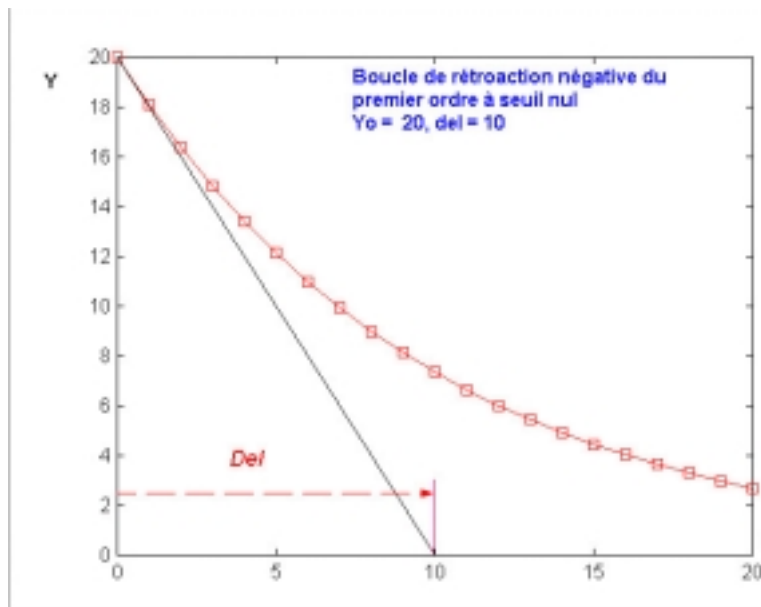
```
» plot(t1,Y1,'r-s',t,tangor,'-k')
» axis([0 20 0 20])
```

on a le graphe désiré. Il est inutile de répéter le `set(gcf,'color','w')` dans la mesure où le graphe n'a pas été fermé entre temps mais simplement rafraîchi.

On obtient le graphe amélioré donné ci-après. On notera que dans ce graphe on a tracé un trait plein en rouge et une flèche tiretée en rose. De fait les caractéristiques des lignes et des flèches peuvent être ajustée en faisant un double clic gauche sur l'objet ou un clic droit

Mais **pour ce faire il faut commencer** par cliquer sur l'icône ($\backslash$) qui permet d'accéder au mode d'édition de tracé (*Plot edit mode*).

On peut aussi modifier les textes (clic gauche + clic droit $\Rightarrow$ *pop up menu*). C'est Properties dans le pop up menu qui permet d'accéder aux fontes de caractères. Pour plus de précisions voir **Matlab, Release 11 New Features** p. 1-25 sq.



Smlk045.

Analogie physique

L'assemblage étudié traduit le comportement du niveau du liquide dans une chasse d'eau que l'on vide.

La vitesse d'écoulement du liquide est fonction de la hauteur d'eau dans le réservoir (moins il y a de liquide, moins la vitesse est forte).

2.2.3 Boucle de rétroaction négative du premier ordre à seuil positif.

Ce n'est autre que le modèle RNSP déjà vu. Du point de vue analogie physique c'est le cas de la chasse d'eau qui se remplit.

2.2.3.1 Etude analytique

Le modèle correspond à l'équation différentielle

$$\dot{Y} = \frac{S - Y}{D} = a(S - Y)$$

où S désigne un seuil que l'on veut atteindre et D un délai de temps. C'est une équation différentielle du premier ordre avec second membre

$$\dot{Y} + \frac{Y}{D} = \frac{S}{D}$$

Recherche d'une solution générale en deux temps.

1 Tout d'abord on intègre l'équation sans second membre

$$\dot{Y} + \frac{Y}{D} = 0 \Rightarrow Y = C \cdot e^{-t/D} \quad (\text{E1})$$

2 Puis on applique la **méthode de la variation de la constante**

$$C \equiv C(t)$$

D'après (E1)

$$Y' = C' \cdot e^{-t/D} + C \cdot e^{-t/D} \cdot \left(-\frac{1}{D}\right)$$

report dans l'équation avec second membre

$$C' \cdot e^{-t/D} = \frac{S}{D} \Rightarrow C' = \frac{S}{D} \cdot e^{t/D}$$

$$C = S \cdot e^{t/D} + K$$

report dans (E1)

$$Y = K \cdot e^{-t/D} + S$$

retour aux conditions initiales

$$K = Y_0 - S$$

solution générale

$$\boxed{\boxed{Y(t) = (Y_0 - S) \cdot e^{-t/D} + S}} \quad \begin{array}{ll} Si & Y_0 > S \Rightarrow Y(t) \rightarrow S + \varepsilon \\ Si & Y_0 < S \Rightarrow Y(t) \rightarrow S - \varepsilon \end{array}$$

L'équation du flux est donnée par

$$X(t) = \frac{S - Y(t)}{D} = -\frac{Y_0 - S}{D} \cdot e^{-t/D} = Y'(t)$$

$$\Rightarrow \begin{array}{ll} Si & Y_0 > S \Rightarrow Y'(t) < 0 \\ Si & Y_0 < S \Rightarrow Y'(t) > 0 \end{array}$$

Selon la position relative de Y_0 par rapport à S on converge vers S par valeurs supérieures ou inférieures, la décroissance ou la croissance se faisant à taux moins que proportionnel ($X(t) \rightarrow 0$)

Equation de la tangente à l'origine

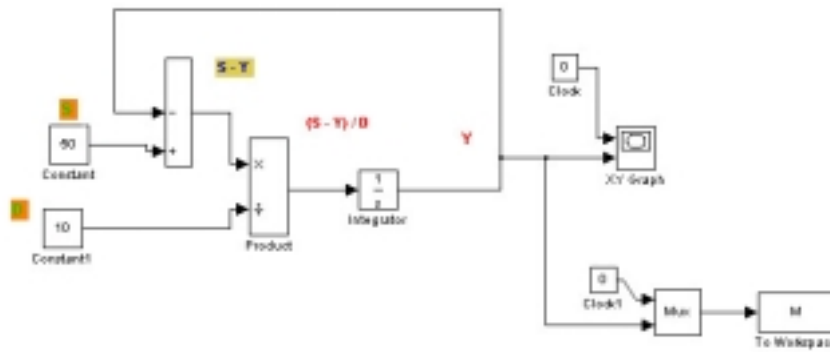
$$Y = Y_0 - \frac{Y_0 - S}{D} \cdot t$$

$$t = D \Rightarrow Y = S$$

D = temps nécessaire, à taux constant, pour que le niveau atteigne son seuil désiré S .

2.2.3.2 Simulation

On reprend le modèle RNSP que l'on transforme en BRNPOSP en utilisant des éléments de BRNPOSN. Dans ce dernier modèle on sélectionne l'assemblage permettant d'aller vers le Workspace (clic gauche + maintenir doigt enfoncé + sélectionner les éléments désirés). Sur tous les éléments sélectionnés on voit apparaître des petits carrés noirs. On fait (Ctrl + C = copier). On va dans RNSP, on clique sur une partie non occupée de la feuille et on fait (Ctrl + V = coller). On raboute la flèche allant vers le bloc Mux avec celle allant vers le bloc XY Graph. On modifie les annotations, on change les valeurs des constantes en prenant $S = 50$ et $D = 10$. On obtient le modèle suivant



Smlk046.

La modification de la taille et de la couleur des caractères, de la couleur du fond des cartouches a été obtenue en utilisant le menu format.

On va réaliser deux simulations avec $t_{start} = 0$ et $t_{stop} = 60$.

S1	$Y_0 = 5$
S2	$Y_0 = 100$

On notera M_i la matrice de la simulation n°i. C'est à dire qu'après avoir réalisé chacune des simulations (na pas oublier d'initialiser le bloc Integrator) on tape les expressions

» M1=M;
» M2=M;

Cela étant on veut tracer un graphe avec les deux tangentes à l'origine On fait déjà les transformations de données

t1=M1(:,1);
Y1=M1(:,2);
t2=M2(:,1);
Y2=M2(:,2);

On définit deux vecteurs pour les tangentes d'équations

$$Y_0 - \frac{Y_0 - S}{D} \cdot t \longrightarrow \begin{cases} 5 + 4.5t \\ 100 - 5t \end{cases}$$

```
tangor1=5+4.5*t;
tangor2=100-5*t;
```

En outre, si on veut faire apparaître graphiquement la valeur seuil on tape l'expression

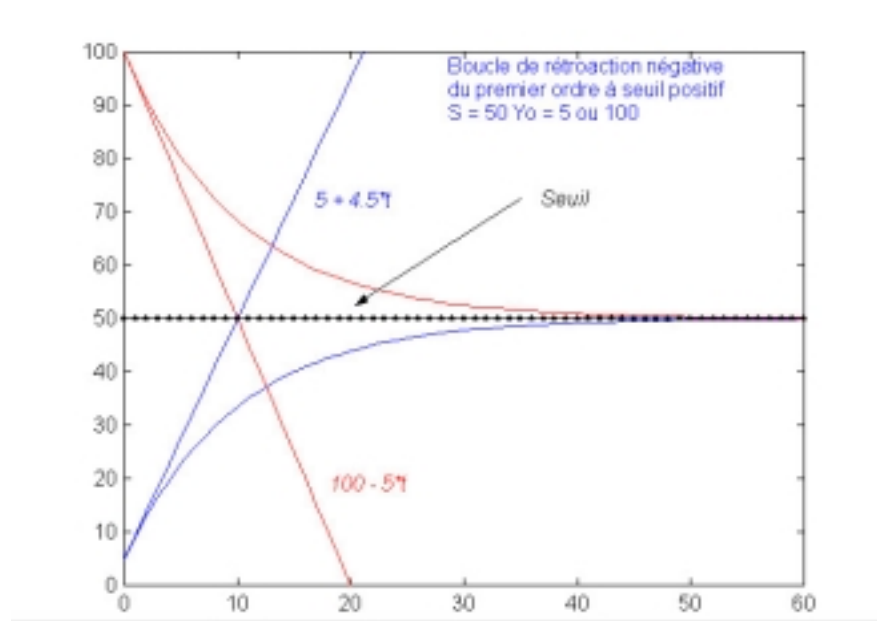
```
seuil = 50 * ones(61, 1);
```

qui veut dire : " je multiplie par 50 un vecteur unitaire de 61 lignes " (ne pas oublier que le vecteur t a 61 lignes).

Puis on trace le graphe en passant les expressions

```
» plot(t1,Y1,'-b',t2,Y2,'-r',t ,tangor1,'b',t,tangor2,'r',t,seuil,'.-k')
» set(gcf,'color','w')
» axis([0 60 0 100])
```

On obtient le graphe suivant



Smlk047.

2.2.4 Boucle de rétroaction positive du premier ordre à seuil.

2.2.4.1 Etude analytique.

C'est le cas d'une politique d'écart sur stock mal formulée sous la forme

$$X = \frac{Y - S}{D}$$

d'où l'équation différentielle et sa solution

$$\dot{Y} - \frac{Y}{D} = -\frac{S}{D} \Rightarrow Y(t) = (Y_0 - S) \cdot e^{t/D} + S$$

$$Si \quad Y_0 > S \Rightarrow Y(t) \rightarrow +\infty$$

$$Si \quad Y_0 < S \Rightarrow Y(t) \rightarrow -\infty$$

On a une divergence à taux plus que proportionnel

$$\dot{Y}(t) = \frac{Y_0 - S}{D} \cdot e^{t/D} \Rightarrow \begin{cases} Y_0 > S \Rightarrow \dot{Y}(t) \rightarrow +\infty \\ Y_0 < S \Rightarrow \dot{Y}(t) \rightarrow -\infty \end{cases}$$

2.2.4.2 Simulation.

On reprend le modèle antérieur en le transformant légèrement. On transforme les signes des entrées du bloc Sum passant de $(- +)$ à $(+ -)$ et on modifie les annotations corrélativement.. On sauve le modèle sous le nom *BRPPOS*. On prend les paramètres

$t_{start} = 0$	$t_{stop} = 20$
$S = 50$	$D = 10$

On réalise deux simulations

S1	$Y_0 = 40$
S2	$Y_0 = 60$

On tape les expressions

» M1=M;

» M2=M;

on réalise les transformations de variables

» t1=M1(:,1);

» Y1=M1(:,2);

» t2=M2(:,1);

» Y2=M2(:,2);

On redéfinit un vecteur seuil

» seuil=50*ones(21,1);

on trace le graphe si on frappe l'expression

» plot(t1,Y1,'-b',t2,Y2,'-r',t,seuil,'.-k')

on reçoit le message d'erreur

??? Error using ==> plot
 Vectors must be the same lengths.

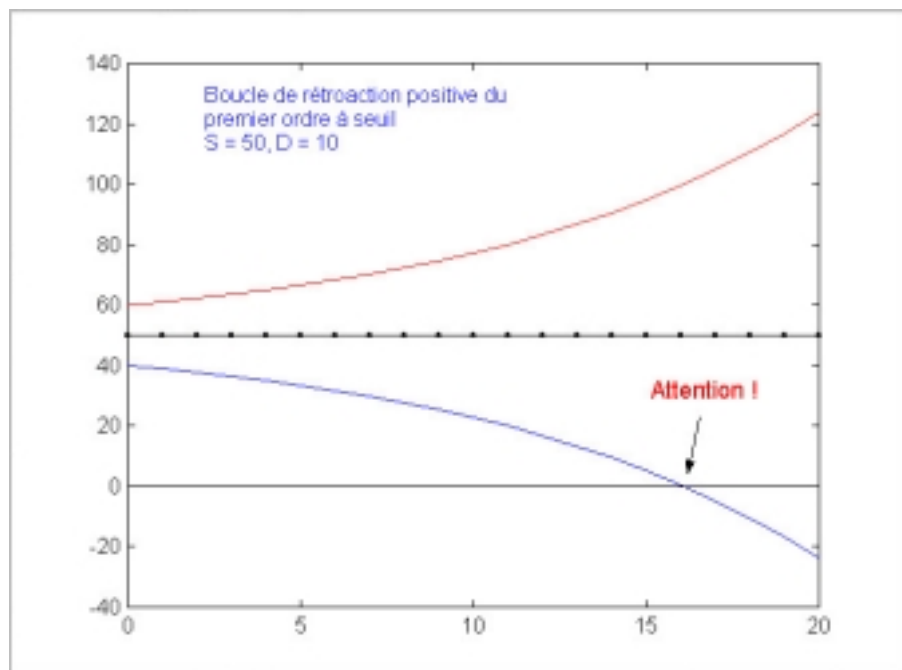
De fait le vecteur t décrit dans l'onglet **Workspace I/O** de *Simulation Parameters* va bien de 0 à 20 mais comporte 61 éléments. On passe donc l'expression

» plot(t1,Y1,'-b',t2,Y2,'-r',t1,seuil,'.-k')

suivie de

» set(gcf,'color','w')

On a le graphe amélioré suivant



Smlk048.

$Y_0 < S \rightarrow$ réservoir dont l'ouverture s'agrandit au fur et à mesure que le liquide s'écoule.

$Y_0 < S \rightarrow t^*$ tel que $Y(t) = 0$

$$t^* = D \cdot \ln \frac{S}{S - Y_0}$$

$$0 < Y_0 < S \Rightarrow \frac{S}{S - Y_0} > 1 \Rightarrow t^* > 0$$

$$D = 10 \quad S = 50 \quad Y_0 = 40 \Rightarrow 10 \cdot \ln \frac{50}{50 - 40} = 16.09438$$