SA - EXERCICE 3

REACTION BIMOLECULAIRE GENERALE

La réaction que nous voulons modéliser,

$$A + B \rightarrow C \qquad k \qquad (r\ 3)$$

comporte 2 réactifs et 1 produit. Nous avons vu cependant que, comme toutes les réactions élémentaires, elle est *monovariable*. Nous devons donc choisir la façon de procéder :

- avec une seule variable dans le programme, et laquelle ?

- avec plusieurs, ou toutes, les variables ?

La question n'est pas cruciale ici, et nous ne programmerons les deux approches que pour montrer une particularité de cette programmation. Cependant, d'une manière générale, le bon choix des variables et des paramètres est une étape importante de la programmation d'un modèle. Cela est toujours vrai du point de vue de la commodité, de la lisibilité et de la maintenance d'un programme (cela peut affecter aussi le point de vue numérique : certains programmes peuvent "planter" dans certaines conditions, simplement à cause d'un mauvais choix des variables).

Si vous le désirez, vous pouvez traiter le problème sous la forme intégrée, comme dans les exercices précédents. Il suffira d'adapter, par exemple, le programme bimolpur_1.cpp pour traduire l'équation *(15)*. La fonction `fappel` deviendra alors :

```
void fappel()
{
   Sa_data delta = ca[1][0]-ca[0][0];
```

```
   for (int i = 0; i < npt; ++i)
   {
      if (delta != 0)
         ca[0][i]=delta/(ca[1][0]/ca[0][0]*exp(delta*p[0]*ind[i])-1);
      else
         ca[0][i] = ca[0][0]/(ca[0][0]*p[0]*ind[i] + 1);

      ca[1][i] = ca[0][i] + delta;
   }
}
```

Noter la déclaration de `delta` et l'affectation de B_0-A_0 à sa valeur, à l'extérieur de la boucle. L'équation *(15)* n'étant valable que pour $\Delta \neq 0$, il est nécessaire de faire un test de sa valeur afin d'utiliser soit l'équation *(15)*, soit l'équation *(8)* sans le facteur 2.

L'expression `delta != 0` est vraie si delta est différent de 0. Le signe ! exprime la négation.

L'expression symétrique `delta == 0` (avec 2 signes =) serait vraie si delta égale 0. Le symbole == est un opérateur de **comparaison**, à ne pas confondre avec = (simple) qui est un opérateur d'**affectation** : l'expression `delta = 0` affecterait 0 à delta. Cette confusion est une source très fréquente d'erreur de programmation.

1. Version à une seule variable

fichier bimolgen_mono.cpp fichier_bimolgen_mono.sac

La particularité de cette version vient du fait que dans l'équation monovariable

$$dA/dt = -kA(A+\Delta) \qquad (14)$$

nous avons besoin, pour évaluer Δ, de la concentration initiale B_0. Celle-ci n'est pas accessible par la méthode utilisée jusqu'ici si nous utilisons une seule variable `ca[0][i]`, pour désigner A. La solution est très simple : il suffit d'utiliser un paramètre, disons `p[1]` pour désigner la concentration initiale B_0. La fonction `eqdiff` va alors s'écrire :

```
void eqdiff (Sa_data x, Sa_data* y, Sa_data* dy)
{
   dy[0] = - p[0]*y[0]*(y[0] + p[1] - ca[0][0]);  // dA/dt
}
```

et il faut créer un paramètre `p[1]` dans le fichier de commande.

Vous pouvez tester l'effet des concentrations A_0 et B_0, notamment les inverser ($B_0 < A_0$).

L'inconvénient de cette version est qu'elle ne vous donne que les concentrations de A. Assurez-vous en, si vous avez des doutes, en regardant le fichier de données (bouton **d** de la fenêtre principale). Naturellement, on pourrait la compléter pour avoir aussi B, C et tout ce qui peut se calculer à partir de A... mais ce serait empiéter sur la version suivante.

Elle est cependant une illustration simple de ce qu'il est possible, et quelquefois nécessaire de faire, ainsi que de la souplesse d'utilisation des variables globales de *Sa*.

2. Version à plusieurs variables

télécharger bimolgen_multi.cpp télécharger bimolgen_multi.sac

Voici le programme complet que nous proposons (les numéros de ligne ont été ajoutés pour faciliter les explications) :

```
1. // bimolgen_multi.cpp
2. //------------------------------------------------------
3. #include"global.h"
4. //------------------------------------------------------
5. void Identification(Modele& modele)
6. {
7. modele.Fichier = String(__FILE__);
8. modele.Version = String(__DATE__) + String(" ") + String(__TIME__);
9. modele.Auteur = "Nuno";
10 nom_syst = "bimolgen_multi";
11 n_diff = 1;
12 first_var = 0;
13 nv_mod = 5;
14 nexp = 1;
15 }
16 //------------------------------------------------------
17 // déclaration d'une variable globale
18 // à l'extérieur de toute fonction :
19 Sa_data delta;
20 //------------------------------------------------------
21 void eqdiff (Sa_data x, Sa_data* y, Sa_data* dy)
22 {
23    dy[0] = - p[0]*y[0]*(y[0] + delta); // dA/dt
24 }
25 //------------------------------------------------------
26 void fappel()
27 {
28    delta = ca[1][0] - ca[0][0]; // B0-A0
29
30    srkvi(n_diff, &ca[first_var], ind, npt, h0, tol, iset, jacob,
```

```
h_compt, c_min);
31
32    for (int i = 1; i < npt; ++i) // exclure i=0 !
33       ca[1][i] = delta + ca[0][i]; // B
34
35    for (int i = 0; i < npt; ++i)
36    {
37       ca[2][i] = 1/ca[0][i]; // 1/A
38       ca[3][i] = ca[0][i]*ca[1][i]/(ca[0][0]*ca[1][0]); // vitesse
normalisée
39       ca[4][i] = (ca[0][0]-ca[0][i]) / ca[0][0]; // chi
40    }
41 }
42 //---------------------------------------------------------
```

Explications

1) Tout d'abord, nous avons choisi de n'écrire qu'une équation différentielle (d*A*/d*t*), puisqu'elle est siffisante et la deuxième variable, B, pouvant être calculée algébriquement à partir de A. C'est l'option que nous choisirons en général. Notre version est donc à la fois "monovariable", dans le respect du vrai caratère du problème, et "multi" parce qu'elle utilise ca[1][0...] pour B, et ne nécessite donc pas l'usage d'un paramètre spécial pour désigner B_0.

2) Il n'était pas indispensable de créer une variable `delta` : on pourrait répéter chaque fois que nécessaire `ca[1][0] - ca[0][0]`. Toutefois, cela rend le programme plus structuré. La valeur de `delta` doit obligatoirement être affectée en tout début de `fappel` (ligne 28), afin de tenir compte des éventuels changements de concentrations initiales que l'on peut opérer avant de lancer une simulation. Mais si l'on déclarait la variable `delta` à cet endroit, elle serait locale à `fappel` et donc inaccessible dans `eqdiff`, où on en a besoin. Il faut donc la déclarer *en dehors de toute fonction*, et *avant toute utilisation* bien sûr, d'où sa place ligne 19.

a) En fait, ce n'est pas tout à fait une variable globale, comme les variables que vous utilisez sans avoir à les déclarer, `ca[][], p[], npt,` etc. Elle est accessible seulement à l'intérieur de ce fichier .cpp ci.

b) Si on lui affectait sa valeur à l'endroit de sa déclaration, ligne 19, il n'y aurait (malheureusement !) pas d'erreur de compilation, mais on ne sait pas à quel moment du déroulement du programme s'effectuerait réellement l'affectation, et certainement pas au bon moment.

3) La première boucle dans `fappel`, ligne 32, a pour but de calculer B, c'est-à-dire la variable n°1 `ca[1][i]`. Comme `ca[1][0]`, B_0, est déjà affecté, il n'est pas nécessaire de faire démarrer la boucle à `i=0`. Noter que le calcul ne serait pas faux, mais inutile. D'autre part, comme il n'y a qu'une seule instruction dans cette boucle, les accolades { } ne sont pas nécessaires.

4) La deuxième boucle, ligne 35, doit commencer, elle, à `i=0`. Nous avons ajouté, ligne 37, le calcul de 1/A. Les lignes 38 et 39 calculent la vitesse et l'avancement normalisés.

Vous avez surement relevé que nous avons introduit des détails qui ne sont pas strictement indispensables. Notre but n'est pas de rendre compliquées des choses qui ne le sont pas, mais de vous aider à être très attentif à ce que fait, ou ne fait pas, un programme. Il nous semble que c'est un des meilleurs moyens de vous éviter des erreurs décourageantes.

Simulations

Avec cette version du programme, vous pouvez tester toutes les propriétés de la réaction bimoléculaire générale décrites dans le cours :

A- ralentissement lorsque A tend vers 0 (augmentez la valeur de *Fin* pour voir à quel point la réaction "traîne")

B- effet de B_0, supérieur ou inférieur à A_0 : comment varie l'échelle de temps ? quelle est la valeur finale de A ?

C- pour différentes valeurs de B_0, tracez 1/A en fonction du temps : pour quelle valeur obtenez-vous une droite ?

D- observez également les courbes vitesse normalisée en fonction de l'avancement normalisé.