Chapitre 6

PROBLÈMES DIFFÉRENTIELS à CONDITIONS INITIALES

1 Introduction

Les cours de mathématiques sont riches de méthodes analytiques de résolution des équations différentielles : on pourrait croire que l'analyse numérique est inutile dans cette partie des mathématiques. La réalité est tout autre : seule une infime minorité des équations différentielles est analytiquement soluble, particulièrement les équations linéaires. A mesure que l'on s'écarte des problèmes scolaires pour s'approcher des questions réelles, les équations différentielles deviennent « de moins en moins» linéaires et de moins en moins solubles. Exemple : le pendule simple dont le mouvement est décrit, en l'absence de frottement par l'équation exacte :

$$y'' + k^2 \sin y = 0.$$

Dans le domaine des équations différentielles, on distingue deux types de problèmes bien différents. Les problèmes à condition(s) initiale(s) (ou « de Cauchy »), dont un exemple s'écrit :

$$y' = f(x, y) \quad ; \quad y(a) = Y_a \quad ; \quad x \geq a, \tag{1}$$

et les problèmes aux limites, comme par exemple :

$$y" = g(x, y, y') \quad ; \quad y(a) = A \quad ; \quad y(b) = B \quad ; \quad a \leq x \leq b.$$

Dans ce chapitre, je ne considère que la première catégorie, les problèmes à condition(s) initiale(s).

Il est aussi important de bien faire la différence entre équation différentielle et problème différentiel. La première admet diverses solutions, dépendant de paramètres qui sont les conditions initiales ; on peut parfois la déterminer analytiquement, mais jamais numériquement. Le second a une seule solution et c'est l'objet de ce chapitre que d'expliquer comment on peut la calculer numériquement.

Toutes les méthodes de résolution numérique des problèmes différentiels utilisent une discrétisation. Ayant fait choix d'un pas h, on va calculer la solution y pour les valeurs kh de la variable indépendante, $y_k = y(kh)$ et $y_0 = y(a)$. Les divers algorithmes se distinguent par les informations qu'ils utilisent pour calculer y_k : certains emploient y_{k-1} uniquement (on parle de méthode à pas séparés), d'autres utilisent un certain nombre de valeurs précédentes de y, dont y_{k-1} (ce sont les méthodes à pas liés ou à pas multiples).

Je supposerai presque toujours que l'équation différentielle à résoudre est du premier ordre et qu'elle est résolue en y' ; en général, j'appelle x la variable indépendante. Il arrive que l'équation différentielle ne soit pas résoluble en y' : on parle alors de problème algébrodifférentiel, dont l'étude sort du cadre de ce cours. L'hypothèse d'un problème du premier ordre peut paraître fort éloignée de la réalité ; c'est tout à fait vrai, mais ce n'est pas trop grave, comme vous allez le voir. Toute équation différentielle d'ordre supérieur à un peut s'écrire comme un système différentiel d'ordre un. Soit par exemple l'équation du second ordre :

$$y" = g(x, y, y'). \tag{2}$$

J'introduis la variable auxiliaire $z = y'$; je peux maintenant écrire l'équation précédente sous la forme équivalente

$$\begin{cases} z' = g(x, y, z), \\ y' = z. \end{cases} \tag{3}$$

Je suis maintenant confronté à un système d'équations différentielles (ou système différentiel) du premier ordre. Plus généralement, un système différentiel du premier ordre à deux fonctions inconnues s'écrit :

$$\begin{cases} z' = g(x, y, z), \\ y' = h(x, y, z). \end{cases} \tag{4}$$

Dans le cas précédent, la fonction h se réduisait à la variable z. Si j'introduis deux vecteurs : $\mathbf{r} = \{y, z\}$ et $\mathbf{s} = \{h, g\}$, le système ci-dessus prend la forme :

$$\mathbf{r}' = \mathbf{s}(x, \mathbf{r}). \tag{5}$$

Cette écriture est commode car elle permet d'obtenir sans peine un algorithme de résolution d'un système différentiel : il suffit de transposer en notation vectorielle l'algorithme valable pour une équation différentielle (du premier ordre). Les ouvrages classiques d'analyse numérique développent largement les théories relatives à la résolution des problèmes différentiels à une fonction inconnue, mais sont discrets quant aux systèmes différentiels ; je suivrai cet exemple et j'admettrai que les algorithmes développés pour une fonction inconnue sont encore valables pour plusieurs fonctions.

Avant d'aborder les algorithmes purement numériques, je passe en revue quelques méthodes analytiques qui peuvent servir à des calculs numériques.

2 Méthodes analytiques

2.1 Développement de Taylor

Je cherche la solution du problème différentiel classique $y' = f(x, y), y(a) = y_a$. Le théorème de Taylor offre, au moins en principe, une solution

$$y(x) = y(a) + (x - a)y'(a) + \frac{1}{2}(x - a)^2 y"(a) + \frac{1}{6}(x - a)^3 y'''(a) + ...$$

Je connais $y(a)$ (condition initiale), $y'(a)$ (par substitution dans l'équation différentielle), $y"(a)$ et toutes les dérivées d'ordre supérieur (par dérivation et substitution dans l'équation différentielle !). Ainsi :

$$y'' = [f(x, y)]' = f_x + f f_y, \tag{6}$$

où f_x, f_y représentent les dérivées partielles de f par rapport aux variables indiquées. Ensuite :

$$y''' = f_{xx} + 2f f_{xy} + f^2 f_{yy} + f_x f_y + f f_y^2.$$

La méthode est simple mais devient rapidement laborieuse. De plus, j'ai montré (chapitre 1, Calcul et tracé de fonctions) que le développement de Taylor pouvait converger très lentement. En conséquence, cette méthode ne s'emploie que «localement» : pour calculer y_{n+1} à partir de y_n ou pour calculer, à l'aide des conditions initiales, les premières valeurs de y dont certains algorithmes ont besoin pour démarrer et calculer la suite des y_k.

2.2 Méthode des coefficients indéterminés (Frobenius)

Je fais l'hypothèse que y peut s'écrire :

$$y = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots$$

et donc que :

$$y' = c_1 + 2c_2x + 3c_3x^2 + \cdots$$

En substituant ces développements dans l'équation différentielle, j'obtiens des relations entre coefficients c_i, que je résous de proche en proche, à partir de $c_0 = y_a$. Ayant un certain nombre de coefficients, il est simple de calculer y, à une certaine précision. Cette méthode est aussi assez laborieuse.

2.3 Méthode de Picard, ou d'approximations successives

Je me propose toujours de résoudre le problème de Cauchy (1). J'intègre les deux membres de l'équation différentielle, entre a et x (en renommant u la variable indépendante) :

$$\int_a^x y'du = y(x) - y(a) = \int_a^x f(u, y)du$$

Je ne sais bien sûr pas calculer l'intégrale du second membre (ce qui reviendrait à savoir résoudre l'équation différentielle), aussi vais-je procéder à une approximation brutale : je remplace y par une «approximation d'ordre zéro», $y^{[0]}$, que je prends égale à y_a ; l'intégrale est alors calculable en principe ; le résultat me donne l'approximation suivante de y, soit $y^{[1]}$:

$$y^{[1]} = y(a) + \int_a^x f(u, y^{[0]})du$$

J'itère ce procédé, jusqu'à ce qu'un critère de convergence convenable soit atteint :

$$y^{[n]} = y(a) + \int_a^x f(u, y^{[n-1]})du.$$

Cette méthode est peut-être encore plus lente que les précédentes !

Depuis quelques années, on trouve dans le commerce des programmes de manipulation algébrique (*DERIVE, MAPLE, MATHEMATICA, MACSYMA, MUPAD, etc.*) fonctionnant sur microordinateurs et capables de faire seuls les calculs algébriques des trois méthodes précédentes. Il est à prévoir que ces algorithmes vont connaître de ce fait une nouvelle jeunesse.

J'aborde maintenant les méthodes véritablement orientées vers l'analyse numérique, en commençant par un cas d'école.

3 Méthode d'Euler et méthodes de Taylor

3.1 Algorithme d'Euler

La méthode d'Euler n'est jamais employée en pratique (sauf peut-être comme constituant d'une méthode de résolution d'équations aux dérivées partielles), mais c'est un excellent exemple introductif, généralisable dans plusieurs directions intéressantes. L'algorithme proposé par Euler pour résoudre (1) s'écrit :

$$y_{n+1} = y_n + hf(x_n, y_n) = y_n + hf_n. \tag{7}$$

Je peux donner trois interprétations de cette formule.

* Approximation de la dérivée. La dérivée de y au point x_n est à peu près $y'_n \simeq (y_{n+1} - y_n)/h$; en écrivant qu'elle vaut f_n, je retrouve la méthode d'Euler.

* Approximation d'une intégrale. J'intègre les deux membres de l'équation (1) :

$$y(x_{n+1}) - y(x_n) = y_{n+1} - y_n = \int_{x_n}^{x_{n+1}} f[t, , y(t)]dt$$

Je remplace l'intégrale du second membre par son approximation la plus simple, l'aire d'un rectangle de base h et de hauteur f_n, ce qui donne :

$$y_{n+1} - y_n = hf_n.$$

* Développement de Taylor. Supposant connu y_n, je peux déterminer y_{n+1} à l'aide du théorème de Taylor :

$$y(x_n + h) = y(x_n) + hy'_n + O(h^2).$$

En négligeant le terme d'erreur et en identifiant y' et f, je retrouve la formule d'Euler.

Je termine par un résumé des caractéristiques de la méthode d'Euler. Pour calculer y_{n+1}, j'utilise la valeur de y_n et pas celles de $y_{n-1}, y_{n-2}, ...$ Il s'agit donc d'une méthode à pas séparés. Pour passer de y_n à y_{n+1}, je ne calcule f qu'une fois (Les méthodes de Runge-Kutta qui seront exposées plus loin utilisent plusieurs valeurs de f pour améliorer la précision sur y'). L'algorithme ne fait appel qu'à $f = y'$ et pas à ses dérivées d'ordre supérieur (Certains algorithmes s'appuient sur un développement de Taylor, et donc sur $y^{[n]}$, qui dépend des dérivées de f).

La programmation de l'algorithme d'Euler est des plus simples. Voici une ébauche de programme.

Lire h, x_0, y_0, x_{max}
Initialiser x à x_0, y à y_0
Tant que $x < x_{max}$ faire :
  calculer f = f(x,y)
  $y := y + hf$
  $x := x + h$
  imprimer x, y

La généralisation à un système différentiel est immédiate, en principe. La fonction inconnue, sa valeur initiale et le second membre sont maintenant remplacés par des vecteurs de $\Re^n$:

Lire $h, x_0, x_{max}, \mathbf{y}_0$
Initialiser x à x_0, $\mathbf{y}$ à $\mathbf{y}_0$
Tant que $x < x_{max}$ faire :
  calculer :
    $\mathbf{f} = \mathbf{f}(x, , \mathbf{y})$
    $\mathbf{y} := y + h\mathbf{f}$
    $x := x + h$
  imprimer $x, \mathbf{y}$

Il faut traiter sur un pied d'égalité toutes les coordonnées de chaque vecteur.

Exemple. Le programme ci-dessous résout, par la méthode d'Euler, l'équation différentielle du pendule simple : $y" + \sin y = 0$. Comme j'ai posé $y' = z$, je résous en fait le système du premier ordre équivalent

$$y' = z,$$
$$z' = -\sin y.$$

Les deux seconds membres apparaissent dans le programme sous forme de `function` ; la `procedure euler` calcule la solution en `tf=ti+pas`. Ces trois sous-programmes sont inutilement compliqués : ils contiennent par exemple la variable indépendante (`t`), sans objet ici, mais ils peuvent s'adapter facilement au cas d'un pendule soumis à un couple dépendant du temps. Le programme principal ne fait que gérer les éléments précédents.

```
PROGRAM pendule;
CONST
      np = 600;
TYPE
     vecteur = ARRAY[0..np] OF SINGLE;

FUNCTION smy(t,y,z: SINGLE): SINGLE;
BEGIN     smy := z; END;
FUNCTION smz(t,y,z: SINGLE):SINGLE;
BEGIN     smz := -sin(y); END;
PROCEDURE euler(h,ti,yi,zi: SINGLE; VAR tf,yf,zf: SINGLE);
BEGIN
     yf := yi + h*smy(ti,yi,zi); zf := zi + h*smz(ti,yi,zi);
     tf := ti + h;
END;

VAR
     t,y,z: vecteur;
     pas,y0,z0: SINGLE;
     i: INTEGER;
     nom_fich: STRING[50];
     fich : TEXT;
BEGIN
     WRITE('nom du fichier de sortie: '); READLN(nom_fich);
     nom_fich := 'D:\an\an_poly-6\' + nom_fich;
     ASSIGN(fich,nom_fich); REWRITE(fich);
     WRITE('position initiale: ');READLN(y0);
     WRITE('vitesse initiale: ');READLN(z0);
     WRITE('longueur du pas: '); READLN(pas);
     t[0] := 0; y[0] := y0; z[0] := z0;
     WRITELN(fich,t[0]:6:2,y[0]:10:4,z[0]:10:4);
     FOR i := 1 TO np DO BEGIN
          euler(pas,t[i-1],y[i-1],z[i-1],t[i],y[i],z[i]);
          WRITELN(fich,t[i]:6:2,y[i]:10:4,z[i]:10:4);
     END;
     CLOSE(fich);
END.
```

La figure montre le résultat de l'exécution. Comme vous pouvez le constater, la qualité de la méthode laisse à désirer : l'amplitude augmente nettement avec le temps.

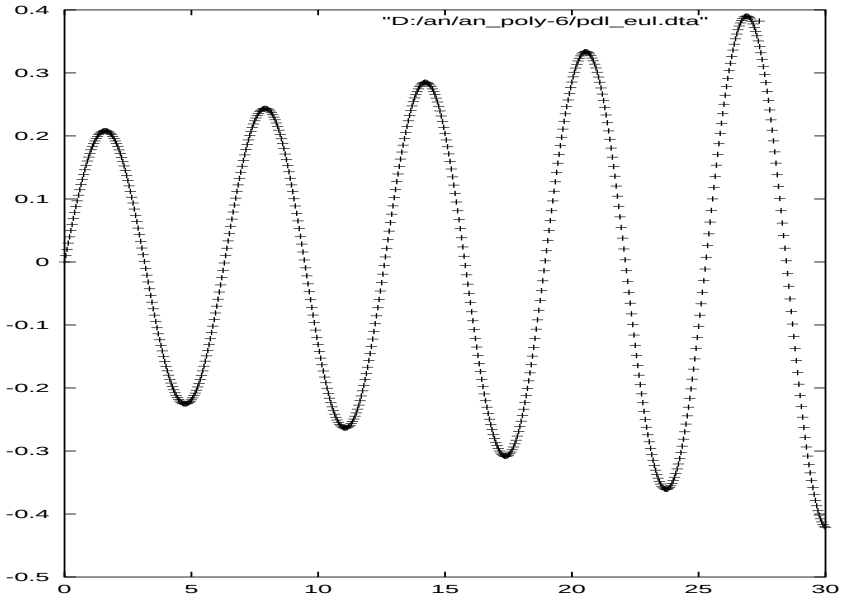

FIG. 1 – : Mouvement du pendule simple obtenu par l'algorithme d'Euler. Le pas était de 0,05.

3.2 Méthodes de Taylor

Comme je l'ai indiqué au §2.1, la méthode d'Euler peut être considérée comme une application, à l'ordre 1, du développement de Taylor. J'ai aussi montré comment l'on pouvait utiliser un développement d'ordre supérieur ((6). Je rappelle qu'il faut calculer, jusqu'à l'ordre souhaité, les dérivées de y, par les relations de récurrence :

$$y' = f(x,y) \quad ; \quad y" = f_x(x,y) + f_y(x,y)f \equiv f^{(1)},$$

$$f^{(k)} = f_x^{(k-1)} + f_y^{(k-1)} f,$$

puis déduire y_{i+1} par :

$$y_{i+1} = y_i + h\left[f(x_i,y_i) + \ldots + \frac{h^{p-1}}{p!} f^{(p-1)}(x_i,y_i)\right]$$

Cet algorithme n'est intéressant que si les $f^{(k)}$ se calculent facilement.

3.3 Erreurs, précision, stabilité

Je n'ai pas la prétention de construire la solution exacte du problème différentiel grâce à l'algorithme d'Euler ; le résultat du calcul pratique, soit y_n, est en réalité entaché d'erreurs dont il faut bien comprendre la nature.

Je suppose d'abord que je peux calculer avec un très grand nombre de chiffres significatifs. Le résultat y_n ne serait pas pour autant exact : il a en effet été obtenu à l'aide d'une approximation. A l'aide de l'une des interprétations données ci-dessus de l'algorithme d'Euler, je pourrais borner

l'erreur commise en passant de y_k à y_{k+1} (erreur locale de troncation), puis estimer l'erreur de troncation globale qui sépare y_n de la solution exacte évaluée en x_n :

$$t_n = y_n - y(x_n).$$

J'imagine maintenant que je dispose d'un algorithme exact capable de calculer y_n sans erreur de troncation. Le résultat a cependant été obtenu par une succession d'opérations arithmétiques, sur une machine comportant un nombre limité de chiffres significatifs : il présente donc une certaine erreur d'arrondi :

$$e_n = y_n - y(x_n).$$

En pratique, les deux sources d'erreur coexistent. Nous ne développerons pas ici l'étude des erreurs.

Le problème traité fournit parfois lui-même le moyen de contrôler la qualité de l'algorithme. C'est le cas en particulier pour les problèmes de mécanique sans frottement, pour lesquels l'énergie est conservée. Il suffit alors de calculer périodiquement l'énergie du système : si la variation d'énergie dépasse le seuil prévu, on rejette les dernières itérations et on reprend le calcul avec un pas plus petit.

Une autre propriété importante d'un algorithme de résolution d'équations différentielles est la stabilité. Dans le cas de l'algorithme d'Euler, il est facile d'apprécier la stabilité en étudiant un «problème modèle». Je considère le problème différentiel :

$$y' = y; y(0) = 1.$$

Ici, $f \equiv y$ et l'algorithme d'Euler s'énonce :

$$y_{n+1} = y_n + hy_n = (1 + h)y_n.$$

Il s'agit d'une relation de récurrence (on dit aussi une équation aux différences) pour y_{n+1}. Comme $y_0 = 1$, je trouve :

$$y_n = (1 + h)^n.$$

Vous voyez que y_n croit indéfiniment pour tout $h > 0$: on dit qu'il n'y a pas stabilité «absolue». Par contre, l'algorithme est «relativement stable», c'est à dire que $y_n/y \to 1$ pour $h \to 0$. Pour le montrer, il suffit de remarquer que $x = nh$ et donc que :

$$y_n = [(1 + h)^{1/h}]^x \to e^x.$$

Si $c < 0$, on a encore $y_n \to e^{-|c|x}$. Il y a toujours stabilité relative. Mais y_n ne sera uniformément décroissant (comme la solution exacte) que si $|1+hc| < 1$. Comme $c = -|c|$, cela implique $h < 2/|c|$.

4 Méthodes de Runge-Kutta

Les méthodes que nous allons décrire dans ce paragraphe peuvent être considérées comme des généralisations de celle d'Euler, où l'on calcule f (le second membre) plusieurs fois par pas, pour réduire l'erreur de troncation. Ce sont aussi probablement les méthodes les plus employées dans la pratique. Nous exposerons la théorie pour une méthode d'ordre 2, bien que les applications fassent généralement appel à des méthodes d'ordre 4 ou plus.

4.1 Méthodes d'ordre 2

Je cherche toujours la solution du problème représenté par (1) en supposant connue la solution numérique approchée y_n au point $x_n = x_0 + nh$. Pour cela, je vais construire une « fonction incrément » $\Phi(x_n, y_n, h)$ telle que :

$$y_{n+1} - y_n = h\Phi(x_n, y_n, h).$$

Cette fonction dépend aussi du second membre f. Soit d'autre part la solution exacte $z(t)$ du problème différentiel :

$$z'(t) = f[t, z(t)] \quad ; \quad z(x) = y.$$

En termes imagés, $z(t)$ est la solution de l'équation différentielle « qui passe par le point de coordonnées x et y » ; x et y sont ici considérés comme arbitraires mais constants. L'incrément exact est défini par :

$$z(x + h) - z(x) = h\Delta(x, y, h).$$

La quantité $\tau(x, y, h) = \Delta(x, y, h) - \Phi(x, y, h)$ est une indication de la qualité de l'approximation, c'est l'erreur locale de troncation de la méthode. Je souhaite bien sûr que τ tende vers 0 avec h, ce qui est équivalent à la condition :

$$\lim_{h \to 0} \Phi(x, y, h) = f(x, y)$$

On dit que la méthode est cohérente si cette condition est vérifiée dans tout le domaine de variation de x. De plus, on dit que la méthode est d'ordre p si $\tau(x, y, h) = O(h^p)$.

Je vais chercher, dans le cas d'une méthode d'ordre 2, une expression de Φ qui coïncide avec Δ jusqu'aux termes en h^2 compris. Je suppose (comme l'on fait Runge et Kutta il y a presque un siècle) que Φ est de la forme :

$$\Phi(x, y, h) = a_1 f(x, y) + a_2 f[x + p_1 h, y + p_2 h f(x, y)]$$

où les quantités a_1, a_2, p_1 et p_2 sont des constantes à déterminer. Il me suffit d'imposer que le développement de Taylor de Φ, à l'ordre 2 en h, coïncide avec celui de Δ au même ordre. Ils s'écrivent

$$\Delta \quad = f(x, y) + \frac{1}{2} h(f_x + f f_y) + O(h^2),$$

$$\Phi \quad = (a_1 + a_2) f + h a_2 p_1 f_x + h a_2 p_2 f f_y + O(h^2).$$

D'où trois conditions à remplir :

$$\begin{cases} a_1 + a_2 = 1, \\ a_2 p_1 = 1/2, \\ a_2 p_2 = 1/2. \end{cases}$$

Je dispose de quatre paramètres inconnus ; il est d'usage de conserver un paramètre libre et de déterminer les trois autres à l'aide des relations ci-dessus ; la dernière quantité sera ajustée après pour obtenir une propriété désirable de l'algorithme (bonne stabilité, erreur d'arrondi faible....). Je choisis de conserver a_2, rebaptisé α pour la circonstance :

$$\begin{cases} a_1 = 1 - \alpha, \\ a_2 = \alpha, \\ p_1 = p_2 = \frac{\alpha}{2}. \end{cases}$$

J'obtiens ainsi une méthode de Runge-Kutta d'ordre 2 caractérisée par la fonction incrément :

$$\Phi(x,y,h) = (1-\alpha)f(x,y) + \alpha f[x + \frac{h}{2\alpha}, y + \frac{h}{2\alpha}f(x,y)].$$

Parmi toutes les valeurs possibles de α, deux cas sont restés dans l'histoire : $\alpha = 1/2$ (on parle alors de méthode de Heun ou d'Euler améliorée) et $\alpha = 1$ (méthode d'Euler modifiée). Vous vérifiez facilement que ces algorithmes s'écrivent :

$$\alpha = 1/2 \quad ; \quad y_{n+1} = y_n + \frac{1}{2}hf(x_n,y_n) + f[x_n + h, y_n + hf(x_n,y_n)] \tag{8}$$

et

$$\alpha = 1 \quad ; \quad y_{n+1} = y_n + hf[x_n + h/2, y_n + \frac{1}{2}hf(x_n,y_n)]. \tag{9}$$

Exemple. Je reprends le problème du pendule simple. Il me suffit de remplacer, dans le programme précédent, la « procedure euler » par une « procedure rk2 ». J'introduis un perfectionnement supplémentaire : les valeurs calculées ne seront rangées en mémoire qu'une fois sur `prd`. Voici la `procedure rk2`.

```
PROCEDURE rk2(h,ti,yi,zi: SINGLE;VAR tf,yf,zf: SINGLE);
BEGIN
   yf := yi +(h/2)*(smy(ti,yi,zi) + smy(ti+h,yi+h*smy(ti,yi,zi),zi+h*smz(ti,yi,zi)));
   zf := zi +(h/2)*(smz(ti,yi,zi) + smz(ti+h,yi+h*smy(tt,yi,zi),zi+h*smz(ti,yi,zi)));
   tf := ti+h;
END;
```

Comme la variable `t` n'est pas utilisée dans les fonctions, la valeur de cet argument au moment de l'appel est sans importance ; elle est néanmoins conforme à la définition de l'algorithme. Le programme principal est lui-même modifié comme suit.

```
WHILE j < np*prd DO BEGIN
    INC(j);
    rk2(pas,tti,yyi,zzi,ttf,yyf,zzf);
    tti := ttf; yyi := yyf; zzi := zzf;
    IF (j mod prd) = 0 THEN
    WRITELN(fich, tti:6:2, yyi:10:4, zzi:10:4);
END;
```

Il est inutile de garnir des tableaux de résultats, puisque ceux-ci sont rangés au fur et à mesure sur disque. La figure 2 montre la loi du mouvement produite par cet algorithme. Vous pouvez constater que, cette fois, l'amplitude reste constante, malgré une valeur initiale très grande (presqu'égale à π). Dans ces conditions, le pendule est un système mécanique non-linéaire, dont le mouvement n'est plus sinusoïdal ni isochrone.

5 Méthode d'ordre 4

Je ne vais pas établir ici les formules de Runge-Kutta d'ordre 3 ou 4 ou plus : en effet, la longueur des calculs (mais non leur difficulté) croît très rapidement avec n. La méthode d'ordre 4 est la plus connue ; elle s'écrit

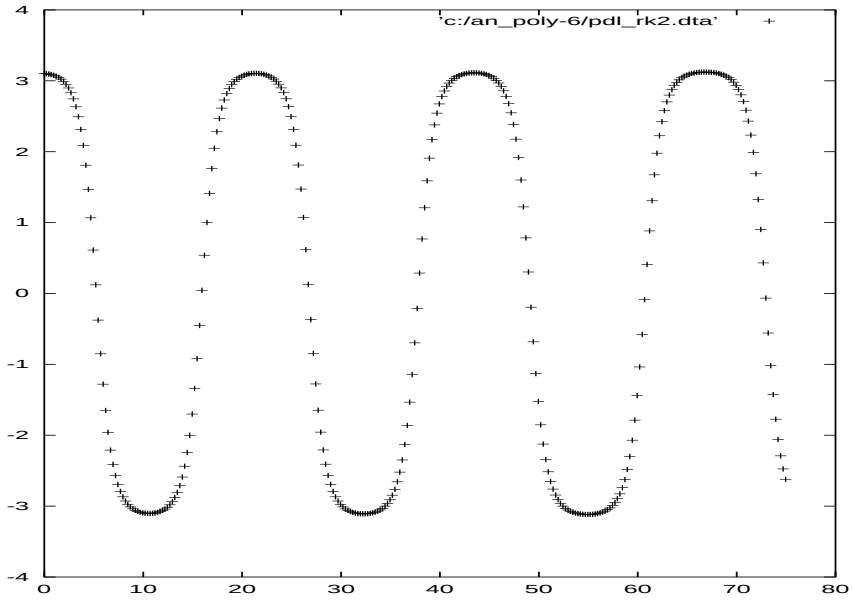

FIG. 2 – : Mouvement du pendule calculé par la méthode RK2. Le pas était de 0,05.

$$\begin{aligned} \Phi(x,y,h) &= \sum_1^4 a_j k_j, \\ k_j &= f(X_j, Y_j), \\ X_j &= x_n + h b_j, \\ Y_j &= y_n + h b_j k_{j-1}. \end{aligned} \tag{10}$$

où les coefficients a_j et b_j ont les valeurs suivantes :

j	1	2	3	4
a_j	1/6	1/3	1/3	1/6
b_j	0	1/2	1/2	1

5.1 Avantages et inconvénients de méthodes de Runge-Kutta

La méthode RK4 se distingue par sa simplicité de mise en oeuvre ; je la recommande d'ailleurs pour tous les problèmes simples. Elle ne requiert que quatre évaluations de f par pas. On peut améliorer aisément la précision des calculs (si le langage et le compilateur le permettent) en calculant $\Phi = \sum a_j k_j$ en double précision, car c'est là que se produisent la plupart des erreurs d'arrondi.

L'algorithme de Runge-Kutta souffre de deux inconvénients liés. On ne dispose d'aucune indication sur l'erreur de troncation en cours de calcul. D'autre part, il y a des problèmes de stabilité dès que le second membre est grand ou rapidement variable. Une méthode simple (mais coûteuse) de surveillance de la précision consiste à conduire deux calculs simultanément, l'un avec le pas h, l'autre avec le pas $h/2$. Tant que la distance entre les deux résultats reste inférieure à un certain

seuil, on admet que le résultat est valable. Si le seuil est franchi, il faut rejeter la (ou les) dernière(s) valeur(s) calculée(s) et repartir avec un pas plus petit.

Il existe des algorithmes plus raffinés (qui portent les noms de Runge-Kutta-Fehlberg ou méthodes de Runge-Kutta emboîtées). Dans ce type de méthode, on calcule, par exemple, six fois f par pas ; on combinant «bien» les résultats, on obtient une estimation précise de y_{n+1} ; une autre combinaison des mêmes valeurs fournit une estimation de l'erreur. La seule difficulté réside dans la forme compliquée des coefficients qui remplacent les a_j et b_j, mais l'ordinateur n'en a cure.

5.2 Organisation d'un programme

Il est commode de construire le programme de résolution sous une forme bien hiérarchisée, qui permette de changer commodément d'algorithme et d'équation. Le sous-programme de niveau le plus bas fait avancer la solution de x_n à x_{n+1} ; c'est lui qui renferme les détails de l'algorithme. Ce «noyau» de niveau zéro est englobé dans un sous-programme qui gère la surveillance de l'erreur (quand elle existe). Avec un pas h, on calcule y_{n+1} et une estimation τ_{n+1} de l'erreur locale de troncation ; si celle-ci est inférieure au seuil, le pas est accepté ; sinon, on rejette le dernier résultat, on diminue le pas et on recommence. Cet ensemble est à son tour inclus dans un sous-programme qui conduit l'intégration de x_{min} à x_{max} et prend en charge le stockage des résultats intermédiaires. Celui-ci est un peu délicat si l'on a affaire à des pas de longueur variable : comment concilier cette démarche avec l'enregistrement des résultats pour des abscisses à peu près régulièrement réparties ? Il est fréquent, quelque soit l'algorithme d'intégration, que l'on doive utiliser un pas assez petit, ce qui conduit à calculer de nombreuses valeurs de x_n, alors qu'on ne peut ou ne veut conserver qu'un petit nombre de valeurs (pour les tracés par exemple). On peut concilier ces deux exigences en ne conservant qu'une valeur de x toutes les p. La couche extérieure du programme est spécifique de l'application ; elle est destinée à la lecture des paramètres et des valeurs initiales, à l'affichage des résultats. Les seconds membres sont en général définis dans un sous-programme spécial.

6 Algorithme de prédiction-correction

Les méthodes de Runge-Kutta posent parfois des problèmes de stabilité ou de précision. Je vais maintenant décrire des méthodes toujours stables et généralement plus précises ; le prix à payer pour ces améliorations sera une structure plus compliquée de l'algorithme.

6.1 Prédiction

Je vais détailler un algorithme élémentaire de prédiction-correction, en commençant par la partie la plus simple, l'algorithme de prédiction. Je cherche la solution du problème différentiel (1). La variable indépendante x sera discrétisée avec un pas h. En reprenant l'une des interprétations de la méthode d'Euler, je remplace y' par une forme approchée, mais plus précise que celle utilisée au §1 :

$$y' = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(\xi)$$

soit, en reportant dans l'équation différentielle :

$$y_{n+1} = y_{n-1} + 2hf_n + \frac{h^3}{3} f'''(\xi).$$

À partir de cette relation exacte, j'obtiens une méthode d'intégration en négligeant le terme d'erreur (qui me fournira l'erreur locale de troncation) :

$$y_{n+1} = y_{n-1} + 2hf_n. \tag{11}$$

Par ailleurs, vous pourrez montrer sans peine que si l'on intègre terme à terme l'équation différentielle et que l'on remplace l'intégrale de $f(x, y)$ par son approximation dite du point milieu, on retrouve le même algorithme. L'amorçage du calcul suppose connus y_0 et y_1, que l'on obtient par Taylor ou Runge-Kutta d'ordre 2 ou encore par Euler. La figure 3 illustre cet algorithme.

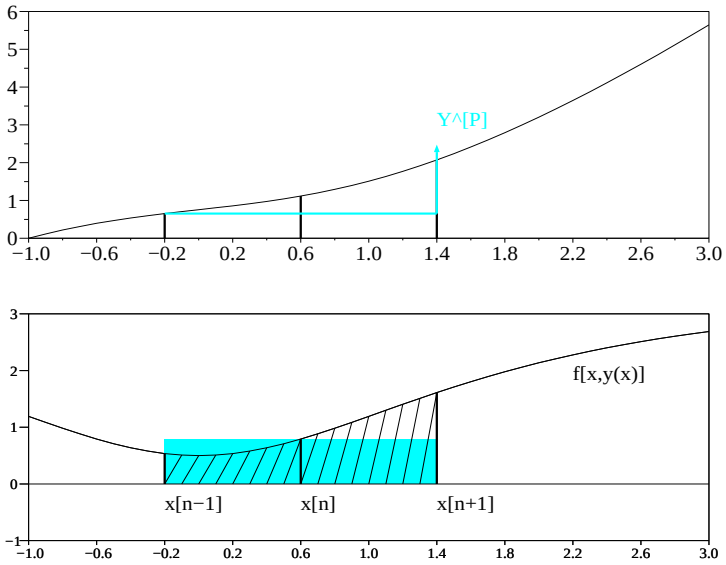

FIG. 3 – : Étape de prédiction. L'aire du rectangle vert est une approximation de l'aire hachurée ; on l'ajoute à y_{n-1} pour obtenir $y_{n+1}^{[P]}$

6.2 Correction

La partie « correction » est plus subtile. Cette fois, j'intègre (de façon approchée) l'équation différentielle entre x_n et x_{n+1} (remarquez les nouvelles limites) par la méthode des trapèzes et je trouve

$$y_{n+1} = y_n + \frac{1}{2}h(f_n + f_{n+1}) - \frac{h^3}{12}y'''(\xi).$$

Comme précédemment, je néglige le terme d'erreur pour obtenir une méthode d'intégration du problème différentiel :

$$y_{n+1} = y_n + \frac{1}{2}h(f_{n+1} + f_n). \tag{12}$$

Cette méthode est stable et précise. Elle souffre cependant d'un petit défaut : elle est inapplicable en l'état. En effet, la relation ci-dessus est une équation implicite en y_{n+1}, puisque cette quantité figure dans f_{n+1}. Il serait dommage d'échouer si près du but ; en réalité, je peux très bien

résoudre l'équation en y_{n+1} par approximations successives, à condition de connaître une valeur initiale $y^{[0]}$. Il suffit d'appliquer la méthode du point fixe

$$y_{n+1}^{[k+1]} = y_n + \frac{1}{2}h[f(x_n, y_n) + f(x_{n+1}, y_{n+1}^{[k]})]$$

jusqu'à ce qu'un critère de convergence convenable soit atteint. La dernière valeur calculée est retenue comme valeur de y_{n+1}.

Comment vais-je me procurer la valeur $y_{n+1}^{[0]}$? Vous l'avez deviné : par la méthode de prédiction. Celle-ci est explicite : elle fournit une valeur approchée de la solution, que j'appelle maintenant la valeur « prédite », $y_{n+1}^{(P)}$ et cette valeur prédite servira à amorcer l'itération du correcteur. L'algorithme complet peut donc s'écrire :

$$y_{n+1}^{[P]} = y_{n-1} + 2hf_n \quad , \quad y_{n+1}^{[C]} = y_n + (h/2)[f_n + f(x_{n+1}, y_{n+1}^{[P]})]. \tag{13}$$

L'itération du correcteur peut, en principe, être répétée plusieurs fois. Cela s'avère inutile en général : en cas de difficultés, il vaut mieux diminuer le pas. Dans tous les cas, $y_{n+1}^{[C]}$ devient la valeur définitive y_{n+1}. La figure 4 est une représentation graphique de ce calcul.

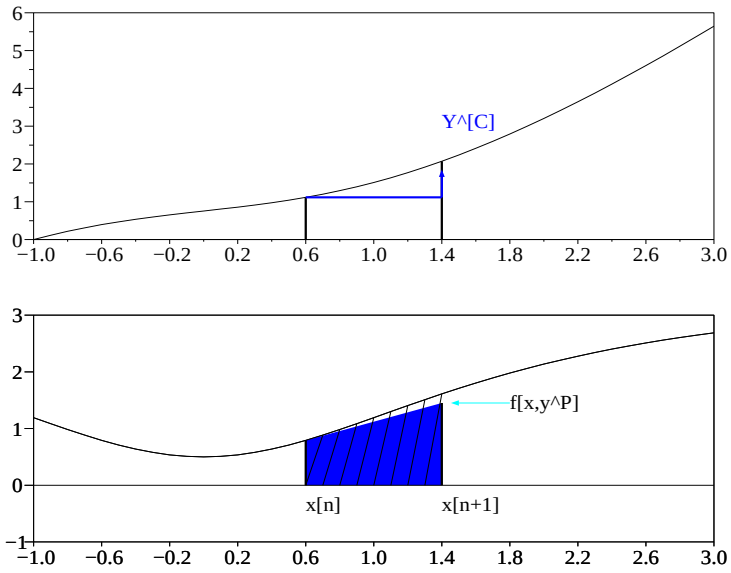

FIG. 4 – : Étape de correction. L'aire du trapèze bleu est une approximation de l'aire hachurée ; on l'ajoute à y_n pour obtenir $y_{n+1}^{[C]}$

6.3 Convergence du correcteur

On montre que si l'équation différentielle obéit à la condition de Lipschitz et si h est assez petit, l'itération du correcteur converge. Plus précisément, la condition de Lipschitz s'énonce ainsi. Il existe une constante positive K telle que, pour toutes valeurs de x, y_1, y_2, on ait

$$|f(x, y_1) - f(x, y_2)| \leq K|y_1 - y_2|.$$

Le pas h devra satisfaire à

$$Kh/2 < 1.$$

La qualité de l'itération dépend de la valeur initiale $y^{[0]}$; il est recommandé de choisir une formule de prédiction et une formule de correction présentant des erreurs du même ordre. On montre que, dans ce cas, une seule itération suffit. Si la valeur initiale provient de la méthode d'Euler (erreur en h^2), il faudra au moins deux itérations.

Exemple. Je résous une fois de plus le problème du pendule simple, à l'aide de l'algorithme de prédiction-correction. Le sous-programme `rk2` est remplacé les `procedure` suivantes.

```
FUNCTION f(x,y,z:REAL):REAL;
BEGIN
     f := z;  {y'}
END;
FUNCTION g(x,y,z: REAL):REAL;
BEGIN
     g := - sin(y); {z' = y"}
END;
PROCEDURE pc(h,tnm1,ynm1,znm1,tn,yn,zn: REAL; VAR tnp1,ynp1,znp1: REAL);
VAR yp,zp: REAL;

  PROCEDURE pred(h,ynm1,znm1,tn,yn,zn: REAL; VAR ynp1,znp1: REAL);
  BEGIN
     ynp1 := ynm1 + 2*h*f(tn,yn,zn);
     znp1 := znm1 + 2*h*g(tn,yn,zn);
  END;

  PROCEDURE corr(h,tn,yn,zn,yp,zp: REAL; VAR ynp1,znp1: REAL);
  BEGIN
     ynp1 := yn + (h/2)*(f(tn,yn,zn) + f(tn+h,yp,zp));
     znp1 := zn + (h/2)*(g(tn,yn,zn) + g(tn+h,yp,zp));
  END;

BEGIN
   pred(h,ynm1,znm1,tn,yn,zn,yp,zp);
   corr(h,tn,yn,zn,yp,zp,ynp1,znp1);
   tnp1 := tn + h;
END;
```

Il faut aussi modifier le programme principal, pour utiliser deux valeurs de y. La gestion de l'affichage tous les `prd` pas a été modifiée.

```
    FOR i := 1 TO nm1  DO BEGIN
       FOR j := 1 TO prd DO BEGIN
          pc(pas,tavder,yavder,zavder,tder,yder,zder,tf,yf,zf);
          tavder := tder; yavder := yder; zavder := zder;
          tder := tf; yder := yf; zder := zf;
       END;
```

```
    {affichage et écriture sur disque}
END;
```

6.4 Surveillance de l'erreur

Une qualité importante des méthode de prédiction-correction est qu'elles permettent, sans calculs supplémentaires, une surveillance de l'erreur de troncation. La solution exacte Y_{n+1} de l'équation différentielle en x_{n+1} est reliée à la valeur prédite par la relation :

$$Y_{n+1} = y_{n+1}^{[P]} + \frac{h^3}{3} y'''(\xi_p).$$

Si le correcteur converge, il diffère de la solution exacte par son erreur de troncation :

$$Y_{n+1} = y_{n+1}^{[C]} - \frac{h^3}{12} y'''(\xi_c).$$

Je dispose ainsi d'un encadrement de Y et d'une majoration de l'erreur. Si je suppose en effet que $\xi_p \cong \xi_c$, je peux écrire $y''' \cong (12/5h^3)(y^{[C]} - y^{[P]})_{n+1}$, d'où une estimation de l'erreur de troncation au point n (pour le calcul de y_{n+1}) :

$$e_n = \frac{1}{5}(y^{[P]} - y^{[C]})_{n+1}$$

L'utilisation de ce résultat est évidente. A chaque pas, je calcule $|e_n|$; si cette quantité est supérieure à un seuil fixé à l'avance, je rejette ce pas, je diminue h et je repars. Je peux augmenter h si $|e_n|$ est « très petit ».

Je pourrais aussi utiliser ma connaissance (approchée) de l'erreur pour améliorer $y_{n+1}^{[C]}$. Il suffit de calculer :

$$y_{n+1}^{modif} = y_{n+1}^{[c]} + (1/5)(y^{[P]} - y^{[C]})_{n+1}.$$

Ce perfectionnement est peu employé, car il nuit à la stabilité.

6.5 Formules d'ordre 4

En pratique, on utilise des méthodes du quatrième ordre. Elles peuvent être obtenues de plusieurs façons : par la recette des coefficients indéterminés ou par intégration approchée de l'équation différentielle, comme je l'ai expliqué à propos de la méthode d'Euler. Si je remplace la fonction compliquée f par un polynôme d'interpolation du second degré (parabole), j'obtiendrai une méthode d'ordre 4.

Je vais présenter une catégorie particulière de méthodes, celles d'Adams. L'algorithme d'Adams-Bashforth est une méthode explicite d'ordre 4 qui s'écrit :

$$y_{n+1} = y_n + (h/24)(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}), \tag{14}$$

où, comme d'habitude, $f_n = f(x_n, y_n)$ est la valeur du second membre au point x_n. Cette équation va servir à calculer la valeur prédite $y_{n+1}^{[}P]$. Le terme d'erreur correspondant est :

$$+\frac{251}{720} h^5 y^{(5)} \tag{15}$$

À cette équation de prédiction, j'associe le correcteur d'Adams-Moulton :

$$y_{n+1} = y_n + (h/24)(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2}). \tag{16}$$

Il s'agit encore une fois d'une équation implicite pour y_{n+1} (qui figure dans f_{n+1}). L'erreur de troncation s'écrit :

$$-\frac{19}{720}h^5 y^{(5)}. \tag{17}$$

Il est encore possible d'estimer l'erreur de troncation, et donc de la surveiller, à partir de la différence $y^{[P]} - y^{[C]}$, qui vaut, d'après (15) et (17) :

$$y_{n+1}^{[P]} - y_{n+1}^{[C]} = \frac{3}{8}h^5 y^{(5)}$$

L'algorithme d'Adams admet la représentation graphique présentée page suivante.

6.6 Démarrage des méthodes de prédiction-correction

Tous les algorithmes de prédiction-correction utilisent plusieurs valeurs déjà connues de la fonction pour calculer y_{n+1} (méthodes à pas multiples ou liés). Ceci complique l'amorçage du calcul. Considérons le cas particulier de la méthode d'Adams, qui nécessite quatre valeurs de y. La première sera fournie par la condition initiale, mais les trois suivantes devront être calculée par un algorithme indépendant : Taylor ou Runge-Kutta. Il est recommandé d'utiliser des formules du même ordre que pour l'algorithme principal.

6.7 Avantages et inconvénients des méthodes à pas multiples

Les méthodes de prédiction-correction sont très appréciées des spécialistes, qui en ont poussé très loin l'analyse théorique. Leur principal avantage est qu'elles permettent une estimation aisée de l'erreur de troncation. Leur stabilité est aussi renommée. Nous n'avons pas examiné ce point, mais l'étude du système modèle $y' = -ky$, montrerait que l'algorithme de correction est stable quelque soit h, alors l'équation de prédiction peut être instable (mais sans danger puisque elle ne sert qu'à fournir une valeur intermédiaire et locale).

Les inconvénients des méthodes "p-c" sont dus à la complexité de l'algorithme. Une méthode d'ordre 4 requiert, pour démarrer, les 3 premières valeurs de y, lesquelles doivent être obtenues sans "trop" d'erreur, par exemple par Runge-Kutta d'ordre 4. La modification du pas h n'est pas aussi simple qu'on pourrait le croire. Si je m'aperçois que l'erreur de troncation lors du calcul de y_{n+1} est trop grande, je rejette cette valeur et et je divise h par 2. Le calcul peut-il repartir pour autant ? Pas tout de suite, car l'évaluation de y_{n+1} demande la connaissance des 4 valeurs précédentes de y ; or, je ne les connais que pour le pas h et non pour $h/2$. Il faut donc interpoler entre valeurs de y pour trouver les précurseurs de y_{n+1} avec le pas $h/2$.

7 Méthodes pour les équations du second ordre

La physique fournit souvent des équations différentielles du second ordre où y' ne figure pas. C'est le cas en mécanique classique (au moins pour les systèmes à un seul degré de liberté) en l'absence de frottement ; c'est aussi le cas en mécanique quantique pour les problèmes à une dimension. Il existe plusieurs algorithmes spécialisés et bien adaptés à ce type de problème. J'en décrirai deux.

7.1 Algorithme de Verlet ou de saute-mouton

Je note $y(t)$ la fonction inconnue, solution de l'équation différentielle particulière :

$$y'' = f(t, y) \tag{18}$$

où la variable indépendante peut être le temps. La méthode résulte du remplacement de la dérivée seconde par une approximation bien connue

$$y''|_n \cong \frac{1}{h^2}[y_{n+1} - 2y_n + y_{n-1}],$$

où l'erreur de troncation est $O(h^2)$. L'algorithme de Stoermer-Verlet s'écrit donc

$$y_{n+1} = 2y_n - y_{n-1} + h^2 f(t, x). \tag{19}$$

Par suite de la symétrie de cette formule, les vitesses (et tous les termes impairs en h) ont disparu. En cas de besoin, je pourrai estimer les vitesses au moyen de la relation

$$y'_n = v_n = \frac{1}{2h}[y_{n+1} - y_{n-1}].$$

Cet algorithme jouit de plusieurs avantages. Il est rapide, la seule étape coûteuse en temps étant le calcul du second membre, qui intervient une fois par pas. Le terme d'erreur est petit et une analyse plus détaillée montrerait que l'énergie totale est conservée à chaque pas, jusqu'aux termes en h^2. Il est de ce fait presque universellement employé pour les calculs de dynamique moléculaire où l'on doit suivre les trajectoires de milliers d'atomes simultanément.

Les bonnes performances de la méthode de Verlet en mécanique classique tiennent à la forme particulière des équations de la dynamique. Celles-ci s'écrivent, sous la forme donnée par Hamilton

$$\dot{p} = -\frac{\partial H}{\partial q} \quad ; \quad \dot{q} = \frac{\partial H}{\partial p}.$$

(q est une coordonnée généralisée et p une quantité de mouvement généralisée). Ce sytème différentiel est particulier : c'est la même fonction (l'énergie totale ou le hamiltonien H) qui figure au second membre. L'algorithme de Verlet respecte bien cette structure et on le qualifie de «symplectique» (entrelacé en grec).

7.2 Algorithme de Numerov

C'est à un astronome russe (publiant vers 1935) que l'on doit une méthode d'intégration spécialement adaptée aux équations de la mécanique céleste et qui porte son nom. Je cherche à résoudre l'équation (18), mais avec une précision bien supérieure à celle permise par la méthode de Verlet.

L'algorithme de Numerov est une méthode de prédiction-correction. Je m'intéresse tous d'abord à la partie «correction», que je vais traiter par la méthode des coefficients indéterminés. Je fais l'hypothèse que y_{n+1} peut s'exprimer comme

$$y_{n+1} = a_0 y_n + a_1 y_{n-1} + a_2 y_{n-2} + h^2 b_{-1} f_{n+1} + b_0 f_n + b_1 f_{n-1} + b_2 f_{n-2}).$$

Le coefficient h^2 est là pour que les b_i soient des nombres sans dimensions. La présence d'un coefficient b_{-1} non nul indique qu'il s'agit d'une méthode implicite (y_{n+1} est donné en fonction

de f_{n+1} qui dépend elle-même de y_{n+1}). Je dispose de sept paramètres inconnus $\{a_i, b_i\}$ entre lesquels je vais imposer six relations ; j'utiliserai à la fin les résultats de Numerov pour lever l'indétermination. J'impose donc que la formule précédente soit exacte pour $y = 1, x, x^2, x^3, x^4$ et x^5. Comme d'habitude, je peux choisir $x = 0$, puisque les relations obtenues doivent être vérifiées quel que soit x. Il vient :

$$\left\{\begin{array}{l} 1 = a_0 + a_1 + a_2, \\ h = -h(a_1 + 2a_2), \\ h^2 = h^2(a_1 + 4a_2) + 2h^2(b_{-1} + b_0 + b_1 + b_2), \\ h^3 = -h^3(a_1 + 8a_2) + 6h^3(b_{-1} - b_1 - 2b_2), \\ h^4 = h^4(a_1 + 16a_2) + 12h^4(b_{-1} + b_1 + 4b_2), \\ h^5 = -h^5(a_1 + 32a_2) + 20h^5(b_{-1} - b_1 - 8b_2). \end{array}\right.$$

J'exprime les coefficients en fonction de a_2

$$\left\{\begin{array}{l} a_0 = 2 + a_2 \quad , \quad a_1 = -1 - 2a_2, \\ b_{-1} = -1/12 \quad , \quad b_0 = (10 - a_2)/12, \\ b_1 = (1 - 10a_2)/12 \quad , \quad b_2 = -a_2/12. \end{array}\right.$$

Selon Numerov, le choix $a_2 = 0$ jouit de qualités intéressantes : c'est celui que je retiendrai. La formule définitive (avec le terme d'erreur que je pourrais obtenir à partir d'un développement de Taylor des y_i) s'écrit :

$$y_{n+1} = 2y_n - y_{n-1} + \frac{h^2}{12}(f_{n+1} + 10f_n + f_{n-1}) - \frac{h^6 y^{(6)}(\xi_c)}{240}. \tag{20}$$

Pour être complet, je cite la formule de prédiction établie par Numerov, que vous pouvez démontrer par la même méthode des coefficients indéterminés :

$$y_{n+1} = 2y_{n-1} - y_{n-3} + \frac{4h^2}{3}(f_n + f_{n-1} + f_{n-2}) + \frac{16h^6 y^{(6)}(\xi_p)}{240}$$

Vous pourrez aussi trouver l'erreur de troncation en fonction de $y^{[P]}$ et de $y^{[C]}$. Je vais particulariser la formule de correction (la plus précise et la plus stable) pour le cas d'un problème linéaire, où l'équation différentielle devient :

$$y'' = A(x)y + B(x).$$

Cette hypothèse entraîne une simplification considérable, car le second membre est résoluble en y et l'équation de correction n'est plus implicite. Je peux donc abandonner l'étape de prédiction pour écrire :

$$y_{n+1} = \frac{1}{1 - \frac{h^2}{12}A_{n+1}}[2y_n - y_{n-1} + \frac{h^2}{12}[B_{n+1} + 10(A_n y_n + B_n) + A_{n-1}y_{n-1} + B_{n-1})]. \tag{21}$$

Cet algorithme est facile à programmer et donne de très bon résultats.

8 Équations "raides"

Avant de terminer ce chapitre, je mentionne une difficulté que l'on rencontre aussi bien en cinétique chimique qu'en dynamique des structures, sans offrir de solution concrète.

Certains problèmes de cinétique chimique ont pour expression mathématique un système différentiel de la forme :

$$\mathbf{y}' = \mathbf{A}\mathbf{y} + \mathbf{f}(t)$$

où $\mathbf{y}$ est un vecteur dont les n coordonnées représentent des concentrations, $\mathbf{A}$ une matrice carrée constante d'ordre n et $\mathbf{f}$ le vecteur de seconds membres. Je suppose que les valeurs propres λ_k de $\mathbf{A}$ soient réelles et distinctes. Je connais alors la forme de la solution :

$$\mathbf{y} = \sum c_k \exp(\lambda_k t)\mathbf{z}_k + \varphi(t).$$

Du point de vue numérique, il se posera un problème de stabilité chaque fois qu'une valeur propre sera négative et le problème sera d'autant plus aigu que les λ_k seront plus différents les uns des autres. En termes qualitatifs, il me faudra choisir un pas très petit pour traiter correctement les décroissances les plus rapides (grand λ) mais, avec ce même pas, il faudra un nombre immense d'itérations pour voir évoluer les décroissances les plus lentes. On dit que le système d'équations différentielles est "raide".

Un problème analogue se rencontre pour des systèmes du second ordre

$$\mathbf{y}'' = \mathbf{A}\mathbf{y} + \mathbf{f}(t)$$

comme on en trouve en mécanique des vibrations. Ici, l'échelle de temps est en principe définie par la période du mouvement, mais il y a plusieurs périodes, puisque la matrice $\mathbf{A}$ a plusieurs valeurs propres, lequelles peuvent être très différentes. Je serai encore une fois obligé de choisir un pas très petit pour suivre le mouvement le plus rapide et, simultanément, de prolonger le calcul pour décrire le mouvement le plus lent.

Dans la pratique, la difficulté est souvent accrue par la non-linéarité des seconds membres. Seules des méthodes implicites sont assez stables pour traiter ce type de problèmes. La convergence de la méthode d'itération décrite au §6 n'est pas assurée dans ce cas et il faut résoudre l'équation implicite du « correcteur » par une méthode de Newton.