Université d'Orléans U.F.R. Faculté des Sciences

Licence de Physique

Analyse Numérique pour les Sciences Physiques

Calcul et Représentation graphique de fonctions

Jean-Philippe Grivet, 2003

1 Représentation graphique de fonctions

L'une des activités les plus fréquentes en informatique scientifique consiste à représenter l'allure d'une fonction à l'aide d'un dessin, et de très nombreux logiciels peuvent répondre à cette attente légitime. On peut distinguer deux cas un peu différents en pratique : la fonction est définie par une ou des formules, ou elle est représentée par un tableau de valeurs. Si le premier cas ne présente aucune difficulté, le deuxième demande que l'on sache faire lire un fichier de données par le logiciel considéré, à moins de devoir entrer toutes les valeurs au clavier. J'explique la marche à suivre, pour quelques logiciels pratiques et faciles d'accès, dans les paragraphes qui suivent.

1.1 Scilab

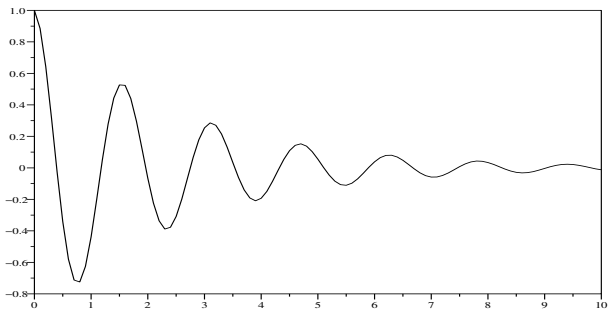
Scilab est un logiciel gratuit, très puissant et disponible sur le site de l'INRIA On peut l'utiliser de façon interactive (comme une calculette) ou préparer, à l'aide d'un éditeur de texte (par exemple celui qui est incorporé dans Scilab depuis la version 2.7), un programme que l'on fera exécuter ensuite.

Supposons que je souhaite tracer une sinusoïde amortie, répondant à l'équation

$$y = \exp(-\alpha x)\cos(\beta x),$$

pour $\alpha = 0.3, \beta = 2$ et $0 \leq x \leq 10$. Les instructions suivantes répondent à mon souhait.

```
deff("y = f(x)", "y = exp(-alfa*x)*cos(beta*x)")
alfa = 0.4; beta = 0.4;
x = 0:0.1:10;
fplot2d(x,f)
```



Le résultat apparaît sur la figure. Ce n'est pas ici le lieu de détailler la syntaxe de Scilab, qui est très bien expliquée dans l'aide en ligne, dans les manuels et sur divers sites (voir les références en fin de chapitre) ; je me contenterais de quelques indications. La première ligne définit une fonction f laquelle dépend d'un seul argument, x ; y est une variable interne à la définition et ne joue aucun rôle dans la suite. Les points-virgules indiquent à Scilab qu'il ne doit pas afficher à l'écran les valeurs qui viennent d'être définies. La troisième ligne initialise un vecteur, x, dont les coordonnées sont $x_1 = 0, x_2 = 0.1, x_3 = 0.2, \ldots, x_{101} = 1$. Vous pouvez constater qu'il se passe pas mal de choses «en douce». Chaque composante du vecteur x est substituée à l'argument formel x de la définition, la valeur de la fonction est calculée et constitue une composante d'un vecteur. Le programme interpole ensuite entre composantes successives pour produire un tracé lisse.

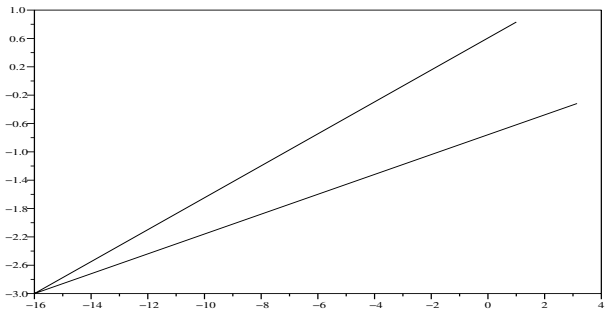
Comment procéder pour tracer la courbe correspondant à un fichier de valeurs numériques ? C'est encore plus simple. Je suppose que le fichier `C:/an_poly/ex112.dta` contient les données qui m'intéressent, à raison de trois par ligne, ces nombres étant séparés par des espaces ou une tabulation :

```
1.0  -2.2       .831
-16 123.456     -3
3.1415      2.718281828    -0.3183098
```

Je procède alors comme suit

```
M = read("C:/an_poly/ex112.dta", -1,3);
plot2d(M(:,1),M(:,3))
```

Scilab admet aussi bien les obliques que les contre-obliques dans les noms de fichiers. à la première ligne, je lis le contenu du fichier et je range les valeurs dans la matrice M. La valeur -1 oblige Scilab à lire toutes les lignes de `C:/an_poly/ex112.dta` quelque soit leur nombre et le point-virgule est là pour l'empêcher d'afficher ces valeurs à l'écran. Je représente ensuite graphiquement tous les nombres de la troisième colonne de M (y) en fonction des valeurs correspondantes de la première colonne (x). Sauf indication contraire, Scilab relie les points par de segments.



1.2 Maple

Le logiciel Maple est orienté vers le calcul algébrique ; il possède cependant des possibilités graphiques extrêmement puisssantes, dont voici un tout petit aperçu. Le dialogue suivant permet de tracer une sinusoïde amortie.

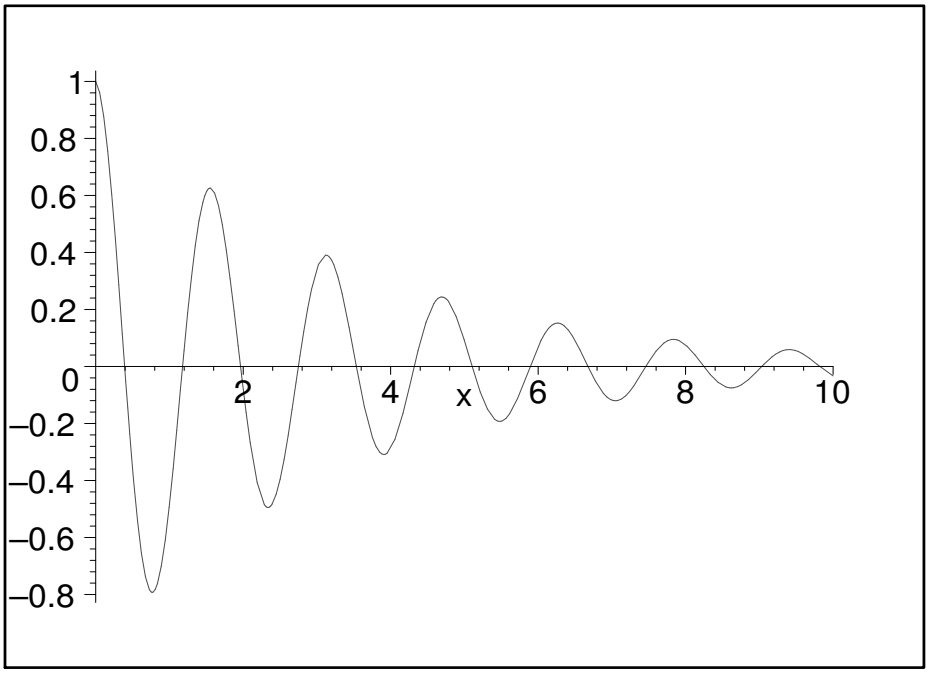
```
> y := exp(-alpha*x)*cos(beta*x);
```

$$y := e^{(-\alpha\, x)} \cos(\beta\, x)$$

```
> y1 := subs(alpha = 0.3, beta = 4,y);
```

$$y1 := e^{(-.3\, x)} \cos(4\, x)$$

```
> plot(y1,x = 0..10);
```



J'aurais pu me contenter de l'instruction unique

> plot(exp(-0.3*x)*cos(4*x),x = 0..10) ;

mais la version précédente me permet de définir une quantité y dépendant de deux paramètres dont je peux modifier la valeur aisément. Le symbole % signifie pour Maple «expression précédente» (c'est ¨ qui joue ce rôle pour la version V4).

La lecture d'un tableau de valeurs externe se fait par l'instruction «readdata» qui admet deux paramètres obligatoires, le nom du fichier et le nombre de colonnes. On peut aussi préciser s'il s'agit d'entiers ou de nombres fractionnaires. La manoeuvre est simple dans le cas d'un tableau à deux colonnes, comme le montre l'exemple ci-dessous. J'ai créé le fichier dont le nom complet est `\"C:\an_poly\ex122.dta\"` et dont le contenu est

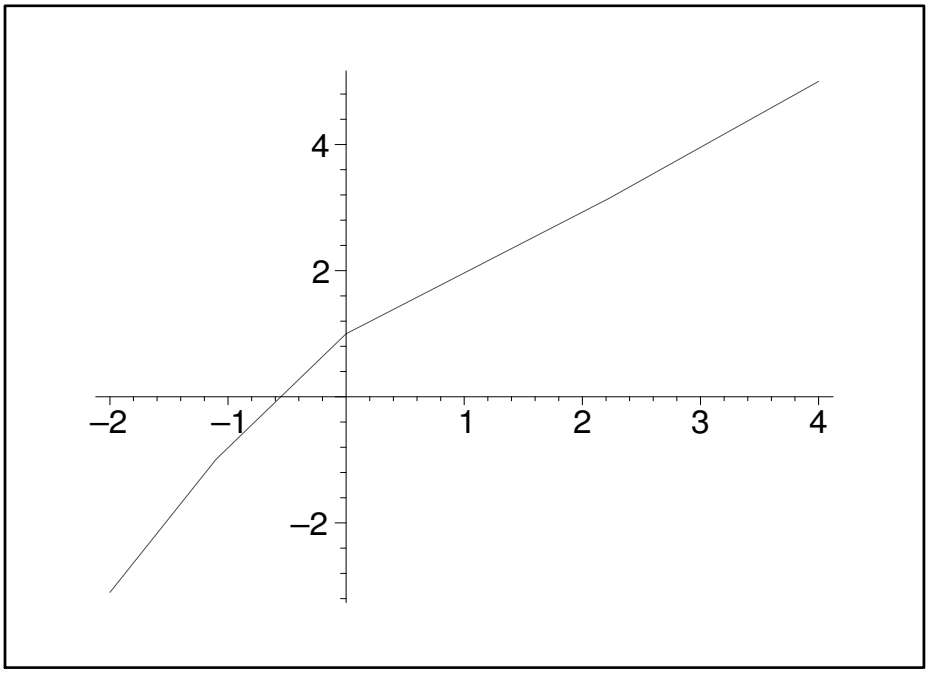
```
-2   -3.1
   -1.1     -.99
0   1
2.222 3.1415
4       5
```

Je trace le graphe correspondant à l'aide des instructions

```
> M := readdata("C:/an_poly/ex122.dta",2);
```

$$M := [[-2.,\ -3.1],\ [-1.1,\ -.99],\ [0.,\ 1.],\ [2.222,\ 3.1415],\ [4.,\ 5.]]$$

```
> plot(M);
```



Les nombres entiers ont été transformés en nombres fractionnaires. Remarquez aussi que les contre-obliques du nom de fichier sont remplacées par des obliques sous Maple. Il est un peu plus compliqué d'extraire d'un fichier une colonne d'abscisses et une colonne d'ordonnées. Il faut procéder comme ceci. Le fichier dont le nom DOS complet est C :\an_poly \ex123.dta contient les valeurs :

```
1.0  -2.2    .831
-16 123.456    -3
3.1415  2.718281828   -0.3183098
```

L'instruction `> M := readdata("C:/an_poly/ex123.dta",3);` lira ces données et produira le résultat

```
M := [[1.0, -2.2, .831],[-16., 123.456, -3.],[3.1415, 2.718281828, -.3183098]]
```

Maple considère M comme une liste de listes, chaque liste représentant une ligne. Le code ci-dessous extrait les nombres de la première colonne (abscisses) et ceux de la troisième colonne(ordonnées), par exemple, puis reporte les points sur un graphique.

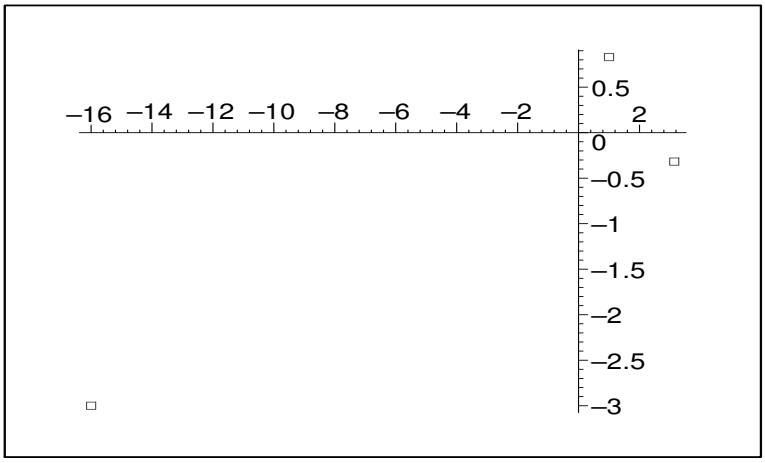
```
> restart;
> with(plots):
> M := readdata("C:/an_poly/ex112.dta",3);
```

$$M := [[1.0, -2.2, .831], [-16., 123.456, -3.], [3.1415, 2.718281828, -.3183098]]$$

```
> points := [seq([M[i,1],M[i,3]],i=1..3)];
```

$$points := [[1.0, .831], [-16., -3.], [3.1415, -.3183098]]$$

```
> plot(points,style=POINT,symbol=BOX, thickness=2,symbolsize=20);
```



1.3 gnuplot

gnuplot est un autre logiciel gratuit et puissant, disponible pour tous les systèmes d'exploitation ; il a l'avantage d'être assez peu encombrant. Il possède un analyseur syntactique qui connaît la plupart des fonctions élémentaires. Il est très facile d'accomplir avec gnuplot les deux tâches qui me servent d'exemples depuis le début. Pour tracer une fonction définie par une formule :

```
>  alfa = 0.3; beta = 4;
>  plot [0:10] exp(-alfa*x)*cos(beta*x);
```

La fonction «plot» admet comme premier paramètre l'intervalle de variation de la variable indépendante, laquelle doit s'appeler x, par convention.

Pour afficher un fonction définie comme une suite de valeurs (x, y) :

```
 > plot [-2.5:0] [-3.5:1.5] "C:/an_poly/ex142.dta"
```

Ici encore, les obliques remplacent les contre-obliques de MS-DOS.

1.4 Excel

Tous les tableurs comportent des outils graphiques puissants. Il n'est pas nécessaire d'expliquer ici comment on trace une fonction analytique ; s'agissant d'un fichier de données, le seul obstacle mineur est la lecture du fichier. Dans le cas d'une version assez récente d'Excel, par exemple, on clique sur fichier/ouvrir, on indique que l'on s'intéresse à tous les types de fichiers, on choisit le bon fichier dans la liste déroulante et on l'ouvre. Il faut alors répondre aux questions simples de «l'assistant d'importation de textes» qui concernent la disposition des données. Le fichier est ensuite importé dans Excel, ligne par ligne et colonne par colonne ; il faut parfois retoucher certaines lignes ou colonnes si la disposition des nombres est très irrégulière. Pour le tracé, on appele «l'assistant graphique» et on répond à ses questions.

1.5 Logiciels sous Linux

Les utiisateurs du système Linux ont à leur disposition de nombreux autres outils graphiques gratuits. Je citerais la collection de programmes «Plotutils» (qui se lancent depuis la ligne de commande), la bibliothèque «pgplot» concue pour s'interfacer facilement avec des programmes en Fortran et enfin le somptueux «xmgrace» interactif.

2 Calcul de Fonctions

Sachant qu'un ordinateur ne connaît que les 4 opérations de l'arithmétique (addition, soustraction, multiplication et division), comment doit-on s'y prendre pour calculer les valeurs de $\sqrt{x}$, $\cos x$ ou $J_3(x)$? On doit faire appel à un algorithme et il existe des algorithmes de calcul pour chaque fonction. Certains datant de l'antiquité (pour le calcul des racines carrées), d'autres n'ont que quelques années d'existence. L'utilisateur devrait donc, en principe, écrire un sous-programme pour calculer la ou les fonctions qui l'intéresse. Une partie du travail est déja fait ; depuis quelques années, les microprocesseurs incorporent un opérateur mathématique capable de calculer vite et bien les fonctions élémentaires. Les compilateurs comportent également des sous-programmes de calcul de fonctions, plus ou moins nombreux selon le compilateur. Cependant, on rencontre souvent des «fonctions spéciales» telles que les fonctions de Bessel, les polynômes de Legendre ou les fonctions elliptiques pour lesquelles il n'existe pas de programme immédiatement disponible. On peut alors rechercher le programme convenable dans les livres ou dans les bibliothèques (voir le «GAMS», Guide to available mathematical software) ou écrire un programme soi-même.

J'estime qu'il est extrêmement intructif et utile d'apprendre à programmer le calcul d'une fonction : on doit découvrir le «bon» algorithme, on apprend à se défier des erreurs d'arrondi ou de troncation, enfin on s'entraine à rédiger un programme correct. Dans tous les cas simples, la vérification du programme est immédiate, par comparaison avec le résultat fourni par le compilateur ou la calculette.

Une catégorie de fonctions se calcule exactement en utilisant uniquement des opérations arithmétiques, ce sont les polynômes et les fractions rationnelles. Ils font l'objet du prochain paragraphe.

2.1 Polynômes et fractions rationnelles

Un polynôme sous forme générale s'écrit

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} + a_n x^n.$$

Les coefficients a_k et la variable x sont réels. S'il n'y a aucune difficulté pour calculer la valeur numérique de $p(x_0)$, pour la valeur $x = x_0$ de la variable, je peux me demander quelle est la façon la plus rapide de parvenir au résultat.

La méthode «naïve» consiste à calculer séparément la valeur de chaque puissance de x_0, puis à multiplier chaque quantité x_0^k par le coefficient a_k correspondant, puis à faire la somme des résultats intermédiaires. Pour obtenir $a_k x_0^k$, je dois faire k multiplications ($n \geq k \geq 1$) et donc $n(n+1)/2$ multiplications en tout ; il s'y ajoute n additions. L'algorithme naïf demande donc un nombre d'opérations qui varie comme n^2 ; on dit qu'il présente une complexité d'ordre n^2.

Un méthode plus économique est fondée sur la remarque que le calcul de x_0^k peut se faire simplement à partir de la valeur de x_0^{k-1} : $x_0^k = x_0 \star x_0^{k-1}$. On calcule l'ensemble des $x_0^k, k = 2 \ldots n$ en $n-1$ multiplications et on garde en mémoire tous les résultats. Il faut ensuite multiplier chaque puissance de x par le bon coefficient ($n-1$ opérations) et faire la somme. Vous vérifierez sans peine que le nombre total d'opérations varie comme $3n$.

L'algorithme de Horner perfectionne le méthode précédente. Il est fondé sur le fait que le polynôme peut s'écrire

$$p(x) = ((a_n \star x + a_{n-1}) + a_{n-2}) \star x + \cdots + a_1) \star x + a_0.$$

Sous cette forme, le calcul de la valeur numérique nécessite n multiplications et n additions, soit une «complexité» d'ordre $2n$. Comment programmer pratiquement ce calcul ? Le polynôme est représenté en mémoire par le vecteur de ses coefficients, $a = [a_0, a_1, \ldots, a_n]$. Si le degré est élevé, il est fastidieux d'écrire en entier la formule précédente, il vaut mieux utiliser la récurrence

$$z_0 = a_n \quad ; \quad z_k = x z_{k-1} + a_{n-k}, \quad k = 1, 2, \ldots, n.$$

On peut faire ce calcul à la main, en utilisant une disposition comme ci-dessous ; il s'agit de calculer la valeur de $x^5 + 2x^3 - 3x^2 + 4x - 1$ pour $x = 2$.

1	0	2	-3	4	-1
1	2	6	9	22	43

Le premier élément de la deuxième ligne est z_0, il est égal à a_n ; le deuxième élément (z_1) vaut $2z_0 + a_{n-1} = 2z_0 = 2$, le troisième est $z_2 = 2z_1 + a_{n-2} = 2 \star 2 + 2 = 6$. La valeur de polynôme est 43. Vous voyez qu'il faut tenir compte des coefficients nuls.

Une fraction rationnelle est le quotient de deux polynômes ; il suffit de calculer comme précédemment le numérateur et le dénominateur.

2.2 Relations de récurrence

Certaines fonctions utiles obéissent à des relations de récurrence, le plus souvent à trois termes. La fonction cosinus en fournit un exemple élémentaire :

$$\cos(k+1)x = 2\cos kx \cos x - cos(k-1)x.$$

On peut utiliser cette relation, avec x «assez petit» pour établir rapidement une table des valeurs de $\cos kx$.

Exemple. Les polynômes de Legendre obéissent à la relation de récurrence

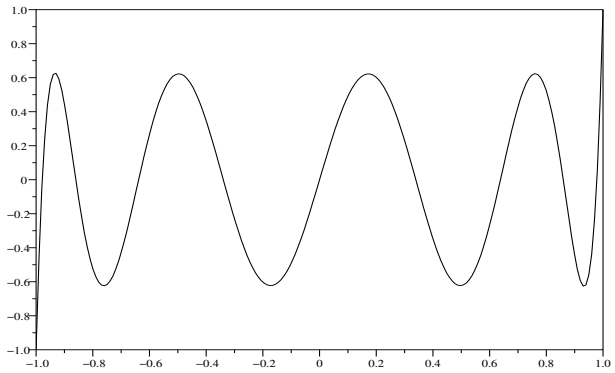
$$(2n+1)xP_n(x) = (n+1)P_{n-1}(x) + nP_{n-2}(x)$$

qui permet un calcul rapide et précis de $P_n(x)$ pourvu que l'on connaisse $P_0 = 1$ et $P_1(x) = x$. Les deux petits fichiers qui suivent permettent le calcul puis l'affichage de P_n sous scilab.

```
getf("C:\an_poly\legendre.sce");
n = input("degre du polynome: ");
x = linspace(-1,1,200);
y = poleg(n,x);
xset("window",0),xbasc(0)
//xtitle("polynome de Legendre d ordre " + string(n))
plot2d(x,y)
```

Remarquez l'instruction `pp = ones(x)` qui crée un vecteur de la même taille que x et dont toutes les composantes sont égales à 1. La construction «.*» permet de faire le produit **composante par composante** de deux vecteurs : si $a = \{a_i\}$ et $b = \{b_i\}$, alors $a. * b = \{a_i b_i\}$. On effectuerait de façon analogue une division ou une élévation à la puissance. Ce type de calcul est beaucoup plus rapide qu'une boucle `for`. La figure montre le résultat dans le cas $n = 9$.

```
function y = poleg(n,x)
//polynome de Legendre
select n
  case 0 then pp = ones(x);
  case 1 then pp = x
  else
      pavder = ones(x), pder = x; i = 1;
      while i < n
        pp = ( (2*n+1)*x.*pder - n*pavder  )/(n+1);
        pavder = pder; pder = pp;
        i = i + 1;
      end
    end
y = pp;
endfunction
```



2.3 Développement limité

La grande majorité des fonctions que l'on rencontre en sciences physiques admet un développement en série; il est donc tentant d'utiliser un développement en sérbie tronqué (un polynôme) pour le calcul numérique d'une telle fonction.

Dans les paragraphes précédents, il n'y avait aucune approximation; si les résultats n'étaient pas tout à fait exacts, cela était du aux erreurs d'arrondi, sur lesquelles je reviendrai à la fin de ce chapitre. L'utilisation d'un développement en série tronqué au terme de rang n fait apparaitre une autre source d'erreur, liée à la méthode utilisée elle-même : au lieu d'une série infinie, je manipule un polynôme à n termes, commettent ainsi ce que l'on appelle une erreur de troncation(ou erreur de méthode). Cette erreur est souvent assez facile à borner, à condition de savoir que la variable indépendante est contenue dans un intervalle défini.

Pour montrer les avantages et inconvénients de cette approche, je choisis l'exemple simple de la fonction e^{-x}, que je voudrais approcher par son développement en série tronqué, avec une erreur absolue inférieure au millième, sur l'intervalle [0,10]. Le terme général du développement est $u_k = (-1)^k \frac{x^k}{k!}$. La série est absolument convergente quelque soit x, ce qui, malheureusement, n'est pas synonyme de rapidement convergente. Je peux estimer le nombre de termes nécessaires à partir du rapport $|u_{k+1}/u_k| = x/(k+1)$. Ce rapport est plus grand que un (les termes de la série sont croissant en valeur absolue) tant que $k+1 < |x|$. Cela implique qu'au bord de l'intervalle choisi, il

faudra bien plus de 10 termes pour approcher la fonction exponentielle. En fait, pour $x = 10$, j'ai trouvé $v_9 \simeq -2755,7$ et $v_{10} \simeq 2755,7$. Plus ennuyeux encore, sachant que $e^3 \simeq 20$, j'estime que $e^{-10} \simeq 5\,10^{-5}$. Ceci signifie que pour atteindre une précision absolue du millième, le premier terme négligé devra être inférieur à $5\,10^{-8}$. Le premier terme qui réponde à ce critère est le 39ième ; on trouve alors $e^{-10} \simeq 0,0000454$. Il faut remarquer que chacun des 39 termes doit être calculé avec une précision absolue de 510^{-8}, soit 12 (douze) chiffres significatifs pour les plus grands d'entre eux, sous peine de perdre toute précision à cause des erreurs d'arrondi pendant l'addition de termes de signes différents. Cet exemple est certes caricatural ; en pratique, on s'arrangerait pour réduire l'intervalle de définition de x. Il montre cependant que le calcul à l'aide de développements tronqués doit faire l'objet d'une attention certaine.

Sans me laisser abattre par les remarques précédentes, j'ai rédigé un programme de calcul de e^{-x} sous Scilab, que je reproduis ci-dessous. Je calcule chaque terme à partir du précédent (ligne 6), en

```
//mon_exp, decroissante.                                    1
eps = 1E-8; kmax = 40;                                      2
terme = 1; somme = 1;k = 1;                                 3
x = input("valeur de x: ");                                 4
while (k<kmax) & (abs(terme) > eps)                         5
    terme = -x*terme/k;                                     6
    somme = somme + terme;                                  7
    [res] = [k,terme,somme];                                8
    write(%io(2),res);                                      9
    k = k+1;                                                10
end                                                         11
somme, exp(-x)                                              12
```

évitant soigneusement de former des factorielles ou des puissances de x : ce serait beaucoup plus long et surtout les résultats intermédiaires déborderaient de la capacité de l'ordinateur. Les lignes (8,9) affichent, de façon plutôt malcommode mais nécessaire sous Scilab, des résultats intermédiaires.

2.4 Fraction continue

L'objet mathématique représenté ci-dessous est une «fraction continue» :

$$f = b_0 + \cfrac{a_1}{b_1 + \cfrac{a_2}{b_2 + \cfrac{a_3}{b_3 + \cfrac{a_4}{b_4 + \cdots}}}}$$

Pour simplifier le travail des imprimeurs, j'écrirais plutôt

$$f = b_0 + \frac{a_1}{b_1+}\frac{a_2}{b_2+}\frac{a_3}{b_3+}\frac{a_4}{b_4+}\cdots$$

Une expression de ce genre n'a d'intérêt que si les a_i et les b_i sont des constantes ou des fonctions simples de x, comme dans la représentation de la fonction tangente :

$$\operatorname{tg} x = \frac{x}{1-}\frac{x^2}{3-}\frac{x^2}{5-}\frac{x^2}{7-}.$$

Cette expression converge rapidement et nécessite moins d'opérations que le développement en série tronqué de la même fonction. Elle souffre cependant d'un inconvénient pratique sérieux : il n'y a pas moyen de déterminer a priori le nombre de termes à conserver pour obtenir une précision fixée à l'avance. Il existe en fait un algorithme itératif qui permet le calcul de f «de gauche à droite»

à partir des a_i et des b_i et que l'on peut interrompre lorsque deux approximations successives ont suffisamment proches (algorithme de Clenshaw). Sans faire appel à ce perfectionnement, on peut apprécier ce type d'approximation sur l'exemple de tg x ; si je tronque la fraction continue précédente après le terme $x^2/9$, j'obtient, après réduction au même dénominateur

$$\operatorname{tg} x = \frac{x}{15}\frac{943 - 105x^2 + x^4}{63 - 28x^2 + x^3}.$$

Les premiers zéros de numérateur sont $\pm 1{,}57080$ et ceux du dénominateur $\pm 3{,}153$, ce qui est proche des valeurs exactes.

En fait, une fonction comme tg x, avec ses asymptotes verticales, a un comportement très éloigné de celui d'un polynôme ; il sera donc malaisé de trouver un développement limité convenable. Au contraire, une fraction rationnelle peut présenter des branches asymtotiques et pourra plus facilement approcher tg x.

2.5 Approximant de Padé

D'après la remarque faite à la fin du paragraphe précédent, j'aurais intérêt, pour approcher une fonction qui n'a pas un comportement polynomial, à essayer une fraction rationnelle. Si la théorie générale de telles approximations existe, je ne la décrirais pas, mais je présenterais un cas particulier, connu sous le nom d'approximant de Padé. On peut décrire un approximant de Padé comme une fraction rationnelle qui obéit à des contraintes très semblables à celles d'un développement en série.

Dans la suite, je considère une fraction rationnelle $R_{m,k}$, quotient d'un polynôme P_m (le numérateur), de degré m et de coefficients $a_j, 0 \leq j \leq m$ par un polynôme Q_m (le dénominateur), de degré n et de coefficients $b_j, 0 \leq j \leq k$. Comme je peux, sans changer la valeur de la fraction, diviser haut et bas par une même constante, j'impose, sans restreindre la généralité, la condition $b_0 = 1$. La fraction $R_{m,k}$ contient donc $m + k + 1$ coefficients à déterminer.

$R_{m,k}$ sera, par définition, l'approximant de Padé d'ordre $n+1$ de la fonction $f(x)$, au voisinage du point $x = x_0$, si $R_{m,k}(x_0)$ et ses n premières dérivées coincident respectivement avec $f(x_0)$ et ses n premières dérivées :

$$[R_{m,k}(x)]^{(p)} \equiv f^{(p)}(x_0), p = 0, 1, 2 \ldots, n.$$

en posant $f^{(0)} = f$. Je peux toujours, à l'aide d'une translation, me ramener au cas $x_0 = 0$, ce que je supposerai dans la suite. Comme je dispose de $m + n + 1$ coefficients inconnus, il me faut, pour déterminer entièrement la fraction rationnelle, autant de conditions ; il faut donc que $m + k = n$. Il est hors de question de calculer les dérivées successives de R et de f pour les identifier, ce qui serait en général impossible ; plus simplement, je vais identifier les développements limités de ces deux expressions, au voisinage de l'origine, jusqu'aux termes de rang $n + k + 1$ (ce qui assure l'égalité des dérivées). On peut économiser encore des calculs en imposant, ce qui revient au même, que le developpemment limité de la quantité $R - f$ commence par un terme en x^{x+k+1}. Je suppose que la fonction f admet un développement de MacLaurin

$$f(x) = \sum_{j=0}^{\infty} c_j x^j.$$

$R - f$ s'écrit

$$\frac{P_m(x) - Q_k(x)f(x)}{Q_k(x)}.$$

Le développement limité de ce rapport est le quotient des développements du numérateur et du dénominateur ; ce dernier ne va pas intervenir pour les premières puissances de x, puisqu'il vaut 1

près de l'origine. Le développement du numérateur doit commencer par le terme en x^{n+1}, ce qui revient à dire que les coefficients des termes en $x^0, x^1 \cdots x^n$ sont tous nuls. J'en déduis les relations

$$\begin{aligned} a_0 &= b_0 c_0 \qquad (x^0), \\ a_1 &= b_0 c_1 + b_1 c_0 \qquad (x^1), \\ a_j &= \sum_{s=0}^{s=j} c_s b_{j-s} \qquad (x^j, j \leq m). \end{aligned}$$

Lorsque tous les coefficients a_j disponibles ont été utilisés, je continue sans eux

$$0 = \sum_{s=0}^{s=j} c_s b_{j-s} \qquad (x^j, j = m+1, m+2 \ldots, n).$$

Il est commode d'utiliser d'abord la dernière série d'équations, plus simples, pour déterminer certains des coefficients b_i, puis la première série pour les coefficients restants.

On a constaté empiriquement que le choix $m = k \pm 1$ était souvent meilleur que d'autres. Le raisonnement précédent ne permet pas d'estimer l'erreur d'approximation, mais, là encore, l'expérience montre que l'approximation selon Padé ets très bonne.

Exemple. Je cherche l'approximant de Padé $R_{2,2}$ de e^x au voisinage de l'origine. Ici, $m = k = 2, n = 4$; je dois donc connaître, pour commencer, les 5 premiers termes du développement de l'exponentielle. Les coefficients correspondants sont

$$c_0 = 1; \quad c_1 = 1; \quad c_2 = \frac{1}{2}; \quad c_3 = \frac{1}{6}; \quad c_4 = \frac{1}{24}.$$

D'autre part, $R_{2,2}$ s'écrit

$$R_{2,2}(x) = \frac{a_0 + a_1 x + a_2 x^2}{1 + b_1 x + b_2 x^2}.$$

Je forme maintenant le numérateur de $R - e^x$ et j'identifie à zéro les coefficients de x^0, x^1, x^2, x^3, x^4, ce qui donne

$$\begin{aligned} a_0 = 1; \qquad a_1 = b_1 + 1; \qquad a_2 = b_1 + b_2 + 1/2; \\ 0 = b_2 + b_1/2 + 1/6; \qquad 0 = b_1/6 + b_2/2 + 1/24. \end{aligned}$$

Je trouve, en réslvant ce système :

$$b_1 = -1/2; \qquad b_2 = 1/12; \qquad a_0 = 1; \qquad a_1 = 1/2; \qquad a_2 = 1/12$$

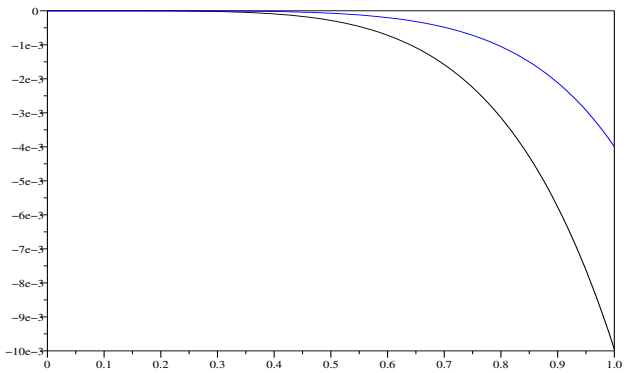
et donc

$$R_{2,2} = \frac{12 + 6x + x^2}{12 - 6x + x^2} = \frac{(x+6)x + 12}{(x-6)x + 12}.$$

Le petit programme Scilab

```
x = linspace(0,1,100);                                          1
p = 1+x+x.^2/2+x.^3/6+x.^4/24;                                  2
r = ((x+6).*x+12)./((x-6).*x+12);                               3
xset("window",0),xbasc(0)                                       4
xtitle("approximations de exp(x)")                              5
plot2d(x',[(p-exp(x))',(r-exp(x))']);                           6
```

a produit le tracé suivant



Vous constatez que pour $x \simeq 1$ l'approximant de Padé est presque trois fois plus précis que le développement de Taylor.

2.6 Approximation de fonction

J'ai présenté, dans les paragraphes précédents, un certain nombre de recettes ou de procédés destinés à fournir des approximations de fonctions, sans aucunement me soucier de rigueur mathématique. En réalité,l'approximation des fonctions est un domaine bien établi des mathématiques, qui a connu un grand développement à partir de la fin du 19ème siècle. Dans les lignes qui suivent, je donne quelques idées générales sur l'approximation, considérée d'un point de vue plus mathématique.

Le problème se pose en ces termes. Je m'intéresse à une fonction $f(x)$ de la variable réelle x. Soit je désire connaître ses valeurs en certains points, soit, plus ambitieux, je voudrais pouvoir calculer les valeurs numériques de sa dérivée ou de son intégrale sur un certain intervalle. Seulement voila, f est compliquée et longue à calculer et je n'ai pas le temps d'accumuler toutes les valeurs nécessaires de f. Je décide alors de remplacer f par une approximation f^* qui se calcule facilement. Pour savoir si cette substitution peut avoir un sens, je dois répondre à plusieurs questions : dans quelle catégorie de fonctions vais-je choisir f^* ? Selon quel critère vais-je m'assurer que l'approximation est «bonne» où «mauvaise» ? Sur quel intervalle les propriétés précédentes doivent-elles être vérifiées ?

Je me limiterai ici au cas de l'approximation polynomiale, c'est-à-dire que f^* sera un polynôme de degré au plus égal à n (une combinaison linéaire de $x^k, k = 0, 1, 2, \ldots, n$). Le théorème suivant, dû à Weierstrass, affirme l'existence de f^* sous des conditions assez générales.

> *Théorème* Si $f(x)$ est continue sur l'intervalle fini $I = [a, b]$ et si ϵ est un nombre positif donné, alors il existe un polynôme $f^*(x)$ tel que
>
> $$\sup_{x \in I} |f(x) - f^*(x)| < \epsilon.$$

Remarquez que le théorème ne donne aucune indication sur la construction de f^*. Ici, la «distance» entre f et f^* est la «norme infinie» ; on pourrait imaginer d'autre types d'approximation où la définition de la «distance» serait différente ; on pourait penser par exemple à l'écart quadratique moyen, qui est défini comme $\| f - f^* \| = \int_I |f - f^*|^2 dx$.

Tschebychef et de la Vallée-Poussin ont prolongé les travaux de Weierstrass. Grâce à eux, on sait qu'un «bon» polynôme f^* est tel que l'erreur d'approximation, $|f - f^*|$, présente une série de maximums assez régulièrement répartis sur I, d'amplitude identique et alternativement positifs et négatifs. Ces polynômes étaient déterminés à l'époque par essais et erreurs, mais on dispose maintenant d'un algorithme puissant d'approximations sucessives (second algorithme de Remes).

2.7 Développement asymptotique

Lorsque la variable indépendante devient très grande, il devient évidemment désespéré d'approcher une fonction par un développement construit au voisinage de 0 ; on peut bien sûr faire une translation d'origine pour chaqe valeur particulière, mais cela complique notablement les calculs. Dans certains cas, on peut avoir recours à une approximation valable lorsque $1/x$ est petit, ce que l'on appelle un développement asymptotique et dont voici la définition.

Soit la série infinie

$$c_0 + \frac{c_1}{x} + \frac{c_1}{x^2} + \cdots$$

J'appelle S_n la somme partielle des n premiers termes

$$S_n = c_0 + \frac{c_1}{x} + \frac{c_1}{x^2} + \cdots + \frac{c_{n-1}}{x^{n-1}}.$$

Habituellement, on s'intéresse à la convergence de la série ou à la limite de la quantité S_n quand n tend vers l'infini. Ici, je procède différemment ; gardant n fixe, je fait croitre x au dela de toute limite. Je suppose qu'il existe une fonction $f(x)$ telle que la différence $|f(x) - S_n(x)|$ tende vers zéro plus vite que $\frac{1}{x^{n-1}}$ quand $x \longrightarrow \infty$. En d'autres termes

$$\lim_{x \longrightarrow \infty} x^{n-1} \, |f(x) - S_n(x)| = 0.$$

Si ces conditions sont remplies, je dis que la série est un développement asymptotique de la fonction f.

Exemple. Je considère la fonction définie pour $x > 0$ par l'intégrale

$$f(x) = \int_x^\infty \frac{1}{t} e^{x-t} dt$$

Des intégrations par partie répétées me permettent de transformer cette expression en

$$f(x) = \frac{1}{x} - \frac{1}{x^2} + \frac{2}{x^3} + \cdots + \frac{(-1)^{n-1}(n-1)!}{x^n} + (-1)^n n! \int_x^\infty \frac{e^{x-t}}{t^{n+1}} dt.$$

Ceci me laisse supposer que la série

$$\frac{1}{x} - \frac{1}{x^2} + \frac{2!}{x^3} - \frac{4!}{x^4} \cdots$$

pourrait représenter le développement asymptotique de f. Pour le prouver, je forme

$$f(x) - S_{n+1}(x) = (-1)^n n! \int_x^\infty \frac{e^{x-t}}{t^{n+1}} dt.$$

Dans l'intégrale, l'exponentielle est comprise entre 0 et 1, d'où la majoration

$$|f(x) - S_{n+1}(x)| < n! \int_x^\infty \frac{1}{t^{n+1}} dt = (n-1)! \frac{1}{x^n},$$

une quantité qui tend évidemment vers zéro lorsque x croit, à n constant. On écrit souvent

$$\int_x^\infty \frac{1}{t} e^{x-t} dt \cong \frac{1}{x} - \frac{1}{x^2} + \frac{2!}{x^3} - \frac{4!}{x^4} \cdots.$$

3 Représentation des nombres en machine

L'arithmétique des ordinateurs présente des propriétés plus compliquées qu'il n'y parait à première vue. Ces complications sont dues à ce qu'un ordinateur (tout comme un humain) ne manipule qu'un nombre limité de chiffres ; il s'en suit que beaucoup de nombres concevables ne sont pas ou sont mal représentés en machine. Les choses se présentent différemment pour les entiers et pour les nombres fractionnaires. Je commence par examiner les entiers.

3.1 Les nombres entiers

J'ai choisi, pour fixer les idées, le compilateur TurboPascal de Borland, mais des considérations presque identiques s'appliquent à tous les compilateurs. TurboPascal définit cinq types d'entiers, dont je reproduit ci-dessous les caractéristiques

types entiers	représentation (nombre d'octets)	intervalle de définition
byte	1	0 :255
shortint	1	-128 :127
integer	2	-32768 :32767
word	2	0 :65535
longint	4	-2147483648 :2147483647

Dans tous les cas, le chiffre binaire le plus à gauche (le plus significatif, de poids le plus grand) indique le signe : 0 pour les nombres positifs, 1 pour les entiers négatifs.

Ainsi, dans le type «shortint», le plus grand nombre positif s'écrit-il 01111111 ; il vaut $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 = 127$. Les nombres négatifs sont représentés selon la méthode dite «du complément à deux», qui prescrit que, si x est négatif (et plus petit en valeur absolue que 2^7, sa représentation, que je note (x), est $x + 2^8$. La recette suivante permet de construire facilement $(-x)$ à partir de (x) : changer tous les «1» en «0» et réciproquement, sauf pour le chiffre le plus à droite. Si $x = 23, (x) = 00010111$ et $(-23) = 11101001$. Vous pouvez vérifier, en appliquant les règles de l'arithmétique binaire, que $(23)+(23) = 100000000$; la dernière retenue est perdue, puisque les nombres n'ont que 8 chiffres.

L'ordinateur ne prévient pas lorsque le résultat d'une addition entre entiers est trop grand. On a par exemple (127)+ (3) = 01111111 + 00000011 = 10000010 = (-124). Cette représentation fonctionne un peu comme un compteur kilométrique de voiture.

La conséquence pratique est que l'amplitude des entiers est très limitée (sauf pour longint). Un piège classique dans lequel chacun se fait prendre au moins une fois est le calcul de la factorielle. Avec le type int, 7 ! est calculé correctement, 8 ! est trouvé négatif. Le problème ne se pose pas avec Scilab qui ne connaît que des nombres fractionnairees, ni avec Maple, qui peut manipuler autant de chiffres que la mémoire de l'ordinateur le permet.

3.2 Nombres fractionnaires

Les nombres décimaux (on dit aussi fractionnaires, flottants, à virgule flottante) sont représentés par une partie fractionnaire et un exposant, comme par exemple 23 $\longrightarrow$ 0,23E2. Le nombre précédent peut aussi bien s'écrire 23E0 ou 0,023E3. Pour éviter ces ambiguités, on invoque une convention de normalisation : la partie fractionnaire devra être comprise entre 0 et 1 par exemple.

Les nombres décimaux définis en Pascal peuvent appartenir à l'un des 4 types ci-dessous.

type	nb équivalent de chiffres décimaux	intervalle de définition	taille (octets)
real	11-12	$2{,}910^{-39}$:1.710^{38}	6
single	7-8	$1{,}510^{-45}$:3.410^{38}	4
double	15-16	510^{-324} :1.710^{308}	8
extended	19-20	1.910^{-4551} :1.1 10^{4932}	10

Il est clair que l'intervalle de définition (ou le nombre de chiffres significatifs) dépend du nombre de chiffres binaires affectés respectivement à la représentation de la partie fractionnaire et de l'exposant. Les 32 chiffres du type single sont répartis ainsi : 1(signe)+ 8 (exposant) + 23(partie fractionnaire). Sous Scilab, les nombres sont uniformément du type double.

Les nombres compris entre 0 et la borne inférieure de l'intervalle de définition, de même que ceux plus grands que la borne supérieure, ne pourront pas être représentés. Tout calcul produisant un nombre répondant à l'une de ces conditions provoquera en principe un message d'erreur ; en fait, les nombres trop petits sont pris égaux à zéro, ce qui **peut** être sans conséquence ; les nombres trop grands interrompent l'exécution du programme et l'apparition du message d'erreur «arithmetic overflow».

Même si un nombre a une taille correcte, son calcul est souvent souvent entaché d'une erreur d'arrondi. Je suppose que j'utilise un ordinateur fictif fonctionnant en numération décimale. Je forme 27/13,1 = 2,06106870229..., un nombre fractionnaire dont la représentation décimale ne se termine jamais et que l'unité centrale calcule avec 20 chiffres significatifs. Si j'ai déclaré que la variable correspondante était du type «single», lorsque l'ordinateur range cette valeur en mémoire, il le fait selon la norme du type, soit avec 7 chiffres. Sur certaines machines on conserve la valeur 2,061068 (troncation), sur d'autres, on retient 2,061069 (arrondi). L'erreur est ici minime (inférieure à 10^{-7}en valeur absolue), mais il faut être conscient qu'elle peut se répéter des milliards de fois au cours d'un calcul. Heureusement, le signe de chaque erreur est pratiquement aléatoire, si bien que celles-ci ne s'ajoutent pas de façon cohérente.

L'erreur relative devient importante lorque l'on forme la différence de deux nombres voisins. On ne remarque rien en général, sauf si, par hasard, ce premier résultat est immédiatement multiplié par un nombre très grand.

Je remarque enfin que l'utilisation, en Pascal, des types «single» et «real», correspond à une économie de mémoire, mais à une perte de temps. Il en est de même pour le type «float» en C. En effet, l'unité centrale calcule avec une très grande précision (80 chiffres binaires pour les Pentiums) ; les résultats sont ensuite mis aux normes avant d'être rangés en mémoire et cette transformation prend du temps (une fraction de microseconde). Du point de vue de la vitesse d'exécution, il vaut mieux travailler avec le type «double», qui est très proche du format utilisé par l'unité centrale.

4 pour en savoir plus

- Scilab :
  pauillac.inria.fr/cdrom/www/scilab/fra.htm
  ftp ://ftp.inria.fr/INRIA/Projects/Meta2/Scilab/distributions/
  http ://www.iecn.u-nancy.fr/ pincon/
  http ://www.dma.ens.fr/ ldumas/aide.html
  www.math-info.univ-paris5.fr/Enseignements/demarre_scilab/demarre_scilab.html
- Maple :
  http ://www.maplesoft.com/
  lumimath.univ-mrs.fr/ jlm/cours/maple/maple.html
  http ://algo.inria.fr/dumas/Maple/
  http ://perso.wanadoo.fr/eddie.saudrais/
  http ://www-math.math.rwth-aachen.de/MapleAnswers/

- gnuplot :
  http ://www.gnuplot.info/
  ftp ://ftp.irisa.fr/pub/gnuplot/
- Grace :
  http ://plasma-gate.weizmann.ac.il/Grace/
- plotutils :
  www.gnu.org/software/plotutils/
- pgplot :
  http ://www.astro.caltech.edu/ tjp/pgplot/
- FAQ du forum sci.math.num-analysis :
  http ://www.mathcom.com/corpdir/techinfo.mdir/scifaq/index.html
- GAMS : http ://gams.nist.org
- histoire des maths, définitions :
  http ://www.sciences-en-ligne.com/momo/chronomath/accueil.htm