Mathématiques - Compte-rendu du TP 2

Dubois Adrien Sudron Pierre

1er février 2010

1 Principe du calcul approximatif d'une fonction avec les séries de Taylor

Tenter de recréer un calculateur sur une machine demande de réfléchir sur la façon dont on réalise l'approximation de certaines fonctions particulières dont on ne connait pas toujours l'expression numérique exacte.
Notre objectif ici est d'utiliser les séries de Taylor associées à certaines fonctions comme l'exponentielle, le logarithme et les fonctions trigonométriques, afin de déterminer la valeur de ces fonctions.
Les calculs seront réalisés grâce au logiciel Scilab.

En effet, pour une fonction f de classe C^∞ définie sur $\mathbb{R}$, au voisinage d'un point a réel, on peut exprimer $f(x)$ par la série suivante, nommée série de Taylor :

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

On peut également la noter sous la forme :

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Cette série fait partie des séries entières. Son rayon de convergence r dépend de la fonction f. La série est donc convergente pour toute valeur de x comprise entre a et r.
Pour nos approximations, nous prendrons $a = 0$.

Ces séries nous permettent alors de calculer une approximation d'une valeur de f, pour peu que l'on reste à l'intérieur du cercle de convergence. En effet, comme la série est convergente, ses termes tendent vers 0. Chaque terme est de plus en plus petit, et ainsi, plus on prend en compte un nombre important de termes, plus on se rapproche de la valeur exacte de $f(x)$.

C'est l'idée de notre fonction d'approximation : on calcule une somme partielle de la série de Taylor associée à f. Plus le nombre de termes pris en compte est important, plus on s'approchera de la valeur exacte de la fonction.

Il reste à choisir le nombre de termes inclus dans la somme partielle. Deux approches sont possibles. On peut par exemple d'un nombre N fixé de termes,

ce qui peut être intéressant si on souhaite contrôler le temps et les ressources qui seront employés pour réaliser le calcul.
Il est aussi possible de mettre en paramètre du calcul la précision souhaitée. En effet, chaque terme de la somme partielle est plus petit que le précédent (pour peu que l'on soit à l'intérieur du rayon de convergence). Ainsi, on peut poursuivre le développement de la série de Taylor jusqu'à ce que le dernier terme ajouté ait une valeur inférieure à un seuil ε fixé.
N'étant pas limités par le temps et le matériel, c'est cette seconde méthode qui a été adoptée.

2 Formules et contraintes des séries de Taylor utilisées

Il est demandé de réaliser des approximations pour 5 valeur particulières : $e^{1.37}$, $ln(1.338)$, $sin(0.21466)$, $cos(0.21466)$, $\sqrt[3]{1.34}$ et $(1.2)^{1.3}$. La première étape est donc de définir quelles sont les séries de Taylor correspondant à ces fonctions.

La fonction exponentielle

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Rayon de convergence : ∞

La fonction logarithme

$$\ln(1+x) = \sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n$$

Rayon de convergence : 1

On remarquera que cette série s'applique sur $(1 + x)$, on prendra donc soin dans le programme de retrancher 1 à la valeur de x. Par ailleurs, en raison du rayon de convergence $r = 1$, on ne pourra pas calculer d'approximation pour des valeurs x supérieures à 2 (la série étant divergente, le programme bouclera).

La fonction sinus

$$sin(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1}$$

Rayon de convergence : ∞

La fonction cosinus

$$cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n}$$

Rayon de convergence : ∞

La fonction binomiale

$$(1+x)^{\alpha} = 1 + \sum_{n=0}^{\infty} \frac{\alpha(\alpha-1)(\alpha-2)\cdots(\alpha-(n-1))}{n!} x^n$$

avec $\alpha \in \mathbb{R}$

Rayon de convergence : 1

On peut faire ici les mêmes remarques que pour la fonction logarithme : on doit retrancher 1 à la valeur de x et il ne sera pas possible de faire d'approximation correcte pour une valeur de x supérieure à 2.
Cette formule permettra à la fois le calcul de $\sqrt[3]{1.34}$ (pour $\alpha = \frac{1}{3}$) et de $(1.2)^{1.3}$.

3 Résultats et appréciation de la marge d'erreur

L'objet de cette partie est de comparer les résultats de nos fonctions pour différentes valeurs de la précision ε avec les valeurs données par les fonctions de calcul intégrées à Scilab.
Pour faciliter la lecture, les chiffres différents de ceux de la fonction de référence dans Scilab ont été soulignés.

Calcul	Fonction Scilab	Approx. $\varepsilon = 0.1$	$\varepsilon = 0.001$	$\varepsilon = 0.0001$
$e^{1.37}$	3.9353507	3.9240083	3.9352965	3.9353434
$sin(0.21466)$	0.2130152	0.2130114	0.2130152	0.2130152
$cos(0.21466)$	0.9770489	0.9769605	0.9770490	0.9770490
$ln(1.338)$	0.2911760	0.280878	0.2913689	0.2911923
$\sqrt[3]{1.34}$	1.1024738	1.1004889	1.1023651	1.102466
$(1.2)^{1.3}$	1.267464	1.2678	1.267436	1.2674669

Si on prend pour référence le résultat donné pas Scilab, les résultats de nos approximations sont proches de ce qu'ils devraient être. Très logiquement, la précision s'améliore sensiblement avec un seuil ε plus faible. Pour la fonction sin, on retrouve même un résultat "exact" avec un ε égal à 0.001.

Tout ceci nous montre l'efficacité des séries de Taylor pour l'approximation de certaines fonctions, qui fournissent de bon résultats pour un nombre raisonnable de calculs. Il faut néanmoins vérifier que la valeur à calculer se trouve bien dans le rayon de convergence de la série à utiliser.

4 Annexe : code Scilab du TP

```
// Factorielle
function result = factorielle (rang )

        result = 1;
        for i = 1 : 1 : rang
                result = result * i ;
        end

endfunction

// Rang n de la serie de la fonction exp
function result = serie_exp_rang(x, n)

        result = x^n / factorielle(n);

endfunction

// Approximation de la fonction exp
function result =  approx_exp(x, epsilon)

        n = 0;
        tmp = serie_exp_rang(x, n);
        result = tmp;

        while tmp>epsilon
                n = n + 1;
                tmp = serie_exp_rang(x, n);
                result = result + tmp;
        end

endfunction

//=============================================================

// Rang n de la serie de la fonction sin
function result = serie_sin_rang(x, n)

        result = (-1)^n*x^(2*n + 1) / factorielle(2*n + 1);

endfunction

// Approximation de la fonction sin
function result =  approx_sin(x, epsilon)

        n = 0;
        tmp = serie_sin_rang(x, n);
        result = tmp;
```

```
        while abs(tmp)>epsilon
                n = n + 1;
                tmp = serie_sin_rang(x, n);
                result = result + tmp;
        end

endfunction

//=========================================================

// Rang n de la serie de la fonction cos
function result = serie_cos_rang(x, n)

        result = (-1)^n*x^(2*n) / factorielle(2*n);

endfunction

// Approximation de la fonction cos
function result =  approx_cos(x, epsilon)

        n = 0;
        tmp = serie_cos_rang(x, n);
        result = tmp;

        while abs(tmp)>epsilon
                n = n + 1;
                tmp = serie_cos_rang(x, n);
                result = result + tmp;
        end

endfunction

//=========================================================

// Rang n de la serie de la fonction ln(1+x)
function result = serie_ln_rang(x, n)

        result =  (-1)^(n+1) * x^n / n;

endfunction

// Approximation de la fonction ln(x)
function result =  approx_ln(x, epsilon)

        x=x-1;
        n = 1;
        tmp = serie_ln_rang(x, n);
        result = tmp;
```

```
        while abs(tmp)>epsilon
                n = n + 1;
                tmp = serie_ln_rang(x, n);
                result = result + tmp;
        end

endfunction

//=============================================================


function result = numerateur(a, n)

        result = a;
        for i = 1 : 1 : (n-1)
                result = result * (a-i);
        end

endfunction


// Rang n de la serie de la fonction racine cubique
function result = serie_rac3_rang(x, n)

        result = (numerateur((1/3), n)/factorielle(n)) * x^n ;

endfunction

// Approximation de la fonction racine cubique
function result =  approx_rac3(x, epsilon)

        x=x-1;
        n = 1;
        tmp = serie_rac3_rang(x, n);
        result = 1 + tmp;

        while abs(tmp)>epsilon
                n = n + 1;
                tmp = serie_rac3_rang(x, n);
                result = result + tmp;
        end

endfunction

//=============================================================

// Rang n de la serie de la fonction puissance 1.3
function result = serie_pow_rang(x, n)

        result = (numerateur(1.3, n)/factorielle(n)) * x^n ;
```

```
endfunction

// Approximation de la fonction puissance
function result =  approx_puissance(x, epsilon)

        x=x-1;
        n = 1;
        tmp = serie_pow_rang(x, n);
        result = 1 + tmp;

        while abs(tmp)>epsilon
                n = n + 1;
                tmp = serie_pow_rang(x, n);
                result = result + tmp;
        end

endfunction
```