LM203 — Maxima — Laurent Koelblen

TP n° 1 — Sommes de Riemann

On va calculer des approximations d'une intégrale à l'aide des sommes de Riemann : $\int_a^b f(x)\mathrm{d}x \simeq \sum_{i=1}^{n} \frac{b-a}{n} f\left(a + i \times \frac{b-a}{n}\right)$.

Premier exemple : l'intégrale $\int_1^2 \log(x)\mathrm{d}x$ a pour valeur approchée : $\sum_{i=1}^{10} \frac{1}{10} \log\left(1 + \frac{i}{10}\right)$.

Executez avec Maxima les instructions suivantes :

```
a:1;
b:2;
n:10;
```

qui affectent aux variables a, b, n les valeurs respectives 1, 2, 10. La somme de Riemann cherchée s'obtient directement à l'aide de la fonction « `sum` » ; cette fonction prend 4 arguments séparés par des « , » ; le premier est une expression dans laquelle il y a une variable indéfinie (ici « `i` »), le second est le nom de cette variable. Ainsi l'expression « `sum(f(i),i,j,k)` » effectue le calcul de la somme $\sum_{i=j}^{k} f(i)$.

```
r:sum(log(a+(b-a)/n*i)*(b-a)/n,i,1,n);
```

On peut en obtenir une valeur approchée :

```
float(r);
```

qu'on peut comparer à la « vraie » valeur de l'intégrale :

```
I:integrate(log(x),x,1,2)$
float(I);
```

L'approximation est ce qu'elle est...

Pour généraliser la méthode on va écrire une procédure pour le calcul des sommes de Riemann. Les paramètres d'entrée sont : une fonction f, les extrémités de l'intervalle, a et b, et le nombre de pas de la subdivision, n.

Mais tout d'abord, on réinitialise les variables enregistrées :

```
kill(all);
```

Voici la procédure (qui est comprise par Maxima grâce à l'opérateur « `:=` ») :

```
SRiemann(f,a,b,n) := sum(f(a+(b-a)/n*i)*(b-a)/n,i,1,n);
```

On la teste sur quelques exemples :

```
SRiemann(log,1,2,10);
```

On en calcule la valeur approchée (le signe « `%` » remplace le résultat de l'instruction qui vient d'être exécutée) :

```
float(%);
SRiemann(log,1,2,20);
float(%);
```

On calcule les sommes de Riemann $S(10)$, $S(20), \ldots, S(100)$ pour des subdivisions de l'intervalle $[1, 2]$ en 10, 20,...,100 pas :

```
r:makelist(SRiemann(log,1,2,10*n),n,1,10)$
```

le signe « `$` » est là pour indiquer à Maxima de ne pas afficher le résultat ; seules les valeurs approchées nous intéressent :

```
float(r);
```

On compare ces valeurs approchées à la « vraie » valeur de l'intégrale :

```
I:integrate(log(x),x,1,2)$
float(I);
```

On calcule les différences entre ces valeurs approchées et la « vraie » valeur de l'intégrale : ceci se fait grâce à l'usage de deux fonctions :
- La première est la fonction « `map` » : si « `r` » est une liste $[r_1, r_2, \ldots, r_k]$, « `map(f,r)` » retourne la liste $[f(r_1), f(r_2), \ldots, f(r_k)]$.
- La deuxième est la fonction « `lambda` » : « `lambda([i,j],...)` » construit une fonction anonyme qui peut être utilisé à la place d'une fonction définie par « `f(i,j):=...` ». Sitôt utilisée, Maxima l'oublie.

Le résultat des instructions :

```
map(lambda([x],x-I),r)$
float(%);
```

est donc la liste des différences entre les sommes de Riemann $S(10)$, $S(20)$,...,$S(100)$ et l'intégrale $I = \int_1^2 \log(x)\mathrm{d}x$. On remarque qu'en passant de 10 à 100 pas, l'approximation a été divisée par 10. Ceci nous amène à considérer les sommes de Riemann $S(10)$, $S(100)$ et $S(1000)$ pour des subdivisions en 10, 100 et 1000 pas :

```
r:makelist(SRiemann(log,1,2,10^n),n,1,3)$
float(r);
map(lambda([x],x-I),r)$
float(%);
```

Les calculs commencent à être plus longs, l'approximation a encore été divisée pas 10 mais à ce rythme là il va falloir de la patience pour avoir 10 décimales...

Pour faire mieux on va utiliser la méthode dite « d'accélération de convergence » de **Richardson** : on admettra que la suite des sommes de Riemann $S(n)$ (d'une fonction f sur un intervalle $[a, b]$ pour une subdivision en n pas) admet un développement limité en $\dfrac{1}{n}$; C'est à dire qu'il existe de nombres c_1, c_2,...,c_k tels que :

$$S(n) = \int_a^b f(x)\mathrm{d}x + \frac{c_1}{n} + \frac{c_2}{n^2} + \cdots + \frac{c_k}{n^k} + \mathrm{o}\left(\frac{1}{n^k}\right)$$

On va manipuler un peu cette expression avec Maxima. Pour cela, au lieu d'un développement limité en $\frac{1}{n}$ on va utiliser une série (une somme infinie) :

```
S(n):=integrate(f(x),x,a,b)+sum(c(i)/n**i,i,1,inf);
```

On peut afficher les premier termes en prenant un développement limité (au voisinage de l'infini, puisque les puissance de n sont négatives) :

```
taylor(S(n),n,inf,4);
```

On fixe maintenant un entier p et on considère la nouvelle suite $S_1(n,p) = \dfrac{p \times S(p \times n) - S(n)}{p-1}$

```
S1(n,p):=(p*S(p*n)-S(n))/(p-1)$
```

Puis prenant $p = n$, on calcule le développement limité :

```
taylor(S1(n,n),n,inf,4);
```

Que constatez-vous sur les termes du développement limité de $S_1(n,n)$?

On continue dans même direction et on pose $S_2(n,p) = \dfrac{p^2 \times S_1(p \times n, p) - S_1(n,p)}{p^2-1}$:

```
S2(n,p):=(p**2*S1(p*n,n)-S1(p,n))/(p**2-1)$
taylor(S2(n,n),n,inf,7);
```

Que constatez-vous sur les termes du développement limité de $S_2(n,n)$?

Et ainsi de suite : on pose $S_k(n,p) = \dfrac{p^k \times S_{k-1}(p \times n, p) - S_{k-1}(n,p)}{p^k - 1}$

Faites une conjecture sur les termes du développement limité de $S_k(n,n)$. (Au besoin calculer les développements limités de $S_3(n,n)$ et $S_4(n,n)$.)

Dans notre exemple, avec $n = 10$, on calcule $S_1(10,10)$ et $S_1(100,10)$:

```
r1:makelist((10*r[i+1]-r[i])/9,i,1,2)$
float(r1);
map(lambda([x],x-I),r1)$
float(%);
```

puis $S_2(10,10)$:

```
r2:makelist((100*r1[i+1]-r1[i])/99,i,1,1)$
float(r2);
map(lambda([x],x-I),r2)$
float(%);
```

Quel est l'ordre de l'approximation ?

À présent on peut écrire une procédure qui calcule de la sorte la valeur approchée de l'intégrale $\displaystyle\int_a^b f(x)\mathrm{d}x$ par la méthode de Richardson à partir des sommes de Riemann $S(n), S(n^2), \ldots, S(n^k)$:

```
SRichardson(f,a,b,n,k) := block(
    [r,i,j],
    r:makelist(SRiemann(f,a,b,n^i),i,1,k),
    for j from 1 thru k-1 do
        r:makelist((n^j*r[i+1]-r[i])/(n^j-1),i,1,k-j),
    r[1]
    );
```

On teste cette procédure sur notre exemple :

```
SRichardson(log,1,2,10,3)$
float(%);
```

On va comparer les durées de calcul pour différentes valeurs de n et de k. Pour cela on indique à Maxima d'afficher après chaque instruction la durée d'exécution :

```
showtime:true$

I2:SRichardson(log,1,2,2,6)$
float(I2);
float(I2-I);

I3:SRichardson(log,1,2,3,5)$
float(I3);
float(I3-I);

I5:SRichardson(log,1,2,5,4)$
float(I5);
float(I5-I);
```

Comparez les approximations obtenues et les temps de calcul. Quel est le calcul le plus rapide ? Expliquer pourquoi.