T.P. d'analyse numérique

Calculs numériques et symboliques

L. Ducos

Année 2011-2012

Table des matières

Chapitre 1

INTERPOLATION ET APPROXIMATION

Au programme : interpolation de Lagrange et polynômes de Bernstein. Ces deux thèmes montrent deux stratégies (polynomiales) pour approximer une fonction sur un intervalle borné de $\mathbb{R}$.

Interpoler est très naturel, mais ne conduit pas nécessairement à une approximation "homogène" sur l'intervalle, ni à une convergence uniforme lorsque l'on augmente le nombre de points d'interpolation : des phénomènes assez désagréables peuvent arriver...

Les polynômes de Bernstein ont pour qualité d'approximer uniformément les fonctions continues. Malheureusement, la convergence (lorsqu'on augmente les degrés des polynômes) est très lente et n'est pas satisfaisante, même quand la fonction approximée est un polynôme de degré 2 !

Dans le chapitre 3, on verra une autre façon d'approximer «au sens L^2» des fonctions continues grâce à un produit scalaire et à une projection orthogonale... On abordera également les séries de Fourrier, un grand classique !

Il existe beaucoup d'autres façon d'approcher une fonction : fractions rationnelles, polynômes trigonométriques, etc.

1.1 Introduction : les polynômes de Lagrange

Le polynôme d'interpolation de Lagrange est l'unique polynôme de degré au plus n passant par une famille de $n+1$ points (x_i, y_i) de $\mathbb{R}^2$, d'abscisses distinctes. Ce polynôme a pour expression

$$p_n(x) = \sum_{i=0}^{n} \left(\prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \right) y_i$$

C'est l'instruction `interp` qui les calcule en s'appliquant à deux listes contenant respectivement les abscisses et les ordonnées. En voici un exemple.

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘lagrange1.mpl‘ ;

# Le polynôme d’interpolation de Lagrange est l’unique polynôme de

# degré au plus n-1 passant par une famille de n points (x_i,y_i)

# de R^2, d’abscisses distinctes.

>

# On definit une fonction

> f := x -> 5+1/(1+x^2) ;
```

$$f := x \to 5 + \frac{1}{1+x^2}$$

```
>

# Puis une liste d’abscisses

> X := [1,2,3,4,5,20,21,22,23,24] ;
```

$$X := [1,\ 2,\ 3,\ 4,\ 5,\ 20,\ 21,\ 22,\ 23,\ 24]$$

```
>

# La liste des ordonnées : images des abscisses par la fontion f

> Y := map(f, X) ;
```

$$Y := [\frac{11}{2},\ \frac{26}{5},\ \frac{51}{10},\ \frac{86}{17},\ \frac{131}{26},\ \frac{2006}{401},\ \frac{2211}{442},\ \frac{2426}{485},\ \frac{2651}{530},\ \frac{2886}{577}]$$

```
>

# On invoque "interp" qui calcule le polynome d’interpolation

# en la variable x (troisième argument de la fonction "interp").

> p := interp(X,Y,x) ;
```

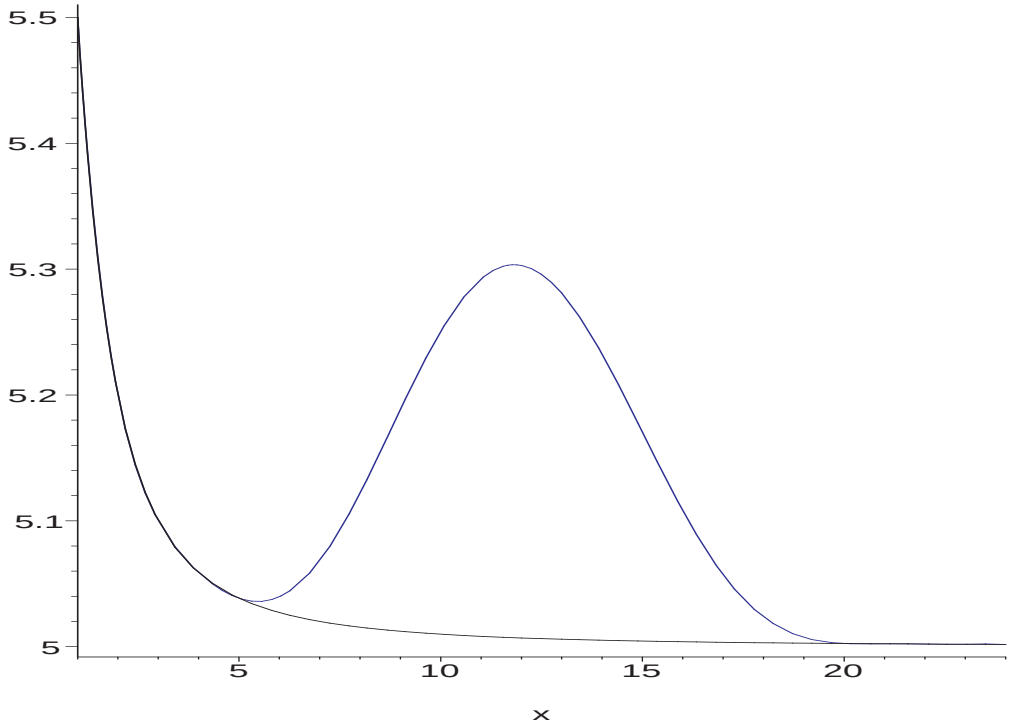
$$\begin{aligned} p := & -\frac{129941}{178759436113960}\,x^9 + \frac{40864157}{446898590284900}\,x^8 - \frac{25484181}{5257630473940}\,x^7 \\ & + \frac{62805014833}{446898590284900}\,x^6 - \frac{433337549589}{178759436113960}\,x^5 + \frac{2845686018257}{111724647571225}\,x^4 \\ & - \frac{7252891015699}{44689859028490}\,x^3 + \frac{68965820059583}{111724647571225}\,x^2 - \frac{5948064162648}{4468985902849}\,x \\ & + \frac{28390542131526}{4468985902849} \end{aligned}$$

```
>
```

```
# On fait alors tracer les graphes de f (en noir) et du polynôme
# d'interpolation p (en bleu)
# Rappel : f est une fonction,  f(x) et p sont des expressions en x
> minX := min(op(X)) ; maxX := max(op(X)) ;
```

$$minX := 1$$
$$maxX := 24$$

```
> plot([f(x), p], x = minX..maxX, color=[black,blue]) ;
```

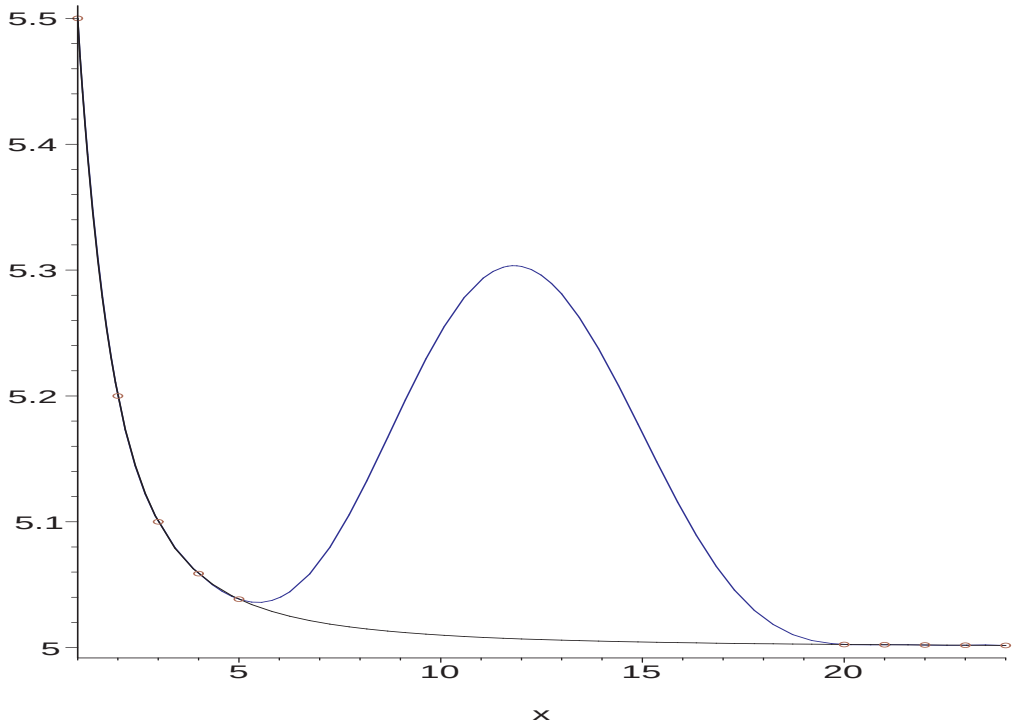



```
> pour_le_fichier(%, 'lagrange1') :
>
# Plaçons les points d'interpolation sur le graphe précédent :
# le dessin des graphes de f(x) et p sont placés dans une variable
# (étonnant, non ?), puis on y ajoute les points d'interpolation
# (déconcertant !).
> with(plots) :   # ce package est chargé une fois pour toute la suite
```

```
Warning, the name changecoords has been redefined

>

> Points := [ seq([X[i],Y[i]], i=1..nops(X)) ] ;
```

$$Points := [[1, \frac{11}{2}], [2, \frac{26}{5}], [3, \frac{51}{10}], [4, \frac{86}{17}], [5, \frac{131}{26}], [20, \frac{2006}{401}], [21, \frac{2211}{442}], [22, \frac{2426}{485}], [23, \frac{2651}{530}], [24, \frac{2886}{577}]]$$

```
> graphe := plot([f(x), p], x = minX..maxX, color = [black,blue]) :

> graphe := graphe, pointplot(Points, color = brown, symbol = CIRCLE):

>

# Et on visionne le tout

> display(graphe) ;
```




```
> pour_le_fichier(%, ‘lagrange1‘) :

>

# Le dessin fait apparaitre que l’approximation par le polynôme
```

```
# d'interpolation n'est pas "homogène" sur tout l'intervalle.
# Cela peut paraître normal puisque l'on n'a pas pris de point
# d'abscisse comprise entre 6 et 19...
# Recommençons en choisissant cette fois les abscisses de manière
# équirépartie (cela ne peut pas être mauvais, c'est si naturel !) :
>
# Voici une fonction affine réalisant l'équirépartition des abscisses
# entre minX et maxX
> equi := unapply(minX + (maxX - minX) * k / (nops(X) - 1) , k ) ;
```

$$equi := k \to 1 + \frac{23}{9}\,k$$

```
>
> X := [seq(equi(k), k=0..nops(X)-1)] ; Y := map(f, X) ;
```

$$X := [1, \frac{32}{9}, \frac{55}{9}, \frac{26}{3}, \frac{101}{9}, \frac{124}{9}, \frac{49}{3}, \frac{170}{9}, \frac{193}{9}, 24]$$

$$Y := [\frac{11}{2}, \frac{5606}{1105}, \frac{15611}{3106}, \frac{3434}{685}, \frac{51491}{10282}, \frac{77366}{15457}, \frac{12059}{2410}, \frac{144986}{28981}, \frac{186731}{37330}, \frac{2886}{577}]$$

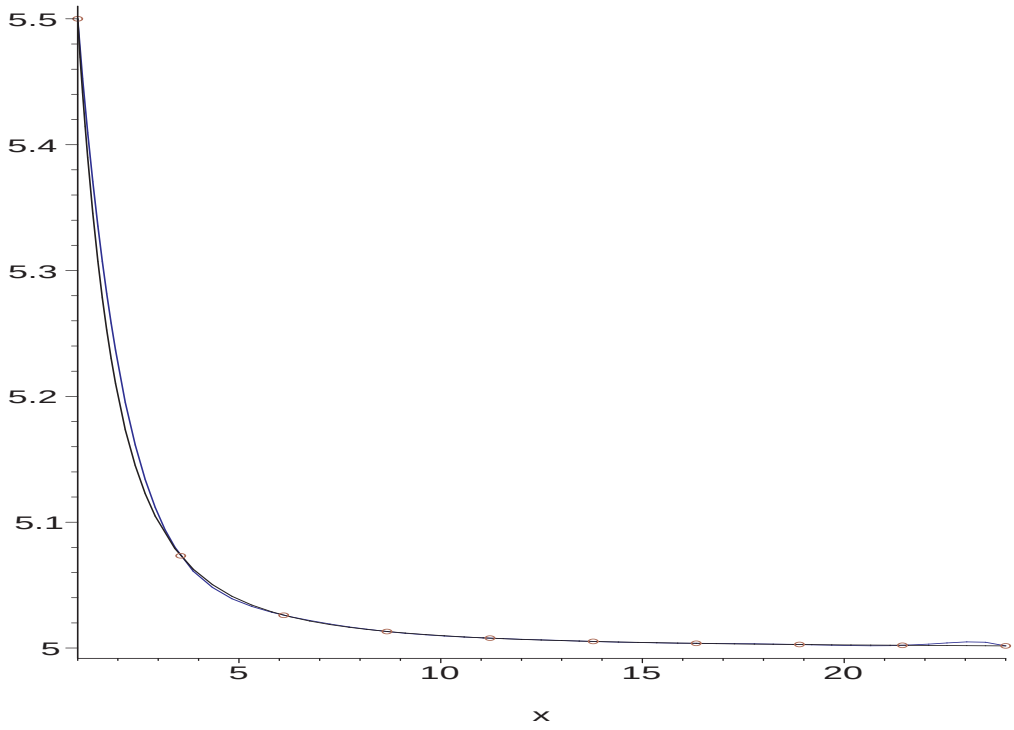
```
>
# On calcule le polynôme interpolateur avec la fonction interp.
# la fonction "evalf" est là pour numériser en flottant le résultat.
> p := interp(X,Y,x) :
> p := evalf(p) ;
```

$$\begin{aligned} p := {} & -.1414564339\,10^{-9}\,x^9 + .1770046924\,10^{-7}\,x^8 - .9586678680\,10^{-6}\,x^7 \\ & + .00002944986059\,x^6 - .0005646889020\,x^5 + .007002480302\,x^4 \\ & - .05618551999\,x^3 + .2824066738\,x^2 - .8178241459\,x + 6.085136692 \end{aligned}$$

```
>
# On fait alors tracer les graphes de f (en noir) et du polynôme
# d'interpolation p (en bleu) et les points d'interpolation (en brun)
> Points := [ seq([X[i],Y[i]], i=1..nops(X)) ] ;
```

$$Points := [[1, \frac{11}{2}], [\frac{32}{9}, \frac{5606}{1105}], [\frac{55}{9}, \frac{15611}{3106}], [\frac{26}{3}, \frac{3434}{685}], [\frac{101}{9}, \frac{51491}{10282}], [\frac{124}{9}, \frac{77366}{15457}], [\frac{49}{3}, \frac{12059}{2410}], [\frac{170}{9}, \frac{144986}{28981}], [\frac{193}{9}, \frac{186731}{37330}], [24, \frac{2886}{577}]]$$

```
> graphe := plot([f(x), p], x = minX..maxX, color = [black,blue]) :

> graphe := graphe, pointplot(Points, color = brown, symbol = CIRCLE):

>

> display(graphe) ;
```




```
> pour_le_fichier(%, ‘lagrange1‘) :

>

# Et bien, l’approximation est plus "homogène" sur tout l’intervalle !
```

1.2 Phénomène de Runge

Le polynôme d'interpolation de Lagrange est l'unique polynôme de degré au plus n passant par une famille de $n+1$ points (x_i, y_i) de $\mathbb{R}^2$, d'abscisses distinctes. Ce polynôme a pour expression

$$p_n(x) = \sum_{i=0}^{n} \left(\prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \right) y_i$$

Interpoler une fonction quelconque f en les abscisses x_i, c'est poser tout simplement $y_i = f(x_i)$. Intéressons-nous (graphiquement pour l'instant) à la différence $f(x) - p_n(x)$. Si les x_i sont choisis au hasard (ou presque), alors cette différence peut être très surprenante et un phénomène curieux peut naître...

Lorsqu'on interpole (en $n+1$ points) un polynôme f de degré inférieur ou égal à n, on obtient évidemment $p_n = f$, et cela quel que soit le choix des abscisses x_i. Mais quand $f = t^{n+1}$, il en va tout autrement : il est impossible d'avoir $f = p_n$ puisque $\deg(p_n) \leqslant n$ par définition ! Ainsi, sur l'intervalle contenant les x_i, la différence $t^{n+1} - p_n(t)$ n'est pas nulle. Les points de Tchebyshev, i.e. les zéros du polynôme $T_{n+1} = \cos((n+1)\arccos)$, sont faits pour minimiser la norme sup. $||t^{n+1} - p_n(t)||_\infty$ sur l'intervalle $[-1, 1]$ (voir page 16).

Lorsque les abscisses x_i sont les points de Tchebyshev transportées par la bijection $x \mapsto \frac{a+b}{2} + x\frac{b-a}{2}$ sur un intervalle $[a, b]$, et que la fonction f est de classe C^{n+1}, il est prouvé que

$$||f - p_n||_\infty \leqslant \left(\frac{b-a}{2} \right)^{n+1} \frac{||f^{(n+1)}||_\infty}{2^n (n+1)!}$$

> Remarque. Si la fonction f est précisée concrètement, ainsi que le nombre de points d'interpolation, alors il existe certainement un meilleur choix adapté à cette fonction f particulière. Exemple extrême : sur $[-1, 1]$, considérer la fonction $f = T_{n+1}^2$. Alors le polynôme interpolateur en les racines du polynôme de Tchebyshev T_{n+1} est le polynôme nul, ce qui est largement perfectible au sens de la norme $||\ \ ||_\infty$!

Voici un dernier résultat (cf J.-P. Demailly, Analyse Numérique et Equations Différentielles, page 51) :

> Si f est lipschitzienne sur $[a, b]$ alors la suite des polynômes d'interpolation aux points de Tchebyshev converge uniformément vers f sur $[a, b]$.

```
> restart : read ‘my_config.mpl‘ : interface(echo=3):
> read ‘lagrange2.mpl‘ ;

# On choisit une fonction f et

# n+1 abscisses équiréparties entre deux réels a et b

> f := x -> 1/(1+10*x^2) ; a := -1 ; b := 1 ; n := 14 ;
```

$$f := x \to \frac{1}{1+10\,x^2}$$

$$a := -1$$

$$b := 1$$

$$n := 14$$

```
> equi := unapply(a + (b-a) * k / n, k ) ;
```

$$equi := k \to -1 + \frac{1}{7}\,k$$

```
> X := [seq(equi(k), k=0..n)] ;
```

$$X := [-1, \frac{-6}{7}, \frac{-5}{7}, \frac{-4}{7}, \frac{-3}{7}, \frac{-2}{7}, \frac{-1}{7}, 0, \frac{1}{7}, \frac{2}{7}, \frac{3}{7}, \frac{4}{7}, \frac{5}{7}, \frac{6}{7}, 1]$$

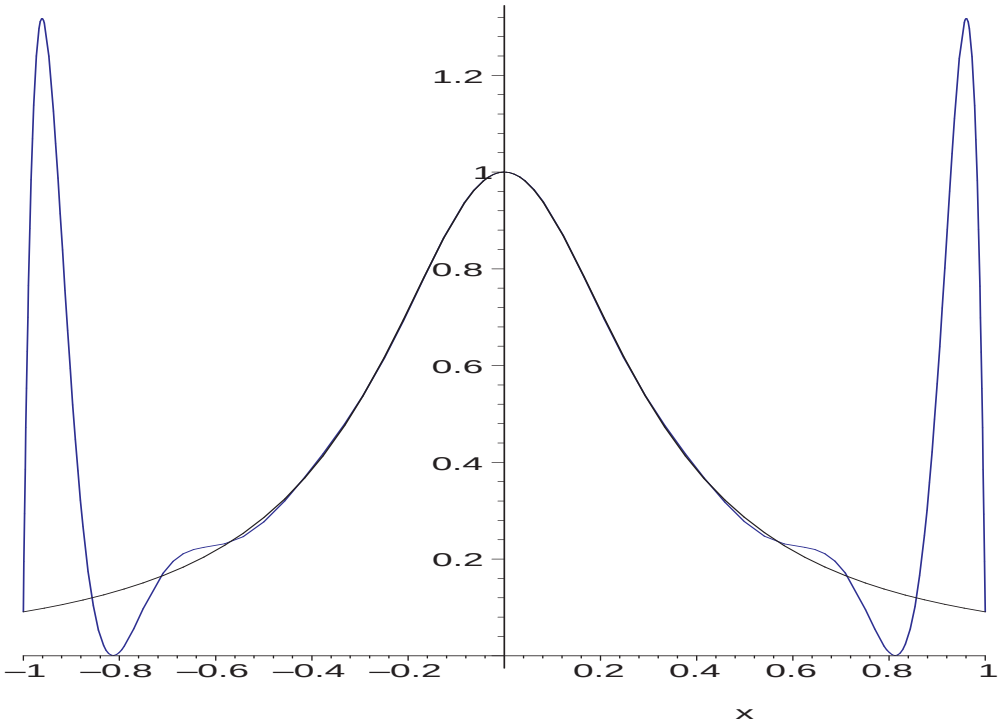
```
>

# On interpole

> Y := map(f, X) ;
```

$$Y := [\frac{1}{11}, \frac{49}{409}, \frac{49}{299}, \frac{49}{209}, \frac{49}{139}, \frac{49}{89}, \frac{49}{59}, 1, \frac{49}{59}, \frac{49}{89}, \frac{49}{139}, \frac{49}{209}, \frac{49}{299}, \frac{49}{409}, \frac{1}{11}]$$

```
> p := interp(X,Y,x) ;
```

$$p := -\frac{2000221092520010}{205206109252001}\,x^2 + \frac{16162004525200100}{205206109252001}\,x^4 + \frac{2252471282810000}{1695918258281}\,x^8 - \frac{7881000841091000}{18655100841091}\,x^6 - \frac{42827283109100000}{18655100841091}\,x^{10} + \frac{17795940687000000}{8922004750087}\,x^{12} + 1 - \frac{138412872010000000}{205206109252001}\,x^{14}$$

```
>

# On fait alors tracer les graphes de f(x) et du polynôme p

> plot([f(x), p], x = a..b, color=[black, blue]) ;
```




```
> pour_le_fichier(%, 'lagrange2') :
>
# Le dessin fait apparaître le phénomène de Runge sur les bords :
# l'approximation par le polynôme d'interpolation n'est pas
# uniforme. Et vous pouvez essayer d'augmenter le nombre de
# points d'interpolation : cela ne fait qu'empirer le résultat !!!
>
# Recommençons en choisissant intelligemment les abscisses :
# on utilise les polynômes de Tchebyshev dont les racines
# fournissent les << meilleurs >> choix pour l'interpolation
# (voir le cours d'analyse numerique of course).
>
> Tcheby := unapply(cos((2*i+1) * Pi / (2*n+2)) , i ) ;
```

$$Tcheby := i \rightarrow \cos(\frac{1}{30}\,(2\,i+1)\,\pi)$$

```
> racines := [seq(Tcheby(i), i= 0..n)] ;
```

$$racines := [\cos(\frac{1}{30}\,\pi),\ \cos(\frac{1}{10}\,\pi),\ \frac{1}{2}\,\sqrt{3},\ \cos(\frac{7}{30}\,\pi),\ \cos(\frac{3}{10}\,\pi),\ \cos(\frac{11}{30}\,\pi),\ \cos(\frac{13}{30}\,\pi),\ 0,$$
$$-\cos(\frac{13}{30}\,\pi),\ -\cos(\frac{11}{30}\,\pi),\ -\cos(\frac{3}{10}\,\pi),\ -\cos(\frac{7}{30}\,\pi),\ -\frac{1}{2}\,\sqrt{3},\ -\cos(\frac{1}{10}\,\pi),\ -\cos(\frac{1}{30}\,\pi)]$$

```
>

# Il est important de "numériser" les valeurs des racines

# sinon les calculs réalisés par Maple sont symboliques

# et deviennent très (parfois trop !) lourds...

> X2 := map(evalf, racines) ;
```

$$X2 := [.9945218954,\ .9510565163,\ .8660254040,\ .7431448255,\ .5877852522,$$
$$.4067366425,\ .2079116904,\ 0.,\ -.2079116904,\ -.4067366425,\ -.5877852522,$$
$$-.7431448255,\ -.8660254040,\ -.9510565163,\ -.9945218954]$$

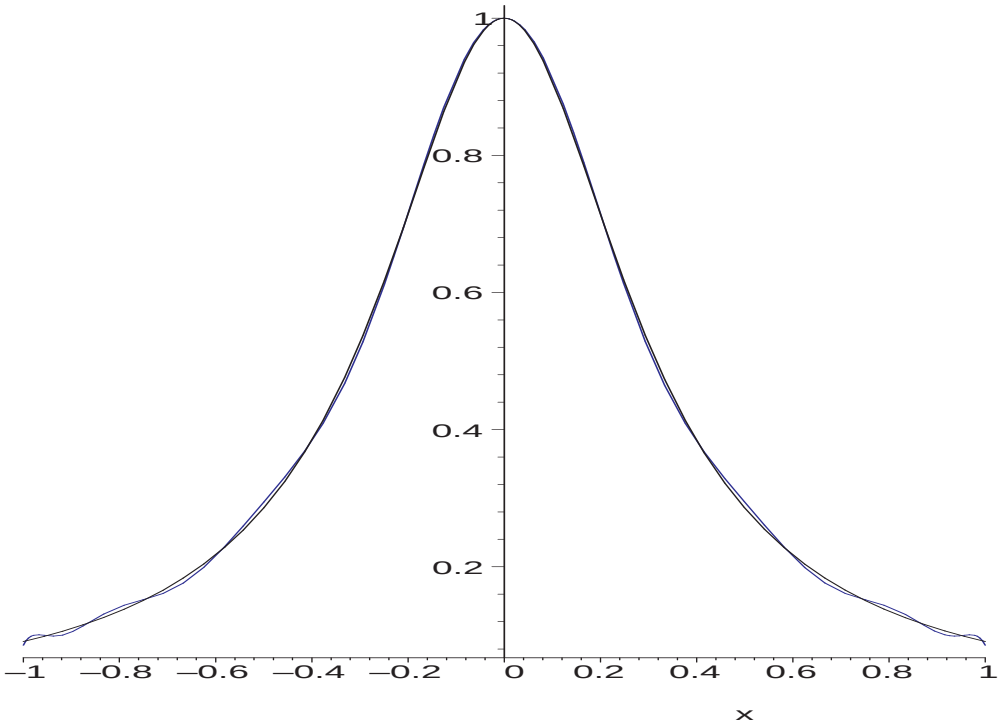
```
> Y2 := map(f, X2) ;
```

$$Y2 := [.09182114196,\ .09955117383,\ .1176470588,\ .1533121014,\ .2244711726,$$
$$.3767404991,\ .6981910589,\ 1.,\ .6981910589,\ .3767404991,\ .2244711726,$$
$$.1533121014,\ .1176470588,\ .09955117383,\ .09182114196]$$

```
> p2 := interp(X2, Y2, x) ;
```

$$p2 := -.0001425\,x^9 + .0000806\,x^7 - .00002040\,x^5 + .2139\,10^{-5}\,x^3 + .0001162\,x^{11}$$
$$- .00003542\,x^{13} + .9999999999 - 9.108759100\,x^2 + 57.81459822\,x^4$$
$$+ 476.7946249\,x^8 - 218.7976843\,x^6 - 585.0568662\,x^{10} + 374.7868090\,x^{12}$$
$$- 97.34721538\,x^{14} - .599\,10^{-7}\,x$$

```
>

# Un petit graphe bien sûr !

> plot([f(x), p2], x = a..b, color=[black,blue]);
```




```
> pour_le_fichier(%, ‘lagrange2‘) :
>
# Y’a pas photo !
>
# Un dernier exemple :
> (a, b, n) := (-1,3,8) ;
```

$$a,\, b,\, n := -1,\, 3,\, 8$$

```
> f := abs ;
```

$$f := \mathrm{abs}$$

```
>
# Pour transporter des abscisses appartenant à l’intervalle [-1,1]
# (les racines des polynômes de Tchebyshev par exemple)
# sur un intervalle [a,b], on considère l’application affine
#           x -> (a+b)/2 + x(b-a)/2
# réalisant une bijection de [-1,1] sur [a,b].
```

```
> bijection := unapply((a + b) / 2 + x * (b - a) / 2, x );
```

$$bijection := x \to 2\,x + 1$$

```
>
```

```
> Tcheby := unapply(cos((2*i+1) * Pi / (2*n+2)) , i ) ;
```

$$Tcheby := i \to \cos(\frac{1}{18}\,(2\,i + 1)\,\pi)$$

```
> racines := [seq(Tcheby(i), i= 0..n)] ;
```

$$racines := [\cos(\frac{1}{18}\,\pi),\ \frac{1}{2}\,\sqrt{3},\ \cos(\frac{5}{18}\,\pi),\ \cos(\frac{7}{18}\,\pi),\ 0,\ -\cos(\frac{7}{18}\,\pi),\ -\cos(\frac{5}{18}\,\pi),\ -\frac{1}{2}\,\sqrt{3},\ -\cos(\frac{1}{18}\,\pi)]$$

```
> X3 := map(bijection@evalf, racines) ; Y3 := map(f, X3) ;
```

$$X3 := [2.969615506,\ 2.732050808,\ 2.285575219,\ 1.684040286,\ 1.,\ .3159597144,\ -.285575219,\ -.732050808,\ -.969615506]$$

$$Y3 := [2.969615506,\ 2.732050808,\ 2.285575219,\ 1.684040286,\ 1.,\ .3159597144,\ .285575219,\ .732050808,\ .969615506]$$

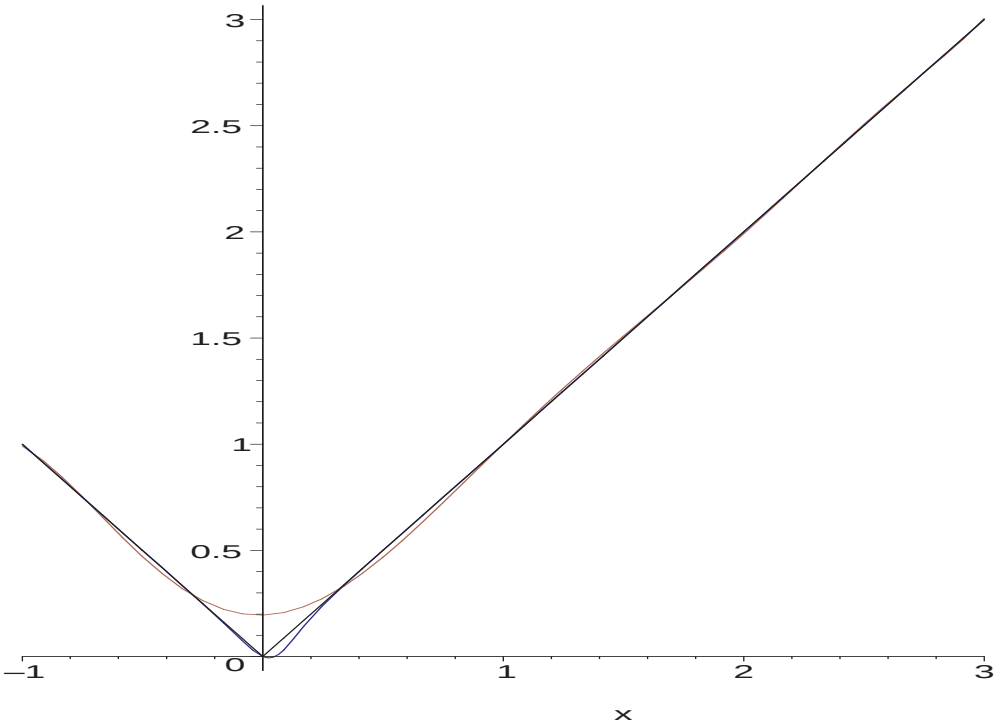
```
> p := interp(X3,Y3,x) ;
```

$$p := .004691082544\,x^8 - .03283757767\,x^7 + .195418943 + .03777175108\,x^6 + .0317607945\,x + .1915071436\,x^5 + 1.204011042\,x^2 - .4459902464\,x^4 - .1863329326\,x^3$$

```
>

# Voici une fonction infiniment dérivable, proche de la fonction f

> g := x -> x * (exp(20*x-1) - 1) / (exp(20*x-1) + 1) ;
```

$$g := x \to \frac{x\,(e^{(20\,x-1)} - 1)}{e^{(20\,x-1)} + 1}$$

```
>

> plot([f(x), g(x), p], x = a..b, color=[black,blue,brown]) ;
```




```
> pour_le_fichier(%, ‘lagrange2‘) :
>
# L’approximation de f par p n’est pas bonne au voisinage
# du point où la fonction change brutalement de comportement.
# Pourquoi ici et pas ailleurs sur l’intervalle [a,b] ???
>
# Par ailleurs, on voit bien que la majoration théorique
# de || f-p_n || est fausse ici puisque la dérivée seconde de f est
# nulle : le majorant de || f-p_n || est donc nul !
# Mais pas de panique : la fonction f n’est pas dérivable en 0,
# donc ne vérifie pas les hypothèses requises pour que la
# majoration soit valide...
```

1.3 Erreur commise lors d'une interpolation

Le polynôme d'interpolation de Lagrange est l'unique polynôme de degré au plus n passant par une famille de $n+1$ points (x_i, y_i) de $\mathbb{R}^2$, d'abscisses distinctes. Ce polynôme a pour expression

$$p_n(x) = \sum_{i=0}^{n} \left(\prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \right) y_i$$

Dans la situation où les points $(y_i)_i$ sont les images des $(x_i)_i$ par une fonction f de classe C^{n+1}, on connaît la formule d'erreur en tout point x :

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\zeta_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{1.1}$$

$$\text{où} \qquad \zeta_x \in \,]\min(x, x_0, \ldots, x_n), \max(x, x_0, \ldots, x_n)[$$

Remarque. Si $||f^{(n+1)}||_\infty$ est bornée par une suite géométrique $u_0 q^n$ alors la convergence est uniforme sur l'intervalle $[a, b]$ et très rapide (en théorie) car

$$||f - p_n||_\infty \leqslant \frac{u_0 q^n}{(n+1)!} (b-a)^{n+1} \;\ll\; \frac{1}{(n/2)!}$$

Avec la formule (1.1), on retrouve évidemment que $f = p_n$ lorsque $f^{(n+1)} = 0$, c'est-à-dire si f est un polynôme de degré inférieur ou égal à n. Par ailleurs, en posant $f(x) = x^{n+1}$, la formule d'erreur donne $x^{n+1} - p_n(x) = \prod_{i=0}^{n}(x - x_i)$. Sur l'intervalle $[-1, 1]$, les points de Tchebyshev sont les réels x_i qui minimisent la norme sup $|| \prod_{i=0}^{n}(x - x_i)||_\infty$: on a exactement $|| \prod_{i=0}^{n}(x - x_i)||_\infty = 2^{-n}$ si les x_i sont les racines de T_{n+1}.

Détermination de ζ_x lorque $f(x) = x^k$

Quels que soient les réels $x_0, \ldots, x_n$, on montre que la différence entre le polynôme x^k et $p_n(x)$ s'écrit (pour $k \geqslant n+1$ bien sûr !) :

$$x^k - p_n(x) = H_d(x_0, \ldots, x_n, x) \prod_{i=0}^{n} (x - x_i) \qquad \text{avec} \qquad d = k - n - 1$$

où $H_d(x_0, \ldots, x_n, x)$ désigne le polynôme symétrique homogène complet de degré d en $x_0, \ldots, x_n, x$, c'est-à-dire H_d est la somme des $\frac{(d+n+1)!}{d!(n+1)!} = \frac{k!}{d!(n+1)!}$ monômes en $x_0, \ldots, x_n, x$ de degré total d :

$$H_d(x_0, \ldots, x_n, x) = \sum_{\substack{i \in \mathbb{N}^{n+2} \\ i_0 + \cdots + i_{n+1} = d}} x_0^{i_0} \cdots x_n^{i_n} x^{i_{n+1}}$$

Ainsi, l'égalité (1.1) se réécrit simplement en

$$\zeta_x^d = \operatorname*{moy}_{z \in \{x_0, \ldots, x_n, x\}^d} z_1 \ldots z_d$$

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘lagrange3.mpl‘ ;
# Dans la formule d’erreur d’une interpolation, on voit le produit
# des (T - x_i). On constate d’ailleurs que l’erreur commise
# pour f(T) = T^(n+1) est exactement le produit (T-x_0)...(T-x_n) :
>
> n := 3 ; X := [seq(x[i], i=0..n)] ; Y := map(f, X) ;
```

$$n := 3$$

$$X := [x_0,\, x_1,\, x_2,\, x_3]$$

$$Y := [\mathrm{f}(x_0),\, \mathrm{f}(x_1),\, \mathrm{f}(x_2),\, \mathrm{f}(x_3)]$$

```
> p := interp(X, Y, T) :
>
> for i from n to n+2 do
>   f := unapply(T^i, T) ; ‘erreur en T‘ = factor(f(T)-p) ;
> od ;
```

$$f := T \to T^3$$

$$erreur\ en\ T = 0$$

$$f := T \to T^4$$

$$erreur\ en\ T = (x_3 - T)\,(-T + x_2)\,(-T + x_1)\,(-T + x_0)$$

$$f := T \to T^5$$

$$erreur\ en\ T = (x_3 - T)\,(-T + x_2)\,(-T + x_1)\,(-T + x_0)\,(x_0 + x_1 + x_2 + x_3 + T)$$

```
>
# Il est donc naturel de chercher à minimiser le produit des T-x_i
# en valeur absolue. Comparons les résultats obtenus par une
# répartition régulière des x_i (entre -1 et 1) et les racines
# du polynôme de Tchebyshev T_{n+1} normalisé (i.e. unitaire).
>
> n := 10 ; p := product(x-1+2*k/n, k=0..n) ;
```

$$n := 10$$

$$p := (x-1)\,(x-\frac{4}{5})\,(x-\frac{3}{5})\,(x-\frac{2}{5})\,(x-\frac{1}{5})\,x\,(x+\frac{1}{5})\,(x+\frac{2}{5})\,(x+\frac{3}{5})\,(x+\frac{4}{5})\,(x+1)$$

```
>
```

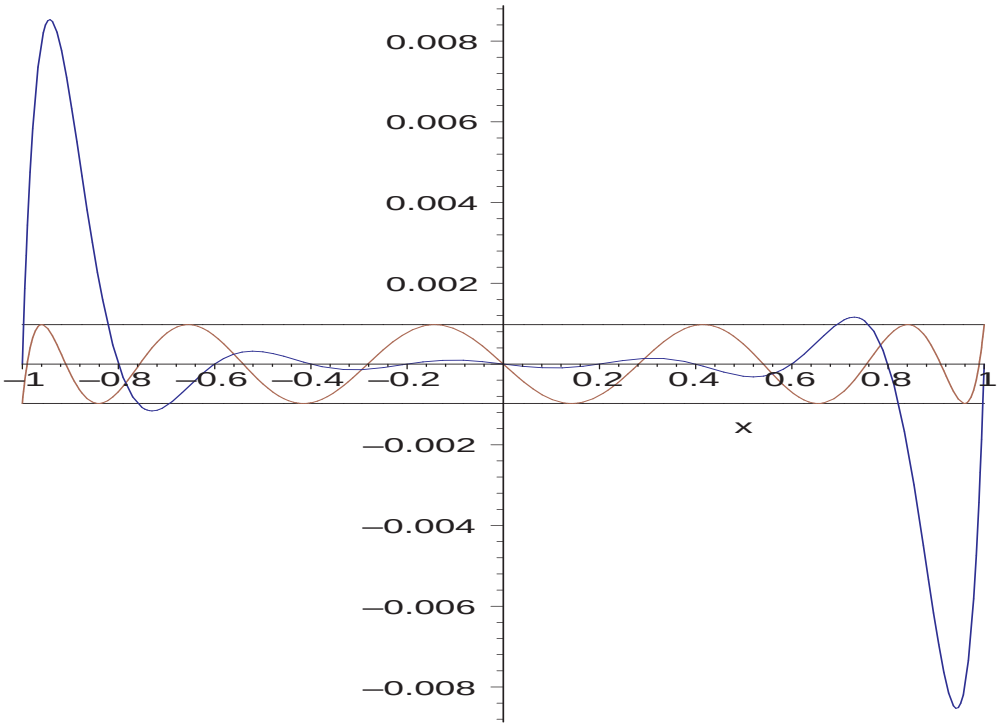
```
# Le package suivant contient des algorithmes calculant
# certains type de polynômes orthogonaux pour différents
# produits scalaires... Celui qui nous intéresse est T(n+1,x)
# qui est égal à cos((n+1).arccos(x)) pour -1 < x < 1 .
>
> with(orthopoly) ;
```

$$[G,\ H,\ L,\ P,\ T,\ U]$$

```
> Tcheby_normalise :=  T(n+1,x)/2^n ; # cos((n+1) * arccos(x)) / 2^n ;
```

$$Tcheby\_normalise := x^{11} - \frac{11}{4}\,x^9 + \frac{11}{4}\,x^7 - \frac{77}{64}\,x^5 + \frac{55}{256}\,x^3 - \frac{11}{1024}\,x$$

```
>
> plot([p, Tcheby_normalise, 1/2^n, -1/2^n], x=-1..1,
>      color=[blue, brown, black, black]) ;
```




```
> pour_le_fichier(%, ‘lagrange3‘) :
>
```

```
# On constate ici plusieurs résultats :
# 1/  T_{n+1} possède une norme sup bien inférieure à celle de p,
#     d'où l'intérêt fondamental des polynômes de Tchebyshev pour
#     le problème de l'interpolation.
# 2/  p oscille fortement aux bords de l'intervalle ce qui peut à la
#     rigueur expliquer grosso-modo les phénomènes de mauvaise
#     approximation sur les bords des intervalles avec une répartition
#     régulière des abscisses des points d'interpolation...
```

Revenons à la formule d'erreur (1.1) et essayons de voir ce(s) ζ_x sur un exemple...

```
> restart : read `my_config.mpl` : interface(echo=3) :
> read `lagrange4.mpl` ;
```

```
> f := x -> cos(2*x) ;
```

$$f := x \to \cos(2\,x)$$

```
> n := 2 ;
```

$$n := 2$$

```
> X := [$1..n+1] ;
```

$$X := [1,\, 2,\, 3]$$

```
>
```

```
> Y := map(f, X) ;
```

$$Y := [\cos(2),\, \cos(4),\, \cos(6)]$$

```
> px := interp(X,Y,x) ;
```

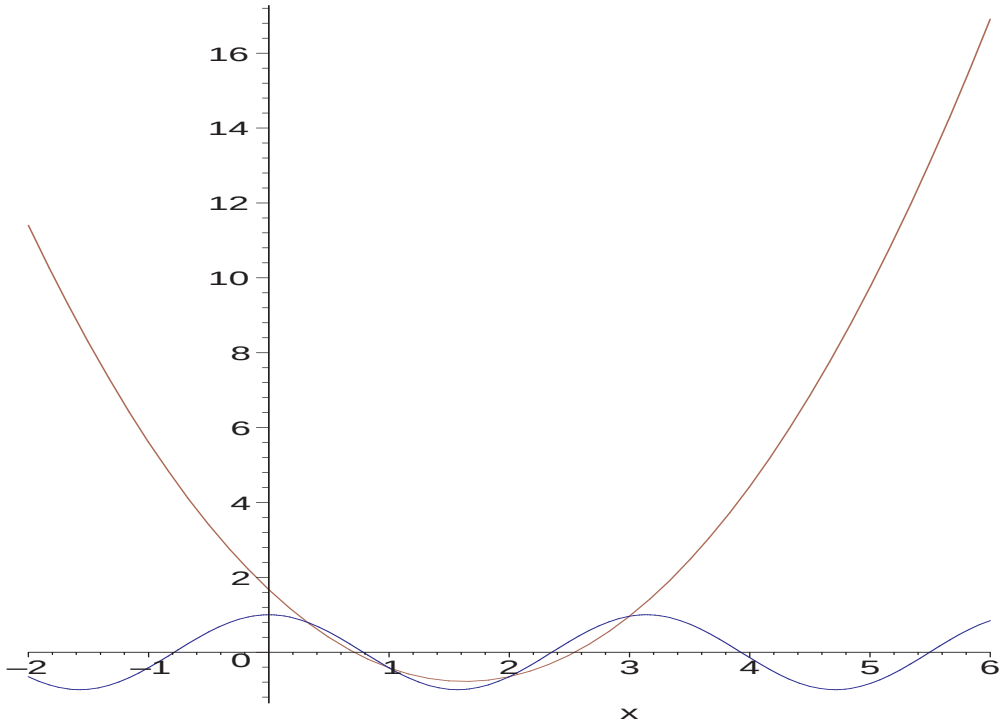
$$px := \frac{1}{2}\cos(6)\,x^2 - \frac{3}{2}\cos(6)\,x + \cos(6) - \cos(4)\,x^2 + 4\cos(4)\,x - 3\cos(4) + \frac{1}{2}\cos(2)\,x^2 - \frac{5}{2}\cos(2)\,x + 3\cos(2)$$

```
>
```

```
> (m, M) := -2,6 ;
```

$$m,\, M := -2,\, 6$$

```
> plot([f(x), px], x = m..M, color=[blue,brown]) ;
```




```
> pour_le_fichier(%, 'lagrange4') :
>
# Construisons l'expression de l'erreur commise
# (cela dépendant de x et zeta)
>
> diff(f(zeta), zeta) ;
```

$$-2\sin(2\,\zeta)$$

```
> diff(f(zeta), zeta, zeta) ;
```

$$-4\cos(2\,\zeta)$$

```
> diff(f(zeta), zeta$3) ;
```

$$8\sin(2\,\zeta)$$

```
>
> produit := mul(x-X[i], i=1..n+1) / (n+1) !;
```

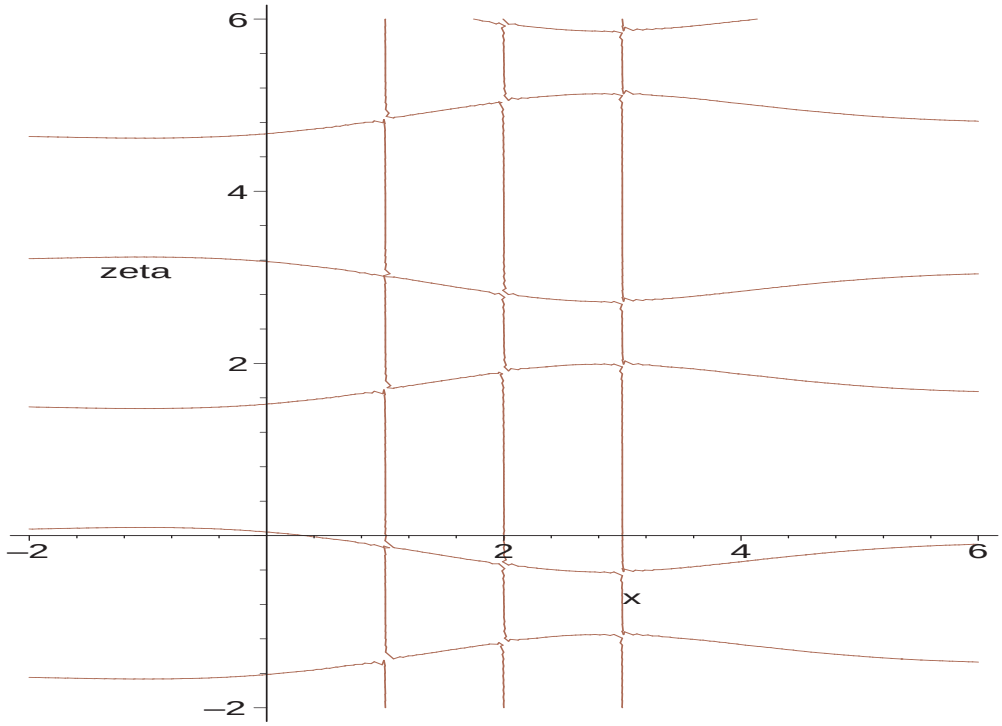
$$produit := \frac{1}{6}\,(x-1)\,(x-2)\,(x-3)$$

```
>
```

```
> erreur := produit * diff(f(zeta), zeta$(n+1)) ;
```

$$erreur := \frac{4}{3}\,(x-1)\,(x-2)\,(x-3)\sin(2\,\zeta)$$

```
>
# A l'aide d'un graphe, déterminons les couples (x, zeta) de R^2
# vérifiant l'égalité donnée par la formule d'erreur d'approximation
>
> with(plots) :
Warning, the name changecoords has been redefined
> implicitplot(f(x)-px = erreur, x = m..M, zeta = m..M,
>                     grid=[100,100], color=brown) ;
```

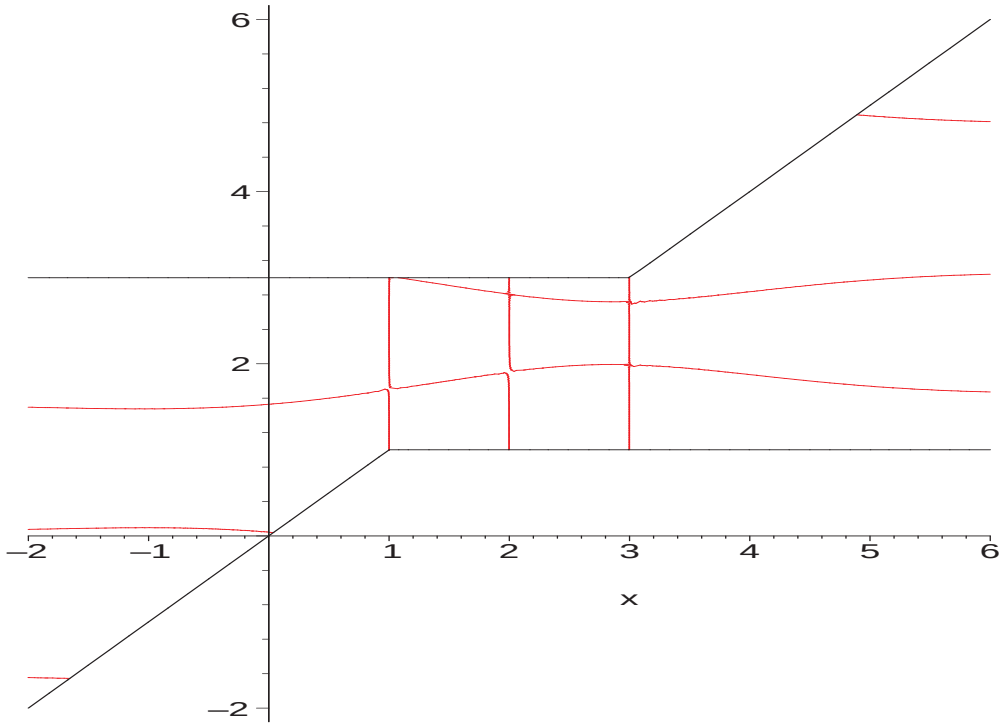



```
> pour_le_fichier(%, 'lagrange4') :
>
# Attention tout de même :
# zeta doit appartenir à l'intervalle ouvert
```

```
# ] min(x, x_0, ..., x_n), max(x, x_0, ..., x_n) [.
>
> (mX, MX) := min(x,op(X)), max(x,op(X)) ;
```

$$mX,\ MX := \min(1,\ x),\ \max(3,\ x)$$

```
>
> le_bord := plot([mX, MX], x = m..M,  color=black):
>
> bons_couples := implicitplot(f(x)-px = erreur,
>         x = m..M, zeta = mX..MX, grid=[100,100], color=red):
>
> display([le_bord,bons_couples]) ;
```




```
> pour_le_fichier(%, ‘lagrange4‘) :
>
# On voit que zeta n’est pas forcément unique pour un x donné
```

```
# et il est interdit d'utiliser une fonction
# (continue, dérivable, etc.) qui enverrait x sur zeta(x)
# sans prendre d'énormes précautions mathématiques !
```

1.4 Polynômes de Newton

On va expérimenter la méthode des différences finies, cas particulier de celle des différences divisées. Elle s'applique à des points d'abscisses équidistantes et permet de rajouter des points d'interpolation supplémentaires sans refaire tous les calculs. On définit l'opérateur aux différences finies par

$$(\Delta y)_i = y_{i+1} - y_i \qquad \text{avec} \qquad y_i = f(a + i.h)$$

En itérant, on trouve une formule binomiale

$$(\Delta^n y)_i = \sum_{j=0}^{n} \binom{n}{j} (-1)^{n-j} y_{i+j} \qquad \text{avec} \qquad \binom{n}{j} = \frac{n!}{j!(n-j)!} = \frac{n(n-1)\cdots(n-j+1)}{j!}$$

Remarque. $\binom{n}{j}$ est un polynôme en n de degré j, qui s'annule en $n = 0, 1, \ldots, j-1$, et qui vaut 1 pour $n = j$.

On a alors, pour une suite de $n+1$ points équidistants $a, a+h, \ldots, a+nh$, la formule de Newton donnant le polynôme interpolateur :

$$p_n(x) = \sum_{k=0}^{n} (\Delta^k y)_0 \binom{z}{k} \qquad \text{avec} \qquad z = (x-a)/h$$

où $\binom{z}{k} = \dfrac{z(z-1)\cdots(z-k+1)}{k!}$ est le $(k+1)$-ième polynôme de la base de Newton.

```
> restart : read `my_config.mpl` : interface(echo=3) :
> read `lagrange5.mpl` ;

# Les coefficients binomiaux existent dans Maple

>

> binomial(5, 2) ;
```

$$10$$

```
> b := binomial(z, 4) ;
```

$$b := \text{binomial}(z,\ 4)$$

```
> eb := expand(b) ;
```

$$eb := \frac{1}{24} z^4 - \frac{1}{4} z^3 + \frac{11}{24} z^2 - \frac{1}{4} z$$

```
> feb := factor(eb) ;
```

$$feb := \frac{1}{24} z\,(z-1)\,(z-2)\,(z-3)$$

```
>
```

```
# Cette procédure a pour but de calculer le polynôme (en z)
# de degré n, qui s'annule en 0,1,...,n-1
# et vaut 1 au point n.
> Binomial := (z,n) -> factor(expand(binomial(z,n))) ;
```

$$Binomial := (z,\, n) \rightarrow \mathrm{factor}(\mathrm{expand}(\mathrm{binomial}(z,\, n)))$$

```
>
# Testons
> Binomial(z, 4) ;
```

$$\frac{1}{24}\, z\, (z-1)\, (z-2)\, (z-3)$$

```
> Binomial((x-a)/h, 4) ;
```

$$\frac{1}{24}\, \frac{(x-a)\, (-a+x-2\, h)\, (x-a-3\, h)\, (-a+x-h)}{h^4}$$

```
>
# Maintenant, définissons (Delta^n y)_0 en posant y_i = f(a+ih).
> Delta := n -> add(Binomial(n,i) * (-1)^(n-i) * f(a+i*h), i=0..n) ;
```

$$\Delta := n \rightarrow \mathrm{add}(\mathrm{Binomial}(n,\, i)\, (-1)^{(n-i)}\, \mathrm{f}(a+i\, h),\, i=0..n)$$

```
>
# Essayons
> Delta(5) ;
```

$$-\mathrm{f}(a) + 5\, \mathrm{f}(a+h) - 10\, \mathrm{f}(a+2\, h) + 10\, \mathrm{f}(a+3\, h) - 5\, \mathrm{f}(a+4\, h) + \mathrm{f}(a+5\, h)$$

```
>
# Appliquons la formule de Newton dans la procédure qui suit.
# La somme de 0 jusqu'à n illustre bien le fait que le calcul
# ne nécessite pas d'être repris au début si on ajoute
# de nouveaux points.
> z := (x-a)/h ;
```

$$z := \frac{x-a}{h}$$

```
> p := n -> add(Binomial(z,k) * Delta(k),  k=0..n) ;
```

$$p := n \to \mathrm{add}(\mathrm{Binomial}(z,\, k)\, \Delta(k),\, k = 0..n)$$

```
>
```

```
# Essayons sur un exemple concret avec des points d'abscisses
-5,-4,...
```

```
> f := x -> 1/(1+x^2) ; (n, a, h) := (10, -5, 1) ;
```

$$f := x \to \frac{1}{1+x^2}$$

$$n,\, a,\, h := 10,\, -5,\, 1$$

```
> px := p(n) ;
```

$$\begin{aligned} px := {} & \frac{31}{221} + \frac{9}{442}\, x + \frac{23}{2210}\, (x+4)\, (x+5) + \frac{7}{1105}\, (x+5)\, (x+4)\, (x+3) \\ & + \frac{19}{4420}\, (x+5)\, (x+4)\, (x+3)\, (x+2) - \frac{9}{4420}\, (x+5)\, (x+4)\, (x+3)\, (x+2)\, (x+1) \\ & - \frac{1}{884}\, x\, (x+5)\, (x+4)\, (x+3)\, (x+2)\, (x+1) \\ & + \frac{6}{5525}\, x\, (x-1)\, (x+5)\, (x+4)\, (x+3)\, (x+2)\, (x+1) \\ & - \frac{19}{44200}\, x\, (x-1)\, (x-2)\, (x+5)\, (x+4)\, (x+3)\, (x+2)\, (x+1) \\ & + \frac{1}{8840}\, x\, (x-1)\, (x-2)\, (x-3)\, (x+5)\, (x+4)\, (x+3)\, (x+2)\, (x+1) \\ & - \frac{1}{44200}\, x\, (x-1)\, (x-2)\, (x-3)\, (x-4)\, (x+5)\, (x+4)\, (x+3)\, (x+2)\, (x+1) \end{aligned}$$

```
>
```

```
# Comparons a la fonction ‘‘interp’’ intégrée dans Maple.
```

```
> X := [seq(a+i*h, i=0..n)] ; Y := map(f,X) ;
```

$$X := [-5,\, -4,\, -3,\, -2,\, -1,\, 0,\, 1,\, 2,\, 3,\, 4,\, 5]$$

$$Y := [\frac{1}{26},\, \frac{1}{17},\, \frac{1}{10},\, \frac{1}{5},\, \frac{1}{2},\, 1,\, \frac{1}{2},\, \frac{1}{5},\, \frac{1}{10},\, \frac{1}{17},\, \frac{1}{26}]$$

```
> q := interp(X,Y,x) ;
```

$$q := -\frac{1}{44200}\, x^{10} + \frac{7}{5525}\, x^8 - \frac{83}{3400}\, x^6 + \frac{2181}{11050}\, x^4 - \frac{149}{221}\, x^2 + 1$$

```
>
```

```
# Rassurons-nous !
```

```
> evalb(simplify(px-q)=0) ;
```

$$\mathit{true}$$

```
>
```

```
# Comparons avec un polynôme d'interpolation pris en 11 points,

# repartis dans l'intervalle [-5,5] a la Tchebyshev pour une

# << meilleure >> interpolation :

# on utilise l'application linéaire  x -> 5x

# réalisant une bijection [-1,1] sur [-5,5].

>

# La fonction evalf nécessaire sinon les calculs sont trop lourds !

> X := [seq(evalf(  5 * cos((2*i+1) * Pi / (2*n+2)) ), i=0..n)] ;
```

$$X := [4.949107210,\ 4.548159976,\ 3.778747872,\ 2.703204086,\ 1.408662782,\ 0.,\ -1.408662782,\ -2.703204086,\ -3.778747872,\ -4.548159976,\ -4.949107210]$$

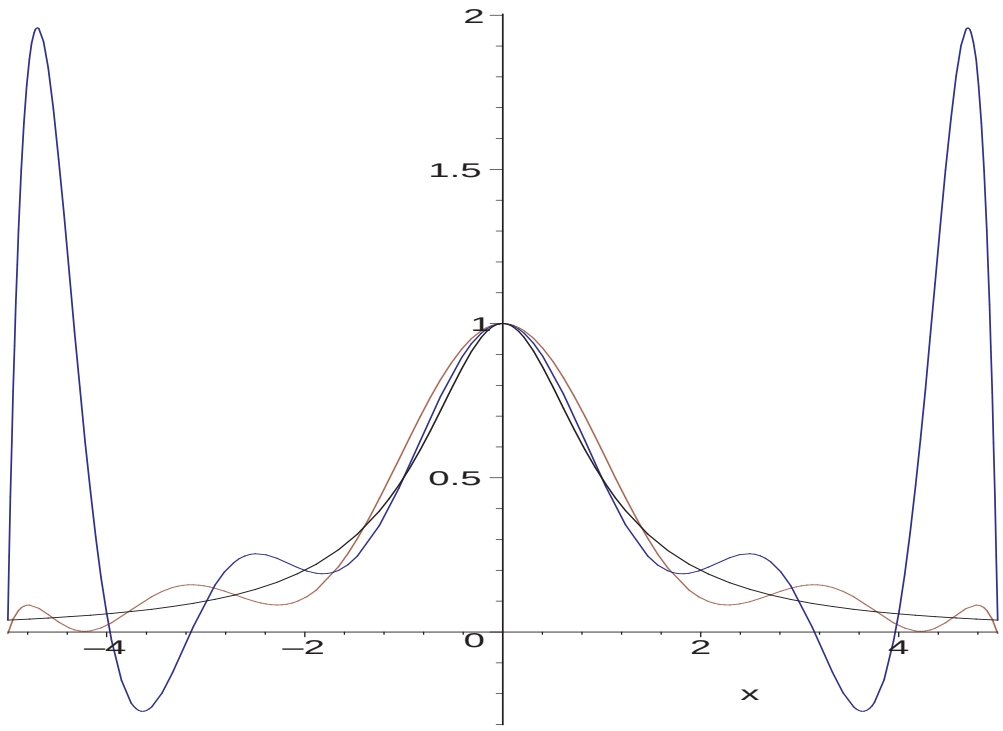
```
> Y := map(f, X) ;
```

$$Y := [.03922543544,\ .04611321154,\ .06544958589,\ .1203758761,\ .3350834931,\ 1.,\ .3350834931,\ .1203758761,\ .06544958589,\ .04611321154,\ .03922543544]$$

```
> q := interp(X, Y, x) ;
```

$$q := -.4775210895\,10^{-5}\,x^{10} + .60\,10^{-12}\,x^9 + 1.000000001 + .0003330709576\,x^8 + .54\,10^{-8}\,x - .34\,10^{-10}\,x^7 - .4990604633\,x^2 - .008540464549\,x^6 - .382\,10^{-8}\,x^3 + .59\,10^{-9}\,x^5 + .09830883112\,x^4$$

```
>

# On ne résiste pas au plaisir de faire un petit dessin...

>

> plot([f(x), px, q], x = -5..5, color=[black,blue,brown]);
```




```
> pour_le_fichier(%, ‘lagrange5‘) :
```

Bonus

Voici de la programmation assez rude (pour des débutants). Il s'agit de laisser le logiciel de calcul formel obtenir tout seul la formule

$$\Delta^n y_i = \sum_{j=0}^{n} \binom{n}{j} (-1)^{n-j} y_{i+j}$$

en itérant suffisamment l'opérateur $\Delta y_i = y_{i+1} - y_i$.

```
>  restart : read ‘my_config.mpl‘ : interface(echo=3) :
>  read ‘lagrange6.mpl‘ ;

# Commençons par l'opérateur de composition :

> f1 := cos ;
```

$$f1 := \cos$$

```
> f2 := x -> x + t ;
```

$$f2 := x \to x + t$$

```
> (f1 @ f2) ;
```

$$\cos@f2$$

```
> (f1 @ f2)(x) ;
```

$$\cos(x+t)$$

```
>

# Définissons les différences itérées.

# L'opérateur de composition rend l'écriture assez naturelle.

> tau := y -> unapply(y(i+1), i) ;
```

$$\tau := y \to \mathrm{unapply}(y(i+1),\, i)$$

```
> Id := y -> y ;
```

$$Id := y \to y$$

```
> Delta := tau-Id ;
```

$$\Delta := \tau - Id$$

```
>

# On essaye

> Delta(y) ;
```

$$(i \to \mathrm{y}(i+1)) - y$$

```
> Delta(y)(n) ;
```

$$\mathrm{y}(n+1) - \mathrm{y}(n)$$

```
>

# De plus en plus compliqué : on compose Delta avec elle-même.

# Voici quatre écritures possibles...

> (Delta(y))(n+1) - (Delta(y))(n) ;
```

$$\mathrm{y}(n+2) - 2\,\mathrm{y}(n+1) + \mathrm{y}(n)$$

```
> Delta( Delta(y) )(n) ;
```

$$\mathrm{y}(n+2) - 2\,\mathrm{y}(n+1) + \mathrm{y}(n)$$

```
> (Delta@Delta)(y)(n) ;
```

$$\mathrm{y}(n+2) - 2\,\mathrm{y}(n+1) + \mathrm{y}(n)$$

```
> (Delta@@2)(y)(n) ;
```

$$\mathrm{y}(n+2) - 2\,\mathrm{y}(n+1) + \mathrm{y}(n)$$

```
>
```

```
# ... et de manière plus générale

> D3y := (Delta@@3)(y) ; # Que c'est beau le calcul formel !!!
```

$$D3y := (i \to \mathrm{y}(i+3) - 2\,\mathrm{y}(i+2) + \mathrm{y}(i+1)) - (i \to \mathrm{y}(i+2) - \mathrm{y}(i+1)) + (i \to \mathrm{y}(i+1)) - y$$

```
> D3yi := (Delta@@3)(y)(i) ;
```

$$D3yi := \mathrm{y}(i+3) - 3\,\mathrm{y}(i+2) + 3\,\mathrm{y}(i+1) - \mathrm{y}(i)$$

```
> (Delta@@5)(y)(i) ;
```

$$\mathrm{y}(i+5) - 5\,\mathrm{y}(i+4) + 10\,\mathrm{y}(i+3) - 10\,\mathrm{y}(i+2) + 5\,\mathrm{y}(i+1) - \mathrm{y}(i)$$

```
>

# Enfin, obtenons le n-ième polynôme de Newton

# (polynôme d'interpolation en a, a+h, a+2h, ...)

> y := i -> f(a+i*h) ;  z := (x-a)/h ;
```

$$y := i \to \mathrm{f}(a + i\,h)$$

$$z := \frac{x-a}{h}$$

```
> n := 3 ;
```

$$n := 3$$

```
> px := add((Delta@@k)(y)(0)*binomial(z,k), k=0..n) :

> p := unapply(px, x) :

> seq(p(a+i*h), i=0..n) ;
```

$$\mathrm{f}(a),\ \mathrm{f}(a+h),\ \mathrm{f}(a+2\,h),\ \mathrm{f}(a+3\,h)$$

```
>

# ...et en guise de final

# (avec a=0 pour plus de lisibilité)

> a := 0 : limit(expand(px), h=0) ;
```

$$\frac{1}{6}\,(\mathrm{D}^{(3)})(f)(0)\,x^3 + \frac{1}{2}\,(\mathrm{D}^{(2)})(f)(0)\,x^2 + x\,\mathrm{D}(f)(0) + \mathrm{f}(0)$$

1.5 Polynômes de Bernstein

Les polynômes de Bernstein fournissent une preuve effective du théorème de Weierstrass : les fonctions continues définies sur un compact sont les limites uniformes des polynômes. Soit f une fonction réelle continue définie sur l'intervalle $[0,1]$. Pour tout $n \in \mathbb{N}$, on pose :

$$B_n(x) = \sum_{k=0}^{n} \binom{n}{k} f\Big(\frac{k}{n}\Big) x^k (1-x)^{n-k} \qquad \text{où} \qquad \binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k!}$$

Alors la suite des polynômes B_n converge uniformément vers f (formule démontrée en TD) :

$$\forall n \geqslant 2 \qquad ||B_n - f||_\infty \leqslant \frac{3}{2}\,\omega\left(f, \frac{1}{\sqrt{n}}\right)$$

où $w(f,h) = \max\limits_{|x-y|\leqslant h} |f(x)-f(y)|$ est le module de continuité (ou l'oscillation) de f en h. En particulier, si f est L-lipschitzienne, alors $\omega(f,h) \leqslant hL$, d'où

$$\forall n \geqslant 2 \qquad ||B_n - f||_\infty \leqslant \frac{3L}{2\sqrt{n}}$$

Mais il existe un résultat « plus fin » : si f est de classe C^2 sur $[0,1]$, alors pour tout $x \in [0,1]$ tel que $f''(x) \neq 0$, on a :

$$B_n(x) - f(x) \underset{n\to\infty}{\sim} \frac{x(1-x)f''(x)}{2n}$$

(voir A.C. Zaanen, «Continuity, Integration and Fourrier Theory», page 26, Springer Verlag 1989)

Grâce à cette équivalence (que nous vérifions en machines sur des exemples), on connaît la position des polynômes de Bernstein par rapport à la fonction f :
– si f est concave au voisinage de x, alors $f''(x) < 0$ et

$$f(x) > B_n(x) \text{ pour } n \text{ assez grand}\,;$$

– si f est convexe au voisinage de x, alors $f''(x) > 0$ et

$$f(x) < B_n(x) \text{ pour } n \text{ assez grand.}$$

```
> restart : interface(echo=3) :read ‘bernstein1.mpl‘;

# Les polynômes de Bernstein fournissent une preuve effective

# du théorème de Weierstrass : les fonctions continues définies
```

```
# sur un compact sont les limites uniformes des polynômes.
# De plus, la définition des polynômes de Bernstein est explicite
# et simple à programmer. Malheureusement, la suite des polynômes
# de Bernstein converge très très lentement...
>
> Bernstein1 := proc(f, n, x)
>   # f : la fonction à approximer sur l'intervalle [0,1]
>   # n : le degré du polynôme d'approximation
>   # x : une variable
>   # Cette fonction calcule le polynôme de Bernstein (en x)
>   # de degré n approximant la fonction f :
>     local k ;
>     RETURN(sum(binomial(n,k) * f(k/n) * x^k * (1-x)^(n-k), k=0..n))
:
> end :
>
# Testons...
> collect(Bernstein1(x->1/(x+1), 3, z), z) ;
```

$$1 - \frac{1}{20}\,z^3 + \frac{3}{10}\,z^2 - \frac{3}{4}\,z$$

```
# A comparer avec la fonction intégrée de Maple :
> bernstein(3, x->1/(x+1), z) ;
```

$$1 - \frac{1}{20}\,z^3 + \frac{3}{10}\,z^2 - \frac{3}{4}\,z$$

```
>
# Vérifions la procédure : remarquer que n est générique !!!
> B0 := Bernstein1(x -> 1   , n , x) ;
```

$$B0 := (1 + \frac{x}{1-x})^n\,(1-x)^n$$

```
> simplify(B0) ;
```

$$(-\frac{1}{-1+x})^n\,(1-x)^n$$

```
>
# Une seconde simplification avec l'option "symbolic" est
# nécessaire depuis la version V.5 de Maple ! C'est une horreur...
# Il faut dire que cela fonctionnait mieux avec Maple V.3 !!!
> B0 := simplify(B0, symbolic) ;
```

$$B0 := 1$$

```
>
> Bernstein2 := (f,n,x) -> simplify(Bernstein1(f,n,x), symbolic) :
>
> B1 := Bernstein2(x -> x   , n , x) ;
```

$$B1 := x$$

```
>
> B2 := Bernstein2(x -> x^2 , n , x) ;
```

$$B2 := \frac{x\,(n\,x+1-x)}{n}$$

```
# Visiblement, ce n'est pas x^2 !
>
> B3 := Bernstein2(x -> x^3 , n , x) ;
```

$$B3 := \frac{x\,(1+3\,n\,x-3\,x+n^2\,x^2-3\,n\,x^2+2\,x^2)}{n^2}$$

```
# Visiblement, ce n'est pas x^3 !!
>
# Regardons la convergence uniforme des polynômes de Bernstein
# sur cet exemple :
> f := x -> sin(4*x^2) ;
```

$$f := x \to \sin(4\,x^2)$$

```
>
# On ne résiste pas au plaisir de faire un petit dessin...
```

```
# Choisissons quelques polynômes de Bernstein.

>

# Remarque : si on utilise Bernstein2 alors il faut monter

# considérablement la précision des calculs...

# Mais avec Bernstein1, nul besoin !!!

# Explication : l'écriture développée des polynômes de Bernstein

# provoque une instabilité des calculs d'évaluation des polynômes.

# Contrairement à Bernstein2 qui donne une écriture développée,

# Bernstein1 donne une écriture semi-factorisée en x^j (1-x)^k,

# ce qui est salvateur dans notre situation !

>

#Digits := 40 : suite_poly := seq(Bernstein2(f, 2^i , x), i = 0..6) :

#Digits := 40 : suite_poly := seq(expand(Bernstein1(f, 2^i , x)), i =
0..6) :

> suite_poly := seq(Bernstein1(f, 2^i , x), i = 0..6) :

> suite_poly[1] ; suite_poly[2] ; evalf(suite_poly[3]) ;
evalf(suite_poly[4]) ;
```

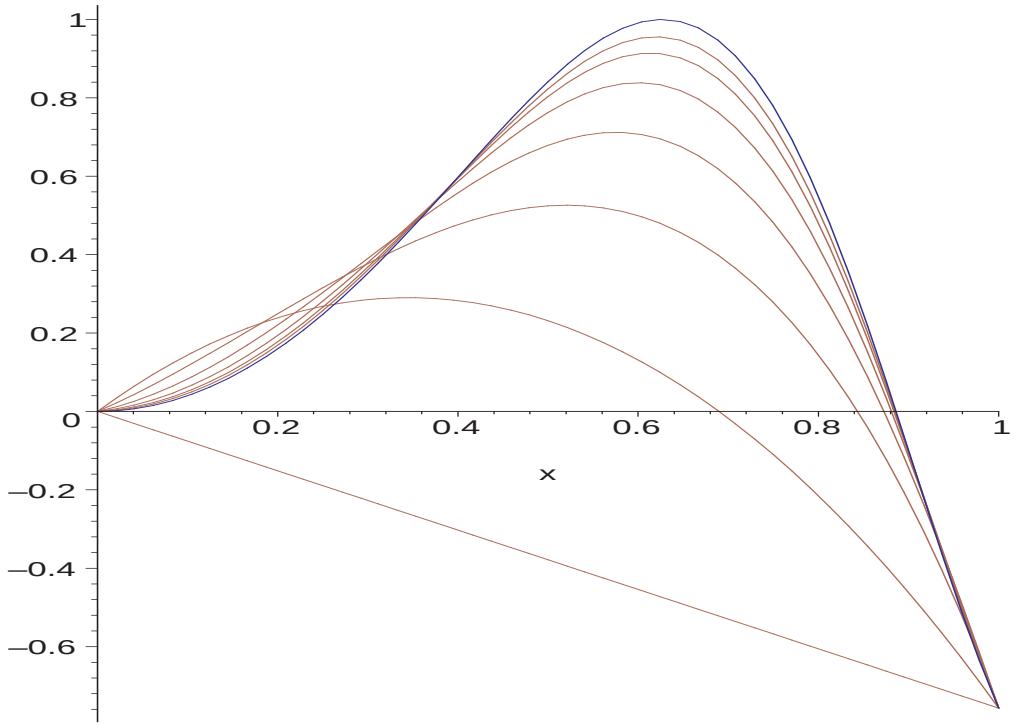
$$\sin(4)\, x$$

$$2\sin(1)\, x\,(1-x) + \sin(4)\, x^2$$

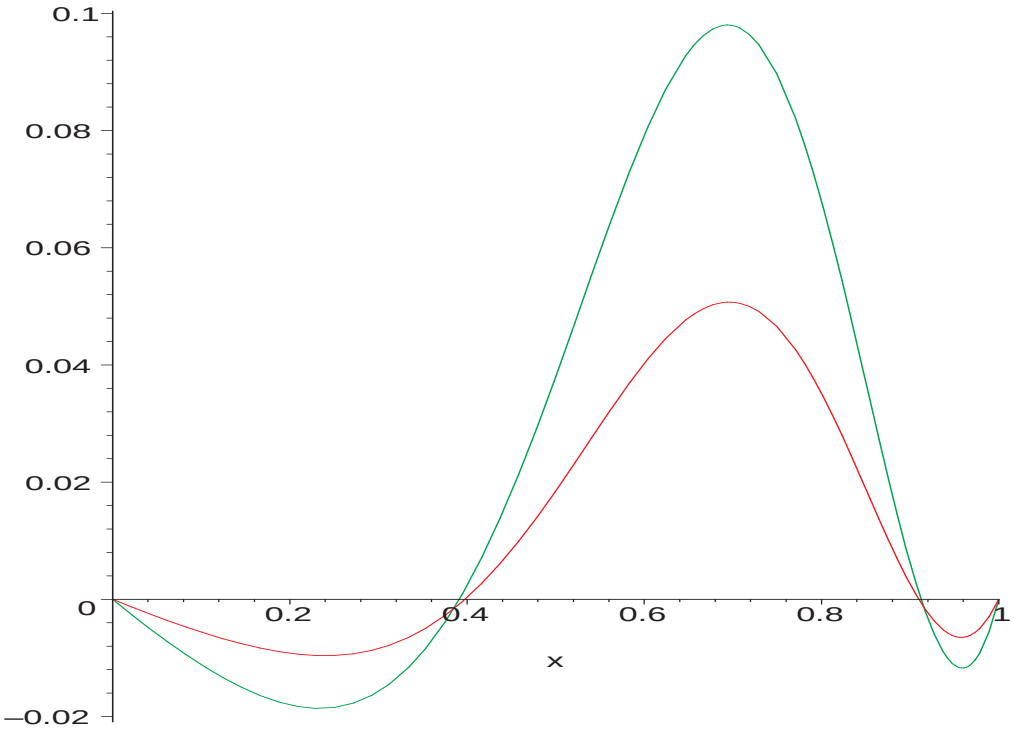
$$0.9896158372\, x\,(1.-1.\, x)^3 + 5.048825909\, x^2\,(1.-1.\, x)^2 + 3.112292788\, x^3\,(1.-1.\, x) - 0.7568024953\, x^4$$

$$0.4996745427\, x\,(1.-1.\, x)^7 + 6.927310860\, x^2\,(1.-1.\, x)^6 + 29.86494972\, x^3\,(1.-1.\, x)^5 + 58.90296894\, x^4\,(1.-1.\, x)^4 + 55.99807280\, x^5\,(1.-1.\, x)^3 + 21.78604951\, x^6\,(1.-1.\, x)^2 + 0.6320817340\, x^7\,(1.-1.\, x) - 0.7568024953\, x^8$$

```
>

> plot([f(x), suite_poly], x = 0..1, color=[blue,brown$7]) ;
```




```
> pour_le_fichier(%, ‘bernstein1‘) :

> plot([f(x)-suite_poly[-1], f(x)-suite_poly[-2]], x = 0..1) ;
```



```
> pour_le_fichier(%, ‘bernstein1‘) :

# On constate graphiquement une convergence linéaire (d’ordre 1) en

# fonction de l’indice des polynômes de Bernstein :

# si on multiplie n par 2 alors  || f - B_n || est divisée par 2
environ.

# Ceci laisse à penser à une amélioration possible (et plus <<
réaliste >>)

# de la majoration de || f - B_n || où on verra clairement un ordre

# de convergence uniforme d’ordre 1.

>

# De plus, la différence   f(x) - Bernstein(f,n,x)

# est équivalente quand n tend vers l’infini a    x(x-1)f’’(x)/2n

# Il est donc clair que la convergence simple est d’ordre 1 sur les

# points qui ne sont pas des points d’inflexion.

# En particulier, pour n assez grand et x dans ]0,1[,

# si le graphe de f est convexe alors f(x) < Bernstein(f,n,x),

# si le graphe de f est concave alors f(x) > Bernstein(f,n,x).

>

> f := x -> (x+1)^5 ;
```

$$f := x \to (x+1)^5$$

```
> l := limit( n*(f(x)-Bernstein2(f, n, x)) , n=infinity) ;
```

$$l := 10\,x\,(-1 + x^4 + 2\,x^3 - 2\,x)$$

```
> d := diff(f(x), x$2) * x * (x-1) / 2 ;
```

$$d := 10\,(x+1)^3\,x\,(-1+x)$$

```
> ‘l’equivalent est correct‘, evalb(simplify(l-d)=0) ;
```

$$l'equivalent\ est\ correct,\ true$$

```
>

>
```

```
#######################################################

### autres exemples avec Maple version 7 ou supérieure

>

> f := exp ;
```

$$f := \exp$$

```
> l := limit( n*(f(x)-Bernstein2(f, n, x)) , n=infinity) ;
```

$$l := -\frac{1}{2}\, e^{x}\, x + \frac{1}{2}\, e^{x}\, x^{2}$$

```
> d := diff(f(x), x$2) * x * (x-1) / 2 ;
```

$$d := \frac{1}{2}\, e^{x}\, x\, (-1 + x)$$

```
> 'l'equivalent est correct', evalb(simplify(l-d)=0) ;
```

$$l'equivalent\ est\ correct,\ true$$

```
>

###################################

>

## Voici comment calculer un équivalent en +infini

> f := x -> (x+1)^5 ;
```

$$f := x \to (x+1)^{5}$$

```
> series( leadterm(f(x)), x=infinity) ;
```

$$x^{5}$$

```
>

> l := series( leadterm(f(x)-Bernstein2(f, n, x)) , n=infinity) ;
```

$$l := \frac{20\, x^{4} - 10\, x - 20\, x^{2} + 10\, x^{5}}{n}$$

```
> d := diff(f(x), x$2) * x * (x-1) / (2*n) ;
```

$$d := \frac{10\, (x+1)^{3}\, x\, (-1 + x)}{n}$$

```
> 'l'equivalent est correct', evalb(simplify(l-d)=0) ;
```

$$l'equivalent\ est\ correct,\ true$$

```
>

>
```

```
## Autre exemple
```

```
> f := exp ;
```

$$f := \exp$$

```
> l := series( leadterm(f(x)-Bernstein2(f, n, x)) , n=infinity) ;
```

$$l := \frac{1}{2}\,\frac{e^x\,x\,(-1+x)}{n}$$

```
> d := diff(f(x), x$2) * x * (x-1) / (2*n) ;
```

$$d := \frac{1}{2}\,\frac{e^x\,x\,(-1+x)}{n}$$

```
> ‘l’equivalent est correct‘, evalb(simplify(l-d)=0) ;
```

$$l'equivalent\ est\ correct,\ true$$

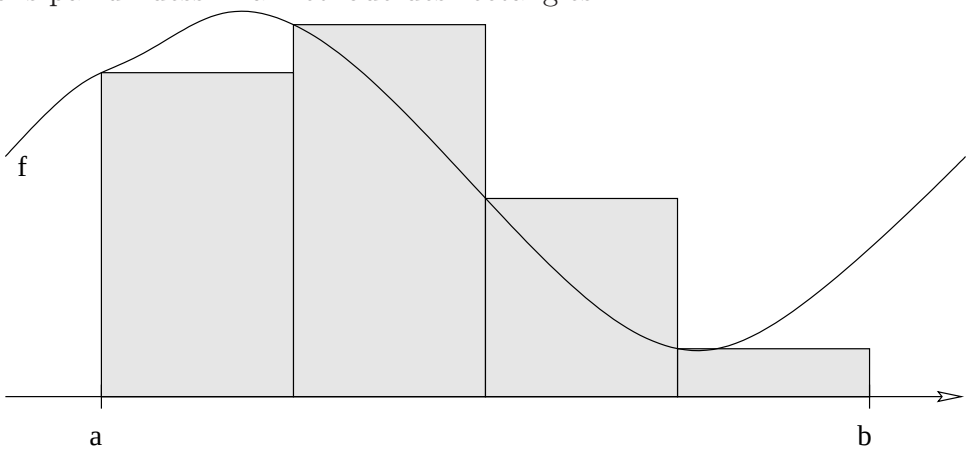
Chapitre 2

INTÉGRATION NUMÉRIQUE

Nous allons expérimenter quelques méthodes d'intégration numérique : les méthodes des rectangles à gauche, du point médian, des trapèzes et de Simpson sont les plus classiques. La méthode de Gauss, particulièrement étonnante, est traitée dans la section 3.5...

2.1 Définitions des méthodes classiques

Voici les listings de ces différentes méthodes ultra-classiques.
Rappelons par un dessin la méthode des rectangles :



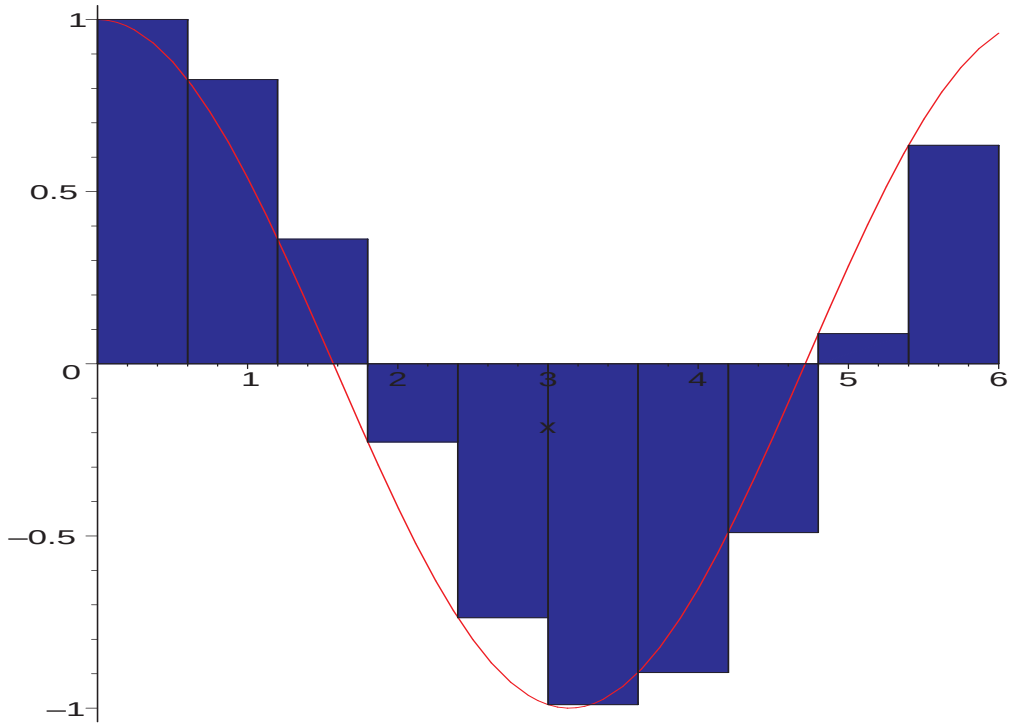
On approche l'intégrale $\int_a^b f$ par l'aire des rectangles grisés, ce qui a comme conséquence le fait suivant :

$$\int_a^b f = \lim_{n\to\infty} \frac{(b-a)}{n} \sum_{k=0}^{n-1} f\left(a + k\frac{(b-a)}{n}\right)$$

d'où la procédure :

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘rectang.mpl‘ ;

# Nous commençons par construire des sommes de Riemann
```

```
# (méthode des rectangles à gauche). Cela fournit
# une méthode peu performante (convergence d'ordre 1).
> rectangles1 := proc(f,a,b,n)
>     # f   : la fonction à intégrer
>     # a,b : les bornes de l'intervalle
>     # n   : le nombre de sous-intervalles
>     local p,k ;
>       # p : le pas de la méthode
>       p := (b-a)/n ;
>       RETURN( p * sum( f(a + k*p) , k = 0..n-1) ) ;
> end :
>
# Cet algorithme est déjà implémenté dans Maple.
> with(student):
>
> leftbox(cos(x), x=0..6, 10, shading=BLUE, color=RED) ;
```




```
> pour_le_fichier(%, `integration1`) :
>
> rectangles2 := proc(f,a,b,n)
>    RETURN(value(leftsum(f(x), x=a..b, n))) ;
> end :
>
# Essayons
> R1 := rectangles1(f, a, b, 2) ;
```

$$R1 := (\frac{b}{2} - \frac{a}{2})\,(\mathrm{f}(a) + \mathrm{f}(\frac{a}{2} + \frac{b}{2}))$$

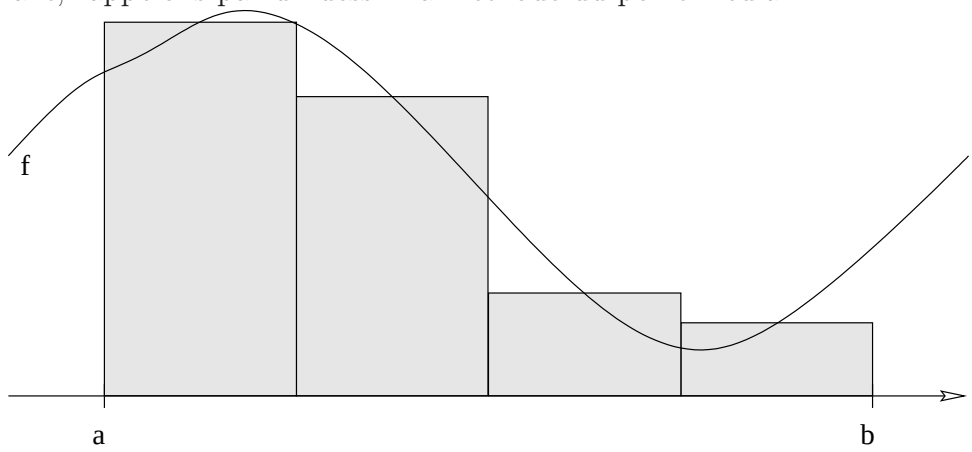
```
> R2 := rectangles2(f, a, b, 2) ;
```

$$R2 := (\frac{b}{2} - \frac{a}{2})\,(\mathrm{f}(a) + \mathrm{f}(\frac{a}{2} + \frac{b}{2}))$$

```
> simplify(R1-R2) ;
```

$$0$$

Maintenant, rappelons par un dessin la méthode du point médian :



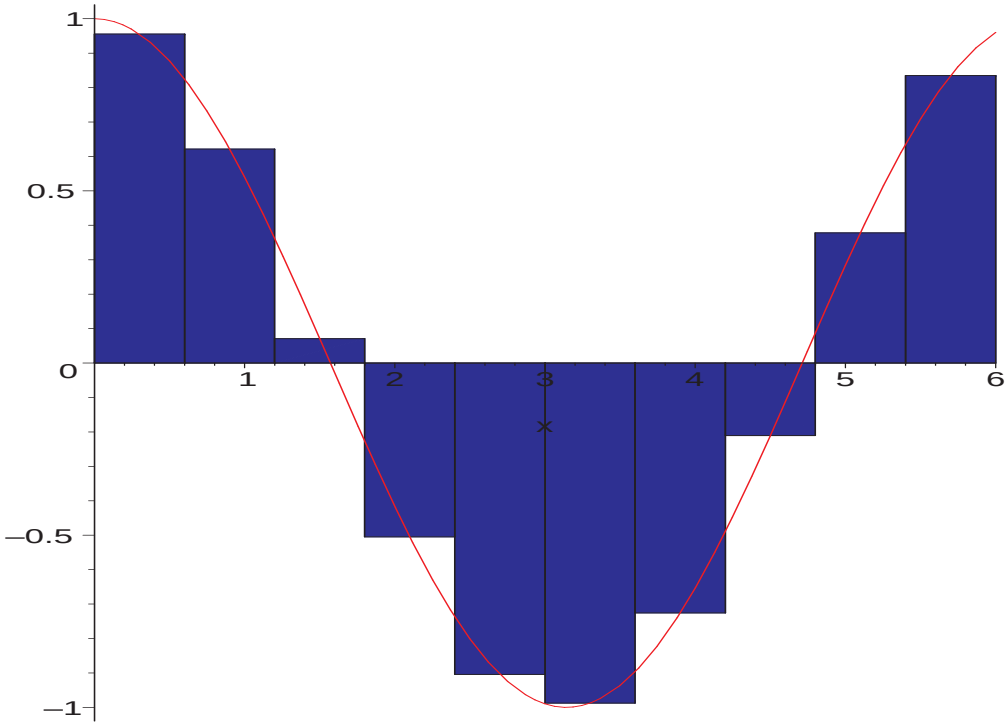
On approche l'intégrale $\int_a^b f$ par l'aire des rectangles grisés, ce qui a comme conséquence le fait suivant :

$$\int_a^b f = \lim_{n\to\infty} \frac{(b-a)}{n} \sum_{k=1}^{n} f\left(a + (k-0.5)\frac{(b-a)}{n}\right)$$

d'où la procédure :

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘median.mpl‘ ;

# On approche l’integrale par des sommes de Riemann

# (methode du point median, convergence d’ordre 2).

>

# On remarque que la méthode du point median est

# liée à la méthode des rectangles :

# median(f,a,b,n) = rectangles(f,a+p/2,b+p/2,n)  où p = (b-a)/n

>

> old_echo := interface(echo, echo=1)[1] :
                              lecture de rectang.mpl

>

> median1 := proc(f,a,b,n)

>     # f    : la fonction à integrer

>     # a, b : les bornes de l’intervalle

>     # n    : le nombre de sous-intervalles
```

```
>     local p ;
>       # p : le pas de la methode
>       p := (b-a)/n ;
>       RETURN( rectangles1(f,a+p/2,b+p/2,n) ) ;
> end :
>
# Cet algorithme est déjà implémenté dans Maple.
> with(student):
>
> middlebox(cos(x), x=0..6, 10, shading=BLUE, color=RED) ;
```




```
> pour_le_fichier(%, ‘integration2‘) :
>
> median2 := proc(f,a,b,n)
>       RETURN( value(middlesum(f(x), x=a..b, n)) ) ;
```

```
> end :

>

# Essayons

> M1 := median1(f, a, b, 2) ;
```

$$M1 := (\frac{b}{2} - \frac{a}{2})\,(\mathrm{f}(\frac{3\,a}{4} + \frac{b}{4}) + \mathrm{f}(\frac{a}{4} + \frac{3\,b}{4}))$$

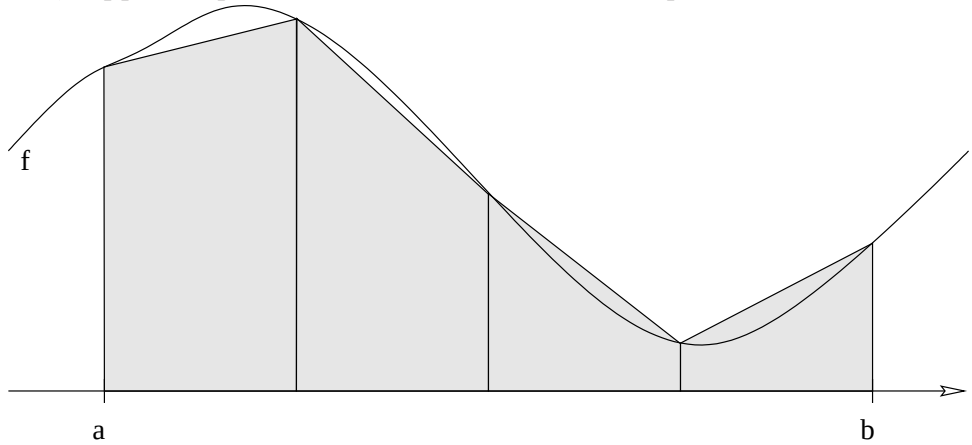
```
> M2 := median2(f, a, b, 2) ;
```

$$M2 := (\frac{b}{2} - \frac{a}{2})\,(\mathrm{f}(\frac{3\,a}{4} + \frac{b}{4}) + \mathrm{f}(\frac{a}{4} + \frac{3\,b}{4}))$$

```
> simplify(M1 - M2) ;
```

$$0$$

Maintenant, rappelons par un dessin la méthode des trapèzes :



On approche l'intégrale $\int_a^b f$ par l'aire des trapèzes grisés, ce qui a comme conséquence le fait suivant :

$$\int_a^b f = \lim_{n\to\infty} \frac{(b-a)}{2n} \sum_{k=0}^{n-1} \left[f\left(a + k\frac{(b-a)}{n}\right) + f\left(a + (k+1)\frac{(b-a)}{n}\right)\right]$$

ou encore

$$\int_a^b f = \lim_{n\to\infty} \frac{(b-a)}{n} \left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f\left(a + k\frac{(b-a)}{n}\right)\right]$$

d'où la procédure :

```
> restart : interface(echo=3) : read ‘trapez.mpl‘ ;

# On approche l’integrale par l’aire des trapezes
```

```
# (convergence d'ordre 2).
>
> trapezes1 := proc(f,a,b,n)
>    # f   : la fonction a integrer
>    # a,b : les bornes de l'intervalle
>    # n   : le nombre de sous-intervalles
>    local p,s,k ;
>      # p : le pas de la methode
>      p := (b-a)/n ;
>      s := ( f(a) + f(b) ) /2 ;
>      s := s + sum( f(a + k*p) , k = 1..n-1) ;
>      RETURN(s*p) ;
> end :
>
# Cet algorithme est deja implemente dans Maple.
>
> with(student):
> trapezes2 := proc(f,a,b,n)
>     RETURN( value(trapezoid(f(x), x=a..b, n)) ) ;
> end :
>
# Essayons
>
> T1 := trapezes1(f, a, b, 3) ;
```

$$T1 := (\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b) + \mathrm{f}(\frac{2}{3}\,a + \frac{1}{3}\,b) + \mathrm{f}(\frac{1}{3}\,a + \frac{2}{3}\,b))\,(\frac{1}{3}\,b - \frac{1}{3}\,a)$$

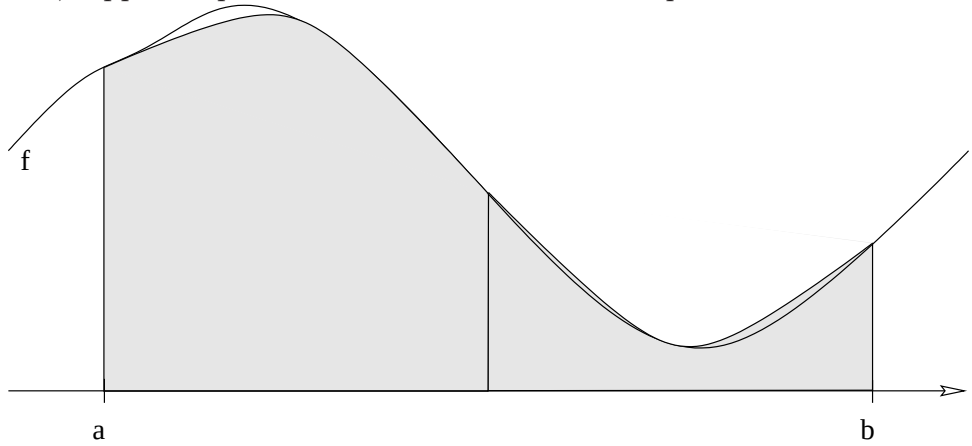
```
> T2 := trapezes2(f, a, b, 3) ;
```

$$T2 := \frac{1}{2}\,(\frac{1}{3}\,b - \frac{1}{3}\,a)\,(\mathrm{f}(a) + 2\,\mathrm{f}(\frac{2}{3}\,a + \frac{1}{3}\,b) + 2\,\mathrm{f}(\frac{1}{3}\,a + \frac{2}{3}\,b) + \mathrm{f}(b))$$

```
> simplify(T1-T2) ;
```

$$0$$

Maintenant, rappelons par un dessin la méthode de Simpson :



L'idée de cette méthode consiste à utiliser (sur un segment $[a, b]$) l'arc de parabole définie par les trois points d'abscisse a, $\frac{(a+b)}{2}$ et b, situés sur le graphe de la fonction f, comme courbe d'approximation de f, et à intégrer sur $[a, b]$ cette fonction du second degré. On obtient :

$$\frac{(b-a)}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

```
>  restart : read ‘my_config.mpl‘ : interface(echo=3) :
>  read ‘simpson.mpl‘ ;

# La méthode de Simpson (d’ordre de convergence 4) consiste à

# approcher la fonction que l’on veut intégrer par des arcs de

# paraboles s’appuyant sur trois points de sa courbe.

# Il faut donc diviser l’intervalle d’intégration en un nombre

# pair de "demi sous-intervalles", puis écrire l’équation de la

# parabole passant par les points (a, f(a)) , ((a+b)/2, f((a+b)/2)

# et (b, f(b)), et enfin intégrer l’expression obtenue

# entre a et b.

# Faisons faire tous ces calculs a Maple. C’est le polynôme
```

```
# d'interpolation de Lagrange qui définit le morceau de parabole
# cherché.
> X := [a,(a+b)/2,b] ;
```

$$X := [a,\ \frac{a}{2}+\frac{b}{2},\ b]$$

```
YY := map(f, X) ;
```

$$Y := [\mathrm{f}(a),\ \mathrm{f}(\frac{a}{2}+\frac{b}{2}),\ \mathrm{f}(b)]$$

```
>
>
# On demande d'interpoler...
> px := interp(X,Y,x) ;
```

$$px := \frac{(-4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2})+2\,\mathrm{f}(a)+2\,\mathrm{f}(b))\,x^2}{(a-b)^2} + \frac{(-3\,\mathrm{f}(b)\,a-\mathrm{f}(b)\,b+4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2})\,a+4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2})\,b-\mathrm{f}(a)\,a-3\,\mathrm{f}(a)\,b)\,x}{(a-b)^2} + \frac{\mathrm{f}(a)\,b^2+\mathrm{f}(b)\,a^2+\mathrm{f}(b)\,b\,a-4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2})\,b\,a+\mathrm{f}(a)\,b\,a}{(a-b)^2}$$

```
>
# ... puis l'intégrale sur [a,b].
# Comme l'expression "brute" est assez touffue, on la factorise.
> int_simpson := int(px, x=a..b) :
>
> int_simpson := factor(int_simpson) ;
```

$$int\_simpson := -\frac{1}{6}\,(a-b)\,(\mathrm{f}(b)+\mathrm{f}(a)+4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2}))$$

```
>
# On retrouve une expression familière, qui permet de justifier
# l'algorithme suivant :
```

```
> lecho := interface(echo, echo=1)[1] :
```

lecture de median.mpl
lecture de rectang.mpl
lecture de trapez.mpl

```
>
> simpson1 := proc(f,a,b,n)
>   RETURN( (2 * median1(f,a,b,n) + trapezes1(f,a,b,n)) / 3 ) ;
> end :
>
# Essayons
> S1 := simpson1(f, a, b, 1) ;
```

$$S1 := \frac{2}{3}(b-a)\,\mathrm{f}(\frac{a}{2}+\frac{b}{2}) + \frac{1}{3}(\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b))\,(b-a)$$

```
> simplify(S1 - int_simpson) ;
```

$$0$$

```
>
# Il existe une fonction simpson dans Maple.
> with(student):
> simpson2 := proc(f,a,b,n)
>      RETURN( value(simpson(f(x), x=a..b, 2*n)) ) ;
> end :
>
# Essayons
> S2 := simpson2(f, a, b, 1) ;
```

$$S2 := \frac{1}{3}(-\frac{a}{2}+\frac{b}{2})\,(\mathrm{f}(b) + \mathrm{f}(a) + 4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2}))$$

```
> simplify(S1-S2) ;
```

$$0$$

2.2 Comparaison des méthodes classiques

Nous allons retrouver les résultats du cours et du TD concernant les ordres des méthodes (de quadratures) d'intégration précédemment écrites et leurs ordres de convergence. En particulier, l'ordre de convergence vaut 1 de plus que l'ordre de la méthode.

1/ L'ordre d'une méthode M est le plus grand entier $d \in \mathbb{N}$ tel qu'une intégrale $\int_a^b f$ est calculée de manière exacte par la méthode M lorsque f est un polynôme de degré inférieur ou égal à d, c'est-à-dire :

$$d \in \mathbb{N} \text{ le plus grand tel que} \qquad \forall\, f \in \mathbb{R}_{\leqslant d}[X], \qquad \int_a^b f = M(f,a,b)$$

2/ Ordre de convergence d'une méthode composée M : soit e_n la différence entre $\int_a^b f$ et sa valeur approchée donnée par M composée sur n intervalles (de longueur $h = \frac{b-a}{n}$). On montre que e_n est équivalent à $\frac{\text{cst}}{n^w}$ pour un certain $w \in \mathbb{N}^*$ (la constante cst dépend de f, a et b). Cet entier w est l'ordre de convergence de M...

Par exemple, quand on approxime $\int_a^b f$ par une des ces méthodes composées (avec un pas h tendant vers 0), on montre que l'erreur commise est équivalente à (sous condition de non nullité de l'équivalent bien sûr !) :

$$\begin{aligned}
\frac{f(b)-f(a)}{2}h &\quad \text{pour la méthode des rectangles,}\\
\frac{f'(b)-f'(a)}{24}h^2 &\quad \text{pour la méthode des points médians,}\\
-\frac{f'(b)-f'(a)}{12}h^2 &\quad \text{pour la méthode des trapèzes,}\\
-\frac{f'''(b)-f'''(a)}{2880}h^4 &\quad \text{pour la méthode de Simpson.}
\end{aligned}$$

```
> restart : read `my_config.mpl` : interface(echo=3) :
> read `integration5.mpl` ;

> ancien_echo := interface(echo, echo=1)[1] :
                          lecture de simpson.mpl
                          lecture de median.mpl
                          lecture de rectang.mpl
                          lecture de trapez.mpl

>

##########

########## PARTIE I
```

```
##########
# Voici une procédure calculant l'ordre d'une méthode
# (de quadrature) d'intégration numérique.
> ordre := proc(m)
>     # m : méthode (procédure) prenant comme arguments
>     #      f   : la fonction à intégrer
>     #      a,b : les deux bornes d'intégration
>     #      n   : nombre de sous-intervalles
>     # m doit faire du calcul algébrique, non numérique !
>     local f,a,b,n,i,x ;
>     # f : la fonction x -> x^i
>     # a,b,n sont indéterminées et le résultat ne dépend pas d'eux !
>     # Bravo le calcul formel !
>        a := 'a' : b:= 'b' : n := 'n' :
>        for i from 0 do
>           f := unapply(x^i, x) ;
>           if simplify(int(f(x), x=a..b) - m(f,a,b,n)) <> 0
>                then RETURN(i-1) ;
>           fi ;
>        od ;
> end :
>
# Testons.
> ordre_rectangles := ordre((f,a,b,n) -> rectangles1(f,a,b,n,n)) ;
```

$$ordre\_rectangles := 0$$

```
> ordre_median := ordre((f,a,b,n) -> median1(f,a,b,n,n)) ;
```

$$ordre\_median := 1$$

```
> ordre_trapezes := ordre((f,a,b,n) -> trapezes1(f,a,b,n,n+1)) ;
```

$$ordre\_trapezes := 1$$

```
> ordre_simpson := ordre((f,a,b,n) -> simpson1(f,a,b,n,2*n+1)) ;
```

$$ordre\_simpson := 3$$

```
>

##########

########## PARTIE II

##########

# Déterminons les ordres de convergences de ces méthodes par

# une expérimentation numérique sur un (seul, certes !) exemple...

>

> f := exp ;
```

$$f := \exp$$

```
aa := -1.0 ;
```

$$a := -1.0$$

```
bb := 1.0 ;
```

$$b := 1.0$$

```
>

> Digits := 15 : iintegrale := int(f(x), x = a..b) ;
```

$$integrale := 2.35040238728760$$

```
>

# Testons différentes manières de calculer cette intégrale

# en découpant l'intervalle [a,b] en 9 sous-intervalles.

> n := 9 ;
```

$$n := 9$$

```
> r[1] := evalf(integrale - rectangles1(f,a,b,n)) ;
```

$$r_1 := 0.25149133450747$$

```
> m[1] := evalf(integrale - median1(f,a,b,n)) ;
```

$$m_1 := 0.00482926221274$$

```
> t[1] := evalf(integrale - trapezes1(f,a,b,n)) ;
```

$$t_1 := -0.00966448630226$$

```
> s[1] := evalf(integrale - simpson1(f,a,b,n)) ;
```

$$s_1 := -0.198729226\,10^{-5}$$

```
>

# On multiplie le nombre de sous-intervalles par 10.

# On constate alors l'ordre de convergence des différentes méthodes.

> r[2] := evalf(integrale - rectangles1(f,a,b,10*n)) ;
```

$$r_2 := 0.02601885849896$$

```
> m[2] := evalf(integrale - median1(f,a,b,10*n)) ;
```

$$m_2 := 0.00004836149249$$

```
> t[2] := evalf(integrale - trapezes1(f,a,b,10*n)) ;
```

$$t_2 := -0.00009672358202$$

```
> s[2] := evalf(integrale - simpson1(f,a,b,10*n)) ;
```

$$s_2 := -0.19902\,10^{-9}$$

```
>

# et maintenant, on constate les ordres de convergences :

# ils sont respectivement égaux à 1,2,2,4 :

> seq( log( k[1] / k[2] ) / log(10.0) ,  k=[r,m,t,s] ) ;
```

$$0.985234786166155,\ 1.99938109156295,\ 1.99964640323737,\ 3.99936501917327$$

```
>

##########

########## PARTIE III

##########

# On va essayer de retrouver formellement les formules

# classiques (cf TD) donnant des équivalents des erreurs globales

# commises par ces quatres méthodes composées...

# ATTENTION, les calculs suivants sont faits uniquement pour
```

```
# les monômes x^d, générique en d tout de même !
# (On pourra par la suite utiliser la linéarité des opérations
# pour généraliser le résultat...)
>
> (a,b,h,d) := 'a','b','h','d' : nn := (b-a)/h ;
```

$$n := \frac{b-a}{h}$$

```
ff := x -> x^d ;
```

$$f := x \to x^d$$

```
>
# Depuis Maple 9.5, il faut une étape supplémentaire pour calculer
# l'intégrale de x -> x^d entre a et b !!! :-(
> primitive := unapply(int(f(x), x), x) ;
```

$$primitive := x \to \frac{x^{(d+1)}}{d+1}$$

```
> integrale := primitive(b) - primitive(a) ;
```

$$integrale := \frac{b^{(d+1)}}{d+1} - \frac{a^{(d+1)}}{d+1}$$

```
>
> erreur_glob_rectangle := integrale - rectangles1(f,a,b,n) :
> erreur_glob_median := integrale - median1(f,a,b,n) :
> erreur_glob_trapeze := integrale - trapezes1(f,a,b,n) :
> erreur_glob_simpson := integrale - simpson1(f,a,b,n) :
>
> equivalent := proc(erreur, o)
>   local eq ;
>     eq := taylor(erreur, h=0, o+1) ;
>     eq := simplify(expand(eq)) ;  # Sacré Maple,
>     RETURN(factor(eq)) ;          # il faut tout lui dire...
```

```
> end :

>

> equivalent(erreur_glob_rectangle, 1) ;
```

$$(-\frac{a^d}{2}+\frac{b^d}{2})\,h+\mathrm{O}(h^2)$$

```
# on voit apparaître   1/2 (f(b)-f(a)) h

>

> equivalent(erreur_glob_median, 2) ;
```

$$-\frac{(a^{(d-1)}-b^{(d-1)})\,d}{24}\,h^2+\mathrm{O}(h^3)$$

```
# on voit apparaître   1/24 (f'(b)-f'(a)) h^2

>

> equivalent(erreur_glob_trapeze, 2) ;
```

$$\frac{(a^{(d-1)}-b^{(d-1)})\,d}{12}\,h^2+\mathrm{O}(h^3)$$

```
# on voit apparaître   -1/12 (f'(b)-f'(a)) h^2

>

> equivalent(erreur_glob_simpson, 4) ;
```

$$-\frac{(d-1)\,(d-2)\,(b^{(d-3)}-a^{(d-3)})\,d}{2880}\,h^4+\mathrm{O}(h^5)$$

```
# on voit apparaître   -1/2880 (f'''(b)-f'''(a)) h^4

>

##########

########## PARTIE IV

##########

# On va essayer de retrouver formellement les formules

# classiques (cf TD) donnant des équivalents des erreurs globales

# commises par ces quatres méthodes composées...

# ATTENTION, les calculs suivants ne sont pas très bien justifiés !

>
```

```
> f := 'f' : aa := 'a' : bb := 'b' : cc := 'c' : hh := 'h' :
>
> integrale_elem := int(f(x), x=c..c+h) :
> erreur_elem_rectangle := integrale_elem - rectangles1(f, c, c+h, 1);
```

$$erreur\_elem\_rectangle := \int_c^{c+h} \mathrm{f}(x)\,dx - h\,\mathrm{f}(c)$$

```
> erreur_elem_trapeze   := integrale_elem - trapezes1(f, c, c+h, 1) ;
```

$$erreur\_elem\_trapeze := \int_c^{c+h} \mathrm{f}(x)\,dx - (\frac{1}{2}\,\mathrm{f}(c) + \frac{1}{2}\,\mathrm{f}(c+h))\,h$$

```
> erreur_elem_median    := integrale_elem - median1(f, c, c+h, 1) ;
```

$$erreur\_elem\_median := \int_c^{c+h} \mathrm{f}(x)\,dx - h\,\mathrm{f}(c + \frac{h}{2})$$

```
> erreur_elem_simpson   := integrale_elem - simpson1(f, c, c+h, 1) ;
```

$$erreur\_elem\_simpson := \int_c^{c+h} \mathrm{f}(x)\,dx - \frac{2}{3}\,h\,\mathrm{f}(c + \frac{h}{2}) - \frac{1}{3}\,(\frac{1}{2}\,\mathrm{f}(c) + \frac{1}{2}\,\mathrm{f}(c+h))\,h$$

```
>
# On peut alors obtenir l'ordre de convergence de chaque
# méthode composée. Les calculs suivant permettent d'obtenir
# un équivalent de l'erreur globale de chaque méthode composée.
# (Depuis Maple 9.5, il faut y aller en douceur !)
>
> erreur := int(erreur_elem_rectangle/h, c) :
> erreur_glob_rectangle := subs(c=b, erreur) - subs(c=a, erreur) :
>
> erreur := int(erreur_elem_trapeze/h, c) :
> erreur_glob_trapeze   := subs(c=b, erreur) - subs(c=a, erreur) :
>
> erreur := int(erreur_elem_median/h, c) :
> erreur_glob_median    := subs(c=b, erreur) - subs(c=a, erreur) :
>
```

```
> erreur := int(erreur_elem_simpson/h, c) :
> erreur_glob_simpson   := subs(c=b, erreur) - subs(c=a, erreur) :
>
# Pour lire les ordres de convergence, il suffit de faire un
# développement de Taylor des erreurs globales.
>
> taylor(erreur_glob_rectangle, h=0, 3) ;
```

$$(\frac{1}{2}\,\mathrm{f}(b) - \frac{1}{2}\,\mathrm{f}(a))\,h + \mathrm{O}(h^2)$$

```
> taylor(erreur_glob_trapeze  , h=0, 4) ;
```

$$(-\frac{1}{12}\,\mathrm{D}(f)(b) + \frac{1}{12}\,\mathrm{D}(f)(a))\,h^2 + \mathrm{O}(h^3)$$

```
> taylor(erreur_glob_median   , h=0, 4) ;
```

$$(\frac{1}{24}\,\mathrm{D}(f)(b) - \frac{1}{24}\,\mathrm{D}(f)(a))\,h^2 + \mathrm{O}(h^3)$$

```
> taylor(erreur_glob_simpson  , h=0, 6) ;
```

$$(-\frac{1}{2880}\,(\mathrm{D}^{(3)})(f)(b) + \frac{1}{2880}\,(\mathrm{D}^{(3)})(f)(a))\,h^4 + \mathrm{O}(h^5)$$

```
>
# Constatons plusieurs phenomènes :
# 1/ On peut corriger facilement la méthode des rectangles
#    en lui ajoutant simplement    (f(b) - f(a)).h / 2
#    afin d'éliminer le terme dominant de l'erreur commise.
#    Ceci donne une méthode d'ordre de convergence plus grand
#    que celui de la méthode des rectangles...
#    ...Mais en fait cette méthode corrigée n'est autre que
#    la méthode des trapèzes !
# 2/ On peut combiner linéairement les méthodes des points médians
#    et des trapèzes pour éliminer les termes dominants des
#    erreurs commises :    2/3 . median  +  1/3 . trapeze
```

```
#    Ceci donne une méthode d'ordre de convergence plus grand
#    que celui des deux méthodes...
#    ...Mais en fait cette combinaison n'est autre que la méthode
#    de Simpson !
```

2.3 Méthode de Newton-Cotes

Pour calculer une approximation de $\int_a^b f$, on généralise la méthode de Simpson. Cette généralisation dite de Newton-Cotes, consiste à interpoler une fonction f par un polynôme de degré quelconque (mais déterminé à l'avance), en des points d'abscisses équidistantes, puis à intégrer ce polynôme d'interpolation.

> Remarque. Pour la méthode de Newton-Cotes (ouverte ou fermée), quel que soit le degré du polynôme d'interpolation, l'ordre de convergence vaut toujours 1 de plus que l'ordre de la méthode car cette méthode est basée sur l'interpolation polynomiale...

On verra (et c'est quand même très surprenant) que, une fois fixé le nombre d'appels de la fonction f, il est préférable d'interpoler la fonction f par un polynôme de degré pair $2k$ plutôt que par le polynôme de degré impair $2k+1$...

```
>  restart : read ‘my_config.mpl‘ : interface(echo=3) :
>  read ‘integration6.mpl‘ ;

> read ‘newcoat.mpl‘ ;

> integ_interpol := proc(f,a,b,d)

>   # f    : la fonction à integrer

>   # a, b : bornes de l’intervalle

>   # d    : le degré du polynôme d’interpolation

>   # Cette procédure calcule l’integrale sur [a,b] du polynôme

>   # d’interpolation de degré d passant par des points

>   # d’abscisses équiréparties a=x0 < x1 <... < xd=b

>   local h,X,Y,j;

>     # h : distance entre deux abscisses

>     h := (b-a)/d ;

>     X := [seq(a+h*j, j=0..d)] ;

>     Y := map(f, X) ;

>     RETURN( int(interp(X, Y, x), x=a..b) );

> end :

>
```

```
# Verifions rapidement les cas où d=1 (methode des trapezes)
# et où d=2 (methode de Simpson).
> factor(integ_interpol(f, a, b, 1)) ;
```

$$-\frac{1}{2}\left(\mathrm{f}(a)+\mathrm{f}(b)\right)(-b+a)$$

```
> factor(integ_interpol(f, a, b, 2)) ;
```

$$-\frac{1}{6}\left(\mathrm{f}(b)+\mathrm{f}(a)+4\,\mathrm{f}(\frac{a}{2}+\frac{b}{2})\right)(-b+a)$$

```
>
# La même méthode, mais composée (n fois)...
> NewtonCotes := proc(f,a,b,n,d)
>   # n : le nombre de sous-intervalles
>   # Cette procédure calcule la méthode composée déduite
>   # de integ_interpol (ci-dessus).
>   local p,k ;
>     # p : pas de la méthode
>     p := (b-a)/n ;
>     RETURN( sum( integ_interpol(f, a+(k-1)*p, a+k*p, d), k=1..n));
> end :
>
# Vérifions encore les cas où d=1 (méthode des trapèzes)
# et où d=2 (méthode de Simpson).
>
> ancien_echo := interface(echo, echo=1)[1] :
                          lecture de simpson.mpl
                          lecture de median.mpl
                          lecture de rectang.mpl
                          lecture de trapez.mpl
>
```

```
> n := 10 :
> t := NewtonCotes(f, a, b, n, 1) : ## méthode des trapèzes
> simplify(t - trapezes1(f,a,b,n))=0 ;
```

$$0 = 0$$

```
> s := NewtonCotes(f, a, b, n, 2) : ## méthode de Simpson
> simplify(s - simpson1(f,a,b,n))=0 ;
```

$$0 = 0$$

```
>
###
### Partie I
###
# Faisons du calcul littéral pour déterminer l'ordre
# de la méthode de quadrature en fonction de d.
> ordre := proc(d)
>   # d : degré du polynôme d'interpolation dans
>   #     la méthode de Newton-Cotes
>   local f,a,b,i,x,n,nc ;
>     # f : la fonction x -> x^i
>     # a,b,n sont indeterminees et le resultat ne depend pas d'eux !
>     a := 'a' : b := 'b' : n := 'n' :
>     for i from 0 do
>       f := unapply(x^i, x) ;
>       nc := NewtonCotes(f,a,b,n,d) ;
>       if simplify(int(f(x), x=a..b) - nc) <> 0 then RETURN(i-1) fi ;
>     od ;
> end :
>
```

```
# Testons.

> for d from 1 to 5 do

>     (degre_interpolation =  d , '=>' , ordre_methode = ordre(d)) ;

> od ;
```

$$degre\_interpolation = 1, =>, ordre\_methode = 1$$
$$degre\_interpolation = 2, =>, ordre\_methode = 3$$
$$degre\_interpolation = 3, =>, ordre\_methode = 3$$
$$degre\_interpolation = 4, =>, ordre\_methode = 5$$
$$degre\_interpolation = 5, =>, ordre\_methode = 5$$

```
>

###

### Partie II

###

# Constatons expérimentalement les ordres de convergences en fonction

# du degré des polynômes d'interpolation, avec la contrainte d'un
nombre

# d'appels d'une la fonction f en n+1 points d'un

# l'intervalle [a,b].

>

> erreur := proc(f, a, b, n, d)

>     RETURN( int(f(x), x = a..b) - NewtonCotes(f,a,b,n,d) ) ;

> end :

>

#f := x -> 1/(1+10*x^2) :

> f := exp :

> a := -1.0 ;
```

$$a := -1.0$$

```
bb := 1.0 ;
```

$$b := 1.0$$

```
iint(f(x), x = a..b) ;
```

$$2.350402387$$

```
>

# Testons différentes manières de calculer cette intégrale

# avec exactement 9 appels de la fonction f.

# La précision des calculs doit être fortement augmentée

# si l'on veut des résultats cohérents !

> Digits := 40 :

>

> n := 8 ;
```

$$n := 8$$

```
> t[1] := erreur(f,a,b, n,   1) ;
```

$$t_1 := -0.012228946297607355468327086154679953799$$

```
## méthode des trapèzes

> s[1] := erreur(f,a,b, n/2, 2) ;
```

$$s_1 := -0.000050629954676553724298024676239530552$$

```
## méthode de Simpson

> q[1] := erreur(f,a,b, n/4, 4) ;
```

$$q_1 := -0.11756454781422372842227036745046651\,10^{-5}$$

```
## polynôme interpolateur de degré 4

> h[1] := erreur(f,a,b, n/8, 8) ;
```

$$h_1 := -0.12321877468637589382470103676201\,10^{-8}$$

```
## polynôme interpolateur de degré 8

>

# SUR CET EXEMPLE, on constate que le degré de l'interpolation a

# beaucoup plus d'importance que le nombre de sous-intervalles.

# MAIS ATTENTION AU PHENOMENE DE RUNGE,

# par exemple, avec f := x -> 1/(1+10*x^2) !
```

```
>

# Maintenant, on multiplie le nombre de sous-intervalles par 10 :

> n := 10*n ;
```

$$n := 80$$

```
> t[2] := erreur(f,a,b, n,   1) ;
```

$$t_2 := -0.000122415515848631874796495838710990958$$

```
## méthode des trapèzes

> s[2] := erreur(f,a,b, n/2, 2) ;
```

$$s_2 := -0.5100320133753119449494117761431\,10^{-8}$$

```
## méthode de Simpson

> q[2] := erreur(f,a,b, n/4, 4) ;
```

$$q_2 := -0.12140539064449209795449071 32\,10^{-11}$$

```
## polynôme interpolateur de degré 4

> h[2] := erreur(f,a,b, n/8, 8) ;
```

$$h_2 := -0.1416326570222506833 03\,10^{-18}$$

```
## polynôme interpolateur de degré 8

>

# et maintenant, on constate les ordres de convergences :

# ils sont respectivement égaux à 2,4,6,10 normalement :

> Digits := 4 :

>

> seq( log( k[1] / k[2] ) / log(10.0) ,  k=[t,s,q,h] ) ;
```

$$1.999, 3.996, 5.983, 9.939$$

```
>

###

### PARTIE III

###

# Bizarrement (voir le cours d'analyse numérique of course),
```

```
# il est préférable que ce degré d'interpolation d soit pair :

# l'erreur est deux fois moindre en prenant d=2k

# à la place de d=2k+1 !!!

> Digits := 40 :

>

> n := 60 ;
```

$$n := 60$$

```
> m[1] := erreur(f,a,b, n,   1) ;
```

$$m_1 := -0.00021762582059897376526656185453143637 9$$

```
## méthode des trapèzes

> m[2] := erreur(f,a,b, n/2, 2) ;
```

$$m_2 := -0.16118597556710888538631591185371\, 10^{-7}$$

```
## méthode de Simpson

> m[3] := erreur(f,a,b, n/3, 3) ;
```

$$m_3 := -0.36262048531660975081104955075203\, 10^{-7}$$

```
## polynôme interpolateur de degré 3

> m[4] := erreur(f,a,b, n/4, 4) ;
```

$$m_4 := -0.68196107530603453404535312 81\, 10^{-11}$$

```
## polynôme interpolateur de degré 4

> m[5] := erreur(f,a,b, n/5, 5) ;
```

$$m_5 := -0.14647107539972048120055502703\, 10^{-10}$$

```
## polynôme interpolateur de degré 5

> m[6] := erreur(f,a,b, n/6, 6) ;
```

$$m_6 := -0.38328900854266278912025 48\, 10^{-14}$$

```
## polynôme interpolateur de degré 6

>

> Digits := 4 :

>
```

```
> rapport_entre_degres_3_et_2 := m[3] / m[2] ;
```

$$rapport\_entre\_degres\_3\_et\_2 := 2.249$$

```
>

> rapport_entre_degres_5_et_4 := m[5] / m[4] ;
```

$$rapport\_entre\_degres\_5\_et\_4 := 2.148$$

```
>

>

###

### Partie IV

###

# Même exercice que pour la partie II, mais en symbolique :

# ici, on prend seulement f(x) = x^d, mais générique en d tout de même
!

>

> (a,b,h,d) := 'a','b','h','d' :

>

> f := x -> x^d ;
```

$$f := x \to x^d$$

```
nn := (b-a)/h ;
```

$$n := \frac{b-a}{h}$$

```
> primit := unapply(int(f(x),x), x) :

>

> er := series(primit(b)-primit(a) - NewtonCotes(f,a,b,n,1), h=0, 4)
:

> er1 := factor(simplify(expand(er))) ;
```

$$er1 := \frac{(a^{(d-1)} - b^{(d-1)})\, d}{12}\, h^2 + \mathrm{O}(h^3)$$

```
>
```

```
> er := series(primit(b)-primit(a) - NewtonCotes(f,a,b,n/2,2), h=0,
8) :

> er2 := factor(simplify(expand(er))) ;
```

$$er2 := -\frac{(d-1)\,(d-2)\,(b^{(d-3)} - a^{(d-3)})\,d}{180}\,h^4 + \mathrm{O}(h^5)$$

```
>

> er := series(primit(b)-primit(a) - NewtonCotes(f,a,b,n/3,3), h=0,
10) :err3 := factor(simplify(expand(er))) ;
```

$$er3 := -\frac{(d-1)\,(d-2)\,(b^{(d-3)} - a^{(d-3)})\,d}{80}\,h^4 + \mathrm{O}(h^5)$$

```
>

##er := series(primit(b)-primit(a) - NewtonCotes(f,a,b,n/4,4), h=0,
14) :

##er4 := factor(simplify(expand(er))) ;

>

##er := series(primit(b)-primit(a) - NewtonCotes(f,a,b,n/5,5), h=0,
16) :

##er5 := factor(simplify(expand(er))) ;
```

2.4 Accélération de convergence

Considérons une fonction f dont la limite en 0 est *Graal*. Si on suppose que la convergence de f est d'ordre d alors on a

$$f(h) = Graal + a_d\ h^d + o(h^d) \qquad a_d \in \mathbb{R} \setminus \{0\}$$

Fixons $m \in \mathbb{R} \setminus \{1\}$. On a alors $f(mh) = Graal + a_d\ m^d\ h^d + o(h^d)$. Une combinaison linéaire (barycentrique, car la somme des coefficients vaut 1) de $f(mh)$ et $f(h)$ permet d'éliminer les termes en h^d :

$$acc_f(h) := \frac{1}{1-m^d} f(mh) - \frac{m^d}{1-m^d} f(h) = Graal + o(h^d)$$

On obtient alors une nouvelle méthode acc_f dont l'ordre de convergence est supérieur à celui de f.

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘integration7.mpl‘ ;

# On illustre ici le procédé d’accélération sur des méthodes

# d’intégration numérique élémentaires, dont on connaît bien

# les ordres de convergence.

>

> vieil_echo := interface(echo, echo=1) :
                              lecture de newcoat.mpl
                              lecture de simpson.mpl
                              lecture de median.mpl
                              lecture de rectang.mpl
                              lecture de trapez.mpl

>

# Voici la formule d’accélération qui correspond à deux

# approximations (par la même méthode bien sûr !) où le pas

# de la seconde approximation est deux fois plus petit que

# celui de la première.

> acceleration := ( 2^ordre_convergence * approx2 - 1 * approx1 )

>               / ( 2^ordre_convergence           - 1 ) ;
```

$$acceleration := \frac{2^{ordre\_convergence}\, approx2 - approx1}{2^{ordre\_convergence} - 1}$$

```
>
# La méthode des rectangles est d'ordre de convergence 1
# On demande de plus une forme factorisée de l'accélération
# plus lisible qu'une forme brute.
> ordre_convergence := 1 ;
```

$$ordre\_convergence := 1$$

```
> approx1 := rectangles1(f,a,b,1) ;
```

$$approx1 := (b-a)\,\mathrm{f}(a)$$

```
> approx2 := rectangles1(f,a,b,2) ;
```

$$approx2 := (\frac{1}{2}\,b - \frac{1}{2}\,a)\,(\mathrm{f}(a) + \mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b))$$

```
> factor(acceleration) ;
```

$$-\mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b)\,(-b+a)$$

```
# On reconnait la méthode du point médian !
>
# La méthode des trapèzes est d'ordre de convergence 2
> ordre_convergence := 2 ;
```

$$ordre\_convergence := 2$$

```
> approx1 := trapezes1(f,a,b,1) ;
```

$$approx1 := (\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b))\,(b-a)$$

```
> approx2 := trapezes1(f,a,b,2) ;
```

$$approx2 := (\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b) + \mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b))\,(\frac{1}{2}\,b - \frac{1}{2}\,a)$$

```
> factor(acceleration) ;
```

$$-\frac{1}{6}\,(-b+a)\,(\mathrm{f}(b) + \mathrm{f}(a) + 4\,\mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b))$$

```
# On reconnait la méthode de Simpson !
>
# La méthode du point médian est d'ordre de convergence 2
```

```
> ordre_convergence := 2 ;
```

$$ordre\_convergence := 2$$

```
> approx1 := median1(f,a,b,1) ;
```

$$approx1 := (b-a)\,\mathrm{f}(\frac{1}{2}\,a+\frac{1}{2}\,b)$$

```
> approx2 := median1(f,a,b,2) ;
```

$$approx2 := (\frac{1}{2}\,b-\frac{1}{2}\,a)\,(\mathrm{f}(\frac{3}{4}\,a+\frac{1}{4}\,b)+\mathrm{f}(\frac{1}{4}\,a+\frac{3}{4}\,b))$$

```
> factor(acceleration) ;
```

$$\frac{1}{3}\,(-b+a)\,(-2\,\mathrm{f}(\frac{3}{4}\,a+\frac{1}{4}\,b)-2\,\mathrm{f}(\frac{1}{4}\,a+\frac{3}{4}\,b)+\mathrm{f}(\frac{1}{2}\,a+\frac{1}{2}\,b))$$

```
# Une nouvelle méthode ???
> X := [seq( j*a + (1-j)*b , j= [1/4, 1/2, 3/4] )] ;
```

$$X := [\frac{1}{4}\,a+\frac{3}{4}\,b,\ \frac{1}{2}\,a+\frac{1}{2}\,b,\ \frac{3}{4}\,a+\frac{1}{4}\,b]$$

```
> Y := map(f, X) ;
```

$$Y := [\mathrm{f}(\frac{1}{4}\,a+\frac{3}{4}\,b),\ \mathrm{f}(\frac{1}{2}\,a+\frac{1}{2}\,b),\ \mathrm{f}(\frac{3}{4}\,a+\frac{1}{4}\,b)]$$

```
> simplify(int(interp(X, Y, x), x=a..b) - acceleration) ;
```

$$0$$

```
# On voit donc qu'accélérer la méthode du point médian
# revient à considérer l'intégration du polynôme interpolateur
# aux points X[1], X[2], X[3] de l'intervalle [a,b].
>
# La méthode de Simpson est d'ordre de convergence 4
> ordre_convergence := 4 ;
```

$$ordre\_convergence := 4$$

```
> approx1 := simpson1(f,a,b,1) ;
```

$$approx1 := \frac{2}{3}\,(b-a)\,\mathrm{f}(\frac{1}{2}\,a+\frac{1}{2}\,b)+\frac{1}{3}\,(\frac{1}{2}\,\mathrm{f}(a)+\frac{1}{2}\,\mathrm{f}(b))\,(b-a)$$

```
> approx2 := simpson1(f,a,b,2) ;
```

$$approx2 := \frac{2}{3}\,(\frac{1}{2}\,b-\frac{1}{2}\,a)\,(\mathrm{f}(\frac{3}{4}\,a+\frac{1}{4}\,b)+\mathrm{f}(\frac{1}{4}\,a+\frac{3}{4}\,b))$$
$$+\frac{1}{3}\,(\frac{1}{2}\,\mathrm{f}(a)+\frac{1}{2}\,\mathrm{f}(b)+\mathrm{f}(\frac{1}{2}\,a+\frac{1}{2}\,b))\,(\frac{1}{2}\,b-\frac{1}{2}\,a)$$

```
> factor(acceleration) ;
```

$$-\frac{1}{90}\,(-b+a)\,(32\,\mathrm{f}(\frac{1}{4}\,a+\frac{3}{4}\,b)+7\,\mathrm{f}(a)+32\,\mathrm{f}(\frac{3}{4}\,a+\frac{1}{4}\,b)+7\,\mathrm{f}(b)+12\,\mathrm{f}(\frac{1}{2}\,a+\frac{1}{2}\,b))$$

```
> simplify(NewtonCotes(f,a,b,1,4) - acceleration) ;
```

$$0$$

```
# On voit donc qu'accélérer la méthode de Simpson

# revient à considérer la méthode de Newton-Cotes

# avec un polynôme de degré 4...
```

Bonus

On vient de voir comment combiner deux approximations pour en obtenir une meilleure. On peut bien sûr réitérer le procédé sur cette nouvelle approximation en accélérant une deuxième fois, puis une troisième, etc.

$$\begin{array}{ccccccc} f(h) & & f(mh) & & f(m^2h) & & f(m^3h) \\ & acc_f(h) & & acc_f(mh) & & acc_f(m^2h) & \\ & & acc_{acc_f}(h) & & acc_{acc_f}(mh) & & \\ & & & aac_{acc_{acc_f}}(h) & & & \end{array}$$

Pour réaliser cela facilement (mais pas vraiment efficacement...) on peut utiliser le calcul suivant (les ordres de convergences des méthodes « centrées » que l'on considère sont toujours pairs, même après accélération, cf cours d'analyse numérique...) :

$$acc_f(h) = \frac{\begin{vmatrix} f(h) & f(mh) \\ 1 & m^d \end{vmatrix}}{\begin{vmatrix} 1 & 1 \\ 1 & m^d \end{vmatrix}} \qquad acc_{acc_f}(h) = \frac{\begin{vmatrix} f(h) & f(mh) & f(m^2h) \\ 1 & m^d & m^{2d} \\ 1 & m^{d+2} & m^{2(d+2)} \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 1 & m^d & m^{2d} \\ 1 & m^{d+2} & m^{2(d+2)} \end{vmatrix}}$$

```
>  restart : interface(echo=3) : read `integration8.mpl` ;
```

```
> vieil_echo := interface(echo, echo=1) :
```

"lecture de newcoat.mpl"
"lecture de simpson.mpl"
"lecture de median.mpl"
"lecture de trapez.mpl"

```
>
```

```
# Formule pour accélérer deux fois :   d -> d+2 -> d+4

> with(linalg) :

Warning, the protected names norm and trace have been redefined and
unprotected

> M1 := matrix([[approx1, approx2, approx3],

>              [1, m^d, m^(2*d)], [1, m^(d+2), m^(2*d+4)]]) ;
```

$$M1 := \begin{bmatrix} approx1 & approx2 & approx3 \\ 1 & m^{d} & m^{(2\,d)} \\ 1 & m^{(d+2)} & m^{(2\,d+4)} \end{bmatrix}$$

```
> M2 := matrix([[1, 1, 1],

>              [1, m^d, m^(2*d)], [1, m^(d+2), m^(2*d+4)]]) ;
```

$$M2 := \begin{bmatrix} 1 & 1 & 1 \\ 1 & m^{d} & m^{(2\,d)} \\ 1 & m^{(d+2)} & m^{(2\,d+4)} \end{bmatrix}$$

```
> acceleration := det(M1)/det(M2) :

>

# Testons sur un exemple connu...

> approx1 := trapezes1(f,a,b,1) ;
```

$$approx1 := (\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b))\,(b-a)$$

```
> approx2 := trapezes1(f,a,b,2) ;
```

$$approx2 := (\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b) + \mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b))\,(\frac{1}{2}\,b - \frac{1}{2}\,a)$$

```
> approx3 := trapezes1(f,a,b,4) ;
```

$$approx3 := (\frac{1}{2}\,\mathrm{f}(a) + \frac{1}{2}\,\mathrm{f}(b) + \mathrm{f}(\frac{3}{4}\,a + \frac{1}{4}\,b) + \mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b) + \mathrm{f}(\frac{1}{4}\,a + \frac{3}{4}\,b))\,(\frac{1}{4}\,b - \frac{1}{4}\,a)$$

```
> m := 1/2 ; d := 2 ;
```

$$m := \frac{1}{2}$$

$$d := 2$$

```
> factor(acceleration) ;
```

$$-\frac{1}{90}\,(7\,\mathrm{f}(a) + 7\,\mathrm{f}(b) + 12\,\mathrm{f}(\frac{1}{2}\,a + \frac{1}{2}\,b) + 32\,\mathrm{f}(\frac{3}{4}\,a + \frac{1}{4}\,b) + 32\,\mathrm{f}(\frac{1}{4}\,a + \frac{3}{4}\,b))\,(-b+a)$$

```
> simplify(NewtonCotes(f,a,b,1,4) - acceleration) ;
```

$$0$$

2.5 Exemples de super-convergence

Par principe, on pense que les méthodes classiques comme celles des rectangles, points médians, ou trapèzes, convergent lentement car leur ordre « théorique » de convergence est 1 ou 2. Il se trouve des situations où il en va tout autrement... Des exemples dans lesquels ces méthodes convergent de manière exponentielle ! Voire des exemples où ces méthodes sont exactes ! !

Fonctions périodiques

Concernant l'intégration des fonctions C^∞ périodiques, voici deux résultats (cf J.-P. Demailly, Analyse Numérique et Equations Différentielles, page 88) :

> *Soit $n \in \mathbb{N}$ et $h = 2\pi/n$. Pour $0 \leqslant i \leqslant n$, on pose $x_i = h.i \in [0, 2\pi]$. Pour toute fonction 2π-périodique et de classe C^p ($p \geqslant 2$), on a*
>
> $$E_h = \int_0^{2\pi} f(x)\ \mathrm{d}x - h\sum_{i=0}^{n} f(x_i) = O(h^p)$$

C'est pourquoi les méthodes des rectangles, des points médians, et des trapèzes sont très précises pour des fonctions lisses périodiques...

> *Soit $n \in \mathbb{N}$ et le polynôme trigonométrique $f(x) = \sum_{p=-k}^{k} c_p e^{ipx}$. Alors les méthodes de rectangles, points médians, et trapèzes sont exactes pour le pas $2\pi/(k+1)$.*

Plus précisément, si le pas est $h = 2\pi/n$ alors

$$\int_{-\pi}^{\pi} e^{ikx}\ \mathrm{d}x - \text{Méthode}_n = \begin{cases} 0 & \text{si } k \notin n\mathbb{Z}^* \\ -2\pi\epsilon_k & \text{si } k = n\alpha \neq 0 \end{cases}$$

où $\epsilon_k = (-1)^k$ pour les méthodes des rectangles et des trapèzes,
et $\epsilon_k = (-1)^{k+\alpha}$ pour la méthode des points médians.

Une intégrale généralisée

On se propose de calculer une approximation de l'intégrale généralisée (dont la valeur est $\sqrt{\pi}/2$).

$$I = \int_0^{+\infty} e^{-x^2}\mathrm{d}x$$

Bien sûr aucune méthode élémentaire ne peut faire l'approximation sur l'intervalle infini $[0, +\infty[$ tout entier. Il faut commencer par choisir un intervalle borné, en l'occurrence $[0, \sqrt{\pi n}]$ où n va être le nombre de subdivisions. En particulier, la longueur des sous-intervalles sera $\sqrt{\pi/n}$.

Une étude fine montre la convergence géométrique de la méthode de Simpson :

$$0 < \int_0^\infty e^{-x^2}\mathrm{d}x - \mathrm{Simpson}_n(x \mapsto \exp(-x^2), 0, \sqrt{\pi n}) < e^{-\pi n}$$

En fait, pour démontrer l'inégalité ci-dessus, on utilise les inégalités ci-dessous valables pour $n \geqslant 6$:

$$\begin{array}{l} 1.4\ e^{-\pi n} \\ \quad < \mathrm{Trapèzes}_n(x \mapsto \exp(-x^2), 0, \sqrt{\pi n}) - \displaystyle\int_0^\infty e^{-x^2}\mathrm{d}x \\ \quad\quad < \sqrt{\pi} e^{-\pi n} \\ \quad\quad\quad < \displaystyle\int_0^\infty e^{-x^2}\mathrm{d}x - \mathrm{Médians}_n(x \mapsto \exp(-x^2), 0, \sqrt{\pi n}) \\ \quad\quad\quad\quad < 1.8\ e^{-\pi n} \end{array}$$

On a également les équivalents suivants quand n tend vers l'infini :

$$\begin{array}{c} \mathrm{Trapèzes}_n(x \mapsto \exp(-x^2), 0, \sqrt{\pi n}) - \displaystyle\int_0^\infty e^{-x^2}\mathrm{d}x \\ \sim \sqrt{\pi} e^{-\pi n} \sim \\ \displaystyle\int_0^\infty e^{-x^2}\mathrm{d}x - \mathrm{Médians}_n(x \mapsto \exp(-x^2), 0, \sqrt{\pi n}) \end{array}$$

Bref, ces trois méthodes convergent extrêmement vite sur cet exemple...

Chapitre 3

VECTEURS (POLYNÔMES) ORTHOGONAUX

3.1 Orthonormalisation de Gram-Schmidt

Le procédé d'orthogonalisation de Gram-Schmidt se prête bien à la programmation : étant donné un produit scalaire sur un espace vectoriel et une base $\{f_1, ...f_i\}$ d'un sous-espace, on construit une famille $\{e_1, ..., e_i\}$ orthogonale, voire $\{o_1, ..., o_i\}$ orthonormale, engendrant le même sous-espace.

Ce procédé utilise le fait que la projection orthogonale d'un vecteur x sur un sous-espace, de base orthonormale $\{o_1, ..., o_{i-1}\}$, a pour expression

$$\sum_{k=1}^{i-1} \langle x|o_k\rangle \, .o_k$$

Par la récurrence suivante, on obtient une famille orthonormale en divisant chaque vecteur e_i par sa norme.

$$e_1 = f_1 \quad , \quad o_1 = e_1/||e_1||$$

$$e_i = f_i - \sum_{k=1}^{i-1} \langle f_i|o_k\rangle \, .o_k \quad , \quad o_i = e_i/||e_i|| \qquad \text{pour } i > 1$$

$$\bigoplus_{k=1}^{i} R.f_k = \bigoplus_{k=1}^{i} R.e_k = \bigoplus_{k=1}^{i} R.o_k$$

```
>  restart : read ‘my_config.mpl‘ : interface(echo=3) :
>  read ‘orthogon1.mpl‘ ;

> read ‘prod_scal.mpl‘ ;

# Notre exercice se situe dans l’espace des fonctions réelles
```

```
# continues muni de différents produits scalaires.
# Voici quelques exemples :
>
# Le produit scalaire de L^2([-1,1]).
# L'orthonormalisation de la base
# canonique de R[X] produit les polynômes de Legendre.
>
> prod_scal_legendre := proc(f,g,x)
>       # f,g : deux expressions (fonctions continues) de x
>       RETURN(int(f*g, x=-1..1)) ;
>       # calcul symbolique "int(...)"
> end :
>
# Les polynômes de Tchebyshev de première et seconde espèce
# s'obtiennent par le produit scalaire suivant :
>
> prod_scal_tchebyshev := proc(f,g,x)
>       # f,g : deux expressions (fonctions continues) de x
>       RETURN(int(f*g/sqrt(1-x^2), x=-1..1)) ;
>       # calcul symbolique "int(...)"
> end :
>
> prod_scal_tchebyshev_2 := proc(f,g,x)
>       # f,g : deux expressions (fonctions continues) de x
>       RETURN(int(f*g*sqrt(1-x^2), x=-1..1)) ;
>       # calcul symbolique "int(...)"
```

```
> end :
>
# ceux de Laguerre par :
>
> prod_scal_laguerre := proc(f,g,x)
>       # f,g : deux expressions (fonctions continues) de x
>       RETURN(evalf( Int(f*g*exp(-x), x=0..infinity) )) ;
>       # calcul numérique "evalf(Int(...))", non symbolique
> end :
>
# et ceux d'Hermite par :
>
> prod_scal_hermite := proc(f,g,x)
>       # f,g : deux expressions (fonctions continues) de x
>       RETURN(evalf( Int(f*g*exp(-x^2), x=-infinity..infinity) )) ;
>       # calcul numérique "evalf(Int(...))", non symbolique
> end :
>
# Écrivons quelques procédures correspondant à des notions
# classiques quand on travaille dans un espace euclidien.
>
> norme := proc(f, scal, x)
>       # f    : l'élément (expression en x) dont on veut la norme
>       # scal : le produit scalaire
>       # x    : la variable de f
>       RETURN( sqrt(scal(f, f, x)) ) ;
```

```
> end :
>
> normaliser := proc(f, scal, x)
>       # f    : l'élément (expression en x) à normaliser
>       # scal : le produit scalaire
>       # x    : la variable de f
>       RETURN( f / norme(f, scal, x) ) ;
> end :
>
> projection_orthogonale := proc(f, scal, B, x)
>       # f    : l'élément (expression en x) à projeter
>       # scal : le produit scalaire
>       # B    : une famille (liste) orthonormale pour scal
>       # x    : la variable de f et des éléments de B
>       local b ;
>       RETURN( add( scal(f, b, x) * b,  b=B )) ;
> end :
>
> distance_orthogonale := proc(f, scal, B, x)
>       # f    : l'expression en x dont on veut la distance a vect(B)
>       # scal : le produit scalaire
>       # B    : une famille (liste d'expressions en x) orthonormale
>       # x    : la variable de f et des éléments de B
>       local p :
>       p := projection_orthogonale(f, scal, B, x) ;
>       RETURN(norme(f-p, scal, x)) ;
```

```
> end :
>
# Écrivons la procédure d'orthonormalisation, avec en paramètres
# le produit scalaire et une famille libre de vecteurs donnée sous
# forme d'une liste.
# Le résultat est une suite formant une base orthonormale.
>
> GramSchmidt := proc(scal, F, x)
>       # scal : le produit scalaire
>       # F : une liste d'expressions en x à orthonormaliser avec scal
>       # x : la variable des éléments de F
>       local B,p,i ;
>         # B : famille orthonormale (à la fin de l'algo)
>         # p : la projection orthogonale sur B d'un vecteur de F
>       B := F ;
>
>       B[1] := normaliser(B[1], scal, x) ;
>       for i from 2 to nops(B) do
>           # B[1,...,i-1] est une famille orthonormalisée
>         p := projection_orthogonale(B[i], scal, B[1..(i-1)], x) ;
>         B[i] := B[i] - p ;
>         B[i] := normaliser(B[i], scal, x) ;
>           # B[1,...,i] est une famille orthonormalisée
>       od ;
>
>       # B est une famille orthonormalisée
```

```
>       RETURN(B) ;
> end :
>
########## fin de prod_scal.mpl ##########
>
# Les polynômes orthogonaux pour ces produits scalaires ont
# déjà ete implémentes dans MAPLE (entre autres).
>
> with(orthopoly) ;
```

$$[G,\ H,\ L,\ P,\ T,\ U]$$

```
>
> poly_legendre := proc(d,x)
>       # d est le degré du polynôme orthogonal (de variable x)
>       local p ;
>       p := P(d, x) ;
>       RETURN( normaliser(p, prod_scal_legendre, x) ) ;
> end :
>
# Vérifions si tout fonctionne bien :
> F := [seq(x^d, d=0..5)] ;
```

$$F := [1,\ x,\ x^2,\ x^3,\ x^4,\ x^5]$$

```
> GS := GramSchmidt(prod_scal_legendre, F, x) ;
```

$$GS := [\frac{1}{2}\sqrt{2},\ \frac{1}{2}x\sqrt{6},\ \frac{3}{4}(x^2-\frac{1}{3})\sqrt{10},\ \frac{5}{4}(x^3-\frac{3}{5}x)\sqrt{14},\ \frac{105}{16}(x^4+\frac{3}{35}-\frac{6}{7}x^2)\sqrt{2},$$
$$\frac{63}{16}(x^5+\frac{5}{21}x-\frac{10}{9}x^3)\sqrt{22}]$$

```
> L := [seq(poly_legendre(d,x), d=0..5)] :
> seq(simplify(GS[i] - L[i]), i=1..nops(F)) ;
```

$$0,\ 0,\ 0,\ 0,\ 0,\ 0$$

```
>
> poly_tchebyshev := proc(d,x)
>       # d est le degré du polynôme orthogonal (de variable x)
>       local p ;
>       p := T(d, x) ;
>       RETURN( normaliser(p, prod_scal_tchebyshev, x) ) ;
> end :
>
# Vérifions encore que tout fonctionne bien :
> F := [seq(x^d, d=0..5)] ;
```

$$F := [1,\ x,\ x^2,\ x^3,\ x^4,\ x^5]$$

```
> GS := GramSchmidt(prod_scal_tchebyshev, F, x) ;
```

$$GS := \left[\frac{1}{\sqrt{\pi}},\ \frac{x\sqrt{2}}{\sqrt{\pi}},\ 2\,\frac{(x^2-\frac{1}{2})\sqrt{2}}{\sqrt{\pi}},\ 4\,\frac{(x^3-\frac{3}{4}\,x)\sqrt{2}}{\sqrt{\pi}},\ 8\,\frac{(x^4+\frac{1}{8}-x^2)\sqrt{2}}{\sqrt{\pi}},\ 16\,\frac{(x^5+\frac{5}{16}\,x-\frac{5}{4}\,x^3)\sqrt{2}}{\sqrt{\pi}}\right]$$

```
> T := [seq(poly_tchebyshev(d,x), d=0..5)] :
> seq(simplify(GS[i] - T[i]), i=1..nops(F)) ;
```

$$0,\ 0,\ 0,\ 0,\ 0,\ 0$$

```
>
> poly_tchebyshev_2 := proc(d,x)
>       # d est le degré du polynôme orthogonal (de variable x)
>       local p ;
>       p := U(d, x) ;
>       RETURN( normaliser(p, prod_scal_tchebyshev_2, x) ) ;
> end :
```

```
>
> poly_laguerre := proc(d,x)
>       # d est le degré du polynôme orthogonal (de variable x)
>       local p ;
>       p := L(d, x) ;
>       RETURN( normaliser(p, prod_scal_laguerre, x) ) ;
> end :
>
> poly_hermite := proc(d,x)
>       # d est le degré du polynôme orthogonal (de variable x)
>       local p ;
>       p := H(d, x) ;
>       RETURN( normaliser(p, prod_scal_hermite, x) ) ;
> end :
```

3.2 Approximation polynomiale au sens L^2

Sur un intervalle $I \subset R$ ouvert non vide, pour une fonction f continue, et pour un degré fixé n, le polynôme obtenu par projection orthogonale de f sur $\mathbb{R}_{\leqslant n}[X]$ via le produit scalaire $\langle f, g\rangle = \int_I fg$ est un polynôme p tel que la valeur de $\int_I (f-p)^2$ soit minimale.

```
> restart : read `my_config.mpl` : interface(echo=3) :
> read `orthogon2.mpl` ;
```

```
# Nous allons comparer deux polynômes approximant une même fonction

# sur l'intervalle [-1,1]. Le premier polynôme vient

# d'interpolations (répartition à la Tchebyshev)

# et le second est le meilleur au sens L^2.

>

> vieil_echo := interface(echo, echo=1)[1] :
```

lecture de orthogon1.mpl

```
>

# voici quelques exemples...

#f := abs ;

#f := x -> cos(10*x) ;

> f := x -> 1/(1+(1+5*x)^2) ;
```

$$f := x \to \frac{1}{1+(1+5\,x)^2}$$

```
nn_max := 16 ;
```

$$n\_max := 16$$

```
>

> suite_L2 := NULL :

> F := [seq(x^d, d=0..n_max)] ;
```

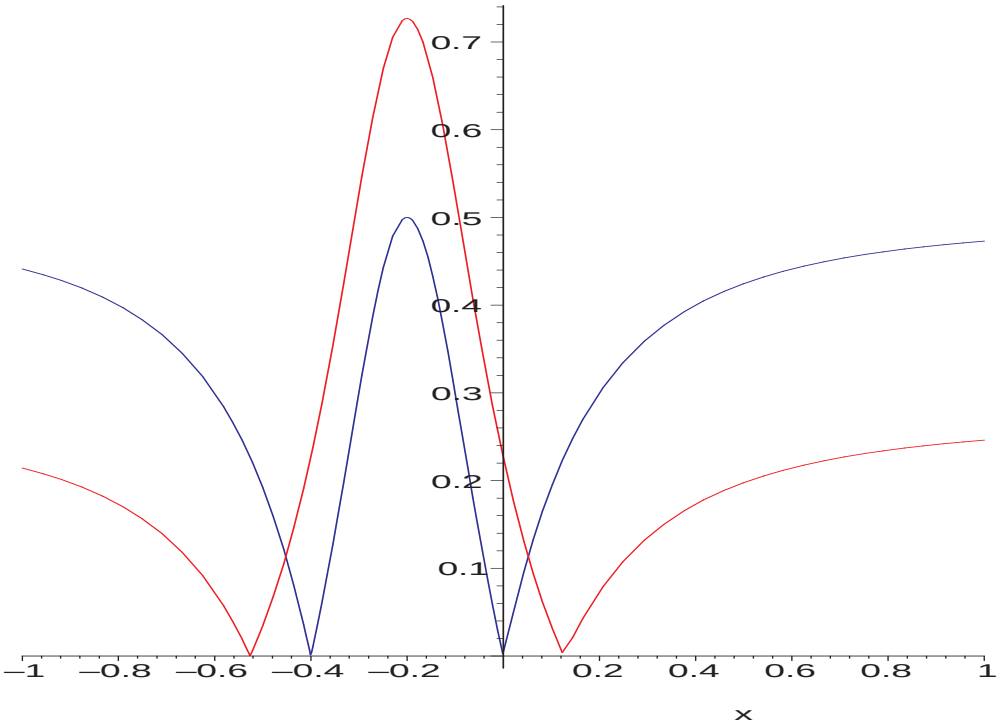
$$F := [1,\, x,\, x^2,\, x^3,\, x^4,\, x^5,\, x^6,\, x^7,\, x^8,\, x^9,\, x^{10},\, x^{11},\, x^{12},\, x^{13},\, x^{14},\, x^{15},\, x^{16}]$$

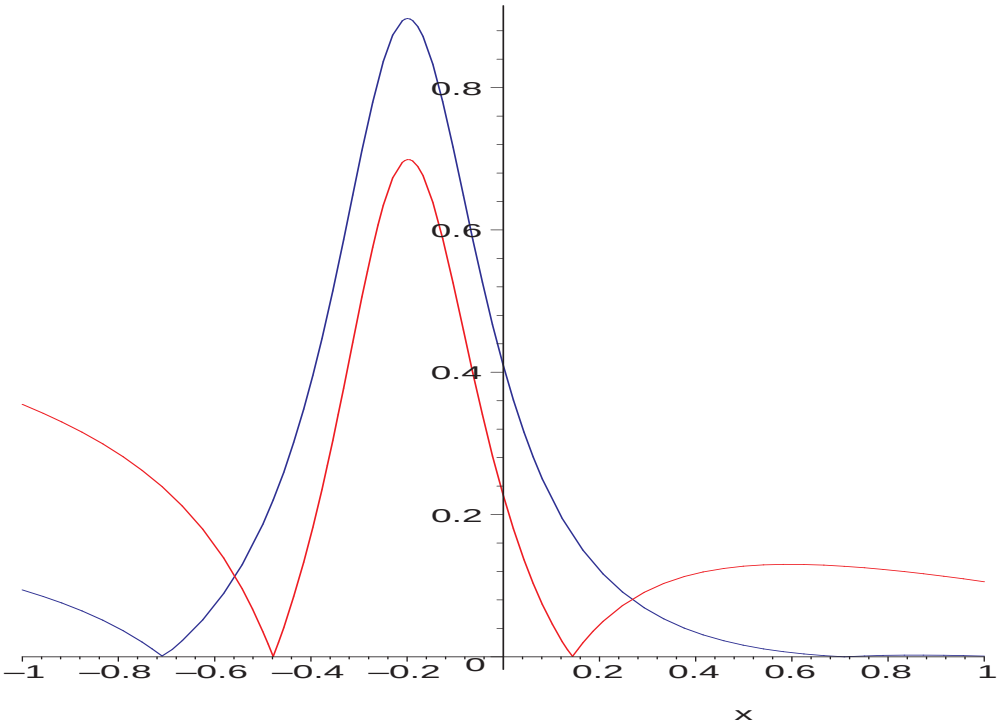
```
> GS := GramSchmidt(prod_scal_legendre, F, x) :

> GS[1..5] ;
```

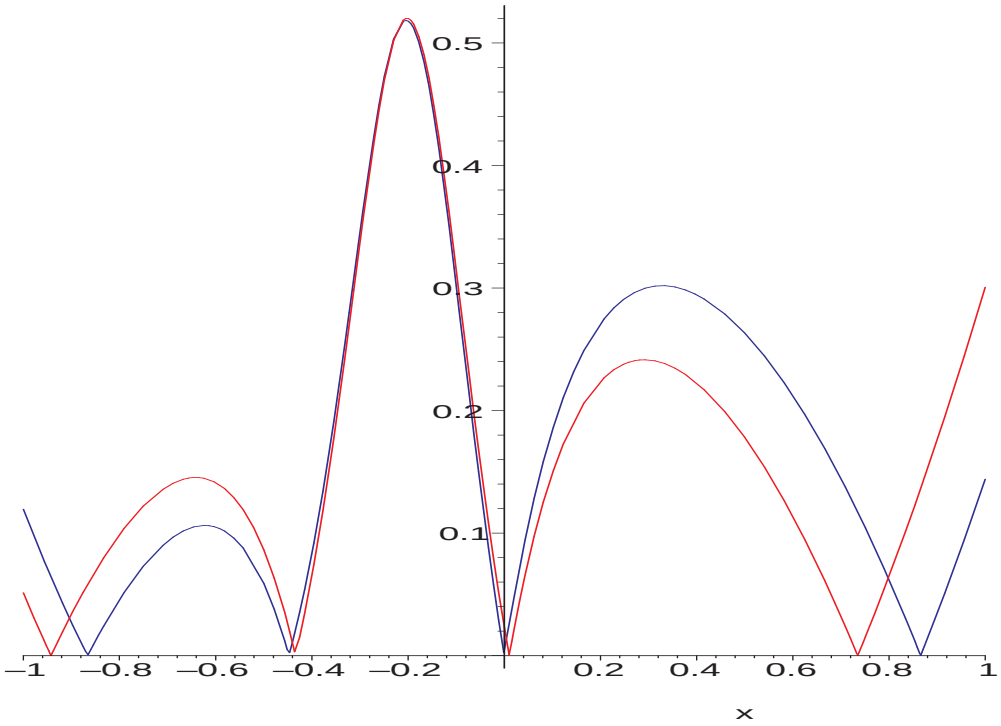
$$\left[\frac{\sqrt{2}}{2},\ \frac{x\sqrt{6}}{2},\ \frac{3\,(x^2-\frac{1}{3})\sqrt{10}}{4},\ \frac{5\,(x^3-\frac{3}{5}\,x)\sqrt{14}}{4},\ \frac{105\,(x^4+\frac{3}{35}-\frac{6}{7}\,x^2)\sqrt{2}}{16}\right]$$

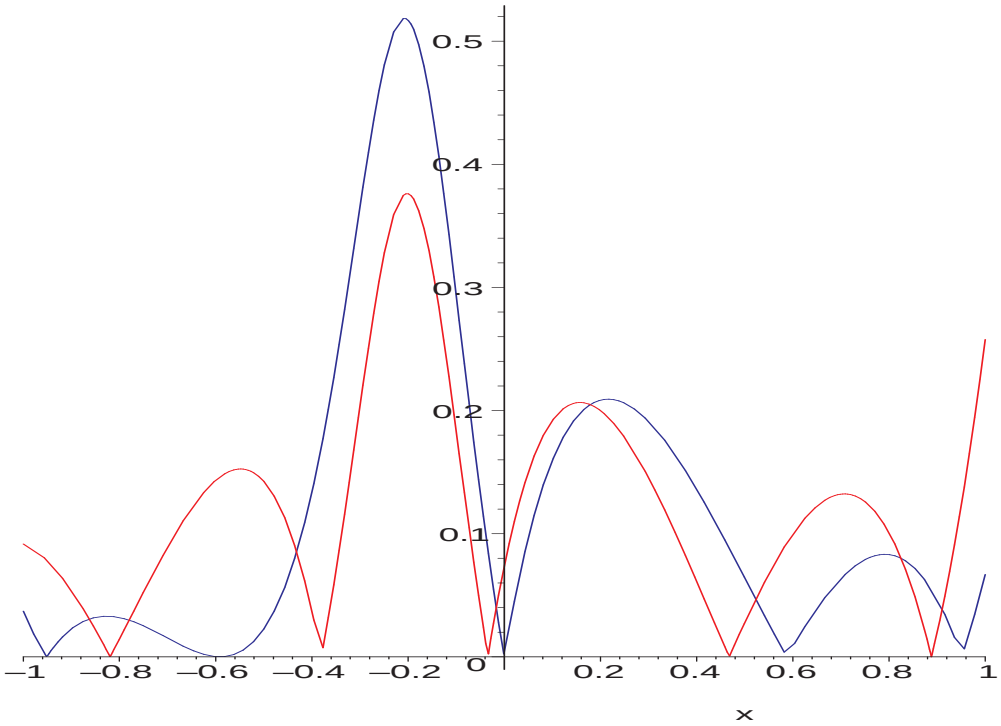
```
> for n from 0 to n_max do
```

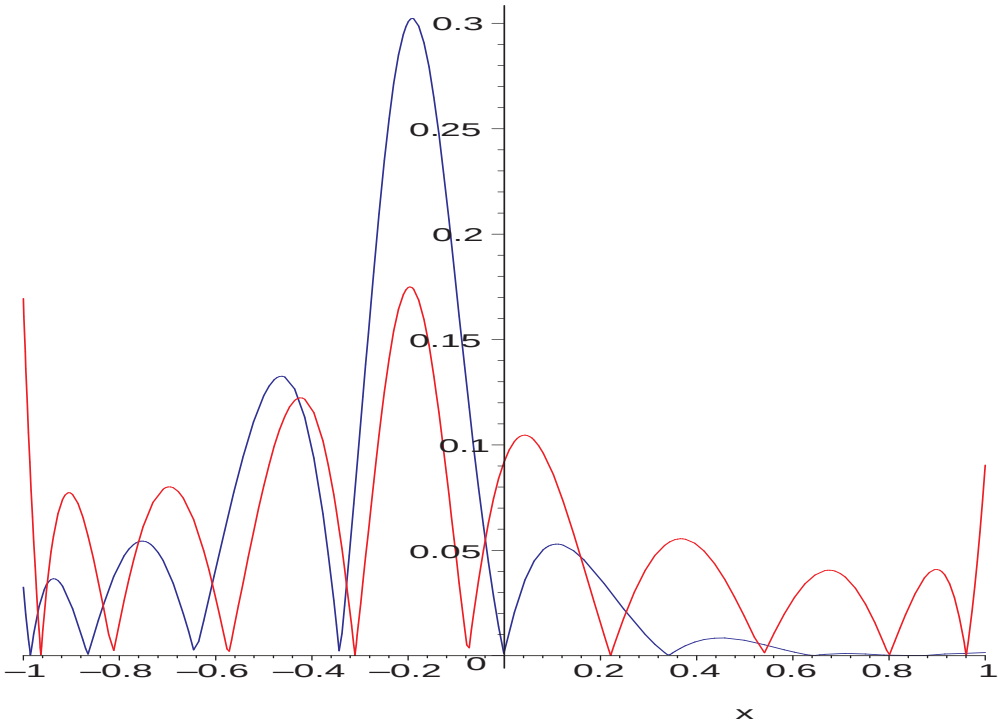
```
>   suite_L2 := suite_L2,
>       evalf(projection_orthogonale(f(x), prod_scal_legendre,
>                       GS[1..n+1], x)) :
> od :
>
> suite_Tchebyshev := NULL :
> for n from 0 to n_max do
>    Tcheby := i -> cos((2*i-1) * Pi / (2*n+2)) ;
>    racines := [seq(evalf(Tcheby(i)), i= 1..n+1)] ;
>    Y2 := map(f, racines) ;
>    suite_Tchebyshev := suite_Tchebyshev, interp(racines, Y2, x) ;
> od :
>
> for i in [0,1,2,4,8,16] do
>   plot([abs(f(x)-suite_L2[i+1]), abs(f(x)-suite_Tchebyshev[i+1])],
>         x = -1..1, color=[red,blue]);
>   pour_le_fichier(%, orthogon2) ;
> od ;
```

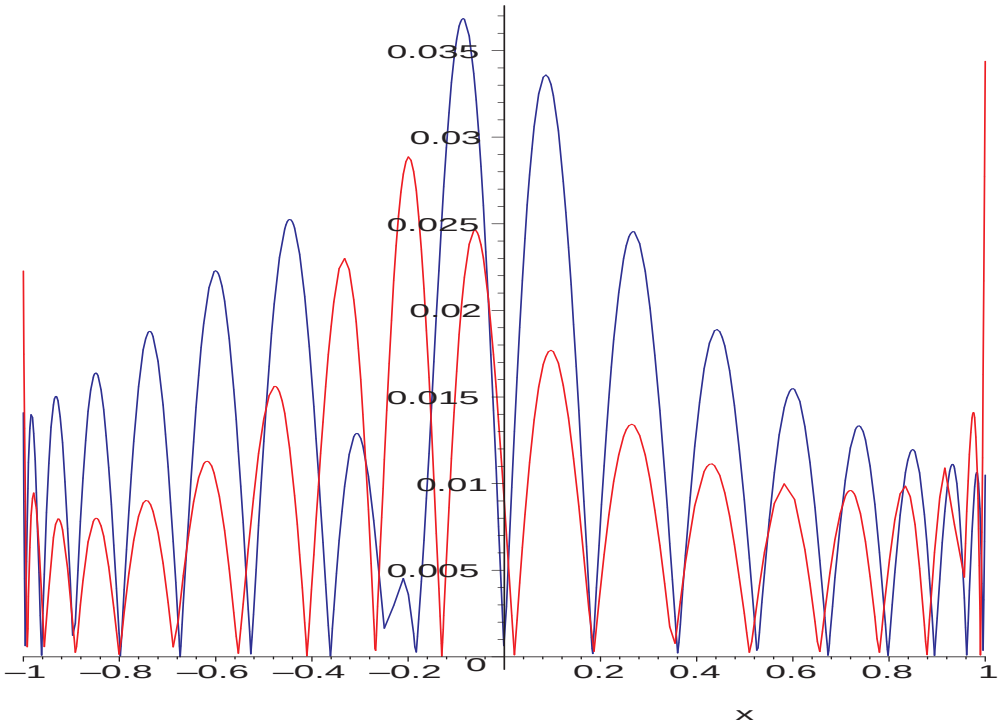

0.7
0.6
0.5
0.4
0.3
0.2
0.1
−1
−0.8
−0.6
−0.4
−0.2
0.2
0.4
0.6
0.8
1
x


0.8
0.6
0.4
0.2
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
x


0.5
0.4
0.3
0.2
0.1
−1
−0.8
−0.6
−0.4
−0.2
0.2
0.4
0.6
0.8
1
x


0.5
0.4
0.3
0.2
0.1
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
x


0.3
0.25
0.2
0.15
0.1
0.05
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
x


0.035
0.03
0.025
0.02
0.015
0.01
0.005
−1
−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8
1
x

3.3 Approximation au sens des moindres carrés

On singe la section 3.2 en l'adaptant au problème discret suivant : on considère des points du plan $(x_0, y_0), \ldots, (x_n, y_n)$ d'abscisses distincts. On veut calculer un polynôme P (en genéral constant, ou linéaire) tel que la somme $\sum_{i=0}^{n} \big(y_i - P(x_i)\big)^2$ soit la plus petite possible.

On présente ici une méthode de calcul qui n'est pas celle utilisée en général pour résoudre ce problème, mais elle a le mérite de traiter des problèmes concrets. Les méthodes qu'utilisent les statisticiens (pour une régression linéaire par exemple) sont beaucoup plus rapides et plus stables...

On définit un produit scalaire sur l'ensemble des polynômes $\mathbb{R}_{\leqslant n}[X]$ simplement par $\langle f, g\rangle = \sum_{i=0}^{n} f(x_i)g(x_i)$. On considère la $\mathbb{R}$-base $\{1, X, \ldots, X^n\}$ de $\mathbb{R}_{\leqslant n}[X]$, que l'on orthonormalise par le procédé de Gram-Schmidt. Pour déterminer le meilleur (au sens des moindres carrés) polynôme P de degré inférieur ou égal à d, il suffit alors de calculer la projection orthogonale du polynôme interpolateur passant les $n+1$ points (x_i, y_i) sur l'espace vectoriel engendré par $\{1, X, \ldots, X^d\}$.

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘orthogon5.mpl‘ ;
```

```
> vieil_echo := interface(echo, echo=1) :
```

lecture de orthogon1.mpl

```
>
```

```
> scal := proc(f,g,x) global X ;
```

```
>      RETURN(simplify( add(subs(x=i, f*g), i=X) ))  ;
```

```
> end :
```

```
>
```

```
> n := 3 : X := [seq(x||i, i=0..n)] ; Y := [seq(y||i, i=0..n)] ;
```

$$X := [x0,\ x1,\ x2,\ x3]$$

$$Y := [y0,\ y1,\ y2,\ y3]$$

```
> PolyInterp := interp(X,Y,x) :
```

```
>
```

```
# En projetant orthogonalement PolyInterp sur l'espace des constante,
```

```
# vérifions que la moyenne des y_i est le meilleur (au sens des
```

```
# moindres carrés) polynôme constant :
```

```
> scal(PolyInterp, 1, x)/scal(1, 1, x) ;
```

$$\frac{1}{4}\,y0 + \frac{1}{4}\,y3 + \frac{1}{4}\,y2 + \frac{1}{4}\,y1$$

```
>

# Autre exemple (numérique cette fois) avec un dessin...

>

> X := [-3, -2, 1, 2, 3, 4, 5] ; Y := [2, 0.5, -1, -1, 0, 2, 4] ;
```

$$X := [-3,\ -2,\ 1,\ 2,\ 3,\ 4,\ 5]$$

$$Y := [2,\ .5,\ -1,\ -1,\ 0,\ 2,\ 4]$$

```
> PolyInterp := interp(X,Y,x) :

> F := [seq(x^k, k=0..nops(X)-1)] ;
```

$$F := [1,\ x,\ x^2,\ x^3,\ x^4,\ x^5,\ x^6]$$

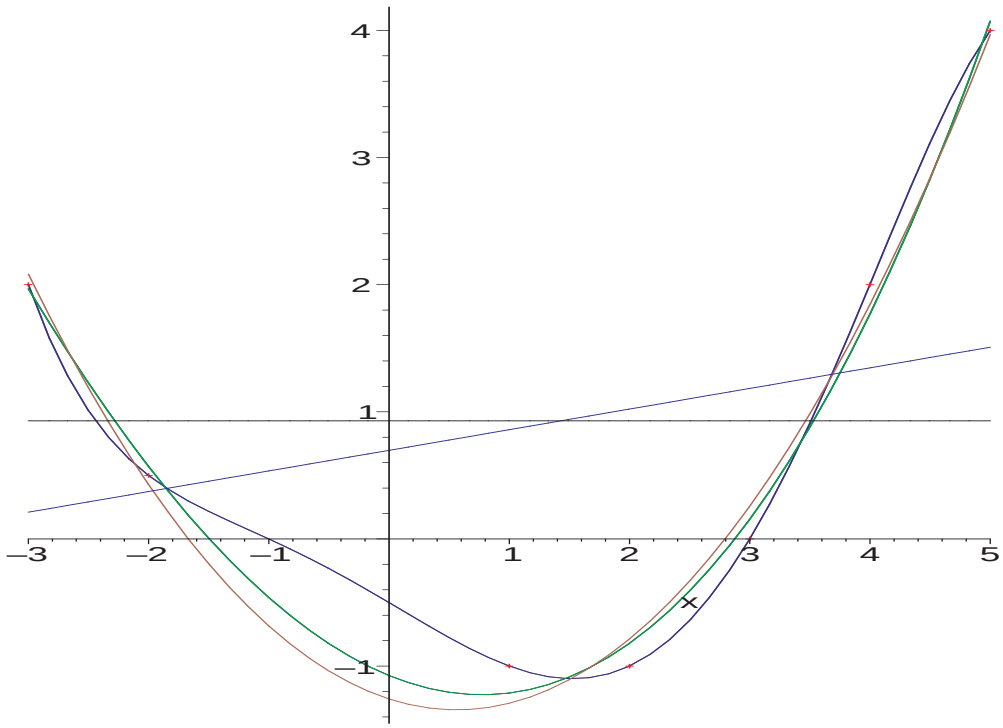
```
> B := GramSchmidt(scal, F, x) ;
```

$$B := [\frac{1}{7}\sqrt{7},\ \frac{1}{188}\,(x - \frac{10}{7})\,\sqrt{658},\ \frac{1}{3423}\,(x^2 - \frac{681}{94} - \frac{325}{188}\,x)\,\sqrt{45966},$$
$$\frac{7}{1100052}\,(x^3 + \frac{161460}{7987} - \frac{70159}{7987}\,x - \frac{28620}{7987}\,x^2)\,\sqrt{29884746},$$
$$\frac{1}{13363944}\,(x^4 + \frac{349272}{30557} + \frac{949060}{30557}\,x - \frac{302177}{30557}\,x^2 - \frac{127720}{30557}\,x^3)\,\sqrt{34030169734},$$
$$\frac{1}{1732080}$$
$$(x^5 - \frac{6977520}{50621} + \frac{6305612}{151863}\,x + \frac{3138320}{50621}\,x^2 - \frac{1941095}{151863}\,x^3 - \frac{253460}{50621}\,x^4)\,\sqrt{313141506}$$
$$, \frac{1}{20160}$$
$$(x^6 + \frac{327600}{1031} - \frac{3363060}{11341}\,x - \frac{1051616}{11341}\,x^2 + \frac{1145745}{11341}\,x^3 - \frac{26165}{11341}\,x^4 - \frac{7695}{1031}\,x^5)$$
$$\sqrt{68046}]$$

```
>

> SuitePolynomes := seq(

>        simplify(projection_orthogonale(PolyInterp, scal, B[1..i],
x)),

>     i = 1..nops(B)) :

> for i from 1 to 3 do SuitePolynomes[i] od ;
```

$$.9285714324$$
$$.6968085122 + .1622340441\,x$$
$$-1.258357333 - .3043069984\,x + .2698760491\,x^2$$

```
>

> with(plots) :

Warning, the name changecoords has been redefined

> LesPoints := pointplot([ seq([X[i],Y[i]], i=1..nops(X)) ],
color=red) :

> LesPolynomes := plot([ SuitePolynomes ], x=min(op(X))..max(op(X)),

>                       color=[black, blue, brown, green]) :

> display([LesPoints,LesPolynomes]);
```



3.4 Petit coup d'œil sur les séries de Fourier

Les séries de Fourier sont un exemple classique d'approximations de fonctions périodiques. Plaçons nous sur $[-\pi, \pi]$. Les fonctions $x \mapsto \cos(kx)$ et $x \mapsto \sin(kx)$ pour $k \in \mathbb{N}^*$ forment une famille orthonormale pour le produit scalaire $\langle f, g\rangle = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)g(x)\mathrm{d}x$.

```
> restart : read `my_config.mpl` : interface(echo=3) :
> read `orthogon4.mpl` ;
```

```
> vieil_echo := interface(echo, echo=1)[1] :
```

lecture de orthogon1.mpl

```
>
```

```
> scal := proc(f,g,x)
```

```
>    RETURN(int(f*g/Pi, x=-Pi..Pi)) ;
```

```
> end :
```

```
>
```

```
# La famille trigonométrique suivante est une base orthonormale
```

```
# pour le produit scalaire ci-dessus.
```

```
> n := 3 ;
```

$$n := 3$$

```
> F := t -> [1/sqrt(2), seq(cos(k*x), k=1..n), seq(sin(k*x),
k=1..t*n)] ;
```

$$F := t \to [\frac{1}{\sqrt{2}}, \mathrm{seq}(\cos(k\,x),\ k = 1..n), \mathrm{seq}(\sin(k\,x),\ k = 1..t\,n)]$$

```
>
```

```
# Vérifions que F1 est orthonormale :
```

```
> matrix([seq([seq( scal(i,j,x) , i=F(1))], j=F(1))]) ;
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>
```

```
# voici un exemple de projections orthogonales fournissant
```

```
# des approximations trigonométriques de la fonction f...
```

```
> f := x -> x/Pi - floor(x/Pi) ;
```

$$f := x \to \frac{x}{\pi} - \mathrm{floor}(\frac{x}{\pi})$$

```
#f := x -> x/abs(x) ;
```

```
>
```

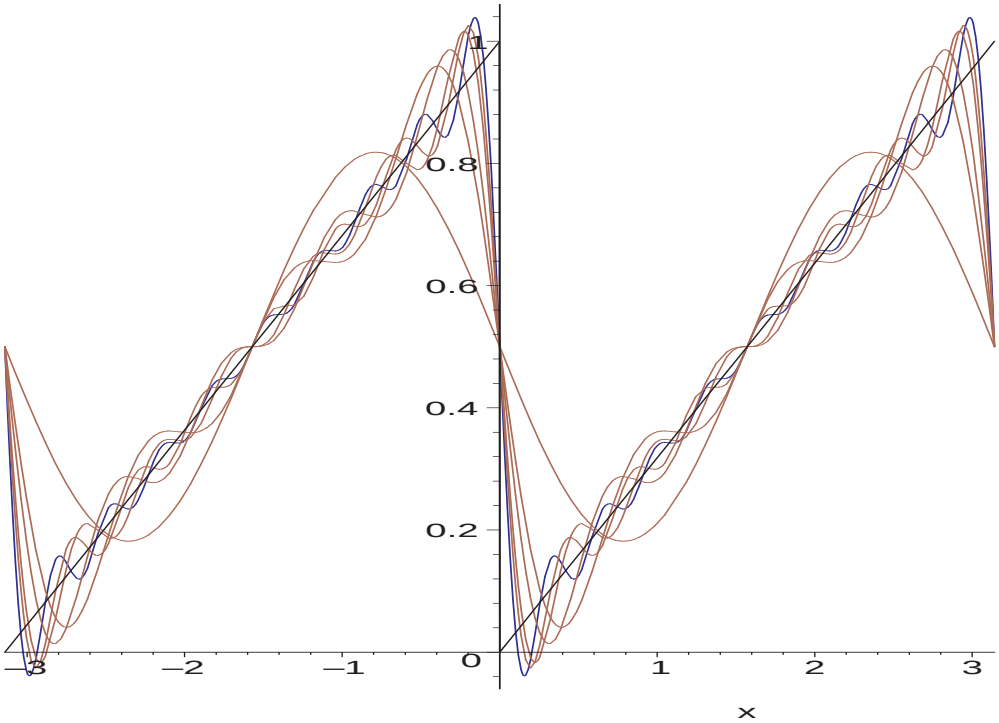
```
> P := t -> projection_orthogonale(f(x), evalf@scal, F(t), x) ;
```

$$P := t \to \mathrm{projection\_orthogonale}(\mathrm{f}(x),\ evalf@scal,\ \mathrm{F}(t),\ x)$$

```
>
```

```
> t := 5 :
```

```
> plot([f(x), seq(P(i), i=1..t), P(t+1)], x = -Pi..Pi,
>                                                color=[black,brown$t,blue]);
```




```
> pour_le_fichier(%, orthogon4):
```

```
>
```

```
# On remarque le phénomène de Gibbs aux alentours des discontinuités :
```

```
# l'écart entre f et P(i) est ne diminue pratiquement pas au voisinage

# de 0- et 0+.
```

3.5 Intégration numérique : méthode de Gauss

Ici, nous étudions la méthode de Gauss : on cherche une valeur approchée de l'intégrale $\int_{-1}^{1} f$ sous la forme d'une combinaison linéaire

$$\int_{-1}^{1} f \simeq a_1 f(x_1) + \cdots + a_n f(x_n)$$

où les coefficients $(a_i)_i$ et les abscisses $(x_i)_i \in [-1,1]$ sont indépendants de la fonction f (méthode de quadrature). Cette approche est ultra-classique (pour tout problème linéaire), mais ici, les a_i et les x_i sont choisis au mieux pour obtenir au final une méthode particulièrement efficace, d'ordre $2n-1$ (convergence d'ordre $2n$) ! Puis, par le changement de variable $[-1,1] \ni x \mapsto \frac{a+b}{2} + x\frac{b-a}{2} \in [a,b]$, on obtient une approximation d'une intégrale $\int_a^b f$.

On considère le produit scalaire de deux fonctions (continues par exemple) $\langle f, g\rangle = \int_{-1}^{1} fg$ et les polynômes orthogonaux L_i qui s'en déduisent par le procédé d'orthonormalisation de Gram-Schmidt (en partant de la base canonique $\{1, X, X^2, \ldots\}$).

Comment trouver les abscisses x_i et les poids a_i ? On considère $P = \prod_{i=0}^{n}(x - x_i)$. Comme la méthode est d'ordre $2n-1$, pour tout polynôme $Q \in \mathbb{R}_{<n}[X]$, on a

$$\langle P, Q\rangle = \int_{-1}^{1} P.Q = \sum_i a_i (P.Q)(x_i) = 0$$

Donc P est orthogonal à $\mathbb{R}_{<n}$. Les abscisses x_i sont donc les racines du n-ième polynôme orthogonal. Par ailleurs, en considérant le polynôme $\prod_{r\in J} X - r$ où J est l'ensemble des $x_i \in]-1,1[$ de multiplicité impaire, on montre que $J = P$ (car $\langle P, J\rangle > 0$), si bien que tous les x_i appartiennent à $]-1,1[$.

Pour obtenir le poids a_i (i fixé entre 1 et n), on considère le i-ème polynôme élémentaire de Lagrange $\delta_i \in \mathbb{R}_{<n}[X]$ valant 1 en x_i et s'annulant en tous les autres x_j. On a alors

$$\langle \delta_i, 1\rangle = \int_{-1}^{1} \delta_i = \sum_j a_j \delta_i(x_j) = a_i$$

Maintenant, on constate que cette formule de quadrature est d'ordre $2n-1$: pour tout $f \in \mathbb{R}_{<2n}[x]$, on peut écrire $f = Q.\prod_{i=1}^{n}(x - x_i) + R$ où $\deg(Q) < n$ et $\deg(R) < n$. Or $R = \sum_i R(x_i)\delta_i$, donc on a

$$\int_{-1}^{1} f = \int_{-1}^{1} Q(x).\prod_{i=1}^{n}(x - x_i)\ \mathrm{d}x + \sum_{i=1}^{n} R(x_i)\int_{-1}^{1}\delta_i = 0 + \sum_{i=1}^{n} R(x_i)a_i = \sum_{i=1}^{n} f(x_i)a_i$$

```
> restart : read ‘my_config.mpl‘ : interface(echo=3) :
> read ‘orthogon3.mpl‘ ;

# La methode de Gauss :
```

```
# on cherche à exprimer l'intégrale d'une fonction f sur [a,b]
# par une combinaison linéaire a_1 f(x_1) + ... + a_n f(x_n)
# où les coefficients (a_i)_i et (x_i)_i sont indépendants de f.
>
> vieil_echo := interface(echo, echo=1)[1] :
```

lecture de orthogon1.mpl

```
>
# Les (x_i)_i sont les racines d'un polynôme orthogonal.
# La procédure suivante détermine le coefficient a_i (ou poids)
# associé à une racine r d'un polynôme orthogonal p pour
# le produit scalaire scal.
>
> poids := proc(p, r, scal, x)
>       # p : polynôme orthogonal pour scal
>       # r : une racine du polynôme p
>       # scal : le produit scalaire
>       # x : la variable de p
>     local delta ;
>     delta := quo(p, x - r, x) ;
>     delta := delta / subs(x=r, delta) ;
>     RETURN(scal(delta, 1, x)) ;
> end :
>
# Que cela donne-t-il sur un exemple simple ?
> deg := 5 ;
```

$$deg := 5$$

```
> pol := evalf(poly_legendre(deg, x)) ;
```

$$pol := 18.46851206\,x^5 - 20.52056895\,x^3 + 4.397264775\,x$$

```
> points := [solve(pol=0)] ;
```

$$points := [0.,\ -0.9061798457,\ 0.9061798457,\ -0.5384693101,\ 0.5384693101]$$

```
> coefficients := [seq(poids(pol, i, prod_scal_legendre, x),
i=points)] ;
```

$$coefficients := [0.5688888889,\ 0.2369268857,\ 0.2369268857,\ 0.4786286703,\ 0.4786286703]$$

```
>

# Ces points et coefficients permettent d'obtenir une excellente

# approximation (avec 5 appels de f ici, seulement !) de

# prod_scal(f, 1) par un calcul final très simple.

> f := exp ;
```

$$f := \exp$$

```
> Digits := 12 :

> approx := sum(coefficients[i] * f(points[i]), i=1..deg) ;
```

$$approx := 2.35040238775$$

```
>

> Graal := evalf(prod_scal_legendre(f(x), 1, x)) ;
```

$$Graal := 2.35040238729$$

```
>

# Écrivons une procédure intégrant f sur un intervalle [a,b]

> gauss := proc(f, a, b, d)

>     # f   : la fonction a integrer

>     # a,b : les bornes de l'intervalle

>     # d   : le degré du polynôme orthogonal utilisé

>     #        i.e. le nombre d'appels de la fonction f

>      local p,r,c,scal,m,h,i ;

>      # p : le polynome orthogonal de degre d
```

```
>      # r : les racines du polynome p
>      # a : les coefficients associes aux racines de p
>      scal := prod_scal_legendre ;
>      p := evalf(poly_legendre(d, x)) ;
>      r := [solve(p=0)] ;
>      c := [seq(poids(p, i, scal, x), i=r)] ;
>      m := (a+b)/2 ;
>      h := (b-a)/2 ;
>      RETURN(h * sum(c[i] * f(m + h*r[i]), i=1..d)) ;
> end :
>
# Un petit test numérique.
> gauss(sin, 0, evalf(Pi), 6) = int(sin, 0..Pi) ;
```

$$1.99999999950 = 2$$

```
>
# Laissons Maple calculer l'erreur élémentaire de cette méthode
# en considérant les racines du n-ième polynôme orthogonal.
# Comme la méthode est "uniquement numérique", il faut
# avoir un certain sens critique sur les valeurs des coefficients du
# développement de Taylor de l'erreur.
>
> n := 9 ;
```

$$n := 9$$

```
>
> Digits := 100 ;
```

$$Digits := 100$$

```
> int_elem := int(f(x), x=c..c+h) : aapp_elem := gauss(f, c, c+h, n) :
```

```
> erreur_elem := taylor(int_elem - app_elem, h=0, 2*n+2) ;
```

$$erreur\_elem := -0.6\,10^{-98}\,e^c\,h - 0.35\,10^{-98}\,e^c\,h^2 - 0.13\,10^{-98}\,e^c\,h^3 - 0.33\,10^{-99}\,e^c\,h^4 - 0.67\,10^{-100}\,e^c\,h^5 - 0.11\,10^{-100}\,e^c\,h^6 - 0.18\,10^{-101}\,e^c\,h^7 - 0.21\,10^{-102}\,e^c\,h^8 - 0.25\,10^{-103}\,e^c\,h^9 - 0.26\,10^{-104}\,e^c\,h^{10} - 0.23\,10^{-105}\,e^c\,h^{11} - 0.20\,10^{-106}\,e^c\,h^{12} - 0.15\,10^{-107}\,e^c\,h^{13} - 0.10\,10^{-108}\,e^c\,h^{14} - 0.68\,10^{-110}\,e^c\,h^{15} - 0.44\,10^{-111}\,e^c\,h^{16} - 0.26\,10^{-112}\,e^c\,h^{17} - 0.14\,10^{-113}\,e^c\,h^{18} + 0.34775667098146311318494864\backslash 83825444756652337513421076310982259762146822916147770196348\backslash 8063310^{-26}e^c h^{19} + \mathrm{O}(h^{20})$$

```
>
# Terminons sur une test de comparaison entre différentes méthodes...
> vieil_echo := interface(echo, echo=1)[1] :
```

lecture de newcoat.mpl
lecture de simpson.mpl
lecture de median.mpl
lecture de rectang.mpl
lecture de trapez.mpl

```
> f := exp ;
```

$$f := \exp$$

```
((a,b) := (-1, 1) ;
```

$$a,\, b := -1,\, 1$$

```
>
> Digits := 30 : iintegrale := int(f, a..b) ;
```

$$integrale := -e^{(-1)} + e$$

```
>
# Testons différentes manières d'approximer cette intégrale
# avec exactement 9 appels (faut pas tricher quand même !)
# de la fonctions f.
> n := 9 ;
```

$$n := 9$$

```
> evalf(integrale - rectangles1(f, a, b, n)) ;
```

$$0.25149133450746515470388896086 7$$

```
>
> evalf(integrale - trapezes1(f, a, b, n-1)) ;
```
$$-0.012228946297607355468327086157$$

```
>
> evalf(integrale - median1(f, a, b, n)) ;
```
$$0.004829262212738744598804851844$$

```
> evalf(integrale - simpson1(f, a, b, (n-1)/2 )) ;
```
$$-0.000050629954676553724298024667$$

```
>
> evalf(integrale - NewtonCotes(f, a, b, 2, (n-1)/2 )) ;
```
$$-0.11756454781422372842227\,12\,10^{-5}$$

```
## degré 4
> evalf(integrale - NewtonCotes(f, a, b, 1, n-1)) ;
```
$$-0.12321877468637589382\,44\,10^{-8}$$

```
      # degré n-1
> evalf(integrale - gauss(f, a, b, n)) ;
```
$$0.184610871\,10^{-20}$$

```
# C'est sans commentaire !
>
# Allez, encore un petit exemple, maintenant que l'on a compris..
> f := x -> exp(sin(x*Pi/2)) ;
```
$$f := x \to e^{\sin(1/2\,x\,\pi)}$$

```
((a, b) := (-1, 1) ;
```
$$a,\, b := -1,\, 1$$

```
>
> n := 17 ;
```
$$n := 17$$

```
> Digits := 60 : iintegrale := evalf(Int(f, a..b)) ;
```

$$integrale := 2.532131755504016671196489250429435075215340622709924413616 27$$

```
> evalf(integrale - rectangles1(f, a, b, n)) ;
```

$$0.13825896395809428904498610007007068413596682133262875678326 4$$

```
>
```

```
> evalf(integrale - trapezes1(f, a, b, n-1)) ;
```

$$-0.3566302075086593\,10^{-44}$$

```
>
```

```
> evalf(integrale - median1(f, a, b, n)) ;
```

$$0.794286773160\,10^{-48}$$

```
> evalf(integrale - simpson1(f, a, b, (n-1)/2 )) ;
```

$$0.9867867048055316669259047736922296156003 34\,10^{-18}$$

```
>
```

```
> evalf(integrale - NewtonCotes(f, a, b, 1, n-1)) ;
```

$$-0.5321964264132712003929950801915659318601366343995 65\,10^{-7}$$

```
## degré n-1
```

```
> evalf(integrale - gauss(f, a, b, n)) ;
```

$$0.7622768524077589266336775186587351156365054\,10^{-18}$$

```
# Very strange, isn't it ?
```

```
>
```

```
> f := x -> 1/(2+cos(x*Pi)) ;
```

$$f := x \to \frac{1}{2+\cos(x\,\pi)}$$

```
((a, b) := (-1, 1) ;
```

$$a,\, b := -1,\, 1$$

```
>
```

```
# ici, on intègre f sur une période entière : f(a) = f(b)
```

```
> n := 8 ;
```

$$n := 8$$

```
> Digits := 10 : iintegrale := evalf(Int(f, a..b)) ;
```

$$integrale := 1.154700538$$

```
> evalf(integrale - rectangles1(f, a, b, n)) ;
```

$$-0.0000613666$$

```
>

> evalf(integrale - trapezes1(f, a, b, n)) ;
```

$$-0.0000613666$$

```
>

> evalf(integrale - median1(f, a, b, n)) ;
```

$$0.0000613630$$

```
> evalf(integrale - simpson1(f, a, b, n/2 )) ;
```

$$0.0039068873$$

```
>

> evalf(integrale - NewtonCotes(f, a, b, 1, n)) ;
```

$$-0.0022183245$$

```
## degré n

> evalf(integrale - gauss(f, a, b, n)) ;
```

$$-0.0000480828$$

```
# Il faut un commentaire sur les erreurs des méthodes

# des rectangles, trapezes, et point médian ! cf. la super-convergence...
```

Chapitre 4

ÉQUATIONS DIFFÉRENTIELLES

4.1 Méthode de Picard

En supposant que le problème de Cauchy

$$y'(x) = f(x, y(x)) \qquad y(x_0) = y_0$$

a une solution unique sur un certain intervalle réel contenant x_0, la méthode de Picard en fournit des solutions approximatives. En partant de la fonction constante $y_0 : x \mapsto y0$, la méthode de Picard calcule la `n`-ième fonction générée par le principe récurrent qui fait correspondre à $x \mapsto y(x)$ la fonction

$$x \mapsto y0 + \int_{x0}^{x} f(t, y(t))\ \mathrm{d}t$$

Bien que sur le plan théorique, cette méthode soit le fondement du théorème d'existence et d'unicité de la solution du problème, des complications dues à l'intégration surgissent dans la pratique.

```
> restart : interface(echo=3) : read ‘equadiff1.mpl‘ ;

# L’idée de base est très simple. L’intégration de

# l’équation différentielle conduit à :

>

> i := ( y(x) = y0 + int(f(t, y(t)), t=x0..x) ) ;
```

$$i := \mathrm{y}(x) = y0 + \int_{x0}^{x} \mathrm{f}(t, \mathrm{y}(t))\, dt$$

```
>

# Cette relation confirme l’équation différentielle et
```

```
# la condition initiale :

> diff(i, x) ;
```

$$\frac{\partial}{\partial x}\,\mathrm{y}(x) = \mathrm{f}(x,\,\mathrm{y}(x))$$

```
> simplify(subs(x=x0, i)) ;
```

$$\mathrm{y}(x0) = y0$$

```
>

# Afin de trouver les approximations de la solution y(x),

# Picard commence avec la fonction constante y1(x) = y0 en tant

# que première solution approximative. Ensuite il remplace dans

# l'équation différentielle l'argument y(t) par y1(t) = y0.

>

# Regardons cela sur un exemple :

> f := (x,y) -> y ; (x0, y0) := (0, 1) ;
```

$$f := (x,\,y) \to y$$

$$x0,\,y0 := 0,\,1$$

```
> yt := subs(x=t, y0) ; y1 := y0 + int(f(t, yt), t=x0..x) ;
```

$$yt := 1$$

$$y1 := 1 + x$$

```
>

# En second lieu, Picard substitue dans l'intégrale l'expression y0

# par la nouvelle approximation y1.

> yt := subs(x=t, y1) ; y2 := y0 + int(f(t, yt), t=x0..x) ;
```

$$yt := 1 + t$$

$$y2 := 1 + x + \frac{1}{2}\,x^2$$

```
>

# Et ainsi de suite...

> yt := subs(x=t, y2) ; y3 := y0 + int(f(t, yt), t=x0..x) ;
```

$$yt := 1 + t + \frac{1}{2}\,t^2$$

$$y3 := 1 + x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3$$

```
> yt := subs(x=t, y3) ; y4 := y0 + int(f(t, yt), t=x0..x) ;
```

$$yt := 1 + t + \frac{1}{2}\,t^2 + \frac{1}{6}\,t^3$$

$$y4 := 1 + x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3 + \frac{1}{24}\,x^4$$

```
> yt := subs(x=t, y4) ; y5 := y0 + int(f(t, yt), t=x0..x) ;
```

$$yt := 1 + t + \frac{1}{2}\,t^2 + \frac{1}{6}\,t^3 + \frac{1}{24}\,t^4$$

$$y5 := 1 + x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3 + \frac{1}{24}\,x^4 + \frac{1}{120}\,x^5$$

```
>
# Ce processus va continuer de générer la suite
# d'expressions y0, y1, y2, ..., yn, ...
> Picard := proc(f, x0, y0, n)
>           # f : une fonction de R^2 -> R
>           # (x0, y0) : un point de R^2
>           # n : le nombre d'iterations de la fonction ``Picard''
>           local i,t,x,yt,yx ;
>           # yt : une expression en t
>           # yx : une expression en x
>             yx := y0 ;
>             for i from 1 to n do
>               yt := subs(x=t, yx) ;
>               yx := y0 + int(f(t, yt), t=x0..x) ;
>             od ;
>             unapply(yx, x) ;
>           end :
>
# Essayons :
```

```
> Picard((x,y) -> y, 0, 1, 10) ;
```

$$x \to 1 + x + \frac{1}{2}\,x^2 + \frac{1}{6}\,x^3 + \frac{1}{24}\,x^4 + \frac{1}{120}\,x^5 + \frac{1}{720}\,x^6 + \frac{1}{5040}\,x^7 + \frac{1}{40320}\,x^8 + \frac{1}{362880}\,x^9 + \frac{1}{3628800}\,x^{10}$$

```
> Picard((x,y) -> sin(x)*y, 0, 1, 5) ;
```

$$x \to \frac{163}{60} - \frac{65}{24}\cos(x) + \frac{4}{3}\cos(x)^2 - \frac{5}{12}\cos(x)^3 + \frac{1}{12}\cos(x)^4 - \frac{1}{120}\cos(x)^5$$

```
> Picard((x,y) -> y/x, 1, 1, 5) ;
```

$$x \to 1 + \ln(x) + \frac{1}{2}\ln(x)^2 + \frac{1}{6}\ln(x)^3 + \frac{1}{24}\ln(x)^4 + \frac{1}{120}\ln(x)^5$$

```
>

# Des petits dessins ?

# Exemple 1 :

> f := (x,y) -> y^2+1 ;
```

$$f := (x,\, y) \to y^2 + 1$$

```
> (x0, y0) := (0, 0) ;
```

$$x0,\, y0 := 0,\, 0$$

```
>

# Des solutions approchées

> sP := seq(Picard(f, x0, y0, i)(x), i=1..3) ;
```

$$sP := x,\, \frac{1}{3}\,x^3 + x,\, \frac{1}{63}\,x^7 + \frac{2}{15}\,x^5 + \frac{1}{3}\,x^3 + x$$

```
>

# On peut utiliser les routines de Maple

> s := dsolve({diff(z(x), x) = f(x,z(x)), z(x0)=y0}, z(x)) ;
```

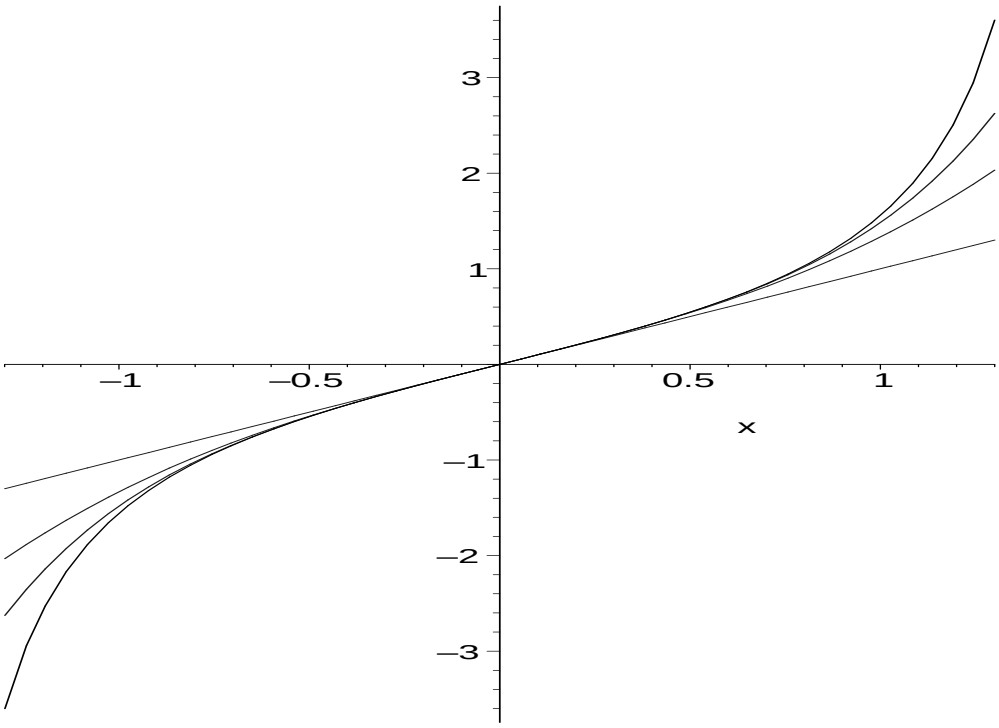
$$s := \mathrm{z}(x) = \tan(x)$$

```
> sd := rhs(s) ;
```

$$sd := \tan(x)$$

```
>

# Comparons

> with(plots) :
```

```
Warning, the name changecoords has been redefined

> solution := plot(sd, x=-1.3 .. 1.3, color=black) :

> approx := plot([sP], x=-1.3 .. 1.3, color=[blue]) :

> display([solution, approx]) ;
```




```
>

# Autres exemples 2 :

> f := (x,y) -> y/x ;
```

$$f := (x,\, y) \to \frac{y}{x}$$

```
#f := (x,y) -> y ;

> (x0, y0) := (1, 1) ;
```

$$x0,\, y0 := 1,\, 1$$

```
> sP := seq(Picard(f, x0, y0, i)(x), i=1..5) ;
```

$$sP := 1 + \ln(x),\ 1 + \ln(x) + \frac{1}{2}\ln(x)^2,\ 1 + \ln(x) + \frac{1}{2}\ln(x)^2 + \frac{1}{6}\ln(x)^3,$$
$$1 + \ln(x) + \frac{1}{2}\ln(x)^2 + \frac{1}{6}\ln(x)^3 + \frac{1}{24}\ln(x)^4,$$
$$1 + \ln(x) + \frac{1}{2}\ln(x)^2 + \frac{1}{6}\ln(x)^3 + \frac{1}{24}\ln(x)^4 + \frac{1}{120}\ln(x)^5$$

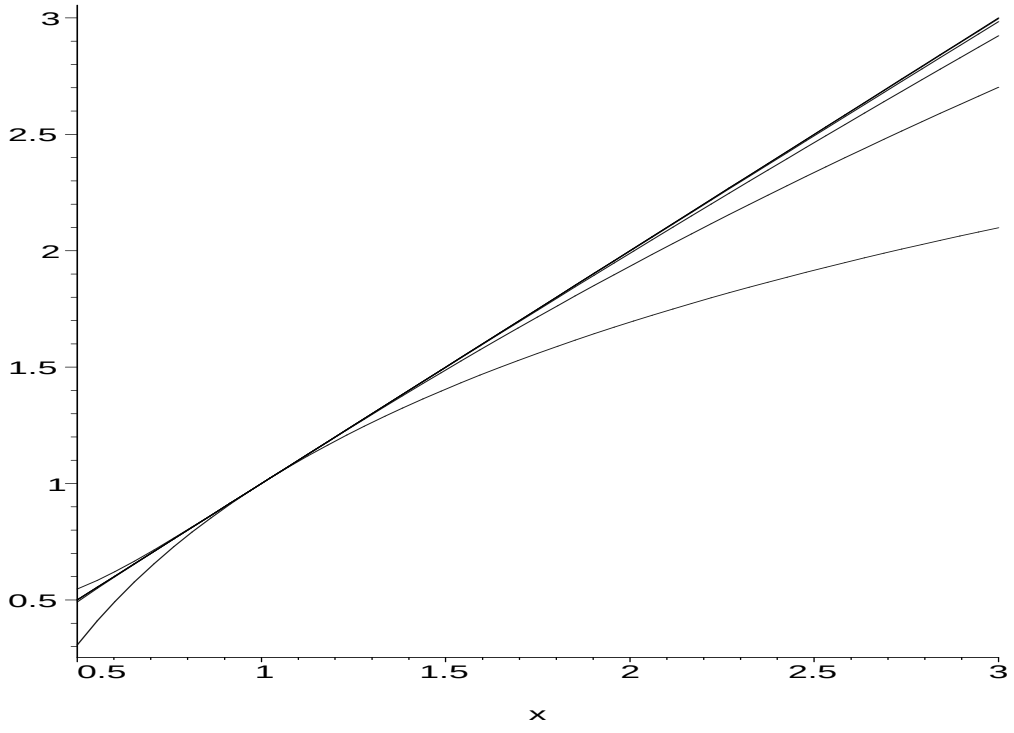
```
> s := dsolve({diff(z(x), x) = f(x,z(x)), z(x0)=y0}, z(x)) ;
```

$$s := \mathrm{z}(x) = x$$

```
> sd := rhs(s) ;
```

$$sd := x$$

```
> solution := plot(sd, x=0.5..3, color=black) :

> approx := plot([sP], x=0.5..3, color=[blue]) :

> display([solution, approx]) ;
```



4.2 Méthodes d'Euler et de Nyström

La méthode d'Euler classique est la méthode de Runge-Kutta explicite la plus simple : en partant d'un point origine $\binom{x_0}{y_0} \in R^2$ et d'une équation différentielle $y'(x) = f(x, y(x))$, on se fixe un pas h et on construit la suite

$$y_{k+1} = y_k + h.f(x_k, y_k) \qquad \text{où} \qquad x_k = x_0 + k.h$$

La valeur y_{k+1} est calculée à partir de x_k et y_k. On approxime ainsi la solution $x \mapsto y(x)$ (supposée unique) du problème de Cauchy, mais l'ordre de convergence de cette méthode est seulement 1.

La méthode de Nyström consiste à calculer y_{k+1} en fonction de x_k, y_k et y_{k-1} par la relation suivante :

$$y_{k+1} = y_{k-1} + 2h.f(x_k, y_k) \qquad \text{où} \qquad x_k = x_0 + k.h$$

La valeur y_1 est obtenue simplement par $y_1 = y_0 + h.f(x_0, y_0)$. L'ordre de convergence de cette méthode améliorée passe alors à 2.

```
>  restart : read ‘my_config.mpl‘ : interface(echo=3) :
>  read ‘equadiff2.mpl‘ ;

# Méthode d’Euler : en partant d’un point origine (x_0, y_0)

# et d’une équation différentielle y’(x) = f(x, y(x)),

# on se fixe un pas h et on construit la suite

# y_{k+1} = y_k + h.f(x_k, y_k)   avec   x_k = x_0 + k.h

>

> Euler := proc(f, x0, y0, h, n)

>      # f : une fonction R^2 -> R

>      # (x0, y0) : un point de R^2

>      # h : le pas de la méthode

>      # n : le nombre de pas

>      # Cette procédure calcule une suite de points approximant

>      # la courbe de la solution y

>      local i,y ;

>        # y : la suite des valeurs approximant la solution
```

```
>        y[0] := y0 ;
>        for i from 0 to n-1 do
>           y[i+1] := y[i] + h*f(x0+i*h,y[i]) ;
>        od ;
>        RETURN([ seq([x0+i*h, y[i]], i=0..n) ]) ;
> end :
>
# Méthode de Nyström : en partant d'un point origine (x_0, y_0)
# on construit la suite
# y_{k+1} = y_{k-1} + 2h.f(x_k, y_k)  avec  y_1 = y_0 + h.f(x_0,y_0)
>
> Nystrom := proc(f, x0, y0, h, n)
>      # f : une fonction R^2 -> R
>      # (x0, y0) : un point de R^2
>      # h : le pas de la methode
>      # n : le nombre de pas
>      # Cette procédure calcule une suite de points approximant
>      # la courbe de la solution y
>      local i,y ;
>        # y : la suite des valeurs approximant la solution
>        y[0] := y0 ;
>        y[1] := y0+h*f(x0,y0) ;
>        for i from 1 to n-1 do
>           y[i+1] := y[i-1] + 2*h*f(x0+i*h,y[i]) ;
>        od ;
>        RETURN([ seq([x0+i*h, y[i]], i=0..n) ]);
```

```
> end :
>
# Déclarons un problème de Cauchy générique :
> D(y) := x -> f(x, y(x)) ;
```

$$\mathrm{D}(y) := x \to \mathrm{f}(x,\, \mathrm{y}(x))$$

```
>
# Étudions la convergence de la méthode d'Euler classique :
# en partant de (a,y(a)), on calcule une approximation de y(b)
# après un nombre de pas fixé à n. On constate que
# l'erreur commise est équivalente à   ( y'(b)-y'(a) ).(b-a) / 2.n
# quand b est proche de a.
>
# On fait varier le nombre de pas numériquement :
# pas de calcul symbolique pour une fois...
> for n from 1 to 6 do
>   yn := Euler(f, a, y(a), (b-a)/n, n)[n+1][2] ;
>   dl := taylor( (y(b) - yn) * n / ( D(y)(b) - D(y)(a) ) , b=a, 3) ;
>   print(simplify(dl)) ;
> od :
```

$$\frac{1}{2}\,(b-a) + \mathrm{O}((b-a)^2)$$
$$\frac{1}{2}\,(b-a) + \mathrm{O}((b-a)^2)$$
$$\frac{1}{2}\,(b-a) + \mathrm{O}((b-a)^2)$$
$$\frac{1}{2}\,(b-a) + \mathrm{O}((b-a)^2)$$
$$\frac{1}{2}\,(b-a) + \mathrm{O}((b-a)^2)$$
$$\frac{1}{2}\,(b-a) + \mathrm{O}((b-a)^2)$$

```
>
```

```
# Étudions la convergence de la méthode de Nyström :
# en partant de (a,y(a)), on calcule une approximation de y(b)
# après un nombre de pas fixé à n. On constate que
# l'erreur commise est équivalente à
# ( y'(b) - y'(a) ).(b-a) / 2.n^2    pour n est impair,
# ( y'(b) - y'(a) )       /   n^2    pour n est pair,
# quand b est proche de a.
# La convergence est d'ordre 2 (une fois a et b fixés).
>
# On fait varier le nombre de pas...
> for n from 1 to 6 do
>   yn := Nystrom(f, a, y(a), (b-a)/n, n)[n+1][2] ;
>   dl := taylor( (y(b) - yn) * n^2 / ( D(y)(b) - D(y)(a) ), b=a, 3 );
>   print(simplify(dl)) ;
> od :
```

$$\frac{1}{2}(b-a)+\mathrm{O}((b-a)^2)$$

$$\mathrm{O}((b-a)^2)$$

$$\frac{1}{2}(b-a)+\mathrm{O}((b-a)^2)$$

$$\mathrm{O}((b-a)^2)$$

$$\frac{1}{2}(b-a)+\mathrm{O}((b-a)^2)$$

$$\mathrm{O}((b-a)^2)$$

```
>
# Testons ces méthodes sur un exemple numérique :
> f := (x,y) -> sin(6*x)*y-y/10 ;
```

$$f := (x,\, y) \to \sin(6\,x)\,y - \frac{1}{10}\,y$$

```
xx0 := 0 ;
```

$$x0 := 0$$

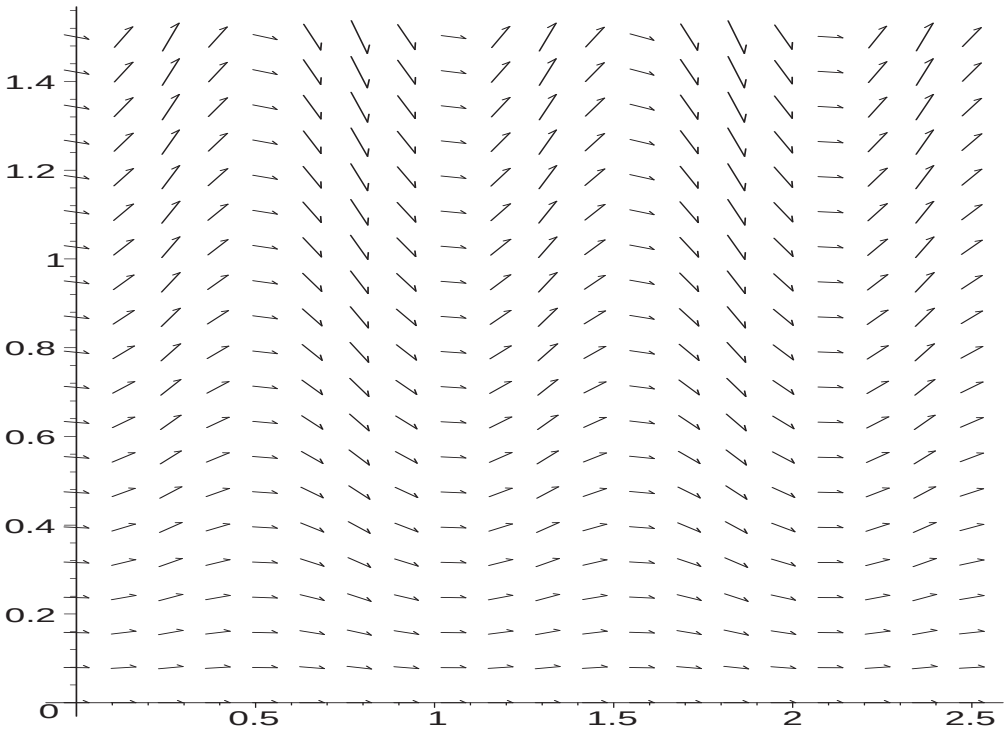
```
yy0 := 1 ;
```

$$y0 := 1$$

```
xx1 := 2.5 ;
```

$$x1 := 2.5$$

```
>

>

# Traçons ce champ de vecteur :

> with(plots) :

Warning, the name changecoords has been redefined

> fieldplot([1, f], x0 .. x1, 0 .. 1.5) ;
```




```
> pour_le_fichier(%, ‘equadiff2‘) :

>

# Calculons la vraie solution
```

```
> sol := dsolve({diff(y(x) , x) = f(x, y(x)), y(x0) = y0}, y(x)) ;
```

$$sol := \mathrm{y}(x) = \frac{e^{(-1/6\cos(6\,x)-\frac{x}{10})}}{e^{(-1/6)}}$$

```
>
> yx := rhs(sol) ;
```

$$yx := \frac{e^{(-1/6\cos(6\,x)-\frac{x}{10})}}{e^{(-1/6)}}$$

```
>
# Constatons l'ordre de convergence des deux méthodes
# Un petit graphique pour comparer le solution exacte et
# les approximations...
>
# En appliquant plusieurs fois ces deux méthodes, on obtient
# une suite de points d'approximation
>
> n := 20 ;
```

$$n := 20$$

```
hh := (x1-x0)/n ;
```

$$h := 0.1250000000$$

```
> pts1_Euler := Euler(f, x0, y0, h, n) ;
```

$$pts1\_Euler := [[0.,\ 1],\ [0.1250000000,\ 0.9875000000],\ [0.2500000000,\ 1.059296034],$$
$$[0.3750000000,\ 1.178135144],\ [0.5000000000,\ 1.277992877],$$
$$[0.6250000000,\ 1.284561762],\ [0.7500000000,\ 1.176729013],$$
$$[0.8750000000,\ 1.018233894],\ [1.000000000,\ 0.8961814436],$$
$$[1.125000000,\ 0.8536783025],\ [1.250000000,\ 0.8910314314],$$
$$[1.375000000,\ 0.9843669713],\ [1.500000000,\ 1.085585023],$$
$$[1.625000000,\ 1.127938917],\ [1.750000000,\ 1.068789914],$$
$$[1.875000000,\ 0.9379037956],\ [2.000000000,\ 0.8127161488],$$
$$[2.125000000,\ 0.7480470125],\ [2.250000000,\ 0.7557705170],$$
$$[2.375000000,\ 0.8222579570],\ [2.500000000,\ 0.9141083953]]$$

```
> pts1_Nystrom := Nystrom(f, x0, y0, h, n) ;
```

$pts1\_Nystrom := [[0., 1], [0.1250000000, 0.9875000000], [0.2500000000, 1.143592069],$
$[0.3750000000, 1.244092037], [0.5000000000, 1.354488435],$
$[0.6250000000, 1.258016181], [0.7500000000, 1.143279684],$
$[0.8750000000, 0.9500366080], [1.000000000, 0.9155239656],$
$[1.125000000, 0.8631956126], [1.250000000, 0.9910630928],$
$[1.375000000, 1.070823325], [1.500000000, 1.211279037],$
$[1.625000000, 1.165338970], [1.750000000, 1.089058521],$
$[1.875000000, 0.8986024662], [2.000000000, 0.8491747960],$
$[2.125000000, 0.7634620468], [2.250000000, 0.8649401221],$
$[2.375000000, 0.9156448938], [2.500000000, 1.069504600]]$

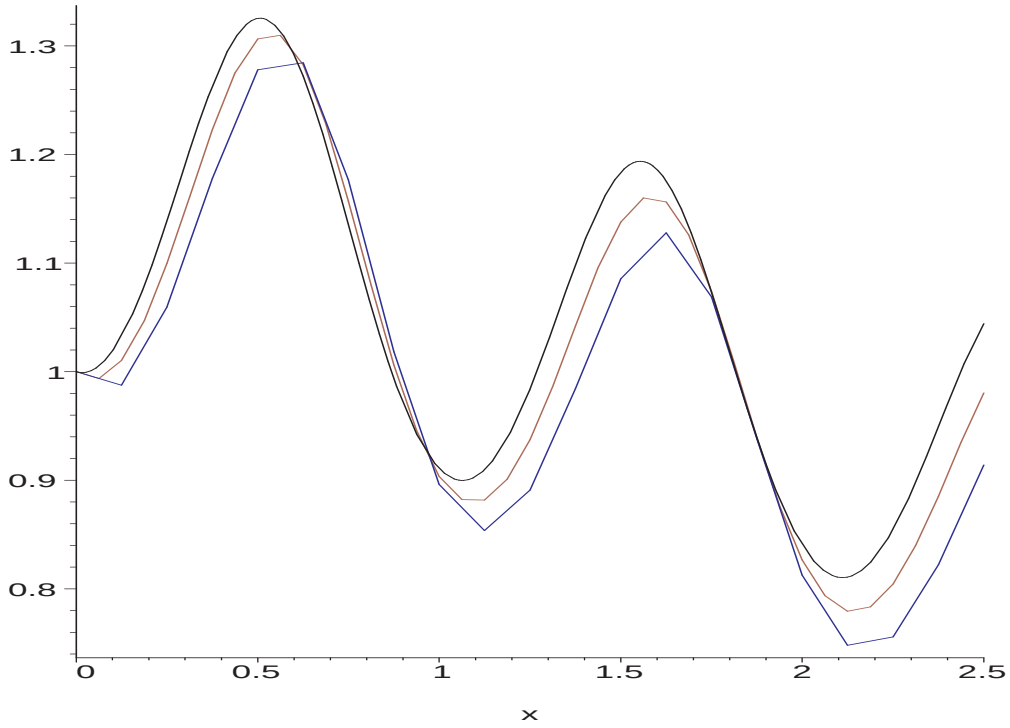
```
> n := 2*n ;
```

$$n := 40$$

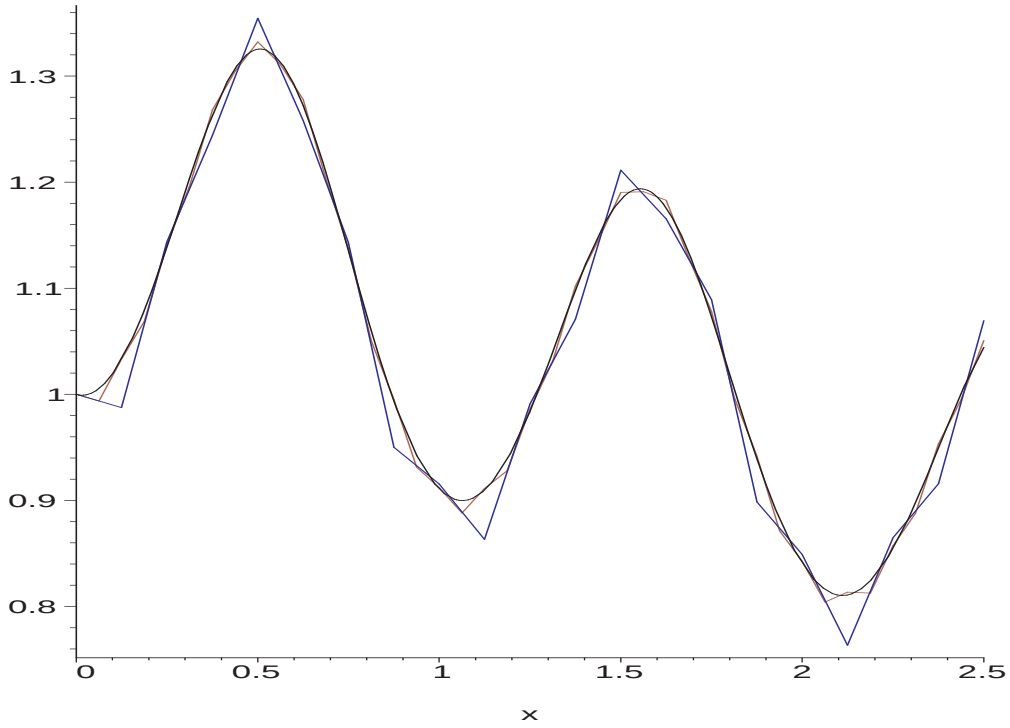
```
hh := (x1-x0)/n ;
```

$$h := 0.06250000000$$

```
> pts2_Euler := Euler(f, x0, y0, h, n) :

> pts2_Nystrom := Nystrom(f, x0, y0, h, n) :

>

> plot([yx, pts1_Euler, pts2_Euler], x=x0..x1,

>                       color=[black,blue,brown]) ;
```



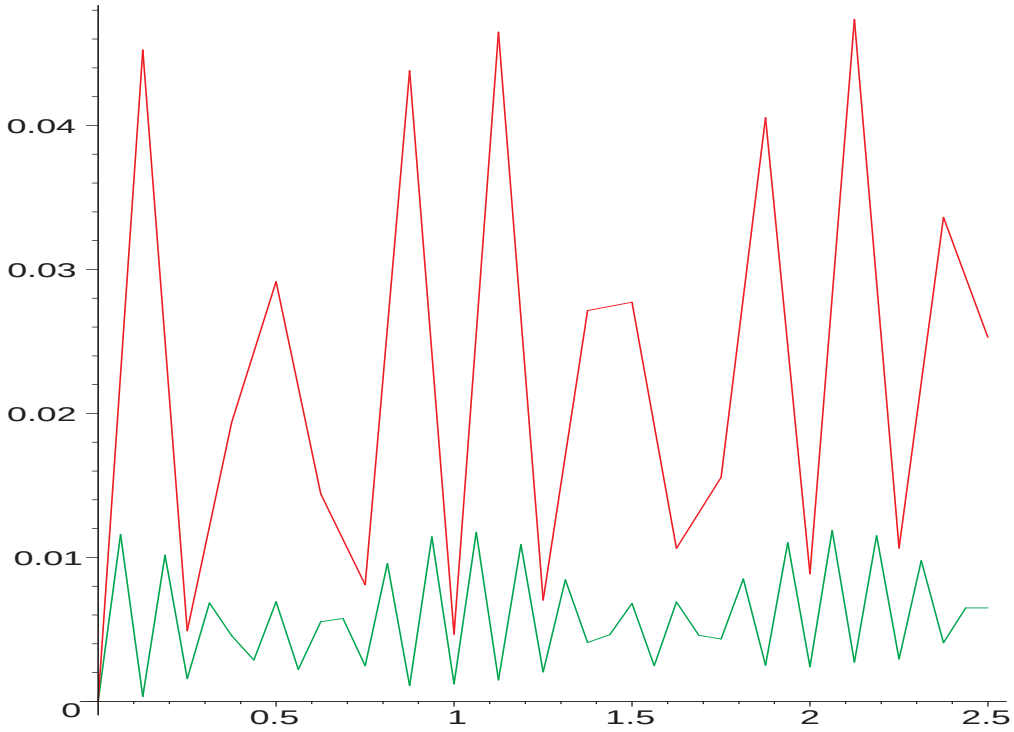

```
> pour_le_fichier(%, ‘equadiff2‘) :
>
# Visiblement, en passant de  n  à  2n  pas (intervalle constant)
# l’erreur commise pa la méthode d’Euler est divisée par 2 :
# l’ordre de convergence est bien log_2(2) = 1...
>
> plot([yx, pts1_Nystrom, pts2_Nystrom], x=x0..x1,
>                          color=[black,blue,brown]) ;
```




```
> pour_le_fichier(%, ‘equadiff2‘) :

>

# Visiblement... pas évident de conclure !

# Visualisons les différences entre f(x) et les approximations

>

> y := unapply(evalf(yx), x) ;
```

$$y := x \to 1.181360413\, e^{(-0.1666666667\cos(6.\,x) - 0.1000000000\,x)}$$

```
>

> d1 := [seq([ i[1], abs( y(i[1]) - i[2] ) ], i= pts1_Nystrom)] :

> d2 := [seq([ i[1], abs( y(i[1]) - i[2] ) ], i= pts2_Nystrom)] :

> plot([d1,d2]);
```




```
> pour_le_fichier(%, ‘equadiff2‘) :
>
# Visiblement, l’erreur est divisée par 4 lorsque h est divisé par 2 :
# La convergence a l’air quadratique.
>
# Essayons avec des calculs statistiques :
> with(stats) : wwith(describe) :
# Erreurs moyennes sur les deux approximations :
> e1 := mean([seq( i[2], i=d1)]) ;
```

$$e1 := 0.02240657346$$

```
> e2 := mean([seq( i[2], i=d2)]) ;
```

$$e2 := 0.005682280127$$

```
# Ce rapport est proche de 2^2 = 4 :
> e1/e2 ;
```

$$3.943236335$$

4.3 Méthode de Runge-Kutta

On considère l'équation différentielle réelle $y' = f(x, y)$ avec la condition $y(x_0) = y_0$. On suppose que ce problème à une solution unique. On se fixe un pas h (i.e. $x_{k+1} = x_k + h$ pour tout k) et on construit une suite $(y_k)_{k\geqslant 0}$ telle que y_k soit le plus proche possible de $y(x_k)$. On sait qu'on a l'églité

$$y(x_{n+1}) = y(x_n) + \int_{x_n}^{x_{n+1}} f(x, y(x))\ \mathrm{d}x$$

En admettant que l'on connaisse $y(x_n)$, le problème revient donc à approximer l'intégrale. La stratégie de Runge-Kutta consiste à approximer celle-ci par $h\sum_{j=1}^{d} b_j f(P_{n,j})$ où les abscisses des points $P_{n,i}$ appartiennent à l'intervalle $[x_n, x_{n+1}]$. On pose $P_{n,1} = (x_n, y_n)$. Il faut maintenant déterminer de bons points $P_{n,2}$, $P_{n,3}$... Pour cela, on utilise une matrice carrée

$$A = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ * & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ * & \cdots & * & 0 \end{pmatrix}$$

codant le principe d'obtention de $P_{n,i} = (t_i, z_i)$ à partir de $P_{n,1}, \ldots, P_{n,i-1}$:

$$t_i = x_n + h\sum_{j<i} A_{ij} \qquad \text{et} \qquad z_i = y_n + h\sum_{j<i} A_{ij} f(t_j, z_j) = \sum_{j<i} A_{ij} f(P_j)$$

Par la suite, nous considérerons une matrice de dimension $(d+1) \times d$.

$$M = \begin{pmatrix} 0 & \cdots & \cdots & 0 \\ * & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ * & \cdots & * & 0 \\ b_1 & \cdots & \cdots & b_d \end{pmatrix}$$

```
> restart : interface(echo=3) : read ‘equadiff3.mpl‘ ;

# Voici quelques matrices correspondant à des méthodes

# de Runge - Kutta explicites.

> with(linalg):

Warning, the protected names norm and trace have been redefined and
unprotected
```

```
>
# Convergence d'ordre 1, erreur de consistance d'ordre 2
# Méthode d'Euler
> M21 := matrix([[0],[1]]) ;
```

$$M21 := \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

```
>
# Convergence d'ordre 2, erreur de consistance d'ordre 3
# Méthodes de Newton, de Heun et du point médian
> M31 := matrix([[0,0], [2/3,0], [1/4,3/4]]) ;
```

$$M31 := \begin{bmatrix} 0 & 0 \\ \frac{2}{3} & 0 \\ \frac{1}{4} & \frac{3}{4} \end{bmatrix}$$

```
> M32 := matrix([[0,0], [ 1 ,0], [1/2,1/2]]) ; # Méthode rétroactive
```

$$M32 := \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

```
> M33 := matrix([[0,0], [1/2,0], [ 0 , 1 ]]) ;
```

$$M33 := \begin{bmatrix} 0 & 0 \\ \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix}$$

```
>
# Convergence d'ordre 3, erreur de consistance d'ordre 4
# Méthode de Heun, et des matrices moins classiques
> M41 := matrix([[0,0,0], [1/3,0,0],[ 0, 2/3 ,0],[1/4 , 0  ,3/4 ]]);
```

$$M41 := \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 \\ 0 & \frac{2}{3} & 0 \\ \frac{1}{4} & 0 & \frac{3}{4} \end{bmatrix}$$

```
> M42 :matrix([[0,0,0],[1/3,0,0],[-5/12,15/12,0],[1/10,5/10,4/10]]);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 \\ \frac{-5}{12} & \frac{5}{4} & 0 \\ \frac{1}{10} & \frac{1}{2} & \frac{2}{5} \end{bmatrix}$$

```
> M43 := matrix([[0,0,0],[ 1 ,0,0],[ 1/4 , 1/4 ,0],[1/6 ,1/6 ,4/6]]);
```

$$M43 := \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ \frac{1}{4} & \frac{1}{4} & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{bmatrix}$$

```
> M44 := matrix([[0,0,0],[1/2,0,0],[  0   , 3/4 ,0],[2/9 ,3/9 ,4/9]]);
```

$$M44 := \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 \\ 0 & \frac{3}{4} & 0 \\ \frac{2}{9} & \frac{1}{3} & \frac{4}{9} \end{bmatrix}$$

```
>

# Convergence d'ordre 4, erreur de consistance d'ordre 5

# (la matrice classique) :

> M51 := matrix([[0,0,0,0],[1/2,0,0,0],[0,1/2,0,0],[0,0,1,0],

>                [1/6,2/6,2/6,1/6]]) ;
```

$$M51 := \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{bmatrix}$$

```
>

> runge_kutta := proc(f, x0, y0, h, M)

>     # f : une fonction R^2 -> R
```

```
>     # (x0, y0) : un point de R^2
>     # h : le pas de la méthode
>     # M : la matrice de la méthode
>     local e,i,j,z,t ;
>       # e : le nombre de lignes de la matrice M
>       e := rowdim(M) ;
>       t := [seq(x0+h*sum(M[i,j], j=1..i-1), i=1..e)] ;
>       for i from 1 to e do
>            z[i] := y0 + h*sum( M[i,j] * f(t[j], z[j]), j=1..i-1) ;
>       od ;
>       RETURN(z[e]) ;
> end :
>
# Essayons sur un exemple. La solution exacte du problème
> f := (x,y) -> (exp(x)+y)/2 ;
```

$$f := (x,\, y) \to \frac{1}{2}\, e^{x} + \frac{1}{2}\, y$$

```
>
# est
> y :='y' : e := dsolve( {diff(y(x), x) = f(x,y(x)), y(x0)=y0}, y(x));
```

$$e := \mathrm{y}(x) = e^{x} + \frac{e^{(1/2\,x)}\,(-e^{x0} + y0)}{e^{(1/2\,x0)}}$$

```
> e := unapply(rhs(e), x) ;
```

$$e := x \to e^{x} + \frac{e^{(1/2\,x)}\,(-e^{x0} + y0)}{e^{(1/2\,x0)}}$$

```
>
# L'approximation de cette solution sur l'intervalle [x0, x0+h] est
> z := runge_kutta(f, x0, y0, h, M51) ;
```

$$z := y0 + h(\frac{1}{12}\,e^{x0} + \frac{1}{2}\,y0 + \frac{1}{3}\,e^{(x0+1/2\,h)} + \frac{1}{6}\,h\,(\frac{1}{4}\,e^{x0} + \frac{1}{4}\,y0)$$
$$+ \frac{1}{6}\,h\,(\frac{1}{4}\,e^{(x0+1/2\,h)} + \frac{1}{4}\,y0 + \frac{1}{4}\,h\,(\frac{1}{4}\,e^{x0} + \frac{1}{4}\,y0)) + \frac{1}{12}\,e^{(x0+h)}$$
$$+ \frac{1}{12}\,h\,(\frac{1}{2}\,e^{(x0+1/2\,h)} + \frac{1}{2}\,y0 + \frac{1}{2}\,h\,(\frac{1}{4}\,e^{(x0+1/2\,h)} + \frac{1}{4}\,y0 + \frac{1}{4}\,h\,(\frac{1}{4}\,e^{x0} + \frac{1}{4}\,y0))))$$

```
>

# On constate bien l'ordre de l'erreur de consistance

# par un développement de Taylor (en h=0) de la différence entre

# la solution exacte et l'approximation avec un seul pas.

>

> difference := e(x0+h) - z : taylor(difference, h=0) ;
```

$$(\frac{1}{1440}\,e^{x0} + \frac{1}{3840}\,y0)\,h^5 + \mathrm{O}(h^6)$$

```
>

# ... et sur un exemple numérique :

> f := (x, y) -> 2*y*sin(6*x) - y/10 ; (x0, y0) := (0, 1) ;
```

$$f := (x,\, y) \to 2\,y\sin(6\,x) - \frac{1}{10}\,y$$
$$x0,\, y0 := 0,\, 1$$

```
>

# La solution exacte du problème est :

> e := dsolve( {diff(y(x), x) = f(x,y(x)), y(x0) = y0} , y(x)) ;
```

$$e := \mathrm{y}(x) = \frac{e^{(-1/3\cos(6\,x) - 1/10\,x)}}{\cosh(\frac{1}{3}) - \sinh(\frac{1}{3})}$$

```
> yx := rhs(e) ;
```

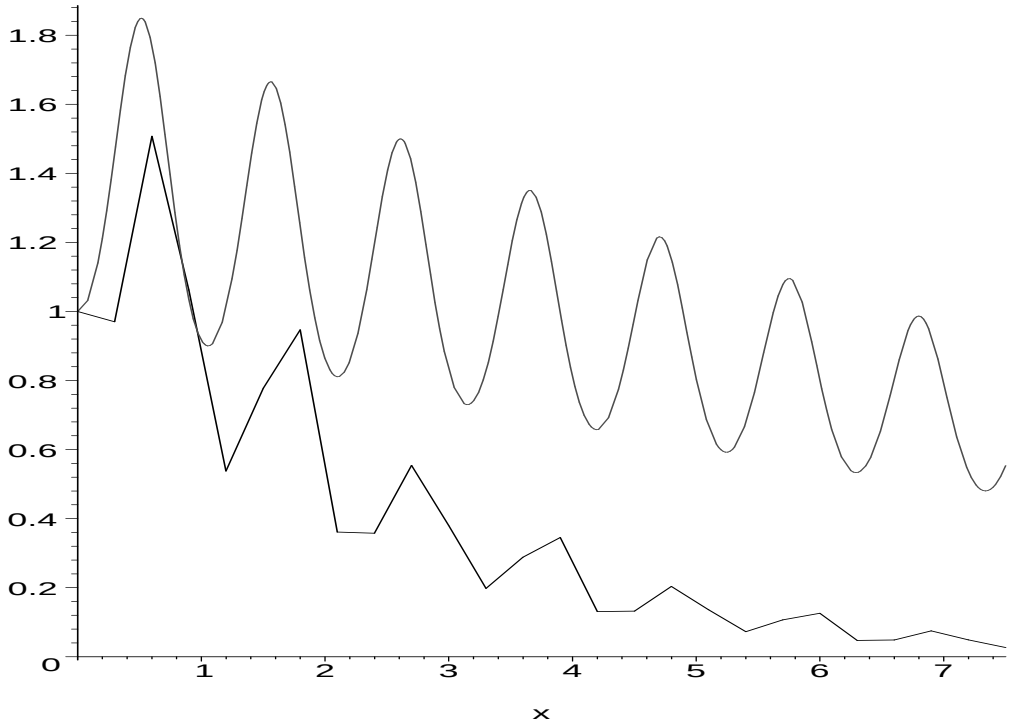
$$yx := \frac{e^{(-1/3\cos(6\,x) - 1/10\,x)}}{\cosh(\frac{1}{3}) - \sinh(\frac{1}{3})}$$

```
>

# En appliquant plusieurs fois la methode de Runge - Kutta avec

# un pas h, on obtient une suite de points d'approximation
```

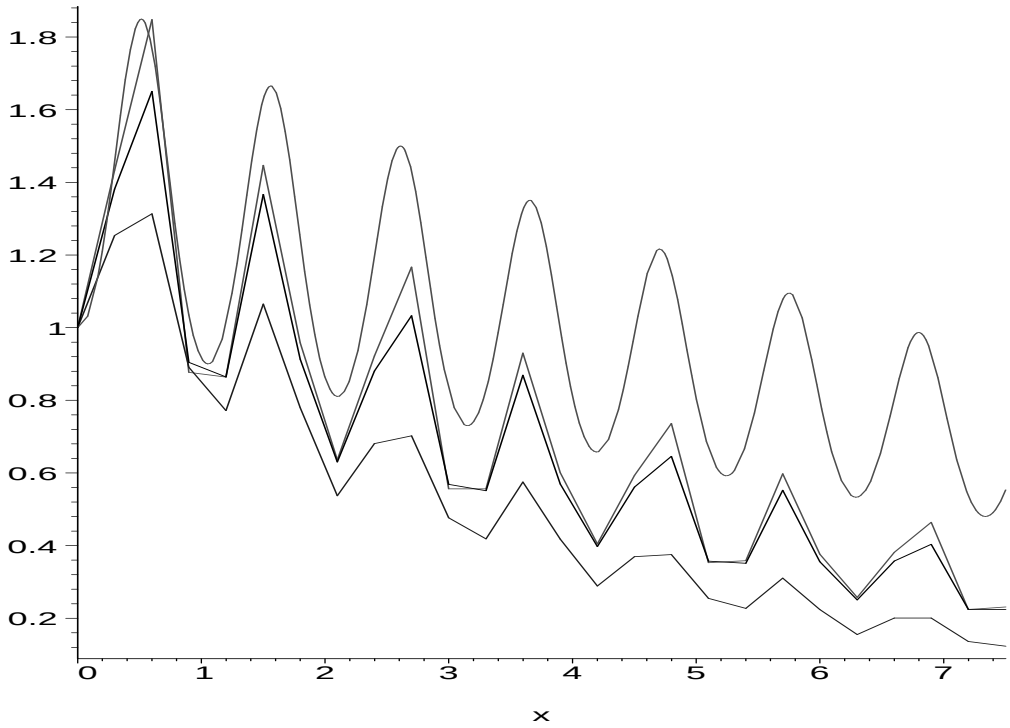
```
> points_rk := proc(f, x0, y0, h, n, M)
>       # f : une fonction de R^2 dans R
>       # (x0, y0) : un point de R^2
>       # h : le pas de la méthode
>       # n : le nombre d'itérations
>       # M : la matrice d'une méthode
>    local i, rk ;
>    rk[0] := y0 ;
>    for i from 1 to n do
>      rk[i] := runge_kutta(f, x0+(i-1)*h, rk[i-1], h, M) ;
>    od ;
>    RETURN([ seq( [x0+i*h, rk[i]], i=0..n) ]) ;
> end :
>
> h := 0.3 ; n := 25 ;
```

$$h := .3$$
$$n := 25$$

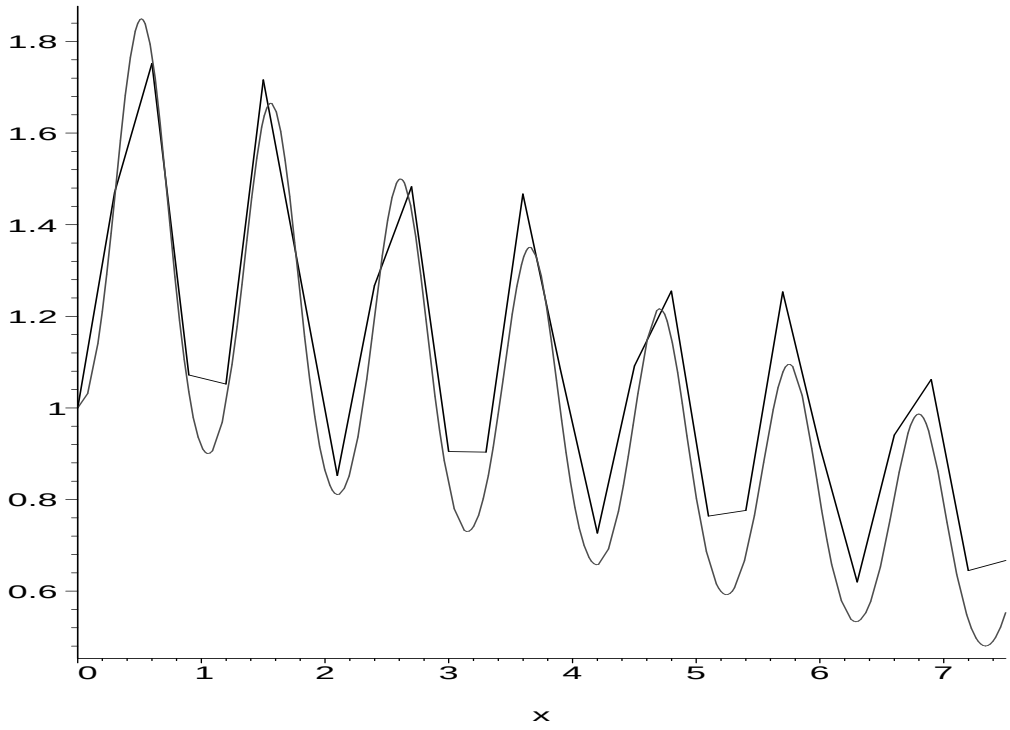
```
> for i in [21,31,32,33,41,51] do
>   pts||i := points_rk(f, x0, y0, h, n, M||i) ;
> od :
>
# Des petits graphiques pour comparer la solution exacte
# et les approximations
> plot([yx, pts21], x=x0..x0+n*h, color=[red,black]) ;
```



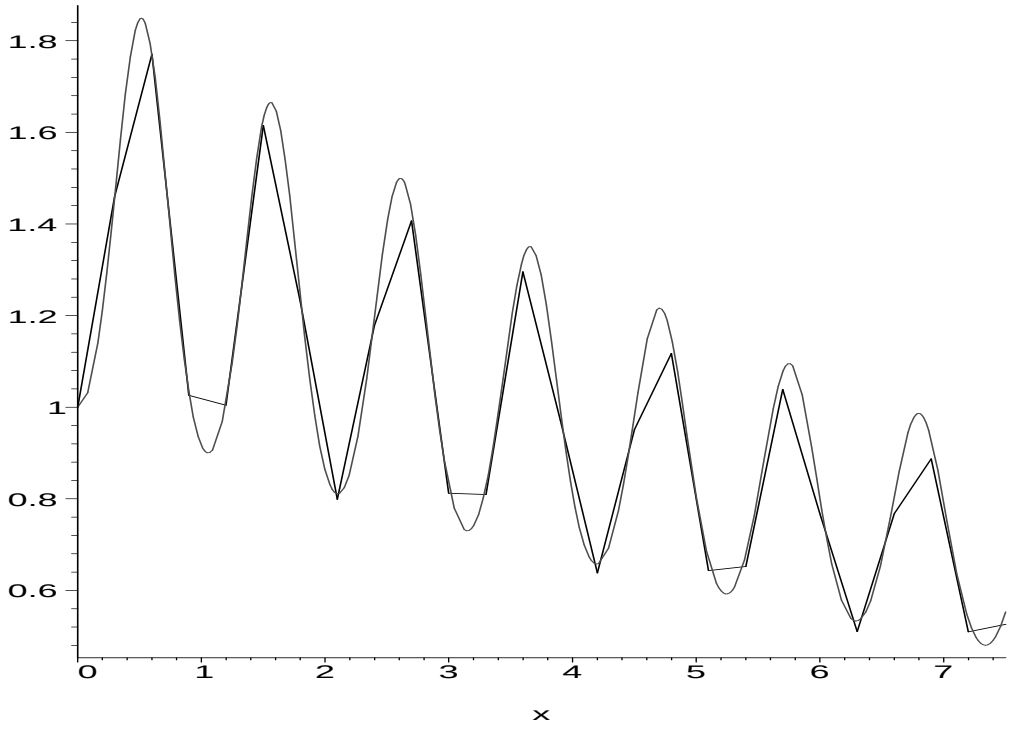

```
> plot([yx, pts31, pts32, pts33], x=x0..x0+n*h,
>                                 color=[red,black,blue,brown]) ;
```



```
> plot([yx, pts41], x=x0..x0+n*h, color=[red,black]) ;
```




```
> plot([yx, pts51], x=x0..x0+n*h, color=[red,black]) ;
```




```
# Malgré toutes les oscillations de la solution,
# la méthode RK-4 tient la route !
>
# Pour les versions V.3 et supérieures de Maple,
# la liste des points calculés avec la méthode RK donnée par M51
# peut également s'obtenir à l'aide de la procédure
# "rungekutta" se trouvant dans "share library" :
>
# with(share) : readshare(ODE, plots) :
# makelist(rungekutta(f, [x0,y0], h, n)) ;
>
# Et avec Maple 7 ???
```

4.4 Matrices pour la méthode de Runge-Kutta

On se propose de calculer des matrices liées à la stratégie de Runge-Kutta donnant lieu à des méthodes de «bonne consistance»...

```
> restart : interface(echo=3) : read ‘equadiff4.mpl‘ ;
# En supposant que le problème de Cauchy    y’(x) = f(x,y(x))
# avec y(x_0) = y_0 a une solution unique sur un certain intervalle
# réel contenant x_0, les méthodes de Runge-Kutta utilisent le
# développement de Taylor de y en x_0 pour obtenir des méthodes
# dont l’ordre de consistance est "aussi grand que l’on veut".
>
> ancien_echo := interface(echo, echo=1) :
```

“lecture de equadiff3.mpl”

```
Warning, the protected names norm and trace have been redefined and
unprotected
>
# Essayons de déterminer toutes les matrices de dimension 2 x 3
# donnant lieu à une méthode Runge-Kutta de consistance d’ordre n=3.
>
# Voici l’équation différentielle générique :
> (f, x0, y, h) := (’f’, ’x0’, ’y’, ’h’) :
> D(y) := x -> f(x, y(x)) ;
```

$$\mathrm{D}(y) := x \to \mathrm{f}(x,\, \mathrm{y}(x))$$

```
>
# Commencons par calculer le developpement de Taylor en h=0,
# d’ordre n de l’équation générique.
> n := 3 ; ty := taylor(y(x0+h), h=0, n) ;
```

$$n := 3$$

$$ty := \mathrm{y}(x0) + \mathrm{f}(x0,\, \mathrm{y}(x0))\, h + (\frac{1}{2}\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0)) + \frac{1}{2}\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, \mathrm{f}(x0,\, \mathrm{y}(x0)))\, h^2 + \mathrm{O}(h^3)$$

```
> typ := simplify(convert(ty, polynom)) ;
```

$$typ := \mathrm{y}(x0) + \mathrm{f}(x0,\, \mathrm{y}(x0))\, h + \frac{1}{2}\, h^2\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0)) + \frac{1}{2}\, h^2\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, \mathrm{f}(x0,\, \mathrm{y}(x0))$$

```
>
# Construisons une matrice M (de dimension (d-1).d ), générique
# pour la méthode de Runge-Kutta.
> d := 3 ;
```

$$d := 3$$

```
> M := matrix([seq([seq(m[i,j], j=1..i-1), seq(0, j=i..d-1)],
>              i=1..d)]) ;
```

$$M := \begin{bmatrix} 0 & 0 \\ m_{2,1} & 0 \\ m_{3,1} & m_{3,2} \end{bmatrix}$$

```
>
# Une incantation magique pour obtenir les inconnues
# de la matrice M...
> inconnues := indets(op(eval(M))[3]) ;
```

$$inconnues := \{m_{2,1},\, m_{3,1},\, m_{3,2}\}$$

```
>
# Le développement de Taylor en h=0, d'ordre n de cette méthode est
> z := runge_kutta(f, x0, y(x0), h, M) ;
```

$$z := \mathrm{y}(x0) + h\, (m_{3,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0)) + m_{3,2}\, \mathrm{f}(x0 + h\, m_{2,1},\, \mathrm{y}(x0) + h\, m_{2,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0))))$$

```
> tz := taylor(z, h=0, n) ;
```

$$tz := \mathrm{y}(x0) + (m_{3,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0)) + m_{3,2}\, \mathrm{f}(x0,\, \mathrm{y}(x0)))\, h + m_{3,2}\, (\mathrm{D}_1(f)(x0,\, \mathrm{y}(x0))\, m_{2,1} + \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, m_{2,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0)))\, h^2 + \mathrm{O}(h^3)$$

```
> tzp := simplify(convert(tz, polynom)) ;
```

$$tzp := \mathrm{y}(x0) + h\, m_{3,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0)) + h\, m_{3,2}\, \mathrm{f}(x0,\, \mathrm{y}(x0)) + m_{3,2}\, h^2\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0))\, m_{2,1} + m_{3,2}\, h^2\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, m_{2,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0))$$

```
>
# Le choix des inconnues est fait de sorte que les développements
# de Taylor ci-dessus de la solution exacte et la méthode de
# Runge-Kutta soient égaux.
```

```
> equat := typ - tzp ;
```

$$equat := \mathrm{f}(x0,\, \mathrm{y}(x0))\, h + \frac{1}{2}\, h^2\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0)) + \frac{1}{2}\, h^2\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, \mathrm{f}(x0,\, \mathrm{y}(x0)) - h\, m_{3,\,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0)) - h\, m_{3,\,2}\, \mathrm{f}(x0,\, \mathrm{y}(x0)) - m_{3,\,2}\, h^2\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0))\, m_{2,\,1} - m_{3,\,2}\, h^2\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, m_{2,\,1}\, \mathrm{f}(x0,\, \mathrm{y}(x0))$$

```
>

# Pour résoudre l'équation, on détermine les variables du problème

> variables := indets(equat) ;
```

$$variables := \{\mathrm{f}(x0,\, \mathrm{y}(x0)),\, x0,\, \mathrm{y}(x0),\, m_{2,\,1},\, m_{3,\,1},\, m_{3,\,2},\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0)),\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0)),\, h\}$$

```
>

# On en extrait les parametres...

> parametres := variables minus inconnues ;
```

$$parametres := \{\mathrm{f}(x0,\, \mathrm{y}(x0)),\, x0,\, \mathrm{y}(x0),\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0)),\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0)),\, h\}$$

```
>

# ...et les coefficients devant s'annuler

> coefficients := coeffs(equat, parametres[1]) ;
```

$$coefficients := \frac{1}{2}\, h^2\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0)) - m_{3,\,2}\, h^2\, \mathrm{D}_1(f)(x0,\, \mathrm{y}(x0))\, m_{2,\,1},\quad h + \frac{1}{2}\, h^2\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0)) - h\, m_{3,\,1} - h\, m_{3,\,2} - m_{3,\,2}\, h^2\, \mathrm{D}_2(f)(x0,\, \mathrm{y}(x0))\, m_{2,\,1}$$

```
>

# C'est terminé ! On peut déterminer toutes les matrices solutions.

> s1 := solve({coefficients}, inconnues) ; subs(s1, eval(M)) ;
```

$$s1 := \{m_{3,\,1} = 1 - m_{3,\,2},\, m_{3,\,2} = m_{3,\,2},\, m_{2,\,1} = \frac{1}{2}\, \frac{1}{m_{3,\,2}}\}$$

$$\begin{bmatrix} 0 & 0 \\ \frac{1}{2}\, \frac{1}{m_{3,\,2}} & 0 \\ 1 - m_{3,\,2} & m_{3,\,2} \end{bmatrix}$$

```
>

# Voici une implémentation générale.

> methode := proc(n, M)
```

```
>    # n : l'ordre de consistance de la méthode
>    # M : une matrice plus ou moins générique
>    # y (globale) : solution du problème générique
>    # Cette procédure renvoie la suite des trois ensembles contenant
>    # 1) les coefficients devant s'annuler
>    # 2) les inconnues de M
>    # 3) les paramètres des différents développements limités
>    local h,typ,tzp,z,equat,var,inco,param ;
>    global y,f,x0 ;
>      typ := taylor(y(x0+h), h=0, n) ;
>      typ := simplify(convert(typ, polynom)) ;
>      z := runge_kutta(f, x0, y(x0), h, M) ;
>      tzp := taylor(z, h=0, n) ;
>      tzp := simplify(convert(tzp, polynom)) :
>      equat := typ - tzp ;
>      var := indets(equat) ;
>      inco := indets(op(eval(M))[3]) ;
>      param := var minus inco ;
>      RETURN({coeffs(equat, param)}, inco, param) ;
> end :
>
# Testons... avec des conditions supplémentaires, sinon
# le nombre de solutions exploses !
#(n,d) := (3,3) ;
#(n,d) := (4,4) ; m[2,1] := 1/3 ; m[3,2] := 2/3 ;
> (n,d) := (5,5) ; m[3,1] := 0 ; m[4,1] := 0 ; m[4,2] := 0 ;
```

$$n,\, d := 5,\, 5$$
$$m_{3,1} := 0$$
$$m_{4,1} := 0$$
$$m_{4,2} := 0$$

```
>
> M := matrix([seq([seq(m[i,j], j=1..i-1), seq(0, j=i..d-1)],
>                  i=1..d)]) ;
```

$$M := \begin{bmatrix} 0 & 0 & 0 & 0 \\ m_{2,1} & 0 & 0 & 0 \\ 0 & m_{3,2} & 0 & 0 \\ 0 & 0 & m_{4,3} & 0 \\ m_{5,1} & m_{5,2} & m_{5,3} & m_{5,4} \end{bmatrix}$$

```
> meth := methode(n, M) :
> coefficients := meth[1] ;
```

$$coefficients := \{-\frac{1}{2}\, m_{5,3}\, m_{3,2}\, {m_{2,1}}^2 - \frac{1}{2}\, m_{5,4}\, m_{4,3}\, {m_{3,2}}^2 + \frac{1}{24},$$
$$-m_{5,3}\, {m_{3,2}}^2\, m_{2,1} - m_{5,4}\, {m_{4,3}}^2\, m_{3,2} + \frac{1}{6} - \frac{1}{2}\, m_{5,4}\, m_{4,3}\, {m_{3,2}}^2 - \frac{1}{2}\, m_{5,3}\, m_{3,2}\, {m_{2,1}}^2,$$
$$-m_{5,3}\, m_{3,2}\, {m_{2,1}}^2 - m_{5,4}\, m_{4,3}\, {m_{3,2}}^2 - m_{5,3}\, {m_{3,2}}^2\, m_{2,1} - m_{5,4}\, {m_{4,3}}^2\, m_{3,2} + \frac{5}{24},$$
$$-m_{5,1} + 1 - m_{5,3} - m_{5,4} - m_{5,2},\ -m_{5,3}\, {m_{3,2}}^2 - m_{5,2}\, {m_{2,1}}^2 - m_{5,4}\, {m_{4,3}}^2 + \frac{1}{3},$$
$$-m_{5,3}\, m_{3,2} - m_{5,4}\, m_{4,3} - m_{5,2}\, m_{2,1} + \frac{1}{2},$$
$$\frac{1}{24} - \frac{1}{6}\, m_{5,2}\, {m_{2,1}}^3 - \frac{1}{6}\, m_{5,4}\, {m_{4,3}}^3 - \frac{1}{6}\, m_{5,3}\, {m_{3,2}}^3,$$
$$-\frac{1}{2}\, m_{5,3}\, {m_{3,2}}^2 - \frac{1}{2}\, m_{5,2}\, {m_{2,1}}^2 - \frac{1}{2}\, m_{5,4}\, {m_{4,3}}^2 + \frac{1}{6},$$
$$-\frac{1}{2}\, m_{5,3}\, {m_{3,2}}^3 - \frac{1}{2}\, m_{5,2}\, {m_{2,1}}^3 - \frac{1}{2}\, m_{5,4}\, {m_{4,3}}^3 + \frac{1}{8},$$
$$-m_{5,3}\, {m_{3,2}}^2\, m_{2,1} - m_{5,4}\, {m_{4,3}}^2\, m_{3,2} + \frac{1}{8},\ -m_{5,4}\, m_{4,3}\, m_{3,2}\, m_{2,1} + \frac{1}{24},$$
$$-m_{5,3}\, m_{3,2}\, m_{2,1} + \frac{1}{6} - m_{5,4}\, m_{4,3}\, m_{3,2}\}$$

```
> inconnues := meth[2] :
> sol1 := solve(coefficients, inconnues) :
> for i in [sol1] do
>   subs(i, eval(M)) ;
```

```
> od ;
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{bmatrix}$$
