Opérandes d'un objet Maple

© Pierre Lantagne
Enseignant retraité du Collège de Maisonneuve

Ce document est une révision de celui produit en 2005. L'objectif principal de ce document est d'initier le lecteur à la notion de structure de données d'un objet Maple. C'est parce que toutes structures de données Maple sont des expressions Maple valides, ces expressions et les sous-expressions qui les composent peuvent être manipulées, sauvegardées et imprimées. La connaisance de ces structures est essentielle pour tous ceux et celles qui aspirent à créer des procédures en Maple. Une lecture attentive de ce document permettra donc aux novices d'approfondir leur connaissance afin d'acquérir une plus grande maîtrise du logiciel Maple.

Ce document se termine en présentant une démarche de coloriage de régions comprises entre deux courbes ou bornées par une courbe fermée. Cette manière de faire fait appel à la compréhension que nous devons avoir des structures de données, en particulier des structures graphiques.

Bonne lecture à tous !

* Ce document Maple est exécutable avec la version 2020.2

Initialisation

```
> restart;
> with(plots,display,setoptions):
> setoptions(axesfont=[times,roman,8],size=[300,300]):
```

Structure d'un objet Maple

Considérons l'expressions $3x+e^{\pi}-\sin\left(\frac{\pi}{4}\right)+\frac{1}{x}-5$. Nommons cette expression `Mon_expression`.

```
> Mon_expression:=3*x+exp(Pi)-sin(Pi/4)+1/x-5;
```

$$Mon\_expression := 3x+e^{\pi}-\frac{\sqrt{2}}{2}+\frac{1}{x}-5 \qquad \textbf{(2.1)}$$

Remarquons d'abord l'intervention du mécanisme de la simplification automatique qui a simplifié $\sin\left(\frac{\pi}{4}\right)$ par $\frac{1}{2}\sqrt{2}$ avant même de mémoriser cette expression. Vous retrouverez plus de détails concernant ce mécanisme dans les documents *Premiers pas en Maple* dans la section *Compléments Maple* de mon site Internet.

La macro-commande `nops` permet l'affichage du nombre d'opérandes composant cette expression.

```
> nops(Mon_expression);
```

$$5 \qquad \textbf{(2.2)}$$

Le nombre 5 correspond au nombre de termes composant la somme `Mon_expression`. Au niveau de `Mon_expression`, il y a effectivement 5 opérandes correspondant à ses 5 sous-expressions.

En effet, voici comment Maple reconnaît `Mon_expression`. La macro-commande `whattype` permet l'affichage du type d'objet Maple.

```
> whattype(Mon_expression);
```

$$`+` \tag{2.3}$$

La macro-commande `op` a comme résultat la séquence des opérandes au niveau de son type.

```
> op(Mon_expression);
```

$$3x, e^{\pi}, -\frac{\sqrt{2}}{2}, \frac{1}{x}, -5 \tag{2.4}$$

Voici une autre façon de faire afficher le type d'objet Maple qu'est `Mon_expression`.

```
> op(0,Mon_expression);
```

$$`+` \tag{2.5}$$

Nous pouvons pointer vers chacun des éléments de la séquence de ses arguments.

```
> op(1,Mon_expression);
```

$$3x \tag{2.6}$$

```
> op(2,Mon_expression);
```

$$e^{\pi} \tag{2.7}$$

```
> op(3,Mon_expression);
```

$$-\frac{\sqrt{2}}{2} \tag{2.8}$$

```
> op(4,Mon_expression);
```

$$\frac{1}{x} \tag{2.9}$$

```
> op(5,Mon_expression);
```

$$-5 \tag{2.10}$$

Comme on le constate, chacun des résultats précédents est également un objet Maple et chacun d'eux, à leur niveau, est composé d'opérandes.

La procédure `allops` retourne la séquence de tous les opérandes et sous-opérandes d'un objet Maple.

```
> allops:=proc(expr);
  if type(expr,name) or type(expr,integer) then expr
   else expr,op(map(allops,[op(expr)]))
  fi
  end:
```

Comme exemple, obtenons la séquence des opérandes et des sous-opérandes de
$3x + e^{\pi} - \sin\left(\frac{\pi}{4}\right) + \frac{1}{x} - 5.$

```
> allops(Mon_expression);
```

$$3x + e^{\pi} - \frac{\sqrt{2}}{2} + \frac{1}{x} - 5, 3x, 3, x, e^{\pi}, \pi, -\frac{\sqrt{2}}{2}, -\frac{1}{2}, -1, 2, \sqrt{2}, 2, \frac{1}{2}, 1, 2, \frac{1}{x}, x, -1, -5 \tag{2.11}$$

Pour avoir la maîtrise complète d'un objet Maple, la connaissance de l'ensemble des opérandes et sous-

opérandes d'un objet Maple n'est pas suffisante, il faut aussi prendre connaissance de la manière dont Maple structure tous ces opérandes. La procédure suivante retourne l'arborescence (la structure) d'un objet Maple.

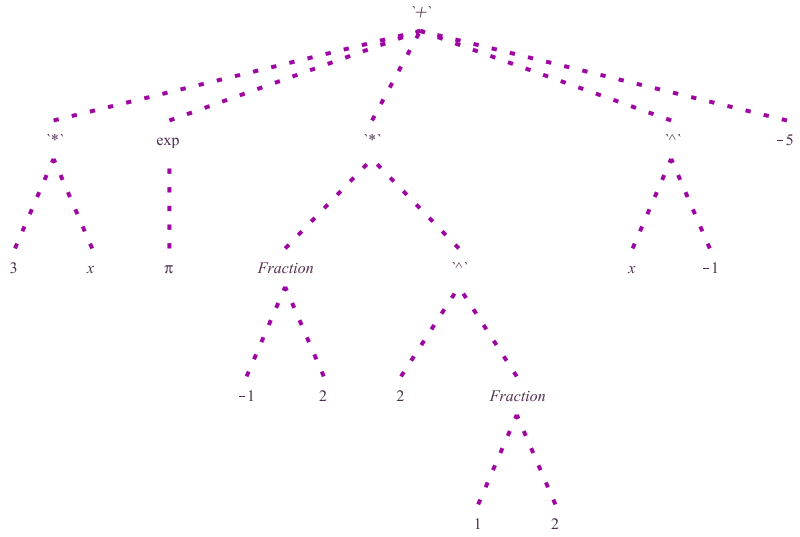
```
> arbre:=proc(expr)
   local k,Options_base,Opts,plot1,plot2,Size;
   global affx,affy,affT,affS,marque;
   Opts:=[args[2..nargs]];
   Options_base:=NULL;
    for k from 1 to nops(Opts) do
     if lhs(map(op,k,Opts))=size then Options_base:=size=rhs(Opts[k])
  fi;
    od;
    marque:=proc(expr)
     local xa,xb,pos,lpos;
     global affx,affy,affT,affS;
      if type(expr,symbol) or type(expr,integer) then
       affT:=affT,[affx,affy,expr];
       xa:=affx;
       affx:=affx+1;
       xa
      else
       affy:=affy-1;
       lpos:=map(marque,[op(expr)]);
       xa:=op(1,lpos);
       xb:=op(nops(lpos),lpos);
       affy:=affy+1;
       pos:=(xa+xb)/2;
       affT:=affT,[pos,affy,op(0,expr)];
       affS:=affS,op(map(proc(a,b) CURVES([[a,affy-0.85],[b,affy-.15]],
  \
              COLOR(RGB,0.6156862700, 0.0156862750, 0.6313725500))end,
  lpos,pos));
        pos;
       fi
      end proc;
   affx:=0; affy:=0;affT:=NULL; affS:=NULL;
   marque(expr);
   plot1:=plots[textplot]({affT},font=[TIMES,ROMAN,10],color="MapleV
  21");
   plot2:=PLOT(affS,THICKNESS(1),LINESTYLE(2));
   unassign('affx,affy,affT,affS,marque');
   plots[display]({plot1,plot2},axes=NONE,Options_base);
  end proc:
```

Remarque: Les macro-commandes `allops` et `arbre` n'appartiennent à aucune bibliothèque Maple. Ce sont deux exemples de procédures qui pourront bien sûr, enrichir votre bibliothèque personnelle. Ces deux procédures ont été obtenues autrefois (bien avant 2005) avec une recherche personnelle sur Internet. Je n'en suis pas l'auteur mais je les ai légèrement modifiées. Je n'ai pu remercier l'auteur jusqu'à présent car la source

de ces procédures n'était pas identifiée au moment de la première version de ce document. En fait, il y a deux auteurs: Philippe Fortin et Roland Pomès. Ces procédures apparaissent dans leur ouvrage « *Premiers pas en Maple: Introduction à l'utilisation du calcul formel en mathématiques, physique et chimie* ».

Dessinons maintenant l'arbre des opérandes composant l'expression $3x + e^{\pi} - \sin\left(\frac{\pi}{4}\right) + \frac{1}{x} - 5$.

```
> arbre(Mon_expression,size=[600,400]);
```



L'arborescence précédente montre l'ordre avec lequel Maple a mémorisé l'expression $3x + e^{\pi} - \sin\left(\frac{\pi}{4}\right) + \frac{1}{x} - 5$, où il y a eu simplification automatique de $\sin\left(\frac{\pi}{4}\right)$. Comme nous l'avons vu précédemment, la macro-commande `op` permets de pointer vers les opérandes d'un objet Maple. En fait, la macro-commande `op` permet de pointer vers n'importe quel opérande de n'importe quel niveau d'un objet Maple. Il faut par contre connaître l'arboresence (la mémorisation de l'objet) afin de pointer correctement vers un sous-opérande particulier de l'objet. On note également, que l'élaboration de la structure d'un objet Maple passe par la validation par Maple du type de chacun des objets le composant.

Avec une lecture de la gauche vers la droite de cette arbre, le troisième opérande de `Mon_expression` est une expression de type `*`.

```
> whattype(op(3,Mon_expression));
```

$$\text{`*`} \tag{2.12}$$

Nous aurions pu aussi saisir la requête suivante.

```
> op(0,op(3,Mon_expression));
```

$$\text{`*`} \tag{2.13}$$

Ce troisième opérande est lui-même un objet possédant deux opérandes.

```
> op(1,op(3,Mon_expression));
```

$$-\frac{1}{2} \tag{2.14}$$

L'arbre nous montre très bien que $-\frac{1}{2}$ est mémorisé comme fraction $\frac{-1}{2}$.

```
> op(2,op(3,Mon_expression));
```

$$\sqrt{2} \tag{2.15}$$

Tandis que $\sqrt{2}$ est mémorisé comme suit: $2^{\frac{1}{2}}$. C'est donc un objet du type `^` possédant deux opérandes: la base 2 et comme exposant la fraction $\frac{1}{2}$.

```
> op(0,op(2,op(3,Mon_expression)));
```

$$\text{`^`} \tag{2.16}$$

```
> op(1,op(2,op(3,Mon_expression)));
  op(2,op(2,op(3,Mon_expression)));
```

$$2$$

$$\frac{1}{2} \tag{2.17}$$

Prenons un autre exemple. Une sacrée expression que celle-ci:

$$(2x^2-3)\ln\left(12+\sqrt{\frac{\sin\left(x+\frac{\pi}{2}\right)}{\pi-\sqrt{2}\tan(x)+\sec(2x)-\sin\left(\frac{\sqrt{3}\,x}{3}\right)}}\right).$$

Saisissons cette expression.

```
> Mon_expression:=(2*x^2-3)*ln(12+sqrt(sin(x+Pi/2)/(Pi-sqrt(2)*tan(x)+
  sec(2*x)-sin(sqrt(3)*x/3))));
```

$$Mon\_expression := (2x^2-3)\ln\left(12+\sqrt{\frac{\cos(x)}{\pi-\sqrt{2}\tan(x)+\sec(2x)-\sin\left(\frac{\sqrt{3}\,x}{3}\right)}}\right) \tag{2.18}$$

Encore une fois, notons l'intervention du mécanisme de la simplification automatique qui a simplié $\sin\left(x+\frac{\pi}{2}\right)$ par $\cos(x)$ au moment de la mémorisation de l'objet.

Supposons qu'il faille extraire le radicand. La connaissance des types d'objets Maple est nécessaire pour pointer vers le radicand. Bien qu'on peut pointer vers le radicand en une unique requête, allons-y plutôt progressivement à ce stade-ci.

`Mon_expression` est du type `*` possédant deux opérandes. Le radicand est un objet du second opérande.

```
> op(2,Mon_expression);
```

$$\ln\left(12+\sqrt{\frac{\cos(x)}{\pi-\sqrt{2}\tan(x)+\sec(2x)-\sin\left(\frac{\sqrt{3}\,x}{3}\right)}}\right) \tag{2.19}$$

Ce résultat est une expression de type `ln` possédant un seul opérande.

```
> op(1,op(2,Mon_expression));
```

$$12+\sqrt{\frac{\cos(x)}{\pi-\sqrt{2}\tan(x)+\sec(2x)-\sin\left(\frac{\sqrt{3}\,x}{3}\right)}} \tag{2.20}$$

Ce résultat est une expression de type `+` possédant deux opérandes. Le radical est dans le second.

```
> op(2,op(1,op(2,Mon_expression)));
```

$$\sqrt{\frac{\cos(x)}{\pi-\sqrt{2}\,\tan(x)+\sec(2x)-\sin\left(\frac{\sqrt{3}\,x}{3}\right)}} \tag{2.21}$$

Ce résultat est une expression du type `^` possédant deux opérandes. Le radicand (base) est le premier opérande de ce type.

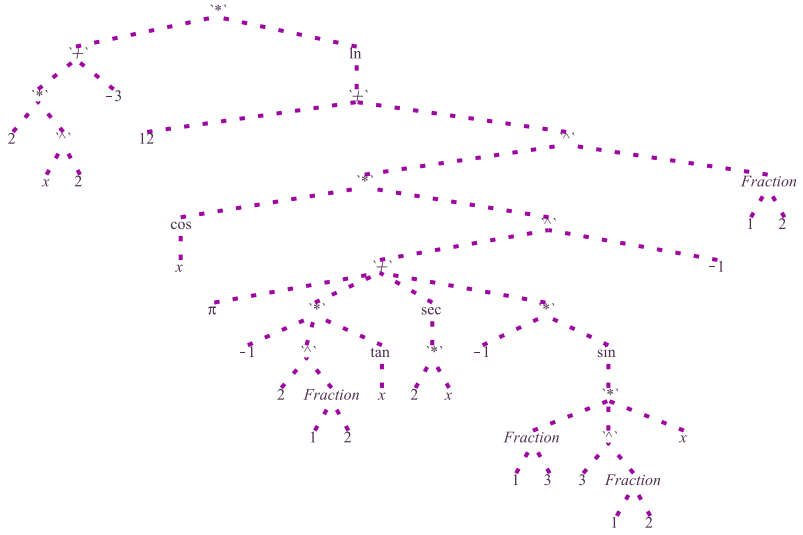
```
> op(1,op(2,op(1,op(2,Mon_expression))));
```

$$\frac{\cos(x)}{\pi-\sqrt{2}\,\tan(x)+\sec(2x)-\sin\left(\frac{\sqrt{3}\,x}{3}\right)} \tag{2.22}$$

Nous y sommes: nous avons extrait le radicand demandé. Nous avons pu extraire le radicand sur la base de l'affichage de `Mon_expression` (montrant sa mémorisation par Maple) ainsi que par la connaissance des types d'objets Maple.

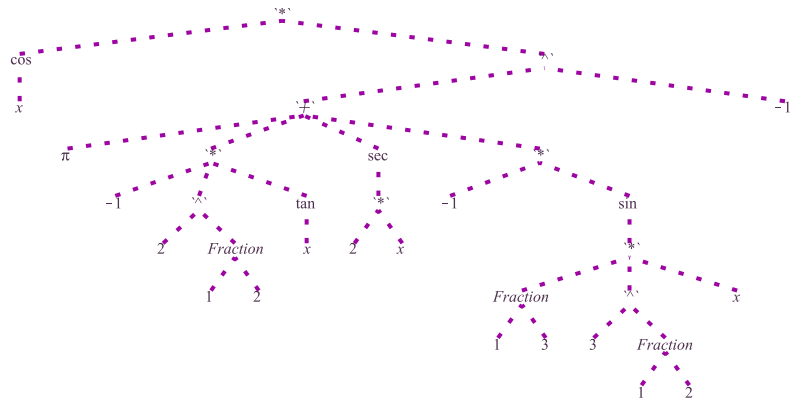
Pour la beauté de la chose, affichons l'arborescence de `Mon_expression`.

```
> display(arbre(Mon_expression),size=[600,400]);
```



Reste à afficher l'arborescence du radicand qui a été extrait.

```
> display(arbre(op(1,op(2,op(1,op(2,Mon_expression))))),size=[600,300])
  ;
```

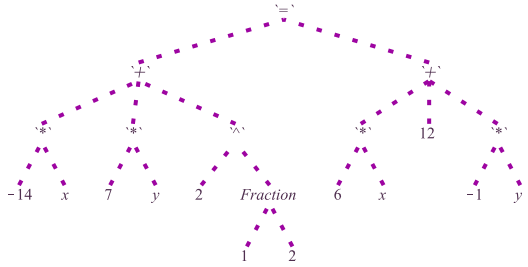


Remarque 1: Pour pointer vers les membres de gauche et de droite d'une équation, utilisons plutôt les macro-commandes **lhs** et **rhs** respectivement.

```
> Mon_Équation:=-7*(2*x-y)+sqrt(2)=6*(x+2)-y;
```

$$Mon\_Équation := -14x + 7y + \sqrt{2} = 6x + 12 - y \tag{2.23}$$

```
> display(arbre(Mon_Équation),size=[400,200]);
```



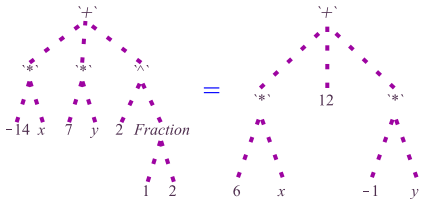

```
> Membre_de_gauche:=lhs(Mon_Équation);
```

$$Membre\_de\_gauche := -14x + 7y + \sqrt{2} \tag{2.24}$$

```
> Membre_de_droite:=rhs(Mon_Équation);
```

$$Membre\_de\_droite := 6x + 12 - y \tag{2.25}$$

```
> arbre(Membre_de_gauche)=arbre(Membre_de_droite);
```

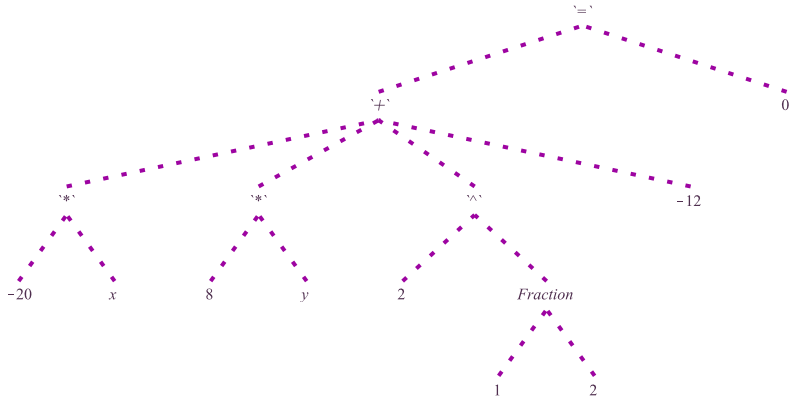


Pour soustraire le membre de droite à Mon_Équation, il suffit de soustraire à Mon_Équation, membre à membre, son membre de droite.

```
> Mon_Équation-(rhs(Mon_Équation)=rhs(Mon_Équation));
```

$$-20x + 8y + \sqrt{2} - 12 = 0 \tag{2.26}$$

```
> arbre((2.26),size=[600,300]);
```

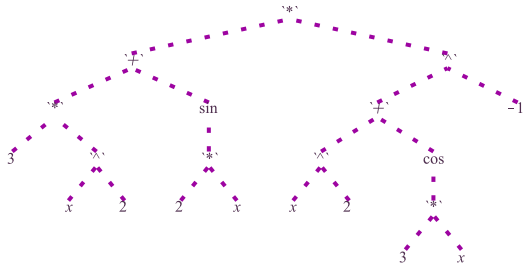


Remarque 2: Pour pointer vers le numérateur et le dénominateur d'un objet de type rationnel, utilisons plutôt les macro-commandes **numer** et **denom** respectivement.

```
> Formule:=(3*x^2+sin(2*x))/(x^2+cos(3*x));
```

$$Formule := \frac{3x^2 + \sin(2x)}{x^2 + \cos(3x)} \tag{2.27}$$

```
> arbre(Formule,size=[400,200]);
```




```
> numer(Formule);
  denom(Formule);
```

$$3x^2 + \sin(2x)$$

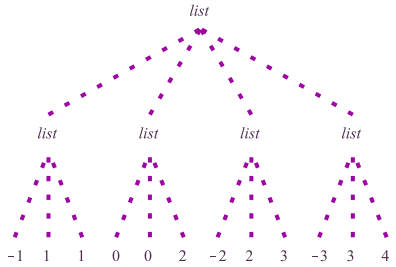
$$x^2 + \cos(3x) \qquad \textbf{(2.28)}$$

Remarque 3: Pour pointer vers certains éléments d'un objet Maple de type liste, en plus de la macro-commande `op` bien sûr, nous pouvons utiliser la **notation indicielle** pour référencer ces éléments.

```
> Liste_de_points:=[[-1,1,1],[0,0,2],[-2,2,3],[-3,3,4]];
```

$$Liste\_de\_points := [[-1, 1, 1], [0, 0, 2], [-2, 2, 3], [-3, 3, 4]] \qquad \textbf{(2.29)}$$

```
> arbre(Liste_de_points,size=[300,200]);
```




```
> op(1,op(3,Liste_de_points));
```

$$-2 \qquad \textbf{(2.30)}$$

```
> Liste_de_points[3][1];
```

$$-2 \qquad \textbf{(2.31)}$$

```
> abscisses:=seq(Liste_de_points[k][1],k=1..nops(Liste_de_points));
  ordonnées:=seq(Liste_de_points[k][2],k=1..nops(Liste_de_points));
  cotes:=seq(Liste_de_points[k][3],k=1..nops(Liste_de_points));
```

$$abscisses := -1, 0, -2, -3$$

$$ordonnées := 1, 0, 2, 3$$

$$cotes := 1, 2, 3, 4 \qquad \textbf{(2.32)}$$

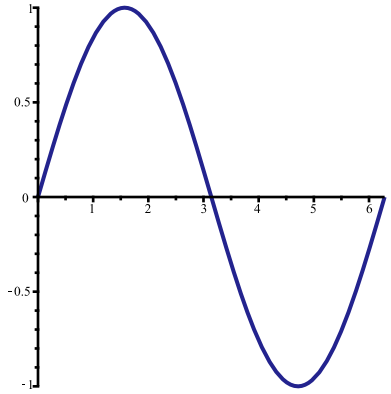
Opérateur op et structure plot

La macro-commande `plot` est utilisée pour tracer un graphique. Par curiosité, appliquons les macro-commandes `allops` et `arbre` sur un objet graphique obtenu avec la macro-commande `plot`.

Essayons avec le graphique du sinus.

```
> Mon_Graphe:=plot([x,sin(x),x=0..2*Pi],color=navy,numpoints=25):
```

```
> Mon_Graphe;
```




```
> allops(Mon_Graphe);
Error, (in allops) too many levels of recursion
> arbre(Mon_Graphe);
Error, (in marque) improper op or subscript selector
```

Dans un cas comme dans l'autre, il y a eu plantage. La macro-commande `plot` a crée une structure graphique `PLOT` dans laquelle il y a une autre structure de données qui est la structure `CURVES`. C'est pourquoi `Allops` a planté. Le plantage de la procédure `arbre` est dû à la manière particulière de pointer vers les éléments d'un objet de type Matrix.

`Mon_Graphe`, en tant qu'objet Maple, a été mémorisé et possède donc des opérandes. En utlisant `lprint`, affichons les opérandes de `Mon_Graphe`.

```
> lprint(Mon_Graphe):
PLOT(CURVES(Matrix(57,2,{(2, 1) = HFloat(.136955558816210993), (2, 2) = HFloat(
.136527818255766431), (3, 1) = HFloat(.273911117632421985), (3, 2) = HFloat(.27\
0498808353584586), (4, 1) = HFloat(.393075417061013455), (4, 2) = HFloat(.38303\
1093796059374), (5, 1) = HFloat(.512239716489604979), (5, 2) = HFloat(.49013072\
1085096033), (6, 1) = HFloat(.646252434202074344), (6, 2) = HFloat(.60219878661\
4086101), (7, 1) = HFloat(.780265151914543709), (7, 2) = HFloat(.70346789456166\
1700), (8, 1) = HFloat(.915166192910842780), (8, 2) = HFloat(.79266391879980113\
7), (9, 1) = HFloat(1.05006723390714196), (9, 2) = HFloat(.867456677279072941),
(10, 1) = HFloat(1.18432711380367084), (10, 2) = HFloat(.926245656221753366), (
11, 1) = HFloat(1.31858699370019972), (11, 2) = HFloat(.968363459696584772), (
12, 1) = HFloat(1.38082504126269789), (12, 2) = HFloat(.982009657695086968), (
13, 1) = HFloat(1.44306308882519585), (13, 2) = HFloat(.991853195789886399), (
14, 1) = HFloat(1.50530113638769381), (14, 2) = HFloat(.997855956608785899), (
15, 1) = HFloat(1.56753918395019198), (15, 2) = HFloat(.999994695514934140), (
16, 1) = HFloat(1.63198307723275127), (16, 2) = HFloat(.998128674719903408), (
17, 1) = HFloat(1.69642697051531055), (17, 2) = HFloat(.992118844592631333), (
18, 1) = HFloat(1.76087086379786983), (18, 2) = HFloat(.981990155413374644), (
19, 1) = HFloat(1.82531475708042912), (19, 2) = HFloat(.967784657228192069), (
20, 1) = HFloat(1.95861069032195889), (20, 2) = HFloat(.925737803714865248), (
21, 1) = HFloat(2.09190662356348867), (21, 2) = HFloat(.867266960473713433), (
22, 1) = HFloat(2.22477507505660066), (22, 2) = HFloat(.793669619373010948), (
23, 1) = HFloat(2.35764352654971265), (23, 2) = HFloat(.706081415752460306), (
24, 1) = HFloat(2.49431506051906027), (24, 2) = HFloat(.603016901695720975), (
25, 1) = HFloat(2.63098659448840788), (25, 2) = HFloat(.488706091946957855), (
26, 1) = HFloat(2.75136695600047609), (26, 2) = HFloat(.380397155183251345), (
27, 1) = HFloat(2.87174731751254431), (27, 2) = HFloat(.266582372911728260), (
28, 1) = HFloat(3.00726709256405478), (28, 2) = HFloat(.133921977970194284), (
29, 1) = HFloat(3.14278686761556525), (29, 2) = HFloat(-.119421374191787844e-2)
, (30, 1) = HFloat(3.27886309988390945), (30, 2) = HFloat(-.136839750428278562)
, (31, 1) = HFloat(3.41493933215225320), (31, 2) = HFloat(-.269955368336901758)
```

```
, (32, 1) = HFloat(3.54607399625548858), (32, 2) = HFloat(-.393542008196293924)
, (33, 1) = HFloat(3.67720866035872351), (33, 2) = HFloat(-.510370873960668825)
, (34, 1) = HFloat(3.79629149828979662), (34, 2) = HFloat(-.608920385122624919)
, (35, 1) = HFloat(3.91537433622086972), (35, 2) = HFloat(-.698845164714163358)
, (36, 1) = HFloat(4.05697534845959673), (36, 2) = HFloat(-.792795892556510728)
, (37, 1) = HFloat(4.19857636069832285), (37, 2) = HFloat(-.870876934843565098)
, (38, 1) = HFloat(4.31853099189987866), (38, 2) = HFloat(-.923435417768834976)
, (39, 1) = HFloat(4.43848562310143357), (39, 2) = HFloat(-.962722408820853581)
, (40, 1) = HFloat(4.50825349949639964), (40, 2) = HFloat(-.979236606326052694)
, (41, 1) = HFloat(4.57802137589136571), (41, 2) = HFloat(-.990986247364154416)
, (42, 1) = HFloat(4.64778925228633177), (42, 2) = HFloat(-.997914163087678663)
, (43, 1) = HFloat(4.71755712868129784), (43, 2) = HFloat(-.999986645151317521)
, (44, 1) = HFloat(4.77931352100255236), (44, 2) = HFloat(-.997761388660626025)
, (45, 1) = HFloat(4.84106991332380687), (45, 2) = HFloat(-.991732027164756391)
, (46, 1) = HFloat(4.90282630564506139), (46, 2) = HFloat(-.981921548448669190)
, (47, 1) = HFloat(4.96458269796631591), (47, 2) = HFloat(-.968367356336175655)
, (48, 1) = HFloat(5.10009579844243355), (48, 2) = HFloat(-.925778468400502486)
, (49, 1) = HFloat(5.23560889891855208), (49, 2) = HFloat(-.866214770160654624)
, (50, 1) = HFloat(5.36464941662853878), (50, 2) = HFloat(-.794713780605332709)
, (51, 1) = HFloat(5.49368993433852548), (51, 2) = HFloat(-.709998002525491612)
, (52, 1) = HFloat(5.62833017732964969), (52, 2) = HFloat(-.609044347965142951)
, (53, 1) = HFloat(5.76297042032077389), (53, 2) = HFloat(-.497066609307781238)
, (54, 1) = HFloat(5.88661219352166842), (54, 2) = HFloat(-.386259690571258585)
, (55, 1) = HFloat(6.01025396672256296), (55, 2) = HFloat(-.269555427284321691)
, (56, 1) = HFloat(6.14671963736128113), (56, 2) = HFloat(-.136042500098811581)
, (57, 1) = HFloat(6.28318530800000019), (57, 2) = HFloat(.820413714061324611e-\
9)},datatype = float[8],storage = rectangular,order = Fortran_order,shape = [])
),COLOUR(RGB,.13725490,.13725490,.55686274),AXESLABELS("","),AXESTICKS(DEFAULT
,DEFAULT,FONT(times,roman,8)),ROOT(BOUNDS_X(0),BOUNDS_Y(0),BOUNDS_WIDTH(300),
BOUNDS_HEIGHT(300)),VIEW(DEFAULT,DEFAULT))
```

Cette structure de données est clairement compris par la fonctionalité `prettyprinter` de Maple car il y a eu affichage du graphique.

Notons que, par défaut, le nombre minimal de points calculés est 200. L'option `numpoints` peut spécifier un autre nombre minimal mais dans son mécanisme, la macro-commande `plot` génère habituellement un nombre de points supérieur. Bref, l'usager n'a pas le contrôle quant au nombre exact de points qui sera généré.

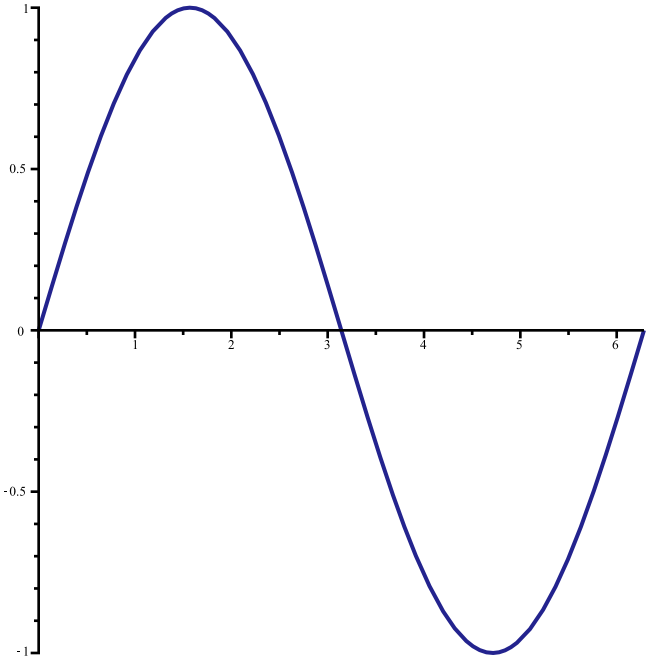
Avec un copier-coller du précédent résultat tel quel, donnons le nom Mon_Graphe_Nouveau à cette structure.

```
> Mon_Graphe_Nouveau:=PLOT(CURVES(Matrix(57,2,{(2, 1) = HFloat
  (.136955558816210993), (2, 2) = HFloat(
  .136527818255766431), (3, 1) = HFloat(.273911117632421985), (3, 2) =
  HFloat(.27\
  0498808353584586), (4, 1) = HFloat(.393075417061013455), (4, 2) =
  HFloat(.38303\
  1093796059374), (5, 1) = HFloat(.512239716489604979), (5, 2) = HFloat
  (.49013072\
  1085096033), (6, 1) = HFloat(.646252434202074344), (6, 2) = HFloat
  (.60219878661\
  4086101), (7, 1) = HFloat(.780265151914543709), (7, 2) = HFloat
  (.70346789456166\
  1700), (8, 1) = HFloat(.915166192910842780), (8, 2) = HFloat
  (.79266391879980113\
```

```
7), (9, 1) = HFloat(1.05006723390714196), (9, 2) = HFloat
(.867456677279072941),
(10, 1) = HFloat(1.18432711380367084), (10, 2) = HFloat
(.926245656221753366), (
11, 1) = HFloat(1.31858699370019972), (11, 2) = HFloat
(.968363459696584772), (
12, 1) = HFloat(1.38082504126269789), (12, 2) = HFloat
(.982009657695086968), (
13, 1) = HFloat(1.44306308882519585), (13, 2) = HFloat
(.991853195789886399), (
14, 1) = HFloat(1.50530113638769381), (14, 2) = HFloat
(.997855956608785899), (
15, 1) = HFloat(1.56753918395019198), (15, 2) = HFloat
(.999994695514934140), (
16, 1) = HFloat(1.63198307723275127), (16, 2) = HFloat
(.998128674719903408), (
17, 1) = HFloat(1.69642697051531055), (17, 2) = HFloat
(.992118844592631333), (
18, 1) = HFloat(1.76087086379786983), (18, 2) = HFloat
(.981990155413374644), (
19, 1) = HFloat(1.82531475708042912), (19, 2) = HFloat
(.967784657228192069), (
20, 1) = HFloat(1.95861069032195889), (20, 2) = HFloat
(.925737803714865248), (
21, 1) = HFloat(2.09190662356348867), (21, 2) = HFloat
(.867266960473713433), (
22, 1) = HFloat(2.22477507505660066), (22, 2) = HFloat
(.793669619373010948), (
23, 1) = HFloat(2.35764352654971265), (23, 2) = HFloat
(.706081415752460306), (
24, 1) = HFloat(2.49431506051906027), (24, 2) = HFloat
(.603016901695720975), (
25, 1) = HFloat(2.63098659448840788), (25, 2) = HFloat
(.488706091946957855), (
26, 1) = HFloat(2.75136695600047609), (26, 2) = HFloat
(.380397155183251345), (
27, 1) = HFloat(2.87174731751254431), (27, 2) = HFloat
(.266582372911728260), (
28, 1) = HFloat(3.00726709256405478), (28, 2) = HFloat
(.133921977970194284), (
29, 1) = HFloat(3.14278686761556525), (29, 2) = HFloat
(-.119421374191787844e-2)
, (30, 1) = HFloat(3.27886309988390945), (30, 2) = HFloat
(-.136839750428278562)
, (31, 1) = HFloat(3.41493933215225320), (31, 2) = HFloat
```

```
(-.269955368336901758)
, (32, 1) = HFloat(3.54607399625548858), (32, 2) = HFloat
(-.393542008196293924)
, (33, 1) = HFloat(3.67720866035872351), (33, 2) = HFloat
(-.510370873960668825)
, (34, 1) = HFloat(3.79629149828979662), (34, 2) = HFloat
(-.608920385122624919)
, (35, 1) = HFloat(3.91537433622086972), (35, 2) = HFloat
(-.698845164714163358)
, (36, 1) = HFloat(4.05697534845959673), (36, 2) = HFloat
(-.792795892556510728)
, (37, 1) = HFloat(4.19857636069832285), (37, 2) = HFloat
(-.870876934843565098)
, (38, 1) = HFloat(4.31853099189987866), (38, 2) = HFloat
(-.923435417768834976)
, (39, 1) = HFloat(4.43848562310143357), (39, 2) = HFloat
(-.962722408820853581)
, (40, 1) = HFloat(4.50825349949639964), (40, 2) = HFloat
(-.979236606326052694)
, (41, 1) = HFloat(4.57802137589136571), (41, 2) = HFloat
(-.990986247364154416)
, (42, 1) = HFloat(4.64778925228633177), (42, 2) = HFloat
(-.997914163087678663)
, (43, 1) = HFloat(4.71755712868129784), (43, 2) = HFloat
(-.999986645151317521)
, (44, 1) = HFloat(4.77931352100255236), (44, 2) = HFloat
(-.997761388660626025)
, (45, 1) = HFloat(4.84106991332380687), (45, 2) = HFloat
(-.991732027164756391)
, (46, 1) = HFloat(4.90282630564506139), (46, 2) = HFloat
(-.981921548448669190)
, (47, 1) = HFloat(4.96458269796631591), (47, 2) = HFloat
(-.968367356336175655)
, (48, 1) = HFloat(5.10009579844243355), (48, 2) = HFloat
(-.925778468400502486)
, (49, 1) = HFloat(5.23560889891855208), (49, 2) = HFloat
(-.866214770160654624)
, (50, 1) = HFloat(5.36464941662853878), (50, 2) = HFloat
(-.794713780605332709)
, (51, 1) = HFloat(5.49368993433852548), (51, 2) = HFloat
(-.709998002525491612)
, (52, 1) = HFloat(5.62833017732964969), (52, 2) = HFloat
(-.609044347965142951)
, (53, 1) = HFloat(5.76297042032077389), (53, 2) = HFloat
(-.497066609307781238)
```

```
, (54, 1) = HFloat(5.88661219352166842), (54, 2) = HFloat
(-.386259690571258585)
, (55, 1) = HFloat(6.01025396672256296), (55, 2) = HFloat
(-.269555427284321691)
, (56, 1) = HFloat(6.14671963736128113), (56, 2) = HFloat
(-.136042500098811581)
, (57, 1) = HFloat(6.28318530800000019), (57, 2) = HFloat
(.820413714061324611e-\
9)},datatype = float[8],storage = rectangular,order = Fortran_order,
shape = [])
),COLOUR(RGB,.13725490,.13725490,.55686274),AXESLABELS("","),
AXESTICKS(DEFAULT
,DEFAULT,FONT(times,roman,8)),VIEW(DEFAULT,DEFAULT));
```



Parce que cette structure de données est une expression Maple valide, elle peut être manipulée, sauvegardée et imprimée comme toute autre expression.

Observons ce que produit la macro-commande `arbre` sur ce nouvel objet.

```
> arbre(Mon_Graphe_Nouveau);
Error, (in marque) improper op or subscript selector
```

Ce plantage est dû à la façon de stocker les points dans la matrice: la procédure `marque` (sous procédure de `arbre`) ne peut accéder avec `op` aux éléments de la matrice de points. Sauvegardons ces points dans un tableau (Array) plutôt que dans une matrice (Matrix) afin que l'indexation utilisée dans la procédure `marque` puisse se faire correctement.

Convertissons la matrice de points en un tableau de points, ce qui permettra une indexation accesible par la procédure `marque`.

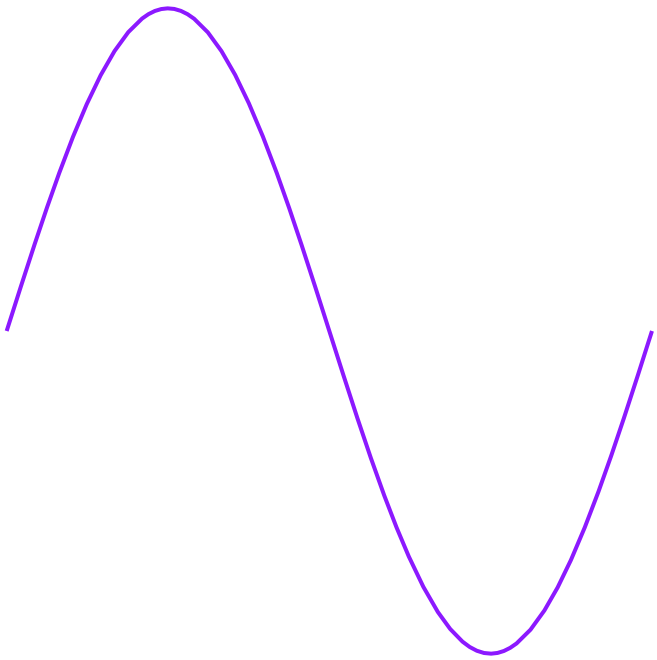
```
> Points:=convert(op(1,op(1,Mon_Graphe_Nouveau)),Array);
```

$$Points := \begin{bmatrix} 0. & 0. \\ 0.1369555588 & 0.1365278183 \\ 0.2739111176 & 0.2704988084 \\ 0.3930754171 & 0.3830310938 \\ 0.5122397165 & 0.4901307211 \\ 0.6462524342 & 0.6021987866 \\ 0.7802651519 & 0.7034678946 \\ 0.9151661929 & 0.7926639188 \\ 1.0500672339 & 0.8674566773 \\ 1.1843271138 & 0.9262456562 \\ \vdots & \vdots \end{bmatrix} \quad \text{(3.1)}$$

57 × 2 Array

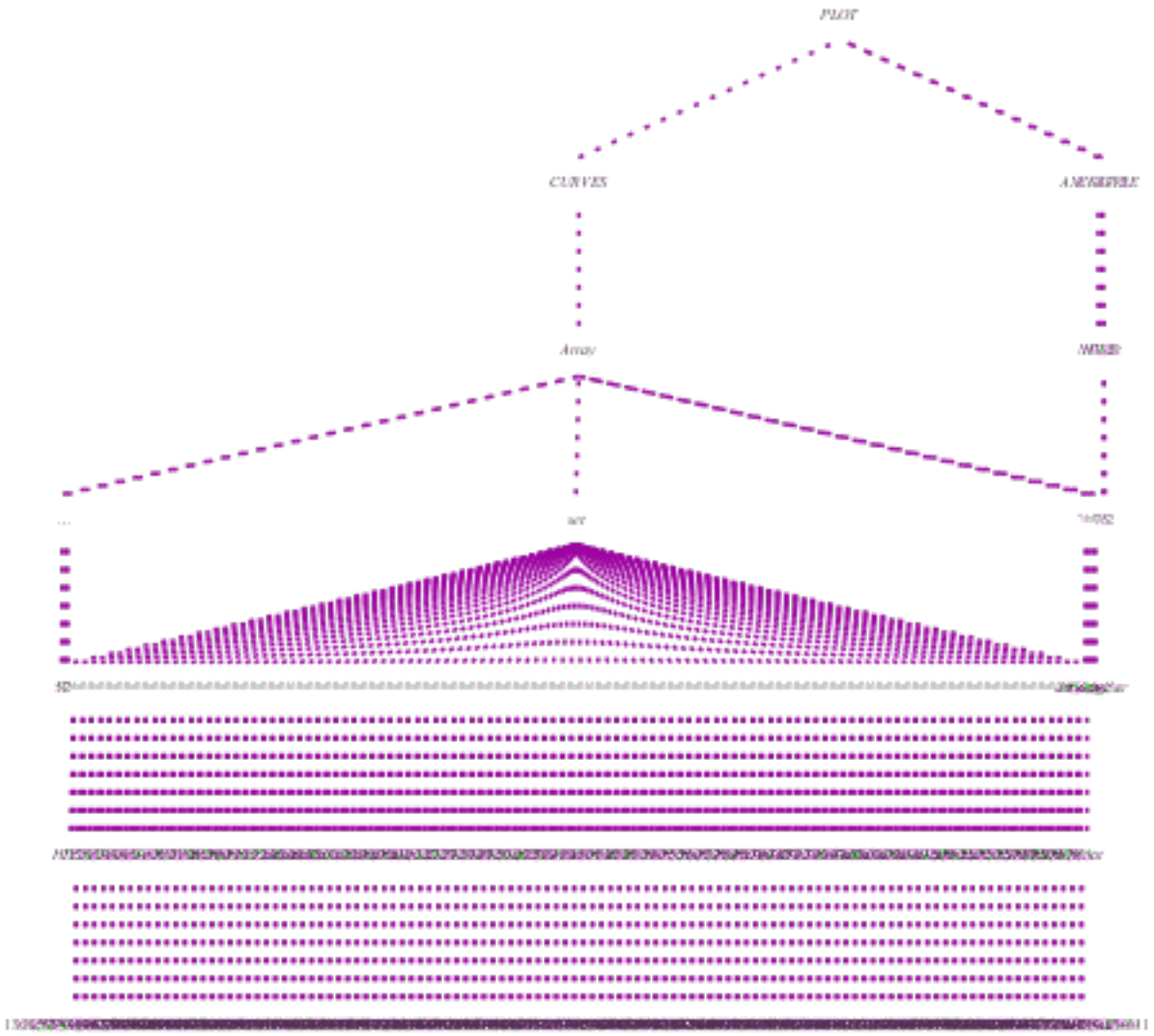
Modifions la stucture `Nouveau_Mon_Graphe` comme suit:

```
> Nouveau_Mon_Graphe:=PLOT(CURVES(Array(1 .. 57, 1 .. 2, Points,
  'datatype' = 'float[8]', 'order' = 'C_order')),AXESSTYLE(NONE),COLOR
  (HUE, .75));
```



Appliquons encore la macro-commande `arbre` mais sur cette structure modifiée.

```
> display(arbre(Nouveau_Mon_Graphe),size=[1200,800]);
```

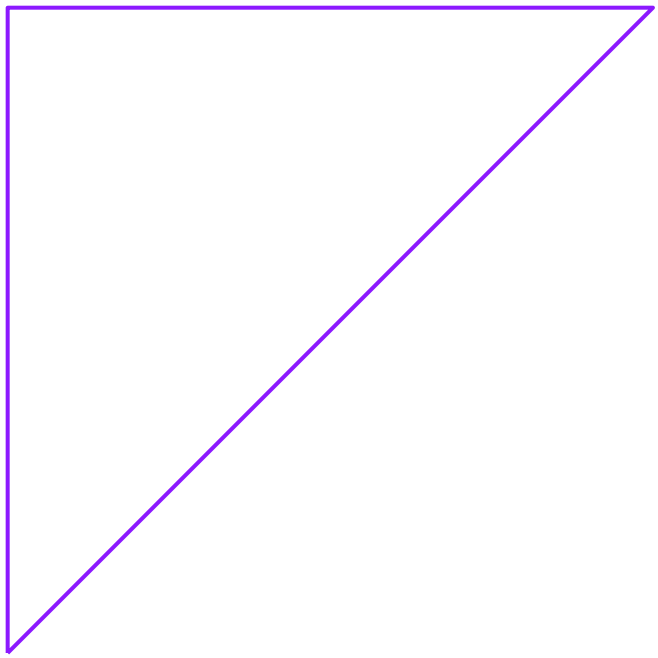


Voilà.

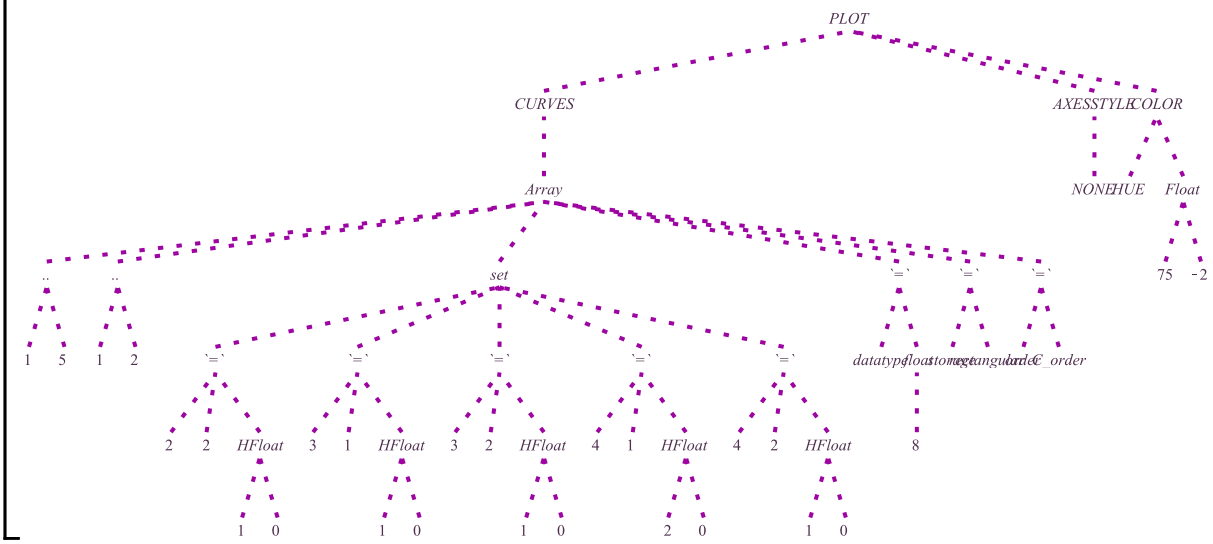
Malheureusement, on ne voit pas clairement l'arborescence... trop d'éléments dans cette struture de données.

Voyons voir avec une structure graphique plus simple.

```
> Graphe:=PLOT(CURVES(Array(1 .. 5, 1 .. 2, [[0, 0], [0, 1], [1, 1],
  [2, 1], [0, 0]], 'datatype' = 'float[8]', 'order' = 'C_order')),
  AXESSTYLE(NONE),COLOR(HUE, .75));
```



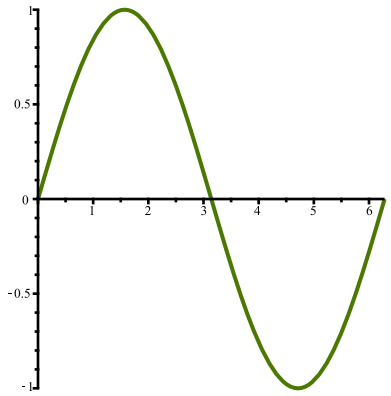
```
> display(arbre(Graphe),size=[1200,400]);
```



En étant convaincu maintenant que la macro-commande `plot` crée une structure de données, voyons comment nous pouvons accéder à ses opérandes.

Revenons à la structure `Mon_graphe` et voyons comment on accède à la matrice de points en tant qu'objet de type `Matrix`.

```
> Mon_Graphe:=plot([x,sin(x),x=0..2*Pi],color="Niagara 3",numpoints=25)
  :
  Mon_Graphe;
```



Obtenons d'abord le nombre d'opérandes que comporte la structure `Mon_Graphe`:

```
> nops(Mon_Graphe);
```

$$6 \tag{3.2}$$

Obtenons la liste de ces 6 opérandes.

```
> L:=[seq(op(k,Mon_Graphe),k=1..nops(Mon_Graphe))];
```

$$L := \left[CURVES\left(\begin{bmatrix} 0. & 0. \\ 0.1369555588 & 0.1365278183 \\ 0.2739111176 & 0.2704988084 \\ 0.3930754171 & 0.3830310938 \\ 0.5122397165 & 0.4901307211 \\ 0.6462524342 & 0.6021987866 \\ 0.7802651519 & 0.7034678946 \\ 0.9151661929 & 0.7926639188 \\ 1.0500672339 & 0.8674566773 \\ 1.1843271138 & 0.9262456562 \\ \vdots & \vdots \end{bmatrix}_{57 \times 2 \text{ Matrix}} \right), COLOUR(RGB, 0.2901960800, \right. \tag{3.3}$$

$$\left. 0.4705882400, 0.), AXESLABELS("", ""), AXESTICKS(DEFAULT, DEFAULT, FONT(times, roman, 8)), ROOT(BOUNDS\_X(0), BOUNDS\_Y(0), BOUNDS\_WIDTH(300), BOUNDS\_HEIGHT(300)), VIEW(DEFAULT, DEFAULT) \right]$$

Considérons ensuite le premier opérande, soit la structure de données `CURVES`.

```
> op(1,L);
```

(3.4)

$$CURVES\left(\begin{bmatrix} 0. & 0. \\ 0.1369555588 & 0.1365278183 \\ 0.2739111176 & 0.2704988084 \\ 0.3930754171 & 0.3830310938 \\ 0.5122397165 & 0.4901307211 \\ 0.6462524342 & 0.6021987866 \\ 0.7802651519 & 0.7034678946 \\ 0.9151661929 & 0.7926639188 \\ 1.0500672339 & 0.8674566773 \\ 1.1843271138 & 0.9262456562 \\ \vdots & \vdots \end{bmatrix}_{57 \times 2 \text{ Matrix}}\right) \quad \textbf{(3.4)}$$

Pointons vers le premier argument de `CURVES`, soit la matrice de points.

```
> Matrice:=op(1,op(1,L));
```

$$Matrice := \begin{bmatrix} 0. & 0. \\ 0.1369555588 & 0.1365278183 \\ 0.2739111176 & 0.2704988084 \\ 0.3930754171 & 0.3830310938 \\ 0.5122397165 & 0.4901307211 \\ 0.6462524342 & 0.6021987866 \\ 0.7802651519 & 0.7034678946 \\ 0.9151661929 & 0.7926639188 \\ 1.0500672339 & 0.8674566773 \\ 1.1843271138 & 0.9262456562 \\ \vdots & \vdots \end{bmatrix}_{57 \times 2 \text{ Matrix}} \quad \textbf{(3.5)}$$

Obtenons le nombre d'opérandes de cette matrice.

```
> nops(Matrice);
```

$$3 \quad \textbf{(3.6)}$$

Le premier opérande est le format de cette matrice. Son format correspond au nombre de points générés mais ne correspond pas nécessairement au nombre demandé (spécifié par `numpoints`) .

```
> Nombre_de_points_généré:=op([1,1],Matrice);
```

$$Nombre\_de\_points\_généré := 57 \quad \textbf{(3.7)}$$

Le troisième opérande nous informe quant à la structure de la matrice de stockage.

```
> op(3,Matrice);
```

$$datatype = float_8, storage = rectangular, order = Fortran\_order, shape = [\ ] \quad \textbf{(3.8)}$$

Notons que le mode de stockage de cette matrice est en ligne: *storage* = *rectangular*. Voici comment pointer

vers les éléments de ce matrice.

```
> [Matrice[1,1],Matrice[1,2]];
  [Matrice[2,1],Matrice[2,2]];
  [Matrice[3,1],Matrice[3,2]];
  # etc
  [Matrice[Nombre_de_points_généré,1],Matrice[Nombre_de_points_généré,
  2]];
```

$$[0., 0.]$$

$$[0.1369555588, 0.1365278183]$$

$$[0.2739111176, 0.2704988084]$$

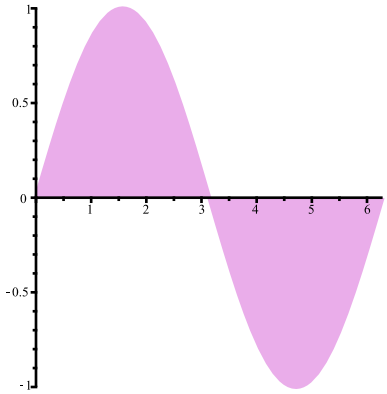
$$[6.2831853080, 8.2041371406\,10^{-10}] \quad \textbf{(3.9)}$$

Obtenons la liste des points générés de la matrice de stockage directement (terminez la requête avec le point-virgule pour visualiser la liste des points).

```
> Points:=[seq([Matrice[k,1],Matrice[k,2]],k=1.
  .Nombre_de_points_généré)]:
```

Voilà ! Retraçons avec `plot`, le graphique sur la base de la liste des points extraits.

```
> plot(Points,filled=true,color=plum);
```



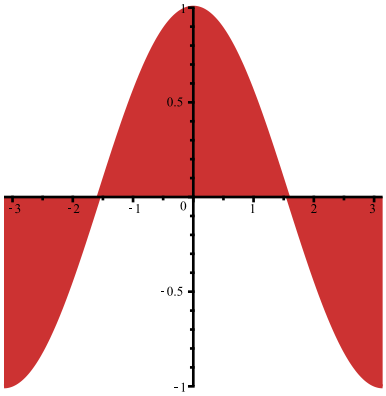
Quelques exemples de remplissage

L'option `filled=true` colore la région comprise en la courbe et l'axe des x. Illustrons un tel coloriage avec le graphique cartésien de la fonction cosinus sur $[-\pi, \pi]$.

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

$$f := x \mapsto \cos(x) \quad \textbf{(4.1)}$$

```
> Surface:=plot([x,f(x),x=-Pi..Pi],color=orange,numpoints=300,filled=
  true):
  Surface;
```



Si le coloriage désiré était plutôt celui de la région fermée limitée par cette courbe et le segment reliant les deux points du bas, il nous faudrait procédé autrement.

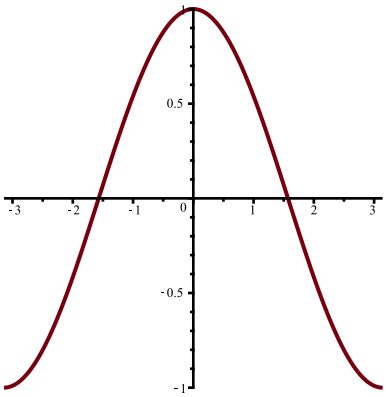
Une façon de faire est de créer une région de remplissage comme étant une région polygonale construite avec les points qui seront extraits du tracé de cette courbe. Puis, utiliser la macro-commande `polygon` de l'extension `plottools` qui permet le tracé d'une région polygonale sur la base d'une liste de points. Cette région polygonale est automatiquement fermée par un segment reliant les premier et dernier points de la liste donnée.

Pour automatiser l'extraction de la liste des points d'une structure graphique déjà créée, écrivons une procédure d'extraction qui va, bien sûr, prendre en charge éventuellement le nombre de points spécifié dans le tracé de cette structure.

```
> Liste_des_points:=proc(Graphe)
   local Matrice,Nombre_de_points_demandé,Points;
   Matrice:=op(1,op(1,Graphe));
   Nombre_de_points_demandé:=op([1,1],Matrice);
   Points:=[seq([Matrice[k,1],Matrice[k,2]],k=1..
  Nombre_de_points_demandé)];
  end proc:
```

Extrayons les points du tracé;

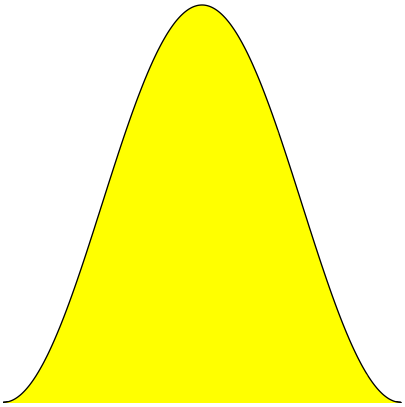
```
> Courbe:=plot([x,f(x),x=-Pi..Pi],numpoints=300);
```



```
> L:=Liste_des_points(Courbe):
```

Colorions maintenant la région désirée.

```
> Surface:=plottools[polygon](L,color=yellow):
  display(Surface,thickness=0,axes=none);
```



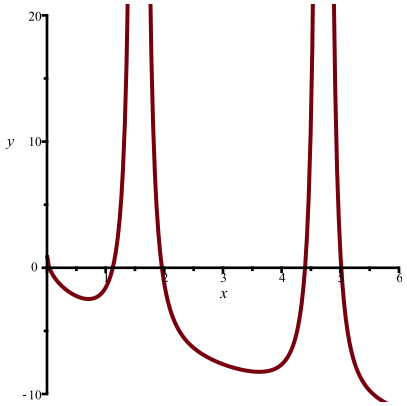
Voici quelques exemples instructifs.

Exemple 1

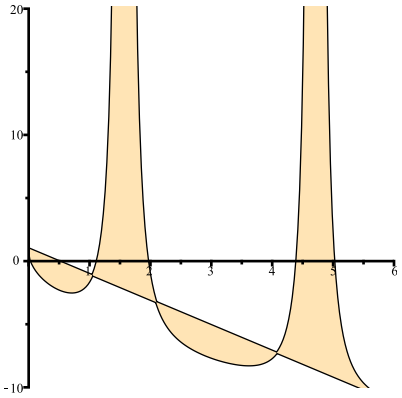
```
> f:=x->1/((cos^2)(x))-5*sqrt(x);
```

$$f := x \mapsto \frac{1}{(\cos^2)(x)} - 5\cdot\sqrt{x} \qquad \textbf{(4.1.1)}$$

```
> Courbe:=plot(f(x),x=0..6,y=-10..20):
  Courbe;
```




```
> L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color="Moccasin"):
  display(Surface,thickness=0,view=[0..6,-10..20]);
```



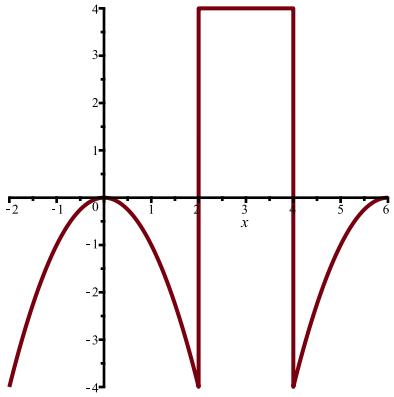
Exemple 2

```
> f:=piecewise(x>=-2 and x<2,-x^2,x>2 and x<=4,4,x>4 and x<=8,-(x-6)
```

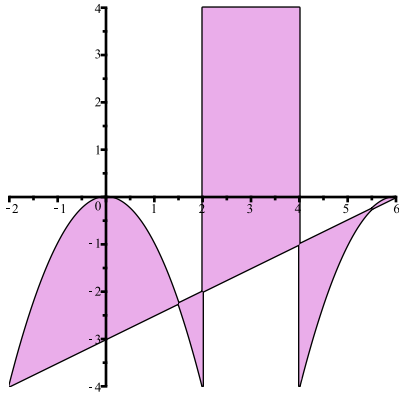
```
^2);
```

$$f := \begin{cases} -x^2 & -2 \le x < 2 \\ 4 & 2 < x \le 4 \\ -(x-6)^2 & 4 < x \le 8 \end{cases} \qquad \textbf{(4.2.1)}$$

```
> Courbe:=plot(f(x),x=-2..6,numpoints=300):
  Courbe;
```




```
> L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color=plum):
  display(Surface,thickness=0);
```

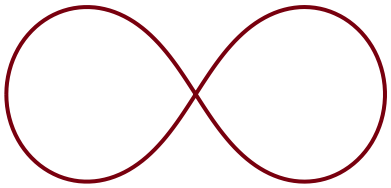


Remarque: L'option `discont` dans le tracé de la courbe précédent aurait dû être utlilisée afin d'obtenir un tracé correct de la fonction f. Par contre, son utilisation n'aurait pas permis d'obtenir le coloriage précédent.

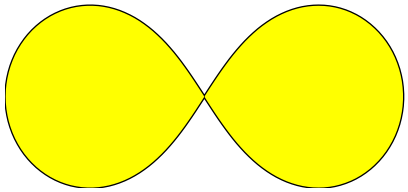
Exemple 3

Les courbes tracées en coordonnées polaires peuvent être fermées assez facilement et donc probablement renmplies convenablement.

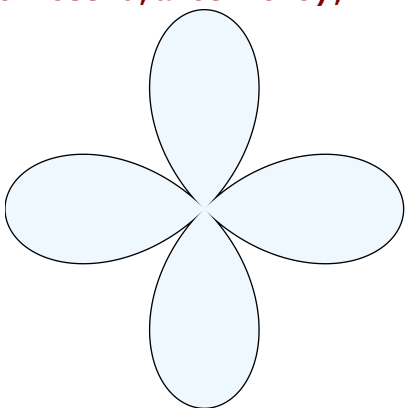
```
> Courbe:=plots[polarplot]([[2*cos(t),sin(t),t=-Pi..Pi]],axes=none):
  Courbe;
```



```
> L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color=yellow):
  display(Surface,thickness=0,axes=none,scaling=constrained);
```

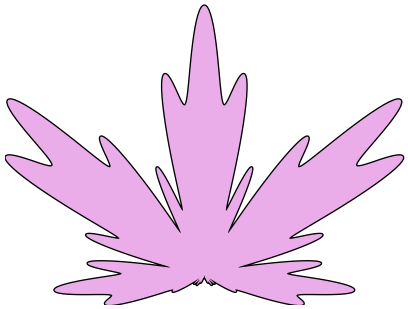


```
> Courbe:=plots[polarplot](cos(2*t),t=0..2*Pi,numpoints=600,axes=
  none):
  L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color="AliceBlue"):
  display(Surface,thickness=0,axes=none);
```



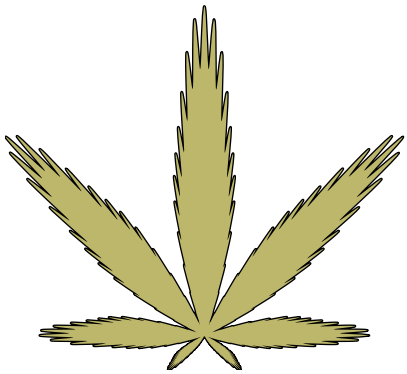
```
> S:=t->100/(100+(t-Pi/2)^8):
  R:=t->S(t)*(2-sin(7*t)-cos(30*t)/2):
  Courbe:=plots[polarplot]([R,t->t,-Pi/2..3/2*Pi],numpoints=1000,
  axes=none):
  L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color=plum):
```

```
display(Surface,thickness=0,axes=none,scaling=constrained);
```

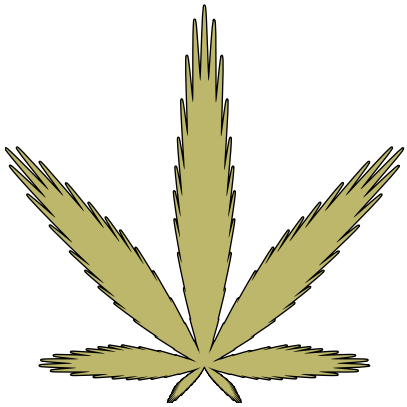


Cannabis curve de Wolfram: http://m.wolframalpha.com/input/?i=cannabis+curve

```
> Courbe:=plots[polarplot]((1+.9*cos(8*t))*(1+.1*cos(24*t))*(1+.1*
  cos(200*t))*(1+sin(t)),t=0..2*Pi,numpoints=1200,axes=none):
  L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color="DarkKhaki"):
  display(Surface,thickness=0,axes=none,scaling=constrained);
```



```
> x:=t->(sin(t)+1)*cos(t)*(9/10*cos(8*t)+1)*(1/10*cos(24*t)+1)*
  (1/10*cos(200*t)+9/10):
  y:=t->sin(t)*(sin(t)+1)*(9/10*cos(8*t)+1)*(1/10*cos(24*t)+1)*
  (1/10*cos(200*t)+9/10):
  Courbe:=plot([x(t),y(t),t=0..2*Pi],numpoints=1200,axes=none):
  L:=Liste_des_points(Courbe):
  Surface:=plottools[polygon](L,color="DarkKhaki"):
  display(Surface,thickness=0,axes=none,scaling=unconstrained);
```



Exemple 4

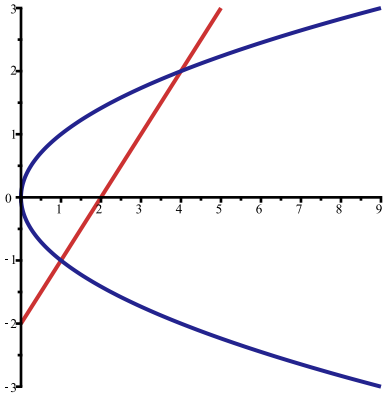
Nous allons colorier la région comprise entre la courbe d'équation $x = y^2$ et la courbe $y = x - 2$.

```
> f:=x->x-2;
  g:=y->y^2;
```

$$f := x \mapsto x - 2$$

$$g := y \mapsto y^2 \qquad \textbf{(4.4.1)}$$

```
> C1:=plot([x,f(x),x=0..5],color=orange):
  C2:=plot([g(y),y,y=-3..3],color=navy):
  display(C1,C2);
```



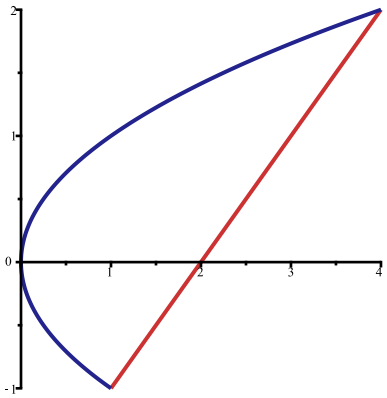
Obtenons l'abcisse des points d'intersection. Cela va permettre de limiter la région à tracer afin que l'on puisse extraire la liste des points servant à préciser la région polygonale à colorier.

```
> solve(x=(x-2)^2,{x});
```

$$\{x = 1\}, \{x = 4\} \qquad \textbf{(4.4.2)}$$

Remarque: Dans cet exemple-ci, nous aurions pu très bien obtenir ces abscisses sans l'aide de la macro-commande `solve`.

```
> Courbe1:=plot([x,x-2,x=1..4],color=orange):
  Courbe2:=plot([y^2,y,y=-1..2],color=navy):
  plots[display](Courbe1,Courbe2);
```



Utilisons la procédure Liste_des_points.

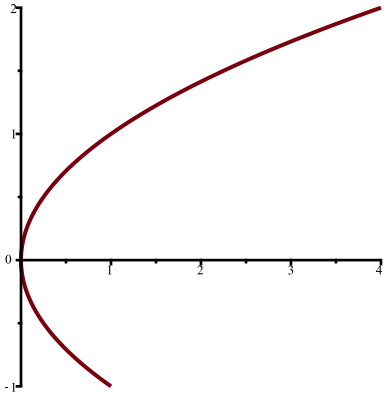
Procédure « Liste_des_points »

```
> Liste_des_points:=proc(Graphe)
   local Matrice,Nombre_de_points_demandé,Points;
   Matrice:=op(1,op(1,Graphe));
   Nombre_de_points_demandé:=op([1,1],Matrice);
   Points:=[seq([Matrice[k,1],Matrice[k,2]],k=1..
  Nombre_de_points_demandé)];
  end proc:
```

Il est suffisant ici d'extraire seulement les points de Courbe2 pour réaliser une région polygonale fermée.

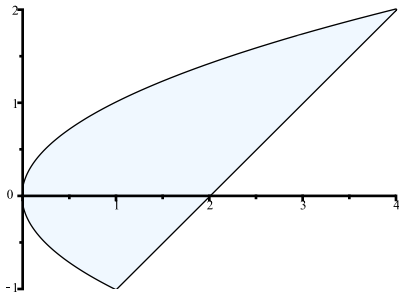
```
> L:=Liste_des_points(Courbe2):
```

```
> plot(L);
```



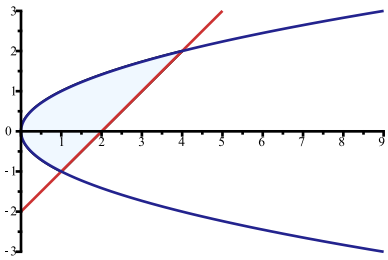
En effet, le segment reliant le premier point et le dernier de cette liste générée par la macro-commande `polygon` se supperpose sur la droite $y = x - 2$.

```
> Surface:=plottools[polygon](L,color="AliceBlue"):
  plots[display](Surface,thickness=0,scaling=constrained);
```



Reste à supperposer dans un même graphique cette région ainsi que les courbes C1 et C2.

```
> plots[display](C1,C2,Surface,thickness=0,scaling=constrained);
```



Exemple 5

Le dernier exemple présente une difficulté. La difficulté consiste à construire une région polygonale en concaténant deux listes de points extraits de deux structures graphiques. Il faut veiller à ce que la concaténation de ces deux listes de points se fasse de telle manière que le coloriage soit correctement effectué par la macro-commande `polygon`.

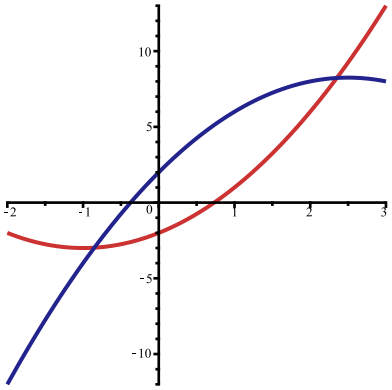
Colorions la région bornée par les paraboles d'équations $y = x^2 + 2x - 2$ et $y = 2 + 5x - x^2$.

```
> f:=x->x^2+2*x-2;
  g:=x->2+5*x-x^2;
```

$$f := x \mapsto x^2 + 2 \cdot x - 2$$

$$g := x \mapsto 2 + 5 \cdot x - x^2 \qquad \textbf{(4.5.1)}$$

```
> C1:=plot([x,f(x),x=-2..3],color=orange):
  C2:=plot([x,g(x),x=-2..3],color=navy):
  plots[display](C1,C2);
```

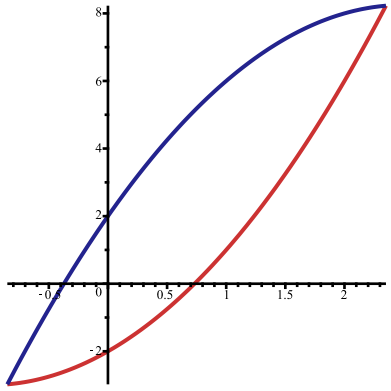


Obtenons l'abcisse des points d'intersection. Cela va permettre de limiter la région à tracer afin que l'on puisse extraire la liste des points servant à préciser la région polygonale à colorier.

```
> solve(f(x)=g(x),{x});
```

$$\left\{x = \frac{3}{4} + \frac{\sqrt{41}}{4}\right\}, \left\{x = \frac{3}{4} - \frac{\sqrt{41}}{4}\right\} \qquad \textbf{(4.5.2)}$$

```
> Courbe1:=plot([x,f(x),x=3/4-(1/4)*sqrt(41)..3/4+(1/4)*sqrt(41)],
  color=orange):
  Courbe2:=plot([x,g(x),x=3/4-(1/4)*sqrt(41)..3/4+(1/4)*sqrt(41)],
  color=navy):
  plots[display](Courbe1,Courbe2);
```



Utilisons la procédure Liste_des_points. Nous devons ici extraire les points de la courbe en `orange` et en `navy`.

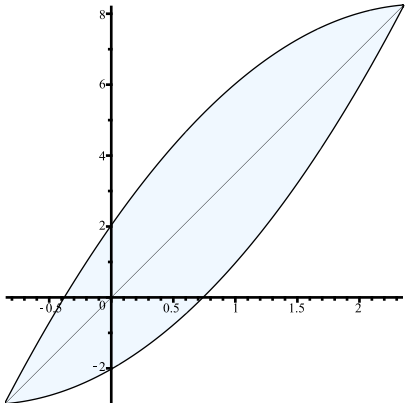
```
> L1:=Liste_des_points(Courbe1):
  L2:=Liste_des_points(Courbe2):
```

Effectuons la concaténation de ces deux listes de points telles quelles.

```
> L:=[op(L1),op(L2)]:
```

Colorions cette région polygonale pour voir.

```
> Surface:=plottools[polygon](L,color="AliceBlue"):
  plots[display](Surface,thickness=0);
```



La ligne médiane est le résultat du parcours de la macro-commande `polygon` de la partie négative des abcisses vers la partie positive de chaque liste L1 et L2 respectivement. Pour aider à mieux comprendre cette explication, vous pouvez faire l'affichage des points de la liste L.

Pour éviter le tracé de cette ligne médiane, nous allons plutôt concaténer à la liste L1, la liste L2 inversé. Ainsi, il y aura une adjacence de tous les points du polygone délimitant la région.

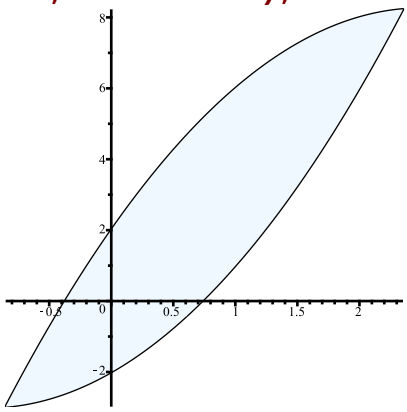
Créons, pour le besoin de la cause, une procédure inversant les éléments d'une liste. Cela pourra vous servir en d'autres occasions.

```
> Inverse:=proc(L) local K;
  K:=[seq(L[nops(L)+1-n],n=1..nops(L))]
  end proc:
```

Concaténons L1 et L2 inversée.

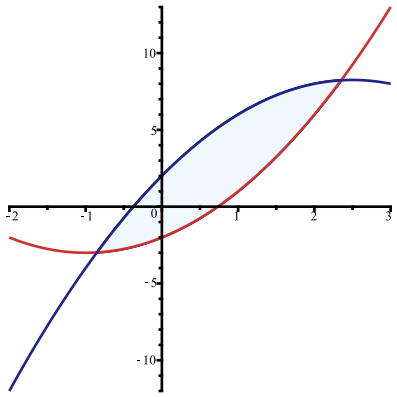
```
> L:=[op(L1),op(Inverse(L2))]:
```

```
> Surface:=plottools[polygon](L,color="AliceBlue"):
  plots[display](Surface,thickness=0);
```



Voilà! Reste à supperposer dans une même graphique cette région ainsi que les courbes C1 et C2.

```
> plots[display](C1,C2,Surface,thickness=0);
```



Exercices

No. 1

– *Si nécessaire, exécuter de nouveau la procédure* `arbre`.

Arbre

```
> arbre:=proc(expr)
   local k,Options_base,Opts,plot1,plot2,Size;
   global affx,affy,affT,affS,marque;
   Opts:=[args[2..nargs]];
   Options_base:=NULL;
    for k from 1 to nops(Opts) do
     if lhs(map(op,k,Opts))=size then Options_base:=size=rhs(Opts
  [k]) fi;
    od;
    marque:=proc(expr)
     local xa,xb,pos,lpos;
     global affx,affy,affT,affS;
      if type(expr,symbol) or type(expr,integer) then
       affT:=affT,[affx,affy,expr];
       xa:=affx;
       affx:=affx+1;
       xa
      else
       affy:=affy-1;
       lpos:=map(marque,[op(expr)]);
       xa:=op(1,lpos);
       xb:=op(nops(lpos),lpos);
       affy:=affy+1;
       pos:=(xa+xb)/2;
       affT:=affT,[pos,affy,op(0,expr)];
```

```
      affS:=affS,op(map(proc(a,b) CURVES([[a,affy-0.85],[b,
    affy-.15]],\
                COLOR(RGB,.624,.624,.373))end,lpos,pos));
       pos;
      fi
     end proc;
    affx:=0; affy:=0;affT:=NULL; affS:=NULL;
    marque(expr);
    plot1:=plots[textplot]({affT},font=[TIMES,ROMAN,8],color=navy);
    plot2:=PLOT(affS,THICKNESS(0),LINESTYLE(2));
    unassign('affx,affy,affT,affS,marque');
    plots[display]({plot1,plot2},axes=NONE,Options_base);
   end proc:
```

– Afficher l'arborescence de l'expression $\frac{1}{x}$.

Qu'observez-vous ?
Quel est l'opérateur (noeud) ?
Combien y a-t-il d'opérandes ?

– Afficher l'arborescence de l'expression $\frac{3}{2}$.

Qu'observez-vous ?
Quel est l'opérateur (noeud) ?
Combien y a-t-il d'opérandes ?

– Afficher l'arborescence de l'expression 1.5.

Qu'observez-vous ?
Quel est l'opérateur (noeud) ?
Combien y a-t-il d'opérandes ?

No. 2

– Créer la somme $\sin(x) + \sqrt{\frac{\sin(3x)}{\sin(4x)}} + \sin(5x)$ en la nommant `Somme` (très original...)

– Afficher l'arborescence de Somme.

À l'aide de la macro-commande `op`,

– Pointer vers le deuxième terme de `Somme`;

– Pointer vers le radicand;

– Pointer vers le dénominateur du radicand;

– Pointer vers l'argument du dénominateur du radicand.

No. 3

– Créer l'expression $\frac{x^2 - 6x^2 - 12x + 49}{(x-2)(x-7)}$ en la nommant `Formule`

– Afficher l'arborescence de `Formule`.

– À l'aide de la maco-commande quo, assigner la division euclidienne comme suit:

```
Quotient := quo(numer(Formule), denom(Formule), x, 'Reste');
```

– Saisir l'égalité

```
Équation := Formule = Quotient + Reste/denom(Formule);
```

– À l'aide de la macro-commande `op`, afficher le numérateur du membre de droite (*Reste*) de l'objet `Équation`.

No. 4

– Créer la liste suivante de points à l'aide de la macro-commande seq

Points := [[0, 1], [−1, 2], [−2, 3], [−3, 4], [−4, 5], [−5, 6], [−6, 7]]

```
>
```

– Afficher l'arborescence de *Points*.

– Afficher la liste des opérandes de la liste *Points* à l'aide de la macro-commande `op`.

– Afficher le troisième opérande de la liste *Points* à l'aide de la macro-commande `op`.

– Afficher le quatrième opérande de la liste *Points* en utilisant la notation indicielle.

– Afficher le second opérande du quatrième opérande de la liste *Points* à l'aide de la macro-commande `op`.

– Afficher le second opérande du quatrième opérande de la liste Points en utilisant la notation indicielle.

No.5

Colorier la région bornée par la droite d'équation $y = \frac{1}{2}x$ et la parabole d'équation $y^2 = 8 - x$.

No. 6

Colorier la région bornée dans le premier quadrant par les courbes d'équations $y = \frac{7x}{(x^2+1)^{\frac{3}{2}}}$ et $y = \frac{1}{3}x^2$.