Assigner. Evaluer. Substituer.

Bernard Dupont

Bernard.Dupont@univ-lille1.fr

Ce chapitre est exclusivement consacré à des questions informatiques dont l'intérêt n'est pas évident en première lecture mais qui se révèlent vite fondamentales en pratique au fur et à mesure qu'on progresse dans l'utilisation de Maple. Trois thèmes son traités :

- L'**assignation**, encore appelée **affectation**, qui a l'immense avantage de stocker une expression longue telle que $2\arctan\left(\frac{\sqrt{6}\sqrt{t}}{e^{366\,t}}+\frac{4.478411055\,10^{8}\ln\left(t^{3}+2\right)}{\ln(10)}\right)$ dans un endroit connu par son adresse, par exemple *x1* , de sorte que tout adressage ultérieur à cette variable nominale permet d'accéder à son contenu. L'opération contraire est la désassignation qui brise la correspondance entre la variable nominale et son contenu.
- L'**évaluation**, qui remplace une variable muette par une autre expression, et permet par exemple de connaître la valeur de $2\arctan\left(\frac{\sqrt{6}\sqrt{t}}{e^{366\,t}}+\frac{4.478411055\,10^{8}\ln\left(t^{3}+2\right)}{\ln(10)}\right)$ quand $t=\cos\left(\frac{\pi}{6}\right)$.
- La **substitution**, qui remplace une partie d'expression par une autre expression.

Assignation et désassignation : règles de base

En informatique, "assigner" - on dit aussi "affecter" ou "identifier"- consiste à associer à une variable nominale - un "nom" - une expression ou objet informatique quelconque (à condition qu'elle ou il soit valide). "Désassigner" un nom est le procédé qui "libère" ce nom d'un contenu ... autre que son propre nom.

Assignation ou affectation

Assignation simple

Maple permet de manipuler des objets qui sont des valeurs numériques (réelles ou complexes), des expressions algébriques symboliques, des fonctions ou des procédures, des listes, ensembles et tableaux, des chaînes de caractères, etc..
En pratique, il est commode de donner un nom à ces objets qui peuvent être longs et pénibles à recopier lors de manipulations, même si l'opération "copier-coller" écourte ces opérations ingrates. Il est clair qu'on préférera attribuer un nom à l'expression suivante :

```
> restart;
  2*arctan(sqrt(6*t)/(exp(0366*t))+40^5.4*log[10](t^3+2));
```

$$2\arctan\left(\frac{\sqrt{6}\sqrt{t}}{e^{366\,t}}+\frac{4.478411055\,10^{8}\ln\left(t^{3}+2\right)}{\ln(10)}\right)$$

afin de la manipuler facilement par la suite.

Donner un nom à un objet s'apelle **assigner** (en anglais : assign), ou **affecter** ou encore **identifier** et **l'identificateur est le nom attribué à un objet**. La méthode est simple. D'abord écrire l'identificateur, ce qui présuppose le choix d'un nom, par exemple **`x1`**, qui peut être n'importe quoi sauf un identificateur existant en interne pour Maple (par exemple **`Pi`**) et, de

préférence, éviter de commencer par un "tiret bas". Ensuite, taper "deux points" puis "égal", soit `:=`. Enfin, écrire l'expression.

```
> x1:=2*arctan(sqrt(6*t)/(exp(0366*t))+40^5.4*log[10](t^3+2)
  );
```

$$x1 := 2 \arctan\left(\frac{\sqrt{6}\,\sqrt{t}}{e^{366\,t}} + \frac{4.478411055\ 10^{8} \ln\left(t^{3}+2\right)}{\ln(10)} \right)$$

Maple lit cette commande en commençant par "évaluer" le membre de droite, puis affecte le résultat dans le membre de gauche, ce qui fait que l'identificateur contient désormais la valeur du membre de droite. On peut alors utiliser l'identificateur dans tout contexte, à condition que les nouvelles expressions formées soient valides.

```
> x1/8+9*t;
```

$$\frac{1}{4} \arctan\left(\frac{\sqrt{6}\,\sqrt{t}}{e^{366\,t}} + \frac{4.478411055\ 10^{8} \ln\left(t^{3}+2\right)}{\ln(10)} \right) + 9\,t$$

Il ne faut pas hésiter à proposer des noms aux objets qu'on va manipuler dans la suite du worksheet, même s'il existe des possibilités pour paresseux cherchant à éviter l'écriture de lignes de programme. Par exemple, il est possible de se référer au dernier output en utilisant le racourci "pour-cent" `%` , à l'avant dernier en utilisant `%%` et à l'antépénultième avec `%%%`.

```
> sqrt(2);evalf(%);(%%-1)^2;
```

$$\sqrt{2}$$

$$1.414213562$$

$$\left(\sqrt{2}-1\right)^{2}$$

Mais cette facilité induit parfois en erreur car Maple se réfère toujours au dernier output chronologique, qui n'est pas nécessairement le dernier output à l'écran! Il est donc bien plus sage d'affecter à une variable l'objet qu'on désire transformer.

Un identificateur étant créé, il a deux propriétés importantes à ce niveau de la présentation.

Propriété 1 : il est possible de connaître le contenu d'un identificateur à tout moment d'une session en tapant simplement son nom suivi de `;` et en validant.

```
> x1;
```

$$2 \arctan\left(\frac{\sqrt{6}\,\sqrt{t}}{e^{366\,t}} + \frac{4.478411055\ 10^{8} \ln\left(t^{3}+2\right)}{\ln(10)} \right)$$

Propriété 2 : Le contenu d'un identificateur se modifie en le réaffectant.

```
> x1:=exp(-rho*t);#l'identificateur x1 est modifié
  x1;#le contenu de la variable x1 a changé
```

$$x1 := e^{-\rho\,t}$$

$$e^{-\rho\,t}$$

En économie, les utilisateurs de Maple n'hésitent jamais à donner des noms au pouvoir évocateur à leurs identificateurs, ce qui permet d'éviter des confusions et des erreurs dans les longues feuilles de travail. Par exemple :

```
> util:=U(c(t));#fonction d'utilité intertemporelle
  actual:=exp(-rho*t);#facteur d'actualisation en temps
  continu
```

```
fonctionnelle:=Int(actual*util,t=0..infinity);
#fonctionnelle objectif dans les théories de la croissance néo-classique
```

$$util := U(c(t))$$

$$actual := e^{-\rho t}$$

$$fonctionnelle := \int_0^{\infty} e^{-\rho t} U(c(t)) \, dt$$

Assignation multiple

Une feuille de travail comporte généralement de très nombreux identificateurs qui sont définis au fur et à mesure des besoins dans la session. Mais si on a besoin de définir plusieurs identificateurs à un moment donné, on peut faire une assignation multiple : à un certain nombre de "noms" séparés par des virgules, on fait correspondre terme à terme un même nombre d'objets par l'opérateur **:=**.

```
> consom,Kt,tx_int,fx:=c*Y+C0,10*1.025^t,0.025,log(x);
```

$$consom, Kt, tx\_int, fx := c\,Y + C0,\ 10\ 1.025^{t},\ 0.025,\ \ln(x)$$

Les assignations individuelles ont bien été faites :

```
> consom;
  Kt;
  tx_int;
  fx;
```

$$c\,Y + C0$$

$$10\ 1.025^{t}$$

$$0.025$$

$$\ln(x)$$

Désassignation

Si l'assignation - et la réassignation - établissent une correspondance entre un nom et un objet, la désassignation brise ce lien et la variable nominale a alors pour contenu ... elle même! Cette manipulation est plus importante qu'on pourrait le croire. Par exemple, il est possible que la variable **i** représente le taux d'intérêt au début d'une session de travail et qu'on est besoin par la suite de cette même variable pour compter le nombre de boucles dans un algorithme. Dans ce cas, il faut absolument la "libérer" de sa première affectation.

Désassignation d'un identificateur

Désassigner un identificateur revient à le libérer de son contenu. L'identificateur étant un nom, on peut y voir une variable informatique et la désassignation s'interprète comme le fait de rendre "libre" cette variable. Elle devient "libre" en ce sens qu'elle n'est plus liée à un objet précis.
Techniquement, la désassignation se fait en assignant à l'identificateur son propre nom précédé et succédé d'une apostrophe **'** .

```
> consom;#rappel du contenu de la variable consom
  Kt;#rappel du contenu de la variable Kt
  consom:='consom';#désassignation de la variable consom
```

```
Kt:='Kt';#désassignation de la variable Kt
consom;#la variable consom a pour contenu son nom
Kt;#la variable Kt a pour contenu son nom
```

$$c\,Y + C0$$

$$10\ 1.025^{t}$$

$$consom := consom$$

$$Kt := Kt$$

$$consom$$

$$Kt$$

La commande **about()** permet toujours de connaître le statut d'un identificateur. Si la variable a un contenu, l'output le rappelle en première ligne et donne quelques commentaires dans les lignes suivantes. Si la variable est libre, l'output est laconique `nothing known about this object` (ce commentaire signifie aussi qu'aucune hypothèse n'est faite sur l'objet).

```
> about(x1);about(fx);about(util);about(tx_int);about
  (consom);
exp(-rho*t):
  nothing known about this object

ln(x):
  nothing known about this object

U(c(t)):
  nothing known about this object

.25e-1:
  All numeric values are properties as well as objects.
  Their location in the property lattice is obvious,
  in this case float.

consom:
  nothing known about this object
```

Réinitialisation de toutes les variables : `restart`

Toutes les variables informatiques sont réinitialisées au cours d'une session en validant la commande **restart;**

```
> restart;
  about(x1,fx,util,tx_int,consom,Kt,actual,fonctionnelle);
x1:
  nothing known about this object

fx:
```

```
  nothing known about this object

util:
  nothing known about this object

tx_int:
  nothing known about this object

consom:
  nothing known about this object

Kt:
  nothing known about this object

actual:
  nothing known about this object

fonctionnelle:
  nothing known about this object
```

Maple vide complétement la mémoire non seulement de tous les identificateurs, mais aussi de tous les autres éléments qui y étaient mémorisés, en particulier les paquetages chargés avant le redémarrage. Cette manoeuvre radicale est souvent effectuée au cours d'une même session tant il est vrai qu'un bon coup de balai rend la maison plus saine et attrayante. Par ailleurs, une tradition bien établie veut qu'une session de travail commence toujours par un **restart**. En principe, c'est inutile mais nous nous y conformerons.

Evaluation

Une expression **symbolique** contient par définition au moins une variable libre. Elle est dite évaluée si cette dernière "pointe" sur un contenu autre que son nom, par exemple un nombre, et modifie en conséquence l'expression de départ. Par exemple, le polynome x^2+x+1 vaut 3 si $x=1$, 1 si $x=-1$, $(a+b)^2+a+b+1$ si $x=a+b$. Cette section traite la question de l'évaluation et commence par exposer le principe général d'évaluation complète.

Principe d'évaluation complète

Soit un identificateur dont le contenu est une expression symbolique, par exemple un polynôme à une variable :

```
> restart;#on vide le buffer
  P:=x^2+x+1;#on assigne à l'identificateur P un polynôme qui
  dépend de la variable libre x
```

$$P := x^2+x+1$$

On obtient les valeurs prises par le polynôme en donnant des valeurs différentes à x. Il suffit d'identifier **x**, puis de valider **P.**

```
> x:=1;P;
  x:=2;P;
  x:=3;P;
```

$$x := 1$$
$$3$$
$$x := 2$$
$$7$$
$$x := 3$$
$$13$$

Que se passe t-il? Maple a en mémoire les assignations. Il évalue en les combinant toutes. A la première ligne, il évalue le polynôme identifié par **P** pour la valeur **1** assignée juste avant à l'identificateur **x**. A la seconde ligne, il évalue le même polynôme à la nouvelle valeur assignée à l'identificateur **x**. A la troisième ligne, même chose. Dans cet exemple élémentaire, l'évaluation de **P** est évidemment complète au sens où elle n'est menée qu'une fois par remplacement d'une variable informatique par sa valeur numérique. Plus généralement, on dira que l'évaluation est complète quand elle est poursuivie tant qu'une variable peut encore pointer sur un contenu. L'exemple suivant permet de mieux le mettre en évidence.

```
> x:=y+2*Pi;#le contenu de x est l'expression y+2*Pi
  y:=c^2;#le contenu de y est l'expression c^2
  P;#évaluation 1 de P
  c:=1;#le contenu de c est le nombre 1
  P;#évaluation 2 de P
```

$$x := y + 2\,\pi$$
$$y := c^2$$
$$\left(c^2 + 2\,\pi\right)^2 + c^2 + 2\,\pi + 1$$
$$c := 1$$
$$\left(1 + 2\,\pi\right)^2 + 2 + 2\,\pi$$

L'évaluation 1 est complète en ce sens qu'elle remplace dans l'expression polynômiale la variable **x** par son contenu, qui est l'expression **y+2*Pi**, puis poursuit en remplaçant **y** par son contenu, qui est **c^2**. Le résultat renvoyé en output tient compte de deux remplacements.
Dans l'évaluation 2, Maple tient compte de l'affectation du nombre 1 à la variable **c**, de sorte que la variable **y** a pour contenu $1^2 = 1$, que **x** s'identifie du coup à $1+2\,\pi$ et que **P** a pour contenu après réaménagement $\left(1 + 2\,\pi\right)^2 + 2 + 2\,\pi$.

Dans tous les cas, le logiciel épuise toutes les possibilités de substitution avant d'évaluer définitivement une expression. Il faut d'ailleurs se méfier de ce principe apparemment plein de bon sens et très pratique. Il peut être source de gros déboires. En effet, **si Maple "travaille" sur et manipule les expressions originales pour mener à bien une évaluation complète, l'utilisateur n'a accès qu'à la dernière évaluation mémorisée** :

```
> P;#le contenu de P n'est plus un polynôme en x
  about(P);
```

$$\left(1 + 2\,\pi\right)^2 + 2 + 2\,\pi$$

```
(1+2*Pi)^2+2+2*Pi:
```

```
  nothing known about this object
```

Pour retrouver le polynôme de départ, il faut "libérer" la variable x.

```
> x:='x';P;about(P);
```

$$x := x$$

$$x^2 + x + 1$$

```
x^2+x+1:
  nothing known about this object
```

L'exemple suivant démontre le caractère déroutant du principe d'évaluation complète. Supposons qu'on ait besoin de travailler avec deux polynômes au cours d'une session.

```
> restart;
  P:=x^2+x+1;#le polynôme x^2+x+1 est assigné à la variable
  informatique P
  x:=2;#la valeur 2 est assignée à la variable informatique x
  P;#évaluation du polynôme pour x=2
```

$$P := x^2 + x + 1$$

$$x := 2$$

$$7$$

Jusque là, tout va bien. Assignons le second polynôme :

```
> Q:=x^3+1;#le polynôme x^3+1 est assigné à la variable
  informatique Q
```

$$Q := 9$$

En vertu du principe d'évaluation totale, Maple "voit" en **Q** un nombre et non un polynôme parce qu'il a rencontré la variable **x** et lui a immédiatement attribué la valeur 2. Dès lors, il n'est plus possible d'évaluer le polynôme **Q** pour d'autres valeurs de **x**.

```
> x:=1;P;Q;#évaluation de P et Q pour une autre assignation de
  x
```

$$x := 1$$

$$3$$

$$9$$

Catastrophe! On pensait calculer la valeur prise par chaque polynôme pour $x = 1$, mais seule la valeur prise par **P** est correcte puisque Maple avait évalué **Q** pour $x=2$ et a conservé en mémoire cette affectation.

Supposons à présent qu'on veuille retrouver l'expression de **P** et **Q** en x. On transforme alors x en variable libre - au sens où elle est libérée d'une valeur fixe - par la commande **x:='x'** . La commande **about()** permet, comme toujours, de connaître le statut d'une variable dans le buffer.

```
> about(x);x:='x';about(x);P;Q;
1:
  All numeric values are properties as well as objects.
  Their location in the property lattice is obvious,
```

```
  in this case integer.
```

$$x := x$$

```
x:
  nothing known about this object
```

$$x^2 + x + 1$$
$$9$$

Le procédé ne marche pas pour **Q** parce que Maple garde toujours en mémoire l'affectation de départ **x^3+1** pour **x:=2**. Il faut donc se résoudre à réécrire l'expression complète de **Q**.

```
> Q:=x^3+1;
```

$$Q := x^3 + 1$$

Puisque **x** est redevenue libre, on a bien une expression polynômiale.

Le comportement de Maple est différent selon qu'il y a présence ou non d'apostrophes dans une expression. S'il rencontre des apostrophes, l'évaluation consiste à les supprimer sans chercher à donner une évaluation numérique de l'expression : **on dit alors que l'évaluation est différée**.

```
> restart;
  P:=x^2+x+1;#le polynôme x^2+x+1 est assigné à la variable
  informatique P
  x:=2*y;#le contenu de l'identificateur x est 2*y
  y:=5;#le contenu de l'identificateur y est 5
  'P';#l'évaluation de cette expression est le nom P
  P;#l'évaluation de P est complète
```

$$P := x^2 + x + 1$$
$$x := 2\,y$$
$$y := 5$$
$$P$$
$$111$$

S'il n'y a pas d'apostrophes, le logiciel est construit de telle manière qu'il obéit à la règle d'évaluation complète : il épuise toutes les possibilités d'affectation définies depuis le précédent **restart**.On peut suivre les différentes étapes d'une évaluation grâce à la commande **eval(expression,n)** où *n* exprime l'étape d'évaluation.

```
> restart;P:=x^2+x+1;x:=y^2;y:=1;
  eval(P,1);#première évaluation
  eval(P,2);#deuxième éévaluation : x est remplacée par y^2
  eval(P,3);#troisième évaluation : y est remplacée par 1
  eval(P,4);#évaluation superflue puisqu'il n'y a plus de
  variable symbolique.
```

$$P := x^2 + x + 1$$
$$x := y^2$$
$$y := 1$$

$$x^2+x+1$$
$$y^4+y^2+1$$
$$3$$
$$3$$

Evaluation au cas par cas : `eval`, `Eval`, `value`

La méthode d'évaluation par assignation présentée dans le paragraphe précédent peut générer des erreurs quand elle est utilisée dans une session de travail longue où l'attention de l'utilisateur diminue tandis que la mémoire de Maple reste implacablement fidèle aux principes de la programmation. Il existe une méthode plus fiable, mais nettement moins commode puisqu'elle oblige à multiplier les écritures. La commande **`eval`** esquive les assignations car sa syntaxe est **`eval(xpr(x),x=a)`** où **`xpr(x)`** est une expression dépendant de la variable libre **`x`** et **l'égalité (et non l'assignation) `x=a` signifie que la variable `x` prend la valeur `a` pour cette commande et cette commande seulement**. Dans ce cas, l'évaluation ne modifie pas le contenu de l'identificateur **`P`** ni celui de **`x`**.

```
> restart;P:=x^2+x+1;eval(P,x=1);eval(P,x=y^2);
  P;#le contenu de P n'est pas modifié
  x;#le contenu de x est le nom x.
```

$$P := x^2+x+1$$
$$3$$
$$y^4+y^2+1$$
$$x^2+x+1$$
$$x$$

A la commande **`eval`** correspond la fonction dite inerte **`Eval`** (soit eval où le "e minuscule" est remplacé par un "e majuscule") qui enregistre une opération d'évaluation sans l'exécuter. L'output est alors une écriture symbolique sans ambiguité.

```
> Eval(P,x=a);
```

$$\left(x^2+x+1\right)\Big|_{x=a}$$

On dispose ainsi d'un moyen commode de documenter sa feuille de travail par combinaison de **`Eval`** et **`eval.`**

```
> Eval(P,x=a)=eval(P,x=a);
```

$$\left(x^2+x+1\right)\Big|_{x=a} = a^2+a+1$$

Il faut savoir que la forme inerte peut être évaluée effectivement par l'utilisation conjointe de **`value`**. Ainsi **`value(Eval(P,x=a))`** équivaut à **`eval(P,x=a)`**. Certains y voient un intérêt pour soigner la présentation des résultats. On procède comme suit :

```
> Eval(P,x=a):#noter les deux points pour empêcher l'édition de
  l'output
  %=value(%);#utilisation du dito
```

$$\left(x^2+x+1\right)\Big|_{x=a} = a^2+a+1$$

Evaluation à valeurs multiples

Quand une expression dépend de plusieurs variables, l'évaluation peut être menée par assignation ou par la commande **eval**. Cette dernière est préférable. En premier argument, on donne l'expression à évaluer. En second argument, on place entre accolades des égalités séparées par des virgules, chacune d'elles indiquant la valeur attribuée à une variable.

```
> restart;
  P:=x^2+y^2+z^2;#assignation d'un polynôme à 3 variables à
  l'identificateur P
  Eval(P,{x=1,y=0,z=4})=eval(P,{x=1,y=0,z=4});
  P;x;y;z;#le contenu des différentes variables informatiques
  n'est pas modifié
```

$$P := x^2 + y^2 + z^2$$

$$\left(x^2 + y^2 + z^2\right)\Big|_{x=1,\, y=0,\, z=4} = 17$$

$$x^2 + y^2 + z^2$$

$$x$$

$$y$$

$$z$$

Substitution

La substitution est une opération fréquente en informatique. On vient d'ailleurs d'en voir de nombreux exemples puisque l'évaluation d'une expression pour une valeur particulière débute nécessairement par le remplacement de la variable libre par cette valeur particulière. Mais d'autres substitutions sont concevables, sinon obligatoires pour mener à bien certaines taches, en particulier des simplifications. Maple propose plusieurs modalités de substitution.

Substitution par **eval**

La commande **eval(Xp,x=a)** permet de remplacer dans l'expression **Xp** toutes les occurrences de **x** par la valeur ou même l'expression **a.**

```
> restart;
  Xp:=2*x^4+cos(x-Pi)+c/z;#le polynôme est identifié à Xp
  Eval(Xp,x=a)=eval(Xp,x=a);#évaluation pour x=a
  Eval(Xp,x=y*z)=eval(Xp,x=y*z);#évaluation pour x=y*z
```

$$Xp := 2\,x^4 - \cos(x) + \frac{c}{z}$$

$$\left(2\,x^4 - \cos(x) + \frac{c}{z}\right)\Big|_{x=a} = 2\,a^4 - \cos(a) + \frac{c}{z}$$

$$\left(2\,x^4 - \cos(x) + \frac{c}{z}\right)\Big|_{x=y\,z} = 2\,y^4\,z^4 - \cos(y\,z) + \frac{c}{z}$$

Cette commande permet d'effectuer des corrections.

```
> consom:=c*Y+C0;#la consommation dépend du revenu
  consom:=eval(consom,Y=Y-T);#repentir! la consommation dépend
  du revenu disponible
```

$$consom := c\,Y + C0$$

$$consom := c\,(Y - T) + C0$$

Plus fort : remplacer un nombre par un autre ...

```
> eval(Xp,2=5);
```

$$5\,x^4 - \cos(x) + \frac{c}{z}$$

De plus en plus fort : pratiquer une substitution multiple au moyen d'accolades.

```
> eval(Xp,{2=5,cos=exp,4=log(z)});
```

$$5\,x^{\ln(z)} - e^x + \frac{c}{z}$$

Substitution par `subs`

Tout comme **`eval`**, la commande **`subs`** effectue une ou plusieurs substututions dans une expression, mais, à la différence de **`eval`**, **`subs`** n'effectue pas d'évaluation automatique.

```
> restart;
  P:=x^2+x+2*y-sin(Pi*a);
  eval(P,a=2);#eval substitue et évalue
  subs(a=2,P);#subs substitue sans évaluer
```

$$P := x^2 + x + 2\,y - \sin(\pi\,a)$$

$$x^2 + x + 2\,y$$

$$x^2 + x + 2\,y - \sin(2\,\pi)$$

On notera l'ordre des arguments de la commande **`subs`**, qui est l'inverse de celui de **`eval`**. Il faut avoir en tête la phrase mnémotechnique : **remplacer `x` par `a` dans l'expression `P`.**

La substitution multiple est possible, de préférence en plaçant les égalités définissant les remplacements entre accolades, faute de quoi Maple opère séquentiellement les substitutions de la gauche vers la droite. Dans l'exemple qui suit, les résultats sont tous différents parce que l'ordre des substitutions n'est pas le même :

```
> subs(a=2,2=3,y=2*exp(z),P);
  subs(y=2*exp(z),a=2,2=3,P);
  subs(y=2*exp(z),2=3,a=2,P);
  subs({a=2,2=3,y=2*exp(z)},P);#les accolades forcent Maple à
  effectuer les substitutions jusqu'à épuisement des
  possibilités
```

$$x^3 + x + 6\,e^z - \sin(3\,\pi)$$

$$x^3 + x + 4\,e^z - \sin(3\,\pi)$$

$$x^3 + x + 4\,e^z - \sin(2\,\pi)$$

$$x^3 + x + 6\,e^z - \sin(2\,\pi)$$

Substitution par subsop

La commande **subsop** (pour **subs**titution d'**op**érandes) remplace un opérande de premier niveau (à ce sujet, voir le chapitre consacré aux expressions) par une expression. Sa syntaxe est la même que celle de **subs.**

```
> P;
  op(P);#demande des opérandes de niveau 1
  subsop(4=cos(a),P);#remplacement du 4° opérande de
  l'expression P par cos(a)
```

$$x^2 + x + 2\,y - \sin(\pi\,a)$$

$$x^2,\ x,\ 2\,y,\ -\sin(\pi\,a)$$

$$x^2 + x + 2\,y + \cos(a)$$

Substitution par algsubs

En pratique, on a souvent besoin de remplacer une partie d'expression algébrique par une autre à des fins de simplification. Par exemple, supposons qu'on veuille remplacer l'expression du revenu d'équilibre keynésien $\frac{C+I+G}{1-c-c\,t}$ par $k\ (C+I+G)$ afin de mettre en valeur le multiplicateur de demande autonome. Les commandes précédentes, **eval**, **subs**, **subsop**, permettent de remplacer $\frac{1}{1-c-c\,t}$ par k parce que c'est un opérande de premier niveau

```
> Ye:=(C+I+G)/(1-c-c*t);#l'identificateur Ye contient
  l'expression du revenu d'équilibre keynésien
  eval(Ye,1/(1-c-c*t)=k);#eval accepte la substitution
  subs(1/(1-c-c*t)=k,Ye);#subs accepte la substitution
  subsop(2=k,Ye);#subsop accepte la substitution
```

$$Ye := \frac{C+\mathrm{I}+G}{1-c-c\,t}$$

$$(C+\mathrm{I}+G)\,k$$

$$(C+\mathrm{I}+G)\,k$$

$$(C+\mathrm{I}+G)\,k$$

Mais ces trois commandes refusent toute substitution dès que la sous-expression à modifier contient plusieurs opérandes de premier niveau. Par exemple, dans l'expression $x^2 + x + 2\,y - \sin(\pi\,a)$, on ne peut pas remplacer $x^2 + x$ par h parce que cette sous-expression algébrique est la somme de deux opérandes de premier niveau.

```
> eval(P,x^2+x=h);#eval refuse la substitution
  subs(x^2+x=h,P);#subs refuse la substitution
  subsop(1+2=h,P);#subsop refuse la substitution
```

$$x^2 + x + 2\,y - \sin(\pi\,a)$$

$$x^2 + x + 2\,y - \sin(\pi\,a)$$

$$x^2 + x + h - \sin(\pi\,a)$$

Heureusement, il existe une parade : la commande **algsub** (**alg**ebraic **sub**stitution) permet de

remplacer une expression, **et une seule à la fois**, par une autre.

```
> algsubs(x^2+x=h,P);
```

$$h + 2\,y - \sin(\pi\,a)$$