# SIMPLIFICATION

**Bernard Dupont**

Bernard.Dupont@univ-lille1.fr

Simplifier une expression mathématique est un processus consistant à rendre le plus lisible possible une expression. Cette opération est d'autant plus utile que les réponses renvoyées par le logiciel au cours d'une session sont souvent indigestes, tout particulièrement les résultats d'un **solve** ou d'un **dsolve**. La lisibilité dépend des intentions de l'auteur qui renvoient à des considérations objectives d'économie et de simplicité mais aussi à des préférences subjectives. Maple ne peut évidemment pas satisfaire tous les utilisateurs dans ce domaine puisqu'il n'existe aucun principe établi donnant une condition nécessaire ou suffisante de la simplification universelle. Au mieux, il existe des techniques permettant de transformer des expressions en expressions équivalentes plus appropriées aux desiderata de l'utilisateur.

Dans le domaine de la simplification, Maple propose la commande "à tout faire" **simplify**. Pour un utilisateur débutant ou un intermittent, elle suffit amplement dans le traitement de questions élémentaires et semble même magique dans des cas apparemment plus ardus.

```
> restart;
  xp:=-(cos(2*x)-cos(x)^2)/tan(x)^2;#qui maîtrise parfaitement la
  trigonométrie?
  simplify(xp);#Maple renvoie un résultat parfait...
```

$$xp := -\frac{\cos(2\,x) - \cos(x)^2}{\tan(x)^2}$$

$$\cos(x)^2$$

Pour un utilisateur régulier, elle devient vite déconcertante et paraît inutile. L'exemple le plus connu est celui de la recherche de la racine carrée ... du carré d'un nombre symbolique. A priori, on est tenté d'écrire simplement :

```
> rac:=sqrt(x^2);
```

$$rac := \sqrt{x^2}$$

Le moins que l'on puisse dire est que la réponse est décevante. Appelons à la rescousse **simplify** :

```
> simplify(rac);
```

$$\mathrm{csgn}(x)\ x$$

Le résultat affiché frise l'inconvenance. Pourtant, Maple a raison. Ignorant tout de *x*, en particulier ignorant que les économistes manipulent des variables réelles, il attribue à ce nombre symbolique les propriétés d'un élément du corps des nombres complexes et simplifie en conséquence (csgn(x) exprime ici le signe complexe - **c**omplex **sign** - du nombre *x*). Fort heureusement, les concepteurs du logiciel permettent à l'utilisateur d'introduire des hypothèses sur les variables à l'aide des instructions **assume**, **assuming** et **additionally**. Pour l'exemple précédent, poser l'hypothèse que **x** est un réel avant de simplifier permet d'obtenir la simplification recherchée.

```
> assume(x,real);
  simplify(rac);
```

$$|x\sim|$$

Le but de ce chapitre est de présenter systématiquement les possibilités offertes par **simplify** d'une part, et par **assume**, **assuming** et **additionally** d'autre part. La combinaison de ces instructions est souvent nécessaire pour obtenir un résultat convenable. C'est pourquoi on commence par exposer les principes permettant d'introduire des hypothèses sur les variables (section 1) avant d'explorer la commande de simplification et ses options (section 2).

# Poser des hypothèses

En économie, les variables et paramètres sont essentiellement des grandeurs réelles. Les nombres complexes peuvent intervenir, mais seulement au titre d'auxiliaires de calcul. Un premier moyen de forcer Maple à ne considérer que les propriétés des réels est d'utiliser le paquetage **RealDomain**. C'est radical : toutes les grandeurs manipulées sont considérées a priori comme réelles et les propriétés de nombreuses fonctions/fonctionnalités sont redéfinies.

```
> restart;
  with(RealDomain);#chargement du paquetage. L'output montre les
  fonctions affectées par le passage dans le corps des réels.
  sqrt(x^2);simplify(%);#reprise de l'exemple de l'introduction.
  L'output est satisfaisant.
  log(-1);#Maple refuse la valeur négative
  log(exp(Pi));#les propriétés des fonctions numériques à
  variable réelle sont respectées
  exp(log(Pi));
```

[ℑ, ℜ, `^`, arccos, arccosh, arccot, arccoth, arccsc, arccsch, arcsec, arcsech, arcsin, arcsinh, arctan, arctanh, cos, cosh, cot, coth, csc, csch, *eval*, exp, *expand*, *limit*, ln, log, sec, sech, signum, *simplify*, sin, sinh, *solve*, sqrt, surd, tan, tanh]

$$\sqrt{x^2}$$

$$|x|$$

*undefined*

$$\pi$$

$$\pi$$

Pour appeler temporairement le paquetage, on utilise aussi la syntaxe **use RealDomain in <instruction> end use;**

```
> restart;use RealDomain in simplify(sqrt(x^2)) end use;
```

$$|x|$$

Mais la méthode la plus utilisée et sans aucun doute la plus fiable est de recourir aux instructions **assume**, **assuming** et **additionally.** D'ailleurs, il est possible, dans une certaine mesure, de poser des hypothèses supplémentaires en y ayant recours à l'intérieur du paquetage.

```
> restart;
  with(RealDomain):
  assume(x<0);
  simplify(sqrt(x^2));
```

$$-x\sim$$

# assume

L'instruction **assume** est utilisée pour imposer une ou plusieurs contraintes sur une ou plusieurs variables. Une contrainte peut être directe, par exemple **x,real**, ou peut se faire à travers une inégalié, par exemple **x+3>0**. Dès qu'une hypothèse est formulée sur une variable, celle-ci est "marquée" lors de l'édition d'un résultat par un signe "tilde" (~) - qui se révèle incommode à l'usage car on le confond parfois avec le signe "moins" (-). Le tilde se supprime à la convenance de l'utilisateur en insérant la commande **interface(showassumed=0);** ou en réglant correctement la boîte de dialogue par **Tools/Outils→ Options→ Display/Affichage→ Assumed variables/Variables avec suppositions**.

```
> restart;
  assume(x>0);
  simplify(sqrt(x^2));
```

$$x\sim$$

```
> interface(showassumed=0);
  simplify(sqrt(x^2));
  interface(showassumed=1);#retour au réglage par défaut
```

$$1$$

$$x$$

$$0$$

L'instruction **assume** ne génére aucun affichage. On s'informe à tout moment des hypothèses concernant une variable en invoquant **about.**

```
> about(x);
Originally x, renamed x~:
  is assumed to be: RealRange(Open(0),infinity)
```

Pour libérer une variable contrainte par des hypothèses, il suffit de la désassigner.

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

$$x := x$$

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

Toute nouvelle contrainte effectuée avec **assume** annule la précédente.

```
> assume(x>0);simplify(sqrt(x^2));
  assume(x+1>0);simplify(sqrt(x^2));
```

$$x\sim$$

$$|x\sim|$$

Il est possible de faire plusieurs hypothèses avec une seule commande **assume** à condition de les séparer par des virgules et de veiller au fait que chaque terme de la suite pose une contrainte et une seule.

```
> assume(0<c<1,Y>=0);#le premier argument pose deux hypothèses
  sur le paramètre c
Error, `<` unexpected
```

```
> assume(c>0,c<1,Y>=0);#correction : remplacement de 0<c<1 par
  c>0,c<1
  is(c*Y>=0);is(c*Y<=Y);#tests à l'aide de is (est-ce que ...?)
```

*true*

*true*

Une hypothèse peut prendre deux formes : inégalité ou propriété. La syntaxe est différente.

- Une contrainte-inégalité s'exprime avec **<**, **>**, **<=** ou **>=** (tout commentaire est superflu). L'hypothèse proprement dite est une expression dépendant de la variable considérée **qui est dans ce cas toujours considérée comme un nombre réel**.

```
> assume(abs(x)<=1,abs(x)>=0);simplify(sqrt(x-1)^2);about(x);
```

$x\sim - 1$

```
Originally x, renamed x~:
  Involved in the following expressions with properties
    -abs(x) assumed RealRange(-1,0)
    abs(x) assumed RealRange(0,1)
  also used in the following assumed objects
  [-abs(x)] assumed RealRange(-1,0)
  [abs(x)] assumed RealRange(0,1)
```

- Pour donner une propriété à une variable, il faut d'abord écrire celle-ci, puis taper une virgule, enfin écrire le nom de code de la propriété.
  Les propriétés suivantes peuvent être attribuées à une variable :
  (1) un type tel que **integer**, **fraction**, **rational**, **real**, **constant**
  (2) un intervalle de valeurs, généralement **RealRange(a,b)** où **a** peut être un nombre inclus, un nombre exclu (auquel cas on précise **Open(a)**) ou **-infinity** et **b** peut être un nombre inclus, un nombre exclu (auquel cas on précise **Open(b)**) ou **infinity.**
  (3) une intersection de propriétés avec **AndProp**, par exemple **AndProp(integer, positive)**
  (4) une réunion de propriétés avec **OrProp**, par exemple **OrProp(positive, negative)**
  (5) enfin, et surtout, un nom de propriété tel que **realcons** (pour constantes réelles; raccourci de **AndProp(real,constant)** ou encore **nonneg** (pour "non négatif", raccourci de **RealRange(0,infinity)**).
  La liste complète des propriétés supportées par **assume** est donnée dans l'aide en ligne (**? property)**.

```
> assume(x,RealRange(Open(-1),Open(1)));
  is(sqrt(1-x^2)>0);
  assume(x,RealRange(-1,1));
  is(sqrt(1-x^2)>=0);
```

*true*

*true*

## additionally

On a vu que la commande **assume** permet d'injecter une hypothèse sur une variable en détruisant les hypothèses précédentes la concernant. Pour éviter ce phénomène de destruction quand on a besoin d'ajouter de nouvelles hypothèses, on utilise la commande **additionally** qui a la même syntaxe.

```
> restart;assume(x<1);is(sqrt(1-x^2)>0);
```

*false*

```
> additionally(x>-1);is(sqrt(1-x^2)>0);
```

*true*

## assuming

L'instruction **assuming** permet de faire des hypothèses temporaires, au sens où elles ne sont prises en compte que pour une seule ligne de commande. L'exemple suivant montre comment l'utiliser.

```
> restart;
  Xp:=sqrt(x^2*y^2)/(x*y);
  simplify(Xp);#Maple ne simplifie parce qu'il raisonne dans le
  corps des complexes
```

$$Xp := \frac{\sqrt{x^2 y^2}}{x y}$$

$$\frac{\sqrt{x^2 y^2}}{x y}$$

```
> simplify(Xp) assuming positive;#Maple dispose de
  renseignements sur les variables
```

1

On peut vérifier que les hypothèses sont temporaires :

```
> about(x,y);
x:
  nothing known about this object

y:
  nothing known about this object
```

L'instruction **assuming** accepte des hypothèses multiples, la syntaxe étant celle de **assume.**

```
> is(s*k^(1/3)>0) assuming s>0,s<1,k>0;
```

*true*

# Règles d'utilisation de simplify

On a vu en introduction que Maple propose la commande **simplify** pour faire des simplifications automatiques mais que ces dernières pouvaient être éloignées des résultats espérés. Examinons

encore deux situations.

```
> restart;
  simplify((x^a)^b);#exemple 1
  simplify(log(exp(x)));#exemple 2
```

$$(x^a)^b$$

$$\ln(e^x)$$

Les résultats ont de quoi faire bondir les économistes, pour qui toutes les variables sont nécessairement des nombres réels et qui exigent d'avoir $(x^a)^b = x^{a\,b}$ et $\ln(e^x) = x$. Or Maple a raison. Faute de connaître les types des variables, le logiciel suppose qu'elles sont complexes et applique les règles de calcul dans $\mathbb{C}$.

Le principe fondamental est donc que, par défaut, Maple considère les expressions qui lui sont soumises comme des fonctions à valeurs complexes. Il en résulte des règles strictes qu'on va détailler pour les fonctions les plus courantes en économie. On sera alors en mesure de les contourner pour obtenir des résultats valables dans $\mathbb{R}$. Autant dire que la simplification automatique relève d'un mythe et qu'il faut se résoudre à pratiquer la simplification assistée.

## Ressources de la commande `simplify`

Soit **Xp** est une expression qu'on veut simplifier. La commande **simplify** admet quatre syntaxes.

1) **simplify(Xp)**: le seul argument de l'instruction **simplify** est l'expression elle-même. On a déjà dit que cette utilisation naïve de la commande est souvent décevante.

2) **simplify(Xp,assume=propriété)**: la simplification tient compte d'une hypothèse attribuant une propriété partagée par toutes les variables et paramètres impliqués dans l'expression. Par exemple **assume=real** force Maple à manipuler toutes les variables symboliques comme des nombres réels.

3) **simplify(Xp,symbolic)**: l'argument **symbolic** permet de surmonter les réticences de Maple dans certains cas - principalement générés par les variables complexes. On donnera des exemples dans les paragraphes suivants montrant que cette option donne de très bons résultats quand l'expression ne contient que des variables réelles.

4) **simplify(Xp,option1,option2,..)**: plusieurs options peuvent être injectées, qu'elles soient génériques ou suggérées par l'utilisateur. Les économistes choisissent de préférence la première possibilité. En effet, les options génériques sont des routines appliquant des résultats et identités valables dans un domaine particulier, ce qui automatise dans une large mesure la simplification d'un résultat. Les options communes **trig**, **ln** (et non : **log**), **power**, **radical**, **sqrt** sont respectivement dédiés aux simplifications trigonométriques, logarithmiques, puissance, racine et racine carrée.

```
> Xp:=h*sin(x)^2+ln(2*x)+h*cos(x)^2;
  simplify(Xp, trig);#simplification sur les fonctions
  trigonométriques
  simplify(Xp, ln);#simplification sur la fonction
  logarithmique
  simplify(Xp,trig,ln);#simplification sur les fonctions
  trigonométriques puis sur la fonction logarithme
```

$$Xp := h\sin(x)^2 + \ln(2\,x) + h\cos(x)^2$$

$$h + \ln(2\,x)$$
$$h \sin(x)^2 + \ln(2) + \ln(x) + h \cos(x)^2$$
$$h + \ln(2) + \ln(x) \qquad \textbf{(2.1.1)}$$

Dans les paragraphes suivants, les fonctionnalités de **simplify** sont détaillées pour les fonctions exponentielle, logarithme, puissance et racine. Quand une limite est mise en évidence, on montre comment elle peut être surmontée en utilisant deux commandes présentées dans le chapitre précédent, à savoir **expand** et **combine.**

## Exponentielle

Pour la fonction exponentielle, **simplify** opère les transformations suivantes :

1) $e^x e^y \to e^{x+y}$ mais pas $e^{x+y} \to e^x\ e^y$

```
> exp(x)*exp(y)=simplify(exp(x)*exp(y));
  exp(x+y)=simplify(exp(x+y));#aucune transformation
```

$$e^x e^y = e^{x+y}$$
$$e^{x+y} = e^{x+y}$$

La simplification dans le sens $e^{x+y} \to e^x\ e^y$ se fait avec **expand.**

```
> exp(x+y)=expand(exp(x+y));
```

$$e^{x+y} = e^x e^y$$

2) $\frac{1}{e^x} \to e^{-x}$ et plus généralement $\frac{1}{e^{a\,x}} \to e^{-a\,x}$

```
> 1/exp(x)=simplify(1/exp(x));
  1/exp(a*x)=simplify(1/exp(a*x));
```

$$\frac{1}{e^x} = e^{-x}$$
$$\frac{1}{e^{a\,x}} = e^{-a\,x}$$

La simplification dans l'autre sens utilise **expand.**

```
> exp(-x)=expand(exp(-x));
  exp(-a*x)=expand(exp(-a*x));
```

$$e^{-x} = \frac{1}{e^x}$$
$$e^{-a\,x} = \frac{1}{e^{a\,x}}$$

3) $(e^x)^n \to e^{n\,x}$ pour $n$ entier naturel :

```
> simplify(exp(x)^5);
  assume(n,integer);simplify(exp(x)^n);#il faut au préalable
  indiquer que n est un entier
```

$$e^{5\,x}$$

$$e^{n\sim x}$$

La simplification dans l'autre sens utilise **expand** quand $n$ est un nombre explicité.

```
> expand(exp(5*x));
  expand(exp(x*n));
```

$$\left(e^{x}\right)^{5}$$

$$e^{n\sim x}$$

## Logarithme

Pour la fonction logarithme, les transformations suivantes sont effectuées :

1) **$\ln\left(e^{x}\right) \rightarrow x$ si $x$ est un réel.**

```
> simplify(log(exp(x)));#x est a priori un nombre complexe.
  Maple ne simplifie pas.
  assume(x::real);log(exp(x));#x est déclaré comme réel. Maple
  simplifie.
  x:='x':#levée de l'hypothèse
  simplify(log(exp(x)),assume=real);#utilisation de l'argument
  assume dans la commande
```

$$\ln\left(e^{x}\right)$$

$$x\sim$$

$$x$$

2) $\ln(x\,y) \rightarrow \ln(x) + \ln(y)$ si $x$ est un réel positif et $\ln(x\,y) \rightarrow \ln(-x) + \ln(-y)$ si $x$ est un réel négatif.

```
> simplify(log(2*x));
  simplify(log(-2*x));
```

$$\ln(2) + \ln(x)$$

$$\ln(2) + \ln(-x)$$

```
> assume(a>0);simplify(log(a*x));
  assume(a<0);simplify(log(a*x));
```

$$\ln(a\sim) + \ln(x)$$

$$\ln(-a\sim) + \ln(-x)$$

La transformation dans le sens contraire utilise **combine** (à condition que les fonctions soient bien définies).

```
> combine(log(2)+log(b));
  assume(a>0);combine(log(a)+log(b));
```

$$\ln(2\,b)$$

$$\ln(a\sim b)$$

3) $\ln\left(x^{y}\right) \rightarrow y\ln(x)$ si $x$ est un réel positif et $y$ est réel
$\ln\left(x^{y}\right) \rightarrow y\ln(-x)$ si $x$ est un nombre négatif et $y$ est un entier pair
$\ln\left(x^{y}\right) \rightarrow y\ln(x)$ si $x$ est un nombre négatif et $y$ est un entier impair
$\ln\left(x^{y}\right) \rightarrow y\ln(x)$ si $x$ est un réel et $y$ est un entier impair

$\ln(x^y) \rightarrow \frac{y \ln(x^2)}{2}$ si $x$ est un réel et $y$ est un entier pair

Si on veut forcer la transformation $\ln(x^y) \rightarrow y \ln(x)$ pour n'importe quelle valeur de $x$ et de $y$, on placera en argument l'option **symbolic**. Le principe est le même pour la transformation $\ln(x\,y)$ -> $\ln(x) + \ln(y)$ .

```
> log(k^alpha)=simplify(log(k^alpha),symbolic);
  log(L^alpha*K^beta)=simplify(log(L^alpha*K^beta),symbolic);
```

$$\ln(k^{\alpha}) = \alpha \ln(k)$$

$$\ln(L^{\alpha} K^{\beta}) = \alpha \ln(L) + \beta \ln(K)$$

La transformation inverse utilise **combine**, mais il faut tâtonner pour trouver son bonheur. L'aide en ligne est importante pour maîtriser les opérations. L'exemple suivant montre comment maénipuler une fonction de production Cobb-Douglas.

```
> assume(L>0,K>0);
  2/3*log(L)=combine(2/3*log(L));
  2/3*ln(L)+1/3*ln(K)=combine(2/3*ln(L)+1/3*ln(K));#cas
  "numérique"
  alpha*log(L)+beta*log(K)=combine(alpha*log(L)+beta*log(K),ln,
  anything,symbolic);#cas "symbolique"
```

$$\frac{2}{3} \ln(L\sim) = \ln(L\sim^{2/3})$$

$$\frac{2}{3} \ln(L\sim) + \frac{1}{3} \ln(K\sim) = \ln(L\sim^{2/3} K\sim^{1/3})$$

$$\alpha \ln(L\sim) + \beta \ln(K\sim) = \ln(L\sim^{\alpha} K\sim^{\beta})$$

## Puissance et racine

Pour les fonctions puissance et racine, les transformations suivantes sont effectuées :

1) $x^y x^z \rightarrow x^{y+z}$ pour tout $x$, tout $y$ et tout $z$ réels :

```
> L:='L';#désassignation de L
  L^(alpha)*L^(1-alpha);%=simplify(%);
```

$$L := L$$

$$L^{\alpha} L^{1-\alpha}$$

$$L^{\alpha} L^{1-\alpha} = L$$

La transformation en sens contraire se fait avec **expand** :

```
> L^(alpha+beta)=expand(L^(alpha+beta));
```

$$L^{\alpha+\beta} = L^{\alpha} L^{\beta}$$

2) $(x^y)^z \rightarrow x^{yz}$ pour tout $x$, tout $y$ et tout $z$ réels demande une syntaxe spéciale si les exposants sont symboliques :

```
> (x^y)^z=simplify((x^y)^z,symbolic);#on force la
  simplification avec l'option symbolic
```

$$(x^y)^z = x^{yz}$$

Ce n'est pas nécessaire si l'un au moins des exposants est chiffré :

```
> simplify((x^(1/2))^(y/3));
```

$$x^{\frac{1}{6}y}$$

3) $\left(\frac{x}{y}\right)^z \rightarrow x^z y^{-z}$ pour tout $x$, tout $y$ et tout $z$ réels demande également une syntaxe spéciale si les exposants sont symboliques :

```
> (x/y)^z=simplify((x/y)^z,symbolic);#simplify est utlisé avec
  l'argument symbolic
```

$$\left(\frac{x}{y}\right)^z = x^z y^{-z}$$

Mais l'option **symbolic** n'a aucun effet sur des exposants chiffrés :

```
> restart;(x/y)^(1/4)=simplify((x/y)^(1/4),symbolic);
  (x/2)^(1/4)=simplify((x/2)^(1/4),symbolic);
```

$$\left(\frac{x}{y}\right)^{1/4} = \frac{x^{1/4}}{y^{1/4}}$$

$$\frac{1}{2}\,2^{3/4}\,x^{1/4} = \frac{1}{2}\,2^{3/4}\,x^{1/4}$$

4) $(x\,y)^z \rightarrow x^z y^z$ pour tout $x$, tout $y$ et tout $z$ réels, demande également une syntaxe spéciale si les exposants sont symboliques :

```
> (x*y)^z=simplify((x*y)^z,symbolic);
```

$$(x\,y)^z = x^z y^z$$

La transformation inverse peut se faire avec **combine** ... en faisant très attention aux conditions de validité de la transformation demandée.

```
> restart;x^(1/4)*y^(1/4)=combine(x^(1/4)*y^(1/4),radical)
  assuming positive;#Les hypothèses sur x et y sont posées avec
  assuming
```

$$x^{1/4}\,y^{1/4} = (x\,y)^{1/4}$$