

Tricks of the Trade

Edited by Paul Abbott

This is a column of programming tricks and techniques, most of which, we hope, will be contributed by our readers.

■ TraditionalForm versus StandardForm

The default input and output **FormatType** is **StandardForm**, which provides good readable two-dimensional typeset input and output and is mathematically unambiguous because it uses *Mathematica* syntax. **TraditionalForm** is an enhanced format corresponding to mathematical syntax, as far as this is consistently possible. Nevertheless, most *Mathematica* users cling stubbornly to **StandardForm**.

Since I find **TraditionalForm** to be the most attractive input and output format—and I use it in my columns—I would like to convince readers of the advantages of using **TraditionalForm**. These advantages include:

1. Matrices can be entered and are displayed in two-dimensional form without requiring **MatrixForm**.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} a & b & c \\ d & e & f \end{pmatrix}$$
$$\begin{pmatrix} a^2 + bd & ab + eb & ac + bf \\ d^2 + ac & bc + de & c^2 + df \end{pmatrix}$$

2. Derivatives can be entered in their familiar two-dimensional form.

$$\frac{dg(u, v)}{dx}$$
$$\frac{dv}{dx} g^{(0,1)}(u, v) + \frac{du}{dx} g^{(1,0)}(u, v)$$

3. Standard notation is available for limits and directional limits.

$$\lim_{x \rightarrow 0^-} \frac{\sin(x)}{|x|}$$

-1

4. Mathematical expressions are easier to read.

$$\int_0^{\frac{\pi}{2}} \frac{1}{\sqrt{1 - m \sin^2(\theta)}} d\theta$$

$K(m)$

5. Where it is not ambiguous, special functions are interpreted automatically. For example, here is the derivative of the Bessel function $J_n(z)$.

$$\frac{\partial J_n(z)}{\partial z}$$

$$\frac{1}{2} (J_{n-1}(z) - J_{n+1}(z))$$

6. Special function names and syntax correspond to those found in standard tables.

$${}_2F_1(1, 1; 2; x)$$

$$\frac{\log(1-x)}{x}$$

7. Complicated functions are easier to digest.

$$\text{MeijerG}[\{\{a\}, \{b, c\}\}, \{\{d, e\}, \{f, g, h\}\}, z]$$

$$G_{3,5}^{2,1}\left(z \left| \begin{array}{c} a, b, c \\ d, e, f, g, h \end{array} \right. \right)$$

TraditionalForm can be selected as the default input and output format using Cell ▸ Default Input FormatType and Cell ▸ Default Output FormatType.

There are, of course, issues involved in using **TraditionalForm**.

1. Because a space is (correctly) interpreted as multiplication, entering a space where it is not required leads to the following type of mistake.

$$\int \sin(x) dx$$

△General::spell1 : Possible spelling error: new symbol name "sin" is similar to existing symbol "Sin".

$$\frac{\sin x^2}{2}$$

2. It may not always be obvious how to input typeset expressions in **TraditionalForm**. Probably the easiest way is to type the expression using normal syntax, for example,

LegendreP[*l*, *m*, *x*]

and then use **Convert To** ▸ **TraditionalForm** (under the **Cell** menu) on the cell to yield

$P_l^m(x)$

Note that this expression is not just a *P* with a subscript *l* and superscript *m*. Unformatting the expression (using **Show Expression** under the **Format** menu) reveals **LegendreP** in a **TagBox** at its heart.

```
Cell [BoxData [
  FormBox [
    RowBox [{
      SubsuperscriptBox [
        TagBox ["P",
          LegendreP], "l", "m"], "(" , "x", ")" }], TraditionalForm]],
  "InputOnly"]
```

Finally, it is possible to include a palette that automatically switches all input and output cells in a notebook from **StandardForm** to **TraditionalForm** and vice versa.

Convert to **StandardForm**

Convert to **TraditionalForm**

To produce a palette, select the above cell and then use **Generate Palette from Selection** (under the **File** menu).

□ Notation Package

It is also worthwhile to highlight the excellent **Utilities`Notation`** package (the latest version of which is available from library.wolfram.com/packages/notation). This package allows you to define arbitrary input and output syntaxes for functions of your choosing.

Consider defining a notation for mappings. For example, the function $f(x) = x^2$ may be written as the mapping $f : x \mapsto x^2$. The mapping notation corresponds directly to **Function** in *Mathematica*.

Load the *Notation* package.

```
<< Utilities`Notation`;
```

Use the Notation palette to define a *two-way* notation (indicated by \Leftrightarrow) for mappings.

```
Notation[f_ : x_ \mapsto y_ \Leftrightarrow f_ := Function[x_, y_]]
```

To make it easier to enter mappings, use the Notation palette to add an input alias for mappings to the current notebook.

```
AddInputAlias[■ : □ \mapsto □, "map"]
```

Now enter “map” templates by simply typing `ESCmapESC`.

```
■ : □ \mapsto □
```

Using the `TAB` key you can cycle through the placeholders in this template. Mappings can now be entered using ordinary mathematical notation. For example, here is the function $f(x) = x^2$.

```
f : x \mapsto x^2
```

```
f(y cos(y))
```

```
y^2 cos^2(y)
```

Also, because we have used a *two-way* notation, conversion to and from ordinary mathematical notation and *Mathematica* notation is possible. For example, define the function $g(x) = x \cos(x^2)$ using **Function**.

```
g := Function[x, x Cos[x^2]]
```

Then use **Convert To** \triangleright **TraditionalForm** (under the **Cell** menu) on the cell.

```
g : x \mapsto x cos(x^2)
```

You can recover the original **Function** form using **Convert To** \triangleright **StandardForm**.

```
g := Function[x, x Cos[x^2]]
```

■ Using CylindricalAlgebraicDecomposition for Optimization

Frank J. Kampas

fkampas@msn.com

The functions **ConstrainedMin**, **ConstrainedMax**, and **LinearProgramming** can find a minimum or maximum of a linear objective function subject to linear constraints. The function **Experimental`Minimize** can find a minimum of a rational polynomial function subject to rational polynomial constraints, and **NumericalMath`NMinimize** and **NumericalMath`NMaximize** can find a minimum or maximum of more general functions subject to more general constraints. None of these functions can find all the minima or maxima of an optimization problem. However, **Experimental`CylindricalAlgebraicDecomposition** can find all the minima and maxima of rational polynomial objective functions subject to rational polynomial constraints, although the time required may become prohibitively long for more than simple cases.

First, consider the trivial example of maximizing $x + y$ subject to $x + y \leq 1$, $x \geq 0$, and $y \geq 0$.

ConstrainedMax gives one solution. (Note that the linear programming functions **ConstrainedMin** and **ConstrainedMax** assume that all variables are greater than or equal to 0. Although **LinearProgramming** assumes nonnegativity by default, it can work with negative variables in Version 4.2.)

```
ConstrainedMax[x + y, {x + y ≤ 1}, {x, y}]
```

```
{1, {x → 1, y → 0}}
```

Reformulating this problem as a set of equalities and inequalities and using

CylindricalAlgebraicDecomposition gives all the maxima as the line $y = 1 - x$ for $0 \leq x \leq 1$ when the objective function has its maximum value of 1.

```
<< Experimental`
```

```
CylindricalAlgebraicDecomposition({obj == x + y, x + y ≤ 1, x ≥ 0, y ≥ 0}, {obj, x, y})
```

```
obj == 0 ∧ x == 0 ∧ y == 0 ∨ 0 < obj ≤ 1 ∧ 0 ≤ x ≤ obj ∧ y == obj - x
```

```
% /. obj → 1
```

$$0 \leq x \leq 1 \wedge y == 1 - x$$

Note that putting the objective function first in the list of variables causes the results to be displayed in order of increasing values of the objective function.

Next consider the nonlinear problem of minimizing $x y$ subject to $x^2 + y^2 \leq 1$. **Minimize** gives one minimum whereas it is clear from symmetry ($x \leftrightarrow y$) that there should be two.

```
Timing[Minimize(x y, {x^2 + y^2 ≤ 1}, {x, y})]
```

$$\{0.06 \text{ Second}, \left\{ -\frac{1}{2}, \left\{ y \rightarrow -\frac{1}{\sqrt{2}}, x \rightarrow \frac{1}{\sqrt{2}} \right\} \right\}$$

CylindricalAlgebraicDecomposition provides the two minima and the two maxima as well as a lot of information (here suppressed) about the function between the minima and the maxima, at the price of taking considerably longer than **Minimize**.

```
Timing[CAD = CylindricalAlgebraicDecomposition[{obj == x y, x^2 + y^2 ≤ 1}, {obj, x, y}]; //
First
```

```
0.42 Second
```

```
First[CAD]
```

$$\text{obj} == -\frac{1}{2} \wedge \left(x == -\frac{1}{\sqrt{2}} \wedge y == \frac{1}{\sqrt{2}} \vee x == \frac{1}{\sqrt{2}} \wedge y == -\frac{1}{\sqrt{2}} \right)$$

This problem can be overcome by first using **Minimize** to find the minimum value of the objective function and then using **CylindricalAlgebraicDecomposition** to find all the solutions for that value of the objective function. The function **multipleMinimize** implements this approach.

```
multipleMinimize[obj_, cons_, vars_] :=
Block[{soln = Minimize[obj, cons, vars], objfunc},
CylindricalAlgebraicDecomposition[
Join[{objfunc == obj == soln[[1]]}, cons], Join[{objfunc}, vars]] /. objfunc → obj]
```

Timing[multipleMinimize(x y, {x² + y² ≤ 1}, {x, y})]

$$\left\{0.07 \text{ Second, } x y == -\frac{1}{2} \wedge \left(x == -\frac{1}{\sqrt{2}} \wedge y == \frac{1}{\sqrt{2}} \vee x == \frac{1}{\sqrt{2}} \wedge y == -\frac{1}{\sqrt{2}} \right) \right\}$$

Note that **Experimental`Infimum** is likely to be a little faster than **Experimental`Minimize** when all constraints are strict inequalities.

multipleMinimize also works for linear programming problems.

Timing[multipleMinimize(-(x + y), {x + y ≤ 1, x ≥ 0, y ≥ 0}, {x, y})]

$$\{0.01 \text{ Second, } -x - y == -1 \wedge 0 \leq x \leq 1 \wedge y == 1 - x\}$$

Finally, here is an example of finding a parametrized minimizing set.

Timing[multipleMinimize((x² + y² - 1)², {x + y ≤ 1}, {x, y})]

$$\left\{0.06 \text{ Second, } (x^2 + y^2 - 1)^2 == 0 \wedge \left(x == -1 \wedge y == 0 \vee \right. \right. \\ \left. \left. -1 < x \leq 0 \wedge \left(y == -\sqrt{1 - x^2} \vee y == \sqrt{1 - x^2} \right) \vee 0 < x \leq 1 \wedge y == -\sqrt{1 - x^2} \right) \right\}$$

■ Anagrams

Roger Germundsson

roger@wolfram.com

It is straightforward to produce all the anagrams of a given string.

```
Anagrams[s_] := Union[StringJoin /@ Permutations[Characters[ToLowerCase[s]]];
```

```
Anagrams["Brad"]
```

```
{abdr, abrd, adbr, adrb, arbd, ardb, badr, bard, bdar, bdra, brad,
brda, dabr, darb, dbar, dbra, drab, drba, rabd, radb, rbad, rbda, rdab, rdab}
```

The Version 4 notebook interface includes spell checking and hyphenation (see www.wolfram.com/products/mathematica/history.html). You can call on the extensive dictionary, using **NotebookGetMisspellingsPacket**, to filter human language anagrams.

```
GetMisspelledWords[text_, language_ : "English"] := Module[{nb, result},
  nb = NotebookPut[Notebook[{Cell[text, "Text", LanguageCategory → NaturalLanguage,
    DefaultNaturalLanguage → language], Visible → False}];
  LinkWrite[$ParentLink, NotebookGetMisspellingsPacket[nb]];
  result = LinkRead[$ParentLink];
  NotebookClose[nb, Interactive → False]; result]
```

Because `NotebookGetMisspellingsPacket` works with notebooks, the text to be filtered is put into an invisible notebook, which is closed after the spelling errors have been determined. Now we can produce all the human language anagrams of a given string.

```
CorrectWords[wl_List, language_ : "English"] :=
  Complement[wl, GetMisspelledWords[ToString[Infix[wl, " "], language]]];
```

```
HumanAnagrams[s_, l_ : "English"] := CorrectWords[Anagrams[s], l];
```

```
HumanAnagrams["Brad"]
```

```
{bard, brad, drab}
```

International dictionaries, available at store.wolfram.com/view/app/dictionaries, allow one to produce human language anagrams in a range of languages.

■ Apropos

Lou D'Andria

lou@wolfram.com

Sometimes searching by name is not enough.

```
Names["*divisor*", IgnoreCase → True]
{Divisors, DivisorSigma}
```

Related functions such as `GCD` are missed by such a search. **Apropos** (Oxford English Dictionary: *adj.* having direct reference to the matter in hand) finds all directly relevant functions by searching through usage messages.

```
Apropos[str_String] := Apropos[str, $ContextPath]
```

```
Apropos[str_String, {cnt__String}] :=
  Select[Flatten[(Names[#1 <> "*" ] &) /@ {cnt}],
    (StringQ[#1] && StringMatchQ[#1, "*" <> str <> "*",
      IgnoreCase -> True] &) [ToExpression[#1 <> "::usage"]] &];
```

```
Apropos["divisor"]

{Divisors, DivisorSigma, ExtendedGCD, GCD, PolynomialGCD}
```

As a second example, we use **Apropos** to find all search functions in the *Combinatorica* package.

```
<< "DiscreteMath`Combinatorica`"
```

```
Apropos["search", {"DiscreteMath`Combinatorica`"}]

{Backtrack, BinarySearch, BreadthFirstTraversal, VertexColoring}
```

■ *J/Link* and LiveGraphics3D

Dale Horton

daleh@wolfram.com

With *J/Link* you can invoke Java applets directly from *Mathematica*. We load *J/Link* and install.

```
<< JLink` ;
InstallJava[];
```

A particularly nice applet for interactive visualization is the LiveGraphics3D applet. First, download the applet from

www.vis.informatik.uni-stuttgart.de/~kraus/LiveGraphics3D/download.html. Then, put it on the CLASSPATH for *J/Link*. The directory given by the following evaluation is recommended.

```
ToFileName[{$UserAddOnsDirectory, "Applications", "Live3D", "Java"}]

/Users/paul/Library/Mathematica/Applications/Live3D/Java/
```

This directory is on the Java class path.

JavaClassPath[]

```
{/Applications/Mathematica 4.2.app/SystemFiles/Java,  
 /Users/paul/Library/Mathematica/Applications/Live3D/Java/  
 /Users/paul/Library/Mathematica/Applications/Live3D/Java/live.jar,  
 /Applications/Mathematica 4.2.app/SystemFiles/Java/Darwin/lib/tools.jar}
```

The following code exports 3D graphics to the LiveGraphics3D applet.

```
Attributes[ShowLive] = {HoldAll};
```

```
ShowLive[gr_] := Block[{$DisplayFunction = Identity}, InstallJava[]; showlive[gr]]
```

```
showlive[gr_Graphics3D] :=  
 AppletViewer["Live", {"INPUT=" <> ToString[InputForm[N[gr]]}]
```

```
showlive[gr_SurfaceGraphics] := showlive[Graphics3D[gr]]
```

```
showlive[_] := $Failed
```

Entering the following command opens an interactive visualization window under *J/Link*.

```
ShowLive[Plot3D[sin(x + y), {x, 0,  $\pi$ }, {y, 0,  $\pi$ }, PlotPoints  $\rightarrow$  30, Mesh  $\rightarrow$  False, Axes  $\rightarrow$  False]]
```

Paul Abbott

Department of Physics

University of Western Australia

35 Stirling Highway

Crawley WA 6009, Australia

tmj@physics.uwa.edu.au

physics.uwa.edu.au/~paul