Classroom Tips and Techniques: Gems 16-20 from the Red Book of Maple Magic

Robert J. Lopez
Emeritus Professor of Mathematics and Maple Fellow
Maplesoft

Introduction

The year 2011 started with three articles containing "gems" gleaned from what started as a small red notebook. These notes were bits of Maple "magic" collected in my interactions with Maple and the Maplesoft programmers. In particular, the articles were Gems, More Gems, and Yet more Gems (from the Little Red Book of Maple Magic). Each article had five "gems" so this month's article picks up the thread with gems 16 through 20. And, by the way, these notes have now migrated to a third home, which while still red, is no longer "little."

Gem 16 - Frenet Vectors and Assumptions

The Frenet-Serret frame of orthonormal vectors along a curve are immediately obtained with the *VectorCalculus* command **TNBFrame**. For example, consider the calculations in Table 16.1.

• Tools>Load Package: VectorCalculus • Set display format for vectors	Loading VectorCalculus $BasisFormat(false)$:
• Define simple helix.	$\mathbf{R} := \langle \cos(t), \sin(t), t \rangle$:
• Apply the **TNBFrame** command.	$TNBFrame(\mathbf{R}, t)$

	$\left[\begin{array}{c} -\frac{1}{2}\sqrt{2}\sin(t) \\ \frac{1}{2}\sqrt{2}\cos(t) \\ \frac{1}{2}\sqrt{2} \end{array}\right], \left[\begin{array}{c} -\cos(t) \\ -\sin(t) \\ 0 \end{array}\right], \left[\begin{array}{c} \frac{1}{2}\sqrt{2}\sin(t) \\ -\frac{1}{2}\sqrt{2}\cos(t) \\ \frac{1}{2}\sqrt{2} \end{array}\right]$

Table 16.1 Frenet-Serret frame for simple helix

Note that the **TNBFrame** command returns *unit* tangent, principal normal, and binormal vectors. The **TangentVector**, **PrincipalNormal** and **Binormal** commands in this same package return unit vectors only with the addition of the *normalized* option. Table 16.2 illustrates this for an expanding helix.

• Define expanding helix.	$\mathbf{R} := \langle t\cos(t), t\sin(t), t\rangle$:
• Apply **TangentVector**, **PrincipalNormal**, and **Binormal** commands, the latter two requiring simplification with assumption on *t*.	$\mathbf{T} := TangentVector(\mathbf{R}, t, normalized)$: $\mathbf{N} := simplify(PrincipalNormal(\mathbf{R}, t,$ $normalized))$ assuming $real$: $\mathbf{B} := simplify(Binormal(\mathbf{R}, t,$ $normalized))$ assuming $real$:

$$\mathbf{T}, \mathbf{N}, \mathbf{B} = \left[\begin{array}{c} \frac{\cos(t) - t\sin(t)}{\sqrt{t^2+2}} \\ \frac{\sin(t) + t\cos(t)}{\sqrt{t^2+2}} \\ \frac{1}{\sqrt{t^2+2}} \end{array}\right], \left[\begin{array}{c} -\frac{3t\cos(t) + t^2\sin(t) + 4\sin(t) + t^3\cos(t)}{\sqrt{t^2+2}\sqrt{t^4+5t^2+8}} \\ \frac{-3t\sin(t) + t^2\cos(t) + 4\cos(t) - t^3\sin(t)}{\sqrt{t^2+2}\sqrt{t^4+5t^2+8}} \\ -\frac{t}{\sqrt{t^2+2}\sqrt{t^4+5t^2+8}} \end{array}\right], \left[\begin{array}{c} -\frac{2\cos(t) - t\sin(t)}{\sqrt{t^4+5t^2+8}} \\ -\frac{2\sin(t) + t\cos(t)}{\sqrt{t^4+5t^2+8}} \\ \frac{t^2+2}{\sqrt{t^4+5t^2+8}} \end{array}\right]$$

Table 16.2 **TangentVector**, **PrincipalNormal**, and **Binormal** commands applied to expanding helix

In an effort to verify orthogonality, compute the dot product of **T** with **N**.

$$\mathbf{T} \cdot \mathbf{N}$$

```
Error, (in VectorCalculus:-DotProduct) cannot combine two rooted vectors with different points of origin
```

What has happened? Associated with each vector in the Frenet frame is its point of contact (root point) on the curve. To see this, apply the **GetRootPoint** command, but first, modify the interface to show the tilde associated with assumed variables. (The showassumed parameter can have the value 0, 1, or 2, depending on whether no annotation, the tilde, or a proviso is displayed.) See Table 16.3.

$interface(showassumed = 1)$:		
$GetRootPoint(\mathbf{T}) =$	$GetRootPoint(\mathbf{N}) =$	$GetRootPoint(\mathbf{B}) =$

$\begin{bmatrix} t\cos(t) \\ t\sin(t) \\ t \end{bmatrix}$	$\begin{bmatrix} t\sim\cos(t\sim) \\ t\sim\sin(t\sim) \\ t\sim \end{bmatrix}$	$\begin{bmatrix} t\sim\cos(t\sim) \\ t\sim\sin(t\sim) \\ t\sim \end{bmatrix}$

Table 16.3 Root points for **T**, **N**, and **B**

Ordinarily, when the **assume** command is used to make a (permanent) assumption on a variable, a following tilde is displayed to indicate that this variable is no longer the same as the variable before the assumption was made. However, when the assuming functionality is used to make a (temporary) assumption on a variable in the course of a calculation, the assumption is removed before the result is returned. (For this reason, I always use "assuming" and never use the **assume** command.)

When a *VectorCalculus* Vector is created under the shadow of an "assuming" clause, that vector has an associated module that encodes the properties of the ambient vector space. When assuming exists, assumptions on terms inside this module cannot be "cleaned up" prior to output. Hence, both **N** and **B** have assumptions attached to *t* in their root points, but **T** does not, and the dot product fails because the root points are seen as different.

Consequently, for **N** or **B**, separate the calculation from the simplification, as shown in Table 16.4.

• Obtain unsimplified, but unit **N** and **B** vectors. • • Simplify with assumption that *t* is real in separate step.	**tempN** := *PrincipalNormal*(**R**, *t*, *normalized*) : **tempB** := *Binormal*(**R**, *t*, *normalized*) : **N** := *simplify*(**tempN**) assuming *real* : **B** := *simplify*(**tempB**) assuming *real* :
$GetRootPoint(\mathbf{N}) = \begin{bmatrix} t\cos(t) \\ t\sin(t) \\ t \end{bmatrix}$	$GetRootPoint(\mathbf{B}) = \begin{bmatrix} t\cos(t) \\ t\sin(t) \\ t \end{bmatrix}$

Table 16.4 Obtaining unit Frenet vectors free of entangling assumptions

The principal normal now has a root point free of the entangling assumption, so the calculation $simplify(\mathbf{T}\cdot\mathbf{N}) = 0$ succeeds.

Gem 17 - $\mathcal{L}$ and $\mathcal{L}^{-1}$ as Laplace Transform Operators

In some texts, the Laplace transform of the function $f(t)$ is denoted with a decorated f such as $\widehat{f}$ or $\overline{f}$. Some texts use upper and lower case letters to distinguish between function and transform, with no standardization on which case denotes what. I have always used $F(s)$ for the transform because it is simplest to type.

Some texts use some form of script "L" as the transform operator. In Maple, the choice is between $\mathcal{L}$ and $\mathfrak{L}$, neither one of which matches the character used in Ian Sneddon's *The Use of Integral Transforms* (McGraw-Hill, 1772). In some recent writing, I've used $\mathcal{L}$ (from the Script palette) for the transform operator, and $\mathcal{L}^{-1}$ for the inverse. Table 17.1 shows how to set these symbols as operators in math mode.

• Load the integral transform package.	$with(inttrans):$
• Alias $\mathcal{L}$ to laplace and $\mathcal{L}^{-1}$ to invlaplace. • Set $\mathcal{L}^{-1}$ as an Atomic Identifier.	$alias(\mathcal{L} = laplace, \mathcal{L}^{-1} = invlaplace):$
• Obtain the Laplace transform of $\sin(t)$.	$\mathcal{L}(\sin(t), t, s) = \frac{1}{s^2+1}$
• Obtain the inverse transform of $1/s^3$. • Set $\mathcal{L}^{-1}$ as an Atomic Identifier.	$\mathcal{L}^{-1}(1/s^3, s, t) = \frac{1}{2}t^2$

Table 17.1 In math mode, setting aliases for **laplace** and **invlaplace** commands

Working in "exposed" math notation, the only challenge is remembering that $\mathcal{L}^{-1}$ must be seen by Maple as a single symbol; hence, the need for setting it as an Atomic Identifier. But what about working in text mode, as when programming, or when putting the **alias** command behind an embedded component? Without using Maple's math mode, it is not immediately apparent how to implement the Atomic Identifier construct, or even set the symbol $\mathcal{L}$ from the Script palette. Table 17.2 summarizes how to discover the text-mode code for the requisite symbols.

$lprint(\mathcal{L}) =$ `` `&Lscr;` ``
$lprint(\mathcal{L}^{-1}) =$ `` `#msup(mi("&Lscr;"),mrow(mo("&uminus0;"),mn("1")))` ``

Table 17.2 Discovering the text-mode code for $\mathcal{L}$ and $\mathcal{L}^{-1}$

For $\mathscr{L}$, simply enter it from the Script palette; for $\mathscr{L}^{-1}$, set it as an Atomic Identifier. Then enter these strings into the **alias** command, as shown below. Of course, copy/paste avoids the tedium of reproducing these character strings.

```
alias(`&Lscr;` = inttrans[laplace], `#msup(mi("&Lscr;"),mrow(mo("&uminus0;"),mn("1")))` =
                                                    inttrans[invlaplace]):
```

Gem 18 - Setting an Iterated Integral from the Expression Palette

At the moment, no palette in Maple has a template for setting a definite iterated integral. Constructing the iterated integral $\int_a^b \int_c^d f(x,y)\,dy\,dx$ by iterating the single definite integral can be expedited by recently discovered "magic."

When the upper- and lower-limit fields in the outer integral have been filled, and the tab key is pressed to move to the integrand field, another click on the definite-integral template will set this template. But it will autoselect the integrand field of this new integral. This requires backing up through the fields by pressing the Shift key while tabbing. I have always found this to be needless busywork, and because of it, have repeatedly asked for double- and triple-integral templates for iterated definite integrals.

However, if after arriving at the integrand field in the first (outer) integral template, the Backspace key is pressed before the next integral template is inserted, the new template will arrive with the lower-limit field autoselected. This is because there is a hidden field-flag that is set to 3 when the integrand field of the outer integral is selected. When a new definite-integral template is inserted, it inherits the flag value of 3, and the new integrand field is autoselected. The intervening Backspace resets the flag to 1, so the new integral template arrives with the first (lower-limit) field autoselected.

I think I would still like to see double- and triple-integral templates added to the Expression palette, but meanwhile, my complaints about it aren't as intense as they used to be.

Gem 19 - Writing a Slider Value to a Label

Table 19.1 contains as Embedded Components, a Slider and a Label, with the slider value being written as the label's caption. The leading "$x =$ " and the equal sign between the slider and the label are simply typed characters.

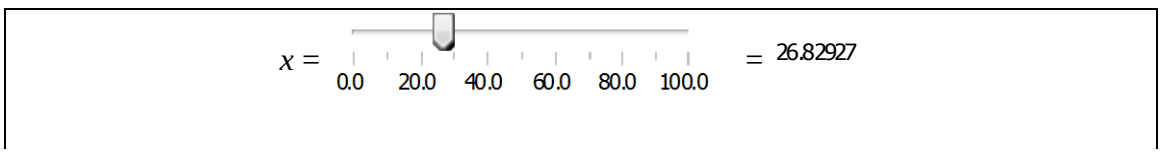

Table 19.1 Slider value written to the caption field of a Label

The code behind the Edit field of the Slider is Do(%Label0(caption) = sprintf("%.5f", Do(%Slider0)));

The **sprintf** command determines how the displayed value is to be formatted. The string "%.5f" indicates five places after the decimal. The value to be written is read from the Slider with the second **Do** command.

Gem 20 - Writing Text to a Math Container

Table 20.1 contains a simple configuration of embedded components, namely, two MathContainers, with a Button between. A function written into the left-hand container will have its derivative written to the right-hand container when the button is pressed. If no function is entered on the left, but the Derivative button is pressed anyway, the message "Please enter a function to be differentiated" is written to the MathContainer on the right.

This Gem shows how to write the text message to the MathContainer on the right.

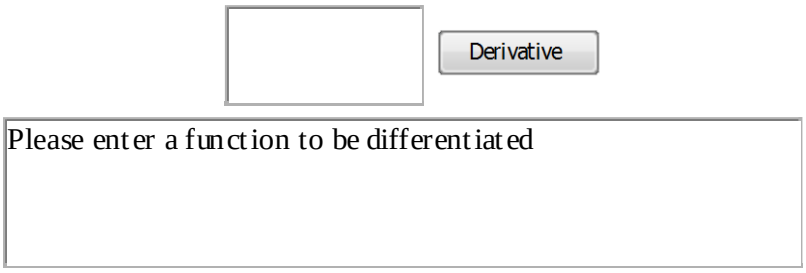

Table 20.1 The derivative of the function entered on the left is written to the container on the right

If the Button is naively coded with Do(%MathContainer1 = diff(Do(%MathContainer0),x), then pressing the Derivative button with no entry on the left generates the error message shown in Figure 20.1.

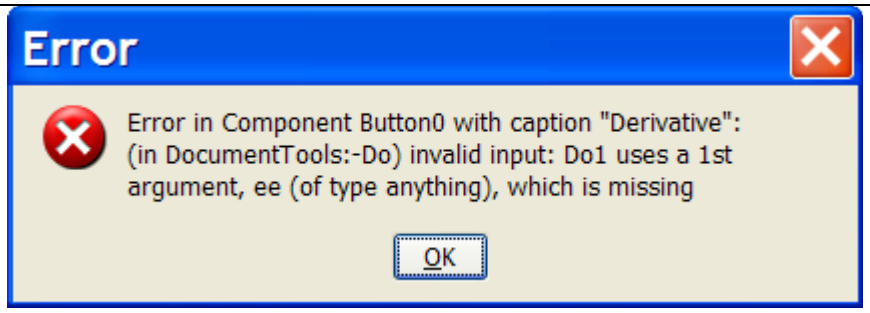

Figure 20.1 Error message naive coding can cause

Table 20.2 contains "defensive" code that catches the empty left-hand container.

Do(f = %MathContainer0);if nops([f]) = 0 then Do(%MathContainer1 = MathML[Export]("Please enter a function to be differentiated"));else Do(%MathContainer1 = diff(f,x));end if;

Table 20.2 Defensive code for detecting the left-hand container is empty

An alternative to MathML[Export]("...") is "<math><mtext>...</mtext></math>".