LuaTEX intro

Nguyen Chien

September 13, 2021

1 Why LuaTEX

LuaTEX is a great way to prepare a dynamic document. By “dynamic”, I mean a document in which the content can be changed. There are many uses for such a document. For example, a calendar layout can be filled in for different months and years. Or there are math papers that calculation exercises are filled. Or a corporate spreadsheet... Graphics are also very relevant to dynamic contents. We can use LuaTEX to systematically generate shapes for illustration.

2 Generating Texts

Lua scripts can be used to generate dynamic contents. Think of “web server pages” written in scripting languages such as PHP or Python. This can be done via the `luacode` package.

In the Lua code, printing TEX contents can be made via `tex.write` and `tex.sprint`. The latter is more flexible in that it can print multiple Lua expressions and enable line breaking using \\\\. Of course, the use of built-in Lua libraries such as `math` is possible:

```
\noindent \begin{luacode}
  tex.write("Special chars: _ ^ & $ { }")
  tex.sprint('\\\\')
  tex.write("Hashes and tildes can be escaped like this:")
  tex.write("\string# and \string~")
  tex.sprint("\\\\ Symbols: \string\\pi \string\\neq", tostring(math.pi))
\end{luacode}
```

Special chars: _ ^ & $ { }
Hashes and tildes can be escaped like this:# and ~
Symbols: $\pi \neq$3.1415926535898

We can use TEX macros, hence increase the interaction between the original TEXmaterial and the Lua scripts:

```
\newcommand\two{2}
\begin{luacode}
  tex.sprint("The square root of two is: ", math.sqrt(\two))
\end{luacode}
```

The square root of two is: 1.4142135623731

The use of `\luadirect` and `\luaexec` is also possible, especially in debugging. But for this time, the comment lines of must be prepended by TEX's `%` instead of Lua's `--`.

3 Reading data files into tables

In the following example, we read a locally-stored data file `near-earth-comets.csv` and print the content of this file:

```
% Lua code
\begin{luacode*}
function readfile(filename, n, m)
    -- Print the first n rows and m columns of a CSV data file
    count = 0
    for line in io.lines(filename) do
    items = {}
    string.gsub(line, '([^,]+)', function(item)
            table.insert(items, item)
        end)
    for j = 1, m do
            tex.sprint(items[j] .. ' & ')
        end
        tex.sprint('\\\\')
        count = count + 1
        if count == n then break end
    end
    tex.sprint('Total: ' .. count .. ' lines.')
end
\end{luacode*}

% TeX code
\newcommand{\readfile}[3]{\luadirect{readfile(#1,#2,#3)}}

\begin{tabular}{*{10}{c}}
\noindent \readfile{"near-earth-comets.csv"}{5}{3}
\end{tabular}
```

Object	Epoch	TP
1P/Halley	49400	2446467.395
2P/Encke	56870	2456618.204
3D/Biela	-9480	2390514.115
5D/Brorsen	7440	2407439.534

Total: 5 lines.

As can be seen, the use of functions can reduce TEX code by a few lines including macro definition and calling the function only (`\readfile`). Functions can receive parameters and they are implemented in TEX as macros. In addition, the functions can be pushed into separate Lua code files, thus further reduces the main TEX file's size.

4 Generating tables

By using Lua, the code logic can be developed using functions. Here, Montijano demonstrates an example of generating a trigonometric table. Rows in a number table can be conveniently implemented with a Lua's `for` loop:

```
\begin{luacode*}
function trigtable()
    for t = 0, 45, 3 do
        x = math.rad(t)
        tex.sprint(string.format('%2d° & %1.9f & %1.9f & %1.9f & %1.9f \\\\',
                    t, x, math.sin(x), math.cos(x), math.tan(x)))
    end
end
\end{luacode*}
\newcommand{\trigtable}{\luadirect{trigtable()}}

\begin{tabular}{rcccc}
\hline
& $x$ & $\sin(x)$ & $\cos(x)$ & $\tan(x)$ \\
\hline
\trigtable
\hline
\end{tabular}
```

	x	$\sin(x)$	$\cos(x)$	$\tan(x)$
0°	0.000000000	0.000000000	1.000000000	0.000000000
3°	0.052359878	0.052335956	0.998629535	0.052407779
6°	0.104719755	0.104528463	0.994521895	0.105104235
9°	0.157079633	0.156434465	0.987688341	0.158384440
12°	0.209439510	0.207911691	0.978147601	0.212556562
15°	0.261799388	0.258819045	0.965925826	0.267949192
18°	0.314159265	0.309016994	0.951056516	0.324919696
21°	0.366519143	0.358367950	0.933580426	0.383864035
24°	0.418879020	0.406736643	0.913545458	0.445228685
27°	0.471238898	0.453990500	0.891006524	0.509525449
30°	0.523598776	0.500000000	0.866025404	0.577350269
33°	0.575958653	0.544639035	0.838670568	0.649407593
36°	0.628318531	0.587785252	0.809016994	0.726542528
39°	0.680678408	0.629320391	0.777145961	0.809784033
42°	0.733038286	0.669130606	0.743144825	0.900404044
45°	0.785398163	0.707106781	0.707106781	1.000000000

5 Plotting

We use LuaTeX to plot the numerical solution of the differential equation $y'(t) = y(t)\cos\left(t+\sqrt{1+y(t)}\right)$. The main parts of the code are:

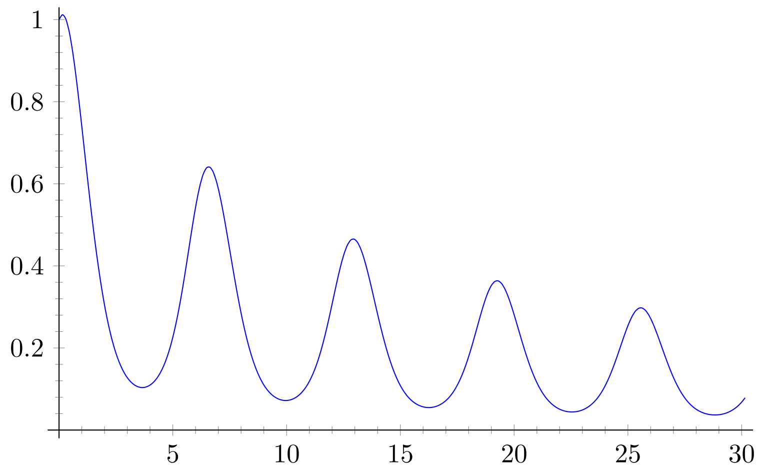
```
% the Lua part
\begin{luacode*}
-- Define function
function f(t,y)
    return y * math.cos(t+math.sqrt(1+y))
end

-- Solving the ODE equation for the plotting coordinates
function print_RKfour(tMax,npoints,option)
    local t0 = 0.0
    local y0 = 1.0
    local h = (tMax-t0)/(npoints-1)
    local t = t0
    local y = y0
    if option~=[[]] then
        tex.sprint("\\addplot["..option.."] coordinates{")
    else
        tex.sprint("\\addplot coordinates{")
    end
    tex.sprint("("..t0..","..y0..")")
    for i=1, npoints do
        k1 = h * f(t,y)
        k2 = h * f(t+h/2,y+k1/2)
        k3 = h * f(t+h/2,y+k2/2)
        k4 = h * f(t+h,y+k3)
        y = y + (k1+2*k2+2*k3+k4)/6
        t = t + h
        tex.sprint("("..t..","..y..")")
    end
    tex.sprint("}")
end
\end{luacode*}

% the macro part
\newcommand\addLUADEDplot[3][]{%
\directlua{print_RKfour(#2,#3,[[#1]])}%
}

% the plotting part
\pgfplotsset{width=0.9\textwidth, height=0.6\textwidth}
\begin{tikzpicture}
    \begin{axis}[xmin=-0.5, xmax=30.5, ymin=-0.02, ymax=1.03,
    xtick={0,5,...,30}, ytick={0,0.2,...,1.0},
    enlarge x limits=true,
    minor x tick num=4, minor y tick num=4,
    axis lines=middle, axis line style={-}
    ]
    \addLUADEDplot[color=blue,smooth]{30}{200};
    \end{axis}
\end{tikzpicture}
```

The output is:

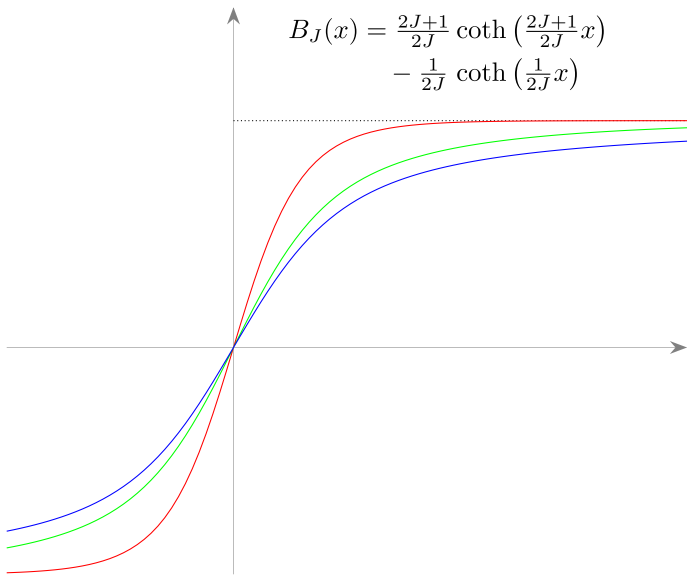


Another plotting example, featuring colour line plots and math formulae. You can clearly see the three parts: Lua code logic, macro definition and plot layout:

```
\directlua{
function coth (i)
  return math.cosh(i) / math.sinh(i)
end

function brillouin (J, x)
  if x == 0 then
    return 0
  else
   return (2*J+1)/(2*J)*coth((2*J+1)/(2*J)*x) -
        1/(2*J)*coth(1/(2*J)*x)
  end
end
}
\pgfmathdeclarefunction{Brillouin}{2}{%
  \edef\pgfmathresult{%
     \directlua{tex.print("" .. brillouin(#1,#2))}%
   }%
}

\begin{tikzpicture}[
    x                = 2cm/10,
    scale            = 3,
    axis/.style      = {help lines, -{Stealth[length = 1.5ex]}},
    brillouin/.style = {domain = -5:10, samples = 100}
  ]
  \draw [axis] (-5,0) -- (10,0);
  \draw [axis] (0,-1) -- (0,1.5);
  \draw [densely dotted] (0,{ Brillouin(1, 100)} ) -- ++(10,0);
  \draw [red]   plot [brillouin] (\x, { Brillouin(1,  \x)});
  \draw [green] plot [brillouin] (\x, { Brillouin(5,  \x)});
  \draw [blue]  plot [brillouin] (\x, { Brillouin(50, \x)});
  \node [align = center, anchor = west] at (1,1.3) {%
    $\begin{alignedat}{2}
      B_J(x) &= \tfrac{2J + 1}{2J}
                &&\coth \left ( \tfrac{2J + 1}{2J} x \right ) \\
             &\quad - \tfrac{1}{2J}
                &&\coth \left ( \tfrac{1}{2J} x \right )
     \end{alignedat}$};
\end{tikzpicture}
```



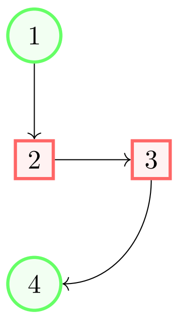
6 Generating Graphics

6.1 TikZ Introductory Examples

Here through a diagram, the basic usage of Lua is demonstrated.

```
\begin{tikzpicture}[ % Define styles
    roundnode/.style={circle, draw=green!60, fill=green!5, very thick,
                    minimum size=7mm},
    squarednode/.style={rectangle, draw=red!60, fill=red!5, very thick,
                    minimum size=5mm},
    ]
    %Nodes
    \node[squarednode]  (maintopic)    {2};
    \node[roundnode]    (uppercircle) [above=of maintopic] {1};
    \node[squarednode] (rightsquare) [right=of maintopic] {3};
    \node[roundnode]    (lowercircle) [below=of maintopic] {4};

    %Lines
    \draw[->] (uppercircle.south) -- (maintopic.north);
    \draw[->] (maintopic.east) -- (rightsquare.west);
    \draw[->] (rightsquare.south) .. controls +(down:7mm) and +(right:7mm)
            .. (lowercircle.east);
\end{tikzpicture}
```



7 Uses

- Within LyX
- Within Overleaf
- Within TeXstudio, https://tex.stackexchange.com/a/236748/208394

8 Documents

- Official guide, http://mirrors.ibiblio.org/CTAN/systems/doc/luatex/luatex.pdf
- LuaTeX Wiki, http://wiki.luatex.org/index.php/Main_Page
- In the ConTeXt environment, https://wiki.contextgarden.net/Programming_in_LuaTeX

9 Credits

- Brillouin Function by Mark Wibrow
- TikZ examples by Overleaf