ISSN 1172-496X (Print)
ISSN 1172-4234 (Online)

TECHNICAL REPORTS FROM THE ELECTRONICS GROUP
AT THE UNIVERSITY OF OTAGO

SympyTeX: Embedding symbolic computation into LaTeX documents

by

Timothy C.A. Molteno

ELECTRONICS TECHNICAL
REPORT No. 2014-1

UNIVERSITY OF OTAGO
DUNEDIN, NEW ZEALAND

Online version has
URL: http://www.physics.otago.ac.nz/reports/electronics/ETR2014-1.pdf
E-mail: tim@physics.otago.ac.nz
Address: Physics Department, University of Otago, P.O. Box 56, Dunedin, New Zealand

Electronics Group at Otago

In 1987 Millman and Grabel discarded the historical definition of 'electronics' as the science and technology of the motion of charges, preferring instead the operational definition that the primary concern of people doing electronics is *information processing*. This makes a distinction from *energy processing* practiced in the rest of electrical engineering. The act of information processing is what gets electronics practicioners invloved in the fours 'C's: communication, computation, control, and components. This practical definition seems to describe well the activities within the Electronics Group in the Physics Department at the University of Otago, and the range of topics covered in this technical report series.

In September 2013, research within the Electronics Group include projects applying inference algorithms to embedded sensors, lightweight GPS tags for birds, modelling and control of a robotic elbow, design and deployment of an under-sea glider, analysis of networks of random resistors, electrical impedance imaging, calibration of numerical models for geothermal fields using Bayesian inference, modelling and sampling of Gaussian processes, and efficient algorithms for Markov chain Monte Carlo applied to inverse problems.

Citing this Report

This report should be cited as:

> Timothy C.A. Molteno, "SympyTeX: Embedding symbolic computation into LaTeX documents", *Electronics Technical Reports* No. 2014-1, ISSN 1172-496X, May 2014.

Alternatively, using BibTeX, as:

```
@techreport{etr2014-1,
title={{SympyTeX: Embedding symbolic computation into LaTeX documents}},
author={Timothy C.A. Molteno},
series={Technical Reports from the Electronics Group at the University
of Otago},
institution={{University of Otago}},
issn={1172-496X (Print) 1172-4234 (Online)},
url={http://www.physics.otago.ac.nz/reports/electronics/ETR2014-1.pdf}
number={2014-1},
date={May 2014} }
```

SympyTeX: Embedding symbolic computation into LaTeX documents

Timothy C.A. Molteno

Abstract

SympyTeX is open-source software that allows you to embed symbolic calculations and their results into LaTeX documents. SympyTeX uses sympy, the symbolic python engine (http://www.sympy.org), to manipulate symbolic expressions. Using SympyTeX, sympy code can be embedded into your document, either hidden from view, or as part of the document, and sympy expressions can be rendered into LaTeX expressions and included in your document output.

Acknowledgment

SympyTeX builds on a lot of work by others; in particular the work of Dan Drake ddrake@member.ams.org who created the sagetex package on which SympyTeX is based. I am grateful to Alexander Steppke who pointed out how SympyTeX could be used to plot with the PGF/TikZ, and provided the example that I include in this document.

Contents

Appendix

List of Figures

Chapter 1

SympyTeX by example

The SympyTeX package allows you to embed symbolic calculations and their results (including plots) into LaTeX documents. SympyTeX uses sympy, the symbolic python engine (`http://www.sympy.org`) to manipulate symbolic expressions. SympyTeX is not limited to sympy – it allows any python code to be embedded into your document – and can incorporate maptplotlib plots, and other python computations into your document.

Using SympyTeX, code is embedded into your document, either hidden from view, or included in the document output. In addition symbolic expressions can be automatically converted into LaTeX expressions and included in your document output.

1.0.1 A first example

The integral $\int_{-1}^{1} \sqrt{1-x^2}dx$ evaluates to π. The code below uses sympy to evaluate this integral and incorporate the output into this document.

```
Using sympy within your LaTeX document is as easy as
$\sympy{2 * sympy.integrate(sympy.sqrt(1-x**2), (x, -1, 1))}$.
```

Using sympy within your LaTeX document is as easy as π.

Using the package

The document preamble must tell LaTeX that you're using the sympytex package. SympyTeX also depends on the amsmath package, so the following lines should be added to the document preamble[1].

```
\usepackage{sympytex}
\usepackage{amsmath}
```

[1] See Section B for installation instructions.

Embedding sympy code

You can use the `sympysilent` environment to execute commands. These can define variables that remain in scope. These variables can be used later in your document. For example, the following defines two variables, x and e, and places a series expansion of e into the document.

```
\begin{sympysilent}
import sympy
x = sympy.Symbol('x')
e = sympy.cos(x)
\end{sympysilent}
The series expansion of $\sympy{e
    }$ is
\[ \sympy{e} \approx
    \sympy{e.series(x, 0, 3)} \]
```

The series expansion of $\cos(x)$ is

$$\cos(x) \approx 1 - x^2/2 + \mathcal{O}\left(x^3\right)$$

You can perform integration, as in the example below. Note that the previously defined variables, x and e are still valid and so need not be redefined.

```
\begin{sympysilent}
h = sympy.integrate(1+x,x)
\end{sympysilent}
The variable $h$, how can be
    called using {\verb \sympy{h}
    }, and you will
get $h = \sympy{h}$. Similarly,
    the integral of
$1+x^4$ is $\sympy{sympy.
    integrate(1+x**4,x)}$.
```

The variable h, how can be called using `\sympy{h}`, and you will get $h = x^2/2 + x$. Similarly, the integral of $1 + x^4$ is $x^5/5 + x$.

Citing SympyTeX

If you use SympyTeX, I would be grateful if you could cite SympyTeX. The best way to do this is to cite this document [4].

```
@techreport{sympytex,
  title={{SympyTeX: Embedding symbolic computation into LaTeX documents}},
  author={T.C.A. Molteno},
  series={Electronics Technical Report},
  number={2014-1}
  year=2014
}
```

1.1 The Zhukovsky airfoil

In aerodynamics, the Zhukovsky transform [3] is used to solve for the two-dimensional potential flow around a class of airfoils known as Zhukovsky airfoils. The transform is

$$z = \zeta + \frac{1}{\zeta}$$

A Zhukovsky airfoil is generated in the z-plane by applying the Zhukovsky transform to a circle in the ζ plane.

The coordinates of the centre of the circle are variables, and varying them modifies the shape of the resulting airfoil. The circle encloses the point $\zeta = 1$ (where the derivative is zero) and intersects the point $\zeta = 1$.

This can be achieved for any allowable centre position $\mu_x + \mathrm{i}\mu_y$ by varying the radius of the circle.

```
\begin{sympysilent}
from sympy import Symbol, I, exp, sqrt
mu_x = Symbol('mu_x', real=True)
mu_y = Symbol('mu_y', real=True)
theta = Symbol('theta', real=True)
\end{sympysilent}
A circle (parametrized by $\theta$) centered at
point $\mu = \mu_x + i \mu_y$, and intersecting the point $(1,0)$ in
    the
complex plane is:
\begin{sympysilent}
mu = mu_x + I*mu_y
r = sqrt((1.0 - mu_x)**2 + mu_y**2)
zeta = mu + r*exp(I * theta)
z = (zeta + 1.0 / zeta)
\end{sympysilent}
\[ \zeta = \sympy{zeta} \]
The parametrized Zhukovsky transformation of the circle is then
\[ z = \sympy{z}. \]
```

A circle (parametrized by θ) centered at point $\mu = \mu_x + i\mu_y$, and intersecting the point $(1, 0)$ in the complex plane is:

$$\zeta = \mu_x + i\mu_y + \sqrt{\mu_y^2 + (-\mu_x + 1.0)^2}\, e^{i\theta}$$

The parametrized Zhukovsky transformation of the circle is then

$$z = \mu_x + i\mu_y + \sqrt{\mu_y^2 + (-\mu_x + 1.0)^2}\, e^{i\theta} + \frac{1.0}{\mu_x + i\mu_y + \sqrt{\mu_y^2 + (-\mu_x + 1.0)^2}\, e^{i\theta}}.$$

We can plot this with the following sympy code:

```
\begin{sympysilent}
from sympy.plotting import plot_parametric
from sympy import re, im, pi
z = z.subs([(mu_x, 0.1), (mu_y, 0.05)])
p = plot_parametric(re(z), im(z), (theta, -pi, pi), autoscale=True,
    show=False)
p.aspect_ratio = 1.0
\end{sympysilent}
The airfoil with $\mu_x=0.1, \mu_y=0.05$, looks like:

\begin{figure}
\sympyplot[width=\linewidth]{p}
\caption{Parametric plot of an airfoil with $\mu_x=0.1, \mu_y=0.05$.}
\end{figure}
```

The airfoil with $\mu_x = 0.1, \mu_y = 0.05$, looks like:

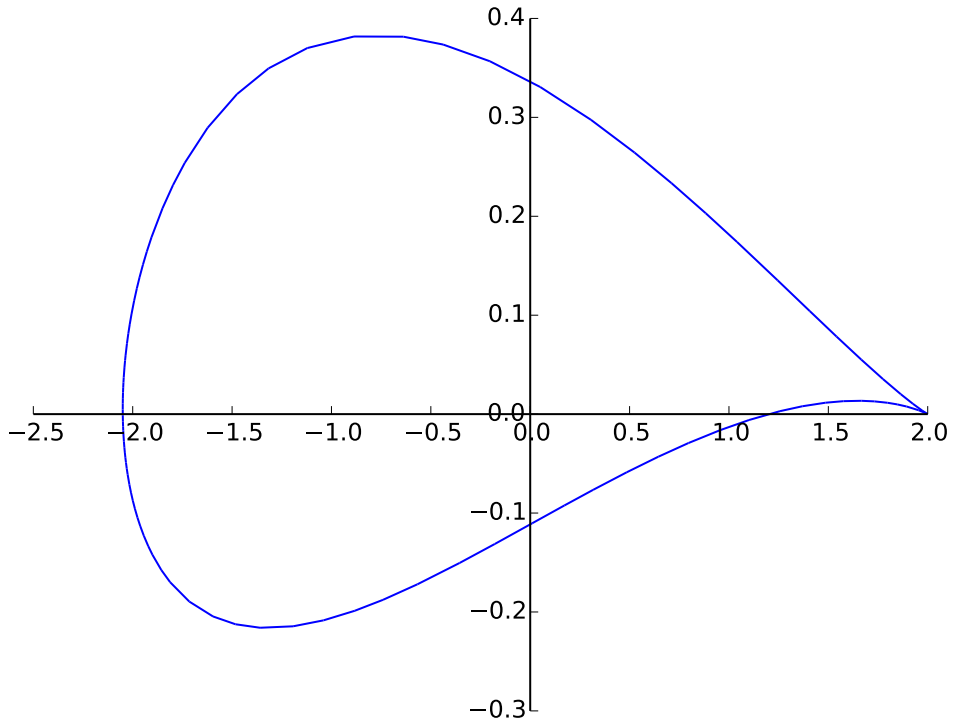


Figure 1.1. Parametric plot of an airfoil with $\mu_x = 0.1, \mu_y = 0.05$.

Chapter 2

Plotting and Graphics

SympyTeX can include graphics in your document. There are three ways to do this. The first, described in Section 2.1 describes how to incorporate plots generated directly by sympy. The second, described in Section 2.2 describes how to use matplotlib [2]. The third, described in Section 2.3, describes how to incorporate plots generated by the `pgfplots` package.

System setup

Plotting requires some python modules to be installed on your system. On Debian-based operating systems, these can be installed using the following commands.

```
sudo aptitude install python-pyglet python-matplotlib
```

The dependency on python-pyglet is not required for more recent versions of sympy.

2.1 Plotting with the sympy.plotting module

Sympy includes plotting functionality in the sympy.plotting module. Plotting works using the `sympyplot` command. This takes a sympy plot object as a parameter. The example below creates a plot object called `p`, and then plots that object inside a figure environment.

```
\begin{sympysilent}
from sympy import symbols, cos, sin
from sympy.plotting import (plot, plot_parametric)
u = symbols('u')
p = plot_parametric(cos(u), sin(u), (u, -5, 5), show=False)
\end{sympysilent}
To include the plot into your document, you can create a figure
    environment as follows
\begin{figure}
\sympyplot[width=\linewidth]{p}
\caption{A parametric plot generated by the sympy.plotting module}
\end{figure}
```

To include the plot into your document, you can create a figure environment as follows

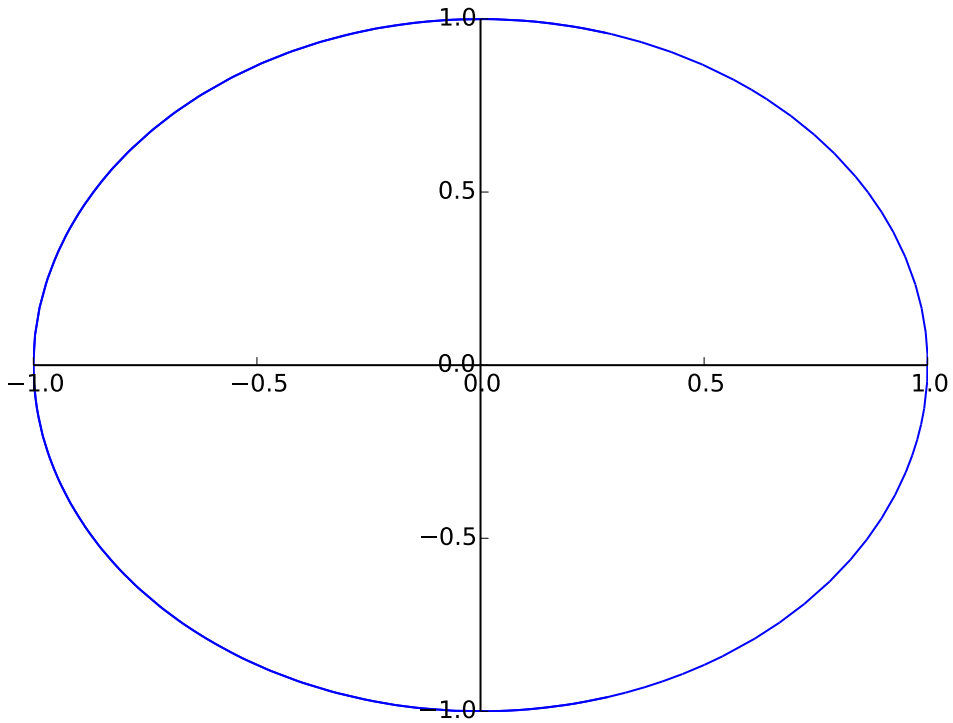


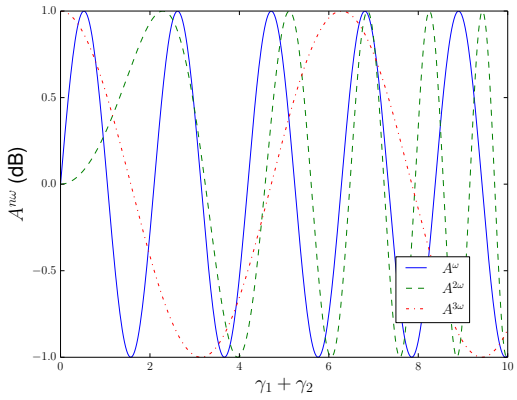
Figure 2.1. A parametric plot generated by the sympy.plotting module

2.2 Plotting with Matplotlib

Matplotlib [2] is a powerful plotting and graphics library. Matplotlib plotting works using the `sympyplot` command. This can take a matplotlib figure as a parameter. The example below does not use the sympy module at all.

```
\begin{sympysilent}
import matplotlib
import matplotlib.pyplot as plt
plt.matplotlib.rc('text', usetex = True)
import pylab
from numpy import sin, cos
fig = plt.figure()
ax = fig.add_subplot(111)
t = pylab.linspace(0,10,400)
ax.plot(t, sin(3*t), '-',
    t, sin(0.3*t**2), '--',
    t, cos(t), '-.')
ax.legend((r'$A^{\omega}$', r'$A^{2\omega}$', r'$A^{3\omega}$'),
    shadow = False, loc = (0.75, 0.1))
ax.set_xlabel(r'$\gamma_1 + \gamma_2$', {'fontsize'   : 20 })
ax.set_ylabel(r'$A^{n\omega}$ (dB)', {'fontsize'   : 20 })
\end{sympysilent}

The resulting figure is shown below:

\begin{center}
\sympyplot[width=0.5\linewidth]{fig}
\end{center}
```

The resulting figure is shown below:



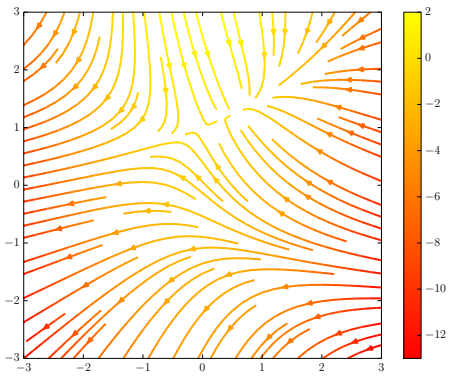
As with the `sympy.plotting` method, the plot is included into your document using the `\sympyplot` command. In this case `\sympyplot[width=\linewidth]{fig}`.

2.2.1 A more sophisticated plot

The following code is one of the demo files for streamline plotting from the matplotlib documentation. It can be entered directly into the LaTeX source as follows:

```
\begin{sympysilent}
import numpy as np
import matplotlib.pyplot as plt

Y, X = np.mgrid[-3:3:100j, -3:3:100j]
U = -1 - X**2 + Y
V = 1 + X - Y**2
speed = np.sqrt(U*U + V*V)

fig = plt.figure()

plt.streamplot(X, Y, U, V, color=U, linewidth=2, cmap=plt.cm.autumn)
plt.colorbar()
\end{sympysilent}

The result is shown below.

\begin{center}
\sympyplot[width=0.5\linewidth]{fig}
\end{center}
```

The result is shown below.



2.2.2 3D plotting

The following example shows a 3D parametric plot using matplotlib and SympyTeX.

```
\begin{sympysilent}
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
import matplotlib.pyplot as plt

mpl.rcParams['legend.fontsize'] = 10

fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='parametric curve')
ax.legend()
\end{sympysilent}

\begin{figure}
\centering
\sympyplot[width=\linewidth]{fig}
\caption{A 3D parametric plot using matplotlib and asymptote.}
\end{figure}
```

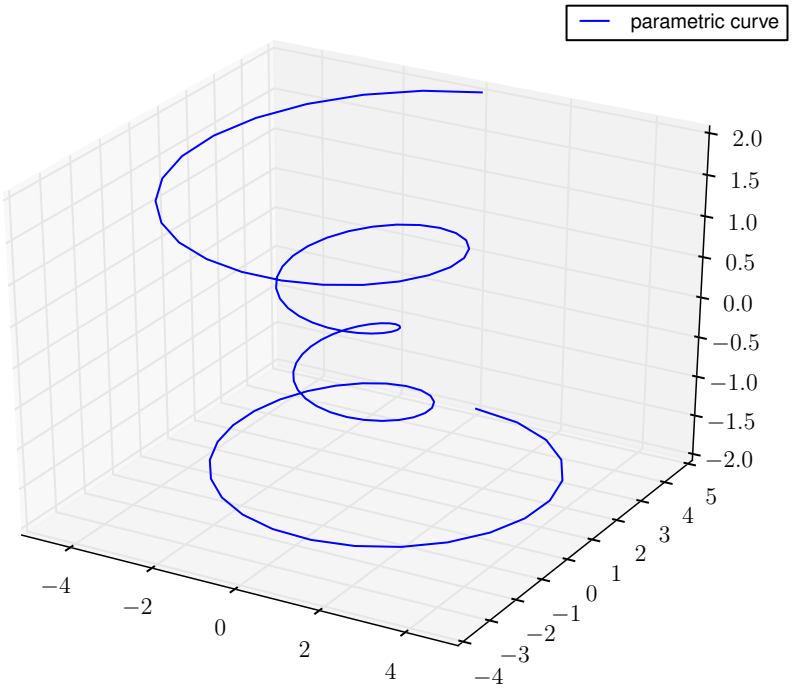


Figure 2.2. A 3D parametric plot using matplotlib and asymptote.

2.3 Plotting with PGF/TikZ

To plot figures with SympyTeX, the LaTeX package pgfplots [1] can be used. Pgfplots is using the PGF/TikZ package as a graphics backend.

The main advantages are the fast output, as we do not need to create any external files, and the good integration of pgfplots into LaTeX. A short example is given below.

2.3.1 Document Preamble

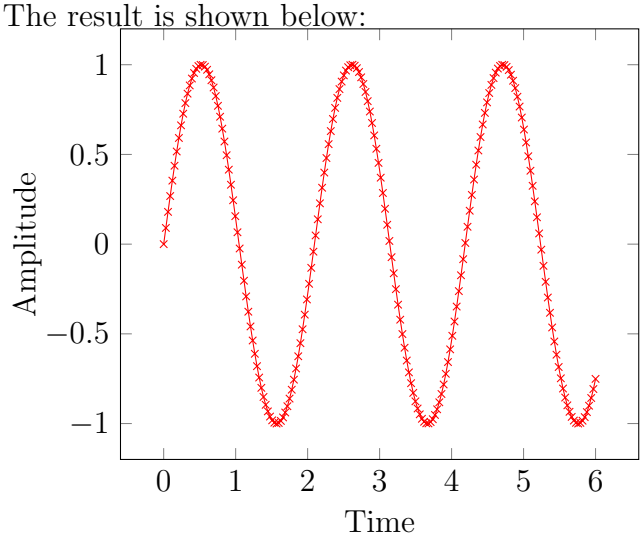
In order to use PGF/TikZ for plotting, the document preamble must contain the following:

```
\usepackage{sympytex}
\usepackage{pgfplots}
```

After calculating the data points using sympy (or any other package) we insert them into the string `pgfplot`.

```
\begin{sympysilent}
import numpy as np
time = np.linspace(0,6,200)
values = np.sin(3*time)
coordinates = ""
for point in zip(time, values):
  coordinates = coordinates + str(point)

pgfplot = r"""
\begin{tikzpicture}
  \begin{axis}[xlabel=Time,ylabel=Amplitude]
    \addplot[color=red,mark=x] coordinates { %s };
  \end{axis}
\end{tikzpicture}
""" % (coordinates)
\end{sympysilent}
The result is shown below:\newline
\begin{figure}
  \sympyplain{pgfplot}
\end{figure}
```

The result is shown below:



References

[1] Christian Feuersänger. Manual for package pgfplots. *URL http://www. ctan. org/tex-archive/help/Catalogue/entries/pgfplots. html. Probablement installé dans votre système sous le nom pgfplots. pdf*, 17, 2011.

[2] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):0090–95, 2007.

[3] Sergey Vladimirovich Lyapunov. Development of the theory of lift in the works of ne zhukovsky. *International Journal of Fluid Mechanics Research*, 26(4):465–470, 1999.

[4] Timothy C.A. Molteno. SympyTeX: Embedding symbolic computation into LaTeX documents. Technical Report 2014-3, University of Otago.

Appendix A

SympyTeX Reference

Each SympyTeX command is described in a section below.

A.1 \sympy command

A.1.1 Automatically breaking lines in long equations

To break equations automatically over long lines you can use the quote environment. This is very useful when you don't know the length of the LaTeX expression that will be produced by sympy. For example.

```
\begin{sympysilent}
x = sympy.Symbol('x')
e = 1/sympy.cos(x)
\end{sympysilent}
\begin{quote}\raggedright
$ \sympy{e} \approx
  \sympy{e.series(x, 0, 15)} $
\end{quote}
```

$$\frac{1}{\cos(x)} \approx 1 + x^2/2 + 5x^4/24 + 61x^6/720 + 277x^8/8064 + 50521x^{10}/3628800 + 540553x^{12}/95800320 + 199360981x^{14}/87178291200 + \mathcal{O}(x^{15})$$

A.2 \sympyplain command

The `sympyplain` command places the sympy output as text (rather than LaTeX formatted). This is really useful if the output is huge as formulae don't line wrap.

```
The integral of $1+x^4$ is
$\sympyplain{sympy.integrate(1+x
    **4,x)}$.
```

The integral of $1 + x^4$ is $x * *5/5 + x$.

```
\begin{sympysilent}
x = sympy.Symbol('x')
e = 1/sympy.cos(x)
\end{sympysilent}
The series expansion of $\sympy{e
    }$
is $\sympyplain{e.series(x, 0,
    10)}$.
```

The series expansion of $\frac{1}{\cos(x)}$ is $1 + x * *2/2 + 5 * x * *4/24 + 61 * x * *6/720 + 277 * x * *8/8064 + O(x * *10)$.

A.3 \sympyplot command

The `\sympyplot` command inserts graphics into a document. It is very similar to the `\includegraphics` command. Syntax is. `\sympyplot[<includegraphics opts>][fmt]{figure}` where:

- `fmt` can be one of pdf,eps,png,jpg or many more. If it is omitted, then an appropriate choice will be made for you.
- `<figure>` is a sympy.plotting object, or a matplotlib figure object

A.4 sympyblock environtment

Placing a `sympyblock` block in your code, allows you to execute sympy code within your document. The sympy instructions are included in your document.

```
\begin{sympyblock}
theta = sympy.Symbol('theta')
from sympy import *
def RotZ(angle):
  return Matrix([\
  [cos(angle), -sin(angle), 0],\
  [sin(angle), cos(angle),0],\
  [0,0,1]])
\end{sympyblock}
Some examples of rotation
    matrices are
\[ R(0) = \sympy{RotZ(0)}. \]
\[ R(\pi/2) = \sympy{RotZ(sympy.
    pi/2)}. \]
\[ R(\theta) = \sympy{RotZ(theta)
    }. \]
```

```
theta = sympy.Symbol('theta')
from sympy import *
def RotZ(angle):
  return Matrix([\
  [cos(angle), -sin(angle), 0],\
  [sin(angle), cos(angle),0],\
  [0,0,1]])
```

Some examples of rotation matrices are

$$R(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$R(\pi/2) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

A.5 sympysilent environment

Placing a `sympysilent` block in your code, allows you to execute sympy code within your document. The instructions are not included printed document output. This is very useful for setting up the sympy environment.

```
\begin{sympysilent}
theta = sympy.Symbol('theta')
from sympy import *
def RotZ(angle):
  return Matrix([\
  [cos(angle), -sin(angle), 0],\
  [sin(angle), cos(angle),0],\
  [0,0,1]])
\end{sympysilent}
Some examples of rotation
   matrices are
\[ R(0) = \sympy{RotZ(0)}. \]
\[ R(\pi/2) = \sympy{RotZ(sympy.
   pi/2)}. \]
\[ R(\theta) = \sympy{RotZ(theta)
   }. \]
```

Some examples of rotation matrices are

$$R(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$R(\pi/2) = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$R(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Appendix B

Installing SympyTeX

You must have LaTeX and the sympy Python package installed on your system. For Debian based systems[1] the following commands should suffice.

```
sudo aptitude install python-sympy
```

B.0.1 Important! Setting up Your LaTeX environment

To use SympyTeX, you have to run latex (or pdflatex) on your file, then python, and then latex again.

```
latex example.tex
python example.sympy
latex example.tex
```

If you are using pdflatex, then you would use the following commands

```
pdflatex example.tex
python example.sympy
pdflatex example.tex
```

If you use Kile, or another LaTeX environment, you can set it up to automatically do the sympy python command as part of your build process.

[1] This includes Debian, Ubuntu and other linux systems.

Appendix C

Making SympyTeX easy to use

Perhaps the simplest way to automate the use of SympyTeX is to use a Makefile. This file describes a series of actions that are triggered when files change. This approach is described in Section C.1.

Another approach is to set up a LaTeX development environment. Section C.2 shows how to do this for Kile – part of the KDE desktop environment. There are many other environments, some popular ones include Texmaker, TeXnicCenter, AUCTeX and TeXShop. Most of these can be configured in a similar way to Kile. The general principle is to run SympyTeX after running LaTeX on the document.

C.1 Using a Makefile

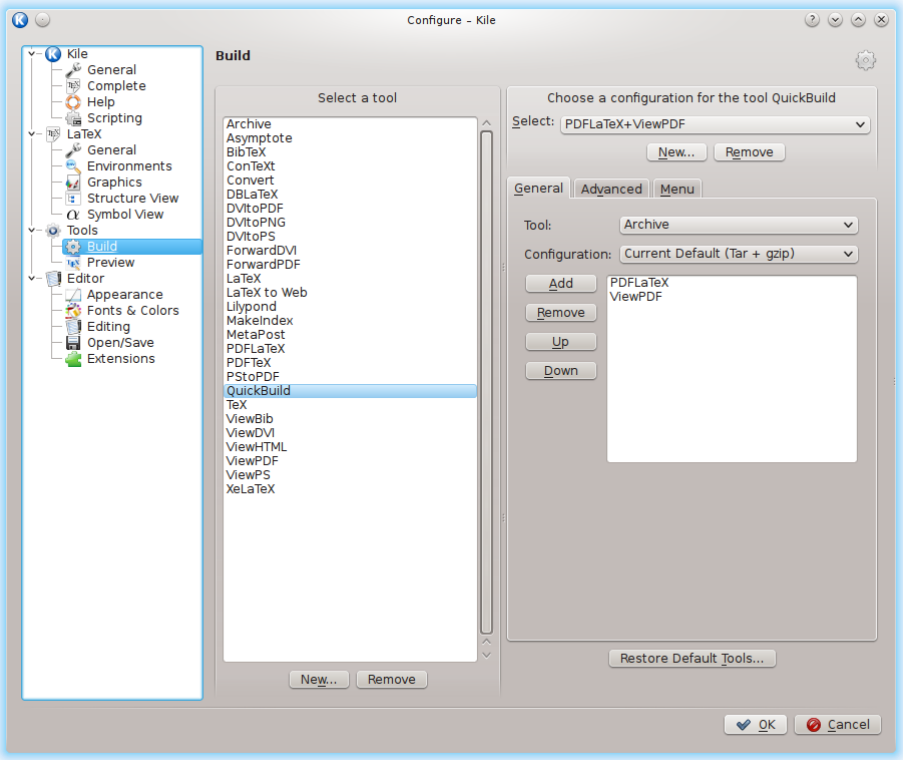
A makefile is a quick way to automate the process of building your document. The following file is an example:

```
#
#   Sample Makefile for generating a pdflatex
#   document that includes symbolic computations
#   that use sympy.
#
#   Requires the following tools
#
#   aptitude install make python-sympy
#

#   Default target. Just type 'make' to build the document.
all:    clean
    pdflatex ETR_sympytex.tex
    python ETR_sympytex.sympy
    pdflatex ETR_sympytex.tex

#   Remove temporary files. These can cause sympy to fail
#   if you have changed the .tex document and the wrong
#   sympy output is inserted.
clean:
    rm -f *.pyc
    rm -f *.sout
    rm -f *.sympy
```

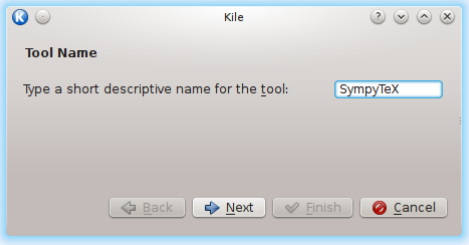
C.2 Configuring Kile to use SympyTeX

The KDE LaTeX tool can be configured to automatically call SympyTeX when a document is built. To do this, we will add a new tool to Kile called 'SympyTeX'.

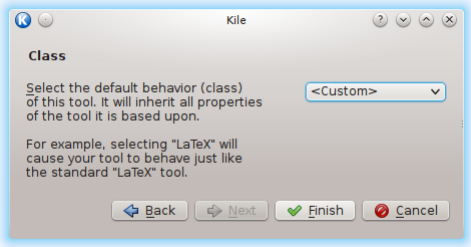
Go to the Settings menu and choose the "Configure Kile" option. Select the Tools/Build panel as shown:



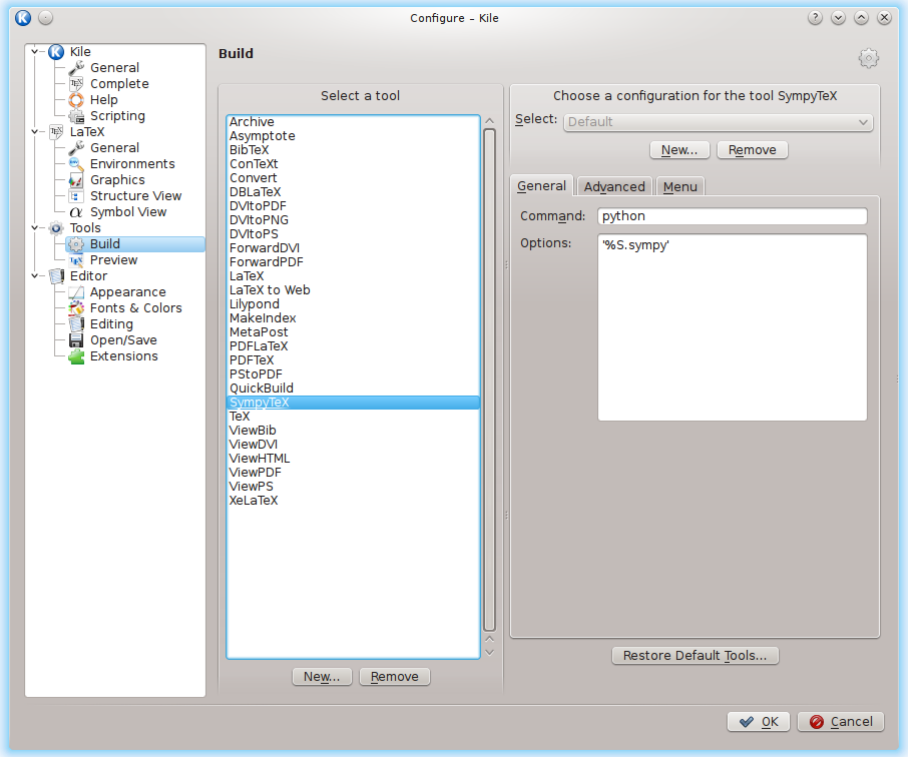
Now click the new button and name the tool SympyTeX and click 'next'



Choose the 'custom' class, and click 'finish'



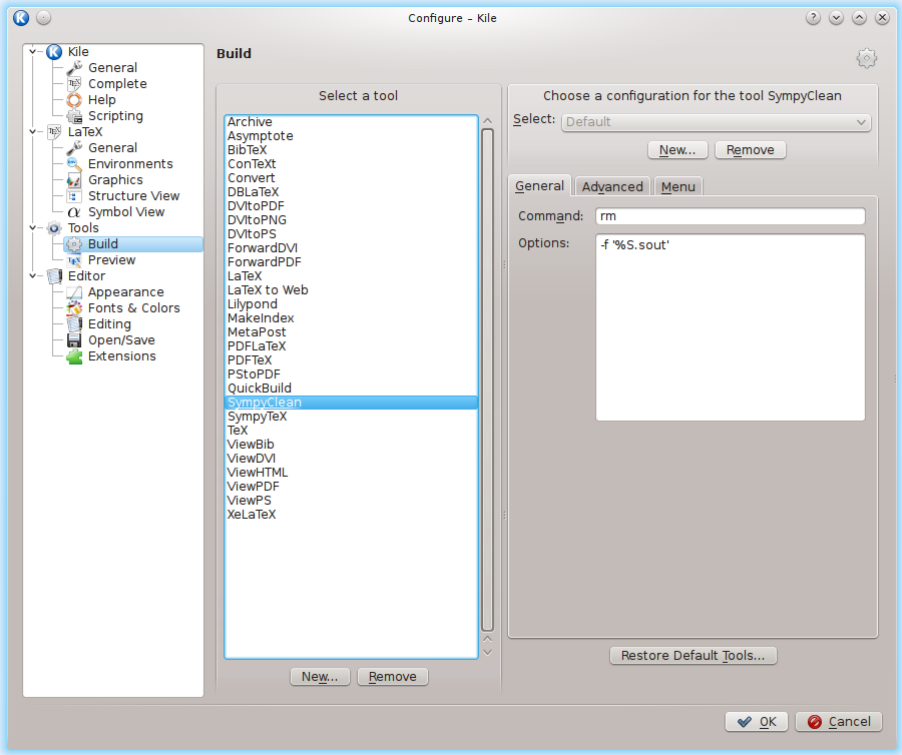
Now select the SympyTeX tool, and edit the command and options as shown below:



At this point, you can now choose the SympyTeX option from the Build → Other menu.

C.2.1 Important! Cleaning up

A command must be included in the quickbuild that removes the sympytex temporary output files. Create a command called SympyClean that removes the `.sout` file. This is shown below.

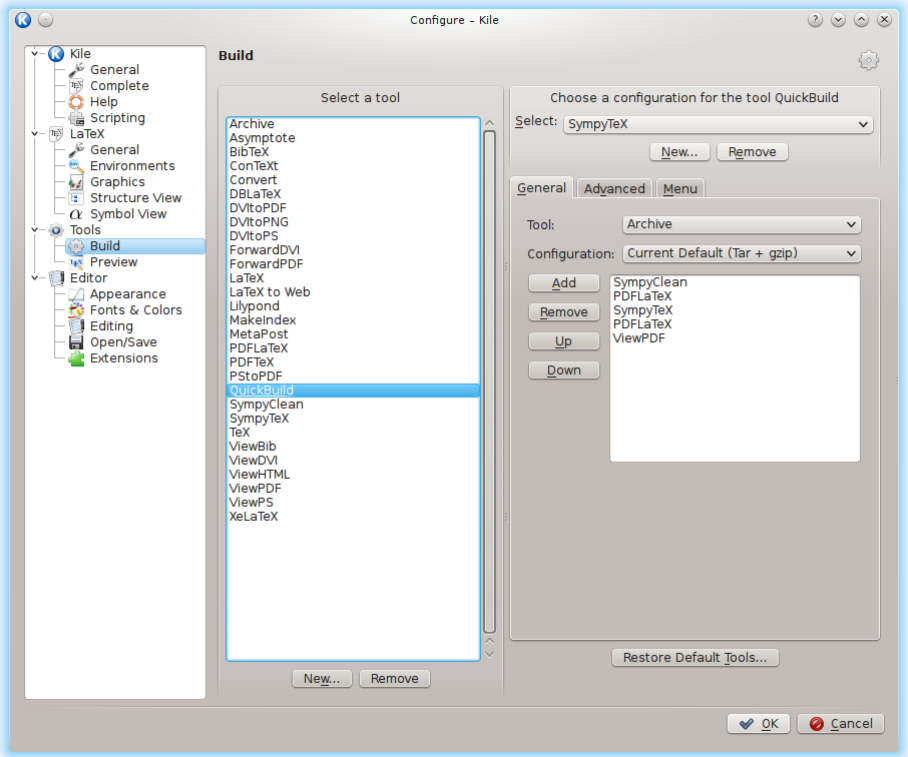


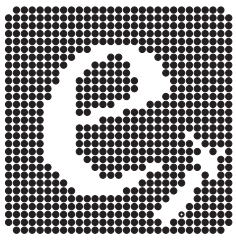
At this point, you can now choose the SympyTeX option from the Build → Other menu.

C.2.2 Setting up QuickBuild

To automate the process, and have SympyTeX called every time your document is built[1], edit the quickbuild option as shown below

[1]This will slow things down a bit so take care when doing this on large documents with complex calculations


Configure – Kile
Kile
General
Complete
Help
Scripting
LaTeX
General
Environments
Graphics
Structure View
Symbol View
Tools
Build
Preview
Editor
Appearance
Fonts & Colors
Editing
Open/Save
Extensions
Build
Select a tool
Archive
Asymptote
BibTeX
ConTeXt
Convert
DBLaTeX
DVItoPDF
DVItoPNG
DVItoPS
ForwardDVI
ForwardPDF
LaTeX
LaTeX to Web
Lilypond
MakeIndex
MetaPost
PDFLaTeX
PDFTeX
PStoPDF
QuickBuild
SympyClean
SympyTeX
TeX
ViewBib
ViewDVI
ViewHTML
ViewPDF
ViewPS
XeLaTeX
New...
Remove
Choose a configuration for the tool QuickBuild
Select:
SympyTeX
New...
Remove
General
Advanced
Menu
Tool:
Archive
Configuration:
Current Default (Tar + gzip)
Add
Remove
Up
Down
SympyClean
PDFLaTeX
SympyTeX
PDFLaTeX
ViewPDF
Restore Default Tools...
OK
Cancel



Electronics Group
Department of Physics
University of Otago
elec.otago.ac.nz