Using TeXShop 5.02 to Write Interactive Documents

Richard Koch

August 31, 2022

1 What's New

TeXShop 5.01 contains an article explaining how to write interactive documents with TeXShop and TeX4ht. Immediately after the release of TeXShop 5.01, I discovered a better way of doing things. This document is a new version of the original one which describes the better way.

The original version of the document was designed to be read while typesetting a document named Fourier. Fourier.tex was also revised in TeXShop 5.02 to reflect the new techniques described here.

What changed? In both versions, TeX4ht is used to convert ordinary LaTeX source to html output and thus a web document. In both versions, interactive elements (2D and 3D graphs using Sage, UTube videos, code to experiment with numerical integration) are added by writing html source directly.

But originally, this html source created independent web pages linked from the main web document.

Now the html source is entered directly into the original Latex source file, and the resulting web document can be read linearly, with technical text and interactive additions mixed on a single web page.

2 Introduction

TeXShop 5 contains several new features. I'm going to illustrate how these features can be used to write course notes and similar documents with interactive content.

I'll use an old project I happen to have around the house. The name of the document is *Fourier*. Please open that document and follow along as I explain the new TeXShop features.

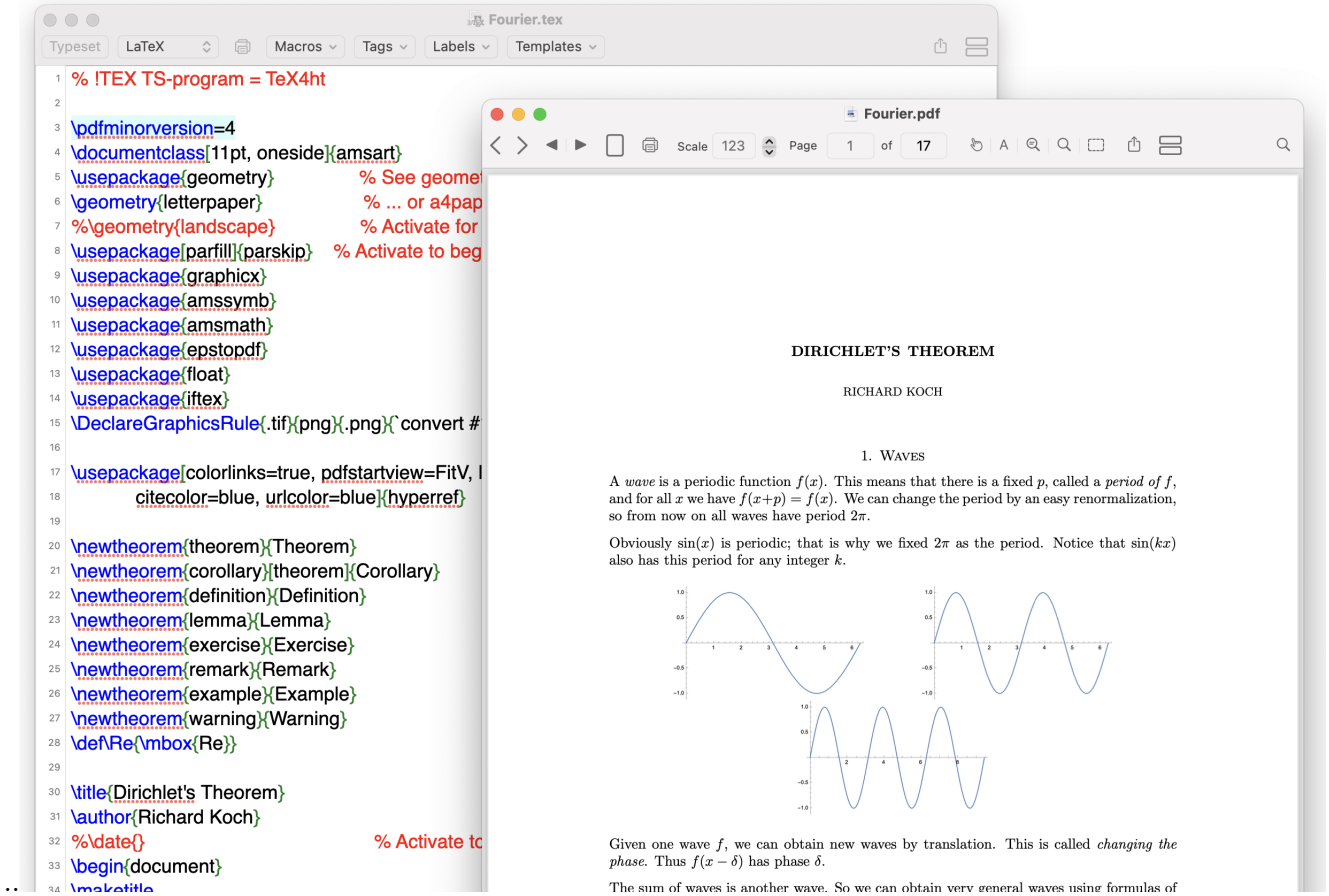
The entire session will be done within TeXShop. I'll use only two other applications for very minor reasons, and I'll point out those moments.

3 A Document about Fourier Series

Fourier is a mathematics document. If you are into math, it introduces Fourier series and contains Dirichlet's 1829 proof that the Fourier series of a piecewise differentiable function converges pointwise to the function. The initial line of this document is a comment, but TeXShop understands that comment. It instructs TeXShop to typeset the document using pdflatex.

Start by opening the document and typesetting. Notice that TeXShop works as it always has. Scroll through the output. The document contains illustrations and extensive mathematics; it uses hyperref and other standard packages, and has a table of contents. The line activating the table of contents is commented out to make the document easier to read. Feel free to activate that line. The standard TeXShop view of such a document is shown at the top of the next page.

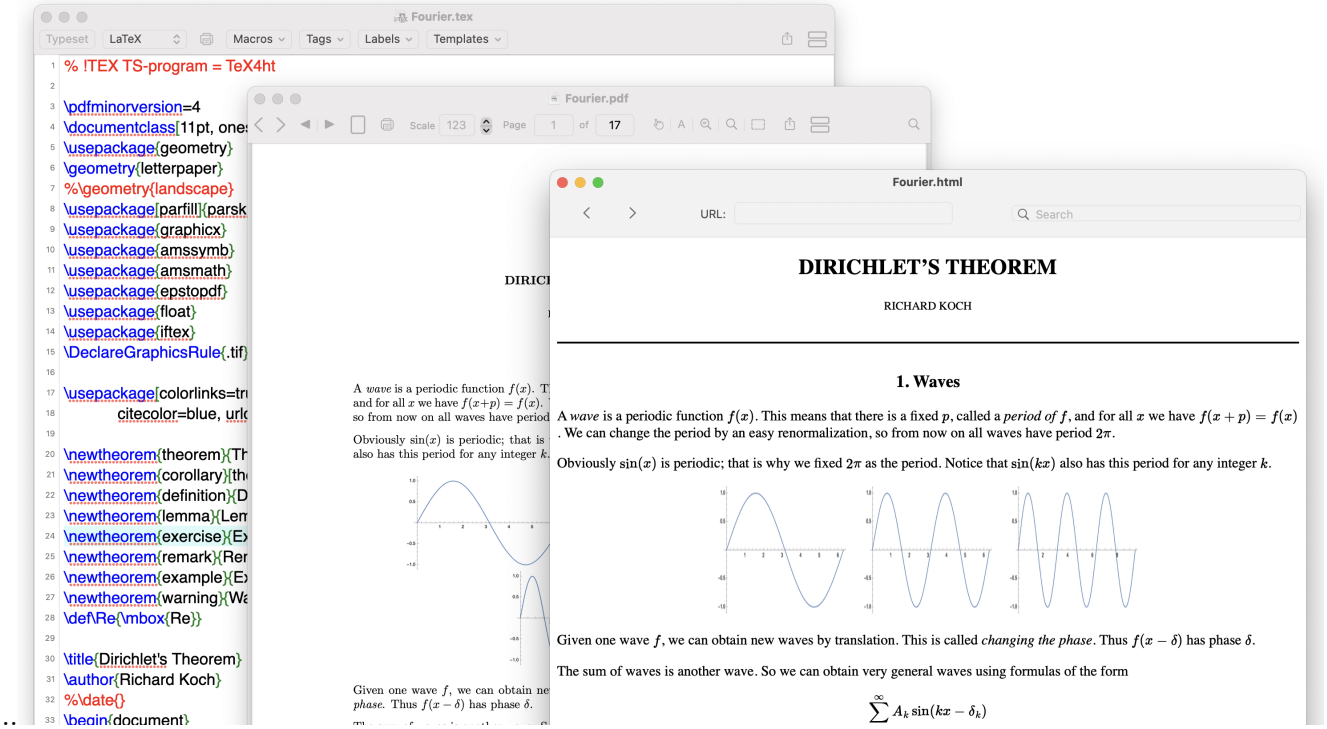
Ready for experiments?



4 Typesetting with TeX4ht

In the first line, change the word "pdflatex" to "TeX4ht". This tells TeXShop to typeset with the engine TeX4ht, a program by Eitan Gurari that accepts standard LaTeX as source, but outputs a web page in html. Notice that the change has been made in the image above. Close the Preview window and then typeset. The result is shown at the top of the next page.

The TeX4ht engine typesets twice, once with pdflatex and once with TeX4ht. The pdf output is opened in a standard Preview window and the html output is opened in a TeXShop web view. Thus three windows are visible.



Scroll through the html document. It is essentially perfect. The illustrations look fine and the mathematical equations are crystal clear. It is difficult to believe that this is an html document. To make sure, resize the pdf window and notice that the contents shrink (if you are in "fit to window mode"). Resize the html and notice that the text reflows.

To be completely honest, there are a couple of minor flaws in the html. Some of the graphic images are too narrow. There is an easy fix for this problem, but I deliberately did nothing because I want to show that TeX4ht is almost perfect.

Fifteen years ago I watched Eitan demonstrate TeX4ht at a TUG Conference. At the time, TeX4ht converted mathematical equations into small pictures and displayed those pictures. The output was a little crude. I thought TeX4ht was an amusing program, but completely misunderstood its significance or the difficult problems that Eitan had solved.

In 2009, just weeks before he was to give another TUG talk, Eitan unexpectedly died. But TUG paid him the ultimate compliment by keeping his program alive. The current programming is done by Michal Hoftich, who is very active with updates that appear almost daily. Eitan's hard work took wings due to two independent and enormous projects: MathML and MathJax.

The web was invented on a NeXt computer by Tim Berners-Lee, a physicist working at CERN. It could do many wonderful things, but surprisingly it could not display mathematical content. This was eventually solved by the invention of MathML, a new tagging scheme able to encode mathematical equations. It was not sufficient to invent the scheme; browsers had to be revised to render the equations. This took many years. I heard several early talks about MathML at TUG and my reaction was "luckily, I'll never have to use that."

Safari and most other browsers can render MathML, but it is not perfect. Integral signs are often far smaller than they should be and there are other flaws. This problem was definitively solved by another project called MathJax, which searches an html document for its MathML content and directly renders that content, bypassing the browser.

MathJax began as a project by Davide Cervone in 2004, when it was called jsMath. Eventually the American Mathematical Society recognized the importance of this project, and designed the MathJax project in 2009 as a successor to jsMath.

Both MathML and MathJax are enormous projects. Please don't ask me how they work because I have no idea. Luckily, the projects are easy to use, and TeXShop's support mainly happens automatically.

5 Why is HTML Important?

Some students have desktop computers, some have portables, some had iPads or other devices. All devices have powerful browsers and web support, and all can display pdf documents. But pdf is a static format, so if a screen is small, the only choice is make all the text smaller. Html can reflow the text to work well on multiple screen sizes.

Html is interactive. Students can enter text into a document, or check boxes, or select answers from a list, or run experiments. Pdf is a static format. Attempts have been made to add interaction to pdf, but progress is slow and uneven across platforms.

Much recent work has been devoted to making documents accessible to readers with disabilities. The area is complicated because a system useful with one disability may be irrelevant for those with a different one. Some members of TUG have made a point of contacting actual users to understand their needs and preferences. One member reported that his contact preferred the tagging system on html documents to those on pdf documents, and avoided pdf documents if possible.

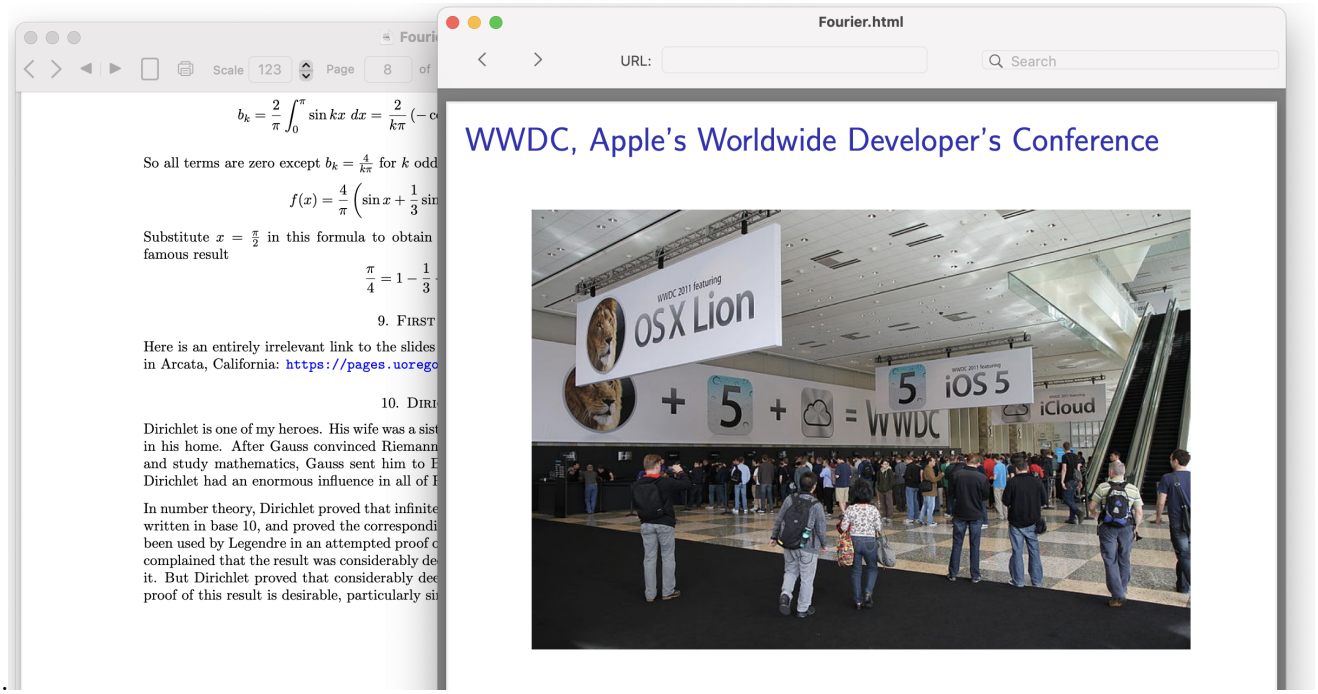
Who knows what is best? We don't have to choose. We can output documents in both pdf and html forms. Put both on the web, and let your users decide which to use.

6 A Standard Link

Activate the Pdf Preview Window. Scroll to section 9 in the document. The text reads "Here is an entirely irrelevant link to the slides of a talk I once gave at Cal Poly Humboldt in Arcata, California." And indeed this is a url to a series of slides on my University of Oregon web site. The slides have nothing to do with Fourier Series. Click the link anyway.

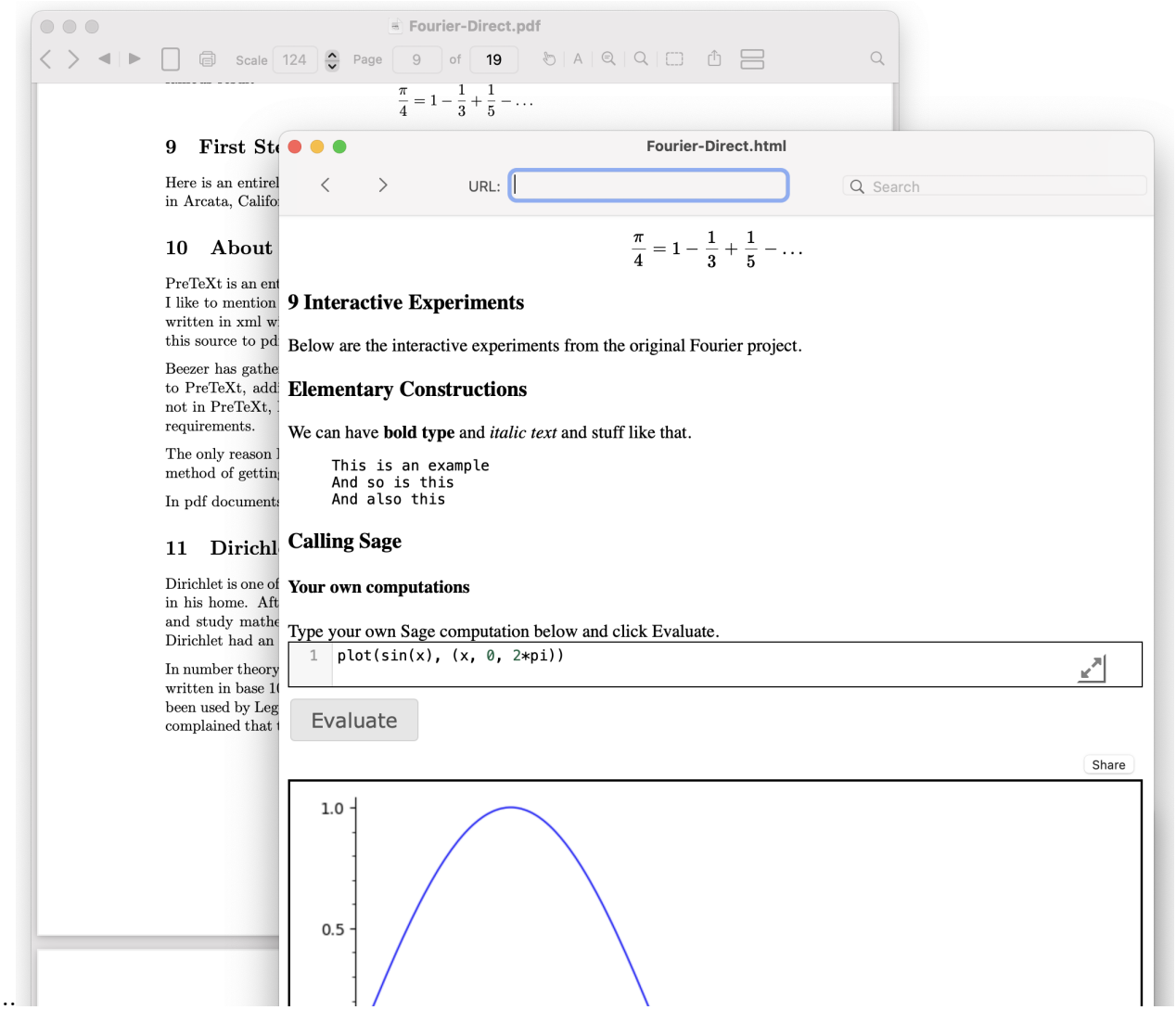
Earlier I warned that I would run two external programs. This is one of those times. Safari opens and displays the link. That's how links work in a pdf document. Quit Safari.

Now find section 9 in the html document. The ninth section has a different name in this document and completely different content. We'll come back to that later. But at the very end of section 9, that link to my slides exists. Click it. This time the link opens directly in the html preview window, because that's how the web works. We aren't distracted by another program. Use the arrows in the toolbar to return to the original document.



7 Different Output

Notice that section 9 looks completely different in the pdf and html documents. Even the name of the section is different. Here are the two versions of section 9:



What is the LaTeX source code which produced this result? It is easy to find that source because synctex works in the pdf Preview, and we discover the following source:

```
\ifx\HCode\undefined
   \section{First Steps}
\else
   \section{Interactive Experiments}
     Below are the interactive experiments from the original Fourier project.
   \begin{html}
        ---- source directly in html ---
   \end{html}
\fi
```

The first line in this source is obscure code from the TeX4ht world; that line is false if TeX4ht is running, and true in all other cases. Consequently, it is easy to produce different output in the pdf document and the html document. The source immediately following the "ifx" line goes to pdf output, the source following the "else" goes to html output, and the source after the "fi" line again goes to both outputs.

But something else happens at this point in the source code. Between the two lines referring to html, the source is no longer in LaTeX. Instead, it is pure html code, defining the interactive pieces of the document.

Comparing the pdf and html documents, we find that they are exactly the same everywhere except sections 9 and 10. So we must look closely at these two sections.

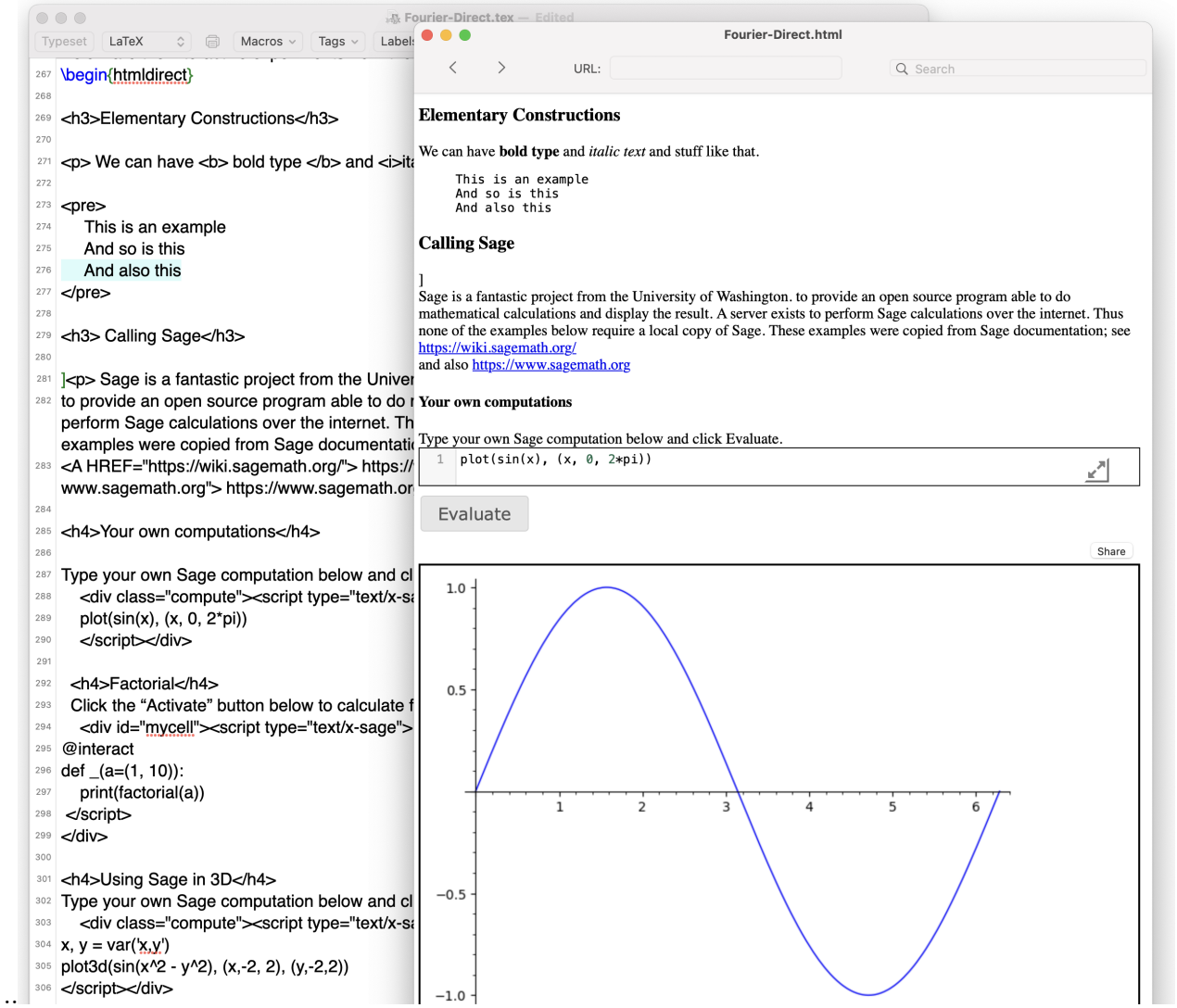
8 The Point of It All

This is the key moment in the demonstration. We need to compare the source code for section 9 with its output in the web window. So arrange the source code on the left and the HTML Preview window on the right, as shown on the next page.

The first thing we notice is that the source is now indeed in html. Note the html tags to create sentences, bold text, italics, and verbatim text.

Next come some examples using SageMath. This open source program will be discussed in the next section. Suffice it to say that SageMath is a project designed to provide an alternative to Maxima, Maple, Mathematica, and similar programs. Sage provides a server which can be contacted by web pages to provide interactive sessions without requiring that the reader has Sage installed on their local machine. Examples on this page come directly from the Sage Project.

The HTML Preview page shows that Sage has been asked to plot sin(x), and it responded with a plot. The line of sage code can be modified directly on the web page, and Sage will then plot any function the user desires.



The source field shows the html input to achieve this output. It consists of just three lines:

```
<div class="compute"><script type="text/x-sage">
plot(sin(x), (x, 0, 2*pi))
</script></div>
```

These lines suffice for any call to Sage, where the middle line can be replaced by any Sage program.

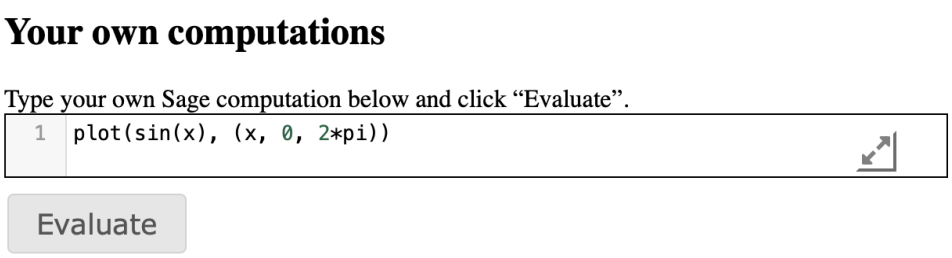
9 Sage

SageMath is an open source alternative to the computer algebra systems Magma, Maple, Mathematica, and MATLAB. The project was created by William Stein, a mathematician at the University of Washington, and released on February 24, 2005. See `https://sagemath.org` and `https://wiki.sagemath.org`. Sage is mostly written in Python, but it integrates many previous open source projects written in C, Lisp, and Fortran. Among these are Gap, GP, Macaulay, Maxima, Octave, and R. The program has been used for serious research on Elliptic Curves, Finite Groups, and many other areas, and has an active support group with contributions by thousands.

The Sage web site has an install package for the Macintosh. Sage has support for LaTeX, so it is possible to write Sage Code in the middle of a LaTeX document and automatically call Sage during typesetting to produce a graph, or compute an integral symbolically. However, the development I'll show does not depend on installing SageMath, and is independent of the facility to integrate Sage calls into a LaTeX source file.

Sage maintains a server which can run Sage over the web. This server allows web pages to contain interactive content based on SageMath. See `https://sagecell.sagemath.org` and other links from that page for details. The servers can be used for free. Our example project contains several examples copied directly from the Sage web pages. To repeat: I did not write any of these programs myself. And to also repeat: the web page constructed from Fourier should work for any user in the world on any operating system without installing Sage.

Section 9 of the html version of Fourier contains the following item:



..

When the Evaluate button is pressed, the item changes to the view at the top of the following page.

But there is more. The text "plot(sin(x), (x, 0, 2*pi))" in this item is editable, so if you change it to "plot(sin(x) + cos(3*x)/3, (x, 0, 2*pi))" and push Evaluate again, you'll produce the second plot on the next page.

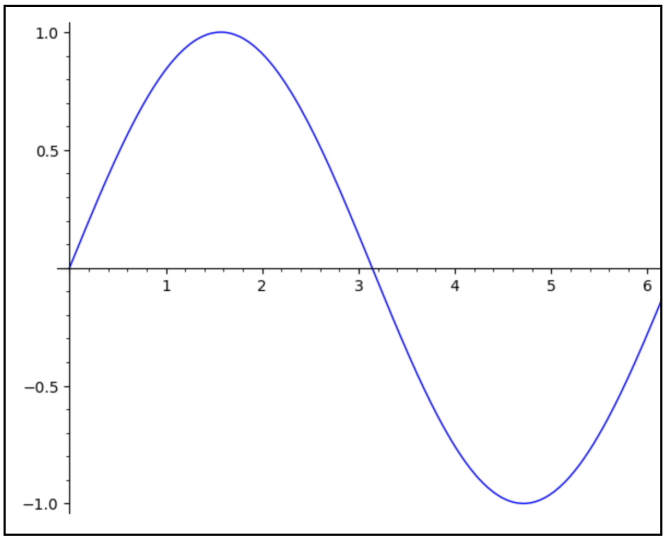
Your own computations

Type your own Sage computation below and click “Evaluate”.

```
1 plot(sin(x), (x, 0, 2*pi))
```

Evaluate

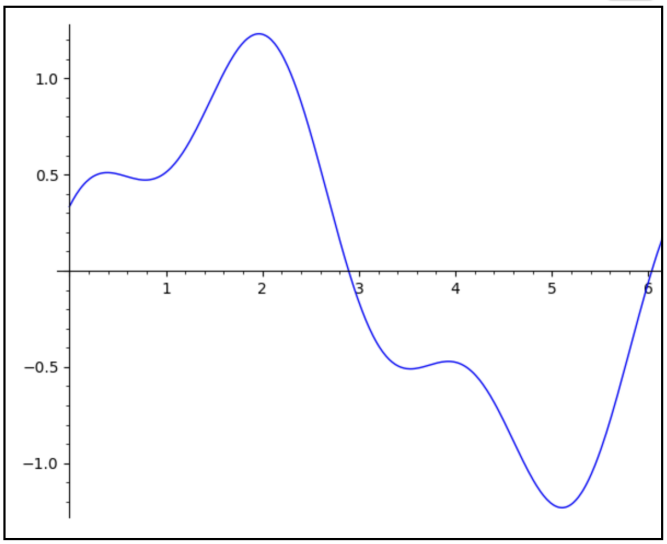
Share



Help | Powered by SageMath

..

Share



Help | Powered by SageMath

..

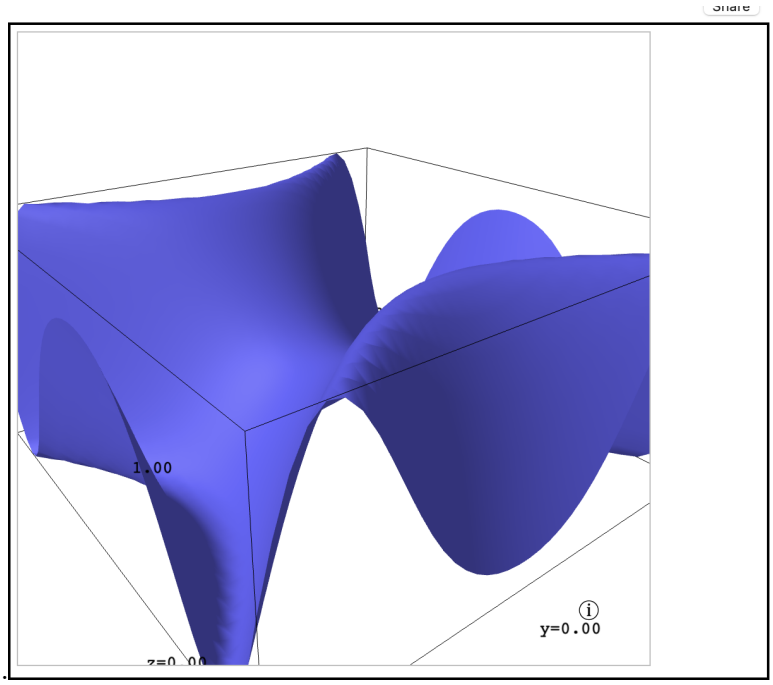
Here is the source code which generated the previous plotting example. Very simple.

```
<div class="compute"><script type="text/x-sage">
plot(sin(x), (x, 0, 2*pi))
</script></div>
```

However, the amazing thing is that the line between the first and last lines of this source can be replaced by *any Sage program*. So very complicated interactive items can be created as soon as the author of the document learns a little Sage (or, like me, just copies the work of others). Here are three examples. I confess that I find these utterly astonishing. If the Sage code is replaced with

```
x, y = var('x,y')
plot3d(sin(x^2 - y^2), (x,-2, 2), (y,-2,2))
```

we get



This picture is not all that impressive on the printed page, but if you are working with the live plot in the Fourier example, you'll notice that you can hold down the mouse button and rotate the figure in real time. This interaction could not possibly happen in real time over the internet, so Sage must be directly communicating with the GPU in the Macintosh. Since the program is platform independent, it must also have different code when the user is running Windows, or Linux with various graphic cards.

The example labeled "A Calculus Course Experience" contains the following code. I list it to show that quite complicated program can be run by the server. This code is by Nick Alexander, based on the work of Marshall Hampton.

```
var('x')
@interact
def midpoint(f = input_box(default = sin(x^2) + 2, type = SR),
    interval=range_slider(0, 10, 1, default=(0, 4), label="Interval"),
    number_of_subdivisions = slider(1, 20, 1, default=4, label="Number of boxes"),
    endpoint_rule = selector(['Midpoint', 'Left', 'Right', 'Upper', 'Lower'], nrows=1, labe

    a, b = map(QQ, interval)
    t = var('t')
    func = fast_callable(f(x=t), RDF, vars=[t])
    dx = ZZ(b-a)/ZZ(number_of_subdivisions)

    xs = []
    ys = []
    for q in range(number_of_subdivisions):
        if endpoint_rule == 'Left':
            xs.append(q*dx + a)
        elif endpoint_rule == 'Midpoint':
            xs.append(q*dx + a + dx/2)
        elif endpoint_rule == 'Right':
            xs.append(q*dx + a + dx)
        elif endpoint_rule == 'Upper':
            x = find_local_maximum(func, q*dx + a, q*dx + dx + a)[1]
            xs.append(x)
        elif endpoint_rule == 'Lower':
            x = find_local_minimum(func, q*dx + a, q*dx + dx + a)[1]
            xs.append(x)
    ys = [ func(x) for x in xs ]

    rects = Graphics()
    for q in range(number_of_subdivisions):
        xm = q*dx + dx/2 + a
        x = xs[q]
        y = ys[q]
        rects += line([[xm-dx/2,0],[xm-dx/2,y],[xm+dx/2,y],[xm+dx/2,0]], rgbcolor = (1,0,0)
        rects += point((x, y), rgbcolor = (1,0,0))
    min_y = min(0, find_local_minimum(func,a,b)[0])
```

```
max_y = max(0, find_local_maximum(func,a,b)[0])

pretty_print(html('<h3>Numerical integral with the {} rule</h3>'.format(endpoint_rule))
show(plot(func,a,b) + rects, xmin = a, xmax = b, ymin = min_y, ymax = max_y)

def cap(x):
    # print only a few digits of precision
    if x < 1e-4:
        return 0
    return RealField(20)(x)
sum_html = "%s \\cdot \\left[ %s \\right]" % (dx, ' + '.join([ "f(%s)" % cap(i) for i i
num_html = "%s \\cdot \\left[ %s \\right]" % (dx, ' + '.join([ str(cap(i)) for i in ys

numerical_answer = integral_numerical(func,a,b,max_points = 200)[0]
estimated_answer = dx * sum([ ys[q] for q in range(number_of_subdivisions)])

pretty_print(html(r'''
<div class="math">
\begin{align*}
\int_{a}^{b} {f(x) \, dx} & = %s \\\
\sum_{i=1}^{%s} {f(x_i) \, \Delta x} & = %s \\\
& = %s \\\
& = %s . \end{align*} </div>'''
                 % (numerical_answer, number_of_subdivisions, sum_html, num_html, esti
```

The result of pushing Evaluate is shown on the next page.

Note that several items are interactive. A user can choose a different function at the top of the page. The number of rectangles can be varied by a slider. The height of the rectangles can be determined by the value of f on the left side, the right side, the midpoint, or by its maximum or minimum. Since Sage can integrate symbolically, the exact value of the integral is given at the bottom, followed by the Riemann Sum approximation and its value. So a wealth of examples could be suggested for students based on this one example.

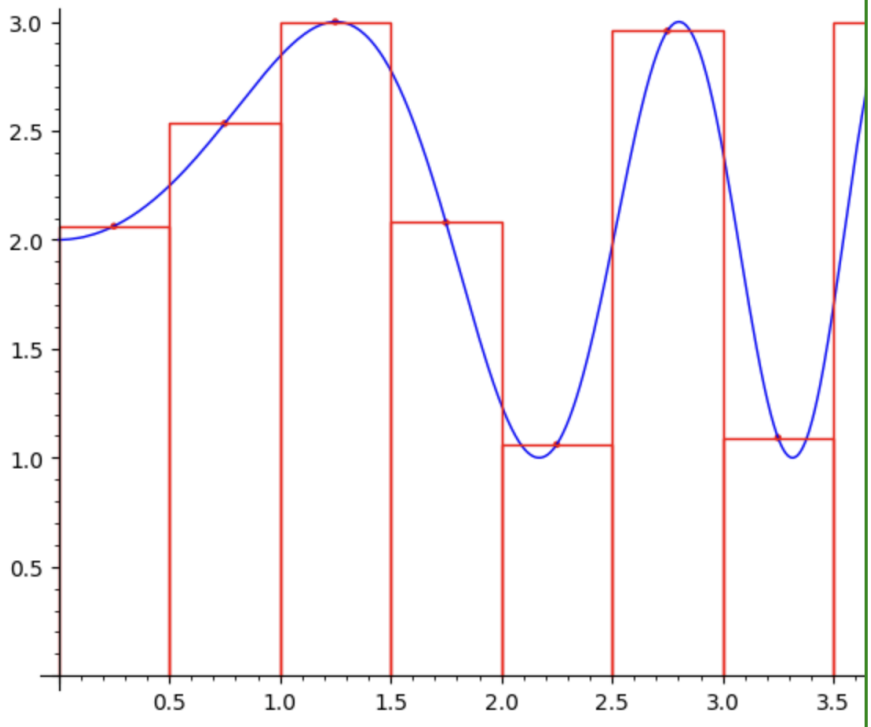
f sin(x^2) + 2

Interval (0, 4)

Number of interval 8

Endpoint rule Midpoint

Numerical integral with the Midpoint rule



$$\int_a^b f(x)\,dx = 8.747133844648115$$

$$\begin{aligned}\sum_{i=1}^{8} f(x_i)\,\Delta x &= 1/2 \cdot [f(0.25000) + f(0.75000) + f(1.2500) + f(1.7500) + f(2.2\\ &= 1/2 \cdot [2.0625 + 2.5333 + 3.0000 + 2.0790 + 1.0607 + 2.9578 + 1\\ &= 8.891378072868944.\end{aligned}$$

Help | Powered by SageMath

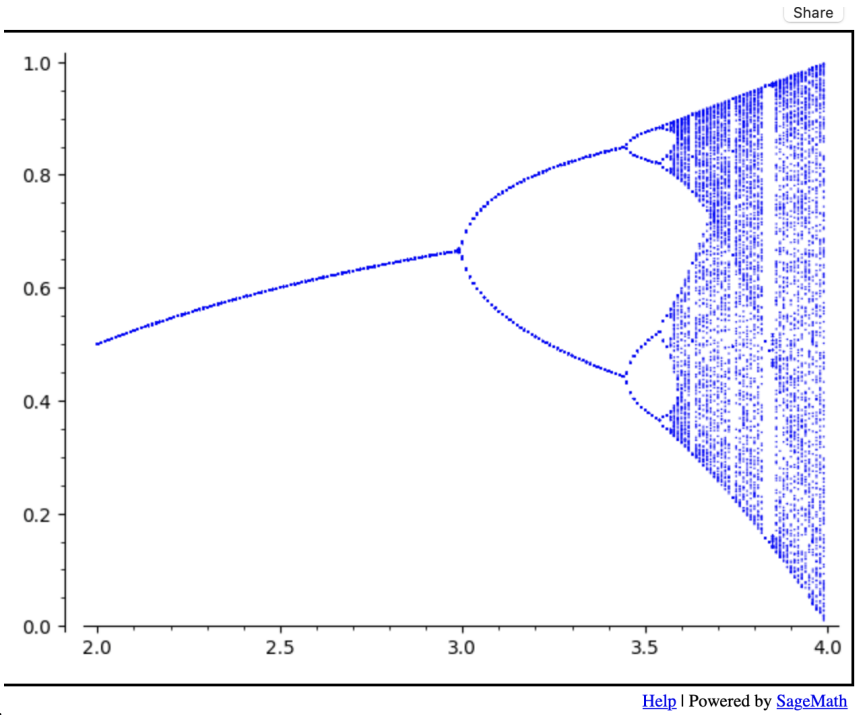
..

Another example contains code to calculate the Feigenbaum diagram, a canonical example in chaos theory. Many of us have probably written this code in one language or another. Here it is written in Sage.

```
N=200
M1=200
M2=200
x0=0.509434

puntos=[]
for t in range(N):
   mu=2.0+2.0*t/N
   x=x0
   for i in range(M1):
       x=mu*x*(1-x)
   for i in range(M2):
       x=mu*x*(1-x)
       puntos.append((mu,x))
point(puntos,pointsize=1)
```

Here is the result:



..

Other Sage examples are shown in our example. I'll let readers explore them.

10 UTube

Leaving Sage behind, the next item in section 9 is a video from UTube. Did you know that if you right click on a UTube video, a contextual menu appears allowing you to copy code which can be pasted in a web page? The code provides that video for viewers of your page.

I found a lecture by John Maynard, one of the four Field's Prize winners at the International Congress of Mathematicians for 2022. It is fun to watch this video for the depth and clarity of his mathematics. I confess that I also watched because Maynard is left-handed and we lefties need to stick together.

Here's the code. I don't understand a word of it.

```
<iframe width="928" height="522" src="https://www.youtube.com/embed/kQqBeuk_xQw"
title="13. Large gaps between primes in subsets - James Maynard (University of Oxford)
[2017]" frameborder="0" allow="accelerometer; autoplay; clipboard-write;
encrypted-media; gyroscope; picture-in-picture" allowfullscreen></iframe>
```

Here's a frame of the movie.

..

11 A Crucial Step to Make This Work

Recall that the interactive code was bracketed by two commands

```
\begin{html}
   --- html source code here ---
\end{html}
```

However, these are not commands which TeX4ht automatically understands. Our Sage examples were bracketed by two canonical lines

```
<div class="compute"><script type="text/x-sage">
plot(sin(x), (x, 0, 2*pi))
</script></div>
```

But these are lines which neither html nor TeX4ht automatically understand.

Before these lines can be used in a TeX4ht source, they must be defined by special code which comes just after "begin{document}" and before any other line of LaTeX is entered. Here are the lines.

```
\ifx\HCode\undefined
\else
 % declare environment html:
 \ScriptEnv{html}
    {\ifvmode\IgnorePar\fi\EndP\NoFonts\hfill\break }
    {\EndNoFonts }
  \fi

 \ifx\HCode\undefined
 \else
 \begin{hrml}
 <script src="https://sagecell.sagemath.org/static/embedded_sagecell.js"></script>
<script>
    // Make the div with id 'mycell' a Sage cell
    sagecell.makeSagecell({inputLocation:  '#mycell',
                           template:       sagecell.templates.minimal,
                          evalButtonText: 'Activate'});
    // Make *any* div with class 'compute' a Sage cell
    sagecell.makeSagecell({inputLocation: 'div.compute',
                          evalButtonText: 'Evaluate'});
    </script>
 \end{html}
 \fi
```

The first set of lines is for html, and the second set is for Sage. Note that both sets are bracketed by "ifx" so they will be read by TeX4ht but not by pdflatex. The second set is also bracketed by an html call so these lines will be put directly into the html file.

TeXShop has a template named "TeX4ht-Header" which contains these lines and inserts them into the source.

12 Mathematics in Interactive HTML code

At the very end of the interactive section 9, we have inserted some mathematics. Recall that the source for this interactive section is written directly in html. A glance at our source shows that it has been entered using LaTeX notation. This is surprising because it is supposed to be impossible to write mathematics that way in an html document. Instead, we ought to have written MathML code. To be sure, TeX4ht converts Latex to html, but remember that the html in the interactive sections is inserted directly into the final TeX4ht output.

The reason this works is that our TeX4ht engine calls MathJax to render mathematics. To explain this, we need to step back and explain how TeX4ht works.

13 More about TeX4ht

The TeX4ht program can process mathematical equations in several different ways, depending on parameters passed to it when it is called. It can provide small pictures, like the original version. It can output MathML and leave the rendering of this code to the browser. It can output MathML, but call MathJax to do the rendering. Or it can output ordinary LaTeX and call MathJax to do the rendering.

I experimented with the last three variations. It is very easy to call TeX4ht to work in each of these ways, and thus to write various kinds of TeX4ht engines. Below is what you would write in a TeXShop engine, where "$1" becomes a complete path to the source file. To output MathML and leave rendering to the browser,

```
make4ht "$1" "mathml,mathml"
```

To output MathML but let MathJax render the result

```
make4ht "$1" "mathml,mathjax"
```

To output Latex and let MathJax render the result

```
make4ht "$1" "mathjax"
```

Recall that when entering html code directly, we enter mathematics in LaTeX notation. Curiously this works in the first and third ways to call TeX4ht, but not the second. In

the second method, we see the source for the mathematics rather than the mathematics itself.

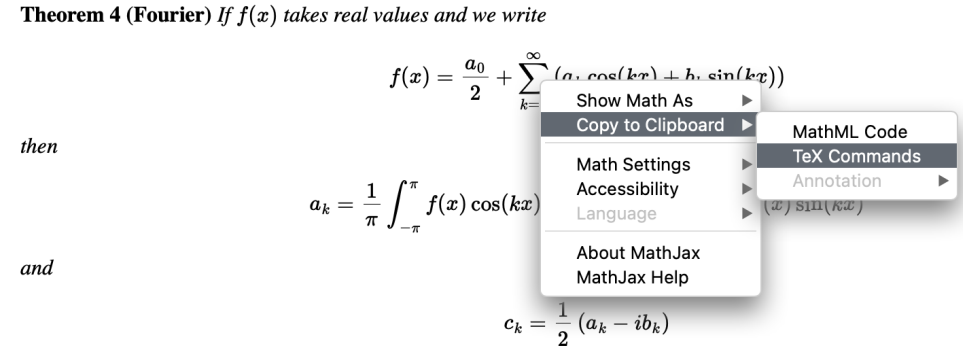
Aside from this point, asking the browser to render MathML was the worse choice; the results were acceptable but nothing to brag about. Outputting MathML but calling MathJax to render it worked very well. Outputting Latex code and asking MathJax to render it worked well for inline code, but produced displayed equations which were too large. However, this problem was fixed by Michal Hoftich one day after I wrote him about it in an email. So if you have updated TeX Live with TeX Live Utility on or after September 1, 2022, I recommend the third option. These notes are written assuming that you use that option.

Even if you do not have the updated TeX4ht, the third option can work once you understand why I ran into trouble. Recall that inline mathematics can be created using a pair of $ signs, or by the paired symbols \(and \). Similarly display mathematics can be created using a pair of $$ signs, or by the paired symbols \[and \]. TeX4ht understands both conventions.

Before September 1, TeX4ht in the third mode inserted LaTeX code for display formulas written using the \[and \] pair, but produced a picture for display formulas written using a pair of $$. As of September 1, 2022, it inserts LaTeX code in both cases.

14 An Extra Feature from MathJax

Select a mathematical formula in the web version of Fourier and click on it while holding down the control key. A dialog appears as shown below.



..

When we select "TeX Commands", the Latex source for the formula is copied to the clipboard. Here it is:

```
f(x) = {{a_0} \over 2} +
\sum _{k = 1}^\infty \left ( a_k \cos (kx) + b_k \sin (kx) \right )
```

Moreover (and now I am going to use the second application that I promised) we can open LaTeXiT and copy this result to the bottom panel and LaTeX it. We get the original integral, which can be dragged to Word, or Pages, or any other application in standard LaTeXiT manner.

This contextual menu is supplied by MathJax. Notice that the menu follows the Apple Design Guidelines. Presumably the MathJax programmers follow Windows guidelines when running on a windows system, and follow Linux guidelines (if such guidelines exist) on Linux machines.

If we call TeX4ht in one of the ways causing it to output MathML to the html file, then the menu will only offer to supply the MathML code for the formula, which can be several pages long. A crucial advantage of calling TeX4ht with an engine which causes it to output Latex code is that we can then get a copy of that code from this contextual menu.

15 Inspecting Typesetting Engines

There is nothing mysterious about the typesetting engines in TeXShop. All are immediately available for inspection by users. To see the TeX4ht engine, go to the TeXShop menu in TeXShop and select "Open ~/Library/TeXShop." The Finder will show a list of folders. Open the "Engines" folder, find TeX4ht.engine, and open it in TeXShop. Inside you will find the following line:

```
make4ht "$1" "mathjax"
```

This is the line which asks TeX4ht to typeset.

If you want to experiment with the other ways to call TeX4ht mentioned earlier, just edit this file in TeXShop. Even better, duplicate the file, give it a different name, and edit that. Then you can easily switch between one method and another.

16 Linking to Independent Pages

There is an alternate way to create interactive pages; it was the primary method discussed in the previous version of this document. The idea is simple. Add an extra folder named "Interactive" to the source directory. Put independent html pages in this directory and link to them from the main document.

Interactive pages created this way are completely separate from the main Fourier.html. Each is a fully independent web page. TeXShop need not create these pages. To take a particularly outrageous example, there is a project called PreTeXt by Robert Beezer at

the University of Puget Sound. Some University of Oregon faculty use his system, and it is so well-organized that I always recommend that users examine it for inspiration. It is described in detail in the TeXShop 5.0 Change Document. PreTeXt authors write source in xml, and then the source is converted to pdf, html, and other formats. The PreTeXt system comes with several examples including a project called Minimal. I reached into this project and pulled out minimal.html, the html version of this example. I placed this file, and its support files, in Inactive/MinimalExample. Section 10 of our sample document has a link to this page. Select it to see what html pages look like in PreTeXt.

When interactive pages like Minimal require support files, create a subfolder for the interactive page and put the page source and the support files in that subfolder. So for instance, if Exercise1.html requires A.html and B.png, put Exercise1.html, A.html, and B.png in a folder, perhaps named FirstExercise, so the path to Exercise1 is Interactive/FirstExercise/Exercise1.html.

17 Viewing the Source of an Independent Page

There is an easy way to view and edit the source of an independent page. Go to the TeXShop Source Page or the TeXShop Preview Page and click on the page title at the top while holding down the control key. A menu drops down showing the various folders in the path to the source file for Fourier. Click the top item in the list and the Finder will open the folder containing the source. In that source you will find the folder "Interactive", and in that folder you will find the independent source file. Drag this file to the TeXShop icon and drop it there to open the source in a TeXShop edit window.

18 Creating an Independent Page in TeXShop

There is a reason independent pages may be necessary. Some interactive elements may require additional code in the Head of the html document as well as code in the Body. This can be difficult to achieve in TeX4ht, since TeX4ht controls the entire html document, and we only insert pieces of html into that document. If we create an independent page, we control everything.

Suppose then that we want to create a new interactive page from scratch. How do we proceed?

As usual in TeXShop, select "New" in the File menu. A blank source page appears. Go to the "Templates" toolbar and select the template "TeX4ht-Interactive". Some standard html boilerplate appears and then a comment reading "Insert html code here." Erase the comment and start typing html code.

Let's make this simple for our first example. Type

```
 <p> I make this <b> bold assertion </b>. I do not believe that
$$1 + 2 + 3 + \ldots = - {1 \over {12}}$$ </p>
```

Notice that the template has a comment at the top telling TeXShop to typeset with the html engine. So press command-T to typeset.

Since the new source has not been saved, a save dialog appears. Navigate to the Fourier source, and to the Interactive folder in this source. Then give the source an appropriate name, like FirstTry. Before pushing the Save button, notice that the Save dialog has a pull-down menu named "File Format" and by default the format is "TeX File". Change this to "HTML File" and then push Save.

Immediately the file is saved, and just as immediately it is typeset. A web page will appear showing the bold assertion and the mathematical equation we typed at the end.

After this, FirstTry can be edited in the standard way. Type additional html source, and typeset to preview the result. Typesetting doesn't actually typeset — it just opens the html file in our web view.

To link up this document with Fourier, add a line to the Fourier source code, perhaps the code in section 10, reading

```
To read more content, click on \url{Interactive/FirstTry.html}.
```

Try this. Typeset Fourier again and notice that your new page is active in the document.

19 About the Template "TeX4ht-Interactive"

The head portion of the template contains the following lines:

```
<script type="text/javascript" id="MathJax-script" async
src="https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-chtml.js">
</script>
```

These are the lines that activate MathJax for this page.

The head portion of the template also contains the following lines:

```
 <script src="https://sagecell.sagemath.org/static/embedded_sagecell.js"></script>
<script>
    // Make the div with id 'mycell' a Sage cell
    sagecell.makeSagecell({inputLocation:  '#mycell',
                           template:       sagecell.templates.minimal,
                           evalButtonText: 'Activate'});
    // Make *any* div with class 'compute' a Sage cell
    sagecell.makeSagecell({inputLocation: 'div.compute',
                           evalButtonText: 'Evaluate'});
    </script>
```

These are the lines that activate Sage experiments for this page.

20 Additional Interactive Elements for HTML

For serious work, additional interactive html elements are needed. For instance, a common ploy in writing interactive text for students is to write a short set of multiple choice questions. If the student answers correctly, they are encouraged to continue with the next section. If not, a dialog appears explaining why their choice was wrong and suggesting that they review a certain section of the text.

I thought about postponing this document until I had collected html code for this and similar interactions. But I'm not an html expert, and even if I found code that worked, it might be a notorious kludge for a task with a known beautiful solution.

Therefore instead of more samples, TeXShop has an expandable help file which you and other users can improve. Select the item "HTML Commands" in TeXShop Help. The result is shown on the next page.

This has got to be the ugliest help document ever designed, but that's because it contains source code. TeXShop can typeset this page, giving

Help.html

URL:

Bold and Italics

"b" and "i"

It is **crucial** that we follow the guidelines in the *Style Manual*

Inserting a Non-Breaking Space

"nbsp"

We indent this sentence.

Verbatim Text

"pre"

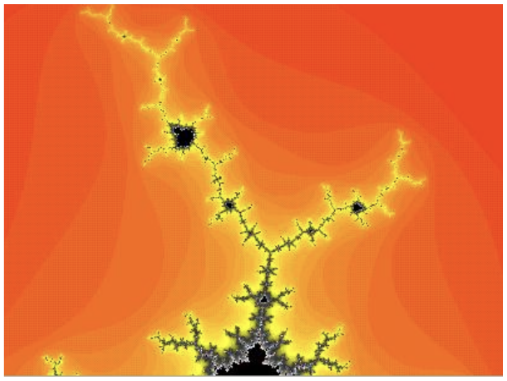
A list in LaTeX is formed with the following code:

```
\begin{itemize}
\item The sky is blue.
\item The grass is green.
\end{itemize}
```

Inserting Images

"img"

Here is a dot near the top of the Mandelbrot set:



Linking to Pdf Documents

"A" and "HREF"

Here are brief notes on the Cayley-Dickson construction and the Octonions: Octonions.pdf

Here is a set of slides at my web site: Slides: Dirac's Belt Trick, Quaternions, and the iPad

Now we have a readable help page. The advantage of this scheme is that you can easily add items to the help page. When you find a new example, add a three part item which lists its purpose, it's tags, and a short example of its use. Typeset to make sure your new example is clear. Keep adding examples as you find useful snippets of html code.

The source for this help document is the file ~/Library/TeXShop/HTML/Help.html. You can also store support files for new items in this folder. Note that the folder already contains a movie, a pdf file, and a jpg file. Your changes are automatically saved in Help.html.

It is easy to share such help documents. Just zip up the folder ~/Library/TeXShop/HTML and send the zip file to others. Feel free to send additions to me.

In this way, we may be able to collate a useful set of interactive elements for users in the future.

21 Putting These Documents on the Web

The file Fourier.pdf is a self-contained document, so it is easy to put it on the web. Upload it to your web site, and add a link to it from an appropriate page.

The TeX4ht project is slightly trickier to put on the web because it is not entirely contained in Fourier.html. The illustrations must be uploaded separately, and the Interactive folder must be uploaded separately. In both cases it is easiest to zip the folders, upload the zip file, and unzip on the server. Finally typesetting creates the file Fourier.css, which is part of the output and must be uploaded to the web. So in summary, the following objects must be uploaded and must live in the same folder on the web server:

- Fourier.html
- Fourier.css
- The entire folder Graphics
- The entire folder Interactive.

22 The Mathematical Moral

Every story should have a moral. This demonstration has two morals, one for mathematicians and one for general users.

There are different ways to teach mathematics. Some mathematicians like to give polished lectures, with every item in the correct place and every proof as elegant as possible. When I was a graduate student, a fellow student was asked in an oral exam "Tell me everything you know about finite fields. And say it in the right order, because you are going to have

to prove it in the order in which you say it." Emil Artin was famous for giving polished lectures.

Other mathematicians like to show students how mathematics is actually created. When I had my first job, a faculty member never prepared graduate lectures. Instead he arrived in class, stated the theorem of the day, and thought out loud as he tried to find a proof strategy. Students told me that he succeeded about 75% of the time.

Still other mathematicians like to make students part of the dialog. "What should we do now?", they ask. And when a student gives bad advice, they say "oh, that won't work because of such and such", but in an extremely friendly manner, so it feels like the subject is being invented on the spot.

A few mathematicians give courses on their current research, and do not know what the next lecture will be about.

Which strategy is correct? They all are correct. The wonderful thing about a University is that you have contact with many different ways to think about mathematics, each valuable.

I like to think of the pdf document as the final polished result. And I like to think of the html document as embodying the way mathematics is actually invented, using experiments, trials, false leads, and all the rest. It is very useful to have both documents, the html to remind us of the joy of discovery, and the pdf to remind us of the logical and elegant final result.

23 The Moral for Computing

The world has thousands and thousands of smart engineers, involved in projects which will take decades to reach maturity. Some of this work is open source; some is commercial; some cannot be categorized. There's no need to gallup after every new idea, but keep paying attention.

I saw TeX4ht in it's youth and dismissed it. A few months ago, I tried it again ... and realized that something fantastic had happened. It wasn't that Eitan had missed the boat; his foundational work was all important. But TeX4ht requires MathML, it requires MathJax, and it requires the social insights that Covid forced on the world.

I watched people write MathML by hand, with giant formulas that spread across pages and pages, and it made no sense to me. Today, twenty years later, I cannot write a single MathML tag. But somehow, mysteriously, that project became essential.

I don't recall even hearing about jsMath. Instead I noticed that the mathematics on the web was improving. It didn't occur to me to ask why.

When I heard about Sage, I installed it, discovered that Sage code could be added to TeX documents, and played with it for several days. I kept using Mathematica for illustrations. Fifteen years later, I tried Sage on the web and got a graph. I said "nice". Then I got a 3D graph, and when I grabbed it, the graph rotated. I said "Wow, this is unbelievable!"

In the TeX world, this lesson should be second nature. We live with the Knuth miracle, and with the work of thousands who write packages, and use their work every day. But still, when something new becomes essential it is a surprise. When XeTeX showed that any language could be typeset with LaTeX, that was a revolution.

Around 1990, LaTeX made its way to the Mac because engineers wrote an editor by hand, and then wrote a dvi viewer by hand, and then modified the compiler to output faster TeX. I am in awe of those folks.

But later, the engineers at NeXt made it possible for everyone to do such tasks, and this culture spread to Apple. I could create TeXShop because Apple handed me the API for a pre-built editor, and then handed me the API for a pre-built pdf viewer, and then handed me the API to call command line programs with two lines of code.

When the web became important, every computer manufacturer built a browser. So did Apple. But then they handed me the API for a pre-built html viewer.

Pay attention. Pay attention.