Course template

Tunç Başar Köse

May 14, 2024

CONTENTS

This is a `jupyter-book`-based template for creating course notes and websites. If you are not familiar with `jupyter-book`, I recommend skimming through their tutorial here. This template shows off a number of features one can use in creating lecture notes/websites. However, to get detailed information, check out the linked references, including

- Jupyter Book
- MyST Markdown.

Check out the pages below.

Part I

Tooling

CHAPTER

ONE

SOURCE FILES AND OUTPUTS

The basic idea of this template is to organize the content of some course using Markdown files and presented it as a reasonably nice looking website. These webpages were all built from Markdown files, some of which are regular and others containing executed code and interactive plots.

1.1 Source files

I assume you have a general idea of what Markdown is and what you can do with it, if not this can give you some idea.

Markdown is nice and its basic features are often enough to organize information such as course notes or websites. Most notably however, executing code and displaying its output is not one of these features. To solve this problem, often people use something like Jupyter Notebooks or Quarto to have code alongside Markdown. In `jupyter-book` projects, MyST-Markdown (standing for Markedly Structured Text) is used, which extends Markdown with

- the ability to encode notebooks as Markdown files (using `MyST-NB`), so we get the niceties of working with code without the annoyance of the notebook file changing its state everytime you open it/execute some cell (preserving the sanity of `git status`), and

> Sphinx is a powerful documentation generator.

- Sphinx compatibility, which allows for a high level of customizability in the output.

To embed a notebook in a MyST-Markdown file, one needs a YAML top-matter. To create it easily, one can create the Markdown file themselves, and ask `jupyter-book` to genereate the top-matter:

```
jupyter-book myst init mymarkdownfile.md --kernel kernelname
```

Important: Every file must have a title, which usually means it should begin with a top-level header `#`, and only one top-level header should be present. Also, don't skip levels, i.e. don't jumpt from `#` to `###`.

1.2 Outputs

This template is primarily geared towards producing HTML files to be served as a static website. `jupyter-book` also allows one to build either a single PDF containing everything (via LaTeX or HTML using `pyppeteer`) or individual PDF files for each Markdown file (LaTeX only). You can read more about this and more here.

1.3 Further reading

- Jupyter Book pages on source files and oublishing
- MyST Markdown
- Sphinx documentation for advanced use.

CHAPTER

TWO

USE AND CONFIGURE

Two very important files in a `jupyter-book` project are `_config.yml` and `_toc.yml`. The former is used to configure `jupyter-book`, where one can control simple books settings like title and author, but also more complicated ones like notebook caching and Sphinx extensions.

For example, the `_config.yml` of this repo contains the setting that uses the QuantEcon theme for the generated HTML. It also explicitly limits the building to files included in the table of contents, so files don't accidentally get published.

The `_toc.yml` file contains the table of contents that determines the structure of the output. This website is organized as follows

```
format: jb-book
root: intro
parts:
- caption: Tooling
  chapters:
  - file: tooling/input_and_output
  - file: tooling/config
- caption: Content
  chapters:
  - file: content/markdown_and_sphinx
  - file: content/math_and_code
  - file: content/interactivity
- caption: Other
  chapters:
  - file: other/about
```

which should reflect the table of contents you see on the website, albeit the titles are slightly different. This is a reasonable, but not the sole way to organize the structure. For more examples, see here.

Note: To add a new page, create a file in the `course` directory and add it to `_toc.yml`.

2.1 Further reading

- More on HTML theming here and here.

CHAPTER

THREE

NOTES ON DESIGN

There are two issues to consider here that work separately: the design of the website and the design of the PDF produced (assuming the latter happens via LaTeX).

Currently, the PDF design follows the default options.

The website uses QuantEcon theme for the generated HTML. But there are a couple things to note:

- The logo on the right margin (page-toc) is easily changeable in `_config.yml`.
- The logo on the toolbar was originally a logo of QuantEcon, hardcoded into the CSS files provided by the theme. You can change it by providing custom CSS with the `!important` keyword, an example is included in `_static/style.css`.
- There is a button included change the contrast/make it dark mode, but it seems to not work with admonitions (see *Math, Code Formatting and Execution*).
- To make the Github button work, make sure you don't rename the folder if you clone the repo.
  - Why? Because this function is checking for that, and gets used here to create the hyperlink for the Github button.

3.1 List of not-obvious modifications

- `_static/style.css` replaces the toolbar logo with `_static/logo.png`
- `_static/remove_pdf_button.js` removes the "Download PDF" button from pages.

CHAPTER

FOUR

BUILDING A PDF

With the default options, the PDF output is not perfect. For example, margins and code-blocks/cells look weird, and admonitions could use some coloring. It is possible to customize these and more via the Sphinx LaTeX options, but this may take some time/effort.

It would make things easier to have access to the LaTeX styling of a pre-existing good looking document. For example, I tried to find something like that for the Algorithms for Optimization book, but the authors have not made it available, as far as I could tell.

Part II

Content

CHAPTER

FIVE

MYST AND SPHINX

There are a lot of features that may come up when designing notes/webpages. In writing this template so far, I've used notes on margins, highlighted text (also called admonitions) and synced tabs. These are often implemented via roles and directives. Which are inline or block-level instructions respectively.

See here for a more detailed explanation, here for examples in MyST and here for Sphinx. Lastly, see here for more MyST features in general.

CHAPTER

SIX

MATH, CODE FORMATTING AND EXECUTION

6.1 Math

There are multiple ways one can include mathematical expressions in MyST-Markdown.

6.1.1 Sphinx role/directive

One is using the `math` role or directive from sphinx to create inline and block math:

````
Let {math}`f(x_1,x_2)` represent our objective function.
```{math}
:label: problem_1
\min_{x_1,x_2} \quad& f(x_1,x_2) \\
\text{s.t.} \quad& x_1\geq 0 \\
& x_2 \geq 0 \\
& x_1+x_2 \leq 0
```
The label option lets you refer to {eq}`problem_1`.
````

will result in:

Let $f(x_1, x_2)$ represent our objective function.

$$\begin{aligned} \min_{x_1,x_2} \quad & f(x_1,x_2) \\ \text{s.t.} \quad & x_1 \geq 0 \\ & x_2 \geq 0 \\ & x_1 + x_2 \leq 0 \end{aligned} \tag{6.1}$$

The label option lets you refer to (6.1).

6.1.2 LaTeX-style math

Having enabled `amsmath` in `_config.yml`, one can also use the math environments

> equation, multline, gather, align, alignat, flalign, matrix, pmatrix, bmatrix, Bmatrix, vmatrix, Vmatrix, eqnarray and the unnumbered equivalents with `*`.

Warning: `\label`s inside these environments may not work properly, so if you'd like to refer to equations, prefer the directive described above.

6.1.3 More

For formatting mathematical text like definitions and proofs, see here.

6.2 Code

6.2.1 Formatted Code

Syntax highlighting is simple:

````
```python
import numpy as np
a = np.random.random()
```
````

gives

```python
import numpy as np
a = np.random.random()
```

You can even have nice synced tabs like this:

Julia

```julia
function square_list(l)
    l .^ 2
end
```

Python

```python
def square_list(l):
    return [x**2 for x in l]
```

Julia

```julia
function add_one(x)
    x+1
end
```

Python

```
def add_one(x):
    return x+1
```

Look at the Markdown file of this page to see how.

6.2.2 Executable Code

Attention: Don't forget to make sure the source file contains the MyST-Markdown top-matter if you want to execute code.

Code to be executed should be included in `{code-cell}`s.

````
```{code-cell}
1+1
```
````

gives

```
1+1
```

```
2
```

Important: While we are working with Markdown files, at the end of the day these are executed while being interpreted as notebooks. Thus, you can only have one kernel associated with a file, which is defined in the top-matter. One cannot execute Python code in one block and Julia in another in the same file.

There are also lots of options to work with `code-cell`s, such as hiding the input, making cells scrollable, and formatting the outputs. You can read more here.

Note: One thing I encountered while preparing this was that when working with Julia, if you create some files but then update Julia version, one may need to rebuild `IJulia` to get things working again.

6.3 Further reading

- Math pages from Jupyter Book, MyST-Markdown, and Sphinx.
- More on syntax highlighting, including numbering and highlighting lines, including code from files and inline highlighting.

CHAPTER

SEVEN

INTERACTIVE PAGES

7.1 Letting users execute

One has multiple options to make things interactive when this is deployed as a website. One idea is to link the pages to interactive computing interfaces like Binder or JupyterHub to allows easy transitioning into a coding environment where people can play around with it themselves. It is also possible to execute code without leaving the page, using services like Thebe. For more on both, read here and here.

7.2 Interactive plots

Another idea that may be nice for an optimization course is interactive plots. In my googling around, the term "interactive plot" seems to cover a more general concept than what I had in mind. For example, there are a lot of libraries in various languages that are interactive in the sense that one can filter data and change aesthetic attributes. Actually changing what is plotted, for example by specifying a new slope for a line or changing the feasible region by adding a constraint, is more difficult because it often requires recomputing some function being plotted. To do this easily seems to require some amount of JavaScript one way or another.

7.2.1 In Python

One nice option is `bokeh`.

Example from here. On my machine, this was working in Chrome but not in Firefox for some reason.

The `output_notebook()` function is needed for interactivity, but the code block could be hidden in actual lecture notes if needed.

```
import numpy as np

from bokeh.layouts import column, row
from bokeh.models import ColumnDataSource, CustomJS, Slider
from bokeh.plotting import figure, show, output_notebook

output_notebook()
```

```
x = np.linspace(0, 10, 500)
y = np.sin(x)
```

(continues on next page)

(continued from previous page)

```
source = ColumnDataSource(data=dict(x=x, y=y))

plot = figure(y_range=(-10, 10), width=400, height=400)

plot.line('x', 'y', source=source, line_width=3, line_alpha=0.6)

amp = Slider(start=0.1, end=10, value=1, step=.1, title="Amplitude")
freq = Slider(start=0.1, end=10, value=1, step=.1, title="Frequency")
phase = Slider(start=-6.4, end=6.4, value=0, step=.1, title="Phase")
offset = Slider(start=-9, end=9, value=0, step=.1, title="Offset")

callback = CustomJS(args=dict(source=source, amp=amp, freq=freq, phase=phase,␣
↪offset=offset),
                    code="""
    const A = amp.value
    const k = freq.value
    const phi = phase.value
    const B = offset.value

    const x = source.data.x
    const y = Array.from(x, (x) => B + A*Math.sin(k*x+phi))
    source.data = { x, y }
""")

amp.js_on_change('value', callback)
freq.js_on_change('value', callback)
phase.js_on_change('value', callback)
offset.js_on_change('value', callback)

show(row(plot, column(amp, freq, phase, offset)))
```

7.2.2 In Julia

I tried getting `PlotlyJS` to work, using an example from here.

I'm not sure if this is usable with jupyter-book. Currently this is not working with the latest version of JupyterLab, due to an outdated extension (`webio_jupyterlab_extension`). Presumably it would work with an earlier version of JupyterLab (or nbclassic or jupyter notebook), but that doesn't explain why jupyter-book is not working. Somehow, it worked on a ipynb in VSCode though.

There are other Julia packages, as described in `Plots.jl` documentation, but I haven't tried them yet.

7.2.3 In JavaScript

At one point, it may be the easiest to just do it directly in JavaScript, using probably `Plotly`.

7.3 Further reading

- Jupyter Book page on interactive plots

CHAPTER

EIGHT

INTERACTIVE PLOTS IN JULIA

As a simple example, one can rotate and zoom in/out with 3D plots.

```
using WGLMakie, Bonito
Page(exportable=true, offline=true)
WGLMakie.activate!()

N = 60
function xy_data(x, y)
    r = sqrt(x^2 + y^2)
    r == 0.0 ? 1f0 : (sin(r)/r)
end
l = range(-10, stop = 10, length = N)
z = Float32[xy_data(x, y) for x in l, y in l]
surface(
    -1..1, -1..1, z,
    colormap = :Spectral
)
```

```
App() do session
    s = Slider(1:5, startvalue=2)
    value = map(s.value) do x
        return x ^ 2
    end
    # Record states is an experimental feature to record all states generated in the␣
 ↪Julia session and allow the slider to stay interactive in the statically hosted␣
 ↪docs!
    return Bonito.record_states(session, DOM.div(s, value))
end
```

```
App(Bonito.var"#8#14"{var"#1#3"}(var"#1#3"()), Base.RefValue{Union{Nothing,␣
 ↪Session}}(nothing), "Bonito App", false)
```

```
app = App() do session
    slider = Slider(1:10)
    slider[] = 2  # startvalue doesn't work, do it manually
    slider_val = DOM.p(slider[])

    onjs(session, slider.value, js"""function on_update(new_val) {
        const p_element = $(slider_val)
        p_element.innerText = new_val
```

(continues on next page)

(continued from previous page)

```
    }

    """)

    Bonito.on_document_load(session, js"""
        const p_element = $(slider_val)
        console.log("slider: ", p_element.innerText)
    """)

    return DOM.div(slider, slider_val)
end
```

```
App(Bonito.var"#8#14"{var"#5#6"}(var"#5#6"()), Base.RefValue{Union{Nothing,␣
↪Session}}(nothing), "Bonito App", false)
```

These sliders can be tied to various aspects of the plot (found under `geometry` or `material`). In the below graph, the slider value is used as the slope. The Javascript code calculates a new end-point for the line segment and updates it. There is also a text box but it is not doing anything yet.

```
app = App() do session
    text = TextField("test")
    slider = Slider(1:10, startvalue = 2)
    slider_val = DOM.p(slider[])
    fig, ax, plot = ablines(0,2)

    onjs(session, text.value, js"(v) => console.log(v)")
    onjs(session, slider.value, js"""function on_update(new_val) {
        $(plot).then(plots=>{
            const abline = plots[0]
            const x = abline.geometry.attributes.linepoint_end.data.array[4]
            abline.geometry.attributes.linepoint_end.data.array[5] = x*new_val
            abline.geometry.attributes.linepoint_end.needsUpdate = true
        })
    }
    """)
    onjs(session, slider.value, js"""function on_update(new_val) {
        const p_element = $(slider_val)
        p_element.innerText = new_val
    }

    """)

    Bonito.on_document_load(session, js"""
        $(plot).then(plots=>{
            const abline = plots[0]
            console.log(abline)
        })

        const p_element = $(slider_val)
        console.log("slider: ", p_element.innerText)
    """)

    return DOM.div(text, slider, slider_val, fig)
end
```

```
App(Bonito.var"#8#14"{var"#7#8"}(var"#7#8"()), Base.RefValue{Union{Nothing,␣
↪Session}}(nothing), "Bonito App", false)
```

Here the slider moves the first point.

```
app = App() do session::Session
    s1 = Slider(1:100)
    slider_val = DOM.p(s1[])
    fig, ax, splot = scatter(1:4)

    Bonito.on_document_load(session, js"""
        $(splot).then(plots=>{
            const scatter_plot = plots[0]
            console.log(scatter_plot)
        })
    """)

    onjs(session, s1.value, js"""function on_update(new_value) {
        $(splot).then(plots=>{
            const scatter_plot = plots[0]
            scatter_plot.geometry.attributes.pos.array[0] = (new_value/100) * 4
            scatter_plot.geometry.attributes.pos.array[1] = (new_value/100) * 4
            scatter_plot.geometry.attributes.pos.needsUpdate = true
        })
    }
    """)
    return DOM.div(s1, fig)
end
```

```
App(Bonito.var"#8#14"{var"#9#10"}(var"#9#10"()), Base.RefValue{Union{Nothing,␣
↪Session}}(nothing), "Bonito App", false)
```

8.1 Further reading

- WGLMakie docs

Part III

Other

CHAPTER

NINE

ABOUT THIS COURSE

9.1 Acknowledgements

This website is based on Quantitative Economics with Julia and makes use of their `quantecon-book-theme` package.