Computational Publishing for Collections

Simon Worthington

9/11/22

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

2 Summary

In summary, this book has no content whatsoever.

3 Quarto Basics

For a demonstration of a line plot on a polar axis, see Figure 3.1.

```
import numpy as np
import matplotlib.pyplot as plt

r = np.arange(0, 2, 0.01)
theta = 2 * np.pi * r
fig, ax = plt.subplots(
  subplot_kw = {'projection': 'polar'}
)
ax.plot(theta, r)
ax.set_rticks([0.5, 1, 1.5, 2])
ax.grid(True)
plt.show()
```

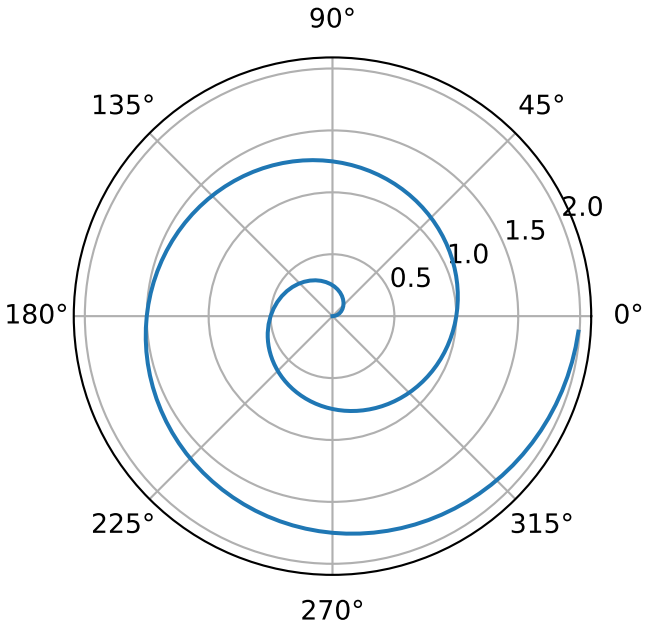


Figure 3.1: A line plot on a polar axis

4 Quarto Computations

```
import numpy as np
a = np.arange(15).reshape(3, 5)
a
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

4.1 Matplotlib

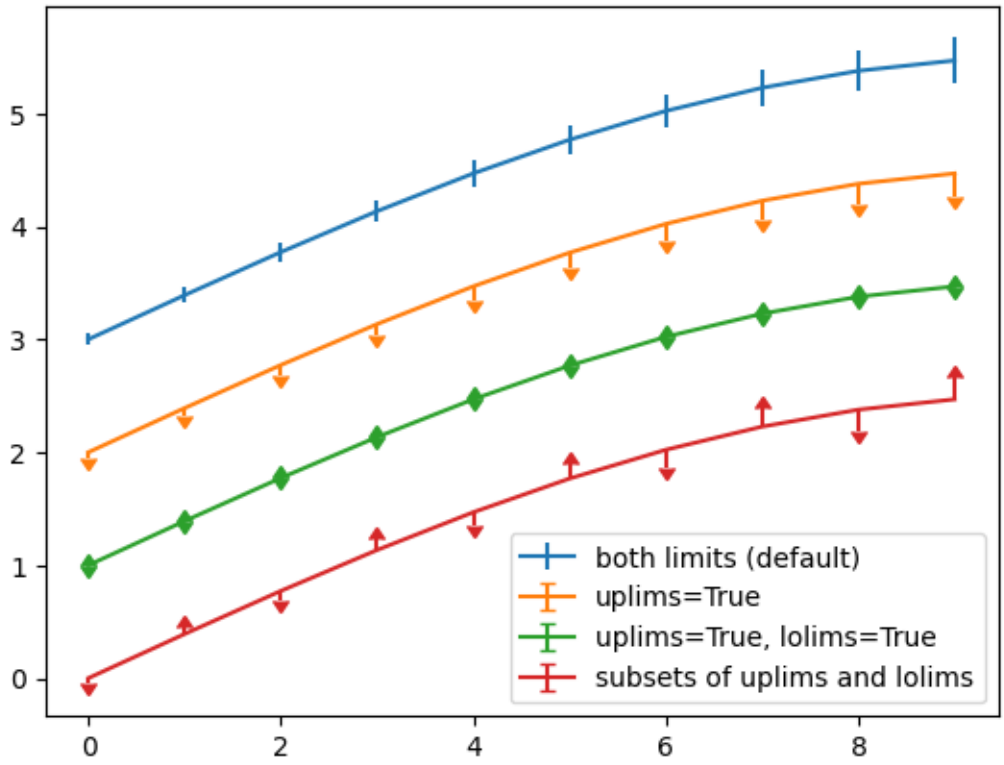
```
import matplotlib.pyplot as plt

fig = plt.figure()
x = np.arange(10)
y = 2.5 * np.sin(x / 20 * np.pi)
yerr = np.linspace(0.05, 0.2, 10)

plt.errorbar(x, y + 3, yerr=yerr, label='both limits (default)')
plt.errorbar(x, y + 2, yerr=yerr, uplims=True, label='uplims=True')
plt.errorbar(x, y + 1, yerr=yerr, uplims=True, lolims=True,
             label='uplims=True, lolims=True')

upperlimits = [True, False] * 5
lowerlimits = [False, True] * 5
plt.errorbar(x, y, yerr=yerr, uplims=upperlimits, lolims=lowerlimits,
             label='subsets of uplims and lolims')

plt.legend(loc='lower right')
plt.show(fig)
```



4.2 Plotly

```
import plotly.express as px
import plotly.io as pio
gapminder = px.data.gapminder()
gapminder2007 = gapminder.query("year == 2007")
fig = px.scatter(gapminder2007,
                 x="gdpPercap", y="lifeExp", color="continent",
                 size="pop", size_max=60,
                 hover_name="country")
fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json
```

5 Image from linked open data API

5.0.1 Image - Siege III: The Fortress of Raab occupied by the Turks, 1594. Painting, https://wikibase.wbworkshop.tibwiki.io/wiki/Item:Q505

The below Python code experiments with retrieving data from Wikibase using the API. This takes approx. 18 seconds to run due to the size of the images.

```
from PIL import Image
import requests

# Global variables
endpoint_url = 'https://wikibase.wbworkshop.tibwiki.io'
resource_url = '/w/api.php'
entity_id = 'Q505'

def get_entity (entity_id):
    resourceUrl = '/w/api.php?action=wbgetentities&format=json&ids='+entity_id
    uri = endpoint_url + resourceUrl
    r = requests.get(uri)
    data = r.json()
    return data

media_data = get_entity(entity_id)
claims = media_data['entities'][entity_id]['claims']
for property, values in claims.items():
    if property == 'P22':
        for value in values:
            image_url = value['mainsnak']['datavalue']['value']
            im = Image.open(requests.get(image_url, stream=True).raw)
    elif property == 'P23':
        for value in values:
            entity_id = value['mainsnak']['datavalue']['value']['id']
            object_data = get_entity(entity_id)
print('English title: ', object_data['entities'][entity_id]['labels']['en']['value'])
display(im)
```

English title: Siege III: The Fortress of Raab occupied by the Turks, 1594


Vestung Raab wie die vom Türcken
belegert gewesen. Ao: 1594

6 Linked open data query from SPARQL

The below Python code experiments with using SPARQLWrapper to retrieve data from NFDI4Culture's Wikibase based on a SPARQL query.

```
from SPARQLWrapper import SPARQLWrapper, JSON
from PIL import Image
from ipypublish import nb_setup
import numpy as np
import pandas
import requests
import sparql_dataframe

def get_delay(date):
    try:
        date = datetime.datetime.strptime(date, '%a, %d %b %Y %H:%M:%S GMT')
        timeout = int((date - datetime.datetime.now()).total_seconds())
    except ValueError:
        timeout = int(date)
    return timeout

def get_image(url, headers):
    r = requests.get(url, headers=headers, stream=True)
    if r.status_code == 200:
        im = Image.open(r.raw)
        return im
    if r.status_code == 500:
        return None
    if r.status_code == 403:
        return None
    if r.status_code == 429:
        timeout = get_delay(r.headers['retry-after'])
        print('Timeout {} m {} s'.format(timeout // 60, timeout % 60))
        time.sleep(timeout)
        get_image(url, headers)
```

```
# Example using NFDI4Culture Wikibase

# Specify the NFDI4Culture Wikibase SPARQL endpoint
endpoint_url = 'https://query.wbworkshop.tibwiki.io/proxy/wdqs/bigdata/namespace/wdq/sparq

# Query
query = """
SELECT ?item ?itemLabel ?creationDateStart ?creationDateEnd ?inscription ?materialLabel ?m
WHERE {
    ?item tibt:P25 tib:Q60.
    ?item tibt:P79 ?creationDateStart.
    ?item tibt:P80 ?creationDateEnd.
    ?item tibt:P18 ?inscription.
    ?item tibt:P38 ?material.
    ?item tibt:P39 ?method.
    ?item tibt:P26 ?description.
    ?item tibt:P16 ?media
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
}
#All paintings from "Belagerungsszenen des Langen Türkenkriegs"
"""

dataframe = sparql_dataframe.get(endpoint_url, query, post=True)

dataframe

# Example using Wikidata

# Specify the Wikidata SPARQL endpoint
#endpoint_url = 'https://query.wikidata.org/bigdata/namespace/wdq/sparql'

#user_agent = 'cp4c_bot/0.0 (https://github.com/SimonXIX/cp4c; ad7588@coventry.ac.uk)'
#sparql = SPARQLWrapper(endpoint_url, agent=user_agent)

# query = """
# #Cats, with pictures
# #defaultView:ImageGrid
# SELECT ?item ?itemLabel ?pic
# WHERE
# {
# ?item wdt:P31 wd:Q146 .
```

```
# ?item wdt:P18 ?pic
# SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en" }
# }
# LIMIT 2
# """

# Retrieve results and convert to JSON format
#sparql.setQuery(query)
#sparql.setReturnFormat(JSON)
#result = sparql.query().convert()

#for item in result['results']['bindings']:
    # image_url=item['pic']['value']
    # headers = {'User-Agent': 'cp4c_bot/0.0 (https://github.com/SimonXIX/cp4c; ad7588@cov
    # im = get_image(image_url, headers)

    # print('Cat name: ', item['itemLabel']['value'])
    # display(im)
```

	item	itemLabel	c
0	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege VII: The city of Waitzen occupied by the...	1
1	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege I: The Fortress of Tottis retaken by the...	1
2	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege II: The Fortress of Gran occupied by the...	1
3	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege XII: The Fortress of Gran occupied by th...	1
4	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege XI: The capital city Offen occupied by t...	1
5	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege III: The Fortress of Raab occupied by th...	1
6	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege IV: The Fortress of Comorna occupied by ...	1
7	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege V: The Fortress of Gran retaken by the C...	1
8	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege VI: The Fortress of Vizzegrad occupied b...	1
9	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege VIII: The Fortress of Raab, which was re...	1
10	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege IX: The capital city Offen occupied by t...	1
11	https://wikibase.wbworkshop.tibwiki.io/entity/...	Siege X: The capital city Offen occupied by th...	1

7 3D model with annotations

This page shows a 3D model .obj file using the obj2html Python library found at https://z-uo.medium.com/visualize-3d-model-in-jupyter-notebook-e5a9deca20c6. This converts a .obj file to HTML and then displays the HTML.

This is currently using a 9.9 MB test model object because the dining room 3D model is 148.6 MB and is currently not working with this script.

```
from obj2html import obj2html
from IPython.display import display, HTML

obj2html('model.obj', 'model.html')

#obj2html('Tafelstube_Final_4_JL_US.obj', 'model.html')
display(HTML('model.html'))
```

```
<IPython.core.display.HTML object>
```

The below code performs a SPARQL query to get the annotations for a specific target entity, in this case Q446 'Weikersheim, Dining room CAD model' (https://wikibase.wbworkshop.tibwiki.io/wiki/Item:Q44

```
import sparql_dataframe

# Specify the NFDI4Culture Wikibase SPARQL endpoint
endpoint_url = 'https://query.wbworkshop.tibwiki.io/proxy/wdqs/bigdata/namespace/wdq/sparq

# Query
query = """
SELECT ?annotation ?annotationLabel ?description ?title
WHERE {
    ?annotation tibt:P50 tib:Q446.
    ?annotation schema:description ?description.
    ?annotation tibt:P30 ?title
    SERVICE wikibase:label { bd:serviceParam wikibase:language "[AUTO_LANGUAGE],en". }
}
```

```
#All annotations for a specific target entity"
"""

dataframe = sparql_dataframe.get(endpoint_url, query, post=True)

dataframe
```

	annotation	annotationLabel	description	title

8 Embedded video

The below Python code experiments with retrieving video data via iframe embedding.

```
from IPython.display import HTML
HTML("""
<iframe width="560" height="315" scrolling="no" src="https://av.tib.eu/player/56162" frame
""")
```

```
<IPython.core.display.HTML object>
```

9 Linked open data API testing

This notebook experiments with querying linked open data in a Jupyter Notebook rendered through Quarto.

The Python code below queries the NFDI4Culture Wikibase at https://wikibase.wbworkshop.tibwiki.io/wiki/Ma and returns data based on the ID inputted by the user.

This executes in Jupyter Notebook which is able to run the Python code and provides a static output when saved in the Notebook. It cannot be executed dynamically in Quarto since the Quarto front-end does not support stdin input requests.

To reset the output, run 'Kernel > Restart kernel and clear all outputs'.

9.1 Wikibase API testing

```
import requests

# Global variables
endpoint_url = 'https://wikibase.wbworkshop.tibwiki.io'
resource_url = '/w/api.php'

entity = input("What's the Q number (including the 'Q')? ")
print('Check out ' + endpoint_url + '/wiki/' + entity + ' to see the GUI.')
resourceUrl = '/w/api.php?action=wbgetclaims&format=json&entity='+entity
uri = endpoint_url + resourceUrl
r = requests.get(uri)
data = r.json()
claims = data['claims']
print('subject: ', entity)
print()
for property, values in claims.items():
    print('property: ', property)
    for value in values:
        try:
            # print Q ID if the value is an item
```

```
            print('item value: ', value['mainsnak']['datavalue']['value']['id'])
        except:
            try:
                # print the string value if the value is a literal
                print('literal value: ', value['mainsnak']['datavalue']['value'])
            except:
                # print the whole snak if the value is something else
                print('other value: ', value['mainsnak'])
    print()
```

10 Query ORCID for works authored by a person

Taken from Project TAPIR's repository of Jupyter Notebooks: https://github.com/Project-TAPIR/pidgraph-notebooks

This notebook queries the ORCID Public API to retrieve works listed in a person's ORCID record. It takes an ORCID URL or iD as input to retrieve the ORCID record of a person and the works listed on it. From the resulting list of works we output all DOIs.

```
# Prerequisites:
import requests                          # dependency to make HTTP calls
from benedict import benedict            # dependency for dealing with json
```

The input for this notebook is an ORCID URL or iD, e.g. '`https://orcid.org/0000-0003-2499-7741`' or '`0000-0003-2499-7741`'.

```
# input parameter
example_orcid="https://orcid.org/0000-0002-2437-589X"
```

We use it to query ORCID's Public API for the person's metadata and all works connected to them.

```
# URL for ORCID API
ORCID_RECORD_API = "https://pub.orcid.org/v3.0/"

# query ORCID for an ORCID record
def query_orcid_for_record(orcid_id):

    response = requests.get(url=requests.utils.requote_uri(ORCID_RECORD_API + orcid_id),
                            headers={'Accept': 'application/json'})
    response.raise_for_status()
    result=response.json()
    return result
```

```
#-- example execution
orcid_id=example_orcid.replace("https://orcid.org/", "")
orcid_record=query_orcid_for_record(orcid_id)
# uncomment next lines to see complete metadata for given ORCID
#import pprint
#pprint.pprint(orcid_record)
```

From the complete ORCID metadata we extract the works section and print out title and DOI of each first `work-summary` (the first item in a personal information section has the highest display index).

Note: works that do not have a DOI assigned, will not be printed.

```
# extract works section from ORCID profile
def extract_works_section(orcid_record):
    orcid_dict=benedict.from_json(orcid_record)
    works=orcid_dict.get('activities-summary.works.group') or []
    return works

# for each work in the work section: extract title and DOI
def extract_doi(work):
    work_dict=benedict.from_json(work)
    title=work_dict.get('work-summary[0].title.title.value')
    dois= [doi['external-id-value'] for doi in work_dict.get('work-summary[0].external-ids
    # if there is a DOI assigned to the work, the list of dois is not empty and we can ext
    doi=dois[0] if dois else None
    return doi, title


# ---- example execution
works=extract_works_section(orcid_record)
for work in works:
    doi,title = extract_doi(work)
    if doi:
        print(f"{doi}, {title}")
```

```
10.21428/785a6451.af466093, What is computational publishing?
10.21428/785a6451.1792b84f, Books Contain Multitudes
10.21428/785a6451.0ed93b68, Part 1: Mapping & Situating Experimental Books
10.21428/785a6451.cd58a48e, Part 2: A Typology of Experimental Books
10.21428/785a6451.c4d3ffa1, Introducing COPIM's new website
10.17613/d4rs-rx03, Posthumanism in Outer Wilds
```

10.17613/5SJE-W447, New Sincerity, the Weird, and the post-ironic turn in contemporary indie
10.20944/preprints202001.0240.v1, Open or Ajar? Openness within the Neoliberal Academy

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.