Hitchhikers guide to transparent, reproducible, and collaborative computational science

Wolfram Barfuss

2024-04-23

Table of contents

1 Why this guide?

What is this?

1.0.1 Science as amateur software development

Figure 1.1: Everything Is AWESOME

Science is one of humanity's greatest inventions. Academia, on the other hand, is not. It is remarkable how successful science has been, given the often chaotic

habits of scientists. In contrast to other fields, like say landscaping or software engineering, science as a profession is largely *unprofessional*—apprentice scientists are taught less about how to work responsibly than about how to earn promotions. This results in ubiquitous and costly errors. Software development has become indispensable to scientific work. I want to playfully ask how it can become even more useful by transferring some aspects of its professionalism, the day-to-day tracking and back-tracking and testing that is especially part of distributed, open-source software development. Science, after all, aspires to be distributed, open-source knowledge development.

1.0.2 Elements

- Computational sciences develop models and analyze data.
- Transparency
- Reproducible
- Collaborative

- It is a hitchhiker's guide because many of the elements we use are very common practices. We don't need to drive ourselves.

What do we want?

- Publications (as a PDF, on the web, with images)
  - Cross-referencing
  - Citations
- Execute and reuse code
- Presentations
- Unit tests
- Animations
- Version control

Tools: How do we get it?

- Jupyter notebooks
- quarto
- nbdev
- Jupyter lab (extensions)
- git & github

2 Scholarly writing

In this note, we will cover the basics of scholarly writing. That is the combination of a - narrative in text form, - possibly including images, - with cross-references throughout the text, - while giving attribution to other authors,

2.1 Narrative text with images and equations

Writing text is obvious and follows basic markdown syntax. More information about that is in the Quarto documentation.

2.1.1 Images

To include images, standard markdown syntax works well

```
![Abstract Shape](graphics/drawing.svg)
```

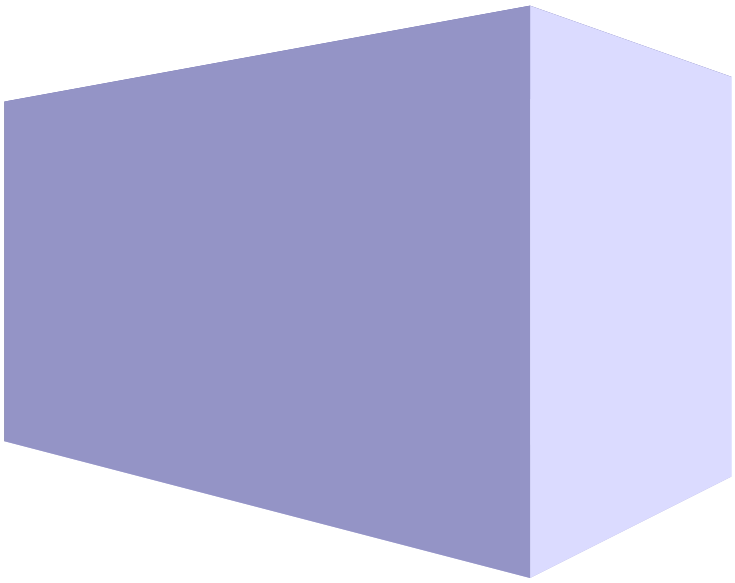


Figure 2.1: Abstract Shape

I recommend working with vector graphics (`.svg` files). They do not require an additional raw file, which keeps the project folder clean. Inkscape is a great open-source vector graphics program.

Quarto offers more customization options to images, such as downscaling the image size or specifying the figure alignment,

```
![Abstract Shape](graphics/drawing.svg){width=50%, fig-align="right"}
```

which results in

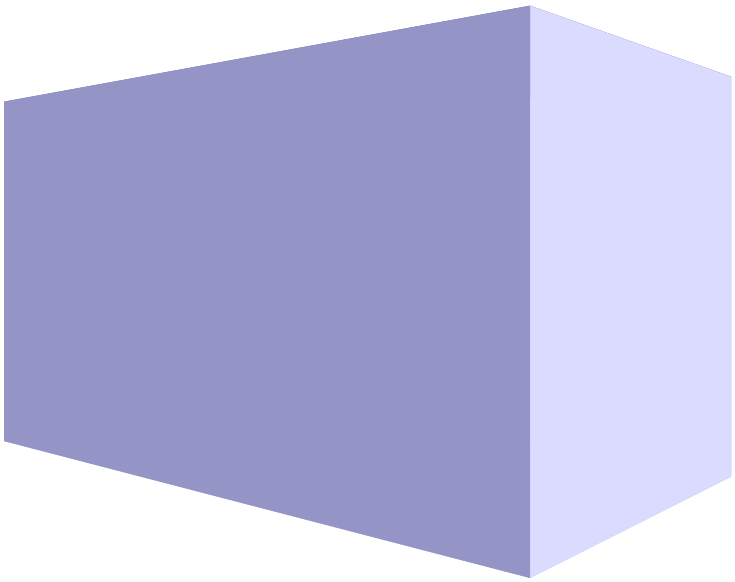


Figure 2.2: Small Abstract Shape

However, these additions to the standard markdown syntax are Quarto-specific, which get rendered in the output files produced by Quarto, but not necessarily while displaying the raw Jupyter notebook.

> **Note**
>
> By adopting a large-to-small width-to-length ratio for our raw images, such as 16-to-9, we can prioritize transparency, collaboration, and reproducibility. This approach allows us to cleanly display our Jupyter notebooks in other places, such as GitHub, NBViewer, or Google Colab.

2.1.2 Equations

Equations follow standard LaTeX syntax, inline by `$ ... $` or in full display by

`$$ ... $$`

For example, $E = mc^2$, or

$$E = mc^2 \tag{2.1}$$

> **Note**
>
> Sometimes, the JuptyerLab Quarto extension has problems rendering equations in my setup. I haven't understood the exact cause of this behavior yet. Reloading the python environment did the trick (once). However, when using the LaTex `align` environment for more complex equations, I couldn't get the Quarto extension to display it correctly. To get most of the other displaying features, I can recommend the MyST Jupyter extension as an alternative.

2.2 Cross-references

Cross-references help readers navigate your document by providing numbered references and hyperlinks to entities like figures and tables. Each entity needs a unique label, e.g. `#fig-element`, to be cross-referenced.

2.2.1 Figures

Figure receive a label by, for example,

```
![Referenced Abstract Shape](graphics/drawing.svg){#fig-shape}
```

The `#fig-shape` label makes the figure referenceable.

```
See @fig-shape for an illustration.
```

See Figure 2.3 for an illustration.

2.2.2 Sections

To reference a section, add a `#sec-` identifier to any heading. For example:

```
See @sec-text for additional context.
```

See Section 2.1 for additional context.

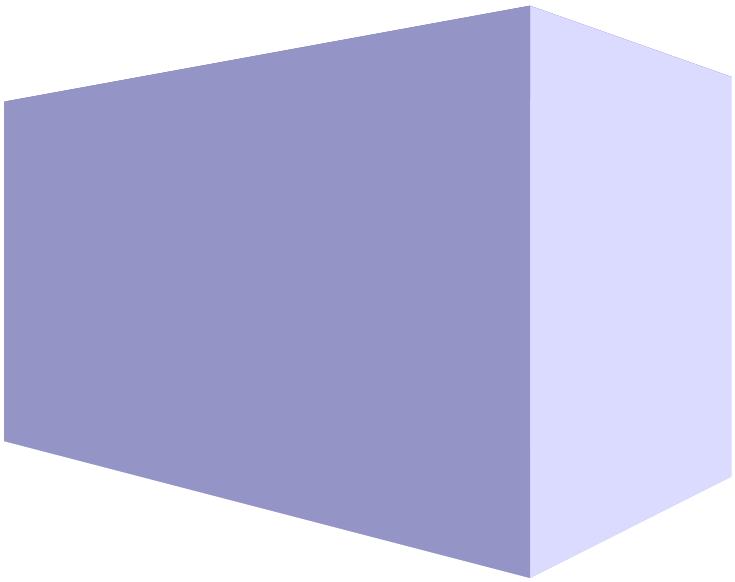


Figure 2.3: Referenced Abstract Shape

2.2.3 Equations

To reference an equation, add a `#eq-` identifier to any display equation. For example:

```
$$
...
$$ {#eq-einstein}
```

See Equation 2.1 for additional context.

2.2.4 Footnotes

Footnotes can be specified using the following syntax:

```
Here is a footnote reference,[^1] and another.[^longnote]

[^1]: Here is the footnote.

[^longnote]: Here's one with multiple blocks.
```

```
    Subsequent paragraphs are indented to show that they
belong to the previous footnote.

        { some.code }

    The whole paragraph can be indented, or just the first
    line.  In this way, multi-paragraph footnotes work like
    multi-paragraph list items.

This paragraph won't be part of the note, because it
isn't indented.
```

Here is a footnote reference,[1] and another.[2]

This paragraph won't be part of the note, because it isn't indented.

Footnotes can be a preferable way to specify links to webpages [3] to ensure that people notice and can follow the link, even if the output format is a printed PDF.

> **Warning**
>
> Note that the cross-referencing syntax is Quarto-specific and makes Jupyter notebooks display less cleanly in other environments. But the intrusions are not huge (imo).

2.3 Citations

Citing other scholars' work is a fundamental part of any scholarly written piece.

2.3.1 Bibliography files

Quarto supports bibliography files in a wide variety of formats. For example, add a bibliography file to your document in the YAML metadata as follows,

[1] Here is the footnote.

[2] Here's one with multiple blocks.

    Subsequent paragraphs are indented to show that they belong to the previous footnote.

```
{ some.code }
```

    The whole paragraph can be indented, or just the first line. In this way, multi-paragraph footnotes work like multi-paragraph list items.

[3] https://quarto.org/

```
---
bibliography: references.bib
---
```

The file `references.bib` must be in the same folder as your Jupyter notebook. In our case it contains the following,

```
!cat references.bib
```

```
@article{BarfussEtAl2020,
  title = {Caring for the Future Can Turn Tragedy into Comedy for Long-Term Collective Actio
  author = {Barfuss, Wolfram and Donges, Jonathan F. and Vasconcelos, V{\'i}tor V. and Kurths
  year = {2020},
  journal = {Proceedings of the National Academy of Sciences},
  volume = {117},
  number = {23},
  pages = {12915--12922},
  publisher = {Proceedings of the National Academy of Sciences},
  doi = {10.1073/pnas.1916545117},
  url = {https://www.pnas.org/doi/abs/10.1073/pnas.1916545117},
  urldate = {2022-03-10},
  copyright = {All rights reserved}
}

@article{Barfuss2022,
  title = {Dynamical Systems as a Level of Cognitive Analysis of Multi-Agent Learning},
  author = {Barfuss, Wolfram},
  year = {2022},
  journal = {Neural Computing and Applications},
  volume = {34},
  number = {3},
  pages = {1653--1671},
  issn = {1433-3058},
  doi = {10.1007/s00521-021-06117-0},
  url = {https://doi.org/10.1007/s00521-021-06117-0},
  urldate = {2023-03-02},
  copyright = {All rights reserved}
}
```

2.3.2 Citations syntax

Citations go inside square brackets and are separated by semicolons, e.g.,

```
very important finding [see @BarfussEtAl2020, pp. 2-3; also @Barfuss2022, Sec. 1]
```

very important finding (see Barfuss et al. 2020, 2–3; also Barfuss 2022, sec. 2)

2.4 Document metadata

Document details can be given via a so-called YAML frontmatter.

```
---
title: "Scholarly writing"
author:
    - name: "Wolfram Barfuss"
    - affiliation: "University of Bonn"
---
```

More information on the Quarto Documentation.

2.5 Generating output

To render this notebook as a standalone file we give it further metadata:

```
---
bibliography: references.bib
format:
    pdf:
        link-citations: true
---
```

Moreover, it must not be part of a project. So we copy it, render the copy, and the remove all unnecessary copies.

```
!cp 002ScholarlyWriting.ipynb 002ScholarlyWriting_.ipynb
!quarto render 002ScholarlyWriting_.ipynb --to pdf
!mv 002ScholarlyWriting_.pdf __output/002ScholarlyWriting.pdf
!rm -r 002ScholarlyWriting_*
```

2.6 References

3 Computational writing

In the following, we will enrich the scholarly writing experience with computations.

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

3.1 Basic example: predator-prey dynamics

Let's assume we want to study predator-prey dynamics in ecology. The predator-prey equations are a famous model showing that species population sizes do not have to be stable, even in equilibrium. Instead, they can continuously oscillate.

Mathematically, we express the model as follows:

Let $x_t \in \mathbb{R}$ be the number of prey and $y_t \in \mathbb{R}$ the number of predators in the population at time step t. They evolve according to

$$x_{t+1} - x_t =: \Delta x = \alpha x_t - \beta x_t y_t \tag{3.1}$$
$$y_{t+1} - y_t =: \Delta y = \gamma y_t x_t - \delta y_t \tag{3.2}$$

where the parameter α represents the preys' birth rate, β the prey's mortality rate, γ the predator efficiency, and δ the predators' death rate.

Right below the mathematical model, we define the computational model:

```
def predprey_model(prey_birth_rate,
                   prey_mortality,
                   predator_efficiency,
                   predator_death_rate,
                   initial_prey,
                   initial_predators,
                   time_length):
    """ Discrete-time predator-prey model. """
    x = -1 * np.ones(time_length)
```

```
    y = -1 * np.ones(time_length)
    x[0] = initial_prey
    y[0] = initial_predators
    for t in range(1, time_length):
        x[t] = x[t-1] + prey_birth_rate * x[t-1]\
            - prey_mortality * y[t-1]*x[t-1]
        y[t] = y[t-1] + predator_efficiency * y[t-1]*x[t-1]\
            - predator_death_rate * y[t-1]
    return x, y
```

We test the model,

```
preys, predators = predprey_model(0.1, 0.1, 0.1, 0.01, 1.0, 1.0, 1000)
```

and portray the model output, make the Figure referenceable in by using the Quarto comment commands

```
#| label: fig-modelrun
#| fig-cap: "An exemplary predator-prey model run"
```

```
plt.plot(preys, label="preys", color='blue')
plt.plot(predators, label="predators", color='orange')
plt.legend();
```

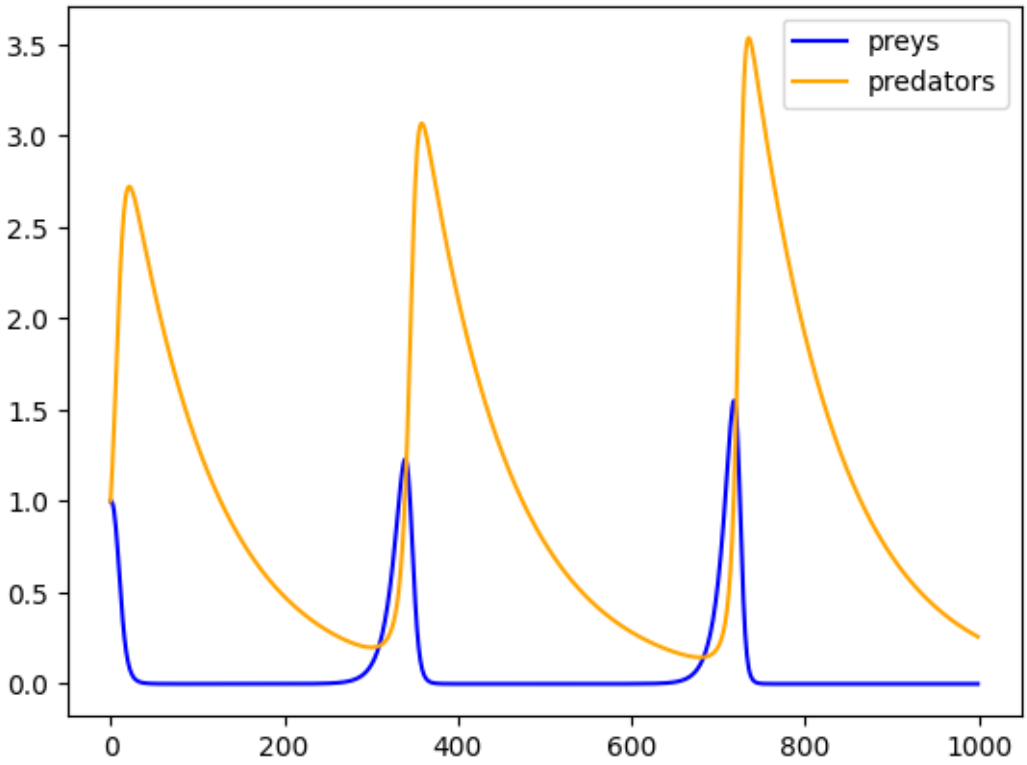


Figure 3.1: An exemplary predator-prey model run.

Figure 3.1 shows an exemplary predator-prey model run.

> **Note**
>
> This basic form of computational writing is the most transparency, collaboration, and reproducibility friednly. This approach allows us to cleanly display our Jupyter notebooks in other places, such as GitHub, NBViewer, or Google Colab.

However, often model code and analysis are too complex to be presented in a single notebook. There a two ways to deal with this problem: Quarto embeds and reusing code written in a notebook with nbdev.

3.2 Quarto embeds

Quarto lets you embed the output of another document with the embed shortcode. To do this, simply provide the document path and block or cell identifier, e.g,

```
{{< embed HeavyComputations.ipynb#fig-scatter-plot >}}
```

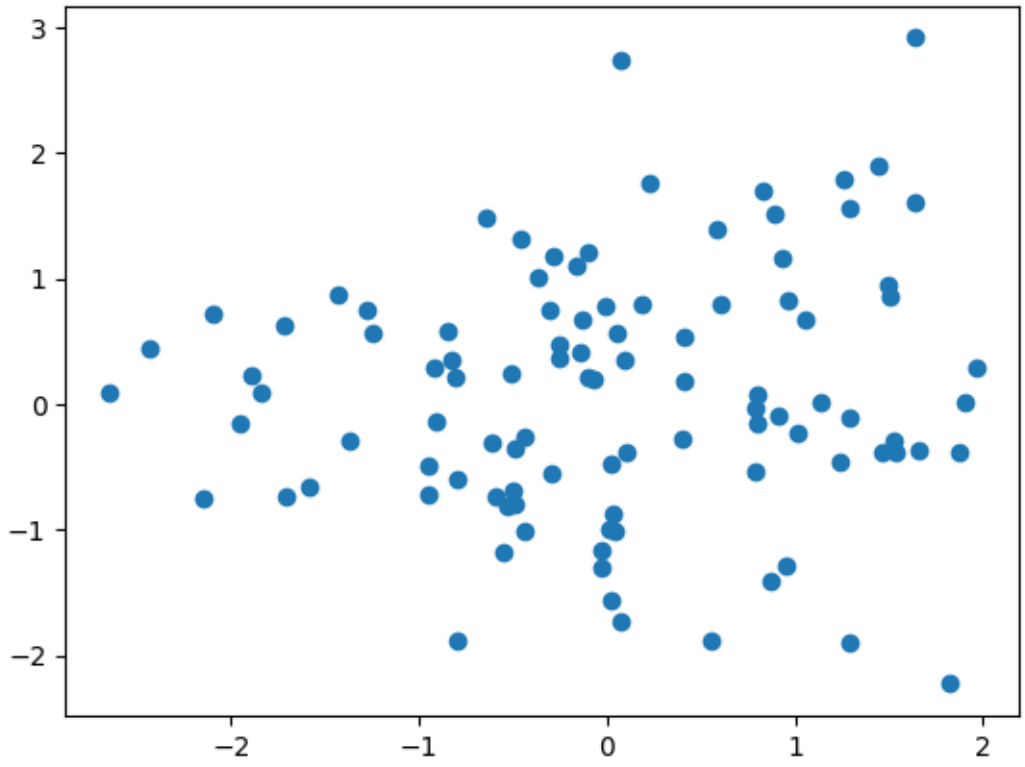


Figure 3.2

Figure 4.1 shows a heavy scatter plot.

> **Note**
>
> Quarto embeds can be a great and easy way to structure a project without using an additional tool. However, the Quarto embeds won't show in any other Jupyter environment.

Another strategy is to reuse code written in a notebook with nbdev.

3.3 Code reuse with nbdev

This part makes use of the excellent nbdev Python package [1]. nbdev provides a ready-to-use ecosystem for creating software packages with Jupyter Notebooks, i.e., writing, testing,

[1] https://nbdev.fast.ai/

documenting, and distributing. We use it, for example, here [2]. For our computational writing experience, however, we resort to individual components of nbdev.

See nbdev's getting-started guide [3] for how to install it.

One of the key features of nbdev is that it lets you export specific Jupyter Notebook cells to a plain Python file. These Python files can then be easily imported into other Jupyter Notebooks.

You only have to specify the name of Python module file to export to via

```
#|default_exp <name>
```

Then, you can a export a Juptyer Notebook cell by simply prepending

```
#|export
```

to it.

> **!** Important
>
> To avoid a small conflict between Quarto and nbdev, make sure to add a space, , before the nbdev directives `#|default_exp <name>` and `#|export`. See here for details.

… Shows an example

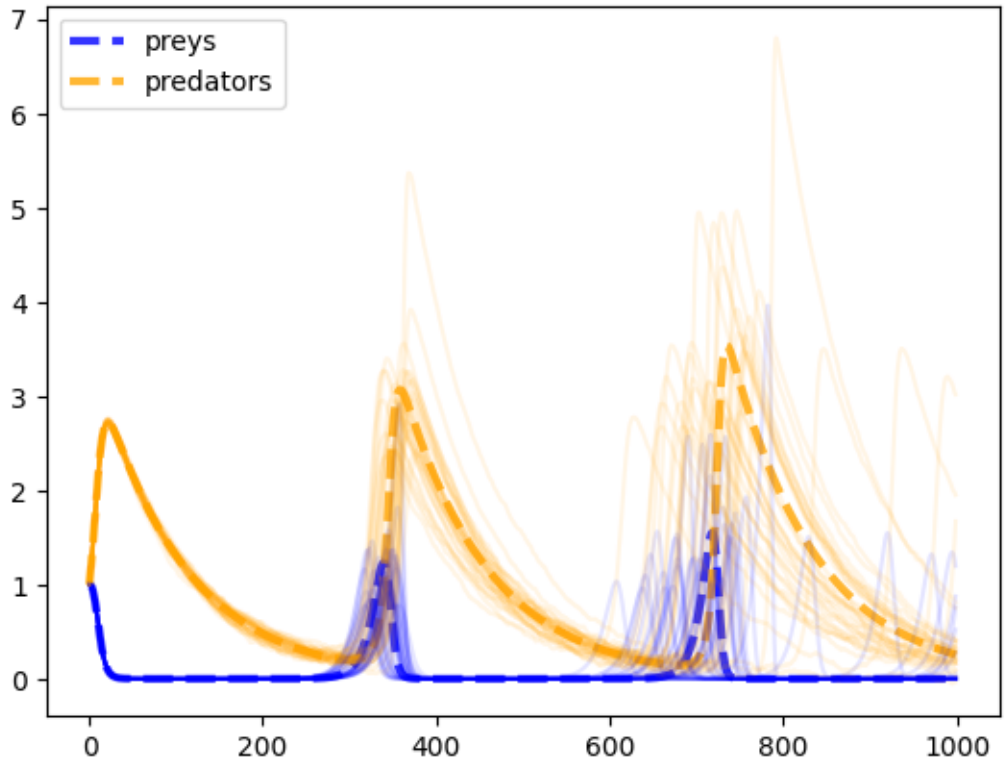
```
from _code.HeavyComputations import noisy_predprey_model
```

```
np.random.seed(42)

preys, predators = noisy_predprey_model(0.1, 0.1, 0.1, 0.01, 1.0, 1.0, 1000, 0.0)
plt.plot(preys, lw=3, ls='--', label="preys", color='blue', alpha=0.8)
plt.plot(predators, lw=3, ls='--', label="predators", color='orange', alpha=0.8)

for _ in range(35):
    preys, predators = noisy_predprey_model(0.1, 0.1, 0.1, 0.01, 1.0, 1.0, 1000, 0.025)
    plt.plot(preys, color='blue', alpha=0.1)
    plt.plot(predators, color='orange', alpha=0.1)
plt.legend();
```

[2] https://wbarfuss.github.io/pyCRLD/ and https://github.com/wbarfuss/pyCRLD
[3] https://nbdev.fast.ai/getting_started.html


preys
predators
0
1
2
3
4
5
6
7
0
200
400
600
800
1000

4 Annex: Heavy computations

```
# Imports for the nbdev development environment
import nbdev
```

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

```
data = np.random.randn(2, 100)
plt.scatter(*data);
```

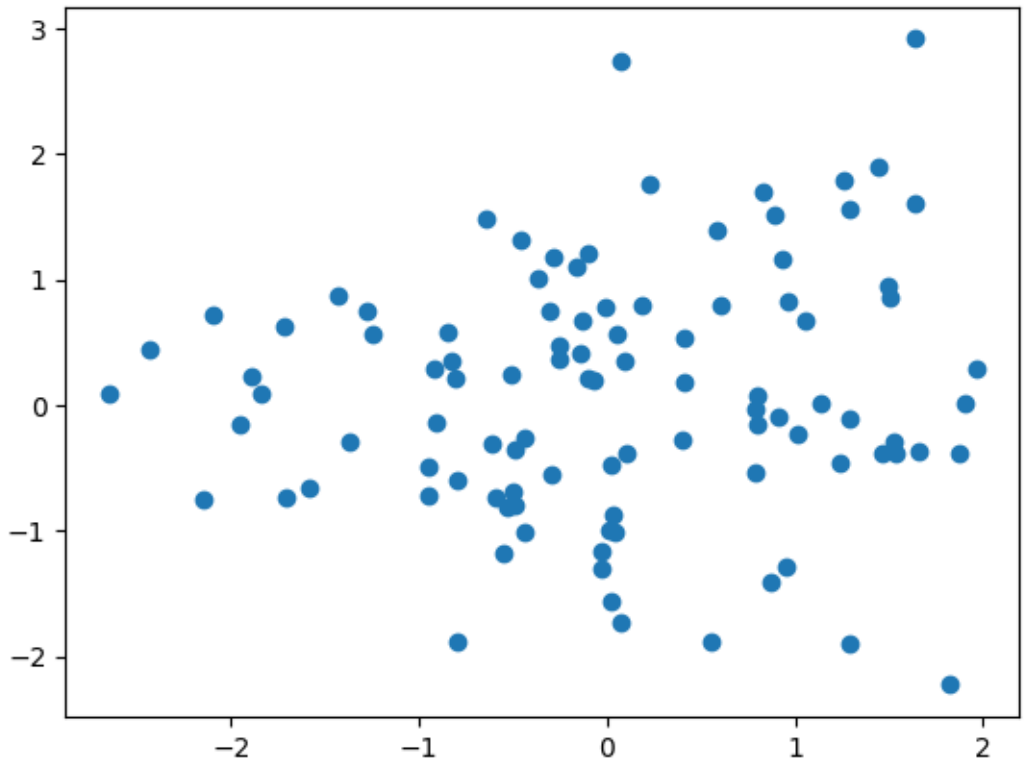


Figure 4.1

```
#| default_exp HeavyComputations
```

```
#| export
import numpy as np
```

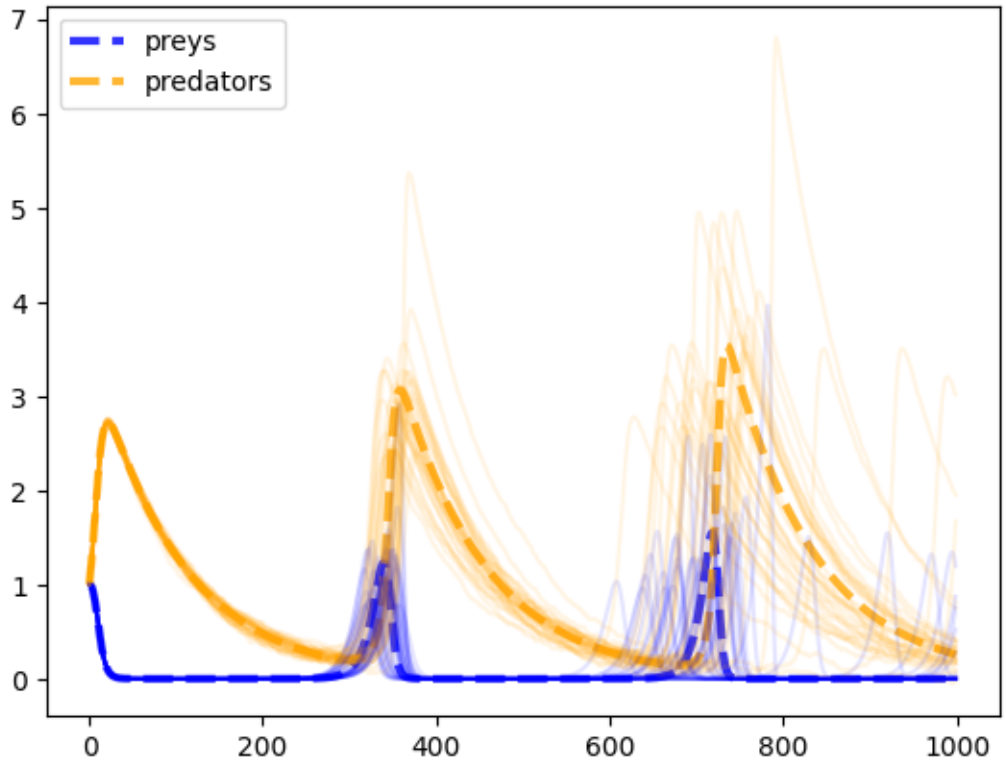
```
#| export
def noisy_predprey_model(prey_birth_rate,
                         prey_mortality,
                         predator_efficiency,
                         predator_death_rate,
                         initial_prey,
                         initial_predators,
                         time_length,
                         noiselevel):
    """ Discrete-time predator-prey model. """
    x = -1 * np.ones(time_length)
    y = -1 * np.ones(time_length)
    x[0] = initial_prey
    y[0] = initial_predators
    for t in range(1, time_length):
        x[t] = x[t-1] + prey_birth_rate * x[t-1]\
            - prey_mortality * y[t-1]*x[t-1]
        y[t] = y[t-1] + predator_efficiency * y[t-1]*x[t-1]\
            - predator_death_rate * y[t-1]\
            + noiselevel * (0.5 - np.random.rand())
    return x, y
```

```
np.random.seed(42)

preys, predators = noisy_predprey_model(0.1, 0.1, 0.1, 0.01, 1.0, 1.0, 1000, 0.0)
plt.plot(preys, lw=3, ls='--', label="preys", color='blue', alpha=0.8)
plt.plot(predators, lw=3, ls='--', label="predators", color='orange', alpha=0.8)

for _ in range(35):
    preys, predators = noisy_predprey_model(0.1, 0.1, 0.1, 0.01, 1.0, 1.0, 1000,
                                            0.025)

    plt.plot(preys, color='blue', alpha=0.1)
    plt.plot(predators, color='orange', alpha=0.1)
plt.legend();
```




```
nbdev.export.nb_export("HeavyComputations.ipynb", "_code")
```

Barfuss, Wolfram. 2022. "Dynamical Systems as a Level of Cognitive Analysis of Multi-Agent Learning." *Neural Computing and Applications* 34 (3): 1653–71. https://doi.org/10.1007/s00521-021-06117-0.

Barfuss, Wolfram, Jonathan F. Donges, Vítor V. Vasconcelos, Jürgen Kurths, and Simon A. Levin. 2020. "Caring for the Future Can Turn Tragedy into Comedy for Long-Term Collective Action Under Risk of Collapse." *Proceedings of the National Academy of Sciences* 117 (23): 12915–22. https://doi.org/10.1073/pnas.1916545117.