Lessons Learned Publishing the First Executable Paper in Heliophysics

Shawn Polson, Rebecca Ringuette,
Lutz Rastaetter, Eric Grimes, Jon Niehof,
Nick Murphy, Yihua Zheng

In This Talk

- ❑ Open Science and Reproducibility Problems
- ❑ Executable Papers Intro
- ❑ Lessons Learned

An Executable Paper

Table of contents

Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility

Shawn Polson[1], Rebecca Ringuette[2,3], Lutz Rastaetter[3], Eric Grimes[4], Jonathan Niehof[5], Nicholas A. Murphy[6], Yihua Zheng[3]

[1]Laboratory for Atmospheric and Space Physics (LASP), University of Colorado, Boulder, CO, United States.

[2]ADNET Systems Inc, Bethesda, MD, United States.

[3]Community Coordinated Modeling Center (CCMC), NASA Goddard Space Flight Center, Greenbelt, MD, United States.

[4]Institute of Geophysics and Planetary Physics, University of California, Los Angeles, CA, United States.

[5]Space Science Center, University of New Hampshire, Durham, NH, United States.

[6]Center for Astrophysics | Harvard & Smithsonian, Cambridge, MA, United States.

Abstract

We share the story of how we made this paper, the first executable paper in Heliophysics, through cross-disciplinary collaboration to highlight the benefits of our process. Executable papers are interactive documents that put a publication's text inline with the code used in the research in a containerized environment with the data and dependencies needed to run the code. This approach enables readers to reproduce every step taken to arrive at the publication's conclusions and to easily build upon and extend the work—all important components of open science. Open science is, broadly speaking, transparent and accessible knowledge that is shared and developed through collaborative networks. In this work, we present an adaptable Python workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. Most of the authors are members of the Python in Heliophysics Community (PyHC), an international, multi-organizational community that serves as a knowledge base for performing Heliophysics research in the Python programming language. PyHC promotes the executable paper format as a supplemental tool to improve the reproducibility of publications and support open science. A key takeaway is that our collaboration made such a complex task an easy feat in the end. Additionally, the executable version of our paper makes it trivial for others to reproduce our work, and it gives them a better launching point to extend it. These facts underscore the success of our approach. In highlighting this new open science approach, we hope to be an example to our field and encourage this way of doing science.

1 Introduction

We recount how we made this paper, the first executable paper in Heliophysics, to highlight the benefits of our process. Executable papers are a new kind of paper written in software that combines text, data, and code to enable readers to reproduce every step taken to arrive at the paper's conclusions (Lasser 2020). Our executable paper is centered around an adaptable Python workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. We detail how our process was facilitated by such a collaboration and guided by open science principles.

The following subsections provide the background to understand this work. We set the scene in section 1.1 by discussing open science and how it relates to issues with reproducibility. Then we describe the organization from which our team originates, the Python in Heliophysics Community (PyHC), and how our goals align with this work in section 1.2. Then we fully explain the concept of executable papers in section 1.3—what they are, why they benefit reproducibility, and how they compare to traditional papers. Next, we discuss the cross-disciplinary collaboration underpinning our work and why it was crucial to our success in section 1.4. Then we give the science background to follow the workflow presented in our executable paper before actually presenting it in section 1.5. The following "Method" section contains the workflow (section 2). We will showcase it fully, from the underlying concepts to the implementation using our PyHC packages. Section 3 covers the results and outcomes from our work. Finally, we end the paper with our closing remarks in section 4.

1.1 Open Science and Reproducibility

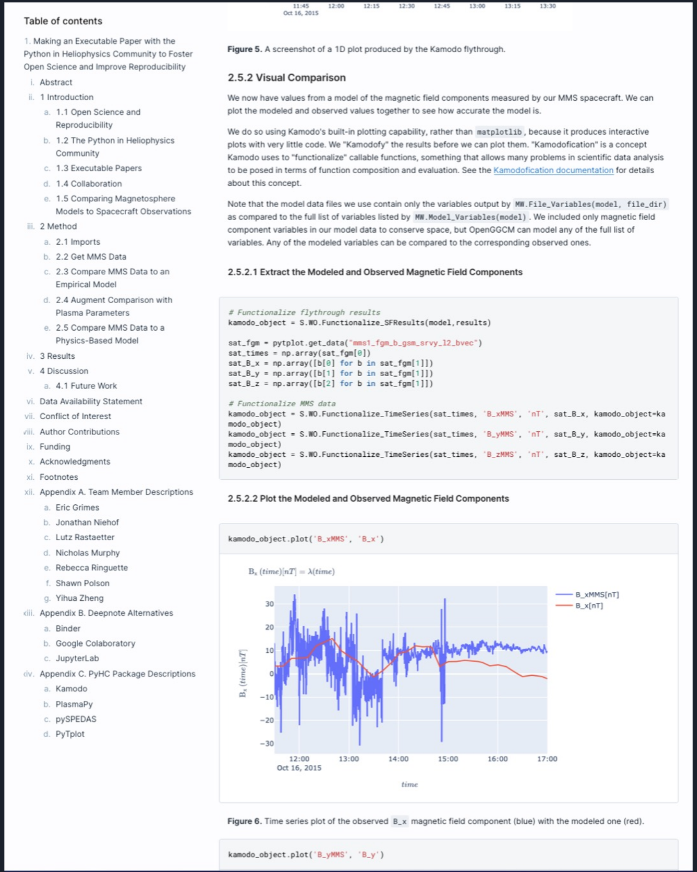


Table of contents

1. Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility
   i. Abstract
   ii. 1 Introduction
      a. 1.1 Open Science and Reproducibility
      b. 1.2 The Python in Heliophysics Community
      c. 1.3 Executable Papers
      d. 1.4 Collaboration
      e. 1.5 Comparing Magnetosphere Models to Spacecraft Observations
   iii. 2 Method
      a. 2.1 Imports
      b. 2.2 Get MMS Data
      c. 2.3 Compare MMS Data to an Empirical Model
      d. 2.4 Augment Comparison with Plasma Parameters
      e. 2.5 Compare MMS Data to a Physics-Based Model
   iv. 3 Results
   v. 4 Discussion
      a. 4.1 Future Work
   vi. Data Availability Statement
   vii. Conflict of Interest
   viii. Author Contributions
   ix. Funding
   x. Acknowledgements
   xi. Footnotes
   xii. Appendix A. Team Member Descriptions
      a. Eric Grimes
      b. Jonathan Niehof
      c. Lutz Rastaetter
      d. Nicholas Murphy
      e. Rebecca Ringuette
      f. Shawn Polson
      g. Yihua Zheng
   xiii. Appendix B. Deepnote Alternatives
      a. Binder
      b. Google Colaboratory
      c. JupyterLab
   xiv. Appendix C. PyHC Package Descriptions
      a. Kamodo
      b. PlasmaPy
      c. pySPEDAS
      d. PyTplot

Figure 5. A screenshot of a 1D plot produced by the Kamodo flythrough.

2.5.2 Visual Comparison

We now have values from a model of the magnetic field components measured by our MMS spacecraft. We can plot the modeled and observed values together to see how accurate the model is.

We do so using Kamodo's built-in plotting capability, rather than `matplotlib`, because it produces interactive plots with very little code. We "Kamodofy" the results before we can plot them. "Kamodofication" is a concept Kamodo uses to "functionalize" callable functions, something that allows many problems in scientific data analysis to be posed in terms of function composition and evaluation. See the Kamodofication documentation for details about this concept.

Note that the model data files we use contain only the variables output by `MW.File_Variables(model, file_dir)` as compared to the full list of variables listed by `MW.Model_Variables(model)`. We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the Modeled and Observed Magnetic Field Components

```
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data("mms1_fgm_b_gsm_srvy_l2_bvec")
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=ka
modo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=ka
modo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=ka
modo_object)
```

2.5.2.2 Plot the Modeled and Observed Magnetic Field Components

```
kamodo_object.plot('B_xMMS', 'B_x')
```

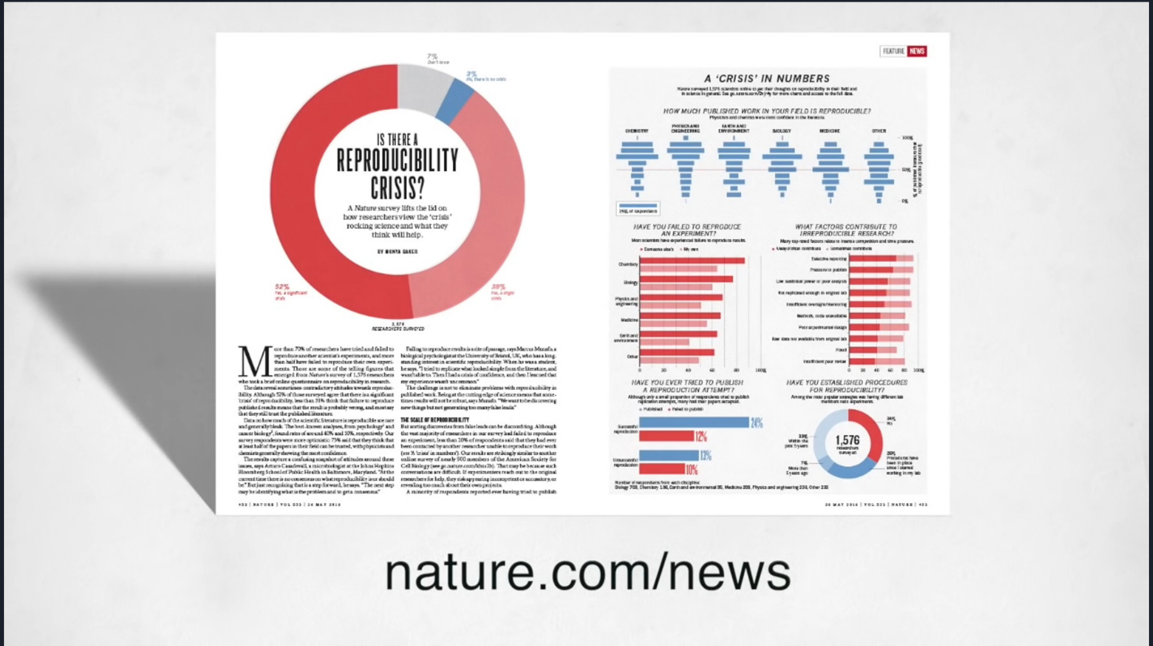

Figure 6. Time series plot of the observed `B_x` magnetic field component (blue) with the modeled one (red).

```
kamodo_object.plot('B_yMMS', 'B_y')
```

Open Science and Reproducibility Problems

Open Science and Reproducibility Problems

Six years since Nature “lifted the lid on the reproducibility crisis”



Open Science and Reproducibility Problems

Traditional publications face reproducibility problems

- ❑ Missing raw or original data
- ❑ Lack of a tidied-up version of the data
- ❑ No source code available
- ❑ Lacking software to run the experiment

Open Science and Reproducibility Problems

Traditional publications face reproducibility problems

- ❑ Missing raw or original data
- ❑ Lack of a tidied-up version of the data
- ❑ No source code available
- ❑ Lacking software to run the experiment

- ❑ Even when available, replication can be non-trivial

Executable Papers

Executable Papers

“Interactive pieces of software that combine text, data, and code to enable readers to reproduce every step taken to arrive at a publication’s conclusions”

Executable Papers

Table of contents

Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility

Shawn Polson[1], Rebecca Ringuette[2,3], Lutz Rastaetter[3], Eric Grimes[4], Jonathan Niehof[5], Nicholas A. Murphy[6], Yihua Zheng[3]

[1]Laboratory for Atmospheric and Space Physics (LASP), University of Colorado, Boulder, CO, United States.

[2]ADNET Systems Inc, Bethesda, MD, United States.

[3]Community Coordinated Modeling Center (CCMC), NASA Goddard Space Flight Center, Greenbelt, MD, United States.

[4]Institute of Geophysics and Planetary Physics, University of California, Los Angeles, CA, United States.

[5]Space Science Center, University of New Hampshire, Durham, NH, United States.

[6]Center for Astrophysics | Harvard & Smithsonian, Cambridge, MA, United States.

Abstract

We share the story of how we made this paper, the first executable paper in Heliophysics, through cross-disciplinary collaboration to highlight the benefits of our process. Executable papers are interactive documents that put a publication's text inline with the code used in the research in a containerized environment with the data and dependencies needed to run the code. This approach enables readers to reproduce every step taken to arrive at the publication's conclusions and to easily build upon and extend the work—all important components of open science. Open science is, broadly speaking, transparent and accessible knowledge that is shared and developed through collaborative networks. In this work, we present an adaptable workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. Most of the authors are members of the Python in Heliophysics Community (PyHC), an international, multi-organizational community that serves as a knowledge base for performing Heliophysics research in the Python programming language. PyHC promotes the executable paper format as a supplemental tool to improve the reproducibility of publications and support open science. A key takeaway is that our collaboration made such a complex task an easy feat in the end. Additionally, the executable version of our paper makes it trivial for others to reproduce our work, and it gives them a better launching point to extend it. These facts underscore the success of our approach. In highlighting this new open science approach, we hope to be an example to our field and encourage this way of doing science.

1 Introduction

We recount how we made this paper, the first executable paper in Heliophysics, to highlight the benefits of our process. Executable papers are a new kind of paper written in software that combines text, data, and code to enable readers to reproduce every step taken to arrive at the paper's conclusions (Lasser 2020). Our executable paper is centered around an adaptable Python workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. We detail how our process was facilitated by such a collaboration and guided by open science principles.

The following subsections provide the background to understand this work. We set the scene in section 1.1 by discussing open science and how it relates to issues with reproducibility. Then we describe the organization from which our team originates, the Python in Heliophysics Community (PyHC), and how our goals align with this work in section 1.2. Then we fully explain the concept of executable papers in section 1.3—what they are, why they benefit reproducibility, and how they compare to traditional papers. Next, we discuss the cross-disciplinary collaboration underpinning our work and why it was crucial to our success in section 1.4. Then we give the science background to follow the workflow presented in our executable paper before actually presenting it in section 1.5. The following "Method" section contains the workflow (section 2). We will showcase it fully, from the underlying concepts to the implementation using our PyHC packages. Section 3 covers the results and outcomes from our work. Finally, we end the paper with our closing remarks in section 4.

1.1 Open Science and Reproducibility

Table of contents

1. Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility
   - i. Abstract
   - ii. 1 Introduction
     - a. 1.1 Open Science and Reproducibility
     - b. 1.2 The Python in Heliophysics Community
     - c. 1.3 Executable Papers
     - d. 1.4 Collaboration
     - e. 1.5 Comparing Magnetosphere Models to Spacecraft Observations
   - iii. 2 Method
     - a. 2.1 Imports
     - b. 2.2 Get MMS Data
     - c. 2.3 Compare MMS Data to an Empirical Model
     - d. 2.4 Augment Comparison with Plasma Parameters
     - e. 2.5 Compare MMS Data to a Physics-Based Model
   - iv. 3 Results
   - v. 4 Discussion
     - a. 4.1 Future Work
   - vi. Data Availability Statement
   - vii. Conflict of Interest
   - viii. Author Contributions
   - ix. Funding
   - x. Acknowledgements
   - xi. Footnotes
   - xii. Appendix A. Team Member Descriptions
     - a. Eric Grimes
     - b. Jonathan Niehof
     - c. Lutz Rastaetter
     - d. Nicholas Murphy
     - e. Rebecca Ringuette
     - f. Shawn Polson
     - g. Yihua Zheng
   - xiii. Appendix B. Deepnote Alternatives
     - a. Binder
     - b. Google Colaboratory
     - c. JupyterLab
   - xiv. Appendix C. PyHC Package Descriptions
     - a. Kamodo
     - b. PlasmaPy
     - c. pySPEDAS
     - d. PyTplot

11:45 12:00 12:15 12:30 12:45 13:00 13:15 13:30
Oct 16, 2015

Figure 5. A screenshot of a 1D plot produced by the Kamodo flythrough.

2.5.2 Visual Comparison

We now have values from a model of the magnetic field components measured by our MMS spacecraft. We can plot the modeled and observed values together to see how accurate the model is.

We do so using Kamodo's built-in plotting capability, rather than `matplotlib`, because it produces interactive plots with very little code. We "Kamodofy" the results before we can plot them. "Kamodofication" is a concept Kamodo uses to "functionalize" callable functions, something that allows many problems in scientific data analysis to be posed in terms of function composition and evaluation. See the Kamodofication documentation for details about this concept.

Note that the model data files we use contain only the variables output by `MW.File_Variables(model, file_dir)` as compared to the full list of variables listed by `MW.Model_Variables(model)`. We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the Modeled and Observed Magnetic Field Components

```python
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data('mms1_fgm_b_gsm_srvy_l2_bvec')
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the Modeled and Observed Magnetic Field Components

```python
kamodo_object.plot('B_xMMS', 'B_x')
```

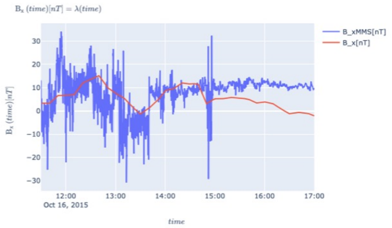


Figure 6. Time series plot of the observed `B_x` magnetic field component (blue) with the modeled one (red).

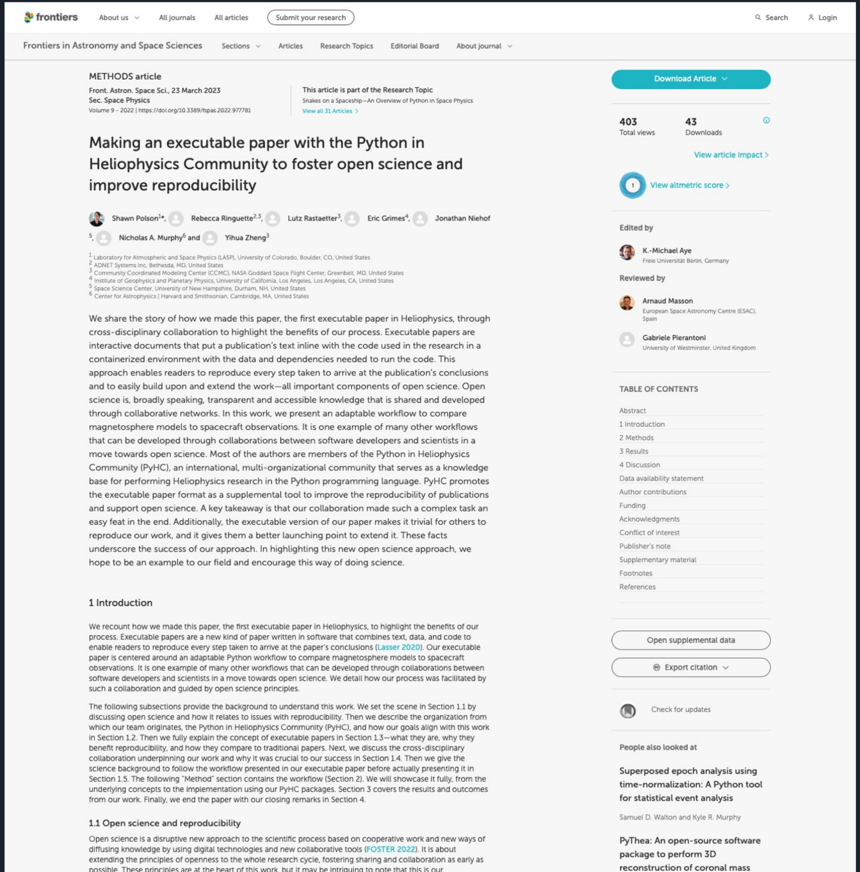
```python
kamodo_object.plot('B_yMMS', 'B_y')
```

Publishing Executable Papers

- ❑ Journals don't yet accept executable papers
  - ■ Not unheard of to reference 80–100 year-old papers
  - ■ Hard to fathom that level of support for any software
  - ■ Who hosts them?

- ❑ Include them with traditional paper submissions

“Include them with traditional paper submissions”



METHODS article
Front. Astron. Space Sci., 23 March 2023
Sec. Space Physics
Volume 9 - 2022 | https://doi.org/10.3389/fspas.2022.977781

This article is part of the Research Topic
Snakes on a Spaceship—An Overview of Python in Space Physics
View all 31 Articles

Making an executable paper with the Python in Heliophysics Community to foster open science and improve reproducibility

Shawn Polson[1]*, Rebecca Ringuette[2,3], Lutz Rastaetter[3], Eric Grimes[4], Jonathan Niehof[5], Nicholas A. Murphy[6] and Yihua Zheng[3]

[1] Laboratory for Atmospheric and Space Physics (LASP), University of Colorado, Boulder, CO, United States
[2] ADNET Systems Inc, Bethesda, MD, United States
[3] Community Coordinated Modeling Center (CCMC), NASA Goddard Space Flight Center, Greenbelt, MD, United States
[4] Institute of Geophysics and Planetary Physics, University of California, Los Angeles, Los Angeles, CA, United States
[5] Space Science Center, University of New Hampshire, Durham, NH, United States
[6] Center for Astrophysics | Harvard and Smithsonian, Cambridge, MA, United States

We share the story of how we made this paper, the first executable paper in Heliophysics, through cross-disciplinary collaboration to highlight the benefits of our process. Executable papers are interactive documents that put a publication’s text inline with the code used in the research in a containerized environment with the data and dependencies needed to run the code. This approach enables readers to reproduce every step taken to arrive at the publication’s conclusions and to easily build upon and extend the work—all important components of open science. Open science is, broadly speaking, transparent and accessible knowledge that is shared and developed through collaborative networks. In this work, we present an adaptable workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. Most of the authors are members of the Python in Heliophysics Community (PyHC), an international, multi-organizational community that serves as a knowledge base for performing Heliophysics research in the Python programming language. PyHC promotes the executable paper format as a supplemental tool to improve the reproducibility of publications and support open science. A key takeaway is that our collaboration made such a complex task an easy feat in the end. Additionally, the executable version of our paper makes it trivial for others to reproduce our work, and it gives them a better launching point to extend it. These facts underscore the success of our approach. In highlighting this new open science approach, we hope to be an example to our field and encourage this way of doing science.

Edited by
K.-Michael Aye
Freie Universität Berlin, Germany

Reviewed by
Arnaud Masson
European Space Astronomy Centre (ESAC), Spain
Gabriele Pierantoni
University of Westminster, United Kingdom

TABLE OF CONTENTS

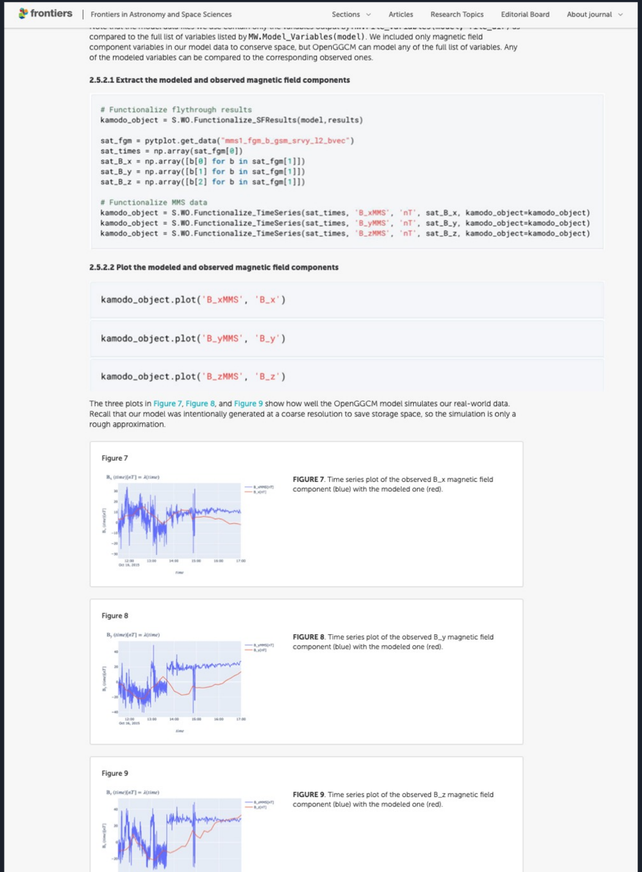
1 Introduction

We recount how we made this paper, the first executable paper in Heliophysics, to highlight the benefits of our process. Executable papers are a new kind of paper written in software that combines text, data, and code to enable readers to reproduce every step taken to arrive at the paper’s conclusions (Lasser 2020). Our executable paper is centered around an adaptable Python workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. We detail how our process was facilitated by such a collaboration and guided by open science principles.

The following subsections provide the background to understand this work. We set the scene in Section 1.1 by discussing open science and how it relates to issues with reproducibility. Then we describe the organization from which our team originates, the Python in Heliophysics Community (PyHC), and how our goals align with this work in Section 1.2. Then we fully explain the concept of executable papers in Section 1.3—what they are, why they benefit reproducibility, and how they compare to traditional papers. Next, we discuss the cross-disciplinary collaboration underpinning our work and why it was crucial to our success in Section 1.4. Then we give the science background to follow the workflow presented in our executable paper before actually presenting it in Section 1.5. The following “Method” section contains the workflow (Section 2). We will showcase it fully, from the underlying concepts to the implementation using our PyHC packages. Section 3 covers the results and outcomes from our work. Finally, we end the paper with our closing remarks in Section 4.

1.1 Open science and reproducibility

Open science is a disruptive new approach to the scientific process based on cooperative work and new ways of diffusing knowledge by using digital technologies and new collaborative tools (FOSTER 2022). It is about extending the principles of openness to the whole research cycle, fostering sharing and collaboration as early as possible. These principles are at the heart of this work, but it may be intriguing to note that this is our



compared to the full list of variables listed by `MW.Model_Variables(model)`. We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the modeled and observed magnetic field components

```
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data("mms1_fgm_b_gsm_srvy_l2_bvec")
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the modeled and observed magnetic field components

```
kamodo_object.plot('B_xMMS', 'B_x')
```

```
kamodo_object.plot('B_yMMS', 'B_y')
```

```
kamodo_object.plot('B_zMMS', 'B_z')
```

The three plots in Figure 7, Figure 8, and Figure 9 show how well the OpenGGCM model simulates our real-world data. Recall that our model was intentionally generated at a coarse resolution to save storage space, so the simulation is only a rough approximation.

Figure 7

FIGURE 7. Time series plot of the observed B_x magnetic field component (blue) with the modeled one (red).

Figure 8

FIGURE 8. Time series plot of the observed B_y magnetic field component (blue) with the modeled one (red).

Figure 9

FIGURE 9. Time series plot of the observed B_z magnetic field component (blue) with the modeled one (red).

“Include them with traditional paper submissions”

Footnotes

[1]Link to the executable version of this paper on Deepnote: https://deepnote.com/@shawn-polson/PyHC-Paper-101b9646-3fd0-4978-a48e-a4f3e708a0ac.

[2]DOI to the files of the executable version of this paper on Zenodo: https://doi.org/10.5281/zenodo.7412347.

[3]OpenGGCM input parameters:• run ID: Yihua_Zheng_031022_1• model: OpenGGCM• version: 5.0• IM_model: RCM• IM_version: 1.0• cs_input: GSM• cs_output: GSE• event_date: 16 October 2015• start_time: 2015/10/16 11:30• end_time: 2015/10/16 17:00• run_type: event• solarwind: var• sw_source: OMNI• despike solar wind data: 0• despike: threshold (sigmas): 3• despike: number of samples: 5• setting option for Bx: user• constant-Bx: 0• By-coefficient: 0• Bz-coefficient: 0• b_abs: 2.19• b_angle: 123.13• iono_conductance: auroral• Pedersen Conductance: default• Hall Conductance: default• f10.7: 108.4.

[4]The OpenGGCM model was run with a low-resolution grid (“dayside emphasis grid with 3,500,000 cells” with 355 × 100 × 100 cells) in the simulation domain extending from -350 to 33 R_E in X_{GSM}, and up to 49 R_E in $|Y_{GSE}|$ and $|Z_{GSE}|$ with a 0.3 R_E resolution around the Earth. During the packaging of magnetosphere model outputs into Kamodo, NetCDF files were reduced to only include magnetic field components (B_x, B_y, B_z) at a 600-s time cadence to limit model outputs to 1.6 GB. The near-Earth boundary conditions at 2.5 R_E distance includes 100 particles/cm^3, a temperature of 100 eV, and a shielding latitude of 45° in the ionosphere electric field potential solver.

zenodo Search Upload Communities Log in Sign up

December 7, 2022 Software Open Access

Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility

Polson, Shawn; Ringuette, Rebecca; Rastaetter, Lutz; Grimes, Eric; Niehof, Jonathan; Murphy, Nick; Zheng, Yihua

Stored here are the files to run the executable version of "Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility" by Shawn Polson et al., recently published in Frontiers in Astronomy and Space Sciences.

The paper is containerized with Docker, and we offer three different ways to spin up the container:

1. Load the Docker image from our TAR file:

With Docker installed and running on your machine, first download the file `pyhc-executable-paper.tar` which contains the Docker image `spolson/pyhc-executable-paper:v2`. Then load the image with:

```
docker load < pyhc-executable-paper.tar
```

Finally, run the image and connect to the JupyterLab server that starts in the container. Unless you are familiar enough with Docker to want to do it differently, we recommend this command:

```
docker run --rm -p 8888:8888 --user root -e GRANT_SUDO=yes spolson/pyhc-executable-paper:v2
```

That will start the container with the permissions necessary to run the notebook that is the executable paper from start to finish with "Run All Cells" in the Jupyter interface (including the initialization script that installs both Python and system packages). It also maps the container's port 8888 (where JupyterLab will start) to your machine's port 8888, such that you can launch JupyterLab by simply following the URL that is generated in your terminal output. Something like: `http://127.0.0.1:8888/lab?token=fd1dc3221fd13a5923bbc494b8553fe03b1183411687899a`.

2. Load the Docker image from DockerHub:

Our Docker image is available on DockerHub. Simply pull the image with:

```
docker pull spolson/pyhc-executable-paper:v2
```

then run the image:

```
docker run --rm -p 8888:8888 --user root -e GRANT_SUDO=yes spolson/pyhc-executable-paper:v2
```

(Note that once the image has been pulled, the remaining steps to run the image are identical to those in #1 above.)

3. Build the Docker image from source:

We also store the source files here so that you can build the Docker image from scratch if you want. First, download both the file `Dockerfile` and the compressed directory `src.zip`. Navigate to the directory where you downloaded them, unzip `src`, then build the Docker image with something like:

```
docker build -t pyhc-executable-paper:v2 .
```

(Refer to Docker's documentation for help using the `docker build` command.)

The remaining steps to run the image are identical to those in #1 and #2 above.

Preview

Files (5.4 GB)

Name	Size	
Dockerfile md5:043d2e1604747836d3be9e470eea3fbb	262 Bytes	Download

Indexed in OpenAIRE

Publication date: December 7, 2022

DOI: 10.5281/zenodo.7412347

Keyword(s): Python in Heliophysics Community; PyHC, Executable paper; Open science; Improving reproducibility; Magnetosphere models; Cross-disciplinary collaboration; Deepnote

Published in: Frontiers in Astronomy and Space Sciences:.

Communities: Python in Heliophysics Community

Versions

Version v2 — Dec 7, 2022 — 10.5281/zenodo.7412347

Version v1 — Jun 23, 2022 — 10.5281/zenodo.6713303

Cite all versions? You can cite all versions by using the DOI 10.5281/zenodo.6713302. This DOI represents all versions, and will always resolve to the latest one. Read more.

Share

Cite as

Polson, Shawn, Ringuette, Rebecca, Rastaetter, Lutz, Grimes, Eric, Niehof, Jonathan, Murphy, Nick, & Zheng, Yihua. (2022). Making an Executable Paper with the Python in Heliophysics Community to Foster Open Science and Improve Reproducibility. In Frontiers in Astronomy and Space Sciences (Version v2). Zenodo.

Lessons Learned

Lessons Learned

- ❑ Executable papers solve problems
- ❑ Big data is a challenge
- ❑ Journals decide how you represent code
- ❑ Working with typesetters was tricky

Executable papers solve problems

Traditional publications face reproducibility problems

- ~~Missing raw or original data~~ Have all data
- ~~Lack of a tidied-up version of the data~~ Have all data
- ~~No source code available~~ Have all source code
- ~~Lacking software to run the experiment~~ Have the software

- ~~Even when available, replication can be non-trivial~~ Trivial replication

Big data is a challenge

Storage Limits

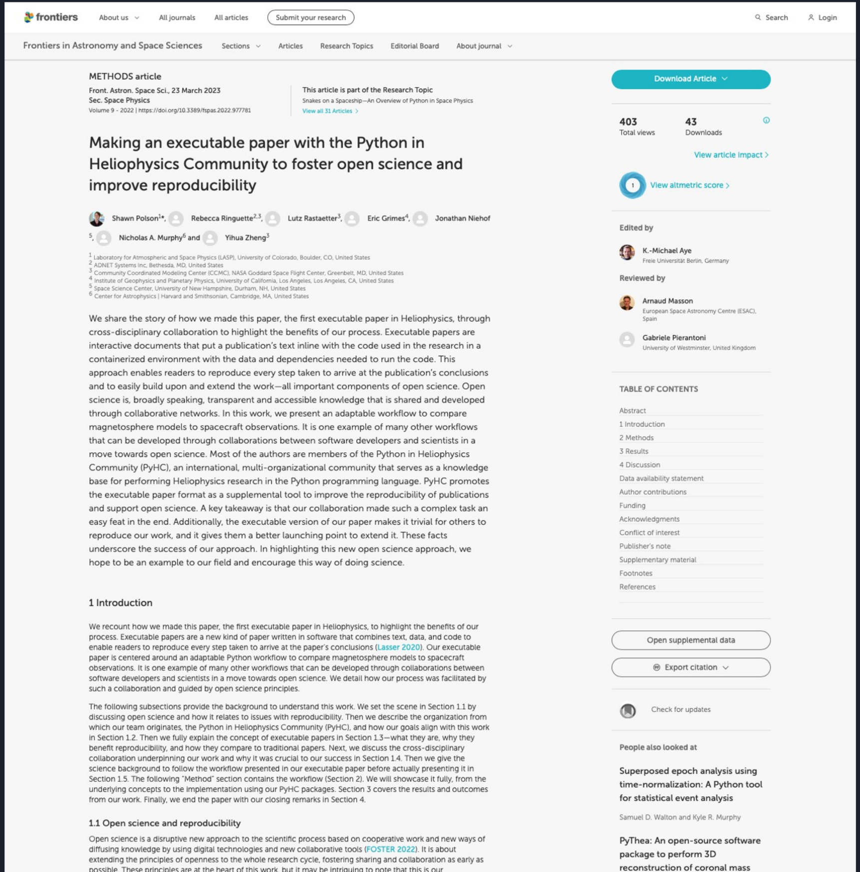
- ❑ Deepnote: 5GB free
- ❑ GitHub: 100MB files / 100GB repos (under 5GB recommended)
- ❑ Google Colab: 2GB files / ~77GB proj free (Pro offers up to 700GB)
- ❑ CoCalc: 3GB free (64GB–64TB offered for astronomical prices)
- ❑ Self-hosted container: whatever you can afford *indefinitely*

Can’t link to cloud storage in your paper

Journals decide how you represent code

- ❑ Text vs images
- ❑ Special fonts?
- ❑ Which images count as figures?
- ❑ Figure limits
- ❑ Typesetting figures

Working with typesetters was tricky



METHODS article
Front. Astron. Space Sci., 23 March 2023
Sec. Space Physics
Volume 9 - 2022 | https://doi.org/10.3389/fspas.2022.977781

This article is part of the Research Topic
Snakes on a Spaceship—An Overview of Python in Space Physics
View all 31 Articles

Making an executable paper with the Python in Heliophysics Community to foster open science and improve reproducibility

Shawn Polson[1]*, Rebecca Ringuette[2,3], Lutz Rastaetter[3], Eric Grimes[4], Jonathan Niehof[5], Nicholas A. Murphy[6] and Yihua Zheng[3]

1 Laboratory for Atmospheric and Space Physics (LASP), University of Colorado, Boulder, CO, United States
2 ADNET Systems Inc, Bethesda, MD, United States
3 Community Coordinated Modeling Center (CCMC), NASA Goddard Space Flight Center, Greenbelt, MD, United States
4 Institute of Geophysics and Planetary Physics, University of California, Los Angeles, Los Angeles, CA, United States
5 Space Science Center, University of New Hampshire, Durham, NH, United States
6 Center for Astrophysics | Harvard and Smithsonian, Cambridge, MA, United States

We share the story of how we made this paper, the first executable paper in Heliophysics, through cross-disciplinary collaboration to highlight the benefits of our process. Executable papers are interactive documents that put a publication's text inline with the code used in the research in a containerized environment with the data and dependencies needed to run the code. This approach enables readers to reproduce every step taken to arrive at the publication's conclusions and to easily build upon and extend the work—all important components of open science. Open science is, broadly speaking, transparent and accessible knowledge that is shared and developed through collaborative networks. In this work, we present an adaptable workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. Most of the authors are members of the Python in Heliophysics Community (PyHC), an international, multi-organizational community that serves as a knowledge base for performing Heliophysics research in the Python programming language. PyHC promotes the executable paper format as a supplemental tool to improve the reproducibility of publications and support open science. A key takeaway is that our collaboration made such a complex task an easy feat in the end. Additionally, the executable version of our paper makes it trivial for others to reproduce our work, and it gives them a better launching point to extend it. These facts underscore the success of our approach. In highlighting this new open science approach, we hope to be an example to our field and encourage this way of doing science.

Edited by
K.-Michael Aye
Freie Universität Berlin, Germany

Reviewed by
Arnaud Masson
European Space Astronomy Centre (ESAC), Spain
Gabriele Pierantoni
University of Westminster, United Kingdom

TABLE OF CONTENTS

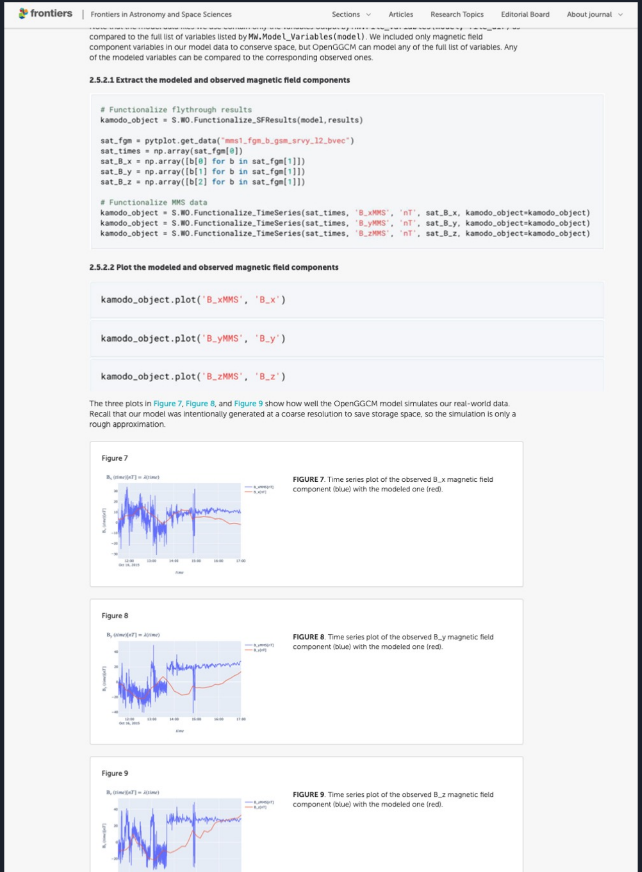
1 Introduction

We recount how we made this paper, the first executable paper in Heliophysics, to highlight the benefits of our process. Executable papers are a new kind of paper written in software that combines text, data, and code to enable readers to reproduce every step taken to arrive at the paper's conclusions (Lasser 2020). Our executable paper is centered around an adaptable Python workflow to compare magnetosphere models to spacecraft observations. It is one example of many other workflows that can be developed through collaborations between software developers and scientists in a move towards open science. We detail how our process was facilitated by such a collaboration and guided by open science principles.

The following subsections provide the background to understand this work. We set the scene in Section 1.1 by discussing open science and how it relates to issues with reproducibility. Then we describe the organization from which our team originates, the Python in Heliophysics Community (PyHC), and how our goals align with this work in Section 1.2. Then we fully explain the concept of executable papers in Section 1.3—what they are, why they benefit reproducibility, and how they compare to traditional papers. Next, we discuss the cross-disciplinary collaboration underpinning our work and why it was crucial to our success in Section 1.4. Then we give the science background to follow the workflow presented in our executable paper before actually presenting it in Section 1.5. The following "Method" section contains the workflow (Section 2). We will showcase it fully, from the underlying concepts to the implementation using our PyHC packages. Section 3 covers the results and outcomes from our work. Finally, we end the paper with our closing remarks in Section 4.

1.1 Open science and reproducibility

Open science is a disruptive new approach to the scientific process based on cooperative work and new ways of diffusing knowledge by using digital technologies and new collaborative tools (FOSTER 2022). It is about extending the principles of openness to the whole research cycle, fostering sharing and collaboration as early as possible. These principles are at the heart of this work, but it may be intriguing to note that this is our



compared to the full list of variables listed by MW.Model_Variables(model). We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the modeled and observed magnetic field components

```
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data("mms1_fgm_b_gsm_srvy_l2_bvec")
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the modeled and observed magnetic field components

```
kamodo_object.plot('B_xMMS', 'B_x')
```

```
kamodo_object.plot('B_yMMS', 'B_y')
```

```
kamodo_object.plot('B_zMMS', 'B_z')
```

The three plots in Figure 7, Figure 8, and Figure 9 show how well the OpenGGCM model simulates our real-world data. Recall that our model data was intentionally generated at a coarse resolution to save storage space, so the simulation is only a rough approximation.

Figure 7

FIGURE 7. Time series plot of the observed B_x magnetic field component (blue) with the modeled one (red).

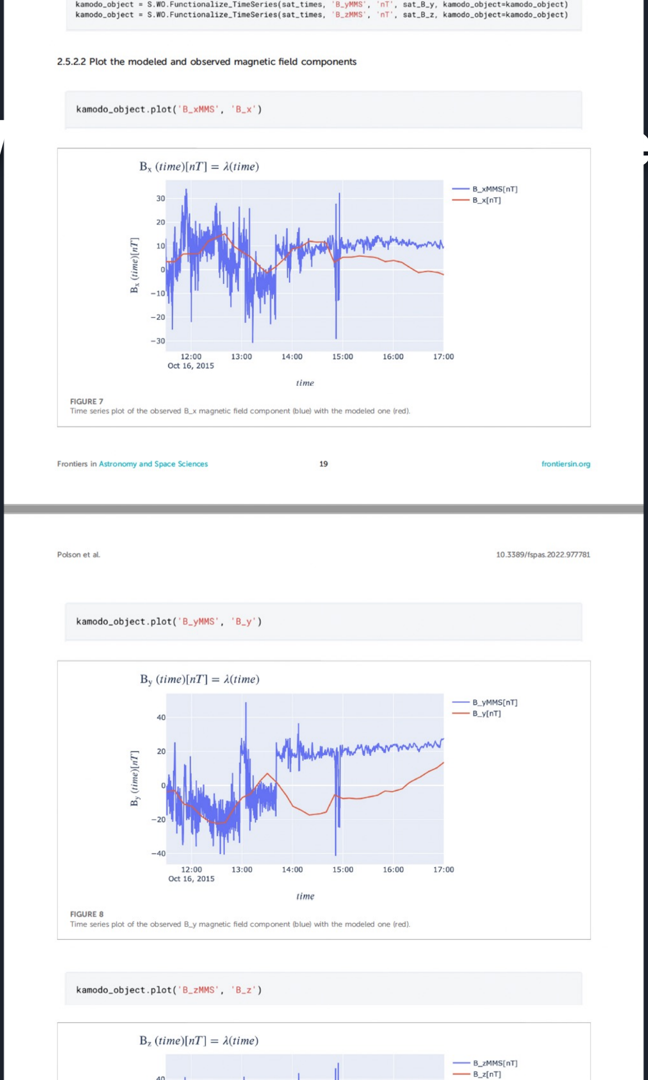
Figure 8

FIGURE 8. Time series plot of the observed B_y magnetic field component (blue) with the modeled one (red).

Figure 9

FIGURE 9. Time series plot of the observed B_z magnetic field component (blue) with the modeled one (red).

W ers was tricky



```
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the modeled and observed magnetic field components

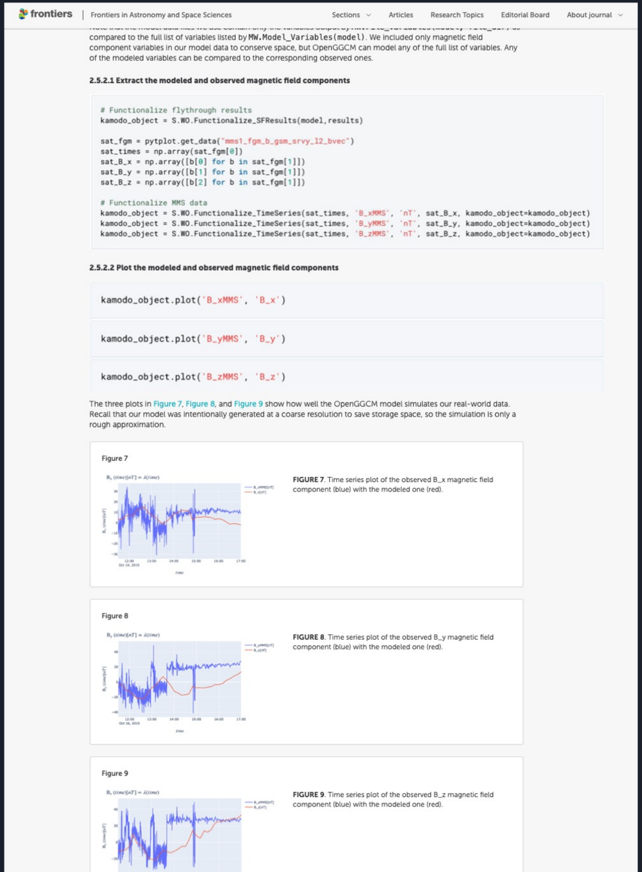
```
kamodo_object.plot('B_xMMS', 'B_x')
```


FIGURE 7
Time series plot of the observed B_x magnetic field component (blue) with the modeled one (red).


```
kamodo_object.plot('B_yMMS', 'B_y')
```


FIGURE 8
Time series plot of the observed B_y magnetic field component (blue) with the modeled one (red).

```
kamodo_object.plot('B_zMMS', 'B_z')
```

frontiers | Frontiers in Astronomy and Space Sciences Sections Articles Research Topics Editorial Board About journal

compared to the full list of variables listed by `MW.Model_Variables(model)`. We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the modeled and observed magnetic field components

```
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data("mms1_fgm_b_gsm_srvy_l2_bvec")
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the modeled and observed magnetic field components

```
kamodo_object.plot('B_xMMS', 'B_x')
```

```
kamodo_object.plot('B_yMMS', 'B_y')
```

```
kamodo_object.plot('B_zMMS', 'B_z')
```

The three plots in Figure 7, Figure 8, and Figure 9 show how well the OpenGGCM model simulates our real-world data. Recall that our model was intentionally generated at a coarse resolution to save storage space, so the simulation is only a rough approximation.

Figure 7

FIGURE 7. Time series plot of the observed B_x magnetic field component (blue) with the modeled one (red).

Figure 8

FIGURE 8. Time series plot of the observed B_y magnetic field component (blue) with the modeled one (red).

Figure 9

FIGURE 9. Time series plot of the observed B_z magnetic field component (blue) with the modeled one (red).

In Conclusion

- Traditional papers face open science and reproducibility problems
- Executable papers combine text and code
- Lessons Learned:
  - Executable papers solve problems
  - Big data is a challenge
  - Journals decide how you represent code
  - Working with typersetters was tricky

Thank you

Contact me:

shawn.polson@lasp.colorado.edu

2.5.2 Visual comparison

We now have values from a model of the magnetic field components measured by our MMS spacecraft. We can plot the modeled and observed values together to see how accurate the model is.

We do so using Kamodo's built-in plotting capability, rather than `matplotlib`, because it produces interactive plots with very little code. We "Kamodofy" the results before we can plot them. "Kamodofication" is a concept Kamodo uses to "functionalize" callable functions, something that allows many problems in scientific data analysis to be posed in terms of function composition and evaluation. See the Kamodofication documentation for details about this concept (Kamodo 2022c).

Note that the model data files we use contain only the variables output by `MW.File_Variables(model, file_dir)` as compared to the full list of variables listed by `MW.Model_Variables(model)`. We included only magnetic field component variables in our model data to conserve space, but OpenGGCM can model any of the full list of variables. Any of the modeled variables can be compared to the corresponding observed ones.

2.5.2.1 Extract the modeled and observed magnetic field components

```
# Functionalize flythrough results
kamodo_object = S.WO.Functionalize_SFResults(model,results)

sat_fgm = pytplot.get_data("mms1_fgm_b_gsm_srvy_l2_bvec")
sat_times = np.array(sat_fgm[0])
sat_B_x = np.array([b[0] for b in sat_fgm[1]])
sat_B_y = np.array([b[1] for b in sat_fgm[1]])
sat_B_z = np.array([b[2] for b in sat_fgm[1]])

# Functionalize MMS data
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_xMMS', 'nT', sat_B_x, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_yMMS', 'nT', sat_B_y, kamodo_object=kamodo_object)
kamodo_object = S.WO.Functionalize_TimeSeries(sat_times, 'B_zMMS', 'nT', sat_B_z, kamodo_object=kamodo_object)
```

2.5.2.2 Plot the modeled and observed magnetic field components

```
kamodo_object.plot('B_xMMS', 'B_x')
```

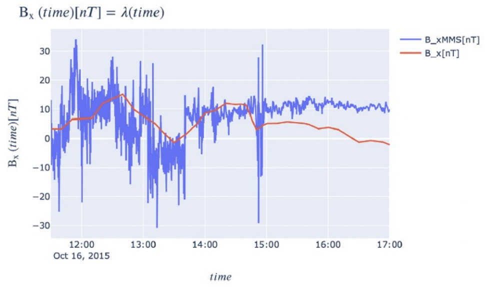


FIGURE 7
Time series plot of the observed B_x magnetic field component (blue) with the modeled one (red).

was tricky