uc3m | Universidad Carlos III de Madrid

Master's Degree in Applied and Computational Mathematics

2022-2023

Master Thesis

BNumMet: A Scholar implementation of Numerical Methods in Python enhanced with interactive widgets

---

Fernando Bellido Pazos

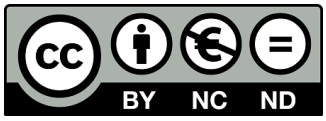
Tutor: Juan Manuel Pérez Pardo

Universidad Carlos III de Madrid

June, 2023

AVOID PLAGIARISM

The University uses the **Turnitin Feedback Studio** for the delivery of student work. This program compares the originality of the work delivered by each student with millions of electronic resources and detects those parts of the text that are copied and pasted. Plagiarizing in a TFM is considered a **Serious Misconduct**, and may result in permanent expulsion from the University.



SUMMARY

The presented end of master's thesis provides an open-source Python package (BNumMet) that gives an academic implementation of numerical methods for the use in undergraduate courses, as well as a graphical user interface over the Jupyter Notebooks framework, all with the intention of assisting students in their learning of the various numerical methods; it also serves as a tool for professors to assist students in learning these methods.

In this project, we discuss previous work done in this field, how the project was carried out, what the fundamental criteria for this project is, what the underlying algorithms are, and whether the solutions proposed are both efficient, comprehensible, and applicable by students.

Overall, this project can be used as a model for the development of different didactic tools in the scientific community in order to increase the number of people who enter the S.T.E.M. (Science, Technology, Engineering, and Mathematics) fields and make the learning and studying of methods that are widely used in the scientific community more accessible for everyone no matter their background.

Keywords: Numerical methods, Open-source, Python, Jupyter, Software development, Linear systems, Interpolation, Nonlinear Equations, Least squares, Random number generators

DEDICATION

For a total of 5 years, I have been expanding my knowledge in Mathematics and Computer Science in higher levels of education; not only was my journey full of ups and downs, but also with outstanding people that made these highs and lows worth every moment.

To say “thank you” would only be an understatement of my true appreciation to these people who have been with me and, I hope, will always be on this journey of enlightenment. To such an extent, I want to give all the credit to the ones that provided me with the opportunities to find new objectives in life, and who always pushed me to strive to be my best self in every aspect of life, that credit is in large part to both my Mother, Maria Pazos Arenal, and Father, Fernando Bellido Alonso-Carriazo, as well as all the members of my family to whom I owe a large part of my very best moments.

I would like to express my deepest and greatest appreciation to my best colleagues who made me laugh even in the darkest of times, my outstanding music group at *Sol Menor* that allowed me to express beyond academic, see beyond reason by expressing and understanding music and teaching me about what teamwork really means.

In addition, I want to express my most sincere thank you to my tutor Juan Manuel Pérez Pardo who has accompanied me through the development of this work as well as the teachers who once pushed me to strive towards the study of Science, those being Concha Elena Alegre, María Pilar Cuesta López, Aurora Insua Ruíz and my other fellow teachers and professors.

> *One looks back with appreciation to the brilliant teachers, but with gratitude to those who touched our human feelings* - Carl Gustav Jung [1]

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

1. INTRODUCTION

Ever since the development of the brain, we humans have been developing various tools and techniques that provided some evolutionary benefits, for instance, the wheel, fire, hunting weapons and all sorts of machinery to make both our life easier and/or have an advantage on the survival of our species. The examples mentioned are just the pinnacles of what has grant us the ability to warm ourselves when we endure the coldest of days, and made us discover the entire world that previously we only could grasp but a small region of it.

Around the Second World War, we, as a society, started to envision a world of computers, machines that will allow us to handle repetitive processes and make calculations that would otherwise take a lot of resources to be carried out. This technological evolution involved a change of mindset in the world of mathematics, machines could now potentially make the more tedious and cumbersome calculations in a short span of time, in contrast to the manual calculations that had been used until then, and the area of – what we now call - numerical methods/analysis was just starting to get its popularity.

It is imperative to acknowledge that prior the rise of computers, numerical methods/analysis has been in a never-ending development and used on a large scale, to name a few:

1. Calculation of logarithms.
2. Interpolation.
3. Finding zeros of a function (Newton's Method).
4. Finite Differences.
5. Least Squares Problem.
6. Quadrature rules.

For a more detailed view of this history, we refer you to [2]. The methods mentioned are only the tip of the iceberg and provide, in some cases, solutions to problems that could only be formulated and not solved using analytical approaches. The impact of computers and numerical approaches combined during its birth can only be grasped, and ever since John Von Neumann the world started to shift towards a computerised mathematical realm.

Today, we have enough powerful computers to take these numerical methods and solve most of these problems, to the extent that even a 6 inch smartphone can perform a vast number of calculations that would have taken years in the twentieth century and, as the legend states, our smartphones have the power required to travel to the moon, but there is one dilemma we still face: How

can we teach future generations these methods in such a way that they find it as natural and intuitive as adding or subtracting? When considering the issue, one could argue that the abrupt rise of technology implies that we must always strive to be on the cutting edge of what our present technological powers are, and how we can apply this knowledge to continue moving forward with this deluge of advances is not to remain stuck in learning the simplest of methods, but it is clear that in order to advance we must know the underlying structure that support all future calculations, all in all, this does not solve the main problem of education.

This end-of-master thesis aims to give an answer to this dilemma, providing both student and professors a framework that contains the scholarly version of the well-known numerical methods developed, using the latest, more popular, economical, easy-to-read, and well-known programming language whilst also granting the ability to see how the algorithms work visually.

To fully achieve our end goal and provide a detailed overview of the process of making this a reality, we will provide the following structure, both to preserve clarity and to provide a road map of the project.

The chapters that will be discussed will be

1. State of the art: In this chapter, we will discuss some of the most recent improvements made to address the challenge described in the introduction, as well as some comments on the answers supplied by the available current solutions.
2. Objectives: Along this chapter, we will establish the key objectives that will drive the main development of this master's thesis, as well as provide the milestones that this project must meet in order to be accepted.
3. Software Development: We will explore the development framework by commenting on the Software development principles and how they relate to project development.
4. Development of BNumMet: This chapter will go through the development of BNumMet, the packages we implemented, and some of the decisions and justifications that went into them, as well as the underlying ideas.
5. Results: As part of the thesis, in this chapter we will describe the results have we obtained in the development of this project

2. STATE OF THE ART

2.1. Mathworks

The underlying idea behind this project comes from [3] in which the author, Cleve Moller, provides a textbook for students to get an introduction to numerical methods using Matlab; not only does this work provide with the guide to understand the fundamental numerical methods, but it also includes attached the code for the various methods he includes, while also creating a Graphical User Interface for each method to help students gain a better understanding of such. The code is written in Matlab, which is a proprietary programming language developed by Mathworks, Inc. and is widely used in the scientific community.

Mathworks has created a series of packages that are freely available to the public. These packages are known as Toolboxes, and they may be downloaded from the Mathwork's website.

Along with the graphical interface, the author created some of the following chapters and the visualization associated:

- Linear Equations: LU Decomposition Visualizer
- Interpolation: Visualization of various interpolation techniques
- Zeros and Roots: Brent-Dekker's Algorithm visualizer
- Least Squares: Applying least squares to the US Census
- Quadrature: Visualization of how quadrature applies
- Ordinary Differential Equations: How different finite difference schemes work
- Fourier Analysis: Visualization of Fourier Analysis
- Random Numbers: Some simple Random Number Generators
- Eigenvalues and Singular Values: Visualization on how they are obtained from the matrix

2.1.1. Author comments on this work

Although C. Moller provides the initial framework for solving the main dilemma we are faced with, there are potential improvements that could be made to the project, among them, the following are proposed:

1. Make use of a programming language that is open-source and free to use while also one that is widely used in the scientific community, in the industry as well as being one that should be popular amongst programmers, this fact will allow for the project to be more accessible and future-proof.
2. Provide an Open Source Licence for the project to be used and improved by anyone while also giving the authors the credit they deserve.
3. Provide a more intuitive and user-friendly Graphical User Interface that is also more aesthetically appealing.

Overall, the work done by C. Moller is a great starting point for the development of this project, but it is not enough to solve the dilemma presented in the Introduction, since it is not open source and it is not free to use and therefore, it is not accessible to everyone to either use or improve. Additionally, it is built on a less popular programming language, according to the TIOBE Index [4].

2.2. End-of-Degree Thesis

One of the attempts to solve the dilemma presented in the Introduction was first tackled by Juan Camilo Bucheli and Juan Manuel Pérez Pardo, as an end-of-degree thesis, [5]. Together, they constructed the first framework for the underlying idea behind BNumMet. They carefully went over the details of the Socio-Economical front and development of the project, and they also discussed both the meaning of what it means for a project to be open source and the licences associated with them. Additionally, they pondered on the reasons of their choice of the programming language and discussed the main and widely-used library of numerical methods (LAPACK).

The authors implemented the Linear Systems Chapter in this project, prioritizing the LU decomposition with Matlab's translation of the LU Decomposition visualizer. They also began developing the Interpolation visualizer using some of the interpolation methods mentioned in the work from Mathworks.

It should be emphasized that the numerical techniques applied are not licensed; they are open-source, with the exception of some tolerance considerations and step selection; the only thing licensed are the visualizers for each chapter.

2.2.1. Author comments on this work

After a closer investigation of the project, we must ask the following questions that derive from this work.

1. Is the licence they had previously chosen still valid with the current evolution of technology?
2. Is Python still the best programming language for these sort of goals?
3. Can we improve the global state of the art?
4. Is the previously done code valid and/or useful for continuing the work?
5. How can we continue the work as the author would have wanted?
6. Are the principles of open source followed?

Even though the code they provided is readable and fairly functional, there exists room for improvement, on the one hand the code must be translated to the English language; reasons vary from being the universal language of coding to the fact that is the most spoken language in the world, thereby enabling access to this project to a vast majority of people. On the other hand, the overall algorithmic could be improved. The following list enumerates some of the weak points behind their project:

1. The use of threading on the LU Graphical User interface can be simplified to not use threading to reduce computer resources as well as improving the readability of the project.

   This raises a few problems such as being more cumbersome to read, understand, maintain and improve. A possible solution is to study carefully the underlying theory of the LU decomposition and then apply it to the code to remove the threading, thereby improving the code.
2. Naming and code conventions to current Python standards.

   This is important because it allows for the code to be more readable and understandable by other programmers. To do this, we must use the different tools that other programmers have created to help us follow the conventions, such as Pylint, Pydocstyle, Flake8, Sonarqube, etc.
3. An overall improvement on how the tests are made.

   The overall testing of the J.C. Bucheli's code is not as good as it could be, it only checks the overall functionality of the code and not the extreme cases that the algorithm has, something which breaks with the definition of testing in the context of programming. In order to fix this, we must make sure that the tests go trhough all the possible cases and lines, whilst making sure that the tests are properly documented.
4. The development of the requirements that PyPI (Python Package Index) requires for it to be properly executable and installed through pip.

   In order for the library to be installed through pip, we must make sure we meet all the requirements PyPi establishes for it to be installed through their installer. Were we to make achieve this, it will be more accessible to students and teachers alike.

5. Updating the Graphical User Interface with the underlying theory of Human-Computer Interaction.

   It is imperative that the G.U.I. is up-to-date to the current standards of usability, as this will help the user to better understand the underlying theory behind the algorithm presented in the interface. This is important because it allows for the project to be more Effective, Efficient, Satisfactory, Safe and Enjoyable which are the measure of usability in the Human-Computer interaction established by the ISO/IEC 9126-4. To do so, we must study the underlying theory of Human-Computer Interaction and then apply it to the Graphical User Interface (controls, layout, contrast, etc.).

2.3. External References

Some external references can be found that attempt to cover this topic, to name a few [6]–[9]. Upon closer examination, these articles attempt to generate a graphical user interface in order to teach, but all of them lack the pedagogical aspect of such, they only develop the solver alongside an interface that only provides the user with input and a click-to-run solver without allowing the end-user to properly choose and see what the algorithm does behind the scenes. Furthermore, the aforementioned publications highlight Matlab's core programming framework, which may be useful in academia but will be a hurdle in the business sector where other languages are employed. Even while the author's work is an outstanding starting point, we must focus on attempting to attain the pedagogical part that they did not fully achieve.

3. OBJECTIVES

Once we have covered the general state of the art, we can extract some essential ideas and goals from the many authors and works that will serve as a road map for this thesis. Consequently, it is critical that we engage in explaining what our goals are and what we want to achieve.

3.1. Description

The primary purpose of this thesis is to create a programming tool that will implement a scholar version of several numerical methods with a graphical user interface (GUI) that will assist students in learning the numerical methods effectively.

With this primary notion, we will offer the requirements for this main goal to be accepted, as well as a project that is getting closer to resolving the fundamental challenge of numerical methods teaching.

The code and the main programming framework should adhere to the following guidelines.

1. It must be developed under the Open-Source Initiative.
2. The language must be in English.
3. The code must be written in an easy-to-read, free, and popular programming language.
4. It must be well documented and readable for students.
5. It must not use external libraries, that is, the numerical methods must be implemented within this library.
6. We should prioritise first being readable but also efficient to the best of our abilities.
7. It must be functional, that is, all methods should pass all unitary tests in the various different versions of the programming language.

On the graphical aspect, the latter objectives also apply, but some new requisites must be pointed out.

1. It must provide insight on one or more aspects of the numerical method.
2. It must let the user interact with the algorithm and observe the changes dynamically.
3. It must be easy to use.

4. It must be modular in the input arguments, that is, we must let the user input their own parameters.

Overall, this objectives will serve to provide a solution to most of the critiques previously done as well as a step closer on solving our dilemma, it must also be noted that if some objectives interfere with each other, the underlying idea of helping the students must be taken into consideration and must be at the core of the project

4. RESULTS

In this chapter we will be stating what are the results obtained from the analysis of previous works, the objectives we set out to achieve as well as giving an overview over the results of the future discussions of the following chapters.

The primary output of this effort is software in the form of a collection of files including the code for the various methods as well as all of the essential prerequisites for it to be a proper Python library. To that purpose, the output will be publicly available on Github, as well as, because it is a Python library, at the primary repository for Python libraries, the links to which are:

- Github Repository: https://github.com/fbpazos/Trabajo-Fin-Master
- PyPi Package: https://pypi.org/project/BNumMet/

One of the outputs, is the documentation provided [Appendix: Documentation] that provides an insight of what the algorithm does as well as some examples that are the same as we will observe in this dissertation.

With all of this, we reckon we have achieved the main objective of the thesis which is developing a self-contained library that contains all sorts of numerical methods written in a way that are both efficient and readable by any undergraduate. Alongside with these algorithms we have developed an intuitive and visually appealing Graphical user interface that gives insight on the algorithms we have developed. We strongly hope that this work will be a valuable contribution to the fields of numerical methods and education.

5. SOFTWARE DEVELOPMENT: THE PROCESS

This project focuses on the creation of BNumMet, a library that implements scientific numerical methods in conjunction with a graphical interface to help students learn. Because of this it is essential to mention the fundamental steps we are going to take on the software development front, since the project is primarily a software development one.

To do this, we will discuss the following aspects using [10] as our project management guide:

1. Initiation: We will establish the framework where BNumMet will take place, indicating the basic roles and the development framework.
2. Planning: We will discuss the scope of the project, the timeline, the required resources, and milestones that indicate progress toward completion.
3. Execution: We will provide comments on how the project will come together structurally and outline the steps this phase must undergo, that is, quality check and monitoring & control.

5.1. Initiation

5.1.1. Roles

The lack of a team per se implied that we had to "simulate" the different roles of an average software development team. To this extent, we have the following division.

- **Juan Manuel Pérez** will execute the client roles providing information on what the end-students and professors would have wanted.
- **Fernando Bellido** will assume the roles of project manager, developer, QA tester and configuration manager.

5.1.2. Life Cycle Model

The development of BNumMet will use an **Iterative Life Cycle**, that is, a cycle in which at each iteration we provide the client or end user with an improved version upon the objections and remark of the previous iteration until all needs are met.

The reason of this choice was predicated on the fact that the requirements were not or could not be a priori estimated, we only had the main objectives

which were general and not specific, they also were client-orientated, that is, the client after each iteration would provide suggestions on what he wanted or how he would want to have the end product.

5.1.3. Development framework

The main framework for the code during the development will be **Python ≥ 3.8**, the reason of this choice was primarly due to being one of the most popular ([4], [11]) programming languages in the market todays and also because the fundamental goal of the project is to help students, that is why, even though C. Moller has made an extensive work on Matlab, Matlab Home Licence is priced at 119€, a price that a vast majority of users will not be able to pay if they want to learn independently. Overall, the choice of Python remains a decent choice for its practicality, readability, future-proof, and economical value.

5.1.4. Licensing

It is imperative that we discuss the licence so that the project is supported by the law. We must first address the general consensus on the categories of licences.

1. Copyleft Licences: Persistent licences, publishing source code is required.
   (a) Strong Copyleft: Viral effect (The aggregation of licences will prioritise the strongest licence to remain).
   i.e. GNU General Public License.
   (b) Weak Copyleft: Without Viral effect.
   i.e GNU Lesser General Public License.
2. Permissive licences: Non-persistent, without viral effect, publishing source code is not required.
   i.e. MIT, Apache 2.0.

5.2. Planning

As stated, we are using an Iterative Life Cycle; therefore, we must indicate how we are going to iterate; in the development of this project the iterations were made every two weeks, with the exceptions of weeks which were incompatible with the correct development of the Master courses.

Each iteration will have the project manager present the changes made to the client and then the client will provide feedback and next steps. In between iterations, the feedback must be implemented as well as ensuring its quality.

The project originally started on October 2022 with iterations starting on the 6th of that month, the ending will be limited to the end of the Master and the deadline for the thesis to be presented; therefore, it must be ended before May 2023, from then on only small fixes will be applied to the project.

5.3. Execution

5.3.1. Quality Assurance

Software has an intrinsic problem, it cannot be tested, and its correct functioning is not ensured; therefore, we shall try to provide quality standards. In particular, during the development of BNumMet, we will provide two different levels of quality assurance.

1. Unitary Tests: Is a type of software testing in which individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit tests isolate a section of code and verify its correctness.
2. Code Reviewer: In particular, we will use the tool known as SonarQube. SonarQube is a self-managed automatic code review tool that systematically helps to deliver clean code. It integrates into our existing workflow and detects issues in our code to help us perform continuous code inspections of our projects. The tool performs an analysis to the adherence to coding standards and core values such as comment percentage, cognitive, and cyclomatic complexity to ensure that our code meets high-quality standards [12], [13]. At the end of this document the latest report of SonarQube from the latest version of BNumMet will be attached [Appendix: Documentation].

5.3.2. Software configuration management

Software configuration management is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management [14].

To do such task, we will proceed by using GitHub, which is defined as an internet hosting service for software development and version control using Git (distributed version control). To such an extent, we will define our general base line for the project.

- .github: Folder dedicated to Github Actions
- Latex: Folder dedicated to Latex files or other closeyly related to the generation of the written thesis.
- Python_BNumMet: Folder dedicated to coding BNumMet

All files within the directories will be added/removed accordingly to the execution phase, all backups will be provided by GitHub and, in the case of latex files, by saving the current state of the 'commit' inside another folder on a separate disk drive

Versioning

As part of configuration management, we should proceed to define how are we going to keep track of changes and to establish which is the latest version of them all

Development Versioning During the first stages of the execution phase, the versioning will be done by using the date of the year in the following format DD-MM-YYYY.

In the final stages, the new versioning scheme will be provided to clarify that the project was in its final stages to be published, and it follows the following scheme.

<major>.<minor>.<patch>**dev**<development Number>

End-User Versioning As per the final versioning, the one made to announce to the end user an updated version will follow a scheme similar to the one in the last stages of the development phase.

<major>.<minor>.<patch>

5.4. Ending

Even though more work can be done, the final stage of the development process will be accompanied by making the development repository public, as well as the PyPi source that users can use to fully use the project.

6. DEVELOPMENT OF BNUMMET

6.1. General framework of the Project

6.1.1. Licensing

One of the first considerations we must take into establishing the development of the project is the License, as discussed in chapter 3, there are 2 major types of licences, but since our project idea was based on the previous work from J.C. Bucheli but ended up not using J.C. Bucheli's code after a careful analysis of their implementation, we are not legally required to use the same licence, and due to the nature of the project, we will proceed to use the *GNU Affero General Public Licence v3.0 (AGPLv3)* which is the strongest copyleft licence. The GNU Affero General Public Licence v3.0 is a free copyleft licence for software and other kinds of work. It is specifically designed to ensure cooperation with the community in the case of network server software [15]. The licence is intended to guarantee the freedom of users to share and change all versions of a program, ensuring that it remains free software for all its users [15].

AGPLv3 was created to address a specific problem: How to protect the rights of a user when the program is used over a network [16]. Its terms effectively consist of the terms of GPLv3, with an additional paragraph in Section 13 to allow users who interact with the licensed software over a network to receive the source of that program [17].

The idea behind it is to address the flaw other licences have, which is, in short, to take profit of the software and modify it internally without having the need to make these changes available to the public.

For more details on what specific licenses have to offer can be looked up at [18], [19].

6.1.2. Software configuration management

To properly execute the Software configuration management we will be using one of the benefits of GitHub and, in particular, the version that provides students free of charge the features of GitHub Pro which is the use of automations, or as they call it actions, that can be done every time a new version of the project is posted on GitHub.

We have used three types of automations that help us both maintain the code in accordance with the tests and provide "quality of life" improvements.

- Latex Compilation: If a file inside the Latex folder is updated, GitHub - once committed - will compile the latex files and provide the PDF that was generated, this is primarily a "quality of life" action, since it is not critical to the development, it only provides us with the latest PDF without needing a computer with a Latex compiler.
- Python Tests: As the name indicates, this tests the correct functioning of our Python library by using the tests made on an external machine. The reason this action is crucial is because during development, it is easier to adapt to one's own machine and not an external one (even with the use of a virtual Python environment). GitHub automates this testing, and, as we have done, it tests it on different operating systems with the four latest versions of Python, to ensure both universality of the code and the correct functioning at every version.
- Upload Python Package: This action provides the scripting required to upload the Python package to the PyPi directory.

All the scripting needed for this workflow to function will be attached in [Appendix: Github Actions].

6.1.3. Quality Assurance

Unitary Tests

As part of assuring that our final results will be functional, we will - as previously discussed - develop unitary test, the library will be Pytest, the reason of the choice was predicated on the fact that this library offers a more detailed analysis if a test has failed as well as it offers coverage reports which will be of great use for the code reviewer and will provide us how many lines of code did we miss in our test.

Code Reviewer

As we indicated in the execution phase, testing or verifying the quality of our project is a critical stage; one of the tools recommended is SonarQube; we will now cover the steps we had to take to correctly develop this Code Reviewer.

We utilized Docker to run Sonarqube (an instance of the service), docker is defined as a set of platform as a service technologies that leverage OS-level virtualization to distribute software in packages called containers [20]. To install and run the Sonarqube instance we have introduced the command into the terminal in order to set up the container and the flow of the container.

```
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

This command will pull the container of the official Sonarqube and set up a bridge between the localhost of our computer and the localhost of the container.

It should be noted, however, that this will not automatically analyze every change in the code, but only when it is used. To invoke it, we must start the Sonar scanner, which will also be employed using docker to conserve memory. This latter tool will do the necessary procedures to build the report that will be allocated in SonarQube's instance; in order to correctly execute this, we have developed the following command.

```
docker run --rm -ti -v "%cd%":"/usr/src" --link sonarqube \
newtmitch/sonar-scanner sonar-scanner \
-Dsonar.login="" \
-Dsonar.password="" \
-Dsonar.projectName="BNumMet" \
-Dsonar.projectKey="BNumMet" \
-Dsonar.sources="src/BNumMet/" \
-Dsonar.python.version=3 \
-Dsonar.python.xunit.reportPath="tests/Reports/testsReport.
   xml" \
-Dsonar.python.coverage.reportPaths="tests/Reports/Coverage/
   xml/coverage.xml" \
-Dsonar.scm.disabled=true \
-Dsonar.tests="tests" \
-Dsonar.test.inclusions="tests/**" \
-Dsonar.test.exclusions="tests/Reports/Coverage/**"
```

This command not only establishes the link between the two containers (SonarQube and SonarScanner), but it also establishes the link between where all the reports, tests, and code will be for it to scan.

The key reason for the pick is that it is extensively used in the industry and provides the mobility that we want during development, since we do not need to install anything and it provides a free community version.

It must be noted that we have extended the functionality of Sonarqube by adding a community plugin known as "sonar-cnes-report" [https://github.com/cnescatlab/sonar-cnes-report], this will provide a Word, Excel, Markdown report of the latest analysis of the code, we have used it to provide the final user the analysis of the code. We can see the latest report in [Appendix: SonarQube Report].

6.2. Linear Systems

During this section, the following algorithms that we have carefully implemented will be discussed:

- LU Decomposition [Algorithm 2]
- Forward Decomposition [Algorithm 3]
- Backward Decomposition [Algorithm 4]
- LU Solver [Algorithm 5]
- QR Factorization [Algorithm 6]
- QR Solver [Algorithm 7]

6.2.1. LU Decomposition

The (P)LU decomposition, developed in this package, factors a matrix into a product of three matrices PLU, P being a permutation matrix, L a lower triangular matrix and finally U an upper triangular matrix. This approach is usually used to solve linear systems; however, it may be applied in a variety of ways.

During the development of such method, some considerations have been taken into account; firstly, we put all the variables in the English language as well as a choosing a naming that will provide the future student some hints on what each variable is doing.

Second, we established an auxiliary function that would execute the permutation, this function will have as an input the matrix A and the number of the two rows to permute i, j; this option was made mostly for students to comprehend the stages prior to the use of notation; the notation for permuting was `A[[i, j], :] = A[[j,i], :]`, so the development of a function that performs it will be easier for students to read.

Finally, we examine three main approaches that a student may take to develop lines 11–15 from [Algorithm 2], we will explore the various ways as well as the one built utilizing the well-known Scipy package in the next subsection, where we confirm that the choice of a for-loop is more efficient and easier to read.

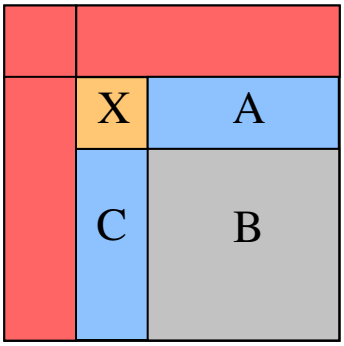
This algorithm was primarly based upon J.C. Bucheli's work [5].

Matrix reduction implementation analysis

Whilst implementing this method, we observed numerous methods a student may implement the LU Decomposition when designing it, which is why some discussion about whose implementation is superior should be addressed.

The multiple ways refer to the critical calculation of the reduction $A'[row, col+1:n] \leftarrow A'[row, col+1:n] - multiplier \cdot A'[col, col+1:n]$, this operation can be visualized as follows:

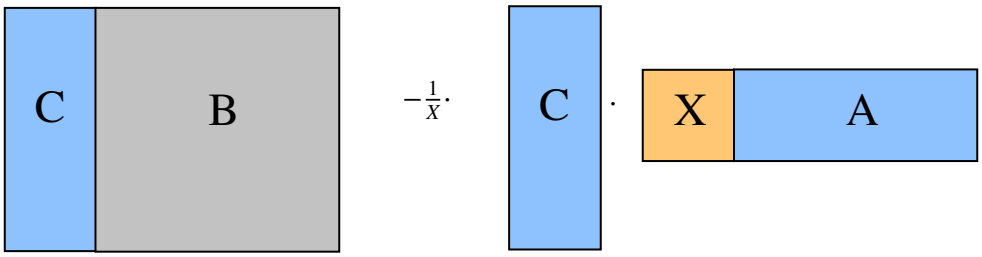
Suppose we have the following matrix, where the red-colored elements are already calculated, and the rest will be the next iteration



The operation we are focused in is the following:

$$[C|B] - \frac{[C]}{X} \cdot [X|A]$$

Visually, this can be seen as:



The main problem with this operation is within how Python's Numpy interprets the matrices $[C]$ and $[X|A]$. It interprets it as a vector, not as a column/row matrix. This is a problem because the operation for multipliying matrices and vectors are different.

In order to solve the problem we have multiple options to choose from:

1. By Submatrices Syntax

   The first option is to use the submatrices syntax, which can be summirized as taking a submatrix of a matrix using numpy syntax, doing so it will preserve the form of the matrix and not as a vector.

```
A[col + 1 :, col] = A[col + 1 :, col] / pivot
A[col + 1 :, col + 1 :] -= A[col + 1 :, :][:, [col]] @ A
    [[col], :][:, col + 1 :]
```

   Note: The '@' symbol is the matrix multiplication operation in numpy.

2. By New Axis Syntax

   The second option is to use the new axis syntax, which can be summirized as tacking directly the vector and making numpy to add a new axis to it, this way we transform the vector into a matrix and we can use the matrix multiplication operation.

```
A[col + 1 :, col] = A[col + 1 :, col] / pivot
A[col + 1 :, col + 1 :] -= A[col + 1 :, col][:, np.
    newaxis] @ A[col, col + 1 :][np.newaxis, :]
```

   Note: The '@' symbol is the matrix multiplication operation in numpy.

3. By using the Outer Product

   The outer product is a matrix operation that takes two vectors and returns a matrix, this operation is the same as the new axis syntax but it is more explicit and easier to understand.

```
A[col + 1 :, col] = A[col + 1 :, col] / pivot
A[col + 1 :, col + 1 :] -= np.outer(A[col + 1 :, col], A[
    col, col + 1 :])
```

4. Using a for loop

   The last option is to use a for loop to iterate over the rows and columns of the matrix, this way we can avoid the problem of the vector interpretation and we can use the normal multiplication operation of elements. This option is by far the first one that comes to mind and the most intuitive one, generally speaking.

```
for row in range(col + 1, n):
    A[row, col] = A[row, col] / A[col, col]  # Calculate
        the multiplier and store in A for later use
    A[row, col + 1 :] -= A[row, col] * A[col, col + 1 :]
         # Update the remaining elements in the row using
         the multiplier
```

To test these methods we will proceed by creating 100 random matrices for the sizes [500, 1000, 1500, 2000, 2500, 3000, 3500, 4000] and measure the length of time it takes for the algorithm to calculate the LU factorisation for each method. Note that all methods have exactly the same syntax before and after the options we have; therefore, we reduce possible noise.

Results The results of the mean of those 100 iterations for each size in seconds are as follows.

Matrix Size	Submatrices	New Axis	Outer Product	Loop	scipy
500	0.41335	0.169461	0.117083	0.465405	0.018257
1000	2.563606	1.522018	1.110119	2.176568	0.03057
1500	8.535015	5.212411	3.8087	5.579113	0.069611
2000	20.30588	12.333257	9.134406	10.942175	0.132066
2500	39.145535	23.998247	17.653031	19.098659	0.216267
3000	68.789119	41.919219	31.590708	29.468995	0.42402
3500	106.423081	64.700578	47.661064	42.445734	0.447992
4000	157.028937	96.519873	70.584879	58.448117	0.603978

TABLE 6.1. LU METHODS DATA IN SECONDS

On a visual plot:

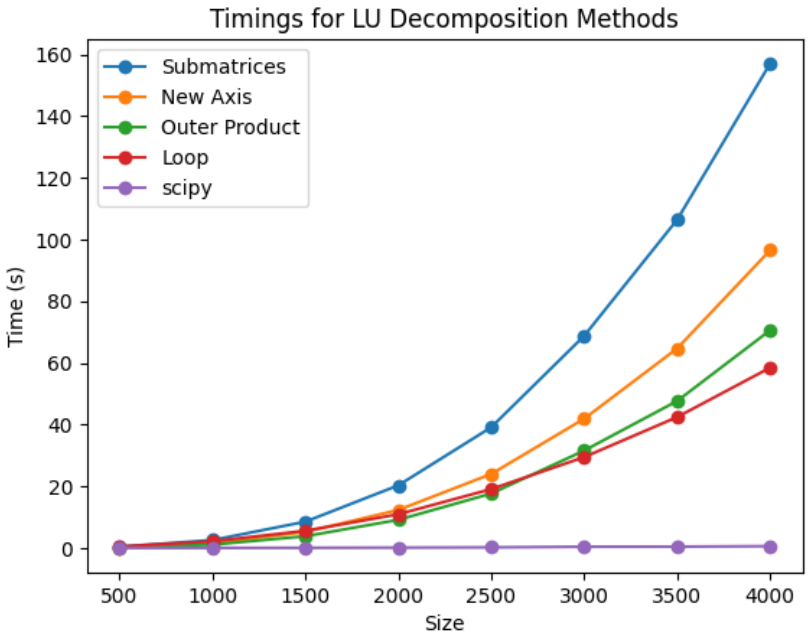


Fig. 6.1. LU Methods timings comparison

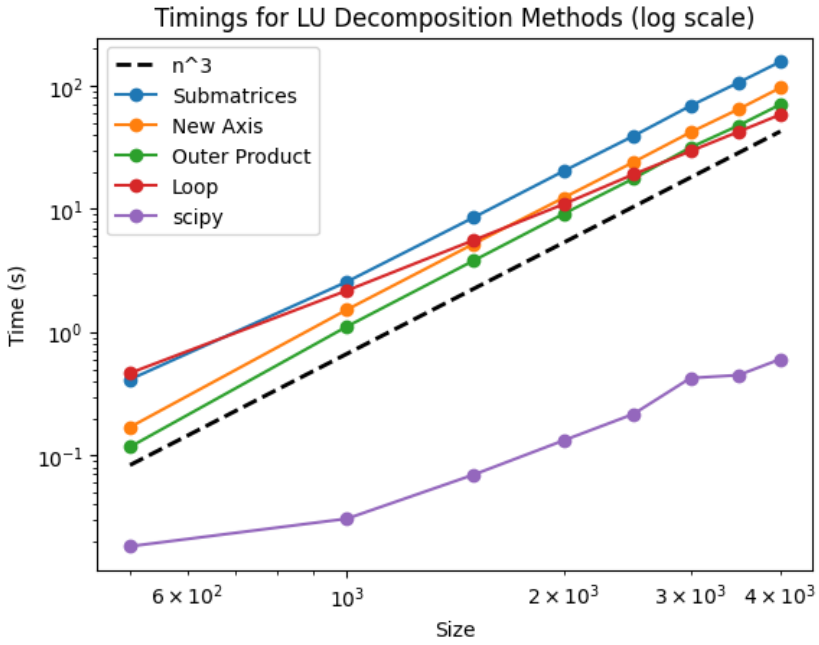


Fig. 6.2. Log plot of LU methods timings

As we can observe from the tabular data and the plot, Scipy's method is by far the fastest, that is, due to the implementation using LAPACK's library [21] which is compiled amongst other things, and the slowest is using the Submatrices syntax; surprisingly, the syntax using a for-loop improves over the other implementations as size increases, therefore even though this works for large sizes, using a for-loop that implements the reduction is the best choice overall the other methods (without accounting for Scipy's) since it is more readable and easier to understand, which is a key aspect in this work since we are trying to make the code as readable as possible for pupils.

For mere due diligence, we have calculate the slopes using linear regression.

	Submatrices	**New Axis**	**Outer Product**	**Loop**	**scipy**
slope	2.885234	3.043451	3.071779	2.334054	1.813654

TABLE 6.2. LU DATA SLOPES OF LINEAR REGRESSION

In this last table, we can see that although the submatrices syntax was the slowest in the previous plots, is better than using the New Axis syntax, it must be noted that this will only be seen when the size of the matrix is humongous, and barely reachable in a sort span of time.

Examples

Example 1

```
from BNumMet.LinearSystems import lu
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
P, L, U = lu(A)
display(P, L, U)

>> P = array([ [1., 0., 0.],
               [0., 0., 1.],
               [0., 1., 0.]])
>> L = array([ [ 1.  ,  0.  ,  0.  ],
               [ 0.5 ,  1.  ,  0.  ],
               [-0.3 , -0.04,  1.  ]])
>> U = array([ [10. , -7. ,  0. ],
               [ 0. ,  2.5,  5. ],
               [ 0. ,  0. ,  6.2]])
```

6.2.2. Forward and Backward Substitution

Undoubtedly, one of the key steps in solving a linear system is substitution, where we find the actual solutions to our system, that is why there exists two main numerical methods that are use, which are Forward and Backward substitution. These methods take a triangular matrix (lower and upper, respectively) and make the necessary substitutions.

The first approach, forward substitution, substitutes the first row and then moves "forward" in the next variables using the already computed values. Backward substitution is analogous, but instead of starting with the first variable, it starts with the last (since this method takes an upper triangular matrix) and moves backwards until it has reached the first variable.

The reason of implementing this method is that we want BNumMet to be self-contained, that is, we want to have all the necessary tools to solve a linear system without the need of external libraries.

The algorithms were based upon Fernando Terán de Vergara's course notes for the Master in Applied & Computational Mathematics [22].

Examples

Forward Substitution

Example 1

```
from BNumMet.LinearSystems import lu, forward_substitution
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
P, L, U = lu(A)
```

```
solution_aux = forward_substitution(L, P @ b)
display(solution_aux)

>> solution_aux = array([7. , 2.5, 6.2])
```

Backward Substitution

Example 1

```
from BNumMet.LinearSystems import lu, forward_substitution,
    backward_substitution
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
P, L, U = lu(A)
solution_aux = forward_substitution(L, P @ b)
solution = backward_substitution(U, solution_aux)
display(solution)

>> solution = array([ 0., -1.,  1.])
```

6.2.3. QR Decomposition

Alongside the LU decomposition, there exists another decomposition known as the QR Decomposition, similarly to the LU decomposition, the concept behind QR is decompose a matrix into a product of two, in this case, it will be a product of an orthogonal matrix (Q) and an upper triangular matrix(R). We have implemented this technique using Householder reflectors which is the fundamental method to implement this decomposition.

The reason of implementing these method as a standalone is to give an additional feature to BNumMet and also to the students.

The algorithm was based upon Fernando Terán de Vergara's course notes for the Master in Applied & Computational Mathematics [22].

Examples

Example 1

```
from BNumMet.LinearSystems import qr_factorization
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
Q, R = qr_factorization(A)
display(Q,R)

>> Q  =
    array([[-0.86386843,  0.42560398,  0.26943013],
           [ 0.25916053, -0.08312578,  0.96225045],
           [-0.43193421, -0.90108343,  0.03849002]])
>> R =
```

```
    array([[-11.5758369 ,   6.99733425,  -0.6047079 ],
           [  0.        ,  -2.24439601,  -5.00417184],
           [  0.        ,   0.        ,   5.96595278]])
>> Q@R =
    array([[ 1.00000000e+01, -7.00000000e+00, -4.67352964e
        -16],
           [-3.00000000e+00,  2.00000000e+00,  6.00000000e
              +00],
           [ 5.00000000e+00, -1.00000000e+00,  5.00000000e
              +00]])
```

6.2.4. LU Solver

We implemented the function that solves a linear system $Ax = b$ using the LU factorization and forward and backward substitution as part of the Linear Systems Package. The reason for implementing these is to add extra features to BNumMet as well as give the student a written version of the solver, as it is trivial to implement once you have the aforementioned methods. Apart from appropriately commenting the code, no other considerations have been made.

Examples

Example 1

```
from BNumMet.LinearSystems import lu_solve
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
solution = lu_solve(A, b)
display(solution)

>> solution = array([ 0., -1.,  1.])
```

6.2.5. QR Solver

Similarly to the LU Solver, we have added a functionality that will be critical on solving the Least Squares Problem, that is the QR solver. The reason for the QR solver, aside from being used in the Least Squares Problem to avoid the ill-conditioning of solving the Least Squares Problem using the well-known formula of $A^TAx = A^Tb$, is that it implements an optimized version of QR taking into account the fact that we are solving a Linear System, in this optimized version, the algorithm does not compute the Q matrix but rather it applies it to the independent terms wihtout storing such matrix and then uses the backward substitution on R (which is computed).

Not only does this method provides students with another approach on solving linear systems or the Least Squares Problem, but it also grants the underlying procedure of the well-known Matlab Backslash.

The algorithm is based upon C.Moller's [3] and Fernando Terán's work [22]

Examples

Example 1

```
from BNumMet.LinearSystems import qr_solve
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
solution = qr_solve(A, b)
display(solution)

>> solution =
        array([-3.83634647e-16, -1.00000000e+00,  1.00000000e
           +00])
```

6.3. Interpolation

Undoubtedly, one of the pinnacles of numerical methods is trying to find a function that passes through a fixed set of points, in hopes of understading how these points are related to each other, all this using what is known as an 'Interpolating Function' which is a function that passes through all the points of a given set. Many different functions exists for a given set of points. In this package, we have built some of the various methods for interpolating a function; however, despite the simplicity of the method's implementation, a general analysis has been conducted on a specific method of implementing the function evaluation; this analysis will be presented after a brief description of the methods.

The following algorithm implementations that we have realized will be detailed in this section:

- Polinomial Interpolation [Algorithm 8]
- Piecewise Linear Interpolation [Algorithm 9]
- Piecewise Cubic Hermite Interpolation [Algorithm 10]
- Piecewise Cubic Interpolatory Splines [Algorithm 11]

All of the algoritmhs have three main parameters,

- **X**: The x-coordinates of the points we want to interpolate.
- **Y**: The y-coordinates of the points we want to interpolate.

- **U**: The x-coordinates of the points we want to evaluate. This is what we call the mesh.

And all of them return the y-coordinates of the points we want to evaluate.

Some of the above algorithms were based uppon C. Moller's reference [3].

6.3.1. Polynomial Interpolation

This approach performs a polynomial interpolation of a given collection of points; it is one of the fundamental ways of interpolation and the root in the solutions of many error estimation problems.

The main goal of this interpolation is to obtain a polynomial of degree $n - 1$ (being n the number of points) that passes through all the points of the set. The algorithm implemented uses the Lagrange interpolation formula [23].

Examples

Example 1

```
from BNumMet.Interpolation import polinomial
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
# Plotting using Matplotlib
v = polinomial(x, y, u)
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("Polinomial Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

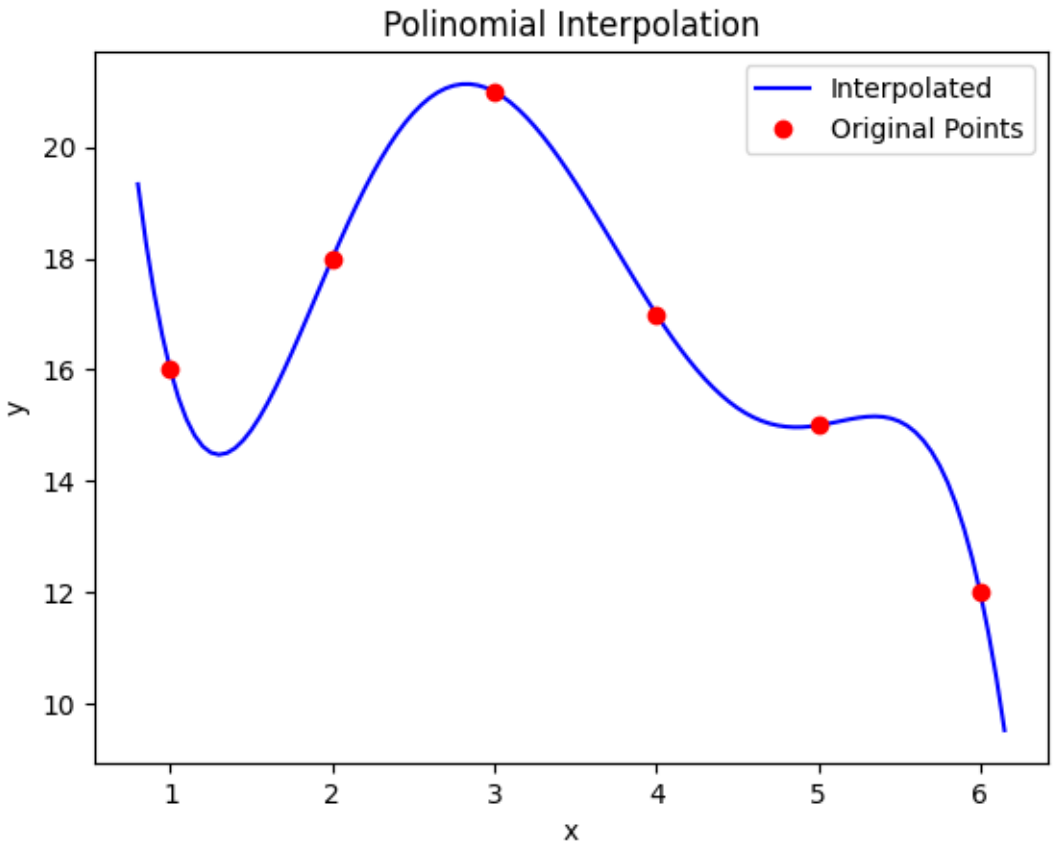


Fig. 6.3. Polinomial Linear Example 1

6.3.2. Piecewise linear interpolation

Following the polynomial interpolation approach, the next step is to extend this notion to piecewise functions; but firstly we will make an intial step using linear functions, that is, we will interpolate the points using a line that passes through two points, this is known as piecewise linear interpolation.

The way the algorithms works is to find the closest two points from the the point of the mesh all within the given dataset, and the use the calculated slope in order to evaluate the point of the mesh.

Examples

Example 1

```
from BNumMet . Interpolation import piecewise_linear
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
v = piecewise_linear(x, y, u)
# Plotting using Matplotlib
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
```

```
plt.legend()
plt.title("Piecewise Linear Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

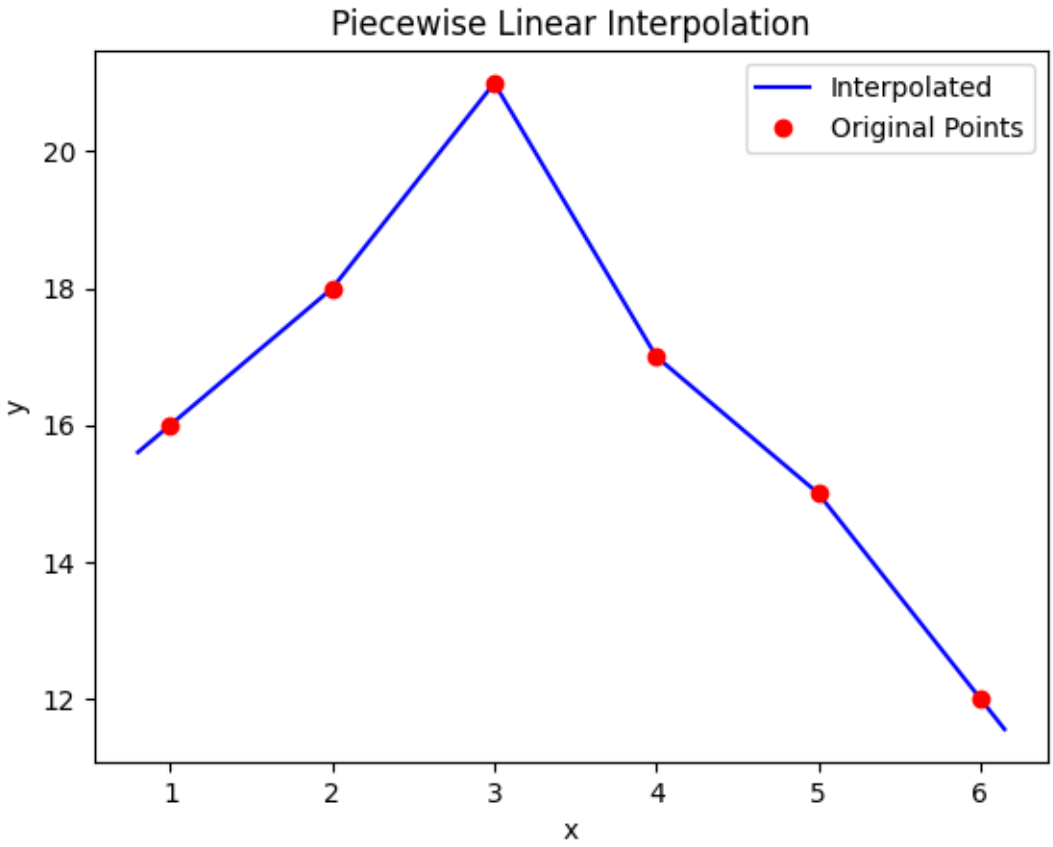


Fig. 6.4. PieceWise Linear Example 1

6.3.3. Piecewise cubic interpolation

Following the Piecewise Linear interpolation we are going to use the same idea but with a cubic function. The way this interpolating method works is between to points we try to obtain a cubic function, to do so we have four parameters that we needed to determine a cubic polynomial we will need extra information, in this case we require the second derivatives (using an approximation).

This interpolating polynomial is twice continuously differentiable, and hence each piecewise function is a cubic spline, as well as, having the boundary conditions known as 'not-a-knot' [3] which as Mathwork's states at the first and last interior break, the third derivative of the interpolating function is continuous (up to round-off error) [24].

Examples

Example 1

```
from BNumMet.Interpolation import splines
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
v = splines(x, y, u)
# Plotting using Matplotlib
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("Spline Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

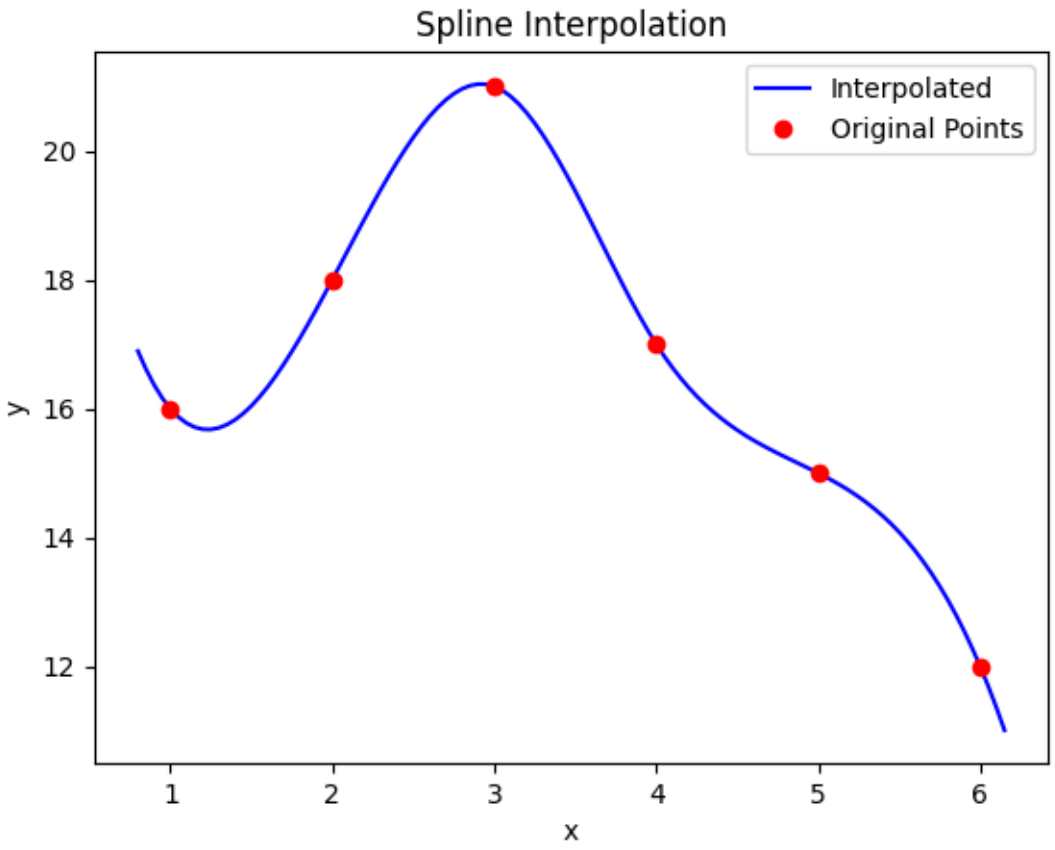


Fig. 6.5. Splines Example 1

6.3.4. Piecewise Cubic Hermite interpolation

To end this section, we will present a different approach to piecewise cubic interpolation, this time we will use the Piecewise Cubic Hermite Interpolation Polynomial (P.C.H.I.P.) which is a form of cubic spline interpolation that

uses, as per the name indicates, cubic polynomials but whose slopes are explicitly given (in the case of this algorithm they are calcutated according to Mathworks implementation), in the algorithm this is done by looking at the sign of the slopes (using an approximation to the derivative) and then using the harmonic mean to calculate the slope of the interpolating polynomial. This works for the interior points, for the exterior ones it uses a non-center, shape-preserving, three-point formula .

The 'pchip' function is a Python implementation of the Piecewise Cubic Hermite Interpolation Polynomial (P.C.H.I.P.) based on an old Fortran program by Fritsch and Carlson [25].

Examples

Example 1

```
from BNumMet.Interpolation import pchip
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
v = pchip(x, y, u)
# Plotting using Matplotlib
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("PCHIP Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

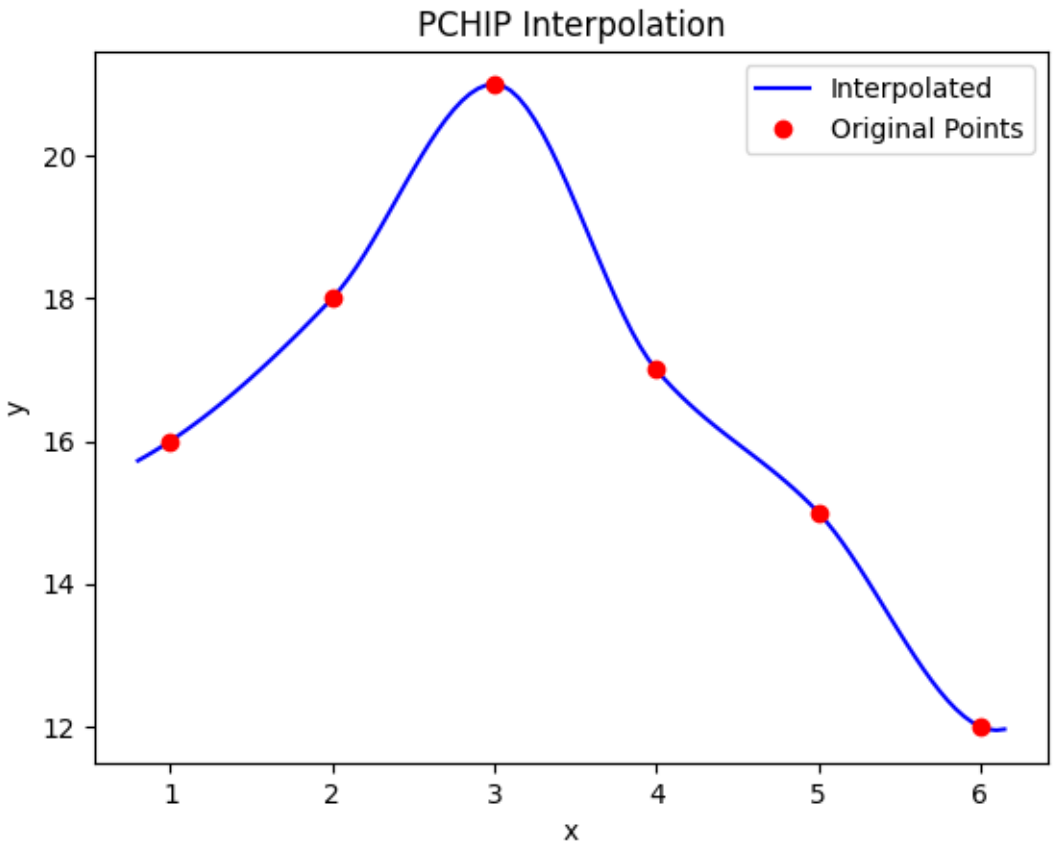


Fig. 6.6. PCHIP Example 1

6.3.5. Interpolation implementation analysis

Similar to the discussion in the linear systems package, we discovered numerous ways a student could implement one part of the method, this one part refers to the lines of code that evaluates the value ($f(x)$) of the points of the mesh.

In short, for interpolating, one needs to compute the interpolating polynomial as well as evaluating it. The following piece of code is related with the evaluation of piecewise interpolation methods. Particularly, one needs to find in which subinterval a point of the mesh is in, all done to then properly evaluate the interpolating function at that point

The code section may be summarized as follows:

Algorithm 1: Extract from Interpolation's algorithm

```
for j ← 1 to n − 2 do
    k[x_j <= u] ← j
end
s ← u − x_k
v ← y_k + s ∗ Δ_k
```

As one can see, this portion of the code assign values to an array k which is then used to access the index of the array y and x. This is done in order to apply the specific formula to the appropiate section of the mesh. But, we know for a fact that k is an array and we cannot use it as index, per se, thus we need to find a way to access the index of the array y and x using the values of k.

Note: It should be noted that other algorithms may have different calculations, however for the sake of simplicity, we are generalizing lines 4 and 5 of code, since all piecewise interpolating functions chooses different formulas for this step.

We might implement this section of code in a variety of ways, including:

1. List Comprehension: Python offers a compact form for adding items to a list, in particular list comprehension, for a general reader a list comprehension is based on mathematical notation that is $\{3n \,|n \in \mathbb{N},\ 0 \leq n \leq 25\}$ can be written in Python as `[3*n for n in range(0,26)]`, generally speaking list comprehension is faster than classical loops [26], however, a better approach would be using functional maps which are cognitively more difficult to understand, thus not the purpose of this project.

   This is an approach that will iterate through the elements in k and therefore we will be able to access the index of the array y and x.

2. Using NumPy's *fancy* Indexing: Similarly to Matlab, NumPy offers a way to broadcast a list, that is we can access the items of a list using another list. This is an approach developed by NumPy that will iterate through the elements in k in a sugar-coated syntax way. In itself, it is a list comprehension but with a different syntax and different internal mechanisms.

   Altough it is cognitively more complicated than list comprehensions; it is a syntax that is widely used in the numerical methods realm and might be a fair competitor over list comprehensions.

Results To correctly test the implementations we will fix 6 interpolation pairs and then run 100 iterations for different meshes sizes with random values (to also check the ability to sort the mesh points), then after those 100 iterations are over, we will calculate the mean, the results are the following:

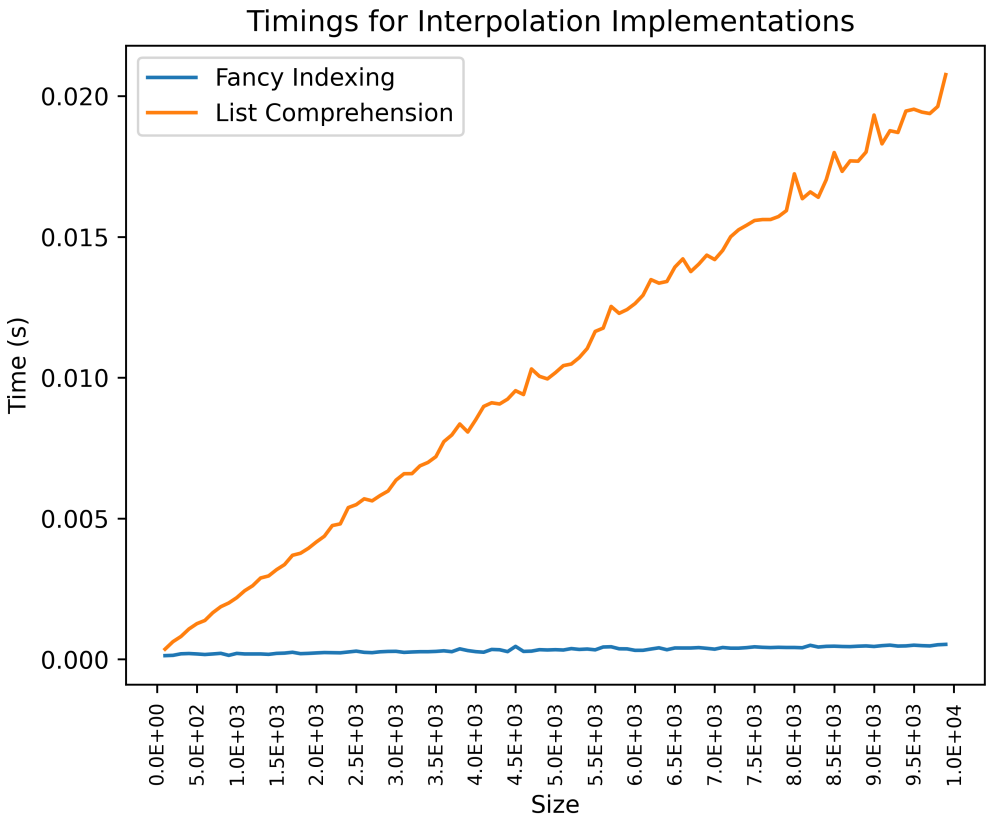


Fig. 6.7. Interpolation Timings

As we can visually observe, the use of List Comprehensions is by far the worst of the two, and should be discarded for this particular case. We can also see how much the *fancy* indexing improves the overall speed.Applying linear regression on both sets of data we get the following results:

1. **Linear Regression *fancy* indexing**:
   $y = 3.4812 \cdot 10^{-08}x + 0.0002$
   Slope: $3.48119 \cdot 10^{-08}$
2. **Linear Regression List Compresion**:
   $y = 2.0237 \cdot 10^{-06}x + 0.0003$
   Slope: $2.02374 \cdot 10^{-06}$

Overall, not only does the choice of *fancy* indexing is by far the fastest (around 2 orders of magnitude) but it also provides students a syntax that will be used during their learning of numerical methods, though it must be noted, the use of for-loops might be ideal for didactic purposes to teach students what could be behind the notation of this *fancy* indexing.

6.4. Non Linear Equation Solver Package

As part of extending BNumMet's functionality, we decided to implement the numerical methods associated with solving nonlinear equations because it is a topic that, like the others, is widely taught in numerical methods courses and is a fundamental tool in numerical analysis since it provides a unique and only way of finding solutions of non-lnear equations by numerical approaches.

The following algorithm implementations that we have realized will be detailed in this section:

- Bisection method [Algorithm 12]
- Secant method [Algorithm 13]
- Newton's Method [Algorithm 14]
- Inverse Quadratic Interpolation [Algorithm 15]
- Brent-Dekker [Algorithm 16]

Except for the last method, all of the approaches are quite straightforward, with the main exception being that the input function may be multidimensional, but only the zero will be located in one of the dimensions. In addition, an iteration counter has been introduced for student study of the efficiency of the various methods.

6.4.1. Bisection

We have implemented the well known method of the bisection method, a method that at every iteration divides the search interval into two half's and takes the one whose sign at the extremes change. No extra considerations have been taken into account except for adding the appropriate comments for any student to know what every step is doing.

The algorithm is based upon C. Moller's implementation [3].

Examples

Example 1

```
from BNumMet.NonLinear import bisect
fun = lambda x: x**2 - 2
interval = [1, 2]
sol = bisect(fun, interval, iters = False)
print("Bisection method: x = %f" % sol)

>> Bisection method: x = 1.414214
```

Example 2

```
from BNumMet.NonLinear import bisect
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
     for the n-th zero

zeros = [bisect(f, interval(n)) for n in range(0, 10)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with bisection method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

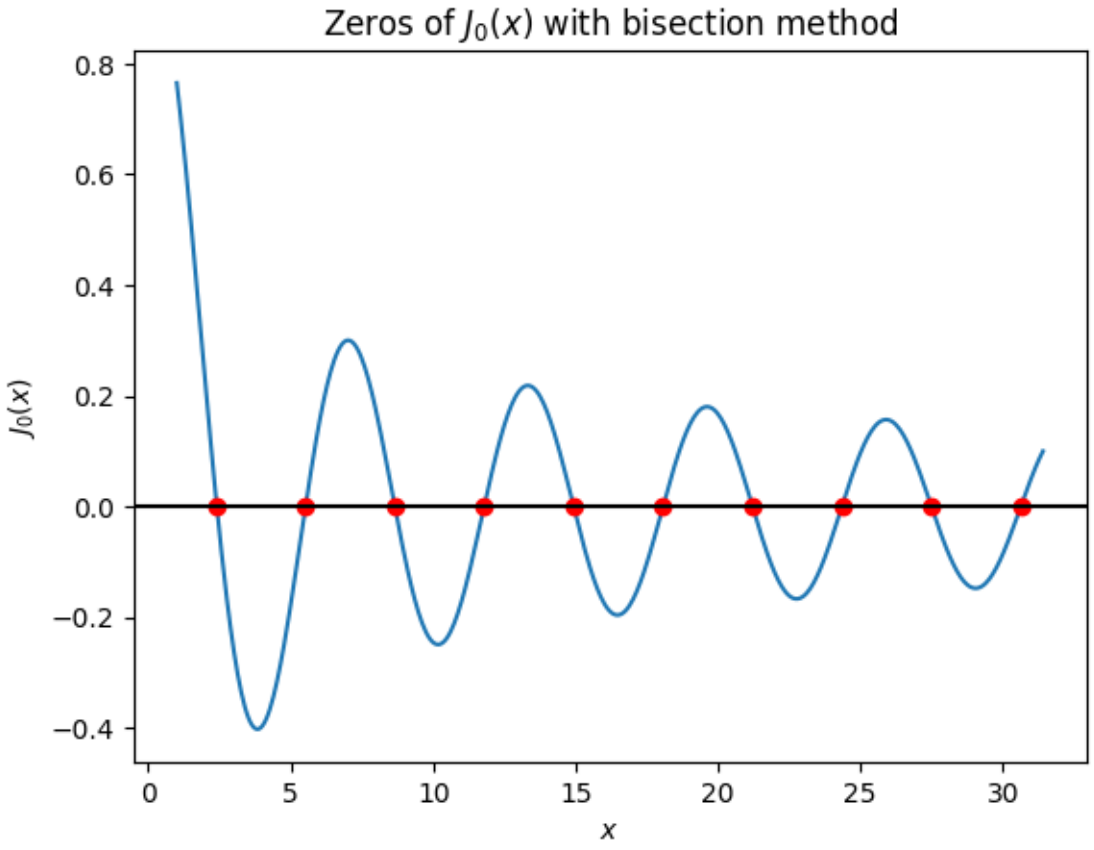


Fig. 6.8. Bisect Example 2

6.4.2. Newton's Method

Most numerical methods classes use this approach to illustrate the secant method, which is why we built it; nevertheless, it should be noted that according to the method, this is the only one that needs the input of the derivative.

Newton's method is an iterative method that uses the derivative of the function to approximate the zero of the function.

The iteration is given by the following formula:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Examples

Example 1

```
from BNumMet.NonLinear import newton
fun = lambda x: x**2 - 2
derivative = lambda x: 2 * x
interval = [1, 2]
sol, nIter = newton(fun, derivative, start_point=2, iters =
    True)
print("Newton's method: x = %f, nIter = %d" % (sol, nIter))
```

Example 2

```
from BNumMet.NonLinear import newton
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
derivative = lambda x: sp.jvp(0, x, 1)  # Derivative of the
    Bessel function
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
     for the n-th zero

zeros = [newton(f, derivative, start_point=interval(n)[0])
    for n in range(1, 11)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with Newton's method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

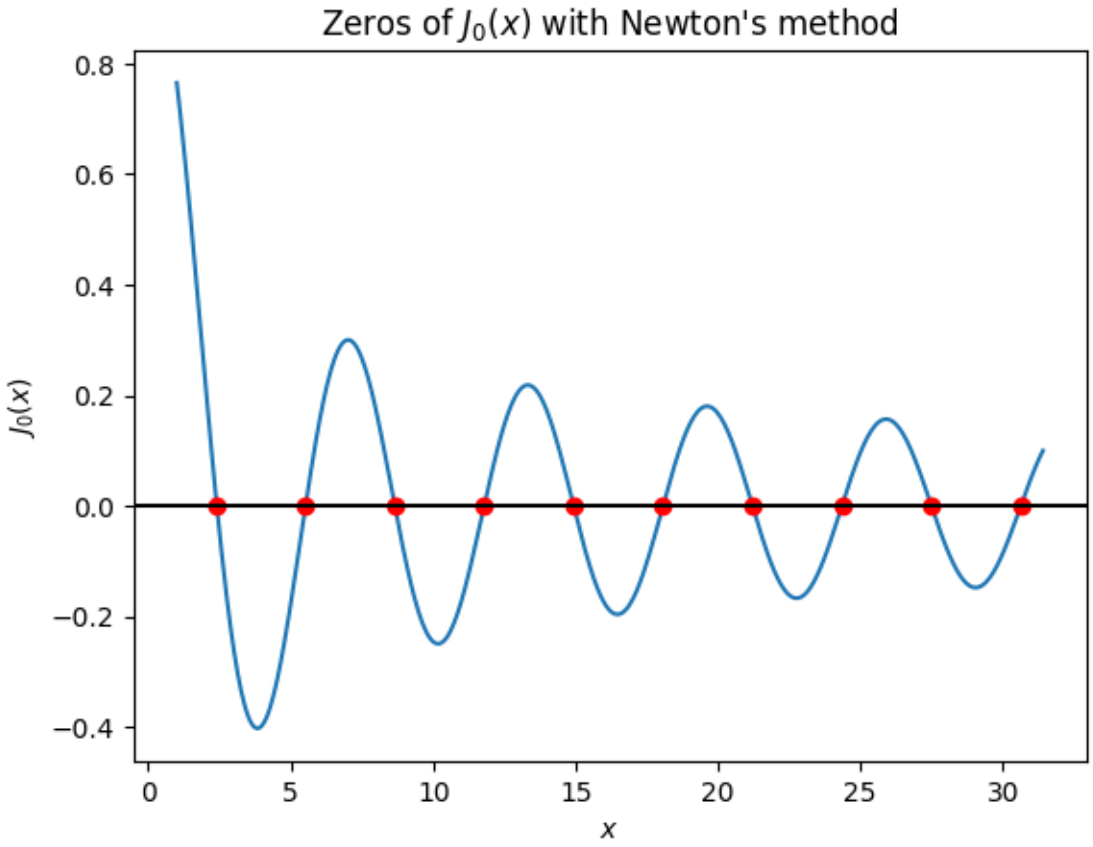


Fig. 6.9. Newton's Method Example 2

6.4.3. Secant Method

As a continuation of Newton's approach, it is critical to apply the Secant method, which generalizes Newton's method by approximating the derivative and eliminating the necessity for the derivative that Newton's method requires.

The algorithm is based upon C. Moller's implementation [3].

Examples

Example 1

```
from BNumMet.NonLinear import secant
fun = lambda x: x**2 - 2
interval = [1, 2]
sol, nIter = secant(fun, interval, iters = True)
print("Secant method: x = %f, nIter = %d" % (sol, nIter))

>> Secant method: x = 1.414214, nIter = 7
```

Example 2

```
from BNumMet.NonLinear import secant
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
     for the n-th zero

zeros = [secant(f, interval(n)) for n in range(0, 10)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with secant method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

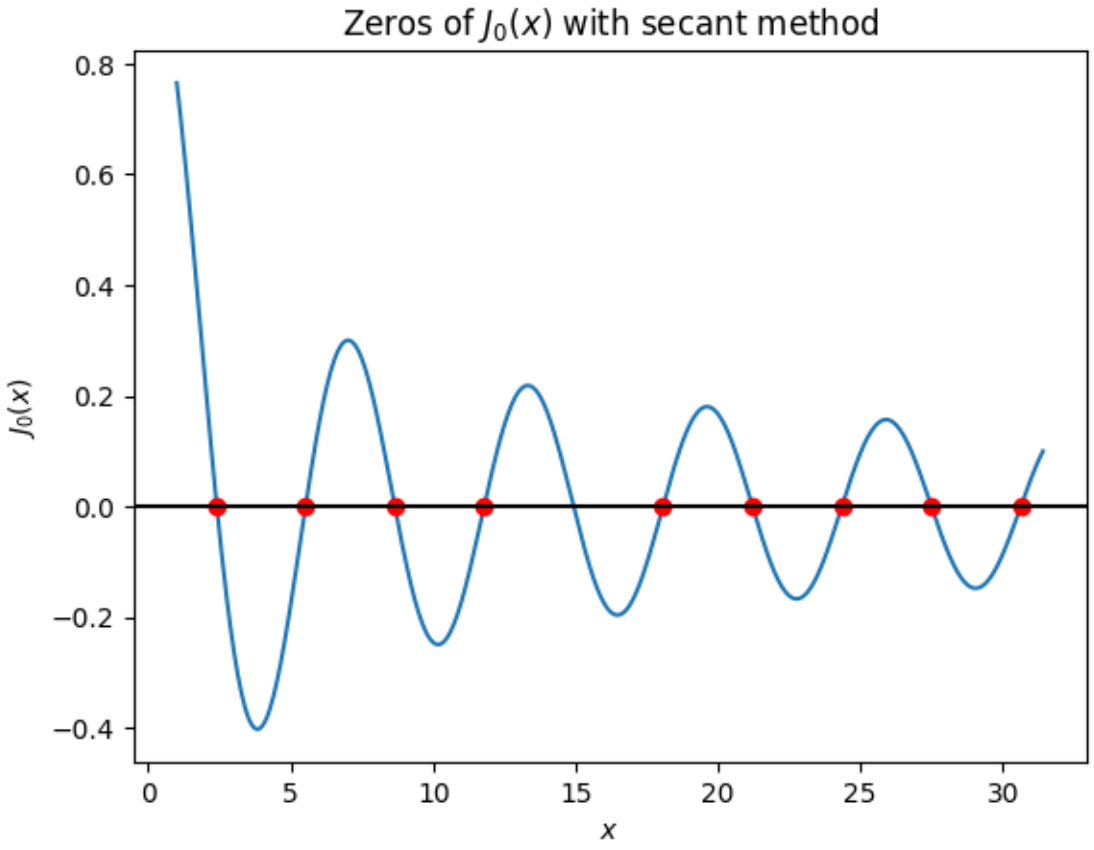


Fig. 6.10. Secant Example 2

6.4.4. Inverse Quadratic Interpolation (I.Q.I.)

The aim of this method is to extent the secant method to use a second order approximation. However, a polynomial of second degree does not always have a crossing with the x-axis. That is why the inverse function of a second order polynomial is used which always posseses such a crossing. It should be noted that this method is often used as part of another method that we will examine later [27], that is why we created it as a standalone to offer students with a glimpse of what the method is about without additional code that may interfere with the learning process. Additionally, this method is a natural extension of the secant method in which a second order polynomial is used instead of a linear one.

The algorithm is primarily based upon [28] as well as Moller's implementation [3].

Examples

Example 1

```
from BNumMet.NonLinear import IQI
fun = lambda x: x**2 - 2
points = [1, 1.5 ,2]
sol, nIter = IQI(fun, points, iters = True)
print("IQI method: x = %f, nIter = %d" % (sol, nIter))

>> IQI method: x = 1.414214, nIter = 6
```

Example 2

```
from BNumMet.NonLinear import IQI
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
interval = lambda n: [
    n * np.pi,
    (((n + 1) * np.pi) - n * np.pi) / 2,
    (n + 1) * np.pi,
]  # Interval for the n-th zero

zeros = [IQI(f, interval(n)) for n in range(0, 7)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with IQI method")
plt.xlabel("$x$")
```

```
19  plt.ylabel("$J_0(x)$")
20  plt.show()
```

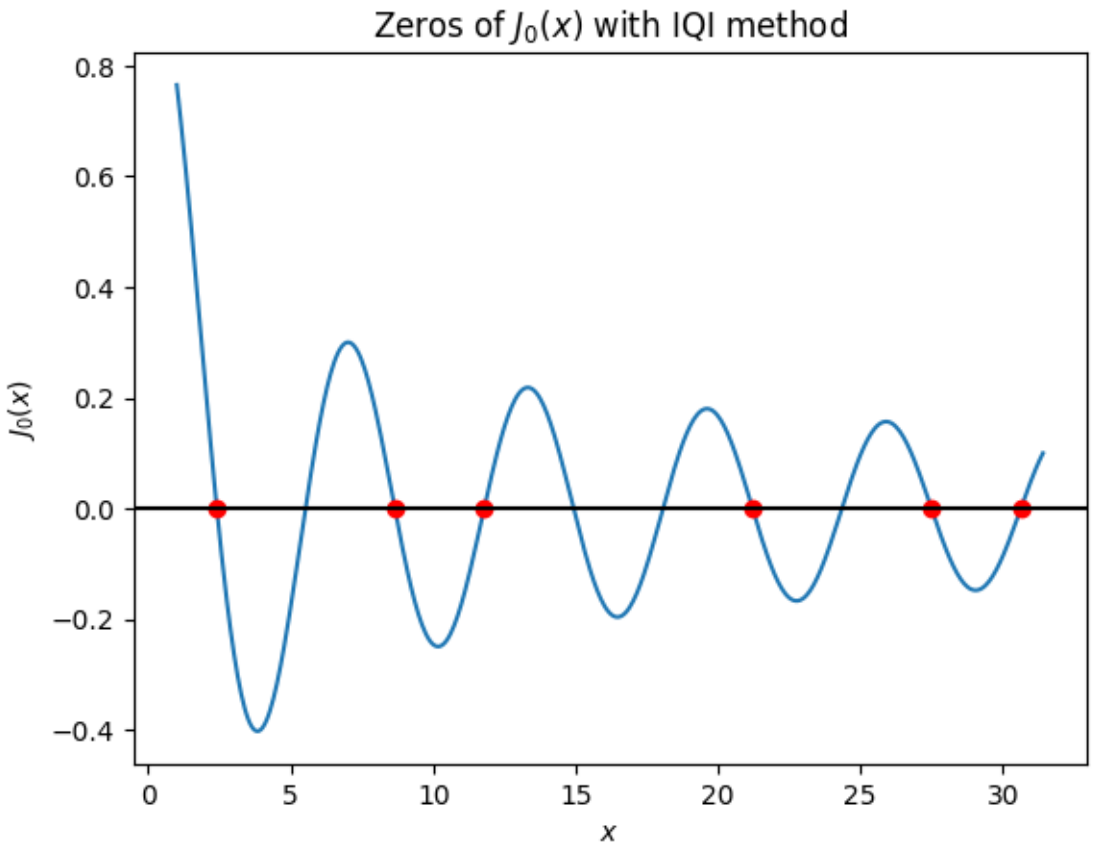


Fig. 6.11. IQI Example 2

6.4.5. Brent-Dekker's Algorithm

Brent-Dekker's algorithm is one of the methods that combines the previously discussed methods into a single one that will improve the number of evaluations a specific method may require while also eliminating some of the caveats that, for example, the secant method has, where there are functions where this latter method does not converge. It should be noted that this approach was first developed by Dekker, Wiengarten, and their colleagues, only to be modified by Brent to improve convergence [29].

The fundamental idea of Brent-Dekker's Algorithm is to apply the bisection, secant, and if possible the I.Q.I. method, the fundamental criteria of choice is firstly the availability of the application of I.Q.I. which requires three distinct points, after checking that, if it applies I.Q.I is saved, otherwise the secant is saved to then be compared with the bisection algorithm, whichever of the two passes a tolerance criteria given then they will be applied and the next iteration will start.

In BNumMet, we used Brent's original approach [30] which has a Fortran implementation of the algorithm. The use of this work rather than Matlab's implementation of this method named as 'fzero' was for a variety of reasons, one of them is the license under which Matlab allows the code to be used; since our goal was to follow open source initiatives, we needed to use the original one. We also wanted to show students the historical accuracy of the algorithm and present a method that preserves the historical context as well as its current properties.

In the next paragraph, we will examine the several implementations we discovered, including Matlab's implementation, the original implementation, and Scipy's implementation.

Analysis of implementations

The analysis of this package differs from the previous two (linear systems and interpolation packages), where we draw our attention to the syntax and it benefits, in this analysis we will focus on comparing the same method over different plausible implementations that might differ slightly on the code but have a humongous impact on the results of such slight algorithmic differences.

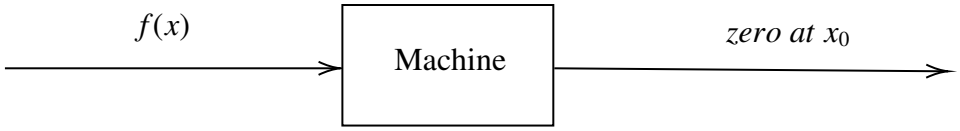
In particular, we will focus on the different implementations the Brentt-Dekker algorithm can have; we will tackle the following:

1. Scipy's Brentt Method: The commonly used library for scientific computing has its own implementation of Brent-Dekker's Algorithm. However, the code is not publicly visible. We will discuss how to obtain results without knowing how it works.
2. Matlab's Implementation: As we discussed, this project has the underlying idea of C. Moller's book [3], that is why we have made the translation of this algorithm into Python to properly test it, though it must be noted we will not be using it in our package as a standalone, since we do not own the right to do so.
3. Original Brent-Dekker's Algorithm: Following the original documentation of Brent [29] we have translated the procedure given in Chapter 4 of Brent's work into Python.

In this analysis we are interested in how many function evaluation does the algorithm need to find the root, the fundamental reason that we are interested in the number of evaluations is because this types of algorithms do not have an excessive algorithmic complexity, making the evaluation of the function that (most likely) will be nonlinear carry all the computational cost, that is why we need to find the number of evaluations,that is, we measure the computational

cost in the number of function evaluations. However, we are unable to look at Scipy's code to tamper with it to obtain our desired result, to properly proceed we want to digress from the discussion at hand and focus on how we can find the number of function evaluations without the need of having the actual implementation.

Finding function evaluations without external code As a thought experiment, imagine you have a machine that is protected so as not to be opened or tampered with, this machine can only take in a function $f(x)$ and outputs its zero (for arguments sake, suppose it can be done without any problems), this machine does not output how many times did it use the function to calculate or has a small light that shows how it is processing. We can only input functions and have as an output the location of the zero.



One could argue why not create a function that counts the evaluation every time it is invoked; something of this sort will reassemble:

```
FUNCTION
    IN : x
    PERFORM :
        evaluations +=1
    OUT : f(x)
```

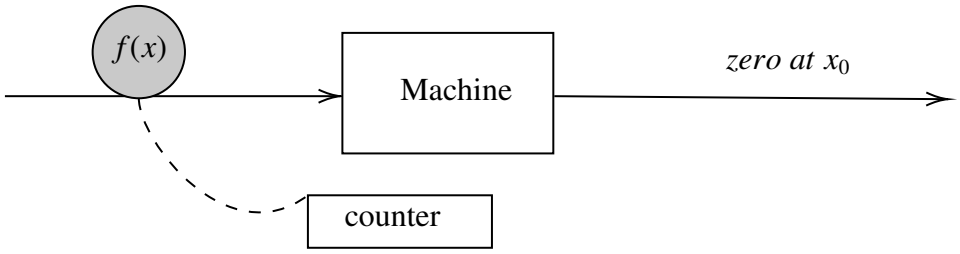
But most programming languages lack the ability to auto-initialise the value, and even if initialised it must be done inside the function itself which will be reset every function evaluation.

Imagine we can create a function (think of it as a piece of hardware) that can safely be input to our machine and still have our expected zero, but we can create one "invisible" cable that is connected to a light outside the machine, and every time it evaluates the function, the light is turned on (or in this case a counter is updated). To do such thing, we must draw our attention to Global Variables, variables that are within the reach of the entire execution of a program and can be edited and accessed anywhere in the code.

This idea will look like

```
global evaluations = 0
FUNCTION
    IN : x
    PERFORM :
        evaluations +=1
    OUT : f(x)
```

Once this link is created, we only need to reset the counter every time we input a new function, and we will obtain the evaluations they did of the function.



Results To test it we will implement the thought experiment in code and proceed in the following manner, we will create a function $f(x) = (x - a) \cdot x^i$, where a will be either 1 or 0.1, a number with exact floating point representation and a number without it, the i will be the exponent of the function which will take odd values. We will run the 3 aforementioned methods once (since in this case the algorithms are purely deterministic, as they do not rely on time) for different interval widths starting at 0.8 and ending in $1.1 + j$ where j will be increasing from 1 to 10000 in step sizes of 1. Using in all the methods the same input tolerance.

Our goal with this test will be firstly to prove the effectiveness of the original algorithm and the implementation in python with respect to the other two, as well as, observing if we have an algorithmic advantage or not with these aforementioned methods.

We will then take the number of evaluations and the value of x at where the zero is found in order to measure the relative error. The plots will cover the whole range of j but to properly show it, they will be taken at step sizes of 200 per bar.

For $a = 1$ As we can observe in the first plot of [Figure 6.12 NonLinear 3 method Results for $a = 1$ same tolerance] Scipy's implementation of Brent-Dekker's Algorithm is the worst in term of number of evaluations, and Matlab's has almost everywhere a similar behaviour in number of evaluations. Looking at the second plot, we see that even though BNumMet's implementation was better in terms of evaluations, it is the worst in terms of relative error, and Matlab's is similar to Scipy's but slightly worst.

One question rises from here: will BNumMets implementation improve, in terms of having a lower number of function evaluations, if we reduce the tolerance, at the risk of number of evaluations - but as seen, we still have room for tweaking. In fact, we can improve the relative error of BNumMets by decreasing the tolerance and still have an advantage over Scipy's implemen-

tation almost everywhere [Figure 6.13 NonLinear 3 method Results for $a = 1$ BNumMet smaller tolerance]

This results proves that the original implementation of Brent-Dekker is better than Scipy's, because at a lower number of function evaluations we achieve a better relative error, it is worth noting that this result will also prove that the method is working as intended.

For $a = 0.1$ The same discussion of $a = 1$ remains valid to the case $a = 0.1$, it must be noted that in this case the plots show a extravagant behaviour, with the number of evaluations, at certain points, the evaluations remain the same regardless of the interval size. On the first plot [Figure 6.14 NonLinear 3 method Results for $a = 0.1$ same tolerance] we observe that BNumMet takes the leading position while on the second plot, there remains a high relative error but closer to Matlab's on the overall scheme.

Incrementing the input tolerance to BNumMet's implementation, we observe [Figure 6.15 NonLinear 3 method Results for $a = 0.1$ BNumMet smaller tolerance] that not only does BNumMet's implementation offer a lower number of function evaluations but also it provides a better relative error in comparison to Scipy's implementation.

All in all, the decision to implement the original Brent-Dekker was a good choice; not only does it present the historical algorithm but also improves on the well-known library of Scipy.

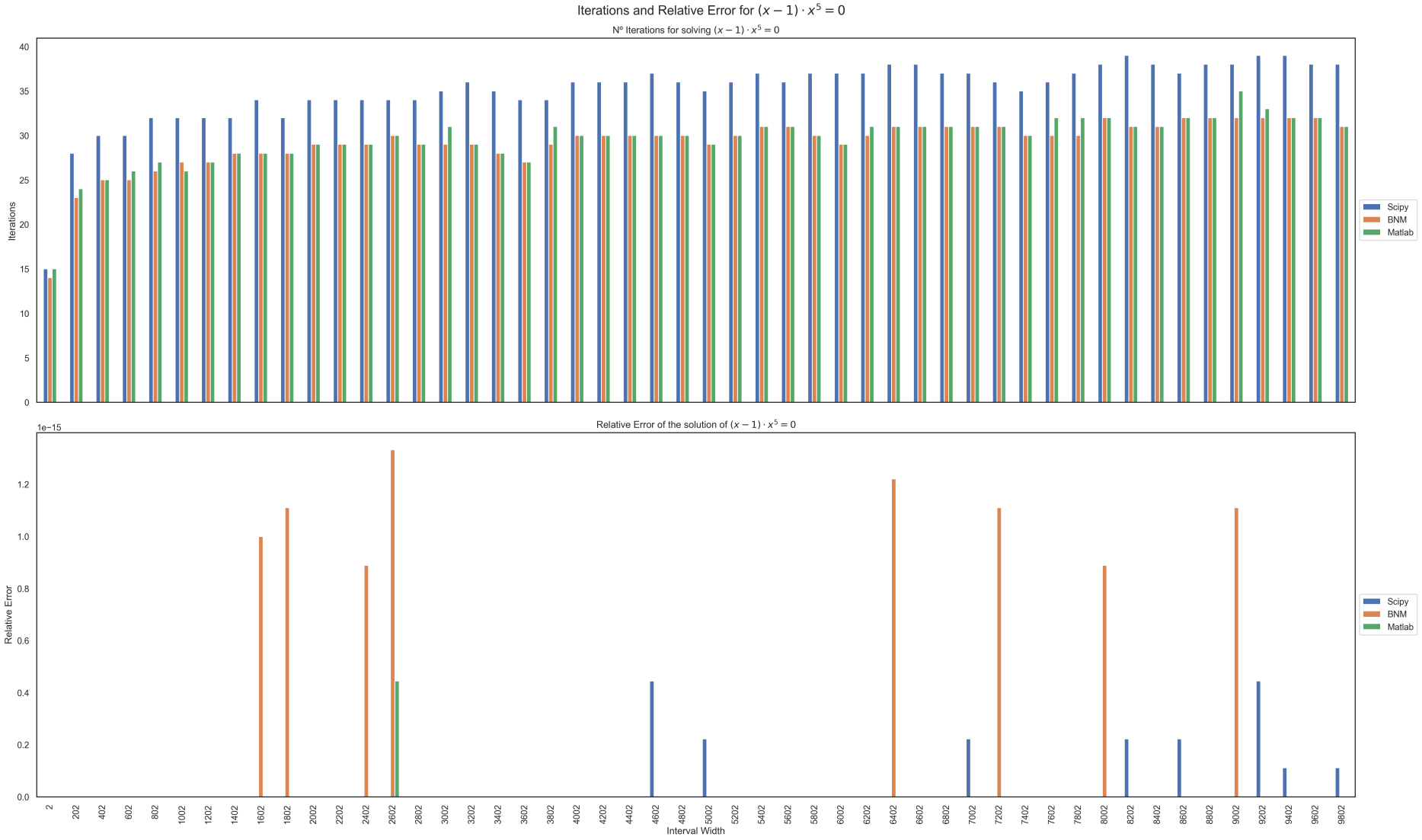


Fig. 6.12. NonLinear 3 method Results for $a = 1$ same tolerance

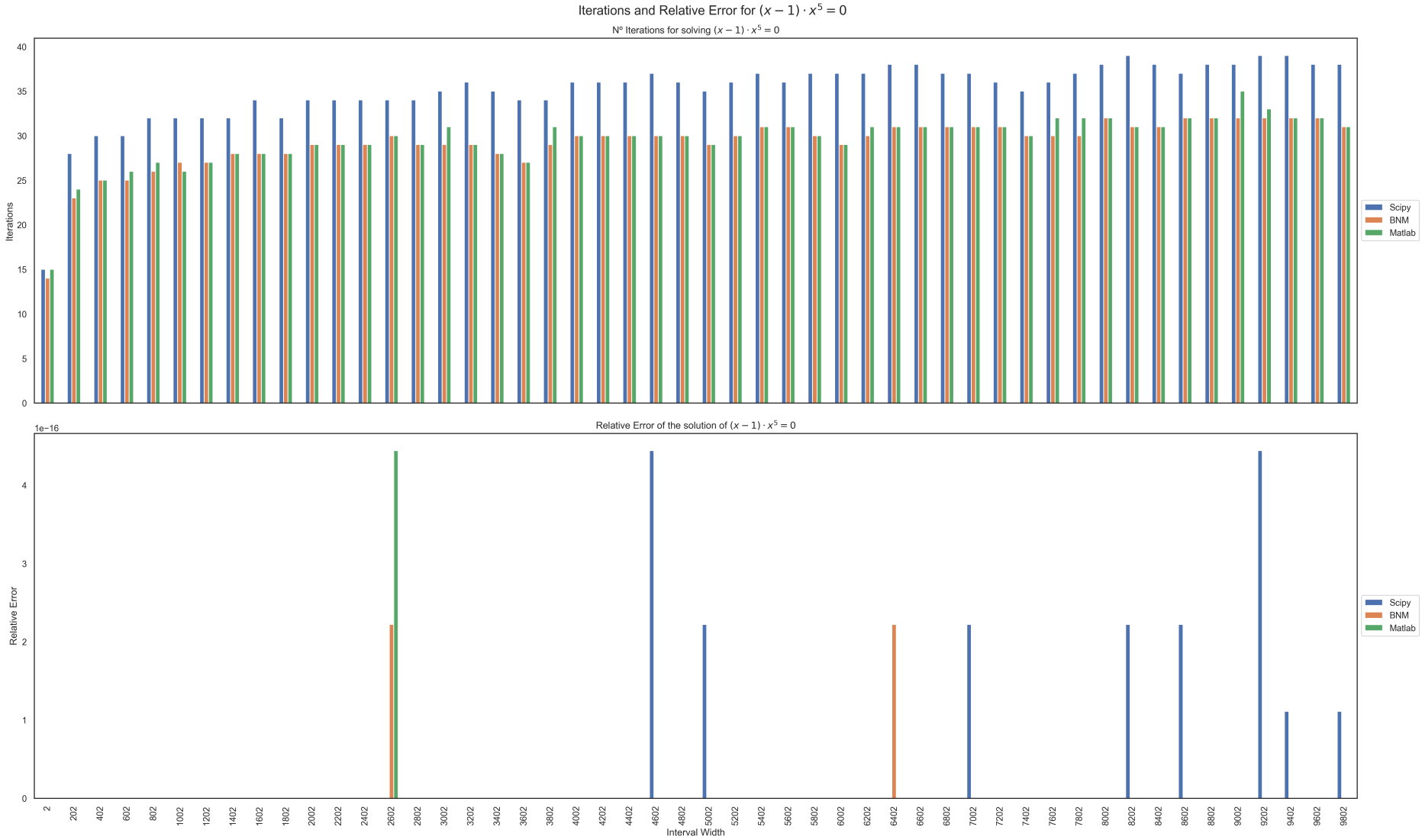


Fig. 6.13. NonLinear 3 method Results for $a = 1$ BNumMet smaller tolerance

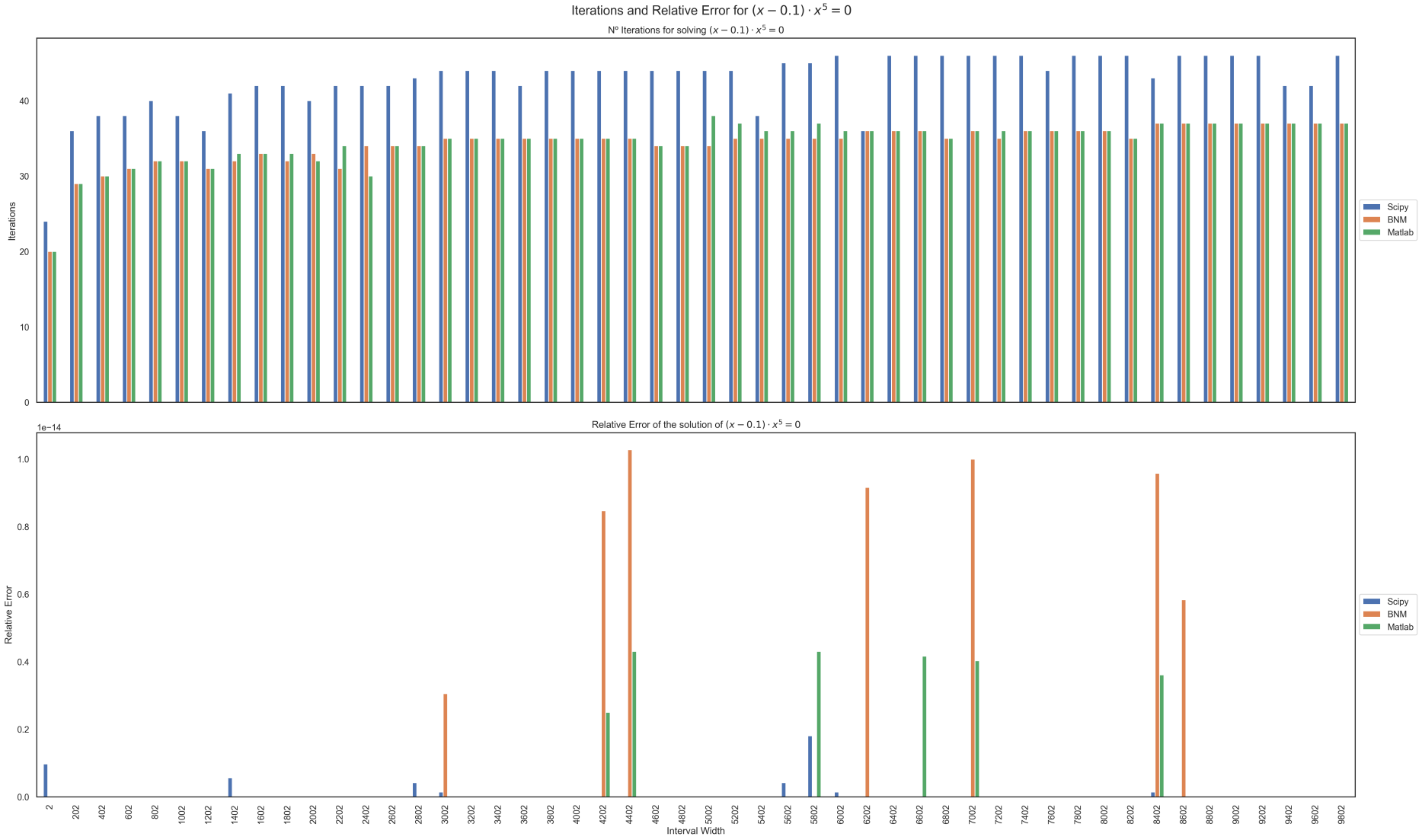


Fig. 6.14. NonLinear 3 method Results for $a = 0.1$ same tolerance

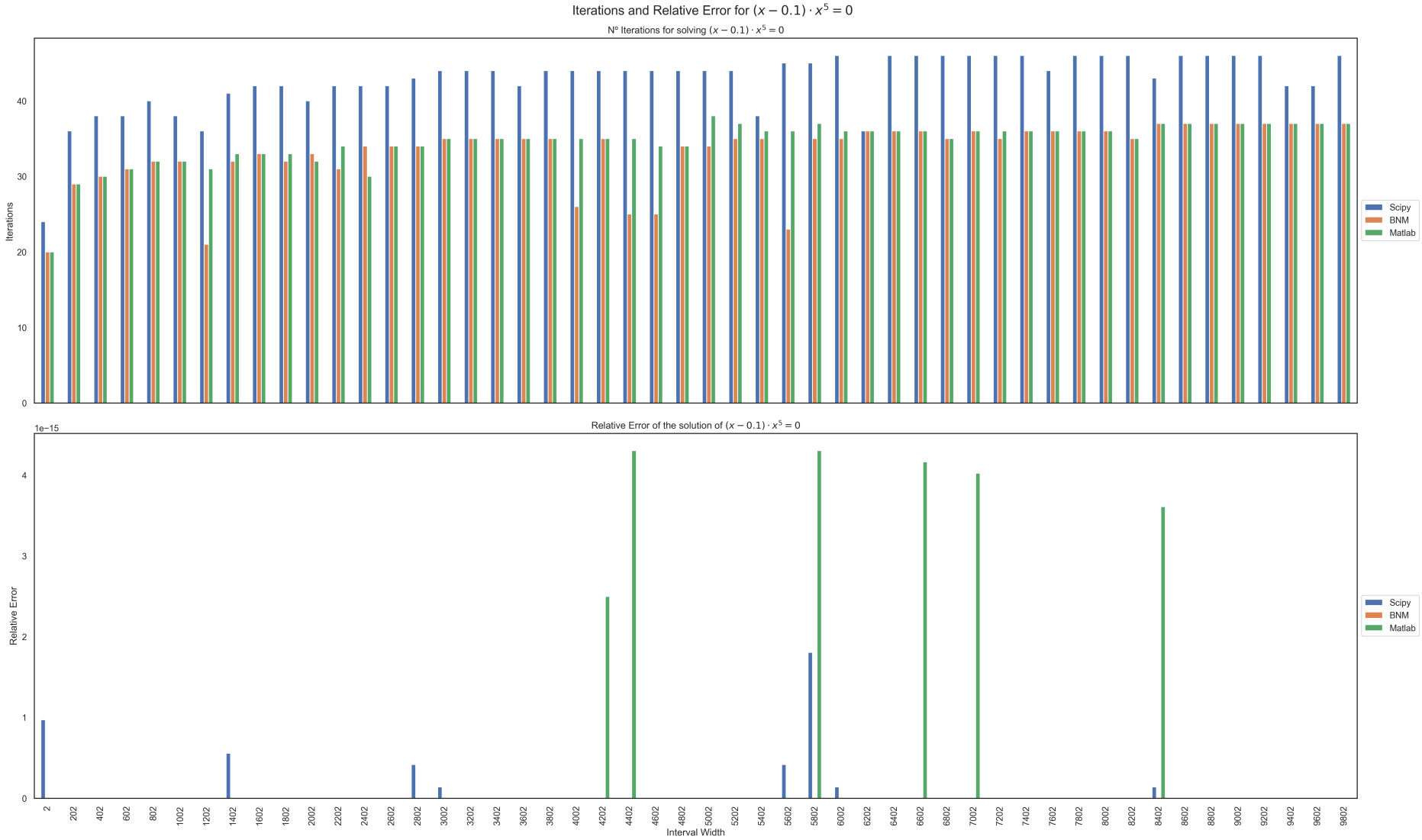


Fig. 6.15. NonLinear 3 method Results for $a = 0.1$ BNumMet smaller tolerance

Examples

Example 1

```
from BNumMet.NonLinear import zBrentDekker
fun = lambda x: x**2 - 2
interval = [1, 2]
sol, nIter = zBrentDekker(fun, interval, iters = True)
print("Brent-Dekker method: x = %f, nIter = %d" % (sol, nIter
    ))

>> Brent-Dekker method: x = 1.414214, nIter = 7
```

Example 2

```
from BNumMet.NonLinear import zBrentDekker
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
     for the n-th zero

zeros = [zBrentDekker(f, interval(n)) for n in range(0, 10)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with Brent-Dekker method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

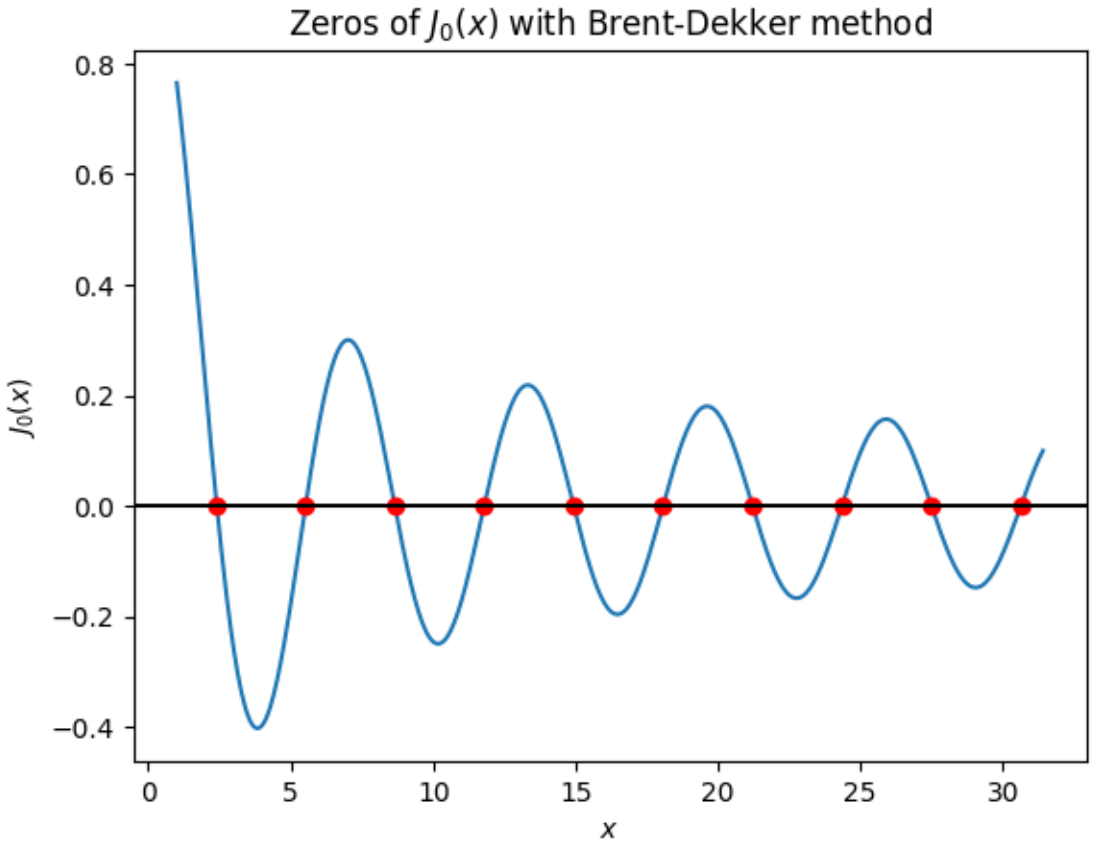


Fig. 6.16. Brentt-Dekker Example 2

6.5. Random Number Generator Package

To expand the techniques available in BNumMet even further, we will incorporate a lesser-known field of numerical methods, namely the implementation of generators for (pseudo) random numbers. The underlying motivation for its development is our interest about how random number generators function, as well as allowing the end-user to study a side of numerical techniques that is not generally taught. We should also mention that this package was not originally intended to be developed, but due to some extra time, we decided it would be beneficial to include it.

Many of the suggested techniques rely on specific values being initialized, so we used a Python dictionary to help students understand the meaning of each variable by associating a key with its value; this dictionary is then used as a global dictionary in the appropriate method.

In this section the following algorithms have been implemented by the author of this work:

- Lehmers Random Number Generator [Algorithm 17]
- Marsaglia Random Number Generator [Algorithm 18]

- Mersenne Twister Random Number Generator [Algorithm 19]

6.5.1. Lehmer's Generator

Lehmer's Random Number Generator is one of the simplest yet most elegant generators available. This type of generator is a Linear Congruential generator, which is defined by the following recurrence relation [31], [32].

$$x_{n+1} = k \cdot x_n \mod n$$

In the case of Lehmer's generator it follows:

$$x_{n+1} = (a \cdot x_n + c) \mod n$$

The selection of a, c, n and x_0 is critical, and some values have been proposed for optimal generation; one of these values has been used as the method's default initializer, in particular:

$$a = 7^5, c = 0, m = 2^{31} - 1, x_0 = 1$$

These values ensure that this generator works properly, but some bad generators can be found, which is an interesting exercise for the student to search for or come up with. Additionally it is worth mentioning that the developped function allows for input parameters to be set.

The algorithm implementation was based upon Moller's work [3].

Examples

Example 1

```
from BNumMet.Random import lehmers_rand
for i in range(10):
    print(lehmers_rand())

>> Lehmers Random Number Generator Initialized with default
   values
        a=16807
        c=0
        m=2147483647
        x=1.0
    7.826369259425611e-06
    0.13153778814316625
    0.7556053221950332
    0.4586501319234493
    0.5327672374121692
    0.21895918632809036
    0.04704461621448613
    0.678864716868319
    0.6792964058366122
    0.9346928959408276
```

Example 2: Predictability of Lehmer's Random Number Generator

```
from BNumMet.Random import lehmers_rand, clear_lehmers_vars
clear_lehmers_vars()
arr = [1]
for i in range(10):
    aux = lehmers_rand(a=2**16 + 3, m=2**31, c=0, x=arr[-1])
    if len(arr) >= 3:
        lehmerFormula = (6 * arr[-1] - 9 * arr[-2]) % 1  # Test Xn = (6Xn-1 - 9Xn-2)
        print(f"Lehmer's = {aux}\nPredicted = {lehmerFormula}\n")
    arr.append(aux)

>>  Lehmer's = 0.0008239871822297573
    Predicted = 0.0008239871822297573

    Lehmer's = 0.003295936156064272
    Predicted = 0.003295936156064272

    Lehmer's = 0.012359732296317816
    Predicted = 0.012359732296317816

    Lehmer's = 0.04449496837332845
    Predicted = 0.04449496837332845

    Lehmer's = 0.15573221957311034
    Predicted = 0.15573221957311034

    Lehmer's = 0.533938602078706
    Predicted = 0.533938602078706

    Lehmer's = 0.8020416363142431
    Predicted = 0.8020416363142431

    Lehmer's = 0.006802399177104235
    Predicted = 0.006802399177104235
```

Example 3: Visual failure

```
from BNumMet.Random import lehmers_rand, clear_lehmers_vars
clear_lehmers_vars()
fail2 = (
    lambda: float(
        (int(lehmers_rand(a=65539, c=0, m=2**31, x=123) * (2**31)) >> 23) & 0xFF
    )
    / 255
)
fail = [(fail2(), fail2()) for i in range(100000)]
plt.scatter(*zip(*fail), s=1, c="black")
```

```
11  plt.show()
```

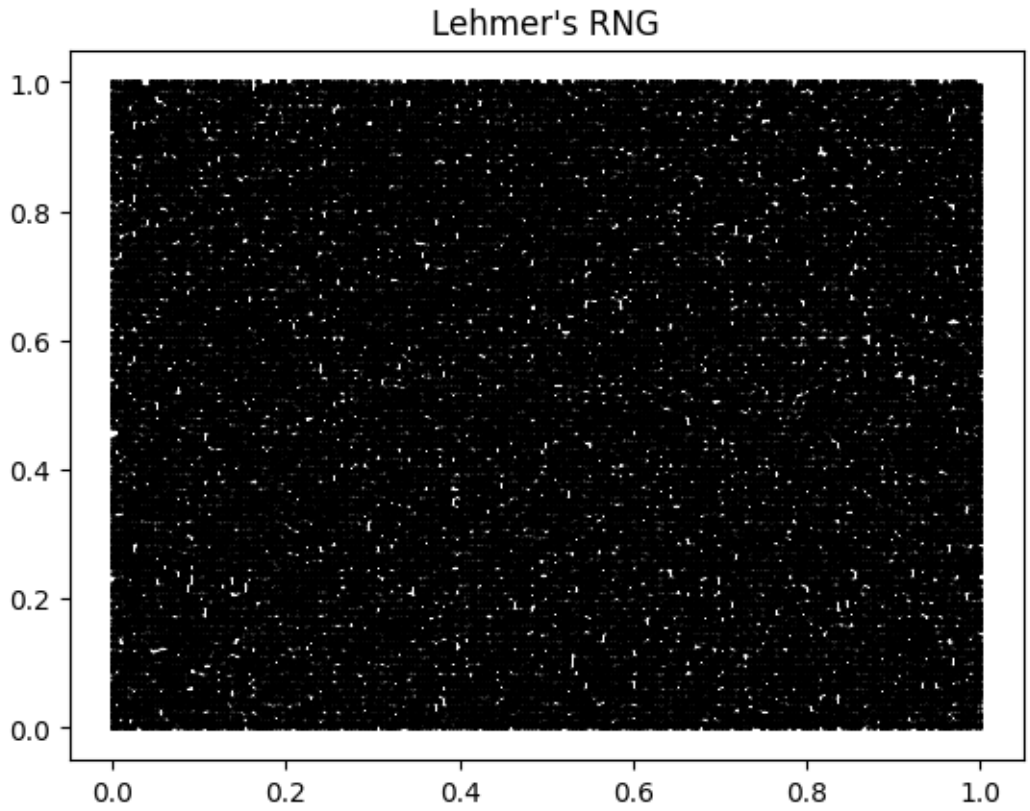


Fig. 6.17. Lehmers Rand Example 3

6.5.2. Marsaglia's Generator

In the original paper [33] Marsaglia expands on random number generators by providing a generator that is born from a type of relation such as Fibonacci, those generators that add or subtract previous values from the current value. But in Marsaglia's case this generator has some *lag* added to it, that is, it extends the appearance of the value that would be expected without *lag*. It is accomplished by introducing two parameters know as lag_s and lag_r as well as a *carry* variable which will be 1 if the generated number is negative - to which the *base* will be added - and 0 if the generator produced a positive number. This *carry* variable is what gives the name to this type of generator which are known as $add - with - carry$ or $subtract - with - borrow$

One of the complexities of this method is that it generates primarily integer numbers and is limited to the base it accepts as input; however, dividing by the base (the largest number in the generator) produces a decimal number between 0 and 1. Some input arguments were proposed, but as we saw in Lehmer's generator, some default parameters were provided.

$$base = 2^{31} - 1, lag_r = 19, lag_s = 7, carry = 1, seed\_tuple = (1, 1)$$

This set of values was proposed in the original paper; it is also worth noting that we used the subtract with borrow method, despite the fact that the addition with carry method is analogous. One of the aims of these implementation is that students find combinations of numbers that give rise to patterns and therefore that dont lead to appropiate random number generators.

We have implemented the algorithm using the original approach of Marsaglia [33].

Examples

Example 1

```
from BNumMet.Random import marsaglia_rand,
    clear_marsaglia_vars
clear_marsaglia_vars()
for i in range(10):
    print(marsaglia_rand(base=41, lag_r=2, lag_s=1, carry=0,
        seed_tuple=(0, 1)))

>>  0.975609756097561
    0.024390243902439025
    0.926829268292683
    0.0975609756097561
    0.8048780487804879
    0.2926829268292683
    0.4878048780487805
    0.8048780487804879
    0.6585365853658537
    0.12195121951219512
```

Example 2

```
from BNumMet.Random import marsaglia_rand,
    clear_marsaglia_vars
clear_marsaglia_vars()
fail = [
    (
        marsaglia_rand(base=100, lag_r=2, lag_s=1, carry=0,
            seed_tuple=(0, 1)),
        marsaglia_rand(base=100, lag_r=2, lag_s=1, carry=0,
            seed_tuple=(0, 1)),
    )
    for i in range(100000)
]
plt.scatter(*zip(*fail), s=1, c="black")
```

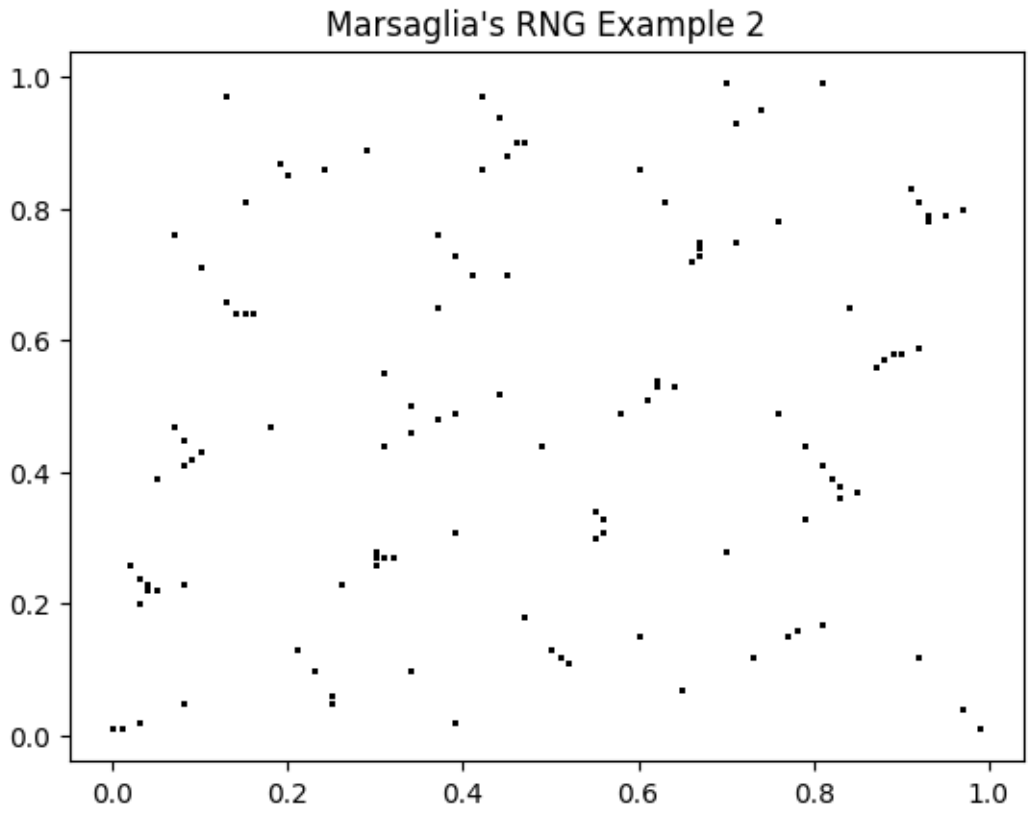


Fig. 6.18. Marsaglia Rand Example 2

Example 3

```
from BNumMet.Random import marsaglia_rand,
   clear_marsaglia_vars
clear_marsaglia_vars()
fail = [
    (
        marsaglia_rand(base=41, lag_r=2, lag_s=1, carry=0,
           seed_tuple=(0, 1)),
        marsaglia_rand(base=41, lag_r=2, lag_s=1, carry=0,
           seed_tuple=(0, 1)),
    )
    for i in range(100000)
]
plt.scatter(*zip(*fail), s=1, c="black")
```

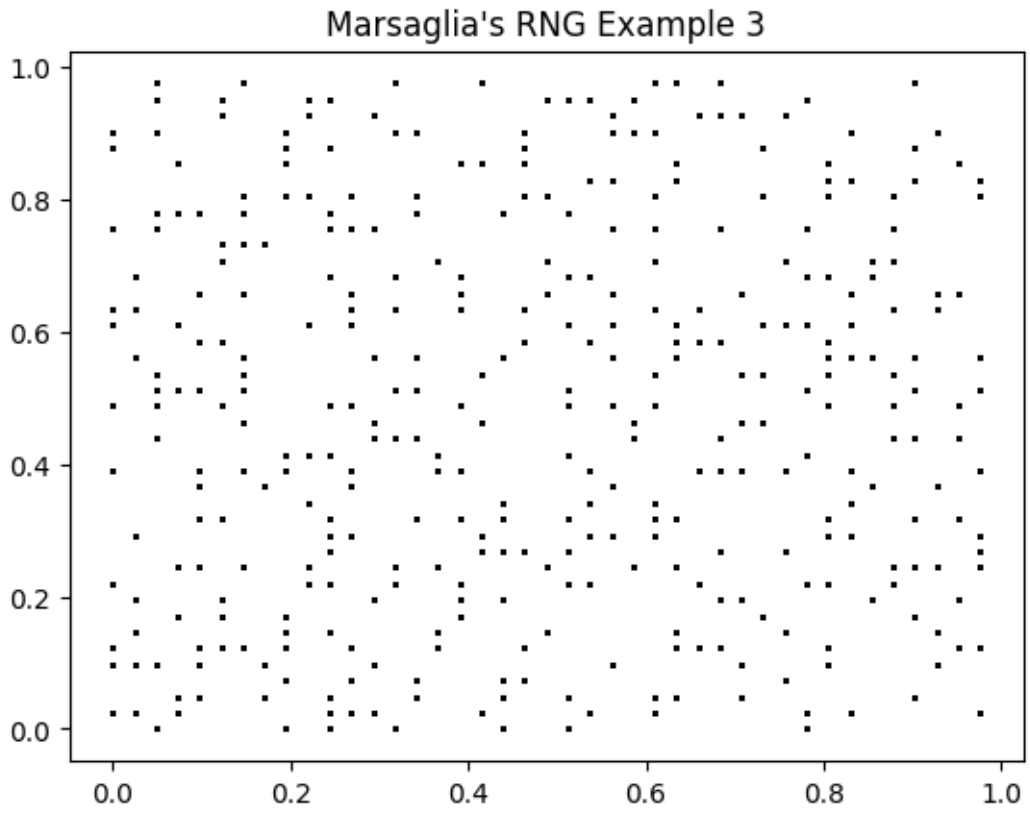


Fig. 6.19. Marsaglia Rand Example 3

6.5.3. Mersenne Twister Generator

The Mersenne Twister is an algorithm first introduced by Makoto Matsumoto and Takuji Nishimura [34]. According to the creators, the choice of parameters may produce a period (repetitions of random numbers) of $2^{19937} - 1$ while only requiring 624 words of memory and every word is made up of 32 bits - on the original implementation, newer ones exists with 64 bits. Most notably, it is the most often used generator, serving as the default generator for the Python and R programming languages.

We opted to implement the technique due to its popularity, even if the mathematics behind it are outside the scope of this project. We reckon the code is simple and does not require a lot of focus to comprehend what it is doing. Furthermore, we wanted to give students with commonly utilized approaches, and the Mersenne Twister is unquestionably one of them.

As Makoto and Takuji did, they ran a battery of tests to prove the generation of random numbers; similarly, we will assess the efficacy of this method.

The implementation of the method was based upon the original work [34] which contains a C implementation.

Analysis of solution

One question that arises in Random Number Generators is, how can we make sure our generator is actually random?, in this section we will dive into how we have tested our Mersenne Twister function.

The reference we have followed is [35], which develops a suite of tests to check different random number generators, and we have used an already developed version of this suite for Python [36], since it was out of the scope of this project to implement it.

Some discussion of the test we are eligible to test, since the test-suite has more tests than the ones presented, are the following.

1. **Monobit test**: This test purpose is to count how many zeros are there in the sequence, if it is close (up to some epsilon) to ½ then the tests will pass. This is a crucial test inside the suite since if this test fails, the rest will not pass.
2. **Frequency within a Block:** The test is analogous to the Monobit but instead of counting number of zeros in the entire string, it counts the frequency of 1 appearing in M-sized blocks, it should be close to M/2 plus or minus epsilon.
3. **Runs Tests:** This test counts the number of runs the sequence has. A run of length k consists of exactly k identical bits and is bounded by a bit of opposite value.

   (a) "01111110" : Run of length 6

   If the lengths are the ones one could expect from a random number, then the test will pass.
4. **Longer Runs Test**: In this case, the goal is to determine if the longest run length of ones is consistent with what one can expect from a random number. An irregularity in the length of ones, implies an irregularity in the length of zeros.
5. **Discrete Fourier Spectral Test**: It tries to detect periodic features of the run, it applies the Discrete Fourier Transform and count the peaks, the main goal is to find if the peaks that exceed the 95% threshold are less than 5%.
6. **Non-overlapping Template Matching:** The test tries to find generators that produce many occurrences of an aperiodic pattern, it searches on blocks of m bits for these types of patterns.
7. **Serial test**: This test focuses on determining the number of occurrences of the 2^m length of bit patterns and checking if they are one that is to be expected, since random sequences have uniformity, so each sequence

has the same probability of appearing. Were we to take $m = 1$ we would have the same results as in the monobit test.

8. **Approximate Entropy Test:** This is exactly the serial test, but compares with 2 consecutive blocks of lengths m and $m + 1$.
9. **Cumulative sums:** This test tries to find which is the maximal length of a random walk, assuming a $0 = -1$ and summing all the sequence, it should be close to zero.
10. **Random Excursion:** It checks that the number of cycles that have k visits in a cumulative sum walk is that of a properly random test.
    (a) A cycle consists of a sequence of steps taken at random that begin at and return to the origin.
11. **Random Excursions Variant Test:** Counts how many times a particular state is visited in a random cumulative sum.

Results To test it we will generate a total of 100 random integer numbers and run the battery of test through that sequence.

```
import numpy
from nistrng import *

clear_mt_vars()
# Test genrand from nistrng
sequence = numpy.array([genrand() * 0xFFFFFFFF for i in range
    (100)], dtype=numpy.uint64)


binary_sequence: numpy.ndarray = pack_sequence(sequence)

# Check the eligibility of the test and generate an eligible
    battery from the default NIST-sp800-22r1a battery
eligible_battery: dict = check_eligibility_all_battery(
    binary_sequence, SP800_22R1A_BATTERY
)
for i in eligible_battery:
    #print(i)
    ...

# Test the sequence on the eligible tests
results = run_all_battery(binary_sequence, eligible_battery,
    False)
# Print results one by one
for test,(res,_) in zip(eligible_battery,results):
    print(f"Test: {test} \n\tResult: {res.passed}")

>> Initialized the global dictionary mtVars with seed 4357
```

```
    Test: monobit
        Result: True
    Test: frequency_within_block
        Result: True
    Test: runs
        Result: True
    Test: longest_run_ones_in_a_block
        Result: True
    Test: dft
        Result: True
    Test: non_overlapping_template_matching
        Result: True
    Test: serial
        Result: True
    Test: approximate_entropy
        Result: True
    Test: cumulative sums
        Result: True
    Test: random_excursion
        Result: False
    Test: random_excursion_variant
        Result: True
```

The results are positive, except for one, on closer investigation it seems like this test suite has one flaw [https://github.com/InsaneMonster/NistRng/issues/9] documented by one of the users, after the change they suggested it works and outputs True. Therefore, a success in all test, we can assure our students that the Mersenne Twister is a true random number generator.

Examples

Example 1

```
from BNumMet.Random import clear_mt_vars,genrand
clear_mt_vars()
for i in range(10):
    print(genrand())

>>  Initialized the global dictionary mtVars with seed 4357
    0.8173300600185361
    0.9990608997175147
    0.5103543725587322
    0.13153290984489324
    0.03541634837990076
    0.9924695345089932
    0.6257087035630151
    0.06259194576707482
    0.4107105111262553
    0.13477367491805314
```

Example 2

```
from BNumMet.Random import clear_mt_vars,genrand
clear_mt_vars()
toPlot = [(genrand(),genrand()) for i in range(100000)]
plt.scatter(*zip(*toPlot), s=1, c="black")
```

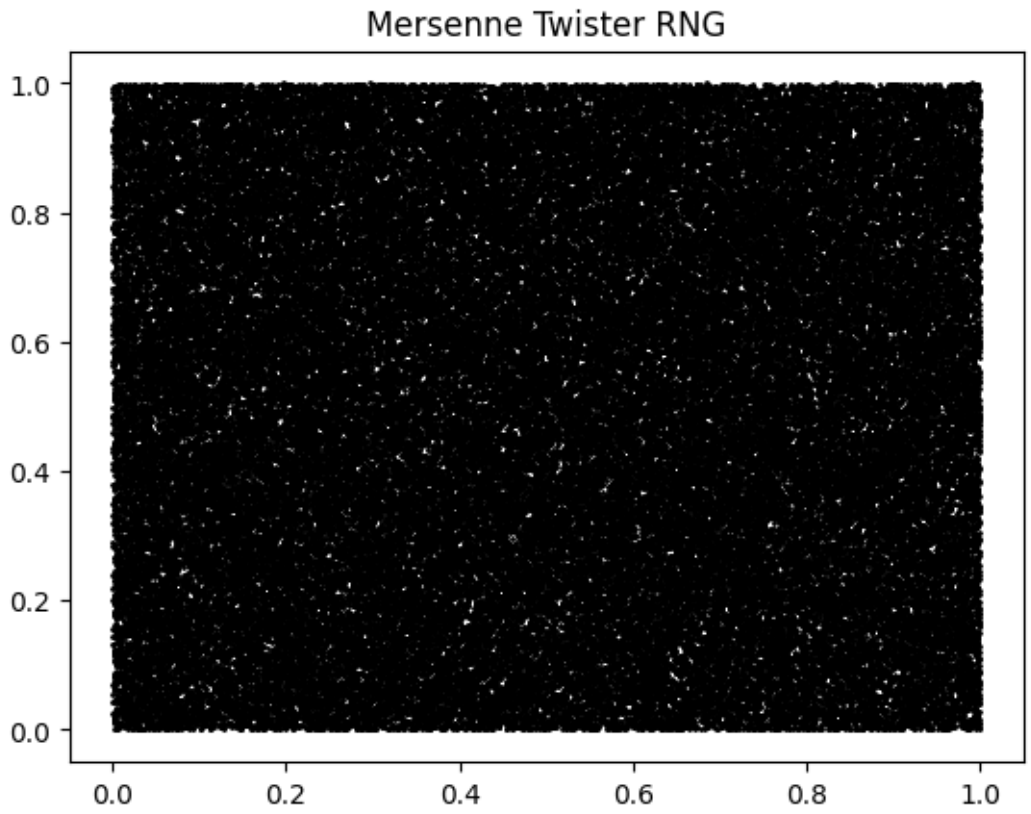


Fig. 6.20. Mersenne Twister Rand Example 2

6.6. Visualizer Packages

One of the most essential goals for BNumMet was to provide a graphical user interface that will allow students to view one or more parts of an underlying algorithm, with the goal of encouraging students to learn such algorithms.

In our quest to create this graphical user interface, we must keep in mind that, while the Python programming language excels in many areas such as data analysis, artificial intelligence, basic scripting, and so on, Python is not currently suitable - in general - for creating the graphical user interfaces that we are accustomed to seeing in our daily lives. It definitely includes modules for such jobs (PyQt, SimpleGUI,...), but Python was not designed with such in mind. To that purpose, as stated in J.C. Bucheli's work, we will be utilizing Jupyter Notebooks, an external Python Library that offers a more natural view of the code as if it were some class notes. We will leverage the

widgets provided by both iPyWidgets and BQPlot in this Jupyter notebook to enhance the capability of the notebook, in our instance, with interactivity. The rationale for adopting the same approach as J.C. Bucheli is that Jupyter is still extensively used in both the scientific and industrial communities for displaying code in an easy-to-read style and for correctly teaching coding skills to students.

Overall, the following packages will be an expansion to the approaches described previously in a format that will improve understanding without requiring an installation outside of what is generally installed and in a format that students will be accustomed to in their professional and academic life.

It is worth mentioning that most of the implementations had the core principles of what Mathworks sought to represent, but we have tried to focus on resolving some of the flaws that may prevent students from seeing the functionality effectively. Furthermore, all of the techniques supplied present default situations in case students just wish to check out the visualizer without thinking about examples.

6.6.1. LU Visualizer

As both Mathworks and J.C. Bucheli's work had implemented, we will strive to improve on the current visualizers made; however, before we start, we must discuss what is the underlying idea and some of the shortcomings that each had before to adopting this new visualizer.

The LU Visualizer's purpose will be to implement a graphical user interface that presents the main steps of the LU decomposition, which are pivot selection and reduction; additionally, at the end of the decomposition, the matrices that represent the permutation and the lower and upper triangular parts must be shown.

On the one hand, Mathworks performed an excellent job of creating the visualizer, allowing users to pick the pivot for each row and then do computations, as well as displaying an automated LU decomposition with three major methods: diagonal, partial, and complete pivoting. It's worth noting that this visualizer uses the standard LU Decomposition we have also implemented as a method without the rank revealing mechanism. Furthermore, it does not specify which pivots are accessible at each step [Figure 6.21 Mathwork's [3] Example 1 - No help in pivots to choose], not to mention that this work permitted the LU method to use zero as a pivot element, resulting in NaN and Inf values, which are not appropriate for a student to understand [Figure 6.21 Mathwork's [3] Example 1 - No help in pivots to choose]. It also does not display the permutation matrix and does not adequately identify which matrix is the L or U matrix, which is inconvenient for students.

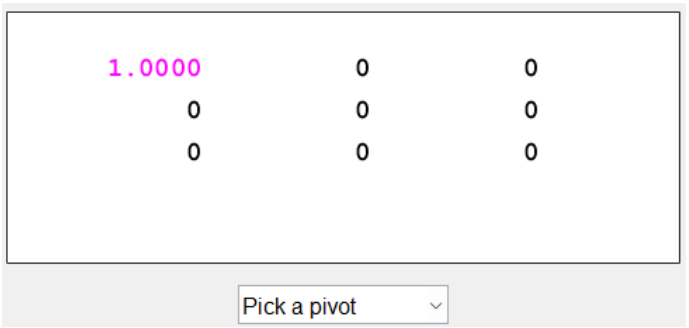


Fig. 6.21. Mathwork's [3] Example 1 - No help in pivots to choose

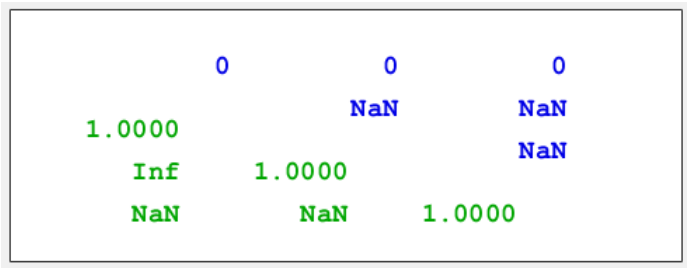


Fig. 6.22. Mathwork's [3] Example 1 - Nan and inf values

On the other side, while several of the issues raised in the Mathworks case were addressed in J.C. Bucheli's work, such as a visual assistance on which pivots may be clicked on, we were unable to use this visualizer, which appears to be malfunctioning with no error signals given. On that point, a deeper examination of J.C. Bucheli's code revealed the usage of threading techniques, which may be improper to maintain as well as inadequate for this sort of algorithm, because the LU algorithm does not require an external process to perform the modifications at each step. Furthermore, a timer was provided for the user to pick, which is contrary to acceptable standards in the development of a user G.U.I.

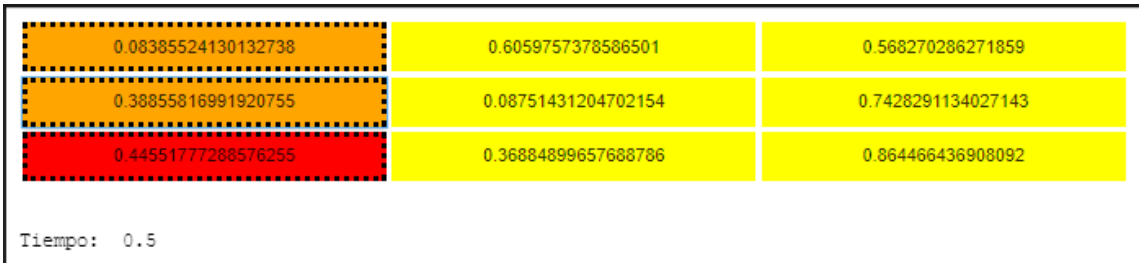


Fig. 6.23. J.C. Bucheli's [5] LU Visualizer Example 1

Once we know what are the main critiques of previously done work we can proceed by commenting on the proposed solution.

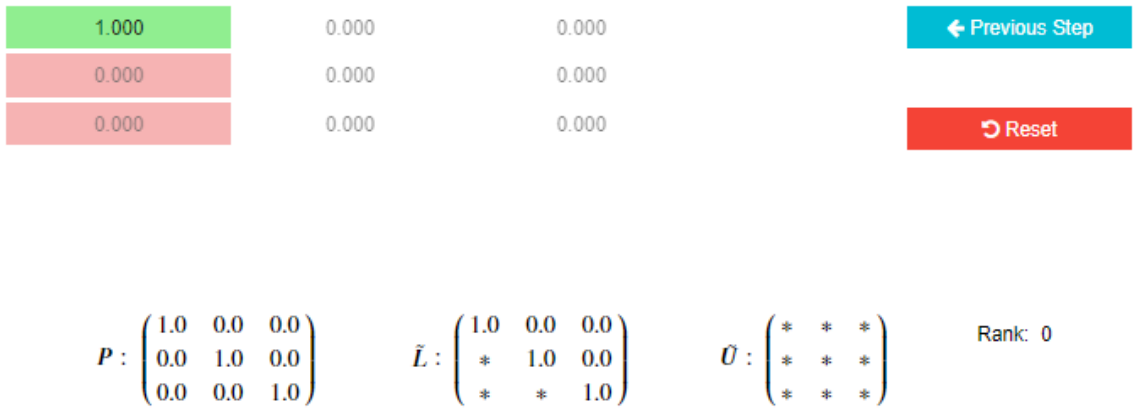


Fig. 6.24. BNumMet Example 1

As we can see, unlike Mathworks, the end-user knows which elements can be selected to pivot, and it prevents the user from dividing by zero, as well as the current output of the P,L,U matrices at each step (note the asterisk to indicate that we do not know the value at that step and it remains to be calculated). We additionally highlight two buttons, "Previous Step" and "Reset," because a user may miss click a button and not reach their intended aim, and we want pupils to feel comfortable when using our interactive visualizer. Furthermore, we indicate the rank we can guarantee the matrix has at each iteration by implementing the rank revealing mechanism, and the final result will be indicated by a system message [Figure 6.25 BNumMet LU Visualizer Example End Result]. Note that if the LU finds a column of possible pivots to be zero during the iterations, it will send a message as a system message explaining to the user that the algorithm has jumped the column until it finds a suitable one to pivot.

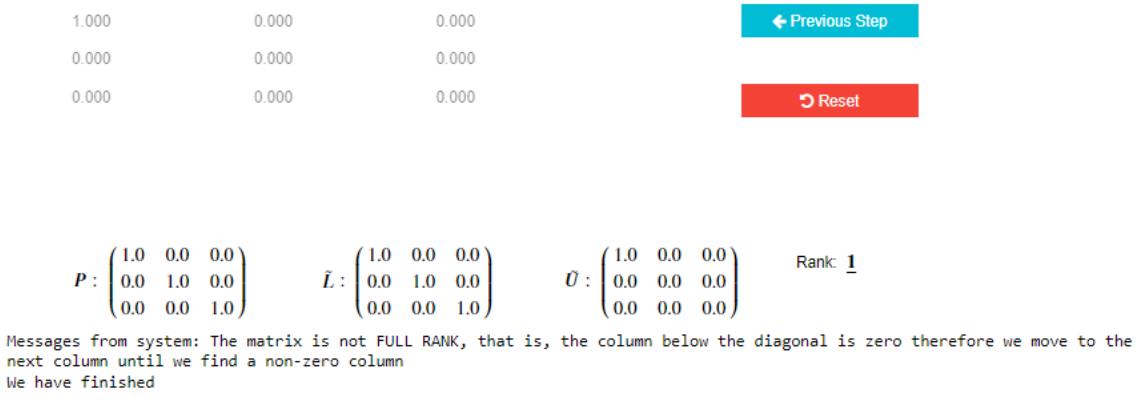


Fig. 6.25. BNumMet LU Visualizer Example End Result

Overall, the proposed solution corrects the previously done work while also improving it by providing the properties of the LU algorithm at a glance (Rank revealing, matrices at each iteration) and by improving the majority of the visual aspect of the overall implementation.

Examples

Example 1: No arguments

```
from BNumMet.Visualizers.LUVisualizer import LUVisualizer
luVisualizer = LUVisualizer()
display(luVisualizer.run())
```

1. Initial State:

1.000	2.000	3.000
4.000	5.000	6.000
7.000	8.000	9.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ * & 1.0 & 0.0 \\ * & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$$

Rank: 0

2. Click on first column, second row:

4.000	5.000	6.000
0.000	0.750	1.500
0.000	-0.750	-1.500

Previous Step

Reset

$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.25 & 1.0 & 0.0 \\ 1.75 & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & * & * \\ 0.0 & * & * \end{pmatrix}$$

Rank: 1

3. Click on second column, third row:

4.000	5.000	6.000
0.000	-0.750	-1.500
0.000	0.000	0.000

Previous Step

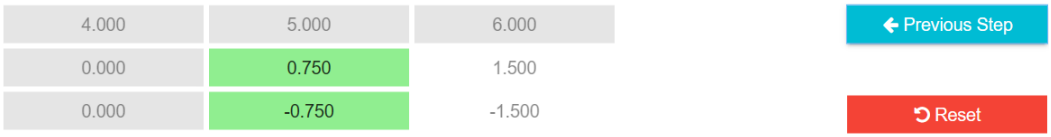
Reset

$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 1.75 & 1.0 & 0.0 \\ 0.25 & -1.0 & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & -0.75 & -1.5 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Rank: 2

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
We have finished
```

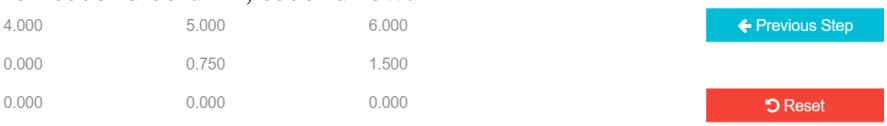
4. Click Previous Step:



$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.25 & 1.0 & 0.0 \\ 1.75 & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & * & * \\ 0.0 & * & * \end{pmatrix}$$

Rank: 1

5. Click on second column, second row:

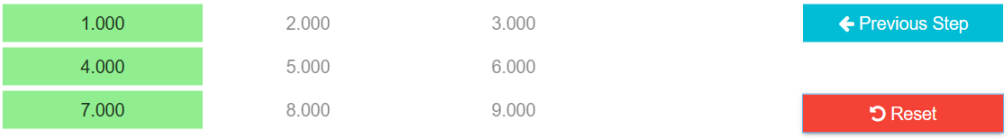


$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.25 & 1.0 & 0.0 \\ 1.75 & -1.0 & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & 0.75 & 1.5 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Rank: 2

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
We have finished
```

6. Click on Reset



$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ * & 1.0 & 0.0 \\ * & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$$

Rank: 0

Example 2: Arguments and Rank Revealing

```
from BNumMet.Visualizers.LUVisualizer import LUVisualizer
A = np.array([[1,2,3,7], [1,2,3,7], [1,2,3,7],[1,2,4,7]],
    dtype=float)
luVisualizer = LUVisualizer(A)
display(luVisualizer.run())
```

1. Initial State:

1.000	2.000	3.000	7.000
1.000	2.000	3.000	7.000
1.000	2.000	3.000	7.000
1.000	2.000	4.000	7.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ * & 1.0 & 0.0 & 0.0 \\ * & * & 1.0 & 0.0 \\ * & * & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

Rank: 0

2. Click on the first column, second row:

1.000	2.000	3.000	7.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	1.000	0.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 1.0 & 2.0 & 3.0 & 7.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & * & * \\ 0.0 & 0.0 & * & * \end{pmatrix}$$

Rank: 1

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
```

3. Click on the third column, fourth row:

1.000	2.000	3.000	7.000
0.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 1.0 & 2.0 & 3.0 & 7.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Rank: 2

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
We have finished
```

6.6.2. Interpolation Visualizer

As with the LU Visualizer, we will begin by discussing previous works and potential areas for improvement.

The interpolation visualizer's goal is to demonstrate the viewer how different interpolation methods compare to one another, as well as how moving the dataset or adding points might affect the initial interpolation given.

To begin, we will highlight Mathwork's implementation, which allows the user to select different interpolation methods while also allowing the user to edit the points and observe the changes. Despite the excellent implementation, it lacks the ability to add more points at runtime and does not allow viewing the entire interpolation when the function exceeds the limits set by the initial plot [Figure 6.27 Mathwork's [3] Interpolation Example - Drawbacks], it also has some bad contrast between the text and the background (on the text "poly"), and this does not obey the Human-Computer Interaction good practices. We must also emphasize the reset button, which may be useful to students if they accidentally change a point.

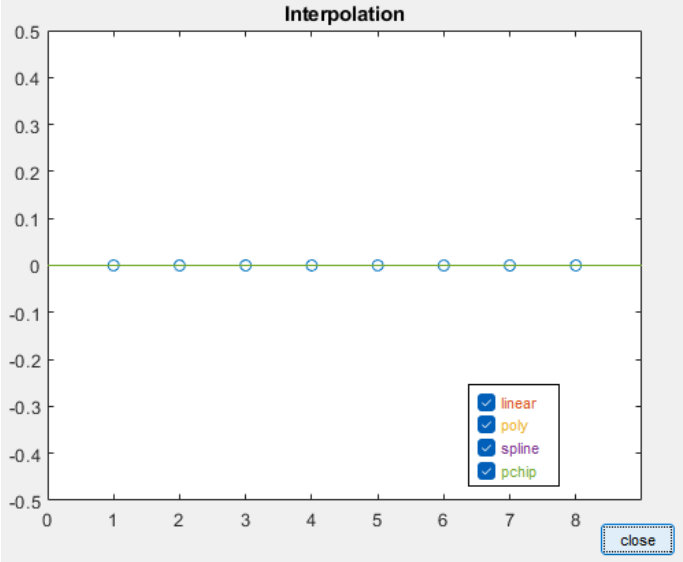


Fig. 6.26. Mathwork's [3] Interpolation Example

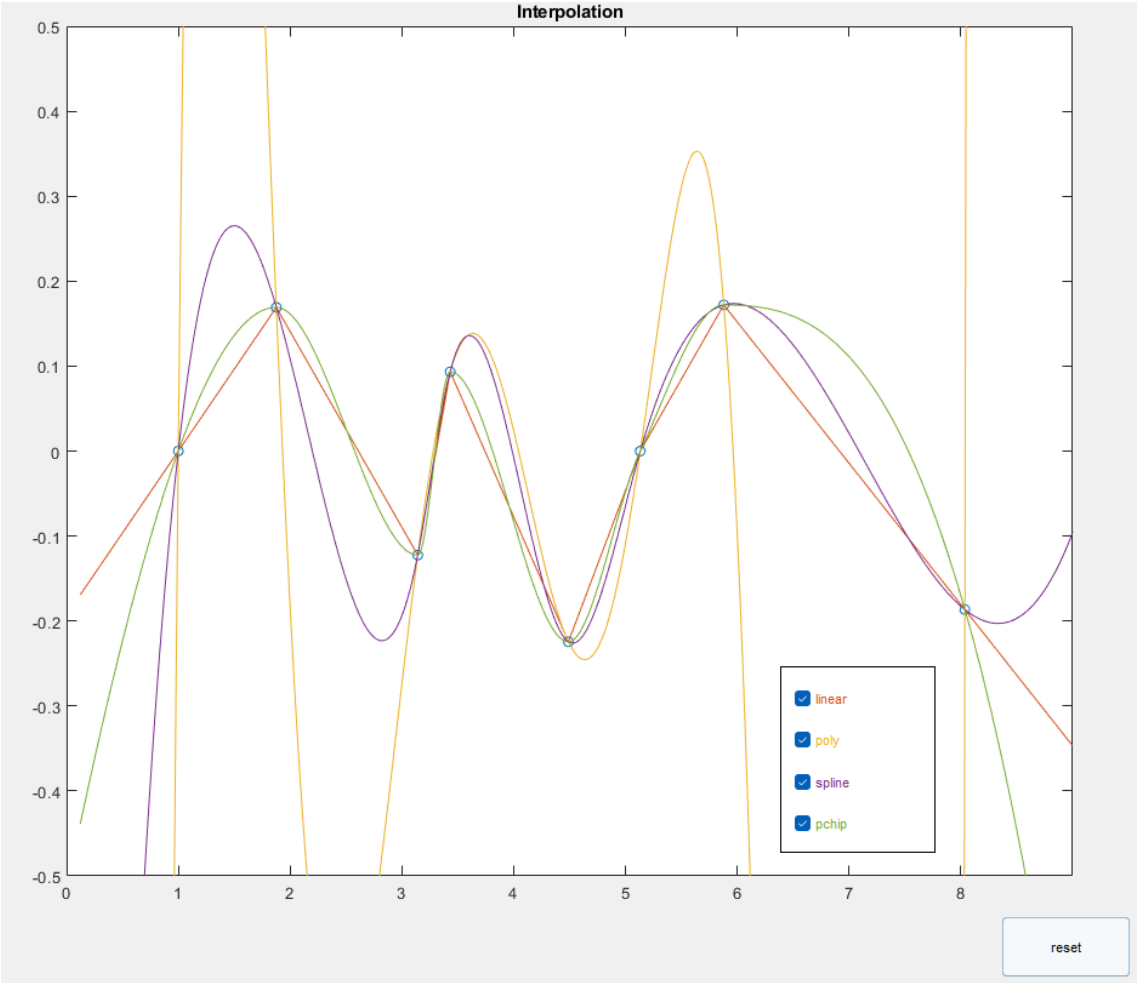


Fig. 6.27. Mathwork's [3] Interpolation Example - Drawbacks

Secondly, concentrating on J.C. Bucheli's effort, we must emphasize that the implementation was not completed, thus any comments will be based on what seems to be implemented. To begin, we should observe that the visualization does not support various interpolation methods; nevertheless, unlike Mathworks, this solution allows for plot movement but not single point movement or addition of points, as we can observe [Figure **?? ??**] moving the points will not update the plot.

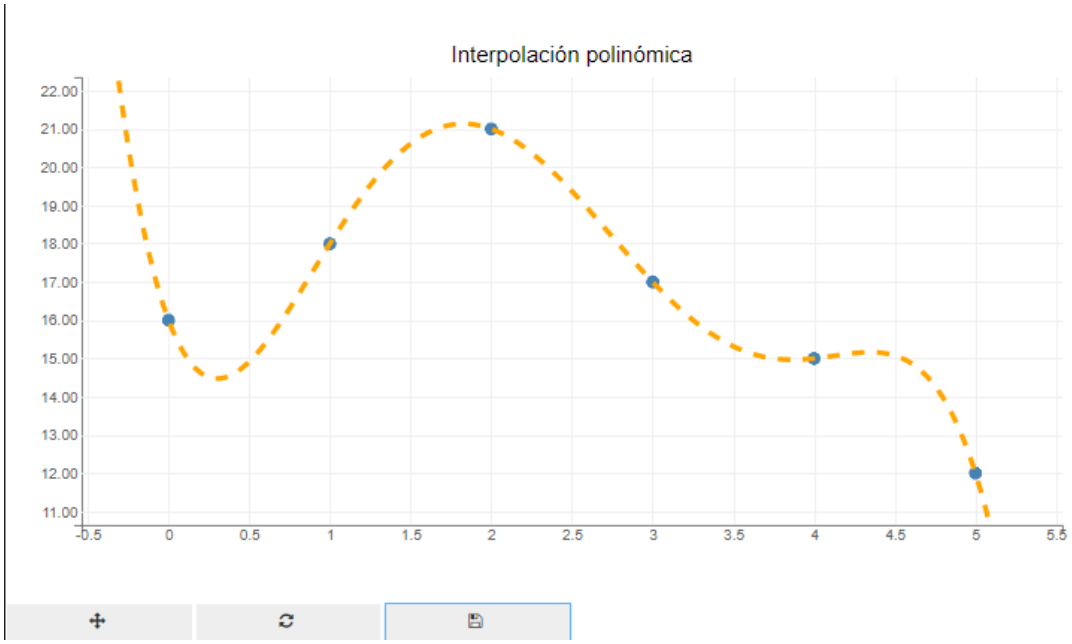


Fig. 6.28. J.C. Bucheli's [5] Interpolation Visualizer Example

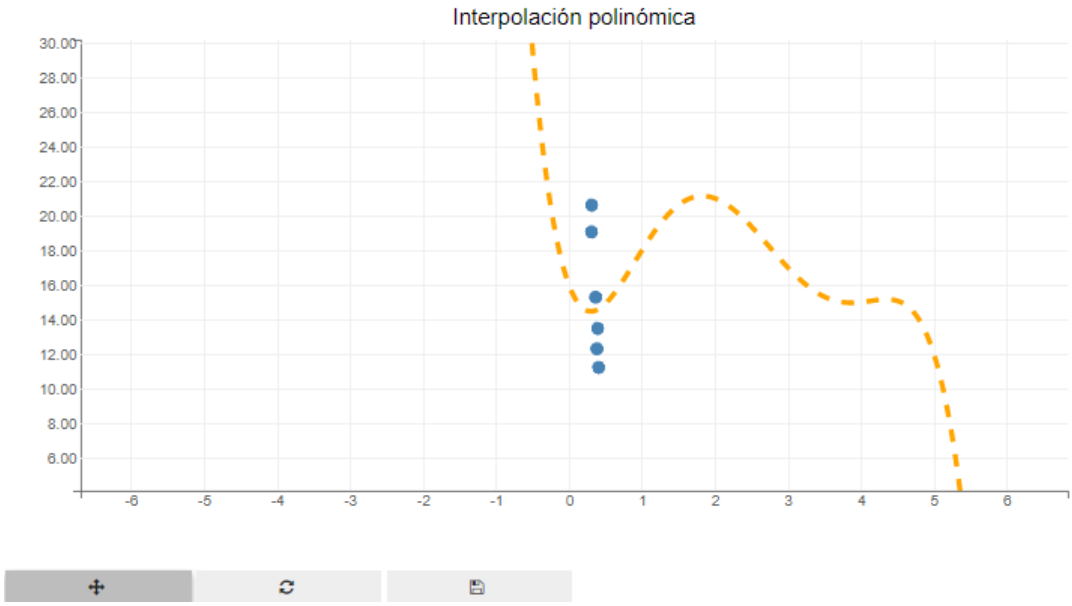


Fig. 6.29. J.C. Bucheli's [5] Interpolation Visualizer Example - Drawbacks

After understanding the important aspects and critiques of the preceding discussion, we create our own implementation, keeping in mind that numerous interpolating methods should be accessible to pick from, it should be able to move and add points, and it should include a button to undo all changes done.

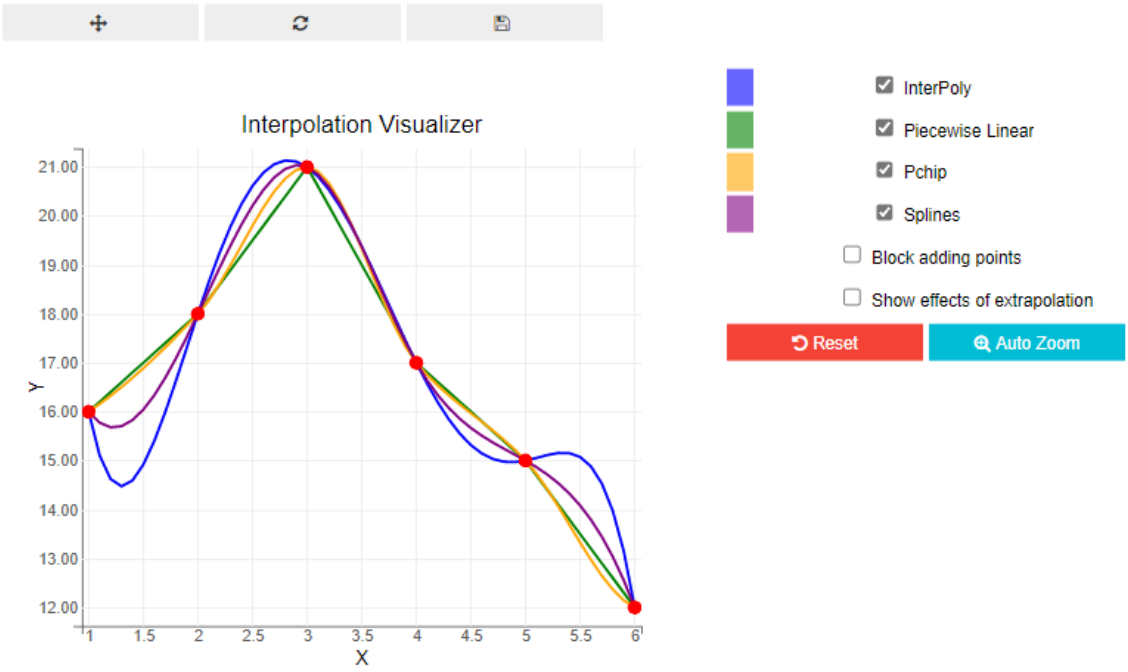


Fig. 6.30. BNumMet Interpolation Visualizer Example

As we can see, it has a similar appearance to Mathwork's implementation, but it adds an auto-zoom button in the case of out-of-bounds graphs [Figure 6.31 BNumMet Interpolation Visualizer Example - Adding points and auto zooming], as well as two check boxes, one that prevents the user from adding more points if the user decides to move the graph, and "show extrapolation effects", which shows the user what effects the extrapolation has on attempting to predict the "past" or "future" [Figure 6.32 BNumMet Interpolation Visualizer Example - Extrapolation].

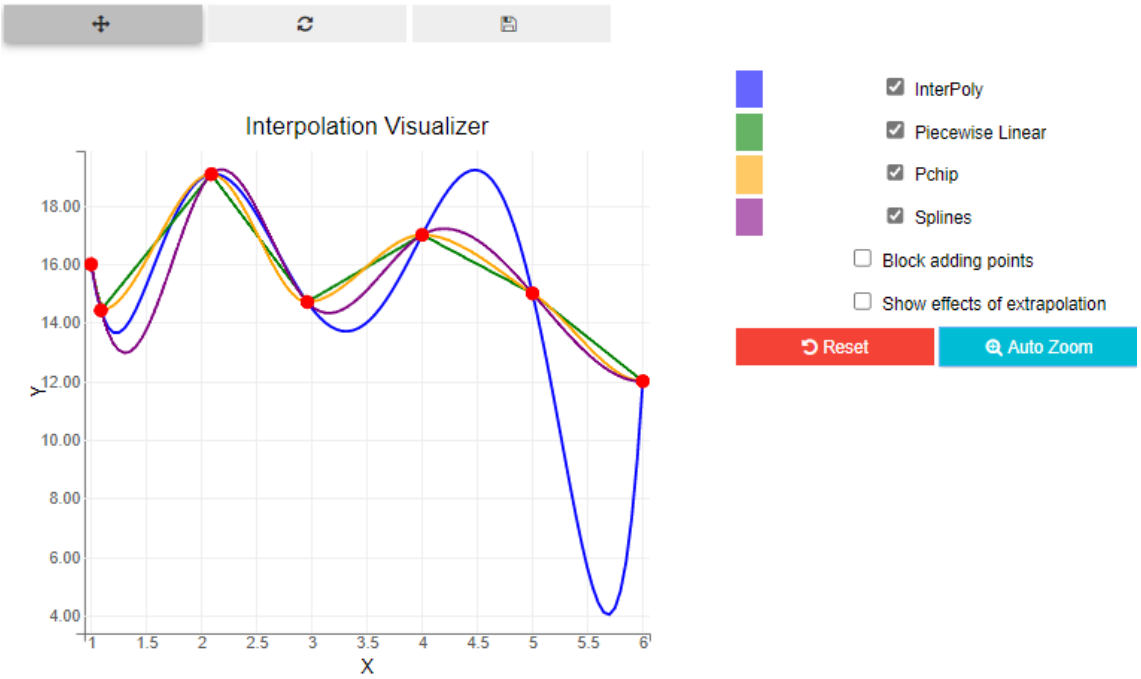


Fig. 6.31. BNumMet Interpolation Visualizer Example - Adding points and auto zooming

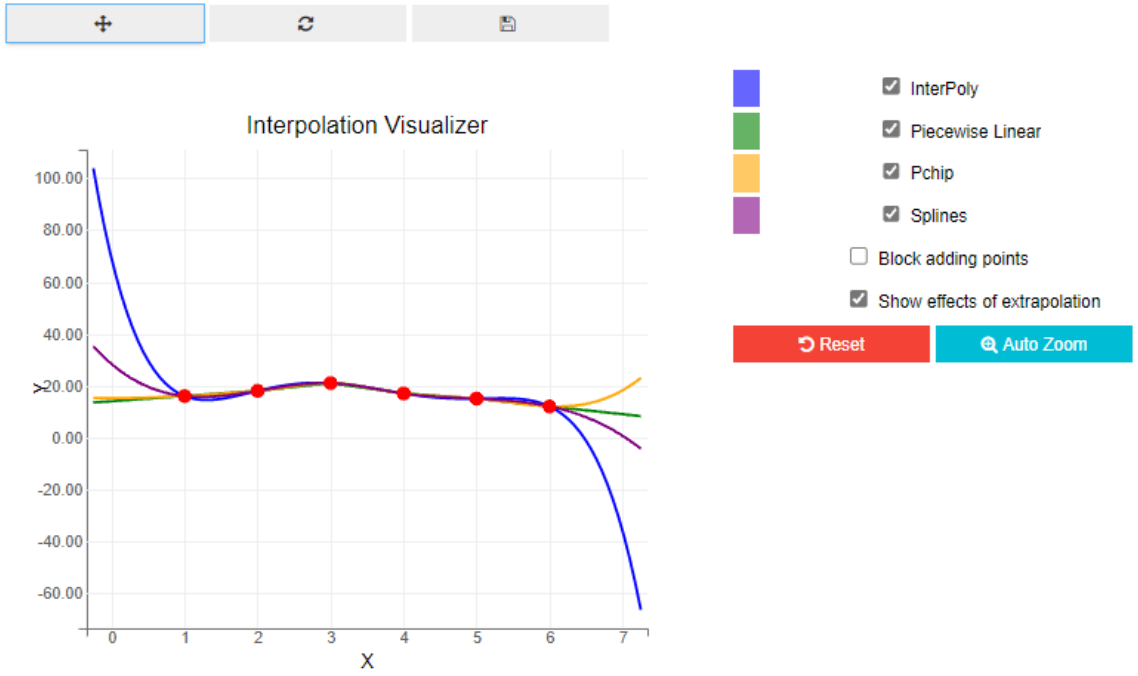


Fig. 6.32. BNumMet Interpolation Visualizer Example - Extrapolation

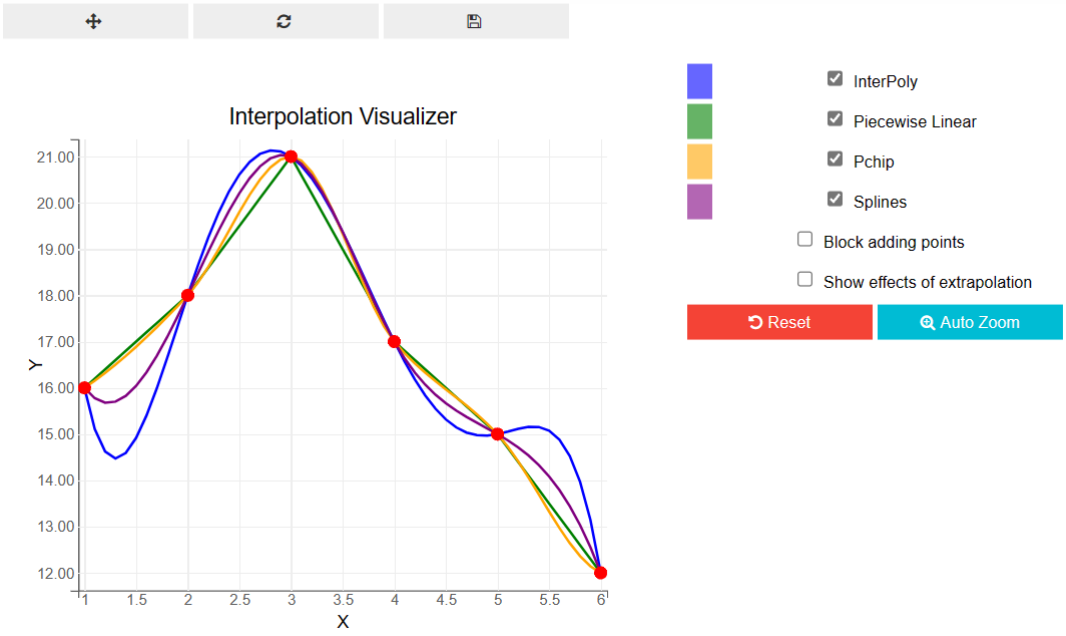
Overall, we reckon that the method supplied corrects some of the minor flaws raised by Mathwork's approach while also providing some concepts that may assist academics in explaining why interpolation should not be utilized for extrapolation.

Examples

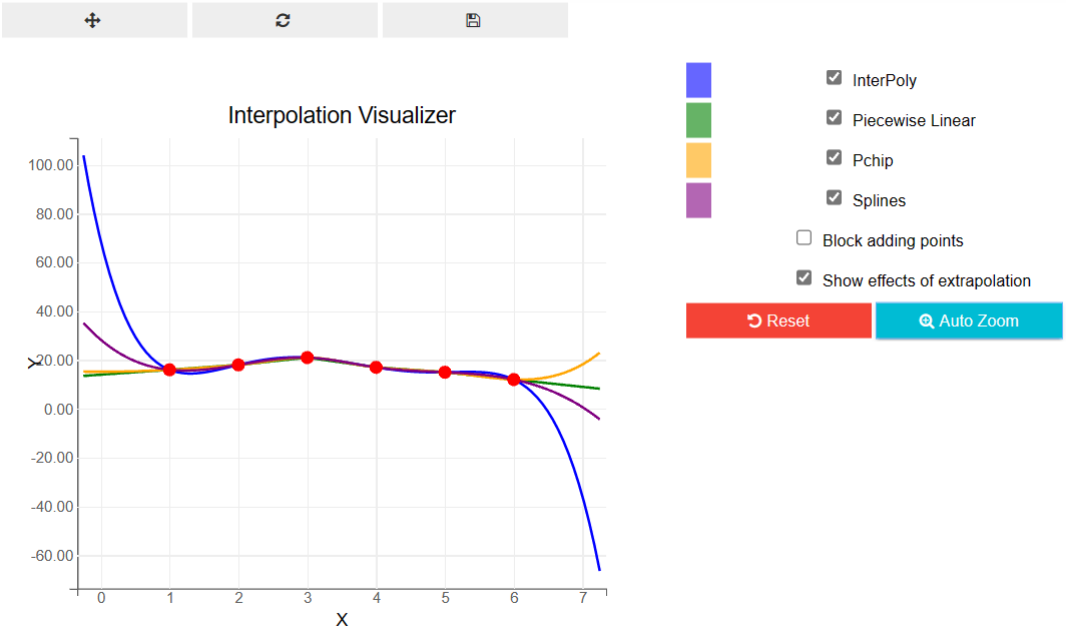
Example 1

```
from BNumMet.Visualizers.InterpolationVisualizer import
    InterpolVisualizer
interpolVisualizer = InterpolVisualizer()
display(interpolVisualizer.run())
```

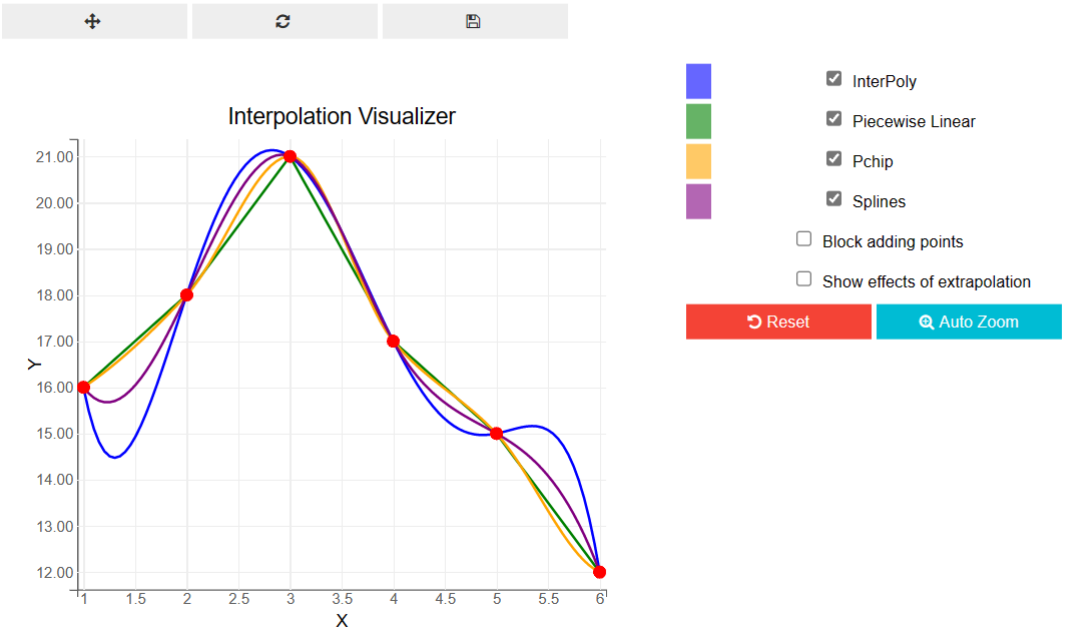
1. Initial State:



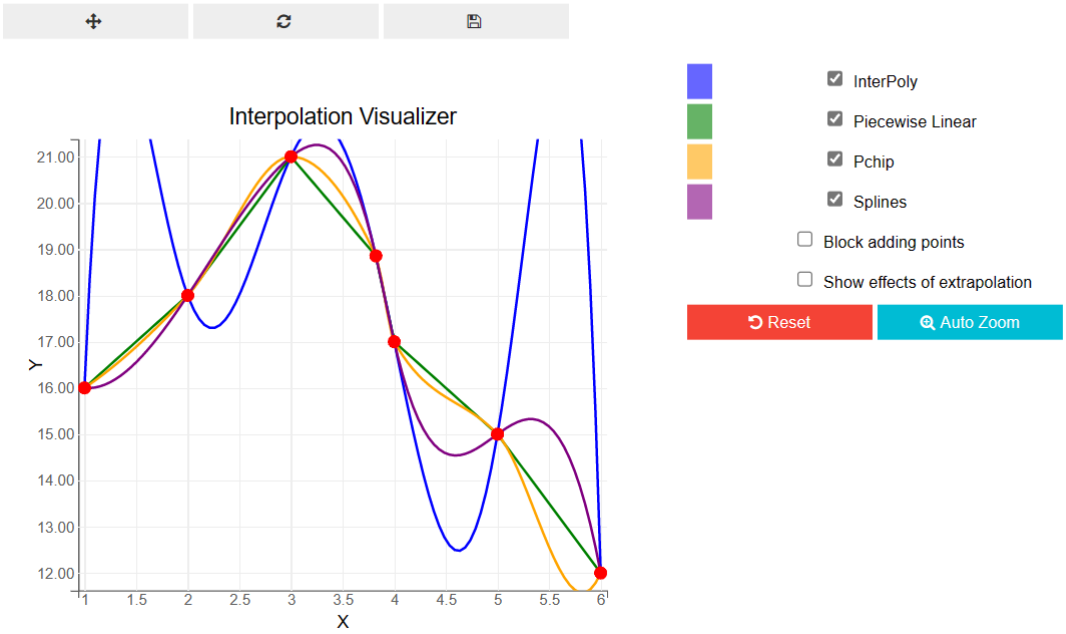
2. Checked Box Extrapolation and AutoZoom:



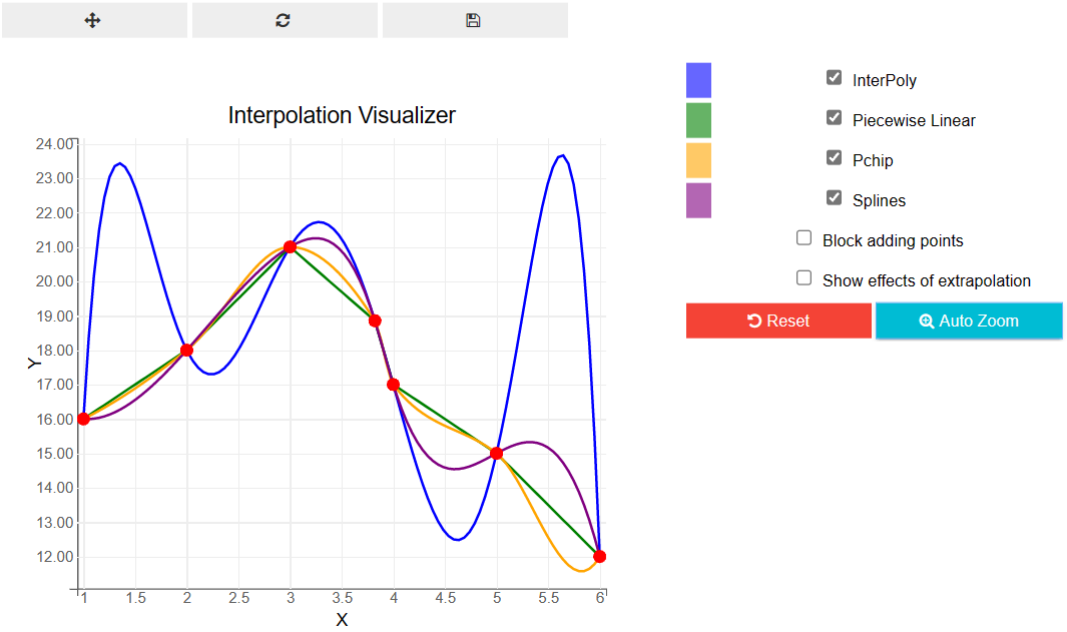
3. Reset Button:



4. Added Point:



5. AutoZoom:



6.6.3. Non Linear Equation Solver Visualizer

Continuing with our visualizers, we have also implemented an interactive visualization for the Non Linear Equation Solver Package. As before, we will first discuss the considerations taken by Mathworks and observe what ideas we have implemented in BNumMet. It is worth noting that J.C. Bucheli's work will not be considered from this point forward because it lacks implementation of these.

To begin, Mathwork's implementation considers making the Brentt-Dekker method interactive; the user will be able to select which step to choose, bisection, secant, or I.Q.I., and at each step it will automatically perform the necessary calculation and continue moving forward with the user's input; it should be noted that on Mathwork's implementation there is a button that automatically performs all the steps in accordance with the Brentt-Dekker Algorithm. Furthermore, it has two distinct colored points that will be interactive, those being the bisection or the secant/I.Q.I., though it should be noted that while the user can guess which point is which, there is no clear visual aid or text that helps the user make such a decision, and it lacks the graph that grants the next possible point, that is, the secant approach should draw a secant showing where the next point will be. We should also handle the problem that the user may not know if the points are interactive, which may cause some confusion, as well as, the final stage of the visualizer which the user may not understand when the algorithm has finished [Figure 6.34 Mathwork's [3] NonLinear Visualizer Example - Ending].

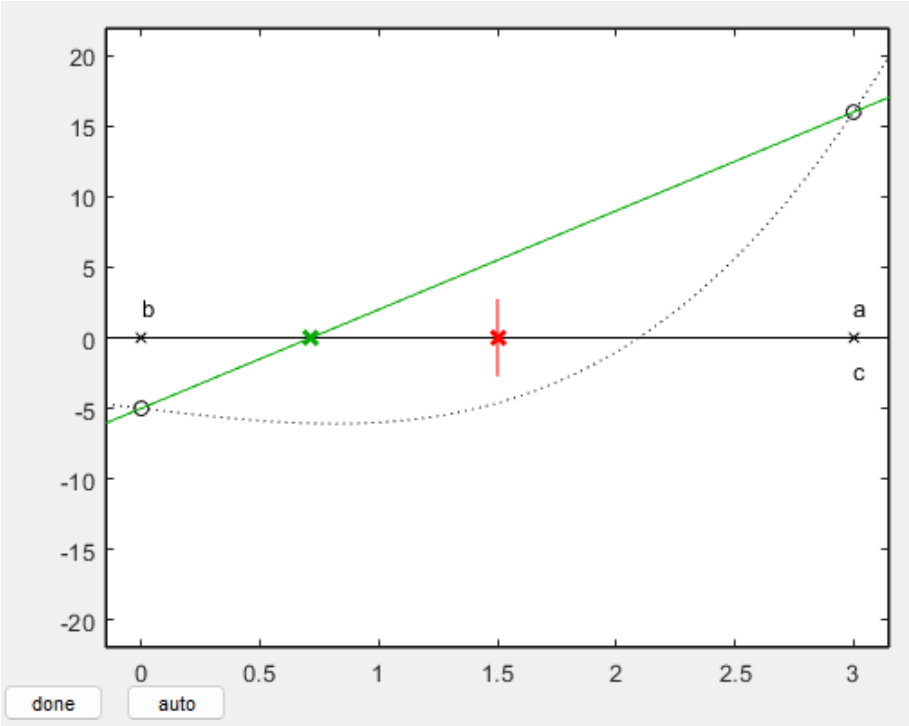


Fig. 6.33. Mathwork's [3] NonLinear Visualizer Example

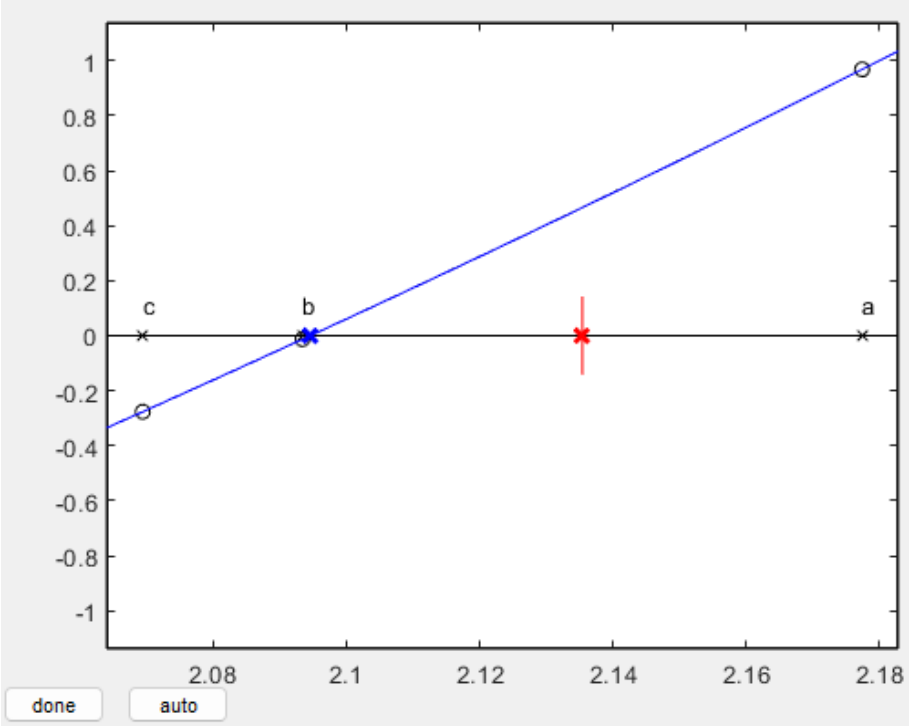


Fig. 6.34. Mathwork's [3] NonLinear Visualizer Example - Ending

We have adopted the same approach as Mathworks by adopting the algorithm established by Brent, Dekker, and his colleagues in our BNumMet implementation. We also took Mathwork's considerations, but extended and corrected

the critiques we made previously. To solve the distinction between the points, we added a legend to indicate which point is which, and we moved the "interactivity" of the points to an external button so that the user knows what he is clicking on in every iteration [Figure 6.35 BNumMet's NonLinear Visualizer Example 1]. We've also included a checkbox to the right of those buttons to enable the user choose whether to display the graph linked with the secant, I.Q.I, or bisection [Figure 6.36 BNumMet's NonLinear Visualizer Example 1 - Graphs for points]. It is also worth noting that we present what the original algorithm would be using, as well as information on the current solution the user has discovered and the amount of iterations the user has taken. It is worth noting that during each step the graph will resize accordingly for the user to properly visualize it.

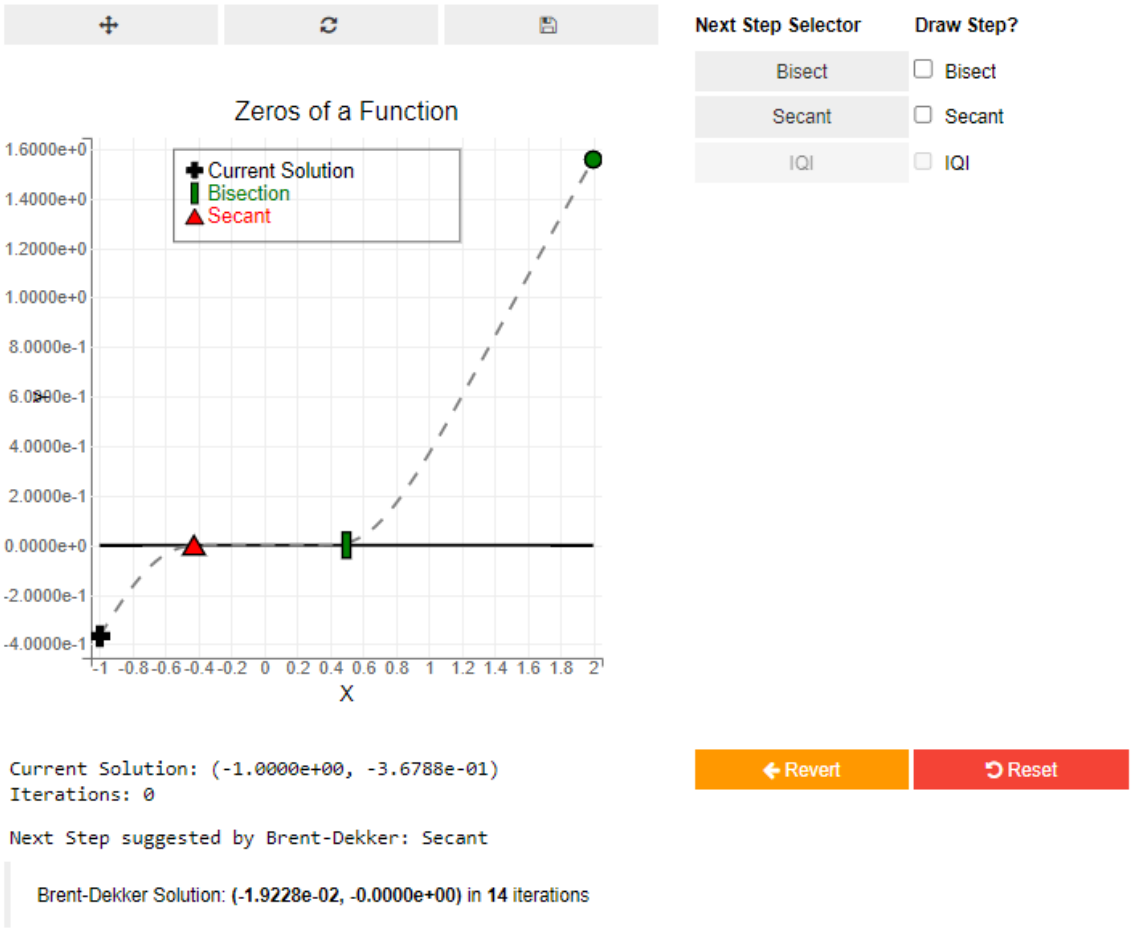


Fig. 6.35. BNumMet's NonLinear Visualizer Example 1

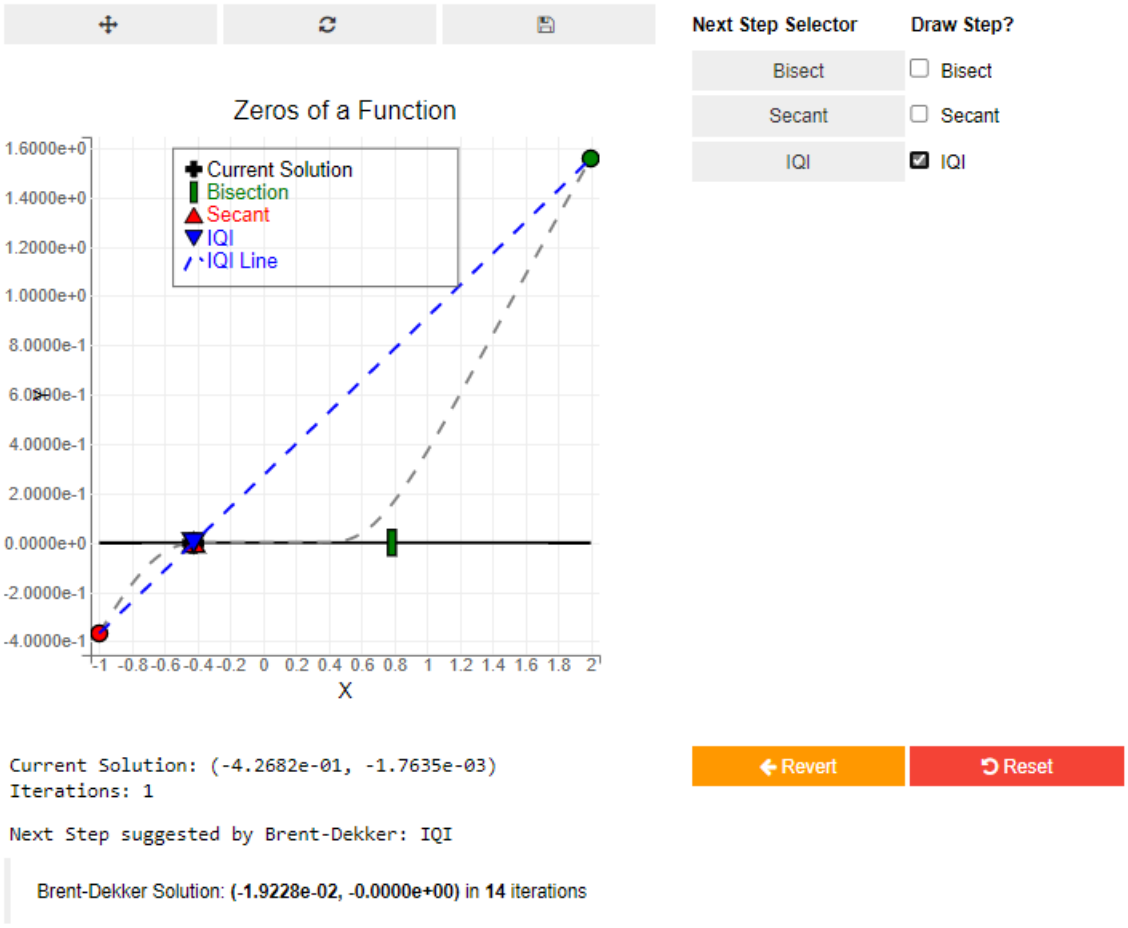


Fig. 6.36. BNumMet's NonLinear Visualizer Example 1 - Graphs for points

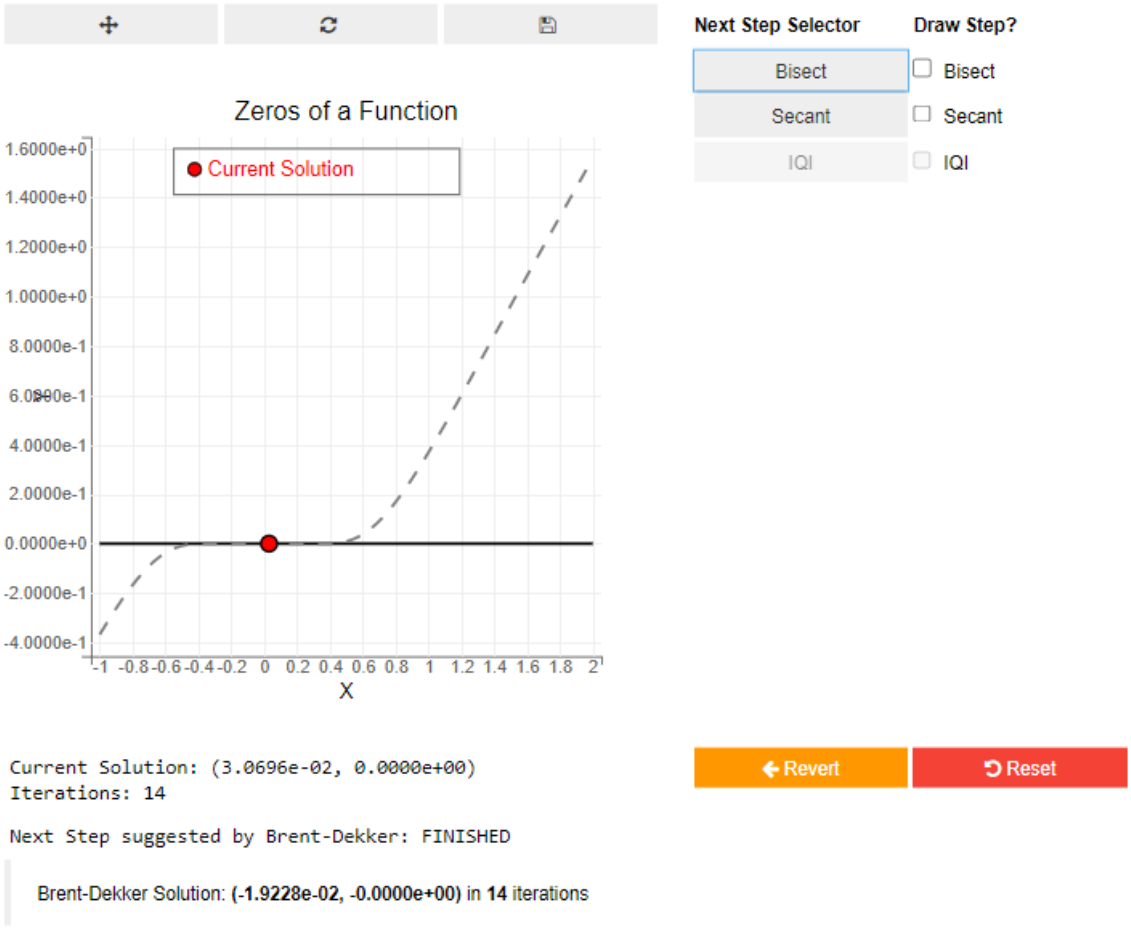


Fig. 6.37. BNumMet's NonLinear Visualizer Example - Ending

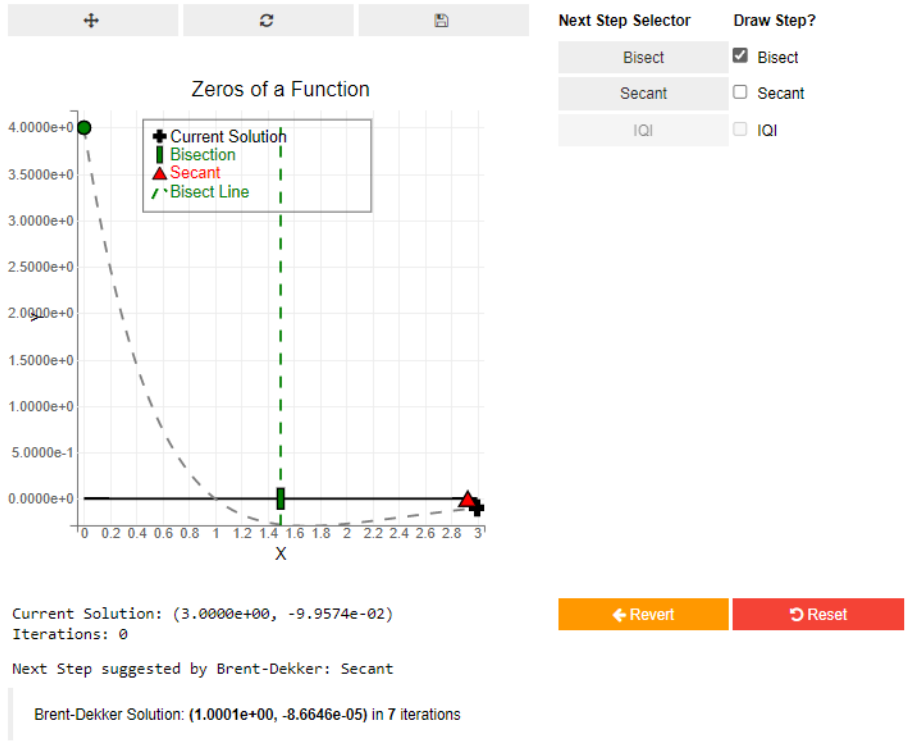
Examples

Example 1: No arguments

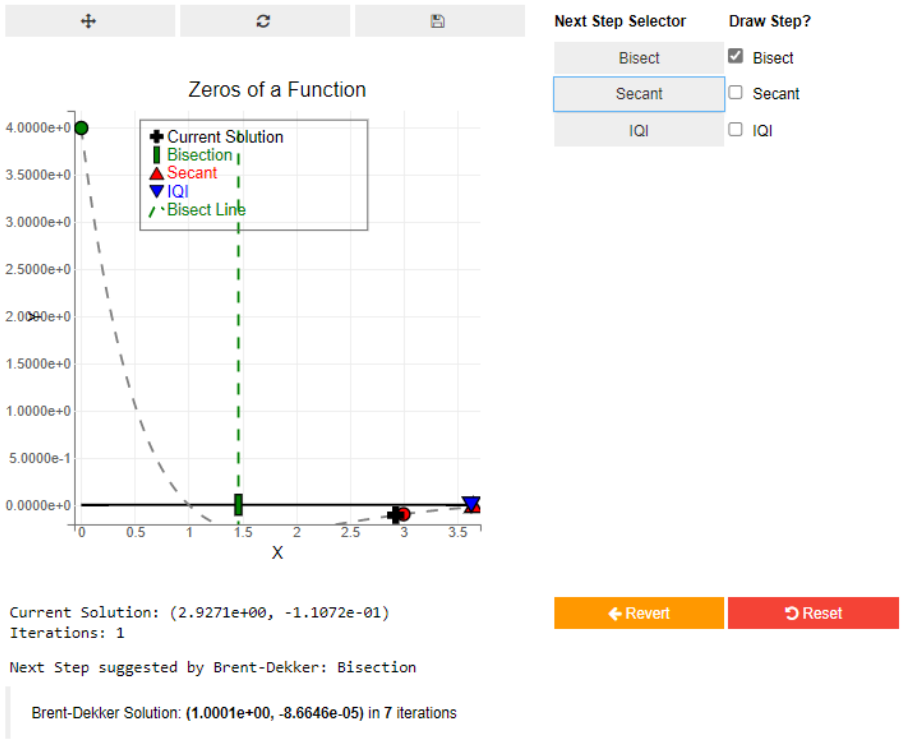
```
from BNumMet.Visualizers.NonLinearVisualizer import
    NonLinearVisualizer
zerosVisualizer = NonLinearVisualizer()
zerosVisualizer.run()
```

1. Initial State

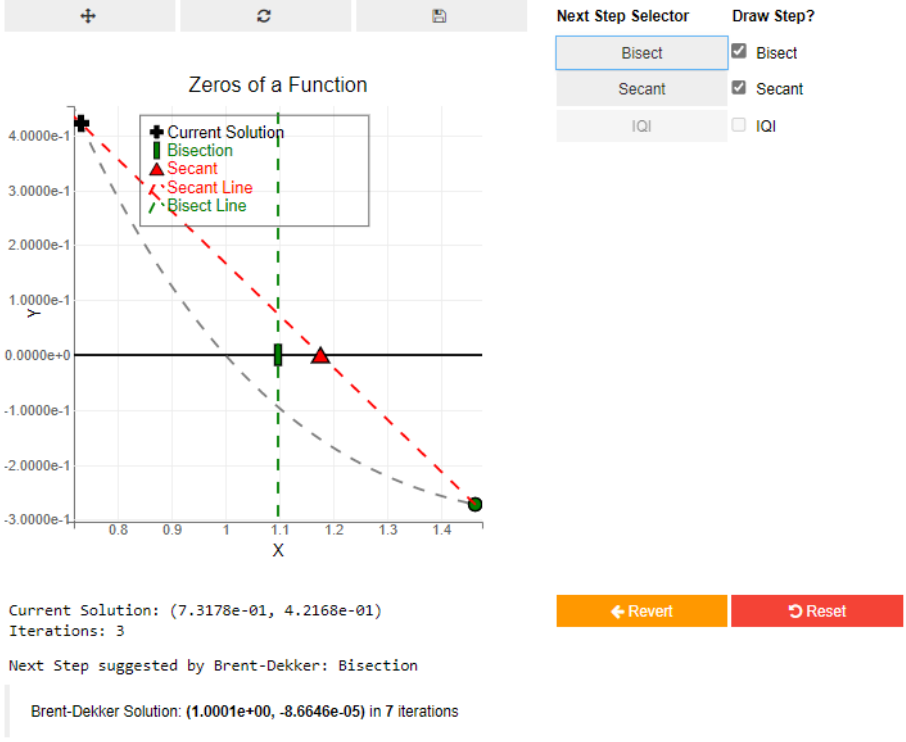
2. Click on bisection checkbox



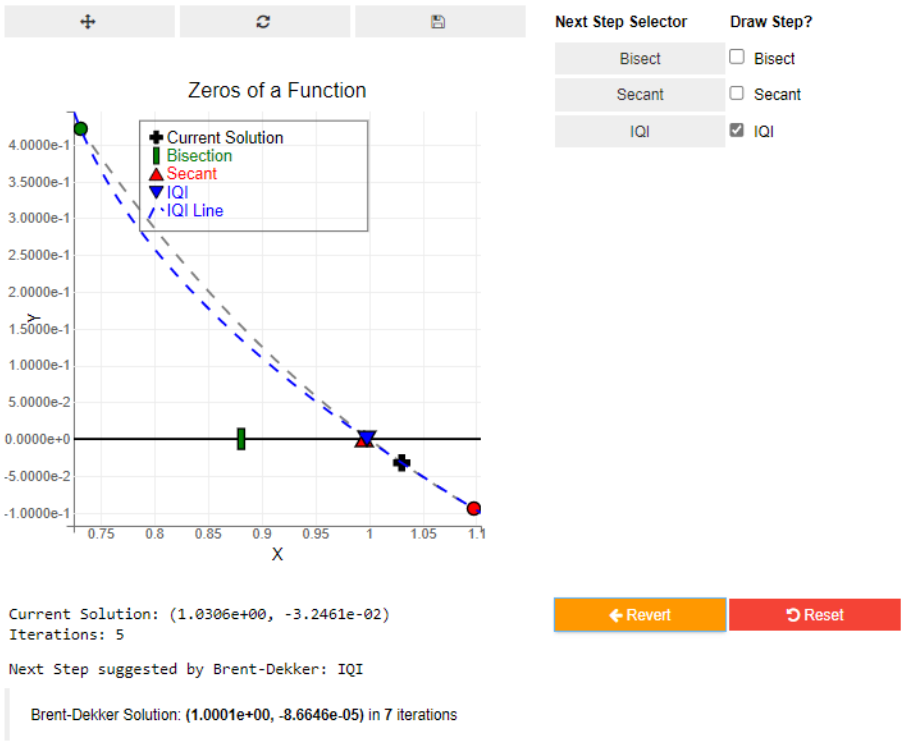
3. Click Secant Button



4. Some iterations after with secant checkbox clicked



5. IQI Checkbox Only



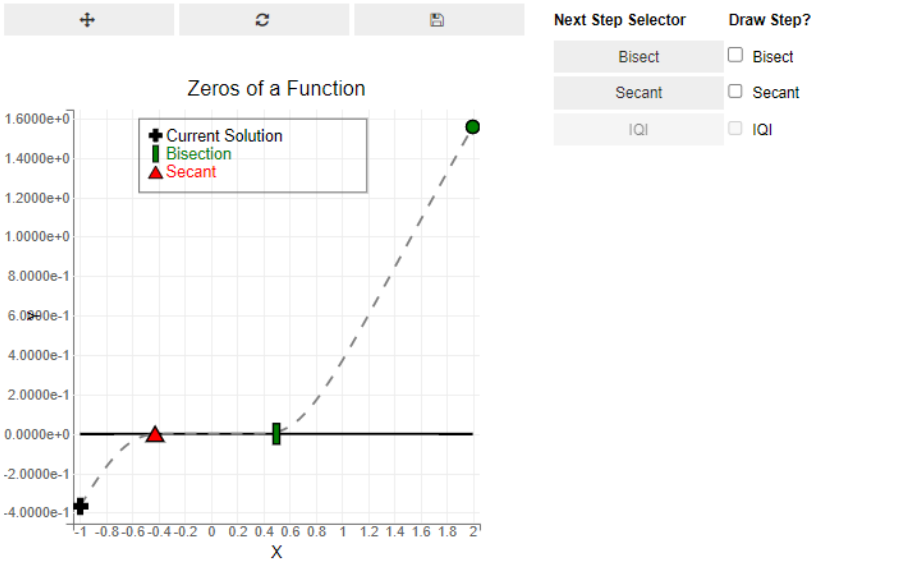
6. Finished after some iterations

Example 2: With Input Arguments

```
from BNumMet.Visualizers.NonLinearVisualizer import
    NonLinearVisualizer
f2 = lambda x: 0 if abs(x) < 3.8 * 10 ** (-4) else float(x *
    np.exp(-1 / x**2))
interval = (-1, 2)
zerosVisualizer = NonLinearVisualizer(f2,interval)
```

```
5 zerosVisualizer.run()
```

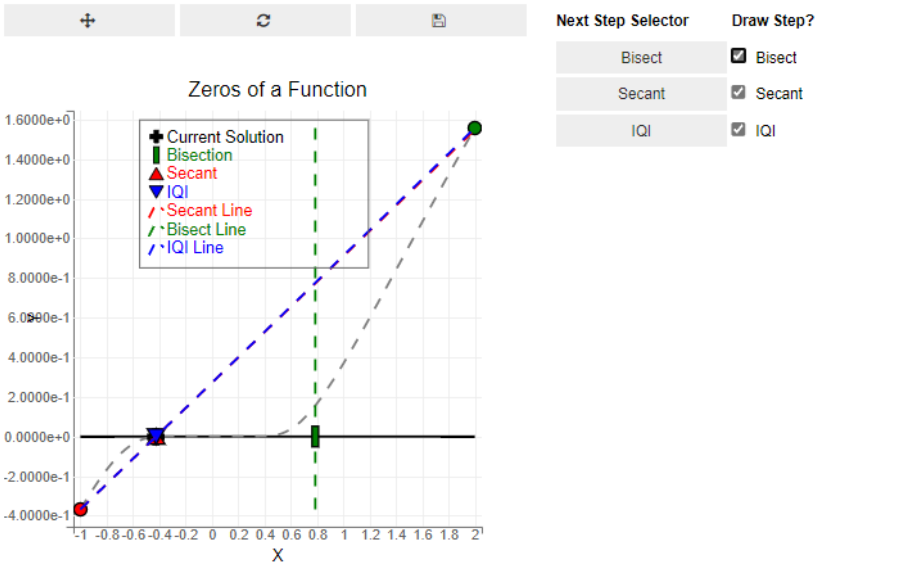
1. Initial State




```
Current Solution: (-1.0000e+00, -3.6788e-01)
Iterations: 0

Next Step suggested by Brent-Dekker: Secant
```

Brent-Dekker Solution: **(-1.9228e-02, -0.0000e+00)** in **14** iterations

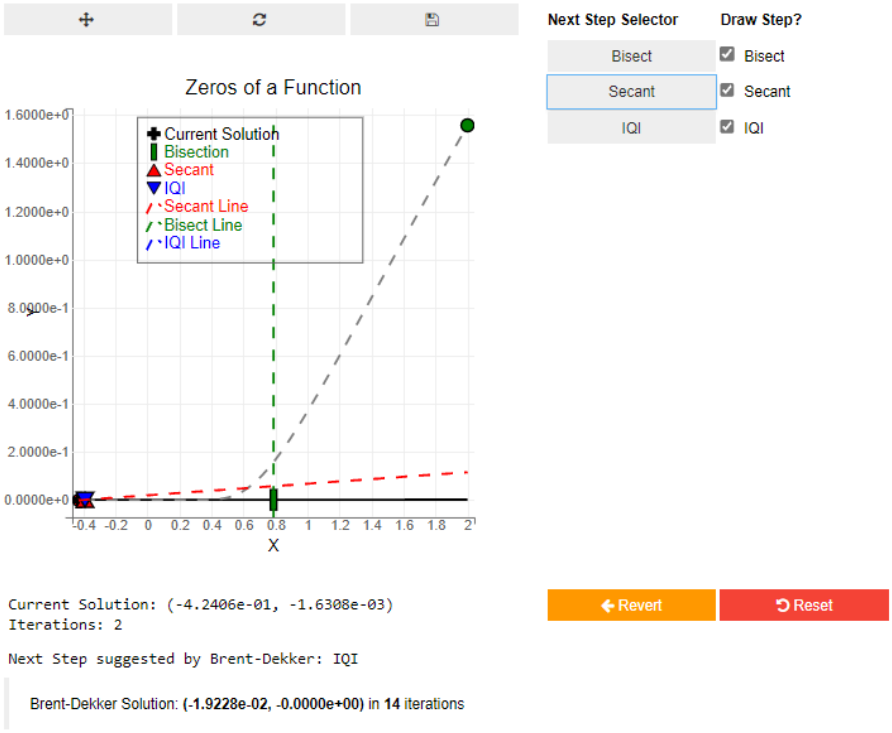
2. All Checkboxes checked



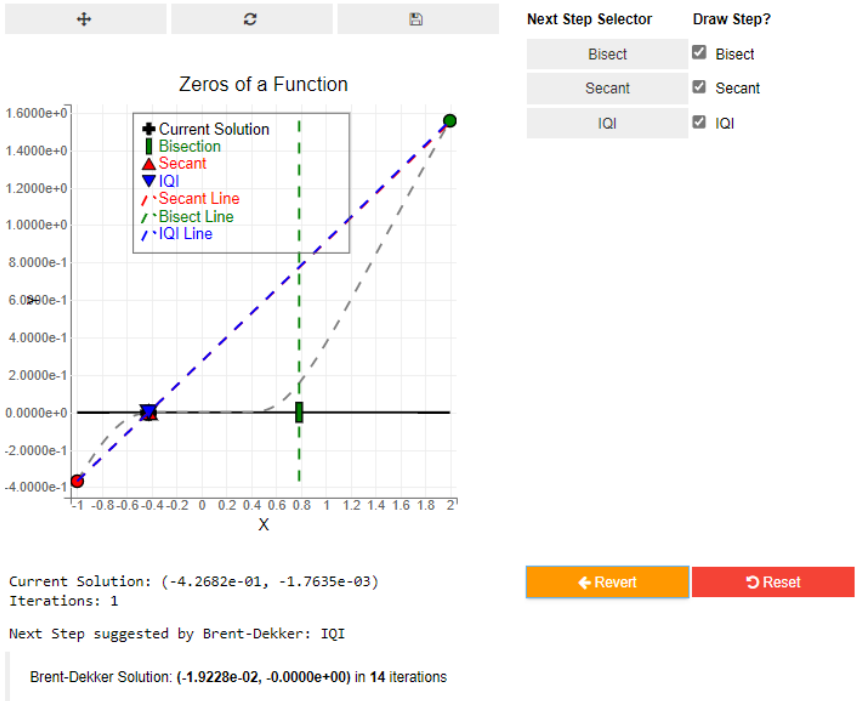

```
Current Solution: (-4.2682e-01, -1.7635e-03)
Iterations: 1

Next Step suggested by Brent-Dekker: IQI
```

Brent-Dekker Solution: **(-1.9228e-02, -0.0000e+00)** in **14** iterations

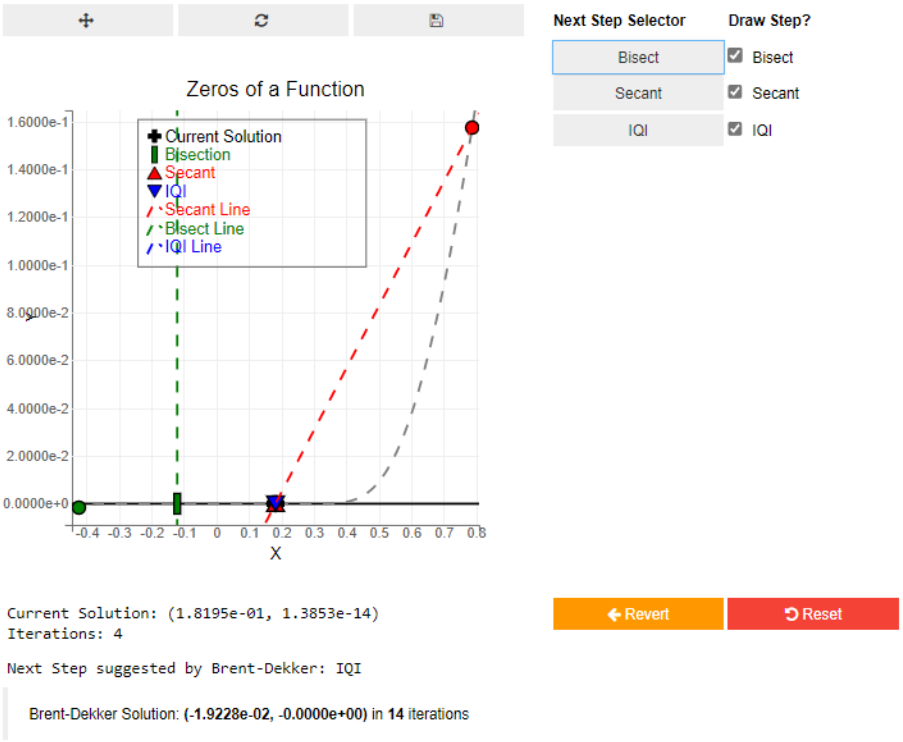
3. Click secant and click select all checkboxes again



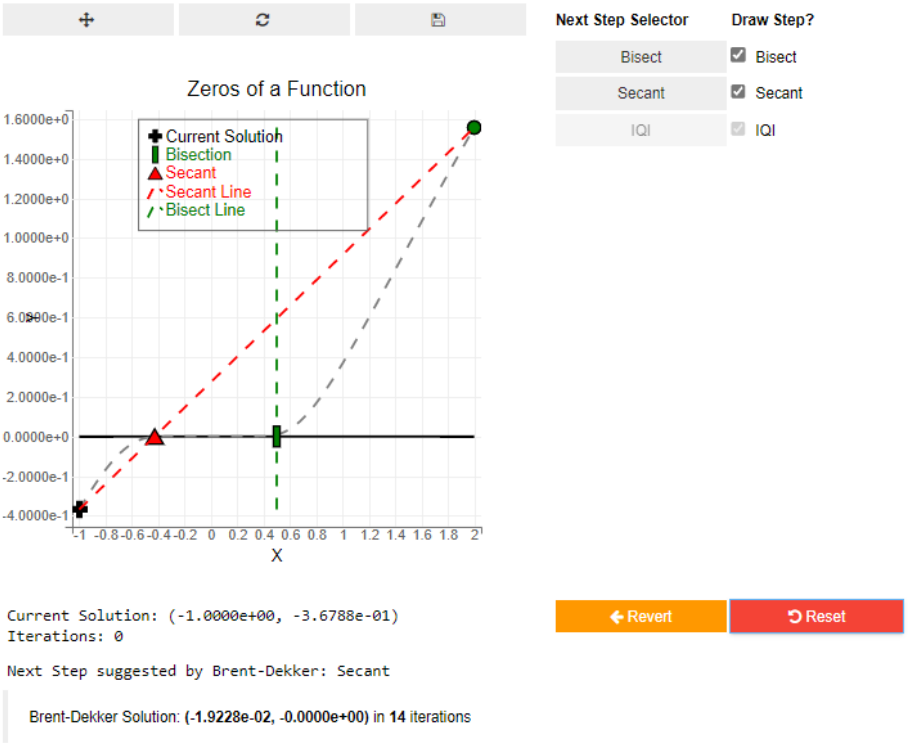
4. Revert Button



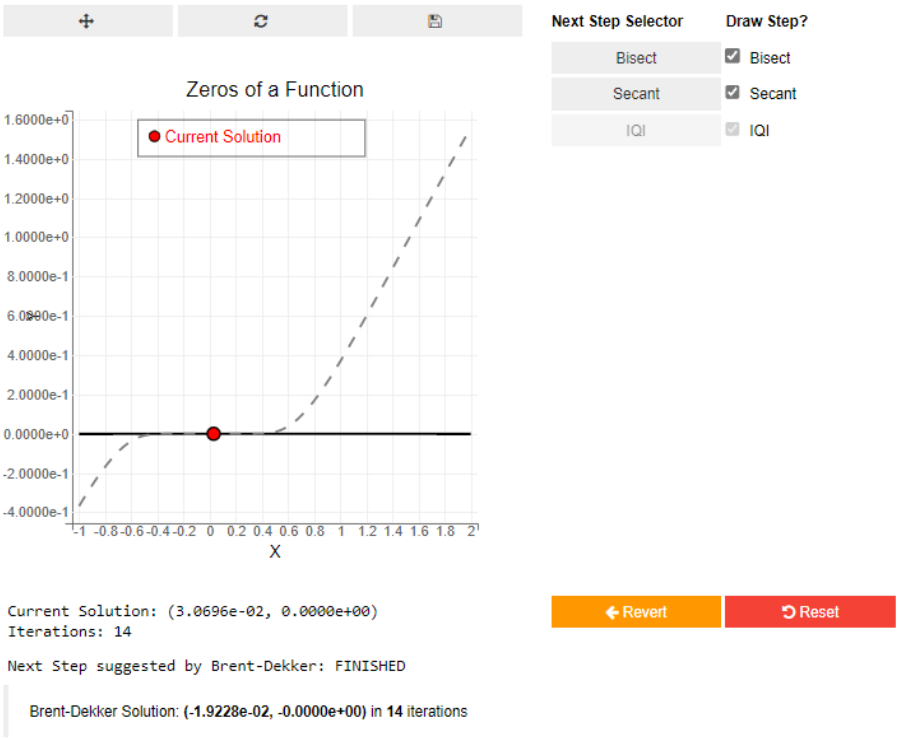
5. Some iterations



6. Reset Button clicked



7. Finished after some iterations



6.6.4. Least Squares Problem Visualizer

Although no mention of Least Squares was made during the discussion of the multiple numerical methods that BNumMet has, this problem is one of the pinnacles of data fitting and a complex problem with a simple approach to solve. As a result, we felt it was necessary to create a visualizer to give students with a complete and interactive tool to examine what least squares perform.

The main idea of the visualizer is to give the student a set of points (or allow them to enter their own data) and show them how to solve the least squares problem using various types of functions. All of the calculations will be done in the background while the final plot will be shown to demonstrate that in these types of problems, the goal is not to interpolate but to find a function that minimizes the error.

As with the previous visualizer, we will begin by discussing Mathwork's implementation; in this case, C. Moller's work is outstanding; he provides data from the US Census and presents the user with a selector of two functions for applying least squares and another two that interpolate the points. It also offers an estimate of the inaccuracy for each approach. It should be noted that it does not include the resultant equation that yields the least squares solution, which may be valuable for any learner checking their own hand-made computations.

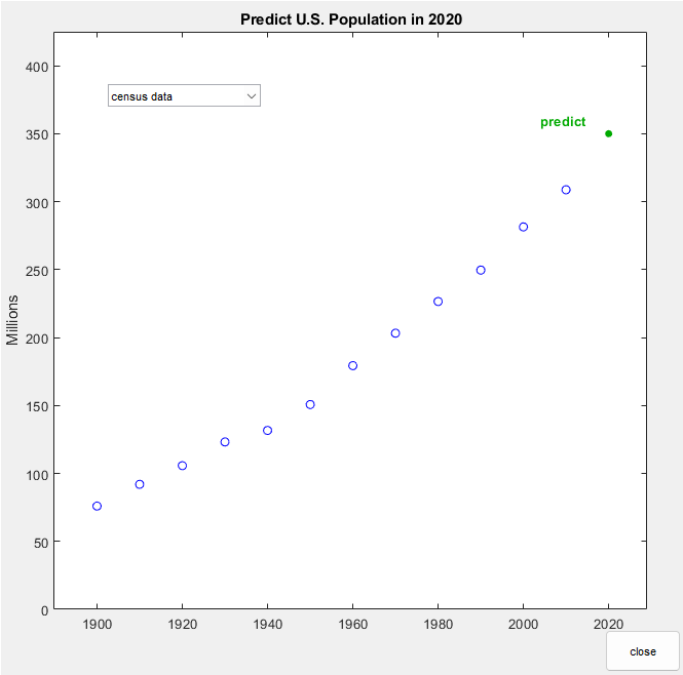


Fig. 6.38. Mathwork's [3] LSP Visualizer Example 1 - Data

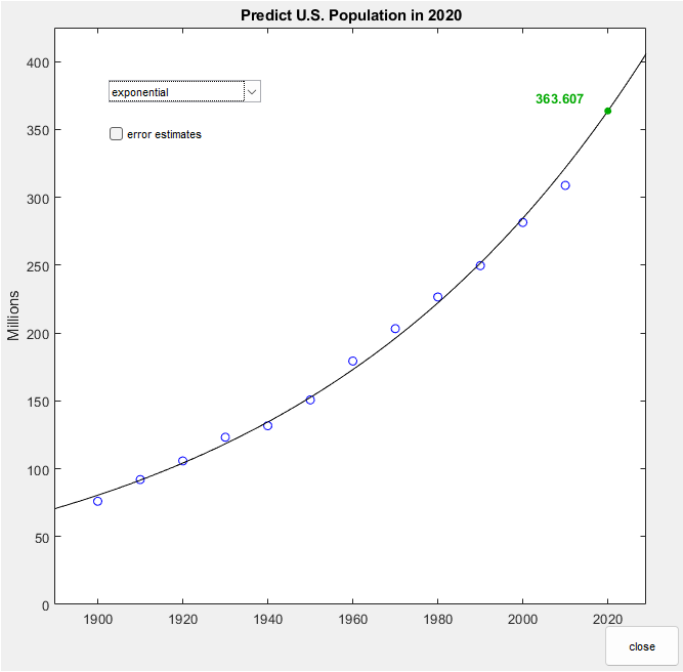


Fig. 6.39. Mathwork's [3] LSP Visualizer Example - Exponential

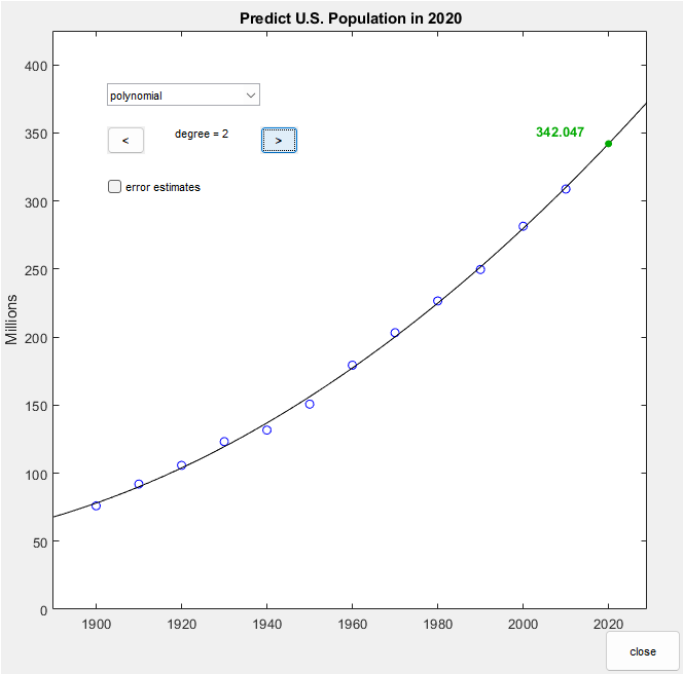


Fig. 6.40. Mathwork's [3] LSP Visualizer Example - Polynomial

We generalized this approach in BNumMet's implementation by implementing just the techniques that relate to the real least squares issue, rather than the interpolating methods supplied in the interpolation package. We also noticed that, while Mathwork's approach demonstrated an interesting application about predicting the future of the population of the United States, it lacked the ability for the user to input their own data set, which we fixed in BNumMet's implementation and it may be regarded as a more interesting approach for students while learning. In addition, we added a little note indicating the output function as well as the inaccuracy that the solution presents. We would also like to mention that we added the Sines and Cosines Basis for this problem since we thought it would provide insight into the fact that the least squares problem is not limited to a small set of functions but includes many more.

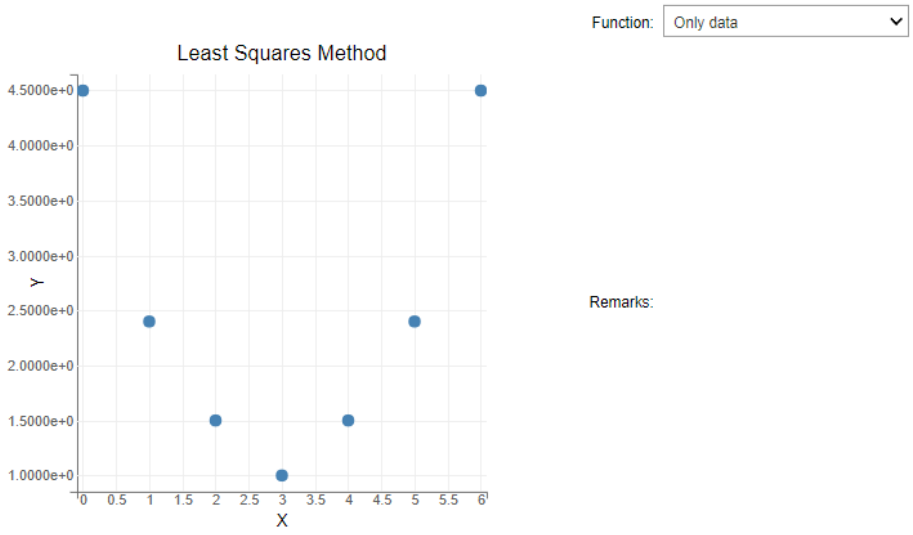


Fig. 6.41. BNumMet's LSP Visualizer Example

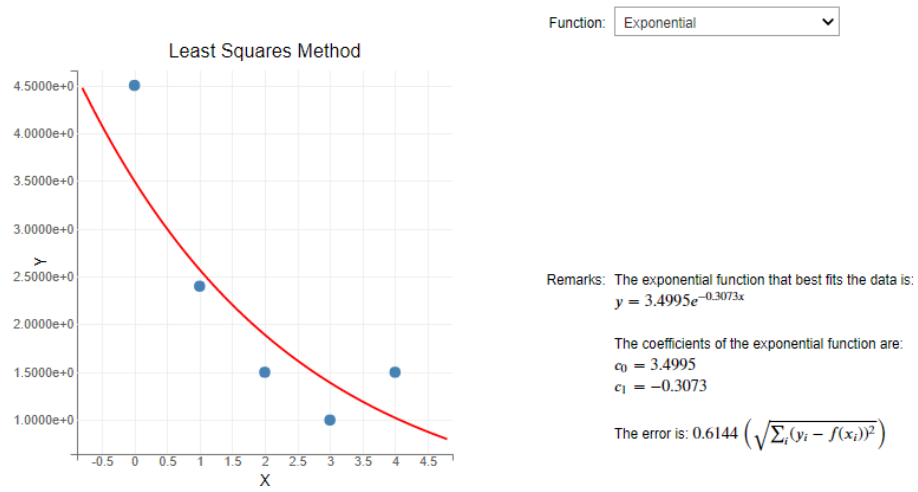


Fig. 6.42. BNumMet's LSP Visualizer Example - Exponential

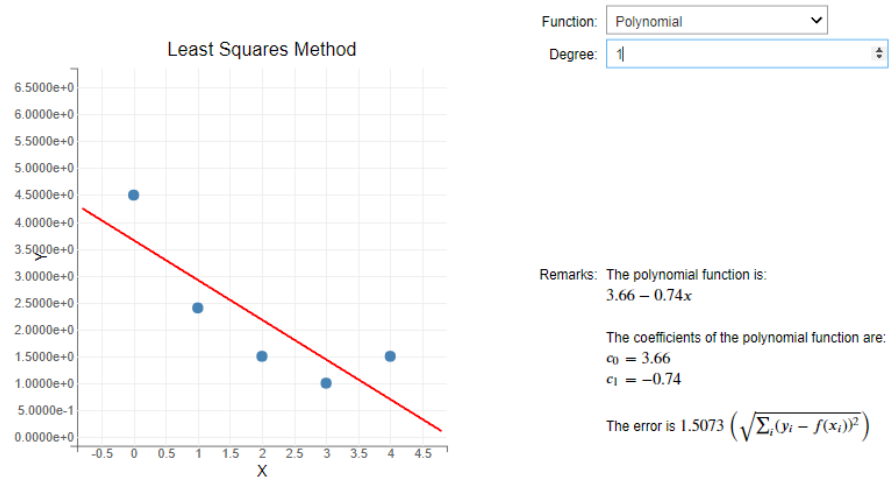


Fig. 6.43. BNumMet's LSP Visualizer Example - Polynomial

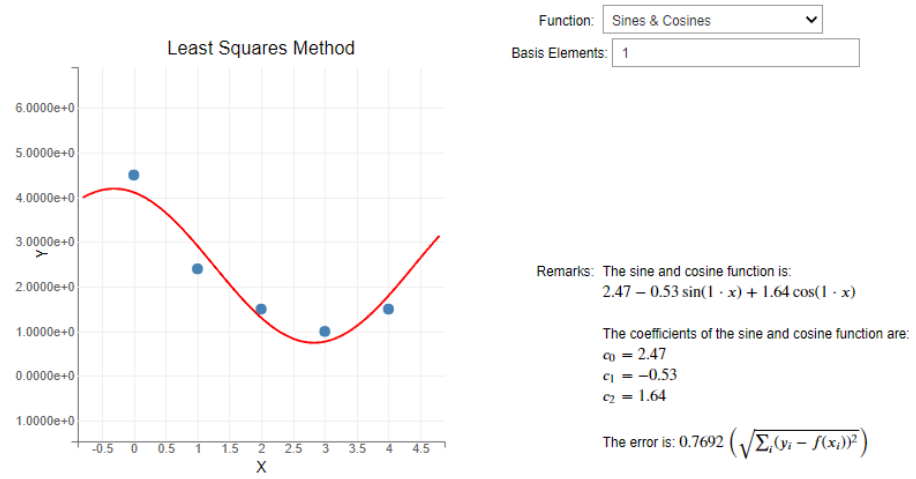


Fig. 6.44. BNumMet's LSP Visualizer Example - Sines and Cosines

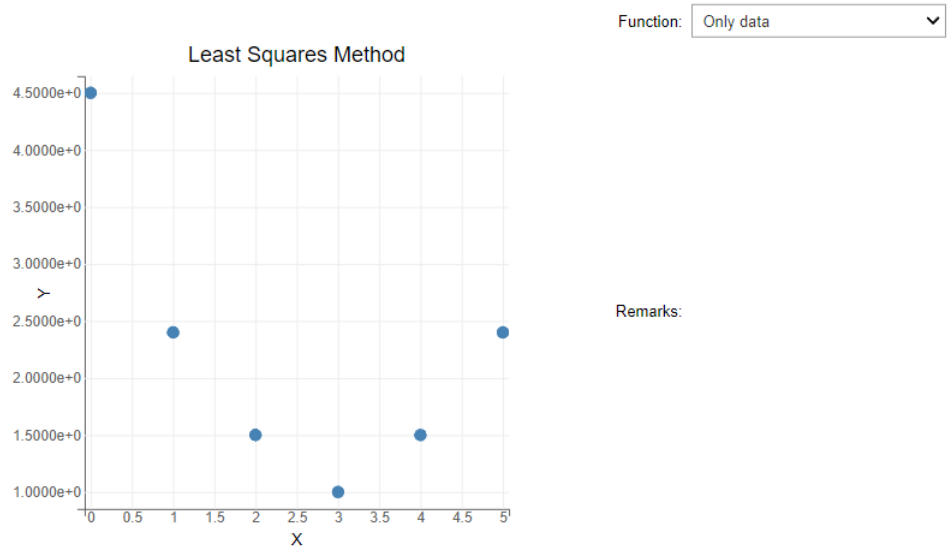
As we can see we have removed the error estimates that Mathwork's implementation presents, we believed it may be confusing and without a proper definition of how it is calculated.

Examples

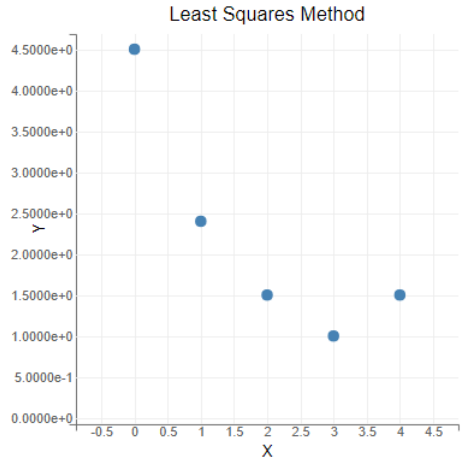
Example 1: Polynomial Fit of Varying degree

```
from BNumMet.Visualizers.LeastSquaresVisualizer import
    LSPVisualizer
xData = np.array([0, 1, 2, 3, 4, 5])
yData = np.array([4.5, 2.4, 1.5, 1, 1.5, 2.4])
lspVisualizer = LSPVisualizer(xData, yData)
lspVisualizer.run()
```

1. Initial State



2. Selector

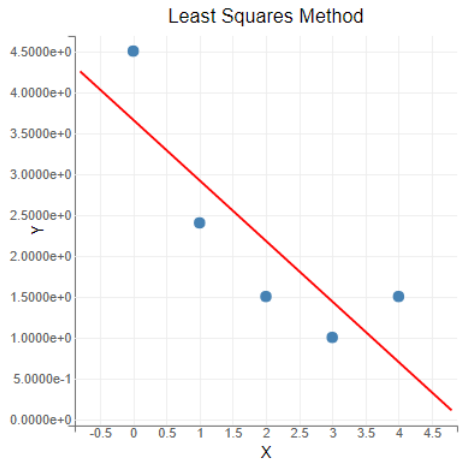


Function: Only data

Only data
Polynomial
Exponential
Sines & Cosines

Remarks:

3. Select Polynomial



Function: Polynomial

Degree: 1

Remarks: The polynomial function is:

$3.66 - 0.74x$

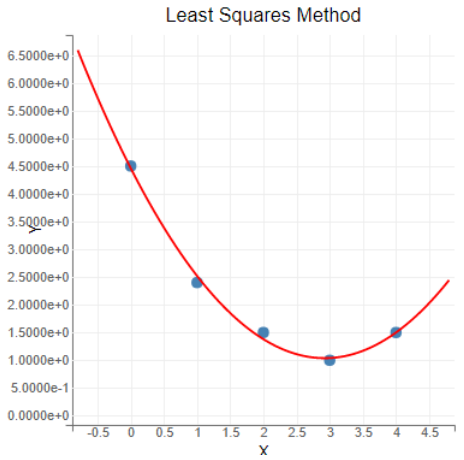
The coefficients of the polynomial function are:

$c_0 = 3.66$

$c_1 = -0.74$

The error is $1.5073 \left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

4. Increment degree by 1 (degree 2)



Function: Polynomial

Degree: 2

Remarks: The polynomial function is:

$4.46 - 2.34x + 0.40x^2$

The coefficients of the polynomial function are:

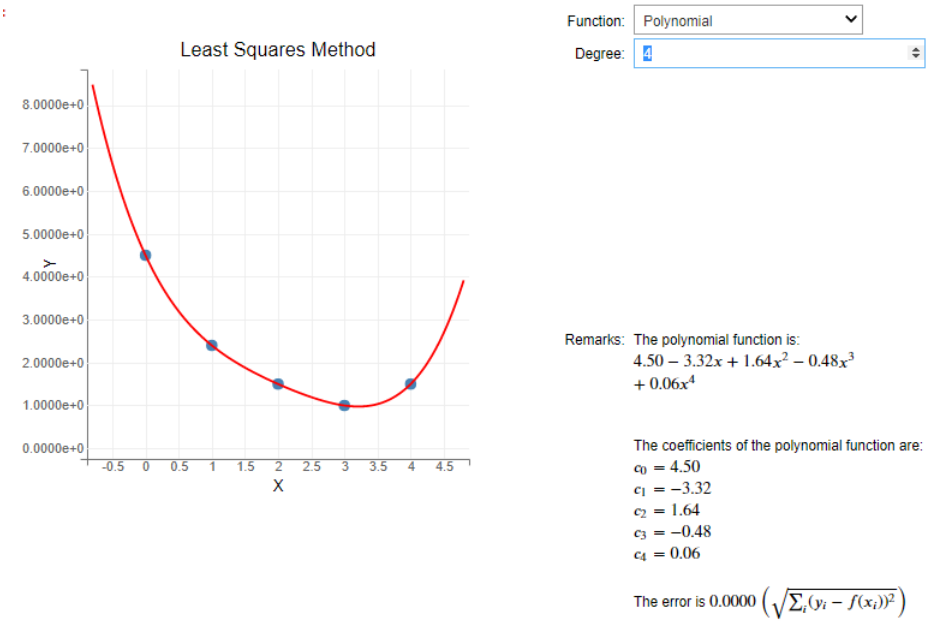
$c_0 = 4.46$

$c_1 = -2.34$

$c_2 = 0.40$

The error is $0.1789 \left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

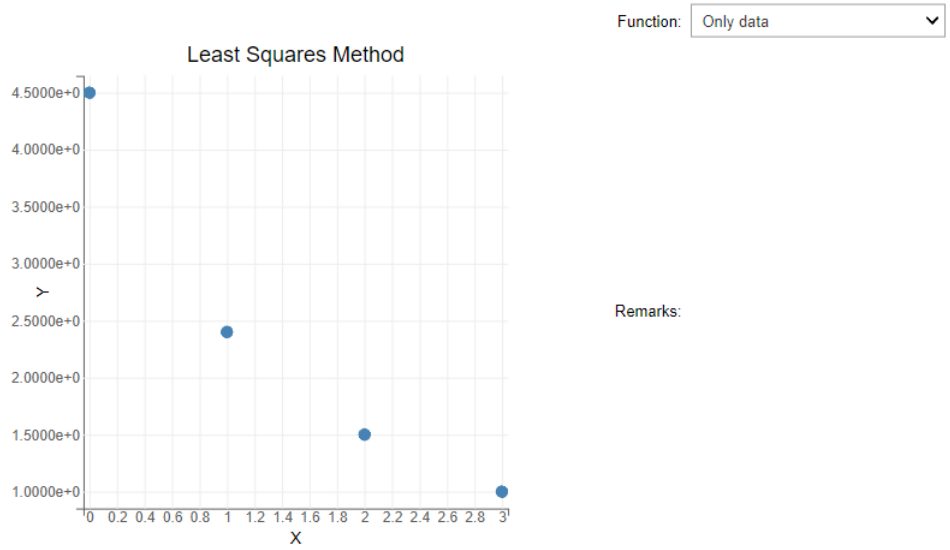
5. Degree 5 (Maximum, cannnot go higher)



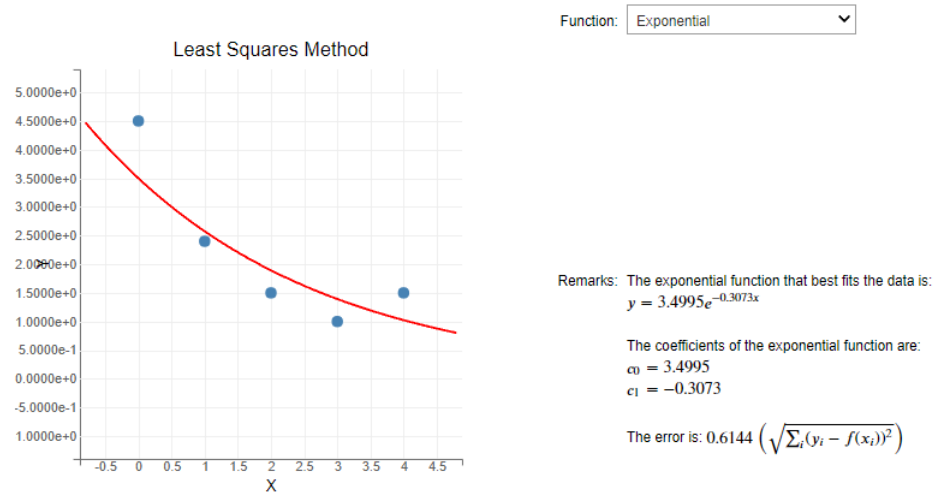
Example 2: No Input Data and Exponential Fit

```
from BNumMet.Visualizers.LeastSquaresVisualizer import
    LSPVisualizer
lspVisualizer = LSPVisualizer()
lspVisualizer.run()
```

1. Initial state



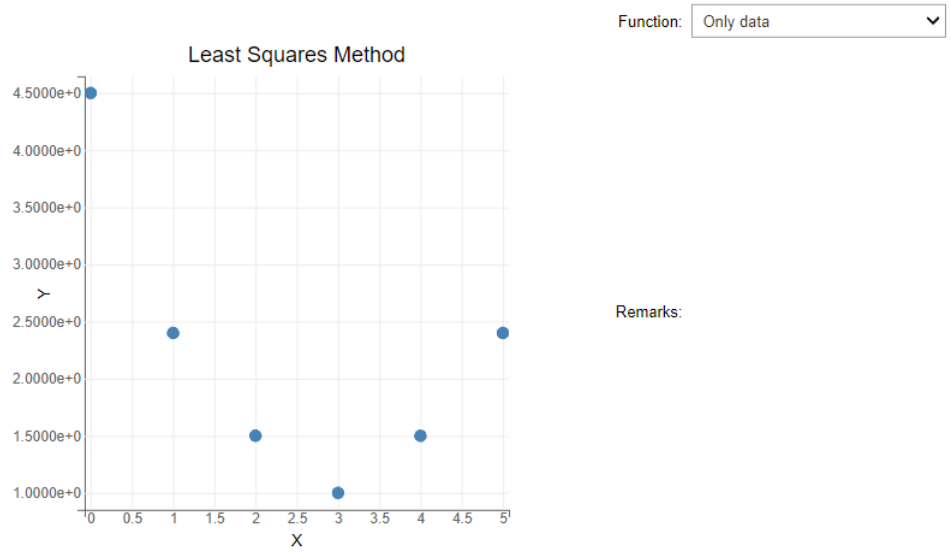
2. Exponential Selection



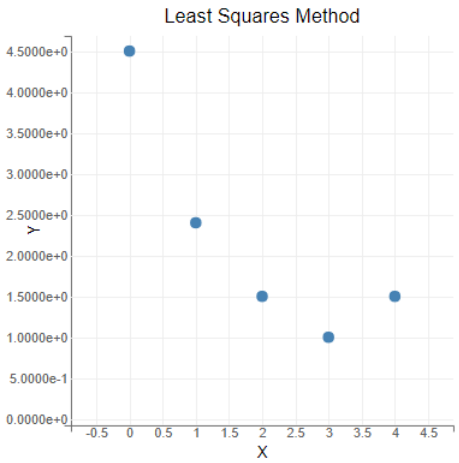
Example 3: Sines and Cosines Fit of Varying degree

```
from BNumMet.Visualizers.LeastSquaresVisualizer import
    LSPVisualizer
xData = np.array([0, 1, 2, 3, 4, 5])
yData = np.array([4.5, 2.4, 1.5, 1, 1.5, 2.4])
lspVisualizer = LSPVisualizer(xData, yData)
lspVisualizer.run()
```

1. Initial State



2. Selector

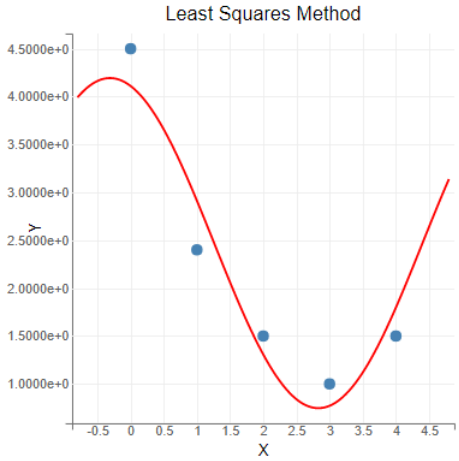


Function: Only data

Only data
Polynomial
Exponential
Sines & Cosines

Remarks:

3. Select Sines & Cosines



Function: Sines & Cosines

Basis Elements: 1

Remarks: The sine and cosine function is:

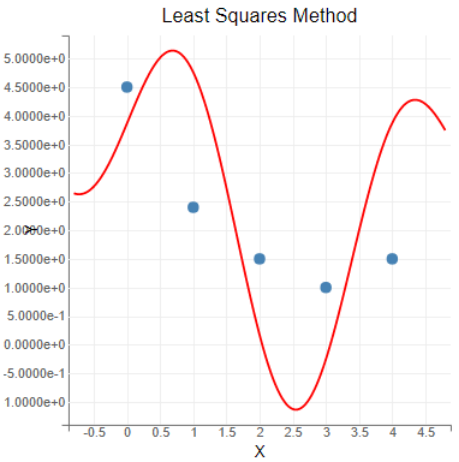
$$2.47 - 0.53\sin(1 \cdot x) + 1.64\cos(1 \cdot x)$$

The coefficients of the sine and cosine function are:

$c_0 = 2.47$
$c_1 = -0.53$
$c_2 = 1.64$

The error is: $0.7692\left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

4. Set degree to the maximum



Function: Sines & Cosines

Basis Elements: 2

Remarks: The sine and cosine function is:

$$2.61 - 0.77\sin(1 \cdot x) + 1.78\cos(1 \cdot x) - 0.53\sin(2 \cdot x) + 0.11\cos(2 \cdot x)$$

The coefficients of the sine and cosine function are:

$c_0 = 2.61$
$c_1 = -0.77$
$c_2 = 1.78$
$c_3 = -0.53$
$c_4 = 0.11$

The error is: $3.8623\left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

6.6.5. Random Visualizer

A part of applied mathematics outreach is the common application that goes as follows, throw at random n-number of darts inside a square with an inner circle with diameter the size of the square, when computing the number of darts that are inside the circle and the ones outside, one can with more or less accuracy find the value of π. For this reason we will implement this simple game as part of the random visualization package, to show students the power of Montecarlo's Simulation, that is, we will implement a target and using a random number generator we will simulate the throwing of darts to then observe how we obtain pi.

First, as with the other visualizers, we examine Mathwork's implementation; in this case, C. Moller chose a 3D version of the game; the method allows for the input of different types of generators; however, we must note that this implementation does not allow for varying the number of points, but it does allow for the simulation to be repeated.

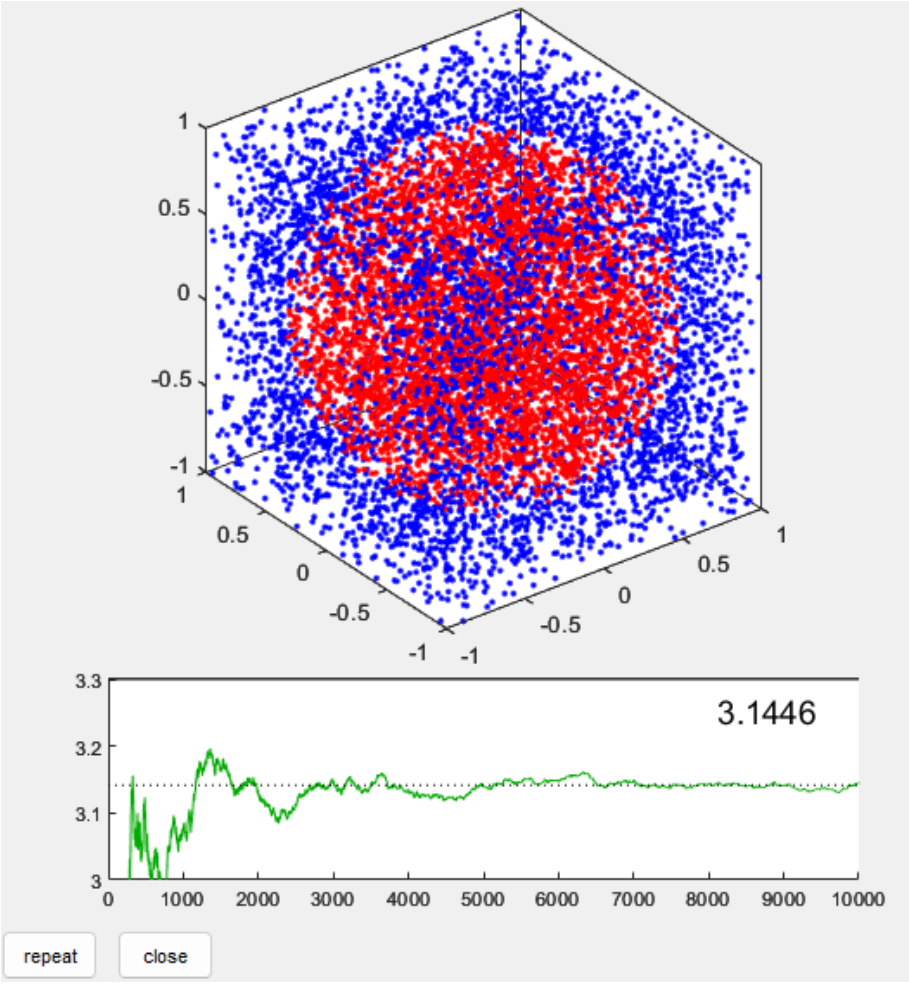


Fig. 6.45. Mathwork's [3] Random Visualizer Example

A similar implementation was done in BNumMet, but in this instance we

used the 2D method since we believe the students would find it easier to grasp, observe, and even write it themselves. We allowed the user to select the number of points they wanted to toss as part of the implementation. We have provided a graph that depicts how the simulation approaches π. To address the major issue with Mathwork's, we created a button that allows the user to run the simulation with the specified number of points anytime they want. And, as C. Moller suggested, we included the value of pi obtained when the program iterated. One disadvantage of BNumMet's implementation over C. Moller's is that we must restrict the number of points to 10000 since for such numbers the animation takes very long. That is why we chose to implement the Montecarlo Simulation with batches, that is, with an n-sized batch that saves the data until they reach the appropriate length and then updates the animation.

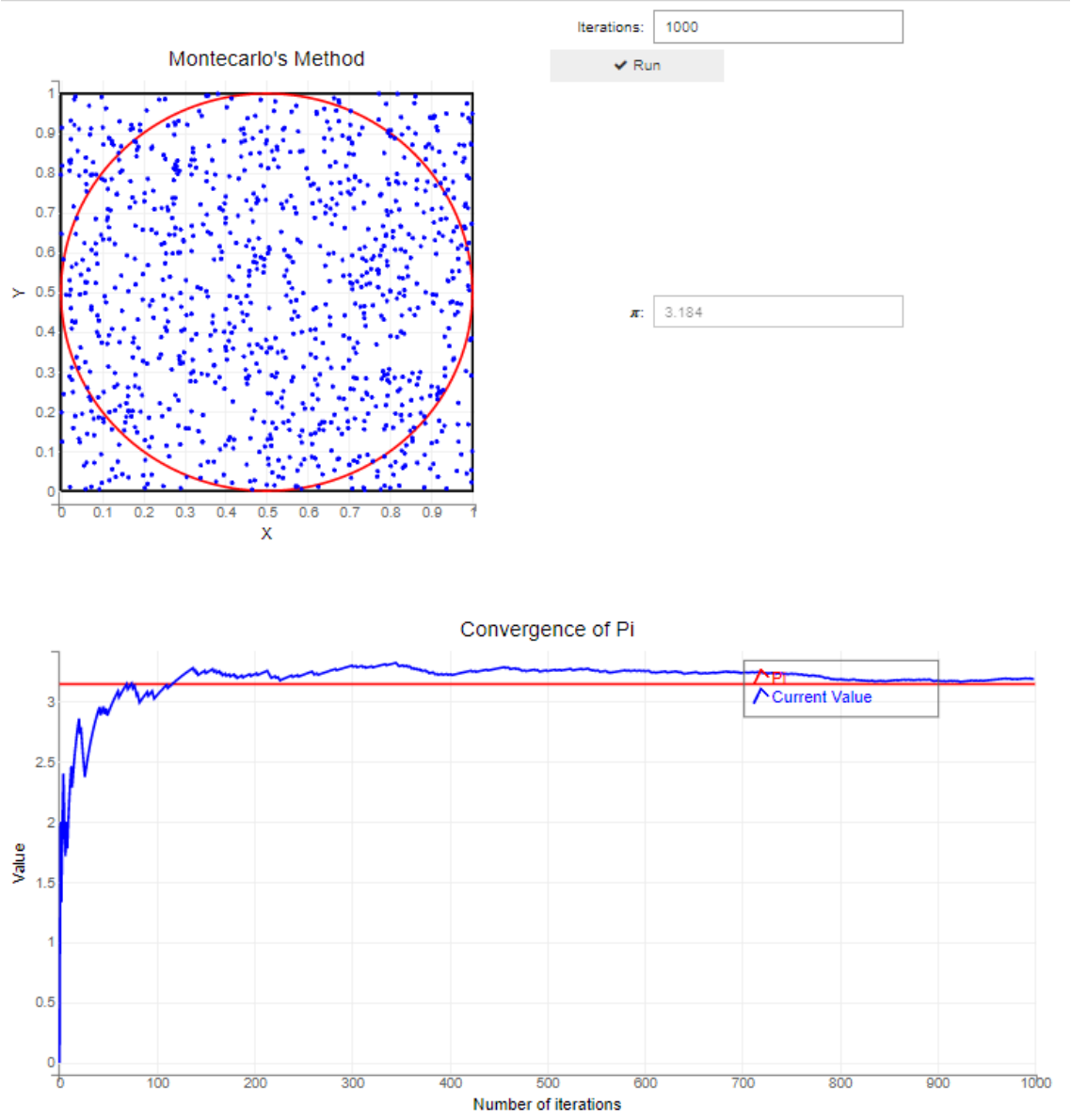


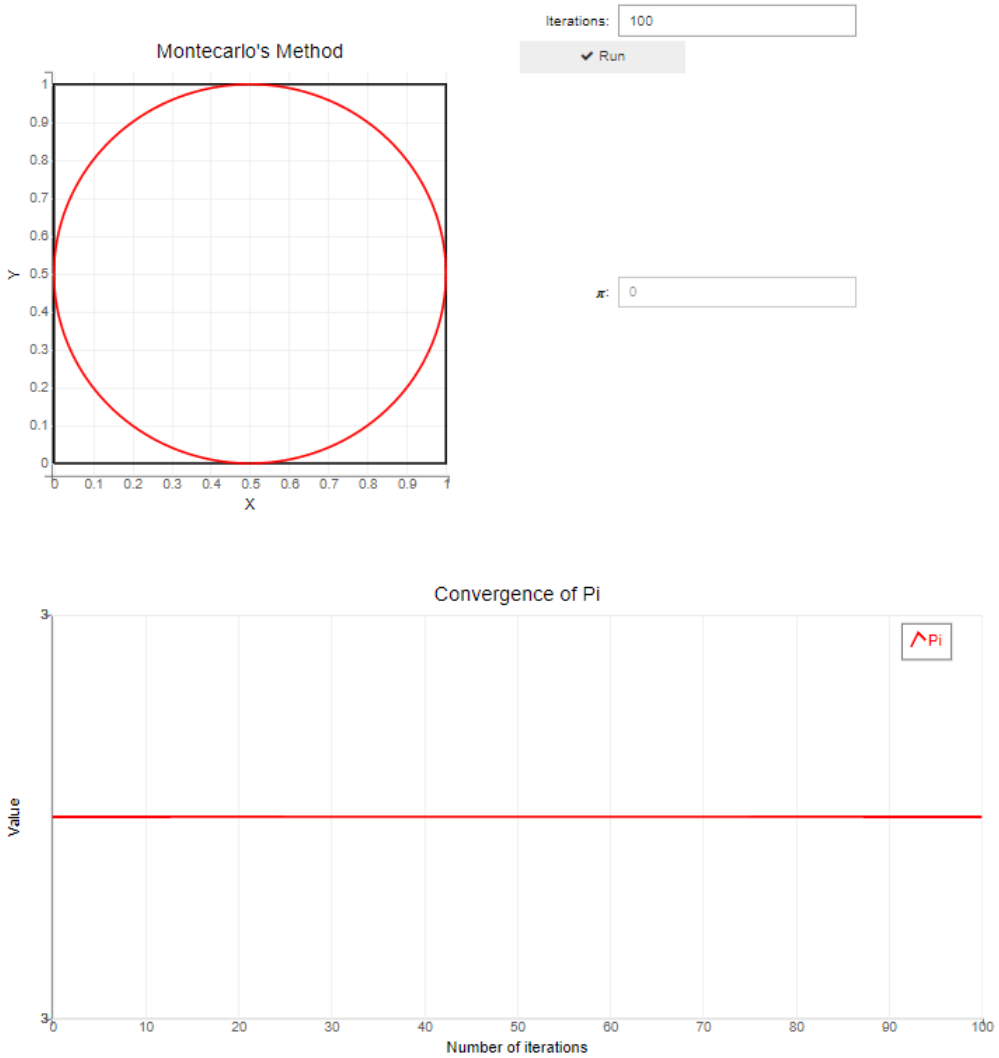
Fig. 6.46. BNumMet's Random Visualizer Example

Examples

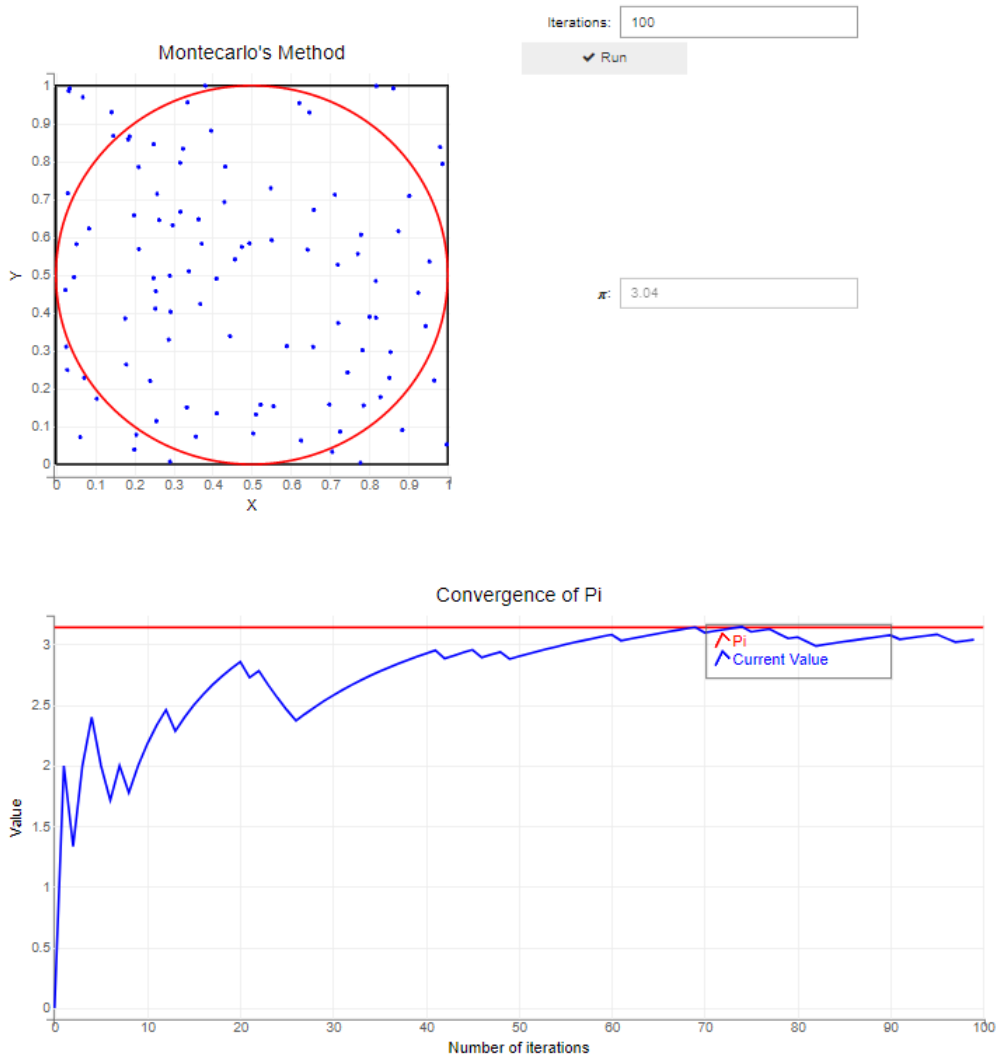
Example 1: No input arguments

```
from BNumMet.Visualizers.RandomVisualizer import
    RandomVisualizer
randomVisualizer = RandomVisualizer()
randomVisualizer.run()
```

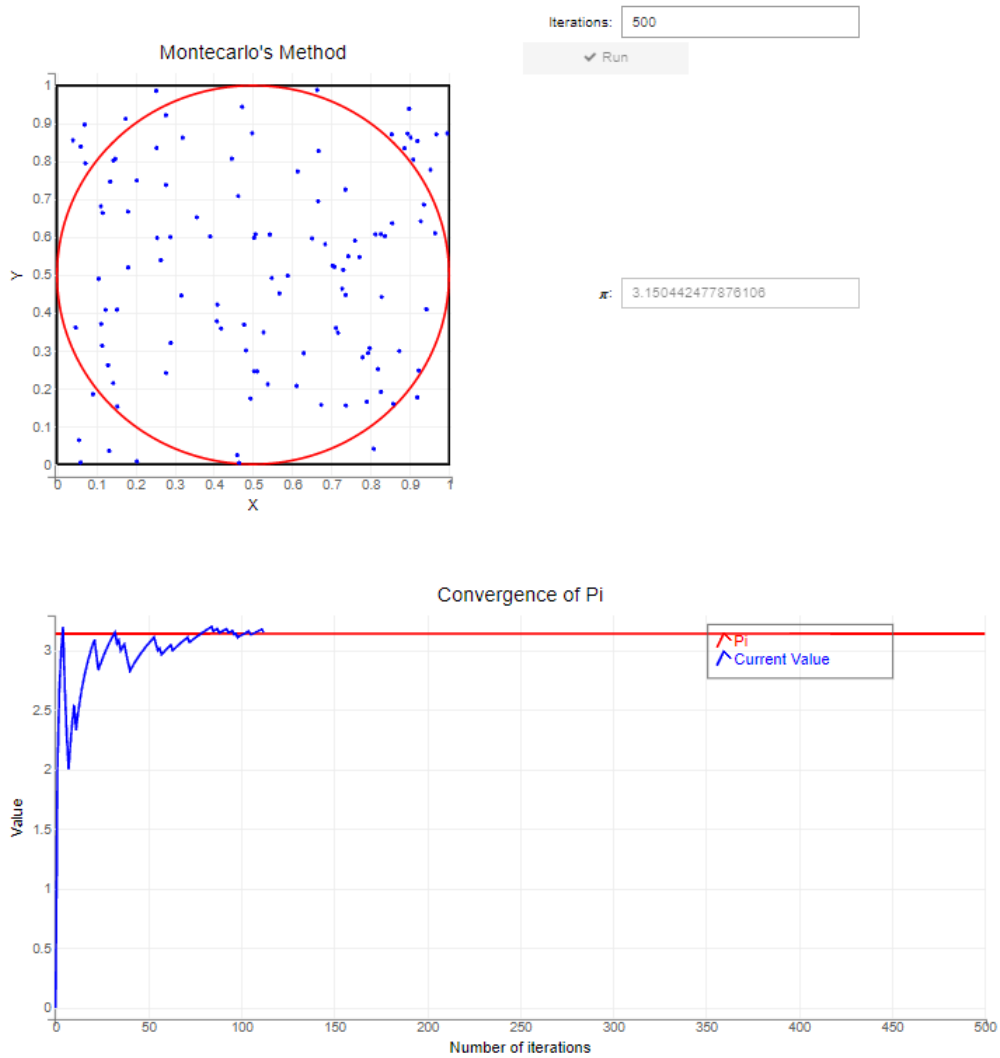
1. Initial State



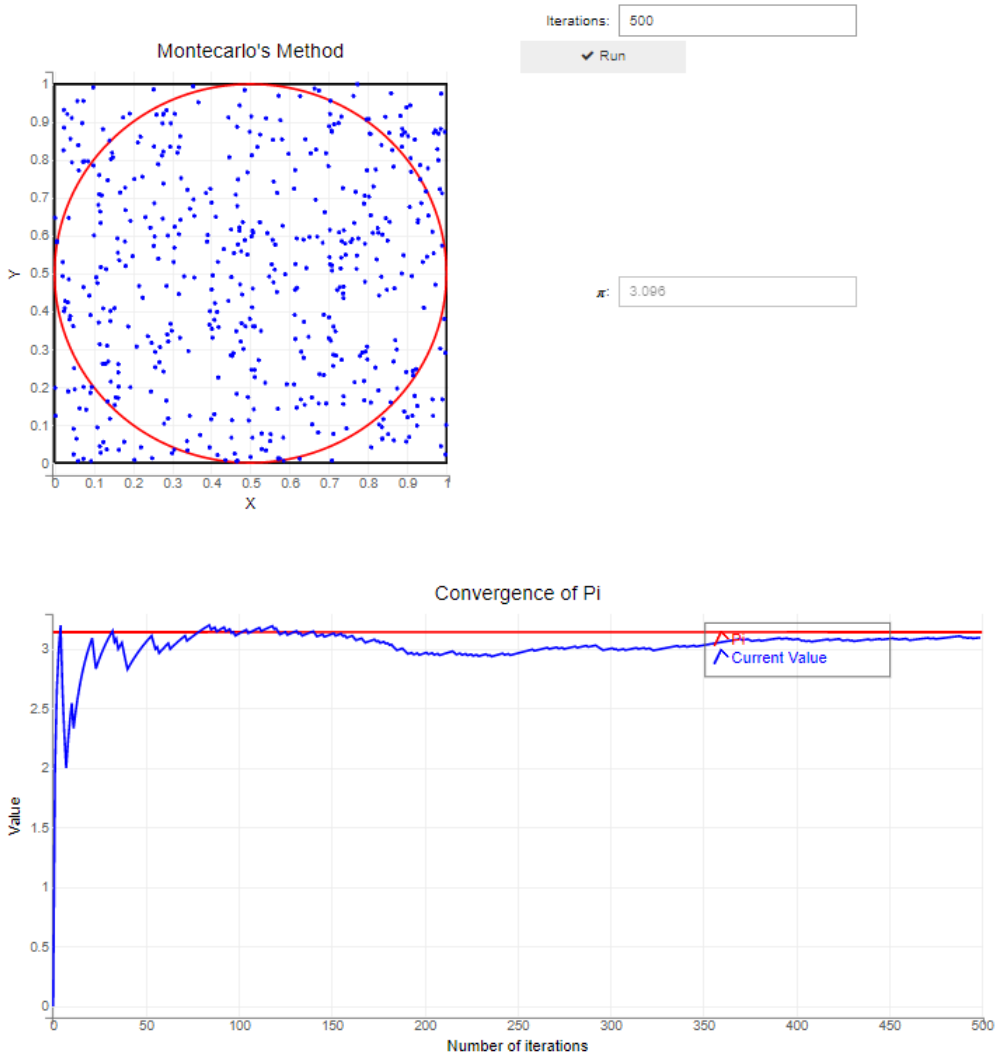
2. Clicked played with 100 default points



3. Change number of points to 500 and clicked play



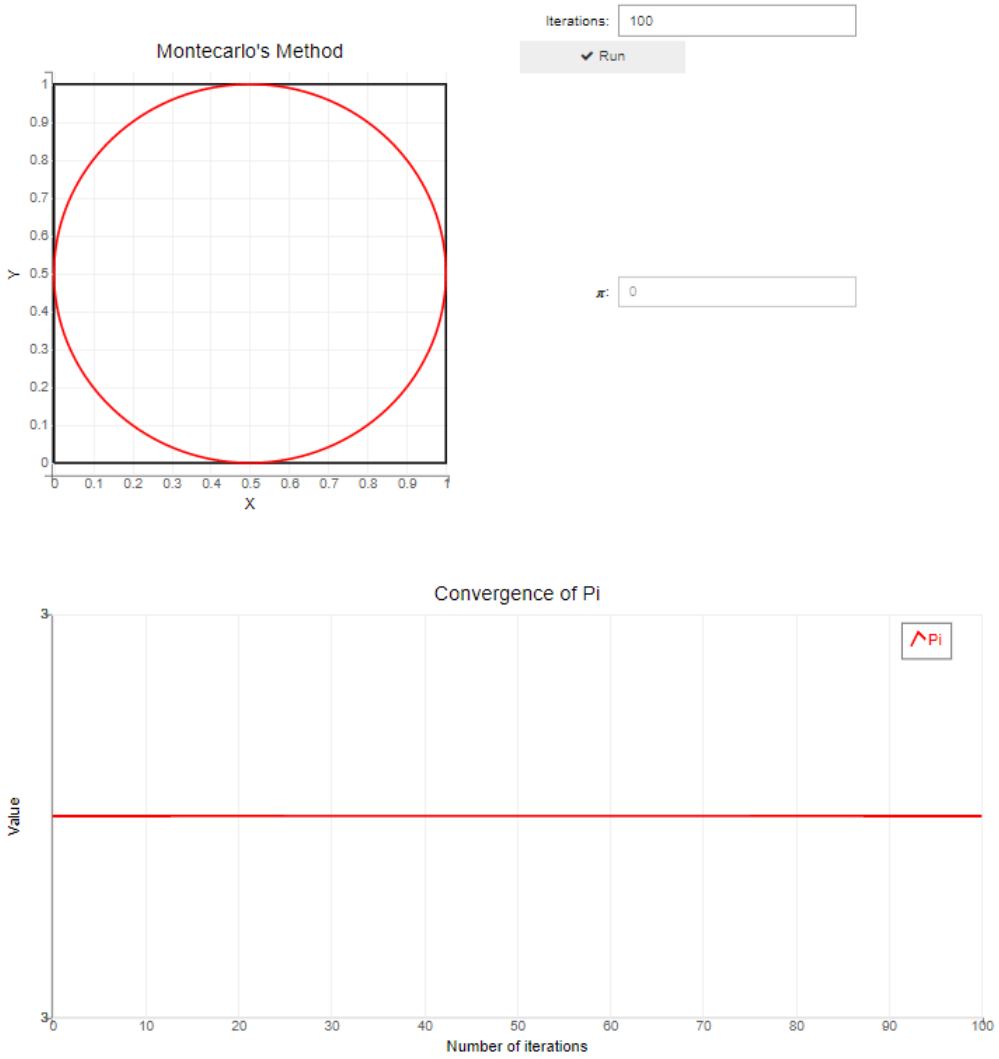
4. Final of animation playing



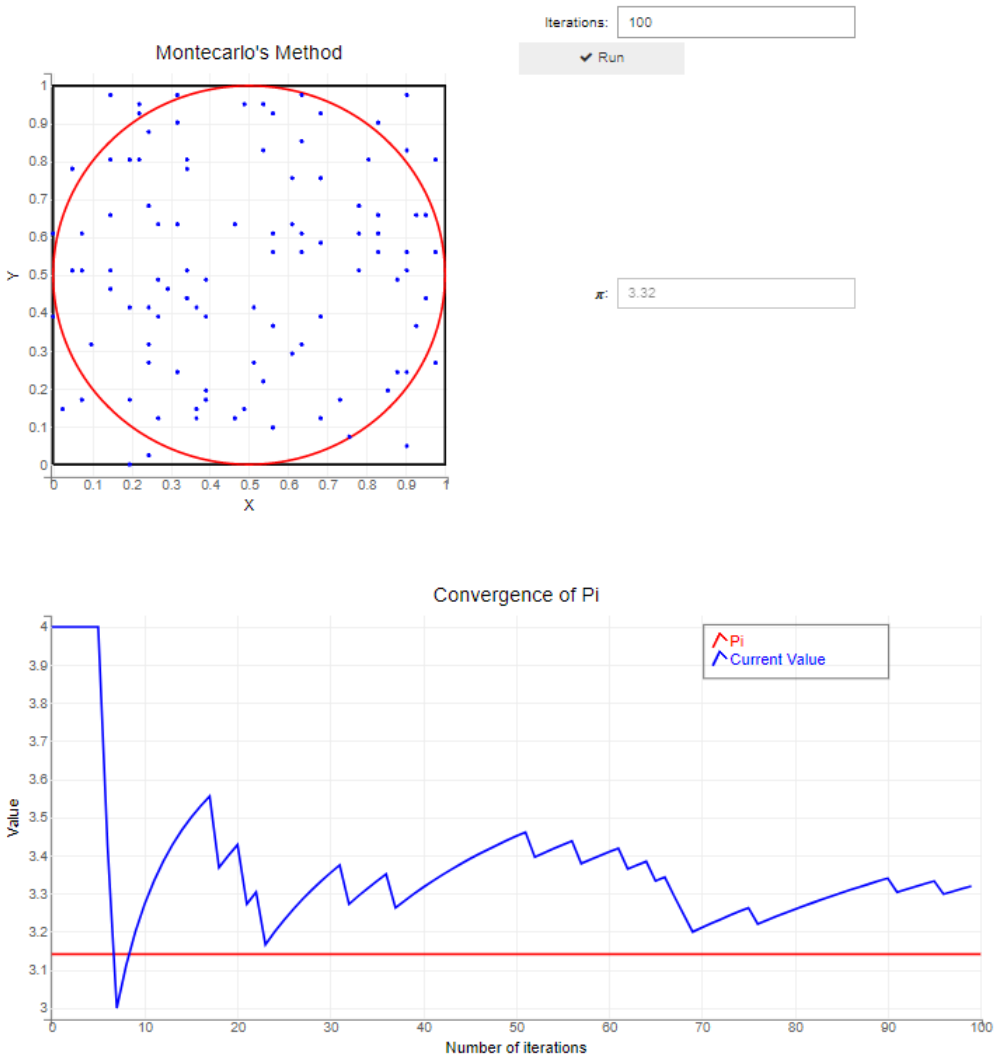
Example 2: Input arguments and Ill-Generator

```
from BNumMet.Visualizers.RandomVisualizer import
    RandomVisualizer
from BNumMet.Random import marsaglia_rand,
    clear_marsaglia_vars

clear_marsaglia_vars()
randomFunc = lambda: marsaglia_rand(
    base=10, lag_r=2, lag_s=1, carry=0, seed_tuple=(0, 1)
)
randomVisualizer = RandomVisualizer(randomFunc)
randomVisualizer.run()
```

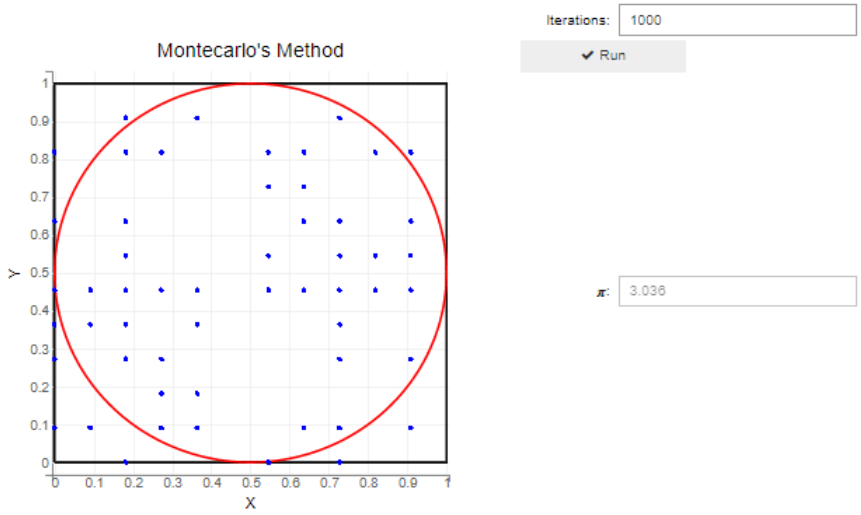
1. Initial State

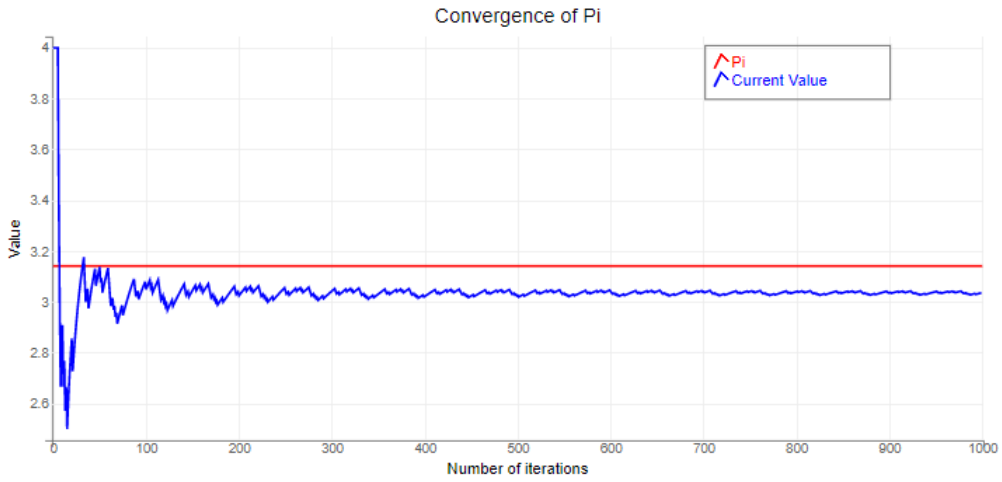


2. Clicked played with 100 default points



3. Change number of points to 1000 and clicked play


Montecarlo's Method
Iterations:
1000
Run
π:
3.036
Y
X


Convergence of Pi
Pi
Current Value
Value
Number of iterations

7. CONCLUSIONS

Overall, we have suggested a tool that is legible and comprehensible for students as well as a UI that enables them to comprehend the various numerical methods that they will use regularly in mathematics and computer science classes as well as in their professional jobs. In their professional jobs, students will typically use pre-implemented libraries, but they will be grown enough to understand why they are there and how they can be improved.

The presented answer is not only straightforward in nature, but has also undergone a series of programmatic and quality tests, as well as critical discussions about potential implementations, so we can ensure a degree of quality in what has been presented.

This endeavor, like any other, has space for further study and development. A viable extension of this work could, for example, delve deeply into the empirical substantiation of the tool by conducting a usage study with a group of students to assess the tool's effectiveness in aiding them in comprehending the numerical methods they are using. Additionally, a comparison study of the proposed tool with other current tools on the market to assess its strengths and weaknesses.

The suggested addition of new modules for numerical methods such as finite-difference methods and graph algorithms would be of interest in terms of expanding the tool's functionality and needing a thorough study of these numerical methods and how they can be applied in the tool.

Finally, a pedagogical analysis of the suggested tool could be performed to investigate how the tool can be used to improve the teaching and learning of numerical methods and computer science, finding possible barriers or challenges to adoption and suggesting solutions to overcome them. These study and development avenues will not only enhance the suggested tool, but will also benefit the broader academic and professional communities in mathematics and computer science.

BIBLIOGRAPHY

[1] C. G. Jung, *Volume 17 Collected Works of C.G. Jung, Volume 17*, G. Adler and R. F. Hull, Eds. Princeton: Princeton University Press, 1954. DOI: doi:10.1515/9781400850839. [Online]. Available: https://doi.org/10.1515/9781400850839.

[2] H. H. Goldstine, *A History of Numerical Analysis from the 16th through the 19th Century*. Springer Science & Business Media, 2012, vol. 2.

[3] C. B. Moler, *Numerical Computing with Matlab*. Society for Industrial and Applied Mathematics, 2004. DOI: 10.1137/1.9780898717952. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898717952. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898717952.

[4] *Tiobe index*, https://www.tiobe.com/tiobe-index/, Accessed: 04 Jun 2023.

[5] J. C. Bucheli Victoria, "Implementación de rutinas básicas de cálculo numérico de código libre con interfaz gráfica," End of Degree Thesis, Universidad Carlos III de Madrid, Madrid, Spain, Sep. 2020.

[6] S. Ruggiero, J. Zhao, and A. Ford Versypt, "Building a matlab graphical user interface to solve ordinary differential equations as a final project for an interdisciplinary elective course on numerical computing," *The Journal of Computational Science Education*, vol. 9, pp. 19–28, May 2018. DOI: 10.22369/issn.2153-4136/9/1/3.

[7] P. W. von Dohlen, "Coding and gui use in the teaching of undergraduate numerical analysis," English, *Electronic Journal of Mathematics and Technology*, vol. 14, pp. 107+, Jun. 2020, Report. [Online]. Available: https://link.gale.com/apps/doc/A673736920/AONE?u=anon~7526d07f&sid=googleScholar&xid=d197c9a1.

[8] P. Kosasih and K. Tieu, "Incorporating the development of a graphical user interface in courses teaching numerical methods for engineers," *International Journal of Mechanical Engineering Education*, vol. 35, pp. 46–55, Jan. 2007. DOI: 10.7227/IJMEE.35.1.3.

[9] B. D. Welfert and R. Aguilar, "Applied numerical methods and graphical visualization," *Computer Applications in Engineering Education*, vol. 4, no. 2, pp. 127–143, 1996.

[10] PMI, Ed., *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*, 5th ed. Newtown Square, PA: Project Management Institute, 2013.

[11] *PYPL Index*, https://pypl.github.io/PYPL.html, Accessed: 4th June 2023, 2023.

[12] *Self-managed | sonarqube | sonar - sonarsource*, https://www.sonarsource.com/products/sonarqube/, Accessed: 2023-04-03.

[13] *Sonarqube 9.9*, https://docs.sonarqube.org/latest/, Accessed: 2023-04-03.

[14] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, European. McGraw-Hill, 1994, Adapted by Darrel Ince.

[15] *A quick guide to gplv3 - gnu project - free software foundation*, https://www.gnu.org/licenses/quick-guide-gplv3.html, Accessed: 2022-04-02.

[16] *Gnu affero general public license - gnu project - free software foundation*, https://www.gnu.org/licenses/agpl-3.0.en.html, Accessed: 2022-04-02.

[17] *The fundamentals of the agplv3 - free software foundation*, https://www.fsf.org/bulletin/2021/fall/the-fundamentals-of-the-agplv3, Accessed: 2022-04-02.

[18] GitHub, *Choose an open source license*, https://choosealicense.com/, Accessed on 03/04/2023, 2023.

[19] O. S. Initiative, *Licenses by name*, https://opensource.org/licenses/, Accessed on 03/04/2023, 2023.

[20] Wikipedia contributors, *Docker (software) — Wikipedia, the free encyclopedia*, [Online; accessed 16-April-2023], 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Docker_(software)&oldid=1148220158.

[21] E. Anderson *et al.*, *LAPACK Users' Guide*, Third. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1999.

[22] F. D. T. Vergara, *Computational and Applied Linear Algebra*. Avenida de la Universidad 30, 28911 Leganes, Spain, Course notes.

[23] B. Archer and E. W. Weisstein, *Lagrange interpolating polynomial*, https://mathworld.wolfram.com/LagrangeInterpolatingPolynomial.html, Accessed: 2023-06-06.

[24] Mathworks, *Cubic spline interpolation*, https://es.mathworks.com/help/curvefit/construct-cubic-spline-interpolants.html, Accessed: 2023-06-06.

[25] F. N. Fritsch and R. E. Carlson, "Monotone piecewise cubic interpolation," *SIAM Journal on Numerical Analysis*, vol. 17, no. 2, pp. 238–246, 1980. DOI: 10.1137/0717021. eprint: https://doi.org/10.1137/0717021. [Online]. Available: https://doi.org/10.1137/0717021.

[26] *Pythonspeed/performancetips - python wiki*, https://wiki.python.org/moin/PythonSpeed/PerformanceTips#Local_Variables, Accessed: 2023-04-04.

[27] J. F. Epperson, *An Introduction to Numerical Methods and Analysis*, 2nd. Wiley Publishing, 2013.

[28] E. Kreyszig, H. Kreyszig, and E. J. Norminton, *Advanced Engineering Mathematics*, Tenth. Hoboken, NJ: Wiley, 2011.

[29] R. Brent, *Algorithms for minimization without derivatives*. Prentice-Hall, 1973.

[30] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[31] W. Payne, J. R. Rabung, and T. Bogyo, "Coding the lehmer pseudo-random number generator," *Communications of the ACM*, vol. 12, no. 2, pp. 85–86, 1969.

[32] S. K. Park and K. W. Miller, "Random number generators: Good ones are hard to find," *Communications of the ACM*, vol. 31, no. 10, pp. 1192–1201, 1988.

[33] G. Marsaglia and A. Zaman, "A New Class of Random Number Generators," *The Annals of Applied Probability*, vol. 1, no. 3, pp. 462–480, 1991. DOI: 10.1214/aoap/1177005878. [Online]. Available: https://doi.org/10.1214/aoap/1177005878.

[34] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator," *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 8, no. 1, pp. 3–30, 1998.

[35] E. B. Smid, S. Leigh, M. Levenson, M. Vangel, A. DavidBanks, and S. JamesDray, "A statistical test suite for random and pseudorandom number generators for cryptographic applications," *Her research interest includes Computer security, secure operating systems, Access control, Distributed systems, Intrusion detection systems*, 2010.

[36] InsaneMonster, *Nistrng*, https://github.com/InsaneMonster/NistRng, 2022.

[37] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C (2nd Ed.): The Art of Scientific Computing*. USA: Cambridge University Press, 1992.

APPENDIX: DOCUMENTATION

General Overview

What is BNumMet?

BNumMet ($'/bi : num \, m\varepsilon t/'$) is short for Basic Numerical Methods, it is a self-contained library to provide students a scholarly implementation of numerical methods alongside a visual interface that captures the essence of the methods explained.

The intend and purpose of this library is to provide students an introduction to both Python and numerical methods that will serve for their future in the academic and enterprise world. It uses NumPy, as students will find it in their daily life, while using numerical methods in Python.

How to install?

There are two main ways to install the package, using the pypi package installer or manual installation

PyPi Package Manager

Since the package is publicly available in the PyPi webpage (https://pypi.org/project/BNumMet/), we can use the 'pip' command.

Assuming a correct installation of Python and/or pip, the following command will install all dependencies and the package:

```
pip install BNumMet
```

Manual Installation

Alternatively, you can download the repository and install the package manually. To do so, the following manual installation instructions will be proposed:

1. Clone the repository: https://github.com/fbpazos/Trabajo-Fin-Master, there is two ways, using git and a manual cloning
   (a) Using git: `git clone https://github.com/fbpazos/Trabajo-Fin-Master`

(b) Manual cloning: Click on https://github.com/fbpazos/Trabajo-Fin-Master/archive/refs/heads/main.zip, this will download a zip file with the latest version. Extracting this will provide the cloning.

2. Install using Python: Once cloned, "cd" into the folder named as "Python_BNumMet", and write in a terminal one of this two options:
   (a) Using pure Python: `python setup.py install`
   (b) Using pip locally: `pip install .`

How to continue development?

If anyone desires to continue the development, respecting the license provided, we recommend the use of virtual environments to externalize the current installation of other libraries when developping BNumMet.

1. Clone the repository: https://github.com/fbpazos/Trabajo-Fin-Master, there is two ways, using git and a manual cloning
   (a) Using git: `git clone https://github.com/fbpazos/Trabajo-Fin-Master`

   (b) Manual cloning: Click on https://github.com/fbpazos/Trabajo-Fin-Master/archive/refs/heads/main.zip, this will download a zip file with the latest version. Extracting this will provide the cloning.
2. (Optional) Create the virtual enviroment and activate it: Once cloned, "cd" into the folder named as "Python_BNumMet", proceed with the following commands
   (a) Using CMD: `python3 -m venv venv && source venv/bin/activate`
   (b) Using Bash: `python -m venv venv && venv\Scripts\activate`
3. Install the package in editable mode: In contrast to normally installing the library as we aforementioned, editable mode allows us to make changes to the library and those changes will automatically be updated into the python installation, to properly install it using this mode: `pip install -e .` , the "-e" indicate editable, it could also be written as `--editable`

When continuing development, make sure to add tests to new/old functions as well as passing a SonarQube's analysis, therefore we assure good-quality standards to the students.

Run tests

In order to properly run the tests, we recommend using one of the following commands:

- Vanilla Pytest: Inside the folder of the project run `pytest`.
- Pytest with Black-Formatting and Coverage: Inside the project folder run `python tests/__init__.py`.

Library Structure Brief

BNumMet.

LinearSystems.

- `lu(matrix)` → P,L,U matrices as `np.array`
- `permute(matrix, row_1, row_2)` → Permuted Matrix as `np.array`
- `forward_substitution(matrix_L, matrix_b)` → Solution to Lx=b as `np.array`
- `backward_substitution(matrix_U, matrix_b)` → Solution to Ux=b as `np.array`
- `lu_solve(matrix_A, matrix_b)` → Solution to Ax=b as `np.array` using LU Decomposition
- `qr_factorization(matrix_A)` → Q,R Matrices as `np.array`
- `qr_solve(matrix_A, matrix_b)` → Solution to Ax=b as `np.array` using QR decomposition
- `interactive_lu(matrix_p, matrix_l, matrix_u, col, row, pivot_row)` → An iteration of LU Decomposition

Interpolation.

- `polynomial(interpolation_x, interpolation_y, mesh)` → Polynomial-Interpolated values over mesh
- `piecewise_linear(interpolation_x, interpolation_y, mesh)` → Piecewise Linear-Interpolated values over mesh
- `pchip(interpolation_x, interpolation_y, mesh)` → Piecewise Cubic Hermite-Interpolated values over mesh
- `splines(interpolation_x, interpolation_y, mesh)` → Piecewise Cubic-Interpolated values over mesh

NonLinear.

- `bisect(function, interval:tuple, stop_iters:int, iters:bool, *args)` → x-value of where the zero is at and, as optional, the number of iterations taken

- `secant(function, interval:tuple, stop_iters:int, iters:bool, *args)` → x-value of where the zero is at and, as optional, the number of iterations taken
- `newton(function, derivative, interval:tuple, stop_iters:int, iters:bool, *args)` → x-value of where the zero is at and, as optional, the number of iterations taken
- `IQI(function, values_of_x:tuple, stop_iters:int, iters:bool, *args)` → x-value of where the zero is at and, as optional, the number of iterations taken
- `zBrentDekker(function, interval:tuple, tol, stop_iters:int, iters:bool, *args)` → x-value of where the zero is at and, as optional, the number of iterations taken

Random.

- `clear_lehmers_vars()` → Cleans the initiated values of the Lehmers random number generator
- `lehmers_init(a, c, m, x)` → Initializes Lehmers R.N.G. with values given
- `lehmers_rand(a, c, m, x)` → Initializes and produces a random number every time it is called
- `clear_marsaglia_vars()`→ Cleans the initiated values of the Marsaglia's random number generator
- `marsaglia_init(base, lag_r, lag_s, carry, seed_tuple)` → Initializes Marsaglia's R.N.G. with values given
- `marsaglia_rand(base, lag_r, lag_s, carry, seed_tuple)` → Initializes and produces a random number every time it is called
- `clear_mt_vars()` → Cleans the initiated values of the Mersenne Twister random number generator
- `sgenrand(seed:int)` → Initializes and produces a random number every time it is called

Visualizers.

Every class can be executed after initializing the class constructor and using the ".run()" method withing a jupyter cell

```
variable = Class()
varibal.run()
```

- **LUVisualizer.**

  - LUVisualizer:Class
- **InterpolationVisualizer.**
  - InterpolVisualizer:Class
- **NonLinearVisualizer.**
  - NonLinearVisualizer:Class
- **LeastSquaresVisualizer.**
  - LSPVisualizer:Class
- **RandomVisualizer.**
  - RandomVisualizer:Class

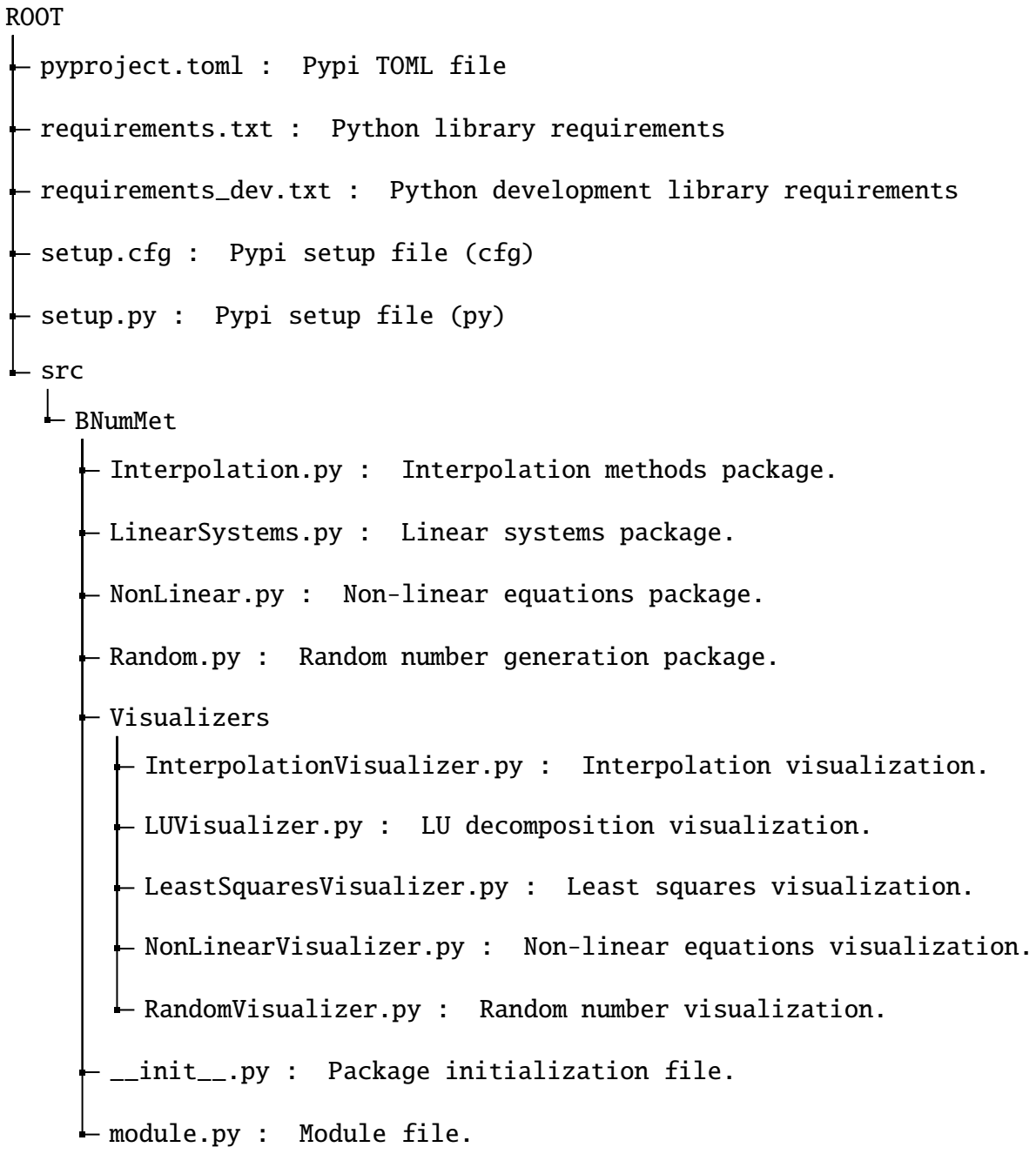
Folder Structure

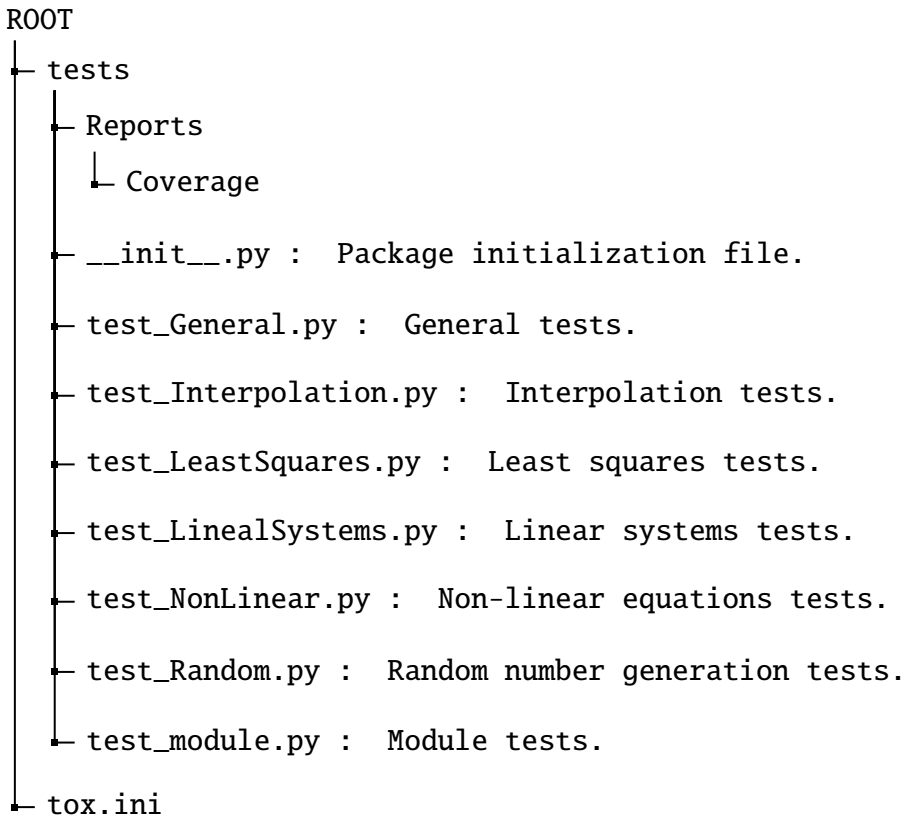
From now on, we will draw our attention to the folder 'Python_BNumMet'
and we will tear down the main structure we have followed.

```
ROOT
├─ Demos :  Contains Jupyter Examples
│  └─ Timings :  Python files for timing or analysis of the methods
│     └─ Results :  Results from Timings files
│        ├─ Interpolation
│        ├─ Linear Systems
│        └─ NonLinear
├─ Report :  Reports produced by SonarQube
├─ Utilities :  Utility programs
├─ src :  Source code
│  └─ BNumMet :  Main
│     └─ Visualizers :  Python files for the visualizers
└─ tests :  Containts python tests
   └─ Reports :  Tests reports generated by pytest
      └─ Coverage :  Tests coverage generated by pytest
         ├─ html :  html output files
         ├─ lcov :  lcov output files
         └─ xml :  xml output files
```

Files Structure

```
ROOT
├─ Demos
│  ├─ Interpolation.ipynb
│  ├─ LinearSystems.ipynb
│  ├─ NonLinear.ipynb
│  ├─ Packages Show.ipynb
│  ├─ Randomness.ipynb
│  └─ Timings
│     ├─ Interpolation Timings.py
│     ├─ Interpolation_Timings_Analysis.ipynb
│     ├─ LU_Timings_Analysis.ipynb
│     ├─ Linear Systems Timings.py
│     ├─ NonLinear Timings.py
│     ├─ NonLinear_Iterations.ipynb
│     └─ Results :  Folder for results generated by Timings files
├─ LICENSE
├─ VERSION
├─ MANIFEST.in
├─ Readme.md
├─ Report :  Directory of SonarQube's Reports
└─ Utilities
   ├─ ReportGenerator.jar :  Generates a report of SonarQube
   ├─ SonarScanner.bat :  Quality testing automation (Windows)
   ├─ SonarScanner.sh :  Quality testing automation (Linux)
   ├─ ngrok.exe :  Tunnel for remote developing
   └─ sonarqubeRemote.bat :  Remote development automation
```


ROOT
pyproject.toml : Pypi TOML file
requirements.txt : Python library requirements
requirements_dev.txt : Python development library requirements
setup.cfg : Pypi setup file (cfg)
setup.py : Pypi setup file (py)
src
BNumMet
Interpolation.py : Interpolation methods package.
LinearSystems.py : Linear systems package.
NonLinear.py : Non-linear equations package.
Random.py : Random number generation package.
Visualizers
InterpolationVisualizer.py : Interpolation visualization.
LUVisualizer.py : LU decomposition visualization.
LeastSquaresVisualizer.py : Least squares visualization.
NonLinearVisualizer.py : Non-linear equations visualization.
RandomVisualizer.py : Random number visualization.
__init__.py : Package initialization file.
module.py : Module file.


ROOT
tests
Reports
Coverage
__init__.py : Package initialization file.
test_General.py : General tests.
test_Interpolation.py : Interpolation tests.
test_LeastSquares.py : Least squares tests.
test_LinealSystems.py : Linear systems tests.
test_NonLinear.py : Non-linear equations tests.
test_Random.py : Random number generation tests.
test_module.py : Module tests.
tox.ini

Linear Systems

LU Decomposition

The LU Method divides a matrix A into three matrices: P (Permutation Matrix), L (Lower Triangular Matrix), and U (Upper Triangular Matrix).

Algorithm 2: LU decomposition using Gaussian elimination

Input: A square matrix A

Output: A permutation matrix P, lower triangular matrix L, and upper triangular matrix U

if *A is not square* **then**
    raise ValueError("Matrix must be square")
end
$n \leftarrow$ number of rows/columns of A
$A' \leftarrow$ a copy of A converted to float
$P \leftarrow$ identity matrix of size n
for $col \leftarrow 1$ **to** n **do**
    $maximum\_index \leftarrow$ index of row with largest pivot element in column col
    if $A'[maximum\_index, col] \neq 0$ **then**
        swap rows col and $maximum\_index$ in A' and P
        for $row \leftarrow col + 1$ **to** n **do**
            $multiplier \leftarrow A'[row, col]/A'[col, col]$
            $A'[row, col{+}1 : n] \leftarrow A'[row, col{+}1 : n] - multiplier \cdot A'[col, col{+}1 : n]$
            $A'[row, col] \leftarrow multiplier$
        end
    end
end
$L \leftarrow$ lower triangular part of A' with ones on the diagonal
$U \leftarrow$ upper triangular part of A'
return *P, L, U*

BNumMet Examples

Example 1

```
from BNumMet.LinearSystems import lu
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
P, L, U = lu(A)
display(P, L, U)

>> P = array([ [1., 0., 0.],
               [0., 0., 1.],
               [0., 1., 0.]])
>> L = array([ [ 1.  ,  0.  ,  0.  ],
```

```
                [ 0.5 ,  1.  ,  0.  ],
                [-0.3 , -0.04,  1.  ]])
>> U = array([ [10. , -7. ,  0. ],
                [ 0. ,  2.5,  5. ],
                [ 0. ,  0. ,  6.2]])
```

Forward Substitution

The forward substitution method implementets the necessary steps that are used to solve a system of linear equations of the form $Lx = b$, where L is a lower triangular matrix and b is a vector of constants.

Algorithm 3: Forward Substitution

Input : *lhs*: np.array, a lower triangular matrix
rhs: np.array, a vector

Output: *x*: np.array, a vector

```
lhs ← lhs.astype(float);
rhs ← np.copy(rhs).astype(float);
n ← lhs.shape[0];
x ← np.zeros(n);
for row ← 0 to n − 1 do
    rhs[row] ← rhs[row] - dot(lhs[row, :row], x[:row]);
    x[row] ←rhs[row] / lhs[row, row];
return x;
```

BNumMet Examples

Example 1

```
from BNumMet.LinearSystems import lu, forward_substitution
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
P, L, U = lu(A)
solution_aux = forward_substitution(L, P @ b)
display(solution_aux)

>> solution_aux = array([7. , 2.5, 6.2])
```

Backward substitution

The backward substitution method implementets the necessary steps that are used to solve a system of linear equations of the form $Ux = b$, where U is an upper triangular matrix and b is a vector of constants.

Algorithm 4: Backward Substitution

Input: lhs: an upper triangular matrix, rhs: a vector

Output: x: a vector

Function *backward_substitution(lhs, rhs)*:
    lhs ← lhs.astype(float);
    rhs ← np.copy(rhs).astype(float);
    n ← lhs.shape[0];
    x ← np.zeros(n);
    for *row* ← *n-1* **to** *0* **do**
        rhs[*row*] ← rhs[row] - np.dot(lhs[row, row+1:], x[row+1:]);
        x[*row*] ← rhs[row] / lhs[row, row];
    end
    return *x*;
end

BNumMet Examples

Example 1

```
from BNumMet.LinearSystems import lu, forward_substitution,
    backward_substitution
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
P, L, U = lu(A)
solution_aux = forward_substitution(L, P @ b)
solution = backward_substitution(U, solution_aux)
display(solution)

>> solution = array([ 0., -1.,  1.])
```

LU Solver

The LU Solver implements all the required steps to solve a system of linear equations of the form $Ax = b$, where A is a square matrix and b is a vector of constants. The LU Solver uses the LU Decomposition, Forward Substitution, and Backward Substitution methods to solve the system of linear equations.

Algorithm 5: LU Solver

Input: A: a square matrix, b: a vector

Output: x: a vector

Function *lu_solve(A, b)*:
    $P, L, U \leftarrow$ lu(A);
    $y \leftarrow$ forward_substitution(L, P $\times$ b);
    $x \leftarrow$ backward_substitution(U, y);
    return x;
end

BNumMet Examples

Example 1

```
from BNumMet.LinearSystems import lu_solve
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
solution = lu_solve(A, b)
display(solution)

>> solution = array([ 0., -1.,  1.])
```

QR Decomposition

The QR Decomposition method implements the necessary steps to decompose a matrix A into the product of an orthogonal matrix Q and an upper triangular matrix R. The QR Decomposition method uses the Householder Reflections method to compute the orthogonal matrix Q and the upper triangular matrix R.

Algorithm 6: QR Factorization using Householder reflections

Input: A: an $m \times n$ matrix

Output: Q: an orthogonal matrix, R: an upper triangular matrix

$m, n \leftarrow \text{shape}(A)$;
$R \leftarrow A$;
$\text{qs} \leftarrow \text{zeros}(m, n)$;
for $k \in [0, n)$ **do**
    $x \leftarrow R_{k:,k}$;
    $q_k \leftarrow \text{sign}(x_0) \times \text{norm}(x) \times I_{x,1} + x$;
    $q_k \leftarrow q_k / \text{norm}(q_k)$;
    $R_{k:,k:} \leftarrow R_{k:,k:} - 2q_k(q_k^T R_{k:,k:})$;
end
$Q \leftarrow I_m$;
for $k \in [n-1, -1]$ **do**
    $q_k \leftarrow \text{qs}_{k:,k}$;
    $Q_{k:,k:} \leftarrow Q_{k:,k:} - 2q_k(q_k^T Q_{k:,k:})$;
end
$R \leftarrow \text{triu}(R)$;
return Q, R;

BNumMet Examples

Example 1

```
from BNumMet.LinearSystems import qr_factorization
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
Q, R = qr_factorization(A)
display(Q,R)

>> Q  =
    array([[-0.86386843,  0.42560398,  0.26943013],
           [ 0.25916053, -0.08312578,  0.96225045],
           [-0.43193421, -0.90108343,  0.03849002]])
>> R =
    array([[-11.5758369 ,   6.99733425,  -0.6047079 ],
           [  0.        ,  -2.24439601,  -5.00417184],
           [  0.        ,   0.        ,   5.96595278]])
>> Q@R =
    array([[ 1.00000000e+01, -7.00000000e+00, -4.67352964e
        -16],
           [-3.00000000e+00,  2.00000000e+00,  6.00000000e
               +00],
           [ 5.00000000e+00, -1.00000000e+00,  5.00000000e
               +00]])
```

QR Solver

The QR Solver method implements the necessary steps to solve a system of linear equations of the form $Ax = b$, where A is a square matrix and b is a vector of constants. The QR Solver method uses the QR Decomposition with the fact that we are solving a system of linear equations.

Algorithm 7: QR Solver

Input: A: a square matrix, b: a vector
Output: x: a vector

$R \leftarrow$ copy of A as float matrix;
$b \leftarrow$ copy of b as float vector;
$n \leftarrow$ number of rows in A;
for $k \leftarrow 0$ **to** $n - 1$ **do**
    $x \leftarrow k$-th column of R starting from k-th row;
    $q_k \leftarrow$ Householder vector of x;
    $R_{k:,k:} \leftarrow R_{k:,k:} - 2q_k(q_k^T R_{k:,k:})$;
    $b_{k:} \leftarrow b_{k:} - 2q_k(q_k^T b_{k:})$;
end
$R \leftarrow$ upper triangular part of R;
$x \leftarrow$ backward substitution$(R[: n, : n], b[: n])$
return x

BNumMet Examples

Example 1

```
from BNumMet.LinearSystems import qr_solve
A = np.array([[10, -7, 0], [-3, 2, 6], [5, -1, 5]])
b = np.array([7, 4, 6])
solution = qr_solve(A, b)
display(solution)

>> solution =
        array([-3.83634647e-16, -1.00000000e+00,  1.00000000e
           +00])
```

Interpolation

Polinomial Interpolation

The polinomial interpolation function interpolates a set of points (x_i, y_i) with a polinomial of degree n.

Algorithm 8: Polinomial Interpolation

Input: x: list of x coordinates, y: list of y coordinates, u: list of points where the interpolation is computed

Output: v: list of values of the interpolation at the points u

$n \leftarrow$ length of x;
$v \leftarrow$ zeros vector with length of u;
for $i \leftarrow 0$ **to** $n - 1$ **do**
    $w \leftarrow$ ones vector with length of u;
    for $j \leftarrow 0$ **to** $n - 1$ **do**
        if $j \neq i$ **then**
            $w \leftarrow w * (u - x_j)/(x_i - x_j)$;
        end
    end
    $v \leftarrow v + w * y_i$
end
return v

BNumMet Examples

Example 1

```
from BNumMet.Interpolation import polinomial
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
# Plotting using Matplotlib
v = polinomial(x, y, u)
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("Polinomial Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

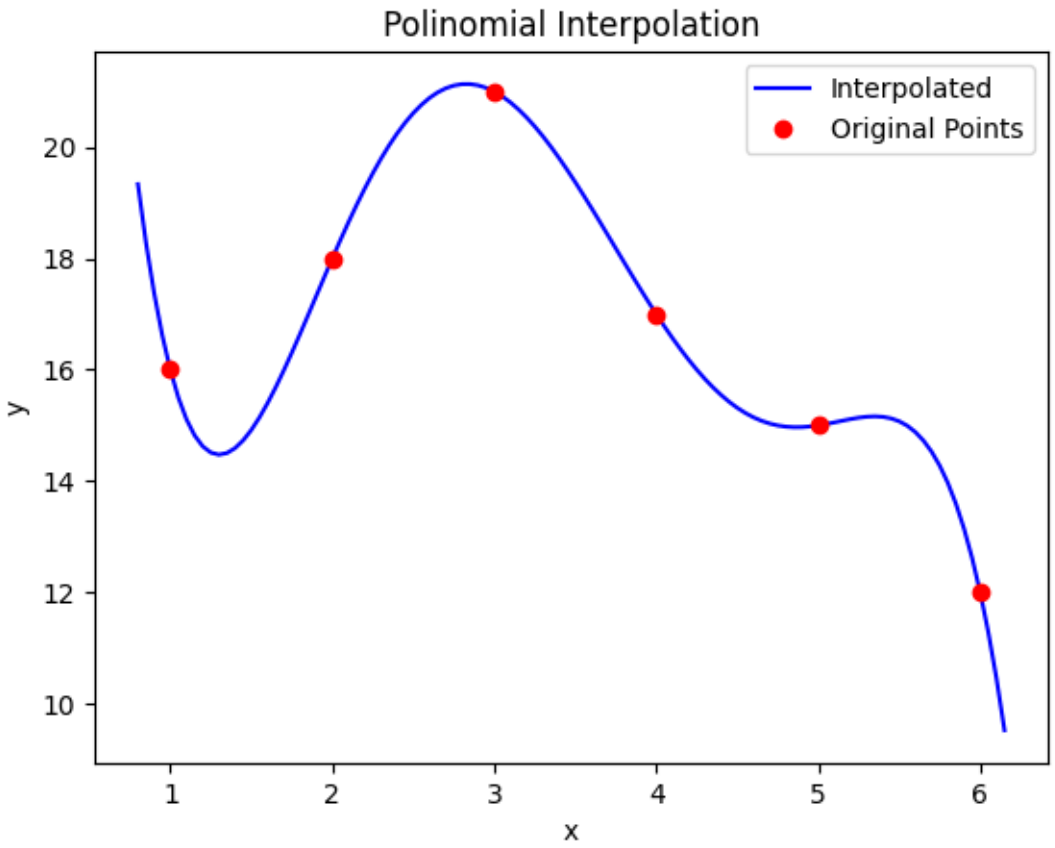


Fig. 7.1. Polinomial Linear Example 1

Piecewise Linear Interpolation

The piecewise linear interpolation function interpolates a set of points using a piecewise linear functions.

Algorithm 9: Piecewise Linear Interpolation

Input: x: list of x coordinates, y: list of y coordinates, u: list of points where the interpolation is computed, *sorted*: if the points are sorted or not (default: False)

Output: v: list of values of the interpolation at the points u

if *not sorted* **then**
    $x \leftarrow$ numpy array of x;
    $y \leftarrow$ numpy array of y;
    $ind \leftarrow$ indices of sorted x;
    $x \leftarrow x[ind]$;
    $y \leftarrow y[ind]$;
end
$\Delta \leftarrow \text{diff}(y)/\text{diff}(x)$;
$n \leftarrow$ length of x;
$k \leftarrow$ zeros array with size of u, dtype=int;
for $j \leftarrow 1$ **to** $n - 2$ **do**
    $k[x_j <= u] \leftarrow j$
end
$s \leftarrow u - x_k$
$v \leftarrow y_k + s * \Delta_k$
return v

BNumMet Examples

Example 1

```
from BNumMet . Interpolation import piecewise_linear
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
v = piecewise_linear(x, y, u)
# Plotting using Matplotlib
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("Piecewise Linear Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

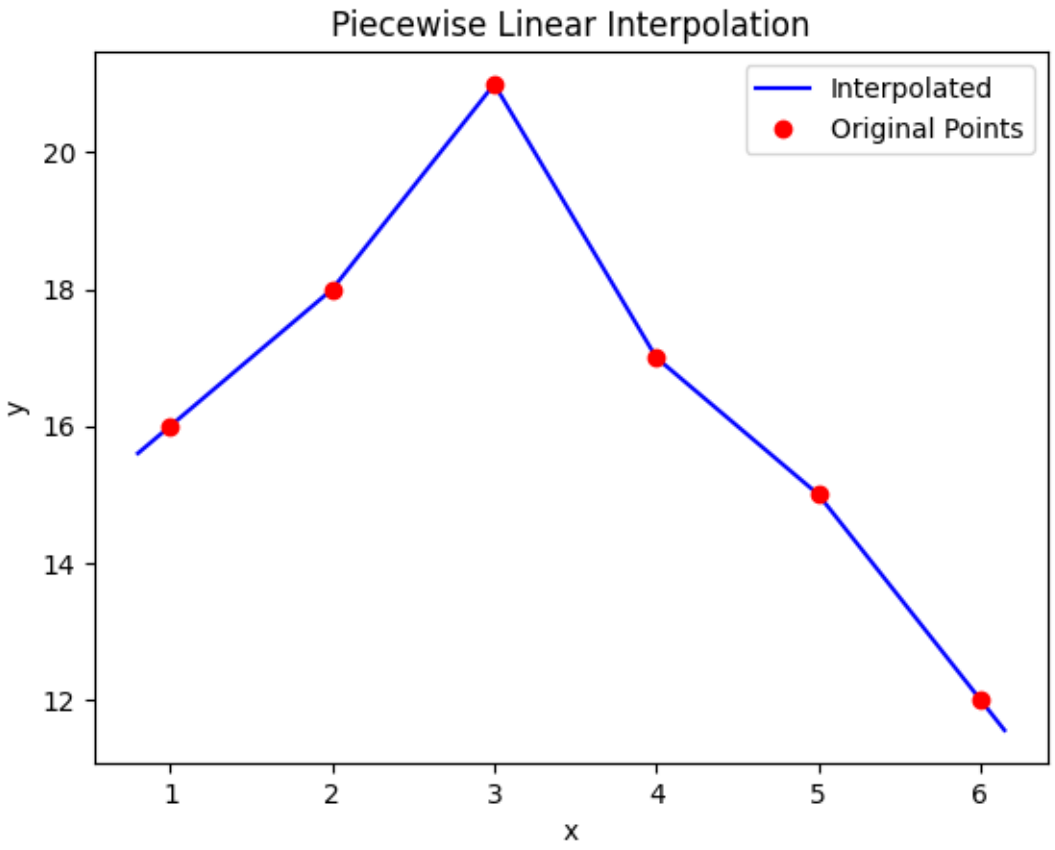


Fig. 7.2. PieceWise Linear Example 1

Piecewise Cubic Hermite Interpolation Polynomial (P.C.H.I.P.)

The pchip function is a Python implementation of the Piecewise Cubic Hermite Interpolation Polynomial (P.C.H.I.P.) based on an old Fortran program by Fritsch and Carlson [25].

Algorithm 10: Piecewise Cubic Hermite Interpolation

Input: List of x coordinates x, list of y coordinates y, list of points where the interpolation is computed u, and parameter to specify if the points are already sorted *sorted*

Output: List of values of the interpolation at the points u

$n \leftarrow$ length of x;
$h \leftarrow x[1:n] - x[0:n-1]$;
$\delta \leftarrow (y[1:n] - y[0:n-1])/h$;
$d \leftarrow$ pchip_slopes(h, δ);
$v \leftarrow$ empty list;
for $i \leftarrow 1$ **to** $len(u)$ **do**
    $j \leftarrow$ index such that $x_j \leq u_i \leq x_{j+1}$;
    $b \leftarrow y[j] - d[j] \cdot h_j$;
    $a \leftarrow (d[j+1] - d[j])/(3 \cdot h_j)$;
    $c \leftarrow \delta[j] - d[j] \cdot h_j - a \cdot h_j^2$;
    $s \leftarrow a \cdot (u_i - x_j)^3 + d[j] \cdot (u_i - x_j)^2 + c \cdot (u_i - x_j) + b$;
    append s to v;
end
return v;

Function *pchip_slopes(h, delta)*:
    $d \leftarrow$ array of zeros with length n;
    $k \leftarrow$ indices of the points where the slopes are of the same sign, i.e., $\text{sign}(\delta_{0:n-2}) \cdot \text{sign}(\delta_{1:n-1}) > 0$;
    $k \leftarrow k + 1$;
    $w1 \leftarrow 2 \cdot h_k + h_{k-1}$;
    $w2 \leftarrow h_k + 2 \cdot h_{k-1}$;
    $d_k \leftarrow (w1 + w2)/(w1/\delta_{k-1} + w2/\delta_k)$;
    $d_0 \leftarrow$ pchip_end$(h_0, h_1, \delta_0, \delta_1)$;
    append pchip_end$(h_{n-1}, h_{n-2}, \delta_{n-1}, \delta_{n-2})$ to d;
    return d;

Function *pchip_end(h1, h2, delta1, delta2)*:
    $d \leftarrow ((2 \cdot h1 + h2) \cdot \delta 1 - h1 \cdot \delta 2)/(h1 + h2)$;
    if $\text{sign}(\delta 1) \neq \text{sign}(\delta 2)$ *or* $|d| > 3 \cdot |\delta 1|$ *or* $|d| > 3 \cdot |\delta 2|$ **then**
        $d \leftarrow 0$
    end
    return d

BNumMet Examples

Example 1

```
from BNumMet.Interpolation import pchip
```

```
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
v = pchip(x, y, u)
# Plotting using Matplotlib
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("PCHIP Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

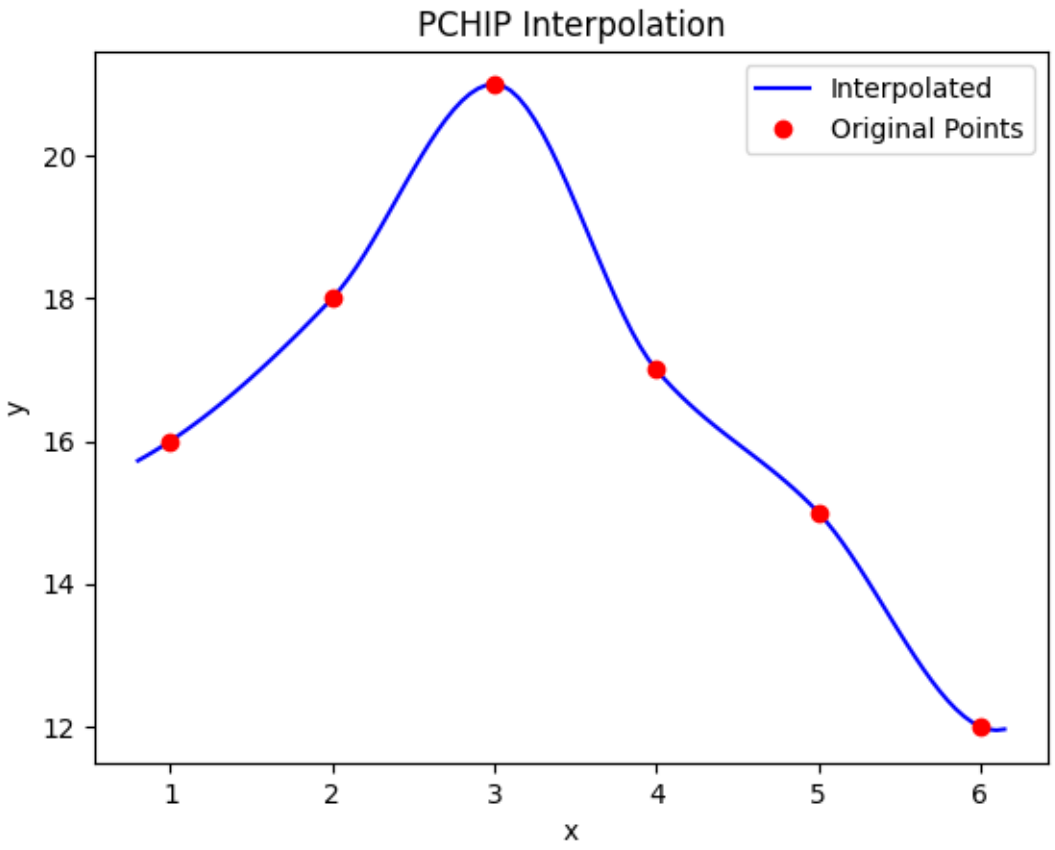


Fig. 7.3. PCHIP Example 1

Piecewise cubic Interpolatory Splines

The splines function interpolates the given points using piecewise cubic interpolatory splines.

Algorithm 11: Piecewise cubic Interpolatory Splines

Input: List of x coordinates x, list of y coordinates y, list of points where the interpolation is computed u, and parameter to specify if the points are already sorted *sorted*

Output: List of values of the interpolation at the points u

$n \leftarrow$ length of x;
$h \leftarrow x[1:n] - x[0:n-1]$;
$\delta \leftarrow (y[1:n] - y[0:n-1])/h$;
$d \leftarrow$ splineslopes(h, δ);
$v \leftarrow$ empty list;
for $i \leftarrow 1$ **to** $len(u)$ **do**
    $j \leftarrow$ index such that $x_j \leq u_i \leq x_{j+1}$;
    $b \leftarrow y[j] - d[j] \cdot h_j$;
    $a \leftarrow (d[j+1] - d[j])/(3 \cdot h_j)$;
    $c \leftarrow \delta[j] - d[j] \cdot h_j - a \cdot h_j^2$;
    $s \leftarrow a \cdot (u_i - x_j)^3 + d[j] \cdot (u_i - x_j)^2 + c \cdot (u_i - x_j) + b$;
    append s to v;
end
return v;
Function *splineslopes*(h, δ) **is**
    $a \leftarrow$ zeros$(len(h))$; $b \leftarrow$ zeros$(len(h))$; $c \leftarrow$ zeros$(len(h))$; $r \leftarrow$ zeros$(len(h))$;
    $a[:-1] \leftarrow h[1:]$;
    $a[-1] \leftarrow h[-2] + h[-1]$;
    $b[0] \leftarrow h[1]$;
    $b[1:] \leftarrow 2 \cdot (h[1:] + h[:-1])$;
    append$(b, h[-2])$;
    $c[0] \leftarrow h[0] + h[1]$;
    $c[1:] \leftarrow h[:-1]$;
    $r[0] \leftarrow ((h[0] + 2 \cdot c[0]) \cdot h[1] \cdot \delta[0] + h[0]^2 \cdot \delta[1])/c[0]$;
    $r[1:] \leftarrow 3 \cdot (h[1:] \cdot \delta[:-1] + h[:-1] \cdot \delta[1:])$;
    append$(r, (h[-1]^2 \cdot \delta[-2] + (2 \cdot a[-1] + h[-1]) \cdot h[-2] \cdot \delta[-1])/a[-1])$;
    $res \leftarrow$ lu_solve(diag$(a, -1)$ + diag(b) + diag$(c, 1), r)$;
    return res.astype$(float)$;
end

BNumMet Examples

Example 1

```
from BNumMet.Interpolation import splines
x = list(np.arange(1, 7, 1))
y = [16, 18, 21, 17, 15, 12]
u = np.arange(0.8, 6.2, 0.05)
```

```
v = splines(x, y, u)
# Plotting using Matplotlib
plt.plot(u, v, "b-", label="Interpolated")
plt.plot(x, y, "ro", label="Original Points")
plt.legend()
plt.title("Spline Interpolation")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

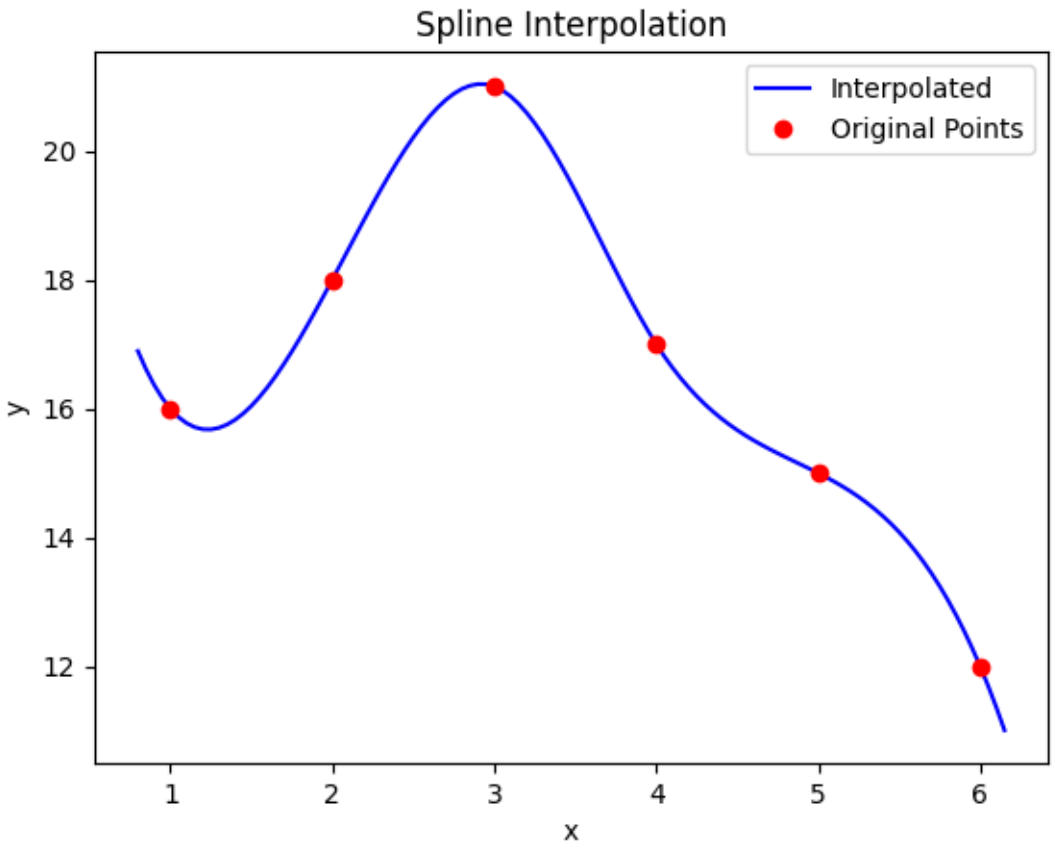


Fig. 7.4. Splines Example 1

NonLinear

Bisection Method

The bisection method bisects in half the interval given and then it keeps the subinterval that has opposing sign and reiterates the process with the subinterval until it has reach a certain number of iterations or the function value is close enough to zero.

Algorithm 12: Bisection Method

Input : Function f, interval $[a, b]$, number of maximum iterations $stop\_iters$, argument list $*args$

Output: Approximate zero of f

Function *bisect(fun, interval, stop_iters, iters, *args)*
- $x_0, x_1 \leftarrow a, b$;
- $f_0, f_1 \leftarrow f(x_0, *args), f(x_1, *args)$;
- **if** $f_0 \times f_1 > 0$ **then**
  - **raise** ValueError("No zeros in the given interval") ;
- **end**
- $x \leftarrow x_1$;
- $iterations \leftarrow 0$;
- **while** $f_1 \neq 0$ ***and*** $iterations < stop\_iters$ **do**
  - $iterations \leftarrow iterations + 1$;
  - $x \leftarrow 0.5 \cdot (x_0 + x_1)$;
  - $f \leftarrow f(x, *args)$;
  - **if** $f \times f_1 < 0$ **then**
    - $x_0 \leftarrow x$;
  - **end**
  - **else**
    - $x_1 \leftarrow x$;
    - $f_1 \leftarrow f$;
  - **end**
- **end**
- **if** *iters* **then**
  - **return** $x, iterations$;
- **end**
- **return** x ;

BNumMet Examples

Example 1

```
from BNumMet.NonLinear import bisect
fun = lambda x: x**2 - 2
```

```
interval = [1, 2]
sol = bisect(fun, interval, iters = False)
print("Bisection method: x = %f" % sol)

>> Bisection method: x = 1.414214
```

Example 2

```
from BNumMet.NonLinear import bisect
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
     for the n-th zero

zeros = [bisect(f, interval(n)) for n in range(0, 10)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with bisection method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

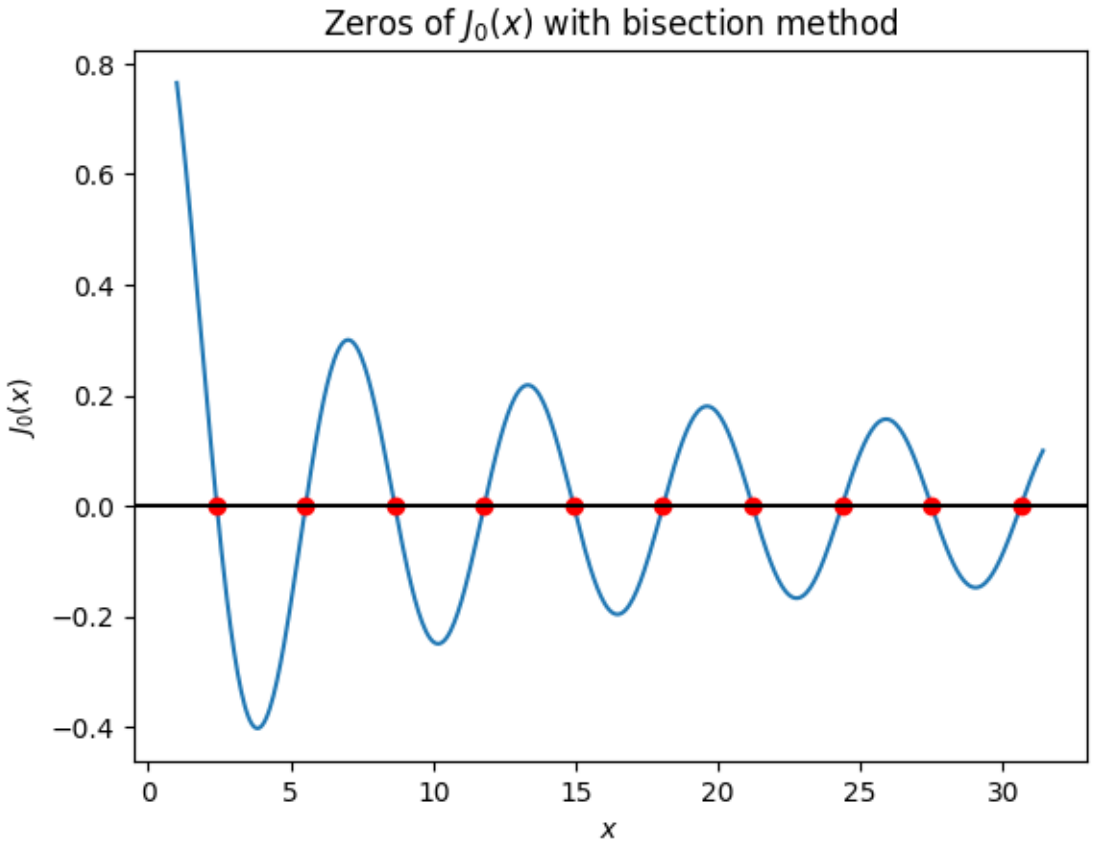


Fig. 7.5. Bisect Example 2

Secant Method

The secant method is a root finding procedure that using the secant line in the form of the approximate derivative iteratively to find the root of a function.

Algorithm 13: Secant Method

Input : Function f, interval $[a, b]$, number of maximum iterations *stop_iters*, argument list $*args$

Output: Approximate zero of f

```
Function secant(fun, interval, stop_iters, iters, *args)
    initialization: x0, x1 = interval;
    f0 = fun(x0, *args);
    f1 = fun(x1, *args);
    if f0 * f1 > 0 then
        raise ValueError("The function has no zeros in the given interval");
    else
        iterations = 0;
        while abs(x1 - x0) > ϵ and iterations < stop_iters do
            iterations += 1;
            x2 = x0;
            x0 = x1;
            x1 = x1 + (x1 - x2) / (fun(x2, *args) / fun(x1, *args) - 1);
        end
        if iters then
            return x1, iterations;
        else
            return x1;
        end
    end
```

BNumMet Examples

Example 1

```
from BNumMet.NonLinear import secant
fun = lambda x: x**2 - 2
interval = [1, 2]
sol, nIter = secant(fun, interval, iters = True)
print("Secant method: x = %f, nIter = %d" % (sol, nIter))

>> Secant method: x = 1.414214, nIter = 7
```

Example 2

```
from BNumMet.NonLinear import secant
f = lambda x: sp.jv(0, x)  # Bessel function of the first
   kind of order 0
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
    for the n-th zero

```

```
zeros = [secant(f, interval(n)) for n in range(0, 10)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with secant method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

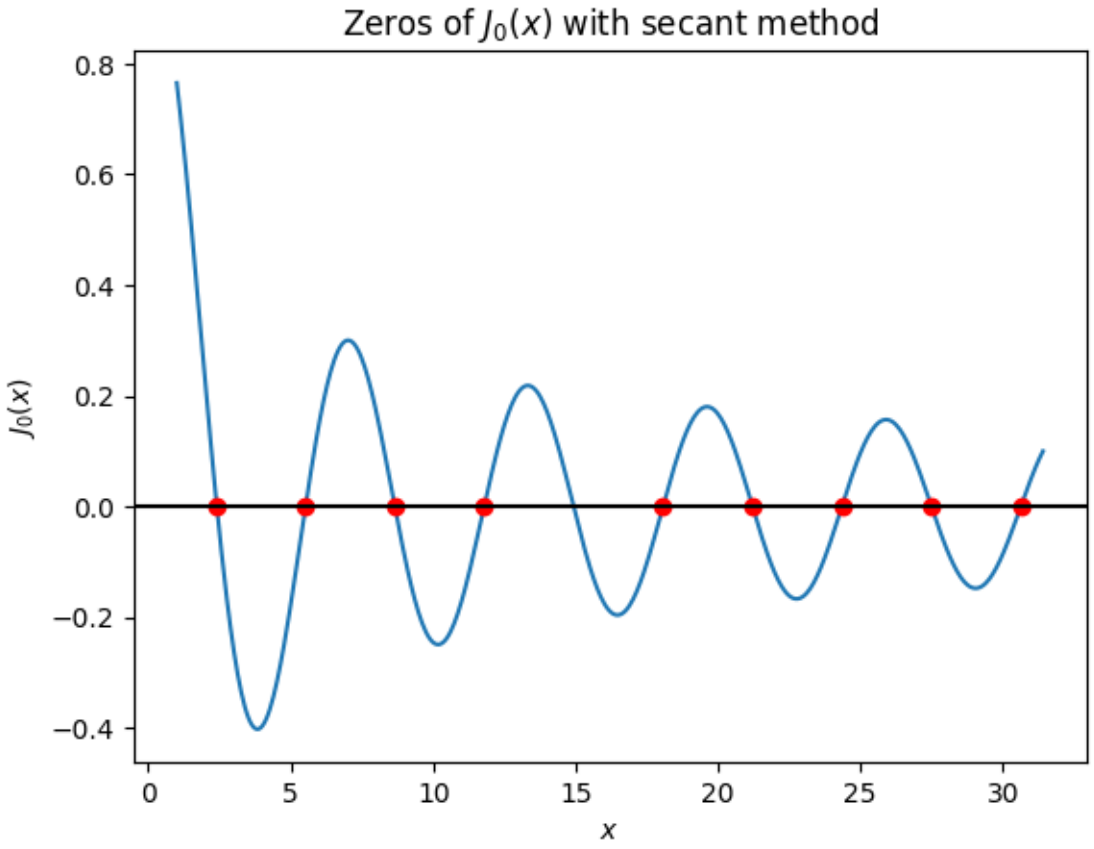


Fig. 7.6. Secant Example 2

Newton's Method

Newton's method is a root finding procedure that uses the tangent line in the form of the exact derivative iteratively to find the root of a function.

Algorithm 14: Newton-Raphson Method

Input : fun, derivative, start_point, stop_iters=100, iters=False, *args

Output: x: zeros of the function fun

Function *newton(fun, derivative, start_point, stop_iters, iters, *args)*
    initialization: previous_x = start_point - 1;
    xn = start_point;
    fn = fun(xn, *args);
    if *derivative(xn, *args) == 0* **then**
        raise ValueError("The derivative of the function is zero");
    else
        iterations = 0;
        while *fn != 0 and not np.isclose(xn - previous_x, 0) and derivative(xn, *args) != 0 and iterations < stop_iters* **do**
            iterations += 1;
            previous_x = xn;
            xn = xn - fn / derivative(xn, *args);
            fn = fun(xn, *args);
        end
        if *iters* **then**
            return xn, iterations;
        else
            return xn;
        end
    end

BNumMet Examples

Example 1

```
from BNumMet.NonLinear import newton
fun = lambda x: x**2 - 2
derivative = lambda x: 2 * x
interval = [1, 2]
sol, nIter = newton(fun, derivative, start_point=2, iters =
    True)
print("Newton's method: x = %f, nIter = %d" % (sol, nIter))
```

Example 2

```
from BNumMet.NonLinear import newton
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
derivative = lambda x: sp.jvp(0, x, 1)  # Derivative of the
    Bessel function
```

```
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
    for the n-th zero

zeros = [newton(f, derivative, start_point=interval(n)[0])
   for n in range(1, 11)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with Newton's method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

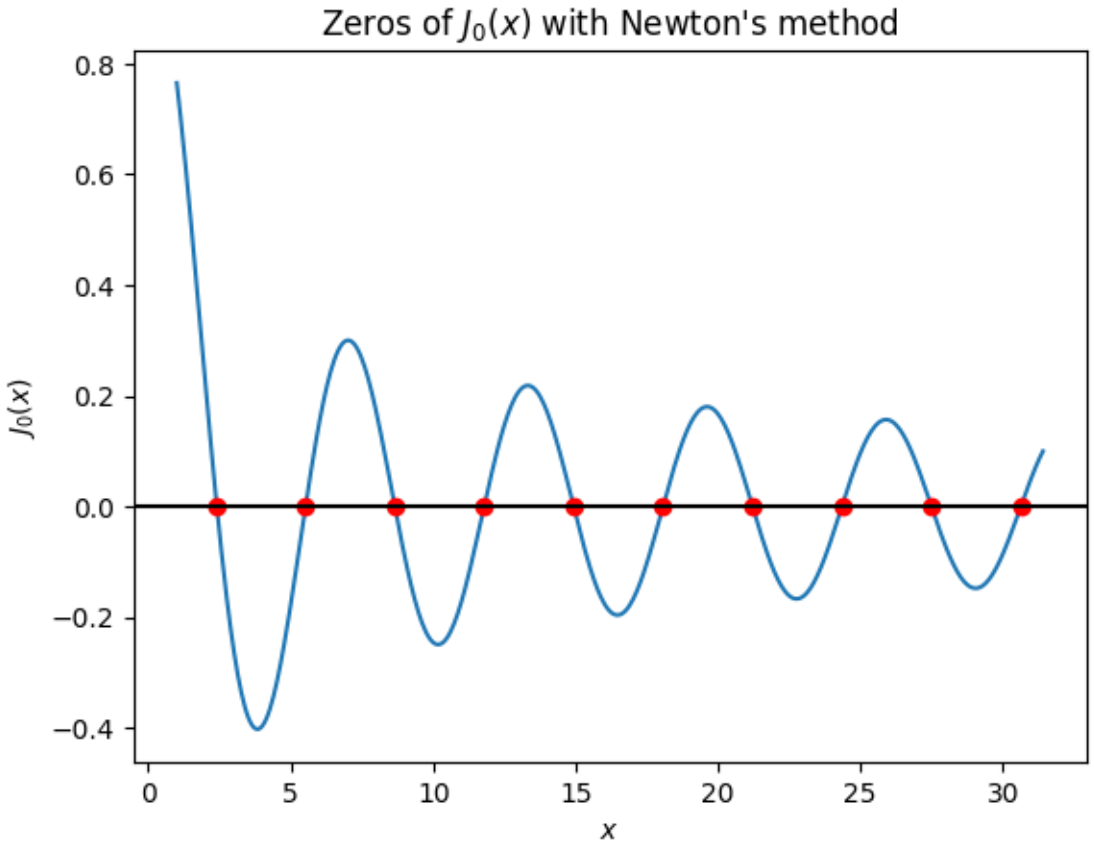


Fig. 7.7. Newton's Method Example 2

Inverse Quadratic Interpolation (I.Q.I.)

The inverse quadratic interpolation method is a root finding procedure that uses the inverse quadratic interpolation formula to find the root of a function.

Algorithm 15: Inverse Quadratic Interpolation

Input : Function f, Initial values of x_0, x_1, x_2, Maximum number of iterations $stop\_iters$, and Optional arguments $args$

Output: Zeros of the function f

Function $IQI(f, x\_values, stop\_iters, iters, *args)$

- $x_0, x_1, x_2 \leftarrow x\_values$;
- $iterations \leftarrow 0$;
- **while** $|x_1 - x_0| > \epsilon$ *and* $iterations < stop\_iters$ **do**
  - $f_0, f_1, f_2 \leftarrow f(x_0, *args), f(x_1, *args), f(x_2, *args)$;
  - $aux1 \leftarrow \frac{x_0 f_1 f_2}{(f_0 - f_1)(f_0 - f_2)}$;
  - $aux2 \leftarrow \frac{x_1 f_0 f_2}{(f_1 - f_0)(f_1 - f_2)}$;
  - $aux3 \leftarrow \frac{x_2 f_1 f_0}{(f_2 - f_0)(f_2 - f_1)}$;
  - $new \leftarrow aux1 + aux2 + aux3$;
  - $x_0, x_1, x_2 \leftarrow new, x_0, x_1$;
  - $iterations \leftarrow iterations + 1$;
- **end**
- **if** *iters* **then**
  - **return** x_0, $iterations$;
- **end**
- **return** x_0;

BNumMet Examples

Example 1

```
from BNumMet.NonLinear import IQI
fun = lambda x: x**2 - 2
points = [1, 1.5 ,2]
sol, nIter = IQI(fun, points, iters = True)
print("IQI method: x = %f, nIter = %d" % (sol, nIter))

>> IQI method: x = 1.414214, nIter = 6
```

Example 2

```
from BNumMet.NonLinear import IQI
f = lambda x: sp.jv(0, x)  # Bessel function of the first
   kind of order 0
interval = lambda n: [
    n * np.pi,
    (((n + 1) * np.pi) - n * np.pi) / 2,
    (n + 1) * np.pi,
]  # Interval for the n-th zero

zeros = [IQI(f, interval(n)) for n in range(0, 7)]

```

```
11
12  x = np.arange(1, 10 * np.pi, np.pi / 50)
13  y = f(x)
14  plt.plot(x, y)
15  plt.plot(zeros, np.zeros(len(zeros)), "ro")
16  plt.axhline(0, color="k")
17  plt.title("Zeros of $J_0(x)$ with IQI method")
18  plt.xlabel("$x$")
19  plt.ylabel("$J_0(x)$")
20  plt.show()
```

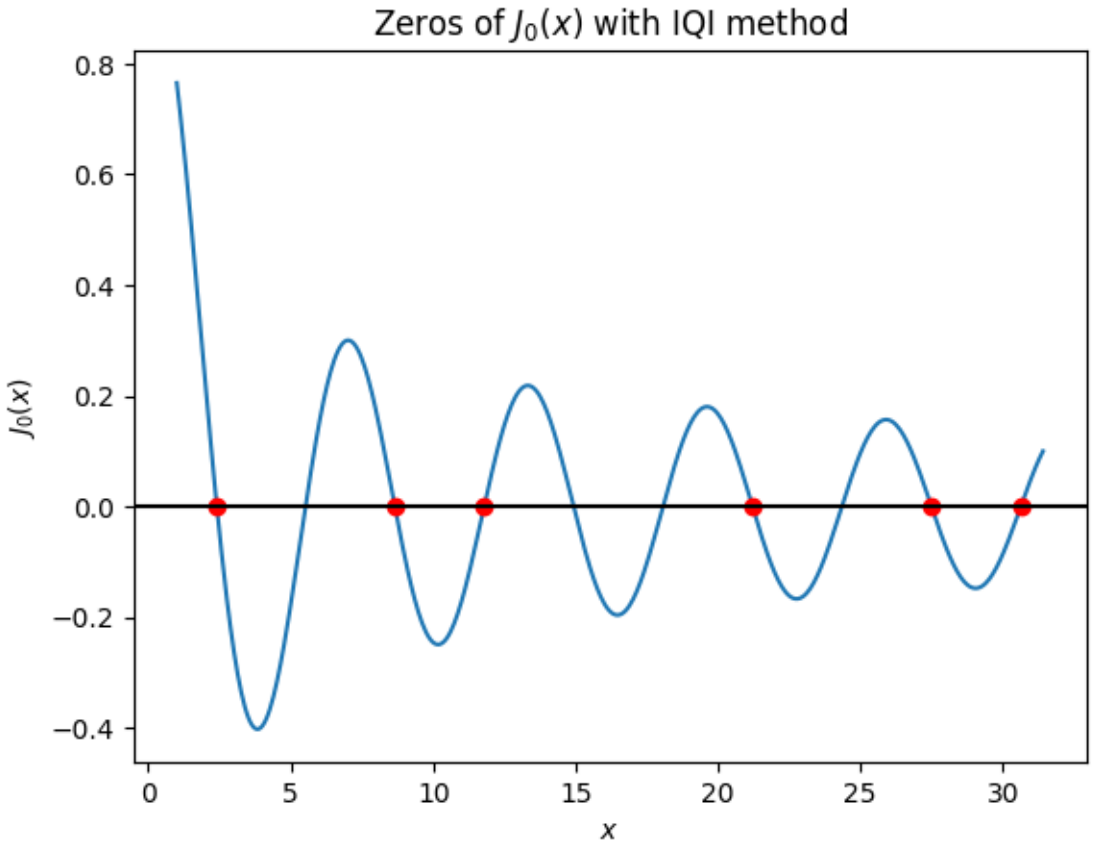


Fig. 7.8. IQI Example 2

Brentt-Dekker

Multiple implementations with small differences exist, we opted for the original one publish by Brent [29], the reason behind the naming of Brent-Dekker Algorithm is because at first Dekker in conjunction with Wingartenn and their respective colleagues proposed an initial version, later Brent came an offer a better approach and convergence [30], in some references like Scipy's they will call this algorithm Brent-Wingartenn-Dekker Algorithm.

Algorithm 16: Brent-Dekker's Algorithm

Input : Function f, interval $[a, b]$, tolerance tol, maximum iterations $stop_iters$, arguments $*args$

Output: Zero of f in $[a, b]$

Function *zBrentDekker*$(f, interval, tol, stop\_iters, iters, steps, *args)$

- $fa \leftarrow f(a, *args)$; $fb \leftarrow f(b, *args)$;
- **if** $fa * fb > 0$ **then Raise** *Error("Function has no zeros in the given interval")*
- $c \leftarrow a$; $fc \leftarrow fa$; $d \leftarrow b - a$; $e \leftarrow b - a$;
- **if** $abs(fc) < abs(fb)$ **then** $a, b, c \leftarrow b, c, b$; $fa, fb, fc \leftarrow fb, fc, fb$;
- $tolerance \leftarrow 2 * \epsilon * abs(b) + tol$;
- $m \leftarrow 0.5 * (c - b)$; $iterations \leftarrow 0$;
- **while** $abs(m) > tolerance$ ***and*** fb ***and*** $iterations < stop\_iters$ **do**
  - **if** $abs(e) < tolerance$ ***or*** $abs(fa) <= abs(fb)$ **then** $d \leftarrow m$; $\bigwedge$ $e \leftarrow m$;
  - **else**
    - $s \leftarrow fb/fa$;
    - **if** $a == c$ **then** $p \leftarrow 2 * m * s$; $\bigwedge$ $q \leftarrow 1 - s$;
    - **else**
      - $q \leftarrow fa/fc$; $r \leftarrow fb/fc$;
      - $p \leftarrow s * (2 * m * q * (q - r) - (b - a) * (r - 1))$;
      - $q \leftarrow (q - 1) * (r - 1) * (s - 1)$;
    - **if** $p > 0$ **then** $q \leftarrow -q$;
    - **else** $p \leftarrow -p$;
    - $s \leftarrow e$; $\bigwedge$ $e \leftarrow d$;
    - **if** $2p < 3m \cdot q - |tolerance * q|$ *and* $p < abs(0.5 \cdot s \cdot q)$ **then** $d \leftarrow p/q$;
    - **else** $d \leftarrow m$; $e \leftarrow m$;
  - $a \leftarrow b$; $\bigwedge$ $fa \leftarrow fb$;
  - $b \leftarrow b + d$ **if** $abs(d) > tolerance$ **else** $b + np.sign(m) * tolerance$;
  - $fb \leftarrow f(b, *args)$;
  - **if** $np.sign(fb) == np.sign(fc)$ **then** $c, fc, d, e \leftarrow a, fa, b - a, b - a$
  - **else if** $abs(fc) < abs(fb)$ **then** $a, b, c \leftarrow b, c, b$; $fa, fb, fc \leftarrow fb, fc, fb$
  - $tolerance \leftarrow 2 * \epsilon * abs(b) + tol$;
  - $m \leftarrow 0.5 * (c - b)$; $iterations \leftarrow iterations + 1$
- $zero \leftarrow b$
- **return** *zero*

BNumMet Examples

Example 1

```
from BNumMet.NonLinear import zBrentDekker
fun = lambda x: x**2 - 2
```

```
interval = [1, 2]
sol, nIter = zBrentDekker(fun, interval, iters = True)
print("Brent-Dekker method: x = %f, nIter = %d" % (sol, nIter
    ))

>> Brent-Dekker method: x = 1.414214, nIter = 7
```

Example 2

```
from BNumMet.NonLinear import zBrentDekker
f = lambda x: sp.jv(0, x)  # Bessel function of the first
    kind of order 0
interval = lambda n: [n * np.pi, (n + 1) * np.pi]  # Interval
     for the n-th zero

zeros = [zBrentDekker(f, interval(n)) for n in range(0, 10)]


x = np.arange(1, 10 * np.pi, np.pi / 50)
y = f(x)
plt.plot(x, y)
plt.plot(zeros, np.zeros(len(zeros)), "ro")
plt.axhline(0, color="k")
plt.title("Zeros of $J_0(x)$ with Brent-Dekker method")
plt.xlabel("$x$")
plt.ylabel("$J_0(x)$")
plt.show()
```

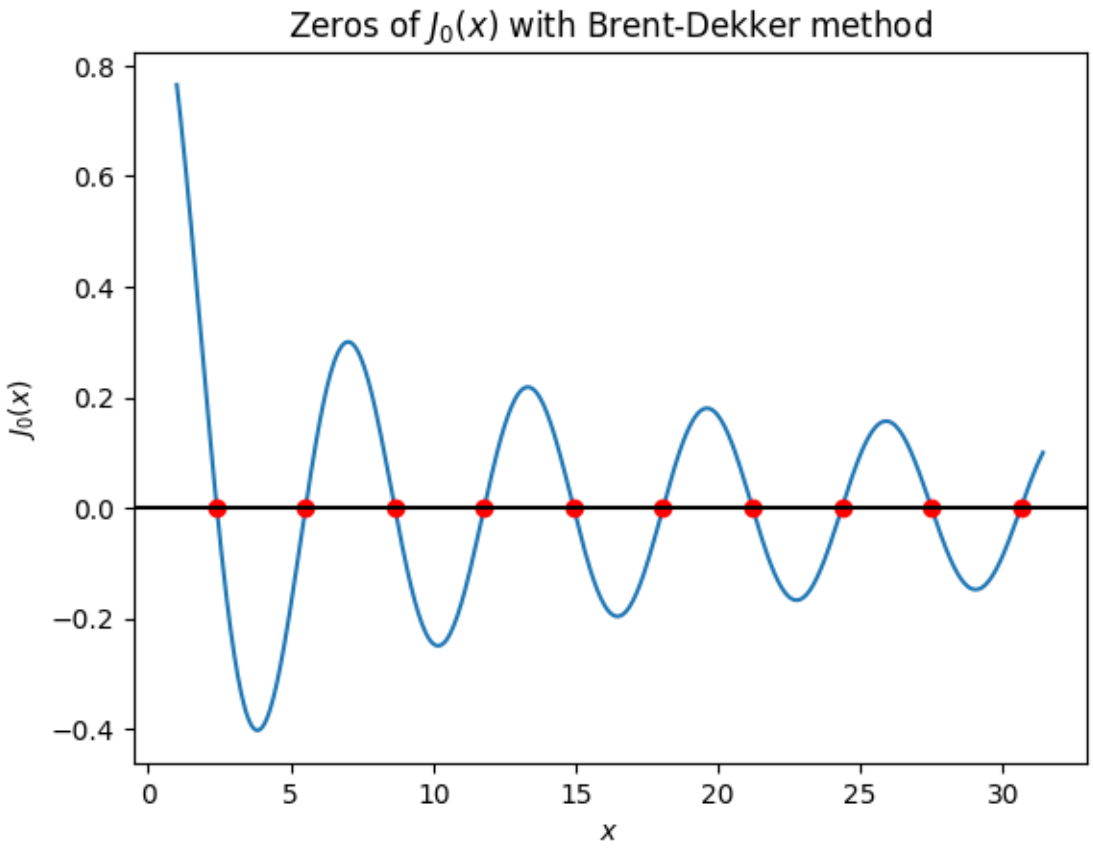


Fig. 7.9. Brentt-Dekker Example 2

Random Number Generators

Lehmers Random

Lehmers Random Number Generator is a congruential algorithm using the following formula:

$$x_{n+1} = (a \cdot x_n + c) \mod m$$

Algorithm 17: Lehmer's Random Number Generator

Input: a, c, m, x

Output: Random number between 0 and 1

```
Function lehmers_rand(a=None, c=None, m=None, x=None):
    global lehmers_vars;
    if Variables Not Initialized then
        if a == None or c == None or m == None or x == None then
            lehmers_init(7^5, 0, 2^31 − 1, 1.0);
        else
            lehmers_init(a, c, m, x);
        end
    else
        // Generate a random number using Lehmer's formula
        x = (a · x + c) mod m;
    end
    return x/m;
end
```

BNumMet Examples

Example 1

```
from BNumMet.Random import lehmers_rand
for i in range(10):
    print(lehmers_rand())

>> Lehmers Random Number Generator Initialized with default values
        a=16807
        c=0
        m=2147483647
        x=1.0
    7.826369259425611e-06
    0.13153778814316625
    0.7556053221950332
    0.4586501319234493
    0.5327672374121692
    0.21895918632809036
```

```
    0.04704461621448613
    0.678864716868319
    0.6792964058366122
    0.9346928959408276
```

Example 2: Predictability of Lehmer's Random Number Generator

```
from BNumMet.Random import lehmers_rand, clear_lehmers_vars
clear_lehmers_vars()
arr = [1]
for i in range(10):
    aux = lehmers_rand(a=2**16 + 3, m=2**31, c=0, x=arr[-1])
    if len(arr) >= 3:
        lehmerFormula = (6 * arr[-1] - 9 * arr[-2]) % 1  # Test Xn = (6Xn-1 - 9Xn-2)
        print(f"Lehmer's = {aux}\nPredicted = {lehmerFormula}\n")
    arr.append(aux)

>>  Lehmer's = 0.0008239871822297573
    Predicted = 0.0008239871822297573

    Lehmer's = 0.003295936156064272
    Predicted = 0.003295936156064272

    Lehmer's = 0.012359732296317816
    Predicted = 0.012359732296317816

    Lehmer's = 0.04449496837332845
    Predicted = 0.04449496837332845

    Lehmer's = 0.15573221957311034
    Predicted = 0.15573221957311034

    Lehmer's = 0.533938602078706
    Predicted = 0.533938602078706

    Lehmer's = 0.8020416363142431
    Predicted = 0.8020416363142431

    Lehmer's = 0.006802399177104235
    Predicted = 0.006802399177104235
```

Example 3: Visual failure

```
from BNumMet.Random import lehmers_rand, clear_lehmers_vars
clear_lehmers_vars()
fail2 = (
    lambda: float(
```

```
        (int(lehmers_rand(a=65539, c=0, m=2**31, x=123) *
            (2**31)) >> 23) & 0xFF
    )
    / 255
)
fail = [(fail2(), fail2()) for i in range(100000)]
plt.scatter(*zip(*fail), s=1, c="black")
plt.show()
```

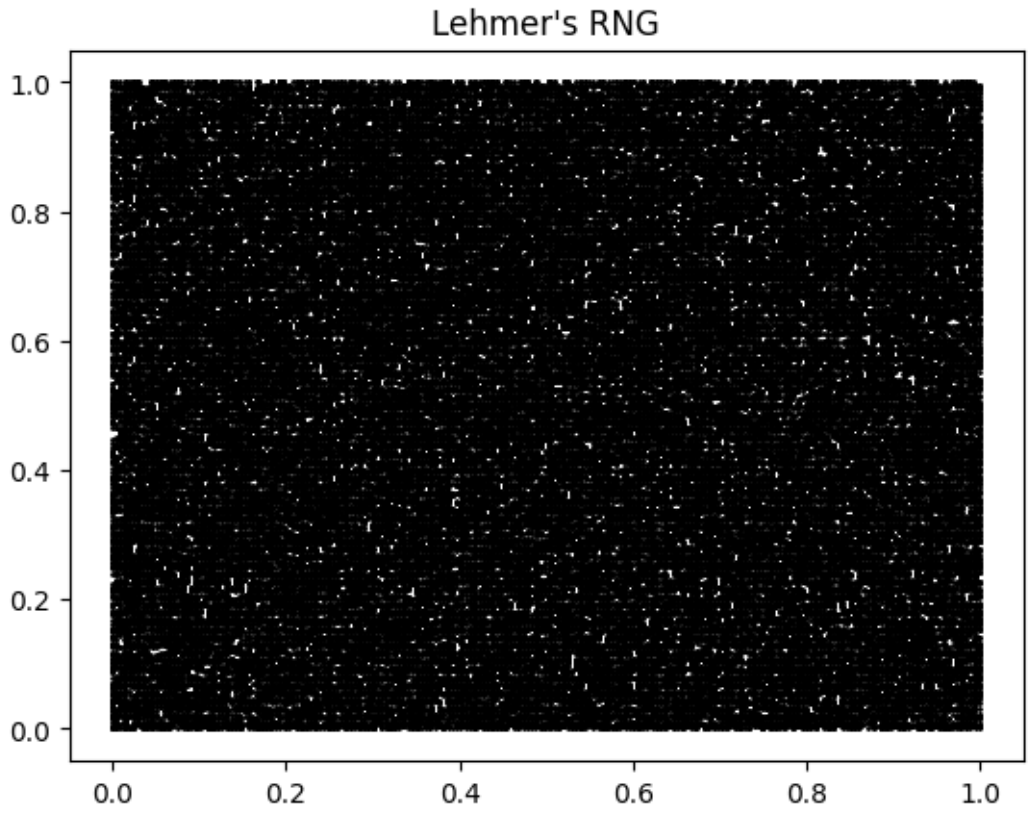


Fig. 7.10. Lehmers Rand Example 3

Marsaglia's Random Generator

The algorithm is based of [33] the original paper where Gearge Marsaglia and Arif Zaman discussed it. It is based on the subtract with borrow generator.

Algorithm 18: Marsaglia Random Number Generator Algorithm

Input : base, lag_r, lag_s, carry, seed_tuple

Output: The generated random number

Function *marsaglia_rand(base, lag_r, lag_s, carry, seed_tuple)***:**
    global marsaglia_vars;
    if *(global dictionary marsagliaVars is NOT initialized)* **then**
        if *(Some inputs are NONE)* **then**
            marsaglia_init($2^{31} - 1, 19, 7, 1, (1, 1)$);
        else
            marsaglia_init(base, lag_r, lag_s, carry, seed_tuple);
        end
    else
    end
    $x_{new} = x_0 - x_{lag_r - lag_s} - carry$;
    if $x_{new} < 0$ **then**
        $x_{new} += base$;
        $carry = 1$;
    else
        $carry = 0$;
    end
    $\overrightarrow{x} = [x(1:),\ x_{new}]$
    return $x_{new}/base$;
end

BNumMet Examples

Example 1

```
from BNumMet.Random import marsaglia_rand,
   clear_marsaglia_vars
clear_marsaglia_vars()
for i in range(10):
    print(marsaglia_rand(base=41, lag_r=2, lag_s=1, carry=0,
       seed_tuple=(0, 1)))

>>  0.975609756097561
    0.024390243902439025
    0.926829268292683
    0.0975609756097561
    0.8048780487804879
    0.2926829268292683
    0.4878048780487805
    0.8048780487804879
    0.6585365853658537
    0.12195121951219512
```

Example 2

```
from BNumMet.Random import marsaglia_rand,
    clear_marsaglia_vars
clear_marsaglia_vars()
fail = [
    (
        marsaglia_rand(base=100, lag_r=2, lag_s=1, carry=0,
            seed_tuple=(0, 1)),
        marsaglia_rand(base=100, lag_r=2, lag_s=1, carry=0,
            seed_tuple=(0, 1)),
    )
    for i in range(100000)
]
plt.scatter(*zip(*fail), s=1, c="black")
```

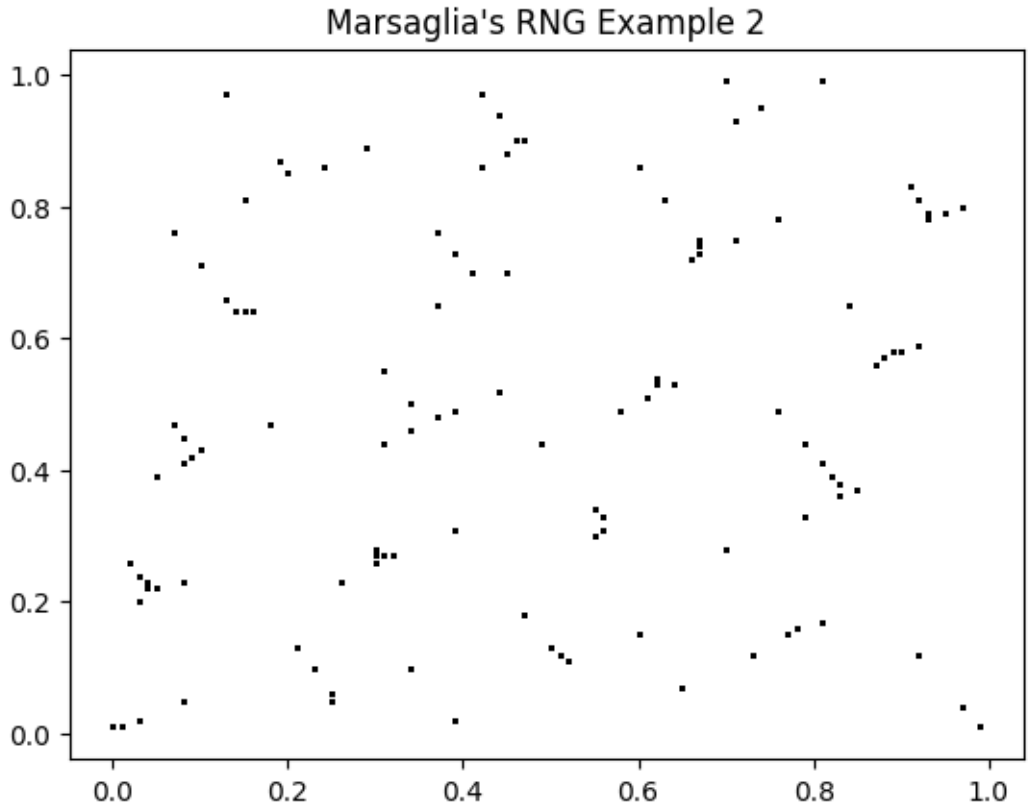


Fig. 7.11. Marsaglia Rand Example 2

Example 3

```
from BNumMet.Random import marsaglia_rand,
    clear_marsaglia_vars
clear_marsaglia_vars()
fail = [
    (
        marsaglia_rand(base=41, lag_r=2, lag_s=1, carry=0,
            seed_tuple=(0, 1)),
```

```
        marsaglia_rand(base=41, lag_r=2, lag_s=1, carry=0,
            seed_tuple=(0, 1)),
    )
    for i in range(100000)
]
plt.scatter(*zip(*fail), s=1, c="black")
```

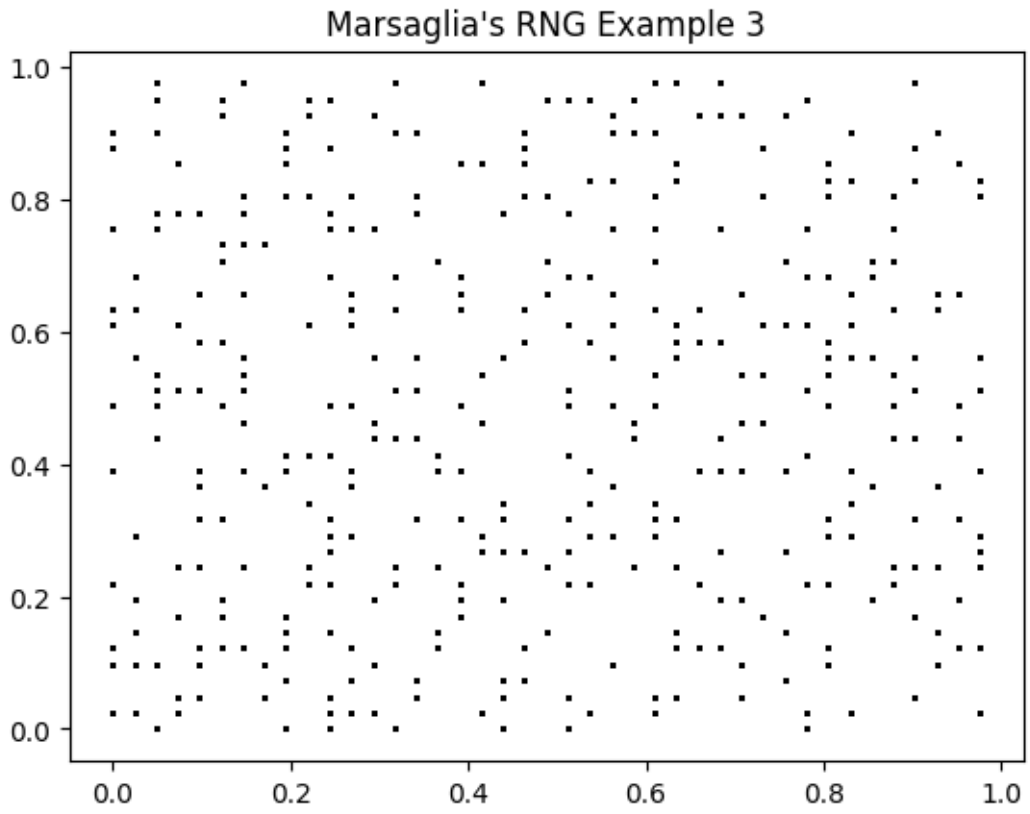


Fig. 7.12. Marsaglia Rand Example 3

Mersenne Twister Random Generator

The algorithm was based of [37] which is an implementation of the Mersnne Twister in the programming language of C. The implementation has been tested using the NIST Random test suite [35].

Algorithm 19: Mersenne Twister Random Number Generator

```
1  Function genrand(seed: int = None):
       // global variables
2      global mt, mti, mag01
3      if mtVars is not initialized then
4          if seed is None then
5              seed = 4357
6          end
7          sgenrand(seed)
8      end
9      mag01 = [0x0, A]
10     if mti ≥ N then
11         kk = 0;
12         y = None;
13         while kk < N - M do
14             y = (mt[kk] && UPPER_MASK) || (mt[kk + 1] &&
                 LOWER_MASK);
15             mt[kk] = mt[kk + M] ⊕ (y >> 1) ⊕ mag01[y & 0x1];
16             kk += 1;
17         end
18         while kk < N - 1 do
19             y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK);
20             mt[kk] = mt[kk + (M - N)] ⊕ (y >> 1) ⊕ mag01[y && 0x1];
21             kk += 1;
22         end
23         y = (mt[N - 1] && UPPER_MASK) || (mt[0] && LOWER_MASK);
24         mt[N - 1] = mt[M - 1] ⊕ (y >> 1) ⊕ mag01[y & 0x1];
25         mti = 0;
26     end
27     y = mt[mti];
28     mti += 1;
       // Tempering
29     y ⊕= TEMPERING_SHIFT_U(y);
30     y ⊕= (TEMPERING_SHIFT_S(y) & TEMPERING_MASK_B);
31     y ⊕= (TEMPERING_SHIFT_T(y) & TEMPERING_MASK_C);
32     y ⊕= TEMPERING_SHIFT_L(y);
33     return y / 0xFFFFFFFF;
```

Remark 1. ⊕ Operator stands for a bitwise XOR

2. && Operator stands for a bitwise AND

3. || Operator stands for a bitwise OR

BNumMet Examples

Example 1

```
from BNumMet.Random import clear_mt_vars,genrand
clear_mt_vars()
for i in range(10):
    print(genrand())

>>  Initialized the global dictionary mtVars with seed 4357
    0.8173300600185361
    0.9990608997175147
    0.5103543725587322
    0.13153290984489324
    0.03541634837990076
    0.9924695345089932
    0.6257087035630151
    0.06259194576707482
    0.4107105111262553
    0.13477367491805314
```

Example 2

```
from BNumMet.Random import clear_mt_vars,genrand
clear_mt_vars()
toPlot = [(genrand(),genrand()) for i in range(100000)]
plt.scatter(*zip(*toPlot), s=1, c="black")
```

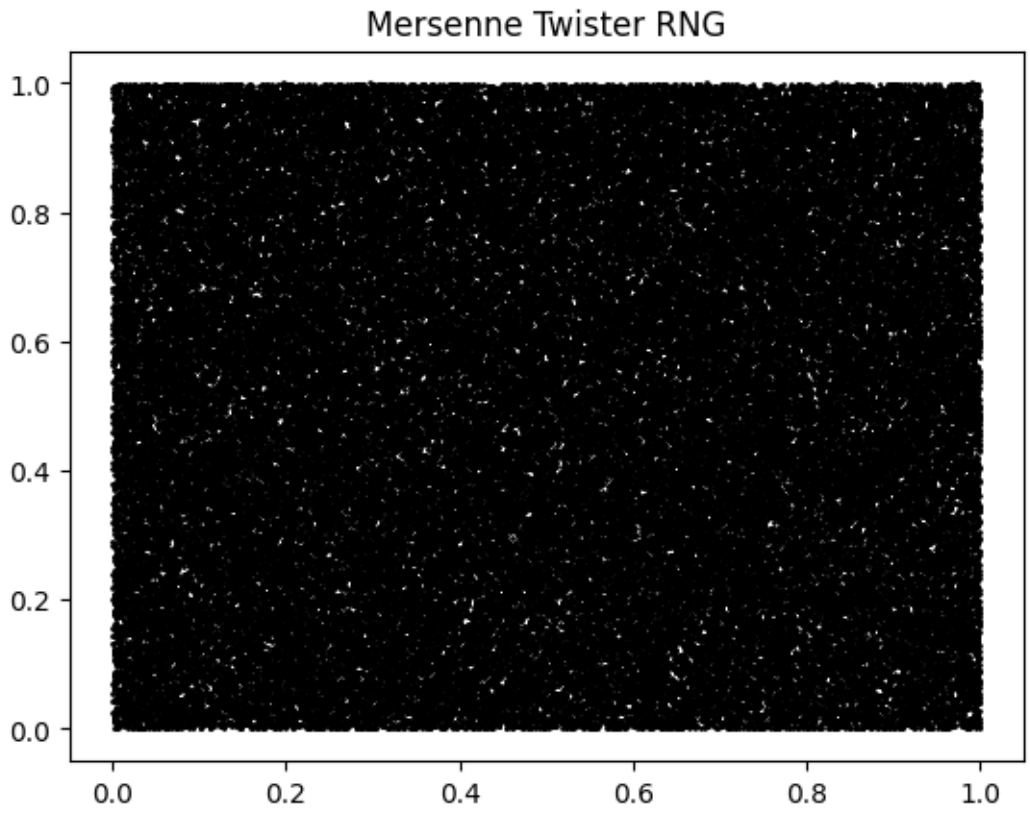


Fig. 7.13. Mersenne Twister Rand Example 2

Visualizers

LU Visualizer

Example 1: No arguments

```
from BNumMet.Visualizers.LUVisualizer import LUVisualizer
luVisualizer = LUVisualizer()
display(luVisualizer.run())
```

1. Initial State:

1.000	2.000	3.000
4.000	5.000	6.000
7.000	8.000	9.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ * & 1.0 & 0.0 \\ * & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$$

Rank: 0

2. Click on first column, second row:

4.000	5.000	6.000
0.000	0.750	1.500
0.000	-0.750	-1.500

Previous Step

Reset

$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.25 & 1.0 & 0.0 \\ 1.75 & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & * & * \\ 0.0 & * & * \end{pmatrix}$$

Rank: 1

3. Click on second column, third row:

4.000	5.000	6.000
0.000	-0.750	-1.500
0.000	0.000	0.000

Previous Step

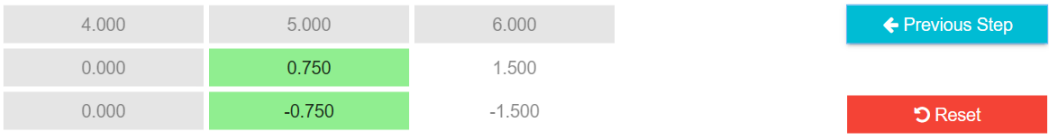
Reset

$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \\ 1.0 & 0.0 & 0.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 1.75 & 1.0 & 0.0 \\ 0.25 & -1.0 & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & -0.75 & -1.5 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Rank: 2

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
We have finished
```

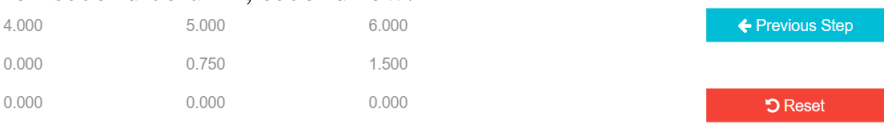
4. Click Previous Step:



$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.25 & 1.0 & 0.0 \\ 1.75 & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & * & * \\ 0.0 & * & * \end{pmatrix}$$

Rank: 1

5. Click on second column, second row:

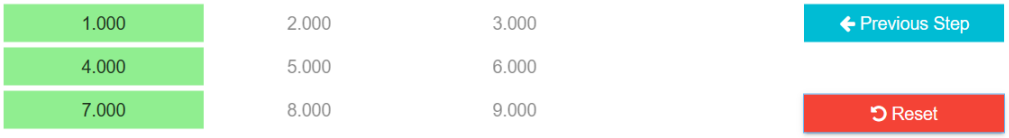


$$P: \begin{pmatrix} 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.25 & 1.0 & 0.0 \\ 1.75 & -1.0 & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 4.0 & 5.0 & 6.0 \\ 0.0 & 0.75 & 1.5 \\ 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Rank: 2

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
We have finished
```

6. Click on Reset



$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ * & 1.0 & 0.0 \\ * & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \end{pmatrix}$$

Rank: 0

Example 2: Arguments and Rank Revealing

```
from BNumMet.Visualizers.LUVisualizer import LUVisualizer
A = np.array([[1,2,3,7], [1,2,3,7], [1,2,3,7],[1,2,4,7]],
    dtype=float)
luVisualizer = LUVisualizer(A)
display(luVisualizer.run())
```

1. Initial State:

1.000	2.000	3.000	7.000
1.000	2.000	3.000	7.000
1.000	2.000	3.000	7.000
1.000	2.000	4.000	7.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ * & 1.0 & 0.0 & 0.0 \\ * & * & 1.0 & 0.0 \\ * & * & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

Rank: 0

2. Click on the first column, second row:

1.000	2.000	3.000	7.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	1.000	0.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & * & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 1.0 & 2.0 & 3.0 & 7.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & * & * \\ 0.0 & 0.0 & * & * \end{pmatrix}$$

Rank: 1

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
```

3. Click on the third column, fourth row:

1.000	2.000	3.000	7.000
0.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000

Previous Step

Reset

$$P: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \end{pmatrix} \quad \tilde{L}: \begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 1.0 \end{pmatrix} \quad \tilde{U}: \begin{pmatrix} 1.0 & 2.0 & 3.0 & 7.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Rank: 2

```
Messages from system: The matrix is not FULL RANK, that is, the column below the diagonal is zero therefore we move to the
next column until we find a non-zero column
We have finished
```

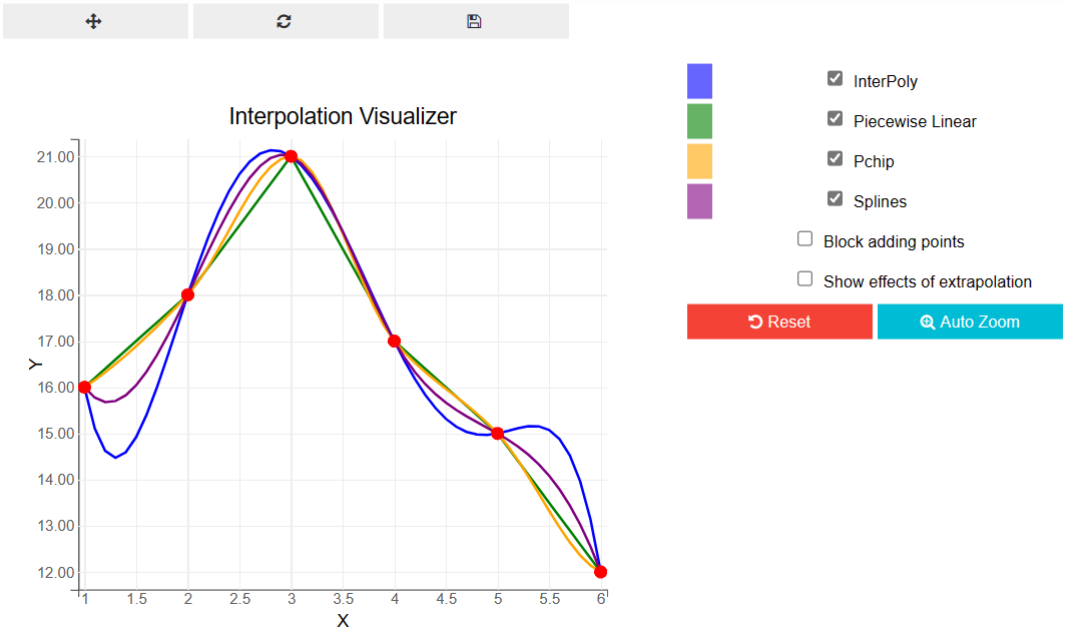
Interpolation Visualizer

Example 1

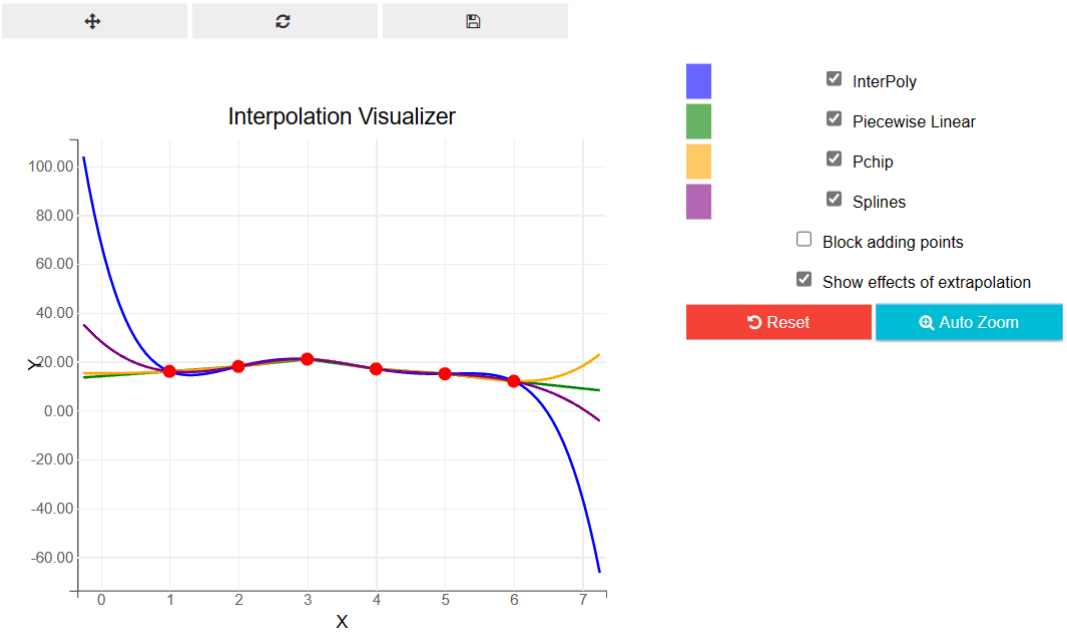
```
from BNumMet.Visualizers.InterpolationVisualizer import
    InterpolVisualizer
```

```
interpolVisualizer = InterpolVisualizer()
display(interpolVisualizer.run())
```

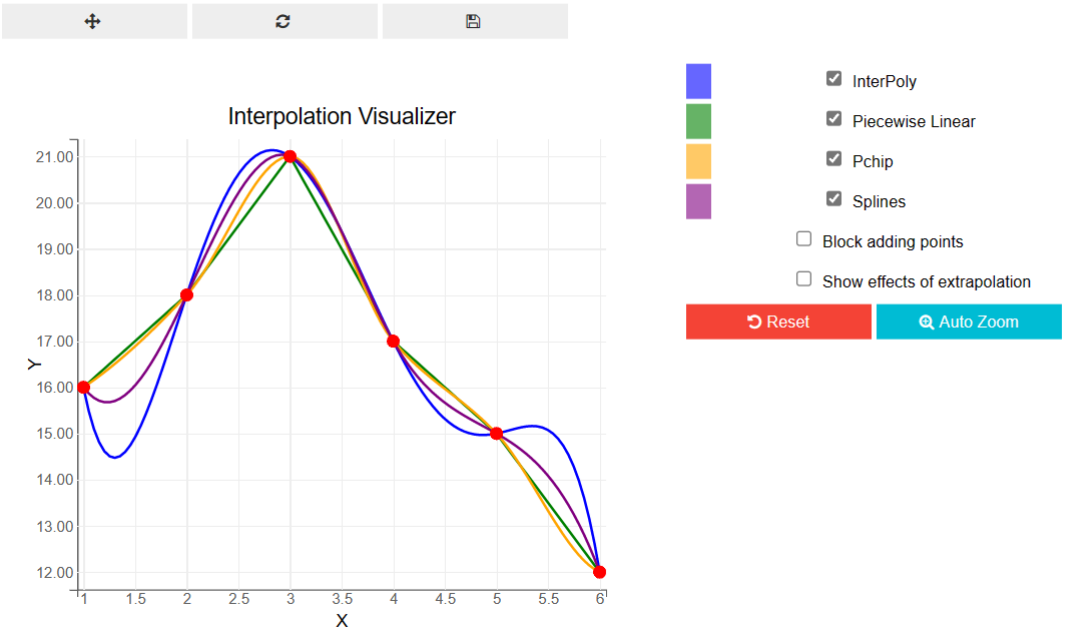
1. Initial State:



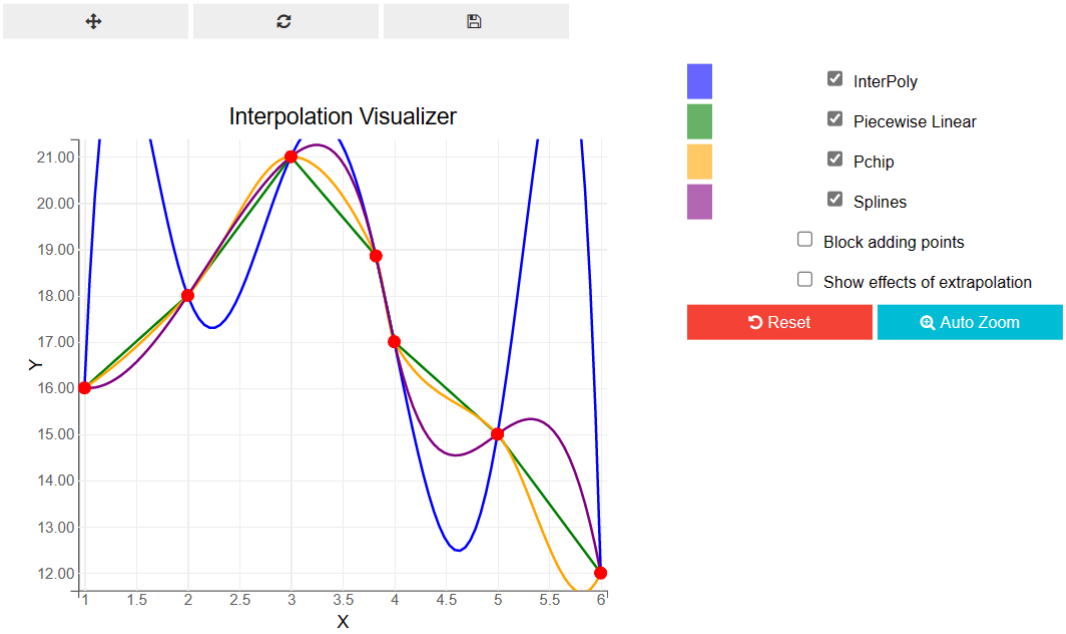
2. Checked Box Extrapolation and AutoZoom:



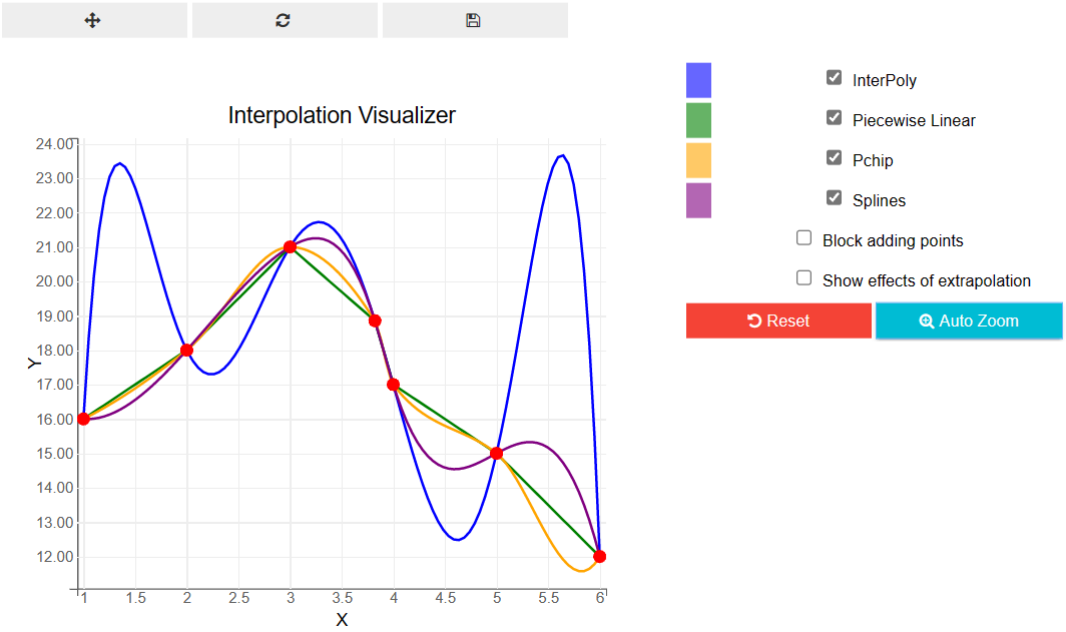
3. Reset Button:



4. Added Point:



5. AutoZoom:

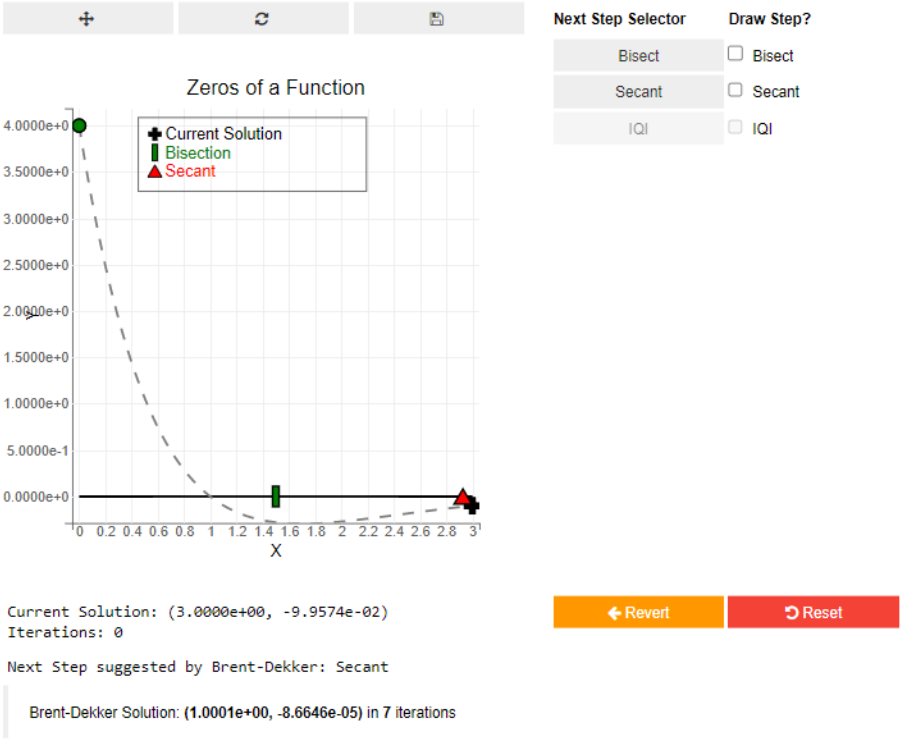


NonLinear Visualizer

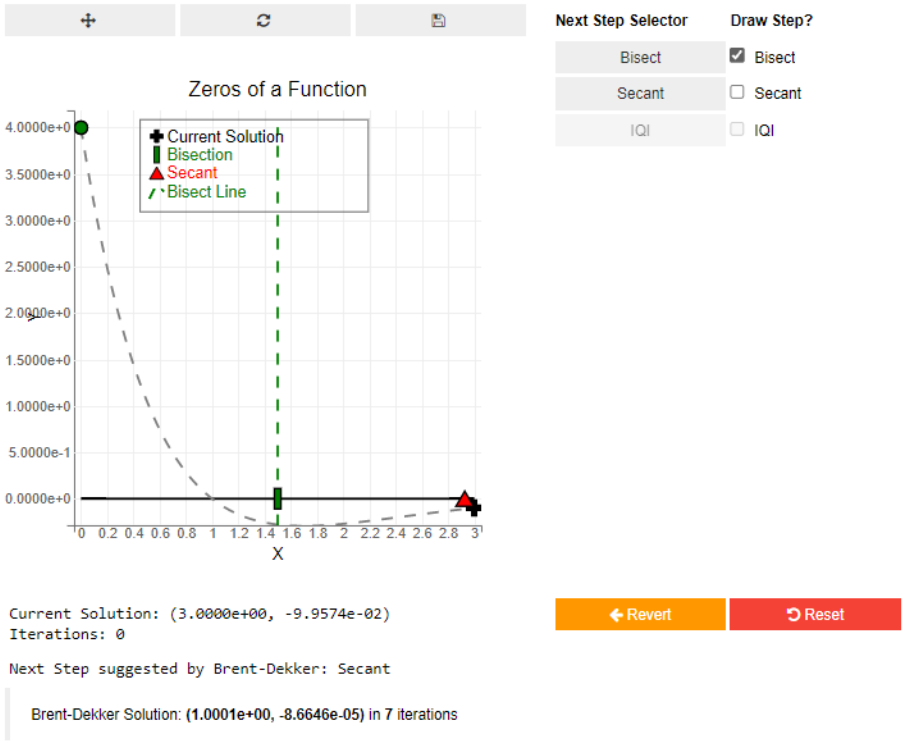
Example 1: No arguments

```
from BNumMet.Visualizers.NonLinearVisualizer import
    NonLinearVisualizer
zerosVisualizer = NonLinearVisualizer()
zerosVisualizer.run()
```

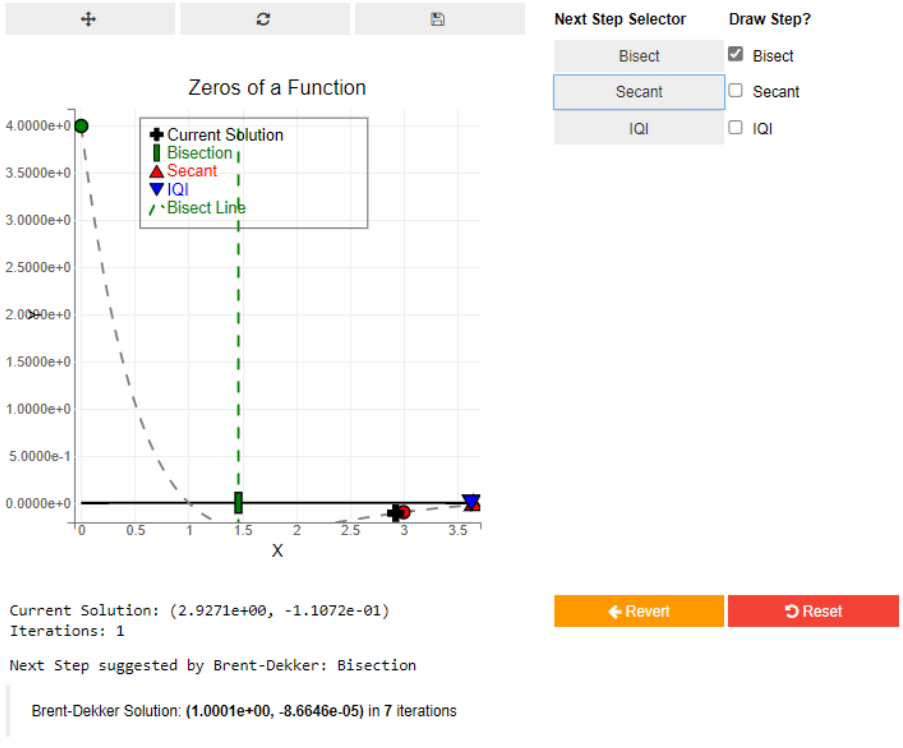
1. Initial State



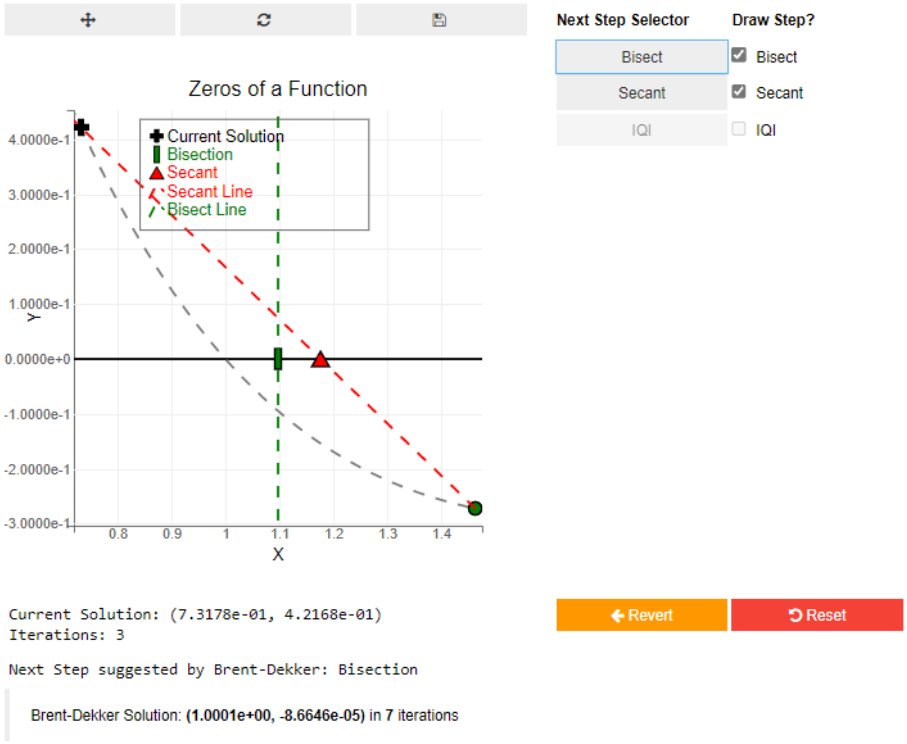
2. Click on bisection checkbox



3. Click Secant Button

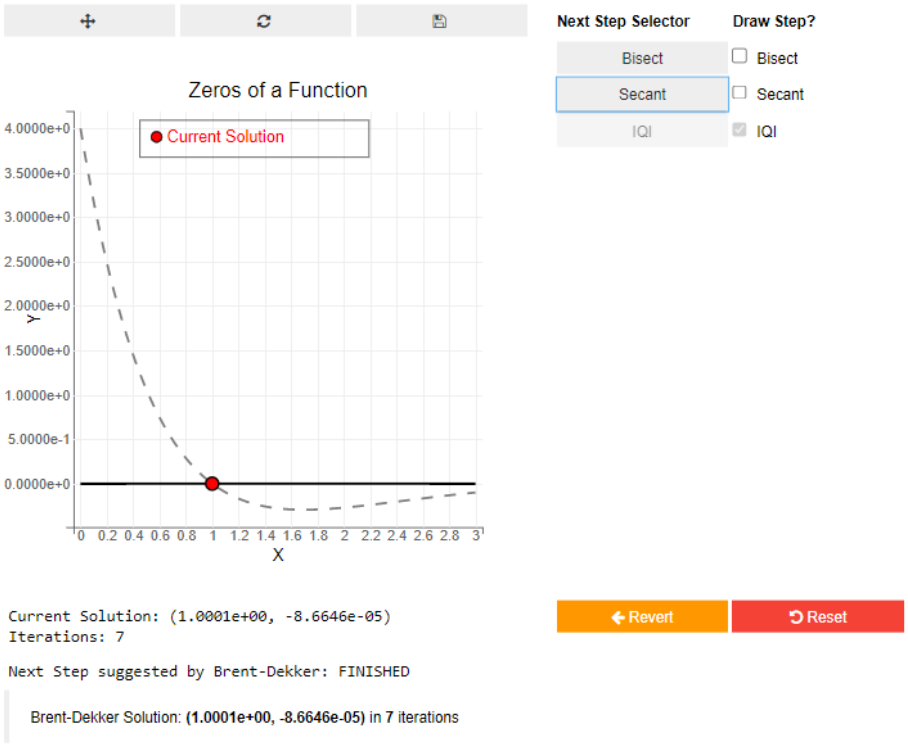


4. Some iterations after with secant checkbox clicked



5. IQI Checkbox Only

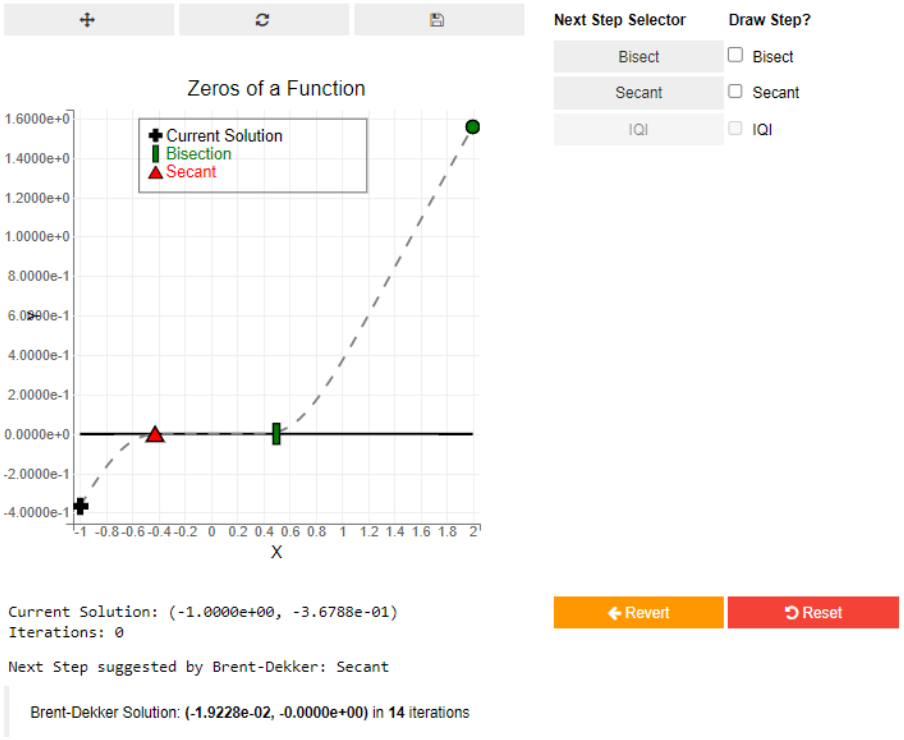
6. Finished after some iterations



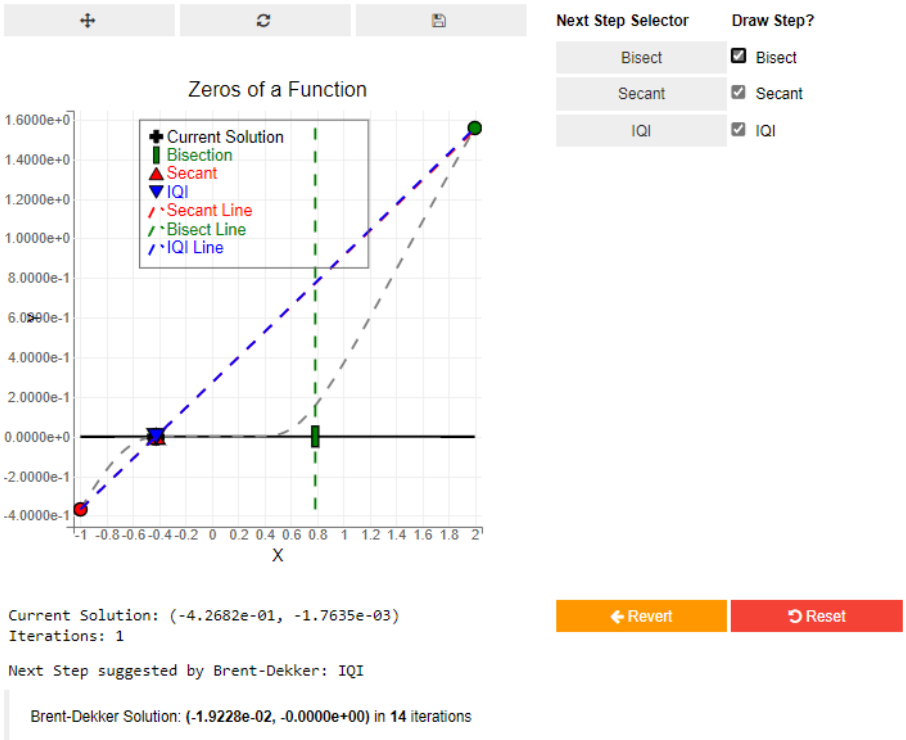
Example 2: With Input Arguments

```
from BNumMet.Visualizers.NonLinearVisualizer import
    NonLinearVisualizer
f2 = lambda x: 0 if abs(x) < 3.8 * 10 ** (-4) else float(x *
    np.exp(-1 / x**2))
interval = (-1, 2)
zerosVisualizer = NonLinearVisualizer(f2,interval)
zerosVisualizer.run()
```

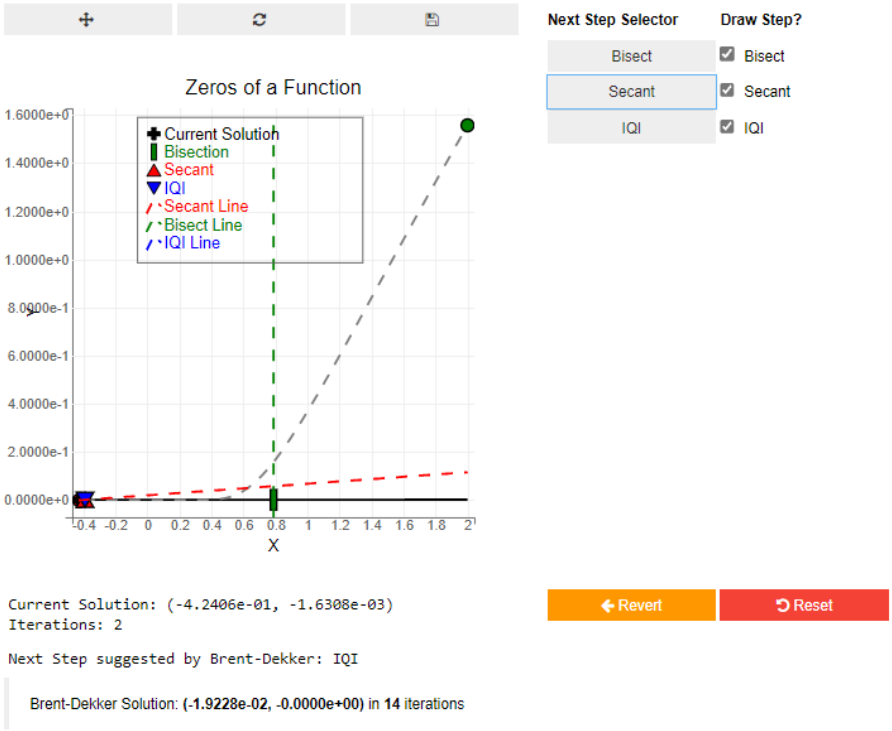
1. Initial State



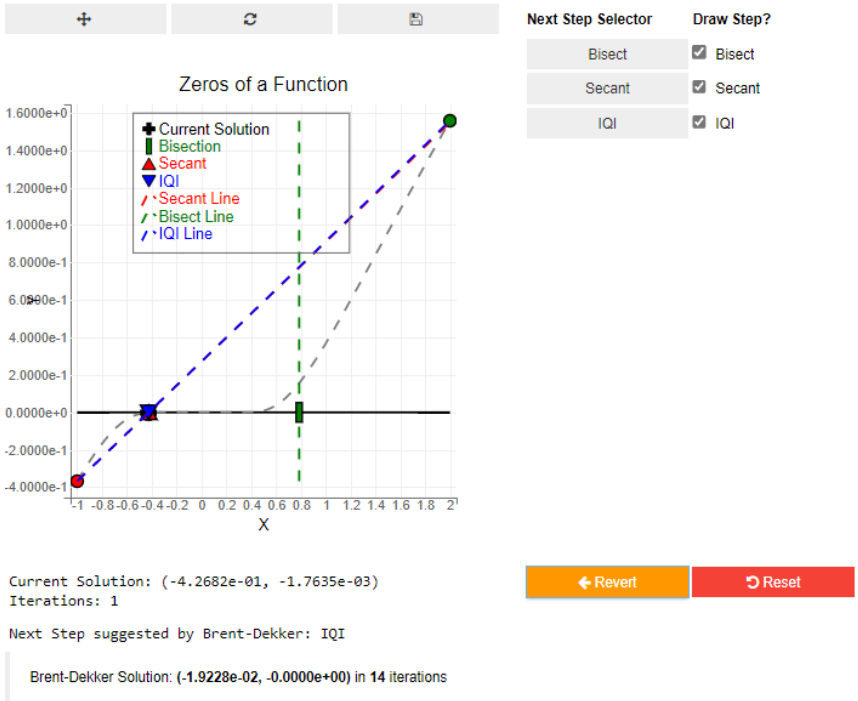
2. All Checkboxes checked



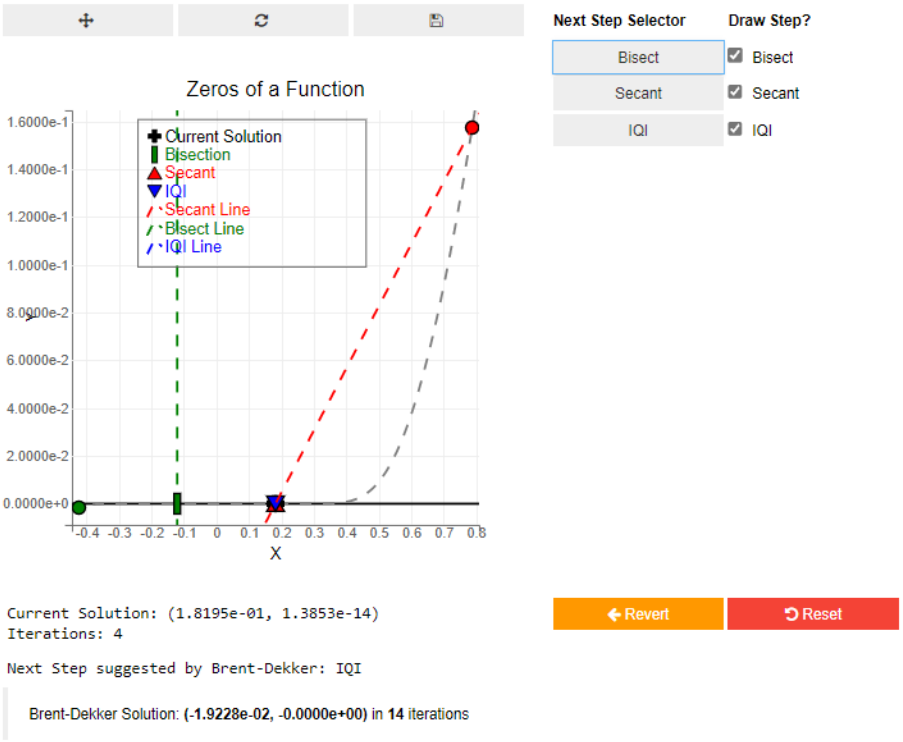
3. Click secant and click select all checkboxes again



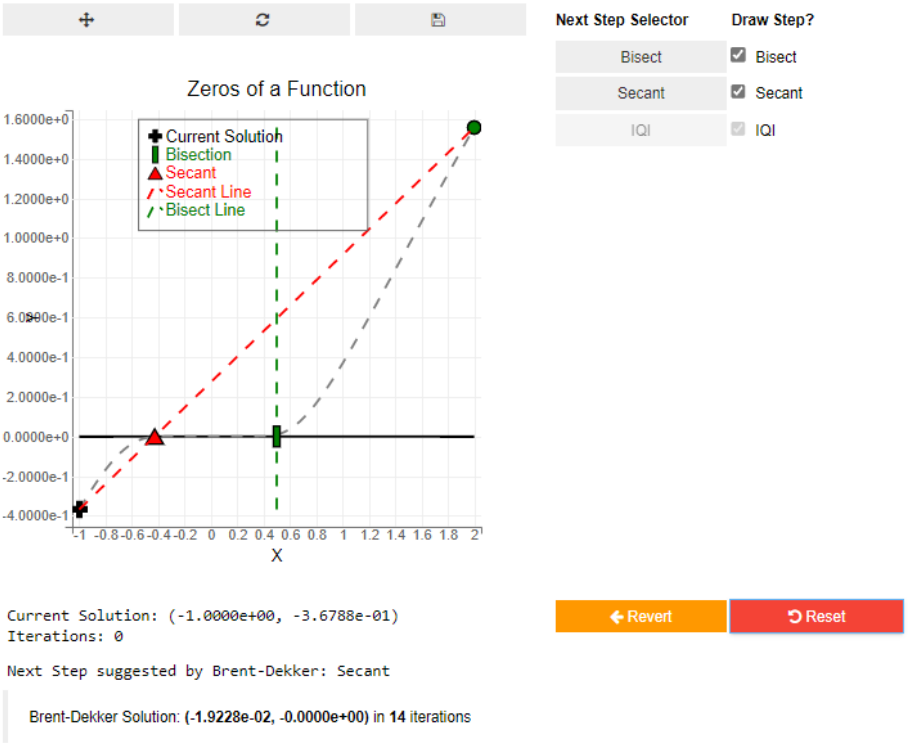
4. Revert Button



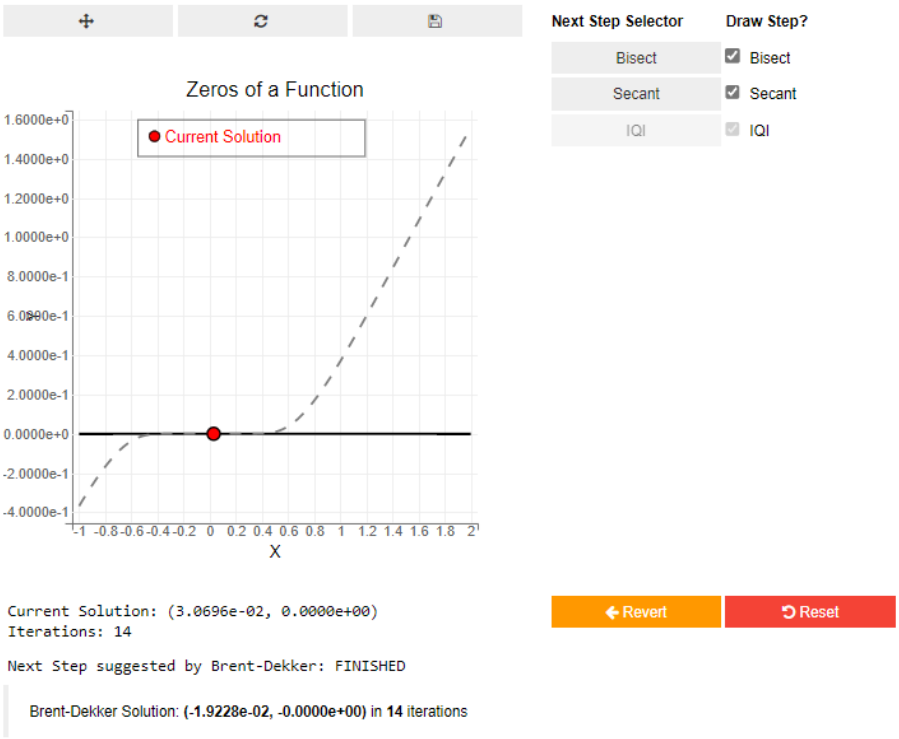
5. Some iterations



6. Reset Button clicked



7. Finished after some iterations

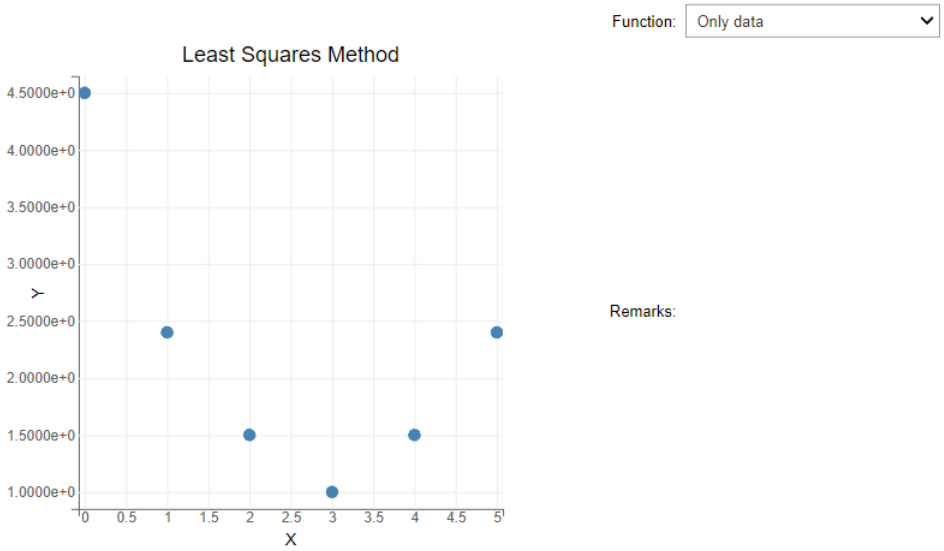


Least Squares Visualizer

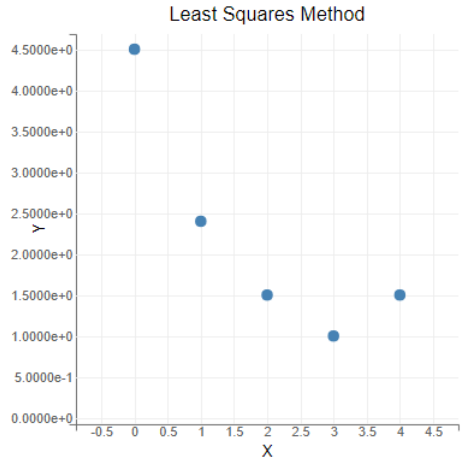
Example 1: Polynomial Fit of Varying degree

```
from BNumMet.Visualizers.LeastSquaresVisualizer import
    LSPVisualizer
xData = np.array([0, 1, 2, 3, 4, 5])
yData = np.array([4.5, 2.4, 1.5, 1, 1.5, 2.4])
lspVisualizer = LSPVisualizer(xData, yData)
lspVisualizer.run()
```

1. Initial State



2. Selector

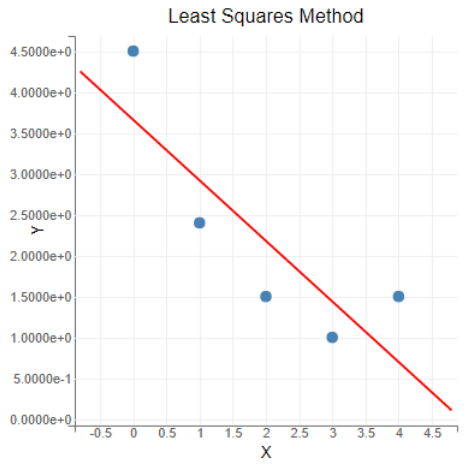


Function: Only data

Only data
Polynomial
Exponential
Sines & Cosines

Remarks:

3. Select Polynomial



Function: Polynomial

Degree: 1

Remarks: The polynomial function is:

$3.66 - 0.74x$

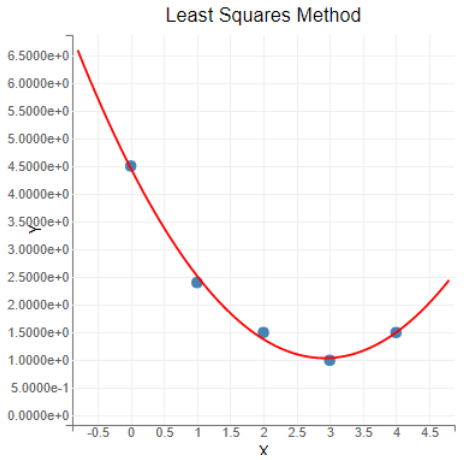
The coefficients of the polynomial function are:

$c_0 = 3.66$

$c_1 = -0.74$

The error is $1.5073 \left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

4. Increment degree by 1 (degree 2)



Function: Polynomial

Degree: 2

Remarks: The polynomial function is:

$4.46 - 2.34x + 0.40x^2$

The coefficients of the polynomial function are:

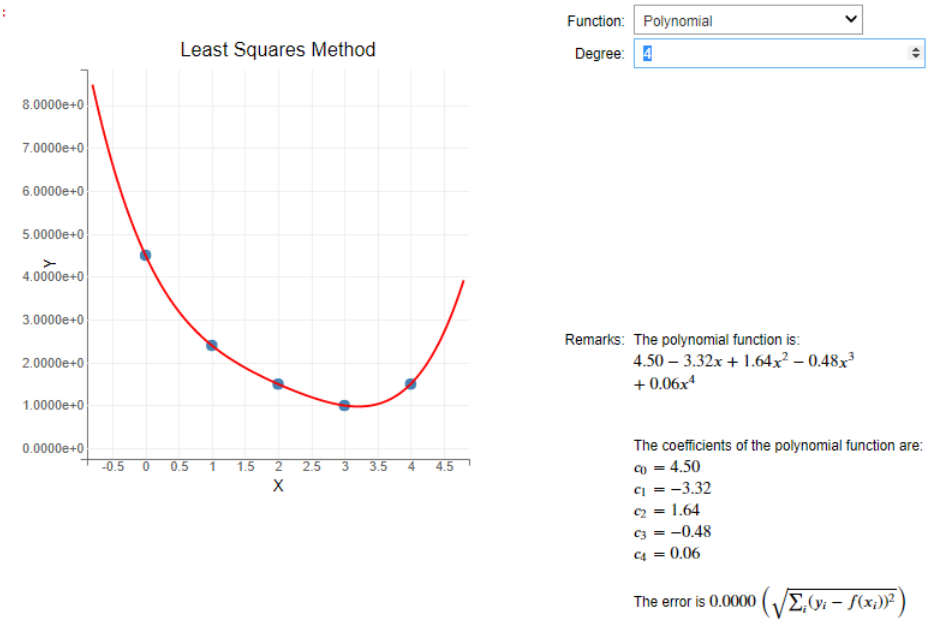
$c_0 = 4.46$

$c_1 = -2.34$

$c_2 = 0.40$

The error is $0.1789 \left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

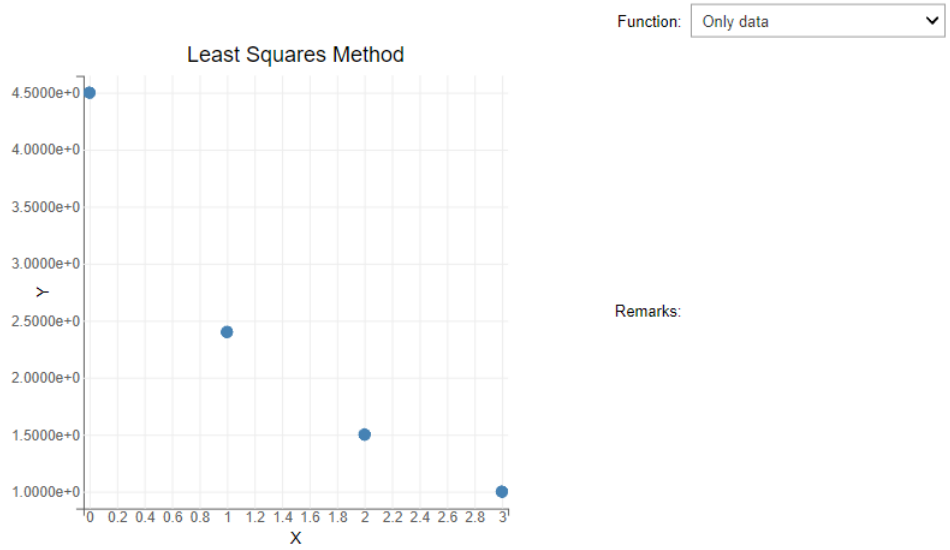
5. Degree 5 (Maximum, cannnot go higher)



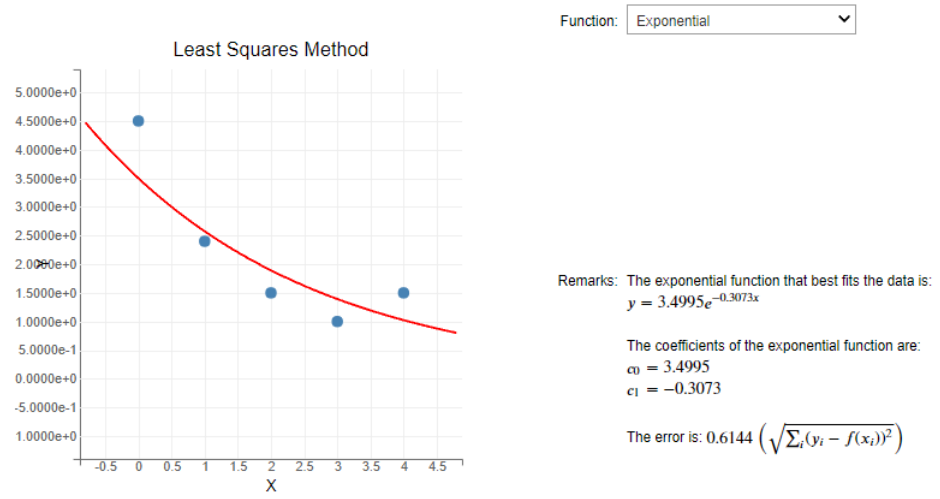
Example 2: No Input Data and Exponential Fit

```
from BNumMet.Visualizers.LeastSquaresVisualizer import
    LSPVisualizer
lspVisualizer = LSPVisualizer()
lspVisualizer.run()
```

1. Initial state



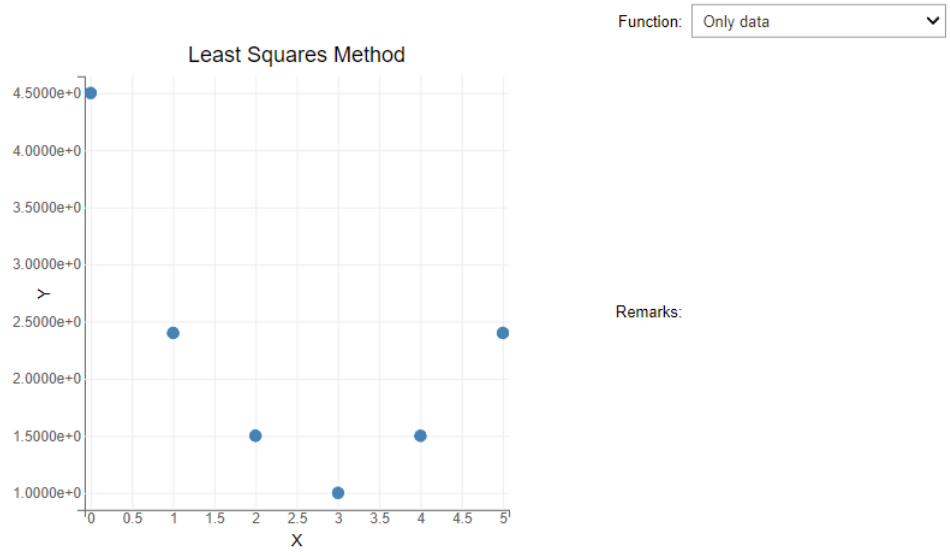
2. Exponential Selection



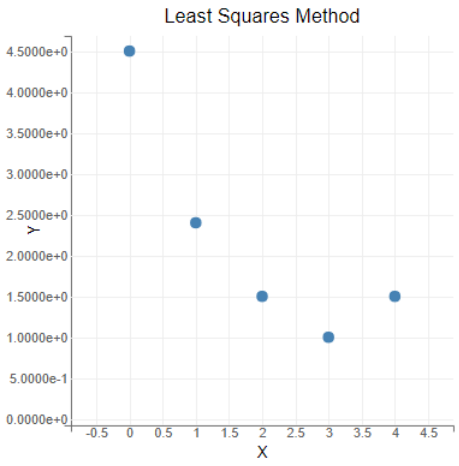
Example 3: Sines and Cosines Fit of Varying degree

```
from BNumMet.Visualizers.LeastSquaresVisualizer import
    LSPVisualizer
xData = np.array([0, 1, 2, 3, 4, 5])
yData = np.array([4.5, 2.4, 1.5, 1, 1.5, 2.4])
lspVisualizer = LSPVisualizer(xData, yData)
lspVisualizer.run()
```

1. Initial State



2. Selector

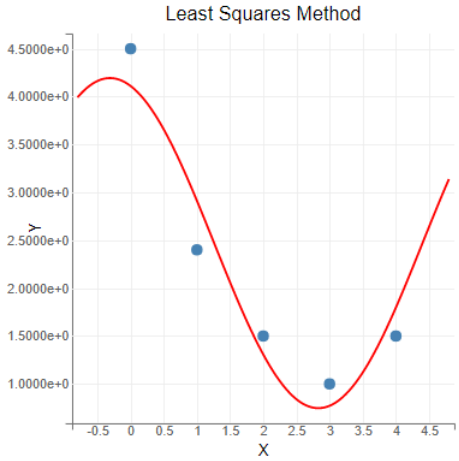


Function: Only data

Only data
Polynomial
Exponential
Sines & Cosines

Remarks:

3. Select Sines & Cosines



Function: Sines & Cosines

Basis Elements: 1

Remarks: The sine and cosine function is:

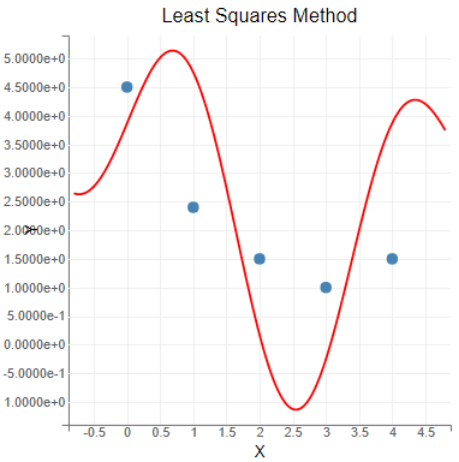
$$2.47 - 0.53\sin(1 \cdot x) + 1.64\cos(1 \cdot x)$$

The coefficients of the sine and cosine function are:

$c_0 = 2.47$
$c_1 = -0.53$
$c_2 = 1.64$

The error is: $0.7692\left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

4. Set degree to the maximum



Function: Sines & Cosines

Basis Elements: 2

Remarks: The sine and cosine function is:

$$2.61 - 0.77\sin(1 \cdot x) + 1.78\cos(1 \cdot x) - 0.53\sin(2 \cdot x) + 0.11\cos(2 \cdot x)$$

The coefficients of the sine and cosine function are:

$c_0 = 2.61$
$c_1 = -0.77$
$c_2 = 1.78$
$c_3 = -0.53$
$c_4 = 0.11$

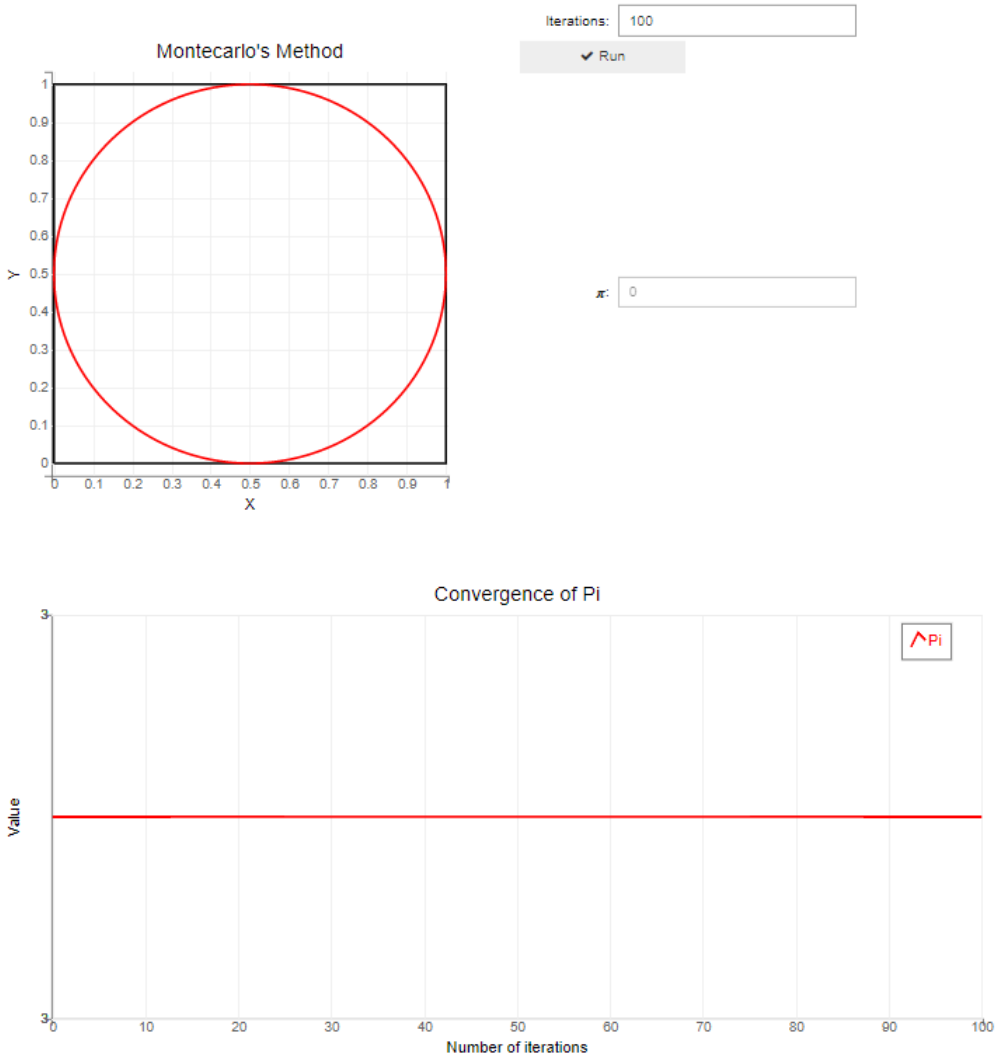
The error is: $3.8623\left(\sqrt{\sum_i (y_i - f(x_i))^2}\right)$

Random Numbers Visualizer

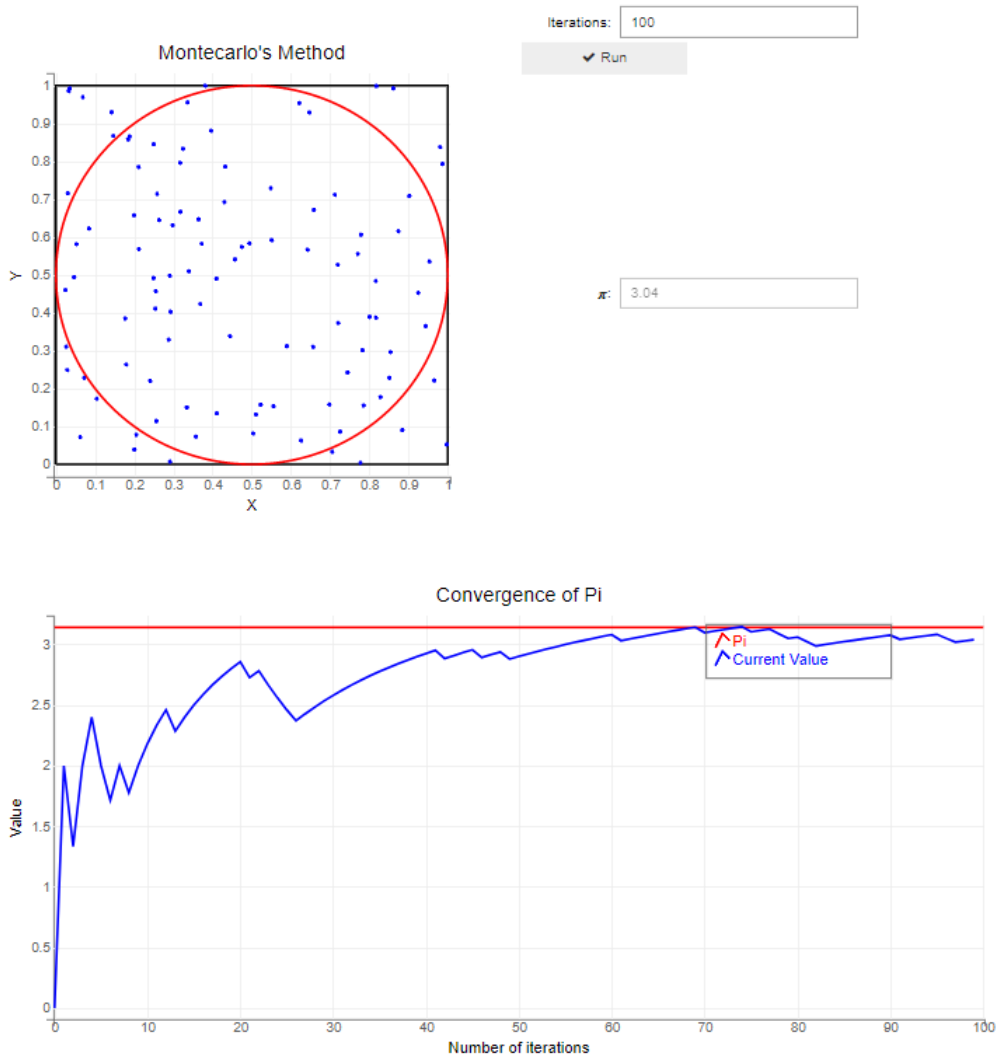
Example 1: No input arguments

```
from BNumMet.Visualizers.RandomVisualizer import
    RandomVisualizer
randomVisualizer = RandomVisualizer()
randomVisualizer.run()
```

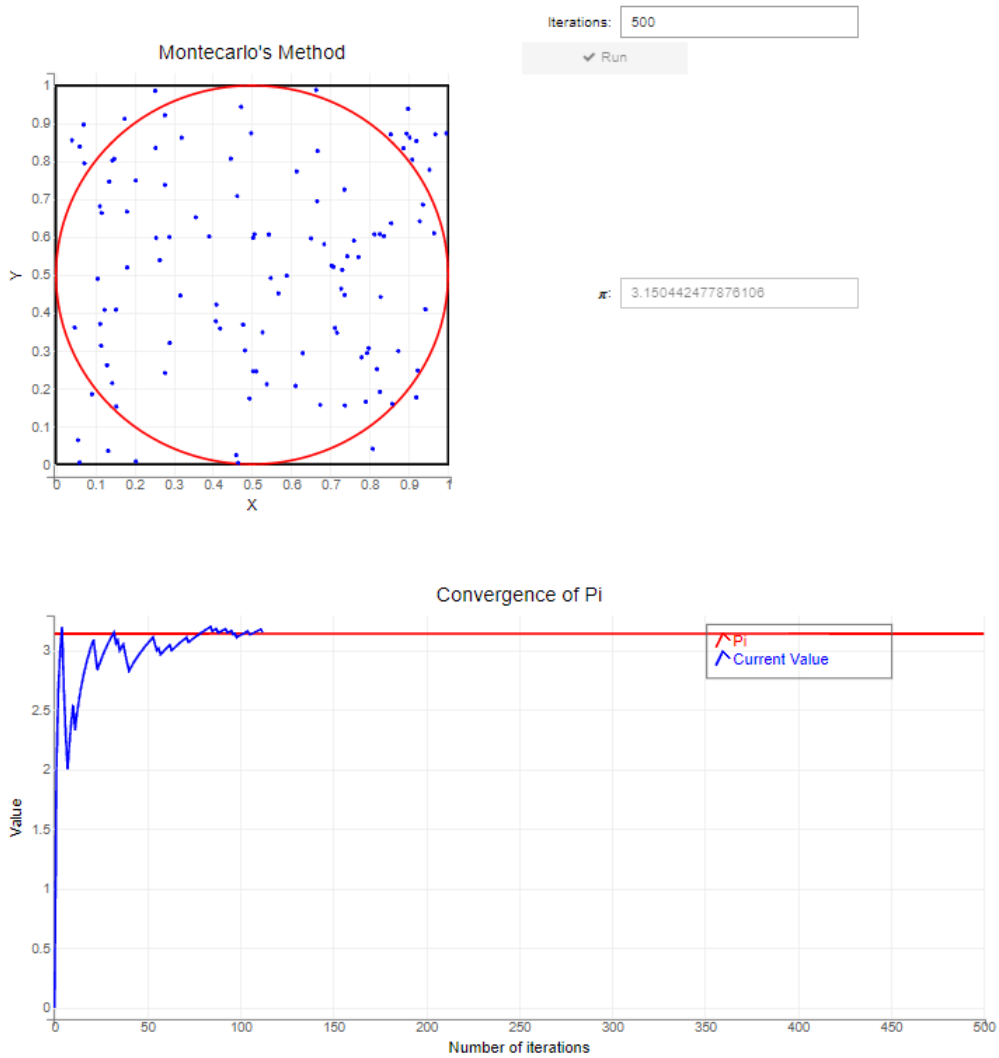
1. Initial State



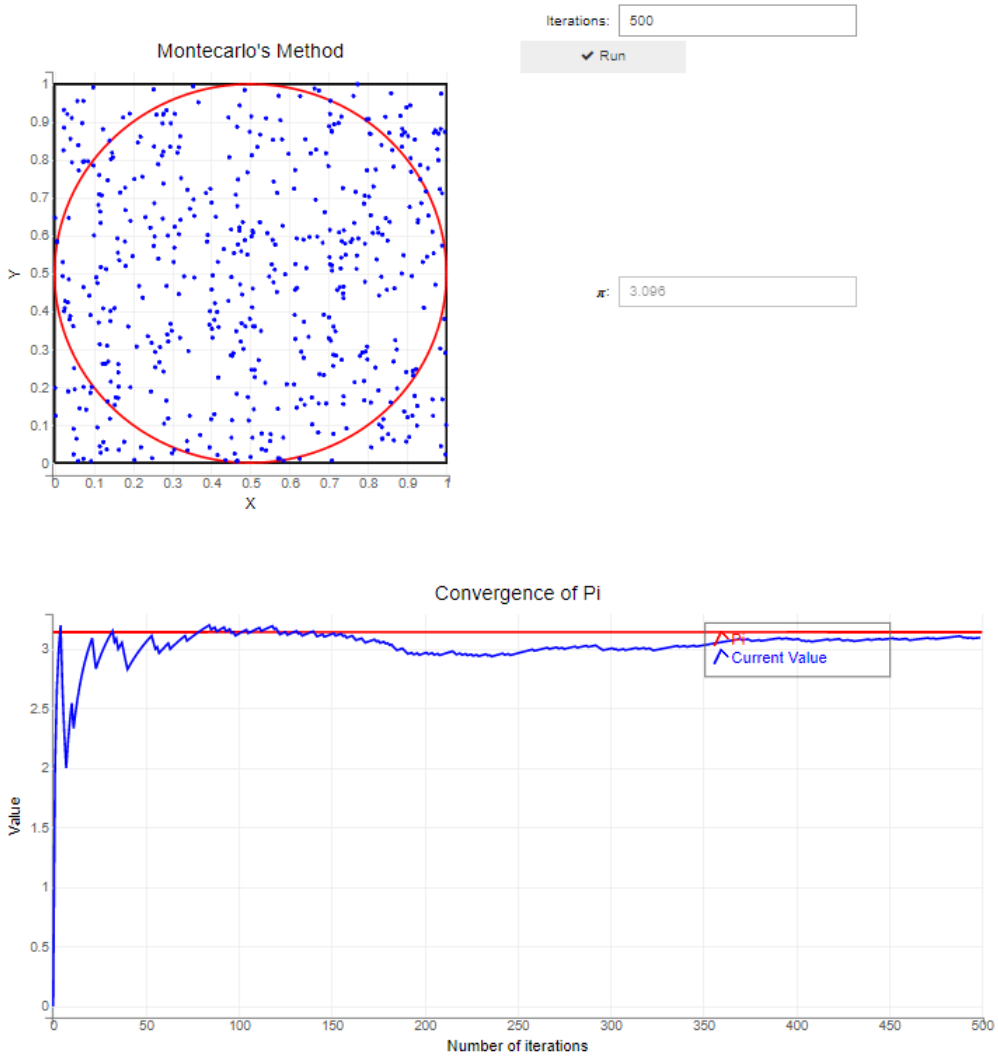
2. Clicked played with 100 default points



3. Change number of points to 500 and clicked play



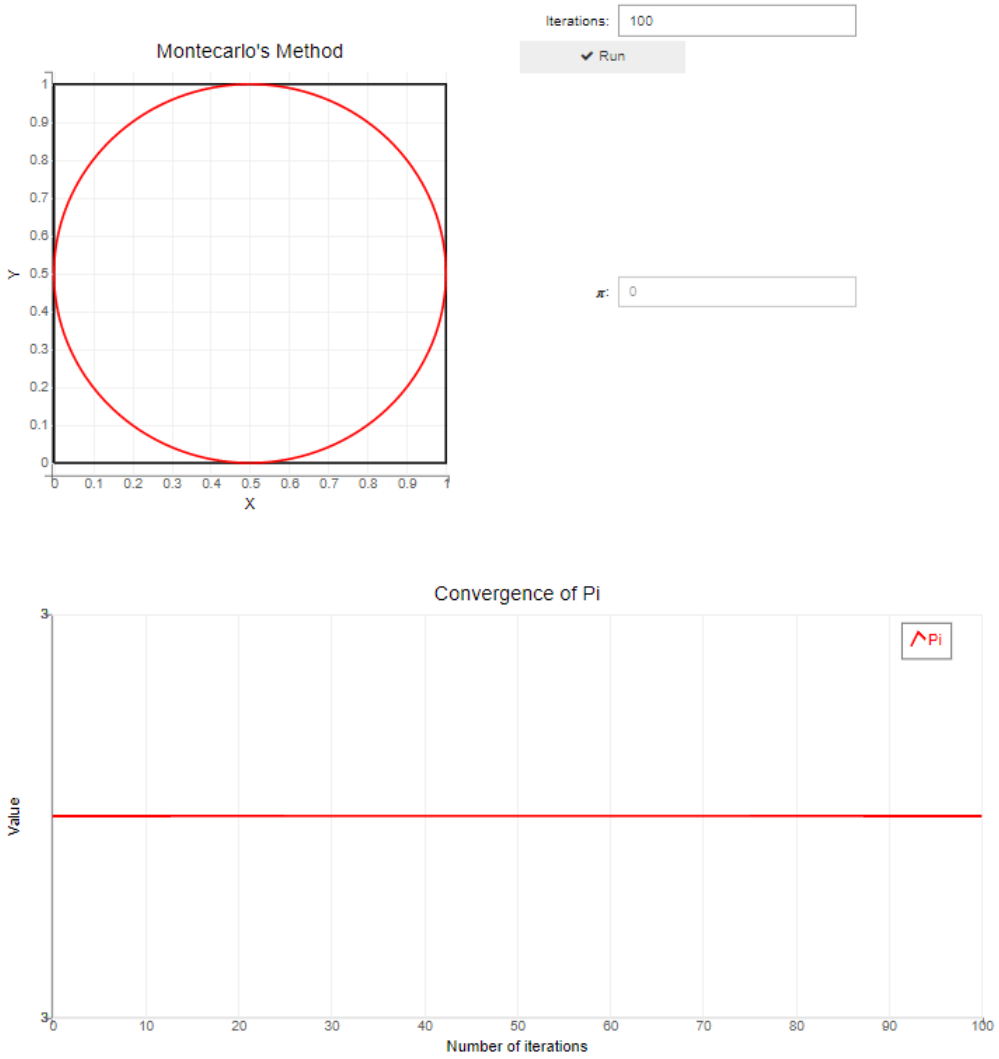
4. Final of animation playing



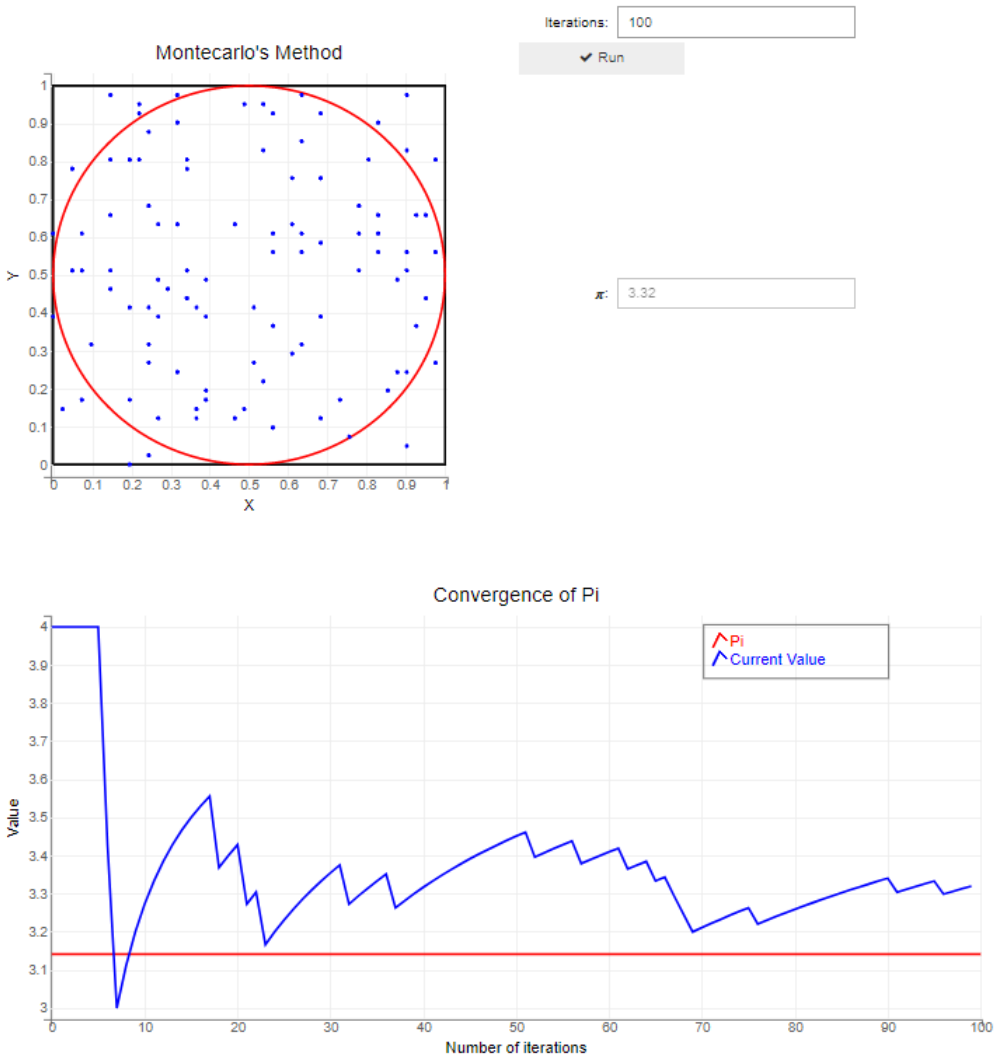
Example 2: Input arguments and Ill-Generator

```
from BNumMet.Visualizers.RandomVisualizer import
    RandomVisualizer
from BNumMet.Random import marsaglia_rand,
    clear_marsaglia_vars

clear_marsaglia_vars()
randomFunc = lambda: marsaglia_rand(
    base=10, lag_r=2, lag_s=1, carry=0, seed_tuple=(0, 1)
)
randomVisualizer = RandomVisualizer(randomFunc)
randomVisualizer.run()
```

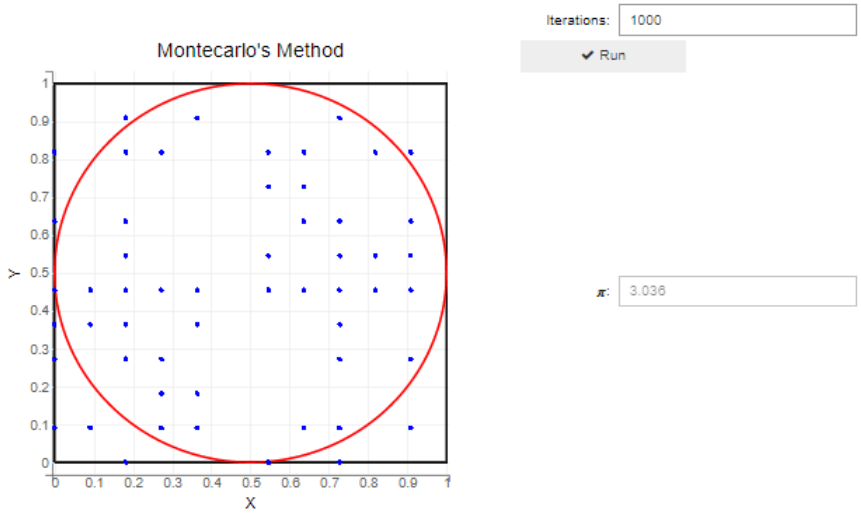
1. Initial State

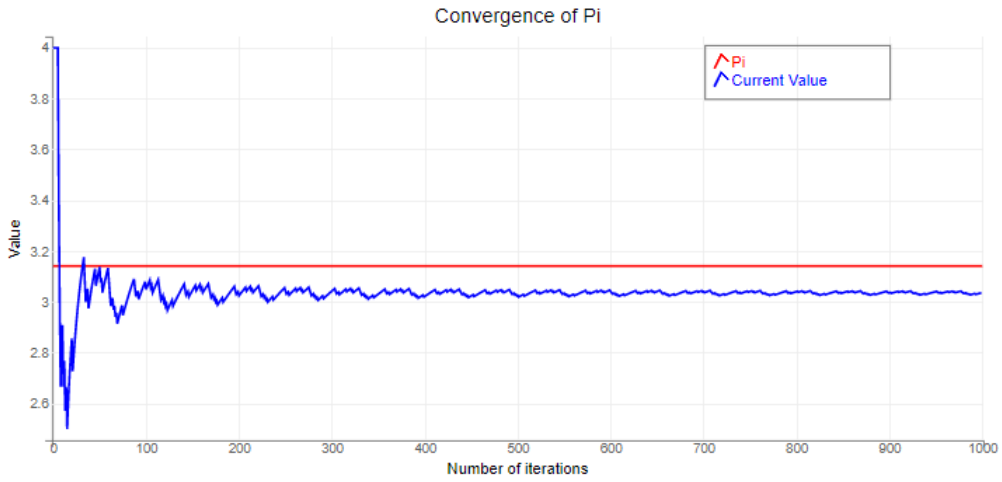


2. Clicked played with 100 default points



3. Change number of points to 1000 and clicked play


Montecarlo's Method
Iterations:
1000
Run
π:
3.036
Y
X


Convergence of Pi
Pi
Current Value
Value
Number of iterations

APPENDIX: GITHUB ACTIONS

Latex Compilation

```
  name: Latex Compilation

# Only trigger, when the build workflow succeeded
on:
  push:
    paths:
      - 'Latex/**'
      - '.github/workflows/LatexCompilation.yml'

jobs:
  build_latex:
    runs-on: ubuntu-latest
    steps:
      - name: Set up Git repository
        uses: actions/checkout@v3

      - name: Compile LaTeX document
        uses: xu-cheng/latex-action@v2
        with:
          root_file: main.tex
          working_directory: ./Latex

      - name: Upload PDF file
        uses: actions/upload-artifact@v3
        with:
          name: PDF
          path: ./Latex/main.pdf
```

Python Tests

```
# This workflow will install Python dependencies, run tests
    and lint with a variety of Python versions
# For more information see: https://docs.github.com/en/
    actions/automating-builds-and-tests/building-and-testing-
    python

name: Python Tests

on:
  push:
    branches:
```

```
      - main
    paths:
      - 'Python_BNumMet/src/**'
      - 'Python_BNumMet/Demos/**'
      - 'Python_BNumMet/tests/**'
      - '.github/workflows/PythonTests.yml'
      - 'Python_BNumMet/requirements.txt'
      - 'Python_BNumMet/setup.py'
      - 'Python_BNumMet/pyproject.toml'
      - 'Python_BNumMet/setup.cfg'
      - 'Python_BNumMet/tox.ini'

jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]
        python-version: ['3.8', '3.9', '3.10', '3.11']

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with:
        python-version: ${{ matrix.python-version }}

    - name: Install dependencies
      run: |
        cd "Python_BNumMet"
        # Upgrade pip
        python -m pip install --upgrade pip

        # tox-gh-actions is a plugin for tox that makes it
            work with GitHub Actions
        pip install tox tox-gh-actions

        # for linting
        pip install flake8

        # Requirements
        pip install -r requirements.txt

    - name: Lint with flake8
      run: |
        cd "Python_BNumMet"

        # stop the build if there are Python syntax errors or
            undefined names
        flake8 . --count --select=E9,F63,F7,F82 --show-source
            --statistics
```

```
57
58          # exit-zero treats all errors as warnings. The GitHub
                editor is 127 chars wide
59          flake8 . --count --exit-zero --max-complexity=10 --
               max-line-length=127 --statistics
60
61      - name: Test with pytest
62        run: |
63          cd "Python_BNumMet"
64          # Run tests
65          tox
```

Python Publish

```
1   # This workflow will upload a Python Package using Twine when
        a release is created
2   # For more information see: https://docs.github.com/en/
        actions/automating-builds-and-tests/building-and-testing-
        python#publishing-to-package-registries
3
4   # This workflow uses actions that are not certified by GitHub
        .
5   # They are provided by a third-party and are governed by
6   # separate terms of service, privacy policy, and support
7   # documentation.
8
9   name: Upload Python Package
10
11  on:
12    workflow_run:
13      workflows: ["Python Tests"]
14      types:
15        - completed # Only run, when the tests workflow Python
              Tests is completed
16    push:
17      branches:
18        - main
19      paths:
20        - 'Python_BNumMet/VERSION' # Only trigger, when the
              version file is changed
21
22
23
24  permissions: # https://docs.github.com/en/actions/reference/
        authentication-in-a-workflow#permissions-for-the-
        github_token
25    contents: read
26
```

```yaml
jobs:
  deploy: # https://packaging.python.org/guides/publishing-package-distribution-releases-using-github-actions-ci-cd-workflows/
    runs-on: ubuntu-latest
    if: ${{ github.event.workflow_run.conclusion == 'success' }} # Only run, if the tests succeeded

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v3
      with:
        python-version: '3.x'
    - name: Install dependencies
      run: |
        cd "Python_BNumMet"
        python -m pip install --upgrade pip
        pip install build
    - name: Build package
      run: |
        cd "Python_BNumMet"
        python -m build
    - name: Publish package
      uses: pypa/gh-action-pypi-publish@release/v1
      with:
        user: __token__
        password: ${{ secrets.PYPI_TOKEN }}
        packages-dir: Python_BNumMet/dist/
```

APPENDIX: SONARQUBE REPORT

BNUMMET

VERSION 1

Code analysis

By: default

2023-03-26

CONTENT

INTRODUCTION

This document contains results of the code analysis of BNumMet.

CONFIGURATION

- Quality Profiles
  - Names: Sonar way [Python]; Sonar way [XML];
  - Files: AYZE7i3U2dITZFwgEli2.json; AYZE7jaV2dITZFwgEmPx.json;
- Quality Gate
  - Name: Sonar way
  - File: Sonar way.xml

SYNTHESIS

ANALYSIS STATUS

Reliability	Security	Security Review	Maintainability
A	A	A	A

QUALITY GATE STATUS

Quality Gate Status	Passed

Metric	Value
Reliability Rating on New Code	OK
Security Rating on New Code	OK
Maintainability Rating on New Code	OK
Coverage on New Code	OK
Duplicated Lines (%) on New Code	OK

METRICS

Coverage	Duplication	Comment density	Median number of lines of code per file	Adherence to coding standard
98.2 %	0.0 %	43.4 %	206.0	99.7 %

TESTS

Total	Success Rate	Skipped	Errors	Failures

BNumMet

95	**100.0 %**	**0**	**0**	**0**

DETAILED TECHNICAL DEBT

Reliability	Security	Maintainability	Total
-	-	0d 4h 5min	**0d 4h 5min**

METRICS RANGE

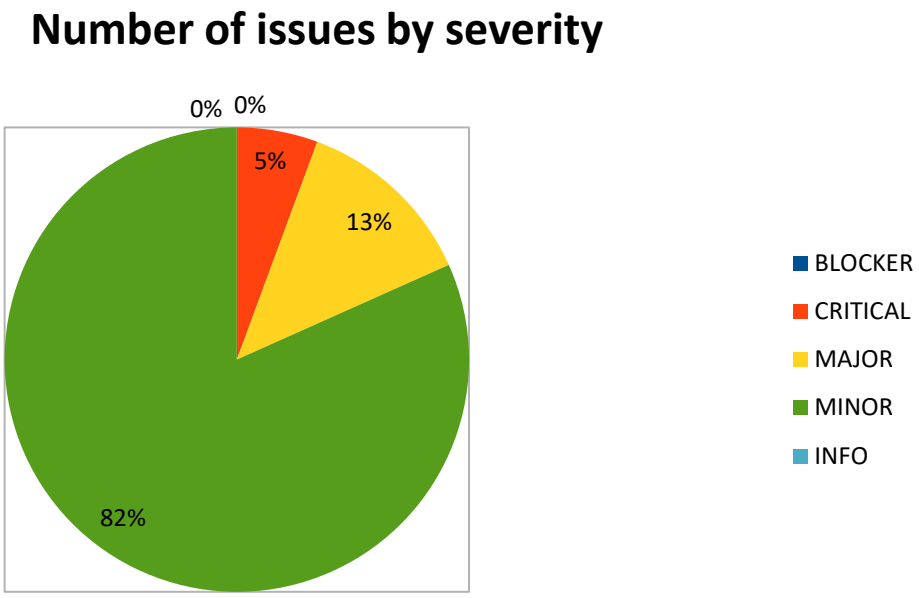
	Cyclomatic Complexity	Cognitive Complexity	Lines of code per file	Comment density (%)	Coverage	Duplication (%)
Min	0.0	0.0	0.0	22.2	95.5	0.0
Max	295.0	269.0	2028.0	74.1	100.0	0.0

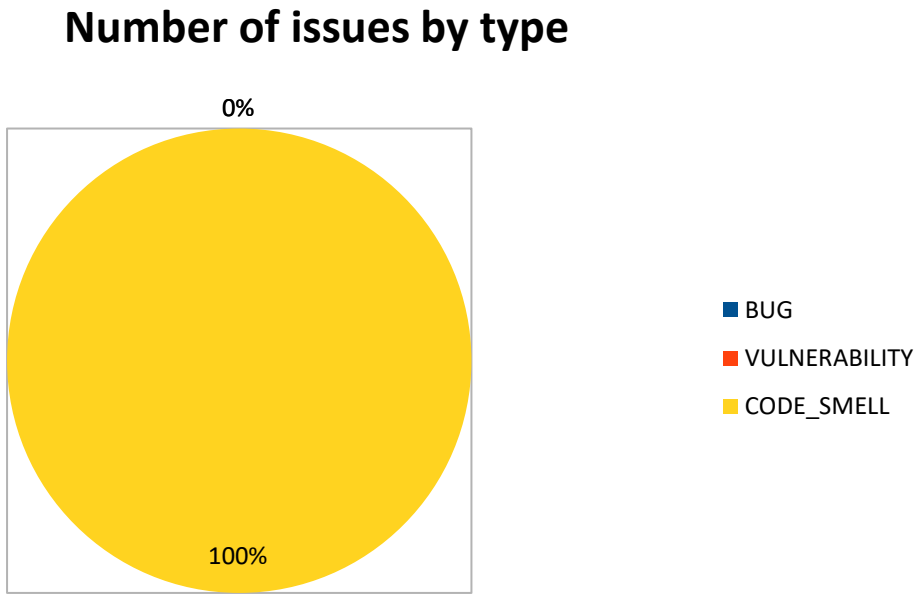
VOLUME

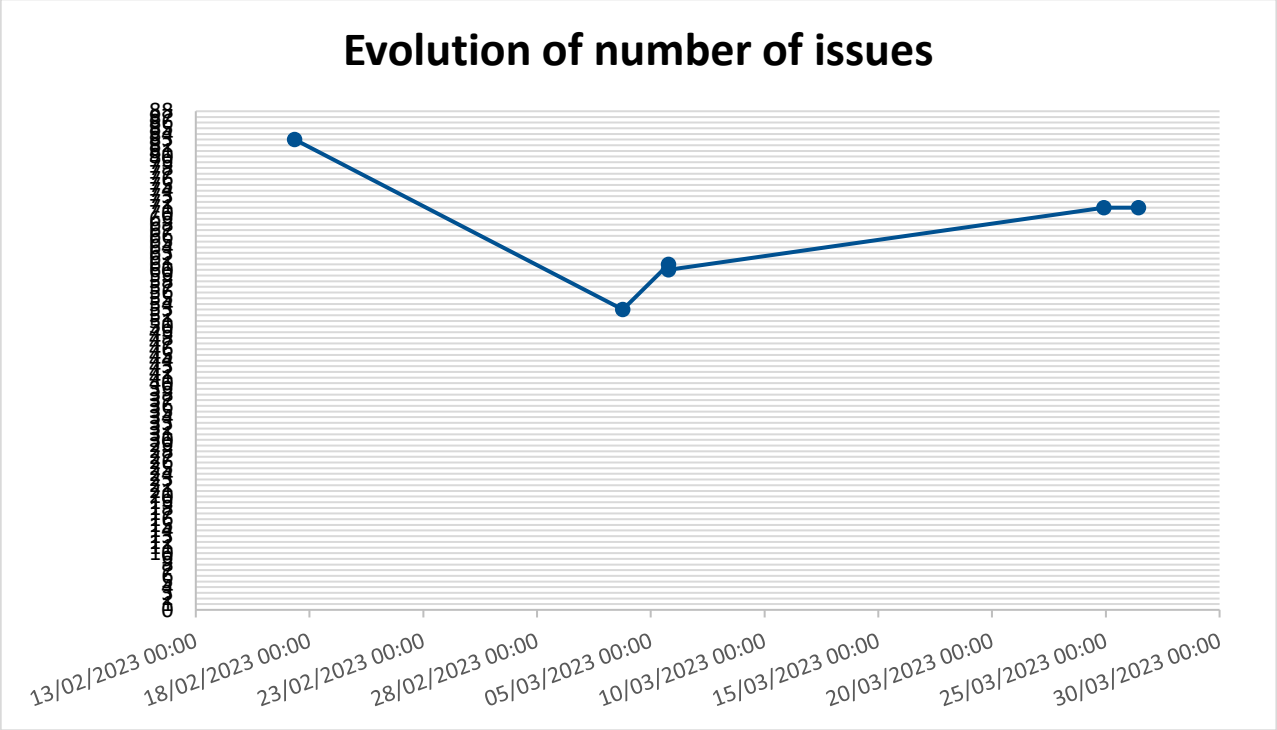
Language	Number
Python	2028
Total	2028

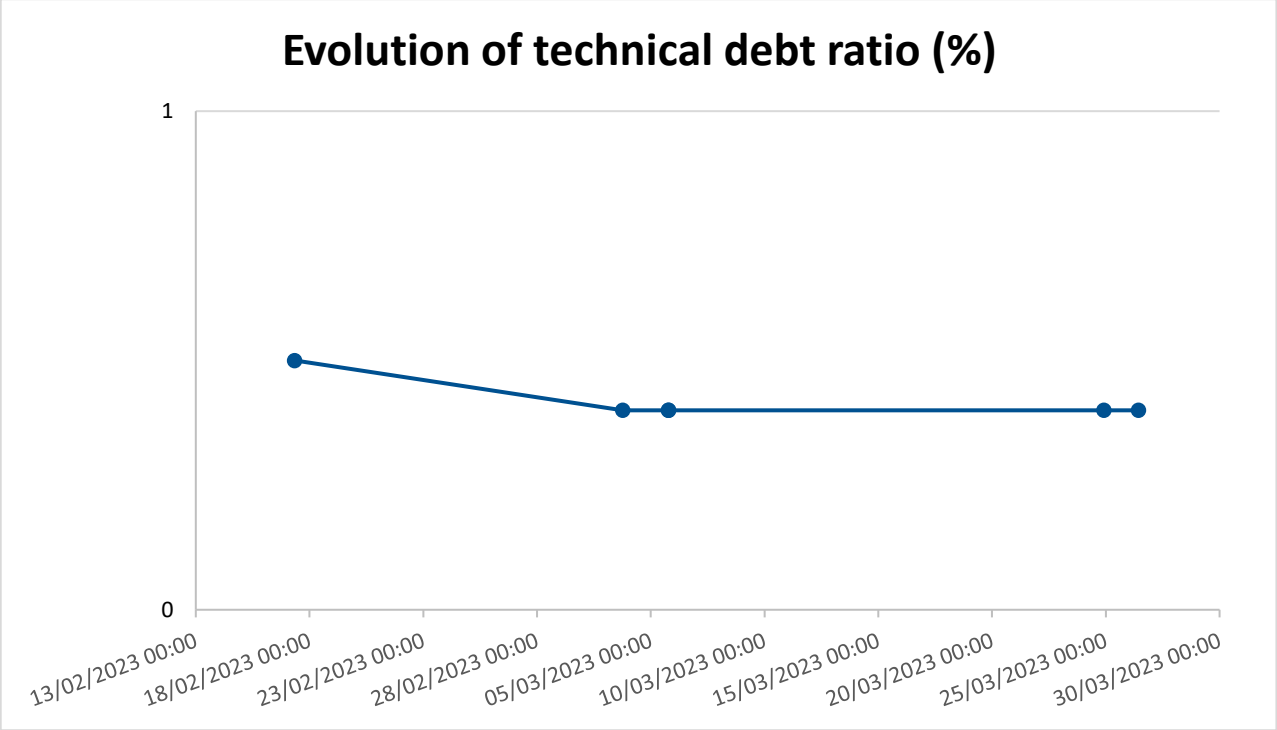
ISSUES

CHARTS






Evolution of number of issues
13/02/2023 00:00
18/02/2023 00:00
23/02/2023 00:00
28/02/2023 00:00
05/03/2023 00:00
10/03/2023 00:00
15/03/2023 00:00
20/03/2023 00:00
25/03/2023 00:00
30/03/2023 00:00


Evolution of technical debt ratio (%)
1
0
13/02/2023 00:00
18/02/2023 00:00
23/02/2023 00:00
28/02/2023 00:00
05/03/2023 00:00
10/03/2023 00:00
15/03/2023 00:00
20/03/2023 00:00
25/03/2023 00:00
30/03/2023 00:00

BNumMet

ISSUES COUNT BY SEVERITY AND TYPE

Type / Severity	INFO	MINOR	MAJOR	CRITICAL	BLOCKER
BUG	0	0	0	0	0
VULNERABILITY	0	0	0	0	0
CODE_SMELL	0	58	9	4	0

ISSUES LIST

Name	Description	Type	Severity	Number
String literals should not be duplicated	Duplicated string literals make the process of refactoring error-prone, since you must be sure to update all occurrences. On the other hand, constants can be referenced from many places, but only need to be updated in a single place. Noncompliant Code Example With the default threshold of 3: def run(): prepare("this is a duplicate") # Noncompliant - "this is a duplicate" is duplicated 3 times execute("this is a duplicate") release("this is a duplicate") Compliant Solution ACTION_1 = "action1" def run(): prepare(ACTION_1) execute(ACTION_1) release(ACTION_1) Exceptions No issue will be raised on: duplicated string in decorators strings with less than 5 characters strings with only letters, numbers and underscores @app.route("/api/users/", methods=['GET', 'POST', 'PUT']) def users(): pass @app.route("/api/projects/", methods=['GET', 'POST', 'PUT']) # Compliant def projects(): pass	CODE_SMELL	CRITICAL	1
Cognitive Complexity of functions should not be too high	Cognitive Complexity is a measure of how hard the control flow of a function is to understand. Functions with high Cognitive Complexity will be difficult to maintain. See Cognitive Complexity	CODE_SMELL	CRITICAL	3
Sections of code should not be commented out	Programmers should not comment out code as it bloats programs and reduces readability. Unused code should be deleted and can be retrieved from source control history if required.	CODE_SMELL	MAJOR	7
Function names should comply with a naming convention	Shared coding conventions allow teams to collaborate efficiently. This rule checks that all function names match a provided regular expression. Noncompliant Code Example With the default provided regular expression:	CODE_SMELL	MAJOR	2

	^[a-z_][a-z0-9_]*$ def MyFunction(a,b): ... Compliant Solution def my_function(a,b): ...			
Method names should comply with a naming convention	Sharing some naming conventions is a key point to make it possible for a team to efficiently collaborate. This rule allows to check that all method names match a provided regular expression. Noncompliant Code Example With default provided regular expression: ^[a-z_][a-z0-9_]*$ class MyClass: def MyMethod(a,b): ... Compliant Solution class MyClass: def my_method(a,b): ...	CODE_SMELL	MINOR	6
Field names should comply with a naming convention	Sharing some naming conventions is a key point to make it possible for a team to efficiently collaborate. This rule allows to check that field names match a provided regular expression. Noncompliant Code Example With the default regular expression ^[_a-z][_a-z0-9]*$: class MyClass: myField = 1 Compliant Solution class MyClass: my_field = 1	CODE_SMELL	MINOR	28
Local variable and function parameter names should comply with a naming convention	Shared naming conventions allow teams to collaborate effectively. This rule raises an issue when a local variable or function parameter name does not match the provided regular expression. Exceptions Loop counters are ignored by this rule. for i in range(limit): # Compliant print(i)	CODE_SMELL	MINOR	24

SECURITY HOTSPOTS

SECURITY HOTSPOTS COUNT BY CATEGORY AND PRIORITY

Category / Priority	LOW	MEDIUM	HIGH
LDAP Injection	0	0	0
Object Injection	0	0	0
Server-Side Request Forgery (SSRF)	0	0	0
XML External Entity (XXE)	0	0	0
Insecure Configuration	0	0	0
XPath Injection	0	0	0
Authentication	0	0	0
Weak Cryptography	0	0	0
Denial of Service (DoS)	0	0	0
Log Injection	0	0	0
Cross-Site Request Forgery (CSRF)	0	0	0
Open Redirect	0	0	0
Permission	0	0	0
SQL Injection	0	0	0
Encryption of Sensitive Data	0	0	0
Traceability	0	0	0
Buffer Overflow	0	0	0
File Manipulation	0	0	0
Code Injection (RCE)	0	0	0

Cross-Site Scripting (XSS)	0	0	0
Command Injection	0	0	0
Path Traversal Injection	0	0	0
HTTP Response Splitting	0	0	0
Others	0	0	0

SECURITY HOTSPOTS LIST