March 2015

Visualization for Finite Element Method Education

Donald Leo Bourque
Worcester Polytechnic Institute

Stephen J. Kelly
Worcester Polytechnic Institute

Worcester Polytechnic Institute

Interdisciplinary Qualifying Project

Visualization for Finite Element Method Education

Authors:
Donald Bourque
Stephen Kelly

Date: December 4, 2013
Advisor: Prof. Nima Rahbar

Abstract

In this project, common practices for visualizing scientific data were studied. In addition, the science of cognition and data display was reviewed. The results of this investigation was applied to augment a Civil Engineering introductory course on Finite Element Method at WPI. Software enhancements allowed three dimensional visualization for simulation of engineering structures. The research on cognition and data graphics was used to improve understanding of these visual aids. The plotting function, developed in MATLAB and Julia environments during the course of this project, can help all students visualize the results of their numerical codes.

Contents

List of Figures

List of Tables

1 Introduction

Visualization is an integral part of conveying scientific data. In the educational process, visualization serves a key role in illustrating objectives and results, increasing the connection and understanding of course topics. With the growth of computing technology in the past few decades, it has become increasingly more feasible to perform numerical analysis and generate intricate and dynamic visuals.

An important numerical method in engineering has been the Finite Element Method, often abbreviated FEM. When this technique is applied to solve problems it is called Finite Element Analysis, or FEA.[1] These abbreviations will be used throughout this paper. Finite Element Analysis is used extensively to study structures and materials. WPI offers courses in mechanics of materials which study the analytical construction and modeling of engineering problems. These courses traditionally blend many problem solving techniques. [2] We will start by considering the cognitive aspects of these courses, particularly on the visualization of computational results.

The objective is to further understand the role that visualization plays in the educational process of the Finite Element Method. We will be focusing on the final aspect of visually presenting the resulting numerical data. In order to accomplish this we will survey the literature relevant to the role of visualization in education. The literature will span cognitive science to graphics design. After building a collection of cognitive considerations, we will look at varying software packages. It is important to note that the great diversity in numerical software leads to many considerations.

2 Literature Review

2.1 The Theory of the Finite Element Method

FEA traces its roots back to the work of Richard Courant, with his paper *Variational methods for the solution of problems of equilibrium and vibrations*, which established the theoretical basis for the method.[3] In this seminal work, he lays out the mathematical framework for reducing a problem existing on a boundary into a variational problem existing on a discrete mesh. This method for solving partial differential equations drew on many previous techniques by earlier mathematicians. A variational problem is concerned with the minimization and maximization of integrals of unknown functions. An integral of an unknown function is called a "functional". In most systems, Hooke's law applies, thus creating a linear system if all properties are uniform. In most natural systems, the properties are non-uniform, and therefore non-linear. Klaus-Jurgen Bathe's book on non-linear FEM elaborates on the differences from linear problems and numerical techniques available for such systems.[4]

Many of the courses concerned with material mechanics and Finite Element Analysis at WPI are taught for linear systems. This makes the material accessible to the students and presumes a knowledge of basic linear algebra. In addition, students are taught how to construct problems from an analytical symbolic representation into a discrete problem for computation. As such, we will carry this consideration into our analysis of software packages.

Table 1: Representational Frameworks for Shape Information

Name	Purpose	Primitives
Images	Represent continuous color intensity.	Discrete hue and saturation values
Sketches	Geometry and organization of color intensity changes.	Boundaries (lines), paths, and discontinuities.
2.5D sketch	Describe orientation, field of view, and depth of surfaces	Surfaces, depth, and discontinuities.
3D model	Describe orientation of shapes and volumes in a hierarchical manner	Meshes, axes, and shape primitives

2.2 Visual Cognition

A significant portion of this project will consist of refining the visual elements used in an educational process. Multiple books were found to be of great use in understanding the cognitive reactions people have to visual representations. Robert Solso's book, *Cognition and the Visual Arts* helped in understanding both the cognitive and psychological effects visuals can have on the mind.[5] While the book is ultimately a treatise on how cognition can be easily understood through the visual arts, the material still proves relevant to our study. In particular, Solso starts with an explanation of a cognitive information processing model, establishing a lens through which we can understand how reactions arise from observation. This model is validated through analysis of both the physical structure of the human eye and the physics of light. A historical account of artistic techniques is presented which helps develop the reader's understanding of cognitive aspects of arts as they were developed.

David Marr was a neuroscientist who pioneered studies in vision and information processing. His posthumously published work, *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information* [6] discusses a framework for extracting shape information from images. This framework, summarized in Table 1, offers a valuable perspective on how information can be encoded in images. The presented hierarchy is helpful for understanding the representational difficulties with imagery. Marshall McLuhan describes this as the difference between "hot and cool" media. [7] In this case, a "hot" media is one which engages the viewer's senses in a way that allows analytic understanding of a scene. A "cool" media is one which gives the user a passive experience. This can be illustrated further in the context of a 3D scene. A "hot" 3D scene would allow camera movement (panning and tilting) which allows the user to gain perspective. A cold scene is one which does not allow this movement, which means the user has a more passive experience with the information.

2.3 Data Representation

Edward Tufte's seminal work, *The Visual Display of Quantitative Information* serves to illustrate the role accurate portrayals plays in perception of data.[8] In

particular, Tufte's thesis is that data cannot be absorbed from data graphics, rather graphics serve best to show relations between data. This is a worthy consideration for our project. Not only should all graphics generated by the software we use present the data in such a way that allows for a proper interpretation of the data, but all graphics should also promote the viewer to compare different levels of the data. Tufte lays out and describes many guidelines for doing exactly this in two of the sections in his book; one on graphical practice and one on graphical integrity.

Tufte's section on graphical practice is about how data can be represented. A time-series plot is the most common graphic used today as it is particularly efficient and effective at presenting information when the passage of time is important.[8] For instance, businesses and affairs in economics, statistics, and meteorology will often use time-series plots to track recent and long-term trends, predict future states and behaviors of the data, and lastly to present this information in an easily understood manner to colleagues, co-workers, and the public.

Another powerful method is discussed is the use of data graphics. Due to the modern advancement of technology, computerized graphics can hold tremendous amounts of data. This method powerfully encourages pattern abstraction by allowing the viewers to have an broad look at the information. The viewers can compare and contrast similarities and difference among the data points and take away meaningful conclusions that would be impossible for other methods to portray. There is no other method for the display of statistical information that is as powerful as a data graphic.[8]

Tufte's section on graphical integrity is about how to accurately represent the data to allow for a proper interpretation of the data. For instance, a graphic will not distort the information if the visual representation of the data is consistent with the numerical representation.[8] This is rather clear, but graphics are constantly between manipulated to sway how the data is interpreted, causing false conclusions which usually are to benefit some group or cause. Tufte discusses guidelines on graphical integrity, especially when considering perspective with 3D graphics. These precautions and guidelines are important in order to maintain a proper interpretation of the data among the viewers.[9]

Leland Wilkson's *The Grammar of Graphics* draws significant inspiration from Tufte's work, but coalesces the details into a more technically relevant manual. [10] This monograph introduces the idea of a syntactic structure underlying the design of graphics for visualizing data. The syntax is what ultimately forms the grammar of the implementation. This idea is valuable towards our work, since it allows us to create extemporaneous environments for data presentation. Wilkison's work is also useful to our study since the approach is textual. Throughout the book, he develops pseudo-codes that can translate into other programming languages. In fact, it his proposed workflow has been implemented in multiple programming languages including, R, Python, Julia, and C. His work is qualified by experience writing an object-oriented graphic system for a commercial company. It should be noted that The Grammar of Graphics centers mostly of the display of datasets in two dimensions, however these ideas are extensible to 3 dimensions.

2.4 Data Interpretation

Until relatively recently in human history the communication of quantitative data resided in tables. In 1786, a Scottish economist named William Playfair, published his "Commercial and Political Atlas" which is widely thought to be the first use of diagrams to convey data.[11] In particular, his goal was to lengthen the impression the data had on the reader. J.H. Lambert used graphics to show the relation between two data sets. We can draw two immediate benefits from graphical representations. First, they are significantly more impressionable on the reader. Second, graphics show relations between data not readily evident with other representations.

We can uncover two important considerations for education from these two benefits of graphics. We know that the interpretation must be invariant among each reader, and that it should encourage the students to inspect the relations between the different layers of the data. This requires simplicity and a complete understanding of all quantities and measures. [12]

2.5 Scientific Visualization

Many works have been published in the past two decades about scientific and computer visualization. Around the early 1990's 3D graphics became practical to use on a personal computer. Thus, many scientists now had more tools for displaying data aside from traditional 2D methods. The blossoming of 3D graphics helped grow many industries. Helen Wright makes very clear there is a distinct difference between computer graphics and computer visualization, saying that visualization is an interactive process to understand what produced the data, rather than the means of presentation.[13]

3 Programming Languages in Scientific Computing

Scientific computing environments vary with language structure and development ecosystems. In this section we will explore different scientific computing packages and describe their relevance to FEM education. The compiled list was selected for both the accessibility to a novice programmer and community provided libraries. There are many programming languages available, and even more which are domain specific to FEA. As such, we recognize that the computing language is a vehicle for generating results and graphics. This review will help establish some considerations in choosing a computing environment.

3.1 MATLAB

MATLAB is a proprietary software package and language developed by The MathWorks, Inc. The standard environment allows for numerical and graphical programming.[14] The programming language was innovative when introduced in 1984 due to an interface with high performance linear algebra libraries written in FORTRAN such as LINPACK, and now LAPACK.[15] MATLAB features an interpreter for mathematical operations which makes this possible. The notion of an interpreted language for mathematics was pioneering at the time, and has

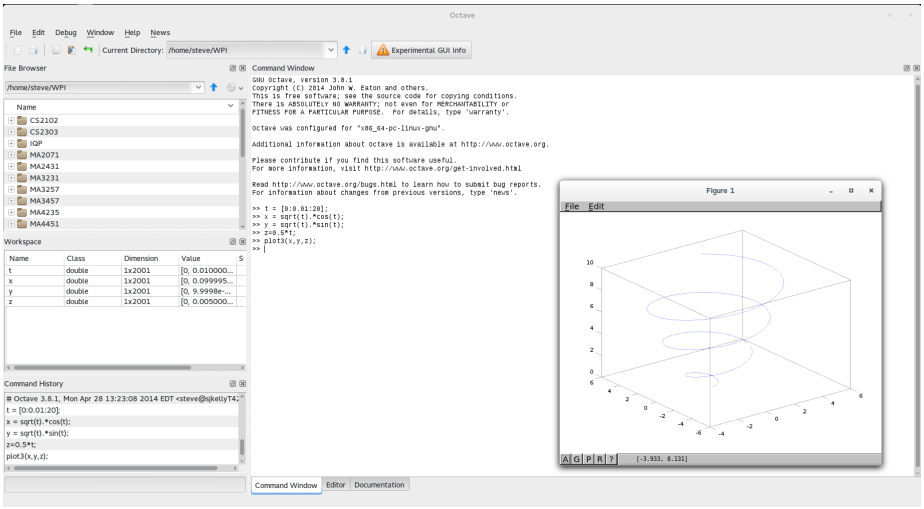

Figure 1: The MATLAB graphical user interface.

lead to much of the market dominance MATLAB experiences today. [1] The language syntax and structure is designed for vector and tensors as the main primitives, which aids in representational brevity for FEA software.

As of this writing in 2014, WPI Computing and Communications Center provides support and network installations for MATLAB on campus. The Scientific/Engineering Software Applications (SESA) group at WPI also teaches courses multiple times per term on MATLAB from the introductory to advanced levels.

The MathWorks has developed extensive libraries for scientific and engineering applications. In addition the community contributes code through the MATLAB file exchange.[2] The MATLAB development environment provides a graphical user interface (GUI) and extensive support for 2D and 3D graphics. Figure 1 shows the MATLAB user interface.

3.2 GNU Octave

GNU Octave is a interpreted language for numerical computing. It is released as free and open source software under the GNU GPL.[16] The language is very similar in structure to MATLAB and the language developers treat incompatibility with MATLAB as a bug. [17] The most recent release in December of 2013 introduced compatibility with a graphical user interface. This significantly improves the user experience and makes the environment more conducive to rapid prototyping. [18] A screenshot is shown in Figure 2.

The Octave language is often cited as a free alternative to MATLAB. This is possible because it leverages many of the same underlying libraries. While Octave is compelling due to the cost, there are some drawbacks, particularly in usability and performance. The developers are actively working to address

[1] `http://www.sagemath.org/talks/2007-05-18-maple_mathematica_matlab_magma/survey.pdf`

[2] `http://www.mathworks.com/matlabcentral/fileexchange/`

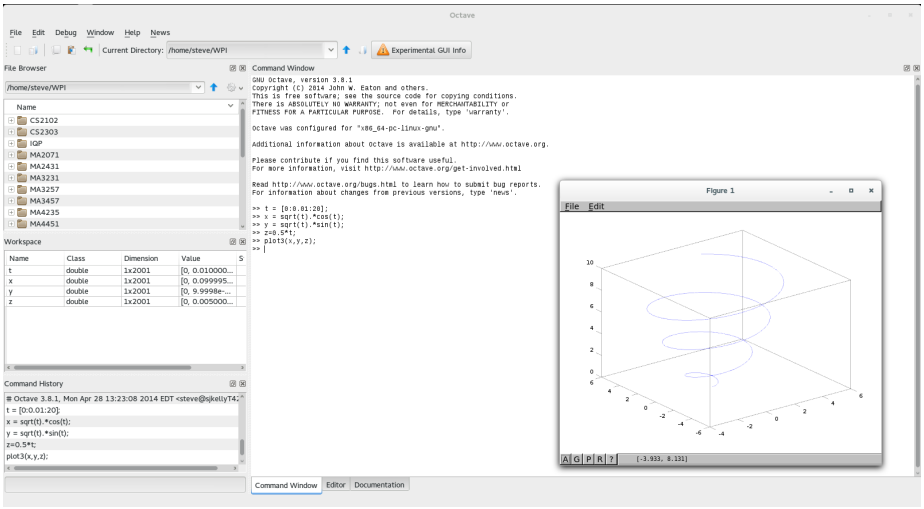

Figure 2: The Octave graphical user interface.

these issues. Usability is being improved with the aforementioned GUI and performance is being improved with a Just-In-Time compilation technique.[3]

3.3 Scilab

Scilab is an open source computing environment for scientific and engineering computations. It is released by an academic consortium under the CeCILL license which is compatible with the GNU GPL.[4] The associated language is designed for numerical computing and shares many similarities to MATLAB, and Octave. This leads to some possibilities for compatibility. The developers are actively trying to build open and license-free standards for scientific computing. This has led to significant popularity within European universities.[19] Figure 3 shows the user interface. Scilab is in direct competition with Octave, as it seeks to displace MATLAB with a free alternative. This is a common theme that will be developed later in this paper.

3.4 R

R is an open source scripting language designed for statistical computing. While the language and libraries are not generally oriented towards FEM, the language implementation and graphics packages can serve as reference implementations for open source scientific computing. [5]

The computing language R uses a package known as "ggplot2" to create graphics. The library is based off of the aforementioned *Grammar of Graphics*.[20] As such, we could expect the plotting package to produce effective and accurate visuals given proper arguments. However many of the libraries, and equally the community, built around R are focused towards statistical computing. The language features first-class data containers which are analogous

[3] `http://www.gnu.org/software/octave/doc/interpreter/JIT-Compiler.html`
[4] `http://opensource.org/licenses/CECILL-2.1`
[5] `http://www.r-project.org/`

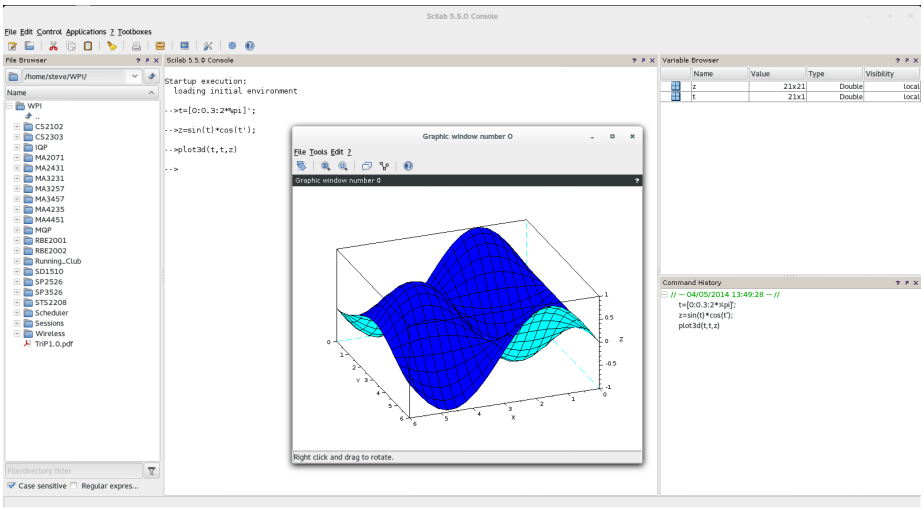

Figure 3: The Scilab graphical user interface.

to objects. It also features vectors and vectors operations for improving code performance. RStudio is an IDE which provides easy interface to ggplot2 and package repositories. [6]. R is gaining popularity in recent years as a numerical computing environment.

3.5 Python

Python should be noted as distinct in our list as it is designed for systems programming. This means the language was designed to interface with operating systems and perform file manipulations. As a systems language, Python can harness tools for documentation and library creation. The language was first release in 1991 by Guido van Rossum.[21]

The Scientific Python [7] project provides high performance utilities and services for numerical and scientific computing projects. In addition, all these projects are open source and have companies willing to provide support and consultation. Companies well known and respected are Enthought and Continuum Analytics. Notable software under this project is NumPy, MatplotLib, and IPython. NumPy provides wrappers for high performance vector operations, through LAPACK and OpenBLAS wrappers. This is similar to the other languages mentioned here.

Matplotlib is an extensive visualization suite capable of 2D and 3D graphics.[22] The facilities Matplotlib provides are highly comparable to MATLAB. IPython is a web-based tool for running Python notebooks. [23] It allows for multiple media representations within a single document. The notebook allows a user to render LaTeX, Markdown, and images. In addition the cell-based utilities allows code to be run and debugged effectively. Figure 4 shows a demonstration of the various media utilities available. The development team has decoupled the notebook platform from the language run in the cells by the kernel. They

[6] `https://www.rstudio.com/`
[7] `http://www.scipy.org/`

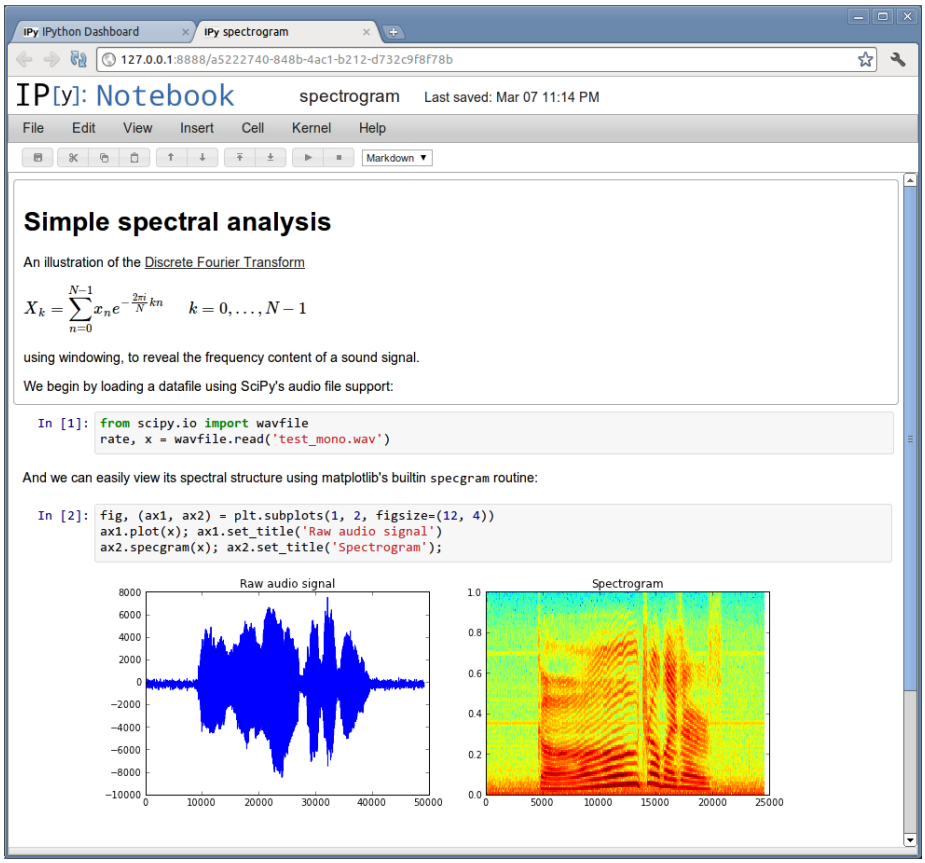

Figure 4: A demo of the IPython notebook.

also have an extensive roadmap for development which will be discussed in later sections.

3.6 Julia

Julia is a programming language first released in early 2012 by a group of developers from MIT. The language targets technical computing by providing a dynamic type system with near-native code performance. This is accomplished by using three concepts: a Just-In-Time (JIT) compiler to target the LLVM framework, a multiple dispatch system, and code specialization.[24] The syntactical style is similar to MATLAB and Python. The language implementation and many libraries are available under the permissive MIT license.[8]

Benchmarks have shown Julia can consistently perform within a factor of two of native C and FORTRAN code.[9] This is enticing for a Finite Element Analysis and numerical computation, as the code abstraction can grow organically without performance penalty. In fact, the authors of Julia call this balance a solution to the "two language problem". The problem is encountered when

[8] `http://opensource.org/licenses/MIT`
[9] `http://julialang.org/benchmarks`

Table 2: Performance benchmarks.

	Fortran gcc 4.8.1	Julia 0.2	Python 2.7.3	R 3.0.2	Matlab R2012a	Octave 3.6.4	Mathematica 8.0	JavaScript V8 3.7.12.22	Go go1
fib	0.26	0.91	30.37	411.36	1992.00	3211.81	64.46	2.18	1.03
parse_int	5.03	1.60	13.95	59.40	1463.16	7109.85	29.54	2.43	4.79
quicksort	1.11	1.14	31.98	524.29	101.84	1132.04	35.74	3.51	1.25
mandel	0.86	0.85	14.19	106.97	64.58	316.95	6.07	3.49	2.36
pi_sum	0.80	1.00	16.33	15.42	1.29	237.41	1.32	0.84	1.41
rand_mat_stat	0.64	1.66	13.52	10.84	6.61	14.98	4.52	3.28	8.12
rand_mat_mul	0.96	1.01	3.41	3.98	1.10	3.41	1.16	14.60	8.51

abstraction in a high-level language will disproportionately affect performance unless implemented in a low-level language.

The visualization tools available within the Julia ecosystem are weak as of this writing in Spring of 2014. There are promising projects that show this might not be the case in the future. In particular, four Google Summer of Code projects for Julia have been funded by Google to allow student developers to work on visualization tools.[10]

While the language is in active development, it has been designed to take advantage of code written in other languages. Julia has bindings using LLVM for Python, C, and Fortran. A polyglot environment is not conducive to uniform structure and comprehensibility, it allows stable code to be used from other languages. Notably, MatplotLib is accessible from Julia.

Recent developments to the IPython notebook have allowed other languages to interface with the rendering framework. The Julia community has created a version called IJulia that is compatible with IPython. Professor Steven J. Johnson at MIT has compiled a compelling demonstration of the facilities available with IJulia[11].

3.7 Review Conclusion

Our analysis of the available programming languages lead us to the following conclusions. First and foremost, there are many excellent languages for numerical computing. Language choice has many aspects. In order to find a good direction in which to proceed, we need to focus on our target audience.

Many of the computations that will be done in the FEA courses at WPI will be low on resources. Performance is not an immediate concern, but it provides an avenue for technique scalability and experimentation. The Julia community created performance benchmarks using various numeric programming languages. [12] The results can be seen in Table 2. A visual representation is given in Figure 5. The graph is particularly insightful, as it shows the disparity in performance between many languages relative to native C performance. Julia provides fairly consistent performance relative to all other languages, while FORTRAN is remarkably superior for some computations. This illustrates why most linear algebra libraries are written in FORTRAN and C.

To conclude, we would like to address the development environments around each language. The screenshots in Figures, 1, 2, and 3 appear remarkably

[10] http://www.google-melange.com/gsoc/project/details/google/gsoc2014/simon_danisch/5757334940811264

[11] http://nbviewer.ipython.org/url/jdj.mit.edu/~stevenj/IJulia20Preview.ipynb

[12] http://nbviewer.ipython.org/url/julialang.org/benchmarks.ipynb

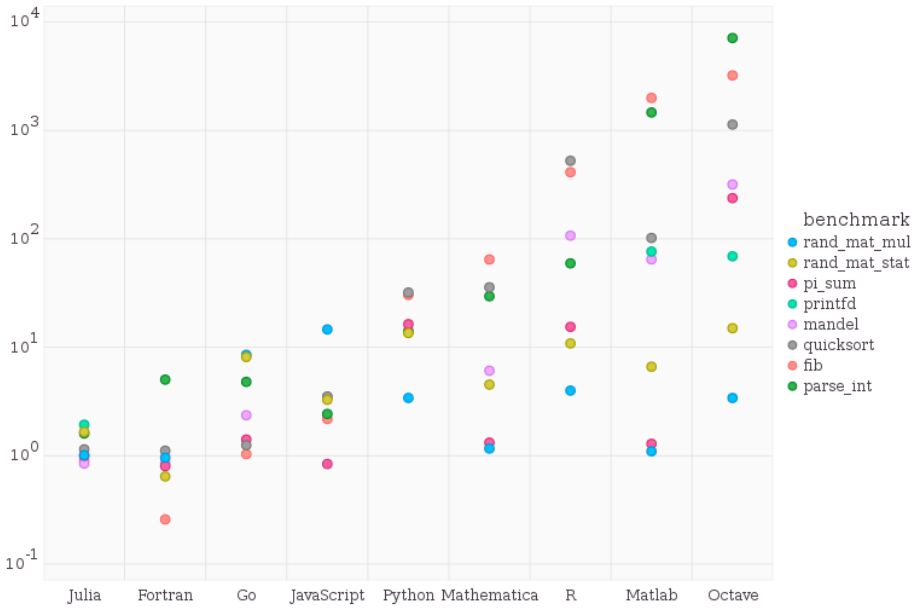

Figure 5: Visual performance benchmarks.

similar. Each development team followed a similar paradigm for interfacing through a Read Evaluate Print Loop (REPL). In MATLAB, Scilab, and Octave they provide the REPL in addition to variable display, command history, and a file browser. This general paradigm is excellent for experimentation. MATLAB, due to its availability and training support at WPI campus, was thus chosen as the primary programming language for development in this project. Julia, due to its comparable performance and recent emergence in the opensource community, was also chosen for experimental development in this project.

4 Results and Discussion

Given our understanding of the proper practice of graphics, we are left with the question of the proper representation of FEA information. The Matrix Analysis of Finite Element Structures course at WPI examines trusses and beams - two common problems in the Finite Element field. Through the study of FEA, students formulate element stiffness matrices for 2D and 3D trusses and beams. Applied forces cause reactions and deflections of the elements, and this is information that could be greatly benefited by a data graphic.

4.1 MATLAB Plotting Modifications

At first, the code for MATLAB visualization of the trusses and beams could only handle 2D structures since 3D visualizations in MATLAB are not easily done. An example of a 2D truss with the original code can be seen in Figure 6. The visuals used for the boundary conditions of the truss elements follow standard

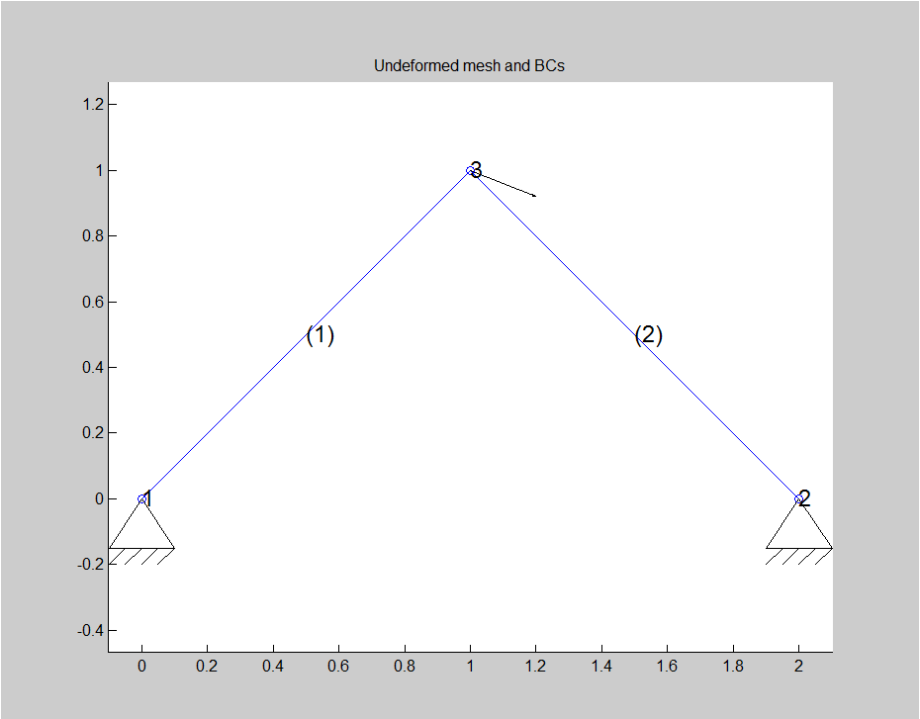

Figure 6: Initial Truss Visualization

graphical practices in engineering and an applied force vector is visualized at node 3.

The resulting deflections of the truss members are shown in Figure 7 with the original truss shown in blue and the deformed truss shown in red. The use of color in a graphic can be a very effective means of visually distinguishing commonalities between the represented data. In this example, the use of color is very intuitive and promotes understanding of the visual.

The 2D visualization code was expanded to plot 3D trusses as well as exhibit other added features that promote comprehension and provide more information on the data. The boundary conditions of the undeformed 3D truss can be seen in Figure 8 and the undeformed and deformed 3D trusses can be seen in Figure 9. The force vector in Figure 8 has been made red to stand out from the truss and the color of the undeformed truss members has been changed to black. Now, color is used in the deformed truss in Figure 9 to show whether the members are in tension (red) or compression (blue). An accompanying legend insures proper interpretation of the color usage.

A large part of the reasoning behind developing better visualization capabilities in this project was to encourage the students to draw meaningful connections between the topics discussed in class, the Finite Element code they've been writing, and the resulting visual graphic. By comparing the 3D truss example figures, one would expect that the red force vector in Figure 8 would push node 4 by some amount in the force vector's general direction. It could also be reasoned that the truss members 1 and 2 could become compressed

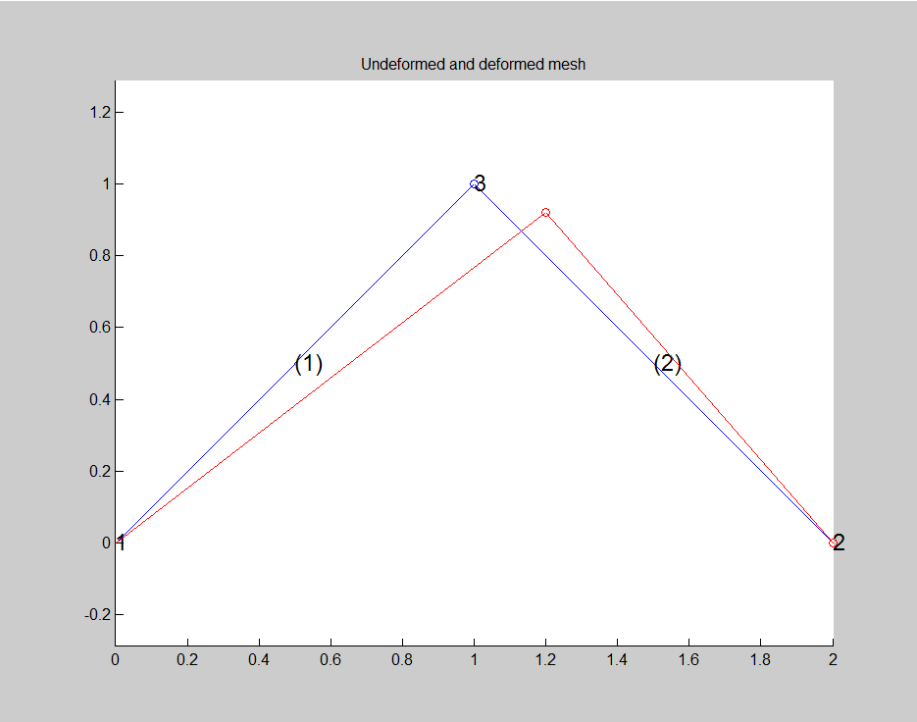

Figure 7: Initial Deformed Truss Visualization

while truss member 3 becomes extended. All of this logic can be confirmed by a visualization generated by correct code. Inconsistencies between the produced graphic and an expected result would prove that either the student's code is not correct or that the underlying concepts taught by the course are not fully understood. Visualizations can be used by the students to easily verify whether their reaction calculations make sense.

4.2 Julia Implementation

During the review of the MATLAB plotting function, some idiosyncracies were noticed with the MATLAB language. In particular it became difficult to design user interfaces. We were also driven by the cognitive and user experience aspects of the MATLAB experience to find an alternative. Some outside work lead to the use of IJulia. This environment was very excellent to present both equations, code, and visuals in a single document.

We started to port some basic Finite Element computation code over to Julia. For the most part, syntactic differences were the main challenge. The Julia manual has some through observations on the differences from other languages.[13] Some additional obstacles were slightly more grammatical. Julia's JIT compiler requires variables be preallocated. In MATLAB this is a performance optimization, however in Julia it is a requirement. The process of porting the code strengthened our understanding of the operations involved.

[13] `http://julia.readthedocs.org/en/latest/manual/noteworthy-differences/`

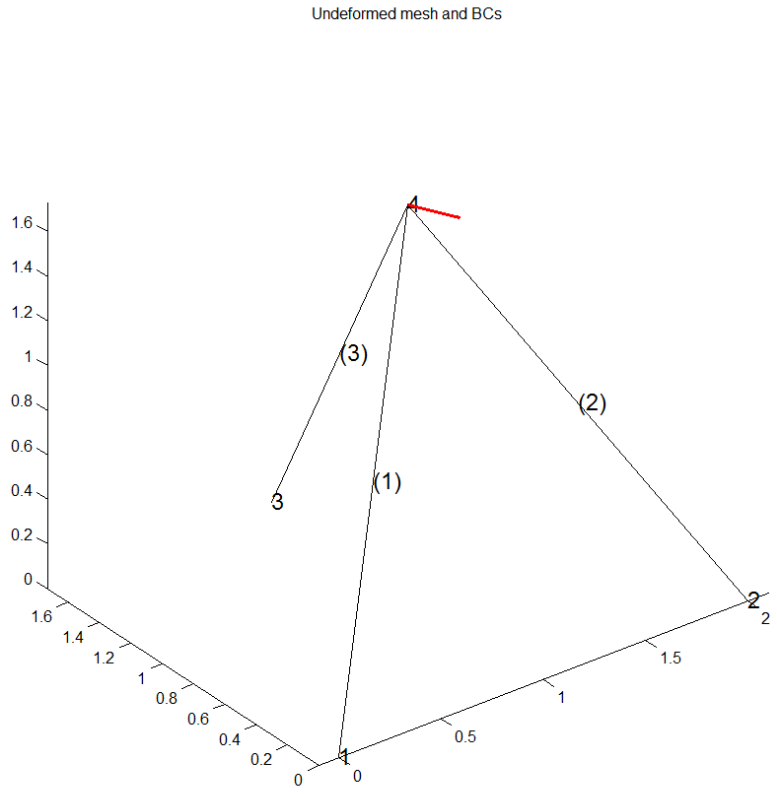

Figure 8: 3D Simple Truss Visualization

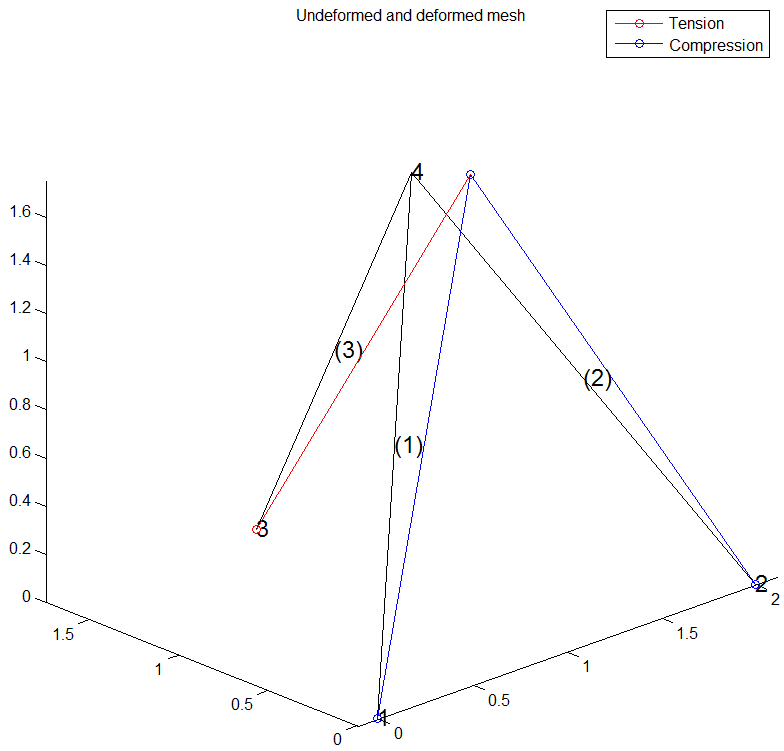

Figure 9: 3D Simple Truss Example

```
Nodal Displacements:
 node d1 d2
    1       0       0    0
    2       0       0    0
    3       0       0    0
    4 0.647024 -1.03658.690214e-02

Nodal Reactions
 node R1 R2
    1 3.86751 3.349366.698730e+00
    2 -53.8675 46.65069.330127e+01
    3       0      50 -100
    4       0       0    0

Element Axial force/stress/strain
  elem     force      stress      strain
    1   -8.42905   122.188   0.610938
    2   -117.402   122.188   0.610938
    3   111.803   122.188   0.610938
```

Figure 10: 3D Simple Truss FEA Data

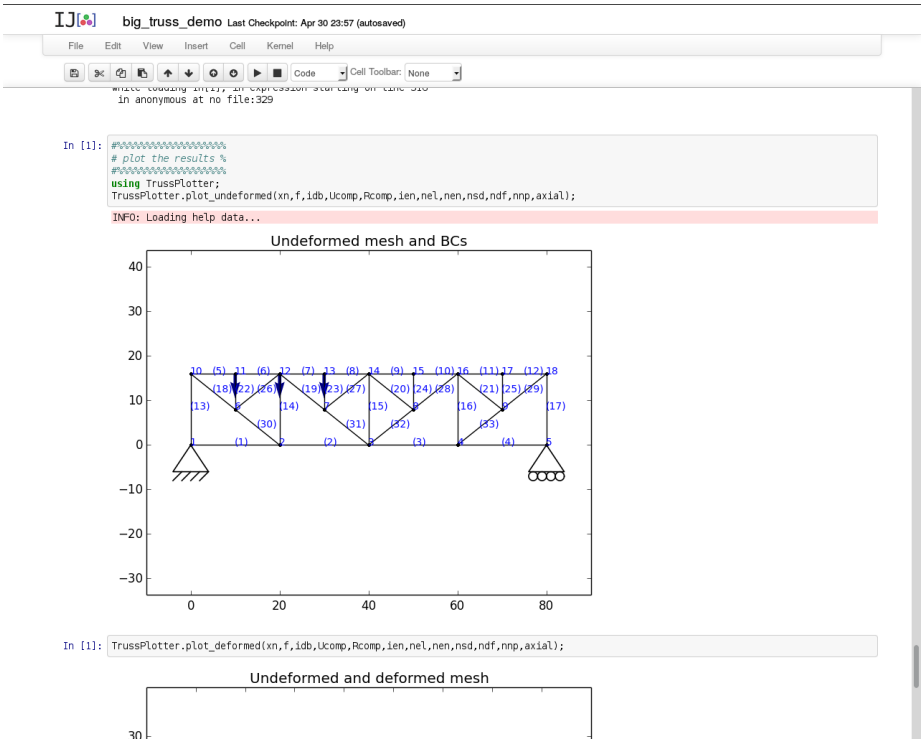

Figure 11: Output of the Truss FEM code in IJulia.

After the numerical computation porting was complete, visual utilities had to be made. Fortunately Julia includes many plotting utilities. One of the most advanced is PyPlot, which is a wrapper for MatplotLib, a plotting utility written in Python. The function calls to MatplotLib are very similar to MATLAB. [25] Again, this made the porting process very easy.

4.3 Future Work

4.3.1 IPython Multi-User Support

The Sloan Foundation granted funding of 1.15 million USD for about two years of development to IPython.[14] With this funding the development team decided to target 4 releases every 6 months. This started with a 1.0 release in August 2013, followed by a 2.0 release in April 2014. The grant objectives are as follows: refinement of the IPython notebook through interactive utilities and various export formats (LaTeX, PDF, HTML, Presentations), support for languages other than Python, and support for multiuser notebooks.[15] With the release of version 2.0, all of these goals except for multiuser support have been realized.

In addition, these changes are to support a pilot of an applied statistics course at Stanford. The 3.0 release of IPython will preview initial support for multiuser notebooks. The developers state that the environment will map one-

[14] `http://ipython.org/sloan-grant.html`
[15] `https://github.com/ipython/ipython/wiki/Roadmap:-IPython`

to-one to a Linux host. This means integration with the WPI Unix accounts should be straight forward. A student might login with Shibboleth and an IPython notebook would start in their account (/home/user/ipython).[16] As with most open-source and pro bono projects, the development is driven by the needs of the users. As such, IPython is being developed to suit university coursework in the 2014-15 academic year.

4.3.2 IJulia for Coursework

With the PyCall package, Julia and IJulia can take advantage of all improvements to the IPython environment. The roadmap for IPython is compelling, and will enrich the user experience with IJulia. Courses in numerical and high performance computing have been taught using Julia at MIT, Cornell, Stanford, Pennsylvania State, and Western University Canada.[17] While this pales in comparison the amount of courses taught with MATLAB, open computing ecosystems are clearly gaining momentum. At the end of this summer we believe the facilities available in Julia and IPython will meet and exceed those available in MATLAB.

Early adoption of IJulia in coursework will both be challenging and enriching. Most of the challenge lies in establishing infrastructure and course material. The similarities between MATLAB and Julia, along with the roadmap of IPython make these less concerning. On the other hand, the student only stands to gain from the switch to an open source and web-based computation environment. A free language, both in license and cost, lets the student apply their skills outside of corporate and academic organizations without financial concern. In addition, web notebooks mean a student does not have to invest in an expensive computer to complete homework assignments.

5 Conclusion

The study of visualizing scientific data has lead to the development of utilities all students can leverage for their courses. Our exploration of computing technologies lead us to possibilities for future improvements in the coursework. In this project, common practices for visualizing scientific data were studied. In addition, the science of cognition and data display was reviewed. The results of this investigation was applied to augment a Civil Engineering introductory course on Finite Element Method at WPI. Software enhancements allowed three dimensional visualization for simulation of engineering structures. The research on cognition and data graphics was used to improve understanding of these visual aids. The plotting function, developed in MATLAB and Julia environments during the course of this project, can help all students visualize the results of their numerical codes.

[16] `https://github.com/ipython/ipython/wiki/IPEP-33A-Multiuser-support-in-the-notebook`
[17] `http://julialang.org/teaching/`

References

[1] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis*. Courier Dover Publications, 2012.

[2] J. J. Rencis and H. T. Grandin, *A New Approach to Mechanics of Materials: An Introductory Course With Integration of Theory, Analysis, Verification and Design*, p. 3–10. American Society of Mechanical Engineers, 2005.

[3] R. Courant, "Variational methods for the solution of problems of equilibrium and vibrations,"

[4] K.-J. Bathe, *Finite element procedures*. Klaus-Jurgen Bathe, 2006.

[5] R. Solso, *Cognition and the visual arts*. Cambridge, Mass: MIT Press, 1994.

[6] D. Marr and A. Vision, "A computational investigation into the human representation and processing of visual information," *WH San Francisco: Freeman and Company*, 1982.

[7] M. McLuhan, *Understanding Media: The Extensions of Man*. New American Library, 1964.

[8] E. Tufte, *The visual display of quantitative information*. Cheshire, Conn: Graphics Press, 2001.

[9] M. Zachry and C. Thralls, "An interview with edward r. tufte," *Technical Communication Quarterly*, vol. 13, no. 4, 2004.

[10] L. Wilkinson, *The grammar of graphics*. Springer, 2005.

[11] I. Spence and H. Wainer, "Who was playfair," *Chance*, vol. 10, no. 1, pp. 35–37, 1997.

[12] B. Victor, "An ill-advised personal note about "media for thinking the unthinkable"."

[13] H. Wright, *Introduction to scientific visualization*. London: Springer, 2007.

[14] S. Attaway, *MATLAB: A practical introduction to programming and problem solving*. Elsevier, 2012.

[15] C. Moler, "Matlab incorporates lapack." `http://www.mathworks.com/company/newsletters/articles/matlab-incorporates-lapack.html`, 2000.

[16] "GNU General Public License." `http://opensource.org/licenses/GPL-3.0`.

[17] J. W. Eaton, D. Bateman, and S. Hauberg, *Gnu octave*. Free Software Foundation, 1997.

[18] J. S. Hansen, *GNU Octave: Beginner's Guide: Become a Proficient Octave User by Learning this High-level Scientific Numerical Tool from the Ground Up*. Packt Publishing Ltd, 2011.

[19] S. L. Campbell, J.-P. Chancelier, and R. Nikoukhah, *Modeling and Simulation in SCILAB*. Springer, 2006.

[20] H. Wickham, *ggplot2: elegant graphics for data analysis*. Springer Publishing Company, Incorporated, 2009.

[21] "General Python FAQ — Python v2.7.6 documentation." `https://docs.python.org/2/faq/general.html`.

[22] S. Tosi, *Matplotlib for Python developers: build remarkable publication quality plots the easy way*. Packt Publishing Ltd, 2009.

[23] F. Pérez and B. E. Granger, "IPython: a system for interactive scientific computing," *Computing in Science and Engineering*, vol. 9, pp. 21–29, May 2007.

[24] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Julia: A fast dynamic language for technical computing," *arXiv preprint arXiv:1209.5145*, 2012.

[25] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 0090–95, 2007.