

**An Analysis of Scientific Computing Environments:
A Consumer's View**

by

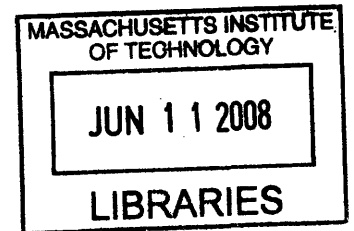
Amit Soni
B.S. Mechanical Engineering (2006)
Indian Institute of Technology Kanpur

Submitted to the School of Engineering
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computation for Design and Optimization

at the

Massachusetts Institute of Technology

JUNE 2008



© 2008 Massachusetts Institute of Technology
All rights reserved

ARCHIVES

Author

.....
School of Engineering
May 16, 2008

Certified by

Alan Edelman
Professor of Applied Mathematics
Thesis Supervisor

Accepted by

.....
Jaime Peraire
Professor of Aeronautics and Astronautics
Codirector, Computation for Design and Optimization

An Analysis of Scientific Computing Environments: A Consumer's View

by

Amit Soni

Submitted to the School of Engineering on May 16, 2008
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computation for Design and Optimization

Abstract

In the last three decades, with rapid advancements in the hardware and software industry, a large number of commercial and free open source languages and software have evolved. Many of these are Very High Level Languages (VHLLs) which can be easily used for scientific computing purposes such as algorithm testing and engineering computations. However, this vast pool of resources has not been utilized to its full potential. In this analysis, we will be looking at various simple and complex problems and how they can be approached in various languages. All the results will be uploaded on a website in the form of a wiki intended to be accessible to everyone. By analyzing standard problems encountered frequently in scientific computing, this wiki provides the users a performance based report which they can use to choose the best option for their particular applications. Simultaneously, a lexicon of standard codes will help them in learning those options which they want to use so that fear is not a barrier. The analysis also addresses some incompatibility issues within languages and their impact.

This work is a preliminary investigation as part of Professor Alan Edelman's participation in the Numerical Mathematics Consortium. We expect the scientific computing community to benefit from this research as a whole, as this analysis will give them better alternatives for their computational needs.

Thesis Supervisor: Prof. Alan Edelman
Title: Professor of Applied Mathematics

Acknowledgment

I thank my advisor Prof. Alan Edelman for coming up with the novel idea of this topic. As a result of his excellent guidance and support, my Masters Program at MIT had turned out to be a great learning experience.

I also thank Singapore MIT Alliance for their support. I am grateful to Dave Hare, Sam Shearman, Shashi Mittal and OPT++ developers for their invaluable help which made this work possible. I would also like to thank the members of the mailing lists of numerous programming languages for their quick responses to my queries.

I thank my friends at MIT, Abhinav, Nikhil, Priyadarshi, Shashi and Vinay for all the moments of joy we spent together. I must acknowledge Laura for her immense helping attitude.

Finally, I would like to dedicate this thesis to my family (my parents and my brother) for their love, support and encouragement.

Contents

Chapter 1: Introduction.....	13
1.1 Motivation.....	13
1.2 Prior Work	13
1.3 Approach.....	14
1.3.1 Raw performance	14
1.3.2 Elegance of code.....	14
1.3.3 Language Compatibility.....	14
1.3.4 Database of Codes.....	14
1.4 Specifications.....	15
1.4.1 Workstation A.....	15
1.4.2 Languages/Software.....	15
1.5 An Overview of the Languages/Software used in this Analysis.....	16
Chapter 2: Linear Algebra	17
2.1 Matrices and Arrays.....	17
2.2 Eigenvalues	20
2.3 Wigner’s Semicircle.....	22
2.4 GigaFLOPS.....	32
Chapter 3: Optimization.....	33
3.1 Linear Programming	33
3.2 Non Linear Programming	34
3.2.1 Convex Programming	35
3.2.2 Non Convex Programming	36
Chapter 4: Miscellaneous.....	41
4.1 Ordinary Differential Equations	41
4.2 Memory Management.....	42
4.3 Activity Index	44
Chapter 5: Incompatibility Issues	45
5.1 Undefined Cases	45
5.2 Sorting of Eigenvalues.....	45
5.3 Cholesky Decomposition.....	46
5.4 Matlab vs Octave	46
5.5 Sine Function	48
5.6 Growing Arrays	48
Chapter 6: Conclusion.....	51
6.1 About the website	51
6.2 Future Work	52
Bibliography	53

List of Figures

Chapter 2

Figure 1 LabVIEW Eigenvalue solving Block Diagram	21
Figure 2 Block Diagram for Wigner's Semicircle	23
Figure 3 Wigner's Semicircle in Labview	23
Figure 4 Wigner's Semicircle in Maple	24
Figure 5 Wigner's semicircle in Mathematica	25
Figure 6 Wigner's semicircle in Matlab	26
Figure 7 Wigner's semicircle in Octave	27
Figure 8 Wigner's semicircle in Python	28
Figure 9 Wigner's semicircle in R	29
Figure 10 Wigner's semicircle in Scilab	30

Chapter 3

Figure 11 Accuracy of Optimization Solvers	38
Figure 12 Time taken by Optimization Solvers	39

Chapter 6

Figure 13 A screenshot of the website	51
---	----

List of Tables

Chapter 2

Table 1 Programming Languages/Software used in this analysis	16
Table 2 Time taken by Hilbert Matrix	17
Table 3 Zero Matrix in VHDLs	18
Table 4 Row vector	19
Table 5 2D Array/Matrix	19
Table 6 Identity Matrix	20
Table 7 Empty Arrays	20
Table 8 Eigenvalue Function	21
Table 9 Eigenvalue solver Performance	22
Table 10 Performance of Languages through Matrix Multiplication	32

Chapter 3

Table 11 Performance of Linear Programming Solvers	34
Table 12 Accuracy of Optimization Solvers	37

Chapter 4

Table 13 Ordinary Differential Equation Solvers	42
Table 14 Memory utilization across Languages	43
Table 15 Free Memory	44
Table 16 Activity Index	44

Chapter 5

Table 17 Ambiguity in Zero raised to power Zero	45
Table 18 Sorting of Eigenvalues	46
Table 19 Cholesky Decomposition	46
Table 20 Sine value for large input argument	48

Chapter 1: Introduction

1.1 Motivation

The demographics of technical computing are undergoing a revolutionary shift. There is an explosion in the number of users fueled in no small way by the ease of use of Very High Level Languages (VHLLs), the ubiquity of computers and the modern ability to use computations effectively in so many new ways.

Not so long ago, FORTRAN held a virtual monopoly in the scientific computing world. Technical computing meant performance! That means a low level language. As the hardware started improving rapidly and high speed computers came right in front of us on our desks, the high level languages are not looking that bad anymore. Moreover, in the trade off between computer time vs. human time, the latter weighs heavily.

In the past two decades, a large number of high-level languages started pouring in, both commercial and open source. The community using high-level languages for computational purposes is largely dominated by a small set of available options. An important reason for this indifference towards a new language or software is that learning a new language from scratch by reading bulky manuals appears to be a huge demotivating factor. So people generally have a tendency to stick to the languages they already know. Therefore there is not much room for new emerging languages as the already existing ones are already dominating the high-level scientific computing world.

We are also addressing here some incompatibility issues we encounter in the languages. Each language has its own choice for addressing issues which are not defined completely anywhere in the literature, for example sorting of complex numbers or the matrix returned after a Cholesky Decomposition. Sometimes we even see differences in approach of these cases within the same software. These kinds of discrepancies may lead to confusion among users and an attempt should be made to standardize everything without restricting innovation and competition.

So, there is a need to analyze these issues and make a consumer report which can benefit the scientific computing community. There is a vast pool of resources which has not been used properly until now. We hope to make it available to the users in a much better way, in the form of a website open to all. Users can give their inputs on the languages they know and get information on others which they want to learn.

1.2 Prior Work

There has been some prior work addressing these issues. Many of the previous comparisons actually provide a very good performance report. However, very rarely someone has talked about the numerical standards and compatibility issues between different languages.

Consider the Wikipedia articles on “Comparison of Programming Languages” [3] and “Comparison of programming languages (basic instructions)” [4]. The first article mostly talks

about the structural part of various languages. The second article is indeed helpful for someone who wants to learn a new language, but it is still in its infancy and requires a lot to be added.

We can find another significant analysis “Comparison of Mathematical Programs for Data Analysis” [1] on Stefan Steinhaus’s webpage. This analysis provides a lot of important performance oriented information on more than half a dozen languages. It also rates languages on various criteria and spans a large number of examples for speed testing.

“Rosetta Stone” [2] is another notable work, originally written by Michael Wester and modified by Timothy Daly and Alexander Hulpke. “Rosetta Stone” provides a collection of synonyms for various mathematical operations in about 17 Languages.

In an optimum analysis, we need the simplicity of presentation and the depth of information. So, we need to combine the good features of whatever work has already been done and expand it further.

1.3 Approach

The most important point of this analysis is to provide the information in an easy and simple way. It should not turn out to be another bulky tutorial. So, maximum possible information has been included while maintaining the simplicity of the report. The major issues addressed here are:

- 1.3.1 **Raw performance:** On a fixed machine, how different languages perform in terms of time taken to get to the solution. This is a major concern of the users for computationally expensive problems.
- 1.3.2 **Elegance of code:** While beauty lies in the eyes of beholder, some languages are certainly more painful to learn and use. Users can sort the available alternatives on the basis of how much they are willing to sacrifice (both time and effort) for the sake of moving to a better alternative.
- 1.3.3 **Language Compatibility:** Differing answers can create portability problems and confusion in the mind of a user using more than one language. We wish to address these cases and give an explanation of them wherever possible.
- 1.3.4 **Database of Codes:** Ultimately we hope to provide a lexicon and methodology that will allow the users to shift among languages easily. Most of the existing comparisons just compare the languages, but the absence of basic relevant codes makes it somewhat hard for the users to actually use that information.

Whenever possible, the computations are done on the same machine (Workstation A). Otherwise the specifications are listed with the results.

All this information will be posted on a website/wiki where people can freely edit the information to provide inputs based on their experiences with various languages.

1.4 Specifications

1.4.1 Workstation A

System

Lenovo

Intel Core 2 CPU

T7200 @ 2.00GHz

1.00GB of RAM

Hard Disk Speed: 7200 RPM

Operating System

Microsoft Windows XP

Home Edition

Version 2002

Service Pack 2

1.4.2 Languages/Software

- a) C on Visual C++ 2005 Express Edition
- b) Java on Eclipse 3.2
- c) LabVIEW 8.2
- d) Maple 11 on MIT Athena Machine
- e) Mathematica 5.2
- f) Matlab 7.2 (R2006a)
- g) Octave 2.1.72
- h) Python 2.4
- i) R 2.4.1
- j) Scilab 4.1.2

Unless stated otherwise, the above mentioned versions of Languages/Software are used on Workstation A.

1.5 An Overview of the Languages/Software used in this Analysis

Language	Appeared In	Developed by
C	1972	Denis Ritchie and Bell Labs
Java	1995	James Gosling and Sun Microsystems
LabView	1986	National Instruments
Maple	1980	Waterloo Maple Inc
Mathematica	1988	Wolfram Research
Matlab	Late 1970s Commercialized in 1984	Cleve Moler and Mathworks
Octave	1994	John W. Eaton
Python	1990	Guido van Rossum and Python Software Foundation
R	1996	R. Ihaka and R. Gentleman
Scilab	1994	INRIA and École nationale des ponts et chaussées (ENPC)

Table 1 Programming Languages/Software used in this analysis

Chapter 2: Linear Algebra

A large number of scientific computing tasks require Linear Algebra methods. Very High Level Languages have made it very easy to use most of the standard Linear Algebra functions such as Cholesky Factorization, Eigenvalue computations and LU decomposition. This chapter shows how various languages deal with the methods of Linear Algebra.

2.1 Matrices and Arrays

Matrices and Arrays are the starting point for any Linear Algebraic calculation. Very High Level Languages provide us with specific functions to construct various matrices. Though they are slower than compiled languages, the ease of use makes them very attractive to the users. Most of the compiled languages would require creating a matrix element-by-element.

Let us first try comparing element wise construction of matrices across languages. Many computational methods like the Finite Difference method require repetitive modification of each element of a matrix. This will reflect the joint performance of working with matrices, loops and simple calculations of various languages. Consider constructing a Hilbert Matrix of size 1000×1000 in each language element wise (some of the VHLLs do provide a specific function for constructing a Hilbert Matrix).

Language	Time(sec)
C	0.03
Java	0.03
Labview	0.07
Maple	13.44*
Mathematica	8.59
Matlab	0.04
Octave	108.26
Python	1.17
R	11.07
Scilab	24.28

Table 2 Time taken by Hilbert Matrix

* Maple programs were executed on an MIT Athena Machine. Using new Linear Algebra constructs may increase the performance significantly

C and Java, as expected, perform better than the VHLLs. Matlab and LabView come next and all the others take a significant amount of time more than these.

Now consider constructing a special matrix, for example a 15×10 Zero Matrix. In VHLLs, we now don't need to construct the matrix element wise. Using specific functions in VHLLs, we can construct this as shown in Table 3.

Language	Code
Maple	A := matrix(100,100,0);
Mathematica	Needs["LinearAlgebra`MatrixManipulation`"] A = ZeroMatrix[15,10];
Matlab	A = zeros(15,10);
Octave	A = zeros(15,10);
Python	From numpy import * A = zeros((15,10))
R	A = matrix(0,15,10)
Scilab	A = zeros(15,10);

Table 3 Zero Matrix in VHLs

But a similar result in C or Java requires considerable human effort.

Zero Matrix in C

```
#include <stdio.h>                                     //import package
main()
{
  int n=1000;                                           //size of matrix
  double **A = (double **)malloc(n * sizeof(double *)); //dynamic memory allocation
  int i,j;
  for(i = 0; i < n; i++) A[i] = (double *)malloc(n * sizeof(double));
  for(i=0;i<n;i++)
    for(j=0;j<n;j++)
      A[i][j]=0;                                       //assigning values to matrix
}
```

Zero Matrix in Java

```
public class zeroM{
  static int n=1000;                                   //size of matrix
  public static double A[][]=new double[n][n];       //defining matrix
  public static void main(String[] args){
    for(int i=1;i<=n;i++){
      for(int j=1;j<=n;j++){
        A[i-1][j-1]=0;                                //assigning values to matrix
      }
    }
  }
}
```

So, there is trade-off between machine time and human effort. Large programs in C, Java or similar languages will be even messier. Debugging these languages is also a difficult task as most of them provide low level access to memory and therefore errors are very complex too.

However, the time saved during simulations would be considerable too.

Some other useful and easily constructible vectors/matrices in VHLLs:

Row Vector

Languages	Code
Maple	<code>a := array([1,2,3]); a := <1 2 3>;</code>
Mathematica	<code>a = {1,2,3}</code>
Matlab/Octave	<code>a = [1,2,3]</code>
Python	<code>a = [1,2,3]</code>
R	<code>a = c(1,2,3)</code>
Scilab	<code>a = [1 2 3]</code>

Table 4 Row vector

2-D Array/Matrix

Languages	Code
Maple	<code>A := matrix([[1,2],[3,4]]); A := <<1 2>,<3 4>>;</code>
Mathematica	<code>A = {{1,2},{3,4}}</code>
Matlab/Octave	<code>A = [1 2;3 4]</code>
Python	<code>A = matrix([[1,2],[3,4]])* A = [[1,2],[3,4]]**</code>
R	<code>A = matrix(c(1,3,2,4),2,2)</code>
Scilab	<code>A = [1 2;3 4]</code>

Table 5 2D Array/Matrix

* Requires numpy

** `A = [[1,2],[3,4]]` is actually a list and does not require numpy. But it is not exactly a matrix and does not work with all matrix operations. For example, Inverse and Determinant (`linalg.inv(A)` and `linalg.det(A)`) work fine with a list but other functions like Eigenvalues and CholeskyDecomposition (`linalg.eigvalsh(A)` and `linalg.cholesky(A)`) return Attribute error.

Identity matrix

Languages	Code
Maple	<code>with(LinearAlgebra); A := IdentityMatrix(5); B := IdentityMatrix(5,3);</code>
Mathematica	<code>A = IdentityMatrix[5]</code>
Matlab/Octave	<code>A = eye(5) B = eye(5,3)</code>

Python	From numpy import * A = eye(5) B = eye(5,3)
R	A = diag(x=1,nrow=5) or A = diag(1,5) B = diag(x=1,nrow=5,ncol=3) or B = diag(1,5,3)
Scilab	A = eye(5,5) B = eye(5,3)

Table 6 Identity Matrix

Empty Arrays

Language	Code	Result
Maple	matrix(5,0,0);	[]
Mathematica	Needs["LinearAlgebra`MatrixManipulation`"] ZeroMatrix[5, 0]	{{}, {}, {}, {}, {}}
Matlab	zeros(5,0)	Empty matrix: 5-by-0
Octave	zeros(5,0)	[](5x0)
Python	z=zeros((5,0))	array([], shape=(5, 0), dtype=float64)
R	z = array(0, c(5,0))	[1,] [2,] [3,] [4,] [5,]
Scilab	zeros(5,0)	[]

Table 7 Empty Arrays

2.2 Eigenvalues

Eigenvalue computation is a computationally expensive process and is required extensively in numerous algorithms.

Mathematically, a scalar λ is an eigenvalue of an $n \times n$ matrix A if it satisfies:

$$A\mathbf{x} = \lambda\mathbf{x}$$

where \mathbf{x} is an eigenvector corresponding to the eigenvalue λ

Suppose A is an $n \times n$ square matrix. Then the functions in Table 8 will return an $n \times 1$ vector containing the eigenvalues of A .

Language	Function
Maple	with(linalg); eigenvalues(A);

Mathematica	Eigenvalues[A]
Matlab	eig(A)
Octave	eig(A)
Python	From numpy import * linalg.eigvalsh(A)
R	eigen(A)\$values
Scilab	spec(A)

Table 8 Eigenvalue Function

In LabVIEW, the block diagram would be:

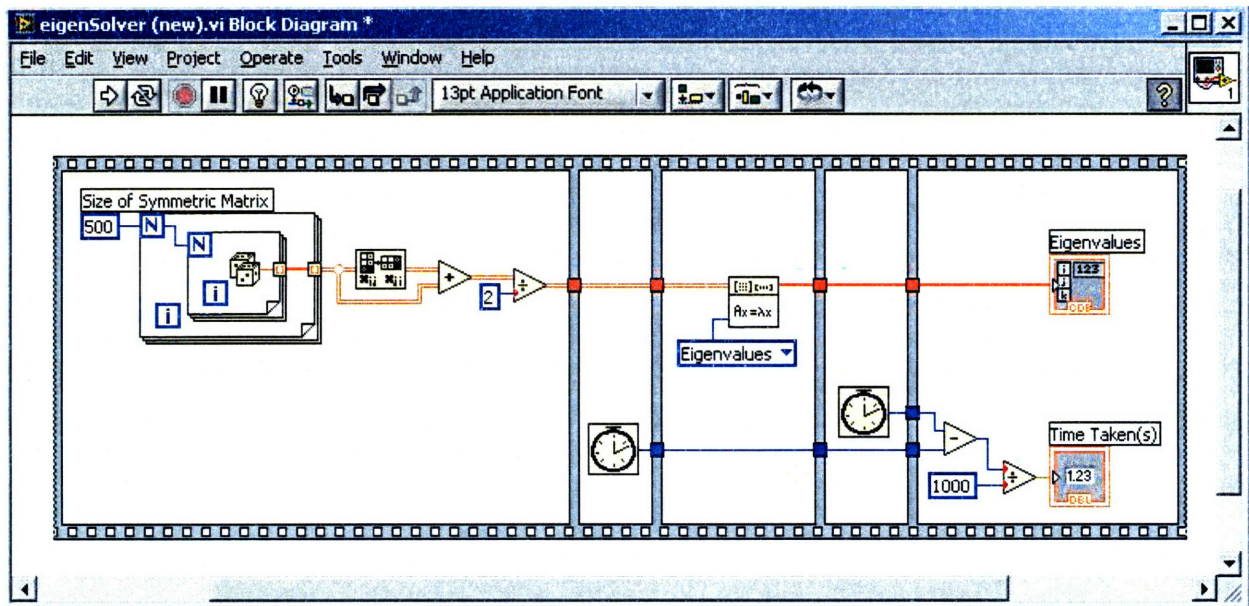


Figure 1 LabVIEW Eigenvalue solving Block Diagram

The dices in the above figure represent a random number generator. Executing it in nested loop returns an $n \times n$ matrix (say A). A symmetric matrix S is created from A:

$$S = (A + A^T)/2$$

S is then fed into Eigenvalues VI which returns the eigenvalue vector. Five blocks in the block diagram represent the sequence in which these VIs should be executed.

Performance

Language/Software	Time in seconds		
	n=500	n=1000	n=2000
Labview	0.29	1.96	14.48
Mathematica	0.42	2.0	12.1
Matlab	0.18	1.13	8.85
Octave	0.2	1.0	7.35
Python	0.24	1.9	19.3

R	0.52	2.91	23.06
Scilab	4.83	5.09	41.12

Table 9 Eigenvalue solver Performance

- All of these functions can utilize symmetry, either by defining or by themselves. So, for most of the cases, we can see that the time taken increases 8 times when the size of the matrix is doubled.
- Matlab and Octave lead others in performance in eigenvalue solving.

2.3 Wigner's Semicircle

When the histogram of the distribution of eigenvalues of a symmetric normalized random matrix is plotted, we get Wigner's semicircle. So, apart from Linear Algebraic functions, here we will be looking at histogram and plot functions as well.

From the next page onwards, we will be looking at the steps required for this exercise and the results we get in each language.

LabView

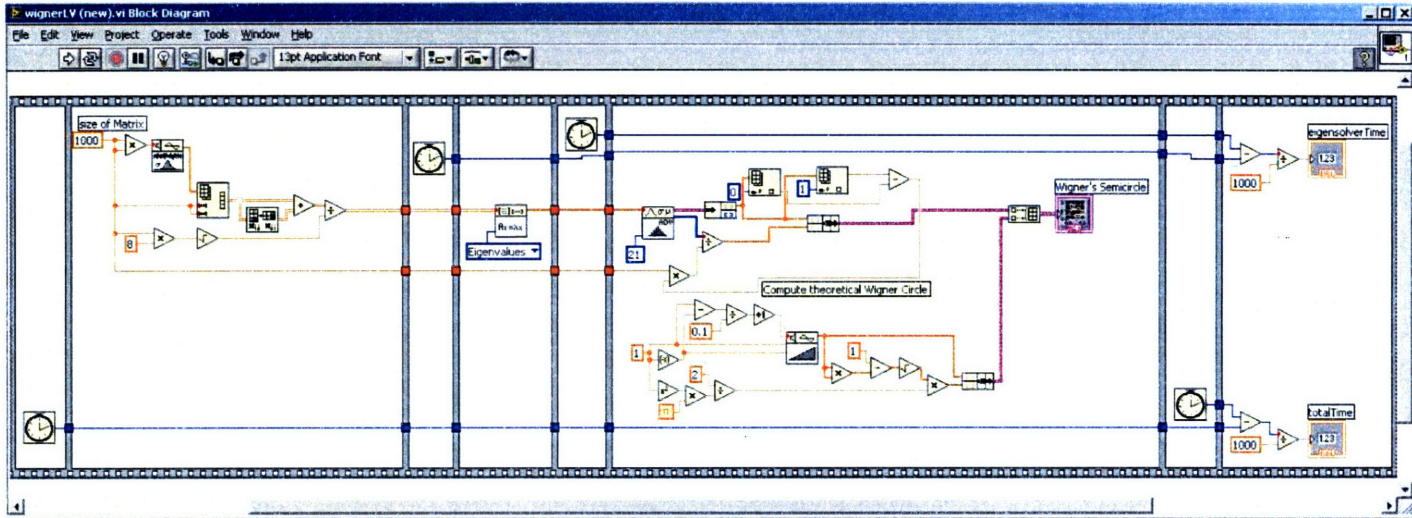


Figure 2 Block Diagram for Wigner's Semicircle

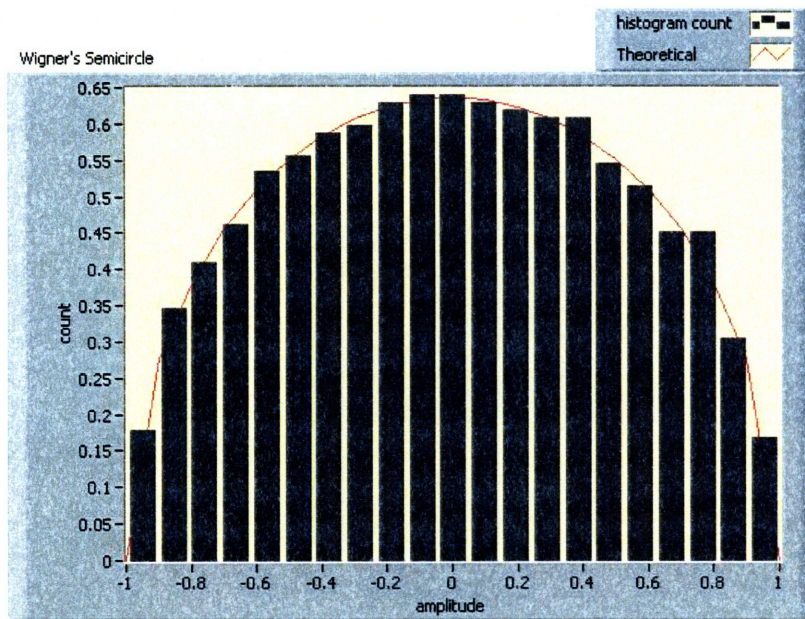


Figure 3 Wigner's Semicircle in Labview

Time Taken:

Eigenvalue Solving: 20.21 s
 Total: 20.74 s

Maple

```
> with(linalg):
> n:=1000:
> A:=matrix(n, n, [stats[random, normald](n*n)]):
> S:= (A+transpose(A))/(sqrt(8*n)):
> eig:=eigenvals(S):
> with(plots,display):
> plot1:=plot(2*n*0.1*(sqrt(1-x*x))/(Pi), x=-1..1):
> plot2:=stats['statplots','histogram'](map(Re@evalf,[eig]),area=count,numbars=20):
> display(plot1,plot2);
```

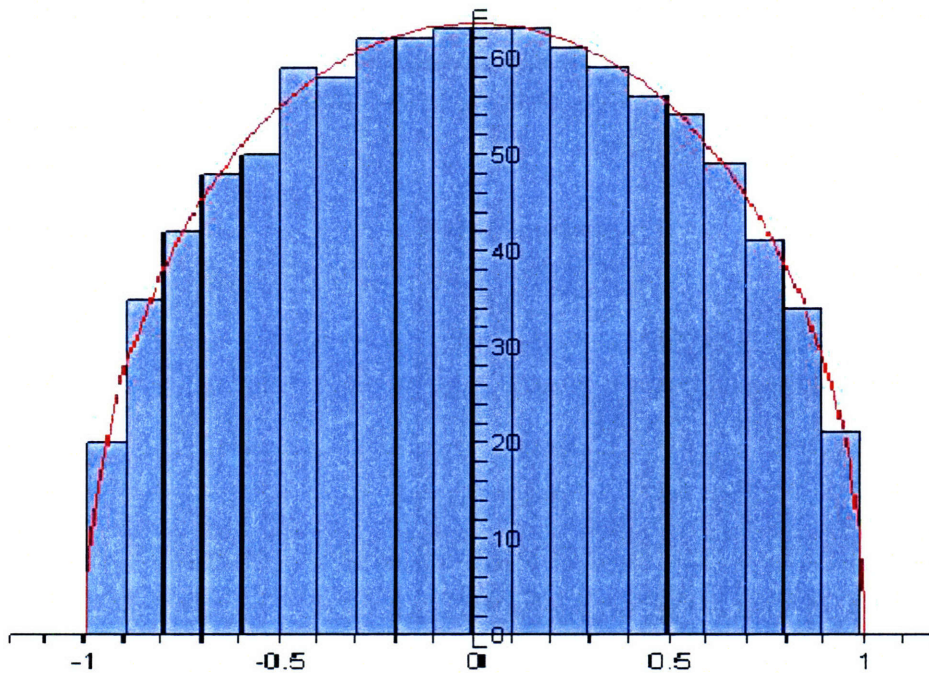


Figure 4 Wigner's Semicircle in Maple

Time Taken*:

Eigenvalue Solving: 545.58 s
Total: 1037.27 s

* Maple programs were executed on MIT Athena Machine. Using new Linear Algebra constructs may increase the performance significantly

Mathematica

```
<<Statistics`NormalDistribution`
<<Graphics`Graphics` (*import packages*)

n=1000;
A=RandomArray[NormalDistribution[],{n,n}]; (*random normal matrix*)
S = (A+Transpose[A])/((8*n)^0.5); (*symmetric matrix*)

eig=Eigenvalues[S]; (*calculating eigenvalues*)

hist = Histogram[eig, HistogramCategories->20, HistogramScale->1];
theory = Plot[(2/Pi)*(1-x*x)^0.5, {x,-1,1}] (*plot histogram and graph*)
```

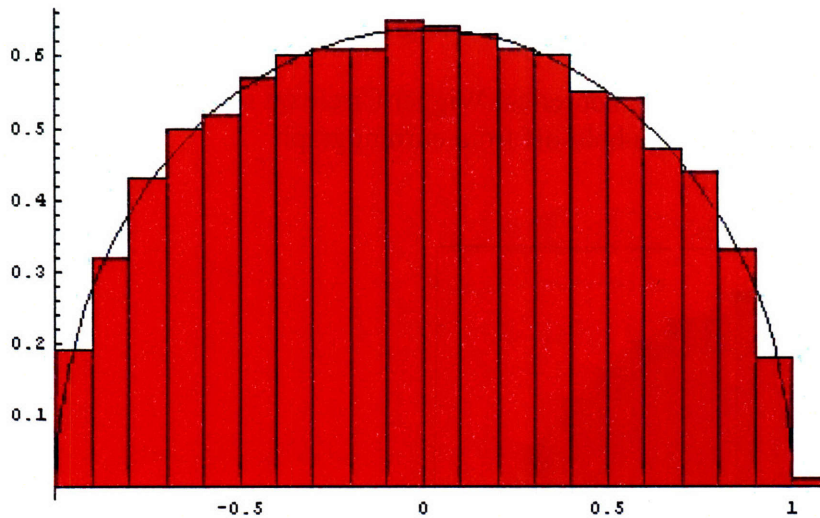


Figure 5 Wigner's semicircle in Mathematica

Time Taken:

Eigenvalue Solving: 4.03 s
Total: 4.78 s

Matlab

```
n=1000; % dimension of matrix

A = randn(n); % generate a normally distributed random matrix
S = (A+A')/(sqrt(8*n)); % a symmetric normally distributed matrix

e=eig(S); % vector of the eigenvalues of matrix S

hold off; % remove the hold on any previous graph
[N,x]=hist(e,-1:1:1); % defines the characteristics of the histogram
bar(x,N/(n*.1)); % plots the histogram
hold on; % holds the graph so that next plot comes on same graph

b=-1:1:1; % bound of theoretical Wigner Circle
y=(2/(pi))*sqrt(1-b.*b); % defines the theoretical Wigner Circle

plot(b,y); % plots the theoretical Wigner Circle on same graph
hold off; % takes the hold off from current graph
```

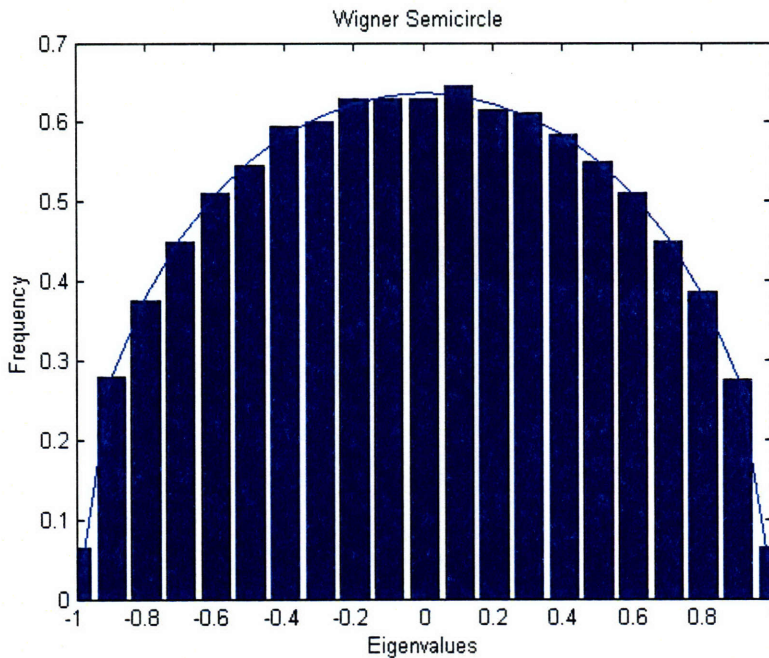


Figure 6 Wigner's semicircle in Matlab

Time Taken:

Eigenvalue Solving: 2.41 s
Total: 3.25 s

Octave

```
n=1000;  
A=randn(n,n);  
S=(A+A')/(2*(n^0.5));  
eigen = eig(S);  
  
[N,x]=hist(eigen,21);  
bar(x,N/(n*(x(2)-x(1))));  
hold on;  
b=-1:.1:1;  
y=(2*/(pi))*sqrt(1-b.*b);  
plot(b,y);
```

Comments: Similar to Matlab

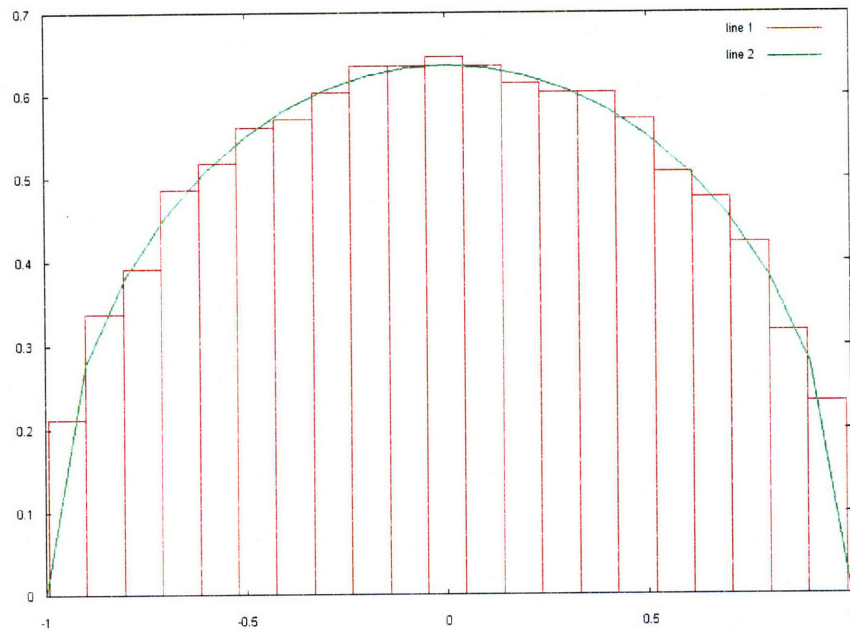


Figure 7 Wigner's semicircle in Octave

Time Taken:

Eigenvalue Solving: 2.12 s
Total: 3.94 s

Python

```
from numpy import *
from matplotlib import pylab
from pylab import *

n=1000
ra = random
A = ra.standard_normal((n,n))
S = (A + transpose(A))/(sqrt(8*n))

eig = linalg.eigvalsh(S)

subplot(121)
hist(eig, 20, normed =1)
grid(True)
ylabel('Frequency')
xlabel('Eigenvalues')

x = linspace(-1, 1, 21)
y = (2/(pi))*sqrt(1 - x*x)
pylab.plot(x, y, 'k-')
show()
```

import the packages from
which functions are used

dimension of matrix

generates a normally dist. random matrix
symmetric random normal matrix

vector of the eigenvalues of matrix S

a histogram is created

theoretical Wigner circle

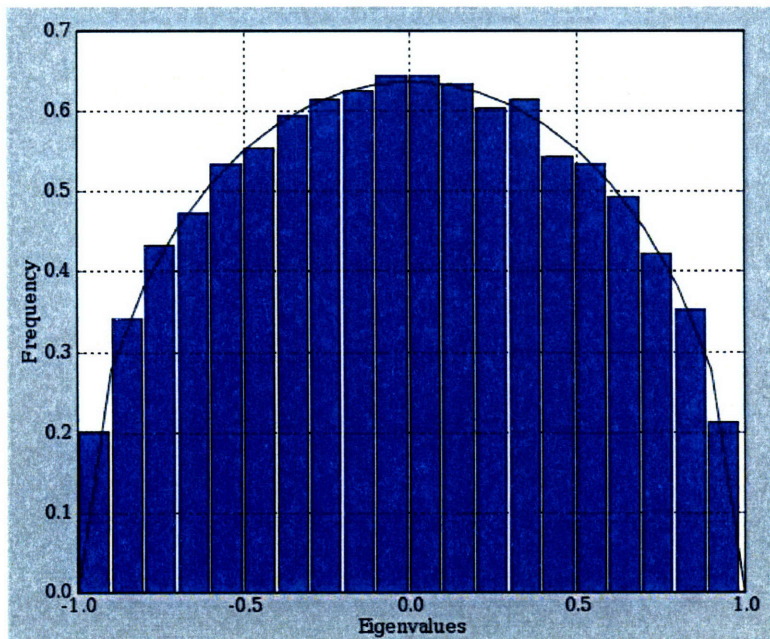


Figure 8 Wigner's semicircle in Python

Time Taken:

Eigenvalue Solving: 4.88 s
Total: 6.05 s

R

```
N<-500;
X<-rnorm(N*N);
Y<-matrix(X,nrow=N);
S = (Y+t(Y))/(sqrt(8*N));           # Symmetric random normal matrix

eigenvalues<-eigen(S)$values;      # calculating eigenvalues

hist(eigenvalues,21, freq=F, ylim =c(0, 0.7)); #Plot histogram

curve((2/(pi))*sqrt(1-x*x), -1,1, add=TRUE)    #Plot semicircle
```

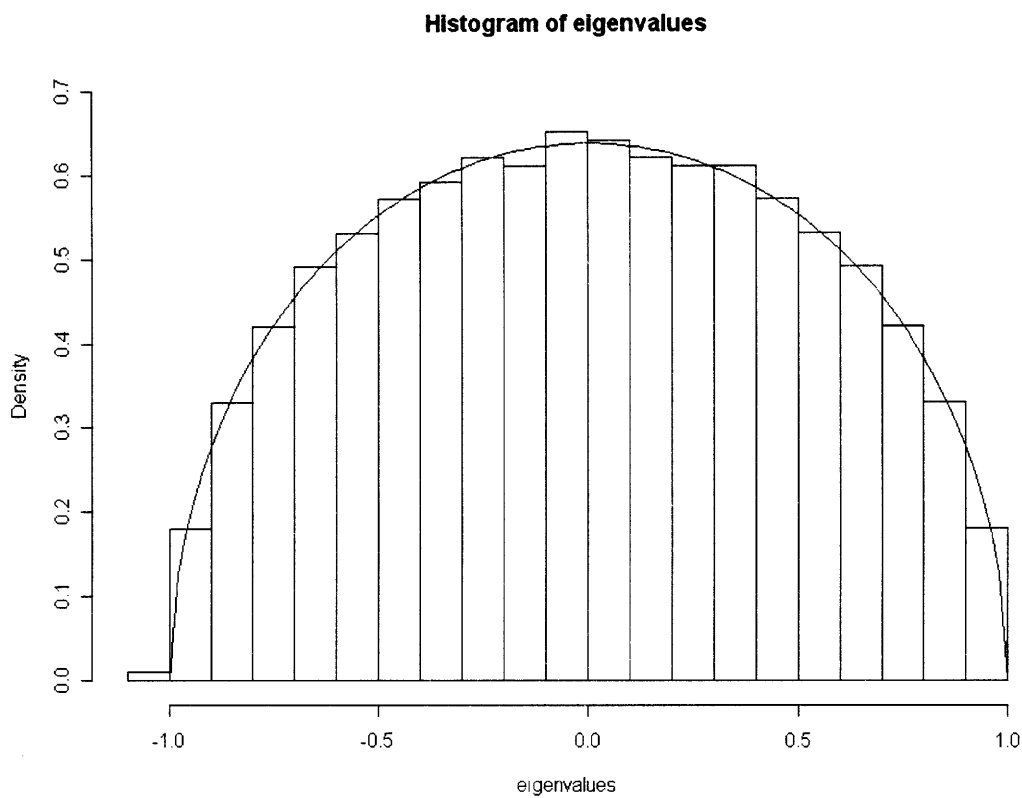


Figure 9 Wigner's semicircle in R

Time Taken:

Eigenvalue Solving: 8.48 s
Total: 9.29 s

Scilab

```
n=1000;  
  
rand('normal');  
A=rand(n,n);  
S=(A + A')/(8*n)^0.5;  
  
eigen = spec(S);  
histplot(20,eigen);  
  
b=-1:.1:1;  
y=(2/%pi)*sqrt(1-b.*b);  
plot(b,y);
```

Comments: Similar to Matlab

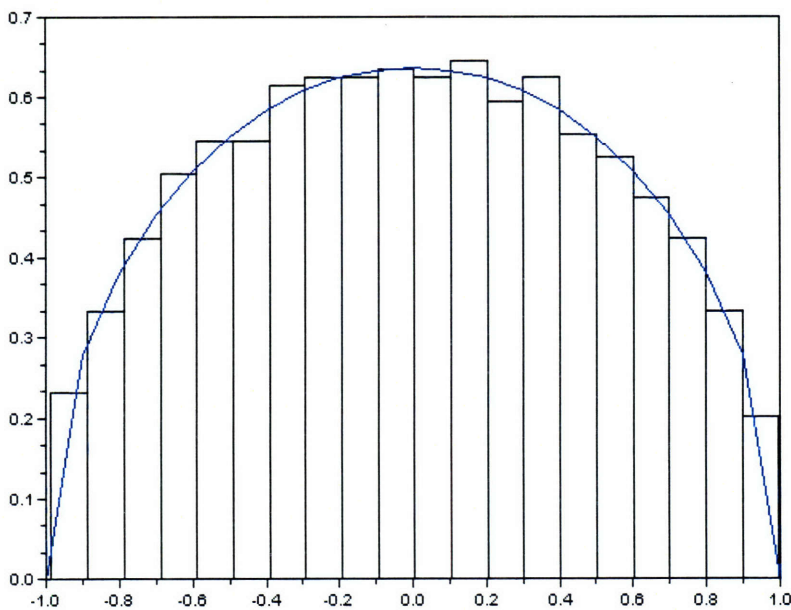


Figure 10 Wigner's semicircle in Scilab

Time Taken:

Eigenvalue Solving: 6.03 s
Total: 6.37 s

In most of the cases, most of the time taken is consumed in eigenvalues computation. All other processes are computationally very cheap compared to eigenvalue solving.

Comments

- Since Labview is a graphical language, it is very expressive and easy to code as well. As seen in Figure 2, the wires represent the flow of variables and blocks (known as Virtual Instruments or VIs) represent the Mathematical operations executed on them. By turning ON the context help, one can see the function of these blocks by placing the mouse cursor over them. Its detailed help also provides a very helpful search option to look for VIs.
- Maple requires importing some packages before using certain functions. Linalg package was imported (with(linalg):) to use the functions ‘transpose()’ and ‘eigenvals()’ and ‘display()’ function required importing with(plot,display). It is fairly easy to learn and use. It has a good help to search for functions.
- Mathematica, like Maple, also requires importing packages to use certain functions, as seen in the first two lines. Mathematica is very expressive as we can see its functions such as ‘RandomArray’, ‘NormalDistribution’, ‘HistogramCategories’ clearly represent the operation they are performing. Best sources to get help are online documentations and mailing lists.
- Matlab’s strongest point is its syntax which is extremely easy to learn and use. Its functions are short like eye() or randn(). Mathematica’s names (merged capitalized words like NormalDistribution[], RandomArray[] and IdentityMatrix[]) are long but extremely consistent and self-expressive. It is hard to say whether the expressiveness of these long command names is to be preferred over shorter names. For some functions (like fmincon in Chapter 4), Matlab requires installing additional packages, but after installing it can directly use those functions. It does not require importing packages like Maple or Mathematica.
- Octave is almost exactly like Matlab. Its main package is also very easy to learn and use as its syntax is similar to Matlab. But additional packages of Octave can be sometimes difficult to use.
- Python also requires installing and importing some packages to use certain functions as seen in first few lines. Most of the computational tasks require ‘numpy’ atleast. Python is not as easy to learn as Matlab. However, after gaining some experience, it is easy to code like Matlab.
- R is somewhat like Python in the level of difficulty to learn and use. It provides a decent search option in help for simple operations. For advanced help, one needs to use www.rseek.org as it is not easy to find help for R on general internet search engines.
- Scilab, like Octave, has a syntax very close to Matlab. Just like Matlab, it is also very easy to learn and use.

2.4 GigaFLOPS

FLOPS or Floating Point Operations per Second is a measure of computer's performance. For Multiplication of two random matrices, GigaFLOPS can be defined as:

$$GF = 2n^3/(t \times 10^9)$$

where $n \times n$ is the size of the matrices and t is the time taken for multiplication.

Language	GigaFLOPS			
	n=1000	n=1500	n=2000	n=2500
LabView	0.93	0.94	0.94	0.94
Mathematica	0.96	0.94	0.96	0.96
Matlab	0.94	0.94	0.94	0.94
Octave	0.60	0.64	0.64	0.55
Python	0.96	0.95	0.95	0.96
R	0.60	0.61	0.60	0.55
Scilab	2.20	2.22	2.44	2.41

Table 10 Performance of Languages through Matrix Multiplication

Chapter 3: Optimization

Most of the languages and software provide some or full support for Optimization Problems. Unlike other mathematical operations, these differ a lot in their approach and results because of the very nature of complexity of optimization problems. Numerous algorithms are available for specific types of problems but none of them can deal with any type of optimization problem efficiently. Therefore there does not exist a single solver which can perform very well for all kinds of problems

In this section, we will discuss various Optimization tools provided by scientific computing languages and also some packages developed solely for the purpose of Optimization.

3.1 Linear Programming

Since solving Linear Programming problems is not as complex as Non-Linear Programming, almost all of the packages mostly give us the correct answer. But packages do differ significantly in the amount of time taken to solve the problem.

Optimization Solvers

Solvers taken into consideration here are Linear Programming functions of Labview, Maple, Mathematica, Matlab, Scilab and IMSL(C).

IMSL Numerical Libraries, developed by Visual Numerics, contain numerous Mathematical and Statistical algorithms written in C, C#, Java and Fortran.

R's basic package does not give too many options for Optimization. `ConstrOptim` function can perform Constrained Linear Programming but requires the initial point to be strictly inside the feasible region. The problem considered here contains equality constraints, so we get an infeasibility error with `constrOptim`. There are additional packages for R which contain linear programming solvers.

Example

The Linear Programming problem taken into consideration here is:

Min

$$3 T_1 + 3 T_2 + 3 T_3 + 3 T_4 + 3 T_5 + 3 T_6 + 3 T_7 + 3 T_8 + 3 T_9 + 3 T_{10} + \\ 3 T_{11} + 2 T_{12} + 3 T_{13} + 3 T_{14} + 2 T_{15} + 3 T_{16} + 3 T_{17} + 3 T_{18} + 3 T_{19} + \\ 2 T_{20} + 3 T_{21} + 3 T_{22} + 3 T_{23} + 2 T_{24} + 2 T_{25} + 3 T_{26} + 2 T_{27} + 2 T_{28} + \\ 3 T_{29} + 3 T_{30} + 3 T_{31} + 3 T_{32} + 3 T_{33} + 3 T_{34} + 3 T_{35} + 3 T_{36} + 3 T_{37}$$

such that:

$$T_1 + T_{11} + T_{16} + T_{25} + T_{26} + T_{29} + T_{34} = 1 \\ T_2 + T_{12} + T_{13} + T_{19} + T_{27} + T_{28} + T_{30} + T_{33} + T_{35} = 1 \\ T_3 + T_{14} + T_{15} + T_{29} + T_{30} = 1 \\ T_4 + T_{16} + T_{20} + T_{22} + T_{25} + T_{27} + T_{29} + T_{31} + T_{34} + T_{36} = 1$$

$$\begin{aligned}
T_5 + T_{17} + T_{18} + T_{31} + T_{32} &= 1 \\
T_6 + T_{12} + T_{17} + T_{33} &= 1 \\
T_7 + T_{15} + T_{19} + T_{21} + T_{23} + T_{26} + T_{28} + T_{30} + T_{32} + T_{33} + T_{35} + T_{37} &= 1 \\
T_8 + T_{13} + T_{18} + T_{20} + T_{21} + T_{27} + T_{28} + T_{31} + T_{32} + T_{34} &= 1 \\
T_9 + T_{11} + T_{22} + T_{23} + T_{24} + T_{25} + T_{26} + T_{35} + T_{36} + T_{37} &= 1 \\
T_{10} + T_{24} + T_{36} + T_{37} &= 1 \\
\text{where all } T_i\text{'s } &\geq 0
\end{aligned}$$

Table 11 shows the time taken by various packages to solve the above problem:

Language/Software	Time(ms)
IMSL(C)	0.1
LabView	15
Maple	5.0*
Mathematica	0.8
Matlab	20
Scilab	1.8

Table 11 Performance of Linear Programming Solvers

* Maple programs were executed on an Athena Machine.

All of these converge to the optimum value $f_{opt} = 10$, but since the problem has multiple solutions, they converge to one of these two points:

$$x_{opt} = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.5449, 0, 0, 0, 0, 0.4551, 0, 0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0.4551, 1.0, 0, 0, 0.5449, 0, 0, 0, 0, 0]$$

or,

$$x_{opt} = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]$$

3.2 Non Linear Programming

A Non Linear Programming problem requires minimization (or maximization) of an objective function under some constraints where atleast one of the constraints or the objective function is non-linear. Most of the scientific computing environments support some simple forms of Non Linear Programming but not all have functions to solve general Non Linear Programming problems.

Optimization Solvers

The solvers taken into consideration are:

- Mathematica 5.2:
 - NelderMead

- SimulatedAnnealing
- RandomSearch
- Matlab 7.2:
 - fmincon
- OPT++ 2.4(on Linux):
 - OptQNIPS
- Python 2.4:
 - Scipy-Cobyla from OpenOpt package

OPT++ is an Object Oriented Nonlinear Optimization library developed by Patty Hough and Pam Williams of Sandia National Laboratories, Juan Meza of Lawrence Berkeley National Laboratory and Ricardo Oliva, Sabio Labs, Inc. The libraries are written in C++.

OpenOpt is an Optimization package developed by SciPy community, initially for Matlab/Octave in 2006. It was later rewritten and can be used for Python with Numpy. The solver taken into consideration is Scipy Cobyla. OpenOpt is also compatible with ALGENCAN solver which is considered better than Scipy Cobyla. ALGENCAN has not been taken into consideration here.

3.2.1 Convex Programming

Convex programming is relatively easy when compared to Non-Convex Programming as we have only one optimum in convex programming unlike several local minima in non-convex programming. Still, Non-Linear Optimization is a complex problem and we can see differences in various solvers when solving convex programming problems.

Problem

Mathematically, the problem solved here is:

$$\text{Min } x_1 + x_1^2 + 2x_2^2 + 3x_3^2$$

such that:

$$x_1 - 2x_2 - 3x_3 \geq 1 \dots\dots\dots (1)$$

$$x_2 + x_3 \leq 4 \dots\dots\dots (2)$$

$$x_1x_3 \geq 4 \dots\dots\dots (3)$$

where,

$$-20 \leq x_1 \leq 10$$

$$-10 \leq x_2 \leq 10$$

$$0 \leq x_3 \leq 10$$

Solution

The optimum for above problem is achieved at $x_{opt} = \{3.10, -0.885, 1.29\}$ and the minimum function value at x_{opt} is 19.273

Mathematica

All three, Nelder Mead, Simulated Annealing and Random Search of Mathematica perform very well for convex problems. They converge to the same solution irrespective of whether the starting point is feasible or not.

Matlab

Matlab's `fmincon` converges to the correct solution if the starting point is strictly inside the feasible set. But it may fail to converge if the starting point is outside or on the boundary of feasible set. For the current problem, the algorithm converges if the starting point satisfies the following conditions: $x_3 > 0$ or, $x_3 = 0$ & $x_1 > 0$ The main problem arises because of the 3rd constraint. When both x_1 & x_3 are negative in the initial point, 3rd constraint is satisfied, but the upper and lower bounds on the variables are not satisfied. If the algorithm satisfies the bounds of x_3 , 3rd constraint is violated. This might be stopping the algorithm to get to the feasible region and therefore algorithm fails to converge in this case.

Python

Python's Scipy Cobyla also converges to the optimum if the starting point is strictly inside the feasible set. Otherwise, it may end up terminating algorithm by saying 'No Feasible Solution.' For the current problem, Scipy Cobyla algorithm converges if the starting point satisfies $x_1 > 0$ or $x_3 > 0$, which is almost similar to `fmincon`'s condition.

OPT++

OptQNIPS solver is again very similar to Matlab's `fmincon` in terms of giving solution. If the starting point is strictly inside the feasible set, the algorithm converges. Otherwise it may or may not converge. For the current problem, OptQNIPS will always converge if the starting point satisfies: $x_3 > 0$ or, $x_3 = 0$ & $x_1 \geq 0$, which is again very similar to `fmincon`'s condition.

3.2.2 Non Convex Programming

Unlike convex programming, non convex programming has multiple solutions, which makes the search of the global optimum very difficult. Local optimums are not very difficult to find, but getting to the Global Optimum requires a search of almost the entire feasible region. So, there is a trade-off between the time taken by the algorithm and the accuracy of the solution.

Problem

The problem used for comparison is a quadratically-constrained problem which is: Given a 4×4 real matrix A , find an orthonormal matrix Q which minimizes the trace ($A*Q$). To summarize: Given a 4×4 matrix A Min: trace ($A*Q$) where, Q is an orthonormal matrix

For this analysis, the given matrix A is taken as:

$$A = M + c*I$$

Where M is an arbitrary fixed matrix

$$M = \begin{bmatrix} 1 & 4 & 5 & 7 \\ 6 & 7 & 3 & 0 \\ 3 & 7 & 3 & 2 \\ 3 & 8 & 6 & 8 \end{bmatrix}$$

I is a 4X4 identity matrix and c is an integer varying from -10 to 10. So each value of c actually corresponds to a different objective function. The starting point taken for each solver is Q = Identity Matrix which is a feasible (but not optimum) solution for this problem.

Solution

Table 12 given below gives the optimum values achieved by various solvers:

c	Nelder Mead	Random Search	Simulated Annealing	fmincon	Scipy Cobyla	OPT++
-10	-39.94	-39.94	-39.94	-31.36	-	-31.36
-9	-38.12	-38.12	-38.12	-30.07	-	-30.07
-8	-36.34	-36.34	-36.34	-29.18	-	-
-7	-34.61	-34.61	-34.61	-28.56	-	-
-6	-32.96	-32.96	-32.96	-28.14	-	-
-5	-31.43	-31.43	-27.85	-27.85	-	-27.85
-4	-30.04	-30.04	-30.04	-27.66	-	-27.66
-3	-27.57	-28.89	-28.89	-27.56	-	-27.56
-2	-28.06	-28.06	-27.58	-27.58	-	-27.58
-1	-27.81	-27.81	-27.66	-27.81	-	-27.81
0	-28.55	-28.55	-28.55	19	-28.54	-28.55
1	-28.34	-30.22	-28.34	-30.22	-28.34	-30.22
2	-32.74	-32.74	-32.74	-32.74	-29.62	-32.74
3	-31.46	-35.75	-31.46	-35.75	-31.46	-
4	-33.38	-39.05	-33.38	-39.05	-33.38	-
5	-35.32	-42.53	-35.32	-42.53	-35.32	-
6	-46.13	-46.13	-46.13	-46.13	-46.13	-
7	-49.81	-49.81	-49.81	-49.81	-49.81	-
8	-53.55	-53.55	-53.55	-53.55	-53.55	-
9	-57.34	-57.34	-57.34	-57.34	-57.34	-
10	-61.16	-61.16	-61.16	-61.16	-61.16	-

Table 12 Accuracy of Optimization Solvers

Fig. 11 gives a comparison of the optimum values achieved by various solvers. Since this is a minimization problem, the solver whose graph remains lowest is the best.

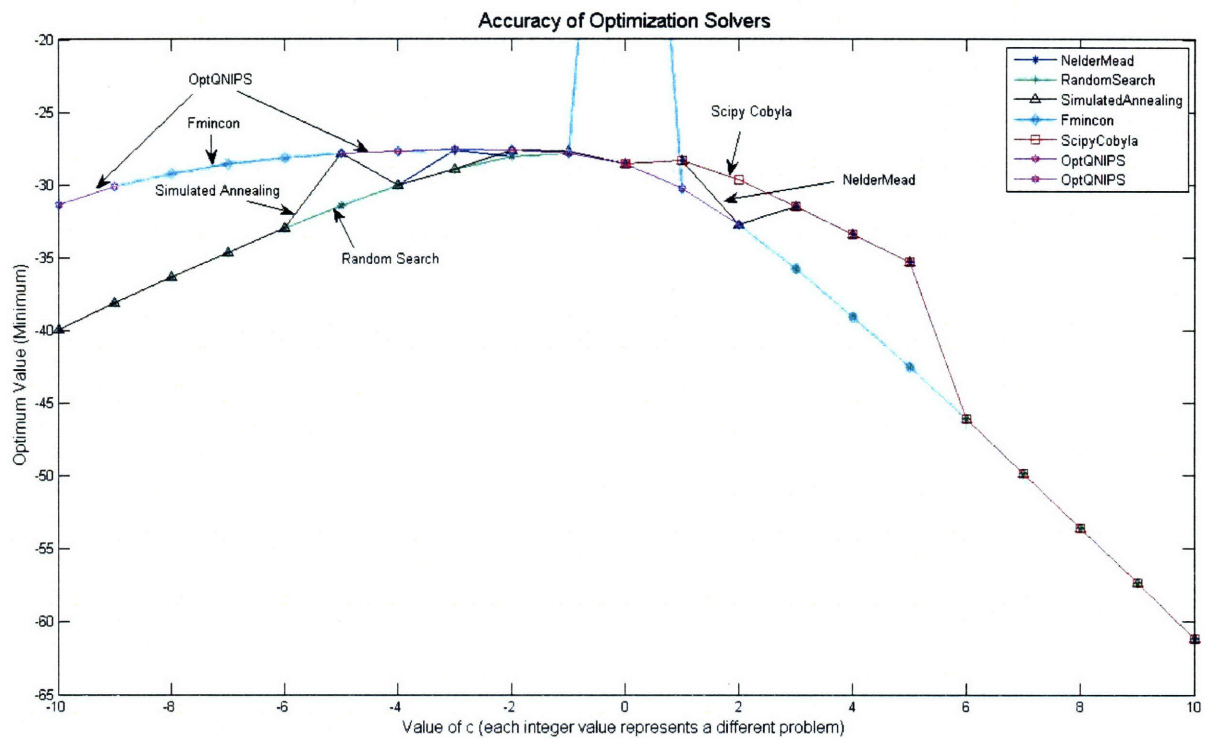


Figure 11 Accuracy of Optimization Solvers

Next graph (Fig. 12) gives a comparison of the time taken by various solvers. RandomSearch takes time of the order of 25-35 seconds, so it is not included in this graph. Also, OPT++ is a Linux software, so its time is also not compared since all others were executed on a Windows machine.

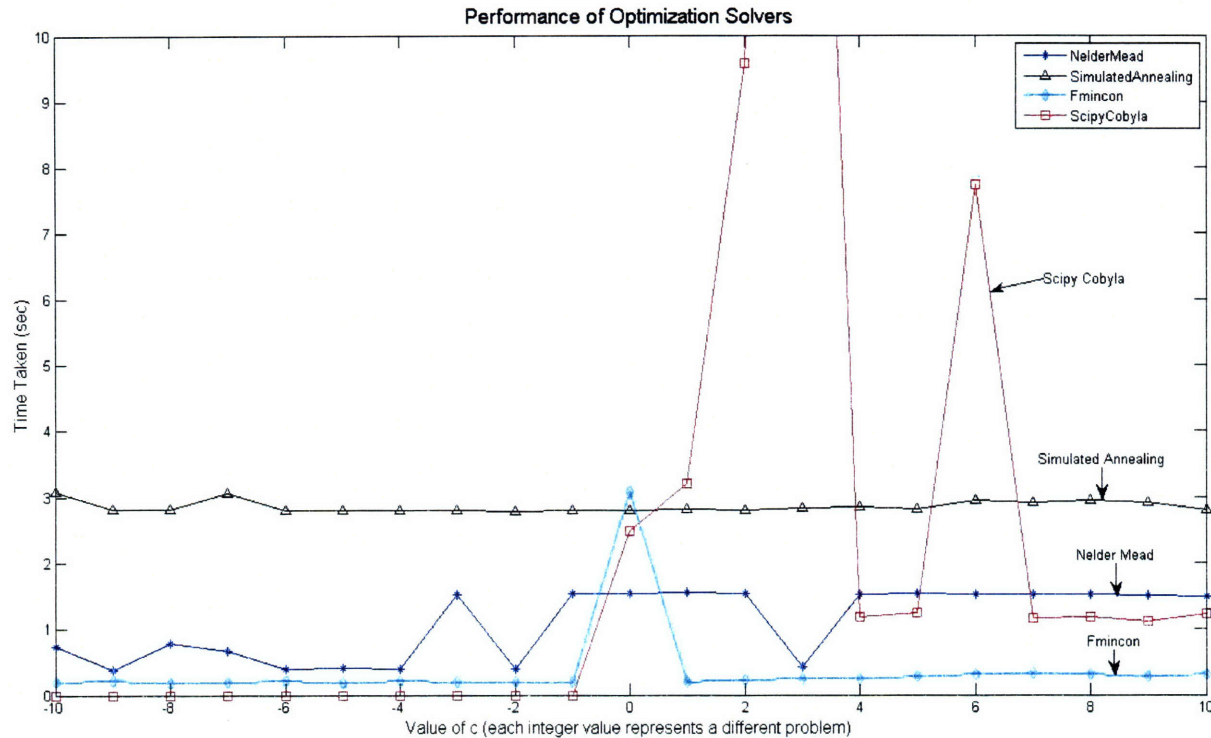


Figure 12 Time taken by Optimization Solvers

Comments

- Random Search from Mathematica performs the best in terms of the accuracy of solution. It always gets the best optimum solution. But it is slower than the others by an order of approximately 10 times.
- fmincon from the Optimization Toolbox of Matlab is the fastest in reaching the solution, and quite reasonable in terms of an accurate answer. But it is not as accurate as the Random Search. It might return a local optimum instead of the global optimum. Global optimum can be achieved by varying the starting point over the feasible region. But that would require a lot of time as the feasible region and number of variables increase.
- Nelder Mead from Mathematica performs somewhat like fmincon of Matlab. It is a little slower than fmincon, but sometimes returns a more accurate solution. But in some cases it also returns a local optimum instead of a global one.
- Simulated Annealing from Mathematica has almost similar performance as Nelder Mead in terms of accuracy of solution. But it is considerably slower (approx. 2-3 times) in achieving the answer.
- Scipy-Cobyla from OpenOpt package of Python is also not perfect with giving optimum solution. It closely follows Nelder Mead method from Mathematica. But it has got serious problems with tolerances. Constraint tolerance needs to be adjusted frequently to get a feasible solution. Even if the initial point is feasible, the user can get a 'No Feasible Solution' message (The blank spaces in the graph and table represent this case). It is

mostly as fast as Nelder Mead method, but sometimes takes exceptionally high time to solve.

- f) OptQNIPS from OPT++ almost traces the characteristics of the results of fmincon. But it does not give a solution at all for a wide range of starting points (The blank spaces in the graph and table represent these cases where an optimal solution was not achieved and the algorithm terminated in between). In these cases, the algorithm reaches the limit of 'Maximum number of allowable backtrack iterations' and increasing this limit also does not help in getting an answer. We cannot compare time for OPT++ since this is LINUX based software while all the others are executed on Windows. OPT++ is also difficult to install and program as compared to other VHHLs given here.

Chapter 4: Miscellaneous

4.1 Ordinary Differential Equations

Most of the ODE solvers considered in this analysis are included in the main software and does not require installing additional packages. Here we have used numerical solvers, but some software can also perform symbolic computations.

For Python, we have used FiPy 1.2 which is a Finite Volume based PDE solver. It was primarily developed in the Metallurgy Division and Center for Theoretical and Computational Materials Science (CTCMS), in the Materials Science and Engineering Laboratory (MSEL) at the National Institute of Standards and Technology (NIST).

Problem

Consider the following Boundary Value Problem:

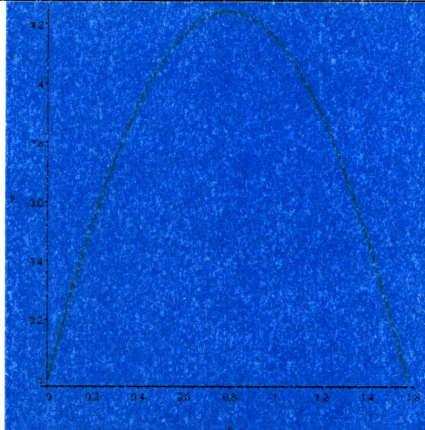
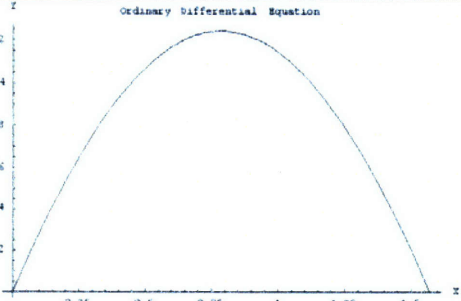
$$y''(x) + y(x) = 0$$

Given:

$$y(x = 0) = 3$$

$$y(x = \pi/2) = 3$$

Table 13 shows the results we get by solving the above problem.

Language	Time(sec)	Graphical Solution
Maple	0.026*	
Mathematica	0.016	

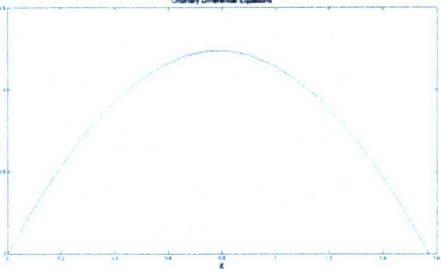
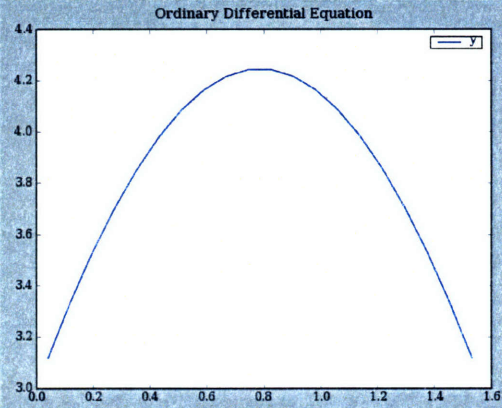
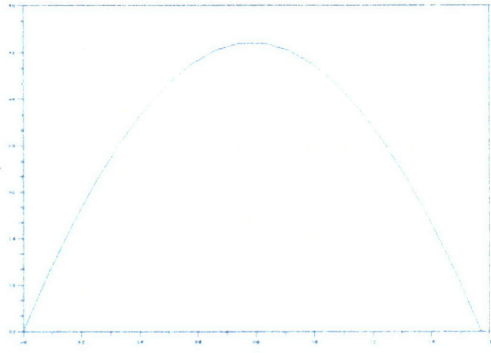
Matlab	0.026	
Python	0.093	
Scilab	0.078	

Table 13 Ordinary Differential Equation Solvers

* On MIT Athena Machine

Comments

- Maple, Mathematica and Matlab are fairly easy to use for solving ODEs. All three of them can also find symbolic solutions for ODEs.
- Scilab is also easy to use, but requires a few more lines of codes than above three. It cannot solve ODEs symbolically.
- FiPy is harder to use than all others. There is also lack of documentation and online support as it does not have too many users yet.

4.2 Memory Management

Each Language has its own way of storing data, and therefore we can see differences in memory related issues. Here we consider creating the largest $n \times n$ random matrix in each Language and

compare their performance. All these simulations were done under identical circumstances to ensure that each software had access to equal amount of memory. Results are displayed below in Table 14.

Language	Largest Possible Matrix	Matrix not Allowed	Error Message
LabView	$10^3 \times 10^3$	$10^4 \times 10^4$	LabView: Memory is Full
Mathematica	$10^3 \times 10^3$	$10^4 \times 10^4$	No more memory available. Mathematica Kernel has shut down.
Matlab	$10^4 \times 10^4$	$10^5 \times 10^5$	Maximum variable size allowed by the program is exceeded
Octave	$10^4 \times 10^4$	$10^5 \times 10^5$	Memory Exhausted – trying to return to prompt
Python	$10^4 \times 10^4$	$10^5 \times 10^5$	ValueError: dimensions too large
R	$10^4 \times 10^4$	$10^5 \times 10^5$	Error in matrix(0, 1e5, 1e5): too many elements specified
Scilab	$10^3 \times 10^3$	$10^4 \times 10^4$	Stack size exceeded!

Table 14 Memory utilization across Languages

Comments

- Mathematica’s Kernel shuts down as soon as memory is exhausted and it is not possible to do any further computation. So, any results obtained previously cannot be stored and the information is lost. All other Languages keep working and it is possible to retrieve previously calculated information. One can check the memory already exhausted by using `MemoryInUse[]`, before performing any memory exhausting exercise.
- In Matlab, `feature('memstats')` gives a nice break-up of the available memory.
- In R, `memory.size()` returns the memory in use and `memory.limit()` returns the total memory available for use. After using the command listed in Table 15 to free the memory, `memory.size()` still shows the memory used by deleted variables. To make the memory, which is not associated with any variable anymore, available to the system, perform garbage collection using `gc()`.
- `stacksize()` in Scilab returns the memory available and memory in use. `stacksize(100000)` will increase/decrease the available memory to 100000.

To free space and clear all the defined variables, use the functions given in Table 15

Language	Function to Free Memory
LabView	Edit → Reinitialize values to default
Mathematica	<code>ClearAll["Global`*"];</code> <code>Remove["Global`*"];</code>

Maple	restart;
Matlab	clear
Octave	clear
Python*	del x
R	rm(list=ls())
Scilab	clear

Table 15 Free Memory

* only deletes the variable x

4.3 Activity Index

The popularity of a language can be estimated by how quickly you get help whenever you are in trouble. Higher number of users implies faster you are going to get responses for your problem. Most common places to ask questions on Languages are Usenet groups and mailing lists. Table 16 presents the activity on one of the most active communities for these languages.

Language	Usenet Group/Mailing List	Approx. emails per day	
		New Topics	Total
LabView	comp.lang.labview	35	115
Maple	comp.soft-sys.math.maple	2	10
Mathematica	comp.soft-sys.math.mathematica	10	45
Matlab	comp.soft-sys.math.matlab	55	175
Octave	help-octave@octave.org	2	15
Python	python-list@python.org	25	150
R	R-help@r-project.org	25	105
Scilab	comp.soft-sys.math.scilab	4	15

Table 16 Activity Index

Python's numpy mailing list also has a very high activity as most of the scientific computing work in Python is not possible without numpy.

Chapter 5: Incompatibility Issues

In mathematics, we usually have one correct answer for a problem. But there exists differences in numerical answers within languages. Then, we have some cases where more than one correct answer does exist, and in the absence of a Standard for Mathematical Results, we see differences in the solutions returned by languages. In Spanish, the verb “embarazar” does not mean “to embarrass”, which itself makes an embarrassing point. Similarly, when numerical languages give differing answers, the language suffers an embarrassing demerit in the minds of users.

Here, we will be discussing some cases where languages fail to agree with each other. These results will help in setting up a standard for numerical computations, absence of which often results in confusions in the mind of users.

5.1 Undefined Cases

Mathematically, zero to the power zero is undefined. But in many cases during scientific computation, it is better to avoid getting an expression like “undefined”. So, many languages define it as 1. Table 17 shows the results obtained in various languages.

Language	Result
LabVIEW	1
Maple	$0^0=1$ $0^{0.0}=\text{Float(undefined)}$
Mathematica	Indeterminate
Matlab	1
Octave	1
Python	1
R	1
Scilab	1

Table 17 Ambiguity in Zero raised to power Zero

5.2 Sorting of Eigenvalues

As seen in chapter 3, the performances of eigenvalue solvers differ significantly across languages. But we also observe differences in the result we get. The array of eigenvalues obtained is not consistent. Table 18 shows the sorting of the array we obtain from eigenvalue solvers of different languages.

Language/Software	Sorting
LabVIEW	Descending order of the absolute value of eigenvalues
Maple	Ascending

Mathematica	Descending order of the absolute value of eigenvalues
Matlab	Ascending
Octave	Ascending
Python	Ascending
R	Descending
Scilab	Ascending

Table 18 Sorting of Eigenvalues

5.3 Cholesky Decomposition

Cholesky Decomposition factorizes a Symmetric Positive Definite matrix into a Left and a Right triangular matrix which are transpose of each other. But again, there are no standards which determine whether the solution returned from Cholesky Decomposition should be the Lower triangular matrix or the Upper triangular.

Suppose we have a symmetric positive-definite matrix A . In the absence of a defined standard output, there could be 4 different representations:

$$A = \begin{matrix} L'L \\ LL' \\ R'R \\ RR' \end{matrix}$$

and the returned matrix could be one of these four. The table below shows the kind of representations used by various languages.

Language	Returns (Lower or Upper)	Representation
Maple	L	$A=LL'$
Mathematica	R	$A=R'R$
Matlab	R	$A=R'R$
Octave	R	$A=R'R$
Python	L	$A=LL'$
R	R	$A=R'R$
Scilab	R	$A=R'R$

Table 19 Cholesky Decomposition

5.4 Matlab vs Octave

Octave is highly compatible with Matlab and both are expected to return same answers for same operations. But the same commands in both may yield different results.

QR Factorization

Consider the following matrix:

```
>> A=ones(4)
```

```
A =
```

```
 1  1  1  1
 1  1  1  1
 1  1  1  1
 1  1  1  1
```

In Matlab, we get:

```
>> qr(A)
```

```
ans =
```

```
-2.0000 -2.0000 -2.0000 -2.0000
 0.3333 -0.0000 -0.0000 -0.0000
 0.3333  0.3660  0  0
 0.3333  0.3660  0  0
```

While in Octave:

```
octave:30> qr(A)
```

```
ans =
```

```
-2.0000e+00 -2.0000e+00 -2.0000e+00 -2.0000e+00
 5.0000e-01 -9.6148e-17 -9.6148e-17 -9.6148e-17
 5.0000e-01  5.7735e-01 -5.0643e-33 -5.0643e-33
 5.0000e-01  5.7735e-01  7.0711e-01 -1.1493e-49
```

Numerical Inconsistency

Consider following operation:

```
log2(2^n) - n
```

For powers of 2, Matlab returns the exact answer while Octave may not.

For n = 1000

In Matlab:

```
>> log2(2^n)-n
```

```
ans =
```

```
0
```

```
In Octave:
octave:32> log2(2^n)-n
ans = 1.1369e-13
```

5.5 Sine Function

For large input arguments, the sine function sometimes returns incorrect answers. Table 20 shows the results we get for an input of $x = 2^{64}$.

Language	Sin(2^{64})
Google Calculator	0.312821315
Maple	0.0235985099044395581
Mathematica	0.023598509904439558634 0.312821
Matlab	0.02359850990444
Octave	0.0235985099044396
Python	0.247260646309 0.312821315 0.023598509904
R	0.24726064630941769
Scilab	0.0235985099044395581214

Table 20 Sine value for large input argument

Correct answer = 0.0235985099044395586343659....

We not only see inconsistency across languages, but also within a language based on operating system and processor architecture. As shown in Table 20, we get different values in same language while working on different platforms. Mathematica and Python have been observed to give the correct answer on some 64-bit Linux machines.

5.6 Growing Arrays

Execute the following commands in Matlab.

```
>> x=[]
x =
 []

>> x(2,:)=1
x =
 0
 1
```


Which is not acceptable as the first column dimension was never touched. Matlab gives the correct result if the above operation is executed as:

```
>> x=[]  
x =  
    []
```

```
>> x(2,1:end)=1  
x =  
Empty matrix: 2-by-0
```

We get similar results in Octave too. But in Octave we also get:

```
octave:1> v=[]  
v = [](0x0)
```

```
octave:2> v(:)=1  
v = 1
```

which is again a flaw, but we do not observe this in Matlab.

```
>> v=[]  
v =  
    []
```

```
>> v(:)=1  
v =  
    []
```


Chapter 6: Conclusion

An analysis of several scientific computing environments has been presented here. We Covered issues such as ease of learning/using and Language Compatibility. The results and codes presented here will be uploaded on a website along with some more results covering other mathematical applications.

We expect software to improve with every release, and that vendors and developers may disagree with our initial impressions. We invite such alternative viewpoints and will certainly correct errors as they reach our attention.

6.1 About the website

Presently, the website is in the form of a password protected wiki. The Main Page has been divided into several sections based mainly on Mathematical areas. There exists an individual page for each topic and each topic is listed on the Main Page under the area of Mathematics it deals with. This structure should help the users to browse through the pages of a particular area that they mostly work in.

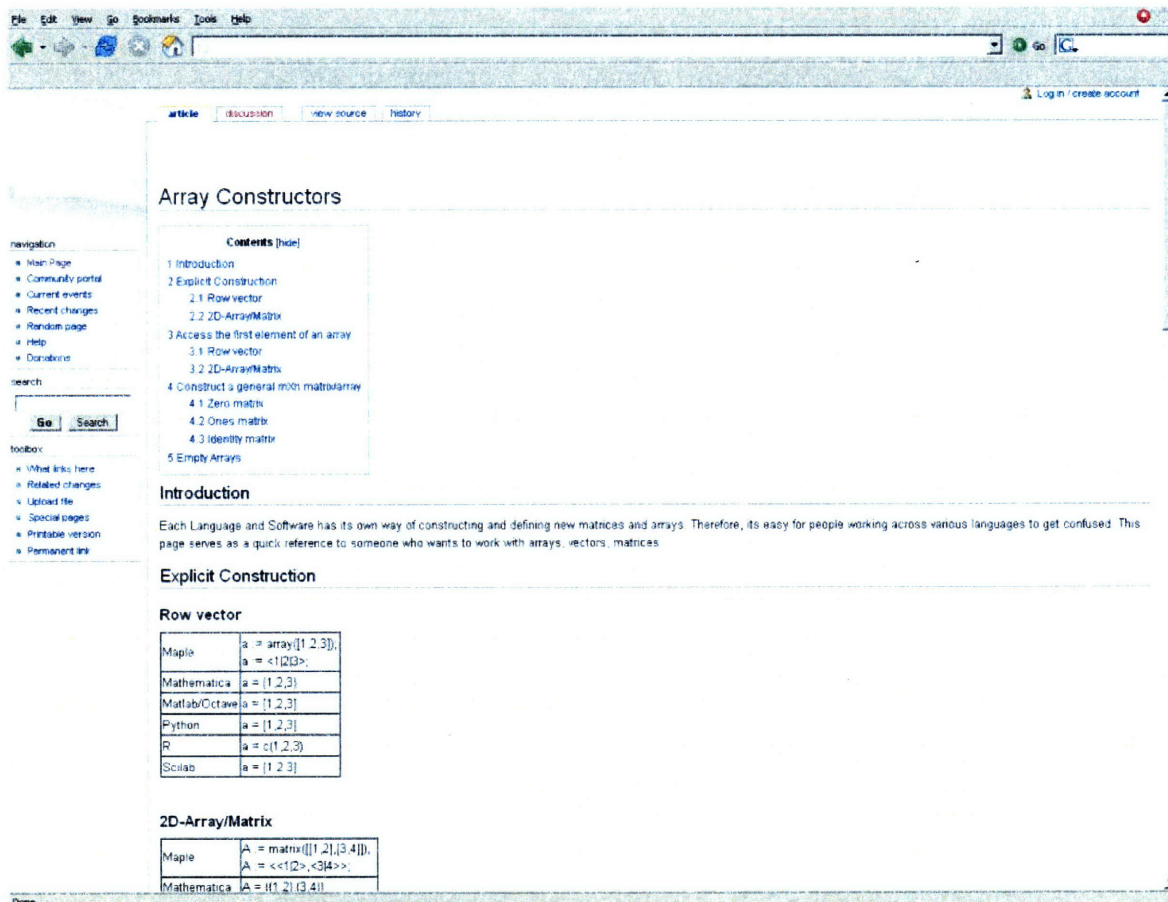


Figure 13 A screenshot of the website

The search option of the wiki software helps in finding the exact function or topic one is interested in. Since the wiki is focused only on Languages and Software, the search is expected to return almost exactly what user wants which is not the case in Wikipedia or Google search. Apart from this, a page contains syntactical keywords from each language. So, someone proficient in one language and looking for an equivalent in another language can easily search for the keyword from the language he already knows.

Most of the information, such as performance represented by time taken and syntax by putting exact functions, has been presented in tabular form. Simplicity is the key for effective communication. An attempt has been made to keep the pages and tables presenting data as clean and short as possible. Verbosity prevents the user from going ahead with anything and therefore, obvious points are avoided wherever possible.

Direct links are also provided for downloading files of the Languages and Software, making it easier for the users to start executing their first codes very early in their learning process. Few initial codes working and printing results on the screen act as a moral booster to go ahead with the process.

6.2 Future Work

The wiki has an option of editing pages which is intended to be open to public after some time. People with experience in any scientific computing language or software are expected to contribute.

Apart from the issues discussed in this analysis, we also hope to put a rating system based on ease of usability and performance of a language. We hope this to develop into a complete package containing information regarding a large number of Languages and Software. It should be like an encyclopedia of languages where any user having trouble with scientific trouble can get his or her answers.

Since new versions of languages and high speed computers keep appearing in the market, the website should also be updated frequently. With the arrival of higher speed computers, the weightage given to ease of use should be increased as compared to the performance of a particular language. Therefore, there is a need to keep updating the site to match the developments in the software and hardware industry. This should be taken care of by the community involved in using this website just like we see in the case of Wikipedia.

Bibliography

- [1] Steinhaus, Stefan 2008, Comparison of Mathematical Programs for Data Analysis, Number crunching test report Edition 5, The Scientific Web, viewed 13 May 2008, <<http://www.scientificweb.de/ncrunch/>>
- [2] Wester, Michael, Timothy Daly and Alexander Hulpke, Last modified 2007, Rosetta Stone, Axiom, viewed 13 May 2008, <<http://axiom-wiki.newsynthesis.org/RosettaStone>>
- [3] Wikipedia Users, Last modified 30 April 2008, Comparison of Programming Languages, Wikipedia, Wikimedia Foundation Inc. viewed 13 May 2008, <http://en.wikipedia.org/wiki/Comparison_of_programming_languages>
- [4] Wikipedia Users, Last modified 13 May 2008, Comparison of Programming Languages (basic instructions), Wikipedia, Wikimedia Foundation Inc., viewed 13 May 2008, <http://en.wikipedia.org/wiki/Comparison_of_basic_instructions_of_programming_languages>