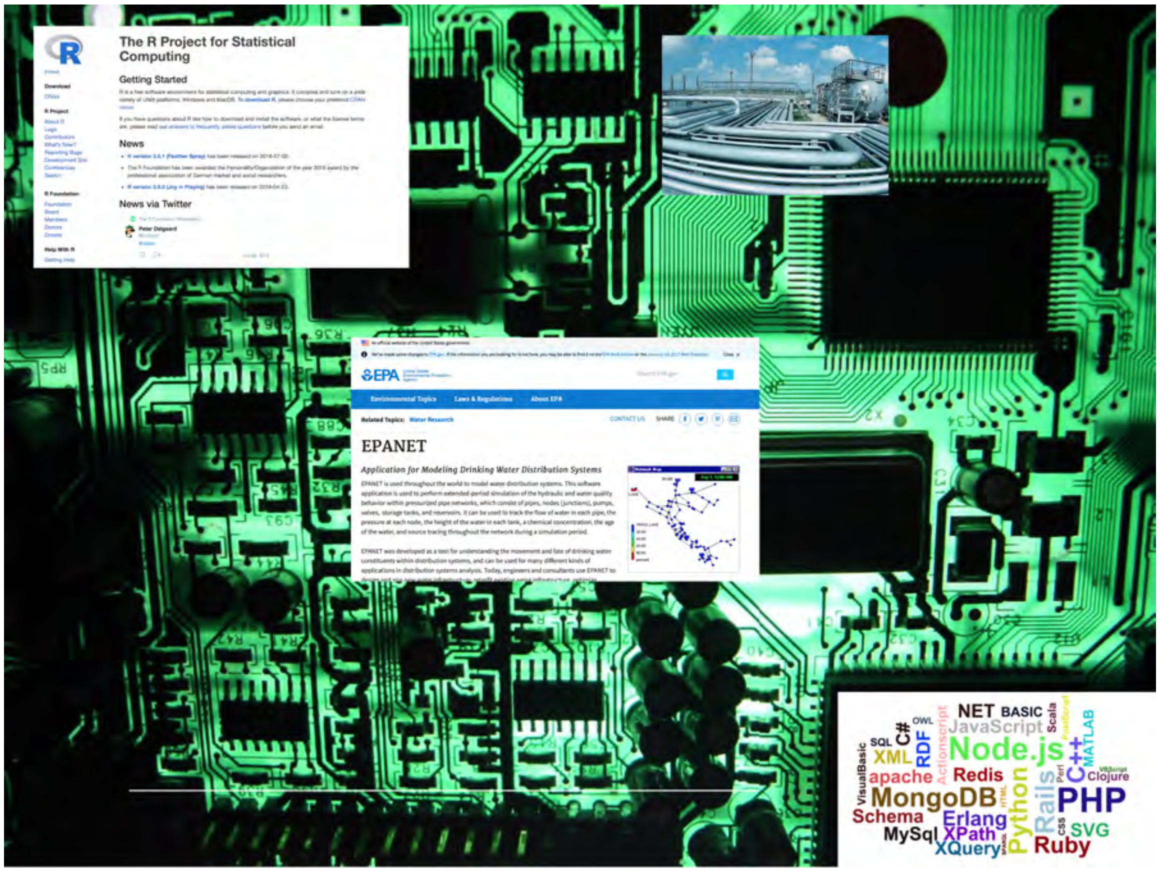
Hydraulic Network Modeling using R and EPANET

An Introduction for ICT Students

by
Theodore G. Cleveland
Department of Civil, Environmental, and Construction Engineering
Texas Tech University



Contents

1 Introduction and Getting Started

In the 1990s Civil Engineering programs reduced programming courses in a effort to recover hours for other topics – a logical decision at the time, but with some consequences. The philosophy was that engineers would not need to be able to write computer programs, but instead just use them. Microsoft Excel and Lotus 1-2-3 were the dominant spreadsheet software programs (Borland QuatroPro was a close third), and with macro instruction capability, much legitimate engineering computation could be performed within these tools. In fact I developed Excel spreadsheets that could solve multi-dimensional diffusion problems (3D groundwater flow) using fully implicit finite difference methods. These spreadsheets were slow relative to MODFLOW, but you could watch the solutions evolve – ultimately the process was deemed a waste, because of the ever present "... there is no longer a need for engineers to be able to write programs." Later on I developed spreadsheets to perform pressurized pipe network simulation, gradually varied flow simulation, and rudimentary water-hammer and St-Venant equation solutions. The spreadsheets were never really practical (yes they worked well, produced the same results as professional products, but were always intended a pedagogical tools), but they proved an important point – if you could teach a computer to follow an algorithm it made you a more self-help user of the professional tools.

In 2014 several of my students expressed desire to understand programming – they all know how to write code, but feel they don't know how to build algorithms (and implement them). This workbook is an attempt to remedy that student self-identified weakness. I conducted several one-to-one classes (as special topics); they learned a lot, I learned even more. This book is a tribute to their interests.

The workbook plan is to introduce a programming tool – I have selected **R** because it has a rich development environment already available, graphics is almost trivial, then apply that tool to selected hydraulics problems of practical value. In the end the reader ends up with a toolkit that can either stand-alone, or more likely supplement professional tools they will eventually use.

R is freeware, but it is built and maintained by a consortium of programmers and statisticians. They have evolved the environment to work on most of the main architectures (MacOS, Windows, Linux); there are even parallel processor and GPU builds available, and a company called RStudio provides the APIs to even run it server side. Much of the underlying code is C, C++, and well proven FORTRAN routines. [1]

[1] This entire document is a work in progress – As of 20-AUG-2018 it was complete only as far as the Introduction to EPANET. As the document is developed updates will be placed onto the class repository.

1.1 About R

R is a open source envrionment that runs on Windows, Linux/UNIX, and Mac OS X. The individual binaries are unique to each OS and architecture, but **R** "source" is interchangeable among machines. With very minor differences, an **R** script will run equally well on any machine.

R is a statistical analysis tool, it is also a programming tool and language, it is also a nearly "publication" quality graphics tool. Naturally all this capability comes at a cost (especially since the software is distributed for "free") — learning to do more than simple calculations takes some time (not much), but the skill is highly perishable. You will need to keep notes, or copies of your **R** scripts for future reference. Relearning after some time away from **R** is pretty simple, so the modeler only has to pay the steep learning cost once.

The remainder of this essay shows how to obtain and install **R** on a Windows machine. Macintosh and Linux installs are accomplished in a similar fashion. For the truly insane, the entire envrionment can be built from source on any machine with PERL, gcc, and gfortran compliers (default in Linux, easy to obtain for other architectures).

1.2 Getting Started

The first step required (for using **R** as a programming tool) is to install **R** on your computer. The source of the binary builds is the same regardless of the underlying operating system – the Comprehensive R Archive Network (CRAN for short). The remainder of this chapter shows how to get the tool running on the three main operating systems in current practice.

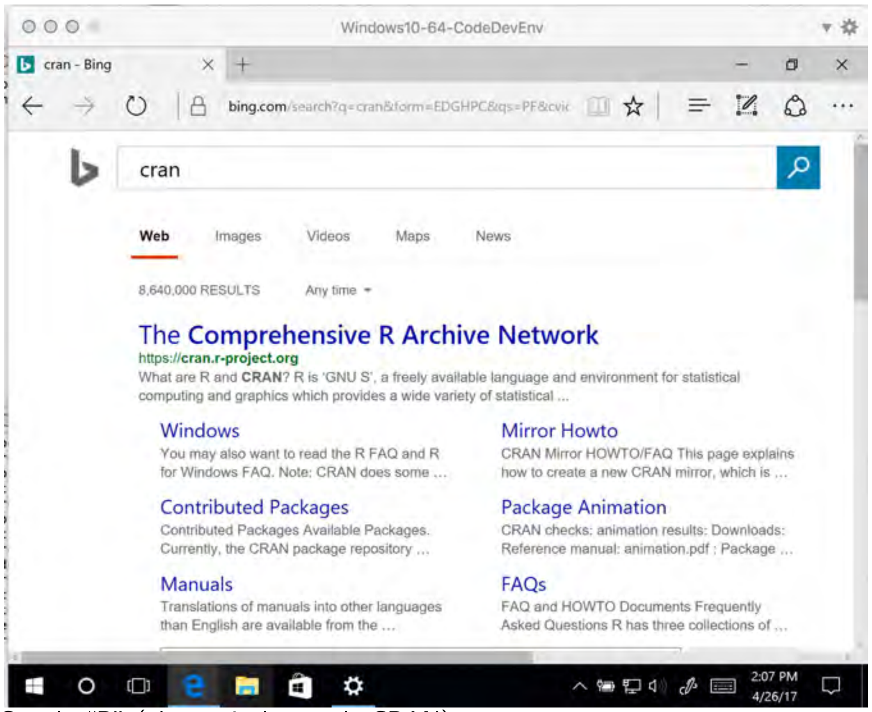
1.2.1 Windows Users

The purpose of this section is to demonstrate how to get **R** running on a Windows computer. This document assumes the following:

1. You have internet connection.
2. You have sufficient user privileges to install software on your machine. (If you need someone else to install, I did my install by running the installer as a local administrator — obviously you need the password)
3. You have 60MB or so of vacant disk space on the system directory.

The step-by-step guide is presented as a series of screen captures. Obviously adjust inputs to fit your machine. The version in these screen captures is quite dated — use the most recent, stable version offered on CRAN (Comprehensive R Archive

Network).[2]



The Comprehensive R Archive Network
https://cran.r-project.org
What are R and **CRAN**? R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical ...

Windows
You may also want to read the R FAQ and R for Windows FAQ. Note: CRAN does some ...

Mirror Howto
CRAN Mirror HOWTO/FAQ This page explains how to create a new CRAN mirror, which is ...

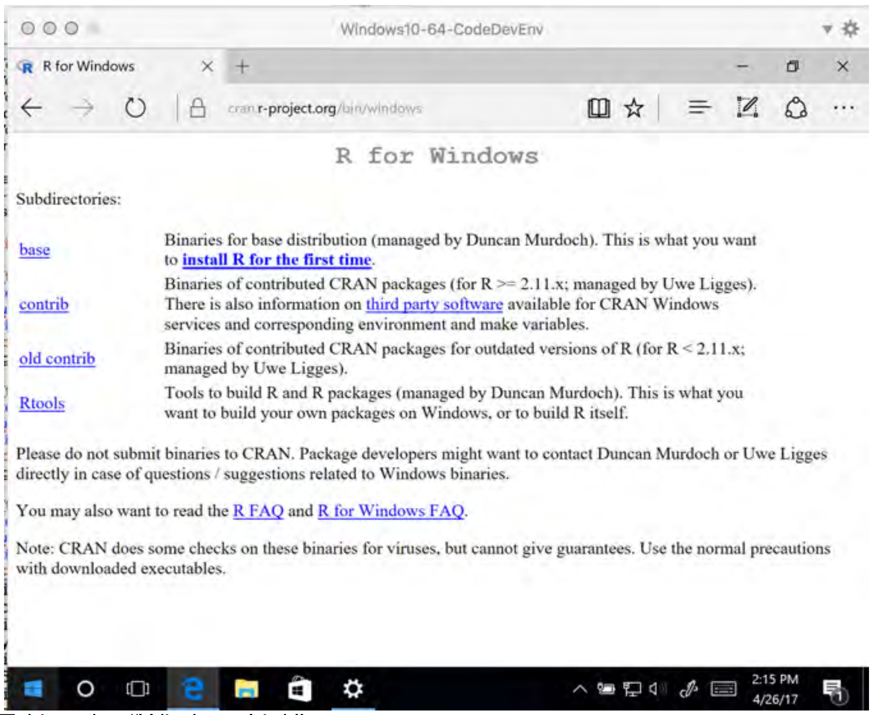
Contributed Packages
Contributed Packages Available Packages. Currently, the CRAN package repository ...

Package Animation
CRAN checks: animation results: Downloads: Reference manual: animation.pdf : Package ...

Manuals
Translations of manuals into other languages than English are available from the ...

FAQs
FAQ and HOWTO Documents Frequently Asked Questions R has three collections of ...

Figure 1. Google "R" (alternatively google CRAN).



R for Windows

Subdirectories:

base	Binaries for base distribution (managed by Duncan Murdoch). This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 2.11.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.11.x; managed by Uwe Ligges).
Rtools	Tools to build R and R packages (managed by Duncan Murdoch). This is what you want to build your own packages on Windows, or to build R itself.

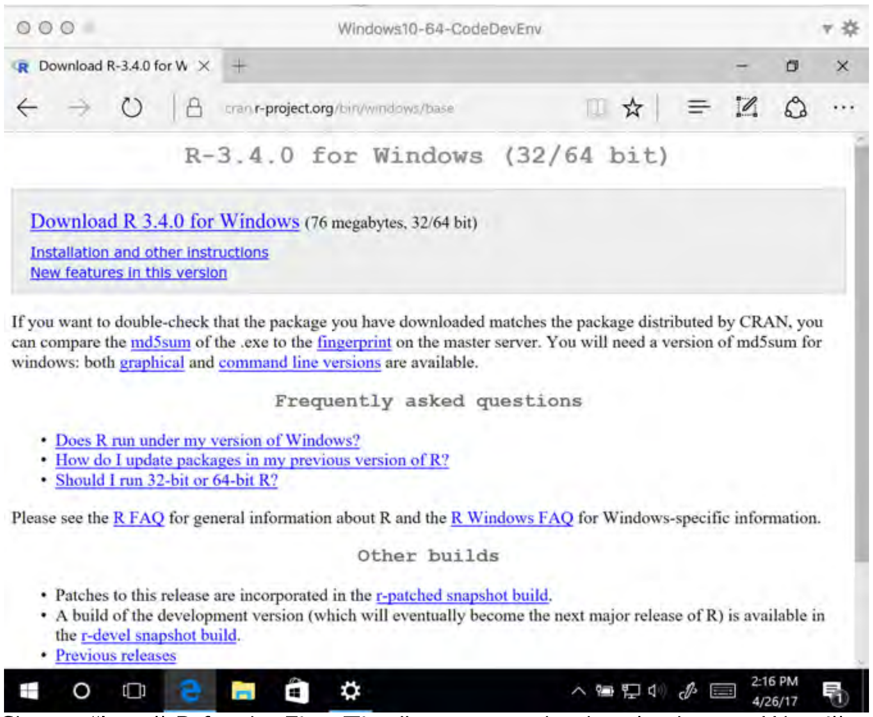
Please do not submit binaries to CRAN. Package developers might want to contact Duncan Murdoch or Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Figure 2. Taking the "Windows Link".

[2]I have updated the screen captures for Windows 10 — so these should replicate the steps.



R-3.4.0 for Windows (32/64 bit)

Download R 3.4.0 for Windows (76 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

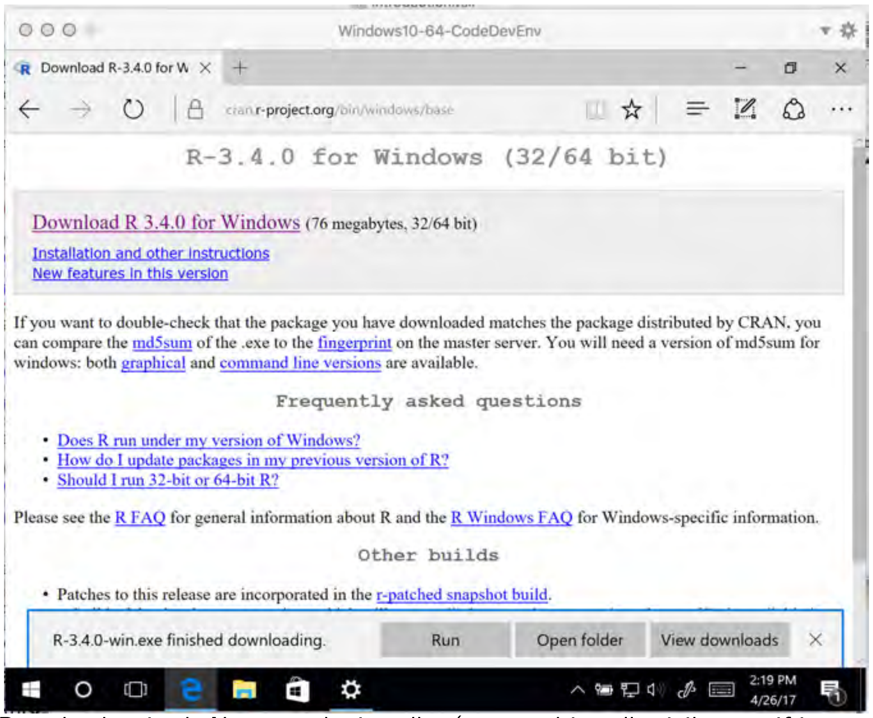
- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Figure 3. Choose "Install R for the First Time" – goes to the download page. We will next select "Download R . . . " and it should download a windows installer. Choose "save" when prompted..



R-3.4.0 for Windows (32/64 bit)

Download R 3.4.0 for Windows (76 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

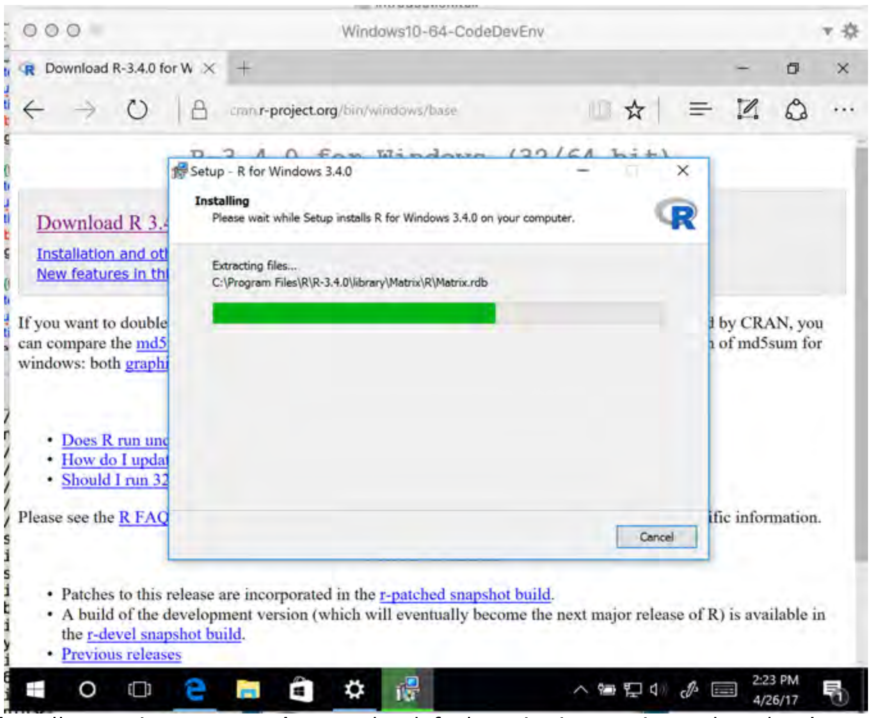
Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Other builds

- Patches to this release are incorporated in the r-patched snapshot build.

R-3.4.0-win.exe finished downloading.

Figure 4. Download arrived. Now run the installer (you need install privileges – if its your personal laptop, then your regular user account should work). .



Installing
Please wait while Setup installs R for Windows 3.4.0 on your computer.

Extracting files...
C:\Program Files\R\R-3.4.0\library\Matrix\R\Matrix.rdb

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Figure 5. Installer run in progress. Accept the defaults – its just easier and works. Later you can re-install elsewhere in your filesystem..

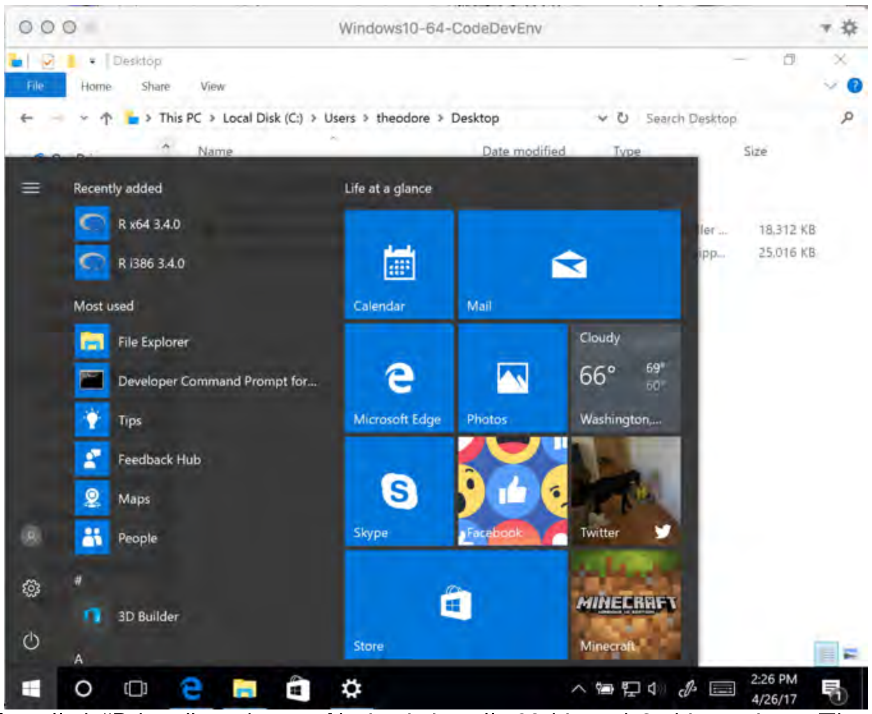
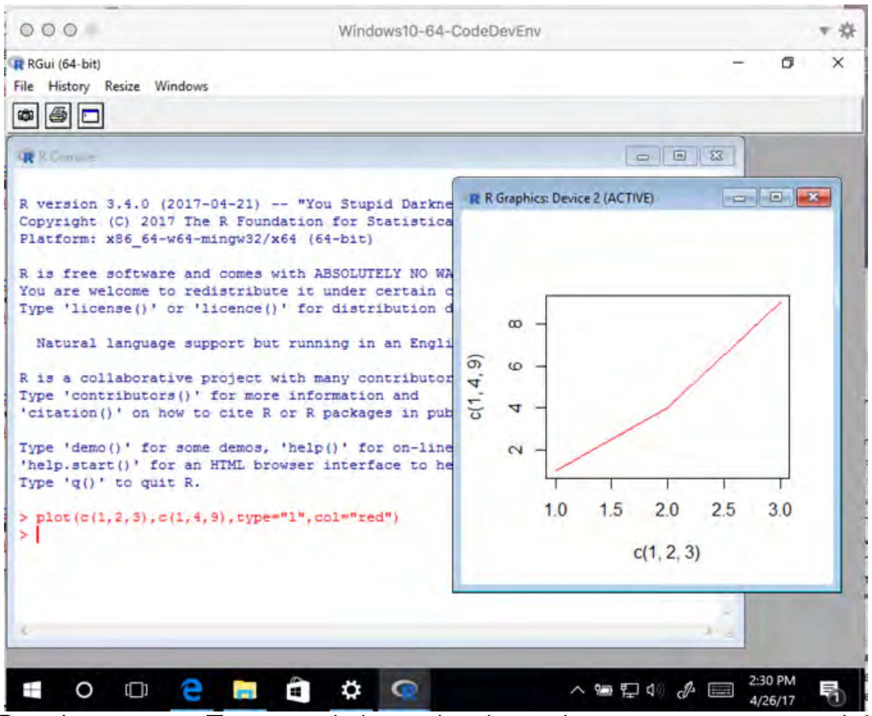
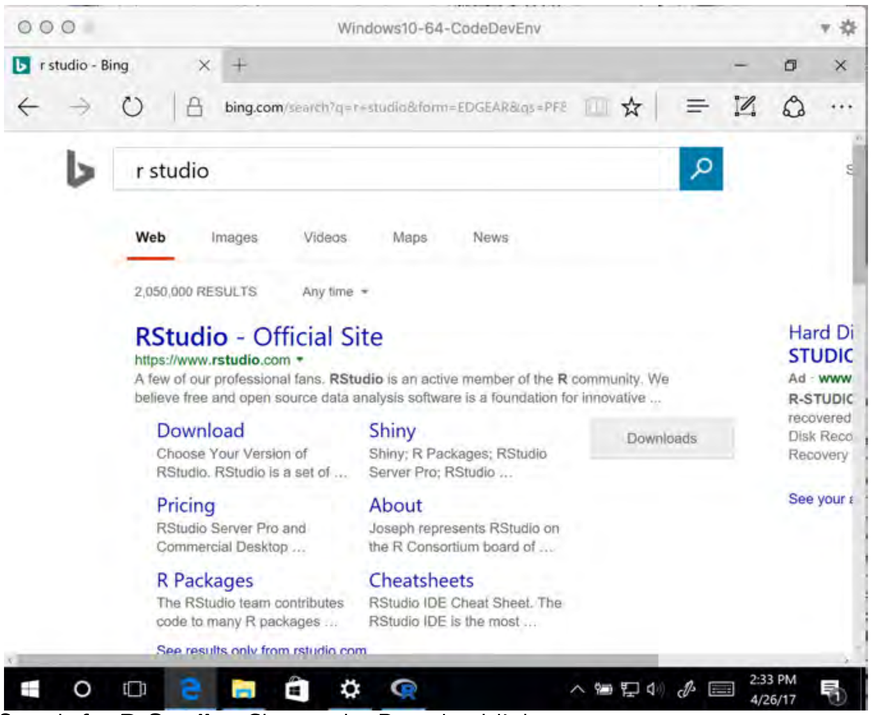


Figure 6. Installed "R base" packages. Notice it installs 32-bit and 64-bit versions. The next step is to verify the install by trying to run the program..



```
R version 3.4.0 (2017-04-21) -- "You Stupid Darkne
Copyright (C) 2017 The R Foundation for Statistica
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WA
You are welcome to redistribute it under certain c
Type 'license()' or 'licence()' for distribution d

  Natural language support but running in an Engli

R is a collaborative project with many contributor
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in pub

Type 'demo()' for some demos, 'help()' for on-line
'help.start()' for an HTML browser interface to he
Type 'q()' to quit R.

> plot(c(1,2,3),c(1,4,9),type="l",col="red")
> |
```

Figure 7. Run the program. Type `plot(c(1,2,3),c(1,4,9),type="l",col="red")` into the console window. A plot should be generated as shown. If this works, then your install is good and now we install **R Studio**.



RStudio - Official Site
https://www.rstudio.com
A few of our professional fans. **RStudio** is an active member of the **R** community. We believe free and open source data analysis software is a foundation for innovative ...

Download
Choose Your Version of RStudio. RStudio is a set of ...

Shiny
Shiny; R Packages; RStudio Server Pro; RStudio ...

Pricing
RStudio Server Pro and Commercial Desktop ...

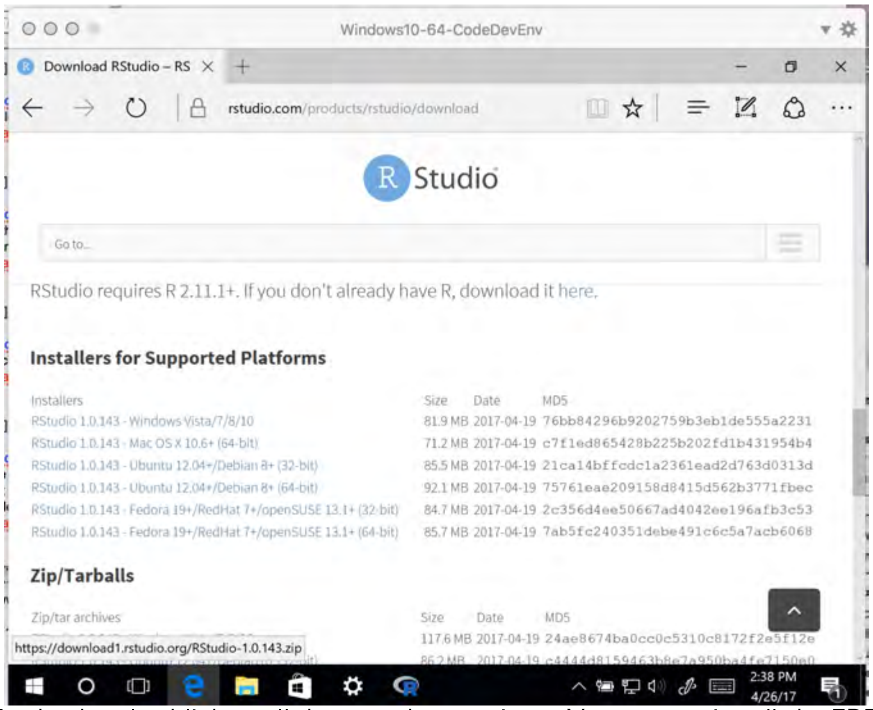
About
Joseph represents RStudio on the R Consortium board of ...

R Packages
The RStudio team contributes code to many R packages ...

Cheatsheets
RStudio IDE Cheat Sheet. The RStudio IDE is the most ...

See results only from rstudio.com

Figure 8. Search for **R Studio**. Choose the Download link..


RStudio requires R 2.11.1+. If you don't already have R, download it here.

Installers for Supported Platforms

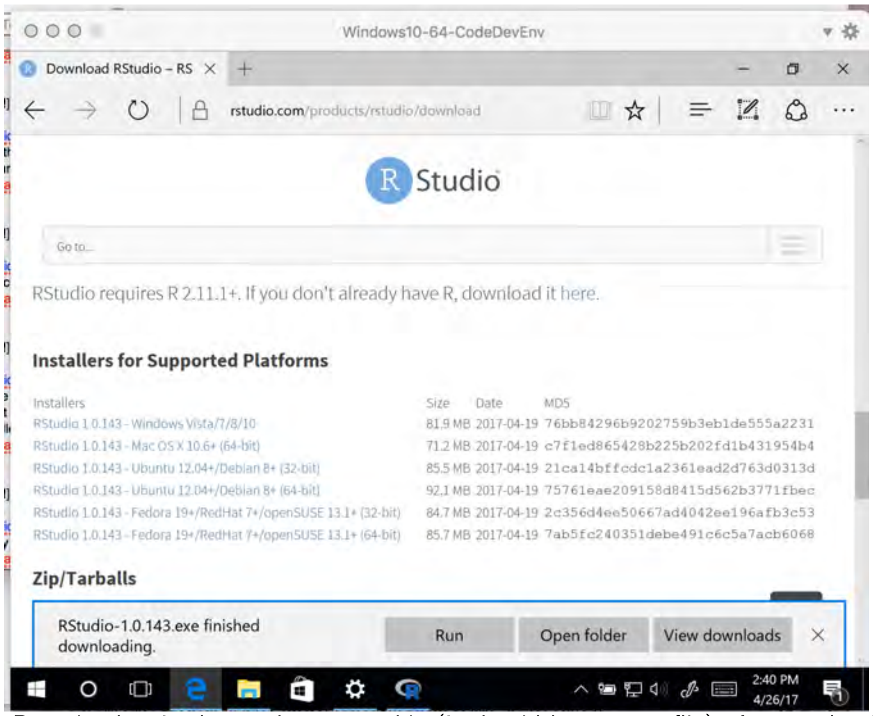
Installers	Size	Date	MD5
RStudio 1.0.143 - Windows Vista/7/8/10	81.9 MB	2017-04-19	76bb84296b9202759b3eb1de555a2231
RStudio 1.0.143 - Mac OS X 10.6+ (64-bit)	71.2 MB	2017-04-19	c7f1ed865428b225b202fd1b431954b4
RStudio 1.0.143 - Ubuntu 12.04+/Debian 8+ (32-bit)	85.5 MB	2017-04-19	21ca14bffcdc1a2361ead2d763d0313d
RStudio 1.0.143 - Ubuntu 12.04+/Debian 8+ (64-bit)	92.1 MB	2017-04-19	75761eae209158d8415d562b3771fbec
RStudio 1.0.143 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	84.7 MB	2017-04-19	2c356d4ee50667ad4042ee196afb3c53
RStudio 1.0.143 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	85.7 MB	2017-04-19	7ab5fc240351debe491c6c5a7acb6068

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
[illegible]	117.6 MB	2017-04-19	24ae8674ba0cc0c5310c8172f2e5f12e
[illegible]	86.2 MB	2017-04-19	c4444d8159463b8e7a950ba4fe7150e0

https://download1.rstudio.org/RStudio-1.0.143.zip

Figure 9. In the download link scroll down to the repository. You want to install the FREE Desktop version. If on a windows machine, it is the top most of the installers. Don't accidentally download the Zip/Tarballs – all that is source code and without the compilers you cannot make much use of it. Building from source is a challenge. Choose the windows installer and download, select "save" when prompted.


rstudio.com/products/rstudio/download

RStudio requires R 2.11.1+. If you don't already have R, download it here.

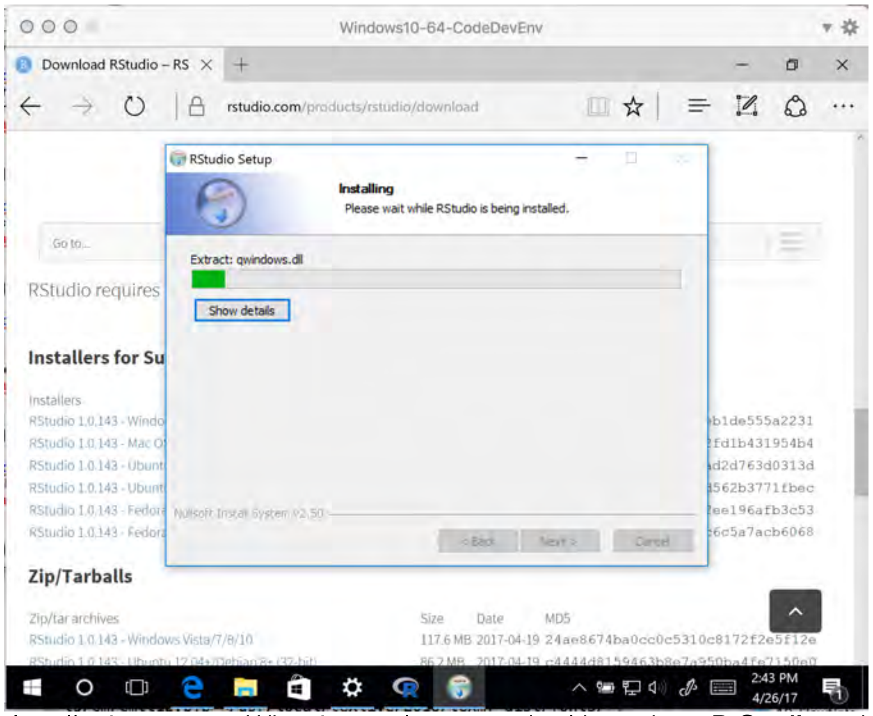
Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.0.143 - Windows Vista/7/8/10	81.9 MB	2017-04-19	76bb84296b9202759b3eb1de555a2231
RStudio 1.0.143 - Mac OS X 10.6+ (64-bit)	71.2 MB	2017-04-19	c7f1ed865428b225b202fd1b431954b4
RStudio 1.0.143 - Ubuntu 12.04+/Debian 8+ (32-bit)	85.5 MB	2017-04-19	21ca14bffcdc1a2361ead2d763d0313d
RStudio 1.0.143 - Ubuntu 12.04+/Debian 8+ (64-bit)	92.1 MB	2017-04-19	75761eae209158d8415d562b3771fbec
RStudio 1.0.143 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	84.7 MB	2017-04-19	2c356d4ee50667ad4042ee196afb3c53
RStudio 1.0.143 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	85.7 MB	2017-04-19	7ab5fc240351debe491c6c5a7acb6068

Zip/Tarballs

RStudio-1.0.143.exe finished downloading.

Figure 10. Download arrived, run the executable (it should be a .exe file). Accept the defaults during the install.


RStudio Setup

Installing

Please wait while RStudio is being installed.

Extract: qwindows.dll

Nullsoft Install System v2.50

Zip/tar archives	Size	Date	MD5
RStudio 1.0.143 - Windows Vista/7/8/10	117.6 MB	2017-04-19	24ae8674ba0cc0c5310c8172f2e5f12e

Figure 11. Installer in-progress. When it completes, you should now have **R Studio** and **R** installed. We will test the **R Studio** install using the same simple plot call.

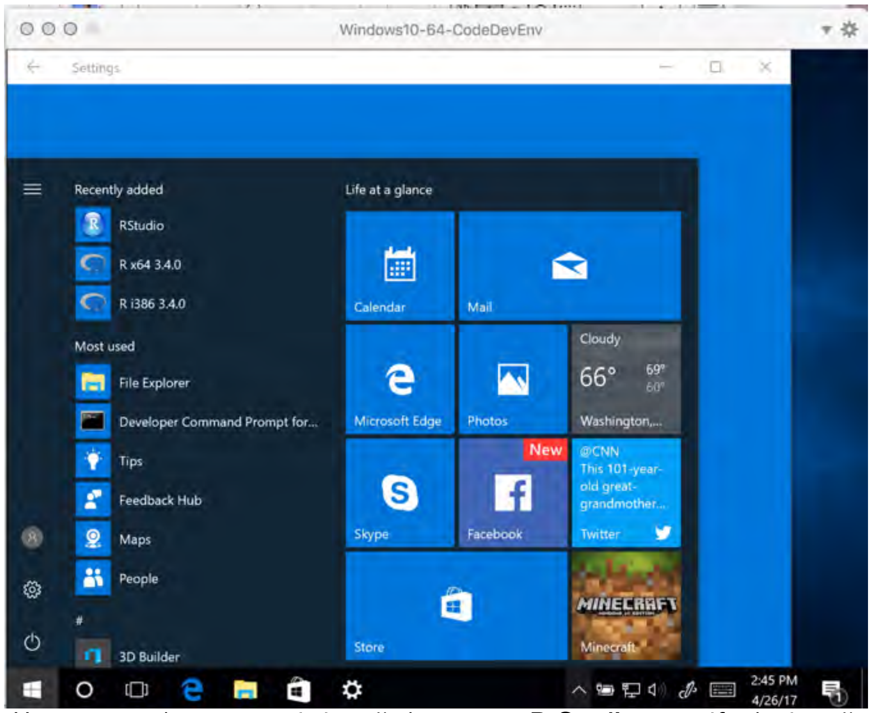
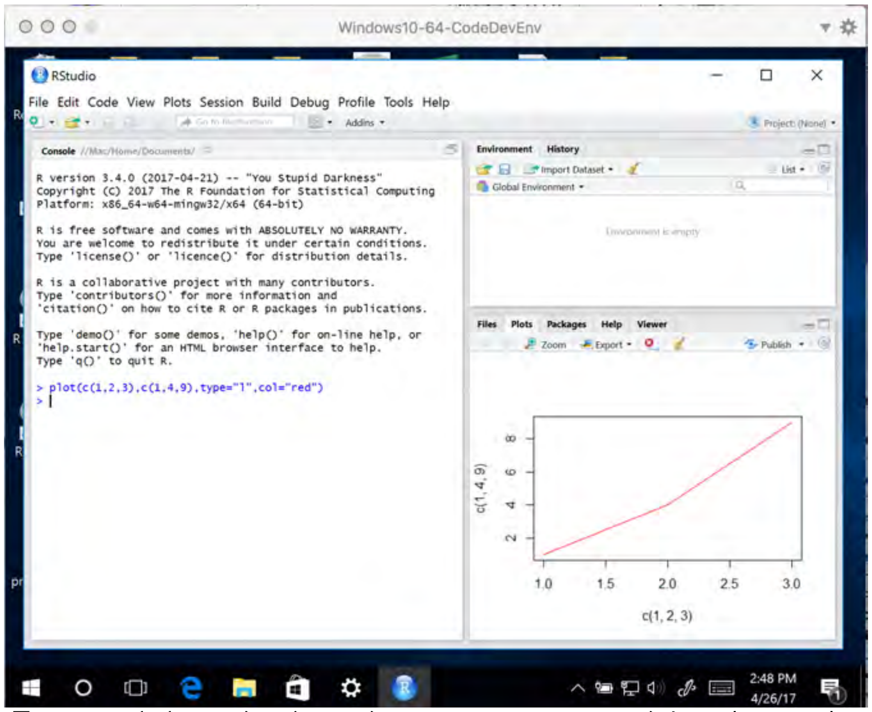


Figure 12. Here we see the program is installed, now run **R Studio** to verify the install.



```
R version 3.4.0 (2017-04-21) -- "You Stupid Darkness"
Copyright (C) 2017 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> plot(c(1,2,3),c(1,4,9),type="l",col="red")
> 
```

Figure 13. Type `plot(c(1,2,3),c(1,4,9),type="l",col="red")` into the console window. A plot should be generated as shown.

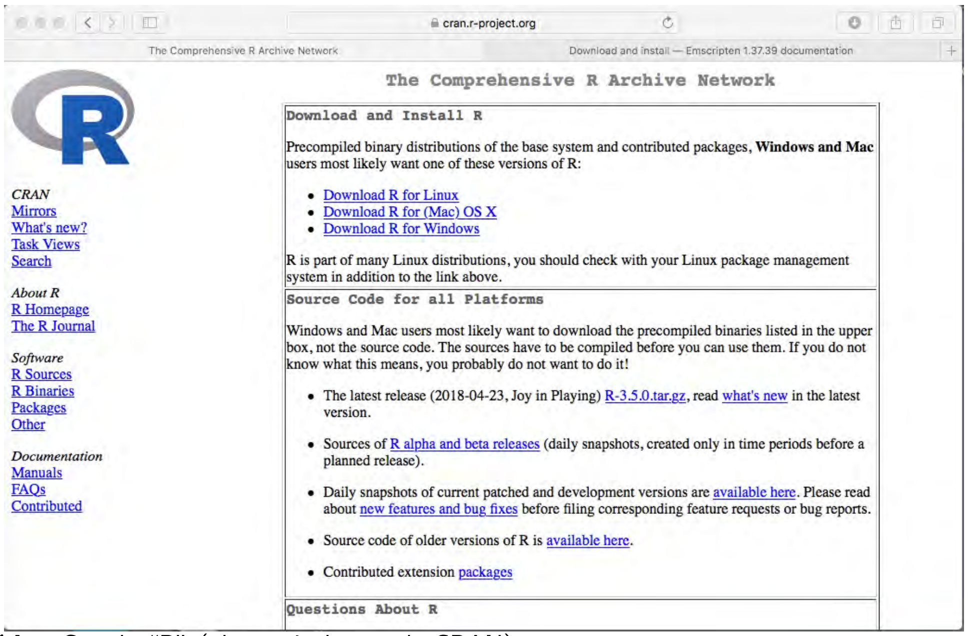
Yipee! It is running. You can install additional packages now or later. You should now have sufficient computation capability for the course.

1.2.2 Macintosh OSX Users

[Replicate Windows using MacOS screen captures] The purpose of this section is to demonstrate how to get **R** running on a Macintosh computer.[3] This document assumes the following:

1. You have internet connection.
2. You have sufficient user privileges to install software on your machine. (If it is your personal machine, the install may request your password, but should install.)
3. You have 60MB or so of vacant disk space on the system directory.

The step-by-step guide is presented as a series of screen captures. Obviously adjust inputs to fit your machine. The version in these screen captures is quite dated — use the most recent, stable version offered on CRAN (Comprehensive R Archive Network).[4]


cran.r-project.org
The Comprehensive R Archive Network
Download and install — Emscripten 1.37.39 documentation

The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

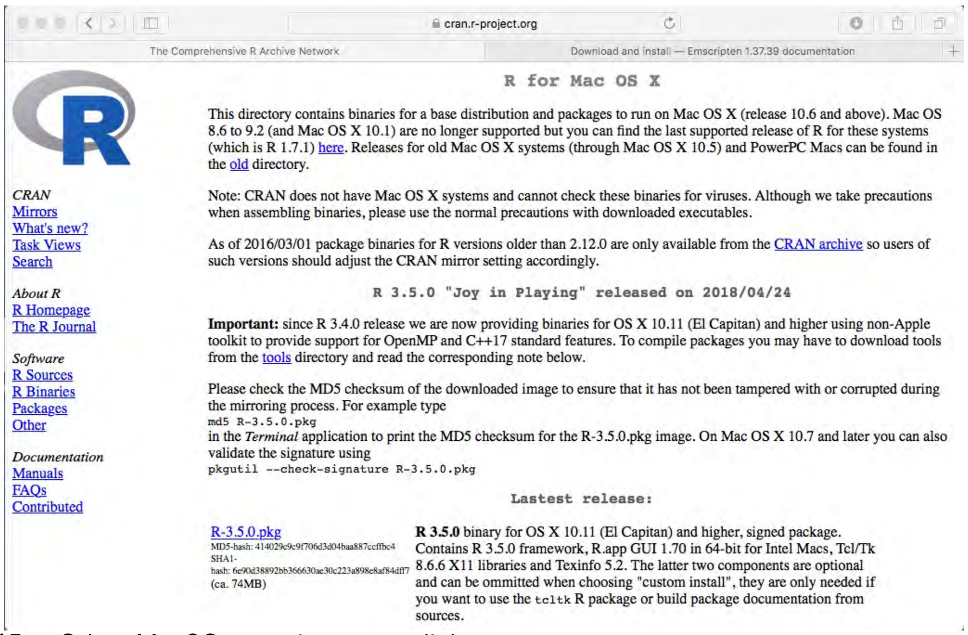
- The latest release (2018-04-23, Joy in Playing) R-3.5.0.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

Figure 14. Google "R" (alternatively google CRAN).

[3] I assume no-one will be using a PPC-Based Mac. If so, the CRAN does have PPC builds of R, but R Studio is not available; you would have to build from the source code.

[4] I have updated the screen captures for Windows 10 — so these should replicate the steps.



cran.r-project.org

The Comprehensive R Archive Network | Download and install — Emscripten 1.37.39 documentation

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only available from the CRAN archive so users of such versions should adjust the CRAN mirror setting accordingly.

R 3.5.0 "Joy in Playing" released on 2018/04/24

Important: since R 3.4.0 release we are now providing binaries for OS X 10.11 (El Capitan) and higher using non-Apple toolkit to provide support for OpenMP and C++17 standard features. To compile packages you may have to download tools from the tools directory and read the corresponding note below.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type

```
md5 R-3.5.0.pkg
```

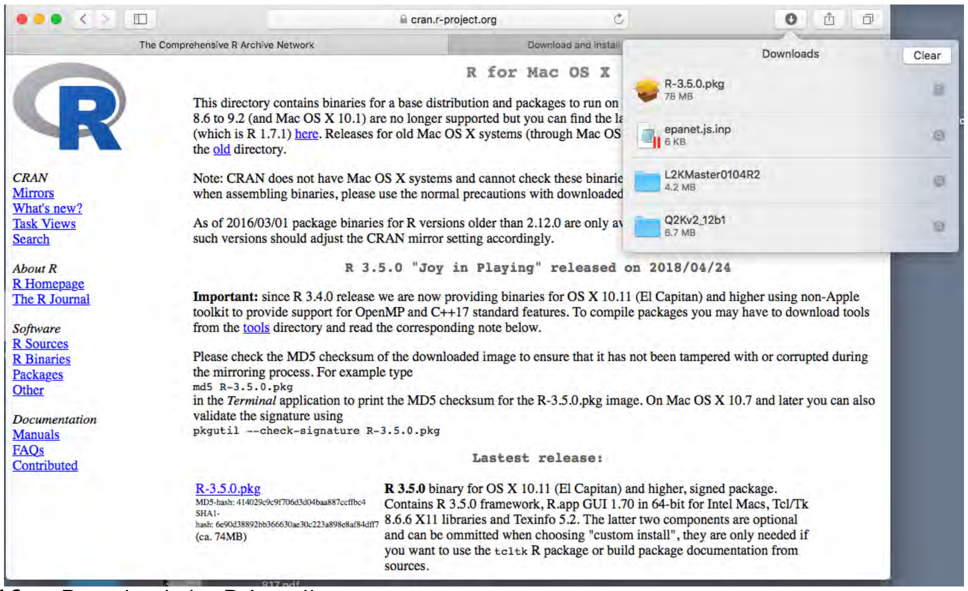
in the *Terminal* application to print the MD5 checksum for the R-3.5.0.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

```
pkgutil --check-signature R-3.5.0.pkg
```

Lastest release:

R-3.5.0.pkg MD5-hash: 414029c9c9f706d3d04baa887ccffbc4 SHA1-hash: 6e90d38892bb366630ae30c223a898e8af84dff7 (ca. 74MB)	**R 3.5.0** binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.0 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Figure 15. Select MacOS operating system link.



cran.r-project.org

The Comprehensive R Archive Network | Download and install

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on
8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the la
(which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS
the old directory.

Note: CRAN does not have Mac OS X systems and cannot check these binarie
when assembling binaries, please use the normal precautions with downloaded

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only av
such versions should adjust the CRAN mirror setting accordingly.

R 3.5.0 "Joy in Playing" released on 2018/04/24

Important: since R 3.4.0 release we are now providing binaries for OS X 10.11 (El Capitan) and higher using non-Apple toolkit to provide support for OpenMP and C++17 standard features. To compile packages you may have to download tools from the tools directory and read the corresponding note below.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type

```
md5 R-3.5.0.pkg
```

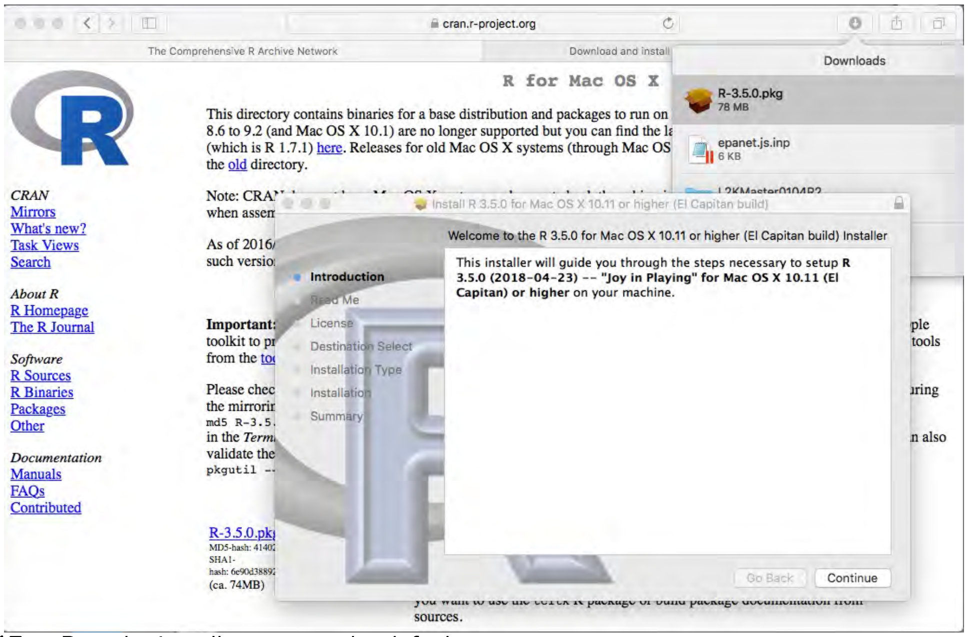
in the *Terminal* application to print the MD5 checksum for the R-3.5.0.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

```
pkgutil --check-signature R-3.5.0.pkg
```

Lastest release:

R-3.5.0.pkg MD5-hash: 414029c9c9f706d3d04baa887ccffbc4 SHA1-hash: 6e90d38892bb366630ae30c223a898e8af84dff7 (ca. 74MB)	**R 3.5.0** binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.0 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Figure 16. Download the R Installer.



R for Mac OS X

This directory contains binaries for a base distribution and packages to run on
8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the la
(which is R 1.7.1) here. Releases for old Mac OS X systems (through Mac OS
the old directory.

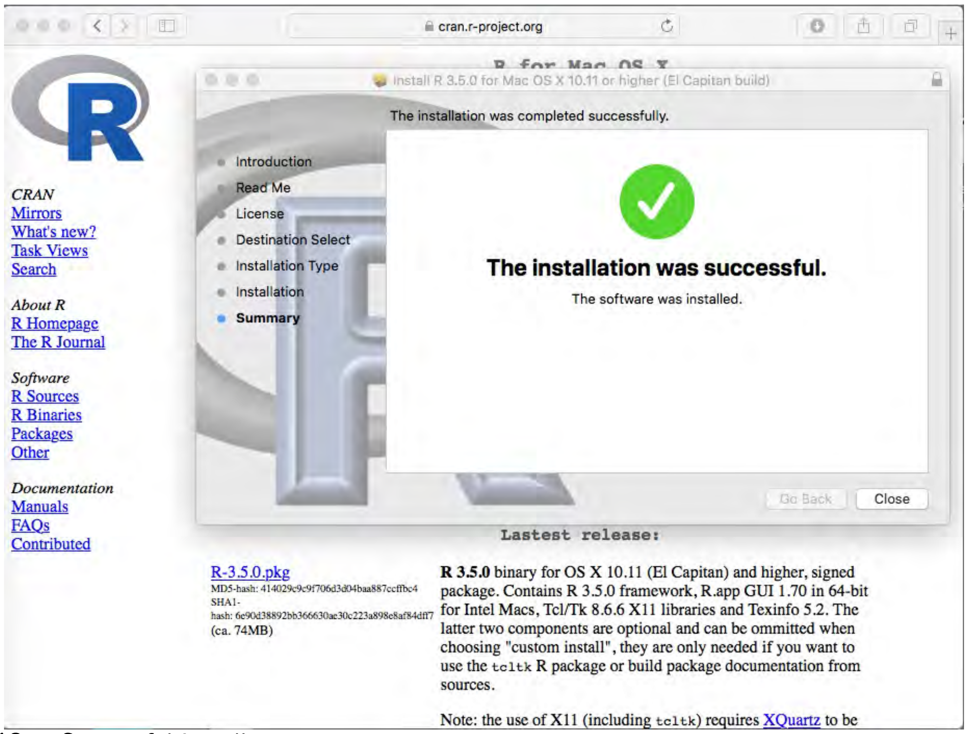
Install R 3.5.0 for Mac OS X 10.11 or higher (El Capitan build)

Welcome to the R 3.5.0 for Mac OS X 10.11 or higher (El Capitan build) Installer

This installer will guide you through the steps necessary to setup **R 3.5.0 (2018-04-23) -- "Joy in Playing" for Mac OS X 10.11 (El Capitan) or higher** on your machine.

R-3.5.0.pkg
(ca. 74MB)

Figure 17. Run the installer, accept the defaults.



Install R 3.5.0 for Mac OS X 10.11 or higher (El Capitan build)

The installation was completed successfully.

The installation was successful.

The software was installed.

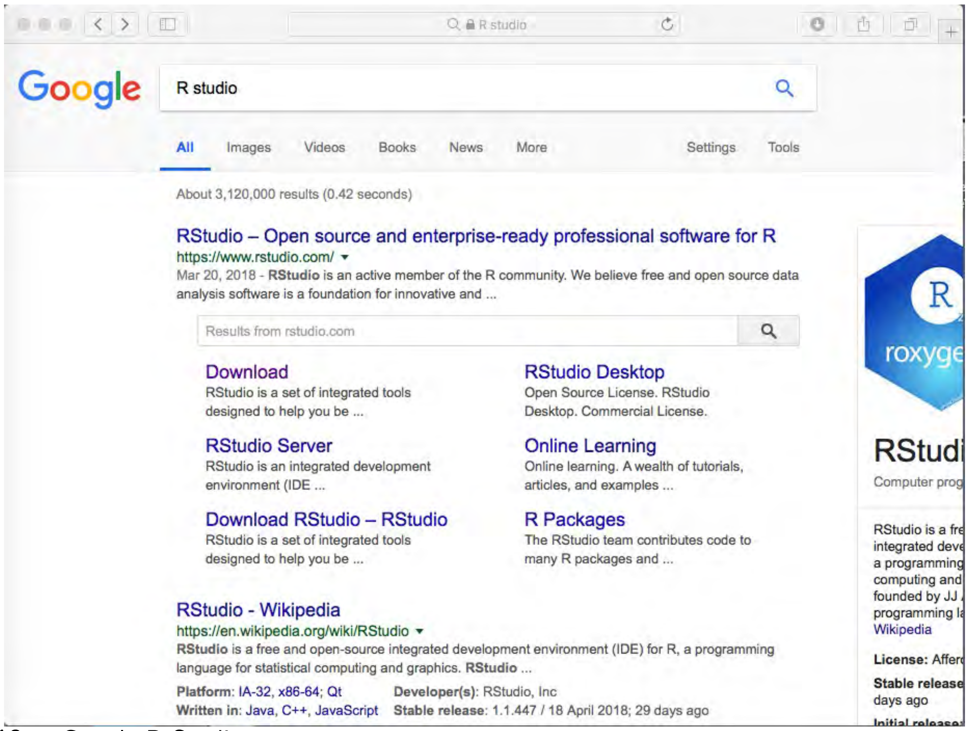
Lastest release:

R-3.5.0.pkg
MD5-hash: 414029c9c9f706d3d04baa887ccffbc4
SHA1-hash: 6e90d38892bb366630ae30c223a898e8af84dff7
(ca. 74MB)

R 3.5.0 binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.0 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be ommitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from sources.

Note: the use of X11 (including `tcltk`) requires XQuartz to be

Figure 18. Successful install.



About 3,120,000 results (0.42 seconds)

RStudio – Open source and enterprise-ready professional software for R
https://www.rstudio.com/
Mar 20, 2018 - RStudio is an active member of the R community. We believe free and open source data analysis software is a foundation for innovative and ...

Results from rstudio.com

Download
RStudio is a set of integrated tools designed to help you be ...

RStudio Desktop
Open Source License. RStudio Desktop. Commercial License.

RStudio Server
RStudio is an integrated development environment (IDE ...

Online Learning
Online learning. A wealth of tutorials, articles, and examples ...

Download RStudio – RStudio
RStudio is a set of integrated tools designed to help you be ...

R Packages
The RStudio team contributes code to many R packages and ...

RStudio - Wikipedia
https://en.wikipedia.org/wiki/RStudio
RStudio is a free and open-source integrated development environment (IDE) for R, a programming language for statistical computing and graphics. RStudio ...

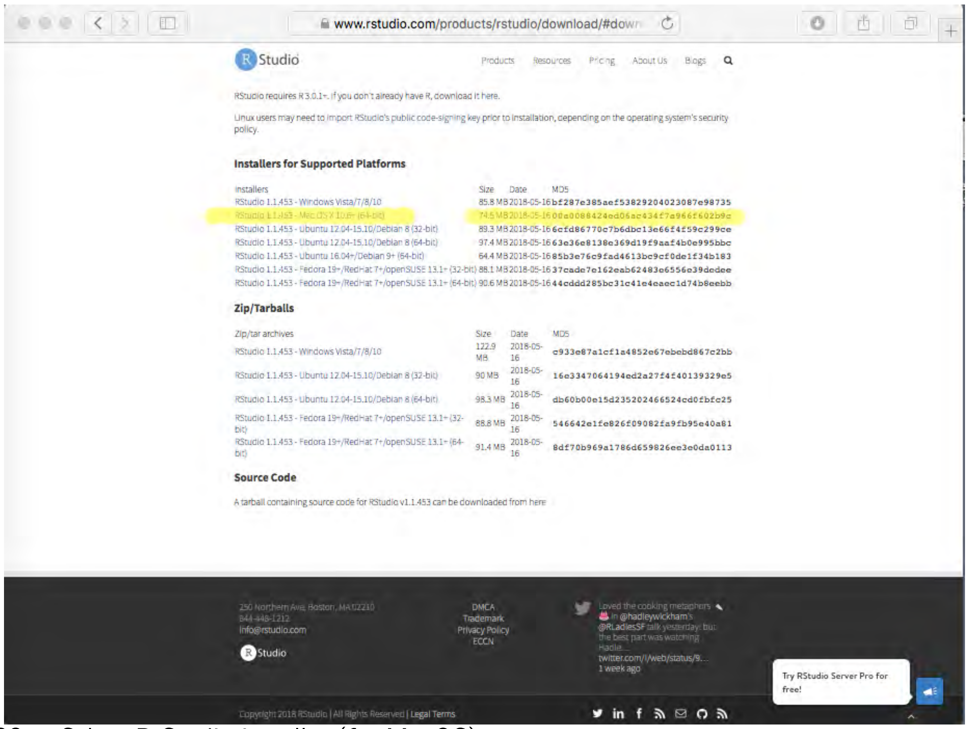
Platform: IA-32, x86-64; Qt
Written in: Java, C++, JavaScript
Developer(s): RStudio, Inc
Stable release: 1.1.447 / 18 April 2018; 29 days ago

RStudi
Computer prog

RStudio is a fre
integrated deve
a programming
computing and
founded by JJ
programming l
Wikipedia

License: Affer
Stable release
days ago

Figure 19. Google R Studio.



RStudio requires R 3.0.1+. If you don't already have R, download it here.

Linux users may need to import RStudio's public code-signing key prior to installation, depending on the operating system's security policy.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.453 - Windows Vista/7/8/10	85.8 MB	2018-05-16	bf287e385aef53829204023087e98735
RStudio 1.1.453 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-05-16	0da0088424ed06ac434f7a966f602b9c
RStudio 1.1.453 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB	2018-05-16	6cfd86770c7b6dbc13e66f4f59c299ce
RStudio 1.1.453 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-05-16	63e36e8138e369d19f9aaf4b0e995bbc
RStudio 1.1.453 - Ubuntu 16.04+/Debian 9+ (64-bit)	64.4 MB	2018-05-16	85b3e76c9fad4613bc9cf0de1f34b183
RStudio 1.1.453 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-05-16	37cade7e162eab62483e6556e39dedee
RStudio 1.1.453 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-05-16	44cddd285bc31c41e4eaec1d74b8eebb

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
RStudio 1.1.453 - Windows Vista/7/8/10	122.9 MB	2018-05-16	c933e87a1cf1a4852e67ebebd867c2bb
RStudio 1.1.453 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	90 MB	2018-05-16	16e3347064194ed2a27f4f40139329e5
RStudio 1.1.453 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	98.3 MB	2018-05-16	db60b00e15d235202466524cd0fbfc25
RStudio 1.1.453 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.8 MB	2018-05-16	546642e1fe826f09082fa9fb95e40a81
RStudio 1.1.453 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	91.4 MB	2018-05-16	8df70b969a1786d659826ee3e0da0113

Source Code

A tarball containing source code for RStudio v1.1.453 can be downloaded from here

250 Northern Ave, Boston, MA 02210
844-448-1212
info@rstudio.com

DMCA
Trademark
Privacy Policy
ECCN

Loved the cooking metaphors in @hadleywickham's @RLadiesSF talk yesterday: but the best part was watching Hadle... twitter.com/i/web/status/9...
1 week ago

Try RStudio Server Pro for free!

Copyright 2018 RStudio | All Rights Reserved | Legal Terms

Figure 20. Select R Studio installer (for MacOS).

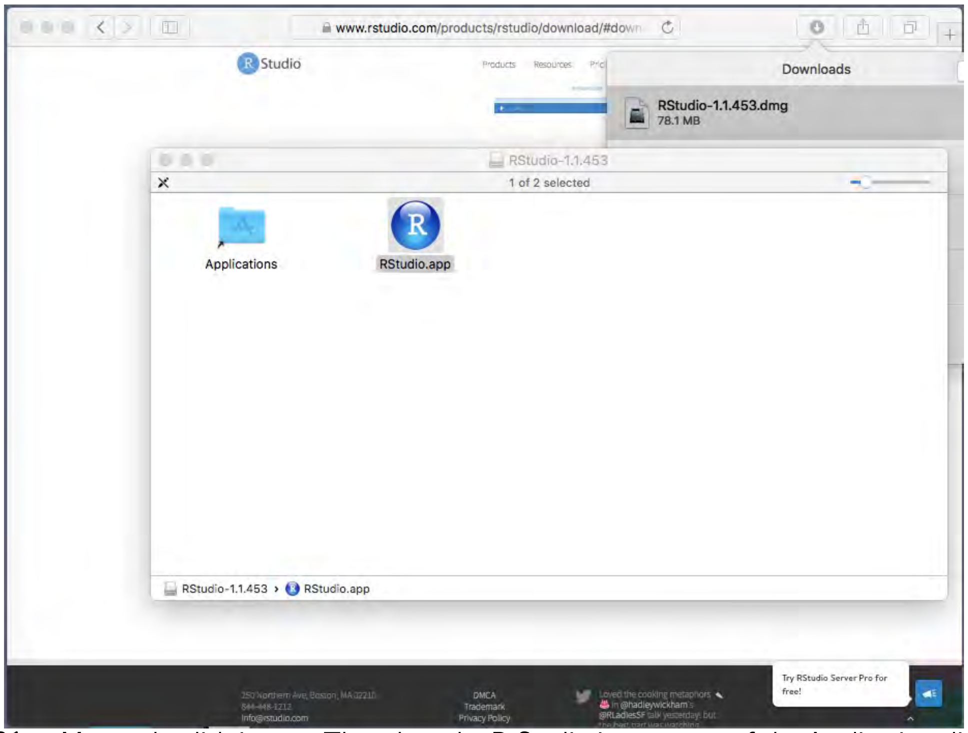

www.rstudio.com/products/rstudio/download/#down
Studio
Downloads
RStudio-1.1.453.dmg
78.1 MB
RStudio-1.1.453
1 of 2 selected
Applications
RStudio.app
RStudio-1.1.453 › RStudio.app
Try RStudio Server Pro for free!

Figure 21. Mount the disk image. Then drag the R Studio icon on top of the Applications link – this action will install the program.

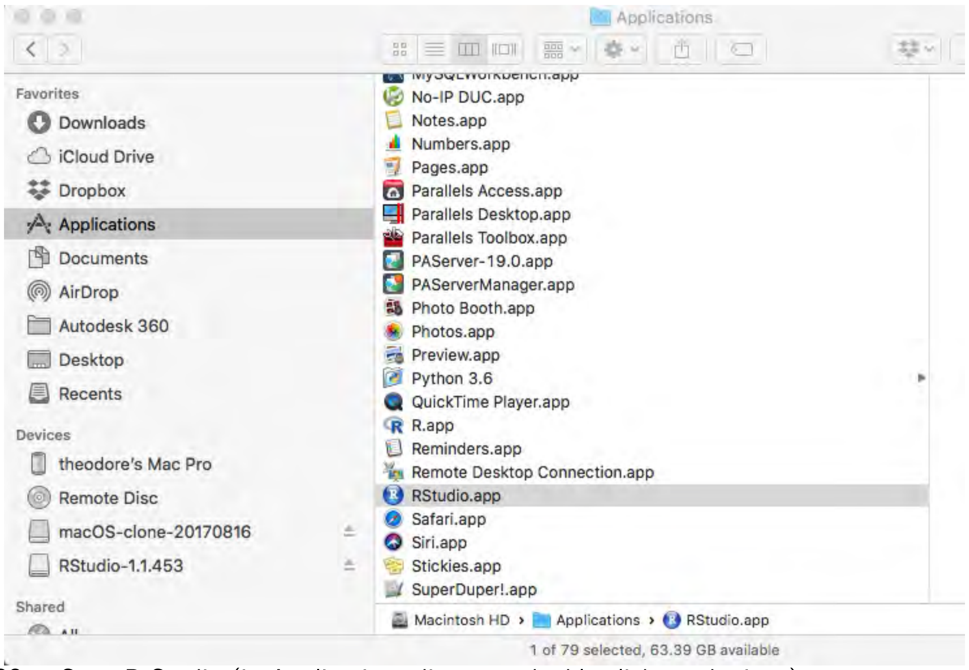
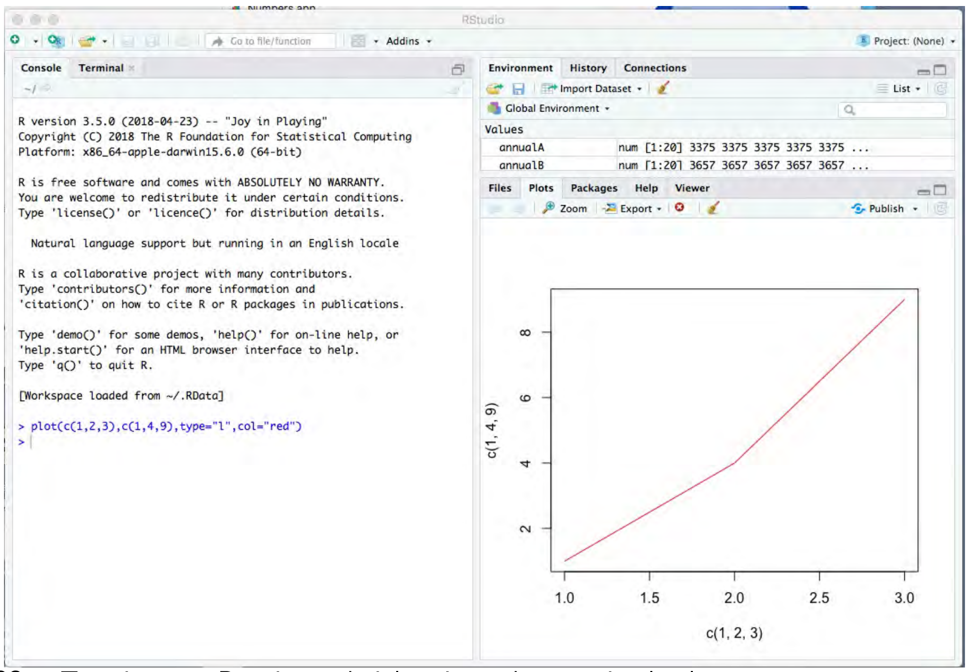


Figure 22. Start R Studio (in Applications directory, double click on the icon).



```
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> plot(c(1,2,3),c(1,4,9),type="l",col="red")
>
```

```
Values
  annualA        num [1:20] 3375 3375 3375 3375 3375 ...
  annualB        num [1:20] 3657 3657 3657 3657 3657 ...
```

Figure 23. Type in some R script and viola – it produces a simple plot.

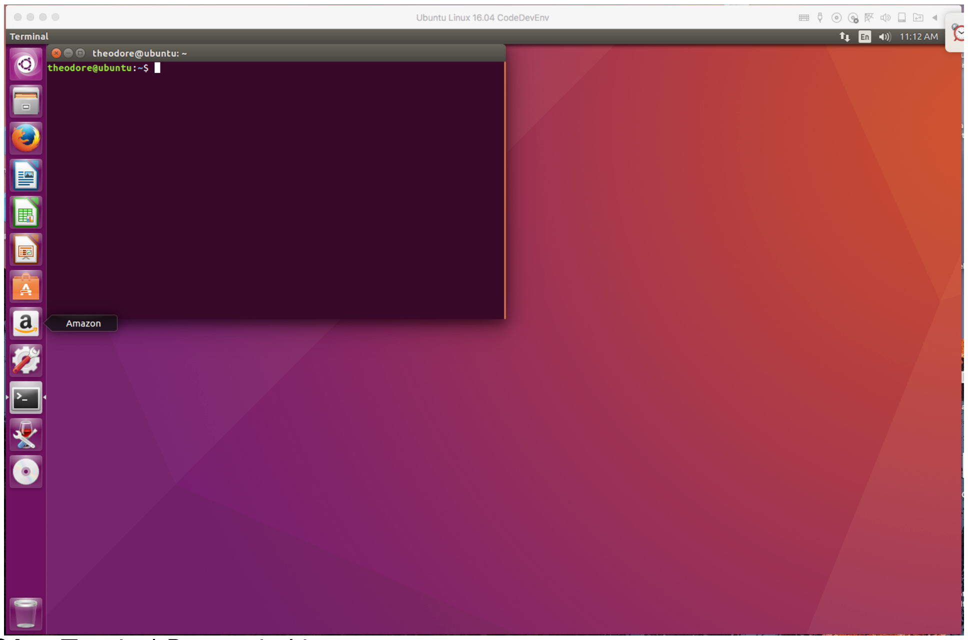
Yipee! It is running. You can install additional packages now or later. You should now have sufficient computation capability for the course.

1.2.3 Linux Users

The purpose of this section is to demonstrate how to get **R** and **R Studio** running on a Linux computer. This document assumes the following:

1. You have internet connection.
2. You have sufficient user privileges to install software on your machine. Generally, if it is your own machine then you have superuser (root) privileges. If it is some network machine maintained by someone else you probably don't have such priviliges. The examples here use the `sudo <command>` to do the installs – on my machine the password I enter is my user password (and not the root password). Alternativley you can switch user `su` to root, and run the installs as root – this approach is considerably more dangerous in terms of wrecking your operating system that using the `sudo` approach.
3. You have 60MB or so of vacant disk space on the system directory.

The step-by-step guide is presented as a series of screen captures. Obviously adjust inputs to fit your machine. Use the most recent, stable version offered on CRAN (Comprehensive R Archive Network).

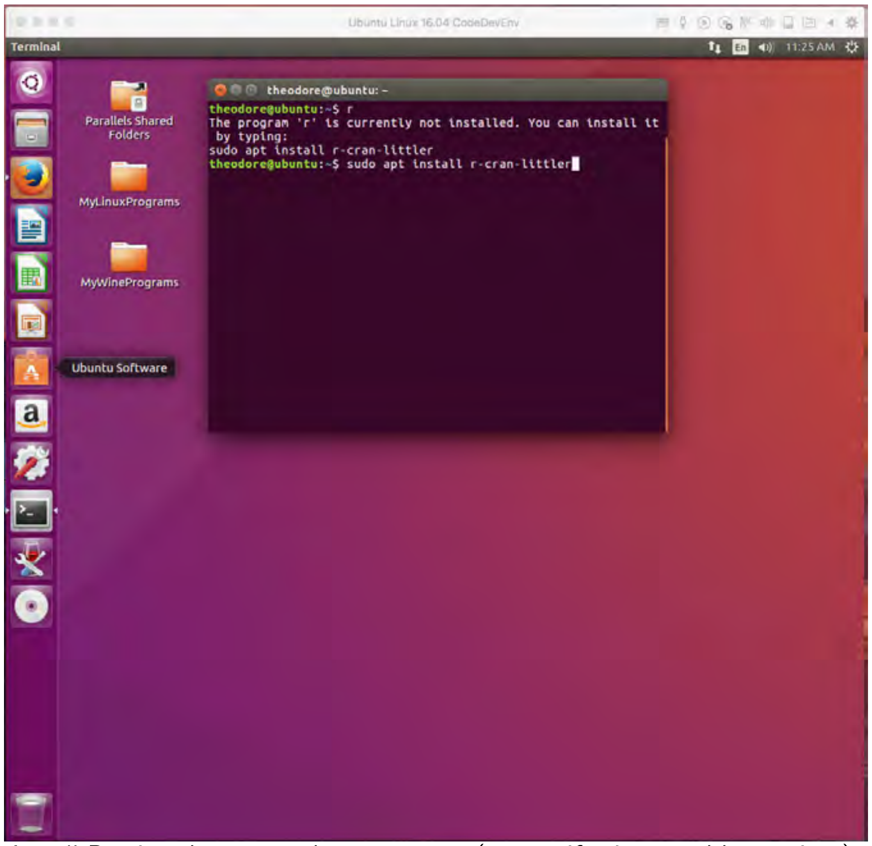


```
theodore@ubuntu:~$
```

Figure 24. Terminal Prompt in Linux.

To get started we need to install **R**. The easiest method is to use the package manager – my Linux distribution is Ubuntu 16.XX. It is built from the debian distribution and uses the `apt` package manager. The package manager is pretty cool because when we request a package it finds the package and its dependencies, downloads everything needed and then we can install. In earlier times (the 1990's) using the Red Hat Package Manager (`rpm`) one would have to find the dependencies themselves (in all fairness `rpm` would identify the dependencies and suggest where to find them!). So here we go, first open a terminal in Linux.

Next in the terminal window issue the command `sudo apt install r-cran-littler`.



```
theodore@ubuntu:~$ r
The program 'r' is currently not installed. You can install it
 by typing:
sudo apt install r-cran-littler
theodore@ubuntu:~$ sudo apt install r-cran-littler
```

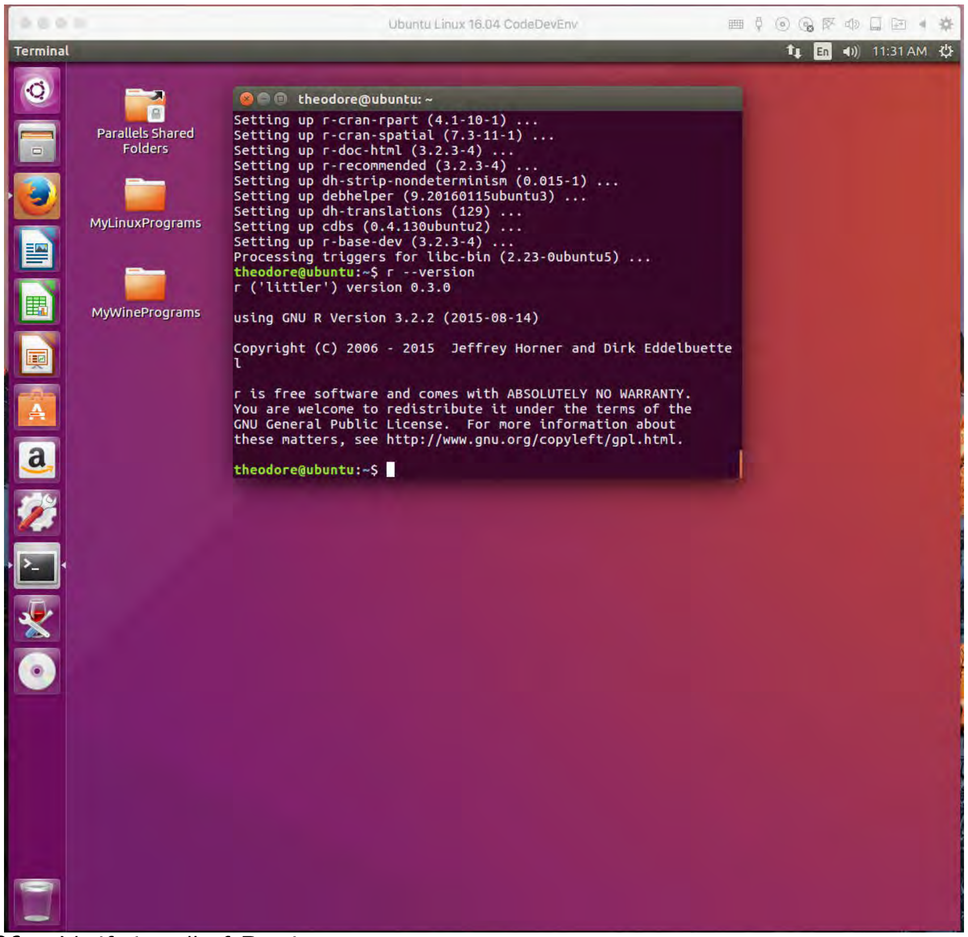
Figure 25. Install R using the apt package manager (or `rpm` if using a red hat variant).

When you press return, the computer should ask for a password – use your user password; if you have install privileges this will work. If not, you will have to switch user to root and either add your user account to the group `wheel`, return to your account – or just install as root.

Next the install will begin and may take a few minutes. Usually there is a lot of installer messages that run across the screen (kind of like in "The Matrix"). The `apt` utility is downloading files and binging them to resources on the system so that **R** can run. Eventually it should get to the end of the install and may look something like the next screen capture.

Our next task is to verify that the install was successful. Usually failure is obvious, but not always. I find the easiest way to verify that the operating system thinks the software is installed is issue the command to run the program with the version switch active. In this case, issue the command `r --version`. This command will try to run **R**, and will return the version number (or build number).

In the next screen capture when we run the command we see that the version number is **R** version 3.2.2. The version numbering system is typical for all software – it identifies something like this is the second subversion, of the second revision, of the third stable release.



```
Setting up r-cran-rpart (4.1-10-1) ...
Setting up r-cran-spatial (7.3-11-1) ...
Setting up r-doc-html (3.2.3-4) ...
Setting up r-recommended (3.2.3-4) ...
Setting up dh-strip-nondeterminism (0.015-1) ...
Setting up debhelper (9.20160115ubuntu3) ...
Setting up dh-translations (129) ...
Setting up cdbs (0.4.130ubuntu2) ...
Setting up r-base-dev (3.2.3-4) ...
Processing triggers for libc-bin (2.23-0ubuntu5) ...
theodore@ubuntu:~$ r --version
r ('littler') version 0.3.0

using GNU R Version 3.2.2 (2015-08-14)

Copyright (C) 2006 - 2015  Jeffrey Horner and Dirk Eddelbuette
l

r is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under the terms of the
GNU General Public License.  For more information about
these matters, see http://www.gnu.org/copyleft/gpl.html.

theodore@ubuntu:~$ 
```

Figure 26. Verify install of **R** using `r --version`.

The next step I recommend is to go ahead and download a development environment. **R Studio** is an integrated development environment for **R**. We don't "need" it, but it enforces some discipline, gives us a place to store and modify **R**-scripts (little programs), and lets us see all of our work going on in one location.

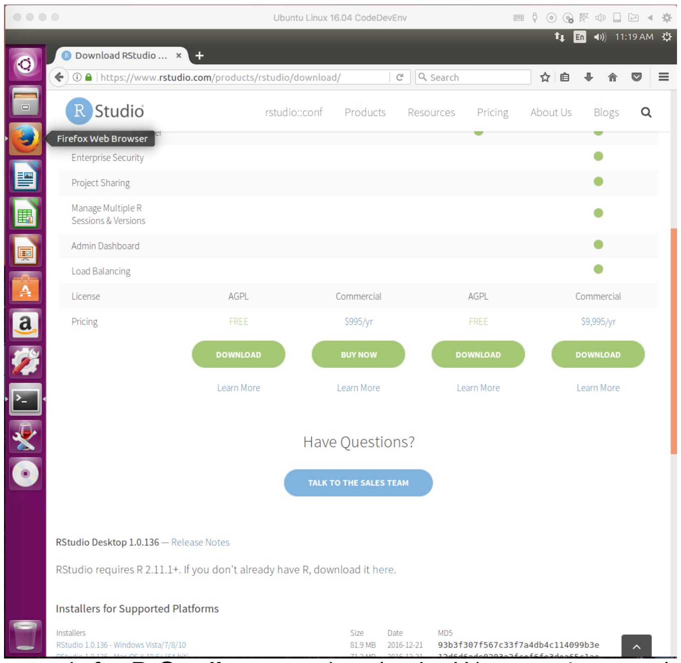
To get **R Studio** we have to download it from the manufacturer, the copy we will use is free. Instead of `apt` this time I used Firefox to navigate to the **R Studio** website, then select download.

Next select the appropriate installer for your platform. Be sure you are selecting an installer and not the source codes for the program.[5]

We will download the 64-bit version (unless you have a 32-bit machine, then you need

[5]In theory we could build the program from source using the make utility and (hopefully) already installed compilers – this is for people with time, training, inclination and need. We are going to use the already built binaries!

32-bit software). We need to select our Linux platform – mine is Unbuntu/Debian. I also have a Red Hat/SuSe machine, so if I were using that machine, i would select its software.


Ubuntu Linux 16.04 CodeDevEnv

Download RStudio ...

https://www.rstudio.com/products/rstudio/download/

Enterprise Security				●
Project Sharing				●
Manage Multiple R Sessions & Versions				●
Admin Dashboard				●
Load Balancing				●
License	AGPL	Commercial	AGPL	Commercial
Pricing	FREE	$995/yr	FREE	$9,995/yr
	DOWNLOAD	BUY NOW	DOWNLOAD	DOWNLOAD
	Learn More	Learn More	Learn More	Learn More

Have Questions?

TALK TO THE SALES TEAM

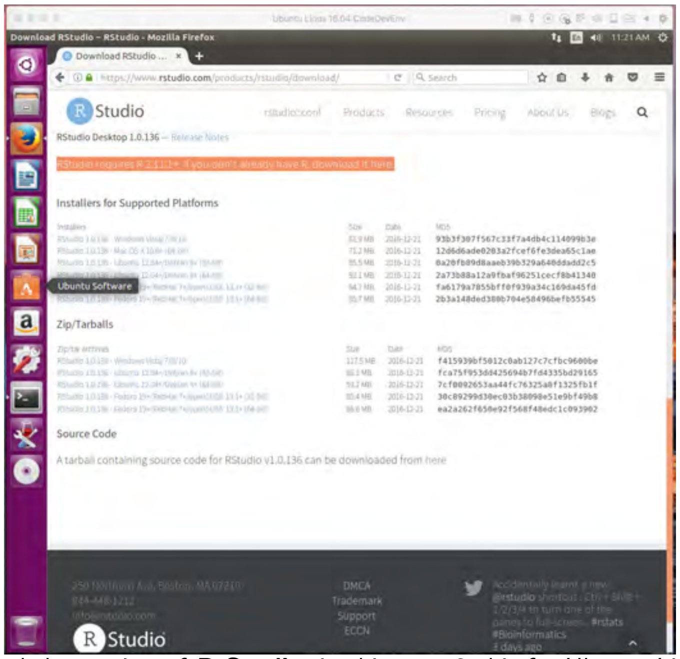
RStudio Desktop 1.0.136 — Release Notes

RStudio requires R 2.11.1+. If you don't already have R, download it here.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.0.136 - Windows Vista/7/8/10	81.9 MB	2016-12-21	93b3f307f567c33f7a4db4c114099b3e

Figure 27. Browser search for **R Studio**; go to downloads. We are going to select the free (leftmost) column.


Download RStudio – RStudio - Mozilla Firefox

https://www.rstudio.com/products/rstudio/download/

RStudio Desktop 1.0.136 — Release Notes

RStudio requires R 2.11.1+. If you don't already have R, download it here.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.0.136 - Windows Vista/7/8/10	81.9 MB	2016-12-21	93b3f307f567c33f7a4db4c114099b3e
RStudio 1.0.136 - Mac OS X 10.6+ (64-bit)	71.2 MB	2016-12-21	12d6d6ade0203a2fcef6fe3dea65c1ae
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (32-bit)	85.5 MB	2016-12-21	0a20fb09d8aaeb39b329a640ddadd2c5
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (64-bit)	92.1 MB	2016-12-21	2a73b88a12a9fbaf96251cecf8b41340
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	84.7 MB	2016-12-21	fa6179a7855bff0f939a34c169da45fd
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	85.7 MB	2016-12-21	2b3a148ded380b704e58496befb55545

Zip/Tarballs

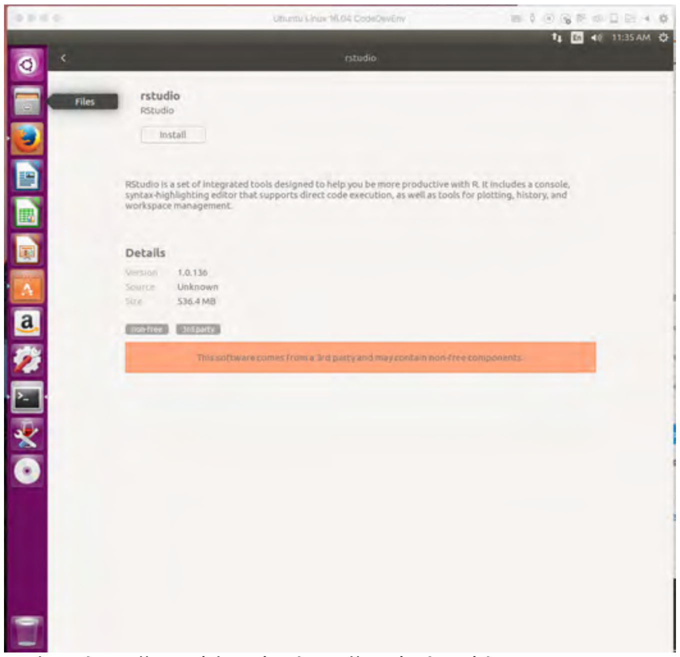
Zip/tar archives	Size	Date	MD5
RStudio 1.0.136 - Windows Vista/7/8/10	117.5 MB	2016-12-21	f415939bf5012c0ab127c7cfbc9600be
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (32-bit)	86.1 MB	2016-12-21	fca75f953dd425694b7fd4335bd29165
RStudio 1.0.136 - Ubuntu 12.04+/Debian 8+ (64-bit)	93.2 MB	2016-12-21	7cf0092653aa44fc76325a0f1325fb1f
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	85.4 MB	2016-12-21	30c89299d30ec03b38098e51e9bf49b8
RStudio 1.0.136 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	88.6 MB	2016-12-21	ea2a262f650e92f568f48edc1c093902

Source Code

A tarball containing source code for RStudio v1.0.136 can be downloaded from here

Figure 28. Download the version of **R Studio**; in this case 64-bit for Ubuntu Linux (what I am using). My computer asks if upon download if I want to run the installer, of course select YES.

Figure 28 is the selection page for selecting the installers. Once I select the package the computer starts the download, and asks me if I want to run the installer, as in Figure 29



rstudio
RStudio

RStudio is a set of integrated tools designed to help you be more productive with R. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, and workspace management.

Details

Version	1.0.136
Source	Unknown
Size	536.4 MB

This software comes from a 3rd party and may contain non-free components.

Figure 29. Here we select install, and let the installer do its thing.

Selecting yes, the installer will attempt to install **R Studio** onto my computer. Once installation is complete, the program is ready for a validation (of install) run.

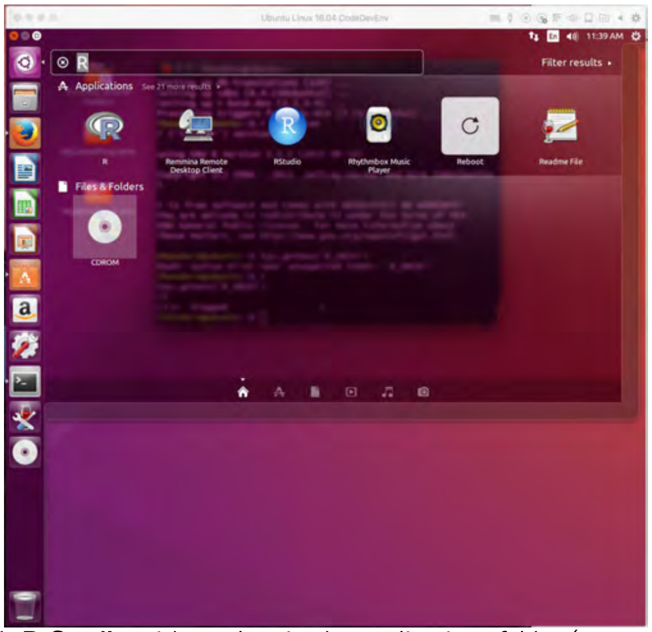
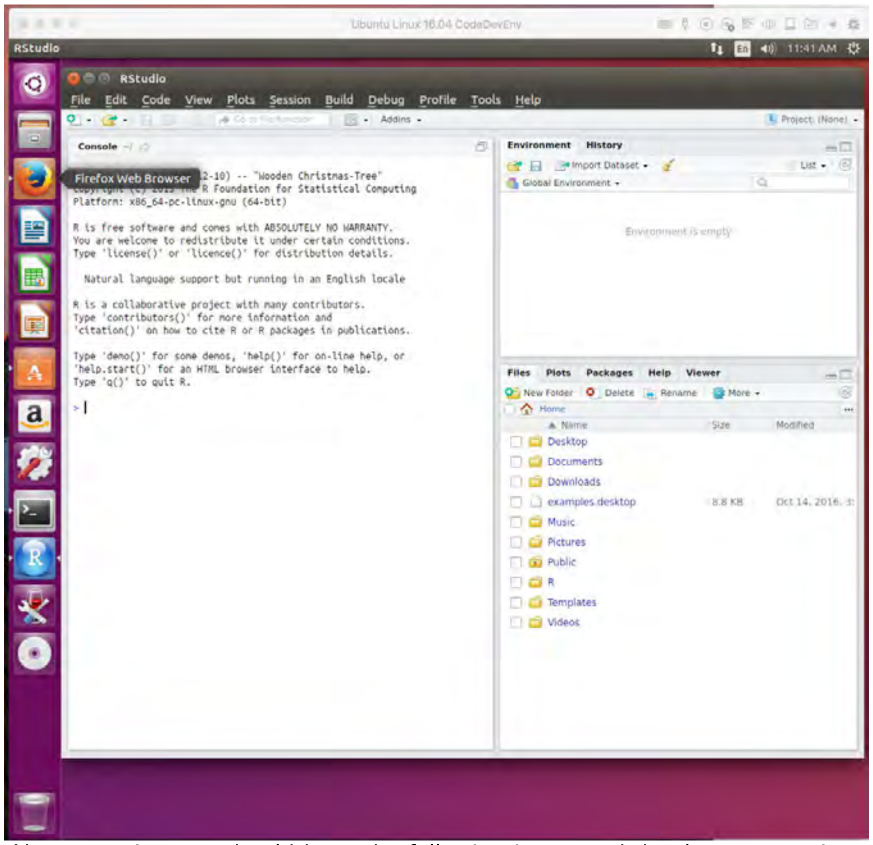


Figure 30. To launch **R Studio**, either select in the applications folder (or type in the terminal `rstudio`.

Figure 30 is a screen capture after installation using the Unbuntu program manager window. Both **R** and **R Studio** are available.

Either select **R Studio** to launch it, or type in a terminal window `rstudio` and the IDE should launch. The IDE itself is a bit complicated, but actually enables us to keep better track of our work and ultimately saves time. MatLab users should notice that the interface looks quite similar (at least it does to me) – its also the same concept.

Figure 31 is the result of launching the program. The left side of the IDE is an **R** console – exactly what would occur if we had just started **R**. The right side of the IDE is divided into an upper and lower panel. Each panel provides information about our programming environment at any instant – and the content is selectable from the icons at the top of each panel.



```
2-10) -- "Wooden Christmas-Tree"
R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> |
```

Figure 31. Upon opening you should have the following integrated development environment (IDE). The left panel is exactly what you would obtain if you just type `r` in the terminal window. .

The next few figures are a step-by-step example to introduce **R Studio** as well as test if it installed correctly. We could simply type in the **R** console within the IDE, but instead to get into the habit of saving our work, we will open a new scripting file.

Figure 32 shows the process of selecting FILE/OPEN to create a new file to store our scripts. We will type a few commands into the file and then run them.

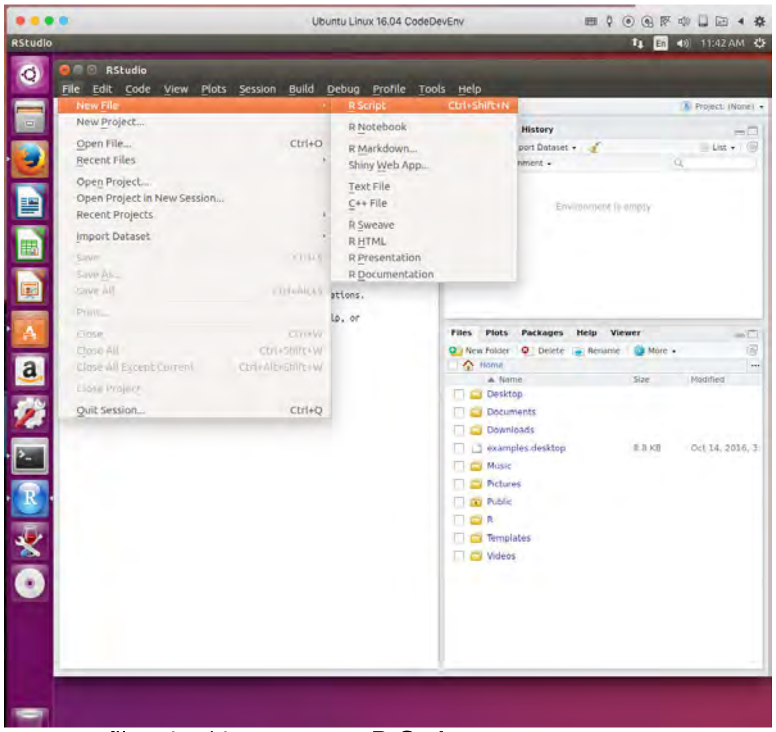
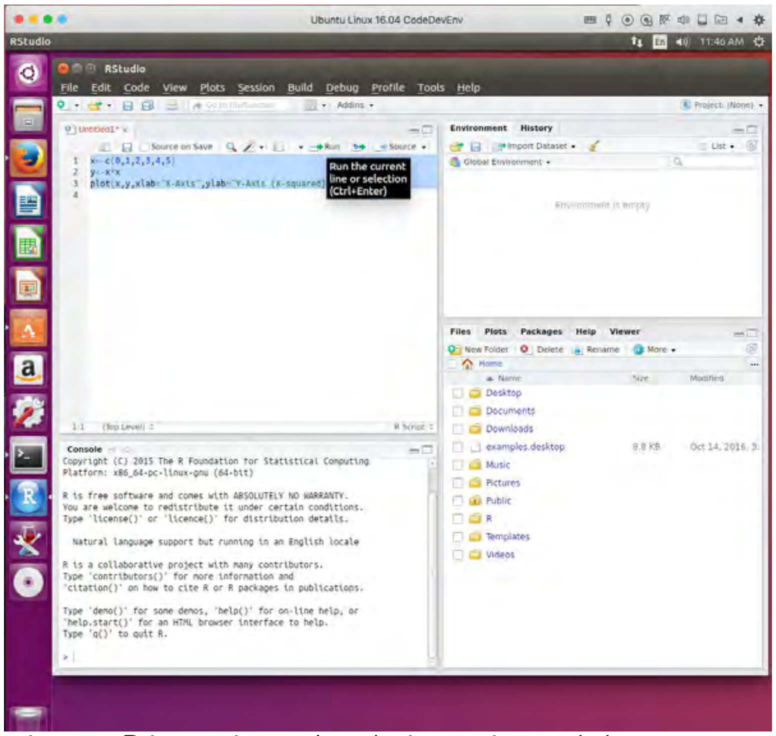


Figure 32. Open a new file – in this case as an **R-Script**.



```
x<-c(0,1,2,3,4,5)
y<-x*x
plot(x,y,xlab="X-Axis",ylab="Y-Axis (X-squared)
```

```
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Figure 33. Type in some **R** instructions, select the instructions and choose RUN.

So once the file is open the left side of the IDE divides into two panels, an upper and lower panel. The lower panel is the console environment, and the upper panel is a script editor.

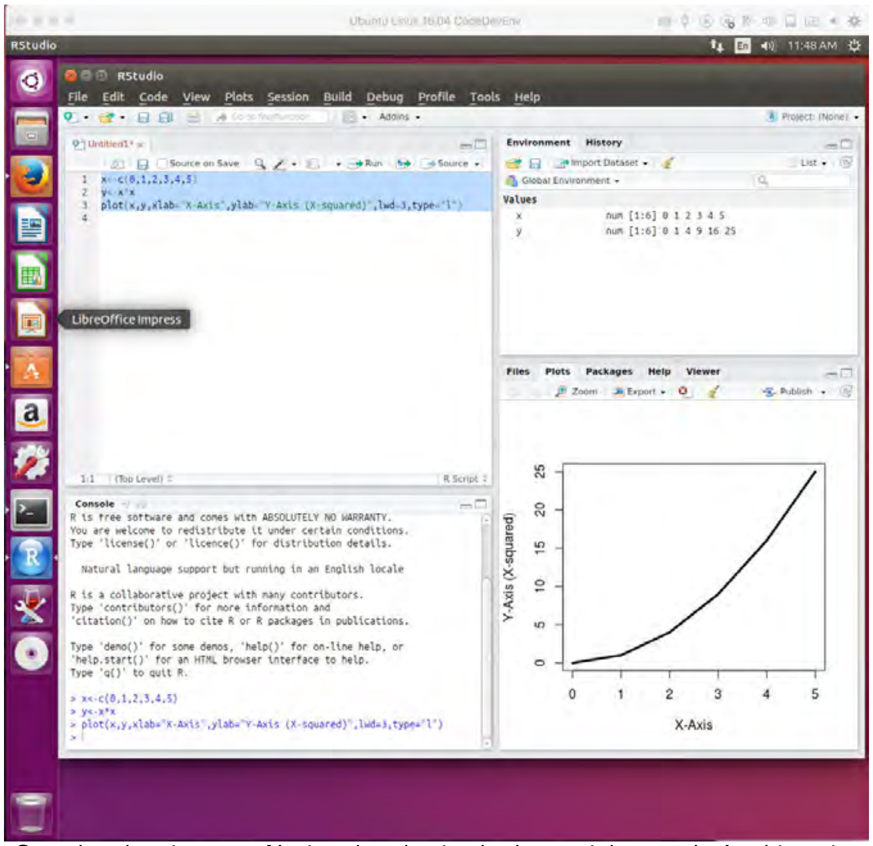
Figure 33 shows the upper panel with the **R** commands shown in Listing 1

Listing 1. R code demonstrating a few commands
This fragment of code generates two vectors X and Y and then plots them.

```
############### Some R Commands #######################
x <- c(0,1,2,3,4,5) # create a vector of 6 elements -- integers 0 to 5.
y <- x*x            # square x, put result in y.
plot(x,y,xlab="X_Axis",ylab="Y-Axis (X-squared)",lwd=3, type="l") # make and label a plot
```

To actually run the code we can either highlight the portion of the script file we wish the program to execute (that's what done here), or we can save the file and run the entire file. Often we will do both – highlight portions to develop a model, then save and run the file as needed. The term "sourcing" a [file] in R is jargon for running the commands contained within a file named [file]. The ability to save and reuse commands is really useful and is what makes **R** (or any other stored program software) really useful.

Figure 34 is the result of highlighting these three lines of code and running them (notice the little run icon above the script editor).



```
x<-c(0,1,2,3,4,5)
y<-x*x
plot(x,y,xlab="X-Axis",ylab="Y-Axis (X-squared)",lwd=3,type="l")
```

```
Values
  x                 num [1:6] 0 1 2 3 4 5
  y                 num [1:6] 0 1 4 9 16 25
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> x<-c(0,1,2,3,4,5)
> y<-x*x
> plot(x,y,xlab="X-Axis",ylab="Y-Axis (X-squared)",lwd=3,type="l")
> 
```

Figure 34. Completed script run. Notice the plot in the lower right panel. At this point you have a functioning programming tool.

2 Algorithms and R for Computing

An algorithm is a recipe. A useful definition is

> An algorithm is a computational method or an ensemble of rules determining the order and form of numerical operations to be applied to a set of data $\mathbf{a}(a_1, a_2, \dots)$ in order to find a new set of values $\mathbf{x}(x_1, x_2, \dots)$ forming the solution of a problem.
>
> An algorithmic procedure can be represented as
>
> $$\mathbf{x} = f(\mathbf{a}) \tag{1}$$
>
> From a mathematical perspective the main concern is that the algorithm is well posed:
>
> 1. A solution exists for a given $\mathbf{a}$.
> 2. The computation must lead to a single solution for $\mathbf{x}$ given $\mathbf{a}$.
> 3. The results for $\mathbf{x}$ must be connected to the input $\mathbf{a}$ through the Lipschitz relation.
>
> $$|\delta \mathbf{a}| < \eta \text{ then } |\delta \mathbf{x}| < M|\delta \mathbf{a}| \tag{2}$$
>
>   where M is a bounded natural number, $M = M(\mathbf{a}, \eta)$[6].
>
> Certain common problems are not well posed as stated but with reasonable assumptions can be forced into such a state.

Thus an algorithm is a recipe to take input data and produce output responses through some relationships. If a well posed problem then each result is related to the inputs, and the same inputs (in an algorithm) produce the same results. By the recipe analogy, if you follow the same recipe each time with the same raw materials then the cake should taste the same when it is baked.

An important concept is that an algorithm operates on data (procedure-oriented); an object-oriented view is that an algorithm performs a task (generate response) based on states established by the data. Both points of view are valid and equivalent.

Most computational hydraulics models are built (by a quirk of history) in a procedure-oriented perspective.

2.1 Tools

A practicing modeler needs a toolkit — these tools range from the actual computation engine (EPA-SWMM, HEC-RAS, FESWMS, HSPF, WSPRO, TR-20, etc.)

[6]Think of M as a mapping function.

to analysis tools for result interpretation (**R**, **Excel**) to actual programming tools (`FORTRAN`,`PERL`, etc.) to construct their own special purpose models or to test results from general purpose professional models.

In this book **R** will be used for programming, analysis, and presentation.

2.2 Programming

Why programming?

There are three fundamental reasons for requiring a programming experience:

1. Teaching someone else a subject or procedure forces the teacher to have a reasonable understanding of the subject or procedure. Teaching a computer (by virtue of programming) forces a very deep understanding of the underlying algorithm.
2. You will encounter situations that general purpose programs are not designed to address; if you have a moderate ability to build your own tools when you need to, then you can. In all likliehood, you will "trick" the professional program, but you cannot invent tricks unless you know a little bit about programming.
3. Programming a computer requires an algorithmic thought process — this process is valuable in many other areas of engineering, hence the act of programming is good discipline for other problems you will encounter.

2.3 Interpolating Tabular Data – A useful algorithm

Material properties in physical systems are usually tabulated values. A frequent task is to interpolate in a set of tabular values to approximate the value between rows (or columns) in the table. Linear interpolation is the common technique used; and the tables can are stores as either separate files, or, if the tables are small enough, they can be directly imbedded into the code.

2.4 Linear Interpolation

Figure 35 is a sketch of a set of ordered pairs (x, y).

These pairs (there are two in the sketch) represent values in a table, for instance x may represent water temperature, and y may represent vapor pressure at that particular temperature. Two adjacent values (in the table) are depicted in the sketch, and the pairs are ordered bases on the x-value.

Now suppose we want to estimate the value of y^* at some intermediate value x^* that lies between x_1 and x_2. As a computational task, the problem statement is to

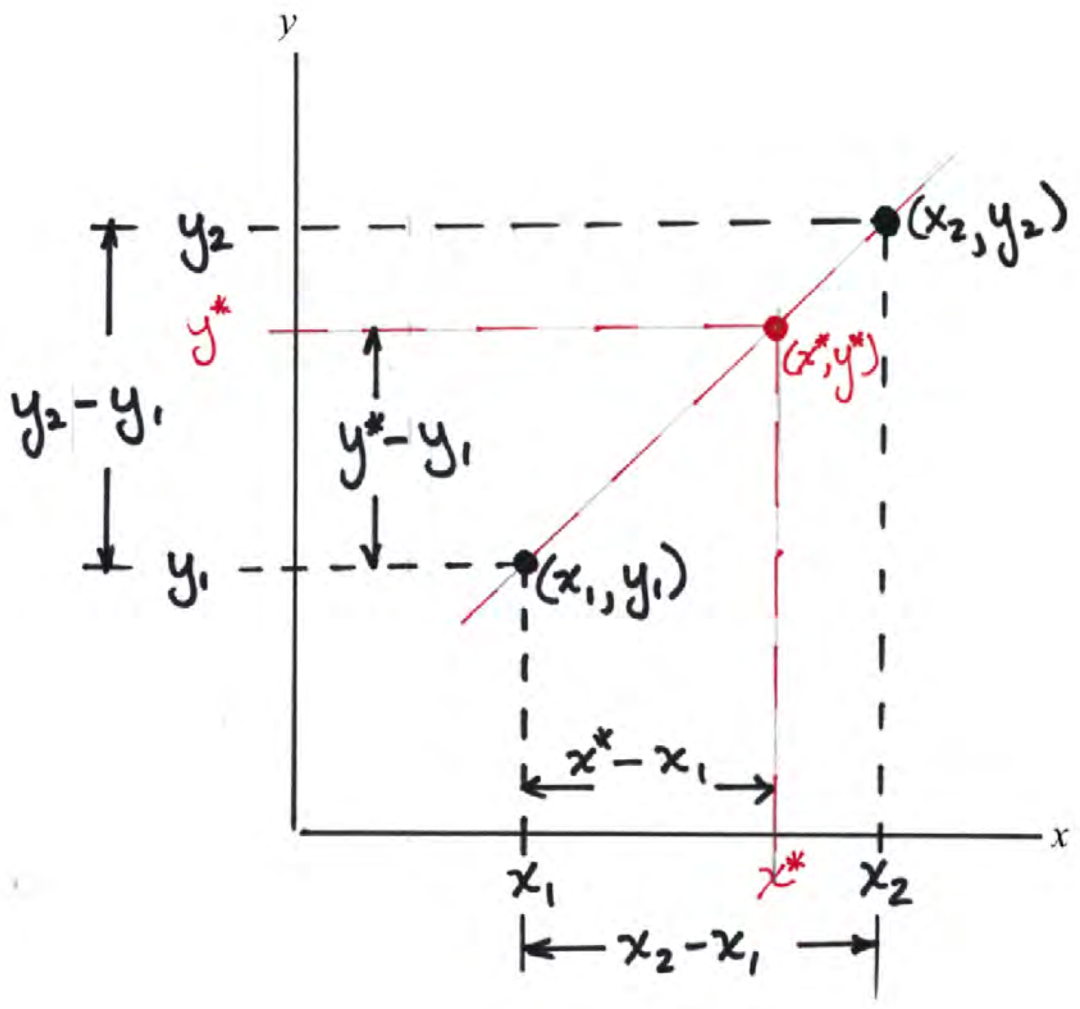


Figure 35. Sketch of two adjacent values from a table, plotted in Cartesian coordinate system..

"Estimate the value of y^* associated with the value x^* given the ordered pairs (x_1, y_1) and (x_2, y_2)."

Linear interpolation simply uses the concept of similar triangles to scale the x and y distances between the ordered pairs to the intermediate location. Equation 3 is the result of application of similar triangles to the situation described by Figure **??** and the problem statement.

$$\frac{x^* - x_1}{x_2 - x_1} = \frac{y^* - y_1}{y_2 - y_1} \tag{3}$$

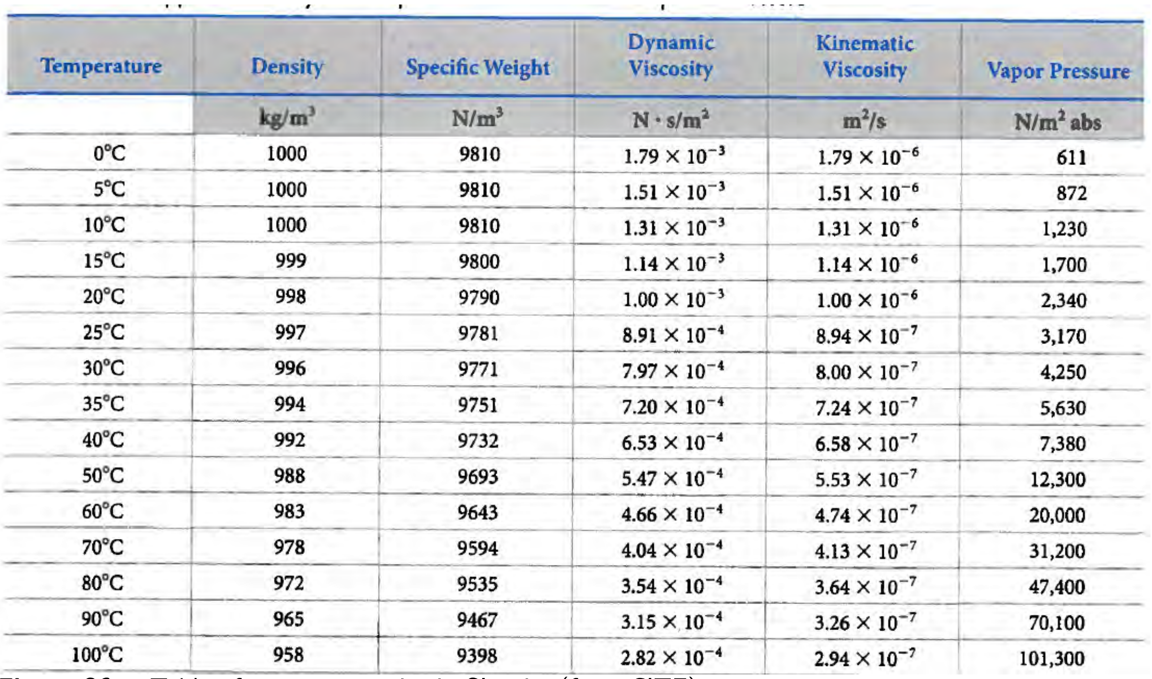
Next, apply algebra to solve Equation 3 for y^*, to obtain Equation 4

$$y^* = y_1 + \frac{(y_2 - y_1)(x^* - x_1)}{(x_2 - x_1)} \tag{4}$$

Now we can use 4 to estimate values between any two data pairs.

2.4.1 Interpolation of Values in Two Pairs

Figure 36 is a table of water properties from (CITE), that represents typically how tabular data are presented. The temperature column is arranged in increasing order and the other properties associate with temperature across a row.



Temperature	Density	Specific Weight	Dynamic Viscosity	Kinematic Viscosity	Vapor Pressure
	kg/m³	N/m³	N · s/m²	m²/s	N/m² abs
0°C	1000	9810	1.79×10^{-3}	1.79×10^{-6}	611
5°C	1000	9810	1.51×10^{-3}	1.51×10^{-6}	872
10°C	1000	9810	1.31×10^{-3}	1.31×10^{-6}	1,230
15°C	999	9800	1.14×10^{-3}	1.14×10^{-6}	1,700
20°C	998	9790	1.00×10^{-3}	1.00×10^{-6}	2,340
25°C	997	9781	8.91×10^{-4}	8.94×10^{-7}	3,170
30°C	996	9771	7.97×10^{-4}	8.00×10^{-7}	4,250
35°C	994	9751	7.20×10^{-4}	7.24×10^{-7}	5,630
40°C	992	9732	6.53×10^{-4}	6.58×10^{-7}	7,380
50°C	988	9693	5.47×10^{-4}	5.53×10^{-7}	12,300
60°C	983	9643	4.66×10^{-4}	4.74×10^{-7}	20,000
70°C	978	9594	4.04×10^{-4}	4.13×10^{-7}	31,200
80°C	972	9535	3.54×10^{-4}	3.64×10^{-7}	47,400
90°C	965	9467	3.15×10^{-4}	3.26×10^{-7}	70,100
100°C	958	9398	2.82×10^{-4}	2.94×10^{-7}	101,300

Figure 36. Table of water properties in SI units (from CITE).

Now suppose we wished to estimate the density of water at 44^o C. The two ordered pairs of temperature and density that surround 44^o C are $(40^o\ C, 992\ kg/m^3)$ and $(50^o\ C, 988\ kg/m^3)$. So, to estimate the unknown density we can apply Equation 4 and obtain the following result

$$y^* = 992 + \frac{(988 - 992)(44 - 40)}{(50 - 40)} = 990.4 \tag{5}$$

We might want to do this a lot, so we could write a simplistic script in R and remember to load it into our environment when we need it

Listing 2. R code demonstrating the interpolation equation.

```
# EXAMPLE # ** Interpolating Between Tabulated Pairs
interpolate2pairs<-function(xstar,x1,y1,x2,y2){
# apply interpolation equation
#   does not trap errors (divide by zero, etc)
  ystar <- y1 + (y2-y1)*(xstar-x1)/(x2-x1)
  return(ystar)
}
# In R Console
> interpolate2pairs(44,40,992,50,988)
[1] 990.4
>
```

For a single interrogation of a table we can stop here, but in many instances we have to interrogate a table a lot – we want some kind of program structure to handle the work so all we have to do is pass the temperature value and have the program return the density.

2.4.2 Interpolation of Values in Two Arrays

To accomplish repeated interpolation we will need to have: (1) an interpolating method (we have the beginning of one above in Listing 2), (2) the entire table so we don't have to enter the pairs, and (3) a way to automatically search the table so we don't have to look up values and supply them to the interpolator.

The table itself in this instance is relatively small, so we can simply assign values to some constant arrays in below in Listing 3.

Listing 3. R code assigning Liquid Properties.

```
# EXAMPLE # ** Assigning Constants
tempSI<-c(0.00,5.00,10.00,15.00,20.00,25.00,30.00,35.00,
   40.00,50.00,60.00,70.00,80.00,90.00,100.00)
densitySI<-c(1000.00,1000.00,1000.00,999.00,998.00,997.00,996.00,
   994.00,992.00,988.00,983.00,978.00,972.00,965.00,958.00)

# In R Console
> cbind(tempSI,densitySI)
      tempSI densitySI
 [1,]      0      1000
 [2,]      5      1000
 [3,]     10      1000
 [4,]     15       999
 [5,]     20       998
 [6,]     25       997
 [7,]     30       996
 [8,]     35       994
 [9,]     40       992
[10,]     50       988
[11,]     60       983
[12,]     70       978
[13,]     80       972
[14,]     90       965
[15,]    100       958
>
```

Returning to our example, the value 44 lies between `tempSI[9]` and `tempSI[10]`, so we desire an algorithm that starts at `tempSI[1]` and determines if the search value is between `tempSI[1]` and `tempSI[2]`, if not, then increment the row counter and determine if the search value is between `tempSI[2]` and `tempSI[3]`, and so on.

Once we locate in the searched array where the value lies then the interpolation uses the lower and upper elements of the range to interpolate. In the case of our example, once we determine the 44 lies between `tempSI[9]` and `tempSI[10]`, then the interpolation equation is

$$y^* = \texttt{densitySI[9]} + \frac{(\texttt{densitySI[10]} - \texttt{densitySI[9]})(44 - \texttt{tempSI[9]})}{(\texttt{tempSI[10]} - \texttt{tempSI[9]})} \quad (6)$$

Listing 4 is an **R** script that implements the search and interpolation just described. The script assumes that the searched array (x) is ordered and increasing – not a trivial assumption! The script has some limited error handling to test if the search value actually lies in the total range of the array before beginning the search. Once these tests are passed, then the code searches in the x array for the search value x^* and finds the two rows that contain the value. Once the rows are located, the interpolation equation is used.

Listing 4. R code to Search and Interpolate.

```
# EXAMPLE # ** Search and Interpolate
  getAvalue <- function(x,xvector,yvector){
    # returns a y value for x interpolated from (xvector,yvector)
    # xvector is assumed to be in a monotonic sequence
    # function performs limited error checks
    # NULL return is error indicator
    # T.G. Cleveland July 2007
    #
    xvlength <- length(xvector)
    yvlength <- length(yvector)
    # check that vector lengths same
    if(xvlength != yvlength){
      message("vectors xvector and yvector different lengths -- exiting function")
      return()
    }
    # check that x in range xvector
    if(x < min(xvector)){
      message(" x too small -- exiting function")
      return()
    }
    if(x > max(xvector)){
      message(" x too big -- exiting function")
      return()
    }
    #
    for (i in 1:(xvlength-1)){
      if( (x >= xvector[i]) & (x < xvector[i+1]) ){
        result = yvector[i]+(yvector[i+1]-yvector[i])*(x - xvector[i])/
        (xvector[i+1]-xvector[i])
        return(result)
      }
      # next row
    }
    # check if at endpoint
    if( (x >= xvector[xvlength-1]) & (x <= xvector[xvlength]) ){
      result = yvector[i]+(yvector[i+1]-yvector[i])*(x - xvector[i])/
      (xvector[i+1]-xvector[i])
      return(result)
    }
    # should never get to next line
    message("something is really wrong -- check the vectors!")
    return()
  }
 # In R Console:
> getAvalue(44,tempSI,densitySI)
[1] 990.4
>
```

Now we can load and run the `getAvalue` script and supply the two vectors plus the search value as shown in Listing 4

This look-up process is readily transferred to other cases, we do have to decide if the data will be coded as constants (as was done here) or read from a file – if the database is large the file read option is best. In terms of building a generic look-up tool several things actually happen in a particular order.

1. The function call loads in the table (of reading from a file, we would have to forward declare the vectors).
2. The function searches the first vector for the bounding location of the search variable.
3. Once the boundaries are located, the interpolation is performed – notice how the last boundary pair is handled.

Now we can combine the data assignment, the search and interpolate into a single function so when we want to evaluate in the future we only need the single function.

Listing 5 is an example of everything combined. Here I have embedded the `getAvalue` script into the function so the whole function itself is portbable (we don't have keep track of `getAvalue`). This embedding can be replaced with a load from a library (but then we must keep track of the path).

The library way is preferable; if `getAvalue` needs changing, we will have to change every instance of it in the code, if we miss one the code may still run and it could be years before we discover the error because a single instance of code fragment within a larger code was missed – its far better to only change a single instance of the function when maintenance is necessary.

Listing 5. R code to Return Water Density for Given Temperature.

```
# Script to return water density in SI units as a function of temperature
getDensitySI<-function(t){
# load the getAvalue() function ###################################################
  getAvalue <- function(x,xvector,yvector){
    # returns a y value for x interpolated from (xvector,yvector)
    # NULL return is error indicator
    #
    xvlength <- length(xvector)
    yvlength <- length(yvector)
    # check that vector lengths same
    if(xvlength != yvlength){
      message("vectors xvector and yvector different lengths -- exiting function")
      return()
    }
    # check that x in range xvector
    if(x < min(xvector)){
      message(" x too small -- exiting function")
      return()
    }
    if(x > max(xvector)){
      message(" x too big -- exiting function")
      return()
    }
    #
    for (i in 1:(xvlength-1)){
      if( (x >= xvector[i]) & (x < xvector[i+1]) ){
        result = yvector[i]+(yvector[i+1]-yvector[i])*(x - xvector[i])/
          (xvector[i+1]-xvector[i])
        return(result)
      }
      # next row
    }
    # check if at endpoint
    if( (x >= xvector[xvlength-1]) & (x <= xvector[xvlength]) ){
      result = yvector[i]+(yvector[i+1]-yvector[i])*(x - xvector[i])/
        (xvector[i+1]-xvector[i])
      return(result)
    }
    # should never get to next line
    message("something is really wrong -- check the vectors!")
    return()
  }
###########################################################################################
# load the data vectors, tempSI and densitySI
tempSI<-c(0.00,5.00,10.00,15.00,20.00,25.00,30.00,35.00,
  40.00,50.00,60.00,70.00,80.00,90.00,100.00)
densitySI<-c(1000.00,1000.00,1000.00,999.00,998.00,997.00,996.00,
  994.00,992.00,988.00,983.00,978.00,972.00,965.00,958.00)
# now call getAValue
result<-getAvalue(t,tempSI,densitySI)
return(result)
}
```

The "library" approach is demonstrated in Listing 6; in this listing the path in the `source()` command is unique to my machine – your path is likely to be different. I find it is useful to contain all the various codes into a single directory and source that directory once to find the path, then change all the source calls to that path. In fact that path can be a string variable and the referencing can be automatic (as long as the files exist!).

Once the look-up function is built then we can interrogate the table many times; and even build a plot of the table – these features are demonstrated in Listing 6.

Listing 6. R code demonstrating use of getDensitySI().

```
## In R Console
> # Example demonstrating use of functions
> # load in the functions (must exist -- use path on your machine)
> source('~/Dropbox/1-CE-TTU-Classes/UnderDevelopment/
   CE4333-PCH-R/6-RScripts/getAvalue.R')
> source('~/Dropbox/1-CE-TTU-Classes/UnderDevelopment/
   CE4333-PCH-R/6-RScripts/getDensitySI.R')
> # Now use them
> getDensitySI(44)
[1] 990.4
> getDensitySI(54)
[1] 986
> getDensitySI(88)
[1] 966.4
> t<-seq(0,100,2) # make a temperature vector 0 to 100 in 2 degree increments
> d<-numeric(0) # forward declare d to store results
> howMany<-length(t)
> for(i in 1:howMany){
+     d[i]<-getDensitySI(t[i])
+ }
> plot(t,d,type="l",xlab="Degrees Celsius",ylab="Density (kg/m^3)")
>
```

The resulting plot is shown on Figure 37 below.

2.5 Sorting

Another frequent task in engineering hydraulics is the seemingly mundane task of sorting or ordering things. Here we explore a couple of simple sorting algorithms, just to show some of the thoughts that go into such a task, then will ultimately resort to the internal sorting routines built into R.

2.5.1 Bubble Sort

The bubble sort is a place to start despite it's relative slowness. It is a pretty reviled algorithm (read the Wikipedia entry), but it is the algorithm that a naive programmer might cobble together in a hurry, and despite its shortcomings (its really slow and inefficient), it is robust.

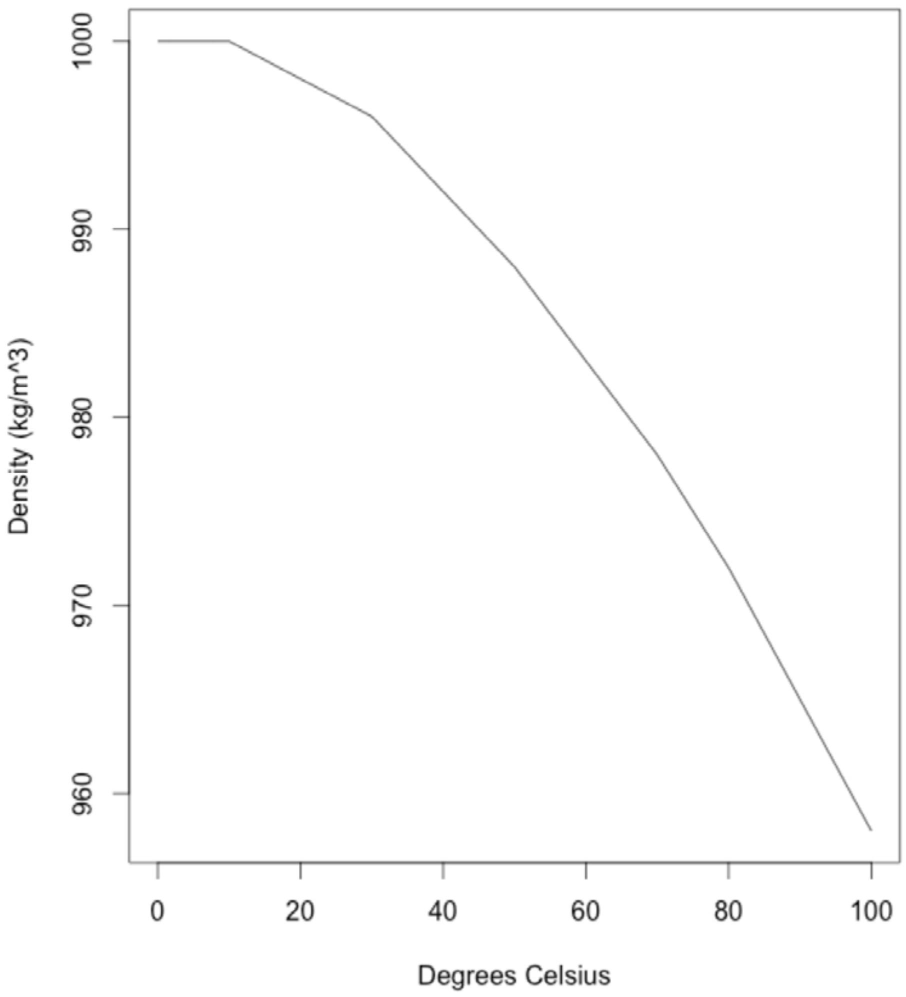


Figure 37. Plot of density versus temperature generated using the `getDensity()` function..

Here is a description of the sorting task as described by Christian and Griffiths (2016) (pg. 65):

> "Imagine you want to alphabetize your unsorted collection of books. A natural approach would be just to scan across the shelf looking for out-of-order pairs – Wallace followed by Pynchon, for instance – and flipping them around. Put Pynchon ahead of Wallace, then continue your scan, looping around to the beginning of the shelf each time you reach the end. When you make a complete pass without finding any more out-of-order pairs on the entire shelf, then you know the job is done.
>
> This process is a Bubble Sort, and it lands us in quadratic time. There are n books out of order, and each scan through the shelf can move each one at most one position. (We spot a tiny problem, make a tiny fix.) So in the worst case, where the shelf is perfectly backward, at least one book will need to be moved n positions. Thus a maximum of n passes through n books, which gives us $O(n2)$ in the worst case.[7] ...... For instance, it means that sorting five shelves of books will take not five times as long as sorting a single shelf, but twenty-five times as long."

Converting the word description into **R** is fairly simple. We will have a vector of n numbers (we use a vector because its easy to step through the different positions), and we will scan through the vector once (and essentially find the smallest thing), and put it into the first position. Then we scan again from the second position and find the smallest thing remaining, and put it into the second position, and so on until the last scan which should have the remaining largest thing. If we desire a decreasing order, simply change the sense of the comparison.

Listing 7 is an **R** script that implements the algorithm – in the script the actual sort is treated as a function (we may actually want to use it again someday) which is loaded into the programming environment first, then an array is defined, and sorted. The program (outside of the sorting algorithm) is really quite simple.

- Load the sorting function.
- Load contents into an array to be sorted.
- Echo (print) the array (so we can verify the data are loaded as anticipated).
- Sort the array, put the results back into the array (an in-place sort).
- Report the results.

Listing 7. R code demonstrating the naive bubble sort.

```
###############################################################
rm(list=ls())  # clear the object list (i.e. deallocate and clear memory)
### Bubble Sort Function -- Needs to be defined before sending array to sort ###
# Bubble Sort Function
```

[7]Actually, the average running time for Bubble Sort isn't any better, as books will, on average, be $n/2$ positions away from where they?re supposed to end up. One would round the $n/2$ passes of n books up to $O(n2)$.

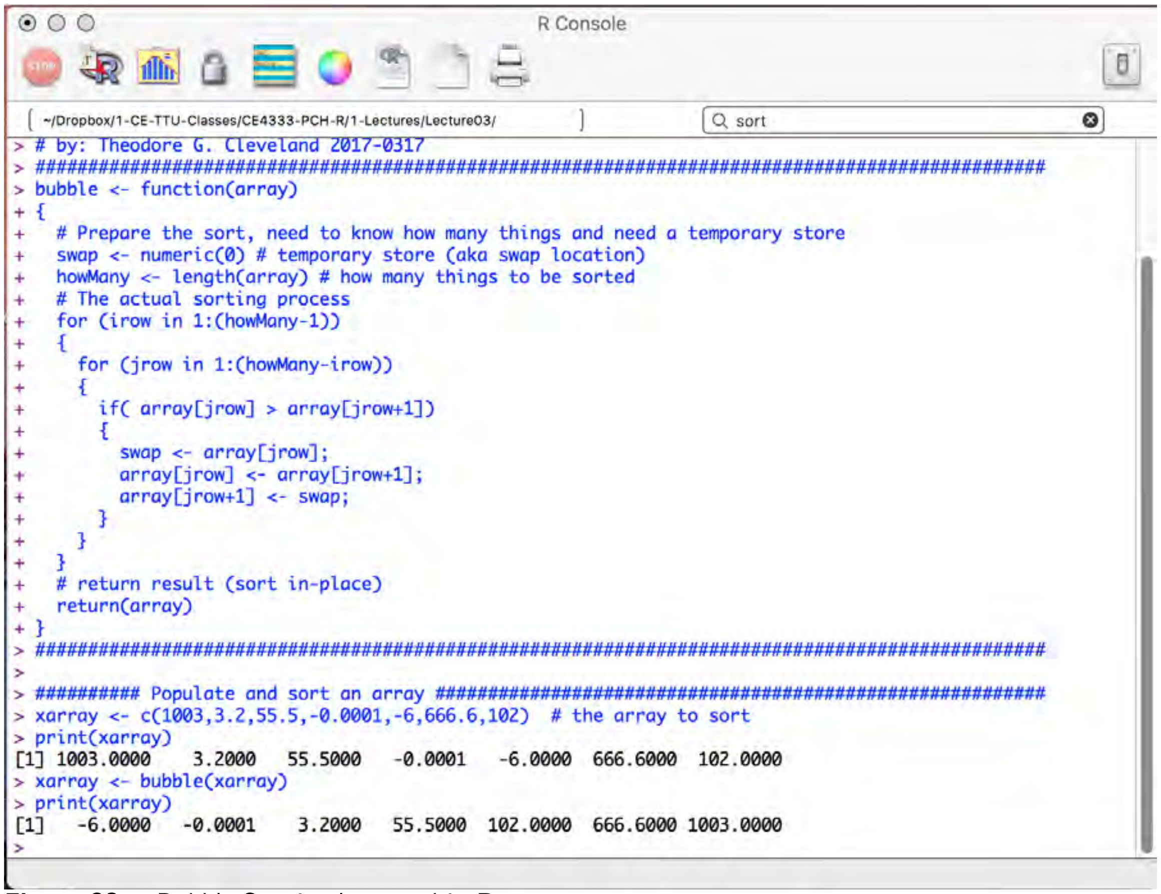
```
# MyLocation: ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/1-Lectures/Lecture03/ScriptsInLecture
# Bubble Sort with array indexing starting at [1]
# Compare to Python or C where arrays start at [0])
# by: Theodore G. Cleveland 2017-0317
#############################################################
bubble <- function(array)
{
  # Prepare the sort, need to know how many things and need a temporary store
  swap <- numeric(0) # temporary store (aka swap location)
  howMany <- length(array) # how many things to be sorted
  # The actual sorting process
  for (irow in 1:(howMany-1))
  {
    for (jrow in 1:(howMany-irow))
    {
      if( array[jrow] > array[jrow+1])
      {
        swap <- array[jrow];
        array[jrow] <- array[jrow+1];
        array[jrow+1] <- swap;
      }
    }
  }
  # return result (sort in-place)
  return(array)
}
############################################################

############################################################
xarray <- c(1003,3.2,55.5,-0.0001,-6,666.6,102)  # the array to sort
print(xarray)
xarray <- bubble(xarray)
print(xarray)
############################################################
```

Figure 38 is a screen capture of the script running. In the figure we see that the program (near the bottom of the file) assigns the values to the vector named array and the initial order of the array is $[1003, 3.2, 55.5, -0.0001, -6, 666.6, 102]$. The smallest value in the example is -6 and it appears in the 5-th position, not the 1-st as it should.

The first pass through the array will move the largest value, 1003, in sequence to the right until it occupies the last position. Repeated passes through the array move the remaining largest values to the right until the array is ordered. One can consider the values of the array at each scan of the array as a series of transformations (`irow`-th scan) – in practical cases we don't necessarily care about the intermediate values, but here because the size is manageable and we are trying to get our feet wet with algorithms, we can look at the values.

The sequence of results (transformations) after each pass through the array is shown in the following list:

1. Initial value: $[1003, 3.2, 55.5, -0.0001, -6, 666.6, 102]$.
2. First pass: $[3.2, 55.5, -0.0001, -6, 666.6, 102, 1003]$.
3. Second pass: $[3.2, -0.0001, -6, 55.5, 102, 666.6, 1003]$.
4. Third pass: $[-0.0001, -6, 3.2, 55.5, 102, 666.6, 1003]$.
5. Fourth pass: $[-6, -0.0001, 3.2, 55.5, 102, 666.6, 1003]$.
6. Fifth pass: $[-6, -0.0001, 3.2, 55.5, 102, 666.6, 1003]$. Sorted, fast scan.
7. Sixth pass: $[-6, -0.0001, 3.2, 55.5, 102, 666.6, 1003]$. Sorted, fast scan.


R Console
~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/1-Lectures/Lecture03/
sort

```
> # by: Theodore G. Cleveland 2017-0317
> ##############################################################################################
> bubble <- function(array)
+ {
+   # Prepare the sort, need to know how many things and need a temporary store
+   swap <- numeric(0) # temporary store (aka swap location)
+   howMany <- length(array) # how many things to be sorted
+   # The actual sorting process
+   for (irow in 1:(howMany-1))
+   {
+     for (jrow in 1:(howMany-irow))
+     {
+       if( array[jrow] > array[jrow+1])
+       {
+         swap <- array[jrow];
+         array[jrow] <- array[jrow+1];
+         array[jrow+1] <- swap;
+       }
+     }
+   }
+   # return result (sort in-place)
+   return(array)
+ }
> ##############################################################################################
>
> ########## Populate and sort an array ##########################################################
> xarray <- c(1003,3.2,55.5,-0.0001,-6,666.6,102)  # the array to sort
> print(xarray)
[1] 1003.0000    3.2000   55.5000   -0.0001   -6.0000  666.6000  102.0000
> xarray <- bubble(xarray)
> print(xarray)
[1]   -6.0000   -0.0001    3.2000   55.5000  102.0000  666.6000 1003.0000
>
```

Figure 38. Bubble Sort implemented in **R**.

We could probably add additional code to break from the scans when we have a single pass with no exchanges – while meaningless in this example, for larger collections of things, being able to break out when the sorting is complete is a nice feature.

2.5.2 Insertion Sort

The next type of sorting would be to select one item and locate it either left or right of an adjacent item based on its size – like sorting a deck of cards, or perhaps a better description – again using the bookshelf analog from Christian and Griffiths (2016) (pg. 65):

> ". . . . . . You might take a different tack – pulling all the books off the shelf and putting them back in place one by one. You'd put the first book in the middle of the shelf, then take the second and compare it to the first, inserting it either to the right or to the left. Picking up the third book, you'd run through the books on the shelf from left to right until you found the right spot to tuck it in. Repeating this process, gradually all of the books would end up sorted on the shelf and you'd be done. Computer scientists call this, appropriately enough, Insertion Sort. The good news is that it's arguably even more intuitive than Bubble Sort and doesn't have quite the bad reputation. The bad news is that it's not actually that much faster. You still have to do one insertion for each book. And each insertion still involves moving past about half the books on the shelf, on average, to find the correct place.
>
> Although in practice Insertion Sort does run a bit faster than Bubble Sort, again we land squarely, if you will, in quadratic time. Sorting anything more than a single bookshelf is still an unwieldy prospect."

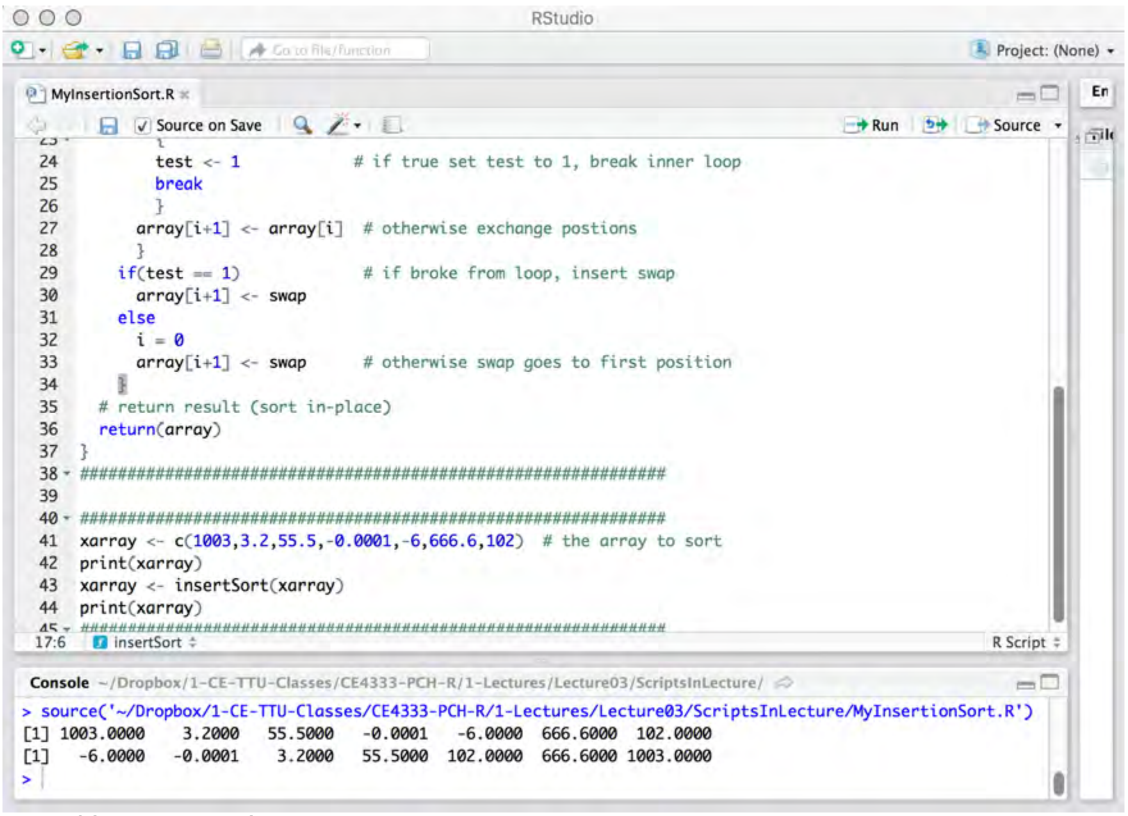
Listing 8 is an **R** implementation of a straight insertion sort. The script is quite compact, and I used indentation and extra line spacing to keep track of the scoping delimiters. The sort works as follows, take the an element of the array (start with 2 and work to the right) and put it into a temporary location (called `swap` in my script). Then compare locations to the left of `swap`. If smaller, then break from the loop, exchange values, otherwise the values are currently ordered. Repeat (starting at the next element) , when all elements have been traversed the resulting vector is sorted. Here are the transformations for each pass through the outer loop:

1. Pass 0: $[1003, 3.2, 55.5, -0.0001, -6, 666.6, 102]$, Initial array.
2. Pass 1: $[3.2, 1003, 55.5, -0.0001, -6., 666.6, 102]$.
3. Pass 2: $[3.2, 55.5, 1003, -0.0001, -6., 666.6, 102]$.
4. Pass 3: $[-0.0001, 3.2, 55.5, 1003, -6., 666.6, 102]$.
5. Pass 4: $[-6, -0.0001, 3.2, 55.5, 1003., 666.6, 102]$.
6. Pass 5: $[-6, -0.0001, 3.2, 55.5, 666.6, 1003, 102]$.
7. Pass 6: $[-6, -0.0001, 3.2, 55.5, 102, 666.6, 1003]$, Sorted array.

Figure 39 is a screen capture of the insertion sort in operation. Insertion sorting is reasonably fast for small lists (about 50 or so elements) and forms the basis of the internal sorts in other routines that divide up the overall list into smaller lists, sort the smaller lists, then uses a merge to collate back to the overall list (now sorted).

Listing 8. R code demonstrating the insertion sort.

```
### Straight Insertion Sort Function by: Theodore G. Cleveland 2017-0317
rm(list=ls())  # clear the object list (i.e. deallocate and clear memory)
################################################################
insertSort <- function(array){
# Prepare the sort, need to know how many things and need a temporary store
  swap <- numeric(0) # temporary store (aka swap location)
  howMany <- length(array) # how many things to be sorted
  for (j in 2:howMany)  # select each position in turn
    {
    test <- 0           # set a test value, used to insert later
    swap <- array[j]    # current position to swap
    for (i in seq(j-1,1,by=-1)) #find place to insert by ...
      {
      if (array[i] <= swap)  # test if current position is bigger
        {
        test <- 1            # if true set test to 1, break inner loop
        break
        }
      array[i+1] <- array[i]  # otherwise exchange postions
      }
    if(test == 1)             # if broke from loop, insert swap
      array[i+1] <- swap
    else
      i = 0
      array[i+1] <- swap      # otherwise swap goes to first position
    }
  return(array) }    # return result (sort in-place)
##############################################################
xarray <- c(1003,3.2,55.5,-0.0001,-6,666.6,102)  # the array to sort
print(xarray)
xarray <- insertSort(xarray)
print(xarray)
##############################################################
```



```
          {
        test <- 1           # if true set test to 1, break inner loop
        break
        }
      array[i+1] <- array[i]  # otherwise exchange postions
      }
    if(test == 1)             # if broke from loop, insert swap
      array[i+1] <- swap
    else
      i = 0
      array[i+1] <- swap      # otherwise swap goes to first position
    }
  # return result (sort in-place)
  return(array)
}
################################################################

################################################################
xarray <- c(1003,3.2,55.5,-0.0001,-6,666.6,102)  # the array to sort
print(xarray)
xarray <- insertSort(xarray)
print(xarray)
################################################################
```

```
~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/1-Lectures/Lecture03/ScriptsInLecture/
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/1-Lectures/Lecture03/ScriptsInLecture/MyInsertionSort.R')
[1] 1003.0000    3.2000   55.5000   -0.0001   -6.0000  666.6000  102.0000
[1]   -6.0000   -0.0001    3.2000   55.5000  102.0000  666.6000 1003.0000
>
```

Figure 39. Insertion Sort implemented in **R**.

2.5.3 Merge Sort

A practical extension of these slow sorts is called the Merge Sort. It is an incredibly useful method. One simply breaks up the items into smaller arrays, sorts those arrays - then merges the sub-arrays into larger arrays (now already sorted), and finally merges the last two arrays into the final, single, sorted array.

Here is a better description, again from Christian and Griffiths (2016):

> "...... information processing began in the US censuses of the nineteenth century, with the development, by Herman Hollerith and later by IBM, of physical punch-card sorting devices. In 1936, IBM began producing a line of machines called "collators" that could merge two separately **ordered** stacks of cards into one. *As long as the two stacks were themselves sorted*, the procedure of merging them into a single sorted stack was incredibly straightforward and took linear time: simply compare the two top cards to each other, move the smaller of them to the new stack you're creating, and repeat until finished.
>
> The program that John von Neumann wrote in 1945 to demonstrate the power of the stored-program computer took the idea of collating to its beautiful and ultimate conclusion. Sorting two cards is simple: just put the smaller one on top. And given a pair of two-card stacks, both of them sorted, you can easily collate them into an ordered stack of four. Repeating this trick a few times, you'd build bigger and bigger stacks, each one of them already sorted. Soon enough, you could collate yourself a perfectly sorted full deck – with a final climactic merge, like a riffle shuffle's order-creating twin, producing the desired result. This approach is known today as Merge Sort, one of the legendary algorithms in computer science."

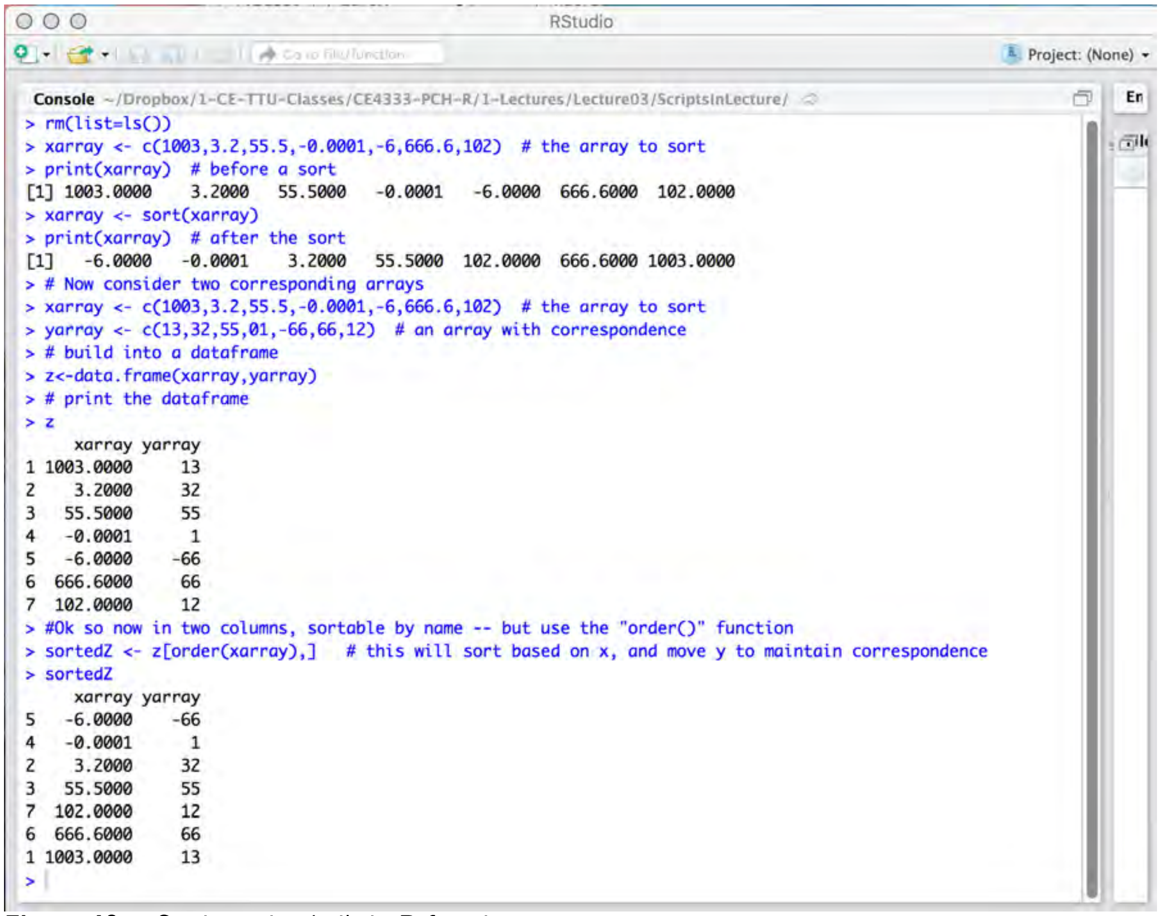
There are several other variants related to Merge Sort; Quicksort and Heapsort being relatives. The creation of a Merge Sort is left to the reader if there is a need, and at this point we can just use the built-in `sort()` and/or `order()` functions in **R** – which implements either a Shellsort (useful if character strings are to be sorted) or Quicksort (used if numeric values are supplied). We also have to supply if we want increasing or decreasing sorts.

2.5.4 Built-In R Sorts

Figure 40 illustrates using the built-in functions. For an ordinary sort, we simply use the function name `sort()` and direct its output into an object (it can even be the same vector as shown in the figure).

If we wish to sort several related columns, based on values in one of the columns, it is easiest to construct a data frame (like a matrix), then order the contents based on one of the columns, and send the results to another data frame, or we can send the result back to itself. Usually when we are manipulating multiple columns, we are operating in a "relational database" kind of mindset, and it is probably to our benefit to not destroy the original association structure. Be aware of the syntax of a dataframe function, you will notice there is a comma that appears at the end of the function that is important for the script to function.

For example, `z <- z[order(xarray),]` will function as shown, whereas `zztop <- z[order(xarray)]` will not.



RStudio
Project: (None)
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/1-Lectures/Lecture03/ScriptsInLecture/

```
> rm(list=ls())
> xarray <- c(1003,3.2,55.5,-0.0001,-6,666.6,102)  # the array to sort
> print(xarray)  # before a sort
[1] 1003.0000    3.2000   55.5000   -0.0001   -6.0000  666.6000  102.0000
> xarray <- sort(xarray)
> print(xarray)  # after the sort
[1]   -6.0000   -0.0001    3.2000   55.5000  102.0000  666.6000 1003.0000
> # Now consider two corresponding arrays
> xarray <- c(1003,3.2,55.5,-0.0001,-6,666.6,102)  # the array to sort
> yarray <- c(13,32,55,01,-66,66,12)  # an array with correspondence
> # build into a dataframe
> z<-data.frame(xarray,yarray)
> # print the dataframe
> z
     xarray yarray
1 1003.0000     13
2    3.2000     32
3   55.5000     55
4   -0.0001      1
5   -6.0000    -66
6  666.6000     66
7  102.0000     12
> #Ok so now in two columns, sortable by name -- but use the "order()" function
> sortedZ <- z[order(xarray),]   # this will sort based on x, and move y to maintain correspondence
> sortedZ
     xarray yarray
5   -6.0000    -66
4   -0.0001      1
2    3.2000     32
3   55.5000     55
7  102.0000     12
6  666.6000     66
1 1003.0000     13
> 
```

Figure 40. Sorting using built-in **R** functions.

Now if we return to the interpolation chapter just before this one, we can immediately see a need for sorting. The interpolation algorithm *assumes* that the explanatory structure (x-axis) is ordered, otherwise the interpolation equation will return garbage.

I conclude the section on sorting with one more quoted section from Christian and Griffiths (2016) about the value for sorting – which is already relevant to a lot of computational hydraulics:

> "The poster child for the advantages of sorting would be an Internet search engine like Google. It seems staggering to think that Google can take the search phrase you typed in and scour the entire Internet for it in less than

half a second. Well, it can't – but it doesn't need to. If you're Google, you are almost certain that (a) your data will be searched, (b) it will be searched not just once but repeatedly, and (c) the time needed to sort is somehow "less valuable" than the time needed to search. (Here, sorting is done by machines ahead of time, before the results are needed, and searching is done by users for whom time is of the essence.) All of these factors point in favor of tremendous up-front sorting, which is indeed what Google and its fellow search engines do."

2.6 Exercise Set

1. Build a function `getDensityUS()` that searches the table in Figure 36 and returns the density of water in US customary units for a value of temperature supplied in degrees Farenheight.

   Submit your code and screen captures of the density for temperatures of $43^o\ F$, $146^o\ F$, and $210^o\ F$.

2. Later in the class we will need functions to return viscosity to compute head losses in pipe networks.
   Build and test a function `getKinViscosityUS()` that searches the table in Figure **??** and returns the kinematic viscosity of water in US customary units for a value of temperature supplied in degrees Farenheight

   Submit the code and screen captures of the kinematic viscosity for temperatures of $43^o\ F$, $146^o\ F$, and $210^o\ F$.

3. Build and test a function `getKinViscositySI()` that searches the table in Figure 36 and returns the kinematic viscosity of water in SI units for a value of temperature supplied in degrees Celsius

   Submit the code and screen captures of the kinematic viscosity for temperatures of $13^o\ C$, $66^o\ C$, and $97^o\ C$.

This exercise set is also located on the class server as `ES-1`.

3 Numerical Methods – Integrals, Derivatives, and Newton's Method

3.1 Numerical Integration of Functions

Numerical integration is the numerical approximation of

$$I = \int_a^b f(x)dx \tag{7}$$

Consider the problem of determining the shaded area under the curve $y = f(x)$ from $x = a$ to $x = b$, as depicted in Figure 41, and suppose that analytical integration is not feasible.

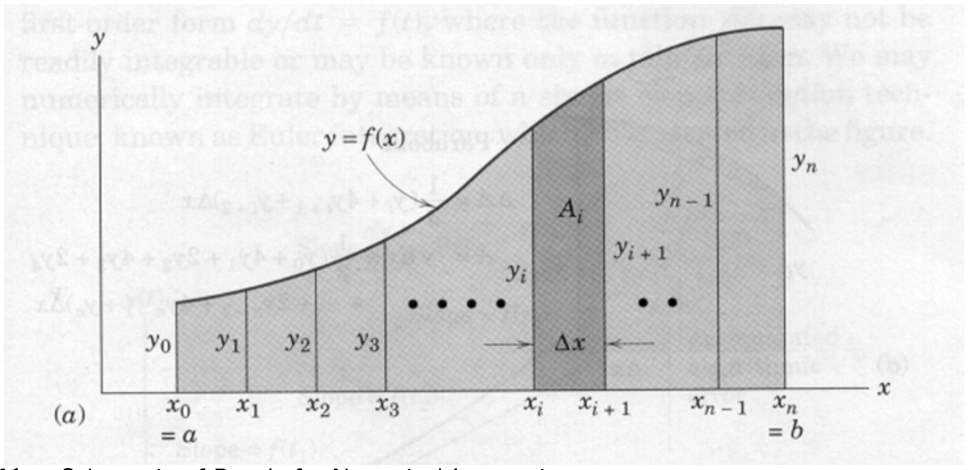


Figure 41. Schematic of Panels for Numerical Integration. .

The function may be known in tabular form from experimental measurements or it may be known in an analytical form. The function is taken to be continuous within the interval $a < x < b$. We may divide the area into n vertical panels, each of width $\Delta x = (b - a)/n$, and then add the areas of all strips to obtain $A \approx \int y dx$.

A representative panel of area A_i is shown with darker shading in the figure. Three useful numerical approximations are listed in the following sections. The approximations differ in how the function is represented by the panels — in all cases the function is approximated by known polynomial models between the panel end points.

In each case the greater the number of strips, and correspondingly smaller value of Δx, the more accurate the approximation. Typically, one can begin with a relatively small number of panels and increase the number until the resulting area approximation stops changing.

3.1.1 Rectangular Panels

Figure 42 is a schematic of a rectangular panels. The figure is assuming the function structure is known and can be evaluated at an arbitrary location in the Δx dimension. Each panels is treated as a rectangle, as shown by the representative panel whose

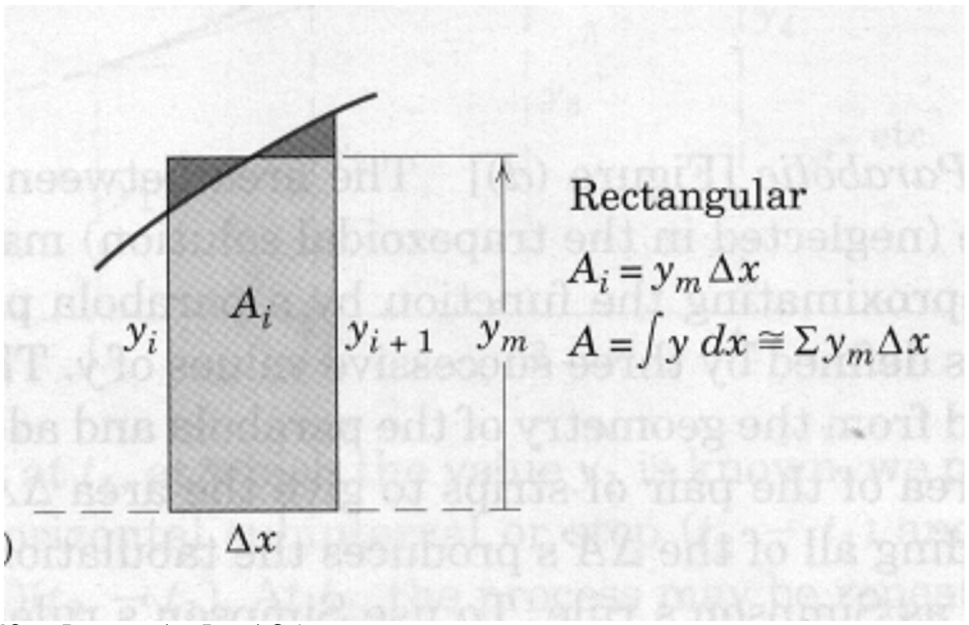


Figure 42. Rectangular Panel Schematic..

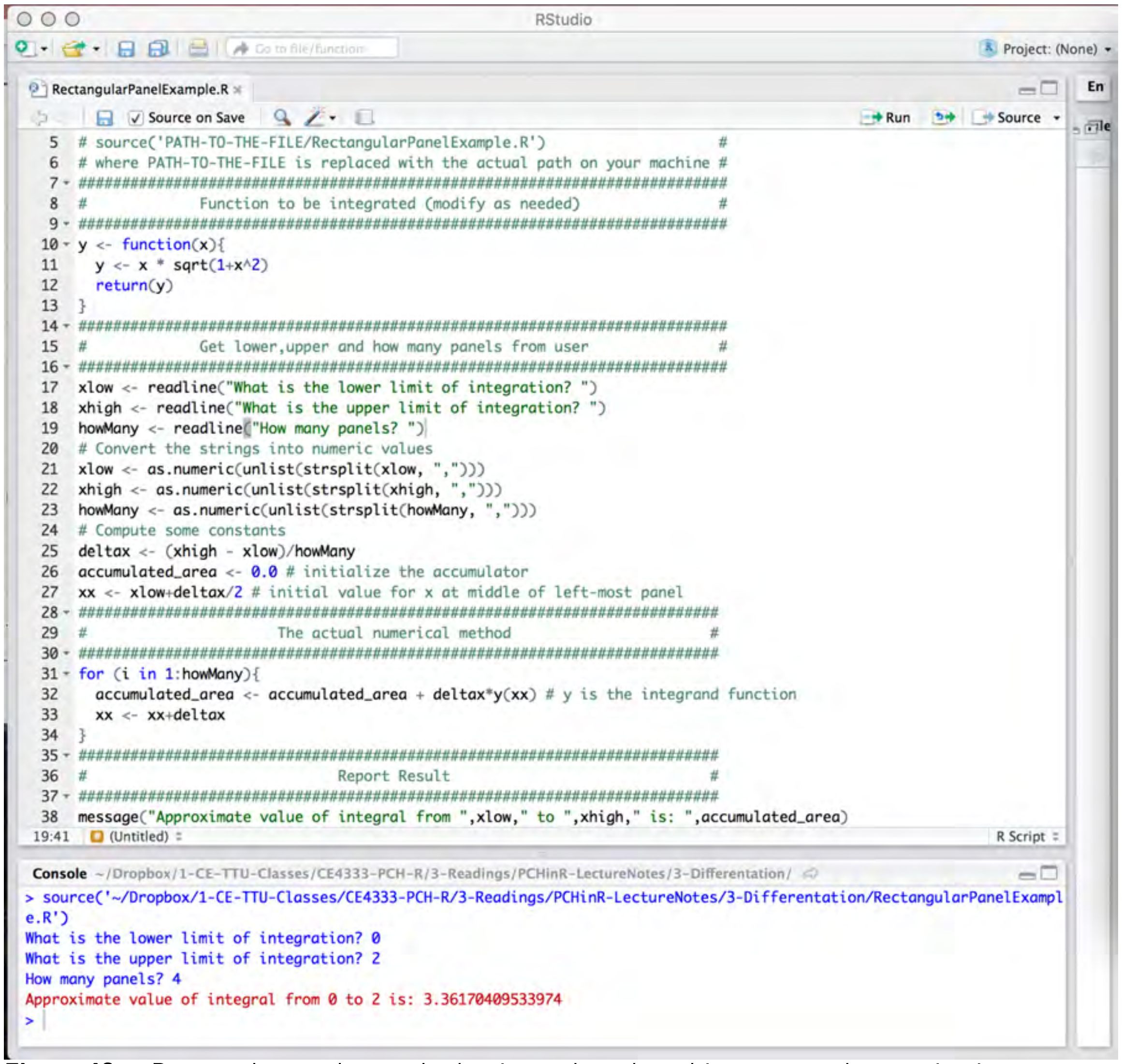
height y_m is chosen visually so that the small cross-hatched areas are as nearly equal as possible. Thus, we form the sum $\sum y_m$ of the effective heights and multiply by Δx. For a function known in analytical form, a value for y_m equal to that of the function at the midpoint $x_i + \Delta x/2$ may be calculated and used in the summation.

For tabulated functions, we have to choose to either take y_m as the value at the left endpoint or right endpoint. This limitation is often quite handy when we are trying to integrate a function that is integrable, but undefined on one endpoint.

Lets try some examples in **R**.

Problem: Find the area under the curve $y = x\sqrt{1+x^2}$ from $x = 0$ to $x = 2$.

Solution: One solution is shown in Figure 43,which is a screen capture of a rudimentary code that implements the rectangular panel method.[8]


RStudio
RectangularPanelExample.R
Source on Save
Run
Source
Project: (None)
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/

```r
# source('PATH-TO-THE-FILE/RectangularPanelExample.R')             #
# where PATH-TO-THE-FILE is replaced with the actual path on your machine #
###########################################################################
#              Function to be integrated (modify as needed)               #
###########################################################################
y <- function(x){
  y <- x * sqrt(1+x^2)
  return(y)
}
###########################################################################
#              Get lower,upper and how many panels from user              #
###########################################################################
xlow <- readline("What is the lower limit of integration? ")
xhigh <- readline("What is the upper limit of integration? ")
howMany <- readline("How many panels? ")
# Convert the strings into numeric values
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
accumulated_area <- 0.0 # initialize the accumulator
xx <- xlow+deltax/2 # initial value for x at middle of left-most panel
##########################################################################
#                      The actual numerical method                       #
##########################################################################
for (i in 1:howMany){
  accumulated_area <- accumulated_area + deltax*y(xx) # y is the integrand function
  xx <- xx+deltax
}
##########################################################################
#                              Report Result                             #
##########################################################################
message("Approximate value of integral from ",xlow," to ",xhigh," is: ",accumulated_area)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RectangularPanelExampl
e.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4
Approximate value of integral from 0 to 2 is: 3.36170409533974
>
```

Figure 43. Rectangular panel example showing code and resulting computed area using just 4 panels..

The script does not implement any kind of error checking – we could enter text values for the lower and upper values of x as well as the number of panels to use, and the script would attempt to run. A better version would force us to enter numeric values, and check for undefined ranges and such; devotion to error trapping is typical for professional programs where you are going to distribute executable modules and not expect the end user to be a programmer.

[8]The exact solution is A=3.393477

For the time being, we will accept this approach (error trapping is left as an exercise), however in your own scripts you should implement error traps where possible – you may start without them but as you maintain your scripts, you will learn where data entry errors occur and trap them.[9]

The actual listing depicted in Figure 43 is shown in Listing 9

Listing 9. R code demonstrating Rectangular Panel Numerical Integration.

```
# R script to implement rectangular panel numerical integration
################################# NOTE ####################################
## The interactive input requires the script to be sourced                #
# In R console the command line would be                                  #
# source('PATH-TO-THE-FILE/RectangularPanelExample.R')                    #
# where PATH-TO-THE-FILE is replaced with the actual path on your machine #
###########################################################################
#             Function to be integrated (modify as needed)                #
###########################################################################
y <- function(x){
  y <- x * sqrt(1+x^2)
  return(y)
}
###########################################################################
#             Get lower,upper and how many panels from user               #
###########################################################################
xlow <- readline("What is the lower limit of integration? ")
xhigh <- readline("What is the upper limit of integration? ")
howMany <- readline("How many panels? ")
# Convert the strings into numeric values
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
accumulated_area <- 0.0 # initialize the accumulator
xx <- xlow+deltax/2 # initial value for x at middle of left-most panel
##########################################################################
#                      The actual numerical method                       #
##########################################################################
for (i in 1:howMany){
  accumulated_area <- accumulated_area + deltax*y(xx) # y is the integrand function
  xx <- xx+deltax
}
##########################################################################
#                             Report Result                              #
##########################################################################
message("Approximate value of integral from ",xlow," to ",xhigh," is: ",accumulated_area)
```

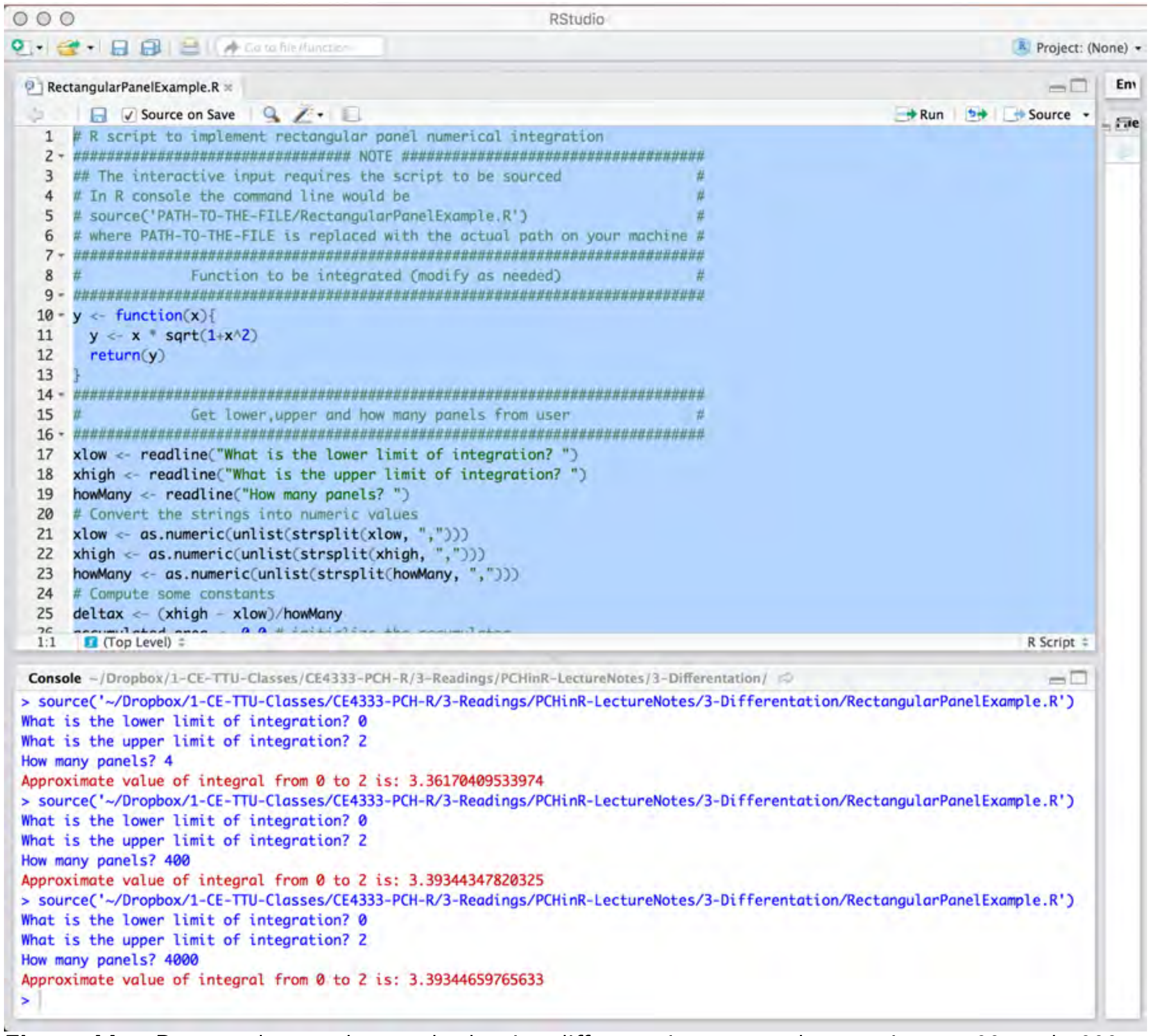
Figure 44 is the same program run using 4,400, and 4000 panels observe the difference in computed area as well as the results closeness to the exact solution.

3.1.2 Trapezoidal Panels

The trapezoidal panels are approximated as shown in Figure 45. The area A_i is the average height $(y_i + y_{i+1})/2$ times Δx. Adding the areas gives the area approximation as tabulated. For the example with the curvature shown, the approximation will be on the low side. For the reverse curvature, the approximation will be on the high side. The trapezoidal approximation is commonly used with tabulated values.

The same example as presented for rectangular panels is repeated, except using trapezoidal panels. The code is changed because we will evaluate at each end of the panel (so no fussing to find an intermediate estimate for where to evaluate the function).

[9]For example, traps are used to force the user to enter a value that actually is meaningful, or select a default value, or internally prevent division by zero.



```
# R script to implement rectangular panel numerical integration
################################ NOTE ####################################
## The interactive input requires the script to be sourced               #
# In R console the command line would be                                 #
# source('PATH-TO-THE-FILE/RectangularPanelExample.R')                   #
# where PATH-TO-THE-FILE is replaced with the actual path on your machine #
##########################################################################
#              Function to be integrated (modify as needed)              #
##########################################################################
y <- function(x){
  y <- x * sqrt(1+x^2)
  return(y)
}
##########################################################################
#              Get lower,upper and how many panels from user             #
##########################################################################
xlow <- readline("What is the lower limit of integration? ")
xhigh <- readline("What is the upper limit of integration? ")
howMany <- readline("How many panels? ")
# Convert the strings into numeric values
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RectangularPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4
Approximate value of integral from 0 to 2 is: 3.36170409533974
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RectangularPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 400
Approximate value of integral from 0 to 2 is: 3.39344347820325
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RectangularPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4000
Approximate value of integral from 0 to 2 is: 3.39344659765633
>
```

Figure 44. Rectangular panel example showing difference in computed area using 4, 400, and 4000 panels. The 4000 panel result is essentially equivalent to the exact solution. Such convergence to exact values is typical.

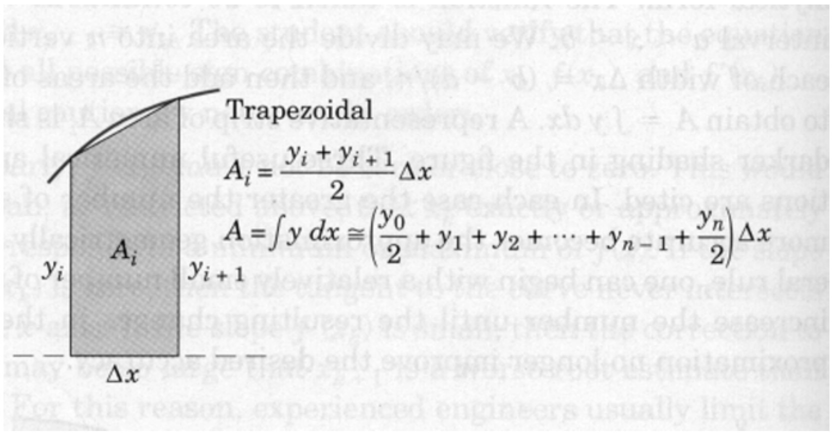
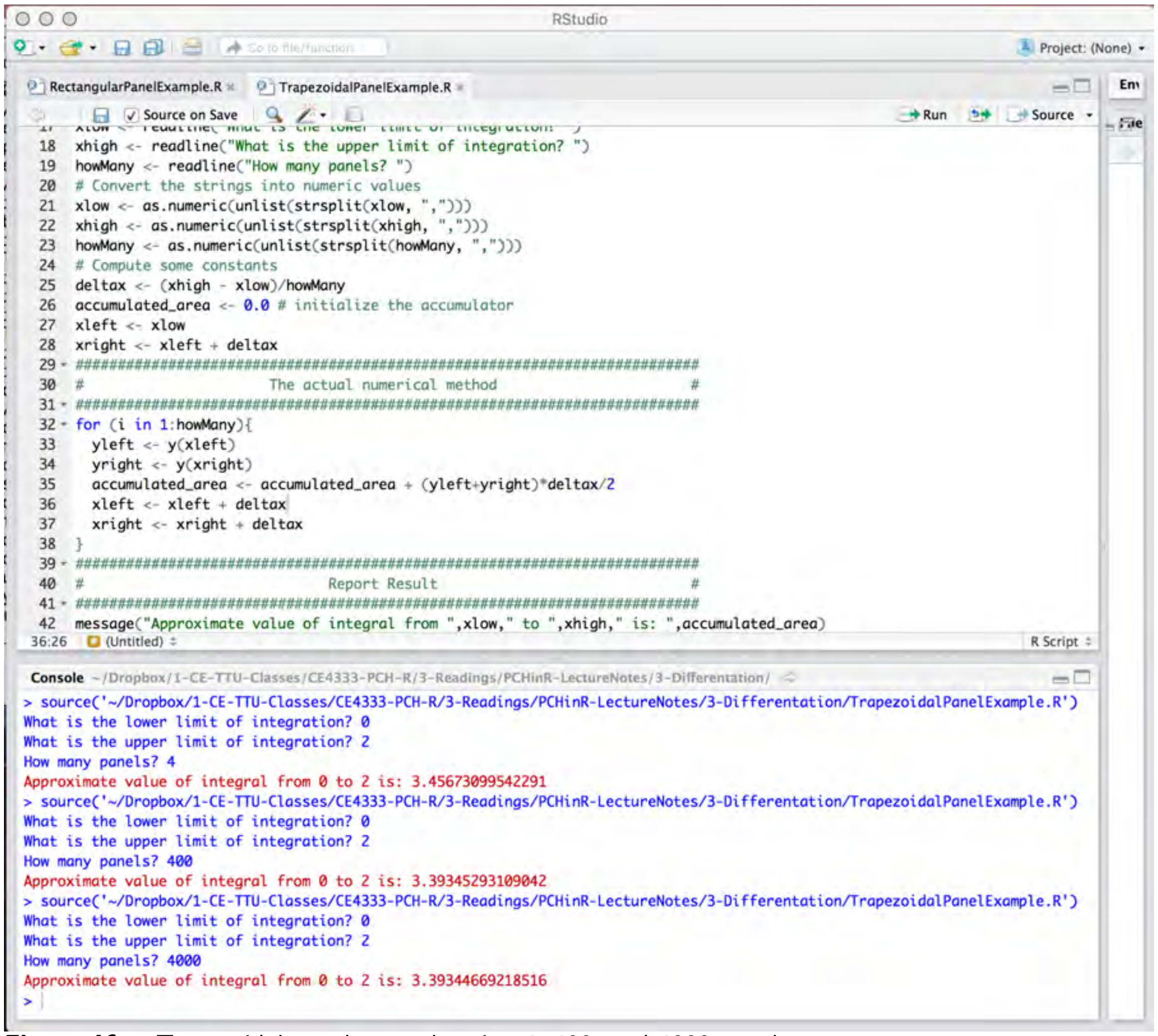

Trapezoidal
$A_i = \frac{y_i + y_{i+1}}{2}\Delta x$
$A = \int y\,dx \cong \left(\frac{y_0}{2} + y_1 + y_2 + \cdots + y_{n-1} + \frac{y_n}{2}\right)\Delta x$
A_i
y_i
y_{i+1}
Δx

Figure 45. Trapezoidal Panel Schematic..

Figure 46 illustrates the trapezoidal method for approximating an integral. In the example, the left and right panel endpoints in x are set as separate variables x_{left} and x_{right} and incremented by Δx as we step through the count-controlled repetition to accumulate the area. The corresponding y values are computed within the loop and averaged, then multiplied by Δx and added to the accumulator. Finally the x values are incremented.


RStudio
Project: (None)
RectangularPanelExample.R
TrapezoidalPanelExample.R
Source on Save
Run
Source
36:26
(Untitled)
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/

```
xlow <- readline("What is the lower limit of integration? ")
xhigh <- readline("What is the upper limit of integration? ")
howMany <- readline("How many panels? ")
# Convert the strings into numeric values
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
accumulated_area <- 0.0 # initialize the accumulator
xleft <- xlow
xright <- xleft + deltax
############################################################################
#                       The actual numerical method                        #
############################################################################
for (i in 1:howMany){
  yleft <- y(xleft)
  yright <- y(xright)
  accumulated_area <- accumulated_area + (yleft+yright)*deltax/2
  xleft <- xleft + deltax
  xright <- xright + deltax
}
############################################################################
#                              Report Result                               #
############################################################################
message("Approximate value of integral from ",xlow," to ",xhigh," is: ",accumulated_area)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/TrapezoidalPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4
Approximate value of integral from 0 to 2 is: 3.45673099542291
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/TrapezoidalPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 400
Approximate value of integral from 0 to 2 is: 3.39345293109042
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/TrapezoidalPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4000
Approximate value of integral from 0 to 2 is: 3.39344669218516
> 
```

Figure 46. Trapezoidal panel example using 4, 400, and 4000 panels..

The actual listing depicted in Figure 46 is shown in Listing 10. Observe (at least for this example) the method appears more accurate that the rectangular method for the same number of panels, however also observe we are making twice as many function calls.

Listing 10. R code demonstrating Trapezoidal Panel Numerical Integration.

```
# R script to implement trapezoidal panel numerical integration
################################ NOTE ##################################
## The interactive input requires the script to be sourced             #
# In R console the command line would be                               #
# source('PATH-TO-THE-FILE/TrapezoidalPanelExample.R')                 #
# where PATH-TO-THE-FILE is replaced with the actual path on your machine #
########################################################################
#             Function to be integrated (modify as needed)             #
########################################################################
y <- function(x){
  y <- x * sqrt(1+x^2)
  return(y)
}
########################################################################
#             Get lower,upper and how many panels from user            #
########################################################################
xlow <- readline("What is the lower limit of integration? ")
xhigh <- readline("What is the upper limit of integration? ")
howMany <- readline("How many panels? ")
# Convert the strings into numeric values
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
accumulated_area <- 0.0 # initialize the accumulator
xleft <- xlow
xright <- xleft + deltax
#######################################################################
#                     The actual numerical method                     #
#######################################################################
for (i in 1:howMany){
  yleft <- y(xleft)
  yright <- y(xright)
  accumulated_area <- accumulated_area + (yleft+yright)*deltax/2
  xleft <- xleft + deltax
  xright <- xright + deltax
}
#######################################################################
#                            Report Result                            #
#######################################################################
message("Approximate value of integral from ",xlow," to ",xhigh," is: ",accumulated_area)
```

3.1.3 Parabolic Panels

Parabolic panels approximate the shape of the panel with a parabola. The area between the chord and the curve (neglected in the trapezoidal solution) may be accounted for by approximating the function with a parabola passing through the points defined by three successive values of y.

This area may be calculated from the geometry of the parabola and added to the trapezoidal area of the pair of strips to give the area ΔA of the pair as illustrated. Adding all of the ΔAs produces the tabulation shown, which is known as Simpson's rule. To use Simpson's rule, the number n of strips must be even.

The same example as presented for rectangular panels is repeated, except using parabolic panels. The code is changed yet again because we will evaluate at each end of the panel as well as at an intermediate value.

Figure 48 is a screen capture of a parabolic panel integration. The actual script is also listed in Listing 11. In the script, I substituted $\frac{\Delta x}{2}$ for Δx from Figure 47, so the accumulation line has a 6 in the denominator (rather than the 3 in the figure).[10]

Observe that the estimated integral for 400 and 4000 panels is nearly the same,

[10] $\dots \frac{\Delta x}{2} \times \frac{1}{3} = \dots \frac{\Delta x}{6}$

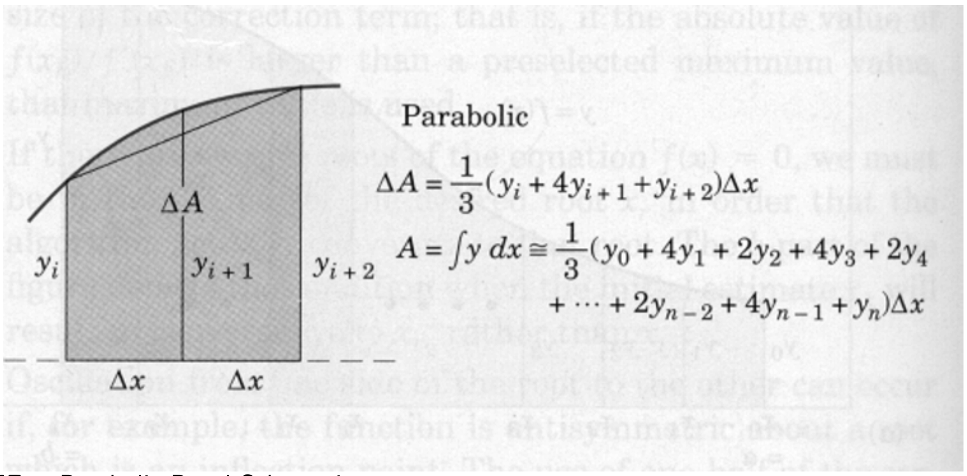
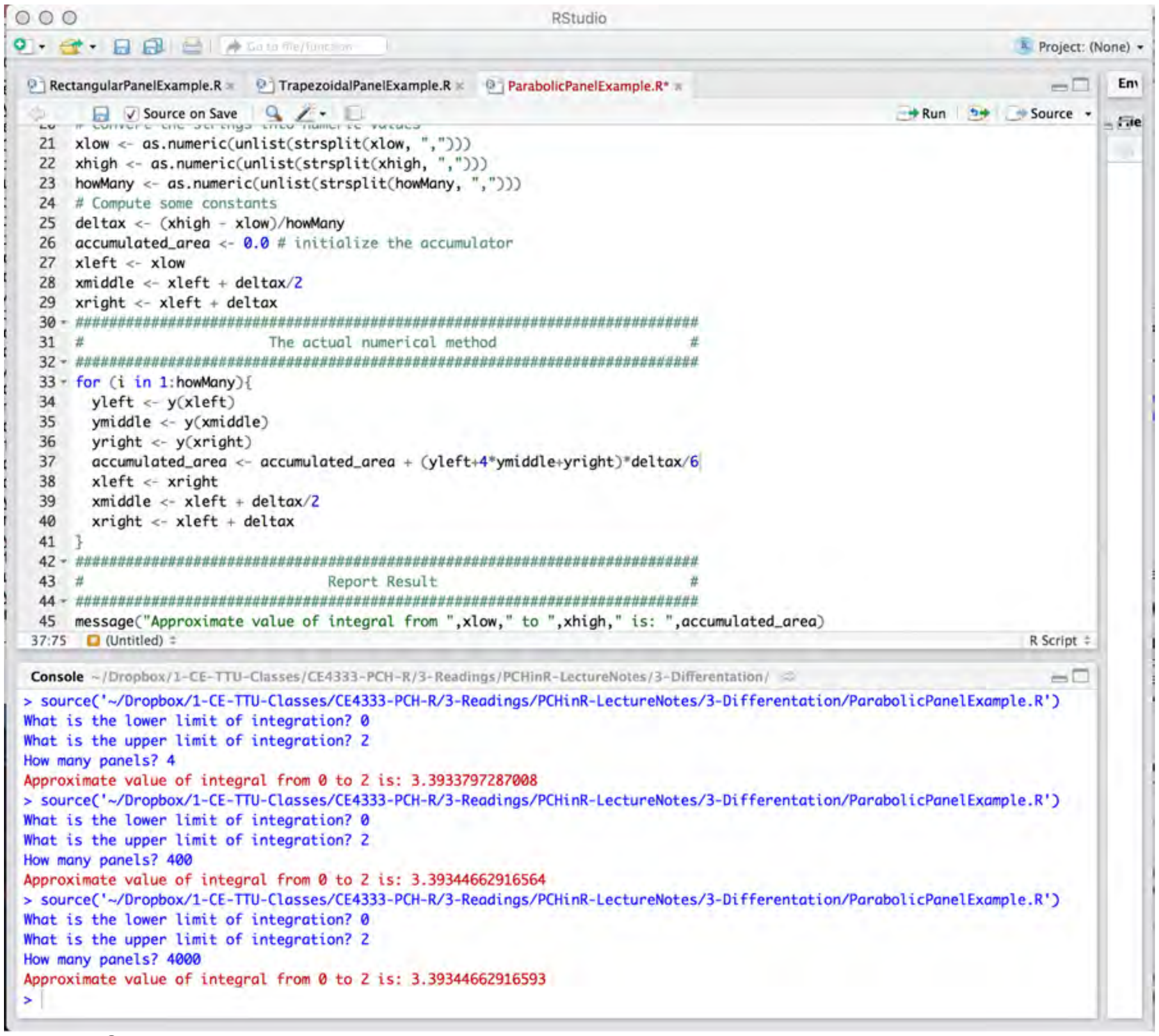

Parabolic
$\Delta A = \frac{1}{3}(y_i + 4y_{i+1} + y_{i+2})\Delta x$
$A = \int y\,dx \cong \frac{1}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n)\Delta x$
ΔA
y_i
y_{i+1}
y_{i+2}
Δx
Δx

Figure 47. Parabolic Panel Schematic..


RStudio
Project: (None)
RectangularPanelExample.R
TrapezoidalPanelExample.R
ParabolicPanelExample.R*
Source on Save
Run
Source
37:75
(Untitled)
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/

```r
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
accumulated_area <- 0.0 # initialize the accumulator
xleft <- xlow
xmiddle <- xleft + deltax/2
xright <- xleft + deltax
###########################################################################
#                      The actual numerical method                        #
###########################################################################
for (i in 1:howMany){
  yleft <- y(xleft)
  ymiddle <- y(xmiddle)
  yright <- y(xright)
  accumulated_area <- accumulated_area + (yleft+4*ymiddle+yright)*deltax/6
  xleft <- xright
  xmiddle <- xleft + deltax/2
  xright <- xleft + deltax
}
###########################################################################
#                             Report Result                               #
###########################################################################
message("Approximate value of integral from ",xlow," to ",xhigh," is: ",accumulated_area)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/ParabolicPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4
Approximate value of integral from 0 to 2 is: 3.3933797287008
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/ParabolicPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 400
Approximate value of integral from 0 to 2 is: 3.39344662916564
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/ParabolicPanelExample.R')
What is the lower limit of integration? 0
What is the upper limit of integration? 2
How many panels? 4000
Approximate value of integral from 0 to 2 is: 3.39344662916593
>
```

Figure 48. Parabolic panel example using 4, 400, and 4000 panels.

suggesting no need to go beyond a certain number of panels. Algorithms that detect when to stop adding panels exist and would be implemented in many scientific and engineering programming applications.

Listing 11. R code demonstrating Parabolic Panel Numerical Integration.

```
# R script to implement trapezoidal panel numerical integration
################################ NOTE ##################################
## The interactive input requires the script to be sourced              #
# In R console the command line would be                                #
# source('PATH-TO-THE-FILE/RectangularPanelExample.R')                  #
# where PATH-TO-THE-FILE is replaced with the actual path on your machine #
#########################################################################
#              Function to be integrated (modify as needed)             #
#########################################################################
y <- function(x){
  y <- x * sqrt(1+x^2)
  return(y)
}
#########################################################################
#              Get lower,upper and how many panels from user            #
#########################################################################
xlow <- readline("What is the lower limit of integration? ")
xhigh <- readline("What is the upper limit of integration? ")
howMany <- readline("How many panels? ")
# Convert the strings into numeric values
xlow <- as.numeric(unlist(strsplit(xlow, ",")))
xhigh <- as.numeric(unlist(strsplit(xhigh, ",")))
howMany <- as.numeric(unlist(strsplit(howMany, ",")))
# Compute some constants
deltax <- (xhigh - xlow)/howMany
accumulated_area <- 0.0 # initialize the accumulator
xleft <- xlow
xmiddle <- xleft + deltax/2
xright <- xleft + deltax
########################################################################
#                       The actual numerical method                    #
########################################################################
for (i in 1:howMany){
  yleft <- y(xleft)
  ymiddle <- y(xmiddle)
  yright <- y(xright)
  accumulated_area <- accumulated_area + (yleft+4*ymiddle+yright)*deltax/6
  xleft <- xright
  xmiddle <- xleft + deltax/2
  xright <- xleft + deltax
}
########################################################################
#                              Report Result                           #
########################################################################
message("Approximate value of integral from ",xlow," to ",xhigh," is: ",accumulated_area)
```

If we study all the forms of the numerical method we observe that the numerical integration method is really the sum of function values at specific locations in the interval of interest, with each value multiplied by a specific weight. In this development the weights were based on polynomials, but other method use different weighting functions. An extremely important method is called gaussian quadrature, which is outside the scope of the discussion herein — Gaussian quadrature routines are readily available within **R**. The method is valuable because one can approximate convolution integrals quite effectively using quadrature routines, while the number of function evaluations for a polynomial based approximation could become hopeless.

When the function values are tabular, we are going to have to accept the rectangular (with adaptations) and trapezoidal as our best tool to approximate an integral because we don't have any really effective way to evaluate the function between the tabulated values – if we were to use our interpolation routine from earlier, its really going to be a kind of trapezoidal rule anyway.

3.2 Exercise Set 2

1. Write a script to approximate $\int_{1.8}^{3.4} e^{x}dx$ using rectangular panels.
   Run your script using 6 and 600 panels.
   (a) What is the analytical solution (e.g. do the calculus!)?
   (b) What is the percent error between the analytical solution and the approximation using 6 panels?
   (c) What is the percent error between the analytical solution and the approximation using 600 panels?
2. Write a script to approximate $\int_{1.8}^{3.4} e^{x}dx$ using trapezoidal panels.
   Run your script using 6 and 600 panels.
   (a) What is the analytical solution (e.g. do the calculus!)?
   (b) What is the percent error between the analytical solution and the approximation using 6 panels?
   (c) What is the percent error between the analytical solution and the approximation using 600 panels?
3. Based on the previous two exercises, which method do you think is more accurate for a given panel count? Why (do you think so)?
4. Write a script to approximate $\int_{0}^{1} ln(x)dx$ using rectangular panels.
   Run your script using 6 and 600 panels.
   (a) What is the analytical solution (e.g. do the calculus!)?
   (b) What is the percent error between the analytical solution and the approximation using 6 panels?
   (c) What is the percent error between the analytical solution and the approximation using 600 panels?
5. Write a script to approximate $\int_{0}^{1} ln(x)dx$ using trapezoidal panels.
   Run your script using 6 and 600 panels.
   (a) Did you get an error message — why?

This exercise set is also located on the class server as `ES-2`

3.3 Numerical Integration of Tabular Data

This subsection is going to work with tabular data — different from function evaluation, but similar. To be really useful, we need to learn how to read data from a file — manually entering tabular data is really time consuming, error prone, and just plain idiotic.

So in this subsection we will first learn how to read data from a file into a list, then we can process the list as if it were a function and integrate its contents.

3.3.1 Reading from a file – open, read, close files

R can read from an ASCII file (or even an Excel .csv file) using a multitude of methods. Common methods are `read.table(...)`, `read.table(...)`, `read.table(...)`, `read.table(...)`, and `read.table(...)`.

One can also use primatives[11] to read individual rows in a file and process them.[12]

First, lets create a file named `MyFile.txt`. The extension is important so that we can examine the file with other tools (a text editor) and remember that it is an ASCII file. The contents of `MyFile.txt` are:

```
1 , 1
2 , 4
3 , 9
4 , 16
5 , 25
```

To read the contents into an **R** script we have to do the following:

1. Open a connection to the file — this is a concept common to all languages, it might be called something different, but the program needs to somehow know the location and name of the file.
2. Read the contents into an object — we have a lot of control on how this gets done, for the time being we won't exercise much control yet. When you do substantial programs, you will depend on the control of the reads (and writes).
3. Disconnect the file — this too is common to all languages. Its a really easy step to forget. Not a big deal if the program ends as planned but terrible if there is a error in the program and the connection is still open. Usually noting bad happens, but with an open connection it is possible for the file to get damaged. If that file represents millions of customers data, that's kind of a problem.

[11] Jargon to describe lower level tools within **R**

[12] We will use this approach later in the book – the interactive prompt and reads in the prior subsection are similar to this approach where input is read into a string, then the string is converted into the appropriate type of object (numeric or text).

The `read.table` class of functions handles all three of these steps for us, we do have to provide the filename and some information about the file structure. Later when we are doing network simulation and other hydraulic techniques, different parts of an input file will be read line-by-line and processed — for this task we will need to handle these three steps using primatives.

Figure 49 illustrates the process. The input file has 5 lines, these get read then echoed (printed) back to us. The actual script is pretty simple, notice how the `filepath` and `filename` character variables are defined, then pasted together to produce a full *absolute* file name.[13]

Listing 12 is a listing of the script used in Figure 49. The analyst should be able to deduce that the filenames could be read from user input using the prompting technique used in the earlier subsections, so if one is going to process a lot of similar files the explicit naming could be replaced with variable naming – it would probably be a good idea to confine the files to a reasonably memorable path.

Listing 12. R code demonstrating Reading from a File.

```
# R script to illustrate reading from a file using read.table
# The script is intentionally complicated to illustrate the
# three steps: open connection, read into object, close connection
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-
     Differentation/RScripts"
filename <- "MyFile.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=",") # comma seperated ASCII, No header
# Echo zz
print(zz)
```

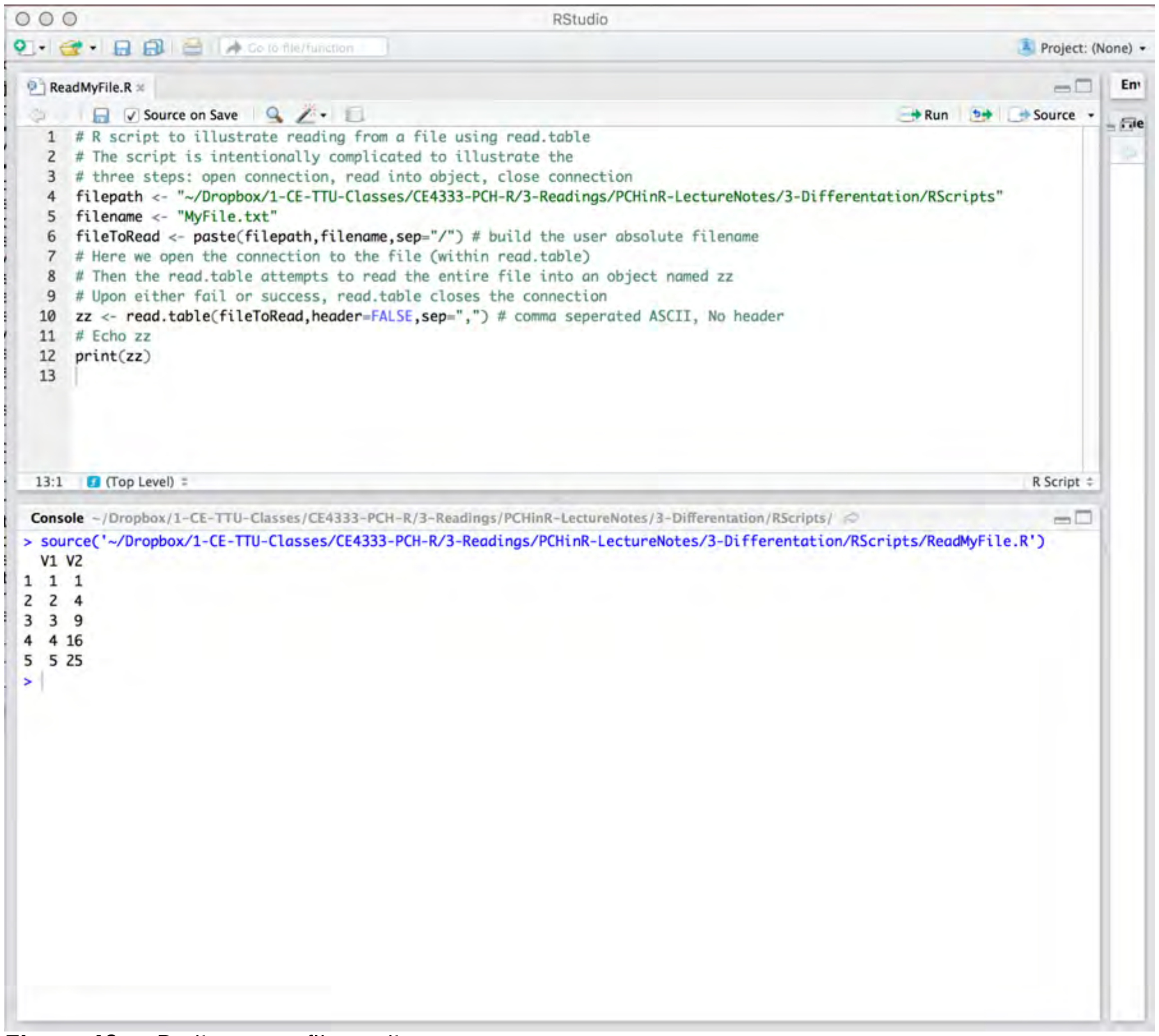
Now that we can read a file, we are now able to integrate tabular data.

3.4 Integrating tabular data

Suppose instead of a function we only have tabulations and wish to estimate the area under the curve represented by the tabular values. Then our integration rules from the prior sections still work more or less, except the rectangular panels will have to be shifted to either the left edge or right edge of a panel (where the tabulation exists).

Lets just examine an example. Suppose some measurement technology produced Table 1 a table of related values. The excitation variable is x and $f(x)$ is the response.

[13]A file name can specify all the directory names starting from the root of the tree; then it is called an absolute file name. Or it can specify the position of the file in the tree relative to a default directory; then it is called a relative file name. On the computer I used to write this workbook, the symbol ˜ , is the root to my user account, then the remaining directories from that location are explicitly listed. The actual absolute name is /Users/cleveland/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts



```
# R script to illustrate reading from a file using read.table
# The script is intentionally complicated to illustrate the
# three steps: open connection, read into object, close connection
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts"
filename <- "MyFile.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=",") # comma seperated ASCII, No header
# Echo zz
print(zz)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts/ReadMyFile.R')
  V1 V2
1  1  1
2  2  4
3  3  9
4  4 16
5  5 25
>
```

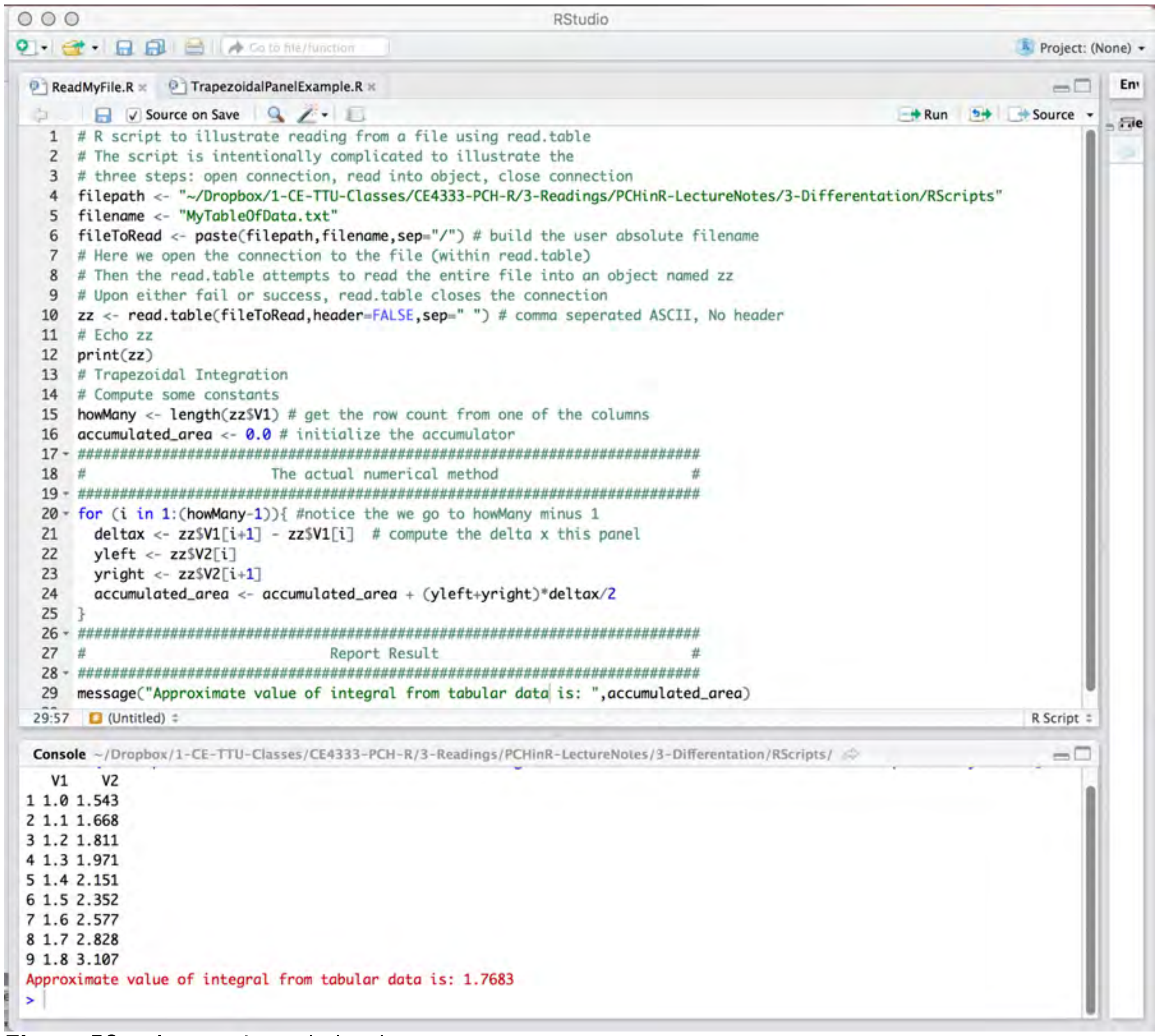
Figure 49. Rudimentary file reading.

Table 1. Tabular values of an excitation–response relationship.

x	$f(x)$
1.0	1.543
1.1	1.668
1.2	1.811
1.3	1.971
1.4	2.151
1.5	2.352
1.6	2.577
1.7	2.828
1.8	3.107

To integrate this table using the trapezoidal method is straightforward. We will modify our earlier code to read the table (which we put into a file), and compute the integral.

Figure 50 is a screen capture of a script that implements the file read and the numerical integration. The conversion of the method from the functional form in the previous section is pretty straightforward. The main nusiance here is the syntax required to access the "x" values and the "y" values. In **R** the most generic approach is `object$name`, where *object* is the data frame name, and *name* is the variable (column) name. If you don't use headers, **R** assigns names as `V1,V2,...,V`$_{max}$.



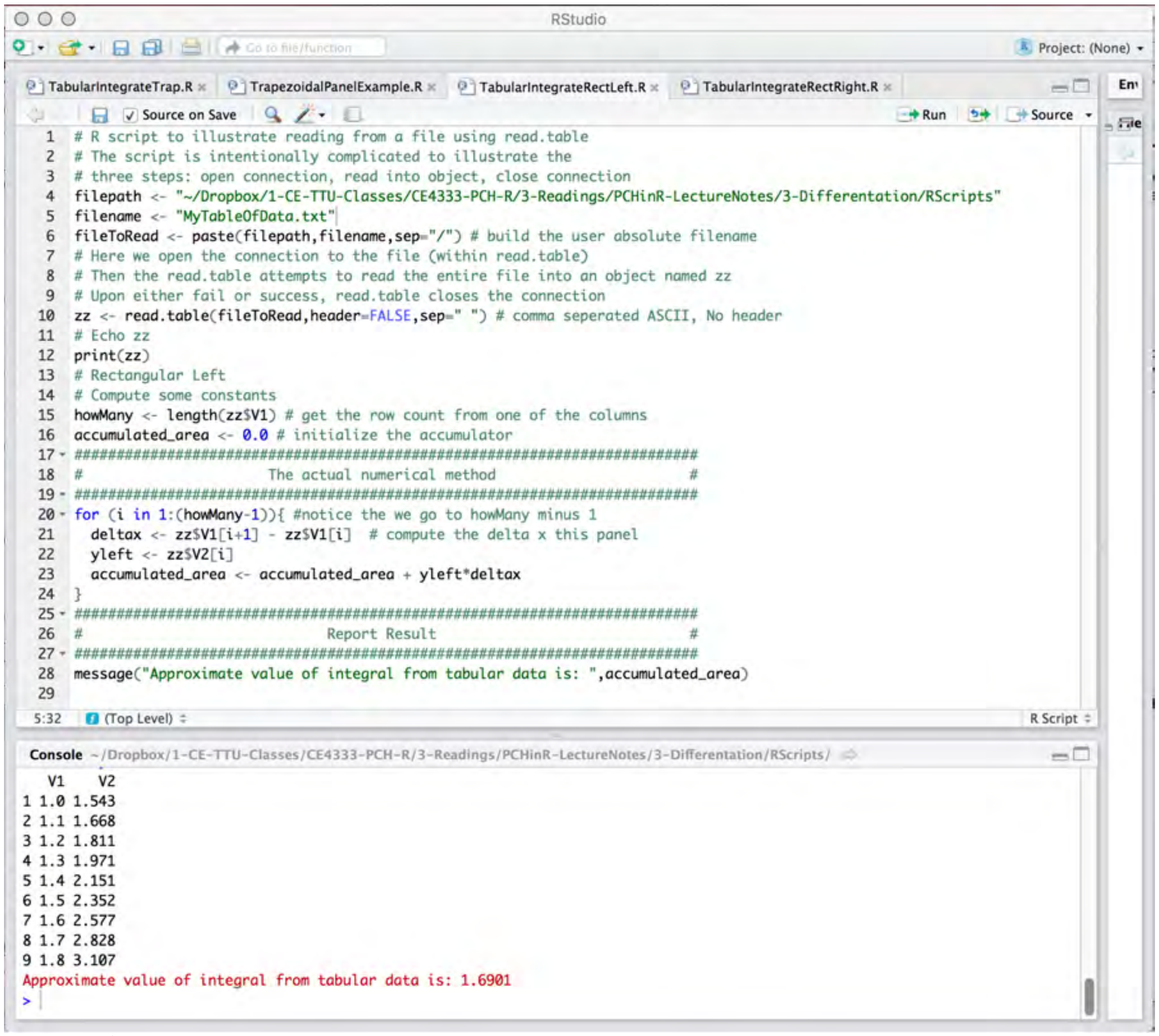
```
# R script to illustrate reading from a file using read.table
# The script is intentionally complicated to illustrate the
# three steps: open connection, read into object, close connection
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts"
filename <- "MyTableOfData.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=" ") # comma seperated ASCII, No header
# Echo zz
print(zz)
# Trapezoidal Integration
# Compute some constants
howMany <- length(zz$V1) # get the row count from one of the columns
accumulated_area <- 0.0 # initialize the accumulator
###########################################################################
#                        The actual numerical method                      #
###########################################################################
for (i in 1:(howMany-1)){ #notice the we go to howMany minus 1
  deltax <- zz$V1[i+1] - zz$V1[i]  # compute the delta x this panel
  yleft <- zz$V2[i]
  yright <- zz$V2[i+1]
  accumulated_area <- accumulated_area + (yleft+yright)*deltax/2
}
###########################################################################
#                               Report Result                             #
###########################################################################
message("Approximate value of integral from tabular data is: ",accumulated_area)
```

```
   V1    V2
1 1.0 1.543
2 1.1 1.668
3 1.2 1.811
4 1.3 1.971
5 1.4 2.151
6 1.5 2.352
7 1.6 2.577
8 1.7 2.828
9 1.8 3.107
Approximate value of integral from tabular data is: 1.7683
>
```

Figure 50. Integrating tabular data..

Realistically the only other simple integration method for tabular data is the rectangular rule, either using the left edge of a panel or the right edge of a panel (and you could do both and average the result which would result in the same outcome as the trapezoidal method). For the sake of completeness lets do both and then compare the results from all four approaches (trapezoidal, rectangular-left, rectangular-right, average rectangular).

First, Figure 51 implements the file read and tabular integration using the rectangular panel method, evaluating the function at the left edge of each panel.


RStudio
Project: (None)
TabularIntegrateTrap.R
TrapezoidalPanelExample.R
TabularIntegrateRectLeft.R
TabularIntegrateRectRight.R
Source on Save
Run
Source
5:32
(Top Level)
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts/

```r
# R script to illustrate reading from a file using read.table
# The script is intentionally complicated to illustrate the
# three steps: open connection, read into object, close connection
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts"
filename <- "MyTableOfData.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=" ") # comma seperated ASCII, No header
# Echo zz
print(zz)
# Rectangular Left
# Compute some constants
howMany <- length(zz$V1) # get the row count from one of the columns
accumulated_area <- 0.0 # initialize the accumulator
##########################################################################
#                    The actual numerical method                         #
##########################################################################
for (i in 1:(howMany-1)){ #notice the we go to howMany minus 1
  deltax <- zz$V1[i+1] - zz$V1[i]  # compute the delta x this panel
  yleft <- zz$V2[i]
  accumulated_area <- accumulated_area + yleft*deltax
}
##########################################################################
#                          Report Result                                 #
##########################################################################
message("Approximate value of integral from tabular data is: ",accumulated_area)
```

```
   V1    V2
1 1.0 1.543
2 1.1 1.668
3 1.2 1.811
4 1.3 1.971
5 1.4 2.151
6 1.5 2.352
7 1.6 2.577
8 1.7 2.828
9 1.8 3.107
Approximate value of integral from tabular data is: 1.6901
>
```

Figure 51. Integrating tabular data. Rectangular panel, evaluate at left edge..

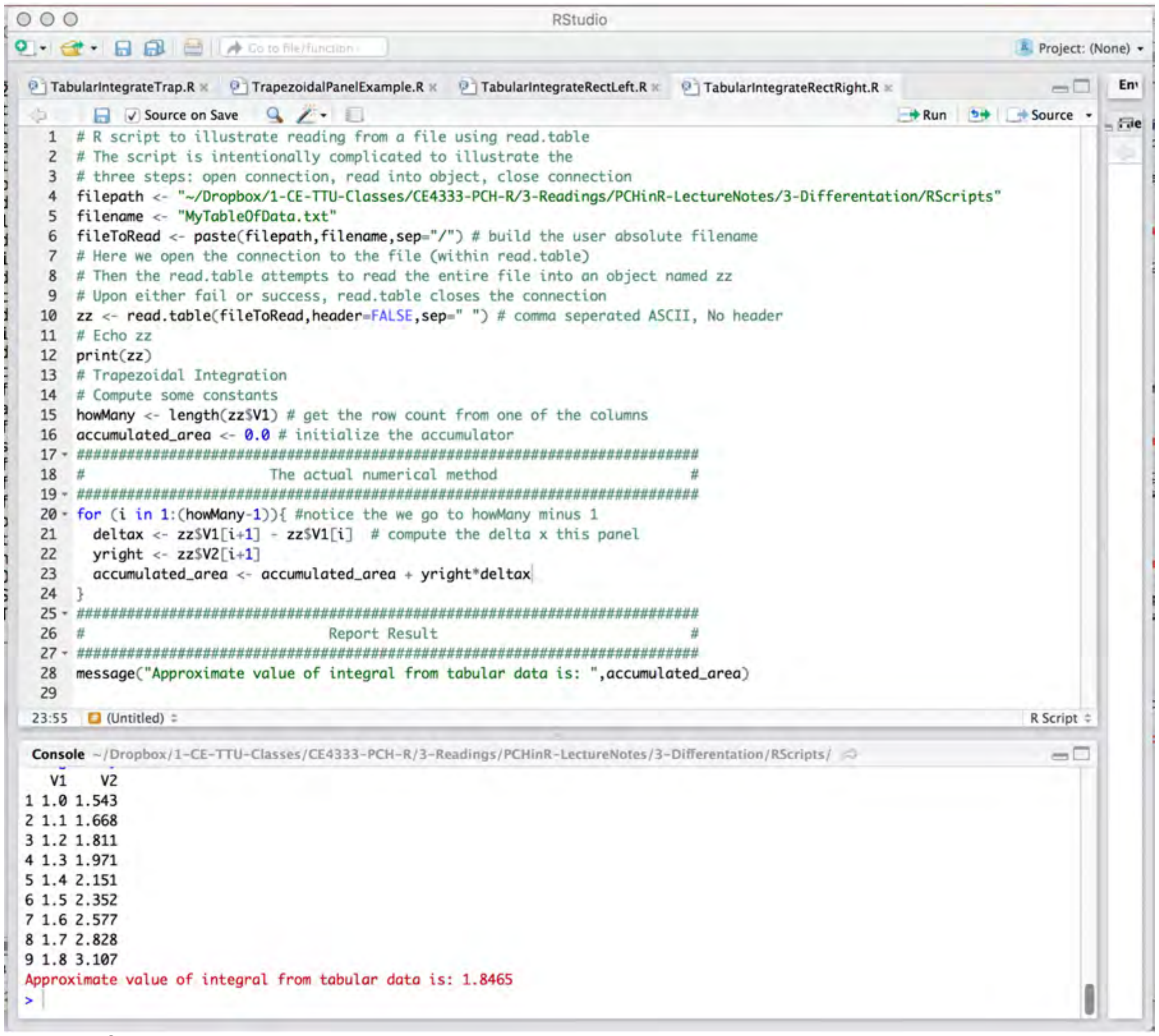
Next, Figure 52 implements the file read and tabular integration using the rectangular panel method, evaluating the function at the left edge of each panel.

Now lets compare the results from using the three (four) approaches. Table 2 are the results by method.

Table 2. Comparison of tabular integration.

Method	Computed Area
Trapezoidal Panels	1.7683
Rectangular - Left Edge	1.6901
Rectangular - Right Edge	1.8465
Arithmetic Mean Rectangular	1.7683

What Table 2 illustrates is that the trapezoidal rule is simply the average of the rectangular rule evaluated at first the left-edge then the right-edge of a panel.


RStudio
TabularIntegrateTrap.R
TrapezoidalPanelExample.R
TabularIntegrateRectLeft.R
TabularIntegrateRectRight.R
Source on Save
Run
Source
Project: (None)
23:55
(Untitled)
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts/

```r
# R script to illustrate reading from a file using read.table
# The script is intentionally complicated to illustrate the
# three steps: open connection, read into object, close connection
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts"
filename <- "MyTableOfData.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=" ") # comma seperated ASCII, No header
# Echo zz
print(zz)
# Trapezoidal Integration
# Compute some constants
howMany <- length(zz$V1) # get the row count from one of the columns
accumulated_area <- 0.0 # initialize the accumulator
##########################################################################
#                       The actual numerical method                      #
##########################################################################
for (i in 1:(howMany-1)){ #notice the we go to howMany minus 1
  deltax <- zz$V1[i+1] - zz$V1[i]  # compute the delta x this panel
  yright <- zz$V2[i+1]
  accumulated_area <- accumulated_area + yright*deltax
}
##########################################################################
#                              Report Result                             #
##########################################################################
message("Approximate value of integral from tabular data is: ",accumulated_area)

```

```
   V1    V2
1 1.0 1.543
2 1.1 1.668
3 1.2 1.811
4 1.3 1.971
5 1.4 2.151
6 1.5 2.352
7 1.6 2.577
8 1.7 2.828
9 1.8 3.107
Approximate value of integral from tabular data is: 1.8465
> 
```

Figure 52. Integrating tabular data. Rectangular panel, evaluate at right edge..

3.5 Numerical Differentiation

Similar in context to numerical integration is approximation of derivatives. If the functions are representable as functions, then differencing degenerates into the selection of an appropriate difference formula. If the function is tabular, the same decision is presented, but we have to pay additional attention to the quantity of observations available.

3.6 Difference Approximations for Tabulated Data

Here we will introduce differencing by an example. Suppose we want to convert a cumulative data series into an incremental data series. It is operationally related to numerical differentiation.

As an example (leading to an algorithm) consider the cumulative rainfall time series in Table 3.

We shall import this data into **R**, then plot the data, then construct a computational procedure to extract the incremental values from the cumulative values. To load the data into **R** we start the program and then read the contents of a data file that contains the data into **R**, then we will introduce the plotting tools in **R**. In addition to plotting, we will also learn about headers and attaching an object (which gives access to header names rather than the `object$name` structure.

Table 3. Cumulative Rainfall Time Series.

hours	cumulative_rain
0.0	0.00
0.5	1.06
1.0	2.99
1.5	4.80
2.0	4.80
2.5	4.80
3.0	4.80
3.5	4.80
4.0	4.80
4.5	4.80
5.0	4.80
5.5	4.80

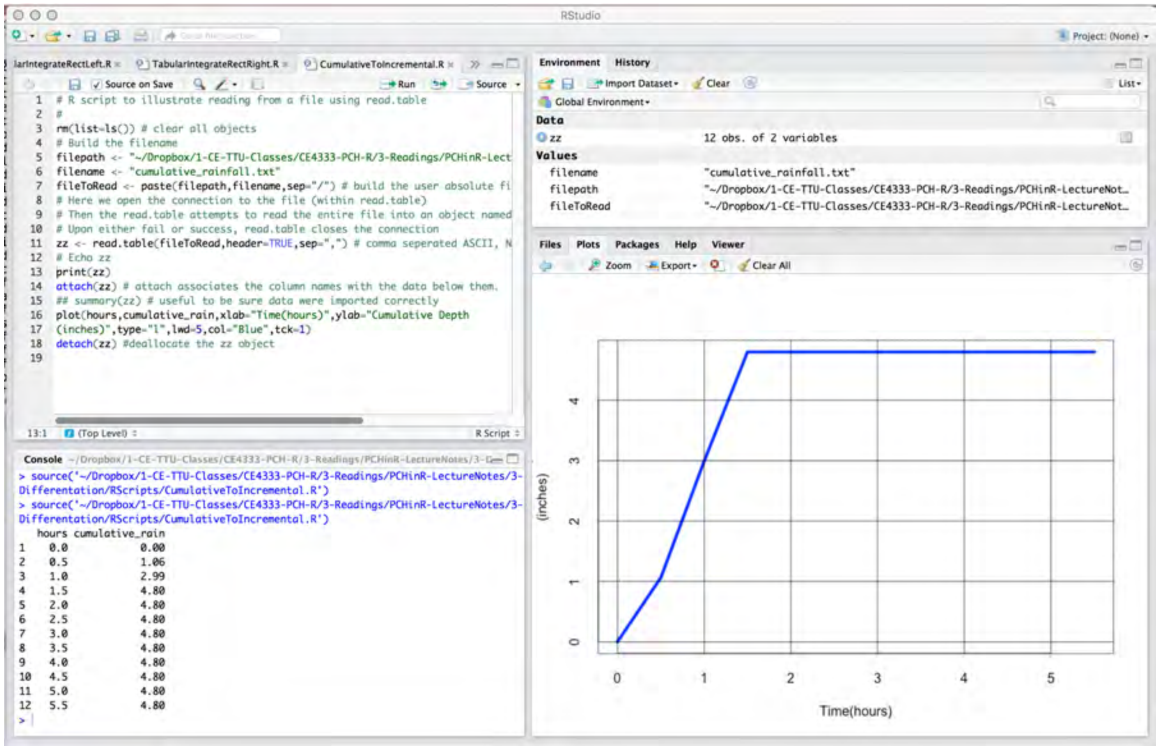
The plot suggests that the values are accumulated at the end of the time interval, thus the value accumulated is some average "rate" multiplied by the time interval. The line segment between each point is called a "secant" line. The slope of each secant line, provides that average "rate". So a fundamental computational step will be a function that computes the slope given any two points (assumed to be adjacent — so that's why sorting can become important, although the program doesn't actually care).

3.6.1 Slope of a Secant Line

`slopeOfSecant` is a prototype function that we write to that computes the slope of the secant line through two known points on a function $f(x_1)$ and $f(x_2)$. The function could be tabular or evaluated. The script assumes tabular in that the function is evaluates external to `slopeOfSecant`.

Listing 13. R code demonstrating the prototype function `slopeOfSecant`.

```
############## slope function prototype ####################
slopeOfSecant<-function(f1,f2,x1,x2){
  slopeOfSecant <- (f2-f1)/(x2-x1);
  return(slopeOfSecant)
}
#####################################################
```



```
# R script to illustrate reading from a file using read.table
#
rm(list=ls()) # clear all objects
# Build the filename
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-Lect
filename <- "cumulative_rainfall.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute fi
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=TRUE,sep=",") # comma seperated ASCII, N
# Echo zz
print(zz)
attach(zz) # attach associates the column names with the data below them.
## summary(zz) # useful to be sure data were imported correctly
plot(hours,cumulative_rain,xlab="Time(hours)",ylab="Cumulative Depth
(inches)",type="l",lwd=5,col="Blue",tck=1)
detach(zz) #deallocate the zz object
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-
Differentation/RScripts/CumulativeToIncremental.R')
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-
Differentation/RScripts/CumulativeToIncremental.R')
   hours cumulative_rain
1    0.0            0.00
2    0.5            1.06
3    1.0            2.99
4    1.5            4.80
5    2.0            4.80
6    2.5            4.80
7    3.0            4.80
8    3.5            4.80
9    4.0            4.80
10   4.5            4.80
11   5.0            4.80
12   5.5            4.80
>
```

```
Data
zz          12 obs. of 2 variables
Values
filename    "cumulative_rainfall.txt"
filepath    "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNot…
fileToRead  "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNot…
```

Figure 53. Plot of Cumulative Rainfall Time Series.

This slope is also a first order approximation of a derivative (forward, backward, and centered differences depending on values supplied). This function can then be used to compute "derivatives" of data series using a disaggregation function.

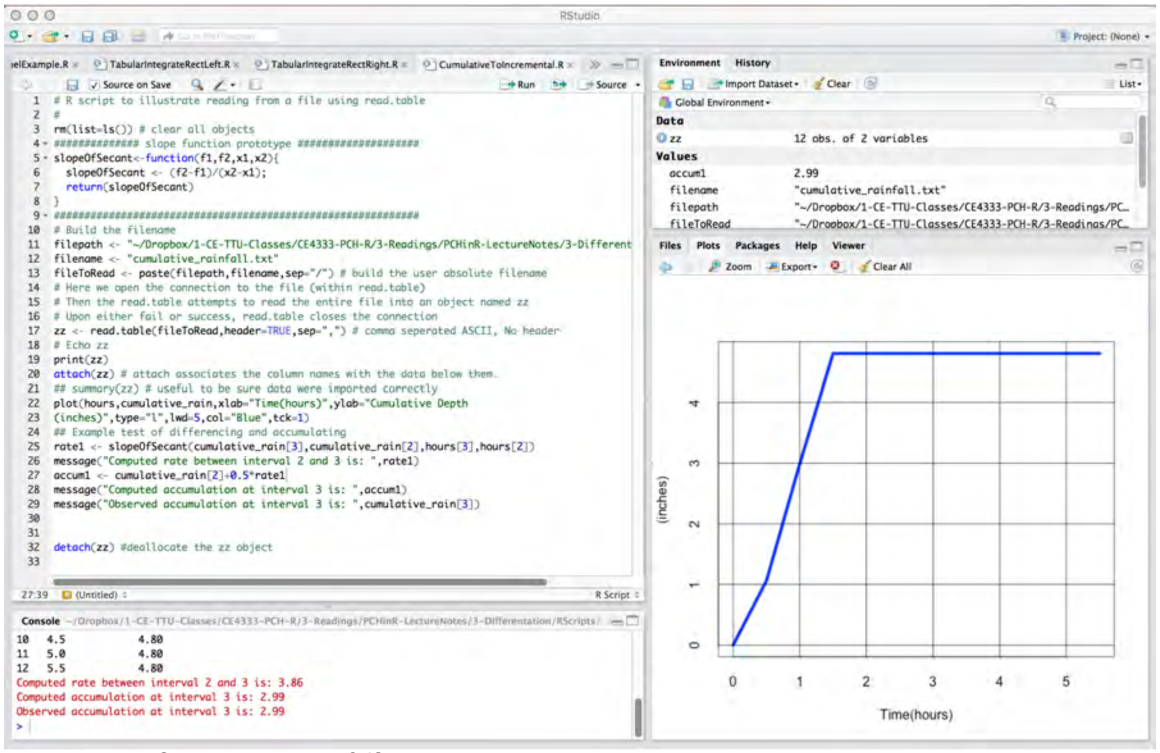
As an illustrative example, if we present parts of the cumulative rainfall data series we can recover the average rate between the inputs. Figure 54 is a screen capture of such a test.

3.6.2 Disaggregation

`disaggregate` is a prototype function that computes the slopes of the secant lines joining adjacent pairs of input data. Depending on the way the input arrays are presented to the `disaggregate` function, the function returns either the backward difference approximation to the function's derivative or if an index is presented instead of actual t values, then the function returns the incremental values that when aggregated reconstruct the original input function.

Listing 14. R code demonstrating the prototype function `disaggregate()`.

```
########      disaggregate function prototype     ###########
# returns a vector of slopes computed by sloepOfSecant
disaggregate<-function(f,x,dfdx){
  n<-length(x) # length of vectors
  dfdx<-rep(0,n); # zero dfdx
  for (i in 2:n){dfdx[i]<-slopeOfSecant(f[i-1],f[i],x[i-1],x[i]);};
  dfdx[1]<-0;
  return(dfdx)}
##############################################################
```



```
# R script to illustrate reading from a file using read.table
#
rm(list=ls()) # clear all objects
############### slope function prototype #####################
slopeOfSecant<-function(f1,f2,x1,x2){
  slopeOfSecant <- (f2-f1)/(x2-x1);
  return(slopeOfSecant)
}
##################################################################
# Build the filename
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Different
filename <- "cumulative_rainfall.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=TRUE,sep=",") # comma seperated ASCII, No header
# Echo zz
print(zz)
attach(zz) # attach associates the column names with the data below them.
## summary(zz) # useful to be sure data were imported correctly
plot(hours,cumulative_rain,xlab="Time(hours)",ylab="Cumulative Depth
(inches)",type="l",lwd=5,col="Blue",tck=1)
## Example test of differencing and accumulating
rate1 <- slopeOfSecant(cumulative_rain[3],cumulative_rain[2],hours[3],hours[2])
message("Computed rate between interval 2 and 3 is: ",rate1)
accum1 <- cumulative_rain[2]+0.5*rate1
message("Computed accumulation at interval 3 is: ",accum1)
message("Observed accumulation at interval 3 is: ",cumulative_rain[3])


detach(zz) #deallocate the zz object

```

```
10  4.5          4.80
11  5.0          4.80
12  5.5          4.80
Computed rate between interval 2 and 3 is: 3.86
Computed accumulation at interval 3 is: 2.99
Observed accumulation at interval 3 is: 2.99
>
```

```
Data
zz            12 obs. of 2 variables
Values
accum1        2.99
filename      "cumulative_rainfall.txt"
filepath      "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PC…
fileToRead    "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PC…
```

RStudio
TabularIntegrateRectLeft.R
TabularIntegrateRectRight.R
CumulativeToIncremental.R
Source on Save
Run
Source
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RScripts/
Project: (None)
Environment
History
Import Dataset
Clear
Global Environment
Files
Plots
Packages
Help
Viewer
Zoom
Export
Clear All
Time(hours)
(inches)
0
1
2
3
4
5

Figure 54. Script with slopeOfSecant prototype inserted and validation that we recover rate and can re-accumulate correctly.

3.6.3 Numerical Differentiation

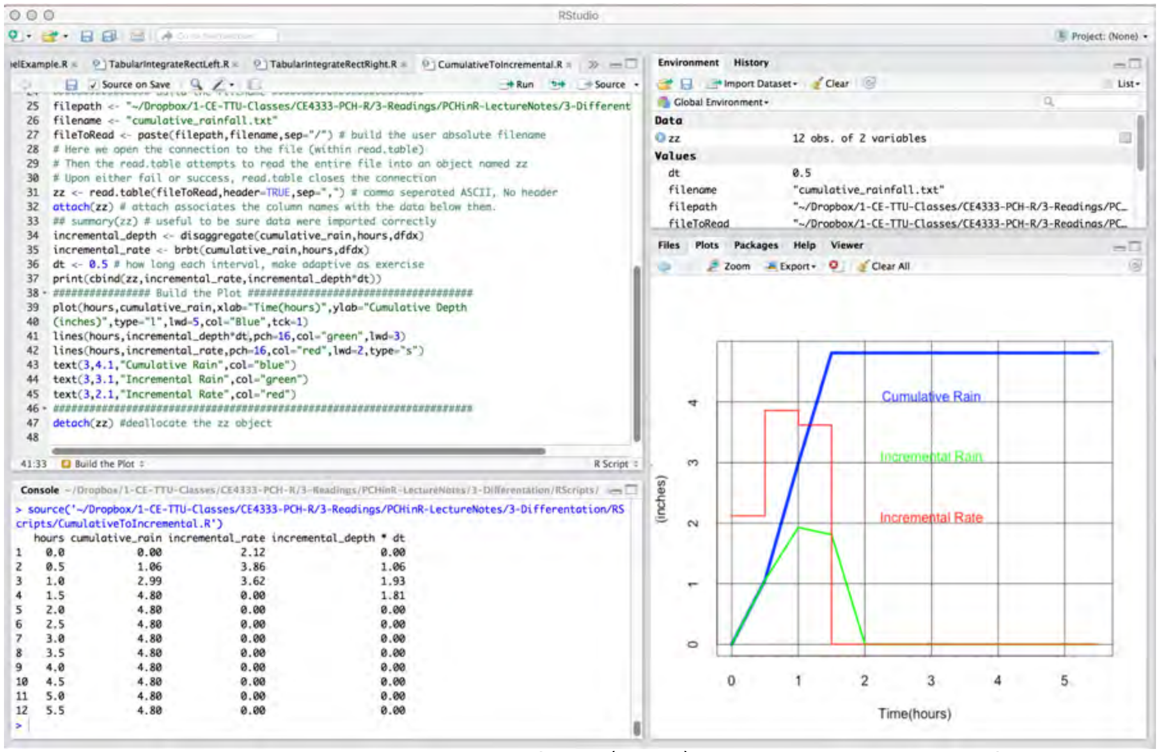
A related concept is to determine the average rate for the time interval, the principal difference is that the rate occurs during the entire time interval and should be assigned to the beginning of the interval instead of the end of the interval. A subtle change in the `disaggregate` function can accomplish the task, we will name that new function `brbt`. The name is a nemonic for "backward-rate, backward-time" differencing.

Listing 15. R code demonstrating the prototype function `brbt()`.

```
########### backward rate, backward time prototype ###########
brbt<-function(f,x,dfdx){
  n<-length(x) # length of vectors
  dfdx<-rep(0,n); # zero dfdx
  for (i in 1:(n-1)){dfdx[i]<-slopeOfSecant(f[i],f[i+1],x[i],x[i+1]);};
  dfdx[n]<-0;
  return(dfdx)}
##############################################################
```

Finally, putting everything together, we have the toolkit to determine the incremental rates (which is an approximation to the derivative of the cumulative rates) and incremental depths which are these individual rates multiplied by the length of the time interval. Figure 55 is a screen capture of the **R** script that implements these functions on the tabular data.

Listing 16 is a listing of the **R** script that produced Figure 55. The intermediate steps from Figure 54 is removed in this listing.



```
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Different
filename <- "cumulative_rainfall.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=TRUE,sep=",") # comma seperated ASCII, No header
attach(zz) # attach associates the column names with the data below them.
## summary(zz) # useful to be sure data were imported correctly
incremental_depth <- disaggregate(cumulative_rain,hours,dfdx)
incremental_rate <- brbt(cumulative_rain,hours,dfdx)
dt <- 0.5 # how long each interval, make adaptive as exercise
print(cbind(zz,incremental_rate,incremental_depth*dt))
################ Build the Plot ######################################
plot(hours,cumulative_rain,xlab="Time(hours)",ylab="Cumulative Depth
(inches)",type="l",lwd=5,col="Blue",tck=1)
lines(hours,incremental_depth*dt,pch=16,col="green",lwd=3)
lines(hours,incremental_rate,pch=16,col="red",lwd=2,type="s")
text(3,4.1,"Cumulative Rain",col="blue")
text(3,3.1,"Incremental Rain",col="green")
text(3,2.1,"Incremental Rate",col="red")
######################################################################
detach(zz) #deallocate the zz object
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-Differentation/RS
cripts/CumulativeToIncremental.R')
   hours cumulative_rain incremental_rate incremental_depth * dt
1    0.0            0.00             2.12                  0.00
2    0.5            1.06             3.86                  1.06
3    1.0            2.99             3.62                  1.93
4    1.5            4.80             0.00                  1.81
5    2.0            4.80             0.00                  0.00
6    2.5            4.80             0.00                  0.00
7    3.0            4.80             0.00                  0.00
8    3.5            4.80             0.00                  0.00
9    4.0            4.80             0.00                  0.00
10   4.5            4.80             0.00                  0.00
11   5.0            4.80             0.00                  0.00
12   5.5            4.80             0.00                  0.00
>
```

```
Data
zz              12 obs. of 2 variables
Values
dt              0.5
filename        "cumulative_rainfall.txt"
filepath        "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PC…
fileToRead      "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PC…
```

Figure 55. Plot of Cumulative Rainfall Time Series (BLUE), Incremental Depth Time Series (GREEN), and Average Rate Time Series (RED).

Listing 16. R code demonstrating Numerical Differencing.

```
# R script to illustrate numerical differencing
rm(list=ls()) # clear all objects
############# Prototype (forward define) Functions #########
############## slope function prototype ####################
slopeOfSecant<-function(f1,f2,x1,x2){
  slopeOfSecant <- (f2-f1)/(x2-x1);
  return(slopeOfSecant)}
########      disaggregate function prototype     ###########
disaggregate<-function(f,x,dfdx){
  n<-length(x) # length of vectors
  dfdx<-rep(0,n); # zero dfdx
  for (i in 2:n){dfdx[i]<-slopeOfSecant(f[i-1],f[i],x[i-1],x[i]);};
  dfdx[1]<-0;
  return(dfdx)}
########### backward rate, backward time prototype ###########
brbt<-function(f,x,dfdx){
  n<-length(x) # length of vectors
  dfdx<-rep(0,n); # zero dfdx
  for (i in 1:(n-1)){dfdx[i]<-slopeOfSecant(f[i],f[i+1],x[i],x[i+1]);};
  dfdx[n]<-0;
  return(dfdx)}
###############################################################

################ Build the filename ########################
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/3-
    Differentation/RScripts"
filename <- "cumulative_rainfall.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=TRUE,sep=",") # comma seperated ASCII, No header
attach(zz) # attach associates the column names with the data below them.
## summary(zz) # useful to be sure data were imported correctly
incremental_depth <- disaggregate(cumulative_rain,hours,dfdx)
incremental_rate <- brbt(cumulative_rain,hours,dfdx)
dt <- 0.5 # how long each interval, make adaptive as exercise
incremental_depth <- incremental_depth*dt
print(cbind(zz,incremental_rate,incremental_depth))
################ Build the Plot #####################################
plot(hours,cumulative_rain,xlab="Time(hours)",ylab="Cumulative Depth
(inches)",type="l",lwd=5,col="Blue",tck=1)
```

```
lines(hours,incremental_depth*dt,pch=16,col="green",lwd=3)
lines(hours,incremental_rate,pch=16,col="red",lwd=2,type="s")
text(3,4.1,"Cumulative Rain",col="blue")
text(3,3.1,"Incremental Rain",col="green")
text(3,2.1,"Incremental Rate",col="red")
####################################################################
detach(zz) #deallocate the zz object
```

3.6.4 Aggregation

Aggregation is the compliment of disaggregation; instead of finding differences we are trying to produce cumulatives from incremental values or rates. Aggregation is to integration as disaggregation is to differentiation. Simple aggregation functions are straightforward to build. Numerical integration (already introduced) is a bit more challenging because there are many different ways to compute areas from tabular data – we will illustrated rectangular, trapezoidal, and parabolic panels.

We can insert a prototype `aggregate` function that simply adds elements in a series to prior elements and stores the value in another series. Another name for this kind of arithmetic is a running sum. Functionally, it is rectangular panel (evaluate from the left), numerical integration.

Listing 17. R code demonstrating the prototype function `aggregate()`.

```
########### aggregate function prototype ###########
aggregate<-function(vector1,vector2){
n<-length(vector1)
# fill vector2 with zeros
vector2<-rep(0,n)
vector2[1]<-vector1[1]+0.0
for(i in 2:n)vector2[i]<-vector2[i-1]+vector1[i]
return(vector2)}
##############################################
```

We add this function to the prototype list ate the top of the script and can run To illustrate the use of `aggregate` we will aggregate the incremental depths into the cumulative rainfall – we should recover the original cumulative rainfall series that was originally supplied.

3.7 Exercises

1. Add the `aggregate` prototype function to collection of prototype functions in the script. The add some code like:

   ```
   new_cum_rain<-aggregate(incremental_depth,dummy)
   plot(hours,cumulative_rain,xlab="Time(hours)",ylab="Cumulative Depth
   (inches)",type="l",lwd=5,col="Blue",tck=1)
   lines(hours,new_cum_rain,col="red",lwd=1.5)
   ```

   Demonstrate that the two series are identical (e.g. plotting on top of one another).

2. (Advanced) Modify the `disaggregate()` prototype function to automatically determine the time spacing (x) and perform the correct multiplication within the function to return the correct increments. You only have to add one line of code to the prototype function at

```
for (i in 2:n){dfdx[i]<-slopeOfSecant(f[i-1],f[i],x[i-1],x[i]);
deltax <-   # you need to define this in terms of x[i] and x[i-1]!
dfdx[i]<- deltax*dfdx[i];};
```

3.8 Finite-Difference Formulas

What we have just done is to explore the use of finite-difference approximations for derivatives. Some common formulas for difference formulas are listed below (without derivation – you should be able to find explanation in any numerical methods text). All the difference equations presented here are the result of truncated Taylor series expansions about x. The "order" refers to the magnitude of truncation error, and this magnitude is proportional to the step size (Δx) raised to a power (the order). Truncation error decreases as the step size is decreased, but one is approaching a divide-by-zero situation (because numerical methods don't do limits just yet!).

3.8.1 First Derivatives

Equation 8 is a first-order backwards difference.

$$\frac{df}{dx} \approx \frac{f(x) - f(x - \Delta x)}{\Delta x} \tag{8}$$

Equation 9 is a first-order backwards difference.

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x} \tag{9}$$

Equation 10 is a second-order central difference.

$$\frac{df}{dx} \approx \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} \tag{10}$$

3.8.2 Second Derivatives

Equation 11 is a second-order central difference.

$$\frac{d^2 f}{dx^2} \approx \frac{f(x + \Delta x) - 2f(x) + f(x - \Delta x)}{\Delta x^2} \tag{11}$$

3.8.3 Third Derivatives

Equation 12 is a sixth-order central difference.

$$\frac{d^3f}{dx^3} \approx \frac{f(x+2\Delta x) - 2f(x+\Delta x) + 2f(x-\Delta x) + f(x-2\Delta x)}{2\Delta x^3} \tag{12}$$

The procedure to generate such difference formulas is general and can supply estimates with approximations of any degree. The accuracy depends on the location and number of field variable values involved in the approximation. The selection of a formula is not at all trivial (especially with tabulations), but beyond the scope of this handbook.

Despite known complications, this is a general tool used in computational hydraulics and we will use it throughout the remainder of the handbook – in some examples it will not be obvious that it is finite differencing, and in others it will be explicitly obvious. The next section introduces Newton's method, and finite-differences will be used to approximate the derivative (Quasi-Newton) to implement the method. The procedure is really quite common and imbedded in a lot of the computational tools we use professionnaly.

3.9 Single Variable Quasi-Newton Methods

The application of fundamental principles of modeling and mechanics often leads to an algebraic or transcendental equation that cannot be easily solved and represented in a closed form. In these cases a numerical method is required to obtain an estimate of the root or roots of the expression.

Newton's method is an iterative technique that can produce good estimates of solutions to such equations. The method is employed by rewriting the equation in the form $f(x) = 0$, then successively manipulating guesses for x until the function evaluates to a value close enough to zero for the modeler to accept.

Figure 56 is a graph of some function whose intercept with the $x-$axis is unknown. The goal of Newton's method is to find this intersection (root) from a realistic first guess. Suppose the first guess is x_1, shown on the figure as the right-most specific value of x. The value of the function at this location is $f(x_1)$. Because x_1 is supposed to be a root the difference from the value zero represents an error in the estimate. Newton's method simply provides a recipe for corrections to this error.

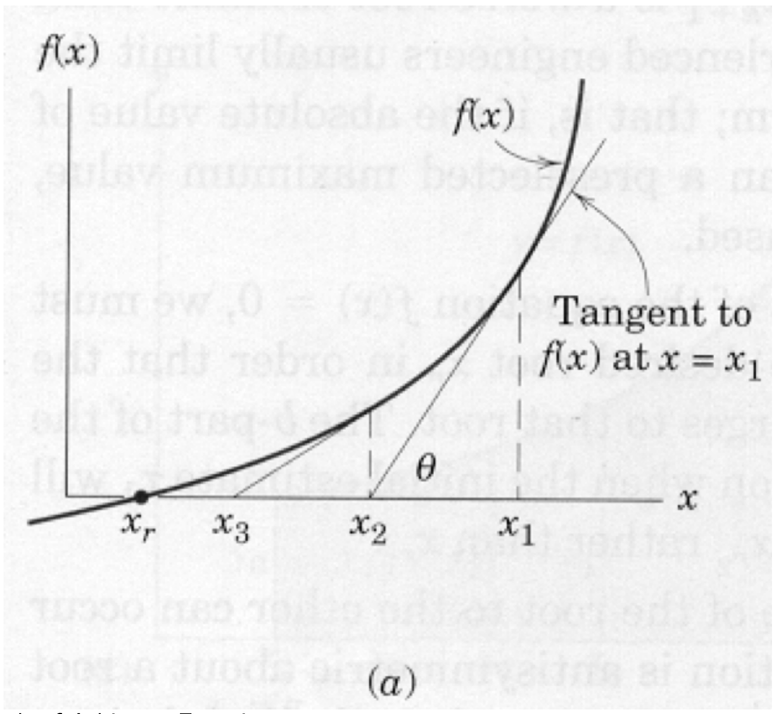


Figure 56. Graph of Arbitrary Function..

Provided x_1 is not near a minimum or maximum (slope of the function is not zero) then a better estimate of the root can be obtained by extending a tangent line from $x_1, f(x_1)$ to the x-axis. The intersection of this line with the axis represents a better estimate of the root.

This new estimate is x_2. A formula for x_2 can be derived from the geometry of the triangle x_2,$f(x_1)$,x_1. Recall from calculus that the tangent to a function at a particular point is the first derivative of the function. Therefore, from the geometry of the triangle and the definition of tangent we can write,

$$tan(\theta) = \frac{df}{dx}\bigg|_{x_1} = \frac{f(x_1)}{x_1 - x_2} \tag{13}$$

Solving the equation for x_2 results in a formula that expresses x_2 in terms of the first guess plus a correction term.

$$x_2 = x_1 - \frac{f(x_1)}{\frac{df}{dx}|_{x_1}} \tag{14}$$

The second term on the right hand side is the correction term to the estimate on the right hand side. Once x_2 is calculated we can repeat the formula substituting x_2 for x_1 and x_3 for x_2 in the formula. Repeated application usually leads to one of three outcomes:

1. a root;
2. divergence to $\pm\infty$; or
3. cycling.

These three outcomes are discussed below in various subsections along with some remedies.

The generalized formula is

$$x_{k+1} = x_k - \frac{f(x_k)}{\frac{df}{dx}|_{x_k}} \tag{15}$$

If the derivative is evaluated using analytical derivatives the method is called Newton's method, if approximations to the derivative are used, it is called a quasi-Newton method.

3.9.1 Newton's Method — Using analytical derivatives

This subsection presents an example in **R** of implementing Newton's method with analytical derivatives. The algorithm itself is:

1. Write the function in proper form, and code it into a computer.
2. Write the derivative in proper form and code it into a computer.
3. Make an initial guess of the solution (0 and 1 are always convenient guesses).
4. Evaluate the function, evaluate the derivative, calculate their ratio.
5. Subtract the ratio from the current guess and save the result as the update.

6. Test for stopping:
   (a) Did the update stay the same value? Yes, then stop, probably have a solution.
   (b) Is the function nearly zero? Yes, then stop we probably have a solution.
   (c) Have we tried too many updates? Yes, then stop the process is probably cycling, stop.
7. If stopping is indicated proceed to next step, otherwise proceed back to step 4.
8. Stopping indicated, report last update as the result (or report failure to find solution), and related information about the status of the numerical method.

The following example illustrates these step as well as a **R** implementation of Newton's method.

Suppose we wish to find a root (value of x) that satisfies Equation 16.

$$f(x) = e^x - 10cos(x) - 100 \tag{16}$$

Then we will need to code it into a script. Here is a code fragment that will work:

Listing 18. R code fragment for the function calculation.

```
# Define Function Here
func <- function(x)
{
  func <- exp(x)-10*cos(x)-100;
  return(func);
}
```

The next step is to code the derivative. In this case, Equation 17 is the derivative of Equation 16.

$$\frac{df}{dx}|(x) = e^x + 10\sin(x) \tag{17}$$

A code fragment to compute the value of the derivative at any value of x that will work is:

Listing 19. R code fragment for the derivative calculation.

```
# Define Derivative Here
dfdx <- function(x)
{
  dfdx <- exp(x) + 10*sin(x);
  return(dfdx);
}
```

Next we will need script to read in an initial guess, and ask us how many trials we will use to try to find a solution, as well as how close to zero we should be before we declare victory.

Listing 20. R code fragment for reading input data from the programmer.

```
# Read some values from the console
message('Enter an initial guess for X for Newton method :  ')
xnow <- as.numeric(readline())
message('Enter iteration maximum :  ')
HowMany <- as.numeric(readline())
message('Enter a tolerance value for stopping (e.g. 1e-06) :  ')
HowSmall <- as.numeric(readline())
## There are several other ways to make these reads!  The scan() function would probably
    also work.
```

The use of `HowSmall`; is called a zero tolerance. We will use the same numerical value for two tolerance tests. Also notice how we are using error traps to force numeric input. Probably overkill for this example, but we already wrote the code in an earlier chapter, so might as well use the code. Professional codes do a lot of error checking before launching into the actual processing — especially of the processing part is time consuming, its worth the time to check for obvious errors before running far a few hours then at some point failing because of an input value error that was predictable.

Now back to the tolerance tests. The first test is to determine if the update has changed or not. If it has not, we may not have a correct answer, but there is no point continuing because the update is unlikely to move further. The test is something like

IF $|x_{k+1} - x_k| <$ Tol. THEN Exit and Report Results

The second test is if the function value is close to zero. The structure of the test is similar, just an different argument. The second test is something like

IF $|f(x_{k+1})| <$ Tol. THEN Exit and Report Results

One can see from the nature of the two tests that a programmer might want to make the tolerance values different. This modification is left as a reader exercise.

Checking for maximum iterations is relatively easy, we just include code that checks for normal exit the loop.[14]

Now we simply connect the three fragments, and we have a working **R** script that implements Newton's method for Equation 16. Listing 21 is the entire code module that implements the method, makes the various tests, and reports results. Figure 57 is a screen capture of the program run in **R**.

The example is specific to the particular function provided, but the programmer could move the two functions `func` and `dfdx` into a user specified module, and then load that module in the program to make it even more generic. The next section will use such an approach to illustrate the ability to build a generalized Newton method and only have to program the function itself.

[14]Rather than breaking from the loop.

Listing 21. R code demonstrating Newton's Method calculations.

```
# Newtons Method in R
# Define Function Here
func <- function(x)
{
  func <- exp(x)-10*cos(x)-100;
  return(func);
}
# Define Derivative Here
dfdx <- function(x)
{
  dfdx <- exp(x) + 10*sin(x);
  return(dfdx);
}
# Newton's Method Here
# Read some values from the console
message('Enter an initial guess for X for Newton method :  ')
xnow <- as.numeric(readline())
message('Enter iteration maximum :  ')
HowMany <- as.numeric(readline())
message('Enter a tolerance value for stopping (e.g. 1e-06) :  ')
HowSmall <- as.numeric(readline())
# Now start the iterations
for (i in 1:HowMany) {
  xnew <- xnow - func(xnow)/dfdx(xnow)
# test for stopping
  if (abs(xnew-xnow) < HowSmall){
    message('Update not changing')
    xnow <- xnew
    print(cbind(xnow,xnew,func(xnew)))
    break
  }
  if (abs(func(xnew) < HowSmall)) {
    message('Function value close to zero')
    xnow <- xnew
    print(cbind(xnow,xnew,func(xnew)))
    break
  }
# next iteration
xnow <- xnew
}
if (i >= HowMany){
  message('Iteration limit reached')
  print(cbind(xnow,xnew,func(xnew)))
}
```

3.9.2 Newton's Method — Using Finite-Differences to estimate derivatives

A practical difficulty in using Newton's method is determining the value of the derivative in cases where differentiation is difficult. In these cases we can replace the derivative by a difference equation and then proceed as in Newton's method.

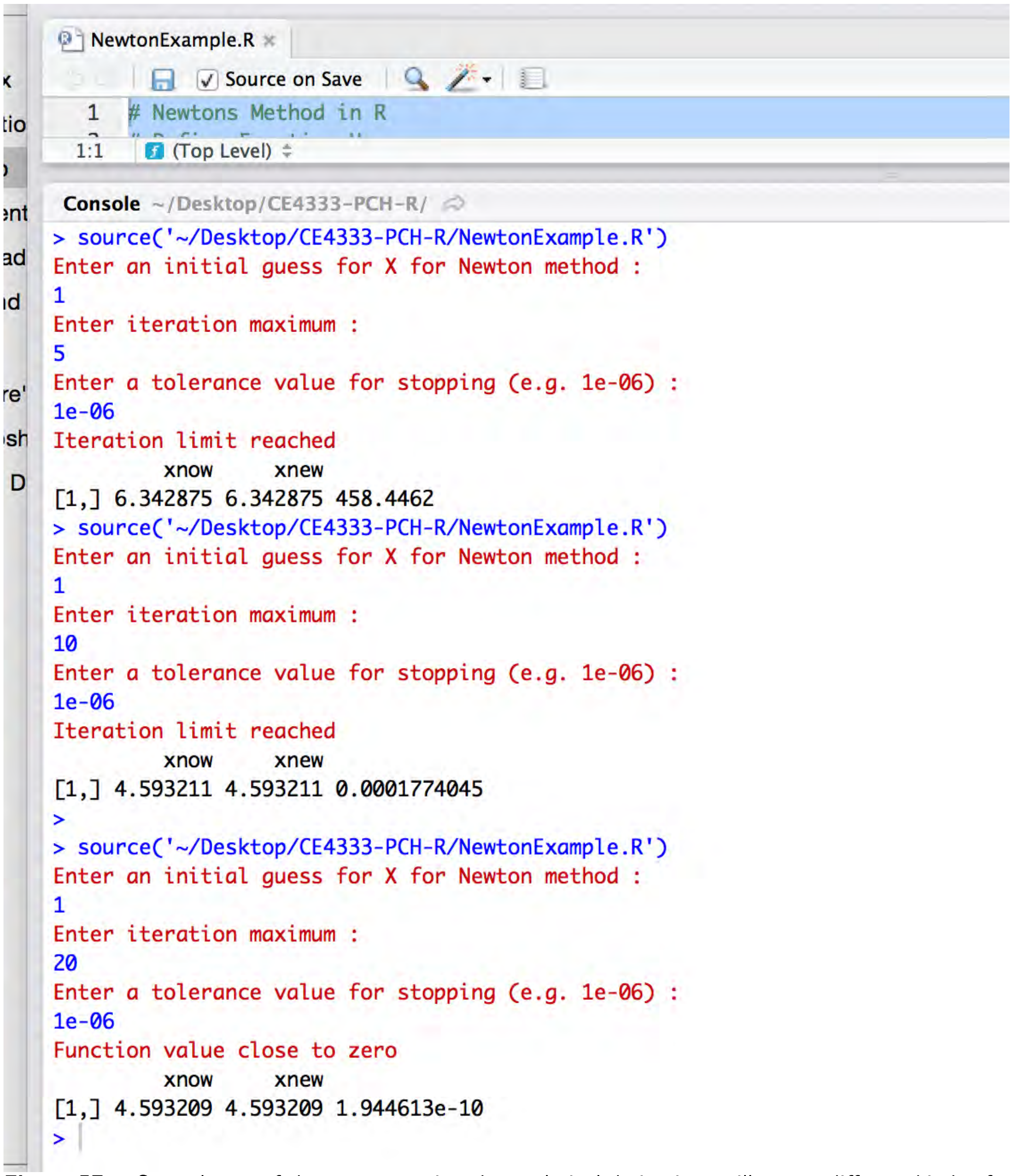
Recall from calculus that the derivative was defined as the limit of the difference quotient:

$$\frac{df}{dx}|_x = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{18}$$

A good approximation to the derivative should be possible by using this formula with a small, but non-zero value for Δx.

$$\frac{df}{dx}|_x \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{19}$$

When one replaces the derivative with the difference formula the root finding method the resulting update formula is



```
> source('~/Desktop/CE4333-PCH-R/NewtonExample.R')
Enter an initial guess for X for Newton method :
1
Enter iteration maximum :
5
Enter a tolerance value for stopping (e.g. 1e-06) :
1e-06
Iteration limit reached
         xnow     xnew
[1,] 6.342875 6.342875 458.4462
> source('~/Desktop/CE4333-PCH-R/NewtonExample.R')
Enter an initial guess for X for Newton method :
1
Enter iteration maximum :
10
Enter a tolerance value for stopping (e.g. 1e-06) :
1e-06
Iteration limit reached
         xnow     xnew
[1,] 4.593211 4.593211 0.0001774045
>
> source('~/Desktop/CE4333-PCH-R/NewtonExample.R')
Enter an initial guess for X for Newton method :
1
Enter iteration maximum :
20
Enter a tolerance value for stopping (e.g. 1e-06) :
1e-06
Function value close to zero
         xnow     xnew
[1,] 4.593209 4.593209 1.944613e-10
>
```

Figure 57. Several runs of the program using the analytical derivative to illustrate different kinds of responses..

$$x_{k+1} = x_k - \frac{f(x_k)\Delta x}{f(x_k + \Delta x) - f(x_k)} \tag{20}$$

This root-finding method is called a quasi-Newton method.

Listing 22 is the code fragment that we change by commenting out the analytical derivative and replacing it with a first-order finite difference approximation of the derivative. The numerical value $1e-06$ is called the step size (Δx) and should be an input value (rather than built-in to the code as shown here) like the tolerance test values, and be passed to the function as another argument.

Listing 22. R code demonstrating Newton's Method calculations.

```
# Define Derivative Here
dfdx <- function(x)
{
#  dfdx <- exp(x) + 10*sin(x);
    dfdx <- (func(x + 1e-06) - func(x) )/ (1e-06);
# func must already exist before first call!
  return(dfdx);
}
```

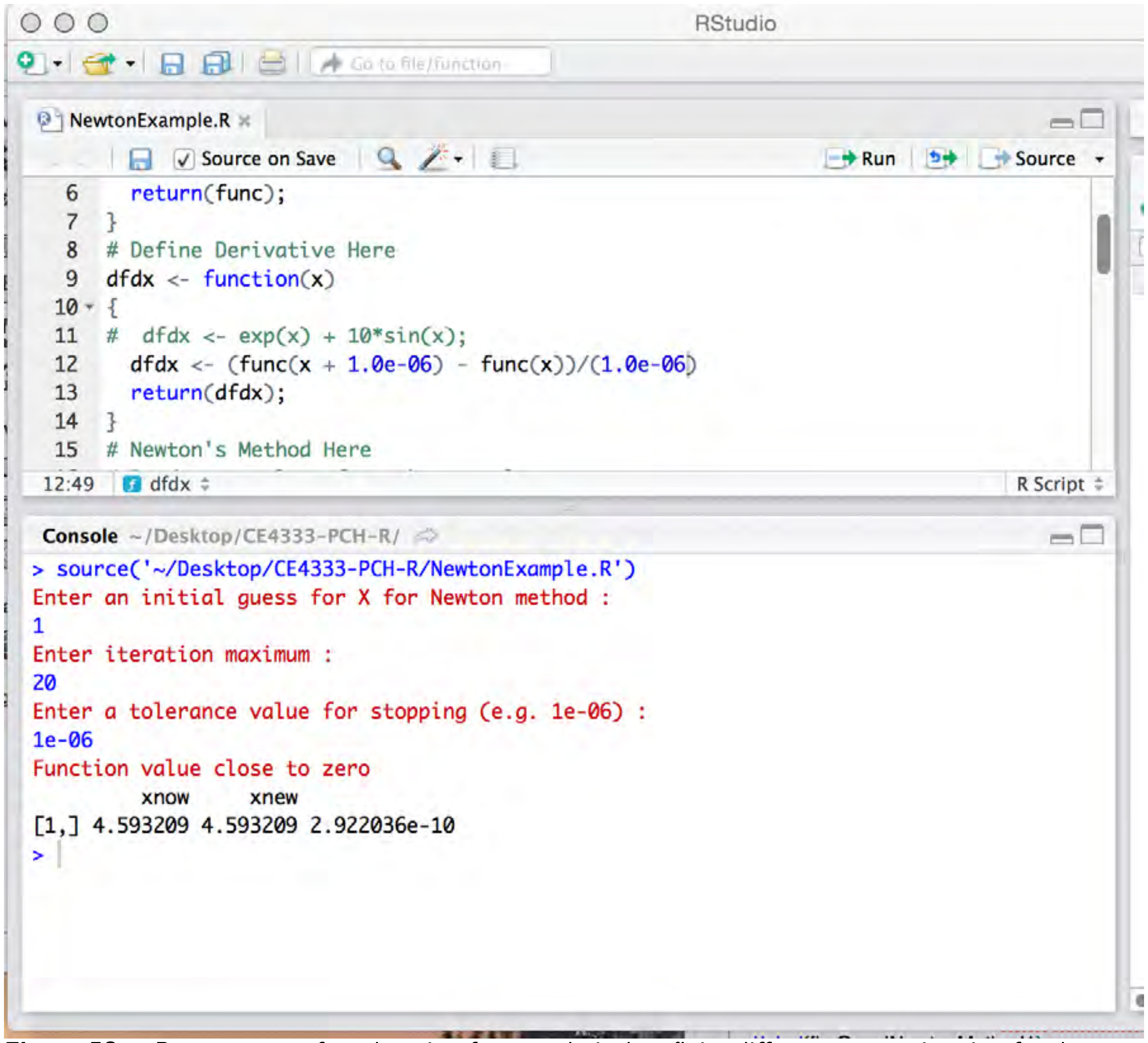
Starting with the last example lets modify the analytical version of the code by inserting the above fragment in place of the analytical derivative. Listing 23 is the listing with the modification in place. Notice we have only changed a single line, and not have a more flexible tool. The next modification (left as an exercise) is to detach the creation of the function from the main algorithm, then we would have a general purpose Quasi-Newton's method.

Listing 23. R code demonstrating Newton's Method calculations using finite-difference approximation for the derivative.

```
# Newtons Method in R
# Define Function Here
func <- function(x)
{
  func <- exp(x)-10*cos(x)-100;
  return(func);
}
# Define Derivative Here
dfdx <- function(x)
{
#  dfdx <- exp(x) + 10*sin(x);
  dfdx <- (func(x + 1.0e-06) - func(x))/(1.0e-06)
  return(dfdx);
}
# Newton's Method Here
# Read some values from the console
message('Enter an initial guess for X for Newton method :  ')
xnow <- as.numeric(readline())
message('Enter iteration maximum :  ')
HowMany <- as.numeric(readline())
message('Enter a tolerance value for stopping (e.g. 1e-06) :  ')
HowSmall <- as.numeric(readline())
# Now start the iterations
for (i in 1:HowMany) {
  xnew <- xnow - func(xnow)/dfdx(xnow)
# test for stopping
  if (abs(xnew-xnow) < HowSmall){
    message('Update not changing')
    xnow <- xnew
    print(cbind(xnow,xnew,func(xnew)))
    break
  }
  if (abs(func(xnew) < HowSmall)) {
    message('Function value close to zero')
    xnow <- xnew
    print(cbind(xnow,xnew,func(xnew)))
    break
  }
# next iteration
xnow <- xnew
}
if (i >= HowMany){
  message('Iteration limit reached')
  print(cbind(xnow,xnew,func(xnew)))
}
```

Listing 23 is the main code. Notice how the function definitions are changed, in particular `dfdx`.

Figure 58 is a screen capture of the program run after the code modification above.



```
  return(func);
}
# Define Derivative Here
dfdx <- function(x)
{
#  dfdx <- exp(x) + 10*sin(x);
  dfdx <- (func(x + 1.0e-06) - func(x))/(1.0e-06)
  return(dfdx);
}
# Newton's Method Here
```

```
> source('~/Desktop/CE4333-PCH-R/NewtonExample.R')
Enter an initial guess for X for Newton method :
1
Enter iteration maximum :
20
Enter a tolerance value for stopping (e.g. 1e-06) :
1e-06
Function value close to zero
         xnow     xnew
[1,] 4.593209 4.593209 2.922036e-10
> 
```

Figure 58. Program run after changing from analytical to finite-difference approximation for the derivative..

The advantage of the approximate derivative is that we don't have to do the calculus — just code in the function.

The obvious advantage of modular coding is to protect the parts of the code that are static, and just modify the function definitions. We can keep a working example around in case we break something and use that to find what we broke.

3.9.3 Method Fails

The three subsections below describe the ways that the method routinely fails, along with some suggestions for remedy. Generally we should plot the function before trying to find a root, but sometimes the root finding is a component of a more complex program and we just want it to work. In that situation, the programmer would build in many more tests that the three above to try to force a result before giving up.

3.9.4 Multiple Roots

Figure 59 illustrates the behavior in the presence of multiple roots. When there are multiple roots the method will converge on the root that that is defined by the initial guess.[15] The initial estimate must be close enough to the desired root to converge to the root.[16] Another challenge is what happens if the initial guess is at the divide

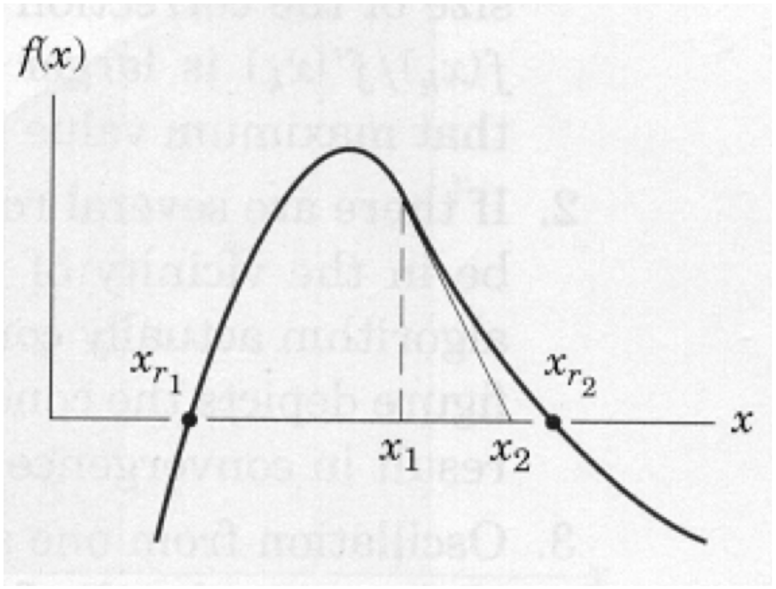


Figure 59. Multiple roots..

(the peak of the function in Figure 59); in such cases we may actually get a divergent solution because the slope of the function at that peak is nearly zero.

3.9.5 Cycling

Cycling can occur when the root is close to an inflection point of the function. Usual practice is to again limit the step size to prevent such behavior. Figure 60 is an illustration of cycling. A good remedy for cycling is to first detect the cycling, then provide a small "shove" to the guess. Examination of root finding codes often reveals a pseudo-random number generator within the code that will provide this shove when cycling is detected.

[15]This behavior is called "sensitive dependence on initial conditions".
[16]Ironically, we need a good idea of the answer before we start the method.

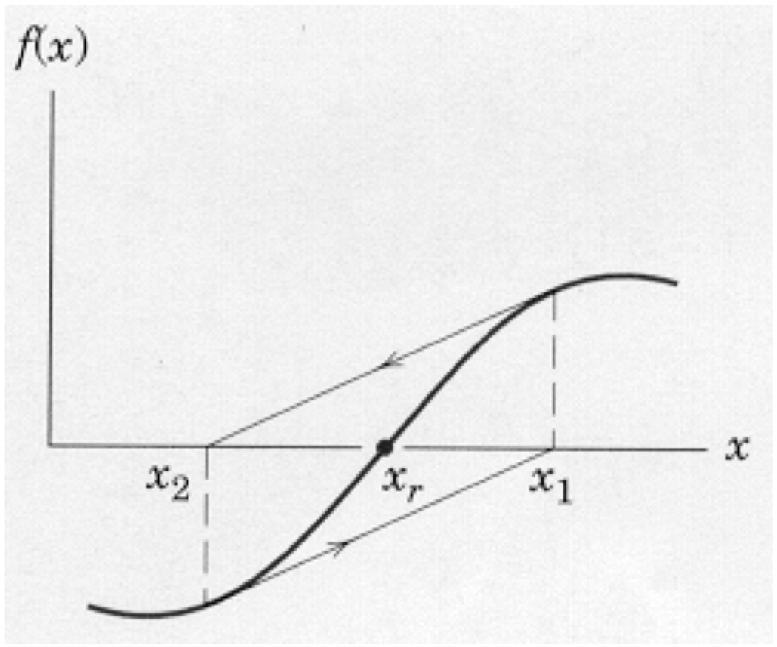


Figure 60. Estimates cycling around a root..

3.9.6 Near-zero derivatives

The derivative in Equation 53 must not be zero, otherwise the guess corresponds to a maximum or minimum of the function and the tangent line will never intersect the x-axis. The derivative must not be too close to zero, otherwise the slope will be so small as to make the correction too large to produce a meaningful update. Usual practice is to limit the size of the correction term to some maximum and to use this maximum value whenever the formula prescribes a larger step. Divergence to $\pm\infty$ is usually explained by near-zero derivatives at the sign change. The bi-section method is a little more robust in this respect.

3.10 Related Concepts

A couple of other root finding methods are worth mentioning because they can sometimes serve as a fallback when Newton's method fails. Two robust methods are bisection and false-positioning. These are discussed in the suggested reading list in Chapra's textbook.

3.11 Exercise Set

1. Build a Newton's Method program (or use mine) and make the program request a tolerance value for "how close to zero" is the function, and "how small is the change in update values." Build your code using the modular approach (two files). Test your code using the same example in the notes.
2. Now modify the main and the function module to use approximate derivatives (the finite-difference formulation) and require the user supply a step size. Test the code using the same example in the notes.

   For each of the exercises above, prepare documentation similar to the notes where you describe the salient points of your program.
3. Now use your program to find roots for the following equations:
   (a) $\exp(x) - 3x^2 = 0$

   (b) $\ln(x) - x + 2 = 0$

   (c) $\tan(x) - x - 1 = 0$

   For these three equations, document your search for roots. Identify if there are bad initial guesses that cause the program to fail to find a root. Also the equations may have multiple roots. If you discover multiple roots, identify the starting values one needs to use to converge to a particular root.

These exercises are also located on the class server in `ES-3`.

4 Simultaneous Linear Systems of Equations

Many engineering simulations require the solution of simultaneous algebraic equations. These algebraic equation systems are either linear or non-linear in the unknown variables. Many computation schemes have been developed to solve the resulting systems, mostly depending on the structure of the systems (and the corresponding coefficient matrices).

The solution of linear (or non-linear for that matter) can be accomplished using either direct methods or iterative (successive approximation) methods. The method choice depends on:

1. The amount of computation required (size of the problem) and computer memory available.[17]

2. The accuracy of the solution required.

3. The ability to control accuracy (i.e. find accurate enough solutions) to improve overall computation speed and throughput.

Direct solution methods lead to results by means of finite and predictable operations count, but at the expense of error amplification and difficulty to deal with near-singular systems. Iterative methods can converge to exact solutions, are robust in near-singular cases, but at the expense of a non-predictable number of operations.

In this chapter we will see how to solve systems using built-in method(s) in **R** and will also see the simplest of the iterative methods, Jacobi iteration. Jacobi iteration is presented for several reasons: it is simple to program, it shows the beauty of iteration when it works, and introduces a concept called pre-conditioning. For problems in this workbook, the built in `solve(...)` is recomended; we will use Jacobi iteration later on the the aquifer flow models, because the model equation structure is quite amenable to this kind of solution method.

For really large systems of equations iterative methods probably dominate because they are quite amenable to out-of-core solution — Jacobi iteration is ideal for parallel processing in a GPU[18]

[17]In the past, the memory was indeed an issue – its less so today; a really big problem of thousands of equations and thousands of variables might indeed be too big for any single computer array and would require out-of-core solver techniques, which I suspect are a slowly dying art.

[18]Graphics Processing Unit — Nearly all our laptops have GPU; either an Intel, NVIDIA, or AMD. These are intended for rendering graphics, but can be directly accessed with the proper software tools and can perform floating point operations really quickly. For example on my laptop I have an NVIDIA GeForce GT750M which I can program using a CUDA toolkit. If I had a really large system to solve, I would try Jacobi iteration, make each equation a thread, the solution guess a thread, and the update a thread. Its relatively easy to multiply, add, and divide threads, so one could compute the update directly from parallel thread multiplication using the guess, then thread addition to update the guess, and repeat. GPU programming is beyond this handbook, but remember that one can trade efficiency for speed if the operations are simple vector arithmetic.

4.1 Numerical Linear Algebra – Matrix Manipulation

This section introduces use of matrices in **R** to learn how to address particular elements of a matrix – once that is understood, the remaining arithmetic is reasonably straightforward.

4.2 The Matrix — A data structure

Listing 24 is script fragment that reads in two different matrices **A** and **B**, and writes them back to the screen. While such an action alone is sort of meaningless, the code does illustrate how to read the two different files, and write back the result in a row wise fashion.

The two matrices are

$$\mathbf{A} = \begin{pmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \tag{21}$$

and

$$\mathbf{B} = \begin{pmatrix} 5 & 8 & 1 & 2 \\ 6 & 7 & 3 & 0 \\ 4 & 5 & 9 & 1 \end{pmatrix} \tag{22}$$

Now that we have a way (albeit pretty arcane) for getting matrices into our program from a file[19] we can explore some elementary matrix arithmetic operations, and then will later move on to some more sophisticated operations, ultimately culminating in solutions to systems if linear equations (and non-linear systems in the next chapter).

[19]The read from a file is a huge necessity — manually entering values will get old fast. I have written matrix generators whose purpose in life is to construct matrices and put them into files for subsequent processing — often these programs are pretty simple because of structure in a problem, at other times they rival the solution tool in complexity; once for a Linear Programming model (circa 1980's) I developed a code to write a 1200 X 1200 matrix to a file, which would be functionally impossible to enter by hand.

Listing 24. R code demonstrating reading in two matrices.

```
# R script for some matrix operations
############## READ IN DATA FROM A FILE ####################
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-
    LinearSystems/RScripts"
filename <- "MatrixA.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Read the first file
yy <- read.table(fileToRead,header=FALSE,sep=",") # comma seperated ASCII, No header
filename <- "MatrixB.txt"   # change the filename
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Read the second file
zz <- read.table(fileToRead,header=FALSE,sep=",") # comma seperated ASCII, No header
############## Get Row and Column Counts ##################
HowManyColumnsA <- length(yy)
HowManyRowsA    <- length(yy$V1)
HowManyColumnsB <- length(zz)
HowManyRowsB    <- length(zz$V1)
###############  Build A and B Matrices  ####################
Amat <- matrix(0,nrow = HowManyRowsA, ncol = HowManyColumnsA)
Bmat <- matrix(0,nrow = HowManyRowsB, ncol = HowManyColumnsB)
for (i in 1:HowManyRowsA){
  for(j in 1:(HowManyColumnsA)){
    Amat[i,j] <- yy[i,j]
  }
}
rm(yy) # deallocate zz and just work with matrix and vectors
for (i in 1:HowManyRowsB){
  for(j in 1:(HowManyColumnsB)){
    Bmat[i,j] <- zz[i,j]
  }
}
rm(zz) # deallocate zz and just work with matrix and vectors
############# Echo Input ##################################
print(Amat)
print(Bmat)
```

4.3 Matrix Arithmetic

Analysis of many problems in engineering result in systems of simultaneous equations. We typically represent systems of equations with a matrix. For example the two-equation system,

$$\begin{aligned} 2x_1 &+ 3x_2 = 8 \\ 4x_1 &- 3x_2 = -2 \end{aligned} \tag{23}$$

Could be represented by set of vectors and matrices[20]

$$\mathbf{A} = \begin{pmatrix} 2 & 3 \\ 4 & -3 \end{pmatrix} \ \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \ \mathbf{b} = \begin{pmatrix} 8 \\ -2 \end{pmatrix} \tag{24}$$

and the linear system then written as

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{25}$$

[20]Usually called "vector-matrix" form. Additionally, a vector is really just a matrix with column rank = 1 (a single column matrix).

So the "algebra" is considerably simplified, at least for writing things, however we now have to be able to do things like multiplication (indicated by $\cdot$) as well as the concept of addition and subtraction, and division (multiplication by an inverse). There are also several kinds of matrix multiplication – the inner product as required by the linear system, the vector (cross product), the exterior (wedge), and outer (tensor) product are a few of importance in both mathematics and engineering.

The remainder of this section will examine the more common matrix operations.

4.3.1 Matrix Definition

A matrix is a rectangular array of numbers.

$$\begin{pmatrix} 1 & 5 & 7 & 2 \\ 2 & 9 & 17 & 5 \\ 11 & 15 & 8 & 3 \end{pmatrix} \tag{26}$$

The size of a matrix is referred to in terms of the number of rows and the number of columns. The enclosing parenthesis are optional above, but become meaningful when writing multiple matrices next to each other. The above matrix is 3 by 4.

When we are discussing matrices we will often refer to specific numbers in the matrix. To refer to a specific element of a matrix we refer to the row number (i) and the column number (j). We will often call a specific element of the matrix, the $a_{i,j}$ -th element of the matrix. For example $a_{2,3}$ element in the above matrix is 17. In **R** we would refer to the element as `a_matrix[i][j]` or whatever the name of the matrix is in the program.

4.3.2 Multiply a matrix by a scalar

A scalar multiple of a matrix is simply each element of the matrix multiplied by the scalar value. Consider the matrix **A** below.

$$\mathbf{A} = \begin{pmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \tag{27}$$

If the scalar is say 2, then $2 \times \mathbf{A}$ is computed by doubling each element of **A**, as

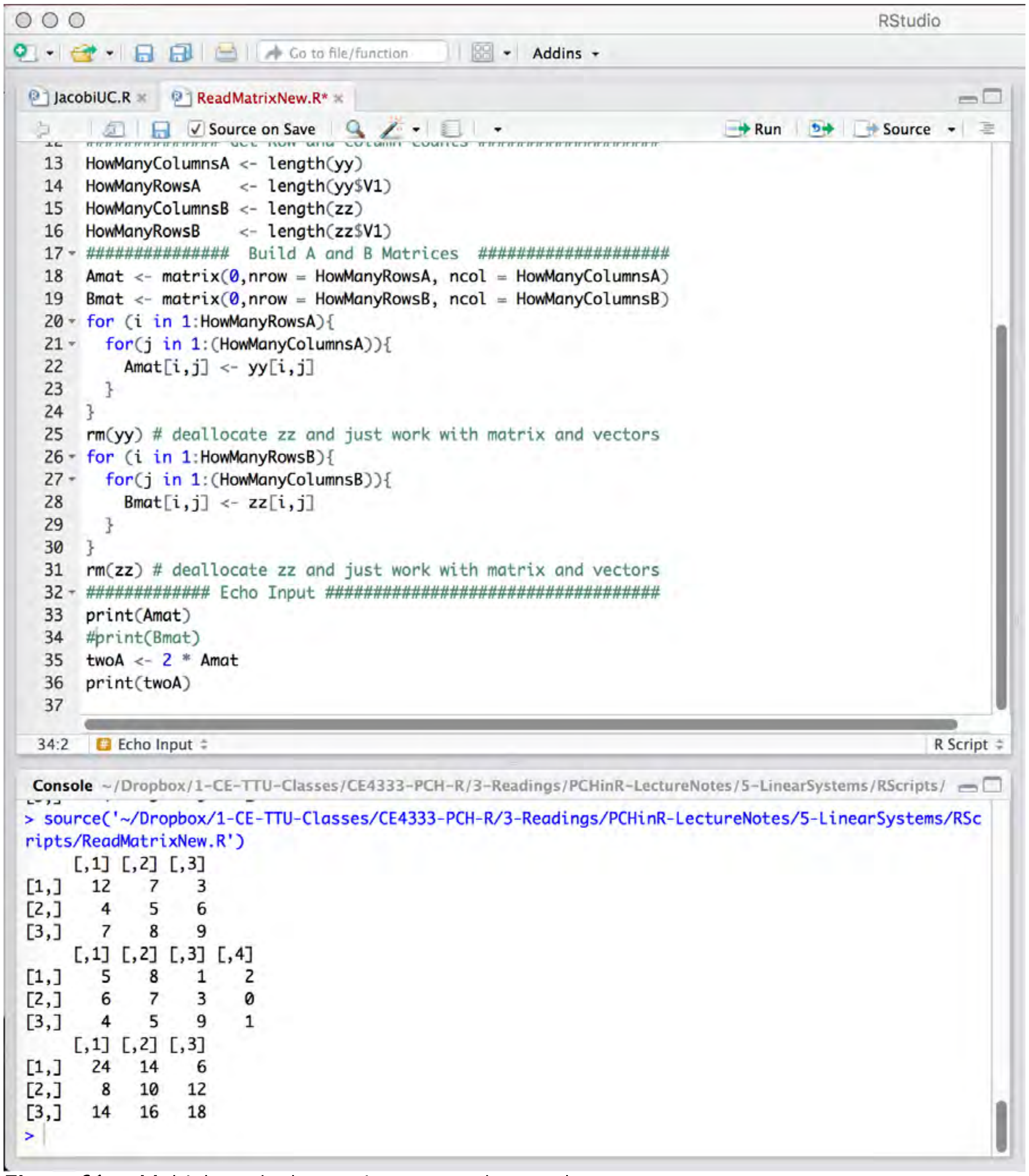
$$2\mathbf{A} = \begin{pmatrix} 24 & 14 & 6 \\ 8 & 10 & 12 \\ 17 & 16 & 18 \end{pmatrix} \tag{28}$$

In **R** we can simply perform the arithmetic as

Listing 25. R code demonstrating scalar multiplication.

```
#######################
twoA <- 2 * Amat
print(twoA)
```

Figure 61 is an example using the earlier **A** matrix and multiplying it by the scalar value of 2.0.


RStudio
Go to file/function
Addins
JacobiUC.R
ReadMatrixNew.R*
Source on Save
Run
Source
34:2
Echo Input
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RScripts/

```
HowManyColumnsA <- length(yy)
HowManyRowsA    <- length(yy$V1)
HowManyColumnsB <- length(zz)
HowManyRowsB    <- length(zz$V1)
###############  Build A and B Matrices  ###################
Amat <- matrix(0,nrow = HowManyRowsA, ncol = HowManyColumnsA)
Bmat <- matrix(0,nrow = HowManyRowsB, ncol = HowManyColumnsB)
for (i in 1:HowManyRowsA){
  for(j in 1:(HowManyColumnsA)){
    Amat[i,j] <- yy[i,j]
  }
}
rm(yy) # deallocate zz and just work with matrix and vectors
for (i in 1:HowManyRowsB){
  for(j in 1:(HowManyColumnsB)){
    Bmat[i,j] <- zz[i,j]
  }
}
rm(zz) # deallocate zz and just work with matrix and vectors
############# Echo Input ###################################
print(Amat)
#print(Bmat)
twoA <- 2 * Amat
print(twoA)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RSc
ripts/ReadMatrixNew.R')
     [,1] [,2] [,3]
[1,]   12    7    3
[2,]    4    5    6
[3,]    7    8    9
     [,1] [,2] [,3] [,4]
[1,]    5    8    1    2
[2,]    6    7    3    0
[3,]    4    5    9    1
     [,1] [,2] [,3]
[1,]   24   14    6
[2,]    8   10   12
[3,]   14   16   18
>
```

Figure 61. Multiply each element in `amatrix` by a scalar .

4.3.3 Matrix addition (and subtraction)

Matrix addition and subtraction are also element-by-element operations. In order to add or subtract two matrices they must be the same size and shape. This requirement means that they must have the same number of rows and columns. To add or subtract a matrix we simply add or subtract the corresponding elements from each matrix.

For example consider the two matrices **A** and **2A** below

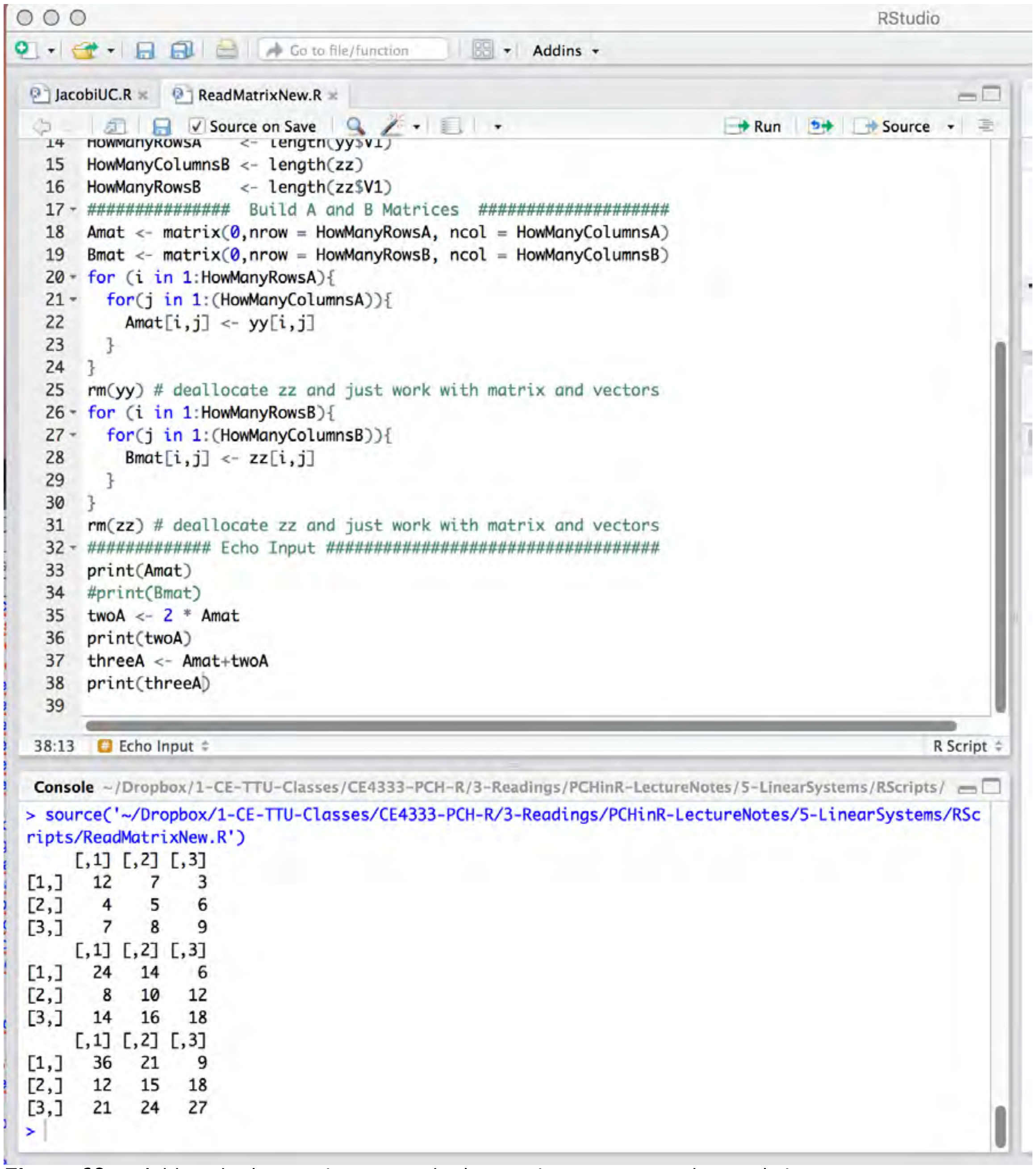
$$\mathbf{A} = \begin{pmatrix} 12 & 7 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \; 2\mathbf{A} = \begin{pmatrix} 24 & 14 & 6 \\ 8 & 10 & 12 \\ 17 & 16 & 18 \end{pmatrix} \tag{29}$$

For example the sum of these two matrices is the matrix named **3A**, shown below:

$$\mathbf{A} + \mathbf{2A} = \begin{pmatrix} 12+24 & 7+14 & 3+6 \\ 4+8 & 5+10 & 6+12 \\ 7+14 & 8+16 & 9+18 \end{pmatrix} = \begin{pmatrix} 36 & 21 & 9 \\ 12 & 15 & 18 \\ 21 & 24 & 27 \end{pmatrix} \tag{30}$$

Now to do the operation in **R**, we need to read in the matrices, perform the addition, and write the result. In the code example in 62 I added a third matrix to store the result – generally we don't want to clobber existing matrices, so we will use the result instead.

Subtraction is performed in a similar fashion, except the subtraction operator is used.


RStudio
Go to file/function
Addins
JacobiUC.R
ReadMatrixNew.R
Source on Save
Run
Source
38:13
Echo Input
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RScripts/

```
HowManyRowsA     <- length(yy$V1)
HowManyColumnsB <- length(zz)
HowManyRowsB     <- length(zz$V1)
###############  Build A and B Matrices  ####################
Amat <- matrix(0,nrow = HowManyRowsA, ncol = HowManyColumnsA)
Bmat <- matrix(0,nrow = HowManyRowsB, ncol = HowManyColumnsB)
for (i in 1:HowManyRowsA){
  for(j in 1:(HowManyColumnsA)){
    Amat[i,j] <- yy[i,j]
  }
}
rm(yy) # deallocate zz and just work with matrix and vectors
for (i in 1:HowManyRowsB){
  for(j in 1:(HowManyColumnsB)){
    Bmat[i,j] <- zz[i,j]
  }
}
rm(zz) # deallocate zz and just work with matrix and vectors
############ Echo Input ##################################
print(Amat)
#print(Bmat)
twoA <- 2 * Amat
print(twoA)
threeA <- Amat+twoA
print(threeA)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RSc
ripts/ReadMatrixNew.R')
     [,1] [,2] [,3]
[1,]   12    7    3
[2,]    4    5    6
[3,]    7    8    9
     [,1] [,2] [,3]
[1,]   24   14    6
[2,]    8   10   12
[3,]   14   16   18
     [,1] [,2] [,3]
[1,]   36   21    9
[2,]   12   15   18
[3,]   21   24   27
>
```

Figure 62. Add each element in `A` to each element in `twoA`, store the result in `threeA`..

4.3.4 Multiply a matrix

One kind of matrix multiplication is an inner product. Usually when matrix multiplication is mentioned without further qualification ,the implied meaning is an inner product of the matrix and a vector (or another matrix).

Matrix multiplication is more complex than addition and subtraction. If two matrices such as a matrix **A** (size l x m) and a matrix **B** (size m x n) are multiplied together, the resulting matrix **C** has a size of l x n. The order of multiplication of matrices is extremely important[21].

To obtain $\mathbf{C} = \mathbf{A}\ \mathbf{B}$, the number of columns in **A** must be the same as the number of rows in **B**. In order to carry out the matrix operations for multiplication of matrices, the i, j-th element of **C** is simply equal to the scalar (dot or inner) product of row i of **A** and column j of **B**.

Consider the example below

$$\mathbf{A} = \begin{pmatrix} 1 & 5 & 7 \\ 2 & 9 & 3 \end{pmatrix} \ \mathbf{B} = \begin{pmatrix} 3 & -2 \\ -2 & 1 \\ 1 & 1 \end{pmatrix} \tag{31}$$

First, we would evaluate if the operation is even possible, **A** has two rows and three columns. **B** has three rows and two columns. By our implied multiplication "rules" for the multiplication to be defined the first matrix must have the same number of rows as the second matrix has columns (in this case it does), and the result matrix will have the same number of rows as the first matrix, and the same number of columns as the second matrix (in this case the result will be a 2X2 matrix).

$$\mathbf{C} = \mathbf{AB} = \begin{pmatrix} c_{1,1} & c_{1,2} \\ c_{2,1} & c_{2,2} \end{pmatrix} \tag{32}$$

And each element of **C** is the dot product of the row vector of **A** and the column vector of **B**.

[21] Matrix multiplication is not transitive; $\mathbf{A}\ \mathbf{B} \neq \mathbf{B}\ \mathbf{A}$.

$$c_{1,1} = \begin{pmatrix} 1 & 5 & 7 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix} = \big((1)(3) + (5)(-2) + (7)(1)\big) = 0 \tag{33}$$

$$c_{1,2} = \begin{pmatrix} 1 & 5 & 7 \end{pmatrix} \cdot \begin{pmatrix} -2 \\ 1 \\ 1 \end{pmatrix} = \big((1)(-2) + (5)(1) + (7)(1)\big) = 10 \tag{34}$$

$$c_{2,1} = \begin{pmatrix} 2 & 9 & 3 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ -2 \\ 1 \end{pmatrix} = \big((2)(3) + (9)(-2) + (3)(1)\big) = -9 \tag{35}$$

$$c_{2,2} = \begin{pmatrix} 2 & 9 & 3 \end{pmatrix} \cdot \begin{pmatrix} -2 \\ 1 \\ 1 \end{pmatrix} = \big((2)(-2) + (9)(1) + (3)(1)\big) = 8 \tag{36}$$

Making the substitutions results in :

$$\mathbf{C} = \mathbf{AB} = \begin{pmatrix} 0 & 10 \\ -9 & 8 \end{pmatrix} \tag{37}$$

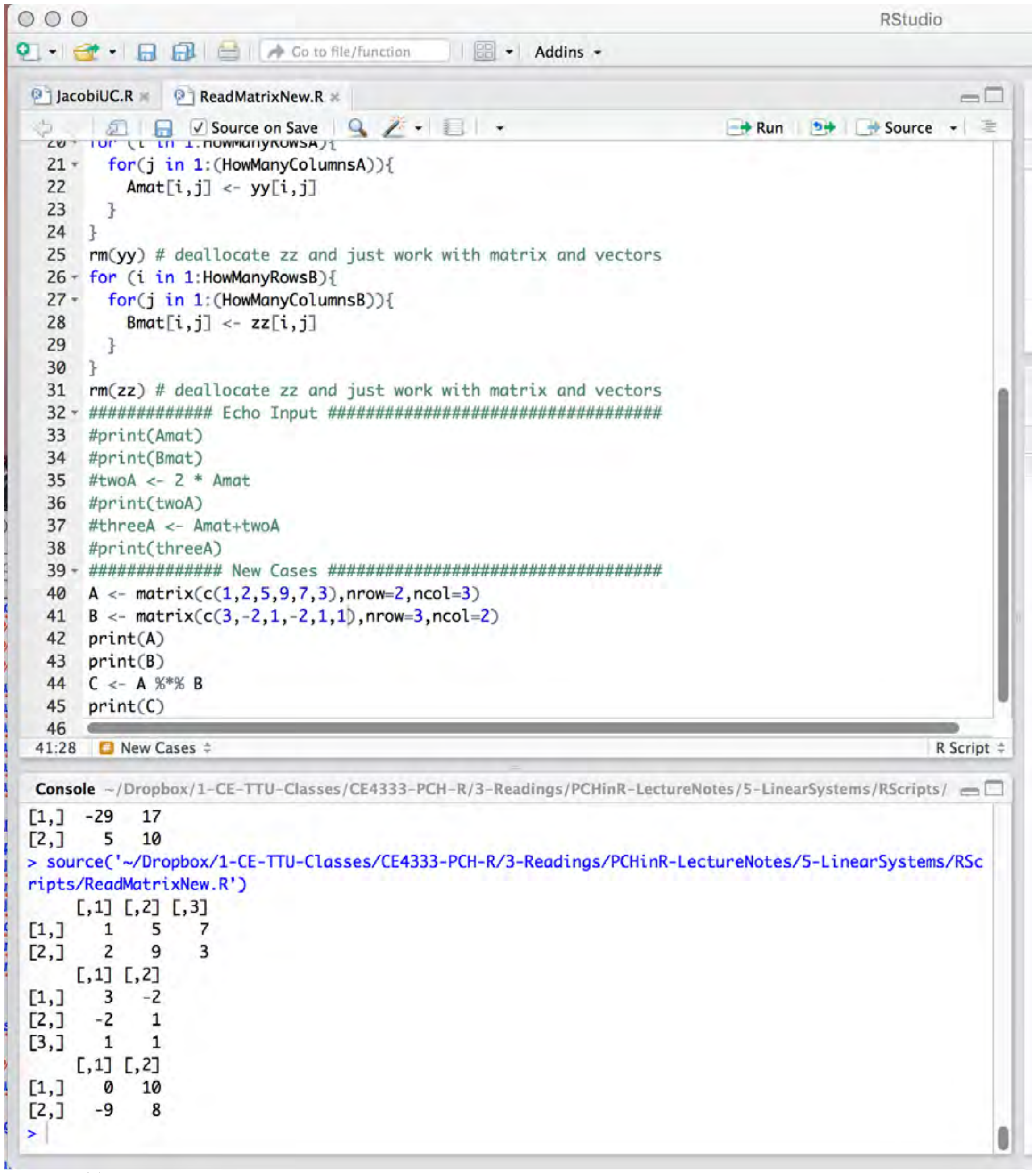
So in an algorithmic sense we will have to deal with three matrices, the two source matrices and the destination matrix. We will also have to manage element-by-element multiplication and be able to correctly store through rows and columns. In **R** this manipulation is handled for us by the matrix multiply operator `% * %`.

Figure 63 is a script that multiplies the two matrices above and prints the result.[22]

4.3.5 Identity matrix

In computational linear algebra we often need to make use of a special matrix called the "Identity Matrix". The Identity Matrix is a square matrix with all zeros except the $i, i0$-th element (diagonal) which is equal to 1:

[22]Internal to **R** the actual code for the multiplication is three nested for-loops. The outer loop counts based rows of the first matrix, the middle loop counts based on columns of the second matrix, and the inner most loop counts based on columns of the first matrix (or rows of the second matrix). In many practical cases we may actually have to manipulate at the element level — similar to how the `zz` object was put into a matrix explicitly above.


RStudio
Go to file/function
Addins
JacobiUC.R
ReadMatrixNew.R
Source on Save
Run
Source
41:28
New Cases
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RScripts/

```
for(j in 1:(HowManyColumnsA)){
  Amat[i,j] <- yy[i,j]
}
}
rm(yy) # deallocate zz and just work with matrix and vectors
for (i in 1:HowManyRowsB){
  for(j in 1:(HowManyColumnsB)){
    Bmat[i,j] <- zz[i,j]
  }
}
rm(zz) # deallocate zz and just work with matrix and vectors
############ Echo Input ##################################
#print(Amat)
#print(Bmat)
#twoA <- 2 * Amat
#print(twoA)
#threeA <- Amat+twoA
#print(threeA)
############# New Cases ##################################
A <- matrix(c(1,2,5,9,7,3),nrow=2,ncol=3)
B <- matrix(c(3,-2,1,-2,1,1),nrow=3,ncol=2)
print(A)
print(B)
C <- A %*% B
print(C)
```

```
[1,]  -29   17
[2,]    5   10
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RSc
ripts/ReadMatrixNew.R')
     [,1] [,2] [,3]
[1,]    1    5    7
[2,]    2    9    3
     [,1] [,2]
[1,]    3   -2
[2,]   -2    1
[3,]    1    1
     [,1] [,2]
[1,]    0   10
[2,]   -9    8
>
```

Figure 63. Matrix multiplication example.

$$\mathbf{I}_{3\times3} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{38}$$

Usually we don't bother with the size subscript i used above and just stipulate that the matrix is sized as appropriate. Multiplying any matrix by (a correctly sized) identity matrix results in no change in the matrix. $\mathbf{IA} = \mathbf{A}$

In **R** the identity matrix is easily created using `<matrix_name> <- diag(dimension)`.

4.3.6 Matrix Inverse

In many practical computational and theoretical operations we employ the concept of the inverse of a matrix. The inverse is somewhat analogous to"dividing" by the matrix. Consider our linear system

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{39}$$

If we wished to solve for $\mathbf{x}$ we would "divide" both sides of the equation by $\mathbf{A}$. Instead of division (which is essentially left undefined for matrices) we instead multiply by the inverse of the matrix[23]. The inverse of a matrix $\mathbf{A}$ is denoted by $\mathbf{A}^{-1}$ and by definition is a matrix such that when $\mathbf{A}^{-1}$ and $\mathbf{A}$ are multiplied together, the identity matrix $\mathbf{I}$ results. e.g. $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$

Lets consider the matrixes below

$$\mathbf{A} = \begin{pmatrix} 2 & 3 \\ 4 & -3 \end{pmatrix} \tag{40}$$

$$\mathbf{A}^{-1} = \begin{pmatrix} \frac{1}{6} & \frac{1}{6} \\ \\ \frac{2}{9} & -\frac{1}{9} \end{pmatrix} \tag{41}$$

We can check that the matrices are indeed inverses of each other using **R** and matrix multiplication — it should return an identity matrix.

Figure 64 is our multiplication script modified where $\mathbf{A} = \mathbf{A}$ and $\mathbf{B} = \mathbf{A}^{-1}$ perform the multiplication and then report the result. The result is the identity matrix regardless of the order of operation.[24]

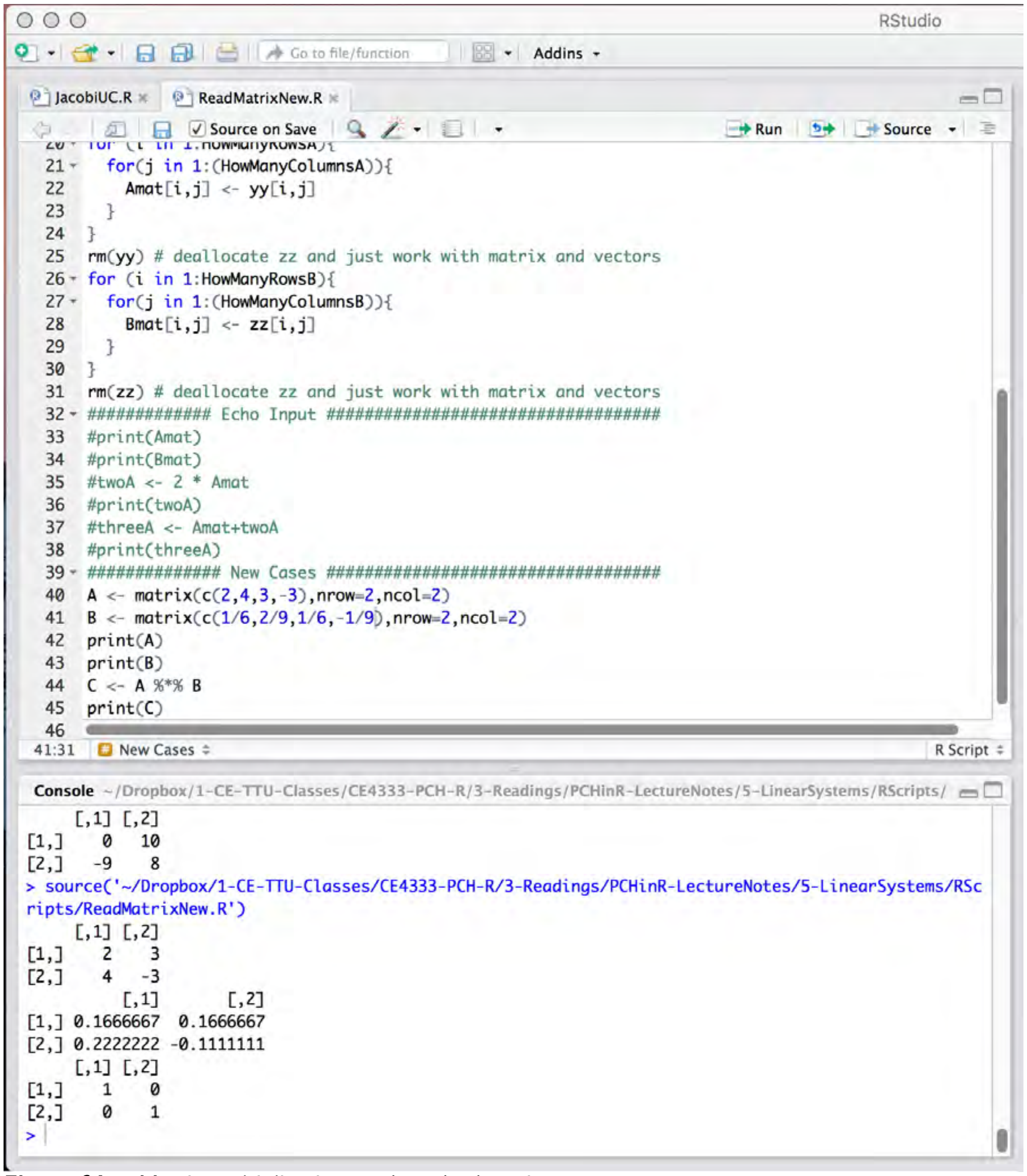
Now that we have some background on what an inverse is, it would be nice to know how to find them — that is a remarkably challenging problem. Here we examine a classical algorithm for finding an inverse if we really need to — computationally we only invert if necessary, there are other ways to "divide" that are faster.

4.3.7 Gauss-Jordan method of finding $\mathbf{A}^{-1}$

There are a number of methods that can be used to find the inverse of a matrix using elementary row operations. An elementary row operation is any one of the three operations listed below:

[23]The matrix inverse is the multiplicative inverse of the matrix – we are defining the equivalent of a division operation, just calling it something else. This issue will be huge later on in our workbook, especially when we are dealing with non-linear systems

[24]Why do you think this is so, when above we stated that multiplication is intransitive?


RStudio
Go to file/function
Addins
JacobiUC.R
ReadMatrixNew.R
Source on Save
Run
Source
41:31
New Cases
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RScripts/

```
for (i in 1:HowManyRowsA){
  for(j in 1:(HowManyColumnsA)){
    Amat[i,j] <- yy[i,j]
  }
}
rm(yy) # deallocate zz and just work with matrix and vectors
for (i in 1:HowManyRowsB){
  for(j in 1:(HowManyColumnsB)){
    Bmat[i,j] <- zz[i,j]
  }
}
rm(zz) # deallocate zz and just work with matrix and vectors
############# Echo Input ##################################
#print(Amat)
#print(Bmat)
#twoA <- 2 * Amat
#print(twoA)
#threeA <- Amat+twoA
#print(threeA)
############## New Cases ##################################
A <- matrix(c(2,4,3,-3),nrow=2,ncol=2)
B <- matrix(c(1/6,2/9,1/6,-1/9),nrow=2,ncol=2)
print(A)
print(B)
C <- A %*% B
print(C)
```

```
     [,1] [,2]
[1,]    0   10
[2,]   -9    8
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RSc
ripts/ReadMatrixNew.R')
     [,1] [,2]
[1,]    2    3
[2,]    4   -3
          [,1]       [,2]
[1,] 0.1666667  0.1666667
[2,] 0.2222222 -0.1111111
     [,1] [,2]
[1,]    1    0
[2,]    0    1
>
```

Figure 64. Matrix multiplication used to check an inverse..

1. Multiply or divide an entire row by a constant.
2. Add or subtract a multiple of one row to/from another.
3. Exchange the position of any 2 rows.

The Gauss-Jordan method of inverting a matrix can be divided into 4 main steps. In order to find the inverse we will be working with the original matrix, augmented with the identity matrix – this new matrix is called the augmented matrix (because no-one has tried to think of a cooler name yet).

$$\mathbf{A}|\mathbf{I} = \begin{pmatrix} 2 & 3 & | & 1 & 0 \\ 4 & -3 & | & 0 & 1 \end{pmatrix} \tag{42}$$

We will perform elementary row operations based on the left matrix to convert it to an identity matrix – we perform the same operations on the right matrix and the result when we are done is the inverse of the original matrix.

So here goes – in the theory here, we also get to do infinite-precision arithmetic, no rounding/truncation errors.

1. Divide row one by the $a_{1,1}$ value to force a 1 in the $a_{1,1}$ position. This is elementary row operation 1 in our list above.

$$\mathbf{A}|\mathbf{I} = \begin{pmatrix} 2/2 & 3/2 & | & 1/2 & 0 \\ 4 & -3 & | & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 3/2 & | & 1/2 & 0 \\ 4 & -3 & | & 0 & 1 \end{pmatrix} \tag{43}$$

2. For all rows below the first row, replace row_j with $row_j - a_{j,1} * row_1$. This happens to be elementary row operation 2 in our list above.

$$\mathbf{A}|\mathbf{I} = \begin{pmatrix} 1 & 3/2 & | & 1/2 & 0 \\ 4-4(1) & -3-4(3/2) & | & 0-4(1/2) & 1-4(0) \end{pmatrix} = \begin{pmatrix} 1 & 3/2 & | & 1/2 & 0 \\ 0 & -9 & | & -2 & 1 \end{pmatrix} \tag{44}$$

3. Now multiply row_2 by $\frac{1}{a_{2,2}}$. This is again elementary row operation 1 in our list above.

$$\mathbf{A}|\mathbf{I} = \begin{pmatrix} 1 & 3/2 & | & 1/2 & 0 \\ 0 & -9/-9 & | & -2/-9 & 1/-9 \end{pmatrix} = \begin{pmatrix} 1 & 3/2 & | & 1/2 & 0 \\ 0 & 1 & | & 2/9 & -1/9 \end{pmatrix} \tag{45}$$

4. For all rows above and below this current row, replace row_j with $row_j - a_{2,2} * row_2$. This happens to again be elementary row operation 2 in our list above. What we are doing is systematically converting the left matrix into an identity matrix by multiplication of constants and addition to eliminate off-diagonal values and force 1 on the diagonal.

$$\mathbf{A}|\mathbf{I} = \tag{46}$$

$$\begin{pmatrix} 1 & 3/2-(3/2)(1) & | & 1/2-(3/2)(2/9) & 0-(3/2)(-1/9) \\ 0 & 1 & | & 2/9 & -1/9 \end{pmatrix} = \tag{47}$$

$$\begin{pmatrix} 1 & 0 & | & 1/6 & 1/6 \\ 0 & 1 & | & 2/9 & -1/9 \end{pmatrix} \tag{48}$$

5. As far as this example is concerned we are done and have found the inverse. With more than a 2X2 system there will be many operations moving up and down the matrix to eliminate the off-diagonal terms.

So the next logical step is to build an algorithm to perform these operations for us.

In **R** inversion is simply performed using the `solve(...)` function where the only argument passed to the function is the matrix.[25]

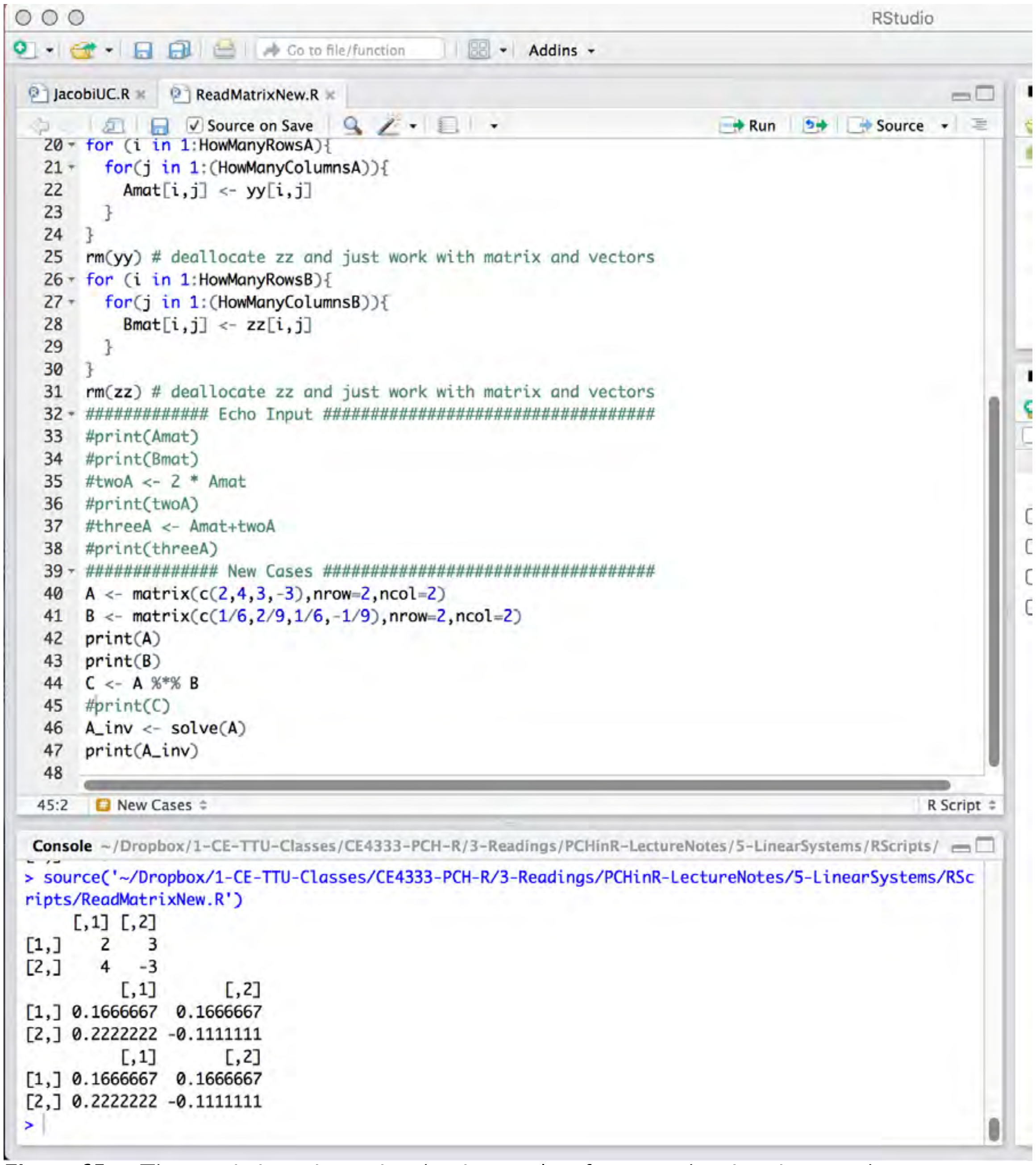
Figure 65 is a screen capture of using `solve(...)` to find the inverse of **A**. The result is identical to the input matrix $\mathbf{A}^{-1}$ above. While we now have the ability to solve linear systems by rearrangement into

$$\mathbf{x} = \mathbf{A^{-1}} \cdot \mathbf{b} \tag{49}$$

this is generally not a good approach (we are solving n linear systems to obtain the inverse, instead of only the one we seek!).

Instead to solve a linear system, we would supply the coefficient matrix **A** and the right hand side **b**, and then supply these two matrices to the solve routine
(e.g. `x <- solve(A,b)`).

[25]If we have to write code ourselves, its not terribly hard, but is lengthy and consequently error-prone. Sometimes we have no choice, but in this workbook, we will use the built-in tool as much as possible. **R** does not use Gaussian reduction unless we tell it to do so, it implements a factorization called LU (or Cholesky) decomposition, then computes the inverse by repeated solution of a linear system with the right hand side being selected from one of the identify matrix columns (as was done above).


RStudio
Go to file/function
Addins
JacobiUC.R
ReadMatrixNew.R
Source on Save
Run
Source
45:2
New Cases
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RScripts/

```
for (i in 1:HowManyRowsA){
  for(j in 1:(HowManyColumnsA)){
    Amat[i,j] <- yy[i,j]
  }
}
rm(yy) # deallocate zz and just work with matrix and vectors
for (i in 1:HowManyRowsB){
  for(j in 1:(HowManyColumnsB)){
    Bmat[i,j] <- zz[i,j]
  }
}
rm(zz) # deallocate zz and just work with matrix and vectors
############# Echo Input ###################################
#print(Amat)
#print(Bmat)
#twoA <- 2 * Amat
#print(twoA)
#threeA <- Amat+twoA
#print(threeA)
############## New Cases ###################################
A <- matrix(c(2,4,3,-3),nrow=2,ncol=2)
B <- matrix(c(1/6,2/9,1/6,-1/9),nrow=2,ncol=2)
print(A)
print(B)
C <- A %*% B
#print(C)
A_inv <- solve(A)
print(A_inv)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RSc
ripts/ReadMatrixNew.R')
     [,1] [,2]
[1,]    2    3
[2,]    4   -3
          [,1]       [,2]
[1,] 0.1666667  0.1666667
[2,] 0.2222222 -0.1111111
          [,1]       [,2]
[1,] 0.1666667  0.1666667
[2,] 0.2222222 -0.1111111
>
```

Figure 65. The matrix inversion script showing results of a run and various input and output..

4.4 Jacobi Iteration – An iterative method to find solutions

Iterative methods are often more rapid and economical in storage requirements than the direct methods in `solve(...)`.[26] The methods are useful (necessary) for non-linear systems of equations — we will use this feature later when we find solutions to networks of pipelines.

Lets consider a simple example:

$$\begin{array}{llll} 8x_1 & +1x_2 & -1x_3 = & 8 \\ 1x_1 & -7x_2 & +2x_3 = & -4 \\ 2x_1 & +1x_2 & +9x_3 = & 12 \end{array} \tag{50}$$

The solution is $x_1 = 1$, $x_2 = 1$, $x_3 = 1$. We begin the iterative scheme by refactoring each equation in terms of a single variable (there is a secret pivot step to try to make the system diagonally dominant – the example above has already been pivoted, or "pre-conditioned" for the solution method):

$$\begin{array}{lllll} x_1 = & 1.000 & & -0.125x_2 & 0.125x_3 \\ x_2 = & 0.571 & 0.143x_1 & & 0.286x_3 \\ x_3 = & 1.333 & -0.222x_1 & -0.111x_2 & \end{array} \tag{51}$$

Then supply an initial guess of the solution (e.g. $(0, 0, 0)$) and put these values into the right-hand side, the resulting left-hand side is an improved (hopefully) solution. Repeat the process until the solution stops changing, or goes obviously haywire.

This sequence of operation for the example above produces the results listed in Table 4.

Table 4. Jacobi Iteration Solution Sequence.

Iteration:	1-st	2-nd	3-rd	4-th	5th	6-th	7-th	8-th
x_1	0	1.000	1.095	0.995	0.993	1.002	1.001	1.000
x_2	0	0.571	1.095	1.026	0.990	0.998	1.001	1.000
x_3	0	1.333	1.048	0.969	1.000	1.004	1.001	1.000

As a practical matter, refactoring the equations can instead be accomplished by computing the inverse of each diagonal coefficient – and matrix multiplication, scalar division, and vector addition are all that is required to find a solution (if the method will actually work).

In linear algebra terms the Jacobi iteration method (without refactoring) performs the following steps:

[26]The **R** solve routine is pretty robust, if you tell it `sparse=TRUE` it has a lot of internal methods to pre-condition the problem for fast solution. But for really big systems we may wish to program our own solver — especially if these systems have some special and predictable structure.

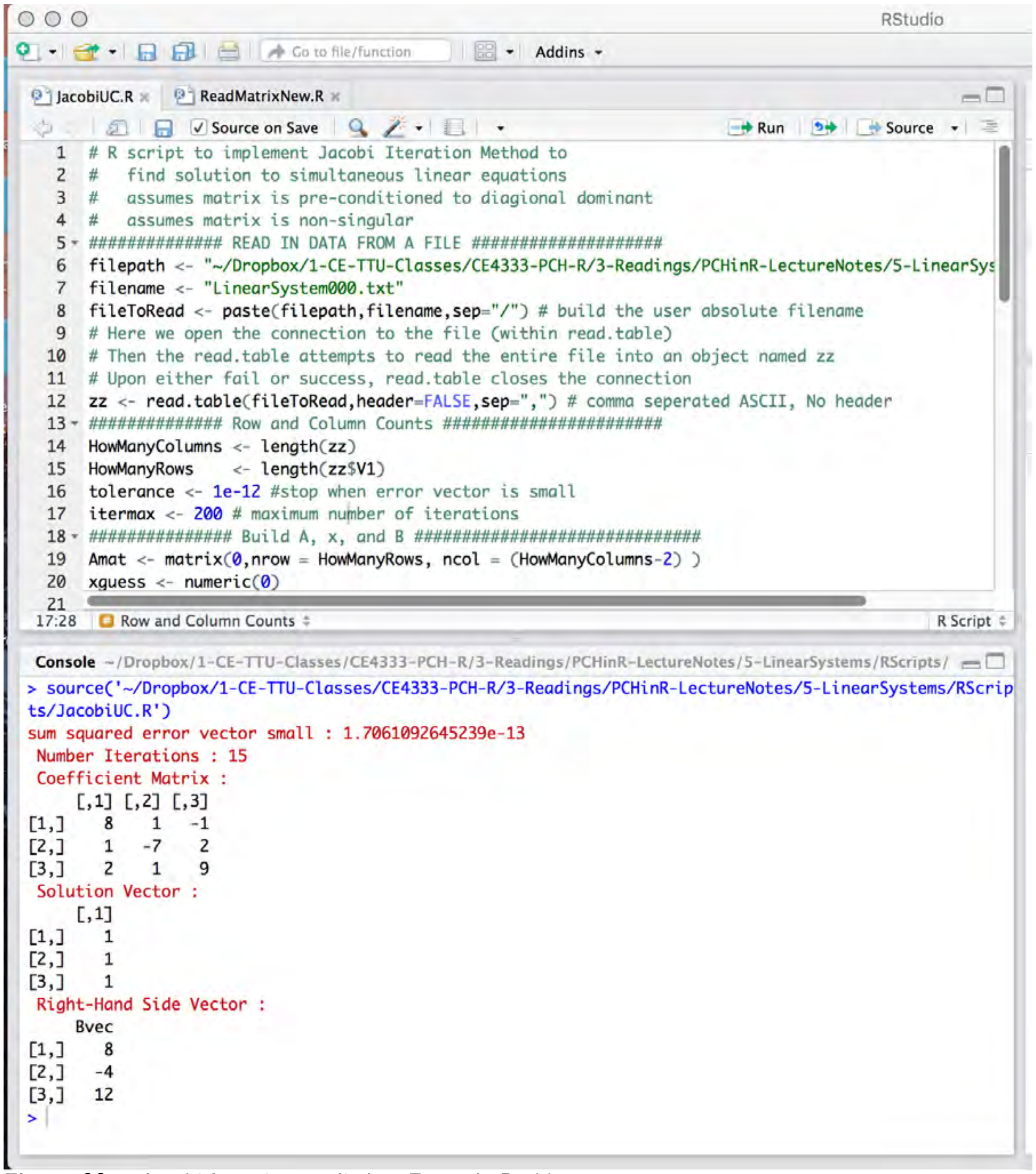
1. Read in $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{x_{guess}}$.
2. Construct a vector from the diagonal elements of $\mathbf{A}$. This vector, $\mathbf{W}$, will have one column, and same number of rows as $\mathbf{A}$.
3. Perform matrix arithmetic to compute an error vector, $\mathbf{residual} = \mathbf{A}\cdot\mathbf{x_{guess}} - \mathbf{b}$.
4. Divide this error vector by the diagonal weights $\mathbf{update} = \mathbf{residual}/\mathbf{W}$
5. Update the solution vector $\mathbf{x_{new}} = \mathbf{x_{guess}} - \mathbf{update}$
6. Test for stopping, if not indicated, move the new solution into the guess and return to step 3.
7. If time to stop, then report result and stop.

Listing 26 implements in **R** the algorithm described above to find solutions by the Jacobi iteration method. The script does not pre-condition the linear system (so we have to do that ourselves).

Listing 26. R code demonstrating Jacobi Iteration.

```
# R script to implement Jacobi Iteration Method to
#   find solution to simultaneous linear equations
#   assumes matrix is pre-conditioned to diagonal dominant
#   assumes matrix is non-singular
############## READ IN DATA FROM A FILE ####################
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-
    LinearSystems/RScripts"
filename <- "LinearSystem000.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=",") # comma seperated ASCII, No header
############## Row and Column Counts ######################
HowManyColumns <- length(zz)
HowManyRows    <- length(zz$V1)
tolerance <- 1e-12 #stop when error vector is small
itermax <- 200 # maximum number of iterations
################ Build A, x, and B ############################
Amat <- matrix(0,nrow = HowManyRows, ncol = (HowManyColumns-2) )
xguess <- numeric(0)
Bvec <- numeric(0)
Wvec <- numeric(0)
############################################################
for (i in 1:HowManyRows){
  for(j in 1:(HowManyColumns-2)){
    Amat[i,j] <- zz[i,j]
  }
  Bvec[i] <- zz[i,HowManyColumns-1]
  xguess[i] <- zz[i,HowManyColumns]
  Wvec[i] <- Amat[i,i]
}
rm(zz) # deallocate zz and just work with matrix and vectors
#####################Implement Jacobi Iteration #############
for(iter in 1:itermax){
Bguess <- Amat %*% xguess
residue <- Bguess - Bvec
xnew <- xguess - residue/Wvec
xguess <- xnew
testval <- t(residue) %*% residue
if (testval < tolerance) {
  message("sum squared error vector small : ",testval);
  break
}
}
if( iter == itermax) message("Method Fail")
message(" Number Iterations : ", iter)
message(" Coefficient Matrix : ")
print(cbind(Amat))
message(" Solution Vector : ")
print(cbind(xguess))
message(" Right-Hand Side Vector : ")
print(cbind(Bvec))
```

Figure 66 is a screen capture of the script in Listing 26 applied to the example problem.



```
# R script to implement Jacobi Iteration Method to
#   find solution to simultaneous linear equations
#   assumes matrix is pre-conditioned to diagional dominant
#   assumes matrix is non-singular
############## READ IN DATA FROM A FILE ####################
filepath <- "~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSys
filename <- "LinearSystem000.txt"
fileToRead <- paste(filepath,filename,sep="/") # build the user absolute filename
# Here we open the connection to the file (within read.table)
# Then the read.table attempts to read the entire file into an object named zz
# Upon either fail or success, read.table closes the connection
zz <- read.table(fileToRead,header=FALSE,sep=",") # comma seperated ASCII, No header
############## Row and Column Counts ######################
HowManyColumns <- length(zz)
HowManyRows    <- length(zz$V1)
tolerance <- 1e-12 #stop when error vector is small
itermax <- 200 # maximum number of iterations
############### Build A, x, and B ##############################
Amat <- matrix(0,nrow = HowManyRows, ncol = (HowManyColumns-2) )
xguess <- numeric(0)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/5-LinearSystems/RScrip
ts/JacobiUC.R')
sum squared error vector small : 1.7061092645239e-13
 Number Iterations : 15
 Coefficient Matrix :
     [,1] [,2] [,3]
[1,]    8    1   -1
[2,]    1   -7    2
[3,]    2    1    9
 Solution Vector :
     [,1]
[1,]    1
[2,]    1
[3,]    1
 Right-Hand Side Vector :
     Bvec
[1,]    8
[2,]   -4
[3,]   12
>
```

Figure 66. Jacobi Iteration applied to Example Problem.

5 Non-Linear Systems by Quasi-Newton Method

This chapter formally presents the Newton-Raphson method as a routine to solve systems of non-linear equations. The method is used later in the document to solve for flows and heads in a pipeline network.

Lets return to our previous example where the function **f** is a vector-valued function of a vector argument.

$$\mathbf{f}(\mathbf{x}) = \begin{matrix} f_1 = & x^2 & + y^2 & -4 \\ f_2 = & e^x & + y & -1 \end{matrix} \tag{52}$$

Lets also recall Newtons method for scalar valued function of a single variable.

$$x_{k+1} = x_k - \frac{f(x_k)}{\frac{df}{dx}|_{x_k}} \tag{53}$$

Extending to higher dimensions, the value x become the vector $\mathbf{x}$ and the function $f()$ becomes the vector function $\mathbf{f()}$. What remains is an analog for the first derivative in the denominator (and the concept of division of a matrix).

The analog to the first derivative is a matrix called the Jacobian which is comprised of the first derivatives of the function **f** with respect to the arguments **x**. For example for a 2-value function of 2 arguments (as our example above)

$$\frac{df}{dx}|_{x_k} => \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} \tag{54}$$

Next recall that division is replaced by matrix multiplication with the multiplicative inverse, so the analogy continues as

$$\frac{1}{\frac{df}{dx}|_{x_k}} => \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix}^{-1} \tag{55}$$

Lets name the Jacobian $\mathbf{J(x)}$.

So the multi-variate Newton's method can be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{J}(\mathbf{x})^{-1}|_{x_k} \cdot \mathbf{f}(\mathbf{x})|_{x_k} \tag{56}$$

In the linear systems chapter we did find a way to solve for an inverse, but its not necessary – a series of rearrangement of the system above yields a nice scheme tthat does not require inversion of a matrix.

First, move the $\mathbf{x}_k$ to the left-hand side.

$$\mathbf{x}_{k+1} - \mathbf{x}_k = -\mathbf{J}(\mathbf{x})^{-1}|_{x_k} \cdot \mathbf{f}(\mathbf{x})|_{x_k} \tag{57}$$

Next multiply both sides by the Jacobian.

$$\mathbf{J}(\mathbf{x})|_{x_k} \cdot (\mathbf{x}_{k+1} - \mathbf{x}_k) = -\mathbf{J}(\mathbf{x})|_{x_k} \cdot \mathbf{J}(\mathbf{x})^{-1}|_{x_k} \cdot \mathbf{f}(\mathbf{x})|_{x_k} \tag{58}$$

Recall a matrix multiplied by its inverse returns the identity matrix (the matrix equivalent of unity)

$$-\mathbf{J}(\mathbf{x})|_{x_k} \cdot (\mathbf{x}_{k+1} - \mathbf{x}_k) = \mathbf{f}(\mathbf{x})|_{x_k} \tag{59}$$

So we now have an algorithm:

1. Start with an initial guess $\mathbf{x}_k$, compute $\mathbf{f}(\mathbf{x})|_{x_k}$, and $\mathbf{J}(\mathbf{x})|_{x_k}$.
2. Test for stopping. Is $\mathbf{f}(\mathbf{x})|_{x_k}$ close to zero? If yes, exit and report results, otherwise continue.
3. Solve the linear system $\mathbf{J}(\mathbf{x})|_{x_k} \cdot (\mathbf{x}_{k+1} - \mathbf{x}_k) = \mathbf{f}(\mathbf{x})|_{x_k}$.
4. Test for stopping. Is $(\mathbf{x}_{k+1} - \mathbf{x}_k)$ close to zero? If yes, exit and report results, otherwise continue.
5. Compute the update $\mathbf{x}_{k+1} = \mathbf{x}_k - (\mathbf{x}_{k+1} - \mathbf{x}_k)$, then
6. Move the update into the guess vector $\mathbf{x}_k <= \mathbf{x}_{k+1}$ =and repeat step 1. Stop after too many steps.

Now to repeat the example from the previous chapter, except we will employ this algorithm.

The function (repeated)

$$\mathbf{f}(\mathbf{x}) = \begin{matrix} f_1 = & x^2 & + y^2 & -4 \\ f_2 = & e^x & + y & -1 \end{matrix} \tag{60}$$

Then the Jacobian, here we will compute it analytically because we can

$$\mathbf{J}(\mathbf{x}) => \begin{pmatrix} 2x & 2y \\ \\ e^x & 1 \end{pmatrix} \tag{61}$$

Listing 27 is a listing that implements the Newton-Raphson method with analytical derivatives.

Listing 27. R code demonstrating Newton's Method calculations.

```
# R script for system of non-linear equations using Newton-Raphson with analytical
    derivatives
# forward define the functions
####### f(x) ########################
func <- function(x_vector){
  func <- numeric(0)
  func[1] <- x_vector[1]^2 + x_vector[2]^2 - 4
  func[2] <- exp(x_vector[1]) + x_vector[2] - 1
  return(func)
}
######## J(x) ########################
jacob <- function(x_vector){
  jacob <- matrix(0,nrow=2,ncol=2)
  jacob[1,1] <- 2*x_vector[1]    ; jacob[1,2] <- 2*x_vector[2];
  jacob[2,1] <- exp(x_vector[1]); jacob[2,2] <- 1 ;
  return(jacob)
}
####### Solver Parameters #############
x_guess <- c(2.,-0.8)
tolerancef <- 1e-9  # stop if function gets to zero
tolerancex <- 1e-9  # stop if solution not changing
maxiter <- 20 # stop if too many iterations
x_now <- x_guess
###### Newton-Raphson Algorithm ########
for (iter in 1:maxiter){
  funcNow <- func(x_now)
  testf <- t(funcNow) %*% funcNow
  if(testf < tolerancef){
    message("f(x) is close to zero : ", testf);
    break
  }
  dx <- solve(jacob(x_now),funcNow)
  testx <- t(dx) %*% dx
  if(testx < tolerancex){
    message("solution change small : ", testx);
    break
  }
  x_now <- x_now - dx
}
########################################
if( iter == maxiter) {message("Maximum iterations -- check if solution is converging : ")}
message("Initial Guess"); print(x_guess);
message("Initial Function Value: "); print(func(x_guess));
message("Exit Function Value : ");print(func(x_now));
message("Exit Vector : "); print(x_now)
```

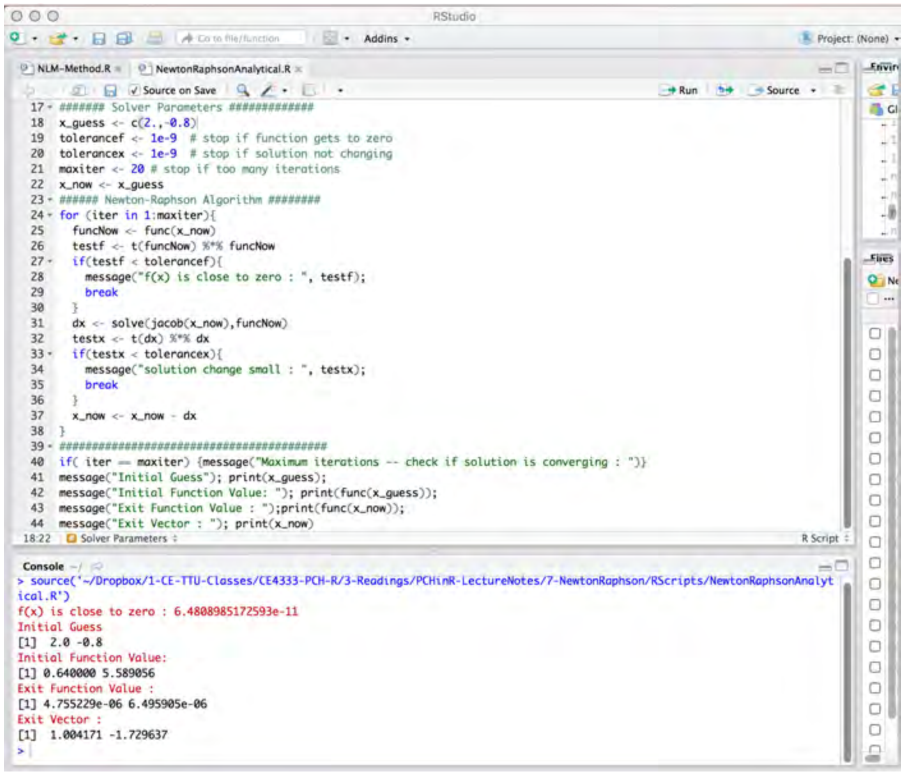
Figure 67 implements the script in Listing 27 for the example problem.

The next variant is to approximate the derivatives – usually a Finite-Difference approximation is used, either forward, backward, or centered differences – generally determined based on the actual behavior of the functions themselves or by trial and error. For really huge systems, we usually make the program itself make the adaptions as it proceeds.

The coding for a finite-difference representation of a Jacobian is shown in Listing 28. In constructing the Jacobian, we observe that each column of the Jacobian is simply the directional derivative of the function with respect to the variable associated with the column. For instance, the first column of the Jacobian in the example is first derivative of the first function (all rows) with respect to the first variable, in this case x. The second column is the first derivative of the second function with respect to the second variable, y. This structure is useful to generalize the Jacobian construction method because we can write (yet another) prototype function that can take the directional derivatives for us, and just insert the returns as columns. The example listing is specific to the 2X2 function in the example, but the extension to more general cases is evident.

Listing 28. R code demonstrating Newton's Method calculations using finite-difference approximations to the partial derivatives.

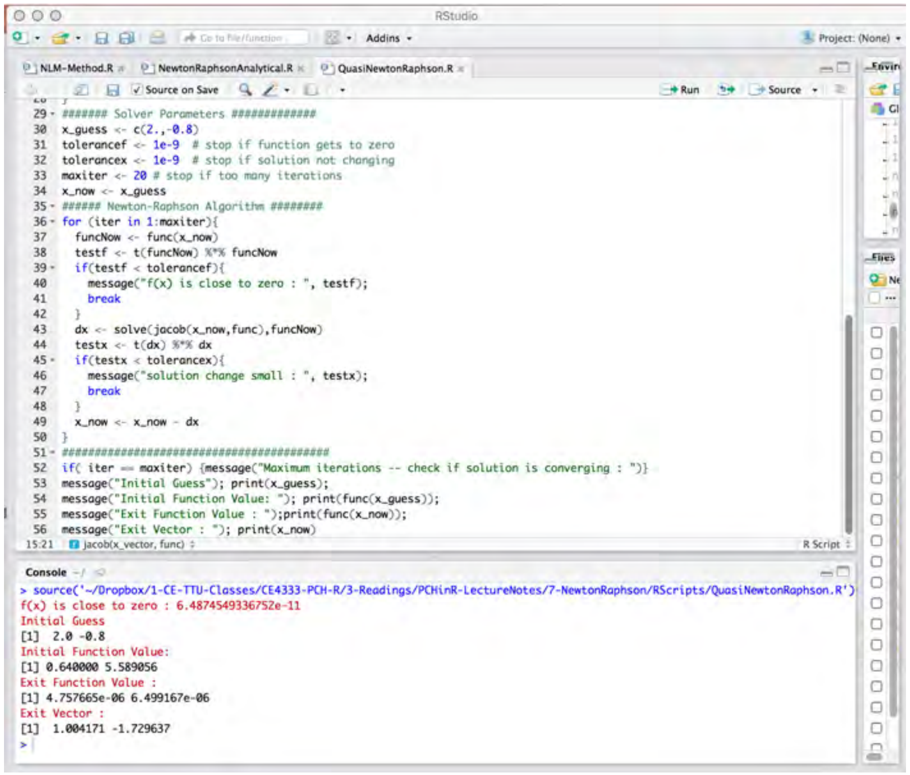
```
# R script for system of non-linear equations using Newton-Raphson with
#    finite-difference approximated derivatives
#    forward define the functions
####### f(x) ########################
func <- function(x_vector){
  func <- numeric(0)
  func[1] <- x_vector[1]^2 + x_vector[2]^2 - 4
  func[2] <- exp(x_vector[1]) + x_vector[2] - 1
  return(func)
}
######## J(x) ########################
jacob <- function(x_vector,func){  #supply a vector and the function name
# the columns of the jacobian are just directional derivatives
  dv <- 1e-06 #perturbation value for finite difference
  df1 <- numeric(0);
  df2 <- numeric(0);
  dxv <- x_vector;
  dyv <- x_vector;
# perturb the vectors
  dxv[1] <- dxv[1]+dv;
  dyv[2] <- dyv[2]+dv;
  df1 <- (func(dxv) - func(x_vector))/dv;
  df2 <- (func(dyv) - func(x_vector))/dv;
  jacob <- matrix(0,nrow=2,ncol=2)
# for a more general case should put this into a loop
  jacob[1,1] <- df1[1]    ;  jacob[1,2] <- df2[1]   ;
  jacob[2,1] <- df1[2]    ;  jacob[2,2] <- df2[2]   ;
  return(jacob)
}
####### Solver Parameters #############
x_guess <- c(2.,-0.8)
tolerancef <- 1e-9  # stop if function gets to zero
tolerancex <- 1e-9  # stop if solution not changing
maxiter <- 20 # stop if too many iterations
x_now <- x_guess
###### Newton-Raphson Algorithm ########
for (iter in 1:maxiter){
  funcNow <- func(x_now)
  testf <- t(funcNow) %*% funcNow
  if(testf < tolerancef){
    message("f(x) is close to zero : ", testf);
    break
  }
  dx <- solve(jacob(x_now,func),funcNow)
  testx <- t(dx) %*% dx
  if(testx < tolerancex){
    message("solution change small : ", testx);
    break
  }
  x_now <- x_now - dx
}
########################################
if( iter == maxiter) {message("Maximum iterations -- check if solution is converging : ")}
message("Initial Guess"); print(x_guess);
message("Initial Function Value: "); print(func(x_guess));
message("Exit Function Value : ");print(func(x_now));
message("Exit Vector : "); print(x_now)
```



```
####### Solver Parameters #############
x_guess <- c(2.,-0.8)
tolerancef <- 1e-9  # stop if function gets to zero
tolerancex <- 1e-9  # stop if solution not changing
maxiter <- 20 # stop if too many iterations
x_now <- x_guess
###### Newton-Raphson Algorithm ########
for (iter in 1:maxiter){
  funcNow <- func(x_now)
  testf <- t(funcNow) %*% funcNow
  if(testf < tolerancef){
    message("f(x) is close to zero : ", testf);
    break
  }
  dx <- solve(jacob(x_now),funcNow)
  testx <- t(dx) %*% dx
  if(testx < tolerancex){
    message("solution change small : ", testx);
    break
  }
  x_now <- x_now - dx
}
###########################################
if( iter == maxiter) {message("Maximum iterations -- check if solution is converging : ")}
message("Initial Guess"); print(x_guess);
message("Initial Function Value: "); print(func(x_guess));
message("Exit Function Value : ");print(func(x_now));
message("Exit Vector : "); print(x_now)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/7-NewtonRaphson/RScripts/NewtonRaphsonAnalyt
ical.R')
f(x) is close to zero : 6.4808985172593e-11
Initial Guess
[1]  2.0 -0.8
Initial Function Value:
[1] 0.640000 5.589056
Exit Function Value :
[1] 4.755229e-06 6.495905e-06
Exit Vector :
[1]  1.004171 -1.729637
>
```

Figure 67. Newton-Raphson using Analytical Derivatives.



```
####### Solver Parameters #############
x_guess <- c(2.,-0.8)
tolerancef <- 1e-9  # stop if function gets to zero
tolerancex <- 1e-9  # stop if solution not changing
maxiter <- 20 # stop if too many iterations
x_now <- x_guess
###### Newton-Raphson Algorithm ########
for (iter in 1:maxiter){
  funcNow <- func(x_now)
  testf <- t(funcNow) %*% funcNow
  if(testf < tolerancef){
    message("f(x) is close to zero : ", testf);
    break
  }
  dx <- solve(jacob(x_now,func),funcNow)
  testx <- t(dx) %*% dx
  if(testx < tolerancex){
    message("solution change small : ", testx);
    break
  }
  x_now <- x_now - dx
}
###########################################
if( iter == maxiter) {message("Maximum iterations -- check if solution is converging : ")}
message("Initial Guess"); print(x_guess);
message("Initial Function Value: "); print(func(x_guess));
message("Exit Function Value : ");print(func(x_now));
message("Exit Vector : "); print(x_now)
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/3-Readings/PCHinR-LectureNotes/7-NewtonRaphson/RScripts/QuasiNewtonRaphson.R')
f(x) is close to zero : 6.4874549336752e-11
Initial Guess
[1]  2.0 -0.8
Initial Function Value:
[1] 0.640000 5.589056
Exit Function Value :
[1] 4.757665e-06 6.499167e-06
Exit Vector :
[1]  1.004171 -1.729637
>
```

Figure 68. Newton-Raphson using Finite-Difference Approximated Derivatives.

6 Pipelines and Networks

Pipe networks, like single path pipelines, are analyzed for head losses in order to size pumps, determine demand management strategies, and ensure minimum pressures in the system. Conceptually the same principles are used for steady flow systems: conservation of mass and energy; with momentum used to determine head losses.

6.1 Pipe Networks – Topology

Network topology refers to the layout and connections. Networks are built of nodes (junctions) and arcs (links).

6.1.1 Continunity (at a node)

Water is considered incompressible in steady flow in pipelines and pipe networks, and the conservation of mass reduces to the volumetric flow rate, Q,

$$Q = AV \tag{62}$$

where A is the cross sectional of the pipe, and V is the mean section velocity. Typical units for discharge is liters per second (lps), gallons per minute (gpm), cubic meters per second (cms), cubic feet per second (cfs), and million gallons per day (mgd). The continuity equation in two cross-sections of a pipe as depicted in Figure 69 is

$$A_1V_1 = A_2V_2 \tag{63}$$

Junctions (nodes) are where two or more pipes join together. A three-pipe junction node with constant external demand is shown in Figure 10. The continuity equation for the junction node is

$$Q_1 - Q_2 - Q_3 - D = 0 \tag{64}$$

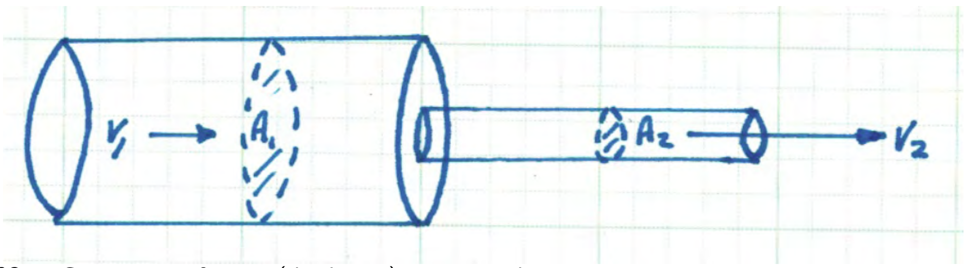


Figure 69. Continuity of mass (discharge) across a change in cross section.

In design analysis, all demands on the system are located at junctions (nodes), and the flow connecting junctions is assumed to be uniform across the cross sections (so

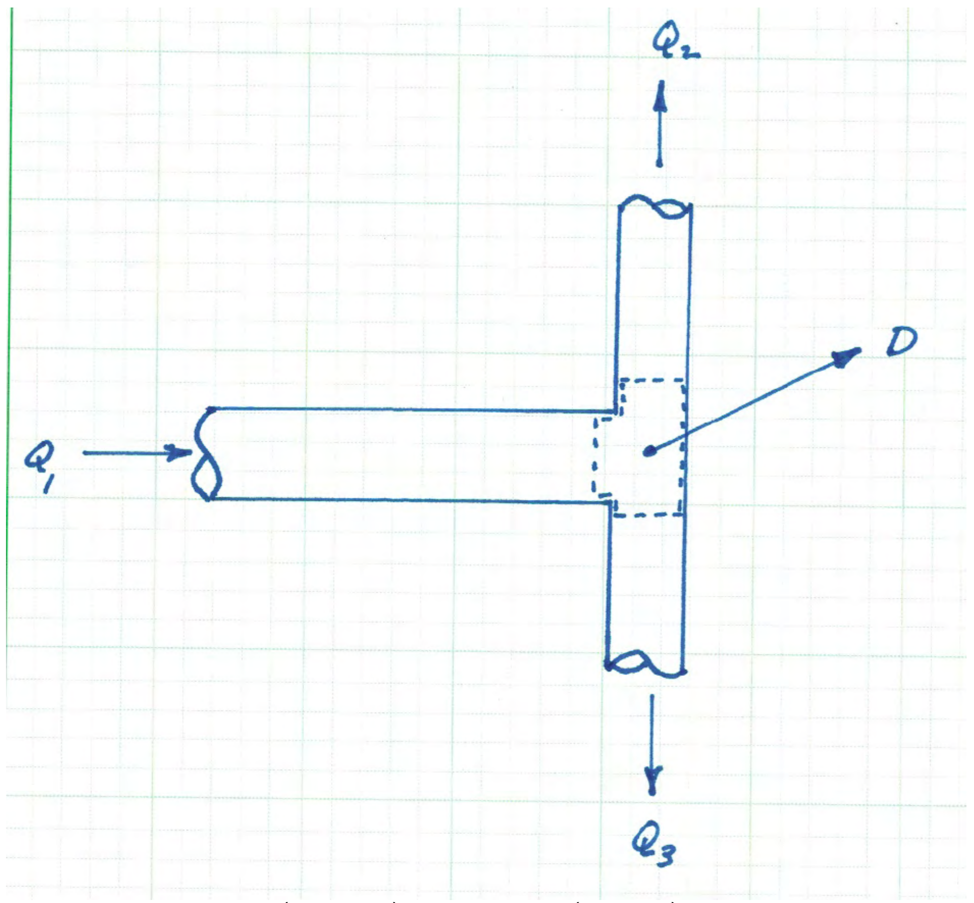


Figure 70. Continuity of mass (discharge) across a node (junction).

that mean velocities apply). If a substantial demand is located between nodes, then an additional node is established at the demand location.

6.1.2 Energy Loss (along a link)

Equation 88 is the one-dimensional steady flow form of the energy equation typically applied for pressurized conduit hydraulics.

$$\frac{p_1}{\rho g} + \alpha_1 \frac{V_1^2}{2g} + z_1 + h_p = \frac{p_2}{\rho g} + \alpha_2 \frac{V_2^2}{2g} + z_2 + h_t + h_l \tag{65}$$

where $\frac{p}{\rho g}$ is the pressure head at a location, $\alpha \frac{V^2}{2g}$ is the velocity head at a location, z is the elevation, h_p is the added head from a pump, h_t is the added head extracted by a turbine, and h_l is the head loss between sections 1 and 2. Figure 76 is a sketch that illustrates the various components in Equation 88.

In network analysis this energy equation is applied to a link that joins two nodes. Pumps and turbines would be treated as separate components (links) and their hy-

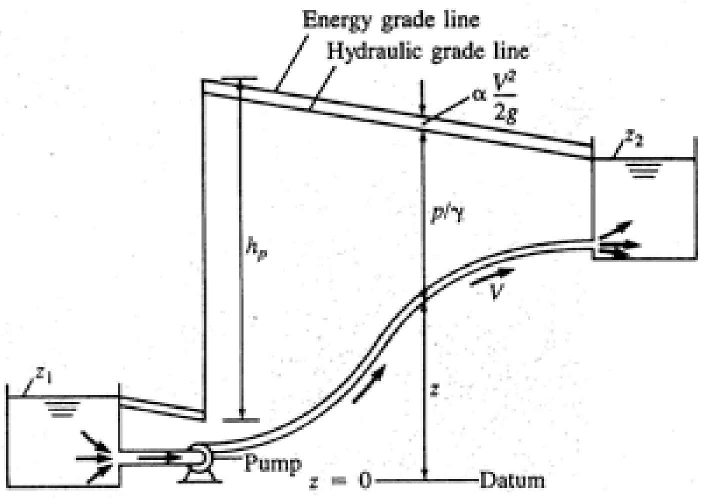


Figure 5-1 Definition sketch for terms in the energy equation

Figure 71. Definition sketch for energy equation.

draulic behavior must be supplied using their respective pump/turbine curves.

6.1.3 Velocity Head

The velocity in $\alpha\frac{V^2}{2g}$ is the mean section velocity and is the ratio of discharge to flow area. The kinetic energy correction coefficient is

$$\alpha = \frac{\int_A u^3 dA}{V^3 A} \tag{66}$$

where u is the point velocity in the cross section (usually measured relative to the centerline or the pipe wall; axial symmetry is assumed). Generally values of α are 2.0 if the flow is laminar, and approach unity (1.0) for turbulent flow. In most water distribution systems the flow is usually turbulent so α is assumed to be unity and the velocity head is simply $\frac{V^2}{2g}$.

6.1.4 Added Head — Pumps

The head supplied by a pump is related to the mechanical power supplied to the flow. Equation 89 is the relationship of mechanical power to added pump head.

$$\eta P = Q\rho g h_p \tag{67}$$

where the power supplied to the motor is P and the "wire-to-water" efficiency is η.

If the relationship is re-written in terms of added head[27] the pump curve is

$$h_p = \frac{\eta P}{Q\rho g} \tag{68}$$

This relationship illustrates that as discharge increases (for a fixed power) the added head decreases. Power scales at about the cube of discharge, so pump curves for computational application typically have a mathematical structure like

$$h_p = H_{\text{shutoff}} - K_{\text{pump}} Q^{\text{exponent}} \tag{69}$$

6.1.5 Extracted Head — Turbines

The head recovered by a turbine is also an "added head" but appears on the loss side of the equation. Equation 96 is the power that can be recovered by a turbine (again using the concept of "water-to-wire" efficiency is

$$P = \eta Q\rho g h_t \tag{70}$$

6.2 Pipe Head Loss Models

The Darcy-Weisbach, Chezy, Manning, and Hazen-Williams formulas are relationships between physical pipe characteristics, flow parameters, and head loss. The Darcy-Weisbach formula is the most consistent with the energy equation formulation being derivable (in structural form) from elementary principles.

$$h_{L_f} = f\frac{L}{D}\frac{V^2}{2g} \tag{71}$$

where h_{L_f} is the head loss from pipe friction, f is a dimensionless friction factor, L is the pipe length, D is the pipe characteristic diameter, V is the mean section velocity, and g is the gravitational acceleration.

The friction factor, f, is a function of Reynolds number Re_D and the roughness ratio $\frac{k_s}{D}$.

$$f = \sigma(Re_D, \frac{k_s}{D}) \tag{72}$$

The structure of σ is determined experimentally. Over the last century the structure is generally accepted to be one of the following depending on flow conditions and pipe properties

1. Laminar flow (Eqn 2.36, pg. 17 Chin (2006)) :

$$f = \frac{64}{Re_D} \tag{73}$$

[27]A negative head loss!

2. Hydraulically Smooth Pipes(Eqn 2.34 pg. 16 Chin (2006)):

$$\frac{1}{\sqrt{f}} = -2log_{10}(\frac{2.51}{Re_d\sqrt{f}}) \tag{74}$$

3. Hydraulically Rough Pipes(Eqn 2.34 pg. 16 Chin (2006)):

$$\frac{1}{\sqrt{f}} = -2log_{10}(\frac{\frac{k_e}{D}}{3.7}) \tag{75}$$

4. Transitional Pipes (Colebrook-White Formula)(Eqn 2.35 pg. 17 Chin (2006)):

$$\frac{1}{\sqrt{f}} = -2log_{10}(\frac{\frac{k_e}{D}}{3.7} + \frac{2.51}{Re_d\sqrt{f}}) \tag{76}$$

5. Transitional Pipes (Jain Formula)(Eqn 2.39 pg. 19 Chin (2006)):

$$f = \frac{0.25}{[log_{10}(\frac{\frac{k_e}{D}}{3.7} + \frac{5.74}{Re_d^{0.9}})]^2} \tag{77}$$

6.3 Pipe Networks Solution Methods

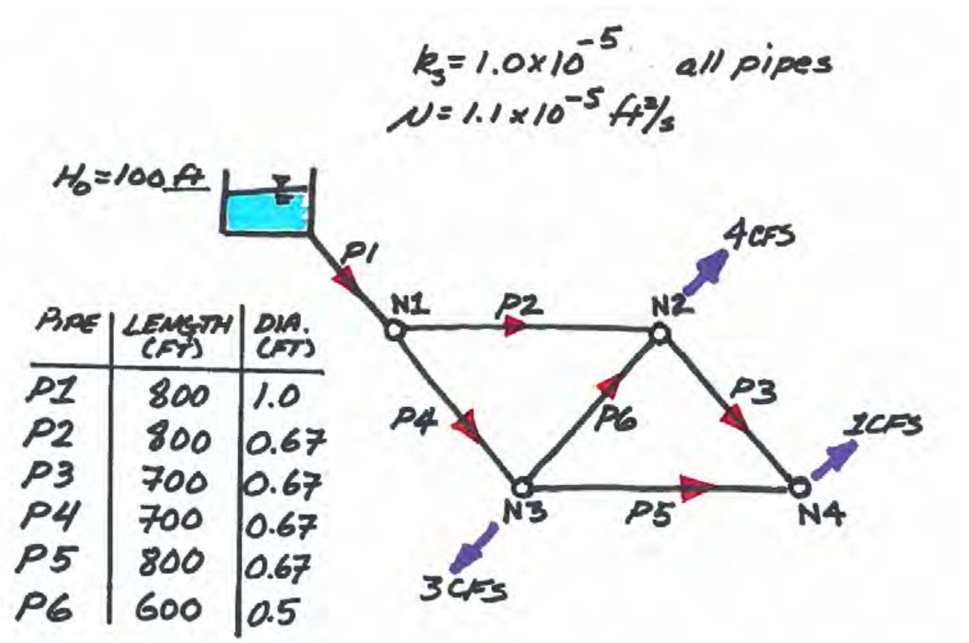
Several methods are used to produce solutions (estimates of discharge, head loss, and pressure) in a network. An early one, that only involves analysis of loops is the Hardy-Cross method. A later one, more efficient, is a Newton-Raphson method that uses node equations to balance discharges and demands, and loop equations to balance head losses. However, a rather ingenious method exists developed by Haman and Brameller (1971), where the flow distribution and head values are determined simultaneously. The task here is to outline the Haman and Brameller (1971) method on the problem below – first some necessary definitions and analysis.

The fundamental procedure is:

1. Continuity is written at nodes (node equations).
2. Energy loss (gain) is written along links (pipe equations).
3. The entire set of equations is solved simultaneously.

6.4 Network Analysis

Figure 72 is a sketch of the problem that will be used. The network supply is the fixed-grade node in the upper left hand corner of the drawing. The remaining nodes (N1 – N4) have demands specified as the purple outflow arrows. The pipes are labeled


$k_s = 1.0\times10^{-5}$ all pipes
$\nu = 1.1\times10^{-5}$ ft²/s
$H_0 = 100$ ft
P1
N1
P2
N2
4CFS
P3
1CFS
P4
P6
N3
P5
N4
3CFS

PIPE	LENGTH (FT)	DIA. (FT)
P1	800	1.0
P2	800	0.67
P3	700	0.67
P4	700	0.67
P5	800	0.67
P6	600	0.5

Figure 72. Pipe network for illustrative example with supply and demands identified. Pipe dimensions and diameters are also depicted..

(P1 – P6), and the red arrows indicate a positive flow direction, that is, if the flow is in the indicated direction, the numerical value of flow (or velocity) in that link would be a positive number.

Define the flows in each pipe and the total head at each node as Q_i and H_i where the subscript indicates the particular component identification. Expressed as a vector, these unknowns are:

$$[Q_1,\ \ Q_2,\ \ Q_3,\ \ Q_4,\ \ Q_5,\ \ Q_6,\ \ H_1,\ \ H_2,\ \ H_3,\ \ H_4] \ = \ \mathbf{x}$$

If we analyze continuity for each node we will have 4 equations (corresponding to each node) for continunity, for instance for Node N2 the equation is

$$Q_2 \quad -Q_3 \qquad\qquad Q_6 \qquad\qquad\qquad = \ 4$$

Similarily if we define head loss in any pipe as $\Delta H_i = f\frac{8L_i}{\pi^2 g D_i^5}|Q_i|Q_i$ or $\Delta H_i = L_i Q_i$, where $L_i = f\frac{8L_i}{\pi^2 g D_i^5}|Q_i|$, then we have 6 equations (corresponding to each pipe) for energy, for instance for Pipe (P2) the equation is[28]

$$-L_2 Q_2 \qquad\qquad\qquad H_1 \quad -H_2 \qquad\qquad = \ 0$$

[28]The seemingly awkward way of writing the equations will become apparent shortly!

If we now write all the node equations then all the pipe equations we could construct the following coefficient matrix below:[29]

$$
\begin{array}{cccccccccc}
\hline
1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
-L_1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & -L_2 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & -L_3 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & -L_4 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & -L_5 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & -L_6 & 0 & -1 & 1 & 0 \\
\hline
\end{array}
$$

Declare the name of this matrix $\mathbf{A(x)}$, where $\mathbf{x}$ denotes the unknown vector of Q augmented by H as above. Next consider the right-hand-side at the correct solution (as of yet still unknown!) as

$$[0, \quad 4, \quad 3, \quad 1, \quad -100, \quad 0, \quad 0, \quad 0, \quad 0, \quad 0] = \mathbf{b}$$

So if the coefficient matrix is correct then the following system would result:

$$\mathbf{A}(\mathbf{x}) \cdot \mathbf{x} = \mathbf{b}$$

which would look like

$$
\begin{pmatrix}
1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
-L_1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & -L_2 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & -L_3 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & -L_4 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & -L_5 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & -L_6 & 0 & -1 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ H_1 \\ H_2 \\ H_3 \\ H_4
\end{pmatrix}
=
\begin{pmatrix}
0 \\ 4 \\ 3 \\ 1 \\ \hline -100 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix}
\tag{78}
$$

Observe, the system is non-linear because the coefficient matrix depends on the current values of Q_i for the L_i terms. However, the system is full-rank (rows == columns) so it is a candidate for Newton-Raphson.

[29]The horizontal lines divide the node and the pipe equations. The upper partition are the node equations in Q and H, the lower partition are the pipe equations in Q and H

Further observe that the upper partition from column 6 and smaller is simply the node-arc incidence matrix, and the lower partition for the same columns only contains L_i terms on its diagonal, the remainder is zero. Next observe that the partition associated with heads in the node equations is the zero-matrix.

Lastly (and this is important!) the lower right partition is the transpose of the node-arc incidence matrix subjected to scalar multiplication of -1. The importance is that all the information needed to find a solution is contained in the node-arc incidence matrix and the right-hand-side – the engineer does not need to identify closed loops (nor does the computer need to find closed loops).

The trade-off is a much larger system of equations, however solving large systems is far easier that searching a directed graph to identify closed loops, furthermore we obtain the heads as part of the solution process.

7 Pipelines Network Analysis

The prior chapter introduced the non-linear system that results from the analysis of the pipeline network. This chapter continues the effort and produces a workable **R** script that can compute flows and heads given just the node-arc incidence matrix, and pipe properties.

Recall from the prior chapter the non-linear system to be solved is

$$\mathbf{A}(\mathbf{x}) \cdot \mathbf{x} = \mathbf{b}$$

which would look like

$$\left(\begin{array}{cccccccccc}
1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
\hline
-L_1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
0 & -L_2 & 0 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\
0 & 0 & -L_3 & 0 & 0 & 0 & 0 & 1 & 0 & -1 \\
0 & 0 & 0 & -L_4 & 0 & 0 & 1 & 0 & -1 & 0 \\
0 & 0 & 0 & 0 & -L_5 & 0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & 0 & 0 & -L_6 & 0 & -1 & 1 & 0
\end{array}\right)
\begin{pmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ Q_5 \\ Q_6 \\ H_1 \\ H_2 \\ H_3 \\ H_4 \end{pmatrix}
=
\left(\begin{array}{c} 0 \\ 4 \\ 3 \\ 1 \\ \hline -100 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right) \tag{79}$$

The system is non-linear because the coefficient matrix depends on the current values of Q_i for the L_i terms. The upper partition from column 6 and smaller is simply the node-arc incidence matrix, and the lower partition for the same columns only contains L_i terms on its diagonal, the remainder is zero. Next observe that the partition associated with heads in the node equations is the zero-matrix. The lower right partition is the transpose of the node-arc incidence matrix subjected to scalar multiplication of -1. So using the Newton-Raphson approach discussed earlier we develop a script in **R** that produces estimates of discharge and total head in the system depicted in Figure 72.

7.1 Script Structure

The script will need to accomplish several tasks including reading the node-arc incidence matrix supplied as the file in Figure 73 and convert the strings into numeric values. The script will also need some support functions defined before constructing the matrix.

The rows of the input file are:

```
4
6
1.00 0.67 0.67 0.67 0.67 0.5
800 800 700 700 800 600
0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
0.000011
1 1 1 1 1 1
1 -1 0 -1 0 0
0 1 -1 0 0 1
0 0 0 1 -1 -1
0 0 1 0 1 0
0 4 3 1 -100 0 0 0 0 0
```

Figure 73. Input file for the problem.

1. The node count.
2. The pipe count.
3. Pipe diameters, in feet.
4. Pipe lengths, in feet.
5. Pipe roughness heights, in feet.
6. Kinematic viscosity in feet^2/second.
7. Initial guess of flow rates (unbalanced OK, non-zero vital!)
8. The next four rows are the node-arc incidence matrix.
9. The last row is the demand (and fixed-grade node total head) vector.

7.1.1 Support Functions

The Reynolds number will need to be calculated for each pipe at each iteration of the solution, so a Reynolds number function will be useful. For circular pipes, the following equation should work,

$$Re(Q) = \frac{4}{\mu \pi D}|Q| \tag{80}$$

The Jain equation (Jain, 1976) that directly computes friction factor from Reynolds number, diameter, and roughness is

$$f(k_s, D, Re) = \frac{0.25}{[log(\frac{k_s}{3.7D} + \frac{5.74}{Re^{0.9}})]^2} \tag{81}$$

Once you have the Reynolds number for a pipe, and the friction factor, then the head loss factor that will be used in the coefficient matrix (and the Jacobian) is

$$L_i = f\frac{8L_i}{\pi^2 g D_i^5}|Q_i| \tag{82}$$

These three support functions are coded in **R** as shown in Listing 29.

Listing 29. R Code to compute Reynolds numbers and friction factors

.

```
##############################################################
##############  Forward Define Support Functions   #################
##############################################################
# Jain Friction Factor Function -- Tested OK 23SEP16
friction_factor <- function(roughness,diameter,reynolds){
  temp1 <- roughness/(3.7*diameter);
  temp2 <- 5.74/(reynolds^(0.9));
  temp3 <- log10(temp1+temp2);
  temp3 <- temp3^2;
  friction_factor <- 0.25/temp3;
  return(friction_factor)
}
# Velocity Function
velocity <- function(diameter,discharge){
  velocity <- discharge/(0.25*pi*diameter^2)
  return(velocity)
}
# Reynolds Number Function
reynolds_number <- function(velocity,diameter,mu){
  reynolds_number <- abs(velocity)*diameter/mu
  return(reynolds_number)
}
# Geometric factor function
k_factor <- function(howlong,diameter,gravity){
  k_factor <- (16*howlong)/(2.0*gravity*pi^2*diameter^5)
  return(k_factor)
}
```

7.1.2 Augmented and Jacobian Matrices

The $\mathbf{A(x)}$ is built using the node-arc incidence matrix (which does not change), and the current values of L_i. You will also need to build the Jacobian of $\mathbf{A(x)}$ to implement the update as-per Newton-Raphson.

A brief review; at the solution we can write

$$[\mathbf{A}(\mathbf{x})] \cdot \mathbf{x} - \mathbf{b} = \mathbf{f}(\mathbf{x}) = \mathbf{0} \tag{83}$$

Lets assume we are not at the solution, so we need a way to update the current value of $\mathbf{x}$. Recall from Newton's method (for univariate cases) that the update formula is

$$x_{k+1} = x_k - (\frac{df}{dx}\mid_{x_k})^{-1} f(x_k) \tag{84}$$

The Jacobian will play the role of the derivative, and $\mathbf{x}$ is now a vector (instead of a single variable). Division is not defined for matrices, but the multiplicative inverse is (the inverse matrix), and plays the role of division. Hence, the extension to the pipeline case is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}(\mathbf{x}_k)]^{-1}\mathbf{f}(\mathbf{x}_k) \tag{85}$$

where $\mathbf{J}(\mathbf{x}_k)$ is the Jacobian of the coefficient matrix $\mathbf{A}$ evaluated at $\mathbf{x}_k$. Although a bit cluttered, here is the formula for a single update step, with the matrix, demand vector, and the solution vector in their proper places.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}(\mathbf{x}_k)]^{-1}\{[\mathbf{A}(\mathbf{x}_k)] \cdot \mathbf{x}_k - \mathbf{b}\} \tag{86}$$

As a practical matter we actually never invert the Jacobian[30], instead we solve the related Linear system of

$$[\mathbf{J}(\mathbf{x}_k)] \cdot \Delta\mathbf{x} = \{[\mathbf{A}(\mathbf{x}_k)] \cdot \mathbf{x}_k - \mathbf{b}\} \tag{87}$$

for $\Delta\mathbf{x}$, then perform the update as $\mathbf{x}_{k+1} = \mathbf{x}_k$ - $\Delta\mathbf{x}$

The Jacobian of the pipeline model is a matrix with the following properties:

1. The partition of the matrix that corresponds to the node formulas (upper left partition) is identical to the original coefficient matrix — it will be comprised of 0 or $\pm$ 1 in the same pattern at the equivalent partition of the $\mathbf{A}$ matrix.
2. The partition of the matrix that corresponds to the pipe head loss terms (lower left partition), will consist of values that are twice the values of the coefficients in the original coefficient matrix (at any supplied value of $\mathbf{x}_k$.
3. The partition of the matrix that corresponds to the head terms (lower right partition), will consist of values that are identical to the original matrix.
4. The partition of the matrix that corresponds to the head coefficients in the node equations (upper right partition) will also remain unchanged.

You will want to take advantage of problem structure to build the Jacobian (you could just finite-difference the coefficient matrix to approximate the partial derivatives, but that is terribly inefficient if you already know the structure).

7.1.3 Stopping Criteria, and Solution Report

You will need some way to stop the process – the three most obvious (borrowed from Newton's method) are:

1. Approaching the correct solution (e.g. $[\mathbf{A}(\mathbf{x})] \cdot \mathbf{x} - \mathbf{b} = \mathbf{f}(\mathbf{x}) = \mathbf{0}$).

[30]Inverting the matrix every step is computationally inefficient, and unnecessary. As an example, solving the system in this case would at worst take 10 row operations each step, but nearly 100 row operations to invert at each step – to accomplish the same result, generate an update. Now imagine when there are hundreds of nodes and pipes!

2. Update vector is not changing (e.g. $\mathbf{x}_{k+1} = \mathbf{x}_k$), so either have an answer, or the algorithm is stuck.

3. You have done a lot of iterations (say 100).

Listing 30 is a code fragment to find the flow distribution and heads for the example problem. Not listed is the forward defined functions already listed above – these should be placed into the script in the location shown (or directly sourced into the code in **R**).

Listing 30. R Code to Implement Pipe Network Solution
This fragment reads the data file and converts it into numeric values and reports back the values.

```
# Steady Flow in a Pipe Network Using Hybrid Method (and Newton-Raphson) based on
# Haman YM, Brameller A. Hybrid method for the solution of piping networks. Proc IEEE
    1971;118(11):1607?12.
#
# Clear all existing objects
 rm(list=ls())
################################################################
##############Forward Define Support Functions Go Here ##########
################################################################
# Read Input Data Stream from File
zz <- file("PipeNetwork.txt", "r") # Open a connection named zz to file named PipeNetwork.
    txt
nodeCount <- as.numeric(readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown",
    skipNul = FALSE))
pipeCount <-as.numeric(readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown",
    skipNul = FALSE))
diameter <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul =
    FALSE))
distance <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul =
    FALSE))
roughness <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul =
    FALSE))
viscosity <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul =
    FALSE))
flowguess <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul =
    FALSE))
nodearcs <- (readLines(zz, n = nodeCount, ok = TRUE, warn = TRUE,encoding = "unknown",
    skipNul = FALSE))
rhs_true <- (readLines(zz, n = pipeCount+nodeCount, ok = TRUE, warn = TRUE,encoding = "
    unknown", skipNul = FALSE))
close(zz) # Close connection zz
#
# Convert Input Stream into Numeric Structures
diameter <-as.numeric(unlist(strsplit(diameter,split=" ")))
distance <-as.numeric(unlist(strsplit(distance,split=" ")))
roughness <-as.numeric(unlist(strsplit(roughness,split=" ")))
viscosity <-as.numeric(unlist(strsplit(viscosity,split=" ")))
flowguess <-as.numeric(unlist(strsplit(flowguess,split=" ")))
nodearcs <-as.numeric(unlist(strsplit(nodearcs,split=" ")))
rhs_true <-as.numeric(unlist(strsplit(rhs_true,split=" ")))
# convert nodearcs a matrix
# We will need to augment this matrix for the actual solution -- so after augmentation will
     deallocate the memory
nodearcs <-matrix(nodearcs,nrow=nodeCount,ncol=pipeCount,byrow = TRUE)
# echo input
message("Node Count = ",nodeCount)
message("Pipe Count = ",pipeCount)
message("Pipe Lengths = "); distance
message("Pipe Diameters = "); diameter
message("Pipe Roughness = "); roughness
message("Fluid Viscosity = ",viscosity)
message("Initial Guess = "); flowguess
message("Node-Arc-Incidence Matrix = "); nodearcs
#
```

Listing 31 is a code fragment to construct the coefficient matrix structure for the non-changing part and allocate variables for the Newton-Raphson method.

Listing 31. R Code to Implement Pipe Network Solution
This fragment constructs the initial **A(x)** matrix and allocates variables used in the iteration loop.

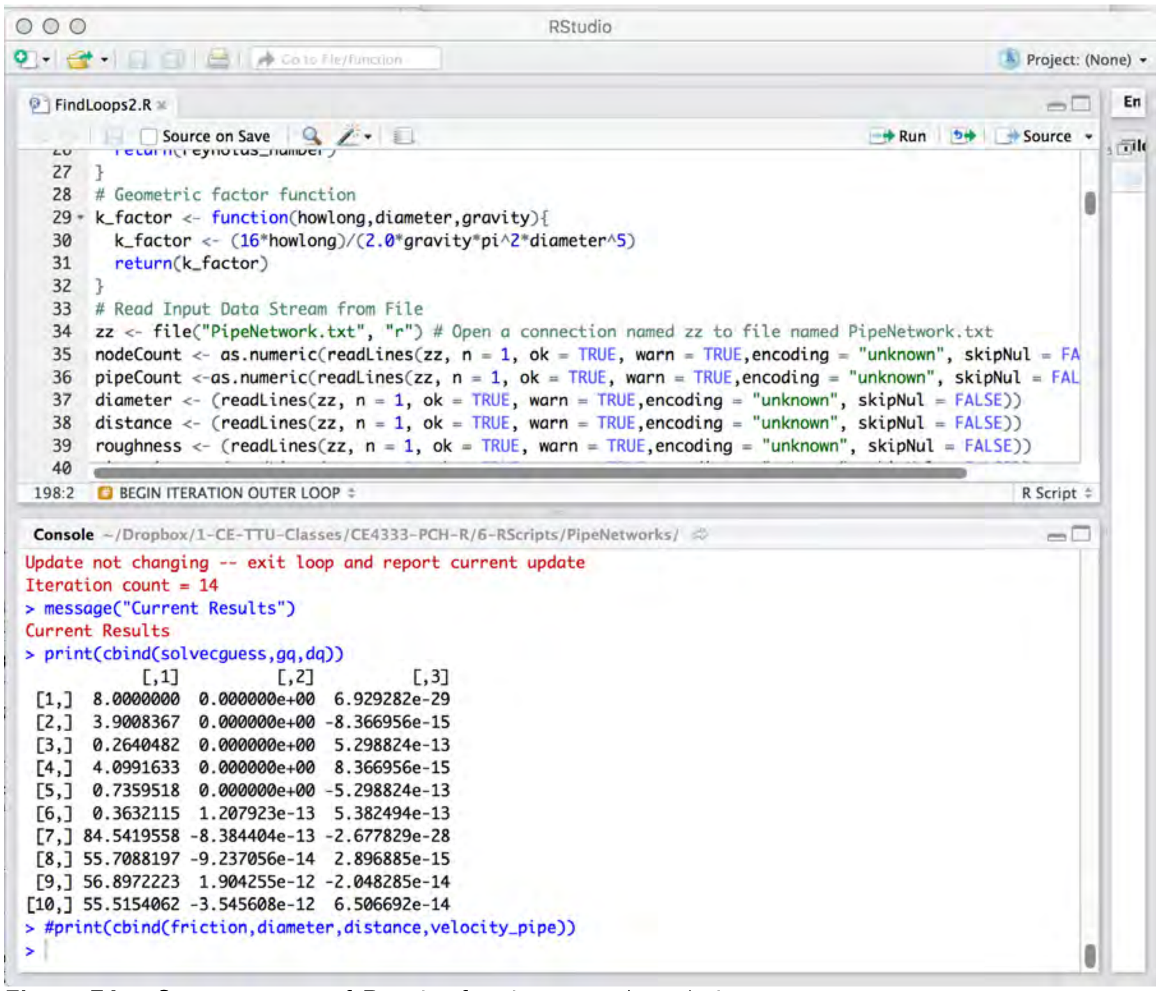
```
# create the augmented matrix
headCount <- nodeCount
flowCount <- pipeCount
augmentedRowCount <- nodeCount+pipeCount
augmentedColCount <- flowCount+headCount
augmentedMat <- matrix(0,nrow=augmentedRowCount,ncol=augmentedColCount,byrow = TRUE)
#
augmentedMat
# build upper left partition of matrix -- this partition is constants from node-arc matrix
for (i in 1:nodeCount){
  for (j in 1:flowCount){
    augmentedMat[i,j] <- nodearcs[i,j]
  }
}
augmentedMat
# build lower right partition of matrix -- this partition is -1*transpose(node-arc) matrix
istart <- nodeCount+1
iend <- nodeCount+pipeCount
jstart <- flowCount+1
jend <- flowCount+headCount
for (i in istart:iend ){
  for(j in jstart:jend ){
    augmentedMat[i,j] <- -1*nodearcs[j-jstart+1,i-istart+1]
  }
}
augmentedMat
# here it should be safe to delete the nodearc matrix
rm(nodearcs)
# Need some vorking vectors
 HowMany <- 50
 tolerance1 <- 1e-24
 tolerance2 <- 1e-24
 velocity_pipe <-numeric(0)
 reynolds <- numeric(0)
 friction <- numeric(0)
 geometry <- numeric(0)
 lossfactor <- numeric(0)
 jacbMatrix <- matrix(0,nrow=augmentedRowCount,ncol=augmentedColCount,byrow = TRUE)
 gq <- numeric(0)
 solvecguess <- numeric(length=augmentedRowCount)
 solvecnew <- numeric(length=augmentedRowCount)
 solvecguess[1:flowCount] <- flowguess[1:flowCount]

 # compute geometry factors (only need once, goes outside iteration loop)
 for (i in 1:pipeCount)
 {
   geometry[i] <- k_factor(distance[i],diameter[i],32.2)
 }
 geometry
```

Listing 32 is the code fragment that implements the iteration loop of the Newton-Raphson method. Within each iteration, the support functions are repeatedly used to construct the changing part of the coefficient and Jacobian matrices, solving the resulting linear system, performing the vector update, and testing for stopping.

Listing 32. R Code to Implement Pipe Network Solution
This fragment executes the iteration loop where the Newton-Raphson method and updates are implemented.

```
 # going to wrap below into an interation loop -- forst a single instance
for (iteration in 1:HowMany){
################### BEGIN ITERATION OUTER LOOP ##########################
 # compute current velocity
 for (i in 1:pipeCount)
 {
   velocity_pipe[i]<-velocity(diameter[i],flowguess[i])
 }
 # compute current reynolds
 for (i in 1:pipeCount)
 {
   reynolds[i]<-reynolds_number(velocity_pipe[i],diameter[i],viscosity)
 }
 # compute current friction factors
 for (i in 1:pipeCount)
 {
   friction[i]<-friction_factor(roughness[i],diameter[i],reynolds[i])
 }
 # compute current loss factor
 for (i in 1:pipeCount)
 {
   lossfactor[i] <- friction[i]*geometry[i]*abs(flowguess[i])
 }
 # build the function matrix
 # operate on the lower left partition of the matrix
 istart <- nodeCount+1
 iend <- nodeCount+pipeCount
 jstart <- 1
 jend <- flowCount
 for (i in istart:iend ){
   for(j in jstart:jend ){
     if ((i-istart+1) == j)  augmentedMat[i,j] <- -1*lossfactor[j]
   }
 }
 # now build the current jacobian
 # slick trick -- we will copy the current function matrix, then modify the lower left
     partition
 jacbMatrix <- augmentedMat
 # build the function matrix
 # operate on the lower left partition of the matrix
 istart <- nodeCount+1
 iend <- nodeCount+pipeCount
 jstart <- 1
 jend <- flowCount
 for (i in istart:iend ){
   for(j in jstart:jend ){
     if ((i-istart+1) == j)  jacbMatrix[i,j] <- 2*jacbMatrix[i,j]
   }
 }
# now build the gq() vector
gq <- augmentedMat %*% solvecguess - rhs_true
gq
dq <- solve(jacbMatrix,gq)
# update the solution vector
solvecnew <- solvecguess - dq
solvecnew
# # now test for stopping
test <- abs(solvecnew - solvecguess)
if( t(test) %*% test < tolerance1){
  message("Update not changing -- exit loop and report current update")
  message("Iteration count = ",iteration)
  solvecguess <- solvecnew
  flowguess[1:flowCount] <- solvecguess[1:flowCount]
  break
}
test <- abs(gq)
if( t(test) %*% test < tolerance2 ){
  message("G(Q) close to zero -- exit loop and report current update")
  message("Iteration count = ",iteration)
  solvecguess <- solvecnew
  flowguess[1:flowCount] <- solvecguess[1:flowCount]
  break
}
solvecguess <- solvecnew
flowguess[1:flowCount] <- solvecguess[1:flowCount]
################### END OF ITERATION OUTER LOOP ############################
}
message("Current Results")
print(cbind(solvecguess,gq,dq))
print(cbind(friction,diameter,distance,velocity_pipe))
```

Figure 74 is a screen capture of the script running the example problem. The first column in the output is the solution vector. The first 6 rows are the flows in pipes P1-P6. The remaining 4 rows are the heads at nodes N1-N4.


RStudio
Go to file/function
Project: (None)
FindLoops2.R
Source on Save
Run
Source
198:2
BEGIN ITERATION OUTER LOOP
R Script
Console ~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/6-RScripts/PipeNetworks/

```
}
# Geometric factor function
k_factor <- function(howlong,diameter,gravity){
  k_factor <- (16*howlong)/(2.0*gravity*pi^2*diameter^5)
  return(k_factor)
}
# Read Input Data Stream from File
zz <- file("PipeNetwork.txt", "r") # Open a connection named zz to file named PipeNetwork.txt
nodeCount <- as.numeric(readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul = FA
pipeCount <-as.numeric(readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul = FAL
diameter <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul = FALSE))
distance <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul = FALSE))
roughness <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul = FALSE))
```

```
Update not changing -- exit loop and report current update
Iteration count = 14
> message("Current Results")
Current Results
> print(cbind(solvecguess,gq,dq))
            [,1]          [,2]          [,3]
 [1,]  8.0000000  0.000000e+00  6.929282e-29
 [2,]  3.9008367  0.000000e+00 -8.366956e-15
 [3,]  0.2640482  0.000000e+00  5.298824e-13
 [4,]  4.0991633  0.000000e+00  8.366956e-15
 [5,]  0.7359518  0.000000e+00 -5.298824e-13
 [6,]  0.3632115  1.207923e-13  5.382494e-13
 [7,] 84.5419558 -8.384404e-13 -2.677829e-28
 [8,] 55.7088197 -9.237056e-14  2.896885e-15
 [9,] 56.8972223  1.904255e-12 -2.048285e-14
[10,] 55.5154062 -3.545608e-12  6.506692e-14
> #print(cbind(friction,diameter,distance,velocity_pipe))
>
```

Figure 74. Screen capture of **R** script for pipe network analysis.

7.2 Exercise Set

1. Figure 75 is a six-pipe network with a water supply source at Node 0, and demands at Nodes 1-5. Table 5 is a listing of the node and pipe data.

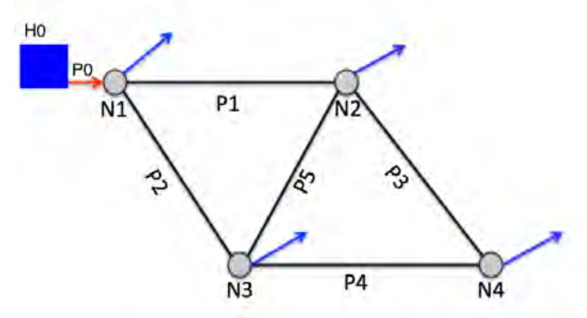


Figure 75. Layout of Simple Network.

Table 5. Node and Pipe Data.

Pipe ID	Diameter (mm)	Length (m)	Roughness (mm)
P0	900	100	0.1
P1	200	800	0.1
P2	200	700	0.1
P3	200	700	0.1
P4	200	800	0.1
P5	150	600	0.1
Node ID	**Demand (LPS)**	**Elevation (meters)**	**Head (meters)**
N0	0.0	0.0	100
N1	2.0	0.0	?
N2	4.0	0.0	?
N3	3.0	0.0	?
N4	1.0	0.0	?

Code the script, build an input file, and determine the flow distribution In your solution you are to supply

(a) An analysis showing the development of the node-arc incidence matrix based on the flow directions in Figure 75,

(b) The input file you constructed to provide the simulation values to your script, and

(c) A screen capture (or output file) showing the results.

2. Repeat the analysis, but adjust the value of head at Node 0 so that the value of head at Node 1 is 100 meters and the Node elevations are those in Table 6. Modify the script to compute and report node pressures.

Table 6. Node Data.

Node ID	Demand (LPS)	Elevation (meters)	Head (meters)
N0	0.0	30.0	?
N1	2.0	28.0	100
N2	4.0	20.0	?
N3	3.0	20.0	?
N4	1.0	18.0	?

8 Pumps and Valves

The addition of pumps, turbines, and valves increases some of the complexity for a network simulator. Valves and other fittings like elbows and such, that have a fixed setting are modeled as links and the resulting equations look much like pipe loss equations.

Pumps while also logically categorized as links are more complex because their head loss behavior is firstly negative – that is they add head to a flow system, and their ability to actually function is governed by their own performance curve. First we will reviwe the modified Bernoulli equation again and then construct a prototype pump function to add to the program and simulate pump performance.

8.1 Energy Loss (along a link)

Equation 88 is the one-dimensional steady flow form of the energy equation typically applied for pressurized conduit hydraulics.

$$\frac{p_1}{\rho g} + \alpha_1 \frac{V_1^2}{2g} + z_1 + h_p = \frac{p_2}{\rho g} + \alpha_2 \frac{V_2^2}{2g} + z_2 + h_t + h_l \tag{88}$$

where $\frac{p}{\rho g}$ is the pressure head at a location, $\alpha \frac{V^2}{2g}$ is the velocity head at a location, z is the elevation, h_p is the added head from a pump, h_t is the added head extracted by a turbine, and h_l is the head loss between sections 1 and 2. Figure 76 is a sketch that illustrates the various components in Equation 88.

In network analysis this energy equation is applied to a link that joins two nodes. Pumps and turbines would be treated as separate components (links) and their hydraulic behavior must be supplied using their respective pump/turbine curves.

8.1.1 Added Head — Pumps

The head supplied by a pump is related to the mechanical power supplied to the flow. Equation 89 is the relationship of mechanical power to added pump head.

$$\eta P = Q \rho g h_p \tag{89}$$

where the power supplied to the motor is P and the "wire-to-water" efficiency is η.

If the relationship is re-written in terms of added head[31] the pump curve is

$$h_p = \frac{\eta P}{Q \rho g} \tag{90}$$

[31] A negative head loss!

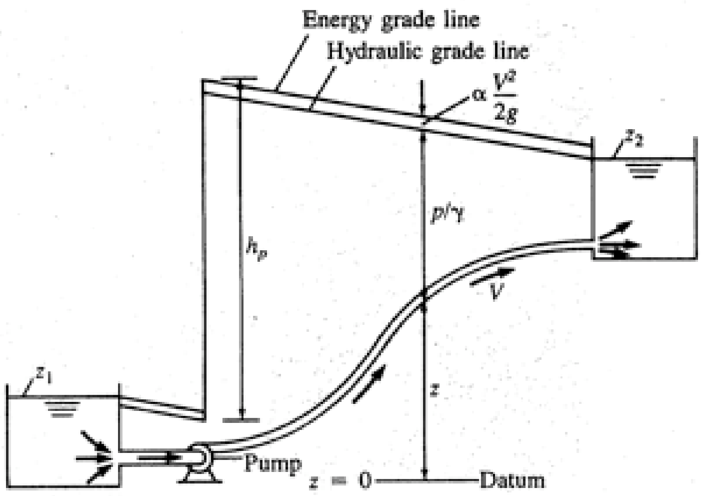


Figure 5-1 Definition sketch for terms in the energy equation

Figure 76. Definition sketch for energy equation.

Figure 77 is a typical pump curve depicting the kind of information available from a manufacturer of a pump.

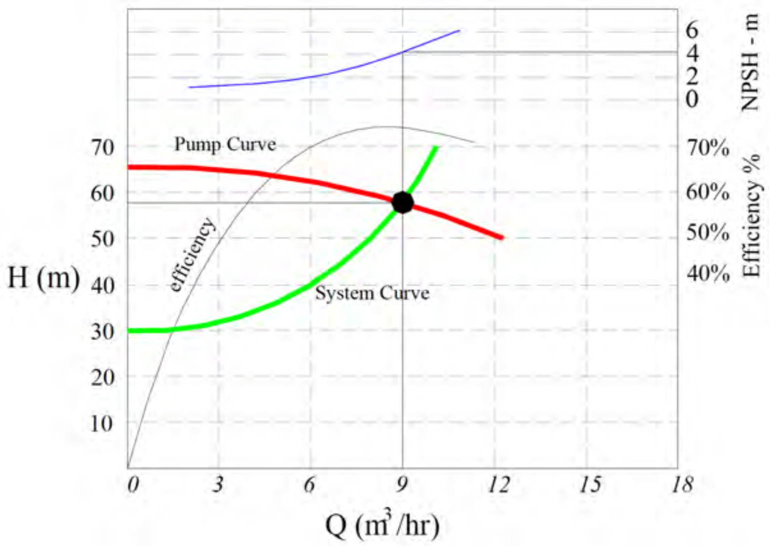


Figure 77. Pump Curve.

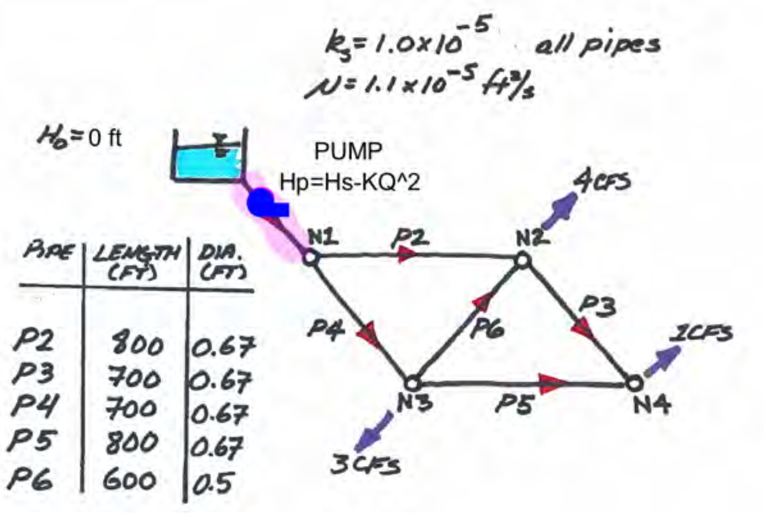
In introductory fluid mechanics we spend effort to match the pump curve to the system curve (head losses in our distribution system) and that match tells us how the pump-system combination should function. The pump curve relationship, as well as Equation 90, illustrates that as discharge increases (for a fixed power) the added head decreases. Power scales at about the cube of discharge, so pump curves for computational application typically have a mathematical structure like

$$h_p = H_{\text{shutoff}} - K_{\text{pump}} Q^{\text{exponent}} \tag{91}$$

In computational hydraulics we will need to represent the added as a head loss term (with opposite sign), and the functional form represented by Equation 91 is a good starting point. Practical (professional) programs will allow the curve to be represented in a tabular form and will use interpolation (just like our examples earlier) to specify the added head at a particular flow rate.

The next example will illustrate how to add pumps into the model.

Example 1: Pipe network with pumps
Figure 78 is a sketch of the problem that will be used. The network supply is the fixed-grade node in the upper left hand corner of the drawing – in this example its head is set at zero. The remaining nodes (N1 – N4) have demands specified as the purple outflow arrows. The pipes are labeled (P2 – P6), and the red arrows indicate a positive flow direction, that is, if the flow is in the indicated direction, the numerical value of flow (or velocity) in that link would be a positive number. The pump replaces


$k_s = 1.0 \times 10^{-5}$ all pipes
$\nu = 1.1 \times 10^{-5}$ ft²/s
H_0 = 0 ft
PUMP
Hp=Hs-KQ^2
N1
P2
N2
4CFS
P3
P4
P6
1CFS
N3
P5
N4
3CFS

PIPE	LENGTH (FT)	DIA. (FT)
P2	800	0.67
P3	700	0.67
P4	700	0.67
P5	800	0.67
P6	600	0.5

Figure 78. Pipe Network with a Pump.

pipe (P1) from the previous version of this example. We will use the observation that we really only need to identify which links are pumps, substitute in the correct added head component and then solve the system as in the earlier example.

We have to specify how the pump curve will be represented. In this example we will use a functional form.

$$h_p(Q) = H_{shutoff} - K_{pump} \times Q^n \tag{92}$$

For this example we will use the following numerical values for the pump function: $H_{shutoff} = 104.54$ feet, $K_{pump} = 0.25$ feet$/cfs^2$, and $n = 2$.

The $h_p(Q)$ is actually written as an added head factor, just like the friction factor, so we will use absolute values of flow so the term at each computational step can be placed in the augmented maxrix as if it were a head loss term; the solver will not know the difference.

The actual functional form employed is

$$h_p(Q) = [H_{shutoff}/|Q| - K_{pump} \times |Q|]Q \tag{93}$$

As before the sign of Q at the solution conveys flow direction. The program example does not trap the potential divide by zero error $H_{shutoff}/|Q|$, but one could test for zero flow, and just apply the shutoff head. Listing 33 implements the prototype function described above.

Listing 33. R Code to pump prototype function
.

```
.....
# Pump Curve factor function
p_factor <- function(shutoff,constant,exponent,flow){
  p_factor <- shutoff/abs(flow) - constant*abs(flow^(exponent-1))
  return(p_factor)
}
```

Next we have to read in the pump characteristics, I decided to just have pumps replace links (so I won't have to rebuild a node-arc-incidence matrix), so the pump characteristics are

1. Link ID – the index of the pipe that is replaced by a pump.
2. Shutoff head.
3. K_{pump}.
4. Exponent on the pump curve, n. Typically it will be larger than 1.0.

Listing 34 implements the reads from the input file, and builds the pump matrix.

Listing 34. R Code to include pumps in a pipeline network
.

```
# Read Input Data Stream from File
zz <- file("PipeNetwork.txt", "r") # Open a connection named zz to file named PipeNetwork.
     txt
pumpCount <- as.numeric(readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown",
     skipNul = FALSE))
nodeCount <- as.numeric(readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown",
     skipNul = FALSE))
.....
rhs_true <- (readLines(zz, n = 1, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul =
     FALSE))
pumps <- (readLines(zz, n = pumpCount, ok = TRUE, warn = TRUE,encoding = "unknown", skipNul
      = FALSE))
close(zz) # Close connection zz
.....
pumps <-as.numeric(unlist(strsplit(pumps,split=" ")))
# convert nodearcs a matrix
# We will need to augment this matrix for the actual solution -- so after augmentation will
     deallocate the memory
nodearcs <-matrix(nodearcs,nrow=nodeCount,ncol=pipeCount,byrow = TRUE)
pumps <-matrix(pumps,nrow=pumpCount,ncol=4,byrow=TRUE)
.....
```

Next we will have to compute the added head factor at each step, just like the friction factor, and we will overwrite the pipe that the pump replaces.[32]

Listing 35 implements the computation of the added head factor, and the pump selection factor.

Listing 35. R Code to include pumps in a pipeline network
.

```
.................
 # compute the current pump factor
 if(pumpCount > 0){
  for (i in 1:pumpCount)
    {
      addedhead[i] <- p_factor(pumps[i,2],pumps[i,3],pumps[i,4],flowguess[pumps[i,1]])
    }
  }
# build the function matrix
 # operate on the lower left partition of the matrix
 istart <- nodeCount+1
 iend <- nodeCount+pipeCount
 jstart <- 1
 jend <- flowCount
 for (i in istart:iend ){
   for(j in jstart:jend ){
     if ((i-istart+1) == j)  {augmentedMat[i,j] <- -1*lossfactor[j];
      if(pumpCount > 0){
        for(ipump in 1:pumpCount) {
          if(j == pumps[ipump,1]) augmentedMat[i,j] <- addedhead[ipump]
        }
      }
    }
   }
 }
# print(augmentedMat)
..................
```

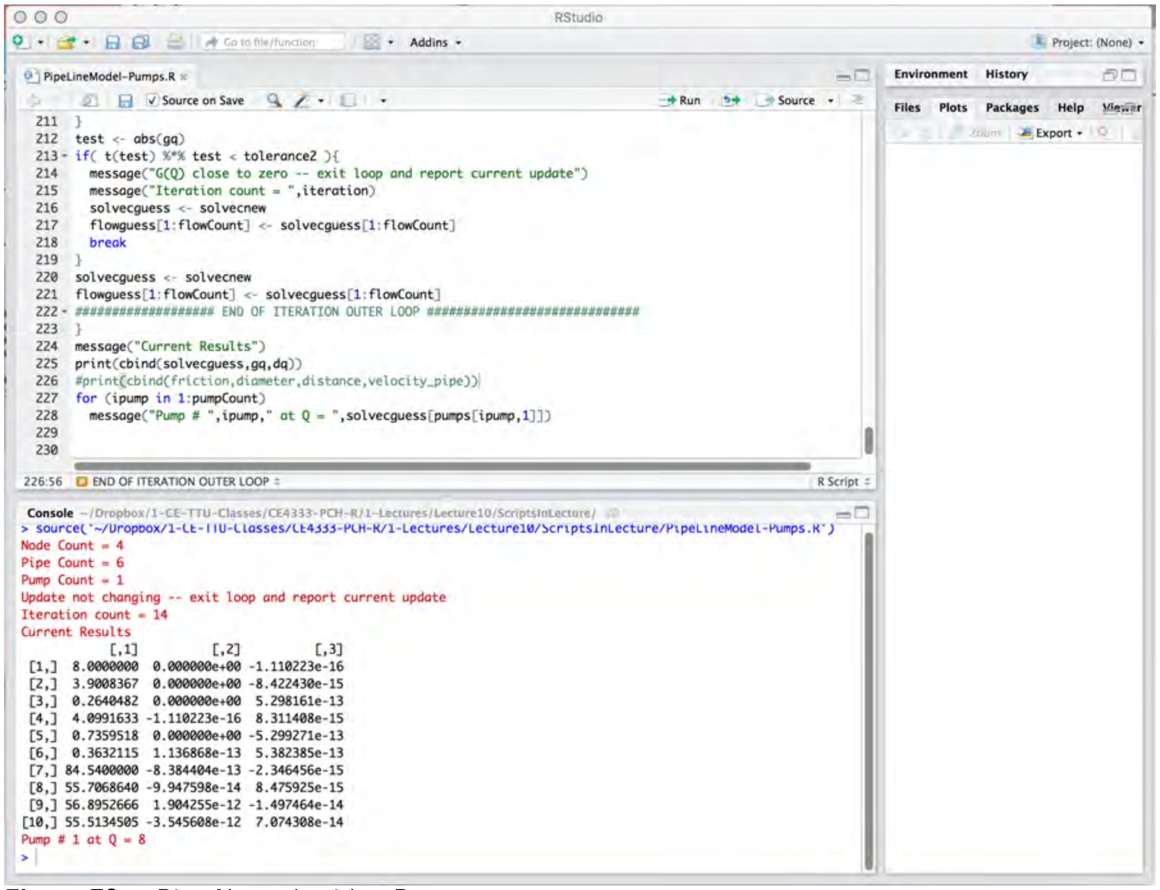
The remainder of the code is unchanged. Listing 36 illustrates the changes in the input file. We have added a row to indicate how many pumps will be used as the first record in the file. The last record after the right-hand side vector is the pump characteristics; one row for each pump. The scripts also test if there are zero pumps and skip code as needed. Observe we still preserve Link #1 data because its part of the node-arc matrix, but the length and diameter of the link is irrelevant (but need to be non-zero because we compute friction factors as if there were a pipe, but never use them.

Listing 36. Input file with pumps at link#1 in a pipeline network
.

```
1  <== how many pumps
4
6
1.00 0.67 0.67 0.67 0.67 0.5  <== link #1 needs values as placeholders, but are not used
800 800 700 700 800 600
0.00001 0.00001 0.00001 0.00001 0.00001 0.00001
0.000011
1 1 1 1 1 1
1 -1 0 -1 0 0
0 1 -1 0 0 1
0 0 0 1 -1 -1
0 0 1 0 1 0
0 4 3 1 0 0 0 0 0 0
1 100.54 0.25 2.0  <== Pump Link ID, H\_shutoff, K\_pump, Exponent
```

Figure 79 is a screen capture of the example problem run in **R Studio**. The script produces the correct flow values, and the pump specified was intended to match the

[32]This approach is decidedly a hack for illustration purposes. A more advanced program would probably just treat everything as a link and use a similar database build structure to determine if a link is a head loss or head add link. My reasoning is that there will be fewer pumps than pipes in any system, so overwriting a fictitious pipe is not too much trouble.



```
}
test <- abs(gq)
if( t(test) %*% test < tolerance2 ){
  message("G(Q) close to zero -- exit loop and report current update")
  message("Iteration count = ",iteration)
  solvecguess <- solvecnew
  flowguess[1:flowCount] <- solvecguess[1:flowCount]
  break
}
solvecguess <- solvecnew
flowguess[1:flowCount] <- solvecguess[1:flowCount]
################## END OF ITERATION OUTER LOOP ############################
}
message("Current Results")
print(cbind(solvecguess,gq,dq))
#print(cbind(friction,diameter,distance,velocity_pipe))
for (ipump in 1:pumpCount)
  message("Pump # ",ipump," at Q = ",solvecguess[pumps[ipump,1]])
```

```
> source('~/Dropbox/1-CE-TTU-Classes/CE4333-PCH-R/1-Lectures/Lecture10/ScriptsInLecture/PipeLineModel-Pumps.R')
Node Count = 4
Pipe Count = 6
Pump Count = 1
Update not changing -- exit loop and report current update
Iteration count = 14
Current Results
            [,1]          [,2]          [,3]
 [1,]  8.0000000  0.000000e+00 -1.110223e-16
 [2,]  3.9008367  0.000000e+00 -8.422430e-15
 [3,]  0.2640482  0.000000e+00  5.298161e-13
 [4,]  4.0991633 -1.110223e-16  8.311408e-15
 [5,]  0.7359518  0.000000e+00 -5.299271e-13
 [6,]  0.3632115  1.136868e-13  5.382385e-13
 [7,] 84.5400000 -8.384404e-13 -2.346456e-15
 [8,] 55.7068640 -9.947598e-14  8.475925e-15
 [9,] 56.8952666  1.904255e-12 -1.497464e-14
[10,] 55.5134505 -3.545608e-12  7.074308e-14
Pump # 1 at Q = 8
>
```

Figure 79. Pipe Network with a Pump.

previous problem closely in that it produces enough head so that node N1 has nearly the same head value as the problem without a pump.

8.1.2 Fitting (Minor) Losses

In addition to head loss in the conduit, other losses are created by inlets, outlets, transitions, and other connections in the system. In fact such losses can be used to measure discharge (think of the orifice plate in the fluids laboratory). The fittings create additional turbulence that generates heat and produces the head loss.

Equation 94 is the typical loss model

$$h_{minor} = K\frac{V^2}{2g} \tag{94}$$

where K is called a minor loss coefficient, and is tabulated (e.g. Table 7) for various kinds of fittings.

The use is straightforward, and multiple fittings are summed in the loss term in the energy equation. In practical computation, these losses make the most sense when

Table 7. Minor Loss Coefficients for Different Fittings.

Fitting Type	K
Tee, Flanged, Line Flow	0.2
Tee, Threaded, Line Flow	0.9
Tee, Flanged, Branched Flow	1.0
Tee, Threaded , Branch Flow	2.0
Union, Threaded	0.08
Elbow, Flanged Regular 90^o	0.3
Elbow, Threaded Regular 90^o	1.5
Elbow, Threaded Regular 45^o	0.4
Elbow, Flanged Long Radius 90^o	0.2
Elbow, Threaded Long Radius 90^o	0.7
Elbow, Flanged Long Radius 45^o	0.2
Return Bend, Flanged 180^o	0.2
Return Bend, Threaded 180^o	1.5
Globe Valve, Fully Open	10
Angle Valve, Fully Open	2
Gate Valve, Fully Open	0.15
Gate Valve, 1/4 Closed	0.26
Gate Valve, 1/2 Closed	2.1
Gate Valve, 3/4 Closed	17
Swing Check Valve, Forward Flow	2
Ball Valve, Fully Open	0.05
Ball Valve, 1/3 Closed	5.5
Ball Valve, 2/3 Closed	200
Diaphragm Valve, Open	2.3
Diaphragm Valve, Half Open	4.3
Diaphragm Valve, 1/4 Open	21
Water meter	7

associated with a particular pipe. If we rewrite the loss equation

$$h_{minor} = \frac{K}{2g}\frac{16Q^2}{\pi^2 D^4} \tag{95}$$

we see that these terms can be added to a pipe either as an additional loss term and placed in the augmented matrix in the same way as the other loss term.

8.1.3 Extracted Head — Turbines

The head recovered by a turbine is also an "added head" but appears on the loss side of the equation. Equation 96 is the power that can be recovered by a turbine (again using the concept of "water-to-wire" efficiency is

$$P = \eta Q \rho g h_t \tag{96}$$

An approach similar to pumps would be employed — the effort in all these cases is to represent the hydraulic components as a loss factor so the non-linear solver we have already built can be used.

9 EPANET by Example

9.1 About

EPANET is a computer program that performs hydraulics computations in pressure-pipe systems. The internal computational engine is similar to that explored in the previous chapter(s), but is far more efficient and well tested. The current version of EPANET (Official EPA Release) is 2.00.12. There is an ongoing Open Source Project that has released a version 2.1 (to follow the release numbering scheme). There is likely to be a version 2.3 within a few years. All the versions so far include a capacity to read an ASCII input file to direct the computations.

A GUI Interface is available that runs in Windows that is quite popular and useful, but it is elderly and newer interfaces are being explored. Many users dispense with the interface entirely and operate the model using custom-built (or general) wrapper programs to call various DLLs (or Shared Objects). Wrappers exist in **R**, Python, Delphi (the Legacy GUI), and probably PERL and Ruby.

The remainder of this chapter shows how to use EPANET by a series of representative examples. These examples are at best a subset of the capabilities of the program, but should be enough to get one started. The program requires some hydraulic insight to interpret the results as well as detect data entry or conceptualization errors. [33]

9.2 Using the Legacy Interface

9.2.1 Installing the Program

The `urltoinstallvideo` shows how to download and install EPANET onto your computer. EPANET will run fine on a laptop computer even a Macintosh that has a guest Windows OS (WM-Ware, Parallels, or BootCamp).

The `urltoinstallvideo` shows how to install an implementation that runs (so-so) on a Macintosh (using a container built using Winebottler and WINE).

The `urltoinstallvideo` shows how to install and run an implementation on a Linux computer (using WINE).

EPANET can also be installed onto a flash-drive and run directly from the drive [34].

[33]This chapter is from "Cleveland, T.G., Tay, C.C., and Neale, C.N. 2015. EPANET by Example. Department of Civil and Environmental Engineering, Texas Tech University." available at `http://cleveland3.ddns.net/university-courses/ce-3372/3-Readings/EPANETbyExample/`

[34]A useful trick on a networked system — be sure you set up the flash drive to be writeable! I have never tried this on a non-Windows implementation so cannot comment much on that.

9.3 EPANET Modeling by Example

EPANET models are comprised of nodes, links,and reservoirs. Pumps are treated as special links (that add head). Valves are also treated as special links depending on the valve types. All models must have a reservoir (or storage tank).

9.3.1 Defaults

The program has certain defaults that should be set at the beginning of a simulation. The main defaults of importance are the head loss equations (Darcy-Weisbach, Hazen-Williams, or Chezy-Manning) and the units (CFS, LPS, etc.)

9.3.2 Example 1: Flow in a Single Pipe

A simple model to consider is a single pipe, a classic problem statement might be something like

> A 5-foot diameter, enamel coated, steel pipe carries 60^{o}F water at a discharge of 295 cubic-feet per second (cfs). Using the Moody chart, estimate the head loss in a 10,000 foot length of this pipe.

In EPANET we will start the program, build a tank-pipe system and find the head loss in a 10,000 foot length of the pipe. The program will compute the friction factor for us (and we can check on the Moody chart if we wish).

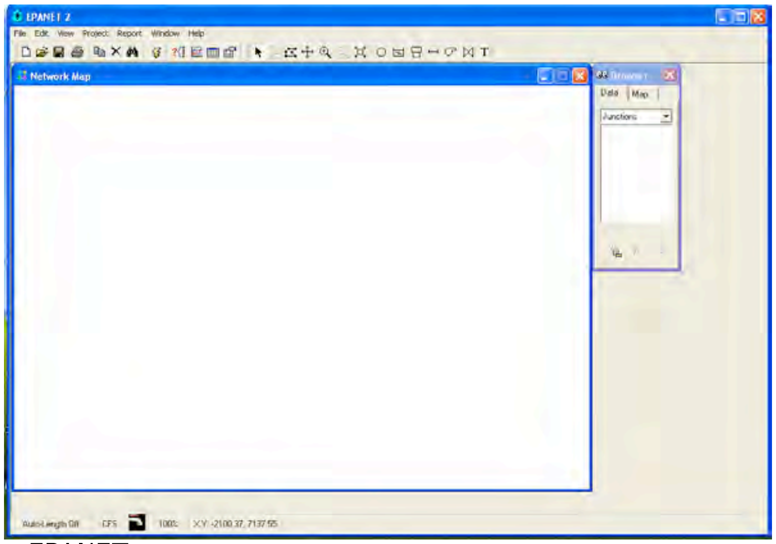
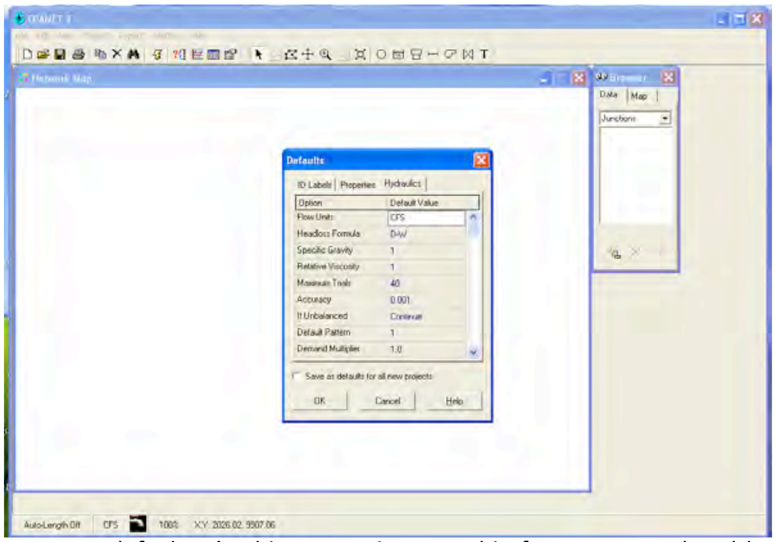

EPANET 2
File Edit View Project Report Window Help
Network Map
Browser
Data Map
Junctions
Auto-Length Off CFS 100% X,Y: -2100.37, 7137.55

Figure 80. Start EPANET program.

The main trick in EPANET is going to be the friction coefficient, in the EPANET manual on page 30 and 31, the author indicates that the program expects a roughness coefficient based on the head loss equation. The units of the roughness coefficient for a steel pipe are 0.15×10^{-3} feet. On page 71 of the user manual the author states that roughness coefficients are in millifeet (millimeters) when the Darcy-Weisbach head loss model is used. So keeping that in mind we proceed with the example.

Figure 116 is a screen capture of the EPANET program after installing the program. The program starts as a blank slate and we will select a reservoir and a node from the tool bar at the top and place these onto the design canvas.

Figure 116 is a screen capture of the EPANET program after setting defaults for the simulation. Failure to set correct units for your problem are sometimes hard to detect (if the model runs), so best to make it a habit to set defaults for all new projects. Next we add the reservoir and the node. Figure 120 is a screen capture after the



Option	Default Value
Flow Units	CFS
Headloss Formula	D-W
Specific Gravity	1
Relative Viscosity	1
Maximum Trials	40
Accuracy	0.001
If Unbalanced	Continue
Default Pattern	1
Demand Multiplier	1.0

Figure 81. Set program defaults. In this case units are cubic-feet-per-second and loss model is Darcy-Weisbach..

reservoir and node is placed. We will specify a total head at the reservoir (value is unimportant as long as it is big enough to overcome the head loss and not result in a negative pressure at the node. We will specify the demand at the node equal to the desired flow in the pipe. Next we will add the pipe.

Figure 122 is a screen capture after the pipe is placed. The sense of flow in this example is from reservoir to node, but if we had it backwards we could either accept a negative flow in the pipe, or right-click the pipe and reverse the start and end node connections.

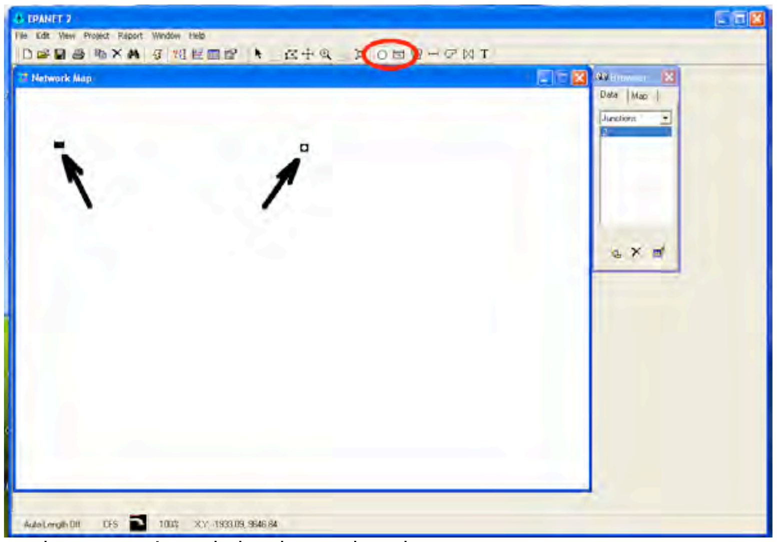


Figure 82. Place the reservoir and the demand node..

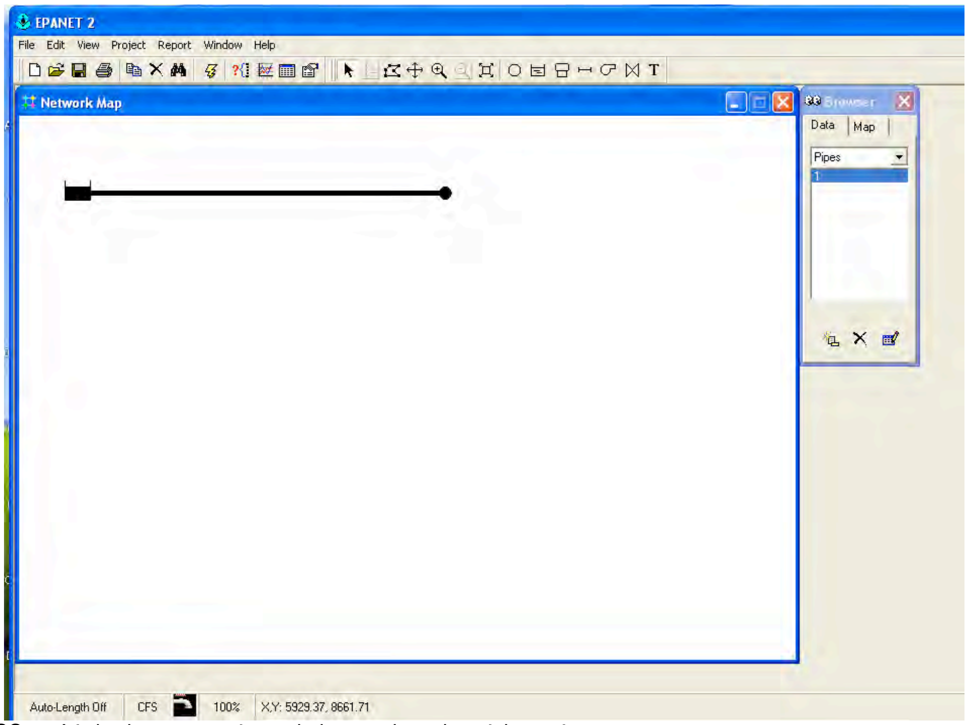
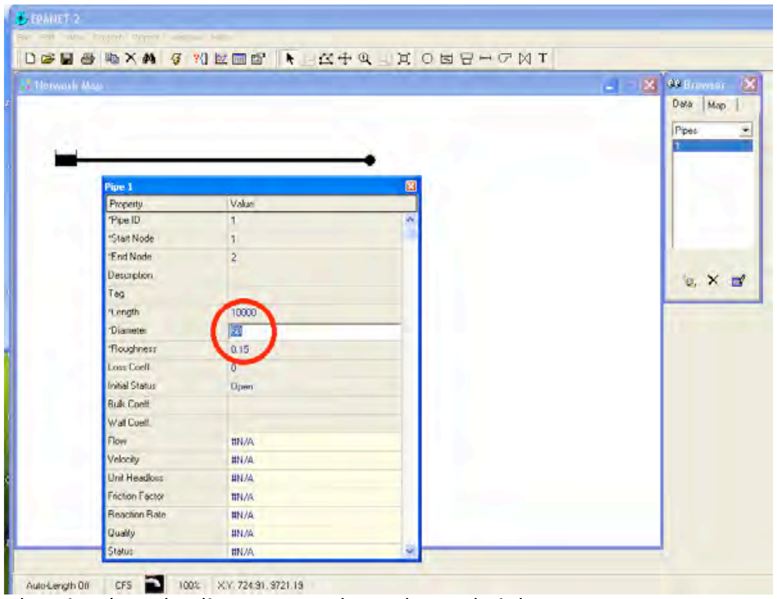

EPANET 2
File Edit View Project Report Window Help
Network Map
Browser
Data Map
Pipes
1
Auto-Length Off CFS 100% X,Y: 5929.37, 8661.71

Figure 83. Link the reservoir and demand node with a pipe..

Now we can go back to each hydraulic element in the model and edit the properties. We supply pipe properties (diameter, length, roughness height) as in Figure 124. We

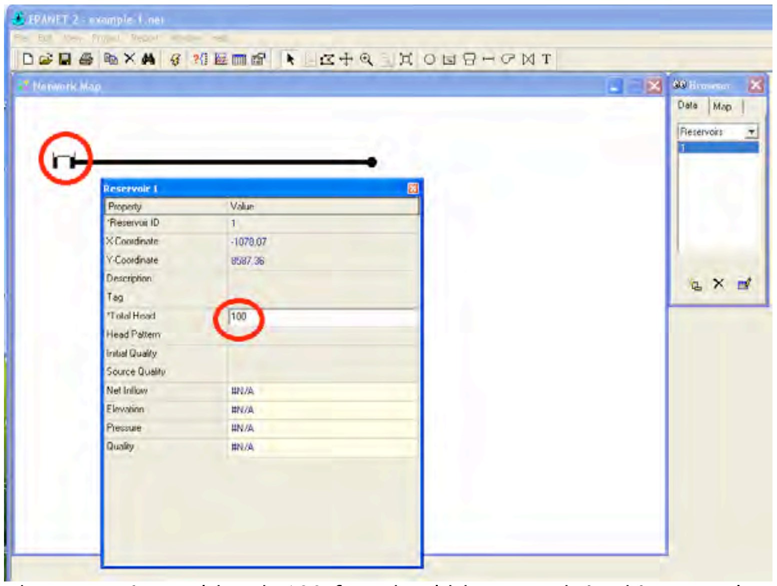


Pipe 1

Property	Value
*Pipe ID	1
*Start Node	1
*End Node	2
Description	
Tag	
*Length	10000
*Diameter	20
*Roughness	0.15
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	#N/A
Velocity	#N/A
Unit Headloss	#N/A
Friction Factor	#N/A
Reaction Rate	#N/A
Quality	#N/A
Status	#N/A

Figure 84. Set the pipe length, diameter, and roughness height..

supply the reservoir total head as in Figure 126.

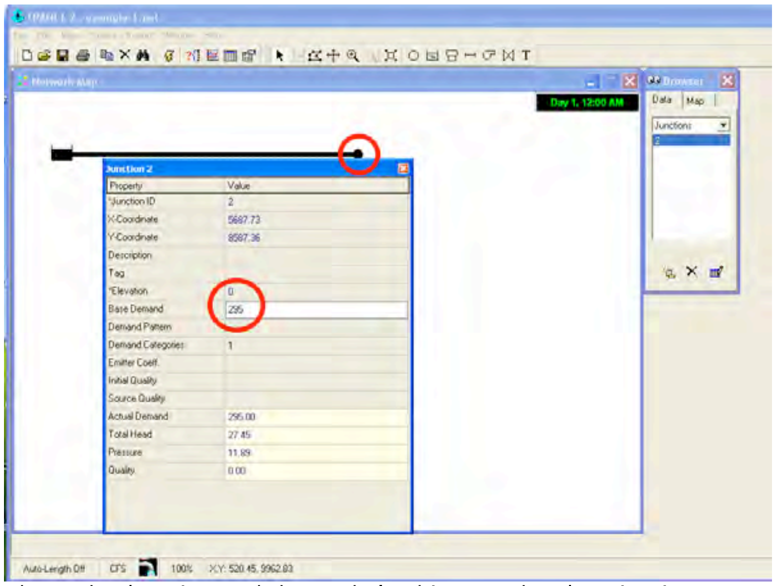


Reservoir 1

Property	Value
*Reservoir ID	1
X-Coordinate	-1078.07
Y-Coordinate	8587.36
Description	
Tag	
*Total Head	100
Head Pattern	
Initial Quality	
Source Quality	
Net Inflow	#N/A
Elevation	#N/A
Pressure	#N/A
Quality	#N/A

Figure 85. Set the reservoir total head, 100 feet should be enough in this example..

We set the demand node elevation and the actual desired flow rate as in Figure 128. The program is now ready to run, next step would be to save the input file

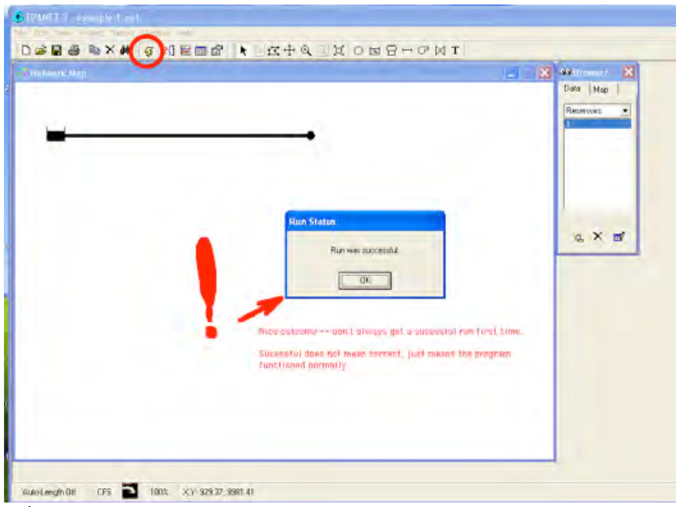


Junction 2

Property	Value
*Junction ID	2
X-Coordinate	5687.73
Y-Coordinate	8587.36
Description	
Tag	
*Elevation	0
Base Demand	295
Demand Pattern	
Demand Categories	1
Emitter Coeff.	
Initial Quality	
Source Quality	
Actual Demand	295.00
Total Head	27.45
Pressure	11.89
Quality	0.00

Figure 86. Set the node elevation and demand. In this case the elevation is set to zero (the datum) and the demand is set to 295 cfs as per the problem statement..

(File/Save/Name), then run the program by selecting the lighting bolt looking thing and the computation engine will start.



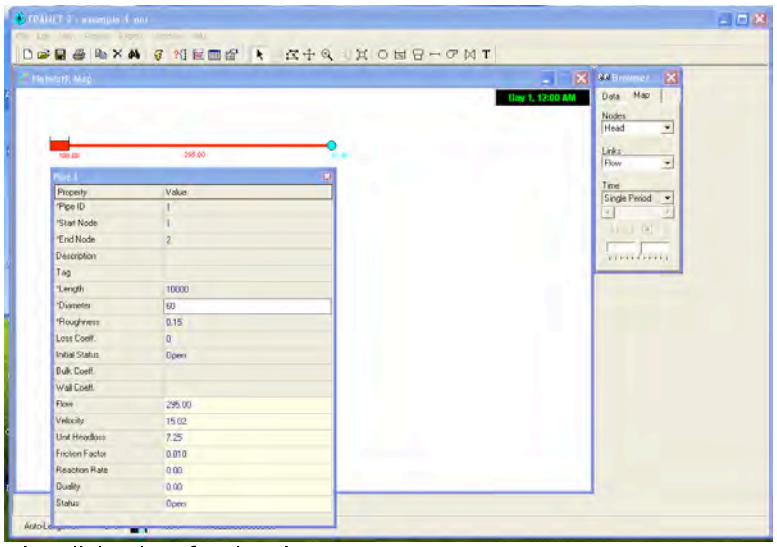
Run was successful

Nice outcome -- don't always get a successful run first time.

Successful does not mean correct, just means the program functioned normally

Figure 87. Running the program.

If the nodal connectivity is OK and there are no computed negative pressures the program will run. Figure 130 is the appearance of the program after the run is complete (the annotations are mine!). A successful run means the program found an answer to the problem you provided – whether it is the correct answer to your problem requires the engineer to interpret results and decide if they make sense. The more common conceptualization errors are incorrect units and head loss equation for the supplied roughness values, missed connections, and forgetting demand somewhere. With practice these kind of errors are straightforward to detect. In the present example we select the pipe and the solution values are reported at the bottom of a dialog box.



Property	Value
*Pipe ID	1
*Start Node	1
*End Node	2
Description	
Tag	
*Length	10000
*Diameter	60
*Roughness	0.15
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	295.00
Velocity	15.02
Unit Headloss	7.25
Friction Factor	0.010
Reaction Rate	0.00
Quality	0.00
Status	Open

Figure 88. Solution dialog box for the pipe..

Figure 133 is the result of turning on the computed head values at the node (and reservoir) and the flow value for the pipe. The dialog box reports about 7.2 feet of head loss per 1000 feet of pipe for a total of 72 feet of head loss in the system. The total head at the demnad node is about 28 feet, so the head loss plus remaining head at the node is equal to the 100 feet of head at the reservoir, the anticipated result.

The computed friction factor is 0.010, which we could check against the Moody chart if we wished to adjust the model to agree with some other known friction factor.

9.3.3 Example 2: Flow Between Two Reservoirs

This example represents the situation where the total head is known at two points on a pipeline, and one wishes to determine the flow rate (or specify a flow rate and solve for a pipe diameter). Like the prior example it is contrived, but follows the same general modeling process.

As in the prior example, we will use EPANET to solve a problem we have already solved by hand.

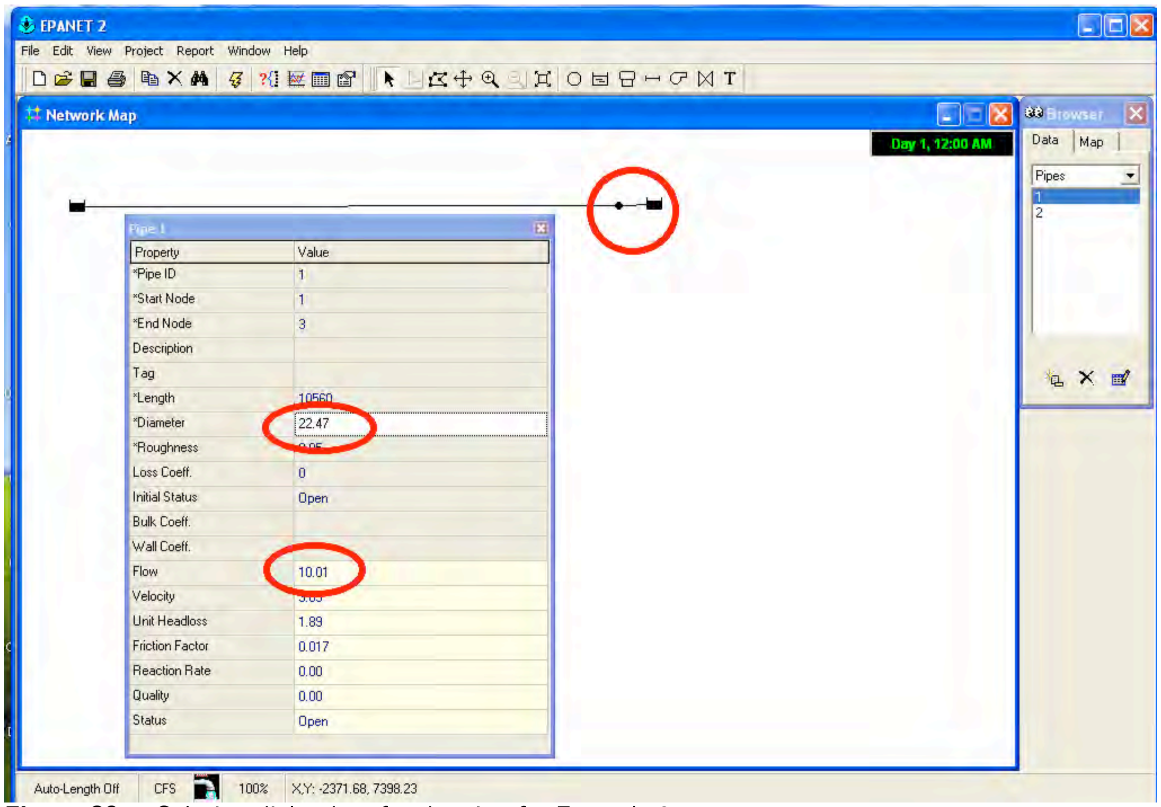
> Using the Moody chart, and the energy equation, estimate the diameter of a cast-iron pipe needed to carry 60^oF water at a discharge of 10 cubic-feet per second (cfs) between two reservoirs 2 miles apart. The elevation difference between the water surfaces in the two reservoirs is 20 feet.

As in the prior example, we will need to specify the pipe roughness terms, then solve by trial-and-error for the diameter required to carry the water at the desired flowrate. Page 31 of the EPANET manual suggests that the roughness height for cast iron is 0.85 millifeet.

As before the steps to model the situation are:

1. Start EPANET
2. Set defaults
3. Select the reservoir tool. Put two reservoirs on the map.
4. Select the node tool, put a node on the map. **EPA NET needs one node!**
5. Select the link (pipe) tool, connect the two reservoirs to the node. One link is the 2 mile pipe, the other is a short large diameter pipe (negligible head loss).
6. Set the total head each reservoir.
7. Set the pipe length and roughness height in the 2 mile pipe.
8. Guess a diameter.
9. Save the input file.
10. Run the program. Query the pipe and find the computed flow. If the flow is too large reduce the pipe diameter, if too small increase the pipe diameter. Stop when within a few percent of the desired flow rate. Use commercially available diameters in the trial-and-error process, so exact match is not anticipated.

Figure 135 is a screen capture after the model is built and some trial-and-error diameter selection. Of importance is the node and the "short pipe" that connects the second reservoir. By changing the diameter (inches) in the dialog box and re-running the program we can find a solution (diameter) that produces 10 cfs in the system for the given elevation differences.


EPANET 2
File Edit View Project Report Window Help
Network Map
Day 1, 12:00 AM
Browser
Data Map
Pipes
1
2
Pipe 1
Auto-Length Off CFS 100% X,Y: -2371.68, 7398.23

Property	Value
*Pipe ID	1
*Start Node	1
*End Node	3
Description	
Tag	
*Length	10560
*Diameter	22.47
*Roughness	0.05
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	10.01
Velocity	3.63
Unit Headloss	1.89
Friction Factor	0.017
Reaction Rate	0.00
Quality	0.00
Status	Open

Figure 89. Solution dialog box for the pipe for Example 2.

We would conclude from this use of EPANET that a 22.75 inch ID cast iron pipe would convey 10 cfs between the two reservoirs. Compare this solution to the "by-hand" soluton to see if they are close.

9.3.4 Example 3: Three-Reservoir-Problem

This example repeats another prior problem, but introduces the concept of a basemap (image) to help draw the network. First the problem statement

> Reservoirs A, B, and C are connected as shown[35] in Figure 137. The water elevations in reservoirs A, B, and C are 100 m, 80 m, and 60 m. The three pipes connecting the reservoirs meet at junction J, with pipe AJ being 900 m long, BJ being 800 m long, and CJ being 700 m long. The diameters of all the pipes are 850 mm. If all the pipes are ductile iron, and the water temperature is 293^{o}K, find the direction and magnitude of flow in each pipe.

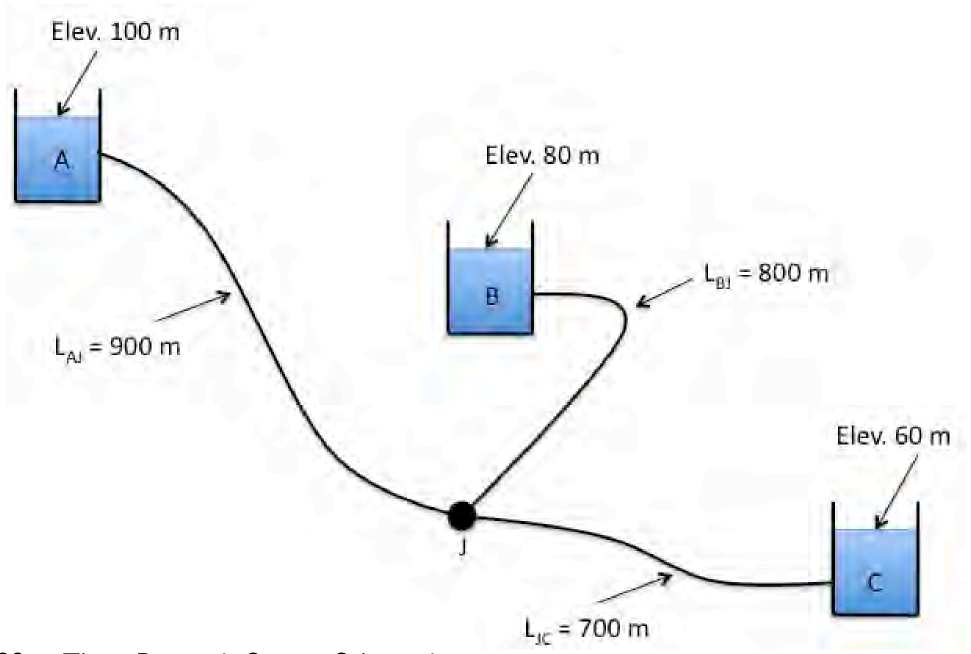


Figure 90. Three-Reservoir System Schematic.

Here we will first convert the image into a bitmap (.bmp) file so EPANET can import the background image and we can use it to help draw the network. The remainder of the problem is reasonably simple and is an extension of the previous problem.

The steps to model the situation are:

1. Convert the image into a bitmap, place the bitmap into a directory where the model input file will be stored.
2. Start EPANET
3. Set defaults
4. Import the background.

[35]This problem is identical to Chin Problem 2.30, Pg. 92

5. Select the reservoir tool. Put three reservoirs on the map.
6. Select the node tool, put the node on the map.
7. Select the link (pipe) tool, connect the three reservoirs to the node.
8. Set the total head each reservoir.
9. Set the pipe length, roughness height, and diameter in each pipe.
10. Save the input file.
11. Run the program.

Figure 138 is the result of the above steps. In this case the default units were changed to LPS (liters per second). The roughness height is about 0.26 millimeters (if converted from the 0.85 millifeet unit).

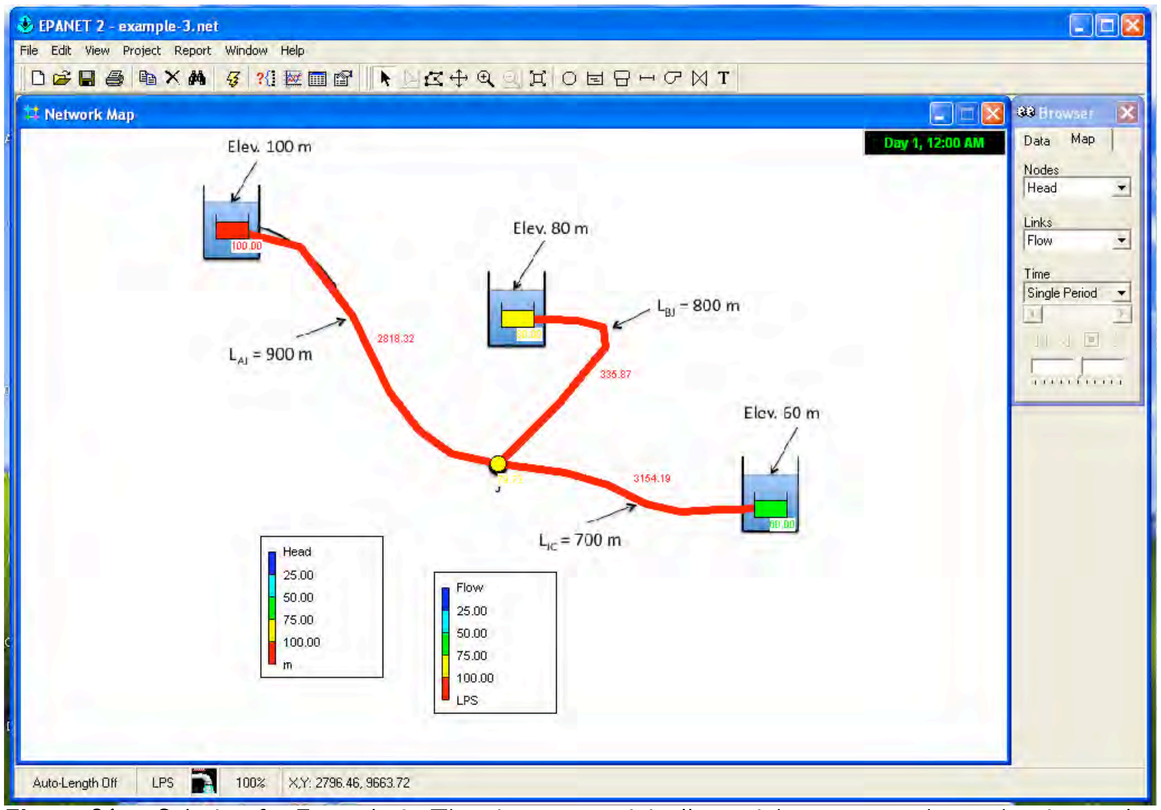


Figure 91. Solution for Example 3. The pipes were originally straight segments, but a drawing tool in EPANET is used to follow the shape of the underlying basemap. The training video shows the pipes as the straight lines. The flowrates are in liters-per-second, divide by 1000 to obtain cubic-meters-per-second..

9.3.5 Example 4: A Simple Network

Expanding the examples, we will next consider a looped network. As before we will use a prior exercise as the motivating example.

> The water-supply network shown in Figure 140 has constant-head elevated storage tanks at A and C, with inflow and outflow at B and D. The network is on flat terrain with node elevations all equal to 50 meters[36]. If all pipes are ductile iron, compute the inflows/outflows to the storage tanks. Assume that flow in all pipes are fully turbulent.

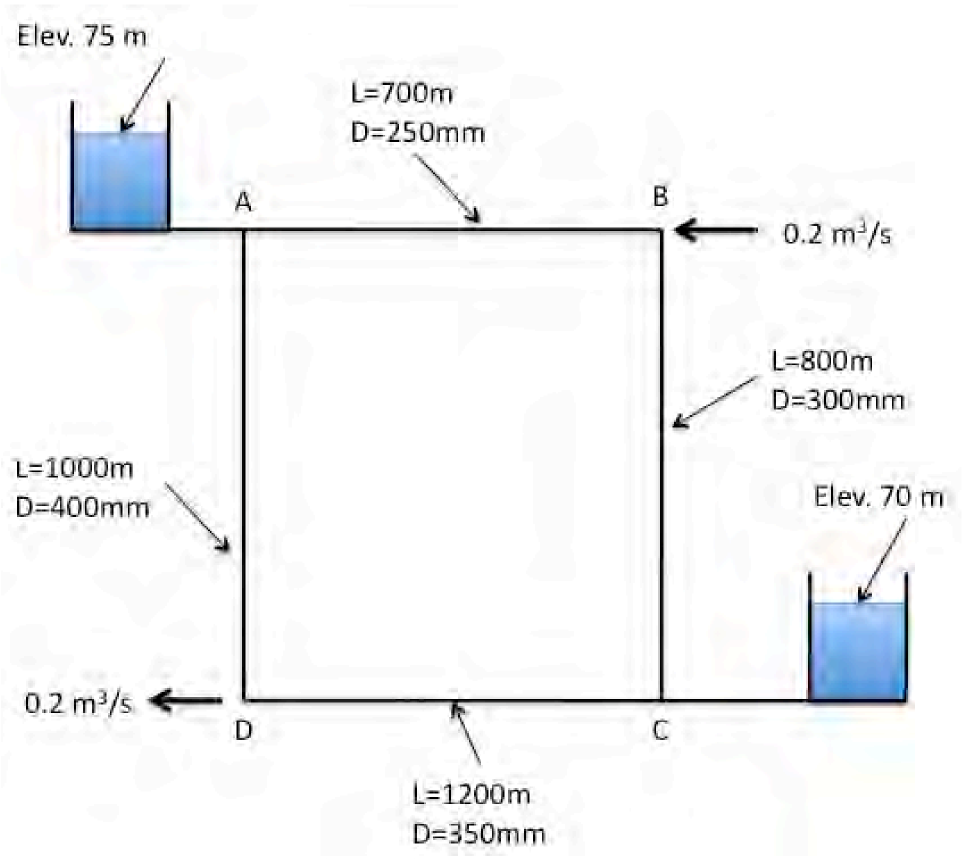


Figure 92. Two-Tank Distribution System Schematic.

As before we will follow the modeling protocol but add demand at the nodes.

The steps to model the situation are:

1. Convert the image into a bitmap, place the bitmap into a directory where the model input file will be stored.
2. Start EPANET
3. Set defaults

[36]This problem is similar to Chin Problem 2.31, Pg. 92

4. Import the background.
5. Select the reservoir tool. Put two reservoirs on the map.
6. Select the node tool, put 4 nodes on the map.
7. Select the link (pipe) tool, connect the reservoirs to their nearest nodes. Connect the nodes to each other.
8. Set the total head each reservoir.
9. Set the pipe length, roughness height, and diameter in each pipe. The pipes that connect to the reservoirs should be set as short and large diameter, we want negligible head loss in these pipes so that the reservoir head represents the node heads at these locations.
10. Save the input file.
11. Run the program.

In this case the key issues are the units (liters per second) and roughness height (0.26 millimeters). Figure 141 is a screen capture of a completed model.

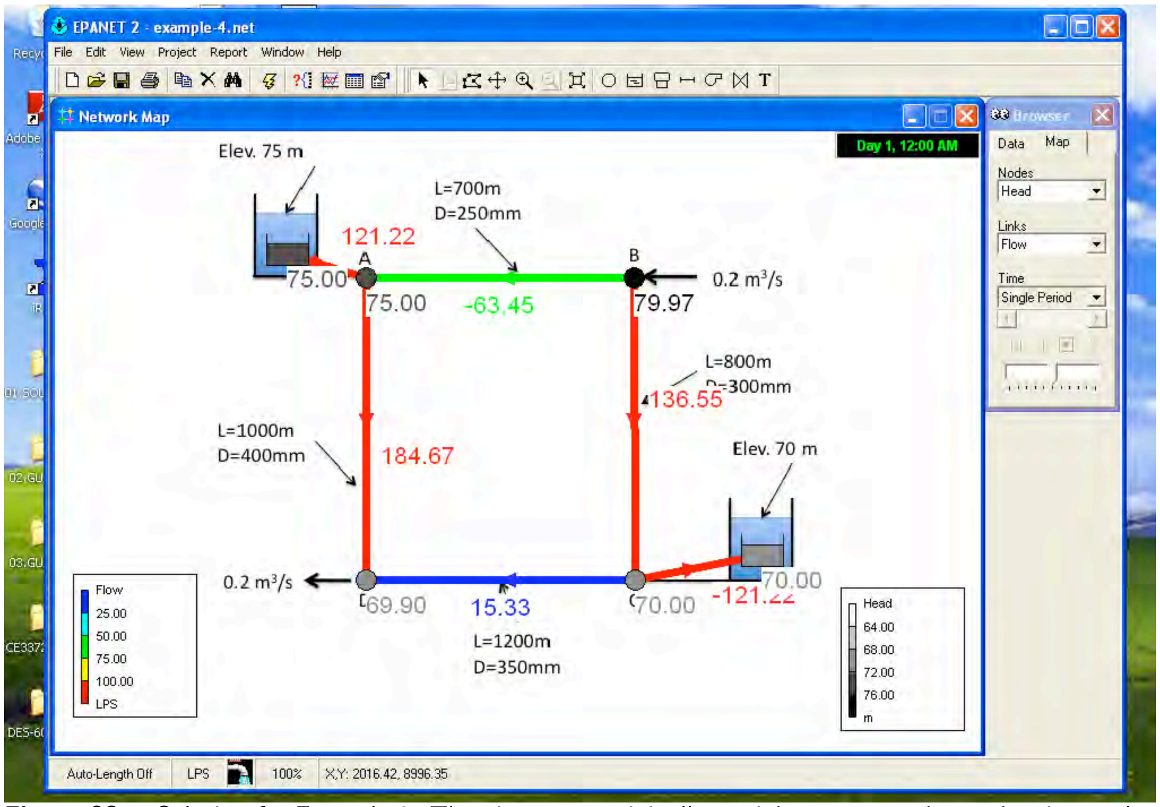


Figure 93. Solution for Example 3. The pipes were originally straight segments, but a drawing tool in EPANET is used to follow the shape of the underlying basemap. The training video shows the pipes as the straight lines. The flowrates are in liters-per-second, divide by 1000 to obtain cubic-meters-per-second..

9.4 EPANET Modeling by Example: Example 5

The next example illustrates how to model a pump in EPANET. A pump is a special "link" in EPANET. This link causes a negative head loss (adds head) according to a pump curve. In additon to a pump curve there are three other ways to model added head — these are discussed in th eunser manual and are left for the reader to explore on their own.

9.4.1 Example 5: Pumping Water Uphill

Figure 143 is a conceptual model of a pump lifting water through a 100 mm diameter, 100 meter long, ductile iron pipe from a lower to an upper reservoir. The suction side of the pump is a 100 mm diameter, 4-meter long ductile iron pipe. The difference in reservoir free-surface elevations is 10 meters. The pump performance curve is given as

$$h_p = 15 - 0.1Q^2 \tag{97}$$

where the added head is in meters and the flow rate is in liters per second (Lps). The analysis goal is to estimate the flow rate in the system.

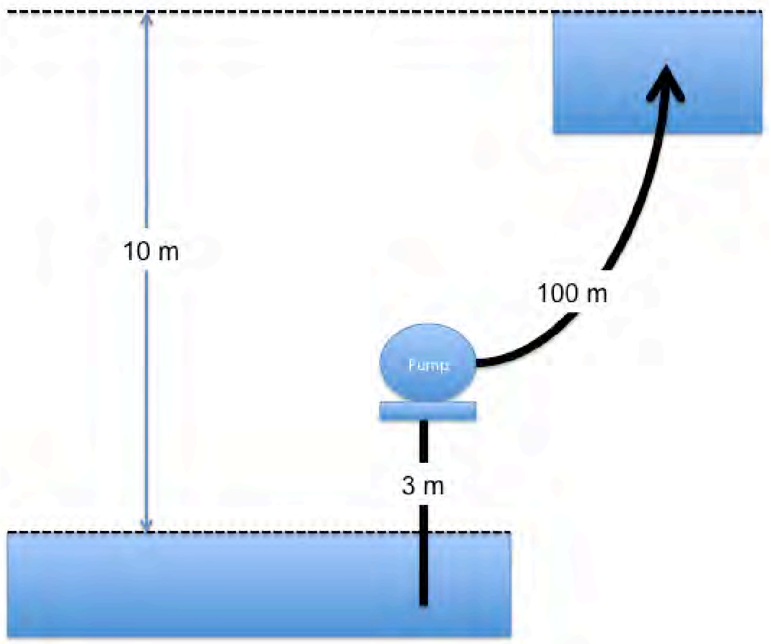
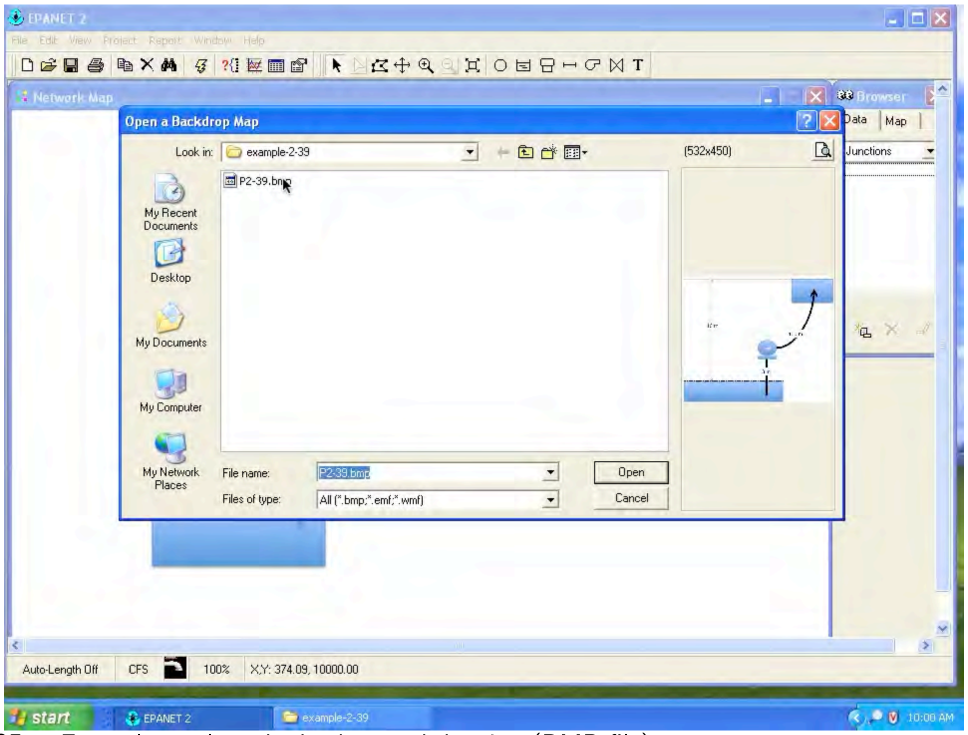


Figure 94. Example 5 conceptual model. The pipes are 100 mm ductile iron..

To model this situation, the engineer follows the modeling protocol already outlined, only adding the special link.

1. Convert the image into a bitmap, place the bitmap into a directory where the model input file will be stored.

2. Start EPANET
3. Set defaults (hydraulics = D-W, units = LPS)
4. Import the background.
5. Select the reservoir tool. Put two reservoirs on the map.
6. Select the node tool, put 2 nodes on the map, these represent the suction and discharge side of the pump.
7. Select the link (pipe) tool, connect the reservoirs to their nearest nodes.
8. Select the pump tool.
9. Connect the nodes to each other using the pump link.
10. Set the total head each reservoir.
11. Set the pipe length, roughness height, and diameter in each pipe.
12. On the Data menu, select Curves. Here is where we create the pump curve. This problem gives the curve as an equation, we will need three points to define the curve. Shutoff ($Q = 0$), and simple to compute points make the most sense.
13. Save the input file.
14. Run the program.



Look in: example-2-39

P2-39.bmp

(532x450)

File name: P2-39.bmp

Files of type: All (*.bmp;*.emf;*.wmf)

Auto-Length Off CFS 100% X,Y: 374.09, 10000.00

Figure 95. Example 5 select the background drawing (BMP file).

Figure 144 is a screen capture of loading the background image. After the image is loaded, we can then build the hydraulic model. The next step is to place the reservoirs.

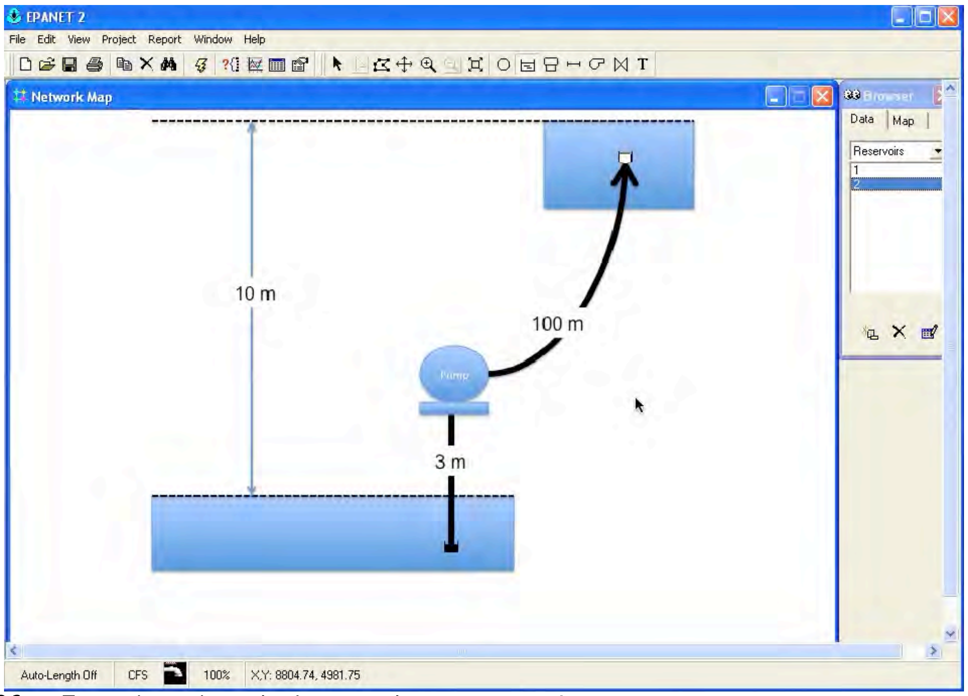

EPANET 2
File Edit View Project Report Window Help
Network Map
10 m
100 m
Pump
3 m
Browser
Data Map
Reservoirs
1
2
Auto-Length Off CFS 100% X,Y: 8804.74, 4981.75

Figure 96. Example 5 place the lower and upper reservoir.

Figure 145 is a screen capture of the reservoirs after they have been placed. The upper reservoir will be assigned a total head 10 meters larger than the lower reservoir — a reasonable conceptual model is to use the lower reservoir as the datum.

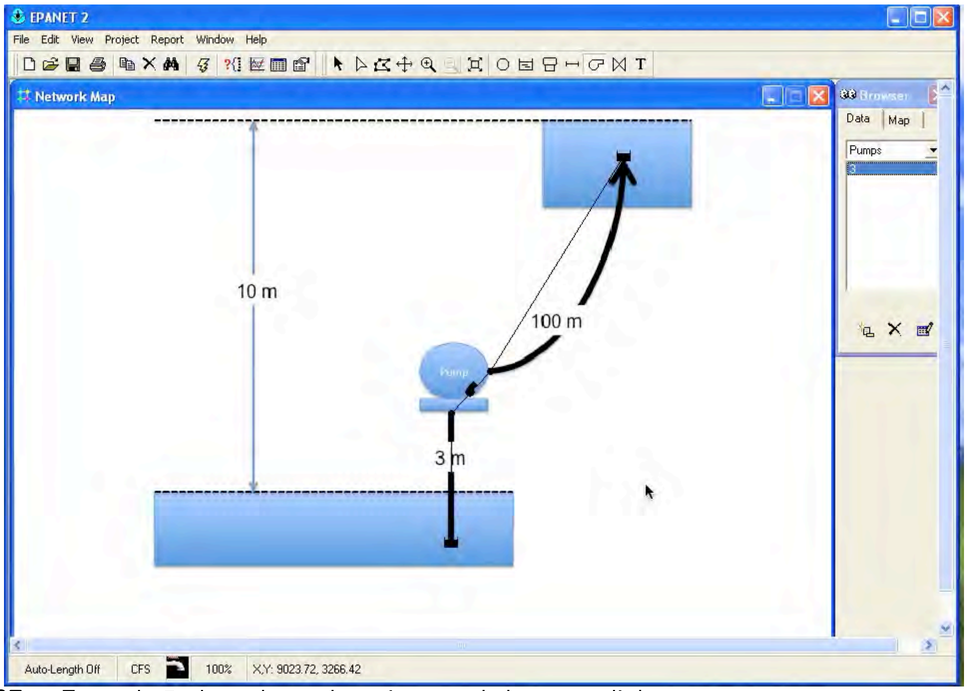
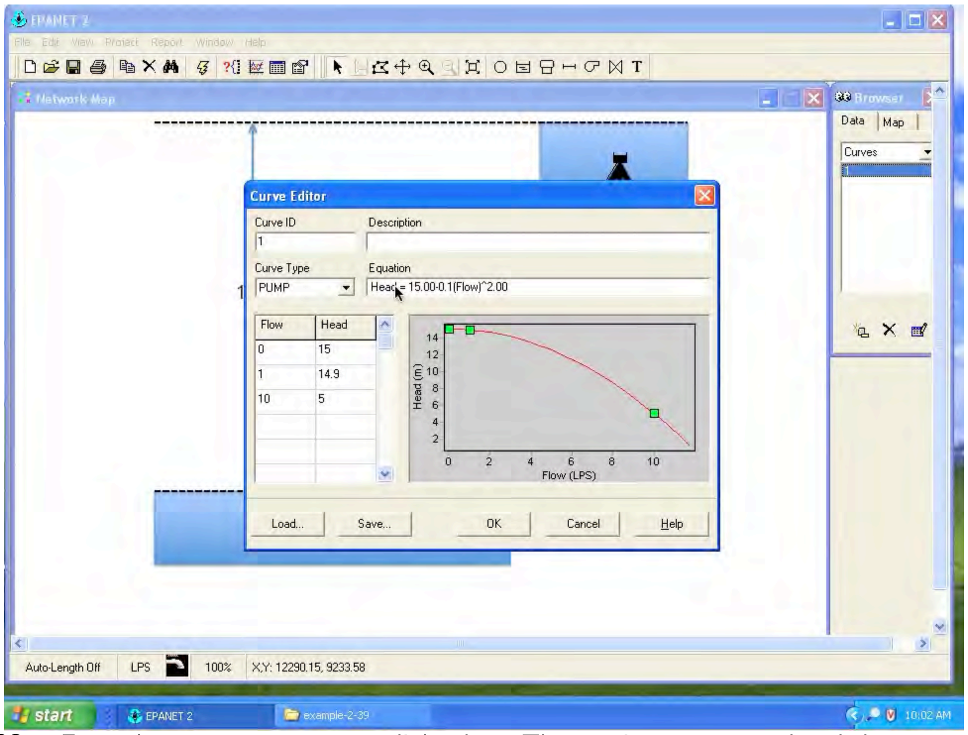

EPANET 2
File Edit View Project Report Window Help
Network Map
10 m
100 m
Pump
3 m
Browser
Data Map
Pumps
Auto-Length Off CFS 100% X,Y: 9023.72, 3266.42

Figure 97. Example 5 place the nodes, pipes, and the pump link..

Figure 146 is a screen capture of model just after the pump is added. The next steps are to set the pipe lengths (not shown) and the reservoir elevations (not shown). Finally, the engineer must specify the pump curve.



Curve ID: 1

Description:

Curve Type: PUMP

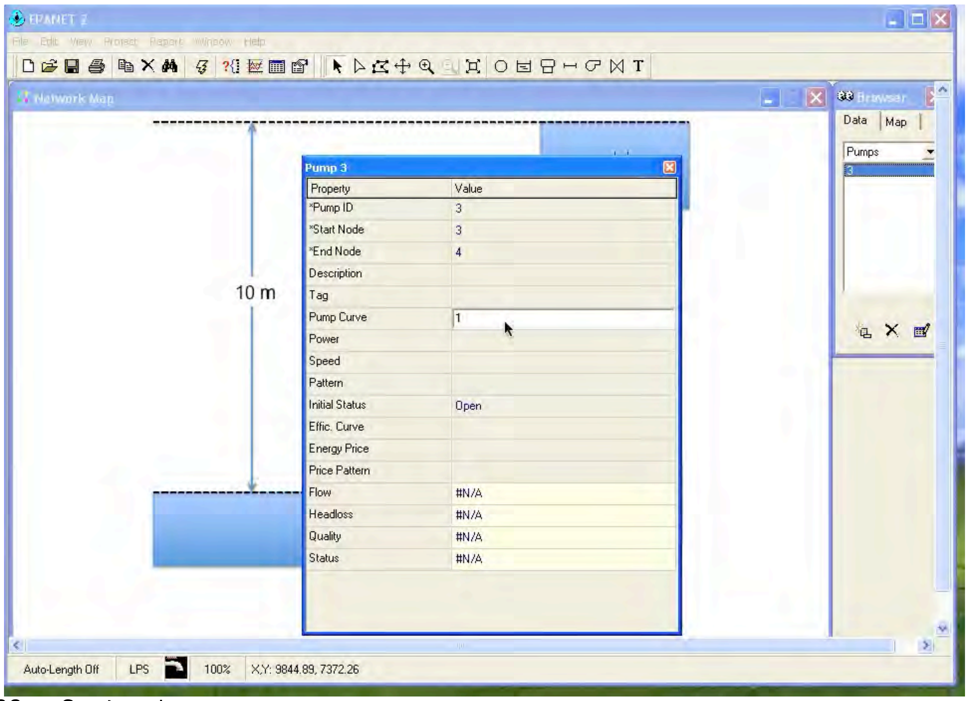
Equation: Head = 15.00-0.1(Flow)^2.00

Flow	Head
0	15
1	14.9
10	5

Figure 98. Example 5 pump curve entry dialog box. Three points are entered and the curve equation is created by the program..

Figure 147 is a screen capture of the pump curve data entry dialog box. Three points on the curve were selected and entered into the tabular entry area on the left of the dialog box, then the curve is created by the program. The equation created by the program is the same as that of the problem – hence we have the anticipated pump curve.

Next the engineer associates the pump curve with the pump as shown in Figure 148.



Pump 3

Property	Value
*Pump ID	3
*Start Node	3
*End Node	4
Description	
Tag	
Pump Curve	1
Power	
Speed	
Pattern	
Initial Status	Open
Effic. Curve	
Energy Price	
Price Pattern	
Flow	#N/A
Headloss	#N/A
Quality	#N/A
Status	#N/A

Figure 99. Setting the pump curve..

Upon completion of this step, the program is run to estimate the flow rate in the system.

9.5 EPANET Modeling by Example: Example 6

Example 6 is an extended period simulation – the files will simply be provided; the concept is well described in most hydraulics textbooks, as well as in the EPANET Documentation, however in the interest of time, it is left to the reader to download and run an extended period simulation model. The goal (for the seminar) is to have a working model of some complexity to explore the other tools discussed later.

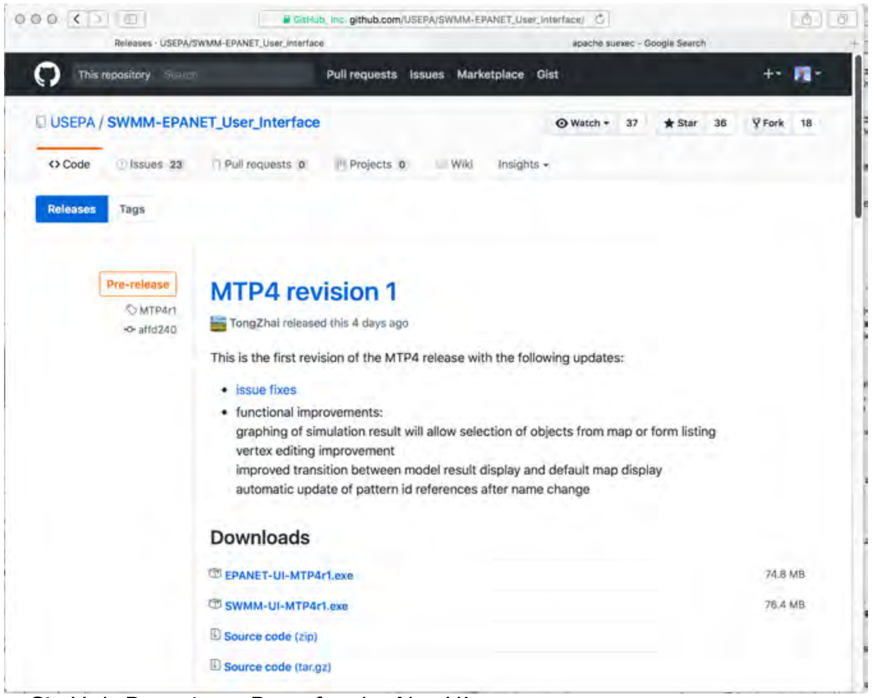
9.6 The Respec Interface

The `urltoinstallvideo` shows how to download and install EPANET onto your computer using the newer Respec GUI. This interface was built to be GIS aware and facilitate integration of an asset management database and the simulation software.

9.6.1 Installing the NewUI

The pre-release of the NewUI is located on a GitHub repository at `https://github.com/USEPA/SWMM-EPANET_User_Interface/releases`.

Figure 100 is a screen capture of the page.


GitHub, Inc. github.com/USEPA/SWMM-EPANET_User_Interface/
Releases · USEPA/SWMM-EPANET_User_Interface
apache suexec - Google Search
This repository Search
Pull requests Issues Marketplace Gist
USEPA / SWMM-EPANET_User_Interface
Watch 37 Star 36 Fork 18
Code Issues 23 Pull requests 0 Projects 0 Wiki Insights
Releases Tags
Pre-release
MTP4r1
affd240

MTP4 revision 1

TongZhai released this 4 days ago

This is the first revision of the MTP4 release with the following updates:

- issue fixes
- functional improvements:
  graphing of simulation result will allow selection of objects from map or form listing
  vertex editing improvement
  improved transition between model result display and default map display
  automatic update of pattern id references after name change

Downloads

EPANET-UI-MTP4r1.exe	74.8 MB
SWMM-UI-MTP4r1.exe	76.4 MB
Source code (zip)	
Source code (tar.gz)	

Figure 100. Git Hub Repository Page for the NewUI..

The file named `EPANET-UI-MTP4r1.exe` is a PC installer file that can be downloaded and installed. Select the file and begin the download.

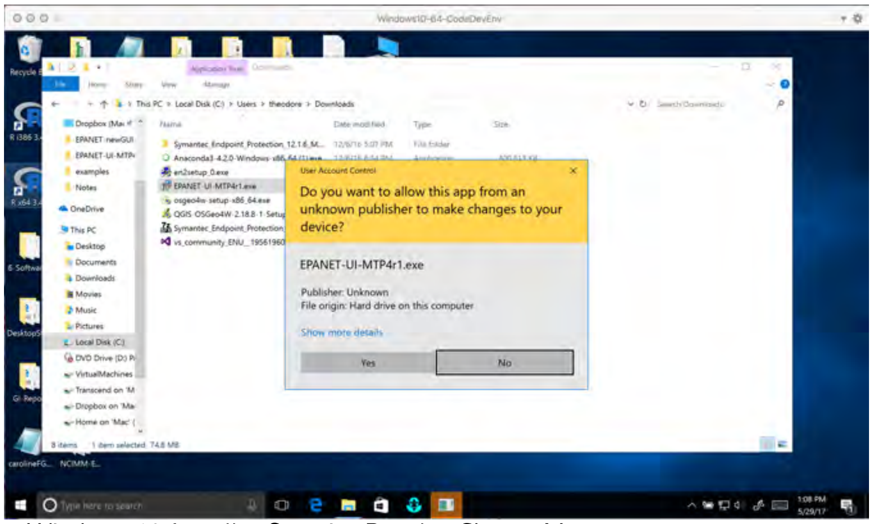
On the computer that is used for the screen captures in this document the following were installed in advance of the NewUI.

1. Python 3 (I used the Anaconda installer and installed the 64-bit Python).
2. QGIS. I used both the stand-alone `QGIS-OSGeo4W-2.18.8-1-Setup-x86_64.exe` and the over-the-wire `osgeo4w-setup-x86_64.exe` installer.

Python probably needs to be present, because the NewUI uses python scripts to access certain QGIS `.DLL` files for the interface. QGIS is handy because an environment variable needs to be set to point to a directory that contains a collection of `.CSV` files – these are located in the QGIS program.

The next series of screen captures document the installation from just after the download, to setting the environment variable.

Figure 101 is the install screen on Windows 10 where the OS is requesting permission to write to the disk. The user would choose yes.


User Account Control

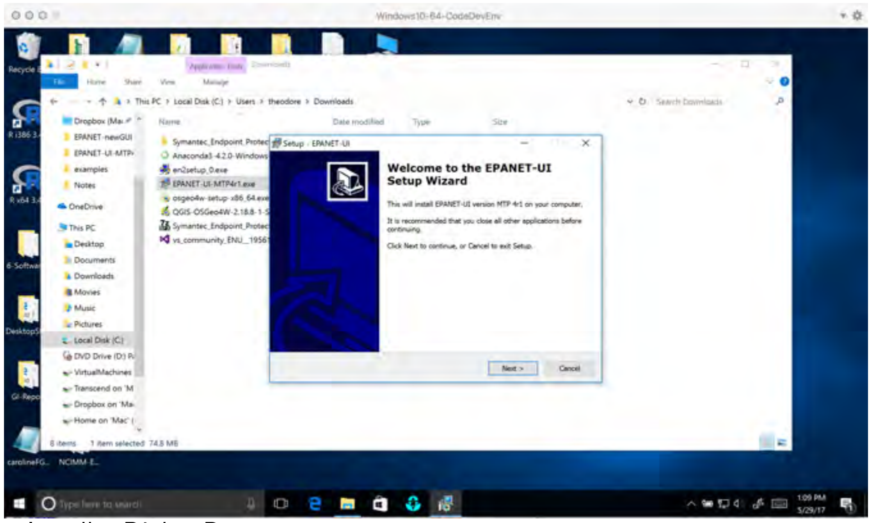
Do you want to allow this app from an unknown publisher to make changes to your device?

EPANET-UI-MTP4r1.exe

Publisher: Unknown
File origin: Hard drive on this computer

Figure 101. Windows 10 Installer Security Panel – Choose Yes.

Figure 102 is the first of several installer dialog boxes. The user would select next to begin the installation.



Welcome to the EPANET-UI Setup Wizard

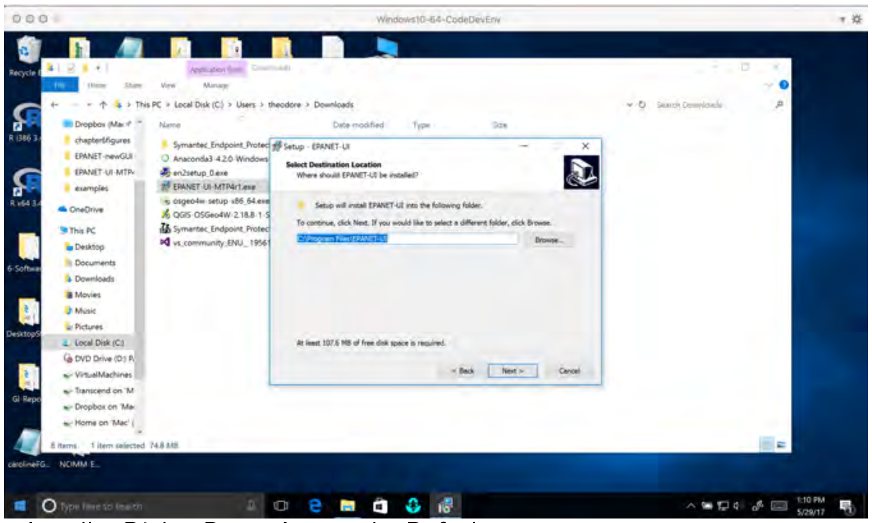
This will install EPANET-UI version MTP 4r1 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Figure 102. Installer Dialog Box.

Figure 103 is the next of several more installer dialog boxes. The user would select next to accept the install defaults and continue through a series of such screens.



Select Destination Location

Where should EPANET-UI be installed?

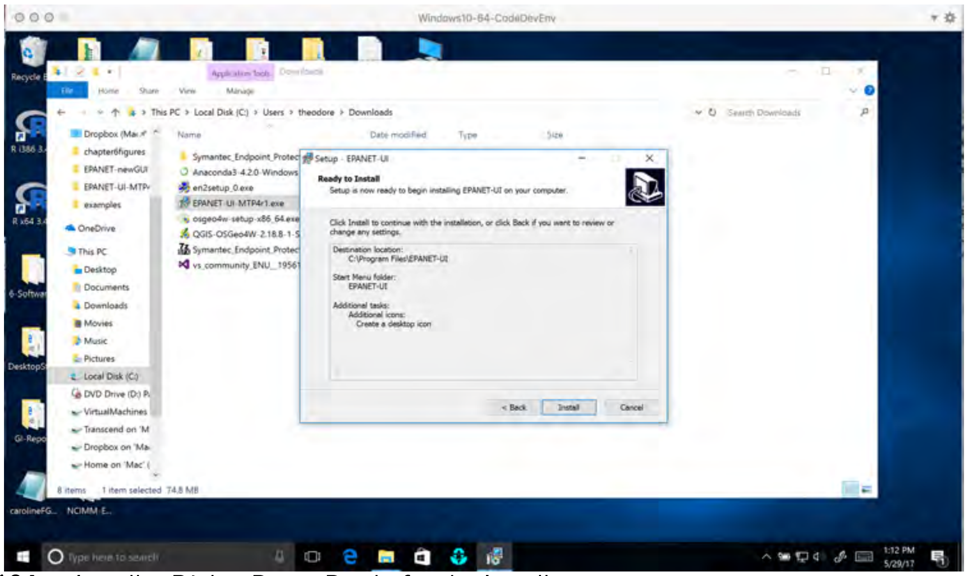
Setup will install EPANET-UI into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\EPANET-UI

At least 107.6 MB of free disk space is required.

Figure 103. Installer Dialog Box – Accept the Defaults.


Setup - EPANET-UI

Ready to Install

Setup is now ready to begin installing EPANET-UI on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

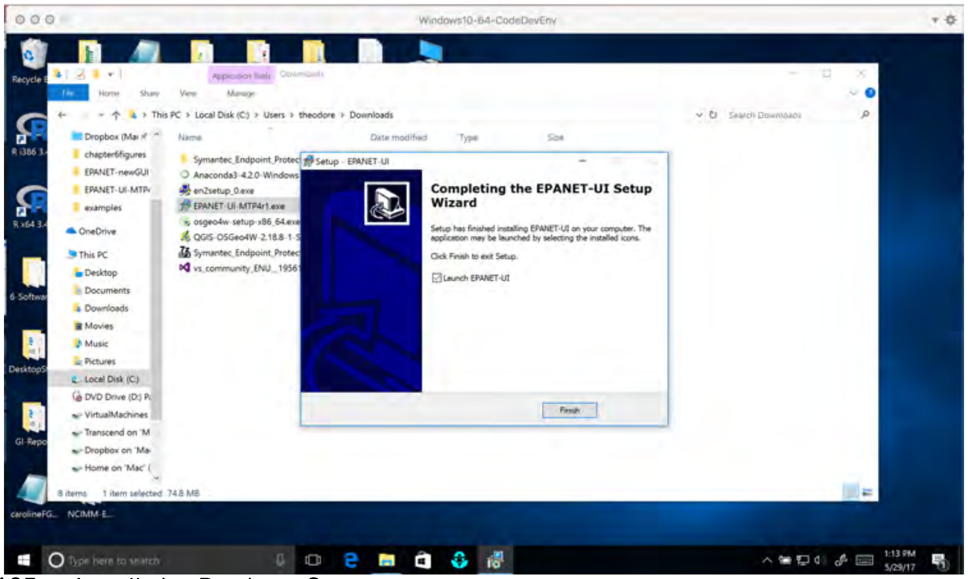
Destination location:
C:\Program Files\EPANET-UI

Start Menu folder:
EPANET-UI

Additional tasks:
Additional icons:
Create a desktop icon

Figure 104. Installer Dialog Box – Ready for the Install.

Figure 104 is the installer ready to begin the install. In the figure, we have selected a default installation and the creation of a desktop alias. The user would select install and the installation would begin. The program installs pretty fast, less than a minute.


Setup - EPANET-UI

Completing the EPANET-UI Setup Wizard

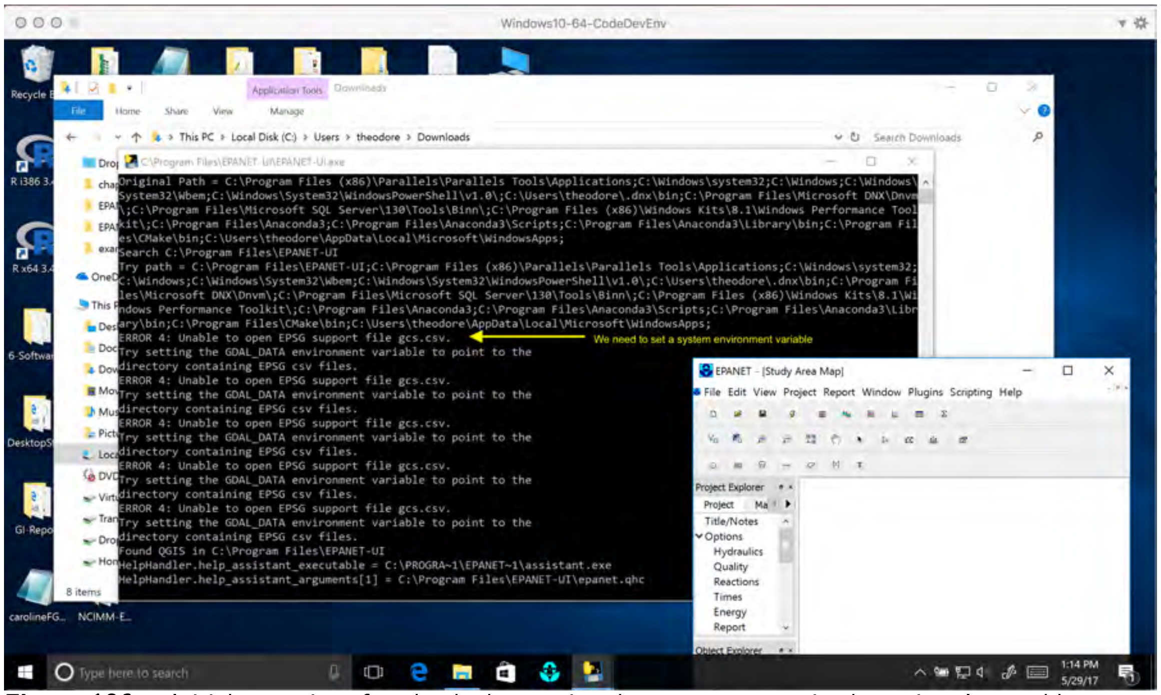
Setup has finished installing EPANET-UI on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

[✓] Launch EPANET-UI

Figure 105. Installed – Ready to Start.

Figure 105 is the completion screen, ready to start the program.



```
Original Path = C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\Windows\system32;C:\Windows;C:\Windows\
System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\theodore\.dnx\bin;C:\Program Files\Microsoft DNX\Dnvm
\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Tool
kit\;C:\Program Files\Anaconda3;C:\Program Files\Anaconda3\Scripts;C:\Program Files\Anaconda3\Library\bin;C:\Program Fil
es\CMake\bin;C:\Users\theodore\AppData\Local\Microsoft\WindowsApps;
Search C:\Program Files\EPANET-UI
Try path = C:\Program Files\EPANET-UI;C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\Windows\system32;
C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\theodore\.dnx\bin;C:\Program Fi
les\Microsoft DNX\Dnvm\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Windows Kits\8.1\Wi
ndows Performance Toolkit\;C:\Program Files\Anaconda3;C:\Program Files\Anaconda3\Scripts;C:\Program Files\Anaconda3\Libr
ary\bin;C:\Program Files\CMake\bin;C:\Users\theodore\AppData\Local\Microsoft\WindowsApps;
ERROR 4: Unable to open EPSG support file gcs.csv.
Try setting the GDAL_DATA environment variable to point to the
directory containing EPSG csv files.
ERROR 4: Unable to open EPSG support file gcs.csv.
Try setting the GDAL_DATA environment variable to point to the
directory containing EPSG csv files.
ERROR 4: Unable to open EPSG support file gcs.csv.
Try setting the GDAL_DATA environment variable to point to the
directory containing EPSG csv files.
ERROR 4: Unable to open EPSG support file gcs.csv.
Try setting the GDAL_DATA environment variable to point to the
directory containing EPSG csv files.
ERROR 4: Unable to open EPSG support file gcs.csv.
Try setting the GDAL_DATA environment variable to point to the
directory containing EPSG csv files.
Found QGIS in C:\Program Files\EPANET-UI
HelpHandler.help_assistant_executable = C:\PROGRA~1\EPANET~1\assistant.exe
HelpHandler.help_assistant_arguments[1] = C:\Program Files\EPANET-UI\epanet.qhc
```

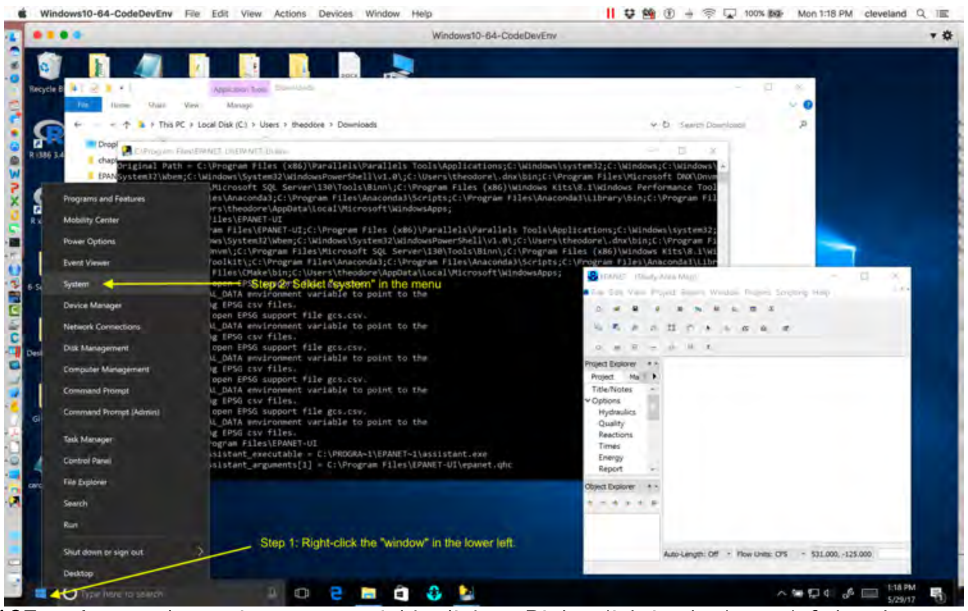
Windows10-64-CodeDevEnv
C:\Program Files\EPANET-UI\EPANET-UI.exe
We need to set a system environment variable
EPANET - [Study Area Map]
File Edit View Project Report Window Plugins Scripting Help
Project Explorer
Title/Notes
Options
Hydraulics
Quality
Reactions
Times
Energy
Report
This PC > Local Disk (C:) > Users > theodore > Downloads
8 items
1:14 PM
5/29/17

Figure 106. Initial run – interface loads, but notice the error message in the script. I was able to use the interface, but dealing with the error now is best. We will have to set an environment variable..

Figure 106 is the initial run of the NewUI. While it seems OK, notice the error message regarding `GDAL_DAT` directories. In my case, because GIS is a novelty to me, I don't have an existing environment for the `GDAL` libraries. The next several figures go through how to set the variable on my machine, the steps would be similar on user machines, with the variable pointing to the correct location. There might be some issue if the user cannot set variables in a network environment and they would need the system administrator to set the variable. The variable can also be set in a script.

In Windows 10, the user would right-click in the lower left hand corner of their desktop, on top of the window pane. A menu column should appear, and the user will select SYSTEM. Figure 107 is a screen capture showing these steps.

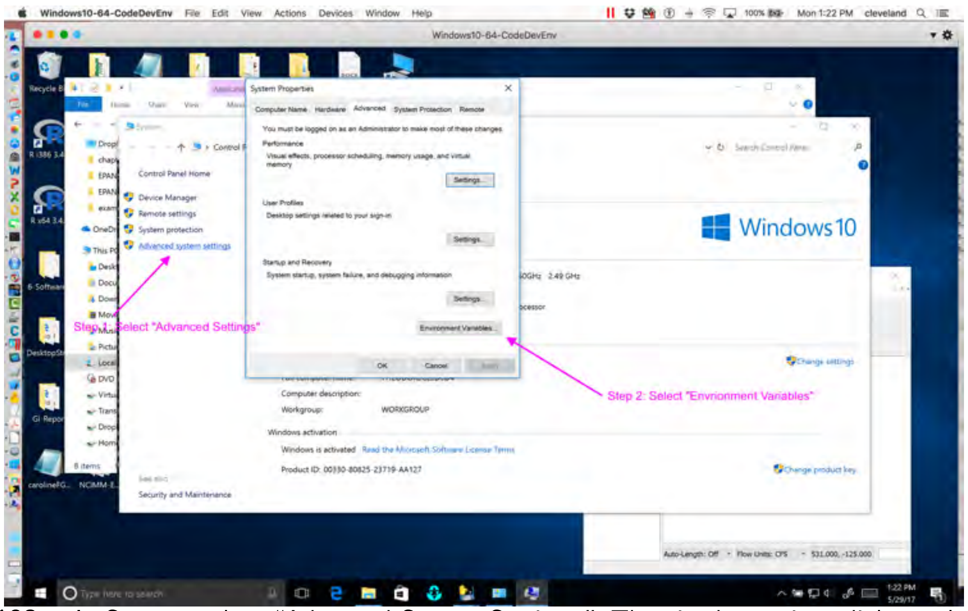
Upon entry into the system setting dialog box there is a link called ADVANCED SYSTEM SETTINGS in the left column. On my machine, it is the bottom item. Select ADVANCED SYSTEM SETTINGS The selection opens another dialog box called SYSTEM PROPERTIES. The bottom selection button is labeled ENVIRONMENT VARIABLES. Select this button to get to the next dialog box which will allow us to set the requisite variable.



```
Original Path = C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\Windows\system32;C:\Windows;C:\Windows\
System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\theodore\.dnx\bin;C:\Program Files\Microsoft DNX\Dnvm
Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Tool
es\Anaconda3;C:\Program Files\Anaconda3\Scripts;C:\Program Files\Anaconda3\Library\bin;C:\Program Fil
rs\theodore\AppData\Local\Microsoft\WindowsApps;
iles\EPANET-UI
am Files\EPANET-UI;C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\Windows\system32;
ws\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\theodore\.dnx\bin;C:\Program Fi
nvm\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Windows Kits\8.1\Wi
oolkit\;C:\Program Files\Anaconda3;C:\Program Files\Anaconda3\Scripts;C:\Program Files\Anaconda3\Libr
Files\CMake\bin;C:\Users\theodore\AppData\Local\Microsoft\WindowsApps;
open EPSG
L_DATA environment variable to point to the
g EPSG csv files.
open EPSG support file gcs.csv.
L_DATA environment variable to point to the
g EPSG csv files.
open EPSG support file gcs.csv.
L_DATA environment variable to point to the
g EPSG csv files.
open EPSG support file gcs.csv.
L_DATA environment variable to point to the
g EPSG csv files.
open EPSG support file gcs.csv.
L_DATA environment variable to point to the
g EPSG csv files.
ogram Files\EPANET-UI
sistant_executable = C:\PROGRA~1\EPANET~1\assistant.exe
sistant_arguments[1] = C:\Program Files\EPANET-UI\epanet.qhc
```

Figure 107. Access the environment variable dialog. Right-click in the lower left hand corner. Then select system..

Figure 108 is a screen capture of these steps.



System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

User Profiles
Desktop settings related to your sign-in
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK Cancel Apply

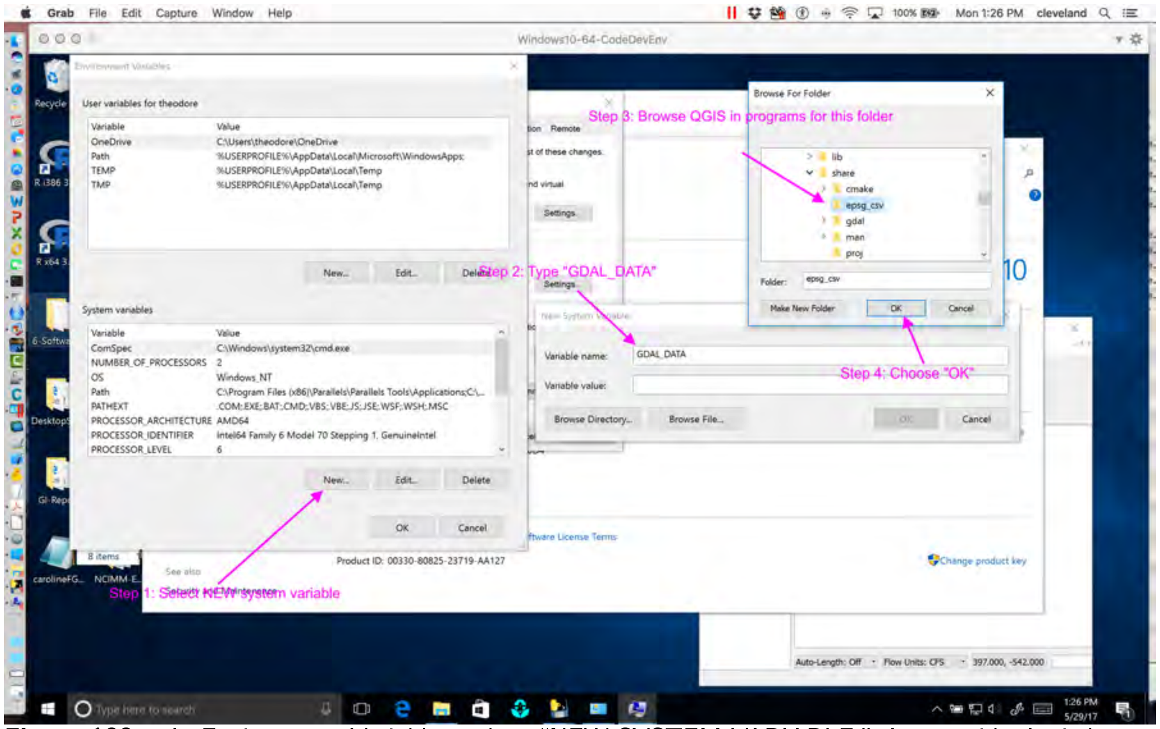
Computer description:
Workgroup: WORKGROUP

Windows activation
Windows is activated Read the Microsoft Software License Terms
Product ID: 00330-80825-23719-AA127

Figure 108. In System, select "Advanced System Settings." Then in the settings dialog, select "Environment Variables.".

Once the user has the ENVIRONMENT VARIABLE dialog box open, then we can set the variable. In the lower panel are the SYSTEM VARIABLES, and the user will select NEW SYSTEM VARIABLE. The new variable dialog box opens and we type the variable name, in this case `GDAL_DAT`. Next we need to set its value. The value is a path to the directory where the particular data structures reside. On my machine the variable is set to `C:/OSGeo4W64/share/epsg_csv`.

Other users will probably have different paths. Figure 109 is a screen capture illustrating these steps.



User variables for theodore

Variable	Value
OneDrive	C:\Users\theodore\OneDrive
Path	%USERPROFILE%\AppData\Local\Microsoft\WindowsApps;
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
NUMBER_OF_PROCESSORS	2
OS	Windows_NT
Path	C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 70 Stepping 1, GenuineIntel
PROCESSOR_LEVEL	6

Variable name: GDAL_DATA

Variable value:

Folder: epsg_csv

Product ID: 00330-80825-23719-AA127

Figure 109. In Environment Variables, select "NEW SYSTEM VARIABLE." As an aside, I tried to set the variable as a user environment variable (the upper part of the dialog) and did not shake the error. So the program is expecting a system variable that points to the directory `epsg_csv`. The directory is part of the Geospatial Data Abstraction Library and may be located in different palces on other machines. In particular, if a user already has ArcGIS, then conceivably the interface could access that directory..

After the selection the dialog box reports the variable and its value (before we select OK and get out of the system menu). Figure 110 illustrates the result. From this condition, the user should select OK and exit in reverse the various system dialog boxes until they are back to their desktop with the NewUI still running.

The next step is to exit the NewUI and restart the program. Upon the restart the error should be addressed and we are ready to build models.[37]

[37]The program runs OK with the error in place, but seemed to work better when the error was addressed.

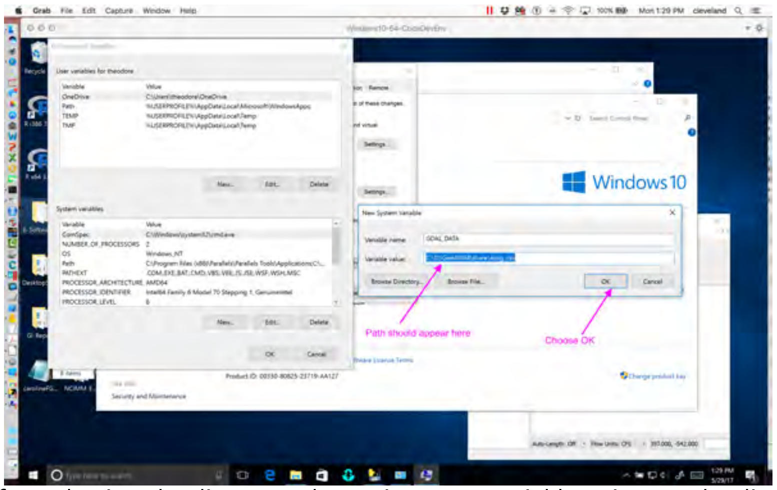
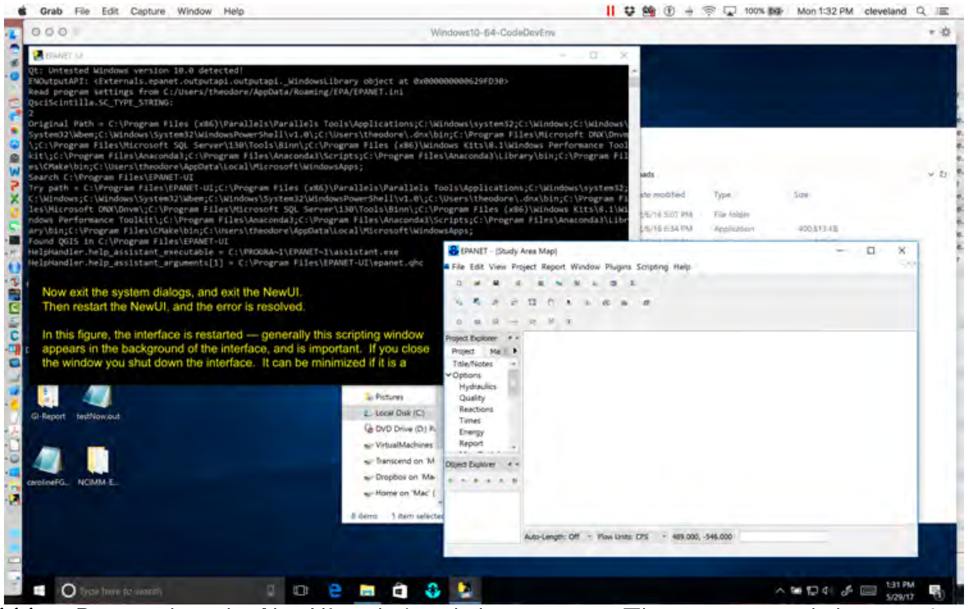


Figure 110. After selecting the directory, the environment variable points to the directory. Now ready to close the settings dialogs and return to the interface..



```
Qt: Untested Windows version 10.0 detected!
ENOutputAPI: <Externals.epanet.outputapi.outputapi._WindowsLibrary object at 0x000000000629FD30>
Read program settings from C:/Users/theodore/AppData/Roaming/EPA/EPANET.ini
QsciScintilla.SC_TYPE_STRING:
2
Original Path = C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\Windows\system32;C:\Windows;C:\Windows\
System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\theodore\.dnx\bin;C:\Program Files\Microsoft DNX\Dnvm
\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Tool
kit\;C:\Program Files\Anaconda3;C:\Program Files\Anaconda3\Scripts;C:\Program Files\Anaconda3\Library\bin;C:\Program Fil
es\CMake\bin;C:\Users\theodore\AppData\Local\Microsoft\WindowsApps;
Search C:\Program Files\EPANET-UI
Try path = C:\Program Files\EPANET-UI;C:\Program Files (x86)\Parallels\Parallels Tools\Applications;C:\Windows\system32;
C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Users\theodore\.dnx\bin;C:\Program Fi
les\Microsoft DNX\Dnvm\;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files (x86)\Windows Kits\8.1\Wi
ndows Performance Toolkit\;C:\Program Files\Anaconda3;C:\Program Files\Anaconda3\Scripts;C:\Program Files\Anaconda3\Libr
ary\bin;C:\Program Files\CMake\bin;C:\Users\theodore\AppData\Local\Microsoft\WindowsApps;
Found QGIS in C:\Program Files\EPANET-UI
HelpHandler.help_assistant_executable = C:\PROGRA~1\EPANET~1\assistant.exe
HelpHandler.help_assistant_arguments[1] = C:\Program Files\EPANET-UI\epanet.qhc
```

Now exit the system dialogs, and exit the NewUI.
Then restart the NewUI, and the error is resolved.

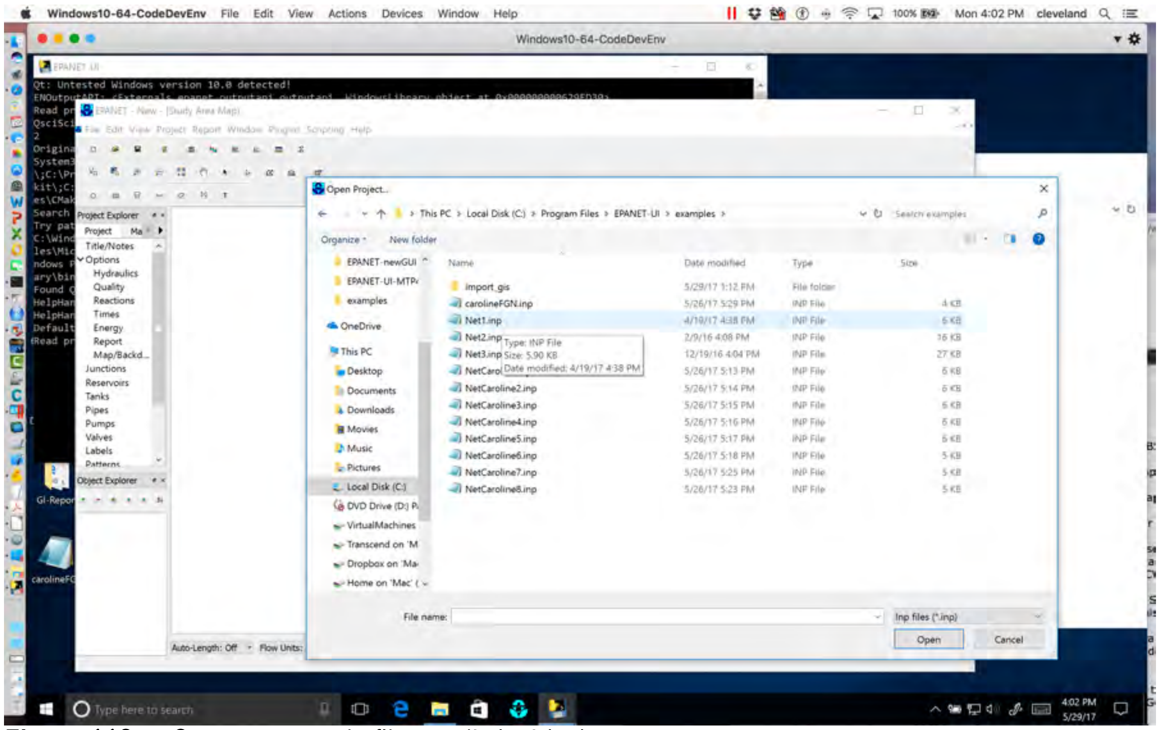
In this figure, the interface is restarted — generally this scripting window appears in the background of the interface, and is important. If you close the window you shut down the interface. It can be minimized if it is a

Figure 111. Returned to the NewUI and closed the program. Then restart, and the error is addressed..

9.7 Verify the Install

The easiest way to verify the install is to open one of the examples supplied with EPANET. They are located in some variant of `C:/ProgramFiles/EPANET-UI/Examples`.

Figure 112 illustrates selecting a file from the examples directory. On my machine I have already palced a few extra examples, the default install will only have three (3) examples.



```
Qt: Untested Windows version 10.0 detected!
```

Open Project...

This PC > Local Disk (C:) > Program Files > EPANET-UI > examples >

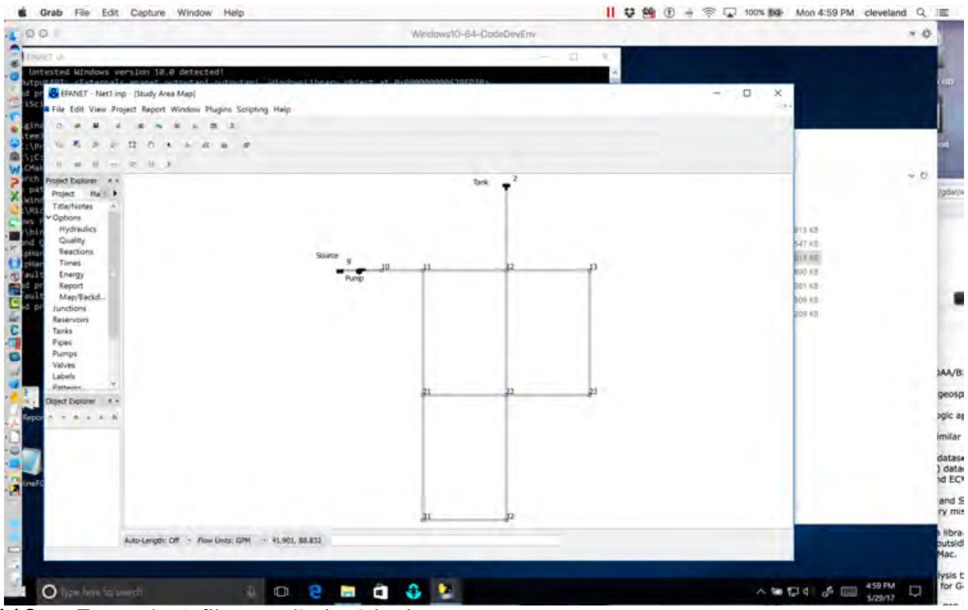
Name	Date modified	Type	Size
import_gis	5/29/17 1:12 PM	File folder	
carolineFGN.inp	5/26/17 5:29 PM	INP File	4 KB
Net1.inp	4/19/17 4:38 PM	INP File	6 KB
Net2.inp	2/9/16 4:08 PM	INP File	16 KB
Net3.inp	12/19/16 4:04 PM	INP File	27 KB
NetCaro	5/26/17 5:13 PM	INP File	6 KB
NetCaroline2.inp	5/26/17 5:14 PM	INP File	6 KB
NetCaroline3.inp	5/26/17 5:15 PM	INP File	6 KB
NetCaroline4.inp	5/26/17 5:16 PM	INP File	6 KB
NetCaroline5.inp	5/26/17 5:17 PM	INP File	6 KB
NetCaroline6.inp	5/26/17 5:18 PM	INP File	5 KB
NetCaroline7.inp	5/26/17 5:25 PM	INP File	5 KB
NetCaroline8.inp	5/26/17 5:23 PM	INP File	5 KB

Type: INP File
Size: 5.90 KB
Date modified: 4/19/17 4:38 PM

Figure 112. Open an example file supplied with the program..

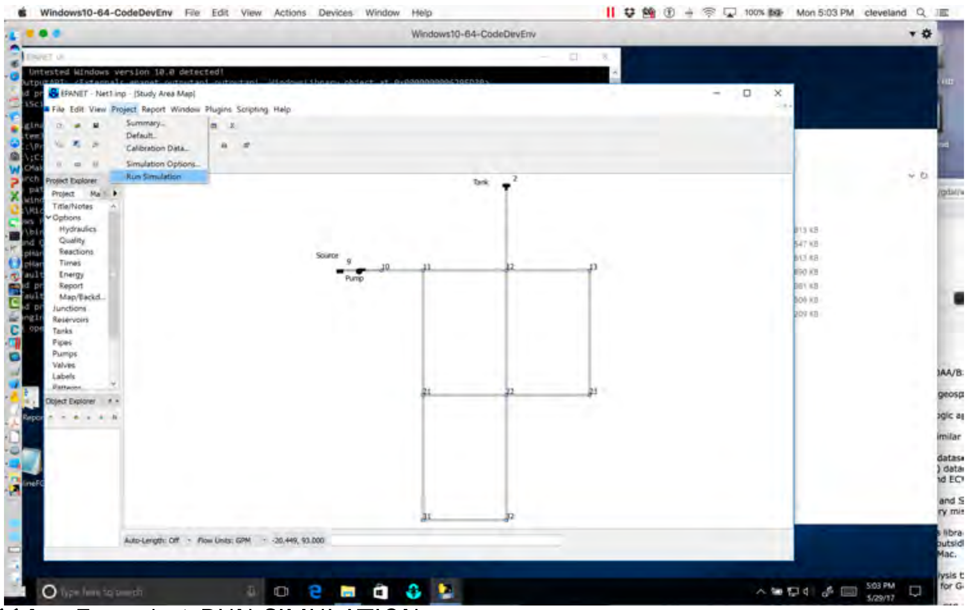
Upon opening the interface displays the image shown below in Figure 113. The example has a dozen pipes, a storage tank, supply reservoir, and a pump. The next step to verify the installation is to try to run EPANET from the NewUI.

Like the OldUI we can either choose to click on the “lightning bolt” icon, or choose RUN SIMULATION. The selection of RUN SIMULATION from the PROJECT menu tab is illustrated in Figure 114.


EPANET - Net1.inp - [Study Area Map]
File Edit View Project Report Window Plugins Scripting Help
Project Explorer
Tank
Source
Pump
Auto-Length: Off
Flow Units: GPM

```
Untested Windows version 10.0 detected!
```

Figure 113. Example 1 file supplied with the program..



```
Untested Windows version 10.0 detected!
```

Figure 114. Example 1 RUN SIMULATION..

Upon completion of the simulation, we can use the interface to interrogate the resultant database, for example, Figure 115 is a plot of discharge in the pipe connecting the pump to the network. It was constructed by activating SELECT OBJECT in the EDIT menu. The selecting the pipe object (double click to be sure the object ID is correct – the example does not have pipe labels activated). Then selecting the time series icon in the interface. The user then selects the plot type (TIME SERIES), the object type (LINK), the object ID (Link 10), the plot parameter (FLOW), and finally OK.

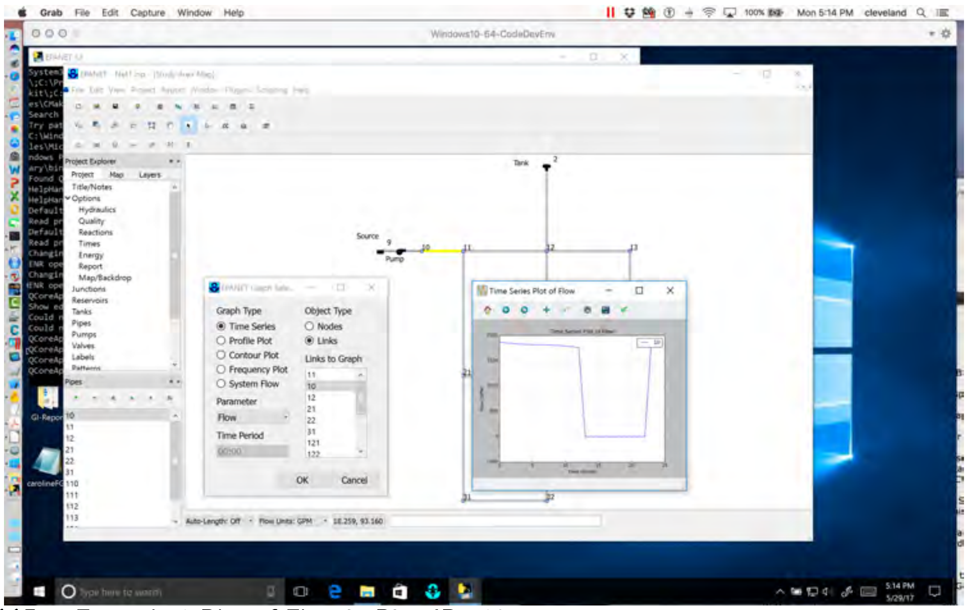


Figure 115. Example 1 Plot of Flow in Pipe ID=10..

The next part of this document illustrates building simple models in the interface without regards to geo-referencing. These examples are based on the examples presented in
`http://www.rtfmps.com/university-courses/ce-3372/3-Readings/EPANETbyExample/`
and are illustrative of elementary examples used in teaching the use of EPANET as a hydraulic tool to college students.

Additional pipe network specific training materials can be found in Lectures 9 through 13 located at `http://www.rtfmps.com/university-courses/ce-3372/1-Lectures/`

9.8 EPA-NET Modeling by Example

EPA-NET models are comprised of nodes, links,and reservoirs. Pumps are treated as special links (that add head). Valves are also treated as special links depending on the valve types. All models must have a reservoir (or storage tank).

9.8.1 Defaults

The program has certain defaults that should be set at the beginning of a simulation. The main defaults of importance are the head loss equations (Darcy-Weisbach, Hazen-Williams, or Chezy-Manning) and the units (CFS, LPS, etc.)

9.8.2 Example 1: Flow in a Single Pipe using OldUI and NewUI

The simplest model to consider is from an earlier exercise in this workbook.

> A 5-foot diameter, enamel coated, steel pipe carries 60^oF water at a discharge of 295 cubic-feet per second (cfs). Using the Moody chart, estimate the head loss in a 10,000 foot length of this pipe.

In EPA-NET we will start the program, build a tank-pipe system and find the head loss in a 10,000 foot length of the pipe. The program will compute the friction factor for us (and we can check on the Moody chart if we wish).

The main trick in EPA-NET is going to be the friction coefficient, in the EPA-NET manual on page 30 and 31, the author indicates that the program expects a roughness coefficient based on the head loss equation. The units of the roughness coefficient for a steel pipe are 0.15×10^{-3} feet. On page 71 of the user manual the author states that roughness coefficients are in millifeet (millimeters) when the Darcy-Weisbach head loss model is used. So keeping that in mind we proceed with the example.

Figure 116 is a screen capture of the EPA-NET program after installing the program using the OldUI. The program starts as a blank slate and we will select a reservoir and a node from the tool bar at the top and place these onto the design canvas.

Figure 116 is a screen capture of the EPA-NET program after installing the program using the NewUI.

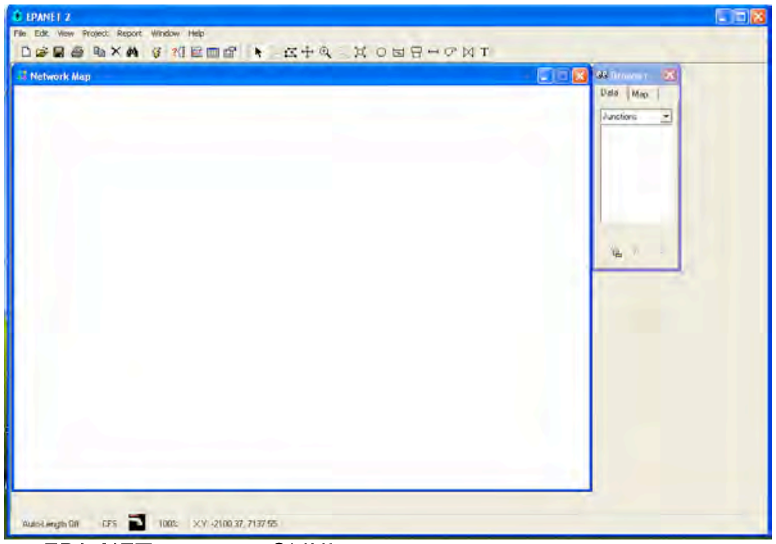

EPANET 2
File Edit View Project Report Window Help
Network Map
Browser
Data Map
Junctions
Auto-Length Off CFS 100% X,Y: -2100.37, 7137.55

Figure 116. Start EPA-NET program, OldUI.

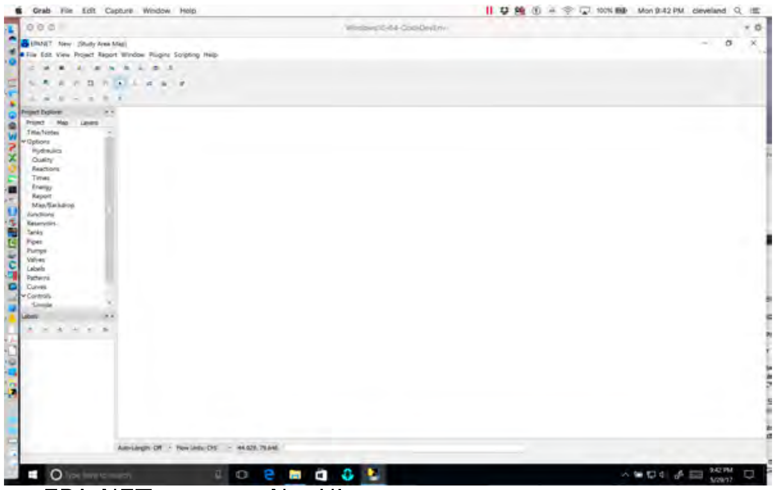
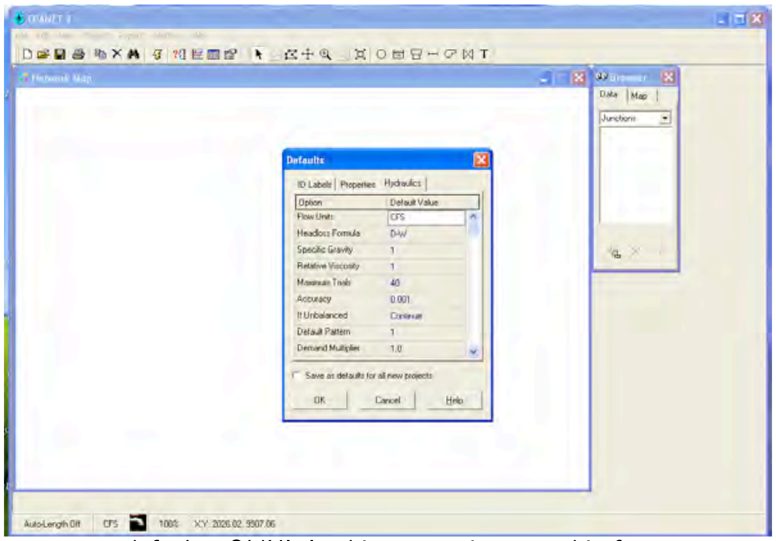


Figure 117. Start EPA-NET program, NewUI.

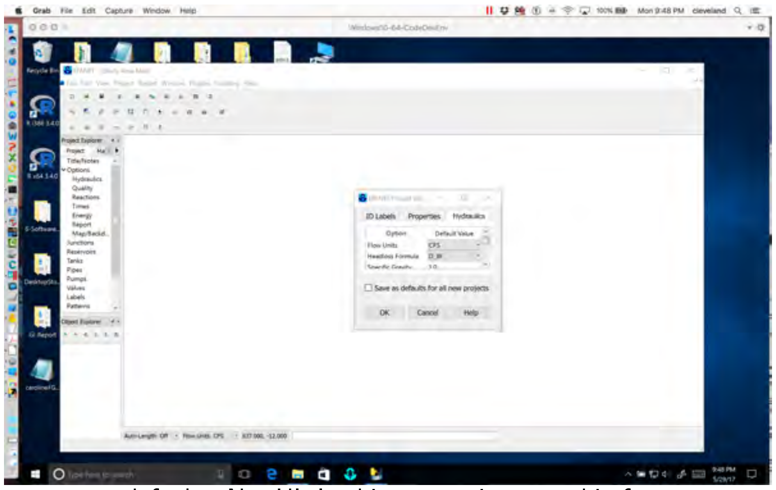
Figure 118 is a screen capture of the EPA-NET program after setting defaults for the simulation. Failure to set correct units for your problem are sometimes hard to detect (if the model runs), so best to make it a habit to set defaults for all new projects.



Option	Default Value
Flow Units	CFS
Headloss Formula	D-W
Specific Gravity	1
Relative Viscosity	1
Maximum Trials	40
Accuracy	0.001
If Unbalanced	Continue
Default Pattern	1
Demand Multiplier	1.0

Figure 118. Set program defaults, OldUI. In this case units are cubic-feet-per-second and loss model is Darcy-Weisbach..

Figure 119 is a screen capture of the EPA-NET program after setting defaults for the simulation using the NewUI.



Option	Default Value
Flow Units	CFS
Headloss Formula	D_W
Specific Gravity	1.0

Figure 119. Set program defaults, NewUI. In this case units are cubic-feet-per-second and loss model is Darcy-Weisbach..

Next we add the reservoir and the node. Figure 120 is a screen capture after the reservoir and node is placed.

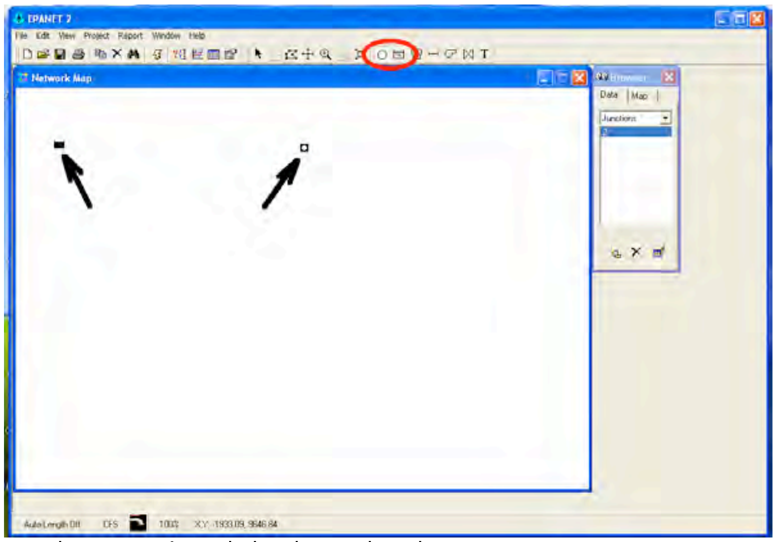


Figure 120. Place the reservoir and the demand node..

Figure 121 is a screen capture after the reservoir and node is placed using the NewUI. There is a caveat here – without a coordinate system the program assigns each object the same coordinates, so the user has to manually insert the X and Y coordinates into the objects. I suspect if a coordinate reference system is pre-assigned by virtue of a GIS vector or raster layer, then the system would correctly report (and input) coordinates.

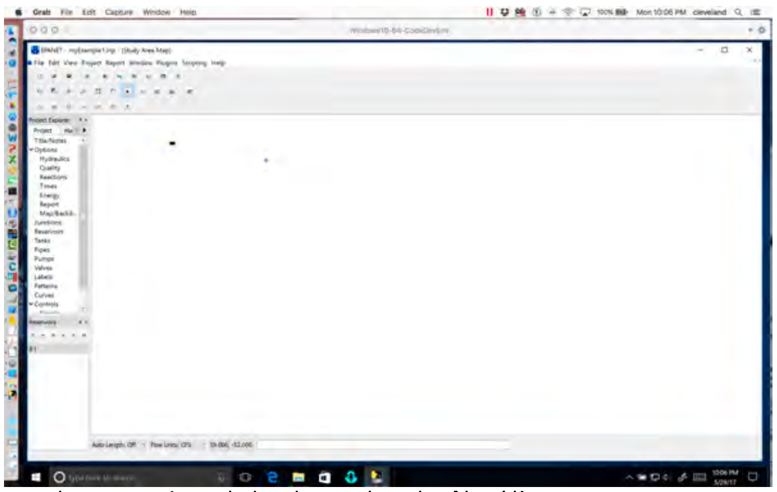


Figure 121. Place the reservoir and the demand node, NewUI..

We will specify a total head at the reservoir (value is unimportant as long as it is big enough to overcome the head loss and not result in a negative pressure at the node. We will specify the demand at the node equal to the desired flow in the pipe. Figure 122 is a screen capture after the pipe is placed. The sense of flow in this example is from reservoir to node, but if we had it backwards we could either accept a negative flow in the pipe, or right-click the pipe and reverse the start and end node connections.

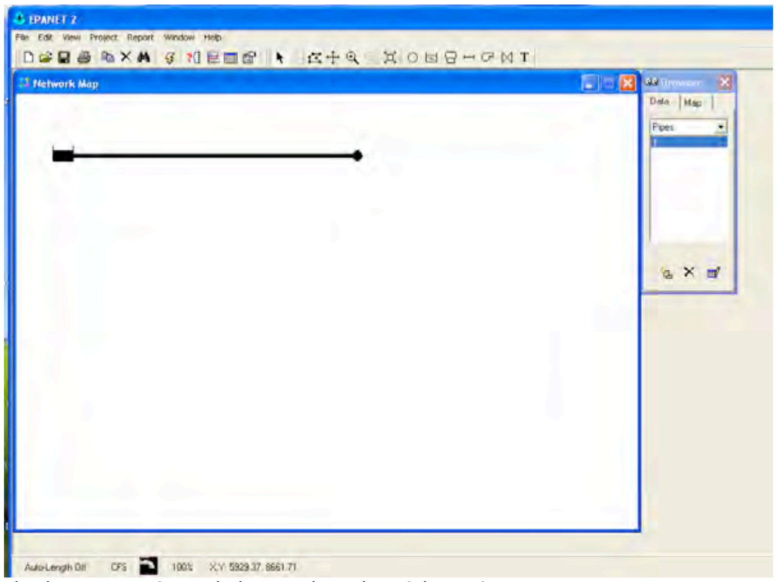

EPANET 2
File Edit View Project Report Window Help
Network Map
Browser
Data Map
Pipes
Auto-Length Off CFS 100% X,Y: 5929.37, 8661.71

Figure 122. Link the reservoir and demand node with a pipe..

Figure 123 is a screen capture after the pipe is placed using the NewUI. The main challenge here is to actually draw and connect the pipe.

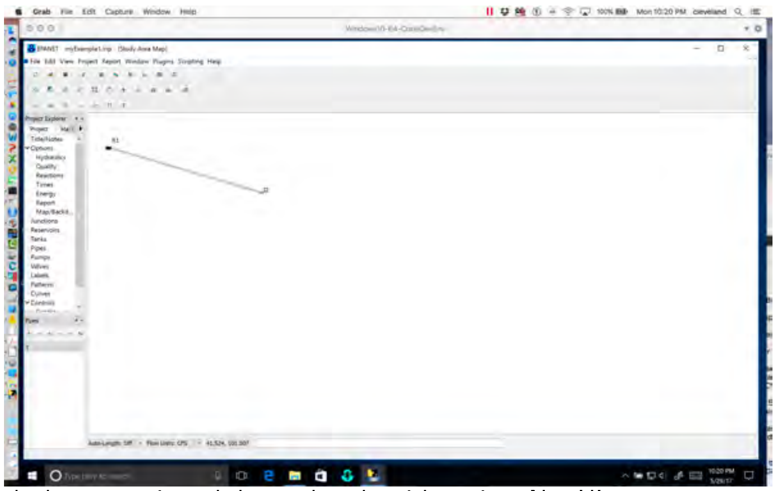
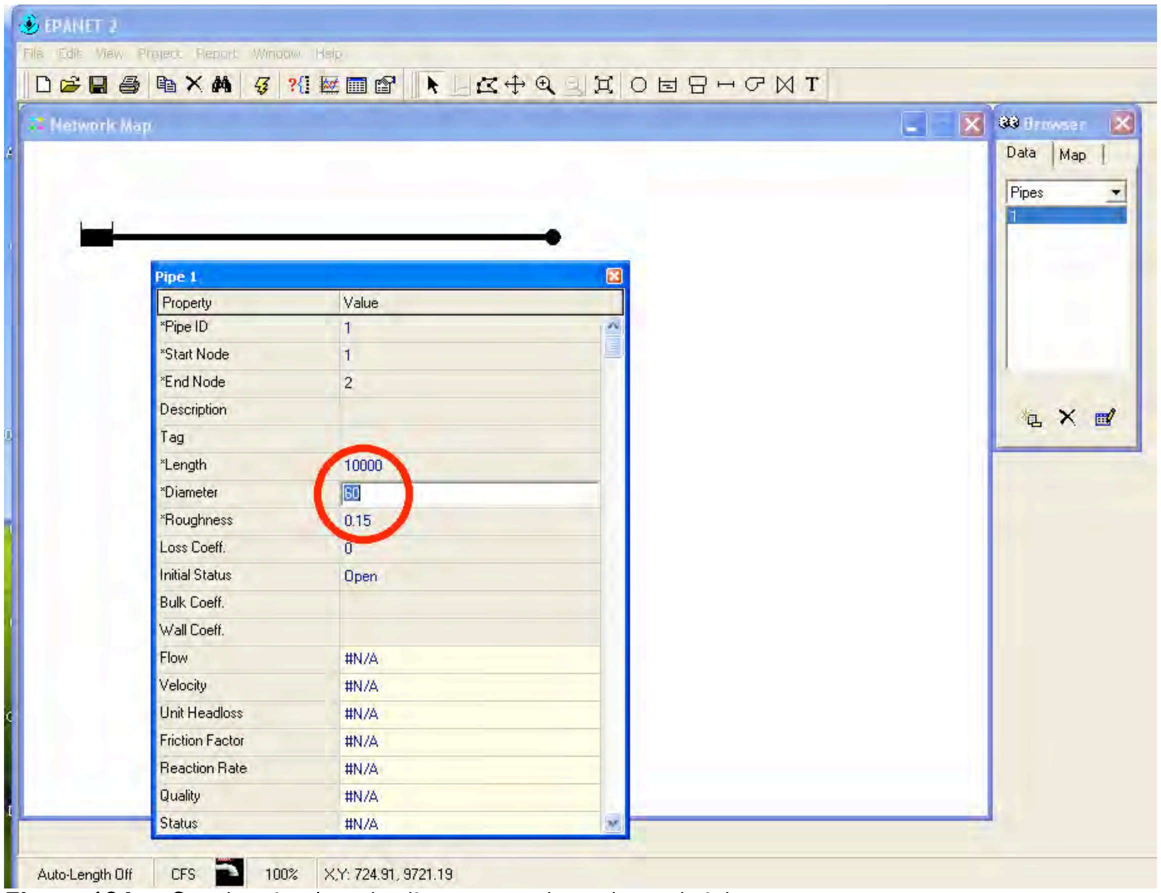


Figure 123. Link the reservoir and demand node with a pipe, NewUI..

The pipe icon is similar to the OldUI, and using the select arrow we select the pipe icon. A drawing cross-hair appears, and we lay that on top of the reservoir and click. Then as we drag the crosshair to the junction, a red dashed line follows the crosshair. When we get to the junction we position the crosshair over the junction and right click – that action connects the two objects with the link object (at least visually).

Now we can go back to each hydraulic element in the model and edit the properties. We supply pipe properties (diameter, length, roughness height) as in Figure 124. Fig-

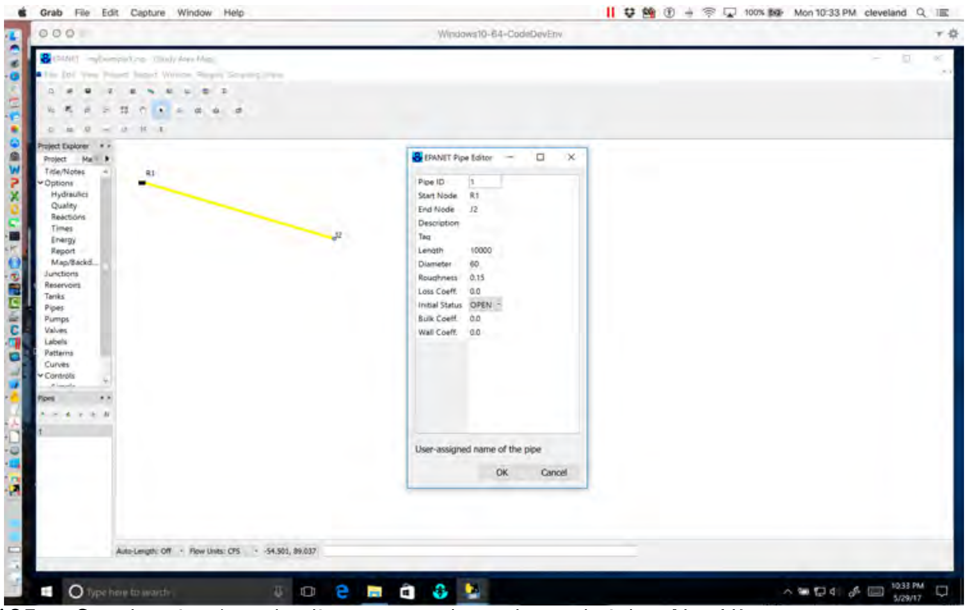


Pipe 1

Property	Value
*Pipe ID	1
*Start Node	1
*End Node	2
Description	
Tag	
*Length	10000
*Diameter	60
*Roughness	0.15
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	#N/A
Velocity	#N/A
Unit Headloss	#N/A
Friction Factor	#N/A
Reaction Rate	#N/A
Quality	#N/A
Status	#N/A

Figure 124. Set the pipe length, diameter, and roughness height..

ure 125 shows setting the properties using the NewUI. We choose SELECT OBJECT from the EDIT menu, then select the pipe. It [pipe] should change color to indicate that it is selected – on my computer it changes to yellow. When it is selected, we can double-click to obtain the properties, or choose the properties from the tool menu on the left side of the design canvass. The properties are then supplied as in the OldUI.

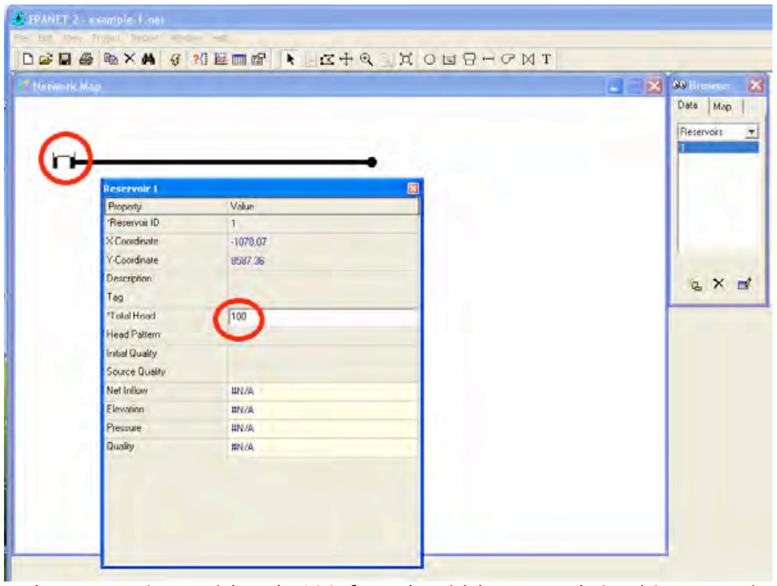


Pipe ID	1
Start Node	R1
End Node	J2
Description	
Tag	
Length	10000
Diameter	60
Roughness	0.15
Loss Coeff.	0.0
Initial Status	OPEN
Bulk Coeff.	0.0
Wall Coeff.	0.0

User-assigned name of the pipe

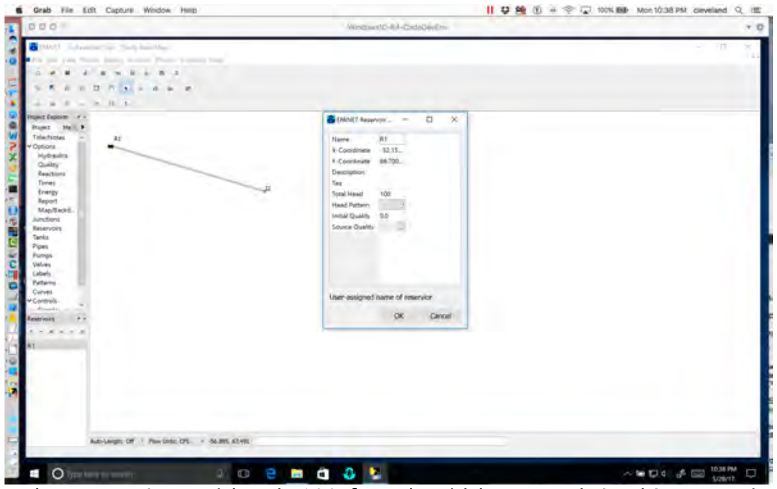
Figure 125. Set the pipe length, diameter, and roughness height. NewUI.

Using a similar selection process we supply the reservoir total head as in Figure 126 and Figure 127



Property	Value
*Reservoir ID	1
X-Coordinate	-1070.07
Y-Coordinate	8587.36
Description	
Tag	
*Total Head	100
Head Pattern	
Initial Quality	
Source Quality	
Net Inflow	#N/A
Elevation	#N/A
Pressure	#N/A
Quality	#N/A

Figure 126. Set the reservoir total head, 100 feet should be enough in this example..

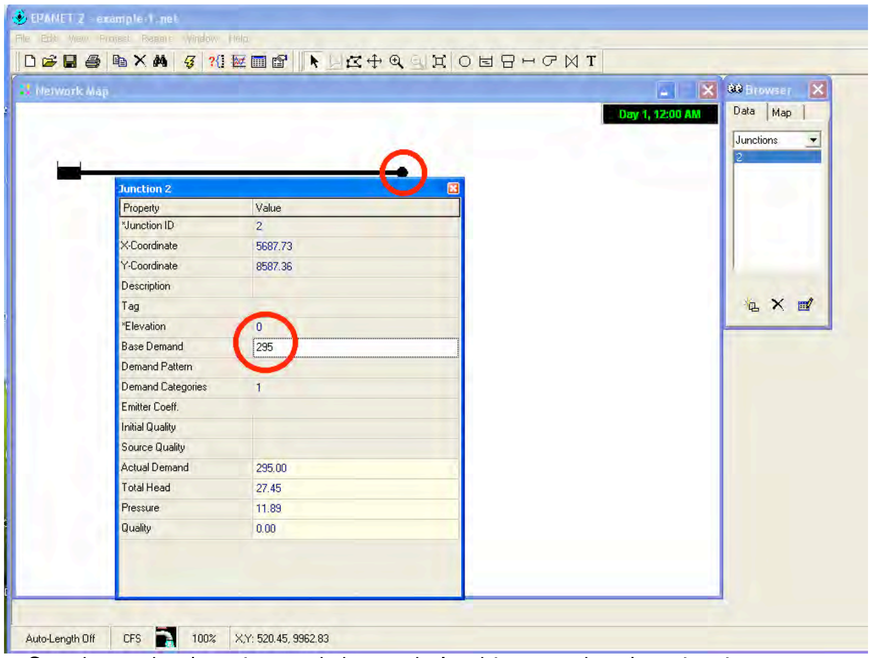

EPANET Reservoir ...

Name	R1
X-Coordinate	-52.15...
Y-Coordinate	66.700...
Description	
Tag	
Total Head	100
Head Pattern	
Initial Quality	0.0
Source Quality	

User-assigned name of reservoir

Figure 127. Set the reservoir total head, 100 feet should be enough in this example..

We then set the demand node elevation and the actual desired flow rate as in Figure 128. Figure 129 illustrates the same process in the NewUI.



Junction 2

Property	Value
*Junction ID	2
X-Coordinate	5687.73
Y-Coordinate	8587.36
Description	
Tag	
*Elevation	0
Base Demand	295
Demand Pattern	
Demand Categories	1
Emitter Coeff.	
Initial Quality	
Source Quality	
Actual Demand	295.00
Total Head	27.45
Pressure	11.89
Quality	0.00

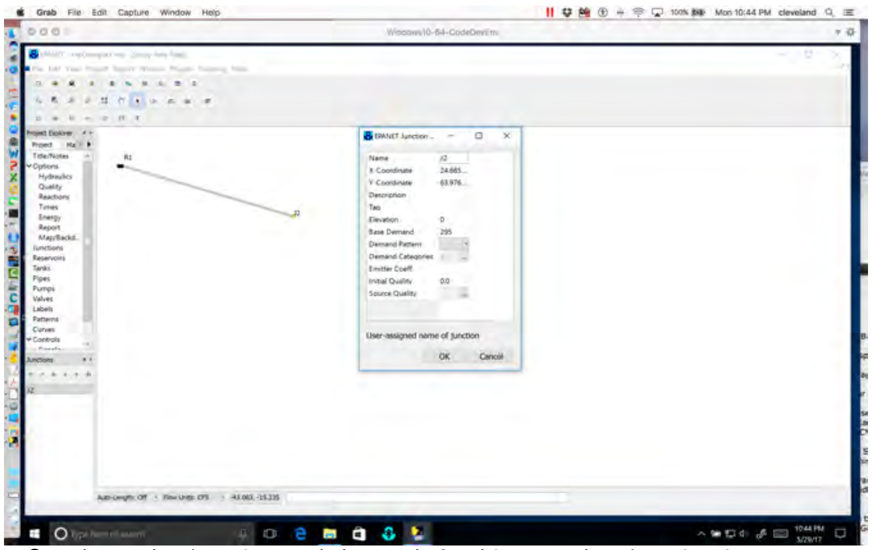
Auto-Length Off CFS 100% X,Y: 520.45, 9962.83

Figure 128. Set the node elevation and demand. In this case the elevation is set to zero (the datum) and the demand is set to 295 cfs as per the problem statement..

Again the procedure is select the object, then set the values.

The program is now ready to run, next step would be to save the input file (File/Save/-Name), then run the program.

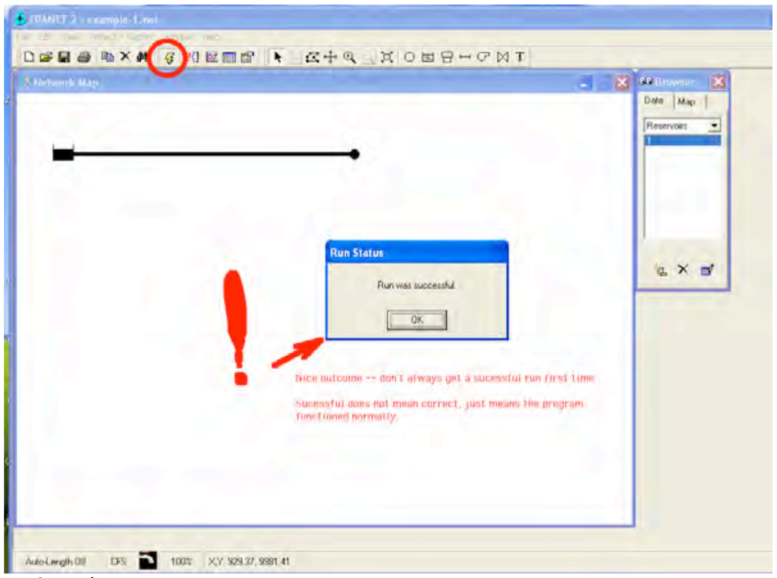
Run the program by selecting the lighting bolt looking thing (kind of channeling Zeus here) and the program will start. If the nodal connectivity is OK and there are no computed negative pressures the program will run. Figure 130 is the appearance of


EPANET Junction ...

Name	J2
X-Coordinate	24.685...
Y-Coordinate	63.976...
Description	
Tag	
Elevation	0
Base Demand	295
Demand Pattern	
Demand Categories	
Emitter Coeff.	
Initial Quality	0.0
Source Quality	

User-assigned name of junction

Figure 129. Set the node elevation and demand. In this case the elevation is set to zero (the datum) and the demand is set to 295 cfs as per the problem statement..


Run Status

Run was successful.

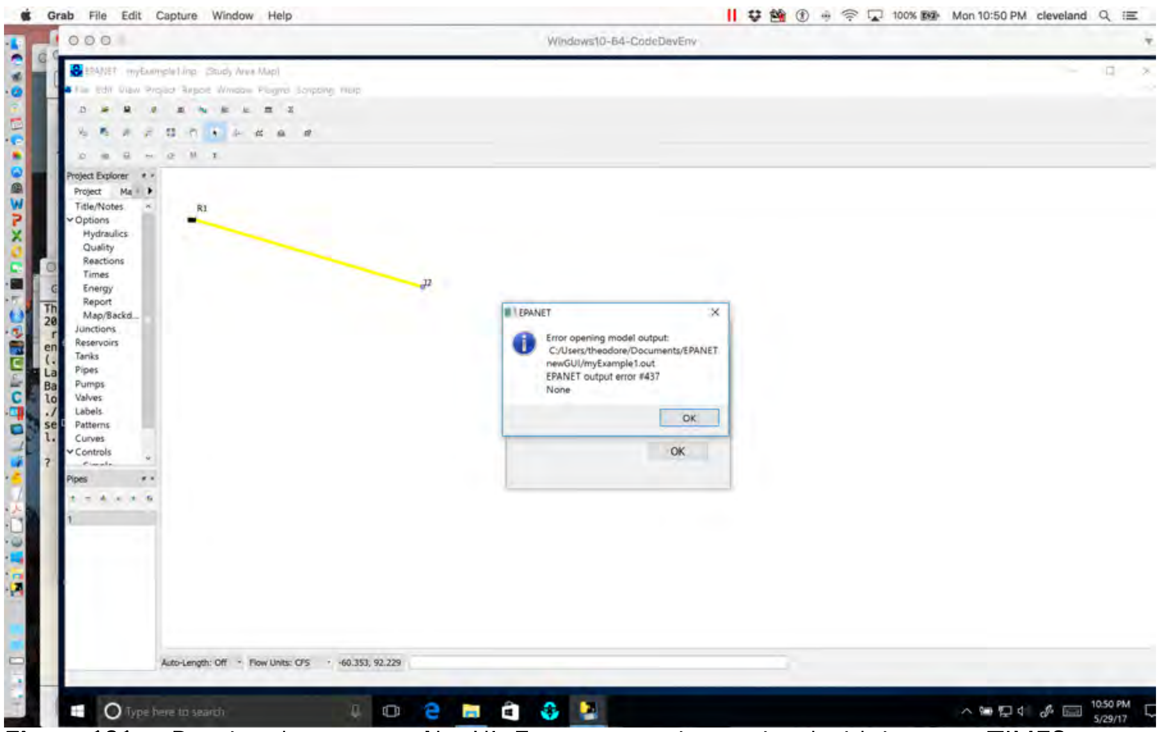
Nice outcome -- don't always get a sucessful run first time

Sucessful does not mean correct, just means the program functioned normally.

Figure 130. Running the program.

the program after the run is complete (the annotations are mine!). A successful run means the program found an answer to the problem you provided – whether it is the correct answer to your problem requires the engineer to interpret results and decide if they make sense. The more common conceptualization errors are incorrect units and head loss equation for the supplied roughness values, missed connections, and forgetting demand somewhere. With practice these kind of errors are straightforward to detect. In the present example we select the pipe and the solution values are reported at the bottom of a dialog box.

Figure 131 is the result using the NewUI. This is where the two systems diverge. This particular error can be addressed by setting the simulation time to be at least one

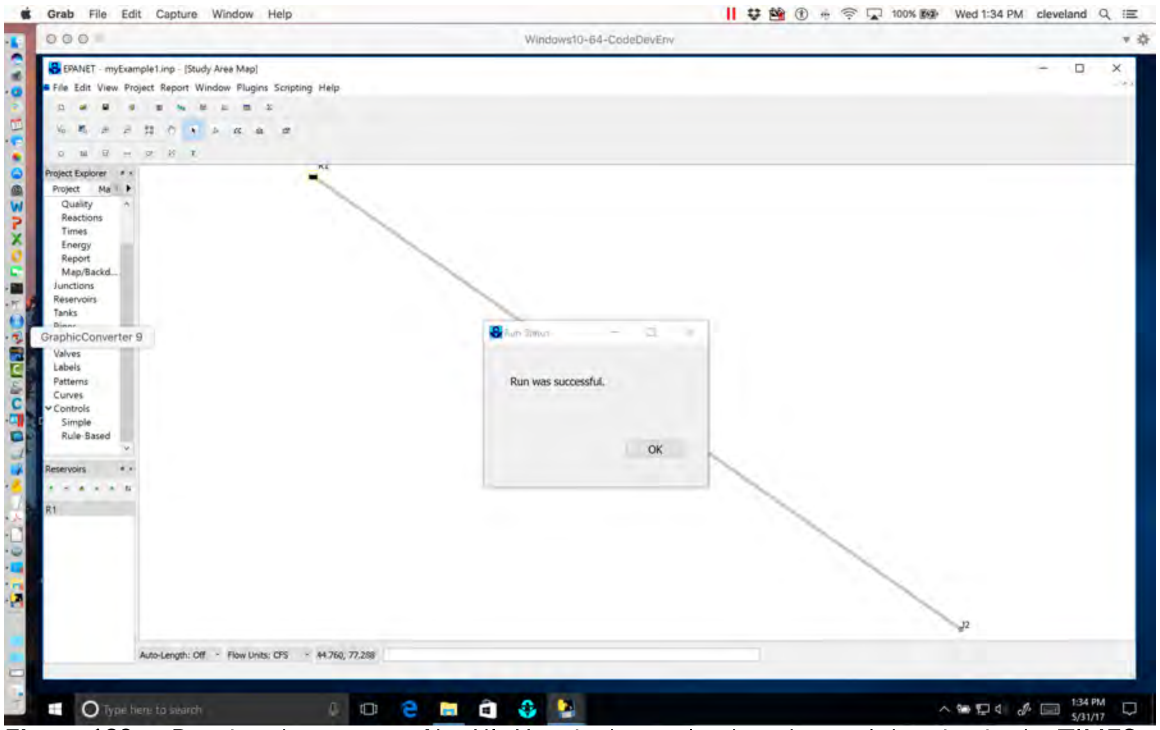


```
Error opening model output:
 C:/Users/theodore/Documents/EPANET
newGUI/myExample1.out
EPANET output error #437
None
```

Figure 131. Running the program, NewUI. Error message is associated with incorrect TIMES entry. Using the NewUI, the program expects a non-zero simulation duration. A simple hack is to specify the duration equal to one hydraulic time step..

time increment (it is still a steady flow, single time period simulation). Once we make this change the program will run successfully.

Figure 132 is the result using the NewUI. To produce this figure the menu item TIMES was selected and the total duration (first record in the dialog box) was changed from the default 0 to a value of 1:00. The change roughly tells the program to simulate 1:00 hours of system operation, using a hydraulic time step of 1:00 hours (the default). The OldUI would have interpreted the 0 as “single period simulation.” The NewUI probably expects a non-zero value because it is designed to support both SWMM and EPANET computation engines. Interestingly, the simulation is still runs, but produces an output file that the interface cannot access.

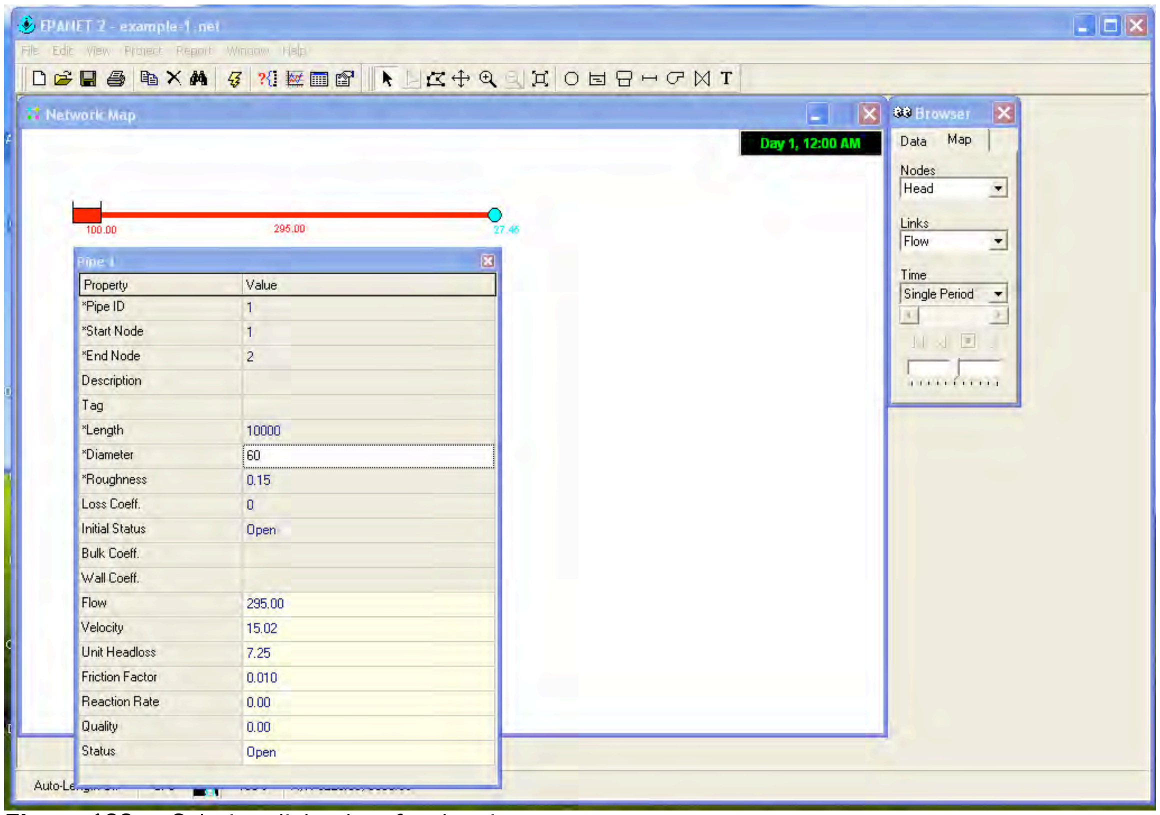


Run was successful.

Figure 132. Running the program, NewUI. Here is the result when the total duration in the TIMES menu item is set to 1:00. The program runs as anticipated..

Figure 133 is the result of turning on the computed head values at the node (and reservoir) and the flow value for the pipe in the OldUI. The dialog box reports about 7.2 feet of head loss per 1000 feet of pipe for a total of 72 feet of head loss in the system. The total head at the demand node is about 28 feet, so the head loss plus remaining head at the node is equal to the 100 feet of head at the reservoir, the anticipated result.

The computed friction factor is 0.010, which we could check against the Moody chart if we wished to adjust the model to agree with some other known friction factor.



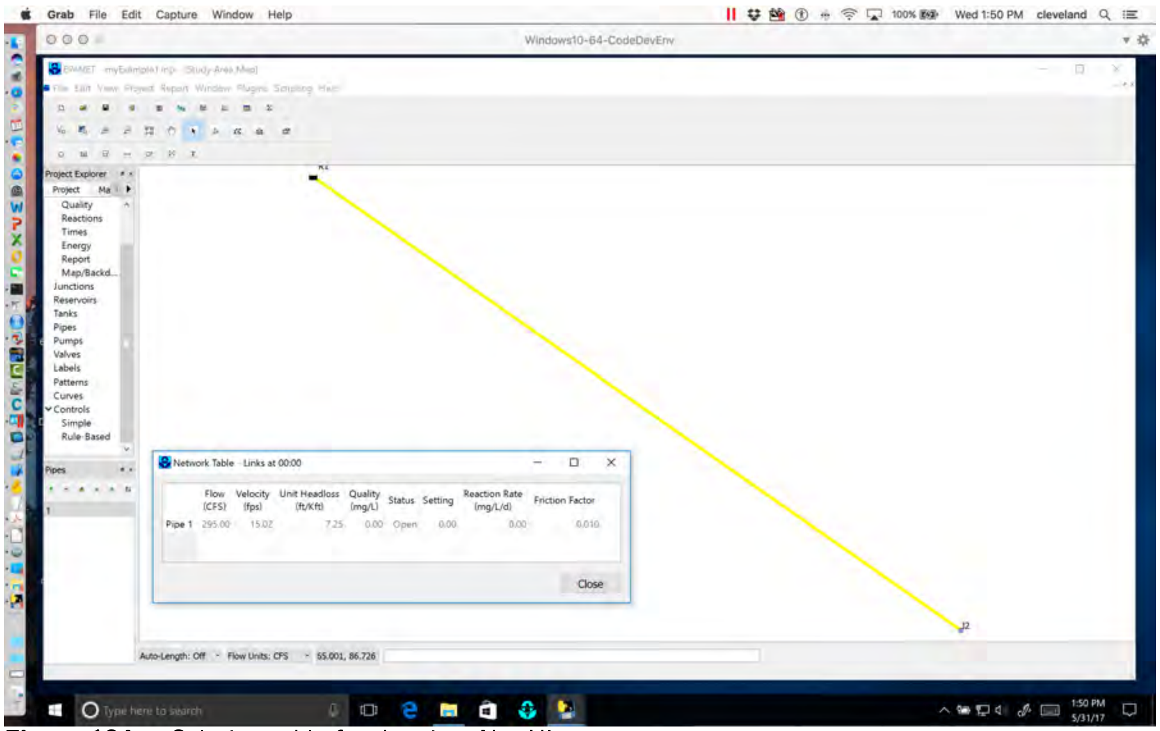
Property	Value
*Pipe ID	1
*Start Node	1
*End Node	2
Description	
Tag	
*Length	10000
*Diameter	60
*Roughness	0.15
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	295.00
Velocity	15.02
Unit Headloss	7.25
Friction Factor	0.010
Reaction Rate	0.00
Quality	0.00
Status	Open

Figure 133. Solution dialog box for the pipe..

Using the NewUI, the object dialog box does not capture the computed information. Instead we can generate a table of values associated with the pipe object. Figure 134 is such a table. It was created after the program run by selecting the pipe object, then selecting the table icon. Once the table dialog box is opened, select NODES and the table is generated.

The table reports 7.25 feet of head loss per 1000 feet of pipe for a total of 72.5 feet of head loss in the system. The total head at the demand node is 27.5 feet, so the head loss plus remaining head at the node is equal to the 100 feet of head at the reservoir, the same result as in the OldUI model.

Concluding remarks for the example are the NewUI produces the same results as the



	Flow (CFS)	Velocity (fps)	Unit Headloss (ft/Kft)	Quality (mg/L)	Status	Setting	Reaction Rate (mg/L/d)	Friction Factor
Pipe 1	295.00	15.02	7.25	0.00	Open	0.00	0.00	0.010

Figure 134. Solution table for the pipe, NewUI..

OldUI for an identical problem.[38] The NewUI reports results in a different fashion – that is the computed results are now separated from the object properties, hence the user will interrogate the results differently (using tables and charts most likely). This different approach makes logical sense (separating input properties and computed properties).

The NewUI **REQUIRES** specification of a simulation duration that is non-zero, for single period simulations the simplest hack is to specify the simulation duration equal to the value of the hydraulic time step, and proceed.

Users familiar with the OldUI will find the NewUI similar, but some things will be different. An improvement is the NewUI generates reports directly to an ASCII file, then opens that file rather than in the OldUI, having to navigate to the file and open it independently. Another improvement is the report file extension being `.TXT`, so it is automatically associated with an ASCII editor.

[38] An expected outcome as the computation engine is unchanged.

9.8.3 Example 2: Flow Between Two Reservoirs

This example represents the situation where the total head is known at two points on a pipeline, and one wishes to determine the flow rate (or specify a flow rate and solve for a pipe diameter). Like the prior example it is contrived, but follows the same general modeling process.

In this example we will lay out a model building protocol and follow the protocol.[39] The example will first be presented using the OldUI then repeated using the NewUI.

> Using the Moody chart, and the energy equation, estimate the diameter of a cast-iron pipe needed to carry 60^{o}F water at a discharge of 10 cubic-feet per second (cfs) between two reservoirs 2 miles apart. The elevation difference between the water surfaces in the two reservoirs is 20 feet.

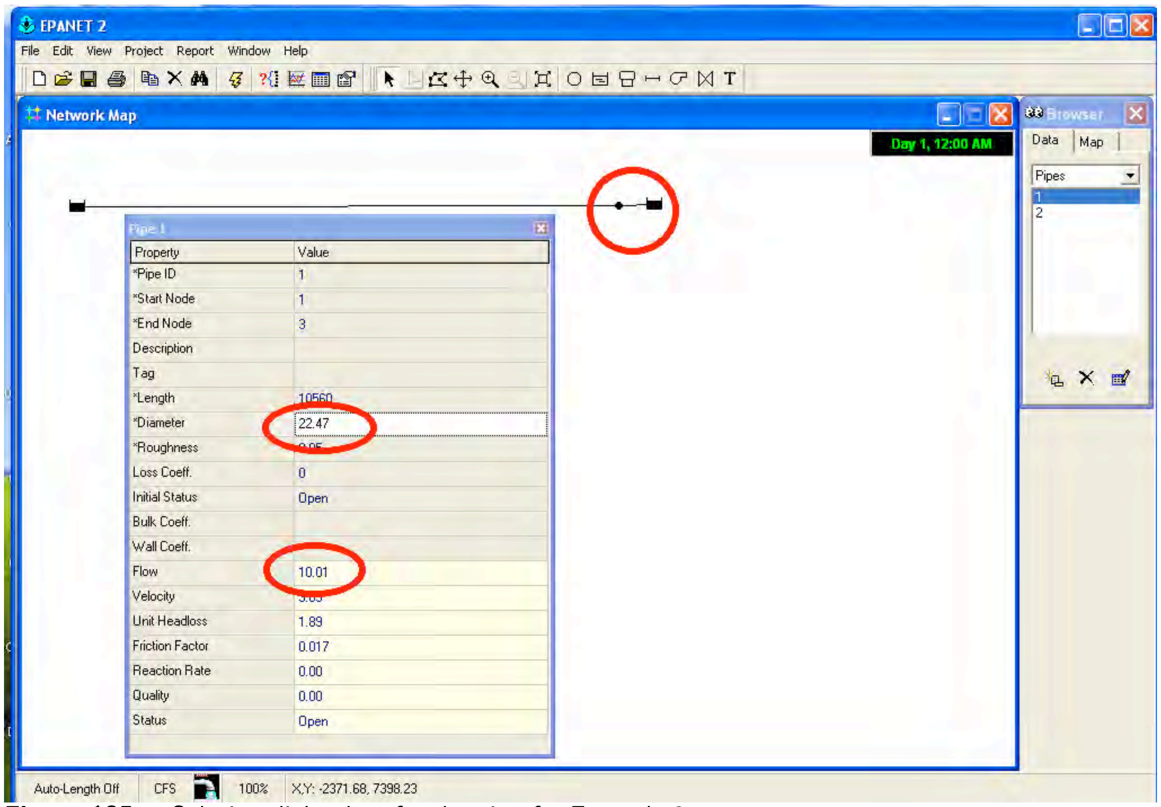
As in the prior example, we will need to specify the pipe roughness terms, then solve by trial-and-error for the diameter required to carry the water at the desired flowrate. Page 31 of the EPA-NET manual suggests that the roughness height for cast iron is 0.85 millifeet.

As before the steps to model the situation are:

1. Start EPA-NET
2. Set hydraulic defaults
3. Select the reservoir tool. Put two reservoirs on the map.
4. Select the node tool, put a node on the map. **EPA NET needs one node!**
5. Select the link (pipe) tool, connect the two reservoirs to the node. One link is the 2 mile pipe, the other is a short large diameter pipe (negligible head loss).
6. Set the total head each reservoir.
7. Set the pipe length and roughness height in the 2 mile pipe.
8. Set the simulation duration to 1:00 (same as the hydraulic time step default value).
9. Guess a diameter.
10. Save the input file.
11. Run the program. Query the pipe and find the computed flow. If the flow is too large reduce the pipe diameter, if too small increase the pipe diameter. Stop when within a few percent of the desired flow rate. Use commercially available diameters in the trial-and-error process, so exact match is not anticipated.

[39]The protocol herein is changed from an earlier version to reflect the need to specify a simulation duration.

Figure 135 is a screen capture after the model is built and some trial-and-error diameter selection. Of importance is the node and the "short pipe" that connects the second reservoir. By changing the diameter (inches) in the dialog box and re-running the program we can find a solution (diameter) that produces 10 cfs in the system for the given elevation differences.



Property	Value
*Pipe ID	1
*Start Node	1
*End Node	3
Description	
Tag	
*Length	10560
*Diameter	22.47
*Roughness	0.05
Loss Coeff.	0
Initial Status	Open
Bulk Coeff.	
Wall Coeff.	
Flow	10.01
Velocity	3.63
Unit Headloss	1.89
Friction Factor	0.017
Reaction Rate	0.00
Quality	0.00
Status	Open

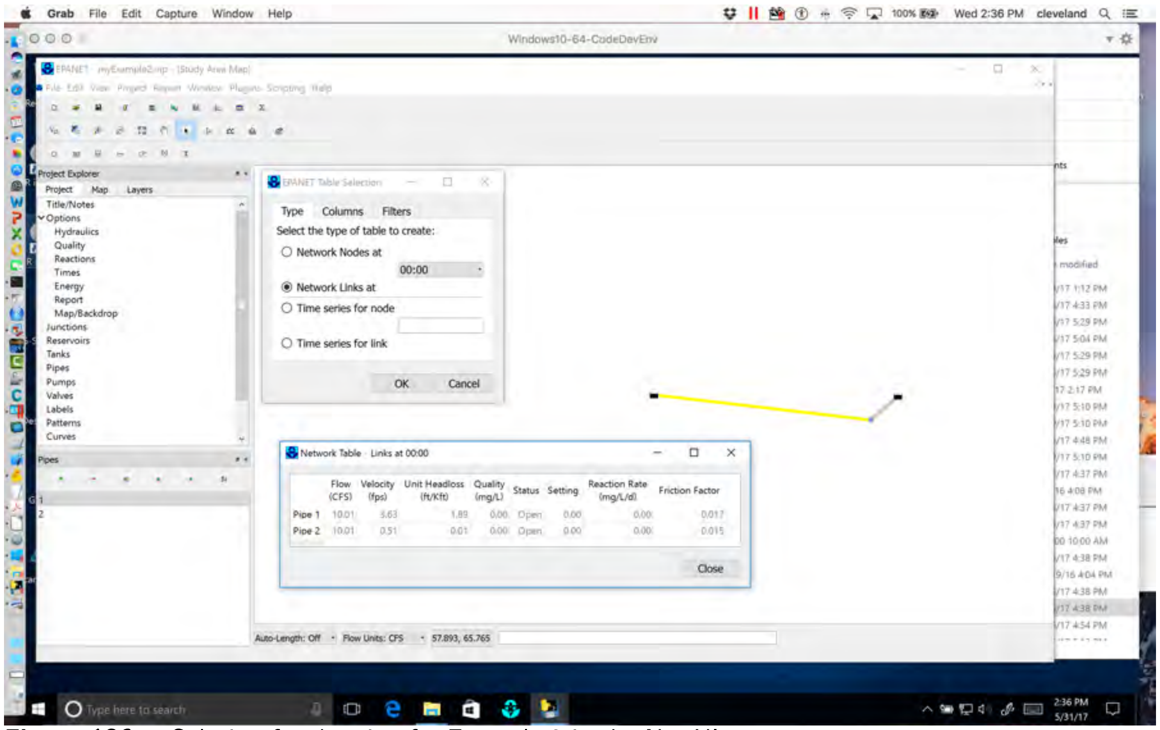
Figure 135. Solution dialog box for the pipe for Example 2.

We would conclude from this use of EPA-NET that a 22.45 inch ID cast iron pipe would convey 10 cfs between the two reservoirs.

The same problem built entirely in the NewUI is displayed in Figure 136. In creating the simulation I built the model exactly as done in the OldUI, and started with a 24-inch pipe as my first guess (result not shown). Then changed to the 22.45 inch pipe to obtain the identical solution as shown in Figure 135.[40]

[40]The resulting pipe diameter is to illustrate the use of the program. The user would surely use a commercially available pipe ID and insert those values. The important point here is that the two interfaces produce the same kind of results.

As a further test of the NewUI, the file built for the problem was then imported into the OldUI and functioned identically. Thus a conclusion of from this example is that the `.INP` files from NewUI are compatible with the OldUI. The reverse direction is also true with the caveat of explicit specification of simulation duration.



Network Table - Links at 00:00

	Flow (CFS)	Velocity (fps)	Unit Headloss (ft/Kft)	Quality (mg/L)	Status	Setting	Reaction Rate (mg/L/d)	Friction Factor
Pipe 1	10.01	5.63	1.89	0.00	Open	0.00	0.00	0.017
Pipe 2	10.01	0.51	0.01	0.00	Open	0.00	0.00	0.015

Auto-Length: Off · Flow Units: CFS · 57.893, 65.765

Figure 136. Solution for the pipe for Example 2 in the NewUI..

9.8.4 Example 3: Three-Reservoir-Problem

This example repeats another classical problem, but introduces the concept of a basemap (image) to help draw the network. First the problem statement

> Reservoirs A, B, and C are connected as shown[41] in Figure 137. The water elevations in reservoirs A, B, and C are 100 m, 80 m, and 60 m. The three pipes connecting the reservoirs meet at junction J, with pipe AJ being 900 m long, BJ being 800 m long, and CJ being 700 m long. The diameters of all the pipes are 850 mm. If all the pipes are ductile iron, and the water temperature is 293^{o}K, find the direction and magnitude of flow in each pipe.

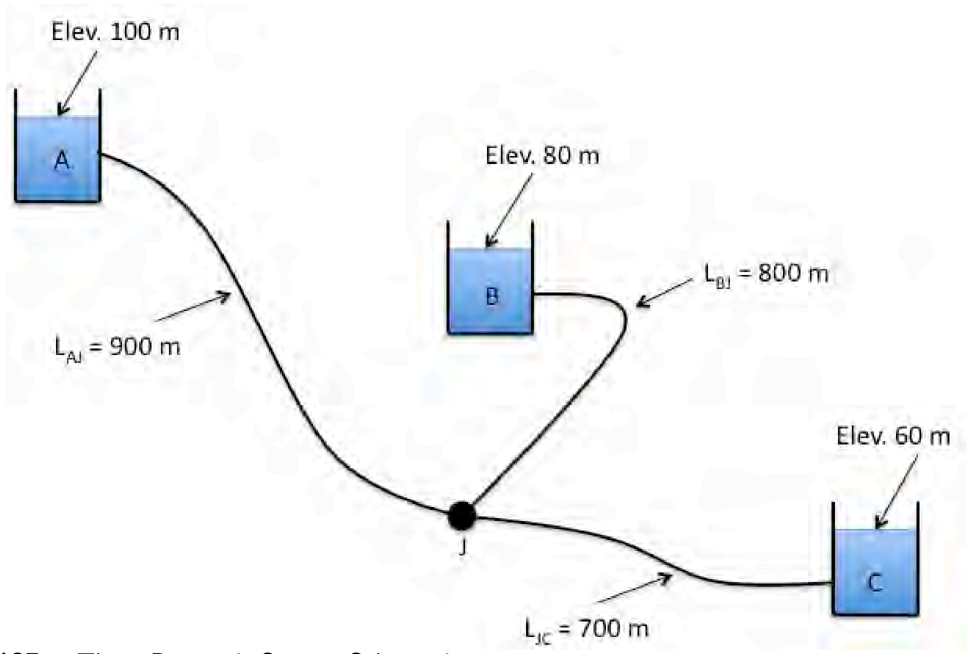


Figure 137. Three-Reservoir System Schematic.

Here we will present the problem worked using OldUI, then repeat the example in the NewUI.

Here we will first convert the image into a bitmap (.bmp) file so EPA-NET can import the background image and we can use it to help draw the network. The remainder of the problem is reasonably simple and is an extension of the previous problem. Bear in mind that the `.BMP` file is simply an image and not geo-referenced – so the example was worked as entirely separate models.

The steps to model the situation are:

1. Convert the image into a bitmap, place the bitmap into a directory where the model input file will be stored.

[41]This problem is identical to Chin Problem 2.30, Pg. 92

2. Start EPA-NET
3. Set defaults
4. Import the background.
5. Select the reservoir tool. Put three reservoirs on the map.
6. Select the node tool, put the node on the map.
7. Select the link (pipe) tool, connect the three reservoirs to the node.
8. Set the total head each reservoir.
9. Set the pipe length, roughness height, and diameter in each pipe.
10. Set the simulation duration to equal the hydraulic time step (1:00 – this setting is to make the file compatible with the NewUI)
11. Save the input file.
12. Run the program.

Figure 138 is the result of the above steps. In this case the default units were changed to LPS (liters per second). The roughness height is about 0.26 millimeters (if converted from the 0.85 millifeet unit).

Figure 139 is the result of the above steps applied using the NewUI. In this example, the image is loaded into the interface, then the network is built as an overlay. The network most likely assumes whatever coordinate reference system (CRS) that the image uses. The computed results are identical to those in the OldUI.

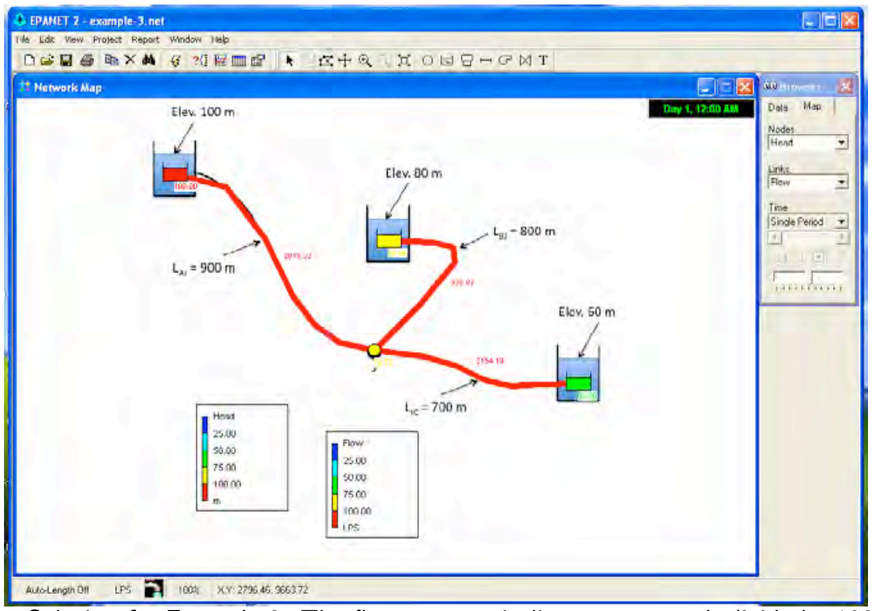
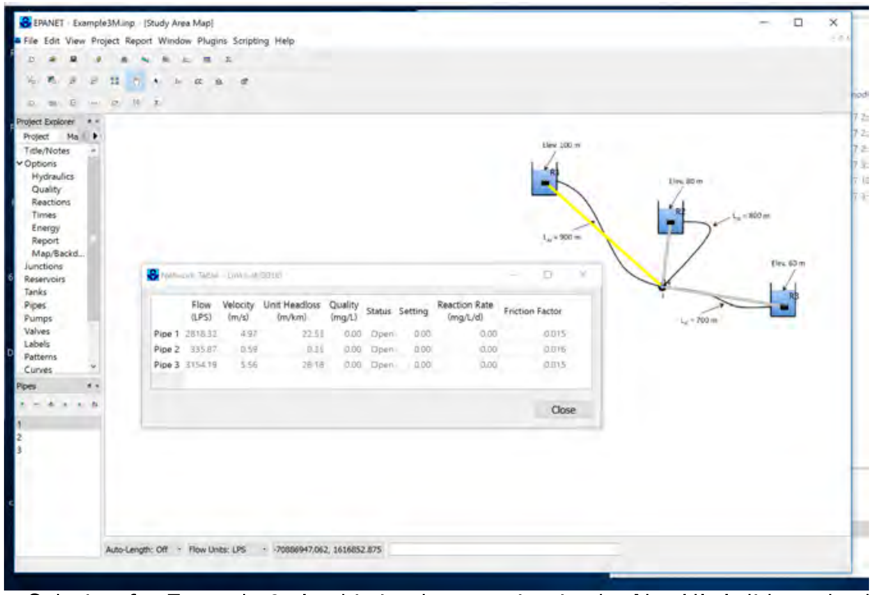


Figure 138. Solution for Example 3. The flowrates are in liters-per-second, divide by 1000 to obtain cubic-meters-per-second..



	Flow (LPS)	Velocity (m/s)	Unit Headloss (m/km)	Quality (mg/L)	Status	Setting	Reaction Rate (mg/L/d)	Friction Factor
Pipe 1	2818.32	4.97	22.51	0.00	Open	0.00	0.00	0.015
Pipe 2	335.87	0.59	0.35	0.00	Open	0.00	0.00	0.016
Pipe 3	3154.19	5.56	28.18	0.00	Open	0.00	0.00	0.015

Figure 139. Solution for Example 3. In this implementation in the NewUI, I did not bother to follow the pipe paths..

9.8.5 Example 4: A Simple Network

Expanding the examples, we will next consider a looped network. As before we will use a prior exercise as the motivating example.

> The water-supply network shown in Figure 140 has constant-head elevated storage tanks at A and C, with inflow and outflow at B and D. The network is on flat terrain with node elevations all equal to 50 meters[42]. If all pipes are ductile iron, compute the inflows/outflows to the storage tanks. Assume that flow in all pipes are fully turbulent.

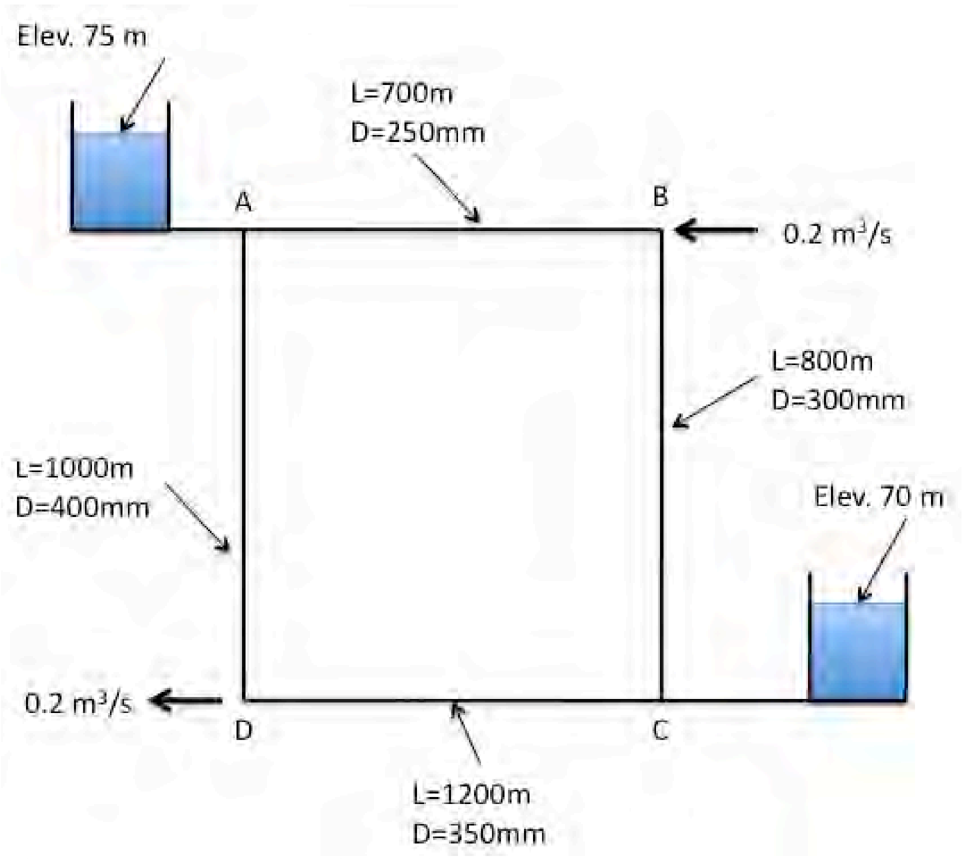


Figure 140. Two-Tank Distribution System Schematic.

As before we will follow the modeling protocol but add demand at the nodes.

The steps to model the situation are:

1. Convert the image into a bitmap, place the bitmap into a directory where the model input file will be stored.
2. Start EPA-NET
3. Set defaults

[42]This problem is similar to Chin Problem 2.31, Pg. 92

4. Import the background.
5. Select the reservoir tool. Put two reservoirs on the map.
6. Select the node tool, put 4 nodes on the map.
7. Select the link (pipe) tool, connect the reservoirs to their nearest nodes. Connect the nodes to each other.
8. Set the total head each reservoir.
9. Set the pipe length, roughness height, and diameter in each pipe. The pipes that connect to the reservoirs should be set as short and large diameter, we want negligible head loss in these pipes so that the reservoir head represents the node heads at these locations.
10. Set the simulation duration to 1:00 hrs (same as the hydraulic time step).
11. Save the input file.
12. Run the program.

In this case the key issues are the units (liters per second) and roughness height (0.26 millimeters). Figure 141 is a screen capture of a completed model built using the OldUI.

As in the prior example the model was entirely rebuilt using the new interface – again because the image is not geo-referenced. The OldUI file would load and run fine, but again an unreferenced image is not easily placed into the interface. So following the same protocol, an entirely new model was built, and run. The results were identical (as anticipated). Figure 142 is a screen capture of a completed model built using the NewUI.

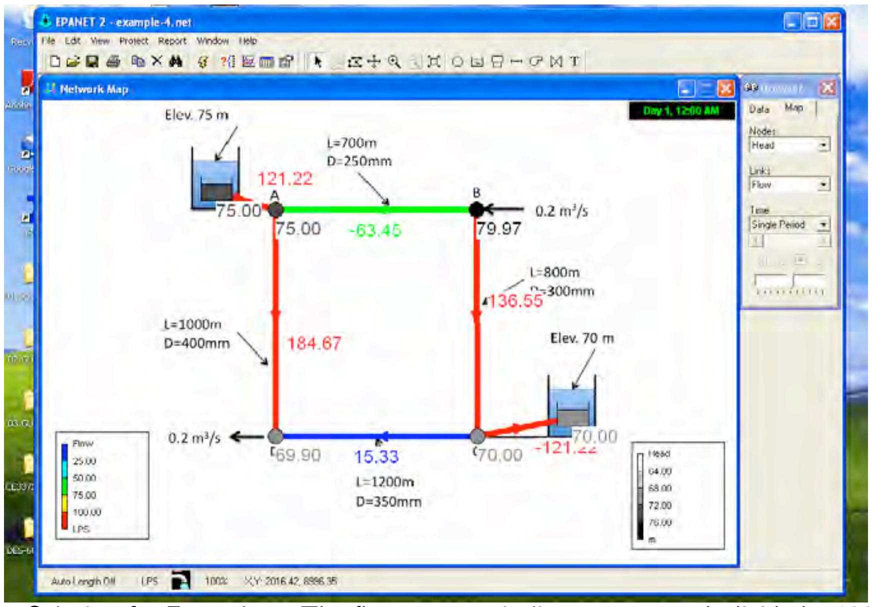
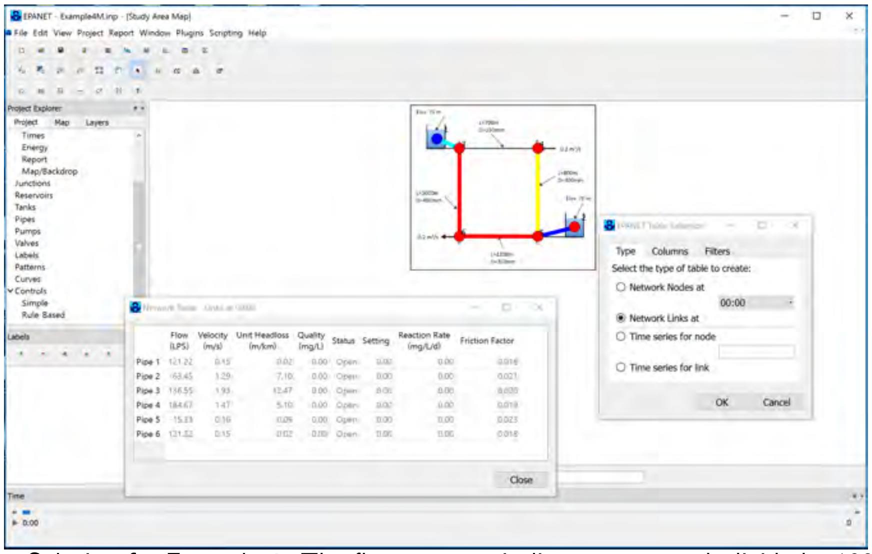


Figure 141. Solution for Example 4. The flowrates are in liters-per-second, divide by 1000 to obtain cubic-meters-per-second..



	Flow (LPS)	Velocity (m/s)	Unit Headloss (m/km)	Quality (mg/L)	Status	Setting	Reaction Rate (mg/L/d)	Friction Factor
Pipe 1	121.22	0.15	0.02	0.00	Open	0.00	0.00	0.018
Pipe 2	-63.45	1.29	7.10	0.00	Open	0.00	0.00	0.021
Pipe 3	136.55	1.93	12.47	0.00	Open	0.00	0.00	0.020
Pipe 4	184.67	1.47	5.10	0.00	Open	0.00	0.00	0.019
Pipe 5	15.33	0.16	0.09	0.00	Open	0.00	0.00	0.023
Pipe 6	121.22	0.15	0.02	0.00	Open	0.00	0.00	0.018

Figure 142. Solution for Example 4. The flowrates are in liters-per-second, divide by 1000 to obtain cubic-meters-per-second. This example used the NewUI. The numerical results are identical to the OldUI implementation..

9.8.6 Example 5: Pumping Water Uphill

The example illustrates how to model a pump in EPA-NET. A pump is a special "link" in EPA-NET. This link causes a negative head loss (adds head) according to a pump curve. In addition to a pump curve there are three other ways to model added head — these are discussed in the user manual and are left for the reader to explore on their own.

Figure 143 is a conceptual model of a pump lifting water through a 100 mm diameter, 100 meter long, ductile iron pipe from a lower to an upper reservoir. The suction side of the pump is a 100 mm diameter, 4-meter long ductile iron pipe. The difference in reservoir free-surface elevations is 10 meters. The pump performance curve is given as

$$h_p = 15 - 0.1Q^2 \tag{98}$$

where the added head is in meters and the flow rate is in liters per second (Lps). The analysis goal is to estimate the flow rate in the system.

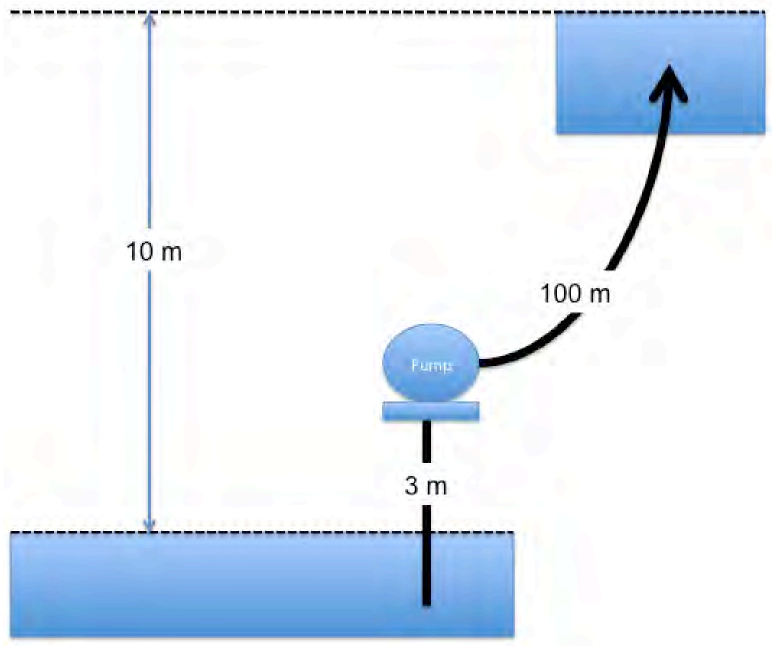


Figure 143. Example 5 conceptual model. The pipes are 100 mm ductile iron..

To model this situation, the engineer follows the modeling protocol already outlined, only adding the special link.

1. Convert the image into a bitmap, place the bitmap into a directory where the model input file will be stored.
2. Start EPA-NET
3. Set defaults (hydraulics = D-W, units = LPS)
4. Import the background.
5. Select the reservoir tool. Put two reservoirs on the map.
6. Select the node tool, put 2 nodes on the map, these represent the suction and discharge side of the pump.
7. Select the link (pipe) tool, connect the reservoirs to their nearest nodes.
8. Select the pump tool.
9. Connect the nodes to each other using the pump link.
10. Set the total head each reservoir.
11. Set the pipe length, roughness height, and diameter in each pipe.
12. On the Data menu, select Curves. Here is where we create the pump curve. This problem gives the curve as an equation, we will need three points to define the curve. Shutoff ($Q = 0$), and simple to compute points make the most sense.
13. Set the simulation duration to 1:00 hours (same as the hydraulic time step).
14. Save the input file.
15. Run the program.

Again as in the other examples, the solution will be presented in the OldUI and the NewUI.

Figure 144 is a screen capture of loading the background image. After the image is loaded, we can then build the hydraulic model. The next step is to place the reservoirs.

Figure 145 is a screen capture of the reservoirs after they have been placed. The upper reservoir will be assigned a total head 10 meters larger than the lower reservoir — a reasonable conceptual model is to use the lower reservoir as the datum.

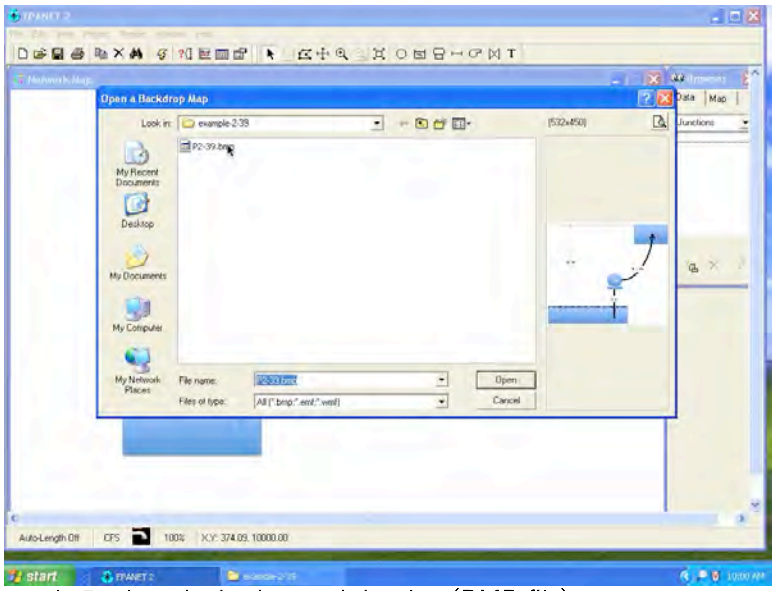


Figure 144. Example 5 select the background drawing (BMP file).

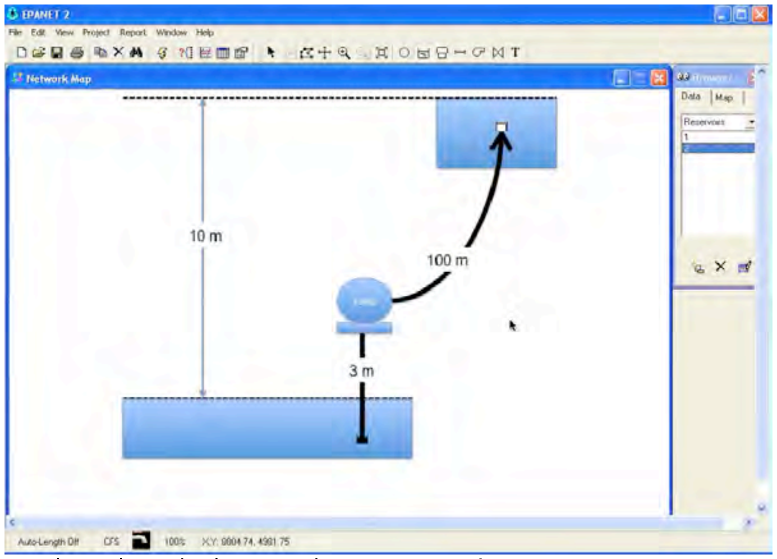


Figure 145. Example 5 place the lower and upper reservoir.

Figure 146 is a screen capture of model just after the pump is added. The next steps are to set the pipe lengths (not shown) and the reservoir elevations (not shown). Finally, the engineer must specify the pump curve.

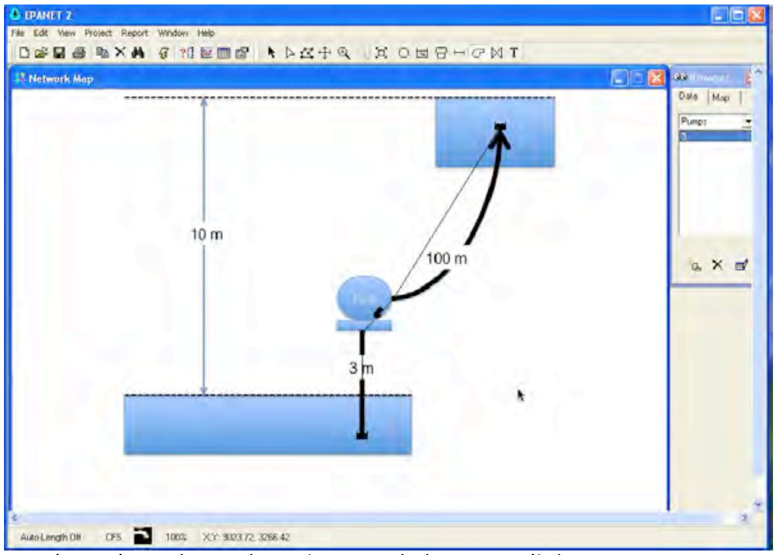
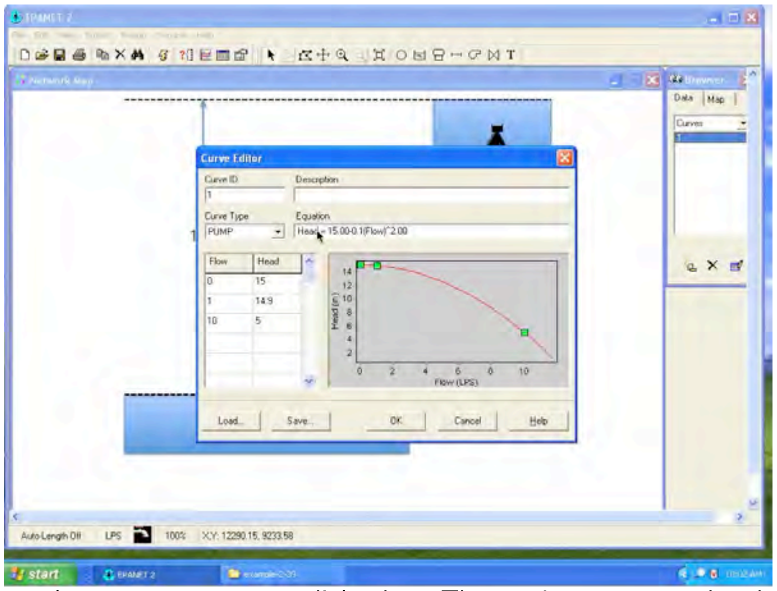

EPANET 2
File Edit View Project Report Window Help
Network Map
10 m
100 m
3 m
Browser
Data Map
Pumps
Auto-Length Off CFS 100% X,Y: 9323.72, 3266.42

Figure 146. Example 5 place the nodes, pipes, and the pump link..


EPANET 2
Network Map
Browser
Data Map
Curves
Curve Editor
Load... Save... OK Cancel Help
Auto-Length Off LPS 100% X,Y: 12290.15, 9233.58
start

Curve ID: 1

Description:

Curve Type: PUMP

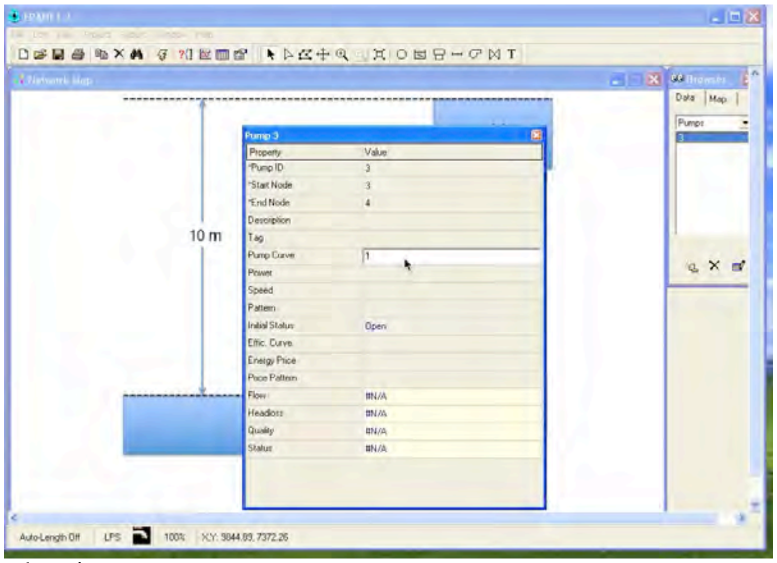
Equation: Head = 15.00-0.1(Flow)^2.00

Flow	Head
0	15
1	14.9
10	5

Figure 147. Example 5 pump curve entry dialog box. Three points are entered and the curve equation is created by the program..

Figure 147 is a screen capture of the pump curve data entry dialog box. Three points on the curve were selected and entered into the tabular entry area on the left of the dialog box, then the curve is created by the program. The equation created by the program is the same as that of the problem – hence we have the anticipated pump curve.

Next the engineer associates the pump curve with the pump as shown in Figure 148.



Pump 3

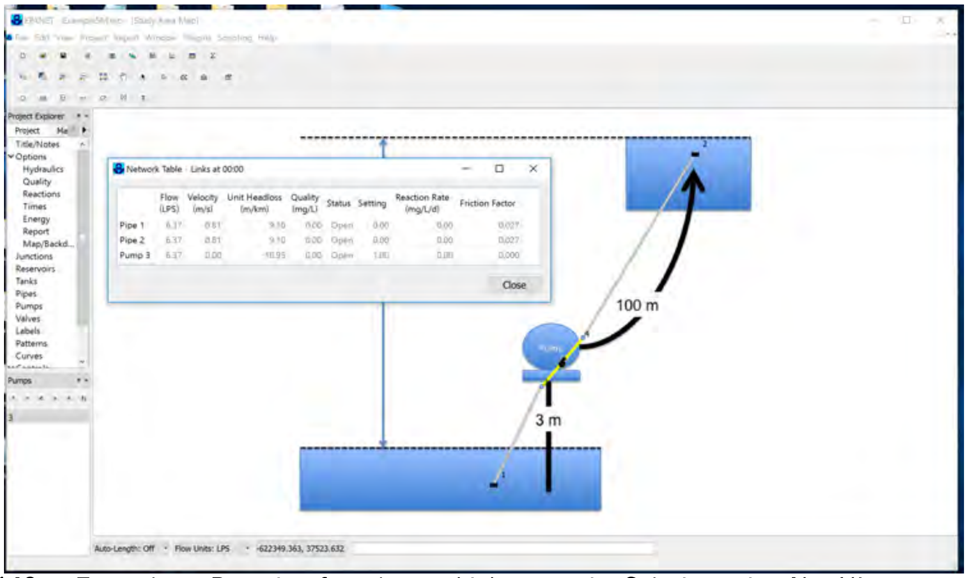
Property	Value
*Pump ID	3
*Start Node	3
*End Node	4
Description	
Tag	
Pump Curve	1
Power	
Speed	
Pattern	
Initial Status	Open
Effic. Curve	
Energy Price	
Price Pattern	
Flow	#N/A
Headloss	#N/A
Quality	#N/A
Status	#N/A

Auto-Length Off LPS 100% X,Y: 9844.89, 7372.26

Figure 148. Setting the pump curve..

Upon completion of this step, the program is run to estimate the flow rate in the system.

The complete example is presented using the NewUI in Figure



Network Table - Links at 00:00

	Flow (LPS)	Velocity (m/s)	Unit Headloss (m/km)	Quality (mg/L)	Status	Setting	Reaction Rate (mg/L/d)	Friction Factor
Pipe 1	6.37	0.81	9.10	0.00	Open	0.00	0.00	0.027
Pipe 2	6.37	0.81	9.10	0.00	Open	0.00	0.00	0.027
Pump 3	6.37	0.00	-10.95	0.00	Open	1.00	0.00	0.000

Auto-Length: Off Flow Units: LPS -622349.363, 37523.632

Figure 149. Example 5. Pumping from low to high reservoir. Solution using NewUI..

9.9 Running from the Command Line

References

Christian, B. and Griffiths, T. (2016). Algorithms to Live By: The Computer Science of Human Decisions Henry Holt and Co.. Kindle Edition.

Cunge, J.A., Holly, F.M., Verwey, A. (1980). Practical Aspects of Computational River Hydraulics. Pittman Publishing Inc. , Boston, MA. pp. 7-50

Zheng, C. and Bennett, G. D. (1995). *Applied Contaminant Transport Modeling.* Van Nostrand Reinhold.

Press, W.H., Flannery, B.P., Teukolsky, S.A., Vettering, W. T. 1986, Numerical Recipes:*The Art of Scientific Computing*. Cambridge University Press, London. 818p.

Gill, P.E., Murray, W, Wright, M. H., 1981. Practical Optimization. Academic Press, San Diego. 401p.

Wikipedia discussion of Newton's method. `http://en.wikipedia.org/wiki/Newton's_method`.

Wikipedia discussion of matrix inversion. `http://en.wikipedia.org/wiki/Invertible_matrix`.

Chin, D. A. (2006). *Water-Resources Engineering.* Prentice Hall.

Haman YM, Brameller A. (1971) Hybrid method for the solution of piping networks. Proc IEEE 1971;118(11):1607-12.