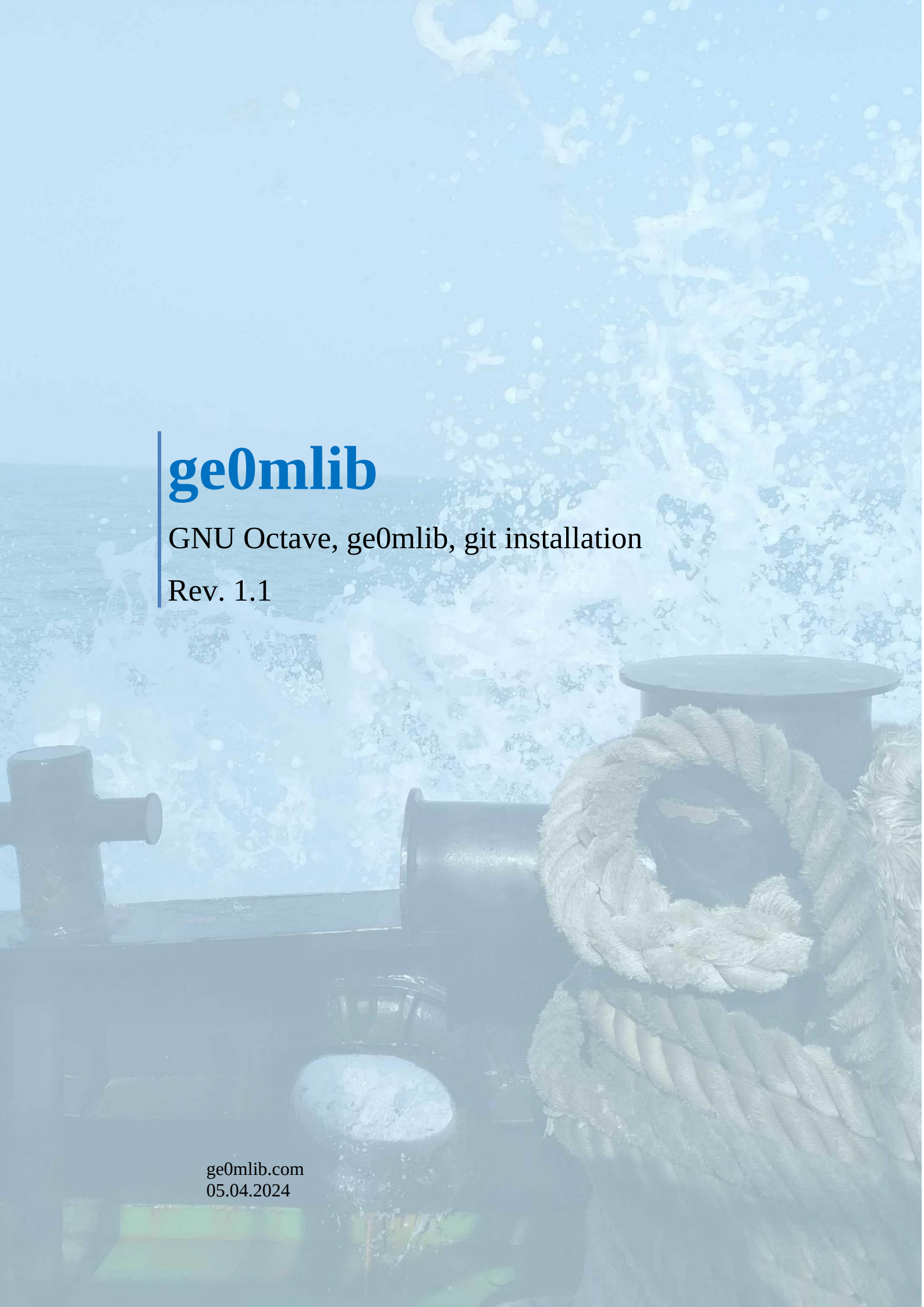




ge0mlib

GNU Octave, ge0mlib, git installation

Rev. 1.1

ge0mlib.com
05.04.2024

Contents

Introduction.....	4
1 GNU Octave.....	5
1.1 GNU Octave installation	5
1.2 Additional packages (libraries) installation.....	6
2 ge0mllib installation (MatLab, GNU Octave).....	10
3 Git.....	12
3.1 Purposes of using Git and Github.....	12
3.2 Git installation	12
3.3 Creating a git project and downloading ge0mllib from the repository.....	15
3.4 Navigating history in Git.....	16
3.5 Launching the GUI.....	18
4 Structure of the script and script commands	19
Conclusion.....	22

Revision History

Revision	Date	Description of Changes
1.0	2023-12-26	First issue document// Ivan D.
1.1	2024-04-05	Edited the text // Ivan D., Margarita B.

Figures list

Figure 1.1 GNU Octave installation.....	5
Figure 1.2 GNU Octave command window interface	5
Figure 1.3 GNU Octave m-file editor interface.....	6
Figure 1.4 Contents of folders with libraries for GNU Octave	8
Figure 1.5 Menu for specifying the library paths in GNU Octave settings.....	9
Figure 1.6 An example of the interface for downloading a library from the Internet on the GNU Octave website.....	9
Figure 2.1 Structure of the ge0mlib library catalog	10
Figure 2.2 MatLab window with paths for the ge0mlib library defined	11
Figure 2.3 GNU Octave window with paths for the ge0mlib library defined	11
Figure 3.1 Git installation.....	14
Figure 3.2 Executing commands for initial git settings.....	15
Figure 3.3 Opening Git Bash from the Windows context menu	15
Figure 3.4 The git window and the contents of the Functions folder after creating a repository	16
Figure 3.5 The git window with the commit list	16
Figure 3.6 Files in the Functions folder as a result of applying the "checkout" command	17
Figure 3.7 A git error telling you to save your last changes as commit	17
Figure 3.8 Opening Git GUI via Windows context menu.....	18
Figure 3.9 Git GUI interface	18
Figure 4.1 MagToolR01 script code example	20
Figure 4.2 MatLab window with executed command sequence for MagToolR01	21

Introduction

The ge0mlib library is designed for marine engineering geophysics data processing. The processing functions are developed in MatLab 2018b (<https://mathworks.com/>) and adapted for the free software GNU Octave 8.4.0 (<https://www.octave.org>). Project website: <https://ge0mlib.com/>. The library is available on Git Hub: <https://github.com/ge0mlib/ge0mlib>.

This document is written for a non-programmer Windows user who needs to perform data processing using a ready-made script written for MatLab or Octave. To complete the task, there are:

- 1) m-file with a script for data processing, previously written for the MatLab and/or GNU Octave programming language (the development date can be found at the end of the script text);
- 2) a list of additional MatLab and/or GNU Octave libraries that must be installed to run the script (written in the comments at the beginning of the script text);
- 3) a description of the script commands (a description of each command should be given in the text of the script, and examples of working command sequences should also be given there).

The purpose of this document is to provide a quick installation of the software to begin processing data. The document describes the following steps:

- download and install GNU Octave (Windows version);
- download and install additional packages (libraries) for GNU Octave (Windows);
- download and install the ge0mlib library;
- download and install Git (Windows version);
- restoring a “snapshot” (commit) of the ge0mlib library to the date the script was developed using Git (commit means the copy of the folder with files containing the library functions at specific times);
- work with script commands (description of the principles of constructing script commands and information contained in the script body-text).

The text of the document includes a little more information than necessary. However, this information makes you feel more comfortable working with GNU Octave, ge0mlib and Git. Such optional (*but it's not for sure*) text to read is [highlighted](#) in blue.

1 GNU Octave

1.1 GNU Octave installation

Octave distribution is downloaded from <https://octave.org/download>. At the moment of writing this text, it is the file octave-8.4.0-w64-installer.exe (for Win64).

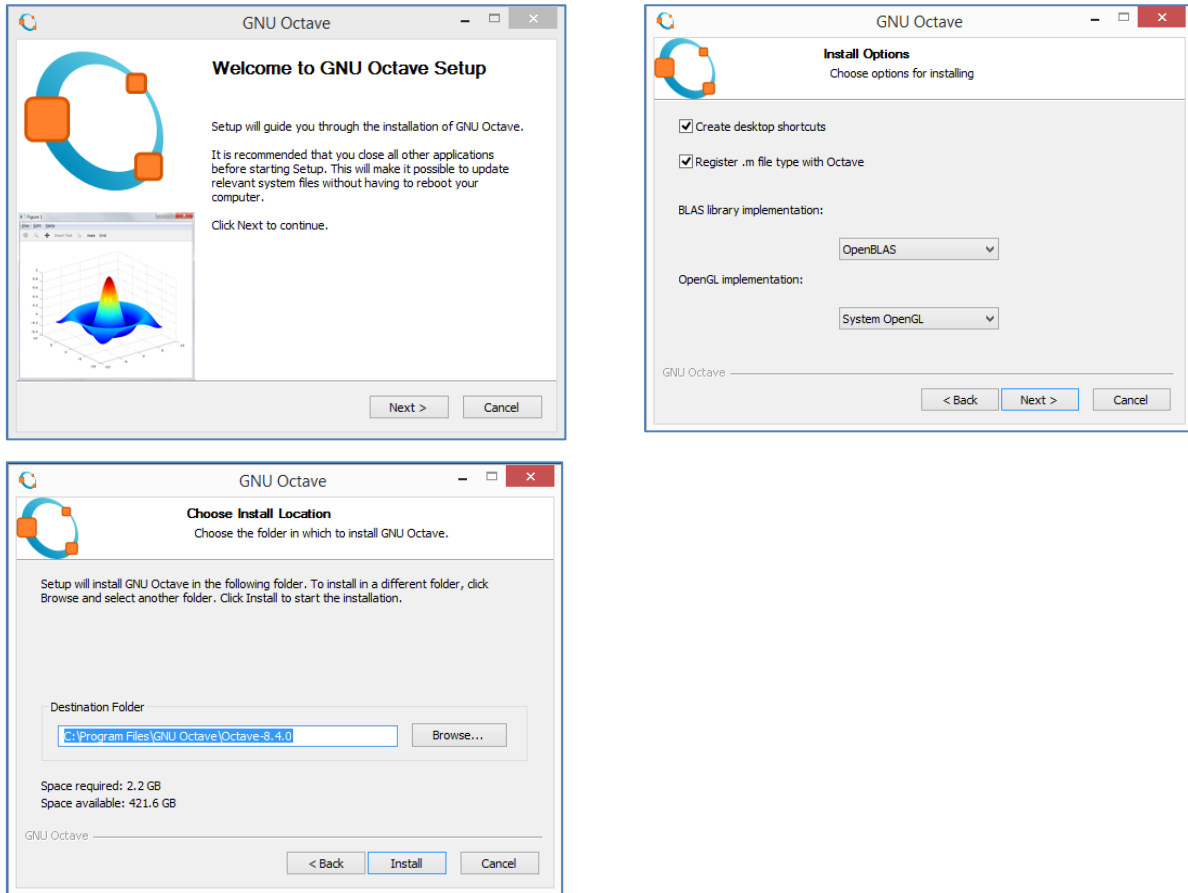


Figure 1.1 GNU Octave installation

The Octave command window after installation with the tabs moved for convenience is shown in Figure 1.2. Pay attention to the Variable Editor, Command History and Workspace tabs.

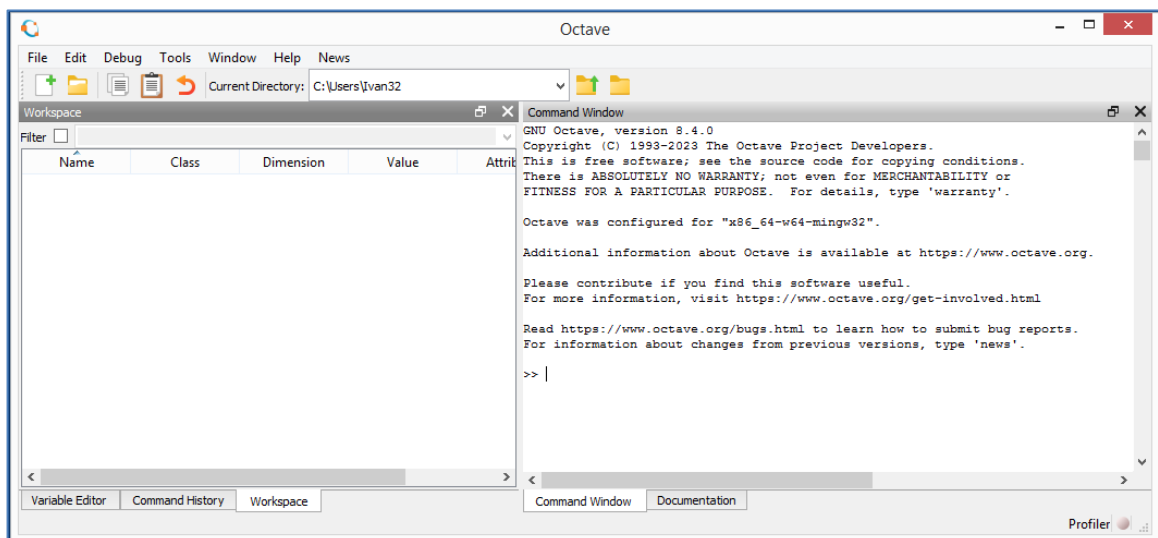


Figure 1.2 GNU Octave command window interface

The m-file debugger window is shown in *Figure 1.3*. It may be convenient to enter a sequence of script commands in the debugger window, then save this file in the Script folder (a description of the folder is given below in section 2, when describing the installation of geOmlib). After this, instead of entering a sequence of script commands into the command window, you can enter only the name of the file generated in the debugger (this file itself, in fact, will be a simple script).

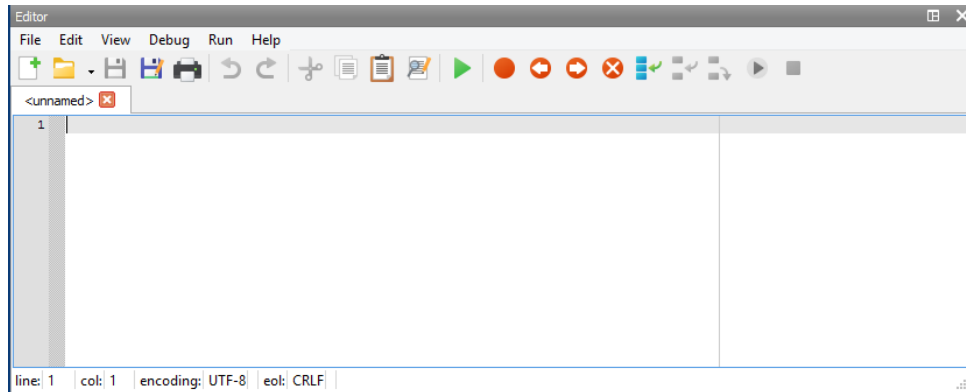


Figure 1.3 GNU Octave m-file editor interface

1.2 Additional packages (libraries) installation

Additional libraries in Octave are usually called “packages”, and in MatLab – “toolbox”. The link to the page with the new version of the official packages for Octave: <https://gnu-octave.github.io/packages/>.

The link to the page with old versions of packages: <https://octave.sourceforge.io/packages.php>

There is a list of packages that can be used for solving marine geophysics problems. The selection is subjective and intended primarily to describe a general idea of the packages included in GNU Octave.

control – Computer-Aided Control System Design (CACSD) Tools for GNU Octave, based on the SLICOT Library.

data-smoothing – Algorithms for smoothing noisy data.

divand – Performs an n-dimensional variational analysis (interpolation) of arbitrarily located observations.

fda – Functional Data Analysis.

fileio – I/O function for files holding structured data, such as JSON and XML files.

fuzzy-logic-toolkit – A mostly MATLAB-compatible fuzzy logic toolkit for Octave.

geographiclib – Native Octave/MATLAB implementations of a subset of the C++ library,

GeographicLib. Key components of this toolbox are: (a) Geodesics, direct, inverse, area calculations; (b) Projections, transverse Mercator, polar stereographic, etc.; (c) Grid systems, UTM, UPS, MGRS; (d) Geoid lookup, egm84, egm96, egm2008 geoids supported; (e) Geometric transformations, geocentric, local cartesian; (f) Great ellipse, direct, inverse, area calculations.

geometry – Library for extending MatGeom functionality.

image – Functions for image processing, feature extraction, image statistics, spatial and geometric transformations, morphological operations, linear filtering, and much more.

io – Input/Output in external formats.

linear-algebra – Additional linear algebra code, including matrix functions.

mapping – Simple mapping and GIS .shp .dxf and raster file functions.

matgeom – Geometry toolbox for 2D/3D geometric computing.

miscellaneous – Miscellaneous tools that don't fit somewhere else.

mvn – Multivariate normal distribution clustering and utility functions.

nan – A statistics and machine learning toolbox for data with and w/o missing values.

octproj – This package allows to call functions of PROJ library for cartographic projections and CRS transformations.

optim – Non-linear optimization toolkit.

optiminterp – An optimal interpolation toolbox providing functions to perform a n-dimensional optimal interpolations of arbitrarily distributed data points.

packajoozle – Enhanced package manager for GNU Octave.

pkg-octave-doc – This package provides functions for generating HTML pages that contain the help texts of the functions of an octave package. The package is designed to work with installed packages and use their INDEX file for creating the respective functions' HTML pages. The default layout is based on bootstrap 5 and it follows the design of the Octave Packages GitHub page.

quaternion – Quaternion package for GNU Octave, includes a quaternion class with overloaded operators.

signal – Signal processing tools, including filtering, windowing and display functions.

sparsersb – Interface to the librsb package implementing the RSB sparse matrix format for fast shared-memory sparse matrix computations.

splines – Additional spline functions

sqlite – Basic Octave implementation of the sqlite toolkit

statistics – The Statistics package for GNU Octave.

statistics-resampling – Estimate bias, uncertainty (standard errors and confidence intervals) and test hypotheses (p-values) using resampling methods. (Note that versions of this package <= 5.4.3 are named the statistics-bootstrap package).

stk – The STK is a (not so) Small Toolbox for Kriging. Its primary focus is on the interpolation/regression technique known as kriging, which is very closely related to Splines and Radial Basis Functions, and can be interpreted as a non-parametric Bayesian method using a Gaussian Process (GP) prior. The STK also provides tools for the sequential and non-sequential design of experiments. Even though it is, currently, mostly geared towards the Design and Analysis of Computer Experiments (DACE), the STK can be useful for other applications areas (such as Geostatistics, Machine Learning, Non-parametric Regression, etc.).

strings – Additional functions for manipulation and analysis of strings.

struct – Additional structure manipulation functions.

In order to use the package functions in Octave, the path to the folder containing files with functions must be registered in Octave settings. There are two folders with packages in the Octave root folder (*Figure 1.4*):

- 1) c:\Program Files\GNU Octave\Octave-8.4.0\mingw64\share\octave\8.4.0\m\ – the folder contains "system-integrated" packages such as signal and statistics;
- 2) c:\Program Files\GNU Octave\Octave-8.4.0\mingw64\share\octave\packages\ – the folder contains non-integrated packages, the paths for which were not initially registered in Octave settings.

With a high probability the required package is located in these folders. So, you won't need to download it from the Internet.

m					packages				
2014_06_HTML					GeOMLib				
\\GNU Octave\Octave-8.4.0\mingw64\share\octave\8.4.0\m\.*					\\GNU Octave\Octave-8.4.0\mingw64\share\octave\packages\.*				
Name	Ext	Size	Date	Attr	Name	Ext	Size	Date	Attr
[.]		<DIR>	24.12.2023 21:55	----	[.]		<DIR>	24.12.2023 21:55	----
[@ftp]		<DIR>	24.12.2023 21:55	----	[audio-2.0.8]		<DIR>	24.12.2023 21:54	----
[+containers]		<DIR>	24.12.2023 21:55	----	[biosig-2.5.2]		<DIR>	24.12.2023 21:55	----
[+matlab]		<DIR>	24.12.2023 21:55	----	[cfitsio-0.0.5]		<DIR>	24.12.2023 21:54	----
[audio]		<DIR>	24.12.2023 21:55	----	[communications-1.2.6]		<DIR>	24.12.2023 21:55	----
[deprecated]		<DIR>	24.12.2023 21:55	----	[control-3.6.1]		<DIR>	24.12.2023 21:54	----
[elfun]		<DIR>	24.12.2023 21:55	----	[database-2.4.4]		<DIR>	24.12.2023 21:54	----
[general]		<DIR>	24.12.2023 21:55	----	[dataframe-1.2.0]		<DIR>	24.12.2023 21:55	----
[geometry]		<DIR>	24.12.2023 21:55	----	[data-smoothing-1.3.0]		<DIR>	24.12.2023 21:55	----
[gui]		<DIR>	24.12.2023 21:55	----	[dicom-0.5.1]		<DIR>	24.12.2023 21:55	----
[help]		<DIR>	24.12.2023 21:55	----	[financial-0.5.3]		<DIR>	24.12.2023 21:55	----
[image]		<DIR>	24.12.2023 21:55	----	[fuzzy-logic-toolkit-0.4.6]		<DIR>	24.12.2023 21:54	----
[io]		<DIR>	24.12.2023 21:55	----	[ga-0.10.3]		<DIR>	24.12.2023 21:55	----
[java]		<DIR>	24.12.2023 21:55	----	[general-2.1.3]		<DIR>	24.12.2023 21:54	----
[legacy]		<DIR>	24.12.2023 21:55	----	[generate_html-0.3.3]		<DIR>	24.12.2023 21:53	----
[linear-algebra]		<DIR>	24.12.2023 21:55	----	[geometry-4.0.0]		<DIR>	24.12.2023 21:53	----
[miscellaneous]		<DIR>	24.12.2023 21:55	----	[gsl-2.1.1]		<DIR>	24.12.2023 21:54	----
[ode]		<DIR>	24.12.2023 21:55	----	[image-2.14.0]		<DIR>	24.12.2023 21:53	----
[optimization]		<DIR>	24.12.2023 21:55	----	[instrument-control-0.9.1]		<DIR>	24.12.2023 21:53	----
[path]		<DIR>	24.12.2023 21:55	----	[interval-3.2.1]		<DIR>	24.12.2023 21:54	----
[pkg]		<DIR>	24.12.2023 21:55	----	[io-2.6.4]		<DIR>	24.12.2023 21:54	----
[plot]		<DIR>	24.12.2023 21:55	----	[linear-algebra-2.2.3]		<DIR>	24.12.2023 21:55	----
[polynomial]		<DIR>	24.12.2023 21:55	----	[lssa-0.1.4]		<DIR>	24.12.2023 21:54	----
[prefs]		<DIR>	24.12.2023 21:55	----	[lftat-2.3.1]		<DIR>	24.12.2023 21:55	----
[profiler]		<DIR>	24.12.2023 21:55	----	[mapping-1.4.2]		<DIR>	24.12.2023 21:54	----
[set]		<DIR>	24.12.2023 21:55	----	[matgeom-1.2.3]		<DIR>	24.12.2023 21:55	----
[signal]		<DIR>	24.12.2023 21:55	----	[miscellaneous-1.3.0]		<DIR>	24.12.2023 21:54	----
[sparse]		<DIR>	24.12.2023 21:55	----	[mqtt-0.0.4]		<DIR>	24.12.2023 21:54	----
[specfun]		<DIR>	24.12.2023 21:55	----	[nan-3.7.0]		<DIR>	24.12.2023 21:55	----
[special-matrix]		<DIR>	24.12.2023 21:55	----	[netcdf-1.0.17]		<DIR>	24.12.2023 21:54	----
[startup]		<DIR>	24.12.2023 21:55	----	[nurbs-1.4.3]		<DIR>	24.12.2023 21:54	----
[statistics]		<DIR>	24.12.2023 21:55	----	[ocs-0.1.5]		<DIR>	24.12.2023 21:54	----
[strings]		<DIR>	24.12.2023 21:55	----	[octproj-3.0.2]		<DIR>	24.12.2023 21:54	----
[testfun]		<DIR>	24.12.2023 21:55	----	[optim-1.6.2]		<DIR>	24.12.2023 21:54	----
[time]		<DIR>	24.12.2023 21:55	----	[optiminterp-0.3.7]		<DIR>	24.12.2023 21:53	----
[web]		<DIR>	24.12.2023 21:55	----	[quaternion-2.4.0]		<DIR>	24.12.2023 21:53	----
					[queueing-1.2.7]		<DIR>	24.12.2023 21:54	----
					[signal-1.4.5]		<DIR>	24.12.2023 21:54	----
					[sockets-1.4.1]		<DIR>	24.12.2023 21:55	----
					[sparsersb-1.0.9]		<DIR>	24.12.2023 21:54	----
					[splines-1.3.5]		<DIR>	24.12.2023 21:54	----
					[statistics-1.6.0]		<DIR>	24.12.2023 21:54	----
					[stk-2.8.1]		<DIR>	24.12.2023 21:54	----
					[strings-1.3.1]		<DIR>	24.12.2023 21:55	----
					[struct-1.0.18]		<DIR>	24.12.2023 21:53	----
					[symbolic-3.1.1]		<DIR>	24.12.2023 21:53	----
					[tisean-0.2.3]		<DIR>	24.12.2023 21:55	----
					[tsa-4.6.3]		<DIR>	24.12.2023 21:54	----
					[video-2.1.1]		<DIR>	24.12.2023 21:53	----
					[windows-1.6.4]		<DIR>	24.12.2023 21:55	----
					[zeromq-1.5.6]		<DIR>	24.12.2023 21:54	----

Figure 1.4 Contents of folders with libraries for GNU Octave

You can use the Path Browser to see which paths are already registered in Octave settings (the paths to all packages in the “m\” folder are registered). Using the “Add Folder” button, you can add paths to additional packages, after which you need to click on the “Save” button to save the list of paths, and it will be automatically loaded on subsequent Octave launches (*Figure 1.5*).

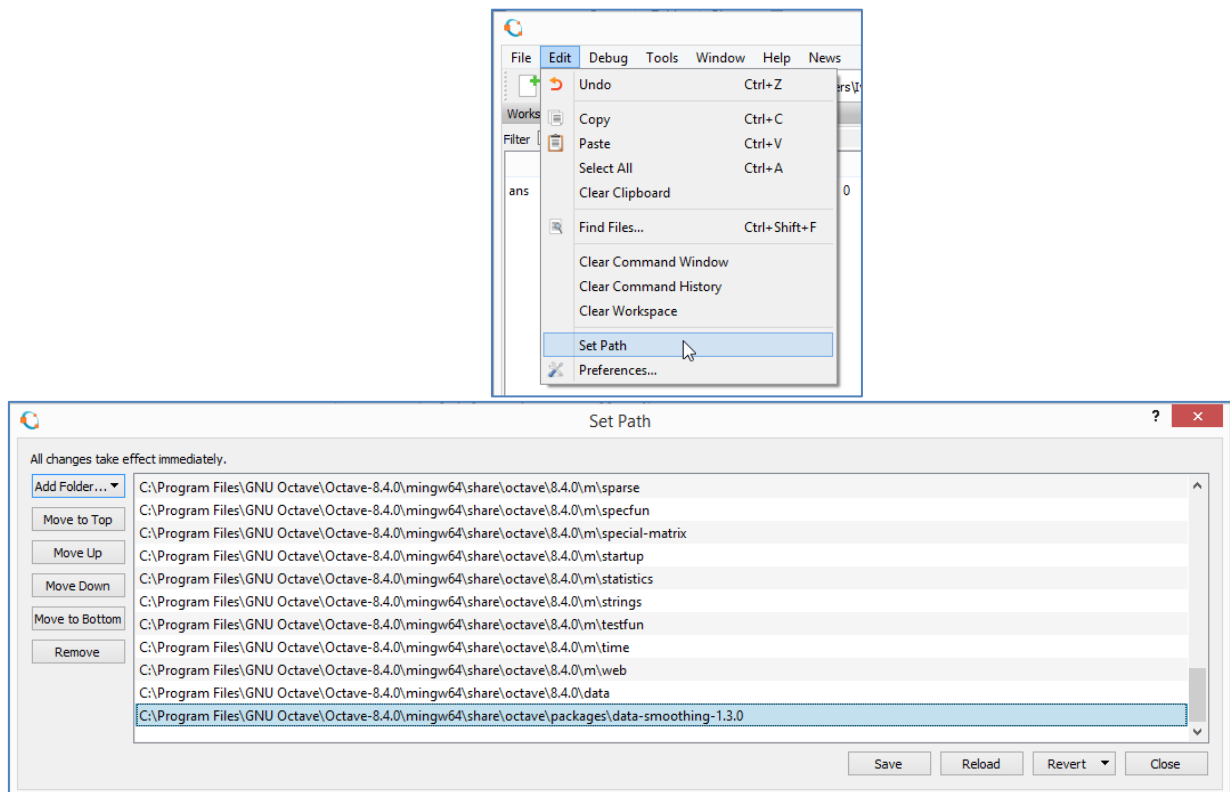


Figure 1.5 Menu for specifying the library paths in GNU Octave settings

If the required package is not located in the folders mentioned above, you need to find it on the Internet. For example, when we open to the data-smooth package page, we can see the window shown in *Figure 1.6*. To install the data-smooth package directly from the Internet, it is needed to copy the “pkg install” command and path into the Octave command window and then run it. You can also download the package and use the “pkg install” command (with the path specified) to install it offline. You may need to additionally register the path to the package after installation.

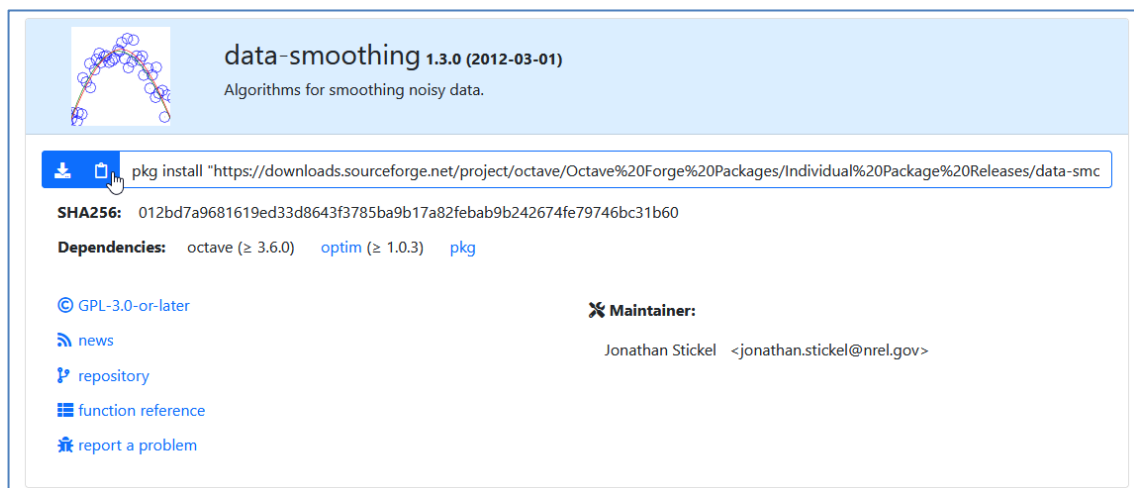


Figure 1.6 An example of the interface for downloading a library from the Internet on the GNU Octave website

2 ge0mlib installation (MatLab, GNU Octave)

You can download the ge0mlib library from the link <https://ge0mlib.com/g/ge0mlib.zip>, and then unpack the archive to a convenient location. The archive contains the folders shown in *Figure 2.1*.

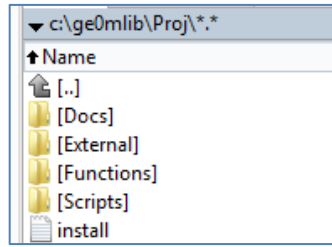


Figure 2.1 Structure of the ge0mlib library catalog

The purpose of the folders is as follows:

- Docs – contains documentation on working with the library (pdf format), as well as examples of scripts (m-files and descriptions in pdf format; video materials are presented on the site, but are not included in the archive with the library);
- External – folder for additional libraries that are used or planned to be used with ge0mlib. Currently it is m_map, mksqlite-2.11-win64, n_vector;
- Functions – functions of the ge0mlib library (the contents of the folder are available on Git Hub: <https://github.com/ge0mlib/ge0mlib>);
- Scripts – an empty folder for scripts, for local use. Here you need to put the m-file with a script that needs to be executed to process the data. If you later write your own script or function, you can also place them in this folder;
- install.txt – a file with commands for registering paths (instead of using Path Browser).

To install ge0mlib, you can register the path to the m-files manually using the Path Browser or write the path to the ge0mlib root folder in the first line of install.txt, then copy the text from the file into the command window and press Enter (to execute the entered commands). An example of such setting paths for the root folder 'c:\ge0mlib\Proj' is shown in the figures below. *When executing commands, the presence of the corresponding subfolders in External, as well as the Functions and Scripts folders, will be checked. If there are corresponding folders on the disk, they will be registered in Path and saved (Figure 2.2, Figure 2.3). If there are no folders, then the paths will not be written (for example, if you delete the External folder in advance, then the paths to the deleted additional libraries that were contained in External will not be written to the Path file).*

It should be noted that the ge0mlib library was originally written for MatLab, and after that the functions were adapted for Octave. The purpose of this adaptation is to use free software that can be installed on any computer without purchasing or manipulating a license (QGIS and Python are good examples of such programs). At the same time, a small part of ge0mlib functions (using MatLab toolboxes) may not work in Octave (at the moment, when developing functions, compatibility with MatLab is primarily taken into account).

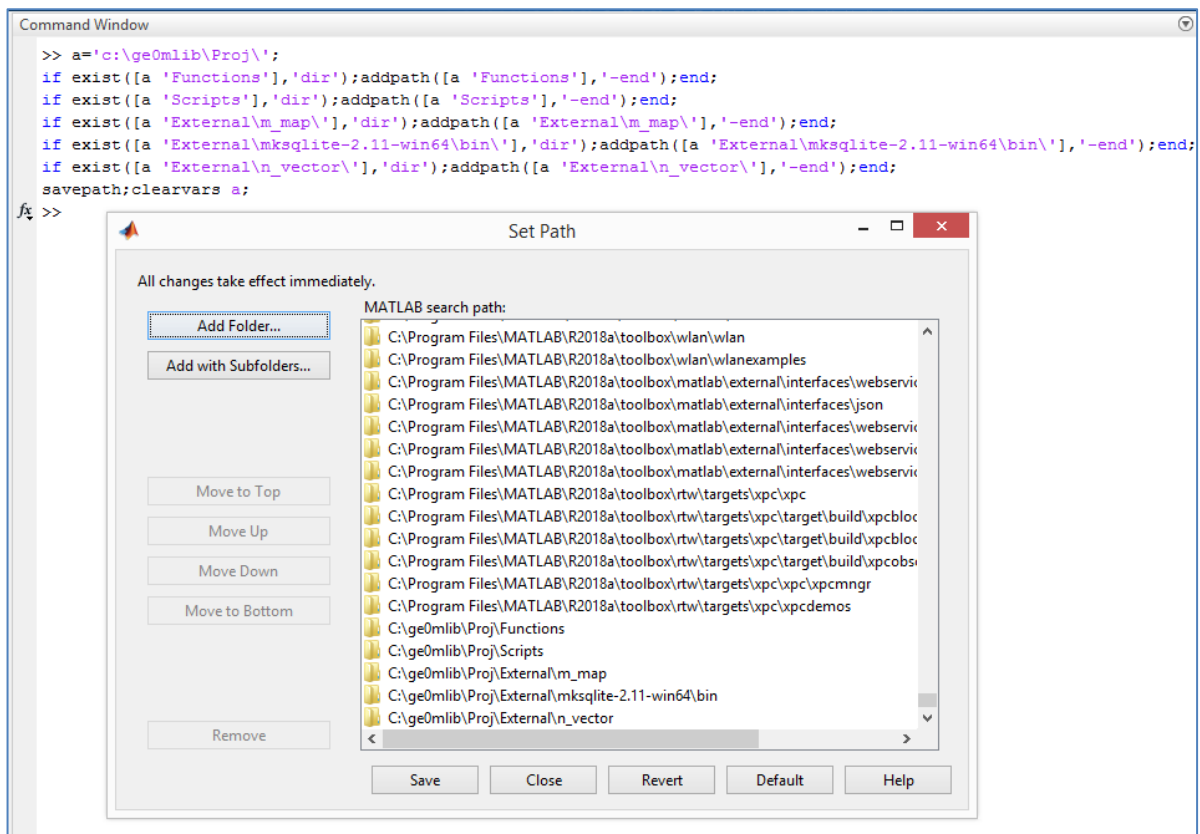


Figure 2.2 MatLab window with paths for the ge0mlib library defined

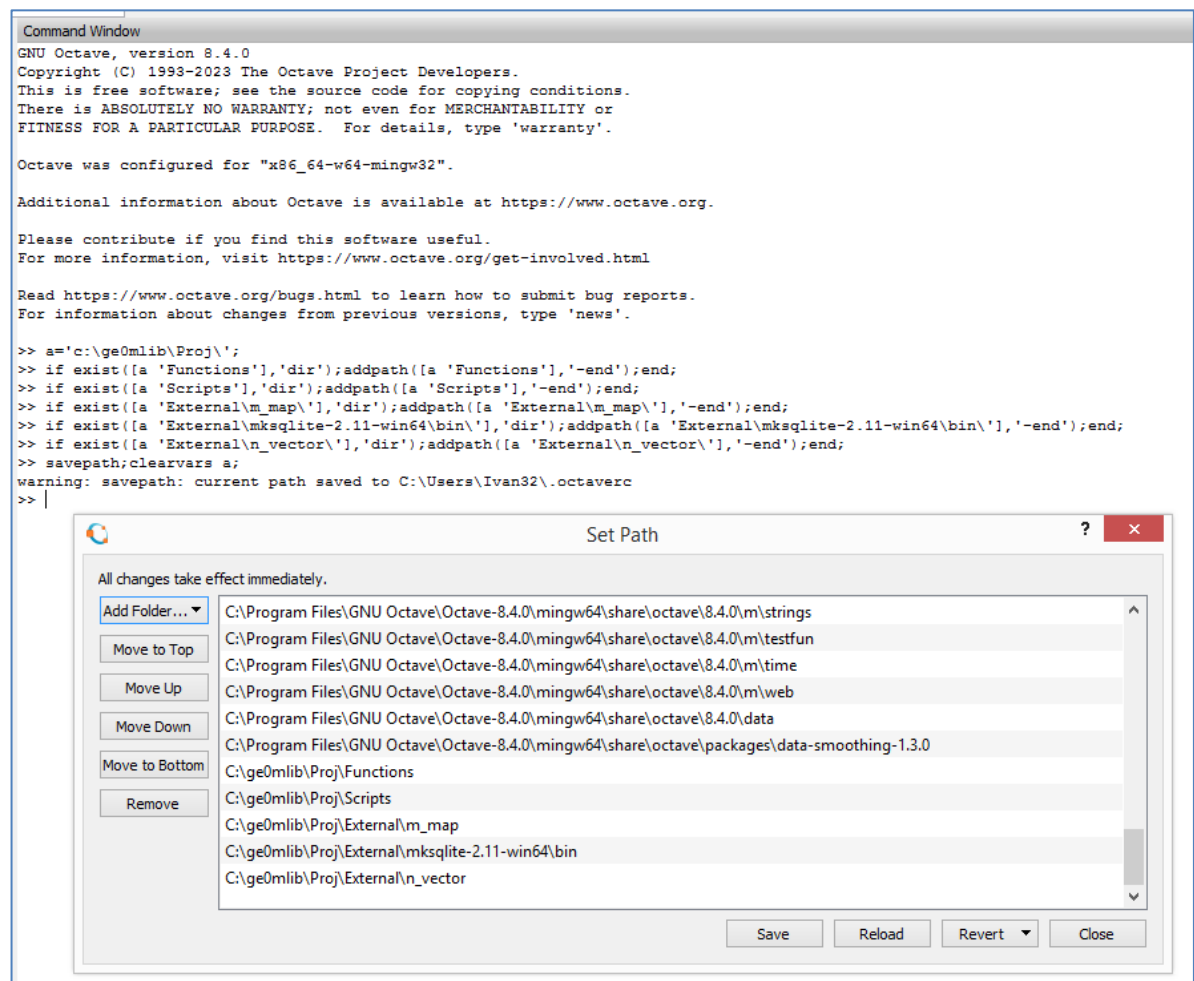


Figure 2.3 GNU Octave window with paths for the ge0mlib library defined

3 Git

3.1 Purposes of using Git and Github

The contents of the Functions folder (ge0mlib library functions) use Git version control. The ge0mlib library is available on Git Hub: <https://github.com/ge0mlib/ge0mlib>. The purpose of using Git, discussed below, is to restore “snapshots” (commits) of a library made in the past and with a 90% chance you will not need this information.

Since the library functions are gradually changing, the “old script” may not run with the “new version of the library”. To run such scripts, it is convenient to be able to select a “snapshot” (commit) of the library while writing the script. To do this, the date of script development must be indicated in the last line of the script code. Using Git, you can select a “snapshot” (commit) prior to the date the script was created and run it without making any corrections to the code. The same applies to example scripts in the Docs folder written after 12/26/2023.

The described solution has its drawbacks – the old version of the library will contain all the old bugs (which is a definite disadvantage), but there will be no new bugs (which is some advantage). We assume that the script, written and tested *at one time*, produced adequate calculation results that were not subject to bugs, and therefore can be used in the future. An alternative solution is to edit the script, making changes to the command syntax (it is assumed that it has changed) for the new version of the library. This is the best solution by far, but it requires time, knowledge, and testing how the modified script works.

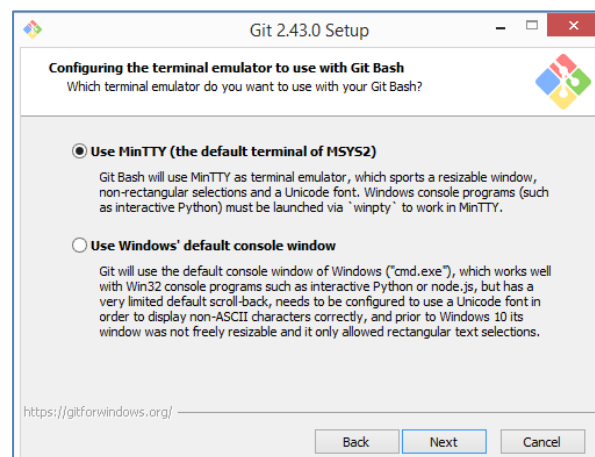
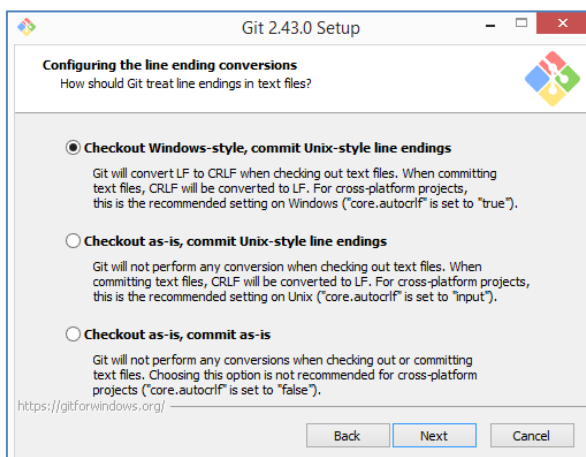
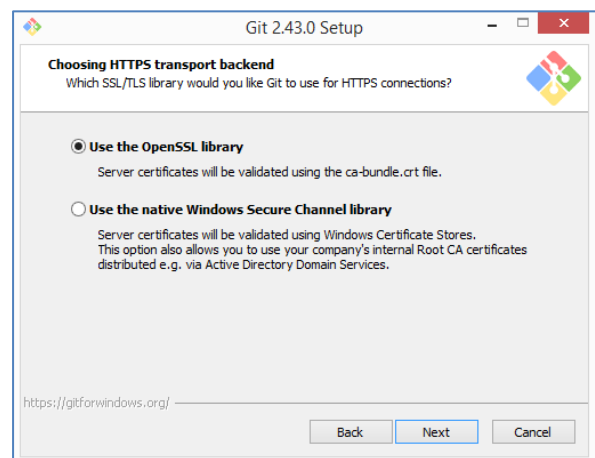
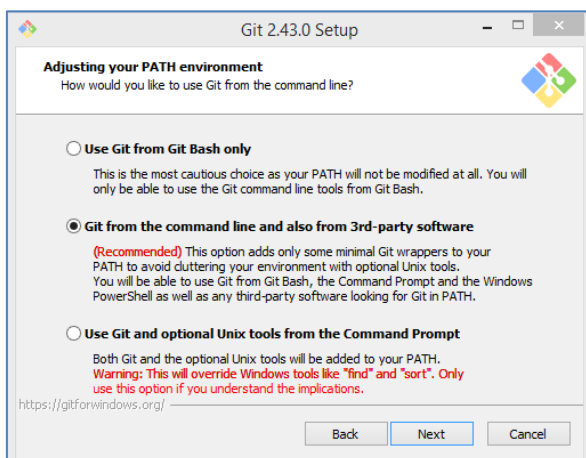
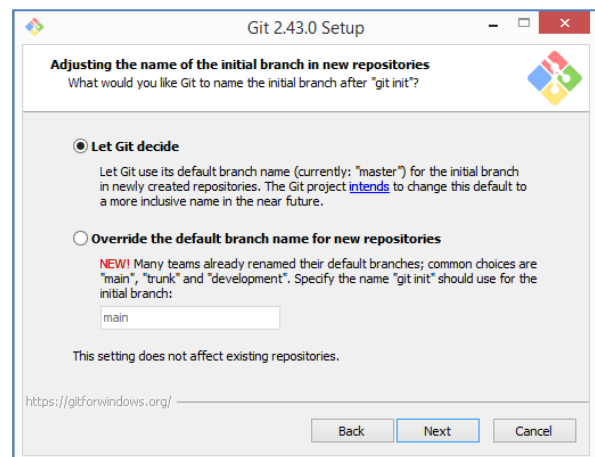
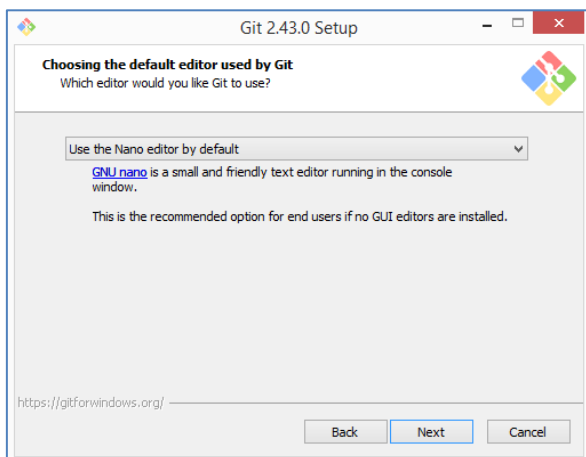
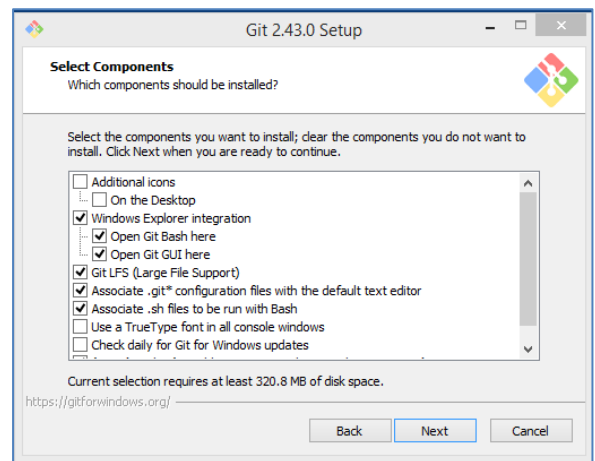
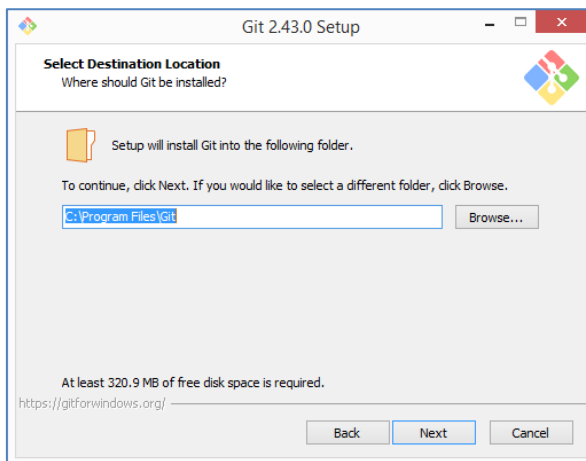
The main purpose of using a tool like Git is a collaborative development. The library is free and open to any changes. If you want to rewrite or add some functions, you can do this in a separate git branch, which you can later upload to Git Hub and merge with the main git branch, including the functions in the library. However, if you don't use Git, you can simply send the m-files and their description to mail@ge0mlib.com, I will add them to Git Hub, documentation and library.

3.2 Git installation

The git distribution is downloaded from <https://git-scm.com/downloads>. At the time of writing this text, the file is Git-2.43.0-64-bit.exe (for Win64). The git installation (*Figure 3.1*) runs with default settings except

- (1) changing the editor to Nano or another convenient one (third screenshot),
- (2) checking the box to launch Git Bash in the last screenshot.

Both these actions are not critical, since after installing git, you can independently replace the editor in the git settings and run git itself.



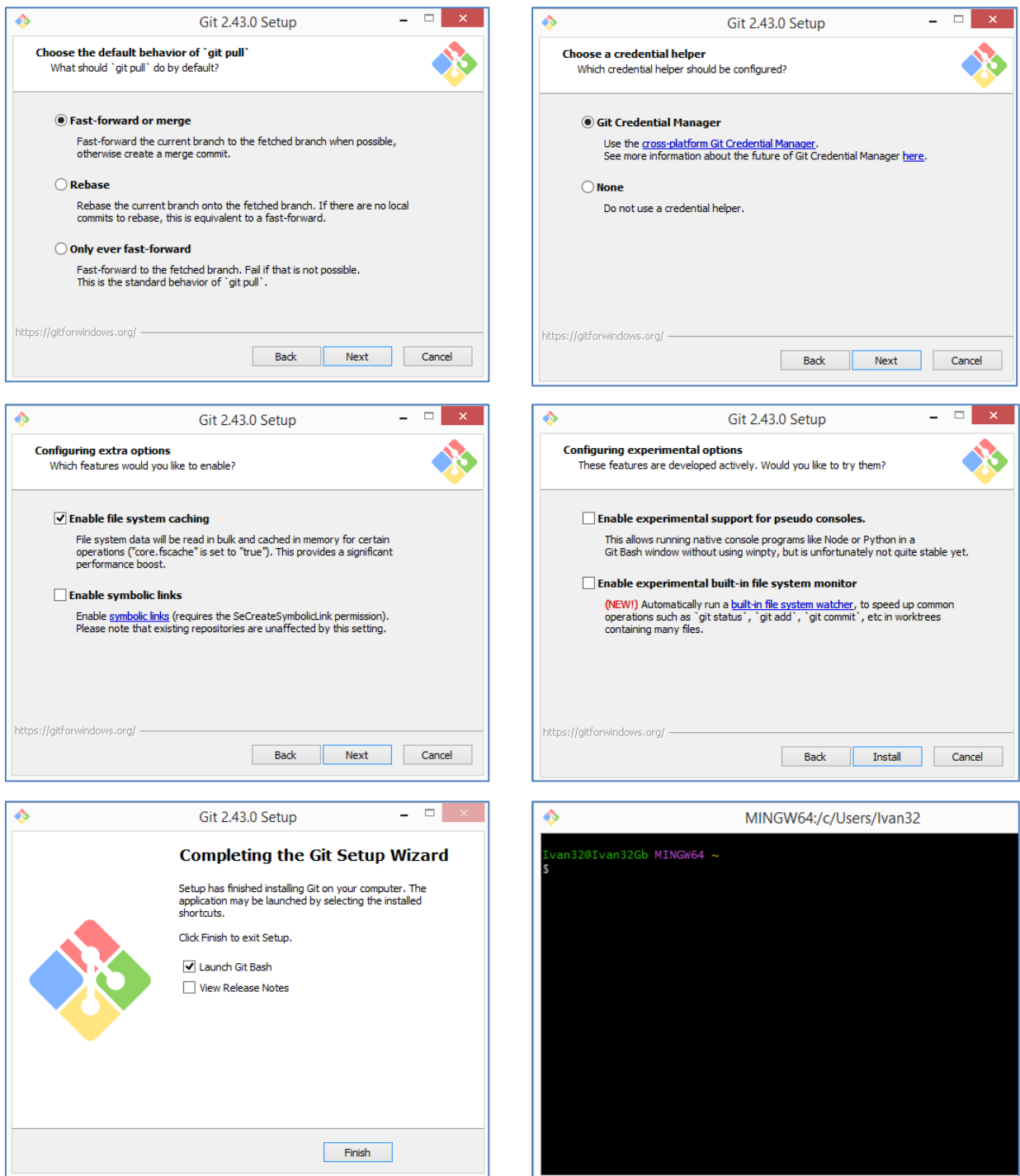
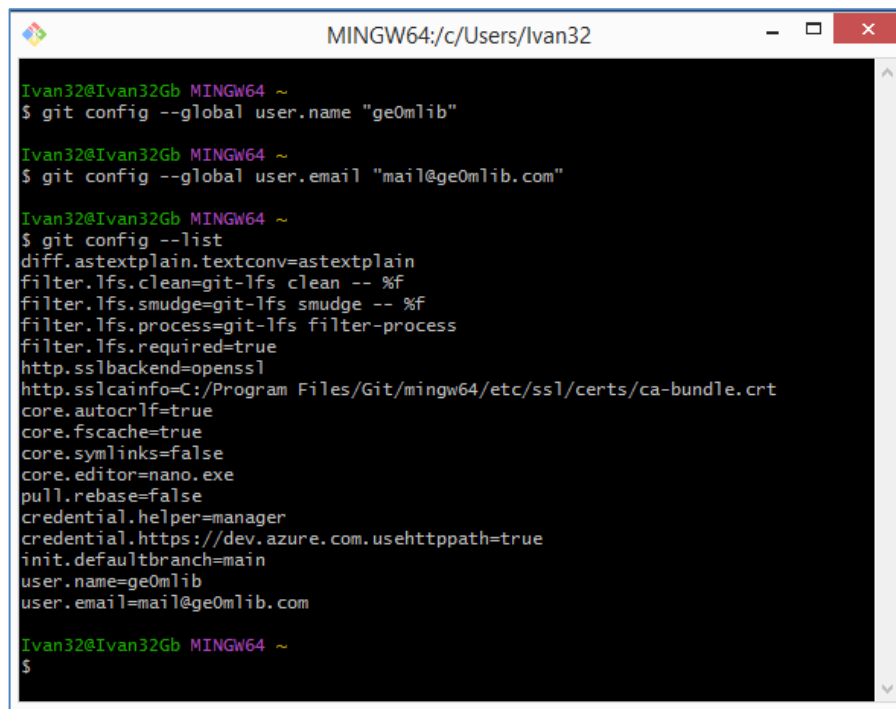


Figure 3.1 Git installation

After installation you need to run the commands below in the git window, by adding your user.name and user.email (**Figure 3.2**). Commands can be copied and pasted using the right mouse button.

```
git config --global user.name "xxx"
git config --global user.email "xxxx@xxx.xxx"
git config --list
exit
```



```
Ivan32@Ivan32Gb MINGW64 ~
$ git config --global user.name "ge0mlib"

Ivan32@Ivan32Gb MINGW64 ~
$ git config --global user.email "mail@ge0mlib.com"

Ivan32@Ivan32Gb MINGW64 ~
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
core.editor=nano.exe
pull.rebase=false
credential.helper=manager
credential.https://dev.azure.com.usehttppath=true
init.defaultbranch=main
user.name=ge0mlib
user.email=mail@ge0mlib.com

Ivan32@Ivan32Gb MINGW64 ~
$
```

Figure 3.2 Executing commands for initial git settings

After executing the exit command, the git window will be closed.

3.3 Creating a git project and downloading ge0mlib from the repository

Before working with git, it is better to delete all files from the Functions folder to avoid conflicts between file versions. In the first step, you can run git and go to the Functions folder using the cd command, for example:

```
cd c:/ge0mlib/Proj/Functions
```

As an alternative, you can open git directly in the required folder by clicking on the right mouse button (**Figure 3.3**).

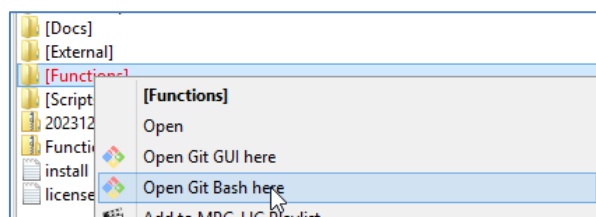


Figure 3.3 Opening Git Bash from the Windows context menu

After running git, you need to enter the following sequence of commands:

-- creating an invisible .git folder containing git information:

```
git init
```

-- adding the path to the repository on Git Hub:

```
git remote add orig https://github.com/ge0mlib/ge0mlib
```

-- copying repository files to the .git folder:

```
git pull orig main
```

An example of the git window and the contents of the Functions folder after executing the commands are shown in [Figure 3.4](#).

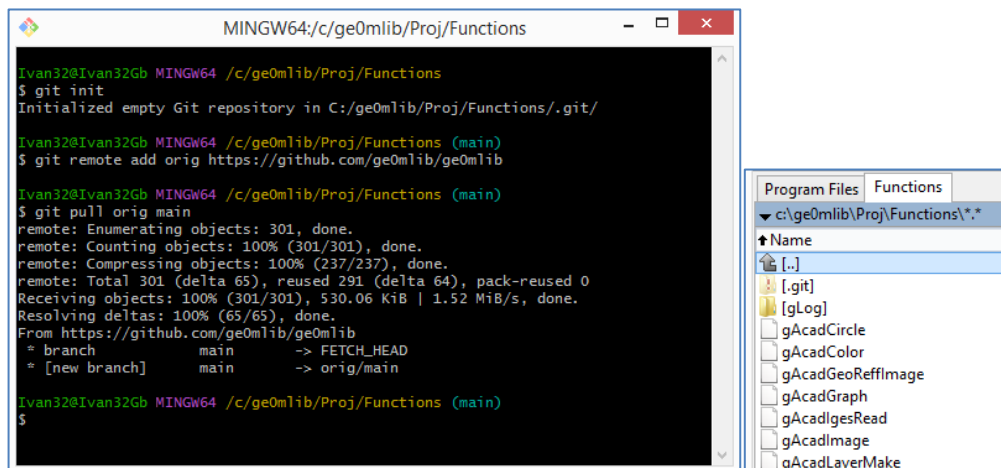


Figure 3.4 The git window and the contents of the Functions folder after creating a repository

3.4 Navigating history in Git

To list the “library snapshots” (commits), you need to use the command:

```
git log
```

Use the “q” key to exit. When the command is executed, git displays a list of commits with information about the date and a unique hash code (name) for each commit ([Figure 3.5](#)).

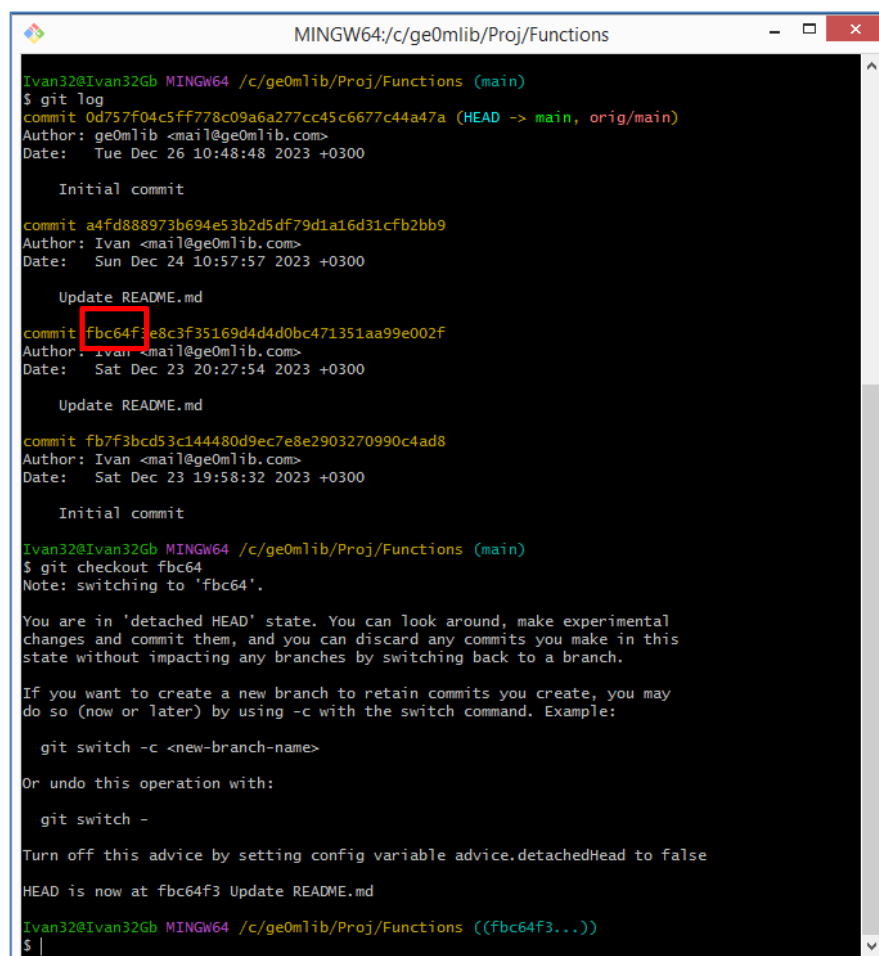


Figure 3.5 The git window with the commit list

You need to select commit by date (so that the date is the closest date before the script date) and use the hash code (in our example it is fbc64) with the “checkout” command:

```
git checkout fbc64
```

After the command is executed, the special git pointer HEAD will move to the commit with hash code fbc64 and git will change the contents of the Functions folder to match the contents on the commit date. The result of changing the contents of the Functions folder, for our example, is shown in [Figure 3.6](#).

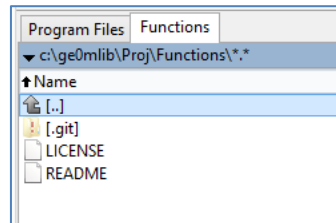


Figure 3.6 Files in the Functions folder as a result of applying the "checkout" command

After such a shift of the HEAD pointer, it is very preferably to create a new branch for this commit (for example, with the name my001) using the command

```
git branch my001
```

Now we can switch between the "latest version" of the library (main branch) and the "my001 version" (my001 branch), using the commands

```
git checkout main
```

```
git checkout my001
```

in this case, the contents of the Functions folder will be replaced each time. *Strictly speaking, not all files will be replaced, but only the *tracked files* contained in the commit (with the appropriate names and hash code).* If we make changes to such a *tracked file*, then when we try to switch between versions of the library, an error message will be displayed, marked with a rectangle in [Figure 3.7](#).

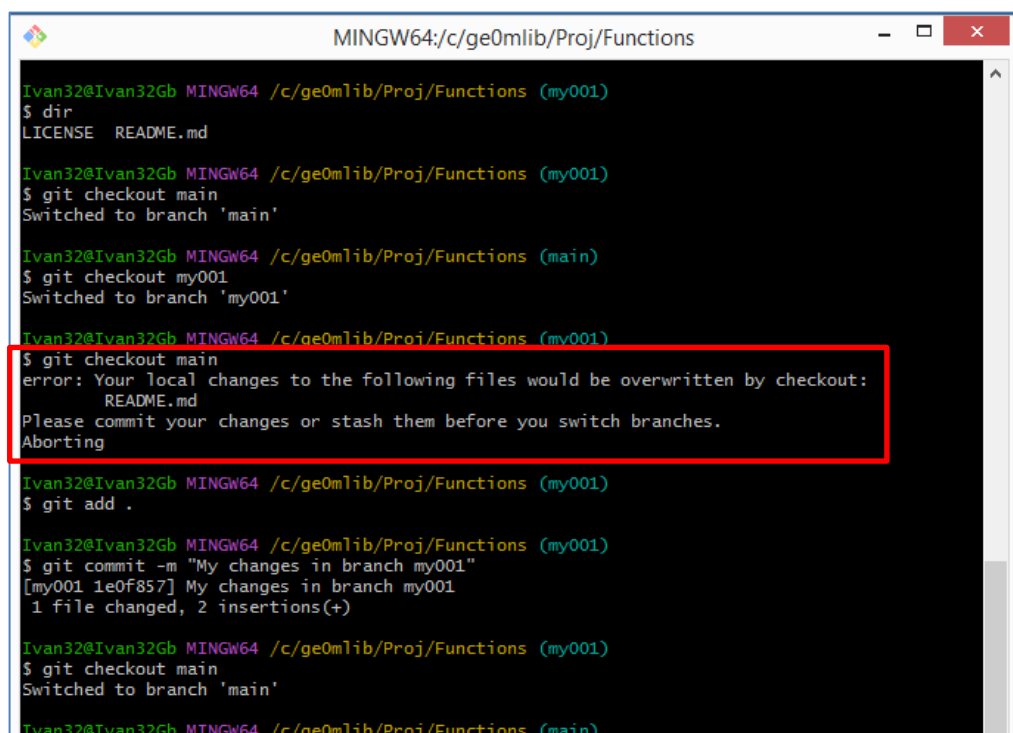


Figure 3.7 A git error telling you to save your last changes as commit

An error message is displayed, because when switching, the recent changes made would be lost (deleted). In order to perform the switch, we need to create a new commit with our changes. To do this, you can run the following command sequence:

```
git add .
git commit -m "My changes in branch my001"
git checkout main
```

Further “deepening into the command system” requires a full study of working with git, which is not included in the objectives of this document.

3.5 Launching the GUI

You can select commit by date not only in the terminal, but also using the graphical interface. To do this, you need to select not “Open Git Bash here”, but “Open Git GUI here” by right-clicking and then – Visualize All Branch History (*Figure 3.8*).

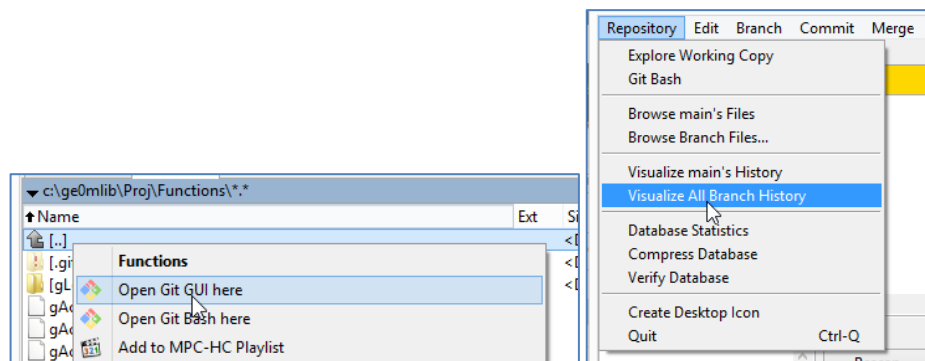


Figure 3.8 Opening Git GUI via Windows context menu

In the window that opens, you can see branches, commits, and hash codes (*Figure 3.9*).

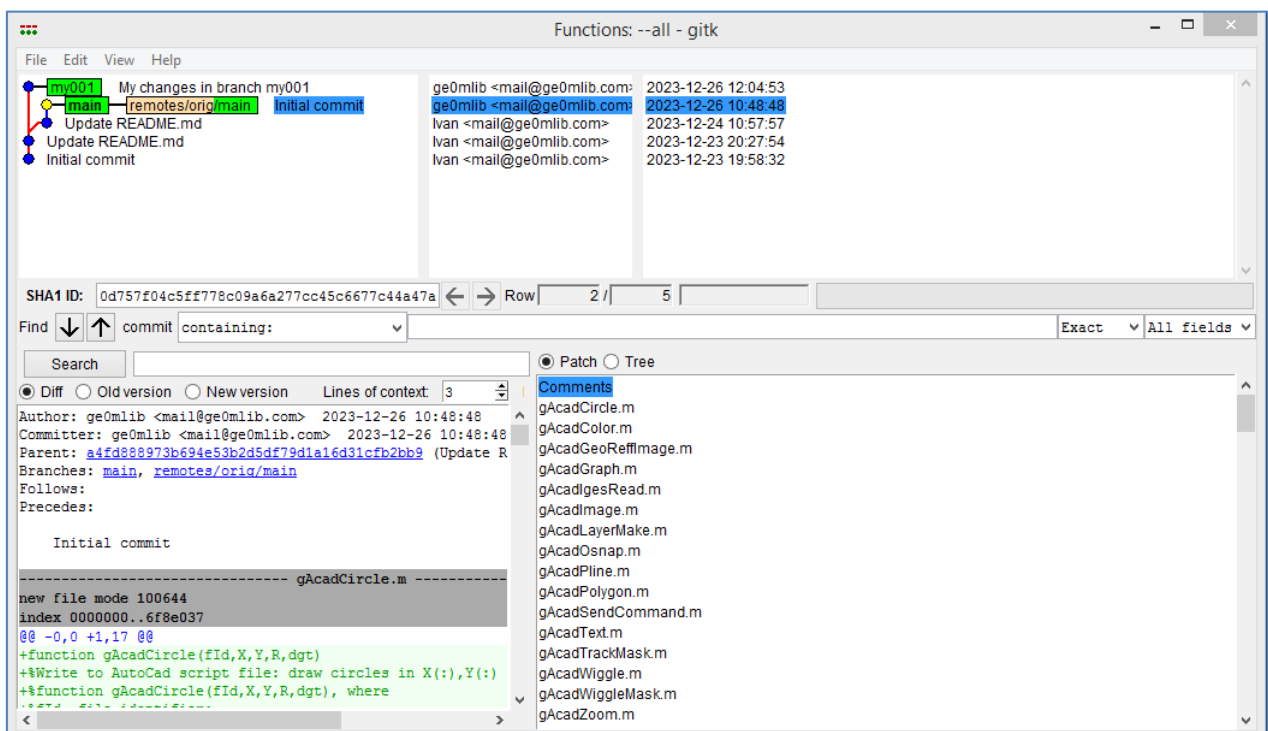


Figure 3.9 Git GUI interface

4 Structure of the script and script commands

There is a certain set of rules for writing ge0mllib library scripts that are recommended to be followed to make the use of scripts by other people is as convenient as possible for them.

Each script can execute multiple *script commands*. For each script command, multiple *script command parameters* can be passed. To execute a script command, you need to enter a *command-to-execute* to the MatLab or Octave window:

```
{'LoadMag1x8_r01','d:\002\1mag\0052_MAG_'};MagToolR01;
```

here:

MagToolR01 – name of the script (name of the file with the script that was copied to the Scripts folder);

{'LoadMag1x8_r01','d:\002\1mag\0052_MAG_'} – parameters of the script command, where the first parameter is always the name of the script command. In curly braces, separated by commas, many parameters of the script command that are required for its execution can be entered;

'LoadMag1x8_r01' – the name of the script command that executes a specific section of the script code;

'd:\002\1mag\0052_MAG_' – a script command parameter which, in our case, contains the path to a specific file that will be used when running the script.

Let's consider the structure of the MagToolR01 script (for recalculating magnetic survey data), which is shown in [Figure 4.1](#) (only part of the code is shown). When creating scripts, it is advisable to adhere to this structure all the time (line numbers, of course, are indicated for a specific example).

- Script name (line 1);
- Brief purpose of the script (line 2)
- List of additional libraries required to execute the script (line 2);
- Description of script commands and, if necessary, command parameters (lines 3-7);
- Several examples *with working command sequences*, written after “Example” (in our case there is only one such sequence, and it is given in line 8);
- Blocks executed according to the conditions for each of the commands (conditions with command names are shown in lines 11, 17 and 31);
- A brief description of the command being executed at the beginning of each block (the description is given as a comment on lines 11, 17 and 31);
- An example of a command and script parameters for each block (examples are given as comments in lines 12, 18, 32);
- A section of code that requests input of script parameters if they were skipped when entering the *command-to-execute*. This code also contains descriptive text for each script parameter (lines 19, 33);
- The clearvars function, executed at the end of each block (corresponding to the script command) to remove extra/temporary variables (lines 29, 37);
- Date of creation of the script, email for communication, MatLab and/or Octave environment indicating the version for which the script was tested (comment in the last line 69: mail@ge0mllib.com 12/10/2023 MatLab2018b&Octave8.4.0).

```

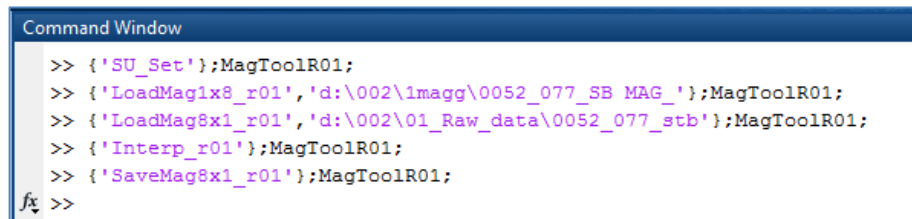
MagToolR01.m x +
1 %script MagToolR01;
2 %Convert eight 1-magy-files and one 8-magys-file to single file where magnetic data carefu
3 %SU_Set -- define Setup values and variables
4 %LoadMag1x8_r01 -- Read file with 1-magy x8-times for the Scanfish
5 %LoadMag8x1_r01 -- Read file with 8-magys for the Scanfis
6 %Interp_r01 -- Create new structure, interpolated to time ti
7 %SaveMag8x1_r01 -- Save structure (was interpolated to ti) to file
8 %Example: {'SU_Set'};MagToolR01;{'LoadMag1x8_r01','d:\002\1magg\0052_MAG_'};MagToolR01;{'I
9
10 gKey=ans;
11 if strcmp(gKey{1},'SU_Set') %define Setup values and variables
12     {'SU_Set'};MagToolR01;
13     SU.Dm=[];
14     SU.ti=[];
15     SU.fName='';
16 end
17 if strcmp(gKey{1},'LoadMag1x8_r01') %Read file with 1-magy x8-times for the Scanfish
18     {'LoadMag1x8_r01','d:\002\1magg\0052_MAG_'};MagToolR01;
19     try fName=gKey{2};catch,fName=input('Magy*8 file names mask for reading=');end
20     ss='Db,DD,MM,YYYY,hh,mm,ss,LineName,CogE,CogN,MruPitch,MruRoll,MruHdt,E,N,Hdt,T,Sig,Dp
21     for n=1:8,
22         [~,M1{n}]=gDataStructRead([fName num2str(n,'%02d.txt')],ss,'%s%f/%f/%f%f:%f:%f%s%f
23         M1{n}.GpsDay=gNavTime2Time('YMD32Dx',M1{n}.YYYY,M1{n}.MM,M1{n}.DD);M1{n}.GpsTime=g
24         M1{n}.ts=smooth(M1{n}.t,60); %p=polyfit(ProfM{n}.GpsNS,AbsTSO,1);ProfM{n}.MagA
25     end;
26     SU.Dm=M1{1}.GpsDay(1);
27     a=zeros(8,1);for n=1:8,a(n)=mean(diff(M1{n}.ts));end;[s,k]=min(a);tmp=fliplr(M1{k}.ts(
28     for n=1:8,s=mean(diff(M1{n}.ts));tmp=fliplr(M1{n}.ts(100):-s:M1{n}.ts(1));M1{n}.ti=[tm
29     clearvars fName ss n a s k tmp ti
30 end;
31 if strcmp(gKey{1},'LoadMag8x1_r01') %Read file with 8-magys for the Scanfis
32     {'LoadMag8x1_r01','d:\002\01_Raw_data\0052_prt'};MagToolR01;
33     try fName=gKey{2};catch,fName=input('8-magys file name for reading=');end;SU.fName=fNa
34     ss='Db,DD,MM,YYYY,hh,mm,ss,LineName,CogE,CogN,MruPitch,MruRoll,MruHdt,E1,N1,Hdt1,T1,Si
35     [~,M8]=gDataStructRead([fName '.txt'],ss,'%s%f/%f/%f%f:%f:%f%s%f%f%f%f%f%f%f%f%f
36     M8.GpsDay=gNavTime2Time('YMD32Dx',M8.YYYY,M8.MM,M8.DD);M8.GpsTime=gNavTime2Time('HMS32
37     clearvars fName ss
38 end;
39 if strcmp(gKey{1},'Interp_r01') %Create new structure, interpolated to ti
40     {'Interp_r01'};MagToolR01;
41     W=M8;

```

Figure 4.1 MagToolR01 script code example

In addition, there are certain variable names (in the example given it is SU) and certain field names for structures (in the example given these are DD, MM, YYYY, hh, mm, ss and others). Which are specialized and common. Using these names will make the script text more readable (and sometimes even necessary for the execution of functions in the code of which such “specialized” fields are written).

With the approach described above, data processing will be implemented as the sequential execution of several script commands with the appropriate parameters. In our example, data processing includes reading several files with magnetometer data (LoadMag1x8_r01, LoadMag8x1_r01 commands), some mathematical calculations based on the loaded data (Interp_r01), outputting the calculation results to a file (SaveMag8x1_r01 command). An example of a window with a completed working sequence of MagToolR01 script commands is shown in *Figure 4.2*.

A screenshot of a MATLAB Command Window. The window has a dark blue title bar with the text "Command Window". The main area is white and contains a sequence of commands entered at the prompt ">>". The commands are: 1. {'SU_Set'};MagToolR01; 2. {'LoadMag1x8_r01','d:\002\1magg\0052_077_SB MAG '};MagToolR01; 3. {'LoadMag8x1_r01','d:\002\01_Raw_data\0052_077_stb'};MagToolR01; 4. {'Interp_r01'};MagToolR01; 5. {'SaveMag8x1_r01'};MagToolR01;. The prompt ">>" is shown at the bottom left of the window, preceded by a small icon of a document with a checkmark.

```
>> {'SU_Set'};MagToolR01;
>> {'LoadMag1x8_r01','d:\002\1magg\0052_077_SB MAG '};MagToolR01;
>> {'LoadMag8x1_r01','d:\002\01_Raw_data\0052_077_stb'};MagToolR01;
>> {'Interp_r01'};MagToolR01;
>> {'SaveMag8x1_r01'};MagToolR01;
fx >>
```

Figure 4.2 MatLab window with executed command sequence for MagToolR01

Conclusion

The document covers:

- installing the freely GNU Octave programming environment and its additional libraries;
- installing the ge0mllib library;
- installing git and using commands to get a commit (“snapshot”) of the ge0mllib library at some point in the past.

A script example for recalculating magnetic survey data (UXO) is given. Considered where the following information can be found in the script text:

- List of additional libraries required to execute the script;
- Description of script commands and command parameters;
- Examples with *working command sequences*;
- Date of the script creation;
- The environment used when testing the script (MatLab, Octave);
- Email of the script developer to write him all what you think about him.

The information provided allows you to install the software relatively quickly and perform data processing using a script (m-file).