EGM722 – Programming for GIS and Remote Sensing

Assessment Guide

Introduction

In this module, you will be learning computer programming for applications in GIS and Remote Sensing. Each week, the practicals will provide you with the skills to apply these techniques, from creating and maintaining repositories for your code using git, writing functions and scripts to perform GIS or Remote Sensing analyses, and reading and understanding documentation in order to be able to use code written by other people.

The purpose of this assessment, then, is to help you to put it all together, and showcase your programming abilities. When writing code and sharing it with others, it is important that the code is well-documented and readable by other people; it is also important that the code is reproducible – that is, it enables us to repeat analyses with the same input data and come up with the same results or conclusions.

Your assessment in this course will be done in two parts, submitted simultaneously: a GitHub repository that hosts your script/code, along with a PDF of a written manual or how-to guide for your code.

Timeline

- **Part 1: The How-to Guide (30%)**. A PDF of a written manual/how-to guide for your code should be uploaded to Blackboard by midnight on the deadline day (check the Home page on Blackboard for this year's date).
- **Part 2**: **The GitHub Repository (70%)**. As part of the written manual, you should provide a link to the GitHub repository where your code is hosted. I will evaluate the code based on the last commit before the deadline – you are welcome to continue developing your code after that, but I will not necessarily use those additions in my evaluation.

Assessment Part 1: The How-to Guide

For the first part of the Assessment, you will submit a written manual/how-to guide for your code. The how-to guide should contain clear information for how to set up and run your code. It should also contain detailed information about what your code does, with appropriate references to scientific literature.

Your guide should be a maximum of 2500 words (excluding any figure/table captions and references), or about 10 pages double-spaced. Please note: as long as the tutorial contains all of the required elements and is clearly written and understandable, there is not technically a minimum number of words here.

Your guide should contain the following elements:

1. An introduction that briefly describes what your code does.
2. A setup/installation section that describes how to install your code, including a list of the main dependencies. You should also include a link to your repository in this section **– this is what I will use to evaluate your code for Part 2 of the Assessment**. If you have developed your code with specific test data, you should clearly instruct the user where they can acquire that test data (or similar test data). You do not necessarily have to provide these datasets in your GitHub repository, but they should be easy for the user to track down.
3. A methods section that clearly explains what your code does – if you're performing a certain kind of analysis, this should explain the steps of the analysis and the theory behind it. This section should be written in the style of a methods section for a journal article or technical report.
4. A section that explains the expected result of running your code.
5. A section that provides some troubleshooting advice in case things go wrong.
6. A section that contains any references used in the text.

In the Assessment resources section of Blackboard, you will find links to examples of tutorials/walk-throughs for various python packages. You are welcome (and encouraged!) to use these as a guide when setting up your tutorial.

Any figures/images should have captions below the figure. They should be numbered in order, and they must be discussed in the text. They should be clearly labelled with a legible font size, and you should choose an appropriate colour map. **If you use figures or images prepared by someone else, make sure they are correctly cited both in the text and in the references list**.

Any tables should have a caption above the table. They should be numbered in order, and they must be discussed in the text. Use appropriate significant figures, and label columns with applicable units in the header.

You should write in a clear style, with correct spelling/grammar. Make sure that the steps of your methodology are clearly described with an appropriate level of detail. You can mention specific functions/methods that your code uses, but you should also describe what those functions/methods actually do.

Assessment Part 2: The GitHub repository

For the final part of the Assessment, I will evaluate your script/code using GitHub repository link provided in your tutorial, based on the last commit before the deadline of midnight GMT at the end of Week 6.

Your code base should have the following elements:

- A README file that contains a brief description of how to install and begin using your code. Please note that this can be essentially the same text as the setup/installation section of your tutorial. This file should be either plaintext (no file extension) or Markdown (.md) so that it is displayed by GitHub on the repository page.
- A LICENSE file. For more information about selecting a license file, see this guide from GitHub.
- A .gitignore file that tells git which files to ignore. You can use this handy tool to create a file using standard examples for python, pycharm, Visual Studio, etc. – you can save the generated file to your computer and include it as part of your repository.
- An environment.yml file that enables a user to easily duplicate your python environment on their computer. I recommend limiting this to the main packages that your script depends on.
- At least one python script that performs the function you are writing the code base for. The script should contain the following elements:
  - External modules imported at the beginning of the script.
  - Functions that you have written to help ensure that the code is readable and reproducible. As a general rule, if you repeat an action more than once, it should be written as a function. Functions should include a docstring that describes what the functions does, the different input parameters, and the outputs (if applicable). There are a number of style guides for writing docstrings – you can find two examples here and here.
  - The code should also have embedded comments, so that someone unfamiliar with your code can read through it and understand what is being done at different steps.
- Evidence of multiple commits – I would like to see that you have been developing the code over time, saving changes as you go. Ideally, as you write the code, you will also be making use of branches. This is not strictly required, but it is a good practice to get into, especially if you end up working collaboratively with other programmers. See this guide for more information about sample workflows.

Please note – I am not necessarily going to be evaluating whether your code produces the “correct” results. What I am looking for is that you have included the necessary elements for someone else to use, work with, and understand your code. I am also looking for evidence that your code is readable, it is documented, and that you have been developing it over time.

Assessment Part 1 Marking Scheme

Your how-to guide will be marked using the rubric below, with additional feedback/comments supplied via TurnItIn/Blackboard.

Introduction (10%)

Distinction (70-100%)	- Introduction briefly and clearly explains the function/purpose of the code and what it does.
Pass (50-69%)	- Introduction is included, but it is not clear what the purpose or function of the code is
Poor/Fail (0-49%)	- Introduction is missing or barely addressed

Setup and Installation (25%)

Distinction (70-100%)	- Installation steps are clearly described and reproducible when followed as written - Dependencies are clearly listed - Instructions for preparing/using test data are included and clearly explained
Pass (50-69%)	- Installation steps are mostly clear, but some steps or important information might be missing - Most dependencies are listed, but not all - Instructions for preparing/using test data are included, but some steps might be missing
Poor/Fail (0-49%)	- Installation steps are unclear or missing - Dependencies are not listed or mostly missing - Instructions for preparing/using test data are not included

Methods (25%)

Distinction (70-100%)	- The steps and and theory behind the analysis are clearly explained with appropriate referencing to relevant scientific literature - Methods are written in the style of a journal article or technical report, without exclusively relying on function/package names.
Pass (50-69%)	- Steps and theory are explained, but not always clearly, or without sufficient references - Methods are described, but the description focuses more on the specific function names rather than what they do
Poor/Fail (0-49%)	- Steps and theory behind the analysis are missing or not clearly explained - Methods are not clearly described, or are written as a series of function names without explaining what those functions do

Expected Results (25%)

Distinction (70-100%)	- Guide clearly describes the expected outcome of running the code, using diagrams, figures, or tables
Pass (50-69%)	- Expected outcomes are described, but not clearly or without adequate examples
Poor/Fail (0-49%)	- Expected outcomes are missing or inadequately described

Troubleshooting (10%)

Distinction (70-100%)	- Guide contains clear steps to take when troubleshooting potential errors
Pass (50-69%)	- Some troubleshooting steps are included, but steps are not clearly described
Poor/Fail (0-49%)	- Troubleshooting steps consist of a link to Stack Overflow

Referencing (5%)

Distinction (70-100%)	- References used are relevant to the report, and are clearly integrated into the text - Any external resources (figures, images, etc.) are properly cited - Reference list is complete, and properly cited according to a format such as Harvard, APA, etc.
Pass (50-69%)	- External references are used, but relevance is not always clear, or they are not well-integrated with the rest of the report - External resources are generally properly attributed/cited - Reference list is mostly correct, but some items might be incorrectly cited
Poor/Fail (0-49%)	- No or few external references are used, or their relevance to the text is not clear - External resources are used without attribution - Reference list is poorly formatted, or items are missing

Assessment Part 2 Marking Scheme

Your GitHub repository will be marked using the rubric below, with additional feedback/comments supplied via TurnitIn/Blackboard.

Required Elements (20%)

Distinction (70-100%)	- GitHub repository contains all of the required elements (README, LICENSE, .gitignore, python script, environment.yml file), and the elements are complete (i.e., not just a blank file)
Pass (50-69%)	- Repository is missing at least one of the required elements, or they are incomplete (i.e., missing dependencies in yml file, blank LICENSE or .gitignore file)
Poor/Fail (0-49%)	- Repository is missing multiple required elements

Documentation (40%)

Distinction (70-100%)	- Code is clearly commented throughout, including function docstrings - Function docstrings contain a description of function purpose, input parameters, and results returned (where appropriate)
Pass (50-69%)	- Code contains some comments, but they do not clearly describe what different steps do - Function docstrings are present but incomplete
Poor/Fail (0-49%)	- Code is barely commented - Functions do not contain docstrings

Programming Elements (20%)

Distinction (70-100%)	- Code makes use of external modules - Code contains functions and is readable
Pass (50-69%)	- Code contains some functions, but some repeated steps of analysis are not included in functions
Poor/Fail (0-49%)	- Code doesn't make use of external modules - Code is not readable, without making use of functions or other elements discussed in the course

Development (20%)

Distinction (70-100%)	- Code shows clear evidence of having been developed over time, with multiple commits that are clearly described/commented - (Optionally) code shows multiple branches of development
Pass (50-69%)	- Code contains very few commits, or commits are not clearly commented
Poor/Fail (0-49%)	- Code consists of a single commit submitted at the last possible moment.