Modern Programming Methods 2021/22 - Assessment 3

Testing, CI & optimization

This file contains instructions for completing the assignment. See the README.md file located in the base folder of this repository for instructions regarding setting up the software.

The assessment is based around testing, documentation and CI for a Gaussian elimination algorithm and then developing and optimising an algorithm for computing the determinant of matrices. ***Note*** that you do not need to understand the details of how to implement a Gaussian elimination algorithm to complete this assignment, however you will need to understand how to multiply two matrices together and how to compute the Determinant of a square matrix. Both of these linear algebra operations are explained below before detailing the assessment.

Matrix multiplication

Let A be an $n \times m$ matrix and B be an $m \times l$ matrix. We define the product of A and B as the dot product/scalar product of each row of the matrix A with each column of the matrix B, that is

$$\begin{aligned} A \cdot B &:= \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m} \\ \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nm} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1l} \\ b_{21} & b_{22} & \dots & b_{2l} \\ \vdots & \vdots & \dots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{ml} \end{pmatrix} \\ &:= \begin{pmatrix} \sum_{j=1}^m a_{1j}b_{j1} & \dots & \sum_{j=1}^m a_{1j}b_{jl} \\ \sum_{j=1}^m a_{2j}b_{j1} & \dots & \sum_{j=1}^m a_{2j}b_{jl} \\ \vdots & \dots & \vdots \\ \sum_{j=1}^m a_{nj}b_{j1} & \dots & \sum_{j=1}^m a_{nj}b_{jl} \end{pmatrix} \end{aligned} \tag{1}$$

and hence the result is an $n \times l$ matrix. A matrix is said to be square if $n = m$.

Example 1

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 6 \end{pmatrix} = \begin{pmatrix} 5 + 12 \\ 15 + 24 \end{pmatrix} = \begin{pmatrix} 17 \\ 39 \end{pmatrix}$$

Example 2

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 1 \\ 6 & 2 \end{pmatrix} = \begin{pmatrix} 5+12 & 1+4 \\ 15+24 & 3+8 \end{pmatrix} = \begin{pmatrix} 17 & 5 \\ 39 & 11 \end{pmatrix}$$

Determinant

Consider an $n \times n$ matrix A. Furthermore, denote B_{ij} the $(n-1) \times (n-1)$ matrix obtained from A by removing the i-th row and the $j - th$ column. Then, it holds true that

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(B_{ij}) \tag{2}$$

for any fixed i (row expansion), and

$$\det(A) = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} \det(B_{ij}) \tag{3}$$

for any fixed j (column expansion). The term $(-1)^{i+j}$ can be found easily by thinking of a chessboard:

$$\begin{pmatrix} + & - & + & \cdots \\ - & + & - & \cdots \\ + & - & & \\ & & \ddots & \vdots \\ \cdots & \cdots & - & + \end{pmatrix}$$

By subsequent application, the computation of a determinant is broken down into a computation of many 2×2 determinants.

Example 3

$$\det \begin{pmatrix} 2 & 3 & 7 & 9 \\ 0 & 0 & 2 & 4 \\ 0 & 1 & 5 & 0 \\ 0 & 0 & 0 & 3 \end{pmatrix} = 2 \cdot \det \begin{pmatrix} 0 & 2 & 4 \\ 1 & 5 & 0 \\ 0 & 0 & 3 \end{pmatrix} = 2 \cdot (-1) \cdot \det \begin{pmatrix} 2 & 4 \\ 0 & 3 \end{pmatrix} = 2 \cdot (-1) \cdot 6 = -12.$$

Here we have chosen which row/column to expand to minimize our workload. This example demonstrates that Laplace's formula saves effort when expanding a row or column containing many zeros.

Assessment

1. The first part of the assessment will be to expand on the current testing suite and improve the documentation. The repository currently contains the following files/folders:

   - `docs`: Contains files for building `sphinx` documentation
   - `documentation`: Contains these assessment instructions. Should not be modified.
   - `mpm_la`: Contains `gauss.py` which in turn contains the functions installed when setting up this package, `gauss`, `matmul` and `zeromat`.
   - `README.md`: Contains instructions for installing the initial package.
   - `environment.yml/requirements.txt`: For installation. See `README.md`.
   - `results`: Empty folder. Will be needed later.
   - `scripts`: Empty folder. Will be needed later.
   - `setup.py`: For installation purposes.
   - `tests`: Folder for placing all tests. Currently contains a single test for `gauss`.

   Complete the following tasks:

   (a) Add docstrings to the functions `matmul` and `zeromat`. Their format should mirror that of `gauss` and they should both contain examples of their use.

   (b) Expand `test_gauss` such that two (or more) additional sets of inputs are being tested. These should be chosen such that a suitable range of inputs are tested.

   (c) To `test_gauss.py`, add additional tests for `matmul` and `zeromat`. (Note that you'll need to make these functions visible to the test file). Your tests should make use of the parameterize decorator to test multiple inputs.

   (d) Next, add an additional file in tests called `test_docstrings.py` that tests the docstring tests in each of the three functions present in `gauss.py`. **Important:** This file should be continuously updated to test the docstrings of any new functions added later.

   (e) Finally for part 1, using `sphinx` and the files present in `docs/` build the documentation. The final form of the documentation should be a pdf file named `mpm_la.pdf` located in the `docs/` folder. Note, using `sphinx` you'll first create html files and then these should be compiled into the pdf. **This documentation should also be continuously updated such that on submission it reflects the final state of your repository.**

[35 marks]

2. The next task is to add some CI in the form of Github Actions workflows. These workflows should be placed within the repository in the `.github/workflows` folder. **All workflows should be passing at the time of your final submission**. Some marks will be reserved for neatness and conciseness.

   (a) Create a workflow that checks the repository is PEP8 compliant. The workflow should trigger when (at the very least) a push is made the main branch.

   (b) Create a workflow that runs `pytest` on all test files present within the `tests/` folder. The workflow should execute the tests on the following operating systems: (i) Ubuntu 20.04, & (ii) Windows Server 2019.

   (c) Create a workflow that creates an Anaconda environment from the `environment.yml` file present in your repository and then executes `pytest` for all tests in the `tests` folder as in part (b). This may be in a separate file or in the same `.yml` as that used for part (b).

   (d) Create a workflow that builds the latest version of your `sphinx` documentation and if necessary commits and pushes it to your repository.

   [35 marks]

3. If we simply wish to compute the determinant of a matrix (e.g. `det = gauss(A, I)`), clearly our current algorithm is not optimal, especially when the matrix becomes large (check for which sizes it becomes painfully slow!). Lets see how bad it is and if we can do better.

   (a) Within the `scripts/` folder add a file called `det_timings.py`. This script should, for many (≈ 10) square matrices of increasing size, record the time taken by the `gauss` algorithm to compute the determinant of these matrices. Additionally, for each of these matrices compute the time taken by `numpy.linalg.det` to calculate the determinant. Timing results should be written automatically by the script to a file named `timings.txt` (or `timings_`*`something`*`.txt`)in the `results/` folder with the formatting illustrated in Table 1.

2	0.001	0.001
4	0.02	0.01
⋮	⋮	⋮
100	5.0	1.0

Table 1: Example formatting of the results table. The first column represents the size of the matrix along one of its axes. The second column represents the corresponding timing in a suitable unit of the `gauss` algorithm and the final column that of `numpy.linalg.det`.

(b) Add a new workflow that, using a single operating system of your choice, executes the script `det_timings.py`, commits the new results (i.e. the new `timings_xxxx.txt` produced) and pushes them to your github repository.

(c) In the `mpm_la` folder add a new file `det.py` that contains a function (that you will write) named `det`. This function should be your own algorithm to compute *only* the determinant of a single square matrix that's passed to it. How does your implementation compare to that of `gauss` and `numpy.linalg.det`? Have your script `det_timings.py` also compute the timing of your `det` algorithm and add your results as a third column in your timings file.

[30 marks]

Notes:

- Any new function or test should follow the same standards as those implemented in part 1.
- Remember to ensure that the final version of your repository is `PEP8` compliant.
- Keep the sphinx documentation up to data.
- Additionally, ensure that your `environment.yml` and `requirements.txt` files have been updated appropriately to reflect any new dependencies you've added.
- For part 3(b) you have have your script and workflow overwrite the existing `timings.txt` file *or* create a new file of the form `timings_{some identifier}.txt` and commit that upon each execution. If you choose the later format, can you think of a simple regression test you could add?