ACSE-1 2020/21 - Assessment 3

Testing, debugging, 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 debugging, adding tests, docstrings 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:

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

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

Example 3

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

Note that 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. Currently, running `flake8` from the base folder of this repository will reveal several errors. Running `pytest tests/` will also reveal that the single test located in the file `tests/test_gauss.py` fails – this is due to two bugs, one bug with the matrix multiplication algorithm and one bug due to the computation of the determinant. Further, attempting to build the `sphinx documentation` located in `docs` will fail due to a couple of errors.

(a) Make the repository PEP8 compliant (i.e. fix the `flake8` errors)

(b) Add docstrings to the functions `matmul` and `zeromat`

(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. In doing this, you'll notice that any suitable test for the `matmul` function fails. Debug this function such that your test passes.

(d) With `matmul` fixed you'll notice that the `gauss` related test is still broken. Fix this bug so that `test_gauss` passes and utilising the parameterize decorator add further (at least one) set of test inputs. **Note:** A properly working `gauss` function should be able to correctly return non integer determinants and this functionality should be tested.

(e) 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`.

(f) Finally for part 1, fix the bugs present in `docs/conf.py`. When fixed, use `sphinx` to build the associated `html` files and compile these into a `pdf` file named `ACSE_1a.pdf` which should be located in the `docs` folder. This documentation should be updated to reflect the final state of your repository.

[40 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.

(a) Create a workflow that checks the workflow 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) MacOS 11.0 & (iii) Windows Server 2019. (You may utilize the default `Python3` distribution available on those operating systems).

[25 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 ($10,000 \times 10,000$). Lets see how bad it is and if we can do better.

2	0.001	0.001
4	0.02	0.01
⋮	⋮	⋮
10000	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`.

(a) Within the `scripts/` folder add a file called `det_timings.py`. This script should, for many (≈ 10) square matrices of increasing size ($2 \times 2 \Rightarrow 10,000 \times 10,000$), compute 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` in the `results/` folder with the formatting illustrated in Table 1.

(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.txt` produced) and pushes them to your github repository.

(c) In the `acse_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 `timings.txt`

[35 marks]

Notes:

- Remember to ensure that the final version of your repository is `PEP8` compliant.
- Keep the sphinx documentation up to data.
- Additionally, ensure that your `requirements.txt` file has 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.