Modern Programming Methods 2022/23 - Assessment 2

Numerical package development

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 python package you will be developing.

Before introducing the specification of the assessment, we review some linear algebra definitions.

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 $n \times n$ square 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, Laplace's formula for computing the determinant is

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

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

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

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.

Transpose

For any matrix A, the transpose of A, denoted by A^T, is defined as the matrix obtained from A by switching rows with columns. That means, the i-th column of A^T is defined as the i-th row of A (or vice versa). If A is a $n \times m$ matrix, A^T is a $m \times n$ matrix.

Example 4

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad A^T = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}.$$

Adjugate matrix

The adjugate (classical adjoint) of a square matrix A is defined as follows. As when defining the determinant, denote again B_{ij} to be the $(n-1)\times(n-1)$ matrix obtained from A by removing the i-th row and the $j-th$ column. Then, if A is an $n \times n$ matrix, $\mathrm{adj}(A)$ is an $n \times n$ matrix with the element i, j given by

$$\mathrm{adj}(A)_{ij} := (-1)^{i+j} \det(B_{ji}).$$

Example 5

Let

$$A = \begin{pmatrix} 1 & 0 & -1 \\ -2 & 3 & 0 \\ 1 & -3 & 2 \end{pmatrix}$$

The cofactors, $\det(B_{ij})$*, of the elements are:*

$$\textit{1st row:} \quad \begin{vmatrix} 3 & 0 \\ -3 & 2 \end{vmatrix} = 6, \quad -\begin{vmatrix} -2 & 0 \\ 1 & 2 \end{vmatrix} = 4, \quad \begin{vmatrix} -2 & 3 \\ 1 & -3 \end{vmatrix} = 3,$$

$$\textit{2nd row:} \quad -\begin{vmatrix} 0 & -1 \\ -3 & 2 \end{vmatrix} = 3, \quad -\begin{vmatrix} 1 & -1 \\ 1 & 2 \end{vmatrix} = 3, \quad -\begin{vmatrix} 1 & 0 \\ 1 & -3 \end{vmatrix} = 3,$$

$$\textit{3rd row:} \quad \begin{vmatrix} 0 & -1 \\ 3 & 0 \end{vmatrix} = 3, \quad -\begin{vmatrix} 1 & -1 \\ -2 & 0 \end{vmatrix} = 2, \quad \begin{vmatrix} 1 & 0 \\ -2 & 3 \end{vmatrix} = 3,$$

Hence

$$\mathrm{adj}(A) = \begin{pmatrix} 6 & 4 & 3 \\ 3 & 3 & 3 \\ 3 & 2 & 3 \end{pmatrix}^T = \begin{pmatrix} 6 & 3 & 3 \\ 4 & 3 & 2 \\ 3 & 3 & 3 \end{pmatrix}.$$

Matrix inverse

The inverse of an $n \times n$ matrix A, written as A^{-1}, is defined as the matrix with the property

$$AA^{-1} = A^{-1}A = I,$$

where I is the identity matrix. There are many, *many*, methods for computing inverses, below we introduce one. Using the definitions above, we have that

$$A^{-1} = \frac{1}{\det(A)} \mathrm{adj}(A).$$

Example 6

Again, let

$$A = \begin{pmatrix} 1 & 0 & -1 \\ -2 & 3 & 0 \\ 1 & -3 & 2 \end{pmatrix}$$

We have that

$$\det(A) = 3, \;\; \text{adj}(A) = \begin{pmatrix} 6 & 3 & 3 \\ 4 & 3 & 2 \\ 3 & 3 & 3 \end{pmatrix},$$

and hence

$$A^{-1} = \begin{pmatrix} 2 & 1 & 1 \\ 4/3 & 1 & 2/3 \\ 1 & 1 & 1 \end{pmatrix},$$

and note that

$$AA^{-1} = A = \begin{pmatrix} 1 & 0 & -1 \\ -2 & 3 & 0 \\ 1 & -3 & 2 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 \\ 4/3 & 1 & 2/3 \\ 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Cramer's rule

Cramer's rule is one method for solving linear systems of the form $Ax = b$ when A is square and $\det(A) \neq 0$. You will encounter, and study in more detail, other methods later in the year.

Given a linear system of the form $Ax = b$, with A an $n \times n$ matrix, Cramer's rule yields solutions according to

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix},$$

where

$$x_i = \frac{1}{\det(A)} \cdot \det(B_i),$$

and B_i is obtained from A by replacing the $i-th$ column with b.

Example 7

Consider $Ax = b$ *with* $A = \begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \end{pmatrix}$, $b = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}$. *First we compute* $\det(A) = 4$ *which tells us that there is a unique solution and we can, for example, use Cramer's rule to compute it.*

$$\det(B_1) = \det \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 1 \\ -1 & -1 & 1 \end{pmatrix} = -2$$

$$\det(B_2) = \det \begin{pmatrix} 1 & 1 & 1 \\ -1 & 2 & 1 \\ -1 & -1 & 1 \end{pmatrix} = 6$$

$$\det(B_3) = \det \begin{pmatrix} 1 & 1 & 1 \\ -1 & 1 & 2 \\ -1 & -1 & -1 \end{pmatrix} = 0$$

$$\Rightarrow x = \frac{1}{4} \begin{pmatrix} -2 \\ 6 \\ 0 \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} \\ \frac{3}{2} \\ 0 \end{pmatrix}$$

Assessment

Upon accepting the assignment you will receive a repository containing the skeleton of the `mpm_la` package. The contents of the repository is as follows

- `assessment.pdf`: File containing background regarding the required linear algebra and specifications of this assessment (and this text!).
- `docs/`: A folder for you to build your autodocumentation in part 2. Is currently empty except for a `.gitignore` file.
- `environment.yml`: For installation. See `README.md`.
- `examples`: Folder for part 3 of the assessment.
- `mpm_la`: Contains `functions.py` which in turn contains the skeleton of the `det` function for computing determinants of matrices.
- `README.md`: Contains instructions for installing the initial package.
- `requirements.txt`: For installation. See `README.md`.
- `setup.py`: For installation.
- `tests`: Currently empty folder to which you will add tests in part 1.

The assessment will be separated into 3 parts as follows:

1. Complete the following tasks:
   (a) Complete the function `det` and add an additional example to its docstring.
   (b) Add, to `function.py`, the functions `mult(a, b)`, `adj(a)`, `inv(a)` and `solve(a, b)` (which uses Cramer's rule to return the solution of $Ax = b$). These functions should be complete with suitable docstrings containing usage examples. You may use

`numpy.transpose` to compute the transpose of matrices but all other operations should be hand coded. These functions should all be exposed to the user.

(c) In the `tests/` folder, add a file/files containing a suitable range of tests for all functions exposed to the user. Your tests should make use of the parameterize decorator to test multiple inputs.

(d) Finally, create a file called `test_docstrings.py` in the `tests/` folder that tests all usage examples contained in docstrings present in your code. Its usage should be via, e.g., `pytest tests/test_docstrings.py`. You may find `import_module` from the `importlib` useful for this implementation (`from importlib import import_module`).

[40 marks]

2. The next task is to add auto-documentation and CI in the form of Github Actions workflows. Workflows should be placed within the repository in the `.github/workflows` folder.

(a) Add suitable `conf.py` and `index.rst` files to the `docs/` folder. Use these in tandem with `sphinx` to generate a pdf file named `mpm_la.pdf` within the `docs/` folder. This pdf should contain a suitable description of the repository and document all functions exposed to users.

(b) 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.

(c) 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) Ubuntu 20.04 and for both Python <= 3.8 and Python >= 3.9.

(d) 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. This may be a separate matrix entry to those jobs of part (c), a separate job or in a separate .yml file.

(e) 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. For the final part of the assignment we will write some usage examples, time some functionality and try and possibly develop a faster method for solving linear systems of the form $Ax = b$.

(a) In the folder `examples/`, write a short notebook introducing and demonstrating to a new user how to use your package. At least one example of each of the user exposed functions should be covered.

(b) In a separate notebook, for roughly 10 square matrices of increasing size, time the speed of the operations `mult(inv(a),b)`, `solve(a, b)` and `numpy.linalg.solve(a, b)`. Compile your results into a `pandas` data frame and display this within the notebook.

(c) Can you implement a different method, of your choice, to solve $Ax = b$ that is quicker than the current `solve(a, b)` using Cramer's rule? New functions added should adhere to the standards of those previously implemented. Compare the speed of your solution to those timed in 3.(b) and display the timings for your new method in the same notebook.

[25 marks]

Notes/tips:

- Within in each section, marks will be awarded for the clarity, quality and structure of the code written and package developed.
- All tests and workflows should be passing at the time of your final submission.
- You may implement utility functions (not exposed to the user) as required. These may be located in `functions.py` or some other suitable utility functions file.
- Remember to ensure that the final version of your repository is `PEP8` compliant.
- Keep the sphinx documentation up to date.
- Additionally, ensure that your `environment.yml` and `requirements.txt` files have been updated appropriately to reflect any new dependencies you've added.
- Jupyter notebooks can be tested using the nbval package.