Instructions for the programming project

This document is explaining to you what you will have to do (and how you will do it) for the CS101 programming project in 2014/15 at *Mahindra École Centrale*.

Topic: You and your team-mates will work on matrices and linear algebra operations. This is not something new, as we are using examples from MA102 since February, but you will have a lot to do. The goal of this project is to use concepts from OOP (as taught by Prof. Vipin) in order to be able to perform linear algebra operations in a really natural manner. At the end, you will be able to define, read and modify matrices with a syntax close to maths (`A[i, j]`), and compute determinant, rank, trace, inverse with `A.det()` or `matrix.trace(A)`, and compute inverse with `A**(-1)` or `A.inv()`.

General instructions: Please refer to the main document we gave you, it contains general advices, details about the deadline, and tells you how to submit your work.

---

1 What do you have to do?

1.1 In a file called `matrix.py`

1. Define a class `Matrix` to represent (n, m) matrices. As previously, implement matrix as list of lists. The `__init__` method should be defined like this : `def __init__(self, M)` where `M` is a list of row vectors. This list of list `M` will be stored in an attribute called `listrows`.

   Here is a snippet of code from where you can start:

```
# In the file matrix.py
class Matrix():
    """ Represents matrices of size (n, m). """

    def __init__(self, M):
        """ Create a matrix from the list of row vectors M. """
        self.listrows = M
        self.n = len(M)
        self.m = len(M[0])

    ...  # keep working from that point
```

2. First, define a method called `__getitem__` (with `def __getitem__(self, i, j)` in the `Matrix` class body) to authorize reading the (i, j) value of `A.listrows` with a syntax as simple as `A[i,j]`.

3. Then define as many **special methods**[1] as you need, in order to be able to manipulate `Matrix` objects easily. A list of such method can be found at docs.python.org/2/reference/datamodel.html#emulating-numeric-types or docs.python.org/2/library/operator.html.

4. At least, implement the sum, difference, multiplication, and integer power for matrices, by using the respective special method. For instance, `def __add__(self, B)` will define the method `A.__add__`, which is called when we ask to compute the matrix $A+B$: `A + B` is computed by calling `A.__add__(B)`.

5. Write a function `rand_matrix(n, m, K)` for generating a random matrix (ie. instance of the Matrix class) of size (n, m) with each coefficients being integers, randomly taken between $-K$ and K.

6. Define a function `gauss` in the module `matrix.py`, and then use it to add a method `A.gauss()` to the class `Matrix`. This function will apply the Gauss elimination algorithm to the matrix self (use the reference algorithm on Wikipedia if needed : en.wikipedia.org/wiki/Gaussian_elimination).

[1] Special methods names are between double underscores : `__init__`, `__getitem__` or `__add__`. They are used to modify the way Python interprets its command.

7. Use the Gauss algorithm to write a function (and a method) `det` that compute the determinant. Hint: you might have to adapt the previous function a little bit (cf. MA102 or Wikipedia).

8. Similarly, use the Gauss method or function to write a function and a method `rank` for computing[2] the rank of the matrix.

9. Write a few utility functions, like `ones(n, m)` and `zeros(n, m)` which make a (n, m) matrix filled of 1 or 0, `eye(n)` for the (n, n)identity, `diag(d)` which takes a n-sized vector $\mathbf{d} = (d_1, \dots, d_n)$ and creates a matrix with these values d_i on the diagonal etc. You can include any function which seems useful, of course.

1.2 In a file called `tests.py`

1. Write *at least two examples* (of different size) for *every* method or function you implemented in the `matrix.py` file.

2. Do not use random matrix ! We need to check that the examples are computed correctly.

3. For example, this `tests.py` file will look like this:

```
1  import matrix
2
3  A = matrix.Matrix( [ [1, 2], [3, 4] ] )
4  print "A =", A
5  B = matrix.Matrix( [ [5, 6], [7, 8] ] )
6  print "B =", B
7
8  print "A + B =", A + B   # uses A.__add__(B)
9  print "A * B =", A * B   # uses A.__mult__(B)
10 print "A ** 13 =", A ** 13   # uses A.__pow__(13)
11 # many more to do !
12 print "det(A) =", A.det() # uses the method A.det()
13 print "det(B) =", matrix.det(B)   # calls the function matrix.det(B)
```

4. Feel free to get examples from the MA102 course, like the examples seen in lectures, or the problems in either the exams or the tutorial sheets. Any interesting examples that you want to illustrate or compute in your project will help you having a better mark.

1.3 In your project report

- Explain any choice or special hypotheses you chose to follow for all the algorithms you implemented (especially Gauss, exponentiation, Gram-Schmidt, det and rank if you implement them).
- If you choose to implement any extra features to the `matrix.py` file (it can be an extra function or method), please explain it in details in your report.
- For *each* function and method you write, give its time complexity ($O(n), O(n^2)$ etc) in its *docstring*[3].

---

2 Possible extra job?

Any of the following task will give you extra marks[4].

[2] The result to use has been seen in MA102, and is also on Apostol, Vol.II, page FIXME FIXME.
[3] A good example of such practice is the solution for Problem III for CS second mid term exam (given on Moodle).
[4] It is not possible to aim the best mark without trying any of these...

1. Implement the GRAM-SCHMIDT process[5] for the usual dot product (which can be defined with `def dot(vx, vy): return sum([ x * y for x,y in zip(xv,xy)])`). The family of n vectors $(x_1, \dots, x_n)$ of $\mathbb{R}^n$ will be given as a matrix A of size (n, n). You have to chose and specify if the (x_k) are rows or columns of this matrix.

2. Look for any method to solve a (square or not) linear system $A.x = b$ (e.g. CRAMER formula, or GAUSS elimination). Chose one algorithm and implement[6] it.

3. After having implemented the GAUSS elimination algorithm, try to also write the **Gauss-Jordan elimination** (which goes further, to obtain a *reduced* row echelon form).

4. The GAUSS-JORDAN elimination can then be used[7] to compute A^{-1} when A is invertible (ie. non-singular). Use this algorithm to implement the `A.inv()` method, and to compute negative integer power : $A^{-k} = \left(A^{-1}\right)^k$ (you should update the `__pow__` method for the class `Matrix`).

5. For all your functions and methods, you can take the time to check that the given arguments are valid. For example, n, m should always be non-negative. Hint: it is easy to perform such checks, by using the `assert` keyword. Example: `assert n > 0 and m > 0` check that $n > 0$ and $m > 0$.

2.1 Bonus: overloading the +, -, *, /, % operators to accept numbers also

If you want to try, by using a simple test `isinstance(B, Matrix)` (to know if `B` is a matrix or not), it is possible to generalize the `__add__`, and the other special methods, in order so that they can accept (float/int/complex) numbers in which case the operation is done coefficient-wise.

Example: `Matrix([[1, 2], [3, 4]]) + 2` is `Matrix([[3, 4], [5, 6]])`.

2.2 Bonus: eigen values and vectors (Harder!)

- Try to implement an algorithm to compute the spectrum (ie. set of eigen values) for a square matrix.
- Is it reasonable to try to obtain an *exact* solution in the general case (for any n, and not just $n < 5$) ?
- Similarly, try to find and implement an algorithm to find eigen vectors.

Hint: this bonus question is quite hard, do not spend too much time on it.

---

3 Extra advices

- Each function or method should be correctly implemented. The *correctness* of everything you write *is crucial* (the required examples are in fact here to help you detect any issue).
- Be careful about the special cases (empty lists, division by zero, integer division etc).
- Do not cheat from your friends or from Internet.
- Apply the rule "the more efficient, the better" for each function/method you write. Do not try to optimize each line, what is important is the *overall complexity* of the algorithms (e.g. $O(\log(k)n^3)$ is better than $O(kn^3)$). You can take some ideas from the exams or the labs.
- But keep in mind that *readability and simplicity of your programs are important*!
- Please contact me if needed (`CS101@crans.org` or `Lilian.Besson@ens-cachan.fr`).

---

[5] Reference is page 22 of APOSTOL, Vol.II, Theorem 1.13, or on Wikipedia.
[6] Its correctness will be **more important** than its complexity.
[7] This result has also been seen in MA102, and is in paragraph 2.19 of APOSTOL Vol.II, page 66 – 67.

4 Linear algebra packages in Python

Of course, all this has already been written in Python. The goal is not to use these packages, but to implement this `Matrix` class, and all these algorithms by yourself.

If you are curious, you can compare your results with the ones obtained with one of the following package.

4.1 With NumPy

docs.scipy.org/doc/numpy/reference/routines.linalg.html for linear algebra operations with the NumPy module (Numerical Python).

4.2 With SymPy

docs.sympy.org/latest/modules/matrices/matrices.html#linear-algebra for linear algebra operations with the SymPy module (Symbolical Python).

Warning: Do not use these modules to cheat, we will see it. **You have to implement everything by yourself.**