Exercise 2

The purpose of these exercises is to study numerically different methods for QR factorization.
Hand-in your results electronically latest Sept. 14, 2016, 24:00h.

This lab has 5 tasks.

Task 1

Write a program which applies Gram–Schmidt orthogonalization to a given \(m \times n\) matrix \(A\) \((m \geq n)\). It should return \(n\) orthogonal basis vectors of range(A). (Note to Python users: Write a class Orthogonalization which takes an \(n \times m\) array for instantiation. Give this class a method gramschmidt. Later this class will get more methods.)

Task 2

Choose some random matrices and test your code. Measure the orthogonality of your matrix by a couple of different criteria:

- Is the 2-norm one?
- How big is the deviation of \(Q^TQ\) from the identity matrix? Measure this in the 2-norm.
- Compute the eigenvalues of \(Q^TQ\). What do you expect they should be for an orthogonal matrix?
- Compute the determinant

Make your tests with matrices \(A\) with increasing dimensions, e.g. \(m = n + 2\) and \(n = 1, 10, 100, 1000, 10000\). Report your result. (Python users can define all these tests by methods of the above mentioned class. Note even the command numpy.allclose).

Task 3

Use MATLAB’s (or scipy.linalg’s) command qr to compute an orthogonal
basis of range(A). Repeat the tests of Task 2. Is there a qualitative difference? What is meant by ”Gram-Schmidt is unstable”?

**Task 4**

Solve Exercise 7.4 on p. 55 of the course book.

**Task 5**

Solve a 500 × 500 linear equation system of your choice with

- Matlab’s backslash operator or Python’s method `scipy.linalg.solve`
- by QR factorization, see p. 54 in the course book

Measure the execution time for both approaches. In MATLAB this can be done by `tic` and `toc`. In Python you may use the IPython’s magic command `%timeit`. (See related hints in the lecture)