Numerical Methods, Fall 2022
Assignment 1 [Vector and matrix norms. Orthogonal matrices. NumPy]
Total: 60, Deadline: 7 Oct

SUGGESTED READING

- Lectures 1-3 of [1]
- Lectures 1-2 of [2]
- From Python to Numpy
- 100 Numpy exercises

EXERCISES

1. **(5)** Consider the matrix $H(v) = 1 - 2vv^*$, where v is a unit column vector. What is the rank of the matrix $H(v)$? Prove that its unitary.

2. **(5)** Prove the following inequalities and provide examples of x and A when they turn into equalities:

   - $\|x\|_2 \leq \sqrt{m}\,\|x\|_\infty$
   - $\|A\|_\infty \leq \sqrt{n}\,\|A\|_2$

   where x is a vector of m components and A is $m \times n$ matrix.

3. **(5)** Assuming u and v and m-vectors, consider the matrix $A = 1 + uv^*$ which is a rank-one perturbation of identity. Can it be singular? Assuming it is not, compute its inverse. You may look for it in a form of $A^{-1} = 1 + \alpha uv^*$ for some scalar α and evaluate α.

4. **(5)** Prove that for any unitary matrix U one has $\|UA\|_F = \|AU\|_F = \|A\|_F$.

5. **(5)** Consider a 2-dimensional vector space $r = (x, y)$ and plot the unit disk $\|x\|_p \leq 1$ for $p = 1, 2, 3$ (use the matplotlib library).

6. **(15)** Consider a function mapping six tensors to one tensor: $Z\left(\lambda^{(1)}, \lambda^{(2)}, \lambda^{(3)}, \Gamma^{(1)}, \Gamma^{(2)}, U\right)$, with

   $$Z_{ahij} = \sum_{bcdefg} \lambda^{(1)}{}_{ab}\Gamma^{(1)}{}_{cbd}\lambda^{(2)}{}_{de}\Gamma^{(2)}{}_{feg}\lambda^{(3)}{}_{gh}U_{ijcf}.$$

   Assume than all indices of the tensors appearing above take values from 1 to χ. Running the numerical experiments, explore the values of χ in the range 3–50 (from slowest to fastest implementation).

   - In the notebook `convolution.ipynb` you may find implemented a *stupid* way to compute this convolution, which takes $\chi^4 \times \chi^6 = \chi^{10}$ flops. In fact, this can be computed much faster!
   - Using the function `numpy.einsum` (its crucial to use the `optimize` argument), you can actually achieve a much faster implementation. In order to understand what it is doing under the hood, explore the function `numpy.einsum_path`. What is the minimal number of flops required for computation of Z?
   - Using the understanding of the output of `numpy.einsum_path`, implement an algorithm to compute Z, which is as effective as `numpy.einsum`, but relying only on more elementary `numpy.dot` and `numpy.tensor_dot`.

7. **(10)** In this exercise your goal will be to study and speed-up an implementation of K-means algorithm. In the notebook `kmeans.ipynb`, you can find a naive implementation. Explore the code, make sure you understand it. You will find there two functions `dist_i` and `dist_ij` which are (on purpose) implemented in a rather inefficient way. Improve them by getting rid of the loops in the favor of a proper numpy vectorized implementation and measure the speed-up of the full algorithm for $N = 10000$.

8. **(10)** Some things just can not be vectorized but still can be speed-up compared to naive implementation. For example, consider computation of the Hofstadter-Conway sequence $a(n)$ such that $a(1) = 1$, $a(2) = 1$ and

$$a(n) = a(a(n-1)) + a(n - a(n-1)), \quad n > 2$$

Write three functions, computing the sequence up to n-th element in three ways: i) pre-allocating `numpy` array and filling it using `for` loop, ii) cumulatively appending `python` list and converting it to `numpy` array, iii) same as i) but compiled (`jit`) version. Time the resulting implementations and conclude which is preferable. With the optimal one, compute $a(10^8)$.

REFERENCES

[1] Lloyd N Trefethen and David Bau III. *Numerical linear algebra.* Vol. 50. Siam, 1997.

[2] Eugene E Tyrtyshnikov. *A brief introduction to numerical analysis.* Springer Science & Business Media, 2012.