| | | |
|---|---|---|
| **Upload Date:** | `[Moodle Assignments]` | March 21th, 2023 |
| **Hand-In Deadline:** | `[Moodle Assignments]` | March 27th, 2023, 23:59:59 |
| **Correction Session:** | `[VU class]` | March 28th, 2023 |

4 **Timing numpy. (8P)** Measure and compare the time it takes for the numpy-implementations of array- and matrix operations and your own custom operations on lists. When you run your program, it should take care of the following tasks:

1. Implement custom linear algebra functions, operating on Python lists as vectors:
   - a 3D cross product
   - a scalar product
   - an element by element product of two vectors
   - sum of two vectors
   - a matrix-matrix multiplication (expressing matrices as list of lists)
2. Check the correct functionality of all your functions with some small **random generated lists** (ideally with simple $3 \times 3$ matrices and arrays of length 3).
   Print as a proof of correctness the results compared to the `numpy`-internal methods, compare the results using the `numpy.allclose` function .
   **Hint:** Look at how lists can be transformed into arrays and vice-versa.
3. For a set of array lengths $\{10^3, 10^4, 10^5, 10^6\}$ and $n \times n$-matrices with $n = \{30, 50, 70, 90\}$, fill the given arrays and matrices with random numbers between 0 and 1. **Hint:** Look into `numpy.array` and `numpy.random`.
4. Compare the performance of your own custom functions (all except the cross product) to the performance of the same numpy built-in functionality when operating on the previously defined set of arrays and matrices of different sizes. Make sure to run these operations several times repeatedly, and then average over the complete time. **Hint:** Import the `time` library or use the %`timeit` command in jupyter. See Fig.1 for a *possible* visualization.

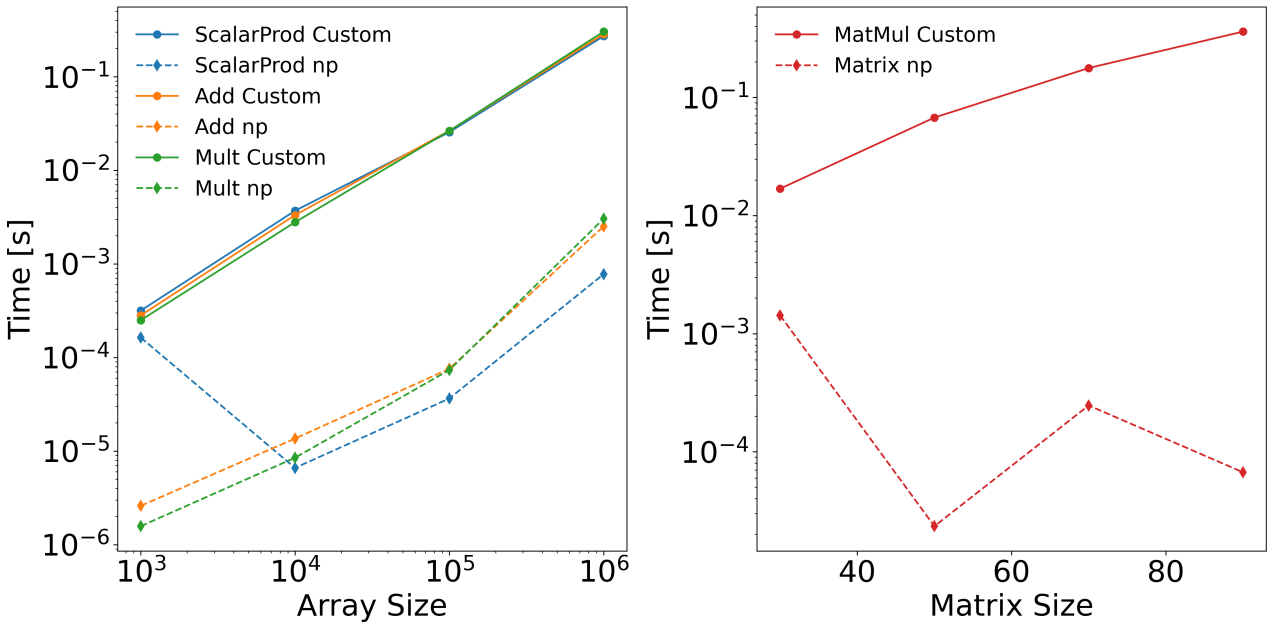


Figure 1: Possible visualization of the performance comparison of native Python vs. numpy.

5 **Principal Component Analysis. (6P)** Download the file `ex5_pca.xyz`. In it, you will find the configuration of a ring polymer. It is probably helpful to look into `numpy` for this exercise. Take care of the following tasks:

1. Read in the bead coordinates from the `.xyz` file. `.xyz` files are typically structured in the following way:

   ```
   [bead number]
   [molecule name and information]
   [type bead 0] [coordinates bead 0]
   [type bead 1] [coordinates bead 1]
   ...
   ```

2. Calculate $\vec{r}_{CM}$ (the polymer's center-of-mass), all the beads have the same mass.
3. Compute and print the gyration tensor of the polymer ring:
   $\hat{G}_{\alpha\beta} = 1/N \cdot \sum_{i=1}^{N} \vec{s}_{i,\alpha} \cdot \vec{s}_{i,\beta}$ (where $\vec{s}_{i,\alpha} = (\vec{r}_i - \vec{r}_{CM})_\alpha$ )
4. Compute the polymer's radius of gyration $R_g$ (the sum of diagonal elements of $\hat{G}$ is $R_g^2$) and print it.
5. Use `numpy` and `numpy.linalg` to perform a principal component analysis on the polymer ring, and print Eigenvalues and corresponding Eigenvectors from highest to lowest.
6. Write down in a Markdown cell a conclusion about the orientation of the ring in space. What can the Eigenvalues tell you about its overall shape? Visualizing the 3D coordinates of the polymer might help to draw conclusions, but is optional.

6 **Solving equation systems. (4P)** Download the file `ex6_eqSystem.dat`. The file includes four matrices and vectors in the following scheme: The first m columns belong to an $n \times m$ matrix A, the last column to an array B with $n$ elements. **Hint:** You may want to look into `numpy.linalg` for this task.

Optimize the equation system $A \cdot x = b$, performing the following tasks for all four equation systems:

1. Read in the configurations. **Hint:** Use `.rstrip("\n")` to get rid of trailing newlines. Also, all matrices have the same number of columns.
2. Compute and print the least-square solution $x$ for $A \cdot x = b$.
3. If the solution of the system is unique, the program should tell the user so by printing out a fitting statement.
4. Compute and print the Euclidean-2 norm of the system, i.e. $\| b - (A \cdot x) \|_2$.