FH University of Applied Sciences TECHNIKUM WIEN

Algorithms - Assignment

MME - Bioinformatics

Benjamin Ramberger

0. Naming the files

Please name your .py files in the following way and don't use special characters (like ö,ß,ñ,á):

assignment_assignment#_exercise#_lastname.py

e.g.: assignment _2_1_fuesschen.py for assignment 2, exercise 1 of student Füßchen

Submit one file for each exercise!

1. LU linear equation solver

Write a python function that solves a system of N linear equations with N variables, Ax=b. The main function `lu_solve(A,b,piv=True)` should take a [N,N] and [N,] `np.darray` with `dtype='float'` for `A` and `b` as input arguments and the optional `boolean` argument `piv` to turn pivoting on and off. It should return the solution vector x as a [N,] `np.darray` with `dtype='float'`. It should not have any side effects!

Start by writing auxiliary functions `Lsub(L,b)` and `Usub(U,y)` that solve a lower and an upper triangular system respectively. Import the auxiliary function that performs a LU factorization `lu(A,piv=True,Tol=10**(-7))` from the module `bioinf_lu.py` provided in Moodle. Use the three auxiliary functions in the function `lu_solve()` appropriately. Note that `lu` intentionally overwrites the input array – this has to be considered when designing `lu_solve()` without side effects.

Hint: You might find it easier to start your implementation for the case without pivoting. Once that works, you just need to rearrange the elements of `b` according to the permutation vector `permu` and leave the rest of your code as it is.

2. Quicksort

The basic idea of a quick sort is as follows (from Wikipedia, 30.11.2020):

1. Pick an element, called a pivot, from the array.
2. Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

Write a python function `myquicksort(list)` that performs a quick sort of a list of floats. It should return the sorted list, but should not have any side effects. Also write a function `check_my_sort()` that compares the performance of your quicksort with the one from numpy (np) - `np.sort()` - in a log-log plot. The function `check_my_sort()` should produce the log-log plot and print the complexity of both, `myquicksort(list)` and `np.sort()`.

Hint: For the `check_my_sort()` function you can use the function `check_my_sort()` of `bioinf_sort.py` provided on Moodle as a template.

3. Monte-Carlo

Write a function `mc_pi(N)` that returns an approximation to π, using a Monte-Carlo algorithm with a sample size of N. Write a function `mc_pi_stat(N,M)` that returns the mean μ and the standard deviation σ for `M` different π-approximations `mc_pi(N)`. Use the following numpy (np) functions:

np.mean(approx_pi)

np.std(approx_pi,ddof=1)

Finally, write a function `mc_pi_plt()` that illustrates the decreasing standard deviation. It should call `mc_pi_stat(N,10)` for different sample sizes N (e.g. 10^2,10^3,10^4,10^5,10^6) and produce a plot that shows (log[N],μ[N]) with error bars σ[N] and the "exact" solution (np.pi). Additionally, it should produce another plot that shows (log[N], log(σ[N])) and print the exponent α of the scaling behavior $\sigma \sim N^{\alpha}$.

Is α confirming your expectations? To reduce σ by 50%, what factor do you need to increase N by?

Example plots,using `matplotlib.pyplot.errorbar()` and `matplotlib.pyplot.plot()`: