ME535 Winter 2023: Homework 5

This is an individual assignment, so the answers you submit must be your own work. Before you upload the file to Canvas, execute the final version of the file to make sure that it runs properly and produces your intended answers. Insert your code to implement the functions as specified (without changing function names and/or arguments). Do not import from libraries other than those that are already imported into the template (such as numpy).

> Please DO NOT use this file as the basis for your homework submission.

> Add your code to `hw5_template.py` to create the file that you submit.

To this point, we have focused on developing your coding skills as a tool for developing and exploring algorithms. Now that you (hopefully) have some appreciation for what is going on "under the hood", it is time for further exploration of python packages/libraries that are made available by members of the python community.

1) Explore the docs for `scipy.integrate`. Your goal for this problem to use the library function `solve_ivp` to compute a numerical approximation of the solution $y(r)$ for the following initial value problem:

$$r^2 y'' + r y' + (r^2 - 1) y = 0; \quad y(0) = 0; y'(0) = 1$$

> If it seems confusing to have an independent variable named `r`, just replace `r` with `t` - but it really should not affect anything.

First enter code (in your version of `hw5_template.py`) to complete the function `ode_rates` that provides the rates (i.e. a list of the right-hand sides of the ODE as a first-order system) in a format consistent with the first input of `solve_ivp`.

Next, complete the code for `library_solve` that calls `solve_ivp` to compute the numerical solution and returns the solution (in the form provided by `solve_ivp`).

Once you have completed `ode_rates` and `library_solve`, execute the `p1` function and ensure that it produces a plot of the solution. Note that the independent values are specified to be `np.linspace(1e-15,15,151)`. Rhetorical question to consider: Why does the interval of r-values not start at zero?

Based on the plot of your numerical solution, estimate the location of the first 4 non-zero roots k_1, k_2, k_3, k_4 (where $y(r) = 0$) of your computed solution.

Based on your estimates, complete the code for `root_estimates` by inserting a string that states your estimates for $k_1, k_2, k_3, \text{ and } k_4$.

2) Explore the docs for `scipy.special`.

It turns out that the solution to problem 1 is a (fairly) well-known special function: the Bessel function of the first kind of order 1, a.k.a. $J_1(r)$.

Find the function that computes the Bessel function $J_1(r)$.

Fill in code to complete the function `evaluate_bessel` that returns an array of the special function evaluated on the same array of r values; i.e. `np.linspace(1e-15,15,151)`.

After completing `evaluate_bessel`, execute `p2()` and inspect the plots of the numerical solution, the special function, and the difference between them.

Then complete `compare_solutions` by entering a string with your response to the following:

"Compare and contrast the results from the ODE solver and the special function. Do the locations of the roots coincide with the estimates from the numerical solution of the ODE?"

3) (More on special functions and `scipy.integrate`...)

Find the function in `scipy.special` that returns the location of zeros of the Bessel function $J_1(r)$.

Fill in code for `precise_roots` that calls the appropriate library function to return an array of high-precision values for the first n non-zero roots of J_1.

Next, find the library function that performs numerical integration/quadrature, and fill in code to complete the following functions:

- `weighted_inner(ki,kj)` that calls a `scipy.integrate` library function to compute the weighted inner product

$$I(k_i, k_j) = \int_0^1 r J_1(k_i r) J_1(k_j r) dr$$

- `inner_array` that calls `weighted_inner` and computes the 2D array of inner products $I(k_i, k_j)$ for $i, j \in 1, 2, 3, 4$.

Describe the array returned by `inner_array` and comment on the implied relationship between the functions $J_1(k_i r)$.

4) Explore the libraries for random numbers and linear algebra.

Fill in the function `random_array` that creates an $n \times n$ array A of numbers randomly selected from the unit interval and returns the associated random symmetric array $B = \frac{1}{2}(A + A^T)$.

~~Fill in code for `test_qr` that calls random array to construct a test matrix, calls the library function to compute the QR decomposition of the symmetric test matrix B, and returns `Q,R`. The `p5()` function executes `test_qr(8)` to check if things are working as intended.~~

Fill in code for `qr_iter` that tests an idea for application of QR-factorization. `qr_iter` constructs a test array using `random_array` then repeatedly performs 2 steps:

(1) compute the QR factorization.

(2) replace the array QR with RQ.

The `qr_iter` function should return `Q,R,diag` corresponding to the QR factors and the 1D array of diagonal elements for the array obtained as a result of the iteration.

The `p5()` function tests your `qr_iter` implementation for 600 iterations on a 20×20 array. The output should include a `spy` visualization of the location of non-zero entries in your returned array R, and a graphical comparison of the diagonal elements to the eigenvalues.

Based on the plots, what is the outcome of the iterated QR-factorization scheme?

5) Learn about the optimization library `scipy.optmize` by visiting:

https://realpython.com/linear-programming-python/

Look through their explanation of and code for linear programming (i.e. optimization of linear functions subject to linear constraints). In particular, consider a modification of their Example 1 where the first (red) inequality constraint is changed from $2x + y \leq 20$ to $x + y \leq 10$.

Fill in code to complete `my_linprog(equality=True)` to return the solution of the revised problem.

When the argument is `equality=True`, the function returns the optimal result with the equality constraint.

When the argument is `equality=False`, the function returns the optimal result without the equality constraint.