Final Project

Due Thursday Dec. 19, 5 pm

The project will be done in groups of three, with students assigned randomly.

The project, when split amongst the group members, is not intended to be a huge undertaking. The goals of the project are to give you experience in working collaboratively and developing a well-designed, well-tested piece of software.

The project will be graded as a letter grade and will count for about as much as two problem sets in your final grade.

Please use standard citation practices to cite any papers/code/online resources/chatbots whose ideas you make use of. Please do not consult with any people (in particular class members) who are not in your group. You are of course welcome to ask Chris or João questions.

Problem

Your task is to implement an adaptive-rejection sampler, described in the Unit 9 notes (Section 5) and with details in Section 2.2 of Gilks et al. (1992) - the PDF is in the `project` directory of the class Git repository. The result should be an Python package that I can **easily** install and use.

My grading will be largely based on the following items.

1. Your solution should allow the user to provide reasonable inputs, including the number of values to sample, and should check the inputs for validity. The primary input should be a Python function that calculates the (possibly unnormalized) density of the distribution of interest in a vectorized fashion (e.g., the pdf functions in scipy). Your code should include numerical checks that catch cases of non-log-concave densities as the calculations proceed. (I.e., you do not need to come up with an overall test of the input density, but you should be able to do some checks that will catch cases where the upper and lower bounds are not actually bounding the density.)
2. Formal testing is required with a set of tests where samples are compared to the true distribution. Given the overall output is stochastic, how to do this will require some thought. Note that an important part of the grade will depend on how your code performs on tests that I have prepared, so think broadly about the various kinds of densities a user might provide. You should also have unit tests for non-trivial individual functions that carry out the individual computations that make up the algorithm.

3. Your solution should involve modular code, with functions or OOP methods that implement discrete tasks. You should have an overall design and style that is consistent across the components, in terms of functions vs. OOP methods, naming of functions/objects, etc.
4. In terms of efficiency, the algorithm is inherently sequential. However, you should try to vectorize as much as possible and consider approaches that allow you to generate multiple points at once.
5. Your solution should include clear, complete documentation, discussed further in the next section.
6. You should start by using numerical differentiation to estimate the gradients you need. But for an "A" solution, I'd like to see the capability to use automatic differentiation (AD) via JAX or PyTorch. Often we'd use JAX/PyTorch on a GPU, but here we'll just use them for their AD capabilities. There is no benefit here from using the GPU as we can't set up this problem to do a large number of identical calculations at once.
7. Your code can use any scipy/numpy/jax/pytorch calls you want as building blocks, but otherwise should not use or mimic any external code that implements adaptive rejection sampling.

Formatting requirements and additional information

1. Your solution to the problem should have two parts:
   a. A Python package named `ars`, which can be as simple as a directory named `ars` with .py modules and an `__init__.py`. (If you'd like to get experience with creating a package that could be installed via `pip` or `conda install` you're welcome to explore that but it's not required.) The package should be made available to me via a private repository named `ars-dev` within the Berkeley GitHub account or github.com account of one of the project members. **Make sure to share the repository with me** (my username is `paciorek` in either `github.berkeley.edu` or `github.com`.)

      The package should include:

      i. A primary function called `ars` that carries out the sampling, including appropriate code comments.

      ii. Other supporting code in the same or additional files (please think about clear organization), including appropriate code comments.

      iii. Formal tests set up using `pytest` and included in the package. I should be able to easily run these tests.

      iv. Help information for the main function, in the form of a standard Python docstring for `ars`. As part of this, you should have working examples in the example section of the docstring. You do not need extensive docstrings for your auxiliary functions but there should be a brief docstring just stating what each function does.

   b. A PDF document describing your solution, provided as a Quarto document (you're welcome to develop this using a Jupyter notebook and then convert to qmd). The description does not need to be more than 2-4 pages, but it should summarize the approach you took in terms of functions/modularity/object-oriented programming, the testing that you carried out, and the results of applying the implementation to various examples. It must include a paragraph

describing the specific contributions of each team member and which person/people were responsible for each component of the work. Please submit a paper copy of the document to me - either directly to me, under my door, or in my mailbox. **On your paper solution, please indicate the URL of the GitHub repository for the project.**

2. You should start the process by mapping out the modular components you need to write and how they will fit together, as well as what the primary function will do. After one person writes a component, another person on the team should test it and, with the original coder, improve it. You could also consider using pair programming for some of your development.

3. You should use Git and GitHub to manage your collaboration.