Homework #2

CS 169/268 Optimization
Fall 2024
Due: Monday October 21 11:59pm on Canvas

Reading:

Undergrads:

Kochenderfer and Wheeler, Chapter 4 Sections 4.1, 4.3 up through Algorithm 4.2, and sections 4.5-4.6; and Chapters 5 Sections 5.1-5.4, and 5.9-5.10.

Also Belegundu and Chandrupatla, Chapter 3: Sections 3.1-3.6.

Grads:

Kochenderfer and Wheeler, Chapters 4 and 5;

Also Belegundu and Chandrupatla (3rd ed.), Chapter 3: Sections 3.1-3.6.

Luenberger reading (Canvas) on Conjugate Gradients algorithm

Bertsekas on convergence: Sections 1.2, 1.3, and 2.1 .

Optional: Shewchuck on conjugate gradients.

Suggested review questions (study for midterm - don't turn in):

K&W 2.1, 2.2, 2.5; 3.1, 3.4 by hand+calculator; 3.5

Recall the Homework Ground Rules in HW0, point #1. In accordance with those rules, …

Remember, for each assignment you must submit a pdf report (ideally generated from your notebook file) to canvas, *and also*, for each assignment you must upload your (readable => commented) code on canvas. Please follow these guidelines unless otherwise stated.

Problem 1 (undergrads and grads) Descent methods for multi-variable unconstrained optimization.

1a. *Write*, or else *obtain, edit, instrument, and fully cite*, a functioning gradient descent optimizer, that is, an optimizer of real-valued functions $f(\mathbf{x})$ of one real-vector-valued argument $\mathbf{x}$ (containing many real-valued arguments x_i), using only function calls that evaluate $f(\mathbf{x})$ *and* its first derivatives $\partial f(\mathbf{x})/\partial \mathbf{x}$ as the descent direction, but no information about higher derivatives $\partial^2 f/\partial x^2$, etc., and no further constraints on the argument $\mathbf{x}$. Here "functioning" just means it always produces some numerical answer.

For gradeability, please use or edit the top-level optimizer itself as the Julia abstract type Descent Method in K&W algorithm 5.1, or else in Python using template provided at the end of this HW.

1b. As in 1a, except: use the method of conjugate gradients.

1c. Compare the performance of algorithms 1a and 1b on two 10-variable optimization problems One of your two problems should be quadratic, but the other should be taken or generalized from K&W Appendix B. For example, 10-dim Rosenbrock would work and is:

$$f(\mathbf{x}) = \sum_{i=1}^{9} [(a - x_i)^2 + b(x_{i+1} - x_i^2)^2] \ ,$$

with $a = 1, b \leq 5$ recommended.(You can use a similar strategy to turn a 2-variable quadratic objective function into a 10-variable one!) Do the comparison as in HW1 problem 1, reporting a table of estimated population means and estimated population standard deviations. You may need to change the numerical parameters of HW1 such as number of runs N per data point and the range of starting points; report the values you used. Describe the details of what you did. Explain your conclusion about the relative merits of the two programs as tested by you.

Problem 2 (grads only - extra credit for undergrads) Advanced descent methods for multi-variable unconstrained optimization.

Choose one of the advanced descent methods in K&W Chapter 5, section 5.4-5.9, to investigate by further systematic computer experiments. Implement it. Compare to the methods of 1a and/or 1b above. Report the results carefully, as usual.

Extra credit on either problem (up to 10% extra credit available on entire HW1):
Plot the results of 1(c) or 2(b) as you vary some important numerical parameter eg. governing stopping criterion, or function parameter b, or any other parameter you held fixed previously that you reasonably expect will affect the results. Plot points must have error bars but please explain whether they are errors on our knowledge of means, or estimated population standard deviations.

Julia: see problem 1a above for the template source.
Python template for gradeability, Problem 1:

```
def conjugate_gradients(func, x0, fprime, restart_frequency, x_tol = 0.0005, f_tol = 0.01):
    #----------------------
    #student code goes here:
    x_final = x0
    f_final = func(x0)
    CG_iterations = 1
    #----------------------
    return x_final, f_final, CG_iterations
def descent_method(func, x0, fprime):
    #----------------------
    #student code goes here:
    x_final = x0
    f_final = func(x0)
    #----------------------
    return x_final, f_final
# define test functions here
    print(conjugate_gradients(test_func, init_pt, test_fprime, 1))
    print(descent_method(test_func, init_pt, test_fprime))
```