IOE 511/MATH 562: Continuous Optimization Methods

Project Description – Winter 2023

Albert S. Berahas

“An investigation of the performance of unconstrained nonlinear optimization algorithms”

Project Overview

- The project for this course involves the implementation of a package of line search and trust region methods for solving unconstrained optimization problems.

- The coding exercises assigned throughout the semester will be part of the complete package.
- You will be required to implement more algorithms and options, allowing flexibility for users of your code.
- All requirements are described on the course website (`Project/Project.pdf`)

Coding Guidelines

- Upon completion of your Matlab software package, a user should be able to run any of your implemented algorithms with various sets of inputs using the following command:

```
[x,f]=optSolver_LastName_FirstName(problem,method,options)
```

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Methods

1. `GradientDescent`, with backtracking line search
2. `GradientDescentW`, with Wolfe line search
3. `Newton`, (modified Newton) with backtracking line search
4. `NewtonW`, (modified Newton) with Wolfe line search
5. `TRNewtonCG`, trust region Newton with CG subproblem solver
6. `TRSR1CG`, SR1 quasi-Newton with CG subproblem solver
7. `BFGS`, BFGS quasi-Newton with backtracking line search
8. `BFGSW`, BFGS quasi-Newton with Wolfe line search
9. `DFP`, DFP quasi-Newton with backtracking line search
10. `DFPW`, DFP quasi-Newton with Wolfe line search

Coding Guidelines

Options

- The minimal requirement for the `options` struct is an empty struct `options=[]`. All options could be set to default values.
- Your code should allow a user the specify the following options:
  1. `term_tol`: optimality tolerance
  2. `max_iterations`: iteration limit
  3. `c_1_ls` and `c_2_ls`: Armijo line search parameters
  4. `c_1_tr` and `c_2_tr`: Parameters for TR radius update
  5. `term_tol_CG`: CG optimality tolerance
  6. `max_iterations_CG`: CG iteration limit

Coding Requirements

- **It is a strict requirement that your code must be well-commented**.
- You should **augment** the output of your code to return the quantities required for investigating the performance of the algorithms.

Test Problems

- The problems that you are asked to solve with your various algorithms are posted on the Course Site (see `Project/problems.zip`).
- A description of each problem is provided in the function files themselves.
- You should use these files as test problems to check that your code is working correctly, but it is also a wise idea to create your own test problems so that your code solves more than these problems!

Project Deliverables

- The final deliverables for the project are a written report and your software package.
- There are two mandatory submissions as part of the project (Phase I and Phase II).
  - Phase I (due: Thursday March 9th, 20% of grade)
  - Phase II (due: Friday April 23rd, 80% of grade)

Project Deliverables–Phase I

- Form a team (2-3 members) and set a team name.
- Decide which **big question** you will investigate and describe how you intend to accomplish your goal(s). Be as precise as possible.
- Report should be short (2 paragraphs, less than 1 page).

Project Deliverables–Phase II

Report

- Summarize each algorithm in a few sentences each.
- Provide a table of default `options` for your code.
- Compare the performance of the algorithms on the given problems.
  - Provide a table ("Table: Summary of Results") of the numbers of iterations, function evaluations, gradient evaluations, and CPU seconds required to solve each of the four given test problems with each of your algorithms.
  - Compare the algorithms using any of the techniques studied in the class. (Function vs. iteration plots, Performance Profiles, etc.)
- If you had to choose one algorithm (that balances cost, convergence speed etc.), which algorithm would be your "algorithm of choice"? Discuss your decision.

Project Deliverables–Phase II

Report (continued)

- Investigate one *big question* thoroughly. (See below for examples.) The idea of this portion of the project is to investigate in depth one aspect of an (or several optimization algorithms). Examples of questions are:
  - Is the curvature condition important in a line search?
  - What are good choices of the line search parameters?
  - What are good choices of the trust region parameters?
  - How does memory affect the performance of quasi-Newton methods?
  - In limited memory quasi-Newton methods, which pair should be removed at every iteration?
  - How much modifying is too much in Newton's method?
  - Is there an optimal quasi-Newton method?
- You may choose to answer one of the questions above or any other question you see fit.
- Please discuss with the instructor and/or GSI if you have questions.

Project Deliverables–Phase II

Code

1. A complete software package as described above.
2. A single script that runs all experiments in "Table: Summary of Results".
3. A single script that runs your "algorithm of choice" on the Rosenbrock function, with your optimal choice of parameters.

Project Grading

- Do not be discouraged if you cannot get all of your algorithms to solve all problems. **It is better to code some of the algorithms correctly than to code all of them poorly.** Your grade for the project will be based on the merits of your (well-commented!) code as well as the clarity of presentation in your report.
- Project grades will be based on the code and your report.
- We will test your code on a "secret" set of problems to evaluate the overall performance. We will summarize the performance of everyone's "algorithm of choice" (using Performance Profiles) on the "secret" set of problems and send the results to the class.
- Bonus points: The 2 teams with the best reports and the 2 teams with the best performing "algorithms of choice" will receive a bonus in their project grades.

Some tips/suggestions for coding/debugging

- Comment your code!
- Printing output is one of the best ways to debug code
- Use unit test to check all components (no matter how small) of your code
- Start small and simple
- *Why are bugs/issues so hard to find?* Usually, bugs/issues are small/tiny (e.g., a plus sign should have been a minus sign), and hard to find since we do not expect such a simple part of the code to be wrong. From personal experience, bugs are either where you least expect to find them or right in front of you in the simplest operations of the code.
- It is very easy to make mistakes when coding derivatives; much easier than getting them right the first time (in my experience). As a debugging tool, you can use techniques from *Section 8.1, Numerical Optimization* to compute approximations of the derivatives (e.g., finite difference approximations to the derivatives) at different points and compare those to the results of your code.
- And, of course, comment your code!

Please ask the instructor and/or GSI if you have any questions about the expectations or anything else!

Finally, note that you are expected to work on the code and report for this project only with your project team. If there is any evidence of sharing code or writing in your report, then there will be serious unfortunate consequences, as this is a violation of the Honor code.