Project in Optimization 2023

Purpose

You should choose one of the following two alternatives:

> **Alternative 1**. *Nonlinear least-squares fitting using the Gauss-Newton method.*
> Files needed (on Canvas): `phi1.m, phi2.m, data1.m, data2.m, grad.m`
>
> **Alternative 2**. *Quasi-Newton methods with application to penalty and barrier problems.*
> Files needed: `grad.m, rosenbrock.m`

The purpose of both alternatives is to implement an optimization algorithm in MATLAB and analyze it with respect to performance. You may use Python. The project can be divided into three steps:

1. Implement the algorithm you have chosen. The block representing your choice of line search method should be implemented and tested separately. Pay attention to a smart structuring of the program that makes it easier for you to test different parts independently.
2. Test your program *very thoroughly* to ensure that it works correctly before you continue with step 3. Try to run many examples and initial points and find out whether the program behaves in an expected way that is consistent with the theory (for instance, fast/slow/no convergence, descent/ascent directions).
3. Use your program to solve the problems under **Tasks to do when your program works** for your chosen alternative and write a report.

Report

You should work in a group of three students and write one report per group. When you have a major question (debugging you do within the group!), email your assigned supervisor on the `Project administration page` in Canvas with some suggested time slots. You should have at least one meeting with your supervisor before you submit the report. One member of the group should upload your report in Canvas at the latest on **Monday 18 December 2023 at 08:00**. A corrected final report should be uploaded before the exam on 11 January. The next opportunity for consultancy and submission will be a week before the re-exam in April. *Any copy-paste similarities between reports will imply that both groups fail.* You are likely to be asked to correct your program and complete your report as well as to give oral explanations and clarifications of some details. The corrected report should be uploaded in Canvas by the same group member every time.

Recommendations and requirements for both alternatives

- Function evaluations may be very expensive in some problems where your code may be used in the future. Therefore, you should write your code so that the number of function calls is kept low. The CPU time is not of primary interest, though it should be within a reasonable bound (see MATLAB's `tic` and `toc` to calculate the time).
- Everything should be calculated numerically. *Do not use symbolic calculations in MATLAB* since they are extremely slow. To calculate the the gradient of a vector you may use the numerical differentiation in the file `grad.m`. As an alternative, you may submit the exact gradient of the objective function as an input to your program.

- Pay attention to MATLAB's warnings and fix the troubles. Learn and use MATLAB's built-in debugger to track variables, values, and function returns. It does not take much time to learn it, but it will save a lot of time for you in your search for errors. The most convenient way to do debugging is from MATLAB's editor graphical interface. Look for videos 'MATLAB debugging' on `www.youtube.com`.

- Your implementations must be written in such a way that there is no need to make any modifications by hand when running different tests or starting points.

- If a function is stored in the m-file `func.m` you can compute its value at a point `x` by `func(x)`. The same can be done via the function handle `my_f=@func` using `my_f(x)`. The function handle is a convenient way to construct new functions inside another function also when there are parameters; for example

```
a=4;
f=@(x)x^2+a/x;                 % e.g. f(2) = 6
g=@(x,y)f(x)+y(1)^2+y(2)^2;    % e.g. g(2,[1;2]) = 11
```

- Implement and test your line search subroutine separately. It is recommended to define `F = @(lambda) f(x+lambda*d);` in your main program so that you can send the scalar one-variable function `F` into your line search subroutine. Make sure that you do not have bad inputs to the line search subroutine, which always should return an acceptable value λ. While testing, you can put something like the following at the end of your line search subroutine:

```
if isnan(F(lambda)) || F(lambda)>F(0)
  error('Bad job of the line search!')
end
```

  Test the performance with several functions; for example, the quadratic functions

```
a=2;   % try a = -2 too, then a = 5 and -5, then a = 10 and -10
F=@(lambda) (1-10^a*lambda)^2;
```

  where the minimum value is equal to zero.

  Optional: Due to the limited bit calculations one can come across functional values that are `Inf` or `NaN`. Figure out how to make necessary modifications to your line search strategy and test it on the following functions:

```
F = @(lambda) (1e58*lambda-1)^100;
F = @(lambda) (1e-58*lambda-1)^2;
```

  The functional values in both tests should be $0 \leq F(\lambda_k) < 1$ (the smaller the better) and the search should be relatively fast (to compare with Stefan's: $F(\lambda_k) \approx 0.005$ found in 0.002 sec; see MATLAB's `tic` and `toc` to calculate the time).

- Nice printing of your results can be obtained by the command `fprintf`. To find out how it works, try the following sequence of commands (see also `help fprintf`):

```
a=1.234;
b=5.678;
c=1.111;
fprintf('%12.4f  %12.1f  %12.4f\n',a,b,c);
fprintf('%12.4f  %12.1f  %12.8f\n',a,b,c);
fprintf('%12.4f  %12.1f  %32.8f\n',a,b,c);
```

Alternative 1

The problem is to fit the function $\phi(\boldsymbol{x}; t_i) = x_1 e^{-x_2 t} + x_3 e^{-x_4 t}$ to a given set of data points (t_i, y_i), $i = 1, \ldots, m$, where $\boldsymbol{x} = (x_1, x_2, x_3, x_4)$ are parameters. This can be formulated as the least-squares problem

$$\underset{\boldsymbol{x}\in\mathbb{R}}{\text{minimize}}\ f(\boldsymbol{x}) \qquad \text{where} \qquad f(\boldsymbol{x}) = \sum_{i=1}^{m} \big(\phi(\boldsymbol{x}; t_i) - y_i\big)^2.$$

Write a MATLAB function

```
function [x,N_eval,N_iter,normg] = gaussnewton(phi,t,y,x0,tol,printout,plotout)
```

that solves this problem with the Gauss-Newton method. Here, `phi` denotes the function handle for ϕ, `t,y` are vectors of the given data, `x0` is the initial point chosen by the user and `tol` is a user-defined tolerance for termination. The parameters `printout` and `plotout` take the values 0 or 1. The output variables should be at least the approximate minimizer `x`, the number of function evaluations `N_eval`, the number of iterations `N_iter` and the norm of the gradient `normg` at the output point. For example, if we use the function stored in the m-file `phi2.m`, the data obtained in the file `data1.m` and the starting point $(1, 2, 3, 4)$, then the commands are

```
[t,y] = data1;
[x,N_eval,N_iter,normg] = gaussnewton(@phi2,t,y,[1;2;3;4],1e-4,1,1);
```

The tolerance is here chosen to 10^{-4} and printing and plotting will be performed. The printing on the screen could look like this (you are free to choose any reasonable data):

iter	x	max(abs(r))	norm(grad)	ls fun evals	lambda
⋮					
6	2.7811	0.1068	1.3234	1	1.0000
	1.3834				
	3.2169				
	3.0136				
7	1.8615	0.0775	1.4456	2	0.5000
	1.0950				
	4.1369				
	2.8589				

Recommendations

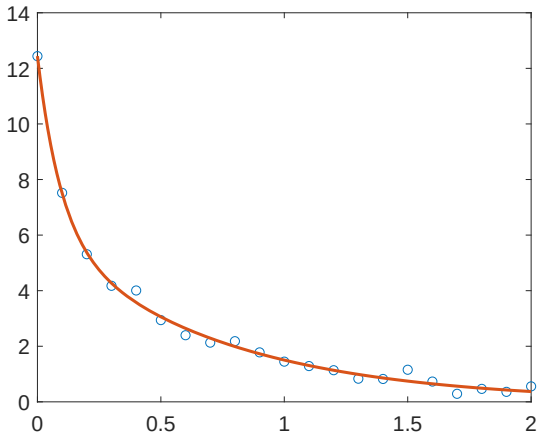
- The data vectors `t` and `y` are easily baked into the main function with function handles:

  ```
  r=@(x) phi(x,t)-y;
  f=@(x) r(x)'*r(x);
  ```

- You should calculate the search directions $\boldsymbol{d}_k$ in the nonlinear Gauss-Newton method without inverting any matrix! You should neither compute the approximate Hessian $2\boldsymbol{\mathcal{J}}^{\mathrm{T}}\boldsymbol{\mathcal{J}}$ nor use `inv` or `pinv`. The system of equation that defines $\boldsymbol{d}_k$ can be solved efficiently in MATLAB with the command \. MATLAB solves the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ with the least-squares method if $\boldsymbol{A}$ is not quadratic, i.e., the normal equations for the problem of minimizing $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|$. Type '`help \`' in MATLAB for more information. When the approximate Jacobian is not of full rank, MATLAB gives a warning but nevertheless delivers one of the possible solutions. To suppress such warnings, you can add the following line:

```
warning('off','MATLAB:RankDeficientMatrix')
```

- Here is an example of a plot of the `data1` and the optimal function $\phi(t)$:



Tasks to do when your program works

- Use your program to fit the function `phi1.m` to `data1.m` and `data2.m` for *many* different starting points.
- Do the same for `phi2.m` for both `data1.m` and `data2.m`. For the latter data, figure out a strategy on how a suitable initial point can be found.

Your report should be a pdf file containing the following:

- The names and personal identification numbers of all the group members and the name of the supervisor.
- A short description on who has done what in the project.
- A suitable title and an introduction readable for a student not taking the optimization course. The introduction should contain a short theoretical background about the method and what is investigated in the project.
- The optimal points in all four cases and the corresponding `max(abs(r))`.
- A motivation for your choices of line search method(s) and stop criterion.
- A table and short discussion on how the convergence (total number of iterations; tolerance) and the solution are affected by different initial points.
- A strategy to choose the initial point for `data2.m`.
- A full printout of a minimization with all information (starting point, tolerance, method, etc.) in the case `phi2.m` and `data2.m`
- Your program in an appendix.

Alternative 2

Write a MATLAB function

```
function [x,N_eval,N_iter,normg] = nonlinearmin(f,x0,method,tol,restart,printout)
```

that minimizes a function f with the DFP and BFGS quasi-Newton algorithms. Here, `f` denotes the function handle for the objective function, `x0` is the initial point chosen by the user, and `method` takes the values `'DFP'` or `'BFGS'`. The parameter `tol` is a user-defined tolerance for termination and the parameters `restart` and `printout` are given the value 1 or 0. The outputs should be at least the point `x`, the total number of function evaluations `N_eval`, the total number of iterations `N_iter` and the norm of the gradient `normg` at the output point. For example, to minimize the function stored in the m-file `func.m` with starting point $(1, 2, 3, 4)$, you may give the command

```
[x,N_eval,N_iter,normg] = nonlinearmin(@func,[1;2;3;4],'DFP',1e-4,1,1)
```

The DFP algorithm is chosen with the tolerance 10^{-4} and the results should be printed on the screen, for example, as:

iteration	x	f(x)	norm(grad)	ls fun evals	lambda
⋮					
6	2.7811	0.1068	1.3234	3	0.5
	1.3834				
	3.2169				
	3.0136				
7	1.8615	0.0775	1.4456	5	16
	1.0950				
	4.1369				
	2.8589				

Tasks to do when your program works

- Test the consistency of your program on functions where you know how the methods should work theoretically. Hint: You can generate a random matrix in MATLAB with `rand(n,n)`. How can you then form a positive definite matrix?
- Use your program to minimize the function `rosenbrock.m`. Try many different initial points for both methods and with or without restart.
- Use `nonlinearmin` as a subroutine in another MATLAB program to solve the following problem by using a penalty function and a sequence of increasing values of the penalty parameter:

  $$\text{(pen)} \qquad \text{minimize } e^{x_1x_2x_3x_4x_5} \qquad \text{subject to } \begin{cases} x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 = 10, \\ x_2x_3 = 5x_4x_5, \\ x_1^3 + x_3^3 = -1. \end{cases}$$

  A suitable initial point is $(-2, 2, 2, -1, -1)$. However, other initial points may give different results. Find them and explain what happens.

  Compare the two algorithms with different tolerances and with or without restart.

- Optional: Apply your program to Exercises 7.10 and 7.20 in the textbook. Use a suitable sequence of μ_k or ε_k to increase the accuracy of the answer.

Your report should be a pdf file containing the following:

- The names and personal identification numbers of all the group members and name of the supervisor.
- A short description on who has done what in the project.
- A suitable title and an introduction readable for a student not taking the optimization course. The introduction should contain a short theoretical background about the method and what is investigated in the project.
- The optimal point(s) and the optimal function value for each problem.
- A motivation for your choices of line search method(s) and stop criterion.
- A short discussion on the consistency in the program behaviour and how you have tested it.
- A table for each problem and a short discussion on how the convergence and the solution are affected by different initial points for a given tolerance.
- A comparison of the two quasi-Newton methods with/without restart.
- A strategy of choosing initial points and the sequence of penalty parameter values in problem (pen).
- Optional: A discussion on how satisfactorily your program solves Exercises 7.10 and 7.20.
- A printout of a minimization with all information (starting point, tolerance, method, etc.).
- Your program in an appendix.