Stochastic optimization algorithms 2016
Home problems, set 1

General instructions. READ CAREFULLY!

Problem set 1 consists of three parts. Problems 1.1 and 1.3 are mandatory, Problem 1.2 is voluntary (but check the requirements for the various grades on the web page). After solving the problems, collect your answers, your programs, and your report (see below) in *one* .zip file which, when opened, generates one main folder containing your report and additional subfolders for each problem (e.g. Problems 1.1 and 1.3) in the assignment. In problems that do not involve computer programming (e.g. Problem 1.2, in this case), no folder is needed.

You should provide a single report (for Problems 1.1-1.3, i.e. the whole set) in the form of a PDF file (Note: *Only* this format is accepted). In the case of analytical problems, make sure to include *all the relevant steps of the calculation* in your report, so that the calculations can be followed. Providing only the answer is *not* sufficient. Whenever possible, use symbolical calculations as far as possible, and introduce numerical values only when needed. You should write the report on a computer, preferably using LaTeX (see `www.miktex.org`). Scanned, handwritten pages are *not* allowed.

In *all* problems requiring programming, use Matlab. The *complete* Matlab program for the problem in question (i.e. all source files) must be handed in, collected in the folder(s) for the problem in question. In addition, clear instructions concerning how to run the programs *should* be given *in the report*. The information should, for example, specify which file is the main script for the problem in question. Do not include unnecessary (unused) Matlab files. It should *not* be necessary to edit the programs, move files etc. Programs that do not function or require editing to function will result in a deduction of points. Furthermore, when writing Matlab programs, you should make sure to follow the coding standard (available on the web page). You may, however, hardcode *parameters* (in your .m-files) (see, for example, the parameters hardcoded in the beginning of `FunctionOptimization.m` in the Matlab introduction).

The maximum number of points for problem set 1 is 10. Incorrect problems will be returned for correction *only* in cases where the mandatory requirements have not been met, so please make sure to check your solutions and programs carefully before e-mailing them to `mattias.wahde@chalmers.se`.

Make sure to keep copies of the files that you hand in!

You may, of course, discuss the problems with other students. However, each student *must* hand in his or her *own* solution. In obvious cases of plagiarism, points will be deducted from all students involved.

NOTE: Don't forget to write your name *and* civic registration number on the front page of the report! Good luck!

(Strict) deadline: 20160916, 23.59.59

Problem 1.1, 3p, Penalty method

In this problem, we shall use the penalty method (see pp. 30-33 in the course book) to find the minimum of the function

$$f(x_1, x_2) = (x_1 - 1)^2 + 2(x_2 - 2)^2, \tag{1}$$

subject to the constraint

$$g(x_1, x_2) = x_1^2 + x_2^2 - 1 \leq 0. \tag{2}$$

1. Define (and specify clearly, in your report, as a function of x_1, x_2, and μ) the function $f_{\mathrm{p}}(\mathbf{x}; \mu)$, consisting of the sum of $f(x_1, x_2)$ and the penalty term.

2. Next, compute (analytically) the gradient $\nabla f_{\mathrm{p}}(\mathbf{x}; \mu)$, and include it in your report.

3. Find and report the unconstrained minimum (i.e. for $\mu = 0$) of the function. This point will be used as the starting point for gradient descent.

4. Write a Matlab program for solving the unconstrained problem of finding the minimum of $f_{\mathrm{p}}(\mathbf{x}; \mu)$ using the method of gradient descent. Specifically, your program *must* contain

   (a) A main file `PenaltyMethod.m` (that calls the other functions, generates and prints output etc. etc.). This program should *not* require any input, i.e. to run it, one should only need to write `PenaltyMethod` in Matlab, without having to specify any input parameters. The necessary parameters may be hardcoded in `PenaltyMethod.m`; see also below.

   (b) A function (in a separate file, `GradientDescent.m`), which takes the starting point $\mathbf{x}_0$ (as a vector with two elements), the value of μ, the step length (for gradient descent) η, and a threshold T (see below) as input, and carries out gradient descent until the modulus of the gradient, $|\nabla f_{\mathrm{p}}(\mathbf{x}; \mu)|$, drops below the threshold T. Use the unconstrained minimum at the starting point; see above.

   (c) A function `Gradient` (in a separate file, `Gradient.m`) which takes as input the values of x_1, x_2, and μ, and returns the gradient of $f_{\mathrm{p}}(\mathbf{x}; \mu)$ (a vector with two elements). Note: You may hardcode the gradient in this method, i.e. you do not need to write a general method for finding the gradient. However, your method should make use of the analytical gradient, computed in Step 2 above. You should not use a numerical approximation of the gradient.

5. Run the program for a suitable sequence of μ values (which you may hard-code in `PenaltyMethod.m`). Select a suitable (small) value for the step length η, and specify it clearly, along with the sequence of μ values, in your report. Example of suitable parameter values: $\eta = 0.0001$, $T = 10^{-6}$, sequence of μ values: 1, 10, 100, 1000.

The output from the program should be a table with three columns, namely μ, x_1^*, and x_2^*. This table should be printed by the program *and* should also be given in your report. Specify the values of x_1^* and x_2^* with 3 decimals. Do *not* just print the raw Matlab output (with many decimals, for example) in your report! You should also check that your results are reasonable, i.e. that the sequence of points appears to be convergent.

Maximum number of points for this problem: 3p.
Maximum number of points if the problem must be returned for correction: 1p

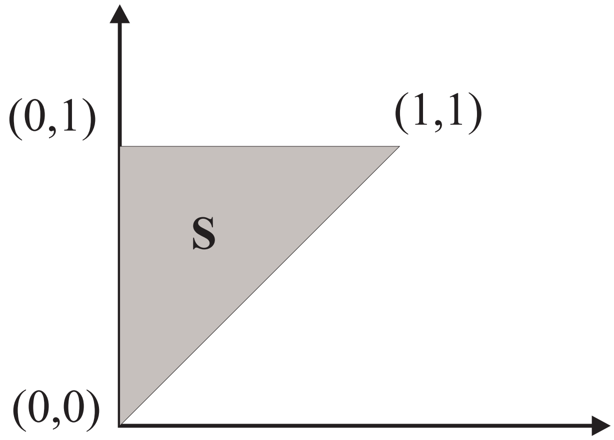

Figure 1: The set S used in Problem 1.2a.

Problem 1.2, 3p, Constrained optimization

a) (2p) Use the analytical method described on pp. 29-30 in the course book to determine the global minimum $(x_1^*, x_2^*)^{\mathrm{T}}$ (as well as the corresponding function value) of the function

$$f(x_1, x_2) = 4x_1^2 - x_1x_2 + 4x_2^2 - 6x_2, \tag{3}$$

on the (closed) set S, shown in the figure. The corners of the triangle are located at $(0, 0)$, $(0, 1)$ and $(1, 1)$.

b) (1p) Use the Lagrange multiplier method described on pp. 25-28 in the course book to determine the minimum $(x_1^*, x_2^*)^{\mathrm{T}}$ (as well as the corresponding function value) of the function $f(x_1, x_2) = 15 + 2x_1 + 3x_2$ subject to the constraint $h(x_1, x_2) = x_1^2 + x_1x_2 + x_2^2 - 21 = 0$.

Problem 1.3, 4p, Basic GA program

a) Write a standard genetic algorithm (GA) using (some of) the components described in Sect. 3.2.1 of the course book. You may start from the Matlab program written during the Matlab introduction, but note that the program needed for this problem is a bit different! In addition to writing the main program (`FunctionOptimization.m`), your task is to write Matlab functions (placed in *separate* M-files) for

1. initializing a population (`InitializePopulation`),
2. decoding a (binary) chromosome (`DecodeChromosome`),
3. evaluating an individual (`EvaluateIndividual`),
4. selecting individuals with tournament selection (`TournamentSelect`),
5. carrying out crossover (`Cross`),
6. carrying out mutations (`Mutate`).
7. carrying out elitism (`InsertBestIndividual`)

A version of each of these functions (except the one handling elitism, see below) has been implemented during the Matlab introduction. However, for this problem you will make (some of) the functions more general.

Matlab functions

In addition to the main program (called `FunctionOptimization.m`) you should write the functions specified below. Make sure to implement the functions exactly as described:

`InitializePopulation`: This function should take the population size and the number of genes as input, and should return the entire population as a matrix of binary numbers (i.e. as in the Matlab introduction).

`DecodeChromosome`: This function should take as input (i) a (binary) chromosome, (ii) the number of variables that are to be extracted, and (iii) the variable range. Let m denote the chromosome length and n the number of variables, and let $k = m/n$. The first k bits should be used when forming x_1, the next k bits should be used for generating x_2 etc. Each variable should be decoded from the k bits according to Eq. (3.9) in the course book. You may assume that m and n have been chosen such that k is an integer.

`EvaluateIndividual`: This function should take the vector of variables (`x`) as input, and should return the (note!) *fitness value* (which does not necessarily equal the function value; see 1.3b below!).

`TournamentSelect`: This function should take as input (i) the vector of fitness values (from the most recently evaluated population) (ii) the tournament selection parameter and (iii) the tournament size, and should return the index of the selected individual, using tournament selection. Note that the function should also handle cases where the tournament size is different from 2! See the description near the top of p. 50 in the course book.

`Cross`: This function should take two chromosomes as input, carry out single-point crossover, and return a chromosome pair (i.e. as in the Matlab introduction).

`Mutate`: The `Mutate` function should take as input (i) a chromosome and (ii) a mutation probability, and should return a mutated chromosome (i.e. as in the Matlab introduction).

`InsertBestIndividual`: This function should take as input (i) a population and (ii) the best individual in the most recently evaluated generation (which should be stored in connection with the evaluation of the population) and (iii) the number of copies n_c of the best individual that are to be inserted (normally one or two). The function should then insert the best individual in the n_c first positions in the population (replacing the individuals that have been placed there during selection, crossover, and mutation), and return the modified population.

After completing any Matlab function, you should always make sure to carry out a *unit test*, i.e. writing a simple wrapper that just provides suitable input to the function in question, and then make sure that the function generates correct output. Also, when writing the Matlab program, make sure to follow the coding standard (available on the web page). Submitting programs that deviate from the coding standard may result in a deduction of points.

b) Next, as a test of your GA, find (and report) the (global) *minimum* value of the function

$$\begin{aligned} g(x_1, x_2) &= \Big(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)\Big) \times \\ &\quad \Big(30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)\Big) \end{aligned} \tag{4}$$

in the interval $x_1, x_2 \in [-5, 5]$ as well as the location $(x_1^*, x_2^*)^{\mathrm{T}}$ of the minimum. Define the fitness function f as $1/g(x_1, x_2)$. At the *end* of a run (and only then), your program should print the minimum value found, as well as the corresponding variable values.

Since the goal of this problem is for you to familiarize yourselves with GAs, you *should* (for full points) try a number of different parameter settings (at least five), using the guidelines provided in Examples 3.5 - 3.7 in the course book. You may vary (between runs) the population size, the crossover probability, the mutation probability, the tournament size and the tournament selection parameter. For each tested parameter set, provide (in your report) a table listing the parameter values as well as the average results (over at least 20 runs) obtained for the parameter set in question. Note that the parameters should be kept fixed *during* any given run of the program. Use at least 25 genes (bits) *per variable* in the chromosomes. In the elitism step, make a single copy of the best individual (which should be inserted in the first position of the new population, as described above).

Check carefully that you enter the function $g(x_1, x_2)$ correctly. Hint: $g(2, 1) = 2275$.

c) Prove *analytically* (i.e. without the help of a computer!) that the point $(x_1^*, x_2^*)^{\mathrm{T}}$ you found in part b) actually is a stationary point of the function g. (You do not need to prove that it is a minimum). Make sure to include the relevant intermediate steps in your report, so that the calculation can be followed from beginning to end.

Maximum number of points for this problem: 4p.
Maximum number of points if the problem must be returned for correction: 2p