Stochastic optimization algorithms 2020
Home problems, set 2

General instructions. READ CAREFULLY!

Problem set 2 consists of four parts. Problems 2.1 and 2.2 are mandatory, the others voluntary (but check the requirements for the various grades on the web page). After solving the problems, collect your programs *and* your report (see below) in *one* zip file which, when extracted, makes the report available and also generates one main folder (which may contain additional subfolders, e.g. 2.1b/, 2.1c/ etc.) for each problem in the assignment.

You should provide a *single* report 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 symbolic 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. 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). Programs that do not function, or require editing to function, or deviate significantly from the coding standard, or otherwise do not follow the requirements given above, will result in a deduction of points.

The maximum number of points for problem set 2 is 15. 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 submitting the (single) compressed file (.zip or .7z) in Canvas.

You may, of course, discuss the problems with other students. However, each student *must* hand in his or her *own* solution. Note that a plagiarism check will be carried out, in which both your report and your code are checked against reports and code from other students (both from this year and earlier years). 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! Make sure to keep copies of the files that you hand in! Good luck!

(Strict) deadline: 20201014, 23.59.59

Problem 2.1, 5p, The traveling salesman problem (TSP) (mandatory)

The traveling salesman problem (TSP) has many applications in, for example, network routing and placement of components on circuit boards. In this problem, you will solve the TSP in a case where the cost function is simply taken as the length of the path.

a) In the TSP, paths that start in different cities but run through the cities in the same order are equivalent (in the sense that the path length is the same). Furthermore, paths that go through a given sequence of cities in opposite order are also equivalent. Thus, for example, in a 5-city case, the paths $(1, 2, 3, 4, 5)$ and $(2, 3, 4, 5, 1)$ are equivalent, as are $(1, 2, 3, 4, 5)$ and $(5, 4, 3, 2, 1)$. Paths that are *not* equivalent are called *distinct paths*. How many distinct paths are there in the general case of N cities? Show clearly how you arrive at your answer!

b) Write a GA (named `GA21b.m`) that can search for the shortest path between N cities, using permutation encoding (see p. 48 in the course book) for the path, i.e. an encoding method such that the chromosomes consist of lists of integers determining the indices of the cities (Hint: use the command `randperm` in Matlab to generate such chromosomes). Examples of five-city paths starting in city 4 are e.g. (4,3,1,2,5), (4,1,5,2,3), (4,5,1,2,3) etc. Thus, for example, the first of these chromosomes encodes the path $4 \to 3 \to 1 \to 2 \to 5 \to 4$. The fitness should be taken as the inverse of the path length (calculated using the ordinary cartesian distance measure, i.e. *not* the city-block distance measure). The program should always generate syntactically correct paths, i.e. paths in which each city is visited once and only once until (NOTE!), in the final step, the tour ends by a return to the starting city.

While the selection operator can (and should) be used just as in a standard GA, specialized operators for crossover and mutation are needed in order to ensure that the paths are syntactically correct. However, in this problem, we shall neglect crossover, thus using only selection and mutation. You may use either roulette-wheel selection or tournament selection. Mutations should be implemented as *swap mutations* that simply swap (the indices of) two cities in a given path. As usual, introduce a mutation probability p_{mut} and check each gene in the chromosome for mutation. If a gene is mutated, select another gene randomly, and swap the two alleles (see p. 147 in the course book).

Next, use your program to search for the shortest possible path between the cities whose coordinates are given in the file `LoadCityLocations.m` (available on the web page). This file contains a 50×2 matrix with the coordinates (x_i, y_i) for city i, $i = 1, \ldots, 50$. You should use the sample code (for plotting paths) available on the web page in the file `TSPGraphics.zip`. Whenever a new best path is found (and *only* for that case), it should be immediately visualized in the plot window, to keep the user informed about the progress of the program (this applies also to the other programs; see below). Thus, it is *not* sufficient just to plot the final path at the end of a run.

(The problem continues on the next page.)

c) Ant colony optimization (ACO) is especially suitable for solving routing problems of the kind described above. Following the description in Chapter 4, implement an Ant System (AS) algorithm (Algorithm 4.1), for solving the same problem as in **b)**.

In this case you *should* use the Matlab script `AntSystem.m` (available on the web page) as the starting point, and you should write the functions that are mentioned in that file (see the comments that include the words "To do"). The inputs to the functions *should* be *exactly* as specified in `AntSystem.m`. Note that, except for changing parameter settings and removing comments (so that the various functions are actually called), you should *not* modify `AntSystem.m`.

The most complex function to write is the `GeneratePath` function that, taking τ_{ij}, η_{ij}, α and β as inputs, should return the path for *one* ant (i.e. *not* all the paths at once). It is a good idea (for clarity) to use at least one additional function in `GeneratePath`, for example `GetNode` that returns the next node in the path, given the tabu list, the pheromone levels (τ_{ij}), the visibility matrix (η_{ij}), α, and β as inputs. Note also that the `GetVisibility` function should return the matrix η_{ij}, *not* η_{ij}^{β}. Similarly, the `pheromoneLevel` matrix should, of course, store τ_{ij}, *not* τ_{ij}^{α}. When updating pheromone levels, follow the equations in Algorithm 4.1 exactly.

d) As described in Chapter 6 (pp. 147-149), if the GA starts from an initial population generated completely randomly, it is at a disadvantage compared to ACO, since the latter easily finds the nearest-neighbour (NN) path. In order to demonstrate the importance of quickly finding the NN path, as ACO does, write a simple program `NNPathLengthCalculator.m` that, after loading the city coordinates, does the following: (1) selects a random starting city; (2) generates the NN path, starting from that city; (3) computes the length of this path (including the final step, such that the tour ends at the city of origin). Thus, for this part, all you need to do is to write (and hand in!) a program that computes the length of the path just mentioned, and then, in your report, compares its length with the length of the best path from the GA. You are allowed to use the function for computing the NN path length from (c) (see `AntSystem.m`).

e) Next, make a long run with each of the two algorithms (GA and ACO) described above. In your report, plot the shortest path found with each of the two algorithms and specify the length of each of these paths. Specify also the length of the nearest-neighbour path from part d). Furthermore, include the shortest path found (i.e. from one of the two algorithms, most likely ACO!) in a Matlab file named `BestResultFound.m` containing a vector called `bestPath` (with 50 elements, not 51!) with the city *indices* for the path in question. Note! Do not just copy from the output window in Matlab! Make sure that the saved vector is directly readable using Matlab. In particular, `BestResultFound.m` should not contain any instances of the `>` character. Note also that the indices *should* be in the interval $[1, 50]$, *not* $[0, 49]$, and that the path should contain exactly 50 elements (the return to the start city is implied, but the start city should *not* be appended at the end of the path). For example, a path may take the form

```
bestPath = [4 7 1 39 50 41 3 ...  etc.
```

The length of the path will be tested using the vector that you provide, assuming that city indices have been enumerated from 1 to 50. For full points on the problem, the length of the best path (obtained with one of your algorithms) should not exceed 123 length units.

Maximum number of points for this problem: 5p.
Maximum number of points if the problem must be returned for correction: 3p

Problem 2.2, 2p, Particle swarm optimization (mandatory)

In this problem, you will implement and use particle swarm optimization (PSO), which is a stochastic optimization method based on the properties of swarms, such as bird flocks, fish schools etc.

Start by implementing a standard PSO algorithm (as described in Chapter 5) in Matlab. Note: The standard PSO algorithm *should* include the (varying) inertia weight, see p. 128 and Eq. (5.20) in the course book! Remember to follow the coding standard and to place separate Matlab functions in separate files. Next, use the `contour` command in Matlab to determine the number of minima of the function

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2, \tag{1}$$

over the range $(x, y) \in [-5, 5]$. Hint: The function varies quite strongly with x and y. In order to make sure that you identify all the local minima from the contour plot, you may therefore instead plot the function $\log(a + f(x, y))$, where a is a small positive constant (e.g. 0.01). You should include the contour plot, with the minima clearly identified, in your report.

Next, use your PSO (named `PSO22.m`) to find the exact location of *all* the local minima (as well as the corresponding function values) of the function $f(x, y)$. In your report, you should provide a table with (x, y, f) for the minima. The values of x and y should be given with six decimal precision.

Since the PSO is stochastic, it will generally find different minima in different runs. Thus, you will need to run the PSO a number of times in order to find all the minima.

Maximum number of points for this problem: 2p.
Maximum number of points if the problem must be returned for correction: 1p

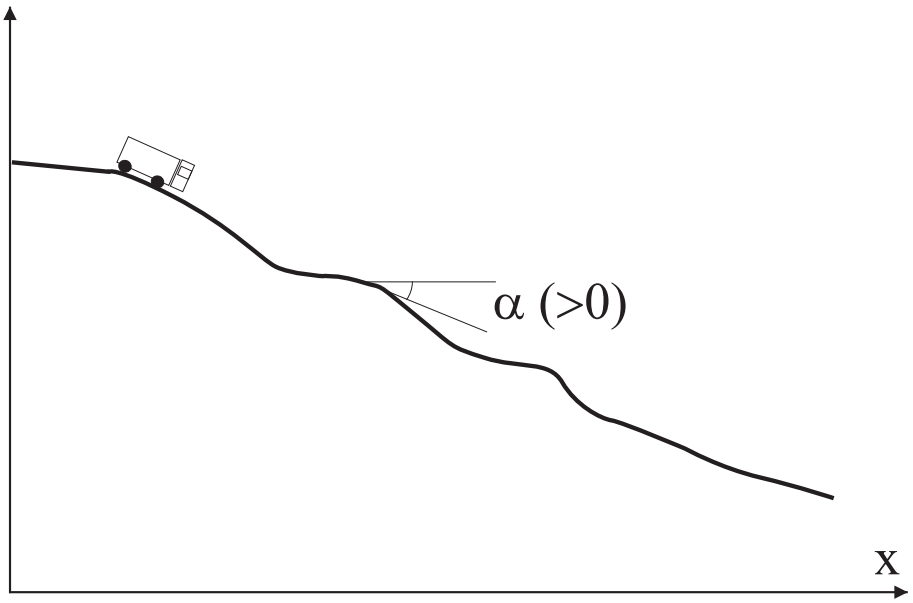

Figure 1: *A schematic illustration of a slope. x measures the horizontal distance from the origin, and $\alpha(x)$ is the instantaneous slope angle at x. Note that α is always positive for a downhill slope.*

Problem 2.3, 4p, Optimization of braking systems (voluntary)

In this problem you will develop an intelligent system for braking heavy-duty trucks during complicated descents (see pp. 83-86 in the course book). Such trucks are equipped with several braking systems, including the ordinary pedal brakes (also called foundation brakes), engine brakes, retarder etc. In order for the (foundation) brakes to work properly, the brake discs must never be overheated. Here, we will use a strongly simplified model of a truck, which will now be described.

Truck model

The truck is described by the following equations. The acceleration ($\dot{v}$) is given by

$$M\dot{v} = F_{\mathrm{g}} - F_{\mathrm{b}} - F_{\mathrm{eb}}, \tag{2}$$

where M is the mass of the truck and F_{g} is the component of the force of gravity in the instantaneous direction of motion, i.e. $Mg\sin\alpha$, where $\alpha(>0)$ is the instantaneous downhill slope angle (see Fig. 1). F_{b} is the force from the foundation brakes, and F_{eb} the force from the engine brakes. The braking force F_{b} is given by

$$F_{\mathrm{b}} = \begin{cases} \frac{Mg}{20} P_{\mathrm{p}} & \text{if } T_{\mathrm{b}} < T_{\max} - 100, \\ \frac{Mg}{20} P_{\mathrm{p}} \mathrm{e}^{-(T_{\mathrm{b}} - (T_{\max} - 100))/100} & \text{otherwise} \end{cases}, \tag{3}$$

where g is the constant of gravity (use SI units!). $P_{\mathrm{p}} \in [0, 1]$ denotes the pressure (applied by the driver) on the brake pedals. The brake temperature ΔT_{b} (relative to the ambient temperature) is assumed to vary according to

$$\frac{\mathrm{d}\Delta T_{\mathrm{b}}}{\mathrm{d}t} = \begin{cases} -\frac{\Delta T_{\mathrm{b}}}{\tau} & \text{if } P_{\mathrm{p}} < 0.01, \\ C_{\mathrm{h}} P_{\mathrm{p}} & \text{otherwise} \end{cases}, \tag{4}$$

where τ and C_{h} are constants. Note that the temperature in the brakes never can fall below the ambient temperature T_{amb}. Thus, the *actual* brake temperature equals $T_b = T_{\mathrm{amb}} + \Delta T_{\mathrm{b}}$.

Furthermore, it is assumed that the truck has 10 gears, and that the force from the engine brakes F_{eb} is given by

$$F_{\mathrm{eb}} = \begin{cases} 7.0C_{\mathrm{b}} & \text{gear 1} \\ 5.0C_{\mathrm{b}} & \text{gear 2} \\ 4.0C_{\mathrm{b}} & \text{gear 3} \\ 3.0C_{\mathrm{b}} & \text{gear 4} \\ 2.5C_{\mathrm{b}} & \text{gear 5} \\ 2.0C_{\mathrm{b}} & \text{gear 6} \\ 1.6C_{\mathrm{b}} & \text{gear 7} \\ 1.4C_{\mathrm{b}} & \text{gear 8} \\ 1.2C_{\mathrm{b}} & \text{gear 9} \\ C_{\mathrm{b}} & \text{gear 10} \end{cases}, \tag{5}$$

where C_{b} is a constant.

Write a program that implements the truck model described above. The program should be able to determine $v(t)$ by integrating Eq. (2) for any downhill slope (specified by $\alpha(x)$). Discretize the differential equations using first-order differences, e.g.

$$\dot{v} \approx \frac{v(t+\Delta t) - v(t)}{\Delta t}, \tag{6}$$

where Δt is the time step length, which should be smaller than the smallest relevant time scale in the problem. In this case Δt should not exceed 0.25 s, and even smaller values may be better.

Next, write a GA that optimizes (the weights of) a feedforward neural network (with a single hidden layer, with N_{h} hidden neurons) responsible for handling the braking system. The network should have three inputs, namely $v/v_{\max}$, $\alpha/\alpha_{\max}$ (see below), and $T_b/T_{\max}$, where $v_{\max}$, $\alpha_{\max}$, and $T_{\max}$ are constants, and two outputs, namely P_{p} and Δ_{gear}. Δ_{gear} denotes the gear choice, i.e. whether to increase or decrease the gear (by a single step, in both cases), or to leave the gear unchanged. The Δ_{gear} output signal can be encoded in various ways - you may select any suitable encoding. Note that Δ_{gear} is the *desired* gear change. However, it is not possible to change gears several times per second: The number of gear changes should be limited to *one* every two seconds (i.e. after changing gears, no further change can be made until at least two seconds later, regardless of the output from the neural network).

The optimization criterion is that the truck should manage to drive as fast as possible down a given slope, *without* ever violating the constraints regarding speed (should not exceed $v_{\max}$ or fall below $v_{\min}$) and brake temperature (should not exceed $T_{\max}$). If any constraint is violated, the evaluation (on that particular slope) should be terminated. A possible fitness measure for slope i is therefore

$$F_i = \overline{v}_i d_i, \tag{7}$$

where $\overline{v}_i$ is the average speed over the evaluation (before termination), and d_i is the distance travelled before termination. Thus, if the truck manages to cover the whole slope, $F_i = \overline{v}_i L$, where L is the length of the slope (see below). Note that other fitness measures are possible (and might work better). You may choose a suitable fitness measure yourself. The complete fitness value F for a given network over, say, the training set (see below) can be taken as either the average $\overline{F}$ over the slopes in the set or as the *worst* evaluation, i.e. $F = \min F_i$.

In order to train your network, generate three sets of slopes: A training set (10 slopes) a validation set (5 slopes) and a test set (5 slopes) and store them in a single file `GetSlopeAngle.m`, using the format and interface specified in the `GetSlopeAngle.m` on the web page. In other words, edit this file to generate your slopes.

Make all slopes $L = 1000$ m long (horizontal distance; see Fig. 1). α should vary with the horizontal distance x (i.e. it should *not* be constant), and should never exceed $\alpha_{\max}$, see below. Note the the function `GetSlopeAngle.m` should give the slope angle in *degrees*.

Next, train the network, using the method of holdout validation, described in Appendix C.2 in the book, i.e. measure both F^{tr} and F^{val}, use F^{tr} as feedback to the GA and F^{val} to determine when to stop the training etc.

Set the parameters according to $T_{\max} = 750$ (K), $M = 20000$ (kg), $\tau = 30$ (s), $C_{\text{h}} = 40$ (K/s), $T_{\text{amb}} = 283$ (K), $C_{\text{b}} = 3000$ (N), $v_{\max} = 25$ (m/s), $v_{\min} = 1$ (m/s), $\alpha_{\max} = 10$ (degrees). Assume that the truck starts at $x = 0$, with speed 20 m/s and with gear 7 in place. In the starting position, set $T_{\text{b}} = 500$ K. You may chose any suitable value (around 3-10, say) for the number of hidden neurons (N_{h}).

In your report, plot the results (i.e. the maximum fitness value in the population, as a function of the number of evaluated generations) obtained on the training and validation sets. Also, in addition to the optimization program, provide a *separate* test program (called `TestProgram.m`), which allows a user quickly to rerun your best network (i.e. the one with smallest validation error) on an arbitrary slope. This program should automatically load (or directly encode) and run your best network (once) on a given slope (e.g. the first slope in the validation set), without the user having to specify any inputs or otherwise modify the code. Moreover, when the truck *reaches the end* of the slope (or violates the constraints so that the simulation is stopped) this test program *should* generate a set of (sub-)plots (in one plot window) showing (i) the slope angle (α), (ii) the brake pedal pressure, (iii) the gear, (iv) the speed, and (v) the brake temperature, as functions of the horizontal distance travelled (x). Repeat: Your test program *must* generate the subplots just described. The plots should be shown once the truck reaches the end, i.e. not during the run.

Note! *Your braking system will be tested on a few test slopes generated as described above, and your score will be based on the performance of the system on these test slopes. Therefore, it is essential that you provide the test program, with clear running instructions. It should not be necessary to rerun the optimization procedure in order to test your best network!*

The implementation of the programs (the main program (with the GA) and the test program) is worth 2p, and the remaining 2p are awarded depending on the performance over our test slopes.

Problem 2.4, 4p, Function fitting using LGP (voluntary)

Consider a case in which a data series has been generated from a function of the form

$$g(x) = \frac{a_0 + a_1 x + a_2 x^2 + \ldots + a_p x^p}{b_0 + b_1 x + b_2 x^2 + \ldots + b_q x^q}. \tag{8}$$

The values of p and q, as well as the constants a_i and b_i are unknown, and should be inferred from the data series using LGP. Start by writing a general LGP program (called `LGP24.m`), with M variable registers, N constant registers, and the operator set $\{+, -, \times, /\}$. The program should evolve linear chromosomes using tournament selection, two-point crossover (see pp. 76-77 in the book), and mutations. The structure of the chromosomes should be of the kind illustrated in Fig. 3.19 in the book, i.e. such that each instruction is defined by four genes. The first gene (in a given instruction) should encode the operator and the second gene should encode the destination register. The third and fourth genes should encode the two operands. Note that crossover should occur between instructions, as shown in Fig. 3.21 in the book. The data set, consisting of values of the function $y = g(x)$ for various values of x, is contained in the file `LoadFunctionData.m`, available on the course web page.

The evaluation of a chromosome should be done as follows: For each data point (x_k, y_k), place the value of x_k in the first variable register (r_1), and set the contents of the other variable registers to 0. Next, execute the sequence of instructions contained in the chromosome, and take the final value contained in r_1 as the output, i.e. as the estimate $\hat{y}_k$ of y_k. When all data points have been considered, form the total error as

$$e = \sqrt{\frac{1}{K} \sum_{k=1}^{K} (\hat{y}_k - y_k)^2}, \tag{9}$$

where K is the number of data points. Finally, set the fitness value as $f = 1/e$. Note that the data is (unrealistically) noise-free. Thus, there is no risk of overfitting in this problem.

You may use as many variable registers and constant registers as you like (but you only need a few of each). Note that the values of the constant registers should be set once and for all, before the evaluation of chromosomes begins.

Run your program, and try to determine the function $g(x)$. In your report, the function should be specified *as in Eq. (8) above* (but with numerical values for all constants). Just providing the best chromosome is *not* sufficient. You should also specify, in the report, how many registers (variable and constant) were used, as well as the values chosen for the constant registers.

In addition, you *must* provide a separate test program (`TestLGPChromosome.m`) that loads the best LGP chromosome (which you must also provide, in a file named `BestChromosome.m`), then decodes and evaluates this chromosome, and then, finally, plots both the original data ($y_k,\ k = 1, \ldots, K$) and your best fit (i.e. the output $\hat{y}_k,\ k = 1, \ldots, K$ obtained from running the best LGP chromosome). Morever, this program *should* print out both the function $g(x)$ (which must be computed from the chromosome when the test program runs) and the error (see above) as text output.

Thus, you must write a Matlab function (for use in the test program only) that takes a chromosome as input and outputs the function $g(x)$ in the form given above. It is *not* allowed to use the Symbolic Math Toolbox in your LGP program, but you may use it in order to simplify your best function in the test program. Note that this can also be done via string operations. (The problem continues on the next page.)

Make sure to include the complete Matlab programs (LGP program and test program, see above) as well as the best chromosome.

Hint: In order to cope with variable-length chromosomes, you may wish to use the `struct` concept in Matlab, see e.g. the help files in Matlab (`>> help struct` etc.) and the simple example available on the web page.

You may also wish to use some of the more advanced EA operators in this problem, such as, for example, varying mutation rates. This is of course allowed. You may also introduce a multiplicative penalty factor (on the fitness values) for chromosomes whose length exceeds a given maximum $m_{\max}$. You must determine the value of $m_{\max}$ yourself (but it should not be below 100, corresponding to 25 instructions).

The implementation of the programs is worth 2p, and the remaining 2p are awarded depending on the quality of the fit. For full points, the error (described above) should be less than 0.01 (Ideally, of course, you should provide the exact function!). If the error exceeds 0.30, no points are given for the performance of the program.