Home Problem 1

Erik Norlin, 19970807-9299

October 12, 2022

Problem 2.1

An ant system algorithm (AS) was implemented in Matlab to find a path for the traveling salesman problem (TSP) shorter than 100 unit lengths for a set of 50 generated coordinates (cities). The shortest path found (see figure 1) using the program for this specific problem was 97.21163 unit lengths.

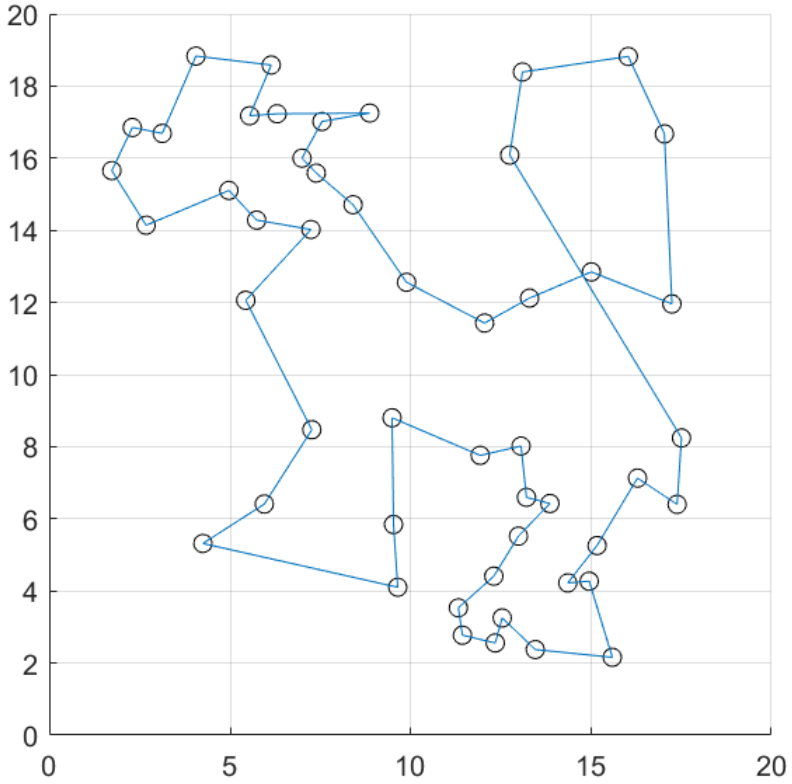

Figure 1: Plot of shortest path obtained by the Matlab program. The nodes represents the visited cities and the lines between them represents the path taken.

Problem 2.2

A particle swarm optimization algorithm (PSO) was implemented in Matlab to find all the minima of the objective function $f(x, y)$ over the range $(x, y) \in [-5, 5]$. The contour of the logarithm of $f(x, y)$, i.e. $log(a + f(x, y))$ where $a = 0.01$, was plotted (see figure 2) to clearer see how many minima there were over the given range. This was done because $f(x, y)$ varies a lot over the range and it can therefore be difficult to distinguish minima off the plot. Once all the number of minima could be identified over the given range the PSO algorithm was run until all the minima was found.

Table 1: The positions and function values of the found minima of $f(x, y)$ over the given range, obtained by implementing a PSO algorithm in Matlab.

x	y	$f(x, y)$
3.5844	-1.8481	0
3	2	0
-2.8051	3.1313	7.8886e-31
-3.7793	-3.2831	7.8886e-31

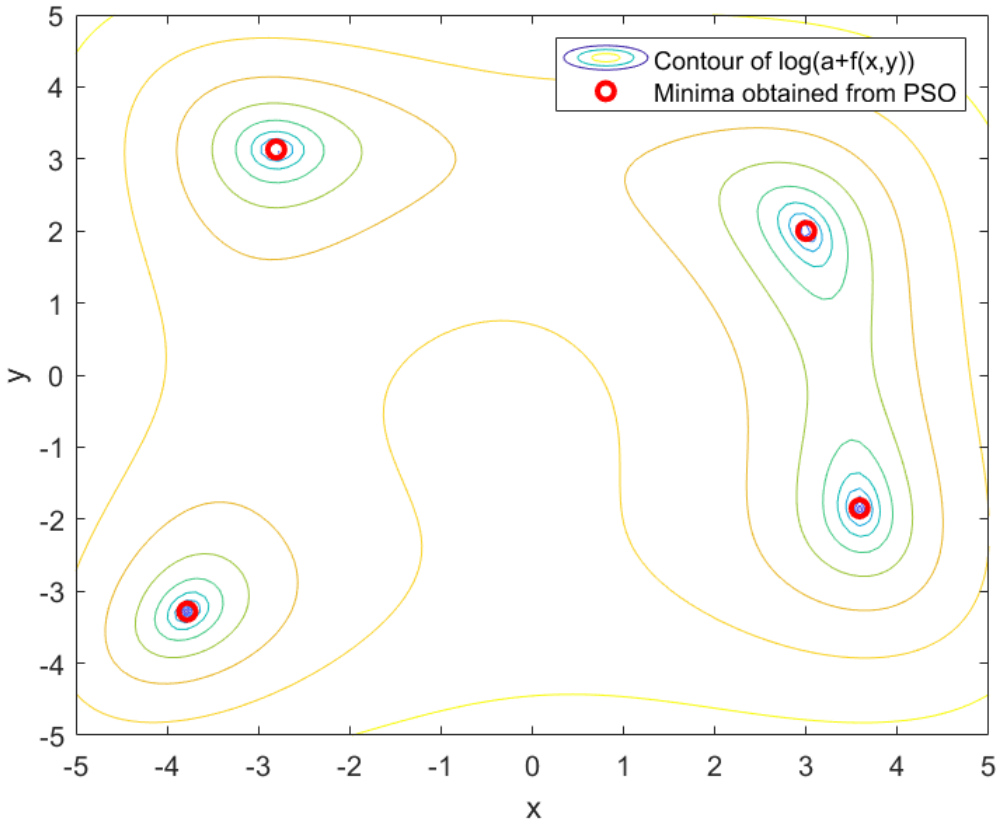

Figure 2: Plot of the contours of $log(a + f(x, y))$ as well as the found minima.

We can read from the plot that the PSO algorithm has found four out of four minima within the range and that the positions of the found minima aligns with where the minima should be based on the contours of the objective function.

Problem 2.3

A genetic algorithm (GA) was implemented to optimize a truck models' braking system when subjected to downhill slopes. The goal was to make the truck drive as fast as possible without ever overheating the pedal breaks due to the pressure from the brake pedal.

The GA was implemented to optimize the output of a feed forward neural network (FFNN) which consisted of two output neurons representing the pedal brake pressure and desired gear change at a given moment. Three input neurons and eight hidden neurons was used for this kind of network. One individual represented one FFNN and the population size was 10 FFNNs.

The choice for fitness measure from a given slope was the average speed of the truck times the horizontal distance travelled. The total fitness value of an individual was experimented with and the average fitness across all slopes was decided to be used. Another way one could measure the fitness of an individual could be to assign the worst fitness value from a given slope to the individuals' fitness value. This was tested and it turned out that all individuals turned out with the same fitness value because one particular slope at a particular place was the most difficult for the entire population. This made it difficult for the GA to make any progress.

Both fitness values for the training and validation set was measured for each individual every generation. The best performing network considered to be trained if the best performing validation fitness value hadn't gotten better within 200 generations.

The result (see figure 3) show that the maximal fitness values of both the training set and validation set steadily becomes better. Though, the best chromosome doesn't get very far before termination in the evaluation (around 100-200 m). One possible reason for this could be that the slopes in the validation set were too different from the slopes in the training set so that the FFNNs didn't learn the properties it needed to learn in order to perform better on the slopes of the validation set. Other reasons for this could be due to programming error, lack of understanding of the algorithm or misinterpretation of the information given in the task.

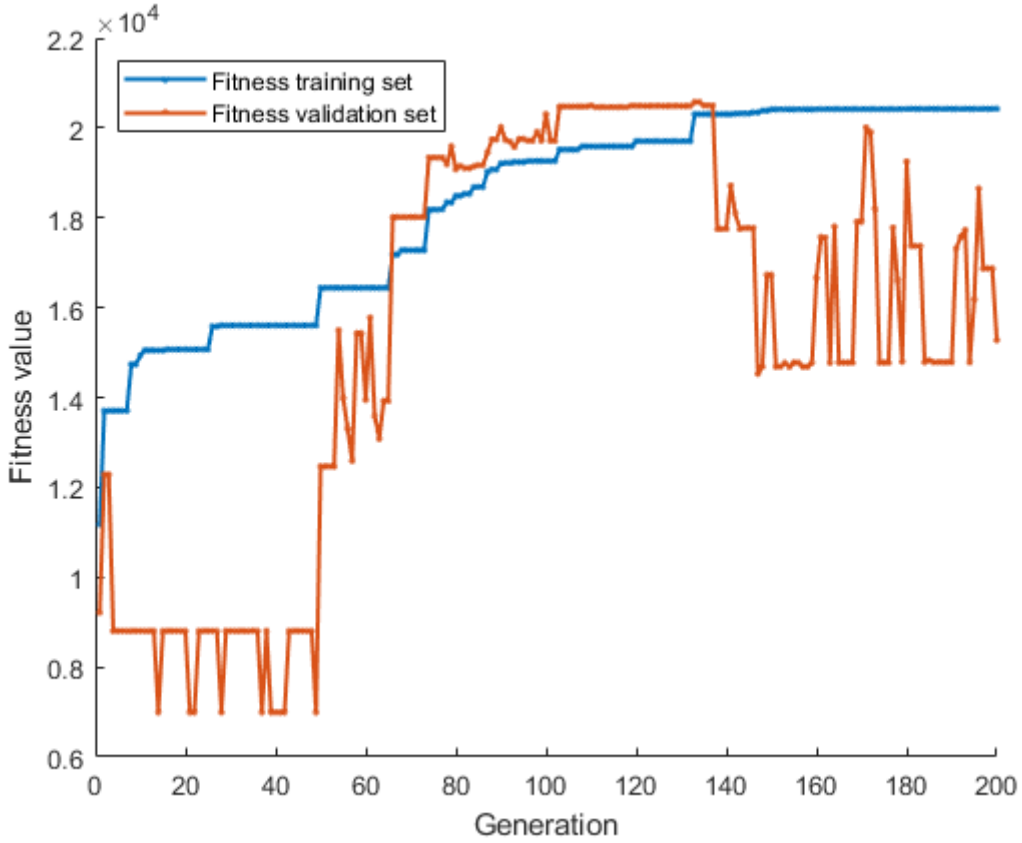

Figure 3: Plot of the maximal fitness values of the training and the validation set of each generation.

Problem 2.4

A linear genetic program (LGP) was implemented to estimate the objective function $g(x)$ given a data set of coordinate points from this function. The structure of the program was similar to a GA consisting of initialization, evaluation, elitism, crossover and mutation, where the difference lying mostly in evaluation, crossover and mutation.

The number of generations for this program was 3 million with a population size of 100 individuals (chromosomes). Three variable registers and four constant registers was chosen, the constant registers being $c_1 = 1, c_2 = 3, c_3 = -1, c_4 = 2$.

The population was initialized randomly as usually done in a GA. For evaluation, the operator set for decoding each instruction in a chromosome consisted of $\{+, -, *, /\}$. To avoid division by zero a protected definition was used so that if division by zero would occur the destination register would be assigned with $c_{max} = 10$ million to influence a low fitness value. $F_i = 1/e$ was used to calculate the fitness value of each individual, e being the root mean square error of the difference between the actual function value and the estimated function value for every point of the data set. Since the chromosomes wasn't length preserving a penalty was added to an individuals' fitness value if its' chromosome was too short or too long (25 respectively 150 genes). This was done to avoid endlessly long chromosomes which is computationally expensive to deal with. If the chromosome

wasn't within the given length a penalty of $1/c_{max}$ was assigned to the fitness value.

Tournament selection was used with a tournament size of 5 to pick candidates for two point cross over where the cross over points were random between instructions.

As for mutation a large mutation rate of 80 ($P_{mut} = (1/number\ of\ genes\ )$*80)was introduced in the beginning of the program to favour exploration of the genes. The mutation rate had a multiplicative decaying rate of 0.9999 which made P_{mut} equal to $1/number\ of\ genes$ (typical value for P_{mut} in a GA) after about 4000 generations which then favoured exploitation more.

The resulting approximation of $g(x)$ using LGP (see figure 4) after 3 million generations gave a root mean square error of 7.06%.

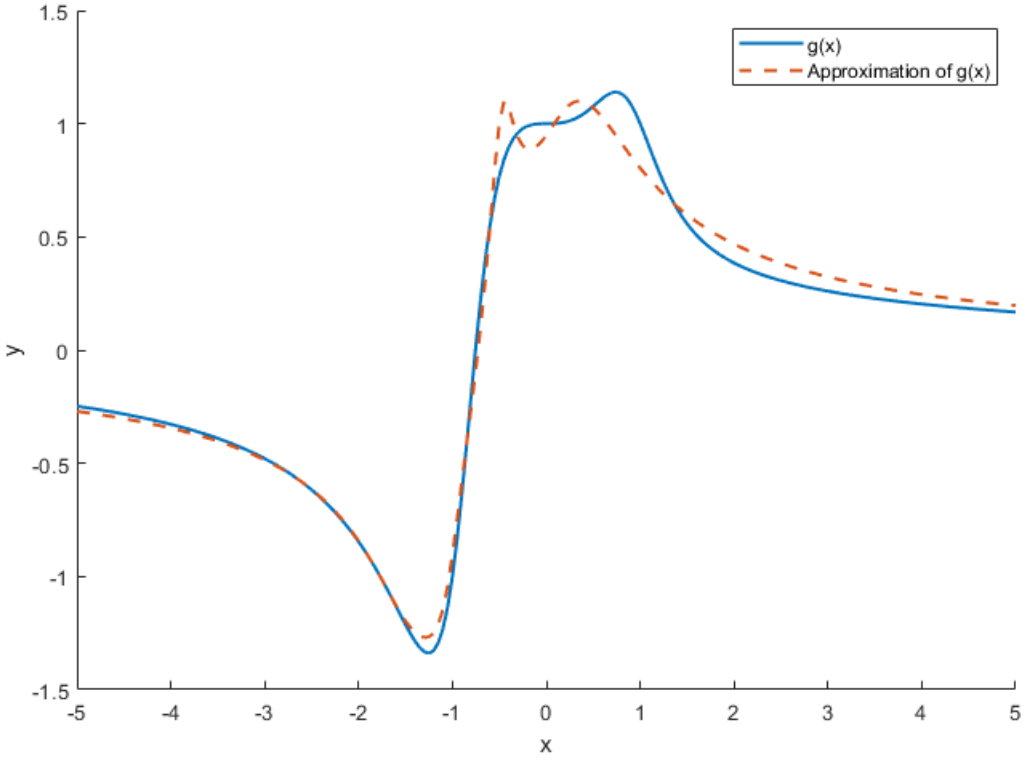

Figure 4: Plot of $g(x)$ as well as the approximation of $g(x)$ using LGP.

The expression of the estimated (and simplified) function of the approximation was incredibly long. This can have to do with that as the LGP approximates closer and closer to the real function the expression becomes more and more complicated. Though, at some point when the LGP converges to the function as the root square error goes to zero, the function expression should become simpler. The estimate of the function is:

$$g(x) = (891 * (72961692458749041 * x + 491213979658906680 * x^2 + 2137785036400432111 * x^3 + 6693739045135671144 * x^4 + 15916892982706524963 * x^5 + 29610142486161659576 * x^6 + 43776659495953699227 * x^7 + 51744893478742751760 * x^8 + 48798568446152576678 * x^9 + 36345453245299450560 * x^{10} + 20959126255203445848 * x^{11} + 9046050743511798016 * x^{12} + 2756494196987561184 * x^{13} + 530268858530780160 * x^{14} + 48607978698654848 * x^{15} + 5319391867626392) * (28586350864577509146 * x + 2061546967684936452491 * x^2 + 977156760952866771057 * x^3 + 339083315733607097631 0 * x^4 + 9089145863966338348211 * x^5 + 19367465306850619700049 * x^6 + 3329335045627818863905 5 * x^7 + 4643955886900994136711 7 * x^8 + 5251198855050863578845 3 * x^9 + 4780135800423834233248 8 * x^{10} + 3455912541218103909032 2 * x^{11} + 1941313424190206181297 6 * x^{12} + 8184400685559696183432 * x^{13} + 244582694668416256064 0 * x^{14} + 466335005093797159584 * x^{15} + 44330476573173221376 * x^{16} + 534687765685203328 * x^{17} + 1977770131909618611)^2)/ (82223069892404535193065470707959346208413464061376530476 6 * x + 17221702351178937401328119182986703135334861134102211404119 * x^2 + 23945037667194210196215844270622474081160235314290478171145 8 * x^3 + 24839295424982992023164432728248655331627803792259371131582 27 * x^4 + 20487316049544903946027973600978570115206281172792606192362068 * x^5 + 13983486764138426062719485725989837288010927268669841530955864 1 * x^6 + 81173755340453896925269272481775512964000703707559715879681066 8 * x^7 + 40879068854795402307081764237709839465807185205131051366445541 00 * x^8 + 18129048789980014046473938751863438322904542069092506984909378163 * x^9 + 71629847433010138957967243427578850109823397008292063636984705480 * x^{10} + 25449266680995477539390062502274103599969906972562962356775835752 6 * x^{11} + 81914449301867145659194297322642679560942713844132115484552753292 4 * x^{12} + 24032182764896280179125157741504718529399404782586674091617468785 17 * x^{13} + 64586483056592788665810533056015543537979499352968951085478497632 27 * x^{14} + 15965655225084360224782301641601507920211326418198454753602341346741 * x^{15} + 36423916410016579907685831961331844458983096720712556775163592368967 * x^{16} + 76899730811807343814390639545513846399760968223335114365535733445445 * x^{17} + 15057304670576026815795672148957775629049631333257500539145574660142 6 * x^{18} + 27389955956798103944748054482232206685963709399004995821095035648384 5 * x^{19} + 46345673327200476195603460531629368353062545620247313855725466009575 7 * x^{20} + 73010663153595312932514737418242294298284657948829820466245037828486 1 * x^{21} + 10714124132052167318763234395262247298109435875430521462783488490133 36 * x^{22} + 14649243505196282689691876551026511451450628735676772288765466417592 83 * x^{23} + 18660664763615075295285611674675483931691962525483746513989729409403 50 * x^{24} + 22137832688789038591891537072349633092410469534215825666253292119125 06 * x^{25} + 24443527149595795465009003623892735283214310708915176075234294472923 99 * x^{26} + \tag{1}$$

$$
\begin{aligned}
&250970215585202114822468216061248742151943138856956532680131238378661 9 * x^{27} + \\
&239332782420863812875277671808964013890604523711022518815466518130377 6 * x^{28} + \\
&211678103516914449468089815721496022751571142066073835075827856704013 0 * x^{29} + \\
&173335950782571864200990218713690274821749386441773716550294481639407 6 * x^{30} + \\
&131143995632370127009953188081880941717959229579216947141953847417563 6 * x^{31} + \\
&914558791690463319732493542771190172738448235648579616664151569542784 * x^{32} + \\
&586223664521908828687713877342759819655559716502950653131555811141576 * x^{33} + \\
&344263689513545880027745231289477102103048718007390718926793385935744 * x^{34} + \\
&184523338585003453870050029681927522842525138347342051219329863063584 * x^{35} + \\
&898717204866671921228186028135924100630733987365508413613713518597 12 * x^{36} + \\
&395685100876100441253166034678550848232870003483232125903126630259 20 * x^{37} + \\
&156515806624952713938590704820219404379295752471757636470972960194 56 * x^{38} + \\
&552141115281886635077389824219652524738012028658194466922951941836 8 * x^{39} + \\
&172166677607747854200856400655567218195230682629309548887799671193 6 * x^{40} + \\
&469335233667123206132127035944132167369689157734996645041264066560 * x^{41} + \\
&110321992364889095091269386997411840659619203262185523490916106240 * x^{42} + \\
&219676070593746574163464507567368132207007475579508940161925447 68 * x^{43} + \\
&361936841840963220188443536092373137701700748012171656830844928 0 * x^{44} + \\
&477677705747268479510554854868514186767796342370348964943560704 * x^{45} + \\
&481807030082419438851193108464037872944701564005964301179289 6 * x^{46} + \\
&345300487722710297730193497770232893491221381627977471361024 0 * x^{47} + \\
&155536709708488395441288638755878509781835049208378418003968 * x^{48} + \\
&355225395285386464127959749412704011310861054433657041715 2 * x^{49} + \\
&380210537347519874287976852714715668411410206449750507 52 * x^{50} + \\
&152862423641181236007593032889688316978271647329943552 * x^{51} + \\
&195659725923678583846427736083828948696927468054803002 03)
\end{aligned}
\tag{2}
$$