UP823599
UP824726

Neural Networks and Genetic Algorithms Coursework

824726 & 823599

Abstract: *This coursework aims to tackle 2 problems as outlined in Appendix A of this document concerning a Neural Network Classification problem and a Genetic Algorithm optimisation problem. The solutions contained within this document are created in MatLab and all data preprocessing was performed using Microsoft Excel and is briefly discussed in this document. All code created for these problems can be found in Appendix B of this document*

Keywords: *Neural Network, Genetic Algorithm, Deep Learning, Back Propagation, Incremental Learning*

I. Neural Network

A. Preprocessing Data

The initial format of the data for the Iris classification problem was a 150 record Excel spreadsheet with 4 columns for attributes of the irises (attribute data) and 1 column for the class of each iris (classification data). The attribute data was normalized by the formula in *fig. 1* then placed into a 4-dimensional matrix and transposing it. This was then stored in an *m-file* for later use. For the classification data a 3-dimensional matrix of 0's was used and the first, second or third column was set to 1 dependent on which classification the iris in the matching row was (e.g. [1 0 0] implies Setosa whereas [0 1 0] implies Versicolor).

$$n_{i,j} = \frac{X_{i,j}}{max(j)}$$

Fig. 1: Normalization formula where $n_{i,j}$ is the normalized value at matrix location i,j. $x_{i,j}$ is the value at matrix location i,j and *max(j)* is the largest value in column j

B. Neural Network Configuration

The neural network was configured using the Matlab Deep Learning Toolbox's *patternnet* function. This allowed for quick creation and easy configuration of a versatile neural network. The initial parameters for the network were arbitrarily decided based on research conducted online and from previous experience in this field (epochs = 200, transferFcn = 'hardlim') and used a random sampling process wherein the data set was split into training, validation and testing randomly. This produced an acceptable, albeit unrefined, neural network.

The split sample learning technique is widely adopted because of its ease of implementation and its high efficiency. This technique involves splitting the data into 3 subsets; training, validation and testing and this is typically done as a 80%-10%-10% split. The network trains on the training set while regularly evaluated to the validation set. Training stops when the training set has achieved the desired confidence level or the number of errors start to increase. The testing set is then used to see if the model can successfully generalise. The drawback of this technique is that the performance of the model can depend on the data selected for the different sets.

The cross validation learning technique uses more tests than the split sample technique to gain a better estimate of the model error and is best used when there is not enough data for the split sample technique. This technique involves splitting the data into equal sized subsets in which one is used for testing and the rest are used for training.

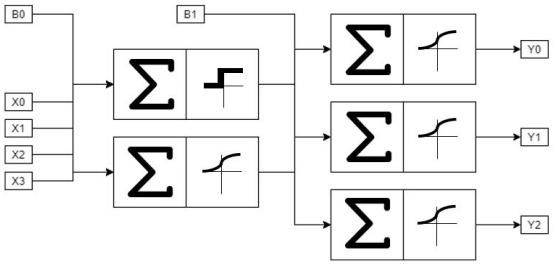

Fig. 2: Initial concept diagram of NN topology. We decided upon using heavy side step as the linear transfer function for the linear separation perceptron because we only want to know if the input matches setosa or not whereas

sigmoid functions allow for a range of outputs. The second perceptron uses binary sigmoid as we need continuous values as our output.

C. Neural Network Refinement

From here we attempted to change our topology using a different activation function and a second layer with a binary sigmoidal activation function (*fig. 2*). We attempted to recreate this topology in Matlab but found it impossible as each neuron in a feed-forward neural network's layer must have the same activation/transfer function. We attempted to use a feed-forward neural network with 2 hidden layers, the first layer with a heavyside step transfer function and the second with a binary sigmoidal (sometimes referred to as logarithmic sigmoidal) transfer function. This network performed relatively badly (*fig. 3*) compared to the initial attempt with significant error margins in all three data sets therefore we decided that keeping to a single hidden layer but modifying its parameters may lead to an ideal solution.

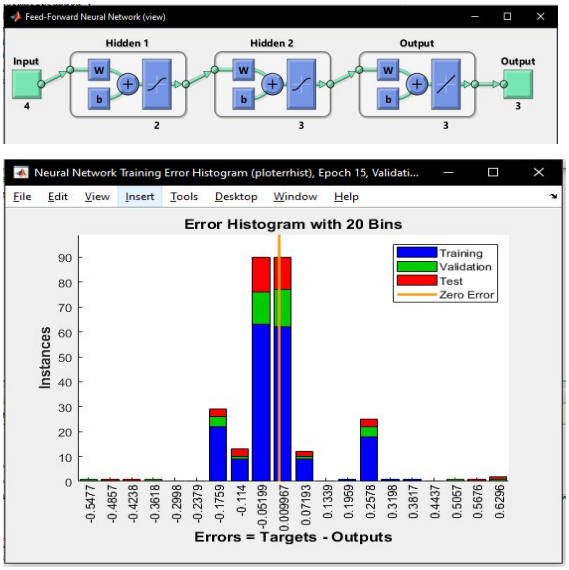

Fig. 3: Configuration of the feed-forward neural network and its associated error histogram

After this we decided to try reconfiguring the initial attempt but lower the maximum number of epochs and introduce a second stopping condition: maximum failed epochs. A failed epoch is defined by an increase in neural network confusion in one of the three categories (training, validation and testing). After reaching the maximum number of consecutive failed epochs the neural network "rewinds" back to its previous optimal state and finishes. Configuring this value depends on how likely you are to have local maxima that are different to the global maxima. If you have a failed epoch limit that is too low you risk stopping the neural network too early and finding only a local maxima whereas having a high failed epoch limit can lead to wasting system resources and time as the network tries to find a new solution despite having already found the optimal global solution. We decided to start with a neural network that had a hidden layer size of 10 neurons, a maximum epoch number of 100 and a maximum failed epoch number of 10. Due to the small dataset we are able to worry much less about the effect of epochs times and wasting time with failed epochs as the number of system resources wasted is minimal. This network also utilized random sampling and converged in 9 epochs (although continued until 19 epochs as per the max_fail parameter. The code for this neural network can be found in *Appendix B*. The confusion matrix (*fig. 4*) for this network shows that it classified all data in testing and validation with perfect accuracy (although had trouble with 2 pieces of data in the testing set). It is also worth noting that the random sampling algorithm decided to split data 104-23-23 which is fairly close to our original approximation of 80%-10%-10% (this one being 69.4%-15.3%-15.3%).

Fig. 4: Confusion matrix for neural network described above

We decided to try again with a different random sample to see how the network would perform with different data divisions and the results were very interesting. We kept the same network parameters (max_fail = 10; epochs = 100; hidden layer size = 10;) but ended up with a very different convergence time and cross-entropy plot. A comparison of our initial network's convergence versus this network's convergence can be found in *fig. 5*.

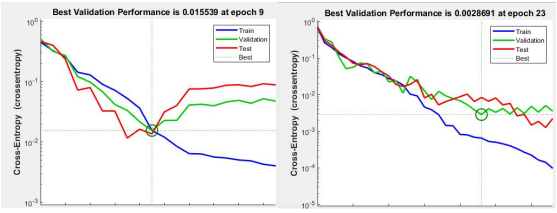

Fig. 5: A comparison of the convergence times and overall accuracy of two neural networks with identical data sets but different random sample splits

D. Neural Network Conclusion

In conclusion, the split sampling training method proved adequate for Iris dataset and was able to produce 2 networks that can successfully classify and generalise new inputs with a high level of confidence. The simple random sampling technique can produce varying models of the network but this is insignificant given the small data sample and few dimensions provided by the Iris dataset.

The faster convergence time of the first network can be explained by the way the data was split and ordered. There is a good chance that the first network's order of data as well as the division it made of the data into the training, validation and testing sets was "good" (i.e. allowed for easier and faster classification of the problem and quicker creation of the neural network) whereas the second network will have most likely split the data such that similar or hard to differentiate pieces of data were analysed first which led to a slower convergence (albeit one of higher accuracy as shown by the cross-entropy plot).

Both networks were able to classify the problem with little difficulty and achieved 100% accuracy in both testing and validation as well as 99.1% accuracy in the training phase. The neural network could be further refined by reducing the maximum number of epochs as well as the maximum number of consecutive failed epochs. This will reduce computational drain on larger data sets.

If a more complex dataset was provided with a much larger sample and many more dimensions the simple random sampling selection method will produce extremely different models to the point of having to explore more complex methods such as the Cadex method where samples are selected based on their euclidean distance. One could also add more neurons to the hidden layer as an additional means of compensating for the increased complexity

II. Genetic Algorithm

A. Introduction of Problem

The shubert function is a multidimensional, multimodal problem that contains several local minima and a single global minima. In this section a genetic algorithm will be developed that will find the global minimum while not being trapped in local minimal. The genetic

algorithm will be defined and measured by its parameters and their effect on reaching the global minimum. Fitness is the depth of the coordinate, the lower the value, the closer the coordinate is to a minima. The selection process will define how the fittest coordinates are selected to be used to create the next generation. Initial population determines how many coordinates will be plotted and assessed for selection. These coordinates are randomised to provide a fair chance that they may find the global minima.

B. Attempted Solution

For this problem we used Matlab to construct an optimization for the schubert function (contained in *Appendix B*). The optimizer was initialised using Matlab's Global Optimization Toolbox and the *optimtool* command. For the initial attempt at solving we set the initial population size to 50 and attempted a solution in 2 dimensions between the bounds of -10 and +10. The initial population's range was set to the entire bounds of the problem (-10, 10) to ensure that the area containing the global minima was not excluded from the problem bounds. We also ensured that the algorithm stopped after 6 failed generations (i.e. fitness was not improving) and after 30 total generations. This value was based off of preliminary tests with the algorithm.

The initial population size can have a profound impact upon the outcome of the GA. A higher initial population means higher diversity due to their different starting points. This provides a better source of genes for genetic crossover to utilise. The outcome from this will result in a hastened find for the global minima but an increase in overall time for mean fitness of the population to match that of the global minima

We used a stochastic uniform selection function with a rank based fitness scaling function. Using this we ran this genetic algorithm several times due to the random nature of the genetic algorithm's initialization and found that it failed to reach a consistent global minimum for the problem (results can be found in *fig. 6*)

Attempt #	1	2	3	4	5	6
Minima Found	-123.6	-38.7	-186.7	-186.7	-186.7	-123.6

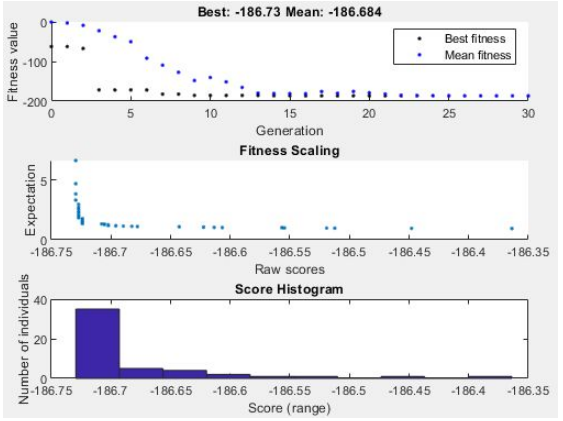

Fig. 6: Table showing minima found through initial application of genetic algorithm over 6 attempts (top) and graphs showing fitness value, expectation and number of individuals in the best attempt (bottom)

After this we decided that increasing the initial population size to 200 would help decrease the chances of falling into a local minima and increase the chances of finding the global minima.

We also decided to switch the fitness scaling from a rank function to a top function. This takes the top percentage of a generation and uses those to populate the next generation. The percentage we chose was 60% as this encourages a more diverse population which ensures we keep population members that have a chance of being near the global minima. A downside to this is we include a large population that are not close to the global or perhaps even local minima.

Through this process we were able to get more promising results. Of the 6 tests we performed all 6 found -186.731 as the global minima. A graph set showing our results can be seen in *fig. 7.* The algorithm performed quite slowly compared to the initial attempt, obviously due to the large increase in initial population size.

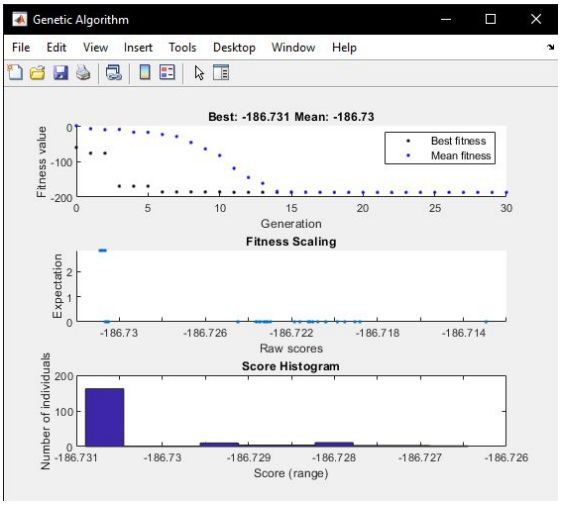

Fig. 7: Graphs showing fitness value, expectation and number of individuals in the best attempt in the second genetic algorithm

Finally we decided to test using tournament selection and roulette selection to see if it would give the same results. Both were equally accurate although roulette selection performed fastest. Tournament selection works by taking the top competitors from the first generation and using them to produce the next generation which they then compete alongside whereas roulette generation works by assigning a percentage chance of selection to each gene depending on its fitness. This ensures that more fit genes are significantly more likely to be chosen but does not completely remove the chance of less fit genes from being selected thus encouraging population diversity.

For the three-dimensional solution to this problem we increased the initial population size to 500 and switched to tournament selection. This yielded fairly consistent results (variance = ±0.1, trials = 10) in the region of -2709.09 (*fig. 8*). While this is not an ideal solution it can be assumed that this is the global minima for this solution

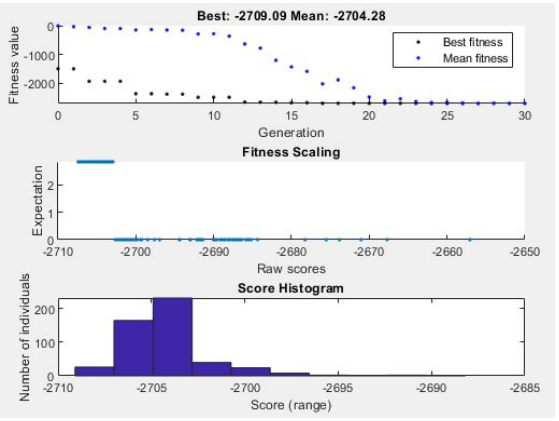

Fig. 8: Graphs showing fitness value, expectation and number of individuals in the best attempt in the 3-dimensional implementation of the genetic algorithm

C. Conclusion

In conclusion some parameters are more impactful in determining the effectiveness of the genetic algorithm. One of the most common issues when producing such an algorithm is getting stuck in a local minima. Having a higher population will help prevent this but if having a higher population is not feasible then increasing the diversity through fitness function changes or selection function changes will also help. This is because including a more diverse set of coordinates over the fittest will keep more options open ensuring that a gene that could lead to a global solution does not end up being discarded early on. The selection method's impact is relative to the initial population size; the higher the population the less impact selecting a specific selection method will have.

Appendix A: Coursework Specification

NENGA U25352

Coursework

Neural Networks and Genetic Algorithms

Demonstration: By Thursday 30th April 2019
Submission: **Thursday 7th May 2020 (by 6pm to Moodle)**

*This is an assessed piece of coursework (worth 40% of your final mark). It is, therefore, essential to be completed and submitted on time. If you are unclear about any aspect of the assignment, including the assessment criteria, please raise this at first opportunity. The usual regulations apply to a late submission of work. The submitted work must use Matlab Neural Network (**NN**) and Genetic Algorithm (**GA**) Toolboxes. The coursework you submit should be your own work. If it includes other people ideas and material, they must be properly referenced or acknowledged. Failing to do so intentionally or unintentionally constitutes plagiarism. The University treats plagiarism as a serious offence.*

Your Task

Includes two parts: NN for solving classification/mapping problem and GA for solving optimization problem.

The first part of the task you are set, is to analyse, design, train, validate and test artificial NN (using Matlab Neural Network Toolbox), that can solve classification/mapping problem for given sets of data that you are provided with (from the UCI Repository of Machine Learning Databases: http://archive.ics.uci.edu/ml/index.php).

You have to choose one of the two provided data sets: *Servo* or *Iris* (see Appendix A), which you will find in Moodle and K drive. *Servo* and *Iris* are popular benchmark problems with continuous and discrete outputs respectively, used for testing and evaluation of simple NN architectures and learning methods.

The purpose of this part of the assignment is to learn how to: analyse, pre-process, and visualise the available data; design neural network topology and architecture; use training algorithm (Backpropagation); and validate and test your NN and post-process the results. You must use Matlab NN toolbox to design and train neural networks that can classify, predict, or map input patterns with expected outputs and subsequently test the NN ability to generalise.

The second part of the coursework is about using GA for solving Global optimization problem. You will have to use Matlab Optimisation Toolbox (GA) to find the global minimum for one of the two provided families of multimodal, multidimensional mathematical functions (see Appendix B). You should choose one function (with minimum of two dimensions), appropriate representation of the GA initial population, fitness function, selection method, reproductive operators, stopping criteria, etc. You will also have to set up adequate parameters (number of chromosomes in the initial population, stochastic parameters (e.g., probability of mutation), bounds and constraints for some of the parameters, etc.).

You will have to use Matlab Optimization Toolbox (GA) to visualize the evolution of the chosen population, the best, the mean and the standard deviation values of the fitness function at each generation and to analyse and discuss the results.

If you have time you can also implement different selection strategies (e.g., fitness-proportional, rank-based and tournament selection), and compare and critically discuss the results. Also, some discussion should be given on premature convergence, how you can avoid it and how you can keep the selection pressure relatively constant throughout the whole run. In your search strategy pay attention to the characteristics/parameters that can improve the balance between the exploration and exploitation abilities of your global search technique.

Deliverable

By **6pm, Thursday 7.V.2020** you should submit (to Moodle) a .pdf file with your report (e.g., Gr1-3.pdf, Gr2-5.pdf, etc.). The main body of the report should be (excluding the Appendix) maximum of six pages (up to 4000 words). It should be well structured and soundly written, presenting critical and comparative analysis of the adopted approaches and interpretation of the results. Typical content:

1. Coursework specification (this document, as an Appendix);
2. Abstract;
3. Introduction;
4. NN part:
   - Data Set - Analysis and Pre-Processing (normalisation, standardisation); NN analysis and design;
   - Learning/Training analysis and results;
   - Test/Post-Processing/Generalisation results – comparison and analysis;
   - Conclusion and recommendations.
5. GA part:
   - Brief analysis of the chosen function to optimise;
   - Analysis and discussion of the adopted selection strategy, fitness function and reproductive operators;
   - Analysis and brief discussion of the chosen parameters, stopping criteria and convergence of the search;
   - Comparative and critical analysis of the results, conclusion and recommendations.
6. Reference list;
7. Appendices (any results, screen shots, diagrams and figures, and your Matlab code related to the coursework).

Assessing your assignment, I shall be looking for the following:

- Some thought and analysis have gone into what is the complexity of designing, training, testing, validating and using a neural network for solving the problem;
- You should show that you have thought about coding the output and appropriate NN topology and architecture, adopted adequate method of training (also appropriate parameters and transfer functions), etc.;
- You have thought about what data to use and for what purpose (subsets for training, testing and validation), and how the subsets are organized;
- You have thought about what kind of selection strategy to adopt for the GA optimization and what kind of operators to use for the reproduction (e.g., standard crossover and mutation, choice of parameters, e.g., probability of mutation, population number, population gap, etc.);
- You have paid attention to premature convergence and the ways of avoiding it, choice of stopping conditions and strategy to keep the balance between exploration and exploitation of your global search and also choice of parameters that could keep the selection pressure relatively constant throughout the whole run;
- Finally, you should demonstrate ability to critically analyse, discuss and interpret the implemented approaches and obtained results, and to give some recommendations for further improvements.

Assessment Criteria

A - An excellent, well-written and structured report that makes an interesting read. NN part - you have the data set explored and analysed in a number of ways (e.g., pre-processing, normalisation, standardisation; how are the different subsets split; will a swap of the training/validation and testing subsets produce a different network); appropriate coding, NN topology and architecture are chosen; the training, validation and testing are well analysed and discussed, and you have neural networks that perform well and can generalise.
GA part – you have analysed the function in hand and have adopted suitable selection strategy and reproductive operators. The choice of particular parameters and conditions is critically analysed and discussed.
You have analysed the complexity of the problems and the performance of the applied methods (different approaches may have been applied and compared), and have critically analysed, interpreted, and presented the results in an interesting and sound way.

B - A well-written report that is well structured into an interesting read. You have neural networks that perform well on the data set and you have analysed their performance. Your GA performance is critically analysed and discussed. You have exhibited some initiative in the approach taken and the results are clearly presented and critically discussed and compared.

C- A reasonable report that presents an account of the adopted approaches with critical analysis and discussion at places. The results are presented reasonably clear.

D - A report that presents the results mostly in descriptive way, but show reasonable understanding of the applied approaches.

F - Either no report submitted, or a report that presents very few results and little or no understanding of how these methods work.

Mark Scheme

Analysis and Pre-Processing (normalisation, standardisation) of the datasets and the multimodal function to be optimized.	**10**
NN and GA assumptions, Analysis and Design.	**20**
NN Training, Testing, Post-Processing, Generalisation and Results. Adopted training, validation and testing strategy analysed and explained – e.g., why 'split-sample' or 'cross-validation' adopted?	**20**
GA analysis of the Run, Adopted Selection strategy, Reproductive operators and Results.	**20**
Report – structure, style, and presentation. Critical and comparative analysis and discussion demonstrating in-depth understanding and grasp of the investigated methods and approaches. Good visualisation and sound discussion and interpretation of the results.	**30**

Appendix B: Matlab Code

Neural Network Section

ClassifyIrises

```
function ClassifyIrises
IN = NormalizedData';
TARGET = TargetData';
p = patternnet(15);
p.trainParam.epochs = 100;
p.trainParam.max_fail = 10;
%p.layers{1}.transferFcn = 'logsig';
p = train(p,IN,TARGET);
```

NormalizedData

```
function ND = NormalizedData
ND = [
0.6456 0.443 0.1772 0.0253
0.6203 0.3797 0.1772 0.0253
0.5949 0.4051 0.1646 0.0253
0.5823 0.3924 0.1899 0.0253
0.6329 0.4557 0.1772 0.0253
0.6835 0.4937 0.2152 0.0506
0.5823 0.4304 0.1772 0.038
0.6329 0.4304 0.1899 0.0253
0.557 0.3671 0.1772 0.0253
0.6203 0.3924 0.1899 0.0127
0.6835 0.4684 0.1899 0.0253
0.6076 0.4304 0.2025 0.0253
0.6076 0.3797 0.1772 0.0127
0.5443 0.3797 0.1392 0.0127
0.7342 0.5063 0.1519 0.0253
0.7215 0.557 0.1899 0.0506
0.6835 0.4937 0.1646 0.0506
0.6456 0.443 0.1772 0.038
0.7215 0.481 0.2152 0.038
0.6456 0.481 0.1899 0.038
0.6835 0.4304 0.2152 0.0253
0.6456 0.4684 0.1899 0.0506
0.5823 0.4557 0.1266 0.0253
0.6456 0.4177 0.2152 0.0633
0.6076 0.4304 0.2405 0.0253
0.6329 0.3797 0.2025 0.0253
0.6329 0.4304 0.2025 0.0506
0.6582 0.443 0.1899 0.0253
0.6582 0.4304 0.1772 0.0253
0.5949 0.4051 0.2025 0.0253
0.6076 0.3924 0.2025 0.0253
0.6835 0.4304 0.1899 0.0506
0.6582 0.519 0.1899 0.0127
0.6962 0.5316 0.1772 0.0253
0.6203 0.3924 0.1899 0.0253
0.6329 0.4051 0.1519 0.0253
0.6962 0.443 0.1646 0.0253
0.6203 0.4557 0.1772 0.0127
0.557 0.3797 0.1646 0.0253
0.6456 0.4304 0.1899 0.0253
0.6329 0.443 0.1646 0.038
0.5696 0.2911 0.1646 0.038
0.557 0.4051 0.1646 0.0253
0.6329 0.443 0.2025 0.0759
0.6456 0.481 0.2405 0.0506
0.6076 0.3797 0.1772 0.038
0.6456 0.481 0.2025 0.0253
0.5823 0.4051 0.1772 0.0253
0.6709 0.4684 0.1899 0.0253
0.6329 0.4177 0.1772 0.0253
0.8861 0.4051 0.5949 0.1772
0.8101 0.4051 0.5696 0.1899
0.8734 0.3924 0.6203 0.1899
0.6962 0.2911 0.5063 0.1646
0.8228 0.3544 0.5823 0.1899
0.7215 0.3544 0.5696 0.1646
0.7975 0.4177 0.5949 0.2025
0.6203 0.3038 0.4177 0.1266
0.8354 0.3671 0.5823 0.1646
0.6582 0.3418 0.4937 0.1772
0.6329 0.2532 0.443 0.1266
0.7468 0.3797 0.5316 0.1899
0.7595 0.2785 0.5063 0.1266
0.7722 0.3671 0.5949 0.1772
0.7089 0.3671 0.4557 0.1646
0.8481 0.3924 0.557 0.1772
0.7089 0.3797 0.5696 0.1899
0.7342 0.3418 0.519 0.1266
0.7848 0.2785 0.5696 0.1899
0.7089 0.3165 0.4937 0.1392
0.7468 0.4051 0.6076 0.2278
0.7722 0.3544 0.5063 0.1646
0.7975 0.3165 0.6203 0.1899
0.7722 0.3544 0.5949 0.1519
0.8101 0.3671 0.5443 0.1646
0.8354 0.3797 0.557 0.1772
0.8608 0.3544 0.6076 0.1772
0.8481 0.3797 0.6329 0.2152
0.7595 0.3671 0.5696 0.1899
0.7215 0.3291 0.443 0.1266
0.6962 0.3038 0.481 0.1392
0.6962 0.3038 0.4684 0.1266
0.7342 0.3418 0.4937 0.1519
0.7595 0.3418 0.6456 0.2025
0.6835 0.3797 0.5696 0.1899
0.7595 0.4304 0.5696 0.2025
0.8481 0.3924 0.5949 0.1899
0.7975 0.2911 0.557 0.1646
0.7089 0.3797 0.519 0.1646
0.6962 0.3165 0.5063 0.1646
0.6962 0.3291 0.557 0.1519
0.7722 0.3797 0.5823 0.1772
0.7342 0.3291 0.5063 0.1519
0.6329 0.2911 0.4177 0.1266
0.7089 0.3418 0.5316 0.1646
0.7215 0.3797 0.5316 0.1519
0.7215 0.3671 0.5316 0.1646
```

0.7848 0.3671 0.5443 0.1646 1 0 0
0.6456 0.3165 0.3797 0.1392 1 0 0
0.7215 0.3544 0.519 0.1646 1 0 0
0.7975 0.4177 0.7595 0.3165 1 0 0
0.7342 0.3418 0.6456 0.2405 1 0 0
0.8987 0.3797 0.7468 0.2658 1 0 0
0.7975 0.3671 0.7089 0.2278 1 0 0
0.8228 0.3797 0.7342 0.2785 1 0 0
0.962 0.3797 0.8354 0.2658 1 0 0
0.6203 0.3165 0.5696 0.2152 1 0 0
0.9241 0.3671 0.7975 0.2278 1 0 0
0.8481 0.3165 0.7342 0.2278 1 0 0
0.9114 0.4557 0.7722 0.3165 1 0 0
0.8228 0.4051 0.6456 0.2532 1 0 0
0.8101 0.3418 0.6709 0.2405 1 0 0
0.8608 0.3797 0.6962 0.2658 1 0 0
0.7215 0.3165 0.6329 0.2532 1 0 0
0.7342 0.3544 0.6456 0.3038 1 0 0
0.8101 0.4051 0.6709 0.2911 1 0 0
0.8228 0.3797 0.6962 0.2278 1 0 0
0.9747 0.481 0.8481 0.2785 1 0 0
0.9747 0.3291 0.8734 0.2911 1 0 0
0.7595 0.2785 0.6329 0.1899 1 0 0
0.8734 0.4051 0.7215 0.2911 1 0 0
0.7089 0.3544 0.6203 0.2532 1 0 0
0.9747 0.3544 0.8481 0.2532 1 0 0
0.7975 0.3418 0.6203 0.2278 1 0 0
0.8481 0.4177 0.7215 0.2658 1 0 0
0.9114 0.4051 0.7595 0.2278 1 0 0
0.7848 0.3544 0.6076 0.2278 1 0 0
0.7722 0.3797 0.6203 0.2278 1 0 0
0.8101 0.3544 0.7089 0.2658 1 0 0
0.9114 0.3797 0.7342 0.2025 1 0 0
0.9367 0.3544 0.7722 0.2405 1 0 0
1 0.481 0.8101 0.2532 1 0 0
0.8101 0.3544 0.7089 0.2785 1 0 0
0.7975 0.3544 0.6456 0.1899 1 0 0
0.7722 0.3291 0.7089 0.1772 1 0 0
0.9747 0.3797 0.7722 0.2911 1 0 0
0.7975 0.4304 0.7089 0.3038 1 0 0
0.8101 0.3924 0.6962 0.2278 1 0 0
0.7595 0.3797 0.6076 0.2278 1 0 0
0.8734 0.3924 0.6835 0.2658 1 0 0
0.8481 0.3924 0.7089 0.3038 1 0 0
0.8734 0.3924 0.6456 0.2911 1 0 0
0.7342 0.3418 0.6456 0.2405 1 0 0
0.8608 0.4051 0.7468 0.2911 1 0 0
0.8481 0.4177 0.7215 0.3165 1 0 0
0.8481 0.3797 0.6582 0.2911 1 0 0
0.7975 0.3165 0.6329 0.2405 1 0 0
0.8228 0.3797 0.6582 0.2532 0 1 0
0.7848 0.4304 0.6835 0.2911 0 1 0
0.7468 0.3797 0.6456 0.2278 0 1 0
] 0 1 0
0 1 0
TargetData 0 1 0
function TD = TargetData 0 1 0
TD = [0 1 0

```
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 1 0
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
0 0 1
]
```

Genetic Algorithm Section

Shubert1_fun ***[As provided by unit lecturer]***

```
% Shubert1 function
% Assume each xi, i=1,...,n is in [-10, +10]
interval.
% Calculates Shubert1 for the actual argument
x
%
function y = Shubert1_fun(x)
n = length(x);
p = 1.0;
for i = 1: n
        s = 0.0;
        for j = 1:5
        s = s+j.*cos((j+1).*x(i)+j);
        end
        p = p.*s;
end
y = p;
```