ML Project 1 Report

Circuit Optimization By Genetic Algorithm

Name: Jinming Ren

UESTC ID: —

UofG ID: —

Date: November 9, 2024

University: —

Contents

1 Problem formulation

1.1 Problem setup

The aim of this problem is to approach the expected voltage-temperature curve $V(t)$ at point B shown in Figure 2 by choosing the best combination of the register values (R_1, R_2, R_3, R_4), and Beta constants (β_1, β_2) of the two thermistors within a particular range, using the Genetic Algorithm (GA). The circuit structure is shown in Figure 1.

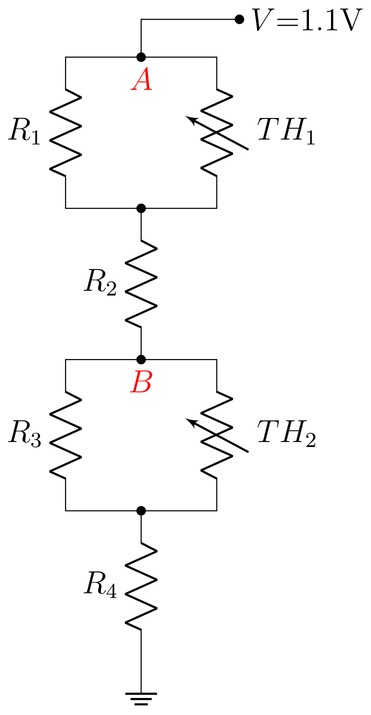


Figure 1: Circuit structure

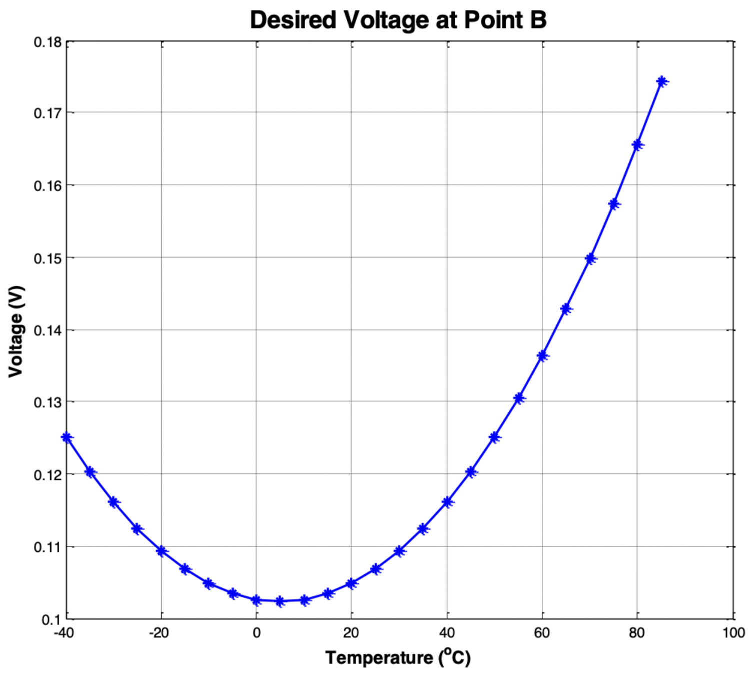


Figure 2: Desired voltage curve [1]

A thermistor is just a normal resistor except that its resistance R_{TH} changes with temperature T according to Equation (1) (T is in Kelvin, t is in Celsius):

$$R_{TH}(T) = R_N \exp\left(\beta\left(\frac{1}{T} - \frac{1}{T_N}\right)\right), \tag{1}$$

where β is the characteristic parameter of a thermistor. R_N is called the nominal resistance, which is the resistance of the thermistor at a reference temperature $T_N = 298.15\text{K} = 25°\text{C}$. In general, R_N and β are independent parameters. However, in this problem, we only have 9 kinds of thermistors (whose R_Ns and βs are listed in `StandardComponentValues.mat`), which means R_N is fixed when β is given.

1.2 Mathematical model

In this part, we will formulate the problem in formal mathematical language.

For the search space, according to file `StandardComponentValues.mat`, there are 70 different register values for R_1-R_4 (Res), 9 Beta values for TH_1 and TH_2 (ThBeta) and the corresponding R_n values (ThVal), all gathered in an ordered list (See Table 1). We *could* define the search space Ω directly as the cartesian product of these candidate sets. However, since the candidate values are *ordered*, we could also use indices for simplicity. Since there are 4 registers and 2 thermistors in total, we define the search space Ω as:

$$\Omega := (\mathbb{Z}/70\mathbb{Z})^4 \times (\mathbb{Z}/9\mathbb{Z})^2 \tag{2}$$

Table 1: Variable Data Table

Variable	Size	Type	Value
Res	70	number	[300, 330, 360, 390, 430, 470, 510, 560, 620, 680, 750, 820, 910, 1000, 1100, 1200, 1300, 1500, 1600, 1800, 2000, 2200, 2400, 2700, 3000, 3300, 3600, 3900, 4300, 4700, 5100, 5600, 6200, 6800, 7500, 8200, 9100, 10000, 11000, 12000, 13000, 15000, 16000, 18000, 20000, 22000, 24000, 27000, 30000, 33000, 36000, 39000, 43000, 47000, 51000, 56000, 62000, 68000, 75000, 82000, 91000, 100000, 110000, 120000, 130000, 150000, 160000, 180000, 200000, 220000]
ThBeta	9	number	[2750, 3680, 3560, 3620, 3528, 3930, 3960, 4090, 3740]
ThVal	9	number	[50, 220, 1000, 2200, 3300, 4700, 10000, 22000, 33000]

Each point $\omega \in \Omega$ determines a actual voltage curve $V_\omega(t)$. Our goal is to minimize the difference between $V(t)$ and $V_\omega(t)$. We choose the $\mathcal{L}^2$ distance (squared) between them to be the loss function L, i.e.

$$L(\omega) := \|V - V_\omega\|_2^2 \tag{3}$$

The detailed implementation of this loss function is in the file `Func/l2squared.m`[1]. Now, the problem in Section 1.1 can be stated in formal language as:

Problem 1. *Solve the optimization problem*

$$\arg\min_{\omega\in\Omega} L(\omega) \tag{4}$$

subject to $\omega \in \Omega$. ($L(\omega)$ and Ω are defined in Equation (3) *and Equation* (2) *respectively).*

2 Implementation

We use Genetic Algorithm (GA) to perform the optimization in Problem 1. The pseudo code of the algorithm is shown in Algorithm 1. The chosen parameters and the full codes in MATLAB is shown in Appendix A.

3 Results

3.1 Convergence plot

The convergence plot is constructed in the following manner:

We took 10 replicates (`runNum=10`) of the GA with the same parameters. In each run, there was `genNum=151` number of generations, and `popNum=201` number of individuals in each generation (we will show how we chose those GA parameters in Section 4.2 in detail). We first pick the best[2] individual in each generation as a representative of this particular generation (using the `popFilter()` function). We then store the loss value, corresponding register configuration (in index form and register parameter form) in the corresponding position in vector `bestLoss` and matrices `idxPop_best`, `RegPara_best`, respectively. Since we run the whole algorithm 10 times, we will add another dimension to these three objects (later two

[1]The full project can be found on https://github.com/Marcobisky/AI-ML-project1.

[2]in the sense that its loss value is the least among others.

Algorithm 1 Genetic Algorithm for Optimizing Thermistor Circuit

```
1: **Input:** $Tdata$, $Vdata$, $lb$, $ub$, $popNum$, $elitismRate$, $crossRate$, $mutateRate$, $genNum$, $runNum$
2: **Output:** Optimal configuration, best, worst, average loss, convergence plot
3: for $run = 1$ to $runNum$ do
4:     Initialize population in index form ($idxPop$)
5:     for $i = 1$ to $genNum$ do
6:         $RegPop \leftarrow$ idx2RegPara($idxPop$)
7:         $popLoss \leftarrow$ l2squared($RegPop$, $Tdata$, $Vdata$)
8:         Store best individual with popFilter
9:         $[elitePop, idxPop\_s] \leftarrow$ selectSUS($idxPop$, $popLoss$, $elitismRate$)
10:        $idxPop\_c \leftarrow$ crossover($idxPop\_s$, $crossRate$)
11:        $idxPop\_m \leftarrow$ mutate($idxPop\_c$, $mutateRate$, $lb$, $ub$)
12:        $idxPop \leftarrow$ mix($elitePop$, $idxPop\_m$)
13:    end for
14: end for
15: Generate convergence plot with cvgPlot
16: Analyze results: best, worst, average loss using runPlot
17: Display optimal curve and comparison to expected $Vdata$
```

becomes three-dimensional tensors). These three arrays store all the information needed for future performance analysis.

As the number of generation increases, the quality of the representive should be generally better, but not always. Therefore, we store the best-so-far representative (the best representative from generation 1 to the current generation) over each runs in another array `bestLoss_SoFar` using the `cummin()` function (this is done inside `cvgPlot()` function). Then, according to the requirements (which is actually not well-defined), we took average along each rows and store it in `bestLoss_Sofar_avg` vector. Finally, we plot the contents in `bestLoss_Sofar_avg` to get the convergence plot shown in Figure 3. The voltage-temperature curve of the best representative among all 10 runs is shown in Figure 4.

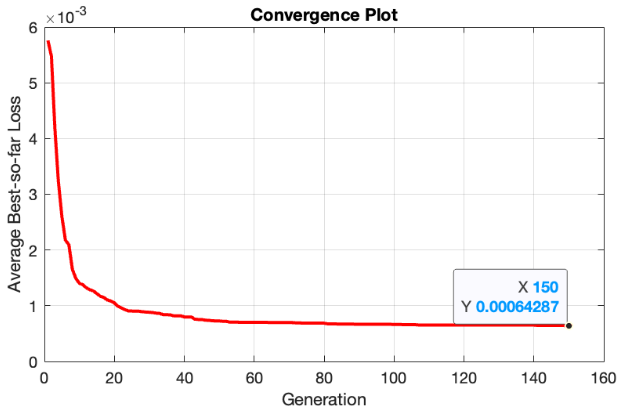


Figure 3: The convergence plot

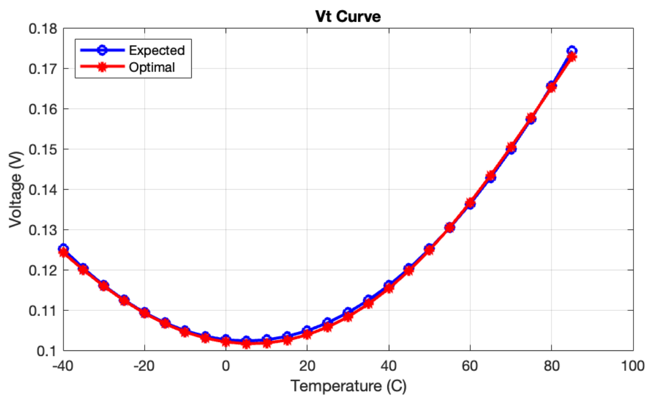


Figure 4: The optimal V-t curve

3.2 Other statistical data

In Section 3.1, we derived the loss matrix (`bestloss`) of each representative in each generation and each run. We then extract the least loss in each run in a 10-dimensional vector `bestLossEachRun`. The content of this vector is shown in Figure 5.

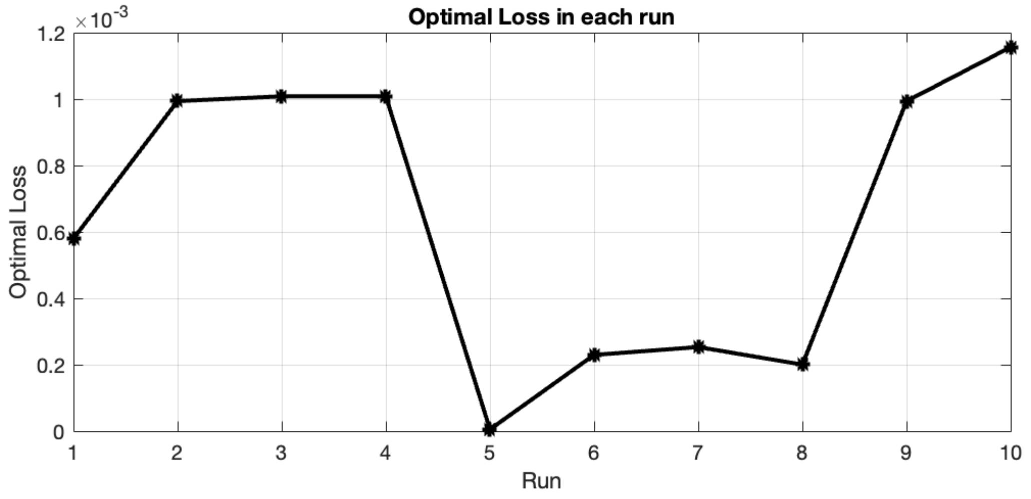


Figure 5: Least loss in each run

The best, worst, and average optimal objective function values and their standard deviations over 10 runs would be the least, most, average and standard deviation of Figure 5, which are shown in Table 2.

Table 2: Statistics of Figure 5

Metric	**Value**
Lowest Loss L_{min}	5.3107×10^{-6}
Highest Loss L_{max}	1.1563×10^{-3}
Average Best Loss L_{avg}	6.4287×10^{-4}
Standard Deviation σ	4.3536×10^{-4}

We then find the corresponding register configuration of optimal loss in Table 2, which is shown in Table 3. Using these parameters, we can construct the voltage response curve of the circuit, which is shown in Figure 4. The optimal curve is extremely close to the expected curve.

Table 3: Register configuration for L_{min} in Table 2

Register	**Index**	**Parameter**
R_1	32	$R_1 = 5600\Omega$
R_2	20	$R_2 = 1800\Omega$
R_3	9	$R_3 = 620\Omega$
R_4	9	$R_4 = 620\Omega$
TH_1	8	$R_{N_1} = 22000\Omega$, $\beta_1 = 4090\text{K}$
TH_2	1	$R_{N_2} = 50\Omega$, $\beta_2 = 2750\text{K}$

4 GA parameter tuning and discussion

In order to perform efficient GA evolution, we had to fine tune the most efficient combination of 5 parameters:

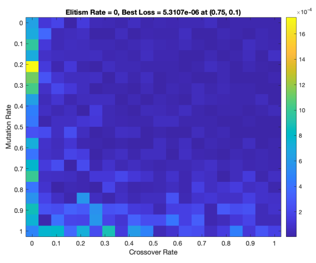
1. `popNum` (`p`): the number of individuals in each generation

2. `genNum` (`g`): the number of generations
3. `crossRate` (`c`): the probability of crossover
4. `mutateRate` (`m`): the probability of mutation
5. `elitismRate` (`e`): the proportion of a generation that is directly copied to the next generation

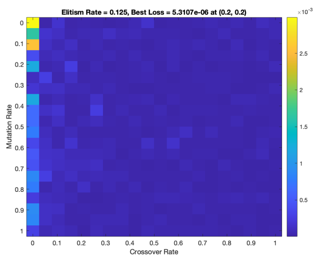
In order to perform systematic tuning, we adopted the following steps.

4.1 Steady state analysis

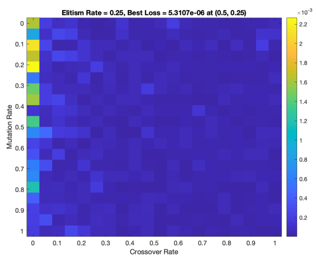
We will show that GA is (c, m, e)-independent when given large enough `genNum` and `popNum`. First, in order to reach the "convergence state" (or "steady state") of GA, we fix a relatively large number of generations (`genNum=400`) and population size (`popNum=800`). Then, we partitioned the rest three parameters $\xi = (c, m, e) \in [0, 1]^3$ into a 3 dimensional matrix of size $20 \times 20 \times 8 = 3200$. We then run the GA with each combination of parameters and record the best loss value. The results are shown in Figure 6.



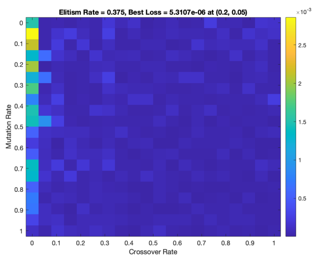
(a) $\xi^* = (.75, 1, 0)$



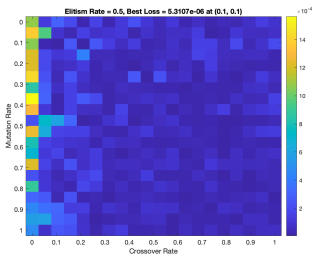
(b) $\xi^* = (.2, .2, .125)$



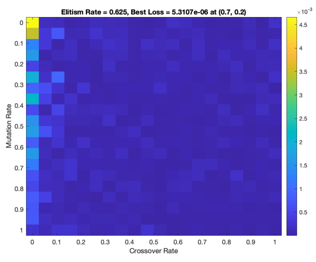
(c) $\xi^* = (.5, .25, .25)$



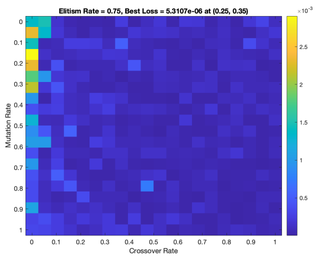
(d) $\xi^* = (.2, .05, .375)$



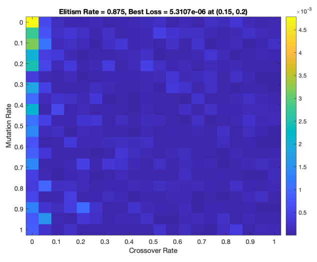
(e) $\xi^* = (.1, .1, .5)$



(f) $\xi^* = (.7, .2, .625)$



(g) $\xi^* = (.25, .35, .75)$



(h) $\xi^* = (.15, .2, .875)$

Figure 6: The best loss values of 3200 different $\xi = (c, m, e)$ parameter combinations

We can see from Figure 6 that the best loss value is always $L_{\min} = 5.3107 \times 10^{-6}$. However, the optimal parameter combination ξ is slightly different. For example, in Figure 6b, the best combination is $\xi^* = (.2, .2, .125)$, which is different from other 7 cases. This means GA is not very sensitive to the choice of `crossRate`, `mutateRate` and `elitismRate` when the number of generations and population size are fixed and large enough.

4.2 Dynamic state analysis

Having considered the behavior of GA when convergent, we now explore how different values of `popNum` and `genNum` affect the efficiency of GA. We define $\zeta = (p, g) \in [1, 501]^2$ (these numbers are slightly larger than the empirical convergence bound). At each (p, g) point, we did the whole process in Section 4.1, and pick the lowest loss value and the corresponding tuple (c, m, e) and store them in array `losscme` and cell `bestcme`, respectively. Then, we plot the `losscme` in logarithmic scale in Figure 7.

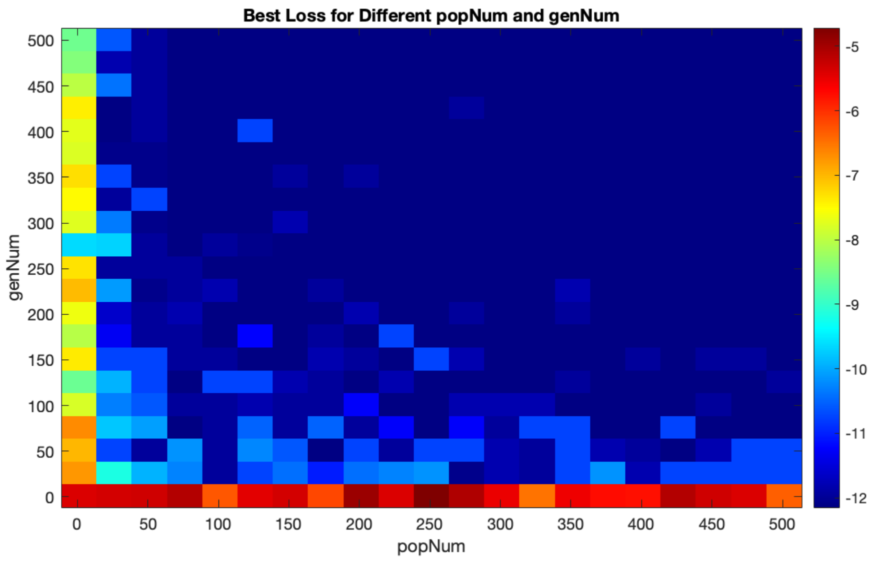


Figure 7: The best loss values of 400 different $\zeta = (p, g)$ parameter combinations

According to Section 4.1, once the algorithm converges, the loss value is garanteed to be $L_{\min} = 5.3107 \times 10^{-6}$. In Figure 7, the optimal $\zeta = (p, g)$ combination is (roughly) $\zeta^* = (201, 151)$, whose corresponding loss reached $L_{\min}$ and the corresponding (c, m, e) is $(.2, .2, .2)$. This means the GA gets converged when the GA parameters are set to $(\zeta, \xi) = (201, 151, .2, .2, .2)$, which is exactly the parameters we used in line 15-19 in Appendix A. We are confident that these set of parameters in the *most efficient* for the particular GA algorithm used in this project.

5 Conclusion

In this project, we solved an engineering optimization problem using Genetic Algorithm (GA). Not only did we found the optimal register configuration of $\mathcal{L}^2$ loss of less than 5.3107×10^{-6}, we also use Steady state and Dynamic state analysis to find the *most efficient* combination of GA parameters. Although this method is a little bit tedious and time-consuming, it may provide some heuristic guidance for understanding the behavior of GAs.

A Main code

The `main` function of the Genetic Algorithm is shown below[3].

Listing 1: Genetic algorithm

```matlab
%% Main function for GA optimization
clear;
addpath('./Func');
addpath('./GAlib');
addpath('./Data');
load('StandardComponentValues.mat')

%% Temperature samples and expected Voltage Data
Tdata = -40:5:85;
Vdata = 1.026E-1 + -1.125E-4 * Tdata + 1.125E-5 * Tdata.^2;

%% Main parameters
lb = [1 1 1 1 1 1];
ub = [70 70 70 70 9 9];
popNum = 201;      % try 300 to get better results
elitismRate = 0.2; % try 0.8 to get better results
crossRate = 0.2;   % try 0.9 to get better results
mutateRate = 0.2;  % try 0.8 to get better results
genNum = 151;      % try 300 to get better results
runNum = 10;

%% For plotting
gen_coor = 1:genNum;
bestLoss = zeros(genNum, runNum);
% store the optimal configuration information in each generation
idxPop_best = zeros(genNum, 6, runNum);  % in index form
RegPara_best = zeros(genNum, 8, runNum); % in register parameter form

%% Perform GA optimization
for runRound = 1:runNum
    % Initialize population
    idxPop = popInit(popNum, lb, ub);

    for i = gen_coor
        % Calculate fitness
        RegPop = idx2RegPara(idxPop, Res, ThVal, ThBeta);
        popLoss = l2squared(RegPop, Tdata, Vdata);

        % record the register configuration of the best individual
        [idxPop_best(i, :, runRound), RegPara_best(i, :, runRound), bestLoss(i
            , runRound)] = popFilter('best', idxPop, popLoss, Res, ThVal,
            ThBeta);

        % Select using SUS
```

[3]The full project can be found on https://github.com/Marcobisky/AI-ML-project1.

```matlab
        [elitePop, idxPop_s] = selectSUS(idxPop, popLoss, elitismRate);

        % Crossover
        idxPop_c = crossover(idxPop_s, crossRate);

        % Mutate
        idxPop_m = mutate(idxPop_c, mutateRate, lb, ub);

        % Mix elite and mutated population
        idxPop_mix = mix(elitePop, idxPop_m);

        idxPop = idxPop_mix; % Replace the old population with the new one
    end
end

%% Plot the average of the best-so-far objective function values over 10 runs
    in each generation
cvgPlot(bestLoss, genNum);

%% Plot the best, worst, and average optimal objective function values and
    their standard deviations over 10 runs
[~, bestBest, worstBest, avgBest, stdDev] = runPlot(bestLoss, runNum, 1);

%% Print the best, worst and average best loss
% Find corresponding register configuration
idxPop_best = idxPop_best(bestBest{2}(1), :, bestBest{2}(2));
RegPara_best = RegPara_best(bestBest{2}(1), :, bestBest{2}(2));

% Print the results
disp(['Over ', num2str(runNum), ' runs:']);
disp(['The lowest loss is: ', num2str(bestBest{1})]);
disp(['at this point in the search space: ', num2str(idxPop_best)]);
disp(['with register configuration: ', num2str(RegPara_best)]);
disp(' ');
disp(['The highest loss is: ', num2str(worstBest{1})]);
disp(['The average best loss is: ', num2str(avgBest)]);
disp(['The standard deviation is: ', num2str(stdDev)]);

%% Plot the final best Vt curve
% Calculate the Vt curve given the best register configuration
Vbest = temp2volt(RegPara_best, Tdata);
figure;
plot(Tdata, Vdata, 'bo-', 'LineWidth', 2); % expected
hold on;
plot(Tdata, Vbest, 'r*-', 'LineWidth', 2); % optimal
xlabel('Temperature (C)');
ylabel('Voltage (V)');
title('Vt Curve');
legend('Expected', 'Optimal', 'Location', 'northwest');
grid on;
```

References

[1] Optimal Component Selection Using the Mixed-Integer Genetic Algorithm, November 2024.