Lecture 6: Single-Objective GA Examples

Rosenbrock's Function (The Banana)

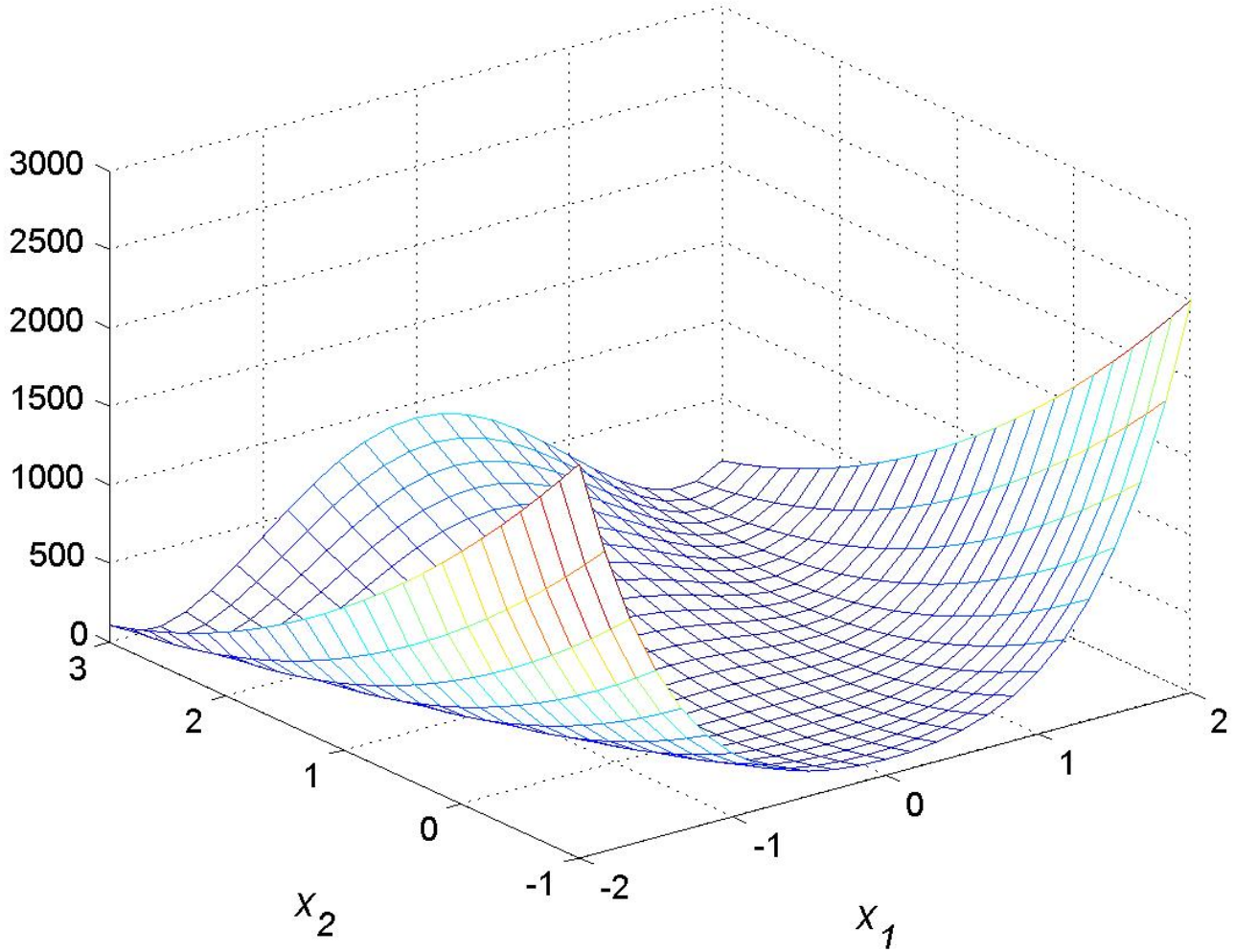
Problem Statement

- Minimization of Rosenbrock's function

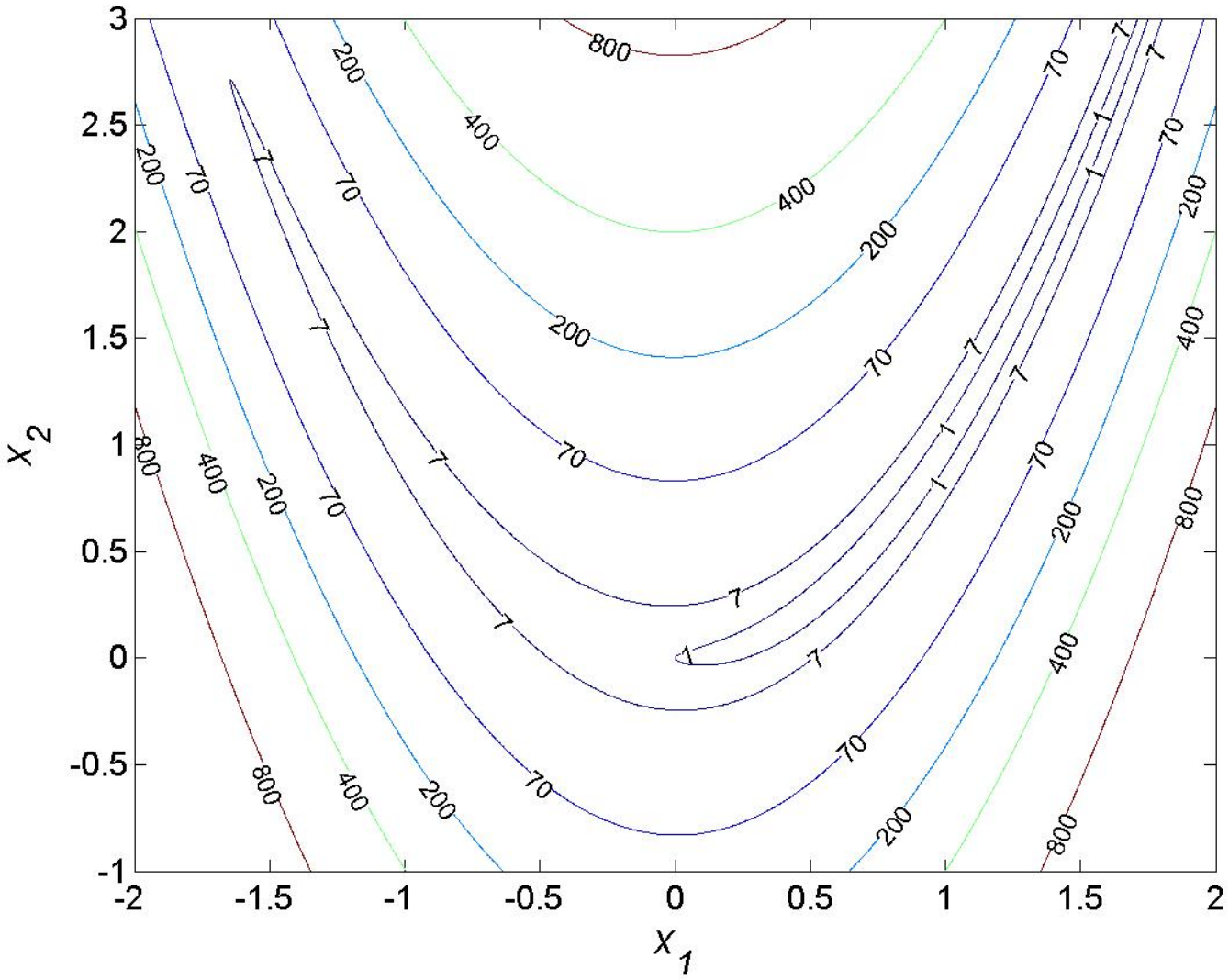
$$\min f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2$$

- Also called as **banana** function due to the shape of the level sets
- The global minimizer is located at (1,1)

Banana Function



Banana Function Level Sets



Fitness Function

- Rosenbrock's function is non-negative
- The inverse of the Rosenbrock's function is used as the fitness function

$$f(x_1, x_2) = \frac{1}{100(x_2 - x_1)^2 + (1 - x_1)^2 + 0.001}$$

- A small positive value is added to the denominator to avoid the singularity at the minimizer

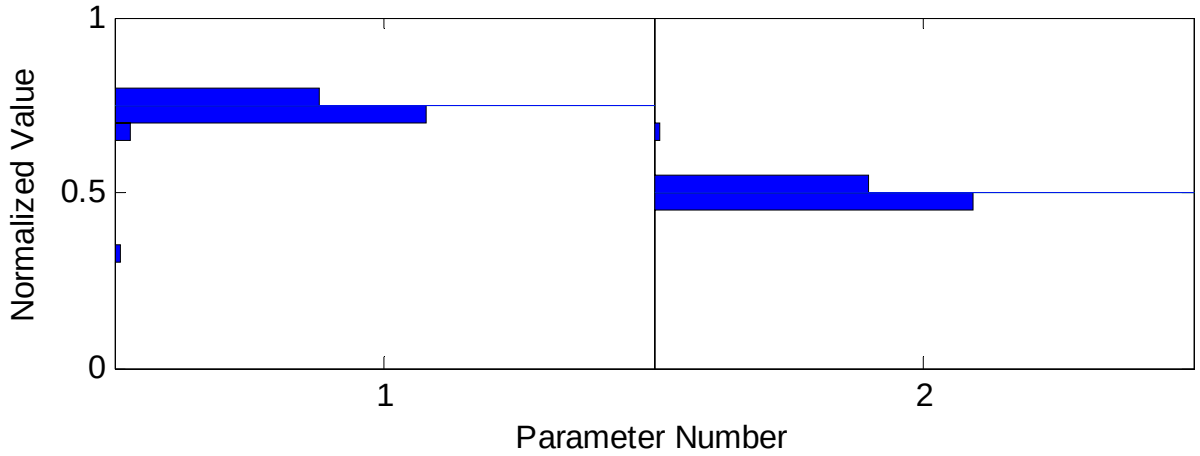
banana.m

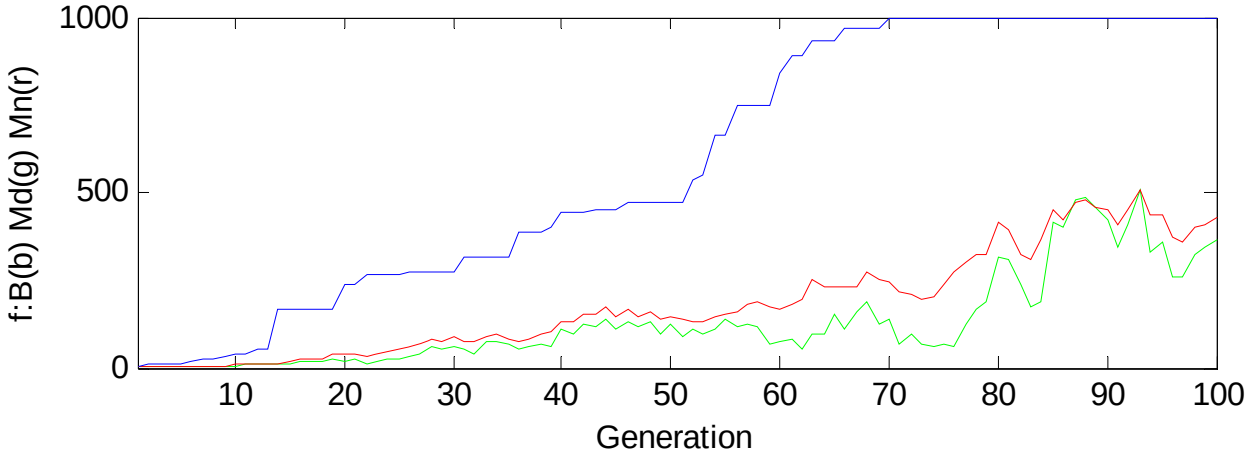
```
% BANANA.M
%
% Rosenbrock's Function

function f = banana(x)

% Rosenbrock's function
f1 = 100*(x(2) - x(1)^2)^2 + ...
         (1 - x(1))^2;

% Fitness function
f = 1/(0.001 + f1);
```

banana_optimize.m

```
% banana_optimize
% Rosenbrock's problem

% Initialize the parameters
GAP=gapdefault;

% Define gene parameters
%                    x1     x2
% gene               1      2
GAP.gd_min  = [    -2     -1    ];
GAP.gd_max  = [     2      3    ];
GAP.gd_type = [     2      2    ];
GAP.gd_cid  = [     1      1    ];

% Execute GOSET
[P,GAS] = gaoptimize(@banana,GAP,[],[],[],[]);

% Display the gene values of the best individual
GAS.bestgenes(:,length(GAS.bestgenes))
```

Results



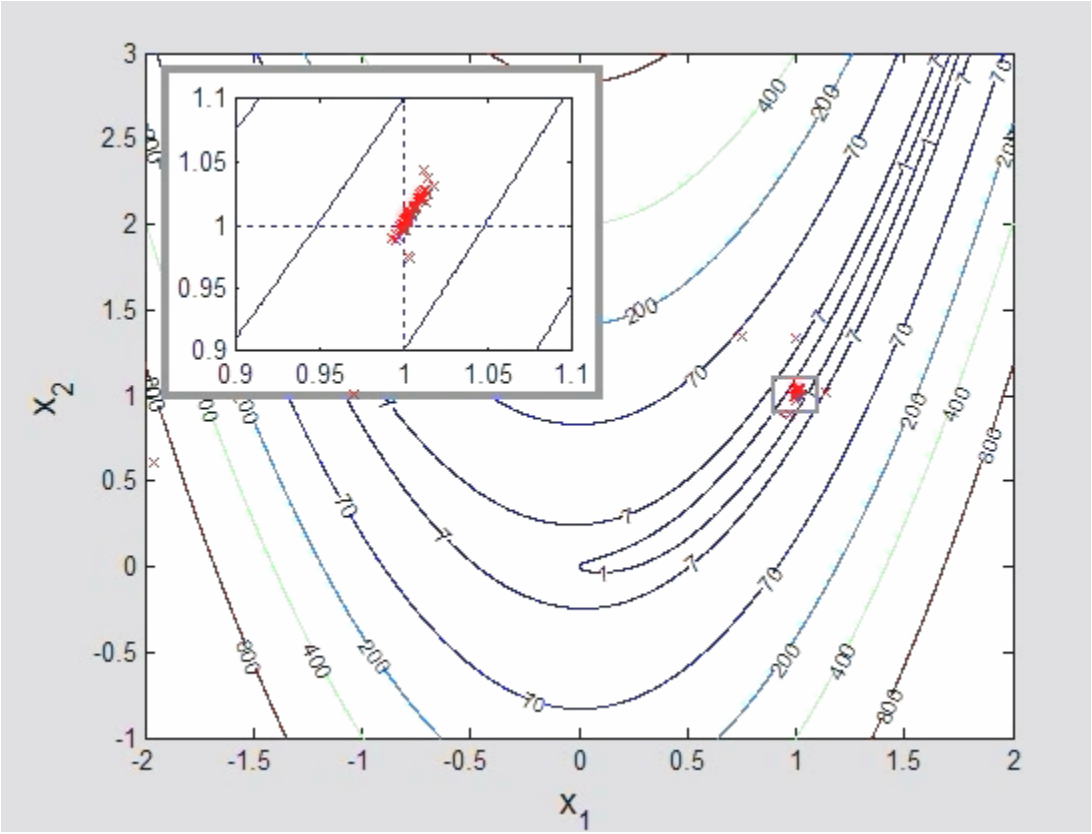


ans =

0.99986681320870

0.99978907003301

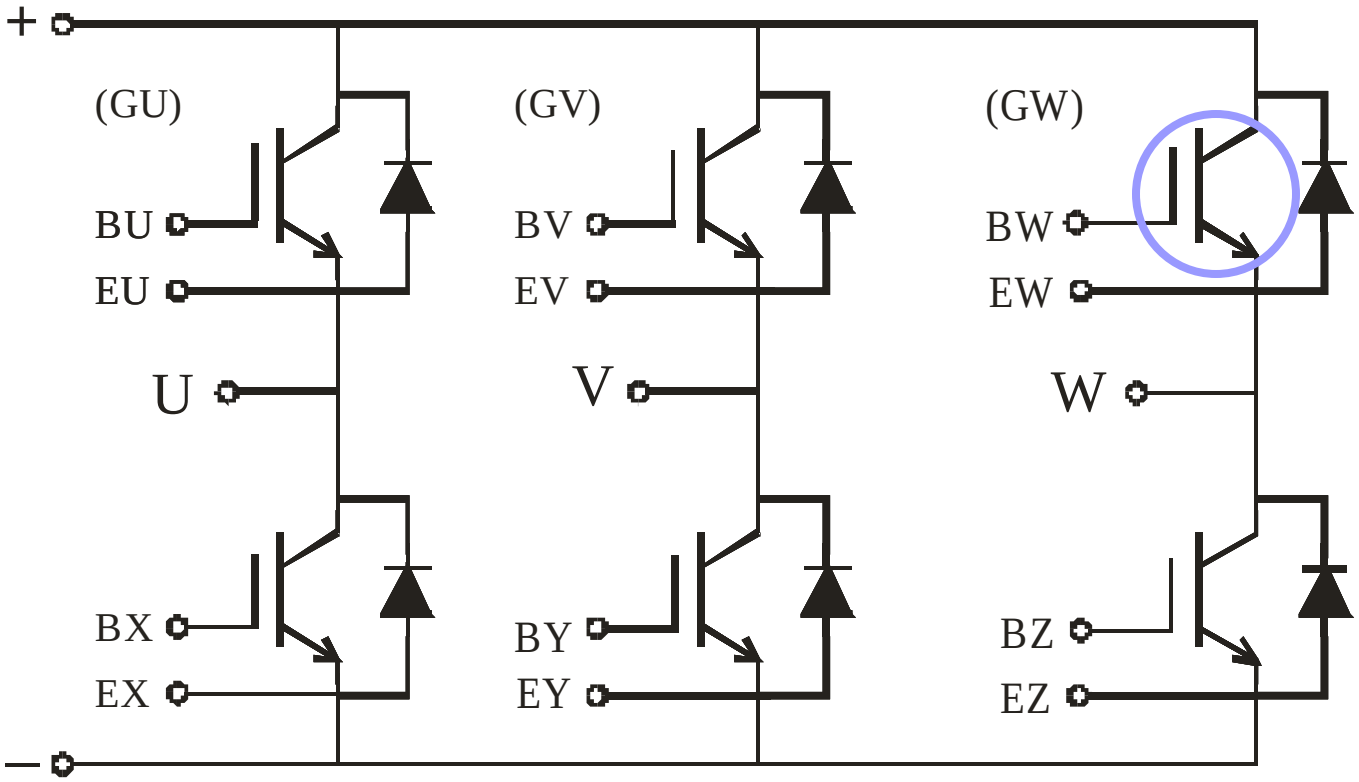
Population Distribution



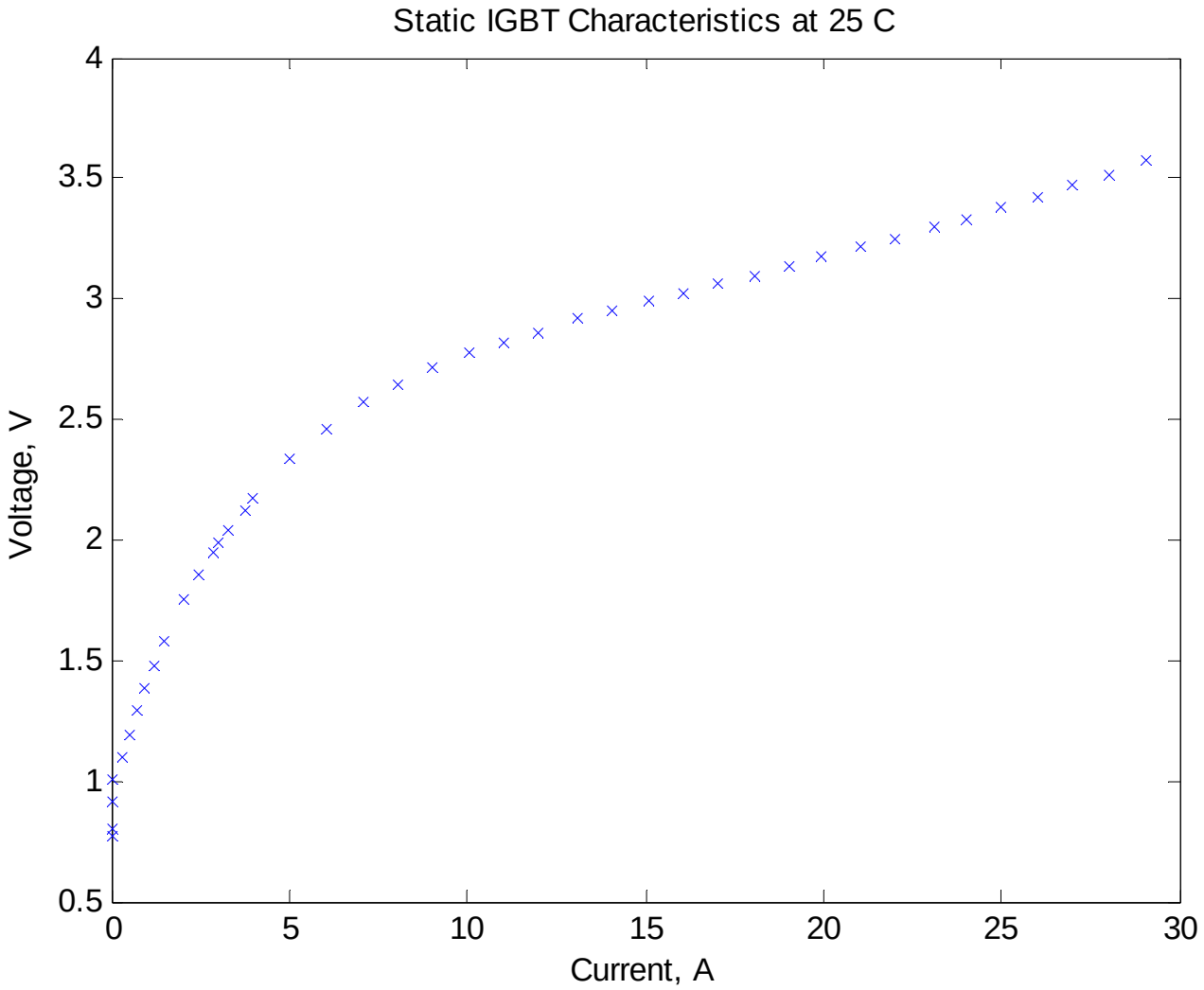
Curve Fitting Application: IGBT Conduction Loss

Problem Statement

- Characterize the i-v curve of the IGBT in a FUGI 6MBI 30L-060 IGBT module (gate voltage = 15 V)



Measured Data



Step 1: Decide on Representation

- Assumed that the voltage (v) can be expressed as the function of current (i) as

$$v = ai + (bi)^c$$

- Terms a, b, and c are constants to be identified using GOSET

Fitness Function

- Take set of K measurements
  - v_k voltage of k'th measurement
  - i_k current at k'th measurement
- Fitness function

$$f(a,b,c) = \frac{1}{\frac{1}{K}\sum_{k=1}^{K}\left|1-\frac{ai_k+(bi_k)^c}{v_k}\right|+10^{-3}}$$

igbt_fit.m

```
% this rountine returns the fitness of the IGBT conduction
% loss parameters based on the mean percentage error
function fitness = igbt_fit(parameters,data,fignum)

% assign genes to parameters
a=parameters(1);
b=parameters(2);
c=parameters(3);

vpred=a*data.i+(b*data.i).^c;
error=abs(1-vpred./data.v);
fitness=1.0/(1.0e-6+mean(error));
```

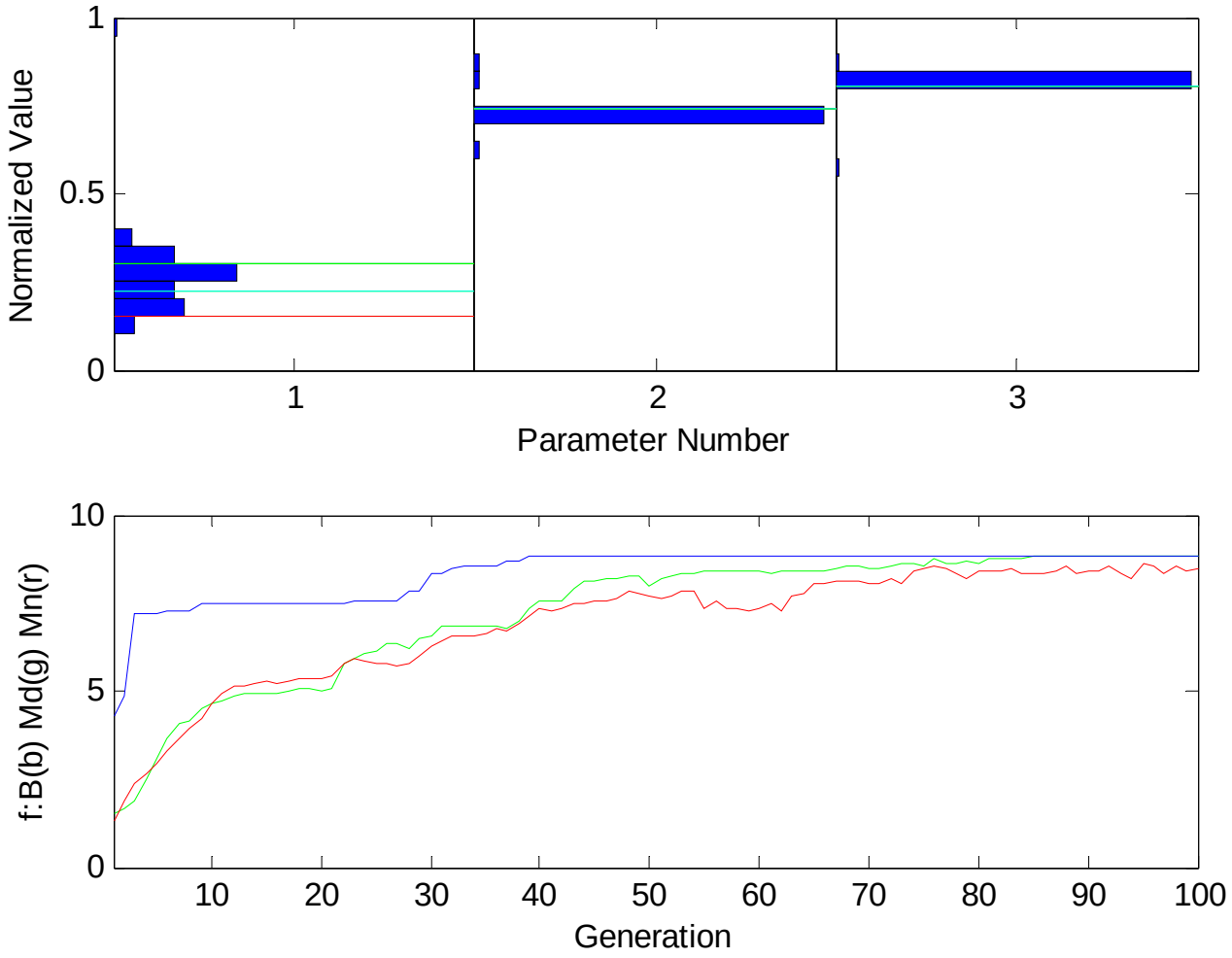
```
if nargin>2

   fitness

   figure(fignum);
   Npoints=200;
   ip=linspace(0,max(data.i),Npoints);
   vp=a*ip+(b*ip).^c;
   plot(data.i,data.v,'bx',ip,vp,'r')
   title('Voltage Versus Currrent');
   xlabel('Current, A');
   ylabel('Voltage, V');
   legend({'Measured','Fit'});

   figure(fignum+1);
   Npoints=200;
   plot(data.i,data.v.*data.i,'bx',ip,vp.*ip,'r')
   title('Power Versus Currrent');
   xlabel('Current, A');
   ylabel('Power, V');
   legend({'Measured','Fit'});

end
```

igbt

```
% load experimental data
load igbt_data.mat

% plot the raw data
figure(1);
plot(igbt_data.i,igbt_data.v,'x');
xlabel('Current, A');
ylabel('Voltage, V');
title('Static IGBT Characteristics at 25 C');

% gafit
GAP=gapdefault;

% set up regions
GAP.mg_nreg=4;
GAP.mg_tmig=20;
GAP.mg_pmig=0.05;
```

```
% declare the types of all the parameters
GAP.gd_min = [1e-8 1.0e-6 1.0e-3];
GAP.gd_max = [1e+2 1.0e+3 1.0e+0];
GAP.gd_type= [  3      3      3 ];
GAP.gd_cid = [  1      1      1 ];

% do the optimization
[fP,GAS]= gaoptimize(@igbt_fit,GAP,igbt_data,[],[],[]);

% plot the best fit
bestparameters=GAS.bestgenes(:,GAS.cg);
igbt_fit(bestparameters,igbt_data,3);
```

Evolution

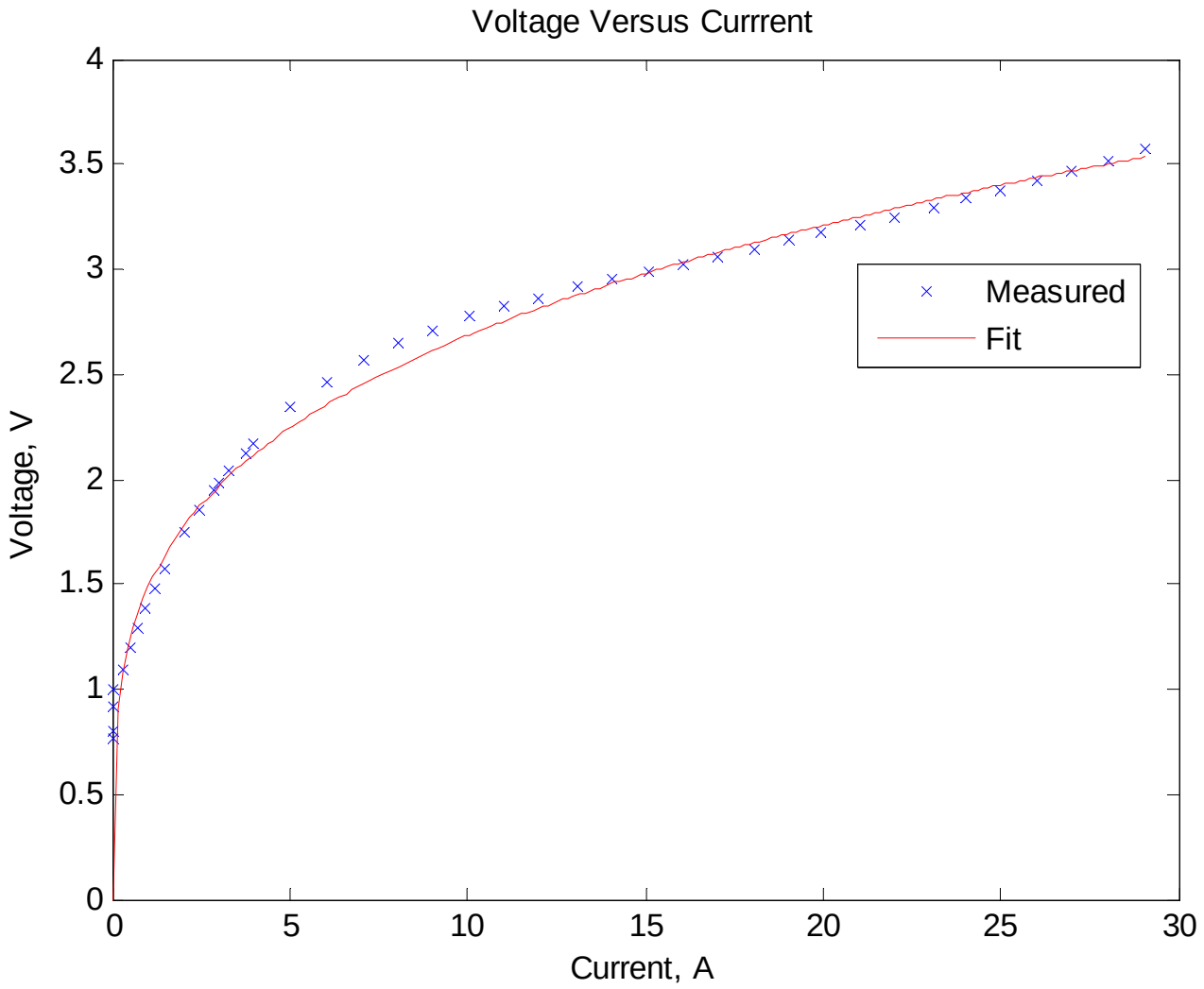

Normalized Value
1
0.5
0
1
2
3
Parameter Number
f:B(b) Md(g) Mn(r)
10
5
0
10
20
30
40
50
60
70
80
90
100
Generation

Result

- After 100 generations, the best individual has the following parameter values.

```
a = 0.00000031500713,
b = 4.57908764908758,
 c =0.25816413210782
```

Measured Data and Fit

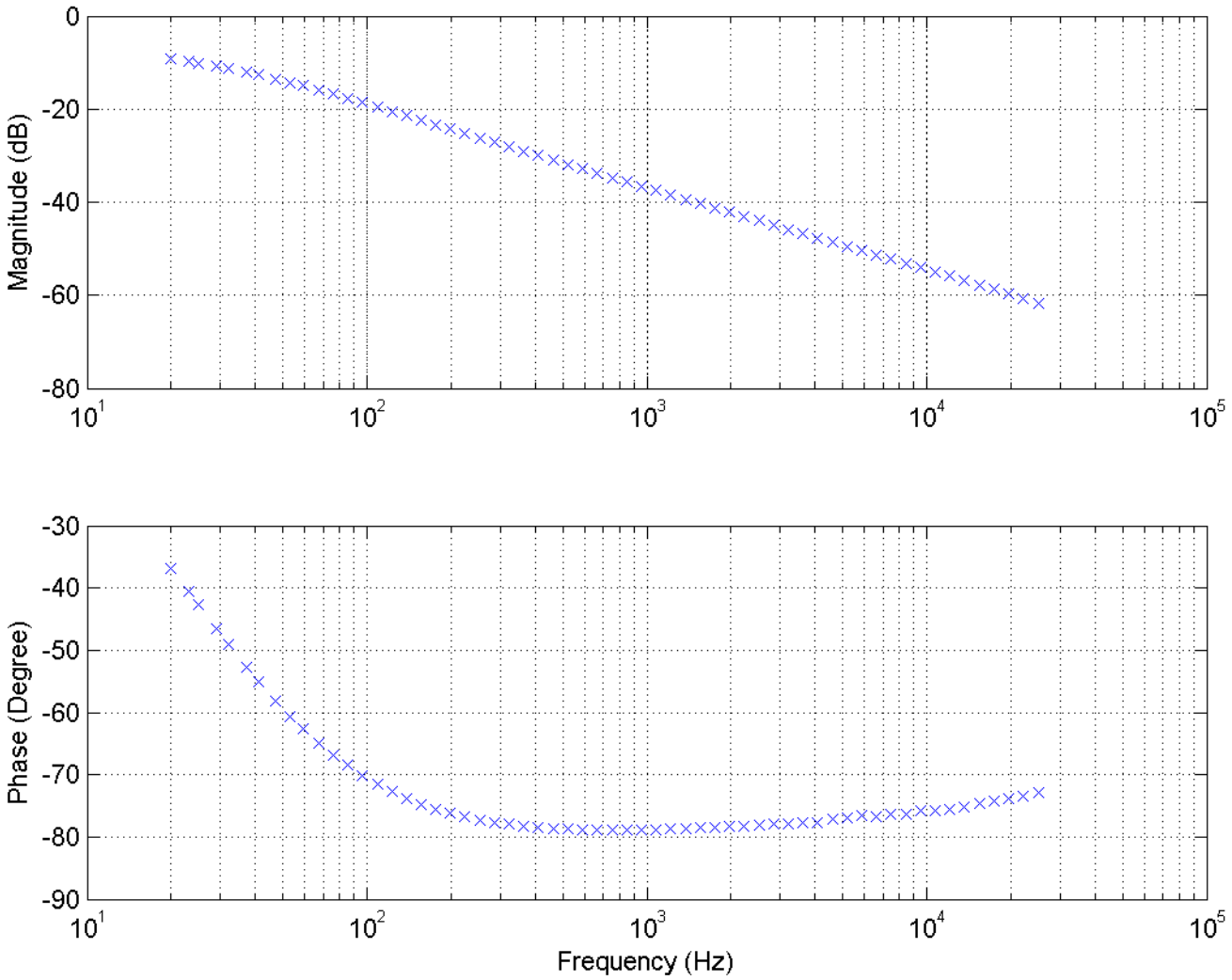


Transfer Function Fitting

Problem Statement

- Identify a transfer function to fit to measured frequency response data
- The transfer function values are admittances looking into the d-axis of brushless DC motor measured at 60 different frequencies

Measured Transfer Function



Transfer Function

- Since no pronounced peaks, let's assume

$$Y(s) = \frac{a_1}{\tau_1 s + 1} + \frac{a_2}{\tau_2 s + 1} + \cdots + \frac{a_n}{\tau_n s + 1}$$

- n is the order of the transfer function
- a_i and τ_i are the parameters to be identified

Fitness Function

- Let K be number of data points
- Let f_k is the k'th frequency
- Let Y_k be the k'th measured admittance
- Then

$$s_k = j2\pi f_k$$

$$F(a_1,\cdots,a_n,\tau_1,\cdots,\tau_n) = \frac{1}{\frac{1}{K}\sqrt{\sum_{k=1}^{K}\frac{\left|Y_k - Y(s_k)\right|}{\left|Y_k\right|}} + 10^{-12}}$$

Files

- ydata - process admittance data
- pftf - evaluate partial fraction transfer function
- pftf_fit - evaluate fitness of transfer function parameters
- pftf_id - main script
- bodeplot - makes a Bode plot

ydata.m

```
% SSFR Impedance Data of Brusless DC Machine
% Rotor is aligned with d-axis.

% frequency vector
f=[20 23 25 29 32 37 41 47 53 59 67 76 85 96 109 123 138 ...
   156 176 199 224 253 286 322 364 410 463 523 590 666 751 ...
   848 957 1079 1218 1375 1551 1750 1975 2229 2516 2839 ...
   3203 3615 4079 4603 5195 5862 6615 7465 8424 9506 10728 ...
   12106 13661 15416 17397 19632 22154 25000];

% resistance measurement from LCR meter
r_lcr=[3.494 3.517 3.533 3.566 3.594 3.64 3.679 3.74 3.804 ...
       3.871 3.964 4.078 4.193 4.329 4.503 4.7 4.916 5.185 5.493 ...
       5.861 6.268 6.753 7.317 7.939 8.685 9.494 10.457 11.523 ...
       12.768 14.193 15.74 17.512 19.651 21.868 24.627 27.256 31.01 ...
       34.455 38.79 43.689 48.77 55.253 61.25 69.14 77.31 89.04 ...
       100.39 114.58 126.56 144.22 160.26 185.05 206.62 232.2 265.52 ...
       307.6 352.7 408.7 465.6 544];

% inductance measurement from LCR meter
l_lcr=[20.781 20.751 20.697 20.638 20.585 20.518 20.457 20.379 ...
       20.301 20.232 20.144 20.05 19.942 19.839 19.728 19.613 19.498 ...
       19.367 19.23 19.08 18.93 18.767 18.593 18.419 18.231 18.044 ...
       17.843 17.64 17.428 17.208 16.996 16.777 16.536 16.322 16.079 ...
       15.874 15.614 15.403 15.165 14.928 14.711 14.466 14.264 14.03 ...
       13.815 13.546 13.323 13.076 12.895 12.664 12.482 12.243 12.077 ...
       11.911 11.739 11.579 11.46 11.364 11.31 11.285]*1e-3;

% convert to d-axis admittance
s=1j*2*pi*f;     % Complex frequency
rd=2*r_lcr/3;    % D-Axis Resistance
ld=2*l_lcr/3;    % D-Axis Inductance
zd=rd+s.*ld;     % D-Axis Impedance
yd=1./zd;        % D-Axis Admittance
```

pftf.m

```
function t = pf_tf(s,a,tau)
% PF_tf calculates transfer function values for a transfer function
%       with real poles in partial fraction expansion form.
%
% t=pf_tf(s,a,tau)
%
% Inputs:
% s       = vector of complex frequencies
% a       = vector of dc gains
% tau     = vector of time constants
%           (should have same dimension as a)
% Outputs:
% t       = vector of transfer function values
%           (same dimension as s)
%
% Written by:
% S.D. Sudhoff
% School of Electrical and Computer Engineering
% 1285 Electrical Engineering Building
% West Lafayette, IN 47907-1285
% Phone: 765-494-3246
% Fax: 765-494-0676
% E-mail: sudhoff@ecn.purdue.edu

n=length(a);

if n~=length(tau)
   error('alpha and tau vectors do not correspond');
end

t=zeros(size(s));
for i=1:n,
   t=t+a(i)./(tau(i)*s+1);
end
```

pftf_fit

```
function fit=pftf_fit(parameters,data)
% pftf_fit calculates fitness of transfer function fit
%
% t=pftf_tf(parameters,data)
%
% Inputs:
% parameters = paramameter vector
%              (first gains, then time constants)
% data       = data.s - complex frequency vector
%              data.t - transfer function values
% Outputs:
% t       = vector of transfer function values
%           (same dimension as s)
%
% Written by:
% S.D. Sudhoff
% School of Electrical and Computer Engineering
% 1285 Electrical Engineering Building
% West Lafayette, IN 47907-1285
% Phone: 765-494-3246
% Fax: 765-494-0676
% E-mail: sudhoff@ecn.purdue.edu
% partial fraction transfer function fitness

o=(length(parameters))/2;

a  =parameters(1:o);
tau=parameters(o+1:2*o);

tpred = pftf(data.s,a,tau);

terror= data.t-tpred;

error=norm(terror./abs(data.t))/length(tpred);

fit=1.0/(1.0e-12+error);
```

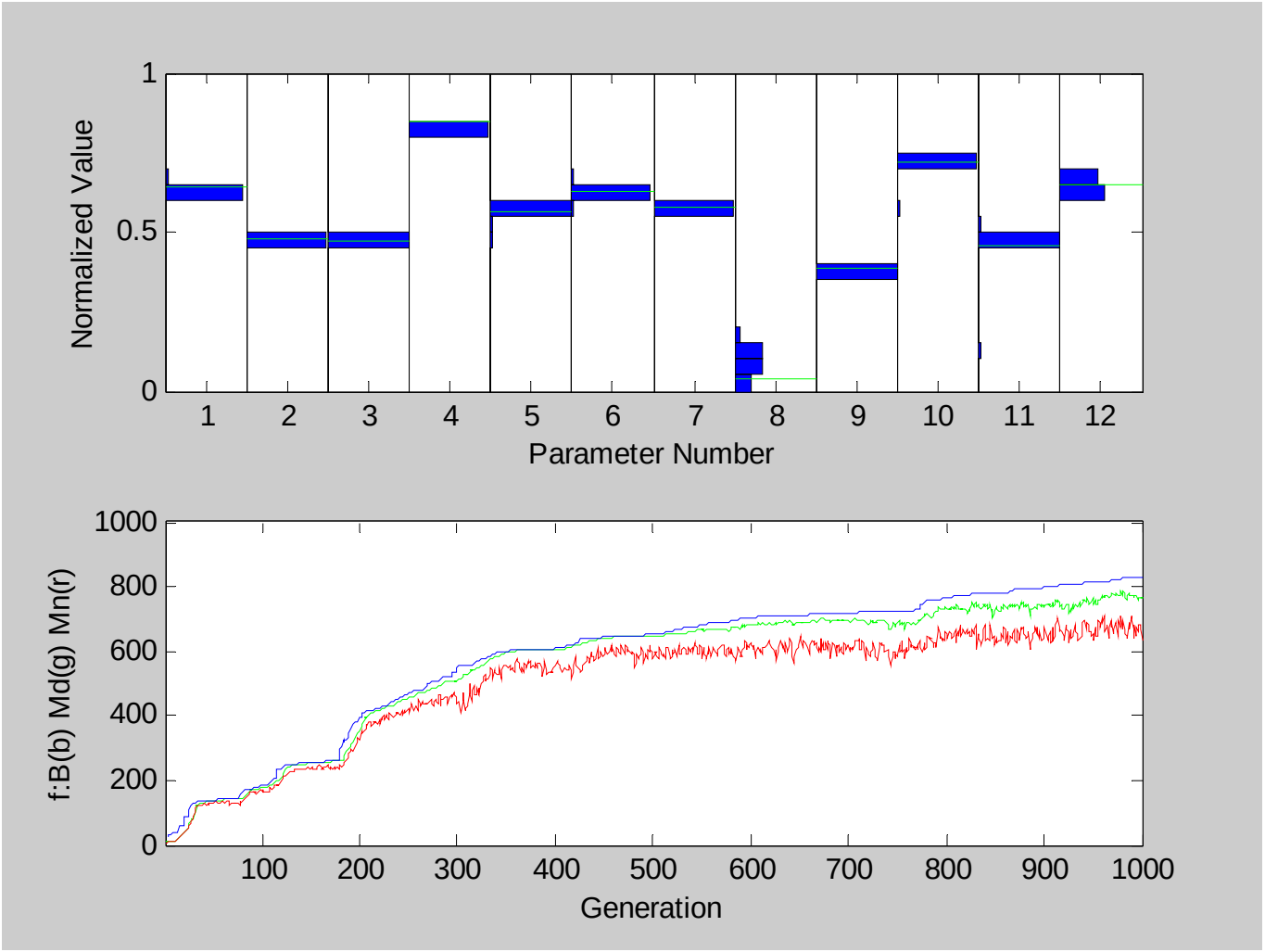
tffit.m

```
% TFFIT.M
%
% Transfer function fitting fitness function

function f = tffit(parameters,data)

a = parameters(1:6);
tau = parameters(7:12);

tpred = zeros(size(data.s));
for i = 1:6,
   tpred = tpred+a(i)./(tau(i)*data.s+1);
end

terror = data.t-tpred;
error = norm(terror./abs(data.t))/length(tpred);

f = 1.0/(1.0e-12+error);        % Fitness function
```

pftf_id

```
% Simple Example in Transfer Function Identification

% Get admittance data
ydata;
data.s= s;
data.t=yd;

% Initialize the parameters
GAP=gapdefault;
GAP.fp_ngen=1000;

% Set range for parameters
order=6;                            % order of transfer function
dim=ones(1,order);                  % dimensioning vector
amin=1.0e-8;                        % minimum gain
amax=1.0e+1;                        % maximim gain
taumin=1.0e-8;                      % minimum time constant
taumax=1.0e+0;                      % maximum time constant
GAP.gd_min = [ amin*dim taumin*dim ];
GAP.gd_max = [ amax*dim taumax*dim ];
GAP.gd_type= [    3*dim      3*dim ];
GAP.gd_cid = [    1*dim      1*dim ];

% Optimize the fit
[P,GAS]= gaoptimize(@pftf_fit,GAP,data,[],[],[]);

% Plot the results
bestgenes=GAS.bestgenes(:,GAP.fp_ngen);
a   = bestgenes(1:order);
tau = bestgenes(order+1:2*order);
ydp=pftf(s,a,tau);
a
tau
figure(2);
bodeplot(2,f,yd,'bx',ydp,'r-');
```

Sample Evolution



Resulting Fit