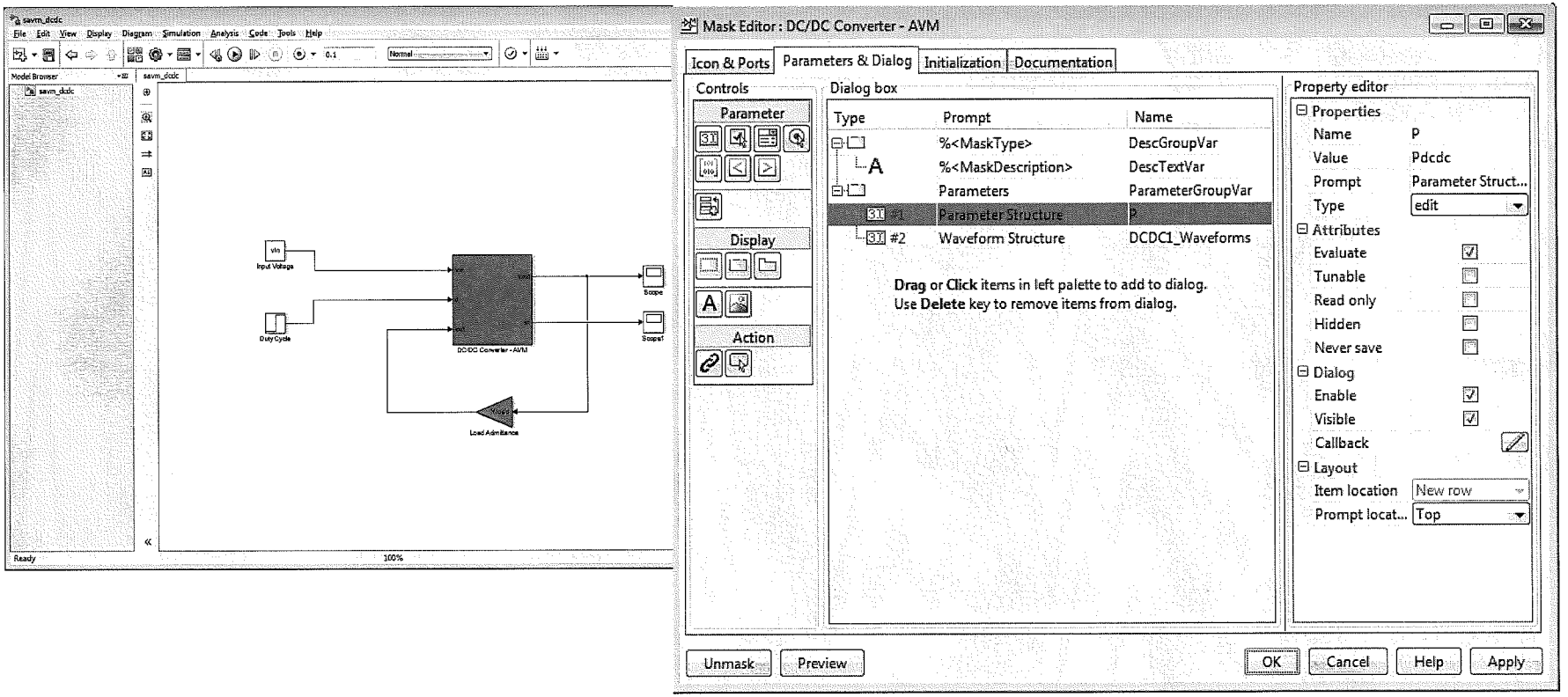
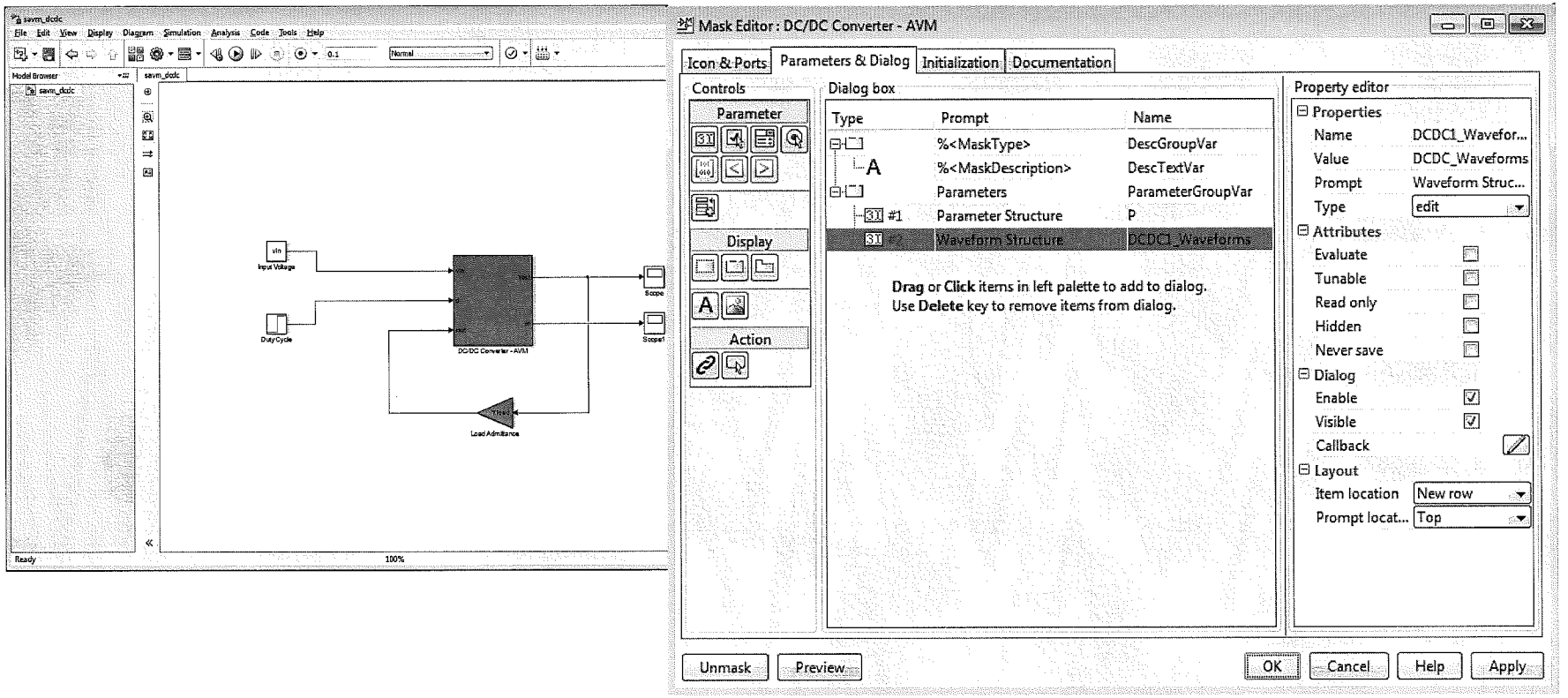
savm_dcdc



Right Click on DC/DC Converter – AVM – Mask – Edit Mask

savm_dcdc

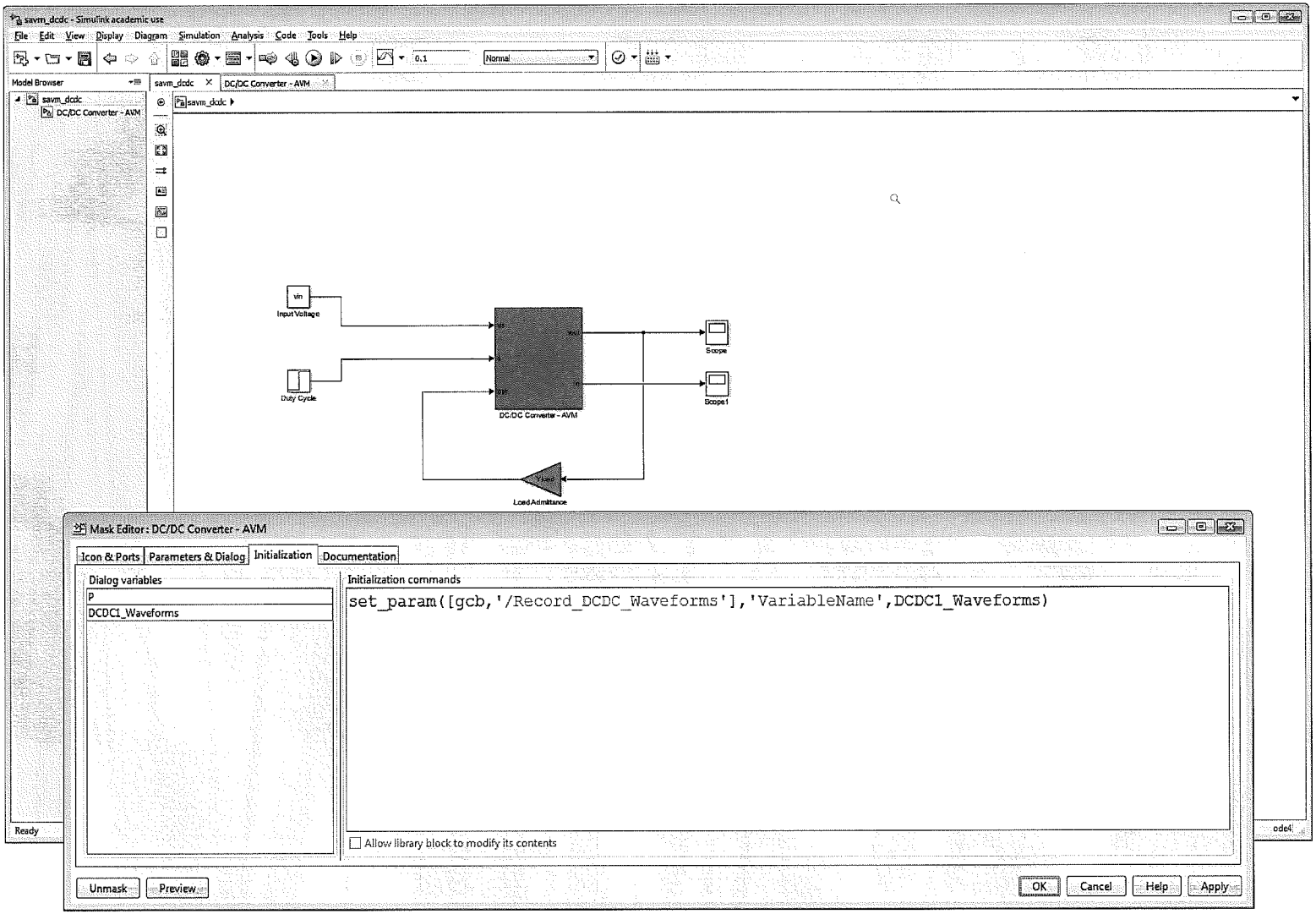


Right Click on DC/DC Converter – AVM – Mask – Edit Mask

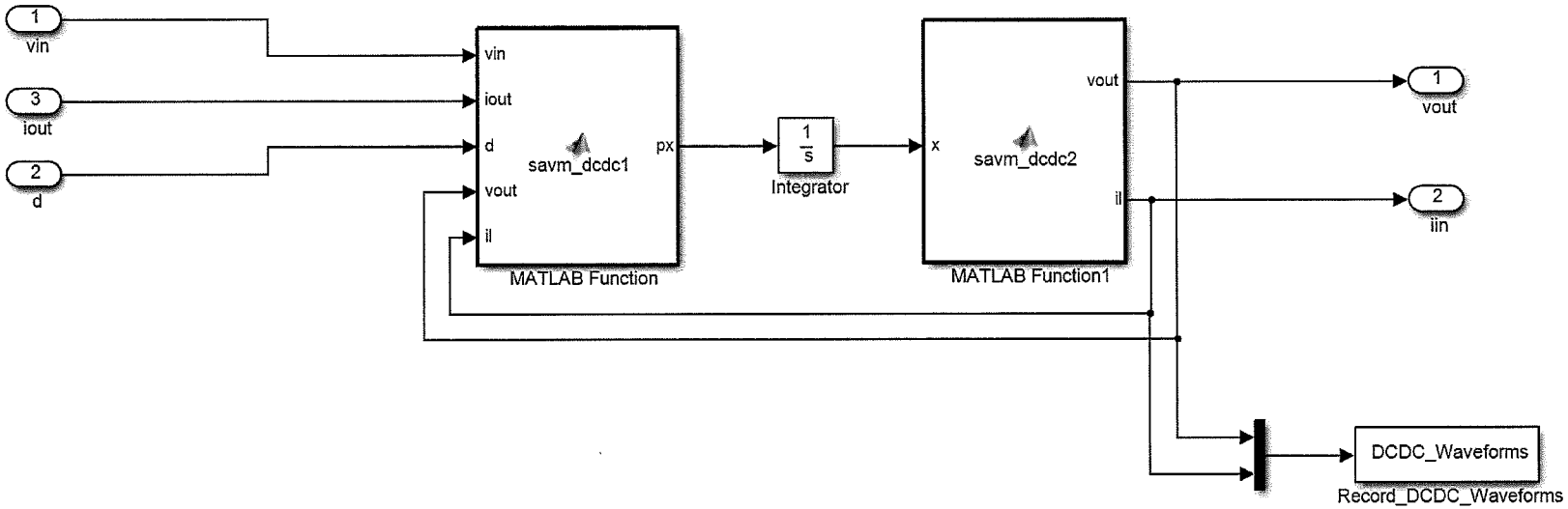
Creating A Subsystem

- Select Portion of Diagram of Interest
- Select *Diagram – Model & Subsystem Reference – Create Subsystem from Selection*

savm_dcdc

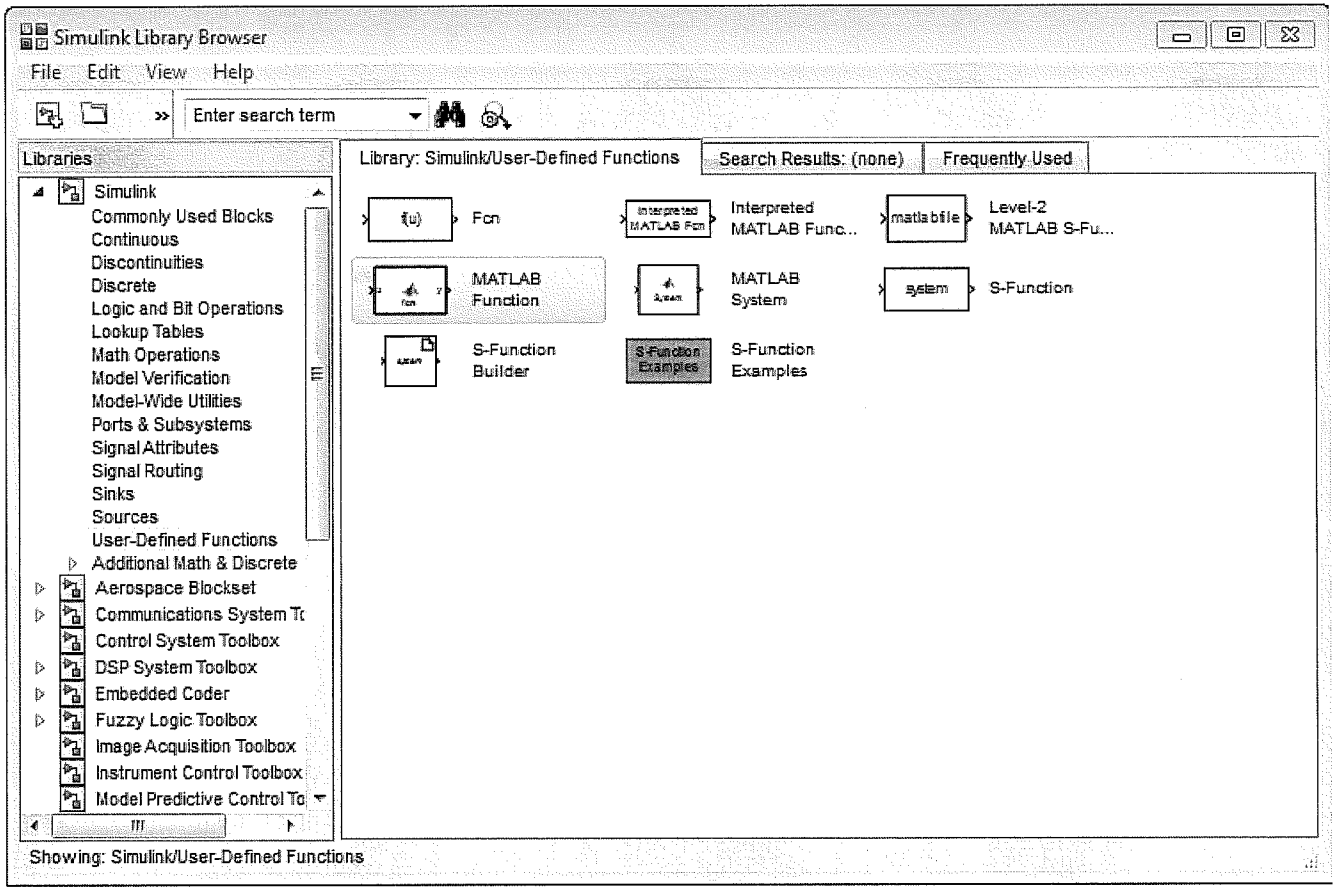

savm_dcdc - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
Model Browser
savm_dcdc
DC/DC Converter - AVM
Input Voltage
Duty Cycle
DC/DC Converter - AVM
Scope
Scope1
Load Admittance
Ready
Mask Editor: DC/DC Converter - AVM
Icon & Ports
Parameters & Dialog
Initialization
Documentation
Dialog variables
P
DCDC1_Waveforms
Initialization commands
set_param([gcb,'/Record_DCDC_Waveforms'],'VariableName',DCDC1_Waveforms)
Allow library block to modify its contents
Unmask
Preview
OK
Cancel
Help
Apply

savm_dcdc


1
vin
3
iout
2
d
vin
iout
d
vout
il
savm_dcdc1
px
MATLAB Function
1
s
Integrator
x
savm_dcdc2
vout
il
MATLAB Function1
1
vout
2
iin
DCDC_Waveforms
Record_DCDC_Waveforms

Creating A Block

- Select *View* then *Library Browser*



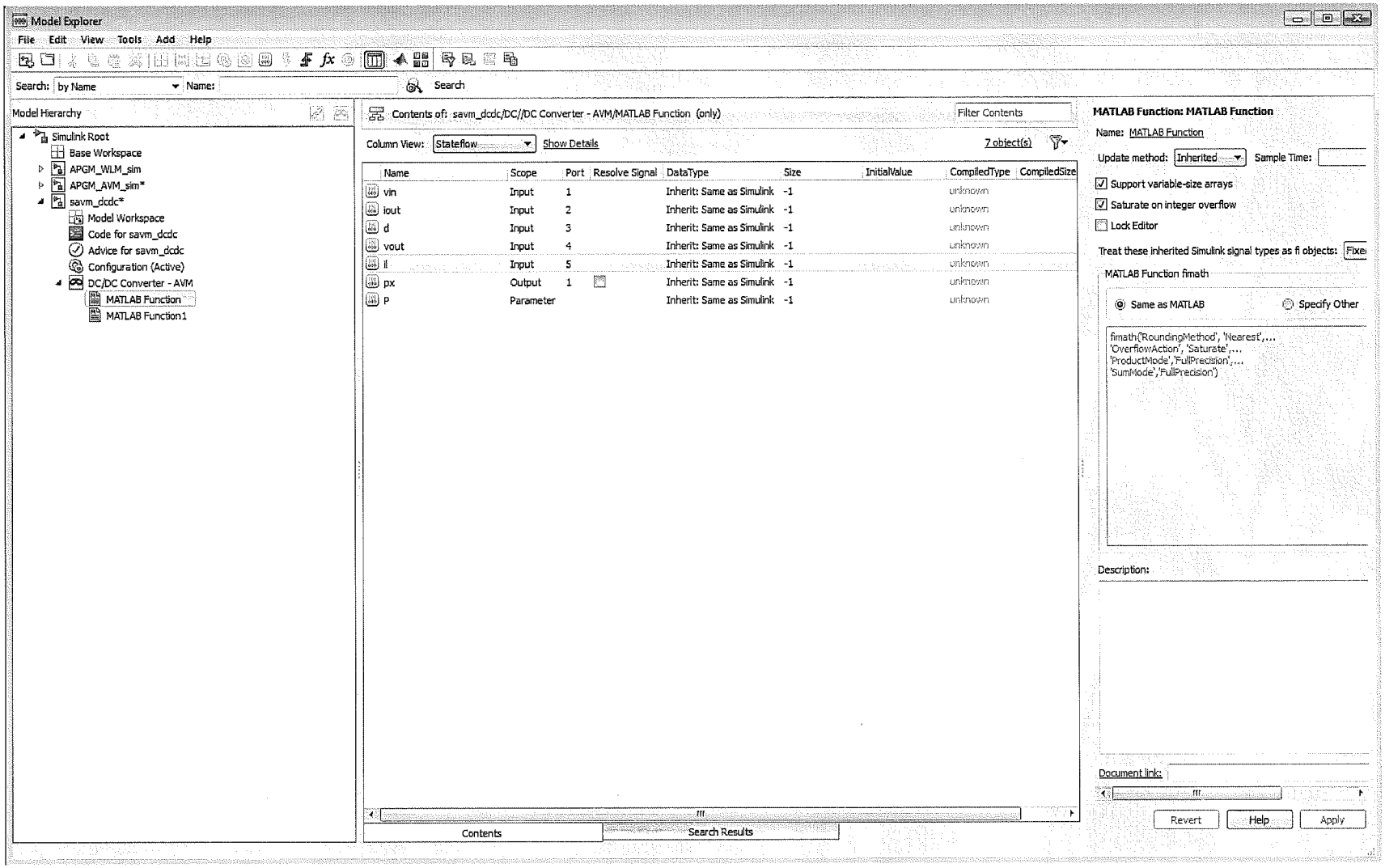
savm_dcdc1

```
function px = savm_dcdc1(vin,iout,d,vout,il,P)
% This routine contains the dynamics of an simple dc/dc (boost) converter
% model. To be used with savm_dcdc2.
%
% Inputs:
% vin       = input voltage (V)
% iout      = output curent (A)
% d         = duty cycle
% P         = structure with parameters
%  P.L      = inductor inductance (H)
%  P.rL     = inductor series resistance (Ohms)
%  P.C      = capacitor capacitance (F)
%
% Outputs:
% px        = time derivative of state vector
%  px(1)    = time derivative of fast average of inductor current (A)
%  px(2)    = time derivative of fast average of output voltage (V)
%
```

savm_dcdc1

```
% Internal:
% vls       = average voltage across lower switch/diode (V)
% ius       = average current into upper switch/diode (A)
% pil       = time derivative of il (A/s)
% pvout     = time derivative of vout (V/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

% variables of interest
vls=d*vout;
ius=d*il;

% compute derivatives of state
pil=(vin-P.rL*il-vls)/P.L;
pvout=(ius-iout)/P.C;

% pack the state vector
px=[pil pvout]';
```

Matlab Function I/O

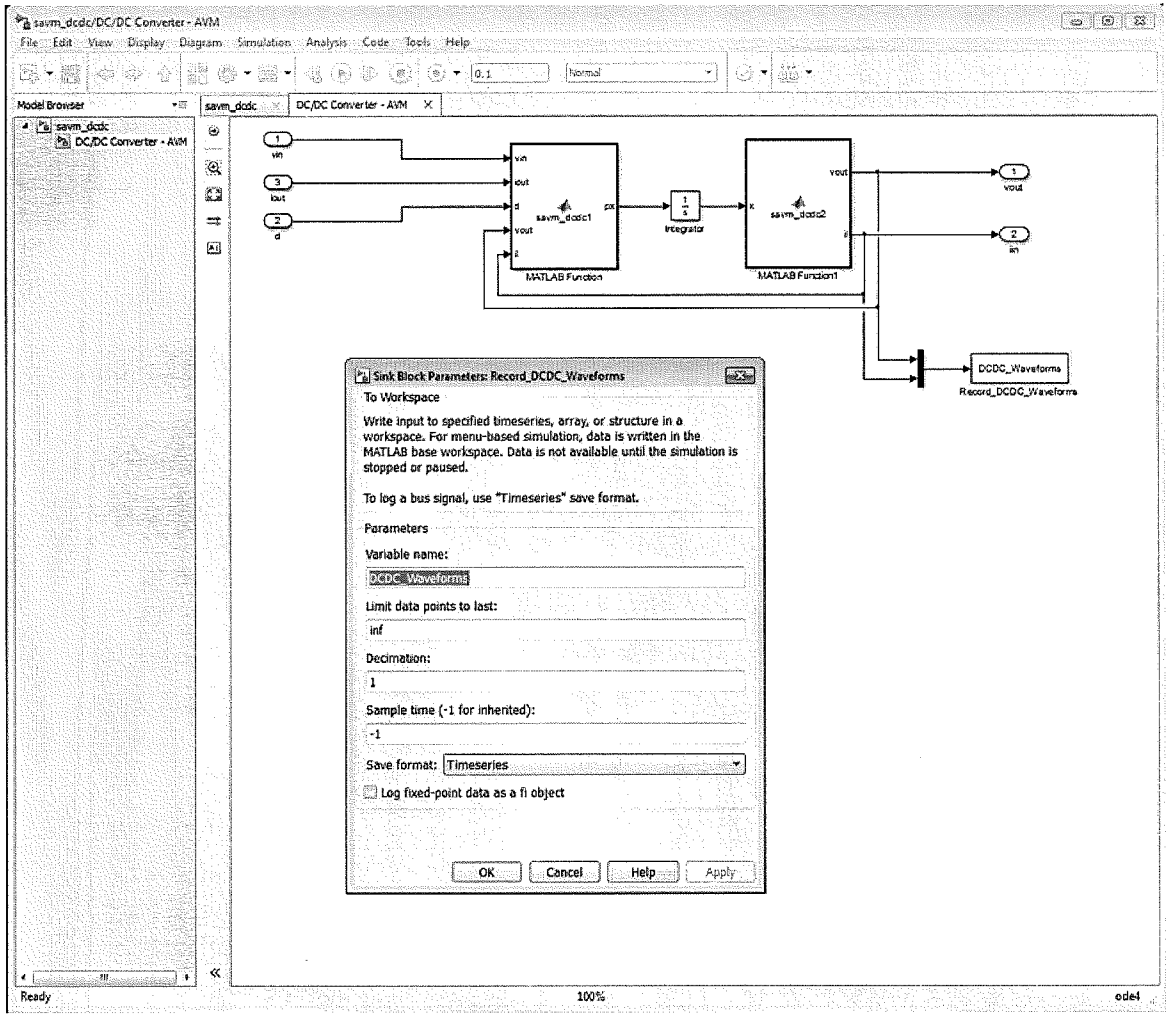


Right Click on MATLAB Function – Explore – Click on Scope to Change

savm_dcdc2

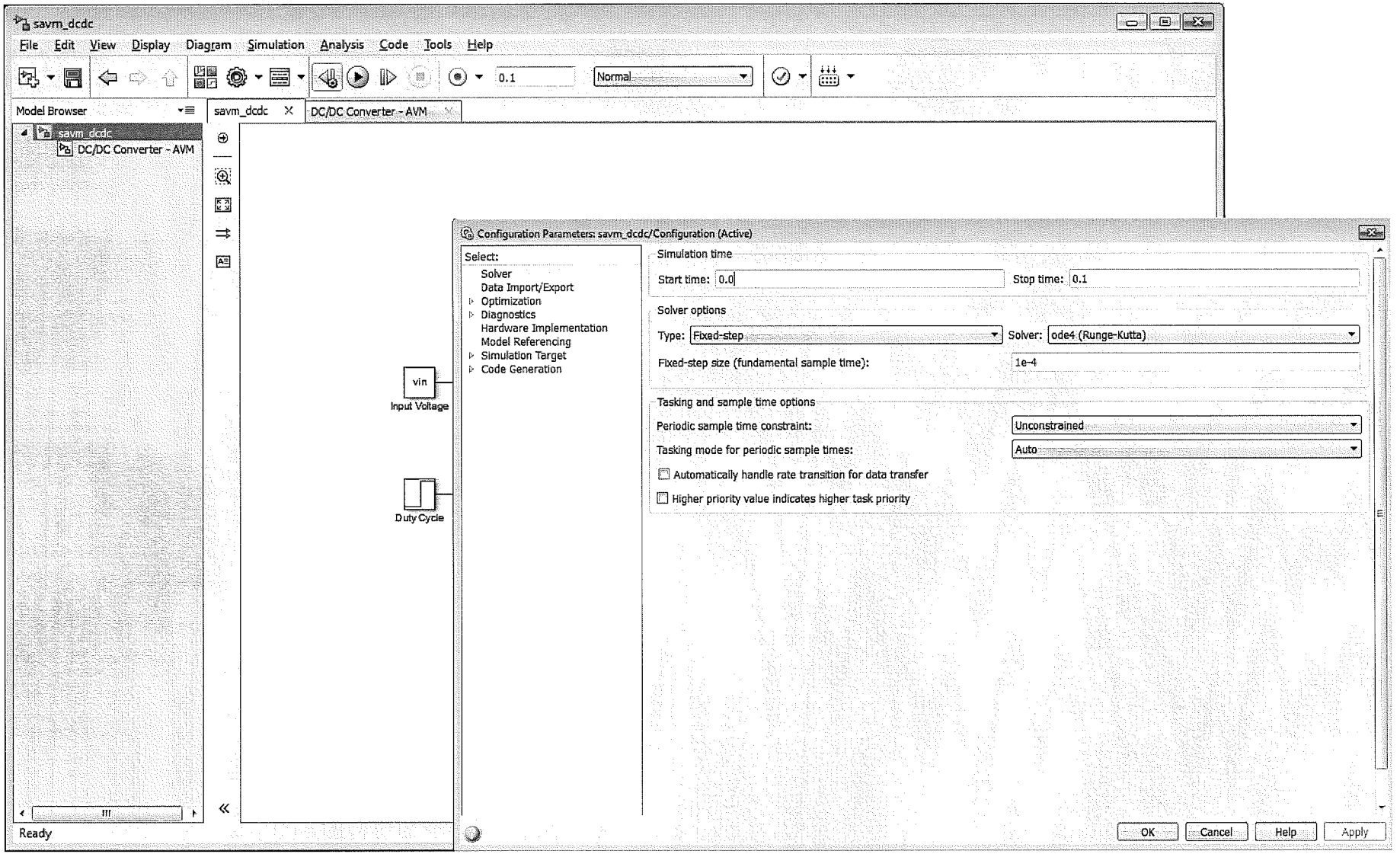
```
function [vout,il] = savm_dcdc2(x)
% This routine decomposes the state vector in a simple average value
% model of a dc/dc (boost) converter. To be used with savm_dcdc1.
%
% Inputs:
% x         = state vector
%  x(1)     = fast average of inductor current (A)
%  x(2)     = fast average of output voltage (V)
%
% Outputs:
% vout      = output voltage (V)
% il        = inductor (input) current (A)
%
% 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

% break apart state vector
il=x(1);
vout=x(2);
```

Output to Workspace



View – Library Browser – Simulink – Sinks – To Workspace

Simulation Settings



Simulation – Model Configuration Parameters

savm_dcdc_plot

```
function savm_dcdc_plot(Name,Wf,fn)
% savm_dcdc_plot plots waveforms associated with a dc/dc converter
%
% savm_dcdc_plot(Name,Wf)
% savm_dcdc_plot(Name,Wf,fn)
%
% Inputs:
% Name           = text Description of component
% Wf             = structure of waveform data
%  Wf.Time       = time vector (s)
%  WF.Data(:,1)  = output voltage (V)
%  Wf.Data(:,2)  = inductor current (A)
% fn             = figure number
%
% Internal:
% t              = time vector (s)
% il             = inductor current (A)
% vout           = output voltage (V)
%
% Written by:
% S.D. Sudhoff
% Purdue University
% 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
```

savm_dcdc_plot

```
% Get data
t=Wf.Time;
vout=Wf.Data(:,1);
il   =Wf.Data(:,2);

% plot dc output current
if nargin>2
   figure(fn)
else
   figure
end
plot(t,vout);
xlabel('t, s');
ylabel('v_{out}, V');
grid on;
title([Name, ' DC Output Voltage']);

% plot torque
if nargin>2
   figure(fn+1)
else
   figure
end
plot(t,il);
xlabel('t, s');
ylabel('i_l, A');
grid on;
title([Name, ' Inductor Current']);
```

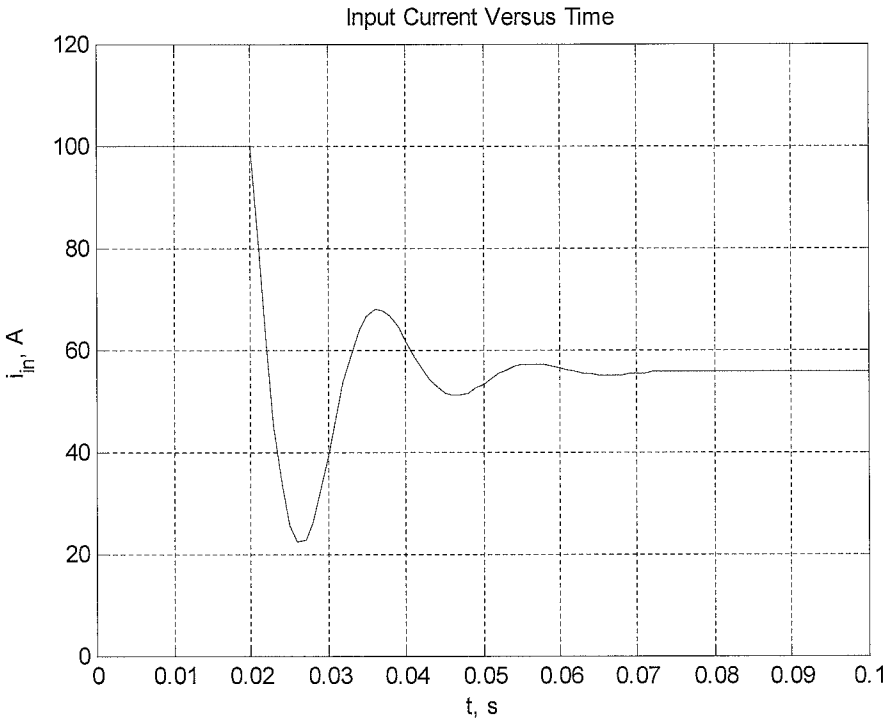
savm_dcdc_pp

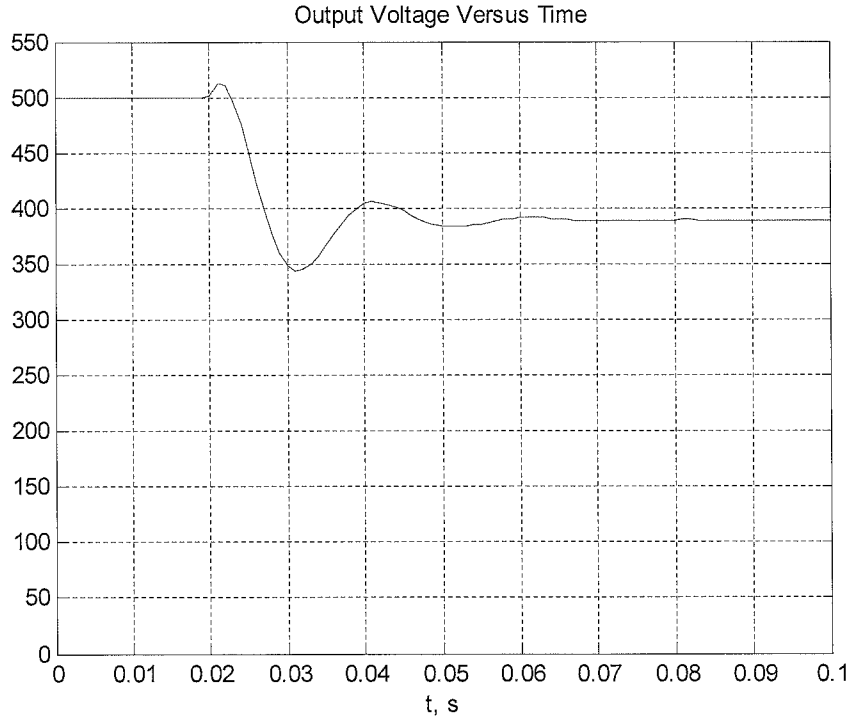
```
% This script imports and plots waveforms from the
% simple simulink average value dc/dc converter simulation savm_dcdc

% 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

savm_dcdc_plot('DCC Converter',DCDC_Waveforms,10);
```

savm_dcdc_study

```
% This script sets parameter and performs initial calculations for
% simple simulink average value dc/dc converter simulation, runs the
% study, and then plots the results

% 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

% initialize parameters
savm_dcdc_setup;

% run the study
sim('savm_dcdc');

% plot the results
savm_dcdc_plot('DCC Converter',DCDC_Waveforms,10);
```

Results


Input Current Versus Time
120
100
80
60
40
20
0
i_{in}, A
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
t, s


Output Voltage Versus Time
550
500
450
400
350
300
250
200
150
100
50
0
0
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0.09
0.1
t, s

Sorting

- Recall: To predict a future value of state, we needed to find the present value of the time derivative of the state variables, based on what we know – that is the present value of state variables and present value of input variables
- Sorting is the evaluation order we need to follow when doing this

Sorting

- Consider the system

$$\frac{dx_1}{dt} = 3x_2 - x_1 + g \quad (3)\ (4)$$

$$\frac{dx_2}{dt} = -3x_2 + z \quad (4)\ (3)$$

$$g = t^2 + 2x_1 + z \quad (2)$$

$$z = x_1^2 x_2 \quad (1)$$

$$x_1 = 3x_2 - \frac{dx_1}{dt} + g$$

Algebraic Loops

- Algebraic Loops occur when the system model is formatted such that it cannot be sorted
- Consider

$$\frac{dx_1}{dt} = -x_1^2 + x_2 + y$$

$$\frac{dx_2}{dt} = -5x_2 + z$$

$$y = 4z + x_1$$

$$z = 2y - x_2$$

Breaking Algebraic Loops

- Method 1 – Model Reformulation
- Start With

$$\frac{dx_1}{dt} = -x_1^2 + x_2 + y \quad (3)(4)$$

$$\frac{dx_2}{dt} = -5x_2 + z \quad (4)(3)$$

$$y = 4z + x_1 \quad (2)$$

$$z = 2y - x_2$$

$$z = 2(4z + x_1) - x_2$$

$$z = 8z + 2x_1 - x_2$$

$$7z = x_2 - 2x_1$$

$$z = \frac{1}{7}(x_2 - 2x_1) \quad (1)$$

Breaking Algebraic Loops

- Method 2 – Cheating
- Start With

F(x,t)

$$\frac{dx_1}{dt} = -x_1^2 + x_2 + y$$

$$\frac{dx_2}{dt} = -5x_2 + z$$

$$y = 4z + x_1$$

$$z = 2y - x_2$$

x_3 (1) (2)

$$\frac{dx_3}{dt} = (z - x_3)/\gamma$$

Stiffness

→ Assume a linear system

→ Assume all real eigenvalues

→ Assume no repeated eigenvalues

$$\dot{x} = Ax + Bu$$

$$y = M'x$$

M = eigenvectors of A

$$M = [v_1 \; v_2 \; \cdots \; v_N]$$

$$Av_i = \lambda_i v_i$$

$$\dot{y} = M'[Ax + Bu] = M'[AMy + Bu]$$

Stiffness

$$py = \dot{y} = M^{-1}AMy + \overbrace{M^{-1}Bu}^{\hat{u}}$$

$$py = M^{-1}AMy + \hat{u}$$

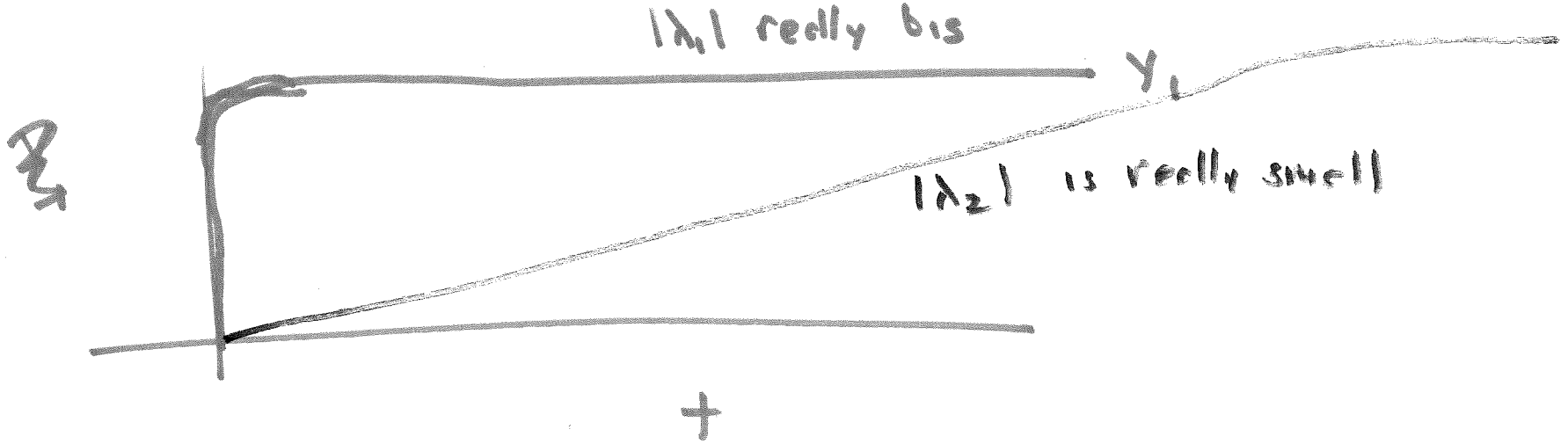
$$M^{-1}AM = [v_1 \ v_2 \ \cdots v_N]^{-1} A [v_1 \ v_2 \ \cdots v_N]$$

$$= [v_1 \ v_2 \ \ v_N]^{-1} [\lambda_1 v_1 \ \lambda_2 v_2 \ \ \lambda_N v_N]$$

$$= [v_1 \ v_2 \ v_N]^{-1} [v_1 \ v_2 \cdots \ v_N] \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_N \end{bmatrix}$$

Stiffness



Stiffness

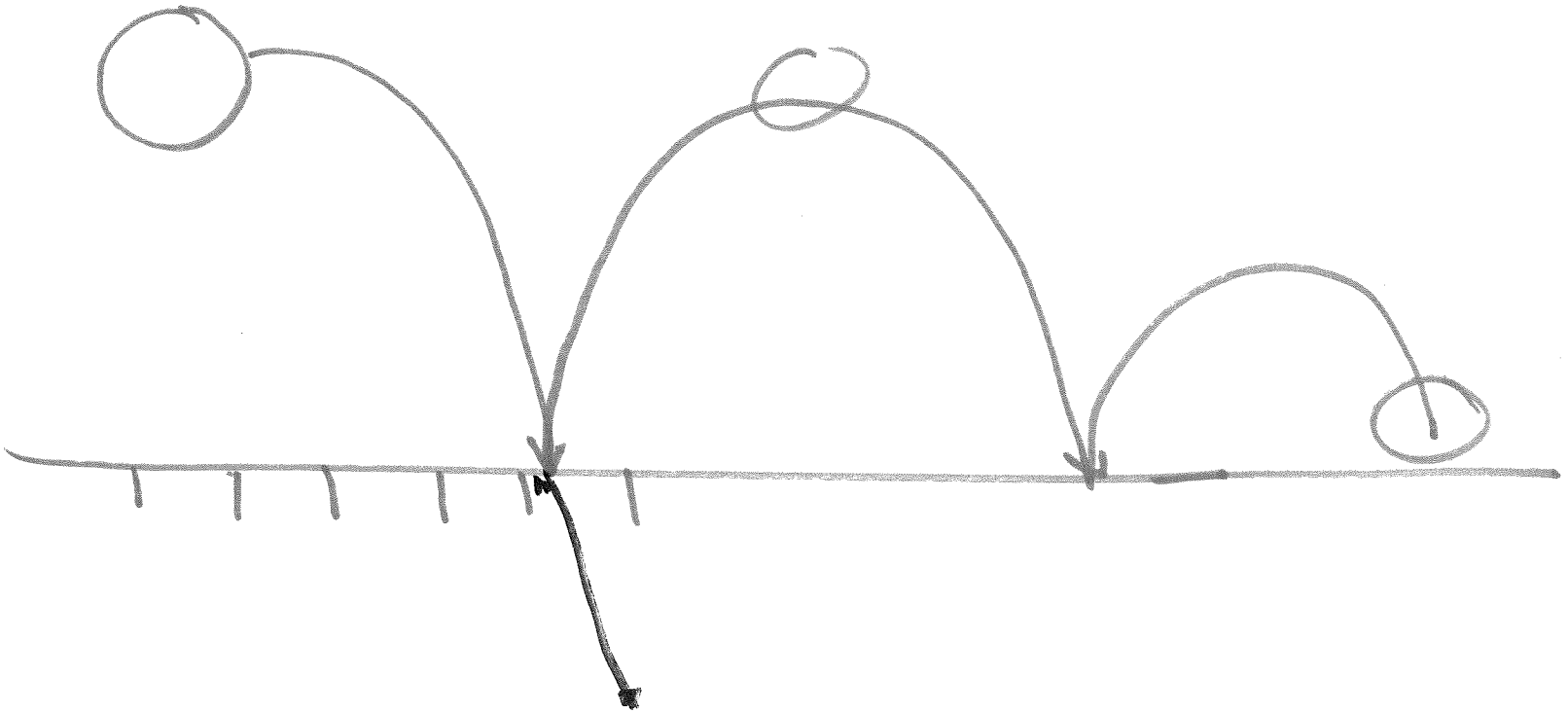
- In general

$$\left|\mathrm{Re}(\bar{\lambda})\right| \geq \left|\mathrm{Re}(\lambda_i)\right| \geq \left|\mathrm{Re}(\underline{\lambda})\right|$$

$$\mathrm{SR} = \frac{\left|\mathrm{Re}(\bar{\lambda})\right|}{\left|\mathrm{Re}(\underline{\lambda})\right|}$$

Other Simulation Issues

- Discrete Events
- Scheduling
- Paralleling Simulations



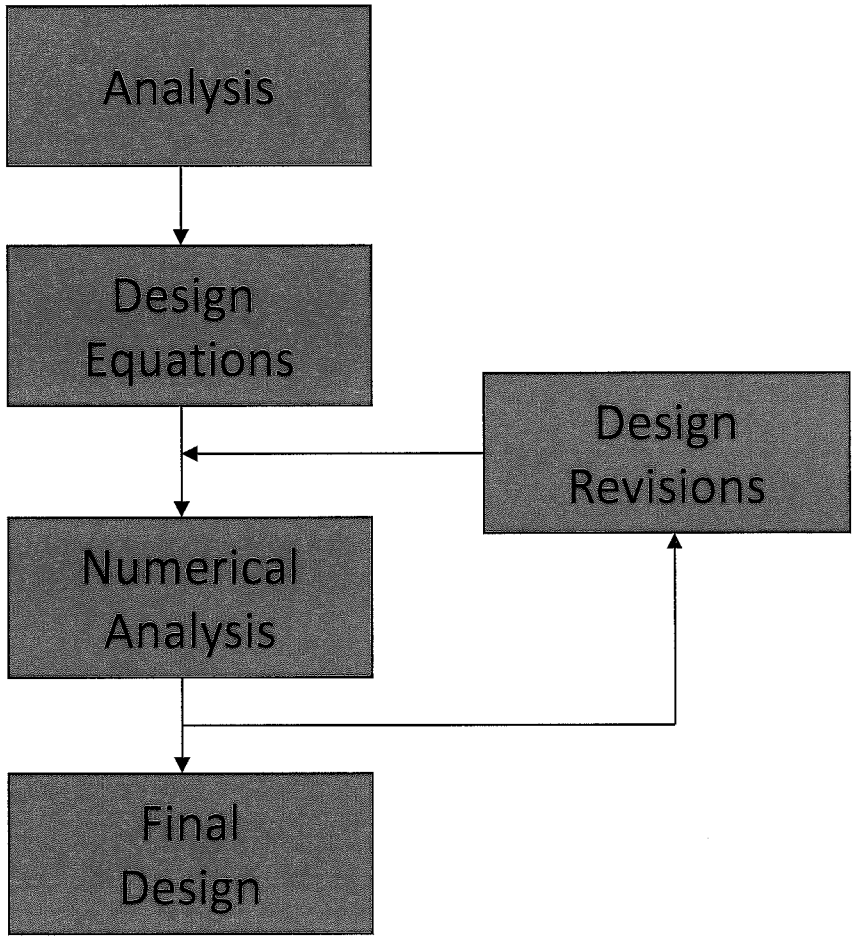
ECE61016 Power Electronics Converters and Systems

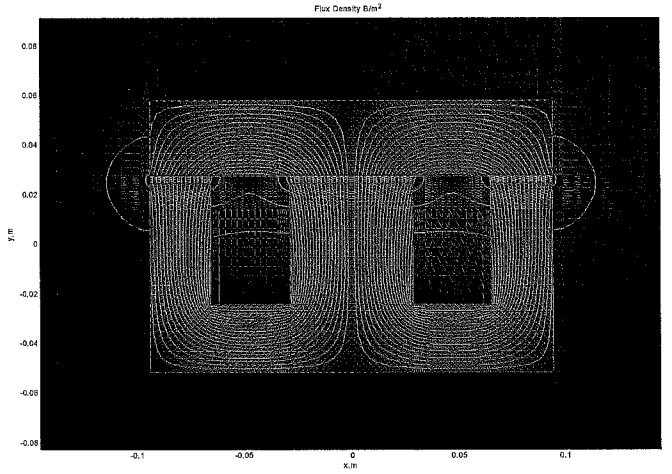
Lecture Set 2: Optimization

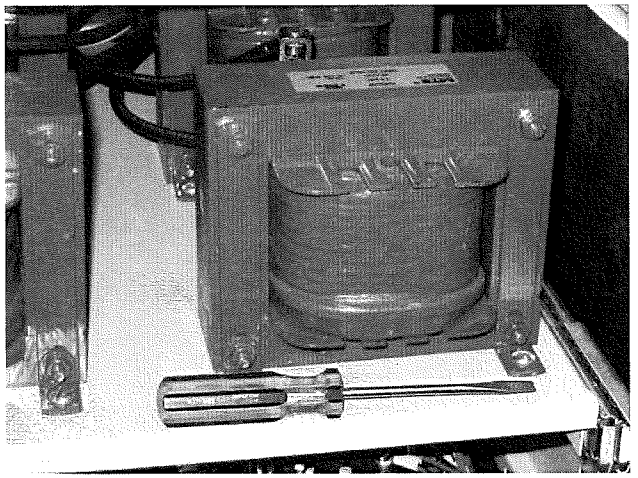
S.D. Sudhoff

Fall 2016

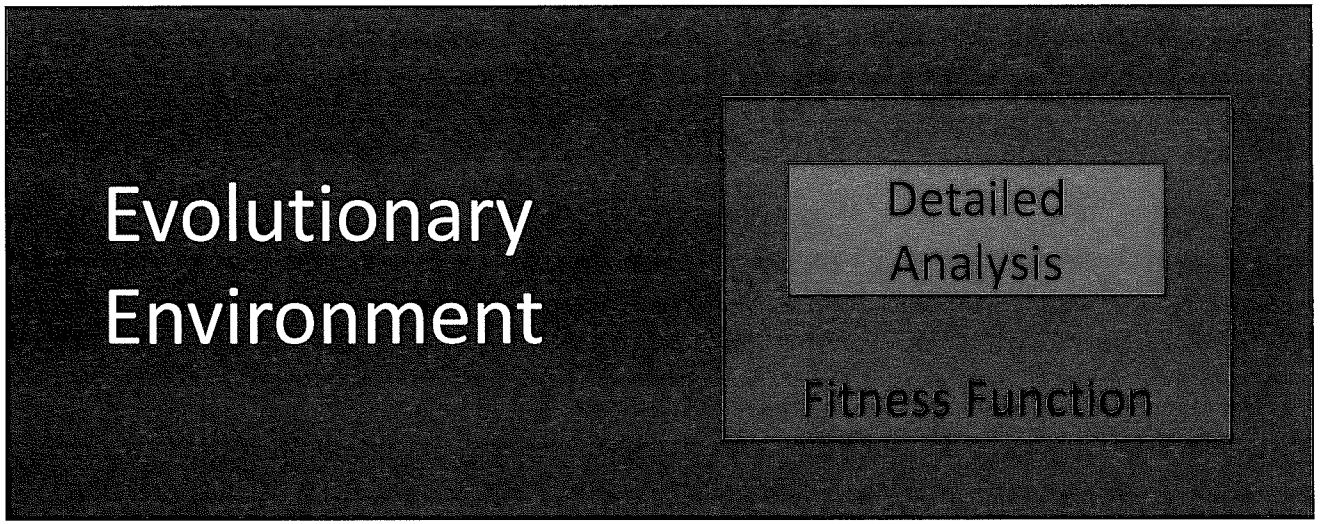
Manual Design Approach





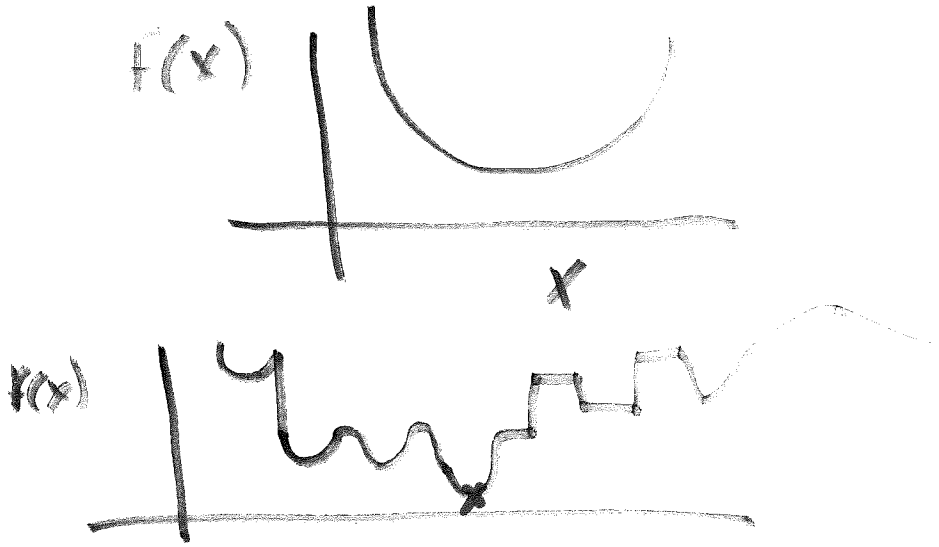


Evolutionary Design Approach



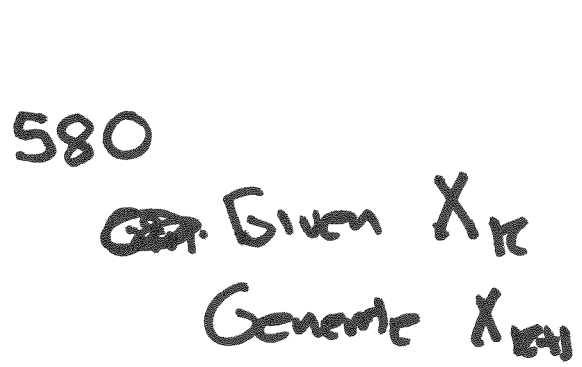
Optimization Based Design

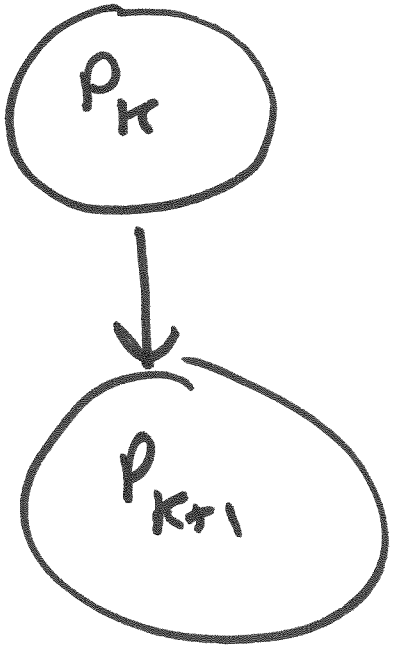
- Pose Design Problem As An Optimization Problem
  - Can be single or multi-objective
  - Systematically encode design constraints and objectives into fitness functions
- Unfortunately Problem Properties Not Always Friendly
  - Not differentiable
  - Not convex
  - Many local extrema
  - No unique global optimum



Optimization Methods

- Classic Methods
  - Newton's Method
  - Gradient Methods
  - Conjugate Direction Methods
  - Quasi-Newton Methods
  - Nedler-Mead Simplex Method
- Populations Based Methods
  - Monte-Carlo
  - Population Based Classical Methods
  - **Genetic Algorithms**
  - Swarm Algorithms





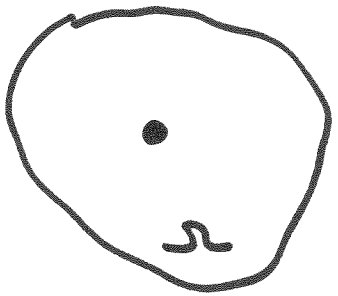
Newton's Method (1/4)

Scalar function of a vector argument $f(x)$

$$f(x) = f(x_0) + \frac{\partial f}{\partial x}(x_0)(x - x_0) + \frac{1}{2}(x - x_0)^T F(x_0)(x - x_0) + H$$

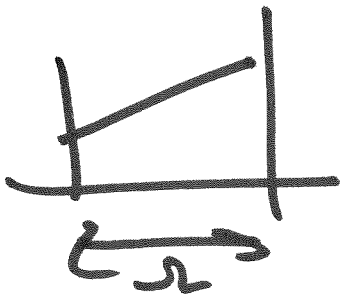
gradient $\nabla f(x_0)$

$$\nabla f(x_0) = 0$$



$$\text{Hessian} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \cdots & \cdots \\ \cdots & \cdots & \cdots \end{bmatrix}$$

$F = F^T$



Newton's Method (2/4)

$$\nabla f_q(x) = \nabla f(x_0) + \frac{1}{2} F(x_0)(x - x_0) + \left[\frac{1}{2}(x - x_0)^T F(x_0)\right]^T$$

$$= \nabla f(x_0) + F(x_0)(x - x_0) = 0$$

$$F(x_0)(x - x_0) = -\nabla f(x_0)$$

$$x = x_0 - F^{-1}(x_0)\nabla f(x_0)$$

$$\boxed{x_{k+1} = x_k - F^{-1}(x_k)\nabla f(x_k)}$$

Newton's Method (3/4)

What Are Genetic Algorithms?

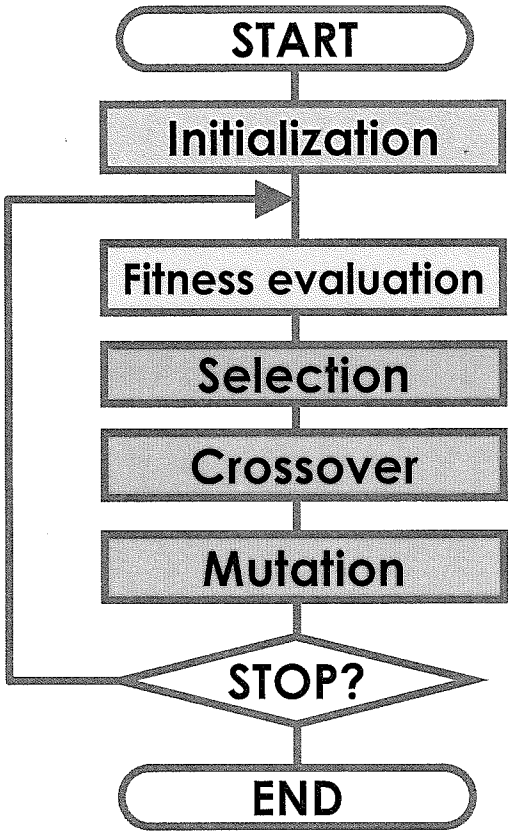
- Genetic algorithms are optimization algorithm inspired from natural selection and genetics
- A candidate solution is referred to as an ***individual***
- Process
  - Parent individuals generate offspring individuals
  - The resultant offspring are evaluated for their **fitness**
  - The fittest offspring individuals survive and become parents
  - The process is repeated

Comparing GAs and Traditional Methods

- Iterate a population not an individual
- Operations are stochastic not deterministic
- No derivatives of any kind

Canonical GA

- The **canonical genetic algorithm** refers to the GA proposed by John Holland in 1965



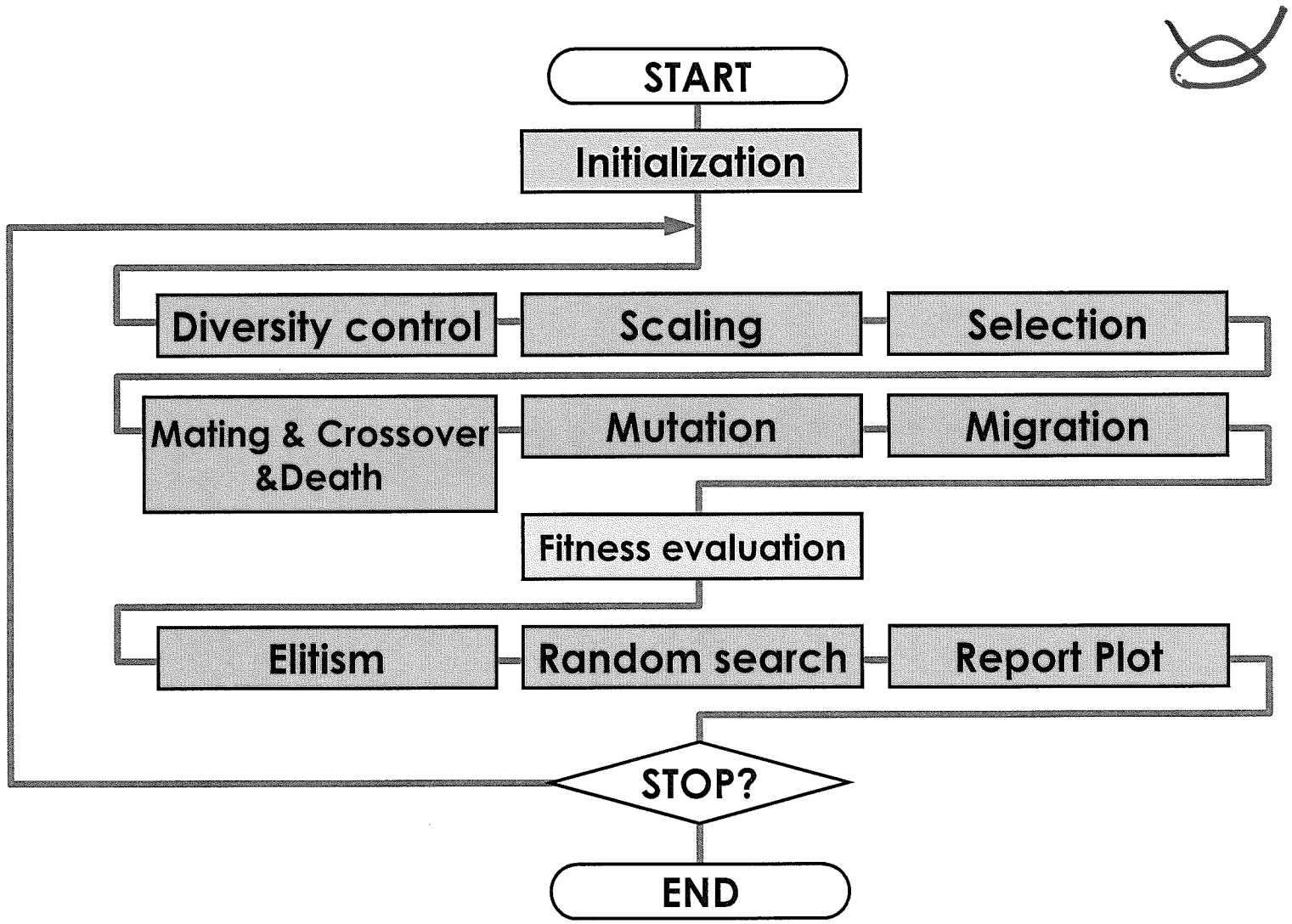
Introduction to GOSET

- **GOSET** stands for **G**enetic **O**ptimization **S**ystem **E**ngineering **T**ool

- GOSET is a MATLAB based genetic algorithm toolbox for solving optimization problems

- Available at: https://engineering.purdue.edu/ECE/Research/Areas/PEDS

Algorithm Execution



Data Structures

To conveniently process the information used in GOSET, the following data structures are employed

Data structure	Contents	No. of fields
P	Population	15
GAP	GA Parameters	86
GAS	GA Statistics	6

Population Data Structure (P)

Block Evaluation

P.blckeval	Block evaluation flag
P.fithandle	Handle to the fitness function
P.size	The number of individuals in the population
P.nobj	Number of objectives
P.mfit	Fitness function values (# obj. by pop. size)
P.fit	Aggregated fitness function values (1 by pop. size)
P.eval	Fitness evaluation flag
P.age	Age of individuals

Population Data Structure (P)

P.ngenes	Number of genes in an individual
P.min	Minimum value of genes (# genes by 1)
P.max	Maximum value of genes (# genes by 1)
P.type	Types of genes
P.chrom_id	Chromosome ID of genes (for multiple chromosome)
P.normgene	Normalized gene values (# genes by pop. size)
P.gene	Gene values (# genes by pop. size)

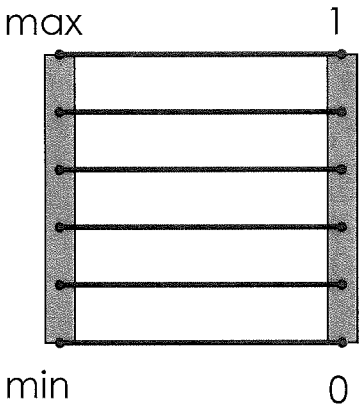
Population Data Structure (P)

P.region	Geographic region of individuals
P.pen	Fitness weight values for penalizing in the diversity control

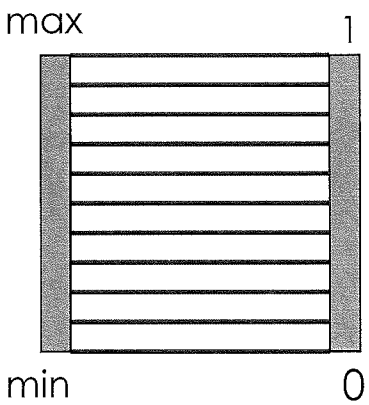
Encoding

- P.type (# genes by 1)
  - Determine the mapping method of the normalized gene value to its actual value
  - There are three different types of mapping

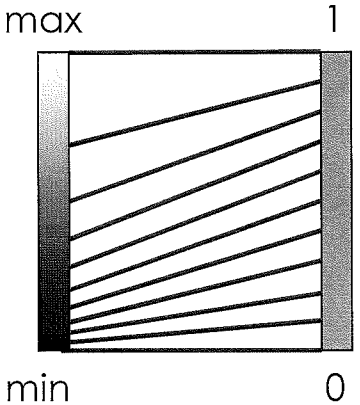
Integer



Linear

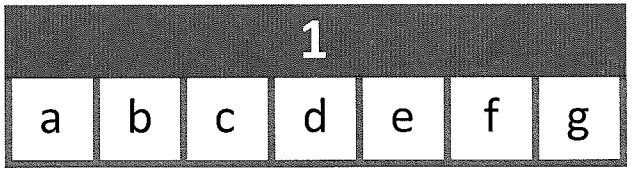
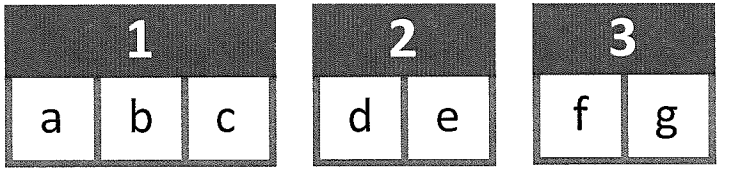
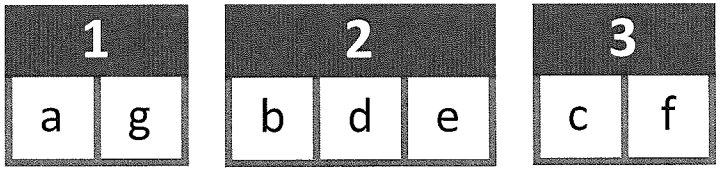


Logarithmic



Chromosome ID

Given the genes of an individual

P.chrom_id	Chromosome structure
P.chrom_id = [1 1 1 1 1 1 1]^T	
P.chrom_id = [1 1 1 2 2 3 3]^T	
P.chrom_id = [1 2 3 2 2 3 1]^T	

Genetic Algorithm Parameters (GAP)

Category	Field names
Fundamental parameters	GAP.fp_ngen GAP.fp_ipop GAP.fp_npop GAP.fp_nobj GAP.fp_obj
Diversity control parameters	GAP.dc_act GAP.dc_alg GAP.dc_spc GAP.dc_mnt GAP.dc_mxt GAP.dc_ntr GAP.dc_mnb GAP.dc_mxb GAP.dc_dc GAP.dc_nt
Selection algorithm parameters	GAP.sl_alg GAP.sl_nts GAP.sl_cah
Death algorithm parameters	GAP.dt_alg GAP.dt_nts GAP.dt_cah
Mating and crossover parameters	GAP.mc_pp GAP.mc_fc GAP.mc_alg GAP.mc_gac GAP.mc_ec
Mutation parameters	GAP.mt_ptgm0 GAP.mt_prgm0 GAP.mt_srgm0 GAP.mt_pagm0 GAP.mt_sagm0 GAP.mt_prvm0 GAP.mt_srvm0 GAP.mt_pavm0 GAP.mt_savm0 GAP.mt_pigm0 GAP.mt_ptgmf GAP.mt_prgmf GAP.mt_srgmf GAP.mt_pagmf GAP.mt_sagmf GAP.mt_prvmf GAP.mt_srvmf GAP.mt_pavmf GAP.mt_savmf GAP.mt_pigmf
Migration parameters	GAP.mg_nreg GAP.mg_tmig GAP.mg_pmig
Evaluation Parameters	GAP.ev_bev GAP.ev_are GAP.ev_ssd
Scaling parameters	GAP.sc_alg GAP.sc_kln GAP.sc_cst GAP.sc_kmxq GAP.sc_kmnq
Gene repair	GAP.gr_alg
Elitism parameters	GAP.el_act GAP.el_fgs GAP.el_fpe
Random search parameters	GAP.rs_fgs GAP.rs_fps GAP.rs_srp GAP.rs_sap GAP.rs_frp GAP.rs_fea
Reporting parameters	GAP.rp_lvl GAP.rp_gbr GAP.rp_crh
Objective plot parameters	GAP.op_list GAP.op_style GAP.op_sign
Pareto plot parameters	GAP.pp_list GAP.pp_xl GAP.pp_yl GAP.pp_title GAP.pp_style GAP.pp_sign GAP.pp_axis
Distribution plot parameters	GAP.dp_type GAP.dp_np GAP.dp_res
Trim parameters	GAP.trimga
Gene data parameters	GAP.gd_min GAP.gd_max GAP.gd_type GAP.gd_cid

Genetic Algorithm Parameters (GAP)

- GAP.fp_ngen — # generations
- GAP.fp_ipop
- GAP.fp_npop — population size
- GAP.fp_nobj — number of objectives
- GAP.fp_obj — objective to optimize

Genetic Algorithm Statistics (GAS)

GAS.[Field name]	Description
GAS.cg	Current generation number
GAS.medianfit	The median fitness values of each objective (No. of objectives × No. of generations)
GAS.meanfit	The average fitness values of each objective (No. of objectives × No. of generations)
GAS.bestfit	The best fitness values of each objective (No. of objectives × No. of generations)
GAS.bestgenes	The best gene values for each objective over the generations (No. of genes × No. of generations × No. of objectives)
GAS.ne	The number of the total objective function evaluations

GOSET Genetic Operators

- Diversity control
- Scaling
- Selection
- Death
- Mating & crossover
- Mutation
- Gene Repair 
- Migration
- Elitism
- Random search
- Deterministic Trim

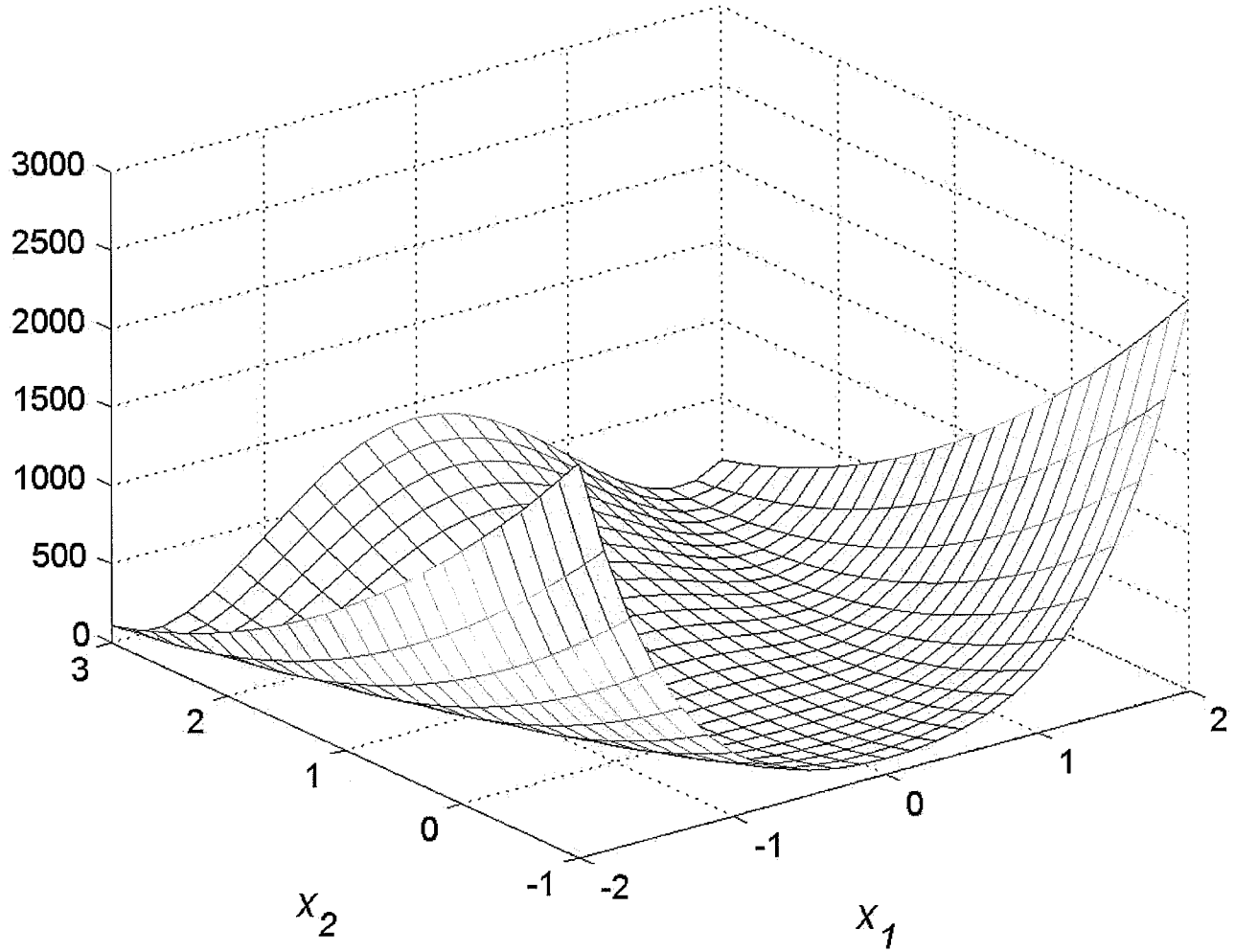
Example: Minimize the Banana

- Minimization of Rosenbrock's function

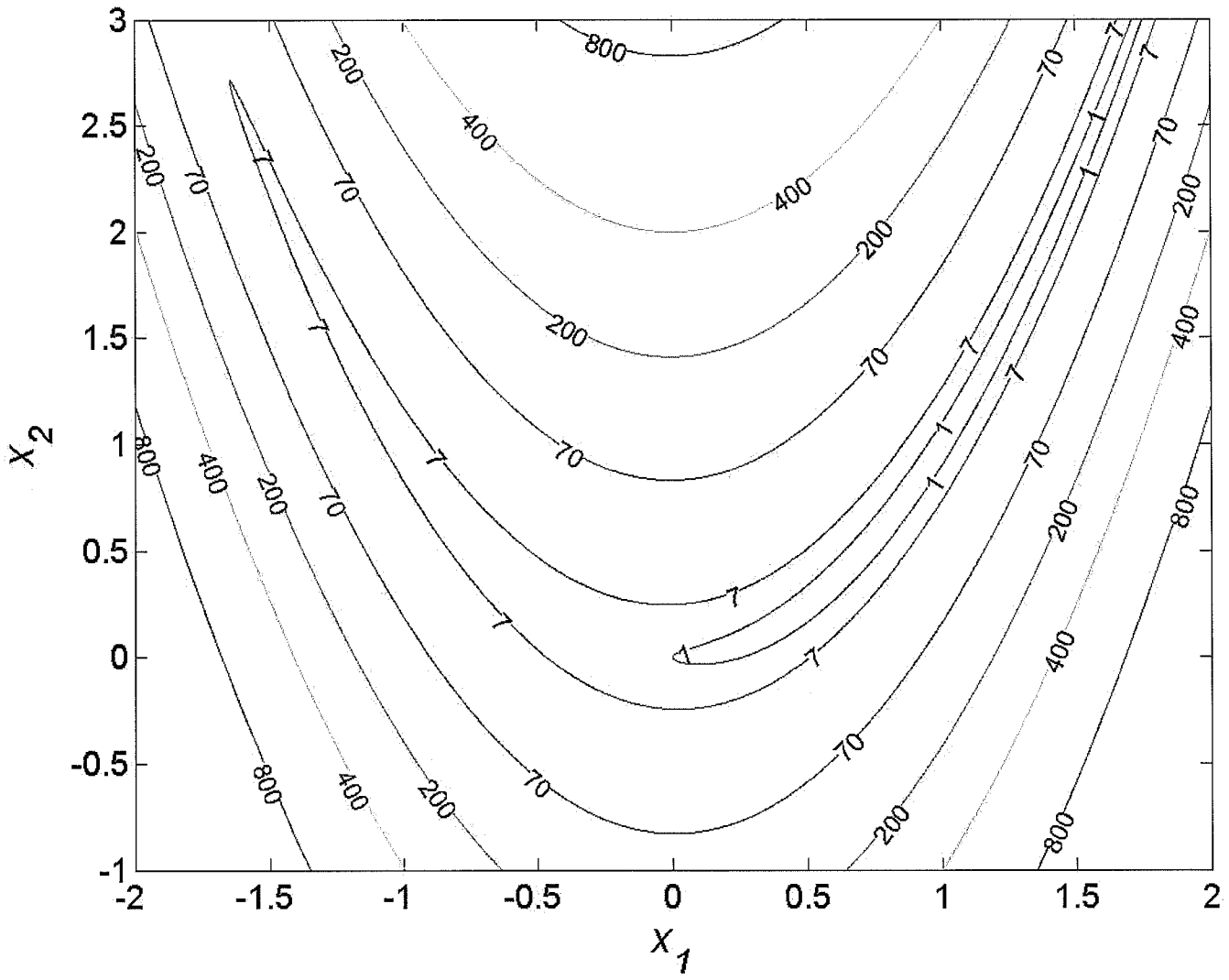
$$\min f(x_1, x_2) = 100(x_2 - x_1)^2 + (1 - x_1)^2$$

- 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

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

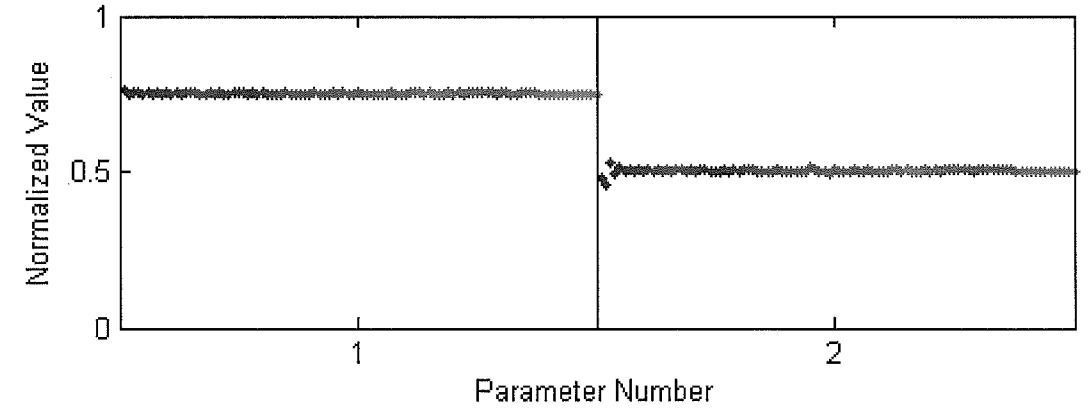
banana.m

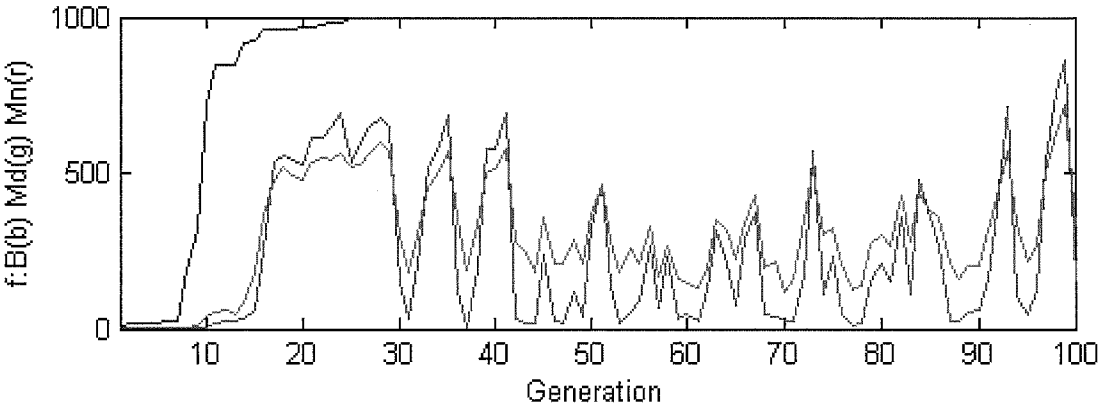
```
% Optimization of Rosenbrock's Problem

% Initialize the parameters
GAP=gapdefault;          % load the default values for GAP

% 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,best_parameters] = gaoptimize(@banana_fitness,GAP);
```

banana_fitness.m

```
% banana_fitness.m
%
% Banana Function

function f = banana_fitness(x)

f1 = 100*(x(2) - x(1)^2)^2 + (1 - x(1))^2;   % Banana function
f = 1/(0.001 + f1);                          % Fitness function
```

Results





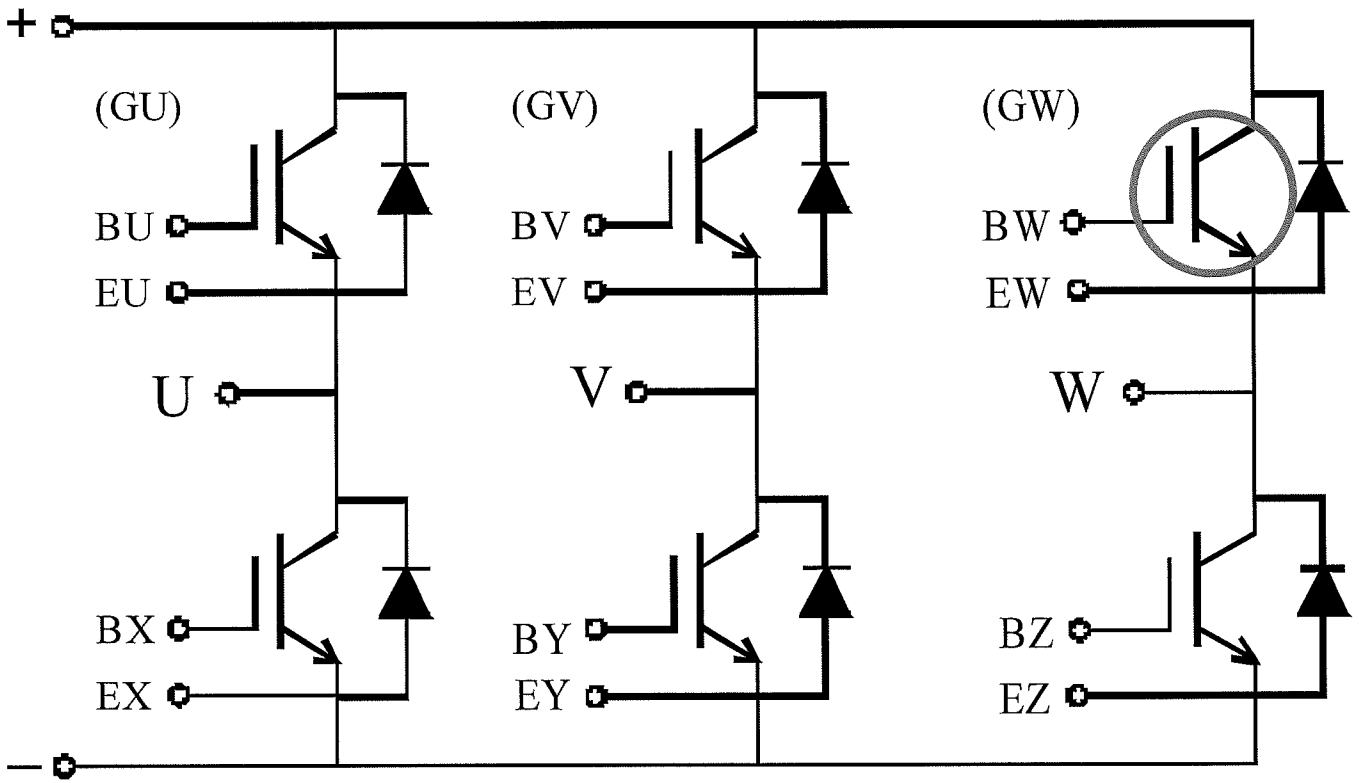
best_parameters =
1.0000
1.0000

Alternate Call (banana2.m)

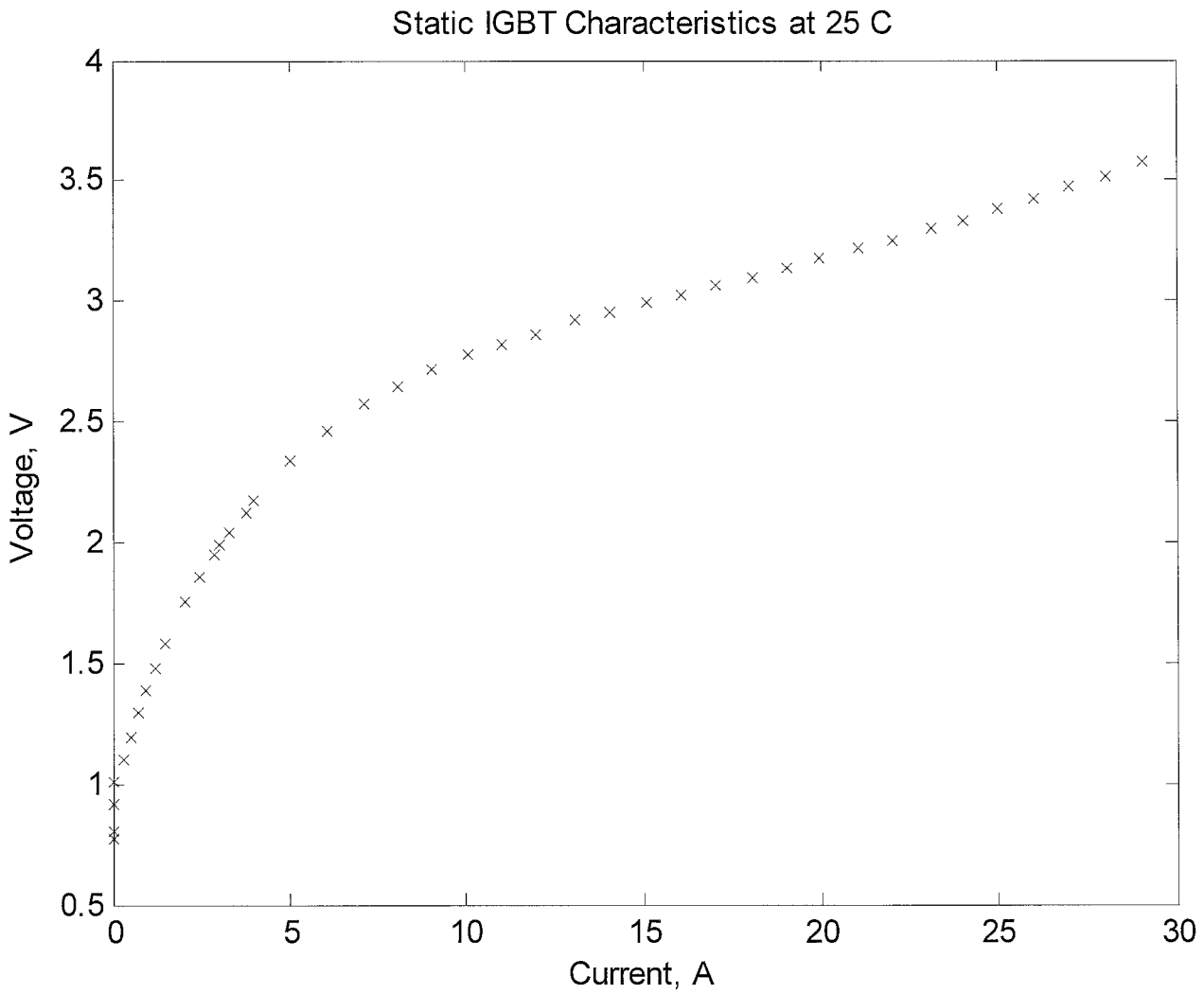
```
% Optimization of Rosenbrock's problem

% Initialize the parameters
GAP=gapdefault;

% Define gene parameters
%           min    max    type    chrom
GAP.gd = [  -2     2      2       1;       % gene 1
            -1     3      2       1];      % gene 2

% Execute GOSET
[P,GAS,best_parameters] = gaoptimize(@banana_fitness,GAP);
```

Example 2: Curve Fitting Losses

- Characterize the i-v curve of the IGBT in a FUJI 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

$$\left| \frac{V_K - V_{pred,K}}{V_K} \right|$$

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

igbt_fit.m

```
% this routine 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));
```

igbt_fit.m (continued)

```
if nargin>2

    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
```

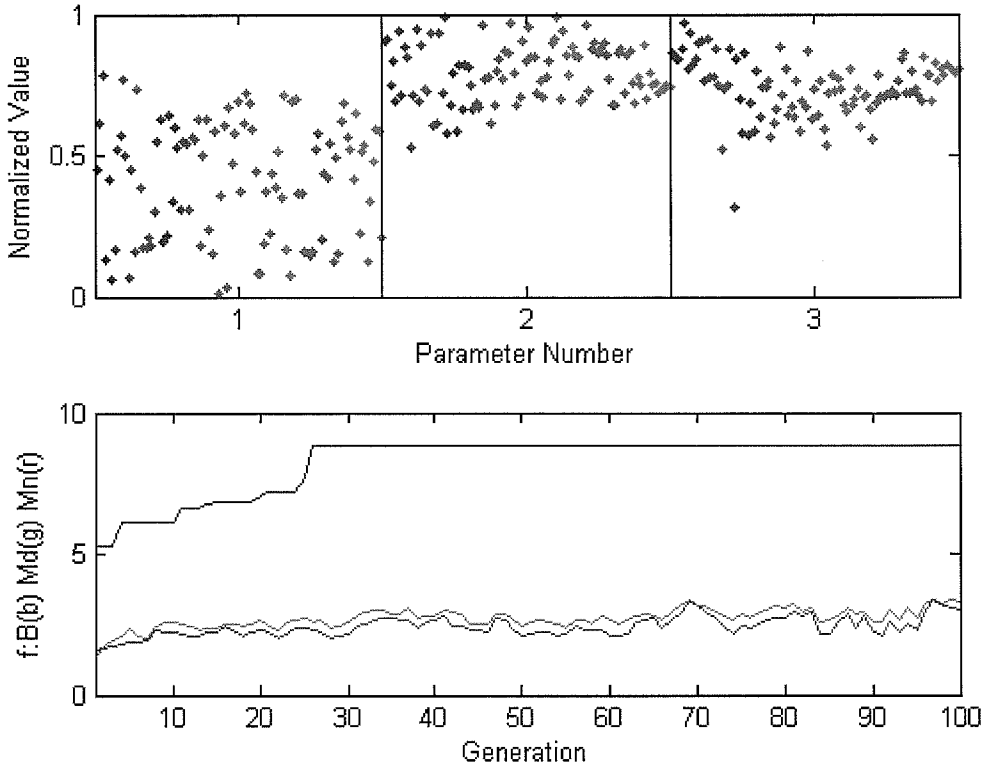
igbt.m

```
% 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;
GAP.mc_alg=10;
```

igbt.m

```
% set up genes
%           min   max type chrom
GAP.gd = [1e-8 1e+2  3    1;       % a
          1e-6 1e+3  3    1;       % b
          1e-3 1e+0  3    1];      % c

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

% plot the best fit
igbt_fit(bestparameters,igbt_data,4);
```

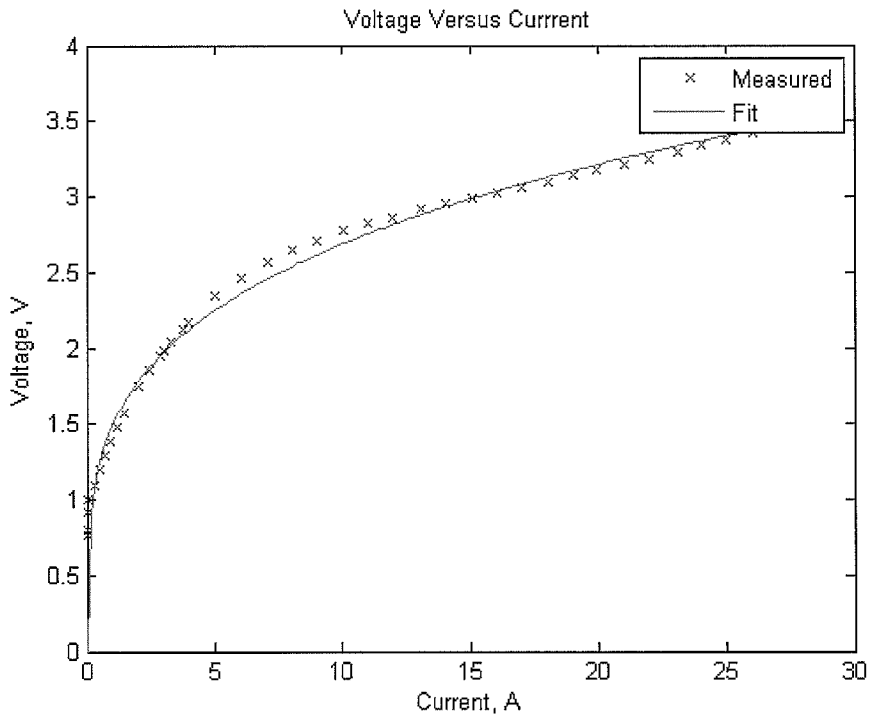
Evolution


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

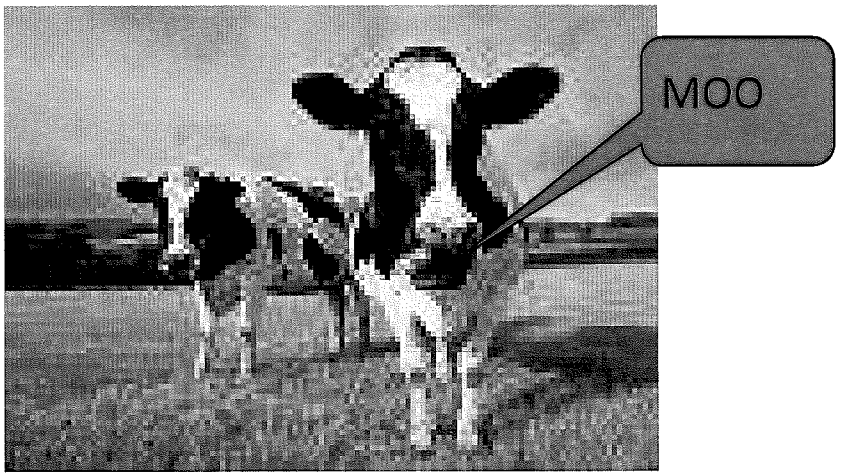
- After 100 generations, the best individual has the following parameter values.

```
a =0, b = 4.6415, c = 0.2574
```



Multi-Objective Optimization (MOO)

- Involve more than one objective function that are to be minimized or maximized
- Answer is set of solutions that define the best tradeoff between competing objectives



General Form of MOOP

- Mathematically

$$\text{min/max} \quad f_m(\boldsymbol{x}), \qquad m = 1, 2, \cdots, M$$

$$\text{subject to } g_j(\boldsymbol{x}) \geq 0, \quad j = 1, 2, \cdots, J$$

$$h_k(\boldsymbol{x}) = 0, \quad k = 1, 2, \cdots, K$$

$$\underset{\text{lower bound}}{x_i^{(L)}} \leq x_i \leq \underset{\text{upper bound}}{x_i^{(U)}}, \quad i = 1, 2, \cdots, n$$

Dominance

- In the single-objective optimization problem:

$$F(x_2) > F(x_1)$$

- In multi-objective optimization problem:

$$f_1(x_2) > f_1(x_1)$$
$$f_2(x_1) > f_2(x_2)$$

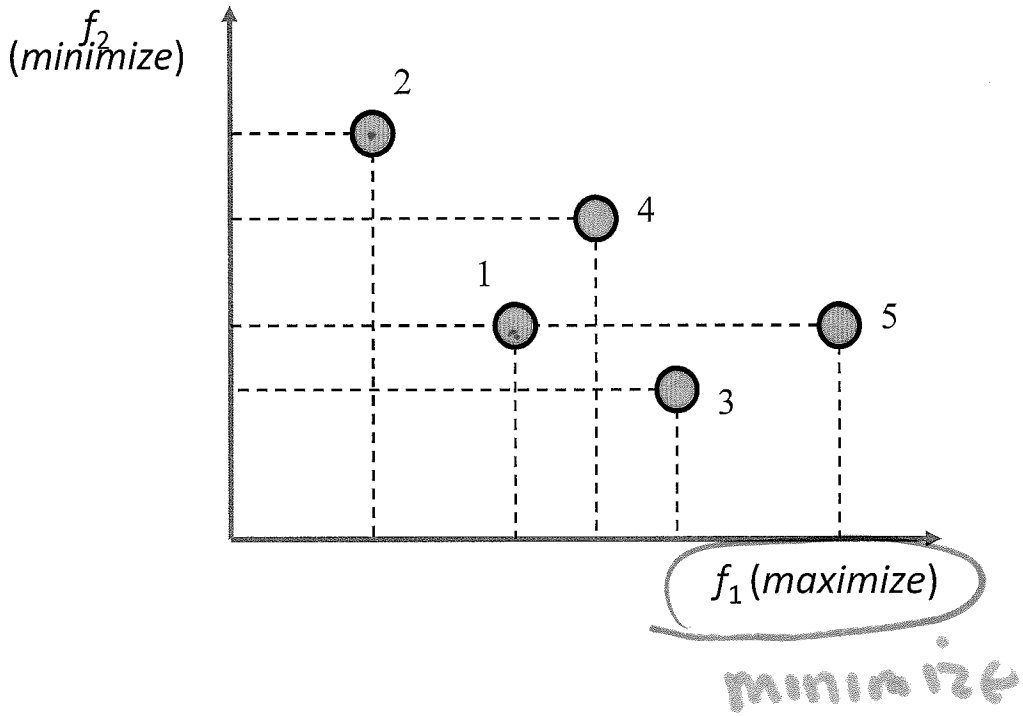
Definition of Dominance

- x_1 dominates x_2, if

  X_1 is at least as good as X_2 in all objectives

  X_1 is strictly better than X_2 in at least on objective

Example Dominance Test

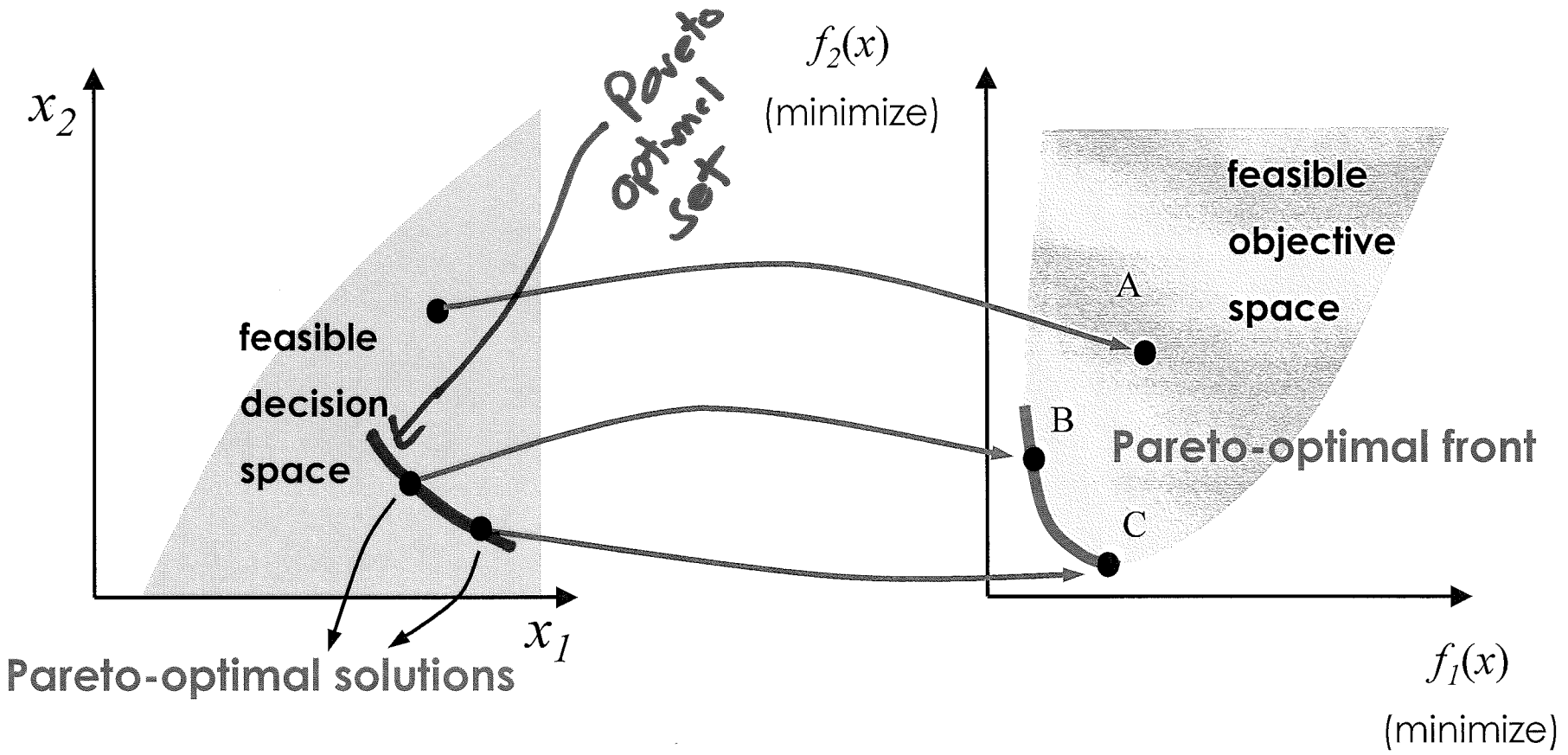


- 1 Vs 2:
- 1 Vs 5:
- 1 Vs 4:

Pareto Optimal Solution

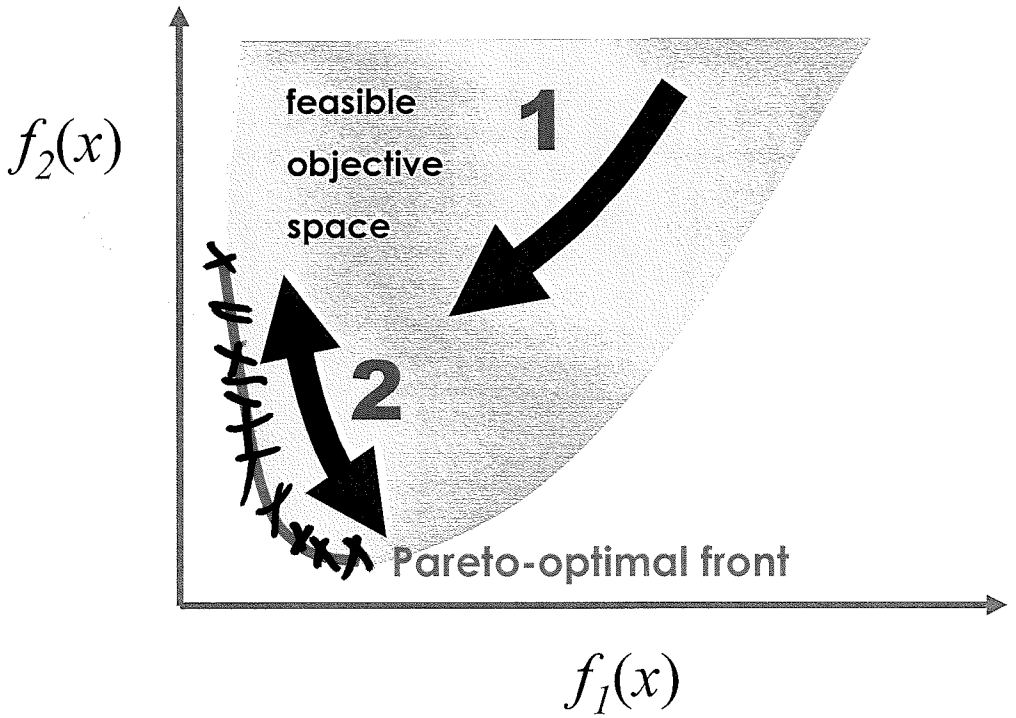
- **Non-dominated solution set**
  - Given a set of solutions, the non-dominated solution set is a set of all the solutions that are not dominated by any member of the solution set
- The non-dominated set of the entire feasible decision space is called the **Pareto-optimal set**
- The boundary defined by the set of all point mapped from the Pareto optimal set is called the **Pareto-optimal front**

Pareto Optimal Solution



Goals in MOO

- Find set of solutions as close as possible to Pareto-optimal front
- To find a set of solutions as diverse as possible



Multi-Objective GAs

- Our desired answer: a set of solutions
- Traditional optimization methods operate on **a** candidate solution
- Genetic algorithms fundamentally operate on a set of candidate solutions

Multi-Objective MOEAs

Non-Elitist MOEAs

- Vector evaluated GA
- Vector optimized ES
- Weight based GA
- Random weighted GA
- Multiple-objective GA
- Non-dominated Sorting GA
- Niched Pareto GA

Elitist MOEAs

- Elitist Non-dominated Sorting GA
- Distance-based Pareto GA
- Strength Pareto GA
- Pareto-archived ES

Multi-Objective Optimization in GOSET

- GOSET employ an **elitist GA** for the multi-objective optimization problem
- **Diversity control algorithms** are also employed to prevent over-crowding of the individuals in a specific region of the solution space
- The non-dominated solutions are identified using the recursive algorithm proposed by **Kung** et al.

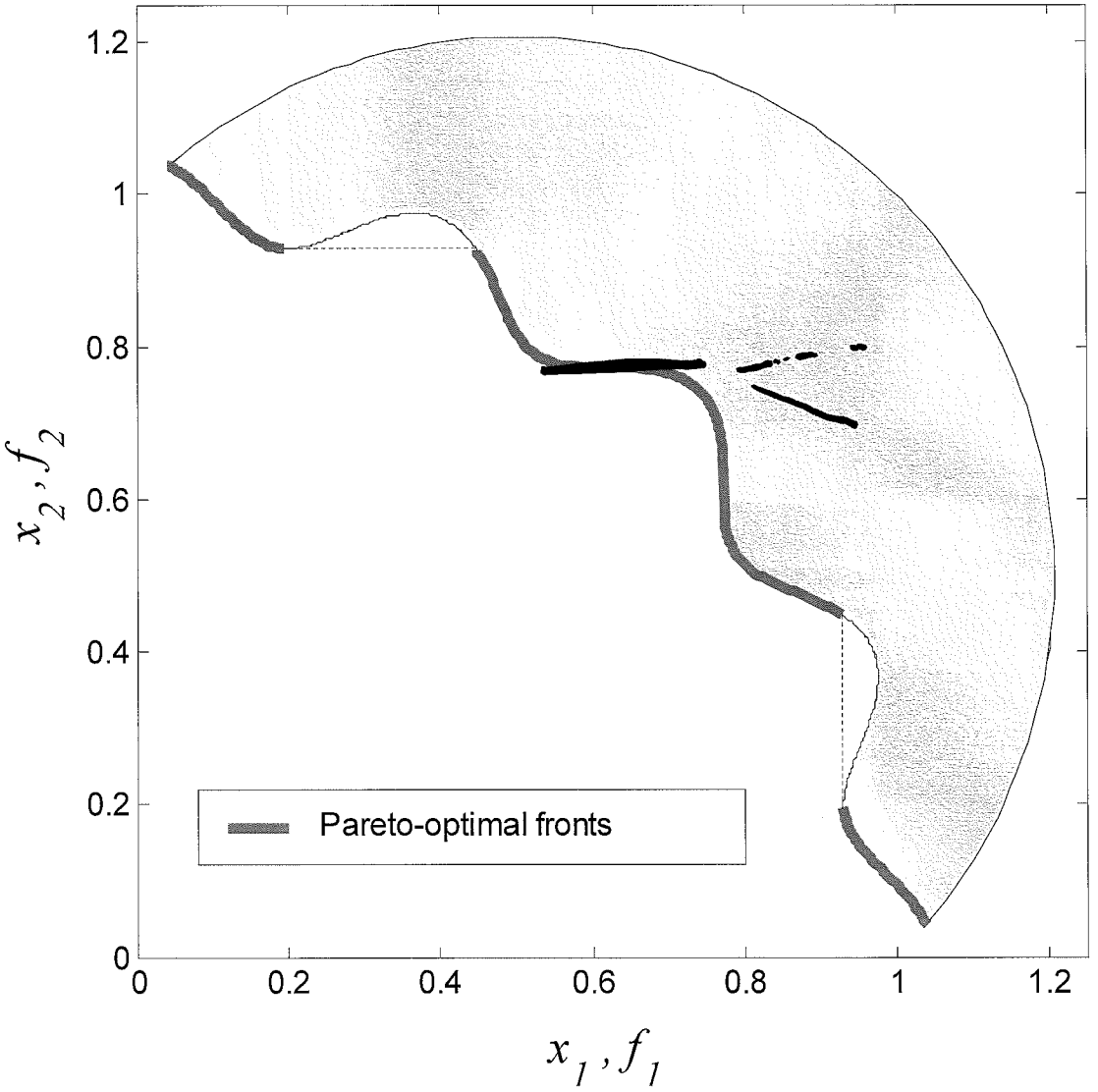
Example: Tanaka Problem

- Tanaka problem is a constrained optimization problem with two objectives to be minimized

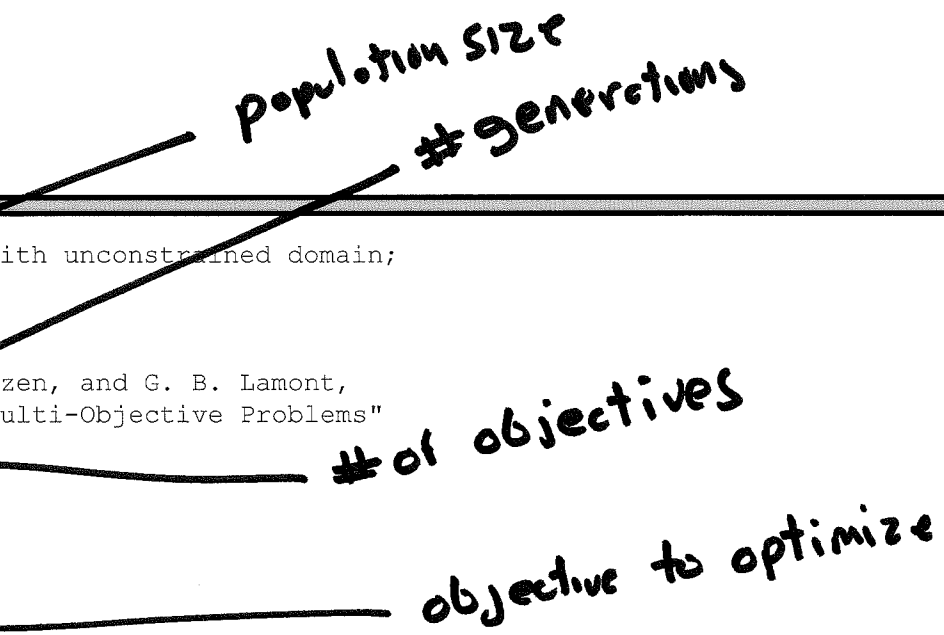
$$\min f_1(x_1, x_2) = x_1$$
$$\min f_2(x_1, x_2) = x_2$$
$$\text{subject to } C_1(x_1, x_2) = x_1^2 + x_2^2 - 1 - 0.1\cos\left(16\arctan\frac{x_1}{x_2}\right) \geq 0, \quad 0 \leq x_1 \leq \pi,$$
$$C_2(x_1, x_2) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5, \quad 0 \leq x_2 \leq \pi.$$

- Note the variable space is also the objective space

Tanaka Problem



tanaka.m

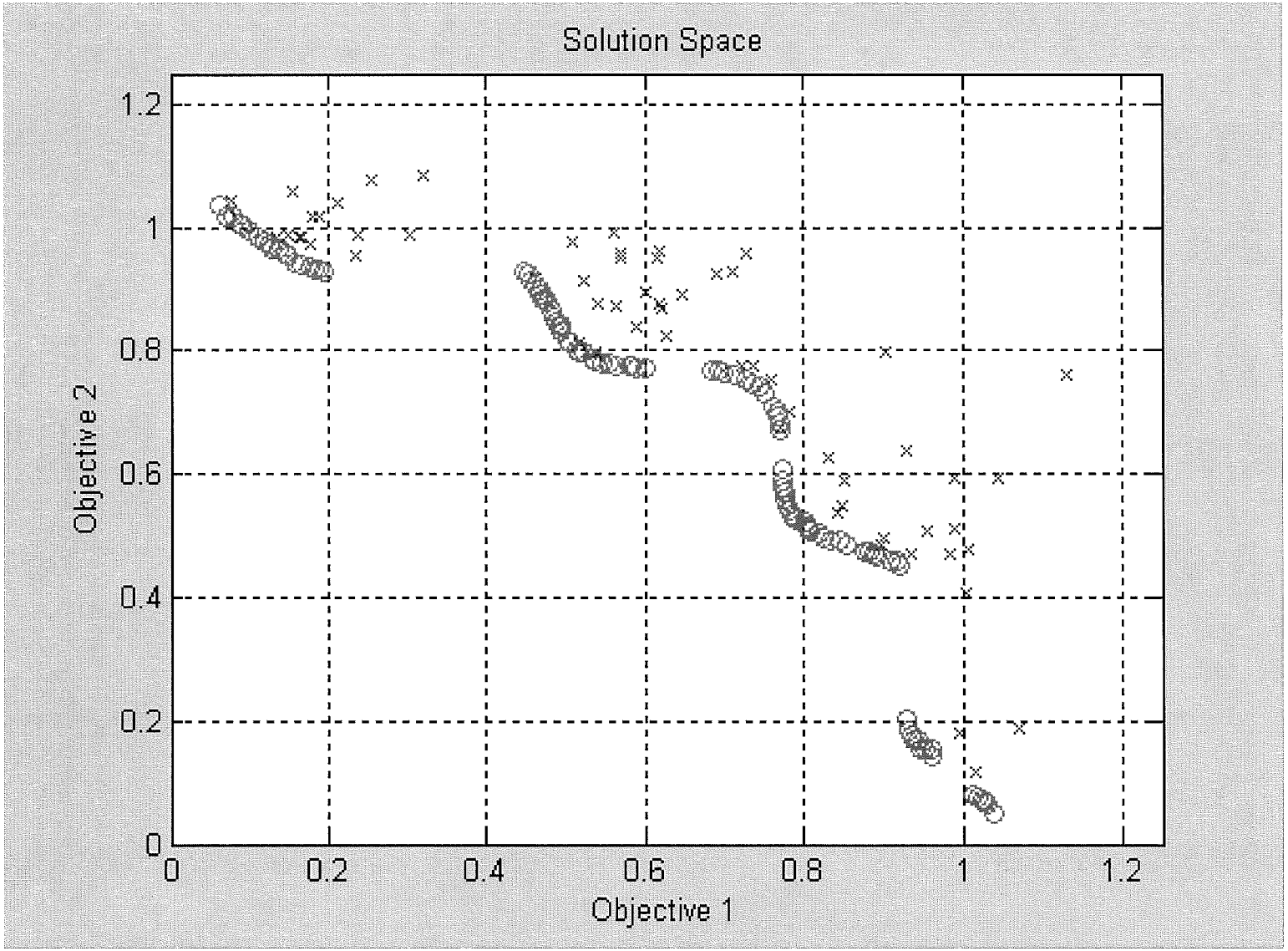



```
% This is a multi-objective test function, with unconstrained domain;
% proposed by Poloni (MOP3, p.110 [1])
%
% Reference:
% [1] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont,
%     "Evolutionary Algorithms for Solving Multi-Objective Problems"
%     Kluwer Academic Publishers, 2002
%
% date: 12.5.2006

% Initialize the parameters
GAP=gapdefault(2,0,200,200);

% Use a pareto plot
GAP.op_list = [];               % objectives list for objective plots
GAP.pp_list = [ 1, 2];          % gene list for Pareto plot
GAP.pp_sign = [-1,-1];          % sign of fitness for each objective
                                % (1=positive,-1=negative)
GAP.pp_axis = [0 1.25 0 1.25];  % axis limits for Pareto plot

%                 x1     x2
% gene             1      2
GAP.gd_min  = [    0      0     ];
GAP.gd_max  = [   pi     pi     ];
GAP.gd_type = [    2      2     ];
GAP.gd_cid  = [    1      1     ];

[P,GAS,best]= gaoptimize(@tanaka_fit,GAP);

% Plot final solutions
figure(2)
plot(best(1,:),best(2,:),'x');
axis([0 1.25 0 1.25]);
xlabel('f_1');
ylabel('f_2');
axis square;
title('Pareto Front')
```

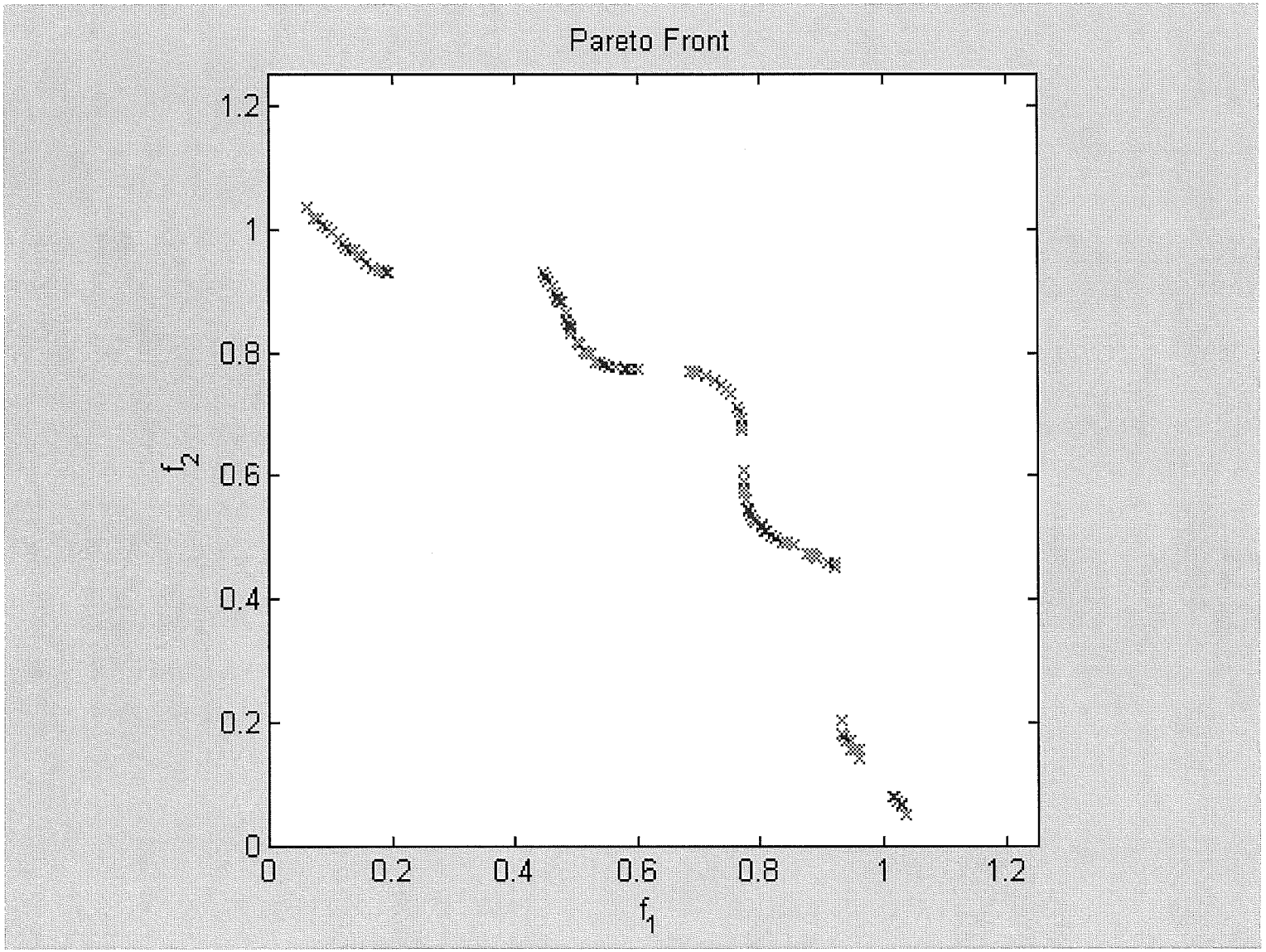
tanaka_fitness.m

```
% This is a multi-objective test function, with unconstrained domain;
% proposed by Poloni (MOP3, p.110 [1])
%
% Reference:
% [1] C. A. Coello Coello, D. A. Van Veldhuizen, and G. B. Lamont,
%     "Evolutionary Algorithms for Solving Multi-Objective Problems"
%     Kluwer Academic Publishers, 2002
%
% date: 12.5.2003

function [f] = tanaka(x)

C1 = x(1)^2+x(2)^2-1-0.1*cos(16*atan(x(1)/x(2))) >= 0;
C2 = (x(1)-0.5)^2+(x(2)-0.5)^2 <= 0.5;

if C1 & C2
    f(1) = -x(1);
    f(2) = -x(2);
else
    f(1) = -10;
    f(2) = -10;
end
```

Pareto Plot



Non Dominated Solutions

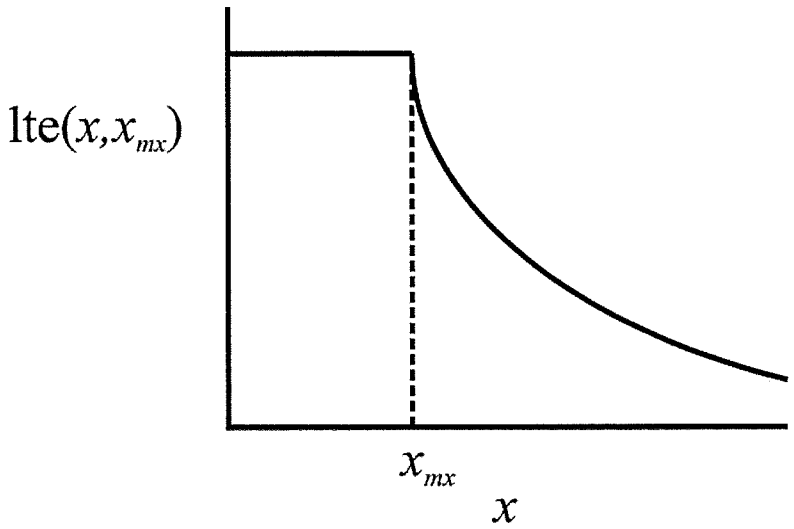


Formulation of Fitness Functions

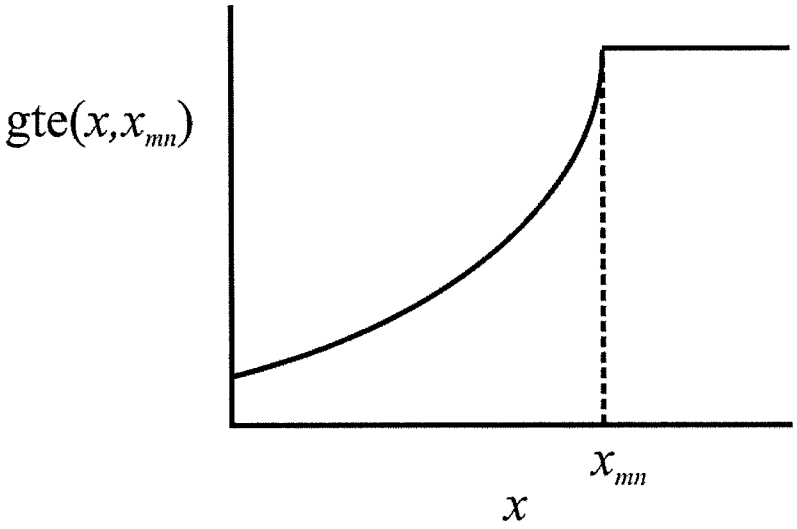
- Setting up constraints

$$\text{lte}(x, x_{mx}) = \begin{cases} 1 & x \le x_{mx} \\ \dfrac{1}{1 + x - x_{mx}} & x > x_{mx} \end{cases}$$

$$\text{gte}(x, x_{mn}) = \begin{cases} 1 & x \ge x_{mn} \\ \dfrac{1}{1 + x_{mn} - x} & x < x_{mn} \end{cases}$$



(a) less-than-or-equal-to funtion



(b) greater-than-or-equal-to function