

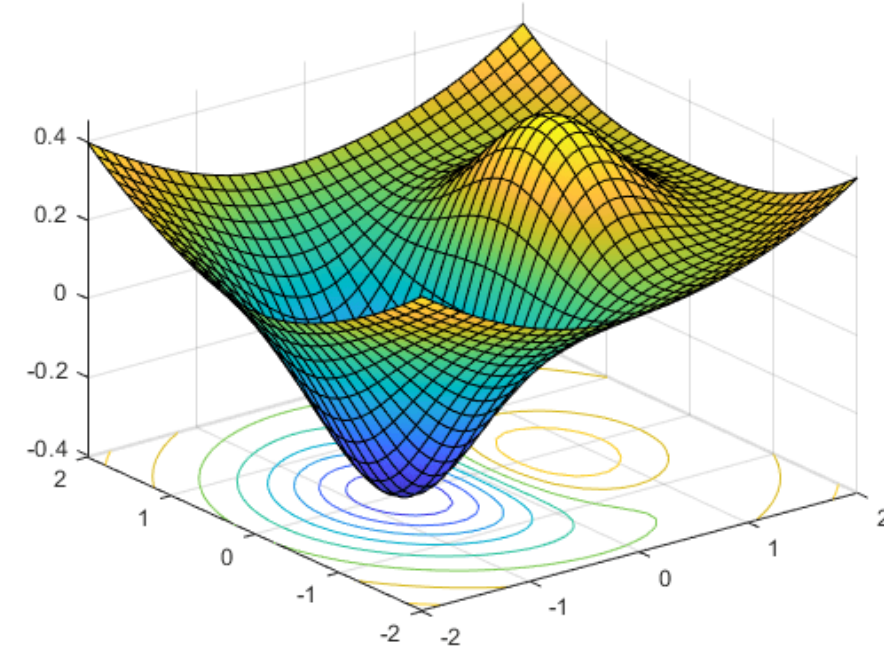
Computational Intelligence Assisted Engineering Design Optimization (using MATLAB®)

Amir Parnianifard

PhD, PMP®, IEEE Senior Member

Glasgow College, University of Electronic Science and
Technology of China, Chengdu, Sichuan, P.R.China
611731.

E-Mail: amir.p@uestc.edu.cn ; amir.parnianifard@glasgow.ac.uk

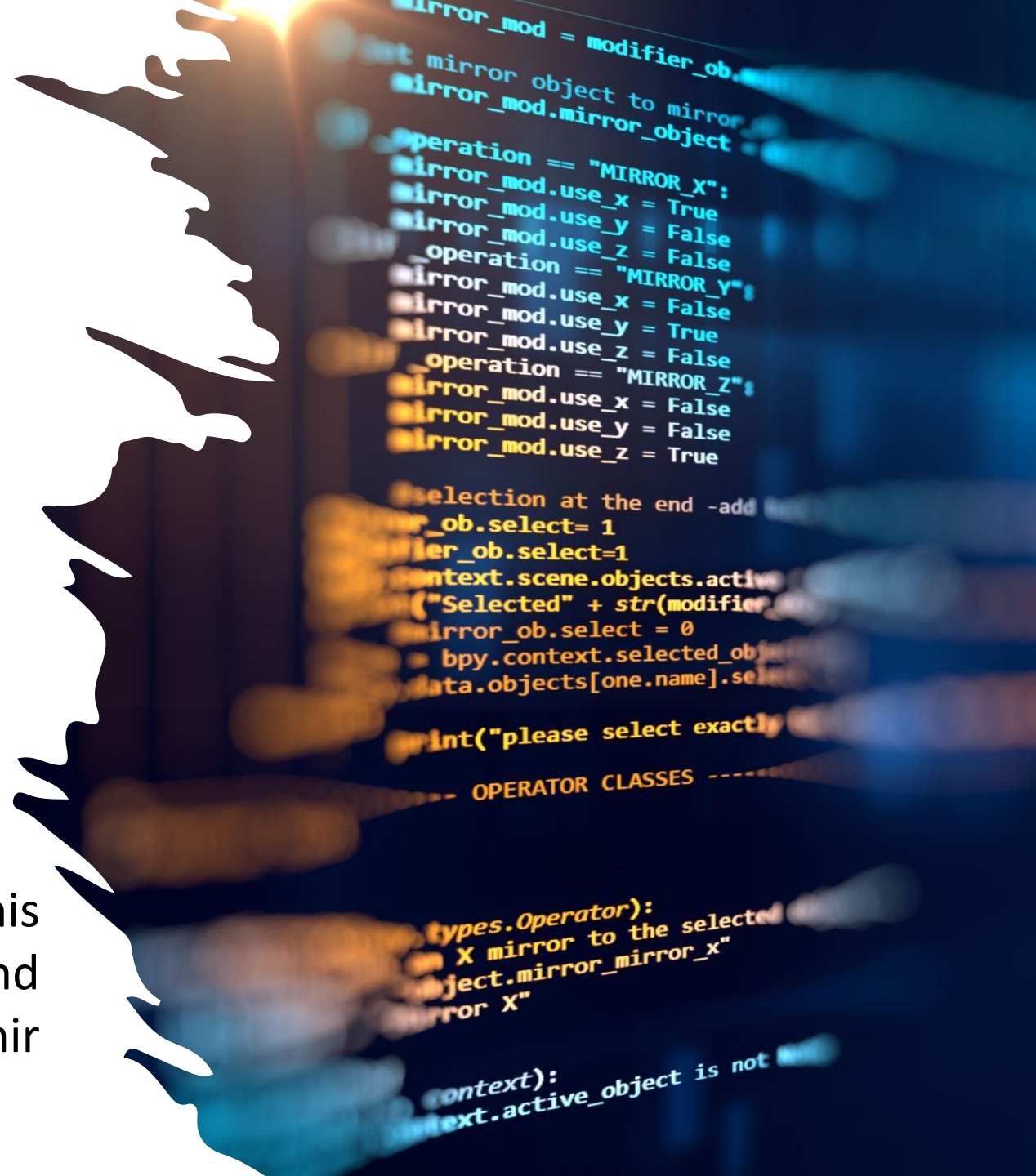


电子科技大学
格拉斯哥学院
Glasgow College, UESTC

Research Interest

- ✓ Engineering Design Optimization
- ✓ Surrogate Modelling
- ✓ Uncertainty Quantification
- ✓ Robust Design
- ✓ Digital-Twins
- ✓ Computational Intelligence
- ✓ Optimal Control
- ✓ Robot Trajectory Planning
- ✓ Wireless Sensor Networks.

Note: All MATLAB® codes provided in this presentation, have been written and augmented by the presenter (Amir Parnianifard).



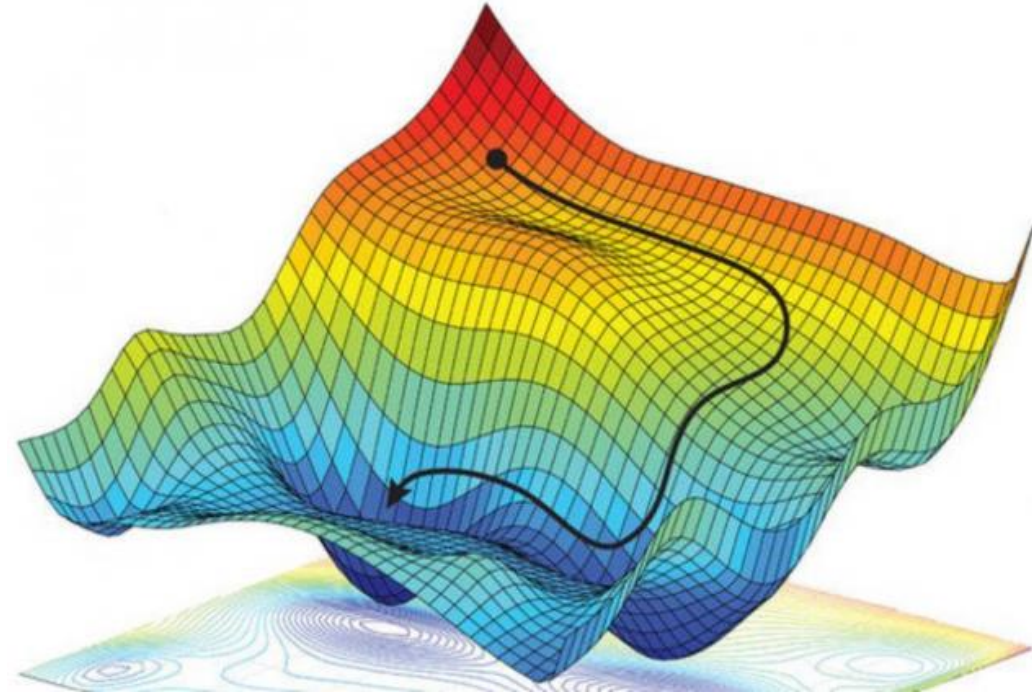
Engineering Optimization

Optimization is the process of finding values of the variables that minimize or maximize the objective function while satisfying the constraints.

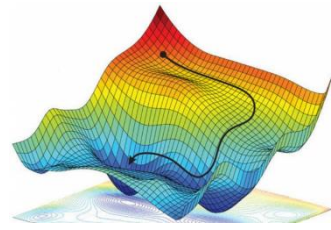
$$\text{Min or Max: } f(X)$$

$$\text{Subject to: } g_j(X) \leq 0, \\ j = 1, 2, \dots, J$$

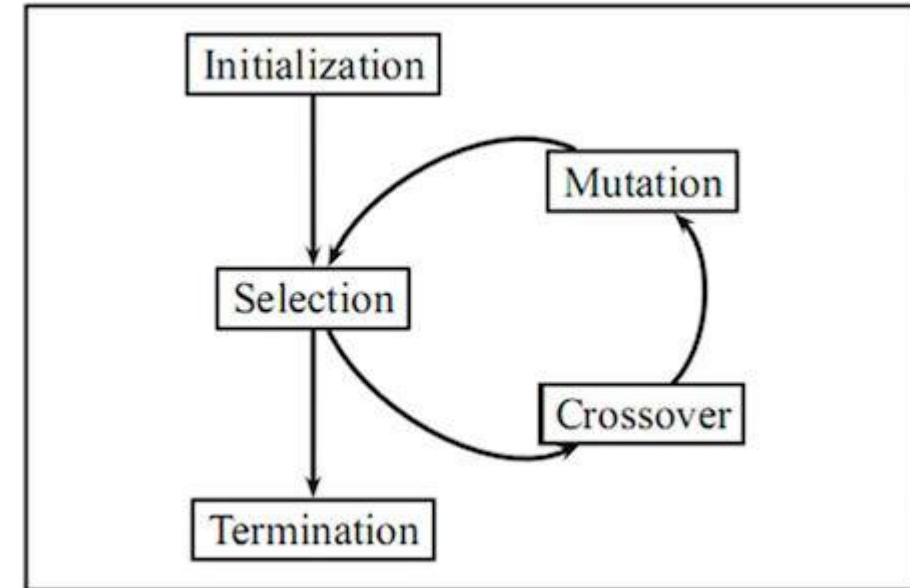
- ✓ Linear versus Nonlinear Programming
- ✓ Continuous Optimization versus Discrete Optimization.
- ✓ Unconstrained Optimization versus Constrained Optimization.
- ✓ Single or Many Objectives.
- ✓ Deterministic Optimization versus Stochastic Optimization.



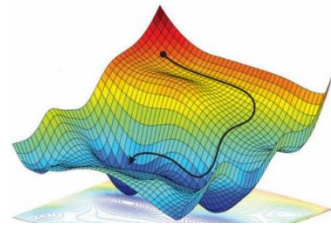
Eng. Optimization using Evolutionary Algorithms



- In computational intelligence, an Evolutionary Algorithm (EA) is a subset of evolutionary computation, a generic population-based metaheuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection.
- **Genetic Algorithm (GA)** is the most popular type of EA. A search heuristic known as a genetic algorithm was motivated by Charles Darwin's theory of natural selection[57], [58].
- **Differential Evolution (DE)** is based on vector differences and is therefore primarily suited for numerical optimization problems.



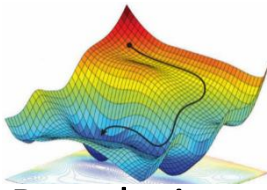
Differential Evolution Algorithm



The algorithmic procedure of DE can be summarized as follows:

1. Randomly select the N individuals uniformly on the intervals $[x_j^L, x_j^U]$, where x_j^L and x_j^U are the upper and lower bounds of decision parameters.
2. Each of the N individuals undergoes mutation, recombination, and selection. Mutation can expand the search space.
3. For a given exact individual u , randomly select three parameters such that the indices i , r_1 , r_2 and r_3 are distinct.
4. Add the weighted difference of two of the vectors to the third $r_{i+1} = r_1 + \alpha(r_2 - r_3)$, which α called a mutation factor and can be a constant from $[0,2]$, and r_{i+1} is called the donor vector.
5. These three parameters are replaced to selected individual u if the $rand() > CR$, where CR is a constant probability in $U[0,1]$.
6. If the fitness function of the updated individual is less than the original individual u , then u is removed from the set of individuals, and an updated one is replaced.
7. Until a stopping condition is met, mutation, recombination, and selection proceed.

DE Optimizer (MATLAB Code)



Input: Fitness function of problem, Lower and Upper limits of design variables, Number of Initial Population, and Maximum Number of Iteration by Algorithm

Output: The Optimum results for design variables and the relevant fitness function (in the optimum point).

```
function [Opt_var,Opt_cost] = DEO(objFcn,LoB,UpB,NoPapulation,MaxIteration)
```

```
% This is the code for Differential evolution (DE)
```

```
%% Parameter Adjustment
```

```
nVar = length(LoB);
```

```
CR = 0.5; % Crossover Rate
```

```
FF = 1-exp(-norm(0.05*(UpB-LoB)))* rand();
```

```
%Initial Population
```

```
individuals = (UpB-LoB).*rand(NoPapulation,nVar) + LoB;
```

```
% figure('Position', [400, 400, 800, 300]); hold on
```

```
%% Optimization
```

```
for iter = 1 : MaxIteration
```

```
for i = 1:NoPapulation
```

```
ind0 = individuals;
```

```
ind0(i,:) = [];
```

```
X = individuals(i,:);
```

```
Cost_X = objFcn(X);
```

```
cost(i,1) = Cost_X;
```

```
var_X(i,:) = X;
```

```
idx = randperm(NoPapulation-1,3);
```

```
X1 = ind0(idx(1),:);
```

```
X2 = ind0(idx(2),:);
```

```
X3 = ind0(idx(3),:);
```

```
d_rand = randperm(nVar,1);
```

```
for d = 1:nVar
```

```
if d == d_rand || rand <= CR
```

```
u(d) = X1(d) + FF *rand()* (X2(d) - X3(d));
```

```
if u(d) < LoB(d) || u(d) > UpB(d)
```

```
u(d) = (UpB(d) - LoB(d))/2 + LoB(d);
```

```
end
```

```
else
```

```
u(d) = X(d);
```

```
end
```

```
end
```

```
if objFcn(u) < Cost_X
```

```
individuals(i,:) = u;
```

```
end
```

```
end
```

```
[best_cost(iter),nr] = min(cost);
```

```
best_var(iter,:) = var_X(nr,:);
```

```
end
```

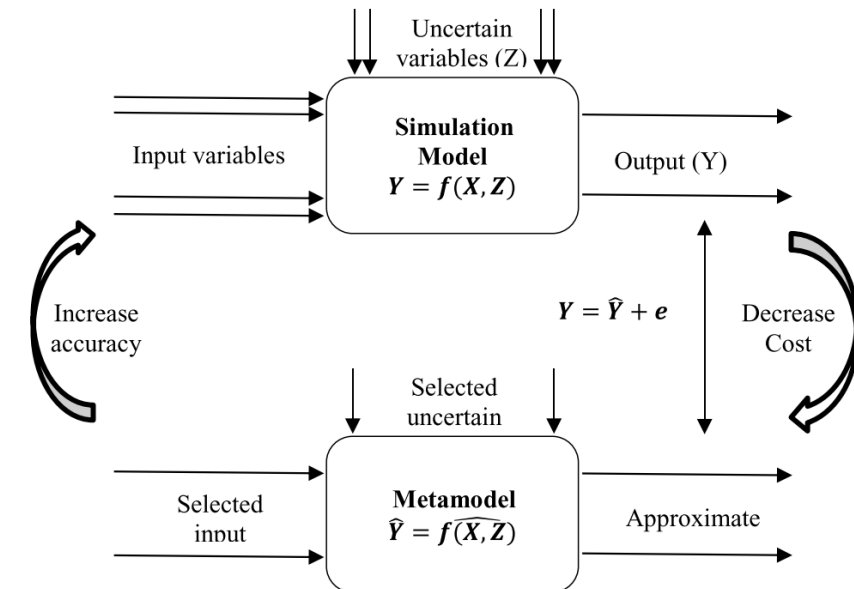
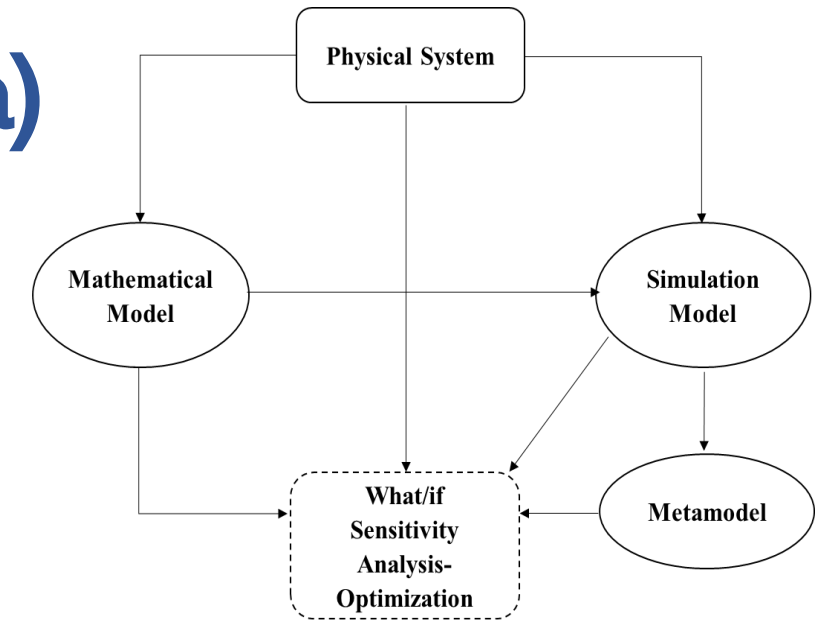
```
[Opt_cost,nr] = min(best_cost);
```

```
Opt_var = best_var(nr,:);
```

```
end
```

Metamodeling (Learning From Data)

- ❑ **Shortcoming:** High Computation Cost in case of using evolutionary algorithm for simulation-optimization
- In solving complex real- world engineering optimization problems, an evolutionary algorithm may require thousands of function evaluations in order to provide a satisfactory solution, whereas each evaluation requires hours of computer run-time.
- ❑ **Less-Expensive methods to handle complex simulation modeling and complex problems**
- To overcome computation cost, researchers have applied sampling-based learning methods such as **Artificial Neural Networks**, **Radial Basis Functions (RBF)**, **Kriging (Gaussian Process)** and **polynomial regression model**. These methods can 'learn' the problem behaviors and approximate the function value. These **approximation models** can **speed up the function evaluation** as well as estimation the function value with an acceptable accuracy.



Metamodel Based Simulation-Optimization

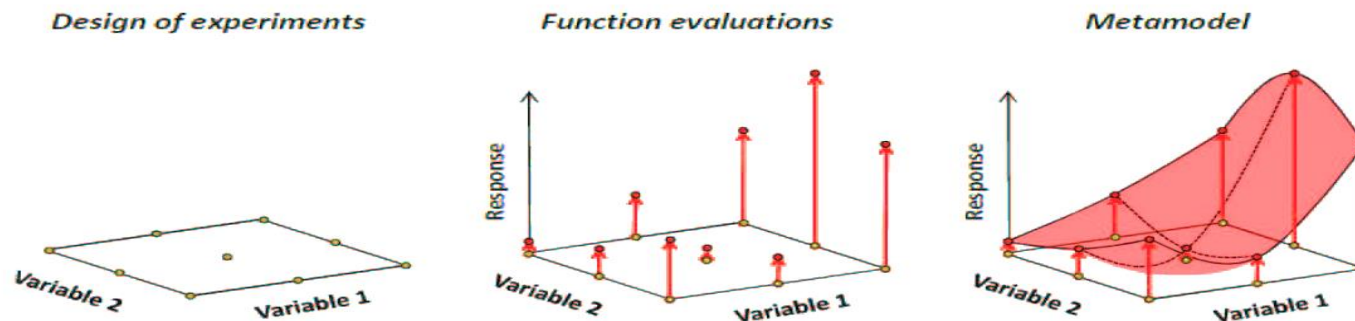
Step 1: Choosing an experimental design for generating design points.

Step 2: Run original simulation model (function evaluation) according to each generated design points and collect the relevant responses.

Step 3: Fitting the metamodel over the observed input-output dataset.

Step 4: Validating the metamodel.

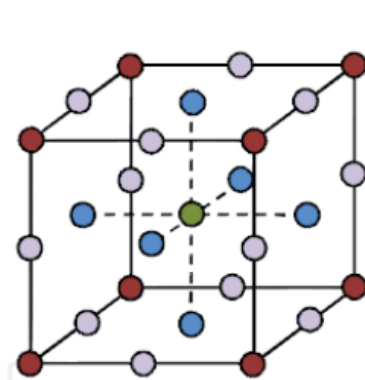
Step 5: Using fitted metamodel (**cheap**) instead of original simulation model (**expensive**) for optimization and sensitivity analysis.



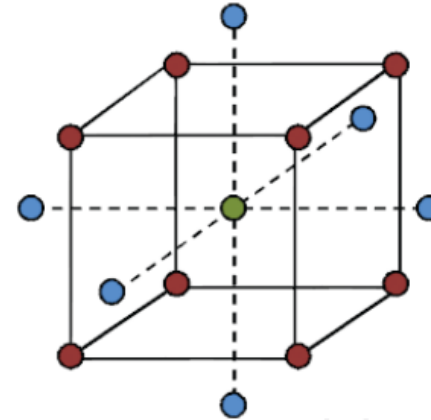
Experimental Designs

The idea of the classical DOE is to reach as much information as possible from a limited number of experiments.

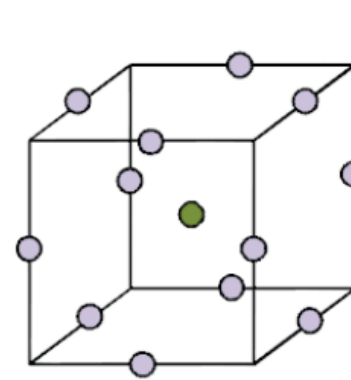
a) Full factorial,
 $n = 27$



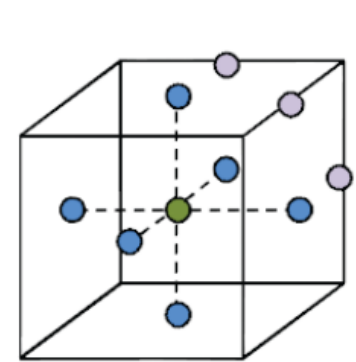
b) CCD,
 $n = 15$



c) BBD,
 $n = 13$

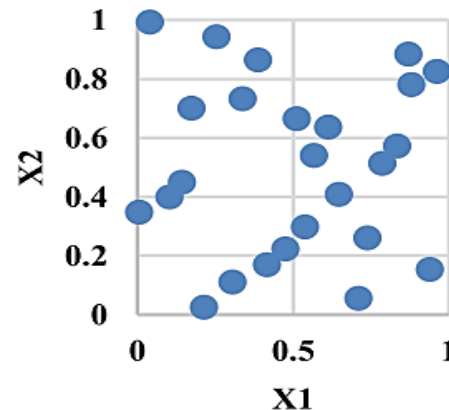


d) Koshal,
 $n = 10$

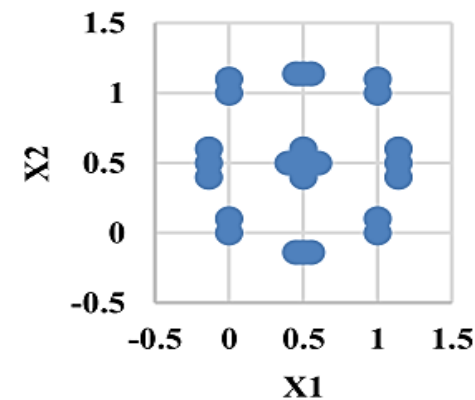


A good experimental design has to be space filling and non-collapsing rather than to concentrate on the boundary.

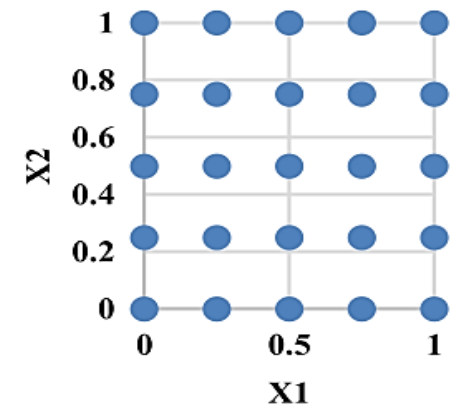
LHS (Randomized)



CCD

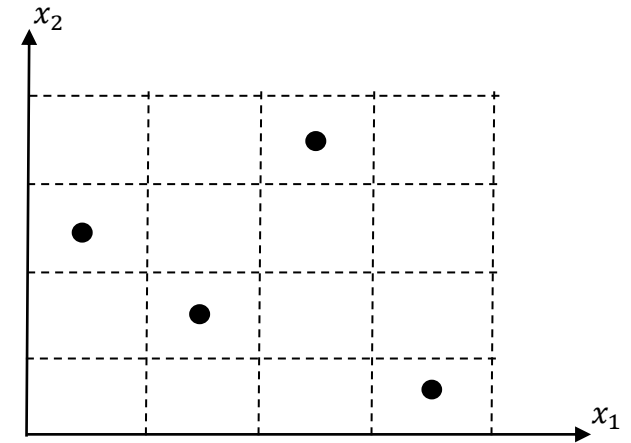


OA



Experimental Designs Latin Hypercube Sampling (LHS)

- One of the common method for designing simulation experiments is LHS.
- LHS was first introduced by McKay et al. (1979).
- In general, for n input variables, m sample points are produced randomly into m intervals or scenarios (with equal probability). For the particular instance the LHS design for $m = 4, n = 2$ is shown in here



1. In LHS, each input range divided into m subranges (integer) with equal probability magnitudes and numbered from 1 to m .
2. LHS place all m intervals by random value between lower and upper bounds relevant to each interval, since each integers $1, 2, \dots, m$ appears exactly once in each row and each column of the design space. Note that, within each cell of design, the exact input value can be sampled by any distribution, e.g., uniform, Gaussian or etc.

Three common choices are available to ensure appropriate space filling of sample points in LHS design:

1. Minimax: This design tries to minimizing the maximum distances in design space between any location for each design point and its nearest design points.
2. Maximin: In this design attempt to be maximized the minimum distance between any two design points.
3. Desired Correlational function: Inspired by (Iman & Conover, 1982) in the case of non-independent multivariate input variables the desired correlation matrix can be used to produce distribution free sample points in LHS.

Latin Hypercube Sampling (LHS)

MATLAB® code

Note: the MATLAB® function “lhsdesign” is located in Deep Learning Toolbox. Make sure install Deep Learning Toolbox.

Input: Number of sample points, Lower and Upper bound for design variables, and type of LHS.

Output: Designed sample points in upper and lower range.

```
function [LHS_Samp] = LHS_Design(NSamp,MinRangeSamp,MaxRangeSamp,Type)
[~,NVar]=size(MinRangeSamp);
switch Type
case 'maximin'
Train_lhs = lhsdesign(NSamp,NVar,'criterion','maximin');
case 'none'
Train_lhs = lhsdesign(NSamp,NVar,'criterion','none');
case 'correlation'
Train_lhs = lhsdesign(NSamp,NVar,'criterion','correlation');
end
LHS_Samp=[];
for sa = 1:NSamp
for va = 1:NVar
L = MinRangeSamp(1,va);
U = MaxRangeSamp(1,va);
X = Train_lhs(sa,va)*(U-L)+L;
XR(1,va) = X;
end
LHS_Samp = [LHS_Samp;XR];
end
end
```

Metamodel 1: Polynomial Regression (PR)

Commonly, the main purpose of PR is the approximation of true response function based on Taylor series expansion around a set of design points.

First Order

$$y = f(X) = \hat{\beta}_0 + \sum_{i=1}^m \hat{\beta}_i x_i + \varepsilon$$

Second Order

$$y = f(X) = \hat{\beta}_0 + \sum_{i=1}^k \hat{\beta}_i x_i + \sum_{i=1}^k \hat{\beta}_{ii} x_i^2 + \sum_{i=1}^k \sum_{i < j=2}^k \hat{\beta}_{ij} x_i x_j + \varepsilon$$

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1m} \\ 1 & x_{21} & x_{22} & \dots & x_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n1} & x_{n2} & \dots & x_{nm} \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{bmatrix}$$

$$y = X\beta + \varepsilon$$

The least-squares estimators must satisfy $\frac{\partial L}{\partial \beta}$, thus the least squares estimator of β is obtained by:

$$\beta = (X'X)^{-1}X'y$$

Note: To train different order of PR, the MATLAB® toolbox below suggested to be used.

<https://www.mathworks.com/matlabcentral/fileexchange/34765-polyfitn>

Metamodel 2: Radial Basis Function (RBF)

The RBF employs a linear combination of independent symmetric functions based on the Euclidean distances to compute the approximation function of response (interpolation model).

Each (x_n, y_n) influences $h(\mathbf{x})$ based on $\|\mathbf{x} - \mathbf{x}_n\|$

Standard form of RBF: $h(\mathbf{x}) = \exp(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2)$

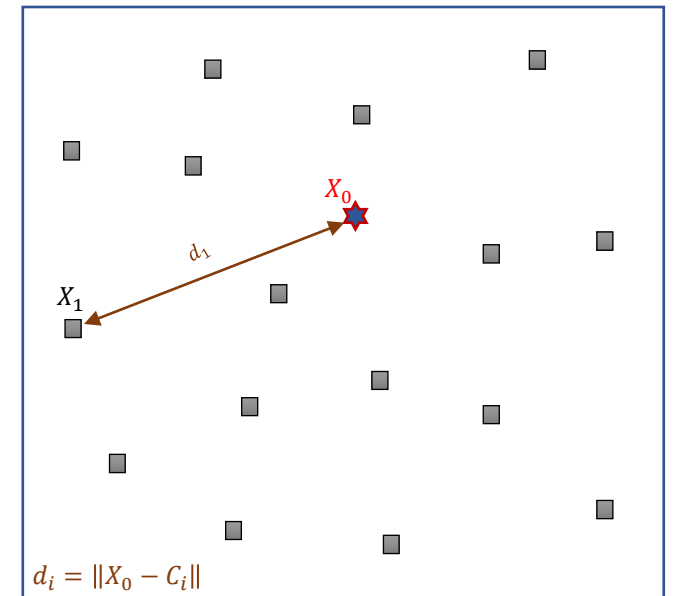
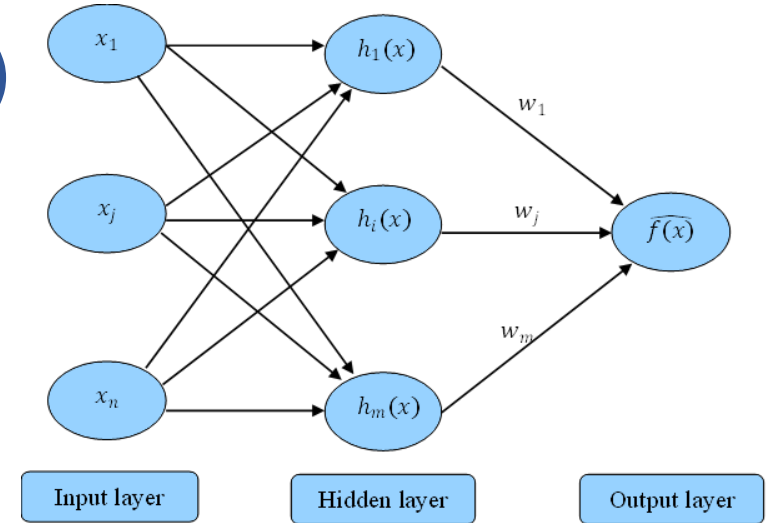
$$\hat{f}(\mathbf{x}) = \sum_{n=1}^N w_n \exp(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2)$$

The learning algorithm to find w_1, \dots, w_N by minimizing the sum square error of model based on the training data points $(x_1, y_1), \dots, (x_N, y_N)$

$$\text{Minimize } SSE = \sum_{n=1}^N \left(f(x_n) - \hat{f}(x_n) \right)^2$$

The RBF interpolate the sample points, the SSE is expected to be $SSE \approx 0$.

N equations N unknowns: $f(x_n) = \hat{f}(x_n)$ for $n = 1, \dots, N$



Radial Basis Function (RBF)

$$\boldsymbol{\varphi} = \begin{bmatrix} \exp(-\gamma\|\mathbf{x}_1 - \mathbf{x}_1\|^2) & \dots & \exp(-\gamma\|\mathbf{x}_1 - \mathbf{x}_N\|^2) \\ \exp(-\gamma\|\mathbf{x}_2 - \mathbf{x}_1\|^2) & \dots & \exp(-\gamma\|\mathbf{x}_2 - \mathbf{x}_N\|^2) \\ \vdots & \vdots & \vdots \\ \exp(-\gamma\|\mathbf{x}_N - \mathbf{x}_1\|^2) & \dots & \exp(-\gamma\|\mathbf{x}_N - \mathbf{x}_N\|^2) \end{bmatrix}$$

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \boldsymbol{\varphi} \cdot \mathbf{w} = \mathbf{Y}$$

If $\boldsymbol{\varphi}$ is invertible then

$$\mathbf{w} = \boldsymbol{\varphi}^{-1} \cdot \mathbf{Y}$$

“exact interpolation”

MATLAB code for RBF:

Input: Input data (designed training sample points), output data (collected responses from original model over designed sample points), and X0 (new sample point that we are interested to predict response for this point using RBF).

Output: The approximated response for point X0 .

MATLAB® code

```
function Ps = RBF_Sur(X_S,R_S,x0)
Nf = size(R_S,2);
for f = 1:Nf
Y = R_S(:,f);
N = size(X_S,1);
phi = zeros(N,N);
gamma = 0.05;
for i = 1 : N
xi = X_S(i,:);
for j = 1 : N
xj = X_S(j,:);
rb = exp(-gamma*norm(xi-xj)^2) ;
phi(i,j) = rb ;
end
end
W = inv(phi)*Y;
for s = 1 : N
xs = X_S(s,:);
rbs = exp(-gamma*norm(x0-xs)^2);
hx(s) = W(s) * rbs ;
end
Ps(f) = sum(hx);
end
end
```

Metamodel Validation

R-square Index: The R^2 coefficient is defined as:

$$R^2 = 1 - \frac{\sum_{j=1}^m (y_j - \hat{y}_j)^2}{\sum_{j=1}^m (y_j - \bar{y})^2} = 1 - \frac{\text{Mean Square Error}}{\text{Variance}}$$

Relative Maximum Absolute Error (RMAE)

While the larger magnitude of R-square indicates better overall accuracy, the smaller amount of RMAE indicates the smaller local error.

$$RMAE = \frac{\max\{|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \dots, |y_m - \hat{y}_m|\}}{\sqrt{\frac{1}{m} \sum_{j=1}^m (y_j - \hat{y}_j)^2}}$$

Cross-validation

The cross-validation method can be used when collecting new data or further information about simulation model is costly. The cross validation uses an existed data and does not require to re-run of the expensive simulation. This method also called leave- k -out cross-validation to validate metamodel (i.e., in each run k sample points would be removed from an initial training sample points). The leave-one-out cross validation ($k = 1$) is briefly explained in follows that is most popular than others.

Leave One-Out Cross-Validation

Step 1: Delete s^{th} input combination and relevant output from the complete set of l combination ($s = 1, 2, \dots, l$).

Step 2: Approximate the new model by employing $l - 1$ remain rows s_{-1} .

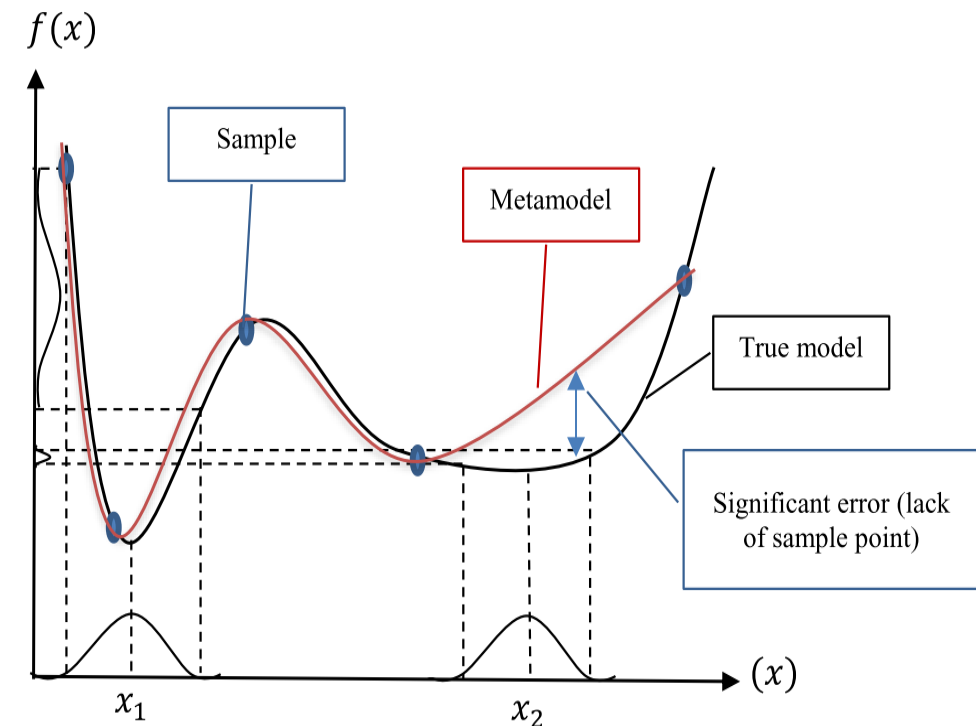
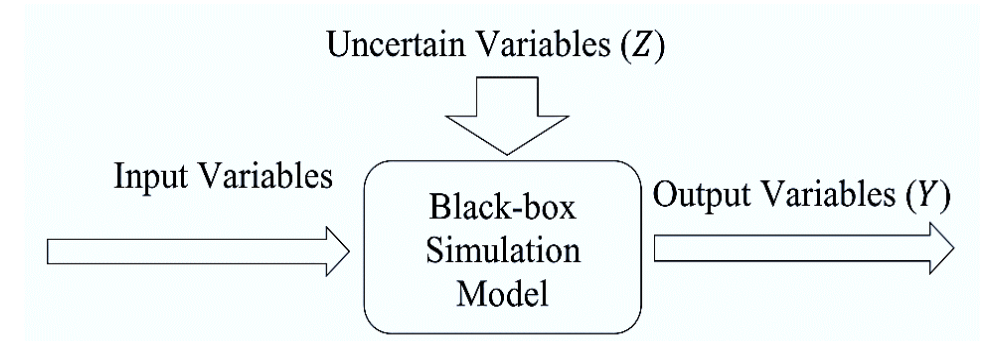
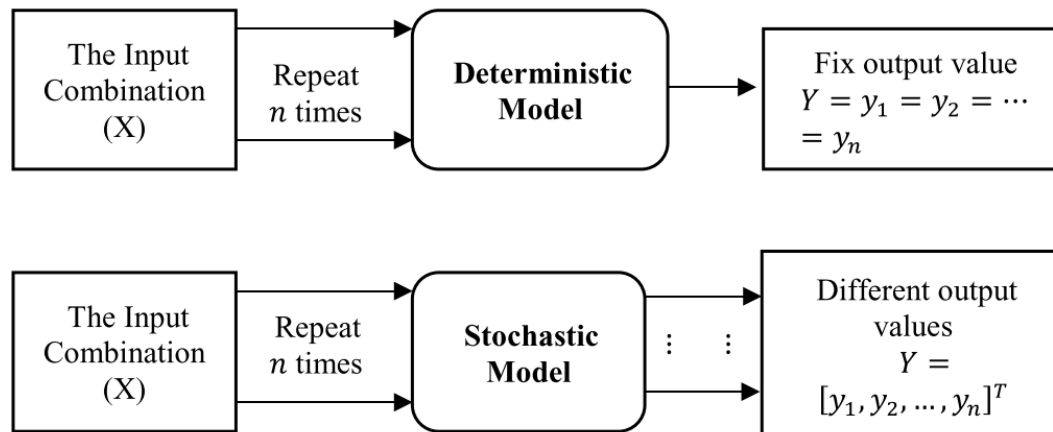
Step 3: Predict output for left-out point (s_{-1}) with metamodel which obtained from Step 2.

Step 4: Implementing the preceding three previous steps for all input combination (sample points) and computing n predictions (\hat{y}_s).

Step 5: The prediction result can be compared with the associated output in original simulation model. The total comparison can be done through a scatter plot or eyeball to decide whether or not metamodel is acceptable

Uncertainty in Model

In practice, most engineering problems have been affected by different sources of variations. One of the main challenges of SO is address uncertainty in the model, by a variety of approaches, such as robust optimization, stochastic programming, random dynamic programming, and fuzzy programming. Uncertainty is undeniable which effect on the accuracy of simulation results while making variability on them. In uncertain condition, robust SO allows us to define the optimal set point for input variables while keeping the output as closer as possible to ideal point, also with at least variation.



Metamodel-Based Robust Simulation Optimization

Robust simulation-optimization is about solving simulation model with uncertain data in a computationally tractable way. The main goal of robustness strategy is to investigate the best level of input factors setting for obtaining desirable output goal which is insensitive to the changeability of uncertain parameters.

Due to the stochastic nature of the simulation model, repeated runs of the simulation model with the same combination of input variables, lead to different outputs. Furthermore, each input combination ($s = 1, 2, \dots, l$) repeats m times ($r = 1, 2, \dots, m$), while m is the number of scenarios with uncertain variables. If $Y = (y_1, y_2, \dots, y_s)$ is the s -dimensional vector with the simulation output, then the mean and variance for each input combination are computed.

Algorithmic Steps

Step 1: Conduct crossed array experimental design

Crossed two sets of sample points with dimension $s \times r$, when ($s = 1, 2, \dots, l$) and ($r = 1, 2, \dots, m$). First for design factors ($X = 1, 2, \dots, n_x$) as an inner array ($s \times n_x$). Second for uncertain variables ($Z = 1, 2, \dots, n_z$) as an outer array ($r \times n_z$). In this context, LHS method is used in order to design sample points in both the inner and the outer array sets.

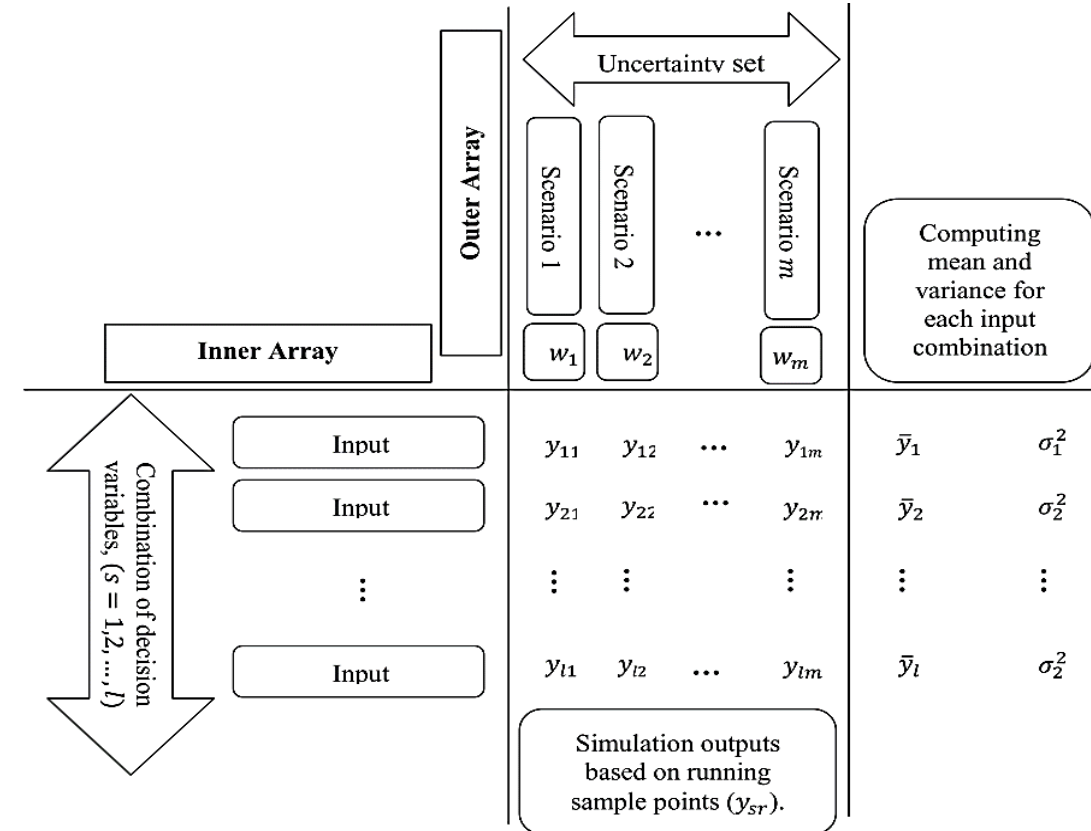
Metamodel-Based Robust Simulation Optimization

Step 2: Run simulation model

Run the simulation model for each crossed ($s \times r$) sample point and collect simulation outputs (y_{sr}).

Step 3: Compute mean and variance for each input combination

Given that $s = (1, 2, \dots, l)$ is the vector of sample points (input combination). Each input combination (s) is repeated m times based on uncertainty scenarios. Therefore, Y is the $s \times m$ matrix of simulation's outputs, and mean \bar{y}_s and variance σ_s^2 of the simulation model are computed by



$$\bar{y}_s = \sum_{r=1}^m w_r \cdot y_{sr}, \quad \text{for } (s = 1, 2, \dots, l) \quad \sigma_s^2 = \sum_{r=1}^m w_r \cdot y_{sr}^2 - \left(\sum_{r=1}^m w_r \cdot y_{sr} \right)^2, \quad \text{for } (s = 1, 2, \dots, l)$$

Metamodel-Based Robust Simulation Optimization

Step 4: Fit input/output data with surrogate model

Fit one surrogate model (e.g., Kriging, RBF, or PR) for mean and another one for variance of output. Here, Kriging as a modern global surrogate model is used.

Step 5: Validate both surrogate models, using leave-one-out cross-validation

Step 6: Estimate the Pareto frontier by obtaining robust optimal solutions performing dual response surface methodology

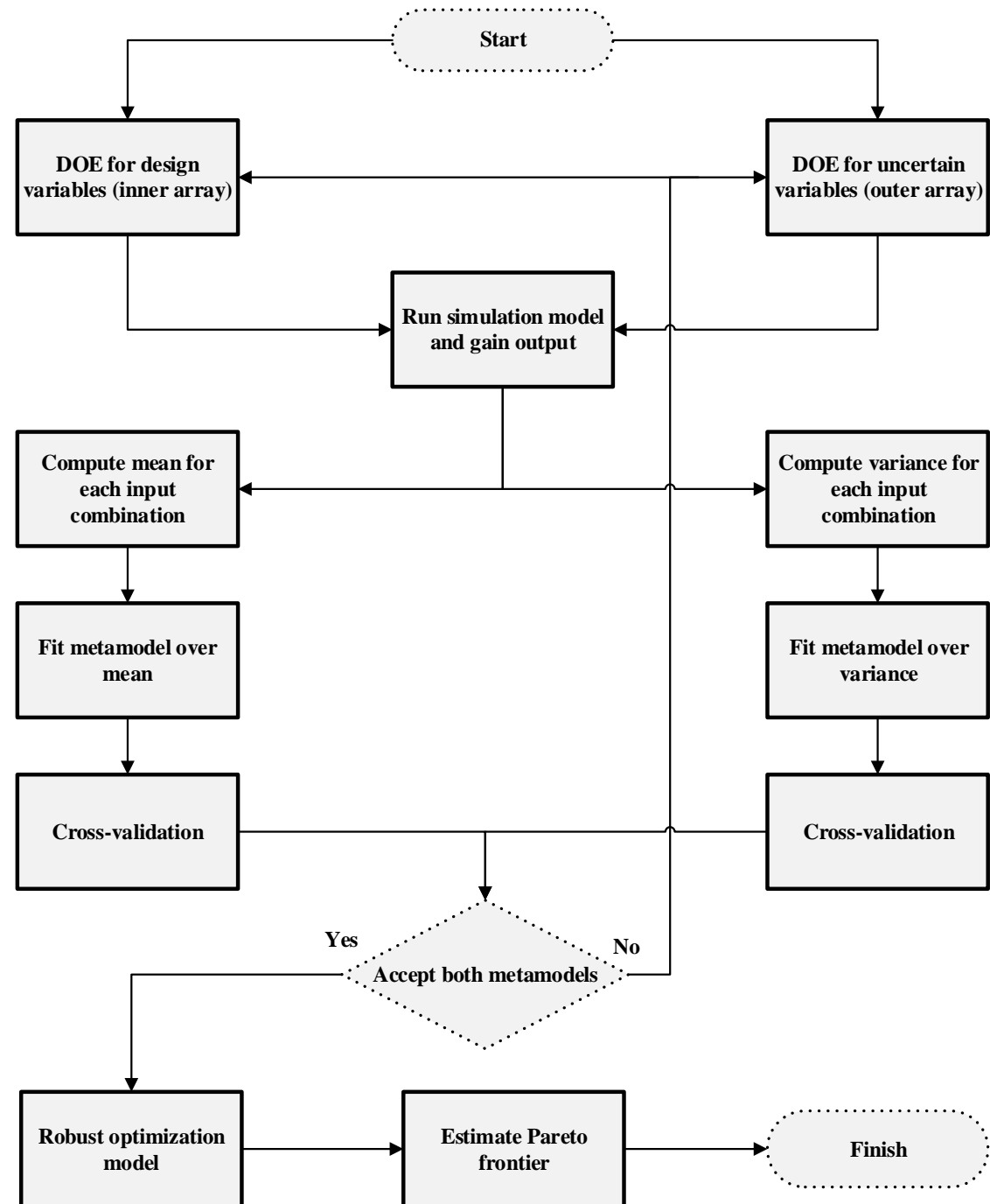
Robust dual
surface model

*Min or Max: **Mean***
*Subject to: **Std** $\leq \epsilon$*

The optimal result depends on value of ϵ which can be chosen by the decision maker. Varying this magnitude provides the capture of Pareto frontier (also called Pareto optimal efficiency) to make trade-off between mean and variance of the model.

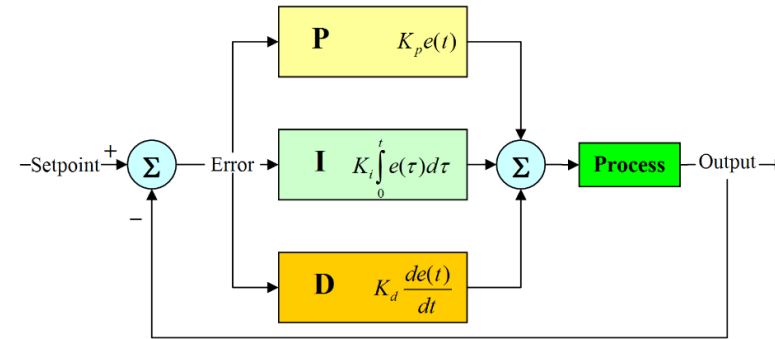
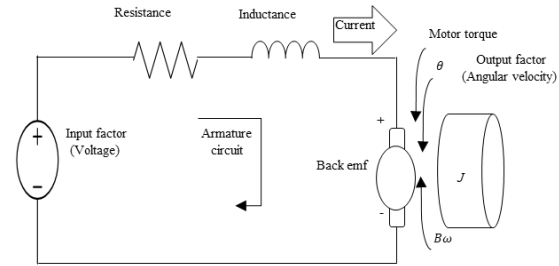
Metamodel-Based Robust Simulation Optimization

Flowchart of the proposed approach

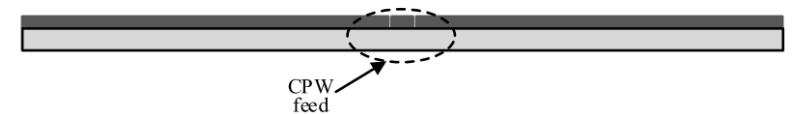
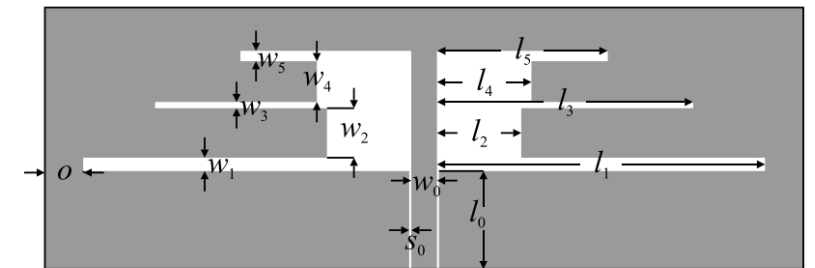
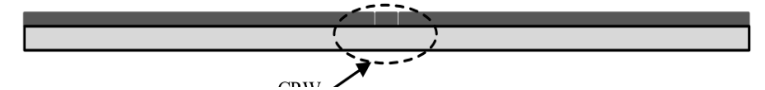
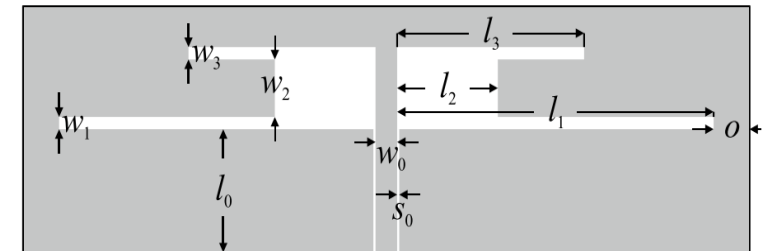
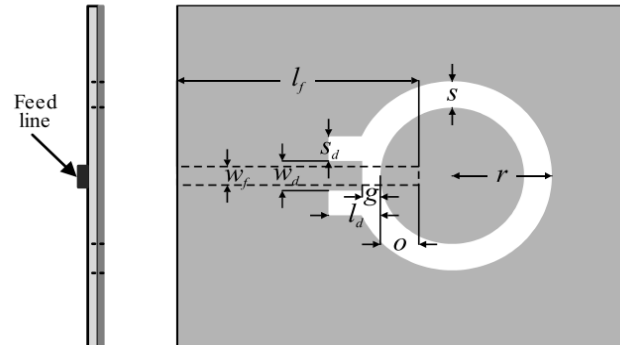


Engineering Application (Examples in Electronics and Electrical Engineering)

Optimal Control Design



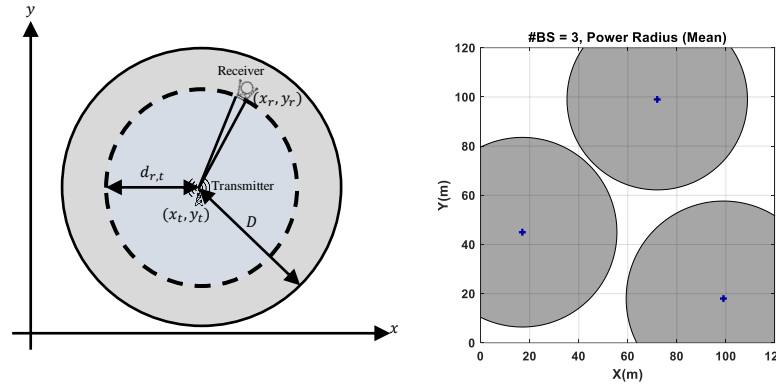
Optimal Antenna Design



Engineering Application (Examples in Electronics and Electrical Engineering)

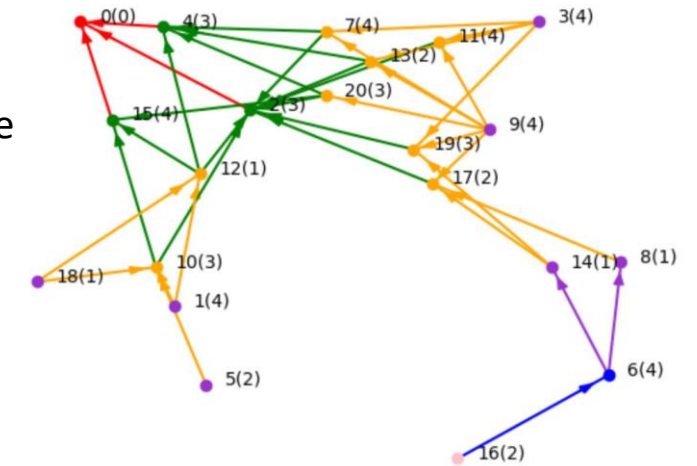
Optimal Placement of Antennas

(Maximize coverage, minimize overlapping, minimize energy consumption)



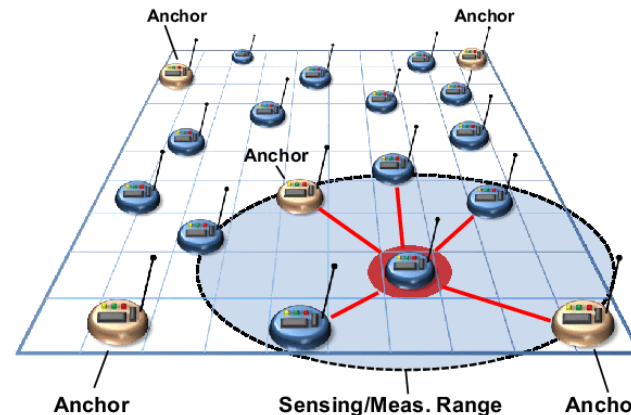
Optimization of Packet Transmission

(Scheduling and Node Parent Selection, Traffic Aware Scheduling Algorithm, IEEE 802.15.4e time slotted channel hopping)



Wireless Sensor Networks

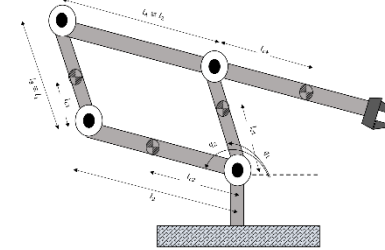
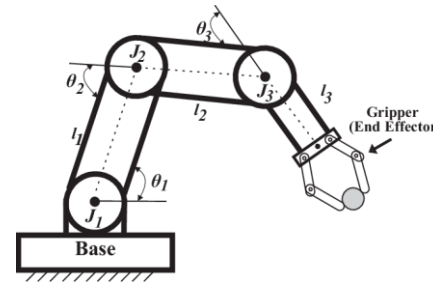
(Localization, positioning of the sensor nodes)



Engineering Application (Examples in Electronics and Electrical Engineering)

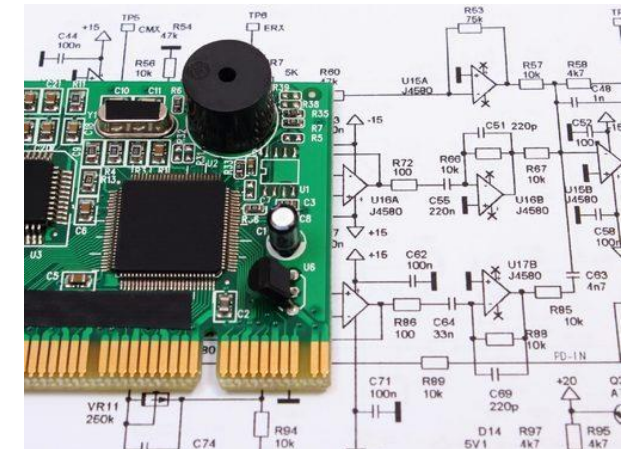
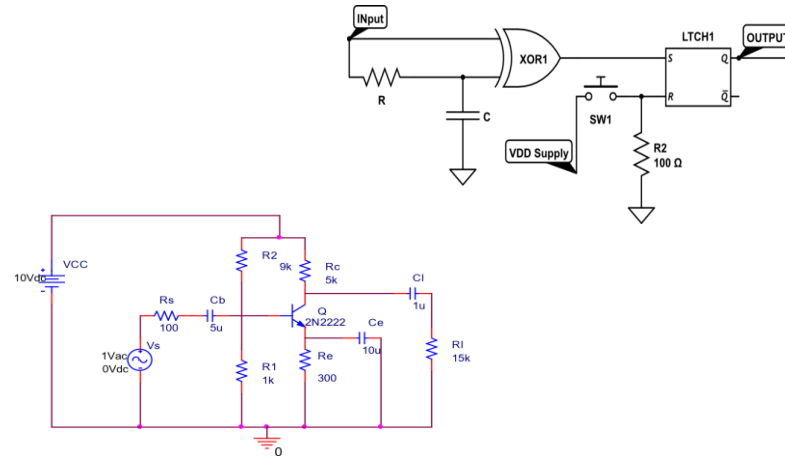
Robotics

(To optimal trajectory path design, real-time intelligent control, and optimizing robot performance.)



Electronics Circuit Design

(Evolutionary electronics, analog circuits, digital electronics, logic optimization)



References

Books

- Blondin, M. J., & Pardalos, P. M. (2019). Computational Intelligence and Optimization Methods for Control Engineering (Vol. 150). Retrieved from <http://link.springer.com/10.1007/978-3-030-25446-9>
- Dellino, G., & Meloni, C. (2015). Uncertainty Management in Simulation- Optimization of Complex Systems. Boston, MA, USA: Springer. Springer.
- Kanevski, M., Pozdnoukhov, A., & Timonin, V. (2009). Machine Learning for Spatial Environmental Data. EPFL press.
- Kleijnen, Jack, P. C. (2020). Chapter 10 Kriging: methods and applications. In Model order reduction (Vol. 1, pp. 1–466).
- Kleijnen, J. P. C. C. (2015). Design and analysis of simulation experiments. Springer Proceedings in Mathematics & Statistics (Vol. 231). Springer, Cham.
- Myers, R., C.Montgomery, D., & Anderson-Cook, M, C. (2016). Response Surface Methodology: Process and Product Optimization Using Designed Experiments-Fourth Edition. John Wiley & Sons.

References

Review Papers:

- Amaran, S., Sahinidis, N. V., Sharda, B., & Bury, S. J. (2016). Simulation optimization: a review of algorithms and applications. *Annals of Operations Research*, 240(1), 351–380.
- Jin, R., Du, X., & Chen, W. (2003). The use of metamodeling techniques for optimization under uncertainty. *Structural and Multidisciplinary Optimization* (Vol. 25).
- Kleijnen, Jack, P. C. (2007). Kriging Metamodeling in Simulation: A Review. Tilburg University, CentER Discussion Paper; Vol. 2007- 13. Operations research.
- Li, Y. F., Ng, S. H., Xie, M., & Goh, T. N. (2010). A systematic comparison of metamodeling techniques for simulation optimization in Decision Support Systems. *Applied Soft Computing*, 10(4), 1257–1273.
- Soares do Amaral, J. V., Montevechi, J. A. B., Miranda, R. de C., & Junior, W. T. de S. (2022). Metamodel-based simulation optimization: A systematic literature review. *Simulation Modelling Practice and Theory*, 114(August 2021).
- Wang, G., & Shan, S. (2007). Review of Metamodeling Techniques in Support of Engineering Design Optimization. *Journal of Mechanical Design*, 129(4), 370–380.

References

Sample Publications by Presenter (Amir Parnianifard) in the scope of this presentation

- Parnianifard, A, Azfanizam, A. S., Ariffin, M. K. A., & Ismail, M. I. S. (2018). Kriging-Assisted Robust Black-Box Simulation Optimization in Direct Speed Control of DC Motor Under Uncertainty. *IEEE Transactions on Magnetics*, 54(7), 1–10.
- Parnianifard, Amir, Azfanizam, A. S., Ariffin, M. K. A., & Ismail, M. I. S. (2018). An overview on robust design hybrid metamodeling : Advanced methodology in process optimization under uncertainty. *International Journal of Industrial Engineering Computations*, 9(1), 1–32.
- Parnianifard, Amir, Azfanizam, A. S., Ariffin, M. K. A., & Ismail, M. I. S. (2019a). Comparative study of metamodeling and sampling design for expensive and semi-expensive simulation models under uncertainty. *SIMULATION*, 96(1), 89–110.
- Parnianifard, Amir, Azfanizam, A. S., Ariffin, M. K. A., & Ismail, M. I. S. (2019b). Crossing weighted uncertainty scenarios assisted distribution-free metamodel-based robust simulation optimization. *Engineering with Computers*, 36(1), 139–150.
- Parnianifard, Amir, Chaudhary, S., Mumtaz, S., Wuttisittikulkij, L., & Imran, M. A. (2023). Expedited surrogate-based quantification of engineering tolerances using a modified polynomial regression. *Structural and Multidisciplinary Optimization*, 66(3), 61.
- Parnianifard, Amir, Mumtaz, S., Chaudhary, S., Imran, M. A., & Wuttisittikulkij, L. (2022). A data driven approach in less expensive robust transmitting coverage and power optimization. *Scientific Reports*, 12, 17725.
- Parnianifard, Amir, Rezaie, V., Chaudhary, S., Imran, M. A., & Wuttisittikulkij, L. (2021). New Adaptive Surrogate-Based Approach Combined Swarm Optimizer Assisted Less Tuning Cost of Dynamic Production- Inventory Control System. *IEEE Access*, 9, 144054–144066.
- Parnianifard, Amir, Zemouche, A., Chancharoen, R., Imran, M. A., & Wuttisittikulkij, L. (2020). Robust optimal design of FOPID controller for five bar linkage robot in a Cyber-Physical System: A new simulation-optimization approach. *PLOS ONE*, 15(11), e0242613.

**THANK
YOU**