
RELIABILITY ANALYSIS USING RBF SURROGATE MODELING

Rahman Alam

Applied Mechanics Department
Indian Institute of Technology Delhi
Haus KhazDelhi
<https://home.iitd.ac.in>

ABSTRACT

Here a brief practical approach on reliability analysis using the RBF function will be done and we will be using direct Monte Carlo method to estimate the reliability compare with the surrogate modeling-based reliability with lesser number of samplings compare to the direct Monte Carlo method. RBF based surrogate modeling is a interpolation method which pass through all the available points. We also want to increase the efficiency of the MCS so we will be making the approximate limit state function using the augmented RBFs.

1 Introduction

Reliability analysis is a very important component of any system/subsystem, process, product design which still getting developed. If we see the MPP point method for reliability analysis, here we find the most probable p[point in the design space and we transform the LS function into the standard Gaussian and the approximate the function using the Taylor series to get our reliability index. We can then use FORM (first order reliability method) or SORM (second order reliability method depending upon the order of approximation. As this method need a LS function which we need to derive or if the system is complex then we must do the FE simulation, which is costly process. The alternate method is the MCS method which do not need derivatives and gradient to give the reliability of the system.

But in case of complex problems, we need very accurate simulation which again make the process very tedious and computationally very expensive. Here comes the concept of surrogate modeling where we develop a response fiction for the practical problem with the limited points. These are approximate model but fairly can give the result for our reliability analysis.

In this method the analysis part with software will be replaced by the surrogate model which will give the output for the reliability analysis algorithms. Then the FORM/SORM method can be applied on these surrogates which is computationally inexpensive and fast.

There are several surrogate modeling methods depending on the practical complexity of the model. The most popular among the surrogate model is the polynomial-based response surface model. Here we can use different sampling method to accurate fit our surrogate's method then can be combined with higher order effect, gradient based search methods to improve the response. Other surrogate models like kriging, ANN, support vector machine and ensemble of surrogates are developed to use with practical problems to fit well with the sample points.

Among all those surrogate model we have RBF surrogate model which is also a interpolation method whose inputs are the scattered data. As RBF model fits well with the global approximation that's why it is most used model. There is various type of basis function which can be use to build your surrogate models as Inverse quadratic, Gaussian multiquadric and spline functions.

2 Aims and Objectives

Our aim is work with RBF to give the surrogate model which will be efficient and well accurate to give us the reliability of the system. We will be using augmented function to perform the adaptive and successive RBF. In SRBF we update the sample to give a better model to predict the reliability as the RBF function is depending on the number of samples. So more the number of samples more accurate our model would be but more samples need more computation power as well, so we need a appropriate number of sample. But as we don't know exactly how much sample will be needed to

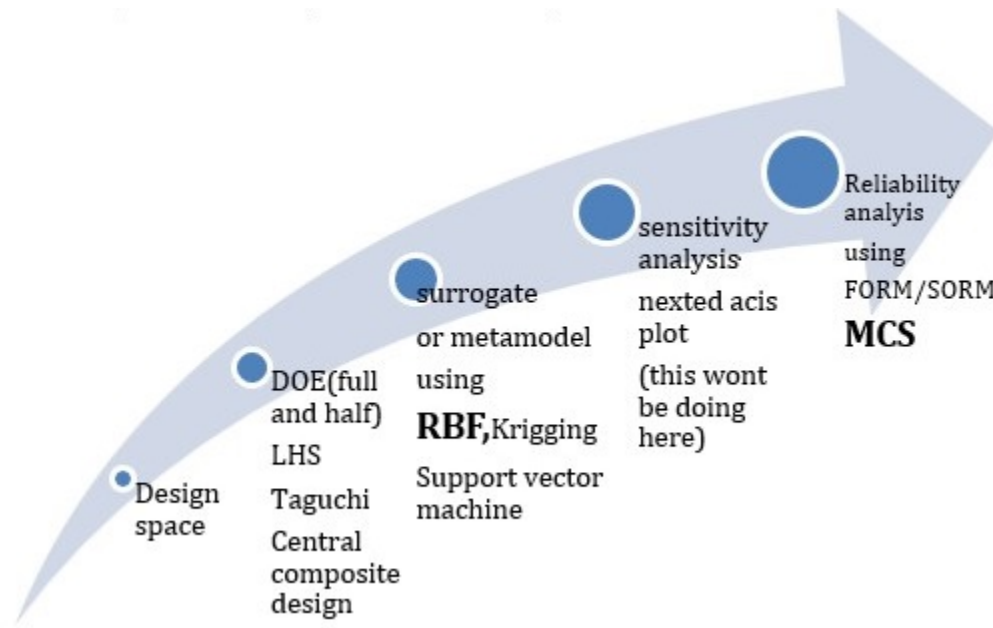


Figure 1: High level map

give most accurate model, so it is a challenge for the researcher.

Some how this appropriate sample need can be done by SRBF as it will automate the process and it will automatically improve the model by changing/increasing the number of samples. That's why we will be using SRBF technique.

So, we will be using 3 augmented RBFs to generate surrogate models of a LS function. We will be using most accurate augmented RBFs on which already work is available. We can use these surrogates with MCS method to get the failure probability and will be comparing with the direct MCS method.

3 High level maps for the study

So here we can see how we can proceed. We can choose first what we want study i.e., what is our system and the design space having the design variables. Then we must choose the samples for which we will be doing the DOE, or any other sampling process mentioned just take care these methods are not limited there are plenty of method for doing the sampling here in this paper we have mentioned the most used sampling process. Then with sample we get our metamodel which can be build again using the method mention in the figure 1. Among those process we will be using RBF modeling to build our model. We won't be doing the sensitivity analysis as we know our accurate function as well in case when we don't know our actual function then we can do sensitivity analysis as well-turned we will be doing MCS for our failure probability and compare it with the direct MCS.

4 Samples Generation for Examples

We will first generate the design variable samples and then evaluate the function to generate limit state values on those random samples. Mean and variance for various distribution is given and we have taken the same.

For the problem 1 there are two variable x_1 and x_2 whose means are 0 and standard deviation are 1 basically they are following standard normal distribution. For the problem 2 we have 3 variable and l, b, h as length width and height of a cantilever beam. where Young's modulus (E) and Load (P) have taken to be deterministic. Length(l) follows log normal distribution with 30 mean and 3.0 as a variance. Width(b) follows Gaussian distribution whose mean is 0.8359 and variance is 0.08. Height(h) follows log normal distribution with mean 2.5093 and 0.25 as standard deviation.

we will generate sample using the python random function. Which will be using standard normal distribution, Log normal distribution and the Gaussian distribution. I will be sharing the data generation for 2 examples as below. Here is the code for 10 sample generation. we will use map, lambda and def function to create the data for our reliability testing. Then we will use the radial basis function to train the surrogate model which will be used to predict the reliability by direct MCS and compare results with SRBF model which will need far less sample than the direct MCS.

Random Variable	Mean	Standard Variation	Distribution type
x1	0	1	std. Normal
x2	0	1	std. Normal

Table 1: Problem 1 Variables

Random Variable	Mean	Standard Variation	Distribution type
l	30	3.0	Log Normal
b	0.8359	0.08	Gaussian
h	2.5093	0.25	Log Normal

Table 2: Problem 2 Variables

4.1 Necessary Library import

```
import numpy
import pandas as pd
import datetime
import random
import math
```

4.2 Problem 1 Data

```
x1=numpy.random.normal(loc=0,scale=1,size=10)
x2=numpy.random.normal(loc=0,scale=1,size=10)
```

```
X_1=array(x1)
```

```
X_2=array(x2)
```

Output-

```
X_1= [ 0.00330996 0.48446039 0.36799503 -0.19686939 0.59304055 0.26374834 -0.2562668 -2.21422784 0.02830131
-1.74183851]
```

```
X_2= [ 1.11552738 -0.76555621 0.63381837 1.51305234 0.25714022 0.86889619 0.03219618 -0.19176232
-1.24587234 -0.02940761]
```

Function evaluation

```
Y=map(lambda x,y: (((math.exp(0.2*x)+6.2) -(math.exp(0.47*y)+5.0))),X_1 X_2)
print(list(Y))
```

Output-

```
Y= [0.5113898690337582, 1.603933559770474, 0.9293552517783432, 0.12509086937670943, 1.1974662953536672,
0.7497851874401977, 1.134790661783752, 0.9283924329726512, 1.6488823400697648, 0.9195657885154578]
```

4.3 problem 2 Data Generation

Problem 2 length data

mul, signal = 30, 3 mean and standard deviation

```
L= numpy.random.lognormal(mul, signal, 10)
```

Problem 2 Width data

```
B = []
```

```
mu = 0.8359
```

```
sigma = 0.8
```

```
for i in range(10):
```

```
    b = random.gauss(mu, sigma)
```

```
    B.append(b)
```

```

# Problem 2 Height data
muh, sigmah = 2.5093, 0.25
H = numpy.random.lognormal(muh, sigmah, 10)

# Problem 2 Function Evaluation
E=10000000
P=80
mul, sigmal = 30, 3
L = numpy.random.lognormal(mul, sigmal, 10)
l=list(L)
B = []
mu = 0.8359
sigma = 0.8
for i in range(10):
    b = random.gauss(mu, sigma)
    B.append(b)
b=list(B)
muh, sigmah = 2.5093, 0.25
H = numpy.random.lognormal(muh, sigmah, 10)
h=list(H)
def Y(x,y,z):
    return ((0.15)-(4*P*x**3)/(E*y*z**3))
s=map(Y,l,b,h)
print (list(s))

```

5 Direct MCS Results

As we have generated the sample so now we would be doing the Monte Carlo simulation to get the probability of failure for the mentioned examples in the research paper. We will first mention all the parameters and define our function so that we can call it whenever we need it.

Counting of failure point is an important part for this technique so we will use list comprehension for the counting of our failure points with help for for loops.

5.1 Problem 1

Problem 1 is just a simple highly non linear function so we can not get a probability of failure associated with the reliability index with the FORM/SORM/FOSM method as these methods are good for weakly non linear function we will approach with two methods here one will be the direct MCS which will work as a benchmark for us to compare with our second method which is surrogate based reliability analysis.

The Limit state function for the problem is-

$$g(\mathbf{x}) = \exp(0.2x_1 + 6.2) - \exp(0.47x_2 + 5.0)$$

Lets Discuss the Code-

we will first import the necessary library

```

import numpy
from numpy import exp, arange
from pylab import meshgrid, cm, imshow, contour, clabel, colorbar, axis, title, show
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats as sys
from scipy.stats import norm
import seaborn as sns
import pandas as pd

```

```
import random
```

Lets define our function-

```
def zfunc(x1, x2) :  
    return(exp(0.2 * x1 + 6.2) - exp(0.47 * x2 + 5.0))
```

Generate samples-

```
x1 = numpy.random.normal(0, 1, 1000)  
x2 = numpy.random.normal(0, 1, 1000)
```

As our function is 2D so we can visualise it in 3-dimensional as-

```
X,Y = meshgrid(x1, x2)gridofpoint  
Z = zfunc(X,Y)evaluationofthefunctiononthe  
fig = plt.figure()  
im = imshow(Z, cmap = cm.RdBu)drawingthe  
addingtheContourlineswithlabels  
cset = contour(Z, arange(-1, 15, 0.2), linewidths = 2, cmap = cm.Set2)  
clabel(cset, inline = True, fmt = ' colorbar(im)addingthecolobarontheright
```

```
latex fashion title  
title('z = exp(0.2 * x1 + 6.2) - exp(0.47 * x2 + 5.0)')  
show()  
from mpl_toolkits.mplot3d import Axes3D
```

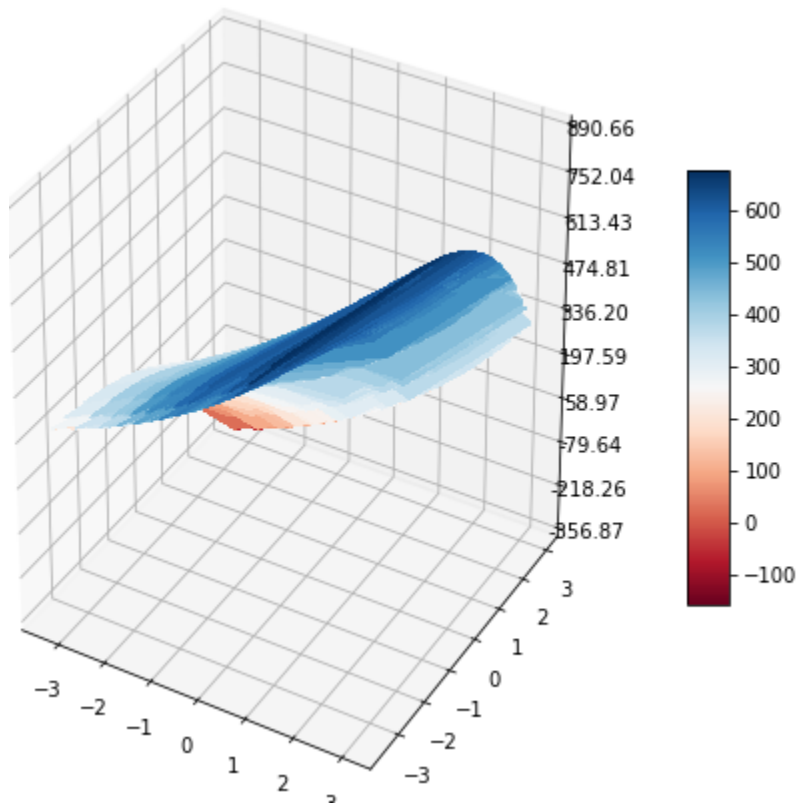


Figure 2: LS function

```
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import matplotlib.pyplot as plt
```

```
fig = plt.figure(figsize=(8,8))
```

```
ax = fig.gca(projection='3d')
surf = ax.plot_surface(X, Y, Z, rstride = 1, cstride = 1,
    cmap = cm.RdBu, linewidth = 0, antialiased = False)
```

```
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter(''))
```

```
fig.colorbar(surf, shrink=0.5, aspect=10)
```

```
plt.show()
```

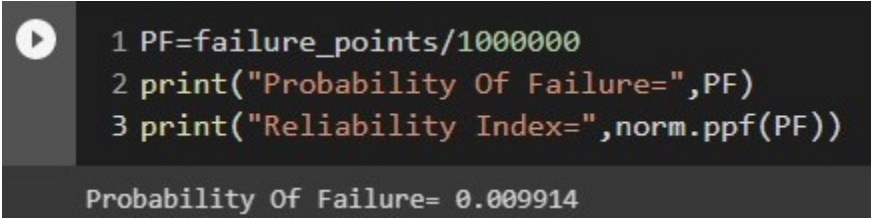
Calculation of Failure Points-

```
def countNegative(Z, n, m):
    count = 0
    Follow the path shown using arrows above
    for i in range(n):
        for j in range(m):
            if Z[i][j] < 0:
                count += 1
    return count
```

Probability of Failure Calculation-

```
PF=failure_points/1000000
print("Probability Of Failure = ", PF)
print("Reliability Index = ", norm.ppf(PF))
failure_points = (countNegative(Z, 1000, 1000))
```

From the above we were able to get the probability of failure and Reliability of index which is close to the mentioned in the research paper which is around 0.009914 which we are getting after multiple runs.



```
1 PF=failure_points/1000000
2 print("Probability Of Failure=",PF)
3 print("Reliability Index=",norm.ppf(PF))

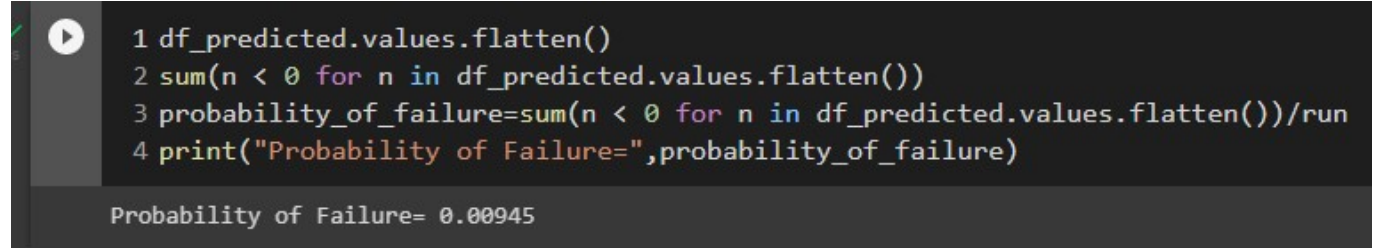
Probability Of Failure= 0.009914
```

Figure 3: Probability of failure from Direct MCS

In paper we are getting probability of failure as 0.009372 which we also have got in few runs. so we can run this simulation 100 times and take a mean of those to get more close result.

5.2 Problem 1-Multi Quadratic RBF model

Here I have developed model using Multi Quadratic RBF model to predict the output using the surrogate and then predicted the probability of failure using the model. which is very close to 0.00945 which is very close to directly predicted MCS. I have fitted the surrogate model only with the 40 points of data points and predicted the output with the trained model. this way the result error is only 4.6 percent.



```
1 df_predicted.values.flatten()
2 sum(n < 0 for n in df_predicted.values.flatten())
3 probability_of_failure=sum(n < 0 for n in df_predicted.values.flatten())/run
4 print("Probability of Failure=",probability_of_failure)

Probability of Failure= 0.00945
```

Figure 4: Probability of failure from RBF

6 Problem 2-Cantilever Beam

In this problem we are going to study the reliability of a cantilever beam which has been acted by a point load. Displacement at tip is limited to 0.15 inch. So the limit state function for this problem is given as.

$$g(l, b, h) = 0.15 - \frac{4Pl^3}{Ebh^3}$$

Here the E and P are deterministic variables whose values are 10000000 psi and 80 lb respectively, and the other three variables are probabilistic whose means and deviations are already given in table 2. As this is a 3-dimensional problem, so we will not be able to visualize it as above problem, so we will calculate the probability of failure as below.

Start the Code-

As necessary libraries are going to remain the same as problem 1.

We will now define the variables and our LS function as below.

```
n=1000000
E=10000000
P=80
muL=30;
muB= 0.8359
muH=2.5093
```

```
sigL=3
sigB=0.08
sigH=0.25
```

```
normal_std_L = np.sqrt(np.log(1 + (sigL/muL)**2))
normal_mean_L = np.log(muL) - normal_std_L**2/2
```

```
normal_std_H = np.sqrt(np.log(1 + (sigH/muH)**2))
normal_mean_H = np.log(muH) - normal_std_H**2/2
```

```
def Y(x,y,z):
return ((0.15)-(4*P*x**3)/(E*y*z**3))
```

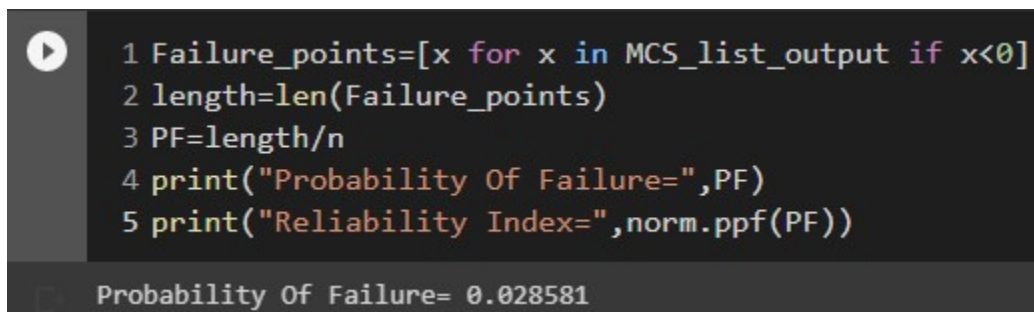
Sample generation and LS function evaluation-

```
MCS_list_output = []
mcs_L_input = []
mcs_B_input = []
mcs_H_input = []
for i in range(n): mcs_L_sample = np.random.lognormal(normal_mean_L, normal_std_L)
mcs_B_sample = random.gauss(muB, sigB)
mcs_H_sample = np.random.lognormal(normal_mean_H, normal_std_H)
```

```
mcs_sample = Y(mcs_L_sample, mcs_B_sample, mcs_H_sample)
MCS_list_output.append(mcs_sample)
mcs_L_input.append(mcs_L_sample)
mcs_B_input.append(mcs_B_sample)
mcs_H_input.append(mcs_H_sample)
```

Probability of failure calculation-

```
Failure_points = [x for x in MCS_list_output if x < 0]
length = len(Failure_points)
PF = length/n
print("Probability Of Failure = ", PF)
print("Reliability Index = ", norm.ppf(PF))
```



```
1 Failure_points=[x for x in MCS_list_output if x<0]
2 length=len(Failure_points)
3 PF=length/n
4 print("Probability Of Failure=",PF)
5 print("Reliability Index=",norm.ppf(PF))
```

Probability Of Failure= 0.028581

Figure 5: Probability of failure

Here as we can see with multiple runs we will be able to get the probability of failure as 0.028581. which is very close to the probability of failure mentioned in the paper as 0.02823 from the direct MCS.

6.1 Problem 2-Multi Quadratic RBF model

Here I have developed model using Multi Quadratic RBF model to predict the output using the surrogate and then predicted the probability of failure using the model, which is very close to 0.0030165 which is very close to directly predicted MCS. I have fitted the surrogate model only with the 20 points of data points and predicted the output with the trained model. This way the result error is 25 percent.

```
1 df_predicted_problem_2.values.flatten()
2 sum(n < 0 for n in df_predicted_problem_2.values.flatten())
3 probability_of_failure=sum(n < 0 for n in df_predicted_problem_2.values.flatten())/n_P2
4 print("Probability of Failure=",probability_of_failure)
```

Probability of Failure= 0.02158

Figure 6: Probability of failure from RBF-Beam Problem

Note-Training Model will be attached as python file.

7 Problem 3-Reinforced Concrete Beam Section

This problem deal with the beam on static form which is following highly nonlinear limit state function given as.

$$g(\mathbf{x}) = x_1 x_2 x_3 - x_4 \frac{x_1^2 x_2^2}{x_5 x_6} - M_n$$

It is consisting of six independent variable as shown in the table and having a deterministic M_n value as 211.20×10^6 .

Random Variable	Unit	Mean	Standard Variation	Distribution type
x_1	(mm ²)	1,260	63	Log-normal
x_2	(N/mm ²)	250	17.5	Log-normal
x_3	(mm)	770	10	Log normal
x_4	(N/A)	0.55	0.055	Log-normal
x_5	(N/mm ²)	30	4.5	Log-normal
x_6	(mm)	250	5	Log-normal

Table 3: Problem 3 Variables

7.1 Probability of failure from Direct MCS

The direct MCS for the LS function is giving as 0.403734 which is not very close to the direct MCS we are getting from the research paper as 0.1102. Hence an error of 266 percent.

So now we will train our model with only few design points and Fit a surrogate model as a radial basis function and get the probability of failure from the trained model.

So we can see here we have various distribution from which we need to sample from. These sample can be generated easily with random generation and then save as a CSV file for the further process

```
1 Failure_points=[x for x in MCS_list_3_output if x<0]
2 length=len(Failure_points)
3 PF=length/n3
4 print("Probability Of Failure=",PF)
```

Probability Of Failure= 0.403734

7.2 Probability of failure from RBF Surrogate Modeling

After training of model for just 80 data points we are able to get the probability of failure very close to 0.4367 which is very close to direct MCS result from our study but not close to the RBF multiquadretic results of the research paper.

```
1 df_predicted_problem_3=pd.read_csv('y_pred.dat')
2 df_predicted_problem_3.values.flatten()
3 sum(n < 0 for n in df_predicted_problem_3.values.flatten())
4 probability_of_failure=sum(n < 0 for n in df_predicted_problem_3.values.flatten())/n3
5 print("Probability of Failure=",probability_of_failure)
```

Probability of Failure= 0.436704

8 Problem 4-Burst Margin of rotating disk

This problems deals with the reliability of a disk under the angular velocity ω the other parameters are as shown in the table below. The burst margin M_b , of the disk can be shown like this- if the lower bound the Burst margin is 0.37473 is

$$M_b(\alpha_m, S_u, \rho, \omega, R_o, R_i) = \sqrt{\frac{\alpha_m S_u}{\left(\frac{\rho \left(\frac{2\omega\pi}{60} \right)^2 (R_o^3 - R_i^3)}{3(385.82)(R_o - R_i)} \right)}}$$

used, then the LS function for our problem will be.

$$g(\mathbf{x}) = M_b(\alpha_m, S_u, \rho, \omega, R_o, R_i) - 0.37473$$

Random Variable	Unit	Mean	Standard Variation	Distribution type
α_m	(N/A)	0.9378	0.04655	weibull ^a
S_u	(lb/in ²)	220000	5000	Gaussian
ρ	(lb/in ³)	0.29	0.00577	Uniform ^b
ω	(rmp)	21000	1000	Gaussian
R_o	(in)	24	0.5	Gaussian
R_i	(in)	8	0.3	Gaussian

Table 4: Problem 4 Variables

^aScaleParameter = 25.508, ShapeParameter = 0.958

^bUniformly Distributed over 0.28 – 0.30

So we can see here we have various distribution from which we need to sample from. These sample can be generated easily with random generation and then save as a CSV file for the further process.

8.1 Probability of failure from Direct MCS

The direct MCS for the LS function is giving as 0.001162 which is very close the direct MCS we are getting from the research paper as 0.001010. Hence an error of 2.5 percent.

So now we will train our model with only few design points and Fit a surrogate model as a radial basis function and get the probability of failure from the trained model.

we will 70 sample as our number of training samples then predict the 1 million points from that trained model for our reliability analysis. this whole process's will take 2 minute to simulate. which is quite fast and efficient.

```
1 Failure_points=[x for x in MCS_list_output_4 if x<0]
2 #print(Failure_points)
3 length=len(Failure_points)
4 PF=length/n_4
5 print("Probability Of Failure=",PF)
6 #print("Reliability Index=",norm.ppf(PF))
```

Probability Of Failure= 0.001162

8.2 Probability of failure from RBF Surrogate Modeling

After training of model for just 70 data points we are able to get the probability of failure very close to 0.001091 which is very close to direct MCS and RBS as 0.001162 and 0.001010 of the research paper.

```
Using problem4_trained_model.csv to predict values...
Predicted values stored in "y_pred.dat"
```

```
[ ] 1 df_predicted_problem_4=pd.read_csv('y_pred.dat')
     2 df_predicted_problem_4.values.flatten()
     3 sum(n < 0 for n in df_predicted_problem_4.values.flatten())
     4 probability_of_failure=sum(n < 0 for n in df_predicted_problem_4.values.flatten())/n_4
     5 print("Probability of Failure=",probability_of_failure)
```

Probability of Failure= 0.001091

9 Conclusion

So as we have discussed the surrogate modelling added with the MCS method is very strong algorithm to get the probability of failure. Further we have used multiquadretic RBF here we can also use different kind of function to get the results as well. The numerical example which are discussed are typical engineering problems which can be easily taken care by these approaches with very efficient manner.

Further a sequential approach for SRBF can be done to get the optimal number of samples size.