Investigate Global Optimization of Nonlinear System Using Generalized Simulated Annealing

Fang YU

Department of Chemical & Environmental Engineering, University of Cincinnati

(Term Project for ENVE 6026 Environmental/Hydraulic Systems Analysis)

(Spring 2016, 04/17/2016)

Executive Summary

This report investigated global optimization of non-linear system using Generalized Simulated Annealing (**GenSA**) algorithm. It was compared with the popular Differential Evolution (**DEoptim**) algorithm for solving Rastrigin function and Thomson's problem in terms of accuracy, number of successful runs and average number of function calls. The R functions and scripts are documented in the APPENDICES of this report. **GenSA** outperformed **DEoptim** in terms accuracy, efficiency and speed. **GenSA** is a great choice for global optimization of complicated nonlinear system with multiple minima/maxima.

Acknowledgements

I would like acknowledge Professor Drew McAvoy for his guidance and help on this project.

Table of Contents

List of Figures

List of Tables

I. INTRODUCTION

This project investigated the problem of how to efficiently determine global optimum value in a constrained non-linear system involving multiple local minima or maxima. Determination of global minima or maxima in a constrained non-linear system has important and wide application in engineering, physics, chemistry, biology and finance. In this project, I compared several general numerical methods for solving the above mentioned problem of global optimization. In particular, this project will focus on Generalized Simulated Annealing (GSA) through R package **GenSA**. [1] Simulated Annealing (SA) algorithm was originally inspired from the process of annealing in metal work. Annealing involves heating and cooling a material to alter its physical properties due to the changes in its internal structure. As the metal cools its new structure becomes fixed, consequently causing the metal to retain its newly obtained properties. [2]

In simulated annealing a temperature variable is kept to simulate this heating process. The temperature was initially set high and then allowed to slowly "cool" as the algorithm runs. When the temperature variable is high, the algorithm allows with high frequency to accept solutions that are worse than current solution. This gives the algorithm the ability to jump out of any local optimums it finds in early stage. As the temperature is reduced so is the chance of accepting worse solutions, this allows the algorithm to gradually focus on an area of the search space in which an approximation to optimum solution can be found. This gradual "cooling" process is what makes the simulated annealing algorithm remarkably effective at finding an approximation to optimum solution when dealing with large problems which contain numerous local optimums. [3]

II. METHODS AND MODELING APPROACH

Generalized Simulated Annealing was used to investigate the global minimization problem in a constrained nonlinear system. It was compared with the Differential Evolution (DE) algorithm, which is implemented using R package **DEoptim**. [4] Rastrigin function [5] and Thomson's problem [6] were used to demonstrate how to solve the minimization problem using the above mentioned algorithms, and to compare them in terms of accuracy, speed and efficiency.

Mathematics of Generalized Simulated Annealing

Generalized Simulated Annealing uses a distorted Cauchy-Lorentz visiting distribution $g_{q_v}(\Delta x(t))$ to calculate a trial jump distance $\Delta x(t)$ for each move of GSA algorithm. This visiting distribution is controlled by a shape parameter q_v as shown in Equation (1), [1, 2]

$$g_{q_v}(\Delta x(t)) \propto \frac{[T_{q_v}(t)]^{-\frac{D}{3-q_v}}}{[1+(q_v-1)\frac{(\Delta x(t))^2}{[T_{q_v}(t)]^{\frac{2}{3-q_v}}}]^{\frac{1}{q_v-1}+\frac{D-1}{2}}} \tag{1}$$

where *t* is artificial time, $T_{q_v}(t)$ is artificial temperature, and *D* is dimension of independent variable *x*. The trial jump is accepted if it is downhill of a given objective function. If the jump is uphill it might be accepted according to an acceptance probability. A generalized Metropolis algorithm is used for acceptance probability as shown in Equation (2),

$$p_{q_a} = \min\{1,[1-[1-q_a]\beta\Delta E]^{\frac{1}{1-q_a}}\} \tag{2}$$

where q_a, β and ΔE are artificial control, Langrange and energy spectrum parameters , respectively. The artificial temperature is decreased according to Equation (3).

$$T_{q_v}(t) = T_{q_v}(t_0)\frac{2^{q_v-1}}{(1+t)^{q_v-1}-1} \tag{3}$$

When q_v = 1 and q_a = 1, GSA reduces to Classical Simulated Annealing (CSA); [7] when q_v = 2 and q_a = 1, GSA reduces to Fast Simulated Annealing (FSA). [8] When q_v > 2, the cooling is faster than that of CSA and FSA. When $T \to 0$, GSA behaves like the steepest gradient descent algorithm. GSA not only converges faster than FSA and CSA, but also can escape from a local minimum more easily than FSA and CSA. Because GSA has a large probability to have a long jump, the probability of finding the global minimum with GSA is larger than those with FSA and CSA. [1, 2]

Install GenSA in R or RStudio

R is an open source statistical package for statistical analysis, optimization and simulation. RStudio provides a great interface to write, diagnose, visualized and execute R functions and scripts. Here are steps to setup R and RStudio and use GenSA.

1. Install the latest version of R (Version 3.2.4) at: https://www.r-project.org/ .
2. Install the latest version of RStudio at: https://www.rstudio.com/ .
3. Open RStudio and install libraries GenSA and DEoptim by issuing command:
   ```
   > install.packages("GenSA")
   > install.packages("DEoptim")
   ```

Examples of Application

Rastrigin function [5] and Thomson's problem [6] are classical examples of multimode, nonlinear and non-convex system. In the following two subsections, we demonstrate how to solve minimize these two systems using General Simulated Annealing and Differential Evolution algorithms, respectively.

Rastrigin's function

Rastrigin function is formulated as Equation (4),

$$f(X) = 10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)], \quad x_i \in [-5.12, 5.12] \tag{4}$$

where X is vector of independent variables, x_i is the i^{th} element, and d is the dimension of X. For two dimensional case, the minimization of Rastrigin function is simplified to Equation (5),

$$\min_{x_0, x_i \in [-5.12, 5.12]} [f(x_0, x_1)] = 20 + x_0^2 + x_1^2 - 10[\cos(2\pi x_0) + \cos(2\pi x_1)] \tag{5}$$

Thomson's problem

The objective of the Thomson problem is to determine the minimum electrostatic potential energy configuration of **N** electrons on the surface of a unit sphere that repel each other with a force given by Coulomb's law. Equation (6) presents the electrostatic interaction energy ($U_{ij}(N)$) occurring between electrons i and j with equal charges ($e_i = e_j = e$, with e the elementary charge of an electron) according to Coulomb's Law,

$$U_{ij}(N) = k_e \frac{e_i e_j}{r_{ij}^2} \tag{6}$$

where k_e is a constant, and r_{ij} is the distance between electrons i and j. The total electrostatic potential energy can then be formulated as Equation (7).

$$U(N) = \sum_{i<j} k_e \frac{e_i e_j}{r_{ij}^2} \tag{7}$$

Assume unit sphere (i.e., r_{ij} = 1) and unit elementary charge (i.e., e = 1), the Thomson's problem is equivalent to minimize the objective function (Equation (8)) under following constraints (Equation (9)) in spherical coordinates (Figure 1).

$$\min_{\vec{r}_i, \vec{r}_j} \{U(N) = \sum_{i<j} \frac{1}{|\vec{r}_i - \vec{r}_j|}\} \tag{8}$$

$$\text{Subject to: } \begin{cases} r_i = r = 1, \quad i = 1, 2, \ldots N \\ 0 \le \theta \le \pi \\ 0 \le \varphi \le 2\pi \end{cases} \tag{9}$$

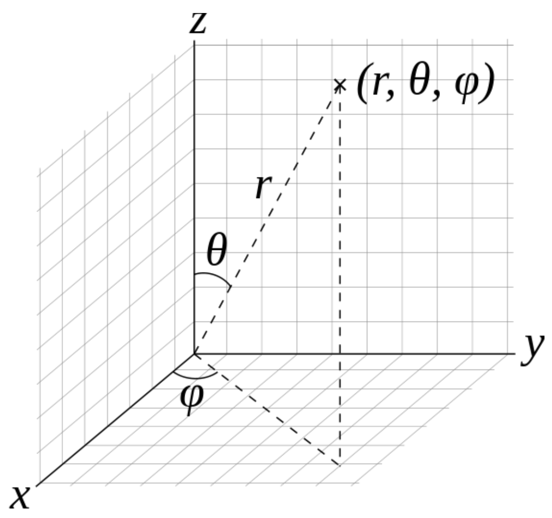

Figure 1 Spherical coordinate system.

The Cartesian coordinates (x, y, z) can be retrieved from spherical coordinates (r, θ, φ) through Equation (10).

$$\begin{cases} x = r \sin\theta \cos\varphi \\ y = r \sin\theta \sin\varphi \\ z = r \cos\theta \end{cases} \tag{10}$$

Table 1 summarizes the algorithms, R packages, application examples and analysis methods of this project.

Table 1 Configuration and implementation of Generalized Simulated Annealing and Differential Evolution.

Mathematic Algorithms	Generalized Simulated Annealing, Differential Evolution
R Packages	GenSA, DEoptim
Application Examples	Rastrigin Function, Thomson’s Problem
Performance Indices	Accuracy, Number of Successful Runs, Number of Function Calls

III. RESULTS AND DISCUSSION

The relevant R scripts and functions used in this project are documented Section VII APPENDICES. The results from these scripts and functions are summarized in the following two sections.

2D Rastrigin function

Figure 2 visualize Rastrigin function in its 2D form. The 2D Rastrigin function has many local minima. It is highly multimodal, but locations of the minima are regularly distributed. Because of these multimodal, non-convex and nonlinear features of 2D Rastrigin function, it is widely used as a performance test problem for optimization algorithms. Theoretically, the 2D Rastrigin function has the global minimum $f_{opt} = 0$ at $x_{opt} = (0,0)$.

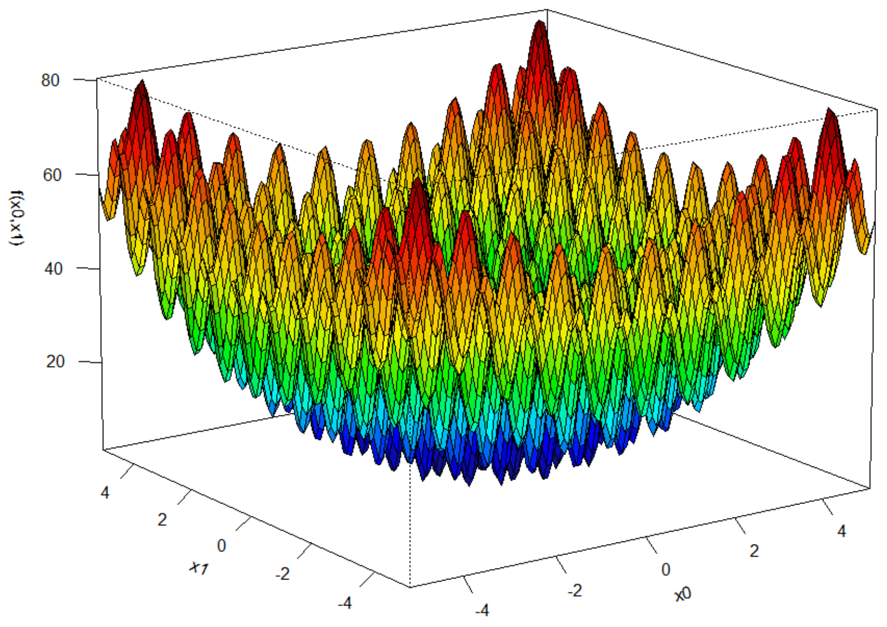

Figure 2 Visualization of 2D Rastrigin's function; red is higher and blue lower.

The best value that **DEoptim** found is **4.61×10^{-11}**, which is close to the global minimum **0**. The Differential Evolution algorithm called the function (Rastrigin.R) **4020** times to reach this approximation.

GenSA is also used to search for a minimum of the 2D Rastrigin function between the same lower and upper bounds as those for **DEoptim**. A random vector was specified as initial values for the parameters by setting "*par*" as "*NULL*" when calling **GenSA** (See **Appendices** for details). **GenSA** stopped search after it found the expected global minimum (i.e., 0) within the specified absolution tolerance (i.e., **1×10^{-13}**). It found this very close approximation (**0**) after **196** function calls. **GenSA** and **DEoptim** both converge to the global minimum but **GenSA** obtains a more accurate result with fewer function calls than **DEoptim**.

The Rastrigin function with 30 dimensions is almost the most difficult class of problems for many optimization algorithms. However **GenSA** can find the global minimum in a few seconds. **DEoptim** could not find the global minimum of Rastrigin function with 30 dimensions by using default settings. [1]

Number of successful runs

To compare the performance of GenSA and DEoptim, each of the methods was run 100 times randomly. The testing function was Rastrigin with 2 dimensions as shown in Figure 2. Only default values of parameters were used in both algorithms for comparison. Various absolute tolerances $\Delta \in \{1\times10^{-5}, 1\times10^{-7}, 1\times10^{-9}, 1\times10^{-11}\}$ were explored. The global minimum $f^{opt} = 0$ was considered to have been reached if the current function value is equal or less than $f^{opt} + \Delta$. A run was successful if the global minimum was reached. Figure 3 shows the percentage of the 100 runs that were successfully reaching the global minimum.

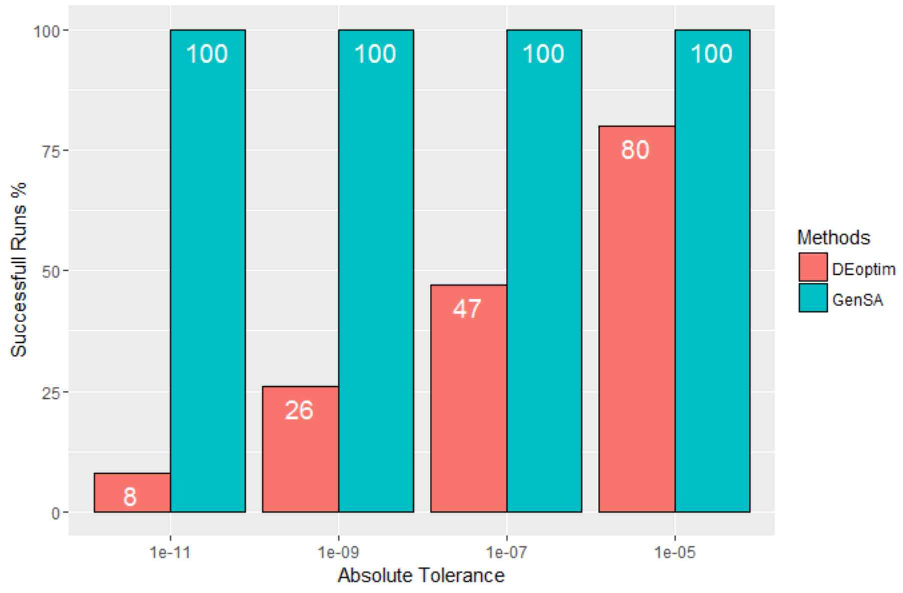

Figure 3 Percentage of successful runs at different absolute tolerances for GenSA (blue) and DEoptim (red) for solving the test 2D Rastrigin function.

The percentage of successful runs for DEoptim was 80% for $\Delta = 1\times10^{-5}$, but decreased to 8% for $\Delta = 1\times10^{-11}$. The percentage of successful runs for GenSA was 100% at all absolute tolerances. GenSA and DEoptim had a similar percentage of successful runs when the absolute tolerance was large, while GenSA was much more likely to be successful than DEoptim as the absolute tolerance was set to smaller values.

Average number of function calls

Figure 4 presents the average number of function calls to reach the global minimum for the successful runs. The error bar represents standard deviation of average number of function calls. The “***” (p-values ≤ 0.01, two sided t test) indicates the significant difference of average number of function calls between GenSA and DEoptim.

The average number of function calls for GenSA was around 500 at all tolerance values. The average number of function calls for DEoptim was less than 3000 for $\Delta = 1\times10^{-5}$, and increased significantly to around 4000 for $\Delta = 1\times10^{-11}$. A student t-test was performed to determine if the average number of function calls of GenSA was significantly smaller than the average number of function calls of DEoptim. It is shown in Figure 4 that the average number of function calls of GenSA is significantly smaller than the average number of function calls of DEoptim.

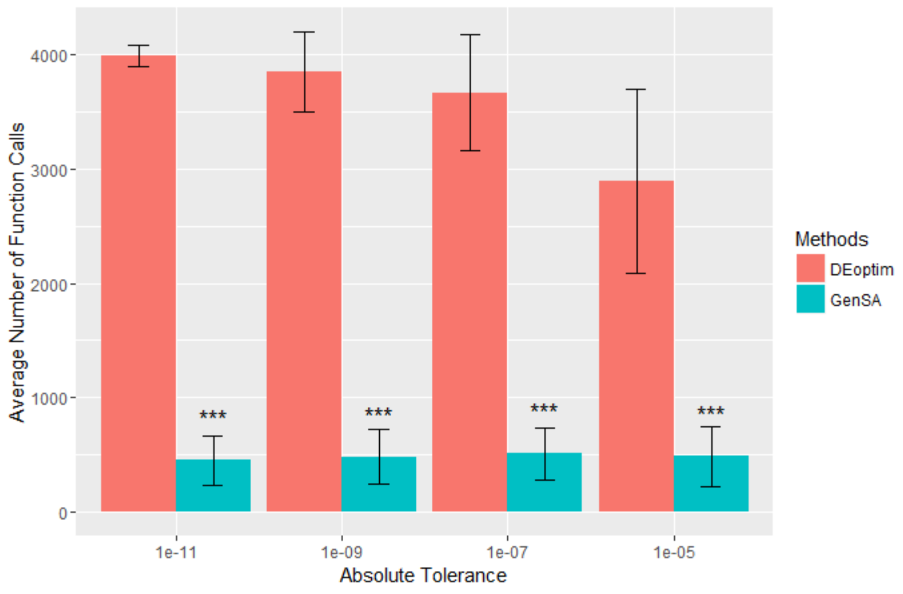

Figure 4 Average number of function calls to reach the global minimum in the successful runs at various absolute tolerances for GenSA (blue) and DEoptim (red) for solving 2D Rastrigin function. The error bar represents standard deviation of average number of function calls. The “*” (p-values ≤ 0.01, two sided t test) indicates the significant difference of average number of function calls between GenSA and DEoptim.**

GenSA and DEoptim for solving Thomson’s problem

GenSA achieved a more accurate solution than DEoptim for Thomson’s problem based on the same number of function calls and default parameters. The R function and scripts are listed in the attachment. Table 2 lists coordinates of an example solution obtained using GenSA based on default parameters. The solution identified by GenSA is almost same as the theoretical solution listed on Wikipedia at https://en.wikipedia.org/wiki/Thomson_problem.

Table 2 Coordinates of an example solutions of Thomson's problem (12 point charges) using GenSA. The approximate solution achieved using GenSA is 49.1653, which is same as the theoretical one listed at https://en.wikipedia.org/wiki/Thomson_problem .

θ	φ
1.5972527	0.3927183
1.5443398	3.5343115
0.5733629	2.8641549
2.0490932	4.5610180
1.6233707	2.4295881
2.5986850	3.1063163
0.5429074	6.2479098
1.5182219	5.5711806
2.5682300	6.0057477
1.0924997	1.4194253
0.9420908	4.5419090
2.1995017	1.4003160

Figure 5 depicts the cumulative minimum of function value versus number of function calls for GenSA and DEoptim for solving the Thomson problem. GenSA reaches the global minimum after few thousands function calls (around 5000) while DEoptim does not reach the global minimum after a much larger number of function calls (around 30000). This indicates that GenSA can solve the Thomson problem with 12 point charges more accurate and fast than DEoptim. For the Thomson problem, both DEoptim and GenSA converge to the global minimum. GenSA obtained more accurate results with fewer function calls than DEoptim.

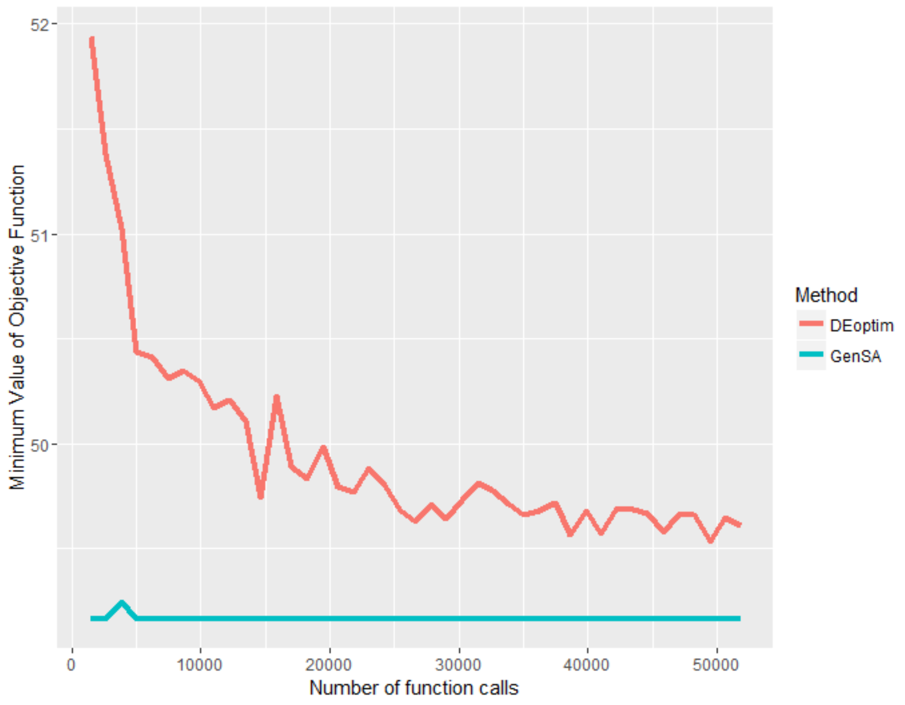

Figure 5 Cumulative minimum of function value vs. number of function calls for GenSA (blue) and DEoptim (red) in the Thomson problem (12 point charges). GenSA reaches the global minimum after few thousands function calls (around 5000) while DEoptim does not reach the global minimum after a much larger number of function calls (around 30000).

IV. CONCLUSIONS

This project investigated global optimization of non-linear system using Generalized Simulated Annealing algorithm. It was compared with the popular Differential Evolution algorithm for solving Rastrigin function and Thomson's problem in terms of accuracy, number of successful runs and average number of function calls. GenSA outperformed DEoptim in terms accuracy, efficiency and speed. GenSA is a great choice for global optimization of complicated nonlinear system with multiple minima/maxima.

V. RECOMMENDATIONS

GenSA algorithm depends on repeated evaluation of the objective function. Users interested in making GenSA run as fast as possible should ensure that evaluation of the objective function is also as efficient as possible. [1] Rstudio and R are great tools to implement Generalized Simulated Annealing and visualize corresponding results.

VI. REFERENCES

1. Xiang, Y.; Gubian, S.; Suomela, B.; Hoeng, J., Generalized simulated annealing for global optimization: the GenSA Package. *R Journal* **2013,** *5* (1), 13-28.
2. Tsallis, C.; Stariolo, D. A., Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications* **1996,** *233* (1), 395-406.
3. JACOBSON, L. Simulated Annealing for beginners. (accessed 04/11).
4. Ardia, D.; Boudt, K.; Carl, P.; Mullen, K. M.; Peterson, B. G., Differential evolution with deoptim. *R Journal* **2011,** *3* (1), 27-34.
5. Rastrigin, L. A., Systems of extremal control. *Zinatne, Riga* **1974**.
6. Ashby, N.; Brittin, W. E., Thomson's problem. *American Journal of Physics* **1986,** *54* (9), 776-777.
7. Kirkpatrick, S.; Vecchi, M. P., Optimization by simmulated annealing. *science* **1983,** *220* (4598), 671-680.
8. Szu, H.; Hartley, R., Fast simulated annealing. *Physics letters A* **1987,** *122* (3), 157-162.

VII. APPENDICES

Visualization of 2D Rastrigin function

```
# Fang Yu, 4/11/2016
# Adapated from https://jamesmccaffrey.wordpress.com/2015/10/07/graphing-rastrigins-function-in-3d-color-
gradient-using-r/
rm(list=ls()) # delete all objects
#set working directory
setwd("F:/ClassProject/Math/RScript")
#source("Rastrigin2D.R")
x0 <- seq(-5.12, 5.12, length=100)
x1 <- seq(-5.12, 5.12, length=100)

Rastrigin2D <- function(x0, x1) { 20 + (x0^2 - 10 *
                    cos(2 * 3.14 *x0)) + (x1^2 - 10 *
                                cos(2 * 3.14 *x1)) }
z <- outer(x0, x1, Rastrigin2D)
jet.colors <- colorRampPalette(c("midnightblue",
                    "blue", "cyan", "green", "yellow", "orange", "red", "darkred"))
nbcol <- 64
color <- jet.colors(nbcol)
nrz <- nrow(z)
ncz <- ncol(z)
zfacet <- z[-1,-1] + z[-1,-ncz] +
  z[-nrz,-1] + z[-nrz,-ncz]
facetcol <- cut(zfacet, nbcol)
persp(x0, x1, z, col=color[facetcol],
    phi=15, theta=-35, ticktype="detailed",
    d=10, r=1, shade=0.1, expand=0.7,zlab = "f(x0,x1)") Appendices
```

Rastrigin function

```
# Rastrigin's function
# Fang Yu, 4/11/2016
Rastrigin <- function(x) {
  fn.call <<- fn.call + 1
  sum(x^2 - 10 * cos(2 * pi * x)) + 10 * length(x)}
```

Use DEoptm to solve 2D Rastrigin function

```
# Fang Yu, 04/11/2016
# Adapted from Xiang Yang (2013)
 rm(list=ls()) # delete all objects
# load library DEoptim
 library("DEoptim")
#set working directory
 setwd("F:/ClassProject/Math/RScript")
 source("Rastrigin.R")
 options(digits = 10)
 dimension <- 2
 lower <- rep(-5.12, dimension)
 upper <- rep(5.12, dimension)
 set.seed(1234)
 sink("tmp.txt")
 fn.call <<- 0
 out.DEoptim <- DEoptim(fn = Rastrigin, lower = lower, upper = upper,
                      control = list(storepopfrom = 1))
 sink(NULL)

#Check output
 out.DEoptim$optim[c(1, 2)]
$bestmem
          par1           par2
-4.775211422e-07  6.390004611e-08
$bestval
[1] 4.60502747e-11
 fn.call
[1] 4020
```

Use GenSA to solve 2D Rastrigin function

```
# Fang Yu, 04/11/2016
# Adapted from Xiang Yang (2013)
# rm(list=ls()) # delete all objects
# load library DEoptim
library("GenSA")
#set working directory
setwd("F:/ClassProject/Math/RScript")
source("Rastrigin.R")
set.seed(1234)
expected.val <- 0
absTol <- 1e-13
fn.call <- 0
out.GenSA <- GenSA(par = NULL, lower = lower, upper = upper, fn = Rastrigin,
                  control = list(threshold.stop = expected.val + absTol))
#Check output
out.GenSA[c("value", "par", "counts")]
$value
[1] 0
$par
[1]  2.750668687e-12 -2.889218652e-12
$counts
[1] 196
cat("GenSA call functions", fn.call, "times.\n")
GenSA call functions 196 times.
```

Comparison between two methods for solving 2D Rastrigin function

```
# This script is used to compare GenSA and DEoptim for solving 2D Rastrigin function
# Fang Yu, 04/11/2016
rm(list=ls()) # delete all objects
# load library DEoptim
library("GenSA")
library("DEoptim")
library("ggplot2")
#set working directory
setwd("F:/ClassProject/Math/RScript")
```

```
source("Rastrigin.R")
set.seed(1234)
options(digits = 10)
dimension <- 2
NumRun <- 100
lower <- rep(-5.12, dimension)
upper <- rep(5.12, dimension)
#
expected.val <- 0
absTol <- c(1e-05, 1e-7, 1e-9, 1e-11)
NumSuccessRun <- matrix(0,nrow=2,ncol=length(absTol))
NumFnCallSA <- matrix(NaN,nrow=NumRun,ncol=length(absTol))
NumFnCallDE <- matrix(NaN,nrow=NumRun,ncol=length(absTol))
sink("tmp.txt") #direct intermidiate output into tmp.txt
for (i in 1:length(absTol)){
  for (j in 1:NumRun){
    #Initialize fn.call as 0 for GenSA
    fn.call <<- 0
    out.GenSA <- GenSA(par = NULL, lower = lower, upper = upper, fn = Rastrigin,
                   control = list(threshold.stop = expected.val + absTol[i]))
    if (out.GenSA["value"] <= (expected.val + absTol[i])) NumSuccessRun[1,i] = NumSuccessRun[1,i] + 1
    NumFnCallSA[j,i] <- fn.call
    #Initialize fn.call as 0 for DEoptim
    fn.call <<- 0
    out.DEoptim <- DEoptim(fn = Rastrigin, lower = lower, upper = upper,
                     control = list(VTR = expected.val + absTol[i]))
    if (out.DEoptim$optim["bestval"] <= (expected.val + absTol[i])) NumSuccessRun[2,i] = NumSuccessRun[2,i] + 1
    NumFnCallDE[j,i] <- fn.call
  }
}
sink(NULL) # show output in console
NumFnCallAveSA <- round(colMeans(NumFnCallSA))
NumFnCallSdSA <- round(apply(NumFnCallSA,2,sd))
NumFnCallAveDE <- round(colMeans(NumFnCallDE))
NumFnCallSdDE <- round(apply(NumFnCallDE,2,sd))
#Test significance in Number of function calls between GenSA and DEoptim
pvalue <- array(NaN,dim=c(1,length(absTol)))
for (i in 1:length(absTol)){
  ttestRes <- t.test(NumFnCallSA[,i],NumFnCallDE[,i])
  pvalue[i] <- ttestRes["p.value"]
}
#Visualize the results
#successfull run
DataSuccessRun <- data.frame(Methods = rep(c("GenSA","DEoptim"),each=length(absTol)),
                        absTol = as.factor(rep(absTol,2)), NumSuccessRun = as.vector(t(NumSuccessRun)))
ggplot(data=DataSuccessRun, aes(x=absTol, y=NumSuccessRun, fill=Methods)) +
  geom_bar(stat="identity", color="black",position=position_dodge()) +
  geom_text(aes(label=NumSuccessRun), vjust=1.6, color="white",
          position = position_dodge(0.9), size=5.0) +
  xlab("Absolute Tolerance") +
  ylab("Successfull Runs %")
  #theme_minimal()
#Average Number of function calls
```

```
DataFnCall <- data.frame(Methods = rep(c("GenSA","DEoptim"),each=length(absTol)),
                         absTol = as.factor(rep(absTol,2)), FnCallAve = c(NumFnCallAveSA,NumFnCallAveDE),
                         FnCallSd = c(NumFnCallSdSA,NumFnCallSdDE))
limits <- aes(ymax = FnCallAve + FnCallSd, ymin=FnCallAve - FnCallSd)
ggplot(DataFnCall, aes(fill=Methods, y=FnCallAve, x=absTol)) +
  geom_bar(position="dodge", stat="identity") +
  geom_errorbar(limits, position=dodge, width=0.25) +
  xlab("Absolute Tolerance") +
  ylab("Average Number of Function Calls") +
  geom_text(aes(y=FnCallAve, label=c("***","***","***","***","\t","\t","\t","\t")), vjust=-1.6, color="black",
        position = position_dodge(0.9), size=5.0)
```

Thomson's function

```
 #Function of Thomson's problem, Fang Yu, 03/07/2016
Thomson.fn <- function(ThetaPhi) {
  fn.call <<- fn.call + 1
  ThetaPhi <- matrix(ThetaPhi, ncol = 2)
  xyz <- t(apply(ThetaPhi, 1, function(thetaphi) {
    c(sin(thetaphi[1]) * cos(thetaphi[2]),
      sin(thetaphi[1]) * sin(thetaphi[2]), cos(thetaphi[1]))}))
  #print(xyz)
  n <- nrow(ThetaPhi)
  tmp <- matrix(NA, nrow = n, ncol = n)
  index <- cbind(as.vector(row(tmp)), as.vector(col(tmp)))
  index <- index[index[, 1] < index[, 2], , drop=F]
  rdist <- apply(index, 1, function(idx) {
    tmp <- 1/sqrt(sum((xyz[idx[1], ] - xyz[idx[2], ])^2))})
  res <- sum(rdist)
  return(res)
}
```

Comparison between two methods for solving Thomson's problem

```
# This script is used to compare GenSA and DEoptim for solving Thomson's problem
# Fang Yu, 04/11/2016
rm(list=ls()) # delete all objects
# load library DEoptim
library("GenSA")
library("DEoptim")
library("ggplot2")
#set working directory
setwd("F:/ClassProject/Math/RScript")
source("Thomson.R")
n.particles <- 12
lower.T <- rep(0, 2 * n.particles)
upper.T <- c(rep(pi, n.particles), rep(2 * pi, n.particles))
#set a random seed so that the results can be reproduced later on
set.seed(1234)
####Solve Thompson problem using GenSA
#Minimum Value
minValue <- out.GenSA["value"]
```

```
#The corresponding Theta and Phi which generated the minValue
out.GenSA <- GenSA(par = NULL, lower = lower.T, upper = upper.T, fn = Thomson.fn)
ThetaPhiFinal <- unlist(out.GenSA["par"])
ThetaFinal <- ThetaPhiFinal[1:n.particles]
PhiFinal <- ThetaPhiFinal[(1:n.particles)+n.particles]
data.frame(theta=as.numeric(ThetaFinal),phi= as.numeric(PhiFinal))
out.GenSA["value"]
####Compare with DEoptim (take long time to run)
itermax <- seq(5,250,by=5)
FminGenSA <- rep(NaN,length(itermax))
FminDEoptim <- rep(NaN,length(itermax))
FnCall <-rep(NaN,length(itermax))
sink("tmp.txt")
for (i in (1:length(itermax))){
  #set the number of function calls as 0 for DEoptim
  fn.call <<- 0
  out.DEoptim <- DEoptim(fn = Thomson.fn, lower = lower.T, upper = upper.T,
                       control = list(itermax=itermax[i]))
  FminDEoptim[i] <- as.numeric(out.DEoptim$optim["bestval"])
  FnCall[i] <- fn.call
  #set the number of function calls as 0 for GenSA
  fn.call <<- 0
  out.GenSA <- GenSA(par = NULL, lower = lower.T, upper = upper.T,
                   fn = Thomson.fn,control = list(max.call=FnCall[i]))
  FminGenSA[i] <- as.numeric(out.GenSA["value"])
}
save.image("ThomsonCompareData")
sink(NULL)
#Visualize the results
df <- data.frame(x=rep(FnCall,2), y=c(FminGenSA, FminDEoptim),
                Method=c(rep("GenSA", length(itermax)), rep("DEoptim", length(itermax))))
ggplot(df, aes(x=x, y=y, color=Method)) + geom_line(size=1.5) +
  xlab("Number of function calls") +
  ylab("Minimum Value of Objective Function")
```