gloptim – a wrapper for global optimization methods in R

Hans Werner Borchers & John C. Nash

2018-03-03

gloptim – a wrapper for global optimization methods in R

??JN: I don't think indexing works in Rmarkdown, but suggest we keep some sort of suggestion of the terms we want indexed.

Introduction and purpose

This vignette is a working document to record efforts to unify the calls to a number of global optimization methods available in packages for **R**. JN: I will use ?? to indicate issues where I need to resolve choices or want to clarify (perhaps in my own mind!)

There are very many methods offered for these problems, both generally and in **R**, and, sadly, many of the offerings are weak and/or simply renamings or very minor adjustments of other programs. This creates a challenge for the novice user. By "novice" we simply mean someone not well-versed in stochastic or global optimization, not a novice in **R** or their field of work. Moreover, it is common even among optimization workers that there are classes of methods which are unfamiliar.

This document will attempt

- to list available stochastic and global optimization methods in **R**
- to select those suitable for inclusion in a generic wrapper call, those that are unsuitable, and those worthy of further investigation
- to document the choices made, and the successes and failures. Where reasonable, the heavy detail will be moved to appendices, but an effort will be made to record all major attempts and failures.

Proposed call structure

This section proposes a call and its arguments.

The `base` package in **R** has the method `SANN` (for Simulated ANNealing) in the function `optim()`, so we will start with the call to that function, namely,

```
ans <- optim(start, function, gradient, method="Name", lower=lb,
        upper=ub, control=list())
```

Key issues:

```
- initiation methods or specification of the starting point or region

- termination criteria

- method controls and appropriate defaults

- agreed names for controls and arguments, since packages use many choices
```

Candidate packages and methods

This is a fairly long and messy section, but is necessary to allow for choices.

There are many names associated with global optimization methods: Simulated Annealing, Genetic Algorithms, Tabu Search, differential evolution, particle swarm optimization, Self-Organizing Migrating Algorithm.

?? JN: I'm happy to have more methods added, as well as to have comments on those to include, exclude, or maybe. I suggest marking them ?include?, ?exclude? and ?maybe? so we can search and find easily.

?? 170117 – replace Rastrigin with ?? as the "example" function. We need something that is quick to evaluate to avoid slow processing of this vignette which will have lots of examples. The purpose is to show the call and check the code works, not to test performance.

simulated annealing

SANN

method `SANN` in function `optim()` is a simulated annealing method that uses ONLY the iteration count as a termination criterion. We STRONGLY recommend against its use, but its presence in the **base** package leads to its use by novices. ?exclude?

GenSA

`GenSA` Generalized simulated annealing. Unlike the `optim()` method `SANN', the function`GenSA()‘ includes several controls for terminating the optimization process Xiang et al. (2013). ?include?

likelihood

`likelihood` uses simulated annealing as its optimization tool. Unfortunately, at the time of writing, this is particularized to the objective functions of the package, but the author (Lora Murphy) has shown interest in generalizing the optimizer, which is written in **R** Murphy (2012).

GrassmannOptim

`GrassmannOptim`: This package for Grassmann manifold optimization (used in image recognition) uses simulated annealing for attempting global optimization. The documentation (JN: I have not tried this package and mention it to show the range of possibilities) suggests the package can be used for general optimization Adragni, Cook, and Wu (2012). ?maybe?

Differential evolution

DEoptim

`DEoptim`: Global optimization by Differential Evolution. This package has many references, but here we will choose to point to Mullen et al. (2011). ?include?

RcppDE

`RcppDE`: Global optimization by differential evolution in C++, aimed at a more efficient interface to the `DEoptim` functionality Eddelbuettel (2012). ?include?

Genetic methods

rgenoud

`rgenoud`: R version of GENetic Optimization Using Derivatives Mebane, Jr. and Sekhon (2011). ?include?

GA

`GA`: Genetic Algorithms – a set of tools for maximizing a fitness function Scrucca (2013). A number of examples are included, including minimization of a 2-dimenstional Rastrigin function (we use a 5-dimensional example below). ?include?

gaoptim

`gaoptim`: Another suite of genetic algorithms. This one has functions to set up the process and create a function that is used to evolve the solution. However, I did not find that this package worked at all well for me, though I suspect I may have misunderstood the documentation. ?maybe?

rgp

`rgp` (an **R** genetic programming framework) likely has many tools that could be used for stochastic optimization, but the structure of the package makes it difficult to modify existing scripts to the task. The documentation of package `nsga2R` ?maybe?

nsga2R

`nsga2R` – Elitist Non-dominated Sorting Genetic Algorithm based on **R** – was, to my view, such that I could not figure out how to use it to minimize a function. ?maybe?

mcga

Package `mcga` (Machine coded genetic algorithms for real-valued optimization problems) appears to recode the parameters to 8-bit values and finds the minimum of the encoded function. However, it took me some time to decide that the "optimization" was a minimization. Many genetic and related codes seek maxima. ?maybe?

Other approaches

Rmalschains

`Rmalschains`: Continuous Optimization using Memetic Algorithms with Local Search Chains (MA-LS-Chains) in R Bergmeir, Molina, and Benítez (2012). I confess to knowing rather little about this approach. ?include?

smco

`smco`: A simple Monte Carlo optimizer using adaptive coordinate sampling Colombia (2012). ?include?

soma

`soma`: General-purpose optimisation with the Self-Organising Migrating Algorithm Ivan Zelinka (2011). Again, this approach is one with which I am not familiar. ?maybe?

pso [particle swarm optimization]

rCMA [CMA-ES]

An example problem

??JN: Do we want one or two more? Rastrigin is over-used. Replace??

To illustrate our attempted wrapper we will use a well-known multimodal test function called the Rastrigin function Törn and Zilinskas (1989), Mühlenbein, Schomisch, and Born (1991). This is defined in the examples for package `GenSA` as follows.

```
Rastrigin <- function(x) {
       sum(x^2 - 10 * cos(2 * pi  * x)) + 10 * length(x)
}
dimension <- 5
lower <- rep(-5.12, dimension)
upper <- rep(5.12, dimension)
start1 <- rep(1, dimension)
```

This function has as many dimensions as there are parameters in the vector `x` supplied to it. The traditional function is in 2 dimensions, but Gubain et al. in `GenSA` suggest that a problem in 30 dimensions is a particularly difficult one. Here we will use 5 dimensions as a compromise that allows for easier display of results while still providing some level of difficulty. Even the 2 dimensional case provides plenty of local minima, as can be seen from the Wikipedia article on the function at `http://en.wikipedia.org/wiki/Rastrigin_function`. Note that we define lower and upper bounds constraints on our parameters, and set a start vector.

??JN: I suggest we always show code to run the problem in the native method, then in our wrapper as a way to ensure we have got things more or less right.

Method `SANN` from `optim()`

```
      For ``SANN'' `maxit' gives the total number of function
      evaluations: there is no other stopping criterion. Defaults
      to `10000'.
```

Nonetheless, let us try this method on our 5-dimensional problem, using all 1s as a start. We will also fix a random number generator seed and reset this for each method to attempt to get some consistency in output. ?? JN: our wrapper is almost certain to need random number setting

```
myseed <- 1234
set.seed(myseed)
asann1 <- optim(par=start1, fn = Rastrigin, method='SANN', control=list(trace=1))
```

```
## sann objective function values
## initial       value 5.000000
## iter     1000 value 5.000000
## iter     2000 value 5.000000
## iter     3000 value 5.000000
## iter     4000 value 5.000000
## iter     5000 value 5.000000
## iter     6000 value 5.000000
## iter     7000 value 5.000000
## iter     8000 value 5.000000
## iter     9000 value 5.000000
## iter     9999 value 5.000000
## final         value 5.000000
## sann stopped after 9999 iterations
```

```
print(asann1)
```

```
## $par
## [1] 1 1 1 1 1
##
## $value
## [1] 5
##
## $counts
## function gradient
##    10000       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Sadly, not only do we make no progress, but the convergence code of 0 implies we have a "success", even though we have used the maximum number of function evaluations permitted. My opinion is that SANN should be dropped from the function `optim()`.

Package `GenSA`

Given that `GenSA` uses a 30-dimensional Rastrigin function as its example, we can anticipate a more successful outcome with this tool, and indeed this package does well on this example. Note that we supply only bounds constraints for this method, and no starting value.

```
suppressPackageStartupMessages(require(GenSA))
global.min <- 0
tol <- 1e-13
set.seed(myseed)
ctrl<-list(threshold.stop=global.min+tol, verbose=TRUE)
aGenSA1 <- GenSA(lower = lower, upper = upper, fn = Rastrigin,
           control=ctrl)
```

```
## Initializing par with random data inside bounds
## It: 1, obj value: 34.82336476
## It: 7, obj value: 9.949580495
## It: 38, obj value: 3.979836228
## It: 63, obj value: 1.989918114
## It: 66, obj value: 0.9949590571
## It: 68, obj value: 7.105427358e-15
```

```
print(aGenSA1[c("value","par","counts")])
```

```
## $value
## [1] 7.105427e-15
##
## $par
## [1] -1.785327e-12 -6.319116e-09 -6.498323e-12  2.238554e-13  2.866994e-14
##
## $counts
## [1] 1332
```

Packages DEoptim and RcppDE

}

The differential evolution approach has been considered by many workers to be one of the more successful tools for global optimization. As with GenSA, we supply only the bounds constraints. Our first try stops well-before a satisfactory solution has been found and a second try with a larger number of iterations is performed. It should, of course, be possible to use the work already done and re-commence the method. However, I have not discovered an obvious and easy way to do this. The same comment applies to several of the other packages discussed in this chapter.

```
suppressPackageStartupMessages(require(DEoptim))
set.seed(myseed)
ctrl <- list(trace=FALSE)
aDEopt1a <- DEoptim(lower = lower, upper = upper, fn = Rastrigin, control=ctrl)
print(aDEopt1a$optim)
```

```
## $bestmem
##          par1          par2          par3          par4          par5
##  0.0052265417 -0.0001110163 -0.0133039069 -0.0176931873 -0.0190440410
##
## $bestval
## [1] 0.1744246
##
## $nfeval
## [1] 402
##
## $iter
## [1] 200
```

```
tmp<-readline("Try DEoptim with more iterations")
```

```
## Try DEoptim with more iterations
```

```
set.seed(myseed)
ctrl <- list(itermax=10000, trace=FALSE)
aDEopt1b <- DEoptim(lower = lower, upper = upper, fn = Rastrigin, control=ctrl)
print(aDEopt1b$optim)
```

```
## $bestmem
##          par1          par2          par3          par4          par5
##  1.901555e-10  2.397707e-10 -2.150811e-09  5.292625e-10 -3.278135e-10
##
## $bestval
## [1] 0
##
## $nfeval
## [1] 20002
##
## $iter
## [1] 10000
```

RcppDE is possibly more efficient, though on the small example problem, I could not really detect much difference.

```
suppressPackageStartupMessages(require(RcppDE))
set.seed(myseed)
```

```
ctrl <- list(trace=FALSE)
aRcppDEopt1a <- DEoptim(lower = lower, upper = upper, fn = Rastrigin, control=ctrl)
print(aRcppDEopt1a$optim)
```

```
## $bestmem
##         par1         par2         par3         par4         par5
## -0.002466512  0.002419258  0.001525090  1.000797241  0.006213076
##
## $bestval
## [1] 1.012207
##
## $nfeval
## [1] 10050
##
## $iter
## [1] 200
```

```
tmp<-readline("Try RcppDE with more iterations")
```

```
## Try RcppDE with more iterations
```

```
set.seed(myseed)
ctrl <- list(itermax=10000, trace=FALSE)
aRcppDEopt1b <- DEoptim(lower = lower, upper = upper, fn = Rastrigin, control=ctrl)
print(aRcppDEopt1b$optim)
```

```
## $bestmem
##          par1          par2          par3          par4          par5
##  9.941200e-10 -2.072267e-10 -9.949586e-01 -1.257501e-09 -7.698191e-10
##
## $bestval
## [1] 0.9949591
##
## $nfeval
## [1] 500050
##
## $iter
## [1] 10000
```

Unfortunately, as simple change in the random number seed gives an unsatisfactory result, even with a lot of computational effort.

```
set.seed(123456)
ctrl <- list(itermax=10000, trace=FALSE)
aRcppDEopt1b <- DEoptim(lower = lower, upper = upper, fn = Rastrigin, control=ctrl)
print(aRcppDEopt1b$optim)
```

```
## $bestmem
##          par1          par2          par3          par4          par5
##  2.693449e-09  1.189177e-09 -8.563510e-11  2.384843e-10  9.420903e-10
##
## $bestval
## [1] 0
##
## $nfeval
## [1] 500050
```

```
##
## $iter
## [1] 10000
```

Package smco

This quite small package (all in **R**) uses an adaptive coordinate sampling, i.e., one parameter at a time approach to finding the minimum of a function. It appears to work quite well on the Rastrigin function. The choices of `LB` and `UB` instead of `lower` and `upper`, and `trc` instead of ‘trace} are yet another instance of minor but annoying differences in syntax, though the general structure and output of this package are reasonable.

```
suppressPackageStartupMessages(require(smco))
```

```
## This is smco package 0.1
```

```
set.seed(myseed)
asmco1 <- smco(par=rep(1,dimension), LB = lower, UB = upper, fn = Rastrigin,
                 maxiter=10000, trc=FALSE)
print(asmco1[c("par", "value")])
```

```
## $par
## [1]  1.137800e-04 -4.889919e-06 -1.537998e-04  9.906723e-07 -6.444124e-05
##
## $value
## [1] 8.089998e-06
```

Package soma

`soma`, which seems to perform reasonably well on the Rastrigin function, is a quite compact method written entirely in **R**. Unfortunately, the particular syntax of its input and output make this package somewhat awkward to control. In particular, the documentation does not indicate how to control the output from the method, which turns out to be in package `reportr` by the same developer. This took some digging to discover. Moreover, the names and structures of output quantities are quite different from those in, for example, `optim()`. `soma` is, however, just a particular example of the problematic diversity of structures and names used in global optimization packages for **R**.

```
suppressPackageStartupMessages(require(soma))
suppressPackageStartupMessages(require(reportr))
# NOTE: Above was not documented in soma!
setOutputLevel(OL$Warning)
set.seed(myseed)
mybounds=list(min=lower, max=upper)
myopts=list(nMigrations=100)
asoma1<-soma(Rastrigin, bounds=mybounds, options=myopts)
# print(asoma1) # Gives too much output -- not obvious to interpret.
print(asoma1$population[,asoma1$leader])
```

```
## [1] -3.267966e-08  1.683638e-07 -9.350006e-08 -8.333456e-08 -6.612692e-07
```

```
print(Rastrigin(asoma1$population[,asoma1$leader]))
```

```
## [1] 9.5703e-11
```

2017-1-23: Experiments with early gloptim() (gloptimj() at the time) showed that **soma** is NOT very good at finding the solution of the Hald function. Moreover, there appears to be no option or control that initializes the random numbers used to generate the “population” of trial points. However, running

```
  set.seed(54321)
```

before each run gives reproducible results.

Package Rmalschains

Quoting from the package DESCRIPTION "Memetic algorithms are hybridizations of genetic algorithms with local search methods." Thus this package is yet another member of a large family of global optimization methods. In contrast to some of the other packages considered here, the output of this package is truly minimal. Unless the 'trace} is turned on, it is not easy to determine the computational effort that has gone into obtaining a solution. Furthermore, interpreting the output from the trace is not obvious.

From the package documentation, the package implements a number of search strategies, and is coded in C++. It performs well on the Rastrigin test.

```
suppressPackageStartupMessages(require(Rmalschains))
set.seed(myseed)
amals<-malschains(Rastrigin, lower=lower, upper=upper,
         maxEvals=10000, seed=myseed)
```

```
## LS: CMAESHansen: cmaesmyrandom
## CMAES::Neighborhood: 0.5
## RatioLS: 0.500000
## Istep: 500
## LS::Effort: 0.500000
## EA::MaxEval: 10000
## Popsize: 50
## EA::PopFitness:   1.193367e+01  2.345648e+01  3.083115e+01  3.474920e+01
## EA::Improvement:  3.112728e+01  4.438530e+01  5.636207e+01  6.946668e+01
## EABest[16]:  4.306094e+01 -> 1.193367e+01  3.112728e+01
## LS [16]: 1.193367e+01 -> 3.980097e+00  7.953569e+00
## EA::PopFitness:   3.980097e+00  1.236908e+01  1.864367e+01  2.228649e+01
## EA::Improvement:  0.000000e+00  1.108741e+01  1.218748e+01  1.246271e+01
## LS [16]: 3.980097e+00 -> 3.979836e+00  2.607968e-04
## EA::PopFitness:   3.979836e+00  7.804856e+00  9.544590e+00  1.170864e+01
## EA::Improvement:  0.000000e+00  4.564219e+00  9.099078e+00  1.057785e+01
## LS [16]: 3.979836e+00 -> 3.979836e+00  9.614070e-10
## EA::PopFitness:   2.106710e+00  3.871350e+00  4.530694e+00  5.335466e+00
## EA::Improvement:  1.873126e+00  3.933506e+00  5.013896e+00  6.373171e+00
## EABest[23]:  3.979836e+00 -> 2.106710e+00  1.873126e+00
## LS [23]: 2.106710e+00 -> 9.949640e-01  1.111746e+00
## EA::PopFitness:   9.949640e-01  1.551440e+00  2.004469e+00  2.279536e+00
## EA::Improvement:  0.000000e+00  2.319910e+00  2.526225e+00  3.055930e+00
## LS [23]: 9.949640e-01 -> 9.949591e-01  4.925373e-06
## EA::PopFitness:   9.949591e-01  9.989434e-01  1.001093e+00  1.002316e+00
## EA::Improvement:  0.000000e+00  5.524962e-01  1.003376e+00  1.277220e+00
## LS [23]: 9.949591e-01 -> 9.949591e-01  1.143974e-12
## EA::PopFitness:   9.949591e-01  9.949842e-01  9.949966e-01  9.950148e-01
## EA::Improvement:  0.000000e+00  3.959155e-03  6.096943e-03  7.300808e-03
## EABest[23]:  9.949591e-01 -> 9.949591e-01  1.143974e-12
## LS [33]: 9.949659e-01 -> 9.949591e-01  6.866658e-06
## EA::PopFitness:   1.671099e-01  9.949592e-01  9.949594e-01  9.949595e-01
## EA::Improvement:  8.278491e-01  2.497897e-05  3.719985e-05  5.529747e-05
## EABest[ 5]:  9.949591e-01 -> 1.671099e-01  8.278491e-01
## LS [ 5]: 1.671099e-01 -> 6.142228e-07  1.671093e-01
## EA::PopFitness:   6.142228e-07  3.465312e-01  9.949591e-01  9.949591e-01
## EA::Improvement:  0.000000e+00  6.484280e-01  2.880154e-07  3.839867e-07
## LS [ 5]: 6.142228e-07 -> 3.595346e-12  6.142192e-07
## EA::PopFitness:   3.595346e-12  2.640586e-06  6.853584e-06  1.348460e-05
## EA::Improvement:  0.000000e+00  3.465286e-01  9.949522e-01  9.949456e-01
## LS [ 5]: 3.595346e-12 -> 0.000000e+00  3.595346e-12
## EA::PopFitness:   3.595346e-12  6.563070e-09  9.669577e-09  1.256773e-08
## EA::Improvement:  0.000000e+00  2.634023e-06  6.843914e-06  1.347203e-05
## EABest[ 5]:  0.000000e+00 -> 3.595346e-12  3.595346e-12
## RatioEffort Alg/LS: [55/45]
## RatioImprovement Alg/LS: [79/21]
## Restarts: 0
## Time[ALG]: 33.15
## Time[LS]: 22.52
## Time[MA]: 56.02
## RatioTime[ALG/MA]: 59.18
## RatioTime[LS/MA]: 40.19
## NumImprovement[EA]:45%
## NumImprovement[LS]:81%
## CMAES::Popsize/Lambda: 8
## CMAES::ParentsSize/Mu: 4
```

```r
print(amals)
```

```
## NumTotalEvalEA: 5462
## NumTotalEvalLS: 4528
## RatioEffort EA/LS: [55/45]
## RatioImprovement EA/LS: [79/21]
## PercentageNumImprovement[EA]: 45%
## PercentageNumImprovement[LS]: 82%
## Time[EA]: 33.15
## Time[LS]: 22.52
## Time[MA]: 56.02
## RatioTime[EA/MA]: 59.18
## RatioTime[LS/MA]: 40.19
## Fitness:
## [1] 3.595346e-12
## Solution:
## [1]  1.144283e-07 -5.452867e-08 -4.537327e-08  1.475018e-09  2.208477e-09
```

Package `rgenoud`

The `rgenoud` package is one of the more mature genetic algorithm tools for **R**. Like others, it has its own peculiarities, in particular, requiring neither a starting vector nor bounds. Instead, it is assumed that the function first argument is a vector, and the `genoud} function simply wants to know the dimension of this via a parameternvars}`. This package is generally set up for maximization, so one must specify `max=FALSE} to perform a minimization.`print.level} is used to control output.

```r
suppressPackageStartupMessages(require(rgenoud))
set.seed(myseed)
agen1<-genoud(Rastrigin, nvars=dimension, max=FALSE, print.level=0)
print(agen1)
```

```
## $value
## [1] 0
##
## $par
## [1] -2.814295e-10  9.779426e-10 -1.215418e-11  3.553025e-10 -3.141589e-11
##
## $gradients
## [1] -1.121113e-07  3.871233e-07 -4.909099e-09  1.414001e-07 -1.384012e-08
##
## $generations
## [1] 18
##
## $peakgeneration
## [1] 7
##
## $popsize
## [1] 1000
##
## $operators
## [1] 122 125 125 125 125 126 125 126   0
```

Package GA

This package has many options. Here we will use only a simple setup, which may be less than ideal. The package is able to use parallel processing, for example. Because this package uses vocabulary and syntax very different from that in the `optimx` (or `optimr`) family, I have not been drawn to use it except in simple examples to see that it did work, more or less.

```r
suppressPackageStartupMessages(require(GA))
set.seed(myseed)
aGA1<-ga(type = "real-valued", fitness = function(x) -Rastrigin(x),
           min = lower, max = upper, popSize = 50, maxiter = 1000,
           monitor=NULL)
print(summary(aGA1))
```

```
## +-----------------------------------+
## |         Genetic Algorithm         |
## +-----------------------------------+
##
## GA settings:
## Type                  =  real-valued
## Population size       =  50
## Number of generations =  1000
## Elitism               =  2
## Crossover probability =  0.8
## Mutation probability  =  0.1
## Search domain =
##        x1    x2    x3    x4    x5
## Min -5.12 -5.12 -5.12 -5.12 -5.12
## Max  5.12  5.12  5.12  5.12  5.12
##
## GA results:
## Iterations             = 1000
## Fitness function value = -1.602816e-05
## Solution =
##               x1            x2            x3            x4            x5
## [1,] 0.0001308884 -7.691151e-06 -0.0001507232 -0.0001859094 -7.949612e-05
```

Package gaoptim

Until I communicated with the maintainer of `gaoptim` Tenorio (2013), I had some difficulty in using it, and from the exchange of messages, I suspect the documentation will be clarified. It appears that it is necessary with this package to specify a `selection` method that guides the choice of which trial results are used to drive the algorithm. The simplest choice is `"uniform"` in the emulation of the mating process used to evolve the population of chosen parameter sets.

```
suppressPackageStartupMessages(require(gaoptim))
set.seed(myseed)
minRast<-function(x) { -Rastrigin(x) } # define for minimizing
## Initial calling syntax -- no selection argument
## agaor1<-GAReal(minRast, lb=lower, ub=upper)
agaor1<-GAReal(minRast, lb=lower, ub=upper, selection='uniform')
agaor1$evolve(200) # iterate for 200 generations
## The same result was returned from 400 generations
agaor1
```

```
## Results for 200 Generations:
## Mean Fitness:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -88.807  -5.364  -3.134  -8.002  -2.427  -1.418
##
## Best Fitness:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -43.75443  -1.43306  -0.54197  -1.69044  -0.00454  -0.00454
##
## Best individual:
## [1]  1.103194e-06 -4.531104e-03 -4.489247e-04  1.475300e-03  5.031733e-06
##
## Best fitness value:
## [1] -0.004544682
```

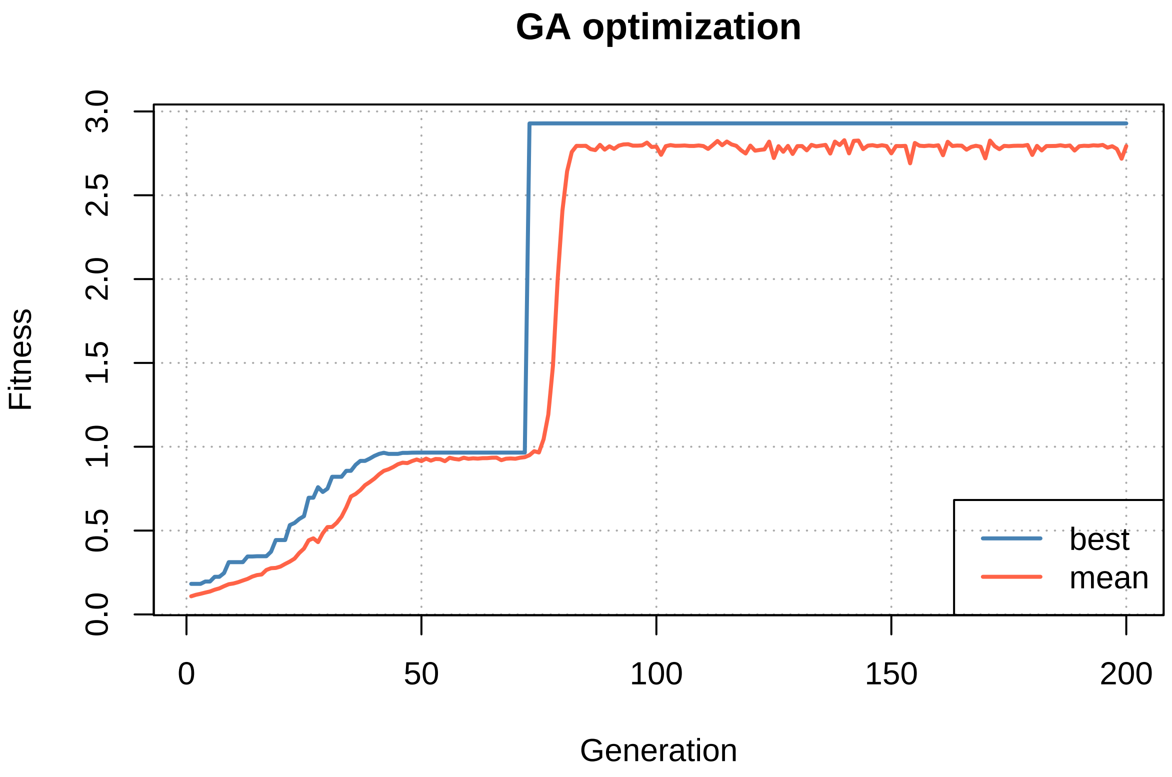
We can also select parameter sets based on the `fitness' function, that is, our function being maximized. Unfortunately, interpreting this as a probability is not a good idea for a function we have defined as negative -- we are, after all, maximizing this negative function to get near zero for the original function. Rather than have to learn how to write a custom selector function, I have chosen to simply use the option`selector='fitness'}, which uses the objective function values as a proxy for the fitness, along with a redefined objective which is positive. We want larger values as 'better'. We take the sqrt() to better scale the graph that displays progress.

```
maxRast<-function(x){ sqrt(1/ abs(Rastrigin(x))) }
agaor2<-GAReal(maxRast, lb=lower, ub=upper, selection='fitness')
agaor2$evolve(200)
agaor2
```

```
## Results for 200 Generations:
## Mean Fitness:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1081  0.9166  2.7690  1.9473  2.7949  2.8284
##
## Best Fitness:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1822  0.9652  2.9286  2.1334  2.9286  2.9286
##
## Best individual:
```

```
## [1]  1.420450e-02 -1.889660e-02  5.420473e-03 -9.671556e-05 -1.381934e-06
##
## Best fitness value:
## [1] 2.928631
```

```
plot(agaor2) # note this special method for displaying results
```



```
Rastrigin(agaor2$bestIndividual())
```

```
## [1] 0.1165925
```

Note that, as with most global methods, there are long stretches of work where we do not see any progress.

Multiple starting values

?? JN: Do we want to even consider such an approach?

In many cases, it is possible to simply use a selection of starting vectors with a **local** minimizer to achieve desired results. In the case of the Rastrigin function, this does not appear to be the case. Let us use 'Rvmmin} with the bounds specified above and generate 500 random starts uniformly distributed across each coordinate. Note we do not do very well.

```
suppressPackageStartupMessages(require(Rvmmin))
nrep<-500
bestval<-.Machine$double.xmax # start with a big number
bestpar<-rep(NA, dimension)
startmat<-matrix(NA, nrow=nrep, ncol=dimension)
set.seed(myseed)
for (i in 1:nrep) {
    for (j in 1:dimension){
        startmat[i,j]<-lower[j]+(upper[j]-lower[j])*runif(1)
    }
}
for (i in 1:nrep){
   tstart<-as.vector(startmat[i,])
    ans<-Rvmmin(tstart, Rastrigin, lower=lower, upper=upper)
   if (ans$value <= bestval) {
      bestval<-ans$value
      cat("Start ",i," new best =",bestval,"\n")
      bestpar<-ans$par
      bestans<-ans
   }
}
```

```
## Start  1  new best = 34.82336
## Start  3  new best = 14.92435
## Start  7  new best = 9.949581
## Start  19  new best = 3.979831
```

```
## Warning in Rvmminb(par, fn, gr, lower = lower, upper = upper, bdmsk =
## bdmsk, : Too many function evaluations
```

```
## Start  434  new best = 2.984877
## Start  458  new best = 1.989918
```

```
## Warning in Rvmminb(par, fn, gr, lower = lower, upper = upper, bdmsk =
## bdmsk, : Too many function evaluations
```

```
print(bestans)
```

```
## $par
## [1] -7.852222e-07  9.949586e-01 -6.155801e-07 -5.282705e-06 -9.949586e-01
##
## $value
## [1] 1.989918
##
## $counts
## function gradient
##       42       17
##
## $convergence
## [1] 2
##
## $message
## [1] "Rvmminb appears to have converged"
##
## $bdmsk
## [1] 1 1 1 1 1
```

On the other hand, we don't have to try very hard to find what turns out to be a global minimum of Branin's function Molga and Smutnicki (2005)}, though it is a bit more difficult to find that there are several such minima at different points in the space. Here is the function and its gradient.

```
## branmin.R
myseed<-123456L # The user can use any seed. BUT ... some don't work so well.
branin<-function(x){ ## Branin's test function
     if (length(x) != 2) stop("Wrong dimensionality of parameter vector")
     x1<-x[1] # limited to [-5, 10]
     x2<-x[2] # limited to [0, 15]
     a <- 1
     b <- 5.1 / (4 * pi * pi)
     c <- 5 / pi
     d <- 6
     e <- 10
     f <- 1/(8*pi)
     ## Global optima of 0.397887
     ## at (-pi, 12.275), (pi, 2.275), (9.42478, 2.475)
     val <- a*(x2 - b*(x1^2) + c*x1 - d)^2 + e*(1 - f)*cos(x1) + e
}
branin.g<-function(x){ ## Branin's test function
     if (length(x) != 2) stop("Wrong dimensionality of parameter vector")
     x1<-x[1]
     x2<-x[2]
     a <- 1
     b <- 5.1 / (4 * pi * pi)
     c <- 5 / pi
     d <- 6
     e <- 10
     f <- 1/(8*pi)
     ## Global optima of 0.397887
     ## at (-pi, 12.275), (pi, 2.275), (9.42478, 2.475)
#     val <- a*(x2 - b*(x1^2) + c*x1 - d)^2 + e*(1 - f)*cos(x1) + e
     g1 <- 2*a*(x2 - b*(x1^2) + c*x1 - d)*(-2*b*x1+c)-e*(1-f)*sin(x1)
     g2 <- 2*a*(x2 - b*(x1^2) + c*x1 - d)
     gg<-c(g1, g2)
}
```

In the following, we actually use external knowledge of the three global minima and look at the termination points from different starting points for 'Rvmmin()}. We create a grid of starting points between -5 and 10 for the first parameter and 0 and 15 for the second parameter. We will use 40 points on each axis, for a total of 1600 sets of starting parameters. We can put these on a 40 by 40 character array, and assign 0 to any result which is not one of the three known minima, and the labels 1, 2 or 3 are assigned to the appropriate array position of the starting parameters for whichever of the minima are found successfully.

There are very few starts that actually fail to find one of the global minima of this function, and most of those are on a bound. As indicated, the program code labels the three optima as 1, 2 or 3, and puts a 0 on any other solution.

To find out which solution has been generated, if any, let us compute the Euclidean distance between our solution and one of the three known minima, which are labelled `y1},y2}` and 'y3}. If the distance is less than 1e-6 from the relevant minima, we assign the appropriate code, otherwise it is left at 0.

```
dist2<-function(va, vb){
    n1<-length(va)
    n2<-length(vb)
    if (n1 != n2) stop("Mismatched vectors")
    dd<-0
    for (i in 1:n1){ dd<-dd+(va[i]-vb[i])^2 }
    dd
}
y1 <- c(-pi, 12.275)
y2 <- c(pi, 2.275)
y3 <- c(9.42478, 2.475)
```

```r
lo<-c(-5, 0)
up<-c(10, 15)
npt<-40
grid1<-((1:npt)-1)*(up[1]-lo[1])/(npt-1)+lo[1]
grid2<-((1:npt)-1)*(up[2]-lo[2])/(npt-1)+lo[2]
pnts<-expand.grid(grid1,grid2)
names(pnts)=c("x1", "x2")
nrun<-dim(pnts)[1]
```

We will not display the code to do all the tedious work.

```r
suppressPackageStartupMessages(require(Rvmmin))
reslt<-matrix(NA, nrow=nrun, ncol=7)
names(reslt)<-c("vmin","bx1","bx2", "sx1", "sx2", "sval")
ctrl<-list(maxit=20000)

cmat<-matrix(" ",nrow=npt, ncol=npt)
kk<-0 # for possible use as a progress counter
for (ii1 in 1:npt) { # row index is ii1 for x2 from grid2, plotted at ii=npt+1-ii1
    for (jj in 1:npt) { # col index is for x1 on graph
       ii<-npt+1-ii1
       strt<-c(grid1[jj], grid2[ii1])
       sval<-branin(strt)
       ans<-suppressWarnings(
            Rvmmin(strt, branin, branin.g, lower=lo, upper=up, control=ctrl)
       )
       kk<-kk+1
       reslt[kk,4]<-strt[1]
       reslt[kk,5]<-strt[2]
       reslt[kk,6]<-sval
       reslt[kk,1]<-ans$value
       reslt[kk,2]<-ans$par[1]
       reslt[kk,3]<-ans$par[2]
       reslt[kk,7]<-0
       if (ans$convergence !=0) {
##     cat("Nonconvergence from start c(",strt[1],", ", strt[2],")\n")
           reslt[kk,1]<- .Machine$double.xmax
       } else {
          if (dist2(ans$par, y1) < 1e-6) reslt[kk,7]=1
          if (dist2(ans$par, y2) < 1e-6) reslt[kk,7]=2
          if (dist2(ans$par, y3) < 1e-6) reslt[kk,7]=3
       }
       cmat[ii, jj]<-as.character(reslt[kk,7])
   } # end jj
} # end ii1
mpts<-kk
cat("Non convergence from the following starts:\n")
```

```
## Non convergence from the following starts:
```

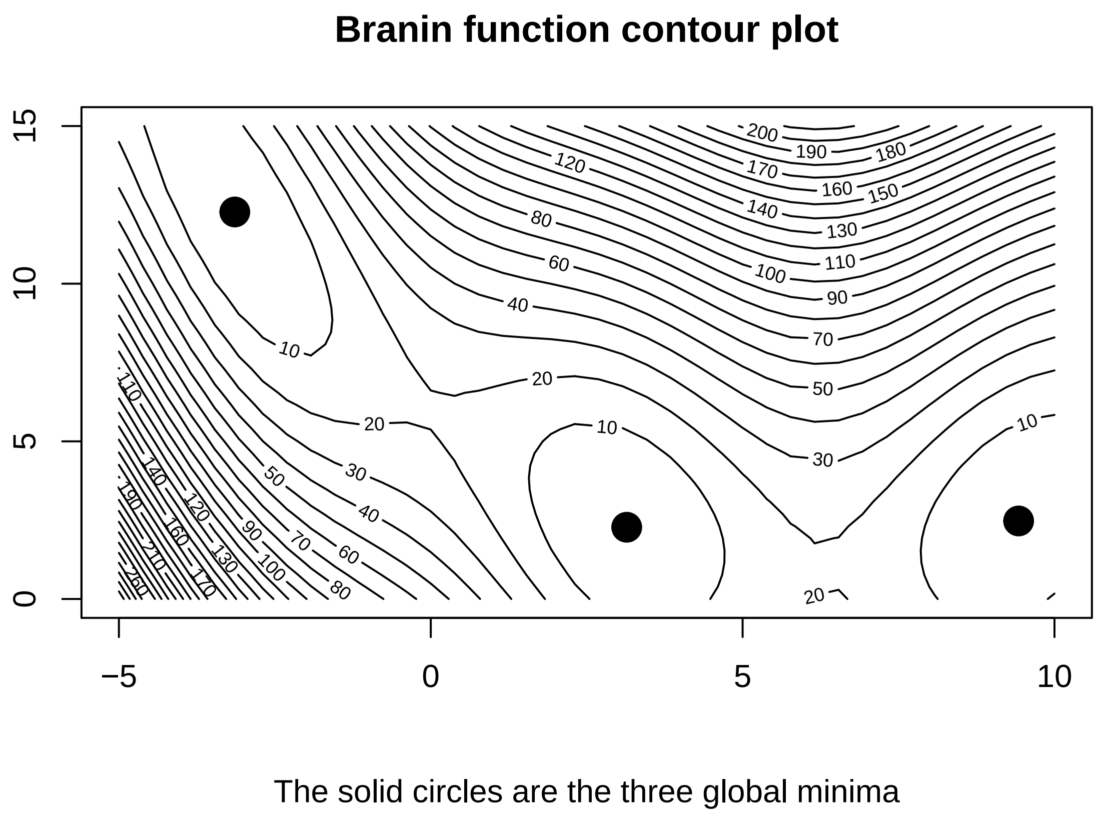
```r
linetrip<-0
for (kk in 1:mpts){
   if (reslt[kk,7]==0) {
   cat("(",reslt[kk,4],", ",reslt[kk,5],")  ")
   linetrip<-linetrip+1
```

```r
  if (4*floor(linetrip/4)==linetrip) cat("\n")
  }
}
```

```
## ( -5 ,  0 )  ( -4.615385 ,  0 )  ( -4.230769 ,  0 )  ( -3.846154 ,  0 )
## ( -3.461538 ,  0 )  ( -3.076923 ,  0 )  ( -2.692308 ,  0 )  ( -2.307692 ,  0 )
## ( -1.923077 ,  0 )  ( -1.538462 ,  0 )  ( -1.153846 ,  0 )  ( -0.7692308 ,  0 )
## ( -0.3846154 ,  0 )  ( 0 ,  0 )  ( 0.3846154 ,  0 )  ( 0.7692308 ,  0 )
## ( 1.153846 ,  0 )  ( 1.538462 ,  0 )  ( 1.923077 ,  0 )  ( 2.307692 ,  0 )
## ( 2.692308 ,  0 )  ( 6.153846 ,  0 )  ( 6.538462 ,  0 )  ( 6.923077 ,  0 )
## ( 7.307692 ,  0 )  ( 7.692308 ,  0 )  ( 8.076923 ,  0 )  ( 8.461538 ,  0 )
## ( -5 ,  0.3846154 )  ( -4.615385 ,  0.3846154 )  ( -4.230769 ,  0.3846154 )  ( -3.846154 ,  0.384615
## ( -3.461538 ,  0.3846154 )  ( -3.076923 ,  0.3846154 )  ( -2.692308 ,  0.3846154 )  ( -2.307692 ,  0
## ( -1.923077 ,  0.3846154 )  ( -1.538462 ,  0.3846154 )  ( -1.153846 ,  0.3846154 )  ( -0.7692308 ,
## ( -0.3846154 ,  0.3846154 )  ( 0 ,  0.3846154 )  ( 0.3846154 ,  0.3846154 )  ( 0.7692308 ,  0.384615
## ( 1.153846 ,  0.3846154 )  ( 1.538462 ,  0.3846154 )  ( 1.923077 ,  0.3846154 )  ( 2.307692 ,  0.384
## ( 2.692308 ,  0.3846154 )  ( 5.769231 ,  0.3846154 )  ( 6.153846 ,  0.3846154 )  ( 6.538462 ,  0.384
## ( 6.923077 ,  0.3846154 )  ( 7.307692 ,  0.3846154 )  ( 7.692308 ,  0.3846154 )  ( 8.076923 ,  0.384
## ( 8.461538 ,  0.3846154 )  ( 8.846154 ,  0.3846154 )  ( -5 ,  0.7692308 )  ( -4.615385 ,  0.7692308
## ( -4.230769 ,  0.7692308 )  ( -3.846154 ,  0.7692308 )  ( -3.461538 ,  0.7692308 )  ( -3.076923 ,  0
## ( -2.692308 ,  0.7692308 )  ( -2.307692 ,  0.7692308 )  ( -1.923077 ,  0.7692308 )  ( -1.538462 ,  0
## ( -1.153846 ,  0.7692308 )  ( -0.7692308 ,  0.7692308 )  ( -0.3846154 ,  0.7692308 )  ( 0 ,  0.76923
## ( 0.3846154 ,  0.7692308 )  ( 0.7692308 ,  0.7692308 )  ( 1.153846 ,  0.7692308 )  ( 1.538462 ,  0.7
## ( 1.923077 ,  0.7692308 )  ( 2.307692 ,  0.7692308 )  ( 2.692308 ,  0.7692308 )  ( 5.769231 ,  0.769
## ( 6.153846 ,  0.7692308 )  ( 6.538462 ,  0.7692308 )  ( 6.923077 ,  0.7692308 )  ( 7.307692 ,  0.769
## ( 7.692308 ,  0.7692308 )  ( 8.076923 ,  0.7692308 )  ( 8.461538 ,  0.7692308 )  ( -5 ,  1.153846 )
## ( -4.615385 ,  1.153846 )  ( -4.230769 ,  1.153846 )  ( -3.846154 ,  1.153846 )  ( -3.461538 ,  1.15
## ( -3.076923 ,  1.153846 )  ( -2.692308 ,  1.153846 )  ( -2.307692 ,  1.153846 )  ( -1.923077 ,  1.15
## ( -1.538462 ,  1.153846 )  ( -1.153846 ,  1.153846 )  ( -0.7692308 ,  1.153846 )  ( -0.3846154 ,  1.
## ( 0 ,  1.153846 )  ( 0.3846154 ,  1.153846 )  ( 0.7692308 ,  1.153846 )  ( 1.153846 ,  1.153846 )
## ( 1.538462 ,  1.153846 )  ( 1.923077 ,  1.153846 )  ( 2.307692 ,  1.153846 )  ( 5.769231 ,  1.153846
## ( 6.153846 ,  1.153846 )  ( 6.538462 ,  1.153846 )  ( 6.923077 ,  1.153846 )  ( 7.307692 ,  1.153846
## ( 7.692308 ,  1.153846 )  ( 8.076923 ,  1.153846 )  ( 8.461538 ,  1.153846 )  ( -5 ,  1.538462 )
## ( -4.615385 ,  1.538462 )  ( -4.230769 ,  1.538462 )  ( -3.846154 ,  1.538462 )  ( -3.461538 ,  1.53
## ( -3.076923 ,  1.538462 )  ( -2.692308 ,  1.538462 )  ( -2.307692 ,  1.538462 )  ( -1.923077 ,  1.53
## ( -1.538462 ,  1.538462 )  ( -0.3846154 ,  1.538462 )  ( 0 ,  1.538462 )  ( 0.3846154 ,  1.538462 )
## ( 0.7692308 ,  1.538462 )  ( 1.153846 ,  1.538462 )  ( 1.538462 ,  1.538462 )  ( 1.923077 ,  1.53846
## ( 2.307692 ,  1.538462 )  ( 5.769231 ,  1.538462 )  ( 6.153846 ,  1.538462 )  ( 6.923077 ,  1.538462
## ( 7.307692 ,  1.538462 )  ( 7.692308 ,  1.538462 )  ( 8.076923 ,  1.538462 )  ( 8.461538 ,  1.538462
## ( -5 ,  1.923077 )  ( -4.615385 ,  1.923077 )  ( -4.230769 ,  1.923077 )  ( -3.846154 ,  1.923077 )
## ( -3.461538 ,  1.923077 )  ( -3.076923 ,  1.923077 )  ( -2.692308 ,  1.923077 )  ( -2.307692 ,  1.92
## ( -1.923077 ,  1.923077 )  ( 0 ,  1.923077 )  ( 0.3846154 ,  1.923077 )  ( 0.7692308 ,  1.923077 )
## ( 1.153846 ,  1.923077 )  ( 1.538462 ,  1.923077 )  ( 1.923077 ,  1.923077 )  ( 2.307692 ,  1.923077
## ( 5.769231 ,  1.923077 )  ( 6.153846 ,  1.923077 )  ( 6.538462 ,  1.923077 )  ( 6.923077 ,  1.923077
## ( 7.307692 ,  1.923077 )  ( 7.692308 ,  1.923077 )  ( 8.076923 ,  1.923077 )  ( 8.461538 ,  1.923077
## ( -5 ,  2.307692 )  ( -4.615385 ,  2.307692 )  ( -4.230769 ,  2.307692 )  ( -3.846154 ,  2.307692 )
## ( -3.461538 ,  2.307692 )  ( -3.076923 ,  2.307692 )  ( -2.692308 ,  2.307692 )  ( -2.307692 ,  2.30
## ( -1.923077 ,  2.307692 )  ( -1.538462 ,  2.307692 )  ( -0.7692308 ,  2.307692 )  ( -0.3846154 ,  2.
## ( 0 ,  2.307692 )  ( 0.3846154 ,  2.307692 )  ( 0.7692308 ,  2.307692 )  ( 1.153846 ,  2.307692 )
## ( 1.538462 ,  2.307692 )  ( 1.923077 ,  2.307692 )  ( 2.307692 ,  2.307692 )  ( 5.769231 ,  2.307692
## ( 6.153846 ,  2.307692 )  ( 6.538462 ,  2.307692 )  ( 6.923077 ,  2.307692 )  ( 7.307692 ,  2.307692
## ( 7.692308 ,  2.307692 )  ( 8.076923 ,  2.307692 )  ( 8.461538 ,  2.307692 )  ( -5 ,  2.692308 )
## ( -4.615385 ,  2.692308 )  ( -4.230769 ,  2.692308 )  ( -3.846154 ,  2.692308 )  ( -3.461538 ,  2.69
## ( -3.076923 ,  2.692308 )  ( -2.692308 ,  2.692308 )  ( -2.307692 ,  2.692308 )  ( -1.538462 ,  2.69
## ( -1.153846 ,  2.692308 )  ( -0.7692308 ,  2.692308 )  ( -0.3846154 ,  2.692308 )  ( 0.3846154 ,  2.6
## ( 0.7692308 ,  2.692308 )  ( 1.153846 ,  2.692308 )  ( 1.538462 ,  2.692308 )  ( 1.923077 ,  2.69230
## ( 2.307692 ,  2.692308 )  ( 5.769231 ,  2.692308 )  ( 6.153846 ,  2.692308 )  ( 6.538462 ,  2.692308
## ( 6.923077 ,  2.692308 )  ( 7.307692 ,  2.692308 )  ( 7.692308 ,  2.692308 )  ( 8.076923 ,  2.692308
## ( 8.461538 ,  2.692308 )  ( 8.846154 ,  2.692308 )  ( -5 ,  3.076923 )  ( -4.615385 ,  3.076923 )
## ( -4.230769 ,  3.076923 )  ( -3.846154 ,  3.076923 )  ( -3.461538 ,  3.076923 )  ( -3.076923 ,  3.07
## ( -2.692308 ,  3.076923 )  ( -2.307692 ,  3.076923 )  ( -1.538462 ,  3.076923 )  ( -1.153846 ,  3.07
## ( -0.7692308 ,  3.076923 )  ( 0 ,  3.076923 )  ( 0.3846154 ,  3.076923 )  ( 0.7692308 ,  3.076923 )
## ( 1.153846 ,  3.076923 )  ( 1.538462 ,  3.076923 )  ( 1.923077 ,  3.076923 )  ( 5.769231 ,  3.076923
## ( 6.153846 ,  3.076923 )  ( 6.538462 ,  3.076923 )  ( 6.923077 ,  3.076923 )  ( 7.307692 ,  3.076923
## ( 7.692308 ,  3.076923 )  ( 8.076923 ,  3.076923 )  ( 8.461538 ,  3.076923 )  ( 8.846154 ,  3.076923
## ( -5 ,  3.461538 )  ( -4.615385 ,  3.461538 )  ( -4.230769 ,  3.461538 )  ( -3.846154 ,  3.461538 )
## ( -3.461538 ,  3.461538 )  ( -3.076923 ,  3.461538 )  ( -2.692308 ,  3.461538 )  ( -1.153846 ,  3.46
## ( 0 ,  3.461538 )  ( 0.7692308 ,  3.461538 )  ( 1.153846 ,  3.461538 )  ( 1.538462 ,  3.461538 )
## ( 1.923077 ,  3.461538 )  ( 5.769231 ,  3.461538 )  ( 6.153846 ,  3.461538 )  ( 6.538462 ,  3.461538
## ( 6.923077 ,  3.461538 )  ( 7.307692 ,  3.461538 )  ( 7.692308 ,  3.461538 )  ( 8.076923 ,  3.461538
## ( 8.461538 ,  3.461538 )  ( -5 ,  3.846154 )  ( -4.615385 ,  3.846154 )  ( -4.230769 ,  3.846154 )
## ( -3.846154 ,  3.846154 )  ( -3.461538 ,  3.846154 )  ( -3.076923 ,  3.846154 )  ( -2.692308 ,  3.84
## ( -0.7692308 ,  3.846154 )  ( -0.3846154 ,  3.846154 )  ( 0.7692308 ,  3.846154 )  ( 1.153846 ,  3.8
## ( 1.538462 ,  3.846154 )  ( 5.769231 ,  3.846154 )  ( 6.153846 ,  3.846154 )  ( 6.538462 ,  3.846154
## ( 6.923077 ,  3.846154 )  ( 7.307692 ,  3.846154 )  ( 7.692308 ,  3.846154 )  ( -5 ,  4.230769 )
## ( -4.615385 ,  4.230769 )  ( -4.230769 ,  4.230769 )  ( -3.846154 ,  4.230769 )  ( -3.461538 ,  4.23
## ( -3.076923 ,  4.230769 )  ( -2.692308 ,  4.230769 )  ( -2.307692 ,  4.230769 )  ( -1.538462 ,  4.23
## ( -1.153846 ,  4.230769 )  ( 0.3846154 ,  4.230769 )  ( 1.153846 ,  4.230769 )  ( 1.538462 ,  4.2307
## ( 5.769231 ,  4.230769 )  ( 6.153846 ,  4.230769 )  ( 6.538462 ,  4.230769 )  ( 6.923077 ,  4.230769
## ( 7.307692 ,  4.230769 )  ( -5 ,  4.615385 )  ( -4.615385 ,  4.615385 )  ( -4.230769 ,  4.615385 )
## ( -3.846154 ,  4.615385 )  ( -3.461538 ,  4.615385 )  ( -3.076923 ,  4.615385 )  ( -2.307692 ,  4.61
## ( -1.923077 ,  4.615385 )  ( 1.923077 ,  4.615385 )  ( 5.769231 ,  4.615385 )  ( 6.153846 ,  4.61538
## ( 6.538462 ,  4.615385 )  ( 6.923077 ,  4.615385 )  ( 7.307692 ,  4.615385 )  ( -5 ,  5 )
## ( -4.615385 ,  5 )  ( -4.230769 ,  5 )  ( -3.846154 ,  5 )  ( -3.461538 ,  5 )
## ( -2.307692 ,  5 )  ( -1.923077 ,  5 )  ( -0.3846154 ,  5 )  ( 0.3846154 ,  5 )
## ( 1.153846 ,  5 )  ( 1.538462 ,  5 )  ( 5.769231 ,  5 )  ( 6.153846 ,  5 )
## ( 6.538462 ,  5 )  ( 6.923077 ,  5 )  ( -5 ,  5.384615 )  ( -4.615385 ,  5.384615 )
## ( -4.230769 ,  5.384615 )  ( -3.846154 ,  5.384615 )  ( -3.461538 ,  5.384615 )  ( 0.7692308 ,  5.38
## ( 1.923077 ,  5.384615 )  ( 5.769231 ,  5.384615 )  ( 6.153846 ,  5.384615 )  ( 6.538462 ,  5.384615
## ( 6.923077 ,  5.384615 )  ( -5 ,  5.769231 )  ( -4.615385 ,  5.769231 )  ( -4.230769 ,  5.769231 )
## ( -3.846154 ,  5.769231 )  ( -0.7692308 ,  5.769231 )  ( -0.3846154 ,  5.769231 )  ( 0 ,  5.769231 )
## ( 0.7692308 ,  5.769231 )  ( 5.769231 ,  5.769231 )  ( 6.153846 ,  5.769231 )  ( 6.538462 ,  5.76923
## ( 6.923077 ,  5.769231 )  ( -5 ,  6.153846 )  ( -4.615385 ,  6.153846 )  ( -4.230769 ,  6.153846 )
## ( -3.846154 ,  6.153846 )  ( -0.7692308 ,  6.153846 )  ( 0.3846154 ,  6.153846 )  ( 2.307692 ,  6.15
## ( 5.769231 ,  6.153846 )  ( 6.153846 ,  6.153846 )  ( 6.538462 ,  6.153846 )  ( 6.923077 ,  6.153846
## ( 9.230769 ,  6.153846 )  ( 10 ,  6.153846 )  ( -5 ,  6.538462 )  ( -4.615385 ,  6.538462 )
## ( -4.230769 ,  6.538462 )  ( -0.3846154 ,  6.538462 )  ( 0.3846154 ,  6.538462 )  ( 2.307692 ,  6.53
## ( 5.769231 ,  6.538462 )  ( 6.153846 ,  6.538462 )  ( 6.538462 ,  6.538462 )  ( 6.923077 ,  6.538462
## ( 8.846154 ,  6.538462 )  ( 9.230769 ,  6.538462 )  ( 9.615385 ,  6.538462 )  ( 10 ,  6.538462 )
## ( -5 ,  6.923077 )  ( -4.615385 ,  6.923077 )  ( -1.538462 ,  6.923077 )  ( 0.7692308 ,  6.923077 )
## ( 1.923077 ,  6.923077 )  ( 5.769231 ,  6.923077 )  ( 6.153846 ,  6.923077 )  ( 6.538462 ,  6.923077
## ( 6.923077 ,  6.923077 )  ( 8.846154 ,  6.923077 )  ( 9.230769 ,  6.923077 )  ( 9.615385 ,  6.923077
## ( 10 ,  6.923077 )  ( -5 ,  7.307692 )  ( -4.615385 ,  7.307692 )  ( 0.3846154 ,  7.307692 )
## ( 1.923077 ,  7.307692 )  ( 5.769231 ,  7.307692 )  ( 6.153846 ,  7.307692 )  ( 6.538462 ,  7.307692
## ( 6.923077 ,  7.307692 )  ( 8.461538 ,  7.307692 )  ( 8.846154 ,  7.307692 )  ( 9.230769 ,  7.307692
## ( 9.615385 ,  7.307692 )  ( 10 ,  7.307692 )  ( -5 ,  7.692308 )  ( 1.153846 ,  7.692308 )
## ( 1.538462 ,  7.692308 )  ( 1.923077 ,  7.692308 )  ( 5.769231 ,  7.692308 )  ( 6.153846 ,  7.692308
## ( 6.538462 ,  7.692308 )  ( 6.923077 ,  7.692308 )  ( 8.076923 ,  7.692308 )  ( 8.461538 ,  7.692308
## ( 8.846154 ,  7.692308 )  ( 9.230769 ,  7.692308 )  ( 10 ,  7.692308 )  ( -5 ,  8.076923 )
## ( 0.7692308 ,  8.076923 )  ( 5.769231 ,  8.076923 )  ( 6.153846 ,  8.076923 )  ( 6.538462 ,  8.07692
## ( 8.076923 ,  8.076923 )  ( 8.461538 ,  8.076923 )  ( 8.846154 ,  8.076923 )  ( 9.230769 ,  8.076923
## ( 10 ,  8.076923 )  ( 5.384615 ,  8.461538 )  ( 5.769231 ,  8.461538 )  ( 6.153846 ,  8.461538 )
## ( 6.538462 ,  8.461538 )  ( 7.692308 ,  8.461538 )  ( 8.076923 ,  8.461538 )  ( 8.461538 ,  8.461538
## ( 8.846154 ,  8.461538 )  ( 9.230769 ,  8.461538 )  ( 10 ,  8.461538 )  ( 1.153846 ,  8.846154 )
## ( 1.923077 ,  8.846154 )  ( 5.384615 ,  8.846154 )  ( 5.769231 ,  8.846154 )  ( 6.153846 ,  8.846154
## ( 6.538462 ,  8.846154 )  ( 7.692308 ,  8.846154 )  ( 8.076923 ,  8.846154 )  ( 8.461538 ,  8.846154
## ( 8.846154 ,  8.846154 )  ( 9.230769 ,  8.846154 )  ( 10 ,  8.846154 )  ( -2.307692 ,  9.230769 )
## ( 5.384615 ,  9.230769 )  ( 5.769231 ,  9.230769 )  ( 6.153846 ,  9.230769 )  ( 6.538462 ,  9.230769
## ( 7.692308 ,  9.230769 )  ( 8.076923 ,  9.230769 )  ( 8.461538 ,  9.230769 )  ( 8.846154 ,  9.230769
## ( 9.230769 ,  9.230769 )  ( 10 ,  9.230769 )  ( 5.384615 ,  9.615385 )  ( 5.769231 ,  9.615385 )
## ( 6.153846 ,  9.615385 )  ( 6.538462 ,  9.615385 )  ( 7.692308 ,  9.615385 )  ( 8.076923 ,  9.615385
## ( 8.461538 ,  9.615385 )  ( 8.846154 ,  9.615385 )  ( 9.230769 ,  9.615385 )  ( 10 ,  9.615385 )
## ( 1.538462 ,  10 )  ( 5.384615 ,  10 )  ( 5.769231 ,  10 )  ( 6.153846 ,  10 )
## ( 6.538462 ,  10 )  ( 7.307692 ,  10 )  ( 7.692308 ,  10 )  ( 8.076923 ,  10 )
## ( 8.461538 ,  10 )  ( 8.846154 ,  10 )  ( 9.230769 ,  10 )  ( 10 ,  10 )
## ( 5.384615 ,  10.38462 )  ( 5.769231 ,  10.38462 )  ( 6.153846 ,  10.38462 )  ( 7.307692 ,  10.38462
## ( 7.692308 ,  10.38462 )  ( 8.076923 ,  10.38462 )  ( 8.461538 ,  10.38462 )  ( 8.846154 ,  10.38462
## ( 9.230769 ,  10.38462 )  ( 10 ,  10.38462 )  ( 5.384615 ,  10.76923 )  ( 5.769231 ,  10.76923 )
## ( 6.153846 ,  10.76923 )  ( 7.307692 ,  10.76923 )  ( 7.692308 ,  10.76923 )  ( 8.076923 ,  10.76923
## ( 8.461538 ,  10.76923 )  ( 8.846154 ,  10.76923 )  ( 9.230769 ,  10.76923 )  ( 10 ,  10.76923 )
## ( 5.384615 ,  11.15385 )  ( 5.769231 ,  11.15385 )  ( 6.153846 ,  11.15385 )  ( 7.307692 ,  11.15385
## ( 7.692308 ,  11.15385 )  ( 8.076923 ,  11.15385 )  ( 8.461538 ,  11.15385 )  ( 8.846154 ,  11.15385
## ( 9.230769 ,  11.15385 )  ( 10 ,  11.15385 )  ( 5.384615 ,  11.53846 )  ( 5.769231 ,  11.53846 )
## ( 6.153846 ,  11.53846 )  ( 7.307692 ,  11.53846 )  ( 7.692308 ,  11.53846 )  ( 8.076923 ,  11.53846
## ( 8.461538 ,  11.53846 )  ( 8.846154 ,  11.53846 )  ( 9.230769 ,  11.53846 )  ( 10 ,  11.53846 )
## ( 5.384615 ,  11.92308 )  ( 5.769231 ,  11.92308 )  ( 6.153846 ,  11.92308 )  ( 7.307692 ,  11.92308
## ( 7.692308 ,  11.92308 )  ( 8.076923 ,  11.92308 )  ( 8.461538 ,  11.92308 )  ( 8.846154 ,  11.92308
## ( 9.230769 ,  11.92308 )  ( 10 ,  11.92308 )  ( 5.384615 ,  12.30769 )  ( 5.769231 ,  12.30769 )
## ( 6.153846 ,  12.30769 )  ( 6.538462 ,  12.30769 )  ( 7.307692 ,  12.30769 )  ( 7.692308 ,  12.30769
## ( 8.076923 ,  12.30769 )  ( 8.461538 ,  12.30769 )  ( 8.846154 ,  12.30769 )  ( 9.230769 ,  12.30769
## ( 10 ,  12.30769 )  ( 0.3846154 ,  12.69231 )  ( 5.384615 ,  12.69231 )  ( 5.769231 ,  12.69231 )
## ( 6.153846 ,  12.69231 )  ( 6.538462 ,  12.69231 )  ( 7.307692 ,  12.69231 )  ( 7.692308 ,  12.69231
## ( 8.076923 ,  12.69231 )  ( 8.461538 ,  12.69231 )  ( 8.846154 ,  12.69231 )  ( 9.230769 ,  12.69231
## ( 9.615385 ,  12.69231 )  ( 10 ,  12.69231 )  ( -0.7692308 ,  13.07692 )  ( -0.3846154 ,  13.07692 )
## ( 0 ,  13.07692 )  ( 0.3846154 ,  13.07692 )  ( 5.384615 ,  13.07692 )  ( 5.769231 ,  13.07692 )
## ( 6.153846 ,  13.07692 )  ( 6.538462 ,  13.07692 )  ( 6.923077 ,  13.07692 )  ( 7.307692 ,  13.07692
## ( 7.692308 ,  13.07692 )  ( 8.076923 ,  13.07692 )  ( 8.461538 ,  13.07692 )  ( 8.846154 ,  13.07692
## ( 9.230769 ,  13.07692 )  ( 9.615385 ,  13.07692 )  ( 10 ,  13.07692 )  ( -1.153846 ,  13.46154 )
## ( -0.7692308 ,  13.46154 )  ( -0.3846154 ,  13.46154 )  ( 0 ,  13.46154 )  ( 0.3846154 ,  13.46154 )
## ( 5.384615 ,  13.46154 )  ( 5.769231 ,  13.46154 )  ( 6.153846 ,  13.46154 )  ( 6.538462 ,  13.46154
## ( 6.923077 ,  13.46154 )  ( 7.307692 ,  13.46154 )  ( 7.692308 ,  13.46154 )  ( 8.076923 ,  13.46154
## ( 8.461538 ,  13.46154 )  ( 8.846154 ,  13.46154 )  ( 9.230769 ,  13.46154 )  ( 9.615385 ,  13.46154
## ( 10 ,  13.46154 )  ( -1.538462 ,  13.84615 )  ( -1.153846 ,  13.84615 )  ( -0.7692308 ,  13.84615 )
## ( -0.3846154 ,  13.84615 )  ( 0 ,  13.84615 )  ( 0.3846154 ,  13.84615 )  ( 0.7692308 ,  13.84615 )
## ( 1.923077 ,  13.84615 )  ( 2.307692 ,  13.84615 )  ( 5.384615 ,  13.84615 )  ( 5.769231 ,  13.84615
## ( 6.153846 ,  13.84615 )  ( 6.538462 ,  13.84615 )  ( 6.923077 ,  13.84615 )  ( 7.307692 ,  13.84615
## ( 8.076923 ,  13.84615 )  ( 8.461538 ,  13.84615 )  ( 8.846154 ,  13.84615 )  ( 9.230769 ,  13.84615
## ( 9.615385 ,  13.84615 )  ( 10 ,  13.84615 )  ( -1.923077 ,  14.23077 )  ( -1.538462 ,  14.23077 )
## ( -1.153846 ,  14.23077 )  ( -0.7692308 ,  14.23077 )  ( -0.3846154 ,  14.23077 )  ( 0 ,  14.23077 )
## ( 0.3846154 ,  14.23077 )  ( 0.7692308 ,  14.23077 )  ( 1.923077 ,  14.23077 )  ( 2.307692 ,  14.230
## ( 5.384615 ,  14.23077 )  ( 5.769231 ,  14.23077 )  ( 6.153846 ,  14.23077 )  ( 6.538462 ,  14.23077
## ( 6.923077 ,  14.23077 )  ( 7.307692 ,  14.23077 )  ( 7.692308 ,  14.23077 )  ( 8.076923 ,  14.23077
## ( 8.461538 ,  14.23077 )  ( 8.846154 ,  14.23077 )  ( 9.230769 ,  14.23077 )  ( 9.615385 ,  14.23077
## ( 10 ,  14.23077 )  ( -5 ,  14.61538 )  ( -1.923077 ,  14.61538 )  ( -1.538462 ,  14.61538 )
## ( -1.153846 ,  14.61538 )  ( -0.7692308 ,  14.61538 )  ( -0.3846154 ,  14.61538 )  ( 0 ,  14.61538 )
## ( 0.3846154 ,  14.61538 )  ( 0.7692308 ,  14.61538 )  ( 1.153846 ,  14.61538 )  ( 1.923077 ,  14.615
## ( 2.307692 ,  14.61538 )  ( 5.384615 ,  14.61538 )  ( 5.769231 ,  14.61538 )  ( 6.153846 ,  14.61538
## ( 6.538462 ,  14.61538 )  ( 6.923077 ,  14.61538 )  ( 7.307692 ,  14.61538 )  ( 7.692308 ,  14.61538
## ( 8.076923 ,  14.61538 )  ( 8.461538 ,  14.61538 )  ( 8.846154 ,  14.61538 )  ( 9.230769 ,  14.61538
## ( 10 ,  14.61538 )  ( -2.307692 ,  15 )  ( -1.923077 ,  15 )  ( -1.538462 ,  15 )
## ( -1.153846 ,  15 )  ( -0.7692308 ,  15 )  ( -0.3846154 ,  15 )  ( 0 ,  15 )
## ( 0.3846154 ,  15 )  ( 0.7692308 ,  15 )  ( 1.153846 ,  15 )  ( 1.538462 ,  15 )
## ( 1.923077 ,  15 )  ( 2.307692 ,  15 )  ( 3.846154 ,  15 )  ( 5 ,  15 )
## ( 5.384615 ,  15 )  ( 5.769231 ,  15 )  ( 6.153846 ,  15 )  ( 6.538462 ,  15 )
## ( 6.923077 ,  15 )  ( 7.307692 ,  15 )  ( 7.692308 ,  15 )  ( 8.076923 ,  15 )
## ( 8.461538 ,  15 )  ( 8.846154 ,  15 )  ( 9.230769 ,  15 )  ( 10 ,  15 )
```

We can draw a contour plot of the function and see what is going on.

```r
zz<-branin(pnts)
zz<-as.numeric(as.matrix(zz))
zy<-matrix(zz, nrow=40, ncol=40)
contour(grid1, grid2, zy, nlevels=25)
points(y1[1], y1[2], pch=19, cex=2)
points(y2[1], y2[2], pch=19, cex=2)
points(y3[1], y3[2], pch=19, cex=2)
title(main="Branin function contour plot")
title(sub="The solid circles are the three global minima")
```



Now display the results.

```r
for (ii in 1:npt){
    vrow<-paste(cmat[ii,],sep=" ", collapse=" ")
    cat(vrow,"\n")
}
```

```
## 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 0 2 2 0 0 0 0 0 0 0 0 0 0 0 0 3 0
## 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 0 0 1 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0
## 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 1 1 0 0 1 2 2 2 2 2 2 0 0 0 0 0 0 3 0 0 0 0 0 0
## 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0
## 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 3 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 2 2 2 2 2 2 2 2 0 0 0 0 3 0 0 0 0 0 0 0 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 0 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 0 0 0 3 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 0 0 0 3 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 0 0 0 3 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 0 0 0 3 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 0 0 0 3 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 2 2 2 2 2 2 2 3 0 0 0 0 3 0 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 0 2 2 2 2 2 2 2 2 0 0 0 0 3 3 0 0 0 0 0 3 0
## 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 0 0 0 0 0 3 0
## 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 3 3 3 0 0 0 0 3 0
## 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 0 0 0 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 0 0 0 0 3 0
## 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 2 2 2 0 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 0 0 0 0 0
## 0 0 1 1 1 1 1 1 1 0 1 1 1 1 2 0 2 2 0 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0
## 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 2 2 2 2 0 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 3 0 0 0 0
## 0 0 0 0 1 1 1 1 1 1 1 0 1 1 0 2 2 2 2 0 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 3 3 0 3 0
## 0 0 0 0 1 1 1 1 1 1 1 0 0 0 2 0 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 3 3 3 3 3
## 0 0 0 0 0 1 1 3 1 1 1 1 1 2 2 0 2 2 0 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 3 3 3 3 3
## 0 0 0 0 0 1 1 0 0 1 1 1 0 2 0 3 0 0 2 2 2 2 2 2 2 2 2 2 0 0 0 0 3 3 3 3 3 3 3 3
## 0 0 0 0 0 0 1 0 0 1 1 1 2 2 2 3 3 3 0 2 2 2 2 2 2 2 2 2 0 0 0 0 0 3 3 3 3 3 3 3
## 0 0 0 0 0 0 0 0 2 0 0 2 3 2 0 3 0 0 3 2 2 2 2 2 2 2 2 2 0 0 0 0 0 3 3 3 3 3 3 3
## 0 0 0 0 0 0 0 2 2 2 2 0 0 2 3 0 0 0 3 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 3 3 3 3 3 3
## 0 0 0 0 0 0 0 3 2 2 0 3 3 0 3 0 0 0 0 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 3 3 3 3
## 0 0 0 0 0 0 0 0 2 0 0 0 3 0 0 0 0 0 0 3 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 3 3 3
## 0 0 0 0 0 0 0 0 2 0 0 0 0 3 0 0 0 0 0 0 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 3 3 3
## 0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 3 3 3 3
## 0 0 0 0 0 0 0 0 0 3 3 3 3 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 3 3 3 3
## 0 0 0 0 0 0 0 0 0 0 3 3 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 0 0 3 0 0 0 0 0 3 3 3 3
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 3 3 3 3
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 3 3 3 3
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 3 3 3
## 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 3 3 3 3
```

References

Adragni, Kofi P., R. Dennis Cook, and Seongho Wu. 2012. "GrassmannOptim: An R Package for Grassmann Manifold Optimization." *Journal of Statistical Software* 50 (5): 1–18. http://www.jstatsoft.org/v50/i05/.

Bergmeir, Christoph, Daniel Molina, and José M. Benítez. 2012. *Continuous Optimization Using Memetic*

Algorithms with Local Search Chains (Ma-Ls-Chains) in R.

Colombia, Prof. Juan David Velasquez H. Universidad Nacional de. 2012. *Smco: A Simple Monte Carlo Optimizer Using Adaptive Coordinate Sampling.* http://CRAN.R-project.org/package=smco.

Eddelbuettel, Dirk. 2012. *RcppDE: Global Optimization by Differential Evolution in C++.* http://CRAN.R-project.org/package=RcppDE.

Ivan Zelinka, Jon Clayden; based on the work of. 2011. *Soma: General-Purpose Optimisation with the Self-Organising Migrating Algorithm.* http://CRAN.R-project.org/package=soma.

Mebane, Jr., Walter R., and Jasjeet S. Sekhon. 2011. "Genetic Optimization Using Derivatives: The rgenoud Package for R." *Journal of Statistical Software* 42 (11): 1–26. http://www.jstatsoft.org/v42/i11/.

Molga, M., and C. Smutnicki. 2005. *Test Functions for Optimization Needs.* http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf.

Mullen, Katharine, David Ardia, David Gil, Donald Windover, and James Cline. 2011. "DEoptim: An R Package for Global Optimization by Differential Evolution." *Journal of Statistical Software* 40 (6): 1–26. http://www.jstatsoft.org/v40/i06/.

Murphy, Lora. 2012. *Likelihood: Methods for Maximum Likelihood Estimation.* http://CRAN.R-project.org/package=likelihood.

Mühlenbein, H., D. Schomisch, and J. Born. 1991. "The Parallel Genetic Algorithm as Function Optimizer." *Parallel Computing* 17 (6-7): 619–32.

Scrucca, Luca. 2013. "GA: A Package for Genetic Algorithms in R." *Journal of Statistical Software* 53 (4): 1–37. http://www.jstatsoft.org/v53/i04/.

Tenorio, Fernando. 2013. *Gaoptim: Genetic Algorithm Optimization for Real-Based and Permutation-Based Problems.* http://CRAN.R-project.org/package=gaoptim.

Törn, A., and A. Zilinskas. 1989. "Global Optimization." *Lecture Notes in Computer Science* 350. New York, NY, USA: Springer-Verlag.

Xiang, Yang, Sylvain Gubian, Brian Suomela, and Julia Hoeng. 2013. "Generalized Simulated Annealing for Global Optimization: The Gensa Package for R." *The R Journal* 5 (1): 13–29. http://journal.r-project.org/archive/2013-1/xiang-gubian-suomela-etal.pdf.