Google Summer of Code 2023

R Project for Statistical Computing

StochOptim: Building a unified interface for Stochastic Optimisation in R

Arqam Patel
arqamrp.github.io
arqamrp@gmail.com
BS Stats and Data Science,
Indian Institute of Technology, Kanpur

Mentors

John C Nash: profjcnash@gmail.com (Evaluating mentor)

Hans W Borchers: hwborchers@gmail.com

Paulo Cortez: pcortez@dsi.uminho.pt

Index

1. Overview
2. Goals
3. Coding plan
4. Timeline
5. Management plan
6. Tests
7. About me
8. References

Overview

R has a multitude of tools for optimization, i.e finding an optimum value of a given objective function.

For local solvers, which take a starting point as input and find a local minimum, there exist multiple packages like optim, optimx, ROI, which aim to provide a layer of abstraction above the individual solvers. This solves the problem of multiple types of input and output by different solvers, which make experimenting with different solvers tedious.

We aim to do something analogous, for stochastic or global optimizers. These aim to find the global optimum parameters and value of an objective function within a certain given search space. More than twenty R packages on CRAN are devoted to this task, with even more individual solvers within those packages. We aim to provide a single, standard interface that wraps these solvers. We also aim to enable users to call and compare multiple solvers with only a few lines of code.

Goals

Primary functionalities

1. Develop a single standard function to call any stochastic optimiser for an optimisation problem using the optimiser name as an argument.
2. Develop a function that allows a user to simultaneously call multiple solvers (and multiple versions of the same solver) and compare performance.

Additional features

1. Automated searching routines to find best/most efficient values of some control parameters for a given optimization problem
2. Visualization of each optimiser’s progress.
3. Include some of Julia’s stochastic optimisers, which often outperform those written in R.

By the end of the project, I expect both primary deliverables and at least 2 out of the 3 additional features to be achieved.

Coding plan

Potential packages and solvers:

1. DEoptim or RcppDE (Differential Evolution)
2. GenSA (Generalized Simulated Annealing)
3. Psoptim (Particle Swarm Optimizer)
    a. SPSO2007
    b. SPSO2011
4. GA (genetic algorithm)
5. rgenoud (GENetic Optimization Using Derivatives)
6. RCEIM (R Cross Entropy Inspired Method for Optimization)
7. nloptr
    a. controlled random search (CRS)
    b. multi-level single-linkage (MLSL
    c. improved stochastic ranking (ISR-ES)
    d. stochastic global optimization (StoGO)
8. metaheuristicOpt:
    a. ABC (Artificial Bee Colony)
    b. ALO (Ant Lion Optimizer)
    c. BA (Bat Algorithm)
    d. BHOA (Black Hole Optimization Algorithm)
    e. CLONALG (Clonal Selection Algorithm)
    f. CS (Cuckoo Search algorithm)
    g. CSO (Cat Swarm Optimization Algorithm)
    h. DA (Dragonfly Algorithm)
    i. FFA (Firefly Algorithm)
    j. GBS (Gravitational Based Search Algorithm)
    k. GWO (Grey Wolf Optimizer)
    l. HS (Harmony Search Algorithm)
    m. KH (Krill-Herd Algorithm)
    n. MFO (Moth Flame Optimizer)
    o. SCA (Sine Cosine Algorithm)
    p. SFL (Shuffled Frog Leaping Algorithm)
    q. WOA (Whale Optimization Algorithm)

Controls setting functions

```
DE1 <- control.DEoptim(NP = 10, maxit = 300, strategy = 1)
DE2 <- control.DEoptim(NP = 12)
pso1 <- control.psoptim(maxit = 2000)
pso2 <- control.psoptim(strategy = "SPSO2011")
```

For every solver, we will define a controls setting function that takes in the control parameters as arguments and outputs a list containing the user defined values for input parameters and default values for the rest.

This ensures that the controls actually conform to a valid set of controls for a solver. The common control options (like itermax and trace) will also be given consistent names in the arguments across solvers to alleviate confusion, and will be translated to the solver specific nomenclature in the function.

This will also allow the user to easily explore the control parameters (including description, defaults, etc) by using the control setting function's documentation.

The contents of the list DE1 will look like this (truncated for brevity):

```
$method
[1] "DEoptim"

$control
$control$VTR
[1] -Inf

$control$strategy
[1] 1

$control$NP
[1] 10

$control$itermax
[1] 300

$control$CR
[1] 0.5

$control$F
[1] 0.8

$control$bs
[1] FALSE

$control$trace
[1] FALSE
```

Default setting function(s)

We could have functions (or a function, that reads in the name from the control list) to change the default configurations of control parameters of a solver in our wrapper:

```
default.psoptim(pso2)
default.DEoptim(DE1)
# or
default.soptim(pso2)
default.soptim(DE1)
```

It could be achieved by storing defaults in an external object that can be modified by the functions, and is read in by soptim() at the time of the loading the library.

Singular wrapper (soptim):

```
# using default controls
soptim(fn = rastrigin, lower = c(-5, -5), upper = c(5, 5), method =
"DEoptim")

# using custom controls
soptim(fn = rastrigin, lower = c(-5, -5), upper = c(5, 5), control = DE1)
```

The wrapper soptim would make function calls to the specific optimiser, e.g. DEoptim, along with the optimisation problem (and if specified, the control parameters). The control setting functions would allow us to bypass the need for having separate wrapping functions for each solver, since we can ensure the conformity of control options.

So, we can directly structure soptim as one somewhat complex if-else statement that calls the optimiser specified in the parameters method or control.

Arguments

- Arguments defining optimisation problem:

Argument	**Description**
fn	Objective function to be optimised
lower, upper	Search space for optimisation
minimize	Boolean indicating whether it is to be minimised (Default = TRUE)

We can infer dimensionality of the optimization problem from the length of lower/upper.

- Arguments giving solver-agnostic control options:

Argument	**Description**
maxf	maximum number of objective function calls
maxtime	maximum time or which algorithm can run
abstol	targeted optimum value (default - -Inf for minimize = TRUE); The method converges once the best fitness obtained is less than or equal to target.
trace	Boolean indicating whether progress should be printed at each iteration

These should override the solver specific control options (e.g. DE1) (or the default controls in case "method" is used) at the time of the function call. In case of soptim, if the value in the control list is non-default, it will take precedence.

Also, while maxf and maxtime are natively supported in many solvers, some may not have explicit options to stop upon exceeding time/no. of function calls allotted. For those, we may have to construct a more elaborate wrapper using withTimeout {R.utils} that runs for a specified amount of time and returns the last values found.

- Arguments giving solver-specific control options:

Argument	**Description**
method	using default controls
control	using custom controls

Output

- A list containing the following should be output:

Element	Description
value	objective function value at the optimum found
time	time taken
fevals	no. of function calls to objective function
niter	number of iterations of algorithm
exitcode	exit code indicating reason for termination (e.g. 0: target reached, 1: maxf reached, 2: maxiter reached, 3: maxtime reached)

In most solvers, exit code is not an output option so we would have to infer the convergence exit code from the solver history.

Comparison wrapper (soptimx):

The comparison wrapper would make calls to the central wrapper for each desired method, and then tabulate the results it receives.

```
# default controls
soptimx(fn = rastrigin, lower = c(-5, -5), upper = c(5, 5), methods =
c("DEoptim", "psoptim"))

# all methods at once, using default controls
soptimx(fn = rastrigin, lower = c(-5, -5), upper = c(5, 5), methods = "all")

# all except specified ones, under default controls
soptimx(fn = rastrigin, lower = c(-5, -5), upper = c(5, 5), exclude =
c("DEoptim", "psoptim"))

# custom controls
soptimx(fn = rastrigin, lower = c(-5, -5), upper = c(5, 5), controls =
list(DE1, DE2, pso1, pso2 ))
```

Arguments:

- Arguments defining optimisation problem: same as soptim
- Arguments giving solver-agnostic control options: same as soptim
- Ways of passing methods to be used:

Argument	**Explanation**
method = c("DEoptim", "psoptim")	vector of methods to be applied (using default controls)
method = "all"	all available methods should be applied
exclude = c("DEoptim", "psoptim")	all methods except these should be applied
control = list(DE1, pso1)	list of custom, solver-specific control lists defined using control setting functions.

- Arguments (Boolean) specifying which fields to include in output

Argument	**Description**
par	Whether the par vectors are to be output separately, default FALSE
time	Whether the time taken should be output
fevals	Whether the no. of function calls taken should be output
niter	Whether the number of iterations of algorithm should be output
exitcode	Whether exit code should be output

Outputs

1. If par = FALSE: Only the dataframe as given below.
2. If par = TRUE: A list containing the following:
    a. A dataframe containing the selected fields for all solvers, ordered by value, for each method:

Column	**Description**

value	objective function value at the optimum found
time	time taken
fevals	no. of function calls to objective function
niter	number of iterations of algorithm
exitcode	exit code indicating reason for termination (e.g. 0: target reached, 1: maxf reached, 2: maxiter reached, 3: maxtime reached)

b. A list of the par vectors of optimum parameters found by the various solvers, with the element names corresponding to the solver name or control variable name

Visualization

Some possible plots that could be generated :

- Objective function value of optimum vs time/iteration (possibly live)
- Simple 2D MDS plot depicting the approach paths of different optimisers
- 3D plot, with MDS on x and y axes, and value on z axis

Test functions

I have already written several functions for testing global optimization such as Rastrigin, Styblinski–Tang and Rosenbrock. We can use these and more such functions to compare solver performance and to write examples. During development, these are also indispensable for testing purposes.

Finding best control parameters

We can iteratively compare different combinations of control parameters on the basis of the time taken, type of convergence and objective function value. To start, we can implement a function to find the best value for a single control parameter, after trying it out with a number of different values. Next, we can attempt to do the same on a two dimensional parameter search space, for example, to find the best values of NP and maxit for DEoptim - that converge fastest with the required accuracy.

Julia solvers

We can use the package JuliaCall in order to integrate Julia's stochastic solvers, which may be faster than those written in R.

Optimization.jl in Julia performs a function similar to the package we propose. It acts as a layer of abstraction between the user and the multiple packages for optimization.

Some specific packages in Julia for Global Optimization that could be included are:

- Metaheuristics.jl:
  metaheuristic algorithms for global optimization that do not require for the optimized function to be differentiable
- BlackBoxOptim.jl:
  (Meta-)heuristic/stochastic algorithms that do not require for the optimized function to be differentiable.
- NOMAD.jl:
  interface to NOMAD, which is a C++ implementation of the Mesh Adaptive Direct Search algorithm (MADS), designed for difficult blackbox optimization problems
- Evolutionary.jl:
  various evolutionary and genetic algorithms
- MultistartOptimization:
  implements a global optimization multistart method which performs local optimization after choosing multiple starting points
- Nonconvex.jl: implements and wraps nonconvex constrained optimization algorithms

Assistance functions

soptim.solvers():

List of all compatible solvers with currently installed libraries

soptim.more():

List of compatible solvers not currently installed

Timeline

Week	Duration	Tasks
	Community Bonding period	
	May 4- May 10	Review the source code of the packages optimx and ROI, which are wrappers for local optimization. Read more about the basics of package development in R.
	May 11-May 17	Select possibly 20-30 solvers to initially include from amongst those listed under the "Global and Stochastic Optimization" heading in the CRAN Task View: Optimization and Mathematical Programming. Delineate the compulsory ones and the ones that can be added later and maintain dev-docs on package dependencies.
	May 18- May 28	Document (in a tabulated form) the arguments, control parameters and outputs of the selected solvers (done already for psoptim, DEoptim and GenSA). Make a list of solver agnostic controls (along with defaults for each solver). Finalize nomenclature scheme for functions, solvers, controls etc. Write introductory documentation to the package.
	Coding period	
1	May 29 - June 4	Develop a prototype of soptim() that works with three methods with default options using the method argument, trace set to off and maxtime, maxf set to NULL. Write some more test functions with known optima and use those to test functioning. Benchmark the wrapper against simple solver calls.
2	June 5 - June 11	Add other functions to soptim() in default mode and test.
3	June 12 - June 18	Define control setting functions for three solvers and test. Implement trace control for each solver.
4	June 19 - June 25	Write control setting functions for the rest of solvers. Write detailed documentation for each, including defaults, examples.
5	June 26 - July 2	Write default setting function(s). Incorporate maxtime and maxf limits for functions without explicit provisions.

6	July 3 - July 9	Add exit code functionality to solvers which do not have it. Build all the functions as an R package and pass R CMD check.
	MIDTERM EVALUATION	
7	July 10 - July 16	Write the code for soptimx() for default and custom controls using control or method inputs. Benchmark it against a loop calling solvers.
8	July 17 - July 23	Implement the method = “all” and except functionalities. Write code for assistance functions. Explore addition of Julia solvers.
9	July 24 - July 30	Try to implement visualization options for optimiser progress.
10	July 31 - August 6	Try to implement search routines to find best control parameters automatically.
11	August 7 - August 13	Write informative error messages for scenarios like non installed solvers, nonconforming controls etc. Build package and perform R CMD check.
12	August 14 - August 21	Buffer week for troubleshooting.
	August 21- August 28	Submission. Mentor evaluations.
		End term evaluation

Coding project management

Code submission

I plan to use Github and push commits in a separate branch and merge it with the main one after consultation with the mentors. I plan to push commits at least every two days.

Documentation

I will create a progress tracking spreadsheet with a list of the tasks accomplished daily (along with the corresponding log reference) and the tasks I plan to accomplish in the coming week.
I plan to document (in a tabulated form) the arguments, control parameters and outputs of the selected solvers. Documentation on the various control parameters can be accessed by users through the control setting functions' documentation.
I will use roxygen2 for creating documentation .Rd files for functions of the package while I also plan to include package vignettes created using Rmarkdown that provide a comprehensive guide on the various optimisers available and how the functions can be used.

For developers, apart from comments, I plan to include a blueprint of how new solvers can be added and a detailed explanation of the wrapper structure as vignettes.

Testing

For testing the overall package state, I plan to run a weekly R CMD check and reinstall the package periodically in my R environment. For testing, I will compare the wrapper outputs to the original package function's output using seeds for reproducibility. I will use rbenchmark to measure the performance of soptim() compared to a simple loop calling the various optimisers. I will also profile using profvis to check for potential efficiency improvements. I also plan to test exception handling by testing using nonconforming arguments etc.

After GSoC

I wish to continue trying to implement the secondary features of the project after the official GSoC period. I will also be glad to continue maintaining and further developing the project afterwards. I intend to pursue research in statistics and related domains in the future and hope to develop more such tools including original ones that can be used to perform analysis.

Tests

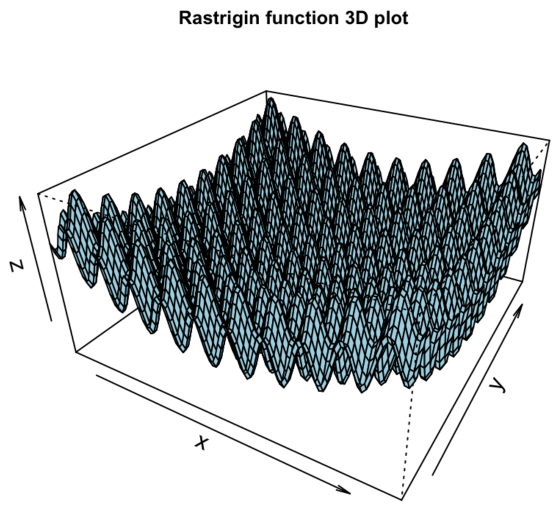
1. Rastrigin function:

I implemented the Rastrigin function, a common test function for stochastic optimisation, for n-dimensional vectors, using the definition on its Wikipedia page.

```
rastrigin <- function(X, A = 10, n= 2){
  n*A + sum(X^2 - A*cos(2*pi*X))
}
```

I then plotted it in three dimensions (for 2D vectors, with the z axis representing function value).

```
x <- seq(-5.12, 5.12, length.out = 100)
y <- seq(-5.12, 5.12, length.out = 100)
z <- matrix(0,100,100)
for(i in 1:length(x)){
 for(j in 1:length(y)){
   z[i,j] = rastrigin(c(x[i],y[j]))
 }
}
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, main = "Rastrigin function 3D
plot", cex.main =.8, col = "lightblue")
```



2. optim():

I used the default Nelder-Mead local optimiser on five randomly initialised starting points (setting a seed for reproducibility). Then, fixing the starting point, I used the five different algorithm options within optim() (using separate calls for each method) to compare them on the basis of the minima found and time taken (measured using rbenchmark).

```
n <- 5
set.seed(1)
init <- matrix(runif(n = 2*n, -5.12, 5.12), nrow = n, ncol = 2)
vals <- numeric(n)
opt <- matrix(0, n, 3)
for(i in 1:n){
  vals[i] <- rastrigin(init[i,])
  optimum <- optim(par = init[i,], fn = rastrigin)
  opt[i, 1] <- optimum$par[1]
  opt[i, 2] <- optimum$par[2]
  opt[i, 3] <- optimum$value
}
mat <- cbind(init, vals, opt)
df <- as.data.frame(mat)
colnames(df) <- c("p1_init", "p2_init", "value_init", "p1_optim", "p2_optim",
"value_optim")
df
```

Output:

```
##      p1_init   p2_init value_init   p1_optim  p2_optim value_optim
## 1 -2.4011913  4.079510   41.76422 -1.9899000  3.979759    19.89908
## 2 -1.3094513  4.553475   55.53880 -0.9949912  4.974703    25.86868
## 3  0.7460184  1.646569   29.56874  0.9949811  1.989927     4.97479
## 4  4.1800478  1.322128   39.34412  3.9797852  1.989949    19.89908
## 5 -3.0547770 -4.487309   50.02228 -2.9848159 -3.979761    24.87385
```

After this, I used the ucminf function to implement another local optimiser (not available in optim) on those five points, and then the optimx function to run and compare multiple methods with the same initial point using one function call.

3. Stochastic optimisation:

I use three different global optimisers: DEoptim, GenSA and psoptim to try and find a global minimum for the Rastrigin function . The three are based on different algorithms: Differential Evolutionary optimisation, Generalized Simulated Annealing, and Particle Swarm Optimisation. As evident, the three have very different formats of input, control and outputs, and using these different interfaces is somewhat tedious.

```
library(DEoptim)
library(GenSA)
library(pso)
df3 <- as.data.frame(matrix(0, 3, 4), row.names = c("DEoptim", "GenSA",
"psoptim"))
colnames(df3) = c("p1", "p2", "value", "time")

df3[,4] <- benchmark(
 deoptim <- DEoptim(fn = rastrigin, lower = c(-5.12,-5.12), upper = c(5.12, 5.12),
control = DEoptim.control(trace = F)),
 gensa <- GenSA(fn = rastrigin, lower = c(-5.12,-5.12), upper = c(5.12, 5.12)),
 pso <- psoptim(fn = rastrigin, lower = c(-5.12, -5.12), upper = c(5.12, 5.12),
 par = c(NA,NA)), columns = "elapsed", replications = replications)
DEopt <- deoptim$optim

df3[1,1:3] <- c(DEopt$bestmem[1], DEopt$bestmem[2], DEopt$bestval)
df3[2,1:3] <- c(gensa$par[1], gensa$par[2], gensa$value)
df3[3,1:3] <- c(pso$par[1], pso$par[2], pso$value)
```

Output:

```
##                      p1            p2       value  time
## DEoptim -1.357104e-06 -8.588545e-06 1.49994e-08 0.133
## GenSA    2.843176e-10  1.177885e-10 0.00000e+00 0.983
## psoptim  3.376046e-09 -6.880894e-10 0.00000e+00 1.802
```

We can observe that DEoptim is the fastest optimiser under the respective default conditions, while psoptim and GenSA are slower but better

Next, for these solvers, I tried to vary the controls and sensitivity and compare time taken and the objective function value.

DEoptim

For DEoptim, tuning the arguments NP (number of population members) and itermax (maximum no. of iterations) may be useful to get better estimates of the global minimum. The default values of NP is 50 and itermax is 250.

```
##                          p1            p2        value  time
## default       6.402712e-06 -2.388698e-06 9.265030e-09 0.134
## NP=75         3.107653e-07  8.397239e-08 2.055955e-11 0.436
## NP=200        2.216914e-06 -1.503676e-06 1.423611e-09 1.154
## itermax=375  -9.068795e-10  4.398613e-09 3.552714e-15 0.255
## itermax=1000 -9.581122e-11  4.609981e-10 0.000000e+00 0.672
```

We can see that the values we get by increasing either NP or itermax are closer to the origin than the previous ones. However, we see that the optimiser is more sensitive to changes in itermax.

GenSA

Here, we can tune maxit (maximum no. of iterations), temperature (a function argument that controls the probability of accepting worse solutions during the search process).

```
##                               p1            p2     value  time
## default             -2.827916e-12  5.476625e-13 0.0000000 0.984
## temperature = 1      1.391409e-14 -7.015557e-13 0.0000000 1.267
## temperature = 10    -2.460481e-09  5.711768e-10 0.0000000 1.068
## temperature = 100    8.868914e-12 -3.636632e-11 0.0000000 1.064
## maxit = 100          1.918515e-10  9.949586e-01 0.9949591 0.013
## maxit = 375         -1.061417e-12  2.191422e-12 0.0000000 0.042
## maxit = 1000        -7.361776e-13  6.851085e-13 0.0000000 0.108
```

In the case of GenSA, the estimated optimum proposed by the default function arguments are quite good, producing a function value that underflows as 0.

It is also quite slow. The package authors note: “The default values of the control components are set for a complex optimization problem. For usual optimization problem with medium complexity, GenSA can find a reasonable solution quickly so the user is recommended to let GenSA stop earlier by setting threshold.stop.”

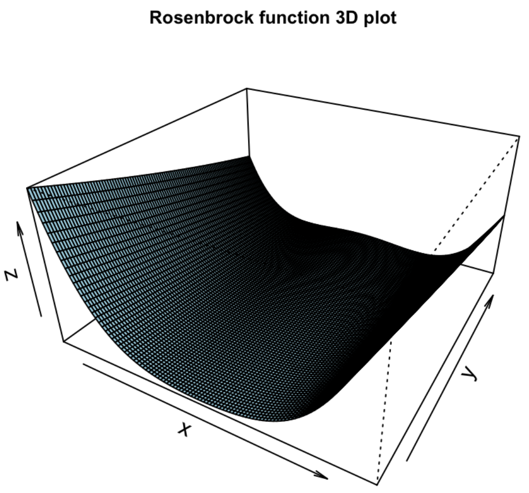
Evaluating the estimates in terms of the distance from the origin, increasing the maxit does seem to improve the estimates. However, in case of temperature, no consistent (monotonous) pattern is visible. According to the GenSA package documentation however: “For very complex optimization problems, the user is recommended to increase maxit and temp.” It is possible that the low dimensionality of the space we’re considering for our Rastrigin function makes it relatively simple and thus tweaking the temperature does not yield much benefit.

4. Other test functions:

I implemented commonly used test functions for stochastic optimisation, for n dimensional inputs.

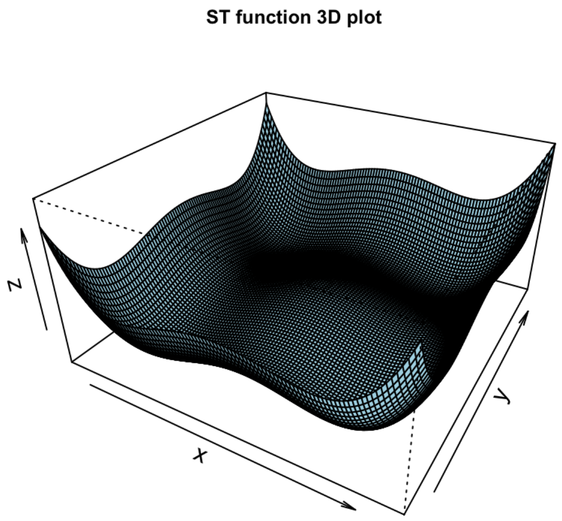
Rosenbrock function:

$$f(\mathbf{x}) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2]$$



Styblinski–Tang function:

$$f(x) = \frac{1}{2}\sum_{i=1}^{n} (x_i^4 - 16x_i^2 + 5x_i)$$



5. Package building

https://github.com/arqamrp/stochoptimtests/tree/main/testfunctions

I packaged the functions I had written as an R package that passed R CMD check. For that, I had to write documentation for the various functions, which I did using Roxygen2. I also had to include a License, which I did using the package usethis.

```
(base) arqam@Arqams-MacBook-Pro stochoptimtests % R CMD check testfunctions_0.1.0.tar.gz
* using log directory '/Users/arqam/Documents/GitHub/stochoptimtests/testfunctions.Rcheck'
* using R version 4.2.2 (2022-10-31)
* using platform: aarch64-apple-darwin20 (64-bit)
* using session charset: UTF-8
* checking for file 'testfunctions/DESCRIPTION' ... OK
* checking extension type ... Package
* this is package 'testfunctions' version '0.1.0'
* package encoding: UTF-8
* checking package namespace information ... OK
* checking package dependencies ... OK
* checking if this is a source package ... OK
* checking if there is a namespace ... OK
* checking for executable files ... OK
* checking for hidden files and directories ... OK
* checking for portable file names ... OK
* checking for sufficient/correct file permissions ... OK
* checking whether package 'testfunctions' can be installed ... OK
* checking installed package size ... OK
* checking package directory ... OK
* checking DESCRIPTION meta-information ... OK
* checking top-level files ... OK
* checking for left-over files ... OK
* checking index information ... OK
* checking package subdirectories ... OK
* checking R files for non-ASCII characters ... OK
* checking R files for syntax errors ... OK
* checking whether the package can be loaded ... OK
* checking whether the package can be loaded with stated dependencies ... OK
* checking whether the package can be unloaded cleanly ... OK
* checking whether the namespace can be loaded with stated dependencies ... OK
* checking whether the namespace can be unloaded cleanly ... OK
* checking loading without being on the library search path ... OK
* checking dependencies in R code ... OK
* checking S3 generic/method consistency ... OK
* checking replacement functions ... OK
* checking foreign function calls ... OK
* checking R code for possible problems ... OK
* checking Rd files ... OK
* checking Rd metadata ... OK
* checking Rd cross-references ... OK
* checking for missing documentation entries ... OK
* checking for code/documentation mismatches ... OK
* checking Rd \usage sections ... OK
* checking Rd contents ... OK
* checking for unstated dependencies in examples ... OK
* checking examples ... OK
* checking PDF version of manual ... OK
* DONE
Status: OK
```

About me

Bio

I am a sophomore undergraduate in Statistics and Data Science at the Indian Institute of Technology, Kanpur. My introduction to R took place through the course, Data Science Lab I, taught by Dr Dootika Vats. In the course, I learned the fundamentals of R, including things like benchmarking, Rshiny, Rmarkdown and Rcpp. I have done college courses on linear algebra and multivariable calculus. I am also currently doing a course on Statistical Computing, which includes optimisation methods, and am familiar with some of the theoretical content of the project.

Contact details

Time Zone: India (UTC+5:30)

Github LinkedIn

Address: Hall 5, IIT Kanpur, Uttar Pradesh, India, 208016

Email: arqamrp@gmail.com arqamrp21@iitk.ac.in | Mobile: +91 7715863670

System: MacOS 13.2.1

Affiliation

BS Statistics and Data Science (2021-25), Indian Institute of Technology, Kanpur

Contact to verify: Dr Suprio Bhar, suprio@iitk.ac.in

Schedule conflicts

I plan on taking a humanities course in the summer term at the institute, which wouldn't hamper my functioning. From July 31, I might be able to commit a bit less time relatively, since the regular term will commence. Consequently, I'll be able to manage ~30 hours a week in August. I plan to compensate for this by getting the maximum possible work on the comparison wrapper done before July 31. I plan to do this by spending more time per week (40-45 hours), and leave a buffer for unexpected delays or obstacles in the end.

Mentor correspondence

I have been in contact with the mentors since 2 March 2023 through email. I am grateful for the inputs, ideas and feedback they have provided for this proposal.

References

1. CRAN Task View: Optimization and Mathematical Programming, Theußl, Borchers, and Schwendinger
2. Continuous Global Optimization in R, Katharine M. Mullen
3. Julia Global Optimization packages
4. Modern Optimization with R, Paulo Cortez
5. Nonlinear Parameter Optimization, John C Nash
6. Unifying Optimization Algorithms to Aid Software System Users: optimx for R, John C. Nash, Ravi Varadhan
7. DEoptim: An R Package for Global Optimization by Differential Evolution, Katharine M. Mullen, et al
8. ROI: An Extensible R Optimization Infrastructure, Stefan Theußl, et al
9. R Packages, Hadley Wickham and Jennifer Bryan
10. Optimization.jl: A Unified Optimization Package
11. JuliaCall: Seamless Integration Between R and 'Julia'