README

Hao Li

20190303

rMathModeling

[1st stable release 1.0.0]

Overview

This package is designed to manipulate general models.

Processes to crunch a problem with models are summarized to 3 groups:

- Optimization models
- Dynamic models
- Probability models

From the structure introduced by the book Mathmatical Modeling 4th Ed. written by M. Meerschaert.

This package is coded from the code to work through problems in the book. And the functions and data structure can be easily reused to work on more general modeling tools.

You may call these functions to build individual models, or wrap them up to manage systems with multiple models.

Installation and a quick start

A copy of R is needed to run the package. You can get a local version from https://www.r-project.org/ (https://www.r-project.org/) An IDE (e.g. Rstudio, Microsoft Visual Studio, Atom …)is recommended/

To install, use

```
install.packages('devtools')# skip this if you already have devtools
devtools:: install_github('HaoLi111/rMathModeling')
```

Type this to start up. 3 groups of functions, optim_, dynam_, and probs_ will be introduced. The use of ply_ will be introduced on the way.

```
library(rMathModeling)
```

to be written

optim_

Functions to operate operation processes are named with optim_…, and generally followed by the abbreviation of the method.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

1 Var optimization

The general structure of input(s) is:

function(f, init1, init2, n)

where:

- f is the function to be optimized (taking minimum)
- init1 is the 1st initial value to start with (generally the lower bound of the domain, see more for each method using help())
- init2 is the 2nd initial value to start with (generally the upper bound)
- if init1 is assigned with a vector with length 2 then the vector will be read as the limit of the 1D domain.
- n is the number of iterations (or length of vectorization) to be done on the function.

The general structure of the output is:

- $opt is the optimized(minimum)value of the range
- $dep is the corresponding value of the dependent variable

for example:

optimize function

y={\sqrt{{35-x^{1/30}}*x^{1/3}}*sin(x)*log(x)

on the range (3.5,20),with 100 random numbers. Time the process, and plot the 1D function.

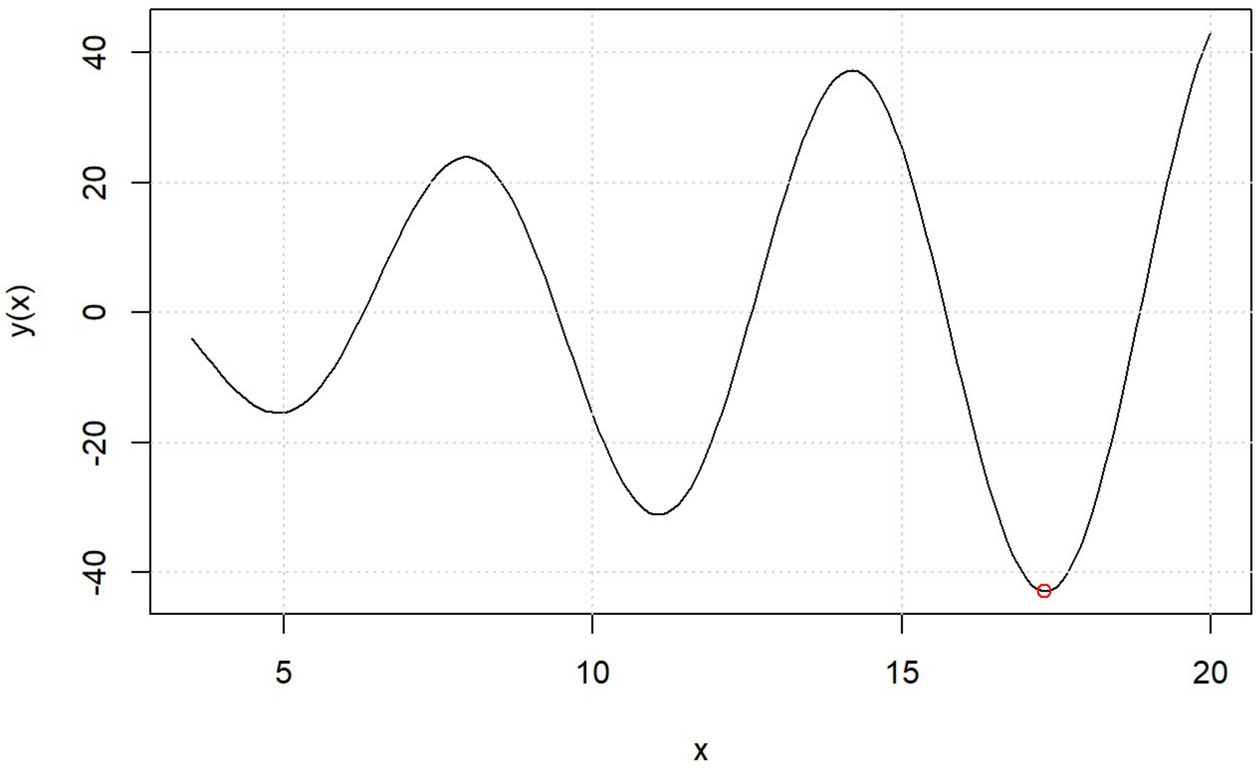
```
y=function(x) sqrt(35-x^(1/30))*x^(1/3)*sin(x)*log(x)
system.time({

re = optim_rand(y,
                init1 = 3.5,init2=20,n=1e2)
print(re)

})
```

```
## $opt
## [1] -42.925
##
## $dep
## [1] 17.31694
```

```
##    user  system elapsed
##       0       0       0
```

```
x = seq(from=3.5,to=20,by=.1)
plot(x,y(x),type = 'l');grid();points(re$dep,re$opt,col = 'red')
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js



- more 1D optimization methods availiable listed on the package document index.

Multi-variable function optimization

You can choose to directly run a multi-variable process or to use the ply_ function to apply a single variable optimization method to the mullti-var problem. An example of applying midpoint 1D optimization with ply_shift_coord id shown.

To find the minimum of function

$$x^2 + y^2 - 3 * z^2$$

in the domain

$$x \in [1,3]; y \in [0,2]; z \in [-1,4]$$

with initial value

$$x \leftarrow 1; y \leftarrow 1.5, z \leftarrow 1$$

```
ply_shift_coord((function(l) l$x^2+l$y^2-3*l$z^2),iter = list(x = c(1,3),
y = c(0,2),
z = c(-1,4)),
init =  list(x=1,y=1.5,z = 1),
optimFunc = optim_midp)#note the last line specifies th 1d func to take in
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

```
## $opt
## [1] -47
##
## $dep
## $dep$x
## [1] 1
##
## $dep$y
## [1] 7.450581e-09
##
## $dep$z
## [1] 4
```

Other methods for multi-variable optimization will be developed in the future to address the limitation of the optim() function in base r. Also, functions with parallel process written is named parOptim_…

The optimization functions may also be helpful for splines and morphs fitting. When handling large amount of multidimentional data, fitting equations may need a starting value and an error functionn to be taken in to optimization functions.

dynam_

Functions to operate dynamic processes are named after dynam_…

All dynamic model simulation functions can be summarized to 2 parts:

- dynamic_update_… : The intrinsic of the process, describes how things change with certain state.
- dynamic_record…/dynamic_cat/dynamic_…step : Recorder of the dynamic simulations

Some dynamic updation equations has already been built in the package with the form

function(x,param)

where - x is a vector or list containing all required infomation of the previous state, and - param is the extra (constant-like) parameters to take in.(constant-like: the constants may in the long run be updated as variables but unecessary to update in the local dynamic runs in the short term)

The dynam_records takes over your initial values, iterables and other conditions and give extra memory-saving options, and save them as matrix-like variables.

The dynam_cat directly saves the contents to files. Use it if recording a large number of iterations are required.

The dynam_…Step only returns the step it takes when certain condition is reached, and can be helpful to check convergence and rendering Fractals(IFS and cellular automata can be analysed as dynamic system).

More complexed structure of variables can be organized as list-like objects. And it is likely to be classed as ‘ObjSys’ in the future, which will take the structure below.

- $attr attribute of the Object System
- $Global $var and $con storing variables and constants
- … local variables

And it will run like a game engine with automatic players.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

dynam_update

A number of update functions are coded for convenience.

```
dynam_update_Lorentz_param
```

```
## function(x,param){
##   dynam_update_Lorentz(x,param$dt,param$Sigma,param$r,param$b)
## }
## <bytecode: 0x000000001c7f0818>
## <environment: namespace:rMathModeling>
```

```
dynam_update_Julia
```

```
## function(z,C) z=z^2+C
## <bytecode: 0x000000001c7825c8>
## <environment: namespace:rMathModeling>
```

```
dynam_update_lin_param
```

```
## function(x,param) param$A%*%x+param$b
## <bytecode: 0x000000001c7220b0>
## <environment: namespace:rMathModeling>
```

```
dynam_update_compete_2species_param
```

```
## function(x,param){
##   dynam_update_compete_2species(x,param$dt,param$k1,param$k2,param$C1,param$C2,param$alpha)
## }
## <bytecode: 0x000000001c6c4030>
## <environment: namespace:rMathModeling>
```

You can write your own ones

```
dynam_update_brownian_0_1=function(x) x+rnorm(2, mean=0, sd=1)
```

Time-series prediction can also be done with this simulation.

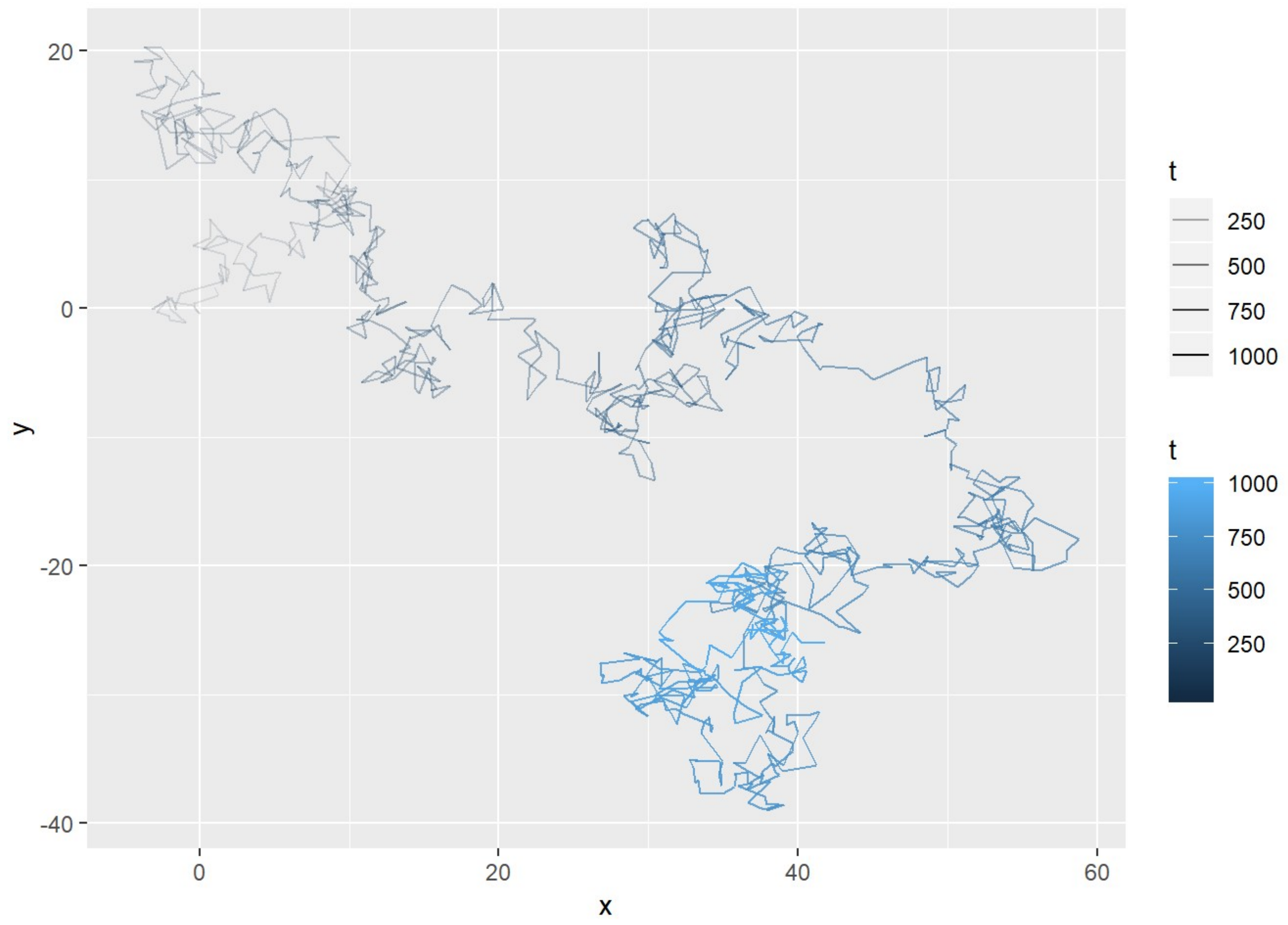
Youcan also write your own ones

dynamic_record

Run the brownian motion defined above.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

```
dynam_runif=dynam_record(init = c(1,1),index=1:1000,
                         Fupdate=dynam_update_brownian_0_1)

colnames(dynam_runif)= c('t','x','y')
dynam_runif = as.data.frame(dynam_runif)
ggplot2 :: ggplot(dynam_runif,aes(x = x,y=y,color=t,alpha=t)) + geom_path()
```



Run a weather report(demo),

```
state = list(dt = .005,Sigma = 10,b=8/3,r = 18)
system.time({
  dynam_l=dynam_record(init = c(6.7,6.7,17),index=1:1e4,
                         Fupdate=dynam_update_Lorentz_param,#make sure no bracket here
                         param = state)#take in optional param

})
```

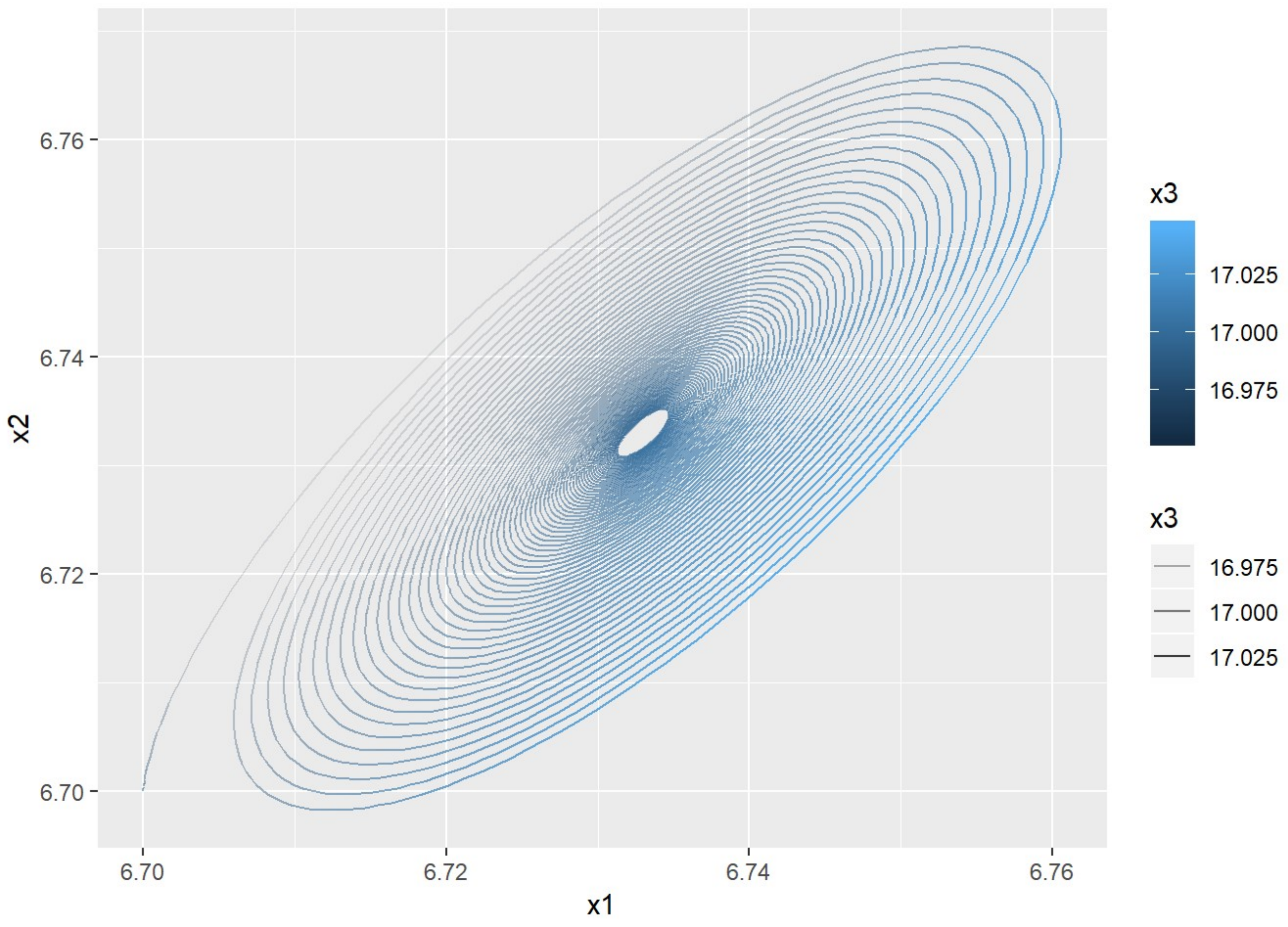
```
##    user  system elapsed
##    0.07    0.00    0.06
```

```
nrow(dynam_l)
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

```
## [1] 10000
```

```
colnames(dynam_l) = c('t','x1','x2','x3')
dynam_l = as.data.frame(dynam_l)
#library(ggplot2)
ggplot2 :: ggplot(dynam_l,aes(x = x1,y=x2,color=x3,alpha=x3)) + geom_path()
```



Here is my memory limit, if I want to run with much smaller time step(e.g 1/100 of the previous) while not using up my memory, I can let the function record each 100 steps.

```
memory.limit()
```

```
## [1] 16245
```

```
state = list(dt = .005/100,Sigma = 10,b=8/3,r = 18)
system.time({
  dynam_l=dynam_record(init = c(6.7,6.7,17),index=1:1e6,#x100
                       Fupdate=dynam_update_Lorentz_param,#make sure no bracket here
                       param = state,
                       record_per = 100)#take in optional param

})
```

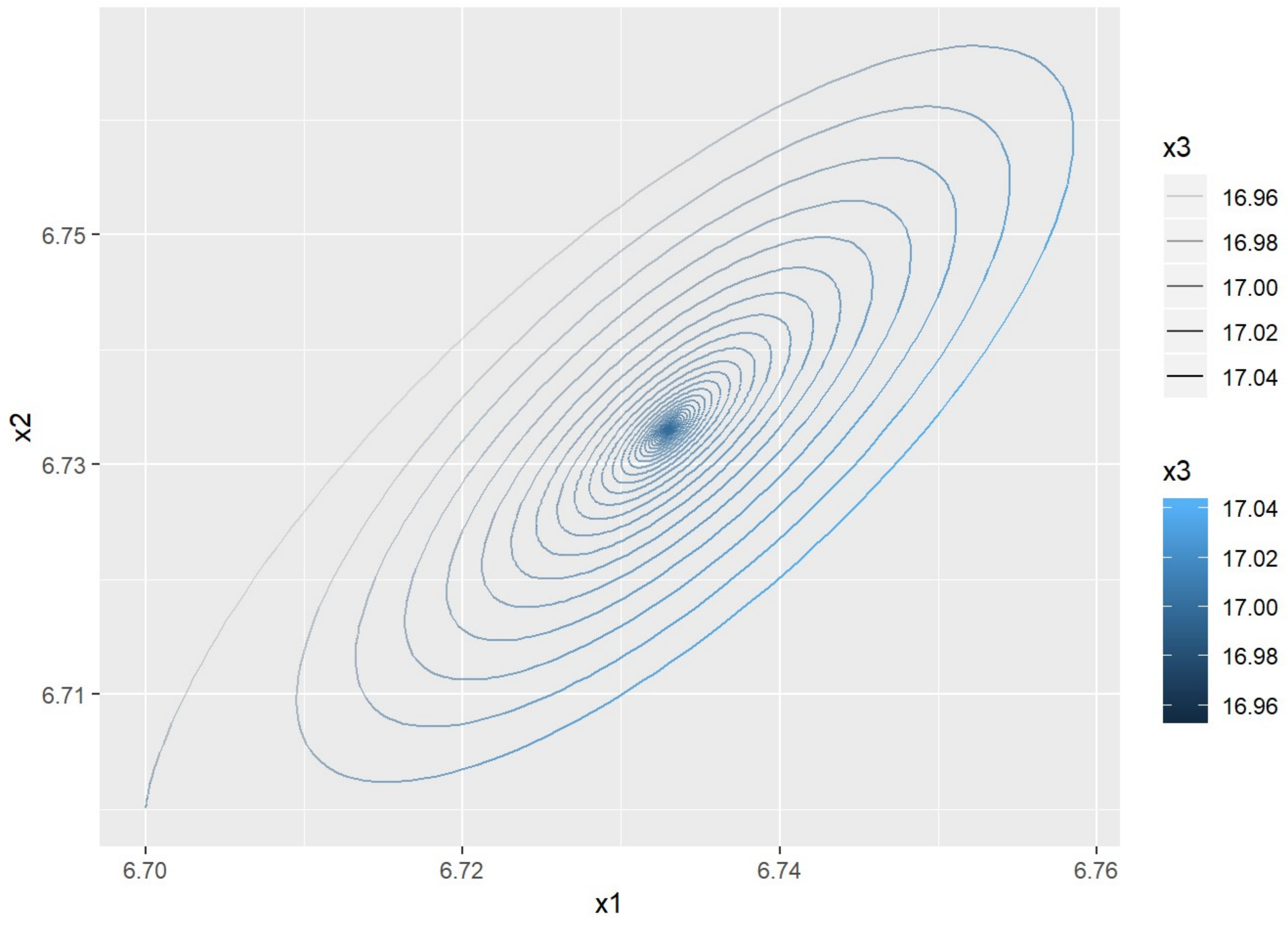
Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

```
##    user  system elapsed
##    4.77    0.00    4.86
```

```
nrow(dynam_l)
```

```
## [1] 10000
```

```
colnames(dynam_l) = c('t','x1','x2','x3')
dynam_l = as.data.frame(dynam_l)
#library(ggplot2)
ggplot2 :: ggplot(dynam_l,aes(x = x1,y=x2,color=x3,alpha=x3)) + geom_path()
```

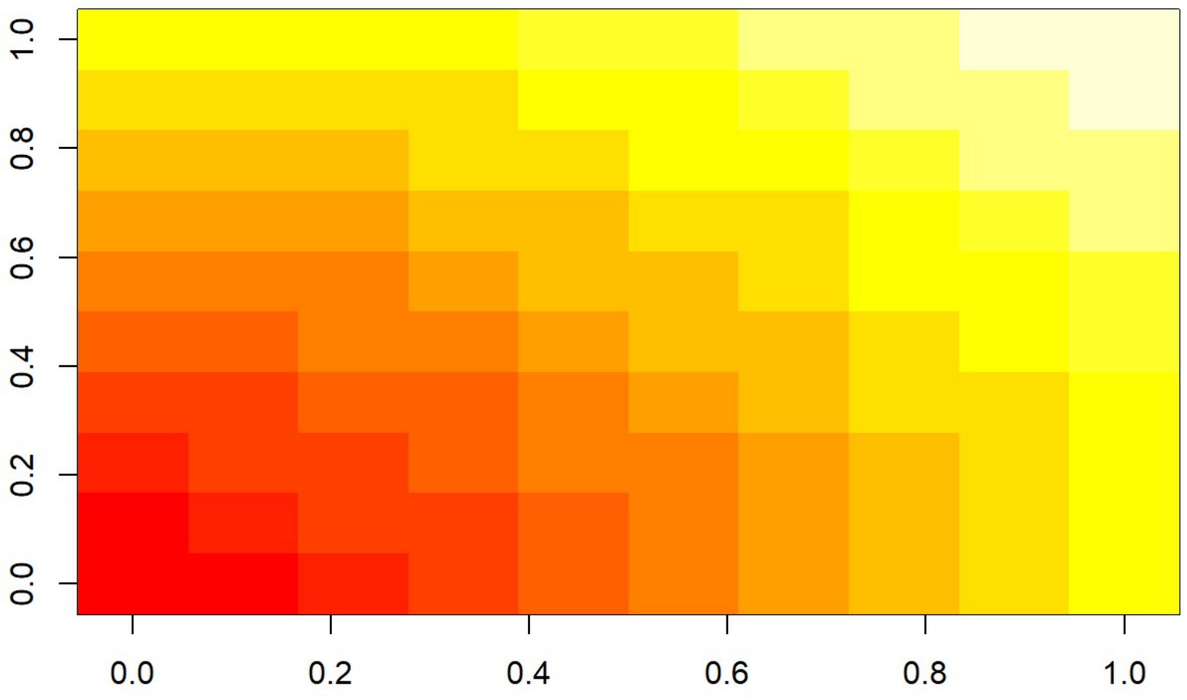


dynamic_…Step

First there is a variation of ply_ function called ply_complexBase which let you loop on a complex plane. E.g. loop to calculate the modular.

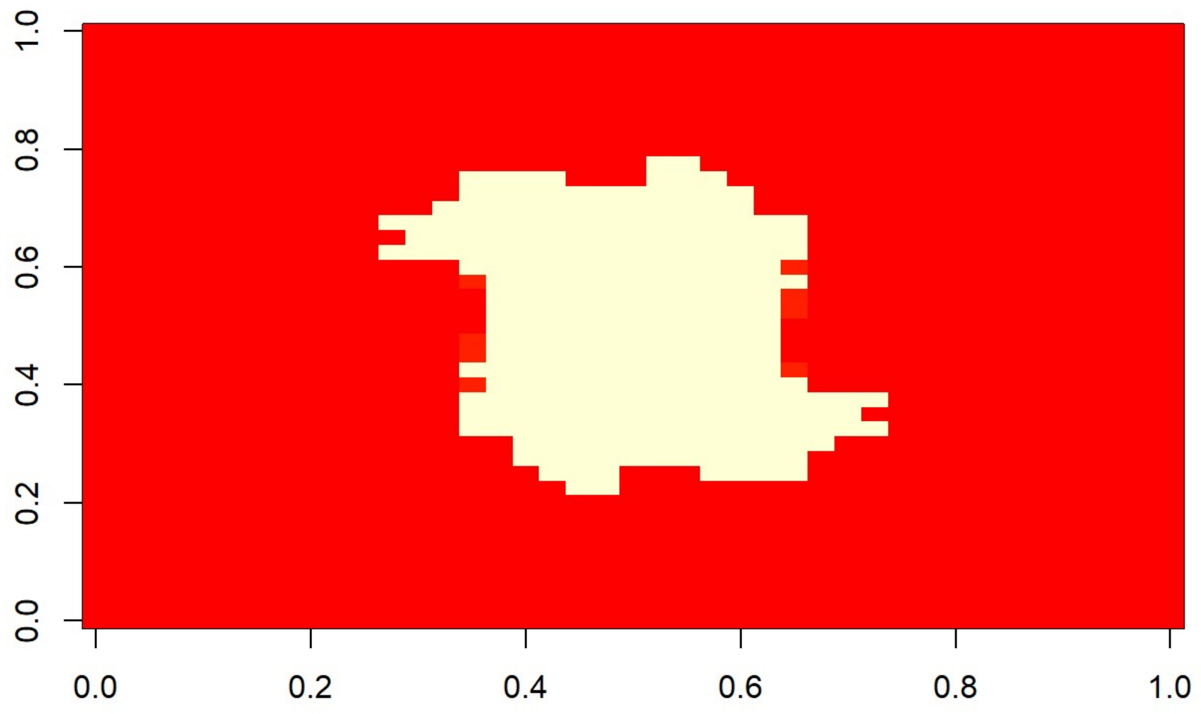
```
image(ply_complexBase(1:10,1:10,Mod,loop=T))#modular of complex nums
```

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js



By counting the escape time on each grid of the Julia function we can render an image of Julia set.

```
ply_Julia = function(re,im,C,n=100){
  ply_complexBase(re,im,dynam_escapeStep,itfunc=dynam_update_Julia,takein=C,n=n)
}
image(ply_Julia(seq(from=-2,to=2,by=.1),seq(from=-2,to=2,by=.1),
    complex(real = .3,imaginary = .25),n=100))
```



Probability models

This part is not covered so far, as R already has lots of probability functionalities. Fitting time series with ply_acf_lag. Prediction will can be done with dynamic_update.

Perspective developments:

Functionality

- condition functions shall be enabled in the next version to run for functions with complicated domains.
- more functions to be added:
- multivariable optimization models
- object systems
- apply multiple lag data and multiple fit models on time series
- enable transfer between ts, data frame and the output of lm

Speed

- predefined parallel computing within functions(with doparallel)
- test speed with JIT compiler for R

Compatibility

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js

Compatibilities of the dynam_record function is to be tested with other packages of R.

Acknowledgement

Special thanks to:

Prof. M. Meerschaert for writing the inspiring book and suggestions on plotting and functional programming.

M. Liu for checking the results for the dynamic model analysis.

Patrick, our tutor during STAR MATH 199 in UC for patiently explaining math concepts for us and answering my endless questions.

All my teachers from BHS for the suggestions on stats and projects.

And my family, all my friends for the toloerance of my stubborness on math.

Loading [MathJax]/jax/output/HTML-CSS/fonts/TeX/fontdata.js