Package 'GrassmannOptim'

June 22, 2022

Type Package

Title Grassmann Manifold Optimization

Version 2.0.1

Date 2013-12-01

Author Kofi Placid Adragni and Seongho Wu

Maintainer Kofi Placid Adragni <kofi@umbc.edu>

Depends Matrix

Description Optimizing a function F(U), where U is a semi-
orthogonal matrix and F is invariant under an orthogonal transformation of U.

URL https://userpages.umbc.edu/~kofi/GrassmannOptim/

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2022-06-22 05:53:19 UTC

R topics documented:

GrassmannOptim 1

Index **7**

`GrassmannOptim` *Grassmann Manifold Optimization*

Description

Maximizing a function `F(U)`, where `U` is a semi-orthogonal matrix and the function is invariant under an orthogonal transformation of `U`. An explicit expression of the gradient is not required and the hessian is not used. It includes a global search option using simulated annealing.

Usage

```
GrassmannOptim(objfun, W, sim_anneal = FALSE, temp_init = 20,
cooling_rate = 2, max_iter_sa = 100, eps_conv = 1e-05,
max_iter = 100, eps_grad = 1e-05, eps_f = .Machine$double.eps,
verbose = FALSE)
```

Arguments

objfun	a required R function that evaluates value and possibly gradient of the function to be maximized. It returns a list of components in which the component value is required whereas gradient is optional. When gradient is not provided, an approximation is used by default. The parameter of objfun is W that is a list of components described next.
W	a list object of arguments to be passed to objfun. It contains all arguments required to compute the objective function and eventually the gradient. It has a required component that is the dimension of the matrix U as dim=c(d, p) where d is the number of columns and p is the number of rows d<p. An initial starting matrix may be provided as a pxp orthogonal matrix Qt. The minimal expression of W is list(dim=c(d,p)).
sim_anneal	If TRUE the program searches for global maximum by simulating annealing using stochastic gradients. If FALSE a local maximum is likely to be reached.
temp_init	a positive scalar that is the initial temperature for simulated annealing if sim_anneal is TRUE. The minimum temperature is set to 0.1.
cooling_rate	a positive scalar greater than 1 that controls the cooling process. A new cooling temperature is obtained as the previous divided by the cooling rate.
max_iter_sa	a positive integer specifying the maximum number of iterations to be performed at a fixed temperature before cooling.
eps_conv	a small positive scalar. The program terminates when the norm of the gradient gets smaller or equal to eps_conv.
max_iter	a positive integer specifying the maximum number of iterations to be performed before the program is terminated.
eps_grad	is a small positive scalar. If gradient is not explicitly provided through objfun, eps_grad is used to estimate gradient based on a finite difference.
eps_f	small positive scalar giving the tolerance at which the difference between the current objective function value and the preceding is considered small enough to terminate the program.
verbose	if TRUE, steps are printed. Otherwise, nothing is printed.

Details

The algorithm was adapted from Liu, Srivastava and Gallivan (2004) who discussed the geometry of Grassmann manifolds. See also Edelman, Arias and Smith (1998) for more expositions.

This is a non-linear optimization program. We describe a basic gradient algorithm for Grassmann manifolds.

Let $G_{p,d}$ be the set of all d-dimensional subspaces of R^n. It is a compact, connected manifold of dimension d(p-d). An element of this manifold is a subspace. It can be represented by a basis or by a projection matrix. Here, the computations are carried in terms of the bases.

Let U such that Span$(U) \in G_{p,d}$. We consider an objective function $F(U)$ to be optimized.

Let $D(U)$ be the gradient of the objective function F at the point U. The algorithm starts with an initial value U_i of U. For step size δ in R^1, a single step of the gradient algorithm is

$$Q_{t+1} = \exp(-\delta A) Q_t$$

where $Q_t = [U_t, V_t]$ and V_t is the orthogonal completion of U_t so that Q_t is orthogonal. The matrix A is computed using the directional derivatives of F. The new value of the objective function is $F(U_t)$.

The matrix A is skewed-symmetric and $\exp(-\delta \boldsymbol{A})$ is orthogonal. The algorithm works by rotating the starting orthonormal basis Q_t to a new basis by left multiplication by an orthogonal matrix.

The iterations continues until a stopping criterion is met. Ideally, convergence is met when the norm of the gradient is sufficiently small. But stopping can be set at a fixed number of iterations.

An explicit expression of the gradient may not be provided; finite difference approximations are used instead. However, deriving the gradient expression may pay off in terms of the efficiency and reliability of the algorithm. But a differentiable function F that maps $G_{p,d}$ to R^1 is necessary.

The choice of the initial starting value U_i of U is important. We recommend not to use random start for the optimization to avoid a local maximum. Liu et al. (2004) suggested a simulated annealing method to attain a global optimum.

Value

A list containing the following components

Qt	optimal orthogonal matrix such that Qt[,1:d] maximizes the objective function.
norm_grads	a vector of successive norms of the directional derivative throughout all iterations. The last scalar is the norm of the gradient at the optimal Qt.
fvalues	a vector of successive values of the objective function throughout all iterations. The last scalar is the value of the objective function at the optimal Qt.
converged	if TRUE, the final iterate was considered optimal by the specified termination criteria.

Warning

This program may search for a global maximizer using a simulated annealing stochastic gradient. The choice of the initial temperature, cooling rate and also of the maximum allowable number of iterations within the simulated annealing process affect the success of reaching that global maximum.

Note

This program uses the objective function objfun provided by the user. An expression of the objective function needs to follow the format illustrated in the example.

Author(s)

Kofi Placid Adragni <kofi@umbc.edu> and Seongho Wu

References

Liu, X.; Srivastava, A,; Gallivan, K. (2004) Optimal linear representations of images for object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol 26, No. 5, pp 662-666

Edelman, A.; Arias, T. A.; Smith, S. T. (1998) The Geometry of Algorithms with Orthogonality Constraints. SIAM J. Matrix Anal. Appl. Vol. 20, No. 2, pp 303-353

See Also

nlm, nlminb, optim, optimize, constrOptim for other optimization functions.

Examples

```
objfun <- function(W){value <- f(W); gradient <- Grad(W);
return(list(value=value, gradient=gradient))}

f <- function(W){d <- W$dim[1]; Y<-matrix(W$Qt[,1:d], ncol=d);
return(0.5*sum(diag(t(Y)%*%W$A%*%Y)))}

Grad <- function(W){
Qt <- W$Qt; d <- W$dim[1]; p <- nrow(Qt); grad <- matrix (0, p, p);
Y <- matrix(Qt[,1:d], ncol=d); Y0 <- matrix(Qt[,(d+1):p], ncol=(p-d));
return(t(Y) %*% W$A %*% Y0)}

p=5; d=2; set.seed(234);
a <- matrix(rnorm(p**2), ncol=p); A <- t(a)%*%a;

# Exact Solution
W <- list(Qt=eigen(A)$vectors[,1:p], dim=c(d,p), A=A);
ans <- GrassmannOptim(objfun, W, eps_conv=1e-5, verbose=TRUE);
ans$converged

# Random starting matrix
m<-matrix(rnorm(p**2), ncol=p); m<-t(m)%*%m;
W <- list(Qt=eigen(m)$vectors, dim=c(d,p), A=A);
ans <- GrassmannOptim(objfun, W, eps_conv=1e-5, verbose=TRUE);
plot(ans$fvalues)

# Simulated Annealing
W <- list(dim=c(d,p), A=A);
ans <- GrassmannOptim(objfun, W, sim_anneal=TRUE, max_iter_sa=35,
verbose=TRUE);

########

set.seed(13); p=8; nobs=200; d=3; sigma=1.5; sigma0=2;
```

```
rmvnorm <- function (n, mean = rep(0, nrow(sigma)), sigma = diag(length(mean)))
{ # This function generates random numbers from the multivariate normal -
# see library "mvtnorm"
ev <- eigen(sigma, symmetric = TRUE)
     retval <- ev$vectors %*% diag(sqrt(ev$values), length(ev$values)) %*%
t(ev$vectors)
     retval <- matrix(rnorm(n * ncol(sigma)), nrow = n) %*% retval;
     retval <- sweep(retval, 2, mean, "+");
    colnames(retval) <- names(mean);
     retval
}

objfun <- function(W){return(list(value=f(W), gradient=Gradient(W)))}

f <- function(W){
Qt <- W$Qt; d <- W$dim[1]; p <- ncol(Qt); Sigmas <- W$sigmas;
U <- matrix(Qt[,1:d], ncol=d); V <- matrix(Qt[,(d+1):p], ncol=(p-d));
return(-log(det(t(V)%*%Sigmas$S%*%V))-log(det(t(U)%*%Sigmas$S_res%*%U)))}

Gradient <- function(W)
{Qt <- W$Qt; d <- W$dim[1]; p <- ncol(Qt); Sigmas <- W$sigmas;
U <- matrix(Qt[,1:d], ncol=d); V <- matrix(Qt[,(d+1):p], ncol=(p-d));
terme1 <- solve(t(U)%*%Sigmas$S_res%*%U)%*% t(U)%*%Sigmas$S_res%*%V;
terme2 <- t(U)%*%Sigmas$S%*%V%*%solve(t(V)%*%Sigmas$S%*%V);
return(2*(terme1 - terme2))}

y<-array(runif(n=nobs, min=-2, max=2), c(nobs, 1));
fy<-scale(cbind(y, y^2, y^3),TRUE,FALSE);

#Structured error PFC model;
Gamma<-diag(p)[,c(1:3)]; Gamma0<-diag(p)[,-c(1:3)];
Omega <-sigma^2*matrix(0.5, ncol=3, nrow=3); diag(Omega)<-sigma^2;
Delta<- Gamma%*%Omega%*%t(Gamma) + sigma0^2*Gamma0%*%t(Gamma0);

Err <- t(rmvnorm(n=nobs, mean = rep(0, p), sigma = Delta))
beta <- diag(3*c(1, 0.4, 0.4));
X <- t(Gamma%*%beta%*%t(fy) + Err);

Xc <- scale(X, TRUE, FALSE);
P_F <- fy%*%solve(t(fy)%*%fy)%*%t(fy);
S <- t(Xc)%*%Xc/nobs; S_fit <- t(Xc)%*%P_F%*%Xc/nobs; S_res <- S-S_fit;
sigmas <- list(S=S, S_fit=S_fit, S_res=S_res, p=p, nobs=nobs);

# Random starting matrix;
Qt <- svd(matrix(rnorm(p^2), ncol=p))$u;
W <- list(Qt=Qt, dim=c(d, p), sigmas=sigmas)

ans <- GrassmannOptim(objfun, W, eps_conv=1e-4);
ans$converged;
ans$fvalues;
ans$Qt[,1:3];
```

```
# Good starting matrix;
Qt <- svd(S_fit)$u;
W <- list(Qt=Qt, dim=c(d, p), sigmas=sigmas)
ans <- GrassmannOptim(objfun, W, eps_conv=1e-4, verbose=TRUE);
ans$converged;
```

Index

∗ **optimize**
GrassmannOptim, 1
∗ **package**
GrassmannOptim, 1
∗ **programming**
GrassmannOptim, 1

constrOptim, *4*

GrassmannOptim, 1

nlm, *4*
nlminb, *4*

optim, *4*
optimize, *4*