Package 'blowtorch'

July 2, 2014

Title Constrained optimization via stochastic gradient descent.

Description Optimize a function with constraints via a transformation of the Lagrangian.

Version 1.0.1

Author Steven Pollack <steven@pollackphoto.net>

Maintainer Steven Pollack <steven@pollackphoto.net>

Depends R (>= 3.0.2)

Imports ggplot2, grid, foreach, iterators

License GPL-2

LazyData true

NeedsCompilation no

Repository CRAN

Date/Publication 2014-05-09 00:48:37

R topics documented:

addDots *Extend a function's signature to include '...'*

Description

modifies the input function to accept extra arguments, if it doesn't already.

Usage

```
addDots(func, .verbose = FALSE)
```

Arguments

func	the function whose signature is to be modified.
.verbose	logical flag indicating whether warnings should be displayed or not.

Value

a function with the same body as func but whose signature now includes ....

Examples

```
## Not run:
f <- function(x,y) x^2 + y^2

g <- addDots(f)
g

h <- addDots(g, .verbose=TRUE)
## End(Not run)
```

aggregateConstraints *Aggregate constraints.*

Description

Collect constraint functions into a vector.

Usage

```
aggregateConstraints(...)
```

Arguments

...	constraint functions. Must have take a single argument and return a single value.

Value

a function which takes a value `X` and returns a vector with length equal to `length(...)`. The i-th entry of the return value is the i-th constraint function evaluated at `X`.

See Also

Other aggregate functions: `aggregateGradients`; `aggregateHessians`

Examples

```
g1 <- function(X) {
as.numeric(c(1,1,1) %*% X^4 - 1)
}

g2 <- function(X) {
  as.numeric(c(1,1,1) %*% X^2 - 1)
}

G <- aggregateConstraints(g1, g2)
G(c(1,1,1))
```

---

aggregateGradients *Aggregate constraint gradients.*

Description

Collect gradients of constraint functions into a matrix.

Usage

```
aggregateGradients(...)
```

Arguments

`...`	gradients of constraint functions.

Value

a function which takes a value `X` and returns a matrix with with dimensions `c(length(X), length(...))`. The (i,j) entry of the return value is the i-th partial derivative of the j-th constraint, evaluated at `X`.

See Also

Other aggregate functions: `aggregateConstraints`; `aggregateHessians`

Examples

```
Dg1 <- function(X){
  4 * X^3
}

Dg2 <- function(X){
  2 * X
}

DG <- aggregateGradients(Dg1, Dg2)
DG(c(1,1,1))
```

`aggregateHessians` *Aggregate constraint Hessians.*

Description

Collect Hessian matrices of constraint functions into a 3D array.

Usage

```
aggregateHessians(...)
```

Arguments

`...`	Hessians of constraint functions.

Value

a function which takes a value `X` and returns a 3D array with with dimensions `c(length(X), length(X), length(...))`. The (i,j,k) entry of the return value is the ij-th partial derivative of the k-th constraint, evaluated at `X`.

See Also

Other aggregate functions: `aggregateConstraints`; `aggregateGradients`

Examples

```
Hg1 <- function(X){
matrix(12 * c(X[1], 0, 0, 0, X[2], 0, 0, 0, X[3])^2, nrow=3, byrow=TRUE)
}

Hg2 <- function(X){
  matrix(c(2, 0, 0, 0, 2, 0, 0, 0, 2), nrow=3, byrow=TRUE)
}

HG <- aggregateHessians(Hg1, Hg2)
HG(c(0,0,0))
```

`blowtorch` *blowtorch*

Description

blowtorch

`buildDh` *Assemble the gradient of h*

Description

Assemble the gradient of h from the second and first derivatives of the objective and constraint functions.

Usage

```
buildDh(G, Df, DG, Hf, HG)
```

Arguments

`G`	a vector valued constraint function which takes input `X` and should yield a vector of 0's when all constraints are satisfied. See `aggregateConstraints`.
`Df`	a vector valued function representing the gradient of the objective function, `f`. `Df` should take `(X,lambda)` as primary arguments, though it must take `(X, lambda, data)` as arguments if `SGD` is to be used.
`DG`	a matrix valued function representing the matrix of gradients of constraint functions. See `aggregateGradients`.
`Hf`	the Hessian matrix of the objective function, `f`. `Hf` should take `(X,lambda)` as primary arguments, if `gradientDescent` is being used to optimize `f`. If `SGD` is being called, it must take `(X, lambda, data)` as arguments.
`HG`	the (aggregate) Hessian of the constrain function `G`. This object must have the same form as the output to `aggregateHessians`.

Details

The definition for `h` and `Dh` can be found in `inst/final_project/final_paper/final_paper.pdf`.

Value

a vector-valued function `Dh(X, lambda, data)` which is the gradient of `h` as created in `build_h`.

See Also

Other blowtorch wrappers: `build_h`

Examples

```
Df <- function(X){
x <- X[1]; y <- X[2]; z <- X[3]
c(1 + 2*x + y + z, 1 + x + 2*y + z, 1 + x + y + 2*z)
}

Hf <- function(X){
  matrix(c(2,1,1,1,2,1,1,1,2), nrow=3, byrow=TRUE)
}

g1 <- function(X) {
  as.numeric(c(1,1,1) %*% X^4 - 1)
}

g2 <- function(X) {
  as.numeric(c(1,1,1) %*% X^2 - 1)
}

Dg1 <- function(X){
  4 * X^3
}

Dg2 <- function(X){
  2 * X
}

Hg1 <- function(X){
  matrix(12 * c(X[1], 0, 0, 0, X[2], 0, 0, 0, X[3])^2, nrow=3, byrow=TRUE)
}

Hg2 <- function(X){
  matrix(c(2, 0, 0, 0, 2, 0, 0, 0, 2), nrow=3, byrow=TRUE)
}

Dh <- buildDh(G=aggregateConstraints(g1, g2),
              Df=Df,
              DG=aggregateGradients(Dg1, Dg2),
              Hf=Hf,
              HG=aggregateHessians(Hg1, Hg2))

Dh(c(1,1,1), c(1,1))
```

---

build_h *Assemble h*

Description

Build h from the first derivatives of the object and constraint functions.

Usage

```
build_h(G, Df, DG)
```

Arguments

G	a vector valued constraint function which takes input X and should yield a vector of 0's when all constraints are satisfied. See aggregateConstraints.
Df	a vector valued function representing the gradient of the objective function, f. Df should take (X,lambda) as primary arguments, though it must take (X, lambda, data) as arguments if SGD is to be used.
DG	a matrix valued function representing the matrix of gradients of constraint functions. See aggregateGradients.

Details

The definition for h and Dh can be found in inst/final_project/final_paper/final_paper.pdf.

Value

a scalar-valued function h(X, lambda, data) which is half the squared euclidean norm of the gradient of the Lagrangian created by f and G.

See Also

Other blowtorch wrappers: buildDh

Examples

```
Df <- function(X){
x <- X[1]; y <- X[2]; z <- X[3]
c(1 + 2*x + y + z, 1 + x + 2*y + z, 1 + x + y + 2*z)
}

g1 <- function(X) {
  as.numeric(c(1,1,1) %*% X^4 - 1)
}

g2 <- function(X) {
  as.numeric(c(1,1,1) %*% X^2 - 1)
}

Dg1 <- function(X){
  4 * X^3
}

Dg2 <- function(X){
  2 * X
}

h <- build_h(G=aggregateConstraints(g1, g2),
```

```
            Df=Df,
            DG=aggregateGradients(Dg1, Dg2))

h(c(1,1,1), c(1,1))
```

`gradientDescent` *Gradient Descent*

Description

Minimze a function `h(X,lambda)` with its gradient, `Dh(X,lambda)` via gradient descent.

Usage

```
gradientDescent(X0, lambda0, alpha, h, Dh, convTol = 1e-10,
  relTol = convTol, iterLimit = 100, trace = 0, plotTrace = trace,
  rngSeed = 1234, objFun = NULL, ...)
```

Arguments

`X0`	initial value for X
`lambda0`	initial value for lambda
`alpha`	learning rate – either a constant of a vector whose size is equal to `c(X0, lambda0)`.
`h`	function whose signature is `(X,lambda, data)` which is vectorized over `data`. `h` is assumed to be non-negative, and SGD will attempt to minimize it.
`Dh`	gradient of `h` as a function of `X`, `lambda`, and `data`. Also assumed to be vectorized over `data`.
`convTol`	positive value which signals `h`'s convergence.
`relTol`	positive value. If all parameter estimates and `h` do not change more than `relTol` in the course of one iteration, SGD will stop.
`iterLimit`	the maximum number of iterations.
`trace`	non-negative integer. At every `trace` iteration, `SGD` will output the current parameter estimates to stdout. A value of zero indicates no desired output.
`plotTrace`	non-negative integer. At every `plotTrace` iteration, `SGD` will graphically output the parameter estimates up to that iteration. A value of zero indicates no desired output
`rngSeed`	the random number generation seed value.
`objFun`	the objective function in the Lagrangian to be optimized.
`...`	layout arguments to be passed to `multiplot`.

Details

Follows the gradient method described in `inst/final_project/final_paper/final_paper.pdf`.

Value

an 'SGD' object with methods print.SGD and plot.SGD. An SGD object is just a glorified list with the following entries

X_final	the last iteration's X values
lambda_final	the last iteration's lambda values
X_initial	the initial X value
lambda_initial	the initia lambda value
X	the matrix whose i-th column contains the i-th iteration's X values
lambda	the matrix whose i-th column contains the i-th iteration's lambda values
delta	the vector whose i-th entry contains the value of h at the i-th iteration
alpha	the given learning rate
beta	the given annealing rate
batchSize	the given batchSize
iter	the number of iterations taken
convTol	the given convergence tolerance
objFun	the original objective function
converged	a boolean indicating if h was driven below convTol
aboveTol	a boolean indicating if any X or lambda values stabilized below relTol

See Also

Other gradient methods: SGD, print.SGD

---

multiplot *Plot multiple ggplot objects.*

Description

Plot multiple ggplot objects on one layer.

Usage

```
multiplot(..., plotlist = NULL, cols = 1, layout = NULL)
```

Arguments

...	ggplot objects
plotlist	a list of ggplot objets
cols	number of columns in layout
layout	a matrix specifying the layout. If present, 'cols' is ignored.

Details

If the layout is something like `matrix(c(1,2,3,3), nrow=2, byrow=TRUE)`, then plot 1 will go in the upper left, 2 will go in the upper right, and 3 will go all the way across the bottom. Taken from: http://www.cookbook-r.com/Graphs/Multiple_graphs_on_one_page_(ggplot2)/

Examples

```
library(ggplot2)
# This example uses the ChickWeight dataset, which comes with ggplot2
p1 <- ggplot(ChickWeight, aes(x=Time, y=weight, colour=Diet, group=Chick)) +
  geom_line() +
  ggtitle("Growth curve for individual chicks")

# Second plot
p2 <- ggplot(ChickWeight, aes(x=Time, y=weight, colour=Diet)) +
  geom_point(alpha=.3) +
  geom_smooth(alpha=.2, size=1) +
  ggtitle("Fitted growth curve per diet")

# Third plot
p3 <- ggplot(subset(ChickWeight, Time==21), aes(x=weight, colour=Diet)) +
  geom_density() +
  ggtitle("Final weight, by diet")

# Fourth plot
p4 <- ggplot(subset(ChickWeight, Time==21), aes(x=weight, fill=Diet)) +
  geom_histogram(colour="black", binwidth=50) +
  facet_grid(Diet ~ .) +
  ggtitle("Final weight, by diet") +
  theme(legend.position="none")  # No legend (redundant in this graph)

multiplot(p1, p2, p3, p4, cols=2)
```

`plot.SGD` *Plot method for SGD object*

Description

plot the value of `h` across iterations.

Usage

```
## S3 method for class 'SGD'
plot(x, y = NULL, ...)
```

Arguments

x	an object of class SGD
y	does nothing
...	does nothing

SGD *Stochastic Gradient Descent*

Description

Minimize a function h via stochastic gradient descent.

Usage

```
SGD(X0, lambda0, alpha, h, Dh, data, convTol = 1e-10, relTol = convTol,
  iterLimit = 100, beta = 0.5, batchSize = 1, trace = 0,
  plotTrace = trace, rngSeed = 1234, objFun = NULL, ...)

## S3 method for class 'SGD'
print(x, ...)
```

Arguments

X0	initial value for X
lambda0	initial value for lambda
alpha	learning rate – either a constant of a vector whose size is equal to c(X0, lambda0).
h	function whose signature is (X,lambda, data) which is vectorized over data. h is assumed to be non-negative, and SGD will attempt to minimize it.
Dh	gradient of h as a function of X, lambda, and data. Also assumed to be vectorized over data.
convTol	positive value which signals h's convergence.
relTol	positive value. If all parameter estimates and h do not change more than relTol in the course of one iteration, SGD will stop.
iterLimit	the maximum number of iterations.
trace	non-negative integer. At every trace iteration, SGD will output the current parameter estimates to stdout. A value of zero indicates no desired output.
plotTrace	non-negative integer. At every plotTrace iteration, SGD will graphically output the parameter estimates up to that iteration. A value of zero indicates no desired output
rngSeed	the random number generation seed value.
objFun	the objective function in the Lagrangian to be optimized.
...	layout arguments to be passed to multiplot.

data	the data set whose rows (the number of which is determined by batchSize) are to be fed to h and Dh.
beta	annealing constant for step size. See details.
batchSize	number of observations to feed to h and Dh at each iteration. When batchSize is equal to nrow(data) SGD will perform exactly like gradientDescent.
x	an object of class SGD.

Details

Follows the gradient method described in inst/final_project/final_paper/final_paper.pdf.

Value

an 'SGD' object with methods print.SGD and plot.SGD. An SGD object is just a glorified list with the following entries

X_final	the last iteration's X values
lambda_final	the last iteration's lambda values
X_initial	the initial X value
lambda_initial	the initia lambda value
X	the matrix whose i-th column contains the i-th iteration's X values
lambda	the matrix whose i-th column contains the i-th iteration's lambda values
delta	the vector whose i-th entry contains the value of h at the i-th iteration
alpha	the given learning rate
beta	the given annealing rate
batchSize	the given batchSize
iter	the number of iterations taken
convTol	the given convergence tolerance
objFun	the original objective function
converged	a boolean indicating if h was driven below convTol
aboveTol	a boolean indicating if any X or lambda values stabilized below relTol

See Also

Other gradient methods: gradientDescent

Index