

# Package ‘support’

July 15, 2019

**Type** Package

**Title** Support Points

**Version** 0.1.4

**Author** Simon Mak

**Maintainer** Simon Mak <sm769@duke.edu>

**Description** The 'support' package provides the functions `sp()` and `sp_seq()` for computing the support points in Mak and Joseph (2018) <DOI:10.1214/17-AOS1629>. Support points can be used as a representative sample of a desired distribution, or a representative reduction of a big dataset (e.g., an “optimal” thinning of Markov-chain Monte Carlo sample chains). This work was supported by USARO grant W911NF-14-1-0024 and NSF DMS grant 1712642.

**License** GPL (>= 2)

**LazyData** TRUE

**Imports** Rcpp (>= 0.12.4), randtoolbox, MHadaptive

**LinkingTo** Rcpp, RcppArmadillo, BH

**RoxygenNote** 6.1.1

**NeedsCompilation** yes

**Repository** CRAN

**Date/Publication** 2019-07-15 13:40:03 UTC

## R topics documented:

support-package . . . . .	2
e_dist . . . . .	2
sp . . . . .	4
sp_seq . . . . .	8

<b>Index</b>	<b>12</b>
--------------	-----------

---

 support-package

*Support Points*


---

### Description

The 'support' package provides functions for computing support points.

### Details

Package: support  
 Type: Package  
 Version: 0.1.4  
 Date: 2019-07-15  
 License: GPL (>= 2)

The 'support' package provides the functions `sp()` and `sp_seq()` for computing the support points in Mak and Joseph (2018) <DOI:10.1214/17-AOS1629>. Support points can be used as a representative sample of a desired distribution, or a representative reduction of a big dataset (e.g., an "optimal" thinning of Markov-chain Monte Carlo sample chains). This work was supported by USARO grant W911NF-14-1-0024 and NSF DMS grant 1712642.

### Author(s)

Simon Mak

Maintainer: Simon Mak <sm769@duke.edu>

### References

Mak, S. and Joseph, V. R. (2018). Support points. *Annals of Statistics*, 46(6A):2562-2592.

---

 e\_dist

*Computes the energy distance of a point set*


---

### Description

`e_dist` computes the energy distance between points  $D$  and a target distribution (or big dataset)  $F$ . The cross-term  $E[\|X - X'\|]$ ,  $X, X' \sim F$  is *NOT* computed in `e_dist` for computational efficiency, since this is not needed for optimizing  $D$ . The target distribution or big dataset can be set using `dist.str` or `dist.samp`, respectively.

### Usage

```
e_dist(D, dist.str=NA, dist.param=vector("list",ncol(D)),
       nsamp=1e6, dist.samp=NA)
```

**Arguments**

D	An $n \times p$ point set.
dist.str	A $p$ -length string vector for target distribution (assuming independence). Current options include uniform, normal, exponential, gamma, lognormal, student-t, weibull, cauchy and beta. Exactly one of dist.str or dist.samp should be NA.
dist.param	A $p$ -length list for desired distribution parameters in dist.str. The following parameters are supported: <ul style="list-style-type: none"> <li>• <b>Uniform:</b> Minimum, maximum;</li> <li>• <b>Normal:</b> Mean, standard deviation;</li> <li>• <b>Exponential:</b> Rate parameter;</li> <li>• <b>Gamma:</b> Shape parameter, scale parameter;</li> <li>• <b>Lognormal:</b> Log-mean, Log-standard deviation;</li> <li>• <b>Student-t:</b> Degree-of-freedom;</li> <li>• <b>Weibull:</b> Shape parameter, scale parameter;</li> <li>• <b>Cauchy:</b> Location parameter, scale parameter;</li> <li>• <b>Beta:</b> Shape parameter 1, shape parameter 2.</li> </ul>
nsamp	Number of samples to draw from dist.str for comparison.
dist.samp	An $N \times p$ matrix for the target big dataset (e.g., MCMC chain). Exactly one of dist.str or dist.samp should be NA.

**References**

Szekely, G. J. and Rizzo, M. L. (2013). Energy statistics: A class of statistics based on distances. *Journal of Statistical Planning and Inference*, 143(8):1249-1272.

**Examples**

```
## Not run:
#####
# Generate 25 SPs for the 2-d i.i.d. N(0,1) distribution
#####
n <- 25 #number of points
p <- 2 #dimension
D <- sp(n,p,dist.str=rep("normal",p))
Drnd <- matrix(rnorm(n*p),ncol=p)
e_dist(D$sp,dist.str=rep("normal",p)) #smaller
e_dist(Drnd,dist.str=rep("normal",p))

#####
# Support points for big data reduction: Franke's function
#####
library(MHadaptive)

#Use modified Franke's function as posterior
franke2d <- function(xx){
  if ((xx[1]>1)||xx[1]<0)||xx[2]>1)||xx[2]<0){
```

```

    return(-Inf)
  }
  else{
    x1 <- xx[1]
    x2 <- xx[2]

    term1 <- 0.75 * exp(-(9*x1-2)^2/4 - (9*x2-2)^2/4)
    term2 <- 0.75 * exp(-(9*x1+1)^2/49 - (9*x2+1)/10)
    term3 <- 0.5 * exp(-(9*x1-7)^2/4 - (9*x2-3)^2/4)
    term4 <- -0.2 * exp(-(9*x1-4)^2 - (9*x2-7)^2)

    y <- term1 + term2 + term3 + term4
    return(2*log(y))
  }
}

#Generate MCMC samples
li_func <- franke2d #Desired log-posterior
ini <- c(0.5,0.5) #Initial point for MCMC
NN <- 1e5 #Number of MCMC samples desired
burnin <- NN/2 #Number of burn-in runs
mcmc_r <- Metro_Hastings(li_func, pars=ini, prop_sigma=0.05*diag(2),
                        iterations=NN, burn_in=burnin)

#Generate ncur SPs
ncur <- 50
D <- sp(ncur,2,dist.samp=mcmc_r$trace)$sp
Drnd <- mcmc_r$trace[sample(1:nrow(mcmc_r$trace),n,F),]
e_dist(D,dist.samp=mcmc_r$trace) #smaller
e_dist(Drnd,dist.samp=mcmc_r$trace)

## End(Not run)

```

---

sp

---

*Computing support points using difference-of-convex programming*


---

## Description

sp is the main function for computing the support points in Mak and Joseph (2018). Current options include support points on standard distributions (specified via `dist.str`) or support points for reducing big data (specified via `dist.samp`). For big data reduction, weights on each data point can be specified via `wts`.

## Usage

```

sp(n, p, ini=NA,
   dist.str=NA, dist.param=vector("list",p),
   dist.samp=NA, scale.flg=T, wts=NA, bd=NA,
   num.subsamp=ifelse(any(is.na(dist.samp)),

```

```
max(10000,10*n),min(10000,nrow(dist.samp))),
iter.max=max(250,iter.min), iter.min=50,
tol=1e-10, par.flg=TRUE, n0=n*p)
```

### Arguments

n	Number of support points.
p	Dimension of sample space.
ini	An $n \times p$ matrix for initialization.
dist.str	A $p$ -length string vector for desired distribution (assuming independence). Current options include uniform, normal, exponential, gamma, lognormal, student-t, weibull, cauchy and beta. Exactly one of <code>dist.str</code> or <code>dist.samp</code> should be NA.
dist.param	A $p$ -length list for desired distribution parameters in <code>dist.str</code> . The following parameters are supported: <ul style="list-style-type: none"> <li>• <b>Uniform:</b> Minimum, maximum;</li> <li>• <b>Normal:</b> Mean, standard deviation;</li> <li>• <b>Exponential:</b> Rate parameter;</li> <li>• <b>Gamma:</b> Shape parameter, scale parameter;</li> <li>• <b>Lognormal:</b> Log-mean, Log-standard deviation;</li> <li>• <b>Student-t:</b> Degree-of-freedom;</li> <li>• <b>Weibull:</b> Shape parameter, scale parameter;</li> <li>• <b>Cauchy:</b> Location parameter, scale parameter;</li> <li>• <b>Beta:</b> Shape parameter 1, shape parameter 2.</li> </ul>
dist.samp	An $N \times p$ matrix for the big dataset (e.g., MCMC chain) to be reduced. Exactly one of <code>dist.str</code> or <code>dist.samp</code> should be NA.
scale.flg	Should the big data <code>dist.samp</code> be normalized to unit variance before processing?
wts	Weights on each data point in <code>dist.samp</code> . Uniform weights are assigned if NA.
bd	A $p \times 2$ matrix for the lower and upper bounds of each variable.
num.subsamp	Batch size for resampling. For distributions, the default is $\max(10000, 10 \times n)$ . For data reduction, the default is $\min(10000, \text{nrow}(\text{dist.samp}))$ .
iter.max	Maximum iterations for optimization.
iter.min	Minimum iterations for optimization.
tol	Error tolerance for optimization.
par.flg	Should parallelization be used?
n0	Momentum parameter for optimization.

### Value

sp	An $n \times p$ matrix for support points.
ini	An $n \times p$ matrix for initial points.

## References

Mak, S. and Joseph, V. R. (2018). Support points. *Annals of Statistics*, 46(6A):2562-2592.

## Examples

```
## Not run:
#####
# Support points on distributions
#####

#Generate 25 SPs for the 2-d i.i.d. N(0,1) distribution
n <- 25 #number of points
p <- 2 #dimension
D <- sp(n,p,dist.str=rep("normal",p))

x1 <- seq(-3.5,3.5,length.out=100) #Plot contours
x2 <- seq(-3.5,3.5,length.out=100)
z <- exp(-outer(x1^2,x2^2,FUN="+")/2)
contour.default(x=x1,y=x2,z=z,drawlabels=FALSE,nlevels=10)
points(D$sp,pch=16,cex=1.25,col="red")

#####
# Generate 50 SPs for the 2-d i.i.d. Beta(2,4) distribution
#####
n <- 50
p <- 2
dist.param <- vector("list",p)
for (l in 1:p){
  dist.param[[l]] <- c(2,4)
}
D <- sp(n,p,dist.str=rep("beta",p),dist.param=dist.param)

x1 <- seq(0,1,length.out=100) #Plot contours
x2 <- seq(0,1,length.out=100)
z <- matrix(NA,nrow=100,ncol=100)
for (i in 1:100){
  for (j in 1:100){
    z[i,j] <- dbeta(x1[i],2,4) * dbeta(x2[j],2,4)
  }
}
contour.default(x=x1,y=x2,z=z,drawlabels=FALSE,nlevels=10 )
points(D$sp,pch=16,cex=1.25,col="red")

#####
# Generate 100 SPs for the 3-d i.i.d. Exp(1) distribution
#####
n <- 100
p <- 3
D <- sp(n,p,dist.str=rep("exponential",p))
pairs(D$sp,xlim=c(0,5),ylim=c(0,5),pch=16)

#####
```

```

# Support points for big data reduction: Franke's function
#####
library(MHadaptive)

#Use modified Franke's function as posterior
franke2d <- function(xx){
  if ((xx[1]>1)||xx[1]<0)||xx[2]>1)||xx[2]<0){
    return(-Inf)
  }
  else{
    x1 <- xx[1]
    x2 <- xx[2]

    term1 <- 0.75 * exp(-(9*x1-2)^2/4 - (9*x2-2)^2/4)
    term2 <- 0.75 * exp(-(9*x1+1)^2/49 - (9*x2+1)/10)
    term3 <- 0.5 * exp(-(9*x1-7)^2/4 - (9*x2-3)^2/4)
    term4 <- -0.2 * exp(-(9*x1-4)^2 - (9*x2-7)^2)

    y <- term1 + term2 + term3 + term4
    return(2*log(y))
  }
}

#Generate MCMC samples
li_func <- franke2d #Desired log-posterior
ini <- c(0.5,0.5) #Initial point for MCMC
NN <- 1e5 #Number of MCMC samples desired
burnin <- NN/2 #Number of burn-in runs
mcmc_r <- Metro_Hastings(li_func, pars=ini, prop_sigma=0.05*diag(2),
  iterations=NN, burn_in=burnin)

#Compute n SPs
n <- 100
D <- sp(n,2,dist.samp=mcmc_r$trace)

#Plot SPs
par(mfrow=c(1,2))
x1 <- seq(0,1,length.out=100) #contours
x2 <- seq(0,1,length.out=100)
z <- matrix(NA,nrow=100,ncol=100)
for (i in 1:100){
  for (j in 1:100){
    z[i,j] <- franke2d(c(x1[i],x2[j]))
  }
}
plot(mcmc_r$trace,pch=4,col="gray",cex=0.75,
  xlab="",ylab="",xlim=c(0,1),ylim=c(0,1)) #big data
points(D$sp,pch=16,cex=1.25,col="red")
contour.default(x=x1,y=x2,z=z,drawlabels=TRUE,nlevels=10) #contour
points(D$sp,pch=16,cex=1.25,col="red")

#####
# Support points for big data: Rosenbrock distribution

```

```
#####

#Use Rosenbrock function as posterior
rosen2d <- function(x) {
  B <- 0.03
  -x[1]^2/200 - 1/2*(x[2]+B*x[1]^2-100*B)^2
}

#Generate MCMC samples
li_func <- rosen2d #Desired log-posterior
ini <- c(0,1) #Initial point for MCMC
NN <- 1e5 #Number of MCMC samples desired
burnin <- NN/2 #Number of burn-in runs
mcmc_r <- Metro_Hastings(li_func, pars=ini, prop_sigma=0.25*diag(2),
  iterations=NN, burn_in=burnin)

#Compute n SPs
n <- 100
D <- sp(n,2,dist.samp=mcmc_r$trace)

#Plot SPs
x1 <- seq(-25,25,length.out=100) #contours
x2 <- seq(-15,6,length.out=100)
z <- matrix(NA,nrow=100,ncol=100)
for (i in 1:100){
  for (j in 1:100){
    z[i,j] <- rosen2d(c(x1[i],x2[j]))
  }
}
plot(mcmc_r$trace,pch=4,col="gray",cex=0.75,
  xlab="",ylab="",xlim=c(-25,25),ylim=c(-15,6)) #big data
points(D$sp,pch=16,cex=1.25,col="red")
contour.default(x=x1,y=x2,z=z,drawlabels=TRUE,nlevels=10) #contour
points(D$sp,pch=16,cex=1.25,col="red")

## End(Not run)
```

---

sp\_seq

*Computing (batch) sequential support points using difference-of-convex programming*

---

### Description

sp\_seq computes (batch) sequential support points to add onto a current point set D. Current options include sequential support points on standard distributions (specified via `dist.str`) or sequential support points for reducing big data (specified via `dist.samp`).

### Usage

```
sp_seq(D, nseq, ini=NA, num.rep=1,
```



```

dist.str=NA, dist.param=vector("list",p),
dist.samp=NA, scale.flg=T, bd=NA,
num.subsamp=ifelse(any(is.na(dist.samp)),
max(10000,10*(nseq+nrow(D))),
min(10000,nrow(dist.samp))),
iter.max=max(200,iter.min), iter.min=50,
tol=1e-10, par.flg=TRUE)

```

### Arguments

D	An $n \times p$ matrix for the current point set.
nseq	Number of support points to add to D.
ini	An $nseq \times p$ matrix for initialization.
num.rep	Number of random restarts for optimization.
dist.str	A $p$ -length string vector for desired distribution (assuming independence). Current options include uniform, normal, exponential, gamma, lognormal, student-t, weibull, cauchy and beta. Exactly one of <code>dist.str</code> or <code>dist.samp</code> should be NA.
dist.param	A $p$ -length list for desired distribution parameters in <code>dist.str</code> . The following parameters are supported: <ul style="list-style-type: none"> <li>• <b>Uniform:</b> Minimum, maximum;</li> <li>• <b>Normal:</b> Mean, standard deviation;</li> <li>• <b>Exponential:</b> Rate parameter;</li> <li>• <b>Gamma:</b> Shape parameter, scale parameter;</li> <li>• <b>Lognormal:</b> Log-mean, Log-standard deviation;</li> <li>• <b>Student-t:</b> Degree-of-freedom;</li> <li>• <b>Weibull:</b> Shape parameter, scale parameter;</li> <li>• <b>Cauchy:</b> Location parameter, scale parameter;</li> <li>• <b>Beta:</b> Shape parameter 1, shape parameter 2.</li> </ul>
dist.samp	An $N \times p$ matrix for the big dataset (e.g., MCMC chain) to be reduced. Exactly one of <code>dist.str</code> or <code>dist.samp</code> should be NA.
scale.flg	Should the big data <code>dist.samp</code> be normalized to unit variance before processing?
bd	A $p \times 2$ matrix for the lower and upper bounds of each variable.
num.subsamp	Batch size for resampling. For distributions, the default is $\max(10000, 10 \cdot n)$ . For data reduction, the default is $\min(10000, nrow(dist.samp))$ .
iter.max	Maximum iterations for optimization.
iter.min	Minimum iterations for optimization.
tol	Error tolerance for optimization.
par.flg	Should parallelization be used?

### Value

D	An $n \times p$ matrix for the current point set.
seq	An $nseq \times p$ matrix for the additional nseq support points.

## References

Mak, S. and Joseph, V. R. (2018). Support points. *Annals of Statistics*, 46(6A):2562-2592.

## Examples

```
## Support points on standard distributions

## Not run:
#####
# Generate 50 SPs for the 2-d i.i.d. N(0,1) distribution
#####
ncur <- 50
cur.sp <- sp(ncur,2,dist.str=rep("normal",2))$sp

#Add 50 sequential SPs
nseq <- 50
seq.sp <- sp_seq(cur.sp,nseq,dist.str=rep("normal",2))$seq

x1 <- seq(-3.5,3.5,length.out=100) #Plot contours
x2 <- seq(-3.5,3.5,length.out=100)
z <- exp(-outer(x1^2,x2^2,FUN="+")/2)
contour.default(x=x1,y=x2,z=z,drawlabels=FALSE,nlevels=10)
points(cur.sp,pch=4,cex=1.25,col="black",lwd=2) # (current in black)
points(seq.sp,pch=16,cex=1.25,col="red") # (new SPs in red)

#####
# Support points for big data reduction: Franke distribution
#####
library(MHadaptive)

#Use modified Franke's function as posterior
franke2d <- function(xx){
  if ((xx[1]>1)||xx[1]<0)||xx[2]>1)||xx[2]<0){
    return(-Inf)
  }
  else{
    x1 <- xx[1]
    x2 <- xx[2]

    term1 <- 0.75 * exp(-(9*x1-2)^2/4 - (9*x2-2)^2/4)
    term2 <- 0.75 * exp(-(9*x1+1)^2/49 - (9*x2+1)/10)
    term3 <- 0.5 * exp(-(9*x1-7)^2/4 - (9*x2-3)^2/4)
    term4 <- -0.2 * exp(-(9*x1-4)^2 - (9*x2-7)^2)

    y <- term1 + term2 + term3 + term4
    return(2*log(y))
  }
}

#Generate MCMC samples
li_func <- franke2d #Desired log-posterior
ini <- c(0.5,0.5) #Initial point for MCMC
```

```

NN <- 1e5 #Number of MCMC samples desired
burnin <- NN/2 #Number of burn-in runs
mcmc_r <- Metro_Hastings(li_func, pars=ini, prop_sigma=0.05*diag(2),
                        iterations=NN, burn_in=burnin)

#Generate ncur SPs
ncur <- 50
cur.sp <- sp(ncur,2,dist.samp=mcmc_r$trace)$sp

#Add nseq sequential SPs
nseq <- 50
seq.sp <- sp_seq(cur.sp,nseq,dist.samp=mcmc_r$trace)$seq

#Plot SPs
par(mfrow=c(1,2))
x1 <- seq(0,1,length.out=100) #contours
x2 <- seq(0,1,length.out=100)
z <- matrix(NA,nrow=100,ncol=100)
for (i in 1:100){
  for (j in 1:100){
    z[i,j] <- franke2d(c(x1[i],x2[j]))
  }
}
plot(mcmc_r$trace,pch=4,col="gray",cex=0.75,
     xlab="",ylab="",xlim=c(0,1),ylim=c(0,1)) #big data
points(cur.sp,pch=4,cex=1.25,col="black",lwd=2) # (current in black)
points(seq.sp,pch=16,cex=1.25,col="red") # (new SPs in red)
contour.default(x=x1,y=x2,z=z,
               drawlabels=TRUE,nlevels=10) #contour
points(cur.sp,pch=4,cex=1.25,col="black",lwd=2) # (current in black)
points(seq.sp,pch=16,cex=1.25,col="red") # (new SPs in red)

## End(Not run)

```

# Index

\*Topic **package**

support-package, [2](#)

e\_dist, [2](#)

sp, [4](#)

sp\_seq, [8](#)

support (support-package), [2](#)

support-package, [2](#)