Hyperspectral unmixing

R output for thesis: http://josipovic.be/statistics/dissertation/Davor,%20Hyperspectral%20Unmixing,%202017.pdf

D. Josipovic

31 mei 2017

Contents

1 Problem specification [1]

Suppose $\boldsymbol{x}_i$ is a (B x 1) column vector which represents a spectrum measured at B wavelengths (or bands). We decompose this vector into components according to the following model:

$$\boldsymbol{x}_i = \boldsymbol{E}\boldsymbol{w}_i + \epsilon \tag{1}$$

where $\boldsymbol{E} \in \mathbb{R}_+^{B \times M}$ matrix of whose columns $\boldsymbol{e}_m$ represent the spectra of the M unknown endmembers; $\boldsymbol{w}_n \in \mathbb{R}_+^{M \times 1}$ column vector of endmember proportions in $\boldsymbol{x}_n$; and ϵ is the errorterm with covariance $\boldsymbol{\Sigma}$.

Note also that

$$w_{nm} \geq 0, m = 1, \ldots, M \ \textit{and} \ \sum_m^M w_{nm} = 1 \tag{2}$$

which is called the closure or convex geometry constraint [1]. This implies that $\boldsymbol{w}_n$ lie on a unit $(M-1)$-simplex.

Equation 1 can be written in matrix form:

$$\boldsymbol{X} = \boldsymbol{W}\boldsymbol{E}^T + \epsilon \tag{3}$$

where $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^T$ is a N x B matrix of the observed spectra and $\boldsymbol{W} = (\boldsymbol{w_1}, \ldots, \boldsymbol{w}_N)^T$ is a M x N matrix whose nth row represents the endmember concentration profile of nth sample.

Furthermore, the non-negativity and additivity constraint in Eq. 2 implies that $\boldsymbol{x}_n$ are contained within a simplex where the endmembers $\boldsymbol{e}_m$ form the vertices. That some $\boldsymbol{x}_n$ fall out of the simplex is due to noise.

1.1 References

[1] Berman, M. e.a., ICE a new method for the multivariate curve resolution of hyperspectral images, 2008
[2] Arngren, M. e.a., Unmixing of Hyperspectral Images using Bayesian Nonnegative Matrix Factorization with Volume Prior, 2011

2 Hyperspectral images

2.1 Objects

2.1.1 W object

```
setClass('W', contains = "matrix",
         slots = c(width='numeric', height='numeric', origin='character'))
```

```
hy$W <- function(data = NULL, width = numeric(0), height = numeric(0),
                 M = if (is.null(data) | length(ncol(data)) == 0)
                        {stop('Specify M or give some data matrix!')}
                     else {ncol(data)},
                 N = if (length(width * height) == 0 & is.null(data))
                        {stop('Can not resolve N')}
                     else {max(width * height, nrow(data), na.rm = TRUE)},
                 origin = 'unknown', ...){
  new('W', ..., data = if (is.null(data)) {NA} else {data},
      width = width, height = height, ncol = M, nrow = N, origin = origin)
```

```
}
hy$W(width = 2, height = 2, M = 3)
```

```
## An object of class "W"
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
## [3,]   NA   NA   NA
## [4,]   NA   NA   NA
## Slot "width":
## [1] 2
##
## Slot "height":
## [1] 2
##
## Slot "origin":
## [1] "unknown"
```

2.1.2 E object

```
setClass('E', contains = "matrix")
```

```
hy$E <- function(data = NA,
                 M = if (is.matrix(data)) {ncol(data)} else {stop('M required!')},
                 B = if (is.matrix(data)) {nrow(data)} else {stop('B required!')},
                 wavelengths = 1:B, eNames = paste('e', 1:M, sep = ''), ...){
  E <- new('E', ..., data = data, ncol = M, nrow = B,
           dimnames = list(wavelengths, eNames))
  E
}
hy$E(M = 3, B = 2)
```

```
## An object of class "E"
##   e1 e2 e3
## 1 NA NA NA
## 2 NA NA NA
```

```
hy$E(matrix(1:4, ncol=2, nrow=2))
```

```
## An object of class "E"
##   e1 e2
## 1  1  3
## 2  2  4
```

2.1.3 Generic prediction object

```
setClass('hyperSpecPred',
         slots = c(type = 'character', height = 'numeric', width = 'numeric',
                   eLabels = 'character'), prototype = prototype(type = 'Generic'))
```

```
new('hyperSpecPred')
```

```
## An object of class "hyperSpecPred"
## Slot "type":
## [1] "Generic"
##
## Slot "height":
```

```
## numeric(0)
##
## Slot "width":
## numeric(0)
##
## Slot "eLabels":
## character(0)
```

2.1.4 Extending *hyperSpec* to hold E, W etc.

```
library(hyperSpec)
```

```
## Loading required package: lattice
```

```
## Loading required package: grid
```

```
## Loading required package: ggplot2
```

```
## Package hyperSpec, version 0.98-20161118
##
## To get started, try
##    vignette ("introduction", package = "hyperSpec")
##    package?hyperSpec
##    vignette (package = "hyperSpec")
##
## If you use this package please cite it appropriately.
##    citation("hyperSpec")
## will give you the correct reference.
##
## The project homepage is http://hyperspec.r-forge.r-project.org
```

```
setClass("hyperSpecExt", contains = "hyperSpec",
         slots = c(W='W', E='matrix', sigma2='numeric',
                   pred = 'list', # list of hyperSpecPred objects
                   usd='list' # list for various ad-hoc data
                   ))
```

```
setGeneric(name = "height", where = hy, function(object) {
  standardGeneric("height")
})
```

```
## [1] "height"
```

```
setMethod("height", where = hy, signature(object = "hyperSpec"), function(object) {
  max(object$y)-min(object$y)+1
})
```

```
## [1] "height"
```

```
setGeneric(name = "width", where = hy, function(object) {
  standardGeneric("width")
})
```

```
## [1] "width"
```

```
setMethod("width", where = hy, signature(object = "hyperSpec"), function(object) {
  max(object$x)-min(object$x)+1
})
```

```
## [1] "width"
```

```
hy$hyperSpecExt <- function(width, height, depth, wavelengths = 1:depth,
                            spc = matrix(numeric(0), nrow = height*width, ncol = depth)){
```

```
  new('hyperSpecExt', spc = spc,
      data = data.frame(x = rep(1:width, each = height), y = rep(1:height, width)),
      wavelength = wavelengths)
}
hy$hyperSpecExt(width = 2, height = 2, depth = 2)
```

```
## hyperSpec object
##    4 spectra
##    3 data columns
##    2 data points / spectrum
## wavelength:  [integer] 1 2
## data:  (4 rows x 3 columns)
##    1. x:  [integer] 1 1 2 2
##    2. y:  [integer] 1 2 1 2
##    3. spc:  [matrix2] NA NA ... NA + NA
```

2.2 Pixel generator

Before we start making images, we need some pixel generation function which will randomly generate $\boldsymbol{X}$. We use the model from Eq. 3. For this we need a function *rW()* which will randomly generate $\boldsymbol{W}$.

2.2.1 Generating random W

```
hy$rdirichlet <- function(alpha, previous = rep(0, length(alpha))){
  stopifnot(length(alpha) == length(previous))
  repeat{
    rG <- previous + rgamma(n = length(alpha), shape = alpha, scale = 1)
    rD <- rG/sum(rG)
    if (sum(rD) == 1) {return(rD)} # check for imprecision
  }
}
hy$rdirichlet(c(1,5,10))
```

```
## [1] 0.00482 0.18945 0.80573
```

```
hy$rW <- function(width = numeric(0), height = numeric(0), M,
                  N = width * height, weightConstraint = NULL) {
  W <- hy$W(width = width, height = height, N = N, M = M, origin = 'rW')
  for (i in 1:N) {
    repeat{
      wi <- hy$rdirichlet(alpha = rep(1, M))
      if (is.null(weightConstraint) || weightConstraint(wi)) {break}
    }
    W[i,] <- wi
  }
  return(W)
}
hy$rW(N = 4, M = 2)
```

```
## An object of class "W"
##       [,1]   [,2]
## [1,] 0.349 0.6509
## [2,] 0.655 0.3454
## [3,] 0.680 0.3195
## [4,] 0.969 0.0306
## Slot "width":
## numeric(0)
```

```
##
## Slot "height":
## numeric(0)
##
## Slot "origin":
## [1] "rW"
```

2.2.1.1 Constraints on w_{nm}

As noted previously, real life abundances of endmembers are never 100 %. To simulate this we add the following *weightConstraint()* to our $\boldsymbol{W}$ generating function.

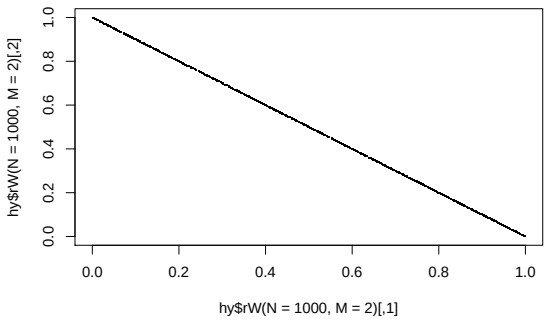
```
# first weight is never higher that 0.6!
hy$rW(N = 5, M = 2, weightConstraint = function(wi){ return( wi[1] < 0.6 ) })
```

```
## An object of class "W"
##       [,1]  [,2]
## [1,] 0.581 0.419
## [2,] 0.124 0.876
## [3,] 0.476 0.524
## [4,] 0.031 0.969
## [5,] 0.129 0.871
## Slot "width":
## numeric(0)
##
## Slot "height":
## numeric(0)
##
## Slot "origin":
## [1] "rW"
```

2.2.1.2 Distribution of w_n

2.2.1.2.1 Uniformity check

```
plot(x = hy$rW(N = 1000, M = 2), pch = ".", xlim = c(0,1), ylim = c(0,1))
```



2.2.1.2.2 Dirlichet distribution

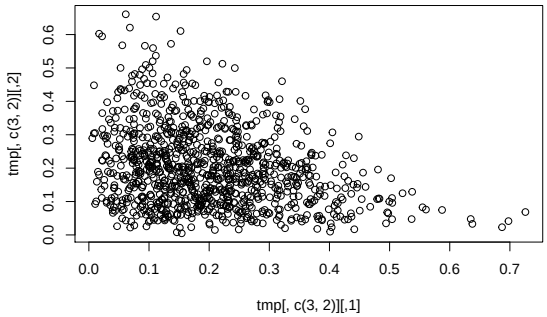
This is a convenience function that calculates the dirlichet parameters for specific expected value of the first and second endmember, and the variance of the first endmember. These three are enough to determine the expecetd value of the third endmember, and variances of the second and third endmembers.

```
hy$dericletParameters <- function(ExpectedW1, ExpectedW2, VarW1){
  a <- rep(NA, 3)
  a0 <- (ExpectedW1-ExpectedW1^2)/VarW1 - 1
  a[1] <- ExpectedW1*a0
  a[2] <- ExpectedW2*a0
  a[3] <- a0 - a[1] - a[2]
  EX <- a/a0 # expected value
  VarX <- a*(a0-a)/(a0^2 * (a0 + 1)) # variance
  CovX <- -a %*% t(a)/(a0^2 * (a0 + 1)) # covariance -- variance is wrong here
  diag(CovX) <- VarX
  list(a0=a0, a=a, EX=EX, CovX=CovX)
}
hy$dericletParameters(ExpectedW1 = 0.6, ExpectedW2 = 0.2, VarW1 = 0.02)
```

```
## $a0
## [1] 11
##
## $a
## [1] 6.6 2.2 2.2
##
## $EX
## [1] 0.6 0.2 0.2
##
## $CovX
##       [,1]     [,2]     [,3]
## [1,]  0.02 -0.01000 -0.01000
## [2,] -0.01  0.01333 -0.00333
## [3,] -0.01 -0.00333  0.01333
```

Check whether the *dericletParameters()* calculation is correct. So we simulate here a dirlechet distribution with expected values 0.6, 0.2 and 0.2 using dirlichet parameters $\alpha = (6.6, 2.2, 2.2)$.

```
tmp <- t(apply(matrix(c(6.6, 2.2, 2.2), ncol = 3, nrow = 1000, byrow = TRUE),
               MARGIN = 1, FUN = hy$rdirichlet))
plot(tmp[,c(3,2)])
```




```
colMeans(tmp)
```

```
## [1] 0.596 0.203 0.200
```

```
var(tmp)
```

```
##          [,1]     [,2]     [,3]
## [1,]  0.01997 -0.01036 -0.00961
## [2,] -0.01036  0.01405 -0.00369
## [3,] -0.00961 -0.00369  0.01330
```

```
rm(tmp)
```

2.2.2 Generating random E

```
hy$rE <- function(M, D, quantization = 8, maxval = 2^quantization-1,
                  wavelengths = NULL){ # 8 bit quantization default
  max <- (2^quantization-1) / (2^quantization-1) * maxval
  E <- hy$E(data = runif(n = M*D, min = 0, max = max), B = D, M = M)
  if (length(wavelengths) >= 0) rownames(E) <- wavelengths
  E
}
hy$rE(M = 5, D = 4)
```

```
## An object of class "E"
##         e1    e2    e3    e4    e5
## [1,]  98.9  71.5 135.8  95.2 155.6
## [2,]  83.3  61.2 245.3  19.6 108.1
## [3,] 135.0  98.5  15.8  34.7 158.4
## [4,]  76.0 220.9 233.8 206.3  40.1
```

2.2.2.1 Fixed E

For sake of simplicity we use the following fixed one- and twodimensional endmembers throughout.

```
Ed1 <- cbind(0.1, 0.5)
Ed2 <- cbind(c(0.2, 0.2), c(0.9, 0.4), c(0.3, 0.8))
Ed1
```

```
##      [,1] [,2]
## [1,]  0.1  0.5
```

```
Ed2
```

```
##      [,1] [,2] [,3]
## [1,]  0.2  0.9  0.3
## [2,]  0.2  0.4  0.8
```

2.2.3 Calculate noise variance based on desired SNR

We need noise variance in his case so we can sample ϵ. For each of our pixel components we have Gaussian noise ϵ with a specific variance σ^2_{noise}.

$$x_{nd} = \boldsymbol{E}_{d:}\boldsymbol{w}_n + \epsilon$$

The signal to noise ratio in dB (SNR_{dB}) is defined as:

$$SNR_{dB} = 10log_{10}\left(\frac{\sigma^2_{signal}}{\sigma^2_{noise}}\right)$$

We estimate $\sigma^2_{signal} \sim \hat{\sigma}^2_{signal} = var(\boldsymbol{W}\boldsymbol{E}^T)$. We can then compute the noise variance in function of desired SNR_{dB}.

```
hy$noiseVarFromSNR <- function(varSignal, SNRdB){
  varSignal/exp(SNRdB/10*log(10)) # denumerator equivalent to 10^(SNRdB/10)
}
```

2.2.4 Generating random X

Function *rX()* will generate random pixels according to Eq. 3.

```
hy$rX <- function(N, E, SNRdb = NULL, sigma2 = 1, ...){
  M <- dim(E)[2] # E is a (D x M) matrix
  D <- dim(E)[1]
  W <- hy$rW(N = N, M = M, ...)
  XnoError <- W %*% t(E)
  if (!is.null(SNRdb)) {
    sigma2 <- hy$noiseVarFromSNR(varSignal = var(as.vector(XnoError)), SNRdB = SNRdb)
  }
  epsilon <- matrix(data = rnorm(N*D, sd = sqrt(sigma2)), ncol = D)
  X <- XnoError + epsilon
  return(structure(X, W=W, E=E, epsilon = epsilon, sigma2 = sigma2, class = "X"))
}
hy$rX(N = 6, E = Ed2, sigma2 = 1, width = 3, height = 2)
```

```
##       [,1]   [,2]
## [1,] -0.44  0.565
## [2,]  1.31  0.246
## [3,]  1.36 -0.164
## [4,]  1.43 -0.474
## [5,] -0.94 -0.160
## [6,]  1.89  0.923
## attr(,"W")
## An object of class "W"
##        [,1]    [,2]   [,3]
## [1,] 0.5359 0.42963 0.0345
## [2,] 0.7770 0.11468 0.1083
## [3,] 0.2201 0.54064 0.2392
## [4,] 0.3433 0.00364 0.6530
## [5,] 0.6511 0.03683 0.3121
## [6,] 0.0801 0.71473 0.2051
## Slot "width":
## [1] 3
##
## Slot "height":
## [1] 2
##
## Slot "origin":
## [1] "rW"
##
## attr(,"E")
##      [,1] [,2] [,3]
## [1,]  0.2  0.9  0.3
## [2,]  0.2  0.4  0.8
## attr(,"epsilon")
##        [,1]    [,2]
## [1,] -0.944  0.2581
## [2,]  1.021 -0.0418
## [3,]  0.759 -0.6158
## [4,]  1.158 -1.0662
## [5,] -1.197 -0.5549
## [6,]  1.171  0.4566
## attr(,"sigma2")
## [1] 1
## attr(,"class")
## [1] "X"
```

2.3 Synthetic image generator

2.3.1 Uniform random pixel generated images

Function to generate a random hyperspectral image object.

```
hy$rImage <- function(height, width, E, SNRdb = NULL, ...){
  X <- hy$rX(N = height*width, E = E, width = width, height = height, SNRdb = SNRdb, ...)
  IM <- hy$hyperSpecExt(width = width, height = height, depth = ncol(X),
                        wavelengths = 1:ncol(X), spc = X)
  IM@E <- attr(X, "E")
  IM@W <- attr(X, "W")
  IM@sigma2 <- attr(X, "sigma2")

  return(IM)
}
hy$rImage(height = 3, width = 3, E = Ed1)
```

```
## hyperSpec object
##    9 spectra
##    3 data columns
##    1 data points / spectrum
## wavelength:  [integer] 1
## data:  (9 rows x 3 columns)
##    1. x:  [integer] 1 1 ... 3
##    2. y:  [integer] 1 2 ... 3
##    3. spc:  [matrix1] -1.83 2.05 ... 0.566
```

2.3.1.1 Random 1D (grayscale) image

```
IM2x2D1 <- hy$rImage(height = 2, width = 2, E = Ed1, sigma2 = 0.001)
IM9x9D1 <- hy$rImage(height = 9, width = 9, E = Ed1, sigma2 = 0.001,
                     weightConstraint=function(wi){ return( wi[1] < 0.6 ) })
```

2.3.1.1.1 Random 2D (2-color) images

```
IM2x2D2 <- hy$rImage(height = 2, width = 2, E = Ed2, sigma2 = 0.001)
IM9x9D2 <- hy$rImage(height = 9, width = 9, E = Ed2, sigma2 = 0.001,
                     weightConstraint=function(wi){ return( wi[1] < 0.6 ) })
IM30x30D2 <- hy$rImage(height = 30, width = 30, E = Ed2, sigma2 = 0.001,
                       weightConstraint=function(wi){ return( wi[1] < 0.6 ) })
IM30x30D2FM <- hy$rImage(height = 30, width = 30, E = Ed2, sigma2 = 0.001) # Full mixing
```

2.4 False color plotting

First we need a function that will construct a 3D data cube from the *hyperSpec* object.

```
hy$dcube <- function(hyImg){
  cube <- array(data = hyImg[[]], dim = c(hy$height(hyImg),
                                          hy$width(hyImg),
                                          ncol(hyImg[[]])),
                dimnames = list(y = min(hyImg$y):max(hyImg$y),
                                x = min(hyImg$x):max(hyImg$x),
                                z = hyperSpec::wl(hyImg)))
  cube
}
hy$dcube(IM2x2D2)
```

```
## , , z = 1
##
##    x
## y       1     2
##   1 0.305 0.506
##   2 0.390 0.256
##
## , , z = 2
##
##    x
## y       1     2
##   1 0.213 0.368
##   2 0.495 0.337
```

Then we need a function that will convert the datacube to a false color raster.

```
hy$fcRaster <- function(dcube, dcubeMaxRef = dcube){
  depth <- dim(dcube)[3]
  dCubeNormalizedDim <- c(dim(dcube)[1], dim(dcube)[2], 3)
  dcubeNormalized <- array(dim = dCubeNormalizedDim)

  # data cube can be 1, 2 or 3-dimensional
  for (i in 1:depth){
    dcubeNormalized[,,i] <- dcube[,,i]/max(dcubeMaxRef[,,i])
  }

  if (depth == 1){
    dcubeNormalized[,,2] <- dcube[,,1]/max(dcubeMaxRef[,,1])
    dcubeNormalized[,,3] <- dcube[,,1]/max(dcubeMaxRef[,,1])
  }

  if (depth == 2){
    dcubeNormalized[,,3] <- matrix(0, nrow = dim(dcube)[1], ncol = dim(dcube)[2])
  }

  as.raster(dcubeNormalized)
}
hy$fcRaster(hy$dcube(IM2x2D2)) # 2 colors are squeezed into one color raster
```

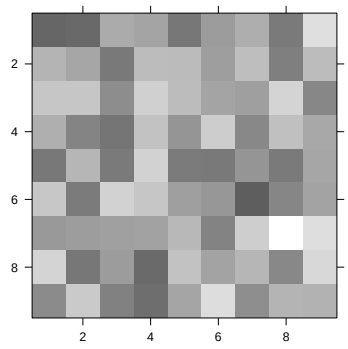
```
##      [,1]      [,2]
## [1,] "#9A6E00" "#FFBE00"
## [2,] "#C5FF00" "#81AE00"
```

Next we need a function that will do the plotting.

```
hy$plotfc <- function(hyImage, printAs = character(0)){
  dc <- hy$dcube(hyImage)
  xl <- as.integer(colnames(dc))
  yl <- as.integer(rownames(dc))

  print(paste('Wavelengths for falsecolor image:',
              paste(hyperSpec::wl(hyImage), collapse = ', ')))

  rgbInt <- apply(hy$fcRaster(dc), MARGIN = c(1,2),
             FUN = function(str){strtoi(substring(str, 2), base = 16)})

  rgbIntUnq <- unique(as.vector(rgbInt))

  plt <- lattice::levelplot(
    x = t(rgbInt), # turn so rows become columns
```

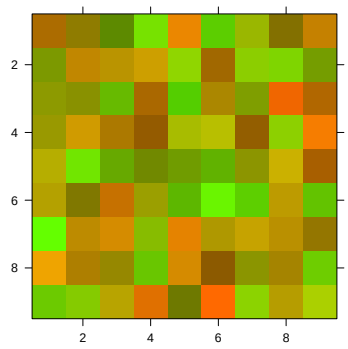
```
    at = c(-0.001,rgbIntUnq),
    col.regions = sprintf("#%06X", rgbIntUnq[order(rgbIntUnq)]),
    ylim = c(max(yl) + 0.5, min(yl) - 0.5), # plot from top to doum
    xlab = '', ylab = '', colorkey = FALSE,
    row.values = xl, column.values = yl)

  print(plt)

  if (length(printAs) != 0) {
    dev.copy2pdf( file = paste('./../fig/', gsub('[.]', '_', printAs),
                               '-FalseColor', '.pdf', sep = ""),
                  compress = FALSE )
  }
}
hy$plotfc(IM9x9D1)
```

```
## [1] "Wavelengths for falsecolor image: 1"
```




```
hy$plotfc(IM9x9D2)
```

```
## [1] "Wavelengths for falsecolor image: 1, 2"
```



2.4.1 RGB band selection

The bands that signify red, green and blue color. These bands are used for false color plotting.

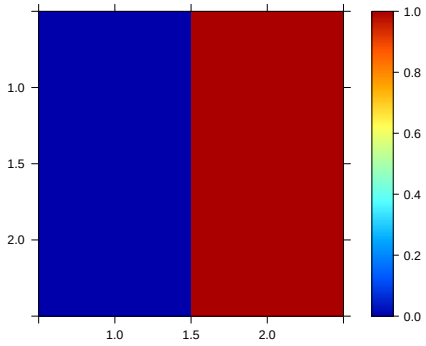
```
hy$rgbBands <- c(750,530,480)
```

2.5 Abundances plotting

First we need something for plotting W.

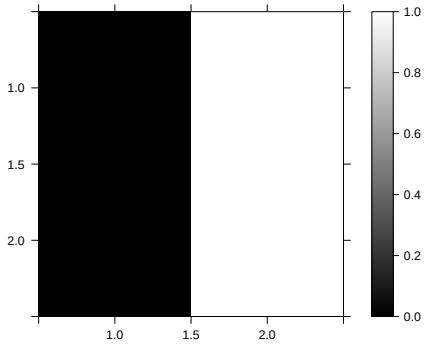
```
hy$plotMapW <- function(W, printAs = character(0), suffix = W@origin,
                        eLabelOverride = NULL, bw = TRUE){

  if (length(W[W<0 | W>1]) > 0) {
    probvals <- W[W<0 | W>1]
    cat(paste("Problematic W-values:",
              "\n Count: ", length(probvals),
              "\n Min: ", min(probvals),
              "\n Max: ", max(probvals),
              '\n', sep = ''))
    W[W<0] <- 0; W[W>1] <- 1 # Correct those problematic values
  }

  colMap <- if (bw) {gray((0:255)/255)} else {colorRamps::matlab.like(256)}

  # Not passing matrix to spc causes 'NAs introduced by coercion' error.
  # It simplty strips W of attributes.
  hyW <- new('hyperSpec', spc = matrix(W, ncol = ncol(W)),
             data = data.frame(x = rep(1:W@width, each = W@height),
                               y = rep(1:W@height, W@width)))
  for(e in 1:ncol(W)){
    if (!is.null(eLabelOverride)) {
      eLabel <- eLabelOverride[e]
    } else {
      eLabel <- if (!is.null(colnames(W))) {colnames(W)[e]} else {paste('E', e, sep = '')}
    }
    print(paste('Plotting:', eLabel))

    plt <- hyperSpec::plotmap(hyW[,,e], col.regions = colMap,
                              ylim = c(max(hyW$y)+0.5, min(hyW$y)-0.5),
                              at = seq(0, 1, 1/256))
    # "at" signifies the ranges between which the "col.regions" colors are used.
    # thus "at" requires one extra components such that it has as many ranges
    # as "col.regions" has colors.

    print(plt)
    if (length(printAs) != 0) {
      dev.copy2pdf( file = paste('./../fig/', gsub('[.]', '_', printAs),
                                 '-W-', eLabel, '-', suffix, '.pdf',sep = ""),
                    compress = FALSE)
    }
  }
}
hy$plotMapW(hy$W(data = matrix(c(-0.5,0,1,1.1)), width = 2, height = 2), bw = FALSE)
```

```
## Problematic W-values:
##  Count: 2
##  Min: -0.5
##  Max: 1.1
## [1] "Plotting: E1"
```



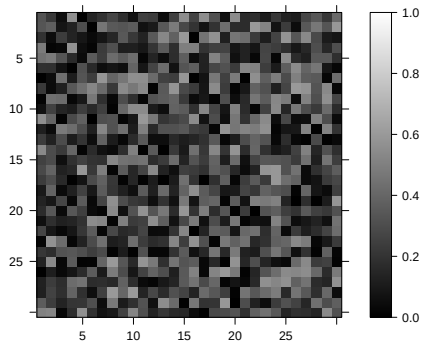

```
hy$plotMapW(hy$W(data = matrix(c(-0.5,0,1,1.1)), width = 2, height = 2), bw = TRUE)
```

```
## Problematic W-values:
##  Count: 2
##  Min: -0.5
##  Max: 1.1
## [1] "Plotting: E1"
```

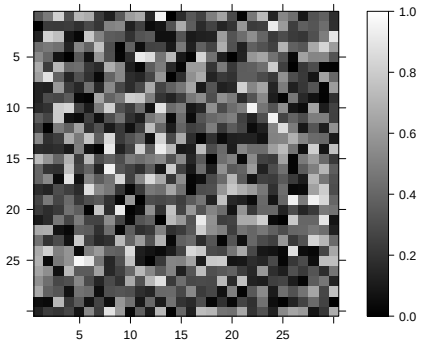



```
hy$plotMapW(IM30x30D2@W)
```

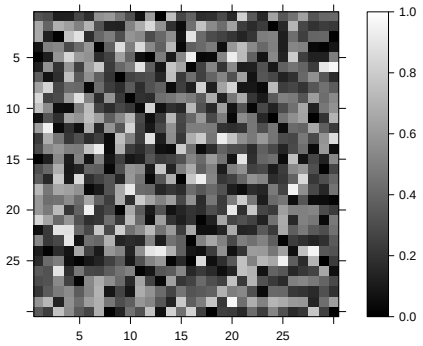
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```



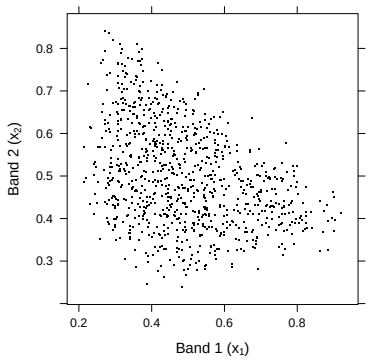

```
## [1] "Plotting: E3"
```



2.6 Simplex plotting

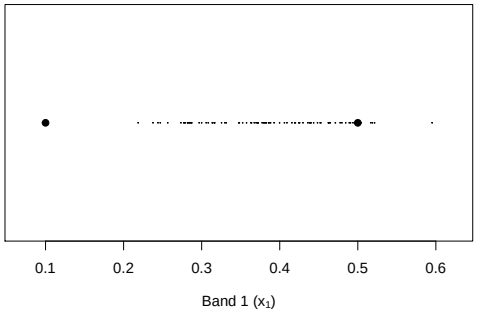
```
hy$plotSimplex <- function(X, E = array(NA, c(ncol(X), ncol(X) + 1)),
                           Ehat = replace(x = E, values = NA),
                           printAs = character(0), pointCex = 0.1,
                           EhatCol = rep(c('red', 'blue', 'orange'), 3),
                           plotPolygons = TRUE, ylim = NULL, xlim = NULL,
                           legendLog = TRUE){

  Ehat <- if (is.list(Ehat)) {Ehat} else {list(Ehat)}

  if (is.null(xlim)) {
    xlim <- c(min(X[,1], E[1,], sapply(Ehat, function(E){E[1,]}), na.rm = TRUE),
              max(X[,1], E[1,], sapply(Ehat, function(E){E[1,]}), na.rm = TRUE))
    xlim <- xlim + c(-0.05, 0.05) * max(xlim)
  }


  plotSimplex2D <- function(){
    if (is.null(ylim)){
      ylim <- c(min(X[,2], E[2,], sapply(Ehat, function(E){E[2,]}), na.rm = TRUE),
                max(X[,2], E[2,], sapply(Ehat, function(E){E[2,]}), na.rm = TRUE))
      ylim <- ylim + c(-0.05, 0.05) * max(ylim)
    }

    plt <- lattice::xyplot(X[,2] ~ X[,1], aspect = 1, pch = '.', cex = pointCex,
                           col = 'black', xlim = xlim, ylim = ylim,
                           panel = function(...) {
```

```
                              lattice::panel.xyplot(...)
                              lattice::lpoints(x = E[1,], y = E[2,], cex = 0.7,
                                               col = 'black', pch = 19)
                              lattice::panel.polygon(x = E[1,], y = E[2,])
                              for (i in 1:length(Ehat)) {
                                if (legendLog)
                                  {print(paste(names(Ehat)[i], 'color:', EhatCol[i]))}
                                E <- Ehat[[i]]
                                lattice::lpoints(x = E[1,], y = E[2,], cex = 0.7,
                                                 col = EhatCol[i], pch = 19)
                                if (plotPolygons)
                                  lattice::panel.polygon(x = E[1,], y = E[2,],
                                                         lty = i + 1,
                                                         border = EhatCol[i])
                              }
                            },
                            xlab = expression('Band 1 (x'[1]*')'),
                            ylab = expression('Band 2 (x'[2]*')'))

    print(plt)
  }

  plotSimplex1D <- function(){
    stripchart(E, pch = 19, xlim = xlim, xlab = expression('Band 1 (x'[1]*')'))
    stripchart(x = X, pch = 19, cex = pointCex, add = TRUE)
    if (!all(is.na(Ehat[[1]]))) {stripchart(Ehat[[1]], pch = 19, col = 'red', add = TRUE)}
  }

  switch (nrow(E),
    "1" = plotSimplex1D,
    "2" = plotSimplex2D,
    stop("Can only plot upt to 2 dimensions!")
  )()

  if (length(printAs) != 0) {
    dev.copy2pdf( file = paste('./../fig/', gsub('[.]', '_', printAs),
                               '-simplex.pdf',sep = ""),
                  compress = FALSE )
  }
}
hy$plotSimplex(IM30x30D2[[]])
```

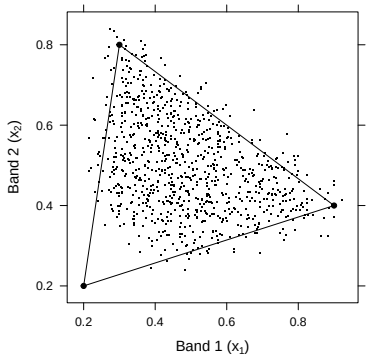



```
## [1] " color: red"
```

```
hy$plotSimplex(IM9x9D1[[]], E = Ed1)
```




```
hy$plotSimplex(IM30x30D2[[]], E = Ed2, printAs = 'IM30x30D2')
```




```
## [1] " color: red"
```

```
## pdf
##   2
```

3 Error measures for hyperspectral data

3.1 Spectral Angle Mapper (SAM) [1]

$$\alpha_{SAM} = \cos^{-1}\left[\frac{\boldsymbol{e}_{ref}\boldsymbol{e}_{test}}{\|\boldsymbol{e}_{ref}\|\|\boldsymbol{e}_{test}\|}\right]$$

SAM has always a result bewteen 0 and 1 (because of arccosine function). 0 means that the vectors are very similar, and 1 that they are not.

```
setGeneric("SAM", where = measure, function(ref, test, ...){standardGeneric("SAM")})
```

```
## [1] "SAM"
```

```
setMethod(where = measure, "SAM", signature(ref = 'numeric', test = 'numeric'), function(ref, test)
  x <- (ref %*% test) / (sqrt(ref %*% ref) * sqrt(test %*% test))
  acos( min(max(x, -1.0), 1.0) ) # because x is sometimes out of the [-1, 1] boundary
})
```

```
## [1] "SAM"
```

```
measure$SAM(1:3, 2*1:3)
```

```
## [1] 0
```

Computing the SAM for E_{ref} with E_{test} is a bit more tricky. If we assume that first endmember of E_{ref} corresponds with first endmember of E_{test}, etc., then we can simply compute SAM 1:1. But if we do not know the correspondence, which is often the case, then we need to compute SAMs for E_{ref} with all possible permutations of endmembers of E_{test}, and report the smallest one.

```
setMethod(where = measure, "SAM", signature(ref = 'matrix', test = 'matrix'),
function(ref, test, perm = TRUE) {
  stopifnot(ncol(ref) == ncol(test))

  min(
    apply(X = iterpc::getall(iterpc::iterpc(ncol(ref), ordered = perm)), MARGIN = 1,
          ref = ref, test = test,
          FUN = function(p, ref, test){
            # print(p)
            sum(sapply(X = 1:ncol(ref),
                       ref = ref, test = test[,p],
                       FUN = function(i, ref, test){
                         measure$SAM(ref[,i], test[,i])
                       }))
          })
  )
})
```

```
## [1] "SAM"
```

Examples

```
measure$SAM(c(1,2,3), c(2,4,6))
```

```
## [1] 0
```

```
measure$SAM(ref = matrix(1:8, ncol = 2), test = matrix(2*1:8, ncol = 2))
```

```
## [1] 0
```

```
measure$SAM(ref = Ed2, test = Ed2[,c(3,1,2)]) # should be 0 (or close to)
```

```
## [1] 3.6e-08
```

```
measure$SAM(ref = Ed2, test = Ed2[,c(3,1,2)], perm = FALSE) # should not be 0
```

```
## [1] 1.59
```

```
showMethods('SAM', where = measure)
```

```
## Function: SAM (package .GlobalEnv)
## ref="integer", test="numeric"
##     (inherited from: ref="numeric", test="numeric")
## ref="matrix", test="matrix"
## ref="numeric", test="numeric"
```

[1] Kruse, F.A. e.a, The Spectral Image Processing System, 1993

4 Endmembers and USGS library

4.1 Loading USGS spectra

```
em$read <- function(filename, type = 'M', skipLines = 16){
  cleanup <- function(data){
    data[data != -1.23e+34]
  }
```

```
  filePath <-  file.path('./..',
                         'data/usgs/ASCII', type, paste(filename, '.asc', sep = ''))
  dta <- read.delim(file = filePath, skip = skipLines, sep = '', header = FALSE,
                    col.names = c('wavelength','reflectance','sd'))


  vec <- dta$reflectance
  names(vec) <- dta$wavelength * 1000 # micro to nano meter

  cleanup(vec)
}
em$read('brick_gds353.27580', 'A')[1:5]
```

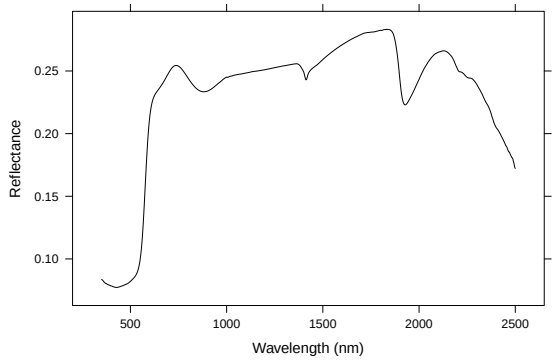
```
##    350    351    352    353    354
## 0.0835 0.0835 0.0835 0.0834 0.0832
```

4.2 Plotting spectra

Assumption is that the rows of a E matrix (or names of vector if e) are wavelengths.

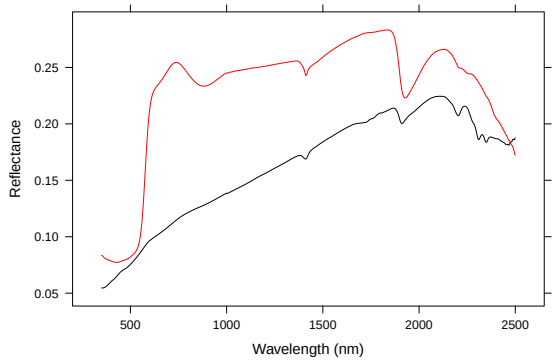
```
em$plot <- function(E, labelSuffix = '', legend = FALSE, columnOrder = NULL,
                      printAs = character(0), suffix = character(0),
                      colPalette = c('black', 'red', 'blue', 'orange')){
  if (!is.null(columnOrder)) {E <- E[,columnOrder]}
  data <- as.data.frame(E)
  colnames(data) <- paste(labelSuffix, colnames(data), sep = '')
  variables = if (is.vector(E)) {'E'} else {colnames(data)}
  vars <- paste(paste('`', variables, '`', sep = ''), collapse = '+')
  fm <- formula(paste(vars, '~', 'as.numeric(rownames(data))', sep = ''))

  if (legend) {
    auto.key = list(space = "right", columns = 1, points = FALSE, lines = FALSE,
                    col = colPalette)
  } else {
    auto.key = FALSE
    cat(paste(variables, ': ', colPalette[1:length(variables)],
              sep = '', collapse = '\n'))
    cat('\n')
  }

  plt <- lattice::xyplot(
    fm, data, type = "l", auto.key = auto.key, col = colPalette,
    ylab = 'Reflectance', xlab = 'Wavelength (nm)')

  print(plt)
  if (length(printAs) != 0) {
    if (length(suffix) != 0) suffix <- paste('-', suffix, sep = '')
    dev.copy2pdf( file = paste('./../fig/', gsub('[.]', '_', printAs),
                               '-E-spectrum', suffix, '.pdf',sep = ""),
                  compress = FALSE )
  }
}
em$plot(em$read('brick_gds353.27580', 'A'))
```

```
## E: black
```




```
em$plot(cbind(Brick = em$read('brick_gds353.27580', 'A'),
              Asphalt = em$read('black_old_asphaltroof_gds376.27384', 'A')),
        columnOrder = c('Asphalt', 'Brick'))
```

```
## Asphalt: black
## Brick: red
```



4.3 USGS material spectra for simulation (B = 2151)

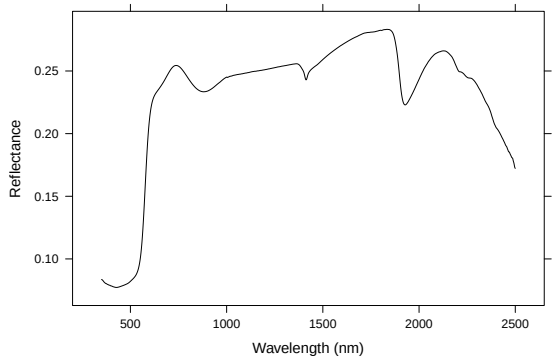
For material spectra we use the USGS library. For spectral consistency, we use only spectra measured with the same apparatus and resolution: W1R1Fx where R1F = Resolution 1 of the ASD Fieldspec spectrometer.

The hyperspectral image is composed of three different regions

4.3.1 Brick

```
em$brick_B2151 <- em$read('brick_gds353.27580', 'A')
em$plot(em$brick_B2151)
```

```
## E: black
```



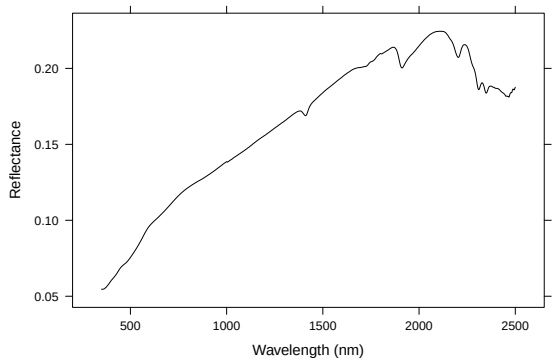

```
length(em$brick_B2151)
```

```
## [1] 2151
```

4.3.2 Asphalt

```
em$asphalt_B2151 <- em$read('black_old_asphaltroof_gds376.27384', 'A')
em$plot(em$asphalt_B2151)
```

```
## E: black
```



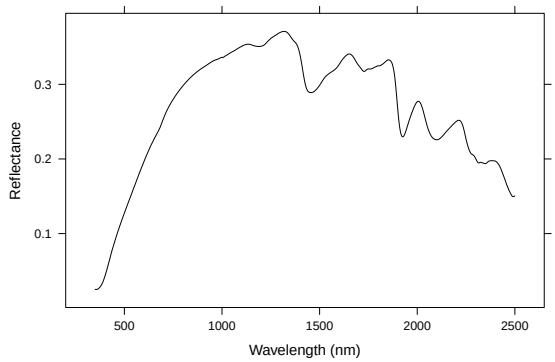

```
length(em$asphalt_B2151)
```

```
## [1] 2151
```

4.3.3 Grass

```
em$grass_B2151 <- em$read('golden_dry_grass.gds480.30093', 'V')
em$plot(em$grass_B2151)
```

```
## E: black
```



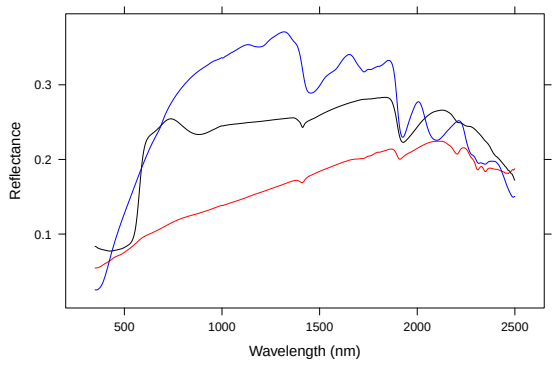

```
length(em$grass_B2151)
```

```
## [1] 2151
```

4.3.4 USGS endmember matrix E

```
em$E_B2151 <- cbind('Brick' = em$brick_B2151,
                    'Asphalt' = em$asphalt_B2151,
                    'Grass' = em$grass_B2151)
rownames(em$E_B2151) <- names(em$brick_B2151)
em$plot(em$E_B2151, printAs = 'USGS')
```

```
## Brick: black
## Asphalt: red
## Grass: blue
```



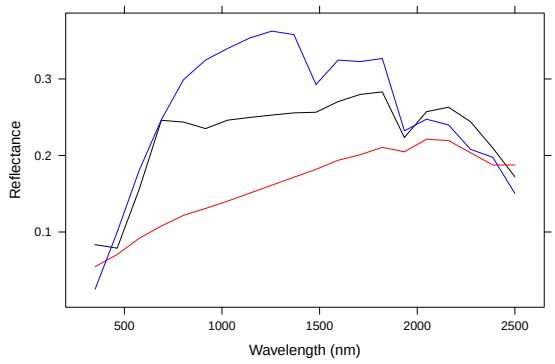

```
## pdf
##   2
```

4.4 Reducing bands with linear interpolation

```
em$interpolate <- function(E, n = NULL, wavelengths = NULL){
  stopifnot(!(is.null(n) & is.null(wavelengths)))
  stopifnot(is.null(n) | is.null(wavelengths))

  interpolateEndmember <- function(e){
    if (!is.null(n)) {
      ipList <- approx(x = as.numeric(names(e)), y = e, n = n)
      e <- ipList$y
      names(e) <- ipList$x
```

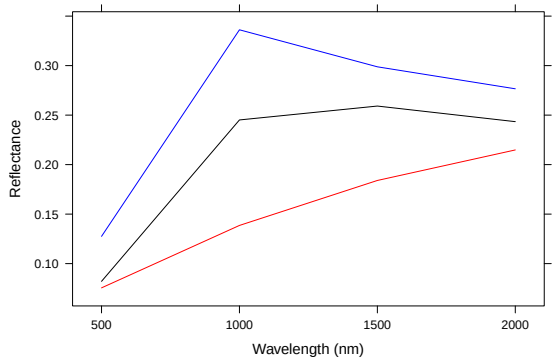
```
    } else {
      fn <- approxfun(x = as.numeric(names(e)), y = e)
      e <- fn(as.numeric(wavelengths))
      names(e) <- wavelengths
    }
    e
  }

  if (!is.matrix(E)) {E <- as.matrix(E)}

  apply(X = E, MARGIN = 2, FUN = interpolateEndmember)
}
em$plot(em$interpolate(em$E_B2151, n = 20))
```

```
## Brick: black
## Asphalt: red
## Grass: blue
```




```
em$plot(em$interpolate(em$E_B2151, wavelengths = c(500, 1000, 1500, 2000)))
```

```
## Brick: black
## Asphalt: red
## Grass: blue
```



4.5 Scaling of spectra

Scaling is done using least squares model. Linear is with intercept, otherwise affine (without intercept).

```
em$scaleSpectrum <- function(refSpectre, toScaleSpectre, linear = FALSE, return = TRUE){
  stopifnot(length(refSpectre) == length(toScaleSpectre))

  if (linear) {
```

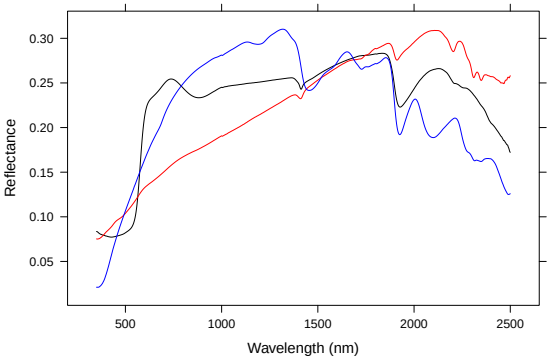
```
    coefs <- lm(refSpectre ~ toScaleSpectre)$coefficients
    return(structure(if (return) { coefs[[1]] + coefs[[2]] * toScaleSpectre } else NA,
                     'intercept' = coefs[[1]], 'slope' = coefs[[2]]))
  } else {
    coefs <- lm(refSpectre ~ toScaleSpectre - 1)$coefficients
    return(structure(if (return) { coefs[[1]] * toScaleSpectre } else NA,
                     'slope' = coefs[[1]]))
  }
}
em$scaleSpectrum(refSpectre = em$asphalt_B2151, em$brick_B2151, return = FALSE)
```

```
## [1] NA
## attr(,"slope")
## [1] 0.699
```

4.5.1 Plotting and scaling

```
em$plotScaled <- function(E, scaleToEndmemberNr = 1, wlLim = NULL, linear = FALSE, ...) {
  if (!is.null(wlLim)) {
    E <- E[findInterval(as.numeric(rownames(E)), wlLim, rightmost.closed = TRUE) == 1L,]
  }

  for (i in setdiff(1:ncol(E), scaleToEndmemberNr)) {
    E[,i] <- em$scaleSpectrum(refSpectre = E[,scaleToEndmemberNr],
                              toScaleSpectre = E[,i], linear = linear)
  }

  em$plot(E = E, ...)
}
em$plotScaled(em$E_B2151, scaleToEndmemberNr = 1)
```

```
## Brick: black
## Asphalt: red
## Grass: blue
```



4.6 Finding best matching spectrum

```
em$bestMatchingSpectrum <- function(refSpectum, E, type = 'SAM', return = FALSE, ...){
  scaled <- list()
  mse <- rep(NA, ncol(E))
  sam <- rep(NA, ncol(E))
  for (j in 1:ncol(E)) {
```

```
    scaled[[j]] <- em$scaleSpectrum(refSpectre = refSpectum, toScaleSpectre = E[,j], ...)
    mse[j] <- sum((refSpectum - scaled[[j]])**2)/length(refSpectum)
    sam[j] <- measure$SAM(ref = refSpectum, test = E[,j])
  }

  if (type == 'SAM') {cmp <- sam}
  if (type == 'MSE') {cmp <- mse}

  cat(paste('MSE:', paste(mse, collapse = ', '), '\n',
            'SAM:', paste(sam, collapse = ', '), '\n',
            'Best:', which.min(cmp), '\n'))
  if (return) { return(E[,which.min(cmp)]) }
}
em$bestMatchingSpectrum(refSpectum = em$asphalt_B2151, E = em$E_B2151)
```

```
## MSE: 0.00108490345651613, 2.25304480292154e-32, 0.00265530297375969
##  SAM: 0.196063692767929, 0, 0.309697250538724
##  Best: 2
```

```
em$matchSpectra <- function(refE, E, type = 'SAM', ...){
  stopifnot(nrow(refE) == nrow(E))
  for (refi in 1:ncol(refE)) {
    cat(paste(colnames(refE)[refi], '\n', sep = ''))
    em$bestMatchingSpectrum(refSpectum = refE[,refi], E = E, type = type, ...)
    cat('\n')
  }
}
em$matchSpectra(refE = em$E_B2151, E = em$E_B2151, linear = TRUE)
```

```
## Brick
## MSE: 1.45419350545205e-31, 0.00141773459568347, 0.000928916556783845
##  SAM: 1.49011611938477e-08, 0.196063692767929, 0.166767168905883
##  Best: 1
##
## Asphalt
## MSE: 0.00106712661572131, 2.13966921984017e-32, 0.00182157909998069
##  SAM: 0.196063692767929, 0, 0.309697250538724
##  Best: 2
##
## Grass
## MSE: 0.00213123298163465, 0.00555240638063496, 1.39847886365953e-32
##  SAM: 0.166767168905883, 0.309697250538724, 0
##  Best: 3
```

[1] Clark, R.N. e.a., Imaging spectroscopy - Earth and planetary remote sensing with the USGS Tetracorder and expert systems, 2003

5 Simplex volume calculation

5.1 Exact measure

Formula for simplex volume [2]:

$$Vol_{exact}(\boldsymbol{S}) = \frac{\sqrt{det(\widetilde{\boldsymbol{S}}^T \widetilde{\boldsymbol{S}})}}{K!} \tag{4}$$

where $\widetilde{\boldsymbol{S}}$ is composed of simplex spanning vectors.

5.2 Approximate measure

5.2.1 Parallelepiped calculation

Surrogate to Eq. 4, where the absolute vectors (columns of $\boldsymbol{E}$) are used instead of the simplex spanning vectors (columns of $\widetilde{\boldsymbol{E}}$).

```
vol$pp <- function(E){
  determinant(t(E) %*% E, logarithm = FALSE)$modulus[[1]]
}
```

Example:

```
vol$pp(E=Ed2)
```

```
## [1] 2.08e-18
```

5.2.2 Berman's calculation

Another approach, which is not based on a determinant, is to form an approximate volume based on Euclidean distance measures.[2] This approach is computationally inexpensive, as it does not require the computation of a determinant.

```
vol$berman <- function(E){
  sum(apply(X = E, MARGIN = 1, FUN = var))
}
```

Example:

```
vol$berman(E=Ed2)
```

```
## [1] 0.237
```

5.3 References

[1] Berman, M. e.a., ICE a new method for the multivariate curve resolution of hyperspectral images, 2008
[2] Arngren, M. e.a., Unmixing of Hyperspectral Images using Bayesian Nonnegative Matrix Factorization with Volume Prior, 2011

6 Quadratic optimization

The following is loosely based on [1].

Here the point is to define a framework for problem optimization. The goal is to find the values that optimize some objective (i.e. the objective function), possibly subject to constraints.

The classification is made on basis of the nature of the objective function and its constraints (linear, nonlinear, convex), smoothness, etc.

For linear and convex programming problems, the local minimizer is also a global minimizer. The linear programming problem is one for which the objective and constraints are linear, and convex programming problem is the one for which the objective and inequality constraints are convex, and equality constraints linear.

Al optimization algorithms are iterative. The strategy used to move from one iterate to the next distinguishes one algorithm from another.

Ow, and the "programming" refers to the 40s when programming was considered an art - pun intended.

For R packages, see here: https://cran.r-project.org/web/views/Optimization.html

6.1 Object for constrained quadratic problem

```
setClass('QP',
         slots = c(H='matrix', c='numeric', A='matrix', b='numeric', I='logical'))
```

```
min$QP <- function(H, c, A = matrix(ncol = 0, nrow = 0), b = vector('numeric'),
                   I = vector()){
  QP <- new('QP', H = H, c = c, A = A, b = b, I = I)
  QP
}
min$QP(H = matrix(), c = vector('numeric'))
```

```
## An object of class "QP"
## Slot "H":
##      [,1]
## [1,]   NA
##
## Slot "c":
## numeric(0)
##
## Slot "A":
## <0 x 0 matrix>
##
## Slot "b":
## numeric(0)
##
## Slot "I":
## logical(0)
```

6.2 Unconstrained quadratic optimization

We assume the objective function:

$$f(x) = \frac{1}{2}x^T Hx + x^T c$$

```
min$f <- function(x, H, c){
  1/2 * t(x) %*% H %*% x + t(x) %*% c
}
```

This is a *convex* function if and only if G is *symmetric* and *positive semidefinite* (SPSD) [1]. Proof for convexity is given in [2].

Necessary conditions are conditions that must be satisfied by any solution point (under certain assumptions). Sufficient conditions are those that, if satisfied at a certain point x^*, guarantee that x^* is in fact a solution.

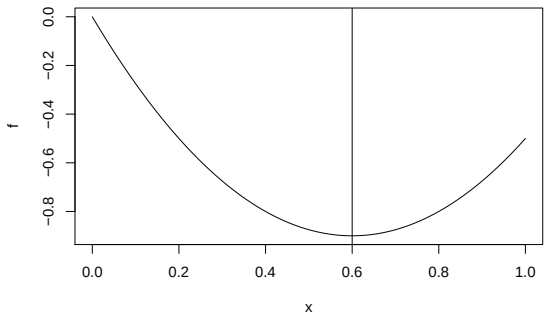
The optimality conditions in unconstrained optimization have the following properties:

- Necessary conditions: Local unconstrained minimizers have $\nabla f(x^*)$ and $\nabla^2 f(x^*)$ positive semidefinite.
- Sufficient conditions: Any point x^* at which $\nabla f(x^*)$ and $\nabla^2 f(x^*)$ is positive definite is a strong (= strict) local minimizer of f (i.e. in neighborhood of x^* $f(x)$ is strictly larger than $f(x^*)$.

6.2.1 Direct solution

Note that $\nabla f(x) = Hx + c$ in case H is symmetric. So the direct solution is solving the equations $Hx = -c$.

```
QPeU <- min$QP(H = matrix(5), c = -3)
plot(sapply(X = seq(0,1,0.01), FUN = min$f, H = QPeU@H, c = QPeU@c) ~ seq(0,1,0.01),
     type = 'l', ylab = 'f', xlab = 'x')
abline(v = -QPeU@c/QPeU@H)
```



6.3 Constrained quadratic optimization

Problem can be written as follows:

$$\min_{x \in \mathbb{R}^n} f(x) \text{ subject to } \begin{cases} c_i(x) = 0, \ i \in \mathcal{E} \\ c_i(x) \geq 0, \ i \in \mathcal{I} \end{cases}$$

The *feasable set* Ω is the set of points x that satisfy the constraints. So we can simply write

$$\min_{x \in \Omega} f(x)$$

6.3.1 Equality constraints

Equality constraints are written as $Ax = b$. We can search for solutions of the equality-constrained problem by seeking stationary points of the Lagrangian function. The Lagrangian function is defined as:

$$\mathcal{L}(x, \lambda_1) = f(x) - \lambda_1 c_1(x)$$

Note that $\nabla f(x) = Hw + c$ when H is symmetric. And thus a stationary point is:

$$\nabla_x \mathcal{L}(x^*, \lambda_1^*) = \nabla f(x) - \lambda_1 \nabla c_1(x) = Hx + c - \lambda_1 \mathbf{1} = 0$$

Furthermore the equality $Ax = b$ conditions must be met. So all this can be written in matrix form:

$$\begin{bmatrix} H & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix}$$

These conditions are a consequence of the general result for first-order optimality conditions, Theorem 12.1. [1] This coefficient matrix is called the Karush–Kuhn–Tucker (KKT) matrix. We make the KKT matrix K with the following function:

```
min$K <- function(H, A, AtMultiplier = 1){
  if (length(A) == 0) return(H)
  top <- cbind(H, AtMultiplier * t(A))
  bottom <- cbind(A, array(0, dim = c(nrow(A), ncol(top) - ncol(A))))
  rbind(top, bottom)
}
```

Assuming row independence of A and ZHZ^T positive definite implies K is non-singular and thus there is an unique solution (x^*, λ^*). Proof in [1].

```
min$Z <- function(A){
  MASS::Null(t(A))
}
```

So lets solve it:

```
min$solveKKT <- function(H, A, c, b, writeLog = FALSE){
  K <- min$K(H = H, A = A)
  g <- c(-c, b)

  sol <- tryCatch(
    expr = {
      # When the reduced hessian ZtHZ is positive definite, then there is a
      # unique solution so we can used the more accurate solve function
      solve(K, g)
    },
    error = function(err){

      if (writeLog) {
        if (length(A) == 0) {
          rH <- H
        } else {
          Z <- min$Z(A = A)
          if (length(Z) == 0) {
            rH <- H
          } else {
            rH <- t(Z) %*% H %*% Z # reduced hessian
          }
        }
        cat(paste('Reduced Hessian not positive definite. Eigenvalues:',
                  paste(eigen(rH)$values, collapse = ', '), '\n'))
      }

      # Note that this kind of inversion removes some very small eigenvalues
      # that might unbalance the inverted matrix when inverted. The effect of
      # this is that the constraints can not be guaranteed to be true. So
      # for example, the solution might sum ip to 0.3 instead of 1. Have no
      # direct solution for this problem, other than decreasing the tolerance.
      MASS::ginv(K, tol = .Machine$double.eps) %*% g
    }
  )

  structure(sol[1:length(c)], lambdas = sol[-(1:length(c))])
}
min$solveKKT(H = QPeU@H, A = QPeU@A, c = QPeU@c, b = QPeU@b)
```

```
## [1] 0.6
## attr(,"lambdas")
## numeric(0)
```

So when Z^THZ is positive definite and rows of A are independent, then first-order necessary conditions for the problem are satisfied. One can go further and say in this quadratic case that the solution is a strict local minimizer and even a global solution.

When the reduced Hessian matrix Z^TGZ is positive semidefinite with zero eigenvalues, the vector x^* satisfying (16.4) is a local minimizer but not a strict local minimizer. If the reduced Hessian has negative eigenvalues, then x^* is only a stationary point, not a local minimizer.

6.3.1.1 Example 1 using constraint $x_1 + x_2 = 1$

Define H and c. Suppose the following equality constraint: $x_1 + x_2 = 1$.

```
QPeE <- min$QP(H = matrix(c(3,2,2,4), ncol = 2),
               c = c(15, 5), A = t(c(1,1)),
               b = 1,
               I = FALSE)
QPeE
```

```
## An object of class "QP"
## Slot "H":
##      [,1] [,2]
## [1,]    3    2
## [2,]    2    4
##
## Slot "c":
## [1] 15  5
##
## Slot "A":
##      [,1] [,2]
## [1,]    1    1
##
## Slot "b":
## [1] 1
##
## Slot "I":
## [1] FALSE
```

```
eigen(QPeE@H)$values
```

```
## [1] 5.56 1.44
```

A should have full row rank.

```
Matrix::rankMatrix(QPeE@A)[1]
```

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

Null space of rows of A

```
min$Z(QPeE@A)
```

```
##         [,1]
## [1,] -0.707
## [2,]  0.707
```

Construct the KKT matrix.

```
min$K(QPeE@H, QPeE@A)
```

```
##      [,1] [,2] [,3]
## [1,]    3    2    1
## [2,]    2    4    1
## [3,]    1    1    0
```

Positive definiteness of $Z^T H Z$.

```
t(min$Z(QPeE@A)) %*% QPeE@H %*% min$Z(QPeE@A) # is 1x1 positive!
```

```
##      [,1]
## [1,]  1.5
```

Solve the equation for (x^*, λ^*):

```
min$solveKKT(QPeE@H, QPeE@A, QPeE@c, QPeE@b)
```

```
## [1] -2.67  3.67
## attr(,"lambdas")
## [1] -14.3
```

6.3.1.2 Example 2 using semidefinite H

```
QPeEsd <- QPeE
QPeEsd@H <- matrix(c(3,3,3,3), nrow = 2)
eigen(QPeEsd@H)
```

```
## $values
## [1] 6 0
##
## $vectors
##       [,1]   [,2]
## [1,] 0.707 -0.707
## [2,] 0.707  0.707
```

```
min$solveKKT(QPeEsd@H, QPeEsd@A, QPeEsd@c, QPeEsd@b)
```

```
## [1] 0.5 0.5
## attr(,"lambdas")
## [1] -13
```

6.3.1.3 Example 16.2 [1]

```
QPeEb <- min$QP(H = matrix(c(6,2,1,2,5,2,1,2,4), ncol = 3),
                c = -c(8,3,3),
                A = matrix(c(1,0,0,1,1,1), ncol = 3),
                b = c(3,0),
                I = c(FALSE, FALSE))
QPeEb
```

```
## An object of class "QP"
## Slot "H":
##      [,1] [,2] [,3]
## [1,]    6    2    1
## [2,]    2    5    2
## [3,]    1    2    4
##
## Slot "c":
## [1] -8 -3 -3
##
## Slot "A":
##      [,1] [,2] [,3]
## [1,]    1    0    1
## [2,]    0    1    1
##
## Slot "b":
## [1] 3 0
##
## Slot "I":
## [1] FALSE FALSE
```

Null space of rows of A

```
min$Z(QPeEb@A)
```

```
##         [,1]
## [1,] -0.577
## [2,] -0.577
## [3,]  0.577
```

```
Matrix::rankMatrix(QPeEb@A)[1]
```

```
## [1] 2
```

Positive definiteness of $Z^T H Z$.

```
matrixcalc::is.positive.definite(t(min$Z(QPeEb@A)) %*% QPeEb@H %*% min$Z(QPeEb@A))
```

```
## [1] TRUE
```

```
min$solveKKT(H = QPeEb@H, A = QPeEb@A, QPeEb@c, QPeEb@b)
```

```
## [1]  2 -1  1
## attr(,"lambdas")
## [1] -3  2
```

Which corresponds to solution in [1].

6.3.2 Ineqality constraints

Inequality-constrained problem differs from the equality-constrained problem in that the sign of the Lagrange multiplier plays a significant role.

6.3.2.1 Example 16.4 [1]

```
QPeIb <- min$QP(H = diag(2, 2),
                c = c(-2,-5),
                A = matrix(data = c(+1, -2,
                     -1, -2,
                     -1, +2,
                     +1, +0,
                     +0, +1), ncol = 2, byrow = TRUE),
                b = c(-2, -6, -2, 0, 0),
                I = c(TRUE, TRUE, TRUE, TRUE, TRUE))
QPeIb
```

```
## An object of class "QP"
## Slot "H":
##      [,1] [,2]
## [1,]    2    0
## [2,]    0    2
##
## Slot "c":
## [1] -2 -5
##
## Slot "A":
##      [,1] [,2]
## [1,]    1   -2
## [2,]   -1   -2
## [3,]   -1    2
## [4,]    1    0
## [5,]    0    1
##
## Slot "b":
## [1] -2 -6 -2  0  0
##
```

```
## Slot "I":
## [1] TRUE TRUE TRUE TRUE TRUE
```

```
eigen(QPeIb@H)$values # is positive definite
```

```
## [1] 2 2
```

Test that H and c are the same as with the exercise $x = (3,1)$.

```
(3-1)^2 + (1-2.5)^2
```

```
## [1] 6.25
```

```
min$f(c(3,1), QPeIb@H, QPeIb@c) + (-1)^2 + (-2.5)^2 # i.e. + the constants
```

```
##      [,1]
## [1,] 6.25
```

6.3.3 Equality and inequality constraints

We define the Lagrangian function

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x)$$

First-Order Necessary Conditions (because they are concerned with properties of the gradients) are known as the Karush–Kuhn–Tucker (KKT) conditions. Translating them to the QP problem we get:

$$\begin{aligned} Hx^* + c - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i^* a_i &= 0, \\ a_i^T x^* &= b_i, \qquad \forall i \in \mathcal{A}(x^*), \\ a_i^T x^* &\geq b_i, \quad \forall i \in \mathcal{I} \setminus \mathcal{A}(x^*), \\ \lambda_i^* &\geq 0, \quad \forall i \in \mathcal{I} \cap \mathcal{A}(x^*). \end{aligned}$$

where $\mathcal{A}(x^*) = \{i \in \mathcal{E} \cup \mathcal{I} \mid a_i^T x^* = b_i\}$, called the active set.

For convex QP, when H is positive semidefinite, these conditions are in fact sufficient for x^* to be a global solution. If H is positive definite then it is even an unique global solution. Proof in [1].

6.3.3.1 Active set algorithm

```
min$ActiveSet <- function(QP, x0, writeLog = FALSE,
                          pTol = sqrt(.Machine$double.eps)){

  # Private member placeholders
  H <- QP@H; c <- QP@c; A <- QP@A; b <- QP@b; I <- QP@I

  W <- vector('list') # working set
  x <- vector('list') # point
  p <- vector('list') # line search step
  a <- vector('list') # step size (0..1)

  kCurrent <- 0
  xFinal <- NULL

  # Private functions
  pLog <- function(string){
    if (writeLog) print(string)
  }
```

```
assertRowsIndependent <- function(Ak){
  stopifnot(qr(Ak)$rank == nrow(Ak))
}

assertEqualityConstraintsMet <- function(xk){
  if ((length(I[!I]) == 0) # no equality constraints
      || all(abs(A[!I,,drop=FALSE] %*% xk - b[!I])
             <= sqrt(.Machine$double.eps)) # all equality constraints met
      ) {}
  else {
    warning('QP: not all equality constraints met in solution!\n')
  }
}

warnANullSpaceEmpty <- function(Ak){
  if (nrow(Ak) >= length(x[[1]])) warning('Only one feasible point!\n')
}

activeConstraints <- function(xk){
  # Due to numerical issues, it might be that an equality constraint
  # for xk does not add up to exactly 0 in A %*% xk - b. That is
  # why some tolerance is used.
  acIndexes <- which(abs(A %*% xk - b) <= sqrt(.Machine$double.eps))
  # All equality constraints must be in working set (always)
  # Since equality constrains are never removed, it is enough
  # assert that they are in the working set at the start.
  if (!all(I) && !all(which(!I) %in% acIndexes)) {
    stop("min$ActiveSet: Not all equality constraints active!\n")
  }
  return(acIndexes)
}

g <- function(xk){
  H %*% xk + c
}

solvepk <- function(xk, Wk){
  min$solveKKT(H = H, A = A[Wk,,drop=FALSE],
               c = g(xk = xk),
               b = rep(0, length(Wk)))
}

pk0LagrangeMultipliers <- function(xk, Wk){
  # Rows of Working set A is are lineary independent (= requirement)
  # We transpose A here and solve for lambdas, which are thus
  # unique too.
  qr.solve(t(A[Wk,,drop=FALSE]), g(xk = xk))
}

pk0IsFoundX <- function(multipliers, Wk){
  # If all multipliers >= 0 for ONLY inequality constraints,
  # then that is the solution
  all(multipliers[I[Wk]] >= 0)
}

pk0InequalityToRemove <- function(multipliers, Wk){
  # Inequality constraint that is causing trouble.
```

```
  Imultipliers <- multipliers[I[Wk]]
  IWk <- Wk[I[Wk]]
  IWk[which.min(Imultipliers)]
}

pk0Iteration <- function(k){
  L <- pk0LagrangeMultipliers(xk = x[[k]], Wk = W[[k]])
  if (pk0IsFoundX(multipliers = L, Wk = W[[k]])) {
    pLog(paste('x* found after', kCurrent, 'steps!'))
    xFinal <<- x[[k]]
  } else {
    W[[k+1]] <<- setdiff(W[[k]], pk0InequalityToRemove(L, Wk = W[[k]]))
    x[[k+1]] <<- x[[k]]
  }
}

pkNNComputeAk <- function(xk, pk, Wk){
  Imin <- intersect(setdiff(1:length(I), Wk), which(A %*% pk < 0))
  alphas <- (b[Imin] - A[Imin,,drop=FALSE] %*% xk) /
            (A[Imin, , drop=FALSE] %*% pk)
  # We order the blocking sets according to the least alpha first.
  structure(min(1, alphas),
            'blockingSet' = Imin[alphas < 1][order(alphas[alphas < 1],
                                                   decreasing = FALSE)])
}

pkNNIteration <- function(k){
  a[[k]] <<- pkNNComputeAk(xk = x[[k]], pk = p[[k]], Wk = W[[k]])
  x[[k+1]] <<- x[[k]] + a[[k]] * as.vector(p[[k]])
  if (length(attr(a[[k]], 'blockingSet')) > 0) {
    # We take the blocking constrant of the lowest alpha. This is - unlike
    # what Nocedal (p472) says - necessary (!) since x_k+1 now satisfies
    # the constraint of the lowest alpha and not any other. Chosing the
    # wrong constraint might end up with wrong p_k+1 = 0 since the
    # wrong constraint is not satisfied by x_k+1!!!
    W[[k+1]] <<- union(W[[k]], attr(a[[k]], 'blockingSet')[1])
    assertRowsIndependent(A[W[[k]],,drop=FALSE]) # See Nocedal p.468
    warnANullSpaceEmpty(A[W[[k]],,drop=FALSE]) # Possible problem
  } else {
    W[[k+1]] <<- W[[k]]
  }
}

iterate <- function(k){
  p[[k]] <<- solvepk(xk = x[[k]], Wk = W[[k]])
  # Problem here is numerical precision
  if (all(abs(p[[k]]) < pTol)){ # if pk == 0
    pk0Iteration(k)
  } else { # pk != 0
    pkNNIteration(k)
  }
}

nextIteration <- function(){
  iterate(k = kCurrent + 1)
  kCurrent <<- kCurrent + 1
}
```

```
  iterateTillFound <- function(){
    while (is.null(xFinal)){
      nextIteration()
      if (kCurrent > 10000) {
        # This should be better examined, but sometimes due to numerical
        # issues the optimal x is found but the algorithm is stuck
        # between adding and removing the same constraint.
        # The x never changes though since at each iteration either p is 0
        # or alpha is 0. So maybe a better check would be to check x difference.
        # For debugging this case, add browser().
        # This only happens in very very boundary cases.
        warning("QP: exit from infinite loop.\n");
        xFinal <<- x[[kCurrent]]
      }
    }
    assertEqualityConstraintsMet(xFinal)
    return(xFinal)
  }

  # initialize
  x[[1]] <- x0
  W[[1]] <- activeConstraints(x[[1]])

  # public functions
  list(nextIteration = nextIteration,
       solve = iterateTillFound,
       getWorkingSetW = function(){W},
       getStepP = function(){p},
       getStepLengthAlpha = function(){a},
       getPointsX = function(){x})
}
```

6.3.3.1.1 Initial feasible point selection

Use the "Phase I" approach for linear programming described in Chapter 13 from [1].

This is actually not necessary since we allready have *hy$rW*.

6.3.3.1.2 Example 16.4 [1] continued

```
AS <- min$ActiveSet(QP = QPeIb, x0 = c(2,0))
AS$solve()
```

```
## [1] 1.4 1.7
```

For reference at each step the parameters:

```
AS$getWorkingSetW()
```

```
## [[1]]
## [1] 3 5
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 5
##
## [[4]]
```

```
## integer(0)
##
## [[5]]
## [1] 1
##
## [[6]]
## [1] 1
```

```
AS$getStepP()
```

```
## [[1]]
## [1] 6.66e-16 4.44e-16
## attr(,"lambdas")
## [1] 2 1
##
## [[2]]
## [1] -1  0
## attr(,"lambdas")
## [1] 5
##
## [[3]]
## [1] 0 0
## attr(,"lambdas")
## [1] 5
##
## [[4]]
## [1] 0.0 2.5
## attr(,"lambdas")
## numeric(0)
##
## [[5]]
## [1] 0.4 0.2
## attr(,"lambdas")
## [1] -0.8
##
## [[6]]
## [1] 1.11e-16 0.00e+00
## attr(,"lambdas")
## [1] -0.8
```

```
AS$getStepLengthAlpha()
```

```
## [[1]]
## NULL
##
## [[2]]
## [1] 1
## attr(,"blockingSet")
## integer(0)
##
## [[3]]
## NULL
##
## [[4]]
## [1] 0.6
## attr(,"blockingSet")
## [1] 1
##
## [[5]]
## [1] 1
```

```
## attr(,"blockingSet")
## integer(0)
```

```
AS$getPointsX()
```

```
## [[1]]
## [1] 2 0
##
## [[2]]
## [1] 2 0
##
## [[3]]
## [1] 1 0
##
## [[4]]
## [1] 1 0
##
## [[5]]
## [1] 1.0 1.5
##
## [[6]]
## [1] 1.4 1.7
```

6.3.3.1.3 Example 16.2 [1]

```
min$ActiveSet(QP = QPeEb, x0 = c(3,0,0))$solve()
```

```
## [1]  2 -1  1
```

6.3.3.1.4 Example 1 cont.

```
min$ActiveSet(QP = QPeE, x0 = c(1/2,1/2))$solve()
```

```
## [1] -2.67  3.67
```

6.3.3.1.5 Example 1 cont. with $x \geq 0$ (Simplex)

Extend the existing example.

```
QPeEI <- QPeE
QPeEI@A <- matrix(c(QPeEI@A, diag(2)), ncol = 2, byrow = TRUE)
QPeEI@b <- c(QPeEI@b, 0, 0)
QPeEI@I <- c(QPeEI@I, TRUE, TRUE)
QPeEI
```

```
## An object of class "QP"
## Slot "H":
##      [,1] [,2]
## [1,]    3    2
## [2,]    2    4
##
## Slot "c":
## [1] 15  5
##
## Slot "A":
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    0
## [3,]    0    1
##
```

```
## Slot "b":
## [1] 1 0 0
##
## Slot "I":
## [1] FALSE  TRUE  TRUE
```

```
min$ActiveSet(QP = QPeEI, x0 = c(1/2, 1/2))$solve()
```

```
## [1] 0 1
```

6.3.3.1.6 Example 2 cont.

```
min$ActiveSet(QP = QPeEsd, x0 = c(1/2, 1/2))$solve()
```

```
## [1] 0.5 0.5
```

6.3.3.2 Interior point algorithm

...

6.4 References

[1] Nocedal, J. e.a., Numerical Optimization, 2ed, 2006 [2] Freund, R.M., Quadratic Functions, Optimization, and Quadratic Forms, MIT, 2004

7 Samplers

7.1 Some helper functions

Sometimes we need the inverse of Σ. Since Σ can be singular, we need a unified approach for inverting it:

```
sampler$invSigma <- function(Sigma){
   tryCatch(expr = {solve(Sigma)},
            error = function(err){
              if (PARAMS$LOG_MP_INVERSE_WARNINGS)
                {warning('Sigma singular. Using MP inverse.\n')}
              MASS::ginv(Sigma)})
}
```

7.2 Metropolis-Hastings (MH) sampler

Note that the normalizing constant is not important here.

7.2.1 Initial value

Sometimes the MH can not start since the Q function within can not be resolved due to division by 0 if the *init.value* is bad. So we make a helper function which will try to find any value around the *init.value* for which the posterior has a probability higher than one.

```
sampler$MH.SeekGoodInitValue <- function(dfnPosterior, rfnProposal, initValList, ...){
  tryList <- initValList
  for (i in 1:100000) {
    if (do.call(what = dfnPosterior, args = c(tryList, list(...))) > 0) {
      cat(paste("Good posterior init value found after",i,"steps.\n"))
      return(tryList)
```

```
    }
    tryList <- do.call(what = rfnProposal, args = tryList)
  }
  stop(paste("Could not find any non-zero posterior to start MH after",i,"steps.\n"))
}
```

7.2.2 Metropolis-Hastings sampler

```
sampler$MH <- function(nrSamples, dfnPosterior, rfnProposal, initValList, burnIn = 0, ...){
  # rfnProposal should always (!) return list of parameters
  current <- sampler$MH.SeekGoodInitValue(dfnPosterior, rfnProposal, initValList, ...)
  samples <- matrix(nrow = nrSamples + burnIn,
                    ncol = length(unlist(initValList)),
                    dimnames = list(NULL, names(unlist(initValList))))
  acceptance = 0
  for (i in 1:dim(samples)[1]) {
      new <- do.call(rfnProposal, current)
      Q <- do.call(dfnPosterior, c(new, list(...)))/do.call(dfnPosterior, c(current, list(...)))
      alpha <- min(1, Q, na.rm = FALSE)
      if (runif(n = 1, min = 0, max = 1) <= alpha) {
        current <- new
        samples[i,] <- unlist(current)
        if (i > burnIn) {acceptance = acceptance + 1}
      } else {
        samples[i,] <- unlist(current)
      }
  }
  cat(paste("Acceptance ratio:",acceptance/nrSamples,"\n"))
  if (burnIn > 0) { samples <- samples[-c(1:burnIn), ] }
  return(structure(samples, 'skelet' = as.relistable(initValList)))
}
```

7.3 Multivariate Normal

7.3.1 Multivariate Normal density

```
sampler$dmnorm <- function(x, mean, Sigma){
  stopifnot( length(x) == length(mean) )

  detSigma <- det(Sigma)
  if (detSigma < 0) {browser()}

  invSigma <- sampler$invSigma(Sigma)

  e <- as.vector( exp( -1/2 * t(x-mean) %*% invSigma %*% (x-mean) ) )
  e / sqrt( (2 * pi) ^ length(x) * detSigma )
}
sampler$dmnorm(x = c(1,2), mean = c(1,4), Sigma = diag(2, nrow = 2))
```

```
## [1] 0.0293
```

```
mvtnorm::dmvnorm(x = c(1,2), mean = c(1,4), sigma = diag(2, nrow = 2))
```

```
## [1] 0.0293
```

```
system.time( replicate(3000, sampler$dmnorm(x = runif(5), mean = runif(5),
                                            Sigma = diag(runif(5))) ) )
```

```
##    user  system elapsed
##    0.25    0.01    0.27
```

```
system.time( replicate(3000, mvtnorm::dmvnorm(x = runif(5), mean = runif(5),
                                              sigma = diag(runif(5))) ) )
```

```
##    user  system elapsed
##    0.36    0.00    0.36
```

7.4 Truncated Univariate Normal

7.4.1 Truncated Univariate Normal sampler

```
sampler$rtnorm <- function(mean, sigma2, lbound = -Inf, ubound = Inf){
  stopifnot(length(mean) == 1)
  stopifnot(length(sigma2) == 1)
  truncnorm::rtruncnorm(n = 1, mean = mean, sd = sqrt(sigma2), a = lbound, b = ubound)
}
sampler$rtnorm(mean = 2, sigma2 = 0.1, lbound = 0, ubound = 1)
```

```
## [1] 0.959
```

7.5 Simplex Multivariate Normal

7.5.1 Simplex Multivariate Normal sampler (Gibbs)

```
# This is a mutivariate normal sampler constrained on a simplex
sampler$rsmnorm.Gibbs <- function(Mu, Sigma, previousX){
  M <- length(Mu)

  # Transformation equations for y = Ax + b, where thus y is independent
  A <- matrix(data = c(diag(M-1), rep(-1,M-1)), ncol = M-1, byrow = TRUE)
  b <- c(rep(0, M-1), 1)

  SigmaInv <- solve(Sigma)
  SigmaYInv <- ( t(A) %*% SigmaInv %*% A )
  SigmaY <- solve(SigmaYInv) # Never singular, probably due to A transformation.
  # MuY has muY in columns! t(gX) - gE %*% b works because gE %*% b is recycled
  # for each column of t(gX)
  MuY <- SigmaY %*% t(A) %*% SigmaInv %*% (Mu - b)

  Y <- previousX[-M]

  for (k in 1:(M-1)) {
    sigma2Yk <- 1/SigmaYInv[k,k]
    muYk <- ( t(SigmaYInv[k,]) %*% MuY - SigmaYInv[k,-k] %*% Y[-k] ) * sigma2Yk
    uBound <- 1 - sum(Y[-k])
    Y[k] <- sampler$rtnorm(mean = muYk, sigma2 = sigma2Yk, lbound = 0, ubound = uBound)
  }

  return(c(Y, 1-sum(Y)))
}
sampler$rsmnorm.Gibbs(Mu = c(1,4,8), Sigma = diag(2,3), previousX = c(0.5,0.5,0))
```

```
## [1] 0.00596 0.36013 0.63390
```

```
sum(sampler$rsmnorm.Gibbs(Mu = c(1,4,8), Sigma = diag(2,3), previousX = c(0.5,0.5,0)))
```

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

7.5.1.1 Test: Compute expectation and variance

```
tmp <- matrix(NA, ncol = 3, nrow = 10000)
tmp[1,] <- hy$rW(N = 1, M = 3)
for (i in 2:10000) {
  tmp[i,] <- sampler$rsmnorm.Gibbs(Mu = c(0.7,2,0),
                                   Sigma = rbind(c( 0.8, 0.5,-0.3),
                                                 c( 0.5, 0.5,-0.1),
                                                 c(-0.3,-0.1, 0.6)),
                                   previousX = tmp[i-1,])
}
colMeans(tmp)
```

```
## [1] 0.109 0.743 0.148
```

```
apply(tmp, 2, var)
```

```
## [1] 0.00945 0.02323 0.01710
```

7.5.1.2 Test: Visualize distribution

```
tmp <- matrix(NA, ncol = 2, nrow = 100000)
tmp[1,] <- hy$rW(N = 1, M = 2)
for (i in 2:100000) {
  tmp[i,] <- sampler$rsmnorm.Gibbs(Mu = c(0.7,2), Sigma = cbind(c(0.8,0.5),c(0.5,0.5)),
                                   previousX = tmp[i-1,])
}
colMeans(tmp)

tmp <- plyr::count(round(tmp[-c(1:50),], 2))
tmp <- tmp[tmp$x.1 != 0 & tmp$x.2 != 0,]
tmp$freq <- tmp$freq/sum(tmp$freq)
rgl::persp3d(Vectorize(function(x,y,...){
  0.47*sampler$dmnorm(x = c(x,y), mean = c(0.7,2), Sigma = cbind(c(0.8,0.5),c(0.5,0.5)))}),
  xlim=c(-0,5), ylim=c(-0,5), zlim=c(-0.000001,0.08))
rgl::plot3d(x = tmp)
rgl::planes3d(0.5, 0.5, 0, -0.5, col = 'red', alpha = 0.6)
```

7.5.2 Simplex Multivariate Normal sampler (MH)

```
sampler$rsmnorm.MH <- function(Mu, Sigma, previousX){
  dnkernel <- function(x, mu, invCov){ # kernel of normal multivariate density
    return( -1/2 * t(x-mu) %*% invCov %*% (x-mu) )
  }

  M <- length(Mu)
  SigmaInv <- solve(Sigma)

  new <- hy$rdirichlet(alpha = rep(1, M))
  Q <- exp(dnkernel(new, Mu, SigmaInv) - dnkernel(previousX, Mu, SigmaInv))
  alpha <- min(1, Q, na.rm = FALSE)
  if (runif(n = 1, min = 0, max = 1) <= alpha) {
```

```
    return(new)
  } else {
    return(previousX)
  }
}
sampler$rsmnorm.MH(Mu = c(1,4,8), Sigma = diag(2,3), previousX = c(0.5,0.5,0))
```

```
## [1] 0.205 0.371 0.425
```

```
sum(sampler$rsmnorm.MH(Mu = c(1,4,8), Sigma = diag(2,3), previousX = c(0.5,0.5,0)))
```

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

7.5.2.1 Test: Compute expectation and variance

```
tmp <- matrix(NA, ncol = 3, nrow = 10000)
tmp[1,] <- hy$rW(N = 1, M = 3)
for (i in 2:10000) {
  tmp[i,] <- sampler$rsmnorm.MH(Mu = c(0.7,2,0),
                                Sigma = rbind(c( 0.8, 0.5,-0.3),
                                              c( 0.5, 0.5,-0.1),
                                              c(-0.3,-0.1, 0.6)),
                                previousX = tmp[i-1,])
}
colMeans(tmp)
```

```
## [1] 0.111 0.751 0.139
```

```
apply(tmp, 2, var)
```

```
## [1] 0.0092 0.0225 0.0160
```

7.6 Inverse-Gamma

7.6.1 Inverse-Gamma density

Note here that α = shape and β = rate.

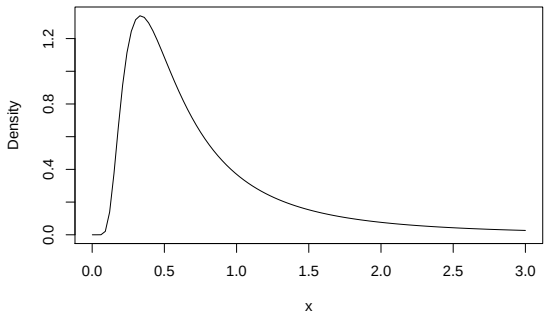
```
sampler$dinvgamma <- function(x, alpha, beta){
  beta^alpha/gamma(alpha) * x^(-alpha -1) * exp(-beta/x)
}
sampler$dinvgamma(1, alpha = 2.01, beta = 1.01)
```

```
## [1] 0.37
```

```
invgamma::dinvgamma(1, shape = 2.01, rate = 1.01)
```

```
## [1] 0.37
```

```
plot(Vectorize(function(x){sampler$dinvgamma(x, alpha = 2.01, beta = 1.01)}),
     from = 0.001, to = 3, ylab = 'Density')
```



7.6.2 Inverse-Gamma sampler

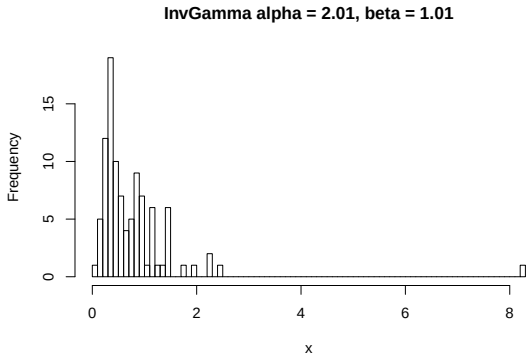
```
sampler$rinvgamma <- function(alpha, beta){
  1/rgamma(n = 1, shape = alpha, rate = beta)
}
sampler$rinvgamma(alpha = 2.01, beta = 1.01)
```

```
## [1] 0.708
```

```
invgamma::rinvgamma(1, shape = 2.01, rate = 1.01)
```

```
## [1] 0.596
```

```
hist(sapply(1:100,function(x){sampler$rinvgamma(alpha = 2.01, beta = 1.01)}),
     main = 'InvGamma alpha = 2.01, beta = 1.01', xlab = 'x', breaks = 100)
```



7.6.3 Performance test

```
system.time( replicate(10000, invgamma::rinvgamma(1, shape = 2.01, rate = 1.01) ) )
```

```
##    user  system elapsed
##    0.14    0.00    0.14
```

```
system.time( replicate(10000, sampler$rinvgamma(alpha = 2.01, beta = 1.01) ) )
```

```
##    user  system elapsed
##    0.05    0.00    0.05
```

8 Parallel processing

See http://gforge.se/2015/02/how-to-go-parallel-in-r-basics-tips/.

8.1 Helper function

8.1.1 Setting up clusters

```
pp$setup <- function(cores) {
  if (exists(envir = pp, x = 'cl') && !is.null(pp$cl)) {
    pp$stop()
  }
  pp$cl <<- parallel::makeCluster(cores, outfile = 'parallel_debug.txt')
  doParallel::registerDoParallel(pp$cl) # needed by foreach::foreach

  # set different seeds to each worker so no correlation exists
  parallel::parSapply(cl = pp$cl, X = 1:cores, FUN = function(i){set.seed(i)})

  invisible()
}
```

8.1.2 Updating clusters

```
pp$update <- function(){
  # copy everything except IM and QP objects
  parallel::clusterExport(pp$cl, setdiff(ls(envir = .GlobalEnv, all.names = TRUE),
                                         ls(envir = .GlobalEnv, pattern = '(^IM)|(^QP)|(^pp$)')))

  # some stuff doesn't get exported well
  # see here: https://stackoverflow.com/q/44110980/1548942
  if (exists(x = 'hyperSpecPred-ICE', envir = `.__T__$:base`)) {
    parallel::clusterEvalQ(pp$cl, setMethod(f = '$', signature = 'hyperSpecPred-ICE',
                                            definition = `.__T__$:base`$`hyperSpecPred-ICE`))
  }

  if (exists(x = 'hyperSpecPred-BayesNMF-Vol', envir = `.__T__$:base`)) {
    parallel::clusterEvalQ(pp$cl, setMethod(f = '$', signature = 'hyperSpecPred-BayesNMF-Vol',
                                            definition = `.__T__$:base`$`hyperSpecPred-BayesNMF-Vol`
  }
}
```

8.1.3 Stopping clusters

```
pp$stop <- function() {
  parallel::stopCluster(pp$cl)
  #showConnections(all = TRUE)
  #closeAllConnections()
  pp$cl <<- NULL
}
```

8.2 Setup parallel environment

```
pp$setup(PARAMS$PPCORES)
pp$update()
```

9 The prediction objects

The base class is defined in *HyperImage.rmd*.

9.1 The ICE Prediction object

```
setClass('hyperSpecPred-ICE', contains = 'hyperSpecPred',
         slots = c(E = 'E', W = 'W', steps='numeric', iterates='list'),
         prototype = prototype(type = 'ICE'))
```

```
alg$predictionICE <- function(E, W, steps, iterates = list()){
  new('hyperSpecPred-ICE', E = E, W = W, steps = steps, iterates = iterates)
}
alg$predictionICE(E = hy$rE(M = 3, D = 2), W = hy$rW(M = 3, N = 4), steps = 20)
```

```
## An object of class "hyperSpecPred-ICE"
## Slot "E":
## An object of class "E"
##          e1  e2  e3
## [1,]   9.61 119 122
## [2,] 239.58 196 206
##
## Slot "W":
## An object of class "W"
##       [,1]   [,2]   [,3]
## [1,] 0.716 0.0129 0.2708
## [2,] 0.539 0.2947 0.1667
## [3,] 0.134 0.7693 0.0967
## [4,] 0.217 0.3360 0.4470
## Slot "width":
## numeric(0)
##
## Slot "height":
## numeric(0)
##
## Slot "origin":
## [1] "rW"
##
##
## Slot "steps":
## [1] 20
##
## Slot "iterates":
## list()
##
## Slot "type":
## [1] "ICE"
##
## Slot "height":
## numeric(0)
```

```
##
## Slot "width":
## numeric(0)
##
## Slot "eLabels":
## character(0)
```

9.1.1 Extraction function

```
setMethod("$", 'hyperSpecPred-ICE',
function(x, name)
{
  ext <- slot(x, name)

  if (name == 'E' || name == 'W') {
    colnames(ext) <- x@eLabels
  }

  return(ext)
})
```

```
## [1] "$"
```

9.2 Samples object

```
# Each S-matrix has rows as a sample sequence from which the skelet can make a
# list of objects with the relist command.
setClass('hyperSpecPred-samples',
         slots = c(S1 = 'matrix', S2 = 'matrix', skelet = 'list'))
```

```
alg$predictionSamples <- function(S1, S2 = matrix(nrow = 0, ncol = 0),
                                  skelet){
  new('hyperSpecPred-samples', S1 = S1, S2 = S2, skelet = skelet)
}
alg$predictionSamples(S1 = matrix(), skelet = list())
```

```
## An object of class "hyperSpecPred-samples"
## Slot "S1":
##      [,1]
## [1,]   NA
##
## Slot "S2":
## <0 x 0 matrix>
##
## Slot "skelet":
## list()
```

9.3 The BayesNMF-Vol prediction object

```
setClass('hyperSpecPred-BayesNMF-Vol', contains = 'hyperSpecPred',
         slots = c(samples='hyperSpecPred-samples', .precomputedMap = 'numeric',
                   defaultEstimatorType = 'character', defaultSampleSequence = 'character'),
         prototype = prototype(type = 'BayesNMF-Vol', defaultEstimatorType = 'MAP',
                               defaultSampleSequence = 'S1'))
```

```
new('hyperSpecPred-BayesNMF-Vol', samples = alg$predictionSamples(matrix(), skelet = list()))
```

```
## An object of class "hyperSpecPred-BayesNMF-Vol"
## Slot "samples":
## An object of class "hyperSpecPred-samples"
## Slot "S1":
##      [,1]
## [1,]   NA
##
## Slot "S2":
## <0 x 0 matrix>
##
## Slot "skelet":
## list()
##
##
## Slot ".precomputedMap":
## numeric(0)
##
## Slot "defaultEstimatorType":
## [1] "MAP"
##
## Slot "defaultSampleSequence":
## [1] "S1"
##
## Slot "type":
## [1] "BayesNMF-Vol"
##
## Slot "height":
## numeric(0)
##
## Slot "width":
## numeric(0)
##
## Slot "eLabels":
## character(0)
```

```
alg$predictionBayes <- function(samples, height, width){
  new('hyperSpecPred-BayesNMF-Vol', samples = samples,
      height = height, width = width)
}
alg$predictionBayes(samples = alg$predictionSamples(matrix(), skelet = list()),
                    height = 2, width = 2)
```

```
## An object of class "hyperSpecPred-BayesNMF-Vol"
## Slot "samples":
## An object of class "hyperSpecPred-samples"
## Slot "S1":
##      [,1]
## [1,]   NA
##
## Slot "S2":
## <0 x 0 matrix>
##
## Slot "skelet":
## list()
##
##
```

```
## Slot ".precomputedMap":
## numeric(0)
##
## Slot "defaultEstimatorType":
## [1] "MAP"
##
## Slot "defaultSampleSequence":
## [1] "S1"
##
## Slot "type":
## [1] "BayesNMF-Vol"
##
## Slot "height":
## [1] 2
##
## Slot "width":
## [1] 2
##
## Slot "eLabels":
## character(0)
```

9.3.1 Extraction function

```
setMethod("$", 'hyperSpecPred-BayesNMF-Vol',
function(x, name)
{
  switch(x@defaultEstimatorType,
         'mean' = {
            ext <- relist(colMeans(slot(x@samples, x@defaultSampleSequence)),
                          x@samples@skelet)[[name]]
         },
         'MAP' = {
            if (length(x@.precomputedMap) == 0) {
              x@.precomputedMap <-
                apply(X = slot(x@samples, x@defaultSampleSequence),
                      MARGIN = 2,
                      FUN = function(x){
                        dx <- density(x, bw = PARAMS$KERNELSBW)
                        dx$x[which.max(dx$y)]}
                      )
            }
            ext <- relist(x@.precomputedMap, x@samples@skelet)[[name]]
         })
    if (name == 'E') {
      colnames(ext) <- x@eLabels
    }

    if (name == 'W') {
      ext <- hy$W(width = x@width, height = x@height, data = ext, M = ncol(ext),
                  origin = 'hyperSpecPred-BayesNMF-Vol')
      colnames(ext) <- x@eLabels
    }

    return(ext)
})
```

```
## [1] "$"
```

10 Iterated constrained endmembers (ICE) algorithm

ICE estimates pure components, or endmembers, via alternating least square.

Estimation is (possibly) carried out in a subspace of reduced dimensionality defined by the minimum noise fraction (MNF) transform.

A difficult issue is the reliable and automated estimation of the number of endmembers from data. We have not solved this problem. [1]

10.1 Objective function

This is the general loss function as used in [1] for estimating $\boldsymbol{E}$ and $\boldsymbol{W}$ subject to constraints.

$$L = \|\boldsymbol{X} - \boldsymbol{W}\boldsymbol{E}^T\|_F^2$$

where $\boldsymbol{\Sigma}$ is taken to be a identiy matrix $\boldsymbol{I}_d$[1].

```
alg$ice$L <- function(X, E, W){
  norm(X - W %*% t(E), type = 'F')^2
}
alg$ice$L(X = IM30x30D2[[]], W = IM30x30D2@W, E = IM30x30D2@E)
```

```
## [1] 1.88
```

10.2 Underspecification

Any simplex E which encapsulates the data vectors will result in the same objective function value.

10.3 Regularization

Penalty term should be proportional to the size of the simplex. This is the volume. The sum of squared distances between all the simplex vertices (i.e. their variance) is proportional to the simplex volume [2].

```
alg$ice$V <- function(E){
  vol$berman(E)
}
alg$ice$V(Ed2)
```

```
## [1] 0.237
```

10.4 Volume-regularized objective function

Objective function should be approximately independent of the sample size N, and the number of endmembers M. That is why Berman divides L by N.

$$L_{reg} = N^{-1}(1-\mu)L + \mu V \tag{5}$$

After analysing several dozen real-world datasets, Berman has found that a value of $\mu = 0.01$ usually gives reasonable solutions. [1]

[1] This is the assumption all ALS methods make.

```
alg$ice$Lreg <- function(X, E, W, mu, divisionByB = TRUE){
  N <- nrow(X); stopifnot(length(N) == 1)
  B <- ncol(X)
  L <- alg$ice$L(X = X, W = W, E = E)
  V <- alg$ice$V(E = E)
  Lreg <- 1/N * (1-mu)*L + mu*V
  if (divisionByB) { Lreg <- Lreg / B }
  Lreg
}
alg$ice$Lreg(X = IM30x30D2[[]], W = IM30x30D2@W, E = IM30x30D2@E, mu = 0.01)
```

```
## [1] 0.00222
```

```
system.time(replicate(10, alg$ice$Lreg(X = IM30x30D2[[]], W = IM30x30D2@W,
                                       E = IM30x30D2@E, mu = 0.01)))
```

```
##    user  system elapsed
##    0.02    0.00    0.02
```

10.5 Minimization with ALS

As do most MCR algorithms, ICE uses an ALS technique to minimize equation 5.

Since we have two matrices $\boldsymbol{E}$ and $\boldsymbol{W}$ to solve, the algorithm will alternate between the two, following these steps:

1. Start with random $\boldsymbol{W}$;
2. Compute $\boldsymbol{E}$;
3. Compute $\boldsymbol{W}$;
4. Repeat step (2) and (3) until convergence.

10.5.1 Compute $\boldsymbol{W}$

The objective function in equation 5 can be reduced to a quadratic programming (QP) problem [3]:

$$\underset{\boldsymbol{w}_i}{\operatorname{argmin}}\ S_{reg}(\boldsymbol{w}_i) = \underset{\boldsymbol{w}_i}{\operatorname{argmin}}\ \frac{1}{2}\boldsymbol{w}_i^T \boldsymbol{H}\boldsymbol{w}_i + \boldsymbol{w}_i^T \boldsymbol{c}$$
$$\text{subject to } \sum_m w_{nm} = 1 \text{ and } w_{nm} \geq 0,\ m = 1, \ldots, M.$$

So we write a function that will translate the problem into the above form. Note that due to stability reasons H and c are divided with B. This is described in thesis, as is the derivation of H and c.

```
alg$ice$translateToQP <- function(E, xi){
  M <- ncol(E)
  B <- nrow(E)

  # Division by B required for balance of KKT matrix. It doesn't affect
  # the solution in any negative way. (i.e. the solution stays the same.)

  min$QP(H = t(E) %*% E / B,
         c = as.vector(- t(E) %*% xi) / B,
         A = matrix(c(rep(1, M), diag(M)), ncol = M, byrow = TRUE),
         b = c(1, rep(0, M)), I = c(FALSE, rep(TRUE, M)))
}
alg$ice$translateToQP(E = IM2x2D1@E, xi = IM2x2D1[[]][1])
```

```
## An object of class "QP"
## Slot "H":
##      [,1] [,2]
## [1,] 0.01 0.05
## [2,] 0.05 0.25
##
## Slot "c":
## [1] -0.0201 -0.1006
##
## Slot "A":
##      [,1] [,2]
## [1,]    1    1
## [2,]    1    0
## [3,]    0    1
##
## Slot "b":
## [1] 1 0 0
##
## Slot "I":
## [1] FALSE  TRUE  TRUE
```

```
system.time(replicate(100, alg$ice$translateToQP(E = IM30x30D2@E,
                                                 xi = IM30x30D2[[]][1,])))
```

```
##    user  system elapsed
##    0.14    0.00    0.14
```

Lets test it.

```
min$ActiveSet(QP = alg$ice$translateToQP(E = IM2x2D2@E, xi = IM2x2D2[[]][1,]),
              x0 = as.vector(hy$rW(N = 1, M = 3)))$solve()
```

```
## [1] 0.856 0.144 0.000
```

Since $\boldsymbol{w}_i$ are independent of each other, we compute them one by one in a function that minimizes $\boldsymbol{W}$.

```
alg$ice$minimizeW <- function(X, E, Winit = hy$W(data = hy$rW(N = 1, M = ncol(E)),
                                                 M = ncol(E), N = nrow(X), byrow = TRUE)){
  N <- nrow(X)
  M <- ncol(E)
  W <- hy$W(Winit, N = N, M = M, origin = 'ICE_W_MINIMIZER',
            width = attr(Winit, 'width'), height = attr(Winit, 'height'))

  for (i in 1:N) {
    QP <- alg$ice$translateToQP(E = E, xi = X[i,])
    W[i,] <- min$ActiveSet(QP = QP, x0 = W[i,])$solve()
  }

  #W[W<0] <- 0 # very small rounding errors can occur, like (-2.50e-17, 0, 1.000)

  return(W)
}
alg$ice$minimizeW(X = IM2x2D2[[]], E = IM2x2D2@E)
```

```
## An object of class "W"
##       [,1]   [,2]  [,3]
## [1,] 0.856 0.1441 0.000
## [2,] 0.368 0.2120 0.420
## [3,] 0.442 0.4163 0.141
## [4,] 0.738 0.0504 0.212
## Slot "width":
```

```
## numeric(0)
##
## Slot "height":
## numeric(0)
##
## Slot "origin":
## [1] "ICE_W_MINIMIZER"
```

```
rowSums(alg$ice$minimizeW(X = IM30x30D2[[]], E = IM30x30D2@E)) # all 1's
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [176] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [211] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [246] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [281] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [316] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [351] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [386] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [421] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [456] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [491] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [526] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [561] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [596] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [631] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [666] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [701] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [736] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [771] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [806] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [841] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [876] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
system.time(replicate(1, alg$ice$minimizeW(X = IM30x30D2[[]], E = IM30x30D2@E)))
```

```
##    user  system elapsed
##    0.83    0.00    0.83
```

10.5.2 Compute E

```
alg$ice$minimizeE <- function(W, X, mu){
  N <- nrow(X)
  M <- ncol(W)
  B <- ncol(X)
  lambda <- (N*mu)/((M-1)*(1-mu))

  t(X) %*% W %*% solve(t(W) %*% W + lambda * (diag(M) - matrix(1, M, M)/M))
}
alg$ice$minimizeE(W = IM2x2D2@W, X = IM2x2D2[[]], mu = 0.01)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.223 0.802 0.313
## [2,] 0.155 0.440 0.880
```

```
system.time(replicate(1, alg$ice$minimizeE(X = IM30x30D2[[]], mu = 0.005,
                                           W = hy$rW(M = 3, N = 30*30))))
```

```
##    user  system elapsed
##    0.05    0.00    0.04
```

10.5.3 ALS

First a small reusable logging function

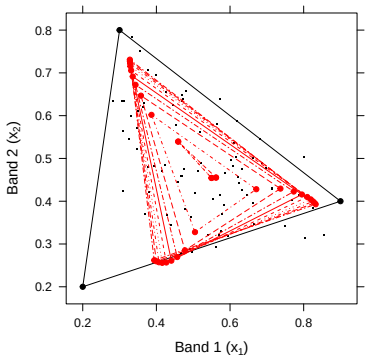
```
alg$ice$logStep <- function(step, L, E = NULL, logFreq = 10){
  if (step %% logFreq == 0) {
    cat(paste('ICE step:', sprintf("%04d", step),
              '| Imp:', L[[step]]-L[[max(1, step-logFreq)]],
              if (!is.null(E)) {paste('| SAM:', measure$SAM(E[[step]], E[[step-1]]))},
              '| Ratio:', L[[step]]/L[[step-1]],
              '\n'))
  }
}
```

And finally the ALS minimization

```
alg$ice$ICE <- function(hyImg = NULL, X = hyImg[[]], mu = 0.01,
                        tol = PARAMS$ICETOL, stepTol = 10000,
                        height = if (!is.null(hyImg)) {hy$height(hyImg)} else {numeric(0)},
                        width = if (!is.null(hyImg)) {hy$width(hyImg)} else {numeric(0)},
                        M = ncol(X) + 1, Einit = NULL, Winit = NULL){

  if (!is.null(Einit) && !is.null(Winit)) {stop('E and W can not both be specified!')}

  L <- vector('list'); E <- vector('list'); W <- vector('list')
  k <- 1

  if (!is.null(Einit)) {
    E[[k]] <- Einit
    W[[k]] <- alg$ice$minimizeW(X = X, E = E[[k]])
  } else {
    if (!is.null(Winit)) {
      W[[k]] <- hy$W(Winit, width = width, height = height) # add height and width
    } else {
      W[[k]] <- hy$rW(M = M, N = nrow(X), height = height, width = width)
    }
    E[[k]] <- alg$ice$minimizeE(W = W[[k]], X = X, mu = mu)
  }

  L[[k]] <- alg$ice$Lreg(X = X, E = E[[k]], W = W[[k]], mu = mu)

  repeat {
    k <- k + 1
    E[[k]] <- alg$ice$minimizeE(W = W[[k-1]], X = X, mu = mu)
    # It is not necessary to give the last W as initial values, but
    # seems appropriate since the solution is probably close.
    W[[k]] <- alg$ice$minimizeW(X = X, E = E[[k]], Winit = W[[k-1]])
    L[[k]] <- alg$ice$Lreg(X = X, E = E[[k]], W = W[[k]], mu = mu)
    if (PARAMS$LOG) alg$ice$logStep(step = k, L = L, E = E)
    stopifnot(L[[k]]/L[[k-1]] <= 1) # coordinate descent function value check
    if (L[[k]]/L[[k-1]] > tol || k >= stepTol) {break}
  }
```

```
  cat(paste("ICE convergence after",k,"steps\n"))

  alg$predictionICE(E = hy$E(E[[k]], wavelengths = hyperSpec::wl(hyImg)),
                    W = W[[k]], steps = k,
                    iterates = list(E = E, W = W, L = L))
}
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E,
               Ehat = Filter(Negate(is.null),
                             alg$ice$ICE(hyImg = IM9x9D2, mu = 0.01,
                                         tol = 0.99)@iterates[['E']]),
               EhatCol = rep('red', 1000), legendLog = FALSE)
```

```
## ICE convergence after 11 steps
```



10.6 References

- Berman, M. e.a., ICE a new method for the multivariate curve resolution of hyperspectral images, 2008
- Berman, M. e.a., ICE An Automated Statistical Approach to Identifying Endmembers in Hyperspectral Images, 2003
- Nocedal, J. e.a., Numerical Optimization, 2ed, 2006

10.7 Testing

10.7.1 Checking correctness of found minima

Test to see whether the found minima is actually the minima of L_reg!

```
mn <- .Machine$double.xmax
for (i in 1:5000) {
  mn <- min(mn,
            alg$ice$Lreg(X = IM2x2D2[[]][1,,drop=FALSE], E = IM2x2D2@E, mu =  0.01,
                         W = t(hy$rdirichlet(alpha = c(1,1,1)))))
}
mn
```

```
## [1] 0.00134
```

```
alg$ice$Lreg(X = IM2x2D2[[]][1,,drop=FALSE], E = IM2x2D2@E, mu =  0.01,
             W = min$ActiveSet(QP = alg$ice$translateToQP(E = IM2x2D2@E, xi = IM2x2D2[[]][1,]),
                               x0 = as.vector(hy$rW(N = 1, M = 3)))$solve())
```

```
## [1] 0.00132
```

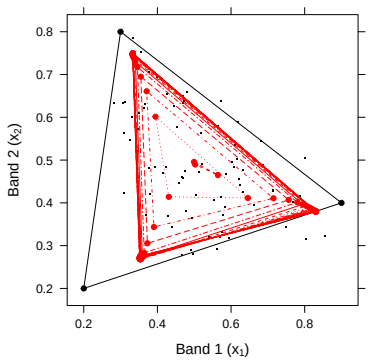
```
rm(mn);rm(i)
```

10.7.2 Minimization process visualized

```
IM9x9D2@pred[['ICE']] <- alg$ice$ICE(hyImg = IM9x9D2, mu = 0.01)
```

```
## ICE convergence after 31 steps
```

```
IM9x9D2@pred[['ICE']]@iterates$E[[1]] <- NULL
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E, legendLog = FALSE,
               Ehat = IM9x9D2@pred[['ICE']]@iterates$E, EhatCol = rep('red', 1000),
               printAs = 'ice-minimization-process')
```




```
## pdf
##   2
```

11 ICE Extended model with Spatial information

11.1 Regularization

```
# returns the indices n-H, n-1, n, n+1, n+H
alg$iceS$u <- function(n, height, N, uTilde = FALSE){
  if( n > N || length(height) == 0 || height == 0 || height > N) browser()
  u <- c(n-height, n-1, n, n+1, n+height)
  mod <- n %% height
  filter <- c(addMinH = n-height >= 1,
              addMin1 = !(mod == 1),
              addn = !uTilde,
              addPlus1 = !(mod == 0),
              addPlusH = n+height <= N)
  u[filter]
}
alg$iceS$u(n = 2, height = 2, N = 4)
```

```
## [1] 1 2 4
```

```
alg$iceS$u(n = 2, height = 2, N = 4, uTilde = TRUE)
```

```
## [1] 1 4
```

```
system.time(replicate(1000, alg$iceS$u(n = 357, height = 50, N = 50*50) ))
```

```
##    user  system elapsed
##    0.01    0.00    0.02
```

U is a matrix with in its rows $(n-H, n-1, n, n+1, N+H)$ from W. The columns are m-components. So it is just a subset of W.

```
alg$iceS$U <- function(k, W, height, N = nrow(W), nToOmmit = -1){
  K <- alg$iceS$u(n = k, height = height, N = N, uTilde = FALSE)
  W[K[K != nToOmmit],,drop=FALSE]
}
alg$iceS$U(k = 1, W = IM2x2D2@W, height = 2)
```

```
##       [,1]  [,2]   [,3]
## [1,] 0.751 0.168 0.0811
## [2,] 0.392 0.267 0.3404
## [3,] 0.472 0.399 0.1294
```

The spatial information is the sum of all S_n.

```
alg$iceS$S <- function(W){
  Sn <- function(n, W, height = attr(W, 'height'), N = nrow(W)){
    U <- W[alg$iceS$u(n = n, height = height, N = N),]
    K <- nrow(U)
    # Quicker way to compute variance than using Var function
    sum(diag(t(U) %*% (diag(K) - matrix(1, nrow = K, ncol = K)/K) %*% U)/(K-1)) # M sums
  }

  N <- nrow(W)
  height <- attr(W, 'height')
  stopifnot(length(height) > 0)

  sum(sapply(X = 1:N, FUN = Sn, W = W, N = N, height = height))
}
alg$iceS$S(W = IM9x9D2@W)
```

```
## [1] 11.2
```

```
system.time(replicate(100, alg$iceS$S(W = IM2x2D2@W)))
```

```
##    user  system elapsed
##    0.02    0.00    0.01
```

```
system.time(replicate(10, alg$iceS$S(W = IM9x9D2@W)))
```

```
##    user  system elapsed
##    0.05    0.00    0.04
```

```
system.time(replicate(1, alg$iceS$S(W = IM30x30D2@W)))
```

```
##    user  system elapsed
##    0.05    0.00    0.04
```

11.2 Objective function

```
alg$iceS$Lreg <- function(X, E, W, mu, nu){
  N <- nrow(X); stopifnot(length(N) == 1)
  B <- ncol(X)
  M <- ncol(E)
  L <- alg$ice$L(X = X, W = W, E = E)
  V <- alg$ice$V(E = E)
  S <- alg$iceS$S(W = W)
  (1-mu)*L/(N*B) + mu*(nu/B*V + (1-nu)/(M*N)*S)
}
```

```
alg$iceS$Lreg(X = IM30x30D2[[]], W = IM30x30D2@W,
              E = IM30x30D2@E, mu = 0.01, nu = 0.99)
```

```
## [1] 0.00221
```

```
system.time(replicate(1, alg$iceS$Lreg(X = IM30x30D2[[]], W = IM30x30D2@W,
                                       E = IM30x30D2@E, mu = 0.01, nu = 0.99)))
```

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

11.3 Compute W

First the translation into QP.

```
alg$iceS$translateToQP <- function(E, xn, W, mu, nu, n, height = attr(W, 'height')){
  stopifnot(length(height) > 0 || height > 0)
  M <- ncol(E)
  B <- nrow(E)
  N <- nrow(W)
  KSet <- alg$iceS$u(n = n, height = height, N = N)
  USet <- lapply(KSet, alg$iceS$U, nToOmmit = n, W = W, N = N, height = height)
  Kk <- sapply(USet, nrow) + 1 # since n is ommited in USet: +1

  min$QP(H = ( (1-mu)/B * t(E) %*% E
               + mu*(1-nu)/M * ( diag(M)*sum(1/Kk) ) ),
         c = - as.vector( (1-mu)/B * t(E) %*% xn
                          + mu*(1-nu)/M * colSums(t(sapply(USet, colSums))/(Kk*(Kk-1))) ),
         A = matrix(c(rep(1, M), diag(M)), ncol = M, byrow = TRUE),
         b = c(1, rep(0, M)), I = c(FALSE, rep(TRUE, M)))
}
alg$iceS$translateToQP(E = IM30x30D2@E, xn = IM30x30D2[[]][2,], W = IM30x30D2@W,
                       mu = 0.4, nu = 0.6, n = 2)
```

```
## An object of class "QP"
## Slot "H":
##        [,1]  [,2]  [,3]
## [1,] 0.0791 0.078 0.066
## [2,] 0.0780 0.346 0.177
## [3,] 0.0660 0.177 0.274
##
## Slot "c":
## [1] -0.0633 -0.1579 -0.1786
##
## Slot "A":
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    1    0    0
## [3,]    0    1    0
## [4,]    0    0    1
##
## Slot "b":
## [1] 1 0 0 0
##
## Slot "I":
## [1] FALSE  TRUE  TRUE  TRUE
```

```
system.time(replicate(10, alg$iceS$translateToQP(E = IM30x30D2@E,
                                                 xn = IM30x30D2[[]][1,], W = IM30x30D2@W,
```

```
                                                  mu = 0.01, nu = 0.01, n = 1)))
```

```
##    user  system elapsed
##    0.01    0.00    0.02
```

And the minimization

```
alg$iceS$minimizeW <- function(X, E, mu, nu,
                               previousW = hy$W(data = hy$rW(N = 1, M = ncol(E)),
                                                M = ncol(E), N = nrow(X), byrow = TRUE),
                               height = attr(previousW, 'heigh')){

  stopifnot(length(height) > 0)

  N <- nrow(X)
  M <- ncol(E)
  W <- hy$W(previousW, N = N, M = M, origin = 'ICE_W_MINIMIZER_H',
            width = attr(previousW, 'width'), height = height)

  for (n in 1:N) {
    QP <- alg$iceS$translateToQP(E = E, xn = X[n,], W = W, mu = mu, nu = nu, n = n,
                                 height = height)
    W[n,] <- min$ActiveSet(QP = QP, x0 = W[n,])$solve()
  }

  return(W)
}
alg$iceS$minimizeW(X = IM2x2D2[[]], E = IM2x2D2@E, mu = 0.01, nu = 0.99,
                   previousW = IM2x2D2@W)
```

```
## An object of class "W"
##       [,1]   [,2]  [,3]
## [1,] 0.856 0.1442 0.000
## [2,] 0.368 0.2120 0.420
## [3,] 0.442 0.4163 0.141
## [4,] 0.737 0.0504 0.212
## Slot "width":
## [1] 2
##
## Slot "height":
## [1] 2
##
## Slot "origin":
## [1] "ICE_W_MINIMIZER_H"
```

```
rowSums(alg$iceS$minimizeW(X = IM30x30D2[[]], E = IM30x30D2@E, mu = 0.01,
                           nu = 0.99, previousW = IM30x30D2@W))
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [71] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [141] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [176] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [211] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [246] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [281] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [316] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [351] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [386] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
## [421] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [456] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [491] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [526] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [561] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [596] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [631] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [666] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [701] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [736] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [771] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [806] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [841] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [876] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
system.time(replicate(1, alg$iceS$minimizeW(X = IM30x30D2[[]], E = IM30x30D2@E,
                                            mu = 0.01, nu = 0.99, previousW = IM30x30D2@W)))
```

```
##    user  system elapsed
##    1.30    0.00    1.32
```

11.4 Compute E

```
alg$iceS$minimizeE <- function(W, X, mu, nu){
  N <- nrow(X)
  M <- ncol(W)
  B <- ncol(X)
  lambda <- (N*mu*nu)/((M-1)*(1-mu))

  t(X) %*% W %*% solve(t(W) %*% W + lambda * (diag(M) - matrix(1, M, M)/M))
}
alg$iceS$minimizeE(W = IM2x2D2@W, X = IM2x2D2[[]], mu = 0.01, nu = 0.99)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.223 0.803 0.312
## [2,] 0.155 0.440 0.882
```

```
system.time(replicate(1, alg$iceS$minimizeE(X = IM30x30D2[[]],
                                            mu = 0.005, nu = 0.99,
                                            W = hy$rW(M = 3, N = 30*30))))
```

```
##    user  system elapsed
##    0.05    0.00    0.04
```

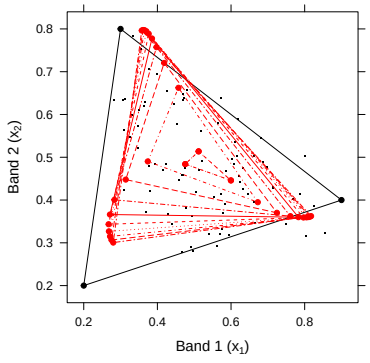
11.5 ALS

And finally the ALS minimization. The whole model is tested ad nauseam to make sure that at each step the objective function is reduced. This makes sure that the minimization procedure is correct (kind of verifies the correctness of E and W minimizations).

```
alg$iceS$ICE <- function(hyImg = NULL, X = hyImg[[]], mu = 0.01, nu,
                         tol = PARAMS$ICETOL, stepTol = 10000,
                         height = if (!is.null(hyImg)) {hy$height(hyImg)}
                                  else {stop('Need valid height!')},
                         width = if (!is.null(hyImg)) {hy$width(hyImg)} else {numeric(0)},
                         M = ncol(X) + 1, Einit = NULL, Winit = NULL){

  if (!is.null(Einit) && !is.null(Winit)) {stop('E and W can not both be specified!')}
```

```
  L <- vector('list'); E <- vector('list'); W <- vector('list')
  k <- 1

  if (!is.null(Einit)) {
    E[[k]] <- Einit
    W[[k]] <- alg$iceS$minimizeW(X = X, E = E[[k]], mu = mu, nu = nu, height = height)
  } else {
    if (!is.null(Winit)) {
      stopifnot(length(attr(Winit, 'height')) > 0 && attr(Winit, 'height') == height)
      W[[k]] <- Winit
    } else {
      W[[k]] <- hy$rW(M = M, N = nrow(X), height = height, width = width)
    }
    E[[k]] <- alg$iceS$minimizeE(W = W[[k]], X = X, mu = mu, nu = nu)
  }

  L[[k]] <- alg$iceS$Lreg(X = X, E = E[[k]], W = W[[k]], mu = mu, nu = nu)

  repeat {
    k <- k + 1
    E[[k]] <- alg$iceS$minimizeE(W = W[[k-1]], X = X, mu = mu, nu = nu)
    # It is not necessary to give the last W as initial values, but
    # seems appropriate since the solution is probably close.
    W[[k]] <- alg$iceS$minimizeW(X = X, E = E[[k]], previousW = W[[k-1]], mu = mu, nu = nu)
    L[[k]] <- alg$iceS$Lreg(X = X, E = E[[k]], W = W[[k]], mu = mu, nu = nu)
    if (PARAMS$LOG) alg$ice$logStep(step = k, L = L, E = E)
    stopifnot(L[[k]]/L[[k-1]] <= 1) # coordinate descent function value check
    if (L[[k]]/L[[k-1]] > tol || k >= stepTol) {break}
  }

  cat(paste("ICE convergence after",k,"steps\n"))

  alg$predictionICE(E = hy$E(E[[k]], wavelengths = hyperSpec::wl(hyImg)),
                    W = W[[k]], steps = k,
                    iterates = list(E = E, W = W, L = L))
}
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E,
               Ehat = Filter(Negate(is.null),
                             alg$iceS$ICE(hyImg = IM9x9D2, mu = 0.01, nu = 0.9,
                                          height = 9, width = 9, tol = 0.99)@iterates[['E']]),
               EhatCol = rep('red', 1000), legendLog = FALSE)
```

```
## ICE convergence after 11 steps
```

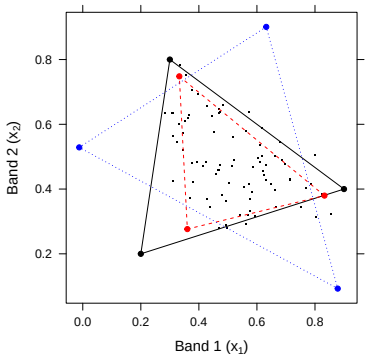


11.6 Testing

11.6.1 Small 9x9 image

```
hy$plotSimplex(X = IM9x9D2[[]],
               E = IM9x9D2@E,
               Ehat = list(
                 'ICE' = alg$ice$ICE(hyImg = IM9x9D2, mu = 0.01)@E,
                 'ICE-S' = alg$iceS$ICE(hyImg = IM9x9D2, mu = 0.1, nu = 0.1, height = 9)@E
               ))
```

```
## ICE convergence after 86 steps
## ICE convergence after 37 steps
```




```
## [1] "ICE color: red"
## [1] "ICE-S color: blue"
```

11.6.2 Kiddo

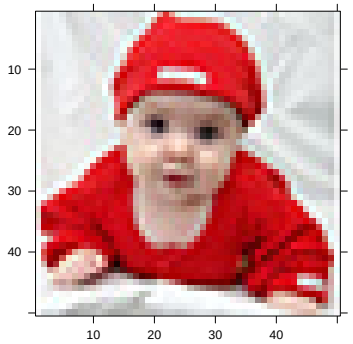
```
IMKiddo <- jpeg::readJPEG("../data/test/kiddo_50x50.jpg")
dim(IMKiddo)
```

```
## [1] 50 50  3
```

```
dim(IMKiddo) <- c(50*50, 3)
```

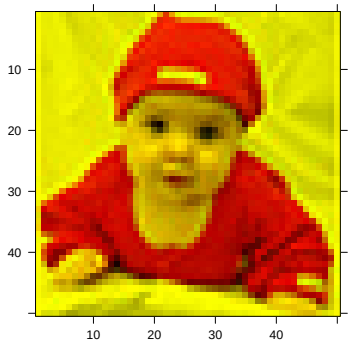
```
IMKiddo <- hy$hyperSpecExt(width = 50, height = 50, depth = 3, wavelengths = c(450, 500, 650), spc =
hy$plotfc(IMKiddo)
```

```
## [1] "Wavelengths for falsecolor image: 450, 500, 650"
```

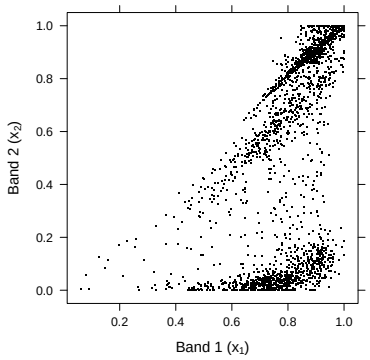


```
IMKiddoD2 <- IMKiddo[,,c(450, 500)]
hy$plotfc(IMKiddoD2)
```

```
## [1] "Wavelengths for falsecolor image: 450, 500"
```




```
hy$plotSimplex(X = IMKiddoD2[[]])
```




```
## [1] " color: red"
```

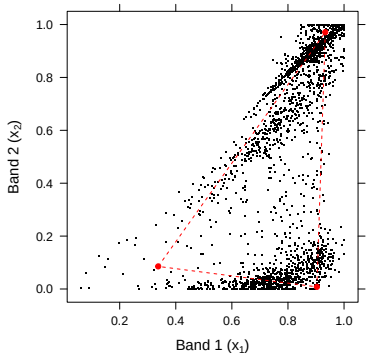
11.6.2.1 Running algorithm

```
IMKiddoD2@pred[['ICE']] <- alg$ice$ICE(hyImg = IMKiddoD2, mu = 0.01)
IMKiddoD2@pred[['ICE-0.5-S-0.02']] <- alg$iceS$ICE(hyImg = IMKiddoD2, mu = 0.5, nu = 0.02)
saveRDS(IMKiddoD2@pred, file = './export/IMKiddoD2@pred.rds')
```

```
IMKiddoD2@pred <- readRDS(file = './export/IMKiddoD2@pred.rds')
```

11.6.2.2 Endmambers and abundance maps

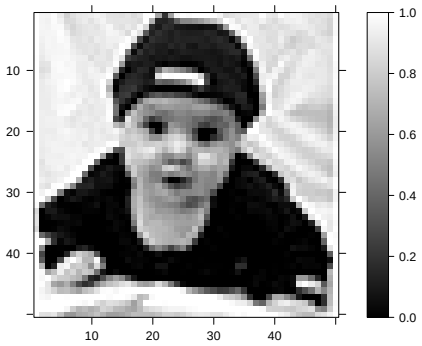
```
hy$plotSimplex(X = IMKiddoD2[[]], Ehat = IMKiddoD2@pred[['ICE']]$E)
```



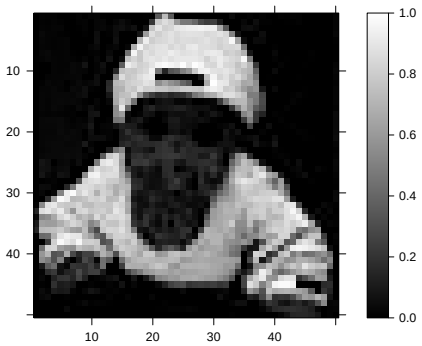

```
## [1] " color: red"
```

```
hy$plotMapW(W = IMKiddoD2@pred[['ICE']]$W)
```

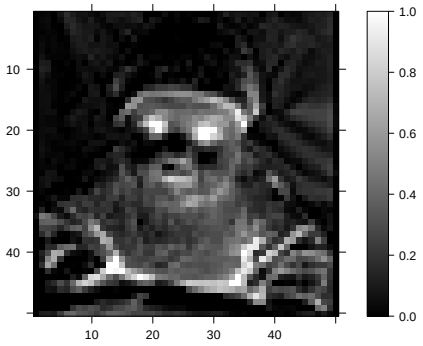
```
## Problematic W-values:
##  Count: 125
##  Min: -1.11022302462516e-16
##  Max: 1
## [1] "Plotting: E1"
```



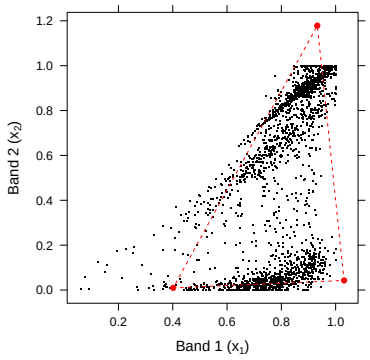

```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```



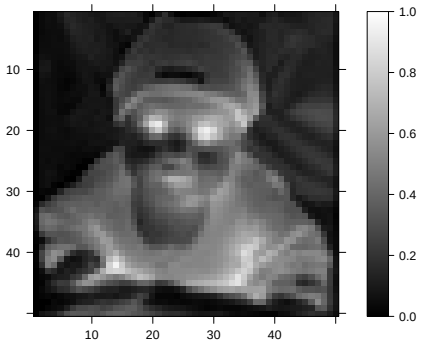

```
hy$plotSimplex(X = IMKiddoD2[[]], Ehat = IMKiddoD2@pred[['ICE-0.5-S-0.02']]$E)
```



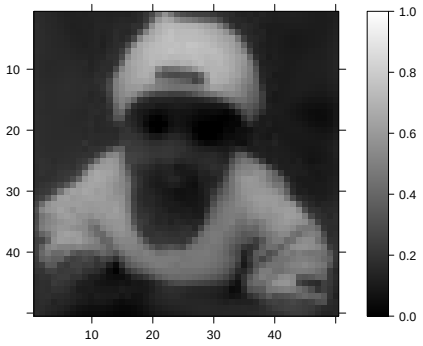

```
## [1] " color: red"
```

```
hy$plotMapW(W = IMKiddoD2@pred[['ICE-0.5-S-0.02']]$W)
```

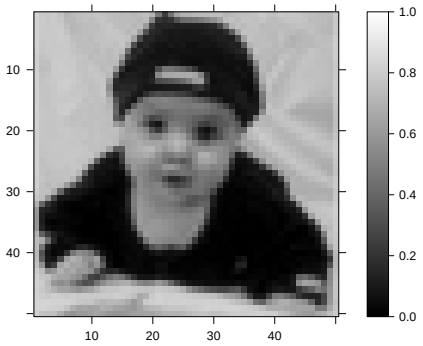
```
## Problematic W-values:
##  Count: 3
##  Min: -1.73472347597681e-18
##  Max: -1.35525271560688e-20
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```



12 Bayesian model

Common to the volume constrained spectral unmixing methods we have discussed so far is that they specify a regularized cost function and solve for the endmembers by numerical optimization. A different approach for hyperspectral unmixing is to build a probabilistic model and treat endmember extraction as a Bayesian inference problem.

This approach also gives credible intervals for the endmembers abundances, which allows us to asses the confidence of the results.

We model the joint probability distribution of the endmembers E, and the fractional abundances W, as well as the noise, conditioned on the observed data X, and a set of model hyper-parameters.

$$p(\boldsymbol{E}, \boldsymbol{W}|\boldsymbol{X}) = \frac{f(\boldsymbol{X}|\boldsymbol{E}, \boldsymbol{W}, \sigma^2) f(\boldsymbol{E}|\gamma) f(\boldsymbol{W}) f(\sigma^2|\alpha, \beta)}{p(\boldsymbol{X})}$$

12.1 The model

12.1.1 Likelihood

```
alg$bVol$likelihood <- function(X, gE, gW, gSigma2){
  Xhat <- gW %*% t(gE)
  exp(sum(log(dnorm(X, mean = Xhat, sd = sqrt(gSigma2)))))
}
alg$bVol$likelihood(X = IM2x2D1[[]], gE = IM2x2D1@E, gW = IM2x2D1@W, gSigma = 1)
```

```
## [1] 0.0253
```

12.1.2 Noise variance prior for σ

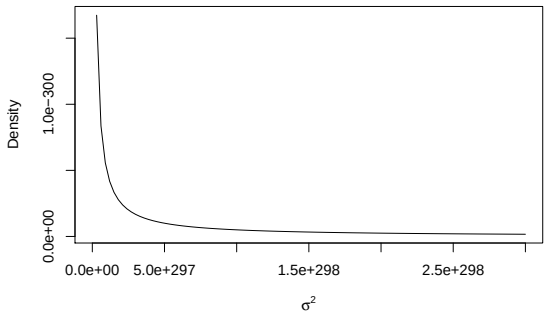
We use the conjugate prior to the normal likelyhood: the inverse-Gamma distribution.

```
alg$bVol$pSigma <- function(sigma2, gAlpha, gBeta){
  sampler$dinvgamma(x = sigma2, alpha = gAlpha, beta = gBeta)
}
```

12.1.2.1 $\alpha = 0.001$ and $\beta = 0.001$ for σ^2 prior density plot

This is approximately what happens when α and β converge to 0: the inverse beta becomes Jeffreys' prior distribution.

```
curve(alg$bVol$pSigma(x, gAlpha = 0.001, gBeta = 0.001), from = 0, to = 3e298,
      xlab = expression(sigma^2), ylab = 'Density')
```




```
dev.copy2pdf( file = './../fig/prior-sigma2-invgamma-alpha2_01-beta1_01.pdf' )
```

```
## pdf
##   2
```

12.1.3 Abundances prior

```
alg$bVol$pW <- function(W){
  # all w_nm are >= 0
  Ia <- prod(W >= 0)
  # all w_n components sum to 1
  Ib <- prod(rowSums(W) == 1)
  return(Ia*Ib)
}
alg$bVol$pW(hy$rW(N = 10, M = 2))
```

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

12.1.4 Endmember prior

```
alg$bVol$pE <- function(E, gGamma){
  # all e_ij are >= 0
  Ia <- prod(E >= 0)
  # regularization
  reg <- exp( -gGamma * vol$berman(E = E) )
  return(reg * Ia)
}
alg$bVol$pE(E = Ed1, gGamma = 0.01)
```

```
## [1] 0.999
```

12.1.5 Posterior

```
alg$bVol$posterior <- function(E, W, sigma2 = 1, gX, gAlpha, gBeta, gGamma){
  l <- alg$bVol$likelihood(X = gX, gE = E, gW = W, gSigma = sigma2)
  pW <- alg$bVol$pW(W = W)
  pE <- alg$bVol$pE(E = E, gGamma = gGamma)
  pSigma2 <- alg$bVol$pSigma(sigma2 = sigma2, gAlpha = gAlpha, gBeta = gBeta)
  return(l*pW*pE*pSigma2)
}
alg$bVol$posterior(E = IM2x2D1@E, W = IM2x2D1@W, sigma2 = 1, gX = IM2x2D1[[]],
                   gAlpha = 0.001, gBeta = 0.001, gGamma = 0.01)
```

```
## [1] 0.0000251
```

12.2 Sampling

Gibbs sampling is applicable when the joint density of the parameters is not known, but the parameters can be partitioned into groups, such that their posterior conditional densities are known. We iteratively sweep through the groups of parameters and generate a random sample for each, conditioned on the current value of the others. This procedure forms a homogenous Markov chain and its stationary distribution is exactly the joint posterior. [2]

12.2.1 Noise variance σ^2 sampler

```
alg$bVol$rSigma2 <- function(gX, gE, gW, gAlpha, gBeta){
  N <- nrow(gX)
  B <- ncol(gX)

  alpha <- gAlpha + 1/2*N*B
  beta <- gBeta + 1/2*sum((gX - gW %*% t(gE))^2) # faster than norm(type='F')

  sampler$rinvgamma(alpha = alpha, beta = beta)
}
alg$bVol$rSigma2(gX = IM30x30D2[[]], gE = IM30x30D2@E, gW = IM30x30D2@W,
                 gAlpha = 2.01, gBeta = 1.01)
```

```
## [1] 0.00211
```

```
alg$bVol$rSigma2.Test <- function(k = 1000, burnin = 100, gX = IM30x30D2[[]],
                                  gE = IM30x30D2@E, gW = IM30x30D2@W,
                                  gAlpha = 2.01, gBeta = 1.01){

  sSamples <- matrix(nrow = k+burnin, ncol = 1)

  ptm <- proc.time()

  for (i in 1:(k+burnin)) {
    sSamples[i,] <- alg$bVol$rSigma2(gW = gW, gE = gE, gX = gX,
                                     gAlpha = gAlpha, gBeta = gBeta)
  }

  list(colMean = colMeans(sSamples[-(1:burnin),,drop=FALSE]),
       colVar = apply(sSamples[-(1:burnin),,drop=FALSE], MARGIN = 2, var),
       time = proc.time() - ptm)
```

```
}
alg$bVol$rSigma2.Test()
```

```
## $colMean
## [1] 0.00216
##
## $colVar
## [1] 5.27e-09
##
## $time
##    user  system elapsed
##    0.06    0.00    0.06
```

12.2.2 Fractional Abundances $\boldsymbol{W}$ sampler

Note that the distribution of interest is the following:

$$f_{\boldsymbol{w}_n}(\boldsymbol{w}_n|\boldsymbol{E},\sigma^2,\boldsymbol{x}_n) \propto \mathcal{N}(\boldsymbol{w}_n|\boldsymbol{\mu}_n,\boldsymbol{\Sigma}) \left(\mathbb{I}\left[\|\boldsymbol{w}_n\|_1 = 1\right] \prod_{m=1}^{M} \mathbb{I}\left[w_{nm} \geq 0\right] \right)$$

with paramters: $\boldsymbol{\mu}_n = (\boldsymbol{E}^T\boldsymbol{E})^{-1}\boldsymbol{E}^T\boldsymbol{x}_n$ and $\boldsymbol{\Sigma} = \sigma^2(\boldsymbol{E}^T\boldsymbol{E})^{-1}$. These parameters we compute next.

12.2.2.1 Computing μ_n and Σ

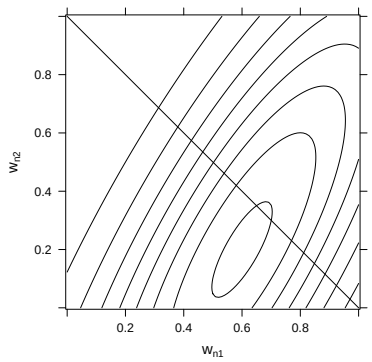
```
alg$bVol$rW.Parameters <- function(E, sigma2, X){
  stopifnot(length(sigma2) == 1)
  EtE <- t(E) %*% E
  EtEInv <- sampler$invSigma(EtE)
  list(Sigma = sigma2 * EtEInv,
       InvSigma = EtE / sigma2,
       Mu = X %*% E %*% t(EtEInv))
}
alg$bVol$rW.Parameters(E = IM2x2D1@E, sigma2 = IM2x2D1@sigma2, X = IM2x2D1[[]])
```

```
## $Sigma
##          [,1]    [,2]
## [1,] 0.000148 0.00074
## [2,] 0.000740 0.00370
##
## $InvSigma
##      [,1] [,2]
## [1,]   10   50
## [2,]   50  250
##
## $Mu
##        [,1]  [,2]
## [1,] 0.0774 0.387
## [2,] 0.0992 0.496
## [3,] 0.1651 0.825
## [4,] 0.1845 0.923
```

12.2.2.2 Visualisation

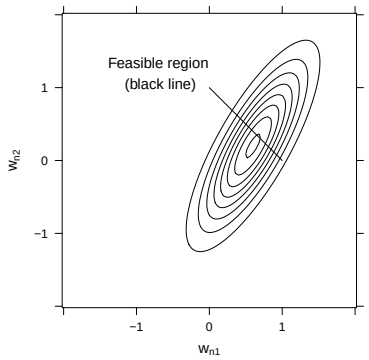
```
alg$bVol$contourplotW2d <- function(mu, Sigma, xlim = c(0,1), ylim = c(0,1),
                                    pointsToCalc = 10000, printAs = character(0)){
  stopifnot(length(Sigma) == 4)
```

```
  stopifnot(length(mu) == 2)

  xGrid <- seq(xlim[1], xlim[2], abs(xlim[1] - xlim[2])/sqrt(pointsToCalc))
  yGrid <- seq(ylim[1], ylim[2], abs(ylim[1] - ylim[2])/sqrt(pointsToCalc))

  dataFrame <- expand.grid(x = xGrid, y = yGrid)
  dataFrame$z <- apply(X = dataFrame, MARGIN = 1,
                       FUN = function(v){sampler$dmnorm(x = v,
                                                        mean = mu,
                                                        Sigma = Sigma)})

  plt <- contourplot(z ~ x * y, data = dataFrame, aspect = 1, labels = FALSE,
                     xlab = expression(w[n][1]), ylab = expression(w[n][2]),
                     panel = function(...) {
                       panel.text(-0.7, 1.2, labels = 'Feasible region\n (black line)')
                       panel.contourplot(...)
                       panel.curve(-x + 1, col = "black", from = 0, to = 1)})

  print(plt)

  if (length(printAs) != 0) {
    dev.copy2pdf( file = paste('./../fig/contourplot_W_density',
                               gsub('[.]', '_', printAs),
                               '', '.pdf', sep = ""),
                  compress = FALSE )
  }
}
alg$bVol$contourplotW2d(mu = c(0.6, 0.2),
                        Sigma = matrix(c(0.2, 0.25, 0.25, 0.5), ncol = 2))
```



Print this image for in thesis.

```
alg$bVol$contourplotW2d(xlim = c(-2,2), ylim = c(-2,2), mu = c(0.6, 0.2),
                        Sigma = matrix(c(0.2, 0.25, 0.25, 0.5), ncol = 2),
                        printAs = 'mu06-02_sigma02-025-025-05')
```




```
## pdf
##   2
```

Now we have to find ways for how to sample from this density.

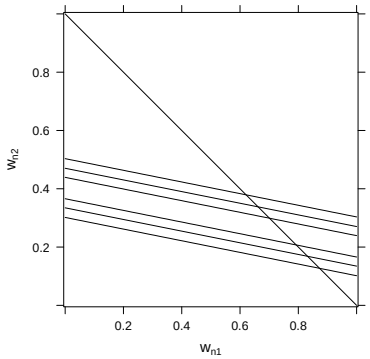
12.2.2.3 Metropolis-Hastings sampler

```
alg$bVol$rW <- function(gE, gX, gSigma2, previousW){

  dnkernel <- function(x, mu, invCov){ # kernel of normal multivariate density
    return( -1/2 * t(x-mu) %*% invCov %*% (x-mu) )
  }

  N <- nrow(gX)
  M <- ncol(gE)

  param <- alg$bVol$rW.Parameters(E = gE, sigma2 = gSigma2, X = gX)

  # for each 1:N rows of Mu, Sigma is the covariance matrix
  Mu <- param$Mu # result is NxM matrix
  InvSigma <- param$InvSigma # result is a MxM matrix

  W <- matrix(nrow = N, ncol = M)

  for (n in 1:N) {
    new <- hy$rdirichlet(alpha = rep(1, M))
    Q <- exp(dnkernel(new, Mu[n,], InvSigma) - dnkernel(previousW[n,], Mu[n,], InvSigma))
    alpha <- min(1, Q, na.rm = FALSE)
    if (runif(n = 1, min = 0, max = 1) <= alpha) {
      W[n,] <- new
    } else {
      W[n,] <- previousW[n,]
    }
  }

  return(W)
}
alg$bVol$rW(gE = IM2x2D1@E, gX = IM2x2D1[[]], gSigma2 = 1,
            previousW = hy$rW(N = 2*2, M = 2))
```

```
##       [,1]   [,2]
## [1,] 0.961 0.0388
## [2,] 0.341 0.6595
## [3,] 0.206 0.7935
## [4,] 0.630 0.3699
```

12.2.2.3.1 Testing the W sampler

```
alg$bVol$contourplotW2d(mu = alg$bVol$rW.Parameters(E = IM2x2D1@E, sigma2 = IM2x2D1@sigma2,
                                                    X = IM2x2D1[[]])$Mu[1,],
                        Sigma = alg$bVol$rW.Parameters(E = IM2x2D1@E, sigma2 = IM2x2D1@sigma2,
                                                       X = IM2x2D1[[]])$Sigma)
```




```
alg$bVol$rW.Test <- function(k = 1000, burnin = 100, gE = IM2x2D1@E,
                             gSigma2 = IM2x2D1@sigma2,
                             gX = IM2x2D1[[1]],
                             initW = hy$rW(N = nrow(gX), M = ncol(gE))){

  wSamples <- matrix(nrow = k+burnin, ncol = ncol(gE))
  wSamples[1,] <- initW

  ptm <- proc.time()

  for (i in 2:(k+burnin)) {
    wSamples[i,] <- alg$bVol$rW(gE = gE, gX = gX, gSigma2 = gSigma2,
                                previousW = wSamples[i-1,,drop=FALSE])
  }

  list(colMean = colMeans(wSamples[-(1:burnin),]),
       colVar = apply(wSamples[-(1:burnin),], MARGIN = 2, var),
       time = proc.time() - ptm)
}
alg$bVol$rW.Test()
```

```
## $colMean
## [1] 0.75 0.25
##
## $colVar
## [1] 0.00662 0.00662
##
## $time
##    user  system elapsed
##    0.34    0.00    0.34
```

```
alg$bVol$rW.Test(gE = IM2x2D2@E, gX = IM2x2D2[[2]], gSigma2 = IM2x2D2@sigma2)
```

```
## $colMean
## [1] 0.369 0.211 0.420
##
## $colVar
## [1] 0.00307 0.00219 0.00376
##
## $time
```

```
##    user  system elapsed
##    0.36    0.00    0.36
```

12.2.2.4 Gibbs sampler

```
alg$bVol$rW <- function(gE, gX, gSigma2, previousW){

  N <- nrow(gX)
  M <- ncol(gE)

  # Transformation equations for y = Ax + b, where thus y is independent
  A <- matrix(data = c(diag(M-1), rep(-1,M-1)), ncol = M-1, byrow = TRUE)
  b <- c(rep(0, M-1), 1)

  SigmaYInv <- ( t(A) %*% t(gE) %*% gE %*% A ) / gSigma2
  SigmaY <- solve(SigmaYInv) # Never singular, probably due to A transformation.
  # MuY has muY in columns! t(gX) - gE %*% b works because gE %*% b is recycled
  # for each column of t(gX)
  MuY <- ( SigmaY / gSigma2 ) %*% t(A) %*% t(gE) %*% ( t(gX) - as.vector(gE %*% b) )

  W <- matrix(nrow = N, ncol = M)

  for (n in 1:N) {
    y <- previousW[n,-M]
    for (k in 1:(M-1)) {
      sigma2Yk <- 1/SigmaYInv[k,k]
      muYk <- ( t(SigmaYInv[k,]) %*% MuY[,n] - SigmaYInv[k,-k] %*% y[-k] ) * sigma2Yk
      uBound <- 1 - sum(y[-k])
      y[k] <- sampler$rtnorm(mean = muYk, sigma2 = sigma2Yk, lbound = 0, ubound = uBound)
    }
    W[n,] <- c(y, 1-sum(y))
  }

  return(W)

}
alg$bVol$rW(gE = IM2x2D1@E, gX = IM2x2D1[[]], gSigma2 = 1,
            previousW = hy$rW(N = 2*2, M = 2))
```

```
##       [,1]   [,2]
## [1,] 0.750 0.2502
## [2,] 0.978 0.0224
## [3,] 0.876 0.1242
## [4,] 0.703 0.2969
```

12.2.2.4.1 Test the new Gibbs sampler

```
alg$bVol$rW.Test()
```

```
## $colMean
## [1] 0.75 0.25
##
## $colVar
## [1] 0.00615 0.00615
##
## $time
##    user  system elapsed
##    0.18    0.00    0.19
```

```
alg$bVol$rW.Test(gE = IM2x2D2@E, gX = IM2x2D2[[2]], gSigma2 = IM2x2D2@sigma2)
```

```
## $colMean
## [1] 0.368 0.213 0.419
##
## $colVar
## [1] 0.00336 0.00220 0.00353
##
## $time
##    user  system elapsed
##    0.26    0.00    0.27
```

12.2.3 Endmembers E sampler

Conditional prior has the form of a truncated Gaussian [1]. This one is from Arngren.

```
alg$bVol$rE <- function(gW, gX, gSigma2, gGamma, previousE){
  D <- dim(gX)[2]
  M <- dim(gW)[2]

  E <- previousE

  library(truncnorm)
  for (d in 1:D) {
    for (m in 1:M) {
      c <- 1/(M - 1) * sum(E[d,-m])
      s2 <- gGamma^-1 * ((M - 1)/M)^-2
      sigma2 <- ( s2^-1 + (t(gW[,m]) %*% gW[,m])*gSigma2^-1 )^-1
      mu <- sigma2 * ( c*s2^-1 + ( t(gX[,d]) %*% gW[,m] -
                                     E[d,-m] %*% t(gW[,-m]) %*% gW[,m] ) * gSigma2^-1 )
      E[d,m] <- rtruncnorm(1, mean = mu, sd = sqrt(sigma2), a = 0)
    }
  }

  return(E)
}
alg$bVol$rE(gW = IM30x30D2@W, gX = IM30x30D2[[]], gSigma2 = 1, gGamma = 0.01,
            previousE = IM30x30D2@E)
```

```
##        [,1]  [,2]  [,3]
## [1,] 0.1550 0.972 0.283
## [2,] 0.0924 0.492 0.796
```

12.2.3.1 Testing the E sampler

```
alg$bVol$rE.Test <- function(k = 1000, burnin = 100, gW = IM30x30D2@W,
                             gSigma2 = IM30x30D2@sigma2, gGamma = 10,
                             gX = IM30x30D2[[]],
                             initE = IM30x30D2@E){

  eSamples <- array(dim = c(ncol(gX), ncol(gW), k + burnin))
  eSamples[,,1] <- initE

  ptm <- proc.time()

  for (i in 2:(k+burnin)) {
    eSamples[,,i] <- alg$bVol$rE(gW = gW, gX = gX, gSigma2 = gSigma2,
                                 gGamma = gGamma, previousE = eSamples[,,i-1])
```

```
  }

  list(mean = apply(eSamples[,,-(1:burnin)], c(1,2), mean),
       colVar = apply(eSamples[,,-(1:burnin)], c(1,2), var),
       time = proc.time() - ptm)
}
alg$bVol$rE.Test()
```

```
## $mean
##       [,1]  [,2]  [,3]
## [1,] 0.194 0.906 0.301
## [2,] 0.188 0.400 0.807
##
## $colVar
##           [,1]      [,2]       [,3]
## [1,] 0.0000228 0.0000095 0.00000938
## [2,] 0.0000215 0.0000103 0.00000938
##
## $time
##    user  system elapsed
##    1.35    0.01    1.37
```

This one is derived by myself. See appendix in thesis. Is disabled currently since I don't get the same results as with the above one.

```
alg$bVol$rE <- function(gW, gX, gSigma2, gGamma, previousE){
  B <- ncol(gX)
  M <- ncol(gW)

  SigmaInv <- t(gW) %*% gW / gSigma2 + 2*gGamma*diag(M)/(M-1) -
              2*gGamma*matrix(1, nrow = M, ncol = M)/(M*(M-1))
  Sigma <- solve(SigmaInv)
  Mu <- Sigma %*% t(gW) %*% gX / gSigma2 # columns are mu_b

  E <- previousE

  for (b in 1:B) {
    for (m in 1:M) {
      sigma2bm <- 1/SigmaInv[m,m]
      mubm <- (t(SigmaInv[m,]) %*% Mu[,b] - t(SigmaInv[m,-m]) %*% E[b,-m]) * sigma2bm
      E[b,m] <- sampler$rtnorm(mean = mubm, sigma2 = sigma2bm, lbound = 0)
    }
  }

  return(E)
}
alg$bVol$rE(gW = IM30x30D2@W, gX = IM30x30D2[[]], gSigma2 = 1, gGamma = 0.01,
            previousE = IM30x30D2@E)
```

12.2.3.2 Testing the E sampler

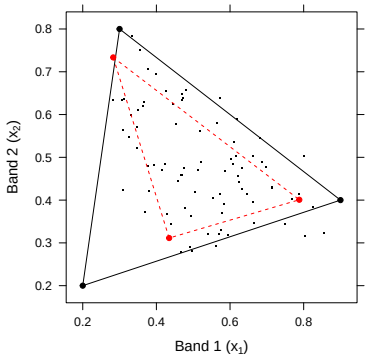
```
alg$bVol$rE.Test()
```

```
## $mean
##       [,1]  [,2]  [,3]
## [1,] 0.193 0.906 0.301
## [2,] 0.188 0.400 0.807
##
## $colVar
```

```
##           [,1]       [,2]       [,3]
## [1,] 0.0000226 0.00001006 0.00000932
## [2,] 0.0000244 0.00000931 0.00000956
##
## $time
##    user  system elapsed
##    1.36    0.03    1.39
```

12.2.4 Joint $(\boldsymbol{E}, \boldsymbol{W}, \sigma^2)$ sampler

```
alg$bVol$sample <- function(nrSamples = PARAMS$SAMPLESIZE, hyImg = NA, gX = hyImg[[]],
                            gAlpha = 0, gBeta = 0, gGamma = 0,
                            burnIn = PARAMS$BURNINSIZE, M = ncol(gX) + 1,
                            initW = hy$rW(N = nrow(gX), M = M),
                            initE = hy$rE(M = M, D = ncol(gX), maxval = max(gX),
                                          wavelengths = hyperSpec::wl(hyImg)),
                            height = hy$height(hyImg), width = hy$width(hyImg)){

  scan <- list('sigma2' = NA, 'E' = initE, 'W' = initW)

  samples <- alg$predictionSamples(S1 = matrix(nrow = nrSamples + burnIn,
                                               ncol = length(unlist(scan)),
                                               dimnames = list(NULL, names(unlist(scan)))),
                                   skelet = scan)

  for (i in 1:nrow(samples@S1)) {
    scan$sigma2 <- alg$bVol$rSigma2(gX = gX, gE = scan$E, gW = scan$W,
                                    gAlpha = gAlpha, gBeta = gBeta)
    scan$W <- alg$bVol$rW(gX = gX, gSigma2 = scan$sigma2,
                          gE = scan$E, previousW = scan$W)
    scan$E <- alg$bVol$rE(gX = gX, gSigma2 = scan$sigma2, gGamma = gGamma,
                          gW = scan$W, previousE = scan$E)
    samples@S1[i,] <- unlist(scan)

    if (i %% 200 == 0 && PARAMS$LOG) {print(i)}
  }

  if (burnIn > 0) { samples@S1 <- samples@S1[-c(1:burnIn), ] }

  return(alg$predictionBayes(samples = samples, height = height, width = width))
}
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E,
               Ehat = alg$bVol$sample(nrSamples = 50, hyImg = IM9x9D2, burnIn = 20)$E)
```



```
## [1] " color: red"
```

12.3 References

[1] Arngren, M. e.a., Unmixing of Hyperspectral Images using Bayesian Nonnegative Matrix Factorization with Volume Prior, 2011

[2] Schmidt, M.N., Linearly constrained Bayesian matrix factorization for blind source separation, 2009

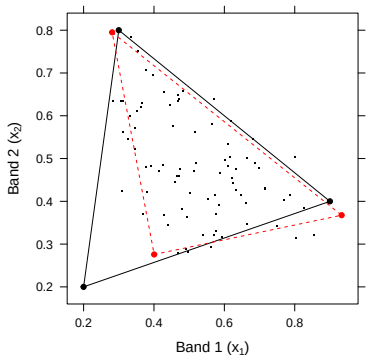
12.4 Testing

12.4.1 Example

```
IM9x9D2@pred[['Bayes']] <- alg$bVol$sample(nrSamples = 500, hyImg = IM9x9D2, burnIn = 200)
IM9x9D2@pred[['Bayes']]$E
```

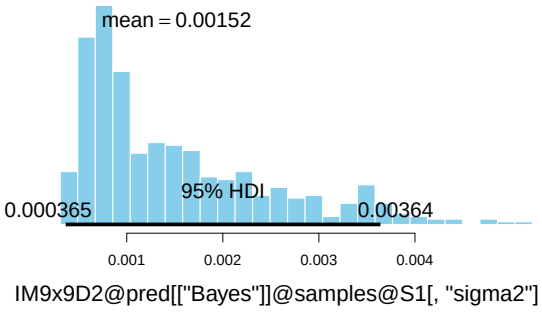
```
##      [,1]  [,2]  [,3]
## 1 0.401 0.933 0.281
## 2 0.276 0.368 0.795
```

```
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E, Ehat = IM9x9D2@pred[['Bayes']]$E)
```




```
## [1] " color: red"
```

```
BEST::plotPost(IM9x9D2@pred[['Bayes']]@samples@S1[,'sigma2'])
```



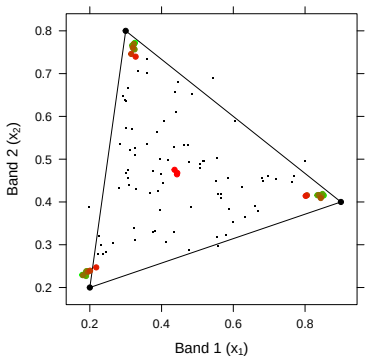
12.5 Bayesian intrinsic regularization

```
set.seed(PARAMS$SEED)
IM9x9D2.FM <- hy$rImage(height = 9, width = 9, E = Ed2, SNRdb = 15)
```

```
IM9x9D2.FM@pred[['Bayes-0']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 0)
IM9x9D2.FM@pred[['Bayes-0.0001']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 0.0001)
IM9x9D2.FM@pred[['Bayes-0.001']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 0.001)
IM9x9D2.FM@pred[['Bayes-0.01']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 0.01)
IM9x9D2.FM@pred[['Bayes-0.1']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 0.1)
IM9x9D2.FM@pred[['Bayes-1']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 1)
IM9x9D2.FM@pred[['Bayes-10']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 10)
IM9x9D2.FM@pred[['Bayes-100']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 100)
IM9x9D2.FM@pred[['Bayes-140']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 140)
IM9x9D2.FM@pred[['Bayes-1000']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 1000)

saveRDS(IM9x9D2.FM@pred, file = './export/IM9x9D2.FM@pred.rds')
```

```
IM9x9D2.FM@pred <- readRDS(file = './export/IM9x9D2.FM@pred.rds')
```

```
hy$plotSimplex(X = IM9x9D2.FM[[]], E = IM9x9D2.FM@E, Ehat = list(
  'Gamma = 0' = IM9x9D2.FM@pred[['Bayes-0']]$E,
  'Gamma = 0.0001' = IM9x9D2.FM@pred[['Bayes-0.0001']]$E,
  'Gamma = 0.001' = IM9x9D2.FM@pred[['Bayes-0.001']]$E,
  'Gamma = 0.01' = IM9x9D2.FM@pred[['Bayes-0.01']]$E,
  'Gamma = 0.1' = IM9x9D2.FM@pred[['Bayes-0.1']]$E,
  'Gamma = 1' = IM9x9D2.FM@pred[['Bayes-1']]$E,
  'Gamma = 10' = IM9x9D2.FM@pred[['Bayes-10']]$E,
  'Gamma = 100' = IM9x9D2.FM@pred[['Bayes-100']]$E,
  'Gamma = 140' = IM9x9D2.FM@pred[['Bayes-140']]$E,
  'Gamma = 1000' = IM9x9D2.FM@pred[['Bayes-1000']]$E
),
EhatCol = colorRamps::green2red(length(IM9x9D2.FM@pred)),
printAs = 'bayes-effect-vol-constraint',
plotPolygons = FALSE)
```




```
## [1] "Gamma = 0 color: #00FF00"
## [1] "Gamma = 0.0001 color: #1CE300"
## [1] "Gamma = 0.001 color: #39C600"
## [1] "Gamma = 0.01 color: #55AA00"
## [1] "Gamma = 0.1 color: #718E00"
## [1] "Gamma = 1 color: #8E7100"
## [1] "Gamma = 10 color: #AA5500"
## [1] "Gamma = 100 color: #C63900"
## [1] "Gamma = 140 color: #E31C00"
## [1] "Gamma = 1000 color: #FF0000"
```

```
## pdf
##   2
```

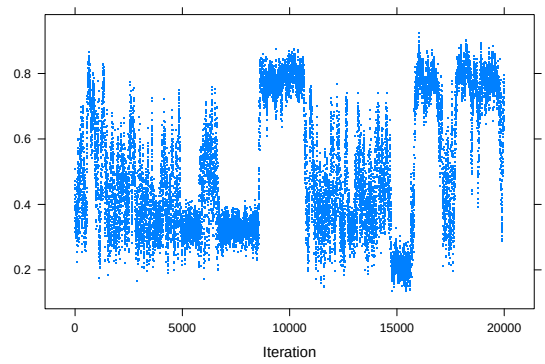
12.5.1 Indeterminacy for medium simplex

```
IM9x9D2.FM@pred[['Bayes-150']] <- alg$bVol$sample(hyImg = IM9x9D2.FM, gGamma = 150,
                                                  nrSamples = 20000, burnIn = 5000)

saveRDS(IM9x9D2.FM@pred[['Bayes-150']], file = './export/IM9x9D2.FM@pred[[Bayes-150]].rds')
```

```
IM9x9D2.FM@pred[['Bayes-150']] <- readRDS(file = './export/IM9x9D2.FM@pred[[Bayes-150]].rds')
```

Trace of e_{11}:

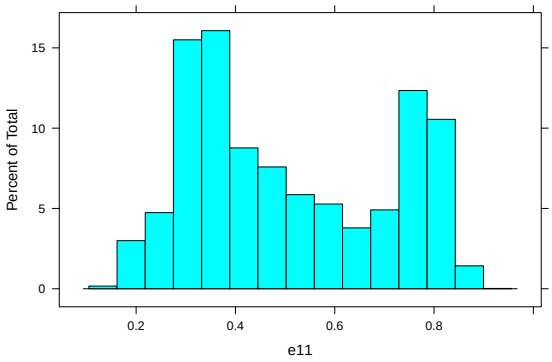
```
lattice::xyplot(
  x = IM9x9D2.FM@pred[['Bayes-150']]@samples@S1[,'E1'] ~
    1:length(IM9x9D2.FM@pred[['Bayes-150']]@samples@S1[,'E1']),
  pch = '.', xlab = 'Iteration', ylab = '')
```




```
dev.copy2pdf( file = './../fig/bayes-trace-of-e1-medium-simplex.pdf' )
```

```
## pdf
##   2
```

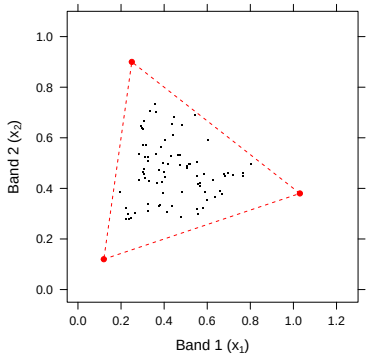
```
lattice::histogram(IM9x9D2.FM@pred[['Bayes-150']]@samples@S1[,'E1'], xlab = 'e11')
```



12.6 The volume regulating effect of prior on W

These are just two simplexes. The explanation is in thesis.

```
m <- as.matrix(rowSums(IM9x9D2.FM@E)/3)
hy$plotSimplex(X = IM9x9D2.FM[[]], Ehat = 1.3*(IM9x9D2.FM@E - rep(m,3)) + rep(m,3),
               xlim = c(-0.05,1.3), ylim = c(-0.05,1.10))
```

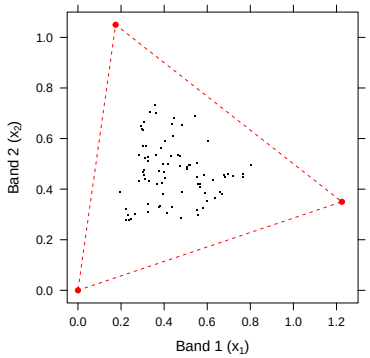



```
## [1] " color: red"
```

```
dev.copy2pdf( file = './../fig/bayes-prior-W-vol-reg-normal-simplex.pdf' )
```

```
## pdf
##   2
```

```
hy$plotSimplex(X = IM9x9D2.FM[[]], Ehat = 1.75*(IM9x9D2.FM@E - rep(m,3)) + rep(m,3),
               xlim = c(-0.05,1.3), ylim = c(-0.05,1.10))
```




```
## [1] " color: red"
```

```
dev.copy2pdf( file = './../fig/bayes-prior-W-vol-reg-large-simplex.pdf' )
```

```
## pdf
##   2
```

```
rm(m)
```

13 Bayesian model with JAGS

13.1 The model

```
alg$bVolJAGS$modelstring <- "
model{
  # Likelihood
  for (n in 1:N){
    for (b in 1:B){
      X[n,b] ~ dnorm(inprod(W[n,], E[b,]), 1/sigma2)
    }
  }
```

```
  # Uniform prior for E
  for (m in 1:M) {
    for (b in 1:B) {
      E[b,m] ~ dunif(0, sensorUpperBound)
    }
  }

  # Dirichlet prior for W
  for (n in 1:N){
    for(m in 1:M) {
      wi_raw[n,m] ~ dgamma(dirichletAlpha[m], 1)
    }
    for(m in 1:M) {
      W[n,m] <- wi_raw[n,m] / sum(wi_raw[n,])
    }
  }

  # Sigma2 prior
  #tmp ~ dgamma(0.000000001,0.000000001)
  #sigma2 <- 1/tmp
  sigma2 ~ dunif(0, 100)
}
"
```

13.2 Sampling

```
alg$bVolJAGS$sample <- function(nrSamples = PARAMS$SAMPLESIZE, hyImg = NA, gX = hyImg[[]],
                            burnIn = PARAMS$BURNINSIZE, M = ncol(gX) + 1,
                            dirichletAlpha = rep(1, M),
                            sensorUpperBound,
                            height = hy$height(hyImg), width = hy$width(hyImg)){
  library('rjags')

  # Initialize the model
  model <- jags.model(textConnection(alg$bVolJAGS$modelstring),
                      data=list('X' = gX, 'N' = nrow(gX), 'B' = ncol(gX),
                                'M' = M, 'dirichletAlpha' = dirichletAlpha,
                                'sensorUpperBound'=sensorUpperBound),
                      n.chains = 1, n.adapt = burnIn,
                          #inits = list(.RNG.name = 'base::Mersenne-Twister', .RNG.seed = 2017)
                          )

  # Get samples from the posterior of W and E
  output <- coda.samples(model = model, variable.names = c("W", "E", "sigma2"),
                         n.iter = nrSamples, thin = 1)

  # Make skeleton for rjags mcmc samples (this is risky!! - no guarantee that it always works)
  colNames <- colnames(output[[1]])
  variables <- unique(substr(colNames, start=0,
                             stop = ifelse(regexpr("\\[", colNames) == -1,
                                           nchar(colNames),
                                           regexpr("\\[", colNames) - 1)))

  skelet_sigma2 <- NA
  skelet_W <- hy$rW(N = nrow(gX), M = M)
  skelet_E <- hy$rE(M = M, D = ncol(gX))
```

```
  scan = list()
  for (v in variables) # will put variables in order they are in the sample output
    scan[[v]] <- get(paste('skelet_', v, sep = ''))

  stopifnot(length(colnames(output[[1]])) == length(names(unlist(scan)))) # check

  # Finaly save the samples are return
  samples <- alg$predictionSamples(S1 = as.matrix(output[[1]]), skelet = scan)

  return(alg$predictionBayes(samples = samples, height = height, width = width))
}
```

13.3 References

[1] Arngren, M. e.a., Unmixing of Hyperspectral Images using Bayesian Nonnegative Matrix Factorization with Volume Prior, 2011

[2] Schmidt, M.N., Linearly constrained Bayesian matrix factorization for blind source separation, 2009

13.4 Example

```
IM9x9D2@pred[['Bayes-JAGS']] <- alg$bVolJAGS$sample(hyImg = IM9x9D2, M = 3, sensorUpperBound = 255)
```

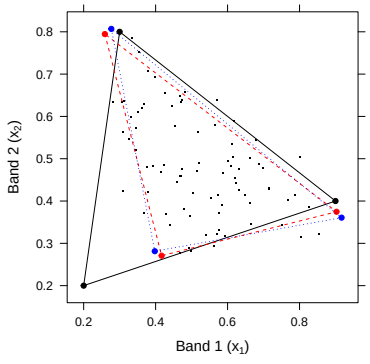
```
## Loading required package: coda
```

```
## Linked to JAGS 4.2.0
```

```
## Loaded modules: basemod,bugs
```

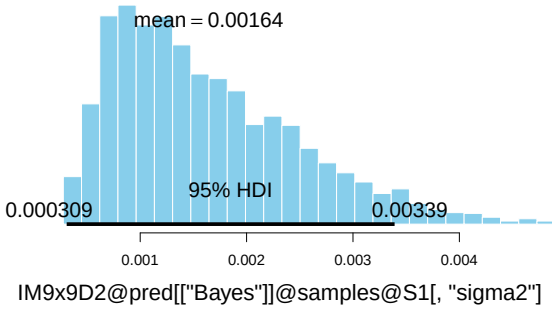
```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 162
##    Unobserved stochastic nodes: 250
##    Total graph size: 1645
##
## Initializing model
```

```
IM9x9D2@pred[['Bayes']] <- alg$bVol$sample(hyImg = IM9x9D2, M = 3)
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E,
               Ehat = list('JAGS' = IM9x9D2@pred[['Bayes-JAGS']]$E,
                           'Bayes' = IM9x9D2@pred[['Bayes']]$E))
```



```
## [1] "JAGS color: red"
## [1] "Bayes color: blue"
```

```
BEST::plotPost(IM9x9D2@pred[['Bayes']]@samples@S1[,'sigma2'])
```



13.5 References

[1] Arngren, M. e.a., Unmixing of Hyperspectral Images using Bayesian Nonnegative Matrix Factorization with Volume Prior, 2011

[2] Schmidt, M.N., Linearly constrained Bayesian matrix factorization for blind source separation, 2009

14 BayesNMF-Vol Extended model

14.1 The τ prior

14.2 Sampling

14.2.1 Tau sampling

```
alg$bVolTau$rTau <- function(gW){
  M <- ncol(gW)
  runif(n = M, min = apply(IM30x30D2@W, 2, max), max = 1)
}
alg$bVolTau$rTau(gW = IM30x30D2@W)
```

```
## [1] 0.671 0.976 0.995
```

14.2.2 W sampling

```
alg$bVolTau$rW <- function(gE, gX, gSigma2, gTau, previousW){

  dnkernel <- function(x, mu, invCov){ # kernel of normal multivariate density
    return( -1/2 * t(x-mu) %*% invCov %*% (x-mu) )
  }

  N <- nrow(gX)
  M <- ncol(gE)

  param <- alg$bVol$rW.Parameters(E = gE, sigma2 = gSigma2, X = gX)
```

```
  # for each 1:N rows of Mu, Sigma is the covariance matrix
  Mu <- param$Mu # result is NxM matrix
  InvSigma <- param$InvSigma # result is a MxM matrix

  W <- matrix(nrow = N, ncol = M)

  for (n in 1:N) {
    repeat{
      new <- hy$rdirichlet(alpha = rep(1, M))
      if (all(new <= gTau)) {break}
    }
    Q <- exp(dnkernel(new, Mu[n,], InvSigma) - dnkernel(previousW[n,], Mu[n,], InvSigma))
    alpha <- min(1, Q, na.rm = FALSE)
    if (runif(n = 1, min = 0, max = 1) <= alpha) {
      W[n,] <- new
    } else {
      W[n,] <- previousW[n,]
    }
  }

  return(W)
}
alg$bVolTau$rW(gE = IM2x2D1@E, gX = IM2x2D1[[]], gSigma2 = 1, gTau = c(0.5, 0.7),
            previousW = hy$rW(N = 2*2, M = 2))
```

```
##       [,1]  [,2]
## [1,] 0.339 0.661
## [2,] 0.358 0.642
## [3,] 0.351 0.649
## [4,] 0.414 0.586
```

14.2.3 E Sampling

```
alg$bVolTau$rE <- function(gW, gX, gSigma2, gGamma, previousE){
  B <- ncol(gX)
  M <- ncol(gW)

  SigmaInv <- t(gW) %*% gW / gSigma2 + 2*gGamma*diag(M)/(M-1) -
              2*gGamma*matrix(1, nrow = M, ncol = M)/(M*(M-1))
  Sigma <- solve(SigmaInv)
  Mu <- Sigma %*% t(gW) %*% gX / gSigma2 # columns are mu_b

  E <- previousE

  for (b in 1:B) {
    for (m in 1:M) {
      sigma2bm <- 1/SigmaInv[m,m]
      mubm <- (t(SigmaInv[m,]) %*% Mu[,b] - t(SigmaInv[m,-m]) %*% E[b,-m]) * sigma2bm
      E[b,m] <- sampler$rtnorm(mean = mubm, sigma2 = sigma2bm, lbound = 0)
    }
  }

  return(E)
}
alg$bVolTau$rE(gW = IM30x30D2@W, gX = IM30x30D2[[]], gSigma2 = 1, gGamma = 0.01,
            previousE = IM30x30D2@E)
```

```
##       [,1]  [,2]  [,3]
## [1,] 0.306 1.037 0.387
## [2,] 0.217 0.485 0.763
```

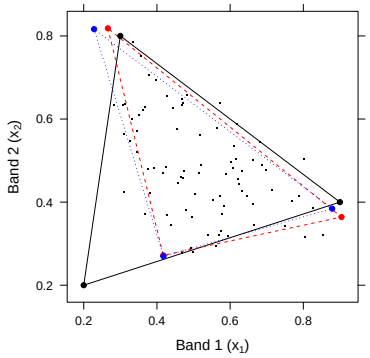
14.2.4 Joint $(\boldsymbol{E}, \boldsymbol{W}, \sigma^2)$ sampler

```
alg$bVolTau$sample <- function(nrSamples = PARAMS$SAMPLESIZE, hyImg = NA, gX = hyImg[[]],
                               gAlpha = 0, gBeta = 0, gGamma = 0,
                               burnIn = PARAMS$BURNINSIZE, M = ncol(gX) + 1,
                               initW = hy$rW(N = nrow(gX), M = M),
                               initE = hy$rE(M = M, D = ncol(gX), maxval = max(gX),
                                             wavelengths = hyperSpec::wl(hyImg)),
                               initTau = runif(ncol(initE)),
                               height = hy$height(hyImg), width = hy$width(hyImg)){

  scan <- list('sigma2' = NA, 'E' = initE, 'W' = initW, 'Tau' = initTau)

  samples <- alg$predictionSamples(S1 = matrix(nrow = nrSamples + burnIn,
                                               ncol = length(unlist(scan)),
                                               dimnames = list(NULL, names(unlist(scan)))),
                                   skelet = scan)

  for (i in 1:nrow(samples@S1)) {
    scan$Tau <- alg$bVolTau$rTau(gW = scan$W)
    scan$sigma2 <- alg$bVol$rSigma2(gX = gX, gE = scan$E, gW = scan$W,
                                    gAlpha = gAlpha, gBeta = gBeta)
    scan$W <- alg$bVolTau$rW(gX = gX, gSigma2 = scan$sigma2,
                             gE = scan$E, previousW = scan$W,
                             gTau = scan$Tau)
    scan$E <- alg$bVolTau$rE(gX = gX, gSigma2 = scan$sigma2, gGamma = gGamma,
                             gW = scan$W, previousE = scan$E)

    samples@S1[i,] <- unlist(scan)

    if (i %% 200 == 0 && PARAMS$LOG) {print(i)}
  }

  if (burnIn > 0) { samples@S1 <- samples@S1[-c(1:burnIn), ] }

  return(alg$predictionBayes(samples = samples, height = height, width = width))
}
```

14.3 Testing

14.3.1 Example

```
IM9x9D2@pred[['Bayes-Tau']] <- alg$bVolTau$sample(nrSamples = 500, hyImg = IM9x9D2, burnIn = 200)
IM9x9D2@pred[['Bayes']] <- alg$bVol$sample(nrSamples = 500, hyImg = IM9x9D2, burnIn = 200)
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E,
               Ehat = list('BayesVol' = IM9x9D2@pred[['Bayes']]$E,
                           'BayesVol-Tau' = IM9x9D2@pred[['Bayes-Tau']]$E))
```

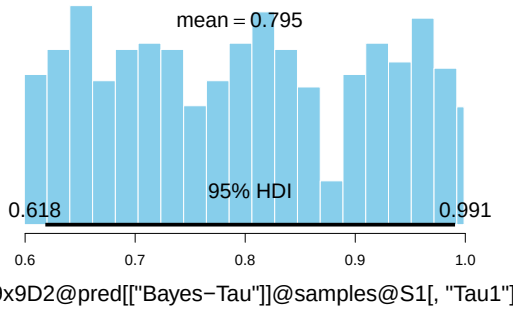



```
## [1] "BayesVol color: red"
## [1] "BayesVol-Tau color: blue"
```

```
IM9x9D2@pred[['Bayes-Tau']]$Tau
```

```
## [1] 0.669 0.982 0.994
```

```
BEST::plotPost(IM9x9D2@pred[['Bayes-Tau']]@samples@S1[,'Tau1'])
```

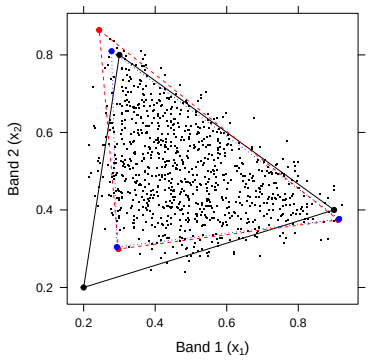


14.3.2 Comparison with BayesNMF-Vol

```
set.seed(PARAMS$SEED)
IM30x30D2@pred[['Bayes-Tau']] <- alg$bVolTau$sample(hyImg = IM30x30D2, gGamma = 0)
IM30x30D2@pred[['Bayes']] <- alg$bVol$sample(hyImg = IM30x30D2, gGamma = 0)

saveRDS(IM30x30D2@pred, file = './export/IM30x30D2@pred.rds')
```

```
IM30x30D2@pred <- readRDS(file = './export/IM30x30D2@pred.rds')
```

```
hy$plotSimplex(X = IM30x30D2[[]], E = IM30x30D2@E,
               Ehat = list('BayesVol-Tau' = IM30x30D2@pred[['Bayes-Tau']]$E,
                           'BayesVol' = IM30x30D2@pred[['Bayes']]$E),
               printAs = 'BayesVol-Default-vs-Tau')
```



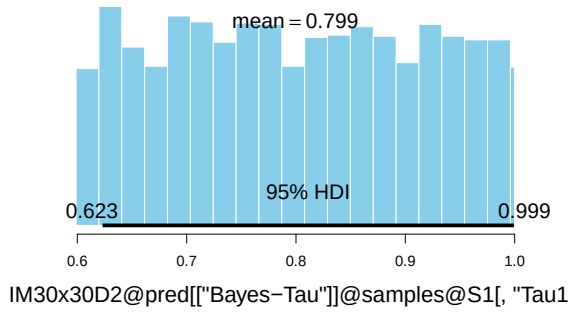

```
## [1] "BayesVol-Tau color: red"
## [1] "BayesVol color: blue"
```

```
## pdf
##   2
```

```
IM30x30D2@pred[['Bayes-Tau']]$Tau
```

```
## [1] 0.706 0.977 0.989
```

```
BEST::plotPost(IM30x30D2@pred[['Bayes-Tau']]@samples@S1[,'Tau1'])
```



15 Getting abundances based on endmembers

If one knows E, then this algorith will extract the W matrix using the reverse process of ICE algorithm.

15.1 Interface for the objective function

We are estimating the vector A which has the gain factors for each of the endmembers. So we are bascially searching for optimal A and W.

```
alg$abun$interface <- function(A, X, E, W, mu){
  alg$ice$Lreg(X = X, E = E %*% diag(A), W = W, mu = mu)
}
alg$abun$interface(A = c(1,1,1), X = IM2x2D2[[]], E = IM2x2D2@E, W = IM2x2D2@W, mu = 0.01)
```

```
## [1] 0.00221
```

```
optim(par = rep(1, 3), fn = alg$abun$interface, lower = 0, upper = 100000,
      X = IM9x9D2[[]], E = IM9x9D2@E, W = IM9x9D2@W, mu = 0.01, method = 'L-BFGS-B')
```

```
## $par
```

```
## [1] 1.230 0.973 0.981
##
## $value
## [1] 0.00211
##
## $counts
## function gradient
##       10       10
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
alg$abun$minimizeA <- function(X, E, W, mu){
  optim(par = rep(1, ncol(E)), fn = alg$abun$interface, lower = 0, upper = 100000,
        X = X, E = E, W = W, mu = mu, method = 'L-BFGS-B')$par
}
alg$abun$minimizeA(X = IM9x9D2[[]], E = IM9x9D2@E, W = IM9x9D2@W, mu = 0.01)
```

```
## [1] 1.230 0.973 0.981
```

```
alg$abun$solve <- function(hyImg = NULL, X = hyImg[[]], mu = 0.01, tol = PARAMS$ICETOL,
                           height = if (!is.null(hyImg)) {hy$height(hyImg)} else {numeric(0)},
                           width = if (!is.null(hyImg)) {hy$width(hyImg)} else {numeric(0)},
                           E){
  L <- vector('list'); A <- vector('list'); W <- vector('list')
  k <- 1
  W[[k]] <- hy$rW(M = ncol(E), N = nrow(X), height = height, width = width)
  A[[k]] <- rep(1, ncol(E))
  L[[k]] <- alg$ice$Lreg(X = X, E = E %*% diag(A[[k]]), W = W[[k]], mu = mu)

  repeat {
    k <- k + 1
    # It is not necessary to give the last W as initial values, but
    # seems appropriate since the solution is probably close.
    A[[k]] <- alg$abun$minimizeA(X = X, W = W[[k-1]], E = E, mu = mu)
    W[[k]] <- alg$ice$minimizeW(X = X, E = E %*% diag(A[[k]]), Winit = W[[k-1]])
    L[[k]] <- alg$ice$Lreg(X = X, E = E %*% diag(A[[k]]), W = W[[k]], mu = mu)
    alg$ice$logStep(step = k, L = L)
    stopifnot(L[[k]]/L[[k-1]] <= 1) # coordinate descent function value check
    if (L[[k]]/L[[k-1]] > tol) {break}
  }

  cat(paste("ICE convergence after",k,"steps\n"))

  structure(A[[k]], W = W[[k]], steps = k)
}
alg$abun$solve(hyImg = IM2x2D2, mu = 0.01, tol = 0.99999, E = IM2x2D2@E)
```

```
## ICE step: 0010 | Imp: -0.0421542356466381  | Ratio: 0.994836283435714
## ICE step: 0020 | Imp: -0.00000632894912663937  | Ratio: 0.99986820357598
## ICE convergence after 28 steps
```

```
## [1] 2.873 0.427 0.454
## attr(,"W")
## An object of class "W"
##       [,1]  [,2]  [,3]
## [1,] 0.000 0.720 0.280
```

```
## [2,] 0.588 0.000 0.412
## [3,] 0.516 0.484 0.000
## [4,] 0.123 0.268 0.609
## Slot "width":
## [1] 2
##
## Slot "height":
## [1] 2
##
## Slot "origin":
## [1] "ICE_W_MINIMIZER"
##
## attr(,"steps")
## [1] 28
```

16 Benchmarking and optimizing

16.1 Procedures for optimizing hyper-parameters

```
bm$resolve <- function(parametersdf, fun, envir, refE, ...){
  print(Sys.time())

  pp$update()

  result <- parametersdf
  `%dopar%` <- foreach::`%dopar%` # for not loading foreach package

  # the output of foreach will be returned
  result <- foreach::foreach(i = 1:nrow(parametersdf),
                             .combine = rbind, .verbose = FALSE) %dopar% {
    pred <- do.call(envir = envir, what = fun,
                    args = c(as.list(parametersdf[i,,drop=FALSE]), list(...)))
    result[i, 'SAM'] <- measure$SAM(refE, pred$E)
    # hack to add E and W into df
    result[i, 'E'][[1]] <- list('E' = pred$E)
    #result[i, 'W'][[1]] <- list('W' = pred$W)
    result[i, 'steps'] <- if (is.null(pred$steps)) {'-'} else {pred$steps}
    print(as.matrix(result[i,]))
    result[i,] # is also the output of each iteration!
  }

  print(Sys.time())

  return(result)
}
```

16.1.1 Quick examples

16.1.1.1 ICE

```
bm$test[['ice']] <- bm$resolve(
  parametersdf = do.call("rbind", replicate(2, simplify = FALSE, data.frame(
      'mu' = exp(seq(log(0.00005), log(0.99), length.out = 10))))),
  fun = 'ICE', envir = alg$ice, refE = IM9x9D2@E,
  hyImg = IM9x9D2, M = 3, tol = 0.9)
```

```
## [1] "2017-05-31 22:44:49 CEST"
## [1] "2017-05-31 22:44:56 CEST"
```

16.1.1.2 ICE-S

```
bm$test[['ice-s']] <- bm$resolve(
  parametersdf =
    do.call("rbind", replicate(1, simplify = FALSE, expand.grid(
      'mu' = exp(seq(log(0.0001), log(0.99), length.out = 5)),
      'nu' = exp(seq(log(0.001), log(1), length.out = 5))))),
  fun = 'ICE', envir = alg$iceS, refE = IM9x9D2@E,
  hyImg = IM9x9D2, M = 3, tol = 0.9)
```

```
## [1] "2017-05-31 22:44:56 CEST"
## [1] "2017-05-31 22:45:04 CEST"
```

16.1.1.3 BayesNMF-Vol

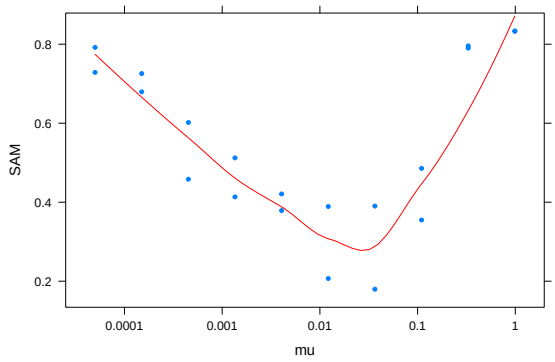
```
bm$test[['bayes']] <- bm$resolve(
  parametersdf =
    do.call("rbind", replicate(2, simplify = FALSE, data.frame(
      'gGamma' = c(rep(0,2), exp(seq(log(1e-06), log(1e5), length.out = 10))))),
  fun = 'sample', envir = alg$bVol, refE = IM9x9D2@E,
  hyImg = IM9x9D2, M = 3, nrSamples = 10, burnIn = 10)
```

```
## [1] "2017-05-31 22:45:04 CEST"
## [1] "2017-05-31 22:45:08 CEST"
```

16.2 Plotting the hyperparameters

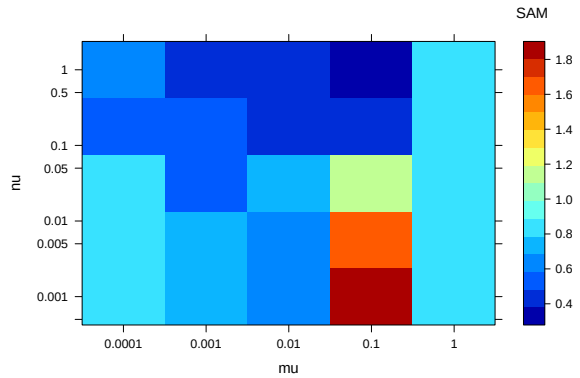
```
bm$plotParameters <- function(df, formula, min = FALSE, loess = TRUE, ylog = FALSE,
                              printAs = character(0)){

  plot1D <- function(){
    plt <- lattice::xyplot(
      par.settings = list("clip" = list('panel' = 'off')), # for adjusting axis from panel
      x = formula, data = df,
      scales = list(x = list(log=10), equispaced.log=FALSE,
                    y = if (ylog) list(log=10) else list()),
      panel = function(x, y, ...){
        if (any(x == -Inf)) {
          x[x == -Inf] <- lattice::current.panel.limits()$xlim[1] + 0.025 # x allready in log
          lattice::panel.axis(side = 'bottom', outside = TRUE, labels = 0, rot = 0,
                              at = lattice::current.panel.limits()$xlim[1])
        }
        lattice::panel.xyplot(x, y, ..., pch = 20)
        minID <- which(y == min(y))
        print(paste('Min SAM:', min(y), 'x:', 10^(x[minID]), 'ID:', minID))
        if (min) lattice::panel.abline(v = x[minID], lty = "dotted", col = "black")
        if (loess) lattice::panel.loess(x, y, degree = 2, col = 'red')
      })

    print(plt)
  }

  plot2D <- function(){
    # take the maximum (worst) SAM value
```

```
    df <- aggregate(formula, df, max)

    plt <- lattice::levelplot(
      x = formula, data = df,
      scales = list(log=TRUE, equispaced.log=FALSE),
      col.regions = colorRamps::matlab.like(256),
      colorkey = list(),
      legend=list(top = list(fun=grid::textGrob('SAM', y=0, x=1.09)))
    )

    print(plt)
  }

  switch (length(attr(terms(formula), "term.labels")),
    '1' = plot1D,
    '2' = plot2D,
    stop("Can only plot up to 2 dimensions!")
  )()

  if (length(printAs) != 0) {
    dev.copy2pdf( file = paste('./../fig/', printAs,
                               '-parameter-comparison.pdf', sep = ""),
                  compress = FALSE )
  }
}
bm$plotParameters(df = bm$test[['ice']], formula = SAM ~ mu)
```

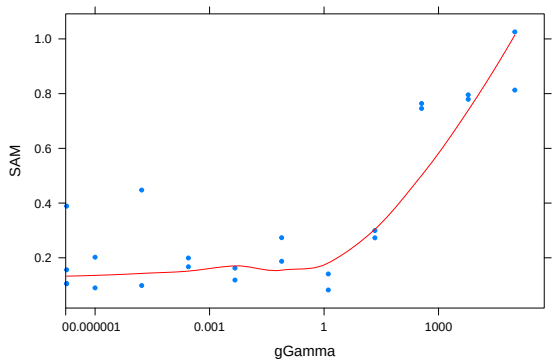



```
## [1] "Min SAM: 0.179781427821658 x: 0.0365943017197028 ID: 7"
```

```
bm$plotParameters(df = bm$test[['ice-s']], formula = SAM ~ mu + nu)
```



```
bm$plotParameters(df = bm$test[['bayes']], formula = SAM ~ gGamma)
```




```
## [1] "Min SAM: 0.0823434566581682 x: 1.29154966501489 ID: 8"
```

17 Synthetic data

17.1 Partial mixing in blocks, D = 2, M = 3

17.1.1 Generate hyperSpec image

```
IM50x50D2.P3MB <- hy$hyperSpecExt(width = 50, height = 50, depth = 2)
```

17.1.2 Generate W

First define groups for in the picture, based on expected values and variances. We calculate the Dirichlet parameter α for each group.

```
IM50x50D2.P3MB@usd <- within(IM50x50D2.P3MB@usd, {
  G <- list()
  G[[1]] <- hy$diricletParameters(ExpectedW1 = 0.8, ExpectedW2 = 0.001, VarW1 = 0.0005)
  G[[2]] <- hy$diricletParameters(ExpectedW1 = 0.0001, ExpectedW2 = 0.90, VarW1 = 0.0000005)
  G[[3]] <- hy$diricletParameters(ExpectedW1 = 0.00001, ExpectedW2 = 0.005, VarW1 = 0.000005)
})
IM50x50D2.P3MB@usd$G
```

```
## [[1]]
## [[1]]$a0
## [1] 319
##
## [[1]]$a
## [1] 255.200   0.319  63.481
##
## [[1]]$EX
## [1] 0.800 0.001 0.199
##
## [[1]]$CovX
##              [,1]         [,2]         [,3]
## [1,]  0.0005000 -0.000002500 -0.000497500
## [2,] -0.0000025  0.000003122 -0.000000622
## [3,] -0.0004975 -0.000000622  0.000498122
##
##
```

```
## [[2]]
## [[2]]$a0
## [1] 199
##
## [[2]]$a
## [1]   0.0199 179.0820  19.8781
##
## [[2]]$EX
## [1] 0.0001 0.9000 0.0999
##
## [[2]]$CovX
##             [,1]        [,2]        [,3]
## [1,]  0.00000050 -0.00000045 -0.00000005
## [2,] -0.00000045  0.00045005 -0.00044959
## [3,] -0.00000005 -0.00044959  0.00044964
##
##
## [[3]]
## [[3]]$a0
## [1] 1
##
## [[3]]$a
## [1] 0.00001 0.00500 0.99497
##
## [[3]]$EX
## [1] 0.00001 0.00500 0.99499
##
## [[3]]$CovX
##              [,1]         [,2]        [,3]
## [1,]  0.000005000 -0.000000025 -0.00000497
## [2,] -0.000000025  0.002487525 -0.00248750
## [3,] -0.000004975 -0.002487500  0.00249247
```

Next, for each pixel in the picture, we set the Dirichlet parameters.

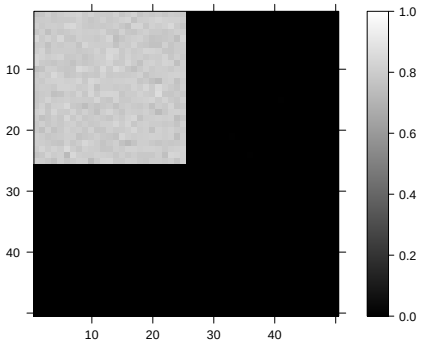
```
IM50x50D2.P3MB@usd <- within(IM50x50D2.P3MB@usd, {
  M <- aperm(array(dim = c(50,50,3)), perm = c(3,1,2)) # make last dim first for filling it.
  M[,1:25,1:25] <- G[[1]]$a # set G1 paramteres for top left 25 pixels etc...
  M[,26:50,1:25] <- G[[2]]$a
  M[,1:50,26:50] <- G[[3]]$a
  M[,15:35,35:40] <- G[[2]]$a
  M <- aperm(M, perm = c(2,3,1))
})
IM50x50D2.P3MB@usd$M[1,1,]
```

```
## [1] 255.200   0.319  63.481
```

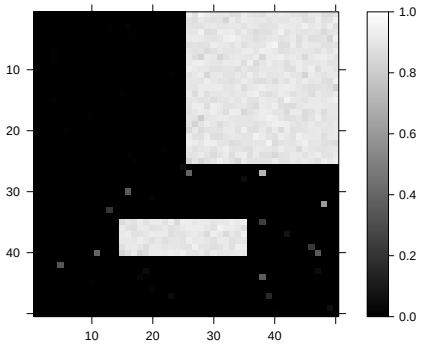
Compute the W based on Dirichlet parameters M for the picture.

```
IM50x50D2.P3MB@W <- hy$W(
  matrix(aperm(apply(IM50x50D2.P3MB@usd$M,
                     MARGIN = c(1,2),
                     FUN = hy$rdirichlet)),
         ncol = 3, byrow = FALSE),
  width = 50, height = 50, origin = 'original')
hy$plotMapW(IM50x50D2.P3MB@W, printAs = 'IM50x50D2.P3MB')
```

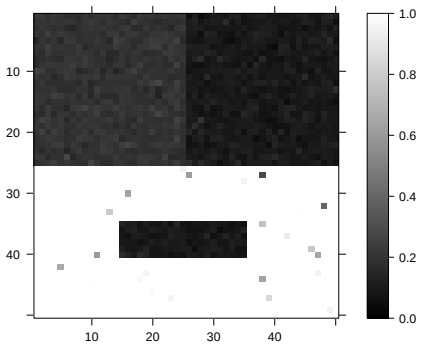
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```



17.1.3 Generate X (i.e. add noise)

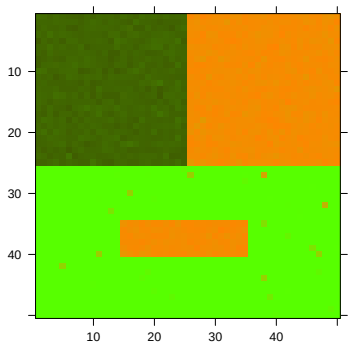
```
IM50x50D2.P3MB@E <- Ed2
IM50x50D2.P3MB@sigma2 <- hy$noiseVarFromSNR(
  varSignal = var(as.vector(IM50x50D2.P3MB@W %*% t(IM50x50D2.P3MB@E))),
  SNRdB = 15)
IM50x50D2.P3MB[[]] <- IM50x50D2.P3MB@W %*% t(IM50x50D2.P3MB@E) + rnorm(50*50*2, sd = sqrt(IM50x50D2.
```

17.1.4 False Image plot

```
hy$plotfc(hy$hyperSpecExt(spc = IM50x50D2.P3MB@W %*% t(IM50x50D2.P3MB@E),
                          width = 50, height = 50, depth = 2),
```

```
          printAs = 'IM50x50D2.P3MB-NoNoise')
```

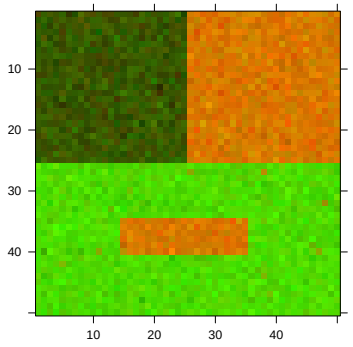
```
## [1] "Wavelengths for falsecolor image: 1, 2"
```




```
## pdf
##   2
```

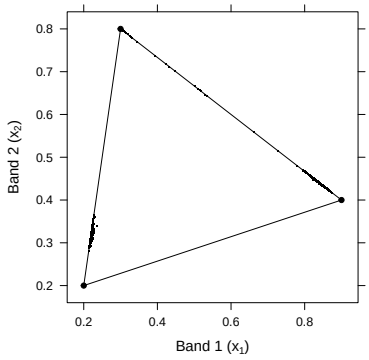
```
hy$plotfc(IM50x50D2.P3MB, printAs = 'IM50x50D2.P3MB')
```

```
## [1] "Wavelengths for falsecolor image: 1, 2"
```




```
## pdf
##   2
```

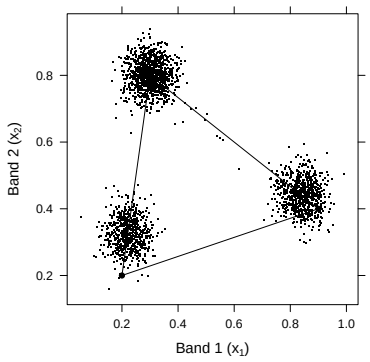
```
hy$plotSimplex(IM50x50D2.P3MB@W %*% t(IM50x50D2.P3MB@E), E = IM50x50D2.P3MB@E,
               printAs = 'IM50x50D2.P3MB-NoNoise')
```




```
## [1] " color: red"
```

```
## pdf
##   2
```

```
hy$plotSimplex(IM50x50D2.P3MB[[]], E = IM50x50D2.P3MB@E,
               printAs = 'IM50x50D2.P3MB')
```




```
## [1] " color: red"
```

```
## pdf
##   2
```

17.1.5 Running the Algorithms

```
IM50x50D2.P3MB@pred[['Bayes']] <-
  alg$bVol$sample(hyImg = IM50x50D2.P3MB, M = 3)
IM50x50D2.P3MB@pred[['ICE']] <-
  alg$ice$ICE(hyImg = IM50x50D2.P3MB, M = 3)
```

```
## ICE convergence after 50 steps
```

```
IM50x50D2.P3MB@pred[['ICE-0.02-S-0.5']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.P3MB, M = 3, mu = 0.02, nu = 0.5)
```

```
## ICE convergence after 83 steps
```

```
IM50x50D2.P3MB@pred[['ICE-0.1-S-0.1']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.P3MB, M = 3, mu = 0.1, nu = 0.1)
```

```
## ICE convergence after 72 steps
```

```
IM50x50D2.P3MB@pred[['ICE-0.5-S-0.02']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.P3MB, M = 3, mu = 0.5, nu = 0.02)
```

```
## ICE convergence after 105 steps
```

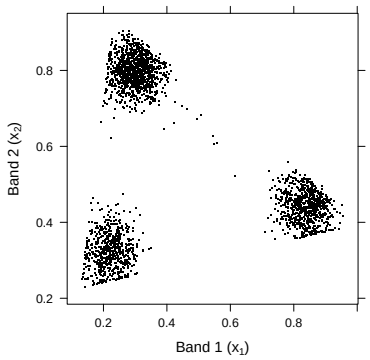
```
saveRDS(IM50x50D2.P3MB@pred, file = './export/IM50x50D2.P3MB@pred.rds')
```

```
IM50x50D2.P3MB@pred <- readRDS(file = './export/IM50x50D2.P3MB@pred.rds')
```

17.1.6 Predicted abundance maps

17.1.6.1 BayesNMF-Vol

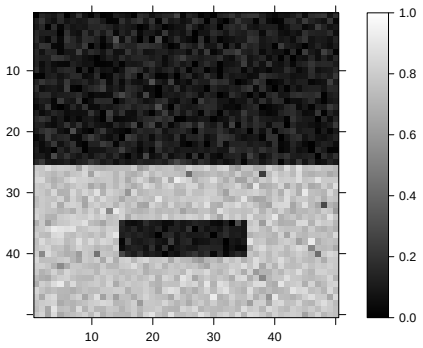
```
hy$plotSimplex(X = IM50x50D2.P3MB@pred$`Bayes`$W %*% t(IM50x50D2.P3MB@pred$`Bayes`$E))
```



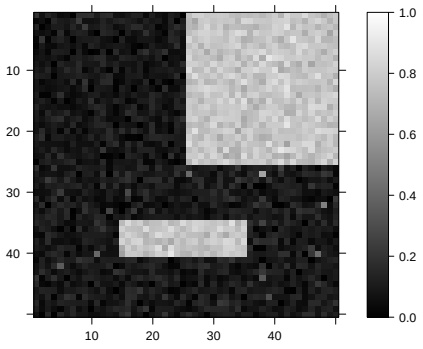

```
## [1] " color: red"
```

```
hy$plotMapW(IM50x50D2.P3MB@pred$`Bayes`$W, printAs = 'IM50x50D2.P3MB',
            suffix = 'bayes-map')
```

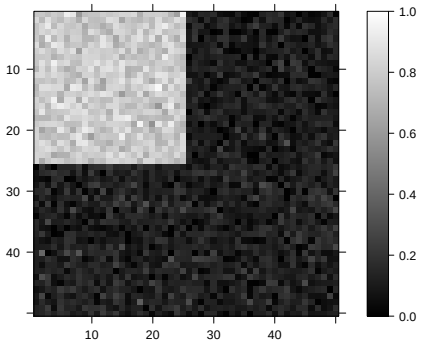
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

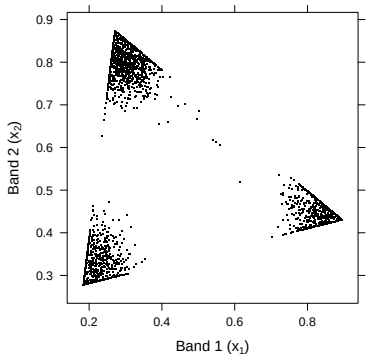



```
## [1] "Plotting: E3"
```



17.1.6.2 ICE

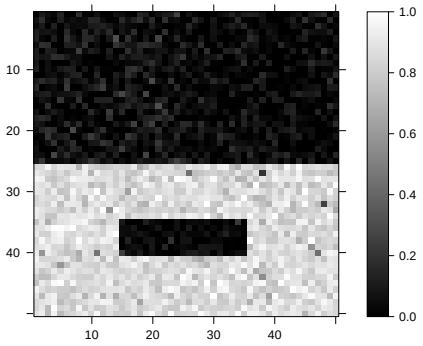
```
hy$plotSimplex(X = IM50x50D2.P3MB@pred$`ICE`$W %*% t(IM50x50D2.P3MB@pred$`ICE`$E))
```



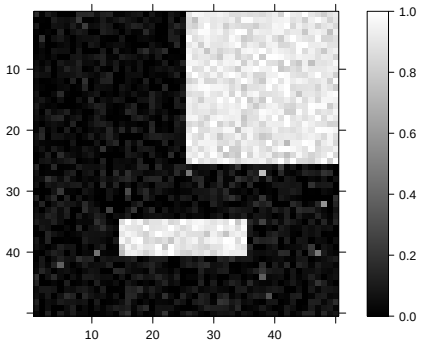

```
## [1] " color: red"
```

```
hy$plotMapW(IM50x50D2.P3MB@pred$`ICE`$W, printAs = 'IM50x50D2.P3MB',
            suffix = 'ice')
```

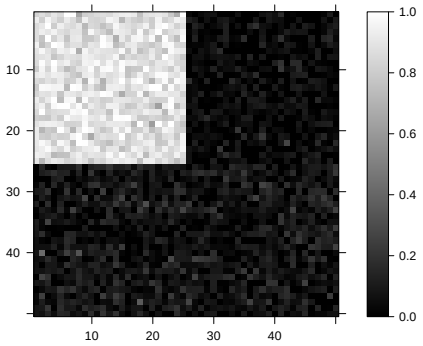
```
## Problematic W-values:
##  Count: 74
##  Min: -1.11022302462516e-16
##  Max: 1
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

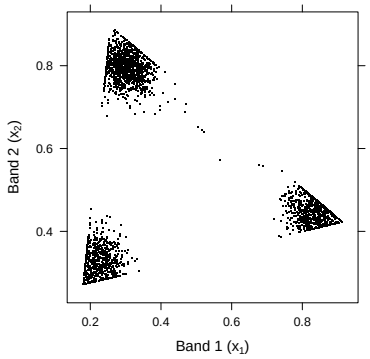



```
## [1] "Plotting: E3"
```



17.1.6.3 ICE with spatial regularization

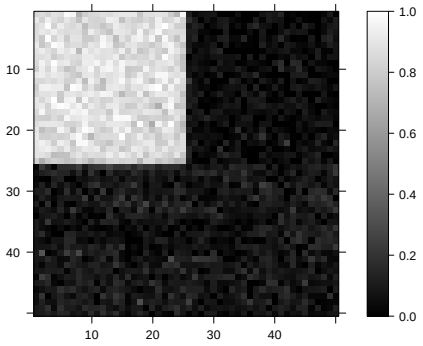
```
hy$plotSimplex(X = IM50x50D2.P3MB@pred$`ICE-0.1-S-0.1`$W %*% t(IM50x50D2.P3MB@pred$`ICE-0.1-S-0.1`$E
```



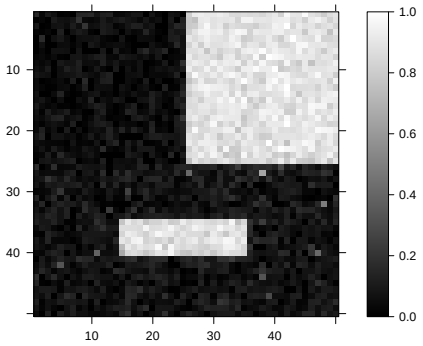

```
## [1] " color: red"
```

```
hy$plotMapW(IM50x50D2.P3MB@pred$`ICE-0.1-S-0.1`$W, printAs = 'IM50x50D2.P3MB',
            suffix = 'ice-0.1-s-0.1')
```

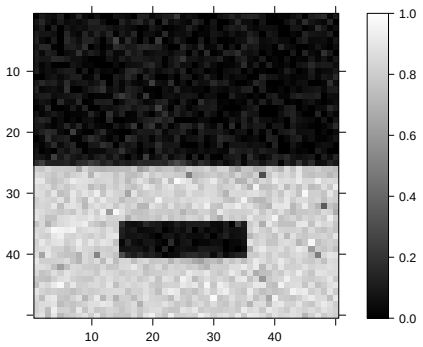
```
## Problematic W-values:
##  Count: 39
##  Min: -5.55111512312578e-17
##  Max: 1
## [1] "Plotting: E1"
```



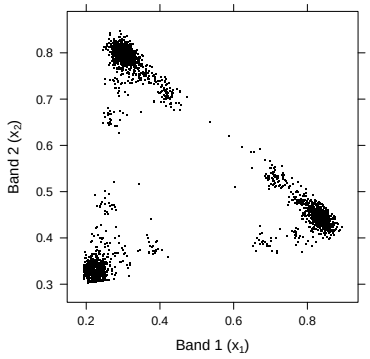

```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```



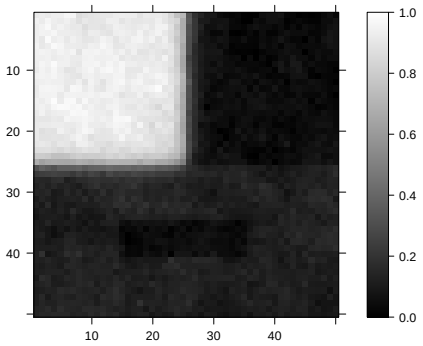

```
hy$plotSimplex(X = IM50x50D2.P3MB@pred$`ICE-0.5-S-0.02`$W %*% t(IM50x50D2.P3MB@pred$`ICE-0.5-S-0.02`
```



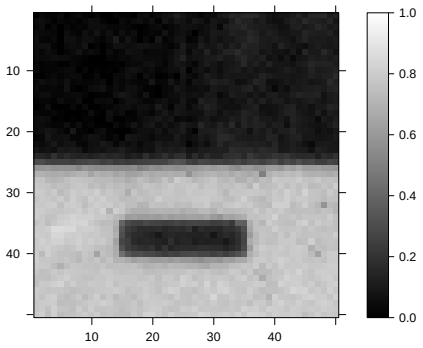
```
## [1] " color: red"
```

```
hy$plotMapW(IM50x50D2.P3MB@pred$`ICE-0.5-S-0.02`$W, printAs = 'IM50x50D2.P3MB',
            suffix = 'ice-0.5-s-0.02')
```

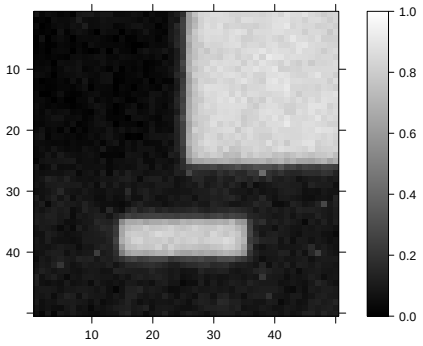
```
## Problematic W-values:
##  Count: 2
##  Min: -1.0842021724855e-19
##  Max: -2.71050543121376e-20
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```



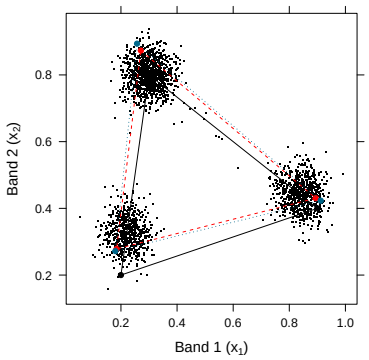

```
## [1] "Plotting: E3"
```



17.1.7 Predicted endmembers

17.1.7.1 General comparison

```
hy$plotSimplex(IM50x50D2.P3MB[[]], E = IM50x50D2.P3MB@E,
               Ehat = list(#'E-BayesNMF-Vol' = IM50x50D2.P3MB@pred$`Bayes`$E,
                           'ICE' = IM50x50D2.P3MB@pred$`ICE`$E,
                           'ICE-0.1-S-0.1' = IM50x50D2.P3MB@pred$`ICE-0.1-S-0.1`$E
                           ),
               EhatCol = c('red', 'deepskyblue4'))
```

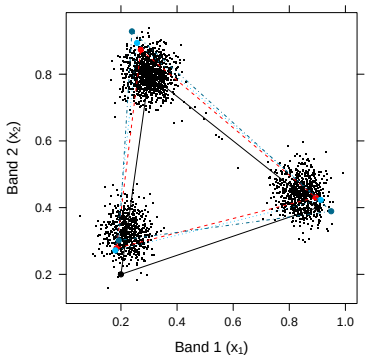



```
## [1] "ICE color: red"
## [1] "ICE-0.1-S-0.1 color: deepskyblue4"
```

17.1.7.2 ICE comparison hyper parameter

17.1.7.3 Effect of spatial regularization with same simplex hyper-param

```
hy$plotSimplex(IM50x50D2.P3MB[[]], E = IM50x50D2.P3MB@E,
               Ehat = list('ICE' = IM50x50D2.P3MB@pred$`ICE`$E,
                           #'ICE-0.02-S-0.5' = IM50x50D2.P3MB@pred$`ICE-0.02-S-0.5`$E,
                           'ICE-0.1-S-0.1' = IM50x50D2.P3MB@pred$`ICE-0.1-S-0.1`$E,
                           'ICE-0.5-S-0.02' = IM50x50D2.P3MB@pred$`ICE-0.5-S-0.02`$E
                           ),
               EhatCol = c('red',
                           'deepskyblue',
                           #'deepskyblue2',
                           'deepskyblue4'),
               printAs = 'IM50x50D2.P3MB-pred-spatial-reg-same-vol')
```




```
## [1] "ICE color: red"
## [1] "ICE-0.1-S-0.1 color: deepskyblue"
## [1] "ICE-0.5-S-0.02 color: deepskyblue4"
```

```
## pdf
##   2
```

17.2 Mixing based on MRF generated abundances

17.2.1 Generate hyperSpec image

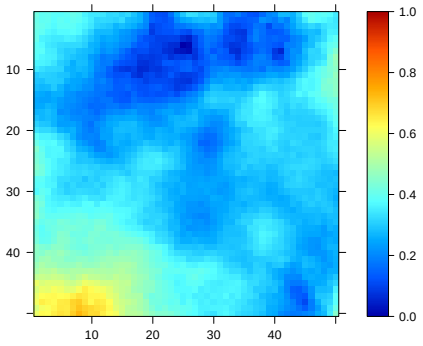
```
IM50x50D2.Gauss <- hy$hyperSpecExt(width = 50, height = 50, depth = 2)
```

17.2.2 Generate W

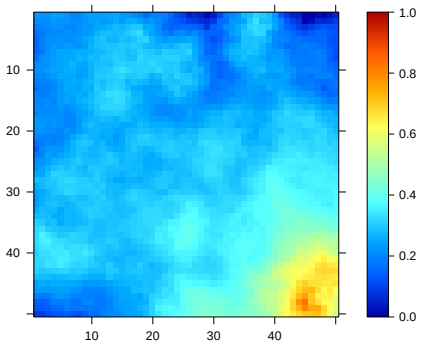
We load W.

```
IM50x50D2.Gauss@W <- hy$W(width = 50, height = 50, origin = 'original',
                          data = array(readRDS(file = './../data/abundances/vgm/abun-50x50.rds'),
                                       dim = c(50*50, 3)))
hy$plotMapW(W = IM50x50D2.Gauss@W, bw = FALSE, printAs = 'IM50x50D2.Gauss')
```

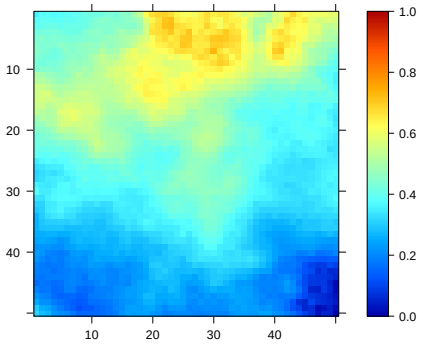
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```



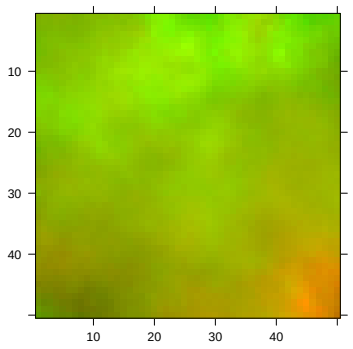
17.2.3 Generate X

```
IM50x50D2.Gauss@E <- Ed2
IM50x50D2.Gauss@sigma2 <- hy$noiseVarFromSNR(
  varSignal = var(as.vector(IM50x50D2.Gauss@W %*% t(IM50x50D2.Gauss@E))),
  SNRdB = 15)
IM50x50D2.Gauss[[]] <- IM50x50D2.Gauss@W %*% t(IM50x50D2.Gauss@E) +
  rnorm(50*50*2, sd = sqrt(IM50x50D2.Gauss@sigma2))
```

17.2.4 False Image plot

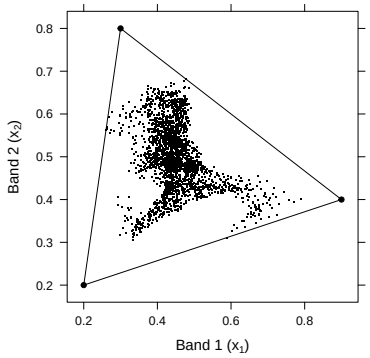
```
hy$plotfc(hy$hyperSpecExt(spc = IM50x50D2.Gauss@W %*% t(IM50x50D2.Gauss@E),
                          width = 50, height = 50, depth = 2),
          printAs = 'IM50x50D2.Gauss-NoNoise')
```

```
## [1] "Wavelengths for falsecolor image: 1, 2"
```




```
## pdf
##   2
```

```
hy$plotSimplex(IM50x50D2.Gauss@W %*% t(IM50x50D2.Gauss@E), E = IM50x50D2.Gauss@E,
               printAs = 'IM50x50D2.Gauss-NoNoise')
```

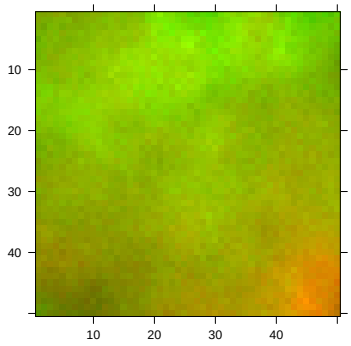



```
## [1] " color: red"
```

```
## pdf
##   2
```

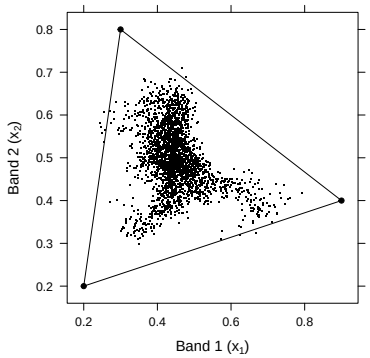
```
hy$plotfc(IM50x50D2.Gauss, printAs = 'IM50x50D2.Gauss')
```

```
## [1] "Wavelengths for falsecolor image: 1, 2"
```




```
## pdf
##   2
```

```
hy$plotSimplex(IM50x50D2.Gauss[[]], E = IM50x50D2.Gauss@E,
               printAs = 'IM50x50D2.Gauss')
```




```
## [1] " color: red"
```

```
## pdf
##   2
```

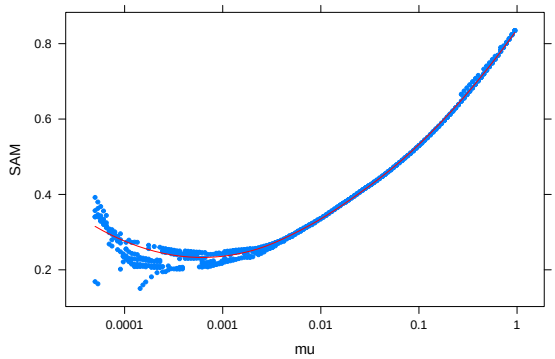
17.2.5 Exploring hyperparameters

17.2.5.1 Optimizing ICE hyper-paramter

```
# 1 step takes 2.15 sec
# avg.steps for stepTol = 1000: 243
# Time: 2.15 * avg.steps * runs / cores = x seconds
IM50x50D2.Gauss@usd[['ICE-HP']] <- bm$resolve(
  parametersdf =
    do.call("rbind", replicate(5, simplify = FALSE, data.frame(
      'mu' = exp(seq(log(0.00005), log(0.95), length.out = 150))))),
  fun = 'ICE', envir = alg$ice, refE = IM50x50D2.Gauss@E,
  hyImg = IM50x50D2.Gauss, M = 3,
  stepTol = 1000)

saveRDS(IM50x50D2.Gauss@usd[['ICE-HP']], compress = FALSE, ascii = TRUE,
        file = 'export/IM50x50D2.Gauss@usd[[ICE-HP]].rds')
```

```
IM50x50D2.Gauss@usd[['ICE-HP']] <- readRDS(file = 'export/IM50x50D2.Gauss@usd[[ICE-HP]].rds')
```

```
bm$plotParameters(df = IM50x50D2.Gauss@usd[['ICE-HP']],
                  formula = SAM ~ mu, printAs = 'IM50x50D2.Gauss-ice')
```




```
## [1] "Min SAM: 0.150446819805377 x: 0.000144023538967569 ID: 467"
```

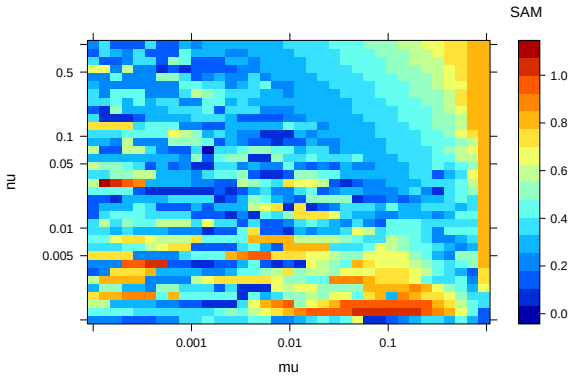
```
## pdf
##   2
```

17.2.5.2 Optimizing ICE-S hyper-paramters

```
# 1 step takes 3.03 sec
# avg.steps for stepTol = 500: 415
# Time: 3.03 * avg.steps * runs / cores = x seconds
IM50x50D2.Gauss@usd[['ICE-S-HP']] <- bm$resolve(
  parametersdf =
    do.call("rbind", replicate(1, simplify = FALSE, expand.grid(
      'mu' = exp(seq(log(0.0001), log(0.95), length.out = 35)),
      'nu' = exp(seq(log(0.001), log(1), length.out = 35))))),
  fun = 'ICE', envir = alg$iceS, refE = IM50x50D2.Gauss@E,
  hyImg = IM50x50D2.Gauss, M = 3,
  stepTol = 500)

saveRDS(IM50x50D2.Gauss@usd[['ICE-S-HP']], compress = FALSE, ascii = TRUE,
        file = 'export/IM50x50D2.Gauss@usd[[ICE-S-HP]].rds')
```

```
IM50x50D2.Gauss@usd[['ICE-S-HP']] <- readRDS(file = 'export/IM50x50D2.Gauss@usd[[ICE-S-HP]].rds')
```

```
bm$plotParameters(df = IM50x50D2.Gauss@usd[['ICE-S-HP']],
                  formula = SAM ~ mu + nu, printAs = 'IM50x50D2.Gauss-ice-s')
```



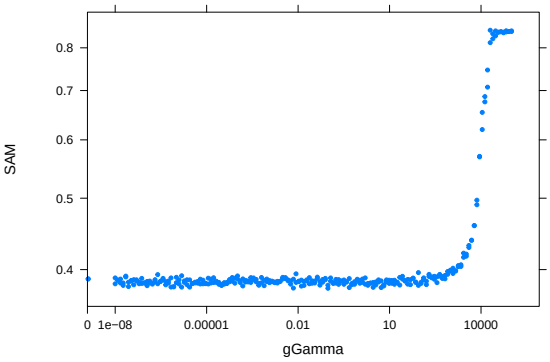

```
## pdf
##   2
```

17.2.5.3 Optimizing BayesNMF-Vol hyper-paramter

```
# 1 sample takes 0.350 sec
# Time: 0.500 * samples * runs / cores = x seconds
IM50x50D2.Gauss@usd[['Bayes-HP']] <- bm$resolve(
  parametersdf =
    do.call("rbind", replicate(2, data.frame(
      'gGamma' = c(rep(0,2), exp(seq(log(1e-08), log(1e5), length.out = 150)))),
      simplify = FALSE)),
  fun = 'sample', envir = alg$bVol, refE = IM50x50D2.Gauss@E,
  hyImg = IM50x50D2.Gauss, M = 3,
  nrSamples = 3000, burnIn = 500)

saveRDS(IM50x50D2.Gauss@usd[['Bayes-HP']], compress = FALSE, ascii = TRUE,
        file = 'export/IM50x50D2.Gauss@usd[[Bayes-HP]].rds')
```

```
IM50x50D2.Gauss@usd[['Bayes-HP']] <- readRDS(file = 'export/IM50x50D2.Gauss@usd[[Bayes-HP]].rds')
```

```
bm$plotParameters(df = IM50x50D2.Gauss@usd[['Bayes-HP']], loess = FALSE, ylog = TRUE,
                  formula = SAM ~ gGamma, printAs = 'IM50x50D2.Gauss-bayes')
```




```
## [1] "Min SAM: -0.423308777555162 x: 0.0954697470328751 ID: 231"
```

```
## pdf
##   2
```

17.2.6 Running the Algorithms

```
set.seed(PARAMS$SEED)
IM50x50D2.Gauss@pred[['ICE']] <-
  alg$ice$ICE(hyImg = IM50x50D2.Gauss, M = 3)
```

```
## ICE convergence after 57 steps
```

```
IM50x50D2.Gauss@pred[['Bayes']] <-
  alg$bVol$sample(hyImg = IM50x50D2.Gauss, M = 3)
IM50x50D2.Gauss@pred[['ICE-0.02-S-0.5']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.Gauss, M = 3, mu = 0.02, nu = 0.5)
```

```
## ICE convergence after 55 steps
```

```
IM50x50D2.Gauss@pred[['ICE-0.1-S-0.1']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.Gauss, M = 3, mu = 0.1, nu = 0.1)
```

```
## ICE convergence after 71 steps
```

```
IM50x50D2.Gauss@pred[['ICE-0.5-S-0.02']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.Gauss, M = 3, mu = 0.5, nu = 0.02)
```

```
## ICE convergence after 244 steps
```

```
IM50x50D2.Gauss@pred[['ICE-opt']] <-
  alg$ice$ICE(hyImg = IM50x50D2.Gauss, M = 3, mu = 0.0005)
```

```
## ICE convergence after 411 steps
```

```
IM50x50D2.Gauss@pred[['ICE-S-opt']] <-
  alg$iceS$ICE(hyImg = IM50x50D2.Gauss, M = 3, mu = 0.0015, nu = 0.55)
```

```
## ICE convergence after 306 steps
```

```
IM50x50D2.Gauss@pred[['Bayes-opt']] <-
  alg$bVol$sample(hyImg = IM50x50D2.Gauss, M = 3, gGamma = 10)

saveRDS(IM50x50D2.Gauss@pred, file = 'export/IM50x50D2.Gauss@pred.rds')
```

```
IM50x50D2.Gauss@pred <- readRDS(file = 'export/IM50x50D2.Gauss@pred.rds')
```

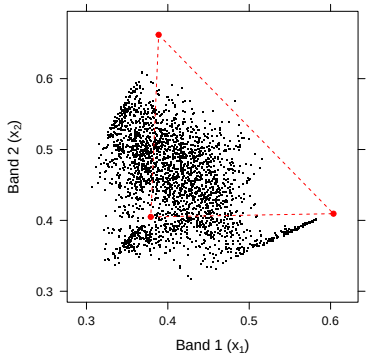
17.2.7 Predicted abundance maps

Helper function

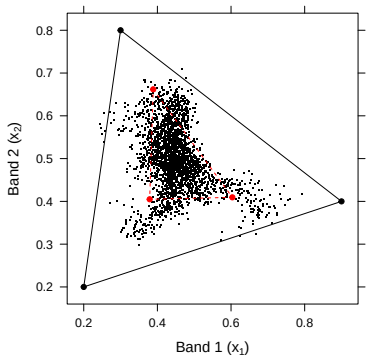
```
analysis <- function(hyImg = IM50x50D2.Gauss, predlbl, printAs = 'IM50x50D2.Gauss'){
  hy$plotSimplex(X = hyImg@pred[[predlbl]]$W %*% t(hyImg@pred[[predlbl]]$E),
                 Ehat = hyImg@pred[[predlbl]]$E)
  hy$plotSimplex(X = hyImg[[]], E = hyImg@E, Ehat = hyImg@pred[[predlbl]]$E,
                 printAs = paste(printAs, 'simplex', predlbl, sep = '-'))
  hy$plotMapW(hyImg@pred[[predlbl]]$W, bw = FALSE,
              printAs = printAs, suffix = tolower(predlbl))
  print(paste('SAM:', measure$SAM(hyImg@E, hyImg@pred[[predlbl]]$E)))
}
```

17.2.7.1 BayesNMF-Vol

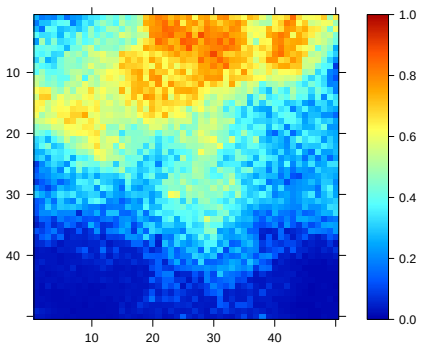
```
analysis(predlbl = 'Bayes')
```



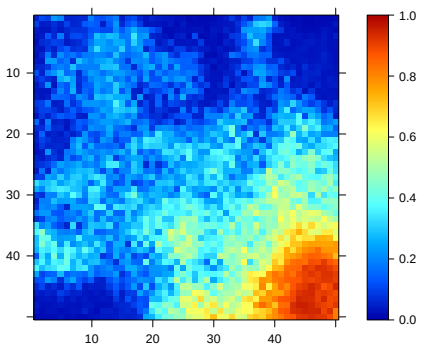

```
## [1] " color: red"
```



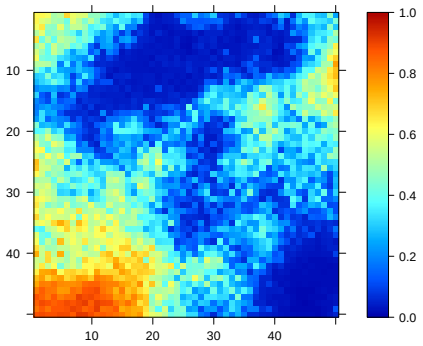

```
## [1] " color: red"
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

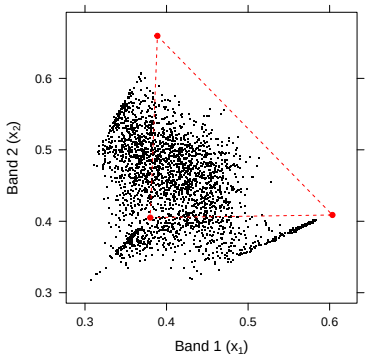


```
## [1] "Plotting: E3"
```

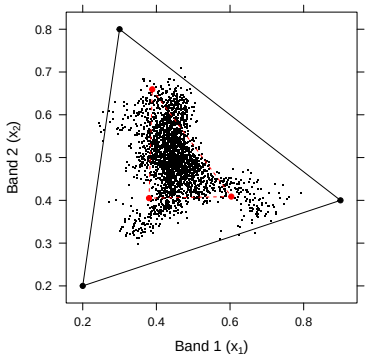



```
## [1] "SAM: 0.383025411771264"
```

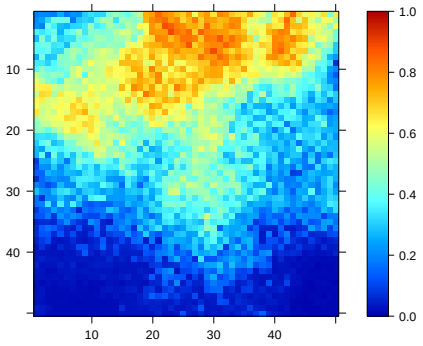
```
analysis(predlbl = 'Bayes-opt')
```



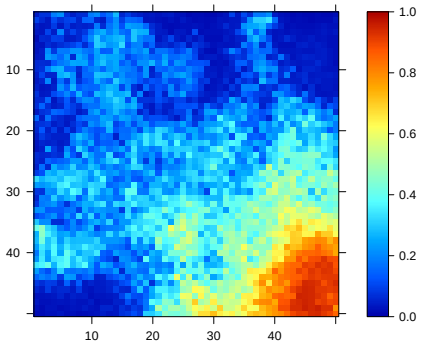

```
## [1] " color: red"
```



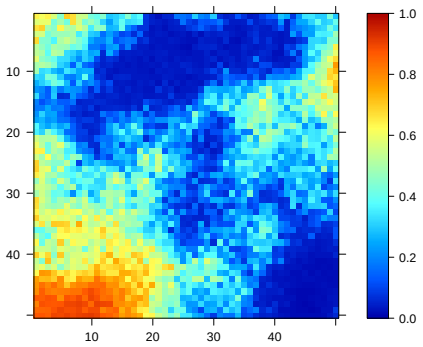

```
## [1] " color: red"
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```



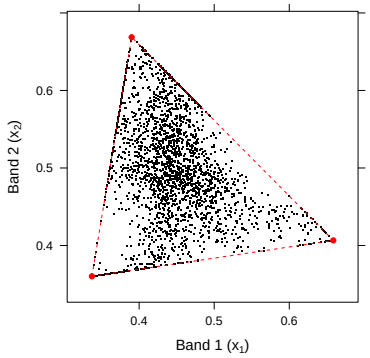

```
## [1] "Plotting: E3"
```



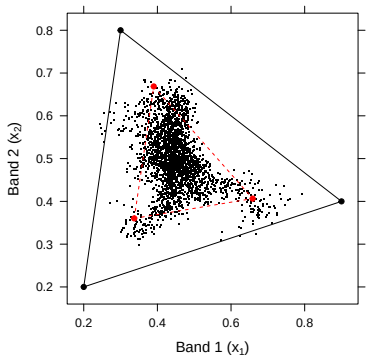

```
## [1] "SAM: 0.383014645879207"
```

17.2.7.2 ICE

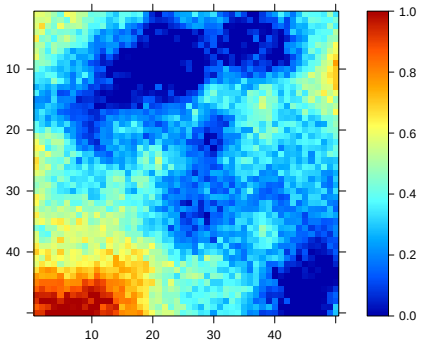
```
analysis(predlbl = 'ICE')
```



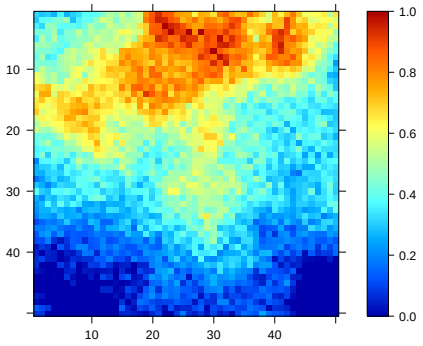

```
## [1] " color: red"
```



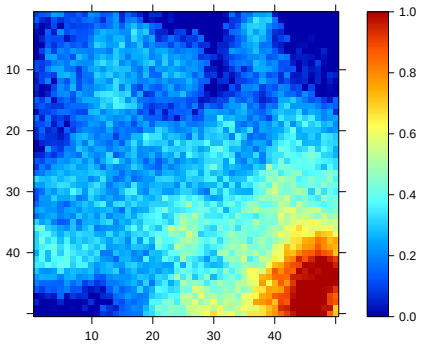

```
## [1] " color: red"
## Problematic W-values:
##  Count: 38
##  Min: -1.11022302462516e-16
##  Max: 1
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

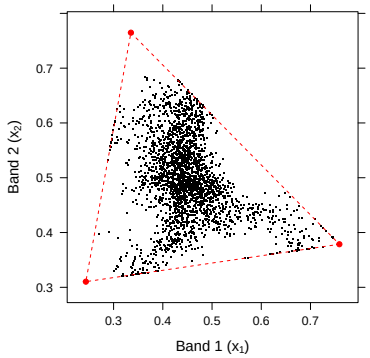



```
## [1] "Plotting: E3"
```

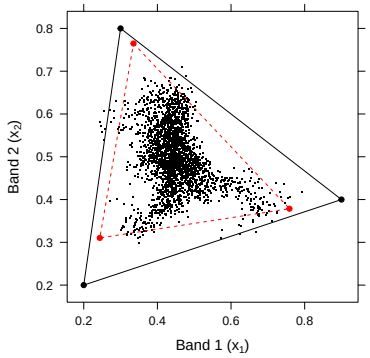



```
## [1] "SAM: 0.337292136652977"
```

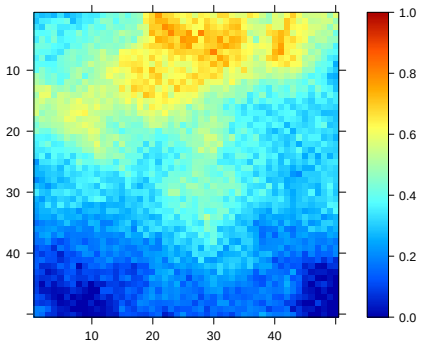
```
analysis(predlbl = 'ICE-opt')
```



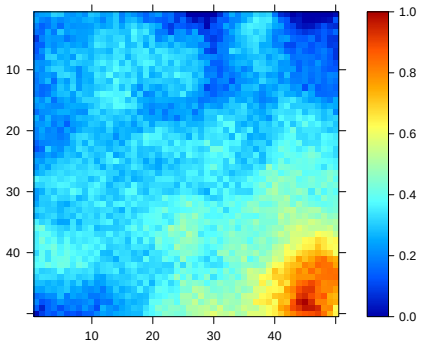

```
## [1] " color: red"
```



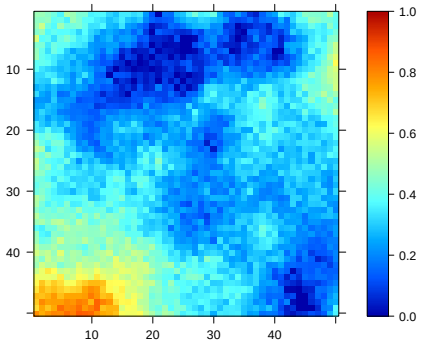

```
## [1] " color: red"
## Problematic W-values:
##  Count: 3
##  Min: -5.55111512312578e-17
##  Max: -2.77555756156289e-17
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```



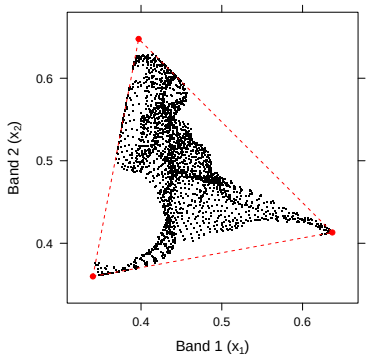

```
## [1] "Plotting: E3"
```



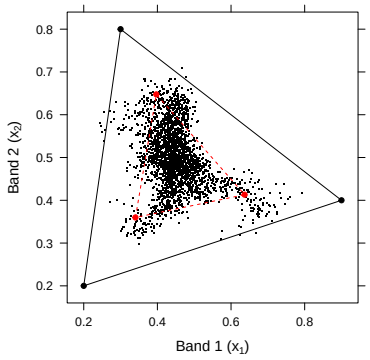

```
## [1] "SAM: 0.218662965406486"
```

17.2.7.3 ICE with spatial regularization

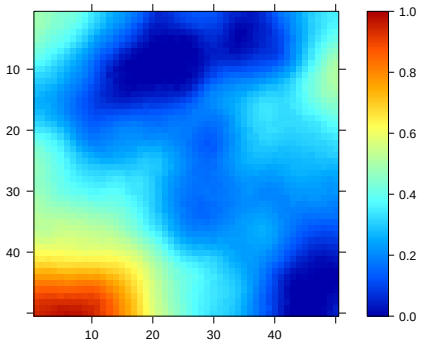
```
analysis(predlbl = 'ICE-0.5-S-0.02')
```



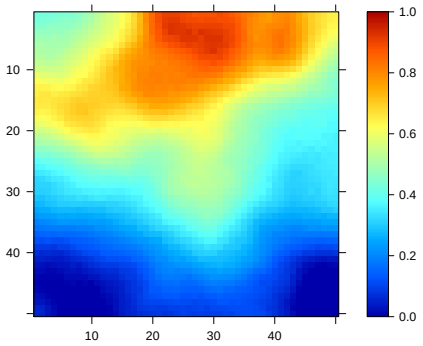

```
## [1] " color: red"
```



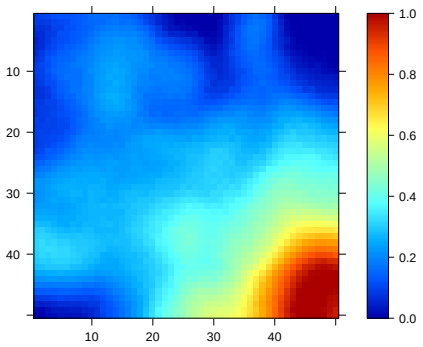

```
## [1] " color: red"
## Problematic W-values:
##  Count: 29
##  Min: -3.46944695195361e-18
##  Max: 1
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

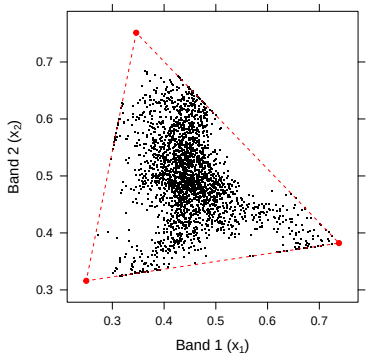



```
## [1] "Plotting: E3"
```

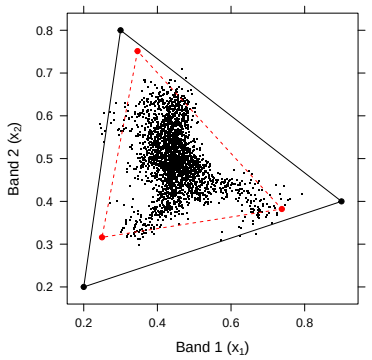



```
## [1] "SAM: 0.376194470048287"
```

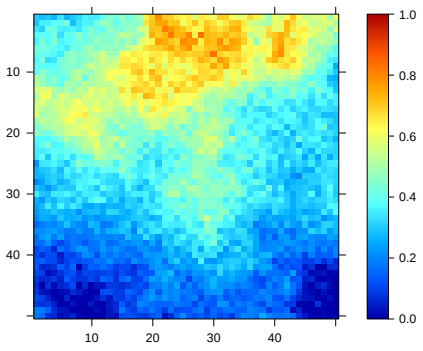
```
analysis(predlbl = 'ICE-S-opt')
```



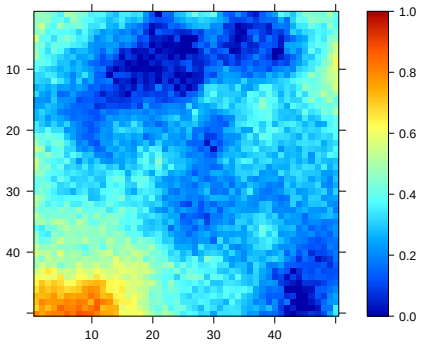

```
## [1] " color: red"
```



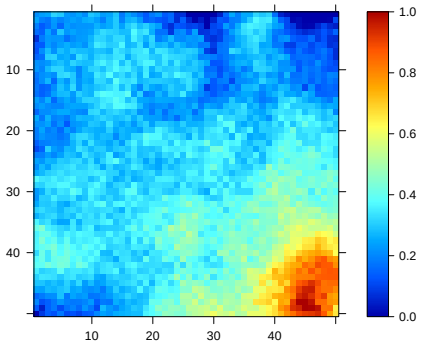

```
## [1] " color: red"
## Problematic W-values:
##  Count: 5
##  Min: -5.55111512312578e-17
##  Max: -3.46944695195361e-18
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```

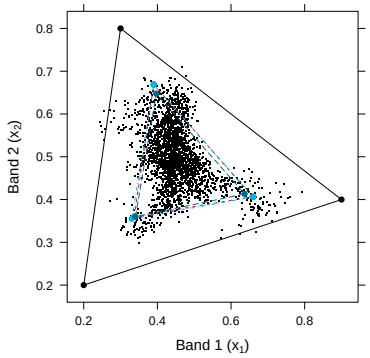



```
## [1] "SAM: 0.24939208995986"
```

17.2.8 Predicted endmembers

17.2.8.1 General comparison

```
hy$plotSimplex(IM50x50D2.Gauss[[]], E = IM50x50D2.Gauss@E,
               Ehat = list('ICE' = IM50x50D2.Gauss@pred$`ICE`$E,
                           'ICE-0.02-S-0.5' = IM50x50D2.Gauss@pred$`ICE-0.02-S-0.5`$E,
                           'ICE-0.1-S-0.1' = IM50x50D2.Gauss@pred$`ICE-0.1-S-0.1`$E,
                           'ICE-0.5-S-0.02' = IM50x50D2.Gauss@pred$`ICE-0.5-S-0.02`$E
                           ),
               EhatCol = c('red',
                           'deepskyblue',
                           'deepskyblue2',
                           'deepskyblue4'),
               printAs = 'IM50x50D2.Gauss-pred-spatial-reg-same-vol')
```

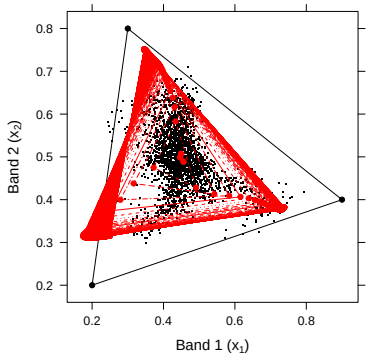



```
## [1] "ICE color: red"
## [1] "ICE-0.02-S-0.5 color: deepskyblue"
## [1] "ICE-0.1-S-0.1 color: deepskyblue2"
## [1] "ICE-0.5-S-0.02 color: deepskyblue4"
```

```
## pdf
##   2
```

17.2.8.2 ICE-S minimization

```
hy$plotSimplex(X = IM50x50D2.Gauss[[]], E = IM50x50D2.Gauss@E,
               Ehat = IM50x50D2.Gauss@pred$`ICE-S-opt`@iterates[['E']],
               EhatCol = rep('red', 10000), legendLog = FALSE)
```



18 Real data - Cuprite scene

18.1 Radiance data

Load image.

```
memory.limit(size = 20000)
```

```
## [1] 20000
```

```
IMCuRa <- read.ENVI(file = "../data/cuprite_radiance/f080611t01p00r06rdn_c_sc01_ort_img.rfl")
```

```
## .read.ENVI.header: Guessing header file name ../data/cuprite_radiance/f080611t01p00r06rdn_c_sc01_
## .read.ENVI.header: Guessing header file name ../data/cuprite_radiance/f080611t01p00r06rdn_c_sc01_
```

```
saveRDS(IMCuRa, file = './export/IMCuRa.rds')
```

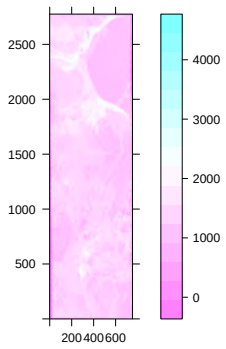
```
IMCuRa <- readRDS(file = './export/IMCuRa.rds')
```

18.1.1 Visualize

```
c("X-range"=range(IMCuRa$x), "Y-range"=range(IMCuRa$y))
```

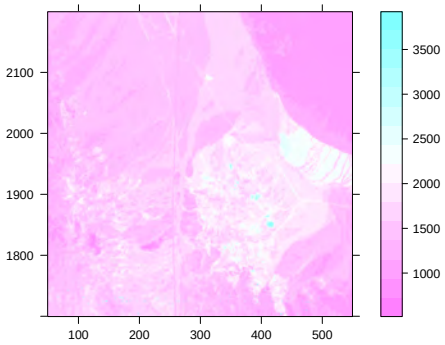
```
## X-range1 X-range2 Y-range1 Y-range2
##        0      753        0     2775
```

```
plotmap(IMCuRa)
```



Slice the needed area.

```
IMCuRa <- IMCuRa[IMCuRa$y>=1700 & IMCuRa$y<2200 & IMCuRa$x>=50 & IMCuRa$x<550]
plotmap(IMCuRa)
```



18.1.2 Generate a false color image.

First lets see how the datacube looks like with RGB bands.

```
hy$dcube(IMCuRa[,,hy$rgbBands])[1:2,1:2,]
```

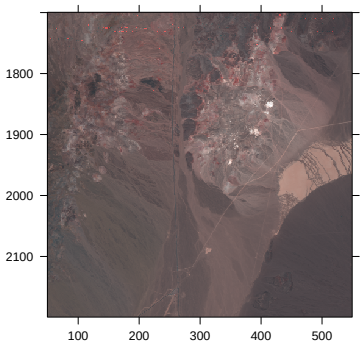
```
## , , z = 753.1287
##
##       x
## y       50   51
##   1700 2845 2725
##   1701 2762 2771
##
## , , z = 530.818
##
```

```
##       x
## y        50   51
##   1700 3202 3050
##   1701 3049 3074
##
## , , z = 482.1898
##
##       x
## y        50   51
##   1700 3325 3197
##   1701 3186 3211
```

Next we do the false color plotting.

```
hy$plotfc(IMCuRa[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```




```
dev.copy2pdf( file = './../fig/radiance-cuprite.pdf' )
```

```
## pdf
##   2
```

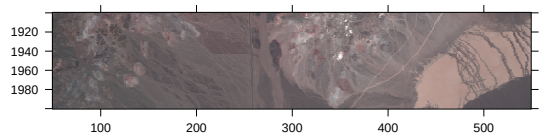
We also need a function for convenience slicing of hyperspectral image.

```
hy$subset <- function(hyimg, bottomy = min(hyimg$y), topy = max(hyimg$y),
                      bottomx = min(hyimg$x), topx = max(hyimg$x)){
  hyimg[hyimg$y >= bottomy & hyimg$y <= topy
        & hyimg$x >= bottomx & hyimg$x <= topx,,]
}
```

18.1.3 Find the sample image as in [1]

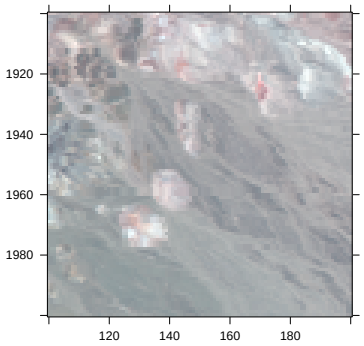
```
hy$plotfc(hy$subset(IMCuRa, 1900, 2000)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



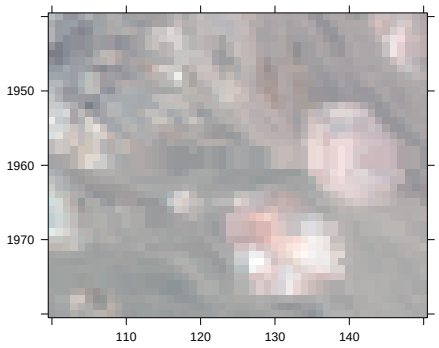

```
hy$plotfc(hy$subset(IMCuRa, 1900, 2000, 100, 200)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



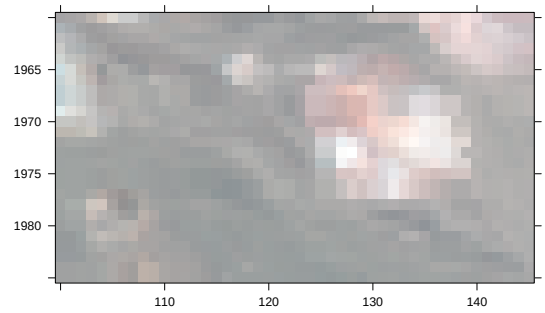

```
hy$plotfc(hy$subset(IMCuRa, 1940, 1980, 100, 150)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



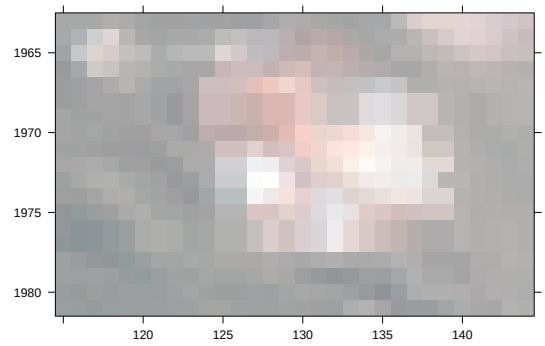

```
hy$plotfc(hy$subset(IMCuRa, 1960, 1985, 100, 145)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```




```
hy$plotfc(hy$subset(IMCuRa, 1963, 1981, 115, 144)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```

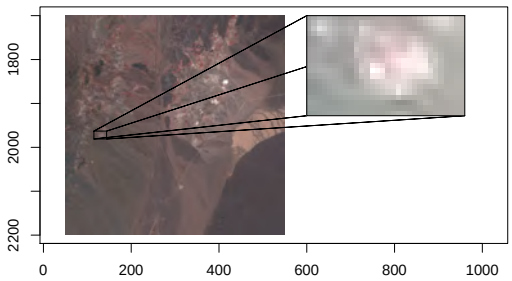



```
IMCuRaD <- hy$subset(IMCuRa, 1963, 1981, 115, 144)
IMCuRaD <- as(IMCuRaD, 'hyperSpecExt')
IMCuRaD[,,hy$rgbBands]
```

```
## hyperSpec object
##    570 spectra
##    4 data columns
##    3 data points / spectrum
## wavelength:  [numeric] 753 531 482
## data:  (570 rows x 4 columns)
##    1. x:  [integer] 115 115 ... 144
##    2. y:  [integer] 1963 1964 ... 1981
##    3. spc:  [matrix3] 2865 2824 ... 3414
##    4. filename: filename [character] ../data/cuprite_radiance/f080611t01p00r06rdn_c_sc01_ort_img.
```

```
hy$plotZoomed <- function(hy3D, zoom, zleftx, zrightx, ztopy, zbottomy, zoffsetx){
  ymin <- min(hy3D$y)
  ymax <- max(hy3D$y)
  xmin <- min(hy3D$x)
  xmax <- max(hy3D$x)

  # canvas
  plot(c(xmin, 1000), c(ymin, ymax), type = "n", xlab = "", ylab = "", asp = 1, ylim = c(ymax, ymin)

  # rasters
  hy3DZoomed <- hy$subset(hy3D, zbottomy, ztopy, zleftx, zrightx)
  im <- hy$fcRaster(hy$dcube(hy3D))
  imSlice <- hy$fcRaster(hy$dcube(hy3DZoomed))
```

```
  # main foto
  rasterImage(im, xmin, ymax, xmax, ymin, interpolate = FALSE)

  # kader rond specimen in echte foto
  polygon(matrix(c(zleftx,zbottomy,zleftx,ztopy,zrightx,ztopy,zrightx,zbottomy), ncol = 2, byrow = T

  # zoom in indicators
  polygon(matrix(c(zleftx,zbottomy,zoffsetx,ymin), ncol = 2, byrow = TRUE))
  polygon(matrix(c(zleftx,ztopy,zoffsetx,ymin+nrow(imSlice)*zoom), ncol = 2, byrow = TRUE))
  polygon(matrix(c(zrightx,ztopy,zoffsetx+ncol(imSlice)*zoom,ymin+nrow(imSlice)*zoom), ncol = 2, byr
  polygon(matrix(c(zrightx,zbottomy,zoffsetx+ncol(imSlice)*zoom,ymin), ncol = 2, byrow = TRUE))

  # zoom foto
  rasterImage(imSlice, zoffsetx, ymin+nrow(imSlice)*zoom, zoffsetx+ncol(imSlice)*zoom, ymin, interpo

  # kader rond zoom foto
  polygon(matrix(c(zoffsetx,ymin,zoffsetx,ymin+nrow(imSlice)*zoom,
                   zoffsetx+ncol(imSlice)*zoom,ymin+nrow(imSlice)*zoom,
                   zoffsetx+ncol(imSlice)*zoom,ymin), ncol = 2, byrow = TRUE))
}
hy$plotZoomed(IMCuRa[,,hy$rgbBands], 12, 115, 144, 1981, 1963, 600)
```



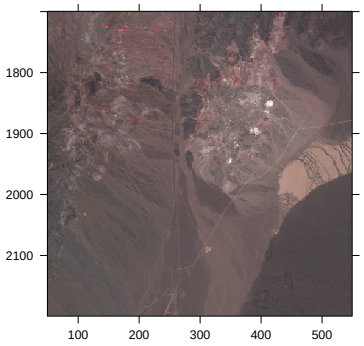
Lets see the spectrum of the top left pixel.

```
IMCuRaD[1,,]
```

```
## hyperSpec object
##    1 spectra
##    4 data columns
##    224 data points / spectrum
## wavelength:  [numeric] 366 376 ... 2497
## data:  (1 rows x 4 columns)
##    1. x:  [integer] 115
##    2. y:  [integer] 1963
##    3. spc:  [matrix224] 1284 1458 ... 125
##    4. filename: filename [character] ../data/cuprite_radiance/f080611t01p00r06rdn_c_sc01_ort_img.
```

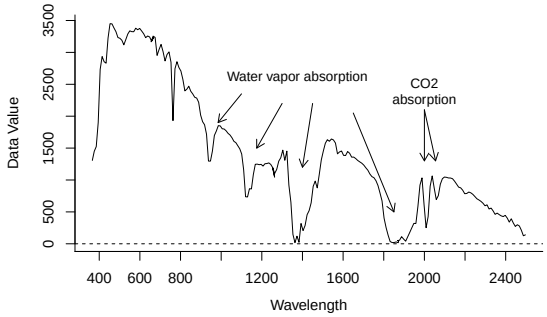
```
hy$plotfc(IMCuRa[,,hy$rgbBands], printAs = 'radiance-cuprite')
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```




```
## pdf
##   2
```

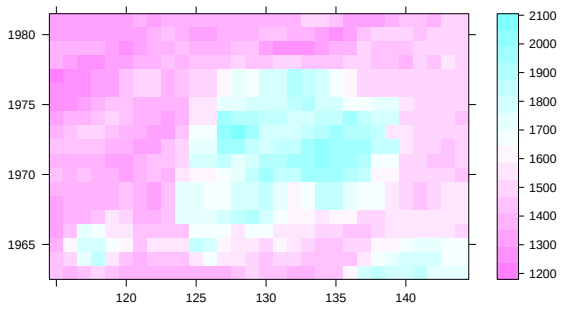
```
plotspc(IMCuRa[1,,], title.args = list(xlab = "Wavelength", ylab = "Data Value"))
text(1375, 2600, labels = 'Water vapor absorption', cex= 0.9)
arrows(1100, 2350, 985, 1900, length=0.1)
arrows(1300, 2200, 1175, 1500, length=0.1)
arrows(1450, 2200, 1400, 1200, length=0.1)
arrows(1650, 2050, 1850, 500, length=0.1)
text(2000, 2400, labels = 'CO2\nabsorption', cex= 0.9)
arrows(2000, 2100, 2000, 1300, length=0.1)
arrows(2000, 2100, 2055, 1300, length=0.1)
```



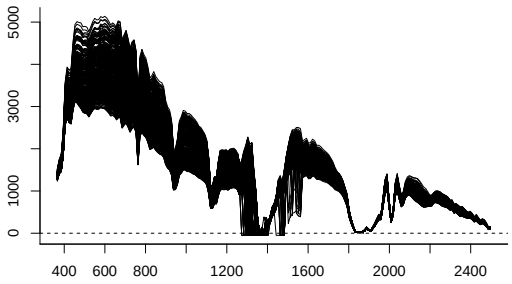

```
dev.copy2pdf( file = './../fig/radiance-cuprite-top-left-pixel-spectrum-annotated.pdf' )
```

```
## pdf
##   2
```

```
plotmap(IMCuRaD)
```



```
plot(IMCuRaD, spc.nmax = 1000)
```



18.2 Reflectance data

```
IMCuRe <- read.ENVI(file = "../data/cuprite_reflectance/f970619t01p02_r02_sc04.a.rfl")
```

```
## .read.ENVI.header: Guessing header file name ../data/cuprite_reflectance/f970619t01p02_r02_sc04.a
```

```
saveRDS(IMCuRe, file = './export/IMCuRe.rds')
```

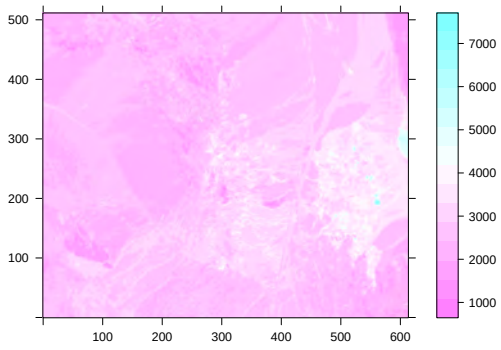
```
IMCuRe <- readRDS(file = './export/IMCuRe.rds')
```

18.2.1 Visualize

```
c("X-range"=range(IMCuRe$x), "Y-range"=range(IMCuRe$y))
```

```
## X-range1 X-range2 Y-range1 Y-range2
##        0      613        0      511
```

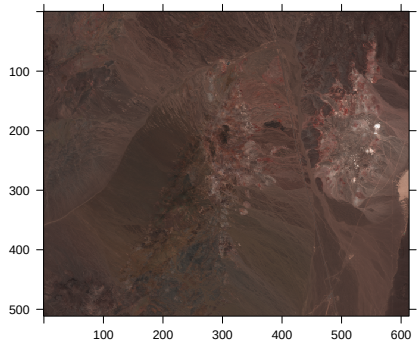
```
plotmap(IMCuRe)
```



18.2.2 Find the sample image as in [1]

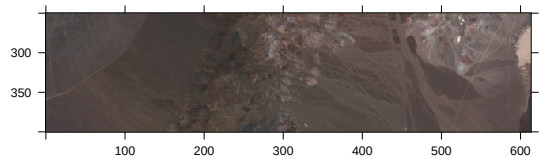
```
hy$plotfc(IMCuRe[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



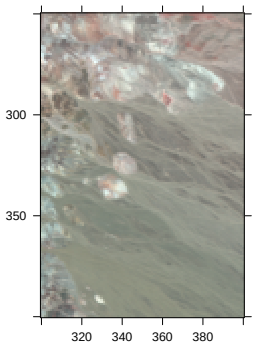

```
hy$plotfc(hy$subset(IMCuRe, 250, 400)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



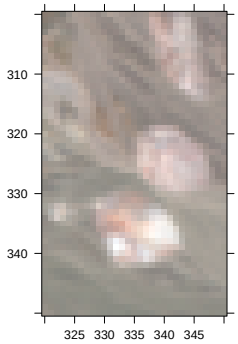

```
hy$plotfc(hy$subset(IMCuRe, 250, 400, 300, 400)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



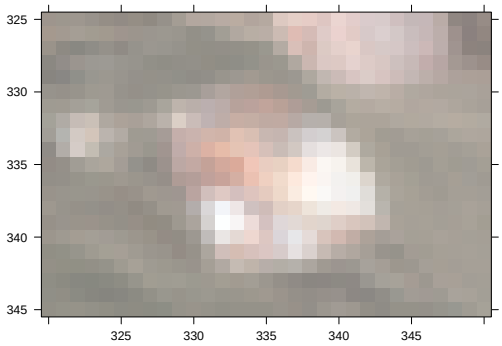

```
hy$plotfc(hy$subset(IMCuRe, 300, 350, 320, 350)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



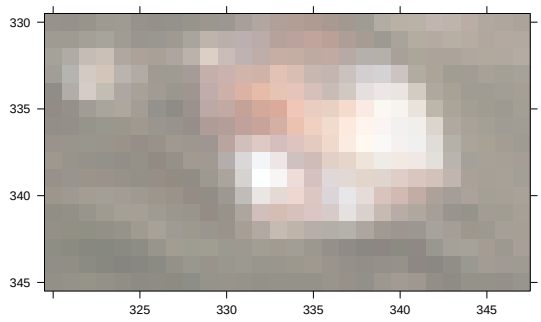

```
hy$plotfc(hy$subset(IMCuRe, 325, 345, 320, 350)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```



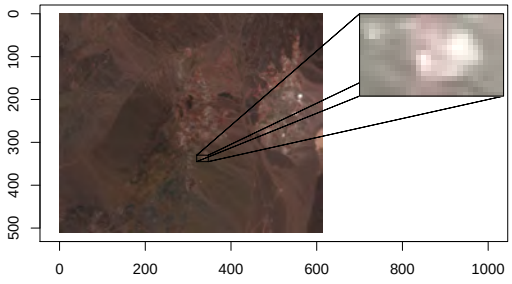

```
hy$plotfc(hy$subset(IMCuRe, 330, 345, 320, 347)[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```




```
IMCuReD <- hy$subset(IMCuRe, 330, 345, 320, 347)
IMCuReD <- as(IMCuReD, 'hyperSpecExt')
```

```
hy$plotZoomed(IMCuRe[,,hy$rgbBands], 12, 320, 347, 345, 330, 700)
```



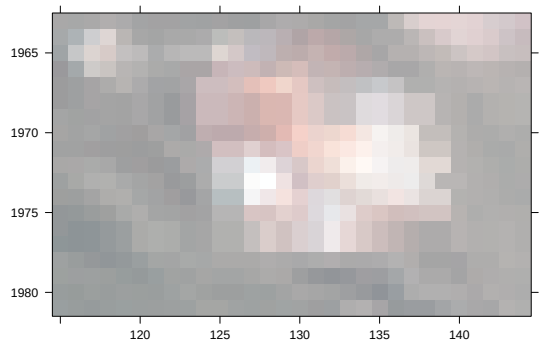

```
dev.copy2pdf( file = './../fig/reflectance-cuprite-aoi-map.pdf' )
```

```
## pdf
##   2
```

18.2.3 Comparison radiance and reflectance images

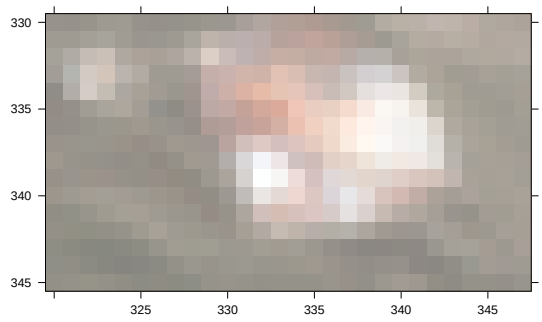
```
hy$plotfc(IMCuRaD[,,hy$rgbBands])
```

```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```

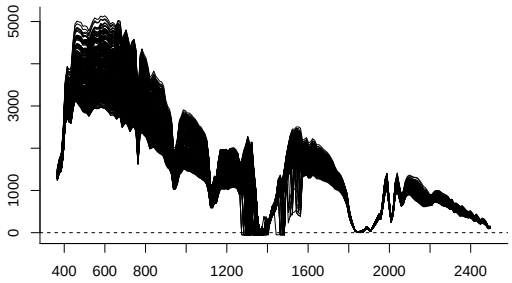



```
hy$plotfc(IMCuReD[,,hy$rgbBands])
```

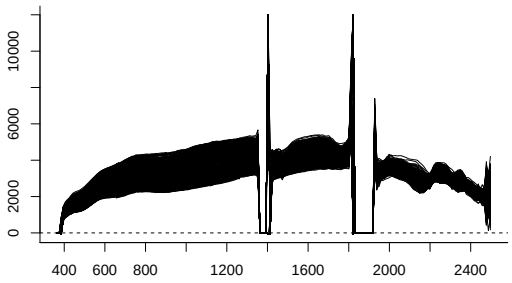
```
## [1] "Wavelengths for falsecolor image: 753.1287, 530.818, 482.1898"
```




```
plot(IMCuRaD, spc.nmax = 1000)
```




```
plot(IMCuReD, spc.nmax = 1000)
```



18.3 Data cleanup

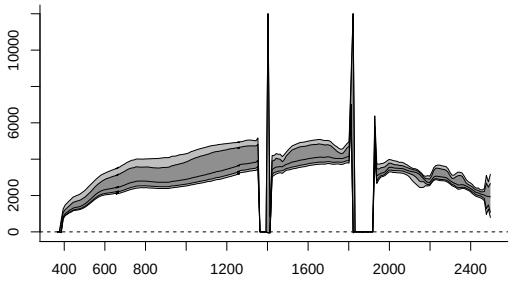
18.3.1 Invalid data

Remove bands with invalida data (-50 as values for example)

```
IMCuReD
```

```
## hyperSpec object
##    448 spectra
##    4 data columns
##    224 data points / spectrum
## wavelength:  [numeric] 366 376 ... 2497
## data:  (448 rows x 4 columns)
##    1. x:  [integer] 320 320 ... 347
##    2. y:  [integer] 330 331 ... 345
##    3. spc:  [matrix224] 0 0 ... 942
##    4. filename: filename [character] ../data/cuprite_reflectance/f970619t01p02_r02_sc04.a.rfl ../
```

```
plot(IMCuReD, "spcprctl5")
```




```
dev.copy2pdf( file = './../fig/reflectance-spcprctl5-unclean.pdf' )
```

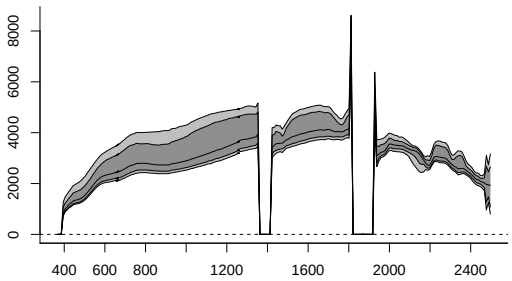
```
## pdf
##   2
```

```
wlWithNegativeValues <- wl(IMCuReD)[apply(IMCuReD[[]] < 0, 2, any)]
wlWithNegativeValues
```

```
## [1]  385 1403 1413 1821
```

```
if (length(wlWithNegativeValues) != 0){
  IMCuReD[[,,wlWithNegativeValues]] <- 0
  #IMCuReD <- IMCuReD[,,-wl2i(IMCuReD, wlWithNegativeValues),wl.index = TRUE]
  print(IMCuReD)
  plot(IMCuReD, "spcprctl5")
}
```

```
## hyperSpec object
##    448 spectra
##    4 data columns
##    224 data points / spectrum
## wavelength:  [numeric] 366 376 ... 2497
## data:  (448 rows x 4 columns)
##    1. x:  [integer] 320 320 ... 347
##    2. y:  [integer] 330 331 ... 345
##    3. spc:  [matrix224] 0 0 ... 942
##    4. filename: filename [character] ../data/cuprite_reflectance/f970619t01p02_r02_sc04.a.rfl ../
```

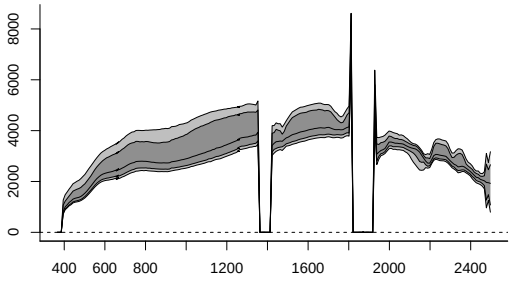



```
rm(wlWithNegativeValues)
```

18.3.2 Exessive large values

Few spikes/glitches around 1811 nm and 1928 nm that were present in reflectance data have been set to 0.

```
plot(IMCuReD, "spcprctl5")
```

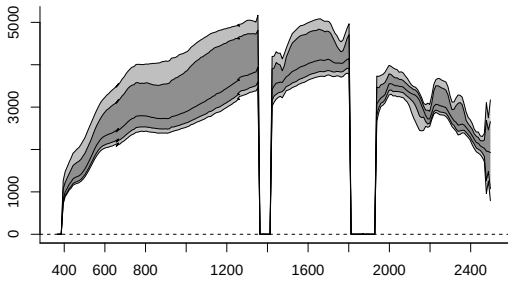



```
IMCuReD[[1]]
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    0    0  758  918  987 1095 1145 1223  1252  1258  1282  1312
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22] [,23] [,24]
## [1,]  1368  1428  1503  1581  1668  1772  1847  1937  1999  2058  2097
##      [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35]
## [1,]  2114  2142  2150  2168  2188  2202  2227  2255  2194  2226  2254
##      [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44] [,45] [,46]
## [1,]  2290  2325  2368  2398  2458  2483  2499  2538  2555  2558  2560
##      [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57]
## [1,]  2559  2558  2566  2571  2541  2550  2513  2534  2515  2530  2525
##      [,58] [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66] [,67] [,68]
## [1,]  2523  2505  2514  2554  2553  2599  2585  2608  2623  2646  2665
##      [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77] [,78] [,79]
## [1,]  2666  2695  2725  2744  2762  2792  2815  2832  2858  2878  2921
##      [,80] [,81] [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90]
## [1,]  2922  2938  2982  3021  3027  3068  3081  3102  3145  3174  3198
##      [,91] [,92] [,93] [,94] [,95] [,96] [,97] [,98] [,99] [,100] [,101]
## [1,]  3236  3260  3286  3323  3363  3380  3386  3416  3473   3464   3472
##      [,102] [,103] [,104] [,105] [,106] [,107] [,108] [,109] [,110] [,111]
## [1,]   3511   3519   3563   3563   3520   3597      0      0      0      0
##      [,112] [,113] [,114] [,115] [,116] [,117] [,118] [,119] [,120] [,121]
## [1,]      0      0   3123   3214   3330   3348   3432   3420   3472   3542
##      [,122] [,123] [,124] [,125] [,126] [,127] [,128] [,129] [,130] [,131]
## [1,]   3541   3582   3594   3628   3665   3692   3699   3727   3748   3750
##      [,132] [,133] [,134] [,135] [,136] [,137] [,138] [,139] [,140] [,141]
## [1,]   3784   3810   3809   3823   3842   3863   3862   3862   3851   3869
##      [,142] [,143] [,144] [,145] [,146] [,147] [,148] [,149] [,150] [,151]
## [1,]   3870   3860   3838   3844   3858   3859   3869   3903   3904   3954
##      [,152] [,153] [,154] [,155] [,156] [,157] [,158] [,159] [,160] [,161]
## [1,]   3898   6260      0      0      0      0      0      0      0      0
##      [,162] [,163] [,164] [,165] [,166] [,167] [,168] [,169] [,170] [,171]
## [1,]      0      0      0      0      0   5239   3018   3199   3108   3146
##      [,172] [,173] [,174] [,175] [,176] [,177] [,178] [,179] [,180] [,181]
## [1,]   3175   3274   3398   3493   3405   3402   3421   3449   3406   3396
##      [,182] [,183] [,184] [,185] [,186] [,187] [,188] [,189] [,190] [,191]
## [1,]   3339   3325   3297   3280   3291   3303   3254   3190   3076   2912
##      [,192] [,193] [,194] [,195] [,196] [,197] [,198] [,199] [,200] [,201]
## [1,]   2812   2681   2668   2816   2922   2978   2947   2964   2890   2916
##      [,202] [,203] [,204] [,205] [,206] [,207] [,208] [,209] [,210] [,211]
## [1,]   2865   2814   2738   2789   2626   2577   2429   2474   2468   2506
##      [,212] [,213] [,214] [,215] [,216] [,217] [,218] [,219] [,220] [,221]
```

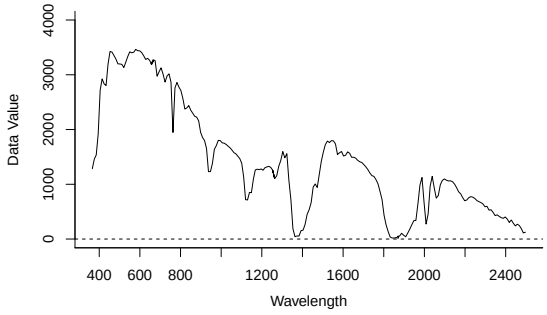
```
## [1,]   2574   2480   2357   2346   2250   2181   2182   2034   1903   2182
##       [,222] [,223] [,224]
## [1,]   1727   2626   1821
```

```
## spectra 1811 nm and 1928 nm are outliers
IMCuReD[[,,c(1811, 1928)]] <- 0
plot(IMCuReD, "spcprctl5")
```



Lets see the spectrum of the top left pixel.

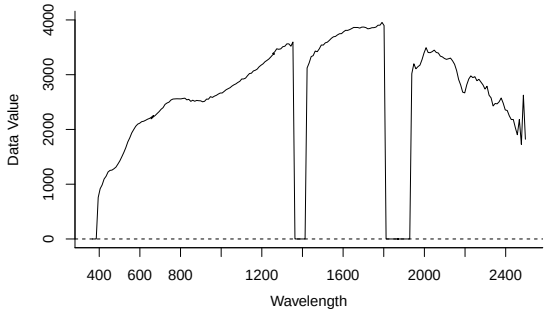
```
plotspc(IMCuRaD[1,,], title.args = list(xlab = "Wavelength", ylab = "Data Value"),
        plot.args = list('ylim' = c(0,4000)))
```




```
dev.copy2pdf( file = './../fig/radiance-cuprite-top-left-pixel-spectrum.pdf' )
```

```
## pdf
##   2
```

```
plotspc(IMCuReD[1,,], title.args = list(xlab = "Wavelength", ylab = "Data Value"),
        plot.args = list('ylim' = c(0,4000)))
```



```
dev.copy2pdf( file = './../fig/reflectance-cuprite-top-left-pixel-spectrum.pdf' )
```

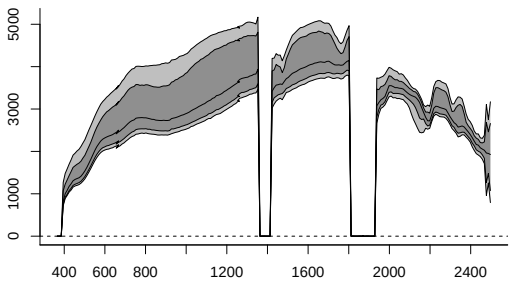
```
## pdf
##   2
```

18.3.3 Remove wavelengths with 0 values

```
IMCuReD
```

```
## hyperSpec object
##    448 spectra
##    4 data columns
##    224 data points / spectrum
## wavelength:  [numeric] 366 376 ... 2497
## data:  (448 rows x 4 columns)
##    1. x:  [integer] 320 320 ... 347
##    2. y:  [integer] 330 331 ... 345
##    3. spc:  [matrix224] 0 0 ... 942
##    4. filename: filename [character] ../data/cuprite_reflectance/f970619t01p02_r02_sc04.a.rfl ../
```

```
plot(IMCuReD, "spcprctl5")
```

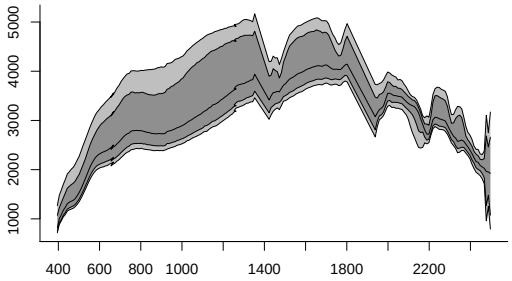



```
wlWithZeroValues <- wl(IMCuReD)[apply(IMCuReD[[]] == 0, 2, any)]
wlWithZeroValues
```

```
##  [1]  366  376  385 1363 1373 1383 1393 1403 1413 1811 1821 1831 1841 1851
## [15] 1861 1871 1873 1868 1878 1888 1898 1908 1918 1928
```

```
if (length(wlWithZeroValues) != 0){
  IMCuReD <- IMCuReD[,,-wl2i(IMCuReD, wlWithZeroValues),wl.index = TRUE]
  print(IMCuReD)
  plot(IMCuReD, "spcprctl5")
}
```

```
## hyperSpec object
##    448 spectra
##    4 data columns
##    200 data points / spectrum
## wavelength:  [numeric] 395 405 ... 2497
## data:  (448 rows x 4 columns)
##    1. x:  [integer] 320 320 ... 347
##    2. y:  [integer] 330 331 ... 345
##    3. spc:  [matrix200] 758 821 ... 942
##    4. filename: filename [character] ../data/cuprite_reflectance/f970619t01p02_r02_sc04.a.rfl ../
```



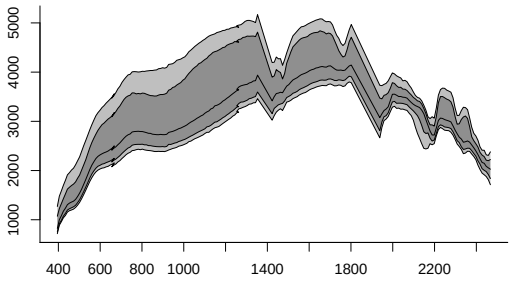

```
rm(wlWithZeroValues)
# 24 wavelengths removed: remaining 200
IMCuReD
```

```
## hyperSpec object
##    448 spectra
##    4 data columns
##    200 data points / spectrum
## wavelength:  [numeric] 395 405 ... 2497
## data:  (448 rows x 4 columns)
##    1. x:  [integer] 320 320 ... 347
##    2. y:  [integer] 330 331 ... 345
##    3. spc:  [matrix200] 758 821 ... 942
##    4. filename: filename [character] ../data/cuprite_reflectance/f970619t01p02_r02_sc04.a.rfl ../
```

18.3.4 Remove high noise bands

These pcitures show that there is high variance within the last 3 bands. Those bands are: 2477, 2487 and 2497. We remove them.

```
IMCuReD <- IMCuReD[,,c(min ~ 2467)]
plot(IMCuReD, "spcprctl5")
```




```
dev.copy2pdf( file = './../fig/reflectance-spcprctl5-clean.pdf' )
```

```
## pdf
##   2
```

[1] Dobigeon, N. e.a., Joint Bayesian Endmember Extraction and Linear Unmixing for Hyperspectral Imagery, 2009 [2] Clark, R.N. e.a., Imaging spectroscopy - Earth and planetary remote sensing with the USGS Tetracorder and expert systems, 2003

18.4 USGS endmembers

18.4.1 Loading USGS spectra

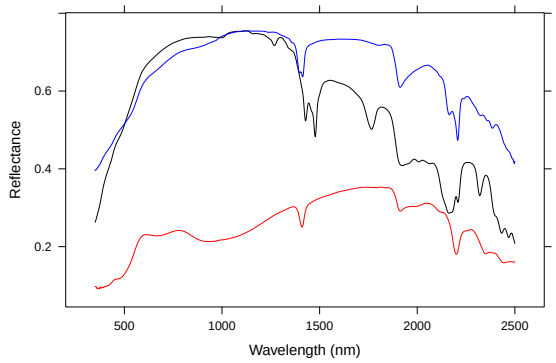
We take here the spectra from the Cuprite scene taken with a ASD Fieldspec spectrometer from the USGS library.

```
em$muscovite_medhi_Al <- em$read(filename = 'muscovite-medhi-Al_CU91-252d.26143', type = 'S')
em$alunite_hi_Temp <- em$read(filename = 'alunite_cu98-5c.24129', type = 'S')
em$kaolinite_others <- em$read(filename = 'kaolinite_wxl+other_cu91-200a.25557', type = 'S')
em$E_cuprite <- cbind('Alunite' = em$alunite_hi_Temp,
                      'Muscovite' = em$muscovite_medhi_Al,
                      'Kaolinite' = em$kaolinite_others)
nrow(em$E_cuprite)
```

```
## [1] 2151
```

```
em$plot(E = em$E_cuprite)
```

```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



18.4.1.1 Interpolate ASD Fieldspec spectrometer bands to AVIRIS

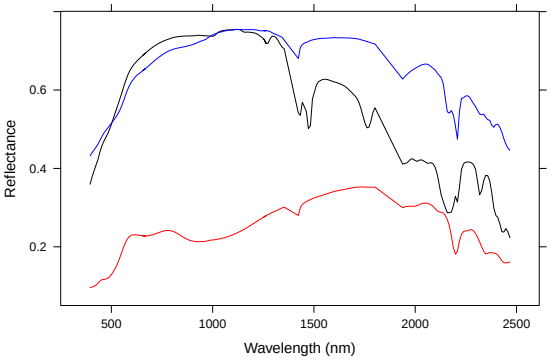
Since we are going from 2151 bands to 197 bands, there is quite some flattening out by linear interpolation.

```
em$E_cuprite_ip <- em$interpolate(E = em$E_cuprite,
                                  wavelengths = hyperSpec::wl(IMCuReD))
em$E_cuprite_ip[c(1:5,194:197),]
```

```
##          Alunite Muscovite Kaolinite
## 394.9355   0.360    0.0964     0.432
## 404.6129   0.378    0.0973     0.440
## 414.2946   0.394    0.0995     0.447
## 423.9808   0.408    0.1019     0.454
## 433.6713   0.423    0.1071     0.462
## 2437.49    0.238    0.1595     0.476
## 2447.42    0.246    0.1589     0.462
## 2457.348   0.238    0.1601     0.452
## 2467.273   0.224    0.1610     0.446
```

```
em$plot(em$E_cuprite_ip)
```

```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



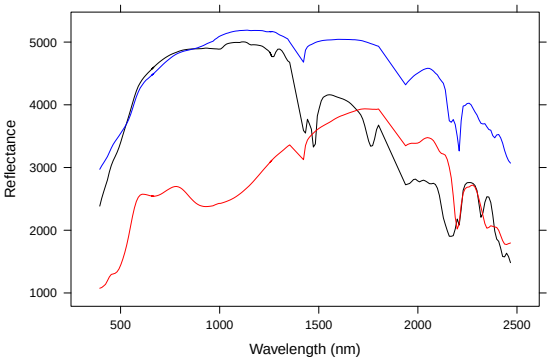
18.5 Abundances based on USGS endmembers

```
IMCuReD@usd[['rev_engineered']] <- alg$abun$solve(hyImg = IMCuReD, E = em$E_cuprite_ip,
                                                  tol = 0.9999999)
```

```
## ICE step: 0010 | Imp: -10413584.1114278  | Ratio: 0.97230424878699
## ICE step: 0020 | Imp: -6188.51793522663  | Ratio: 0.999595203624468
## ICE step: 0030 | Imp: -116.723526280839  | Ratio: 0.999967898366253
## ICE step: 0040 | Imp: -9.3304193369695  | Ratio: 0.999997441863931
## ICE step: 0050 | Imp: -0.737048008770216  | Ratio: 0.999999798995038
## ICE convergence after 53 steps
```

```
em$plot(E = em$E_cuprite_ip %*% diag(IMCuReD@usd[['rev_engineered']]),
        printAs = 'cuprite', suffix = 'original')
```

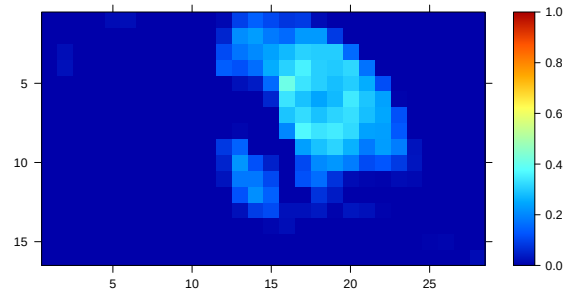
```
## V1: black
## V2: red
## V3: blue
```



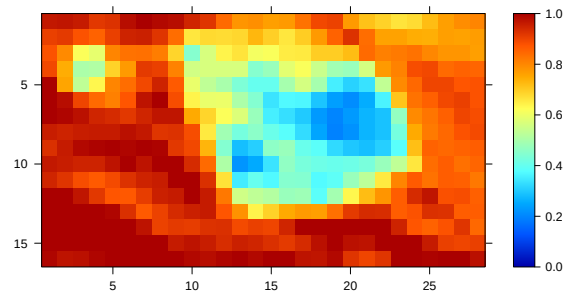

```
## pdf
##   2
```

```
hy$plotMapW(W = attr(IMCuReD@usd[['rev_engineered']], 'W'),
            printAs = 'cuprite', suffix = 'original', bw = FALSE,
            eLabelOverride = colnames(em$E_cuprite_ip))
```

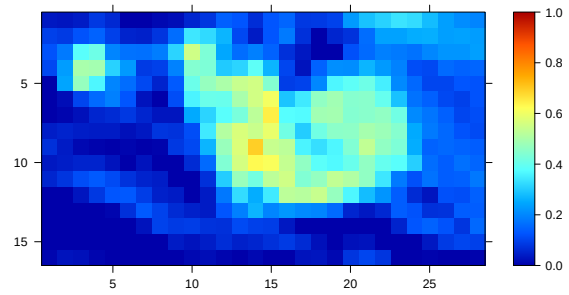
```
## Problematic W-values:
##  Count: 117
##  Min: -2.53718156892912e-15
##  Max: 1
## [1] "Plotting: Alunite"
```




```
## [1] "Plotting: Muscovite"
```




```
## [1] "Plotting: Kaolinite"
```



18.6 Analysis

18.6.1 Running the algorithms

```
IMCuReD@pred[['ICE']] <-
  alg$ice$ICE(hyImg = IMCuReD, M = 3)
```

```
## ICE convergence after 128 steps
```

```
IMCuReD@pred[['ICE-opt']] <-
  alg$ice$ICE(hyImg = IMCuReD, M = 3, mu = 0.0005)
```

```
## ICE convergence after 225 steps
```

```
IMCuReD@pred[['Bayes']] <-
  alg$bVol$sample(hyImg = IMCuReD, M = 3)
IMCuReD@pred[['Bayes-opt']] <-
  alg$bVol$sample(hyImg = IMCuReD, M = 3, gGamma = 0.001) # 1 is noisy, 0.1 is all the same
IMCuReD@pred[['ICE-0.5-S-0.02']] <-
  alg$iceS$ICE(hyImg = IMCuReD, M = 3, mu = 0.5, nu = 0.02)
```

```
## ICE convergence after 58 steps
```

```
IMCuReD@pred[['ICE-S-opt']] <-
  alg$iceS$ICE(hyImg = IMCuReD, M = 3, mu = 0.0015, nu = 0.55)
```

```
## ICE convergence after 534 steps
```

```
IMCuReD@pred[['Bayes-Tau']] <-
  alg$bVolTau$sample(hyImg = IMCuReD, M = 3)

saveRDS(IMCuReD@pred, file = 'export/IMCuReD@pred.rds')
```

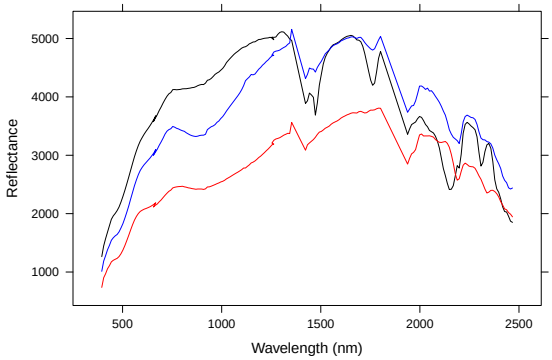
```
IMCuReD@pred <- readRDS(file = './export/IMCuReD@pred.rds')
```

18.6.2 Endmembers

18.6.2.1 Predicted endmembers

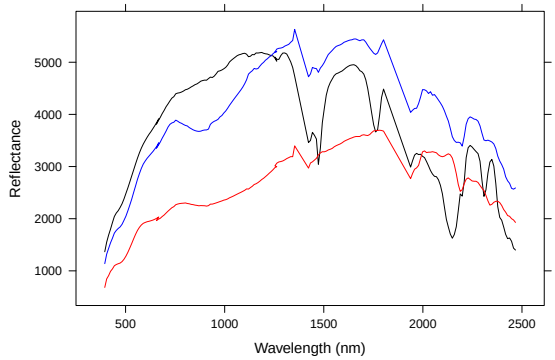
```
em$plot(E = IMCuReD@pred[['ICE']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```



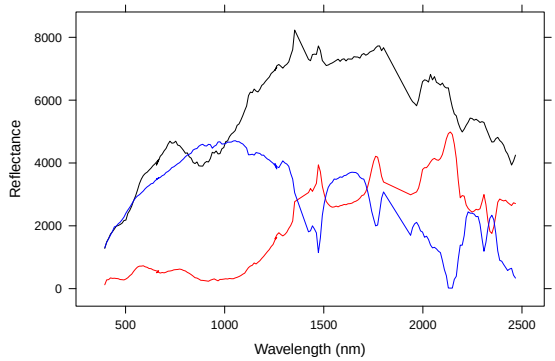

```
em$plot(E = IMCuReD@pred[['ICE-opt']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```



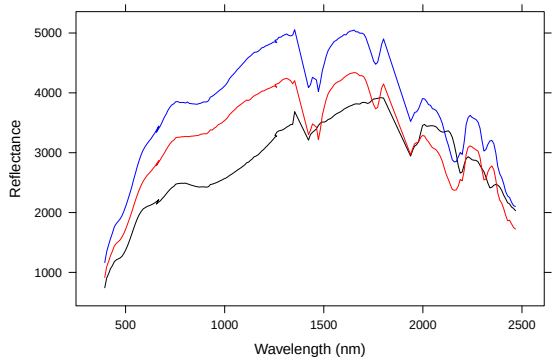

```
em$plot(E = IMCuReD@pred[['Bayes']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```



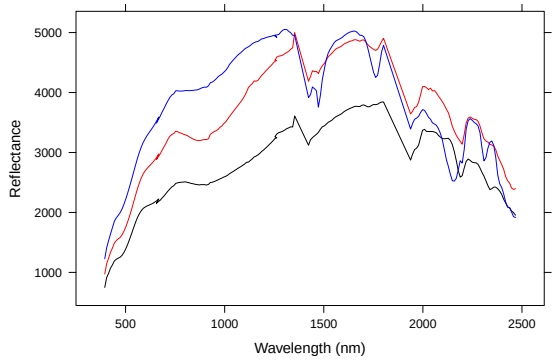

```
em$plot(E = IMCuReD@pred[['Bayes-opt']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```



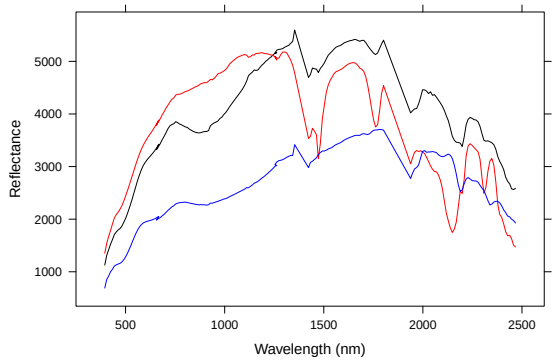

```
em$plot(E = IMCuReD@pred[['ICE-0.5-S-0.02']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```



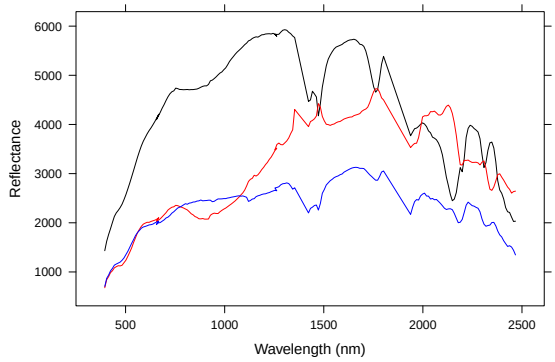

```
em$plot(E = IMCuReD@pred[['ICE-S-opt']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```




```
em$plot(E = IMCuReD@pred[['Bayes-Tau']]$E)
```

```
## V1: black
## V2: red
## V3: blue
```

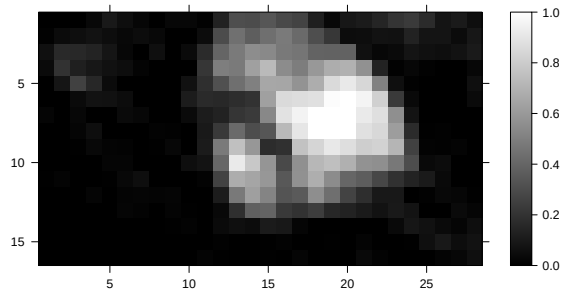


18.6.2.2 Predicted abundance maps

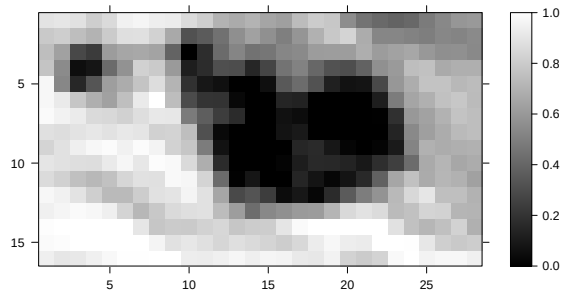
```
hy$plotMapW(IMCuReD@pred[['ICE']]$W)
```

```
## Problematic W-values:
##  Count: 107
##  Min: -1.2390906838574e-14
##  Max: 1.00000000000001
```

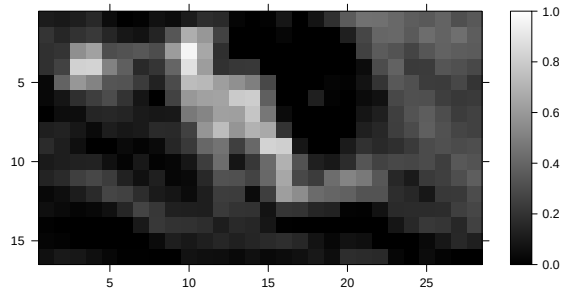
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

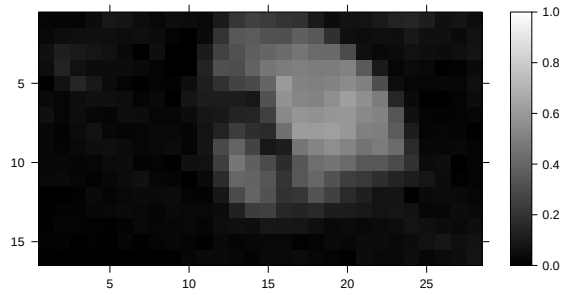



```
## [1] "Plotting: E3"
```

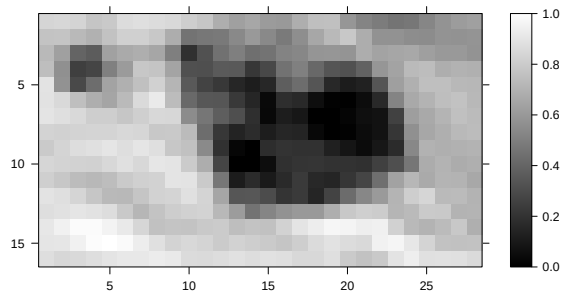



```
hy$plotMapW(IMCuReD@pred[['ICE-opt']]$W)
```

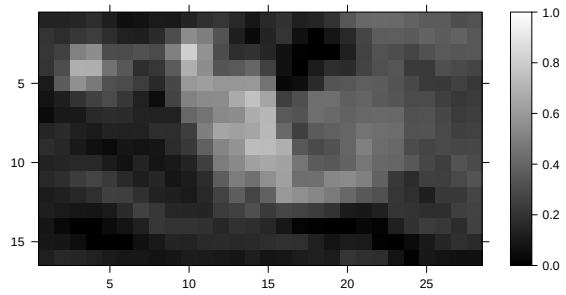
```
## Problematic W-values:
##  Count: 24
##  Min: -8.0646789753681e-15
##  Max: -5.91618127966114e-18
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

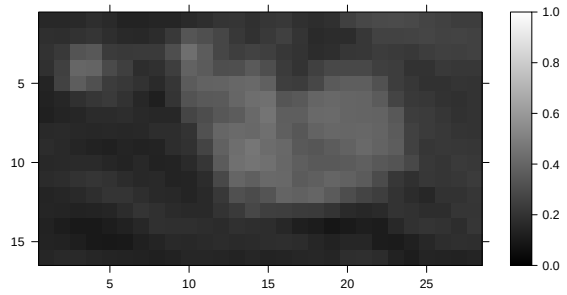



```
## [1] "Plotting: E3"
```

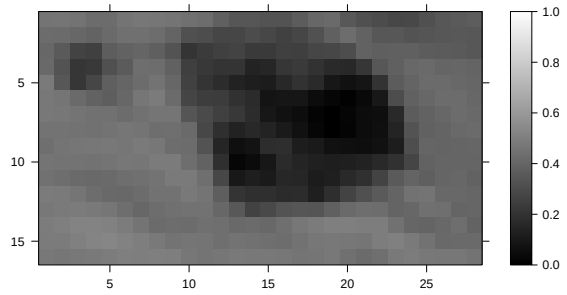



```
hy$plotMapW(IMCuReD@pred[['Bayes']]$W)
```

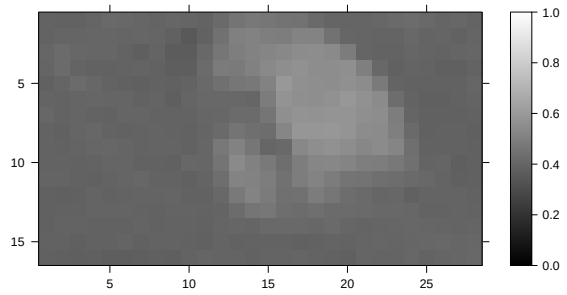
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

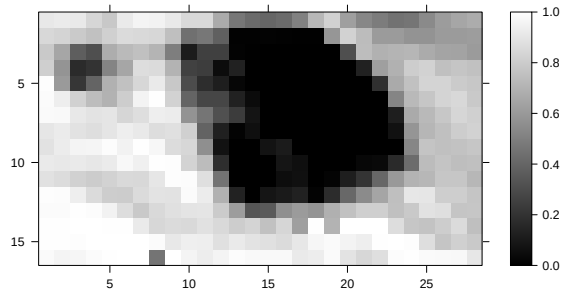



```
## [1] "Plotting: E3"
```

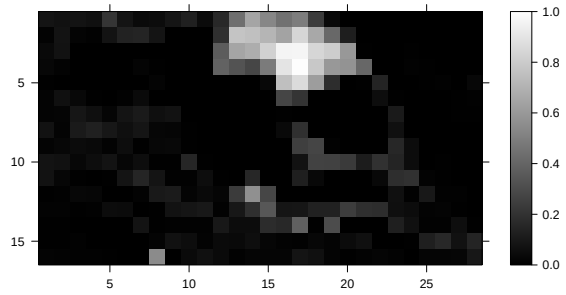



```
hy$plotMapW(IMCuReD@pred[['Bayes-opt']]$W)
```

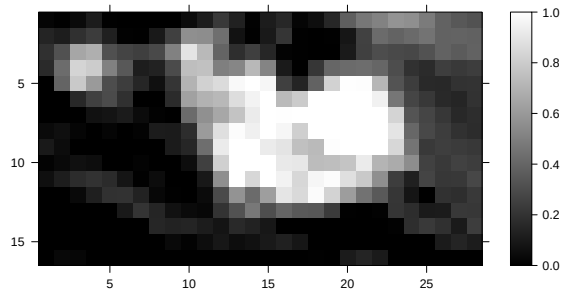
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

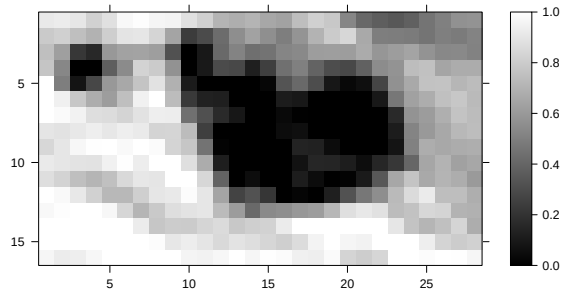



```
## [1] "Plotting: E3"
```

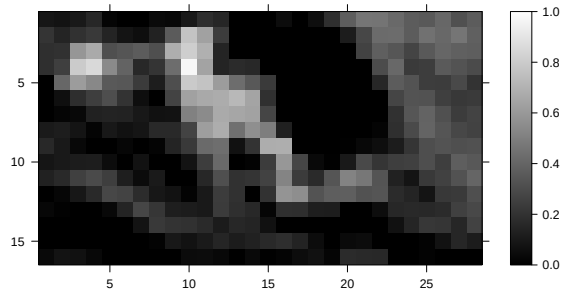



```
hy$plotMapW(IMCuReD@pred[['ICE-0.5-S-0.02']]$W)
```

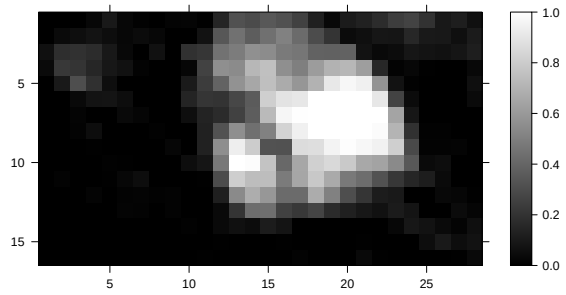
```
## Problematic W-values:
##  Count: 116
##  Min: -1.2620674231995e-14
##  Max: 1
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

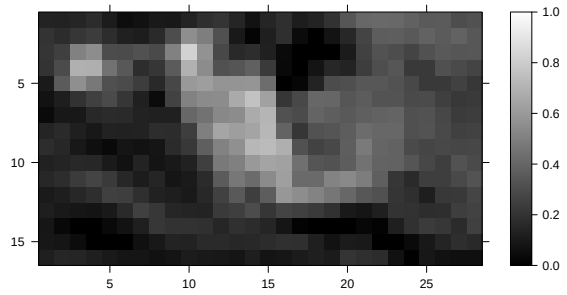



```
## [1] "Plotting: E3"
```

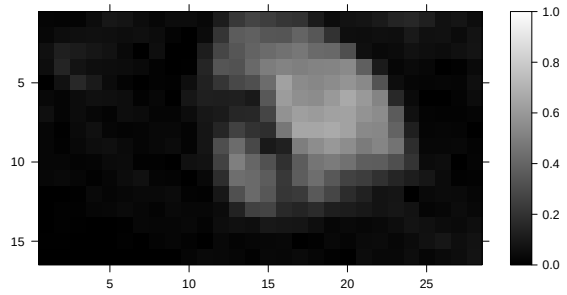



```
hy$plotMapW(IMCuReD@pred[['ICE-S-opt']]$W)
```

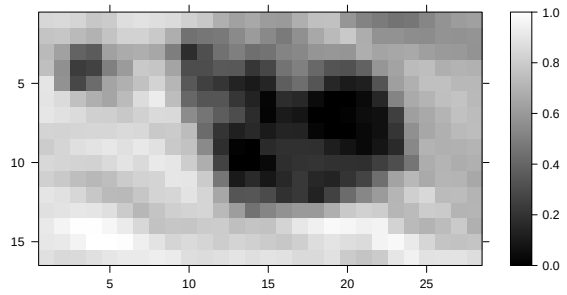
```
## Problematic W-values:
##  Count: 26
##  Min: -4.92951727260689e-15
##  Max: -4.34054256622271e-19
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```

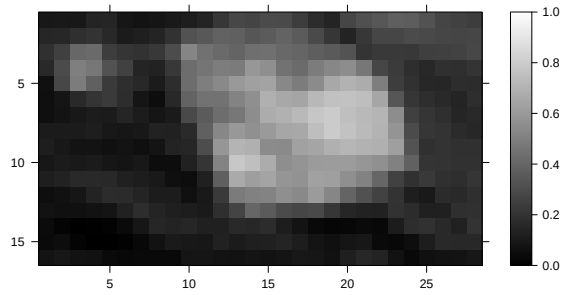



```
## [1] "Plotting: E3"
```

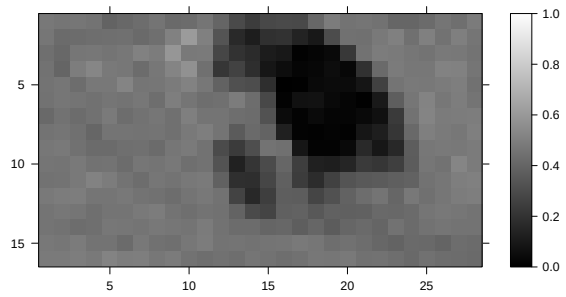



```
hy$plotMapW(IMCuReD@pred[['Bayes-Tau']]$W)
```

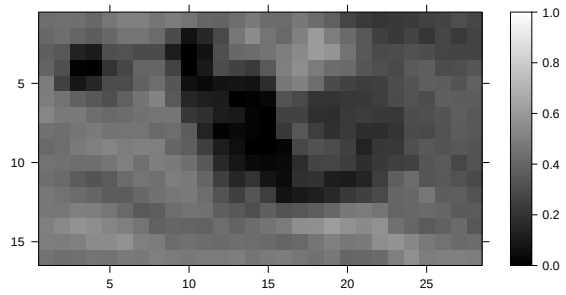
```
## [1] "Plotting: E1"
```




```
## [1] "Plotting: E2"
```




```
## [1] "Plotting: E3"
```



18.6.2.3 Prediction exports

18.6.2.3.1 ICE

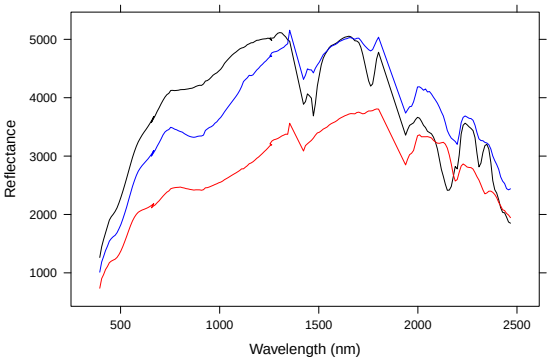
```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['ICE']]$E, linear = TRUE)
```

```
## Alunite
## MSE: 0.0108105323606919, 0.0253207374204999, 0.0227071836079463
##  SAM: 0.18041330006273, 0.343407213823624, 0.298624680304907
##  Best: 1
##
## Muscovite
## MSE: 0.00225537835858555, 0.000367066152505344, 0.000703594828214392
##  SAM: 0.202663795396691, 0.0745610657333382, 0.105293338555023
```

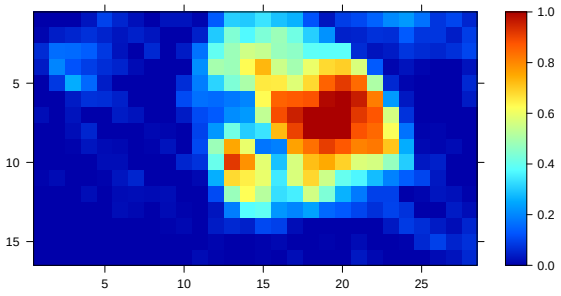
```
##  Best: 2
##
## Kaolinite
## MSE: 0.000863809216906234, 0.00469646817284637, 0.00295649463960596
##  SAM: 0.126578479250989, 0.181731615516772, 0.158705565282397
##  Best: 1
```

```
analysis <- function(hyImg, predlbl, printAs = 'cuprite'){
  em$plot(E = hyImg@pred[[predlbl]]$E, columnOrder = colnames(em$E_cuprite_ip),
          printAs = printAs, suffix = tolower(predlbl))
  hy$plotMapW(hyImg@pred[[predlbl]]$W, bw = FALSE,
              printAs = printAs, suffix = tolower(predlbl))
}
IMCuReD@pred[['ICE']]@eLabels <- c('Alunite', 'Muscovite', 'Kaolinite')
analysis(hyImg = IMCuReD, predlbl = 'ICE')
```

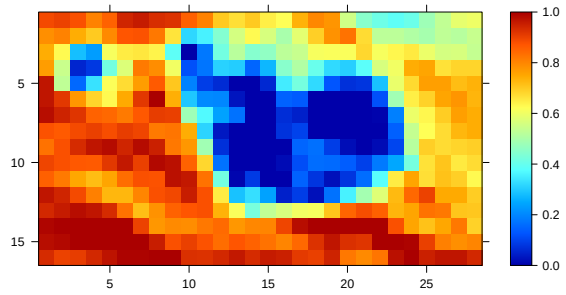
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



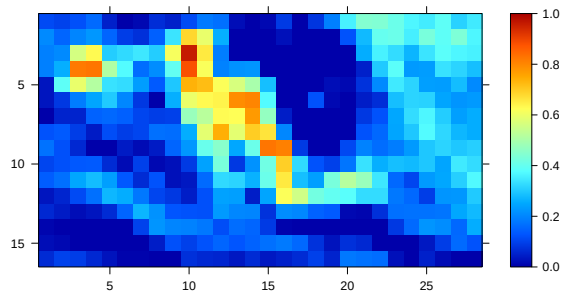

```
## Problematic W-values:
##  Count: 107
##  Min: -1.2390906838574e-14
##  Max: 1.00000000000001
## [1] "Plotting: Alunite"
```




```
## [1] "Plotting: Muscovite"
```




```
## [1] "Plotting: Kaolinite"
```



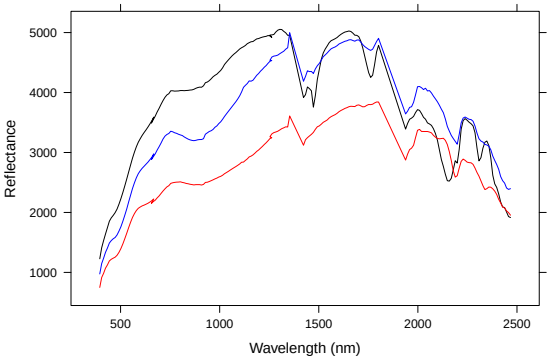
18.6.2.3.2 ICE-S

```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['ICE-0.5-S-0.02']]$E, linear = TRUE)
```

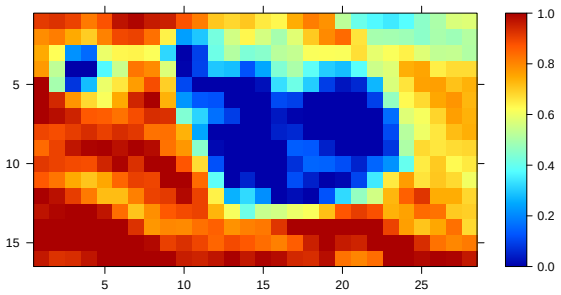
```
## Alunite
## MSE: 0.0252027688509065, 0.0232350005876469, 0.0122729254759273
##  SAM: 0.338777034951882, 0.305883333390541, 0.19309861993634
##  Best: 3
##
## Muscovite
## MSE: 0.000368888887348566, 0.000634609548605595, 0.00203442748405039
##  SAM: 0.0746486137960632, 0.0997898220068084, 0.188565894602668
##  Best: 1
##
## Kaolinite
## MSE: 0.00453703502716013, 0.00319001010107122, 0.000886816751581339
##  SAM: 0.178028592411862, 0.162515922249012, 0.122861463185657
##  Best: 3
```

```
IMCuReD@pred[['ICE-0.5-S-0.02']]@eLabels <- c('Muscovite', 'Kaolinite', 'Alunite')
analysis(hyImg = IMCuReD, predlbl = 'ICE-0.5-S-0.02')
```

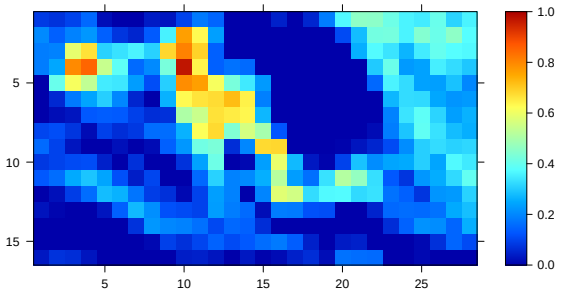
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



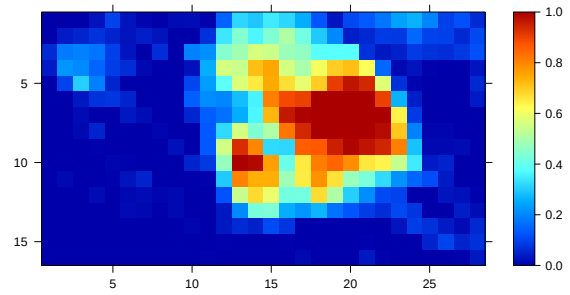

```
## Problematic W-values:
##  Count: 116
##  Min: -1.2620674231995e-14
##  Max: 1
## [1] "Plotting: Muscovite"
```




```
## [1] "Plotting: Kaolinite"
```




```
## [1] "Plotting: Alunite"
```



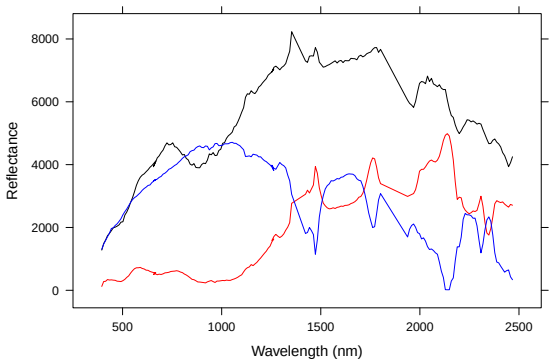
18.6.2.3.3 Bayes

```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['Bayes']]$E, linear = TRUE)
```

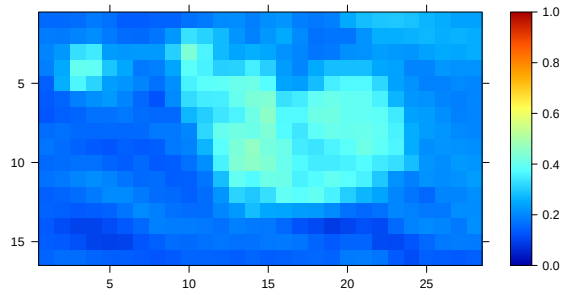
```
## Alunite
## MSE: 0.0254178505919392, 0.0150793203812684, 0.00255509548051903
##  SAM: 0.379289147554477, 0.83738982917785, 0.176642093845487
##  Best: 3
##
## Muscovite
## MSE: 0.000647600030378, 0.00248737052519529, 0.00403053551447522
##  SAM: 0.117651116822118, 0.517800339793939, 0.459650183483825
##  Best: 1
##
## Kaolinite
## MSE: 0.00502195402528758, 0.008590302605634, 0.00424275763817521
##  SAM: 0.228923217602364, 0.660703335023029, 0.328206731698702
##  Best: 1
```

```
IMCuReD@pred[['Bayes']]@eLabels <- c('Alunite', 'Muscovite', 'Kaolinite')
analysis(hyImg = IMCuReD, predlbl = 'Bayes')
```

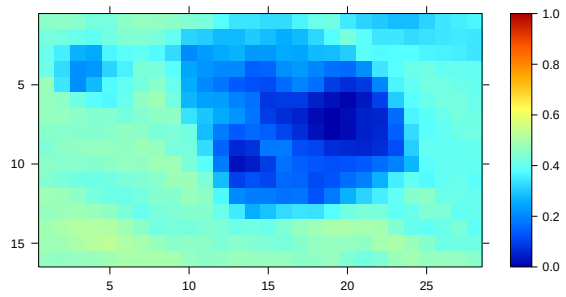
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



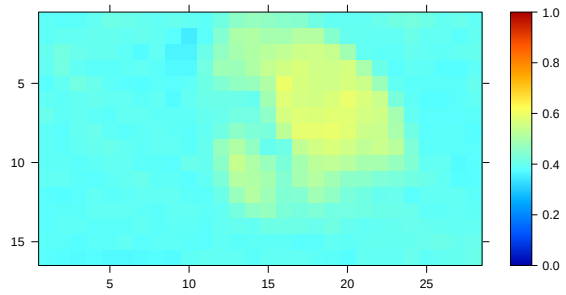

```
## [1] "Plotting: Alunite"
```




```
## [1] "Plotting: Muscovite"
```




```
## [1] "Plotting: Kaolinite"
```



18.6.2.3.4 ICE optimal

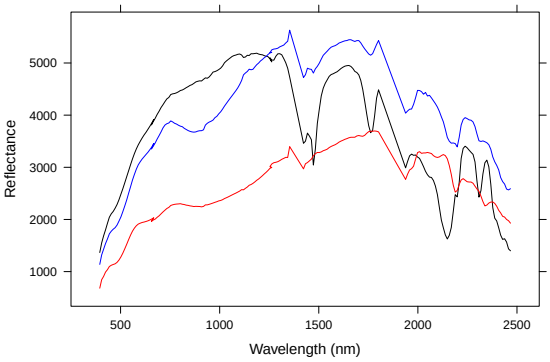
```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['ICE-opt']]$E, linear = TRUE)
```

```
## Alunite
## MSE: 0.0052210494590033, 0.025606392766681, 0.0215518426931818
##  SAM: 0.130179208400162, 0.365365344515789, 0.283537342862164
##  Best: 1
##
## Muscovite
## MSE: 0.00323427359085768, 0.000404825840740594, 0.000826997970382507
##  SAM: 0.28251211719314, 0.0797238950882079, 0.11424640764125
##  Best: 2
##
## Kaolinite
```

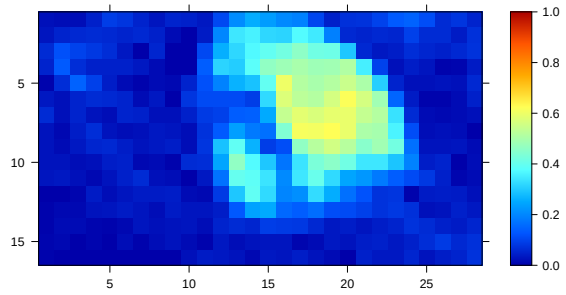
```
## MSE: 0.00157554148216875, 0.00538580011136632, 0.00250787699349513
##  SAM: 0.174383527576004, 0.200458641961294, 0.149993542797318
##  Best: 3
```

```
IMCuReD@pred[['ICE-opt']]@eLabels <- c('Alunite', 'Muscovite', 'Kaolinite')
analysis(hyImg = IMCuReD, predlbl = 'ICE-opt')
```

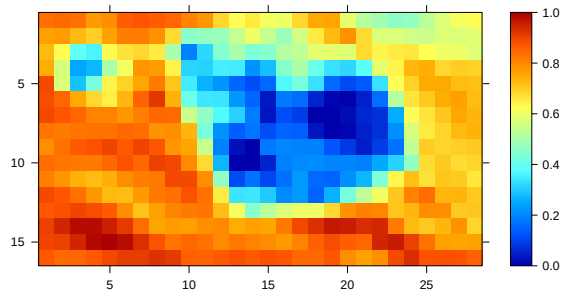
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



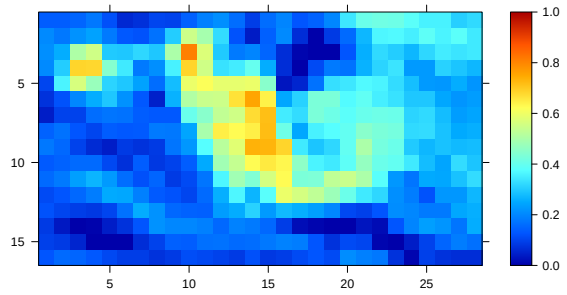

```
## Problematic W-values:
##  Count: 24
##  Min: -8.0646789753681e-15
##  Max: -5.91618127966114e-18
## [1] "Plotting: Alunite"
```




```
## [1] "Plotting: Muscovite"
```




```
## [1] "Plotting: Kaolinite"
```



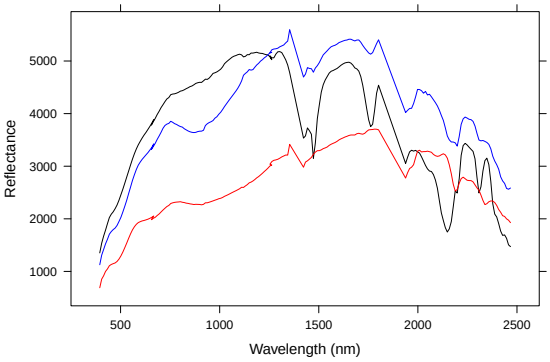
18.6.2.3.5 ICE-S optimal

```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['ICE-S-opt']]$E, linear = TRUE)
```

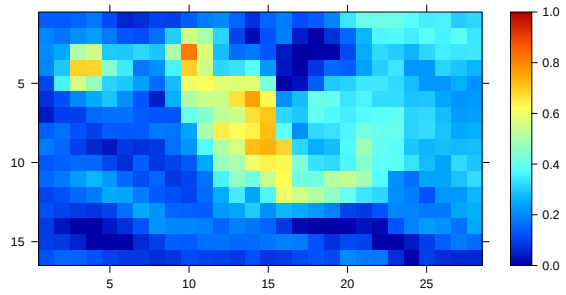
```
## Alunite
## MSE: 0.0217343020696105, 0.00585482024117057, 0.025585420240195
##  SAM: 0.285783041624042, 0.135768436089531, 0.361666156130063
##  Best: 2
##
## Muscovite
## MSE: 0.000806527712132458, 0.0031122390137222, 0.000395121570136928
##  SAM: 0.112792545554379, 0.269965485957763, 0.0784133409992431
##  Best: 3
##
## Kaolinite
## MSE: 0.00257233963713578, 0.00140920089611247, 0.00527617445206584
##  SAM: 0.151234115584813, 0.165136235007196, 0.197227478728643
##  Best: 1
```

```
IMCuReD@pred[['ICE-S-opt']]@eLabels <- c('Kaolinite', 'Alunite', 'Muscovite')
analysis(hyImg = IMCuReD, predlbl = 'ICE-S-opt')
```

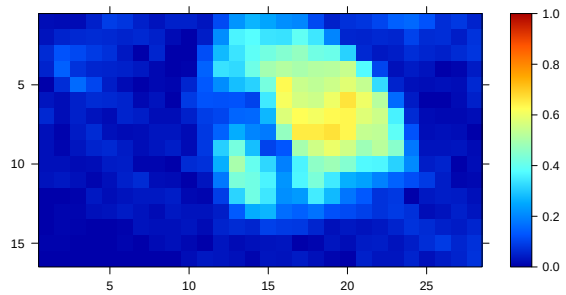
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



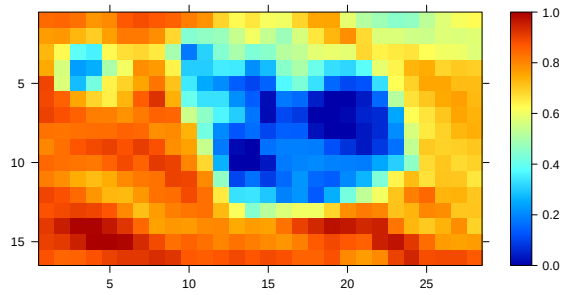

```
## Problematic W-values:
##  Count: 26
##  Min: -4.92951727260689e-15
##  Max: -4.34054256622271e-19
## [1] "Plotting: Kaolinite"
```




```
## [1] "Plotting: Alunite"
```




```
## [1] "Plotting: Muscovite"
```



18.6.2.3.6 Bayes optimal

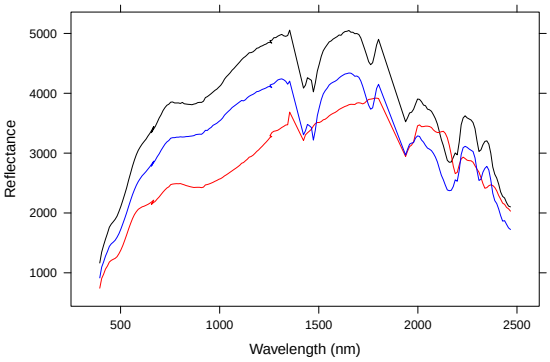
```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['Bayes-opt']]$E, linear = TRUE)
```

```
## Alunite
## MSE: 0.0254571298332433, 0.0161042139131584, 0.0163018737099096
##  SAM: 0.35211622168501, 0.228420135054492, 0.227878707641497
##  Best: 3
##
## Muscovite
## MSE: 0.000379702544157143, 0.00155719438761905, 0.00144461792956529
##  SAM: 0.0764816392423484, 0.161275339424931, 0.152630546176226
##  Best: 1
##
## Kaolinite
```

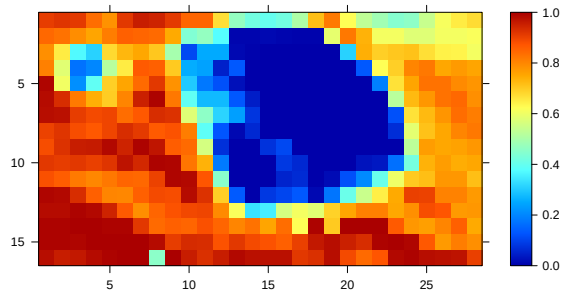
```
## MSE: 0.00492458985993584, 0.00131649010241189, 0.0012355564877563
##  SAM: 0.189732321219672, 0.130198016976013, 0.122328084797155
##  Best: 3
```

```
IMCuReD@pred[['Bayes-opt']]@eLabels <- c('Muscovite', 'Kaolinite', 'Alunite')
analysis(hyImg = IMCuReD, predlbl = 'Bayes-opt')
```

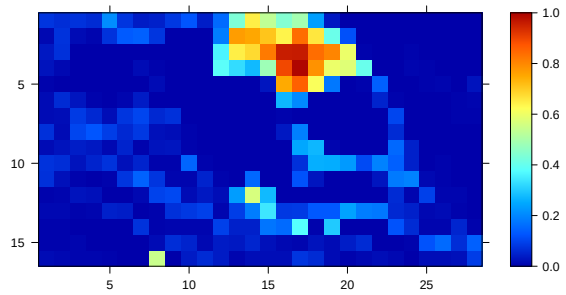
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



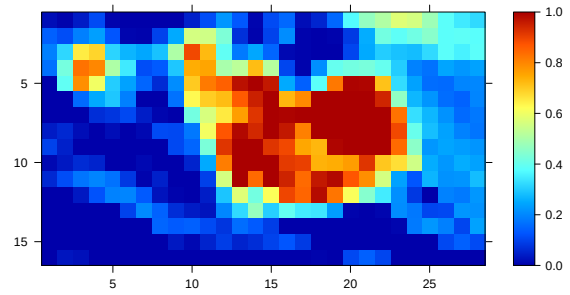

```
## [1] "Plotting: Muscovite"
```




```
## [1] "Plotting: Kaolinite"
```




```
## [1] "Plotting: Alunite"
```



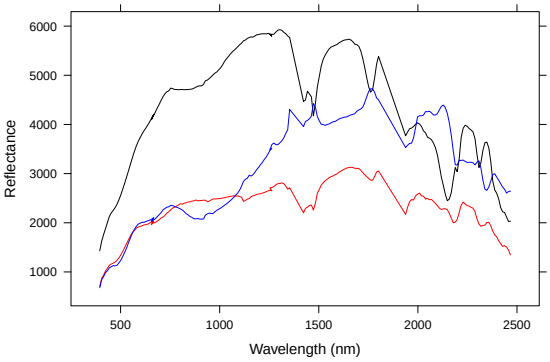
18.6.2.3.7 Bayes-Tau

```
em$matchSpectra(refE = em$E_cuprite_ip, E = IMCuReD@pred[['Bayes-Tau']]$E, linear = TRUE)
```

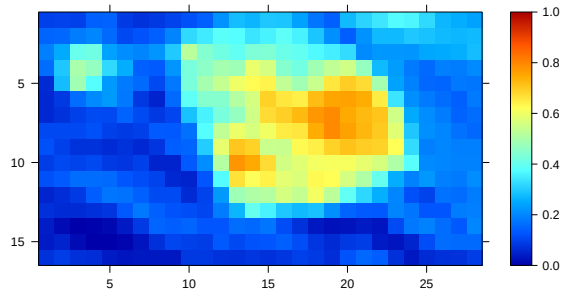
```
## Alunite
## MSE: 0.00988430191523429, 0.0245778456230239, 0.0206950008718514
##  SAM: 0.17483525490648, 0.456197618023758, 0.259726484944841
##  Best: 1
##
## Muscovite
## MSE: 0.00245442533596271, 0.000820060963537421, 0.000914774967075517
##  SAM: 0.220050065909204, 0.14603949942124, 0.117798676529586
##  Best: 3
##
## Kaolinite
## MSE: 0.000961748742477862, 0.0070693964828516, 0.00232813928036642
##  SAM: 0.141000028626029, 0.287894089863876, 0.117366396735935
##  Best: 3
```

```
IMCuReD@pred[['Bayes-Tau']]@eLabels <- c('Alunite', 'Kaolinite', 'Muscovite')
analysis(hyImg = IMCuReD, predlbl = 'Bayes-Tau')
```

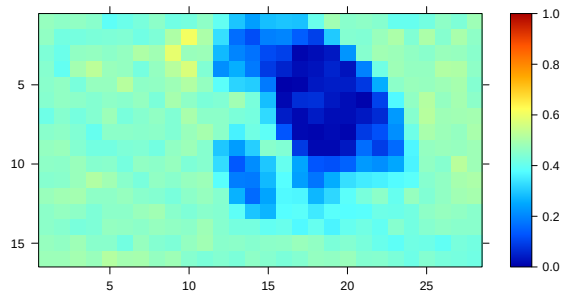
```
## Alunite: black
## Muscovite: red
## Kaolinite: blue
```



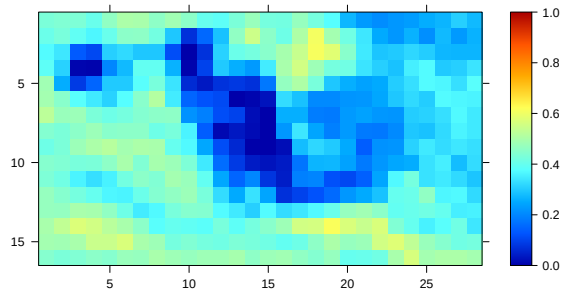

```
## [1] "Plotting: Alunite"
```




```
## [1] "Plotting: Kaolinite"
```




```
## [1] "Plotting: Muscovite"
```

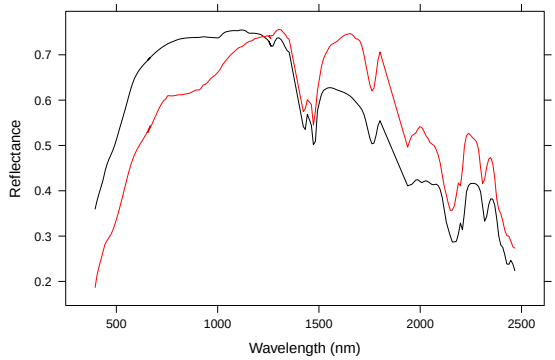


18.6.3 Comparing endmembers over the whole spectrum

18.6.3.1 Alunite

```
em$plotScaled(printAs = 'cuprite-spectra-alunite-full', legend = FALSE,
              cbind('Alunite' = em$E_cuprite_ip[,'Alunite'],
                    'ICE Alunite' = IMCuReD@pred[['ICE']]$E[,'Alunite'])
              )
```

```
## Alunite: black
## ICE Alunite: red
```

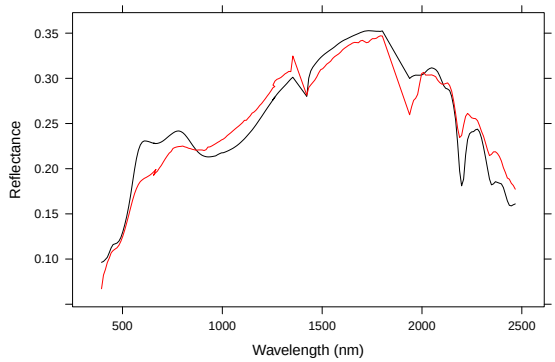



```
## pdf
##   2
```

18.6.3.2 Muscovite

```
em$plotScaled(printAs = 'cuprite-spectra-muscovite-full', legend = FALSE,
              cbind('Muscovite' = em$E_cuprite_ip[,'Muscovite'],
                    'ICE Muscovite' = IMCuReD@pred[['ICE']]$E[,'Muscovite']))
```

```
## Muscovite: black
## ICE Muscovite: red
```

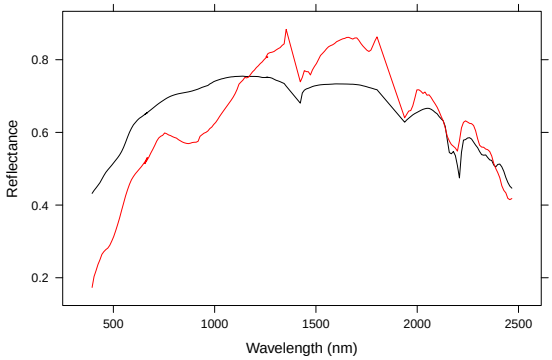



```
## pdf
##   2
```

18.6.3.3 Kaolinite

```
em$plotScaled(printAs = 'cuprite-spectra-kaolinite-full', legend = FALSE,
              cbind('Kaolinite' = em$E_cuprite_ip[,'Kaolinite'],
                    'ICE Kaolinite' = IMCuReD@pred[['ICE']]$E[,'Kaolinite']))
```

```
## Kaolinite: black
## ICE Kaolinite: red
```




```
## pdf
##   2
```

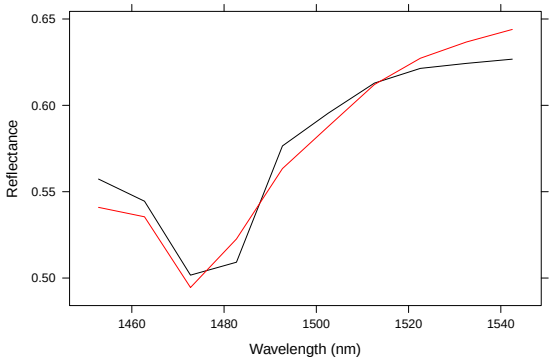
18.6.4 Comparing endmembers over diagnstic spectral ranges

18.6.4.1 Alunite

This is what Clark [2] calls the first step or first innovation. We do not go further into normalization.

```
em$plotScaled(printAs = 'cuprite-spectra-alunite-1450-1550',
              wlLim = c(1450, 1550), legend = FALSE,
    cbind('Alunite' = em$E_cuprite_ip[,'Alunite'],
          'ICE Alunite' = IMCuReD@pred[['ICE']]$E[,'Alunite']))
```

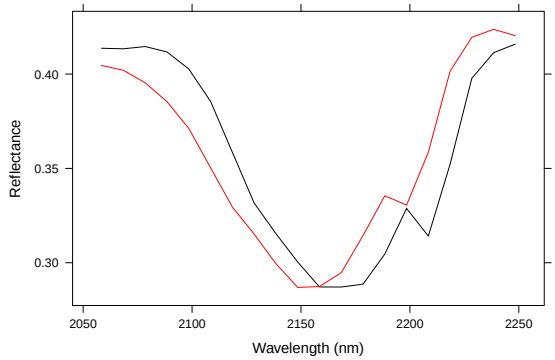
```
## Alunite: black
## ICE Alunite: red
```




```
## pdf
##   2
```

```
em$plotScaled(printAs = 'cuprite-spectra-alunite-2050-2250',
              wlLim = c(2050, 2250), legend = FALSE,
    cbind('Alunite' = em$E_cuprite_ip[,'Alunite'],
          'ICE Alunite' = IMCuReD@pred[['ICE']]$E[,'Alunite']))
```

```
## Alunite: black
## ICE Alunite: red
```

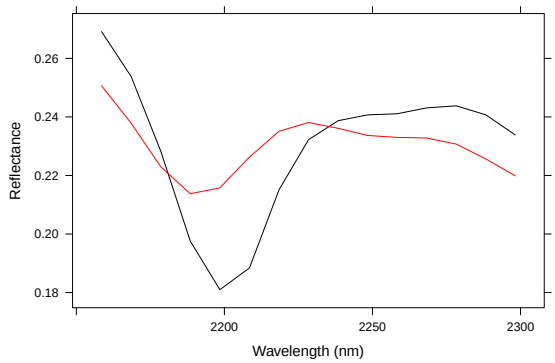



```
## pdf
##   2
```

18.6.4.2 Muscovite

```
em$plotScaled(printAs = 'cuprite-spectra-muscovite-2150-2300',
              wlLim = c(2150, 2300), legend = FALSE,
    cbind('Muscovite' = em$E_cuprite_ip[,'Muscovite'],
          'ICE Muscovite' = IMCuReD@pred[['ICE']]$E[,'Muscovite']))
```

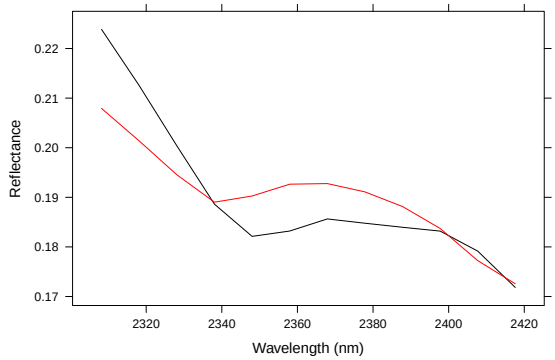
```
## Muscovite: black
## ICE Muscovite: red
```




```
## pdf
##   2
```

```
em$plotScaled(printAs = 'cuprite-spectra-muscovite-2300-2420',
              wlLim = c(2300, 2420), legend = FALSE,
    cbind('Muscovite' = em$E_cuprite_ip[,'Muscovite'],
          'ICE Muscovite' = IMCuReD@pred[['ICE']]$E[,'Muscovite']))
```

```
## Muscovite: black
## ICE Muscovite: red
```

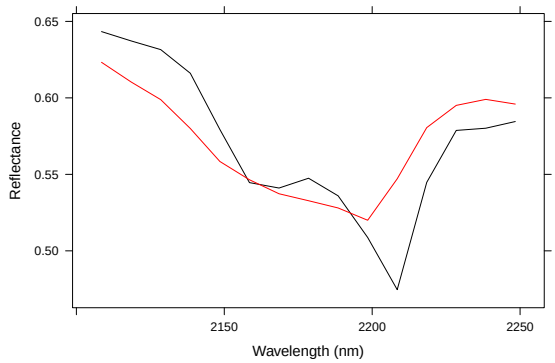



```
## pdf
##   2
```

18.6.4.3 Kaolinite

```
em$plotScaled(printAs = 'cuprite-spectra-kaolinite-2100-2250',
              wlLim = c(2100, 2250), legend = FALSE,
    cbind('Kaolinite' = em$E_cuprite_ip[,'Kaolinite'],
          'ICE Kaolinite' = IMCuReD@pred[['ICE']]$E[,'Kaolinite']))
```

```
## Kaolinite: black
## ICE Kaolinite: red
```




```
## pdf
##   2
```

18.7 Endmember MAP values

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['ICE']]$E)
```

```
## [1] 0.414
```

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['ICE-0.5-S-0.02']]$E)
```

```
## [1] 0.43
```

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['Bayes']]$E)
```

```
## [1] 0.923
```

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['ICE-opt']]$E)
```

```
## [1] 0.36
```

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['ICE-S-opt']]$E)
```

```
## [1] 0.365
```

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['Bayes-opt']]$E)
```

```
## [1] 0.427
```

```
measure$SAM(em$E_cuprite_ip, IMCuReD@pred[['Bayes-Tau']]$E)
```

```
## [1] 0.438
```

18.8 References

[1] Dobigeon, N. e.a., Joint Bayesian Endmember Extraction and Linear Unmixing for Hyperspectral Imagery, 2009 [2] Clark, R.N. e.a., Imaging spectroscopy - Earth and planetary remote sensing with the USGS Tetracorder and expert systems, 2003

19 Model comparison

19.1 Quick comparison of ICE and BayesNMF-Vol

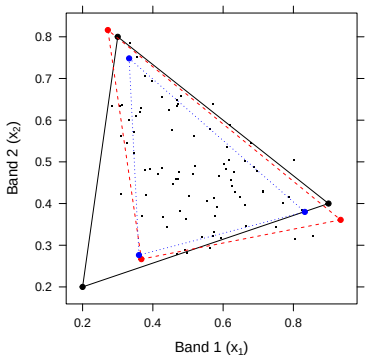
Now we can compute the endmembers $\boldsymbol{E}$ for the following image.

- ICE (red)
- Bayesian (blue)

```
IM9x9D2@pred[['Bayes']] <- alg$bVol$sample(hyImg = IM9x9D2,
                                           gAlpha = 0, gBeta = 0, gGamma = 0)
IM9x9D2@pred[['ICE']] <- alg$ice$ICE(hyImg = IM9x9D2, mu = 0.01)
```

```
## ICE convergence after 39 steps
```

```
hy$plotSimplex(X = IM9x9D2[[]], E = IM9x9D2@E,
               Ehat = list('Bayes' = IM9x9D2@pred[['Bayes']]$E,
                           'ICE' = IM9x9D2@pred[['ICE']]$E))
```




```
## [1] "Bayes color: red"
## [1] "ICE color: blue"
```