A Vignette for the R package, `larkPeaks`

The package `larkPeaks` provides functions in R for feature discovery in MALDI-TOF Spectra using LARK [Clyde and Wolpert, 2007, Wolpert et al., 2010]. We take a Lévy random field modeling approach to estimate, with uncertainty, the location and abundance of protein peaks in spectra. In House et al. [2010], we describe our model in detail.

For this vignette, we assess a MALDI-TOF spectrum of a protein mixture for which we know the contents. This mixture was used as an example in Section 6.3 of House et al. [2010] and includes five known proteins. Measurements of one protein may appear across multiple peaks in spectra due to properties of MALDI-TOF mass spectrometry. In the spectrum that we analyze for the known mixture, there are 11 peaks (Figure 1).

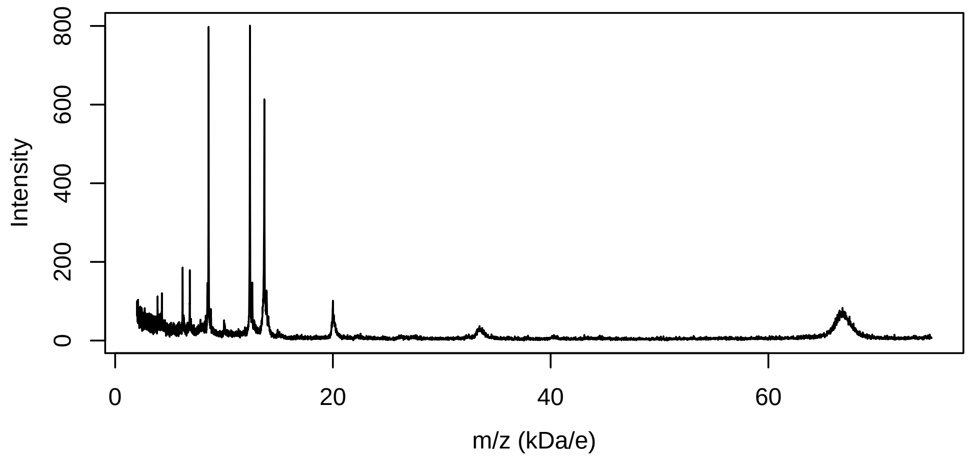

Figure 1: Spectrum of protein sample with known contents.

The primary function of interest in `larkPeaks` that may discovery the location and abundance of the 11 peaks is called `findPeaks`. This function is complex and requires several argument specifications to operate. Figure 2 lists the arguments and groups them by purpose within the function, `findPeaks`:

There are two arguments to which final estimates for peak location and abundance are sensitive: `gauErr` (the distribution for the random error observed in spectra) and `gauKern` (the peak shape).

- MCMC specifications
  - `id`: Numerical identifier for output stored as text.
  - `iterations`: Number of MCMC iterations
  - `thin`: Number by which to thin the MCMC draws; `iterations/thin` == total stored
  - `keepEnd`: Number of samples to store at the end of MCMC; e.g., if iterations=1000, thin=10, keepEnd=35, then 121 samples are stored
  - `movePr` 5x1 vector of probabilities for the MCMC to Birth, Death, Update, Merge, Split; e.g. `movePr`=c(.1,.1, .6, .1, .1)
  - `rwStd`: 7 or 8 values for std. dev. of random walks of the following parameters: `eta`, `loc`, `res`, `phi`, `S`, `expPar`[1], `expPar`[2], c
- Data
  - `y`: Vector of intensities
  - `clockticks`: Vector of clock ticks or time
- Starting values, model specifications, prior parameters
  - `loc`: Vector of starting locations
  - `eta`: Vector of starting concentrations
  - `res`: Vector of starting resolutions
  - `gauKern`: =0 if modeling with Gaussian kernel, =1 if modeling with Cauchy kernel
  - `exPks`: Prior expected number of peaks
  - `minSignal`: Percent of minimum signal, e.g., =0.075
  - `res.var`: Variance for res given specRes
  - `specRes`: Starting values for the overall, spectrum resolution (i.e., varRho in paper); see below for details
  - `specRes.bounds`: If `specRes` changes within a spectrum, then `specRes` is a vector, otherwise it is a scalar, `specRes`=specRes.cutoff=max(`clockticks`)
  - `specRes.mean`: Prior mean for specRes; length(specRes.mean)=length(specRes)
  - `specRes.var`: Prior variance for specRes; length(specRes.var)=length(specRes)
  - `expPar` 2 x 1 vector of exponential decay parameters; expPar[1]=eta0, expPar[2]=lambda0
  - `decay.var`: Variance for expPar[2]
  - `phi`: Starting value for phi
  - `phi.mean`: Prior shape parameter for phi
  - `phi.var`: Prior rate parameter for phi
  - `S`: Starting value for percent signal
  - `S.mean`: Prior mean of percent signal
  - `S.var`: Prior variance of percent signal
  - `c`: Starting value for zeta, Intensity mean; if fixC==1, c=mean(y)
  - `c.mean`: Prior mean of zeta
  - `c.var`: Prior variance of zeta
  - `fixC`: Fix c, rather than solving for it
  - `gauErr`: 1 if fitting a Gaussian Error model, and 0 if Gamma Error model
- Miscellaneous
  - `asBernOnly`: 1 if return Bernoulli output only
  - `dump`: Ignore; Set to 0
  - `plotStart`: 1 if plot the starting locations, res, eta
  - `plotMa`: Plot the Model Average
  - `micro`: Conversion for microseconds; 1 clocktick = micro microseconds

Figure 2: Arguments for the function `findPeaks`

Each of these arguments has two options, thus there are four scenarios that we can explore.

Scenario	**Notation**	**Parameter Specification**	**Interpretation in Model**
00	CG	`gauKern`=0, `gauErr`=0	Cauchy shaped peaks and Gamma error
01	CN	`gauKern`=0, `gauErr`=1	Cauchy shaped peaks and Gaussian error
10	NG	`gauKern`=1, `gauErr`=0	Gaussian shaped peaks and Gamma error
11	NN	`gauKern`=1, `gauErr`=1	Gaussian shaped peaks and Gaussian error

In the sections that we follow we will implement `findPeaks` for the four scenarios.

1 Scenario 00: CG

To start, we load the library and data. `knownPeaks` includes three variables that provides the data for one spectrum. These variables are *clock*, *mz*, and *intensity*. For this application, $knownPeaks\$clock \times 0.016\mu s$ scales the x-axis of Figure 1 to microseconds. `knownPeaksLoc` includes the peak locations in mz units.

```
> library(larkPeaks)
> data(knownPeaks)
> data(knownPeaksLoc)
```

Plot the data with the peak locations.

```
> plot(knownPeaks$mz,  knownPeaks$intensity,  type="l", xlab="mz", ylab="Intensity")
> rug(knownPeaksLoc$mz)
```

To re-create the spectrum with peak locations so that the x-axis is in clock-tick units, use the function `interp.m2t`.

```
> locCl <- apply(matrix(knownPeaksLoc$mz), 1, interp.m2t, knownPeaks$clock,knownPeaks$mz)
> plot(knownPeaks$clock,  knownPeaks$intensity,  type="l",
            xlab="Clock-ticks", ylab="Intensity")
> rug(locCl)
```

To start our analysis of `knownPeaks`, we make the following MCMC specifications.

```
> ID <- 1
> I <- 1000000
> T <- 1000
> KE <- 0
> MP <- c(.1,.1,.6,.1,.1)  #Move probabilities
> RW <- c(.1,10,15,.01,.05,.05,.1,.1)  #Random walk standard deviations
```

These specifications are application specific and are not default values. With the exception for ID (the identification number is arbitrary), users of `findpeaks` must select the remaining parameters to assure convergence and useful acceptance and rejection rates.

Now, we make the following model specifications. The rational for these specifications is described in House et al. [2010]. Again, these specifications are not recommended as default values and are relevant only for the data at hand, `knownPeaks`.

```
> microseconds <- .016
> J <- 20
> R <- 200; Rvar <- .35^2; Rvar2 <- .7^2
> miS <- .075
> SM <- 0.90
> EP <- c( 7956.4, 2024.79)
> DV <- .5^2
> PH <- 0.00965
> PHV <- (PH*2)^2
```

In `larkPeaks`, we provide some functions to help with parameter specifications and starting values, including `getMomentS`, `getStartEta` and `getStartRes`. Given an expected value for S (defined in House et al., 2010 as 'percent signal'), the function `getMomentS` provides both a default variance specification for S and a set scale parameters for the prior distribution of S. Below, we define the variance of S as `varS` using `getMomentS`.

```
> varS <- getMomentS(SM,1)[[2]]
```

The functions `getStartEta` and `getStartRes` provide, respectively, starting values for protein abundances and peak resolutions given peak locations. For this application, we will start the MCMC with peak locations that are scattered uniformly across the x-axis. Using `getStartRes`, we set each peak resolution based on the FWHM procedure and the mean spectrum resolution R that we defined above.

```
> startLoc <- seq(min(knownPeaks$clock)+5, max(knownPeaks$clock)-5, length=J)
> startRes <- getStartRes(length(startLoc),mu=R, sd=.35)
```

Selecting starting values for peak intensities can be complex. We may set these values using information in the data or to arbitrary values of our choosing. For either approach, we can use the function `getStartEta`. If we decide to use data-determined intensity specifications, we set the argument *dataDeter* to 1. If not, we set *dataDeter* to a number less than zero (e.g., -2) so that intensities are calculated as a multiple of the smallest possible intensity the model will recognize. The multiple equals the absolute value of what was assigned to *dataDeter*.

```
> startEta <- getStartEta(length(startLoc), dataDeter=-2, minSig=miS,
                         x=knownPeaks$clock, y=knownPeaks$intensity,
                        C=mean(knownPeaks$intensity), S=SM, startLoc, startRes, EP[1])
```

If we want the multiple to change for each peak, we simply set *dataDeter* to a numeric vector of length equal to J, i.e., `length(startLoc)`. In this case, $J = 20$, thus we could set *dataDeter* to `c(rep(-2,5), rep(-3, 5), rep(-4, 5), rep(-5,5))`. Notice that each entry for *dataDeter* is negative as well.

Now we are ready to implement `findPeaks`. Since we want to model the data so that we assume the peaks shapes are Cauchy and the error distribution is Gamma, we set both arguments *gauKern* and *gauErr* to 0.

```
> knPksCG <- findPeaks(id=ID,iterations=I, thin=T,keepEnd=KE, movePr=MP, rwStd=RW,
                         y=knownPeaks$intensity,clockticks=knownPeaks$clock,
                         loc=startLoc,eta=startEta,res=startRes,gauKern=0,
                         exPks=J, minSig=miS,
                         res.var=Rvar, specRes=R, specRes.bounds=max(knownPeaks$clock),
                         specRes.mean=R, specRes.var=Rvar2,
                         expPar= EP, decay.var=DV,
                         phi=PH,phi.mean=PH, phi.var=PHV,
                         S=SM, S.mean=  SM, S.var= varS,
                         c=mean(knownPeaks$intensity), c.mean=mean(knownPeaks$intensity),
                         c.var=c(),fixC=1,gauErr=0,
                         asBernOnly=0, dump=0, plotStart=1, plotMa=1, micro=microseconds)
```

The function `findPeaks` returns a list. The second element in the list is also a list that contains 12 elements which pertain directly to the MCMC parameter samples. We may visualize the model average (MA) and the corresponding MA peak location estimates by selecting the appropriate element from `knPksCG` and using the function, `getMApks`.

```
> mafCG <- knPksCG[[2]][[12]][,8]
> maCG <- getMApks(knPksCG[[2]][[12]][,1], knownPeaks$mz)
> plot(knownPeaks[,2:3], type="l", col="grey")
> lines(knownPeaks[,2], mafCG)
> points(knownPeaksLoc[,1], rep(0, dim(knownPeaksLoc)[[1]]), pch=19, col="green")
> rug(maCG[[1]], col="red")
```

To learn the peak locations from the model with the highest posterior density (HP), we use the functions `getMXpks` and `interp.t2m`. The latter function scales the units for peak locations from clock-ticks to mz.

```
> mxCGt <- getMXpks(findPeaksRes=knPksCG, burnin=0)
> mxCG <- apply(matrix(mxCGt[[3]]), 1, interp.t2m, knownPeaks$clock,knownPeaks$mz)
> rug(mxCG, col="blue")  #add to the plot above
```

2 Scenarios 01, 10, 11

We use the above MCMC and model specifications, but we change the values for arguments $gauKern$ and $gauErr$.

```
> knPksCN <- findPeaks(id=ID,iterations=I, thin=T,keepEnd=KE, movePr=MP, rwStd=RW,
                        y=knownPeaks$intensity,clockticks=knownPeaks$clock,
                        loc=startLoc,eta=startEta,res=startRes,gauKern=0,
                        exPks=J, minSig=miS,
                        res.var=Rvar, specRes=R, specRes.bounds=max(knownPeaks$clock),
                        specRes.mean=R, specRes.var=Rvar2,
                        expPar= EP, decay.var=DV,
                        phi=PH,phi.mean=PH, phi.var=PHV,
                        S=SM, S.mean=  SM, S.var= varS,
                        c=mean(knownPeaks$intensity), c.mean=mean(knownPeaks$intensity),
                        c.var=c(),fixC=1,gauErr=1,
                        asBernOnly=0, dump=0, plotStart=1, plotMa=1, micro=microseconds)


> knPksNG <- findPeaks(id=ID,iterations=I, thin=T,keepEnd=KE, movePr=MP, rwStd=RW,
                        y=knownPeaks$intensity,clockticks=knownPeaks$clock,
                        loc=startLoc,eta=startEta,res=startRes,gauKern=1,
                        exPks=J, minSig=miS,
                        res.var=Rvar, specRes=R, specRes.bounds=max(knownPeaks$clock),
                        specRes.mean=R, specRes.var=Rvar2,
                        expPar= EP, decay.var=DV,
                        phi=PH,phi.mean=PH, phi.var=PHV,
                        S=SM, S.mean=  SM, S.var= varS,
                        c=mean(knownPeaks$intensity), c.mean=mean(knownPeaks$intensity),
                        c.var=c(),fixC=1,gauErr=0,
                        asBernOnly=0, dump=0, plotStart=1, plotMa=1, micro=microseconds)


> knPksNN <- findPeaks(id=ID,iterations=I, thin=T,keepEnd=KE, movePr=MP, rwStd=RW,
                        y=knownPeaks$intensity,clockticks=knownPeaks$clock,
                        loc=startLoc,eta=startEta,res=startRes,gauKern=1,
                        exPks=J, minSig=miS,
                        res.var=Rvar, specRes=R, specRes.bounds=max(knownPeaks$clock),
                        specRes.mean=R, specRes.var=Rvar2,
                        expPar= EP, decay.var=DV,
                        phi=PH,phi.mean=PH, phi.var=PHV,
                        S=SM, S.mean=  SM, S.var= varS,
                        c=mean(knownPeaks$intensity), c.mean=mean(knownPeaks$intensity),
                        c.var=c(),fixC=1,gauErr=1,
                        asBernOnly=0, dump=0, plotStart=1, plotMa=1, micro=microseconds)
```

To assess the differences between `knPksCG`, `knPksCN`, `knPksNG` and `knPksNN`, we use the function provided in the appendix to create the following table.

Table 1: Compares results for analyzing a spectrum with varying model assumptions that pertain to the peak shape and the random error distribution. For Scenario CG, we assumed Cauchy shaped peaks and Gamma error; for Scenario CN, we assumed Cauchy shaped peaks and Normal error; for Scenario NG, we assumed Normal shaped peaks and Gamma error; and for Scenario NN, we assumed Normal shaped peaks and Normal error. Notice that the number of peak estimates differ dramitcally between the Gamma error models and the Gaussian Error models.

Scenario	S	ϕ	ϱ	η_0	ω_0	J^{PM}	J^{HP}	$J^{\triangledown}$
CG	0.90	2.61	96.43	78.26	22.80	42.24	42.00	28.00
CN	0.94	0.09	156.71	91.56	30.44	84.90	85.00	34.00
NG	0.89	2.35	73.90	91.29	26.21	49.11	49.00	29.00
NN	1.00	0.09	84.93	104.92	39.15	83.90	83.00	30.00

Appendix

We use the code in this appendix to assess and compare outcomes from the function `findPeaks()` that are stored in objects `knPksCG`, `knPksCN`, `knPksNG`, and `knPksNN`.

```
> postSum <- function(out, BI,map, mxp, micro){
  #out: results from findpeaks()
  #BI: Burn-in; this BI must match that used to determine MA and HP peak estimates.
  #map: peak locations from MA
  #mxp: peak locations from HP model
  #micro:  number of microseconds per clock-tick

  keep <- floor(BI*length(out$JVec)):length(out$JVec)
  fpke  <- micro*(unlist(out$expParVec[keep]))[seq(2,4*length(keep), by=4)]
  fpke2 <- micro*(unlist(out$expParVec))[seq(2,4*length(keep), by=4)]
  fpkl  <- micro/(unlist(out$expParVec[keep]))[seq(4,4*length(keep), by=4)]
  fpkl2 <- micro/(unlist(out$expParVec))[seq(4,4*length(keep), by=4)]

  par(mfrow=c(3,2))
  plot(out$JVec, type="l")
  plot(out$specResVec, type="l")
  plot(fpke2/micro, type="l")
  plot(fpkl2/micro, type="l")
  plot(out$phiVec, type="l")
  plot(out$noSignalVec, type="l")


  mnJ <- mean(out$JVec[keep])
  mnSR <- mean(out$specResVec[keep])
  mnFE <- mean(fpke)
  mnFL <- mean(fpkl)
  mnPh <- mean(out$phiVec[keep])
  mnNS <- mean(out$noSignalVec[keep])

  vaJ <- var(out$JVec)#[keep])
  vaSR <- var(out$specResVec[keep])
  vaFE <- var(fpke)
  vaFL <- var(fpkl)
  vaPh <- var(out$phiVec[keep])
  vaNS <- var(out$noSignalVec[keep])

  maJ <- length(map)
  mxJ <- length(mxp)

  rowMn <- c(mnSR,mnFE,mnFL,mnPh,1-mnNS,mnJ, maJ,mxJ)
  rowVa <- c(vaSR,vaFE,vaFL,vaPh,vaNS,vaJ, 0,0)

  sumry <- rbind(rowMn, rowVa)
  colnames(sumry) <-  c("SpecRes", "ExpInt","ExpDec", "Phi", "S", "stochJ", "MA J", "HP J")

  return(sumry)
}

> allSum <- matrix(0,8,8)
> colnames(allSum) <- c("SpecRes", "ExpInt","ExpDec", "Phi", "S", "stochJ", "MA J", "HP J")
> rownames(allSum) <- c("mnCG", "vaCG","mnCN", "vaCN","mnNG", "vaNG","mnNN", "vaNN")

> allSum[1:2,] <- postSum(knPksCG[[2]],.5,maCG[[1]], mxCG,microseconds)
> allSum[3:4,] <- postSum(knPksCN[[2]],.5,maCN[[1]], mxCN,microseconds)
> allSum[5:6,] <- postSum(knPksNG[[2]],.5,maNG[[1]], mxNG,microseconds)
> allSum[7:8,] <- postSum(knPksNN[[2]],.5,maNN[[1]], mxNN,microseconds)
> allSum <- allSum[,c(5,4,1,2,3,6,8,7)]
> allSum
```

References

M. A. Clyde and R. L. Wolpert. *Bayesian Statistics 8 (J. M. Bernardo, M. J. Bayarri, J. O. Berger, A. P. Dawid, D. Heckerman, A. F. M. Smith and M. West, eds.)*, chapter Nonparametric function estimation using overcomplete dictionaries, page 91114. Oxford Univ. Press, Oxford, UK, 2007.

L. L. House, M. A. Clyde, and R. L. Wolpert. Bayesian nonparametric models for peak identification in maldi-tof mass spectroscopy. *Annals of Applied Statistics (to appear)*, 2010.

R. L. Wolpert, M. A. Clyde, and C. Tu. Stochastic expansions using continuous dictionaries: L'evy adaptive regression kernels. Technical Report 2006-08, Duke, 2010.