A Bioconductor workflow for the Bayesian Analysis of Spatial proteomics

Oliver M. Crook, Lisa Breckels, Kathryn S. Lilley, Paul D.W. Kirk, Laurent Gatto

Version

R version: R version 3.5.1 (2018-07-02) **Bioconductor version**: 3.8

Introduction

Quantifying uncertainty in the spatial distribution of proteins allows for novel insight into protein function. Many proteins live in a single location within the cell, however there are those that reside in mutiple locations and those that dynamically relocalise. Functional comparmentalisation of proteins allows the cell to control biomolecular pathways and biochemical process within the cell. Therefore, proteins with multiple localisation may have mutiple functional roles. Machine learning algorithms that fail to quantify uncertainty are unable to draw deeper insight into understanding cell biology from mass-spectrometry (MS) based spatial proteomics experiments.

Bayesian approaches to machine learning and statistical analysis can provide more insight into the data, since uncertainty quantification arises as a consequence of a generative model for the data. In a Bayesian framework, a model with paramters for the data is proposed, along with a statement about our prior beliefs of the model paramters. Bayes' theorem tells us how to update the prior distribution of the parameters to obtain the posterior distribution of the parameters after observing the data. It is the posterior distribution which quantifies the uncertainty in the parameters and quantities of interest derived from the data. This contrasts from a maximum-likelihood approach where we obtain only a point estimate of the parameters.

Adopting a Bayesian framework for data analysis, though of much interest to experimentalists, can be challenging. Once we have obtained a probabilistic model, complex algorithms are used to obtain the posterior distribution upon observation of the data. These algorithms can have tuning parameters and many settings, hindering their practical use for those not versed in Bayesian methodology. Even once the algorithms have been correctly set-up, assessments of convergence and guidance on how to intepret the results are often sparse. This workflow presents a Bayesian analysis of spatial proteomics to elucidate the process to any practioners. We hope that it goes beyond simply the methods, data structures and biology presented here, but provides a template for others to design tools using Bayesian methodology for the biological community.

Our model for the data is the t-augmented Gaussian mixture (TAGM) model proposed in:

> A Bayesian Mixture Modelling Approach For Spatial Proteomics Oliver M Crook, Claire M Mulvey, Paul D. W. Kirk, Kathryn S Lilley, Laurent Gatto bioRxiv 282269; doi: https://doi.org/10.1101/282269

The above manuscript provides a detailed description of the model, rigorous comparisons and testing on many spatial proteomics datasets and a case study on a hyperLOPIT experiment on mouse pluripotent stem cells. Revisiting these details is not the purpose of this computational protocol, rather we present how to correctly use the software and provide step by step guidance for interpreting the results.

In brief, the TAGM model posits that each annotated sub-cellular niche can be described by a Gaussian distribution. Thus the full complement of proteins within the cell is captured as a mixture of Gaussians. The highly dynamic nature of the cell means that many proteins are not well captured by any of these multivariate Gaussian distributions, and thus the model also includes an outlier component, mathematically described as multivariate student's t distribution. The heavy tails of the t distribution allow it to better capture dispersed proteins.

To perform inference in the TAGM model there are two approaches. The first, which we refer to as TAGM-MAP, allows us to obtain *maximum a posteriori* estimates of posterior localisation probabilities; that is, the modal posterior probability that a protein localises to that class. This approach uses the expectation-maximisation (EM) algorithm to perform inference. Whilst this is a interpretable summary of the TAGM model, it only provides point estimates. For a richer analysis, we present a Markov-chain Monte-Carlo (MCMC) method to perform fully Bayesian inference in our model, allowing us to obtain full posterior localisation distributions. This method is refered to as TAGM-MCMC throughout the text.

This workflow begins with a brief review of some of the basic features of mass-spectrometry based spatial proteomics data, including the state-of-the-art computational infrastructure and bespoke software suite. We then present each method in turn, detailing how to obtain high quality results. We provide an extended dicussion of the TAGM-MCMC method to highlight some of the challenges when apply this method. This includes how to assess convergence of MCMC methods, as well as methods for manipulating the output. We then take the processed output and explain how to intepret the results, as well as providing some tools for visualisation. We conclude with some remarks and directions for the future.

Getting started and infrastructure

In this workflow, we are currently using the development version of `pRoloc` and the current Bioconductor version of `pRolocdata` and `MSnbase`. The pacakge `pRoloc` contains algorithms and methods for analysing spatial proteomics data, building on the `MSnSet` structure provided in `MSnbase`. The `pRolocdata` package provides many annotated datasets from a variety of species and experimental procedures. The following code chunk installs the software suite of packages require for analysis.

```r
# require(devtools)
#install_github("lgatto/pRoloc")
#BiocManager::install(c("MSnbase", "pRolocdata"))
require(pRolocdata)
```

We assume that we have a MS-based spatial proteomics dataset contained in a `MSnSet` structure. For information on how to import data, perform basic data processing, quality control, supervised machine learning and transfer learning see Lisa's workflow. We use a spatial proteomics dataset on Mouse E14TG2a embryonic stem cells. The LOPIT protocol was used and normalised intensity of proteins from eight iTRAQ 8-plex labelled fraction are provided. The methods provided here are independent of labelling procedure, fractionation process or workflow. Examples of valid experimental protocols are LOPIT, hyperLOPIT, label-free methods such as PCP, and when fractionation is perform by differential centrifugation.

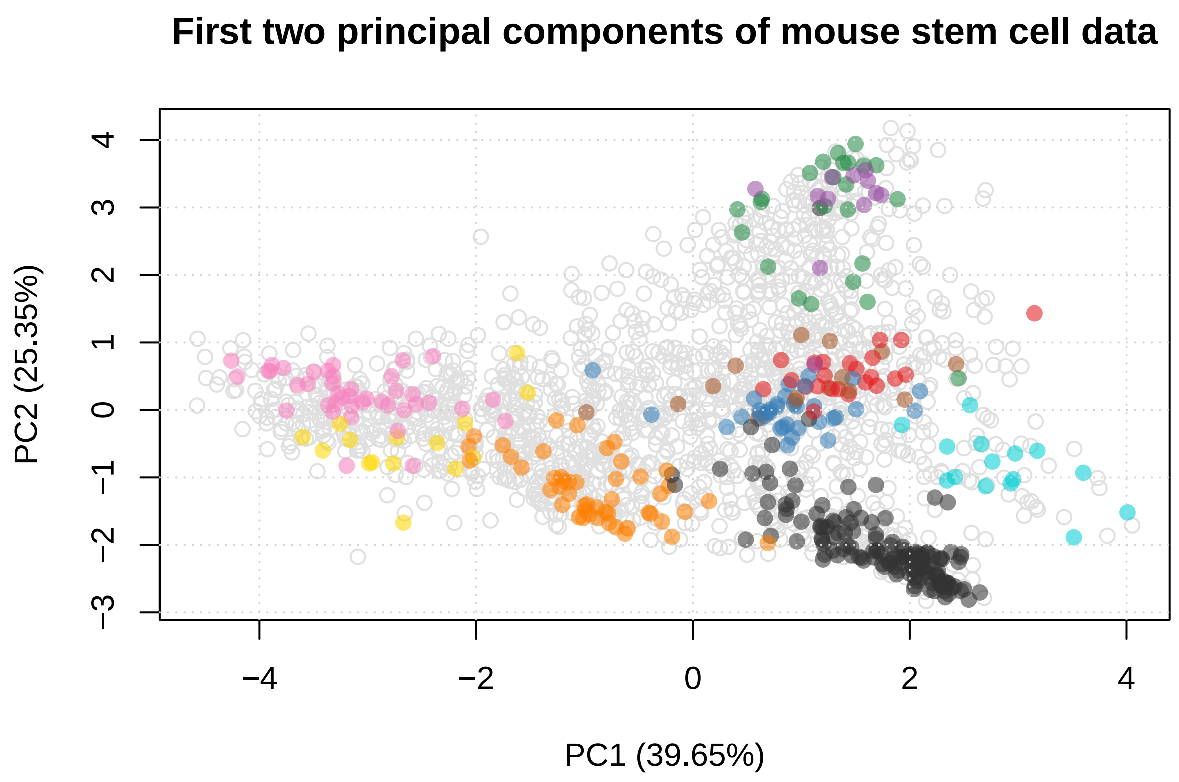
In the code chunk below, we load the aforementioned dataset. The printout demonstrates that this experiment quantified 2031 proteins over 8 fractions.

```r
data("E14TG2aR") # load experimental data
E14TG2aR
```

```
## MSnSet (storageMode: lockedEnvironment)
## assayData: 2031 features, 8 samples
##   element names: exprs
## protocolData: none
## phenoData
##   sampleNames: n113 n114 ... n121 (8 total)
##   varLabels: Fraction.information
##   varMetadata: labelDescription
## featureData
##   featureNames: Q62261 Q9JHU4 ... Q9EQ93 (2031 total)
##   fvarLabels: Uniprot.ID UniprotName ... markers (8 total)
##   fvarMetadata: labelDescription
## experimentData: use 'experimentData(object)'
## Annotation:
## - - - Processing information - - -
## Loaded on Thu Jul 16 15:02:29 2015.
## Normalised to sum of intensities.
## Added markers from  'mrk' marker vector. Thu Jul 16 15:02:29 2015
##  MSnbase version: 1.17.12
```

We can visualise the mouse stem cell dataset use the `plot2D` function. We observe that some of the organlles classes overlap and this is a typical feature of biological datasets. Thus, it is vital to perform uncertainty quantification when analysising biological data.

```
plot2D(E14TG2aR, main = "First two principal components of mouse stem cell data")
```



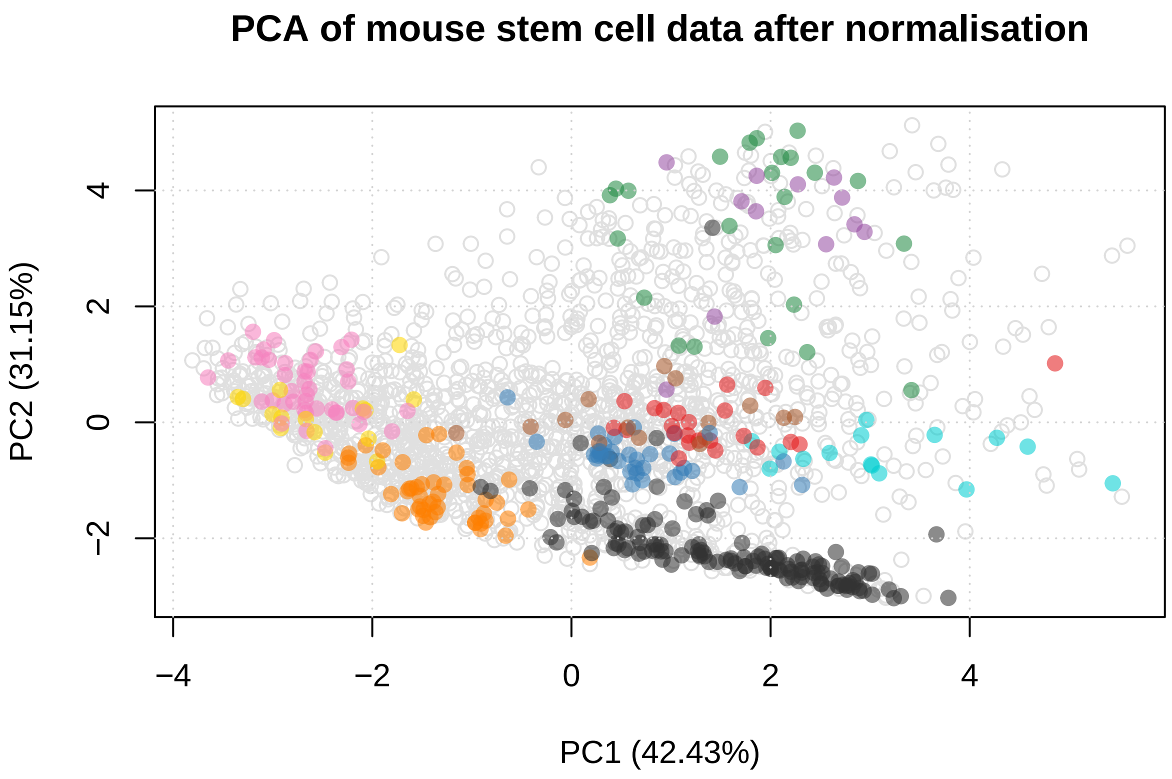
Aside: normalisation

We have found that the TAGM model sometimes fails due to floating point arithemtic errors. Error messages such at `error: chol(): decomposition failed` are indicative of this issue. Though theortically this shouldn't happen and most of the time the issue doesn't appear, it can occur. The failure can happen for a number of reasons such as proteins have almost identical profiles; highly correlated or co-linear fractions; and/or all quantitation values in a particular fraction are close to zero. We find performing variance stabilsation normalisation (vsn) can reduce the chances of numerical issues. The following code chunk demonstrates performing this normalisation within R. Though this step is not always necessary and if you experience no such issues then you should skip this step.

```r
E14TG2aR <- normalise(E14TG2aR, "vsn")
```

We can visualise the results again by using `plot2D`

```r
plot2D(E14TG2aR,
       main = "PCA of mouse stem cell data after normalisation")
```



Methods: *TAGM MAP*

Introduction to TAGM MAP

We can perform *maximum a posteriori* (MAP) estimation to perform Bayesian inference in our model. The *maximum a posteriori* estimate equals the mode of the posterior distribution and can be used to provide a point estimate summary of the posterior localisation probabilities. It does not provide samples from the posterior distribution, however an extended version of the expectation-maximisation (EM) algorithm can be used in our case, allowing fast inference. The EM algorithm is an algorithm that iterates between an expectation step and a maximisation step. This allows us to find parameters which maximise the logarithm of the posterior, in the presence of latent (unobserved) variables. The EM algorithm is guaranteed to converge to a local mode. The code chunk below excutes the `tagmMapTrain` function for a default of 100 iterations. We use the default priors for simplicity and convenience, however they can be changed, which we explain in a later section. The output is an object of class `MAPParams`.

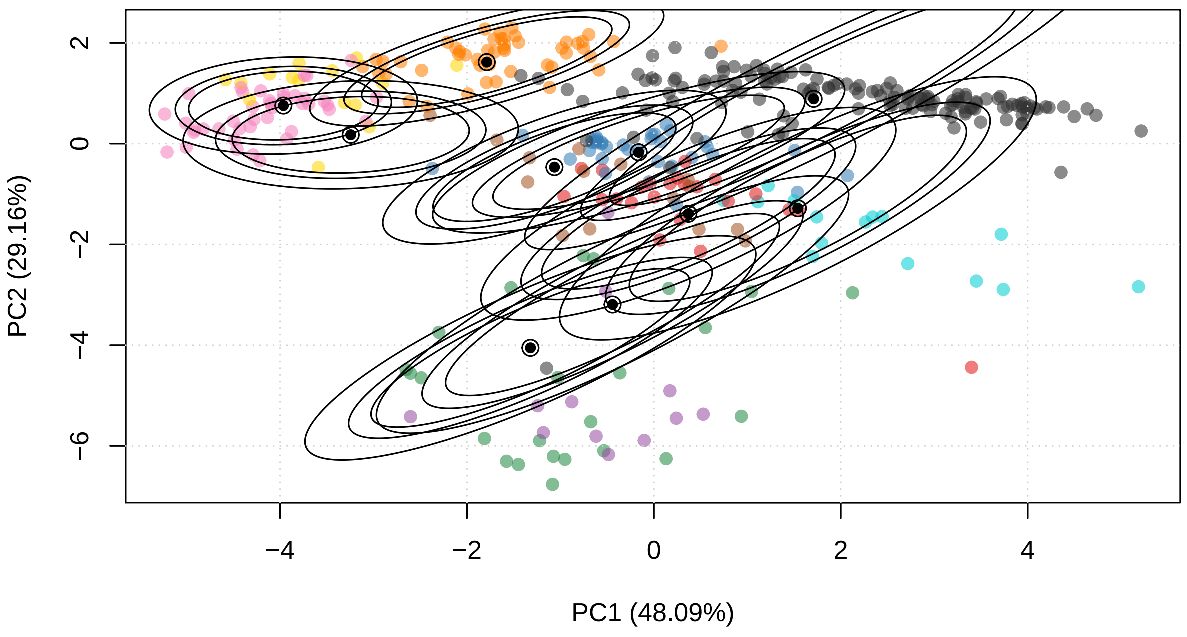
```
set.seed(2)
mapRes <- tagmMapTrain(E14TG2aR)
mapRes
```

```
## Object of class "MAPParams"
##  Method: MAP
```

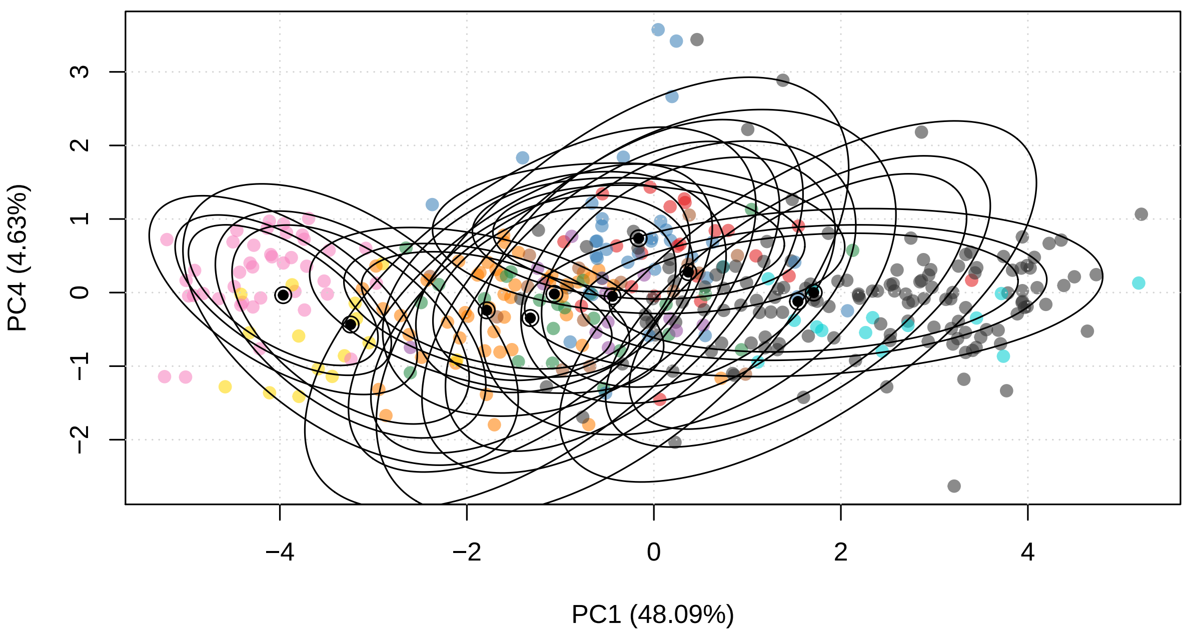
The results of the modelling can be visualised with the `plotEllipse` function. The outer ellipse contains 99% of the total probability whilst the middle and inner ellipses contain 95% and 90% of the probability respectively. The centres of the clusters are represented by black circumpunct (circled dot). We can also plot the model in other principal components. The code chunk below plots the probability ellipses along the first and second, as well as the fourth prinipal component. The user can change the components visualised by altering the `dims` argument.

```
par(mfrow = c(2, 1)) ## Create two panels
plotEllipse(E14TG2aR, mapRes, main = "PCA plot with probability ellipses")
plotEllipse(E14TG2aR,
            mapRes,
            dims = c(1, 4),
            main = "Ellipse plot along 1st and 4th prinipal components")
```

PCA plot with probability ellipses



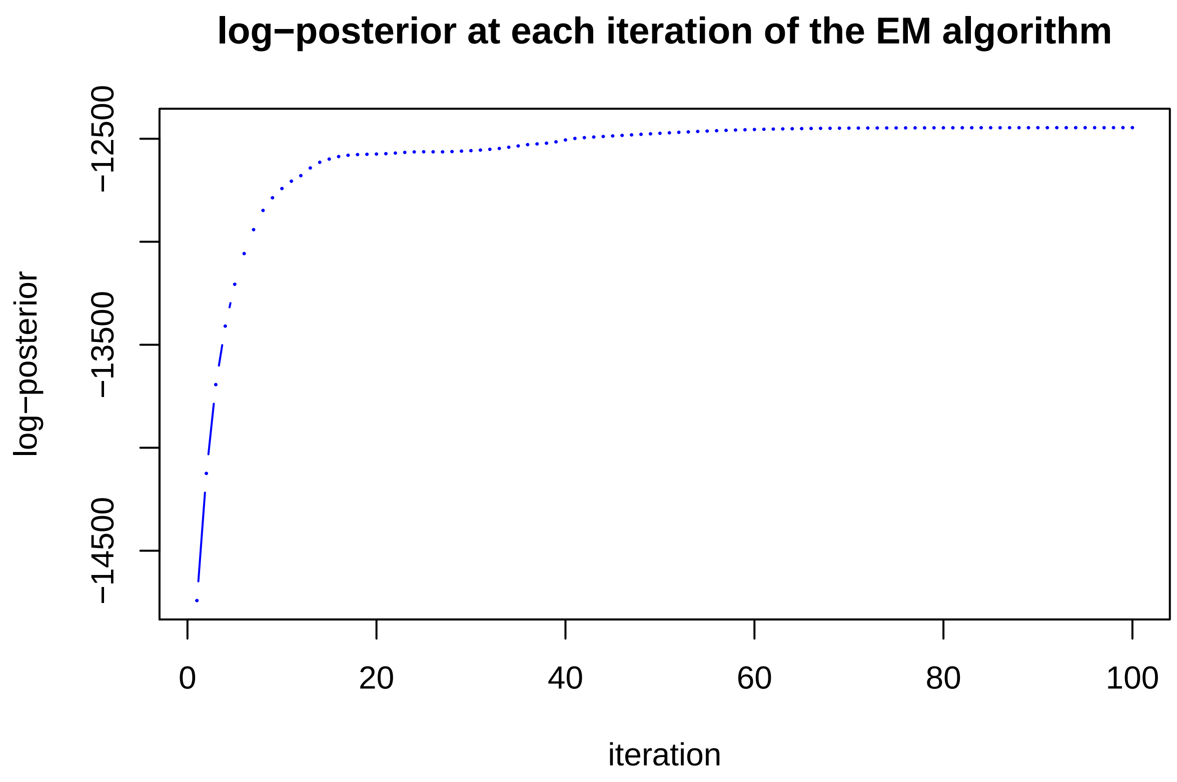
Ellipse plot along 1st and 4th prinipal components



The expectation-maximisation algorithm

The EM algorithm is iterative; that is, the algorithm iterates between an expectation step and a maximisation step until the value of the log-posterior does not change. This fact can be used to assess the convergence of the EM algoritm. The value of the log-posterior at each iteration is contained within the `posteriors` slot within the `MAPParams` object. The code chuck below plots the log posterior at each iteration and we see the algorithm rapidly plateaus and so we have acheived convergence. If convergence has not been reached during this time, increase the number of iteration by changing the parameter `numIter` in the `tagmMapTrain` method. In practice, it is not unexpected to observe small fluctations due to numerical errors and this should not be concerned users.

```
plot(mapRes@posteriors$logposterior, type = "b", col = "blue",
     cex = 0.1, ylab = "log-posterior", xlab = "iteration",
     main = "log-posterior at each iteration of the EM algorithm")
```



The code chuck below uses the `MAPParams` object to classify the proteins of unknown localisation using `tagmPredict` function. This method appends new columns to the `fData` columns of the `MSnSet`.

```
E14TG2aR <- tagmPredict(E14TG2aR, mapRes) # Predict protein localisation
```

The new feature variables that are generated are:

- `tagm.map.allocation`: the TAGM-MAP predictions for the most probable protein sub-cellular allocation.

```
table(fData(E14TG2aR)$tagm.map.allocation)
```

```
##
##           40S Ribosome           60S Ribosome                Cytosol
##                   109                        53                       179
## Endoplasmic reticulum                  Lysosome             Mitochondrion
##                   288                       157                       331
##   Nucleus - Chromatin   Nucleus - Nucleolus           Plasma membrane
##                   104                       335                       310
##            Proteasome
##                   165
```

- `tagm.map.probability`: the posterior probability for the protein sub-cellular allocations.

```r
summary(fData(E14TG2aR)$tagm.map.probability)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.9121  0.9898  0.9015  0.9998  1.0000
```

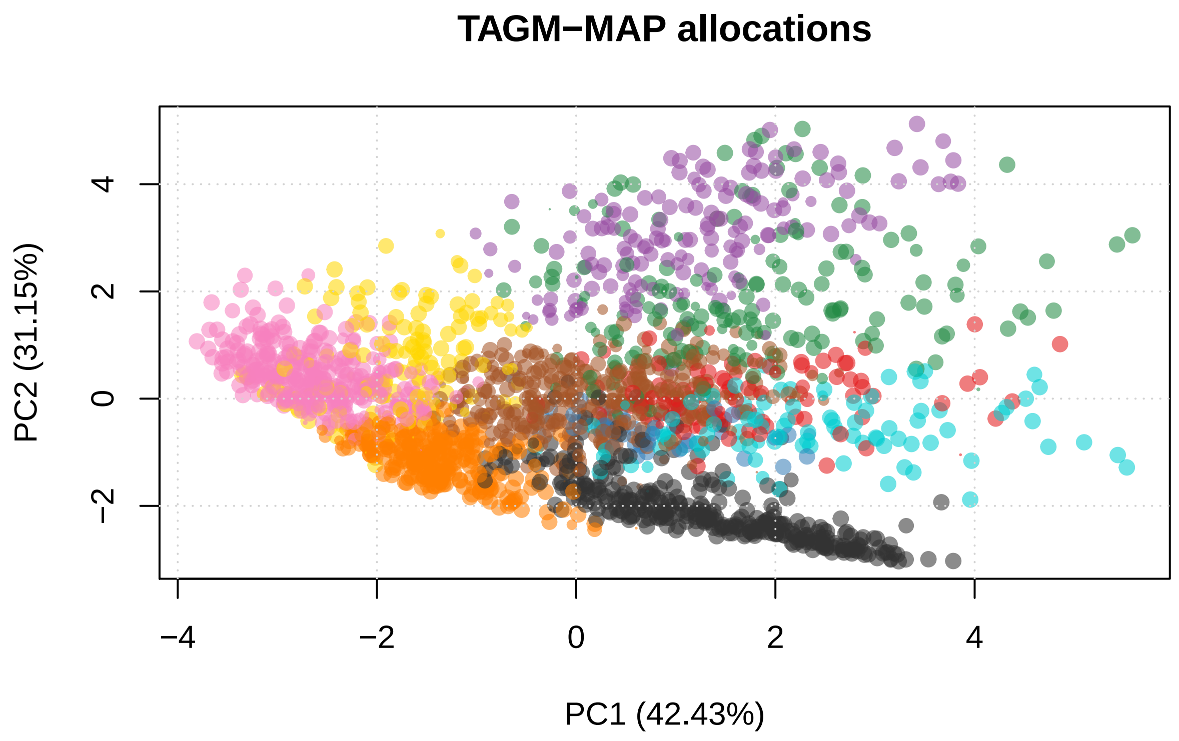
- `tagm.map.outlier`: the posterior probability for that protein to belong to the outlier component rather than any annotated component.

```r
summary(fData(E14TG2aR)$tagm.map.outlier)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.000000 0.000105 0.001827 0.042451 0.008715 1.000000
```

We can visualise the results by scaling the pointer according the posterior localisation probabilities. To do this we extract the MAP localisation probabilities from the feature columns of the the `MSnSet` and pass these to the `plot2D` function.

```r
ptsze <- fData(E14TG2aR)$tagm.map.probability # Scale pointer size

plot2D(E14TG2aR, fcol = "tagm.map.allocation", cex = ptsze,
       main = "TAGM-MAP allocations")
```



The TAGM MAP method is easy to use and it is simple to check convergence, however it is limited in that it can only provide point estimates of the posterior localisation distributions. To obtain the full posterior distributions and therefore a rich analysis of the data, we resort to using Markov-Chain Monte-Carlo methods. In our particular case, we use a so-called collapsed Gibbs sampler.

Methods: *TAGM MCMC* a brief overview

The TAGM MCMC method allows a fully Bayesian analysis of spatial proteomics datasets. It employs a collapsed Gibbs sampler to obtain samples from the posterior distribution of localisation probablities, providing a rich analysis of the data. This section demonstrates the advantage of taking a Bayesian approach and the biological information that can be extracted from this analysis.

Since our audience is unlikely to be versed in Bayesian methodology, we explain some of the key ideas for a more complete understanding. Firstly, MCMC-based inference constrasts with MAP based inference in that in produces *samples* from the posterior distribution of localisation probabilities. Hence, we do not just have a single estimate for each quantity but a distribution (or histogram) of estimates. MCMC methods are a large class of algorithms used to sample from a probability distribution, in our case the posterior distribution of the parameters. They design a Markov-chain; that is, a random sequence of events where the probability of the next event only depends on the current state, which, after convergence obtains samples form the posterior distribution. A specific example of an MCMC algorithm is the Gibbs sampler, which can be applied when the parameters are conditionally conjugate. Often one can perform Rao-Blackwellisation, a method to reduce posterior variance, to obtain a collapsed Gibbs sampler. Once one has obtained samples from the posterior distribution, we can estimate the true mean of the posterior distribution by simply taking the mean of the samples. In a similar fashion, we can obtain other summaries of the posterior distribution.

The TAGM MCMC method is computationally intensive and requires at least modest processing power. Leaving the MCMC algorithm to run overnight on a modern desktop is usually sufficient, however this, of course, depends on the exact system properties. Do note expect the analysis to finish in a couple of hours on a medium specification laptop, for example.

To demonstrate the class structure and expected outputs of the TAGM MCMC method, we run a brief analysis on the a subsest of the `tan2009r1` dataset from the `pRolocdata` purely for illustration. This is to provide a bare bones analysis of these data without being held back by computational requirements. We perform a complete demonstration and provide precise details of the analysis of the stem cell dataset considered above in the next section.

```r
set.seed(1)
data(tan2009r1)
tan2009r1 <- tan2009r1[sample(nrow(tan2009r1), 400), ]
```

The first step is run two MCMC chains for a few iterations of the algorithm using the `tagmMcmcTrain` function. This function will generate a object of class `MCMCParams`. The summary slot of which is currently empty.

```r
library("pRoloc")
p <- tagmMcmcTrain(object = tan2009r1, numIter = 3,
                   burnin = 1, thin = 1, numChains = 2)
p
```

```
## Object of class "MCMCParams"
## Method: TAGM.MCMC
## Number of chains: 2
```

Information for each MCMC chain is contained within the chains slot. If needed, this information can be accessed manually. The function `MCMCProcess` populates the summary slot of the `MCMCParams` object

```
p <- tagmMcmcProcess(p)
p
```

```
## Object of class "MCMCParams"
## Method: TAGM.MCMC
## Number of chains: 2
## Summary available
```

The summary slot has now been populated to include basic summaries of the `MCMCChains`, such as organelle allocations and localisation probabilities. Protein information can be appended to the feature columns of the `MSnSet` by using the `tagmPredict` function, which extracts the required information from the summary slot of the `MCMCParams` object.

```
res <- tagmPredict(object = tan2009r1, params = p)
```

One can now access new features variables:

- `tagm.mcmc.allocation`: the TAGM-MCMC prediction for the most likely protein sub-cellular annotation.

```
table(fData(res)$tagm.mcmc.allocation)
```

```
##
##   Cytoskeleton             ER          Golgi       Lysosome mitochondrion
##             12             98             22              9            39
##        Nucleus     Peroxisome             PM     Proteasome   Ribosome 40S
##             25              3            101             31            31
##   Ribosome 60S
##             29
```

- `tagm.mcmc.probability`: the mean posterior probability for the protein sub-cellular allocations.

```
summary(fData(res)$tagm.mcmc.probability)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.3422  0.8950  0.9879  0.9086  1.0000  1.0000
```

As well as other useful summaries of the MCMC methods:

- `tagm.mcmc.outlier` the posterior probability for the protein to belong to the outlier component.
- `tagm.mcmc.probability.lowerquantile` and `tagm.mcmc.probability.upperquantile` are the lower and upper boundaries to the equi-tailed 95% credible interval of `tagm.mcmc.probability`.
- `tagm.mcmc.mean.shannon` a Monte-Carlo averaged shannon entropy, which is a measure of uncertainty in the allocations.

Methods: *TAGM MCMC* the details

```
load("C:/Users/OllyC/Desktop/TAGMworkflow/tagmE14.rda")
```

This section explain how to manually manipulate the MCMC output of the TAGM model. The data file 'tagmE14.rda' is available online and is not directly loaded into this package for size. The file itself if around 500mb, which is too large to directly load into a package. The following code, which is not evaluated, was used to produce the `tagmE14` MCMCParams object. We run the MCMC algorithm for 20000 iterations with 10000 iterations discarded for burnin. We then thin the chain by 20. We ran 6 chains in parallel and so we obtain 500 samples for each of the 6 chains, totalling 3000 samples.

```r
tagmE14 <- tagmMcmcTrain(E14TG2aR2,
                         numIter = 20000,
                         burnin = 10000,
                         thin = 20,
                         numChains = 6)
```

Manually inspecting the object we see that it is a `MCMCParams` object with 6 chains.

```r
tagmE14
```

```
## Object of class "MCMCParams"
## Method: TAGM.MCMC
## Number of chains: 6
```

Data exploration and convergence diagnostics

Assessing whether or not an MCMC algorithm has converged is challenging. Assessing and diagnosing convergence is an active area of research and throughout the '90s many approaches were proposed, (see . . .). Converged MCMC algorithm should be oscillating rapidly around a single value with no monotonicity. We provide a more detailed exploration of this issue, but the readers should bare in mind that the methods provided below are diagnostics and cannot guarantee success. We direct readers to several important works in the literature discussing the assesment of convergence. Users that do not assess convergence and base their downstream analysis on unconverged chains are likely to obtain poor quality results.

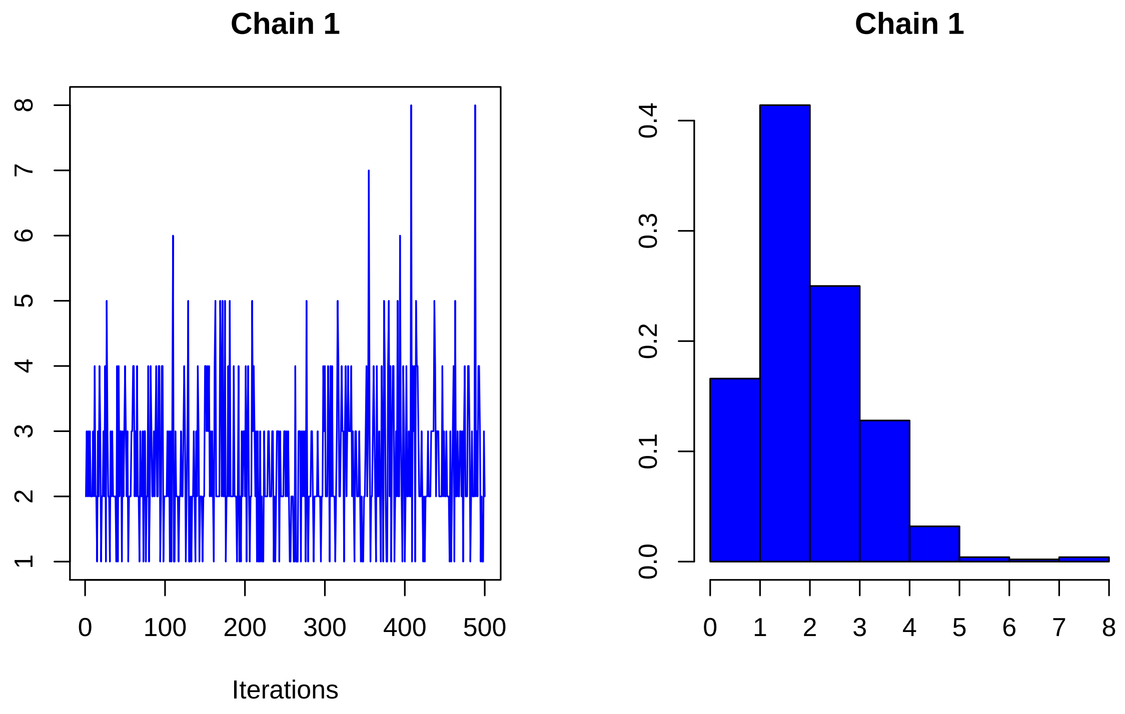
We first assess convergence using a parallel chains approach. We find producing multiple chains is benifical not only for computational advantages but also for analysis of convergence of our chains.

```r
## Get number of chains
nChains <- length(tagmE14)
nChains
```

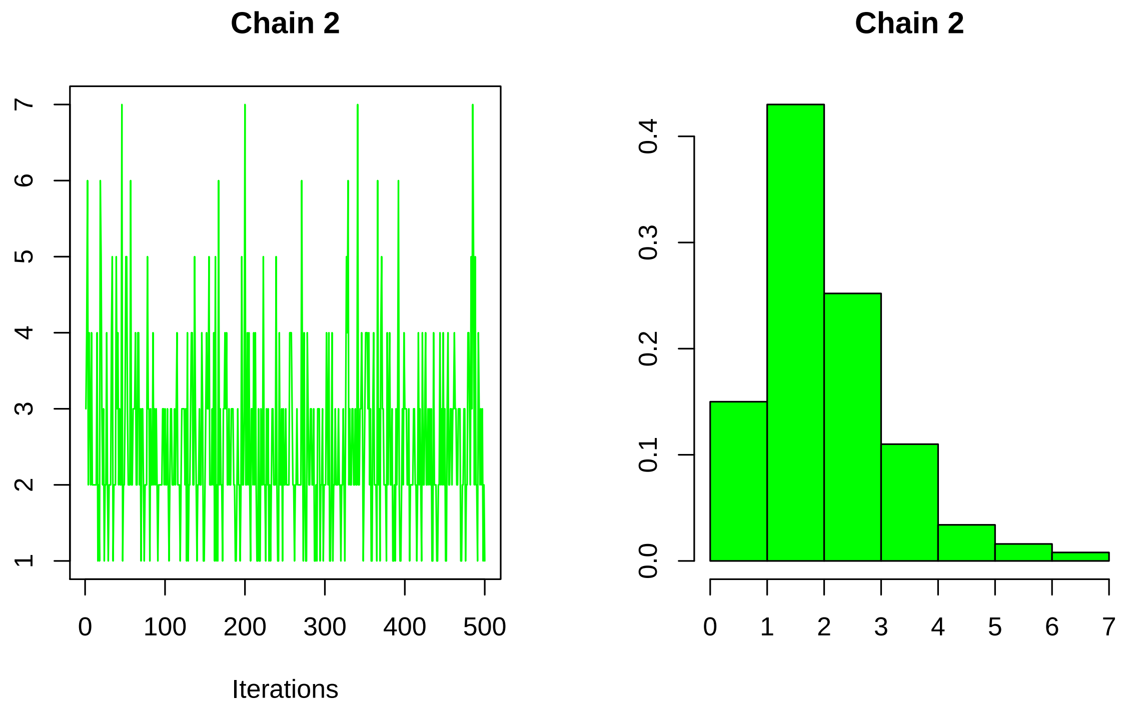
```
## [1] 6
```

The following code chunks sets up a manual convegence diagnostic check. We make use of objects and methods in the package *coda* to peform this analysis [@coda]. Our function below automatically coerces our objects into *coda* for ease of analysis. We calculate the total number of outliers at each iteration of each chain and if the algorithm has converged this number should be the same (or very similar) across all 6 chains. We can observe this from the trace plots and histrograms for each MCMC chain. Unconverged chains are discharded from downstream analysis.

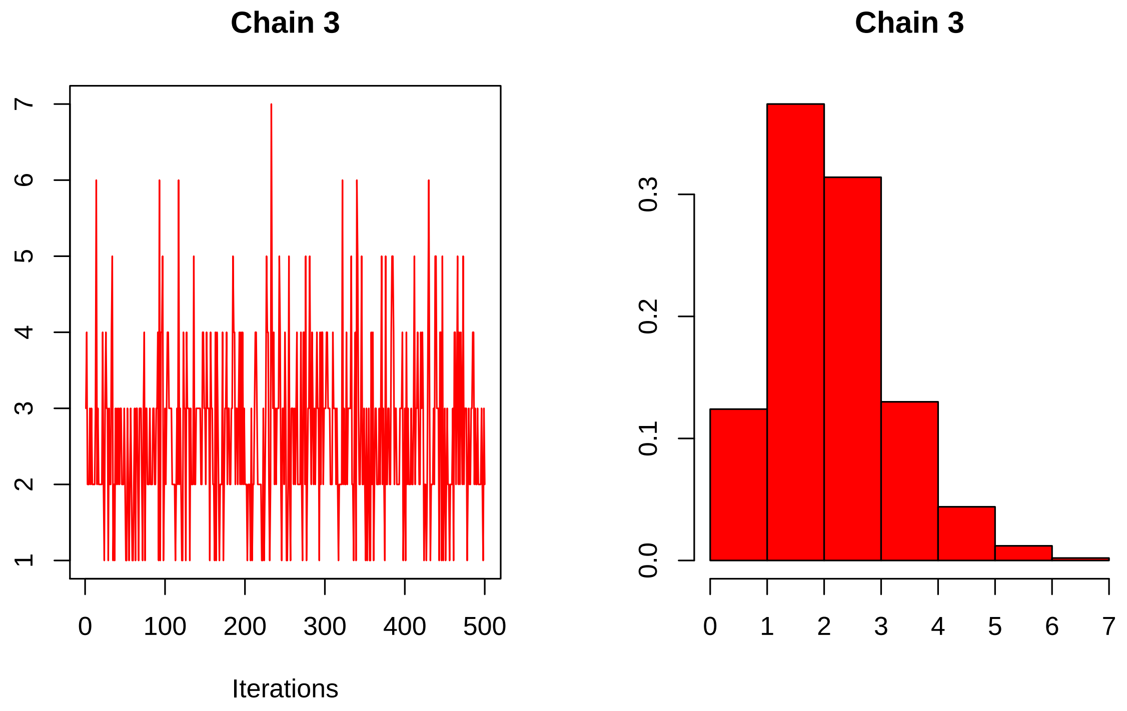
```r
## Convergence diagnostic to see if more we need to discard any
## iterations or entire chains: compute the number of outliers for
## each iteration for each chain
out <- mcmc_get_outliers(tagmE14)

## Using coda S3 objects to produce trace plots and histograms
plot(out[[1]], col = "blue",   main = "Chain 1")
```



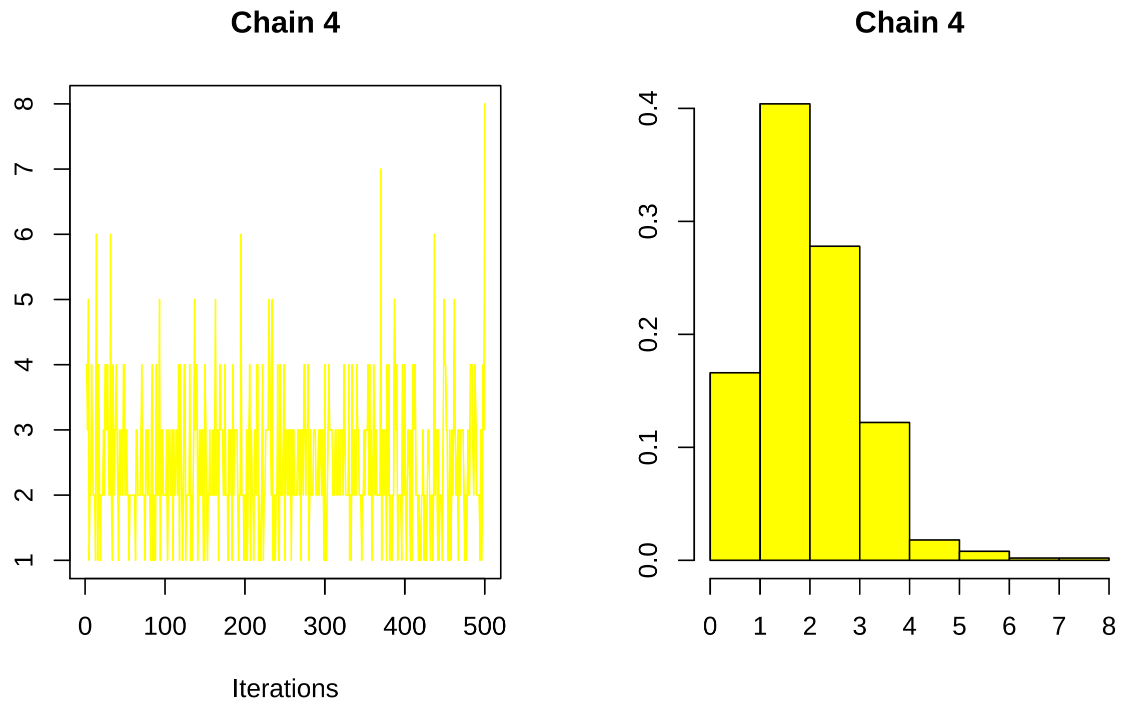
```r
plot(out[[2]], col = "green",  main = "Chain 2")
```



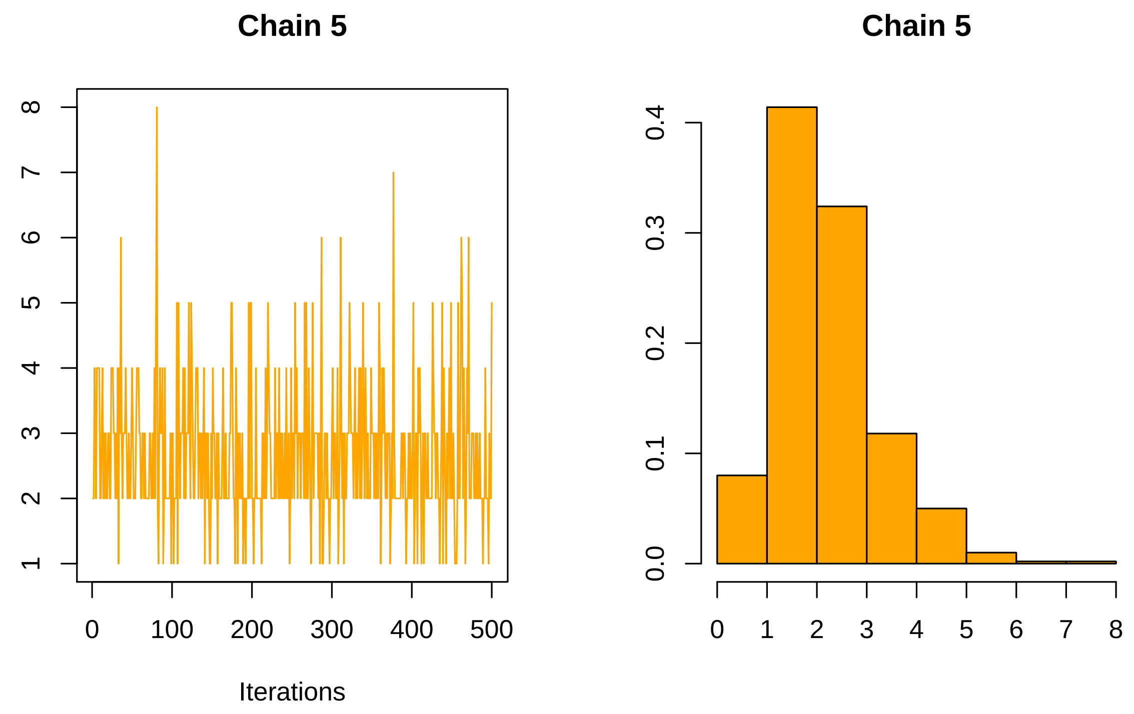
```r
plot(out[[3]], col = "red",     main = "Chain 3")
```



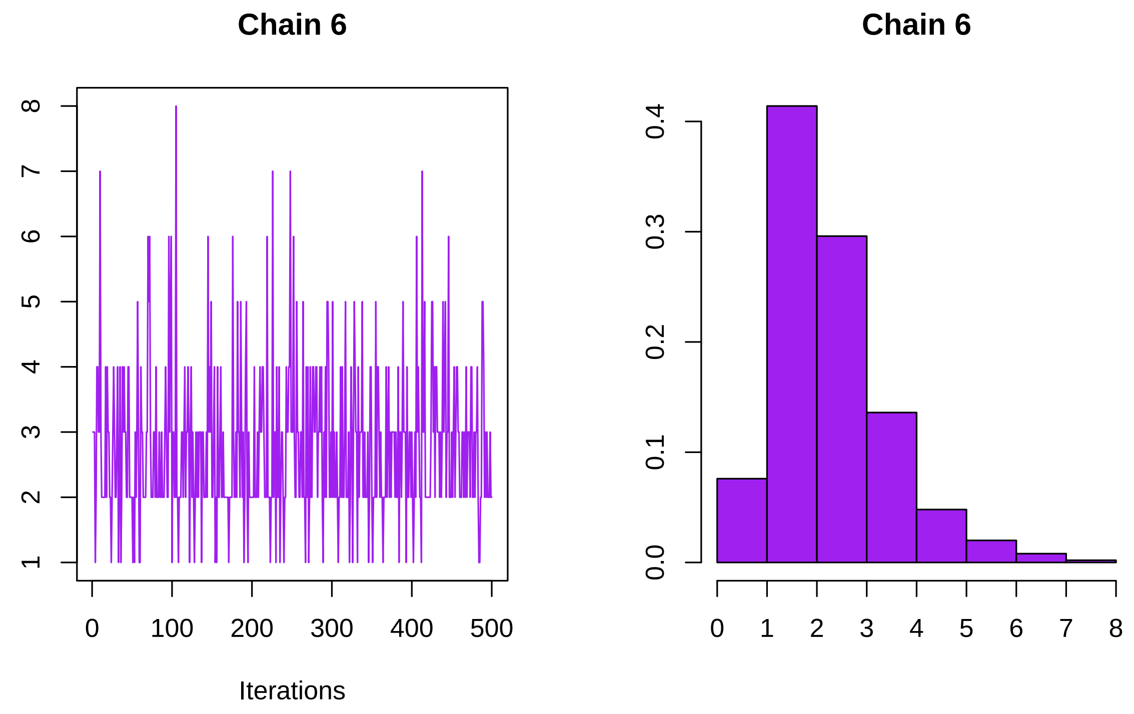
```r
plot(out[[4]], col = "yellow", main = "Chain 4")
```



```r
plot(out[[5]], col = "orange", main = "Chain 5")
```



```r
plot(out[[6]], col = "purple", main = "Chain 6")
```



All of the chains are are oscillating around 2.5 and demonstrate similar structure. This is indicative of convergence. We can use the *coda* package to produce summaries of our chains. Here is the `coda` summary for the first chain.

```r
## all chains average around 2.5 outliers
summary(out[[1]])
```

```
##
## Iterations = 1:500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##        2.48600        1.11368        0.04981        0.04981
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
##     1     2     2     3     5
```

Applying the Gelman diagnostic

Thus far, our analysis appears promising. Each chain oscillate around an average of 2.5 outliers and there is no observed monotonicity in our output. However, for a more rigorous and unbiased analysis of convergence we can calculate the Gelman diagnostics using the *coda* package [@Gelman:1992,@Brooks:1998]. This statistics is often refered to as $\hat{R}$ or the potential scale reduction factor. The idea of the Gelman diagnostics is to compare the inter and intra chain variances. The ratio of these quantities should be close to one. The actual statistics computed is more complicated, but we do not go deeper here and a more detailed and in depth discussion can be found in the references. The *coda* package also reports the 95% upper confidence interval of the $\hat{R}$ statistic. In this case ,our samples are not normally distributed. The *coda* package allows for transformations to improve normality of the data, in our case a log tranform is performed. The original paper (cite) suggests that chains with $\hat{R}$ value of less than 1.2 are likely to have converged.

```r
## We can check Gelman diagnostic for convergence
## (values less than <1.2  are good for convergence)
gelman.diag(out, transform = TRUE) ## the Upper C.I. is 1.03 so mcmc has likely converged
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.01       1.03
```

We can also look at the Gelman diagnostics statistics for groups or pairs of chains. The first line below compute the Gelman diagnostic across the first three chains, whereas the second calculates between chain 2 and chain 5.

```r
## We can also check individual pairs of chains for convergence
gelman.diag(out[1:3], transform = TRUE) # the upper C.I is 1.02
```

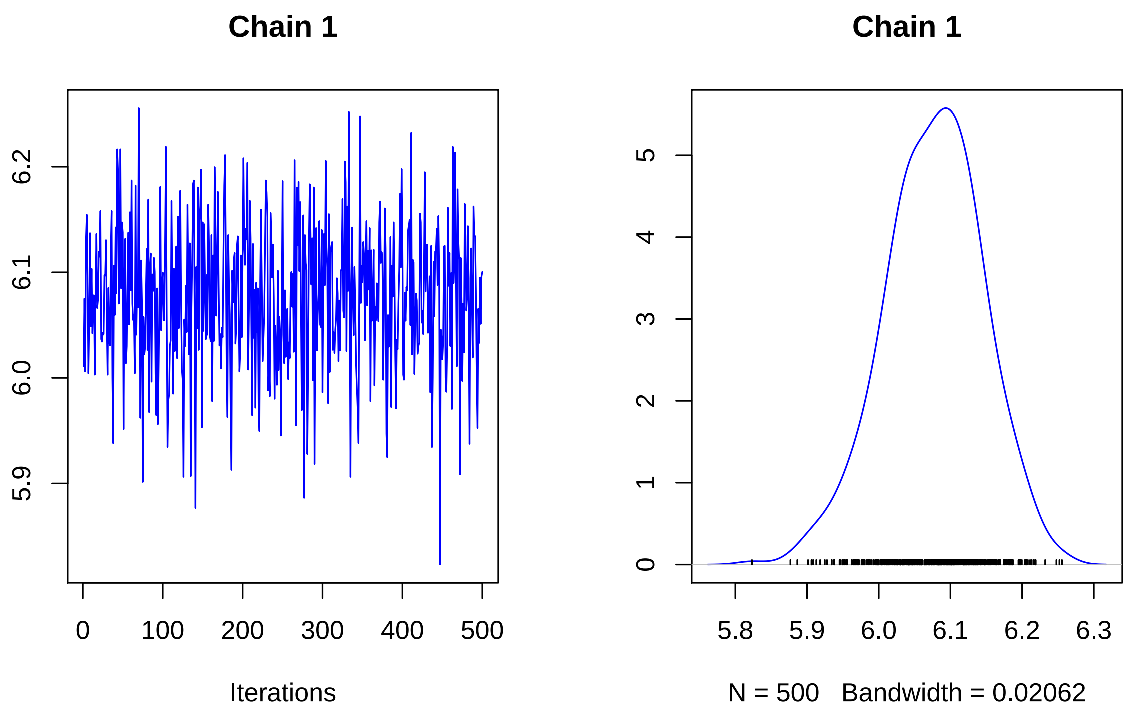
```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.01       1.02
```

```r
gelman.diag(out[c(2,5)], transform = TRUE) # the upper C.I is 1.08
```

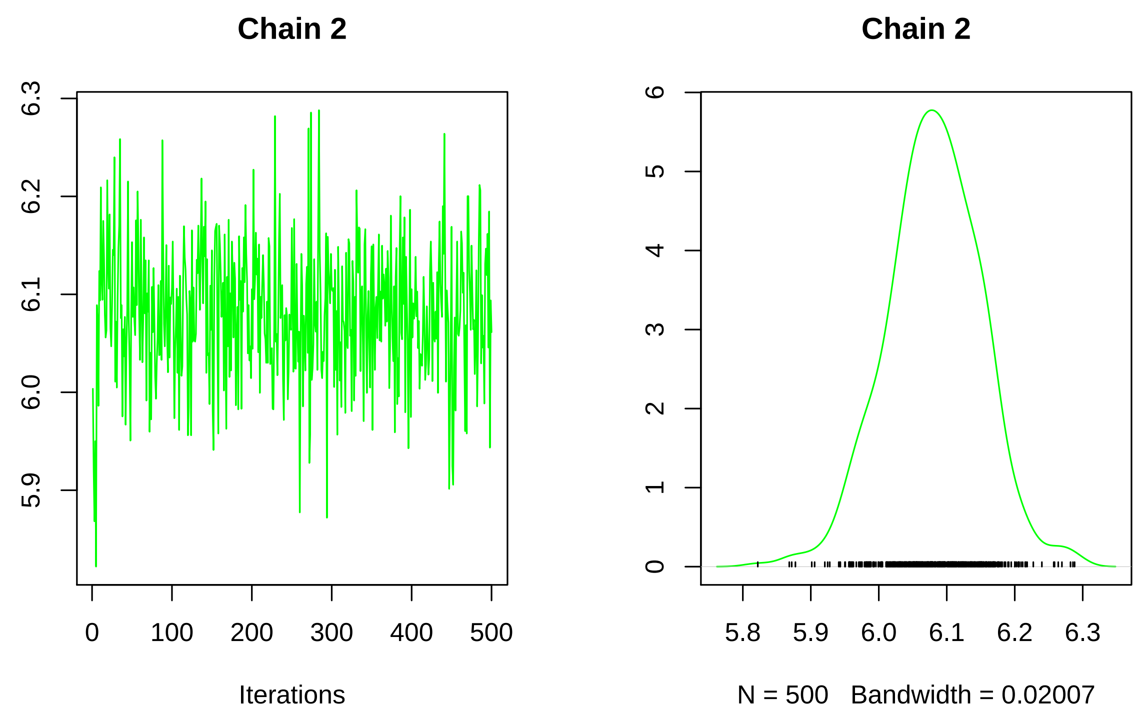
```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.02       1.08
```

To assess another summary statistics, we can look at the mean component allocation at each iteration of the MCMC algorithm and as before we produce trace plots of this quantity.

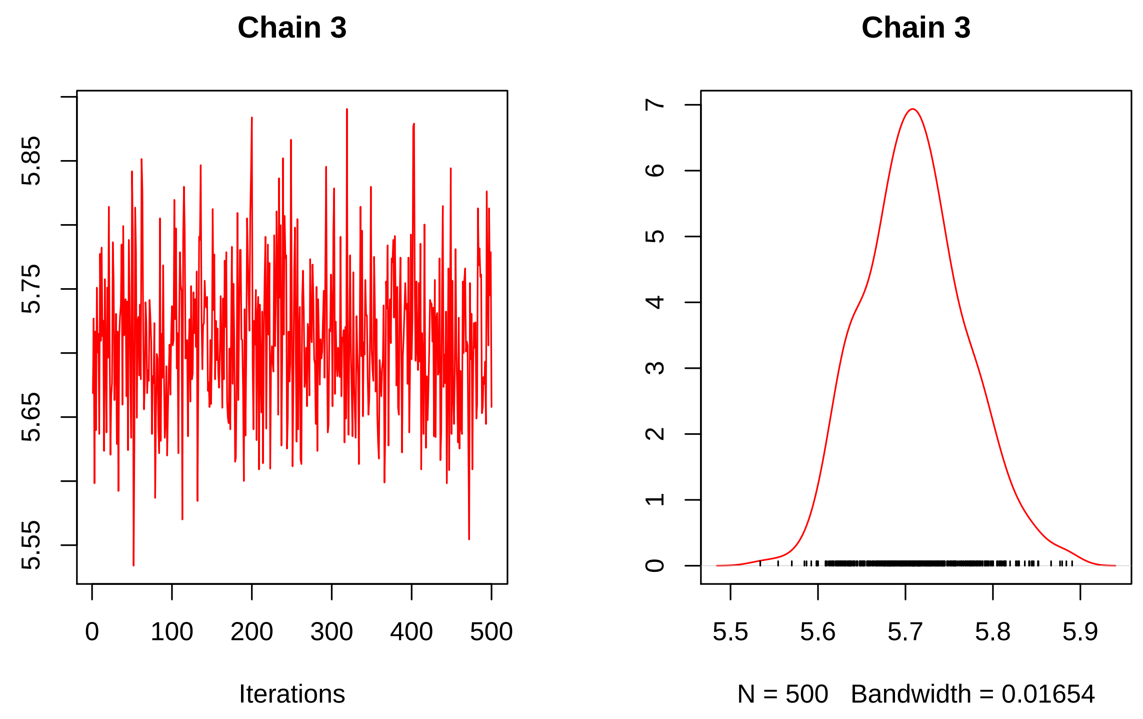
```r
# Compute the mean component allocation at each mcmc iterations
meanAlloc <- mcmc_get_meanComponent(tagmE14)


plot(meanAlloc[[1]], col = "blue",   main = "Chain 1")
```



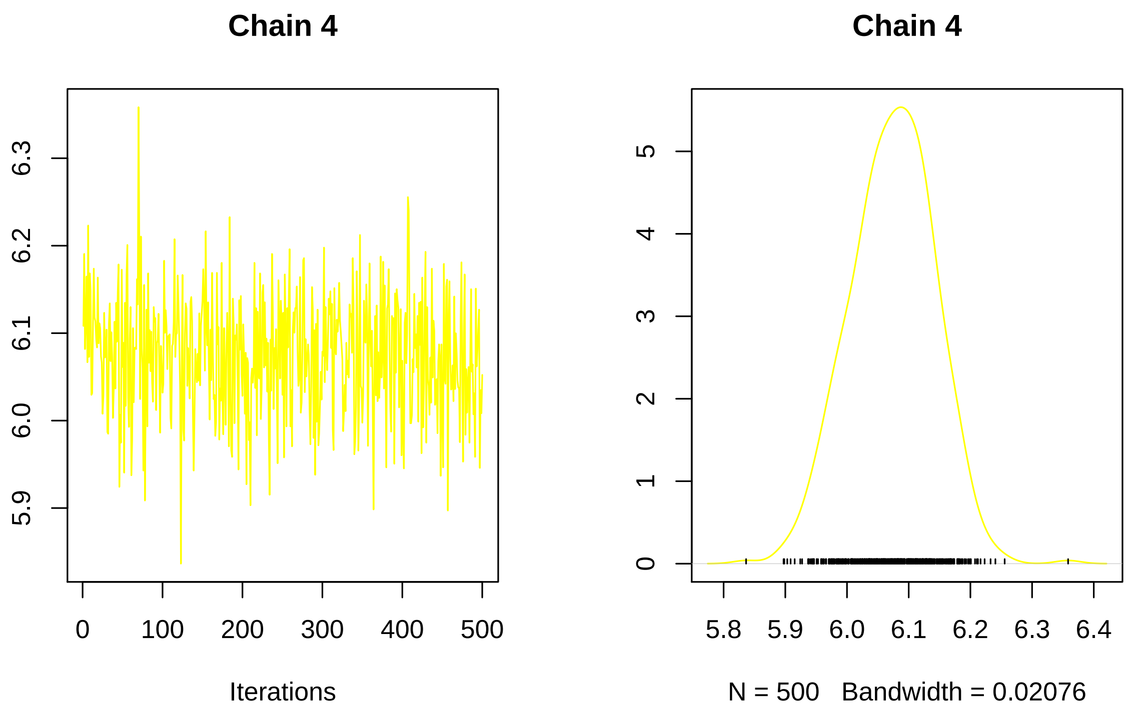
```r
plot(meanAlloc[[2]], col = "green",  main = "Chain 2")
```



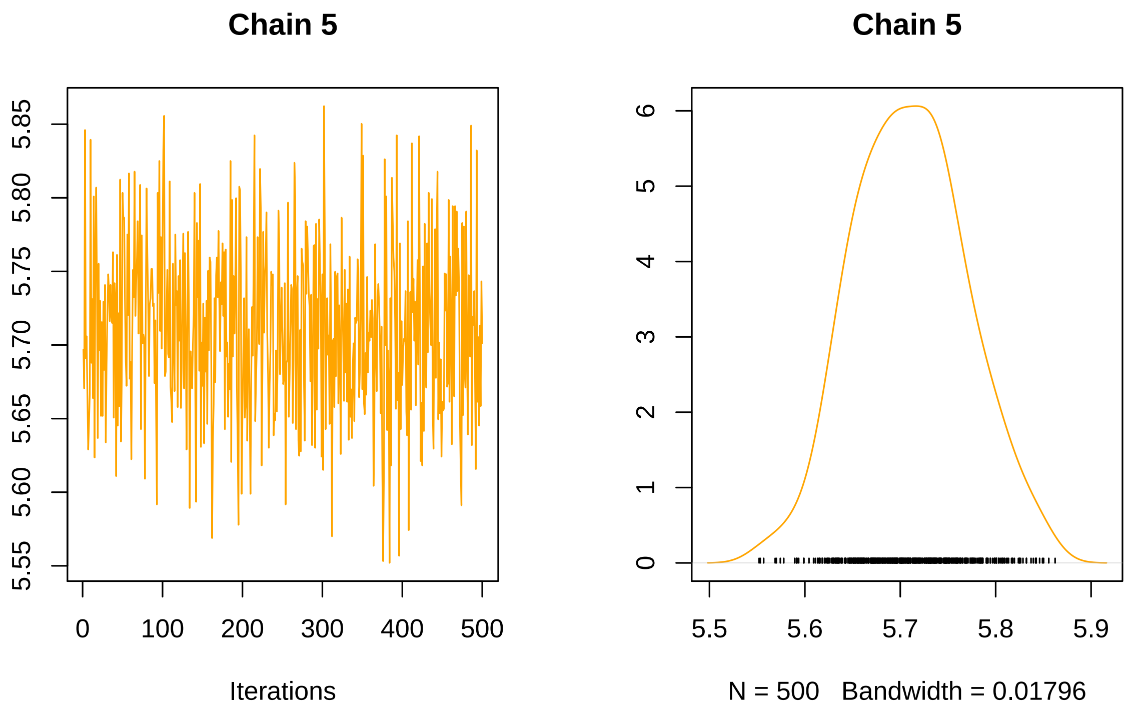
```
plot(meanAlloc[[3]], col = "red",     main = "Chain 3")
```



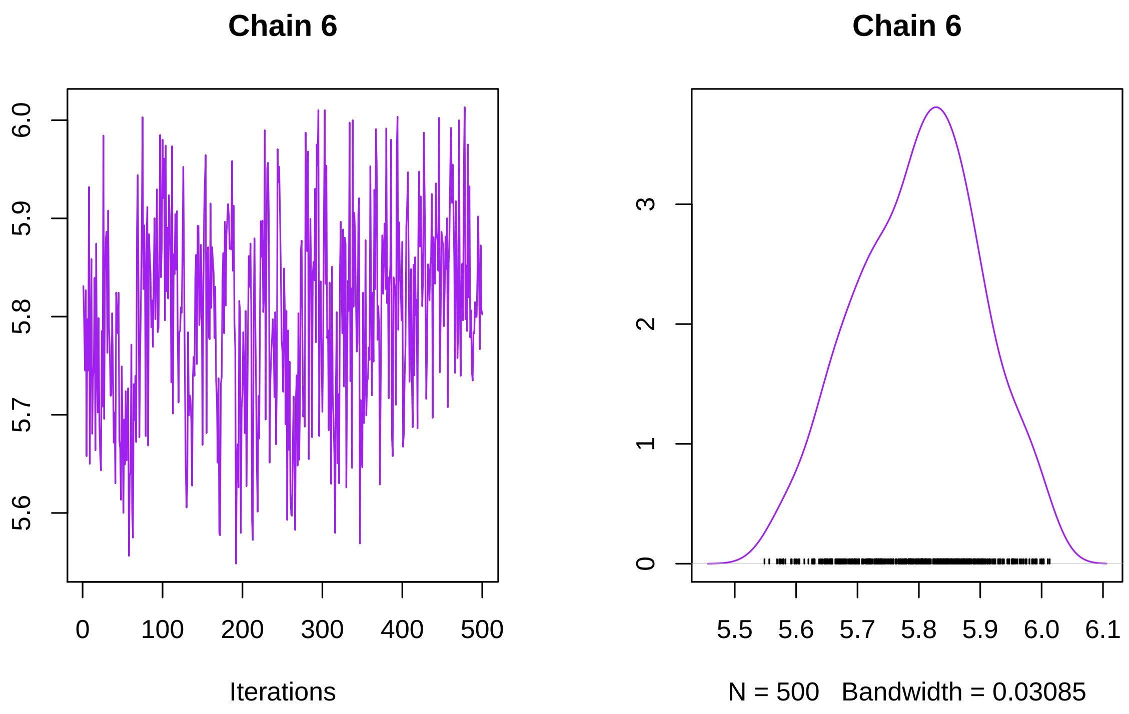
```r
plot(meanAlloc[[4]], col = "yellow", main = "Chain 4")
```



```r
plot(meanAlloc[[5]], col = "orange", main = "Chain 5")
```



```r
plot(meanAlloc[[6]], col = "purple", main = "Chain 6")
```



As before we can produce summaries of the data.

```r
summary(meanAlloc[[1]])
```

```
##
## Iterations = 1:500
## Thinning interval = 1
## Number of chains = 1
## Sample size per chain = 500
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##           Mean             SD       Naive SE Time-series SE
##       6.076683       0.068538       0.003065       0.003271
##
## 2. Quantiles for each variable:
##
##  2.5%   25%   50%   75% 97.5%
## 5.934 6.033 6.081 6.123 6.205
```

We can already observe that there are difference between these chains and they oscillate around slightly different values, this raises suspicion that some of the chains may not have converged. For example chains $1, 2$ and 4 appear to oscillate around a mean value of 6.1. Chains 3 and 5 appear to increase and decrease rather than rapidly osclate. Chain 5 seems to be oscillating around 5.7, which is not in consensus with the other chains. For a more quantitaive analysis, we again apply the Gelamn diagnostics to these summaries.

```r
gelman.diag(meanAlloc)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       3.38       5.46
```

The above values are quite distant from 1 and clearly greater than 1.2, therefore we should believe these chains have not converged. As observed previously, chains $3, 5, 6$ look quite different from the other chains and so we recalculate the diagnostic excluding these chains. The computed Gelman diagnostic below suggest that chains $1, 2$ and 4 have converged and that we should discard chains $3, 4$ and 6 from further analysis.

```
gelman.diag(meanAlloc[c(1,2,4)])
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1       1.01
```

For a further check, we can look at the mean outlier probability at each iteration of the MCMC algorithm and again computing the Gelman diagnostics between chains $1, 2$ and 4. An $\hat{R}$ statistics of 1 is indicative of convergence, since it is less than the recommend value of 1.2.

```
meanoutProb <- mcmc_get_meanoutliersProb(tagmE14)
gelman.diag(meanoutProb[c(1,2,4)])
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

Applying the Geweke diagnostic

Along with the Gelman diagnostics, which uses parallel chains, we can also apply a single chain analysis using the Geweke diagnostic. The Geweke diagnostic tests to see whether the mean calculate from the first 10% of iterations is significantly different from the the mean calculated from the last 50% of iterations. If they are significantly different, at say a level 0.01, then this is evidence that particular chains has not converged. The following code chunk calculates the Geweke diagnostic for each chain on the summarising quantities we have previously computed.

```
geweke_test(out)
```

```
##             chain 1   chain 2     chain 3    chain 4   chain 5    chain 6
## z.value -0.9201585 1.2916455 -2.43855877 1.69642764 1.6253508 -0.6003274
## p.value  0.3574900 0.1964799  0.01474596 0.08980492 0.1040878  0.5482881
```

```
geweke_test(meanAlloc)
```

```
##            chain 1     chain 2   chain 3    chain 4   chain 5      chain 6
## z.value 0.4307900 -0.08817564 0.1194190 1.82904303 0.7542722 -3.197185476
## p.value 0.6666211  0.92973708 0.9049434 0.06739316 0.4506858  0.001387757
```

```
geweke_test(meanoutProb)
```

```
##             chain 1   chain 2    chain 3   chain 4   chain 5    chain 6
## z.value -0.8845583 1.6385473 -0.8954649 1.0480938 0.6630499 -1.0246155
## p.value  0.3763949 0.1013076  0.3705386 0.2945954 0.5072986  0.3055447
```

The first test suggest chain 3 has not converged, since the p-value is roughly 0.01 suggesting that the mean in the first 10% of iterations is significantly different from those in the final 50%. Moreover, the second test suggests that chain 6 has not converged, as can be seen from a p-value close to 0.001, supporting our earlier beliefs that these chains have not conveged. These convergence diagnostics are not limited to the quantities we have computed here and further diagnostics can be perform on any summary of the data.

An important question to consider is whether removing an early portion of the chain might lead to improvment of the convergence diagonistics. This my particularly relevant if a chain converges some iterations after our orginally specified `burin`. For example let us take the first Geweke test above, which suggested chain 3 had not converged and see if discarding the initial 10% of the chain improves the statistic. The function below removes 50 samples , informaly known as burning, from the beginning of each chain and the output shows that we now have 450 samples in each chain.

```
burntagmE14 <- mcmc_burn_chains(tagmE14, 50)
burntagmE14@chains@chains
```

```
## [[1]]
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 450
##
## [[2]]
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 450
##
## [[3]]
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 450
##
## [[4]]
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 450
##
## [[5]]
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 450
##
## [[6]]
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 450
```

The following function recomputes the number of outliers in each chain at each iteration of each Markov-chain.

```
newout <- mcmc_get_outliers(burntagmE14)
```

The code chuck below compute the Geweke diagonstic for this new truncated chain and demonstrates that chain 3 has an improved Geweke diagnostic. Thus, in practice, it maybe useful to remove iterations from the beginning of the chain. However, as chain 3 did not pass the Gelman diagnostics we still discard it from downstream analysis.

```
geweke_test(newout)
```

```
##             chain 1   chain 2    chain 3     chain 4   chain 5    chain 6
## z.value -0.4375964 0.7167981 -1.8993795 -1.77905509 0.4345828 -0.2522779
## p.value  0.6616789 0.4734987  0.0575146  0.07523073 0.6638653  0.8008262
```

Processing converged chains

Having made an assessment of convergence, we decide to discard chains $3, 5$ and 6 from any further analysis. The code chunk below remove these chains and creates and new object to store the converged chains.

```
removeChain <- c(3, 5, 6) # The chains to be removed
tagmE14_converged <- tagmE14[seq_len(nChains)[-removeChain]] # Create new object
```

The `MCMCParams` object can be large and therefore if we have a large number of samples we may want to subsample our chain, informally known as thinning to reduce the number of samples. Thinning also has another purpose. We may desire indepedent samples from our posterior distribution but the MCMC algorithm produces autocorrelated samples. Thinning can be applied to reduce the autocorrelation between samples. The code chuck below, which is not evaluate demonstrates, retaining every 5^{th} iteration. Recall that we we thinned by 20 when we first ran the MCMC algorithm.

```
tagmE14_converged_thinned <- mcmc_thin_chains(tagmE14_converged, freq  = 5)
```

We initially ran 6 chains and after having made an assesssment of convergence we decided to discard 3 of the chains. We desire to make inference using sample from all 3 chains, since this leads to better posterior estimates. In their current class strucutre all the chains are stored separately, so the following function pools all sample for all chains together to make a single longer chain with all samplers. Pooling a mixture of converged and unconverged chains in likely to lead to poor quality results.

```
tagmE14_converged_pooled <- mcmc_pool_chains(tagmE14_converged)
tagmE14_converged_pooled
```

```
## Object of class "MCMCParams"
## Method: TAGM.MCMC
## Number of chains: 1
```

```
tagmE14_converged_pooled[[1]]
```

```
## Object of class "MCMCChain"
##  Number of components: 10
##  Number of proteins: 1663
##  Number of iterations: 1500
```

Analysis, visualisation and interpreting results