# Pseudoresiduals of Multivariate Guassian Hidden Markov Models

Tazman Libson

2024-03-04

## Methods:

### Example Data Set: 4 Tech Stocks

To demonstrate the methods described below, daily returns data for 4 companies: Apple, Microsoft, Meta, and Intel between Feb 1st 2019 and Feb 1st 2024 were used. For the data, the returns are defined as $100 \log(s_t/s_{t-1})$, where $s_t$ is the price on day $t$. This data was downloaded from https://www.nasdaq.com/market-activity/stocks/ on Feb 1st, 2024.

### Introduction to HMMs:

Hidden Markov models are a stochastic model with a wide range of applications. They are most often used with either time series or space series data. First there is an unobserved process described by a Markov chain. This chain will have m states. The state at time $t$ will be written as $c_t$. These states are discrete. The transitions between states are described by a one step transition probability matrix.

$$\Gamma = [p_{ij}]; \quad p_{ij} = \mathbb{P}[c_t = j | c_{t-1} = i]$$

For all our cases, the markov chains will all be homogeneous, meaning that $\Gamma$ is independent of time. In addition to $\Gamma$, the initial distribution needs to be specified. There are two options, either an initial distribution can be directly given, or one can assume the chain is stationary. The stationary distribution is defined as

$$\mathbf{u}\Gamma = \mathbf{u}, \quad \mathbf{u} = (u_1, u_2, ..., u_m)$$

$$u_i = \mathbb{P}[c_t = i]$$

Essentially the stationary distribution is the eigenvector of $\Gamma$ with eigenvalue of 1 whose entries sum to 1.

One of the key attributes of an HMM is that the markov chain is unobserved (or hidden). The current state of the markov chain determines the distribution of the observed variable. In our case, the state dependent distributions will be an $n$-dimension multivariate Gaussian. For every state $i \in (1, ..., m)$, the state-dependent distribution will be defined as follows:

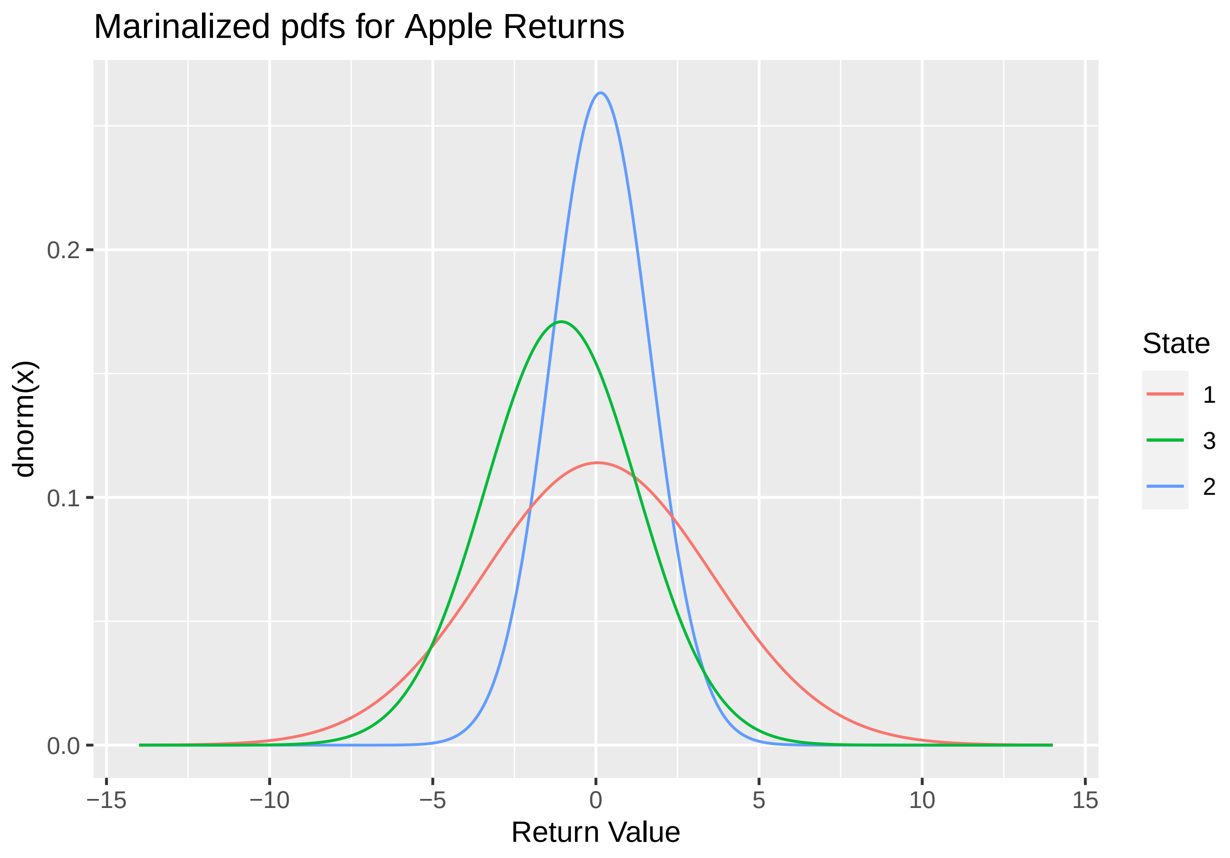
$$p_i(\mathbf{x}_t) = \mathbb{P}(\mathbf{X}_t = \mathbf{x}_t | c_t = i)$$

$$p_i(\mathbf{x_t}) = \frac{1}{\sqrt{(2\pi)^2 |\Sigma_i|}} \exp(-\frac{1}{2}(\mathbf{x_t} - \mu_\mathbf{i}{}^T)\Sigma_i^{-1}(\mathbf{x_t} - \mu_\mathbf{i}))$$

$\Sigma_i$ is the $n \times n$ Variance-Covariance matrix for state $i$ and $\mu_i$ is a length $n$ vector of means for state $i$.

So an HMM is fully described by $\Gamma$, a transition probability matrix; if the model isn't assumed stationary, $\delta$, an initial state distribution; and the parameters for each state dependent distribution. Since only multivariate gaussians will be considered as state dependent distributions, for each state, a vector of means, $\mu$ and a variance-covariance matrix $\Sigma$ must be specified.

In the figure below, one can see how each state has a different distribution from which the returns come. It only shows the marginalized distributions, as the full 4 dimenional distribution is harder to show.



## Model Fitting Specifics:

## Reparameterization

As described by Zucchini and MacDonald, reparameterization is neccesary for direct likelihood maximization. The optimization functions `optim` and `nlm` both need unrestricted parameters. In the following descriptions, working parameters will be the reparameterized values, natural parameters will be the non transformed value.

### Transistion Probability Matrix and Initial Distribution

Every element of the TPM must be between 0 and 1 inclusive. Each row of the TPM must sum to 1. This values can be reparameterized, as described by Zucchini and MacDonald. This is done by rescaling each tranistion probability by the sum of the non-diagonal elements and then taking the log of the rescaled values. This maps the working parameters to the full real line.

### Means

The means for the multivariate gaussians don't need to be reparameterized because they are already unrestricted.

### Correllation Matrices

The correlation matrices are all positive definite, meaning they are symmetric and its eigenvalues are positive. In addition, the values on the diagonal are all 1, since each return always has a correlation of 1 with itself. The non-diagonal values all have values between -1 and 1 inclusive. With all of these restrictions, only the upper triangular values of the correlation matrix need to be estimated. A scaled tan function is used to map the correlation values onto the reals. Here the working values for the correlations will be $\kappa$.

$$\kappa_{ij} = \tan(\frac{\pi c_{ij}}{2}); i \neq j$$

In order to get back to the natural value:

$$c_{ij} = \frac{2\arctan(\kappa_{ij})}{\pi}$$

### Variances

The variances are all strictly positive. So log can be used to map the variances to the real line. The working variances will be $\epsilon_i$.

$$\epsilon_i = \log(\sigma_i^2); \quad \sigma^2 = \exp(\epsilon_i)$$

Zucchini and Macdonald describe a different method of reparameterization, using a log function on the 'Cholesky upper-triangular square root'. This however doesn't allow for negative correlation values. Allowing negative correlation values resulted in models with higher likelihoods.

```r
#mod is a list with the following elements:
#m: number of states (not an element of list)
#n: dimension of the multivariate gaussian (not an element of list)
#MEANS <- matrix (m x n) Mean. Each row is are the means for each state
#VARS <- matrix (m x n) Variances. Each row are the variances for each state
#CORR <- 3D array (n x n x m)  correlation matrices
#TPM <- matrix (m x m) one step transition probabilities
#ID <- vector (m) initial distribution

###Natural Parameters to Working parameters function:
#Inputs:

mvn.n2w <- function(mod, stationary){
  #Reparameterization for tpm:
  tpm <- mod$TPM
  m <- dim(tpm)[1]
  tpm <- log(tpm/diag(tpm)) #Rescale tpm values by the diagonals, then take log.
  tpm <- as.vector(tpm[!diag(m)]) #turn into vector, but exclue diagonal values.
  #Initial Distribution:
```

```r
  if(stationary == F){
    id <- mod$ID
    id<-log(id[-1]/id[1]) #If model isn't stationary, rescale the
    #initial distribution similarly to the tpm
  }
  #correlation:
  corr <-mod$CORR
  corr <- mvn.ar_to_vec(corr) #turn array of into vector, only taking the upper diagonal of
  #each matrix in the array
  corr <- tan(corr*pi/2) #take tangent of the correlation values.
  #variances:
  vars <- mod$VARS
  vars <- as.vector(vars)
  vars <- log(vars)
  #means (don't need reparam, just vectorization)
  mns <- as.vector(mod$MEANS)
  params <- c(tpm, corr, vars, mns)
  if(!stationary){
    params <- c(params, id)
  }
  return(params) #return a single vector of parameters. need a single vector
}

mvn.w2n <- function(params, m, n, stationary){
  #Indices for the various parameters
  tpm_last <- (m*(m-1))
  corr_last <- tpm_last + (n*n - n*(n+1)/2)*m
  vars_last <- corr_last + m*n
  mns_last <- vars_last + m*n
  #Transistion Probability Matrix
  tpm <- params[1:tpm_last]
  TPM <- diag(m)
  TPM[!TPM] <- exp(tpm) #Taking exp to move back to normal parameters
  TPM <- TPM/apply(TPM,1,sum) # Divide by sum of row to ensure row sums to 1.
  #Correlation Array:
  corr <- params[(tpm_last+1):corr_last]
  corr <- atan(corr)*2/pi #Take inverse tan to turn back to normal
  CORR <- mvn.vec_to_ar(corr, n, m) #Turn back into an array
  #Variance Matrix:
  vars <- params[(corr_last + 1):vars_last]
  vars <- exp(vars) #turn back into normal parameters using exp.
  VARS <- matrix(vars, nrow = m, ncol  = n) #turn back into matrix
  #Means:
  means <- params[(vars_last+1):mns_last]
  MEANS <- matrix(means, nrow = m, ncol = n, byrow = F) #only need to turn back into matrix.
  if(stationary){
    #If stationary, sets the initial distribution to the stationary distribution
    ID <- stat_dist(TPM)
  }else{
    #If not stationary, takes the last values and turns back to normal parameters
    id <- tail(params, n = (m-1))
    foo<-c(1,exp(id))
    ID<-foo/sum(foo)
  }
  return(
    list(
      MEANS = MEANS,
      CORR = CORR,
      VARS = VARS,
      TPM = TPM,
      ID = ID
    )
  )
}
```

**Code:**

## Likelihood Calculation

The likelihood of a series of observations from a HMM can be calculated recursively through matrix multiplication as described by Zucchini and MacDonald.

$$L_T = \delta P(\mathbf{x}_1)\Gamma P(\mathbf{x}_2)\Gamma P(\mathbf{x}_3)...\Gamma P(\mathbf{x}_T)\mathbf{1}'$$

Where $P(\mathbf{x}_i)$ is a diagonal matrix with entries:

$$p_i(\mathbf{x}_t) = \mathbb{P}[\mathbf{x} = \mathbf{x}_t | c_t = i]$$

In order to avoid underflow, the log-likelihood is calculated. From the likelihood formula above, the likelihood is a recursive calucaltion where a vector of legnth $m$ is continually multiplied by $\Gamma$ and $P(x)$. One can't simply take the log of the entire calculation. There is a way around this, Zucchini and Macdonald describe a method where this vector is continually rescaled for it to always sum to 1, and the log likelihood is just the sum of the logs of these rescaling factors. This avoids underflow problems in the log-likelihood caluclation.

```
#Log Liklihood Calculation Function
#Inputs:
#parvec: a model object as described above which has been transformed by mvn.n2w
#X: Observed Dataset
#m: number of states, a natural number
#n: dimensionality of state depenedent gaussians, a natural number
#stationary: boolean indicating if the model is stationary
#print: boolean indicating whether or not to print the given model and the calculated likelihood.
mvn.HMM_mllk <- function(parvec, X, m, n, stationary, print = F){
  mod <- mvn.w2n(parvec, m, n, stationary) #Get model from working parameters
  if(!stationary){
    mod$ID <- sapply(mod$ID, FUN = threshold)
  }
  tpm <- mod$TPM
  t <- dim(X)[1] # number of observations
  phi <- mod$ID * diag(mvn.p_matrix(mod, X[1,])) #Start with
  l <- log(sum(phi)) #log likelihood
  phi <- phi/sum(phi)
  for(i in 2:t){
    v <- phi %*% tpm * diag(mvn.p_matrix(mod, X[i,]))
    u <- sum(v)
    l <- l + log(u)
    phi <- v/u #rescaled vector of forward probabilities
  }
  if(print){
    print(-l)
    print(mod)}
  return(-l)
}
```

**Code:**

### Model Fitting

Model fitting is done by using a optimizing function to find the parameters that find the smallest negative log-likelihood. Zucchini and MacDonald used the `nlm` function. Tests were used for both `optim` and `nlm`. When fitting models with `nlm`, the `NA/Inf replaced by maxmimum positive value` error occurs when fitting most models, but it does not stop the function from working. `optim` does not have these errors, but it does take significantly longer to converge. `nlm` was ultimately used because it found models with higher likelihoods.

For the example dataset, the a model was fitted to the returns from the most recent 100 days of the dataset, Oct 24 2023 - Feb 1st 2024. This model was then used as an initial condition to fit the data to the entire 5 years of returns. The results of the model fitting will be discussed later.

## Pseudoresiduals

Pseudoresiduals are a method of model diagnostics described by Zucchini and MacDonald. They create a cumulative density distribution for each time by conditioning off of the rest of the data set. For a properly fitted model, one would expect that these cdfs calculated for each observation would be uniformly distributed. Zucchini and MacDonald describe these cdf values as uniform pseudoresiduals. For the purpose of outlier identification, one can perform the inverse cdf of a standard normal (`qnorm` in `R`) on uniform pseudoresiduals. These are called normal pseudoresiduals and they are expected to be standard normal, given the model is a good fit.

In Zucchini and MacDonald, these pseudoresiduals are only described in depth for univariate data. For multivariate data there is a decision to be made. Now that each time has a vector of observations, one can calculate pseudoresiduals of either the entire vector, or individual elements.

For both methods, the state-dependent cdfs are scaled by what are described by Zucchini and Macdonald as forward and backward probabilities ( $\alpha_t$ and $\beta_t$ respectively).

#### Forward Probabilites:

Forward probabilities are defined as follows:

$$\alpha_t = \delta P(\mathbf{x}_1)\Gamma P(\mathbf{x}_2)\Gamma P(\mathbf{x}_3)...\Gamma P(\mathbf{x}_t)$$

Essentially each $\alpha_t$ a vector of the probabilities of observing $\mathbf{x}^{(t)}$ given the state at time $t$ is $1, ..., m$.

$$\alpha_t(j) = \mathbb{P}(\mathbf{X}^{(t)} = \mathbf{x}^{(t)} | c_t = j)$$

**Backward Probabilities**

Whereas forward probabilities are defined as the probability of observing all the observations up to and including time $t$, backward probabilities are the probabilities of observing all the observations after time $t$. The backward probability vector is defined as follows:

$$\beta_t = \Gamma P(\mathbf{x}_{t+1})\Gamma P(\mathbf{x}_{t+2})...\Gamma P(\mathbf{x}_T)\not\Vdash$$
$$\beta_T = \not\Vdash$$

These values will be used to define the conditional probabilities for each observation. The pdf for each observation is going to be the probability of an observation conditioned on the rest of the observations. To be explicit:

$$\mathbb{P}(X_t = \mathbf{x}|\mathbf{X}^{(-t)} = \mathbf{x}^{(-t)}) = \mathbb{P}(\frac{X_t = \mathbf{x}, \mathbf{X}^{(-t)} = \mathbf{x}^{(-t)}}{\mathbf{X}^{(-t)} = \mathbf{x}^{(-t)}})$$

The numerator is just the likelihood as described above, just with the observation at time $t$ replaced with the generic observation vector $\mathbf{x}$. The above probability can be calculated using the following values:

$$\mathbb{P}(X_t = \mathbf{x}|\mathbf{X}^{(-t)} = \mathbf{x}^{(-t)}) = \frac{\delta P(\mathbf{x}_1)\Gamma P(\mathbf{x}_2)...\Gamma P(\mathbf{x}_{t-1})\Gamma P(\mathbf{x})...\Gamma P(\mathbf{x}_T)\mathbf{1'}}{\delta P(\mathbf{x}_1)\Gamma P(\mathbf{x}_2)\Gamma P(\mathbf{x}_3)...\Gamma P(\mathbf{x}_{t-1})\Gamma P(\mathbf{x}_{t+1})...\Gamma P(\mathbf{x}_T)\mathbf{1'}}$$

This can be simiplified using our notation from above for forward and backward probabilities to then get the pdf for an observation:

$$\mathbb{P}(\mathbf{X}_t = \mathbf{x}|\mathbf{X}^{(-t)} = \mathbf{x}^{(-t)}) = \frac{\alpha_{t-1}\Gamma P(\mathbf{x})\beta_t'}{\alpha_{t-1}\Gamma\beta_t'}$$

From here, getting the conditional cumulative distribution function is trivial, one need only calculate the cdf for $P(\mathbf{x})$ instead of the pdf (Essentially `pmvnorm` instead of `dmvnorm`).

$$\mathbb{P}(\mathbf{X} < \mathbf{x}|\mathbf{X}^{(-t)} = \mathbf{x}^{(-t)}) = \frac{\alpha_{t-1}\Gamma\mathbf{P}(\mathbf{x})\beta_t'}{\alpha_{t-1}\Gamma\beta_t'}$$

Here $\mathbf{P}(\mathbf{x})$ indicates the use of the cdf instead of the pdf.

###Pseudoresidual of Entire Vector:

Using the entire vector for the pseudoresidual produces a single residual for each time. For each observation from $t = 1, ..., T$ the conditional cdf as described above. In order to avoid underflow, Zucchini and Macdonald describe a method of evaluating the matrix multiplication. The forward and backward probabilities are often smaller than the machine tolerance so having as much of the calculations be done with logs is needed. A more in depth description of this is done in Chapter 5.

```
##Function Which Gives pmvnorm evaluated for an entire vector of observations evaluted for each state
mvn.cumul_vec <- function(x, mod){
  mvn <- function(m){
    #Get VarCoVar Matrix from model
    sig <- diag(mod$VARS[m,]) %*% mod$CORR[,,m] %*% diag(mod$VARS[m,])
    #Get Means from Model
    means <- mod$MEANS[m,]
    #Evaluate pvmnorm with Vector of observaions as lower bound:
    return(pmvnorm(lower = X, upper = Inf, mean = means, sigma = sig))
  }
  m <- dim(mod$TPM)[1] #number of states from dimensionality of TPM
  probs <- sapply(1:m,  FUN =mvn)
```

```r
  return(probs)
}


### Function to Produce conditional cdf evaluted at each observation.
#x: Matrix of Observations
#mod: model to evaluate
#Returns a 1 x t matrix where t is the number of observations
mvn.cdf_vector <- function(x,mod){
  lenx         <- dim(x)[1] #number of observations
  m         <- dim(mod$TPM)[1]
  dxc       <- matrix(NA,nrow=lenx,ncol=1) #output matrix
  Px        <- matrix(NA,nrow=lenx,ncol=m) #evaluating pmvnorm at each observation
  for (j in 1:lenx){ Px[j,] <- mvn.cumul_vec(x[j,], mod)}
  la        <- mvn.lforward(x,mod) #Forward Probabilities
  lb        <- mvn.lbackward(x,mod) #Backward Probabilities
  la        <- rbind(log(mod$ID),la)
  lafact    <- apply(la,1,max)
  lbfact    <- apply(lb,1,max)
  #Scale pmvnorm of observations by forward and backward probabilities:
  for (i in 1:lenx)
  {
    foo      <- (exp(la[i,]-lafact[i])%*%mod$TPM)*exp(lb[i,]-lbfact[i])
    foo      <- foo/sum(foo)
    dxc[i]  <- sum(Px[i,]%*%t(foo))
  }
  return(dxc)
}
```

**Code:**

**Pseudoresiduals of Sigle Elements:**

For a single element, the pseudoresidual is calculated in the same way above, but instead of inputing the entire observation as the lower bound, each element of the observation is inputed individually and the other variables are marginalized by taking the cdf over their entire supports. Using this method, the uniform pseudoresiduals were close to uniform for the model fitted to the example dataset as described above.

This method produces a pseudoresidual for each stock for each day.

```r


#Function which provides the separate probabilities of each individual return for each state.
#Returns an m x n matrix of the cdf evaluated for each variable in each state with the other variables
#marginalized.
#x: full observation (vector of legnth n)
#mod: model,

mvn.cumul_mat <- function(x, mod){
  m <- dim(mod$TPM)[1]
  n <- dim(mod$CORR)[2]
  output <- matrix(nrow = m, ncol = n)
  prob_fun <- function(i){
    prob_sub_fun <- function(m){
      sig <- diag(mod$VARS[m,]) %*% mod$CORR[,,m] %*% diag(mod$VARS[m,])
      means <- mod$MEANS[m,]
      return(pmvnorm(lower = gen_lbound(i, X[i], n), upper = Inf, mean = means, sigma = sig))
    }
    probs <- sapply(1:m,  FUN =prob_sub_fun)
  }
  probs <- sapply(1:n,  FUN =prob_fun, simplify = "matrix")
  return(probs)
}

##Function to Produce Conditional cdf evaluated for each return.
mvn.cdf_element <- function(x,mod){
  lenx        <- dim(x)[1]
  m         <- dim(mod$TPM)[1]
  n <- dim(mod$CORR)[2]
  dxc       <- matrix(NA,nrow=lenx,ncol=n)
  Px        <- array(NA,dim = c(m, n, lenx))
  #Using cumul_mat instead of cumul_vec to get it for individual returns, rather than the whole vector.
  for (j in 1:lenx){ Px[,,j] <- mvn.cumul_mat(x[j,], mod)}
  la        <- mvn.lforward(x,mod)
  lb        <- mvn.lbackward(x,mod)
  la        <- rbind(log(mod$ID),la)
  lafact    <- apply(la,1,max)
  lbfact    <- apply(lb,1,max)
  for (i in 1:lenx)
  {
    foo      <- (exp(la[i,]-lafact[i])%*%mod$TPM)*exp(lb[i,]-lbfact[i])
    foo      <- foo/sum(foo)
    for(j in 1:n){
      dxc[i,j]  <- sum(Px[,j,i]%*%t(foo))
    }
  }
  return(dxc)
}
```

**Code:**

## Generated Data Testing:

In order to investigate the difference between these methods data was generated from randomly specified models. For every trial, a random model was generated using the following code:

```r
create_arb_2d_mod <- function(seed){
  set.seed(seed)
  #Generate Means:
  arb_means <- matrix(runif(4,-1, 1), nrow =2 )
  #Generate Variances:
  arb_vars <- runif(4, min = 1, max = 10)
  arb_vars <- matrix(arb_vars, nrow =2 )
  #Generate Correlations:
```

```
  arb_corr <- c(symMat(runif(1, min = -1), diag = F),
                symMat(runif(1, min = -1), diag = F))
  arb_corr <- array(arb_corr, dim = c(2,2,2))
  #Genertate TPM:
  d <- runif(2, min= 0.5, max = 1)
  arb_tpm <- stan_starting_tpm(d)
  #stan_starting tpm makes a matrix with diagonal elements d
  #and the other x elements in the row(1-d_i)/x
  #Generate Initial Distribution
  s1 <- runif(1)
  arb_id <- c(s1, 1-s1)
  #Put model together:
  arb_mod <- list(
    MEANS = arb_means,
    CORR = arb_corr,
    VARS = arb_vars,
    ID = arb_id,
    TPM = arb_tpm
  )
  return(arb_mod)
}
```

2-Dimensional Gaussians were used in order to reduce the number of parameters to be fitted for each trial. The means were set to be close to zero with variances much greater than the means, which was observed in the returns data. For each model, 200 observations were generated, then a second random model was used as an initial condition for fitting a new model to the generated data. For each fitted model, the `mvn.cdf_vector` and `mvn.cdf_element` functions were used. The inverse standard normal function `qnorm` was then used to get the two kinds of normal pseudoresiduals as described above. The mean and variance of the pseudoresiduals was then stored. Note, for the element normal pseudoresiduals, the values for each element were combined into a single vector and that was used for the variance calculation. (NOTE: If this is wrong to do, I can run the numbers again and separate the variances and means for the separate variables).

**Some Caviats about the model Fitting** There were some issues with the model fitting functions actually converging on a model where the pseudoresiduals could be calculated. The exact reason for this issue wasn't able to be idenitified. This never occurred when working with real world data. When this error occured, a different seed was used until there were 200 valid runs. The error happened for less than 5% of the trials. The main purpose of the generated data experiment was just to make sure that it wasn't the example data set that was causing the vector normal pseudoresiduals to not be standard normal.

## Results:

### Model Fitting to the Example Data Set:

There were 2 different models generated from the example returns data set. Both were 3 state models. First was a model fitted to the 100 most recent returns. The second was a model fitted to the entire 5 years of returns.

#### Resulting Model:

Table 1: Means

| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| 0.1458 | 0.1056 | 0.0548 | 0.1363 |
| 0.0567 | 0.1014 | -0.4038 | -0.2420 |
| -1.0593 | 0.2534 | 1.0226 | -0.4097 |

Table 2: Variances

| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| 1.5149 | 1.4020 | 1.6733 | 1.8198 |
| 3.4999 | 3.4949 | 4.7567 | 5.4408 |
| 2.3339 | 0.6887 | 1.2559 | 0.4435 |

Table 3: Correlation State 1

| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| 1.0000 | 0.7677 | 0.5921 | 0.6122 |
| 0.7677 | 1.0000 | 0.5783 | 0.6758 |
| 0.5921 | 0.5783 | 1.0000 | 0.4941 |
| 0.6122 | 0.6758 | 0.4941 | 1.0000 |

Table 4: Correlation State 2

| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| 1.0000 | 0.7640 | 0.5671 | 0.5879 |
| 0.7640 | 1.0000 | 0.5894 | 0.5916 |
| 0.5671 | 0.5894 | 1.0000 | 0.4320 |
| 0.5879 | 0.5916 | 0.4320 | 1.0000 |

Table 5: Correlation State 3

| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| 1.0000 | 0.6752 | -0.3192 | 0.0301 |
| 0.6752 | 1.0000 | -0.7581 | 0.0097 |
| -0.3192 | -0.7581 | 1.0000 | -0.4641 |
| 0.0301 | 0.0097 | -0.4641 | 1.0000 |

Table 6: Transition Probability Matrix

| State 1 | State 2 | State 3 |
|---|---|---|
| 0.90234 | 0.09031 | 0.00735 |
| 0.44961 | 0.55039 | 0.00000 |
| 0.52896 | 0.00000 | 0.47104 |

**Pseudoresiduals of Fitted Model:**

**Density of Element Pseudoresiduals for Each State**

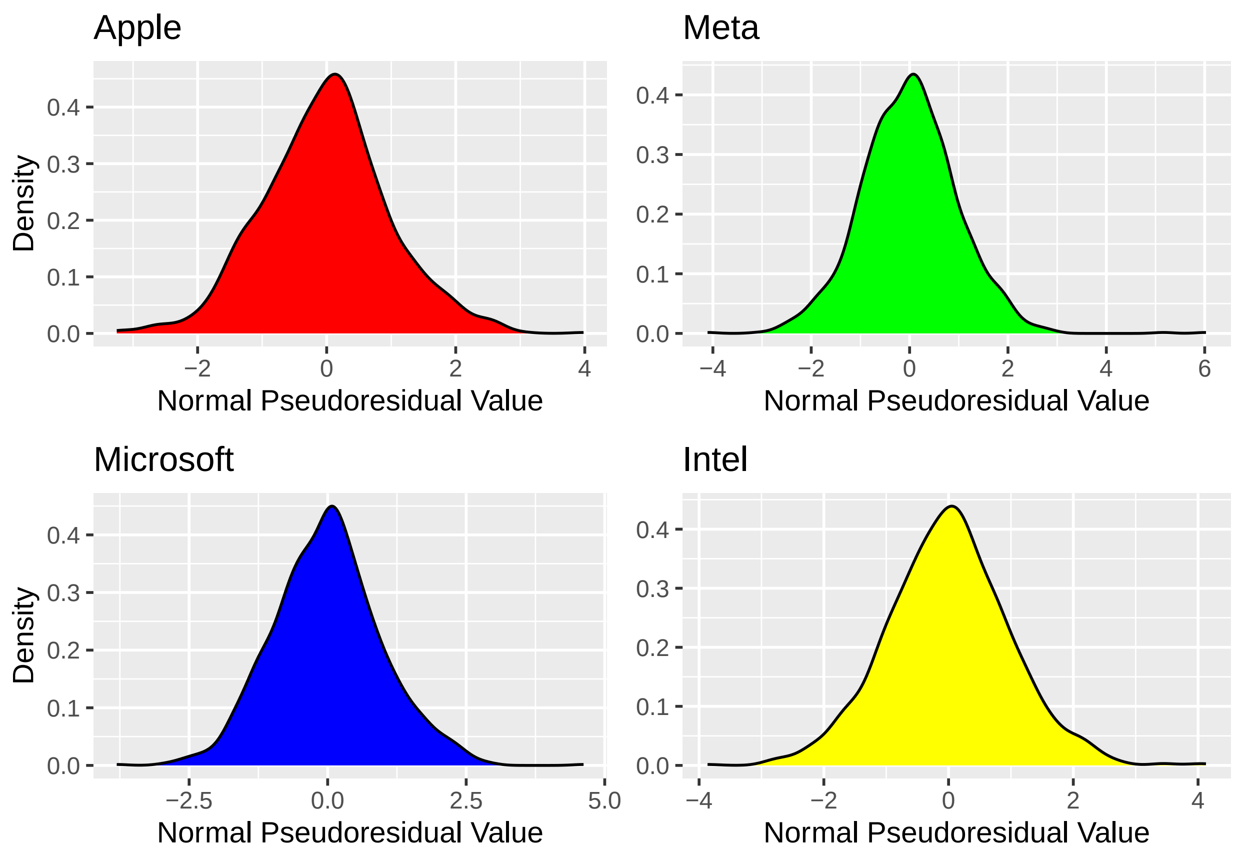


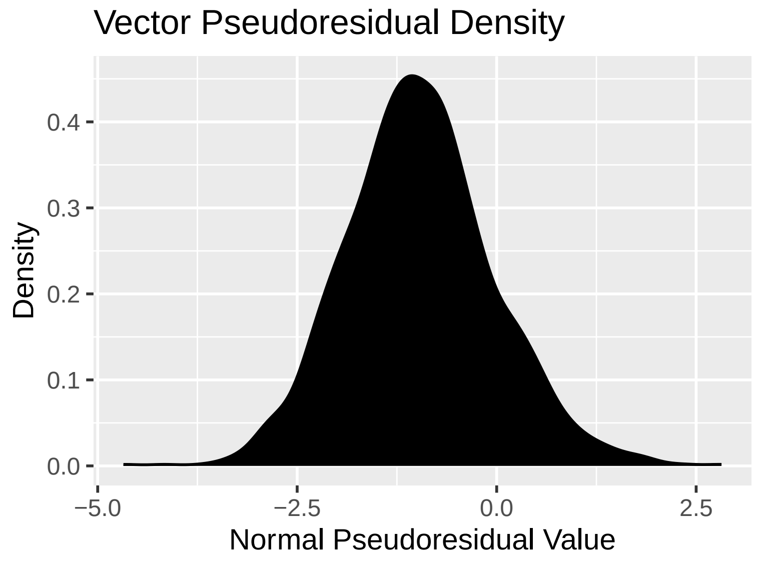
Table 7: Means of Element Pseudoresiduals

| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| -0.0094 | -0.0042 | -0.001 | -0.0048 |

Table 8: Variances of Element Pseudoresiduals

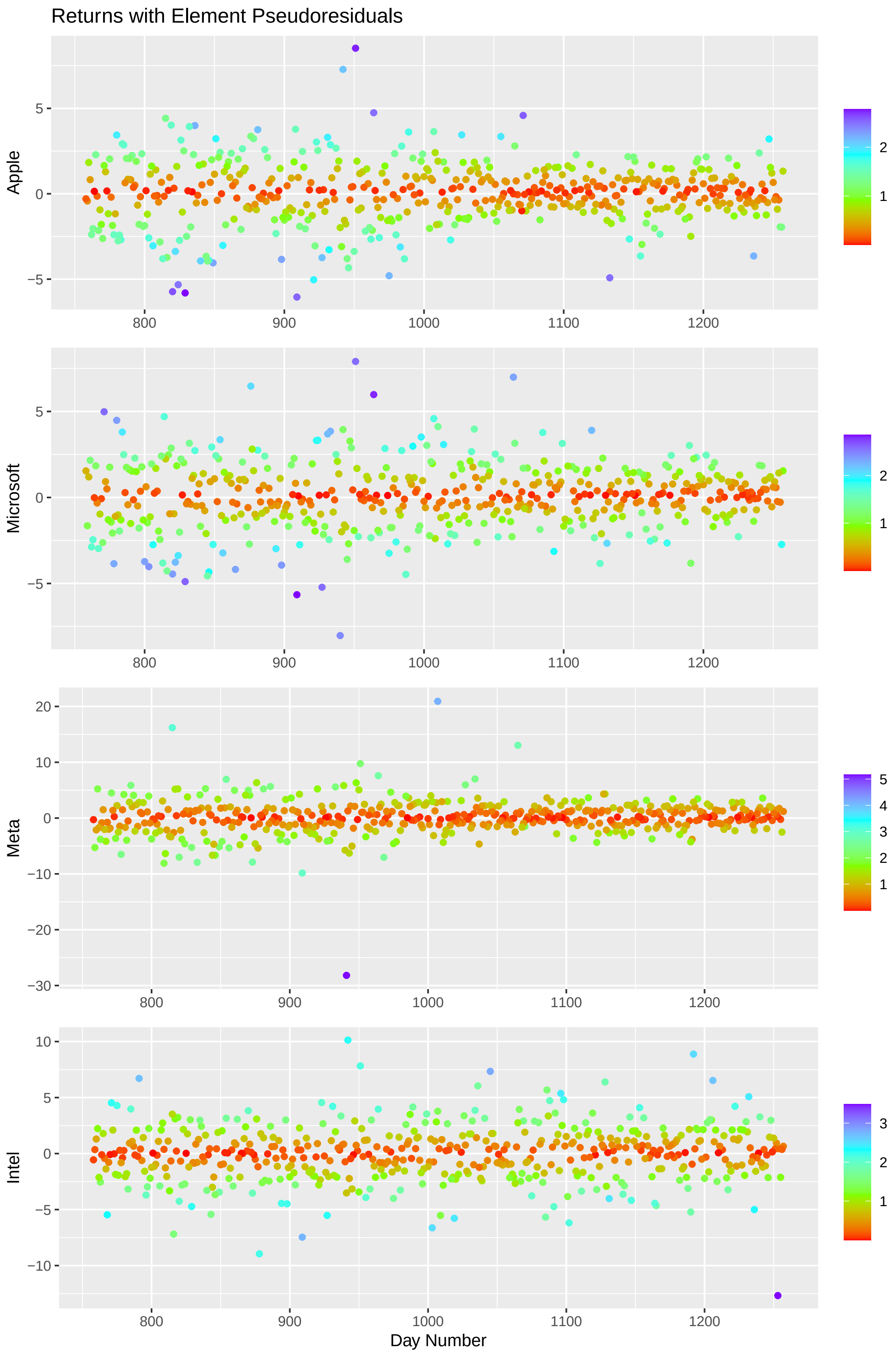
| Apple | Microsoft | Meta | Intel |
|---|---|---|---|
| 0.9515 | 0.9457 | 0.9557 | 0.9688 |

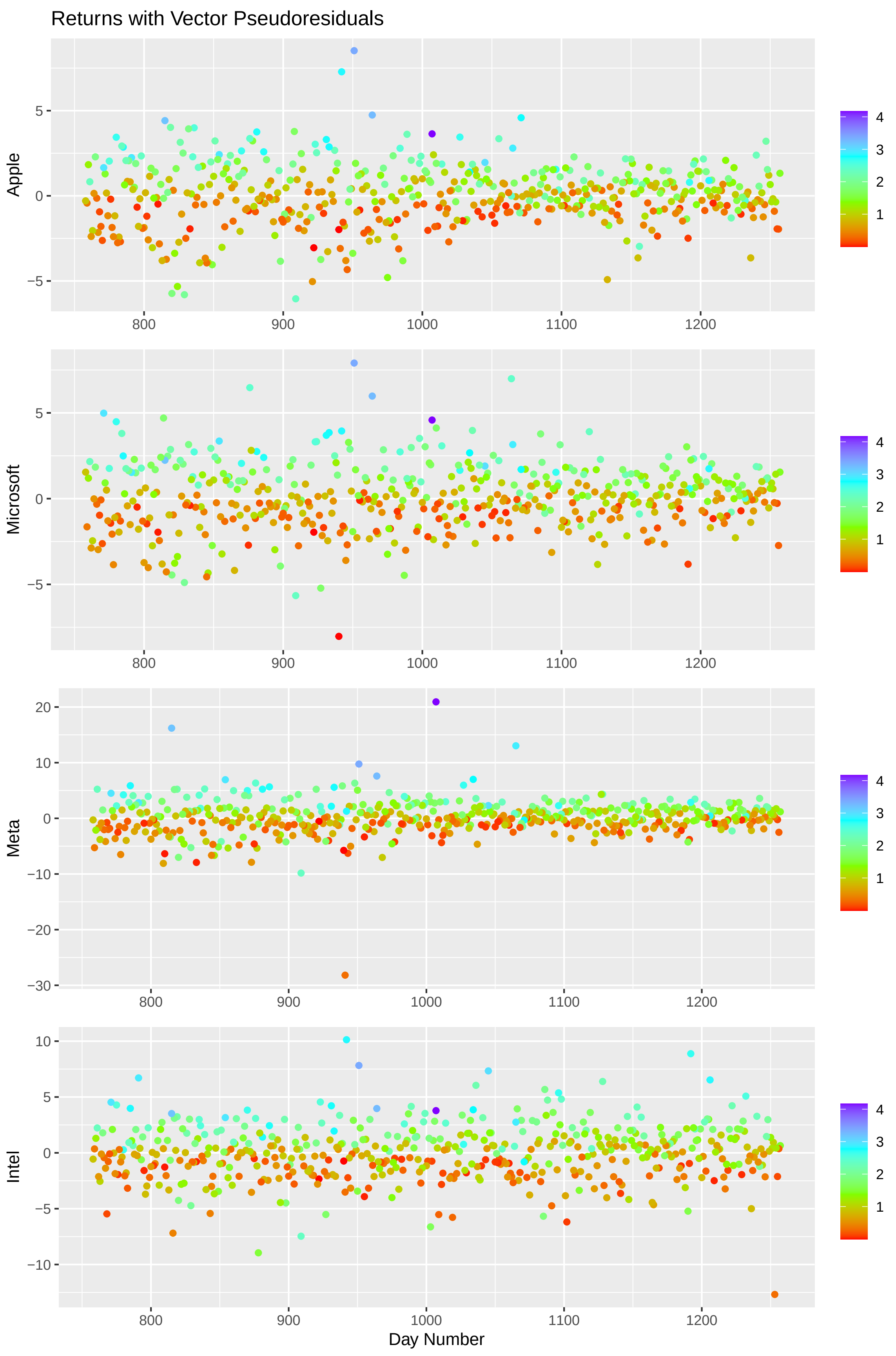
**Density of Normal Vector Pseudoresiduals**



The normal vector pseudoresiduals have a mean of -0.9674067 and a variance of 0.8596349. The means of normal vector pseudoresiduals being far from 0 will be shown as a trend from the results of the generated data trials.

**Pseudoresidual Plots of Observations:**



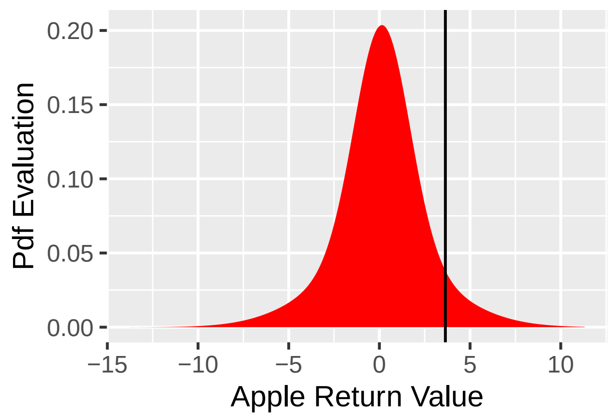

Returns with Vector Pseudoresiduals
Apple
Microsoft
Meta
Intel
Day Number
800
900
1000
1100
1200
5
0
−5
20
10
−10
−20
−30
4
3
2
1

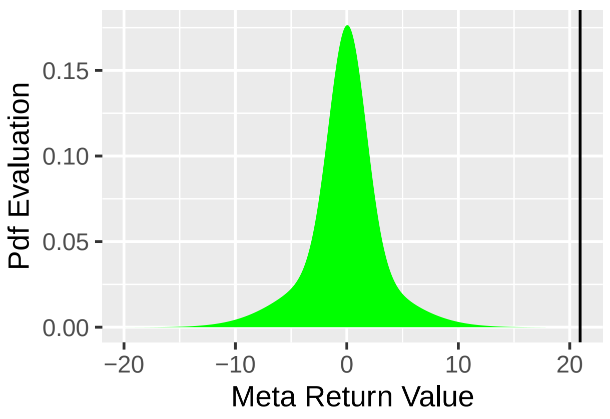
Notes on Element Pseudoresiduals: Outliers occur at different times for different times. -Particular note for intel stock where there is a particularly high pseudoresidual

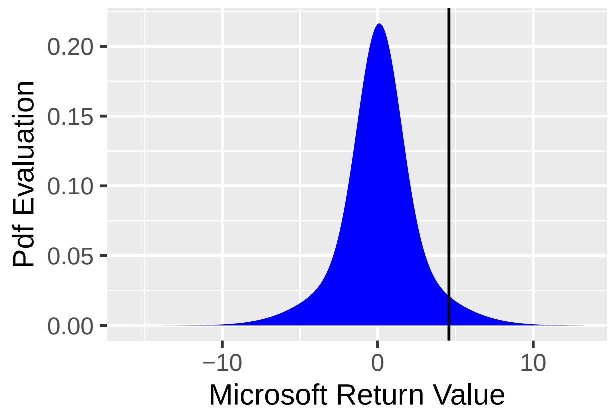
Notes on Vector Pseudoresiduals: Pseudoresiduals are the same for each day. Marks entire vectors as outliers rather than individual stocks.

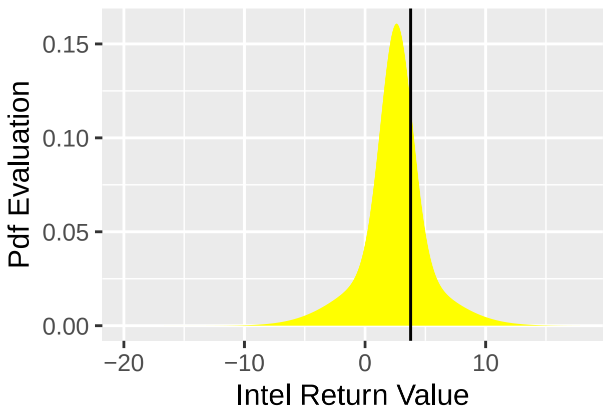
**Graphs of Conditional cdfs**

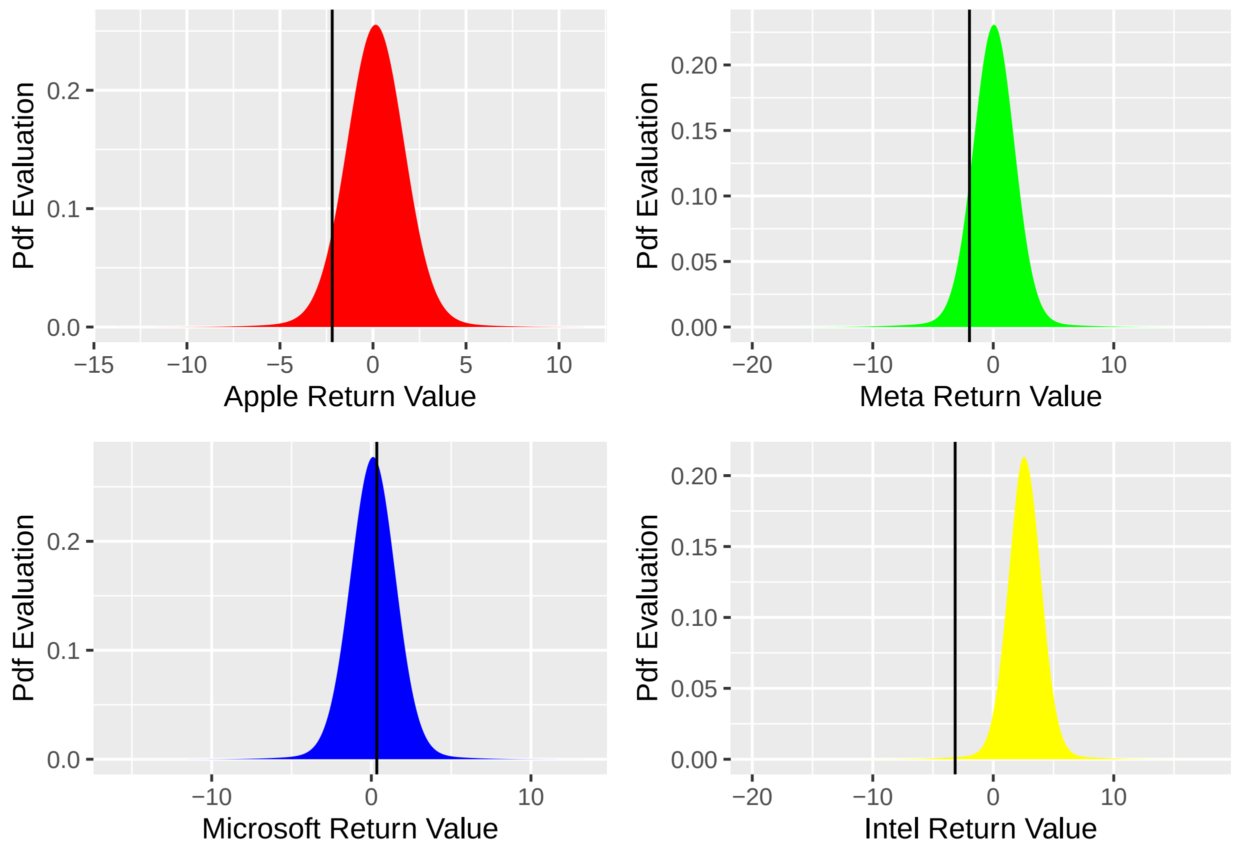
####Conditional Distributions for Smallest Vector Pseudoresidual







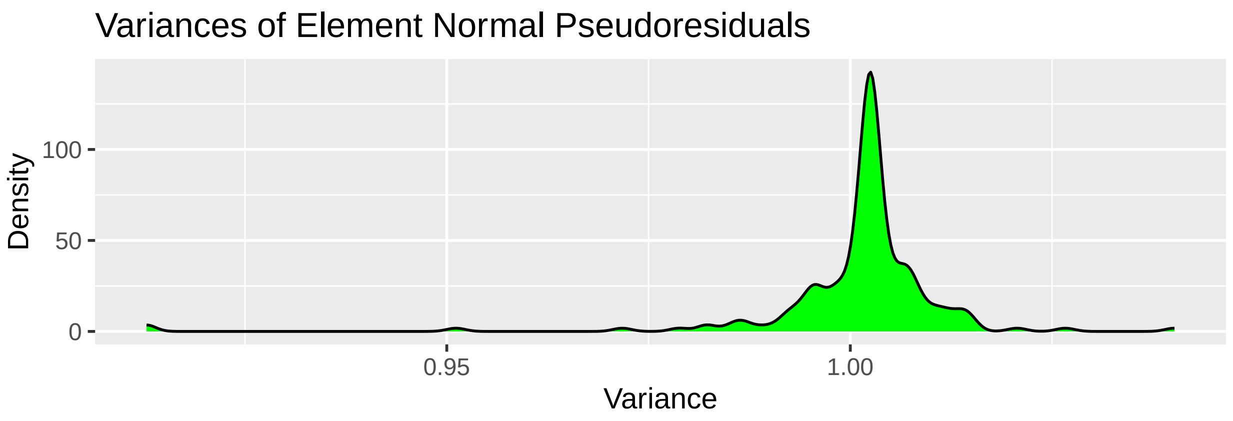


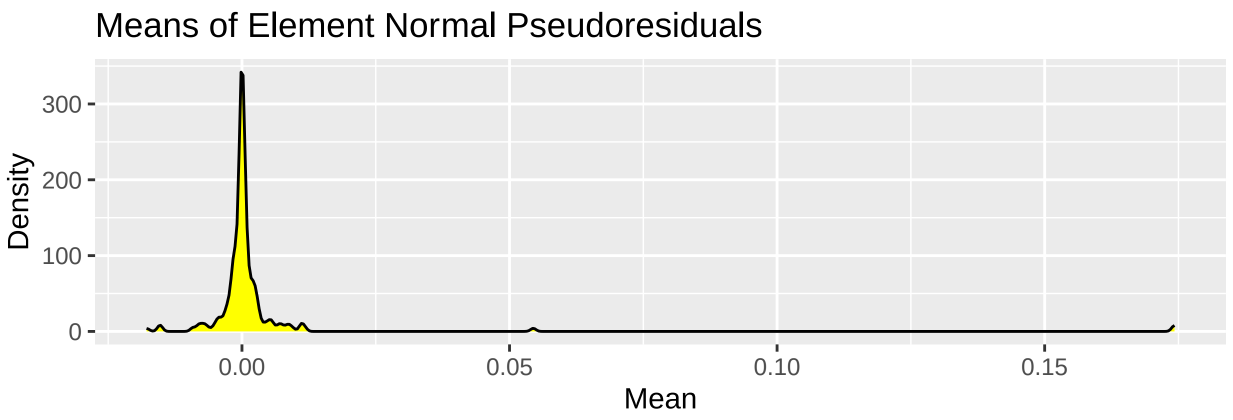


## Generated Data Results:

Generated data testing was used in the case that the results from model fitting to the example dataset were an anomaly. The results show that this most likely is not the case. Overwhelmingly, the means and variances for the element normal pseudoresiduals were in line with standard normal, whereas the same values for the vector normal pseudoresiduals were much more widely distributed and clearly diverge from standard normal.

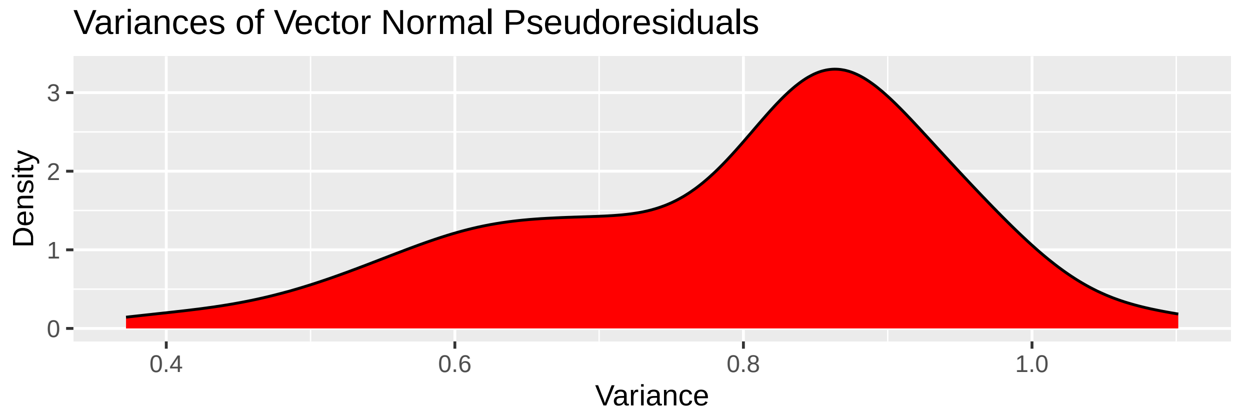
**Density Plots for Element Normal Pseudoresiduals:**

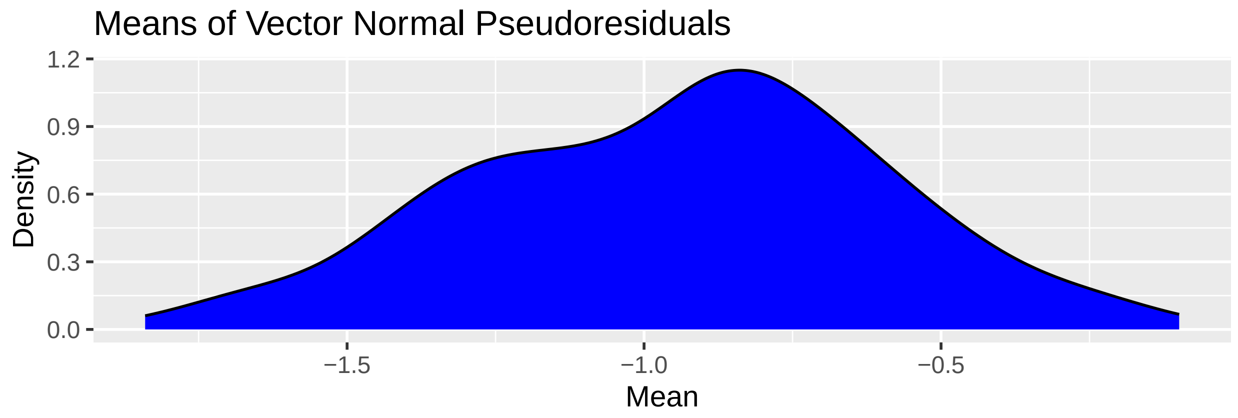




The densities for the means and variances of the element normal pseudoresiduals clearly peak at 1 and 0 respectively, which is expected for normal pseudoresiduals of well fitted models.

**Density Plots for Vector Normal Pseudoresiduals:**





## Conclusion:

The element pseudoresiduals are clearly the pseudoresidual described by Zucchini and Macdonald. They allow for outlier identification for individual stock returns. Vector pseudoresiduals are more of an enigma. They don't provide as clear of an indication for how well a model fits. More investigation is needed

## Extensions:

Topics:

-Using Pseudoresiduals for Model Selection -Comparing Pseudoresiduals for different numbers of states. -Comparing Pseudoresiduals for different levels of dimensionality -Which stocks to include in model fitting

-Exploring Patterns in Vector Normal Pseudoresiduals -Perhaps there is some relationship in the dimensionality of the state-dependent distribution

-Using Different State-Dependent Distributions -t-distribution