Example Stochastic Reserving

Roger Hayne

9/6/2019

```
library(mvtnorm)
library(MASS)
library(abind)
library(stochasticreserver)
```

Initialize Triangle

Input (B0) is a development array of cumulative averages with a the exposures (claims) used in the denominator appended as the last column. Assumption is for the same development increments as exposure increments and that all development lags with no development have # been removed. Data elements that are not available are indicated as such. This should work (but not tested for) just about any subset of an upper triangular data matrix.
Another requirement of this code is that the matrix contain no columns that are all zero.

```
B0 = matrix(c(670.25868,1480.24821,1938.53579,2466.25469,2837.84888,3003.52391,
            3055.38674,3132.93838,3141.18638,3159.72524,
            767.98833,1592.50266,2463.79447,3019.71976,3374.72689,3553.61387,3602.27898,
            3627.28386,3645.5656,NA,
            740.57952,1615.79681,2345.85028,2910.52511,3201.5226,3417.71335,3506.58672,
            3529.00243,NA,NA,
            862.11956,1754.90405,2534.77727,3270.85361,3739.88962,4003.00219,4125.30694,
            NA,NA,NA,
            840.94172,1859.02531,2804.54535,3445.34665,3950.47098,4185.95298,NA,NA,NA,NA,
            848.00496,2052.922,3076.13789,3861.03111,4351.57694,NA,NA,NA,NA,NA,
            901.77403,1927.88718,3003.58919,3881.41744,NA,NA,NA,NA,NA,NA,
            935.19866,2103.97736,3181.75054,NA,NA,NA,NA,NA,NA,NA,
            759.32467,1584.91057,NA,NA,NA,NA,NA,NA,NA,NA,
            723.30282,NA,NA,NA,NA,NA,NA,NA,NA,NA),10,10,byrow = TRUE)
dnom = c(39161.,38672.4628,41801.048,42263.2794,41480.8768,40214.3872,43598.5056,
       42118.324,43479.4248,49492.4106)

# Identify model to be used
#   Berquist for the Berquist-Sherman Incremental Severity
#   CapeCod for the Cape Cod
#   Hoerl for the Generalized Hoerl Curve Model with trend
#   Wright for the Generalized Hoerl Curve with individual accident year levels
#   Chain for the Chain Ladder model
#model = "Berquist"
model = "CapeCod"
#model = "Hoerl"
#model = "Wright"
#model = "Chain"
# Toggle graphs off if desired
graphs = TRUE

# Toggle simulations off if desired
```

```r
simulation = TRUE
```

```r
# Set tau to have columns with entries 1 through 10
tau = t(array((1:10), c(10, 10)))

# Calculate incremental average matrix
A0 = cbind(B0[, 1], (B0[, (2:10)] + 0 * B0[, (1:9)]) -
             (B0[, (1:9)] + 0 * B0[, (2:10)]))

# Generate a matrix to reflect exposure count in the variance structure
logd = log(matrix(dnom, 10, 10))

# Set up matrix of rows and columns, makes later calculations simpler
rowNum = row(A0)
colNum = col(A0)

# msk is a mask matrix of allowable data, upper triangular assuming same
# development increments as exposure increments, msn picks off the first
# forecast diagonal, msd picks off the to date diagonal
msk = (10 - rowNum) >= colNum - 1
msn = (10 - rowNum) == colNum - 2
msd = (10 - rowNum) == colNum - 1

# Amount paid to date
ptd = rowSums(B0 * msd, na.rm = TRUE)
```

START OF MODEL SPECIFIC CODE

```r
  if (model == "Berquist") {
    model_lst <- berquist(tau, B0, ptd, msk)
  } else if (model == "CapeCod") {
    model_lst <- capecod(tau, B0, ptd, msk)
  } else if (model == "Hoerl") {
    model_lst <- hoerl(tau, B0, ptd, msk)
  } else if (model == "Wright") {
    model_lst <- wright(tau, B0, ptd, msk)
  } else if (model == "Chain") {
    model_lst <- chain(tau, B0, ptd, msk)
  }
g.obj <- model_lst$g.obj
g.grad <- model_lst$g.grad
g.hess <- model_lst$g.hess
a0 <- model_lst$a0
```

Negative Loglikelihood Function to be Minimized

Note that the general form of the model has parameters in addition to those in the loss model, namely the power for the variance and the constant of proprtionality that varies by column. So if the original model has k parameters with 10 columns of data, the total objective function has k+11 parameters

```r
l.obj = function(a, A) {
  npar = length(a) - 2
```

```
  e = g.obj(a[1:npar])
  v = exp(-outer(logd[, 1], rep(a[npar + 1], 10), "-")) * (e ^ 2) ^ a[npar +
                                                                          2]
  t1 = log(2 * pi * v) / 2
  t2 = (A - e) ^ 2 / (2 * v)
  sum(t1 + t2, na.rm = TRUE)
}
# Gradient of the objective function
l.grad = function(a, A) {
  npar = length(a) - 2
  p = a[npar + 2]
  Av = aperm(array(A, c(10, 10, npar)), c(3, 1, 2))
  e = g.obj(a[1:npar])
  ev = aperm(array(e, c(10, 10, npar)), c(3, 1, 2))
  v = exp(-outer(logd[, 1], rep(a[npar + 1], 10), "-")) * (e ^ 2) ^ p
  vv = aperm(array(v, c(10, 10, npar)), c(3, 1, 2))
  dt = rowSums(g.grad(a[1:npar]) * ((p / ev) + (ev - Av) / vv - p * (Av -
                                                                         ev) ^ 2 / (vv * ev)),
              na.rm = TRUE,
              dims = 1)
  yy = 1 - (A - e) ^ 2 / v
  dk = sum(yy / 2, na.rm = TRUE)
  dp = sum(yy * log(e ^ 2) / 2, na.rm = TRUE)
  c(dt, dk, dp)
}
```

Hessian of the objective function

- e is the expectated value matrix
- v is the matrix of variances
- A, e, v all have shape c(10,10)
- The variables _v are copies of the originals to shape c(npar,10,10), paralleling the gradient of g.
- The variables _m are copies of the originals to shape c(npar,npar,10,10), paralleling the hessian of g

```
l.hess = function(a, A) {
  npar = length(a) - 2
  p = a[npar + 2]
  Av = aperm(array(A, c(10, 10, npar)), c(3, 1, 2))
  Am = aperm(array(A, c(10, 10, npar, npar)), c(3, 4, 1, 2))
  e = g.obj(a[1:npar])
  ev = aperm(array(e, c(10, 10, npar)), c(3, 1, 2))
  em = aperm(array(e, c(10, 10, npar, npar)), c(3, 4, 1, 2))
  v = exp(-outer(logd[, 1], rep(a[npar + 1], 10), "-")) * (e ^ 2) ^ p
  vv = aperm(array(v, c(10, 10, npar)), c(3, 1, 2))
  vm = aperm(array(v, c(10, 10, npar, npar)), c(3, 4, 1, 2))
  g1 = g.grad(a[1:npar])
  gg = aperm(array(g1, c(npar, 10, 10, npar)), c(4, 1, 2, 3))
  gg = gg * aperm(gg, c(2, 1, 3, 4))
  gh = g.hess(a[1:npar])
  dtt = rowSums(
    gh * (p / em + (em - Am) / vm - p * (Am - em) ^ 2 / (vm * em)) +
      gg * (
        1 / vm + 4 * p * (Am - em) / (vm * em) + p * (2 * p + 1) * (Am - em) ^ 2 /
```

```
          (vm * em ^ 2) - p / em ^ 2
      ),
    dims = 2,
    na.rm = TRUE
  )
  dkt = rowSums((g1 * (Av - ev) + p * g1 * (Av - ev) ^ 2 / ev) / vv, na.rm = TRUE)
  dtp = rowSums(g1 * (1 / ev + (
    log(ev ^ 2) * (Av - ev) + (p * log(ev ^ 2) - 1) * (Av - ev) ^ 2 / ev
  ) / vv),
  na.rm = TRUE)
  dkk = sum((A - e) ^ 2 / (2 * v), na.rm = TRUE)
  dpk = sum(log(e ^ 2) * (A - e) ^ 2 / (2 * v), na.rm = TRUE)
  dpp = sum(log(e ^ 2) ^ 2 * (A - e) ^ 2 / (2 * v), na.rm = TRUE)
  m1 = rbind(array(dkt), c(dtp))
  rbind(cbind(dtt, t(m1)), cbind(m1, rbind(cbind(dkk, c(
    dpk
  )), c(dpk, dpp))))
}
```

End of funciton specificaitons now on to the minimization

Minimization

Get starting values for kappa and p parameters, default 10 and 1

```
ttt = c(10, 1)
```

For starting values use fitted objective function and assume variance for a cell is estimated by the square of the difference between actual and expected averages. Note since log(0) is -Inf we need to go through some machinations to prep the y values for the fit

```
E = g.obj(a0)
yyy = (A0 - E)^2
yyy = logd + log(((yyy != 0) * yyy) - (yyy == 0))
```

```
## Warning in log(((yyy != 0) * yyy) - (yyy == 0)): NaNs produced
```

```
sss = na.omit(data.frame(x = c(log(E^2)), y = c(yyy)))
ttt = array(coef(lm(sss$y ~ sss$x)))[1:2]
a0 = c(a0, ttt)

set.seed(1) # to check reproducibility with original code
max = list(iter.max = 10000, eval.max = 10000)
```

Actual minimization

```
mle = nlminb(a0, l.obj, gradient = l.grad, hessian = l.hess,
             scale = abs(1 / (2 * ((a0 * (a0 != 0)) + (1 * (a0 == 0))))),
             A = A0, control = max)
```

Model statistics

- **mean** and **var** are model fitted values
- **stres** is the standardized residuals

```
npar = length(a0) - 2
p = mle$par[npar + 2]
mean = g.obj(mle$par[1:npar])
var = exp(-outer(logd[, 1], rep(mle$par[npar + 1], 10), "-")) * (mean ^
                                                                    2) ^ p
stres = (A0 - mean) / sqrt(var)
g1 = g.grad(mle$par[1:npar])
gg = aperm(array(g1, c(npar, 10, 10, npar)), c(4, 1, 2, 3))
gg = gg * aperm(gg, c(2, 1, 3, 4))
meanv = aperm(array(mean, c(10, 10, npar)), c(3, 1, 2))
meanm = aperm(array(mean, c(10, 10, npar, npar)), c(3, 4, 1, 2))
varm = aperm(array(var, c(10, 10, npar, npar)), c(3, 4, 1, 2))
```

Masks to screen out NA entries in original input matrix

```
s = 0 * A0
sv = aperm(array(s, c(10, 10, npar)), c(3, 1, 2))
sm = aperm(array(s, c(10, 10, npar, npar)), c(3, 4, 1, 2))
```

Calculate the information matrix

- Using second derivatives of the log likelihood function Second with respect to theta parameters

```
tt = rowSums(sm + gg * (1 / varm + 2 * p ^ 2 / (meanm ^ 2)), dims = 2, na.rm = TRUE)
```

Second with respect to theta and kappa

```
kt = p * rowSums(sv + g1 / meanv, na.rm = TRUE)
```

Second with respect to p and theta

```
tp = p * rowSums(sv + g1 * log(meanv ^ 2) / meanv, na.rm = TRUE)
```

Second with respect to kappa

```
kk = (1 / 2) * sum(1 + s, na.rm = TRUE)
```

Second with respect to p and kappa

```
pk = (1 / 2) * sum(s + log(mean ^ 2), na.rm = TRUE)
```

Second with respect to p

```
pp = (1 / 2) * sum(s + log(mean ^ 2) ^ 2, na.rm = TRUE)
```

Create information matrix in blocks

```
m1 = rbind(array(kt), c(tp))
inf = rbind(cbind(tt, t(m1)), cbind(m1, rbind(c(kk, pk), c(pk, pp))))
```

Variance-covariance matrix for parameters, inverse of information matrix

```
vcov = solve(inf)
```

Simulation

Initialize simulation array to keep simulation results

```
sim = matrix(0, 0, 11)
smn = matrix(0, 0, 11)
spm = matrix(0, 0, npar + 2)
```

Simulation for distribution of future amounts

Want 10,000 simulations, but exceeds R capacity, so do in batches of 5,000

```
nsim = 5000
smsk = aperm(array(c(msk), c(10, 10, nsim)), c(3, 1, 2))
smsn = aperm(array(c(msn), c(10, 10, nsim)), c(3, 1, 2))

if (simulation) {
  for (i in 1:5) {
    # Randomly generate parameters from multivariate normal
    spar = rmvnorm(nsim, mle$par, vcov)

    # Arrays to calculate simulated means
    esim = g.obj(spar)

    # Arrays to calculate simulated variances
    ksim = exp(aperm(outer(array(
      spar[, c(npar + 1)], c(nsim, 10)
    ), log(dnom), "-"), c(1, 3, 2)))
    psim = array(spar[, npar + 2], c(nsim, 10, 10))
    vsim = ksim * (esim ^ 2) ^ psim

    # Randomly simulate future averages
    temp = array(rnorm(nsim * 10 * 10, c(esim), sqrt(c(vsim))), c(nsim, 10, 10))

    # Combine to total by exposure period and in aggregate
    # notice separate array with name ending in "n" to capture
    # forecast for next accounting period
    sdnm = t(matrix(dnom, 10, nsim))
    fore = sdnm * rowSums(temp * !smsk, dims = 2)
    forn = sdnm * rowSums(temp * smsn, dims = 2)

    # Cumulate and return for another 5,000
    sim = rbind(sim, cbind(fore, rowSums(fore)))
    smn = rbind(smn, cbind(forn, rowSums(forn)))
    spm = rbind(spm, spar)
  }
}
```

Print Results

```r
model
```

```
## [1] "CapeCod"
```

```r
model_description(model)
```

```
## [1] "Cape Cod"
```

```r
summary(sim)
```

```
##        V1          V2                  V3                  V4
##  Min.   :0   Min.   :-3131534   Min.   :-3355892   Min.   :-2357200
##  1st Qu.:0   1st Qu.:  179389   1st Qu.:  546055   1st Qu.: 2732549
##  Median :0   Median :  610105   Median : 1121175   Median : 3678811
##  Mean   :0   Mean   :  671936   Mean   : 1149286   Mean   : 3703355
##  3rd Qu.:0   3rd Qu.: 1098939   3rd Qu.: 1721977   3rd Qu.: 4654189
##  Max.   :0   Max.   : 4703800   Max.   : 5305053   Max.   : 9887583
##        V5                 V6                 V7
##  Min.   : -178238   Min.   : 8941253   Min.   :26672404
##  1st Qu.: 6423460   1st Qu.:17158336   1st Qu.:40036688
##  Median : 7689605   Median :19019559   Median :42770332
##  Mean   : 7694355   Mean   :19019577   Mean   :42814287
##  3rd Qu.: 8952854   3rd Qu.:20864018   3rd Qu.:45546543
##  Max.   :16316983   Max.   :32940749   Max.   :61021297
##        V8                  V9                 V10
##  Min.   : 54841910   Min.   : 63824172   Min.   : 87538902
##  1st Qu.: 73235936   1st Qu.: 87429567   1st Qu.:137770116
##  Median : 77089892   Median : 92305276   Median :146677019
##  Mean   : 77125659   Mean   : 92421447   Mean   :146719666
##  3rd Qu.: 80943825   3rd Qu.: 97335959   3rd Qu.:155664979
##  Max.   :100722736   Max.   :123302293   Max.   :195353892
##       V11
##  Min.   :308412796
##  1st Qu.:377772994
##  Median :391401200
##  Mean   :391319567
##  3rd Qu.:404806253
##  Max.   :477432628
```

```r
summary(smn)
```

```
##        V1          V2                  V3                  V4
##  Min.   :0   Min.   :-3131534   Min.   :-2152986   Min.   :-1923258
##  1st Qu.:0   1st Qu.:  179389   1st Qu.:  107295   1st Qu.: 1629315
##  Median :0   Median :  610105   Median :  411681   Median : 2290307
##  Mean   :0   Mean   :  671936   Mean   :  446646   Mean   : 2319646
##  3rd Qu.:0   3rd Qu.: 1098939   3rd Qu.:  761132   3rd Qu.: 2969085
##  Max.   :0   Max.   : 4703800   Max.   : 3419724   Max.   : 7397148
##        V5                 V6                 V7
##  Min.   : -908291   Min.   : 3174720   Min.   :11561769
##  1st Qu.: 3100934   1st Qu.: 9508082   1st Qu.:20302869
##  Median : 3894930   Median :10746265   Median :22013486
##  Mean   : 3919449   Mean   :10771881   Mean   :22059626
##  3rd Qu.: 4714187   3rd Qu.:12007446   3rd Qu.:23796741
```

```
##  Max.   :10042460   Max.   :19309684   Max.   :33849329
##        V8                  V9                 V10
##  Min.   :21109920   Min.   :20047148   Min.   :25244932
##  1st Qu.:32286137   1st Qu.:31129358   1st Qu.:38918262
##  Median :34415052   Median :33377623   Median :41977680
##  Mean   :34464398   Mean   :33447397   Mean   :42049782
##  3rd Qu.:36637687   3rd Qu.:35726477   3rd Qu.:45155484
##  Max.   :50010785   Max.   :47949173   Max.   :59744054
##       V11
##  Min.   :117459058
##  1st Qu.:145055032
##  Median :150171289
##  Mean   :150150760
##  3rd Qu.:155163275
##  Max.   :182277910
```

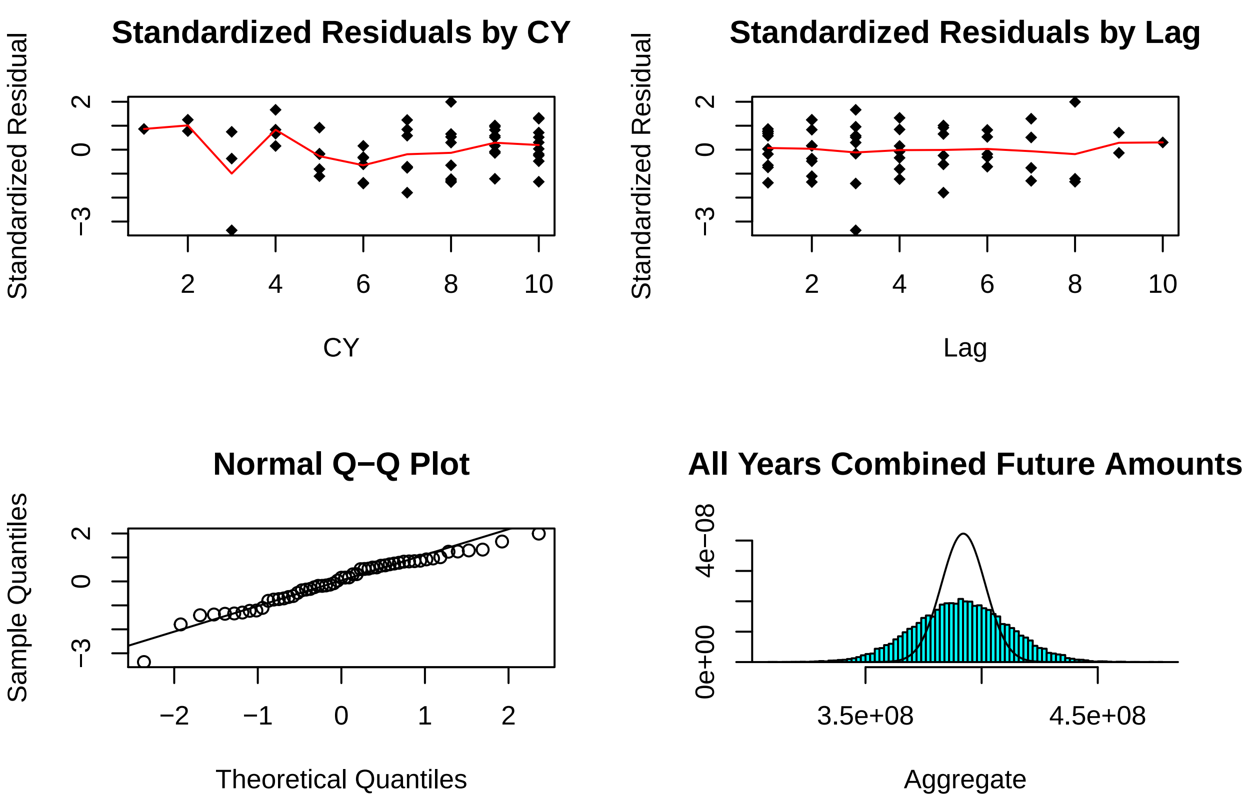
Plots

```
# Scatter plots of residuals & Distribution of Forecasts
if (graphs) {
  #x11(title = model_description(model))

  # Prep data for lines for averages in scatter plots of standardized residuals
  ttt = array(cbind(c(rowNum + colNum - 1), c(stres)), c(length(c(stres)), 2, 19))
  sss = t(array((1:19), c(19, length(c(
    stres
  )))))

  # Plotting
  par(mfrow = c(2, 2))
  plot(
    na.omit(cbind(c(rowNum + colNum - 1), c(stres))),
    main = "Standardized Residuals by CY",
    xlab = "CY",
    ylab = "Standardized Residual",
    pch = 18
  )
  lines(na.omit(list(
    x = (1:19),
    y = colSums(ttt[, 2, ] *
                  (ttt[, 1, ] == sss), na.rm = TRUE) /
      colSums((ttt[, 1, ] == sss) +
                0 *
                ttt[, 2, ], na.rm = TRUE)
  )), col = "red")
  plot(
    na.omit(cbind(c(colNum), c(stres))),
    main = "Standardized Residuals by Lag",
    xlab = "Lag",
    ylab = "Standardized Residual",
    pch = 18
  )
  lines(na.omit(list(
```

```
    x = colNum[1, ],
    y = colSums(stres, na.rm = TRUE) /
      colSums(1 + 0 * stres, na.rm = TRUE)
  )), col = "red")
  qqnorm(c(stres))
  qqline(c(stres))
  if (simulation) {
    proc = list(x = (density(sim[, 11]))$x,
                y = dnorm((density(sim[, 11]))$x,
                          sum(matrix(c(
                            dnom
                          ), 10, 10) * mean * !msk),
                          sqrt(sum(
                            matrix(c(dnom), 10, 10) ^ 2 * var * !msk
                          ))))
    truehist(sim[, 11],
             ymax = max(proc$y),
             main = "All Years Combined Future Amounts",
             xlab = "Aggregate")
    lines(proc)
  }
}
```



Summary From Simulation

Summary of mean, standard deviation, and 90% confidence interval from simulation, similar for one-period forecast

```
sumr = matrix(0, 0, 4)
sumn = matrix(0, 0, 4)
```

```r
for (i in 1:11) {
  sumr = rbind(sumr, c(mean(sim[, i]), sd(sim[, i]), quantile(sim[, i], c(.05, .95))))
  sumn = rbind(sumn, c(mean(smn[, i]), sd(smn[, i]), quantile(smn[, i], c(.05, .95))))
}
sumr
```

```
##                                           5%       95%
##  [1,]         0.0        0.0         0.0         0
##  [2,]    671935.7   696372.3   -328742.7   1913459
##  [3,]   1149285.8   887330.3   -229548.5   2636824
##  [4,]   3703355.0  1441210.3   1383228.2   6122593
##  [5,]   7694355.3  1908103.6   4583735.6  10851955
##  [6,]  19019576.6  2766535.1  14501788.3  23564999
##  [7,]  42814286.9  4105863.4  36105669.6  49615927
##  [8,]  77125659.1  5764244.3  67831387.6  86618553
##  [9,]  92421446.6  7344752.8  80460410.3 104609228
## [10,] 146719666.4 13323202.6 125044396.2 168857160
## [11,] 391319567.3 20001970.5 358361425.8 424176619
```

```r
sumn
```

```
##                                         5%       95%
##  [1,]         0.0       0.0         0.0         0
##  [2,]    671935.7  696372.3   -328742.7   1913459
##  [3,]    446645.6  505592.3   -306994.3   1308757
##  [4,]   2319645.9 1014486.1    718891.6   4041066
##  [5,]   3919449.1 1208283.5   1981100.0   5932877
##  [6,]  10771880.6 1855458.3   7764707.0  13840854
##  [7,]  22059626.1 2604582.5  17801922.4  26376612
##  [8,]  34464398.5 3253111.1  29188860.5  39898568
##  [9,]  33447396.6 3426905.4  27938660.6  39151741
## [10,]  42049781.6 4612965.9  34557653.4  49738153
## [11,] 150150759.7 7513268.4 137847708.4 162547512
```