CAS Annual Meeting 2008

Loss Reserving with R

Markus Gesmann, Lloyd's of London
Vincent Goulet, Université Laval
Daniel Murphy, Trinostics

November 18, 2008

Agenda

- Introduction to R (15min) Vincent
- The *actuar* package (15min) Vincent
- Exchanging data between R and Excel (10min) Dan
- The *ChainLadder* package (30min) Markus
- The *copula* package (5min) Dan
- Q&A (15min)

CAS Annual Meeting 2008

Introduction to R and *actuar*

Vincent Goulet
Université Laval

November 18, 2008

R ?

- R is a language and environment for statistical computing and graphics
- Free/Open Source implementation of S
  - R is *not unlike S-PLUS*
- Some important differences, but most code written for S-PLUS runs unaltered in R
- Designed around a true programming language
- Comprehensive statistical system

The R Programming Language

- Strong mathematical orientation
  - Inspired by APL and Lisp
- Extensive suite of operators for calculations on vectors and arrays
- Rapid development
- Allows users to add functionality
- C, C++ and Fortran code can be linked and called at run time

The Statistical System

- Integrated software facilities for data manipulation, calculation and graphical display
- Extensive collection of tools for data analysis
- Outstanding graphical facilities

Other Advantages

- Very active project
- High quality and cutting edge code
- Open for everyone to contribute
- Runs on Windows, Linux/Unix and MacOS
- Free

R is Like Guiness

R Package

- Coherent collection of functions, data sets and documentation
- Distributed through the Comprehensive R Archive Network (CRAN)
- *actuar*: tools typically used in Actuarial Science
- *ChainLadder*: loss reserving using the chain-ladder method

Fields Currently Covered by *actuar*

- Loss distributions modeling
- Risk and ruin theory
- Simulation of compound hierarchical models
- Credibility theory

New Probability Laws

- Support for 18 probability laws not in base R
  - Transformed beta family (9)
  - Transformed gamma family (5)
  - Loggamma
  - Single parameter Pareto
  - Generalized beta
  - Phase-type
- Usual R functions `dfoo`, `pfoo`, `qfoo` and `rfoo`
- Package adds
  - `mfoo` (raw moments)
  - `levfoo` (limited moments)
  - `mgffoo` (moment generating function)

Estimation

- `emm` to compute the k^{th} empirical moment
- `elev` to compute the empirical limited expected value
- `mde` for minimum distance estimation
- `coverage` to compute the density function under coverage modifications (censored data)

Support for Grouped Data

- Creation and manipulation of grouped data objects

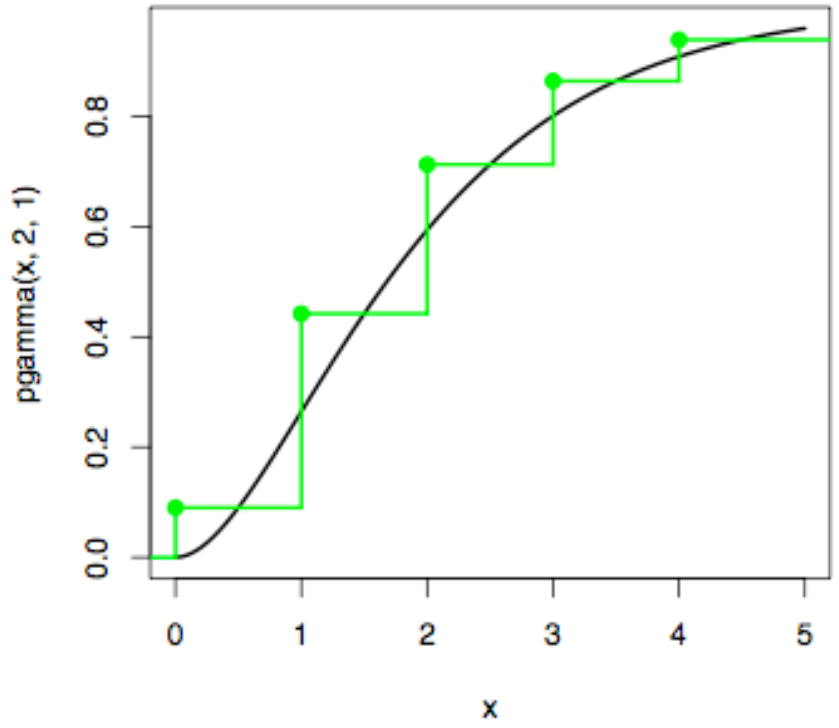
```
> grouped.data(Group = c(0, 25, 50, 100),
               Line.1 = c(30, 31, 57),
               Line.2 = c(26, 33, 31))

      Group Line.1 Line.2
1   (0,  25]     30     26
2  (25,  50]     31     33
3  (50, 100]     57     31
```

- Calculation of empirical moments
- Plot of the ogive and histogram

Discretization of Continuous Distributions

```
> discretize(pgamma(x, 2, 1), from = 0, to = 5,
                    method = "rounding")
```



Aggregate Claim Amount Distribution

- Compute the cdf $F_S(x)$ of

$$S = C_1 + \cdots + C_N$$

- Supported methods:
  - recursive (Panjer algorithm)
  - convolutions
  - simulation
  - normal approximation
  - normal power approximation
- Output is a function to compute $F_S(x)$ in any x

Example

- Assume

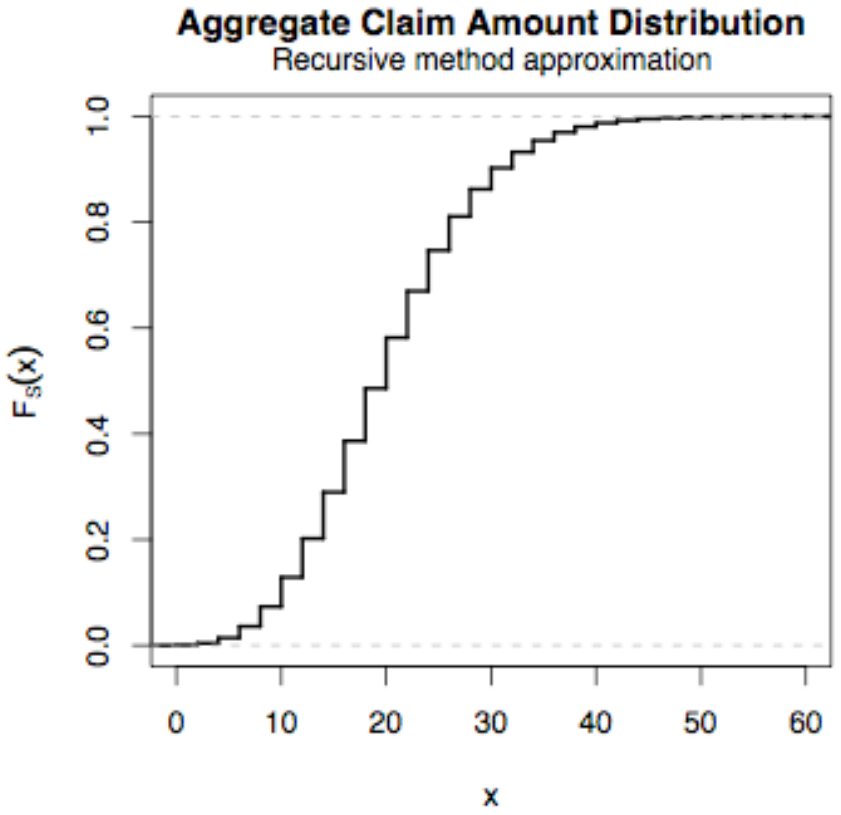
$$N \sim \text{Poisson}(10)$$
$$C \sim \text{Gamma}(2, 1)$$

```
> fx <- discretize(pgamma(x, 2, 1), from = 0,
                   to = 22, step = 2,
                   method = "unbiased",
                   lev = levgamma(x, 2, 1))

> Fs <- aggregateDist("recursive",
                      model.freq = "poisson",
                      model.sev = fx,
                      lambda = 10, x.scale = 2)
```

Example (continued)

```
> plot(Fs)
```



Example (continued)

```
> Fs(c(10, 15, 20, 70))
  [1] 0.1288 0.2897 0.5817 1.0000
> mean(Fs)
  [1] 20
> VaR(Fs)
  90% 95% 99%
   30  34  42
> CTE(Fs)
    90%   95%   99%
  35.08 36.10 36.37
```

Ruin Probabilities and Adjustment Coefficient

- `ruin` and `adjCoef`
- Claim amounts and waiting times can be
  - mixtures of exponentials
  - mixtures of Erlang
  - phase-type
- In most cases probabilities computed with `pphtype`
- Output is a function to compute $\psi(u)$ in any u

Simulation of Compound Hierarchical Models

- Consider

$$S = C_1 + \cdots + C_N ,$$

with

$$N|\Lambda, \Phi \sim \text{Poisson}(w \times \Lambda)$$
$$\Lambda|\Phi \sim \text{Gamma}(\Phi, 1)$$
$$\Phi \sim \text{Exponential}(2)$$

$$C|\Theta, \Psi \sim \text{Lognormal}(\Theta, 1)$$
$$\Theta|\Psi \sim N(\Psi, 1)$$
$$\Psi \sim N(2, 0.1)$$

Solution

```
> w <- runif(18, 0.5, 2.5)

> nodes <- list(class = 2, contract = c(3, 2),
                year = c(4, 4, 4, 3, 3))

> mf <- expression(class = rexp(2),
                   contract = rgamma(class, 1),
                   year = rpois(weights * contract))

> ms <- expression(class = rnorm(2, sqrt(0.1)),
                   contract = rnorm(class, 1),
                   year = rlnorm(contract, 1))

> pf <- simul(nodes = nodes, model.freq = mf,
              model.sev = ms, weights = w)
```

Portfolio Object

```
> pf
Portfolio of claim amounts


  Frequency model
    class    ~ rexp(2)
    contract ~ rgamma(class, 1)
    year     ~ rpois(weights * contract)
   Severity model
    class    ~ rnorm(2, sqrt(0.1))
    contract ~ rnorm(class, 1)
    year     ~ rlnorm(contract, 1)

   Number of claims per node:

        class contract year.1 year.2 year.3 year.4
   [1,]     1        1      0      1      0      3
   [2,]     1        2      0      0      0      1
   ...
```

Extraction of Aggregate Amounts and Frequencies

```
> aggregate(pf)
     class contract year.1 year.2 year.3 year.4
[1,]     1        1   0.00  3.770      0  51.27
[2,]     1        2   0.00  0.000      0 100.15
[3,]     1        3   0.00  0.000      0   0.00
[4,]     2        1   0.00  0.000      0     NA
[5,]     2        2  16.61 72.448      0     NA
```

```
> frequency(pf)
     class contract year.1 year.2 year.3 year.4
[1,]     1        1      0      1      0      3
[2,]     1        2      0      0      0      1
[3,]     1        3      0      0      0      0
[4,]     2        1      0      0      0     NA
[5,]     2        2      1      3      0     NA
```

Extraction of Individual Claim Amounts

```
> severity(pf)

$first
      class contract claim.1 claim.2 claim.3 claim.4
 [1,]     1        1   3.770   10.37   6.931   33.97
 [2,]     1        2 100.152      NA      NA      NA
 [3,]     1        3      NA      NA      NA      NA
 [4,]     2        1      NA      NA      NA      NA
 [5,]     2        2  16.606   21.59  10.762   40.10


$last

NULL
```

Credibility Theory

- One function to rule them all: `cm`
- Models currently supported:
  - Bühlmann
  - Bühlmann–Straub
  - Jewell (hierarchical)
  - Hachemeister (regression)
- Three sets of estimators
- Function returns a fitted model
- Use `predict` to get credibility premiums

More Information

- Project's web site
  - `http://www.actuar-project.org`
- Package vignettes
  - `actuar` Introduction to actuar
  - `coverage` Formulas used by `coverage`
  - `credibility` Credibility theory features
  - `lossdist` Loss distributions features
  - `risk` Risk theory features
- Demo files

CAS Annual Meeting 2008

Exchanging data between R and Excel

Daniel Murphy
Trinostics

November 18, 2008

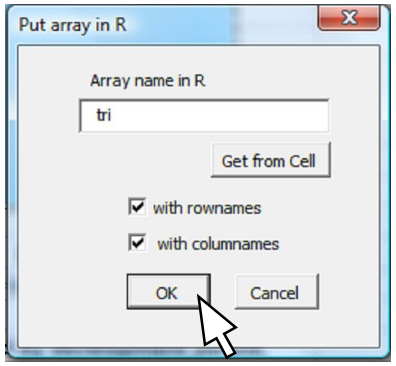
RExcel extends Excel's functionality

- Download and install RExcel
  - http://sunsite.univie.ac.at/rcom/
  - For step-by-step screen shots installing under Vista, see link at http://www.trinostics.com/Projects
- In Excel: Tools, Add-Ins, ☑RExcel, OK
  - Excel's functionality is now extended by R
- Three ways to use R from Excel (see RExcel, RExcel Help)
  - "Scratchpad Mode": Use the RExcel Menu
  - "Worksheet Mode": use RExcel functions within cells
  - "Macro Mode": Visual Basic (not covered here)
- Three most common uses of RExcel
  - Send cells to R for calculations
  - Retrieve variables from R for use in Excel
  - Execute R functions unavailable or too complicated in Excel

Send a triangle to R for statistical analysis

	A	B	C	D	E	F	G	H	I	J	K
1	Accident	Age (months)									
2	Year	12	24	36	48	60	72	84	96	108	120
3	1998	5012	8269	10907	11805	13539	16181	18009	18608	18662	18834
4	1999	106	4285	5396	10666	13782	15599	15496	16169	16704	
5	2000	3410	8992	13873	16141	18735	22214	22863	23466		
6	2001	5655	11555	15766	21266	23425	26083	27067			
7	2002	1092	9565	15836	22169	25955	26180				
8	2003	1513	6445	11702	12935	15852					
9	2004	557	4020	10946	12314						
10	2005	1351	6947	13112							
11	2006	3133	5395								
12	2007	2063									

- Highlight cells to send to R
  - Click RExcel, Put R Var, Array, give it a name (say, tri)
  - Click ☑ with rownames and ☑ with columnames if you also highlighted the accident years and ages, respectively
  - Can also right-click, Put R Var, etc.



Can run regressions on triangle in R

- In R:

```
> tri
      12    24    36    48    60    72    84    96   108   120
1998 5012  8269 10907 11805 13539 16181 18009 18608 18662 18834
1999  106  4285  5396 10666 13782 15599 15496 16169 16704    NA
2000 3410  8992 13873 16141 18735 22214 22863 23466    NA    NA
2001 5655 11555 15766 21266 23425 26083 27067    NA    NA    NA
2002 1092  9565 15836 22169 25955 26180    NA    NA    NA    NA
2003 1513  6445 11702 12935 15852    NA    NA    NA    NA    NA
2004  557  4020 10946 12314    NA    NA    NA    NA    NA    NA
2005 1351  6947 13112    NA    NA    NA    NA    NA    NA    NA
2006 3133  5395    NA    NA    NA    NA    NA    NA    NA    NA
2007 2063    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

- Do a weighted regression of all rows of column 1 on column 2, with a zero intercept
  - Slope of line (“Coefficient”) is the volume weighted average link ratio = 2.999
  - Note that missing values are automatically excluded from the regression

```
> lm(tri[,2]~tri[,1]+0,weights=1/tri[,1])
Call:
lm(formula = tri[, 2] ~ tri[, 1] + 0, weights = 1/tri[, 1])
Coefficients:
tri[, 1]
   2.999
```

Can also send cells to R as a "dataframe" with RExcel, or without RExcel by using the clipboard

- Highlight cells to send to R
- With RExcel
  - Click RExcel, Put R Var, Dataframe, give it a name (say, tri)
  - Click ☑ with rownames if you also highlighted the accident years
  - Variable is automatically created in R
- Without RExcel
  - Edit, Copy (or ctrl-C) to copy to clipboard
  - In R:

```
> tri<-read.table(file="clipboard",header=TRUE,row.names=1,sep="\t")
```

- Notes
  - `header=TRUE` if you highlighted column headings, otherwise FALSE
  - `row.names=1` if the first column contains the names of the accident years, otherwise omit
  - `sep="\t"` because Windows sends cells to clipboard separated by tab characters
  - `read.tables` always produce dataframes

Dataframes are analogous to databases – think 'Access table' with columns of named fields

- Now *dataframe* tri is just like *array* tri, but new column names

```
> tri
      X12   X24   X36   X48   X60   X72   X84   X96  X108  X120
1998 5012  8269 10907 11805 13539 16181 18009 18608 18662 18834
1999  106  4285  5396 10666 13782 15599 15496 16169 16704    NA
2000 3410  8992 13873 16141 18735 22214 22863 23466    NA    NA
2001 5655 11555 15766 21266 23425 26083 27067    NA    NA    NA
2002 1092  9565 15836 22169 25955 26180    NA    NA    NA    NA
2003 1513  6445 11702 12935 15852    NA    NA    NA    NA    NA
2004  557  4020 10946 12314    NA    NA    NA    NA    NA    NA
2005 1351  6947 13112    NA    NA    NA    NA    NA    NA    NA
2006 3133  5395    NA    NA    NA    NA    NA    NA    NA    NA
2007 2063    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

- Specifying the linear regression model is cleaner, clearer

```
> lm(X24~X12+0,weights=1/X12,data=tri)
Call:
lm(formula = X24 ~ X12 + 0, data = tri, weights = 1/X12)

Coefficients:
  X12
2.999       # same result
```

Can use RExcel or clipboard to send data R -> Excel

- Let's look at Glenn Meyers' *EForum* data (Bayesian analysis of the collective risk model using R)

```
> GM<-read.csv("The Rectangle.csv"); GM  # it's a dataframe
   premium ay lag  loss
1    50000  1   1  7168      # R knows how to work with this common
2    50000  1   2 11190      # 'long' format of data storage
...
54   50000  9   2  8898
55   50000 10   1  4824
```

- Convert to a triangle ('wide') format using built-in `reshape` function

```
> L = reshape(GM[-1], idvar='ay', timevar='lag', v.names='loss',
              direction='wide')
```

  - Ie, ignore 1st 'field'; "idvar='ay'" ≡ new ay ⇨ new row, "timevar= 'lag'" ≡ new lag ⇨ new column; v.names≡variables to keep in each column

```
> L
   ay loss.1 loss.2 loss.3 loss.4 loss.5 loss.6 loss.7 loss.8 loss.9 loss.10
1   1   7168  11190  12432   7856   3502   1286    334    216    190       0
11  2   4770   8726   9150   5728   2459   2864    715    219      0      NA
20  3   5821   9467   7741   3736   1402    972    720     50     NA      NA
 ...
55 10   4824     NA     NA     NA     NA     NA     NA     NA     NA      NA
```

- Evidently, Glenn's losses are incremental amounts

Convert incremental amounts to cumulative triangle

- Accumulate values across the rows to get cumulative triangle

```
> Lsum=apply(L[-1],1,cumsum) # Ie, "Ignoring 1st (ay) field, apply
   the cumulative sum function to 1st dimension (row) of array L"
```

  - Rows are cumulated one after the other, but stored in FORTRAN order (by columns), so transpose the matrix

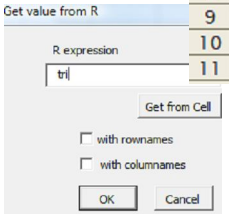
```
> tri=t(Lsum)
```

- Either send triangle to Excel using clipboard (preferred) …

```
> write.table(tri,file="clipboard",sep="\t",row.names=FALSE,na="")
```

  - In Excel: Edit, Paste

- … or retrieve triangle with RExcel
  - In Excel: RExcel, Get R Value, array, tri



	A	B	C	D	E	F	G	H	I	J
1	loss.1	loss.2	loss.3	loss.4	loss.5	loss.6	loss.7	loss.8	loss.9	loss.10
2	7168	18358	30790	38646	42148	43434	43768	43984	44174	44174
3	4770	13496	22646	28374	30833	33697	34412	34631	34631	
4	5821	15288	23029	26765	28167	29139	29859	29909		
5	5228	12278	18855	21745	23345	25501	26093			
6	4185	10758	15954	18823	22432	23715				
7	4930	12964	18279	23828	25719					
8	4936	12293	18110	23388						
9	4762	13145	19713							
10	5025	13923								
11	4824									

Can run R functions from Excel cells

- Define an R function to calculate weighted average link ratios

```
> WtdAvg <- function(x,y) lm(y~x+0,weights=1/x)$coef
```

	A	B	C	D	E	F	G	H	I	J	K
1	Accident	Age (months)									
2	Year	12	24	36	48	60	72	84	96	108	120
3	1998	5,012	8,269	10,907	11,805	13,539	16,181	18,009	18,608	18,662	18,834
4	1999	106	4,285	5,396	10,666	13,782	15,599	15,496	16,169	16,704	
5	2000	3,410	8,992	13,873	16,141	18,735	22,214	22,863	23,466		
6	2001	5,655	11,555	15,766	21,266	23,425	26,083	27,067			
7	2002	1,092	9,565	15,836	22,169	25,955	26,180				
8	2003	1,513	6,445	11,702	12,935	15,852					
9	2004	557	4,020	10,946	12,314						
10	2005	1,351	6,947	13,112							
11	2006	3,133	5,395								
12	2007	2,063									
13											
14	Wtd Avg	2.999	1.624	1.271	1.172	1.113	1.042	1.033	1.017	1.009	

⇧ =RApply("WtdAvg",B3:B12,C3:C12)

- Copy 'RApply' formula in B14 across row 14 w/o worrying about which AY ends that period's average! ☺
- For many other examples of running R code from Excel, see ChainLadder_in_Excel.xls

Sending R graph to Powerpoint w/o clipboard

- Run the Mack model from the ChainLadder package

```
> M <- MackChainLadder(RAA)
```

- Plot the results to a virtual Windows graphics device

```
> plotMack=tempfile()
> win.metafile(file=plotMack)
> plot(M)
> dev.off()
```

- Now let's put the graphs into PowerPoint
  - Start Powerpoint, create a blank presentation, add a slide to that presentation

```
> ppt<-comCreateObject("Powerpoint.Application")
> comSetProperty(ppt,"Visible",TRUE)
> myPresColl<-comGetProperty(ppt,"Presentations")
> myPres<-comInvoke(myPresColl,"Add")
> mySlides<-comGetProperty(myPres,"Slides")
> mySlide<-comInvoke(mySlides,"Add",1,12)
> myShapes<-comGetProperty(mySlide,"Shapes")
```

  - Add a “shape” to slide (“shape” = picture on virtual graphics device)

```
> myPicture<-
   comInvoke(myShapes,"AddPicture",plotMack,0,1,100,10)
```

CAS Annual Meeting 2008

The *ChainLadder* package in R

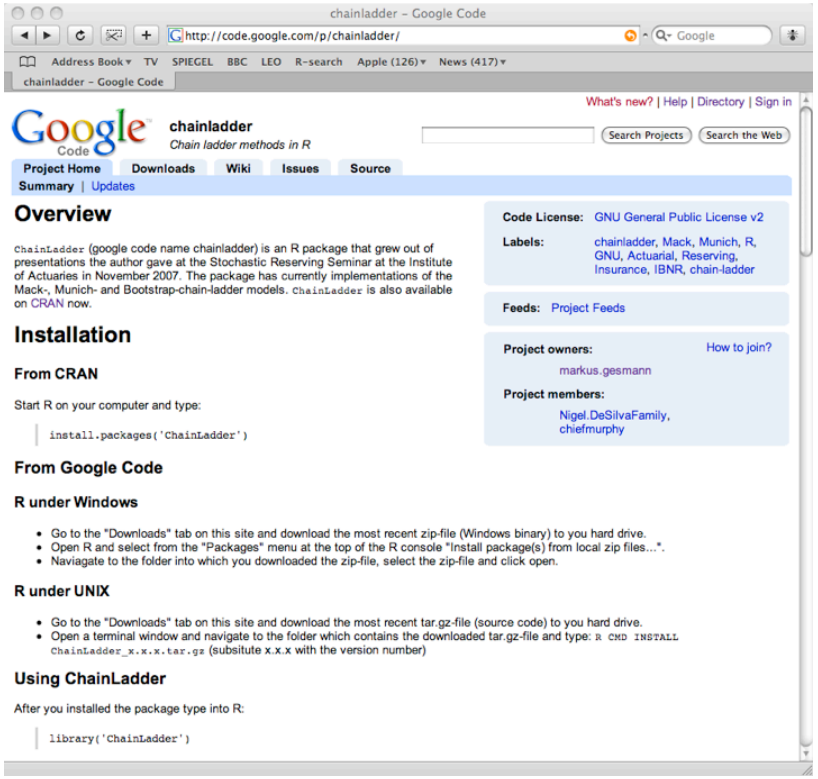
Markus Gesmann
Lloyd's of London

November 18, 2008

The *ChainLadder* package

Agenda:

- Getting started
- *ChainLadder* package philosophy
- Examples for
  - *MackChainLadder*
  - *MunichChainLadder*
  - *BootChainLadder*



Project web page: http://code.google.com/p/chainladder/
Current version: 0.1.2-7

Starting position

- Over recent years stochastic methods have been developed and published, but have been rarely used in practice
- Excel is the standard tool in the industry, but is not an ideal environment for implementing those stochastic methods
- Idea: Use R to implement stochastic reserving methods, and CRAN to share them
- Use the RExcel Add-in as a front end for Excel to use R functions

The *ChainLadder* package for R

- Started out of presentations given at the Institute of Actuaries on stochastic reserving
- Mack-, Munich- and Bootstrap-chain-ladder implemented; Log-normal model in experimental stage
- Example spreadsheet shows how to use the functions within Excel using the RExcel Add-in
- Available from CRAN - sources and binaries
- Home page: http://code.google.com/p/chainladder/
- Contribution most welcome!

Getting started

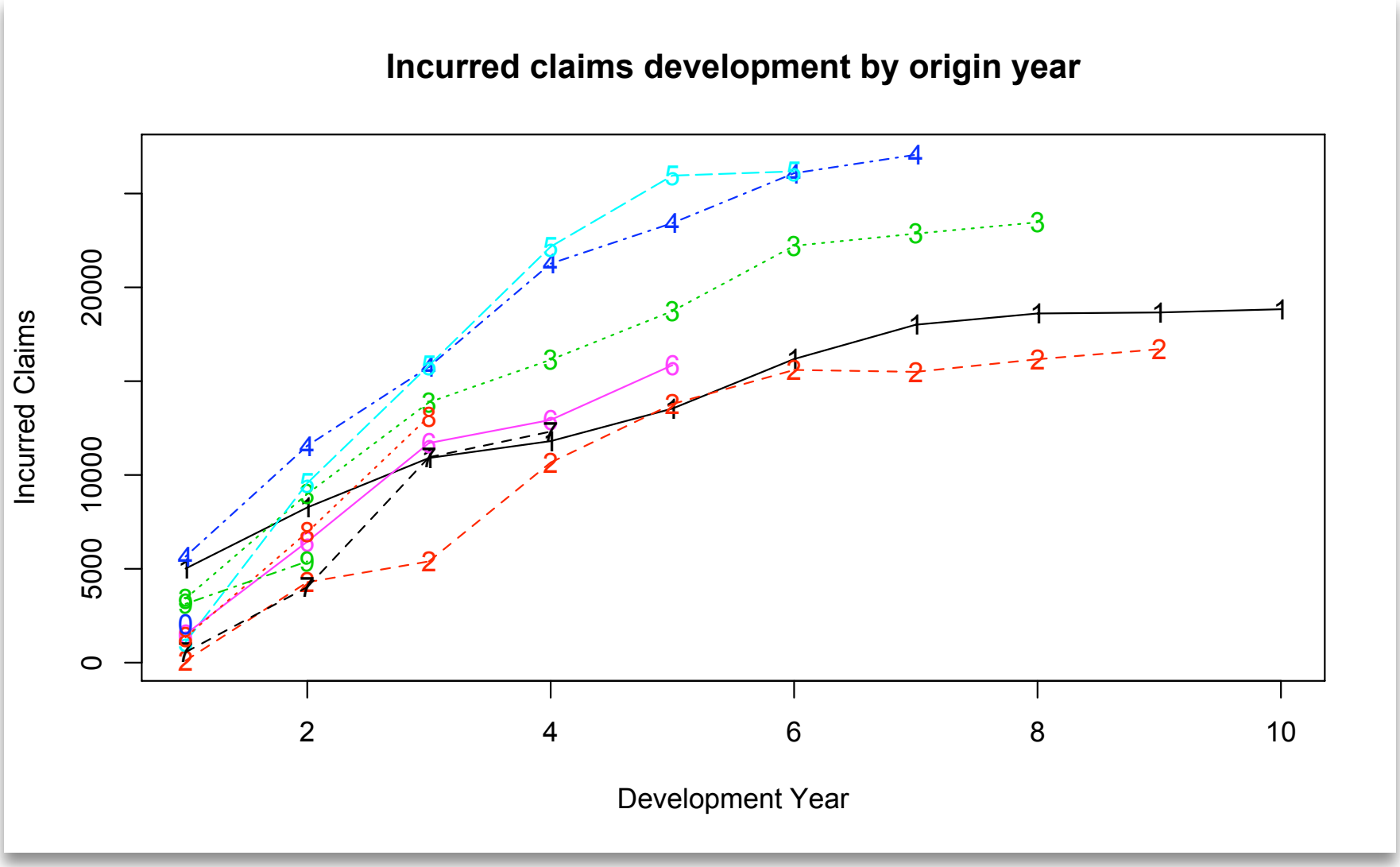
- Start R and type for
  - Installation:
    `install.packages("ChainLadder")`
  - Loading the package:
    `library(ChainLadder)`
  - Help:
    `?ChainLadder`
  - Examples:
    `example(ChainLadder)`

Example data sets

- The *ChainLadder* package comes with some example data sets, e.g.

```
> library(ChainLadder)
> RAA
      dev
origin    1     2     3     4     5     6     7     8     9    10
  1981 5012  8269 10907 11805 13539 16181 18009 18608 18662 18834
  1982  106  4285  5396 10666 13782 15599 15496 16169 16704    NA
  1983 3410  8992 13873 16141 18735 22214 22863 23466    NA    NA
  1984 5655 11555 15766 21266 23425 26083 27067    NA    NA    NA
  1985 1092  9565 15836 22169 25955 26180    NA    NA    NA    NA
  1986 1513  6445 11702 12935 15852    NA    NA    NA    NA    NA
  1987  557  4020 10946 12314    NA    NA    NA    NA    NA    NA
  1988 1351  6947 13112    NA    NA    NA    NA    NA    NA    NA
  1989 3133  5395    NA    NA    NA    NA    NA    NA    NA    NA
  1990 2063    NA    NA    NA    NA    NA    NA    NA    NA    NA
```

Triangle plot




```
> matplot(t(RAA), t="b")
```

Working with triangles

- Transform from cumulative to incremental

```
incRAA <- cbind(RAA[,1], t(apply(RAA,1,diff)))
```

- Transform from incremental to cumulative

```
cumRAA <- t(apply(incRAA,1, cumsum))
```

- Triangles to long format

```
lRAA <- expand.grid(origin=as.numeric(dimnames(RAA)
$origin), dev=as.numeric(dimnames(RAA)$dev))

lRAA$value <- as.vector(RAA)
```

- Long format to triangle

```
reshape(x, timevar="dev", idvar="origin",
v.names="value", direction="wide")
```

ChainLadder package philosophy

- Use the linear regression function `"lm"` as much as possible and utilise its output
- The chain-ladder model for volume weighted average link ratios is expressed as a formula:

```
y ~ x + 0, weights=1/x
```

  and can easily be changed
- E.g. Mack's formula needs s.e(f), sigma which fall out of the linear regression directly
- Provide tests for the model assumptions

Chain-ladder as linear regression

Chain-ladder can be regarded as weighted linear regression through the origin:

```
x <- RAA[,1] # dev. period 1
y <- RAA[,2] # dev. period 2

model <- lm(y ~ x + 0, weights=1/x)

Call:
lm(formula = y ~ x + 0, weights = 1/x)

Coefficients:
     x
2.999
```

Force it through the origin

Volume weighted

chain-ladder link-ratio

Full regression output

```
> summary(model)

Call:
lm(formula = y ~ x + 0, weights = 1/x)

Residuals:
    Min      1Q  Median      3Q     Max
-95.54 -71.50   49.03   99.55 385.32

Coefficients:
  Estimate Std. Error t value Pr(>|t|)
x    2.999      1.130   2.654   0.0291 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 167 on 8 degrees of freedom
Multiple R-squared: 0.4682,      Adjusted R-squared: 0.4017
F-statistic: 7.043 on 1 and 8 DF,  p-value: 0.02908
```

The output shows:

- model formula
- chain-ladder link ratio
- std. error of the link ratio
- P-value
- Residual std. error

Chain-ladder using the "lm" function

Idea: Create linear model for each development period

```
ChainLadder <- function(Triangle, weights=1/Triangle){
    n <- ncol(Triangle)
    myModel <- vector("list", (n-1))
    for(i in c(1:(n-1))){
        dev.data <- data.frame(x=Triangle[,i],
                               y=Triangle[,i+1])
      myModel[[i]] <- lm(y~x+0,
                         weights=weights[,i],
                         data=dev.data)
    }
    return(myModel)
  }
```

Accessing regression statistics

```
CL <- ChainLadder(RAA)

# Get chain-ladder link-ratios
sapply(CL, coef)
# 2.999359 1.623523 1.270888 1.171675 1.113385
# 1.041935 1.033264 1.016936 1.009217

# Get residual standard errors
sapply(lapply(CL, summary), "[[", "sigma")
# 166.983470  33.294538  26.295300   7.824960  10.928818
#   6.389042   1.159062   2.807704         NaN

# Get R squared values
sapply(lapply(ChainLadder(RAA), summary), "[[",
"r.squared")
# 0.4681832 0.9532872 0.9704743 0.9976576 0.9959779
# 0.9985933 0.9999554 0.9997809 1.0000000
```

Mack-chain-ladder

Mack's chain-ladder method calculates the standard error for the reserves estimates.

The method works for a cumulative triangle C_{ik} if the following assumptions are hold:

- $$E\left[\frac{C_{i,k+1}}{C_{ik}}|C_{i1}, C_{i2}, \ldots, C_{ik}\right] = f_k$$
- $$\text{Var}\left(\frac{C_{i,k+1}}{C_{ik}}|C_{i1}, C_{i2}, \ldots, C_{ik}\right) = \frac{\sigma_k^2}{C_{ik}}$$
- All accident years are independent

MackChainLadder

Usage:

```
MackChainLadder(Triangle,
                weights = 1/Triangle,
                tail=FALSE,
                est.sigma="log-linear")
```

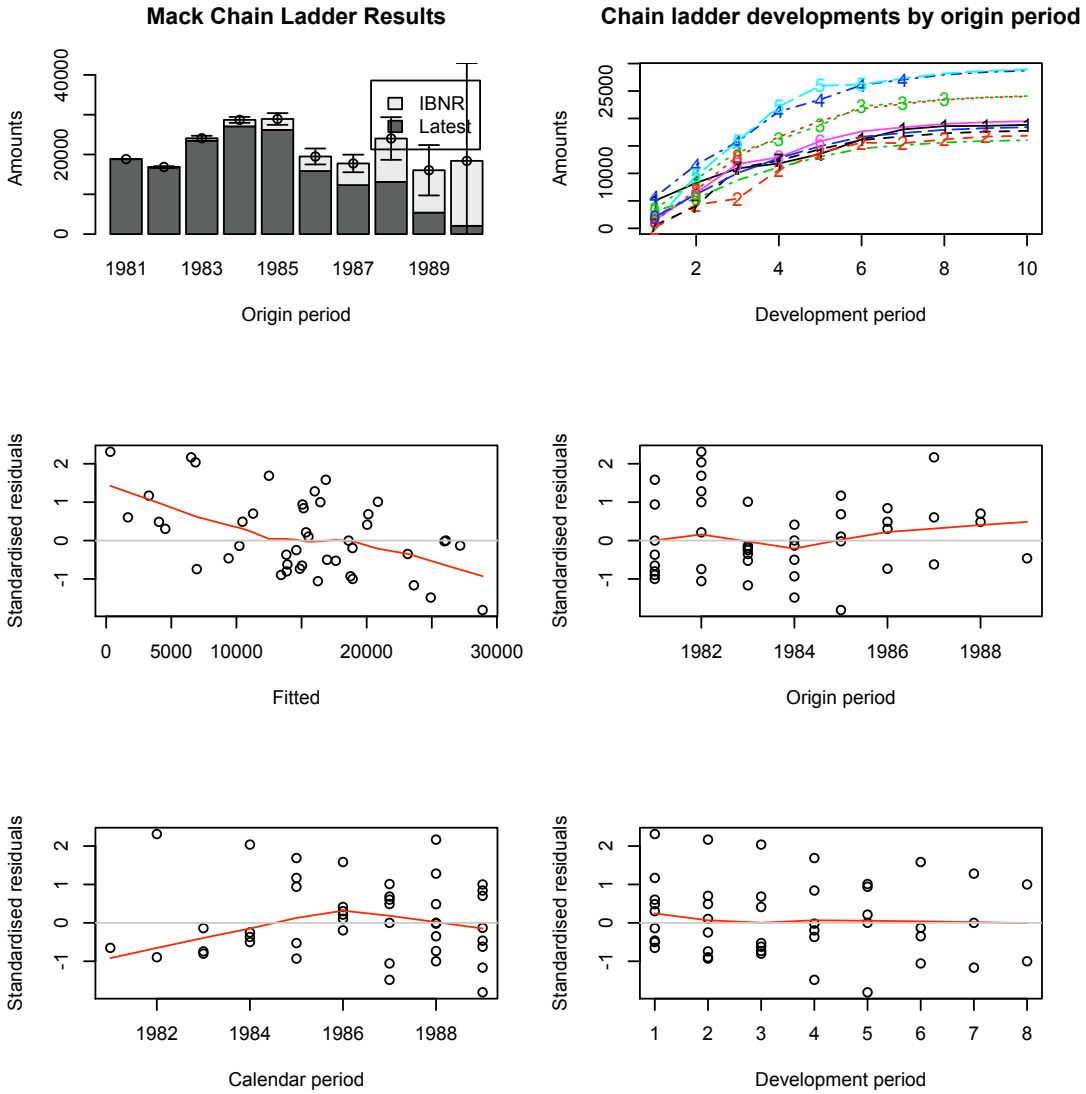
- Triangle: cumulative claims triangle
- weights: default (1/Triangle) volume weighted CL
- tail: estimator for the tail
- est.sigma: Estimator for $sigma_{n-1}$

MackChainLadder example

```
library(ChainLadder)
M <- MackChainLadder(Triangle = RAA, est.sigma = "Mack")
M
     Latest Dev.To.Date Ultimate   IBNR Mack.S.E CV(IBNR)
1981 18,834       1.000   18,834      0         0      NaN
1982 16,704       0.991   16,858    154       206    1.339
1983 23,466       0.974   24,083    617       623    1.010
1984 27,067       0.943   28,703  1,636       747    0.457
1985 26,180       0.905   28,927  2,747     1,469    0.535
1986 15,852       0.813   19,501  3,649     2,002    0.549
1987 12,314       0.694   17,749  5,435     2,209    0.406
1988 13,112       0.546   24,019 10,907     5,358    0.491
1989  5,395       0.336   16,045 10,650     6,333    0.595
1990  2,063       0.112   18,402 16,339    24,566    1.503

                Totals
Latest:     160,987.00
Ultimate:   213,122.23
IBNR:        52,135.23
Mack S.E.:   26,909.01
CV(IBNR):         0.52
```

plot.MackChainLadder




```
plot(M)
```

The residual plots show the standardised residuals against fitted values, origin period, calendar period and development period.

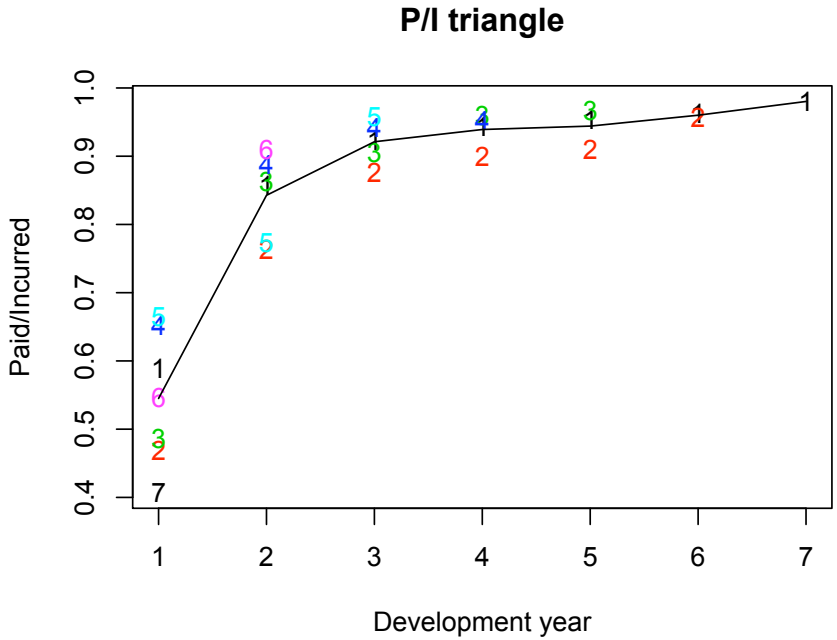
All residual plot should show no pattern or direction for Mack's method to be applicable.

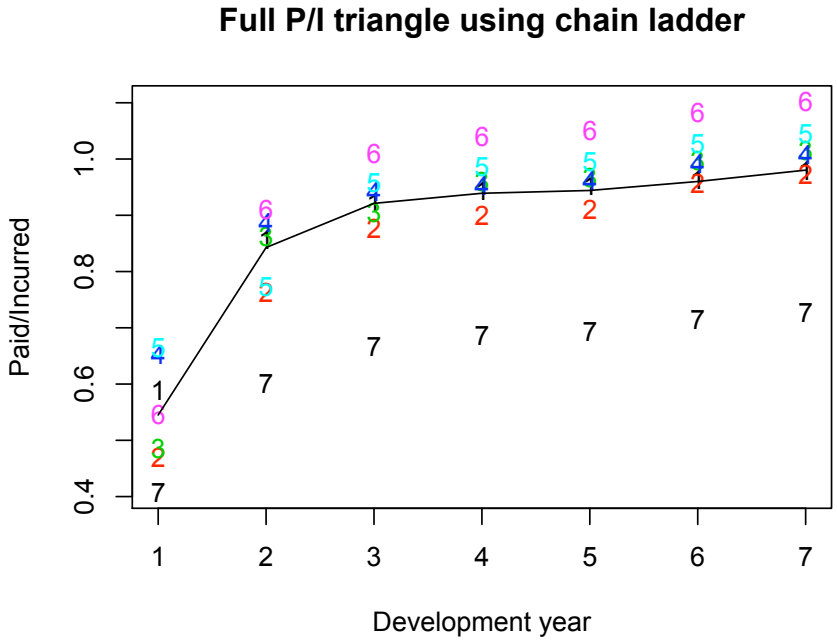
Pattern in any direction can be the result of trends and require further investigations.

Munich-chain-ladder

- Munich-chain-ladder (MCL) is an extension of Mack's method that reduces the gap between IBNR projections based on paid (P) and incurred (I) losses
  - Mack has to be applicable to both triangles
- MCL adjusts the chain-ladder link-ratios depending if the momentary (P/I) ratio is above or below average
- MCL uses the correlation of residuals between P vs. (I/P) and I vs. (P/I) chain-ladder link-ratio to estimate the correction factor

Munich-chain-ladder example





MCLpaid

origin \ dev	1	2	3	4	5	6	7
1	576	1804	1970	2024	2074	2102	2131
2	866	1948	2162	2232	2284	2348	NA
3	1412	3758	4252	4416	4494	NA	NA
4	2286	5292	5724	5850	NA	NA	NA
5	1868	3778	4648	NA	NA	NA	NA
6	1442	4010	NA	NA	NA	NA	NA
7	2044	NA	NA	NA	NA	NA	NA

MCLincurred

origin \ dev	1	2	3	4	5	6	7
1	978	2104	2134	2144	2174	2182	2174
2	1844	2552	2466	2480	2508	2454	NA
3	2904	4354	4698	4600	4644	NA	NA
4	3502	5958	6070	6142	NA	NA	NA
5	2812	4882	4852	NA	NA	NA	NA
6	2642	4406	NA	NA	NA	NA	NA
7	5022	NA	NA	NA	NA	NA	N

MunichChainLadder

Usage:

```
MunichChainLadder(Paid, Incurred,
                  est.sigmaP = "log-linear",
                  est.sigmaI = "log-linear",
                  tailP=FALSE, tailI=FALSE)
```

- Paid: cumulative paid claims triangle
- Incurred: cumulative incurred claims triangle
- est.sigmaP, est.sigmaI: Estimator for $sigma_{n-1}$
- tailP, tailI: estimator for the tail

MunichChainLadder example

```
MCL <- MunichChainLadder(Paid = MCLpaid, Incurred = MCLincurred, est.sigmaP = 0.1,
       est.sigmaI = 0.1)
MCL
```

	Latest Paid	Latest Incurred	Latest P/I Ratio	Ult. Paid	Ult. Incurred	Ult. P/I Ratio
1	2,131	2,174	0.980	2,131	2,174	0.980
2	2,348	2,454	0.957	2,383	2,444	0.975
3	4,494	4,644	0.968	4,597	4,629	0.993
4	5,850	6,142	0.952	6,119	6,176	0.991
5	4,648	4,852	0.958	4,937	4,950	0.997
6	4,010	4,406	0.910	4,656	4,665	0.998
7	2,044	5,022	0.407	7,549	7,650	0.987

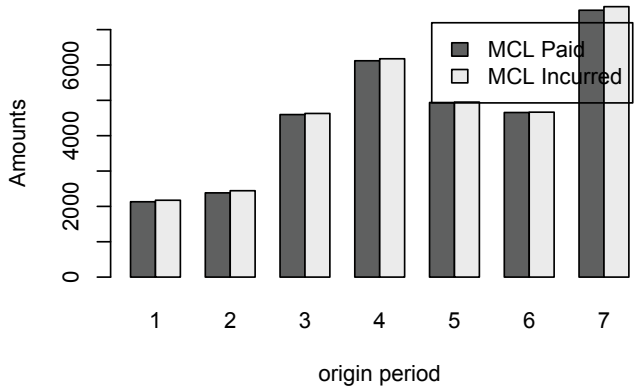
Totals

	Paid	Incurred	P/I Ratio
Latest:	25,525	29,694	0.86
Ultimate:	32,371	32,688	0.99

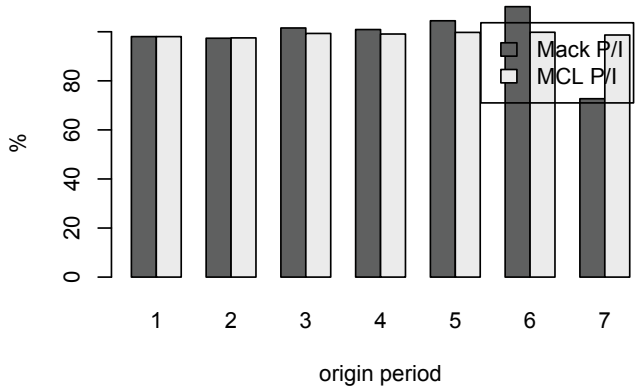
Munich-chain-ladder forecasts based on paid and incurred losses

plot.MunichChainLadder

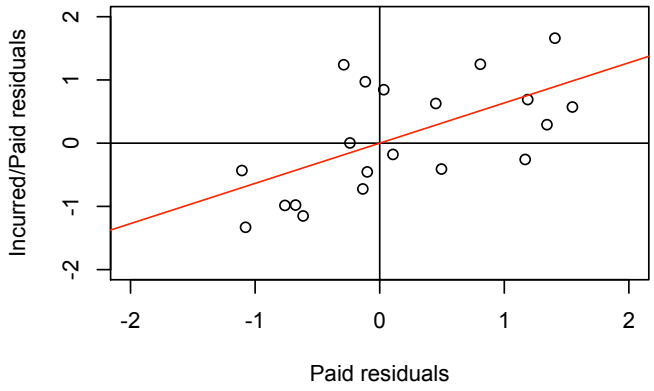
1.



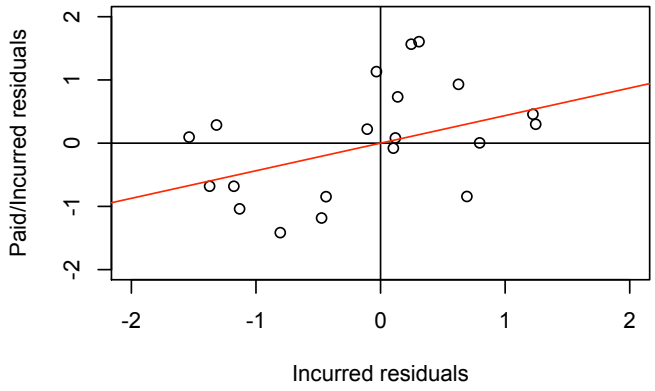
2.



3.



4.




```
plot(MCL)
```

1. MCL forecasts on P and I
2. Comparison of Ultimate P/I ratios of MCL and Mack
3. I/P link-ratio residuals against P link-ratio residuals
4. P/I link-ratio residuals against I link-ratios residuals

Bootstrap-chain-ladder

- *BootChainLadder* uses a two-stage approach.
  1. Calculate the scaled Pearson residuals and bootstrap R times to forecast future incremental claims payments via the standard chain-ladder method.
  2. Simulate the process error with the bootstrap value as the mean and using an assumed process distribution.
- The set of reserves obtained in this way forms the predictive distribution, from which summary statistics such as mean, prediction error or quantiles can be derived.

BootChainLadder

Usage:

```
BootChainLadder(Triangle, R = 999,
                process.distr=c("gamma",
                                "od.pois"))
```

- Triangle: cumulative claims triangle
- R: Number of resampled bootstraps
- process.distr: Assumed process distribution

BootChainLadder example

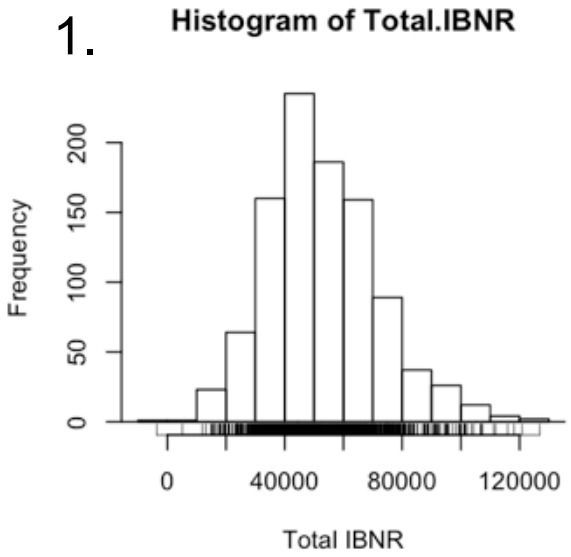
```
set.seed(1)
BootChainLadder(Triangle = RAA, R = 999, process.distr = "od.pois")
```

	Latest	Mean Ultimate	Mean IBNR	SD IBNR	IBNR 75%	IBNR 95%
1981	18,834	18,834	0	0	0	0
1982	16,704	16,921	217	710	253	1,597
1983	23,466	24,108	642	1,340	1,074	3,205
1984	27,067	28,739	1,672	1,949	2,679	4,980
1985	26,180	29,077	2,897	2,467	4,149	7,298
1986	15,852	19,611	3,759	2,447	4,976	8,645
1987	12,314	17,724	5,410	3,157	7,214	11,232
1988	13,112	24,219	11,107	5,072	14,140	20,651
1989	5,395	16,119	10,724	6,052	14,094	21,817
1990	2,063	18,714	16,651	13,426	24,459	42,339

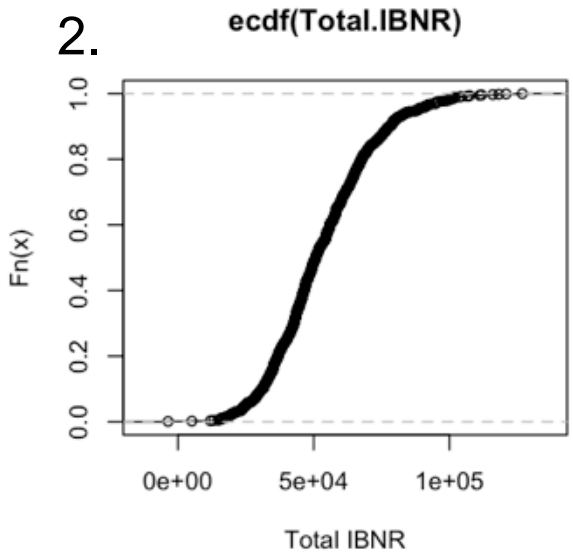
	Totals
Latest:	160,987
Mean Ultimate:	214,066
Mean IBNR:	53,079
SD IBNR:	18,884
Total IBNR 75%:	64,788
Total IBNR 95%:	88,037

plot.BootChainLadder

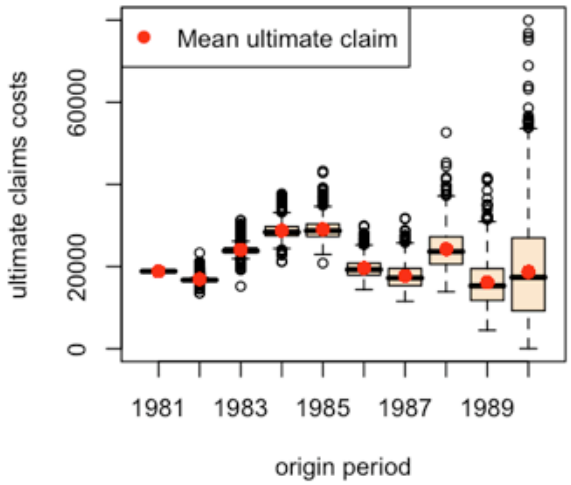
1.



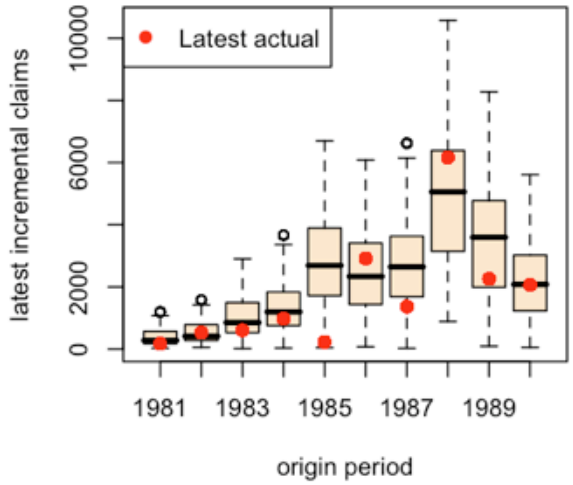
2.



3. Simulated ultimate claims cost



4. Latest actual incremental claims against simulated values



1. Histogram of simulated total IBNR
2. Empirical distribution of total IBNR
3. Box-whisker plot of simulated ultimate claims cost by origin period
4. Test if latest actual incremental loss could come from simulated distribution of claims cost

Generic Methods

- *Mack-, Munich-, BootChainLadder*
  - names: gives the individual elements back
  - summary: summary by origin and totals
  - print: nice formatted output
  - plot: plot overview of the results
- *MackChainLadder*
  - residuals: chain-ladder residuals
- *BootChainLadder*
  - mean: mean IBNR by origin and totals
  - quantile: gives quantiles of the simulation back
  - residuals: chain-ladder residuals

More help

- See example on project web page
- Read documentation on CRAN: http://cran.r-project.org/web/packages/ChainLadder/ChainLadder.pdf
- Read help pages in R:
  - `?MackChainLadder`
  - `?MunichChainLadder`
  - `?BootChainLadder`
- Follow examples in R:
  - `example(MackChainLadder)`
  - `example(MunichChainLadder)`
  - `example(BootChainLadder)`

Conclusions

- R is ideal for reserving
  - Built-in functions for statistical modelling
  - Powerful language for data manipulations
  - Fantastic graphical capabilities for analysis and presentation
  - RExcel add-in allows to share R functions with colleagues without R knowledge
  - rcom allows to control of MS Office from R
  - Easy to set-up connections to databases (ODBC)
  - Effective knowledge transfer - plain text files

References

- Thomas Mack. Distribution-free calculation of the standard error of chain ladder reserve estimates. Astin Bulletin. Vol. 23. No 2. 1993. pp 213-225.
- Thomas Mack. The standard error of chain ladder reserve estimates: Recursive calculation and inclusion of a tail factor. Astin Bulletin. Vol. 29. No 2. 1999. pp 361-366.
- Murphy, Daniel M. Unbiased Loss Development Factors. Proceedings of the Casualty Actuarial Society Casualty Actuarial Society - Arlington, Virginia 1994: LXXXI 154-222.
- Zehnwirth and Barnett. Best estimates for reserves. Proceedings of the CAS, LXXXVI I(167), November 2000.
- P.D.England and R.J.Verrall, Stochastic Claims Reserving in General Insurance, British Actuarial Journal, Vol. 8, pp.443-544, 2002.
- Gerhard Quarg and Thomas Mack. Munich Chain Ladder. Blätter DGVFM 26, Munich, 2004.
- Nigel De Silva. An Introduction to R: Examples for Actuaries. Actuarial Toolkit Working Party, version 0.1 edition, 2006. http://toolkit.pbwiki.com/RToolkit.

R code of the examples in this section

```
library(ChainLadder)
## Example data
RAA

## Triangle developement plot
matplot(t(RAA), t="b")

## Chain ladder ratio
x <- RAA[,1]
y <- RAA[,2]
model <- lm(y~x+0, weights=1/x)

## Full model output
summary(model)

## Simple chain ladder function
ChainLadder <- function(Triangle, weights=1/Triangle){
    n <- ncol(Triangle)
    myModel <- vector("list", (n-1))
    for(i in c(1:(n-1))){
        dev.data <- data.frame(x=Triangle[,i],
               y=Triangle[,i+1])
                   myModel[[i]] <- lm(y~x+0,
          weights=weights[,i],
          data=dev.data)
    }
    return(myModel)
}

CL <- ChainLadder(RAA)

# Get chain-ladder link-ratios
sapply(CL, coef)
# 2.999359 1.623523 1.270888 1.171675 1.113385
# 1.041935 1.033264 1.016936 1.009217

# Get residual standard errors
sapply(lapply(CL, summary), "[[", "sigma")
# 166.983470  33.294538  26.295300   7.824960  10.928818
#   6.389042   1.159062   2.807704        NaN

# Get R squared values
sapply(lapply(ChainLadder(RAA), summary), "[[", "r.squared")
# 0.4681832 0.9532872 0.9704743 0.9976576 0.9959779
# 0.9985933 0.9999554 0.9997809 1.0000000

## MackChainLadder
M <- MackChainLadder(Triangle = RAA, est.sigma = "Mack")
M
plot(M)
```

```
## MunichChainLadder
MCL <- MunichChainLadder(Paid = MCLpaid, Incurred = MCLincurred, est.sigmaP = 0.1,
est.sigmaI = 0.1)
MCL
plot(MCL)

## BootChainLadder
set.seed(1)
B <- BootChainLadder(Triangle = RAA, R = 999, process.distr = "od.pois")
B
plot(B) quantile(B, probs=c(0.75, 0.995))
## IBNR distribution fit
fit <- fitdistr(B$IBNR.Totals[B$IBNR.Totals>0], "lognormal")
fit
curve(plnorm(x,fit$estimate["meanlog"], fit$estimate["sdlog"]), col="red", add=TRUE)

## Fancy 3 plot, requires package "rgl"
library(rgl)
MCL=MackChainLadder(GenIns)
FT <- MCL$FullTriangle
FTpSE <- FT+MCL$Mack.S.E
FTpSE[which(MCL$Mack.S.E==0, arr.ind=TRUE)] <- NA
FTmSE <- FT-MCL$Mack.S.E
FTmSE[which(MCL$Mack.S.E==0, arr.ind=TRUE)] <- NA

zr <- round(FT/FT[1,10]*100)
zlim <- range(zr, na.rm=TRUE)
zlen <- zlim[2] - zlim[1] + 1

colorlut <- terrain.colors(zlen) # height color lookup table
cols <- colorlut[ zr -zlim[1]+1 ] # assign colors to heights for each point

x <- as.numeric(dimnames(FT)$origin)
y <- as.numeric(dimnames(FT)$dev)
persp3d(x, y=y,
        z=(FT), col=cols, xlab="origin", ylab="dev", zlab="loss",back="lines")

mSE <- data.frame(as.table(FTmSE))
points3d(xyz.coords(x=as.numeric(as.character(mSE$origin)),
    y=as.numeric(as.character(mSE$dev)),z=mSE$Freq), size=2)
pSE <- data.frame(as.table(FTpSE))
points3d(xyz.coords(x=as.numeric(as.character(pSE$origin)),
    y=as.numeric(as.character(pSE$dev)),z=pSE$Freq), size=2)
```

CAS Annual Meeting 2008

The *copula* package in R

Daniel Murphy
Trinostics

November 18, 2008

copula package – convenient constructors for correlated calculations

- Gentlemen, start your copulas!

```
> library(copula)
```

- Construct a gaussian copula, two lines, average correlation of 50%

```
> my_normalcop=normalCopula(param=.50, dim=2)
```

- Construct the joint distribution of two lognormally distributed lines (X,Y) whose copula is assumed to be my_normalcop just constructed

```
> my_joint=mvdc(my_normalcop, margins=c("lnorm", "lnorm"),
            paramMargins=list(list(meanlog=12.5, sdlog=.5),
                              list(meanlog=12.9, sdlog=.5)))
```

- Randomly simulate N=10 million (X,Y) pairs from my_joint

```
> pairs=rmvdc(my_joint,N) # That's all there is to it!
```

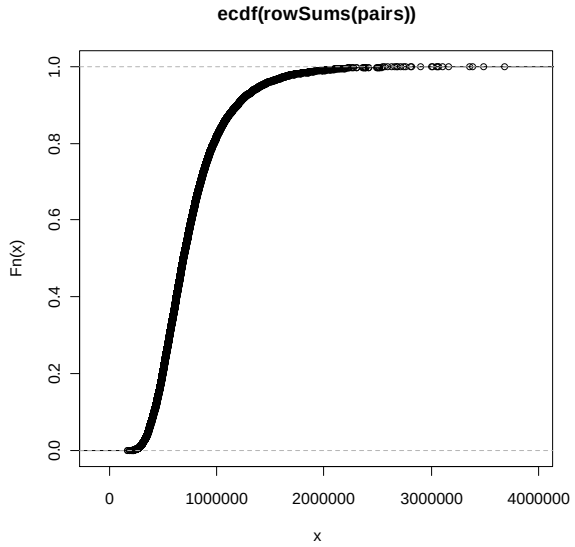
- How about a cop-ple of graphs?
  - Portfolio's empirical cumulative distribution function

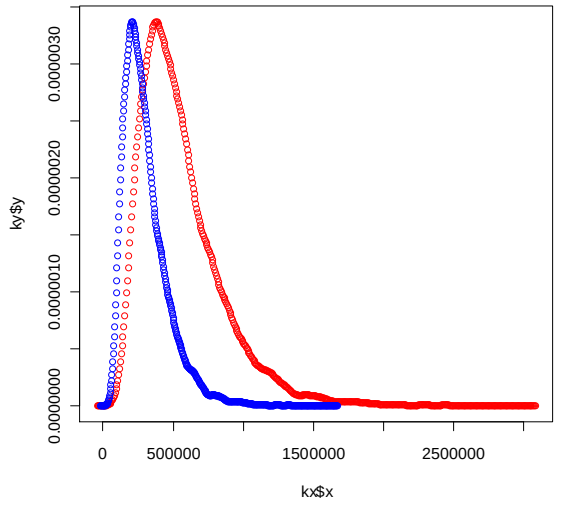
```
> plot(ecdf(rowSums(pairs)) #rowSums for portfolio=X+Y
```

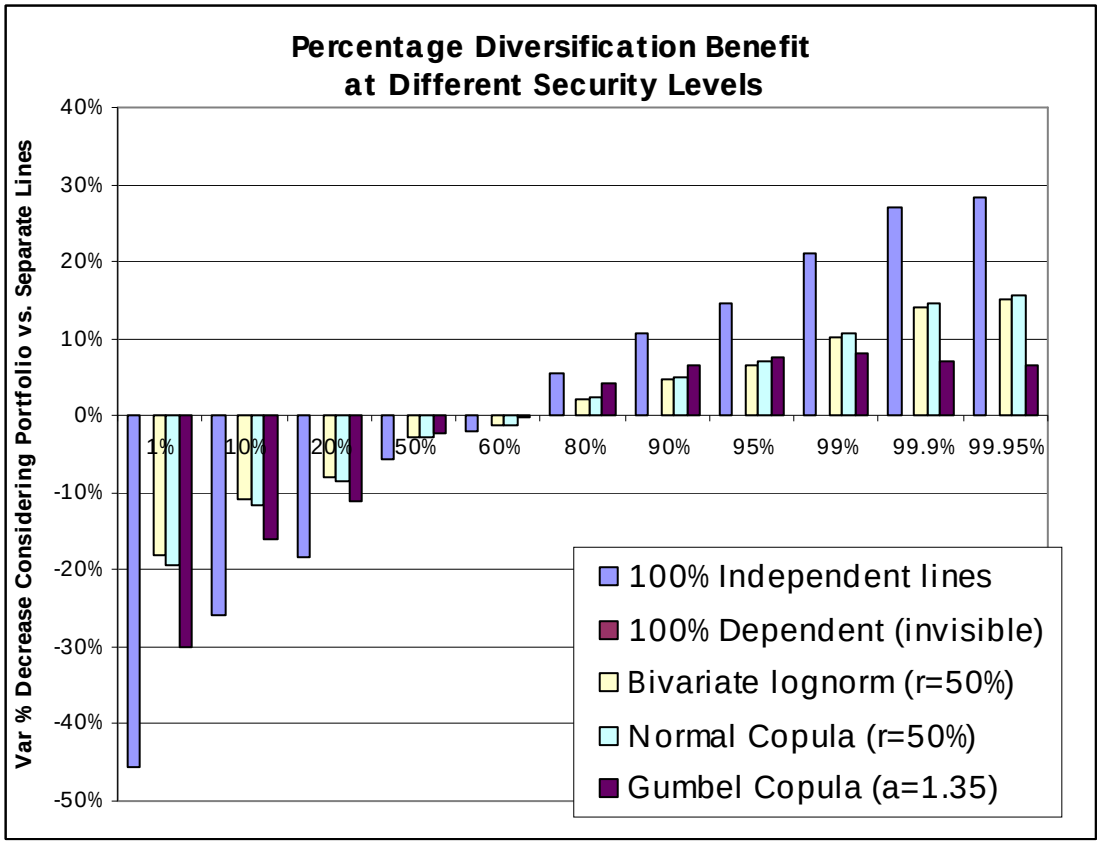
  - Individual lines' density functions

```
> X=rlnorm(N,12.5,.5); Y=rlnorm(N,12.9,.5);
  kx=density(X); ky=density(Y);
  plot(kx$x,ky$y,col="red"); points(ky$x,ky$y,col="blue")
```

cdf's and pdf's are interesting to look at; more concerned with diversification benefit, tail correlation







- 10 million simulations for each of the 5 options took R ~ 5 minutes

Q&A