

Actuariat de l'Assurance Non-Vie # 11

A. Charpentier (Université de Rennes 1)



ENSAE ParisTech, Octobre / Décembre 2016.

<http://freakonometrics.hypotheses.org>

Regression Models in Claims Reserving

A natural idea is to assume that incremental payments $Y_{i,j}$ can be explained by two factors: one related to occurrence year i , and one development factor, related to j . Formally, we assume that

$$Y_{i,j} \sim \mathcal{L}(\theta_{i,j}), \text{ where } \theta_{i,j} = \alpha_i \cdot \beta_j$$

i.e. $Y_{i,j}$ is a random variable, with distribution \mathcal{L} , where parameter(s) can be related to the two factors.

Poisson regression in claims reserving

Renshaw & Verrall (1998) proposed to use a Poisson regression for incremental payments to estimate claim reserve, i.e.

$$Y_{i,j} \sim \mathcal{P}(\exp[\gamma + \alpha_i + \beta_j]).$$

```
1 devF=as.factor(development); anF=as.factor(year)
2 REG=glm(vec.C~devF+anF, family = "Poisson")
```

Here,

```
1 > summary(REG)
2 Call:
3 glm(formula = vec.C ~ anF + devF, family = poisson(link = "log"),
4      data = triangle)
5
6 Deviance Residuals:
7
8      Min           1Q       Median           3Q       Max
-2.343e+00 -4.996e-01  9.978e-07  2.770e-01  3.936e+00
```

```

9
10 Coefficients:
11             Estimate Std. Error z value Pr(>|z|)
12 (Intercept) 8.05697  0.01551 519.426 < 2e-16 ***
13 anF1989     0.06440  0.02090  3.081  0.00206 **
14 anF1990     0.20242  0.02025  9.995 < 2e-16 ***
15 anF1991     0.31175  0.01980 15.744 < 2e-16 ***
16 anF1992     0.44407  0.01933 22.971 < 2e-16 ***
17 anF1993     0.50271  0.02079 24.179 < 2e-16 ***
18 devF1      -0.96513  0.01359 -70.994 < 2e-16 ***
19 devF2      -4.14853  0.06613 -62.729 < 2e-16 ***
20 devF3      -5.10499  0.12632 -40.413 < 2e-16 ***
21 devF4      -5.94962  0.24279 -24.505 < 2e-16 ***
22 devF5      -5.01244  0.21877 -22.912 < 2e-16 ***
23 ---
24
25 (Dispersion parameter for poisson family taken to be 1)
26

```

```

27 Null deviance: 46695.269 on 20 degrees of freedom
28 Residual deviance: 30.214 on 10 degrees of freedom
29 AIC: 209.52
30
31 Number of Fisher Scoring iterations: 4

```

Again, it is possible to summarize this information in triangles....

Predictions can be used to complete the triangle.

```

1 ANew=rep(1 :Ntr),times=Ntr) ; DNew=rep(0 :(Ntr-1),each=Ntr)
2 P=predict(REG, newdata=data.frame(A=as.factor(ANew),D=as.factor(
3 DNew)))
4 payinc.pred= exp(matrix(as.numeric(P),nrow=n,ncol=n))
5 noise = payinc-payinc.pred

6 year development paycum payinc payinc.pred noise
7 1 1988 0 3209 3209 3155.699242 5.330076e+01
8 2 1989 0 3367 3367 3365.604828 1.395172e+00
9 3 1990 0 3871 3871 3863.737217 7.262783e+00

```

5	4	1991	0	4239	4239	4310.096418	-7.109642e+01
6	5	1992	0	4929	4929	4919.862296	9.137704e+00
7	6	1993	0	5217	5217	5217.000000	1.818989e-12
8	7	1988	1	4372	1163	1202.109851	-3.910985e+01
9	8	1989	1	4659	1292	1282.069808	9.930192e+00
10	9	1990	1	5345	1474	1471.824853	2.175147e+00
11	10	1991	1	5917	1678	1641.857784	3.614222e+01
12	11	1992	1	6794	1865	1874.137704	-9.137704e+00
13	12	1988	2	4411	39	49.820712	-1.082071e+01
14	13	1989	2	4696	37	53.134604	-1.613460e+01
15	14	1990	2	5398	53	60.998886	-7.998886e+00
16	15	1991	2	6020	103	68.045798	3.495420e+01
17	16	1988	3	4428	17	19.143790	-2.143790e+00
18	17	1989	3	4720	24	20.417165	3.582835e+00
19	18	1990	3	5420	22	23.439044	-1.439044e+00
20	19	1988	4	4435	7	8.226405	-1.226405e+00
21	20	1989	4	4730	10	8.773595	1.226405e+00
22	21	1988	5	4456	21	21.000000	-2.842171e-14

The `pearson` residuals are

$$\varepsilon_{i,j}^P = \frac{X_{i,j} - \hat{\mu}_{i,j}}{\sqrt{\hat{\mu}_{i,j}}},$$

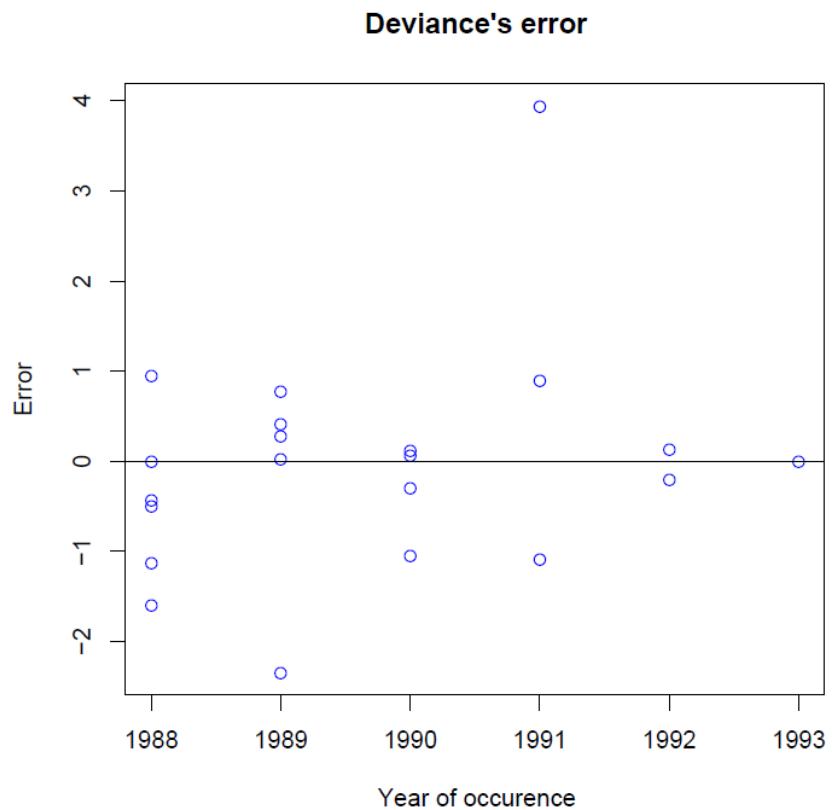
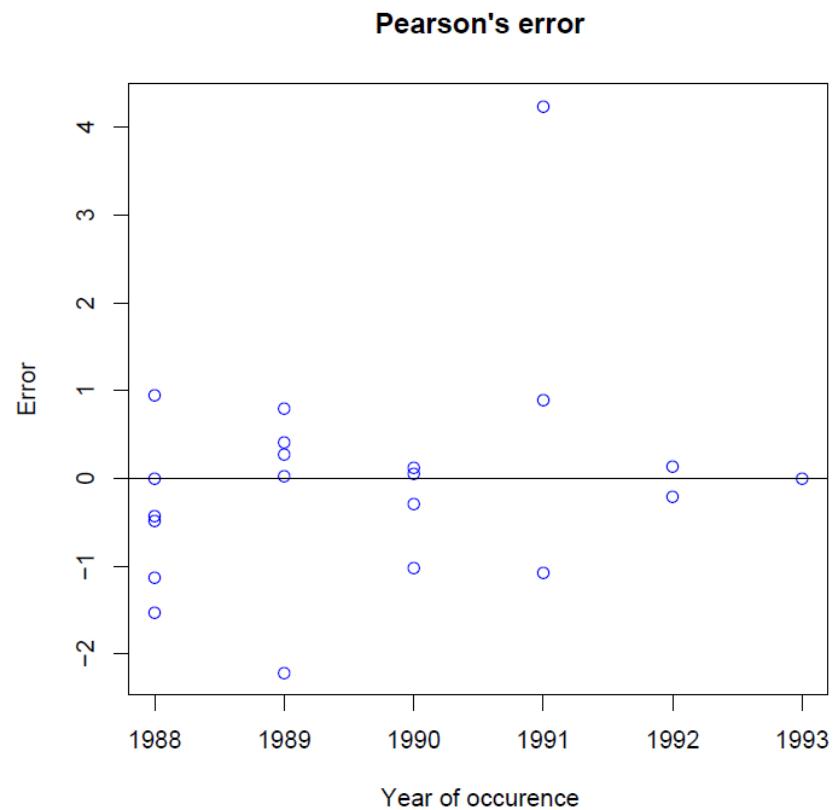
The `deviance` residuals are

$$\varepsilon_{i,j}^D = \frac{X_{i,j} - \hat{\mu}_{i,j}}{\sqrt{d_{i,j}}},$$

Pearson's error can be obtained from function `resid=residuals(REG, "pearson")`, and summarized in a triangle

```
1 > PEARSON
2
3      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
4 [1,] 9.4882e-01 -1.128012 -1.5330 -0.48996 -0.42759 -6.2021e-15
5 [2,] 2.4048e-02  0.277333 -2.2134  0.79291  0.41404          NA
6 [3,] 1.1684e-01  0.056697 -1.0241 -0.29723          NA          NA
7 [4,] -1.0829e+00  0.891963  4.2373          NA          NA          NA
8 [5,] 1.3027e-01 -0.211074          NA          NA          NA          NA
9 [6,] 2.5183e-14          NA          NA          NA          NA          NA
```

Errors in GLMs



log-Poisson regression and Chain-Ladder

The log-Poisson is interesting since it (usually) provides the same amount of reserves as Chain Ladder.

```

1 > library(ChainLadder)
2 > an <- 10; ligne = rep(1:an, each=an); colonne = rep(1:an, an)
3 > passe = (ligne + colonne - 1)<=an; n = sum(passe)
4 > PAID=GenIns; INC=PAID
5 > INC[,2:an]=PAID[,2:an]-PAID[,1:(an-1)]
6 > Y = as.vector(INC)
7 > lig = as.factor(ligne)
8 > col = as.factor(colonne)
9 > base = data.frame(Y,col,lig)
10 > reg=glm(Y~col+lig,data=base,family="poisson")
11 > sum(exp(predict(reg,newdata=base))[passe!=TRUE])
12 [1] 18680856

```

log-Poisson regression and Chain-Ladder

```

1 > MackChainLadder(GenIns)
2 MackChainLadder(Triangle = GenIns)
3   Latest Dev.To.Date Ultimate      IBNR Mack.S.E CV(IBNR)
4 1 3,901,463     1.0000 3,901,463       0        0    NaN
5 2 5,339,085     0.9826 5,433,719 94,634    71,835 0.759
6 3 4,909,315     0.9127 5,378,826 469,511   119,474 0.254
7 4 4,588,268     0.8661 5,297,906 709,638   131,573 0.185
8 5 3,873,311     0.7973 4,858,200 984,889   260,530 0.265
9 6 3,691,712     0.7223 5,111,171 1,419,459  410,407 0.289
10 7 3,483,130     0.6153 5,660,771 2,177,641  557,796 0.256
11 8 2,864,498     0.4222 6,784,799 3,920,301  874,882 0.223
12 9 1,363,294     0.2416 5,642,266 4,278,972  970,960 0.227
13 10 344,014      0.0692 4,969,825 4,625,811 1,362,981 0.295
14
15   Totals
16   Latest: 34,358,090.00
17   Ultimate: 53,038,945.61
18   IBNR: 18,680,855.61

```

18 Mack S.E.: 2 ,441 ,364.13

19 CV(IBNR): 0.13

An explicit expression to quantify uncertainty

Recall that we while to estimate

$$\begin{aligned}\mathbb{E}([R - \hat{R}]^2) &= [\mathbb{E}(R) - \mathbb{E}(\hat{R})]^2 + \text{Var}(R - \hat{R}) \\ &\approx \text{Var}(R) + \text{Var}(\hat{R})\end{aligned}$$

Classically, consider a **log-Poisson model**, were incremental payments satisfy

$$Y_{i,j} \sim \mathcal{P}(\mu_{i,j}) \text{ where } \mu_{i,j} = \exp[\eta_{i,j}] = \exp[\gamma + \alpha_i + \beta_j]$$

Using the delta method, we get that *asymptotically*

$$\text{Var}(\hat{Y}_{i,j}) = \text{Var}(\hat{\mu}_{i,j}) \approx \left| \frac{\partial \mu_{i,j}}{\partial \eta_{i,j}} \right|^2 \text{Var}(\hat{\eta}_{i,j})$$

where, since we consider a log link,

$$\frac{\partial \mu_{i,j}}{\partial \eta_{i,j}} = \mu_{i,j}$$

i.e., with an ODP distribution (i.e. $\text{Var}(Y_{i,j}) = \varphi \mathbb{E}(Y_{i,j})$),

$$\mathbb{E} \left([Y_{i,j} - \widehat{Y}_{i,j}]^2 \right) \approx \widehat{\varphi} \cdot \widehat{\mu}_{i,j} + \widehat{\mu}_{i,j}^2 \cdot \widehat{\text{Var}}(\eta_{i,j})$$

and

$$\text{Cov}(Y_{i,j}, Y_{k,l}) \approx \widehat{\mu}_{i,j} \cdot \widehat{\mu}_{k,l} \cdot \widehat{\text{Cov}}(\widehat{\eta}_{i,j}, \widehat{\eta}_{k,l})$$

Thus, since the overall amount of reserves satisfies

$$\mathbb{E} \left([R - \widehat{R}]^2 \right) \approx \sum_{i+j-1 > n} \widehat{\varphi} \cdot \widehat{\mu}_{i,j} + \widehat{\mu}' \widehat{\text{Var}}(\widehat{\eta}) \widehat{\mu}.$$

```

1 > an <- 6; ligne = rep(1:an, each=an); colonne = rep(1:an, an)
2 > passe = (ligne + colonne - 1) <= an; np = sum(passe)
3 > futur = (ligne + colonne - 1) > an; nf = sum(passe)
4 > INC=PAID
5 > INC[,2:6]=PAID[,2:6]-PAID[,1:5]
6 > Y = as.vector(INC)

```

```

7 > lig = as.factor(ligne); col = as.factor(colonne)
8 >
9 > CL <- glm(Y~lig+col, family=quasipoisson)
10 > Y2=Y; Y2[is.na(Y)]=.001
11 > CL2 <- glm(Y2~lig+col, family=quasipoisson)
12 > YP = predict(CL)
13 > p = 2*6-1;
14 > phi.P = sum(residuals(CL,"pearson")^2)/(np-p)
15 > Sig = vcov(CL)
16 > X = model.matrix(CL2)
17 > Cov.eta = X%*%Sig%*%t(X)
18 > mu.hat = exp(predict(CL,newdata=data.frame(lig,col)))*futur
19 > pe2 = phi.P * sum(mu.hat) + t(mu.hat) %*% Cov.eta %*% mu.hat
20 > cat("Total reserve =", sum(mu.hat), "prediction error =", sqrt(pe2),
21   ), "\n")
21 Total reserve = 2426.985 prediction error = 131.7726

```

i.e. $\hat{\mathbb{E}}([R - \bar{R}]^2) = 131.77$.

Uncertainty and bootstrap simulations

Based on that *theoretical* triangle, it is possible to generate residuals to obtain a simulated triangle. Since the size of the sample is small (here 21 observed values), assuming normality for Pearson's residuals can be too restrictive. Resampling bootstrap procedure can then be more robust.

In order to get the loss distribution, it is possible to use bootstrap techniques to generate a matrix of errors, see Renshaw & Verrall (1994). They suggest to bootstrap Pearson's residuals, and the simulation procedure is the following

- estimate the model parameter (GLM), $\hat{\beta}$,
- calculate fitted values $\hat{\mu}_{i,j}$, and the residuals $r_{i,j} = \frac{Y_{i,j} - \hat{\mu}_{i,j}}{\sqrt{V(\hat{\mu}_{i,j})}}$,
- forecast with original data $\hat{\mu}_{i,j}$ for $i + j > n$.

Then can start the bootstrap loops, repeating B times

- resample the residuals with `resample`, and get a new sample $r_{i,j}^{(b)}$,
- create a pseudo sample solving $Y_{i,j}^* = \hat{\mu}_{i,j} + r_{i,j}^{(b)} \times \sqrt{V(\hat{\mu}_{i,j})}$,
- estimate the model using GLM procedure and derive bootstrap forecast

Let `resid.sim` be resampled residuals. Note that `REG$fitted.values` (called here `payinc.pred`) is the vector containing the $\hat{\mu}_{i,j}$'s. And further $V(\hat{\mu}_{i,j})$ is here simply `REG$fitted.values` since the variance function for the Poisson regression is the *identity* function. Hence, here

$$Y_{i,j}^* = \hat{\mu}_{i,j} + r_{i,j}^{(b)} \times \sqrt{\hat{\mu}_{i,j}}$$

and thus, set

```

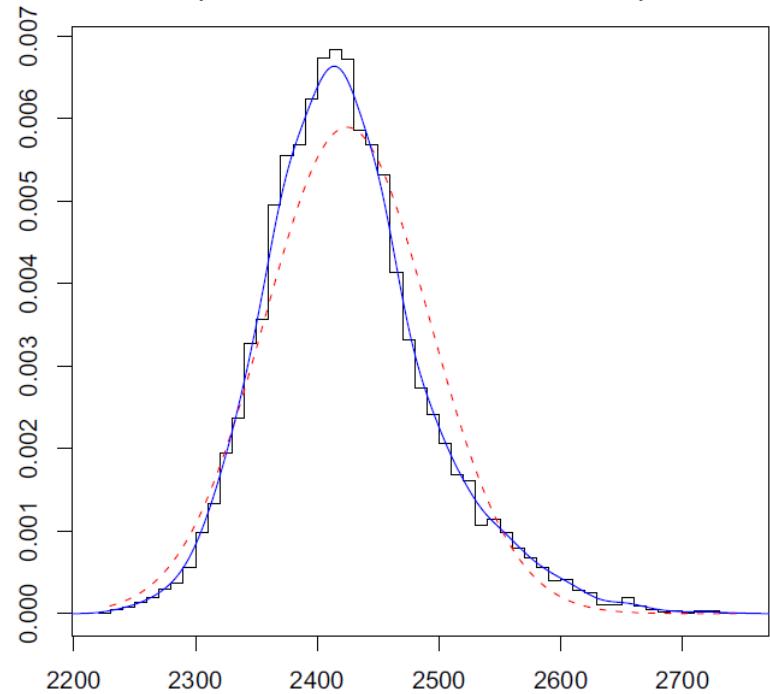
1 resid.sim = sample(resid,Ntr*(Ntr+1)/2,replace=TRUE)
2 payinc.sim = resid.sim*sqrt(payinc.pred)+payinc.pred
3
4      [,1]      [,2]      [,3]      [,4]      [,5]      [,6]

```

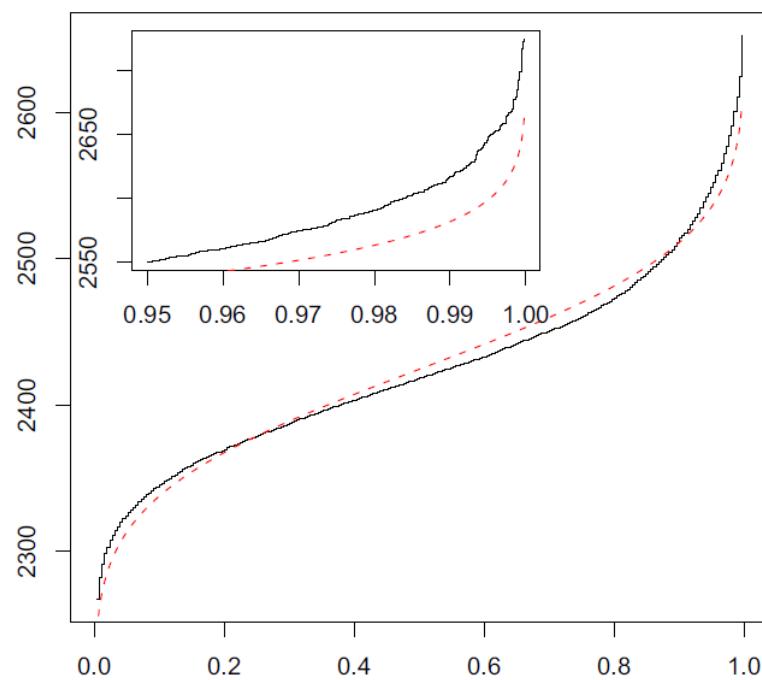
```
5 [1,] 3155.699 1216.465 42.17691 18.22026 9.021844 22.89738
6 [2,] 3381.694 1245.399 84.02244 18.20322 11.122243 NA
7 [3,] 3726.151 1432.534 61.44170 23.43904 NA NA
8 [4,] 4337.279 1642.832 74.58658 NA NA NA
9 [5,] 4929.000 1879.777 NA NA NA NA
10 [6,] 5186.116 NA NA NA NA NA
```

For this simulated triangle, we can use Chain-Ladder estimate to derive a simulated reserve amount (here [2448.175](#)). Figure below shows the empirical distribution of those amounts based on 10,000 random simulations.

Estimated density of total reserves
(with Gaussian fitted distribution)

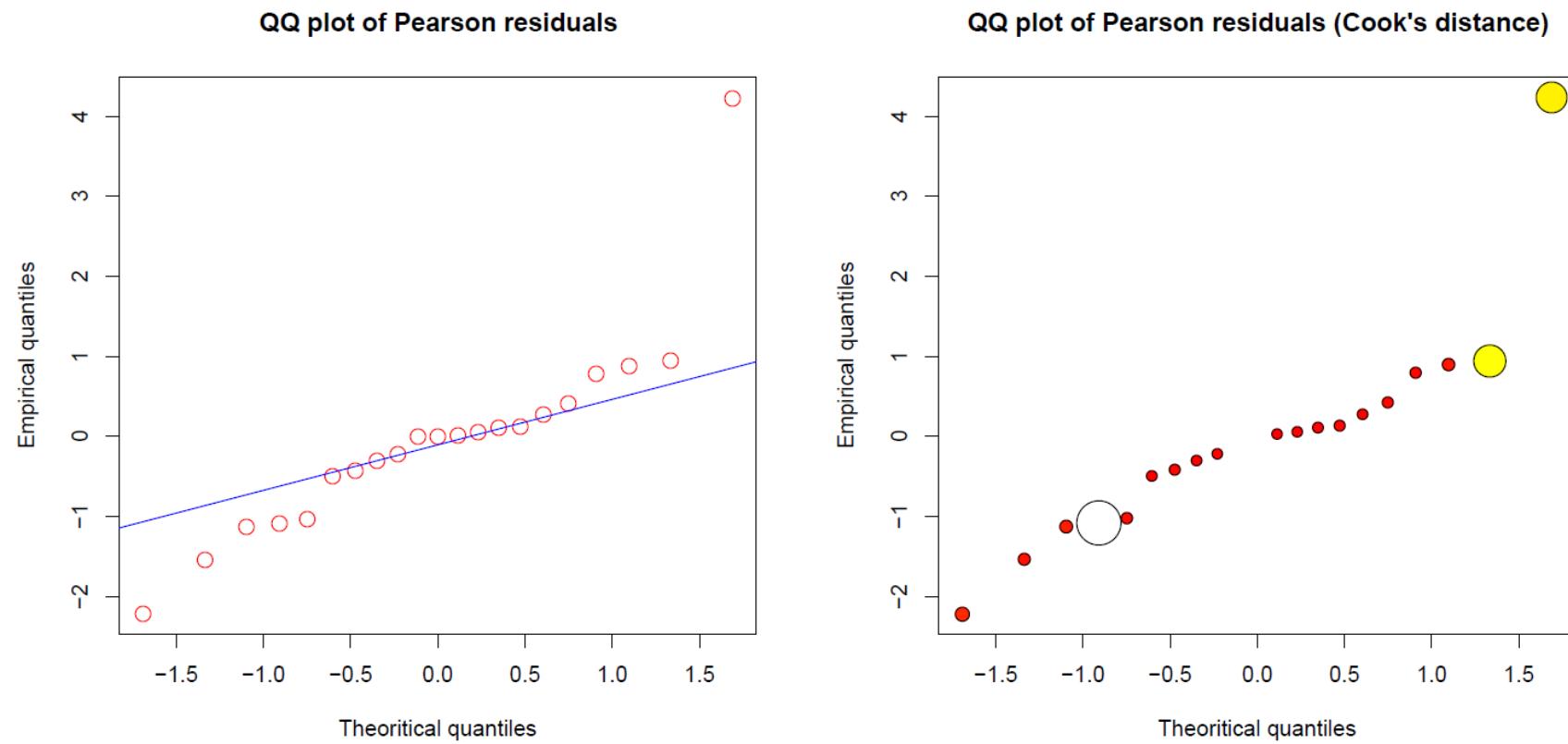


Estimated quantile of total reserves
(with Gaussian fitted distribution)



Parametric or nonparametric Monte Carlo ?

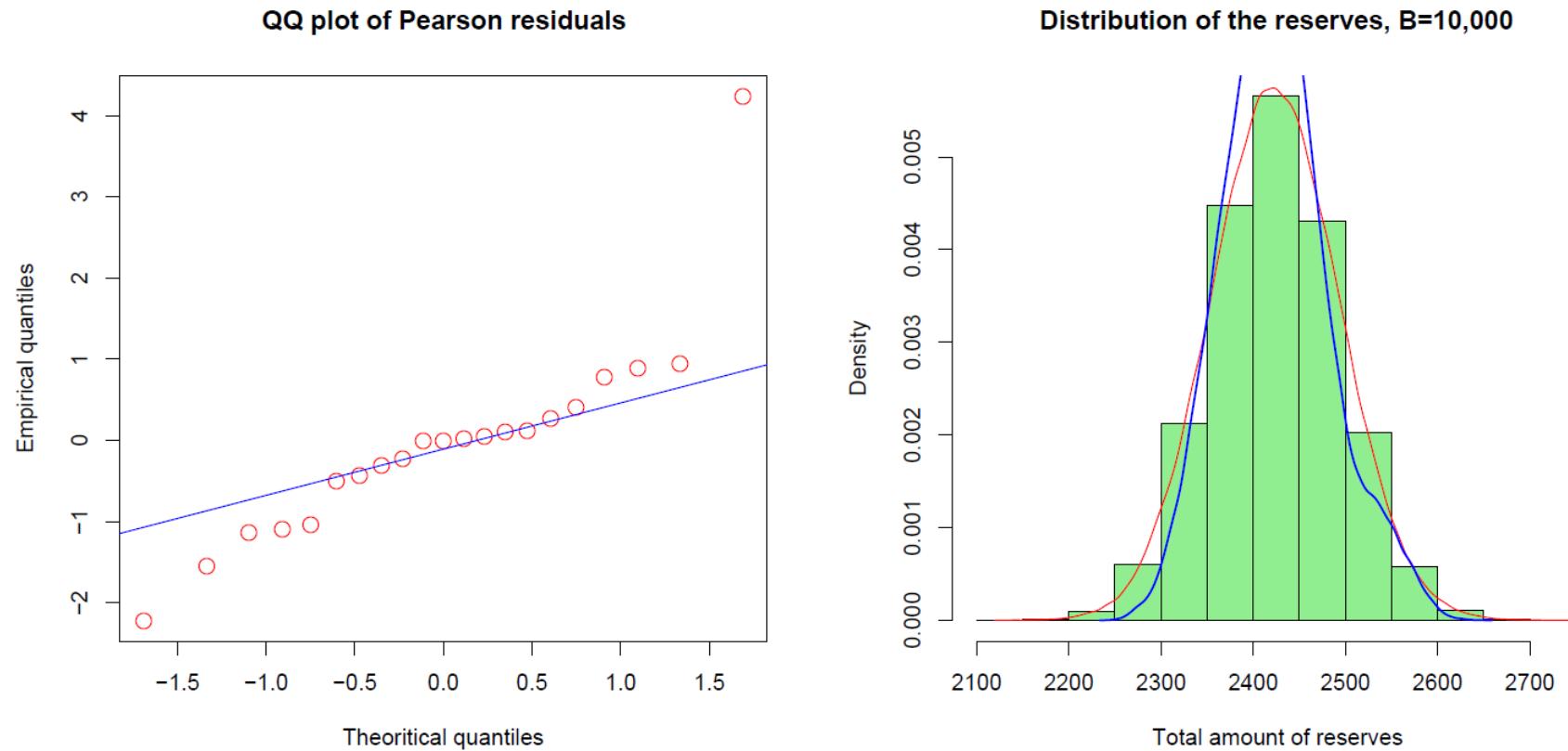
A natural idea would be to assume that Pearson residual have a Gaussian distribution, `qqnorm(R)`; `qqline(R)`



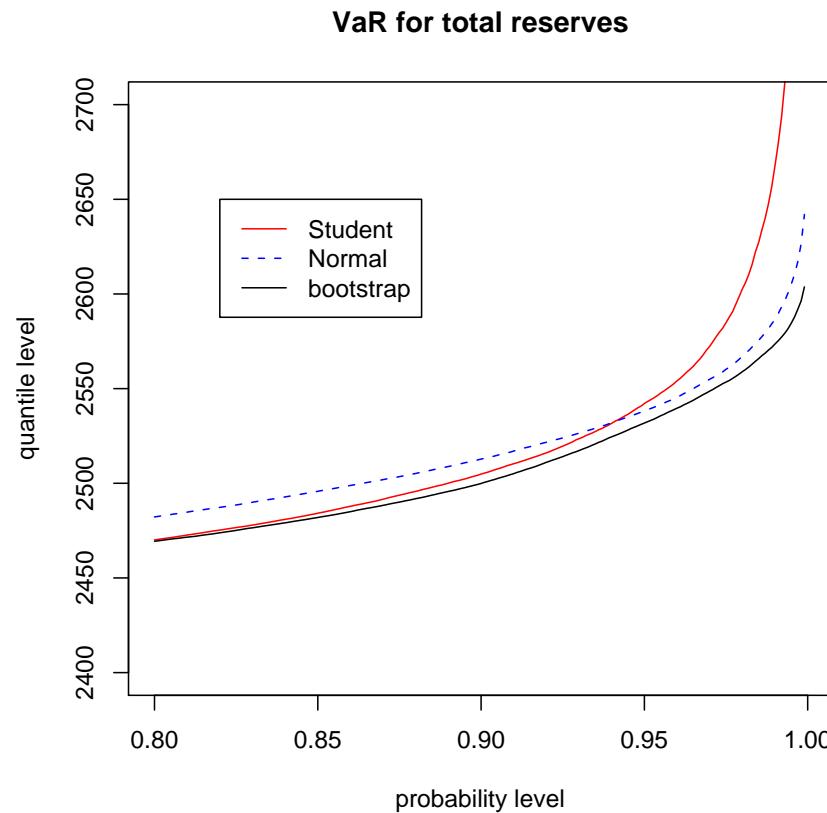
The graph on the right draw point with a size proportional to its Cook's distance.

Instead of resampling in the sample obtained, we can also directly draw from a normal distribution, i.e.

```
1 > rnorm(length(R), mean=mean(R), sd=sd(R))
```



The second triangle is obtained using a Student t distribution (the blue line being the bootstrap estimate).



Note that the bootstrap technique is valid only in the case where the residuals are perfectly independent.

In R, it is also possible to use the `BootChainLadder(Triangle , R = 999, process.distr = "od.pois")` function.

Going further

So far, we have derived a ditrisbution for the best estimate of total reserves.

Note tat it is possible to estimate a scale parameter ϕ . [England & Verrall \(1999\)](#) suggested

$$\hat{\phi} = \frac{\sum \varepsilon_{i,j}^2}{n - p}$$

where the summation is over all past observations.

Bootstrap Chain-Ladder

```

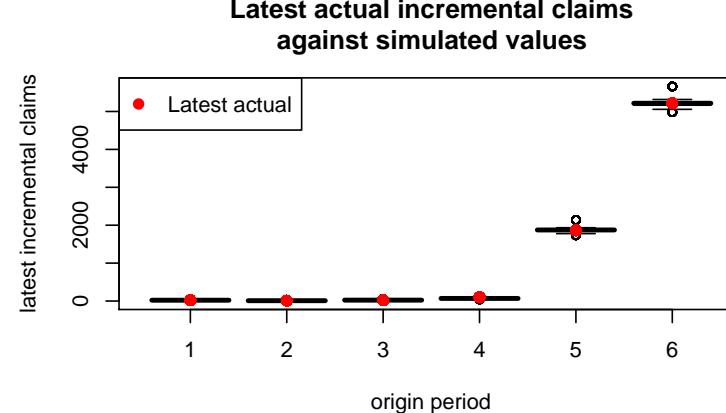
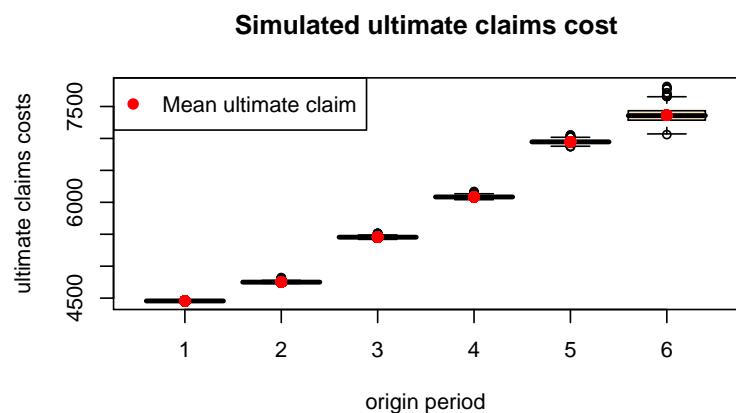
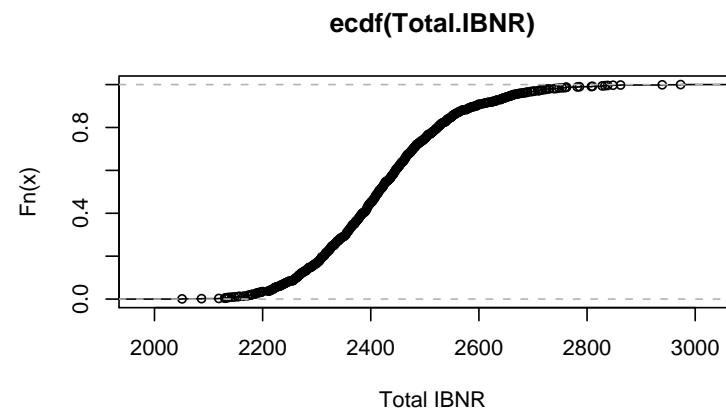
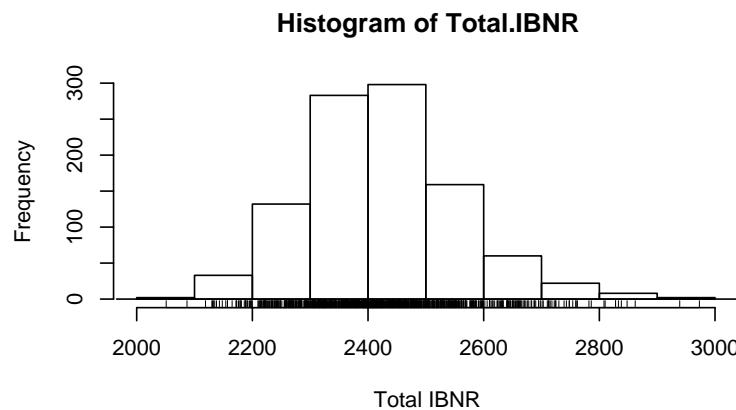
1 > I=as.matrix(read.table("D:\\triangleC.csv",sep=";",header=FALSE))
2 > BCL <- BootChainLadder(Triangle = I, R = 999, process.distr = "od.
   pois")
3 > BCL
4 BootChainLadder(Triangle = I, R = 999, process.distr = "od.pois")
5
6   Latest Mean Ultimate Mean IBNR SD IBNR IBNR 75% IBNR 95%
7 1 4,456      4,456     0.0    0.0    0    0
8 2 4,730      4,752    22.0   11.8   28   45
9 3 5,420      5,455    35.3   14.6   44   61
10 4 6,020      6,086    66.2   20.8   78  102
11 5 6,794      6,947   152.7  29.1  170  205
12 6 5,217      7,364   2,146.9 112.5  2,214 2,327
13
14           Totals
15 Latest:      32,637
16 Mean Ultimate: 35,060

```

```

17 Mean IBNR : 2 ,423
18 SD IBNR : 131
19 Total IBNR 75% : 2 ,501
20 Total IBNR 95% : 2 ,653

```



From Poisson to Over-Dispersed Poisson

Classical, in GLMs we consider distributions with density

$$f(z|\theta, \phi) = \exp\left(\frac{z\theta - b(\theta)}{\phi} + c(z, \phi)\right),$$

e.g. for the Poisson distribution $\mathcal{P}(\lambda)$ then

$$f(z|\lambda) = \exp(-\lambda) \frac{\lambda^z}{z!} = \exp\left(z \log \lambda - \lambda - \log z!\right), \quad z \in \mathbb{N},$$

with $\theta = \log \lambda$, $\phi = 1$, $b(\theta) = \exp \theta = \lambda$ and $c(z, \phi) = -\log z!$.

Assume that $\phi \neq 1$ becomes an additional parameter (that should be estimated).

Note that in that case $f(z|\lambda)$ is not any more a density, but it is a **quasi**density.

Further, note that

$$\text{Var}(Z) = \phi \mathbb{E}(Z).$$

Thus, if $\phi > 1$ there is **overdispersion**.

On quasiPoisson regression

In order to understand the rule of the additional parameter, recall that for the Gaussien linear model, $\mathcal{N}(\mu, \sigma^2)$ it is an exponential distribution with $\theta = \mu$, $b(\theta) = \theta^2/2$, $\phi = \sigma^2$ and

$$c(z, \phi) = -\frac{1}{2} \left(\frac{y^2}{\sigma^2} + \log(2\pi\sigma^2) \right).$$

Thus, ϕ is the variance parameter

$$Y|\mathbf{X} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2)$$

In that linear model, estimation is done based on the following process,

- estimate $\boldsymbol{\beta}$ as $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y}$
- derive the implied residuals, $\hat{\boldsymbol{\epsilon}} = \mathbf{Y} - \mathbf{X}\hat{\boldsymbol{\beta}}$
- estimate σ as the variance of the implied residuals

Thus, ϕ does not impact the estimation of the coefficient, but it will impact their significativity.

```

1 > an <- 10; ligne = rep(1:an, each=an); colonne = rep(1:an, an)
2 > passe = (ligne + colonne - 1)<=an; n = sum(passe)
3 > PAID=GenIns; INC=PAID
4 > INC[,2:an]=PAID[,2:an]-PAID[,1:(an-1)]
5 > Y = as.vector(INC)
6 > lig = as.factor(ligne)
7 > col = as.factor(colonne)
8 > base = data.frame(Y,col,lig)
9 > reg1=glm(Y~col+lig,data=base,family="poisson")
10 > reg2=glm(Y~col+lig,data=base,family="quasipoisson")
11 > summary(reg1)
12 Call:
13 glm(formula = Y ~ col + lig, family = "poisson", data = base)
14 Coefficients:
15
16 (Intercept) 12.5064047  0.0007540 16587.372 < 2e-16 ***

```

17	col12	0.3312722	0.0006694	494.848	< 2e-16	***
18	col13	0.3211186	0.0006877	466.963	< 2e-16	***
19	col14	0.3059600	0.0007008	436.570	< 2e-16	***
20	col15	0.2193163	0.0007324	299.461	< 2e-16	***
21	col16	0.2700770	0.0007445	362.755	< 2e-16	***
22	col17	0.3722084	0.0007606	489.344	< 2e-16	***
23	col18	0.5533331	0.0008133	680.377	< 2e-16	***
24	col19	0.3689342	0.0010429	353.772	< 2e-16	***
25	col10	0.2420330	0.0018642	129.830	< 2e-16	***
26	lig2	0.9125263	0.0006490	1406.042	< 2e-16	***
27	lig3	0.9588306	0.0006652	1441.374	< 2e-16	***
28	lig4	1.0259970	0.0006840	1499.927	< 2e-16	***
29	lig5	0.4352762	0.0008019	542.814	< 2e-16	***
30	lig6	0.0800565	0.0009364	85.492	< 2e-16	***
31	lig7	-0.0063815	0.0010390	-6.142	8.14e-10	***
32	lig8	-0.3944522	0.0013529	-291.560	< 2e-16	***
33	lig9	0.0093782	0.0013963	6.716	1.86e-11	***
34	lig10	-1.3799067	0.0039097	-352.946	< 2e-16	***

```

35 ---
36 (Dispersion parameter for poisson family taken to be 1)
37 Null deviance: 10699464 on 54 degrees of freedom
38 Residual deviance: 1903014 on 36 degrees of freedom
39 (45 observations deleted due to missingness)
40 AIC: 1903877
41 Number of Fisher Scoring iterations: 4

1 > summary(reg2)
2 Call:
3 glm(formula = Y ~ col + lig, family = "quasipoisson", data = base)
4 Coefficients:
5             Estimate Std. Error t value Pr(>|t|)
6 (Intercept) 12.506405   0.172924 72.323 < 2e-16 ***
7 col2        0.331272   0.153537  2.158  0.03771 *
8 col3        0.321119   0.157719  2.036  0.04916 *
9 col4        0.305960   0.160736  1.903  0.06499 .
10 col5       0.219316   0.167970  1.306  0.19994
11 col6       0.270077   0.170756  1.582  0.12247

```

```

12 col7      0.372208   0.174451   2.134   0.03976 *
13 col8      0.553333   0.186525   2.967   0.00532 **
14 col9      0.368934   0.239181   1.542   0.13170
15 col10     0.242033   0.427562   0.566   0.57485
16 lig2      0.912526   0.148850   6.131   4.65e-07 ***
17 lig3      0.958831   0.152569   6.285   2.90e-07 ***
18 lig4      1.025997   0.156883   6.540   1.33e-07 ***
19 lig5      0.435276   0.183914   2.367   0.02344 *
20 lig6      0.080057   0.214770   0.373   0.71152
21 lig7      -0.006381  0.238290  -0.027   0.97878
22 lig8      -0.394452  0.310289  -1.271   0.21180
23 lig9      0.009378   0.320249   0.029   0.97680
24 lig10     -1.379907  0.896690  -1.539   0.13258
25 ---
26 (Dispersion parameter for quasipoisson family taken to be 52601.93)
27 Null deviance: 10699464 on 54 degrees of freedom
28 Residual deviance: 1903014 on 36 degrees of freedom
29 (45 observations deleted due to missingness)

```

```
30 AIC: NA
31 Number of Fisher Scoring iterations: 4
```

Thus, coefficients are identical so it not affect the best estimate of claims reserves.... unless we take into account the fact that some variates are no longer significant....

```
1 > base = data.frame(Y,col,lig)
2 > base$lig[base$lig=="7"]="1"
3 > base$lig[base$lig=="9"]="1"
4 > base$lig[base$lig=="6"]="1"
5 > base$col[base$col=="5"]="1"
6 > base$col[base$col=="10"]="1"
7 > base$col[base$col=="9"]="1"
8 > base$col[base$col=="6"]="1"
9 > base$col[base$col=="4"]="1"
10 > base$col[base$col=="3"]="1"
11 > base$col[base$col=="7"]="1"
12 > base$col[base$col=="2"]="1"
```

```

13 > base$lig[base$lig=="8"]="1"
14 > summary(glm(Y~col+lig,data=base,family="quasipoisson"))
15 Call:
16 glm(formula = Y ~ col + lig, family = "quasipoisson", data = base)
17 Coefficients:
18             Estimate Std. Error t value Pr(>|t|)
19 (Intercept) 12.73401   0.07764 164.022 < 2e-16 ***
20 col8        0.28877   0.14109   2.047  0.04618 *
21 lig2        0.96246   0.10984   8.763 1.59e-11 ***
22 lig3        0.99721   0.11232   8.878 1.07e-11 ***
23 lig4        1.06465   0.11481   9.273 2.82e-12 ***
24 lig5        0.45513   0.14622   3.113  0.00312 **
25 lig10       -1.60752   0.85482  -1.881  0.06611 .
26 ---
27 (Dispersion parameter for quasipoisson family taken to be 49241.53)
28 Null deviance: 10699464 on 54 degrees of freedom
29 Residual deviance: 2442092 on 48 degrees of freedom
30 (45 observations deleted due to missingness)

```

```
31 AIC: NA
32 Number of Fisher Scoring iterations: 4
```

Thus,

```
1 > M= cbind(Y,predict(reg1,newdata=base0,type="response"),
2 +                  predict(reg2,newdata=base0,type="response"),
3 +                  predict(reg3,newdata=base , type="response"))
4 > sum(M[is.na(Y)==TRUE ,2])
5 [1] 18680856
6 > sum(M[is.na(Y)==TRUE ,3])
7 [1] 18680856
8 > sum(M[is.na(Y)==TRUE ,4])
9 [1] 18226919
```

Including an overdispersion parameter ϕ might impact the estimation of the overall reserves.

Testing for overdispersion

In order to test for overdispersion in an econometric model, we need to specify how overdispersion appears. A classical test is to assume that

$$\text{Var}(Y|\mathbf{X}) = \mathbb{E}(Y|\mathbf{X}) + \tau \mathbb{E}(Y|\mathbf{X})^2$$

which is a standard econometric model with random effect. We want to test

$$H_0 : \tau = 0 \text{ against } H_1 : \tau > 0$$

A standard test statistics is

$$T = \frac{\sum_{i=1}^n [Y_i - \hat{\lambda}_i]^2 - Y_i}{\sqrt{2 \sum_{i=1}^n \hat{\lambda}_i^2}}$$

which has a $\mathcal{N}(0, 1)$ distribution under H_0 . An alternative is to consider

$$T = \frac{\sum_{i=1}^n [Y_i - \hat{\lambda}_i]^2 - Y_i}{\sqrt{\sum_{i=1}^n [(Y_i - \hat{\lambda}_i)^2 - Y_i]^2}}$$

Those test can be found in R, respectively

```
1 > library(AER)
2 > dispersiontest(reglmp)
3 > dispersiontest(reglmp,trafo = 2)
```

An alternative is simply the following

```
1 > library(ChainLadder)
2 > an <- 10; ligne = rep(1:an, each=an); colonne = rep(1:an, an)
3 > passe = (ligne + colonne - 1)<=an; n = sum(passe)
4 > PAID=GenIns; INC=PAID
5 > INC[,2:an]=PAID[,2:an]-PAID[,1:(an-1)]
6 > Y = as.vector(INC)
7 > lig = as.factor(ligne)
8 > col = as.factor(colonne)
9 > base = data.frame(Y,col,lig)
10 > reg1=glm(Y~col+lig,data=base,family="poisson")
11 > reg2=glm(Y~col+lig,data=base,family="quasipoisson")
12 > dispersiontest(reg1)
```

```
13      Overdispersion test
14 data: reg1
15 z = 4.3942, p-value = 5.558e-06
16 alternative hypothesis: true dispersion is greater than 1
```

Alternative models for overdispersion

There is overdispersion if $\text{Var}(Y) > \mathbb{E}(Y)$, which can be obtained with a **negative binomial distribution** (which belongs to the exponential family)

```

1 > library(MASS)
2 > reg3=glm.nb(Y~col+lig,data=base)
3 > summary(reg3)
4 (Dispersion parameter for Negative Binomial(13.8349) family taken to
   be 1)
5
6             Theta:    13.83
7             Std. Err.:   2.61
8 2 x log-likelihood: -1460.766
9 > sum(exp(predict(reg3,newdata=base))[passe!=TRUE])
[1] 18085795

```

Uncertainty and overdispersion

Based on the explicit expression for the prediction error, it is possible to obtain prediction error for those three models,

```

1 > predCL=function(reg=reg1,regb=reg1b){
2 +   p = 2*6-1;
3 +   phi.P = sum(residuals(reg,"pearson")^2)/(np-p)
4 +   Sig = vcov(reg)
5 +   X = model.matrix(regb)
6 +   Cov.eta = X%*%Sig%*%t(X)
7 +   mu.hat = exp(predict(reg,newdata=data.frame(lig,col)))*futur
8 +   pe2 = phi.P * sum(mu.hat) + t(mu.hat) %*% Cov.eta %*% mu.hat
9 +   cat("Total reserve =", sum(mu.hat), "prediction error =", sqrt(pe2
+     ),sqrt(pe2)/sum(mu.hat),"\n")
10 + }
```

Avec nos trois modèles, Poisson, ODP et binomiale négative, on obtient,

```

1 > predCL(reg1,reg1b)
```

```
2 Total reserve = 18680856 prediction error = 896876.9 0.04801048
3 > predCL(reg2,reg2b)
4 Total reserve = 18680856 prediction error = 4736425 0.2535443
5 > predCL(reg3,reg3b)
6 Total reserve = 18085795 prediction error = 2058134 0.1137984
```

On the prediction error

In order to derive an estimation of the prediction error using bootstrap techniques, we have not only to generate randomly possible triangles, but also to add uncertainty in the developpement, using e.g. the fact that

$$C_{i,j+1} = \lambda_j C_{i,j} + \sigma_j \sqrt{C_{i,j}} + \varepsilon_{i,j}$$

where the noise can be assume to be Gaussian, $\mathcal{N}(0, 1)$.

The statistical interpretation is that

$$C_{i,j+1}|C_{i,j} \sim \mathcal{N}(\hat{\lambda}_j C_{i,j} + \hat{\sigma}_j^2 C_{i,j})$$

Classically we use

```

1 > CL=function(triangle){
2 +   n=nrow(triangle)
3 +   LAMBDA=rep(NA,n-1)
4 +   for(i in 1:(n-1)){

```

```

5 + LAMBDA[i]=sum(triangle[1:(n-i),i+1])/
6 +           sum(triangle[1:(n-i),i]) }
7 + DIAG=diag(triangle[,n:1])
8 + TOTO=c(1,rev(LAMBDA))
9 + return(sum(cumprod(TOTO)*DIAG-DIAG)) }
```

a natural idea is to consider

```

1 > CLboot=function(triangle,l,s){
2 + m=nrow(triangle)
3 + for(i in 2:m){
4 +   triangle[(m-i+2):m,i]=rnorm(i-1,
5 +           mean=triangle[(m-i+2):m,i-1]*l[i-1],
6 +           sd=sqrt(triangle[(m-i+2):m,i-1])*s[i-1])
7 + }
8 + ULT=triangle[,m]
9 + DIAG=diag(triangle[,m:1])
10 + return(sum(ULT-DIAG)) }
```

Then, we can run bootstrap simulations,

```
1 > base=data.frame(Y,lig,col)
2 > REG=glm(Y~lig+col,family=poisson)
3 > YP=predict(REG,newdata=base)
4 > E=residuals(REG,"pearson")
5 > PROV.BE=rep(NA,5000)
6 > PROVISION=rep(NA,5000)
7 > for(k in 1:50000){
8 + simE=sample(E,size=36,replace=TRUE)
9 + bruit=simE*sqrt(exp(YP))
10 + INCsim=exp(YP)+bruit
11 + INCM=matrix(INCsim,6,6)
12 + CUMM=INCM
13 + for(j in 2:6){CUMM[,j]=CUMM[,j-1]+INCM[,j]}
14 + PROV.BE[k]=CL(CUMM)
15 + PROVISION[k]=CLboot(CUMM,lambda,sigma)}
```

Random Generation of a Quasi-Distribution

It is also possible to generate Poisson, or quasi-Poisson random variables.

Recall that the negative binomial distribution has probability function

$$\mathbb{P}[N = k] = \frac{\Gamma(k + r)}{k! \Gamma(r)} \cdot [1 - p]^r p^k$$

where the expected value and the variance are

$$\mu = r \cdot \frac{p}{1 - p} \text{ and } \sigma^2 = \mu = r \cdot \frac{p}{(1 - p)^2}$$

Assume that $\sigma^2 = \varphi \cdot \mu$, then

$$r = \frac{\mu}{\varphi - 1} \text{ and } p = \frac{1}{\varphi}$$

```
1 > rqpois = function(n, lambda, phi) {
2 +   return( rnbinom(n, size = lambda/(1-phi), prob = 1/phi) }
```

Using GAM for claims reserving

In the case of GAM's, assume that

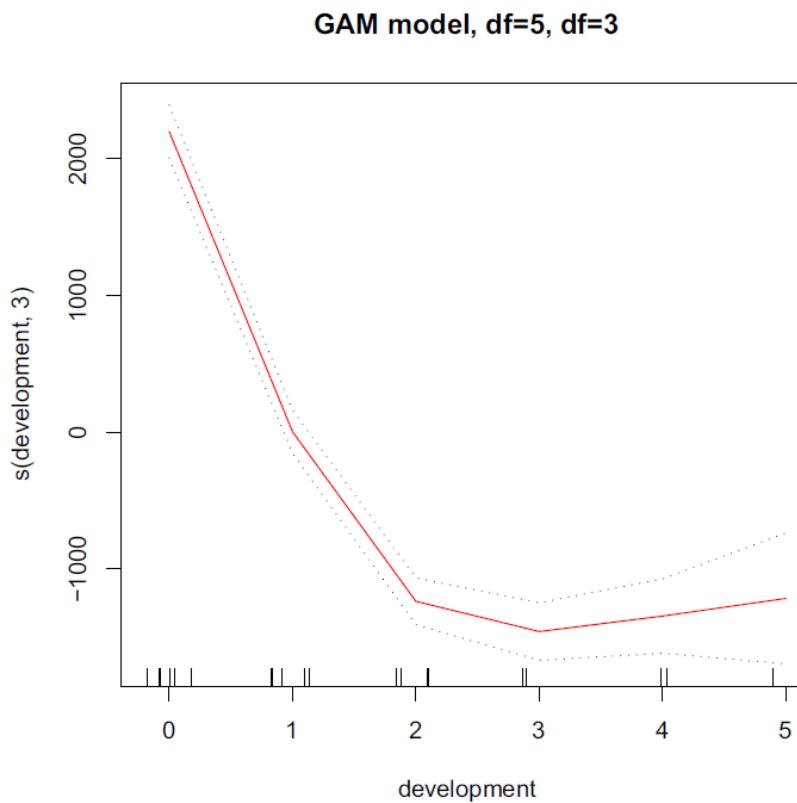
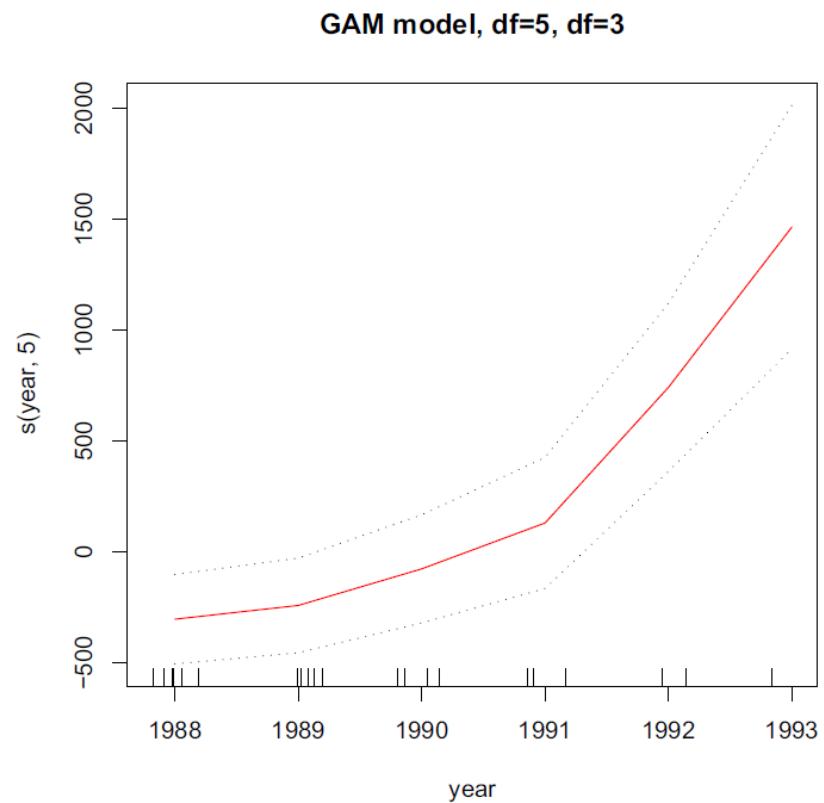
$$Y_{i,j} \sim \mathcal{L}(\theta \zeta i, j), \text{ where } \theta_{i,j} = \varphi(u(i) + v(j)),$$

where here u and v are two **unknown** functions. We still have an additive form, but on unknown transformations of explanatory variates.

Spline functions are considered to estimation functions u and v .

```

1 > library(gam)
2 > GAM=gam(payinc~s(year,5)+s(development,3),data=D,familly="Poisson")
3 > plot.gam(GAM,se=T,col="red",ask=TRUE,main="GAM model, df=5, df=3")
```



Dealing with negative increments

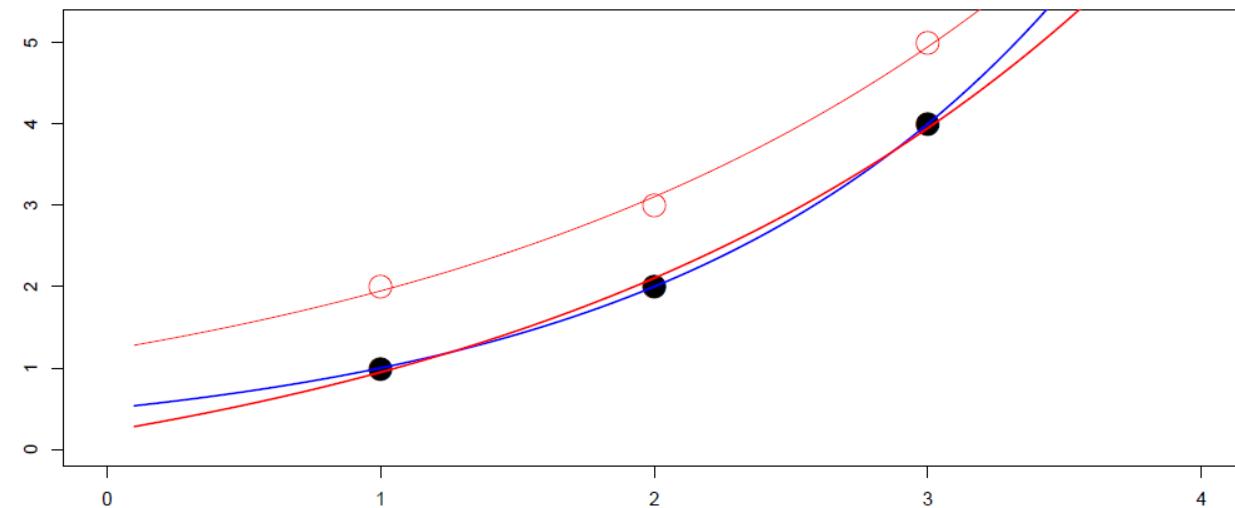
Negative incremental values can arise due to timing of reinsurance, recoveries, cancellation of outstanding claims.

One might argue that the problem is **more with the data than with the methods**.

[England & Verall \(2002\)](#) mention that the Gaussian model is less affected by the presence of negative incremental values. Unfortunately, one can hardly assume that data are Gaussian because of the skewness. [Renshaw & Verrall \(1994\)](#) suggested to add a “small constant” to the past data and to subtract this constant from forecasts at the end.

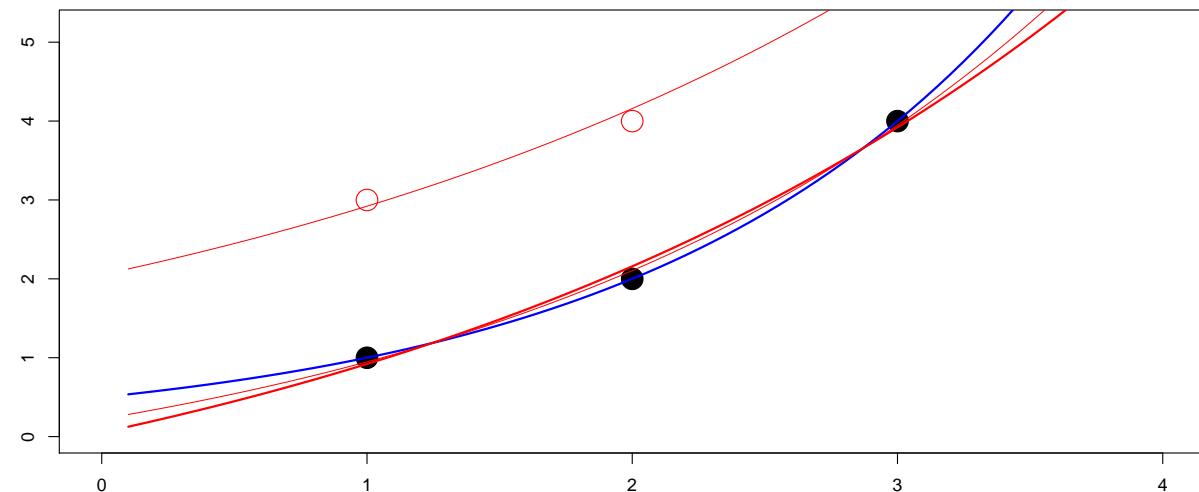
Dealing with negative increments

A *classical* technique to avoid negative payments is to consider a **translation** of the incremental triangle, i.e. $Y_{i,j}^+ = Y_{i,j} + \kappa$ such that $Y_{i,j}^+ > 0$ for all i, j .



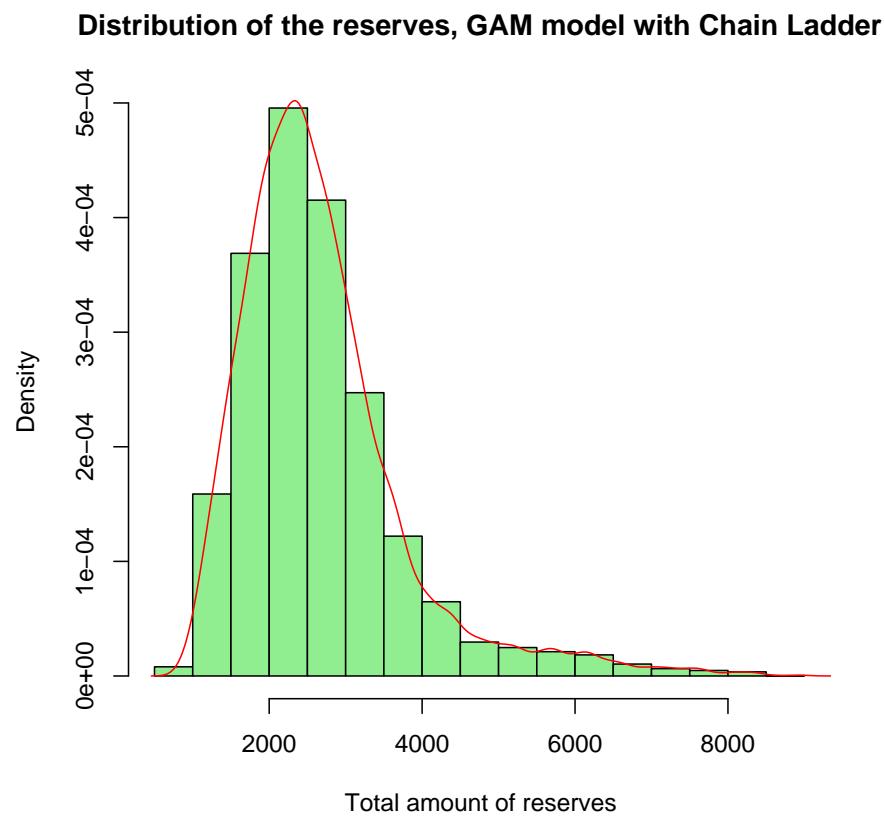
Dealing with negative increments

A *classical* technique to avoid negative payments is to consider a **translation** of the incremental triangle, i.e. $Y_{i,j}^+ = Y_{i,j} + \kappa$ such that $Y_{i,j}^+ > 0$ for all i, j .



Why a Poisson regression model ?

The is no reason to assume that incremental payments are Poisson distribution.
The only motivation here is that the expected value is the same as the Chain Ladder estimate.



Tweedie ?

The density of a tweedie model with power function p would be

```

1 > ftweedie = function(y,p,mu,psi){
2 + if(p==2){f = dgamma(y, 1/psi, 1/(psi*mu))} else
3 + if(p==1){f = dpois(y/psi, mu/psi)} else
4 + {lambda = mu^(2-p)/psi /(2-p)
5 + if(y==0){ f = exp(-lambda)} else
6 + { alpha = (2-p)/(p-1)
7 +   beta = 1 / (psi * (p-1) * mu^(p-1))
8 +   k = max(10, ceiling(lambda + 7*sqrt(lambda)))
9 +   f = sum(dpois(1:k,lambda) * dgamma(y, alpha*(1:k),beta))
10+ }
11+ return(f)
12+ }
```

A numerical problem is that we should have *no* missing values in the regression, so artificially, consider

```

1 > source("http://freakonometrics.free.fr/bases.R")
```

```

2 > library(statmod)
3 > an <- 6; ligne = rep(1:an, each=an); colonne = rep(1:an, an)
4 > passe = (ligne + colonne - 1) <= an; n = sum(passe)
5 > INC=PAID
6 > INC[,2:6]=PAID[,2:6]-PAID[,1:5]
7 > Y = as.vector(INC)
8 > lig = as.factor(ligne)
9 > col = as.factor(colonne)
10 > y = Y[passe]
11 > Y[is.na(Y)]=.01

```

Then, we can run an econometric regression

```

1 > pltweedie <- function(pow){
2 +   regt = glm(Y~lig+col, tweedie(pow,0))
3 +   reserve = sum(fitted.values(regt)[!passe])
4 +   dev = deviance(regt)
5 +   phi.hat = dev/n
6 +   mu = fitted.values(regt)[passe]

```

```

7 + hat.logL = 0
8 + for (k in 1:length(y)){
9 +   hat.logL <- hat.logL + log(ftweedie(y[k], pow, mu[k], phi.hat))
10 + }
11 + cat("Puissance =", round(pow,3), "phi =", round(phi.hat,2),
12 + "Reserve (tot) =", round(reserve), "logL =", round(hat.logL,3))
13 + hat.logL}
14 > for(pow in c(1,1.25,1.5,1.75,2)){pltweedie(pow)}
14 Puissance = 1      phi = 166.95  Reserve (tot) = 1345  logL = -Inf
15 Puissance = 1.25  phi = 42.92   Reserve (tot) = 1216  logL = -151.72
16 Puissance = 1.5   phi = 15.8    Reserve (tot) = 996   logL = -145.232
17 Puissance = 1.75  phi = 9.02   Reserve (tot) = 609   logL = -153.997
18 Puissance = 2     phi = 6.78   Reserve (tot) = 125   logL = -170.614

```

It is also possible to run a optimization routine,

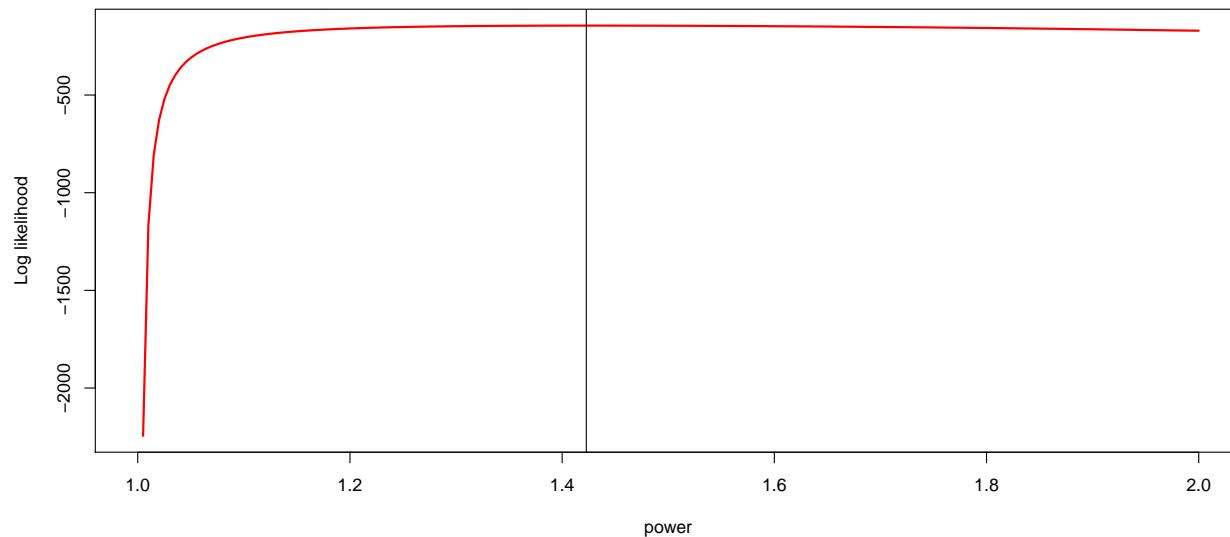
```

1 > optimize(pltweedie, c(1.01,1.99), tol=1e-4,maximum = TRUE)
2 -144.624
3 $maximum

```

```
4 [1] 1.427873  
5  
6 $objective  
7 [1] -144.6237
```

Thus, here the Poisson model might not be the appropriate one,



Bayesian Models in Claims Reserving

The first idea is to consider some credibility based model, with

$$\widehat{C}_{i,n} = Z \cdot \widehat{C}_{i,n}^{\text{Mack}} + [1 - Z] \cdot \mu_i$$

given some a priori μ_i .

For instance [Benktander \(1976\)](#) and [Hovinen \(1981\)](#) suggested

$$Z = 1 - [1 - \beta_i]^2 \text{ where } \beta_i = \prod_{k=n-i}^{n-1} \frac{1}{\widehat{\lambda}_k}$$

Note that

$$\widehat{C}_{i,n} = C_{i,n-i} + [1 - \beta_i] \left(\beta_i \cdot \widehat{C}_{i,n}^{\text{Mack}} + [1 - \beta_i] \cdot \mu_i \right)$$

More generally, consider the [Cape-Code](#) technique,

$$C_{i,n} = C_{i,n-i} + \left(1 - \frac{C_{i,n-i}}{C_{i,n}} \right) C_{i,n}$$

sous la forme

$$C_{i,n} = C_{i,n-i} + \left(1 - \frac{C_{i,n-i}}{C_{i,n}}\right) LR_i \cdot P_i,$$

où LR_i correspond au *loss ratio* pour l'année i , i.e. $LR_i = C_{i,n}/P_i$. L'idée de la méthode dite *Cape-Code* est d'écrire une forme plus générale,

$$C_{i,n} = C_{i,n-i} + (1 - \pi_{n-i}) LR_i P_i$$

où π_{n-i} correspond à une cadence de paiement, et peut être estimé par la méthode Chain Ladder. Quant aux LR_i il s'agit des *loss ratio* cibles, correspondant à un avis d'expert. On peut aussi proposer un même ratio cible pour plusieurs années de survenance. On posera alors

$$R_i = C_{i,n} - C_{i,n-i} = (1 - \pi_{n-i}) LR_{\mathcal{A}} P_i.$$

pour $i \in \mathcal{A}$, où

$$LR_{\mathcal{A}} = \frac{\sum_{k \in \mathcal{A}} C_{n,n-k}}{\sum_{k \in \mathcal{A}} \pi_{n-k} P_k}.$$

Dans un premier temps, on peut calculer les π_i à partir de la méthode Chain Ladder, i.e.

$$\pi_{n-i} = \frac{C_{i,n-i}}{C_{i,n}}$$

où la charge ultime est celle prédictive pas la méthode Chain-Ladder.

```

1 > Cultime = MackChainLadder(PAID)$FullTriangle[,6]
2 > (PI <- (1-Cdiag/Cultime))
3      1       2       3       4       5       6
4 0.00000 0.00471 0.00656 0.01086 0.02204 0.29181
5 > LR <- TRIANGLE[,6]/PREMIUM
6 > Cdiag <- diag(PAID[,6:1])
7 > (Cultime-Cdiag)/(LR*PREMIUM)
8      1       2       3       4       5       6
9 0.00000 0.00471 0.00656 0.01086 0.02204 0.29181

```

Si on suppose ensuite que $\mathcal{A} = \{1, 2, \dots, n\}$, alors

```
1 > LR=sum(TRIANGLE[,6])/sum(PREMIUM)
```

```
2 > PI*LR*PREMIUM
3      1       2       3       4       5       6
4     0.0    24.6   35.6   62.7  139.6  2095.3
5 > sum(PI*LR*PREMIUM)
6 [1] 2358
```

On obtient ici un montant de provision total inférieur à celui obtenu par la méthode Chain Ladder puisque le montant de provisions vaut ici 2357.756.

Modèles bayésiens et Chain Ladder

De manière générale, un méthode bayésienne repose sur deux hypothèses

- une loi **a priori** pour les paramètres du modèle ($X_{i,j}$, $C_{i,j}$, $\lambda_{i,j}$, $LR_{i,j} = C_{i,j}/P_j$, etc)
- une technique pour calculer les lois **a posteriori**, qui sont en général assez complexes.

Modèles bayésiens pour les nombres de sinistres

Soit $N_{i,j}$ l'incrément du nombre de sinistres, i.e. le nombre de sinistres survenus l'année i , déclarés l'année $i + j$.

On note M_i le nombre total de sinistres par année de survenance, i.e.

$M_i = N_{i,0} + N_{i,1} + \dots$. Supposons que $M_i \sim \mathcal{P}(\lambda_i)$, et que $\mathbf{p} = (p_0, p_1, \dots, p_n)$ désigne les proportions des paiements par année de déroulé.

Conditionnellement à $M_i = m_i$, les années de survenance sont indépendantes, et le vecteur du nombre de sinistres survenus année l'année i suit une loi multinomiale $\mathcal{M}(m_i, \mathbf{p})$.

Modèles bayésiens pour les nombres de sinistres

La vraisemblance $\mathcal{L}(M_0, M_1, \dots, M_n, \mathbf{p} | N_{i,j})$ est alors

$$\prod_{i=0}^n \frac{M_i!}{(M_i - N_{n-i}^\star)! N_{i,0}! N_{i,1}! \cdots N_{i,n-i}!} [1 - p_{n-i}^\star]^{M_i - N_{n-i}^\star} p_0^{N_{i,0}} p_1^{N_{i,1}} \cdots p_{n-i}^{N_{i,n-i}}$$

où $N_{n-i}^\star = N_0 + N_1 + \cdots + N_{n-i}$ et $p_{n-i}^\star = p_0 + p_1 + \cdots + p_{n-i}$.

Il faut ensuite de donner une loi **a priori** pour les paramètres. La loi **a posteriori** sera alors proportionnelle produit entre la vraisemblance et cette loi a priori.

Modèles bayésiens pour les montants agrégés

On pose $Y_{i,j} = \log(C_{i,j})$, et on suppose que $Y_{i,j} = \mu + \alpha_i + \beta_j + \varepsilon_{i,j}$, où $\varepsilon_{i,j} \sim \mathcal{N}(0, \sigma^2)$. Aussi, $Y_{i,j}$ suit une loi normale,

$$f(y_{i,j} | \mu, \boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma^2) \propto \frac{1}{\sigma} \exp \left(-\frac{1}{2\sigma^2} [y_{i,j} - \mu - \alpha_i - \beta_j]^2 \right),$$

et la vraisemblance est alors

$$\mathcal{L}(\boldsymbol{\theta}, \sigma | \mathbf{Y}) \propto \sigma^{-m} \exp \left(\sum_{i,j} [y_{i,j} - \mu - \alpha_i - \beta_j]^2 \right)$$

où $m = (n(n+1)/2)$ désigne le nombre d'observations passées. La difficulté est alors de spécifier une loi a priori pour $(\boldsymbol{\theta}, \sigma^2)$, i.e. $(\mu, \boldsymbol{\alpha}, \boldsymbol{\beta}, \sigma^2)$.

MCMC and Bayesian Models

We have a sample $\mathbf{x} = \{x_1, \dots, x_d\}$ i.i.d. from distribution $f_\theta(\cdot)$.

In predictive modeling, we need $\mathbb{E}(g(X)|\mathbf{x}) = \int g(x)f_{\theta|\mathbf{x}}(x)dx$ where

$$f_{\theta|\mathbf{x}}(x) = f(x|\mathbf{x}) = \int f(x|\theta) \cdot \pi(\theta|\mathbf{x})d\theta$$

How can we derive $\pi(\theta|\mathbf{x})$? Can we sample from $\pi(\theta|\mathbf{x})$ (and use monte carlo technique to approximate the integral) ?

Hastings-Metropolis

Back to our problem, we want to sample from $\pi(\theta|\mathbf{x})$

i.e. generate $\theta_1, \dots, \theta_n, \dots$ from $\pi(\theta|\mathbf{x})$.

Hastings-Metropolis sampler will generate a Markov Chain (θ_t) as follows,

- generate θ_1
- generate θ^* and $U \sim \mathcal{U}([0, 1])$,

$$\text{compute } R = \frac{\pi(\theta^*|\mathbf{x})}{\pi(\theta_t|\mathbf{x})} \frac{P(\theta_t|\theta^*)}{P(\theta^*|\theta_{t-1})}$$

if $U < R$ set $\theta_{t+1} = \theta^*$

if $U \geq R$ set $\theta_{t+1} = \theta_t$

R is the acceptance ratio, we accept the new state θ^* with probability $\min\{1, R\}$.

Hastings-Metropolis

Observe that

$$R = \frac{\pi(\theta^*) \cdot f(\mathbf{x}|\theta^*)}{\pi(\theta_t) \cdot f(\mathbf{x}|\theta_t)} \frac{P(\theta_t|\theta^*)}{P(\theta^*|\theta_{t-1})}$$

In a more general case, we can have a Markov process, not a Markov chain.

E.g. $P(\theta^*|\theta_t) \sim \mathcal{N}(\theta_t, 1)$

Using MCMC to generate Gaussian values

```
1 > metrop1 <- function(n=1000,eps=0.5){  
2 + vec <- vector("numeric", n)  
3 + x=0  
4 + vec[1] <- x  
5 + for (i in 2:n) {  
6 +   innov <- runif(1,-eps,eps)  
7 +   mov <- x+innov  
8 +   aprob <- min(1,dnorm(mov)/dnorm(x))  
9 +   u <- runif(1)  
10 +  if (u < aprob)  
11 +    x <- mov  
12 +  vec[i] <- x  
13 + }  
14 + return(vec)}
```

Using MCMC to generate Gaussian values

```

1 > plot.mcmc <- function(mcmc.out){
2 +   op <- par(mfrow=c(2,2))
3 +   plot(ts(mcmc.out), col="red")
4 +   hist(mcmc.out, 30, probability=TRUE,
5 +     col="light blue")
6 +   lines(seq(-4,4, by=.01), dnorm(seq
7 +     (-4,4,
8 +       by=.01)), col="red")
9 +   qqnorm(mcmc.out)
10 +  abline(a=mean(mcmc.out), b=sd(mcmc.
11 +    out))
12 +  acf(mcmc.out, col="blue", lag.max
13 +    =100)
14 +  par(op)}
15 > metrop.out<-metrop1(10000,1)
16 > plot.mcmc(metrop.out)

```

Heuristics on Hastings-Metropolis

In standard Monte Carlo, generate θ_i 's i.i.d., then

$$\frac{1}{n} \sum_{i=1}^n g(\theta_i) \rightarrow \mathbb{E}[g(\theta)] = \int g(\theta) \pi(\theta) d\theta$$

(strong law of large numbers).

Well-behaved Markov Chains (\mathbf{P} aperiodic, irreducible, positive recurrent) can satisfy some ergodic property, similar to that LLN. More precisely,

- \mathbf{P} has a unique stationary distribution λ , i.e. $\lambda = \lambda \times \mathbf{P}$
- ergodic theorem

$$\frac{1}{n} \sum_{i=1}^n g(\theta_i) \rightarrow \int g(\theta) \lambda(\theta) d\theta$$

even if θ_i 's are not independent.

Heuristics on Hastings-Metropolis

Remark The conditions mentioned above are

- aperiodic, the chain does not regularly return to any state in multiples of some k .
- irreducible, the state can go from any state to any other state in some finite number of steps
- positively recurrent, the chain will return to any particular state with probability 1, and finite expected return time

MCMC and Loss Models

Example A Tweedie model, $\mathbb{E}(X) = \mu$ and $\text{Var}(X) = \varphi \cdot \mu^p$. Here assume that φ and p are given, and μ is the unknown parameter.

→ need a predictive distribution for μ given x .

Consider the following transition kernel (a Gamma distribution)

$$\mu | \mu_t \sim \mathcal{G} \left(\frac{\mu_t}{\alpha}, \alpha \right)$$

with $\mathbb{E}(\mu | \mu_t) = \mu_t$ and $\text{CV}(\mu) = \frac{1}{\sqrt{\alpha}}$.

Use some a priori distribution, e.g. $\mathcal{G}(\alpha_0, \beta_0)$.

MCMC and Loss Models

- generate μ_1
- at step t : generate $\mu^* \sim \mathcal{G}(\alpha^{-1}\mu_t, \alpha)$ and $U \sim \mathcal{U}([0, 1])$,

$$\text{compute } R = \frac{\pi(\mu^*) \cdot f(\mathbf{x}|\mu^*)}{\pi(\mu_t) \cdot f(\mathbf{x}|\theta_t)} \frac{P_\alpha(\mu_t|\theta^*)}{P_\alpha(\theta^*|\theta_{t-1})}$$

if $U < R$ set $\theta_{t+1} = \theta^*$

if $U \geq R$ set $\theta_{t+1} = \theta_t$

where

$$f(\mathbf{x}|\mu) = \mathcal{L}(\mu) = \prod_{i=1}^n f(x_i|\mu, p, \varphi),$$

$f(x \cdot | \mu, p, \varphi)$ being the density of the Tweedie distribution, `dtweedie function (x, p, mu, phi)` from `library(tweedie)`.

```

1 > p=2 ; phi=2/5
2 > set.seed(1) ; X <- rtweedie(50,p,10,phi)
3 > metrop2 <- function(n=10000,a0=10,
4 + b0=1,alpha=1){
5 +   vec <- vector("numeric", n)
6 +   vec[1] <- rgamma(1,a0,b0)
7 +   for (i in 2:n){
8 +     mustar <- rgamma(1,vec[i-1]/alpha,alpha)
9 +     R=prod(dtweedie(X,p,mustar,phi)/dtweedie
10 +    (X,p,vec[i-1],phi))*dgamma(mustar,a0,b0)
11 +    dgamma(vec[i-1],a0,b0)* dgamma(vec[i-1],
12 +    mustar/alpha,alpha)/dgamma(mustar,
13 +    vec[i-1]/alpha,alpha)
14 +    aprob <- min(1,R)
15 +    ifelse(runif(1) < aprob,vec[i]<-mustar,
16 +    vec[i]<-vec[i-1])}
17 +   return(vec)}
18 > metrop.output<-metrop2(10000,alpha=1)

```

Gibbs Sampler For a multivariate problem, it is possible to use Gibbs sampler.

Example Assume that the loss ratio of a company has a lognormal distribution, $LN(\mu, \sigma^2)$, .e.g

```
1 > LR <- c(0.958, 0.614, 0.977, 0.921, 0.756)
```

Example Assume that we have a sample \mathbf{x} from a $\mathcal{N}(\mu, \sigma^2)$. We want the posterior distribution of $\boldsymbol{\theta} = (\mu, \sigma^2)$ given \mathbf{x} . Observe here that if priors are Gaussian $\mathcal{N}(\mu_0, \tau^2)$ and the inverse Gamma distribution $IG(a, b)$, then

$$\begin{cases} \mu | \sigma^2, \mathbf{x} \sim \mathcal{N}\left(\frac{\sigma^2}{\sigma^2 + n\tau^2}\mu_0 + \frac{n\tau^2}{\sigma^2 + n\tau^2}\bar{x}, \frac{\sigma^2\tau^2}{\sigma^2 + n\tau^2}\right) \\ \sigma^2 | \mu, \mathbf{x} \sim IG\left(\frac{n}{2} + a, \frac{1}{2} \sum_{i=1}^n [x_i - \mu]^2 + b\right) \end{cases}$$

More generally, we need the conditional distribution of $\theta_k | \boldsymbol{\theta}_{-k}, \mathbf{x}$, for all k .

```
1 > x <- log(LR)
```

Gibbs Sampler

```

1 > xbar <- mean(x)
2 > mu <- sigma2=rep(0,10000)
3 > sigma2[1] <- 1/rgamma(1,shape=1,rate=1)
4 > Z <- sigma2[1]/(sigma2[1]+n*1)
5 > mu[1] <- rnorm(1,m=Z*0+(1-Z)*xbar,
6 + sd=sqrt(1*Z))
7 > for (i in 2:10000){
8 + Z <- sigma2[i-1]/(sigma2[i-1]+n*1)
9 + mu[i] <- rnorm(1,m=Z*0+(1-Z)*xbar,
10 + sd=sqrt(1*Z))
11 + sigma2[i] <- 1/rgamma(1,shape=n/2+1,
12 + rate <- (1/2)*(sum((x-mu[i])^2)+1)
13 + }

```

Gibbs Sampler

Example Consider some vector $\mathbf{X} = (X_1, \dots, X_d)$ with independent components, $X_i \sim \mathcal{E}(\lambda_i)$. We sample to sample from \mathbf{X} given $\mathbf{X}^\top \mathbf{1} > s$ for some threshold $s > 0$.

- start with some starting point \mathbf{x}_0 such that $\mathbf{x}_0^\top \mathbf{1} > s$
- pick up (randomly) $i \in \{1, \dots, d\}$

X_i given $X_i > s - \mathbf{x}_{(-i)}^\top \mathbf{1}$ has an Exponential distribution $\mathcal{E}(\lambda_i)$

draw $Y \sim \mathcal{E}(\lambda_i)$ and set $x_i = y + (s - \mathbf{x}_{(-i)}^\top \mathbf{1})_+$ until $\mathbf{x}_{(-i)}^\top \mathbf{1} + x_i > s$

E.g. losses and allocated expenses

Gibbs Sampler

```
1 > sim <- NULL
2 > lambda <- c(1,2)
3 > X <- c(3,3)
4 > s <- 5
5 > for(k in 1:1000){
6 + i <- sample(1:2,1)
7 + X[i] <- rexp(1,lambda[i])+
8 + max(0,s-sum(X[-i]))
9 + while(sum(X)<s){
10 + X[i] <- rexp(1,lambda[i])+
11 + max(0,s-sum(X[-i])) }
12 + sim <- rbind(sim,X) }
```

JAGS and STAN

Martyn Plummer developed **JAGS** *Just another Gibbs sampler* in 2007 (stable since 2013) in `library(rjags)`. It is an open-source, enhanced, cross-platform version of an earlier engine BUGS (Bayesian inference Using Gibbs Sampling).

STAN `library(Rstan)` is a newer tool that uses the Hamiltonian Monte Carlo (HMC) sampler.

HMC uses information about the derivative of the posterior probability density to improve the algorithm. These derivatives are supplied by algorithm differentiation in C/C++ codes.

JAGS on the $\mathcal{N}(\mu, \sigma^2)$ distribution

```

1 > jags.data <- list(n=length(LR),
2 + logLR=log(LR), mu0=-.2, sigma0=0.02,
1 library(runjags)
2 jags.model <- "
3 + model {
4 + mu ~ dnorm(mu0, 1/(sigma0^2))
5 + g ~ dgamma(k0, theta0)
6 + sigma <- 1 / g
7 + for (i in 1:n) {
8 + logLR[i] ~ dnorm(mu, g^2)
9 +
10 + }"
11 > jags.init <- list(list(mu=log(1.2),
12 + g=1/0.5^2),
13 + list(mu=log(.8),
14 + g=1/.2^2))
15 > jags.out <- autorun.jags(jags.model,
16 + data=jags.data, inits=jags.init,
17 + monitor=c("mu", "sigma"), n.chains=2)
18 traceplot(model.out$mcmc)
19 summary(model.out)

```

STAN on the $\mathcal{N}(\mu, \sigma^2)$ distribution

```

1 > library(rstan)           1 + model }
2 > stan.model <- "          2 + mu ~ normal(mu0, sigma0);
3 + data {                   3 + sigma ~ inv_gamma(k0, theta0);
4 + int<lower=0> n;         4 + for (i in 1:n)
5 + vector[n] LR;           5 + log(LR[i]) ~ normal(mu, sigma);
6 + real mu0;               6 + }
7 + real<lower=0> sigma0;   7
8 + real<lower=0> k0;        8 > stan.data <- list(n=length(LR), r=LR, mu0=
9 + real<lower=0> theta0;    9 + mu0,
10 + }                      9 + sigma0=sigma0, k0=k0, theta0=theta0)
11 + parameters {           10 > stan.out <- stan(model_code=stan.model,
12 + real mu;                11 + data=stan.data, seed=2)
13 + real<lower=0> sigma;   12 > traceplot(stan.out)
14 + }                      13 > print(stan.out, digits_summary=2)

```

MCMC and Loss Models

Example Consider some simple time series of Loss Ratios,

$$LR_t \sim \mathcal{N}(\mu_t, \sigma^2) \text{ where } \mu_t = \phi\mu_{t-1} + \varepsilon_t$$

E.g. in JAGS we can define the vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_T)$ recursively

```

1 + model {
2 + mu[1] ~ dnorm(mu0, 1/(sigma0^2))
3 + for (t in 2:T) { mu[t] ~ dnorm(mu[t-1], 1/(sigma0^2)) }
4 }
```

A Bayesian version of Chain Ladder

Assume that $\lambda_{i,j} \sim \mathcal{N}\left(\mu_j, \frac{\tau_j}{C_{i,j}}\right)$.

We can use Gibbs sampler to get the distribution of the transition factors, as well as a distribution for the reserves,

```
1 > source("http://freakonometrics.free.fr/triangleCL.R")
2 > source("http://freakonometrics.free.fr/bayesCL.R")
3 > mcmcCL<-bayesian.triangle(PAID)
```

```
1 > plot.mcmc(mcmcCL$Lambda[,1])
2 > plot.mcmc(mcmcCL$Lambda[,2])
3 > plot.mcmc(mcmcCL$reserves[,6])
4 > plot.mcmc(mcmcCL$reserves[,7])
5 > library(ChainLadder)
6 > MCL<-MackChainLadder(PAID)
7 > m<-sum(MCL$FullTriangle[,6]-
8 + diag(MCL$FullTriangle[,6:1]))
9 > stdev<-MCL$Total.Mack.S.E
10 > hist(mcmcCL$reserves[,7],probability=
    TRUE,
11 > breaks=20,col="light blue")
12 > x=seq(2000,3000,by=10)
13 > y=dnorm(x,m,stdev)
14 > lines(x,y,col="red")
```

Autres modèles bayésiens

Dans le cadre des modèles de provisionnement, on suppose

$$\lambda_{i,j} | \lambda_j, \sigma_j^2, C_{i,j} \sim \mathcal{N} \left(\lambda_j, \frac{\sigma_j^2}{C_{i,j}} \right)$$

Notons $\gamma_j = \log(\lambda_j)$. λ désigne l'ensemble des observations, i.e. $\lambda_{i,j}$, et le paramètre que l'on cherche à estimer est γ . La log-vraisemblance est alors

$$\log \mathcal{L}(\lambda | \gamma, C, \sigma^2) = \sum_{i,j} \left(\log \left(\frac{C_{i,j}}{\sigma_j^2} \right) - \frac{C_{i,j}}{\sigma_j^2} [\lambda_{i,j} - \exp(\gamma_j)]^2 \right)$$

En utilisant le théorème de Bayes

$$\underbrace{\log \mathcal{L}(\lambda | \gamma, C, \sigma^2)}_{\text{a posteriori}} = \underbrace{\log \pi(\gamma)}_{\text{a priori}} + \underbrace{\log \mathcal{L}(\gamma | \lambda, C, \sigma^2)}_{\text{log vraisemblance}} + \text{constante}$$

Si on utilise une loi uniforme comme loi a priori, on obtient

$$\log \mathcal{L}(\lambda | \gamma, C, \sigma^2) = \log \mathcal{L}(\gamma | \lambda, C, \sigma^2) + \text{constante}$$

Les calculs de lois conditionnelles peuvent être simples dans certains cas (très limités). De manière générale, on utilise des méthodes de simulation pour approcher les lois. En particulier, on peut utiliser les algorithmes de **Gibbs** ou d'**Hastings-Metropolis**.

On part d'un vecteur initial $\gamma^{(0)} = (\gamma_1^{(0)}, \dots, \gamma_m^{(0)})$, puis

$$\left\{ \begin{array}{l} \gamma_1^{(k+1)} \sim f(\cdot | \gamma_2^{(k)}, \dots, \gamma_m^{(k)}, \lambda, C, \sigma) \\ \gamma_2^{(k+1)} \sim f(\cdot | \gamma_1^{(k+1)}, \gamma_3^{(k)}, \dots, \gamma_m^{(k)}, \lambda, C, \sigma) \\ \gamma_3^{(k+1)} \sim f(\cdot | \gamma_1^{(k+1)}, \gamma_2^{(k+1)}, \gamma_4^{(k)}, \dots, \gamma_m^{(k)}, \lambda, C, \sigma) \\ \vdots \\ \gamma_{m-1}^{(k+1)} \sim f(\cdot | \gamma_1^{(k+1)}, \gamma_2^{(k+1)}, \dots, \gamma_{m-2}^{(k+1)}, \gamma_m^{(k)}, \lambda, C, \sigma) \\ \gamma_m^{(k+1)} \sim f(\cdot | \gamma_1^{(k+1)}, \gamma_2^{(k+1)}, \dots, \gamma_{m-1}^{(k+1)}, \lambda, C, \sigma) \end{array} \right.$$

A l'aide de cet algorithme, on simule alors de triangles C , puis on estime la process error.

L'algorithme d'**adaptive rejection metropolis sampling** peut alors être utiliser pour simuler ces différentes lois conditionnelle (cf Balson (2008)).

La **méthode de rejet** est basé sur l'idée suivante

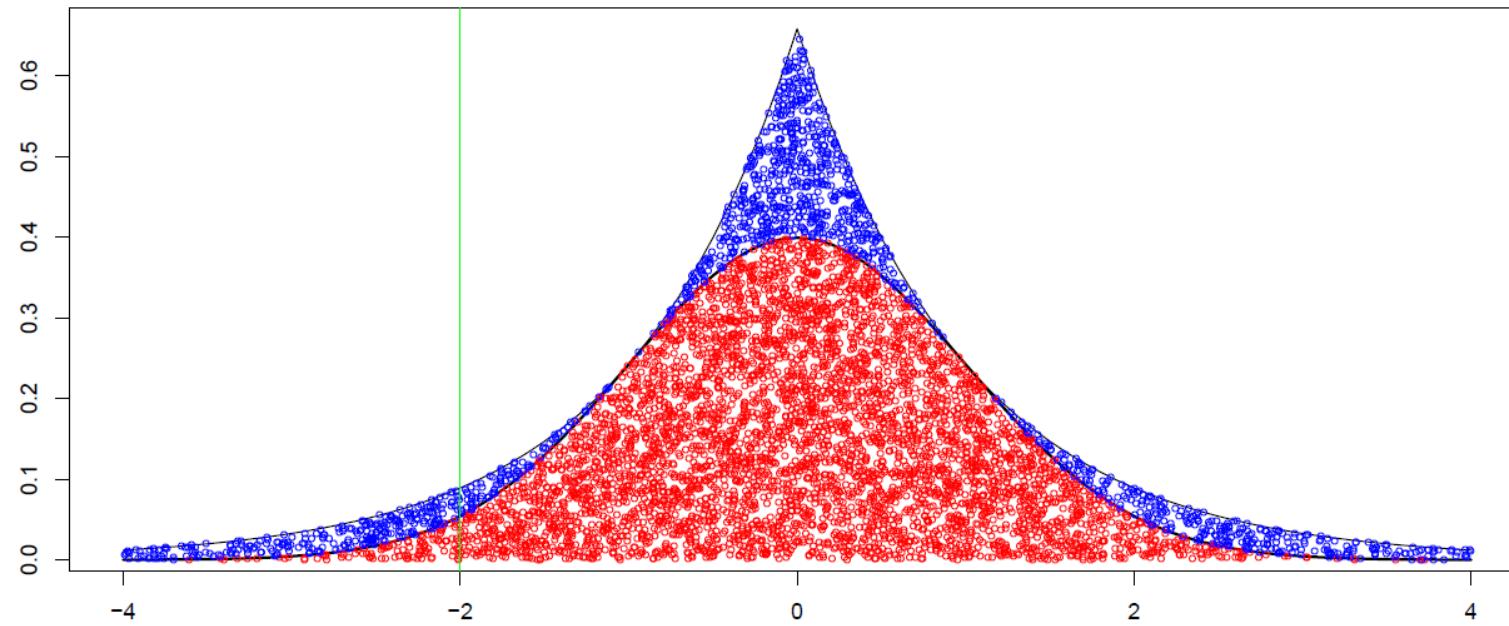
- on souhaite tirer (indépendemment) suivant une loi f , qu'on ne sait pas simuler
- on sait simuler suivant une loi g qui vérifie $f(x) \leq Mg(x)$, pour tout x , où M peut être calculée.

L'algorithme pour tirer suivant f est alors le suivant

- faire une boucle
 - tirer Y selon la loi g
 - tirer U selon la loi uniforme sur $[0, 1]$, indépendamment de Y ,
- tant que $U > \frac{f(Y)}{Mg(Y)}$.

- poser $X = Y$.

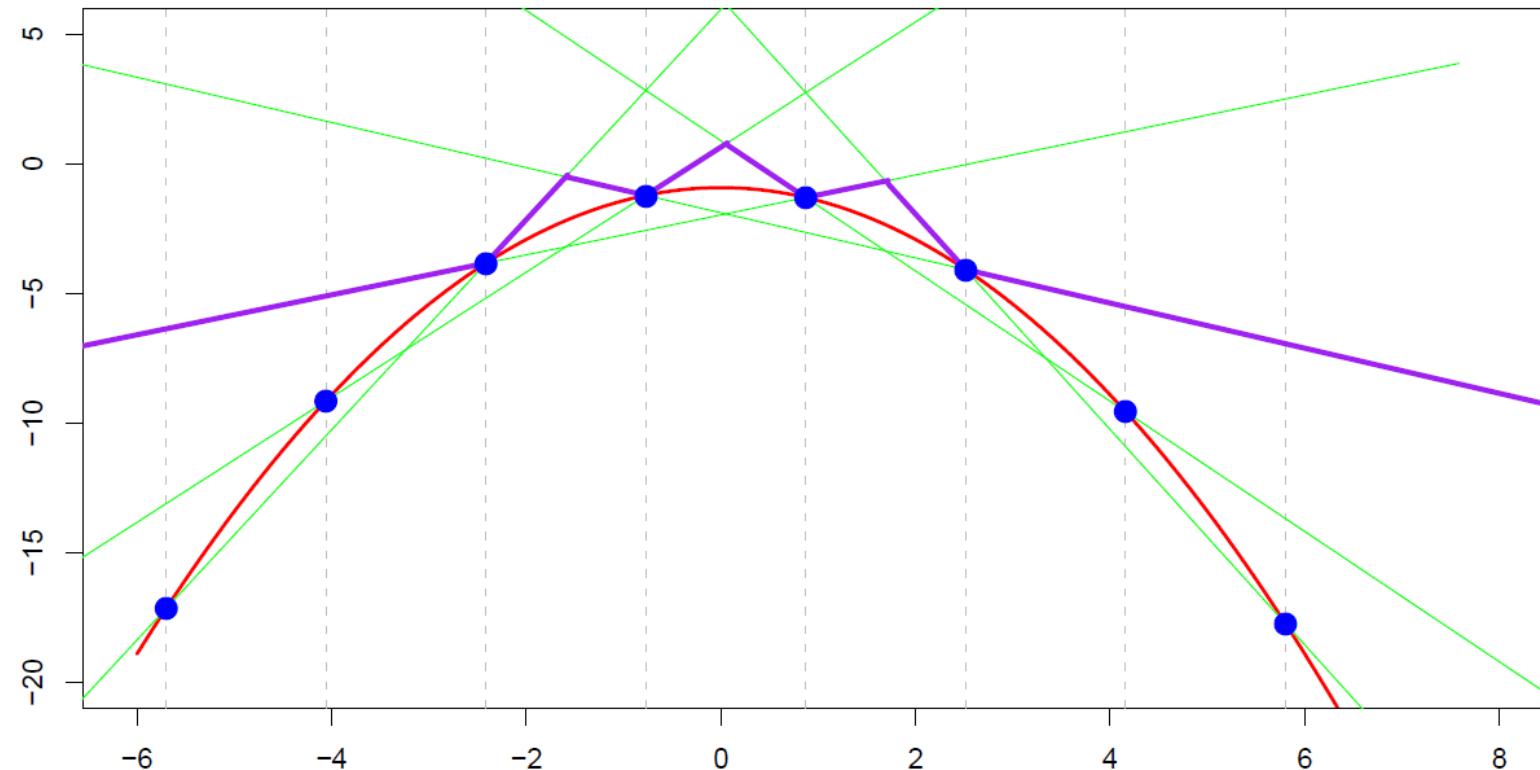
On peut utiliser cette technique pour simuler une loi normale à partir d'une loi de Laplace, de densité $g(x) = 0.5 \cdot \exp(-|x|)$, avec $M = \sqrt{2e\pi^{-1}}$. Mais cet algorithme est très coûteux en temps s'il y a beaucoup de rejets,



L'**adaptative rejection sampling** est une extension de cet algorithme, à condition d'avoir une densité **log-concave**. On parle aussi de **méthode des cordes**.

On majore localement la fonction $\log f$ par des fonctions linéaires. On construit alors une enveloppe à $\log f$.

On majore alors f par une fonction g_n qui va dépendre du pas.



Formellement, on construit $L_{i,j}(x)$ la droite reliant les points $(x_i, \log(f(x_i)))$ et

$(x_j, \log(f(x_j)))$. On pose alors

$$h_n(x) = \min \{L_{i-1,i}(x), L_{i+1,i+2}(x)\},$$

qui définit alors une enveloppe de $\log(f)$ (par concavité de $\log(f)$). On utilise alors un algorithme de rejet avec comme fonction de référence

$$g_n(x) = \frac{\exp(h_n(x))}{\int \exp(h_n(t))dt} \text{ normalisée pour définir une densité.}$$

- faire une boucle
 - tirer Y selon la loi g_n
 - tirer U selon la loi uniforme sur $[0, 1]$, indépendamment de Y ,
- tant que $U > \frac{f(Y)}{\exp(h_n(Y))}$.
- poser $X = Y$.

Enfin, l'**adaptive rejection metropolis sampling** rajoute une étape supplémentaire, dans le cas des densité non log-concave. L'idée est d'utiliser la technique précédente, même si h_n n'est plus forcément une enveloppe de $\log(f)$, puis de rajouter une étape de rejet supplémentaire. Rappelons que l'on cherche à implémenter un algorithme de Gibbs, c'est à dire créer une suite de variables X_1, X_2, \dots .

Supposons que l'on dispose de X_{k-1} . Pour tirer X_k , on utilise l'algorithme précédent, et la nouvelle étape de rejet est la suivante

- tirer U selon la loi uniforme sur $[0, 1]$, indépendamment de X et de X_{k-1} ,
 - si $U > \min \left\{ 1, \frac{f(X) \min\{f(X_{k-1}), \exp(h_n(X_{k-1}))\}}{f(X_{k-1}) \min\{f(X), \exp(h_n(X))\}} \right\}$ alors garder $X_k = X_{k-1}$
 - sinon poser $X_k = X$

Code R pour l'algorithme ARMS

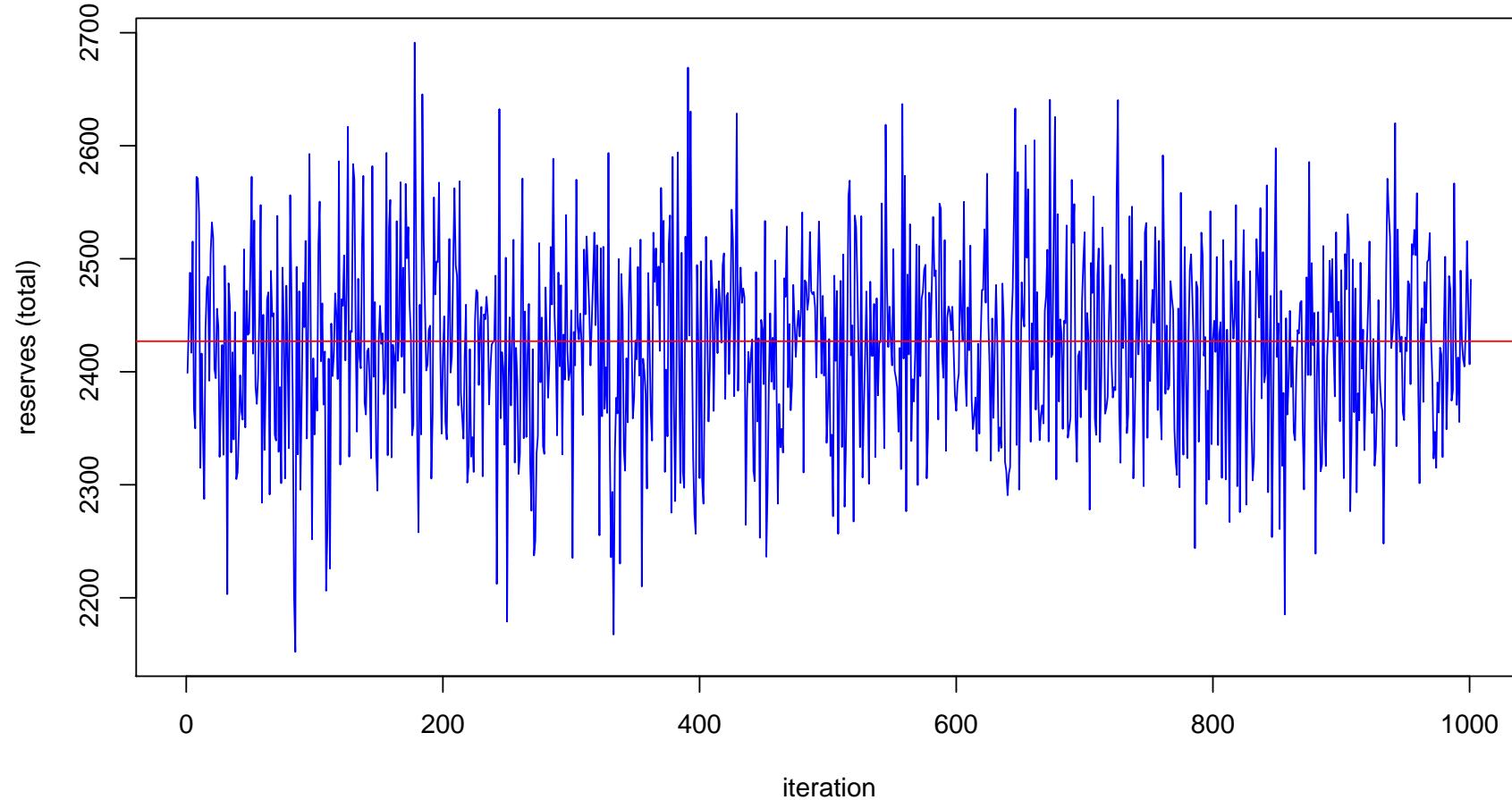
Ces fonctions exponentielles par morceaux sont intéressantes car elles sont faciles à simuler. La fonction h_n est linéaires par morceaux, avec comme noeuds N_k , de telle sorte que

$$h_n(x) = a_k x + b_k \text{ pour tout } x \in [N_k, N_{k+1}].$$

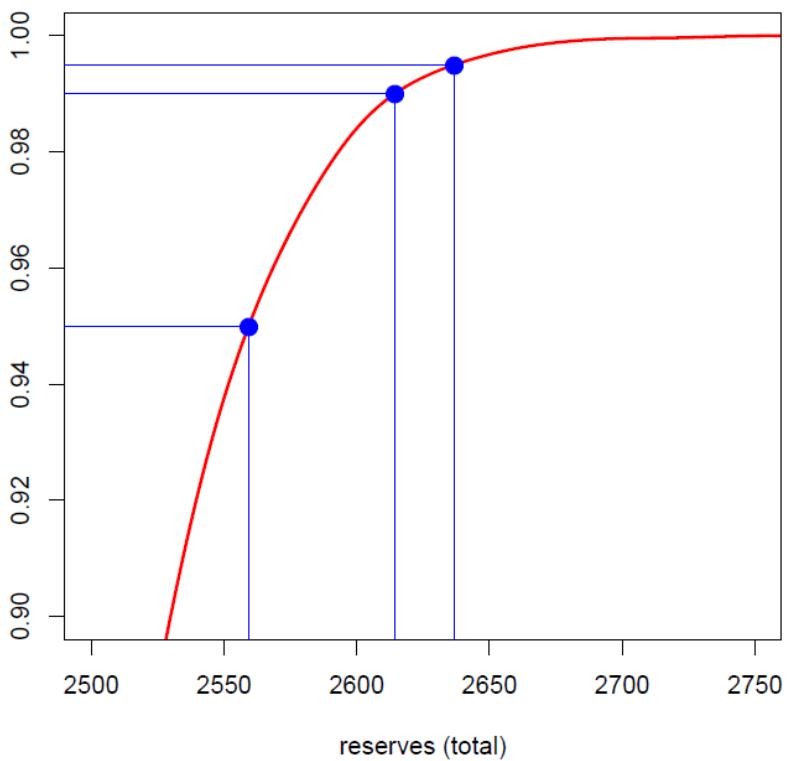
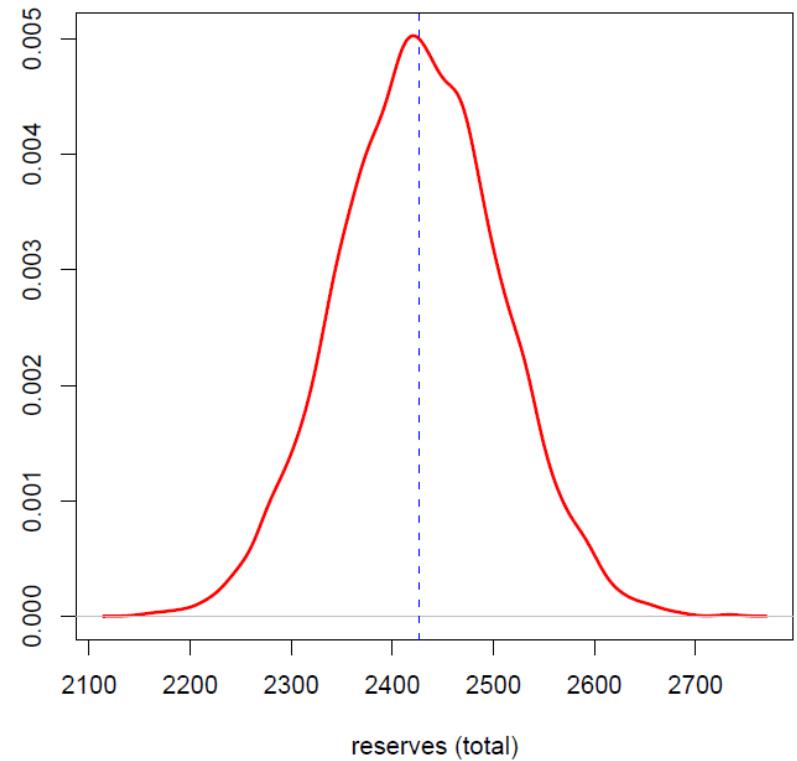
Alors $g_n(x) = \frac{\exp(h_n(x))}{I_n}$ où

$I_n = \int \exp(h_n(t))dt = \sum \frac{\exp[h_n(N_{k+1})] - \exp[h_n(N_k)]}{a_k}$. On calcule alors G_n , la fonction de répartition associée à g_n , et on fait utiliser une méthode d'inversion pour tirer suivant G_n .

Bayesian estimation for reserves



Bayesian estimation for reserves



Bayesian estimation for reserves

