Non-life (re)insurance modeling with GEMAct

Gabriele Pittarello, Manfred Marvin Marchione, Edoardo Luini

March 9, 2022

Abstract

GEMAct is an actuarial Python package, based on the collective risk theory framework, that offers a comprehensive set of tools for insurance and reinsurance pricing, stochastic claims reserving, risk theory modeling, and risk aggregation.

1 Introduction

The GEMAct package is a software based on a collective risk model apparatus to price non-life non-proportional (re)insurance contracts [4], including individual and aggregate conditions and allowing for reinstatements [6]; and to estimate loss reserves, see [2] and [5]. Furthermore, it is possible to compute quantiles of the risks aggregate distribution, via an efficient implementation of the AEP algorithm, [1]; to implement multi-annual ruin theory analysis, and to perform profitability tests over the selected time-horizon. The variety of available functionalities makes GEMAct modeling very flexible and provides actuarial scientists and practitioners with a powerful tool that fits into the expanding community of Python programming language. GEMAct aims at increasing the base Python tools available in SciPy and NumPy for acturial scientists, see [3] and [7]. For instance, GEMAct contribution to the Python community is also about adding the distributions belonging to the (a,b,k) class, see [4] and table 1. GEMAct provides the first Python implementation of average cost method for claims reserving such as the Fisher-Lange, see [2]. Based on the Fisher-Lange results it is possible to fit the collective risk model for claims reserving, see [5].

2 (Re)insurance pricing

2.1 Severity discretization with GEMAct

Given a continuous severity distribution with c.d.f. $F_z(\cdot)$ an approximate discrete distribution can be constructed [4]. In order to discretize the severity distribution, a span h and an integer m are chosen and a probability f_j is assigned to each point $j \cdot h$, $j = 0, \ldots, m$. Two different methods for determining the probabilities fj are implemented in GEMAct:

1. Mass dispersal

$$f_0 = F\left(\frac{h}{2}\right)$$

$$f_j = F_z\left(j \cdot h + \frac{h}{2}\right) - F_z\left(j \cdot h - \frac{h}{2}\right) \; j = 0, \ldots, m$$

2. Local moment matching

$$f_0 = 1 - \frac{E[Z \wedge h]}{h}$$

$$f_j = \frac{2E[Z \wedge (i \cdot h)] - E[Z \wedge ((i-1) \cdot h)] - E[Z \wedge (i+1) - h)]}{h} \; j = 1, \ldots, m$$

Examples of severity discretization are provided in appendix A.

2.2 (Re)insurance pricing

The GemAct package can be used for pricing the following reinsurance treaties and their combinations.

1. Excess-of-Loss (XL).
2. Reinstatements.
3. Quota-share (QS).
4. Deductibles.
5. Stop-loss (SL).

The computational methods offered by the GemAct package for the collective risk model computation are either simulative or non-simulative.

1. Recursive method (exact computation).
2. Discrete Fourier transform, via the Fast Fourier Transform (FFT) algorithm.
3. Monte Carlo simulation.

An example of (re)insurance pricing is provided in appendix B.

2.3 More distributions available to practitioners

A complete view on the discrete and continuous distribution supported is available on the GEMAct website. See [4] for an extensive description of the (a,b,k) distributions. In table 1 new distributions not yet supported in SciPy added by GEMAct.

Distribution	GEMAct	SciPy
Zero-Modified Binomial.	v	x
Zero-Modified Geometric.	v	x
Zero-Modified Logarithmic.	v	x
Zero-Modified Negative Binomial.	v	x
Zero-Modified Poisson.	v	x
Zero-Truncated Binomial.	v	x
Zero-Truncated Geometric.	v	x
Zero-Truncated Negative Binomial.	v	x
Zero-Truncated Poisson.	v	x

Table 1: A list of discrete distributions for actuarial modeling GEMAct with respect to SciPy.

3 Average cost methods for loss reserves

Average cost methods predict separately frequency and severity amounts in the lower triangle.

$$X_{i,j} = n_{i,j} \cdot m_{i,j} \tag{1}$$

Denote the incremental payments $X_{i,j}$ in each cell as the product between the number of claims in the i, j, $n_{i,j}$ cell and the severity average cost $m_{i,j}$, see equation 1. The code snippet in appendix D shows how to use the GEMAct LossReserve class to compute the reserve according to the Fisher-Lange method. GEMAct provides several methods to check the estimates consistency both numerically and visually; and sample data to test reserving on a sand-box. In figure 2 the output of the method LossReserve.SSPlot. Observe that it is possible to pass as an argument an integer that specify the accident year from which the settlement speed plot should begin. Figure 2 show the settlement speed for accident years greater or equal to 7.

4 The AEP algorithm

Assume to be interested in:

$$\mathbb{P}\left[X_1 + \cdots + X_d \leq s\right] \tag{2}$$

Assuming:

- $X = (X_1, ..., X_d)$ vector of strictly positive random components.
- The joint c.d.f. $H(x_1, \ldots, x_d) = P\left[X_1 \leq x_1, \ldots, X_d \leq x_d\right]$ is known or it can be computed numerically.

```
       time            ultimate FL             reserve FL
=======================================================
        0.0                86841.0                 1068.0
        1.0       98161.954237115     1837.9542371150096
        2.0    107140.16750863047     2133.1675086304704
        3.0    117422.58617931853      2747.586179318526
        4.0    122773.45583108162      4485.455831081627
        5.0    138139.75521426514      6000.755214265135
        6.0    148047.83900370973      9756.839003709756
        7.0    145115.34662015972     15214.346620159751
        8.0     146550.9374895417     21764.937489541713
        9.0     155700.0816042082     33900.081604208186
       10.0    164364.24141488288     52022.241414882876
       11.0    164403.34082380703    104042.34082380703

 FL reserve:   254973.7059267201
```

Figure 1: Output of the LossReserve.claimsreserving() method.

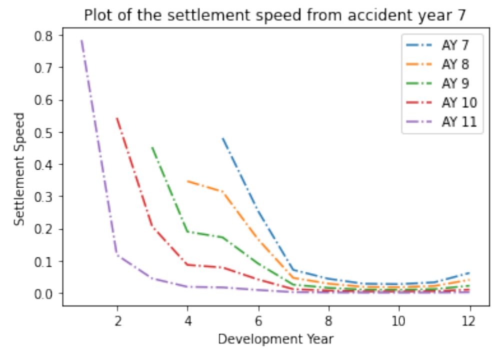

Figure 2: Output of the LossReserve.SSPlot() method.

Input description	GEMAct parameter	Parameter type
s in equation 2	s_{la}	float
Number of iterations, n in equation 3	n_{la}	int
First marginal	m1	str
First marginal parameters	m1par	dict
Second marginal	m2	str
Second marginal parameters	m2par	dict
Copula	cop	str
Copula parameters	coppar	dict

Table 2: Inputs required to compute the 2-d AEP algorithm.

The probability in 2 can be computed iteratively via the AEP algorithm, which is implemented for the first time in Python in the GEMAct package. See [1].

$$\mathbb{P}\left[X_1 + \cdots + X_d \leq s\right] \approx P_n(s) \tag{3}$$

The inputs required for the computation of the 2-d AEP algorithm are provided in table 2.

5 Conclusion

GEMAct is a comprehensive package that can satisfy both practitioners and academics needs. Further adds to our package will make our functionalities even more flexible. For instance, the LossReserve class will include more methods to estimate the claims reserving under the Solvency II; and the current AEP code can be fasten with a C implementation.

A Code: pricing an excess of loss

```
import numpy as np

m_ = 100000 #int
h_ = 1. # float
d_ = 200 # float
u_ = 5000 # float
spar_ = {'a': 2000, 'scale':1/2} # dict
severity_ = 'gamma' # str

mysv = gemact.Severity(spar=spar_,d=d_,u=u_,sdist=severity_,m=m_,h=h_)
massD = mysv.massDispersal()
localM = mysv.localMoments()

meanMD = np.sum(massD['severity_seq']*massD['fj'])
meanLM = np.sum(localM['severity_seq']*localM['fj'])

print(meanMD)
#800.0
print(meanLM)
#800.0
```

B Code: loss model

B.1 Aggregate cost computation

```
frequency_ = 'poisson' #str
spar = {'a': 6, 'scale': 1/3} #dict
severity_ = 'gamma' #str
d_ = .2 #float
u_ = 5. #float
nsim = 10000 #int
fpar = {'mu':4} #dict

lm_mc = gemact.LossModel(
                method='mcsimulation',
                fpar=fpar,
                fdist=frequency_,
                spar=spar,
                sdist=severity_,
                nsim=int(nsim),
                d=d_,
```

```
                    u=u_,
                    setseed=1
)

lm_rec = gemact.LossModel(
                    method='recursive',
                    fpar=fpar,
                    fdist=frequency_,
                    spar=spar,
                    sdist=severity_,
                    d=d_,
                    u=u_,
                    m=int(1e+03),
                    h=1,
                    n=int(1e+05)
)

lm_fft = gemact.LossModel(
                    method='fft',
                    fpar=fpar,
                    fdist=frequency_,
                    spar=spar,
                    sdist=severity_,
                    d=d_,
                    u=u_,
                    m=int(1e+03),
                    h=1,
                    n=int(1e+05)
)

print('MC', lm_mc.empiricalmoments())
#MC 7.161077556659078

print('RECURSIVE', lm_rec.empiricalmoments())
#RECURSIVE 7.19283615308811

print('FFT', lm_fft.empiricalmoments())
#FFT 7.192555408630492
```

B.2 Pricing an excess of loss

```
lm = gemact.LossModel(
               method='fft',
```

```
            fpar={'mu':.5},
            fdist= 'poisson',
            spar={'loc':0,'scale': 83, 'c': 0.83},
            sdist='genpareto',
            u=100,
            discretizationmethod='massdispersal',
            nsim=1e+05,
            m=int(10000),
            h=.01,
            n=int(100000)
)

lm.Pricing()
```

C Code: an application of the Fisher-Lange

```
# Fisher-Lange data
ip_ = gemact.gemdata.IPtriangle # numpy.ndarray
in_ = gemact.gemdata.in_triangle # numpy.ndarray
cp_ = gemact.gemdata.cased_amount_triangle # numpy.ndarray
cn_ = gemact.gemdata.cased_number_triangle # numpy.ndarray
reported_ = gemact.gemdata.reported_ #numpy.ndarray
infl_ = gemact.gemdata.claims_inflation # numpy.ndarray
rm_ = 'fisherlange' # str
tail_ = True #bool

lr = gemact.LossReserve(
              tail=tail_,
              incremental_payments=ip_,
              cased_payments=cp_,
              cased_number=cn_,
              reported_claims=reported_,
              incurred_number=in_,
              reserving_method=rm_,
              claims_inflation=infl_
)

lr.claimsreserving()
```

D Code: 2-d AEP

```
la = gemact.LossAggregation(
                   s_la=1,
```

```
                        n_la=7,
                        m1='genpareto',
                        m1par={'loc':0, 'scale':1/.9, 'c':1/.9},
                        m2='genpareto',
                        m2par={'loc':0, 'scale':1/1.8, 'c':1/1.8},
                        copdist='gumbel',
                        coppar={'par': 1.2}
)

print(la.out)
```

References

[1] Philipp Arbenz, Paul Embrechts, and Giovanni Puccetti. The aep algorithm for the fast computation of the distribution of the sum of dependent random variables. *Bernoulli*, 17:562–591, 2011.

[2] Wayne H. Fisher, Jeffrey T. Lange, and John Balcarek. Loss reserve testing: a report year approach. 1999.

[3] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[4] Stuart A. Klugman, Harry H. Panjer, and Gordon E. Willmot. *Loss Models: From Data to Decisions*. 1998.

[5] Alessandro Ricotta and Gian Paolo Clemente. An extension of collective risk model for stochastic claim reserving. 2016.

[6] Bjørn Sundt. On excess of loss reinsurance with reinstatements. *Insurance Mathematics & Economics*, 12:73, 1993.

[7] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0

Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.