GEMAct: a comprehensive actuarial package for non-life (re)insurance

Gabriele Pittarello  
Sapienza, Università di Roma  
gabriele.pittarello@uniroma1.it

Manfred Marvin Marchione  
Sapienza, Università di Roma  
manfredmarvin.marchione@uniroma1.it

Edoardo Luini, Ph.D.  
Credit Suisse  
edoardo.luini@uniroma1.it

Table of contents

Introduction
- Why Python?
- Usage

Loss Model
- Discrete severity
- Discrete severity with GEMAct
- Aggregate loss – Panjer recursion
- Distributions
- Aggregate loss - FFT (1/2)
- Aggregate loss - FFT (2/2)
- Aggregate loss with GEMAct
- Pricing models

Loss Reserve
- Loss reserves (1/2)
- Loss reserves (2/2)
- Fisher-Lange reserve with GEMAct

Risk aggregation
- Loss aggregation - (1/2)
- Loss aggregation - (2/2)
- The AEP algorithm in GEMAct
Why Python?

<table>
<thead>
<tr>
<th>Programming language</th>
<th>Tiobe ranking</th>
<th>PyPi</th>
<th>Stackoverflow</th>
<th>KDNuggles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>R</td>
<td>13</td>
<td>7</td>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
<td>4</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>Java</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 1: Programming language popularity according to different websites rating, February 2022.

Usage

1. pip install gemact

Listing 1: how to install GEMAct from the PyPi index.

Figure 1: GEMAct in the PyPi index.
Discrete severity

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 $j \cdot h$, $j = 0, ..., m$.

Two different methods for determining the probabilities $f_j$ are implemented in GEMAct:

1) Mass dispersal method

$$f_0 = F \left( \frac{h}{2} \right), \quad f_j = F_Z \left( j \cdot h + \frac{h}{2} \right) - F_Z \left( j \cdot h - \frac{h}{2} \right), \quad j = 1, ..., m$$

2) Local moment matching

$$f_0 = 1 - \frac{E[Z \wedge h]}{h}, \quad f_j = \frac{2 E[Z \wedge (i \cdot h)] - E[Z \wedge ((i-1) \cdot h)] - E[Z \wedge ((i+1) \cdot h)]}{h}, \quad j = 1, ..., m.$$ 

Discrete severity with GEMAct

```python
1. import numpy as np
2. 3. m_= 100000 #int
4. h_= 1. # float
5. d_= 200 # float
6. u_= 5000 # float
7. spar_ = {'a' : 2000, 'scale' : 1/2} # dict
8. severity_ = 'gamma' # str
9.
10. Nysv = gemact.Severity(
11.     spar=spar_,
12.     d=d_,
13.     u=u_,
14.     sdist=severity_,
15.     m=m_,
16.     h=h_,
17. )
18. massD = nysv.massDispersal()
19. localM = nysv.localMoments()
20. 21. meanMD = np.sum(massD['severity_seq']*massD['fj'])
22. meanLM = np.sum(localM['severity_seq']*localM['fj'])
23. 24. print(meanMD)
25. # 800.0
26. print(meanLM)
27. # 800.0
```

Listing 2: Severity discretization with GEMAct.
Aggregate loss – Panjer recursion

In the C.R.M. framework, assume that the frequency distribution belongs to the \((a, b, 0)\) class, that is \(p_k = \left(\frac{a + \frac{b}{k}}{k}\right) \cdot p_{k-1}\) \(k = 1, 2, \ldots, a, b \in \mathbb{R}\).

where \(p_k = P(N = k)\).

If the severity \(Z\) has an arithmetic distribution with \(f_j = P(Z = jh), j = 0,1,\ldots,\) then the following recursive formula holds:

\[
g_k = \frac{1}{1 - af_0} \sum_{j=1}^{k} \left(\frac{a + bj}{k}\right) f_j \cdot g_{k-j}
\]

where \(g_k = P(\sum_{j=1}^{N} Z_j = k \cdot h)\).

Distributions

<table>
<thead>
<tr>
<th>Distribution</th>
<th>GEMAct</th>
<th>SciPy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero-Truncated Poisson.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Modified Poisson.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Truncated Binomial.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Modified Binomial.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Truncated Geometric.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Modified Geometric.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Truncated Negative Binomial.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Modified Negative Binomial.</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>Zero-Modified Logarithmic.</td>
<td>✓</td>
<td>×</td>
</tr>
</tbody>
</table>

Table 2: GEMAct enhances the number distributions available to Python users. See [2].
Given the probabilities \( f = (f_0, ... , f_m) \) of the arithmetic severity distribution, denote by \( \hat{f} = (f_0, ... , f_m) \) the Discrete Fourier Transform of \( f \).

The probabilities \( g = (g_0, ... , g_m) \) of the aggregate loss can be approximated by the inverse DFT of

\[
\hat{g} := P_N(\hat{f})
\]

where \( P_N(\cdot) \) is the probability generating function of the frequency \( N \).

The distribution of the aggregate loss can be computed efficiently by means of a Fast Fourier Transform algorithm.

**Note**: the product of DFTs is the DFT of the circular convolution \([H] \Rightarrow \) the procedure leads to aliasing error.

---

A tilting procedure can be used to reduce the aliasing error [7]:

1. Set \( f = (f_0, ... , f_m) \)

2. Tilt the sequence: \( f \rightarrow E_\theta f := (e^{-\theta j}f_j)_{j=0,...,m} \) for a suitable \( \theta > 0 \)

3. Calculate the DFT of the tilted sequence: \( E_\theta \hat{f} \)

4. Calculate the inverse DFT of \( P_N(E_\theta \hat{f}) \).

5. Untilt the obtained sequence by applying \( E_{-\theta} \).
Aggregate loss with GEMAct

1. `frequency_ = 'poisson'` #str
2. `spar = {'a':6, 'scale':1/3}` #dict
3. `severity_ = 'gamma'` #str
4. `d_ = .2` #float
5. `u_ = 5.` #float
6. `Fpar = {'mu':4}` #dict
7. 
8. `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)
)

9. print('RECURSIVE',lm_rec.empiricalmoments())
10. RECURSIVE 7.19283615308811

Listing 3: aggregate loss compute via recursive formula.

---

Pricing models

The notation for the pure reinsurance premium is \( P = E(X) \).

\[ X' = \min(\max(0,X - L), (K + 1) \cdot m) \]

Given \( X = \sum_{i=1}^{N} Y_i \), where:

\[ Y_i = \min(\max(0,Z_i - l), m) \]

The following equation shows the reinsurance premium for excess of loss treaties from [4]. It is possible to obtain a plain-vanilla XL with \( L = 0, K = +\infty \) and \( c = 0 \).

\[
P = \frac{D_{L,K}}{1 + \frac{1}{m} \sum_{k=1}^{K} c_k d_{L,k-1}}
\]
1. `lossm=gemact.LossModel(
2.     method='fft',
3.     fpar={'mu':.5},
4.     fdist='poisson',
5.     spar=('loc':0,'scale':.0333, 'c': 0.033),
6.     sdist='genpareto', XL upper priority,
7.     u=100,
8.     discretizationmethod='massdispersal',
9.     n=int(10000),
10.    h=.01,
11.   n=int(100000),
12.  L=0,
13.   K=1,
14.  c=1)

Listing 5: Pricing with reinstatements, GEMAct implementation.

Contractual limits parameter value
==================================================================
deductible d 0.0
priority (severity) u 100.0
priority (aggregate) L 0
alpha (qs) alphaqs 1
reinstatements K 1
Pure premium P 24.97398396010884

Figure 2: Output of lossm.pricing()

Loss reserves (1/2)

GEMAct provides the first Python implementation of average cost methods for claims reserving, in which

\[ P_{ij} = n_{ij} \cdot m_{ij}, \]

It is also possible to compute the loss reserve by means of the collective risk model method in [6]:

\[ X_{ij} = \sum_{h=1}^{N_{ij}} z_{ij,h}. \]

The model assumes:

- **Frequency:** \( N_{ij} \sim \text{Poisson} \left( \bar{n}_{ij} \cdot q \right) \), where \( q \) is distributed as a Gamma, and \( E[q] = 1 \).
- **Severity:** \( E[Z_{ij}] = m_{ij} \) and \( c_{Z_{ij}} = c_{Z_{ij}} \cdot r \), where \( r \) is distributed as a Gamma and \( E[r] = 1 \).
Loss reserves (2/2)

There is a unique class to compute the loss reserve in GEMAct: **reserving_method** allows to choose whether to fit the reserve with the Fisher-Lange or the C.R.M. for claims reserving.

Triangular data should be provided as:

- `ip_tr` = (numpy.ndarray) – Incremental payments triangle
- `cp_tr` = (numpy.ndarray) – Cased payments triangle
- `in_tr` = (numpy.ndarray) – Incurred number
- `cn_tr` = (numpy.ndarray) – Cased number

Fisher-Lange reserve with GEMAct

```python
1. # Fisher-Lange data
2. ip_ = gemact.gemdata.IPtriangle # numpy.ndarray
3. in_ = gemact.gemdata.in_triangle # numpy.ndarray
4. cp_ = gemact.gemdata.cased_amount_triangle # numpy.ndarray
5. cn_ = gemact.gemdata.cased_number_triangle # numpy.ndarray
6. reported_ = gemact.gemdata.reported_ # numpy.ndarray
7. infl_ = gemact.gemdata.claims_inflation # numpy.ndarray
8. rm_ = 'fisherlange' # str
9. tail_ = True # bool
10.
11. 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_
)
12.
13. lr.claimsreserving()
```

GEMAct provides users with data to test the LossReserve class. Data are then passed through LossReserve. The self.claimsreserving() method allows to print out the computation.

Listing 6: loss reserve computed with the Fisher-Lange method.
<table>
<thead>
<tr>
<th>time</th>
<th>ultimate FL</th>
<th>reserve FL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>86841.0</td>
<td>1068.0</td>
</tr>
<tr>
<td>1.0</td>
<td>98161.954237115</td>
<td>1837.954237115</td>
</tr>
<tr>
<td>2.0</td>
<td>107140.16750863047</td>
<td>2133.16750863047</td>
</tr>
<tr>
<td>3.0</td>
<td>117422.58617931853</td>
<td>2747.586179318526</td>
</tr>
<tr>
<td>4.0</td>
<td>122773.45583108162</td>
<td>4485.455831081627</td>
</tr>
<tr>
<td>5.0</td>
<td>138139.75521426514</td>
<td>6000.755214265135</td>
</tr>
<tr>
<td>6.0</td>
<td>148047.83900370973</td>
<td>9756.839003709756</td>
</tr>
<tr>
<td>7.0</td>
<td>145115.34662015972</td>
<td>15214.346620159751</td>
</tr>
<tr>
<td>8.0</td>
<td>146550.34662015972</td>
<td>21764.346620159751</td>
</tr>
<tr>
<td>9.0</td>
<td>155700.08160420828</td>
<td>33900.081604208186</td>
</tr>
<tr>
<td>10.0</td>
<td>164364.24141488288</td>
<td>52022.241414882876</td>
</tr>
<tr>
<td>11.0</td>
<td>164403.34082380703</td>
<td>104042.34082380703</td>
</tr>
</tbody>
</table>

FL reserve: 254973.706

Figure 3: gemact.LossReserve has plenty of methods implemented to understand the estimates consistency, either visually or numerically. On the left-hand side the results of gemact.LossReserve.claimsreserving(). On the right-hand-side the output of gemact.LossReserve.SSPlot(start_=7).

Loss aggregation - (1/2)

The following probability can be computed iteratively via the AEP algorithm, which is implemented for the first time in Python in the GEMAct package:

\[ P[X_1 + \cdots + X_d \leq s] \approx P_{R_1}(s) \]

Assuming:

- \( X = (X_1, \ldots, X_d) \) vector of strictly positive random components.

- The joint c.d.f. \( H(x_1, \ldots, x_d) = P[X_1 \leq x_1, \ldots, X_d \leq x_d] \) is known or it can be computed numerically.
Loss aggregation - (2/2)

\[ P[X_1 + \cdots + X_d \leq s] \approx P_n(s) \]

The AEP algorithm (2-dimensional case)

Define:

\[ V_H[Q] = \int_Q dH \approx P[X_1 + X_2 \leq s] \]

Figure 3: Picture from [1]. Geometrical interpretation of the 2-dimensional AEP first three steps. Blue parameters were added for sake of simplicity.

The AEP algorithm in GEMAct

```python
1. la=gemact.LossAggregation(
2.     s_la=1, The value of s \( P[X_1 + \cdots + X_d \leq s] \)
3.     n_la=7, Number of iterations: n in \( P_n(s) \)
4.     m1='genpareto',
5.     m1par={'loc':0,'scale':1/9,'c':1/9},
6.     m2='genpareto',
7.     m2par={'loc':0,'scale':1/1.8,'c':1/1.8},
8.     copdist='gumbel',
9.     coppar={'par':1.2}) Copula to model the joint c.d.f.
10.)
11.      The resulting probability is stored in the attribute out.
12.
13.print(la.out)
14.
15.#0.28329979582555087
```

Listing 7: AEP algorithm application.
References


