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

GEMAct
Actuarial Science in Python

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?

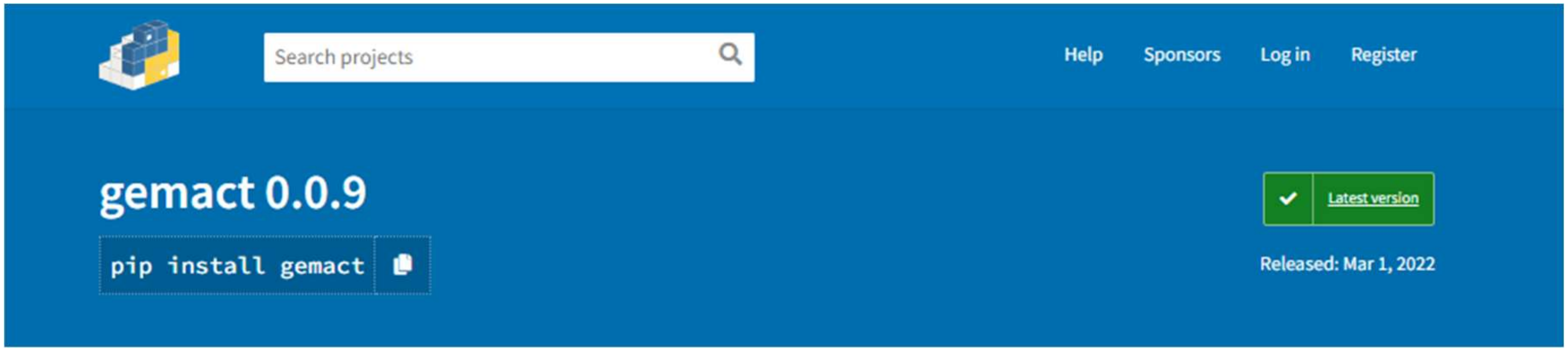
Programming language	Tiobe ranking	PyPl	Stackoverflow	KDNuggets
Python	1	1	3	1
R	13	7	21	2
C	2	4	12	6
Java	3	2	5	4

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

Usage

```
pip install gemact
```

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

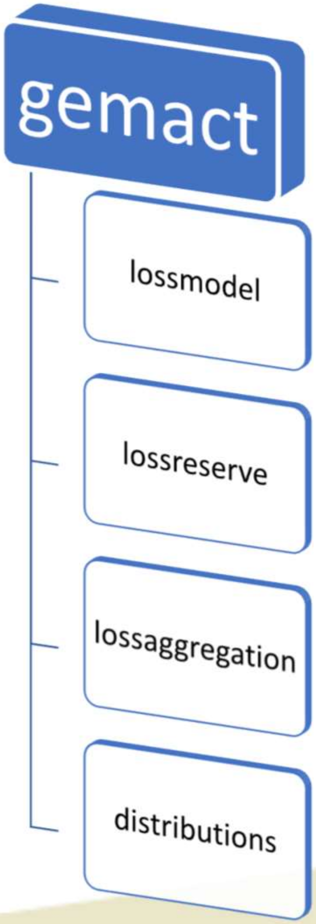


gemact 0.0.9

```
pip install gemact
```

Released: Mar 1, 2022

Figure 1: GEMAct in the PyPi index.



gemact

lossmodel

lossreserve

lossaggregation

distributions

Discrete severity

Given a continuous severity distribution with c.d.f. $F_Z(\cdot)$, an approximate discrete distribution can be constructed [2].

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

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

2) Local moment matching

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

Discrete severity with GEMAct

```
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
```

Listing 2 : Severity discretization with GEMAct.

Severity assumptions

Data are then passed through Severity.

Use the results to compute the mean of the distribution.

Aggregate loss – Panjer recursion

In the C.R.M. framework, assume that the frequency distribution belongs to the $(a, b, 0)$ class, that is

$$p_k = \left(a + \frac{b}{k}\right) \cdot p_{k-1} \qquad k = 1, 2, \dots, \qquad a, b \in \mathbb{R}$$

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

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

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

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

Distributions

Distribution	GEMAct	SciPy
Zero-Truncated Poisson.	√	×
Zero-Modified Poisson.	√	×
Zero-Truncated Binomial.	√	×
Zero-Modified Binomial.	√	×
Zero-Truncated Geometric.	√	×
Zero-Modified Gemetric.	√	×
Zero-Truncated Negative Binomial.	√	×
Zero-Modified Negative Binomial.	√	×
Zero-Modified Logarithmic.	√	×

Table 2: GEMAct enhances the number distributions available to Python users. See [2].

Aggregate loss - FFT (1/2)

Given the probabilities $f = (f_0, \dots, f_m)$ of the arithmetic severity distribution, denote by $\hat{f} = (f_0, \dots, f_m)$ the Discrete Fourier Transform of f.

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

$$\widehat{g} := P_N(\hat{f})$$

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

The ditribution 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] ⇒ the procedure leads to aliasing error.

Aggregate loss - FFT (2/2)

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

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

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

3. Calculate the DFT of the tilted sequence: $\widehat{E_\theta f}$

4. Calculate the inverse DFT of $\mathrm{P}_\mathrm{N}(\widehat{E_\theta f})$.

5. Untilt the obtained sequence by applying $E_{-\theta}$.

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 $c = 0$.

$$P = \frac{D_{LK}}{1 + \frac{1}{m}\sum_{k=1}^{K} c_k d_{L,k-1}}$$

Aggregate loss with GEMAct

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

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)
)

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

Listing 3: aggregate loss compute via recursive formula.

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

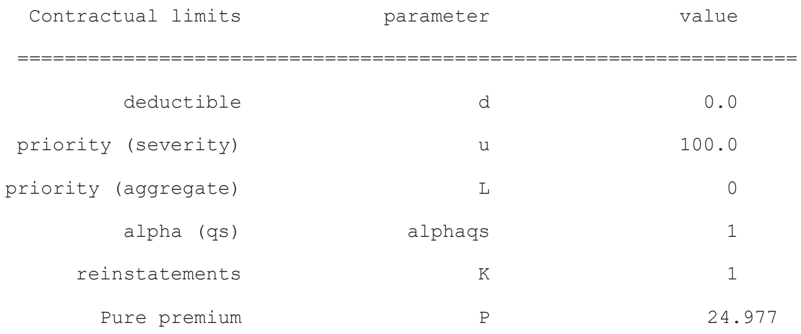
Listing 4: aggregate loss compute via Fast Fourier Transform.

```
lossm=gemact.LossModel(
             method='fft',
             fpar={'mu':.5},
             fdist= 'poisson',
             spar={'loc':0,'scale': 83.33, 'c’: 0.833},
             sdist='genpareto',
             u=100,
             discretizationmethod='massdispersal',
             m=int(10000),
             h=.01,
             n=int(100000),
             L=0,
             K=1,
             c=1)
```

XL upper priority.

Aggregate conditions.

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                 24.977
```

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

$$\mathrm{P}_{\mathrm{i,j}} = n_{i,j} \cdot m_{i,j}.$$

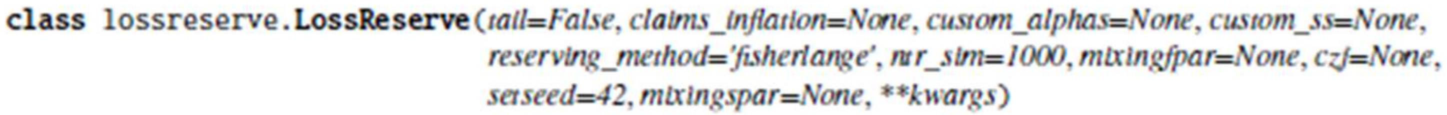
It is also possible to compute the loss reserve by means of the **collective risk model** method in [6]:

$$\mathrm{X}_{\mathrm{i,j}} = \sum_{\mathrm{h=1}}^{\mathrm{N_{i,j}}} \mathrm{Z}_{\mathrm{i,j,h}}\,.$$

The model assumes:

- **Frequency**: $N_{i,j} \sim \text{ Poisson } \left(\widehat{n_{i,j}} \cdot q\right)$, where q is distributed as a Gamma, and $\mathrm{E}[\mathrm{q}] = 1.$
- **Severity**: $E\left[Z_{i,j}\right] = m_{i,j}$ and $c_{Z_{i,j}} = \widehat{c_{Z_j}} \cdot r$, where r is distributed as a Gamma and $E[r] = 1$

Loss reserves (2/2)



```
class lossreserve.LossReserve(tail=False, claims_inflation=None, custom_alphas=None, custom_ss=None,
                              reserving_method='fisherlange', ntr_sim=1000, mixingfpar=None, czj=None,
                              setseed=42, mixingspar=None, **kwargs)
```

There is a unique class to compute the loss reserve in GEMAct: **reserving_method** allows to choose wheter 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

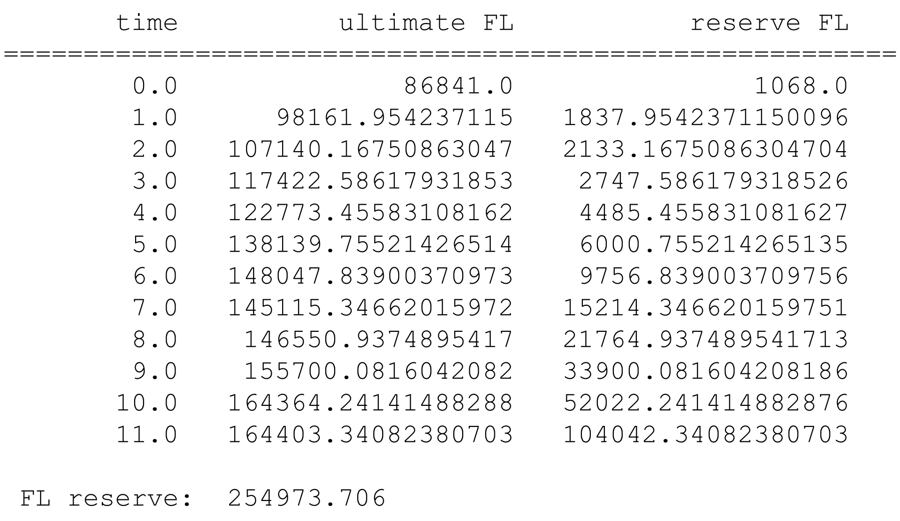
```
# Fisher-Lange data
ip_= gemact.gemdata.IPtriangle # numpy.ndarray
in_= gemact.gemdata.in_triangle # numpy.ndarray
cp_= gemact.gemdata.cased_amount_triangle # numpy.ndarray
cn_= gemact.gemdata.cased_number_triangle # numpy.ndarray
reported_ = gemact.gemdata.reported_ #numpy.ndarray
infl_ = gemact.gemdata.claims_inflation # 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()
```

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.



```
      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:  254973.706
```

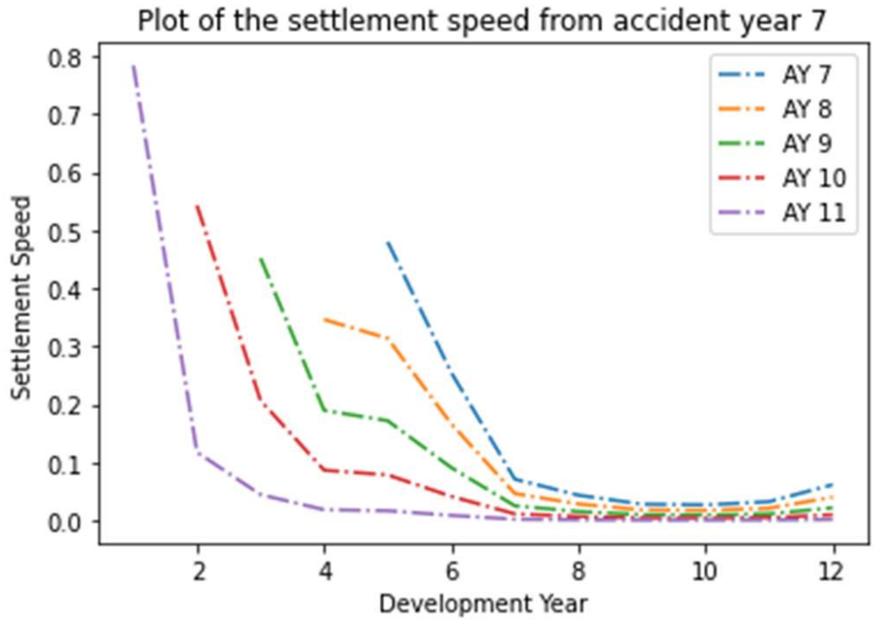


Figure 3: gemact.LossReserve has plenty of methods implemented to understand the estimates consistency, either visually or numerically.

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:

$$P[X_1 + \cdots + X_d \leq s] \approx P_n(\mathrm{s})$$

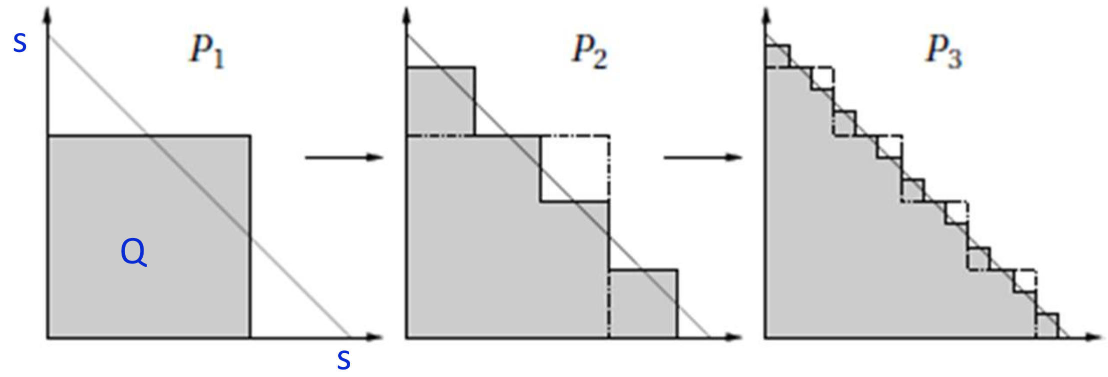
Assuming:

- $X = (X_1, \ldots, X_d)$ vector of strictly positive random components.
- The joint c.d.f. $\mathrm{H}(\mathrm{x}_1, \ldots, \mathrm{x}_\mathrm{d}) = \mathrm{P}[\mathrm{X}_1 \leq \mathrm{x}_1, \ldots, \mathrm{X}_\mathrm{d} \leq \mathrm{x}_\mathrm{d}]$ is known or it can be computed numerically.

Loss aggregation - (2/2)

$P[X_1 + \cdots + X_d \leq s] \approx P_n(\mathrm{s})$

The AEP algorithm (2-dimensional case)

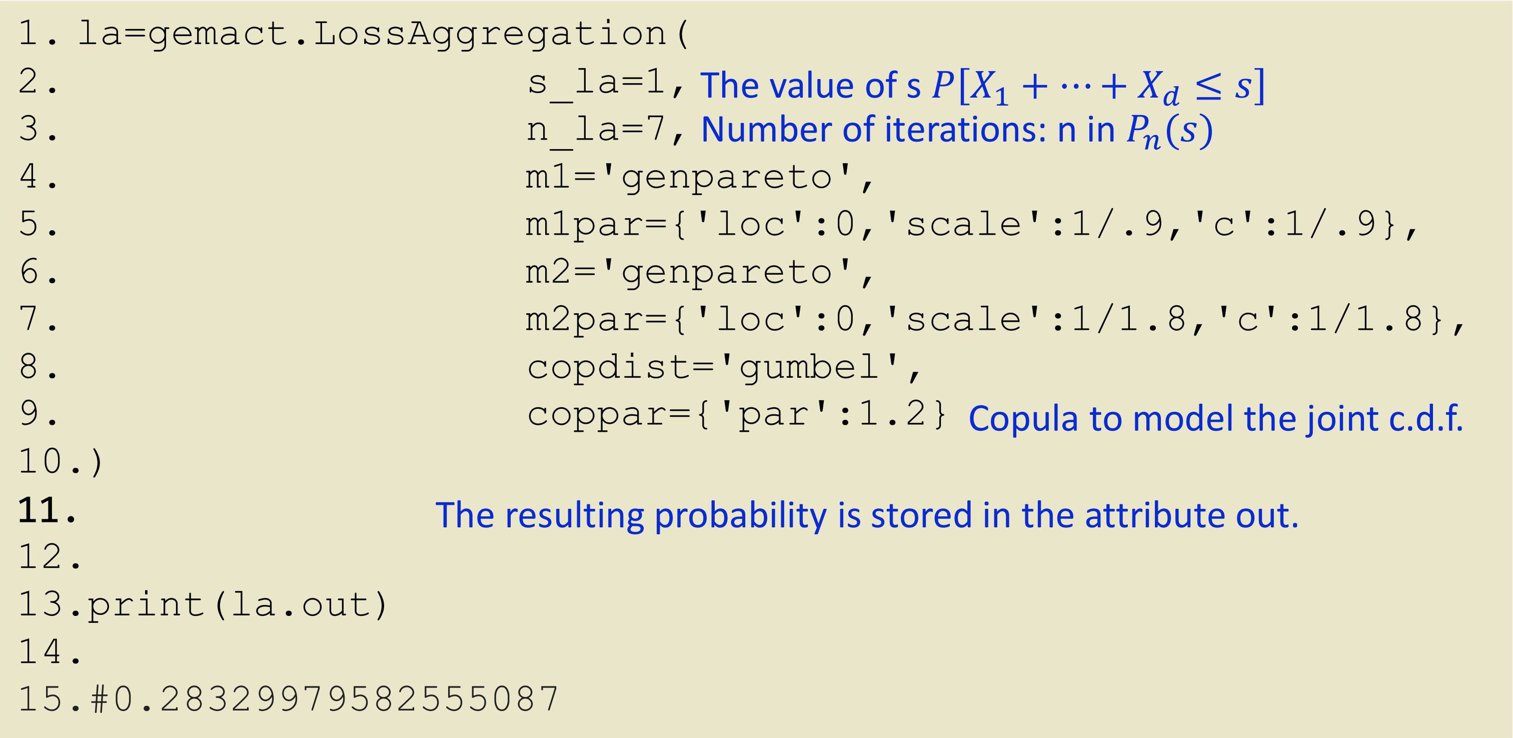


Define:

$$V_H[Q] = \int_Q d\,H \approx P[X_1 + X_2 \leq s]$$

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

The AEP algorithm in GEMAct



```
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)

#0.2832979582555087
```

The value of s $P[X_1 + \cdots + X_d \leq s]$

Number of iterations: n in $P_n(s)$

Copula to model the joint c.d.f.

The resulting probability is stored in the attribute out.

m stands for marginals. Parametersvare provided as
dictionaries (GEMAct standard).

Listing 7: AEP algorithm application.

References

1. Arbenz, Philipp, Paul Embrechts and Giovanni Puccetti. “The AEP algorithm for the fast computation of the distribution of the sum of dependent random variables.” Bernoulli 17 (2011): 562-591.

2. S.A. Klugman, H.H. Panjer and G.E. Willmot (1998): Loss Models: From Data to Decisions. Wiley, New York.

3. Fisher, Wayne H., Jeffrey T. Lange and John Balcarek. “Loss reserve testing: a report year approach.” (1999).

4. Sundt, Bjørn. “On excess of loss reinsurance with reinstatements.” Insurance Mathematics & Economics 12 (1993): 73.

5. Arbenz, Philipp, Paul Embrechts and Giovanni Puccetti. “The GAEP algorithm for the fast computation of the distribution of a function of dependent random variables.” Stochastics 84 (2012): 569 - 597.

6. Ricotta, Alessandro and Gian Paolo Clemente. “An Extension of Collective Risk Model for Stochastic Claim Reserving .”(2016).

7. Embrechts, Paul and Marco Frei. “Panjer recursion versus FFT for compound distributions.” Mathematical Methods of Operations Research 69 (2009): 497-508.