

Earthquake Catastrophe Insurance

Michelle Intan Handa 6701013556

Import Data

```
In[1]:= dataClean = Import["C:\\\\Users\\\\ASUS\\\\Documents\\\\Wolfram\\\\data_cleaned.xlsx"];
```

```
In[2]:= cleanDataset = Dataset[dataClean]
```

```
Out[2]=
```

Origin Time	Latitude	Longitude	Depth	Magnitude
2022/01/01	24.8	125.0	28.0	4.0
2022/01/02	34.1	135.1	9.0	3.7
2022/01/02	35.7	140.6	50.0	3.8
2022/01/02	37.5	137.2	13.0	3.6
2022/01/03	38.8	142.0	46.0	3.7
2022/01/03	23.9	122.2	27.0	6.3
2022/01/04	37.5	137.2	13.0	3.5
2022/01/04	37.6	141.6	55.0	4.2
2022/01/05	36.1	140.0	49.0	3.8
2022/01/07	27.4	128.5	49.0	4.3
2022/01/07	33.9	135.4	52.0	3.8
2022/01/08	27.4	128.6	43.0	3.8
2022/01/10	28.3	129.3	15.0	3.8
2022/01/11	32.9	131.9	10.0	3.9
2022/01/11	38.8	142.1	44.0	3.9
2022/01/11	36.2	140.6	82.0	3.7
2022/01/13	31.0	131.4	30.0	4.6
2022/01/13	30.4	131.0	32.0	4.4
2022/01/14	37.5	137.2	13.0	3.6

988 total

Magnitudo

```
In[6]:= magnitudo = cleanDataset[All, 2 ;, {5}]
```

```
Out[6]=
```

4.0
3.7
3.8
3.6
3.7
6.3
3.5
4.2
3.8
4.3
3.8
3.8
3.8
3.9
3.9
3.7
4.6
4.4
3.6
3.5
987 total >

Depth

```
In[6]:= depth = cleanDataset[All, 2 ;;, {4}]
```

```
Out[6]=
```

28.0
9.0
50.0
13.0
46.0
27.0
13.0
55.0
49.0
49.0
52.0
43.0
15.0
10.0
44.0
82.0
30.0
32.0
13.0
58.0
987 total >

Date

```
In[6]:= date = cleanDataset[All, 2 ;;, {1}]
```

Out[6]=

2022/01/01
2022/01/02
2022/01/02
2022/01/02
2022/01/03
2022/01/03
2022/01/04
2022/01/04
2022/01/05
2022/01/07
2022/01/07
2022/01/08
2022/01/10
2022/01/11
2022/01/11
2022/01/11
2022/01/13
2022/01/13
2022/01/14
2022/01/14
987 total >

Analyze Magnitude and Its Frequency

```
In[7]:= cumMagnitudo = Sort[DeleteDuplicates[Flatten@magnitudo]];
```

```
In[8]:= Normal@cumMagnitudo
```

Out[8]=

```
{1.4, 2., 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 3., 3.1, 3.2, 3.3, 3.4,
3.5, 3.6, 3.7, 3.8, 3.9, 4., 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5., 5.1,
5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8, 5.9, 6., 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 7.4}
```

```
In[1]:= count = Counts[Normal@Flatten@magnitudo]
Out[1]= <| 4. → 63, 3.7 → 38, 3.8 → 54, 3.6 → 51, 6.3 → 2, 3.5 → 42, 4.2 → 52, 4.3 → 58, 3.9 → 49, 4.6 → 34,
  4.4 → 32, 2.8 → 11, 3.3 → 41, 4.1 → 63, 3.4 → 42, 4.7 → 28, 4.5 → 23, 6.6 → 3, 5.4 → 11,
  5.6 → 9, 3.2 → 21, 5.8 → 5, 5. → 21, 5.3 → 13, 3.1 → 24, 3. → 27, 4.9 → 18, 4.8 → 15, 2.5 → 7,
  5.1 → 14, 2.2 → 3, 7.4 → 1, 6.1 → 1, 5.9 → 9, 5.5 → 8, 5.2 → 12, 2.6 → 9, 2.3 → 5, 1.4 → 1,
  2.9 → 28, 2.1 → 2, 5.7 → 5, 2.7 → 12, 6. → 5, 2.4 → 6, 6.2 → 5, 6.4 → 1, 6.5 → 2, 2. → 1 |>

In[2]:= count = Flatten /@ List @@@ Normal@count
Out[2]= {{4., 63}, {3.7, 38}, {3.8, 54}, {3.6, 51}, {6.3, 2}, {3.5, 42}, {4.2, 52}, {4.3, 58},
  {3.9, 49}, {4.6, 34}, {4.4, 32}, {2.8, 11}, {3.3, 41}, {4.1, 63}, {3.4, 42}, {4.7, 28},
  {4.5, 23}, {6.6, 3}, {5.4, 11}, {5.6, 9}, {3.2, 21}, {5.8, 5}, {5., 21}, {5.3, 13},
  {3.1, 24}, {3., 27}, {4.9, 18}, {4.8, 15}, {2.5, 7}, {5.1, 14}, {2.2, 3}, {7.4, 1},
  {6.1, 1}, {5.9, 9}, {5.5, 8}, {5.2, 12}, {2.6, 9}, {2.3, 5}, {1.4, 1}, {2.9, 28},
  {2.1, 2}, {5.7, 5}, {2.7, 12}, {6., 5}, {2.4, 6}, {6.2, 5}, {6.4, 1}, {6.5, 2}, {2., 1}}
```

Find the Distribution

```
In[3]:= xValues = Flatten@count[[All, 1]]
Out[3]= {4., 3.7, 3.8, 3.6, 6.3, 3.5, 4.2, 4.3, 3.9, 4.6, 4.4, 2.8, 3.3, 4.1, 3.4, 4.7,
  4.5, 6.6, 5.4, 5.6, 3.2, 5.8, 5., 5.3, 3.1, 3., 4.9, 4.8, 2.5, 5.1, 2.2, 7.4, 6.1,
  5.9, 5.5, 5.2, 2.6, 2.3, 1.4, 2.9, 2.1, 5.7, 2.7, 6., 2.4, 6.2, 6.4, 6.5, 2.}

In[4]:= FindDistribution[xValues]
Out[4]= NormalDistribution[4.30784, 1.54993]
```

The normal distribution implies that most earthquakes occur near the mean magnitude, with fewer small or large events.

```
In[5]:= fittedDistX = EstimatedDistribution[xValues, NormalDistribution[4.30784, 1.54993]];
DistributionFitTest[xValues, fittedDistX]
Out[5]= 0.961592

In[6]:= yValues = Flatten@count[[All, 2]]
Out[6]= {63, 38, 54, 51, 2, 42, 52, 58, 49, 34, 32, 11, 41, 63, 42, 28, 23, 3, 11, 9, 21, 5, 21,
  13, 24, 27, 18, 15, 7, 14, 3, 1, 1, 9, 8, 12, 9, 5, 1, 28, 2, 5, 12, 5, 6, 5, 1, 2, 1}

In[7]:= FindDistribution[yValues]
*** FindDistribution: The data will be treated as continuous. Use the option TargetFunctions→Discrete otherwise.
Out[7]= ExponentialDistribution[0.04842]
```

The exponential distribution suggests that earthquake frequencies decrease rapidly as they

increase in size.

```
In[8]:= fittedDistY = EstimatedDistribution[yValues, ExponentialDistribution[0.04842]];
DistributionFitTest[yValues, fittedDistY]

Out[8]= 0.682269
```

Use Cramer-Lundberg Theory

$$R(t) = u + ct - \sum_{i=1}^{N(t)} S_i$$

- Initial Reserve (u): The starting capital of the insurer
- Premium Rate (c): amount collected per unit time
- Number of Claims ($N(t)$): Modeled using a Poisson process with rate $\lambda(t)$, influenced by x .
- Claim Sizes (S_i): Depends on the magnitude x .

Model the Number of Claims $N(t)$

```
In[9]:= poissonRate = 1 / 0.04842;
poissonProcess = PoissonProcess[poissonRate];
```

```
In[10]:= numEarthquakes = RandomFunction[poissonProcess, {0, 2}];
```

Model the Claim Sizes (S_i)

```
In[11]:= magnitudeDist = NormalDistribution[4.30784, 1.54993];
```

Claim sizes (S_i) depends on magnitude (x). The larger the magnitude, the larger the claim. So, the possible model is:

$$S = a \times e^{b \times x}$$

```
In[12]:= claimSizes = claimSizeFunction /. RandomVariate[magnitudeDist, 100];
```

Because there is no real data about the damage that each magnitude of earthquake produce, we suppose:

- $a = 0.01$ that represents minimal damage from very small earthquake
- $b = 0.5$ makes the function grow exponentially but not too aggressively

```
In[13]:= claimSizeFunction[magnitude_, propertyValue_] :=
  propertyValue * (0.01 * Exp^(0.5 * magnitude));
```

Expected Claim Size

$$E[S] = \int_{-\infty}^{\infty} S(x) \times f(x) dx$$

```
In[®]:= expectedClaimSize =
  NIIntegrate[claimSizeFunction[x] × PDF[magnitudeDist, x], {x, -Infinity, Infinity}];

  ••• NIIntegrate: The integrand 0.257394 e-0.208135 (-4.30784+x)2 claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. i

  ••• NIIntegrate: The integrand 0.257394 e-0.208135 (-4.30784+x)2 claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. i

In[®]:= expectedClaimSize[propertyValue_] := Module[
  {magnitudeRange, probabilities, claims, total}, magnitudeRange = Range[1.4, 7.4, 0.1];
  probabilities = PDF[magnitudeDist, #] & /@ magnitudeRange;
  claims = claimSizeFunction[#, propertyValue] & /@ magnitudeRange;
  total = Sum[claims[[i]] * probabilities[[i]], {i, 1, Length[magnitudeRange]}] * 0.1;
  total];

  ••• NIIntegrate: The integrand 0.257394 e-0.208135 (-4.30784+x)2 claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. i

  ••• SetDelayed: Tag NIIntegrate in NIIntegrate[claimSizeFunction[x] PDF[magnitudeDist, x], {x, -∞, ∞}][propertyValue_] is Protected. i
```

Expected Loss

$$E[Loss] = E[S] \times \lambda \times t$$

```
In[®]:= expectedTotalLoss = expectedClaimSize * poissonRate * t;

  ••• NIIntegrate: The integrand 0.257394 e-0.208135 (-4.30784+x)2 claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. i

  ••• NIIntegrate: The integrand 0.257394 e-0.208135 (-4.30784+x)2 claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. i
```

Calculate Premium

$$\text{Premium} = (1 + \text{safetyLoading}) \times E[\text{Loss}]$$

```
In[6]:= safetyLoading = 0.2;
premiumPrice = (1 + safetyLoading) * expectedTotalLoss;
```

... NIntegrate: The integrand $0.257394 e^{-0.208135 (-4.30784+x)^2}$ claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. [i](#)

... NIntegrate: The integrand $0.257394 e^{-0.208135 (-4.30784+x)^2}$ claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. [i](#)

... NIntegrate: The integrand $0.257394 e^{-0.208135 (-4.30784+x)^2}$ claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. [i](#)

... General: Further output of NIntegrate::inumr will be suppressed during this calculation. [i](#)

```
In[6]:= calculatePremium[propertyValue_, years_, safetyLoading_, deductible_] := Module[
{expectedLoss}, expectedLoss = expectedClaimSize[propertyValue] * poissonRate * years;
expectedLoss * (1 + safetyLoading) * (1 - deductible)];
```

Stimulate Reserve Process (R(t))

```
In[7]:= reserve[t_] := initialReserve + premiumRate * t - Total[RandomVariate[
ExponentialDistribution[expectedClaimSize], RandomVariate[poissonProcess[t]]]];
```

Assess Ruin Probability

```
In[•]:= ruinProbability = Probability[reserve[t] < 0,
{t, 0, Infinity}, Method → "MonteCarlo"];
```

••• **NIntegrate**: The integrand $0.257394 e^{-0.208135 (-4.30784+x)^2}$ claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. [i](#)

••• **RandomVariate**: Parameter 20.6526 t at position 1 in PoissonDistribution[20.6526 t] is expected to be positive.

••• **RandomVariate**: The array dimensions RandomVariate[PoissonDistribution[20.6526 t]] given in position 2 of RandomVariate[ExponentialDistribution[NIntegrate[claimSizeFunction[x] PDF[magnitudeDist, x], {x, -∞, ∞}]]], RandomVariate[PoissonDistribution[20.6526 t]]] should be a list of non-negative machine-sized integers giving the dimensions for the result.

••• **NIntegrate**: The integrand $0.257394 e^{-0.208135 (-4.30784+x)^2}$ claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. [i](#)

••• **RandomVariate**: The array dimensions RandomVariate[PoissonDistribution[20.6526 t]] given in position 2 of RandomVariate[ExponentialDistribution[NIntegrate[claimSizeFunction[x] PDF[magnitudeDist, x], {x, -∞, ∞}]]], RandomVariate[PoissonDistribution[20.6526 t]]] should be a list of non-negative machine-sized integers giving the dimensions for the result.

••• **Probability**: Invalid input.

••• **NIntegrate**: The integrand $0.257394 e^{-0.208135 (-4.30784+x)^2}$ claimSizeFunction[x] has evaluated to non-numerical values for all sampling points in the region with boundaries {{1., 0.}}. [i](#)

••• **General**: Further output of NIntegrate::inumr will be suppressed during this calculation. [i](#)

••• **RandomVariate**: The array dimensions RandomVariate[PoissonDistribution[20.6526 t]] given in position 2 of RandomVariate[ExponentialDistribution[NIntegrate[claimSizeFunction[x] PDF[magnitudeDist, x], {x, -∞, ∞}]]], RandomVariate[PoissonDistribution[20.6526 t]]] should be a list of non-negative machine-sized integers giving the dimensions for the result.

••• **General**: Further output of RandomVariate::array will be suppressed during this calculation. [i](#)

••• **Probability**: Invalid input.

Break Even Point

```
In[•]:= calculateBreakEven[propertyValue_, annualPremium_] := PropertyValue / annualPremium;
```

Interactive Visualization

```
In[•]:= (*Clear any existing definitions*)
Clear[expectedClaimSize, claimSizeFunction, calculatePremium, calculateBreakEven];

(*Constants*)
meanMagnitude = 4.30784;
stdDevMagnitude = 1.54993;
```

```

poissonRate = 1 / 0.04842;
magnitudeDist = NormalDistribution[meanMagnitude, stdDevMagnitude];

(*Define claim size function separately to avoid protected function issues*)
claimSizeFunction[magnitude_, propertyValue_] :=
  propertyValue * (0.01 * E^(0.5 * magnitude));

(*Define expected claim size using discrete sum*)
expectedClaimSize[propertyValue_] := Module[
  {magnitudeRange, probabilities, claims, total}, magnitudeRange = Range[1.4, 7.4, 0.1];
  probabilities = PDF[magnitudeDist, #] & /@ magnitudeRange;
  claims = claimSizeFunction[#, propertyValue] & /@ magnitudeRange;
  total = Sum[claims[[i]] * probabilities[[i]], {i, 1, Length[magnitudeRange]}] * 0.1;
  total];

calculatePremium[propertyValue_, years_, safetyLoading_, deductible_] := Module[
  {expectedLoss}, expectedLoss = expectedClaimSize[propertyValue] * poissonRate * years;
  expectedLoss * (1 + safetyLoading) * (1 - deductible) ];

calculateBreakEven[propertyValue_, annualPremium_] := propertyValue / annualPremium;

(*Main visualization*)
DynamicModule[{lastPropertyValue = 100 000,
  lastYears = 10, lastSafetyLoading = 0.2, lastDeductible = 0.05},
 Manipulate[Module[{is, ip, premium, annualPremium, breakEvenYears,
   paymentsData, premiumsData}, is = {280, 140};
  ip = {{50, 2}, {40, 2}};
  premium = calculatePremium[propertyValue, years, safetyLoading, deductible];
  annualPremium = premium / years;
  breakEvenYears = calculateBreakEven[propertyValue, annualPremium];
  paymentsData =
    Table[expectedClaimSize[propertyValue] * poissonRate * t, {t, 0, years}];
  premiumsData = Table[annualPremium * t, {t, 0, years}];
  Grid[{{Labeled[ListLinePlot[{Thread[{Range[0, years], premiumsData}],
    Thread[{Range[0, years], paymentsData}]], Axes -> False, Filling -> 0, Frame -> True,
    ImageSize -> is, PlotMarkers -> {Automatic, 4}, PlotRange -> All, ImagePadding -> ip,
    PlotRangePadding -> {{Automatic, Scaled[0.1]}, {Automatic, Scaled[0.05]}},
    AspectRatio -> 1/2, FrameLabel -> {"Years from inception", "Payment $"},

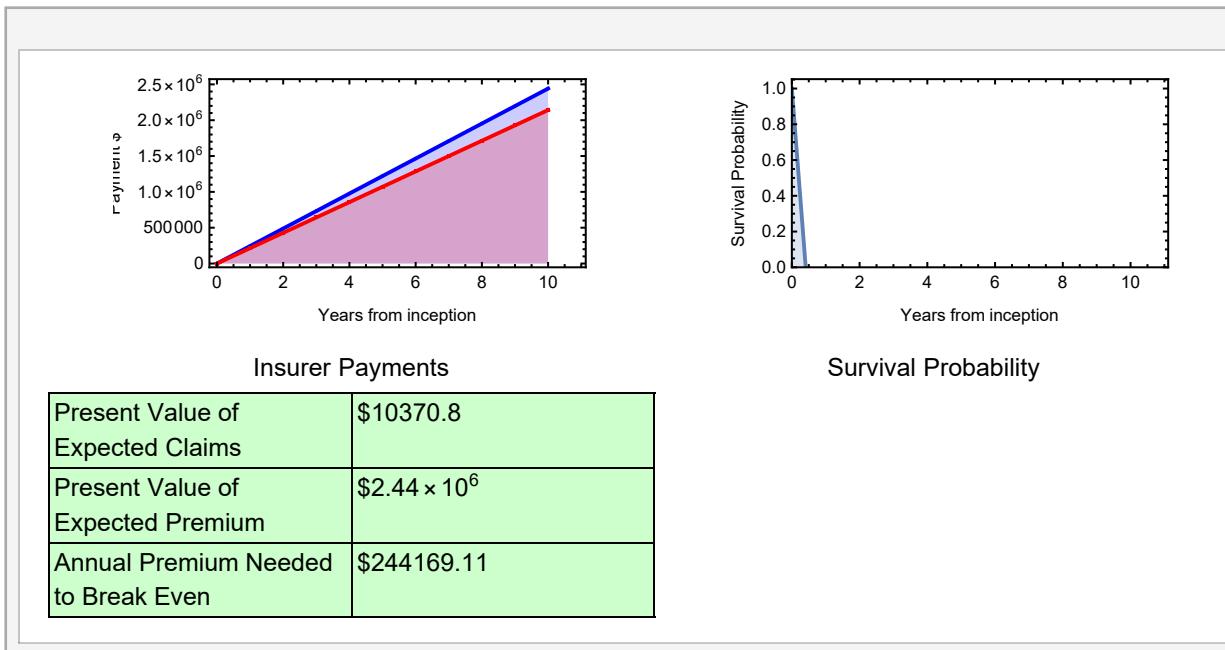
    PlotStyle -> {{Blue}, {Red}}], Text@"Insurer Payments"],

    Labeled[ListLinePlot[Table[{t, 1 - t / breakEvenYears}, {t, 0, years}], Axes -> False,
    Filling -> 0, Frame -> True, ImageSize -> is, PlotMarkers -> {Automatic, 4},
    PlotRange -> {{0, years}, {0, 1}}, ImagePadding -> ip, PlotRangePadding ->
      {{Automatic, Scaled[0.1]}, {Automatic, Scaled[0.05]}}, AspectRatio -> 1/2,
    FrameLabel -> {"Years from inception", "Survival Probability"}],
    Text@"Survival Probability"]}], {TextGrid[{{"Present Value of Expected Claims",
      Row[{"$", NumberForm[expectedClaimSize[propertyValue]]}]}, {"Present Value of Expected Premium", Row[{"$", NumberForm[premium, {9, 2}]}]}]}];

```

```
{"Annual Premium Needed to Break Even",
 Row[{"$", NumberForm[annualPremium, {9, 2}]}]], Frame → All,
 ItemSize → {14, 1}, Dividers → All, Background → Lighter[Green, 0.8]]}]],
 {{propertyValue, 100000, "Property Value"}, 10000, 1000000, 10000, Appearance → "Labeled", ImageSize → Medium},
 {{years, 10, "Coverage Period (Years)"}, 1, 30, 1, Appearance → "Labeled", ImageSize → Medium},
 {{safetyLoading, 0.2, "Safety Loading"}, 0.1, 0.5, 0.05, Appearance → "Labeled", ImageSize → Medium},
 {{deductible, 0.05, "Deductible"}, 0.01, 0.20, 0.01, Appearance → "Labeled", ImageSize → Medium},
 Initialization :> (lastPropertyValue = propertyValue;
 lastYears = years;
 lastSafetyLoading = safetyLoading;
 lastDeductible = deductible), SaveDefinitions → True] ]
```

Out[]=



```

In[ ]:= (*Clear any existing definitions*)
Clear[expectedClaimSize, claimSizeFunction, calculatePremium,
      calculateBreakEven, simulateReserveProcess];

(*Constants*)
meanMagnitude = 4.30784;
stdDevMagnitude = 1.54993;
poissonRate = 1 / 0.04842;
magnitudeDist = NormalDistribution[meanMagnitude, stdDevMagnitude];

(*Define claim size function separately to avoid protected function issues*)
claimSizeFunction[magnitude_, propertyValue_] :=
  propertyValue * (0.01 * E^(0.5 * magnitude));

(*Define expected claim size using discrete sum*)
expectedClaimSize[propertyValue_] := Module[
  {magnitudeRange, probabilities, claims, total}, magnitudeRange = Range[1.4, 7.4, 0.1];
  probabilities = PDF[magnitudeDist, #] & /@ magnitudeRange;
  claims = claimSizeFunction[#, propertyValue] & /@ magnitudeRange;
  total = Sum[claims[[i]] * probabilities[[i]], {i, 1, Length[magnitudeRange]}] * 0.1;
  total];

calculatePremium[propertyValue_, years_, safetyLoading_, deductible_] := Module[
  {expectedLoss}, expectedLoss = expectedClaimSize[propertyValue] * poissonRate * years;
  expectedLoss * (1 + safetyLoading) * (1 - deductible)];

calculateBreakEven[propertyValue_, annualPremium_] := propertyValue / annualPremium;

```

```
(*Simulate Reserve Process with Poisson Claim Arrivals*)
simulateReserveProcess[initialReserve_, premiumRate_, years_, propertyValue_] :=
Module[{timeSteps, totalTime, claimTimes, claimSizes,
reserveTrajectory, premiumAccrued}, totalTime = years * 365;
(*Simulate daily time steps for given years*)
(*Generate random claim times using Poisson process*)claimTimes =
Accumulate[RandomVariate[ExponentialDistribution[poissonRate], totalTime]];
claimTimes = Select[claimTimes, # ≤ totalTime &];
(*Filter within the time range*)(*Generate claim sizes based on the
magnitude distribution*)claimSizes = claimSizeFunction[#, propertyValue] & /@
RandomVariate[magnitudeDist, Length[claimTimes]];
(*Initialize reserve and simulate*)reserveTrajectory = {};
premiumAccrued = 0;
For[t = 1, t ≤ totalTime, t ++, premiumAccrued += premiumRate / 365;
(*Premium accrues daily*)If[MemberQ[Round[claimTimes], t],
(*Subtract claims when they occur*)premiumAccrued -= claimSizes[[1]];
claimSizes = Rest[claimSizes]; (*Remove processed claim*]);
AppendTo[reserveTrajectory, initialReserve + premiumAccrued];];
reserveTrajectory];

(*Main visualization*)
DynamicModule[{lastPropertyValue = 100 000,
lastYears = 10, lastSafetyLoading = 0.2, lastDeductible = 0.05},
Manipulate[Module[{is, ip, premium, annualPremium, breakEvenYears,
paymentsData, premiumsData, reserveData}, is = {280, 140};
ip = {{50, 2}, {40, 2}};
premium = calculatePremium[propertyValue, years, safetyLoading, deductible];
annualPremium = premium / years;
breakEvenYears = calculateBreakEven[propertyValue, annualPremium];
paymentsData =
Table[expectedClaimSize[propertyValue] * poissonRate * t, {t, 0, years}];
premiumsData = Table[annualPremium * t, {t, 0, years}];
(*Generate reserve data using the corrected process*)reserveData =
simulateReserveProcess[initialReserve, annualPremium, years, propertyValue];
Grid[{{Labeled[ListLinePlot[{Thread[{Range[0, years], premiumsData}],
Thread[{Range[0, years], paymentsData}]}, Axes → False, Filling → 0, Frame → True,
ImageSize → is, PlotMarkers → {Automatic, 4}, PlotRange → All, ImagePadding → ip,
PlotRangePadding → {{Automatic, Scaled[0.1]}, {Automatic, Scaled[0.05]}},
AspectRatio → 1/2, FrameLabel → {"Years from inception", "Payment $"},

PlotStyle → {{Blue}, {Red}}], Text@"Insurer Payments"],

Labeled[ListLinePlot[Thread[{Range[Length[reserveData]], reserveData}],

Axes → False, Filling → Axis, Frame → True, ImageSize → is, PlotStyle → Green,
PlotRange → All, AspectRatio → 1/2, FrameLabel → {"Days", "Reserve Amount"},

PlotLabel → "Cramer-Lundberg Reserve Process"], Text@"Reserve Process"]}],

{TextGrid[{{"Present Value of Expected Claims",
Row[{"$ ", NumberForm[expectedClaimSize[propertyValue]]}]}]}},
```

```

    {"Present Value of Expected Premium", Row[{"$", NumberForm[premium, {9, 2}]}]}, 
    {"Annual Premium Needed to Break Even",
     Row[{"$", NumberForm[annualPremium, {9, 2}]}]}, Frame → All,
    ItemSize → {14, 1}, Dividers → All, Background → Lighter[Green, 0.8]]}]], 
{{propertyValue, 100000, "Property Value"}, 
 10000, 
 1000000, 
 10000, 
 Appearance → "Labeled",
 ImageSize → Medium},
 {{years, 10, "Coverage Period (Years)"}, 
 1, 
 30, 
 1, 
 Appearance → "Labeled",
 ImageSize → Medium},
 {{safetyLoading, 0.2, "Safety Loading"}, 
 0.1, 0.5, 0.05, 
 Appearance → "Labeled",
 ImageSize → Medium},
 {{deductible, 0.05, "Deductible"}, 0.01, 0.20, 
 0.01, Appearance → "Labeled",
 ImageSize → Medium},
 {{initialReserve, 100000, "Initial Reserve"}, 
 10000, 1000000, 10000, 
 Appearance → "Labeled", ImageSize → Medium},
 Initialization :> (lastPropertyValue = propertyValue;
 lastYears = years;
 lastSafetyLoading = safetyLoading;
 lastDeductible = deductible), SaveDefinitions → True] ]

```

Out[•]:=



```

In[•]:= (*Main visualization with an additional plot*)
Manipulate[Module[{is, ip, premium, annualPremium, breakEvenYears, paymentsData,
  premiumsData, reserveProcessData, initialReserve, claimSizeData, magnitudeRange},
  (*Set initial reserve as 20% of property value*)initialReserve = 0.2 * propertyValue;
  is = {280, 140};
  ip = {{50, 2}, {40, 2}}];

```

```

premium = calculatePremium[propertyValue, years, safetyLoading, deductible];
annualPremium = premium / years;
breakEvenYears = calculateBreakEven[propertyValue, annualPremium];
paymentsData = Table[expectedClaimSize[propertyValue] * poissonRate * t, {t, 0, years}];
premiumsData = Table[annualPremium * t, {t, 0, years}];
(*Simulate reserve process*)
reserveProcessData = simulateReserveProcess[initialReserve, annualPremium, years];
(*Cramer-Lundberg Reserve Process Simulation*)
simulateReserveProcess[initialReserve_, premiumRate_, years_] :=
Module[{claimSizes, numClaims, reserveTrajectory, time}, SeedRandom[42];
(*For reproducibility*) claimSizes =
claimSizeFunction[#, 1] & /@ RandomVariate[magnitudeDist, 1000];
numClaims = RandomVariate[PoissonDistribution[poissonRate * years]];
reserveTrajectory =
Table[initialReserve + premiumRate * t - Total[Take[claimSizes, UpTo[Min[Count[
Accumulate[claimSizes] \leq t, True], Length[claimSizes]]]]], {t, 0, years}];
reserveTrajectory];
(*Generate claim size data*) magnitudeRange = Range[1.4, 7.4, 0.1];
claimSizeData = claimSizeFunction[#, PropertyValue] & /@ magnitudeRange;
Grid[{{Labeled[ListLinePlot[{Thread[{Range[0, years], premiumsData}],
Thread[{Range[0, years], paymentsData}]}, Axes \rightarrow False, Filling \rightarrow 0, Frame \rightarrow True,
ImageSize \rightarrow is, PlotMarkers \rightarrow {Automatic, 4}, PlotRange \rightarrow All, ImagePadding \rightarrow ip,
PlotRangePadding \rightarrow {{Automatic, Scaled[0.1]}, {Automatic, Scaled[0.05]}},
AspectRatio \rightarrow 1/2, FrameLabel \rightarrow {"Years from inception", "Payment $"},

PlotStyle \rightarrow {{Blue}, {Red}}}, Text["Insurer Payments"]], Labeled[ListLinePlot[Thread[{Range[1.4, 7.4, 0.1], claimSizeData}],
Axes \rightarrow False, Filling \rightarrow 0, Frame \rightarrow True, ImageSize \rightarrow is,
PlotMarkers \rightarrow {Automatic, 4}, PlotRange \rightarrow All, ImagePadding \rightarrow ip,
PlotRangePadding \rightarrow {{Automatic, Scaled[0.1]}, {Automatic, Scaled[0.05]}},
AspectRatio \rightarrow 1/2, FrameLabel \rightarrow {"Magnitude", "Claim Size"},

PlotStyle \rightarrow {Purple}}, Text["Claim Size Distribution"]]}, {TextGrid[{{"Present Value of Expected Claims",
Row[{"$", NumberForm[expectedClaimSize[propertyValue]]}]}, {"Present Value of Expected Premium", Row[{"$", NumberForm[premium, {9, 2}]]]}, {"Annual Premium Needed to Break Even",
Row[{"$", NumberForm[annualPremium, {9, 2}]]]}], Frame \rightarrow All,
ItemSize \rightarrow {14, 1}, Dividers \rightarrow All, Background \rightarrow Lighter[Green, 0.8]}]}], (*Sliders and Controls*) {{PropertyValue, 100000,
"Property Value"}, 10000, 1000000, 10000, Appearance \rightarrow "Labeled", ImageSize \rightarrow Medium}, {{years, 10, "Coverage Period (Years)"}, 1, 30, 1}
]

```

```
Appearance → "Labeled",
ImageSize → Medium},
{{safetyLoading, 0.2, "Safety Loading"},  
 0.1,  
 0.5,  
 0.05,
Appearance → "Labeled",
ImageSize → Medium},
{{deductible, 0.05, "Deductible"},  
 0.01,  
 0.20, 0.01,
Appearance → "Labeled",
ImageSize → Medium},
{{a, 0.01, "Damage Coefficient (a)"},  
 0.001, 0.1, 0.001,
Appearance → "Labeled",
ImageSize → Medium},
{{b, 0.5, "Growth Rate (b)"}, 0.1, 1, 0.05,
Appearance → "Labeled",
ImageSize → Medium}]
```

Out[]=



```
In[ ]:= (*Clear any existing definitions*)
Clear[expectedClaimSize, claimSizeFunction, calculatePremium,
calculateBreakEven, simulateReserveProcess];

(*Constants*)
meanMagnitude = 4.30784;
```

```

stdDevMagnitude = 1.54993;
poissonRate = 1 / 0.04842;
magnitudeDist = NormalDistribution[meanMagnitude, stdDevMagnitude];

(*Define claim size function*)
claimSizeFunction[magnitude_, propertyValue_, damage_, growthRate_] :=
  propertyValue * (damage * E^(growthRate * magnitude));

(*Define expected claim size using discrete sum*)
expectedClaimSize[propertyValue_, damage_, growthRate_] := Module[
  {magnitudeRange, probabilities, claims, total}, magnitudeRange = Range[1.4, 7.4, 0.1];
  probabilities = PDF[magnitudeDist, #] & /@ magnitudeRange;
  claims = claimSizeFunction[#, propertyValue, damage, growthRate] & /@ magnitudeRange;
  total = Sum[claims[[i]] * probabilities[[i]], {i, 1, Length[magnitudeRange]}] * 0.1;
  total];

calculatePremium[propertyValue_, years_, safetyLoading_,
  deductible_, damage_, growthRate_] := Module[{expectedLoss}, expectedLoss =
  expectedClaimSize[propertyValue, damage, growthRate] * poissonRate * years;
  expectedLoss * (1 + safetyLoading) * (1 - deductible)];

calculateBreakEven[propertyValue_, annualPremium_] := propertyValue / annualPremium;

(*Simulate Reserve Process with Poisson Claim Arrivals*)
simulateReserveProcess[initialReserve_,
  premiumRate_, years_, propertyValue_, damage_, growthRate_] :=
Module[{totalTime, claimTimes, claimSizes, reserveTrajectory, premiumAccrued},
  totalTime = years * 365;
  claimTimes =
    Accumulate[RandomVariate[ExponentialDistribution[poissonRate], totalTime]];
  claimTimes = Select[claimTimes, # ≤ totalTime &];
  claimSizes = claimSizeFunction[#, propertyValue, damage, growthRate] & /@
    RandomVariate[magnitudeDist, Length[claimTimes]];
  reserveTrajectory = {};
  premiumAccrued = 0;
  For[t = 1, t ≤ totalTime, t++, premiumAccrued += premiumRate / 365;
    If[MemberQ[Round[claimTimes], t], premiumAccrued -= claimSizes[[1]];
     claimSizes = Rest[claimSizes]];
    AppendTo[reserveTrajectory, initialReserve + premiumAccrued];];
  reserveTrajectory];

(*Main visualization*)
DynamicModule[{lastPropertyValue = 100000, lastYears = 10, lastSafetyLoading = 0.2,
  lastDeductible = 0.05, lastDamage = 0.01, lastGrowthRate = 0.5},
Manipulate[Module[{is, ip, premium, annualPremium, breakEvenYears,
  paymentsData, premiumsData, reserveData, claimSizePlot}, is = {280, 140};
  ip = {{50, 2}, {40, 2}}];

```

```

premium = calculatePremium[propertyValue,
    years, safetyLoading, deductible, damage, growthRate];
annualPremium = premium / years;
breakEvenYears = calculateBreakEven[propertyValue, annualPremium];
paymentsData = Table[expectedClaimSize[propertyValue, damage, growthRate] *
    poissonRate * t, {t, 0, years}];
premiumsData = Table[annualPremium * t, {t, 0, years}];
reserveData = simulateReserveProcess[initialReserve,
    annualPremium, years, propertyValue, damage, growthRate];
claimSizePlot = Plot[claimSizeFunction[x, propertyValue, damage, growthRate] *
    PDF[magnitudeDist, x], {x, 1.4, 7.4}, Axes → False, Frame → True,
    ImageSize → is, FrameLabel → {"Magnitude", "Claim Size Density"},

    PlotStyle → Blue, Filling → Axis, AspectRatio → 1/2, PlotRange → All,
    PlotLabel → Row[{("Claim Size Distribution (Damage: ", NumberForm[damage, {2, 2}],
        ", Growth Rate: ", NumberForm[growthRate, {2, 2}], ")")}]];

Grid[{{Labeled[ListLinePlot[{Thread[{Range[0, years], premiumsData}],
    Thread[{Range[0, years], paymentsData}], Axes → False, Filling → 0, Frame → True,
    ImageSize → is, PlotMarkers → {Automatic, 4}, PlotRange → All, ImagePadding → ip,
    PlotRangePadding → {{Automatic, Scaled[0.1]}, {Automatic, Scaled[0.05]}},
    AspectRatio → 1/2, FrameLabel → {"Years from inception", "Payment $"},

    PlotStyle → {{Blue}, {Red}}}, Text@"Insurer Payments"],

    Labeled[ListLinePlot[Thread[{Range[Length[reserveData]], reserveData}],
        Axes → False, Filling → Axis, Frame → True, ImageSize → is, PlotStyle → Green,
        PlotRange → All, AspectRatio → 1/2, FrameLabel → {"Days", "Reserve Amount"},

        PlotLabel → "Cramer-Lundberg Reserve Process"],

        Text@"Reserve Process"]}], {claimSizePlot}}]],

{{propertyValue, 100000, "Property Value"}, 10000,
  1000000,
  10000,
  Appearance → "Labeled",
  ImageSize → Medium},
{{years, 10, "Coverage Period (Years)" },
  1, 30, 1,
  Appearance → "Labeled",
  ImageSize → Medium},
{{safetyLoading, 0.2, "Safety Loading"}, 0.1, 0.5,
  0.05, Appearance → "Labeled",
  ImageSize → Medium},
{{deductible, 0.05, "Deductible"}, 0.01, 0.20, 0.01,
  Appearance → "Labeled", ImageSize → Medium},
{{damage, 0.01, "Damage"}, 0.005, 0.05, 0.005,
  Appearance → "Labeled", ImageSize → Medium},
{{growthRate, 0.5, "Growth Rate"}, 0.1, 1, 0.1,
  Appearance → "Labeled", ImageSize → Medium},
{{initialReserve, 100000, "Initial Reserve"}, 10000, 1000000,
  10000, Appearance → "Labeled", ImageSize → Medium},
Initialization :> (lastPropertyValue = PropertyValue;

```

```
lastYears = years; lastSafetyLoading = safetyLoading;
lastDeductible = deductible; lastDamage = damage;
lastGrowthRate = growthRate), SaveDefinitions → True] ]
```

Out[\circ] =

