Independent Assets Model (IAM)

Model Parameters

In this simple example, we have a group of 50 identical assets. They fail or work independently of one another. Their expected performance is a constant, “200” which can be thought of as 200 Watts if these assets each represent a solar panel. We aren’t doing anything diurnal yet, let alone meteorological. 50 panels with an expected 200 Watt output is a 10kW array.

In[2]:=
```
assetCount = 50
```

Out[2]=
```
50
```

In[3]:=
```
expectedAssetPerformance := 200
```

Next, we’ll assume a reliability of 0.98, meaning that in any simulation period, the asset has 98% chance of performing, and a 2% chance of failing. If it fails, we are going to assume it is repaired or replaced just in time for the next simulation period. 50 panels with a 2% chance of failure means that in any given time period, 1 panel is expected to be down.

In[4]:=
```
assetReliability = 0.98
```

Out[4]=
```
0.98
```

assetReliability is essentially the only tunable parameter relating to reliability in this model. All other parameters can be treated as known. assetReliability could be determined from a claims/awards history.

Next is the conversion from performance in watt-hours to dollars (the value chosen is equivalent to $0.10 per kilowatt hour). All units of time in this simulation will be in hours. One hour is going to be one simulation period. The units of performance value are dollars per kilowatt hour.

In[5]:=
```
performanceValue = 0.0001
```

Out[5]=
```
0.0001
```

Parameters like these would not be captured in code. Instead, they would live in their own file, and a user interface would exist to edit the asset groups, and the models for each type of asset in the asset groups. As assets and their descriptions get more complex, and assets groups get assembled into portfolios, how we model assets gets trickier and getting this out of code gets rapidly important.

Insurance Policy Parameters

The example policy duration is 365 days. Comvert to simulation units (24 hours per day).

In[6]:=
```
policyDuration = 365 * 24
```

Out[6]=
```
8760
```

There will be no payout unless more than 5% of the assets are down.

In[7]:=
```
deductible = 0.05 * assetCount * expectedAssetPerformance * performanceValue
```

Out[7]=
```
0.05
```

This solar panel array produces $1/hour of solar power. The deductible is $0.05. Only if the losses in any one-hour period exceed 5 cents is there a payout.

Probability Distributions

The following vector is a bernoulli distribution, with one element in the vector for each asset. *Mathematica's* RandomVariate function generates a vector assetCount long, with each component of the vector distributed to the Bernoulli distribution.

In[8]:=
```
assetPerformanceV[t_] := expectedAssetPerformance *
  RandomVariate[BernoulliDistribution[assetReliability], assetCount]
```

Example: asset performance factors at time t=32 (e.g., 8am-9am on the second day of the simulation).

In[9]:=
```
assetPerformanceV[32]
```

Out[9]=
```
{200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200}
```

In[10]:=
```
assetPerformanceV[32]
```

Out[10]=
```
{200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200,
 200, 200, 0, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200, 200}
```

Because there are 50 assets in our example, there are 50 elements in the vector. Re-evaluating the vector generates a new vector.

Next we aggregate to get the performance of the asset group:

In[11]:=
```
performance[t_] := Total[assetPerformanceV[t]]
```

Performance Deficits and Payouts

Next we find the performance deficit, and convert that into a dollar loss.

In[12]:=
```
loss[t_] := performanceValue * (expectedAssetPerformance * assetCount - performance[t])
```

In[20]:=
```
loss[72]
```

Out[20]=
```
0.02
```

Convert the performance deficit to a dollar value and compare it with the deductible.

In[14]:=
```
unclampedPayout[t_] := loss[t] - deductible
```

There is a major *Mathematica* gotcha avoided in the next step:

In[15]:=
```
payout[t_] := UnitStep[x] * x /. x -> unclampedPayout[t]
```

Time Series Generation

In[16]:=
```
award := Sum[payout[t], {t, 0, policyDuration - 1}]
```

In[22]:=
```
award
```

Out[22]=
```
10.67
```

Ensembles / Monte Carlo Averages

In[17]:=
```
ensembleCount = 10
```

Out[17]=
```
10
```

In[18]:=
```
averageAward = Sum[award, {i, ensembleCount}] / ensembleCount
```

Out[18]=
```
10.355
```

The preceding expression takes about 6 seconds to evaluate on 2.4 GHz Core 2 Duo Mac Laptop.

Exact Expression Using Binomial Distribution

In[19]:=
```
policyDuration * performanceValue * expectedAssetPerformance *
 Sum[Binomial[assetCount, failures] * (1.0 - assetReliability)^failures
   assetReliability^(assetCount - failures) * (failures - 2.5), {failures, 3, 50}]
```

Out[19]=
```
10.6394
```

So Far Unimplemented, but Very Important to Flush Out Modeling, Analysis and Workflow

- **Multiple Parameters**

  It is very important that we are able to demonstrate multiple failure modes and to distinguish between expected underperformance of components as happens say, when the sun doesn't shine, versus failures that we are more likely to include policy coverage for, such as a fire in an inverter.

  To illustrate that the model will need at least two parameters.

- **Calibration (also "known as parameter estimation")**

  If given a hypothetical claims/awards history, we will want to go back and estimate the one tunable parameter in this model (assetReliability).

- **Award Distribution**

  We have displayed average but have as yet done no other characterization or plotting of the award distribution.