Asset Management: Tutorial 2: Simulation

Luuk Pentinga MSc
l.pentinga@rug.nl

Contents

- Details on simulating deterioration paths
  - Examples of code
  - Considerations with simulating

- Dealing with complexities
  - Imperfect maintenance
  - Uncertain failure levels
  - Nonstationary degradation

Simulating deterioration paths

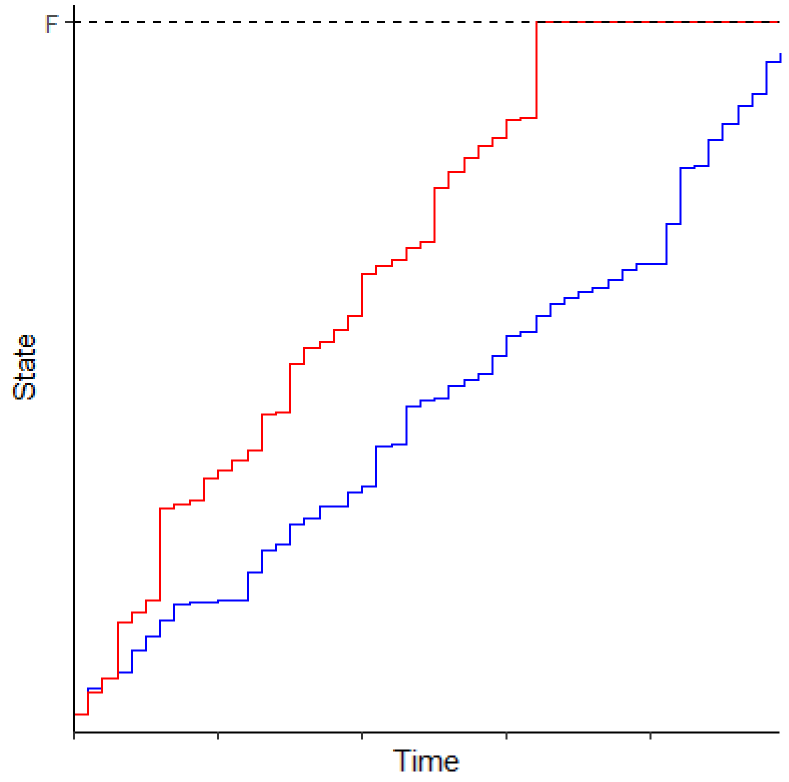
Deterioration paths

Remember, deterioration modelling usually consists of two parts:

1. A **state**, a value representing the current condition/wear level of the system.
2. A **failure level**, the state at which a machine fails.

Example: deterioration paths

The state increases randomly, and is measured every unit of time.



Simulation

Goal: Recreate these system dynamics many times.

We create many (1000+) of these deterioration paths, and observe some performance metrics from these paths.

Of course, this relies on us being able to accurately recreate deterioration paths (subject to a maintenance policy).

Simulation

In our data, we see that the state increases by a random amount every unit of time.

The idea is that we can randomly sample increases from this data, and use them as increases for a new deterioration path.

This works if each datapoint in our dataset is “equally likely” to happen at any point in time, for example when there is no interaction between the current state and the size of the increment.

Code

Let’s look at some code.

For our purposes, we only care about the length of a deterioration path, and which type of maintenance we do at the end.

Example:

Data

Increments
1.4
1.8
0.2
1.5
1.8
1.2
2.0
3.1
0.6
8.0
⋮

Random samples $\rightarrow$

Simulated path

Time	x	Δx
0	0	1.2
1	1.2	3.1
2	4.3	0.2
3	4.5	2.0
4	6.5	1.8
5	8.3	

$\}$ x1000

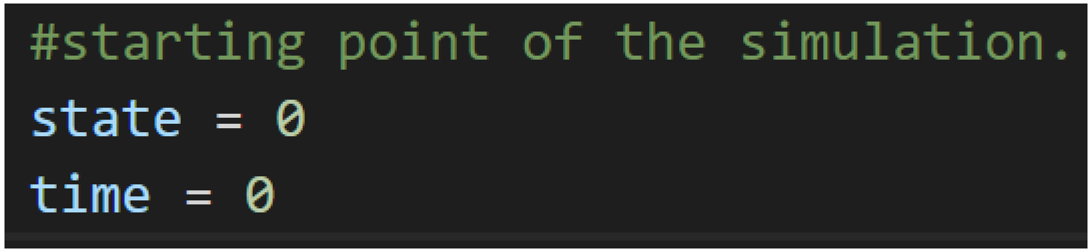
Code

We look at a function that takes a few inputs:

1. **input_data:** the dataset containing the increments.
2. **failure_level:** a number denoting the failure level.
3. **threshold:** a value denoting the preventive maintenance threshold.

Code

To create a deterioration path, we start at time 0 and state 0.



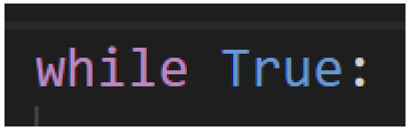
```
#starting point of the simulation.
state = 0
time = 0
```

Then, we need to repeatedly add random increments to the state -> a loop is required.

Problem: we don’t know how many iterations we should run in the loop.

Infinite loops

One solutions is the following:



```
while True:
```

You may be familiar with the while loop, which normally follows the structure:

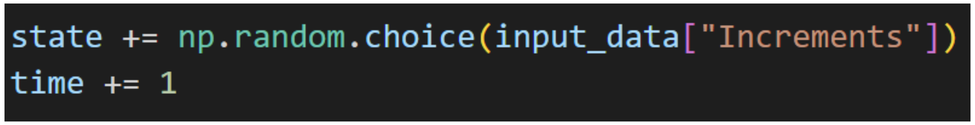
“while *condition*:”

Where the loop continues until the *condition* is false.

In this case, the loop will continue indefinitely. Instead, we can determine when to stop iterating somewhere else.

Code

Next, we increase the time and the state.



```python
state += np.random.choice(input_data["Increments"])
time += 1
```

› We use np.random.choice() to pick a random increment from our dataset.

› In this case, time is measured in integers, so it is increased by 1.

Code

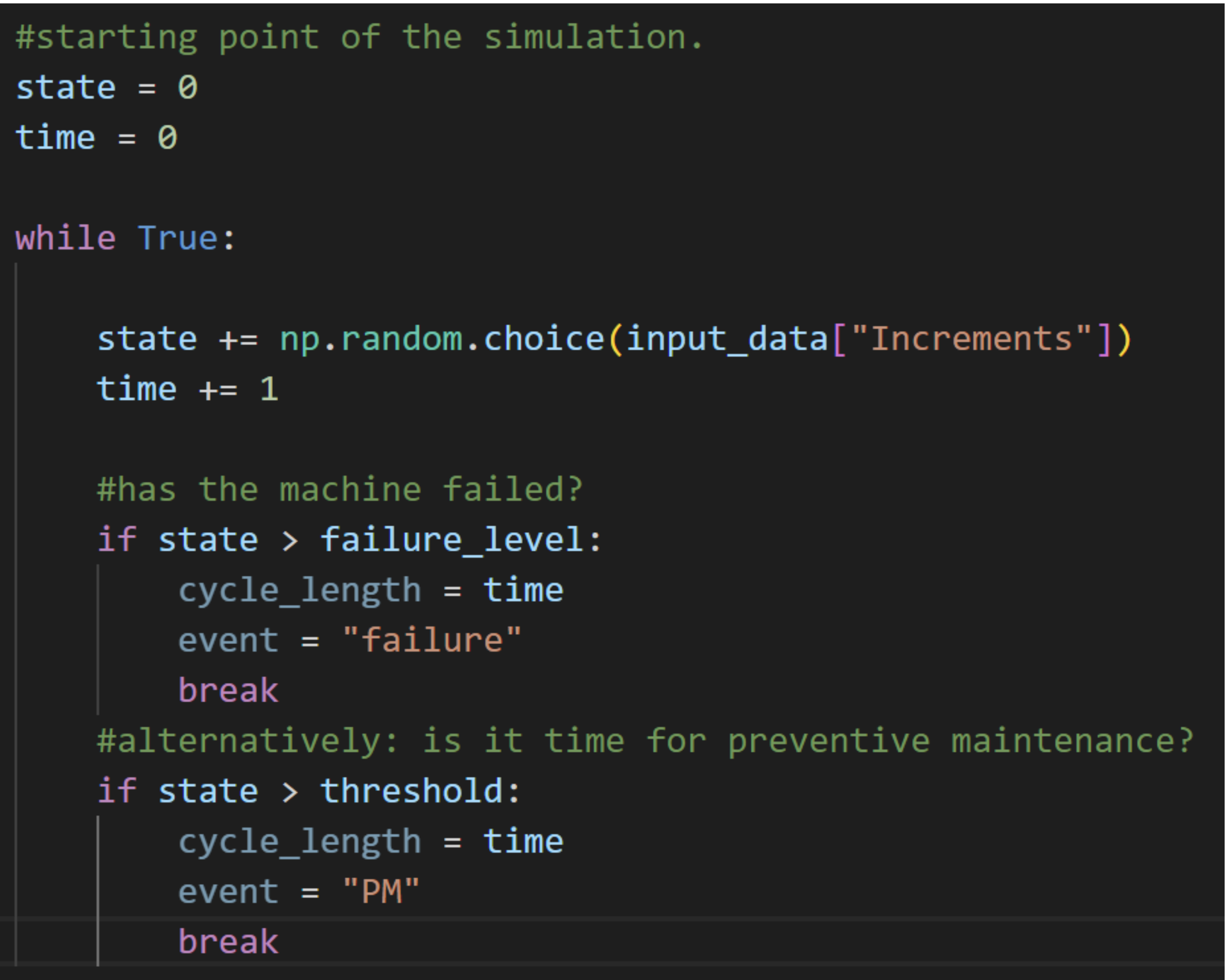
What remains, is to decide when to stop our path. We have two *exit conditions.*

1. The state is greater than the failure level.
2. The state is greater than the preventive maintenance threshold (but less than the failure level).

If one of these two conditions is met:

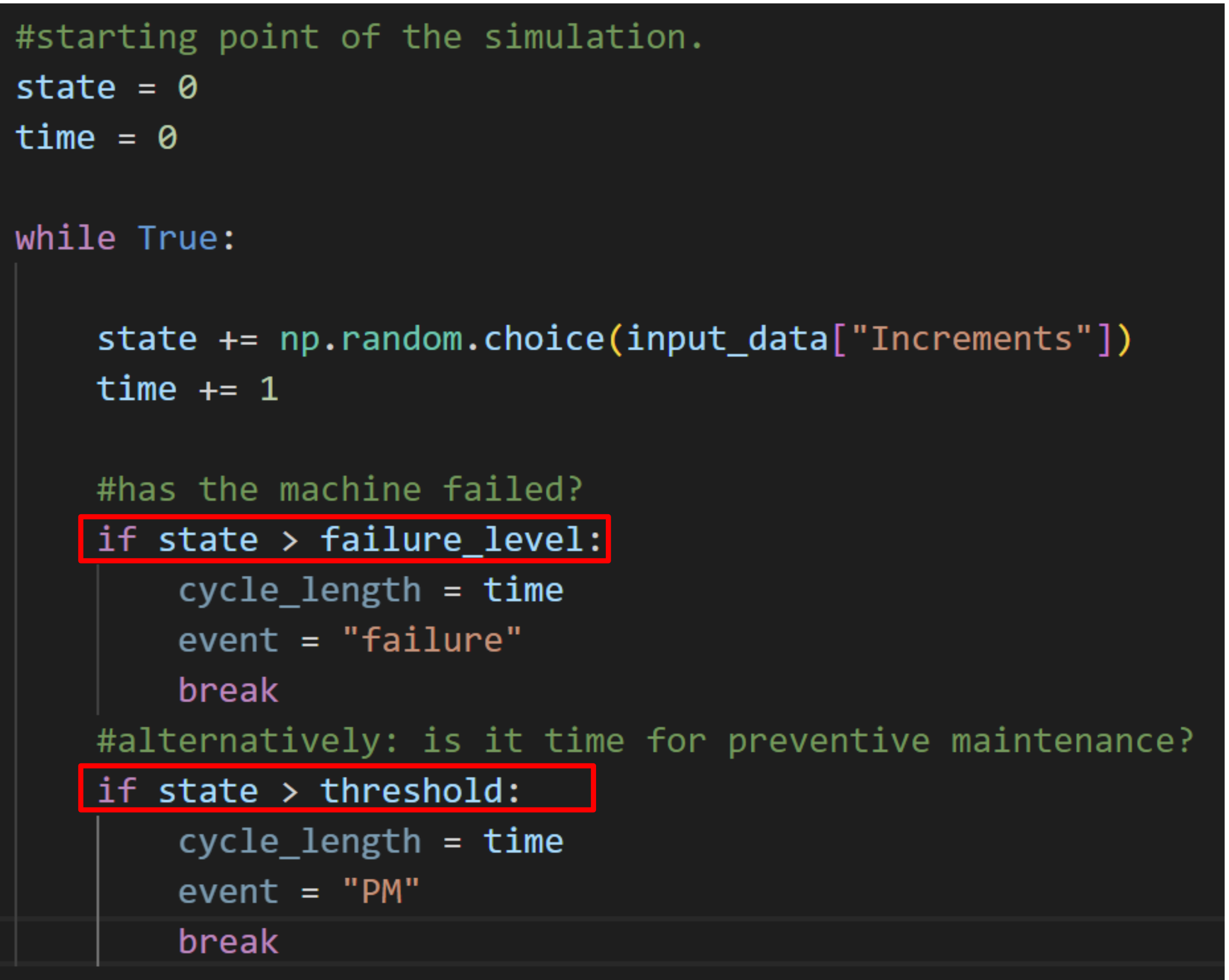
- Save the current time.
- Save the exit condition (failure or preventive maintenance)
- *Break* the loop.

Code



```
#starting point of the simulation.
state = 0
time = 0

while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

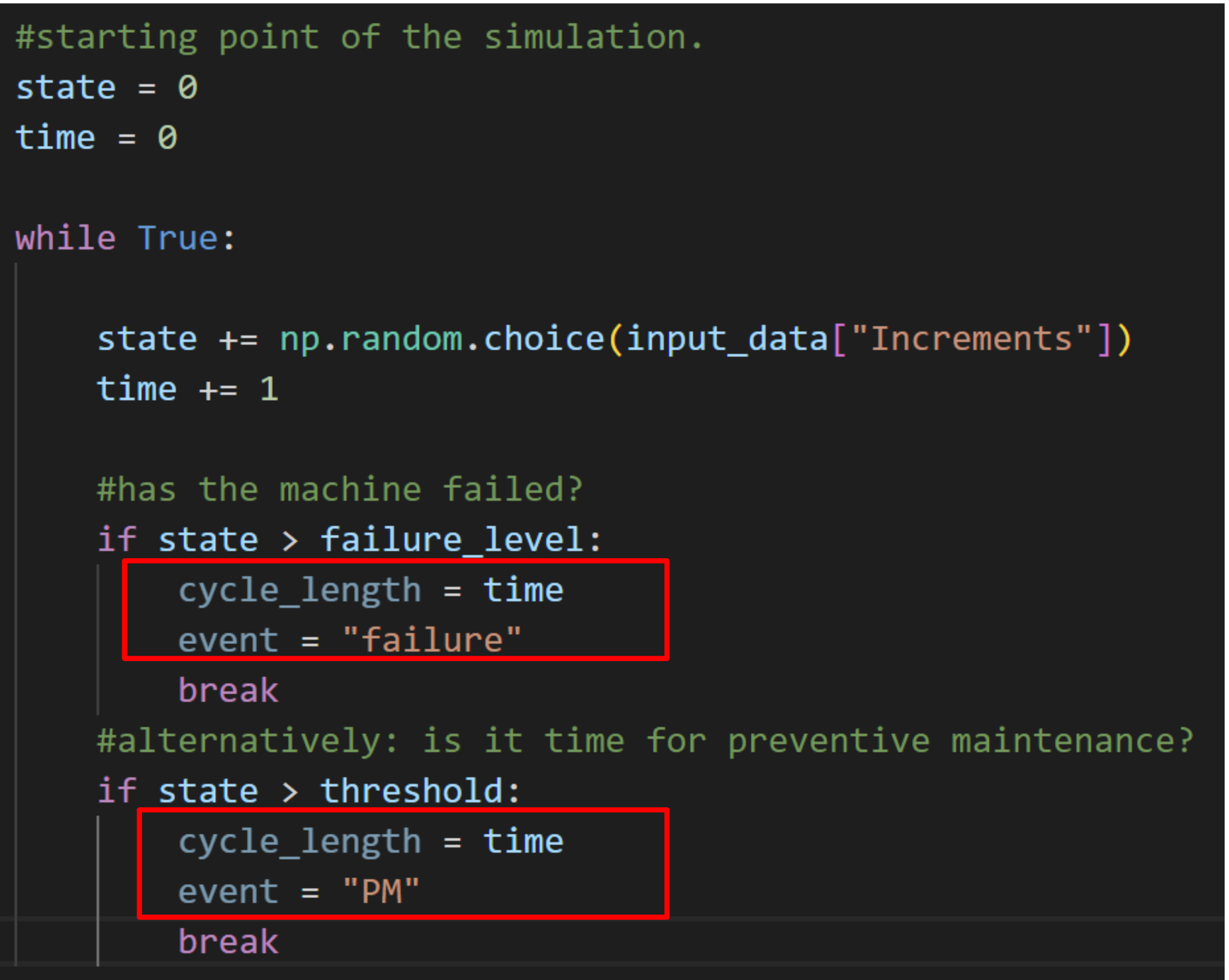
Code



```python
#starting point of the simulation.
state = 0
time = 0

while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

These if-statements determine the *exit condition.*

Code

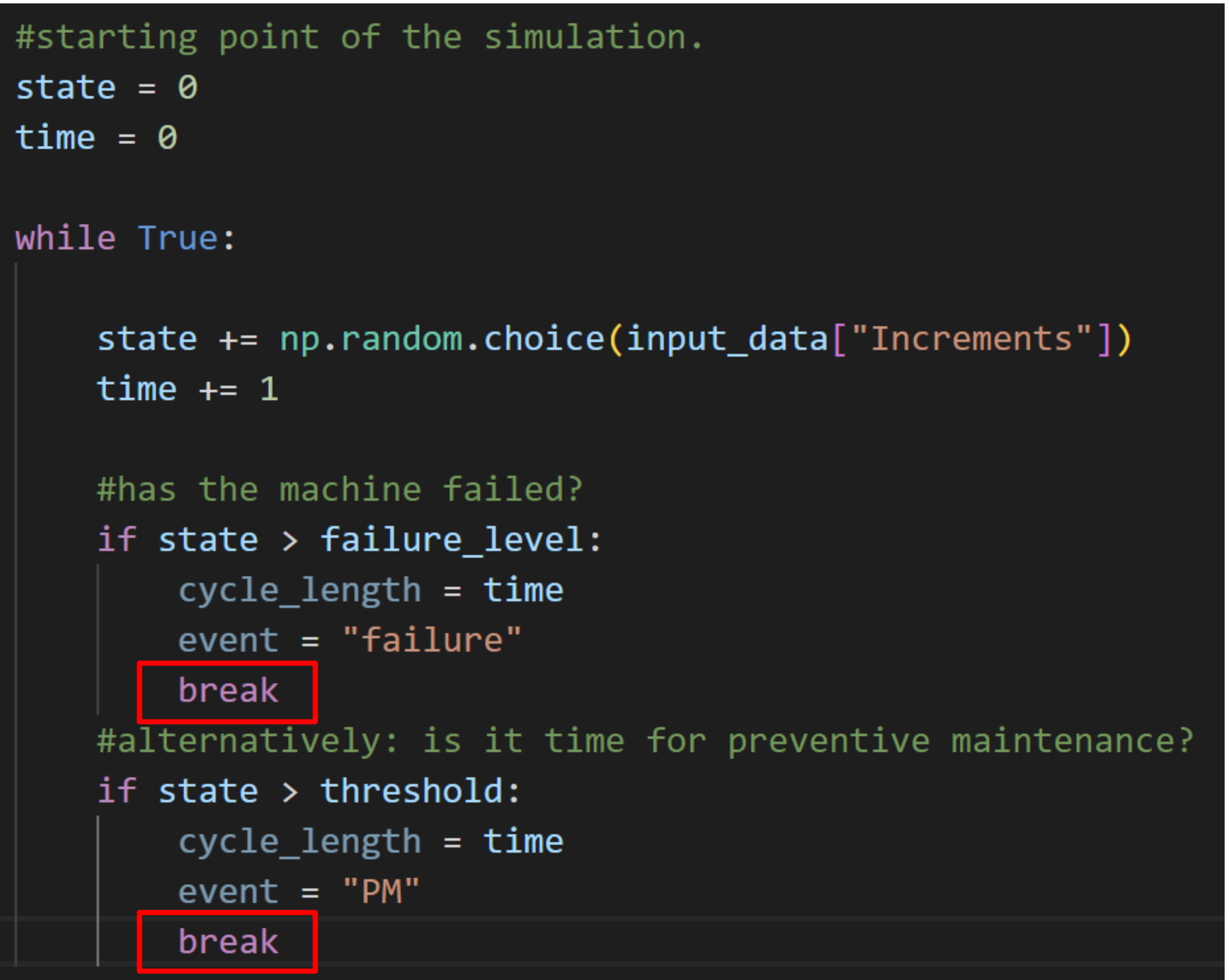


```
#starting point of the simulation.
state = 0
time = 0

while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

We save the cycle length and event.

When you do many simulations, you can also immediately put this in your output data frame!

Code



```
#starting point of the simulation.
state = 0
time = 0

while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

We use the *break* keyword to stop our loop.

When we call the break keyword, it ends the *innermost* loop that we are in.

So, if we are in a *nested loop*, we will only exit the deepest loop, but remain in the loop one level above this.

Code

And that is all you need to create one deterioration path!

Of course, you will need to do this many times, and for many different maintenance thresholds.

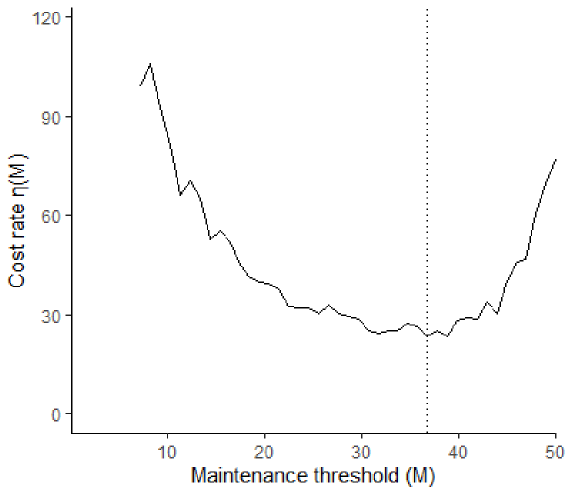
Just place this code in another loop, and make sure you store the results (event and cycle length) in some data frame.

Considerations with simulation

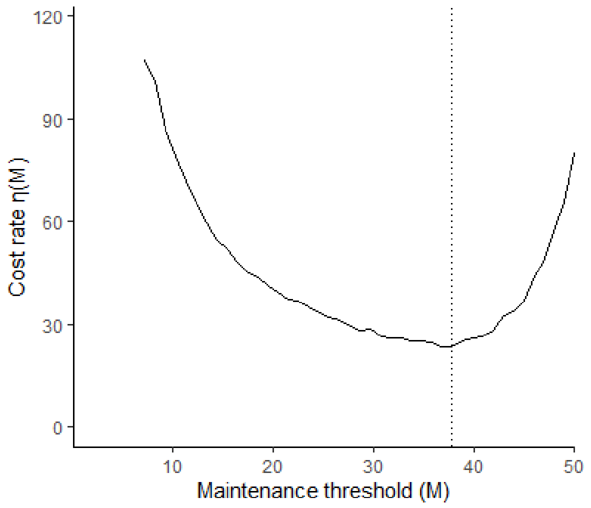
Number of simulations

Remember that simulating a large number of deterioration paths is vital for good results:

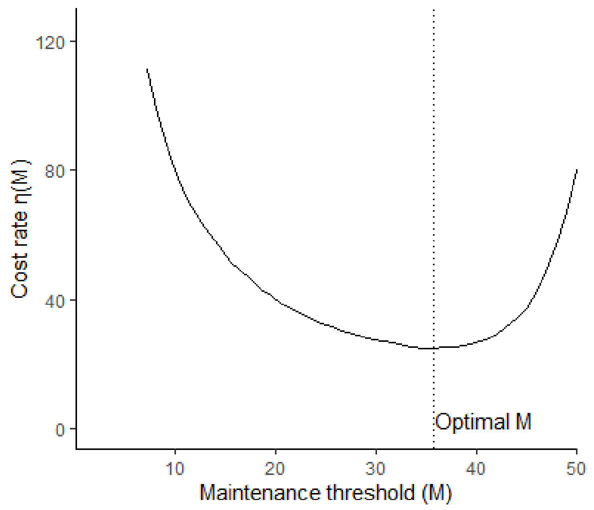
100 simulations



1,000 simulations



10,000 simulations



Number of simulations

With an extremely large set of simulations, our plot will be perfectly smooth.

Does this mean that our policy is perfect? -> **NO**

Instead, we will have the best policy **for our dataset**. But, if our dataset contains relatively many large deterioration increases, so will our simulations.

Number of simulations

Of course, large amounts of simulations will take more computational power.

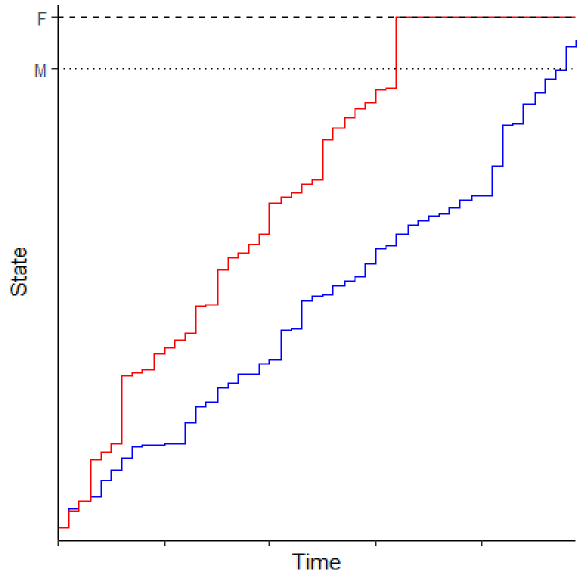
10,000 simulations for 100 different thresholds = 1,000,000 simulations

For more complicated problems, we may even need more simulations. For example, when we do a sensitivity analysis.

Number of simulations

We can think of ways to speed up the process.

Idea: If we have a deterioration path to failure, we can use this path to check the results for **each** threshold policy.



For complete deterioration paths, we can also check if a threshold M was successful afterwards.

Number of simulations

Instead of simulating deterioration paths for each maintenance threshold separately, we simulate deterioration paths to failure.

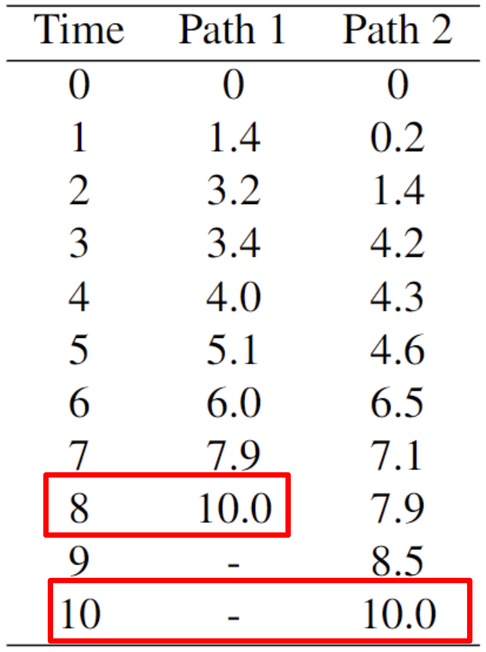
Then, we check afterwards if a deterioration path contains a state between the maintenance threshold M and the failure level F.

Yes: We do preventive maintenance

No: We do corrective maintenance

Example

We take an example with two deterioration paths:



Time	Path 1	Path 2
0	0	0
1	1.4	0.2
2	3.2	1.4
3	3.4	4.2
4	4.0	4.3
5	5.1	4.6
6	6.0	6.5
7	7.9	7.1
8	10.0	7.9
9	-	8.5
10	-	10.0

For M = F = 10: (only corrective)

#cycles ending in failures = 2
#cycles ending in PM = 0
average cycle length = (8 + 10) / 2 = 9

Example

We take an example with two deterioration paths:

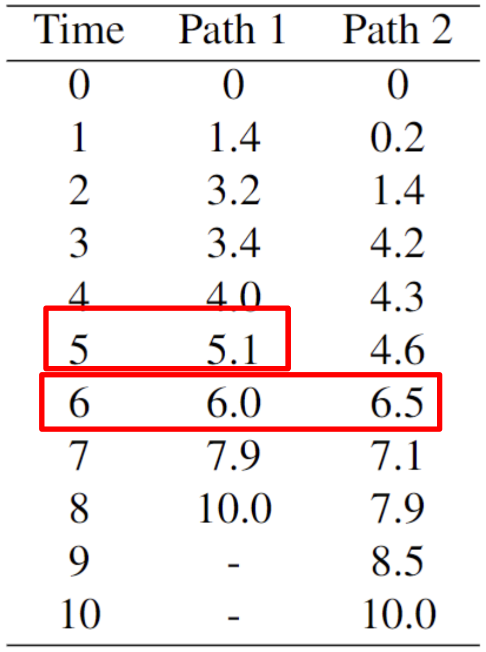
Time	Path 1	Path 2
0	0	0
1	1.4	0.2
2	3.2	1.4
3	3.4	4.2
4	4.0	4.3
5	5.1	4.6
6	6.0	6.5
7	7.9	7.1
8	10.0	7.9
9	-	8.5
10	-	10.0

For M = 8:

#cycles ending in failures = 1
#cycles ending in PM = 1
average cycle length = (8 + 9) / 2 = 8.5

Example

We take an example with two deterioration paths:



Time	Path 1	Path 2
0	0	0
1	1.4	0.2
2	3.2	1.4
3	3.4	4.2
4	4.0	4.3
5	5.1	4.6
6	6.0	6.5
7	7.9	7.1
8	10.0	7.9
9	-	8.5
10	-	10.0

For M = 5:

#cycles ending in failures = 0
#cycles ending in PM = 2
average cycle length = (5 + 6) / 2 = 5.5

Simulations

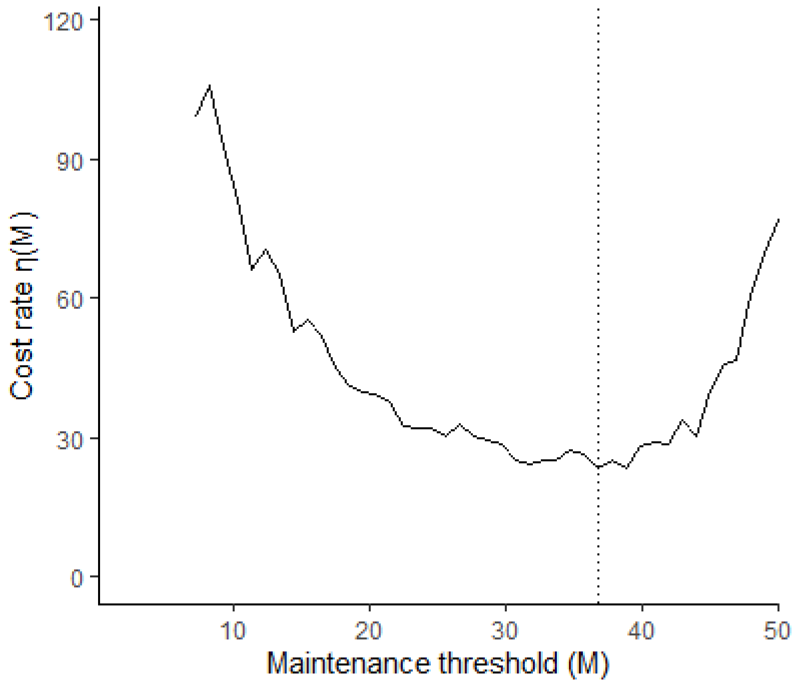
This way, we can reuse our deterioration paths!

Result: Instead of 10,000 simulations for each threshold, we only need to make 10,000 paths *once.*

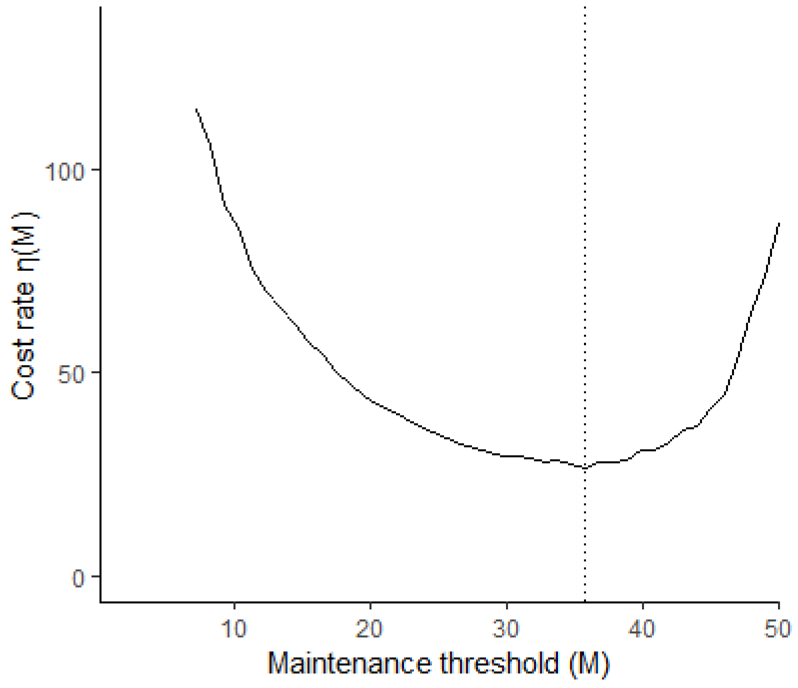
Though, we still need to check ~100 thresholds for 10,000 paths, but this is faster.

100 simulations

Original



Reusing paths



Why is this path far smoother?

Simulations

When we reuse deterioration paths, variation is shared over all policies:

› If our simulations are positive, each threshold will underestimate the cost rate.
› If our simulations are negative, each threshold will overestimate the cost rate.

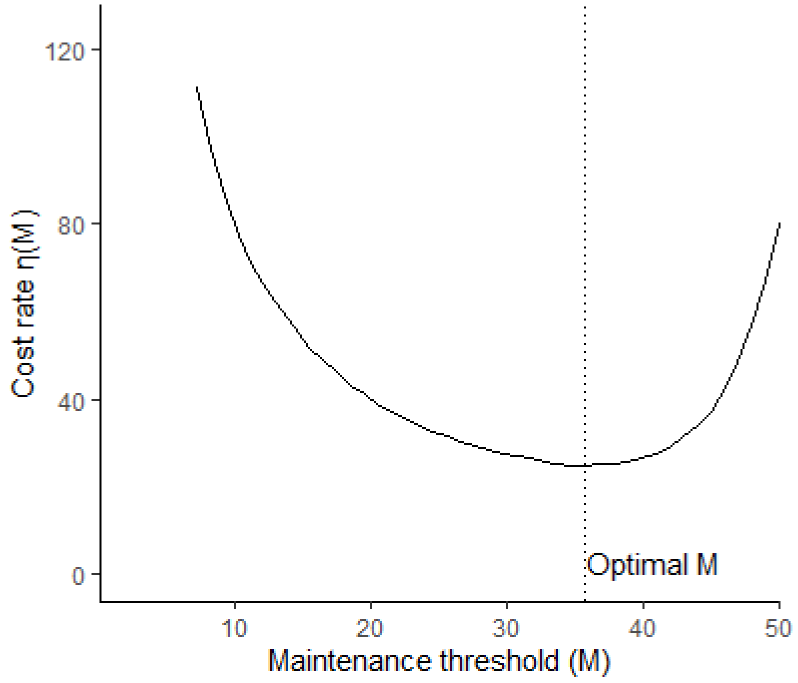
The “smoothness” of the curve is no longer a indicator of our precision.

Instead, rerun the simulations and check if the results are the same.

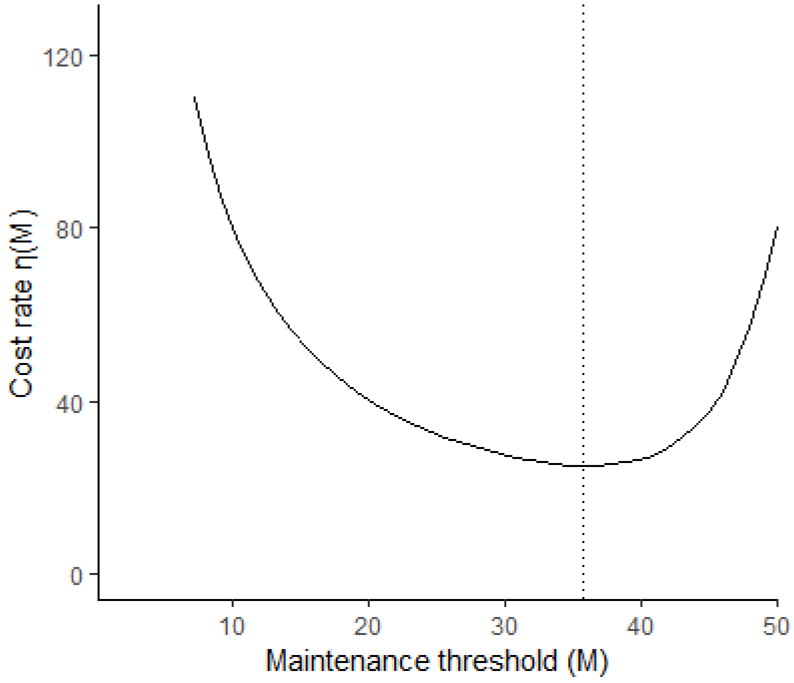
10,000 simulations

Though, with enough samples the methods should give the same results. In my case, it did so in about half the time.

Original



Reusing samples



Dealing with complexities:

Dealing with complexities

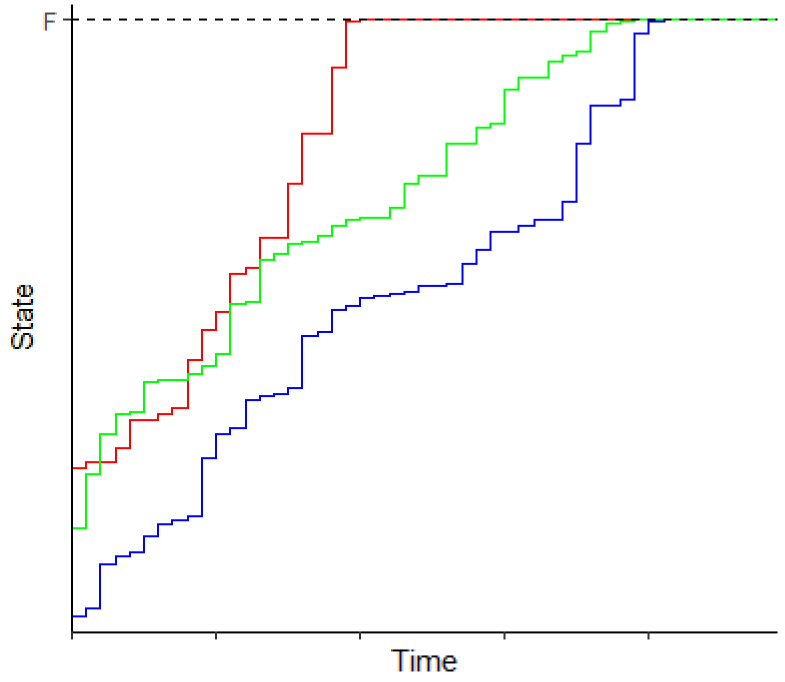
Recall that our deterioration information is hardly never this nice.

- Imperfect maintenance.
- Uncertain failure levels
- Nonstationary deterioration.
- Few measurable states available.
- Machine context influencing sensor output.

We will look at dealing with the first three.

Imperfect maintenance

Maintenance actions do not always return the machine to a as-good-as-new state.



If we create some deterioration paths, we can see if we have imperfect maintenance by watching the start of the paths.

Modelling imperfect maintenance

We will assume that the state that we go to after maintenance is stochastic. So, we first need to gather data on this.

This is best done during the data preparation phase:

Time	State	Increment
0	0	
1	1.4	1.4
2	3.2	1.8
3	3.4	0.2
⋮		
7	9.6	2.1
8	10	0.4
8	0.8	-9.4
9	1.2	1.2
⋮		

When we observe a negative increment, we store the state that we return to before deleting this entry.

Starting values
1.4
4.0
0.8
⋮

Modelling imperfect maintenance

Then, during simulations, we start each deterioration path at a randomly selecting point from this table!

Starting values
1.4
4.0
0.8
⋮

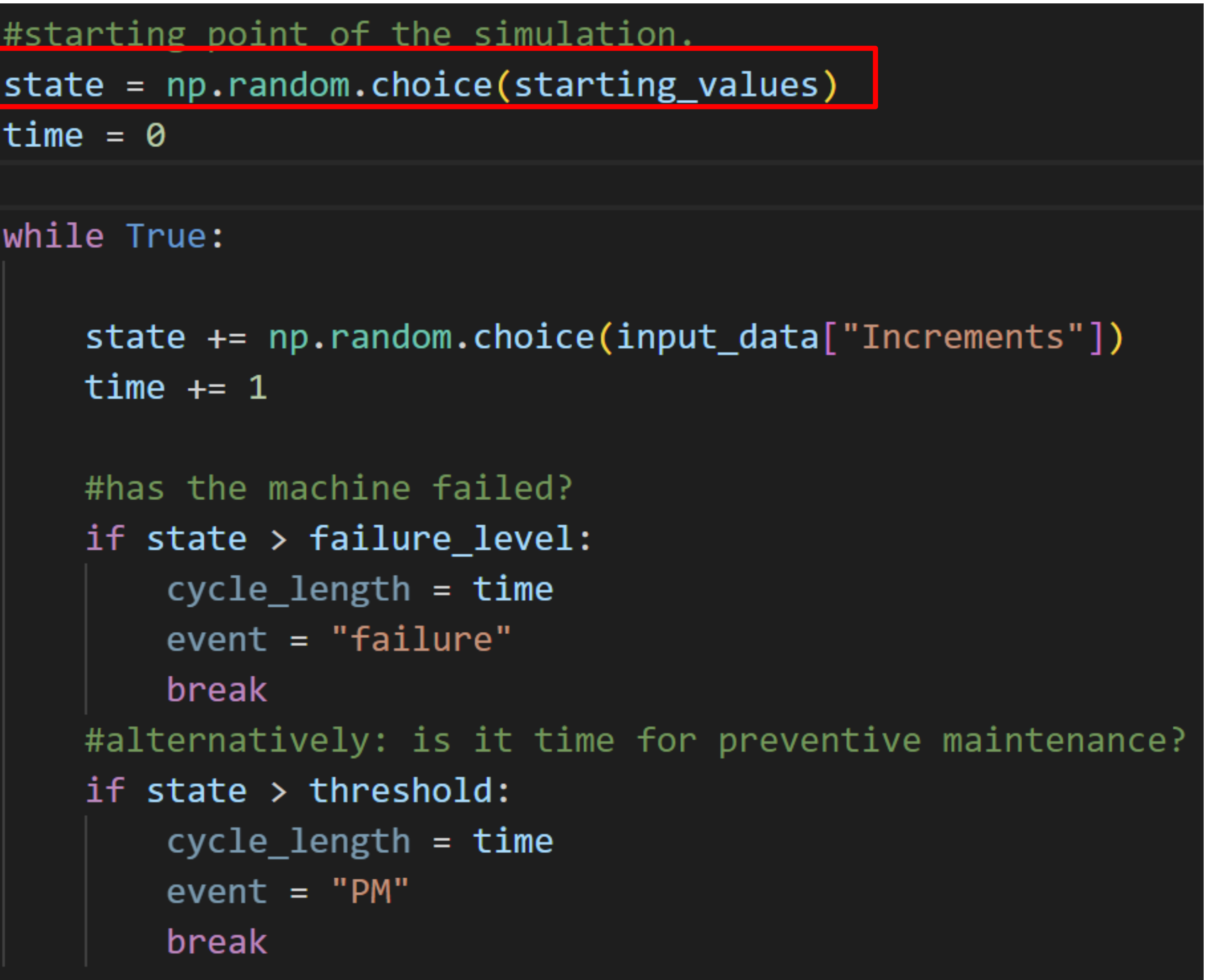
Starting points are picked randomly from the observed starting values

Time	Path 1	Path 2
0	0.8	4.0
1	1.4	5.1
2	3.2	5.4
3	3.4	6.0
4	4.0	6.5
5	5.1	6.7
6	6.0	6.9
7	7.9	7.1
8	10.0	7.9
9	-	8.5
10	-	10.0

Increments are still randomly sampled from our table, as normal.

Increments
1.4
1.8
0.2
1.5
1.8
1.2
2.0
3.1
0.6
8.0
⋮

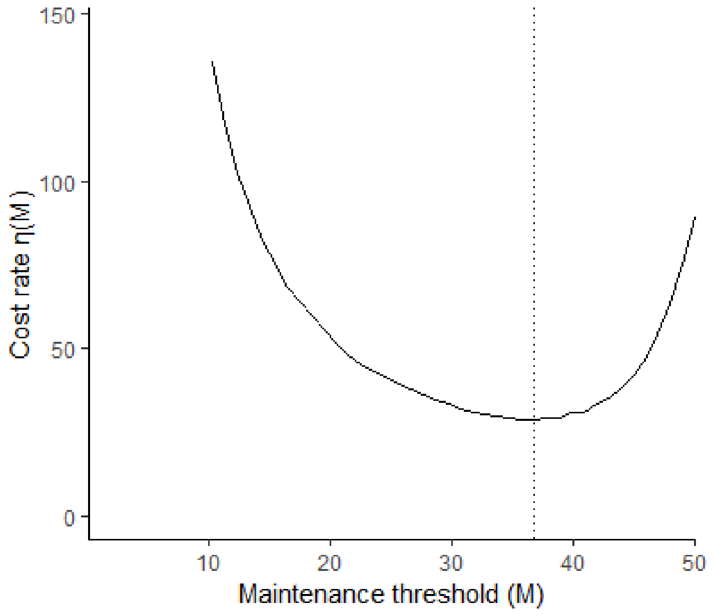
Code



```
#starting point of the simulation.
state = np.random.choice(starting_values)
time = 0


while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

Sampling a starting value is straightforward if we have the table of starting values.

Impact of imperfect maintenance



Example:

Starting with an average state of 10% of the failure level

η(M) = 23.36 -> η(M) = 28.90

M = 37.78 -> M = 36.76

› Imperfect maintenance only decreases the optimal threshold slightly.

› But, costs are significantly higher.

Imperfect maintenance

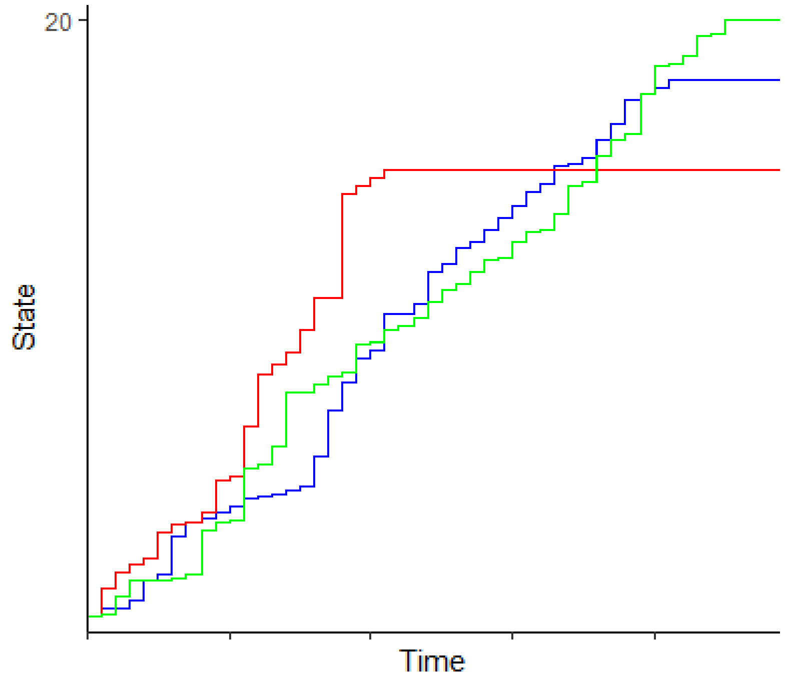
However, our modelling approach has some issues.

- Are preventive and corrective maintenance equally effective?
- Is there no relationship between the state before and after maintenance?

Ultimately, you need to verify that this is not the case before you can use this approach.

Uncertain failure levels

It can be hard to characterize an exact failure level.



We can determine if failure levels are uncertain by checking the end of our deterioration paths.

Uncertain failure levels

When simulating, we will need to account for this.

First, we need to gather some information about the failure level. Again, this is best done in the data preparation phase.

Time	State	Increment
0	0	
1	1.4	1.4
2	3.2	1.8
3	3.4	0.2
⋮		
7	9.6	2.1
8	$\boxed{9.8}$	$\boxed{0.4}$
8	$\boxed{0}$	$\boxed{-9.8}$
9	1.2	1.2
⋮		

Collect all states where the machine failed in one table.

$\longrightarrow$

Failure levels
8.7
11.0
9.8
⋮

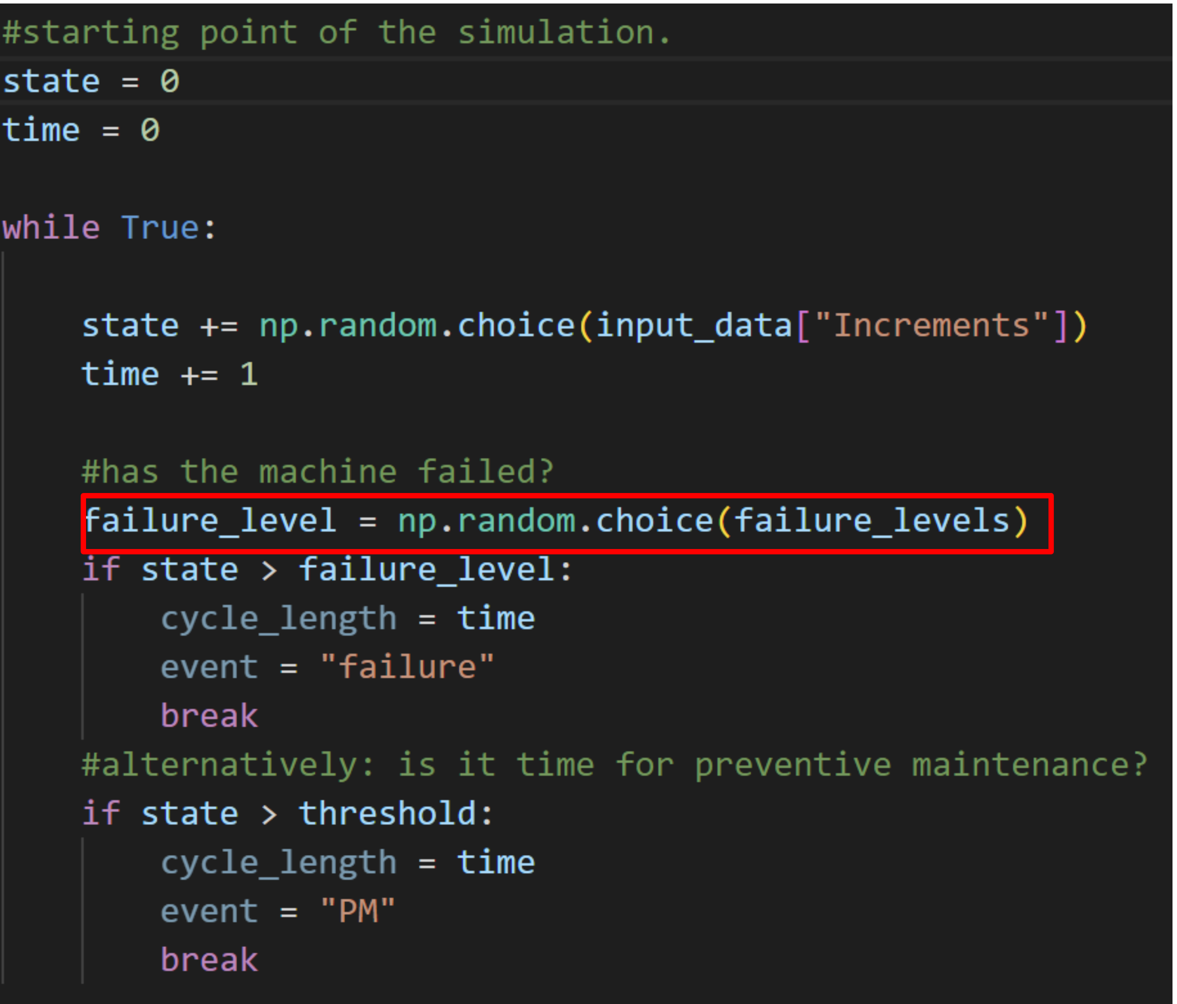
Uncertain failure levels

But, there is a problem!

How can we differentiate between historical preventive and corrective maintenance actions if this is the case?

We require a dataset with *only* failures, or we need additional info.

Code



```
#starting point of the simulation.
state = 0
time = 0

while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    failure_level = np.random.choice(failure_levels)
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

Will this work?

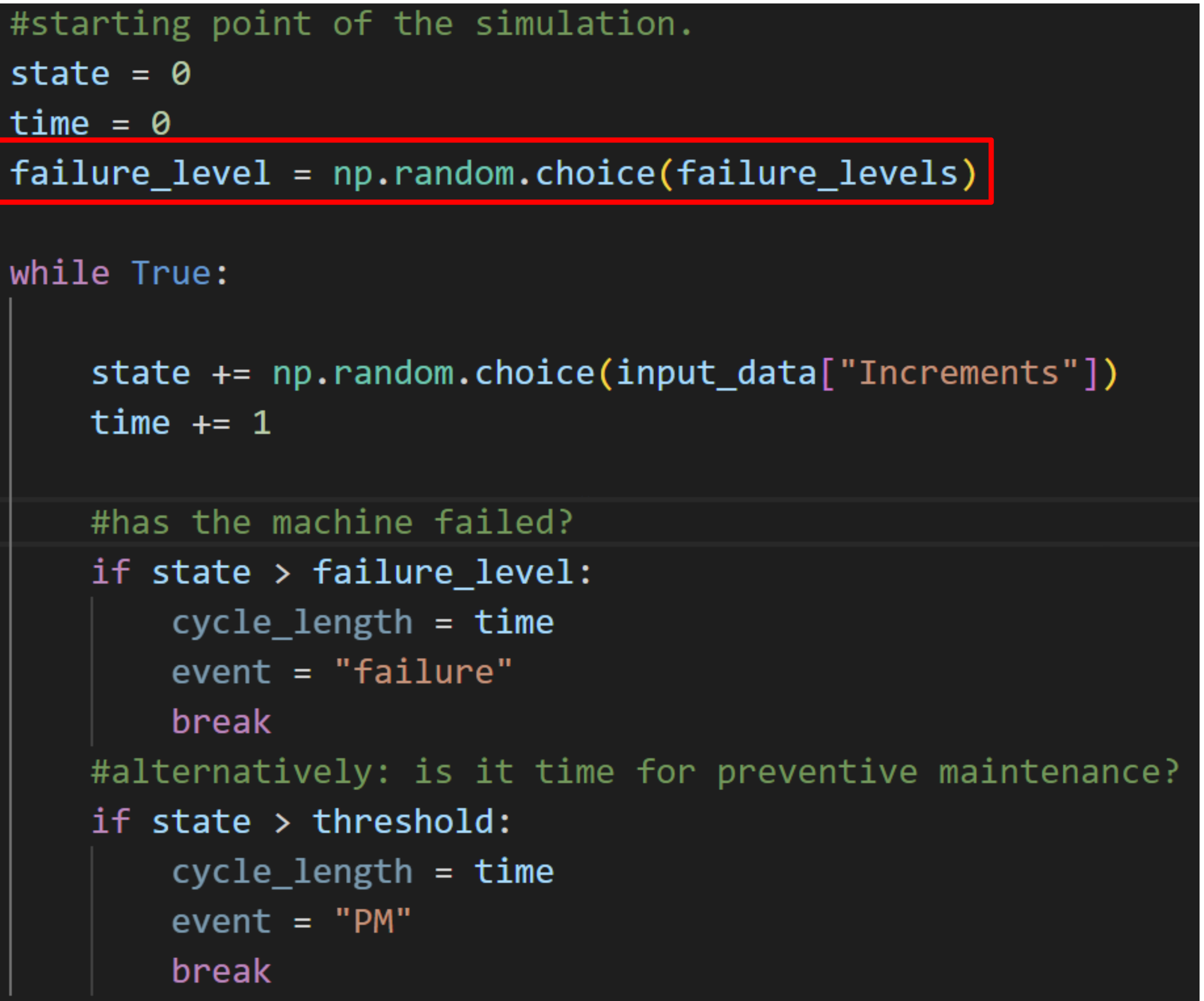
Code

Suppose we are in state 8, and we randomly select a failure level of 9. -> No problem.

Then, our state is increased to 8.1, and we randomly select a failure level of 7 -> Suddenly we have a failure.

We should therefore only select a failure level once per simulation.

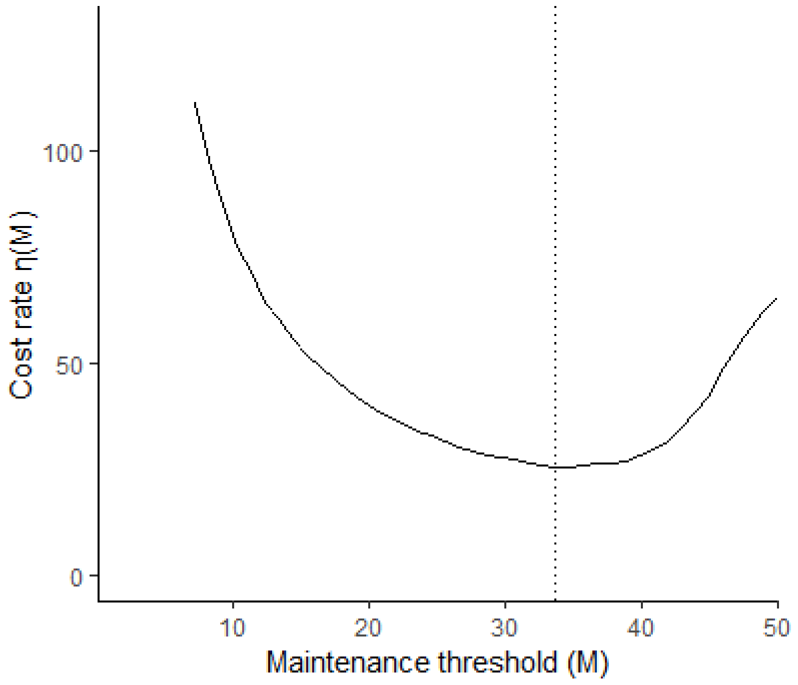
Code



```
#starting point of the simulation.
state = 0
time = 0
failure_level = np.random.choice(failure_levels)

while True:

    state += np.random.choice(input_data["Increments"])
    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

Now, the failure level is randomly selected at the start of a path

Impact of uncertain failure levels



Example:

In the same example, having the failure level fluctuate up and down by 10%:

η(M) = 23.36 -> η(M) = 25.59
M = 37.78 -> M = 33.71

Here, the impact on our policy is quite large.

Impact of uncertain failure levels

Takeaways:

› Uncertainty in failure levels increases the cost rate.

› Maintenance will need to be planned earlier to accommodate for uncertainty.

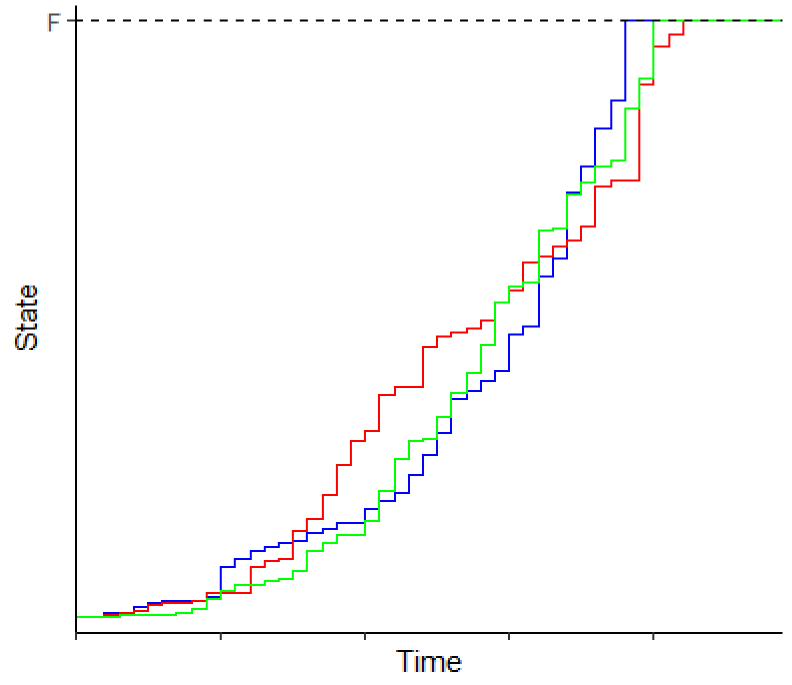
But, uncertain failure levels may suggest that our information on deterioration is incomplete:

› Sensors may not accurately predict the state.

› Machine context may explain earlier failures.

Nonstationary deterioration

Deterioration does not need to happen at a constant speed



We can see that this is the case, by watching the slope of the deterioration paths.

Nonstationary deterioration

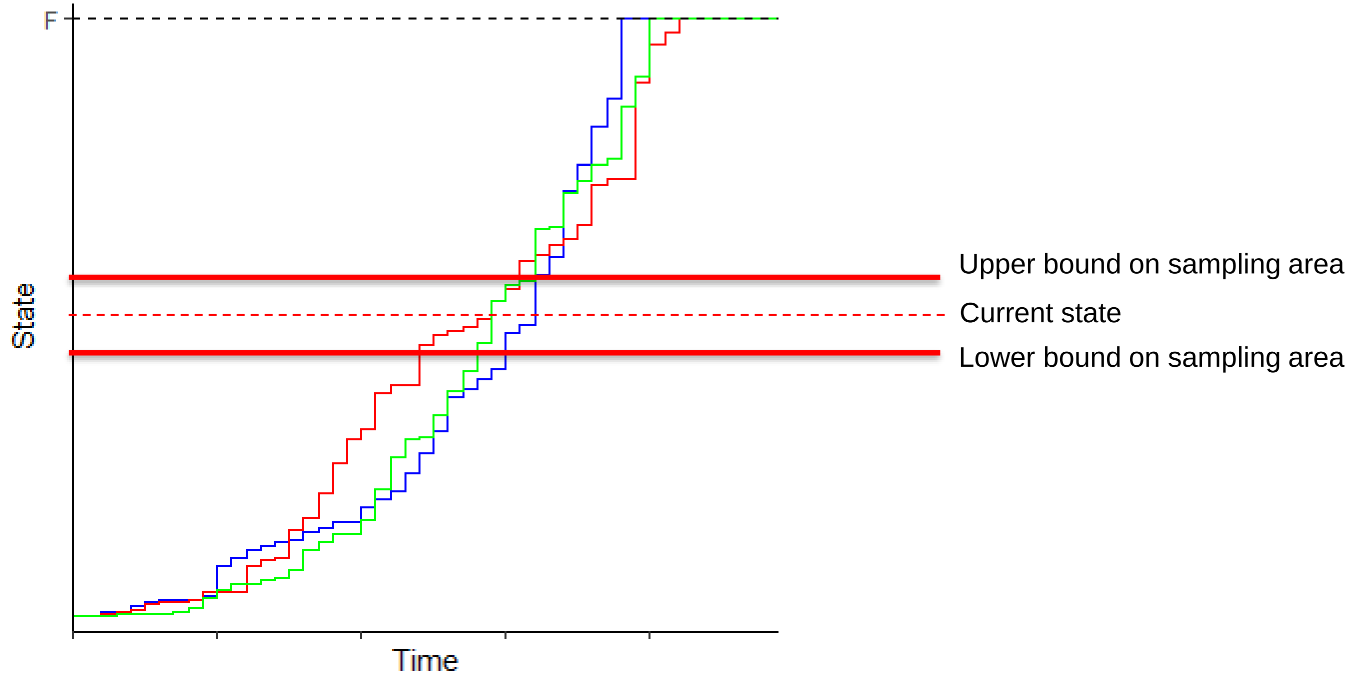
This is a problem, because we simulate by randomly selecting increments.

We did not care about when the increments happened. We would pick large jumps that often happen at the end of the lifetime for a new machine.

So, our way of selecting increments needs to change.

Sampling nonstationary deterioration

Idea: We only pick increments that happened around the current state.



Sampling nonstationary deterioration

At this point, getting enough data becomes a massive issue.

Normally, 10 deterioration paths with ~50 increments each gives us 500 increments to sample from.

Now, we can only use a subset of these increments.

Trade-off: A smaller sampling region will be more accurate for simulating the trend, but a larger sampling region will let us use more data.

Sampling nonstationary deterioration

Also, we need to consider how (computationally) difficult it is to get the increments from a region.

Example: We are in state 10. We sample from states 9 to 11. Then, we need to:

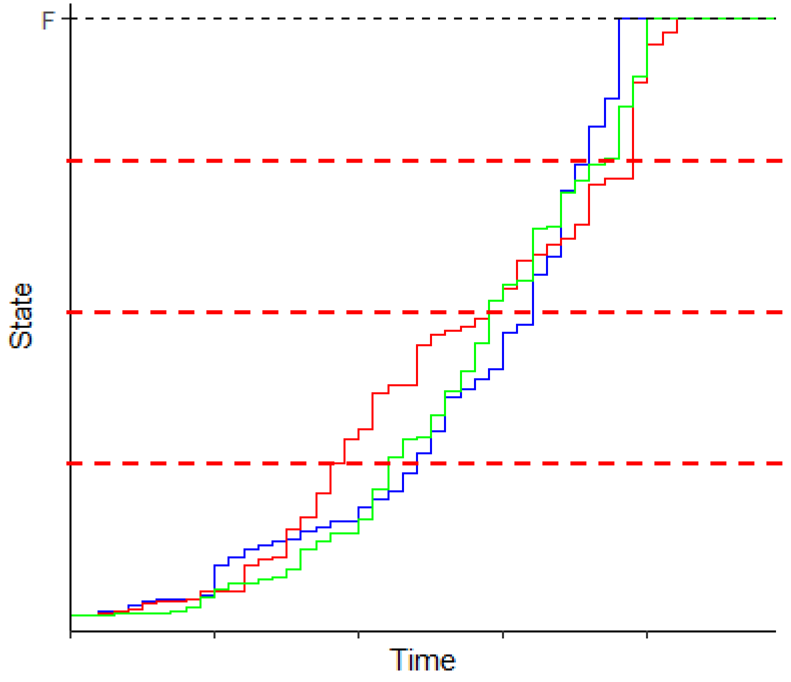
1. Go through our ENTIRE dataset to find which increments happened between states 9 and 11.
2. Store these increments somewhere else.
3. Sample from this subset

And we need to do this thousands or millions of times.

Sampling nonstationary deterioration

We want an approach that is reasonable, but requires as few calculations as possible during simulation.

New idea: Split the states into several zones:



We can split our dataset into these zones in the data preparation.

Then, all we need to do when simulating is check the zone of our current state!

Sampling nonstationary deterioration

Setting the zones still comes with a trade-off, as more zones result in a better trend but less data per zone.

Example: F = 10.

An easy way to set zones is to round to the nearest integer.
Zone 1: 0 – 0.499
Zone 2: 0.5 – 1.499
Zone 3: 1.5 – 2.499
Etc.

Sampling nonstationary deterioration

Setting the zones still comes with a trade-off, as more zones result in a better trend but less data per zone.

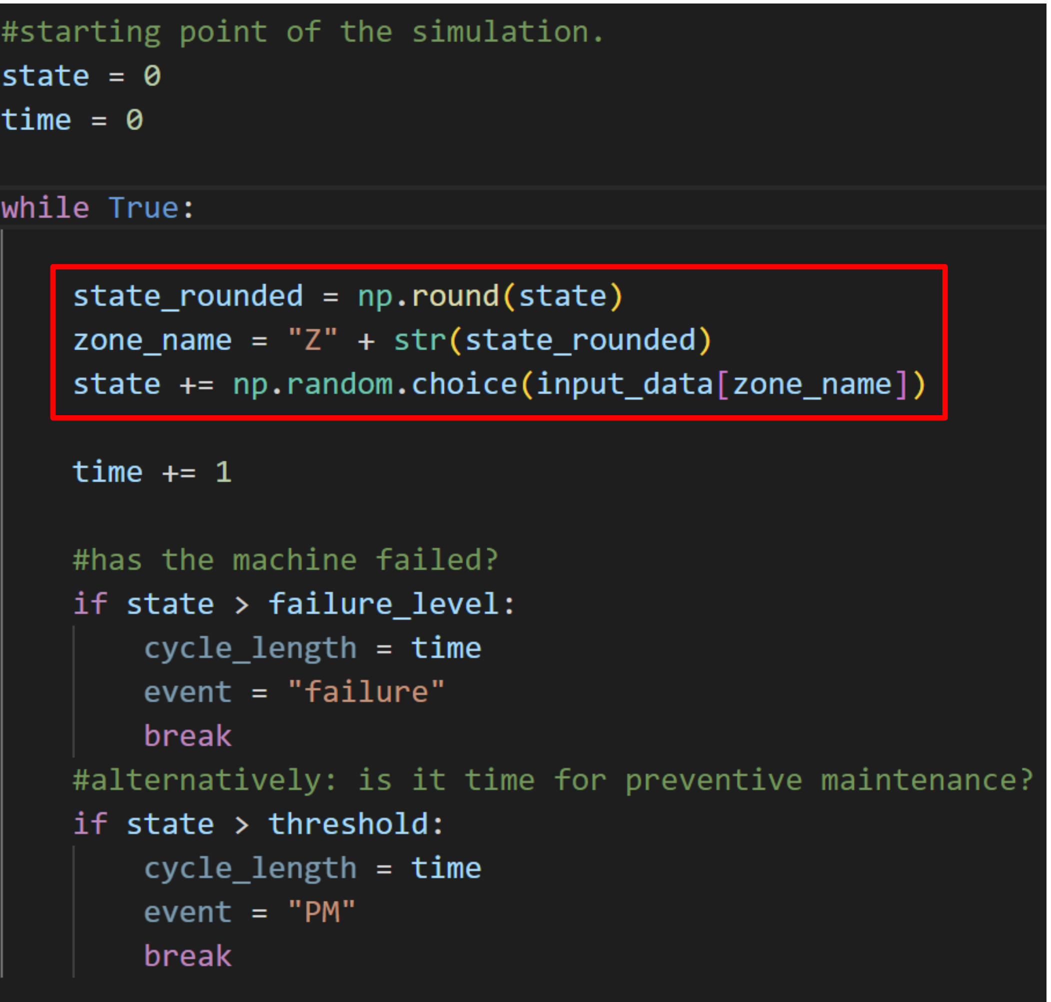
Example: F = 10.

Time	State	Increment
0	0	
1	1.4	1.4
2	3.2	1.8
3	3.4	0.2
⋮		
7	9.6	2.1
8	10	0.4
8	0	-10
9	1.2	1.2
⋮		

Increments			
Zone 1	Zone 2	Zone 3	. . .
0.2	0.5	1.3	
1.4	0.8	2.4	
0.5	1.8	2.0	
. . .			

The result is a table with increments per zone

Code

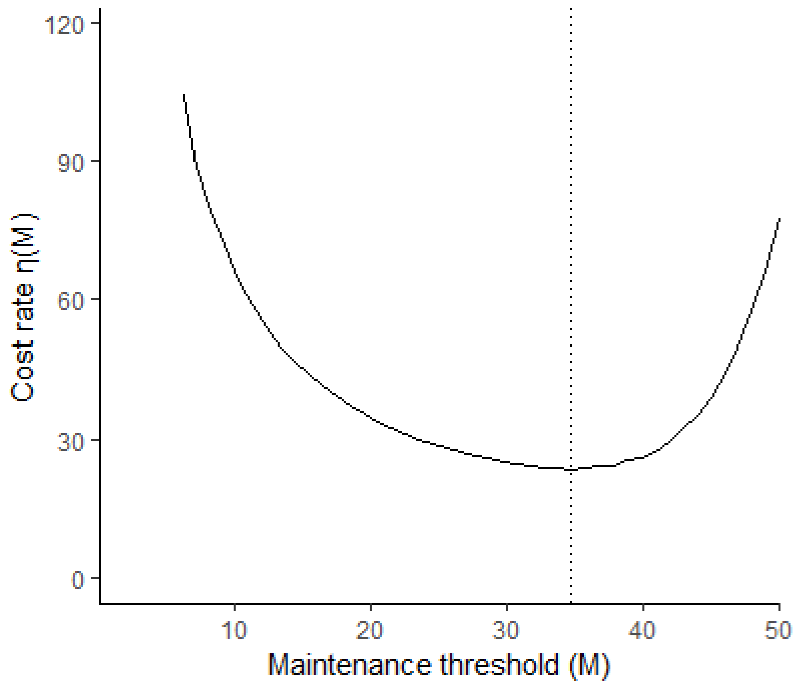


```
#starting point of the simulation.
state = 0
time = 0

while True:

    state_rounded = np.round(state)
    zone_name = "Z" + str(state_rounded)
    state += np.random.choice(input_data[zone_name])

    time += 1

    #has the machine failed?
    if state > failure_level:
        cycle_length = time
        event = "failure"
        break
    #alternatively: is it time for preventive maintenance?
    if state > threshold:
        cycle_length = time
        event = "PM"
        break
```

Now, getting the correct increments is fast. We round the current state, get the correct column name, and sample.

If we do not round to integers, this will be a bit more complex, but still faster than searching through a large dataset.

Impact of nonstationary deterioration



Example: We take a machine with approximately the same MTBF as before, but deterioration is about 4x faster at the end than start.

η(M) = 23.36 -> η(M) = 23.48
M = 37.78 -> M = 34.72

We maintain a bit earlier, but costs are approximately the same.

Why? The system is more predictable.

Impact of nonstationary deterioration

Takeaways:

› We can use simulation to model more complex deterioration processes.

› This will require more data!

› Since simulation relies on repeating calculations many times, you need to make sure that these calculations are efficient.

As always, remember the context of your data.

› Are we sure that the system is utilized at the same intensity over time?

Conclusions

Simulation is a very powerful tool, and we can use it to model many different systems.

But, before you draw conclusions from your data, make sure that you understand the process behind it!