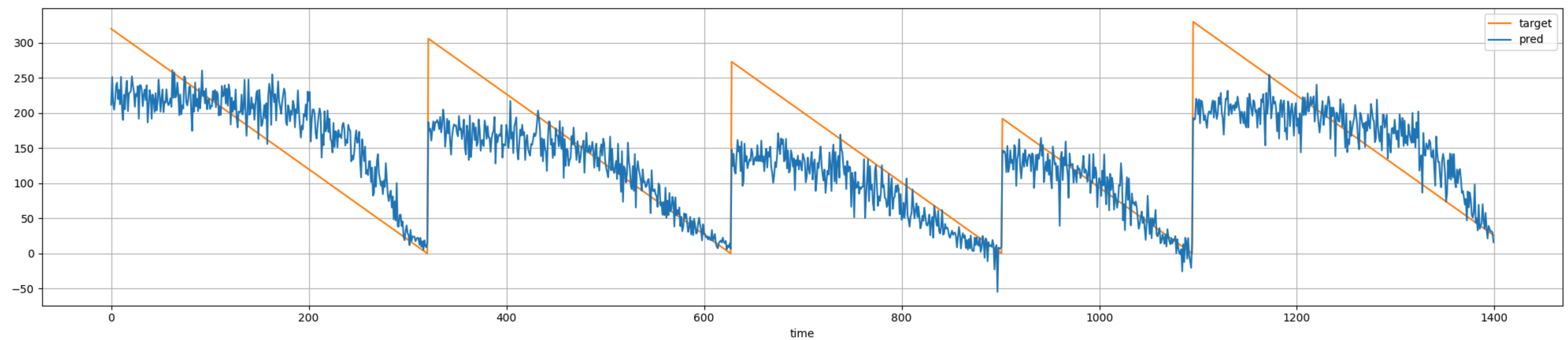
Policy Threshold Calibration

Our Current Situation

The results so far are not comforting

...But it's worth seeing what is going on over time

In [2]:
```
stop = 1400
util.plot_rul(tr_pred[:stop], tr['rul'][:stop], figsize=figsize)
```



...And we get the same shapes also on the validation and test set

...And How to Go Forward

Our goal is not to regress RUL values with high accuracy

...But rather to define a maintenance policy in the form:

$$f(x, w) < \theta \quad \Rightarrow \quad \text{trigger maintenance}$$

- For this, we just need to stop at the right time

Our model...

Makes large estimation errors when the RUL is high

- ...But we do not care about those!

Works reasonably well for low RUL values

- ...I.e. exactly where it matters

Threshold Calibration as an Optimization Problem

Given a RUL estimator

...We can choose when to trigger maintenance by calibrating θ

- This is in fact an(other) optimization problem
- ...And to formulate it we need a cost function

Our cost function will rely on this simplified cost model:

- Whenever an engine operates for a time step, we gain a profit of 1 unit
- A failure costs C units (i.e. the equivalent of C operation days)
- We never trigger maintenance before s time steps (safe interval)

Some comments:

- C is actually an offset over the cost of maintenance
- The last rule mimics using preventive maintenance as a fail-safe mechanism

The Cost Function

Normally, we would determine s and C by talking to a domain expert

...In our case wi well pick reasonable values based on our data

- First, we collect all failure times:

In [3]:
```
tr_failtimes = tr.groupby('machine')['cycle'].max()
```

- Then, we define s and C based on statistics:

In [4]:
```
safe_interval = tr_failtimes.min()
maintenance_cost = tr_failtimes.max()
```

- For the safe interval s, we choose the minimum failure time
- For the maintenance cost C we choose the largest failure time

We are taling about jet engines, so failing is BAD

Solving the Calibration Problem

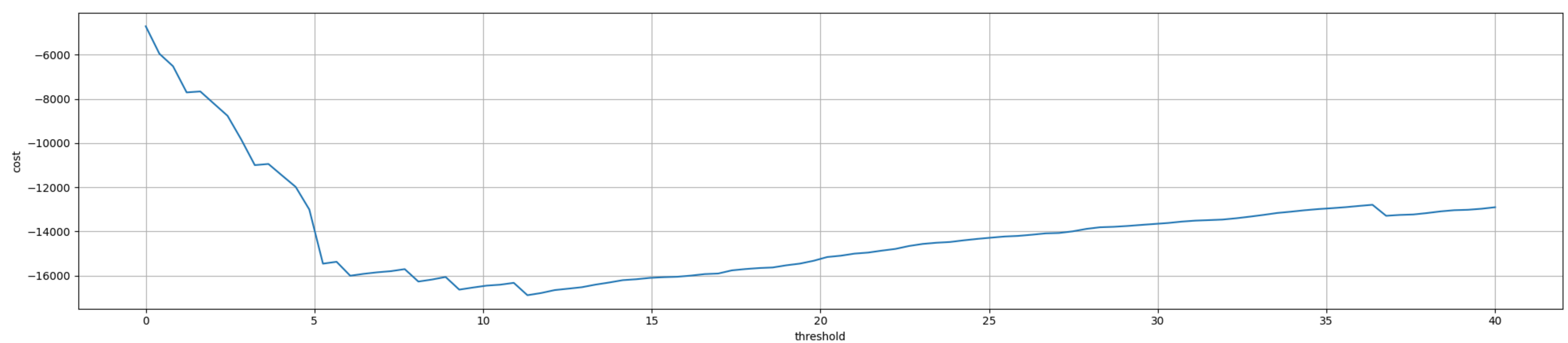
We can sample a range of values for the θ parameter

...Then simply pick the value with the smallest cost

- The code in `optimize_threshold` can also plot the corresponding cost surface

In [5]:
```
cmodel = util.RULCostModel(maintenance_cost, safe_interval)
th_range = np.linspace(0, 40, 100)
tr_thr = util.optimize_threshold(tr['machine'].values, tr_pred, th_range, cmodel, plot=True,
print(f'Optimal threshold for the training set: {tr_thr:.2f}')
```

```
Optimal threshold for the training set: 11.31
```



Evaluation

Finally, we can check how we are doing on the test set:

```
In [6]: tr_c, tr_f, tr_sl = cmodel.cost(tr['machine'].values, tr_pred, tr_thr, return_margin=True)
        ts_c, ts_f, ts_sl = cmodel.cost(ts['machine'].values, ts_pred, tr_thr, return_margin=True)
        print(f'Cost: {tr_c/len(tr_mcn):.2f} (training), {ts_c/len(ts_mcn):.2f} (test)')
```

```
Cost: -90.81 (training), -104.38 (test)
```

We can also evaluate the margin for improvement:

```
In [7]: print(f'Avg. fails: {tr_f/len(tr_mcn):.2f} (training), {ts_f/len(ts_mcn):.2f} (test)')
        print(f'Avg. slack: {tr_sl/len(tr_mcn):.2f} (training), {ts_sl/len(ts_mcn):.2f} (test)')
```

```
Avg. fails: 0.01 (training), 0.00 (test)
Avg. slack: 22.35 (training), 19.56 (test)
```

- Slack = distance between when we stop and the failure
- The results are actually quite good and we also generalize fairly well

Some Considerations

In principle, RUL regression is a very hard problem

- Our linearly decreasing RUL assumption is just a rough oversimplification
- ...RUL is inherently subject to stochastisticy
- ...And depends on the how the machine will be used

But we don't care, since RUL prediction was not our true problem

The real problem involved both prediction and optimization

- We had to optimize the NN parameters (to obtain good predictions)
- We had to optimize the threshold

The ultimate goal was to reduce maintenance cost

It's worth to keep in mind the big picture

- In a "predict, then optimize" setting
- ...Quality should be judged on the final cost