

Components Wear

mlumina.components_wear

Симуляция износа и отказов графа взаимозависимых компонентов (по докстрингам и примерам — модель ориентирована на автомобильные узлы, но структура графа предметно-независима). Два класса: `ComponentsGraph` (описание графа и симуляции для многих объектов) и `ComponentsObject` (один конкретный экземпляр системы и его текущий износ).

“ ⚠ **Известный баг:** `ComponentsObject.update_wear()` и `ComponentsObject.predict_fix()` не работают — см. [известные проблемы](#) ниже. `get_probabilities()`, а также все методы `ComponentsGraph` (`simulate_objects*`) — рабочие.

???? ????????????

Граф описывается словарём: узлы (`nodes`) — группы взаимозаменяемых компонентов, каждый со своей функцией вероятности отказа; рёбра (`edges`) — зависимость износа одного узла от состояния другого.

```
{
  "A": {
    "edges": { "B": { "bend": 0.4, "koef": 2.3 } },
    "nodes": {
      "A1": {"func": "weibull_failure_prob", "a": 1.0, "maximum": 100000},
      "A2": {"func": "weibull_failure_prob", "a": 1.5, "maximum": 100000}
    }
  },
  "B": {
    "edges": {},
    "nodes": {
      "B1": {"func": "weibull_failure_prob", "a": 1.0, "maximum": 100000}
    }
  }
}
```

```
}  
}
```

- `"func"` — ключ в `mlumina.components_wear.functions.utils_functions_mapping` (сейчас единственная зарегистрированная функция — `"weibull_failure_prob"`, вычисляет CDF распределения Вейбулла по параметрам `a` (форма) и `maximum` (износ, при котором вероятность отказа ≈ 0.999)).
- `edges[parent] = {"bend": ..., "koef": ...}` — до достижения родителем износа `bend * maximum` дочерний узел изнашивается с обычной скоростью (коэффициент 1); после достижения родителем своего `maximum` — со скоростью `koef`; между этими точками — линейная интерполяция.

```
import json  
from mlumina.components_wear import ComponentsGraph  
  
with open("examples/components_wear/data/graph.json", encoding="utf-8") as f:  
    graph_dict = json.load(f)  
  
graph = ComponentsGraph(graph_dict)
```

????????? ????? ???? ??????

```
df = graph.simulate_objects(n_objects=100, total_wearout=100_000, step=1000, seed=42,  
    fix_coeff=1.0)
```

Для каждого из `n_objects` объектов случайно выбирается по одному компоненту из каждого узла (начальный износ — усечённое экспоненциальное распределение), затем пошагово (`step`) наращивается износ с учётом зависимостей, на каждом шаге проверяется отказ (по вероятности из `func`) и «тихий» ремонт (та же вероятность, умноженная на `fix_coeff`). Возвращает `pandas.DataFrame` с колонками `failed_node`, `wear`, `all_parts` — по одной строке на каждый зафиксированный отказ.

`simulate_objects_ray(...)` — та же логика, но каждый объект симулируется в отдельной Ray-задаче (`@ray.remote`). Требуется `ray.init()` заранее, иначе `RuntimeError`.

`simulate_objects_bn(...)` — экспериментальный вариант: вероятность отказа считается как $0.6 * P_{\text{weibull}} + 0.4 * P_{\text{bn}}$, где `P_bn` — упрощённая байесовская оценка на основе дискретизации износа в 5 бинов (`b0...b4`) для узла и его непосредственного родителя. Возвращает те же колонки плюс `prob_weibull`, `prob_bn`, `prob_combined`.

`create_random_object()` -> `ComponentsObject` — генерирует один случайный объект (без симуляции износа во времени).

??????? ???????? (ComponentsObject)

```
from mlumina.components_wear import ComponentsObject
```

```
obj = ComponentsObject([["A1", 0], ["B1", 0]]) # [[имя_компонента, начальный_износ], ...]
```

```
probs = obj.get_probabilities(graph) # {component_name: вероятность_отказа_в_%}
```

`get_probabilities(graph)` — рабочий метод: по каждому компоненту находит его функцию отказа в графе и считает вероятность (в процентах) от текущего износа.

??????????? ???????? update_wear ? predict_fix

`ComponentsObject.update_wear(graph, step=1000)` (`src/mlumina/components_wear/components_object.py:163`) и `ComponentsObject.predict_fix(...)` (там же, строка 272) вызывают `self._get_wear_dict()`. В классе определён метод `_get__dict__()` (строка 74, с двойным подчёркиванием — похоже на опечатку), а не `_get_wear_dict()`. Метода `_get_wear_dict` в классе нет — оба вызова гарантированно падают с `AttributeError: 'ComponentsObject' object has no attribute '_get_wear_dict'`.

Это воспроизводится и в штатном примере `examples/components_wear/example.py:33` (`component_object.update_wear(graph, 10000)`), т.е. пример из репозитория в текущем виде не отрабатывает до конца.

Если нужно пользоваться этими методами — до исправления в коде их можно обойти монки-патчем:

```
from mlumina.components_wear.components_object import ComponentsObject
ComponentsObject._get_wear_dict = ComponentsObject._get__dict__
```

??????????? ???????? (functions.py)

```
from mlumina.components_wear.functions import weibull, utils_functions_mapping
```

`weibull(wear, a, maximum) -> float` — CDF Вейбулла, масштаб подобран так, что `weibull(maximum, a, maximum) ≈ 0.999`. `utils_functions_mapping = {"weibull_failure_prob": weibull}` — реестр функций, на который ссылаются графы по строковому ключу `"func"`. Чтобы добавить свою функцию отказа, нужно дополнить этот словарь (единой точки регистрации "снаружи" пакета сейчас нет — граф всегда резолвит `"func"` через этот фиксированный маппинг).

Revision #1

Created 2026-09-23 06:40:09 UTC by Маслов Алексей

Updated 2026-09-23 06:40:10 UTC by Маслов Алексей