NUR Hand-in 1

Lucas Pouw

February 24, 2025

1 Utilities

Short helper functions are grouped in a separate utilities file, `utils.py`:

2 Interpolator class

Whenever interpolation is necessary, we use the following class in `interpolator.py`:

3 Matrix class

All matrix computations and routines are done through the following class in `matrix.py`:

4 Exercise 1: Poisson distribution

```python
import numpy as np
from utils import cumsum


def log_factorial(array):
    assert np.sum(array < 0) == 0, "Input should be greater than or equal than 0."

    array = np.array(array).astype(np.int32)  # Force list to array of integers
    max_idx = np.max(array) + 1

    all_factorials = np.zeros(max_idx, dtype=np.float32)
    all_factorials[1:] = cumsum( np.log(np.arange(1, max_idx)) )  # nth element contains log(n!)
    return all_factorials[array]  # Requested factorials


def poisson(lam, k):
    lam = np.array(lam).astype(np.float32)
    k = np.array(k).astype(np.int32)
    log_poisson = k * np.log(lam).astype(np.float32) - lam - log_factorial(k)
    return np.exp(log_poisson.astype(np.float32))


if __name__ == '__main__':

    lambdas = np.array([1, 5, 3, 2.6, 100, 101])
    ks = np.array([0, 10, 21, 40, 5, 200])
    results = poisson(lambdas, ks)
    for i, result in enumerate(results):
        print(f'$(\\lambda, k)$ = ({lambdas[i]}, {ks[i]}) -> $P_{{\\lambda}}(k)$ = {results[i]:.6e}')
```

poisson.py

```
$(\lambda, k)$ = (1.0, 0) -> $P_{\lambda}(k)$ = 3.678794e-01
$(\lambda, k)$ = (5.0, 10) -> $P_{\lambda}(k)$ = 1.813279e-02
$(\lambda, k)$ = (3.0, 21) -> $P_{\lambda}(k)$ = 1.019340e-11
$(\lambda, k)$ = (2.6, 40) -> $P_{\lambda}(k)$ = 3.615103e-33
$(\lambda, k)$ = (100.0, 5) -> $P_{\lambda}(k)$ = 3.100058e-36
$(\lambda, k)$ = (101.0, 200) -> $P_{\lambda}(k)$ = 1.269495e-18
```

poisson_output.txt

5 Exercise 2: Vandermonde matrix

Finally, we compare the runtime of each method with the `timeit` module. This is done with the following code in `timing.py`: