Dreaming Spires

Optimising the Epioncho Model

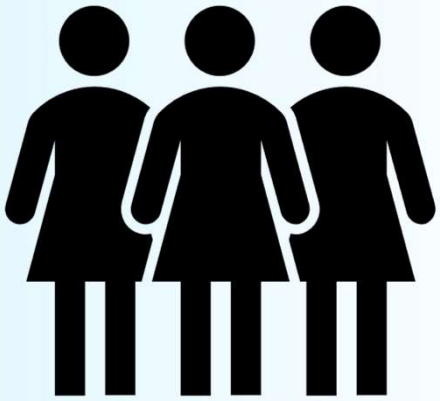
Mark Todd, CEO Dreaming Spires

Optimising the Epioncho Model

- Brief description of the model
- How we assess sections of the model for optimisation
- Optimisations:
  - Optimisation 1: Array Operations
  - Optimisation 2: Fast Binomial
  - Optimisation 3: Worm Incubation
- Code Clarity
- Results

Overview of onchocerciasis population-based model

The people in the model are defined



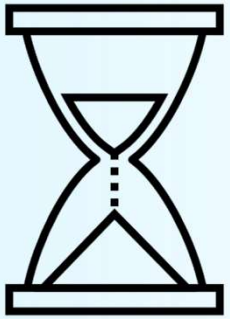
People have “properties”:

Age
Gender
Individual Exposure

People are modelled as each containing:

Blackflies
Worms
Microfilaria

Time moves forward



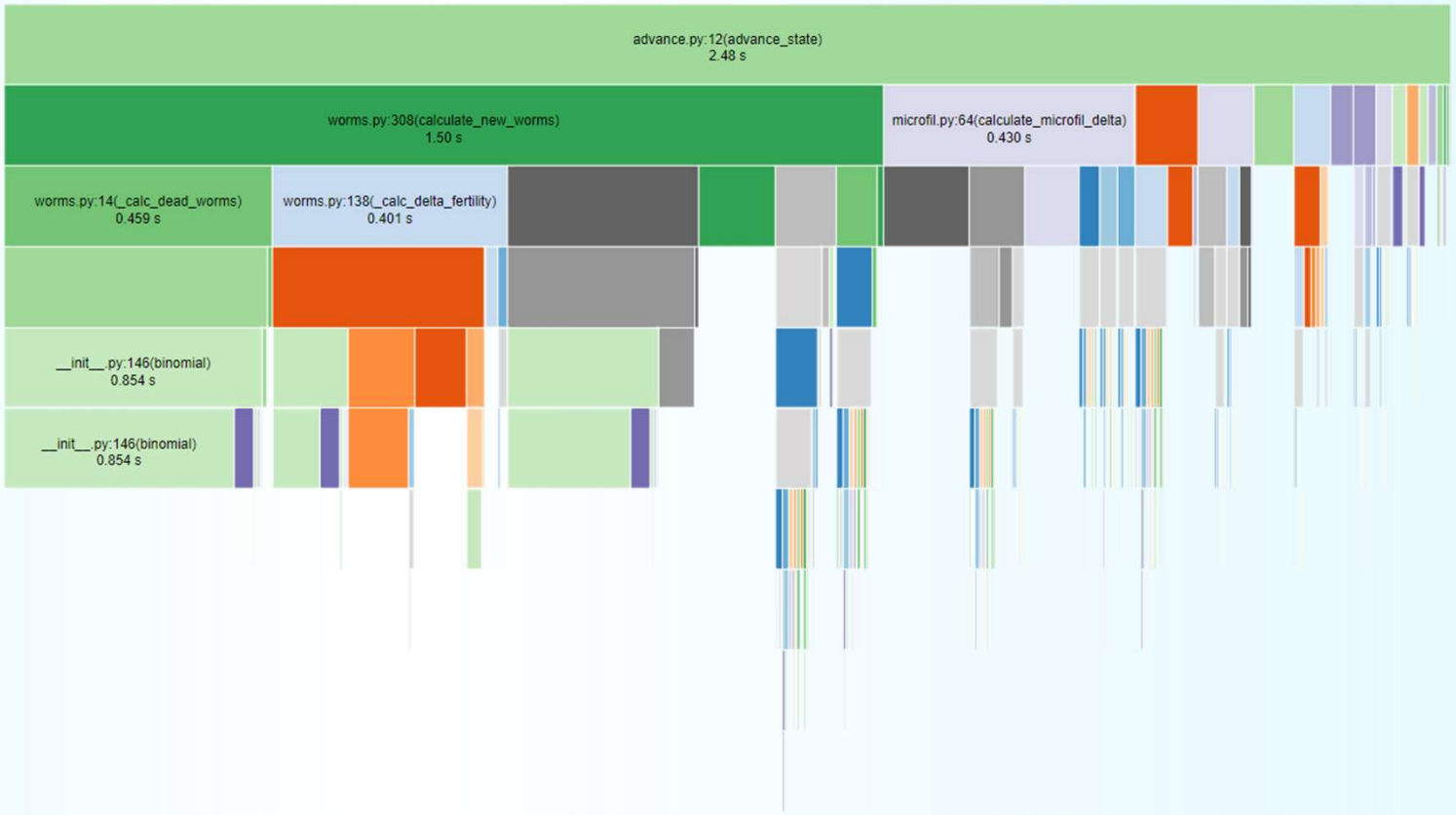
- Worms grow, are born and die
- People are born and die
- Treatment is performed
- Worms have an incubation time

Optimisation

Assessment

The profiler allowed us to:

- Identify slow sections of code
- Choose which functions to prioritise for optimisation
- Confirm speed up after changes



Iteration Methods

- Removal of for loop iterations, in exchange for numpy level vectorisation.

- Results in both clarity increase & speed up

Before:

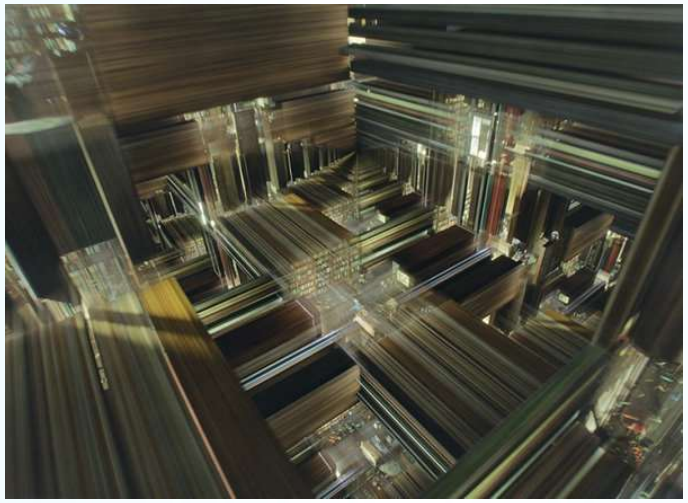
```
array = np.array([1,2,3])
new_list = []
for i in array:
   new_i = 2*i
   new_list.append(new_i)
new_array = np.array(new_list)
```

After:

```
array = np.array([1,2,3])
new_array = 2*array
```

Why not start with that?

- Major use case was operations over worm compartment
- The worm compartment calculation is very complex
- Each part of the calculation has to be converted to this form
- 3D/4D arrays are required

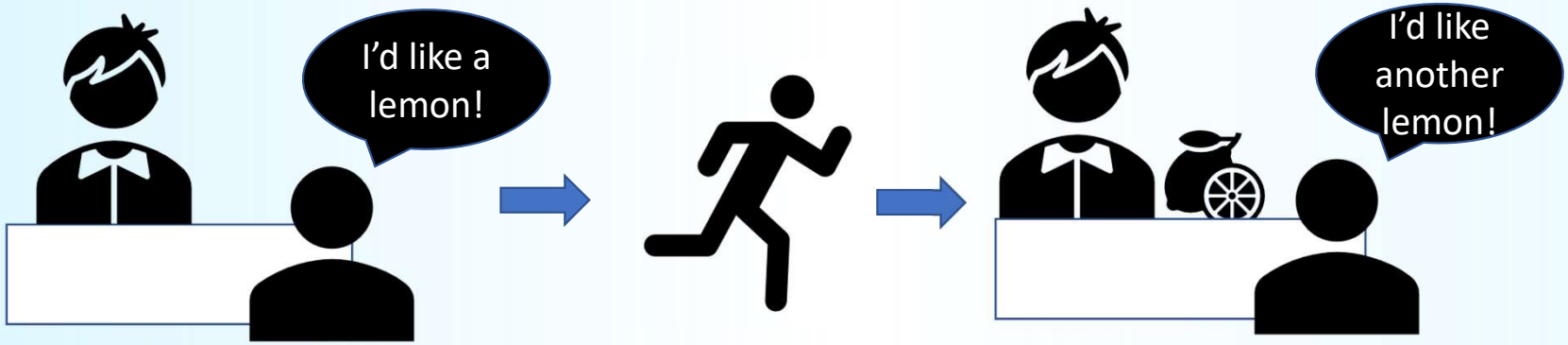


Binomials

- After the iteration method optimisation, the next slowest function was the sampling of binomial distributions.
- This proved challenging – the binomials were being generated by a part of the “numpy” library, which already uses C++.
- This is a widely used library in python for dealing with array structures.
- To make the model faster, we would have to beat “numpy”.

Fast Binomial

- Step 1: Generate binomial values in bulk
  By requesting more values at once it saves time on memory allocation



- Step 2: Python is slow – time for C++

- Step 3: Connect it back into python with bindings

Worm incubation time data storage

- Worm incubation time means we have to store values in a large array – prior to optimisation it worked as below:

Before

After

Other Improvements

- Pytest-Trust-Random – Code changes are now identifiable
- Code clarity – Bugs are easier to spot, and functions are readable by people who may not be familiar with the code
- Endgame Simulations – Disease models with fixed delta time can now share code for how to run a model.
  - In future this could allow for shorter and more readable code bases, as well as more code re-use between projects.

Code Clarity

Clear Function Names

Clear Variable Names

Type hints

Docstrings

```
def _calc_outbound_worms(
    current_worms: WormGroup,
    worm_age_rate_generator: Generator,
    dead_worms: WormGroup,
) -> WormGroup:
    """
    Calculates the number of worms leaving each compartment due to aging.

    Args:
        current_worms (WormGroup): The current number of worms
        worm_age_rate_generator (Generator): Generates worms at a pre-defined rate.
            The rate at which worms move from one compartment to the next
        dead_worms (WormGroup): Worms dying in each compartment

    Returns:
        WormGroup: The number of worms leaving each compartment due to aging.
    """
```

Results

- Order of magnitude speed up from original code
- Higher code readability
- More re-usable tools like Pytest Trust Random, Endgame Simulations

Any Questions?