Scientific Computing Project 4a

Jakob Schauser, pwn274

October 2023

1 The simulation in general

Being unsure as to how much of my code you want to see, I will mainly be showing the architecture i ended up creating.

I have a large dictionary called `parameters`. Using this I can get the differential equations I have simply copied from the assignment:

```
def get_equations(parameters):
    def dx1_dt(x1, x2):
        # defining the first differential equation

    def dx2_dt(x1, x2, y):
        # defining the second differential equation

    def dy_dt(x2, y, z):
        # defining the third differential equation

    def dz_dt(y, z):
        # Example:
        return d1*y*(r-z) + (e*(r-z) if with_transfusion else 0) - (r4*z if with_death else 0)

    return dx1_dt, dx2_dt, dy_dt, dz_dt
```

These equations can then be used in a second Closure that defines a single integration-step:

```
def get_stepfn(parameters, dt, mode):
    assert mode in ['euler', 'rk4'], 'mode must be either euler or rk4' # I love strings

    dx1_dt, dx2_dt, dy_dt, dz_dt = get_equations(parameters)

    def step_euler(x1, x2, y, z):
        # defining the euler steps
        return x1_new, x2_new, y_new, z_new

    def step_rk4(x1, x2, y, z):
        # defining the Runge-Kutta steps
        return x1_new, x2_new, y_new, z_new

    return step_euler if mode == 'euler' else step_rk4
```

Having a function for the single step, I can define a simulation loop

```
def do_simulation(stepfn, x1, x2, y, z, total_time, mode, dt):
    total_steps = int(total_time//dt + 1)
    data = np.empty((total_steps, 4))

    for i in range(total_steps):
        x1, x2, y, z = stepfn(x1, x2, y, z)
        data[i] = [x1, x2, y, z]

    return data
```

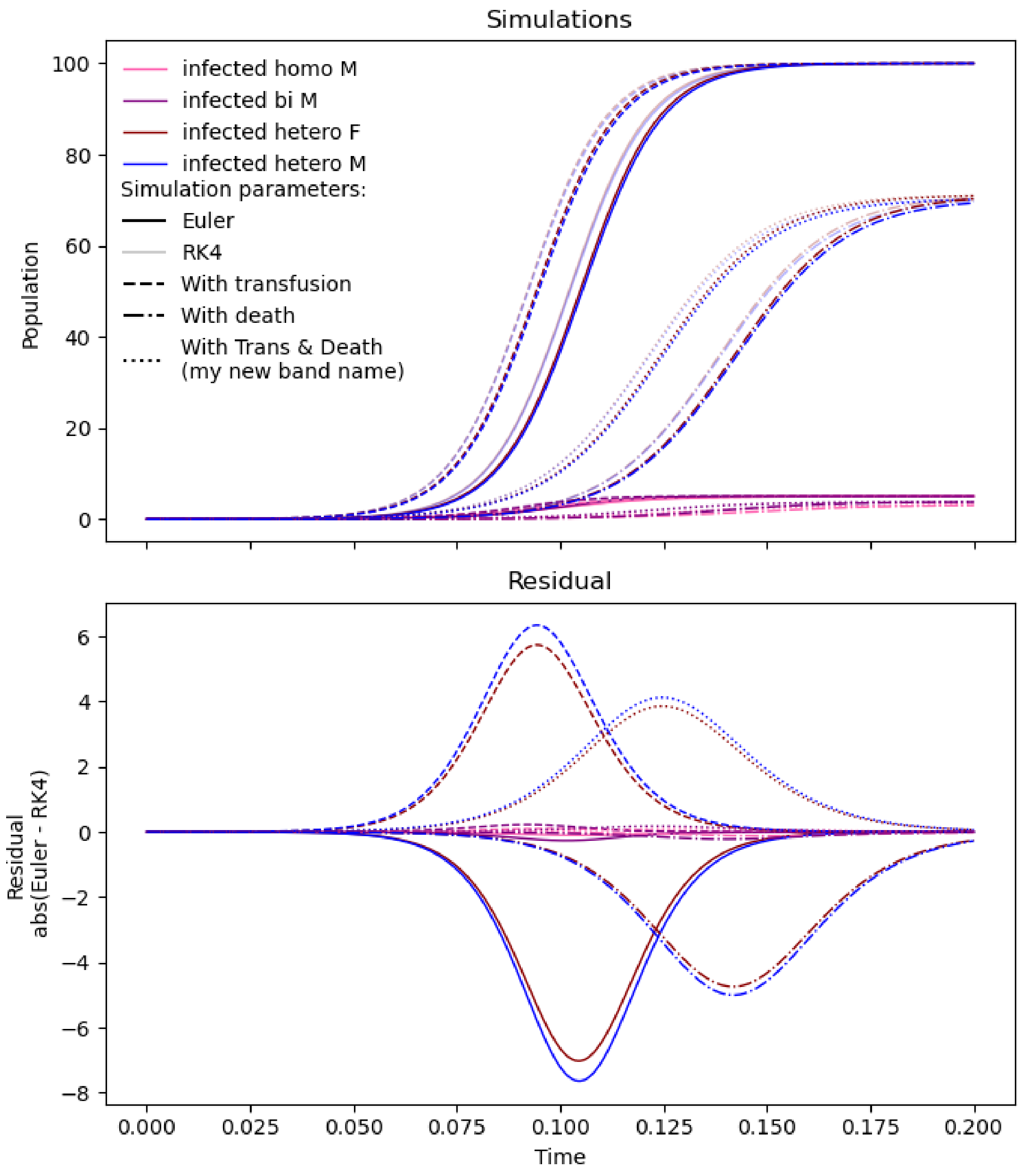
Calling both like this.

```
def simulate(parameters, x1, x2, y, z, total_time, mode = 'euler', dt = 0.001):
    stepfn = get_stepfn(parameters, dt, mode)
    data = do_simulation(stepfn, x1, x2, y, z, total_time, dt)
    return data
```

I am certain you are as tired of looking at my code as I am, so lets go look at the results and plots.

2 Comparison of different methods and parameters

Having implemented the integration methods and the given differential equations, I could simulate using all the given parameters and possible variations. This gives me the following, slightly unhinged plot, where colour, line-style and alpha value all have a meaning:



What we can gather: With Runge-Kutta simulation, given the same simulation parameters, the spread of the HIV-infection starts slightly earlier. This can be seen form the lower alpha lines in the first sub-plot. I expect this comes from the exponential nature of the spread, as the two integration-methods seems to agree when the slope is smaller.

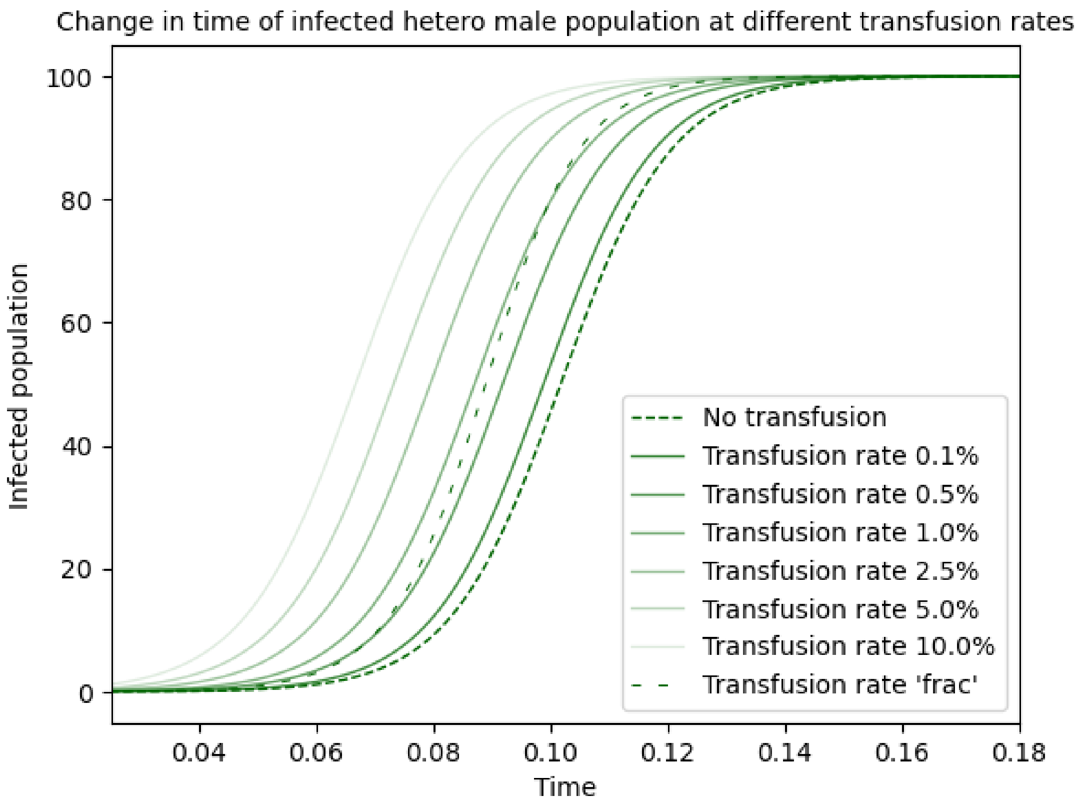
We can also see, that transfusion allows for a quicker spread of the disease, both with and without simulating death. This can be seen, as the dashed line (with transfusions) rises earlier than the full line (no transfusions) and the dotted line (with transfusions and death) rises quicker than the dash-dotted line (only simulating death).

Simulating death by AIDS does what one would expect: The total infected population drops across the board, while also delaying the spread of the disease. Given a high enough death rate, it seems like the HIV will plateau and not spread to every adult.

Important note: For the residual plot, the absolute value is plottet. I have chosen to move two of the residuals into the negative relm for readability! Analyzing it, I gather, that it is not as much a residual between simulation and ground-truth, as it is the system of equations being very sensitive in time to different ways of solving them. The residual is also not scaled with population size. To see the "quality" difference between the two integration-methods, you will have to read on and wait (or scroll down to section 4). This plot does not say much (as it is not

3 The influence of blood transfusions

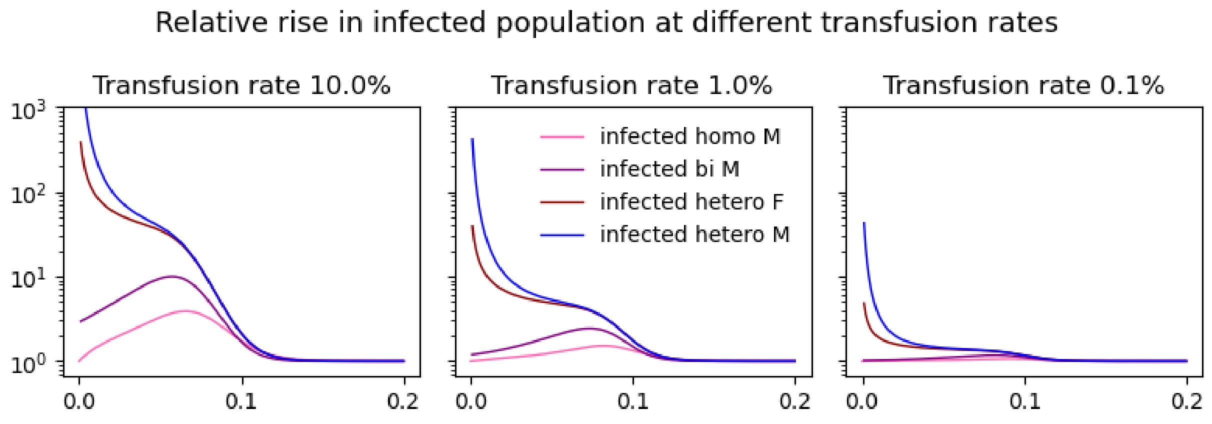
The parameter e in the set of equations govern the transfusion rate between the different parts of the population.



What we can gather: Raising the transfusion rate will allow for the HIV virus to spread from the homosexual community to the hetero community in a much quicker way, as it will now not only go through the bi people in our model. Having the transfusion rate depend on the fraction sick/healthy people in the population, we get a steeper, but delayed, increase, corresponding to the blood bank having to first recieve a critical amount of infected blood.[1]

[1] I had trouble seeing the change from e_{frac}, so I ended up using $e = 10 \cdot frac$, ie. scaling the effect up by a factor 3

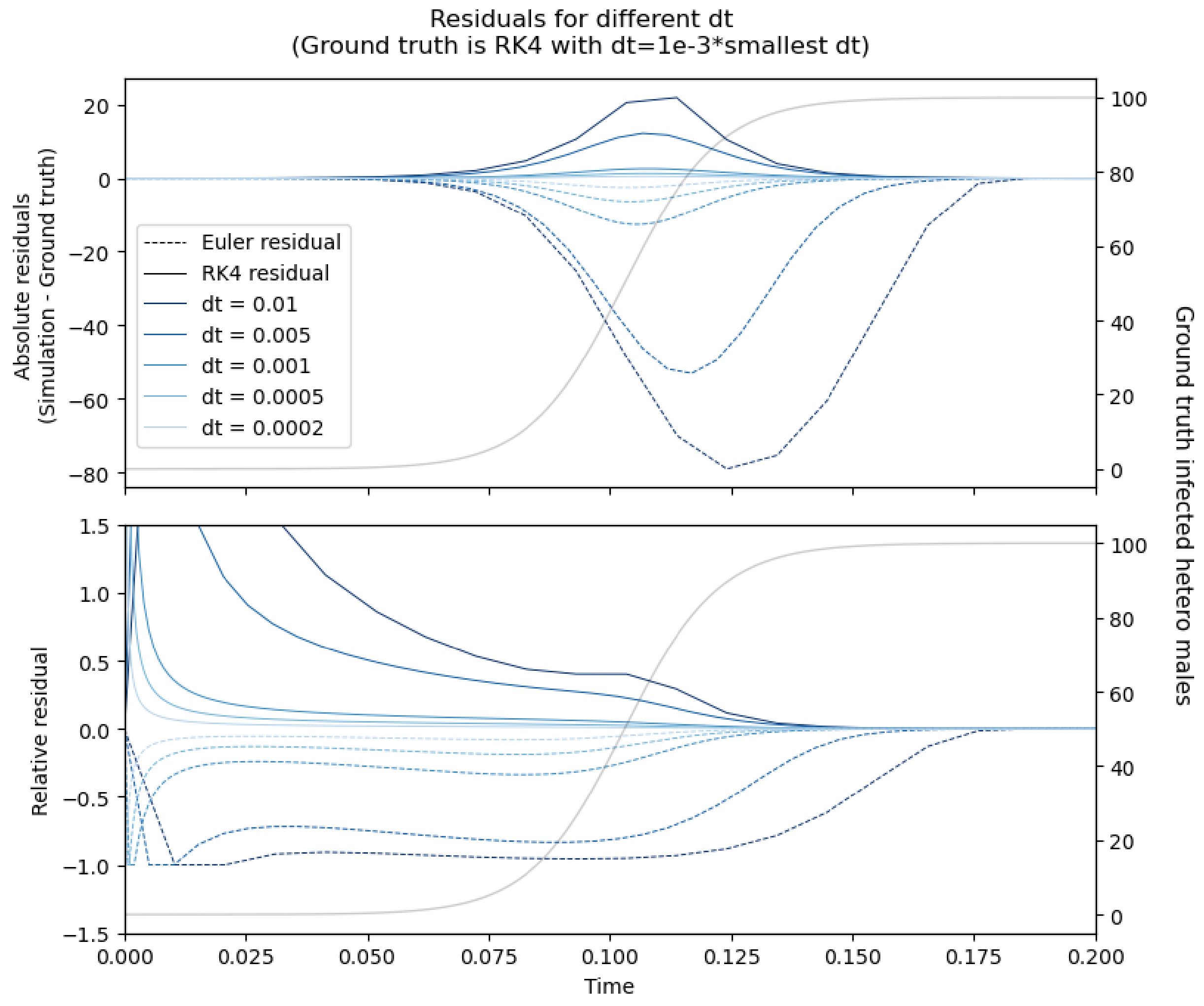
To not only look at heterosexual males (they get enough screen time already), I have also plotted the relative increase in infected number of people for all the different population groups:



Here it can be seen that, as would be expected, the part of the population that does not directly have sex with the disease-originating homosexual males by far have the highest relative increase in incidents, while the more natural bearers of HIV only have a slightly higher (in relative terms) risk of contracting AIDS.

4 The importance of dt for different numerical integration algorithms

Now this is what we are really after: Seeing the difference of switching from Euler-integration to Runge-Kutta of fourth order. For heterosexual males, I get the following:



As can be seen, once this third, overly complicated plot has been parsed, is the following:

Looking at the top plot we see, that Euler in absolute terms are missing the mark on the rising part of the infected population, with Runge-Kutta always being better. A fun thing is, that Euler integration seems to be delayed (as can be seen from the residual being negative) while Runge-Kutta is early.

This made me realize, that looking at the absolute residual might not carry a lot of meaning, as the value we are simulating is spanning multiple orders of magnitude. This leads us to the bottom plot, where we can see that Euler integration us worse for most of the simulation, but, surpirisingly, better than Runge-Kutta for the first 25% of the run when using a too large step size.[2] At reasonable step sizes, fourth order Runge-Kutta is the clear winner.

Thanks for coming to my TED Talk.

[2] As it has a relative residual of magnitude 1, I think it means that the model is predicting 0, giving $(pred - truth)/pred = -1$ whereas RK is calculating a too high number of infected.