FYTN03: Computational Physics
Module A Project

Author: Joseph Alexander Binns.

Due: 19th September 2021.

1 Introduction

The main purpose of this project was to explore and experiment with a variety of models involved on the topic of disease spreading. As the starting point of the mathematical models focused on how the infected and susceptible population **change** over time, the equations would need to be solved in order to yield the number of infected at a given time. As this is a computational physics course, I was directed to computationally solve the equations! Computational solutions are often practical, but introduce new sources of errors. And as such the project also served as a lesson on error estimation, as well as error reduction through improved techniques and solvers. Further improvements were made to the models, such as the introduction of vaccinations and the movement of populations between discrete locations. Once the appropriate toolkit had been assimilated, an appropriate model for a chosen case-study disease outbreak was simulated: HIV in western Africa in the middle of the 20th century.

2 Theory

2.1 Ordinary Differential Equations (ODEs)

An ODE is an equation relating a derivative to it's respective function. The function must contain only one independent variable for it to be called ordinary.

2.2 ODE solvers

ODEs can be solved analytically and computationally. Computational solutions are often more practical, but come with the expense of the introduction of computational error to the result. A few computational ODE solvers follow.

2.2.1 Euler method

The Euler method works by taking the gradient at a point, and then using that to approximate the solution after a small step. This procedure is repeated iteratively for as long as many steps as desired.

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{1}$$

where h is the step size and $f(t_n, y_n)$ is the derivative function at time t_n and height y_n.

2.2.2 Runge-Kutta method (RK)

The Runge-Kutta method works by calculating various gradients, and taking the weighted average of them to approximate the solution after a small step. The Runge-Kutta method has a number attached to it, which denotes the number of slopes sampled. The most common version of Runge-Kutta is RK-4, for reasons of error

reduction which will be discussed.

$$k_1 = f(t_n, y_n), \tag{2}$$
$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1), \tag{3}$$
$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2), \tag{4}$$
$$k_4 = f(t_n + h, y_n + hk_3), \tag{5}$$
$$y_{n+1} = y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \tag{6}$$

2.3 Disease spreading models

2.3.1 Susceptible-Infected model (SIS)

The SIS model assumes each member of the population is either infected I, or susceptible to infection S. The parameter β is the rate of infection of the susceptible, whilst α is the rate of recovery of the infected.

$$\frac{dS}{dt} = -\beta\frac{SI}{N} + \alpha I, \tag{7}$$
$$\frac{dI}{dt} = \beta\frac{SI}{N} - \alpha I. \tag{8}$$

After enough time has passed, S and I tend to settle to values determined by α and β. This results in a ratio f of the total population settling as infected ($\frac{I}{N}$). f is derived as follows,

$$\frac{dS}{dt} = \frac{dI}{dt} = 0 \tag{9}$$
$$\implies -\beta\frac{SI}{N} + \alpha I = \beta\frac{SI}{N} - \alpha I \tag{10}$$
$$\implies \alpha = \beta\frac{S}{N} \tag{11}$$
$$\implies \alpha = \beta\frac{N - I}{N} \tag{12}$$
$$\implies 1 - \frac{\alpha}{\beta} = \frac{I}{N} = f. \tag{13}$$

2.3.2 Susceptible-Infected-Resistant model (SIR)

The SIR model builds upon the SIS model by introducing a variable to represent the recovered (and immune) population R. The parameter γ is additionally the rate of vaccinations of the susceptible population.

$$\frac{dS}{dt} = -\beta\frac{SI}{N} - \gamma S, \tag{14}$$
$$\frac{dI}{dt} = \beta\frac{SI}{N} - \alpha I, \tag{15}$$
$$\frac{dR}{dt} = \gamma S + \alpha I. \tag{16}$$

2.3.3 SIR with mobility model

Multiple populations are now considered independently, with mobility taking place between their populations. The rate at which members of a population n travels to population m is given by w_{nm}.

$$\frac{dS_n}{dt} = -\beta\frac{S_n I_n}{N_n} - \gamma S_n + \sum_{m\neq n}\left(w_{mn}S_m - w_{nm}S_n\right), \tag{17}$$
$$\frac{dI_n}{dt} = \beta\frac{S_n I_n}{N_n} - \alpha I_n + \sum_{m\neq n}\left(w_{mn}I_m - w_{nm}I_n\right), \tag{18}$$
$$\frac{dR_n}{dt} = \gamma S_n + \alpha I_n + \sum_{m\neq n}\left(w_{mn}R_m - w_{nm}R_n\right). \tag{19}$$

2.4 Estimating errors

2.4.1 Truncation error

The truncation error is that which comes from the approximation of an analytical solution by a simpler solution. Using the Taylor polynomial, the Euler method and it's error can be derived:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{3}f'''(x) + ... \tag{20}$$

$$\implies f'(x) = \frac{f(x+h) - f(x)}{h} - \left(\frac{h}{2}f''(x) + \frac{h^2}{3}f'''(x) + ...\right) \approx \frac{f(x+h) - f(x)}{h}. \tag{21}$$

So the truncation error $\epsilon_t = \frac{h}{2}f''(x) + \frac{h^2}{3}f'''(x) + ... \approx h|f''(x)|$, or $O(h^2)$. Similar procedure for the RK-4 method yields $O(h^5)$.

2.4.2 Local and global errors

The local error is the error at accumulated at a single step of an iterative process. The global error is the sum of these local errors. For a local error of $E_L = C_i h^m \approx h^m$, where C_i is a constant, and h^m is dependent on the step size. The global error would be $E_G = \sum_{i=1}^{N} C_i h^m$. The scaling of the errors with step size can be approximated, since the period $T = Nh$ over which a simulation is run is fixed:

$$E_G < max(C_i)\sum_{i=1}^{N} h^m \tag{22}$$

$$= max(C_i)Nh^m \tag{23}$$

$$= max(C_i)Th^{m-1} \tag{24}$$

$$\approx h^{m-1}. \tag{25}$$

So, for the Euler method, $E_L \approx h^2$ and $E_G \approx h$. And for the RK-4 method, $E_L \approx h^5$ and $E_G \approx h^4$.

2.4.3 Numerically calculating errors

By substituting in the local error,

$$Y_n^{\text{exact}} = y_n^h + E_n^h + ... \tag{26}$$

$$= y_n^h + C(h)^m + ... \tag{27}$$

And by comparing this to the same result if the step size were instead $2h$,

$$Y_n^{\text{exact}} = y_n^{2h} + E_n^{2h} + ... \tag{28}$$

$$= y_n^{2h} + C(2h)^m + ... \tag{29}$$

Subtracting the equations for step sizes h and $2h$ gives

$$0 = y_n^h - y_n^{2h} + C(h)^m - C(2h)^m \tag{30}$$

$$C(h)^m(2^m - 1) = y_n^h - y_n^{2h} \tag{31}$$

$$C(h)^m = \frac{y_n^h - y_n^{2h}}{2^m - 1}. \tag{32}$$

So, the error at any given point n in the simulation is $E_n^h = \frac{y_n^h - y_n^{2h}}{2^m - 1}$. And so by running two simulations of the same model of known error complexity m, using step sizes h and $2h$ respectively, the error can be approximated.

3 Methodology

3.1 Case study background

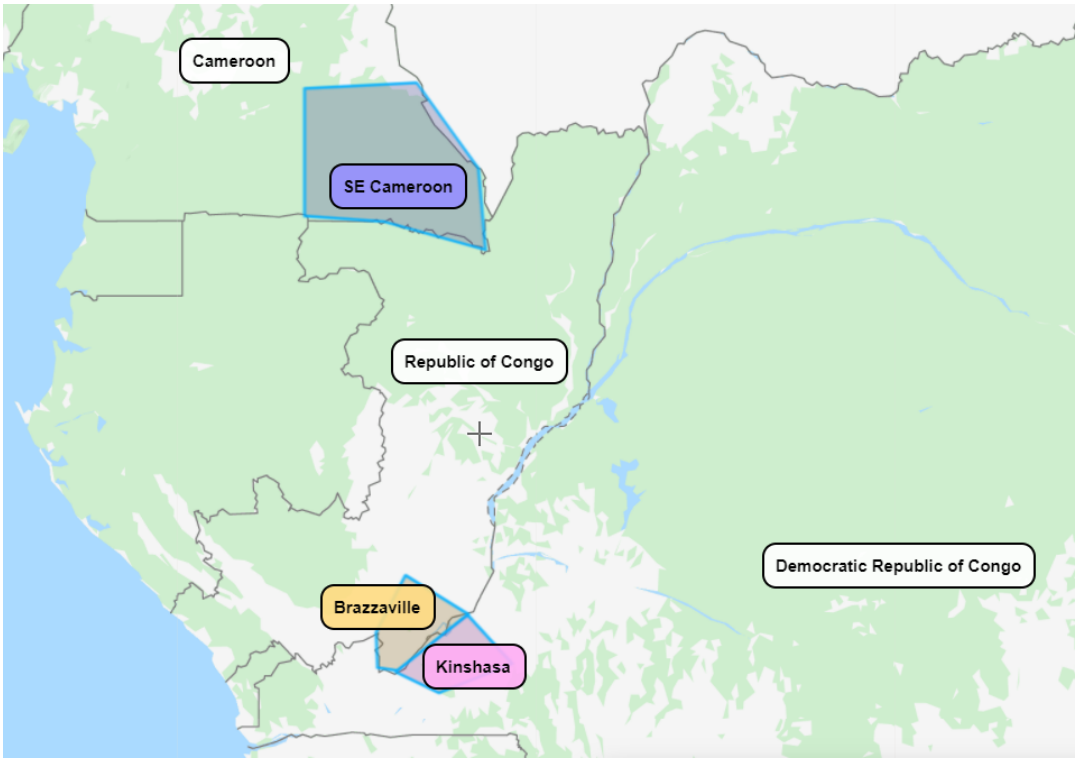

Figure 1: Map of SE Cameroon and the Congo river basin.

Patient 0 was in South-East Cameroon, the disease then spread through to what became it's main breeding ground in the Capitals of the Democratic Republic of Congo (DRC), Kinshasa (formerly Léopoldville), and the Republic of Congo (RC), Brazzaville. Both cities surround the huge Congo river basin, generating and receiving lots of mobility. Migration from South-East Cameroon to the Congo basin was fairly significant, and driven by the need for improved standards of living. The Congo basin area is strongly considered the ground-zero for the growth of HIV. There was a very slow increase in infected population in Kinshasa in the first half of 20^{th} century, until colonial-led attempts at Syphilis treatment by injection changed things... for the worse. Due to a lack of understanding of blood-borne diseases at the time, needles were regularly reused and not properly cleansed. This resulted in a huge boom in HIV cases, with a corresponding 150,000 injections in 1953. Prostitutes were targeted by these injections. In the 1960s, poverty and policy caused prostitution to change such that prostitutes worked per-service as opposed to a regular-fee. This promoted sex with a greater number of varied clients, up to 1000 clients per year per prostitute, drastically increasing the spread of the disease. [1]

3.2 Choice of variables and values

Firstly, I have chosen to take the two capitols of the RC and the DRC respectively into the model, Kinshasa and Brazzaville. Since HIV is believed to have originated in SE Cameroon, this rural region will be a third population in consideration of the model. The model has been chosen to run from 1950 to 1970. This is a large window of time, appropriate for a slow spreading disease (compared to the likes of Covid 19 seen today), and allowing for governmental policy changes to hopefully be seen taking effect. The step size h of the simulation was chosen to be $1day$, meaning there will be 7300 total steps of the simulation.

3.2.1 SE Cameroon

Approximating SE Cameroon to have a small fraction of the total population of Cameroon, or roughly double that of the capitol. $N_0 = 77,000,$ [2] [3]
$S_0 = 27,000,$

$I_0 = 50,000,$
$R_0 = 0.$

3.2.2 Brazzaville

$N_0 = 85,000,$[4]
$S_0 = 85,000,$
$I_0 = 0,$
$R_0 = 0.$

3.2.3 Kinshasa

$N_0 = 200,000,$[5]
$S_0 = 200,000,$
$I_0 = 0,$
$R_0 = 0.$

3.2.4 ws

$w_{C\to B} = w_{C\to K} = 0.0002$
$w_{B\to C} = w_{K\to C} = 0.00005$
$w_{B\to K} = w_{K\to B} = 0.00005$

3.2.5 α and γ

Normally, since vaccinations were not available at this time (though they have become available in recent years), $\gamma = 0$. And since HIV is non-curable, $\alpha = 0$.

This puts us in a good position to look into how the spread of the disease may have differed if vaccinations were available at the time, or if there was a cure available. For these experimental cases, values of $\gamma = 0.0001$ and $\alpha = 0.0001$ will be used respectively.

3.2.6 β

The rate of infection between 1950 and 1960 due to prostitution, assuming transmission rate of 0.05 from intercourse, 0.001 of the total population are prostitutes and each prostitute serves a single customer per week is $\beta^s_{50\to 60} = 0.05 * 0.001 * (\frac{1}{7})$.
The rate of infection between 1960 and 1970 due to prostitution, with prostitutes serving 1000 yearly customers is $\beta^s_{60\to 70} = 0.05 * 0.001 * (\frac{1000}{365})$.
The rate of infection due to re-use of infected needles, assuming 150000 infected injections with absolute transmission rate over the entire population of the DRC is $\beta^i = \frac{150000}{365}\frac{1}{12000000}$.[6]
For each time period the appropriate rate of infection due to prostitution and the constant rate due to infected needles was totalled and used.

4 Analysis

Figure 2 was run using $\alpha = 0.01$, $\beta = 0.05$ implies $f = 1 - \frac{0.01}{0.05} = 0.8$.

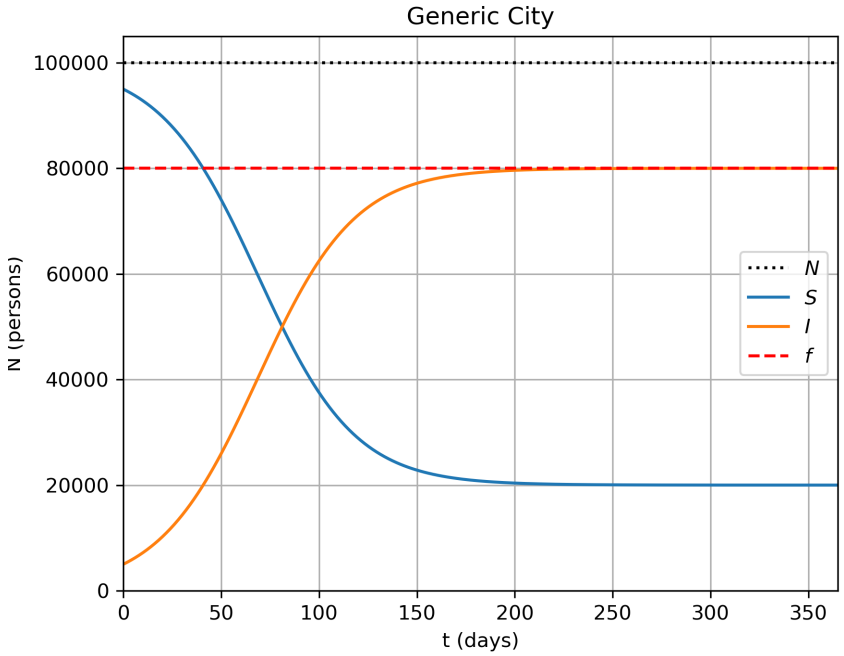

Figure 2: Evidence of the f-formula working.
SIS model, solved using Euler method.

Figure 3 shows the results of the normal model simulation. From this model, the maximum absolute differences between I_n using simulations with step-sizes $h = 1$ day and $2h$ was found to be 5.76, 3.31 and 3.37 for the populations of SE Cameroon, Brazzaville and Kinshasa respectively. Resulting in corresponding max $(E_n^{1\text{day}}) =$ 1.92, 1.10 and 1.12 when considering the error complexity of the Euler method.

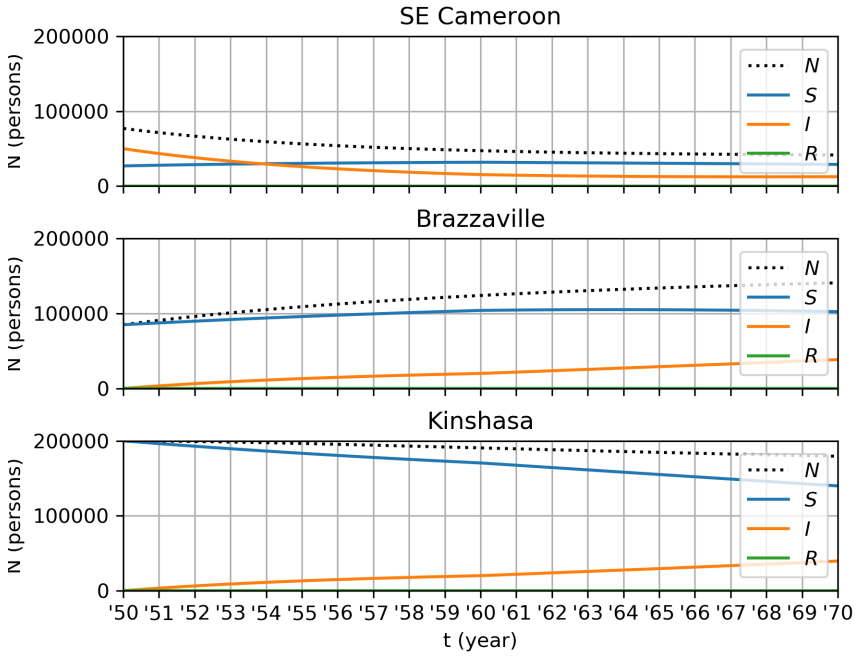

Figure 3: Infections of HIV in West Africa (1950 to 1970).
SIR model with mobility, solved using Euler method.

Figure 4 shows the results of the vaccinations-enabled model simulation.

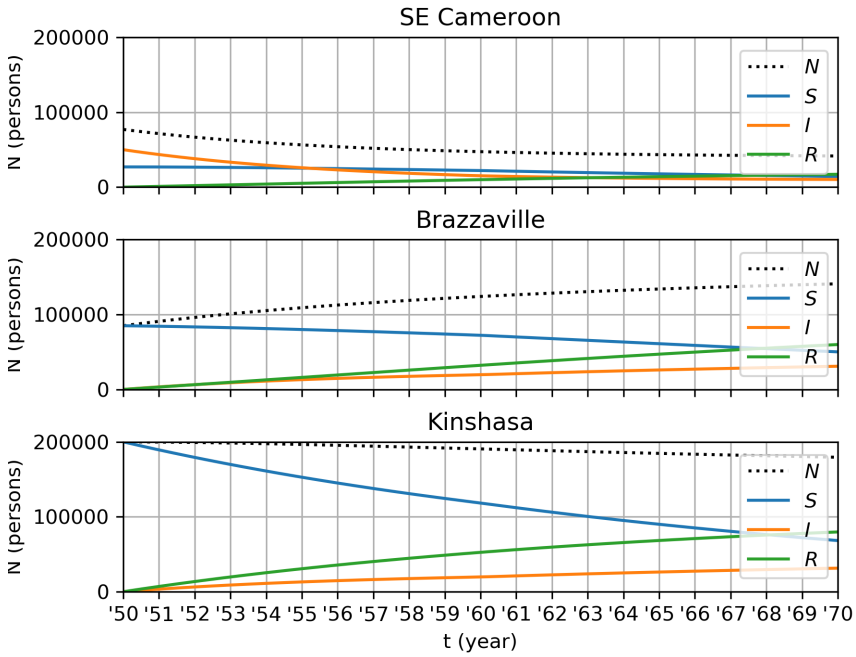

Figure 4: Infections of HIV in West Africa (1950 to 1970). If HIV were being vaccinated against. SIR model with mobility, solved using Euler method.

Figure 4 shows the results of the recoverable-enabled model simulation.

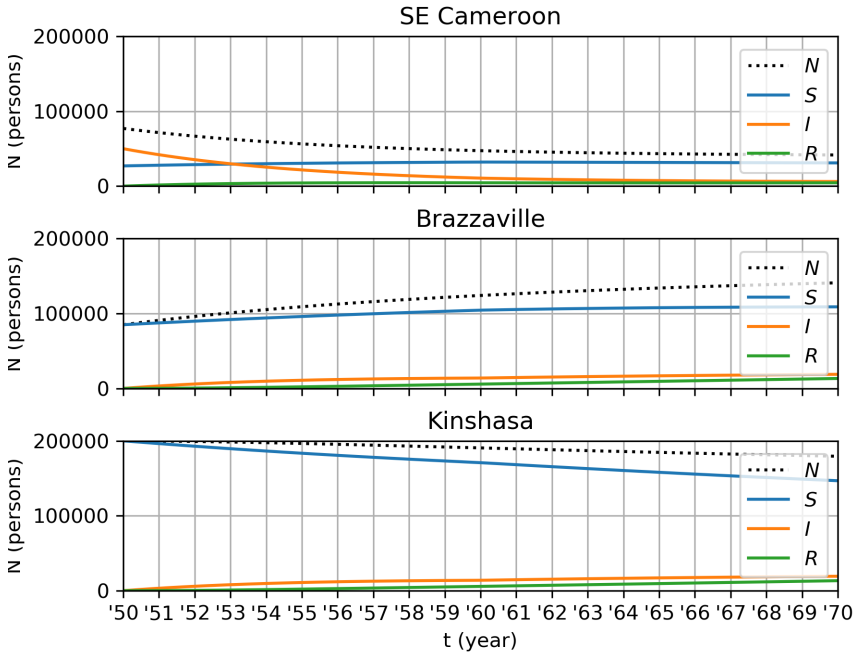

Figure 5: Infections of HIV in West Africa (1950 to 1970). If HIV were recoverable from. SIR model with mobility, solved using Euler method.

5 Discussion

As seen in figure 2, the f formula held up as expected in the SIS case. With the expected 80% of the population becoming infected after a substantial time frame.

Though the model is run across a huge period of 20 years, the predictions do not grow wild. This can be seen as a success of the model, compared to models of more chaotic systems such as weather forecasts, which rarely hold up for a week. In fact, I think the normal model impressively predicts reasonable infection rates in 1970 of proportions of the population ranging from roughly 20-40%.

As seen in figure 3, there was not as significant a change in infection rates from the 1950s to the 1960s as expected. This is even with other noisy factors being neglected from the model, and with a sharp change in prostitution assumed to take place overnight. Furthermore, the number of clients of prostitution chosen for the '50s in retrospect seems to have been chosen to be too low. And even yet, what I expect to be a exaggerated difference in rates between the '50s and '60s is just barely noticeable. Of course, this is not sufficient proof that the hypothesis that prostitution played a large role in the spread of HIV is not creditable. The models are far too crude and lacking in finesse to draw any such conclusions as of yet. It could be that the percentage of the population working as prostitutes was vastly underestimated in the model, amongst other inaccuracies.

Comparing the effects that the rate of vaccinations has compared to an equal rate of recovery: As seen in figure 4, it can be seen that the roll-out of vaccinations before large portions of the population are infected results in a great deal of immunity. Whilst as seen in figure 5, recover-ability from the disease reduces the peak of infection, similar to how the f factor operates in the simpler SIS model.

Unfortunately, I am unable to experimentally compare the results of the Euler method to those of RK-4. This is because great trouble was met trying to get RK-4 working with the difference functions. Attempts to alter the difference functions were made to facilitate RK-4, but unfortunately it is becoming overly complex and requires a pretty major overhaul to the current underlying code that I have built my program around. Of course this huge time cost is not something which I can undertake at this stage in the project. The lesson to be learnt here is to be cautious of features in a programming language that are not fully understood. I opted to use lambda functions in python as a fancy maths tool that I was unfamiliar with, this ended up ultimately hindering me in the long run, as regular python functions would have been much simpler... and actually work. I have also realised that my code could be made clearer to read and easier to adapt if I were using Object Oriented Programming (OOP) in python, which is something I tend to shy-from in the language.

Regardless, the errors $\max\left(E_n^{1\text{day}}\right)$ of the Euler method were surprisingly low, with a maximum of just 1.92 infected persons. I expect this is due to the generally understated infection rates used in the simulation. In comparison to a super fast spreading pandemic such as Covid-19, much higher errors would be expected to be seen, as sharp peaks in infection rates leave room for larger differences between simulations of differing step sizes.

6 Conclusion

It has been seen how ODEs can be computationally solved, with the application of various models of the spread of HIV in Eastern Africa. The initial values have been fed by research on the disease and the surrounding history of the region. The final results have been compared to the known data from the time, and feedback and criticisms offered to the models. The Euler method has been used to perform the simulations, and the resulting computational errors have been calculated. The project has done well to develop my understanding in the practical applications of computational physics, and has helped tidy up my programming skills.

References

[1] Jacques Pépin. The origins of AIDS: from patient zero to ground zero. `https://jech.bmj.com/content/67/6/473`. Accessed 2021-09-15.

[2] macrotrends, United Nations - World Population Prospects. Yaounde, Cameroon Metro Area Population 1950-2021. `https://www.macrotrends.net/cities/20365/yaounde/population`, . Accessed 2021-09-15.

[3] macrotrends, United Nations - World Population Prospects. Cameroon Population 1950-2021. `https://www.macrotrends.net/countries/CMR/cameroon/population`, . Accessed 2021-09-15.

[4] macrotrends, United Nations - World Population Prospects. Brazzaville, Congo Metro Area Population 1950-2021. https://www.macrotrends.net/cities/20848/brazzaville/population, . Accessed 2021-09-15.

[5] macrotrends, United Nations - World Population Prospects. Kinshasa, Republic of Congo Metro Area Population 1950-2021. https://www.macrotrends.net/cities/20853/kinshasa/population, . Accessed 2021-09-15.

[6] PopulationPyramid. Democratic Republic of the Congo 1950. https://www.populationpyramid.net/democratic-republic-of-the-congo/1950/. Accessed 2021-09-15.

Appendix

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MultipleLocator
import copy

''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Euler ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
def ForwardEuler(t, ys, fs, Ns):
    for n in range(0, len(t) - 1):
        for c in range(len(ys)): # cities
            for i in range(len(ys[c])): # variables
                h = t[n+1] - t[n]
                ys[c][i][n+1] = ys[c][i][n] + fs[i](n, c) * h
                #Ns[c][n+1] += ys[c][i][n+1]
                Ns[c][n+1] += ys[c][i][n+1]
    return (ys, Ns)


''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ RK 4 ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
'''
def RK4(): # https://blog.tonytsai.name/blog/2014-11-24-rk4-method-for-solving-sir-model/
    for n in range(0, len(t) - 1):
        for c in range(len(ys)): # cities
            for i in range(len(ys[c])): # variables
                h = t[n+1] - t[n]

                #
                k1 = h * fs[i](n, c)
                k2 = h * fs[i](n + h/2, c) # So... need to double lengths of S, I, R, N lists (n+h/2).
                k3 = h * fs[i](n + h/2, c)
                k4 = h * fs[i](n + h, c)


        k1 = h * (f(x0, y0))
        k2 = h * (f((x0+h/2), (y0+k1/2)))
        k3 = h * (f((x0+h/2), (y0+k2/2)))
        k4 = h * (f((x0+h), (y0+k3)))
        k = (k1+2*k2+2*k3+k4)/6
        yn = y0 + k
        print('%.4f\t%.4f\t%.4f'% (x0,y0,yn) )
        print('-------------------------')
        y0 = yn
        x0 = x0+h

    print('\nAt x=%.4f, y=%.4f' %(xn,yn))
'''


''' ~~~~~~~~~~~~~~~~~~~~~~~~ Defining Variables ~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
'''
    sigma is the the number of contacts per units time between individuals - region dependent.
    T is the infection probability for each contact     disease dependent.
    beta is the rate of infection.
    alpha is the recovery rate.
    gamma is the rate of immunity and vaccination.
    S_n = S_n(t) is the number of susceptible members of population n at time t.
    I_n = I_n(t) is the number of infected members of population n at time t.
    R_n = R_n(t) is the number of recovered and immune members of population n at time t.
    N_n = S_n + I_n + R_n size of population n. Constant if travel is balanced.
'''
```

```
deltaT = 365 * 20 #Taking the unit of time to be 1 day. Between [t, t + deltaT]
#deltaT = 365 * 1

h = 1
t = np.linspace(0, 0 + deltaT, deltaT / h) # calculating once each day.

# SE Cameroon / Generic City
S_1_0 = 27000
I_1_0 = 50000
R_1_0 = 0
N_1_0 = S_1_0 + I_1_0 + R_1_0
'''
S_1_0 = 95000
I_1_0 = 5000
R_1_0 = 0
N_1_0 = S_1_0 + I_1_0
'''

# Brazzaville
S_2_0 = 85000
I_2_0 = 0
R_2_0 = 0
N_2_0 = S_2_0 + I_2_0 + R_2_0

# Kinshasa
S_3_0 = 200000
I_3_0 = 0
R_3_0 = 0
N_3_0 = S_3_0 + I_3_0 + R_3_0

beta_i = (150000 / 365) / 12000000

#T = 0.02 # for vaginal sex : https://stanfordhealthcare.org/medical-conditions/sexual-and-reproductive-health/
    hiv-aids/causes/risk-of-exposure.html#:~:text=Therefore%2C%20unprotected%20sex%20with%20an,exposures)%20for
    %20receptive%20anal%20sex.
beta_s_50 = 0.05 * 0.001 * (1 / 7) # rate of transmission from vaginal sex, population of which sex workers,
    number of clients.
beta_s_60 = 0.05 * 0.001 * (1000 / 365)

beta_50 = beta_i + beta_s_50
beta_60 = beta_i + beta_s_60

'''
alpha = 0.01
beta = 0.05
gamma = 0
beta_50 = beta
'''

# Normal:
alpha = 0
gamma = 0


# If vaccine was available:
'''
alpha = 0
gamma = 0.0001
'''

# If recoverable from:
'''
alpha = 0.0001
gamma = 0
'''

# If both:
'''
alpha = 0.0001
gamma = 0.0001
'''


''' ~~~~~~~~~~~~~~~~~~~~~~~~~ Mobility Variables ~~~~~~~~~~~~~~~~~~~~~~~~~ '''
'''
    w_mn is the probability per unit time that a random person in population m travels to population n.
```

```
'''
w_12 = w_13 = 0.0002
w_21 = w_31 = 0.00005
w_23 = w_32 = 0.00005

w_1 = [0, w_12, w_13]
w_2 = [w_21, 0, w_23]
w_3 = [w_31, w_32, 0]

'''
w_1 = [0, 0, 0]
w_2 = [0, 0, 0]
w_3 = [0, 0, 0]
'''

w = []
w.append(w_1)
w.append(w_2)
w.append(w_3)

#w = [0]


''' ~~~~~~~~~~~~~~~~~~~~~~~~~~ Generating Lists ~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
def GenEmptyVarLists(vs, v_0s):
    for i in range(len(vs)):
        vs[i] = np.zeros(len(t))
        vs[i][0] = v_0s[i]

    return(vs)

S_1, I_1, R_1, N_1 = [], [], [], []
S_1, I_1, R_1, N_1 = GenEmptyVarLists([S_1, I_1, R_1, N_1], [S_1_0, I_1_0, R_1_0, N_1_0])

S_2, I_2, R_2, N_2 = [], [], [], []
S_2, I_2, R_2, N_2 = GenEmptyVarLists([S_2, I_2, R_2, N_2], [S_2_0, I_2_0, R_2_0, N_2_0])

S_3, I_3, R_3, N_3 = [], [], [], []
S_3, I_3, R_3, N_3 = GenEmptyVarLists([S_3, I_3, R_3, N_3], [S_3_0, I_3_0, R_3_0, N_3_0])


''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Functions ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
''' HAVING TROUBLE IMPLEMENTING RUNGE-KUTTA COMPATABILITY.
dS_nodt = lambda t, n, i_k: (- beta * (ys[n][0][t] + ks[n][0][i_k]) * (ys[n][1][t] + ks[n][1][i_k] / 2) / Ns[n
    ][t] - gamma * (ys[n][0][t] + ks[n][0][i_k] / 2)) + mobilitySum(t, n, 0, i_k)
dI_nodt = lambda t, n, i_k: (beta * (ys[n][0][t] + ks[n][0][i_k]) * ys[n][1][t] / Ns[n][t] - alpha * ys[n][1][t
    ]) + mobilitySum(t, n, 1, i_k)
dR_nodt = lambda t, n, i_k: (gamma * ys[n][0][t] + alpha * ys[n][1][t]) + mobilitySum(t, n, 2, i_k)
# NOTE: when setting the values of ks, need to multiply by h.
# NOTE: Is there gonna be a problem with N... I.e. N not calculated at h/2, so instead just assume it is Ns[n][
    t].

#ks[city][variable][kutta number]

def mobilitySum(t, n, v, i_k):
    output = 0
    for m in range(len(Ns)): # number of cities
        if (m != n):
            output += (w[m][n] * (ys[m][v][t] + ks[m][v][i_k] / 2) - w[n][m] * (ys[n][v][t] + ks[n][v][i_k] /
    2))
    return(output)
'''

dS_nodt = lambda t, n: (- Beta(t) * ys[n][0][t] * ys[n][1][t] / Ns[n][t] - gamma * ys[n][0][t]) + MobilitySum(t
    , n, 0)
dI_nodt = lambda t, n: (Beta(t) * ys[n][0][t] * ys[n][1][t] / Ns[n][t] - alpha * ys[n][1][t]) + MobilitySum(t,
    n, 1)
dR_nodt = lambda t, n: (gamma * ys[n][0][t] + alpha * ys[n][1][t]) + MobilitySum(t, n, 2)

'''
dS_nodt = lambda t, n: (- Beta(t) * ys[n][0][t] * ys[n][1][t] / Ns[n][t] + alpha * ys[n][1][t]) + MobilitySum(t
    , n, 0)
dI_nodt = lambda t, n: (Beta(t) * ys[n][0][t] * ys[n][1][t] / Ns[n][t] - alpha * ys[n][1][t]) + MobilitySum(t,
    n, 1)
'''
```

```
def MobilitySum(t, n, v):
    output = 0
    for m in range(len(Ns)): # number of cities
        if (m != n):
            output += (w[m][n] * ys[m][v][t] - w[n][m] * ys[n][v][t])
    return(output)

def Beta(t):
    beta = 0
    if (t * h < deltaT / 2):
        beta = beta_50
    else:
        beta = beta_60
    return(beta)


''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Solving ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
ys = [[S_1, I_1, R_1], [S_2, I_2, R_2], [S_3, I_3, R_3]]
fs = [dS_nodt, dI_nodt, dR_nodt]
Ns = [N_1, N_2, N_3]

'''
ys = [[S_1, I_1]]
fs = [dS_nodt, dI_nodt]
Ns = [N_1]
'''

ys, Ns = ForwardEuler(t, ys, fs, Ns)


''' ~~~~~~~~~~~~~~~~~~~~~~~~~~ Calculating error ~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
def CalcError(yh, y2h, m):
    error = (yh - y2h) / (2**m - 1)
    #error = (yh - y2h)
    return(error)

calcError = False
if (calcError == True):
    # Create a renamed deep copy of ys for error stuff.
    ys_h = copy.deepcopy(ys)
    Ns_h = copy.deepcopy(Ns)

    ys.clear()
    Ns.clear()

    # Repurpose ys for 2h.
    h *= 2
    t = np.linspace(0, 0 + deltaT, deltaT / h)

    S_1, I_1, R_1, N_1 = [], [], [], []
    S_1, I_1, R_1, N_1 = GenEmptyVarLists([S_1, I_1, R_1, N_1], [S_1_0, I_1_0, R_1_0, N_1_0])

    S_2, I_2, R_2, N_2 = [], [], [], []
    S_2, I_2, R_2, N_2 = GenEmptyVarLists([S_2, I_2, R_2, N_2], [S_2_0, I_2_0, R_2_0, N_2_0])

    S_3, I_3, R_3, N_3 = [], [], [], []
    S_3, I_3, R_3, N_3 = GenEmptyVarLists([S_3, I_3, R_3, N_3], [S_3_0, I_3_0, R_3_0, N_3_0])

    ys = [[S_1, I_1, R_1], [S_2, I_2, R_2], [S_3, I_3, R_3]]
    Ns = [N_1, N_2, N_3]
    ys, Ns = ForwardEuler(t, ys, fs, Ns)

    ys_2h = ys
    Ns_2h = Ns

    # Calculate errors at all (overlapping) timesteps
    errors = [[], [], []]
    m = 2

    for c in range(len(ys)): # cities.
        for i in range(len(ys_2h[c][1])): # 2h values.
            errors[c].append(CalcError(ys_h[c][1][2*i], ys_2h[c][1][i], 2)) # just looking at the I variable.

    #plt.plot(t, errors[1])
    #plt.plot(t, ys_2h[1][1])
    print(max(np.abs((errors[0]))), max(np.abs(errors[1])), max(np.abs(errors[2])))
```

```
''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ f-Factor ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
'''
f = 1 - alpha / beta
fList = []
for i in range(0,len(t)):
    fList.append(f * max(N_1))
'''


''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Plotting ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
if (calcError == False):
    yRoof = max(max(N_1), max(N_2), max(N_3))
    #yRoof = max(N_1) * 1.05

    fig, axs = plt.subplots(3)
    #fig, axs = plt.subplots(1)

    fig.subplots_adjust(left = 0.125, right = 0.9, bottom = 0.1, top = 0.9, wspace = 0.2, hspace = 0.35)
    #fig.suptitle("")

    def PlotAx(ax, t, N_n, S_n, I_n, R_n, cityName, deltaT, yRoof):
    #def PlotAx(ax, t, N_n, S_n, I_n, cityName, deltaT, yRoof):
        ax.plot(t,N_n,'k:')
        ax.plot(t,S_n,'-')
        ax.plot(t,I_n,'-')

        ax.plot(t,R_n,'-')
        #ax.plot(t,fList,'r--')



        #ax.legend(["$N_"+cityName[0]+"$", "$S_"+cityName[0]+"$", "$I_"+cityName[0]+"$", "$R_"+cityName[0]+"$
     "])
        ax.legend(["$N$", "$S$", "$I$", "$R$"], loc = "center right")
        #ax.legend(["$N$", "$S$", "$I$", "$f$"], loc = "center right")

        ax.set_xlim(0, 0 + deltaT)
        ax.set_ylim(0, yRoof)

        ax.set_xlabel("t (year)")
        #ax.set_xlabel("t (days)")

        ax.set_ylabel("N (persons)")
        ax.grid(True)
        ax.set_title(cityName)

        return()

    PlotAx(axs[0], t, Ns[0], ys[0][0], ys[0][1], ys[0][2], "SE Cameroon", deltaT, yRoof)
    PlotAx(axs[1], t, Ns[1], ys[1][0], ys[1][1], ys[1][2], "Brazzaville", deltaT, yRoof)
    PlotAx(axs[2], t, Ns[2], ys[2][0], ys[2][1], ys[2][2], "Kinshasa", deltaT, yRoof)

    #PlotAx(axs, t, Ns[0], ys[0][0], ys[0][1], "Generic City", deltaT, yRoof)

    for ax in fig.get_axes():
        ax.label_outer()

        ax.xaxis.set_major_locator(MultipleLocator(365))

        labels = [item.get_text() for item in ax.get_xticklabels()]
        labels = ["", "'50", "'51", "'52", "'53", "'54", "'55", "'56", "'57", "'58", "'59", "'60", "'61", "'62"
     , "'63", "'64", "'65", "'66", "'67", "'68", "'69", "'70"]

        ax.set_xticklabels(labels)


    ''' ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Saving ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ '''
    #figname = "normal"
    #figname = "vaccine"
    #figname = "recoverable"
    #figname = "both"

    #figname = "f"
```

```
#fig.savefig(figname, dpi=300)
```