University of Texas at Arlington

MavMatrix

2021 Fall Honors Capstone Projects

Honors College

12-1-2021

PARAMETER ESTIMATION AND SENSITIVITY ANALYSIS THROUGH MATHEMATICAL MODELING OF COLON CANCER

Achyuth Manoj

Follow this and additional works at: https://mavmatrix.uta.edu/honors_fall2021

Recommended Citation

Manoj, Achyuth, "PARAMETER ESTIMATION AND SENSITIVITY ANALYSIS THROUGH MATHEMATICAL MODELING OF COLON CANCER" (2021). *2021 Fall Honors Capstone Projects*. 24.
https://mavmatrix.uta.edu/honors_fall2021/24

This Honors Thesis is brought to you for free and open access by the Honors College at MavMatrix. It has been accepted for inclusion in 2021 Fall Honors Capstone Projects by an authorized administrator of MavMatrix. For more information, please contact leah.mccurdy@uta.edu, erica.rousseau@uta.edu, vanessa.garrett@uta.edu.

Copyright © by Achyuth Manoj 2021

All Rights Reserved

PARAMETER ESTIMATION AND SENSITIVITY ANALYSIS THROUGH MATHEMATICAL MODELING OF COLON CANCER

by

ACHYUTH MANOJ

Presented to the Faculty of the Honors College of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

HONORS BACHELOR OF SCIENCE IN MATHEMATICS

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2021

ACKNOWLEDGMENTS

Firstly, I would like to thank Dr. Roy and Dr. Pal for the support they have provided me as mentors, and for all the knowledge that they have given me throughout the research. I would also like to thank Juan Villegas, Susanth Kakarla, and Mesfer Alajmi who were the other students who collaborated with me on this project. It has been a pleasure working with them on this project. This work was supported by the National Institutes of Health Grant Number R21CA242933.

Most importantly, I would like to thank my parents for their love and emotional support, which has allowed me to grow as a person, for cultivating my interests, and for their hard work that allowed me to pursue my interests and receive this level of education in a foreign country far from home.

November 19, 2021

ABSTRACT

PARAMETER ESTIMATION AND SENSITIVITY

ANALYSIS THROUGH MATHEMATICAL

MODELING OF COLON CANCER

Achyuth Manoj, B.S. Mathematics

The University of Texas at Arlington, 2021

Faculty Mentor: Souvik Roy

We formulated a new and efficient method to propose a personalized treatment platform for colorectal cancer. A mathematical model of colon cancer was used and is comprised of a system of differential equations, which model various cell dynamics. The dynamics are dependent on patient-specific parameters that are unknown which we estimate given patient data in form of cell measurements. We approached this estimation as an inverse problem based on an optimization framework and developed computational optimization techniques created on non-linear conjugate gradient methods to solve for the optimal set of parameters for a specific patient. These optimal parameters are then ranked by conducting a sensitivity analysis using the Latin Hypercube Sampling-Partial Rank Correlation Coefficient method, to determine the most sensitive parameters with respect to

the tumor cell count. Using this information, we can deduce the types of feasible treatment strategies which can be utilized for curing the patient.

TABLE OF CONTENTS

ACKNOWLEDGMENTS iii

ABSTRACT iv

LIST OF ILLUSTRATIONS viii

LIST OF TABLES ix

Chapter

1. INTRODUCTION 1

1.1 Colorectal Cancer 1

1.1.1 Modelling and Parameter Estimation 1

1.1.2 Colorectal Cancer Model 2

2. OPTIMIZATION TECHNIQUES 4

2.1 Need for Numerical Optimization 4

2.2 Methodology 4

2.2.1 Forward Solver 4

2.2.2 Reverse Solver 5

2.2.3 Nonlinear Conjugate Gradient Algorithm 6

2.2.4 Sensitivity Analysis 8

3. NUMERICAL RESULTS 10

3.1 Test Case 10

4. CONCLUSION 12

Appendix

A. PYTHON SCRIPT OF OPTIMIZATION ALGORITHM 13

REFERENCES 26

BIOGRAPHICAL INFORMATION 27

LIST OF ILLUSTRATIONS

Figure Page

3.1 Evolution of T, N, L, C with true (red) and optimal (blue) parameter set 11

LIST OF TABLES

Table | Page

1.1 Variables of the system of ODEs 2

1.2 Coefficients of the system of ODEs 3

1.3 Sensitivity analysis of test case 10

CHAPTER 1

INTRODUCTION

1.1 Colorectal Cancer

Colorectal cancer is one of the leading causes of cancer related deaths in the United States. The American Cancer society estimates 100,000 new cases of colon cancer will be diagnosed in 2021, with more than 50,000 deaths [1]. This large death rate is due to difficulties in early diagnosis, treatment being managed on an individual level, and lack of cost-effective experimental testing of drug efficacy. For this reason, computational models are a cost-effective alternative which can feasibly provide optimal treatments at the individual level without the need for extensive clinical testing.

1.1.1 Modelling and Parameter Estimation

Dynamical modelling can be used for systems which evolve over time, such as colorectal cancer. Several dynamical models have already been developed to model colon cancer. For our purposes, we used the model proposed by dePillis *et al.* as dynamic models are usually represented by a set of differential equations. In biological systems, these equations are often quite complex and dependent on parameters which vary from patient to patient. These unknown parameters appear as coefficients in the differential equations and need to be estimated as the tumor properties can also vary between individuals. Thus, the parameter estimation is to be done only given the individual tumor data and is a key part of recommending treatment based on computational modelling methods. This is considered a reverse Ordinary Differential Equation (ODE) problem where we are given

the evolution of the variables and must estimate the coefficients.

1.1.2 Colorectal Cancer Model

We used a dynamic colorectal cancer model developed by dePillis *et al.* [2]. The system of ODEs that govern the model are as follows.

$$\frac{dT}{dt} = aT(1 - bT) - cNT - DT,\ T(0) = T_0$$

$$\frac{dN}{dt} = eC - fN - pNT,\ N(0) = N_0$$

$$\frac{dL}{dt} = j\frac{T}{k + T}L - qLT + (r_1N + r_2C)T,\ L(0) = L_0$$

$$\frac{dC}{dt} = \alpha - \beta C,\ C(0) = C_0$$

Where the variable D is given by

$$D = d\frac{(L/T)^l}{s + (L/T)^l}.$$

Here, the unknown patient parameters are d, l, s, p, k, and q, which are represented by the set θ. The physical quantities that are represented by the set of equations are below.

Table 1.1: Variables of the system of ODEs [2]

Variable	Physical Quantity
T	Total tumor cell population
N	Concentration of Natural Killer (NK) cells per liter of blood (cells/L)
L	Concentration of cytotoxic T lymphocytes ($CD8^+$) per liter of blood (cells/L)
C	Concentration of lymphocytes per liter of blood, not including NK cells and active $CD8^+$T cells (cells/L)

Table 1.2: Coefficients of the system of ODEs [2]

Coefficient	Physical quantity
a	Growth rate of tumor
b	Inverse of carrying capacity
c	Rate of NK-induced tumor death
d	Immune-system strength coefficient
e	Rate of NK cell synthesis
f	Rate of NK cell turnover
j	Rate of $CD8^+$T-cell lysed tumor cell debris activation of $CD8^+$T cells
k	Tumor size for half maximal $CD8^+$T lysed debris $CD8^+$T activation
l	Immune-system strength scaling coefficient
p	Rate of NK cell death due to tumor interaction
q	Rate of $CD8^+$T cell death due to tumor interaction
r_1	Rate of NK lysed tumor cell debris activation of $CD8^+$T cells
r_2	Rate of $CD8^+$T-cell production from circulating lymphocytes
s	Value of $(L/T)^l$ necessary for half maximal $CD8^+$T cell effectiveness against tumor
α	Lymphocyte synthesis in bone marrow
β	Rate or lymphocyte turnover

CHAPTER 2

OPTIMIZATION TECHNIQUES

2.1 Need for Numerical Optimization

Due to the complexity of the system of ODEs, we cannot solve for the parameter values analytically. Instead we look to numerical optimization methods that give us an estimate of parameter values, which are well established to find solutions for ordinary and partial differential equations. These methods have a wide range of applications in mathematical science, examples of which include the Navier-Stokes equation in aerodynamics, Schrödinger's equation in quantum mechanics, and Maxwell's equations in electrodynamics [3]. In the following sections, we outline the computational algorithms used to estimate the unknown parameter set θ.

2.2 Methodology

The first step is to use Explicit Euler's method to obtain the evolution of the system with an initial guess of the parameter set, starting from time t_0 and moving in equal intervals up to a time t_n, also known as the forward solver. We then use the Explicit Euler's method again on an adjoint set of equations, for which we solve backwards from t_n to t_0. Finally, we can use a nonlinear conjugate gradient (NCG) method, which is our optimization algorithm, that minimizes a functional giving an optimal parameter set.

2.2.1 Forward Solver

Explicit Euler's method is an easy to implement, efficient computational method for solving differential equations [4]. The algorithm of the forward solver used for the

above set of equations is outlined below. This is considered a general first order initial value problem.

$$\frac{dx}{dt} = f(x,t), x(t_0) = x_0$$

It can be assumed that f and f_x are continuous so a solution exists for this problem. We consider an interval of time [t_0, t_1, …., t_n] for which we want to solve the equation by finding the corresponding x values [x_0,x_1,….,x_n]. We approximate x_1 by using the equation of the tangent line at (t_0,x_0), taking f(x_0,t_0) as the slope of the tangent line.

$$x_1 = x_0 + f(x_0,t_0)(t_1 - t_0)$$

We solve for the other values of x in a similar fashion by using the tangent line at the previously solved point considering the values of the function f at the previous point as the slope.

$$x_{i+1} = x_i + f(x_i,t_i)(t_{i+1} - t_i)$$

Thus, we obtain the values of x for different t values in the interval, solving forward from t_0 to t_n. We apply the forward solver to all the equations in the governing ODE model, so that we have T, N, L, and C solved for at all the values of t_i in the interval. The Python 3 code used for the forward solver is given in the appendices section.

2.2.2 Reverse Solver

Next, the Explicit Euler method is applied to a set of adjoint equations. This set of adjoint equations is given as follows.

$$\frac{dT_a}{dt} = abTT_a - a(1-bT)T_a - cNT_a - DT_a - pNN_a + j\frac{k+TLL_a}{(k+T)^2} - qLL_a$$

$$+ (r_1N + r_2C)L_a, T_a(t_f) = T_a^f$$

$$\frac{dN_a}{dt} = -fN_a - pTN_a - cTT_a + r_1TL_a, N_a(t_f) = N_a^f$$

$$\frac{dL_a}{dt} = j\frac{TL_a}{k+T} + qTL_a, L_a(t_f) = L_a^f$$

$$\frac{dC_a}{dt} = -\beta C_a + eN_a + r_2 TL_a, C_a(t_f) = C_a^f$$

The values of T_a, N_a, L_a, and C_a are the adjoint values of T, N, L, C from the ODE model discussed in 1.1.2. The Explicit Euler method is applied in reverse to this set of equations, where we start from t_n and step down in equal intervals to t_0. This gives adjoint values of the model variables at all the points $[t_n, \ldots, t_0]$. Data obtained from the forward solver and the reverse solver will be used in the NCG algorithm.

2.2.3 Nonlinear Conjugate Gradient Algorithms

The primary optimization technique used for parameter estimation is outlined in the functional to be minimized below. We use the NCG method proposed by Dai and Yuan [5].

$$\boldsymbol{\theta}^* = argmin_{\boldsymbol{\theta}>0} J(\boldsymbol{\theta})$$

$$:= \sum_{i=1}^{n} \Big[\frac{\alpha_T}{2}(T(t_i) - T_i)^2 + \frac{\alpha_N}{2}(N(t_i) - N_i)^2 + \frac{\alpha_L}{2}(L(t_i) - L_i)^2$$

$$+ \frac{\alpha_C}{2}(C(t_i) - C_i)^2\Big] + \|\boldsymbol{\theta}\|_{L^2}^2$$

The α values in the formula are obtained by using an Armijo Line-search method. The values of T_i, N_i, L_i, C_i, are from the given data while $T_i(t)$, $N_i(t)$, $L_i(t)$, $C_i(t)$, are acquired using the forward solver. The optimization is subject to the following constraint:

$$\nabla_\theta J(t, \theta) = \left(\frac{\delta J}{\delta d}(t, \theta), \frac{\delta J}{\delta l}(t, \theta) \frac{\delta J}{\delta s}(t, \theta) \frac{\delta J}{\delta p}(t, \theta) \frac{\delta J}{\delta k}(t, \theta) \frac{\delta J}{\delta q}(t, \theta)\right) = 0$$

The algorithm for NCG optimization for our minimization problem is given below:

(1) Start

(2) Input: initial approximation θ^0.

(3) Evaluate $d^0 = -\nabla_\theta J(t, \theta^0)$

(4) Index k = 1, maximum $k = k_{max} = (50)$, tolerance = tol =(10−5).

(5) While (k < kmax) do

(6) Compute α^{k-1} using the Armijo line-search algorithm.

(7) Set $\theta^k = \theta^{k-1} + \alpha^{k-1} d^{k-1}$

(8) Compute $g^k = \nabla_\theta J(t, \theta^k)$.

(9) Compute β^{k-1} using Dai-Yuan formula [5].

(10) Set $d^k = -g^k + \beta^{k-1} d^{k-1}$.

(11) If $||\nabla_\theta J(t, \theta^k)|| <$ tol or $k = k_{max}$, terminate loop.

(12) Set k = k + 1.

(13) End while.

(14) If $||\nabla_\theta J(t, \theta^k)|| <$ tol, then print θ^k as the minimum else convergence not achieved.

(15) Stop

Next, we outline the line search algorithm:

(1) Start

(2) Input: initial approx. α k−1 = 1, c = 0.4, θ^{k-1} , $\nabla_\theta J(t, \theta^{k-1})$.

(3) Index j = 0, maximum $j = j_{max} = (10)$

(4) While $(j < j_{max})$ do

(5) Compute $d_1 = J(\theta^{k-1} - \alpha^{k-1}\nabla_\theta J(t, \theta^{k-1}))$

(6) Compute $d_2 = J(\theta^{k-1}) - c\alpha^{k-1}||\nabla_\theta J(t, \theta^{k-1})||^2$

(7) If $d_1 \leq d_2$, return α^{k-1} and terminate loop.

(8) $\alpha^{k-1} = \alpha^{k-1}/2$

(9) Set $j = j + 1$.

(10) End while.

(11) If $j = j_{max}$, then return $\alpha^{k-1} = 0$.

(12) Stop

The NCG algorithm gives us a result of an optimal parameter set θ^*. The Python 3 code used for the NCG algorithm is given in the appendices.

2.2.4 Sensitivity Analysis

A study of the uncertainty of the parameter values is important, as the effect of the model parameters on the outputs need to be understood. Sensitivity analysis follows an uncertainty analysis as it helps allocate the outputs of the model to input sources. Uncertainty and sensitivity analysis ranks the parameters in the magnitude by which they contribute to the inaccuracy of the outputs. We make use of the Latin Hypercube Sampling (LHC) scheme and Partial Rank Correlation Coefficient Analysis (PRCC) [6]. Latin Hypercube Sampling generates a random set of parameter values such that each parameter is set to follow a normal distribution with a 10% standard deviation. Each of these distributions are divided into intervals of equal probability, and these intervals are sampled once without replacement so that the entire range for each parameter is explored. PRCC analysis involves rank transforming the LHC matrix and then partial correlation on the rank transformed data from the matrix. The significance of the rank assigned to each parameter is determined through the calculated p-value. If the p-value is less than the chosen level of

significance, then the partial rank correlation coefficient is considered to be significant implying that the parameter is sensitive.

CHAPTER 3

NUMERICAL RESULTS

3.1 Test Case

We consider a test case with a true parameter set and synthetic data. This data is used for the NCG algorithm applied on the ODE system of equations. A 5D interpolation is performed to obtain the data function at all the points in the interval $[t_1, \ldots, t_n]$. We used the true parameter set $\theta = (1.1, 1.6, 1.0, 1.0, 0.1, 1.0)$ for our test case. The time interval is chosen to be $t = [0,30]$ with $N = 10$ steps. The initial guess of the parameter set is (0.1,0.1,0.1,0.1,0.1,0.1).

3.1.1 Results

The optimal parameter set obtained for the above test case is $\theta^* = (2, 1.5, 0.7, 1.8, 0.4, 1.6)$. The results of the sensitivity analysis are below.

Table 3.1: Sensitivity analysis of test case

Parameter	p-value	PRCC value
d	6.3e-8	-0.77
l	1e-27	0.99
s	7e-6	0.72
p	0.058	-0.07
k	0.70	-0.34
q	0.07	0.18

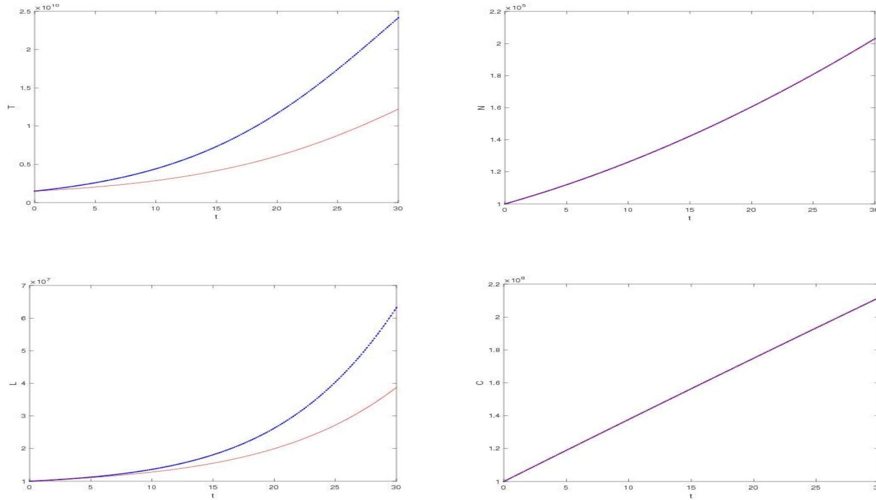

Figure 3.1: Evolution of T, N, L, C with true (red) and optimal (blue) parameter set

The plots show the ODE system estimated parameters show similar evolution of T, N, L, and C with respect to time. This shows that the optimal parameter set predicted is reasonably accurate. The p-values of the parameters show that d, l, and s are the most sensitive parameters, with the PRCC values showing that l is more sensitive that d and s.

CHAPTER 4

CONCLUSION

Numerical optimization techniques have been utilized to estimate patient specific parameters, which are present as coefficients of the governing system of ODEs previously discussed. The Dai-Yuan nonlinear conjugate gradient optimization was used to obtain the optimal parameter set. A test case was considered by generating data from a true parameter set. The forward solver applied on the optimal parameter set and the true parameter set have comparable characteristics, showing that the parameters can be predicted with high accuracy. This shows that parameter estimation is a viable cost-effective alternative to clinical testing to measure the parameters. A future direction to further this work would be to find optimal drug dosages for individual cases. This would provide a mechanism where we are able to propose individual treatment for colon cancer. This could significantly reduce the number of colon-cancer related deaths. Another direction of further research would be to improve the parameter estimation technique by exploring other frameworks of colon cancer that may better model the system. Improving the accuracy of parameter estimation may provide more optimal treatment strategies.

APPENDIX A

PYTHON SCRIPT OF OPTIMIZATION ALGORITHM

The Python 3 script used to perform the algorithms outlined in chapter 2 are provided below.

1. Parameters

```
import numpy as np
import math

def parameters():

    a = 0.531;
    b = 0.021;
    c = 5.2*pow(10,-9);
    e = 0.11;
    f = 0.01;
    j = 1.245*pow(10,-4);
    m = 5*pow(10,-3);
    r1 = 5.2*pow(10,-2);
    r2 = pow(10,-5);
    alpha = 0.18;
    beta = 6.3*pow(10,-3);


    # No. of time interval points
    Nt = 200;
    t = np.linspace(0, 20, num = Nt, endpoint=True)
    step = t[2]-t[1];
    step = (1-math.exp(-50*step))/50

    # Regularization parameter
    nu = 0.001;

    return a, b, c, e, f, j, m, r1, r2, alpha, beta, t, Nt, step, nu
```

2. Forward Model

```
import numpy as np
import math

from parameters import parameters


a, b, c, e, f, j, m, r1, r2, alpha, beta, t, Nt, step, nu = parameters()
```

```
def forward_model(X,theta):
    # Y will be the matrix of F(X)
    Y=np.zeros(4)
    #reshaping to make it a column vector
    Y=Y.reshape((4,1))

    d = theta[0];
    l = theta[1];
    s = theta[2];
    p = theta[3];
    k = theta[4];
    q = theta[5];

    #calculating D from given values to make the following line less bulky

    #Individually writing each equation in F(x)
    T = X[0,0]
    N = X[1,0]
    L = X[2,0]
    C = X[3,0]

    divL_T = L/T
    #D_mul = (math.pow(divL_T,l))/(math.pow(divL_T,l)+s)
    D_mul = (d * pow(divL_T,l))/(4*s*pow(10,-3)*pow(200,l) + pow(divL_T,l));



    Y[0,0] = a * T *( 1 - b * T ) - c * N * T - D_mul * T;
    Y[1,0]= e * C - f * N - p * pow(10,-10) * N * T;
    Y[2,0]= m*L + (j * T * L)/(k+T) -  q *pow(10,-8)* L * T + (r1 * N + r2 *
C)*T;
    Y[3,0]= alpha - beta * C;

    return Y
```

3. <u>Forward Solver</u>

```
import numpy as np


from forward_model import forward_model
from parameters import parameters


a, b, c, e, f, j, m, r1, r2, alpha, beta, t, Nt, step, nu = parameters()
```

```
def forward_solver(theta):

        #----------------------------------Zeros_vector---------------------------
        # Values of T,N,L,C at t = 0

    T_0= 1.5
    N_0= 1.0
    L_0= 1.0
    C_0= 1.0


    X = np.array([T_0,N_0,L_0,C_0])
    X = X.reshape(4,1) # Transforming X to a column vector

    sol = np.zeros(4)
    sol = sol.reshape((4,1))
    sol = np.hstack((sol,X))
    sol = np.delete(sol,0,axis=1)




    #------------------------------Explicit Euler----------------------------
    for i in range(1,Nt):
       X = X + np.multiply(step , forward_model(X,theta))
       sol = np.hstack((sol,X))


    total = np.vstack((t,sol))
    f = open("forward_sol.txt","w")
    np.savetxt(f,total,delimiter=' ')
    f.close()

    return sol


#data = np.delete(data,0,axis=1)

#print(data)


# # #----------------------------------Plotting----------------------------
# plt.plot(t,sol[0])
# plt.plot(t,sol[1])
# plt.plot(t,sol[2])
# plt.plot(t,sol[3])
```

```
# total = np.vstack((t,sol))
# f = open("explicit_sol.txt","w")
# np.savetxt(f,total,delimiter=' ')
# f.close()


# plt.xlabel('t')
# plt.ylabel('sol')
# plt.title("Solution using explicit Eulers method")
```

4. Adjoint Model

```
import numpy as np
import math

from parameters import parameters


a, b, c, e, f, j, m, r1, r2, alpha, beta, t, Nt, step, nu = parameters()



def adjoint_model(X,i,f_sol,f_data,theta):
  # Y will be the matrix of F(X)

    d = theta[0];
    l = theta[1];
    s = theta[2];
    p = theta[3];
    k = theta[4];
    q = theta[5];

    Y = np.zeros(4)
    Y = Y.reshape((4,1))

    T_a = X[0,0]
    N_a = X[1,0]
    L_a = X[2,0]
    C_a = X[3,0]

    # Forward solution at the time point t[i]

    T = f_sol[0,i]
    N = f_sol[1,i]
    L = f_sol[2,i]
    C = f_sol[3,i]
```

```
    # Data at the time point t[i]
    T_d = f_data[0,i]
    N_d = f_data[1,i]
    L_d = f_data[2,i]
    C_d = f_data[3,i]

    t_div0 = (d*s*l*(L**l)*(T**l)*T_a)/((s*(T**l)+(L**l))**2)
    t_div1 = -(d*L**l)/(4*s*pow(10,-3)*pow(200,l) * T**l + L**l)

    t_div2 = (k*L*L_a)/(k+T)**2

    dT_a = -a*b*T*T_a + a*(1-b*T)*T_a - c*N*T_a + t_div0+ t_div1 - p*
pow(10,-10) * N *N_a + j*t_div2 - q*pow(10,-8)*L*L_a + (r1*N + r2*C)*L_a
    #np.negative(dT_a)
    Y[0,0] = dT_a - (T-T_d)

    dN_a = -f*N_a  - p*T*N_a -  c*T*T_a + r1*T*L_a
    #np.negative(dN_a)
    Y[1,0] = dN_a - (N-N_d)



    L_div = (l*(L**(1-l)) * (s*(T**l)))/((s*(T**l) + L**l)**2)

    dL_a = m*L_a + j*((T*L_a)/(k+T)) - q*T*L_a + d*T*T_a*L_div
    #np.negative(dL_a)
    Y[2,0] = dL_a - (L-L_d)



    dC_a = -beta*C_a + e*N_a + r2*T*L_a
    #np.negative(dC_a)
    Y[3,0] = dC_a - (C-C_d)


    return Y
```

5. Adjoint Solver

```
import numpy as np

from adjoint_model import adjoint_model
from parameters import parameters
```

```
a, b, c, e, f, j, m, r1, r2, alpha, beta, t, Nt, step, nu = parameters()

def adj_solver(f_sol,f_data,theta):

    # Terminal condition
    T_aN = -(f_sol[0,Nt-1]-f_data[0,Nt-1])
    N_aN = -(f_sol[1,Nt-1]-f_data[1,Nt-1])
    L_aN = -(f_sol[2,Nt-1]-f_data[2,Nt-1])
    C_aN = -(f_sol[3,Nt-1]-f_data[3,Nt-1])



    X = np.array([T_aN, N_aN,L_aN,C_aN])
    X = X.reshape((4,1))

    result = np.zeros(4)
    result = result.reshape((4,1))

    result = np.hstack((result,X))



    for i in range(Nt-1,0,-1):
       X = X + np.multiply(step, adjoint_model(X,i-1,f_sol,f_data,theta))
       result = np.hstack((result,X))

    result = np.delete(result,0,axis=1)

    total = np.vstack((t,result))
    f = open("adjoint_sol.txt","w")
    np.savetxt(f,total,delimiter=' ')
    f.close()
    return result

    # print(result)



    # total = np.vstack((t,result))
    # #print("total:",total)
```

6. NCG Algorithm

```
import numpy as np
import math
from numpy import linalg
```

```
from parameters import parameters
from forward_solver import forward_solver
from adj_solver import adj_solver


a, b, c, e, f, j, m, r1, r2, alpha, beta, t, Nt, step, nu = parameters()

def gradient(f_sol,a_sol,theta):

    d = theta[0];
    l = theta[1];
    s = theta[2];
    p = theta[3];
    k = theta[4];
    q = theta[5];

    # grad(J) matrix
    grad = np.zeros([6])

    for i in range(0,Nt):

        # Forward solution at t[i]
        T = f_sol[0,i]
        N = f_sol[1,i]
        L = f_sol[2,i]
        C = f_sol[3,i]

        # Adjoint solution at t[i]
        T_a = a_sol[0,i]
        N_a = a_sol[1,i]
        L_a = a_sol[2,i]
        C_a = a_sol[3,i]


        LT_div = L/T

        LT_div_exp = LT_div**l

        #partial derviative J/d
        grad[0] = grad[0] + (LT_div_exp * T * T_a) / (4*s*pow(10,-3)*pow(200,l) +
LT_div_exp) + nu*d
        grad[0] = grad[0] * step

        #partial derviative J/l
```

```
        grad[1] = grad[1] + (d * s * T_a * (LT_div_exp * math.log(LT_div) +
4*s*pow(10,-3)*pow(200,l) * math.log(200)))/ (4*s*pow(10,-3)*pow(200,l) +
LT_div_exp ) **2 + nu*l
        grad[1] = grad[1] * step

        #partial derviative J/s
        grad[2] = grad[2] -(d * LT_div_exp * T * T_a * 4*pow(10,-3)*pow(200,l))/
(4*s*pow(10,-3)*pow(200,l) + LT_div_exp)**2 + nu*s
        grad[2] = grad[2] * step

        #partial derviative J/p
        grad[3] = grad[3] + pow(10,-10)*N * N_a * T + nu*p
        grad[3] = grad[3] * step

        #partial derivative J/k
        grad[4] = grad[4] + (j * L * L_a * T)/ (k + T)**2 + nu*k
        grad[4] = grad[4] * step

        #partial derivative J/q
        grad[5] = grad[5] + pow(10,-8)*L * L_a * T + nu*q
        grad[5] = grad[5] * step

    return grad



# Functional
def J(f_sol,f_data,theta):

    d = theta[0];
    l = theta[1];
    s = theta[2];
    p = theta[3];
    k = theta[4];
    q = theta[5];

    result = 0.0

    for k in range(0,Nt):
        result = result + (f_sol[0,k] - f_data[0,k])**2 + (f_sol[1,k] - f_data[1,k])**2
+ (f_sol[2,k] - f_data[2,k])**2 + (f_sol[3,k] - f_data[3,k])**2

    regularization_term = nu*(d**2 + l**2 + s**2 + p**2 + k**2 + q**2)
    result = 0.5*(result*step + regularization_term)

    return result
```

```
def armijo_line_search(theta,grad,des,f_sol,f_data):

    #index
    j = 0

    #max
    j_max = 10

    alpha = 0.5
    c = 0.25

    while j < j_max:
        theta_new = theta + alpha*des
        f_new = forward_solver(theta_new)

        d1 = J(f_new,f_data,theta_new)
        d2 = J(f_sol,f_data,theta) + c * alpha * np.inner(grad,des)

        if d1 <= d2:
            return alpha
            quit()
        else:
            alpha = alpha/2.0
            j = j+1

    if j == j_max:
        alpha = 0.0
        return alpha



def Fletcher_Reeves(grad_old,grad):

    vec = grad_old
    vec2 = grad

    #numerator = np.dot(vec,vec2)
    numerator = linalg.norm(vec2)**2

    #denominator = np.dot(vec2,vec2)
    denominator = linalg.norm(vec)**2
    result = numerator/denominator

    return result

def Dai_Yuan(grad_old,grad,des):
```

```
    vec = grad_old
    vec2 = grad

    #numerator = np.dot(vec,vec2)
    numerator = linalg.norm(grad)**2

    #denominator = np.dot(vec2,vec2)
    denominator = np.inner(des,grad-grad_old)
    result = numerator/denominator

    return result



# Starting the NCG algorithm

def NCG(theta,f_data):

    #index
    k = 0

    #max
    k_max = 50

    #tolerance
    tol = 10**(-5)

    f_sol = forward_solver(theta)
    a_sol = adj_solver(f_sol,f_data,theta)
    des = np.zeros([6])

    # Computing the gradient
    grad = gradient(f_sol,a_sol,theta)
    grad_norm = linalg.norm(grad)
    des0 = -grad

    des = des0



    while k < k_max:

        # Obtaining alpha through the line search algorithm
        alpha = armijo_line_search(theta,grad,des,f_sol,f_data)
        if (alpha == 0):
            print ('Line search fails')
```

```
            break

        # Updating the parameter
        theta_old = theta

        theta = theta_old + alpha * des


        # Projection Step
        theta[0] = min(2,max(theta[0],0))
        theta[1] = min(2,max(theta[0],0))
        theta[2] = min(3,max(theta[0],0))
        theta[3] = min(1.5,max(theta[0],0))
        theta[4] = min(0.5,max(theta[0],0))
        theta[5] = min(1.5,max(theta[0],0))


        # Old gradient
        grad_old = grad

        # New updates

        f_sol = forward_solver(theta)
        a_sol = adj_solver(f_sol,f_data,theta)

        # Computing the new gradient
        grad = gradient(f_sol,a_sol,theta)



        # Updating the conjugate directions
        #beta = Fletcher_Reeves(grad_old,grad)
        beta = Dai_Yuan(grad_old,grad,des)



        des_old = des;
        des = -grad + np.multiply(beta,des_old)

        grad_norm = linalg.norm(grad)

        print('k = ',k, ', Alpha = ', alpha, ', Iterate value  = ', theta, ', Gradient Norm =
',grad_norm, 'J = ', J(f_sol,f_data,theta))
```

```
    if (linalg.norm(grad) < tol or k == k_max):
        exit()

    k += 1


return theta
```

REFERENCES

[1] “Colorectal Cancer Statistics: How Common Is Colorectal Cancer?” Key Statistics for Colorectal Cancer, American Cancer Society, 12 Jan. 2021,

[2] L.G. dePillis, H. Savage, A.E. Radunskaya, “Mathematical Model of Colorectal Cancer with Monoclonal Antibody Treatments”*, British Journal of Medicine & Medical Research* 4(16): 3101-3131, 2014

[3] Computational Optimization of Systems Governed by Partial Differential Equations. 2012, Alfio Borzí, Volker Schulz.

[4] Nurujjaman, Md. (2020). Enhanced Euler's Method to Solve First Order Ordinary Differential Equations with Better Accuracy. 10.5281/zenodo.3731020.

[5] Y. H. Dai and Y. Yuan, “A Nonlinear Conjugate Gradient Method with a Strong Global Convergence Property”, SIAM Journal on Optimization 1999 10:1, 177-182

[6] S. Marino, I. B. Hogue, C. J. Ray and D. E. Kirschner. A methodology for performing global uncertainty and sensitivity analysis in systems biology. Journal of Theoretical Biology, 254(1):178–196, 2008.

[7] S. Roy, S. Pal, A. Manoj, S. Kakarla, J. Villegas, M. Alajmi, a Fokker-Planck framework for parameter estimation and sensitivity analysis in colon cancer, AIP Conference Proceedings, 2021.

BIOGRAPHICAL INFORMATION

Achyuth Manoj is a senior international student at the University of Texas at Arlington majoring in Physics with a second major in Mathematics. His research interests are in Condensed Matter Physics and in Space Physics. He has completed one Honors senior project on predicting band gaps in transition metal oxides and has co-authored a paper on parameter estimation and sensitivity analysis on colon cancer. He plans to attend graduate school after completing his bachelor's degree.