Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра вычислительных систем

**ОТЧЕТ**

по курсовой работе

по дисциплине «**Вычислительная математика**»

|  |  |  |
| --- | --- | --- |
| Выполнил:  студент гр. ИВ-121  «4» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Бессонов А.Е./ |
|  |  |  |
| Проверил:  ассистент кафедры ПМиК  «4» мая 2023 г. | \_\_\_\_\_\_\_\_\_\_\_\_ | /Агалаков А.А./ |

Оценка «\_\_\_\_\_\_\_\_\_\_\_\_\_»

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[ЗАДАНИЕ 3](#_Toc133909641)

[РЕАЛИЗАЦИЯ 4](#_Toc133909644)

[РЕЗУЛЬТАТЫ](#_Toc133909645) 9

[ЛИСТИНГ ПРОГРАММЫ 12](#_Toc133909647)

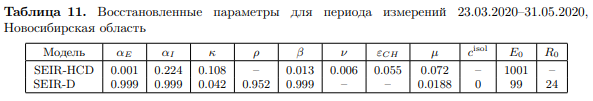
# **ЗАДАНИЕ**

Решите систему уравнений (5) модель SEIR-D для Новосибирской области с коэффициентами из таблицы 11. Решение найдите с помощью метода Эйлера на участке времени от 0 до 90 дней с точностью до 2 знака после запятой.

Описываемая модель SEIR-D:



Данные в таблице 11:



Начальные данные:



**РЕАЛИЗАЦИЯ**

**Начальные данные**

Имеем следующие данные: размер шага step = 0,01, первый день отсчета – t = 0, второй день отсчета – T = 90, все люди – N0 = 2.798.170, восприимчивые люди – S, S0 = N0 – E0 – R0, бессимптомно инфицированные – E, E0 = 99, инфицированные с симптомами – I, I0 = 0, выздоровевшие (или с хорошим иммунитетом) – R, R0 = 24, умершие – D, D0 = 0. Усовершенствованая формула Элера, по которой будут сделаны расчёты.

Изображение выглядит как диаграмма

Автоматически созданное описание

**Реализация функций**

int eler(double \*S, double\* E, double\* I, double\* R, double\* D, double \*t)

Эта функция - это реализация улучшенного метода Эйлера для решения модели SEIR-D в C. Она принимает указатели на массивы, содержащие начальные значения S, E, I, R, D и t, а также параметры модели.

Функция инициализирует массивы начальными значениями, а затем выполняет итерации в цикле для заданного количества шагов времени. На каждом шаге времени она вычисляет значения dS, dE, dI, dR и dD, используя текущие значения S, E, I, R и D.

Затем она использует метод Эйлера, чтобы рассчитать оценку значений S, E, I, R и D в конце шага времени. Затем она вычисляет вторую оценку, используя тот же метод, но с значениями, рассчитанными в первой оценке.

Наконец, она вычисляет среднее значение двух оценок и обновляет значения S, E, I, R и D с помощью средних значений.

Функция возвращает 0 по завершении.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | int eler(double \*S, double\* E, double\* I, double\* R, double\* D, double \*t){      S[0] = N0 - E0 - R0;      E[0] = E0;      I[0] = I0;      R[0] = R0;      D[0] = D0;      t[0] = t0;      for (int i = 0; i < num\_steps; i++) {          double N = S[i] + E[i] + I[i] + R[i] + D[i];          double dS = dSdt(S[i], E[i], I[i], R[i], N);          double dE = dEdt(S[i], E[i], I[i], N);          double dI = dIdt(E[i], I[i]);          double dR = dRdt(E[i], I[i], R[i]);          double dD = dDdt(I[i]);          S[i + 1] = S[i] + dt \* dS;          E[i + 1] = E[i] + dt \* dE;          I[i + 1] = I[i] + dt \* dI;          R[i + 1] = R[i] + dt \* dR;          D[i + 1] = D[i] + dt \* dD;          t[i + 1] = t[i] + dt;          double dS1 = dSdt(S[i+1], E[i+1], I[i+1], R[i+1], N);          double dE1 = dEdt(S[i+1], E[i+1], I[i+1], N);          double dI1 = dIdt(E[i+1], I[i+1]);          double dR1 = dRdt(E[i+1], I[i+1], R[i+1]);          double dD1 = dDdt(I[i+1]);          double deltaS = 0.5 \* dt \* (dS + dS1);          double deltaE = 0.5 \* dt \* (dE + dE1);          double deltaI = 0.5 \* dt \* (dI + dI1);          double deltaR = 0.5 \* dt \* (dR + dR1);          double deltaD = 0.5 \* dt \* (dD + dD1);          S[i] += deltaS;          E[i] += deltaE;          I[i] += deltaI;          R[i] += deltaR;          D[i] += deltaD;      }      return 0;  } |

int print\_res(double \*S, double\* E, double\* I, double\* R, double\* D, double \*t, lxw\_worksheet \*worksheet)

Данная функция выводит результаты решения системы уравнений модели SEIR-D на экран и записывает их в указанный лист Excel-файла с помощью библиотеки libxlsxwriter. Функция принимает шесть аргументов: указатели на массивы S, E, I, R, D и t, которые содержат значения функций S(t), E(t), I(t), R(t), D(t) и t на интервале времени от 0 до 90 дней с шагом 0,1 дня, а также объект lxw\_worksheet, представляющий лист Excel, в который нужно записать результаты.

Функция начинает с вывода заголовка таблицы на экран и записи заголовка в первую строку листа Excel. Затем она перебирает все элементы массивов S, E, I, R, D и t с помощью цикла for и выводит их значения на экран с точностью до двух знаков после запятой. Каждая строка таблицы соответствует моменту времени t[i] и содержит значения функций S(t[i]), E(t[i]), I(t[i]), R(t[i]) и D(t[i]).

Кроме того, функция проверяет, является ли текущий индекс i кратным 100, и если да, то записывает соответствующую строку таблицы в лист Excel с помощью функции worksheet\_write\_number(). Здесь переменная o используется для отслеживания номера строки в листе Excel.

В результате выполнения функции на экране будет выведена таблица с результатами решения системы уравнений модели SEIR-D, а также эти результаты будут записаны в указанный лист Excel-файла.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | int print\_res(double \*S, double\* E, double\* I, double\* R, double\* D, double \*t, lxw\_worksheet \*worksheet){      int o = 0;      printf("t,S,E,I,R,D\n");      for (int i = 0; i < num\_steps + 1; i++) {          if(i % 100 == 0){              o++;              worksheet\_write\_number(worksheet, o, 0, t[i], NULL);              worksheet\_write\_number(worksheet, o, 1, S[i], NULL);              worksheet\_write\_number(worksheet, o, 2, E[i], NULL);              worksheet\_write\_number(worksheet, o, 3, I[i], NULL);              worksheet\_write\_number(worksheet, o, 4, R[i], NULL);              worksheet\_write\_number(worksheet, o, 5, D[i], NULL);          }          printf("%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n", t[i], S[i], E[i], I[i], R[i], D[i]);      }      return 0;  } |

Эта функция возвращает значение коэффициента c, который используется в формуле для вычисления приращения числа инфицированных людей в модели SEIR-D при использовании метода улучшенного Эйлера. Значение коэффициента c зависит от значения параметра a и значения коэффициента c\_isol. .

Значение коэффициента c\_isol - это коэффициент изоляции, который определяет, какая часть населения находится в изоляции и не может заразиться вирусом.

В целом, эта функция используется для вычисления коэффициента c, который является частью формулы для вычисления приращения числа инфицированных людей в модели SEIR-D.

|  |  |
| --- | --- |
| 1  2  3 | double c() {      return 1.0 + c\_isol \* ((1.0 - 2.0 / 5.0) \* a);  } |

Эта функция вычисляет производную количества умерших (D) по времени на основе текущего количества инфицированных (I) и коэффициента смертности (mu). Формула для вычисления производной D по времени имеет вид:

dD/dt = mu \* I

Функция принимает текущее количество инфицированных I в качестве аргумента и возвращает значение производной dD/dt.

|  |  |
| --- | --- |
| 1  2  3 | double dDdt(double I) {      return mu \* I;  } |

Эта функция вычисляет производную числа умерших по отношению к времени на основе текущего числа инфицированных и коэффициента смертности.

Функция dRdt вычисляет скорость изменения числа умерших в зависимости от текущего числа инфицированных и выздоровевших, а также коэффициента смертности.

beta - это коэффициент инфицирования, rho - коэффициент выздоровления от инфекции, а \_gamma - коэффициент выздоровления от инфекции с учетом смертности.

Таким образом, выражение beta \* I + rho \* E - \_gamma \* R означает, что производная числа умерших по отношению к времени определяется как сумма числа умерших, вызванных текущим числом инфицированных, плюс количество выздоровивших, которые впоследствии умирают, за вычетом числа выздоровевших, которые не умирают.

|  |  |
| --- | --- |
| 1  2  3 | double dRdt(double E, double I, double R) {      return beta \* I + rho \* E - \_gamma \* R;  } |

Функция dIdt вычисляет производную числа инфицированных по отношению к времени на основе текущего числа подверженных риску контакта людей, коэффициента инфицирования, коэффициента выздоровления, а также коэффициента смертности.

kappa - это коэффициент, определяющий скорость появления новых инфицированных, beta - коэффициент инфицирования, mu - коэффициент смертности инфицированных, а E и I - текущее число подверженных риску контакта людей и текущее число инфицированных соответственно.

Таким образом, выражение kappa \* E - beta \* I - mu \* I означает, что производная числа инфицированных по отношению к времени определяется как скорость появления новых инфицированных, уменьшенная на количество инфицированных, которые выздоравливают или умирают.

|  |  |
| --- | --- |
| 1  2  3 | double dIdt(double E, double I) {      return kappa \* E - beta \* I - mu \* I;  } |

Функция dEdt вычисляет производную числа подверженных риску контакта людей по отношению к времени на основе текущего числа подверженных риску контакта людей, текущего числа инфицированных, текущего числа подверженных риску контакта людей, которые уже заражены, и общего числа людей в популяции.

S - текущее число подверженных риску контакта людей, E - текущее число подверженных риску контакта людей, которые уже заражены, I - текущее число инфицированных, N - общее число людей в популяции, kappa - коэффициент выздоровления, rho - коэффициент выздоровления от инфекции, c() - коэффициент перехода от восприимчивых к подверженным риску контакта людям, alpha\_I - коэффициент инфицирования от инфицированных, alpha\_E - коэффициент инфицирования от подверженных риску контакта людей, которые уже заражены.

Таким образом, выражение c() \* (alpha\_I \* S \* I / N + alpha\_E \* S \* E / N) - (kappa + rho) \* E означает, что производная числа подверженных риску контакта людей по отношению к времени определяется как скорость появления новых инфицированных, увеличенная на скорость появления новых подверженных риску контакта людей, которые уже заражены, уменьшенная на количество подверженных риску контакта людей, которые становятся инфицированными, и на количество подверженных риску контакта людей, которые выздоравливают.

|  |  |
| --- | --- |
| 1  2  3  4 | double dEdt(double S, double E, double I, double N) {      return c() \* (alpha\_I \* S \* I / N + alpha\_E \* S \* E / N) - (kappa + rho) \* E;  } |

Функция dSdt вычисляет производную числа восприимчивых людей по отношению к времени на основе текущего числа восприимчивых людей, текущего числа подверженных риску контакта людей, текущего числа инфицированных, текущего числа выздоровевших и общего числа людей в популяции.

S - текущее число восприимчивых людей, E - текущее число подверженных риску контакта людей, которые уже заражены, I - текущее число инфицированных, R - текущее число выздоровевших, N - общее число людей в популяции, c() - коэффициент перехода от восприимчивых к подверженным риску контакта людям, alpha\_I - коэффициент инфицирования от инфицированных, alpha\_E - коэффициент инфицирования от подверженных риску контакта людей, которые уже заражены, \_gamma - коэффициент выздоровления от инфекции с учетом смертности.

Таким образом, выражение -c() \* (alpha\_I \* S \* I / N + alpha\_E \* S \* E / N) + \_gamma \* R означает, что производная числа восприимчивых людей по отношению к времени определяется как скорость появления новых инфицированных, увеличенная на скорость появления новых подверженных риску контакта людей, которые уже заражены, уменьшенная на количество восприимчивых людей, которые становятся инфицированными, и на количество выздоровевших, которые больше не могут быть инфицированы.

|  |  |
| --- | --- |
| 1  2  3 | double dSdt(double S, double E, double I, double R, double N) {      return -c() \* (alpha\_I \* S \* I / N + alpha\_E \* S \* E / N) + \_gamma \* R;  } |

**РЕЗУЛЬТАТЫ**

Как итог, построены 5 графиков с зависимостями количества дней от количество каждого из параметров S, E, I, R, D:

Графики были построены по результатам программы, записаных в Excel.

# **ЛИСТИНГ ПРОГРАММЫ**

**Program.c:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147 | #include <stdio.h>  #include <math.h>  #include <xlsxwriter.h>  #define t0 0.0  #define T 90.0  #define dt 0.01  #define num\_steps ((int) ((T - t0) / dt))  double a = 1.0;  const double alpha\_E = 0.999;  const double alpha\_I = 0.999;  const double kappa = 0.042;  const double rho = 0.952;  const double beta = 0.999;  const double mu = 0.0188;  const double c\_isol = 0;  const double E0 = 99;  const double R0 = 24;  const double N0 = 2798170.0;  const double S0 = N0 - E0 - R0;  const double I0 = 0.0;  const double D0 = 0.0;  const double \_gamma = 0.0;  double c() {      return 1.0 + c\_isol \* ((1.0 - 2.0 / 5.0) \* a);  }  double dSdt(double S, double E, double I, double R, double N) {      return -c() \* (alpha\_I \* S \* I / N + alpha\_E \* S \* E / N) + \_gamma \* R;  }  double dEdt(double S, double E, double I, double N) {      return c() \* (alpha\_I \* S \* I / N + alpha\_E \* S \* E / N) - (kappa + rho) \* E;  }  double dIdt(double E, double I) {      return kappa \* E - beta \* I - mu \* I;  }  double dRdt(double E, double I, double R) {      return beta \* I + rho \* E - \_gamma \* R;  }  double dDdt(double I) {      return mu \* I;  }  int eler(double \*S, double\* E, double\* I, double\* R, double\* D, double \*t){      S[0] = N0 - E0 - R0;      E[0] = E0;      I[0] = I0;      R[0] = R0;      D[0] = D0;      t[0] = t0;      for (int i = 0; i < num\_steps; i++) {          double N = S[i] + E[i] + I[i] + R[i] + D[i];          double dS = dSdt(S[i], E[i], I[i], R[i], N);          double dE = dEdt(S[i], E[i], I[i], N);          double dI = dIdt(E[i], I[i]);          double dR = dRdt(E[i], I[i], R[i]);          double dD = dDdt(I[i]);          S[i + 1] = S[i] + dt \* dS;          E[i + 1] = E[i] + dt \* dE;          I[i + 1] = I[i] + dt \* dI;          R[i + 1] = R[i] + dt \* dR;          D[i + 1] = D[i] + dt \* dD;          t[i + 1] = t[i] + dt;          double dS1 = dSdt(S[i+1], E[i+1], I[i+1], R[i+1], N);          double dE1 = dEdt(S[i+1], E[i+1], I[i+1], N);          double dI1 = dIdt(E[i+1], I[i+1]);          double dR1 = dRdt(E[i+1], I[i+1], R[i+1]);          double dD1 = dDdt(I[i+1]);          double deltaS = 0.5 \* dt \* (dS + dS1);          double deltaE = 0.5 \* dt \* (dE + dE1);          double deltaI = 0.5 \* dt \* (dI + dI1);          double deltaR = 0.5 \* dt \* (dR + dR1);          double deltaD = 0.5 \* dt \* (dD + dD1);          S[i] += deltaS;          E[i] += deltaE;          I[i] += deltaI;          R[i] += deltaR;          D[i] += deltaD;      }      return 0;  }  int print\_res(double \*S, double\* E, double\* I, double\* R, double\* D, double \*t, lxw\_worksheet \*worksheet){      int o = 0;      printf("t,S,E,I,R,D\n");      for (int i = 0; i < num\_steps + 1; i++) {          if(i % 100 == 0){              o++;              worksheet\_write\_number(worksheet, o, 0, t[i], NULL);              worksheet\_write\_number(worksheet, o, 1, S[i], NULL);              worksheet\_write\_number(worksheet, o, 2, E[i], NULL);              worksheet\_write\_number(worksheet, o, 3, I[i], NULL);              worksheet\_write\_number(worksheet, o, 4, R[i], NULL);              worksheet\_write\_number(worksheet, o, 5, D[i], NULL);          }          printf("%.2f,%.2f,%.2f,%.2f,%.2f,%.2f\n", t[i], S[i], E[i], I[i], R[i], D[i]);      }      return 0;  }  int main() {      double S[num\_steps + 1], E[num\_steps + 1], I[num\_steps + 1], R[num\_steps + 1], D[num\_steps + 1];      double t[num\_steps + 1];      // Создаем новый файл Excel      lxw\_workbook \*workbook = workbook\_new("table.xlsx");      // Создаем новый лист      lxw\_worksheet \*worksheet = workbook\_add\_worksheet(workbook, NULL);      // Задаем заголовки столбцов      worksheet\_write\_string(worksheet, 0, 0, "t", NULL);      worksheet\_write\_string(worksheet, 0, 1, "S", NULL);      worksheet\_write\_string(worksheet, 0, 2, "E", NULL);      worksheet\_write\_string(worksheet, 0, 3, "I", NULL);      worksheet\_write\_string(worksheet, 0, 4, "R", NULL);      worksheet\_write\_string(worksheet, 0, 5, "D", NULL);      eler(S, E, I, R, D, t);      print\_res(S, E, I, R, D, t, worksheet);      // Сохраняем файл Excel      workbook\_close(workbook);      return 0;  } |