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

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.
Информацию взять из статьи (ссылка на статью).

Описываемая модель SEIR-D:

В рамках модели SEIR-D распространение коронавируса COVID-19 описывается системой из 5 нелинейных обыкновенных дифференциальных уравнений на отрезке $t \in [t_0, T]$ [31] (схема модели приведена на рис. 1 справа):

$$\begin{cases} \dfrac{dS}{dt} = -c(t-\tau)\left(\dfrac{\alpha_I S(t)I(t)}{N} + \dfrac{\alpha_E S(t)E(t)}{N}\right) + \gamma R(t), \\ \dfrac{dE}{dt} = c(t-\tau)\left(\dfrac{\alpha_I S(t)I(t)}{N} + \dfrac{\alpha_E S(t)E(t)}{N}\right) - (\kappa+\rho)E(t), \\ \dfrac{dI}{dt} = \kappa E(t) - \beta I(t) - \mu I(t), \\ \dfrac{dR}{dt} = \beta I(t) + \rho E(t) - \gamma R(t), \\ \dfrac{dD}{dt} = \mu I(t). \end{cases} \tag{5}$$

Здесь $N = S + E + I + R + D$ — вся популяция.

Функция, использующая ограничения на передвижения граждан:

$$c(t) = 1 + c^{\text{isol}}\left(1 - \frac{2}{5}a(t)\right), \quad c(t) \in (0, 2).$$

Начальные данные:

$$S(t_0) = S_0, \quad E(t_0) = E_0, \quad I(t_0) = I_0, \quad R(t_0) = R_0, \quad D(t_0) = D_0. \tag{6}$$

Данные в таблице 11:

Таблица 11. Восстановленные параметры для периода измерений 23.03.2020–31.05.2020, Новосибирская область

Модель	α_E	α_I	κ	ρ	β	ν	ε_{CH}	μ	c^{isol}	E_0	R_0
SEIR-HCD	0.001	0.224	0.108	–	0.013	0.006	0.055	0.072	–	1001	–
SEIR-D	0.999	0.999	0.042	0.952	0.999	–	–	0.0188	0	99	24

Начальные данные:

а для математической модели SEIR-D в следующем виде:

$$S_0 = 2\ 798\ 170 - q_8 - q_9, \quad E_0 = q_8, \quad I_0 = 0, \quad R_0 = q_9, \quad D_0 = 0.$$

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.

$$y_{i+1} = y_i + \Delta y_i, \text{ где } \Delta y_i = hf\left(x_i + \frac{h}{2}; y_i + \frac{h}{2} f(x_i; y_i)\right)$$

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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

```
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.

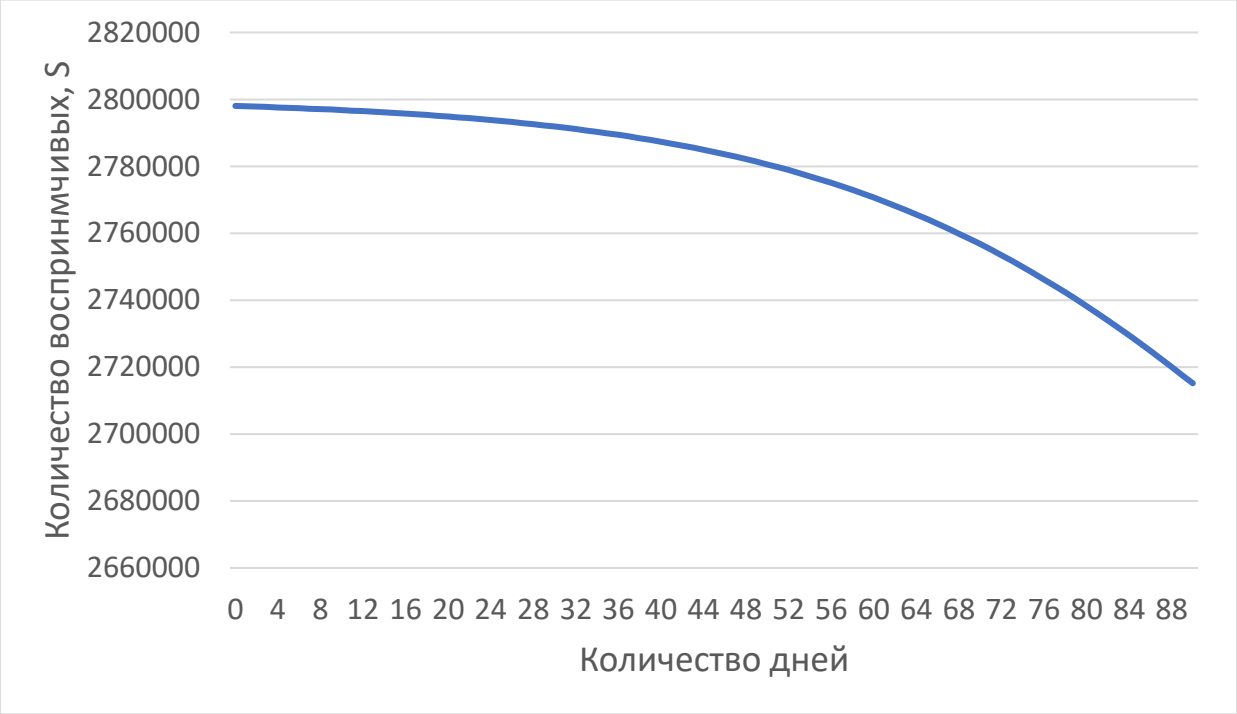
S - текущее число восприимчивых людей, E - текущее число подверженных риску контакта людей, которые уже заражены, I - текущее число инфицированных, R - текущее число выздоровевших, N - общее число людей в популяции, c() - коэффициент перехода от восприимчивых к подверженным риску контакта людям, alpha_I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

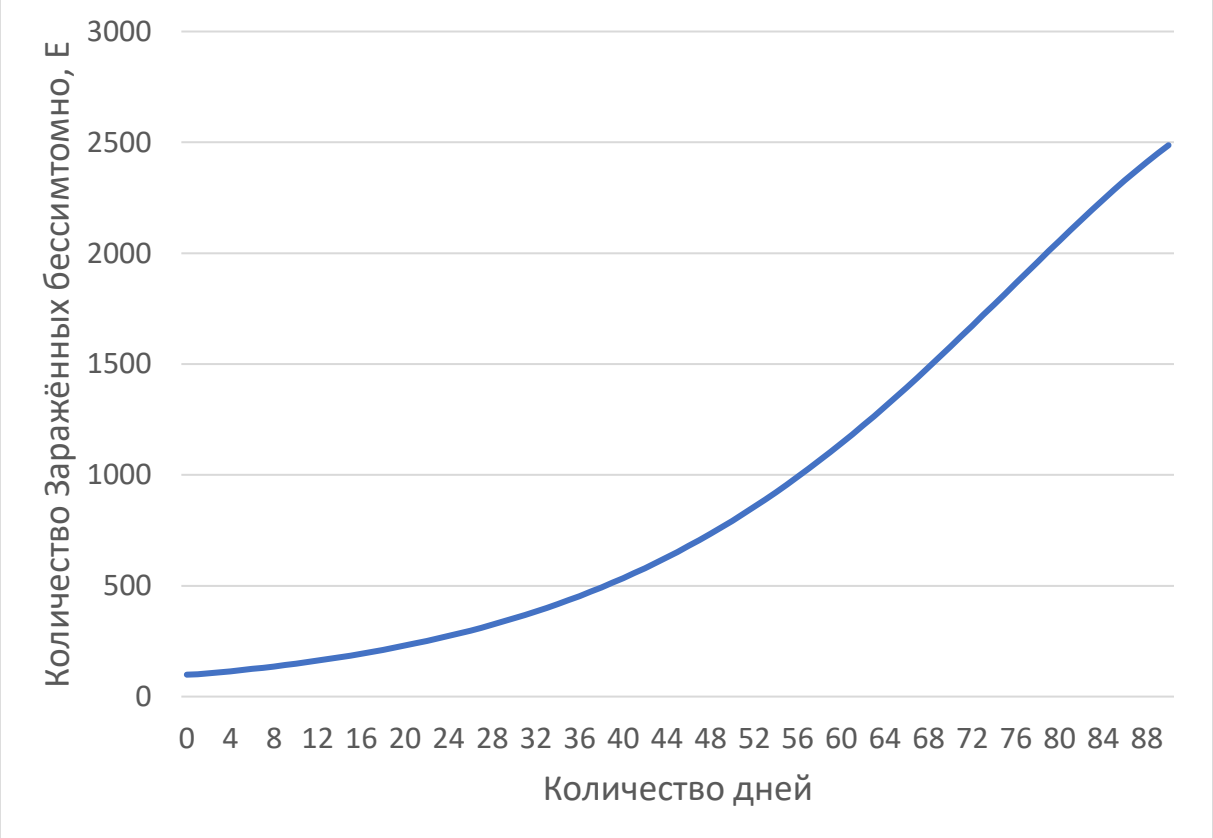
Таким образом, выражение -c() * (alpha_I * S * I / N + alpha_E * S * E / N) + _gamma * R означает, что производная числа восприимчивых людей по отношению к времени определяется как скорость появления новых инфицированных, увеличенная на скорость появления новых подверженных риску контакта людей, которые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

```
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

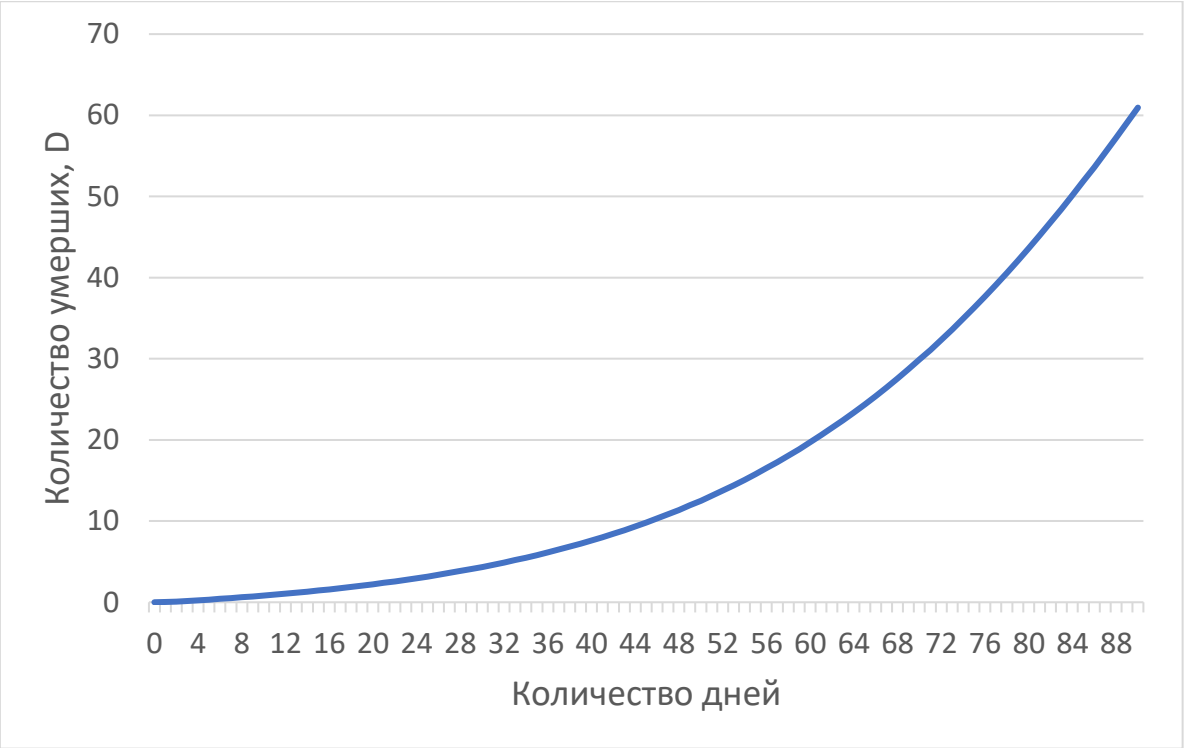
Как итог, построены 5 графиков с зависимостями количества дней от количество каждого из параметров S, E, I, R, D:






Количество инфицированных симптомами. I
120
100
80
60
40
20
0
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88
Количество дней


Количество вылеченных, R
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0 4 8 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72 76 80 84 88
Количество дней



Графики были построены по результатам программы, записаных в Excel.

ЛИСТИНГ ПРОГРАММЫ

Program.c:

```c
#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;
}
```

```c

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;
}

```