Лр №6. решение обыкновенных дифференциальных уравнений.

1. Решить ОДУ методом Эйлера.

2. Решить ОДУ модифицированным методом Эйлера.

3. Решить ОДУ методом Рунге Кутта.

Варианты дифференциальных уравнений.

1 у'=(xy2+x)/(y-x2y)

2 у'=cos(t)-y

**3 у'=(1-2x)/y2**

4 y'=ebx-ay

5 у'=(1-x2)/xy

6 у'=-2y/(y2-6x)

7 у'=(y2-y)/x

8 у'=1/(2x-y2)

9 y'=(1+y)/(tg(x)

10 у'=ex-1

**Решение ОДУ методом Эйлера.**

Рассмотрим дифференциальное уравнение

 (1)

с начальным условием



Подставив в уравнение (1), получим значение производной в точке :



При малом имеет место:



Обозначив  , перепишем последнее равенство в виде:

 (2)

Принимая теперь за новую исходную точку, точно также получим:



**В общем случае будем иметь:**



Это и есть метод Эйлера. Величина называется шагом интегрирования. Пользуясь этим методом, мы получаем приближенные значения у , так как производная на самом деле не остается постоянной на промежутке длиной . Поэтому мы получаем ошибку в определении значения функции у , тем большую, чем больше . Метод Эйлера является простейшим методом численного интегрирования дифференциальных уравнений и систем. Его недостатки – малая точность и систематическое накопление ошибок.

**Листинг программы метода Эйлера**

//Решение ДУ методом Эйлера



//Драйвер решения ДУ методом Эйлера

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

float F(float x);

float f1(float x);

float Eyler(float x[],float y[],int n);

int main()

{

int i;

const int n=10;

float x[n],y[n],yr[n];

float r,;

for(i=0;i<n;i++)

{

x[i]=0.1\*i;

yr[i]=F(x[i]);

}

y[0]=1;

r=Eyler(x,y,n);

for(i=0;i<n;i++)

printf("x= %f y= %f yr= %f\n",x[i],y[i],yr[i]);

system("pause");

return 0;

}

//Функция вычисления решения ДУ

float F(float x)

{return (2\*exp(x)-x-1);}

//Функция вычисления правых частей ДУ

float f(float x,float y)

{return(x+y);}

//Функция решения ДУ методом Эйлера

float Eyler(float x[],float y[],int n)

{

int i;

float h=x[1]-x[0];

for(i=0; i<n; i++)

y[i+1]=y[i]+ h\*f(x[i],y[i]);

return (0.);

}

**Модифицированный метод Эйлера**

Более точным является модифицированный метод Эйлера с пересчетом. Его суть в том, что сначала по формуле (3) находят так называемое «грубое приближение» (прогноз):



а затем пересчетом получают тоже приближенное, но более точное значение (коррекция):

 (4)

Фактически пересчет позволяет учесть, хоть и приблизительно, изменение производной на шаге интегрирования , так как учитываются ее значения в начале и в конце шага (рис. 1), а затем берется их среднее. Метод Эйлера с пересчетом (4) является по существу методом Рунге-Кутта 2-го порядка [2], что станет очевидным из дальнейшего.

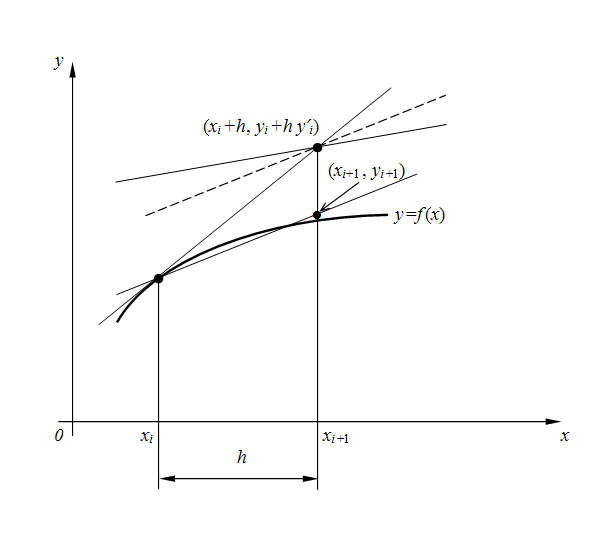


Рис. 1. Геометрическое представление метода Эйлера с пересчетом.

Листинг Модифицированного метода Эйлера

// Модифицированный метод Эйлера

//Драйвер Модифицированного метода Эйлера

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

float F(float x);

float f1(float x);

float EylerM(float x[],float y[],int n);

int main()

{ int N=1000;

int i;

const int n=10;

float x[n],y[n],yr[n];

float r,s;

for(i=0;i<n;i++)

{

x[i]=0.1\*i;

yr[i]=F(x[i]);

}

y[0]=1;

r=EylerM(x,y,n);

for(i=0;i<n;i++)

printf("x= %f y= %f yr= %f\n",x[i],y[i],yr[i]);

system("pause");

return 0;

}

//Функция вычисления решения ДУ

float F(float x)

{return (2\*exp(x)-x-1);}

//Функция вычисления правых частей ДУ

float f(float x,float y)

{return(x+y);}

//Функция решения ДУ Модифицированным методом Эйлера

float EylerM(float x[],float y[],int n)

{ int i;float h=x[1]-x[0],r;

for(i=0; i<n; i++)

{ r=f(x[i],y[i]);

y[i+1]=y[i]+ h\*r;

y[i+1]=y[i]+h\*(r+f(x[i+1],y[i+1]))/2.;

}

return (0.);

}

**Метод Рунге-Кутта**

Вновь рассмотрим дифференциальное уравнение

 (1)

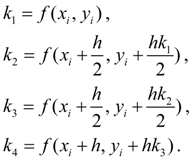
с начальным условием 

Классический метод Рунге-Кутта 4-го порядка описывается следующей системой пяти равенств:



(5)

где



Строго говоря, существует не один, а группа методов Рунге-Кутта, отличающихся друг от друга порядком, т.е. количеством параметров . В данном случае мы имеем метод 4-го порядка, который является одним из наиболее применяемых на практике, так как обеспечивает высокую точность и в то же время отличается сравнительной простотой. Поэтому в большинстве случаев он упоминается в литературе просто как «метод Рунге-Кутта» без указания его порядка.

Листинг Решения ДУ методом Рунге- Кутта.

//// Метод Рунге- Кутта

////Драйвер метода Рунге- Кутта

#include <stdio.h>

#include <stdlib.h>

#include<math.h>

float F(float x);

float f1(float x);

float RKutta(float x[],float y[],int n);

int main()

{ int N=1000;

int i;

const int n=10;

float x[n],y[n],yr[n];

float r,s;

for(i=0;i<n;i++)

{

x[i]=0.1\*i;

yr[i]=F(x[i]);

}

y[0]=1;

r=RKutta(x,y,n);

for(i=0;i<n;i++)

printf("x= %f y= %f yr= %f\n",x[i],y[i],yr[i]);

system("pause");

return 0;

}

//Функция вычисления решения ДУ

float F(float x)

{return (2\*exp(x)-x-1);}

////Функция вычисления правых частей ДУ

float f(float x,float y)

{return(x+y);}

//Функция решения ДУ методом Рунге- Кутта

float RKutta(float x[],float y[],int n)

{ int i;float h2,h=x[1]-x[0];

float k1,k2,k3,k4;

h2=h/2.;

for(i=0; i<n; i++)

{

k1=f(x[i],y[i]);

k2=f(x[i]+h2,y[i]+h2\*k1);

k3=f(x[i]+h2,y[i]+h2\*k2);

k4=f(x[i]+h,y[i]+h\*k3);

y[i+1]=y[i]+ h\*(k1+2.\*k2+2.\*k3+k4)/6.;

}

return (0.);

}