

ПЯВУ. Лекция 14.

Основы программирования.

А.М. Задорожный

Контрольные вопросы

1. Почему в C# рекомендуют использовать свойства вместо полей данных?
2. Что такое рекурсия?
3. В чем заключается метод Гаусса решения системы линейных уравнений?
4. Какие варианты решения системы уравнений могут быть?

План лекции

1. Решение системы уравнений методом Гаусса
 - Вариант с выбором ведущего элемента
2. Вычисление детерминанта методом Гаусса
3. Вычисление числа π методом “Монте-Карло”.

Системы линейных уравнений и Матрицы

$$\begin{cases} a_{11} * x_1 + a_{12} * x_2 + \dots + a_{1N} * x_N = b_1 \\ a_{21} * x_1 + a_{22} * x_2 + \dots + a_{2N} * x_N = b_2 \\ \dots \\ a_{N1} * x_1 + a_{N2} * x_2 + \dots + a_{NN} * x_N = b_N \end{cases}$$

В векторном виде:

$$A * \mathbf{x} = \mathbf{b}$$

$$\mathbf{x} = \{x_1, x_2, \dots, x_N\}$$

Метод Гаусса

$$a'_{11} * x_1 + a'_{12} * x_2 + \dots a'_{1N} * x_N = b'_1$$

$$a'_{22} * x_2 + \dots a'_{2N} * x_N = b'_2$$

...

$$a'_{NN} * x_N = b'_N$$

Представление системы уравнений

Матрицу представим в виде двумерного массива:

```
double [,] M = new double[N, N+1];
```

Число строк матрицы N . Число столбцов $N + 1$, т. к. включает и свободный вектор b .

...

Заполняем матрицу коэффициентами...

Алгоритм вычитания строк

Здесь k – номер строки, которую надо вычитать ...

```
static void SubtractRow(double [,] M, int k)
{
    double m = M[k, k];
    for(int i = k+1; i < M.GetLength(0); i++)
    {
        double t = M[i, k]/m;
        for(int j = k; j < M.GetLength(1); j++)
        {
            M[i, j] = M[i, j] - M[k, j]*t;
        }
    }
}
```

Приведение матрицы к верхнетреугольному виду

```
static void TriangleMatrix(double [,] M)
{
    for(int i = 1; i < M.GetLength(0); i++)
        SubtractRow(M, i-1);
}
```


Решение

Последнее уравнение содержит только 1 неизвестное – x_N . После решения последнего уравнения, можно решить предпоследнее, т.к. после подстановки x_N в нем останется неизвестным только x_{N-1} и т.д.

```
public static double [] Solve(double [,] M)
{
    TriangleMatrix(M);
    double v[] = new double[M.GetLength(0)];
    int Nb = M.GetLength(1)-1;
    for(int n = v.Length-1; n >= 0; n--)
    {
        double sum = 0;
        for(int i = n+1; i < Nb; i++)
            sum += v[i]*M[n, i];
        v[n] = (M[n, Nb]-sum)/M[n, n];
    }
    return v;
}
```

“Слабые” места

SubtractRow:

```
double m = M[k, k];
```

```
...
```

```
M[i, j] = M[i, j] - M[k, j]*t/m;
```

Возможно, m окажется равным 0!

Возможно окажется неравным 0, в результате ошибок вычислений!

Улучшение метода

Выбор “ведущего элемента”.

Идея заключается в том, что бы каждый раз выбирать из оставшихся строк строку с наибольшим элементом в текущей позиции (столбце, который зануляем).

От перестановки уравнений решение системы не изменяется.

Выбор ведущего элемента

```
//Находит строку, в которой элемент n имеет наибольшее по модулю значение,  
// и меняет ее местами со строкой n
```

```
static void SelectLeading(double [,] M, int n)  
{  
    //Найдем номер строки, с наибольшим  
    //элементом в столбце n  
    int iMax = n;  
    for(int i = n+1; i < M.GetLength(0); i++)  
        if(Math.Abs(M[iMax, n]) < Math.Abs(M[i, n]))  
            iMax = i;  
  
    // Переставить строки iMax и n  
    if(iMax != n)  
        for(int i = n; i < M.GetLength(1); i++)  
        {  
            double t = M[n, i];  
            M[n, i] = M[iMax, i];  
            M[iMax, i] = t;  
        }  
}
```

Усовершенствованная триангуляция матрицы

```
static void TriangleMatrix(double [,] M)
{
    for(int i = 1; i < M.GetLength(0); i++)
    {
        SelectLeading(M, i-1);
        SubtractRow(M, i-1);
    }
}
```

Решение есть не всегда

```
static bool TriangleMatrix(double [,] M)
{
    for(int i = 1; i < M.GetLength(0); i++)
    {
        SelectLeading(M, i-1);
        if(Math.Abs(M[i-1, i-1]) > 0.0001)
            SubtractRow(M, i-1);
        else
            return false;
    }
    return true;
}
```

Решение

```
public static double [] Solve(double [,] M)
{
    if(!TriangleMatrix(M))
        return null;
    double v[] = new double[M.GetLength(0)];
    int Nb = M.GetLength(1)-1;
    for(int n = v.Length-1; n >= 0; n--)
    {
        double sum = 0;
        for(int i = n+1; i < Nb; i++)
            sum += v[i]*M[n, i];
        v[n] = (M[n, Nb]-sum)/M[n, n];
    }
    return v;
}
```

Решение

```
double[,] M = new double[4, 5] {  
    { 2, 3, 1, 1, 1 }, { 1, 2, 1, 5, 1 },  
    { 1, 1, 2, 1, 1 }, { 1, 1, 4, 2, 1 } };  
double[] x = Gauss.Solve(M);  
if(x != null)  
    for (int i = 0; i < x.Length; i++)  
        Console.WriteLine(x[i]);  
else  
    Console.WriteLine("Единственного решения системы нет.");
```


Детерминант и метод Гаусса

Если не применять выбор ведущего элемента, то детерминант – это просто произведение диагональных элементов верхнетреугольной матрицы.

Перестановка строк может приводить к изменению знака.

Если меняются местами строки i и j , то знак будет меняться на противоположенный.

Выбор ведущего элемента

```
static bool SelectLeading(double [,] M, int n)
{
    //Найдем номер строки, с наибольшим
    //элементом в столбце n
    int iMax = n;
    for(int i = n+1; i < M.GetLength(0); i++)
        if(Math.Abs(M[iMax, n])
            < Math.Abs(M[i, n]))
            iMax = i;

    // Переставить строки iMax и n
    if(iMax != n)
    {
        for(int i = n; i < M.GetLength(1); i++)
        {
            double t = M[n, i];
            M[n, i] = M[iMax, i];
            M[iMax, i] = t;
        }
        return true;
    }
    return false;
}
```

Вычисляем детерминант

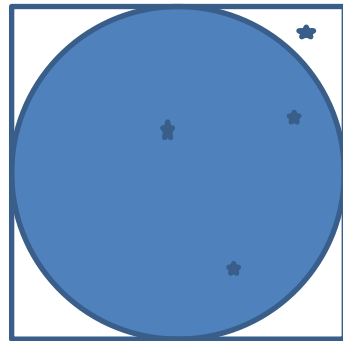
```
static double Determinant(double [,] M)
{
    double d = 1;
    for(int i = 1; i < M.GetLength(0); i++)
    {
        if(SelectLeading(M, i-1))
            d *=-1;
        if(Math.Abs(M[i-1, i-1]) > 0.0001)
            SubtractRow(M, i-1);
        else
            return 0;
    }
    for(int i = 0; i < M.GetLength(0); i++)
        d*=M[i, i];
    return d;
}
```

Контрольные вопросы

1. Как в решении задачи проявился характер вычислений с числами с плавающей точкой?
2. Какие преобразования числовых типов компилятор выполняет сам?
3. Как преобразовать числовые типы, если компилятор не позволяет неявное преобразование?

Вычисление числа Пи методом “Монте-Карло”

Метод основан на применении для вычислений случайных чисел.



Если моделировать попадание точек в квадрат равномерно по его площади, то доля точек попавших в круг, будет пропорциональна отношению площади круга к площади квадрата.

Вычисление числа Пи

$$S_{\text{окр}} = \text{Пи} * R^2$$

Удобно взять $R = 1$ и ограничиться 1-ой четвертью математической плоскости.

$$S_{\text{окр}} = \text{Пи}$$

Площадь квадрата при этом равна 4.

Вычисление Пи

Реализация

```
int N=1000000;  
int n=0;  
Double x, y;  
Random r = new Random();  
for(int i = 0; i < N; i++)  
{  
    x = r.NextDouble(); y = r.NextDouble();  
    if(x*x + y*y < 1)  
        n++;  
}  
Double pi = (4.0 * n) / N;  
  
Console.WriteLine("Pi = {0:0.###}", pi);
```

Вопросы для повторения

1. В чем особенность статических методов (методов класса) по сравнению с методами объектов?
2. Какие преимущества ООП дает на примере класса гистограммы?
3. На какой идее был основан алгоритм определения принадлежности точки полигону?
4. Как можно вычислить площадь полигона?