Лекция № не помню какая. Случайные числа и их моделирование

В общем, про системы, определяемые через системы дифференциальных уравнений в целом поговорили, пусть и не так длительно и эффективно, как хотелось, но все же.

Дальнейшие темы, посвященные той же тематике, не менее интересны, на самом деле, к тому же можно было бы подвести плавненько к важным для нас темам вроде теории управления и методам управления какими-то параметрами, в том числе рассмотреть ШИМ и ПИД, коие, похоже так ни в каких курсах наших не мелькают. А зря.

Но оставлю пока это — еще есть некоторое количество лекций, может еще расскажу. Сегодня же поговорим про не столь детерминированные модели, например — клеточные автоматы. В отличие от систем, задаваемых с помощью дифур, в которых зависимость от времени является непрерывной (можно взять совершенно любое время, хоть разницу между точками времени в $10^{-30}c$), в этих системах зависимость от времени является на базовом уровне дискретной. Например, отсчет идет в ходах, поколениях, кадрах и т.п. Кроме того, во многих моделях кроме времени также и координатные зависимости являются дискретными, то есть положения каких-то объектов могут располагаться строго в узлах какой-то регулярной сетки. Например, координаты могут принимать строго целые значения.

Такие ограничения существенно влияют на точность модели по отношению к какой-то реальной системе, но сильно упрощают моделирование.

Для примера надо понять что и как вообще может моделироваться с помощью таких «простых» моделей. Вообще, именно модели можно обсудить позже, а на сегодня хотелось бы поделиться основными методами, для этого моделирования используемыми.

Первое. Для моделирования постоянно используются псевдослучайные числа. Про них разговор может быть совершенно отдельный, ведь и методов их моделирования множество, и критериев, по которым определяется их «случайность», довольно много. Но есть несколько главных для нас сейчас особенностей.

1. Их значения достаточно мало зависят друг от друга и предсказать следующее выпавшее число без знания затравки генератора довольно сложно или нневозможно;

2. Однако притом одинаковые затравки генератора рождают одинаковые последовательности. То есть если не управлять начальным значением, затравкой, зерном генератора ПСЧ (псевдослучайных чисел),

то каждый запуск какой-то программмы будет получать ровно один и тот же вариант выполнения каждый раз;

3. Все ПСЧ дискретны, но в некоторых режимах ими можно пользоваться как непрерывными СЧ.

Как пример случайного числа можно взять бросок монеты или игральной кости. У них есть строго определенное количество вариантов результата, если не рассматривать вариант падения на ребро как крайне маловероятный, то у обычной монеты есть две стороны и, соответственно два возможных результата броска. У типичной игральной кости есть шесть граней и шесть вариантов результата. Правда, существуют разные игральные кости, такие как d4, d8, d10, d12, d20, где число обозначает количество граней, но они в целом распространены меньше шестигранной. Кстати, считается, что вероятность выпадения каждой грани игральной кости или стороны монеты одинаковы для одного предмета и равны $p_{\text{вып}} = \frac{1}{N}$, где N — количество вариантов результата.

Так вот. Пока как раз поговорим про дискретные ПСЧ. Они часто требуются для моделирования в чистом виде, но для разного количества вариантов. Например, хотим смоделировать случайное движение какого-то тела в одномерной системе координат, то есть которое может двигаться по одной оси, например X. Ход представляет собой передвижение тела в сторону увеличения или уменьшения нынешней координаты на единицу. Пример:

n	x
1	4
2	5
3	6
4	5
5	4
6	5
7	6
8	7
9	8
10	9

Здесь n — номер хода, x — координата. X меняется каждый ход. Для этого требуется как раз модель монеты — два варианта исхода. Но результат броска еще нужно превратить в инкремент или декремент и тут есть варианты. Можно написать условие вида: если выпала решка (0), то идем налево ($x = x - 1$), иначе — направо ($x = x + 1$), а можно и просто задать жестко величину инкремента. Предположим, что все так же решка-лево и орел-право, или $0- > -1$, $1- > +1$, тогда можно задать величину $dx = 2R - 1$, где R ноль или один, число, данное нам генератором ПСЧ. Тогда шаг выполняется без условий: $x = x + 2R - 1$.

Таким образом, нам только что удалось смоделировать одномерное движение броуновской частицы.

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main() {
  srand(time(NULL));
  int x, n, R;
  x = 4;
  printf("#1\n");
  for(n = 1; n <= 10; n++) {
    R = rand()%2;
    if(R == 0) {
      x = x - 1;
    } else {
      x = x + 1;
    }
    printf("%i %i\n", n, x);
  }
  printf("#2\n");
  x = 4;
  for(n = 1; n <= 10; n++) {
    R = rand()%2;
    x = x + 2 * R - 1;
    printf("%i %i\n", n, x);
  }
}
```

Здесь приведен код программы, реализующей оба способа. Замечу, что используется та самая затравка/зерно генератора ПСЧ с помощью команды srand(). rand() — вызов генератора. В зависимости от применяемого компилятора или среды разработки эта подпрограмма может вести себя по-разному. Но в любом случае это генератор равновероятных целых чисел. В MS Visual Studio диапазон генерируемых чисел 0-32767, для gcc это весь диапазон целого числа одинарной точности. Для приведения числа к заданному диапазону, в данном случае [0,1] использовано целочисленное деление, а именно остаток от целочисленного деления на 2: rand()%2 дает результаты 0 и 1. Все остальное уже было описано.

У случайных чисел есть еще интересные свойства, которые можно было бы изучить в лабораторной работе №4, но боюсь, вы все «сдохнете», если

не дать вам поблажку с этим «каран...» эмм самоизоляцией. Так что для «автоматчиков».

У меня есть еще небольшая программка для изучения и объяснения. Но комментарии там есть, поэтому объясню только неописанное внутри.

```
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <ctime>

#define QNT 20
#define XQNT 100
#define YQNT 100

using namespace std;
int main() {
  int x = 0, y = 0, dx, dy, c, i;
  srand(time(NULL));
  cout << "Броуновское движение\nМоделируем в двумерной сетке\n";
  cout << "Выражение получает число от 0 до 3 и выбирает движения\n";
  cout << "0 - влево\n1 - вверх\n2 - вправо\n3 - вниз\n";
  //  c dx dy c%2 c/2 (2*(1-c/2)-1)*c%2 (2*(c/2)-1)*(1-c%2)
  //  0 -1  0  0   0       0                      -1
  //  1  0  1  1   0       1                       0
  //  2  1  0  0   1       0                       1
  //  3  0 -1  1   1      -1                       0
  //  cout << "   c  dx  dy\n";
  for(c = 0; c < 4; c++) {
    printf("%4i%4i%4i\n", c, (2*(c/2)-1)*(1-c%2), (2*(1-c/2)-1)*c%2);
  }
  cout << "Движение в ограниченной замкнутой кольцом области\n";
  cout << "x=(x+dx+x_qnt)%x_qnt\n   c   x   y\n";
  dx = 1;
  dy = -1;
  for(int c = 0; c < 10; c++) {
    x=(x + dx + XQNT)%XQNT;
    y=(y + dy + YQNT)%YQNT;
    printf("%4i%4i%4i\n", c, x, y);
  }
  cout << "Броуновское движение\n   i   x   y\n";
  printf("%4i%4i%4i\n",0,x=49,y=49);
  for(i = 0; i < 10; i++) {
```

```
    c = rand()%4;
    dx = (2*(c/2)-1)*(1-c%2);
    dy = (2*(1-c/2)-1)*c%2;
    x=(x + dx + XQNT)%XQNT;
    y=(y + dy + YQNT)%YQNT;
    printf("%4i%4i%4i%s\n",i,x,y,
           c<2?(c<1?" Влево":" Вверх"):(c<3?" Вправо":" Вниз"));
  }
}
```

Здесь приведен процесс получения модели движения броуновской частицы на двумерной поверхности. Выражение {условие}?{результат$_{\text{да}}$} : {результат$_{\text{нет}}$} это что-то вроде оператора если, только пишется по-другому.

Повехность замкнута по виду тора — переход за какую-либо из границ области приводит к приходу в область в ту же точку, но с противоположной стороны. Вариантов исполнения, опять же несколько, но способ обработки координат через условия лично мне не нравится — почему-то студенты очень неуверенно пользуются условиями и совершенно не понимают, что пишут (некоторые), поэтому привожу свой способ.

Предположим, хотим, чтобы при переходе через верхнюю границу частица оказалась на нижней границе по какой-то координате. Предположим, что размер поля по этой координате 10, то есть координата принимает значения 0..9. То есть при передвижении из 9 вверх следующая координата должна быть 0, а не 10. Для этого нам тоже пригодится остаток от деления. Выражение $x = (x+1)\%10$ при $x = 9$ как раз и даст нам число 0. Для верхней границы это работает. Кстати, под то выражение спокойно можно добавить любое количество десяток и ничего принципиально не изменится. То есть выражение $x = (x + 1 + 10n)\%10$ для целых n не меняет ничего в выполнении. Именно это нам скоро и пригодится. Предположим, что наше выражение должно позволять получать также и 9 при переходе через нижнюю границу. Посмотрим, что получится для $x = 0$: $x = (x - 1)\%10 = -1$. Как же так? На самом деле, отрицательные числа точно так же делятся на 10, но для них остатки от деления не будут положительны, там диапазон остатков — -9..0. Это нам не подходит. Но что, если воспользоваться сдвигом, описанным ранее. Например $x = 0$: $x = (x - 1 + 10)\%10 = 9$. Победа. Это выражение позволит автоматически и без условий получать нужную координату в заданных пределах.

$$x = (x + dx + Q_x)\%Q_x \tag{1}$$

Здесь dx — величина сдвига, в данном случае -1 или 1. Q_x — размер поля по координате x.

На эту лекцию, думаю, достаточно.

Контрольные вопросы к лекции.

1. Как выполняется операция взятия целочисленного остатка в C? Какие результаты дает остаток от деления числа на N?
2. Что значит поле замкнуто по виду тора?
3. Какова вероятность выпадения 6 на 8-гранной игральной кости?
4. Если собрать приведенные примеры программ, каждый запуск будет приводить к строго одним и тем же результатам или будет меняться?