Classes, Methods, and Object Oriented Design

Plus a few slides remaining on visualization of randomness…

Wednesday, March 25, 2009

1

Clicker Question

Your computer does decimal arithmetic, but can represent only 3-digits precision. You want to compute:

0.007 + 0.153 + 0.201 + 0.008 + 3.12 + 0.876 + 0.015

(which is 4.38)

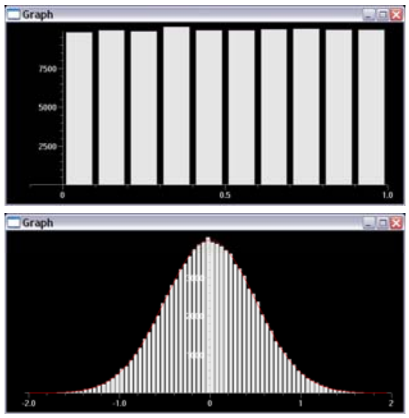
What order achieves the highest accuracy?

A. Any order will generate the same result
B. 0.007 + 0.008 + 0.015 + 0.153 + 0.201 + 0.876 + 3.12, evaluated from right to left
C. 0.007 + 0.008 + 0.015 + 0.153 + 0.201 + 0.876 + 3.12, evaluated from left to right
D. [((3.12 + 0.008) + (0.201 + 0.015)) + (0.153 + 0.876)] + 0.007, evaluated as indicated

2

Visual Randomness Test

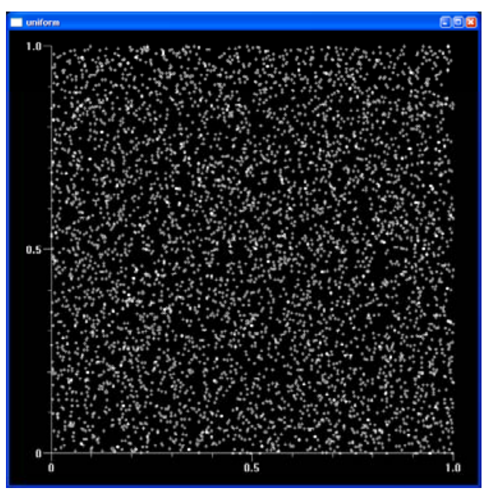
- Histogram is the basic tool



100,000 trials each

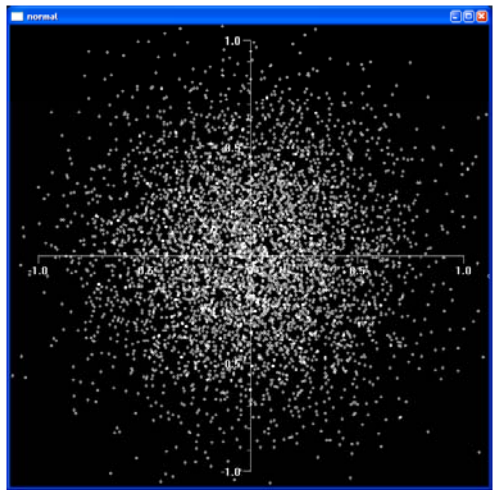
3

•2D Mappings to visualize randomness



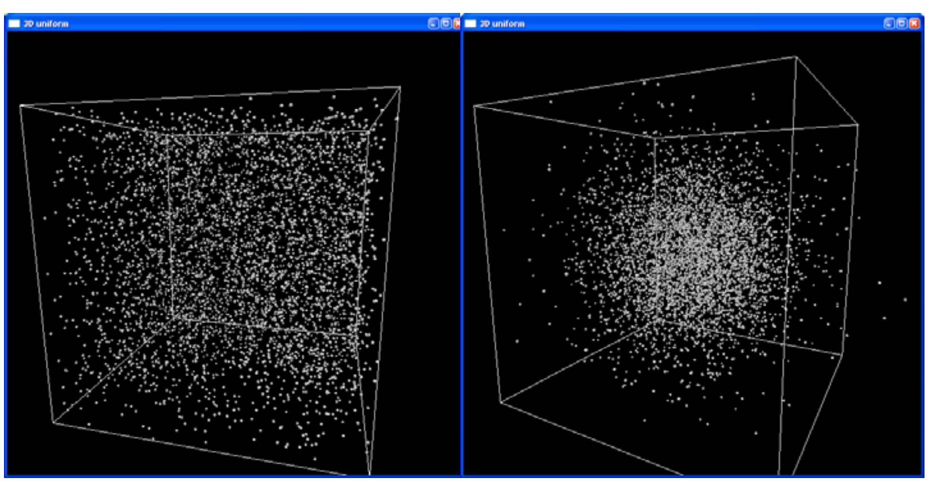
5,000 points, uniform

2D Mappings



5,000 points, normal

3D Mappings



5,000 points

Poor random number generators

- Linear congruence: $X_{k+1} = (aX_k+c)$ % m
  - period is at most m
  - careful with choice of a, c, m: a and m should be relative primes
- But note that even then things tend to lie in hyperplanes, exhibiting an unwanted correlation … see Map3Dbad.py

7

Using Python’s random class

```
from random import uniform
g = grid(size, 0)
   for row in range(size):
      for col in range(size):
         if uniform(0,1) < p:        # true with probability p
            g[row][col] = 0
         else:
            g[row][col] = 1
   return g
```

8

Python's Random Class

`random()` – uniformly distributed numbers in [0,1]:

```
from random import *
 X = Random()
 X.seed(1)
 v = X.random()
```

```
X = Random()
v = X.random()
```

`randrange(b,e,s)` – **`int`** in **`range(b,e,s)`**

`randint(l,u)` – **`randrange(l,u+1)`**

`normalvariate(μ,σ)` – Gauss distribution

9

Operations, functions, methods

Lists

- +, *, indexing — *operations*
- len(L) — *functions*
- myL.append('x')
- myL.sort
- myL.count(1)
- *plus others*

Methods (Zelle Appendix, pp 478)

10

VPython

rod = cylinder(pos=(0,2,1), axis=(5,0,0), radius=1)

 # creates a cylinder named rod

rod.pos = (15,11,9)

 # changes the position of rod

rod.color = (0,0,1)

 # changes the color of rod

11

Clicker Question: what is printed?

```
myL1 = ['a', 'b', 'c', 'd']
myL2 = []

for i in range(len(myL1)):
    myL2.append(myL1[i])

myL1[0] = 0
myL2[0] = "A"

print myL1
print myL2
```

A. `[0, 'b', 'c', 'd']`
`['A', 'b', 'c', 'd']`

B. `[‘A’, 'b', 'c', 'd']`
`['A', 'b', 'c', 'd']`

C. `[0, 'b', 'c', 'd']`
`[0, 'b', 'c', 'd']`

D. `[0, ‘c’, ‘b’, ‘a’]`
`[‘A’, ‘c’, ‘b’, ‘a’]`

12

Classes and Objects

- So far, the programs we have seen and written were viewed as logical procedures that
  - take input data
  - process it, generate random data, run simulations
  - generate output data
- The programming challenge was defining the logic, not how to define the data.
- **Object-oriented programming (OOP)** takes the view that we also care about the objects we want to manipulate.

13

Object Oriented Programming

- In almost all modern languages, a programmer can define new types (classes)
- Not used in mainstream software development until the early 1990s
- Defining a **class**:
  - create **objects** that are **instances** of this class
  - use **methods** to operate on objects
- Vpython has a class sphere
  ball = sphere(pos(0,4,4), radius = 2)
- Object oriented programming can lead to easier to write code and more readable code

14

Objects

An object consists of:

1. A collection of related information.
2. A set of operations to manipulate that information.

- The information is stored inside the object in *instance variables*.
- The operations, called *methods*, are functions that "live" inside the object.
- Collectively, the instance variables and methods are called the *attributes* of an object.

15

Intro to Objects

- `sphere` is a class in VPython
- A sphere object will have instance variables
  - pos, which remembers the center point of the circle
  - radius, which stores the length of the circle's radius.
- Creating a sphere object:
  ball = sphere(pos(0,4,4), radius = 2)
  ball is drawn in VPython window
- VPython monitors the values of pos and radius to decide which pixels in a window should be colored.

16

```
>>> from visual import *
>>> ball = sphere(pos=(0,0,0), radius=2, color = color.red)
>>> ball
<visual.primitives.sphere object at 0x02C058A0>
>>> ball.radius
2.0
>>> ball.color
(1.0, 0.0, 0.0)
>>> sphere
<class 'visual.primitives.sphere'>
>>> ball.rotate(angle=pi/4)
```

17

Intro to Objects

- New objects are created from a class by invoking a *constructor*. You can think of the class itself as a sort of factory for stamping out new instances.
- Consider making a new circle object:
  `ball = sphere(pos(0,4,4), radius = 2)`
  - `sphere`, the name of the class, is used to invoke the constructor.
  - Creates a sphere instance and stored the reference to it in the variable ball
- Parameters of the constructor generally initialize some of the attributes
- Once an instance has been created, it can be manipulated by calling its methods; e.g., ball.rotate(angle=pi/4.)

18

Example: Multi-Sided Dice

- Zelle, Chapter 10.3
- A normal die (singular of dice) is a cube with six faces, each with a number from one to six.
- Some games use special dice with a different number of sides.
- **Goal:** design a generic class MSDie to model multi-sided dice.

19

Example: Multi-Sided Dice

- Each MSDie object will know two things:
  - How many sides it has
  - It's current value
- When a new MSDie is created, we specify *n*, the number of sides it will have.

20

Example: Multi-Sided Dice

- Three methods will operate on the die:
  - roll – set the die to a random value between 1 and *n*, inclusive
  - setValue – set the die to a specific value (i.e. cheat)
  - getValue – see what the current value is.

21

Example: Multi-Sided Dice

```
>>> die1 = MSDie(6)
>>> die1.getValue()
1
>>> die1.roll()
>>> die1.getValue()
5
>>> die2 = MSDie(13)
>>> die2.getValue()
1
>>> die2.roll()
>>> die2.getValue()
9
>>> die2.setValue(8)
>>> die2.getValue()
8
```

22

Example: Multi-Sided Dice

- Using object-oriented vocabulary, we create a die by invoking the MSDie *constructor* and providing the number of sides as a *parameter*.
- Die objects will keep track of this number internally as an *instance variable*.
- Another *instance variable* is used to keep the current value of the die.
- We initially set the value of the die to be 1 because that value is valid for any die.
- That value can be changed by the roll and setRoll methods, and returned by the getValue method.

23

Example: Multi-Sided Dice

```
# msdie.py
#     Class definition for an n-sided die.

from random import randrange

class MSDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value
```

24