Some Simple Numeric Programs

Chapter 3

Exhaustive Enumeration

- A form of a **Guess and Check** algorithm
  - Tests all possible values (exhaustive)
    - Until an answer is found
    - We run out of space
  - May seem tedious
    - We'll talk about complexity in a few weeks
  - Typified by a **Decrementing Function**

Decrementing Functions

- Maps a set of program variables into an integer
- When the loop is entered the initial value of function is non-negative
- When the function is $\leqq 0$ the loop terminates
- The value of the function is decremented each iteration through the loop

Exhaustive Enumeration Example

```
#Find the cube root of a perfect cube
x = int(input('Enter an integer: '))
ans = 0
while ans**3 < abs(x):
    ans = ans + 1
if ans**3 != abs(x):
    print (x, 'is not a perfect cube')
else:
    if x < 0:
        ans = -ans
    print ('Cube root of', x,'is', ans)
```

Exhaustive Enumeration is “inelegant”

- But modern computers are fast

- Implementation considerations
  - What range of values are expected?
  - How often will this be used?
  - How much time will it take to develop a more elegant solution?
  - Has someone already done this for me?
  - Are there quick short cuts that can be used?

Prime numbers

```
# Test if an x > 2 is prime. If not, print the smallest divisor
x = int(input('Enter an integer greater than 2: '))
smallest_divisor = None
for guess in range(2, x):
    if x % guess == 0:
        smallest_divisor = guess
        break
if smallest_divisor != None:
    print('Smallest divisor of', x, 'is', smallest_divisor)
else:
    print(x, 'is a prime number')
```

Approximation

- Determine the Square Root
  - 25
  - 9
  - .25
  - 2

- Some problems need a solution that will be “close enough” or an approximation
- Typically identified by a constant called **epsilon** ε

Calculate a square root

Exhaustive Enumeration

```
x = 25
epsilon = 0.01
step = epsilon**2
numGuesses = 0
ans = 0.0
while abs(ans**2 - x) >= epsilon and ans <= x:
    ans += step
    numGuesses += 1
print ('numGuesses =', numGuesses)
if abs(ans**2 - x) >= epsilon:
    print ('Failed on square root of’, x)
else:
    print (ans, 'is close to square root of’, x)
```

Switching Gears

- Searching for text in a hard cover reference
  - Dictionary, encyclopedia, phone book, book index, etc
- List is sorted
- Could search from start to until the word/term is found
  - Average time for search n/2
  - n if not in list
- OR could search by look at the middle and go higher or lower if we miss
- **Bisection Search**
  - More in later lessons

Let’s try Square Root again

```
x = 25
epsilon = 0.01
numGuesses = 0
low = 0.0
high = max(1.0, x)
ans = (high + low)/2.0
while abs(ans**2 - x) >= epsilon:
    print ('low =', low, 'high =', high, 'ans =', ans)
    numGuesses += 1
    if ans**2 < x:
        low = ans
    else:
        high = ans
    ans = (high + low)/2.0
print ('numGuesses =', numGuesses)
print (ans, 'is close to square root of’, x)
```

A Word about Floats

- There is a difference between real numbers and **`float`**s
- Computers store data in binary form (bits)
  - At least non-quantum computers use binary ;)
- A byte is classically 8 bits
  - ASCII and UTF-8 characters are one byte
  - UNICODE is 2 or 4 bytes
  - Often expressed in Octal (base 8) or Hexadecimal (base 16)
  - 4 bits are a “nibble”

Dec	Bin	Oct	Hex		Dec	Bin	Oct	Hex
00	00000000	000	00		13	00001101	15	OD
01	00000001	001	01		14	00001110	16	OE
02	00000010	002	02		15	00001111	17	0F
03	00000011	003	03		16	00010000	20	10
04	00000100	004	04		17	00010001	21	11
05	00000101	005	06		18	00010010	22	12
06	00000110	006	06		19	00010011	23	13
07	00000111	007	07		20	00010100	24	14
08	00001000	010	08		21	00010101	25	15
09	00001001	011	09		22	00010110	26	16
10	00001010	012	0A		23	00010111	27	17
11	00001011	013	0B		24	00011000	30	18
12	00001100	014	0C		25	00011001	31	19

So what about floats?

- We can express any decimal number using an exponent
  - 10,000 = 1.0*104 $1*10^4$
  - 124 = $1.24*10^2$ = $124*10^0$
  - 1.24 = $1.24*10^0$ = $124*10^{-2}$
- The first integer represents the **significant digits** or **mantissa**
- The second integer represents the **exponent**
- The number of digits available to the mantissa gives the **precision**

And so, binary floats?

- Same concept
  - Mantissa and exponent must be binary numbers –
  - $5 = 5*2^0$ = 101 0
  - $½ = 1*2^{-1}$ = 1 -1
  - $5/8 = 5*2^{-3}$ = 101 -11

So what about 0.1 or 1/10

- ½ = 0.5
- ¼ = 0.25
- 1/8 = 0.125
- 1/16 = 0.0625
- 3/32 = 0.09375
- 7/64 = 0.109375
- 13/128 = 0.1015625
- 25/256 = 0.09762625
- 51/512 = 0.099609375
- 103/1024 = 0.100585938

Two Consequences

- Floating point leads to approximate results
  - Need to test within some margin of error (**epsilon**)
- Rounding errors can accumulate

Newton-Raphson Method

- Isaac Newton & Joseph Raphson
- Find the "real root" of a function
  - A polynomial with one variable is either 0 or a finite number of non-zero terms
  - Each term is either a **coefficient** multiplied by the **variable** raised to a non-negative integer **exponent**
  - The exponent gives the **degree** of the term
  - The highest degree of all terms gives the degree of the polynomial
  - The **root** of a polynomial is the value p(r) = 0
    - r is the value of x that resolves the polynomial

Example

- $p = 3 + 2x + 4x^2$
  - What is the degree of the polynomial?
    - 2
  - What is p(5)?
    - $3 + 2(5) + 4(5)^2$
    - 113

How does that help me find the square root

- Square root of 24
  - $x^2 - 24 = 0$
  - p(0) is the square root of 24

- Newton (and Raphson) provide that a guess can be improved by subtracting p(guess)/p’(guess) from the original guess
  - First derivative of x^2 is 2x
  - We can continue improving our guess until within epsilon
  - **Successive approximation**

In Python

```
#Newton-Raphson for square root
#Find x such that x**2 - 24 is within epsilon of 0
epsilon = 0.01
k = 24.0
guess = k/2.0
number_guesses = 1
while abs(guess*guess - k) >= epsilon:
    guess = guess - (((guess**2) - k)/(2*guess))
    number_guesses += 1
print('Square root of', k, 'is about', guess)
print(‘That took’, number_guesses, ’guesses’)
```