Computing Iterative Functions

cs4: Computer Science Bootcamp

Çetin Kaya Koç
http://koclab.cs.ucsb.edu/teaching/cs4
cetinkoc@ucsb.edu

Computing Iterative Sums

- Consider the sum:

$$\sum_{i=1}^{100} i^2 \;=\; 1^2 + 2^2 + 3^2 + \cdots + 100^2$$

- It is known that this sum is equal to 338,350
- We can also parameterize the sum with n

$$\sum_{i=1}^{n} i^2 \;=\; 1^2 + 2^2 + 3^2 + \cdots + n^2$$

for any $n \geq 1$

Computing Iterative Sums

- If write the sum as a function of the parameter n:

$$f(n) = \sum_{i=1}^{n} i^2$$

  we can compute $f(n)$ for a given n

- For example, we know $f(100) = 338,350$
- What about $f(200)$?
- How can we compute $f(n)$ for a given n?

Compact Formulas for Sums

- There is a compact formula for this sum:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

- This identity can be proven using mathematical induction
- We verify:

$$f(100) = \sum_{i=1}^{100} i^2 = \frac{100 \cdot 101 \cdot 201}{6} = 338,350$$

- Now we can easily compute $f(200)$:

$$f(200) = \frac{200 \cdot 201 \cdot 401}{6} = 2,686,700$$

Computing General Iterative Sums

- What if we don't have a compact formula for a sum expression $g(n)$?
- For example, consider:

$$g(n) = \sum_{i=1}^{n} i(i+1)^2 = 1 \cdot 2^2 + 2 \cdot 3^2 + 3 \cdot 4^2 + \cdots + n \cdot (n+1)^2$$

- How do we compute $g(100)$ or $g(200)$?
- Answer: We write an iterative program for $g(n)$

Computing $g(n)$ Iteratively

- Computing $g(n) = \sum_{i=1}^{n} i(i+1)^2$ for a given n

```
sum = 0
n = 100
for i in range(1,n+1):
    term = i*(i+1)*(i+1)
    sum = sum + term
print(sum)
```

- The above program computes $g(100)$, and it gives 26,184,250
- For a general n, we write a function

An Iterative Python Function for $g(n)$

```
def gsum(n):
    sum = 0
    for i in range(1,n+1):
        term = i*(i+1)*(i+1)
        sum = sum + term
    return(sum)
```

- We can now compute $g(n)$ for any n we provide as input to the function gsum
- $g(100) = 26,184,250$
- $g(200) = 409,403,500$
- $g(1000000) = 250,001,166,668,416,667,500,000$

Iterative Function for General Sums

- A general sum is in the form

$$h(n) \; = \; \sum_{i=n_0}^{n} a_i$$

  such that n_0 is the starting point, n is the ending point, and a_i is the general term
- For example, for $f(n) = \sum_{i=1}^{n} i^2$, we have $n_0 = 1$ and $a_i = i^2$
- For $g(n) = \sum_{i=1}^{n} i(i+1)^2$, we have $n_0 = 1$ and $a_i = i(i+1)^2$

Iterative Function for General Sums

- Once the starting and ending points and the general term is available, we can easily write a Python function
- The first rule is that, we start with: `sum = 0`
- The for loop boundary conditions give: `range(n0,n+1)`
- The general term is computed using: `term = ai`
- The iteration rule is: `sum = sum + term`
- Thus, the function becomes:

```
def fsum(n):
    sum = 0
    for i in range(n0,n+1):
        term = ai
        sum = sum + term
    return(sum)
```

Euler Formula for π

- Another interesting formula involving π was given by Euler:

$$\frac{\pi^2}{6} = \sum_{i=1} \frac{1}{i^2} = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots$$

- Applying our rules from the previous slide, we see that:
  The sum starts with `sum = 0`
  The for loop boundary conditions are: `range(1,n+1)`
  The general term is computed using: `term = 1/(i**2)`
  The iteration rule is: `sum = sum + term`
- Finally we compute π from `sum`, we use

$$\pi = \sqrt{6 \cdot \mathtt{sum}}$$

- To compute π with higher accuracy, we increase the value of n

Python Code for Euler Formula

```
def eulerPi(n):
    sum = 0
    for i in range(1,n+1):
        term = 1/(i**2)
        sum = sum + term
    return(math.sqrt(6*sum))
```

- Computing π for n from 1,000 to 8,000:

```
n    Pi
1000 3.1406380562059946
2000 3.1411152718364823
4000 3.141353941945064
8000 3.141473292575064 6
```

Computing Terms Iteratively

- One issue that often comes up is the details of the computation of the ith term, `term`
- There could be some savings in the number of arithmetic operations in the computation of `term`
- Generally this is achieved by using the $(i-1)$st term in the computation of ith term, for $i = 1, 2, 3, \ldots$
- In other words, we compute `term` not directly but *iteratively*
- The computation of `sum` is also iterative, since the sum in the previous iteration is utilized

Computing Terms Iteratively

- To illustrate iterative computation of `term`, we use a formula involving the Euler's constant e

$$e \;=\; 1 + \sum_{i=1} \frac{1}{i!} \;=\; 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots$$

- The numerator of these fractions $\frac{p_i}{q_1}$ is always $p_i = 1$, while the denominator is $q_i = i!$ at the ith iteration
- The ith denominator can be derived from $(i-1)$st denominator:

$$q_i = q_{i-1} \cdot i = (i-1)! \cdot i \;=\; i!$$

- Therefore, if `term` from the $(i-1)$st iteration is available, we compute `term` for the ith iteration as `term = term/i` since

$$\text{term} \;=\; \frac{p_i}{q_i} \;=\; \frac{1}{q_i} \;=\; \frac{1}{q_{i-1}} \cdot \frac{1}{i} \;=\; \frac{p_{i-1}}{q_{i-1}} \cdot \frac{1}{i} \;=\; \text{term}/\text{i}$$

Python Code for Computing *e*

- To compute `term` iteratively, we to start with `term = 1`
- Meanwhile the initial value of `sum = 0`
- In the final step, we add 1 to `sum` to obtain e

```
def e(n):
    sum = 0
    term = 1
    for i in range(1,n+1):
        term = term/i
        sum = sum + term
    return(1+sum)
```

Iterative Functions for Products

- Wallis formula for computing $\pi/2$ was a product formula:

$$\frac{\pi}{2} = \prod_{i=1}^{n} \left(\frac{2i}{2i-1} \cdot \frac{2i}{2i+1} \right) = \frac{2 \cdot 2}{1 \cdot 3} \cdot \frac{4 \cdot 4}{3 \cdot 5} \cdot \frac{6 \cdot 6}{5 \cdot 7} \cdot \frac{8 \cdot 8}{7 \cdot 9} \cdots$$

- An iterative method for this formula would be computing a product, instead of a sum
- Therefore, it would start with: `prod = 1`
- The for loop boundary conditions give: `range(1,n+1)`
- The general term is: `term = (2*i)**2/((2*i-1)*(2*i+1))`
- The iteration rule is: `prod = prod * term`

Iterative Computation of Wallis Formula

```
def wallisPi(n):
    prod = 1
    for i in range(1,n+1):
        term = (2*i)**2/((2*i-1)*(2*i+1))
        prod = prod * term
    return(2*prod)
```

- Computing π for n from 1000 to 8000:

```
n    Pi
1000 3.1408085296644828
2000 3.1412002733216604
4000 3.141396383784121
8000 3.1414944987571713
```

Iterative Functions for Sums and Products

- An iterative function may have both sum and product terms
- In this case, we keep a `sum` and a `prod` variables and run them through the iteration
- For example, Vieta's formula allows us to compute $\frac{2}{\pi}$:

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \cdot \frac{\sqrt{2+\sqrt{2}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \cdot \frac{\sqrt{2+\sqrt{2+\sqrt{2+\sqrt{2}}}}}{2} \cdots$$

- The general term a_i is a ratio: $\frac{p_i}{q_i}$
- The numerator p_i in ith step is used to computed the numerator in the $(i+1)$st step: $a_{i+1} = \sqrt{2+a_i}$ by starting with $a_0 = 0$
- The denominator is always 2

Computing π using Vieta's Formula

```
def vieta(n):
    prod = 1
    num = 0
    for i in range(1,n+1):
        num = math.sqrt(2+num)
        prod = prod*num/2
    return(2/prod)
```

- Vieta's formula produces more accurate results for smaller n
- Computing π for n from 5 to 40:

```
n   Pi
 5  3.140331156954753
10  3.141591421511199
20  3.141592653588618
40  3.141592653589794
```