Lecture 6: Iteration and Recursion

Math 98

Reminders and Agenda

- Agenda
  - Recursion vs. Iteration
  - Exercises

Iteration: Motivations

Many tasks in life are boring or tedious because they require doing the same basic actions over and over again – `iterating` – in slightly different contexts.

- So let's get the computer to do this!
- `for` loops and `while` loops.

Iteration: `for` loops and `while` loops

A statement to repeat a section of code a specified number of times.

```
for countVariable = 1 : numberOfIterations
%   do something here
%   this part will run
%   (numberOfIterations) times
end
```

A statement to repeat a section of code *until* some condition is satisfied.

```
while [EXPRESSION is true]
%    repeat this part until
%    (EXPRESSION) is false
%    be sure to modify (EXPRESSION) in this loop
end
```

Fixed Point Iteration: Example

Let's say we're interested in this fixed iteration

$$\varphi(x) = \sqrt{1+x} \qquad x_0 = 3$$

After 10 iterations.

```
>> x = 3;
x = sqrt(1+x)
x =
    2
.........
x = sqrt(1+x)
x =
    1.618064196086926
x = sqrt(1+x)
x =
    1.618043323303466
```

Fixed Point Iteration: For Loop

I claim this converges to $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618033988749895$. This is the golden ratio, one of the most famous numbers in mathematics.

I probably should have done the above calculation with a for loop.

```
>> x = 3;
for k = 1:10
    x = sqrt(1+x);
end
x
x =
    1.618043323303466
```

Fixed Point Iteration: While Loop

Let's do this with a while loop until it "converges", until the computer can't tell the difference anymore.

```
>> x = 3;
while x~= sqrt(1+x)
    x = sqrt(1+x)
end
x =
    1.618033988749895
>> x == (1+sqrt(5))/2
ans =
   logical
    1
```

Infinite Loops

Careful with infinite loops!

```
>> N = 0;
while N > -1
    N = N + 1;
end
```

Put maximum iteration limits and breaks in your loops to guard for this.

Factorial as an Iteration

How do we compute the factorial of a number?

$$n! = \begin{cases} 1 & n == 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

A `for` loop will do nicely.

```
function nfac = myFactorial(n)
    nfac = 1;
    for i = 1:n
        nfac = nfac * i;
    end
end
```

Factorial as a Recursion

How do we compute the factorial of a number?

$$n! = \begin{cases} 1 & n == 0 \\ n \times (n-1)! & n > 0 \end{cases}$$

We can also take advantage of the recursive definition, and define our function recursively:

```
function nfac = myFactorial(n)
    if n == 0
        nfac = 1;
    else
        nfac = n*myFactorial(n-1);
    end
end
```

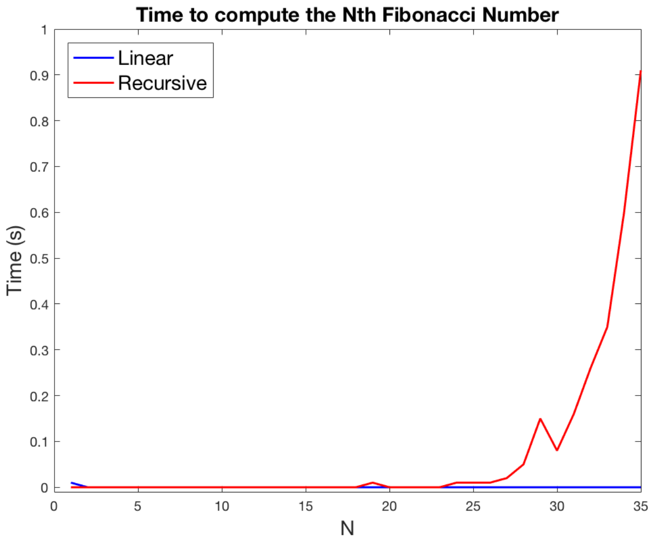
Exercise: Fibonacci Numbers

Define the Fibonacci numbers as

$$f(n) = \begin{cases} 0 & n == 0 \\ 1 & n == 1 \\ f(n-1) + f(n-2) & n >= 2 \end{cases}$$

Write a recursive function to compute $f(n)$, then write a non-recursive function (`for` loop) to do the same. The non-recursive function should compute all numbers $f(0), f(1), \ldots, f(n)$.

Fibonacci Numbers: Compute Times



Fibonacci Numbers: Compute Times

The problem: our recursive definition did lots of unnecessary computation by not using previously computed values.

```
>> fiboRec(4)
Computing f(4)
Computing f(2)
Computing f(0)
Computing f(1)
Computing f(3)
Computing f(1)
Computing f(2)
Computing f(0)
Computing f(1)
ans =
     3
```

Iteration Exercise: `nested_sqrt.m`

Write a function

```
function a = nested_sqrt(n)
```

that takes an integer n and returns the nth term in the following sequence:

$$a_1 = 1, a_2 = \sqrt{1+2}, a_3 = \sqrt{1+2\sqrt{1+3}}, a_4 = \sqrt{1+2\sqrt{1+3\sqrt{1+4}}}, \ldots$$

Guess the limiting value of the sequence $a = \lim_{n\to\infty} a_n$ and make a plot of $\ln(|a_n - a|)$ vs. n. Also plot the line $y = 3 - (\ln 2)n$.
What sequence β_n would you guess is appropriate for $a_n - a = O(\beta_n)$?

Recursion Exercise: `qsort.m`

How do we sort a list of numbers v?
There are many ways, but `quickSort` offers a simple recursive implementation.

1. Pick an element $x \in v$ to be the **pivot** element. (say, the first one).
2. Divide the rest of the list in two: those **smaller** than x and those **larger** than x.
3. `output = [quickSort(Smaller), x, quickSort(Larger)]`

A few questions we need to answer when working out the details:

- What are the base cases that we need to handle?
- What if some numbers are the same size as x?

Implement

```
function w = qsort(v)
```