COMPSCI 590N
Lecture 4: Computing Special Functions

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

Outline

Miscellaneous Python Stuff

- **Comments** allow you to write notes about your code.

Miscellaneous Python Stuff

- **Comments** allow you to write notes about your code.

Miscellaneous Python Stuff

- **Comments** allow you to write notes about your code.

```
# Single line comment

"""
Multi-line
Comment
"""
```

Miscellaneous Python Stuff

- **Comments** allow you to write notes about your code.

```
# Single line comment

"""
Multi-line
Comment
"""
```

- Mixing tabs and spaces when indenting code will cause an error.

Outline

Numerical Algorithms

- So far we discussed how numbers are represented in a computer and how these representations can result in errors

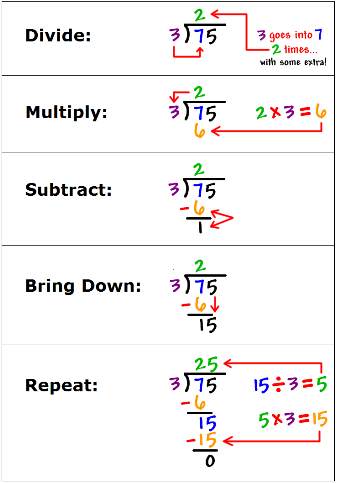
Numerical Algorithms

- So far we discussed how numbers are represented in a computer and how these representations can result in errors
- In numerical computing we must also manipulate numbers, which can introduce further errors.

Numerical Algorithms

- So far we discussed how numbers are represented in a computer and how these representations can result in errors
- In numerical computing we must also manipulate numbers, which can introduce further errors.
- **Numerical algorithms** approximate mathematical functions under the restriction of finite precision and finite time.

Long Division



Types of Numerical Algorithms

- Generally, numerical methods fall into one of two categories:

Types of Numerical Algorithms

- Generally, numerical methods fall into one of two categories:

Types of Numerical Algorithms

- Generally, numerical methods fall into one of two categories:

Direct

A direct method is guaranteed to terminate in a finite number of steps and would return the exact answer if we had infinite precision.

Types of Numerical Algorithms

- Generally, numerical methods fall into one of two categories:

Direct

A direct method is guaranteed to terminate in a finite number of steps and would return the exact answer if we had infinite precision.

Iterative

An iterative method is never guaranteed to terminate, but instead builds a sequence of increasingly accurate approximations and terminates when a desired accuracy is achieved.

Numerical Algorithms

- We have already seen how errors can be caused by rounding.

Numerical Algorithms

- We have already seen how errors can be caused by rounding.
- Rounding can affect numerical methods in different ways.

Numerical Algorithms

- We have already seen how errors can be caused by rounding.
- Rounding can affect numerical methods in different ways.
- Two algorithms that should mathematically give the same result can behave very differently when confronted with the errors caused by finite precision.

Numerical Algorithms

- We have already seen how errors can be caused by rounding.
- Rounding can affect numerical methods in different ways.
- Two algorithms that should mathematically give the same result can behave very differently when confronted with the errors caused by finite precision.

Numerical Algorithms

- We have already seen how errors can be caused by rounding.
- Rounding can affect numerical methods in different ways.
- Two algorithms that should mathematically give the same result can behave very differently when confronted with the errors caused by finite precision.

DEMO

Numerical Algorithms

The main notion of how well a numerical algorithm performs is called **stability**:

Numerical Algorithms

The main notion of how well a numerical algorithm performs is called **stability**:

Stability

A numerical algorithm is said to be **stable** if small errors, no matter what their source is, do not cause large errors in the final result.

Numerical Algorithms

The main notion of how well a numerical algorithm performs is called **stability**:

Stability

A numerical algorithm is said to be **stable** if small errors, no matter what their source is, do not cause large errors in the final result.

- Examples:

Numerical Algorithms

The main notion of how well a numerical algorithm performs is called **stability**:

Stability

A numerical algorithm is said to be **stable** if small errors, no matter what their source is, do not cause large errors in the final result.

- Examples:
  1. If an algorithm allows representability errors to accumulate through addition, then we would call this algorithm unstable.

Numerical Algorithms

The main notion of how well a numerical algorithm performs is called **stability**:

Stability

A numerical algorithm is said to be **stable** if small errors, no matter what their source is, do not cause large errors in the final result.

- Examples:
  1. If an algorithm allows representability errors to accumulate through addition, then we would call this algorithm unstable.
  2. Most iterative algorithms start with an initial guess and refine it. If the accuracy of a method is highly sensitive to the initial guess, then we would call this method unstable.

Special Functions

- Unfortunately, most of the standard mathematical functions we use cannot be computed directly and so must be approximated using numerical methods.

Special Functions

- Unfortunately, most of the standard mathematical functions we use cannot be computed directly and so must be approximated using numerical methods.
- Examples: square root, logarithm, exponentiation, and sine/cosine

Special Functions

- Unfortunately, most of the standard mathematical functions we use cannot be computed directly and so must be approximated using numerical methods.
- Examples: square root, logarithm, exponentiation, and sine/cosine
- While you will likely never need to implement these algorithms (most are implemented in hardware!), it is important to know how they are implemented as it can have a large impact on the speed of your programs.

Special Functions

- Unfortunately, most of the standard mathematical functions we use cannot be computed directly and so must be approximated using numerical methods.
- Examples: square root, logarithm, exponentiation, and sine/cosine
- While you will likely never need to implement these algorithms (most are implemented in hardware!), it is important to know how they are implemented as it can have a large impact on the speed of your programs.
- In the following slides we will talk about some general techniques for approximating these functions and then look at a few examples.

Bit-by-bit Methods

Bit-by-bit Methods

1	3	5	7
9	11	13	15
17	19	21	23
25	27	29	31

Bit-by-bit Methods

2	3	6	7
10	11	14	15
18	19	22	23
26	27	30	31

Bit-by-bit Methods

4	5	6	7
12	13	14	15
20	21	22	23
28	29	30	31

Bit-by-bit Methods

8	9	10	11
12	13	14	15
24	25	26	27
28	29	30	31

Bit-by-bit Methods

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

Bit-by-bit Methods

As the name suggests, bit-by-bit methods set one bit at a time.

- Increase accuracy at every iteration.

Bit-by-bit Methods

As the name suggests, bit-by-bit methods set one bit at a time.

- Increase accuracy at every iteration.
- Can often be implemented using only simple operations: add, shift (multiplying/dividing by two), comparison.

Bit-by-bit Methods

As the name suggests, bit-by-bit methods set one bit at a time.

- Increase accuracy at every iteration.
- Can often be implemented using only simple operations: add, shift (multiplying/dividing by two), comparison.
- Bit-by-bit methods exist for many standard math functions.

Bit-by-bit Methods

As the name suggests, bit-by-bit methods set one bit at a time.

- Increase accuracy at every iteration.
- Can often be implemented using only simple operations: add, shift (multiplying/dividing by two), comparison.
- Bit-by-bit methods exist for many standard math functions.
- In particular, trigonometric functions (sine, cosine, etc.) are often calculated using a bit-by-bit algorithm called the CORDIC (**CO**ordinate **R**otation **DI**gital **C**omputer) method.

Bit-by-bit Methods

Bit-by-bit method for calculating $f(x) = \sqrt{x}$:

```
Input: x, n_bits   Output: y = sqrt(x)
base = 2**(n_bits-1)
y = 0
for i = 1,...,n_bits:
    if (y + base)**2 <= x:
        y += base
    base = base / 2
return y
```

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Taylor Series

For a given value a any infinitely differentiable function can be rewritten as:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \dots$$

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Taylor Series

For a given value a any infinitely differentiable function can be rewritten as:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots$$

- This is particularly useful for easily differentiable functions like e^x and $log(x)$.

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Taylor Series

For a given value a any infinitely differentiable function can be rewritten as:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots$$

- This is particularly useful for easily differentiable functions like e^x and $log(x)$.
- Values for $f(a)$ can be precomputed and stored in a table.

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Newton's Method

Newton's Method is used to find the solution to problems of the form $g(x) = 0$. Newton's method repeatedly applies the following update:

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Newton's Method

Newton's Method is used to find the solution to problems of the form $g(x) = 0$. Newton's method repeatedly applies the following update:

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

- May require some manipulation to get updates with a nice form.

Iterative Methods

Assume we want to evaluate the function $f(x)$. One strategy is to pick a number a that is close to x such that we know $f(a)$. Using $f(a)$ as our initial guess we can refine it using a number of methods:

Newton's Method

Newton's Method is used to find the solution to problems of the form $g(x) = 0$. Newton's method repeatedly applies the following update:

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

- May require some manipulation to get updates with a nice form.
- Can be unstable depending on the initialization.

Newton's Method

For example: Let $f(x) = \sqrt{x}$. Note that if we find $1/\sqrt{x}$, we can find $\sqrt{x}$ as $\sqrt{x} = x/\sqrt{x}$. So, let

$$y = \frac{1}{\sqrt{x}}$$
$$0 = y^{-2} - x$$

If we apply Newton's method to solve for y and apply some algebraic simplifications we get updates of the form:

$$y_{n+1} = \frac{y_n}{2}\left(3 - xy_n^2\right)$$