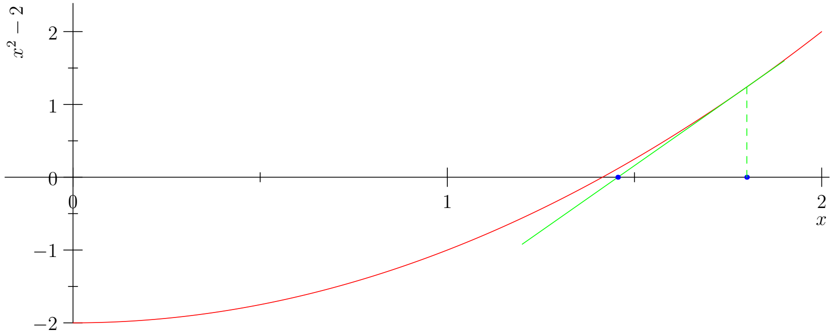
Algorithms and Analysis

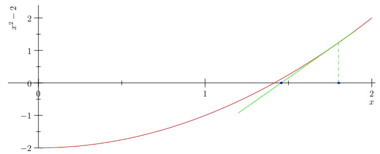
Lesson 29: *Understand Numerics*



Representing reals, rounding error, convergence, stability, conditioning

Outline

1. **Numerical Approximations**
2. Iterating to a Solution
3. Linear Algebra

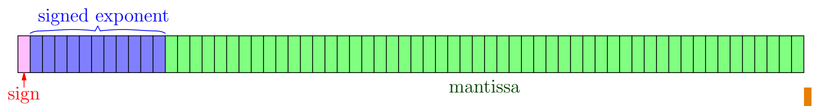


Numerical Analysis

- Numerical algorithms are usually taught separately from the "discrete algorithms" we have predominantly looked at
- The main difference stems from the fact that numerical algorithms model continuous variables
- Computers can only approximate continuous variables
- Numerical algorithms have to take into account this approximation

Representing Reals

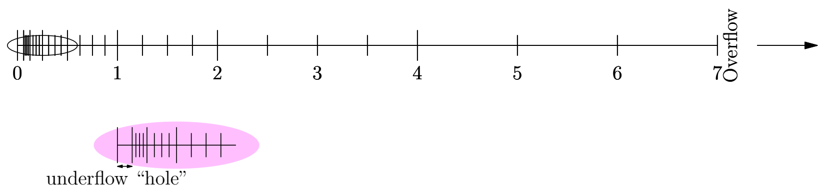
- All real numbers are approximated by a binary encoding



- $x = m \times 2^{e-t}$
- t is precision so that if $e = t$, then $0.5 \leq x < 1$
- For IEEE double $t = 1023$, $expon_{\min} = -1021$, $expon_{\max} = 1024$
- Typical rounding error is $u = 1 \times 10^{-16}$

The Number Line

- We approximate the continuous number line by a set of discrete values
- Imagine using a mantissa of 3 bits and an exponent of 2 bits (and a sign)

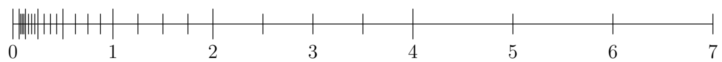


- The rounding error is half the gap between the discrete values

Overflow and Underflow

- An overflow will cause a program to fall over at run time
- An underflow is ignored
- This is usually innocuous, but can lead to trouble
- If you call `log(x)` or `1.0/x` but `x` has underflowed then your program will crash

Rounding Error



- The distance between two real numbers Δx grows with the number such that $\Delta x/x \leq u$ where $u \approx 10^{-16}$ for doubles
- Measure relative error

$$\text{Relative error} = \left|\frac{\text{Approx} - \text{Exact}}{\text{Exact}}\right|$$

- Thus almost every operation is only accurate up to this small (relative) rounding error
- Most operations are carefully designed that these rounding errors are unbiased so that the sum of random errors grows sub-linearly

Losing Precision

- There seems to be plenty of precision, so what’s the problem?
- One issue is that its easy to lose precision
- Consider estimating derivatives by finite differencing

$$f'(x) \approx \frac{f(x+\epsilon) - f(x-\epsilon)}{2\,\epsilon} = \frac{0.841470984861927 - 0.841470984753866}{2.0 \times 10^{-10}}$$

- The problem is $f(x+\epsilon)$ and $f(x-\epsilon)$ are very close so in taking their difference we lose precision
- $f(x) = \sin(x)$, $f'(x) = \cos(x)$ at $x = 1.0$

ϵ	10^{-6}	10^{-8}	10^{-10}	10^{-12}	10^{-14}
relative error	5×10^{-11}	5×10^{-9}	1×10^{-7}	2×10^{-5}	6×10^{-3}

Solving Quadratic Equations

- A classic example where you can lose precision is in solving a quadratic equation $a\,x^2 + b\,x + c = 0$

$$x = \frac{-b \pm \sqrt{b^2 - 4\,a\,c}}{2\,a}$$

- If $b^2 \gg |4\,a\,x|$ then for one solution we end up subtracting numbers very close
- We rather use this equation to compute one solution

$$x_1 = \frac{-b - \mathrm{sgn}(b)\sqrt{b^2 - 4\,a\,c}}{2\,a}$$

- Use the identity $x_1\,x_2 = c/a$ to find x_2

Accumulation of Rounding Error

- With many significant figures surely we can afford to lose some accuracy?
- This is sometimes true, but we often use "for loops" where we might be losing accuracy all the time

```
x = 1.6;
for (i=0; i<50; i++)
   x = sqrt(x);
for (i=0; i<50; i++)
   x = x*x;
```

1.6
1.264911064067352
1.124682650380698
1.060510561183008
1.029810934678307
1.014796006435927
1.007370838587224
1.003678653049483
1.001837638067907
1.000918397307147
1.000459093270258
1.000229520295346

- Gave the answer 1.2840 (if I run the for loop 60 times it gives the answer 1 for almost any input)

Coping With Truncation Errors

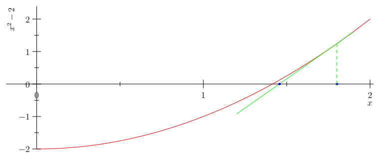
- Nothing is exact so to check that $x = y$ we use

```
Math.abs(x-y) < 1.0e-10  // a small constant
```

- Sometimes sums that add up to 1 don't quite so we have not to rely on anything being exact
- Avoid operations that are likely to lose accuracy (e.g. by taking the difference of similar numbers) where possible
- Sometimes it pays to do some operations using higher precision **long double**
- Make sure that errors are unbiased

Outline

1. Numerical Approximations
2. **Iterating to a Solution**
3. Linear Algebra



Iterative Algorithms

- We solve many numerical tasks by obtaining successively better solutions

$$x^{(0)},\ x^{(1)},\ x^{(2)},\ x^{(3)},\ x^{(4)},\ \ldots$$

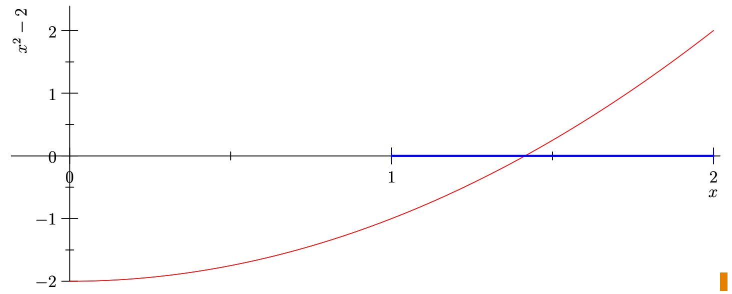
- We often stop when the change in solution is below some threshold, e.g. $|x^{(i+1)} - x^{(i)}| \leq \epsilon \approx u$
- The time complexity depends on the speed of convergence
- This can range from very fast to miserably slow

Bisection

- Suppose we want to compute $\sqrt{2}$ (without using `sqrt(2)`)

$$f(x) = x^2 - 2 = 0$$

- One of the classic methods of solving $f(x) = 0$ is **bisection**

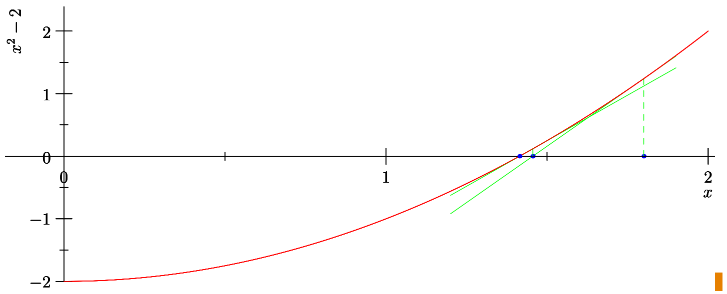


Newton Raphson

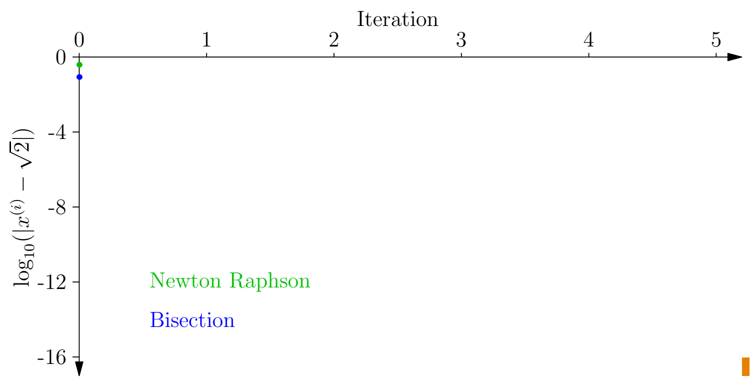
- A second classic method to solve $f(x) = 0$ is Newton-Raphson's method

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}$$

- For $f(x) = x^2 - 2$ so $x^{(i+1)} = ((x^{(i)})^2 - 1)/(2\,x^{(i)})$



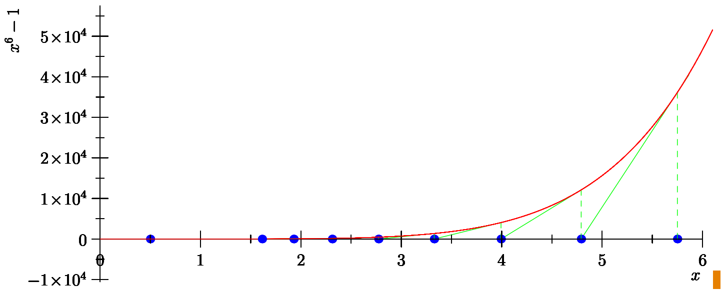
Convergence



- Bisection shows linear convergence (exponential increase in accuracy)
- Newton Raphson shows quadratic convergence

Beware of Asymptotic Convergence

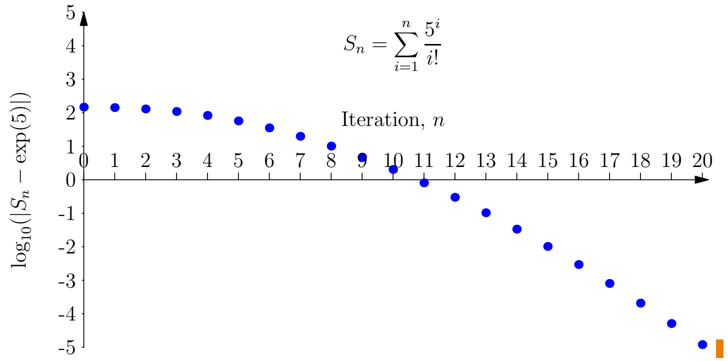
- Newton Raphson only converges quadratically if you start close enough to the solution
- Consider solving $x^6 - 1 = 0$ starting with $x^{(0)} = 0.5$



Evaluating Functions

- We can evaluate many functions using a series expansion

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$



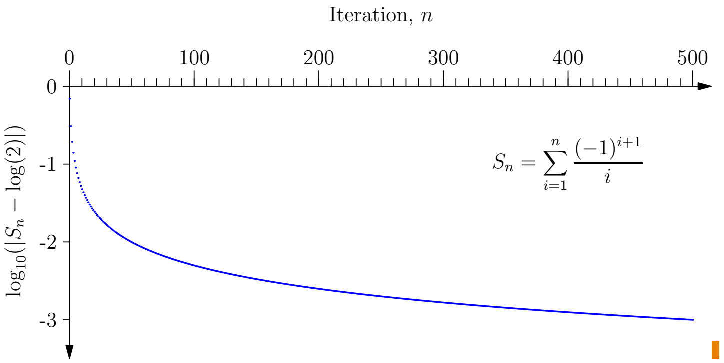
- For large i this converges since $i! \gg x^i$

Slow convergence

- Some expansions converge rather slowly (or even diverge)

$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \cdots$$

- Converges for $-1 < x \leq 1$, but converges slowly for $x = 1$



Convergence

- Many functions can be approximated by a sum
- We get a truncation error by taking only a finite number of elements
- We want the truncation error to be around machine accuracy
- For quick evaluation we need a strongly convergent series
- This often depend on the value of the argument we give to the function
- Most special functions are approximated by different series depending on the input argument

Differential Equations

- Differential equations are used in many applications, for example in modelling the motion of object
- A typical equation of motion might be

$$\frac{\mathrm{d}^2 x(t)}{\mathrm{d}t^2} = 2\,\frac{\mathrm{d}x(t)}{\mathrm{d}t} + 3\,x(t)$$

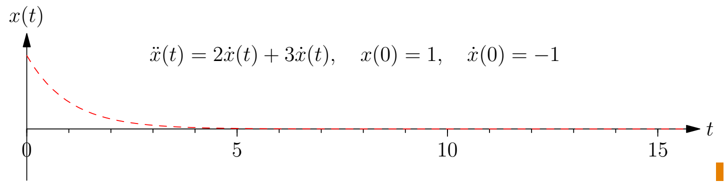
- Which has a general solution $x(t) = c_1 \mathrm{e}^{-t} + c_2 \mathrm{e}^{3\,t}$
- The constants are determined by initial conditions, for example, if $x(0) = 1$ and $\dot{x}(0) = -1$ then $x(t) = \mathrm{e}^{-t}$

Euler's Method

- To solve a differential equation we use an approximate update equation

$$x(t+\epsilon) \approx x(t) + \epsilon \dot{x}(t)$$
$$\dot{x}(t+\epsilon) \approx \dot{x}(t) + \epsilon \ddot{x}(t)$$

- This becomes more exact as $\epsilon \to 0$

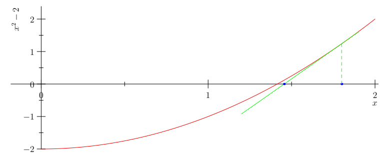


Stability

- Some iterative equations are unstable
- Round off errors can push a system of equations towards an unstable solution
- This can sometimes be overcome by cunning (e.g. running the equations backwards)
- Finding stable algorithms and avoiding unstable algorithms can be key to getting accurate predictions

Outline

1. Numerical Approximations
2. Iterating to a Solution
3. **Linear Algebra**



Solving Simultaneous Equations

- When problems involve many variables it is convenient to use matrices and vectors to store the numbers

$$3x + 2y = 5 \qquad 7x - 8y = -11 \qquad \begin{pmatrix} 3 & 2 \\ 7 & -8 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 5 \\ -11 \end{pmatrix}$$

- Or $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ with solution $\boldsymbol{x} = \mathbf{A}^{-1}\mathbf{b}$
- Linear algebra is an abstraction allowing mathematicians, scientists and engineers to write solutions at a higher level
- The job of the numerical analyst is to write the code that does this

Linear Algebra

- There are a large number of problems with matrices that people care about
- The solution often depends on the problem
- These include
  - Multiply matrices together
  - Solving linear equations $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$
  - Finding eigenvalues of symmetric and non-symmetric matrices
  - Performing singular valued decomposition
- These are important tasks that need to be done efficiently and reliably

Solving Linear Equations

- We consider the classic problem of solving $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$
- Although we can solve this by computing $\mathbf{A}^{-1}\boldsymbol{b}$, finding the inverse of a matrix is typically a $\Theta(n^3)$ operation
- It is preferable to decompose $\mathbf{A}$ into a product of a lower triangular matrix $\mathbf{L}$ and an upper triangular matrix $\mathbf{U}$ which takes $\Theta(n^2)$ operations

$$\mathbf{A} = \begin{pmatrix} 4 & 2 & 6 \\ 3 & 5 & 9 \\ 1 & 2 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0.75 & 1 & 0 \\ 0.25 & 0.428 & 1 \end{pmatrix} \begin{pmatrix} 4 & 2 & 6 \\ 0 & 3.5 & 4.5 \\ 0 & 0 & -4.28 \end{pmatrix} = \mathbf{LU}$$

- Solving $\boldsymbol{x} = \mathbf{U}^{-1}(\mathbf{L}^{-1}\boldsymbol{b})$ is also $\Theta(n^2)$ because of the structure of $\mathbf{L}$ and $\mathbf{U}$

LU-Decomposition

- LU-decomposition is achieved by Gaussian-elimination
- This is a straightforward procedure, but if done carelessly can lead to large rounding errors
- The standard solution is to permute the rows of the matrix (aka pivoting) to prevent loss of accuracy
- In addition we can "polish" solutions

$$\mathbf{A}(\boldsymbol{x} + \delta\boldsymbol{x}) - \boldsymbol{b} = \boldsymbol{\epsilon}$$

- Thus $\delta\boldsymbol{x} = \mathbf{A}^{-1}\boldsymbol{\epsilon}$ which we can use to get an improved estimate of $\boldsymbol{x}$

Norms

- With some work we can get a good approximation to $\boldsymbol{x}$ such that $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$
- But what if we have some error in $\boldsymbol{b}$, this induces an error $\delta\boldsymbol{x} = \mathbf{A}^{-1}\,\delta\boldsymbol{b}$
- How big is $\delta\boldsymbol{x}$?
- To measure the size of a vector we use a norm $\|\delta\boldsymbol{x}\|$, which is a number encoding the size of $\delta\boldsymbol{x}$
- There are a number of different norms, e.g.

$$\|\delta\boldsymbol{x}\|_2 = \sqrt{\delta x_1^2 + \cdots + \delta x_n^2}, \quad \|\delta\boldsymbol{x}\|_1 = |\delta x_1| + \cdots + |\delta x_n|$$

Conditioning

- The size of the error is given by

$$\|\delta\boldsymbol{x}\| = \|\mathbf{A}^{-1}\delta\boldsymbol{b}\| \leq \|\mathbf{A}^{-1}\|\,\|\delta\boldsymbol{b}\|$$

- Where $\|\mathbf{A}^{-1}\|$ provides a measure of the size of the error in the worst case
- For large matrices $\|\mathbf{A}^{-1}\|$ can be large meaning that any error in $\boldsymbol{b}$ is potentially magnified significantly
- Such matrices are said to be ill-conditions
- Ill-conditioning is not to due with rounding errors but the structure of the matrix

Linear Algebra

- Linear algebra packages provide an important set of tools used for solving linear equations
- Care has to be taken to ensure that needless operations (such as inverting a matrix) are not done
- Algorithms must ensure that as little accuracy as possible is lost (e.g. by permuting rows in LU-decomposition)
- Even when the algorithms are precise, small errors can get amplified in some operations, which requires care in formulating the problem
- The idea of poor conditioning (errors being amplified) is useful in understanding many numerical tasks

Lessons

- Be wary of numerical algorithms, because computers approximate real numbers you don't always get what you expect
- Don't avoid numerical algorithms, they are hugely important with vast areas of applications
- This is a well studied area with large libraries of reliable algorithms that work most of the time
- There are some good books such as "Numerical Recipes" by Press, *et al.*, which describes the issues and provides code