Computers in Engineering COMP 208

Root Finding

Michael A. Hawker

Root Finding

- Many applications involve finding the roots of a function f(x).
- That is, we want to find a value or values for x such that f(x)=0

Roots of a Quadratic

- We have already seen an algorithm for finding the roots of a quadratic
- We had a closed form for the solution, given by an explicit formula
- There are a limited number of problems for which we have such explicit solutions

Root Finding

- What if we don't have a closed form for the roots?
- We try to generate a sequence of approximations $x_1, x_2, \dots, x_n$ until we (hopefully) obtain a value very close to the root

Example: Firing a Projectile

Find the angle at which to fire a projectile at a target

Given:

- the velocity, *vel*
- the distance to the base of the target, *x*
- the height of the target, *hgt*

Find: the angle at which to aim, *ang*

Example: Firing a Projectile

The physics of the problem tells us that

```
hgt = vel*sin(ang)*t - ½*g*t²
t = x / (vel*cos(ang))
```

where g is the gravitational constant.

Example: Firing a Projectile

Taking the equations:

```
hgt = vel*sin(ang)*t - ½ g*t²
t = x / (vel*cos(ang))
```

By substituting, we have

```
hgt = x*tan(ang)
    - 0.5*g*(x²/(vel² cos²(ang))
```

The angle `ang` is a root of

```
f(a) = x*tan(ang)
    - 0.5*g*(x²/(vel²*cos²(ang)) - hgt
```

Nov. 8th, 2007 Root Finding 7

The Bisection Method

- We start with an interval that contains exactly one root of the function
- The function must change signs on that interval.
- (If the function changes signs, in fact there must be an odd number of roots in the interval)

Nov. 8th, 2007 Root Finding 8

The Bisection Method

- To get started, we must bracket a root
- How do we bracket the root?
  - From our knowledge of the function
  - By searching along axis at fixed increments until we find that the sign of f(x) changes

Nov. 8th, 2007 Root Finding 9

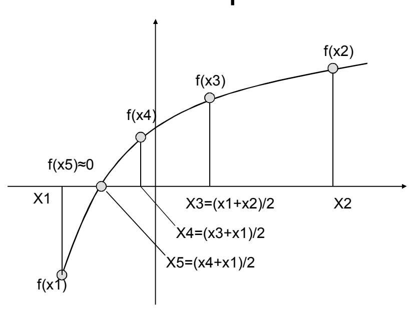
The Bisection Method

- Once we have an interval containing the root(s), we narrow down the search
- Similar to binary search: divide the interval in half and look for a sign change in one of the two subintervals half of the interval.
- From the Intermediate Value Theorem, if there is a sign change in an interval, there must be a root must be in that interval

The Bisection Method

- We check the first subinterval for a change in sign.
- If there is one, that interval must have a root.
- If there is no change in sign, a root must be in the other half
- When do we stop?
  - If the length of the interval is very small, we must be close to the root.
  - Just take the midpoint as the approximation

Bisection example



Convergence Condition

- Root finding algorithms compute a sequence of approximations to the root, r, of f: $x_1, x_2, \ldots, x_i, \ldots$
- When does the bisection method stop?
- We know the root must be between x_i and x_j.
- When these values are very close, we must be close to the root.

A Function Argument?

- We want to write a bisection function that takes a function as an argument and returns a root of the function
- How can a function be an argument?
- We have to go back to first principals

Functions as Arguments

- How can a function be an argument?
  - The code defining a function is stored in memory, just like data
  - It has an address just like any block of memory cells
  - We can pass the address of that code
  - We just have to be careful about the type of the pointer

Function Pointer Syntax

- Want to point to an existing function
- void (*foo)(int);
  - A pointer called "foo" to a function which:
    - Returns void
    - Takes an integer parameter
- In General:

  *return_type (*pointer_name) (parameter_list)*

Function Pointer Examples

- void *(*foo)(int *)
  - Returns a void* (anything)
  - Takes an Integer Pointer
- int (*bar) (int, int)
  - Returns an Integer
  - Takes two Integer parameters
- double (*baz) (double)

Function Pointer Example

```
#include <stdio.h>
double johnny_five(double x)
{
    return x+5.0;
}

int main()
{
    double(*func_ptr)(double);
    func_ptr = &johnny_five;

    printf("%lf\n", func_ptr( 0.0 ));

    return 0;
}
```

Bisection Header

- To define the bisection function we can use the header:

```
double bisection_rf(double (*f) (double),
        double x0, double x1, double tol)
```


Bisection Header

C provides a typedef declaration that can simplify this code:

```
typedef double (*DfD) (double);

double bisection_rf(DfD f, double x0,
  double x1, double tol)
```


The Bisection Method

```
typedef double (*DfD) (double);

double bisection_rf(DfD f, double x0, double x1,
                    double tol){
   double middle = (x0 + x1) / 2.0;

   if ((middle - x0) < tol)
     return middle;
   else if (f(middle) * f(x0) < 0.0)
     return bisection_rf(f, x0, middle, tol);
   else
     return bisection_rf(f, middle, x1, tol);
   }
```

Other Convergence Conditions

There are typically three ways of determining when to stop

1. $f(x_i)$ is close to zero
2. $f(x_i)$ is close to r
3. x_i is close to x_{i+1} so it doesn't pay to continue

The Secant Method

- We also begin with two initial approximations
- However they do not have to bracket the root
- We essentially approximate the function using straight lines forming the secant at the two points
- This is probably the most popular method
- It is not guaranteed to converge to the root

The Secant Method

- Start with two values, x_0 and x_1 that don't necessarily bracket the root
- Compute a new approximation, the point at which the line drawn between $f(x_0)$ and $f(x_1)$ intersects the x-axis

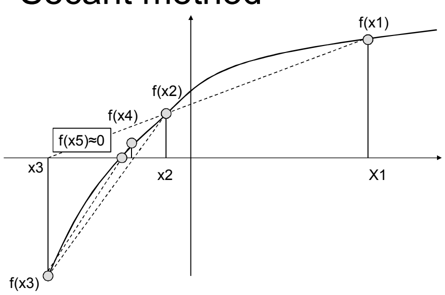
Computing the Approximation

The approximation is given by:

```
x2 = f(x0)*(x1-x0)/(f(x0)-f(x1))+x0
```

Iterate this process using x_1 and x_2 as the new pair of points

Secant method



Convergence Criteria

- When do we stop this process?
- We use the first of the criteria we described
- That is, we stop when the value of $f(x_i)$ is close to zero
- We then say that x_i is an approximate root

The Secant Method

- This method is one of the most popular ones in use
- It may not converge because successive intervals become larger or because it oscillates
- Therefore we terminate the algorithm after a specified number of steps if it has not converged

The Secant Method

```
double secant_rf(DfD f, double x1, double x2,
                            double tol, int count){
   double f1 = f(x1), f2 = f(x2),
          slope = (f2 - f1) / (x2 - x1),
          distance = -f2 / slope,
          point = x2 + distance;
   if(!count)
       return point;
   if(fabs(f(point)) < tol)
       return point;
   return secant_rf(f, x2, point, tol, count - 1);
}
```


The False Position Method

- The secant method uses the most recent approximation and the previous one
- Even if the root was bracketed, it may not remain bracketed
- The False Position Method combines the secant method with bisection to guarantee that the root remains bracketed

Regula Falsi Method

- Regula falsi is another common name for "false position"
- It can be thought of as a refinement of bisection
- Instead of using the midpoint of the interval we use the secant to interpolate the value of the root

The False Position Method

```
double fixedpoint_rf(DfD f, double x1, double x2,
                     double tol){
  double f1 = f(x1), f2 = f(x2),
         slope = (f2 - f1) / (x2 - x1),
         distance = -f1 / slope,
         point = x1 + distance;

  if(fabs(f(point)) < tol)
      return point;
  if((f1 * f(point)) < 0)
      return fixedpoint_rf(f, x1, point, tol);
  else
      return fixedpoint_rf(f, point, x2, tol);
}
```


Newton-Raphson

- Newton's method starts with a single initial guess at the root, x_0
- It is like the secant method with the value of the derivative replacing the slope
- At each step, the next value is given by

  $$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

  where f' is the derivative of f

Convergence Criteria

- This method could use the third convergence condition to terminate
- That is, it could terminates when xi and xj are very close to each other
- In our implementation we use the first, that f(xi) is close to zero

Newton-Raphson

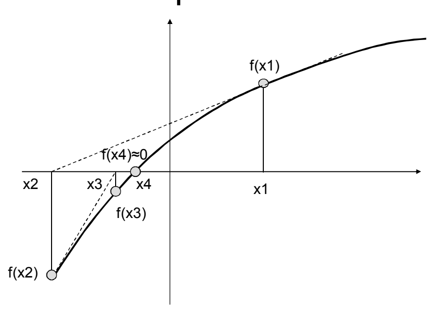
For example, to compute the square root of a number, a, we can find the root of the function $x^2 - a$

Newton's method guarantees that we keep getting closer to the root

However, in general we may not converge to a root

Therefore we stop after a certain number of steps (count) if we have not yet found a root

Newton-Raphson



Newton_Raphson

```
double newton_rf(DfD f, DfD df,
                 double x,
                 double tol, int count){

  double distance = -f(x) / df(x);
  x += distance;
  if((fabs(f(x)) < tol) || !count)
     return x;
  return newton_rf(f, df, x, tol, count - 1);
}
```

Nov. 8th, 2007 Root Finding 37

The Derivative

- This algorithm assumes that we know the derivative of the function
- If the function is complex or we generate values of the function empirically without having an explicit analytic form for the function, we may have to estimate the derivative
- Using a “centered three point” method, we can rewrite the algorithm as follows

Nov. 8th, 2007 Root Finding 38

Newton-Raphson with Numerical Differentiation

```
double newton_rf(DfD f, double x,
                 double tol, int count){

  double distance =
          -f(x) / centered3_diff(f,x,1e-6);
  x += distance;
  if((fabs(f(x)) < tol) || !count)
     return x;
  return newton_rf(f, x, tol, count - 1);
}
```

Nov. 8th, 2007 Root Finding 39

Numerical Differentiation

- Engineers often deal with functions represented as a collection of data points
- We might not have an analytic closed form for the function
- For example, we might measure the position of a vehicle at different points in time

Numerical Differentiation

- Given the position of a vehicle at different points in time:
  - To compute the velocity, i.e. the derivative, we must compute an estimate based on the observed position
  - Knowing the value of the function at two different points in time allows us to approximate the derivative

Two Point Approximations

Central Difference Formula:

```
f'(t) = (f(t+h)-f(t-h))/2h
```

Forward Difference Formula:

```
f'(t) = (f(t+h)-f(t))/h
```

Backward Difference Formula:

```
f'(t) = (f(t)-f(t+h))/h
```

Two Point Differentiation

- The error is O(h)
- If h is big, the approximation will not be very accurate
- If h is small, there may be large roundoff errors.
- It might not be possible to sample the data at intervals for which h is very small

Three Point Differentiation

Three point methods are more precise

We look at the formulae for three methods but in this course we will not derive them

- Forward 3 point differentiation
- Backward 3 point differentiation
- Centered 3 point differentiatio

Forward three point

$$\frac{dy}{dx} = \frac{-3*f(x)+4*f(x+\Delta x)-f(x+2*\Delta x)}{2*\Delta x}$$

```
double forward3_diff(DfD f, double x,
                     double h){
  return (-3*f(x) + 4*f(x+h) - f(x+2*h)) /
         (2 * h);
}
```

Backward three point

$$\frac{dy}{dx}=\frac{f(x-2*\Delta x)-4*f(x-\Delta x)+3*f(x)}{2*\Delta x}$$

```
double backward3_diff(DfD f, double x,
                      double h){
  return (f(x - 2*h) - 4*f(x - h) + 3*f(x))
         / (2 * h);
}
```

Nov. 8th, 2007 Root Finding 46

Centered three point

In deriving the centered three point formula, terms cancel out and the final result is the same as the centered two point

This has the fewest function evaluations and is as accurate as the 3 point methods

```
double centered3_diff(DfD f, double x,
                      double h){
  return (-f(x - h) + f(x + h)) / (2 * h);
}
```

Nov. 8th, 2007 Root Finding 47