Contents

Hilary Term

Summary of Numerical Analysis for this term

- Root finding; maxima and minima
- Ordinary differential equations (ODE)
- Numerical integration techniques
- Matrices & vectors – addition, multiplication, Gaussian elimination, etc

Sources of error in numerical calculation

Real World

↓

Mathematical Model

↓

Computer

Solving Problems

Using

integration

differentiation

matrix determinants

.

ANALYTICALLY: the "pen and paper" method - you can find the exact solution.
NUMERICALLY: use computational techniques to find a solution. It might not be exact. Therefore it is important to understand and quantify errors.

Chapter 1

Root Finding

This is a basic computational task - to solve equations numerically.
ie. find a solution of

$$f(x) = 0$$

1 independent variable ie x implies a 1-dimensional problem $\Rightarrow$ solve for the roots of the equation.

Definition If α is the root of a function $f(x)$ then $f(\alpha) = 0$.

DRAWING

To find the root numerically

- find 2 values of x, say a & b, that BRACKET the root;
- decrease the interval $[a, b]$ to converge on the solution.

This is known as Bracketing and Bisection.

1.1 Bracketing and Bisection

A root is BRACKETED in an interval (a, b) if $f(a)$ and $f(b)$ have opposite signs.
DRAWING

$$\left.\begin{array}{l} f(a) < 0 \\ f(b) > 0 \end{array}\right\} \rightarrow \ a, b \text{ bracket a root}$$

Between a and b, $|f(x)|$ decreases until

$$f(x_1) = 0$$

where x_1 is a root of the function $f(x)$.

1.1.1 Finding the root numerically

Given a function $f(x)$ and an initial range x_1 to x_2

- Check for a root between x_1 and x_2
  ie
  $$f(x_1) * f(x_2) < 0$$

IF true $\Rightarrow$ there is a root in (x_1, x_2)
ELSE
expand the range and try again.
When the interval contains a root start BISECTING to converge on it.
BISECTING PROCEDURE:

- in some interval the function passes through zero because it changes sign;
- evaluate function at interval MIDPOINT and examine its sign;
- use midpoint to replace whichever limit has same sign.

DRAWING
When to stop?

1. After a fixed number of bisection iterations eg 40
2. When you reach the 'CONVERGENCE CRITERIA'.

CONVERGENCE:
Computers use a fixed number of bits to represent floating point numbers. So, while the function may ANALYTICALLY pass through zero, its COMPUTED (NUMERICAL) value may never be zero, for any floating point argument.

- You must decide what accuracy on the root is attainable.

- A good guide: continue until interval is smaller than

$$\varepsilon \frac{(|x_1| + |x_2|)}{2}$$

where ε = machine precision ($\approx 10^{-12}$) and (x_1, x_2) = original interval

1.1.2 Pseudo BRACKET code

Choose initial interval (a, b)
Check

```
IF(f(a)*f(b)<0.0){
Call BISECTION CODE
}
ELSE IF(abs(f(a)) < abs(f(b)){
expand interval to the left
a=a+factor*(a-b)
}
ELSE{
expand interval to the right because abs(f(a)) > abs(f(b))
b=b+factor*(b-a)
}
re evaluate (f(a)*f(b)<0) and repeat til true.
```

factor - you choose between 1 - 2 to extend the range, by a 'little'.
DRAWING

1.1.3 Drawbacks

- It there are and EVEN number of roots in an interval, bisection won't find any ⇒ no sign change
  DRAWING

- it can converge to a pole rather than a root
  DRAWING

Because we only look at the sign of $f(x)$ not $|f(x)|$ appears like a root.

- Check $|f(x)|$ of final answer, it should be small for a root, will be large it a pole.
- there are faster converging methods
- doesn't generalise to complex variables or several variables

1.1.4 Tips for success with Bracketing & Bisection

- Get an idea what the function looks like
- Good initial guess important
- check that $|f(x)| \approx 0$ for x a root

1.1.5 Virtues

- If an initial bracket is found it will converge on the root-regardless of interval size.
- easy to decide reliably when the approx is good enough
- converges reasonably quickly and independent of the function smoothness.

1.1.6 Pseudocode to Bisect an interval until a root is found

```
#define EPS 1e-12

double root(double f(double), double a, double b){

double m = (a+b)/2.0;

if(f(m)==0.0||fabs(b-a) <EPS){

return m;

}else if(f(a)*f(m)<0.0){

return root(f,a,m);

}else{

return root(f,m,b);

}

}
```

An aside on programming:
'C' function 'root' takes as its first argument a function → more precisely this is a pointer to a function.
In 'C' a function name by itself is treated as a pointer to that function.
cf. how 'C' treat array names.
→ in 'root' when we pass 'f' we actually pass the address of 'f'

Calling 'root' from 'main'

```
#include ...
```

```
    .
    .
    .
double func_form(double);
double root(double f(double), double, double);
main()
{
    .
    .
    .

/* find an interval bracketing a root */
/* (a,b) */

/* Now call 'root' */
solution = root(func_form,a,b);

}

double func_form(double x)
{
return (x*x*x*x-7.0*x-3.0);
}
}
double root( .... )
   .
   .
   .
```

1.2 Root Finding – Newton-Raphson method

The Newton-Raphson method requires evaluation of
$f(x)$

and
$f^{'}(x)$ - the derivative at arbitrary points of x.
METHOD:

- extend tangent line at current pt. x_i until it crosses zero
- set the next guess, x_{i+1} to abscissa of that zero crossing

DRAWING

Algebraically, the method derives from a Taylor series expression of a function, f about a point, x.

$$f(x+\delta) \approx f(x) + f^{'}(x)\delta + \frac{f^{''}(x)}{2!}\delta^2 + \frac{f^{'''}(x)}{3!}\delta^3$$

where
$f^{'}(x)$ - first derivative of f wrt x
$f^{''}(x)$ - second derivative of f wrt x
$f^{'''}(x)$ - third derivative of f wrt x
where δ is small and f is a well-behaved function, terms beyond the linear term are unimportant. ie

$$f(x+\delta) \approx f(x) + f^{'}(x)\delta$$

So if there's a root at $(x+\delta)$ say

$$f(x+\delta) = 0 = f(x) + f^{'}(x)\delta$$

from this we get

$$\boxed{\delta = -\frac{f(x)}{f^{'}(x)}}$$

In words:

at $x \Rightarrow$ know $f(x)$, $f^{'}(x)$ if there is a root $x+\delta$ then

$$f(x+\delta)=0$$

and δ is the distance you move from x,
if $f(x+\delta)=0$ then

$$\delta = -\frac{f(x)}{f^{'}(x)} \text{ by Taylor's expansion}$$

So if $\delta \sim 0$ then we've found a root at $f(x)$ because $f(x) \sim 0$.
Therefore the condition to find a root with Newton -Raphson (N-R) is

$$\begin{aligned} \delta &\sim 0 \\ \Rightarrow \frac{f(x)}{f^{'}(x)} &\sim 0 \end{aligned}$$

Newton–Raphson Formula

If we have x_i, how do we choose a new x_{i+1}.

$$\text{new guess} \; = \; \text{guess} \; + \delta$$

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

N-R can give "grossly inaccurate, meaningless results".
Consider
An initial guesses, far from the root so the search interval includes a local max. or min.
$\Rightarrow$ Bad News because $f^{'}(x)=0$ at a local max or min.
DRAWING.

DRAWING 2.

So, Why use N-R?
It has good convergence.
Proof
If ε is the distance x to the true root, x_{true}
Then at iteration step i

$$x_i + \varepsilon_i = x_{true}$$

$$x_{i+1} + \varepsilon_{i+1} = x_{true}$$

There for

$$x_{i+1} + \varepsilon_{i+1} = x_i + \varepsilon_i$$

$$x_{i+1} - x_i = \varepsilon_i - \varepsilon_{i+1}$$

Using Taylor

$$-\frac{f(x_i)}{f^{'}(x_i)} = \varepsilon_i - \varepsilon_{i+1}$$

$$\Rightarrow \varepsilon_{i+1} = \varepsilon_i + \frac{f(x_i)}{f^{'}(x_i)}$$

When a trial solution x_i differs from the root by ε_i we can write

$$\begin{aligned} f(x_i+\varepsilon_i) = \quad 0 \quad &= f(x_i) + \varepsilon_i f^{'}(x_i) + \varepsilon_i^2 \frac{f^{''}(x_i)}{2!} + ... \\ &= \frac{f(x_i)}{f^{'}(x_i)} + \varepsilon_i + \frac{\varepsilon_i^2}{2!}\frac{f^{''}(x_i)}{f^{'}(x_i)} + ... \\ &= \frac{f(x_i)}{f^{'}(x_i)} + \varepsilon_i + \frac{\varepsilon_i^2}{2!}\frac{f^{''}(x_i)}{f^{'}(x_i)} \end{aligned}$$

$$\varepsilon_{i+1} = -\frac{\varepsilon_i^2}{2}\frac{f''(x)}{f'(x)}$$

This is recurrence relation for derivations of the trial solution
$\Rightarrow$ N-R converges quadratically (ε_i^2) *near the root*
this means it has a poor global solution but good local convergence.
◇
The convergence criteria for **Bracketing & Bisection** is

$$\varepsilon_{i+1} = \frac{1}{2}\varepsilon_i$$

Which is linear convergence.
This means: near a root, the number of significant digits approximation doubles with each step.
Good tip:
Use the more stable 'bracketing and bisection' to find a root, to "polish up" the solution with a few N-R steps.

Writing a program to solve a problem with N-R

We need $f(x)$ and $f'(x)$ from the user specific to the problem being solved
If you only get $f(x)$ – could use N-R with a numerical approximation to the derivative.
EXAMPLE

1.2.1 Specific Code $x^2 - 4$

```
#include ...

#define xacc 1e-12
#define JMAX 20
main()
{
double x1,x2,fx,df,dx,rtn;
```



```
int j,i;
x1=1.0;
x2=2.5;
rtn=0.5*(x1+x2); /* first guess */

for(j=1;j<=JMAX;j++)
{
fx=rtn*rtn-4; /* (x*x-4) */
df=2.0*rtn;
dx=-fx/df;
rtn+=dx; /* x=x-fx/df  */
 /* x=x+delta  */
 /* x_i+1=x_i+delta  */

if((x1-rtn)*(rtn-x2)<0.0){
printf("jumped outside of bounds");
exit(1);
}

if(fabs(dx)<xacc){
printf("found root after %d attempts at %lf \n",j,rtn);
exit(0);
}
}
printf("Error - exceeded max tries - no root");
}
```

1.2.2 A word on the roots of Polynomials

There are a number of methods – useful for most practical problems
Eg
Muller's method

Laguerres method

Eigenvalue method

.

.

We don't have time for these – any good Numerical Analysis book has them

...

Keep in mind, a real polynomial of degree 'n' has 'n' roots.

They can be real or complex, and might not be distinct.

If polynomial coeffs are real – complex roots in conjugate pairs

ie if $x_1 = a + bi$ is a root

then $x_2 = a - bi$ is also root

complex coeffs $\Rightarrow$ unrelated complex roots.

Multiple roots, or closely spaced roots therefore it is most difficult for numerical techniques eg

$$P(x) = (x - a)^2$$

has a double real root at x=a.

Cannot bracket the root in the usual way, nor will N-R work well since function and derivative vanish at a multiple root.

N-R may work - but slowly since large roundoff can occur.

Need special techniques for this.

EXAMPLE