Ch. 7 Nonlinear Algebraic Systems

- motivating examples
  - implicit methods, Backward Euler
    Trapezoid
    BDF ...
    Implicit RK
    ⋮
- review of nonlinear solvers in 1D
- extension to systems of equations

other ... Iserles Thm 7.1 + Proof (p. 125–126).

Recall the Backward Euler Method for

$$\begin{cases} \vec{y}\,' = \vec{f}(t,\vec{y}) \\ \vec{y}(t_0) = \vec{y}_0 \end{cases}$$

$$\vec{y}_{k+1} = \vec{y}_k + h\,\vec{f}(t_{k+1}, \vec{y}_{k+1}) \qquad k = 0, 1, 2, \ldots$$

This is implicit because of the unknown $\vec{y}_{k+1}$ appearing on the right-hand-side. So, rather than evaluate a formula (like forward Euler) to "advance" the solution to the next time step we need to solve a (nonlinear, in general) system of equations. That is, we define a new function

$$\boxed{\vec{F}_{BE}(\vec{z}) \equiv \vec{z} - \vec{y}_k - h\,\vec{f}(t_{k+1}, \vec{z})}$$

that is zero at the desired solution. That is, $\vec{F}(\vec{y}_{k+1}) = 0$.

In the above function $\vec{F}(\vec{z})$ it is assumed that $\vec{y}_k$ is a known quantity (i.e. from the initial condition or previous time step.)

Similarly, for the Trapezoid method we have

$$\vec{y}_{K+1} = \vec{y}_K + \tfrac{1}{2}h\left[\vec{f}(t_K,\vec{y}_K) + \vec{f}(t_{K+1},\vec{y}_{K+1})\right] \qquad K=0,1,2,\ldots$$

Again this must be solved (not just simply evaluated) to find $\vec{y}_{K+1}$. Define

$$\boxed{\vec{F}_{TRAP}(\vec{z}) \equiv \vec{z} - \vec{y}_K - \tfrac{1}{2}h\left[\vec{f}(t_K,\vec{y}_K) + \vec{f}(t_{K+1},\vec{z})\right]}$$

and again the goal is to find roots of this function — i.e. solve $\vec{F}(\vec{y}_{K+1}) = \vec{0}$ to get $\vec{y}_{K+1}$.

For the case of a BDF (Backward Differentiation Formula) we have a similar situation. Consider the BDF $(s=3)$ example (Iserles, p.27, eq. (2.16))

$$\vec{y}_{K+3} - \frac{18}{11}\vec{y}_{K+2} + \frac{9}{11}\vec{y}_{K+1} - \frac{2}{11}\vec{y}_K = \frac{6}{11}h\vec{f}(t_{K+3},\vec{y}_{K+3})$$

$K=0,1,2,\ldots$

Here we introduce the function

$$\boxed{\vec{F}(\vec{z}) = \vec{z} - \frac{18}{11}\vec{y}_{K+2} + \frac{9}{11}\vec{y}_{K+1} - \frac{2}{11}\vec{y}_K - \frac{6}{11}h\vec{f}(t_{K+3},\vec{z})}$$

and seek solutions $\vec{F}(\vec{y}_{K+3}) = 0$.

We've addressed

- convergence
- order (error)
- stability (A-stability)

of all of these types of methods so the key issue at the moment is finding the solution to $\vec{F} = \vec{0}$.

We'll discuss methods for solving nonlinear equations in one dimension as well as methods for solving nonlinear systems of equations — and the related issues.

One Dimension

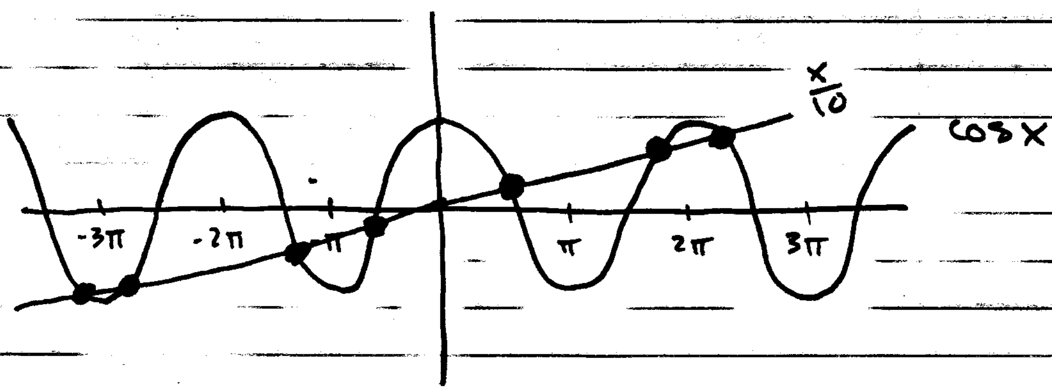
The basic problem is $f(x)=0$

(i.e. find x such that $f(x)=0$).

EX

$$f(x)=\frac{x}{10}-\cos x=0 \quad ; \quad f'(x)=\frac{1}{10}+\sin x$$

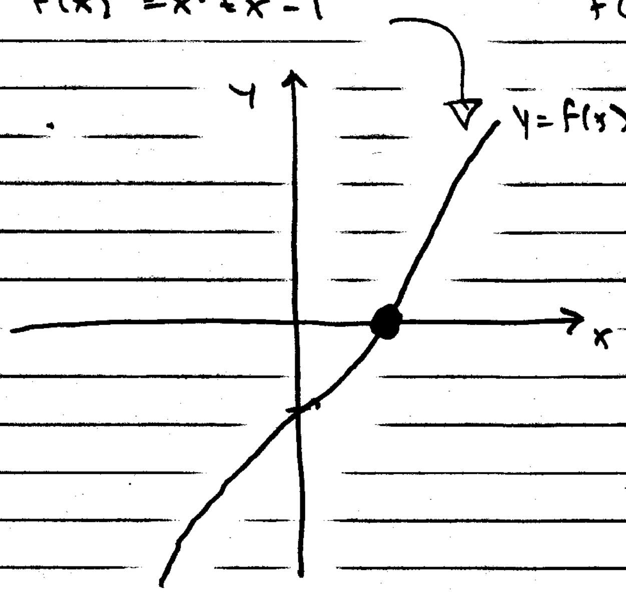
(sign changes)



In this example there are 7 solutions to $f(x)=0$. Clearly if the slope of the line were to change, the number of solutions could easily change.

EX

$f(x)=x^3+x-1$ $\qquad f'(x)=3x^2+1 > 0$



Here $f(x)=0$ has one real solution.

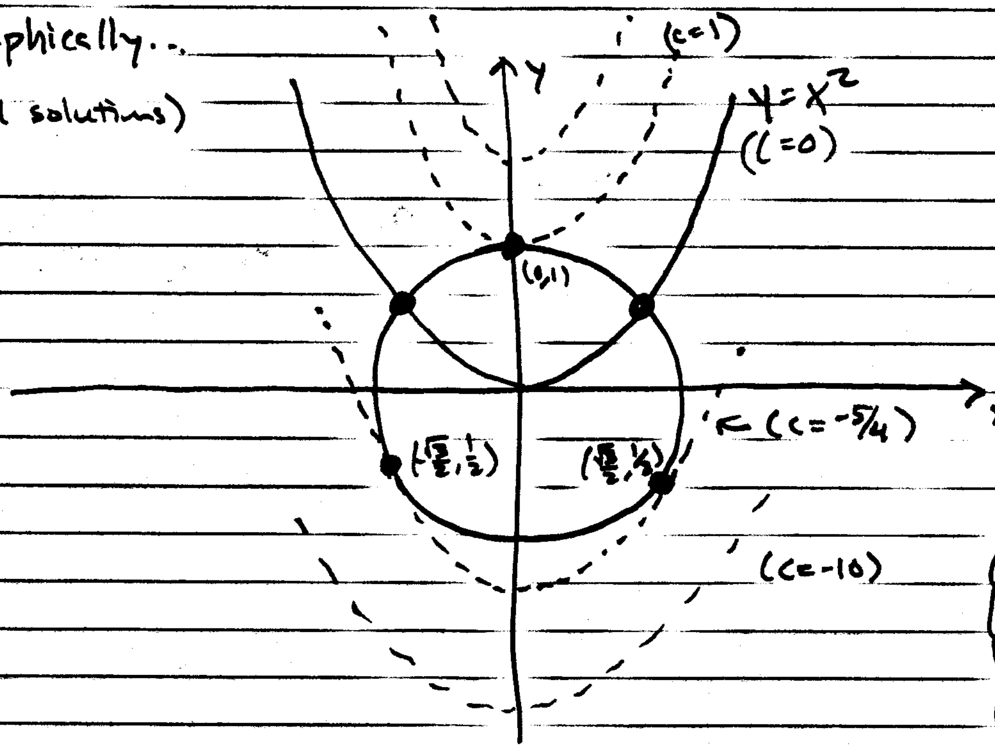
EX (2 dimensions)

$$\vec{F}(\vec{x}) = 0 \qquad \vec{x} = (x, y)$$

$$\vec{F}(\vec{x}) = \begin{bmatrix} f_1(x,y) \\ f_2(x,y) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - 1 \\ y - x^2 - c \end{bmatrix}$$

c = constant parameter.

graphically...
(real solutions)



Zero, one or two solutions depending on C

also 4?

Algebraically... $x^2 + (x^2+c)^2 - 1 = 0$

$$x^4 + (2c+1)x^2 + (c^2-1) = 0$$

$$x^2 = \frac{-(2c+1) \pm \sqrt{(2c+1)^2 - 4(c^2-1)}}{2}$$

$$= \frac{-(2c+1) \pm \sqrt{4c+5}}{2}$$

or ... in terms of y ...

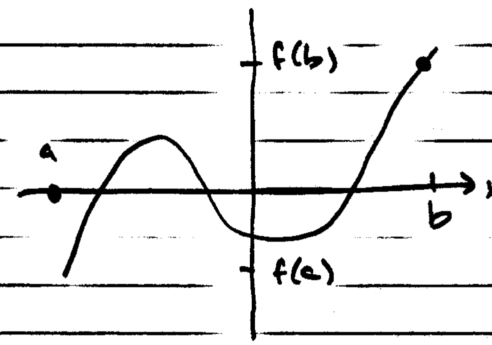
$$y - c + y^2 - 1 = 0 \qquad y = \frac{-1 \pm \sqrt{1+4(c+1)}}{2} = \frac{-1 \pm \sqrt{4c+5}}{2}$$

Some theory

Existence of Solutions

1D: Given a continuous function on a closed interval $[a,b]$ we can apply the intermediate value theorem — i.e. f takes on all values between $f(a)$ and $f(b)$

This applies to root finding for $f(x)=0$ if we can "bracket" the solution i.e. find a and b such that $f(a) \cdot f(b) < 0$.



Then since $f(a)$ and $f(b)$ have different signs and, assuming f is continuous, we know there exists at least one c on (a,b) such that $f(c)=0$.

At least one root exists!

Higher Dim:

In general there is no simple & practical way of "bracketing" the solution (but see Heath, p. 237 and p. 219)

Uniqueness of Solutions

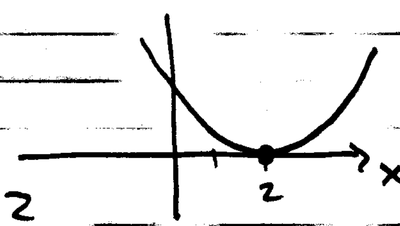
As we've seen in the examples a wide variety of scenerios are possible. In general $\vec{F}(\vec{x})=0$ can have any number of solutions $(0, 1, 2, \ldots, \infty)$

In general, unless you know otherwise, you should be prepared to find more than one solution.

(1D): Degeneracy: occurs when the function and one or more of its derivatives are zero at a root.

Ex

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



$x=2$ is a root of multiplicity 2

$$f(2)=0$$
$$f'(x) = 2(x-2) \quad \text{so } f'(2)=0.$$

In general, if x^* is a root $(f(x^*)=0)$ and $f'(x^*)=0$, $f''(x^*)=0$, $\ldots$, $f^{(n-1)}(x^*)=0$ then x^* is a degenerate root of multiplicity n.

Note:

$$f(x) \approx \cancelto{0}{f(x^*)} + \cancelto{0}{f'(x^*)}(x-x^*) + \ldots + \frac{1}{(n-1)!}\cancelto{0}{f^{(n-1)}}(x^*)(x-x^*)^{n-1}$$

$$\approx \frac{f^{(n)}(x^*)}{n!}(x-x^*)^n + \frac{f^{(n+1)}(x^*)(x-x^*)^{n}}{n!}$$

(Higher Dimensions) : Degeneracy of roots is related to the Jacobian Matrix. For $\vec{F}(\vec{x})$ the Jacobian is defined as

$$J(\vec{x}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \cdots & & \vdots \\ \vdots & & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \cdots & & \frac{\partial F_n}{\partial x_n} \end{bmatrix}$$

If J is singular at a root then a degeneracy occurs.

<u>EX</u>

Recall our previous system

$$\vec{F}(\vec{x}) = \begin{bmatrix} x^2 + y^2 - 1 \\ y - x^2 - c \end{bmatrix}$$

$$J(\vec{x}) = \begin{bmatrix} 2x & 2y \\ -2x & 1 \end{bmatrix}$$

This matrix is singular when $2x + 2x \cdot 2y = 0$

$$2x(1+2y) = 0$$

i.e. when $x = 0$ or $y = -1/2$

We saw these two singular cases in the previous diagram as the "switching" points from 2 to 1 to 0 solutions and from 2 solutions to 0 solutions.

Sensitivity + Conditioning

1D : • The sensitivity associated with evaluating a function $y = f(x)$ near $x = x^*$ can be measured by the absolute condition number

$$\approx \left|\frac{\Delta \text{ output}}{\Delta \text{ input}}\right| = \left|\frac{\Delta y}{\Delta x}\right| \approx |f'(x^*)|$$

• The sensitivity associated with finding a root $y^* = f(x^*) = 0$ is like that for evaluating the inverse function $x^* = f^{-1}(y^*)$

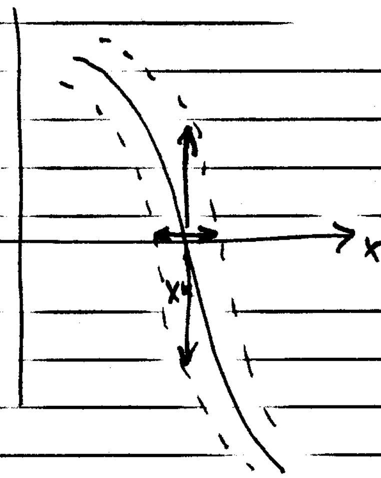
$$= \left|\frac{\Delta \text{ output}}{\Delta \text{ input}}\right| = \left|\frac{\Delta x}{\Delta y}\right| = \left|f^{-1\prime}(y^*)\right| = \frac{1}{|f'(x^*)|}$$

but $f^{-1}(f(x)) = x \implies f^{-1\prime}(f(x))\, f'(x) = 1$

$$\Downarrow$$

$$f^{-1\prime}(y) = \frac{1}{f'(x)}$$

So, if $|f'(x^*)|$ is large



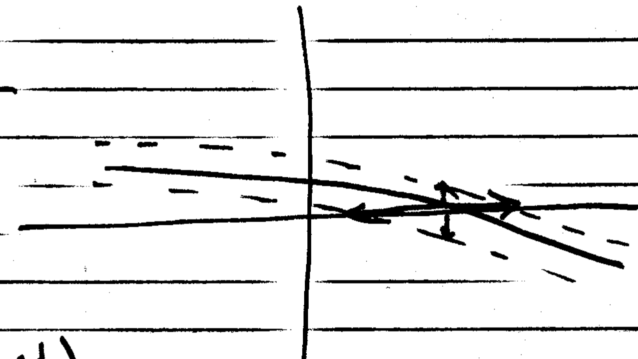
- root finding (well conditioned)

  $\frac{1}{|f'(x^*)|}$ small

- evaluating near x^* (poorly conditioned)

  $|f'(x^*)|$ large

Alternatively, if $|f'(x^*)|$ is small



- root finding (poorly conditioned)

  $\frac{1}{|f'(x^*)|}$ large

- evaluating near x^* (well conditioned)

  $|f'(x^*)|$ small.

In higher dimensions ... sensitivity + conditioning

- absolute condition number for evaluating $\vec{F}(\vec{x})$ near $\vec{x}^*$: $\| J(\vec{x}^*) \|$

- absolute condition number for root finding $\vec{F}(\vec{x}) = \vec{0}$: $\| J^{-1}(\vec{x}^*) \|$ — singular matrix not invertible... (degenerate root case)

In general, root finding requires an iterative process. That is, to find $\vec{x}^*$ with $\vec{F}(\vec{x}^*)=\vec{0}$ we generally use a scheme that generates a sequence $\vec{x}_1, \vec{x}_2, \vec{x}_3, \ldots$ that we hope converges to $\vec{x}^*$.

Convergence Rates

- optimistically, we hope for convergence and so define the notion of convergence rate as follows. First, define the error

$$\vec{e}_k \equiv \vec{x}_k - \vec{x}^*$$

Convergence rates are determined by examining

$$\lim_{k\to\infty} \frac{\| e_{k+1} \|}{\| e_k \|^r} = C$$

(in practice k is "large" rather than $k \to \infty$)

where r is the convergence rate and C is a constant. ~~[illegible]~~ That is, an iterative method is said to converge at rate r if this limit exists with $C > 0$.

$r = 1$: linear convergence

$r = 2$: quadratic convergence

$r = 3$: cubic

$r > 1$: superlinear, etc.

linear: the number of significant digits increases at a constant rate

e.g. $\|e_1\| = 10^{-1}$
$\|e_2\| = 10^{-2}$
$\|e_3\| = 10^{-3}$ ...

$$\lim_{k\to\infty} \frac{\|e_{k+1}\|}{\|e_k\|} = 10^{-1}$$

quadratic: the number of significant digits doubles every iteration

e.g. $\|e_1\| = 10^{-1}$
$\|e_2\| = 10^{-2}$
$\|e_3\| = 10^{-4}$
$\|e_4\| = 10^{-8}$
$\vdots$

very fast convergence once the error is small.

cubic: the number of sig. digits triples every iteration

e.g. $\|e_1\| = 10^{-1}$
$\|e_2\| = 10^{-3}$
$\|e_3\| = 10^{-9}$
$\vdots$

Note: if

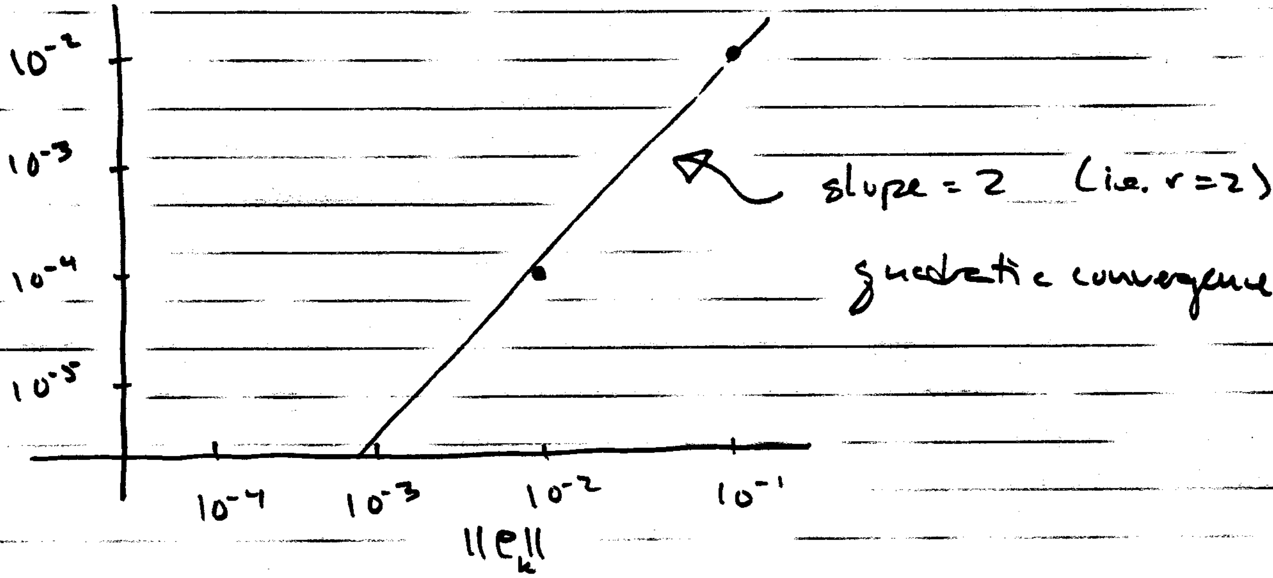
$$\frac{\|e_{k+1}\|}{\|e_k\|^r} \approx C \quad \text{for } k \text{ large}$$

we can write

$$\ln\|e_{k+1}\| - r\ln\|e_k\| = \ln C$$

so $\ln\|e_{k+1}\| = r\ln\|e_k\| + \ln C$

Using log scales (i.e. loglog($\|e_k\|$, $\|e_{k+1}\|$)



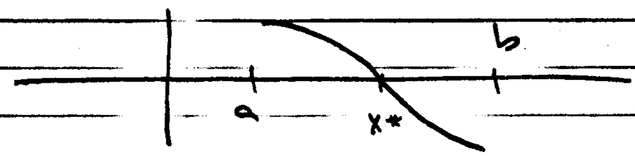
In matlab loglog(x,y) plots x,y in $\log_{10}$ scales.

This procedure of plotting the error can be useful in assessing the convergence properties of a new scheme, checking the convergence properties of a known scheme, detecting degenerate roots (sometimes), . . .

Methods for Solving $\vec{F}(\vec{x}) = \vec{0}$

1D Bisection:

- identify a "bracket"



$f(a) > 0$
$f(b) < 0$
$f(a)f(b) < 0$

- choose midpoint $m = \frac{(a+b)}{2}$ and evaluate $f(m)$

  if $f(m) > 0$
  $\quad a = m$ (m becomes new left endpoint)
  if $f(m) < 0$
  $\quad b = m$ (m becomes new right endpoint)

- repeat until convergence criterion is met...

  e.g.
  $|f(m)| < ftol \leftarrow$ specified tolerance related to f
  $|b-a| < inttol \leftarrow$ specified tolerance related to x^*.

Note $|f(m)| \approx |f'(m)|\,|b-a|$

so depending on $|f'(m)|$ these stopping conditions could differ and using $|f(m)|$ small may not guarantee $|b-a|$ small if the root finding problem is poorly conditioned $|f'(m)|$ small $\left(\frac{1}{|f'(m)|} \text{ large}\right)$.

Bisection is nice in the sense that we know how things are going to go. In particular, if the interval initially is $[a,b]$ then after k iterations the interval size is

$$\frac{(b-a)}{2^k}$$

and so the error at step k, if we pick the midpoint as the root approximation, is

$$\frac{1}{2}\frac{(b-a)}{2^k}$$

So if you want an error in x^* equal to 10^{-8} this tells you how many iterations are required

$$10^{-8} = \frac{(b-a)}{2^{k+1}}$$

$$2^{k+1} = (b-a)10^8$$

$$(k+1)\ln 2 = \ln(b-a) + 8\ln 10$$

$$k+1 = \frac{\ln(b-a)}{\ln 2} + 8\frac{\ln 10}{\ln 2}$$

$$k = \frac{\ln(b-a)}{\ln 2} + 8\frac{\ln 10}{\ln 2} - 1$$

pros/cons

- Assuming an initial bracket is found, convergence to a root is guaranteed. (+)
- Requires only function evaluations to proceed (+)
- Convergence is only linear

$$\frac{|e_{k+1}|}{|e_k|} = \frac{\frac{1}{2}|e_k|}{|e_k|} = \frac{1}{2}$$

} convergence is slow but guaranteed...

Newton's Method ($\vec{F}(\vec{x}) = \vec{0}$)

~~[illegible]~~
This method is a type of fixed point iteration. $(x_{k+1} = g(x_k))$
It can be applied in 1D and higher dimensions

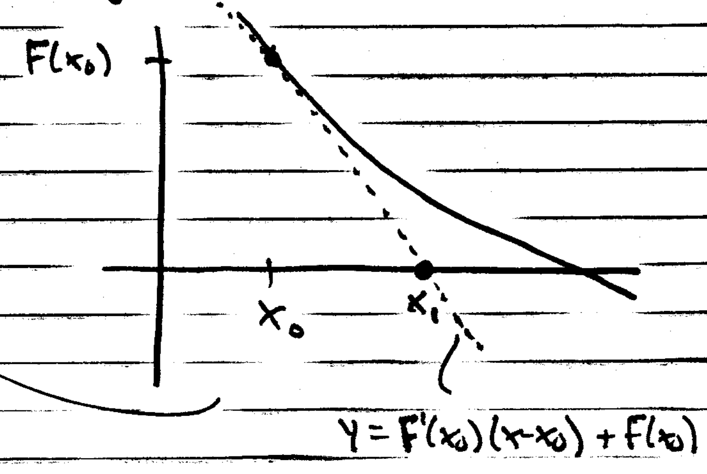
1D

$$\boxed{x_{k+1} = x_k - \frac{F(x_k)}{F'(x_k)} \qquad k = 0, 1, 2, \ldots}$$

graphically...

set $y=0$

$$x_1 = x_0 - \frac{F(x_0)}{F'(x_0)}$$



In terms of a general fixed point iteration scheme

$x_{k+1} = g(x_k)$ (i.e. $\underbrace{x = g(x)}_{x \text{ is a fixed pt. of the iteration.}} \Leftrightarrow f(x)=0$)

we have

$$g(x) = x - \frac{f(x)}{f'(x)}$$

In general, for $x_{k+1} = g(x_k)$ note

$$e_{k+1} = x_{k+1} - x^* = g(x_k) - g(x^*) \qquad x^* = g(x^*)$$

$$= g'(\theta_k)(x_k - x^*) \quad \text{by Mean Value Thm}$$

$$= g'(\theta_k)\, e_k \qquad \theta_k \in [x^*, x_k]$$

so $\dfrac{|e_{k+1}|}{|e_k|} = |g'(\theta_k)|$

So FPI converges if $|g'(\theta_k)| < 1$ for θ_k near x^*

What happens if $g'(x^*) = 0$? This is the case in Newton's method

$$g(x) = x - \frac{f(x)}{f'(x)}$$

$$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2}$$

$$g'(x^*) = 1 - \frac{[f'(x^*)]^2 - \cancelto{0}{f(x^*)} f''(x^*)}{[f'(x^*)]^2}$$

If $f'(x^*) \neq 0$ then $g'(x^*) = 0$.

So if x_k is sufficiently close to x^* note that

$$g(x_k) \approx g(x^*) + \cancel{g'(x^*)}^{\,0}(x_k - x^*) + \frac{1}{2} g''(x^*)(x_k - x^*)^2 + \ldots$$

So

$$g(x_k) - g(x^*) \approx \frac{1}{2} g''(x^*)(x_k - x^*)^2$$

$$x_{k+1} - x^* \approx \frac{1}{2} g''(x^*)(x_k - x^*)^2$$

So $\|e_{k+1}\| \approx \frac{1}{2}\|g''(x^*)\| \|e_k\|^2$

So

$$\lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|^2} = \frac{1}{2}\|g''(x^*)\|$$

$\Rightarrow$ quadratic convergence! (if $f'(x^*) \neq 0$).

$\Rightarrow$ Newton's method is fast if it converges to a simple root!

What if x^* is a degenerate root of multiplicity m.

That is, what if, for x near x^* we have

$$f(x) \approx \cancel{f(x^*)}^{0} + \cancel{f'(x^*)}^{0}(x-x^*) + \dots + \frac{1}{(m-1)!}\cancel{f^{(m-1)}(x^*)}^{0}(x-x^*)^{m-1}$$

$$+ \frac{1}{m!} f^{(m)}(x^*)(x-x^*)^m + \dots$$

So

$$f(x) = \frac{1}{m!} f^{(m)}(x^*)(x-x^*)^m + \dots \qquad f^{(m)}(x^*) \neq 0.$$

then

$$f'(x) = \frac{1}{m!} f^{(m)}(x^*) \cdot m (x-x^*)^{m-1} + \dots$$

So Newton's method now looks like

$$x_{k+1} = x_k - \frac{\frac{1}{m!} f^{(m)}(x^*)(x_k-x^*)^m}{\frac{1}{m!} f^{(m)}(x^*)\, m\, (x_k-x^*)^{m-1}}$$

$$= x_k - \frac{1}{m}(x_k-x^*)$$

So

$$x_{k+1} - x^* = x_k - x^* - \frac{1}{m}(x_k-x^*) = \left(1-\frac{1}{m}\right)(x_k-x^*)$$

$$|e_{k+1}| = \left(1-\frac{1}{m}\right)|e_k|$$

So $\displaystyle\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|} = 1 - \frac{1}{m}$ $\longleftarrow$ linear convergence with constant $C = 1 - \frac{1}{m}$

$\rightarrow$ This is not so fast convergence.

Pros/Cons (Newton's method)

- Quadratic convergence if the root is simple and if your approximate solution is close.
- Convergence (even slowly) is not guaranteed.
- Convergence may depend strongly on your initial guess.
- Need to be able to compute $f'(x)$.

Secant Method

(1D)

$$\boxed{x_{k+1} = x_k - f(x_k)\left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}\right)} \quad k = 1, 2, \ldots$$

Pros/Cons:

- does not require the functions derivative
- does require two previous iteration values (need to have two to start, need to store two values each iteration)
- convergence rate is superlinear

$$|e_{k+1}| = C|e_k|^{\frac{1+\sqrt{5}}{2}} \approx 1.6$$

Hybrid Methods

See Matlab fzero, fsolve — (higher Dim)

1D (bisection, secant, inverse quadratic interp.) Dekker

- combine "safety" of methods like bisection with "speed" of methods like Newton's method

Newton's Method for System of Equations

$$\vec{F}(\vec{x}) = \vec{0}$$

$$\boxed{\vec{x}_{k+1} = \vec{x}_k - \underbrace{\left[J_{\vec{F}}(\vec{x}_k)\right]^{-1}}_{\text{matrix}} \cdot \underbrace{\vec{F}(\vec{x}_k)}_{\text{vector}}} \qquad k=0,1,2,\ldots$$

In general, numerically it is not efficient to compute the inverse of the matrix directly. That is, we do not actually need the inverse matrix. We do need to solve the system

$$J_{\vec{F}}(\vec{x}_k)\underbrace{\left(\vec{x}_{k+1}-\vec{x}_k\right)}_{\equiv \vec{s}_{k+1}} = -\vec{F}(\vec{x}_k)$$

$$\text{“}\; A \cdot x = b \;\text{”}$$

So at each [redacted] iteration [redacted] we solve a linear system of equations

That is, at each time step we iteratively solve the nonlinear system of equations by [redacted] solving "many" linear systems — [redacted]

Convergence: → quadratic if Jacobian is nonsingular.

So now let's return to solving $\vec{y}' = \vec{f}(t, \vec{y})$ using backward Euler

$$\vec{F}(\vec{z}) = \vec{z} - \vec{y}_k - h\,\vec{f}(t_{k+1}, \vec{z})$$

where $\vec{F}(\vec{y}_{k+1}) = \vec{0}$.

So to get $\vec{y}_{k+1}$ ... use the iteration

$$z^{(j+1)} = z^{(j)} - J_F(\vec{z}^{(j)})^{-1} \cdot \vec{F}(\vec{z}^{(j)})$$

i.e.

$$\vec{z}^{(j+1)} = \vec{z}^{(j)} + \vec{s}^{(j)}$$

where $s^{(j)}$ is the solution to the linear system

$$J_F(\vec{z}^{(j)}) \cdot \vec{s}^{(j)} = -\vec{F}(\vec{z}^{(j)})$$

e.g. in Matlab...

$\vec{z}^{(1)} = \vec{z}_{guess}$ (maybe $\vec{z}_{guess} = \vec{y}_k$)

for $j = 1, 2, 3, \ldots, jmax$

(*) $\vec{s}^{(j)} = -J_F(\vec{z}^{(j)}) \backslash \vec{F}(\vec{z}^{(j)})$; — " \ " = Matlab Matrix Solve Command

$\vec{z}^{(j+1)} = \vec{z}^{(j)} + \vec{s}^{(j)}$

convergence criterion (maybe if $\|\vec{s}^{(j)}\| < tol.$)

end

$\vec{y}_{k+1} = \vec{z}^{(j+1)}$

The procedure (*) must be done at each time step.

Recall the system of equations describing 3 competing species

$$\begin{cases} \dfrac{dx}{dt} = ax - bxy \\ \dfrac{dy}{dt} = -cy + dxy - eyz \\ \dfrac{dz}{dt} = -fz + gyz \end{cases} \qquad a,b,c,d,e,f,g > 0$$

$$+\text{i.c.} \begin{cases} x(0) = x_0 \\ y(0) = y_0 \\ z(0) = z_0 \end{cases} \qquad \vec{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

or $$\frac{d\vec{y}}{dt} = \vec{f}(t,\vec{y}) = \begin{cases} ay_1 - by_1y_2 \\ -cy_2 + dy_1y_2 - ey_2y_3 \\ -fy_3 + gy_2y_3 \end{cases}$$

We've already studied this system of equations using forward Euler ...

$$\vec{y}_{k+1} = \vec{y}_k + h\vec{f}(t_k, \vec{y}_k) \qquad k = 0,1,2,\ldots$$

Let's now consider backward Euler

$$\vec{y}_{k+1} = \vec{y}_k + h\vec{f}(t_{k+1}, \vec{y}_{k+1}) \qquad k = 0,1,2,\ldots$$

That is, given an initial condition $\vec{y}_0 = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$

we find $\vec{y}_1$ by solving the <u>nonlinear system of equations</u>

$$\vec{y}_1 = \vec{y}_0 + h\vec{f}(t_1, \vec{y}_1)$$

That is, find the solution of

$$\vec{F}(\underset{\substack{\uparrow \\ \text{unknown} \\ \text{variable} \\ \text{(vector)}}}{\vec{y}_1}; \underbrace{\vec{y}_0, t_1, h}_{\text{parameters}}) = \vec{y}_1 - \vec{y}_0 - h\vec{f}(t_1, \vec{y}_1) = 0$$

In general to find $\vec{y}_{K+1}$ we solve

$$\boxed{F(\vec{y}_{K+1}) \equiv \vec{y}_{K+1} - \vec{y}_K - h\vec{f}(t_{K+1}, \vec{y}_{K+1}) = 0}$$

One way to do this is to use Newton's method.

$$\vec{F}(\vec{y}_{K+1}) = \vec{0}$$

is solved by iterating

$$\boxed{\vec{y}_{K+1}^{(j+1)} = \vec{y}_{K+1}^{(j)} - \underline{\underline{J}}^{-1} \vec{F}(\vec{y}_{K+1}^{(j)})} \quad j = 1, 2, 3, \ldots$$

where $\vec{y}_{K+1}^{(1)}$ is the initial guess for $\vec{y}_{K+1}$

and the idea of Newton's method is that

$$\vec{y}_{K+1}^{(j)} \longrightarrow \vec{y}_{K+1} \quad \text{as } j \to \infty \quad (\text{"}\infty\text{" usually "small"})$$

~~Application~~ where $\underline{\underline{J}} \equiv \dfrac{\partial \vec{F}}{\partial \vec{y}}$ = Jacobian Matrix

$$J_{ij} = \frac{\partial F_i}{\partial y_j}$$

Now let's apply this method to our particular system.

$$\vec{F}(\vec{y}) = \vec{y} - \vec{y}_{prev.} - h\,\vec{f}(\vec{y}) = 0$$

where $\vec{f}(\vec{y}) = \begin{cases} a y_1 - b y_1 y_2 \\ -c y_2 + d y_1 y_2 - e y_2 y_3 \\ -f y_3 + g y_2 y_3 \end{cases}$

$\underline{\underline{J}} = \underline{\underline{I}} - h \underline{\underline{J}}_f$ where $\underline{\underline{J}}_f = \begin{bmatrix} a - b y_2 & -b y_1 & 0 \\ d y_2 & -c + d y_1 - e y_3 & -e y_2 \\ 0 & g y_3 & -f + g y_2 \end{bmatrix}$

So the solution $\vec{y}$ (ie. representing $\vec{y}_{k+1}$)

$$\vec{y}^{(j+1)} = \vec{y}^{(j)} - \underline{\underline{J}}^{-1} * \vec{F}(\vec{y}^{(j)})$$

See Matlab code

$\boxed{\begin{array}{l}\texttt{back\_euler\_system.m} \\ \texttt{F\_J\_back\_euler\_system.m}\end{array}}$

Revisit the two numerical examples we examined previously using forward Euler (see euler_system.m and feuler_system.m)

Num Exp. #1

($a = b = d = e = 1$, $c = 2$, $f = 100$, $g = 0.1$) $x_0 = 0.5$, $y_0 = 1$, $z_0 = 2$

			FE	BE
run 1:	$t_F = 5$, $nk = 1000$	($h = \frac{5}{1000} = 0.005$)	✓	✓
run2	$t_F = 10$, $nk = 1000$	$h = 0.01$	Some observable differences	
run3	$t_F = 20$, $nk = 1000$	$h = 0.02$	BAD!	maybe ok ≈
run4	$t_F = 40$, $nk = 1000$	$h = 0.04$	Blow up!	at least z still behaving as expected

Num. Exp. #2

			FE	BE
run 1	$t_F = 0.5$, $nk = 100$	$h = 0.005$	✓	✓
run 2	$t_F = 1.0$, $nk = 100$	$h = 0.01$	✓	✓
run 3	$t_F = 1.5$, $nk = 100$	$h = 0.015$	osc. in z	✓
run4	$t_F = 2.0$, $nk = 100$	$h = 0.02$	BAD!	✓
			⋮	⋮

Other comments on solving systems $\vec{F}(\vec{z}) \in \mathbb{R}^n$

- cost of each Newton step
  - computing $\vec{J}_F(\vec{z}^{(s)})$ $O(n^2)$
  - solving linear system $O(n^3)$

Quasi-Newton
- reuse old Jacobian Matrix (update occasionally)
- solve the linear system to rough approximation during early iterations (just to get the next guess - don't need 16 digits for a guess)

see also "fsolve" in Matlab