5.5.4 Secant Method

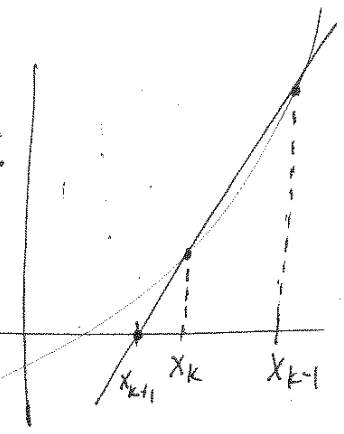
Drawback of Newton's Method: need an expression for $f'(x)$
- may not be available.
- may be expensive

use finite difference approximation instead.
use successive iterates — no need for extra function evaluations

$$f'(x_k) \cong \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

$r \cong 1.618$ convergence rate.

$$x_{k+1} = x_k - \frac{f(x_k)}{\left[\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}\right]}$$



- need two starting guesses.
- a few more iterations are needed than Newton's method
+ but ½ function evaluations per iteration than Newton's method.

→ often better than Newton's method for total cost.

~~5.5.5~~ ~~5.5.6~~

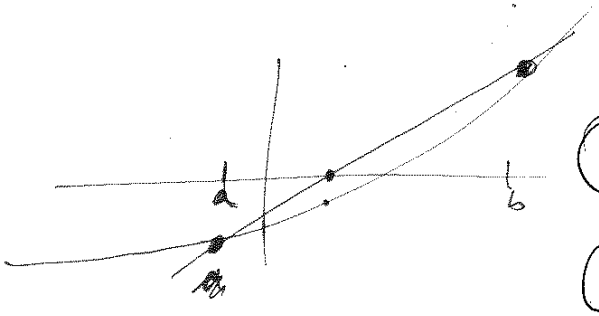
§5.5.7 Safeguarded Methods

- Newton's method, & secant method unsafe
  —need to start close to solution
- Bisection safe slow

Hybrid

- if fast method gives iterate outside bracket
  do one iteration w/ safe method.
- go back to fast method.

E.g., bisection for safety
secant for speed.



only one evaluation
$f(x_k)$
per iteration

let $m = x_{k+1}$
for updating bracket

(1) bracketing interval $[a,b]$
$x_0 = a$, $x_1 = b$.

(2) secant method x_{k+1}

(3) if $x_{k+1} \in [a,b]$, $m = x_{k+1}$
update bracket ~~if bisection~~

(4) if $x_{k+1} \notin [a,b]$, or $f(b) - f(a)$ too small
to apply secant method, use bisection

~~5.5.8~~ Zeros of Polynomials

- methods above + deflate $\frac{p(x)}{(x-x_1)}$
- companion matrix MATLAB
  reliable/less efficient
- all roots
  - Laguerre, Bairstow
  - Jenkins, Traub

§ 5.6. Systems of Nonlinear Equations — Lecture 9

- wide range of behaviors is possible
  - theoretical analysis more complex
- not simple to bracket solution
- computationally expensive.

In multi-D, <u>Jacobian</u> plays role of f'

$$J(x) = \frac{\partial F}{\partial x}(x) \qquad J_{ij} = \frac{\partial F_i}{\partial x_j}$$

$$F(\vec{x}) = \begin{pmatrix} F_1(\vec{x}) \\ F_2(\vec{x}) \\ \vdots \\ F_m(\vec{x}) \end{pmatrix} = \begin{pmatrix} F_1(x_1, x_2, \dots, x_n) \\ F_2(x_1, x_2, \dots, x_n) \\ \vdots \\ F_m(x_1, x_2, \dots, x_n) \end{pmatrix}$$

<u>e.g.)</u> $F_1(x_1, x_2) = x_1^2 + \sin x_2 + 5$

$F_2(x_1, x_2) = x_1 + x_2^3$

$\frac{\partial F_1}{\partial x_1} = 2x_1$, $\frac{\partial F_1}{\partial x_2} = \cos x_2$

$\frac{\partial F_2}{\partial x_1} = 1$, $\frac{\partial F_2}{\partial x_2} = 3x_2^2$

$$\Rightarrow \boxed{J(x_1, x_2) = \begin{pmatrix} 2x_1 & \cos x_2 \\ 1 & 3x_2^2 \end{pmatrix}}$$

<u>fixed pt. iter</u>

local conv. cond $|g'(x^*)| < 1$

analogous cond'n $\rho(J(x^*)) < 1$ spectral radius.

(don't necessarily need to compute eigenvalues)

$\rho(A) \le \|A\|$ $\forall$ vector-norm induced matrix norms

$J(x^*) = 0 \Rightarrow$ <u>quadratic conv.</u>

§ 5.6.2 Newton's Method

truncated Taylor series

$$f(x+s) \approx f(x) + J(x)s \overset{\text{set}}{=} 0.$$

Solve: $J(x)s = -f(x)$ for s.

$$x_{k+1} = x_k + s_k$$

Ex.

$$F(x) = \begin{pmatrix} x_1 + 2x_2 - 2 \\ x_1^2 + 4x_2^2 - 4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$x_0 = (1,2)^T$$

Notes: Must repeatedly solve linear systems!

- computing $J(x_k)$ may be expensive
  dense $\rightsquigarrow$ n^2 function evaluations
- Solving $J(x_k)s = -f(x_k)$ by LU $O(n^3)$ operations

"Quasi – Newton Methods"

cut some corners: e.g.,
- don't reevaluate J each iteration
- don't solve eq. exactly $Js = f$

Secant-like Methods: Secant updating Methods.

Broyden's Method
- build up J incrementally
- update factorizations of J to save

$$B_{k+1} = B_k\left(I - \frac{s_k s_k^T}{s_k^T s_k}\right) + \frac{(f(x_{k+1}) - f(x_k))}{s_k^T s_k} s_k^T$$

---

damped Newton method $\alpha_k \le 1$
modify step $x_{k+1} = x_k + \alpha_k s_k$

trust region
modify step + direction

Multi-D Fixed Pt. Iteration

$$\vec{g} : \mathbb{R}^n \to \mathbb{R}^n$$

$$\vec{x} = \vec{g}(\vec{x})$$

$$\vec{x}_{k+1} = \vec{g}(\vec{x}_k)$$

convergence if started close to solution and

$$\rho(J(x^*)) < 1$$

- the smaller ρ, the faster the convergence

$J(x^*) = 0 \Rightarrow$ convergence rate at least quadratic

Newton's Method

x_0 = initial guess

for $k = 0, 1, 2, \ldots$

solve $J(x_k)\vec{s}_k = -\vec{f}(\vec{x}_k)$ Newton step

$\vec{x}_{k+1} = \vec{x}_k + \vec{s}_k$ update solution

end

Secant-updating Methods

Broyden's Method.

e.g., true J, finite diff, or I

x_0 = initial guess

B_0 = initial Jacobian approximation

for $k = 0, 1, 2, \ldots$

Solve $B_k \vec{s}_k = -\vec{f}(\vec{x}_k)$ for $\vec{s}_k$

$\vec{x}_{k+1} = \vec{x}_k + \vec{s}_k$

$\vec{y}_k = \vec{f}(\vec{x}_{k+1}) - \vec{f}(\vec{x}_k)$

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) s_k^T}{s_k^T s_k} \quad (*)$$

end.

(*) update eq. for Jacobian

least change to B_k s.t. <u>Secant equation</u> is satisfied

$$B_{k+1}(x_{k+1} - x_k) = f(x_{k+1}) - f(x_k)$$

$$B_{k+1} s_k = f(x_{k+1}) - f(x_k)$$

$$\boxed{B_{k+1} s_k = y_k}$$

$$B \approx \frac{\partial f}{\partial x} \qquad B\,\delta x \approx \delta f$$

$$B_k - \frac{B_k s_k s_k^T}{s_k^T s_k} + \frac{y_k s_k^T}{s_k^T s_k}$$

orthogonal projection

- update a factorization of B_k instead of updating B_k & refactoring
  - ⇝ avoid $O(n^3)$ operations need to factor
  - ⇝ $O(n^2)$ operations per iteration instead