Chapter 2 – Newton w. Gaussian Elimination

The most basic / str. forward approach to solving for Newton direction is to use 'direct solver' → LU or Cholesky & factorization

⋮

while $\|F(x)\| > \tau$ do

Compute $F'(x)$;

Compute $LU = F'$ (or $LL^T = F'$)

if fact. fails terminate with failure

else

Solve $LUd = -F(x)$

$$\begin{cases} L\tilde{d} = -F(x) & \text{(forward subst.)} \\ Ud = \tilde{d} & \text{(backw. subst.)} \end{cases}$$

end if

compute step $s = dd$ using (polynom.) line search

Compute $F(x)$

end while

→ Possible and efficient to compute & and (typically) store Jacobian

→ Efficient to factorize & and store Jacobian

Note that if F' is sparse we may be able to store F' but not its factors (too dense). also often too expensive in time

→ No problems with convergence linear solver.

In some cases the last issue is misleading. If F' is very ill-conditioned the answer is questionable even if direct solver (almost) always produces one.

We will discuss <u>accuracy</u> and <u>sensitivity</u> issues later.

Direct solver will do (typically) partial or complete pivoting.

$\rightarrow PA = LU$ (or $PAQ = LU$)

$PAd = Pb \rightarrow LUd = Pb$ $\begin{cases} L\tilde{d} = Pb \\ Ud = \tilde{d} \end{cases}$

In software packages the permutation P (Q) is usually taken care of transparently

In matlab just use $[L, U] = lu(A)$

$\tilde{d} = L \backslash b$

$d = U \backslash \tilde{d}$

L only lower triangular up to some reordering (P)

A good package should provide information on the "quality" of the LU decomposition and the solution d

Possibly through additional function calls

Although in principle cost of storing F' and solving linear system are $O(N^2)$ and $O(N^3)$ in practice often much cheaper.

Many large problems arise from nonlinear PDEs and integral equations →

Often F' very sparse →

store only nonzeroes: $O(N)$

use sparse direct solver: $\sim O(N^2)$ or less
(can be more) ↑ often

Matlab → store F' as sparse matrix then matlab uses sparse direct solver

(UMFPACK by Tim Davis, Univ. of Florida)

Finite Difference Jacobian

In some cases it is hard or impossible to get the analytic Jacobian. In such cases a finite difference approximation can be used.

Typical examples involve F where F itself calls additional complicated functions, like an ODE solver, PDE solver, etc.

j-th column of Jacobian

$$(\nabla_h F)(x)_j = \begin{cases} \dfrac{F(x + h\sigma_j e_j) - F(x)}{\sigma_j h}, & (x)_j \neq 0 \\ \dfrac{F(x + he_j) - F(x)}{h}, & (x)_j = 0 \end{cases}$$

where $e_j = (0, \dots, 0, \underset{\text{j-th pos.}}{1}, 0, \dots, 0)^T$ is j-th Cartesian (canonical) basis vector.

Why scaling σ_j and how to pick h?

Consider $f : \mathbb{R} \to \mathbb{R}$ and we approx.

$$f'(x) = (f(x+h) - f(x))/h$$

What choice of h gives best approx.?

Assume computing $f(z)$ gives error $\approx \varepsilon$ (of some order of accuracy)

IEEE arithmetic requires all basic operations (including builtin functions) to have relative error at machine precision

x, y are machine repr. numbers

$$fl(x+y) \equiv [x+y] = (x+y)(1+\varepsilon_{mach})$$

longer computations may lead to accumulation and possible amplification (instability) of errors.

$$[f(x+h) - f(x)] = f(x+h) - f(x) + 2\varepsilon$$

↳ worst case, errors accumulate

$$= f(x) + hf'(x) + \frac{1}{2}h^2 f''(\zeta) - f(x) + 2\varepsilon$$

$$= hf'(x) + \frac{1}{2}h^2 f''(\zeta) + 2\varepsilon$$

$$\left[\frac{f(x+h) - f(x)}{h}\right] = f'(x) + \frac{1}{2}hf''(\zeta) + \frac{2\varepsilon}{h} + \varepsilon$$

bound $f''(\zeta)$ by $M = \max_{\zeta \in (x, x+h)} |f''(\zeta)|$

minimize error $\frac{Mh}{2} + \frac{2\varepsilon}{h} + \varepsilon \equiv e(h)$

$$e'(h) = \frac{M}{2} - \frac{2\varepsilon}{h^2} = 0 \quad h^2 = \frac{4\varepsilon}{M} \rightarrow h = 2\sqrt{\frac{\varepsilon}{M}}$$

So, we should take h at roughly $O(\sqrt{\varepsilon})$

Taking h smaller blows up the $\frac{2\varepsilon}{h}$ term.

However, if h too small relative to $(x)_j$ nothing (or very little) happens

So, a scale increment $\sigma_j = \max(|(x)_j|, 1)\, \mathrm{sgn}((x)_j)$

where $\mathrm{sgn}(z) = \begin{cases} z/|z| & \text{if } z \neq 0 \\ 1 & \text{if } z = 0 \end{cases}$

Since we get a column per function eval →
N function evals for FD Jacobian

F' sparse → exploit sparsity to make FD computation cheaper. Main idea is that if nonzero pattern in $F'_{:,j}$ and $F'_{:,k}$ do not overlap → compute simultaneously

$$\frac{F(x + h\sigma_j e_j + h\sigma_k e_k) - F(x)}{h\sigma}$$

$$\simeq \left(\cancel{F(x)} + (h\sigma)\cdot F'(x)(e_j + e_k) - \cancel{F(x)}\right)/h\sigma$$

$$= F'(x)\,e_j + F'(x)\,e_k \quad \text{(single vector)}$$

$F'e_j$ nonzero coeffs $F'_{i_1,j}, F'_{i_2,j}, \dots F'_{i_\ell,j}$

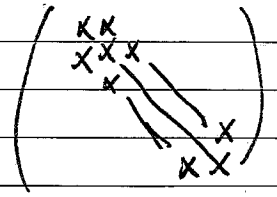
$F'e_k$ " " $F'_{l_1,k}, F'_{l_2,k_1} \dots$

→ distinct so we can pick them out of vector

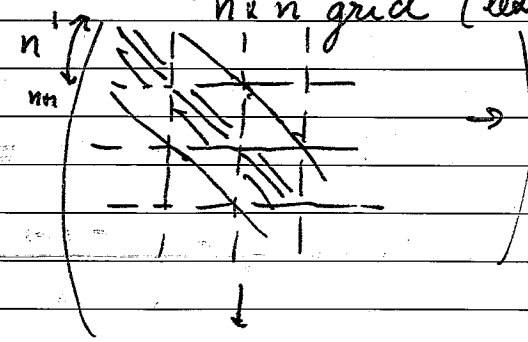
Simplest case F' banded with band b →

$$J_{ij} = 0 \quad \text{if } |i-j| > b$$

$b=1$



2D fin. diff discr. 5pt stencil
$n \times n$ grid (lexicogr. ord)



→ bandwidth $b = n$

bandwidth b:

row i nonzeros col $j = i-b \cdots j = i+b$

row $i+2b+1$ nonzeros $j = i+b+1 \cdots j = i+3b+1$

$b = 1$

$$J = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & (2) & (3) & 4 & (5) & (6) & 7 \end{array} \\ \begin{bmatrix} x & x & & & & & \\ x & x & x & x & & & \\ & x & x & x & & & \\ & & x & x & x & & \\ & & & x & x & x & \\ & & & & x & x & x \\ & & & & & x & x \end{bmatrix} \end{array}$$

$$J(e_1+e_4+e_7) = (J_{11}\ J_{21}\ 0\ 0\ 0\ 0\ 0)^T +$$
$$(0\ 0\ J_{34}\ J_{44}\ J_{54}\ 0\ 0)^T +$$
$$(0\ 0\ 0\ 0\ 0\ J_{67}\ J_{77})^T$$
$$= (J_{11}\ J_{21}\ J_{34}\ J_{44}\ J_{54}\ J_{67}\ J_{77})^T$$

pick coeff.s out of vector and store in (appr.) Jac.

Book treats upper- and lower bandwidth seperately (more general)

See references on lecture notes page for ~~gener~~ strategies for general sparse Jacobians.

Cost of LU fact. (with pivoting restricted to band) is only $O(b^2 N)$

Based on bandwidth b →

$p_1 = e_1 + e_{2b+2} + e_{4b+3} + \cdots$

$p_2 = e_2 + e_{2b+3} + e_{4b+4} + \cdots$

$p_3 = e_3 + e_{2b+4} + e_{4b+5} + \cdots$

$\vdots$

$p_{2b+1} = \cdots$

$$(D_h^1 F)(x) = \begin{cases} \dfrac{F(x + h\|x\| p_1) - F(x)}{h\|x\|} & x \neq 0 \\ \dfrac{F(x + h p_1) - F(x)}{h} & x = 0 \end{cases}$$

$(D_h^2 F)(x) =$ (same for p_2) etc

$(D_h^1 F)(x)$ coeffs $1 .. 1+b \rightarrow$ col. 1
" $b+2 .. 3b+2 \rightarrow$ col. $2b+2$
" $3b+3 .. 5b+3 \rightarrow$ col. $4b+3$

$\vdots$

$(D_h^2 F)(x)$ coeffs $1 .. b+2 \rightarrow$ col. 2
" $b+3 .. 3b+3 \rightarrow$ col. $2b+3$
" $3b+4 .. 5b+4 \rightarrow$ col. $4b+4$

etc

Only need $2b+1$ function eval.s rather than N

Chord Method → compute Jacobian once,
factorize once

Only $F'(x_0)$ and $LU = F'(x_0)$

more cheap iterations but slower convergence
(or no convergence)

go from $O(N^3)$ work per nonlinear step to
$O(N^2)$ (form. + backw subst.)

↳ or less for sparse Jacobian

Shamanskii method → in between Newton
an Chord : update every m nonlinear it.s
more sophisticated versions also check progress
(convergence) and effectiveness of line search
either too poor → compute new Jacobian
(in nsold.m)