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

\[
\text{while } \| F(x) \| > \epsilon \text{ do }
\]

Compute \( F'(x) \);

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

if fact. fails terminate with failure
else

Solve \( LUd = -F(x) \)

\[
\begin{align*}
\text{Forward subst.} & : Ld = -F(x) \\
\text{Backw. subst.} & : Ud = d
\end{align*}
\]

end if

compute step \( s = dd \) using (polynom.)

line search

Compute \( F(x) \)

end while

→ Possible and efficient to compute \( \xi \) and (typically) store Jacobian

→ Efficient to factorize \( F' \) 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 accuracy and sensitivity issues later.

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

$$PA = LU \quad \text{(or } PAQ = LU)$$

$$PAd = Pb \quad \Rightarrow \quad LUd = Pb$$

$$\begin{align*}
Ld &= Pb \\
 Ud &= \tilde{d}
\end{align*}$$

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

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

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

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

L only lower triangular up to some reordering

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 non-zeroes: $O(N)$

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

\[
(\nabla_h F)(x)_j = \begin{cases} 
\frac{F(x + h e_j) - F(x)}{h}, & (x)_j \neq 0 \\
\frac{F(x + h e_j) - F(x)}{h}, & (x)_j = 0 
\end{cases}
\]

$j$-th pos. $\eta_j = (0, \ldots, 0, 1, 0, \ldots, 0)$ is $j$-th Cartesian (canonical) basis vector.

Why scaling $\eta_j$ and how to pick $h$?

Consider $F : \mathbb{R} \to \mathbb{R}$ and use approx.

\[
F'(x) = \frac{F(x+h) - F(x)}{h}
\]

What choice of $h$ gives best approx. $\eta$?

Assume computing $F'(z)$ gives error $\approx \varepsilon$

(of some order of accuracy)
IEEE arithmetic requires all basic operations (including built-in functions) to have relative error at machine precision.

\[ \text{x, y are machine repr. numbers} \]
\[ \text{round(} x+y \text{) } = (x+y)(1 + \varepsilon_{\text{mach}}) \]

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

\[ \frac{\text{error}}{\text{worst case, errors}} \]
\[ = \frac{f(x) + h f'(x) + \frac{1}{2} h^2 f''(x) - f(x) + \varepsilon}{h} \]
\[ = \frac{h f'(x) + \frac{1}{2} h^2 f''(x) + 2 \varepsilon}{h} \]

Bound \( f''(x) \) by \( M = \max_{j \in (x,x+h)} |f''(x_j)| \)

Minimize error \( \frac{M h}{2} + \frac{2 \varepsilon}{h} + \varepsilon \)

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

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

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

So, scale increment \( \alpha_j = \max(1,|x_j|,1) \) \( sgn(|x|) \)

\[ \text{where } sgn(z) = \begin{cases} \frac{z}{|z|} & \text{if } z \neq 0 \\ 1 & \text{if } z = 0 \end{cases} \]
Since we get a column per function eval $\rightarrow$ 
\[ N \text{ function evals for FO Jacobian} \]

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

\[
\frac{F(x + h\sigma e_j + h\sigma e_k) - F(x)}{h\sigma} = \left( F(x) + (h\sigma \cdot F'(x)(e_j + e_k) - F'(x)) \right) / h\sigma
\]

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

$F'_{e_j}$ nonzero coeffs $F'_{i,j}$, $F'_{i+1,j}$, $F'_{i,j+1}$

$F'_{e_k}$ $n \times n$ $F'_{e_k,1}$, $F'_{e_k,2}$, $\ldots$

$\rightarrow$ distinct so we can pick them out of vector

Simplest case $F'$ banded with band $b \rightarrow$

\[ g_{i,j} = 0 \quad \text{if} \quad |i-j| > b \]

\[ b=1 \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} 2D finite difference stencil \]

\[ n \times n \text{ grid (left to right)} \]

\[ \text{bandwidth} b = n \]
bandwidth b:

row i nonzero cols j = i-b ... j = i+b

row d+b+1 nonzeroes j = i+b+1 ... j = i+2b+1

\[ b = 1 \]

\[ \begin{array}{cccc}
1 & x & x & \\
2 & x & x & x \\
3 & x & x & x \\
4 & x & x & x \\
5 & x & x & x \\
6 & x & x & \\
\end{array} \]

\[ y = \left( \begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16 \\
17 & 18 & 19 & 20 \\
\end{array} \right) \]

\[ y^T (e_1 + e_4 + e_7) = \left( \begin{array}{cc}
y_{11} & y_{21} \\
y_{44} & y_{54} \\
y_{67} & y_{77} \\
\end{array} \right) + \\
\left( \begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array} \right) + \\
\left( \begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{array} \right) \]

pick coeffs out of vector and store in (approx) jac.

book treats upper- and lower bandwidth separately (more general)

see references on lecture notes page for general strategies for general sparse jacobians.

cost of LU fact. (with pivoting restricted to band)

is only \( O(b^2 N) \)
Based on bandwidth \( b \rightarrow \)

\[ p_1 = 1 + e_{2b+2} + e_{4b+3} + \cdots \]

\[ p_2 = 1 + e_{2b+3} + e_{4b+4} + \cdots \]

\[ p_3 = 1 + e_{2b+4} + e_{4b+5} + \cdots \]

\[ \vdots \]

\[ p_{2b+1} = \cdots \]

\[
(D^-_h F)(x) = \begin{cases} 
\frac{F(x + h\|x\|p_n) - F(x)}{h \|x\|} & x \neq 0 \\
\frac{F(x + hp_n) - F(x)}{h} & x = 0
\end{cases}
\]

\[
(D^2_h F)(x) = (\text{same for } p_3) \quad \text{etc.}
\]

\[
(D^-_h F)(x) \quad \text{coeffs} 1 \ldots 1+b \rightarrow \text{col. 1} \\
\quad b+2 \ldots 3b+2 \rightarrow \text{col. } 2b+2 \\
\quad 3b+3 \ldots 5b+3 \rightarrow \text{col. } 4b+3 \\
\quad \vdots \\
\]

\[
(D^2_h F)(x) \quad \text{coeffs} 1 \ldots b+2 \rightarrow \text{col. 2} \\
\quad b+3 \ldots 3b+3 \rightarrow \text{col. } 2b+3 \\
\quad 3b+4 \ldots 5b+4 \rightarrow \text{col. } 4b+4 \\
\quad \text{etc.}
\]

Only need \( 2b+1 \) function evals 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
$O(N^2)$ (from + backw subst.)
⇒ or less for sparse Jacobian

Shamanshii method → in between Newton
and 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)