Chapter 6. Practical Newton Methods

Recall: $\nabla^2 f(x_k)\, p_k^N = -\nabla f(x_k)$ ~ (*)

We want:

* robust and efficient in all cases
* p_k^N to be a descent direction:
  - → will be true if $\nabla^2 f(x_k)$ is positive definite.
  - → else, ...

Consider line-search and trust-region implementations using Newton + CG:

* terminate at negative curvature, or
* modify $\nabla^2 f(x_k)$ to be pos. def.

Also, use Hessian sparsity to get efficient methods.

NB: Some of the most reliable and powerful methods for both small or large are based on Newton methods.

6.1 Inexact Newton Steps

Start with (*):

$$\nabla^2 f(x_k)\, P_k^N = -\nabla f(x_k)$$

Then, the residual error is:

$$r_k = \nabla^2 f(x_k)\, P_k + \nabla f(x_k) \quad \text{(**)}$$

for some $P_k \approx P_k^N$.

The problem here is that $cf(\cdot)$ for some constant c, will also scale the residual:

<u>New</u> $r_k' = c\left(\nabla^2 f(x_k)\, P_k + \nabla f(x_k)\right)$

$\therefore$ Eventhough both $f(\cdot)$ and $cf(\cdot)$ have the same stationary points, if we choose to terminate on $|r_k|$, we will stop at different locations for $f(\cdot)$ and $cf(\cdot)$ since $r_k \neq r_k'$.

Instead, we stop for:

$$\|r_k\| \leq \eta_k \|\nabla f(x_k)\|$$

for some sequence, the forcing sequence

$$\eta_1, \eta_2, \ldots$$

and $0 < \eta_k < 1$.

We then have convergence according to:

Thm 6.1 Assume:

* $\nabla f(x)$ is contsly diff'ble in a neighborhood of the minimizer x^*.
* $\nabla^2 f(x)$ is positive definite.

Then: $x_{k+1} = x_k + p_k$, and if each p_k is such that:

$$\|r_k\| \leq \eta_k \|\nabla f(x_k)\|, \quad \eta_k < 1,$$

and if the starting point is sufficiently close to x^*, we have convergence.

We will have:

Linear convergence if:

$$\limsup_{k\to\infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|} < 1,$$

Quadratic Convergence if:

$$\limsup_{k\to\infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|^2} = c < \infty, \text{ some } c$$

Superlinear Convergence if:

$$\limsup_{k\to\infty} \frac{\|\nabla f(x_{k+1})\|}{\|\nabla f(x_k)\|} = 0$$

To get the desired convergence rate, simply set the termination criterion as suggested by theorem 6.2:

Thm 6.2 Assume the setup in thm 6.1. Then:

(I) $\eta_k \to 0 \implies$ superlinear convergence

(II) $\eta_k = O(\|\nabla f(x_k)\|) \implies$ quadratic convergence.

The book recommends:

(I) Superlinear convergence:

Use $\eta_k = \min\left(0.5, \sqrt{\|\nabla f(x_k)\|}\right)$.

(II) Quadratic Convergence:

Use $\eta_k = \min\left(0.5, \|\nabla f(x_k)\|\right)$.

Make sure you can show how (I) + (II) yield the promised convergence...

Line Search for Large Problems

Algorithm 6.1 (Line-search Newton-CG)

Given x_0

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

$\left\{\begin{array}{l}\text{Solve } \nabla^2 f(x_k)\, p = -\nabla f_k \\ \text{using CG, } x^{(0)} = 0. \text{ (see below)}\end{array}\right.$

$\left\{\begin{array}{l}\text{Compute } \alpha_k \text{ using line-search} \\ \text{that satisfies Wolfe, Goldstein, or} \\ \text{Armijo conditions}\end{array}\right.$

$x_{k+1} = x_k + \alpha_k p_k$

end

Modified CG

$x_0 = 0, \quad A = (\nabla^2 f_k), \quad b = -\nabla f_k.$

$r_0 = -b, \quad p_0 = -r_0, \quad k = 0.$

while $\|r_k\| \le \min\left(0.5, \sqrt{\|\nabla f_k\|}\right) \|\nabla f(x_k)\|$

$\quad \text{den} = p_k^T A p_k$

$\left\{\begin{array}{l}\textbf{if } (\text{den} \le 0) \\ \quad \left\{\begin{array}{l}\textbf{if } (k > 0) \text{ or } (\text{den "small"}) \ \underline{\text{stop}} \\ \underline{\text{else}} \ \ \alpha_0 = (r_k^T r_k)/\text{den}, \ x_1 = -\alpha_0 r_0, \ \underline{\text{Stop}}\end{array}\right. \\ \textbf{end}\end{array}\right.$

$$\alpha_k = r_k^T r_k \,/\, \text{den}$$

$$x_{k+1} = x_k + \alpha_k p_k$$

$$r_{k+1} = r_k + \alpha_k A p_k$$

note that this could have been pre-computed with den:

$d = A p_k$

$\text{den} = p_k^T d$

$$\beta_{k+1} = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

$$p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$$

$k = k+1$

<u>end</u>.

(* returns with p_k for the next step *).

<u>Algorithm 6.2 (Line Search with Preconditioning)</u>

Given x_0

<u>for</u> $k = 0, 1, 2, \ldots,$ max_iterations

{Modify B_k to $B_k + E_k$ (discussed next)

{<u>Solve</u> $B_k p_k = -\nabla f(x_k)$ for p_k

{Compute α_k using line-search conditions.

$x_{k+1} = x_k + \alpha_k p_k$

<u>end</u>

Notes:

* Section 6.3 shows how to compute appropriate E_k.
* thm 6.3 says that if:

  (I) $\{x \in D : f(x) \leq f(x_0)\}$ compact (closed + bdd set that contains its boundary)

  (II) $\text{cond}(B_k) = \|B_k\| \, \|B_k^{-1}\| \leq C < \infty$

then $\lim_{k \to \infty} \nabla f(x_k) = 0$.

Proof: Easy!

* $B_k p_k = -\nabla f_k$ can be solved using Gaussian elimination or any standard method since B_k is modified to be positive definite. CG or Gaussian elimination will work.

6.3 Hessian Modifications.

We consider:

* $\nabla^2 f(x_k) + \lambda I$, already discussed,
* Controlling selected eigenvalues:

Start from:

$\nabla^2 f(x_k) = Q \Lambda Q^T$ as before.

Add $Q \Lambda' Q^T$ to have:

$$Q\Lambda Q^T + Q\Lambda' Q^T = Q(\Lambda Q^T + \Lambda' Q^T)$$
$$= Q(\underbrace{\Lambda + \Lambda'}) Q^T$$

The new eigenvalue matrix satisfies:

$$\Lambda + \Lambda' = \mathrm{diag}(\lambda_1 + \tau_1, \lambda_2 + \tau_2, \ldots, \lambda_n + \tau_n)$$

where:

$$\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n),$$
$$\Lambda' = \mathrm{diag}(\tau_1, \tau_2, \ldots, \tau_n).$$

Basic idea:

if original eigenvalues are sufficiently positive:

$\lambda_i > \varepsilon \geq 0$, some ε, then

set $\tau_i = 0$ (don't change them),

else

Either:

(I) Set $\lambda_i + \tau_i = 0 \Rightarrow \lambda_i' = -\tau_i$,

(II) Flip the sign: $\lambda_i + \tau_i = |\lambda_i|$, or

$\Rightarrow \lambda_i' = |\lambda_i| + \tau_i$

(III) Make them "sufficiently positive":

Set $\lambda_i + \tau_i = \varepsilon \Rightarrow \tau_i = \varepsilon - \lambda_i$.

The last option is optimal in the sense that the change ΔA to guarantee that we have "sufficient positive definiteness" will have $\|\Delta A\|_F$ minimum for III,

where

$$\|\Delta A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} \Delta A_{ij}^2} \,.$$

Using the same notation, for $\Delta A = \tau I$, set $\tau = \max(0, \delta - \lambda_{min}(A))$ with new matrix $A + \tau I$.

The previous methods: I, II, III are optimal, but they can be very slow.

The textbook describes a fast algorithm in Algorithm 6.5, for computing:

$$PAP^T + E = LDL^T \sim (*)$$

where: L is lower-triangular, D is diagonal, and E is used to make $PAP^T + E$ positive definite.

The book does not explain how to use the factorization in (*) to solve $Ax=b$, which is what we need!

We have:

$$PAP^T = LDL^T \Rightarrow PA = (LDL^T)P^{-T}$$

Thus: $Ax = b \Rightarrow PAx = Pb$

$$\Rightarrow (L\underbrace{DL^T)P^{-T}x}_{\text{set as } z_1 = DL^TP^{-T}x} = Pb$$

We have:

<u>Step 1</u>: Solve $Lz_1 = Pb$ for z_1.

<u>step 2</u>: Solve $Dz_2 = z_1$ for z_2.

<u>step 3</u>: Solve $L^Tz_3 = z_2$ for z_3.

<u>step 4</u>: Solve $P^{-T}x = z_3$ for x.

In other words:

$$z_1 = DL^TP^{-T}x,$$
$$z_2 = L^TP^{-T}x, \text{ and}$$
$$z_3 = P^{-T}x.$$

Another view:

$$L \underbrace{D \underbrace{L^T \underbrace{P^{-T} x}_{z_3}}_{z_2}}_{z_1} = Pb$$

and:

$L z_1 = Pb$ "gives" z_1,
z_1 "gives" z_2 "gives" z_3,
and z_3 "gives" x.

Computing P^{-T} from P and computing Pb.

The book does not show how to compute P, P^T, or P^{-T}!

In algorithm 6.5, we have the step:
"Interchange row and column j and q" (of A)
corresponds to:

$$P_{ij} A' P_{ij}^T$$

where P_{ij} interchanges row i & j, where A' is the current value of modified A.

At the output; $A' = P A P^T$.

Now, we can compute Pb by adding
"Interchange rows j and q of b"
at this step.

For P^T, note that P_{ij}^T can be implemented effectively by simply:

"Interchange column j and q".

Also, note that $P_{ij}^T P_{ij}^T = I$ since interchanging columns j and q twice, brings us back to where we started. $\Rightarrow P_{ij}^{-T} = P_{ij}^T$

Also: $$P^T = \underset{\substack{\uparrow \\ \text{1st} \\ \text{iteration}}}{P_{ij(1)}^T} P_{ij(2)}^T \cdots \underset{\substack{\uparrow \\ \text{nth iteration}}}{P_{ij(n)}^T}$$

and
$$P^{-T} = (P^T)^{-1} = \left(P_{ij(1)}^T P_{ij(2)}^T \cdots P_{ij(n)}^T \right)^{-1}$$
$$= \left(P_{ij(n)}^T\right)^{-1} \cdots \left(P_{ij(1)}^T\right)^{-1}$$
$$= P_{ij(n)}^T \cdots P_{ij(1)}^T$$

Thus, to implement multiplication by P^{-T}, simply apply the column exchanges in the reverse order than what was done by algorithm 6.5.

So store the column exchange using:
$P[j] = q$ (see below)

Corrected Algorithm 6.5 (see online corrections!)

Given A, b

$\gamma = \max_{1 \le i \le n} |a_{ii}|$ ← largest diagonal element

$\xi = \max_{i \ne j} |a_{ij}|$ ← largest off-diagonal element

$\delta = u \max(\gamma + \xi, 1)$ ← u is machine precision

$\beta = \max\left(\gamma, \frac{\xi}{\sqrt{n^2-1}}, u\right)^{1/2}$

for $k = 1, 2, \ldots, n$
  $c_{kk} = a_{kk}$ ← diagonal
end

for $j = 1, 2, \ldots, n$ (jth column of L).
  Find q so that: $|c_{qq}| \ge |c_{ii}|$, $i = j, j+1, \ldots, n$
  Interchange row and column j and q of A.
  ↑ (Corrected position of the step!)
  Interchance j and q elements of b. (Pb)
  $P_j = q$ ← for inverting: P^{-T}.

(Jth column of L:)

for $s = 1, 2, \ldots, J-1$

$l_{Js} = c_{Js} / d_s$

end

for $i = J+1, \ldots, n$

$c_{iJ} = a_{iJ} - \sum_{s=1}^{J-1} l_{Js} c_{is}$

end

$\theta_J = 0$ correct this (see online comments.)

if $J < n$

$\theta_J = \max_{J < i \leq n} |c_{iJ}|$

end

$d_J = \max\{ |c_{JJ}|, (\theta_J / \beta)^2, \delta \}$

if $J < n$

for $i = J+1, \ldots, n$

$c_{ii} = c_{ii} - c_{iJ}^2 / d_J$

end

end

end

Note that that the output of the algorithm keeps:

$$d_j \geq \delta, \quad |m_{ij}| \leq \beta$$

which keeps eigenvalues away from zero and $A = MM^T$ not very large.

If you implement the new algorithm, then:

- check correctness with example 6.2.

6.4 Trust-Region Newton Methods

We always want to compute:

$$\min_{p \in \mathbb{R}^n} m_k(p) \stackrel{\text{def}}{=} f_k + \nabla f_k^T p + \frac{1}{2} p^T B_k p,$$

such that: $\|p\| \leq \Delta_k$.

For general problems, that are also large, consider Newton-CG methods.

Trust-region Newton-CG

Use CG to solve: $B_k p_k = -\nabla f_k$,
stopping if:

(i) current solution exceeds trust region,

(ii) we have achieved sufficient accuracy (small residuals), or

(iii) we have encountered negative curvature ($p_k^T A p_k < 0$),
(if yes, follow negative curvature to the boundary).

* If we are at a "well-behaved solution", use η_k to go to the minimum

* Avoid using the Hessian since the CG method only needs $\nabla^2 f(x_k) v$.

→ Get $\nabla^2 f(x_k) v$ symbolically or

→ Use:

$$\nabla^2 f(x_k) p \approx \frac{\nabla f(x_k + hp) - \nabla f(x_k)}{h}$$

(finite differencing)

NB: **Advantages:**

* globally convergent:
  - starting with Cauchy point and improving on it.
* No matrix factorizations needed!
* Can be executed in parallel using a matrix times vector routine that executes in parallel.
* Can exploit sparsity in product (avoid multiplying zeros)
* Can move away from nonminimizing points (unlike line-search)

Disadvantages:

* Accepts all directions of negative curvature, even when they lead to an insignificant reduction.

Eg: For: $m(p) = -10^{-3} p_1 - 10^{-4} p_1^2 - p_2^2$

subject to $\|p\| \le 1$, $p = \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$.

Correct book!

Now:

$$\nabla m(\underline{p}) = \begin{bmatrix} -10^{-3} - 2\times10^{-4} p_1 \\ -2p_2 \end{bmatrix}$$

At $p=0$: $-\nabla m(0) = \begin{bmatrix} 10^{-3} \\ 0 \end{bmatrix}$ steepest descent.

Note:

$$\nabla^2 m(\underline{p}) = \begin{bmatrix} -2\times10^{-4} & 0 \\ 0 & -2 \end{bmatrix}$$

When $\underline{p}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, along steepest descent:

$$\underline{p}_1^T \begin{bmatrix} -2\times10^{-4} & 0 \\ 0 & -2 \end{bmatrix} \underline{p}_1 = -2\times10^{-4} < 0.$$

So, we have negative curvature along $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Follow this to boundary of $\|\underline{p}\|=1$ to get:

$$m\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = -10^{-3} - 10^{-4} \simeq -10^{-3}$$

reduction in model

If we follow $p_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, note:

$$p_2^T \begin{bmatrix} -2\times 10^{-4} & 0 \\ 0 & -2 \end{bmatrix} p_2 = -2 < 0,$$

we get:

$$m\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = -p_2^2 = -1$$

<u>much more reduction</u>.

<u>Solution I:</u>

Must use "Lanczos method" in references [177], [121].

more robust.

<u>Heuristic</u>: ignore "insignificant" curvatures

<u>Solution II:</u>

Precondition the Newton-CG method to a better distribution of the eigenvalues:

<u>See</u>: Algorithm 6.6, and get code from: MINPACK-2 (NMTR) or LANCELOT.

Final remarks:

For convergence, we set $\eta_k \to 0$ as discussed previously, with Newton-CG, then the trust-region algorithm 4.1 will converge, where the line:

"Obtain p_k ---"

is replaced by Newton-CG, for positive definite Hessians, $x_k \to x^*$ for these Hessians.

Thm 6.9 says that $x_k \to x^*$, for $k \to \infty$ is solved by ignoring the trust region.

$\therefore$ Constraint Optimization becomes Unconstraint.

$\Rightarrow$ Trust-regions are for global convergence!!