
Answers to Exercises for Part 3

Exercises

0.1. `x = [1;2];`
`for i=1:5,`
`g = [4*(x(1) - 5)^3 - x(2); 4*(x(2) + 1)^3 - x(1)];`
`H = [12*(x(1) - 5)^2, -1; -1, 12*(x(2) + 1)^2];`
`p = -H \ g;`
`x = x + p;`
`end`

0.2. If f is quadratic, then

$$\mathbf{H}^{(k)} \mathbf{s}^{(k)} = \mathbf{g}(\mathbf{x}^{(k+1)}) - \mathbf{g}(\mathbf{x}^{(k)})$$

where \mathbf{H} is the Hessian matrix of f . Close to $\mathbf{x}^{(k+1)}$, a quadratic model is a close fit to any function, so we demand this property to hold for our approximation to \mathbf{H} .

0.3. `function Hv = Htimes(v)`
`% Given a vector v, this function returns an approximation to`
`% Hv, where H is the Hessian matrix of myfnct.`
`% We use a function [f,g] = myfnct(x) in order to approximate`
`% Hv by a finite difference. We choose our step length h=.001.`
`% For poorly scaled functions, this value might need to be changed.`
`%`
`% This function needs to know the point myxnow at which the Hessian`
`% should be approximated. This value is assumed to be already`
`stored`
`% in the global variable myxnow.`
`global myxnow`
`h = .001;`
`[f, g] = myfnct(myxnow);`
`[fp, gp] = myfnct(myxnow + h * v);`
`Hv = (gp - g)/h;`

0.4. • The first line does n^2 divisions. It would be better to add parentheses to drop this to n : $\mathbf{B} = \mathbf{B} + (\mathbf{y}-\mathbf{B}*\mathbf{s})*(\mathbf{s}'/(\mathbf{s}'*\mathbf{s}))$.

• It is silly to refactor the matrix \mathbf{B} each time, when it is just a rank-1 update of the previous matrix. Instead, update a factorization or (less desirable) update the inverse.

0.5. a. Advantage: can use our software for unconstrained optimization.

Disadvantage:

- The function becomes more complicated. For instance, if it was originally quadratic, it now becomes quartic.

- Function evaluation is more expensive.
- For every minimizer \hat{x} for the original problem, we now have two minimizers: $\pm\sqrt{\hat{x}}$.

b. **Some possible answers:** $x = e^y/(1 + e^y)$, $x = \sin^2 y$, $x = .5(\sin y + 1)$, $x = 1/(1 + y^2)$.

0.6. Here is how I would make the decision:

- If the function is not differentiable, use Nelder-Meade.
- If 2nd derivatives (Hessians) are cheaply available and there is enough storage for them, use Newton.
- Otherwise, use quasi-Newton (with a finite-difference 1st derivative if necessary).

0.7.

$$f(\mathbf{x}) = x_1^4 + x_2(x_2 - 1)$$

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} 4x_1^3 \\ 2x_2 - 1 \end{bmatrix}, \quad \mathbf{H}(\mathbf{x}) = \begin{bmatrix} 12x_1^2 & 0 \\ 0 & 2 \end{bmatrix}.$$

Step 1:

$$\mathbf{p} = - \begin{bmatrix} 12x_1^2 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 4x_1^3 \\ 2x_2 - 1 \end{bmatrix} = - \begin{bmatrix} 48 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 32 \\ -3 \end{bmatrix} = \begin{bmatrix} -32/48 \\ +3/2 \end{bmatrix}$$

so

$$\mathbf{x} \leftarrow \begin{bmatrix} 2 \\ -1 \end{bmatrix} + \begin{bmatrix} -2/3 \\ +3/2 \end{bmatrix} = \begin{bmatrix} 4/3 \\ 1/2 \end{bmatrix}$$

0.8. Since $\mathbf{p}^T \mathbf{g} = -\mathbf{p}^T \mathbf{H} \mathbf{p}$, we are assured that $\mathbf{p}^T \mathbf{g} < 0$ if \mathbf{H} is positive definite, and this means we are walking downhill.

- 0.9.
- Newton: often converges with a quadratic rate when started close enough to a solution, but requires both first and second derivatives (or good approximations of them) as well as storage and solution of a linear system with a matrix of size 2000×2000 .
 - Quasi-Newton: often converges superlinearly when started close enough to a solution, but requires first derivatives (or good approximations of them) and storage of a matrix of size 2000×2000 .
 - Pattern search: converges only linearly, but has good global behavior and requires only function values, no derivatives.

If first derivatives (or approximations) were available, I would use QN, with updating of the matrix factorization (or a limited memory version, but we did not talk about this option). Otherwise, I would use pattern search.

0.10.

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} e^{x_1+x_2}(1+x_1) \\ e^{x_1+x_2}x_1 + 2x_2 \end{bmatrix} = \begin{bmatrix} 8 \\ 4.7726 \end{bmatrix};$$

$$\mathbf{H}(\mathbf{x}) = \begin{bmatrix} e^{x_1+x_2}(2+x_1) & e^{x_1+x_2}(1+x_1) \\ e^{x_1+x_2}(1+x_1) & e^{x_1+x_2}x_1+2 \end{bmatrix} = \begin{bmatrix} 12 & 8 \\ 8 & 6 \end{bmatrix}$$

Now $\det(\mathbf{H}) = 72 - 64 = 8$, so

$$\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g} = \frac{1}{8} \begin{bmatrix} 6 & -8 \\ -8 & 12 \end{bmatrix} \begin{bmatrix} -8 \\ -4.7726 \end{bmatrix} = \begin{bmatrix} -1.2274 \\ 0.8411 \end{bmatrix}$$

(We'd never use this inverse formula on a computer, except possibly for 2x2 matrices. Gauss elimination is generally better:

$$\mathbf{L} = \begin{bmatrix} 1 & 0 \\ 2/3 & 1 \end{bmatrix}, \mathbf{U} = \begin{bmatrix} 12 & 8 \\ 0 & 2/3 \end{bmatrix}.)$$

Note that $\mathbf{p}^T\mathbf{g} = -5.8050 < 0$, so the direction is downhill.

Method	convergence rate	Storage	f evals/itn	\mathbf{g} evals/itn	\mathbf{H} evals/itn
Truncated Newton	> 1	$O(n)$	1	$\leq n + 1$	0
Newton	2	$O(n^2)$	0^1	1	1
Quasi-Newton 0.11.	$> 1^2$	$O(n^2)$	0^1	1	0
steepest descent	1	$O(n)$	0^1	1	0
Conjugate gradients	1	$O(n)$	0^1	1	0

Notes on Answer:

- Once the counts for the linesearch are omitted, no function evaluations are needed, but credit was given for 1, too, as long as you were consistent about it.
- For a single step, Quasi-Newton is superlinear; it is n -step quadratic.
 - Assume that all of these methods are convergent and that any line search is exact (i.e., the true optimal value of the steplength parameter is used).
 - Don't include the cost of the line search in your table entries. We are omitting this cost because it is the same, independent of method.
 - f is the function, \mathbf{g} is the gradient, and \mathbf{H} is the Hessian matrix. "evals/itn" means the number of evaluations per iteration.
 - The convergence rate should be "1" for linear, " > 1 " for superlinear, or "2" for quadratic.
 - Storage should be either $O(1)$, $O(n)$, or $O(n^2)$, where n is the number of variables (i.e., the dimension of \mathbf{x}).
 - "Conjugate gradients" means the nonlinear cg method, not the one for solving linear systems (minimizing quadratics).

- 0.12. I would use pattern search to minimize $F(\mathbf{x}) = -y(1)$ as a function of \mathbf{x} . When a function value $F(\mathbf{x})$ is needed, I would call one of MATLAB's stiff ode solvers, since I don't know whether the problem is stiff or not, and return the value computed as $y(1)$. The value of \mathbf{x} would need to be passed to the function that evaluates f for the ode solver; one way to do this is to use a global variable.

I chose pattern search because it has proven convergence and does not require derivatives of F with respect to \mathbf{x} . Note that these derivatives are not available for this problem: we can compute derivatives of y with respect to t but not with respect to \mathbf{x} . And since our value of $y(1)$ is only an approximation, the use of finite differences to estimate derivatives with respect to \mathbf{x} would yield values too noisy to be useful.

- 0.13. One particular solution to the equality constraint: $[4, 0]^T$.
A basis for the nullspace of the matrix: $[2, -1]^T$.
General solution to equality constraint:

$$x = \begin{bmatrix} 4 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ -1 \end{bmatrix} z$$

where z is an arbitrary scalar.

The problem becomes

$$\min_z (4 + 2z)^2 + (-z)^2$$

subject to $4 + 2z \geq 0$, $-z \geq 0$, so the barrier problem that is equivalent to the original problem is

$$\min_z (4 + 2z)^2 + (-z)^2 - \mu \log(4 + 2z) - \mu \log(-z)$$