CMPUT 340—Introduction to Numerical Methods

Assignment 2

Winter 2007
Department of Computing Science
University of Alberta

Due: *Sunday, March 4 at 23:59:59 local time*
Worth: 15% of final grade

Instructor: Dale Schuurmans, Ath409, x2-4806, dale@cs.ualberta.ca

---

Note: This assignment is to be submitted electronically by using the Unix "astep" command which is available on the undergraduate machines. You need to submit the answers to all written questions in hard copy to the course drop box. Matlab functions need to be submitted in .m files (all of the Matlab functions required are discussed below). You can submit your files by typing "`astep -c c340 -p a2` *file1 file2 ...*". See http://ugweb.cs.ualberta.ca/~astep

Note: For this assignment you *are* allowed to use the functions in `help matfun`, but not those in `help optim`.

Part 1 (Constraint satisfaction—7%)

(a) (1%) Write a Matlab function `[x,flag] = Bisection(f,x0,TOL)` which uses the *bisection* method to find a point x such that $f(x) = 0$. (See Page 224 in the text.) You will first have to find a bracket (a, b) of the solution x that includes the initial point x_0 as one of the endpoints (that is, either $a = x_0$ or $b = x_0$). Try to find a solution x with absolute error bounded by *TOL*. Return *flag* $= 0$ if the subroutine fails and *flag* $= 1$ if it succeeds. Submit the function in a file called "Bisection.m".

Note that your routine should take the name of a function f as the first argument and call this function at selected points x. You can do this by using the `feval` command (see "`help feval`"). For example, `feval('exp',2)` calls `exp` on `2` and returns `7.3891`.

(b) (1%) Write a Matlab function `[x,flag] = SafeNewton1D(f,g,x0,TOL)` which combines *bisection* with *Newton's* method to find a point x such that $f(x) = 0$ in a safe manner. (See Pages 230 and 236 in the text.) Note that the function g must return the derivative of f at points x. Try to find a solution x with absolute error bounded by *TOL*. Return *flag* $= 0$ if the subroutine fails and *flag* $= 1$ if it succeeds. Submit the function in a file called "SafeNewton1D.m".

(c) (1%) Write a Matlab function `[x,flag] = SafeSecant1D(f,x0,TOL)` which combines *bisection* with the *secant* method to find a point x such that $f(x) = 0$ in a safe manner. (See Pages 232 and 236 in the text.) Try to find a solution x with absolute error bounded by *TOL*. Return *flag* $= 0$ if the subroutine fails and *flag* $= 1$ if it succeeds. Submit the function in a file called "SafeSecant1D.m".

(d) (2%) Computer Problem 5.15 on Page 251.
(Use any of the techniques implemented above to solve the problems specified.)

(e) (2%) Computer Problem 5.28(b) on Page 254. (**Note** only do Part (b).)
Submit a Matlab function `[Ainv] = NewtInv(A)` in a file "NewtInv.m".

Remember to submit all written answers in the course drop box.

Part 2 (Optimization—8%)

(a) (2%) Write a Matlab function `[x,flag] = LineSearch(f,g,x0,dir,TOL)` which finds a local minimum $\mathbf{x}$ of the function f along the ray $\mathbf{x}_0 + \alpha\mathbf{dir}$ starting at $\mathbf{x}_0$, where **dir** is a vector specifying the direction and $\alpha \geq 0$. Note that the function g must return the *gradient* of f at points $\mathbf{x}$. Return *flag* $= 0$ if the subroutine fails and *flag* $= 1$ if it succeeds. Submit the function in a file called "LineSearch.m".

Note You can implement the line search using any 1-dimensional optimization method you wish. Some possible choices include *golden section search* (Page 271), *Newton's method* (Page 275) or *successive parabolic interpolation* (Page 273). This function will be needed as a subroutine in the two functions below.

(b) (2%) Write a Matlab function `[x,flag] = SteepDescent(f,g,x0,TOL)` which uses the *steepest descent* method to find a local minimum $\mathbf{x}$ of the function f. (See Page 277 in the text.) Note that the function g must return the *gradient* of f at points $\mathbf{x}$. Return *flag* $= 0$ if the subroutine fails and *flag* $= 1$ if it succeeds. Submit the function in a file called "SteepDescent.m".

Note that you will need to call the `LineSearch` routine implemented above.

(c) (2%) Write a Matlab function `[x,flag] = ConjGradient(f,g,x0,TOL)` which uses the *conjugate gradient* method to find a local minimum $\mathbf{x}$ of the function f. (See Page 283 in the text.) Note that the function g must return the *gradient* of f at points $\mathbf{x}$. Return *flag* $= 0$ if the subroutine fails and *flag* $= 1$ if it succeeds. Submit the function in a file called "ConjGradient.m".

Note that you will need to call the `LineSearch` routine implemented above.

(d) (1%) Run your steepest descent and conjugate gradient methods on problems defined by $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top A\mathbf{x} - \mathbf{x}^\top\mathbf{b} + c$, where A is a positive semi-definite matrix, $\mathbf{b}$ is a vector and c is a scalar. (You can construct A using the observation that $A = B^\top B$ is positive semi-definite for any matrix B. Also note that the gradient of f is given by $A\mathbf{x} - \mathbf{b}$.)

Compare the accuracy, efficiency and robustness of the two methods for solving these problems.

(e) (1%) Use your steepest descent and conjugate gradient methods to find the minimum of Rosenbrock's function $f(x, y) = 100(y - x^2)^2 + (1 - x)^2$. Try each method from the starting points $[-1 \;\; 1]^\top$, $[0 \;\; 1]^\top$ and $[2 \;\; 1]^\top$. Plot the path taken in the plane by each method from each starting point.

Remember to submit all written answers in the course drop box.

Appendix: Exercise descriptions from the textbook

Computer Problem 5.15. (Page 251)

If an amount a is borrowed at interest rate r for n years, then the total amount to be repaid is given by

$$a(1+r)^n.$$

Yearly payments of p each would reduce this amount by

$$\sum_{0}^{n-1} p(1+r)^i \;=\; p\frac{(1+r)^n-1}{r}.$$

The loan will be repaid when these two quantities are equal.

(a) For a loan of $a = \$100,000$ and yearly payments of $p = \$10,000$, how long will it take to pay off the loan if the interest rate is 6 percent, i.e., $r = 0.06$?

(b) For a loan of $a = \$100,000$ and yearly payments of $p = \$10,000$, what interest rate r would be required for the loan to be paid off in $n = 20$ years?

(c) For a loan of $a = \$100,000$, how large must the yearly payments p be for the loan to be paid off in $n = 20$ years at 6 percent interest?

You may use any method you like [*among those you have implemented*] to solve the given equation in each case. For the purpose of this problem, we will treat n as a continuous variable (i.e., it can have fractional values).

Computer Problem 5.28(b). (Page 254) (**Note** only do Part (b).)

Newton's method can be used to compute the inverse of a nonsingular $n \times n$ matrix $\mathbf{A}$. If we define the function $\mathbf{F} : I\!R^{n\times n} \to I\!R^{n\times n}$ by

$$\mathbf{F}(\mathbf{X}) \;=\; \mathbf{I} - \mathbf{A}\mathbf{X},$$

where $\mathbf{X}$ is an $n\times n$ matrix, then $\mathbf{F}(\mathbf{X}) = \mathbf{O}$ precisely when $\mathbf{X} = \mathbf{A}^{-1}$. Because $\mathbf{F}'(\mathbf{X}) = -\mathbf{A}$, Newton's method for solving this equation has the form

$$\begin{aligned}\mathbf{X}_{k+1} \;&=\; \mathbf{X}_k - [\mathbf{F}'(\mathbf{X}_k)]^{-1}\mathbf{F}(\mathbf{X}_k)\\ &=\; \mathbf{X}_k + \mathbf{A}^{-1}(\mathbf{I} - \mathbf{A}\mathbf{X}_k).\end{aligned}$$

But $\mathbf{A}^{-1}$ is what we are trying to compute, so instead we use the current approximation to $\mathbf{A}^{-1}$, namely $\mathbf{X}_k$. Thus, the iteration scheme takes the form

$$\mathbf{X}_{k+1} \;=\; \mathbf{X}_k + \mathbf{X}_k(\mathbf{I} - \mathbf{A}\mathbf{X}_k).$$

(b) Write a program to compute the inverse of a given input matrix $\mathbf{A}$ using this iteration scheme. A reasonable starting guess is to take

$$\mathbf{X}_0 \;=\; \frac{\mathbf{A}^\top}{\|\mathbf{A}\|_1 \cdot \|\mathbf{A}\|_\infty}.$$

Test your program on some random matrices and compare its accuracy and efficiency with conventional methods for solving the inverse, such as LU factorization or Gauss-Jordan elimination.