

Homework 3

CS 726, Semester I, 2011–12

October 3, 2011

This assignment should be submitted electronically using the instructions on the course web page. The assignment name is hwk3 and you should hand in exactly 5 files with the following names:

geodesic.m, SteepDescent.m, sdplot.ps, Newton.m, ndplot.ps

Consider the following optimization problem: computing geodesics, i.e. the shortest route between two points, on a two-dimensional surface on which it is easier to move quickly on some parts of the surface than on others. You can think of the reason for this as some parts of the surface being more hilly than others, or more “sticky”. Mathematically, the problem is to minimize

$$\int_0^1 \rho(x(t), y(t)) \left\{ \left(\frac{dx(t)}{dt} \right)^2 + \left(\frac{dy(t)}{dt} \right)^2 \right\} dt$$

over smooth functions $x(t)$ and $y(t)$ defining the path on the two-dimensional surface as a function of time, with the boundary conditions $(x(0), y(0)) = (a, b)$, $(x(1), y(1)) = (c, d)$ (any two given points in \mathbf{R}^2). Here $\rho(x, y)$ is a function describing how difficult it is to move at the given point on the surface (big values of mean difficult to move (very sticky, or big hill, depending on how you want to interpret things)). If $\rho(x, y) \equiv 1$, the surface is uniformly flat, and the unique solution is the straight line from (a, b) to (c, d) , but if has large values along this line and smaller values off it, it may be better to “go around”. That’s the idea of the problem: what is the shortest route?

Let us approximate the infinite-dimensional problem by a finite-dimensional one, discretizing the functions $x(t)$ and $y(t)$, i.e. replacing them by corresponding vectors $X \in \mathbf{R}^{N+2}$ and $Y \in \mathbf{R}^{N+2}$, with (X_0, Y_0) and (X_{N+1}, Y_{N+1}) fixed to the given boundary points (a, b) and (c, d) , and the other vector elements $(X_1, Y_1), \dots, (X_N, Y_N)$ unknown (these define the points along the path at discrete time intervals). Approximating the derivatives in the objective function by finite differences, we wish to minimize

$$F(X, Y) = \Delta t \sum_{i=0}^N \rho(X_i, Y_i) \left\{ \left(\frac{X_{i+1} - X_i}{\Delta t} \right)^2 + \left(\frac{Y_{i+1} - Y_i}{\Delta t} \right)^2 \right\}$$

where $\Delta t = \frac{1}{N+1}$.

1. Show that if $\rho(x, y) \equiv 1$, the solution is the vector of equally spaced points on the line between (a, b) and (c, d) . If you find this difficult, try it first for $N = 1$ (one pair of variables X_1, Y_1), and then $N = 2$, etc. This, and any other questions not requiring code should be handed in directly to Ferris in class.
2. Using $\rho(x, y) = 1 + \alpha e^{-\beta(x^2+y^2)}$, where α and β are given constants, compute the gradient of F with respect to the $2N$ variables $X_1, Y_1, \dots, X_N, Y_N$, and write a Matlab function geodesic.m

(modeled exactly after obja.m) to compute F and its gradient for given X, Y . You'll want to collect X and Y together into one vector of length $2N$, say z . You will need to use the chain rule. Check whether your gradient computation is correct by comparing it with a vector of finite differences, i.e.,

$$v_k = \frac{F(z + he_k) - F(z)}{h}, \quad k = 1, \dots, 2N,$$

where the e_k are unit coordinate vectors and h is small, e.g. 10^{-6} . If the discrepancy is much bigger than h , you have probably made a mistake in your formulas. It's essential to find and correct this before continuing.

3. Download the StepSize.m routine from the web site. The header line of the subroutine is:

```
function [alfa,x] = StepSize(fun, x, p, alfa, params)
```

The input parameters are

fun a pointer to a function (such as obja, objb, objc)

x a structure that on input has x.p set to the starting point values

```
x = struct('p', [-1.2, 1.0]);
```

p a vector containing the search direction

alfa the initial value of the step length

params a structure containing parameter values for the routine

```
params = struct('c1', 0.1, 'c2', 0.7, 'maxit', 100);
```

4. Test this subroutine by writing a Matlab program (SteepDescent.m) to implement the method of steepest descent where $p = -\nabla f(x)$. Stop the method when either the gradient of the objective function has norm less than 10^{-4} or 1,000 steps have been made. The header of the function should be

```
function [inform,x] = SteepDescent(fun,x,sdparams)
```

(The inputs fun and x are as above, and sdparams is the following structure

```
sdparams = struct('maxit', 1000, 'toler', 1.0e-4);
```

The output inform is a structure containing two fields, **inform.status** is 1 if the gradient tolerance was achieved and 0 if not, **inform.iter** is the number of steps taken and **x** is the solution structure, with point, function and gradient evaluations at the solution point).

5. Matlab functions that implement the functions in the following list

(a) $f(x) = x_1^2 + 5x_2^2 + x_1 - 5x_2$

(b) $f(x) = x_1^2 + 5x_1x_2 + 100x_2^2 - x_1 + 4x_2$

(c) $f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$

can be found (as `obja.m`, `objb.m`, `objc.m`) on the web-site for this course (along with a more complicated `objd.m`). The program will be tested using the script `hwk3a.m` that is available on the web site. For each of the above three functions, the algorithm will be run using the starting point $(-1.2, 1)$, and with an initial estimate of the step size at each call to your `linesearch` routine being 1.

Do not print out the value of x at each iteration! Make sure that the values of `numf` and `numg` get reported correctly.

6. Use your steepest descent code to get a rough approximation to a minimizing path for various values of N, a, b, c, d, \dots . To get started, try $N = 10$, $(a, b) = (-1, -1)$, $(c, d) = (1, 1)$, $\alpha = \beta = 1$. Convergence will probably be very slow so do not attempt to find an accurate solution. It will help a lot if you pick a physically sensible initial guess. (The straight line between the two boundary points is an obvious choice, but this may not be a good idea: why? Try it.) Is there sometimes more than one local minimum?
7. Use Matlab to plot your approximate minimizing paths (try “help plot”) in the same figure as the contours of the weight function ρ (try “help contour”). If the minimizing path makes no physical sense, you must have made a mistake: try again with a smaller value of N . Hand in a hard copy (see “help print” to generate the required “`sdplot.ps`” file). (Suggest using values of $\alpha = 4$, $\beta = 5$).
8. Derive the formula for the Hessian of the objective function in the geodesic problem, and add this to `geodesic.m`. The code should check the mode input from the call and compute the Hessian only if it is requested. This is important as otherwise calls to the function in the line search will all result in Hessian evaluations! If you order the variables appropriately, H is banded, i.e. $H_{ij} = 0$ for $|i - j| > m$, for some small integer m independent of n , the order of the matrix. (We say that the matrix has half-bandwidth m ; if $m = 1$ we say the matrix is tridiagonal.) What is the bandwidth of the geodesic Hessian? Make sure your code returns the Hessian as a sparse matrix. Type “help sparse” at the Matlab prompt to learn more about this convenient and painless way to work with sparse matrices. You get an n by n zero sparse matrix by typing “`A = sparse(n,n)`” and convert an ordinary full matrix A to the sparse format by “`A=sparse(A)`”. The sparse data structure stores the matrix efficiently, just keeping track of the nonzero entries. Also, make sure that the Hessian returned by your function is exactly symmetric, that is, $H - H'$ is the zero matrix.
9. This question asks you to compare several ways to implement Newton’s method with a line search, testing it on the geodesic problem. The only difference is the way of dealing with an indefinite Hessian.

Modify your steepest-descent program from above to minimize a function by Newton’s method. You can simply modify the code in `SteepestDescent` that computes the direction. You should use line search parameters $c_1 = 10^{-4}$ and $c_2 = 0.9$ and always initialise your `StepSize` routine with $\alpha = 1$.

```
function [inform,x] = Newton(fun,x,nparams)
```

Within your code you need to solve for the Newton step, $HS = -g$. The direction should be calculated in two different ways. Use `nparams.method` to switch between the two.

- (a) `nparams.method = "direct"`: Using the `\` built in solver of MATLAB. If the direction fails to be a descent direction you should take a scaled steepest descent direction based on the diagonal of the Hessian:

$$\max\{|H_{ii}|, 0.01\}$$

- (b) `"chol"`: A simple algorithm that tries a Cholesky factorization of H , calling Matlab's `chol`, and if it fails, repeatedly adds a multiple of the identity to H until the matrix is sufficiently positive definite that Cholesky succeeds.

One way to do this is Algorithm 3.3 on p.51 of the text. Notice that here the notation $A = LL^T$ is used, where L is lower triangular, while Matlab's Cholesky returns an upper triangular matrix R so that $A = R^T R$. By requesting a second output argument, as in `[R, fail]=chol(A)`, you can check whether the factorization succeeded (because A is sufficiently positive definite) or failed (because it is not). Once a Cholesky factor R is obtained, with $H + \alpha I = R^T R$ for some $\alpha \geq 0$, the step is obtained from $-R \setminus (R' \setminus g)$, which does forward and back substitution to solve the triangular systems. Use `spy` to display the sparsity pattern of R and compare it with the sparsity of H . Is it more dense (i.e., is there "fill-in")? Does this depend on whether or not it was necessary to add a multiple of the identity?

They should both generate the same Newton step when the Hessian H is (sufficiently) positive definite; be sure to check that they do.

10. Compare your two versions of Newton's method on the geodesic problem. Use a starting point where the Hessian is not positive definite so that you see a difference between the methods. If you have implemented them correctly, you should see different initial iterates but then all of them should behave very similarly once they find points where H is positive definite. You should clearly see quadratic convergence. If not, there is probably a bug in your program. Demonstrate the quadratic convergence by outputting how the norm of the gradient varies with the iteration count. In testing your codes, I will choose the starting point, use a reasonably large N (say 100; remember that the Hessian should be stored as a sparse matrix) and parameters α and β to make the problem hard enough that you see a big difference between Newton and steepest descent. Plot the Newton solution overlaid on a contour plot of ρ and hand in this plot as `ndplot.ps`. Is it more accurate than what you obtained from steepest descent, and how much faster is it to compute? How do the two versions of Newton's method compare in terms of efficiency and exploitation of sparsity?