7 Mathematical Methods

7.3 Minimisation Methods (8 units)

There are no prerequisites for this project.

You should write your own minimisation method programs: it is not sufficient to use routines that are distributed as part of MATLAB, other software packages, or other programming languages. You can, however, exploit the matrix manipulation capabilities of MATLAB, other software packages, or other progamming languages. Note that this project is particularly concerned with clarity and conciseness of expression. Overly long write-ups in which the important points are buried within lots of irrelevant detail will be penalised. You should therefore aim to produce concise, relevant answers that address the issues, together with a carefully chosen selection of diagrams and graphs.

1 Introduction

There are many numerical methods for finding the least value of a function of N variables, $f(x_1, x_2, x_3, \ldots) = f(\mathbf{x})$, say, given that the first derivatives

$$g_i = \frac{\partial f}{\partial x_i}, \qquad i = 1, 2, \ldots N \ , \tag{1}$$

can be calculated. Most of the methods are iterative and each iteration reduces the value of $f(\mathbf{x})$ by searching along a descent direction in the space of the variables in the following way.

The iteration begins with a starting point $\mathbf{x}_0$, and at this point the gradient vector $\mathbf{g}$ is calculated. Then a search direction, $\mathbf{s}$, say, is chosen, that satisfies the condition $\mathbf{g} \cdot \mathbf{s} < 0$ (the dot denotes a scalar product). It follows that if we move from $\mathbf{x}_0$ in the direction of $\mathbf{s}$, then the value of $f(\mathbf{x})$ becomes smaller initially. In other words the function of one variable

$$\phi(\lambda) = f(\mathbf{x}_0 + \lambda \mathbf{s}) \tag{2}$$

satisfies the condition $\phi'(0) < 0$ which is equivalent to $\mathbf{g} \cdot \mathbf{s} < 0$. The next stage is to consider the function $\phi(\lambda)$, and choose a value of λ, λ^* say, that satisfies the inequality

$$f(\mathbf{x}_0 + \lambda^* \mathbf{s}) < f(\mathbf{x}_0) \ . \tag{3}$$

Usually λ^* will be chosen [*] to minimise $\phi(\lambda)$. The vector $\mathbf{x}_0$ is replaced by $\mathbf{x}_1 = \mathbf{x}_0 + \lambda^* \mathbf{s}$ and another iteration is begun.

The project is to investigate three well-known algorithms (*Steepest Descents*, the *Conjugate Gradient algorithm*, and the *DFP algorithm*) by applying them to two functions. Using x for x_1, y for x_2, etc., consider the "bedpan function"

$$x + y + \frac{x^2}{4} - y^2 + (y^2 - \frac{x}{2})^2 \ , \tag{4}$$

and the following function, which has similar properties to the Rosenbrock function,

$$(1 - x)^2 + 80(y - x^2)^2 \ . \tag{5}$$

[*] In your program for this project, you should allow for manual input of estimates for λ^*, based on plots of $\phi(\lambda)$. To speed up longer runs you may *if you wish* also use MATLAB routines or other automated search algorithms to minimise $\phi(\lambda)$, but this is not required. In either case, the values of λ^* used should appear in the hard copy of results.

In addition the following quadratic function of three variables will be used to demonstrate some properties of the DFP algorithm:

$$0.4x^2 + 0.2y^2 + z^2 + xz \ . \tag{6}$$

2 Steepest Descents

The Steepest Descents method simply uses the search direction $\mathbf{s} = -\mathbf{g}$. Write a program to implement the algorithm as described above. Use a simple x–y plot of $\phi(\lambda)$ to help you determine λ^* at each stage (you need never determine λ^* to more than 2 significant figures). At each stage after the first, arrange for your program to display the current value of $f(\mathbf{x})$ and the decrease achieved over the last step. Also arrange for a plot of the iteration points $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, etc., (a sequence of line segments will illustrate the methods well). The iteration point plot may be built up as the calculation proceeds, or you can store the data and produce it on command from your programme at a point of your choosing.

[N.B. A well-implemented fully automatic algorithm for general use will need to have checks for special cases and exceptions built into it. For example, if a point $\mathbf{x}_n$ is encountered for which $\mathbf{g} \approx \mathbf{0}$ then a stationary point has been found and the process should quit. Likewise, if the iteration points are not changing significantly a fully automatic algorithm ought to quit. You may find it helpful to include such features in your program. If you wish to proceed semi-automatically, with λ^* being decided by eye from the plot at each stage, there is no need to include the special checks in your code.]

Question 1 Obtain contour plots and/or surface plots of functions (4) and (5) (this should be fairly straightforward to do using MATLAB).

Work out analytically where they have minima and find their minimum values. Suitable axis intervals are $-1.5 \leqslant x, y \leqslant 1.5$.

Question 2 Using function (4) and starting from $(-1.0, -1.3)$, run the Steepest Descents method for 10 iterations. Produce a plot of the progress of the iteration. On the basis of your numerical results (i.e., imagine that you do not know the analytical answer), estimate the minimum value of the function at the point to which your iteration is converging, and estimate intervals in which the co-ordinates of the minimum lie. What general statement can you make about the precision with which the minimum value itself can be found, compared to the precision with which the minimum point is known? What property of the function being minimised gives rise to this effect?

Question 3 Using function (5) and starting from $(0.676, 0.443)$, run the Steepest Descents method for at least 9 iterations, and produce a plot of the progress of the iteration. To what point do you think the iteration will eventually converge? Comment on the rate of convergence. How sensitive is the iteration path to variations in the choice of λ^* at each stage? Comment on the circumstances that can make steepest descents inefficient.

3 Conjugate Gradients

The conjugate gradients algorithm uses steepest descents for its first step and then adjusts the search direction in an attempt to overcome the problems of steepest descents alone. Let $\mathbf{x}_0$,

$\mathbf{x}_1$ be two successive points where $\mathbf{x}_1$ has been obtained using steepest descents from $\mathbf{x}_0$, and let $\mathbf{g}_0$, $\mathbf{g}_1$ be the corresponding gradients (the initial search direction is $\mathbf{s}_0 = -\mathbf{g}_0$). Take the second search direction as

$$\mathbf{s}_1 = -\mathbf{g}_1 + \beta\,\mathbf{s}_0 = -\mathbf{g}_1 - \beta\,\mathbf{g}_0 \;\text{ where }\; \beta = \frac{\mathbf{g}_1 \cdot \mathbf{g}_1}{\mathbf{g}_0 \cdot \mathbf{g}_0}. \tag{7}$$

If $f(\mathbf{x})$ is a quadratic function of N variables then the choice of directions may be continued up to the N^{th} search direction to give the N conjugate directions

$$\mathbf{s}_k = -\mathbf{g}_k + \beta\,\mathbf{s}_{k-1} \;\text{ where }\; \beta = \frac{\mathbf{g}_k \cdot \mathbf{g}_k}{\mathbf{g}_{k-1} \cdot \mathbf{g}_{k-1}}\;.$$

In this case, if all the values of λ^* had been chosen to minimise the $\phi\,(\lambda)$ exactly at each stage, the algorithm would have converged. In practice of course $f(\mathbf{x})$ may not be quadratic and the values λ^* may not be chosen exactly, and in this case it is usual in practice to restart the method after N steps. When $N = 2$, as it is for functions (4) and (5), this implies that every other step is a steepest descent.

Write a program to implement the conjugate gradients algorithm, with the same features as used for the steepest descents method, but with the search direction determined as just described.

> **Question 4** For the function (4), repeat Q2 using the conjugate gradients algorithm, and compare results.

> **Question 5** For the function (5), repeat Q3 using the conjugate gradients algorithm, and compare results.

Does the conjugate gradients algorithm offer much of an improvement over steepest descents?

4 DFP Algorithm

The Taylor Series expansion of any smooth function $f(\mathbf{x})$ may be written

$$f(\mathbf{a} + \Delta\mathbf{x}) \cong f(\mathbf{a}) + \mathbf{g} \cdot (\Delta\mathbf{x}) + \tfrac{1}{2}(\Delta\mathbf{x})^T \mathbf{H}^{-1} (\Delta\mathbf{x}) + \cdots$$

where the gradient vector $\mathbf{g}$ is evaluated at $\mathbf{x} = \mathbf{a}$ and $\mathbf{H}^{-1} \equiv \mathbf{G}$ is the Hessian matrix, i.e., the matrix of second derivatives

$$G_{ij} = \frac{\partial^2 f}{\partial x_i\,\partial x_j}\;.$$

Finding a point where $\mathbf{g}$ vanishes is therefore similar to the Newton–Raphson method for a system of equations, and if $\mathbf{H}$ were known and $f(\mathbf{x})$ were a quadratic function, the point could be found in a single step. However the matrix $\mathbf{H}$ is not available initially unless second derivatives are calculated; this is not always easy and in any case can be time-consuming, especially for large N. Therefore we now study a very successful technique that extends the steepest descent method by forming a suitable $\mathbf{H}$-matrix as the calculation proceeds. It is known as the DFP algorithm and is one of the class of "variable metric methods". It can be shown that $\mathbf{H}^{-1}$ converges to the Hessian (you are not required to prove this).

The DFP algorithm works as follows. The search direction is taken as $\mathbf{s} = -\mathbf{H}\mathbf{g}$, where $\mathbf{H}$ is taken initially as the identity matrix. At each stage $\phi\,(\lambda)$ is minimised by choosing a value λ^* as before, but then $\mathbf{H}$ is modified by replacing it with

$$\mathbf{H}^* = \mathbf{H} - \frac{\mathbf{H}\mathbf{p}\mathbf{p}^T\mathbf{H}}{\mathbf{p}^T\mathbf{H}\mathbf{p}} + \frac{\mathbf{q}\mathbf{q}^T}{\mathbf{p}^T\mathbf{q}}, \tag{8}$$

where $\mathbf{p}$ and $\mathbf{q}$ are column vectors giving the changes in $\mathbf{g}$ and $\mathbf{x}$ respectively during the step, that is

$$\mathbf{p} = \mathbf{g}\left(\mathbf{x}_0 + \lambda^* \mathbf{s}\right) - \mathbf{g}\left(\mathbf{x}_0\right), \quad \mathbf{q} = \lambda^* \mathbf{s} \tag{9}$$

(Note: $\mathbf{H}^* \mathbf{p} = \mathbf{q}$ which is useful when checking your program.)

Write a program to implement the DFP algorithm, with the same features as used for the two preceding programs, but with the search direction determined as just described. Include provision to print out $\mathbf{H}$.

Question 6 A property of the DFP algorithm is that it calculates the least value of a quadratic function in at most N iterations for any initial choice of $\mathbf{x}_0$ if on each iteration the value of λ^* is calculated to minimise exactly the function $\phi(\lambda)$. Apply the DFP algorithm to (6) for three iterations from starting point $\mathbf{x}_0 = (1, 1, 1)$ using the sequence of values

$$\lambda^* = 0.3942,\ 2.5522,\ 4.2202.$$

There is no need to verify these values to this precision, but your program will already have facilities for checking that these values are appropriate. Investigate how sensitive the result obtained after three iterations is to small changes in these values. Verify that $\mathbf{H}$ does indeed tend to the inverse Hessian matrix. You may note that

$$\begin{pmatrix} 0.8 & 0 & 1 \\ 0 & 0.4 & 0 \\ 1 & 0 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} 3.3333 & 0 & -1.6667 \\ 0 & 2.5 & 0 \\ -1.6667 & 0 & 1.3333 \end{pmatrix} \tag{10}$$

Question 7 For the function (4), repeat Q2 using the DFP algorithm. Examine $\mathbf{H}$ and compare with the true value.

Question 8 For the function (5), repeat Q3 using the DFP algorithm. Examine $\mathbf{H}$ and compare with the true value.

Question 9 Compare the performance of the three methods for these functions.

References

[1] Fletcher, R. and Powell, M.J.D. *Rapidly convergent descent method for minimisation*, Computer Journal, 7 (1963).

[2] McKeown, J.J., Meegan, D., and Sprevak, D. *An Introduction to Unconstrained Optimisation - A Computer Illustrated Text*, IOP Publishing (1990). Although references to the computer programs (designed for BBC micro) are best ignored, the text is still relevant.