Numerical Methods in MATLAB

Center for Interdisciplinary Research and Consulting

Department of Mathematics and Statistics

University of Maryland, Baltimore County

www.umbc.edu/circ

Winter 2008

Mission and Goals: The Center for Interdisciplinary Research and Consulting (CIRC) is a consulting service on mathematics and statistics provided by the Department of Mathematics and Statistics at UMBC. Established in 2003, CIRC is dedicated to support interdisciplinary research for the UMBC campus community and the public at large. We provide a full range of consulting services from free initial consulting to long term support for research programs.

CIRC offers mathematical and statistical expertise in broad areas of applications, including biological sciences, engineering, and the social sciences. On the mathematics side, particular strengths include techniques of parallel computing and assistance with software packages such as MATLAB and COMSOL Multiphysics (formerly known as FEMLAB). On the statistics side, areas of particular strength include Toxicology, Industrial Hygiene, Bioequivalence, Biomechanical Engineering, Environmental Science, Finance, Information Theory, and packages such as SAS, SPSS, and S-Plus.

Copyright © 2003–2008 by the Center for Interdisciplinary Research and Consulting, Department of Mathematics and Statistics, University of Maryland, Baltimore County. All Rights Reserved.

This tutorial is provided as a service of CIRC to the community for personal uses only. Any use beyond this is only acceptable with prior permission from CIRC.

This document is under constant development and will periodically be updated. Standard disclaimers apply.

Acknowledgements: We gratefully acknowledge the efforts of the CIRC research assistants and students in Math/Stat 750 Introduction to Interdisciplinary Consulting in developing this tutorial series.

MATLAB is a registered trademark of The MathWorks, Inc., `www.mathworks.com`.

1 Introduction

In this tutorial, we will introduce some of the numerical methods available in Matlab. Our goal is to provide some snap-shots of the wide variety of computational tools that Matlab provides. We will look at some optimization routines, where we mainly focus on unconstrained optimization.Next, we discuss curve fitting and approximation of functions using Matlab. Our final topic will be numerical ODEs in Matlab.

Matlab provides a number of specialized toolboxes, which extend the capabilities of the software. We will have a brief overview of the various toolboxes in Matlab and will provide a list of some available toolboxes.

- Numerical Optimization
- Data Fitting / Approximation
- Numerical ODEs
- Matlab Toolboxes

2 Unconstrained Optimization

The commands we discuss in this section are two of the several optimization routines available in Matlab. First we discuss `fminbnd`, which is used to minimize functions of one variable. The command,

```
[x fval] = fminbnd(f, a, b)
```

finds a local minimizer of the function `f` in the interval `[a, b]`. Here `x` is the local minimizer found by the command and `fval` is the value of the function `f` at that point. For complete discussion of `fminbnd` readers can refer to the Matlab's documentations. Here we illustrate the use of `fminbnd` in an example.

Consider the function $f(x) = \cos(x) - 2\ln(x)$ on the interval, $[\frac{\pi}{2}, 4\pi]$. The plot of the function is depicted in Figure 1. We can first define the function $f(x)$ in Matlab using

```
f = @(x)(cos(x) - 2*log(x))
```

Then we proceed by the following,

```
>> [x fval] = fminbnd(f, 2, 4)

x =

    3.7108
```

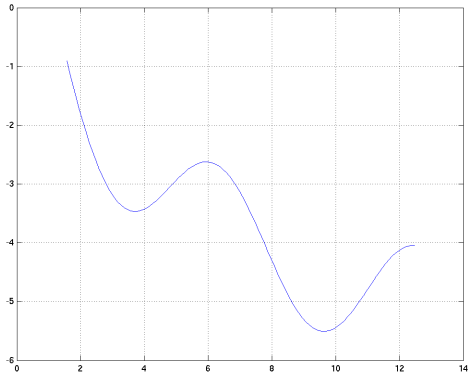

Figure 1: Plot of $f(x) = \cos(x) - 2\ln(x)$

.

```
fval =

   -3.4648
```

We see that `fminbnd` returns the (approximate) minimizer of $f(x)$ in the interval $[2, 4]$ and also computes the value of $f(x)$ at the minimizer. The readers can try to find the (global) minimizer of $f(x)$ which is seen to be somewhere in $[8, 10]$ using `fminbnd`.

The next (unconstrained) optimization command we discuss is `fminsearch` which can be used to find a local minimizer of a function of several variables. The command,

```
[x fval] = fminsearch(f,x0)
```

finds a local minimizer of the function `f` given the initial guess `x0`. For complete discussion of `fminsearch` readers can refer to the Matlab's documentations. Here we illustrate the use of `fminsearch` in an example.

For a simple example, we minimize the function $f(x, y) = x^2 + y^2$ which clearly has its minimizer at $(0, 0)$. Let's choose an initial guess of $x_0 = (0, 0)$. The following Matlab commands illustrate the usage of `fminsearch`

```
>> f = @(x)(x(1)^2+x(2)^2);
>> x0 = [1;1];
>> [x fval] = fminsearch(f, x0)

x =

   1.0e-04 *

   -0.2102
```

```
    0.2548


fval =

   1.0915e-09
```

Of course, reader can try `fminsearch` on more complicated problems and see the results.

For more information on optimization routines in Matlab, reader can investigate Matlab's Optimization Toolbox which includes several powerful optimization tools which can be used to solve both unconstrained and constrained linear and non-linear optimization problems.

3 Curve-Fitting

Here we consider the problem of fitting a polynomial of degree (at most) k into the data points (x_i, y_i); to do this, we use the command,

```
p = polyfit(x, y, k)
```

which fits a polynomial of degree k into the given data points. The output argument `p` is a vector which contains the coefficients of the interpolating polynomial. The method used is least squares, in which we choose a polynomial P of degree k which minimizes the following:

$$\sum_{i=1}^{m} [P(x_i) - y_i]^2. \tag{3.1}$$

For example, say we are given the data points

```
xi   yi
--------
1    1
2    0.5
3    1
4    2.5
5    3
6    4
7    5
```

Suppose we would like to fit a polynomial of degree three into the given data points. We can use the following Matlab commands to get the interpolating polynomial.

```
>> xi = 1 : 7;
>> yi = [1 0.5 1 2.5 3 4 5];
>> p = polyfit(xi,yi,3);
```

Once we have the vector `p` which contains the coefficients of the interpolating polynomial we can use the Matlab function `ployval` to evaluate the approximating polynomial over a given range of x values. We can proceed as follows:

```
>> x = 1 :0.1: 7;
>> y = polyval(p, x);
>> plot(xi, yi, '*', x, y)
```

Which produces the Figure 2.

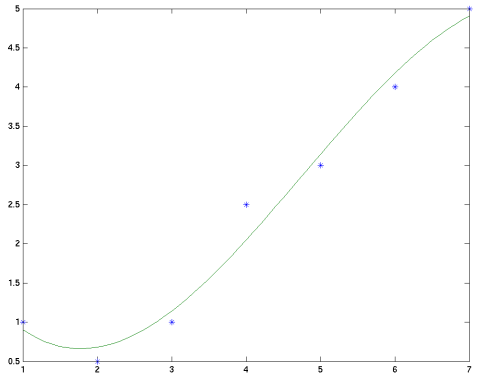

Figure 2: Data fitting in Matlab

Another useful idea is using `polyfit` to find an approximating polynomial for a given function. The idea is useful because polynomials are much simpler to work with. For example one can easily integrate or differential polynomials while it may not be so easy to do the same for a function which is not so well behaved. As an example, we consider the problem of approximating the function $\sin(\sqrt(x))$ on the interval $[0, 2\pi]$. The following Matlab commands show how one selects a numerical grid to get data points which can be used to approximate the function using a polynomial of degree k (in a least-squares sense).

```
>> f = @(x)(sin(sqrt(x)));
>> xi = [0 : 0.1 : 2*pi];
>> yi = f(xi);
>> p = polyfit(xi,yi,4);
```

To see how the function and its approximation compare, we can use the following commands,

```
>> x = [0 : 0.01 : 2*pi];
>> y = polyval(p, x);
>> plot(x, f(x), x, y, '--');
```

The result can be seen in Figure 3.

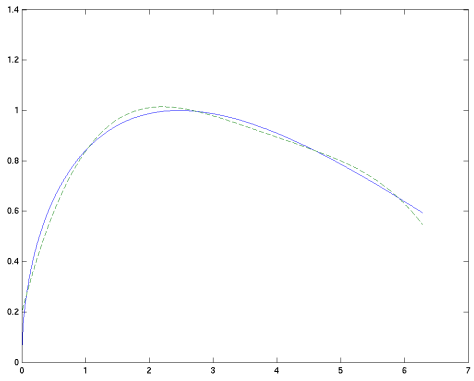

Figure 3: Approximation of a function using polynomial data fitting