Math 240: Homework 2

Jay Grinols

January 28, 2019

7.7

In 7.7, we want to employ various methods to find the maximums of f(x), which is defined as:

$$f(x) = 4x - 1.8x^2 + 1.2x^3 - 0.3x^4 \tag{1}$$

a)

First, we want to use the golden-section search algorithm, with $x_l = -2$, $x_u = 4$, and $\varepsilon_s = 1\%$. The provided golden-section search algorithm:

```
function [x,fx,ea,iter]=goldmin(f,xl,xu,es,maxit,varargin)
% goldmin: minimization golden section search
%   [x,fx,ea,iter]=goldmin(f,xl,xu,es,maxit,p1,p2,...):
%     uses golden section search to find the minimum of f
% input:
%   f = name of function
%   xl, xu = lower and upper guesses
%   es = desired relative error (default = 0.0001%)
%   maxit = maximum allowable iterations (default = 50)
%   p1,p2,... = additional parameters used by f
% output:
%   x = location of minimum
%   fx = minimum function value
%   ea = approximate relative error (%)
%   iter = number of iterations
if nargin<3,error('at least 3 input arguments required'),end
if nargin<4|isempty(es), es=0.0001;end
if nargin<5|isempty(maxit), maxit=50;end
phi=(1+sqrt(5))/2;
iter=0;
while(1)
  d = (phi-1)*(xu - xl);
  x1 = xl + d;
  x2 = xu - d;
  if f(x1,varargin{:}) < f(x2,varargin{:})
```

```
    xopt = x1;
    xl = x2;
  else
    xopt = x2;
    xu = x1;
  end
  iter=iter+1;
  if xopt~=0, ea = (2 - phi) * abs((xu - xl) / xopt) * 100;end
  if ea <= es | iter >= maxit,break,end
end
x=xopt;fx=f(xopt,varargin{:});
```

Using `7-7.m`, the output for a) was $x_a = 2.3282$, with $f_a = -5.8853$.

b)

Second, we want to use the parabolic interpolation algorithm, with $x_1 = 1.75$, $x_2 = 2$ and $x_3 = 2.5$, with the maximum number of iterations as 5. The provided parabolic interpolation algorithm:

```
function [x,fx,ea,iter]=parabmin2(g,xl,xr,xu,es,maxit,varargin)

% parabmin: minimization by parabolic interpolation
% adapted from Capra's goldmin script
%    finds the minimum of approximating parabolas to find min of f
% input:
%  f = name of function
%  xl, xu = lower and upper bracket, which should contain exactly
%  one local minimum and no local maxima.
% xr is any point inside these brackets
%  es = desired relative error (default = 0.0001%)
%  maxit = maximum allowable iterations (default = 50)
%  p1,p2,... = additional parameters used by f
% output:
%  x = location of minimum
%  fx = minimum function value
%  ea = approximate relative error (%)
%  iter = number of iterations

% In addition to the error check below inside the loop we should
% (but did not) include two more checks:

% (A) Is y(4) really a new value? Look at y(4)==y(1) || y(4)==y(2) ||
    y(4)==y(3)

% Also should include a check for a zero in the denominator of y(4).

% (B) If ( f(y(2))-f(y(3)) )/(y(2)-y(3))==( f(y(2))-f(y(1)) )/(y(2)-y(1))
```

```
% then the three points used to create y(4) are all on
% a line and this function fails.

% As written, this function is sloppy with function calls,
% including MANY function calls each iteration, and all
% but ONE of them are repeated calculations. These values should be
    placed
% in a vector at first use and NEVER recalculated.

if nargin<4,error('at least 4 input arguments required'),end
if nargin<5|isempty(es), es=0.0001;end
if nargin<6|isempty(maxit), maxit=50;end

f=@(x) g(x,varargin{:});

y(1)=xl; y(2)=xr; y(3)=xu;

iter=0;
Z2=y(1);

while(1)

if   f(y(2))>max( [f(y(1)) f(y(3))] ),error('The function has a local
    maximum on the interval between %8.4f and %8.4f. Choose another
    interval.', y(1), y(3) ),end

Z1=Z2;

y(4)=y(2)-( (y(2)-y(1))^2*( f(y(2))-f(y(3)) )-(y(2)-y(3))^2*(
    f(y(2))-f(y(1)) ) )/( 2*( (y(2)-y(1))*( f(y(2))-f(y(3))
    )-(y(2)-y(3))*( f(y(2))-f(y(1)) )));

Z2=y(4);

y=sort(y);

  if f(y(1))==min(f(y)) || f(y(2)) == min(f(y))
   y=[y(1) y(2) y(3)];
  else
  y=[y(2) y(3) y(4)];
  end

  iter=iter+1;

  if Z2~=0, ea = abs((Z2-Z1)/Z2) * 100;end
  if ea <= es | iter >= maxit,break,end

end

x=Z2;fx=f(Z2);
```

Using part b) of `7-7.m`, the output was: $x_b = 2.3264$ and $f_b = -5.8853$.

```
%7-7.m
f = @(x) -(4*x - 1.8*x.^2+1.2*x.^3 - 0.3*x.^4);

%a)Golden-section search
goldmin(f, -2, 4, 1)

%b)Parabolic Interpolation
parabmin2(f, 1.75, 2, 2.5, [], 5)
```

7.21

For 7.21, we are given that the trajectory of a ball can be computed with:

$$y = \tan(\theta_0)x - \frac{g}{2v_0^2\cos^2(\theta_0)}x^2 + y_0 \tag{2}$$

where y = the height (m), θ_0 = the initial angle (radians), v_0 = the initial velocity (m/s), $g = 9.81$ m/s^2, and y_0 = the initial height (m). We want to use golden-section search to determine the maximum height, given that $y_0 = 2$ m, $v_0 = 20$ m/s, and $\theta_0 = 45°$, using the criteria that $\varepsilon_s = 10\%$, $x_l = 10$ m and $x_u = 30$ m.

```
%7-21.m
y = @(x, theta, vinit, g, yinit) tan(theta).*x -
    g./(2.*vinit.^2.*(cos(theta)).^2).*x.^2 + yinit;
f = @(x) -y(x, 45*pi/180, 20, 9.81, 2);
x = goldmin(f,10, 30, 10)
-f(x)
```

The output for `7-21.m` is $y_{max} = 12.193$ meters, found when $x_{ymax} = 20.557$ meters.

7.27

For 7.27, we are given that a multivariable function f is defined as:

$$f(x, y) = 2y^2 - 2.25xy - 1.75y + 1.5x^2 \tag{3}$$

We want to find the minimum of f using various ways.

a)

First, we want to take a graphical approach. The plot, shown in Figure 1 shows that the minimum is located at around (3, -0.5, -17).

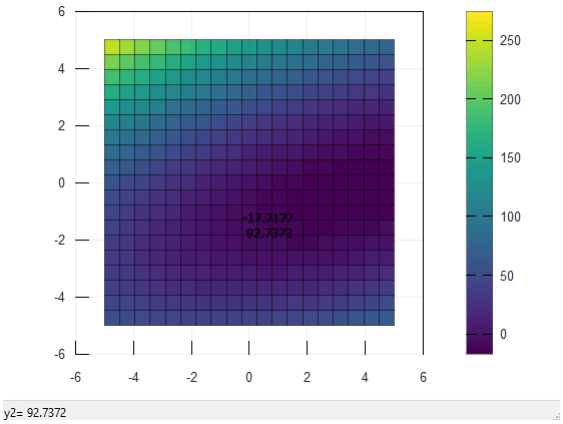

Figure 1: Plot for 7-27.

b)

Second, we want to use the built-in `fminsearch` function. The output using `7-27.m` was $(x, y) = (3.33339, -0.6664)$, with $z = -17.333$.

c)

Finally, we want to substitute the result of **b)** back into the function to determine the minimum of f. The output I got was $z_{min} = -17.333$, which seems consistent with both parts a) and b).

```
%7-27.m
f = @(x, y) -8*x + x.^2 + 12*y + 4*y.^2 - 2*x.*y;
%a) graphical
[X, Y] = meshgrid(linspace(-5, 5, 20), linspace(-5, 5, 20));
Z = f(X, Y);
surf(X, Y, Z)
colorbar
pause

%b) fminsearch
fnew = @(x) f(x(1), x(2));
[x,z] = fminsearch(fnew, [0,0])
%c) substitute b) into a)
fnew(x)
```

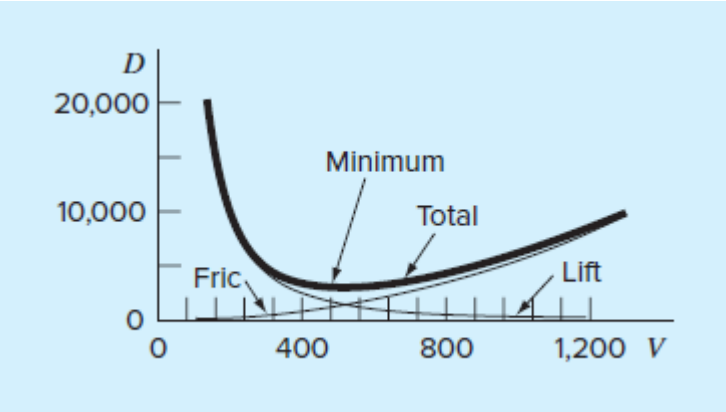

Figure 2: Graph for 7-35.

7.35

For 7.35, we are given that the total drag on an airfoil can be estimated by:

$$D = 0.01\sigma V^2 + \frac{0.95}{\sigma}\left(\frac{W}{V}\right)^2 \tag{4}$$

where D = drag, σ = ratio of air density between the flight altitude and sea level, W = weight and V = velocity. D can be split into the friction ($0.01\sigma V^2$) and the lift ($\frac{0.95}{\sigma}\left(\frac{W}{V}\right)^2$) terms. As seen in Figure 2, these two factors that contribute to drag are affected differently as the velocity increases. While the friction drag increases with velocity, the drag due to lift decreases as the velocity increases. The combination of these two factors leads to a minimum drag.

a)

For part a), we want to find the minimum drag and the velocity at which it occurs if $\sigma = 0.6$ and $W = 16,000$. Using `7-35.m`, I found the output to be $v_{optimal} = 509.82$ m/s and $D_{min} = 3119.0$ N.

b)

For part b), we want to develop a "sensitivity analysis" to determine how this optimum varies in response to a range of W = [12,000, 20,000] with $\sigma = 0.6$. This can be found in Figure 3. As we can see, the drag is somewhat proportional to the weight at this "optimal" value for velocity.

```
%7-35.m
D = @(sig, W, V) 0.01*sig.*V.^2 + 0.95./sig.*(W./V).^2;
```

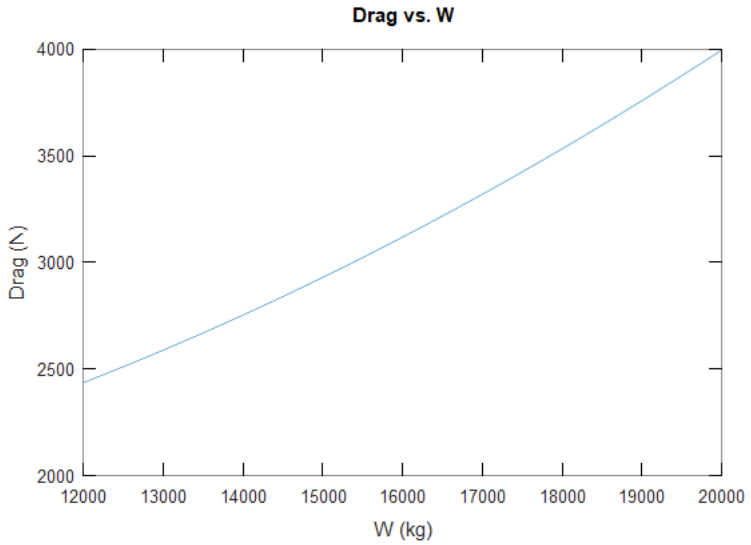

Figure 3: Sensitivity Analysis for 7-35.

```
%a)
Dnew = @(V) D(0.6, 16000, V);
[voptimal, mindrag] = fminsearch(Dnew, 10)

%b)
wrange = linspace(12000, 20000, 1000);
Dnewb = @(W) D(0.6, W, voptimal)
plot(wrange, Dnewb(wrange))
pause
```

8.6

For 8.6, we want to create a function file that handles matrix multiplication, without using any of the built in matrix operations. `matrixmult.m` is my function file that handles this.

```
%matrixmult.m
function out = matrixmult(A, B)
[mA, nA] = size(A);
[mB, nB] = size(B);
if nA ~= mB
   error('Number of A columns != Number of B rows, matrices are
       incompatible.')
end
```

```
out = zeros(mA, nB);  %initialization of output matrix
for i = 1:size(out)(1)
   for j = 1:size(out)(2)
      val = 0;
      for k = 1:nA  %dot product operation between row of A and column
          of B.
         val = val + A(i, k)*B(k, j);
      end
      out(i, j) = val;  %Set output value for location i, j
   end
end
end
```

8-6.m shows some tests of `matrixmult.m`.

```
%8-6.m
A = [6 -1; 12 8; -5 4;];
B = [4 0;0.5 2;];
C = [2 -2; 3 1;];
%unit tests
isequal(A*B, matrixmult(A, B))
isequal(B*C, matrixmult(B,C))
isequal(C*B, matrixmult(C,B))
```

The output for the 3 tests were all true, which indicates that our function seems to be working.

8.11

Here, we are considering a three mass, four spring system, shown in Figure 4. Using Newton's second law, $\Sigma F_x = ma_x$, we can produce the following

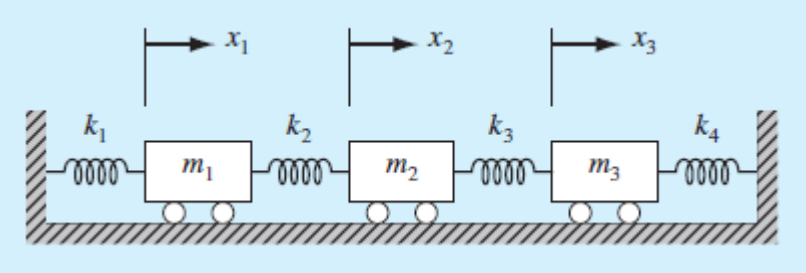

Figure 4: Diagram for 8-11.

differential equations:

$$\ddot{x_1} + \left(\frac{k_1 + k_2}{m_1}\right) x_1 - \left(\frac{k_2}{m_1}\right) x_2 = 0 \tag{5}$$

$$\ddot{x_2} - \left(\frac{k_2}{m_2}\right) x_1 + \left(\frac{k_2 + k_3}{m_2}\right) x_2 - \left(\frac{k_3}{m_2}\right) x_3 = 0 \tag{6}$$

$$\ddot{x_3} - \left(\frac{k_3}{m_3}\right) x_2 + \left(\frac{k_3 + k_4}{m_3}\right) x_3 = 0 \tag{7}$$

where $k_1 = k_4 = 10$ N/m, $k_2 = k_3 = 30$ N/m, and $m_1 = m_2 = m_3 = 1$ kg. These equations can be written in matrix form:

$$\begin{bmatrix} \ddot{x_1} \\ \ddot{x_2} \\ \ddot{x_3} \end{bmatrix} = \begin{bmatrix} -\left(\frac{k_1+k_2}{m_1}\right) & \left(\frac{k_2}{m_1}\right) & 0 \\ \left(\frac{k_2}{m_2}\right) & -\left(\frac{k_2+k_3}{m_2}\right) & \left(\frac{k_3}{m_2}\right) \\ 0 & \left(\frac{k_3}{m_3}\right) & -\left(\frac{k_3+k_4}{m_3}\right) \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{8}$$

$$\ddot{\vec{x}} = A \cdot \vec{x} \tag{9}$$

Given this information, we want to use Octave to solve for the acceleration $\ddot{\vec{x}}$ when $\vec{x} = [0.05\ 0.04\ 0.03]^T$ m.

```
%8-11.m
[k1, k2, k3, k4] = deal(10, 30, 30, 10);
[m1, m2, m3] = deal(1, 1, 1);
[x1, x2, x3] = deal(0.05, 0.04, 0.03);
x = [x1; x2; x3];
km = [-(k1 + k2)/m1, k2/m1, 0; k2/m2, -(k2+k3)/m2, k3/m2; 0, k3/m3, -(k3
    + k4)/m3;];
km*x
```

The output for the acceleration vector, which is found by the last line of `8-11.m`, was $\ddot{\vec{x}} = [-8,\ 0,\ 0]^T$ m/s.

9.1

For 9.1, we want to determine the total number of FLOPs necessary as a function of the n, the total number of equations of a tridiagonal matrix. The provided tridiagonal algorithm:

```
%Tridiag.m
function x = Tridiag (e,f,g,r)
% Tridiag: Tridiagonal equation solver banded system
% x = Tridiag (e,f,g,r): Tridiagonal system solver.
% input:
% e = subdiagonal vector
% f = diagonal vector
```

```
% g = superdiagonal vector
% r = right hand side vector
% output:
% x = solution vector
n = length(f);
% forward elimination
for k = 2:n
factor = e(k)/f(k-1);
f(k) = f(k) - factor*g(k - 1);
r(k) = r(k) - factor*r(k - 1);
end
% back substitution
x(n) = r(n)/f(n);
for k = n-1:-1:1
x(k) = (r(k) - g(k)*x(k+1))/f(k);
end
```

From forward elimination, we run a loop with $(n-1)$ iterations, with 5 FLOPs within the loop. From back substitution, we have one FLOP outside of the loop, with $(n-1)$ iterations, with 3 FLOPs within the loop. $5(n-1)+3(n-1)+1 = 8(n-1)+1$ total FLOPs.

9.10

Here, we have three different pits with varying amount of sand, fine gravel and coarse gravel each, which we want to utilize for a building project. We want to obtain 4800, 5800, and 5700 m^3 of sand, fine gravel and coarse gravel respectively. The composition of these three pits are listed in the table.

	Sand (%)	Fine Gravel (%)	Coarse Gravel (%)
Pit 1	55	30	15
Pit 2	25	45	30
Pit 3	25	20	55

Assuming a uniform distribution in each pit, we want to find the number of cubic meters required from each pit to meet the needs of this project.

We can model this problem as a system of linear equations.

$$\begin{bmatrix} p_1 & p_2 & p_3 \end{bmatrix} \cdot \begin{bmatrix} 0.55 & 0.30 & 0.15 \\ 0.25 & 0.45 & 0.30 \\ 0.25 & 0.20 & 0.55 \end{bmatrix} = \begin{bmatrix} 4800 & 5800 & 5700 \end{bmatrix} \tag{10}$$

$$\begin{bmatrix} 0.55 & 0.30 & 0.15 \\ 0.25 & 0.45 & 0.30 \\ 0.25 & 0.20 & 0.55 \end{bmatrix}^T \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} 4800 \\ 5800 \\ 5700 \end{bmatrix} \tag{11}$$

```
%9-10.m
```

```
comp = [55, 30, 15; 25, 45, 30; 25, 20, 55;];
comp = (comp / 100)';
need = [4800, 5800, 5700]';
answer = comp\need
```

Running the script, we find that answer = [5515, 5190, 5595]^T, which corresponds to 2416.7 m^3 from pit 1, 9193.3 m^3 from pit 2, and 4690.0 m^3 from pit 3.