Contents

Answers to Exercises: week 1.

```
clear all;
```

Exercise set 1

```
% part a
clc; format;

2/3
2^3
2^(1/3)
log2(3)
cos(3)*sin(3)
sin(2+3i)
sqrt(-1i)
```

```
ans =

    0.6667


ans =

     8


ans =

    1.2599


ans =

    1.5850


ans =

   -0.1397


ans =

   9.1545 - 4.1689i


ans =

   0.7071 - 0.7071i
```

part b

need 2*x and tan(x)

e not a predefined constant; use exp(x)

variable name cannot start with numeral

Exercise Set 2

part a

```
type newtonForTwoOutputs
[root, resid] = newtonForTwoOutputs(2)
```

```
function [root5, residual] = newtonForTwoOutputs(x0)
% A function to compute the square root of 5
% Uses Newton's method
% Now with a for loop
% Input:    x0          the initial guess
% Output:   root5       the fourth iterate
%           residual    root5^2-5
% Usage:    y = newtonForTwoOutputs(2)
%
if x0 == 0
    disp('Error: zero not allowed as input');
    root5 = nan;    % give return value
else
    xold = x0;      % initialize xold
    for iteration = 1:4
        xnew = xold/2+5/(2*xold);
        xold = xnew;
    end
    root5 = xnew;
    residual = root5^2-5;
end

root =

    2.2361


resid =

   8.8818e-16
```

part b

```
type newtonForTwoInputs
root = newtonForTwoInputs(2,7)
root^2-7
```

```
function root = newtonForTwoInputs(x0, a)
% A function to compute the square root of a
% Uses Newton's method
% Now with a for loop
% Input:    x0          the initial guess
%           a           the number you are finding the square root of
% Output:   root       the fourth iterate
% Usage:    y = newtonForTwoInputs(2,7)
%
if x0 == 0
    disp('Error: zero not allowed as input');
    root = nan;    % give return value
else
    xold = x0;      % initialize xold
    for iteration = 1:4
        xnew = xold/2+a/(2*xold);
        xold = xnew;
    end
    root = xnew;
end

root =

    2.6458


ans =

   5.4357e-13
```

part c

```
type newtonIterationsInput
root = newtonIterationsInput(2,2)
root^2-5
root = newtonIterationsInput(2,5)
root^2-5
```

```
function root5 = newtonIterationsInput(x0, count)
% A function to compute the square root of 5
% Uses Newton's method
% Now with a for loop
% Input:    x0          the initial guess
%           count           the number of iterations
% Output:   root5       the result after count iterations
% Usage:    y = newtonIterationsInput(2,2)
%
if x0 == 0
    disp('Error: zero not allowed as input');
    root5 = nan;    % give return value
else
    xold = x0;      % initialize xold
    for iteration = 1:count
        xnew = xold/2+5/(2*xold);
        xold = xnew;
    end
    root5 = xnew;
end

root =

    2.2361


ans =

   1.9290e-04


root =

    2.2361


ans =

   8.8818e-16
```

part d

```
type newtonWhileLoop
[root,number]=newtonWhileLoop(2)
[root,number]=newtonWhileLoop(2.2)
```

```
function [root5,count] = newtonWhileLoop(x0)
% A function to compute the square root of 5
% Uses Newton's method
% Now with a while loop
% Input:    x0      the initial guess
% Output:   root5      the  estimate of the root
%           count      the number of iterations required
% Usage:    [root, iterations] = newtonWhileLoop(2)
%
if x0 == 0
    disp('Error: zero not allowed as input');
    root5 = nan;    % give return value
    return
else
    xold = x0;      % initialize xold
    count = 1;      % initialize count
    xnew = xold/2+5/(2*xold);   % do 1 iteration before test
    while abs(xnew^2-5) >= 1e-10
        xold = xnew;
        xnew = xold/2+5/(2*xold);
        count = count + 1;
    end
    root5 = xnew;
end
end

root =

    2.2361


number =

     4


root =

    2.2361


number =

     3
```

part e check the logic by stepping through the calculation in the debugger

Exercise Set 3

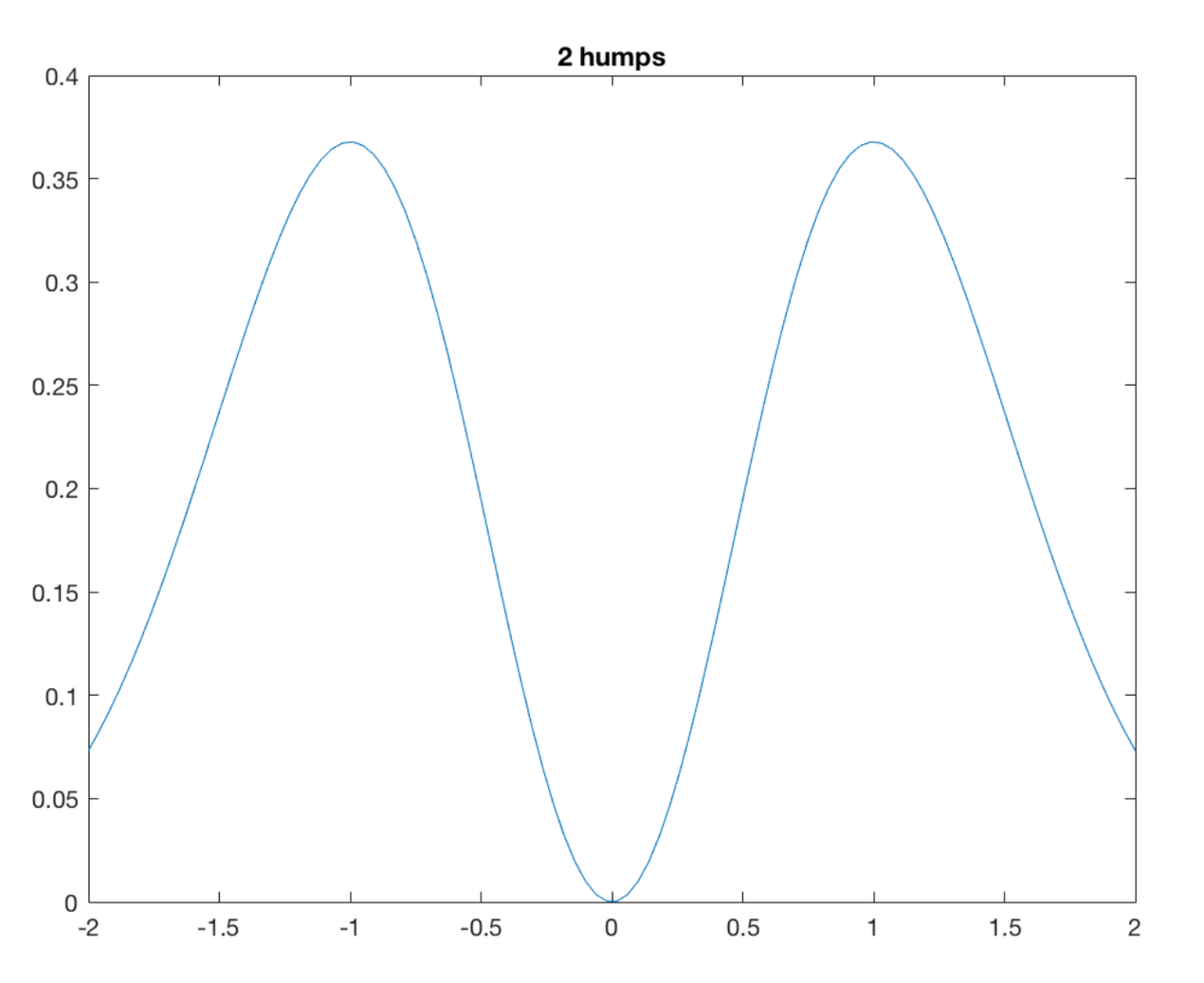
part a

```
type mydiff
x=(1:10).^2;
y = mydiff(x)
z = diff(x)
```

```
function diffx = mydiff( x )
%MYDIFF mimics diff for vector input
% Input     x            vector
% Output    y       same as diff(x)
diffx = x(2:end)-x(1:end-1);
end


y =

     3     5     7     9    11    13    15    17    19


z =

     3     5     7     9    11    13    15    17    19
```


part b

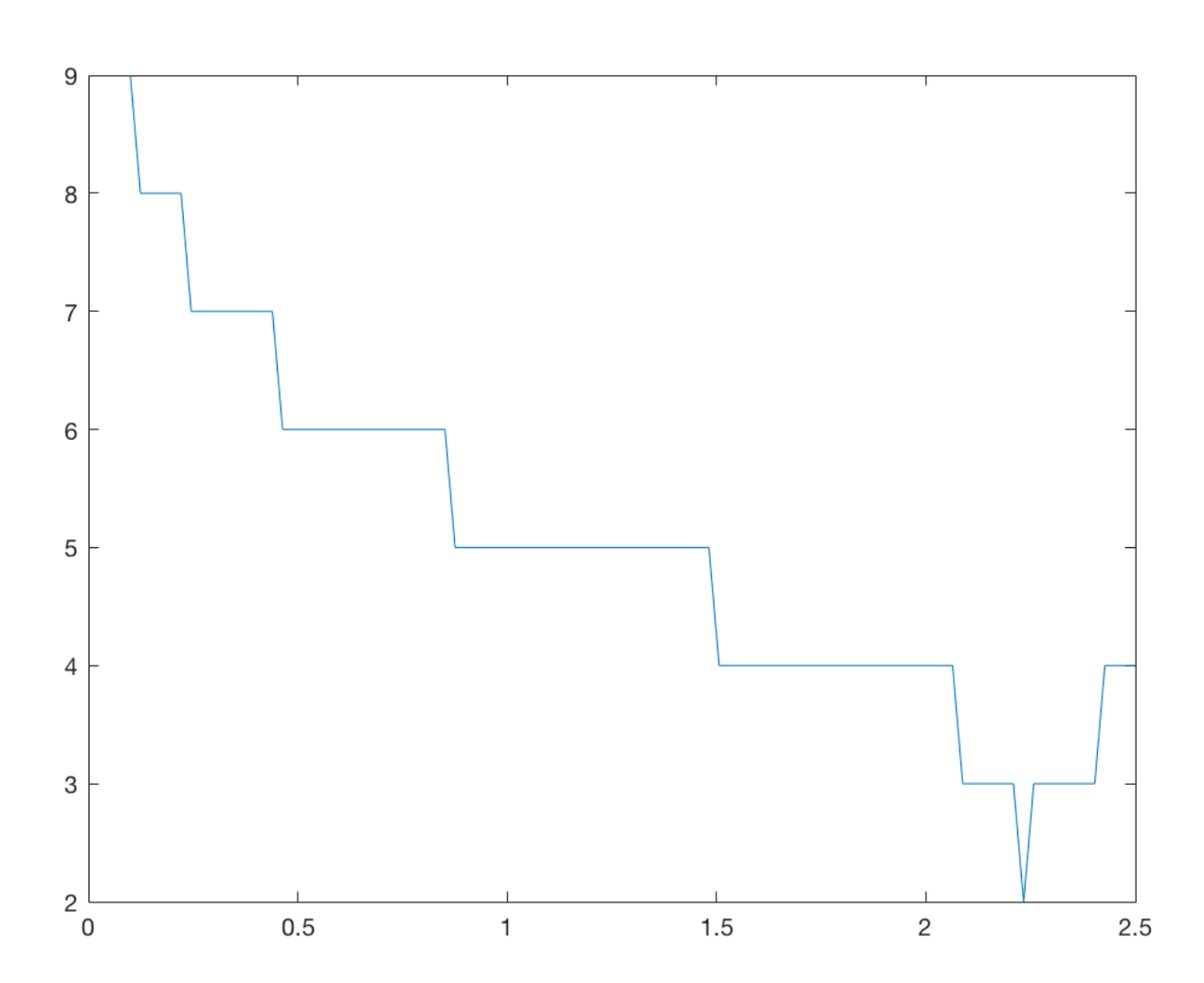
```
x=linspace(-2,2);
bactrian = x.^2.*exp(-x.^2);
plot(x,bactrian); title('2 humps');
```



part c

```
type Ex3c
Ex3c
```

```
function  Ex3c(  )
%EX4C A driver for newtonWhileLoop
%   Plots the number of iterations required for 10 figure accuracy

x0 = linspace(0.1,2.5);  % a range of initial values
count = 0;
its = zeros(1,length(x0));
for init = x0       % loop over initial values
    count = count+1;
    [~,its(count)] = newtonWhileLoop(init);
end
plot(x0,its);

end
```



Exercise Set 4

part a

```
fprintf('%2d',123)      % no newline so will print next character on same line
fprintf('%2d\n',123)    % needs 3 spaces so uses 3
fprintf('%8d\n',123)    % given 8 spaces so pads with 5 zeros
fprintf('%8d\n',123.456) % given a noninteger uses e format by default
fprintf('%8f\n',123.456)    % uses 6 d.p. as default
fprintf('%10.2f\n',123.456) % overides to 2 d.p.; . counts in field width
fprintf('%10.8e\n',123.456) % needs field width > 10 so uses 14
fprintf('%8d\n',1:4)     % if # outputs > # format specifiers, cycles thru
                         %  format specifiers
```

```
123123
     123
1.234560e+02
123.456000
    123.46
1.23456000e+02
       1
       2
       3
       4
```

part b

```
type newtonFormattedTable

sequence = newtonFormattedTable(2,4)
```

```
function iterates = NewtonFuncFormattedTable(x0,numIter)
% A function to compute the square root of 5
% Uses Newton's method
% Now with a for loop
% and a nice table of output
% Input:    x0      the initial guess
%           numIter the number of iterations
% Output:   iterates      vector:  the iterates
% Usage:    y = NewtonFuncFormattedTable(2,4)
%
if x0 == 0
    disp('Error: zero not allowed as input');
    iterates = nan;    % give return value
else
    fprintf('\n   n \t   iterate \t |xn-x_nm1|   \t residual\n');

    x(1) = x0;      % initialize xold
    for n = 1:numIter
        x(n+1) = x(n)/2+5/(2*x(n));
        fprintf('%2d \t %12.8f \t %12.6e \t %12.6e\n', ...
            n,x(n+1),abs(x(n+1)-x(n)),x(n+1)^2-5);
    end
    iterates = x;  %     the whole sequence
end

   n       iterate       |xn-x_nm1|      residual
 1         2.25000000    2.500000e-01    6.250000e-02
 2         2.23611111    1.388889e-02    1.929012e-04
 3         2.23606798    4.313320e-05    1.860471e-09
 4         2.23606798    4.160139e-10    8.881784e-16

sequence =

    2.0000    2.2500    2.2361    2.2361    2.2361
```

More

part a

```
type nfacIteration
x = nfacIteration(6)
x = nfacIteration(100)
% this version uses iteration
```

```
function nfact = nfacIteration( n )
% A function to compute the factorial of n
% Uses iteration
% Input:    n      integer n
% Output:   nfact      n!
% Usage:    y = nfacIteration(6)

if n < 0
    disp('Error: n must be non-negative');
    nfact = nan; % give return value
    return          % return control to calling program
elseif (n == 0) || (n == 1)
    nfact = 1;
else
    intermediate = 1;     % initialize intermediate variable
    for count = 1:n
        intermediate = intermediate*count;
    end
    nfact = intermediate;
end






x =

   720


x =

  9.3326e+157
```

```
type nfacRecursion
x = nfacRecursion(6)
x = nfacRecursion(100)
% this version uses recursion
```

```
function nfact = nfacRecursion( n )
% A function to compute the factorial of n
% Uses recursion
% Input:    n      integer n
% Output:   nfact      n!
% Usage:    y = nfacRecursion(6)

if n < 0
    disp('Error: n must be non-negative');
    nfact = nan; % give return value
    return          % return control to calling program
elseif (n == 0) || (n == 1) % base cases
    nfact = 1;
else
    nfact = n*nfacRecursion(n-1);   % a recursive call
end






x =

   720


x =

  9.3326e+157
```

part b

```
type summaryStats

type sumstatsDriver

sumstatsDriver
```

```
function [ mean,var, tmean ] = summaryStats( x, k )
%SummaryStats on Lab Sheet 1
%   Calculates some summary statistics of a vector x
%   Inputs: a vector x containing data
%           k  an integer , the amount of trimming
%   Outputs: mean   the sample mean of x
%            var   the sample variance of the data
%            tmean   the trimmed mean of the data
n = length(x);
if k<1 || k>= n/2
    disp('Invalid input: try again');
    mean = nan;
    return
end

mean = sum(x)/n;
var = sum((x-mean).^2)/(n-1);
sorted = sort(x);
tmean = sum(sorted(k+1:end-k))/(n-2*k);
end


%  sumstatsDriver

x = ones(1,10);
x(5)=100;
k = 2;

[mymean,myvar,tmean] = summaryStats(x,k);

stats = [mymean,myvar,tmean];

fprintf('%10.4f\n',stats);   % to illustrate a vector input

   10.9000
  980.1000
    1.0000
```

Published with MATLAB® R2016b