Module 1 Practice Problems (Solutions)

Loops, Arrays, and Vectorization

1. Only part (a) is explained, the others are done in the same fashion.

   (a) Let $x = 5.4$ and $s = \cos(x) = \cos(5.4)$. Denote by s_k the kth partial sum of the Taylor series given,

   $$s_k = \sum_{n=0}^{k} \frac{(-1)^n x^{2n}}{(2n)!} = \sum_{n=0}^{k} \frac{(-1)^n (5.4)^{2n}}{(2n)!},$$

   and so

   $$s = \lim_{k\to\infty} s_k.$$

   We calculate $s_0, s_1, \ldots, s_N$ using a for-loop:

```
N = 10; x = 5.4; s = cos(x);
sk = 0;
for n = 0:N
    sk = sk + (-1)^n * x^(2*n) / factorial(2*n);
    fprintf('%3d %22.16f %24.16e\n', n, sk, abs(sk-s))
end
```

2. Recall that a vector, by default, is a column vector in this class; this is why each of the following answers has the transpose `.'`. If a row vector is desired, it will be explicitly mentioned.

   - **LM** 3.1–3(b): Note that $b_k = (2k-1)^2$ for $k = 1, 2, \ldots, p$ where p is the number of elements of $\mathbf{b}$, which is to be determined. Since the elements are all required to be $\leqslant n^2$, it must be that p is the largest positive integer such that $2p - 1 \leqslant n$. This implies that $p = \lfloor (n+1)/2 \rfloor$ (floor function). However, as far as generating the vector on MATLAB is concerned, we can simply do

```
b = ([1:2:n].') .^ 2;
```

     without needing to type p out explicitly. (Think about why and make sure you understand why.)

   - **LM** 3.1–3(c): In this part, $c_k = (2k-1)^2$ for $k = 1, 2, \ldots, q$ where q is the number of elements of $\mathbf{c}$. The requirement is that the elements are $\leqslant n$ not n^2. Thus q is the largest integer such that $(2q-1)^2 \leqslant n$, *i.e.*, $q = \lfloor \left(\sqrt{n} + 1\right)/2 \rfloor$. Though the expression for q is messy, it can be compactly coded as

```
c = ([1:2:sqrt(n)].') .^ 2;
```

since the colon operator ensures that the last elements of `1:2:sqrt(n)` does not exceed $\sqrt{n}$.

- **LM** 3.1–3(e):

```
        e = [2:n^2, 999999].';
```

It is important that you do not omit `.'` at the end since $\mathbf{e} = (2, 3, \ldots, n^2, 999999)^{\mathrm{T}}$, a column vector!

- **LM** 3.1–3(g): All angles are in radians.

```
        g = [sin(1:n), cos(n:-1:1)].';
```

- **LM** 3.1–4(c): The terms can be viewed as

$$\begin{aligned}
t_1 &= 1^2 + 1 = 2 \\
t_2 &= 2^2 + 1 = 5 \\
t_3 &= 3^2 + 1 = 10 \\
&\vdots \\
t_j &= j^2 + 1, \quad \text{for } j \geqslant 1.
\end{aligned}$$

Therefore,

```
t = ([1:n].^n + 1).';
```

- **LM** 3.1–4(e): Note that the elements of $\mathbf{v}$ are powers of 2:

$$2^{-1}, 2^0, 2^1, \cdots, 2^{n-1}.$$

The last term is 2^{n-1} since n terms are needed. Hence,

```
v = 2 .^ ([-1:n-2].');
```

- **LM** 3.1–5(d):

```
d = ( 1 ./ sin(n:-1:1).^3 ).';
```

- **LM** 3.1–5(f):

```
f = factorial(1:n+1).';
```

- **LM** 3.1–16:

```
s = A(1,:) + A(2,:);         % sum of first two rows
A(2,:) * A(3,:)';            % inner product
B = A(:,3) * A(:,7)';        % outer product
C = A(4,:)' * A(:,9)';       % outer product
d = diag(A);                 % diag can take a rectangular matrix
e = [diag(A); zeros(10,1);   % vertical concatenation
```

- **LM** 3.2–7: As per the instruction found at the beginning of the exercises, we answer this question without using loops. For part (a), first make a copy of `A` and call it `B`. Then find where `B` is positive using the `find` function and change the corresponding elements of `B` to zeros.

```
B = A;
B(find(B<0)) = 0;
```

Alternately, logical arrays can be used instead of `find` as shown below.

```
B = A;
B(B<0) = 0;
```

Part (b) is done similarly.

```
C = A^2 - 100;
C(find(C<0)) = 0;
```

Note that A is a square matrix and the expression A^2 is equivalent to $A \cdot A$, the multiplication of two matrices learned in linear algebra, not an elementwise multiplication. For part (c), use the `max` function:

```
D = max( A-10, B-100 );
```

3. (a) Already given as a hint.

   (b) Similar to part (a).

   ```
   p = 1;
   for j = 1:n
       p = p * x(j);
   end
   fprintf('Result using ''prod'' function: %24.16f\n', prod(x));
   fprintf('Result using a ''for'' loop  : %24.16f\n', p);
   ```

   (c) See p. 91 of Module 1 lecture slides.

   (d) Same as above.

   (e) The given expression is the average or the *mean* of the elements of $\mathbf{x}$. It is computed using a loop similar to the one for part (a) because of the obvious reason. To vectorize, use can either do `sum(x)/n` or `mean(x)`.

   (f) See Problem 1.

   (g) Modify the previous part.

4. (a) $\mathbf{x}$ consists of 11 equispaced points on $[0, 2\pi]$, including both endpoints. So use `linspace`. Assume that `n` is already stored.

   ```
   x = linspace(0, 2*pi, n);
   y = sin(x);
   ```

   (b) Assume again that `n` is already stored. Using the `cumsum` function:

   ```
   y = cumsum(1:n);
   fprintf('Sum of integers 1 to %2d: %5d\n', [1:n; y]);
   ```

Graphics

1. One way is to let MATLAB determine colors by itself.

```
x = linspace(-1, 1, 51);
plot(x, sinh(x), x, cosh(x), x, tanh(x));
legend('sinh', 'cosh', 'tanh')
grid on    % not required
```

If you really want to be concise, the second line can be replaced by

```
plot(x, [sinh(x); cosh(x); tanh(x)]);
```

If you want to have a finer control over colors and specify them on your own, do

```
plot(x, sinh(x), 'r', x, cosh(x), 'g', x, tanh(x), 'b');
```

2. Plots for all three parts are drawn in a single figure window using `subplot`.

```
f = @(x) atan(x);                % or f = @atan;
g = @(x) nthroot(x, 3);          % g = @(x) x.^(1/3) not recommended
h = @(x) x.^3 + (5-x).^2 - 7;
x = -5:0.1:5;

clf
subplot(3,1,1)
plot(x, [f(x); f(x/10); f(10*x)])
title('(a)')

subplot(3,1,2)
plot(x, g(f(x)))
title('(b)')

subplot(3,1,3)
plot(x, g(x) .* f(10*h(x)))
title('(c)')
```

Note that defining g by `g = @(x) x.^(1/3)` will yield unexpected results since MATLAB evaluates $g(x)$ as complex numbers when $x < 0$; see Lecture 1 around 29:30 mark.

3. Since we are interested in the convergence behavior of a sequence, it is suitable to plot the errors $e - r_n$ against n using dots rather than connecting the data points with line segments.

```
e = exp(1);                             % the Euler number
n = 5:5:500;
r_n = (1 + 1./n).^n;
e_n = abs(e - r_n);                     % absolute error

clf
subplot(2,1,1)
plot(n, e_n, '.')
grid on
xlabel('n'), ylabel('|e -r_n|')
title('|e - r_n|: Standard plot')
```

```
subplot(2,1,2)
loglog(n, e_n, '.')
grid on
xlabel('n'), ylabel('|e -r_n|')
title('|e - r_n|: Log-log plot')
```

The log-log plot looks linear with a negative slope, more and more so as n gets larger. This means that the log of the dependent and independent variables are linearly related, that is,

$$\log |e - r_n| \approx -\alpha \log n + \beta, \quad \text{for some } \alpha, \beta > 0.$$

Exponentiating both sides and arranging the terms, we find that

$$|e - r_n| \approx \frac{e^\beta}{n^\alpha}.$$

This suggests that the absolute error $|e - r_n|$ decays like $1/n^\alpha$ for large n. Later in the course, we will learn the jargon which describes such asymptotic behavior.

4. The matrices x and y created by the given script are

```
x =
     0     1     2     3     4     5     6     7
     1     2     3     4     5     6     7     8
y =
     0     0     0     0     0     0     0     0
     1     1     1     1     1     1     1     1
```

The statement plot(x,y) is equivalent to plotting each column of y against the corresponding column of x. That is,

```
hold on
for j = 1:size(x,2)
    plot(x(:,j), y(:,j),'b')
end
axis equal
```

On the other hand, x(:) and y(:) are column vectors

```
x(:)'
ans =
     0   1   1   2   2   3   3   4   4   5   5   6   6   7   7   8
y(:)'
ans =
     0   1   0   1   0   1   0   1   0   1   0   1   0   1   0   1
```

and plot(x(:), y(:)) draws vertical line segments which were not found in the other figure.

5. Following the instruction and the hint provided,

```
subplot(1,2,1)
for i = 1:100                           % repeat below 100 times
    E = eig(randn(100));                % eigenvalues
    plot(E, 'b.')
    hold on
end
axis equal, title('(a)')

subplot(1,2,2)
for i = 1:100
    E = eig(complex(randn(100), randn(100)));
    plot(E, 'b.')
    hold on
end
axis equal, title('(b)')
```

6. The general strategy is

   - Define $f(x, y)$ as an anonymous function using elementwise operations when appropriate.
   - Create sample points on the given 2-D domain using the `meshgrid` function.
   - Use the `surf` function to plot the surface.

   Only part (a) is done; the rest are done in the same fashion.

```
f = @(x,y) (x.^2 + 3*y.^2) .* exp(-x.^2-y.^2);
x = linspace(-3, 3, 61);
y = linspace(-3, 3, 61);
[X,Y] = meshgrid(x,y);
surf(X, Y, f(X,Y))
```

7. *Hint.* Study the code on p. 128 of Module 1 lecture slides. The variables `th` and `ph` in the code play the roles of u and v here, respectively.