

## Module 1 Practice Problems

This problem set consists of three sections:

- The first section contains problems on loops, arrays, and vectorization techniques.
- The second section is all about drawing 2-D or 3-D graphics.
- Additional practice problems in the last section resemble exam problems.

### Loops, Arrays, and Vectorization

1. Using a for-loop, demonstrate the convergence of the series expansions of the following functions. Evaluate each function at the indicated value.

(a)  $\cos(x) = \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n}}{(2n)!}$  at  $x = 5.4$ .

(b)  $\exp(x) = \sum_{n=0}^{\infty} \frac{x^n}{n!}$  at  $x = 0.2$ .

(c)  $\ln(1+x) = \sum_{n=0}^{\infty} \frac{(-1)^{n-1} x^n}{n}$  at  $x = -0.5$ .

2. Do the following problems on construction and manipulation of 1-D and 2-D arrays.

- LM 3.1–3(b,c,e,g)
- LM 3.1–4(c,e)
- LM 3.1–5(d,f)
- LM 3.1–16.
- LM 3.2–7.

3. Compute the value of each mathematical expression by two different ways: using a single MATLAB statement and using a for-loop. Confirm that two methods yield the same result for each case. To begin, let  $n = 2021$  and create a (random) vector  $\mathbf{x} \in \mathbb{R}^n$  by

```
n = 2021; x = rand(n, 1);
```

Use this array for each of the following parts.

(a)  $\sum_{k=1}^n x_k$

(b)  $\prod_{k=1}^n x_k$

(c)  $\min_{1 \leq k \leq n} x_k$

(d)  $\max_{1 \leq k \leq n} x_k$

(e)  $\frac{1}{n} \sum_{k=1}^n x_k$

(f)  $\mathbf{s} \in \mathbb{R}^n$  where  $s_j = \sum_{k=1}^j x_k$ , for  $j = 1, 2, \dots, n$

(g)  $\mathbf{p} \in \mathbb{R}^n$  where  $p_j = \prod_{k=1}^j x_k$ , for  $j = 1, 2, \dots, n$

*Hint.* An example answer to part (a):

```
n = 2021; x = rand(n, 1);
s = 0;
for j = 1:n
    s = s + x(j);
end
fprintf('Result using ''sum'' function: %24.16f\n', sum(x));
fprintf('Result using a ''for'' loop : %24.16f\n', s);
```

4. In the following example, a for-loop is replaced by a simpler vectorized code. Note that print statement (using fprintf) is readily vectorized as well.

```
% Example: Calculate y = 10*x + 1 where x is a random vector
% for-loop
x = rand(1, 5);
y = zeros(size(x)); % preallocation
for i = 1:5
    y(i) = 10*x(i) + 1;
    fprintf('x(%d) = %8.4f; y(%d) = %8.4f\n', i, x(i), i, y(i));
end

% clear y
clear y

% vectorized equivalent
y = 10*x + 1;
fprintf('x(%d) = %8.4f; y(%d) = %8.4f\n', [1:5; x; 1:5; y]);
```

In a similar fashion, replace the following for-loops with vectorized statements.

- (a) (Evaluation on equispaced points)  $\mathbf{y} = \sin \mathbf{x}$ , where  $\mathbf{x}$  is the vector of  $n$  equispaced points on  $[0, 2\pi]$ .

```
n = 11;
for j = 1:n
    x(j) = 2*pi*(j-1)/n;
    y(j) = sin(x(j));
end
```

- (b) (Cumulative summation)  $\mathbf{y} = (y_k)$ , where  $y_k = \sum_{j=1}^k j$  for  $k = 1, 2, \dots, n$ .

```

n = 10;
s = 0;
y = zeros(1,n);
for k = 1:10
    s = s + k;
    y(k) = s;
    fprintf('Sum of integers 1 to %2d: %5d\n', k, y(k));
end

```

### Note: Loop v.s. Vectorization – Timing Comparison

Sometimes, but not always, a vectorized code will be much faster than an equivalent loop. For example:

```

N = 1e7;
theta = linspace(0, 2*pi, N);
nRepeat = 5;

% loop
tic
for j = 1:nRepeat
    for i = 1:N
        y(i) = sin(theta(i));
    end
end
t1 = toc/nRepeat;

% vectorized
clear y

tic
for j = 1:nRepeat
    y = sin(theta);
end
t2 = toc/nRepeat;

fprintf('Time in loop           : %8.4f\n', t1);
fprintf('Time in vectorized code : %8.4f\n', t2);
fprintf('Vectorized code is %6.2f times faster.\n', t1/t2);

```

The result:

```

Time in loop           : 0.2045
Time in vectorized code : 0.0536
Vectorized code is    3.82 times faster.

```

## Graphics

1. On a single graph, make a plot of the functions  $\sinh$ ,  $\cosh$ , and  $\tanh$  for  $-1 \leq x \leq 1$ . Give each curve a different color and add a legend.
2. Create anonymous functions for each of the following functions, using the "dot" operator in your function definition where necessary.

$$f(x) = \tan^{-1}(x), \quad g(x) = \sqrt[3]{x}, \quad h(x) = x^3 + (5 - x)^2 - 7.$$

Then, for each of the following parts, plot the requested expressions over the interval  $[-5, 5]$ .

- (a) Plot  $y = f(x)$ ,  $y = f(x/10)$ , and  $y = f(10x)$  on a single graph.
- (b) Plot  $y = g(f(x))$ .
- (c) Plot  $y = g(x)f(10h(x))$ .

3. Recall the identity

$$e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n.$$

Make a standard and a log-log plot<sup>1</sup> of  $e - r_n$  for  $n = 5, 10, 15, \dots, 500$ . What does the log-log plot reveal about the asymptotic behavior of  $e - r_n$  as  $n \rightarrow \infty$ ?

4. Here are two different ways of plotting a sawtooth wave. Study the code.

```
x = [0:7; 1:8];
y = [zeros(1,8); ones(1,8)];
subplot(121)
plot(x, y, 'b'), axis equal
subplot(122)
plot(x(:), y(:), 'b'), axis equal
```

5. In MATLAB, the eigenvalues of a square matrix  $A$  are computed using `eig` function; type `help eig` in the Command Window.
  - (a) Generate a hundred random matrices using `randn(100)`, and plot all of their eigenvalues as dots in the complex plane on one graph. (Thus, you should see  $100 \times 100 = 10,000$  dots.) Use `axis equal` to make the aspect ratio one-to-one. What do you observe?
  - (b) Repeat the experiment with a hundred random complex matrices generated by `complex(randn(100), randn(100))`. You should be able to see one very clear qualitative difference between the previous case and this one.

*Hint.* If  $z$  is complex, then `plot(z)` is equivalent to `plot(real(z), imag(z))`.

6. Make surface plots of the following functions over the given ranges:

(a)  $f(x, y) = (x^2 + 3y^2)e^{-x^2 - y^2}$ , for  $|x| \leq 3$ ,  $|y| \leq 3$ .

(b)  $f(x, y) = \frac{-3y}{x^2 + y^2 + 1}$ , for  $|x| \leq 2$ ,  $|y| \leq 4$ .

---

<sup>1</sup>Type `help loglog` in the command window. Similar functions are `semilogx` and `semilogy`, which draw log-linear plots.

(c)  $f(x, y) = |x| + |y|$ , for  $|x| \leq 1$ ,  $|y| \leq 1$ .

7. Plot the surface represented by

$$x = u(3 + \cos(v)) \cos(2u),$$

$$y = u(3 + \cos(v)) \sin(2u),$$

$$z = u \sin(v) - 3u,$$

for  $u \in [0, 2\pi]$ ,  $v \in [0, 2\pi]$ .

## More Problems

Below are some more practice problems which resemble the style of exam problems.

1. (Guess-The-Number) Write the following game in which a user is to guess the integer randomly generated by the computer. In the program:
  - User inputs the lower and the upper bounds of the range.
  - The program generates a random integer within the specified range and stores it in a variable.
  - Use a while-loop for repeated guessing.
    - If the user guessed a number larger than the generated number, print out “Your guess is too high. Try again!”.
    - If the user guessed a number smaller than the generated number, print out “Your guess is too low. Try again!”.
    - If the user guessed the number correctly, print out “Congratulations!” and terminate the program.

Below is an example run of the program.

```
>> guess
Enter the lower bound: 1
Enter the upper bound: 100
Guess a number: 50
Your guess is too low. Try again!
Guess a number: 75
Your guess is too low. Try again!
Guess a number: 87
Your guess is too high. Try again!
Guess a number: 81
Your guess is too low. Try again!
Guess a number: 84
Your guess is too high. Try again!
Guess a number: 82
Congratulations!
```

2. (Handling large numbers and scientific notation; Adapted from **LM** 5.6) A product of terms can grow or decay much faster than a sum of terms, leading to an *overflow* or an *underflow* in a floating-point architecture. This difficulty can usually be avoided by replacing

$$P_n = \prod_{i=1}^n a_i \quad \text{by} \quad \log |P_n| = \sum_{i=1}^n \log |a_i| \quad (\spadesuit)$$

as long as none of the terms are 0; if one or more terms are 0 the product is immediate. The result is then  $P_n = f \times 10^m$  in (base-10) scientific notation where  $|f| \in [1, 10)$  and  $f$  can be positive or negative, and where  $m$  is an integer.

Write a MATLAB function which calculates the product of all the elements of an input vector  $a$  by using  $(\spadesuit)$ .

- The name of the function should be `logprod.m` and must output  $f$  and  $m$ .
- $f = 0$  if one of the elements of  $a$  is 0.
- The code must check each element to determine if it is positive, negative, or zero, and also keep track of the overall sign of the product.
- If a zero element is found, the function must exit immediately with  $f = m = 0$ .

3. (Continued fraction revisited) A *continued fraction* is an infinite expression of the form

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{a_3 + \dots}}}$$

If all the  $a_k$ 's are equal to 1, the continued fraction is equal to  $\phi$ , the golden ratio:

$$\phi = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1 + \dots}}} = \frac{1 + \sqrt{5}}{2}.$$

Denote by  $\phi_n$  the  $n$ -term truncation of the continued fraction representation of the above, that is,

$$\begin{aligned}\phi_0 &= 1, \\ \phi_1 &= 1 + \frac{1}{1} = 1 + \frac{1}{\phi_0}, \\ \phi_2 &= 1 + \frac{1}{1 + \frac{1}{1}} = 1 + \frac{1}{\phi_1}, \\ \phi_3 &= 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{1}}} = 1 + \frac{1}{\phi_2}, \\ &\vdots \\ \phi_n &= 1 + \frac{1}{\phi_{n-1}}, \quad \text{for any } n \geq 1.\end{aligned}$$

They can be used to approximate the golden ratio, *i.e.*,  $\phi_n \rightarrow \phi$  as  $n \rightarrow \infty$ .

Use a loop to evaluate  $\phi_j$  for  $j = 1, 2, \dots$  until you get 16 correct digits after the decimal place. Use `fprintf` in each iteration to show that the iterates are converging to the correct value. Your first few iterates and error should look like this:

n	phi_n	error
1	2.0000000000000000	3.8197e-01
2	1.5000000000000000	1.1803e-01
3	1.6666666666666665	4.8633e-02
4	1.6000000000000001	1.8034e-02
5	1.6250000000000000	6.9660e-03
6	1.6153846153846154	2.6494e-03
.....		