Phys234, 2018, Problem set #2: In-lab questions, Thursday Lab

Question 2

Write a function m-file `function p = horner(b,x)` that uses Horner's rule to evaluate a polynomial of arbitrary degree. (Horner's rule is explained in the attachment on the next page). Here, b is a vector of coefficients that define the polynomial. The output p is the value of the polynomial at x. *Make sure the ordering of the coefficients follows that given by equations 3.1 and 3.2 of the next page*. To test your function, write a function file `ps2q2tuesday.m` that calls your `horner` function with `b = [3 -1 2 2]` and the following 5 different values of $x = 1, 2, 3, 4, 5$.

Question 3:

Extend the function developed in question 2 so that it returns a vector of polynomial values if the input x is a vector. Write this function m-file as `function p = hornerv(b,x)` and test it with a function file `ps2q3tuesday.m` that calls it with `b = [3 -1 2 2]` and `x = 1:5`.

Question 4:

Extend the function developed in question 3 and write it as `function [p,pp] = hornerDer(b,x)` so that it can be called in two ways:

```
p = hornerDer(b,x)
```

or

```
[p,pp] = hornerDer(b,x)
```

where p is the value of the polynomial at x and pp is the value of the first derivative of $p(x)$ evaluated at x. Make sure that your function returns p and pp as vectors if x is a vector. Test it with a function file `ps2q4tuesday.m` that calls it with `b = [3 -1 2 2]` and `x=1:5`, and for using the second example of the call option above (with 2 outputs).

preceding statements carefully. Consider, for example, what would happen if the line `for v=A` were replaced with `for v=A'`.

Example 3.7: Horner's Rule

The fourth-order polynomial

$$p_4(x) = b_1 + b_2x + b_3x^2 + b_4x^3 + b_5x^4 = \sum_{i=1}^{5} b_i x^{i-1}$$

can be evaluated with

```
p = b(1) + b(2)*x + b(3)*x^2 + b(4)*x^3 + b(5)*x^4;        (3.1)
```

assuming that the b_i coefficients are stored in the `b` vector. Although this statement is straightforward (and algebraically correct), it is not a good implementation of polynomial evaluation. A better implementation, one that uses fewer calculations, is *Horner's rule*, which is also called *nested multiplication*:

```
p = b(1) + x*( b(2) + x*( b(3) + x*( b(4) + x*b(5) ) ) );   (3.2)
```

Equation (3.1) requires 10 multiplications and 4 additions, whereas Equation (3.2) requires 4 multiplications and 4 additions. Because it involves fewer floating-point calculations, Horner's rule is both more efficient and less susceptible to round-off error. (Cf. Chapter 5.)

Storing the polynomial coefficients in an array allows Horner's rule to be implemented with a `for` loop:

```
n = ... ;          % the polynomial is of order n-1
x = ... ;          % evaluate the polynomial at this value
p = b(n);
for k=n-1:-1:1
   p = p*x + b(k);
end
```

Occasionally, a polynomial and its derivative(s) are simultaneously required. Since

$$p_n(x) = \sum_{i=1}^{n+1} b_i x^{i-1} \quad \Longrightarrow \quad \frac{dp_n}{dx} = \sum_{i=2}^{n+1} b_i(i-1)x^{i-2},$$

a single loop can compute both $p_n(x)$ and $dp_n/dx|_x$. (See Exercise 20.) Although Horner's rule is more efficient than using powers of x to evaluate the polynomial, the built-in `polyval` function should be used for routine polynomial evaluation in MATLAB. (See § 2.3.3 and Exercise 21.)

Note that the built-in functions for manipulating polynomials define a polynomial in *descending* powers of x. (See § 2.3.3 on page 51.) The idea of nested multiplication can be applied to polynomials defined in ascending or descending powers of x.