

COMPUTER SCIENCE 349A ASSIGNMENT #5

DUE Monday February 20, 2012 (in class)

1. (a) Given
 - a positive integer n
 - a vector a with $n+1$ entries a_1, a_2, \dots, a_{n+1}
 - a vector y with n entries y_1, y_2, \dots, y_n
 - a scalar x

write a MATLAB function with header

function p = PolyEval (n, a, y, x)

to evaluate the polynomial

$$p(x) = a_1 + a_2(x - y_1) + a_3(x - y_1)(x - y_2) + a_4(x - y_1)(x - y_2)(x - y_3) + \dots + a_{n+1}(x - y_1)(x - y_2)(x - y_3) \cdots (x - y_n)$$

using Horner's algorithm. PRINT AND HAND IN a copy of your function M-file.

Note. Your function, which can be obtained by modifying the algorithm given on the first page of Handout #13, should use exactly $3n$ flops (floating-point operations). It is not necessary to store all of the partial results as b_n, b_{n-1}, \dots, b_0 as in Handout #13; just use a simple variable p to store all of these results.

- (b) Use your function M-file from (a) to evaluate $p(1.234)$ when $n = 4$ and
$$a = [-1 \ 0 \ 2.33 \ -1.2 \ 2.2],$$
$$y = [-1 \ 1 \ -2 \ 2].$$

2. MATLAB has several functions for standard polynomial operations, including *roots*, *poly* and *polyval*. If the coefficients of a polynomial are stored in a vector p (starting with the coefficient of the highest power of x), then

$$r = \text{roots}(p)$$

will compute and store in r all of the zeros of the polynomial. For example, if $p(x) = x^3 - 2x - 5$, in MATLAB you would enter

$$p = [1 \ 0 \ -2 \ -5];$$
$$r = \text{roots}(p)$$

or simply

$$r = \text{roots}([1 \ 0 \ -2 \ -5])$$

in order to compute the 3 roots of $p(x)$.

On the other hand, if r is any vector, then

$$p = \text{poly}(r)$$

will result in p being a vector whose entries are the coefficients of a polynomial $p(x)$ that has as its roots the entries of r . For example,

$$r = [1 \ 2 \ 3 \ 4];$$
$$p = \text{poly}(r)$$

will result in $p = [1 \ -10 \ 35 \ -50 \ 24]$, since $p(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$ has roots equal to 1, 2, 3 and 4.

Finally, the function *polyval* evaluates a polynomial at a specified value. If $p = [1 \ -10 \ 35 \ -50 \ 24]$, then

$$\text{polyval}(p, -1)$$

gives the value of $p(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$ at $x = -1$, namely $p(-1) = 120$. Note that MATLAB uses HORNER'S ALGORITHM for *polyval*.

(a) It is known that polynomial zeros that have multiplicity greater than 1 are ill-conditioned. Verify this for the polynomial $p(x)$ of degree 8 that has its 8 zeros all equal to 1, by letting $r = [1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1 \ 1]$ and using *poly* and *roots* to compute approximations to the 8 zeros of $p(x)$. Then add 0.005 to the coefficient of x^5 in $p(x)$ to obtain a new polynomial, say $q(x)$, and use *roots* to approximate the 8 zeros of $q(x)$.

Note: You can do this by executing

$$q = p;$$

in MATLAB and then modifying the appropriate entry of the vector p . For example,

$$q(k) = q(k) + 0.005$$

will increase the k^{th} entry of the vector q by 0.005.

Note: MATLAB doesn't compute the zeros of $p(x)$ very accurately (because the zeros are very ill-conditioned), but they are all close to 1 (although some are complex), whereas none of the zeros of $q(x)$ are close to 1.

(b) Use *polyval* to evaluate $q(x)$ at the first zero of $q(x)$ computed by MATLAB in (a). This should give a result very close to 0 (although it is complex).

NOTE: If above you computed the roots of $q(x)$ with the statement

$$z = \text{roots}(q)$$

then the zeros of $q(x)$ will be stored as $z(1), z(2), \dots, z(8)$ and you can use $z(1)$ to answer (b).

3. (a) Develop a MATLAB function M-file *polynewton.m* to compute one zero of a polynomial $f(x)$ of degree n using Newton's method, in which Horner's algorithm (page 2 of Handout #13) is used to evaluate $f(x)$ and $f'(x)$. You can do this by writing a MATLAB function M-file for Horner's algorithm as follows:

```
function [b, c] = horner(x0, n, a)
b(n+1)=a(n+1);
c(n+1)=a(n+1);
for j = n: -1: 2
    b(j) = a(j) + b(j+1)*x0;
    c(j) = b(j) + c(j+1)*x0;
end
b(1) = a(1) + b(2)*x0;
```

The variables b and c in the function definition line (within the $[]$ brackets) are the output variables, and are used to pass values computed in the function back to the point at which the function was called.

The above function is as in the course handouts, except that here we assume that

$$f(x) = a_1 + a_2x + a_3x^2 + \dots + a_{n+1}x^n$$

since MATLAB does not allow 0-indexing; that is, you cannot have a subscript of 0 in MATLAB. Save this as *horner.m*. Execution of *horner* returns the two vectors b and c such that $b(1) = f(x_0)$ and $c(2) = f'(x_0)$.

You can easily modify your MATLAB function M-file *newton* from Assignment #4 to obtain a new MATLAB function M-file *polynewton* for computing one zero of a polynomial.

(i) Change the function definition line to

```
function [root,b] = polynewton(n, a, x0, imax, eps)
```

Thus, the input variables are

n the polynomial degree

a a vector of the $n+1$ coefficients of the polynomial $f(x)$

x_0 the initial approximation to a zero

$imax$ the maximum number of iterations allowed

eps relative error tolerance used to test for convergence

and the output variables are

$root$ the final computed approximation to a zero of $f(x)$

b the final vector of values b computed by Horner's algorithm, from which the deflated polynomial can be obtained.

(ii) Replace the following statement of *newton*

```
root = x0 - f(x0)/fp(x0);
```

by the two statements

```
[b, c] = horner(x0, n, a);
```

```
root = x0 - b(1)/c(2);
```

The first of the latter two MATLAB statements will call the function *horner* with the specified arguments, and return the computed vectors b and c to the MATLAB function M-file *polynewton*.

SAMPLE CALLS of *polynewton* to compute one zero of $f(x) = 5x^4 + 2x^3 - x - 3.3$ using $x0 = 1.3$, $imax = 20$, $eps = 10^{-8}$:

(i) `polynewton(4, [-3.3 -1 0 2 5], 1.3, 20, 1e-8);`

-- will output to the screen each computed approximation to the zero

(ii) `[x, y] = polynewton(4, [-3.3 -1 0 2 5], 1.3, 20, 1e-8);`

-- will output to the screen each computed approximation to a zero, will store the final computed approximation in the variable x , and will store the final computed vector of values b from Horner's algorithm in the vector y

(iii) `[x, y] = polynewton(4, [-3.3 -1 0 2 5], 1.3, 20, 1e-8)`

-- will output to the screen each computed approximation to a zero and the values of x and y (and will store the final computed approximation in the variable x and will store the final computed vector of values b from Horner's algorithm in the vector y)

PRINT AND HAND IN a copy of your MATLAB function M-files *horner* and *polynewton*.

(b) Use *polynewton* to compute one zero of

$$f(x) = x^7 - 3.5x^6 + 2.2x^5 - 1.11x^4 + 2.3x^2 - 5x + 3$$

with $x0 = 1$, $imax = 20$ and $eps = 10^{-8}$.

(c) As described in Handout #14, the computations in (b) also determine the coefficients of the deflated polynomial (with respect to the zero computed in (b)). Use *polynewton* to compute another zero of the polynomial $f(x)$ in (b) by computing a zero of this deflated polynomial with $x0 = -1$, $imax = 20$ and $eps = 10^{-8}$.

Note that after execution of, for example,

```
[x, y] = polynewton(n, a, x0, imax, eps);
```

in (b), the vector y will contain the coefficients of the deflated polynomial (plus one other value). To compute this second zero of $f(x)$, you simply need to call *polynewton* again with the appropriate value of n and the appropriate vector of coefficients. To determine the vector of the coefficients of the deflated polynomial, note for example that if b is the vector $[5.2 \ 3.1 \ 2.2 \ 6.6]$, then execution of $v = b([2 \ 3 \ 4])$ in MATLAB will result in the vector v having the value $[3.1 \ 2.2 \ 6.6]$.

4. Solve (by hand, no programming) the linear system

$$\begin{bmatrix} -2 & 0 & 2 & 4 \\ 1 & 1 & -2 & -2 \\ 0 & 3 & -1 & -3 \\ 4 & -2 & -1 & -9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 4 \\ -7 \\ -9 \\ 5 \end{bmatrix}$$

using Naïve Gaussian elimination (that is, without pivoting). Show each of the derived linear systems, and the back substitution.