

Bent Petersen 351u2001-assignment-01.tex

This assignment is fairly simple in spite of its formidable appearance.

I have written a routine in C to illustrate Horner's method. Those of you who are C++ or Java programmers are probably horrified to find a malloc() call in my code. By all means rewrite it in your favorite language and in your favorite style.

Once you have your code working, or if you elect to use my stone-age code, then evaluate the polynomial

$$\begin{aligned} p(x) &= x^{13} - 91x^{12} + 3731x^{11} - 91091x^{10} + 1474473x^9 \\ &\quad - 16669653x^8 + 135036473x^7 - 790943153x^6 + 3336118786x^5 - 9957703756x^4 \\ &\quad + 20313753096x^3 - 26596717056x^2 + 19802759040x - 6227020800 \end{aligned}$$

at

12.99, 13.00, 13.01.

Problem. Write some comments about what you observe. For example, do you think the size of $|p(a)|$ is a good estimator of when a is close to a root of $p(x)$ (without further analysis)? Include a copy of your code and its output with your submission.

Here is my code - you can download the source file `horner.c` from my web page to save some typing, though you would be better off writing your own code. Note the source code has MSDOS line endings, rather than UNIX line endings, but gcc does not seem to mind.

```
/* horner.c
** Mth 351 Summer 2001
** Bent Petersen
**
** Usage: horner datafile abscissae
**
** Only minimal error checking is done.
**
** Typical: horner poly.dat 3.40 4.21 5.98
** where the numbers are the evaluation points (abscissae).
**
** Format of the data file
**   degree n
**   coefficient of term of degree n
**   coefficient of term of degree n-1
**   ...
**   coefficient of term of degree 0
**
** The numbers can be all on one line, or several lines,
** as long as they are separated by white space. Blank
```

```

** lines are ignored.
**
** Microsoft C/C++    cl horner.c
** GNU Project GCC    gcc -i horner.c -o horner
*/

#include <stdio.h>
#include <math.h>    /* atof() */
#include <stdlib.h>  /* exit() */

typedef struct Poly {
    int deg;
    double * coef; } POLY;

POLY getpoly( char * filename );
void showpoly( POLY poly );
double horner( int degree, double * coefs, double abscissa );

/*****
int main( int argc, char ** argv ) {

POLY    poly;
int     k;
double  temp;

if ( argc < 3 )
    printf("\nUsage: %s datafile abscissae\n", argv[0] );
else {
    poly = getpoly( argv[1] );
    showpoly( poly );

    for ( k=2; k < argc; k++ ) {
        temp = atof(argv[k]);
        printf( "\nThe value at %+18.8f is %+15.8e",temp,
                horner( poly.deg, poly.coef, temp ) );
    }
    printf("\n");
}
return 0;
}

/*****
POLY getpoly( char * filename ) {

POLY    poly;
FILE    *fileptr;
int     k;

if ( (fileptr = fopen( filename, "r" )) == NULL ) {

```

```

    printf( "\nData file not found." );
    exit( 1 );
}

fscanf( fileptr, "%d", &poly.deg );
if ( poly.deg < 1 ) {
    printf( "\nBad degree: %d", poly.deg );
    exit( 1 );
}

poly.coef = (double *)malloc( (size_t)((1+poly.deg)*sizeof(double)) );
if ( poly.coef == NULL ) {
    printf( "\nUnable to allocate memory." );
    exit( 1 );
}

for ( k=0 ; k<=poly.deg ; k++ )
    fscanf( fileptr, "%lf", &poly.coef[poly.deg-k] );

fclose( fileptr );
return poly;
}

/*****/
void showpoly( POLY poly ) {

    int    k;

    printf("\nDegree: %d", poly.deg);

    for (k=poly.deg; k>=0; k--)
        printf("\nCoefficient of term of degree %5d:%4s%+19.12e",
            k, " ",poly.coef[k] );
    printf("\n");
}

/*****/
double horner( int degree, double * coefs, double abscissa ) {

    double  x;
    int     k;

    x = coefs[degree];

    for ( k=degree-1; k>=0; k-- )
        x = x * abscissa + coefs[k];

    return x;
}

```

/*** END ***/

For the polynomial that you are supposed to test just do `horner coef.dat 12.99 13.00 13.01` where `coef.dat` is the following file:

```
13
1
-91
3731
-91091
1474473
-16669653
135036473
-790943153
3336118786
-9957703756
20313753096
-26596717056
19802759040
-6227020800
```

You can also download `coef.dat` from my web page.

We will modify this code later to also compute derivatives and then use it to explore Newton's method for approximating roots. So be sure to save a copy of your code, especially if you modify mine.

If you elect to use a spreadsheet, everything you need is in the `horner()` routine in `horner.c`. If you are committed to using a spreadsheet you may find useful the (ancient) book

William J. Orvis, *1-2-3 for Scientists and Engineers*, Sybex, San Francisco, 1987.

Rules. You may talk to anyone and get help wherever you can for any assignment, but at some point you must write up your work by yourself. If you ask me to help to debug your code you will get a well-deserved large frown. Life's too short to debug my own code, let alone others' code.