

Bent Petersen 351u2001-assignment-03.tex

Your assignment is to investigate Newton's method by approximating some roots to polynomials. I have provided some c code you may use if you wish, though it is somewhat limited by the use of only double precision.

If you apply the provided code to the polynomial $x^2 - 2$ with initial root guess 1.0 you will obtain the output

```
Degree: 2
Coefficient of term of degree 2: +1.000000000000e+00
Coefficient of term of degree 1: +0.000000000000e+00
Coefficient of term of degree 0: -2.000000000000e+00
```

iteration	root estimate	error estimate
0	+1.0000000000000000	+0.5000000000000000
1	+1.5000000000000000	-0.0833333333333333
2	+1.4166666666666667	-0.0024509803921569
3	+1.4142156862745099	-0.0000021238998200
4	+1.4142135623746899	-0.0000000000015947
5	+1.4142135623730951	+0.0000000000000000
6	+1.4142135623730951	+0.0000000000000000

The error estimates are pretty close and the behavior is as expected. **Task 1.** Comment on what, in fact, is expected and why.

If however we consider the polynomial $(x^2 - 2)^2 = x^4 - 4x^2 + 4$ with initial root guess 1.0 our results will be quite different. **Task 2.** Try it with 20 or more iterations and comment on what you see. Try to make a (technical) statement concerning the behavior of the estimated error and of the actual error.

Task 3. Consider the polynomial of degree 13 specified by the data file `coef.dat`

13

1

-91

3731

-91091

1474473

-16669653

135036473

-790943153
3336118786
-9957703756
20313753096
-26596717056
19802759040
-6227020800

All of the roots of this polynomial are simple. Try the command `newton coef.dat 3.5 12 0`. Are the results what you expect? What do you think is happening?

Compile and run code successfully	25 pts.
Write your own code (including spreadsheet)	5 pts.
Discussion of results - tasks 1, 2 and 3	20 pts.

Here is the code for `newton.c`. More conveniently, you can download it, and also the file `coef.dat`, from my web page.

```
/* newton.c
** Mth 351 Summer 2001
** Bent Petersen
**
** This program estimates real roots of polynomials by using
** Newton's method.
**
** Usage:  newton datafile guess iterations start_output
**
** Only minimal error checking is done.
**
** Typical:  newton coef.dat 3.42 12 10
**
** to do 12 iterations but printout only numbers 10, 11 and 12.
**
** Format of the data file coef.dat
**   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 newton.c
** GNU Project GCC      gcc -i newton.c -o newton
*/

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

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

typedef struct Jet1 {
    double d0;
    double d1; } JET1;

#define SIGN(x) ( ( (x) < 0 ) ? -1 : 1 )
#define MAX(x,y) ( ( (x) < (y) ) ? (y) : (x) )

#define BIGNUM 20      /* Largest step we will take */
#define LILNUM 1.0e-8  /* Replacement for 0 derivative */

POLY getpoly( char * filename );
void showpoly( POLY poly );
JET1 horner1( POLY poly, double abscissa );
double newton( POLY poly, double abscissa );

/*****
** main() simply orchestrates the action
*****/

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

    POLY      poly;
    int       k;
    double    abscissa,temp;
    int       iterations,startout;

    if ( argc < 5 )
        printf("\nUsage: %s datafile guess iterations start_output\n", argv[0] );
    else {
        poly = getpoly( argv[1] );
        showpoly( poly );
        abscissa = atof( argv[2] );
        iterations = MAX( atoi( argv[3] ), 0 );
    }
}

```

```

startout = (int)atof( argv[4] );

printf( "\n%10s %22s %22s", "iteration", "root estimate", "error estimate" );

for ( k=0; k<=iterations; k++ ) {
    temp = abscissa;
    abscissa = newton( poly, temp );
    if ( k >= startout )
        printf( "\n%10d %+22.16f %+22.16f", k, temp, abscissa-temp );
    }
printf("\n");
}
return 0;
}

/*****
** getpoly() allocates space for the array
** of polynomial coefficients and reads the
** data from the specified data file.
*****/

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;
}

/*****
** showpoly() prints the array
** of polynomial coefficients
*****/

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");
}

/*****
** horner1() evaluates the given polynomial and
** its first derivative at the specified point
*****/

JET1 horner1( POLY poly, double abscissa ) {

JET1     jet;
int      k;

jet.d0 = poly.coef[poly.deg];
jet.d1 = 0;

for ( k=poly.deg-1; k>=0; k-- ) {
    jet.d1 = jet.d1 * abscissa + jet.d0;
    jet.d0 = jet.d0 * abscissa + poly.coef[k];
}
return jet;
}

/*****
** newton() performs a single Newton iteration

```

```

** and returns the new root estimate. We try to
** avoid division by 0, but otherwise perform
** no error checking.
*****/

double newton( POLY poly, double abscissa ) {

double      quotient;
JET1        jet;

jet = horner1( poly, abscissa );

if ( jet.d0 == 0 )
    return (double)0;

if ( jet.d1 == 0 )
    quotient = jet.d0/LILNUM;
else
    quotient = jet.d0/jet.d1;

if ( fabs(quotient) > BIGNUM )
    quotient = (double)SIGN(quotient) * BIGNUM;

return abscissa - quotient;
}
/**/ END ***/

```