

Bent Petersen 351u2001-assignment-04.tex

Maple has a command `interp()` for computing interpolation polynomials. It is easy to use and provides a convenient way to experiment with interpolation. In this assignment however I will provide C code so you can easily convert the code to your favorite language (which probably does not have the high-level facilities provided by Maple).

The program `divdiff` (source below) takes 5 parameters:

$a$	left end point of interval
$b$	right end point of interval
$n$	degree of interpolation polynomial $p(x)$ for $f(x)$
$N$	number of equispaced points in $[a, b]$ where we compare $f(x)$ and $p(x)$
$t$	type, "e" for equispaced nodes and "c" for Chebyshev nodes

Here the function  $f$  is defined in the source code (sorry) and starts out as

$$f(x) = \frac{1}{1 + 50x^2}.$$

If you change  $f(x)$  (as in task (C) below) you will have to recompile the program.

**Task (A).** Try the commands `divdiff -1, 1, 16, 30, e` and `divdiff -1, 1, 16, 30, c`. Explain what each command is doing and comment on your results.

Note if you are working at an MSDOS prompt or at a Unix (or Linux) prompt you can use

```
divdiff -1, 1, 16, 30, c > ceb.dat
```

to capture the output in the file `ceb.dat`.

**Task (B).** Repeat Task (A) with polynomials of degree 24 and comment on you results.

**Task (C).** Repeat tasks (A) and (B) with  $f(x) = \exp(2x)$  and comment on your results.

Here is the code. You can also download it from my web page to save typing.

```
/* divdiff.c
** Mth 351 Summer 2001
** Bent Petersen
**
** Usage:  divdiff a b n N t
**
** where [a,b] is the interval we interpolate on, n+1 is the
** number of nodes, t is the type (t=e for equispaced and
** t=c for Chebyshev), N+1 is the number of equispaced points
```

```

** at which to compare the specified function and the
** interpolation polynomial.
**
** The results are written to stdout.
**
** Note: the function we are interpolating is hardwired in the
** code. This is not the best arrangement but it suffices for
** our experiments. It would be a lot of work to write routines
** to parse a function definition.
**
** Microsoft C/C++    cl divdiff.c
** GNU Project GCC    gcc -i divdiff.c -o divdiff -lm
**
#include <stdio.h>
#include <math.h>    /* atof(), fabs() */
#include <stdlib.h>  /* exit() */

#define PI    4.0 * atan(1.0)
#define MAX(x,y) ((x)<(y) ? (y) : (x) )

/* -----
** Prototypes
** -----
*/
double F( double x );
int main( int argc, char ** argv );
void calcdds( int n, double * nodes, double * dds );
double calcpoly( int n, double * nodes, double * dds, double x);

/* -----
** Here's our test function. It would be more
** efficient to define it as a macro, but for a
** complicated function definition a macro may
** require numerous nuisance parantheses.
** -----
*/
double F( double x ) {
    double y;

    y = (double)1.0 / ( 1.0 + 50.0*x*x );

    return y;
}

/* -----
** Calculate Newton divided differences.
** -----
*/

```

```

void calcds( int n, double * nodes, double * dds ) {
    int k,j;

    for ( k=1; k<=n; k++ )
        for ( j=n; j>=k; j-- )
            dds[j] = (dds[j]-dds[j-1]) / (nodes[j]-nodes[j-k]);
}

/* -----
** Calculate value of interpolation polynomial by using
** a modified Horner method and Newton's formula.
** -----
*/
double calcpoly( int n, double * nodes, double * dds, double x ) {
    int k;
    double temp;

    temp = dds[n];
    for ( k=n-1; k>=0; k-- )
        temp = temp * (x - nodes[k]) + dds[k];
    return temp;
}

/* -----
** Here's where we start
** -----
*/
int main( int argc, char ** argv ) {
    double a,b,x,y,z,error,maxabserr;
    int n,N,k;
    char *type;
    double *nodes,*divdiffs;

    if ( argc < 6 ) {
        printf( "\nUsage: %s a b n N type{e,c}\n", argv[0] );
        exit(1);
    }

    a = atof( argv[1] );
    b = atof( argv[2] );
    n = atoi( argv[3] );
    N = atoi( argv[4] );

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

```

```

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

if ( (argv[5][0] == 'c') || (argv[5][0] == 'C') ) {
    type = "Chebyshev nodes";
    for (k=0; k<=n; k++ )
        nodes[k] = a + (b-a)*( 1 - cos( PI*(2*k+1)/(2*n+2) ) ) / 2.0;
}
else {
    type = "equispaced nodes";
    for (k=0; k<=n; k++ )
        nodes[k] = a + (b-a)*k/n;
}

printf( "\nInterpolation polynomial using %d %s", n+1, type );
printf( "\non the interval [%f, %f] and compared", a, b );
printf( "\nwith the original function at %d equispaced points\n", N+1 );

/* calculate Newton divided differences */

for ( k=0; k<=n; k++ )
    divdiffs[k] = F( nodes[k] ); /* initialize divdiffs before calling calcds() */
calcds( n, nodes, divdiffs );

/* evaluate at test points */

printf( "\n%18s %18s %18s %18s", "abscissa", "F(x)", "P(x)", "error" );
maxabserr = 0;

for ( k=0; k<=N; k++ ) {
    x = a + k*(b-a)/N;
    y = calcpoly( n, nodes, divdiffs, x );
    z = F(x);
    error = z - y;
    maxabserr = MAX( maxabserr, fabs(error) );

    printf( "\n%+18.12f %+18.12f %+18.12f %+18.12f", x, z, y, error );
}
printf( "\n\nMaximum absolute error (%s): %+18.12f\n", type, maxabserr );
return 0;
}

```