Math 128A Spring 2002 Handout # 13
Sergey Fomel February 26, 2002

Answers to Homework 4: Interpolation: Polynomial Interpolation

---

1. Prove that the sum of the Lagrange interpolating polynomials

$$L_k(x) = \prod_{i \neq k} \frac{x - x_i}{x_k - x_i} \tag{1}$$

is one:

$$\sum_{k=1}^{n} L_k(x) = 1 \tag{2}$$

for any real x, integer n, and any set of distinct points $x_1, x_2, \ldots, x_n$.

Solution: When we interpolate the function $f(x) = 1$, the interpolation polynomial (in the Lagrange form) is

$$P(x) = \sum_{k=1}^{n} f(x_k)\, L_k(x) = \sum_{k=1}^{n} L_k(x)\,.$$

For any $x_1, \ldots, x_n$, the data are perfectly interpolated by the zeroth-order polynomial $P(x) = f(x) = 1$. Since the interpolation polynomial is unique, we have

$$1 = P(x) = \sum_{k=1}^{n} L_k(x)$$

for any x.

2. Let $f(x) = x^{n-1}$ for some $n \geq 1$. Find the divided differences

$f[x_1, x_2, \ldots, x_n]$ and $f[x_1, x_2, \ldots, x_n, x_{n+1}]$,

where $x_1, x_2, \ldots, x_n, x_{n+1}$ are distinct numbers.

Solution: We can use the formula

$$f[x_1, x_2, \ldots, x_n] = \frac{f^{(n-1)}(\xi)}{(n-1)!}\,,$$

where ξ is a point between the maximum and minimum of $x_1, x_2, \ldots, x_n$. Differentiating $f(x) = x^{n-1}$ produces

$$\begin{aligned}
f'(x) &= (n-1)\,x^{n-2} \\
f''(x) &= (n-1)(n-2)\,x^{n-3} \\
&\ldots \\
f^{(n-1)}(x) &= (n-1)! \\
f^{(n)}(x) &= 0
\end{aligned}$$

Therefore,

$$\begin{aligned}
f[x_1, x_2, \ldots, x_n] &= \frac{f^{(n-1)}(\xi)}{(n-1)!} = \frac{(n-1)!}{(n-1)!} = 1\,. \\
f[x_1, x_2, \ldots, x_n, x_{n+1}] &= \frac{f^{(n)}(\xi)}{n!} = 0\,.
\end{aligned}$$

3. (a) Consider a set of regularly spaced nodes on interval $[a,b]$:

$$h = \frac{b-a}{n}\,, \quad x_k = a + (k-1)\,h\,, \quad k = 1,2,\ldots,n+1\,. \tag{3}$$

Prove that the polynomial

$$N(x) = (x-x_1)(x-x_2)\cdots(x-x_{n+1}) \tag{4}$$

satisfies

$$|N(x)| \le n!\,h^{n+1}\,, \quad a \le x \le b \tag{5}$$

Solution: For any $x \in [a,b]$, let x_{k_1} be the node closest to x. The distance $\left|x - x_{k_1}\right| \le h$. If x_{k_2} is the next closest node, then $\left|x - x_{k_2}\right| \le h$. Similarly,

$$\begin{aligned} \left|x - x_{k_3}\right| &\le 2h \\ \left|x - x_{k_4}\right| &\le 3h \\ &\cdots \\ \left|x - x_{k_{n+1}}\right| &\le n\,h \end{aligned}$$

Therefore,

$$\begin{aligned} |N(x)| &= |x-x_1|\,|x-x_2|\cdots|x-x_{n+1}| = \left|x-x_{k_1}\right|\left|x-x_{k_2}\right|\cdots\left|x-x_{k_{n+1}}\right| \\ &\le h\cdot h\cdot 2h\cdot\cdots\cdot n\,h = n!\,h^{n+1}\,. \end{aligned}$$

Note: There is, in fact, a tighter bound

$$|N(x)| \le \frac{n!}{4}\,h^{n+1}\,,$$

but it is a little bit more difficult to prove.

(b) Using the result of problem (a), prove that if $f(x) = e^x$ and $P_n(x)$ is the interpolating polynomial of order n defined at the $n+1$ regularly spaced nodes

$$x_k = \frac{k-1}{n}\,, \quad k = 1,2,\ldots,n+1 \tag{6}$$

then the interpolation error

$$e_n = \max_{0\le x\le 1} |f(x) - P_n(x)| \tag{7}$$

goes to zero as n goes to infinity:

$$\lim_{n\to\infty} e_n = 0 \tag{8}$$

Solution: The error of polynomial interpolation is

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,N(x)\,,$$

where ξ is a point in $[0,1]$. Therefore,

$$e_n = \max_{0\le x\le 1} |f(x) - P_n(x)| \le \max_{0\le \xi\le 1} \frac{e^{\xi}}{(n+1)!}\, n!\, h^{n+1} = \frac{e}{(n+1)} \left(\frac{1}{n}\right)^{n+1}$$

Since the limit of the right hand side is zero:

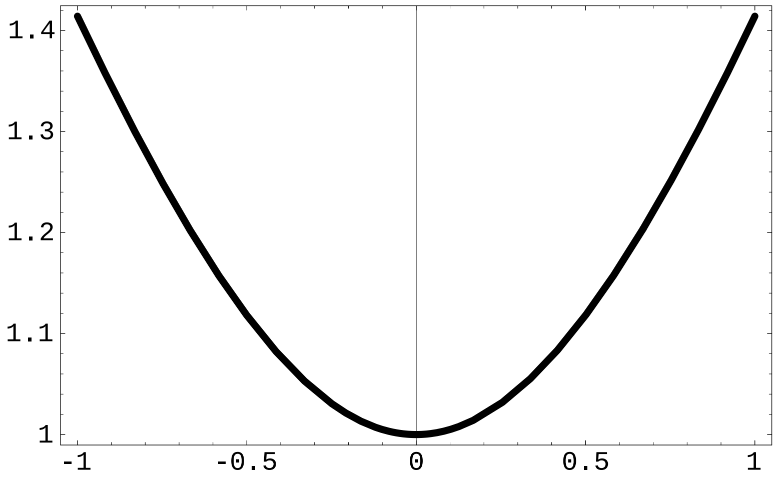
$$\lim_{n\to\infty} \frac{e}{(n+1)} \left(\frac{1}{n}\right)^{n+1} = 0\,,$$

we can conclude that

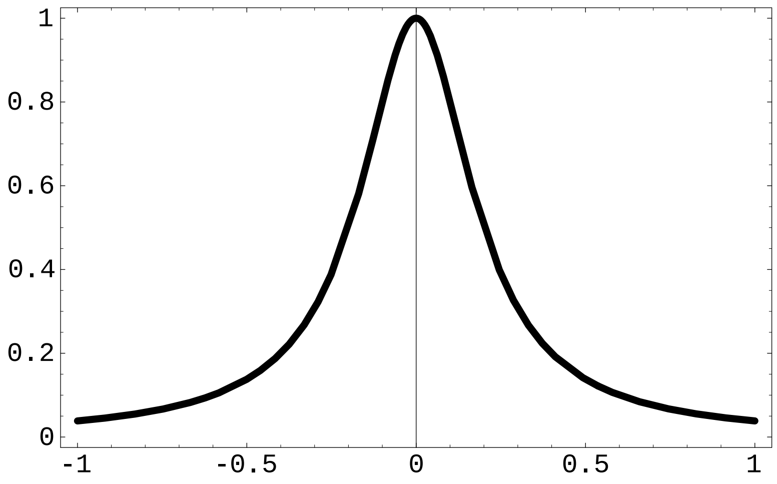
$$\lim_{n\to\infty} e_n = 0\,.$$

4. (Programming) Implement one of the algorithms for polynomial interpolation and interpolate

   (a) hyperbola $f(x) = \sqrt{1+x^2}$



   (b) Runge's function $f(x) = \frac{1}{1+25x^2}$



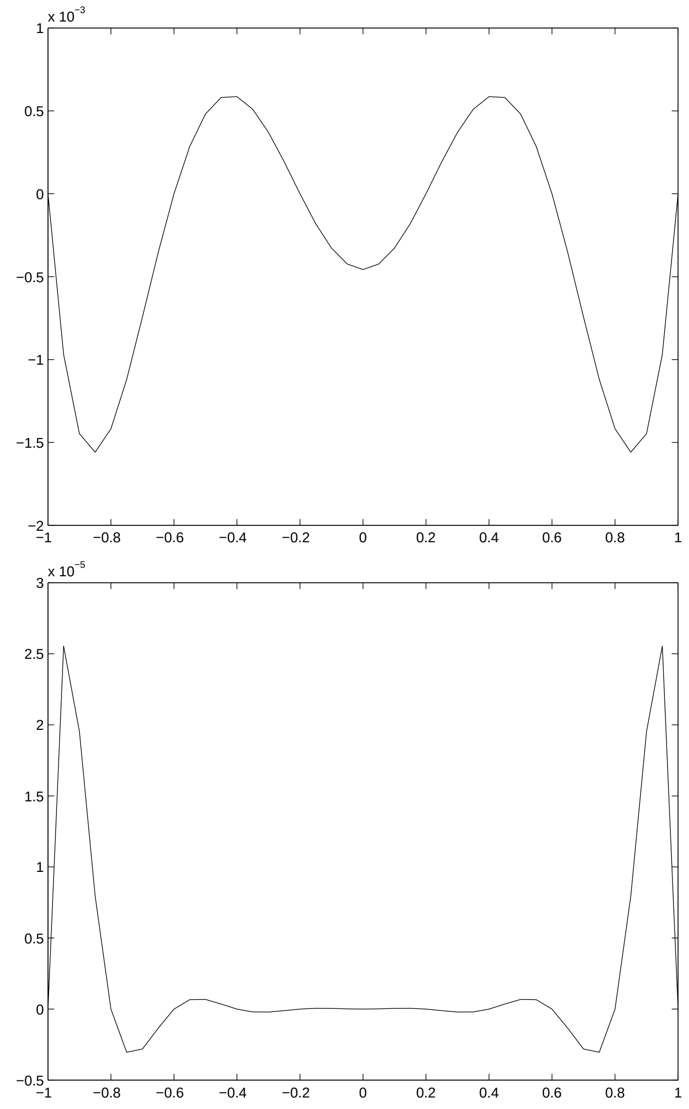
using a set of $n+1$ regularly spaced nodes

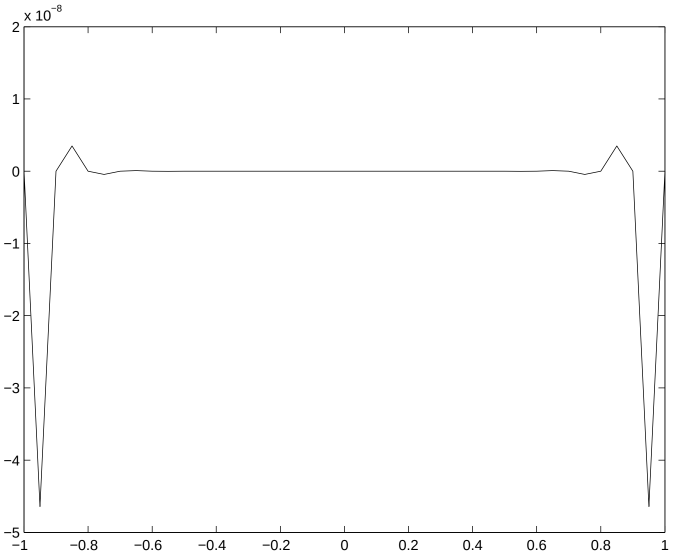
$$x_k = -1 + \frac{2(k-1)}{n}\,, \quad k = 1,2,\ldots,n+1\,.$$

Take $n = 5, 10, 20$ and compute the interpolation polynomial $P_n(x)$ and the error $f(x) - P_n(x)$ at 41 regularly spaced points. You can either plot the error or output it in a table. Does the interpolation accuracy increase with the order n?

Answer:

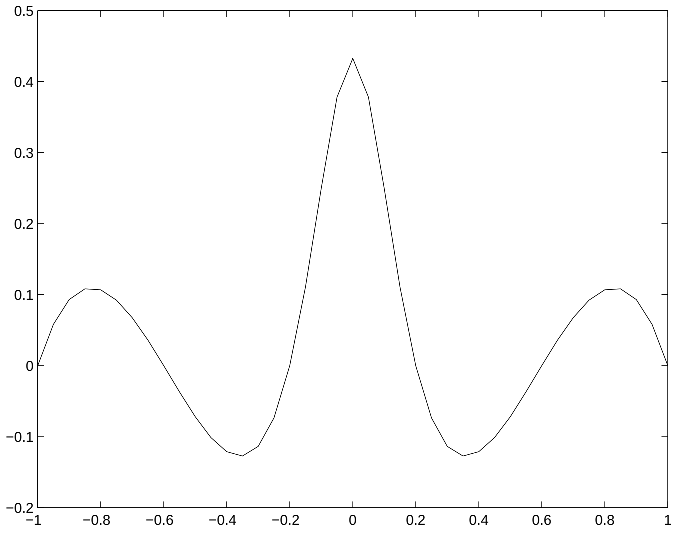
(a)

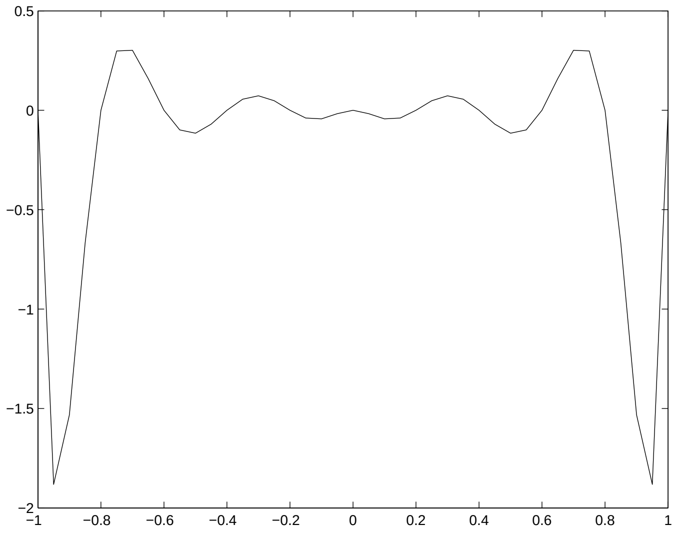


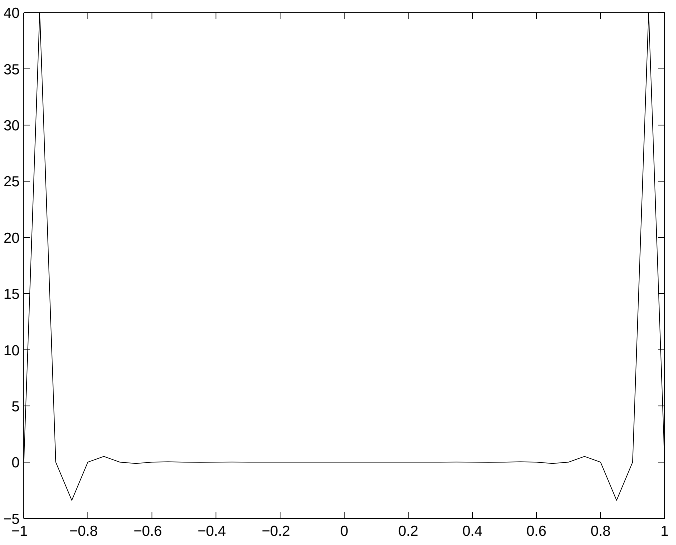


The maximum error decreases (accuracy increases) with the increase of the polynomial order.

(b)







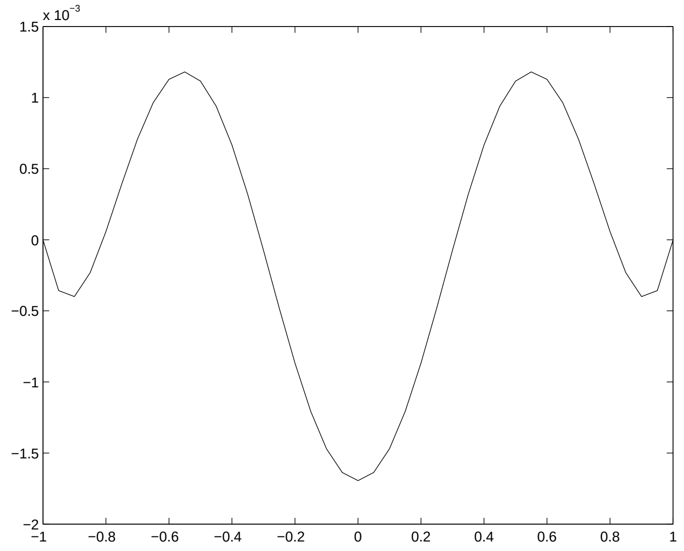
The maximum error increases (accuracy decreases) with the increase of the polynomial order.

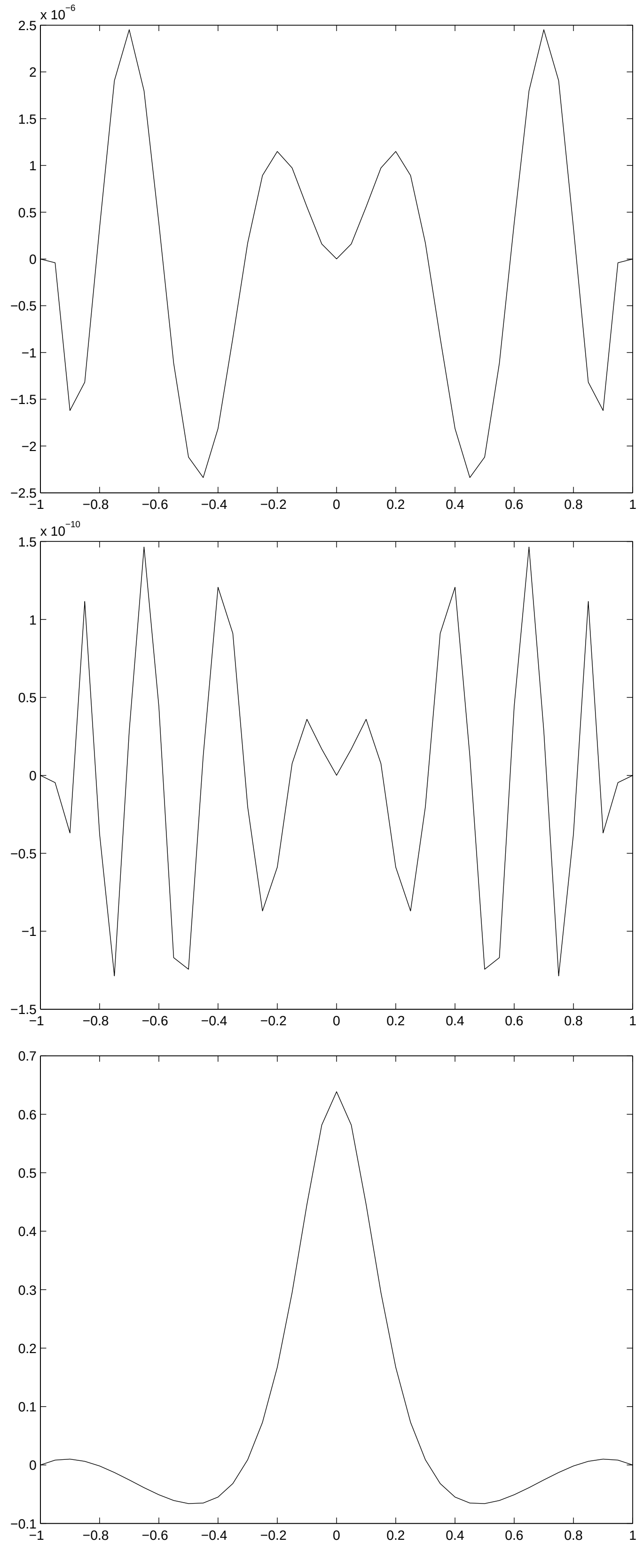
5. (Programming) Repeat the experiments of the previous problem replacing the regularly spaced nodes with nodes

$$x_k = \cos\left(\frac{\pi\,(k-1)}{n}\right)\,, \quad k = 1,2,\ldots,n+1\,.$$

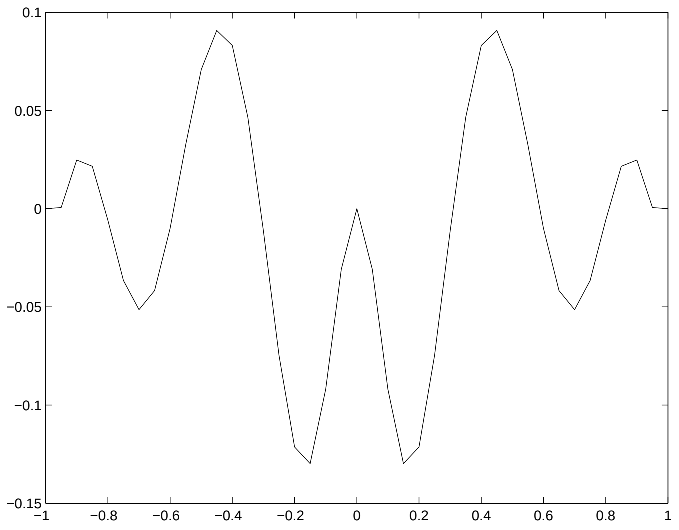
Compare the accuracy.

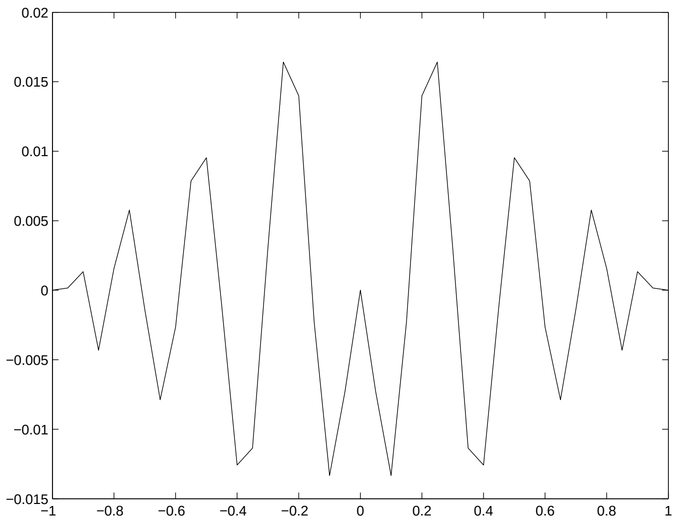
(a)





(b)





Now, in both cases, the maximum error decreases (accuracy increases) with the increase of the polynomial order. The second method of placing the interpolation nodes leads to more accurate results.

Solution: C program

```
#include <stdlib.h> /* for allocations */
#include <math.h>   /* for mathematical functions */
#include <stdio.h>  /* for output */
#include <assert.h> /* for assertion */

/*
   Lagrange interpolation
   n     - number of points
   x     - where to evaluate
   xk[n] - nodes
   fk[n] - function values
*/
double lagrange (int n, double x, double* xk, double* fk)
{
    int i, k;
    double p, lk;
    p = 0.;
    for (k=0; k < n; k++) {
lk = 1.;
for (i=0; i < n; i++) {
    if (i==k) continue;
    /* accumulate Lk(x) */
    lk *= (x - xk[i])/(xk[k] - xk[i]);
}
/* accumulate the sum */
p += lk*fk[k];
    }
    return p;
}

/* test function */
static double func (double x, int function)
{
    if (function == 1) {
return (sqrt(1. + x*x)); /* Hyperbola function */
    } else {
return (1./(1. + 25*x*x)); /* Runge’s function */
    }
}

/* main program */
int main (void)
{
    const int function=2, method=2;
    int i, k, n[]={5,10,20}, nx, ny=41;
    double p, e, xk, *x, *y, *f, pi;

    y = (double*) malloc (ny*sizeof(double));
    assert (y != NULL);

    /* regular grid for plotting */
    for (k=0; k < ny; k++) {
y[k] = -1. + 2.*k/(ny-1.);
    }

    pi = acos(-1.); /* the number pi */
    for (i=0; i < 3; i++) {
nx = n[i]+1;

/* allocate space */
x = (double*) malloc (nx*sizeof(double));
assert (x != NULL);

f = (double*) malloc (nx*sizeof(double));
assert (f != NULL);

/* build the table */
for (k=0; k < nx; k++) {
    xk = (method == 1)? -1. + 2.*k/(nx-1.): cos(pi*k/(nx-1.));
    f[k] = func(xk,function);
    x[k] = xk;
}

/* evaluate the interpolation polynomial */
for (k=0; k < ny; k++) {
    xk = y[k];
    p = lagrange (nx, xk, x, f);  /* polynomial */
    e = func(xk,function)-p;      /* error */
    /* print out the table */
    printf("%d %f %f %g\n", k, xk, p, e);
}

free(x);
free(f);
    }

    free (y);
    exit(0);
}
```