Math 128A Spring 2002 Handout # 15
Sergey Fomel March 5, 2002

Answers to Homework 5: Interpolation: Polynomial Interpolation 2

1. Consider $2n+1$ regularly spaced interpolation nodes $x_{-n}, x_{-n+1}, \ldots, x_{-1}, x_0, x_1, \ldots, x_n$ with $x_k = x_0 + k\,h$, $k = -n, -n+1, \ldots, -1, 0, 1, \ldots, n-1, n$.

   (a) Derive a formula for the interpolation polynomial of order $2n$, using Newton's method and adding points in the order $x_0, x_{-1}, x_1, x_{-2}, x_2, \ldots, x_{-n}, x_n$.

   (b) Derive another formula, adding points in the order $x_0, x_1, x_{-1}, x_2, x_{-2}, \ldots, x_n, x_{-n}$.

   (c) Take the average of the two formulas and show that it is equivalent to Stirling's interpolation formula

$$\begin{aligned} P(x) \;=\; & f(x_0) + s\,\frac{\Delta f(x_{-1}) + \Delta f(x_0)}{2} + \frac{s^2}{2}\,\Delta^2 f(x_{-1}) + \\ & \frac{s\left(s^2-1^2\right)}{3!}\,\frac{\Delta^3 f(x_{-2}) + \Delta^3 f(x_{-1})}{2} + \frac{s^2\left(s^2-1^2\right)}{4!}\,\Delta^4 f(x_{-2}) + \ldots + \\ & \frac{s\left(s^2-1\right)\left(s^2-2^2\right)\ldots\left(s^2-(n-1)^2\right)}{(2n-1)!}\,\frac{\Delta^{2n-1} f(x_{-n}) + \Delta^{2n-1} f(x_{-n+1})}{2} + \\ & \frac{s^2\left(s^2-1\right)\left(s^2-2^2\right)\ldots\left(s^2-(n-1)^2\right)}{(2n)!}\,\Delta^{2n} f(x_{-n})\;, \end{aligned} \tag{1}$$

   where $s = (x-x_0)/h$, and $\Delta f(x) = f(x+h) - f(x)$.

   (d) Check each of the three formulas using $n=2$, $h=1$, and $f(x) = x^2$.

   Solution:

   (a) The Newton interpolation formula for this case is

$$\begin{aligned} P(x) \;=\; & f(x_0) + f\left[x_0, x_{-1}\right](x-x_0) + f\left[x_0, x_{-1}, x_1\right](x-x_0)(x-x_{-1}) + \ldots + \\ & f\left[x_0, x_{-1}, \ldots, x_{-n}\right](x-x_0)(x-x_{-1})\cdots(x-x_{n-1}) + \\ & f\left[x_0, x_{-1}, \ldots, x_n\right](x-x_0)(x-x_{-1})\cdots(x-x_{-n})\;. \end{aligned}$$

   Taking into account the regular spacing of the nodes, we get

$$\begin{aligned} x-x_0 &= s\,h \\ (x-x_0)(x-x_{-1}) &= s\,h\,(s\,h+h) = s\,(s+1)\,h^2 \\ (x-x_0)(x-x_{-1})(x-x_1) &= s\,(s+1)\,h^2\,(s\,h-h) = s\,(s^2-1)\,h^3 \\ \ldots & \quad \ldots \\ (x-x_0)(x-x_{-1})\cdots(x-x_{n-1}) &= s\,(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)h^{2n-1} \\ (x-x_0)(x-x_{-1})\cdots(x-x_{-n}) &= s\,(s+n)(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)h^{2n} \end{aligned}$$

   The divided differences transform accordingly, as follows:

$$f\left[x_0, x_{-1}\right] = f\left[x_{-1}, x_0\right] = \frac{f(x_0) - f(x_{-1})}{x_0 - x_{-1}} \;=\; \frac{\Delta f(x_{-1})}{h}$$

$$
\begin{aligned}
f\left[x_0,x_{-1},x_1\right] = f\left[x_{-1},x_0,x_1\right] = \frac{f\left[x_0,x_1\right]-f\left[x_{-1},x_0\right]}{x_1-x_{-1}} &= \frac{\Delta^2 f(x_{-1})}{2h} \\
\cdots &\quad \cdots \\
f\left[x_0,x_{-1},\ldots,x_{-n}\right] = f\left[x_{-n},x_{-n+1},\ldots,x_{n-1}\right] &= \frac{\Delta^{2n-1} f(x_{-n})}{(2n-1)!\,h^{2n-1}} \\
f\left[x_0,x_{-1},\ldots,x_{n}\right] = f\left[x_{-n},x_{-n+1},\ldots,x_{n}\right] &= \frac{\Delta^{2n} f(x_{-n})}{(2n)!\,h^{2n}}
\end{aligned}
$$

We used here the fact that divided differences do not depend on the order of the nodes. Putting it all together, we obtain the first formula:

$$
\begin{aligned}
P(x) \;=\;& f(x_0) + \Delta f(x_{-1}) s + \frac{\Delta^2 f(x_{-1})}{2}\, s\,(s+1) + \frac{\Delta^3 f(x_{-2})}{3!}\, s\,(s^2-1) + \ldots + \\
& \frac{\Delta^{2n-1} f(x_{-n})}{(2n-1)!}\, s\,(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right) + \\
& \frac{\Delta^{2n} f(x_{-n})}{(2n)!}\, s\,(s+n)(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)
\end{aligned}
$$

Note: This formula is known as the Gauss interpolation formula.

(b) The derivation of the second formula is analogous.

The Newton interpolation formula for this case is

$$
\begin{aligned}
P(x) \;=\;& f(x_0) + f\left[x_0,x_1\right](x-x_0) + f\left[x_0,x_1,x_{-1}\right](x-x_0)(x-x_1) + \ldots + \\
& f\left[x_0,x_1,x_{-1},\ldots,x_n\right](x-x_0)(x-x_1)(x-x_{-1})\cdots(x-x_{-n+1}) + \\
& f\left[x_0,x_1,x_{-1}\ldots,x_{-n}\right](x-x_0)(x-x_1)(x-x_{-1})\cdots(x-x_n)\;.
\end{aligned}
$$

Taking into account the regular spacing of the nodes, we get

$$
\begin{aligned}
x-x_0 &= s\,h \\
(x-x_0)(x-x_1) &= s\,h\,(s\,h-h) = s\,(s-1)\,h^2 \\
(x-x_0)(x-x_1)(x-x_{-1}) &= \\
s\,(s-1)\,h^2\,(s\,h+h) = s\,(s^2-1)\,h^3 & \\
\cdots &\quad \cdots \\
(x-x_0)(x-x_1)(x-x_{-1})\cdots(x-x_{-n+1}) &= \\
s\,(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)\,h^{2n-1} & \\
(x-x_0)(x-x_1)(x-x_{-1})\cdots(x-x_n) &= \\
s\,(s-n)(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)\,h^{2n} &
\end{aligned}
$$

The divided differences transform accordingly:

$$\begin{aligned}
f\left[x_0,x_1\right] = \frac{f(x_1)-f(x_0)}{x_1-x_0} &= \frac{\Delta f(x_0)}{h} \\
f\left[x_0,x_1,x_{-1}\right] = f\left[x_{-1},x_0,x_1\right] = \frac{f\left[x_0,x_1\right]-f\left[x_{-1},x_0\right]}{x_1-x_{-1}} &= \frac{\Delta^2 f(x_{-1})}{2h} \\
\cdots \quad & \quad \cdots \\
f\left[x_0,x_1,x_{-1},\ldots,x_n\right] = f\left[x_{-n+1},x_{-n+2},\ldots,x_n\right] &= \frac{\Delta^{2n-1} f(x_{-n+1})}{(2n-1)!\,h^{2n-1}} \\
f\left[x_0,x_1,x_{-1},\ldots,x_{-n}\right] = f\left[x_{-n},x_{-n+1},\ldots,x_n\right] &= \frac{\Delta^{2n} f(x_{-n})}{(2n)!\,h^{2n}}
\end{aligned}$$

Putting it all together, we obtain the second formula:

$$\begin{aligned}
P(x) &= f(x_0)+\Delta f(x_0)\,s+\frac{\Delta^2 f(x_{-1})}{2}\,s\,(s-1)+\frac{\Delta^3 f(x_{-1})}{3!}\,s\,(s^2-1)+\ldots+ \\
&\quad \frac{\Delta^{2n-1} f(x_{-n+1})}{(2n-1)!}\,s\,(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)+ \\
&\quad \frac{\Delta^{2n} f(x_{-n})}{(2n)!}\,s\,(s-n)(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)
\end{aligned}$$

(c) Taking the average term by term yields

$$\begin{aligned}
\frac{1}{2}\left[f(x_0)+f(x_0)\right] &= f(x_0) \\
\frac{1}{2}\left[\Delta f(x_{-1})\,s+\Delta f(x_0)\,s\right] &= \frac{\Delta f(x_{-1})+\Delta f(x_0)}{2}\,s \\
\frac{1}{2}\left[\frac{\Delta^2 f(x_{-1})}{2}\,s\,(s+1)+\frac{\Delta^2 f(x_{-1})}{2}\,s\,(s-1)\right] &= \frac{\Delta^2 f(x_{-1})}{2}\,s^2 \\
\frac{1}{2}\left[\frac{\Delta^3 f(x_{-2})}{3!}\,s\,(s^2-1)+\frac{\Delta^3 f(x_{-1})}{3!}\,s\,(s^2-1)\right] &= \frac{\Delta^3 f(x_{-2})+\Delta^3 f(x_{-1})}{2}\,\frac{s\,(s^2-1)}{3!} \\
\cdots \quad & \quad \cdots
\end{aligned}$$

$$\begin{aligned}
&\frac{1}{2}\left[\frac{\Delta^{2n-1} f(x_{-n})}{(2n-1)!}\,s\,(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)+\right. \\
&\quad \left.\frac{\Delta^{2n-1} f(x_{-n+1})}{(2n-1)!}\,s\,(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)\right] = \\
&\quad \frac{\Delta^{2n-1} f(x_{-n})+\Delta^{2n-1} f(x_{-n+1})}{2}\,\frac{s\,(s^2-1)(s^2-2^2)}{(2n-1)!} \\
&\frac{1}{2}\left[\frac{\Delta^{2n} f(x_{-n})}{(2n)!}\,s\,(s+n)(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)+\right. \\
&\quad \left.\frac{\Delta^{2n} f(x_{-n})}{(2n)!}\,s\,(s-n)(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)\right] = \\
&\quad \Delta^{2n} f(x_{-n})\,\frac{s^2(s^2-1)(s^2-2^2)\cdots\left(s^2-(n-1)^2\right)}{(2n)!}
\end{aligned}$$

Summing the term leads directly to the Stirling interpolation formula.

(d) The difference table for the specified case is

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$
x_0-2	$(x_0-2)^2$	$2x_0-3$	2	0
x_0-1	$(x_0-1)^2$	$2x_0-1$	2	0
x_0	x_0^2	$2x_0+1$	2	
x_0+1	$(x0+1)^2$	$2x_0+3$		
x_0+2	$(x0+2)^2$			

The first formula produces

$$P(x) = x_0^2 + (2x_0-1)(x-x_0) + \frac{2}{2}(x-x_0)(x-x_0+1) = x_0^2 + 2x_0(x-x_0) + (x-x_0)^2 = x^2 .$$

The second formula produces

$$P(x) = x_0^2 + (2x_0+1)(x-x_0) + \frac{2}{2}(x-x_0)(x-x_0-1) = x_0^2 + 2x_0(x-x_0) + (x-x_0)^2 = x^2 .$$

The Stirling formula produces

$$P(x) = x_0^2 + (x-x_0)\frac{2x_0-1+2x_0+1}{2} + \frac{(x-x_0)^2}{2}2 = x_0^2 + 2x_0(x-x_0) + (x-x_0)^2 = x^2 .$$

2. Let $S(n) = 1^2 + 2^2 + \ldots + n^2$. Note that $\Delta S(n) = S(n+1) - S(n) = (n+1)^2$ is a quadratic polynomial.

(a) Verify that $\Delta^4 S(n) = 0$. This indicates that $S(n)$ is a polynomial of degree 3.

(b) Find $S(n)$ by evaluating it at four points $n = 1,2,3,4$ and interpolating a cubic polynomial through them.

Solution:

(a) Indeed,

$$\begin{aligned}
\Delta^2 S(n) &= (n+2)^2 - (n+1)^2 = 2n+3 ; \\
\Delta^3 S(n) &= 2(n+1) + 3 - (2n+3) = 2 ; \\
\Delta^4 S(n) &= 2 - 2 = 0 .
\end{aligned}$$

(b) The difference table in this case is

x	$f(x)$	$\Delta f(x)$	$\Delta^2 f(x)$	$\Delta^3 f(x)$
1	1	4	5	2
2	5	9	7	
3	14	16		
4	30			

The Newton form of the interpolation polynomial at the points $1,2,3,4$ is

$$\begin{aligned}
S(n) &= f(1) + \Delta f(1)(n-1) + \frac{\Delta^2 f(1)}{2}(n-1)(n-2) + \frac{\Delta^3 f(1)}{3!}(n-1)(n-2)(n-3) \\
&= 1 + 4(n-1) + \frac{5}{2}(n-1)(n-2) + \frac{2}{6}(n-1)(n-2)(n-3) \\
&= \frac{n}{6} + \frac{n^2}{2} + \frac{n^3}{3} \\
&= \frac{n(n+1)(2n+1)}{6}
\end{aligned}$$

3. Neville's method can be implemented with the following algorithm

```
NEVILLE($x, x_1, x_2, \ldots, x_n, f_1, f_2, \ldots, f_n$)
 1  for $i \leftarrow 1, 2, \ldots, n$
 2  do
 3      $N_{i,1} \leftarrow f_i$
 4      $d_i \leftarrow x - x_i$
 5  for $k \leftarrow 2, 3, \ldots, n$
 6  do
 7      for $i \leftarrow k, k+1, \ldots, n$
 8      do
 9          $N_{i,k} \leftarrow N_{i,k-1} + d_i \left(N_{i,k-1} - N_{i-1,k-1}\right) / (x_i - x_{i-k+1})$
10  return $N_{n,n}$
```

(a) How many floating-point operations (additions, subtractions, multiplications, and divisions) does this algorithm require?

(b) Modify the algorithm so that only one column of the matrix N is stored in memory instead of the whole matrix.

Solution:

(a) The first loop takes n subtractions. The second loop repeats $(n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$ times, and each time it performs 2 subtractions, 1 addition, 1 multiplication, and 1 division. In total, the algorithm requires $n + n(n-1) = n^2$ subtractions, and $\frac{n(n-1)}{2}$ additions, multiplications, and divisions. The total number of floating-point operations is $n^2 + 3\,\frac{n(n-1)}{2} = \frac{n(5n-3)}{2}$.

(b) To avoid overwriting previously evaluated entities, we need to reverse the inner loop. The modified algorithm takes the form

```
MODIFIED-NEVILLE($x, x_1, x_2, \ldots, x_n, f_1, f_2, \ldots, f_n$)
 1  for $i \leftarrow 1, 2, \ldots, n$
 2  do
 3      $N_i \leftarrow f_i$
 4      $d_i \leftarrow x - x_i$
 5  for $k \leftarrow 2, 3, \ldots, n$
 6  do
 7      for $i \leftarrow n, n-1, \ldots, k$
 8      do
 9          $N_i \leftarrow N_i + d_i\,(N_i - N_{i-1}) / (x_i - x_{i-k+1})$
10  return $N_n$
```

4. (Programming) In this assignment, we revisit nonlinear equations. One method of solving nonlinear equations $f(x) = 0$ is *inverse interpolation*. Taking several initial guesses for the root $c_0, c_1, \ldots, c_n$ and the corresponding function values $f_0, f_1, \ldots, f_n$, the inverse function $x(f)$ is interpolated (i.e., with a polynomial of degree n) and then evaluated at $f = 0$. This produces the next iteration

$$c_{n+1} = P_n(0)\,; \tag{2}$$

$$f_{n+1} = f(c_{n+1})\,. \tag{3}$$

   (a) Show that the method of inverse interpolation with $n = 1$ is equivalent to the secant method.

   (b) Implement both the secant method and the method of quadratic inverse interpolation ($n = 2$).

   (c) Use your programs to solve the equation

$$f(x) = x + e^x = 0\,. \tag{4}$$

   Use the initial guesses $c_0 = -1$ and $c_1 = 0$ in the secant method. Use the initial guesses $c_0 = -2$, $c_1 = -1$, and $c_2 = 0$ in the inverse quadratic interpolation. Perform the computations with the double precision and output tables of the form

n	c_n	$\lvert f_n \rvert$

   Which of the two methods takes less iterations to find the root with the tolerance

$$|f_n| < 10^{-15}\ ?$$

   Which method takes less operations?

Solution:

   (a) Indeed, the inverse linear interpolation of the table

c_{n-1}	f_{n-1}
c_n	f_n

   is (in the Newton form):

$$c(f) = c_{n-1} + \frac{c_n - c_{n-1}}{f_n - f_{n-1}}\,(f - f_{n-1})\,.$$

   Evaluating at zero, we get

$$c_{n+1} = c(0) = c_{n-1} - f_{n-1}\,\frac{c_n - c_{n-1}}{f_n - f_{n-1}} = c_n - f_n\,\frac{c_n - c_{n-1}}{f_n - f_{n-1}}\,.$$

   The last form is equivalent to the usual definition of the secant method.

(b) C program:

```
#include <stdio.h>  /* for output */
#include <math.h>   /* for mathematical functions */

/* function: secant
   ----------------
   Implements Secant method
   func       - a pointer to a function
   c0, c1     - initial values
   xtol, ftol - tolerance in position and value
   nmax       - maximum number of iterations
*/
double secant(double (*func)(double),
      double c0, double c1,
      double xtol, double ftol, int nmax)
{
  int n;
  double f0, f1, c;

  f0 = func(c0);

  for (n=0; n < nmax; n++) {
    f1 = func(c1);

    /* print out the table */
    printf("n=%d c=%f |f(c)|=%e\n",n,c1,fabs(f1));

    /* return if the root is located to the tolerance */
    if (fabs(c1-c0) <= xtol && fabs(f1) <= ftol) return c1;

    if (c0 == c1 || f0 == f1) {
      fprintf(stderr,"Error: The line is degenerate\n");
      return c1;
    }

    c = c1 - f1*(c1-c0)/(f1-f0);

    c0 = c1;
    c1 = c;

    f0 = f1;
  }

  fprintf(stderr,"Warning: Exact root is not found after %d iterations\n",
  nmax);
  return c1;
}

/* function: invint
   ----------------
   Implements inverse quadratic interpolation
   func       - a pointer to a function
   c0, c1, c2 - initial values
   xtol, ftol - tolerance in position and value
   nmax       - maximum number of iterations
*/
double invint(double (*func)(double),
      double c0, double c1, double c2,
      double xtol, double ftol, int nmax)
{
  int n;
  double f0, f1, f2, f10, f20, f21, c;
  double n1, n2;

  f0 = func(c0);
  f1 = func(c1);
  f10 = f1 - f0;

  if (c0 == c1 || f10 == 0.) {
    fprintf(stderr,"Error: The parabola is degenerate\n");
    return c1;
  }

  for (n=0; n < nmax; n++) {
    f2 = func(c2);
    f20 = f2 - f0;
    f21 = f2 - f1;

    /* print out the table */
    printf("n=%d c=%f |f(c)|=%e\n",n,c2,fabs(f2));

    /* return if the root is located to the tolerance */
    if (fabs(c2-c1) <= xtol && fabs(f2) <= ftol) return c2;

    if (c0 == c2 || c1 == c2 || f20 == 0. || f21 == 0.) {
      fprintf(stderr,"Error: The parabola is degenerate\n");
      return c2;
    }

    /* Neville’s method for quadratic interpolation */
    n1 = c1 - f1*(c1-c0)/f10;
    n2 = c2 - f2*(c2-c1)/f21;
    c  = n2 - f2*(n2-n1)/f20;

    c0 = c1;
    c1 = c2;
    c2 = c;

    f0 = f1;
    f1 = f2;
    f10 = f21;
   }

  fprintf(stderr,"Warning: Exact root is not found after %d iterations\n",
  nmax);
  return c2;
}


/* test function */
static double function (double x)
{
  return (x + exp(x));
}

/* main program */
int main (void)
```

```
{
  int nmax=20; /* maximum number of iterations */
  double xtol=1.e-7, ftol=1.e-15, c2=-2., c1=-1., c0=0., c;

  c = secant(&function, c1, c0, xtol, ftol, nmax);
  c = invint(&function, c2, c1, c0, xtol, ftol, nmax);

  return 0;
}
```

(c) Secant method:

n	c_n	$\lvert f_n \rvert$
1	0.000000	1.000000e+00
2	-0.612700	7.081395e-02
3	-0.572181	7.888273e-03
4	-0.567102	6.458283e-05
5	-0.567143	5.883093e-08
6	-0.567143	4.387601e-13
7	-0.567143	0.000000e+00

Inverse quadratic interpolation:

n	c_n	$\lvert f_n \rvert$
2	0.000000	1.000000e+00
3	-0.568870	2.704940e-03
4	-0.567140	4.689793e-06
5	-0.567143	5.394762e-11
6	-0.567143	1.110223e-16

The second method takes 6 iterations to converge to the specified tolerance. The inverse quadratic interpolation method takes only 4 iterations. However, each iteration of quadratic interpolation uses about 3 times more operations. Therefore, in this example, the number of operations is higher in the second case.

5. (Programming) The following table contains the number of medals won by the United States at the winter Olympic games:

Year	Location	Gold	Silver	Bronze	Total	Points
1924	CHAMONIX	1	2	1	4	8
1928	SAINT MORITZ	3	2	2	7	15
1932	LAKE PLACID	6	4	2	12	28
1936	GARMISH PARTENKIRCHEN	1	0	3	4	6
1948	SAINT MORITZ	3	5	2	10	21
1952	OSLO	4	6	1	11	25
1956	CORTINA D'AMPEZZO	2	3	2	7	14
1960	SQUAW VALLEY	3	4	3	10	20
1964	INNSBRUCK	1	2	4	7	11
1968	GRENOBLE	1	4	1	6	12
1972	SAPPORO	3	2	3	8	16
1976	INNSBRUCK	3	3	4	10	19
1980	LAKE PLACID	6	4	2	12	28
1984	SARAJEVO	4	4	0	8	20
1988	CALGARY	2	1	3	6	11
1992	ALBERTVILLE	5	4	2	11	25
1994	LILLEHAMMER	6	5	2	13	30
1998	NAGANO	6	3	4	13	28

The points are computed with the formula

$$\text{Points} = 3 \times \text{Gold} + 2 \times \text{Silver} + \text{Bronze}\ .$$

(a) Implement the Neville interpolation algorithm.

(b) Apply it to predict the number of points that US would have won if the Olympic games took place in 1944. First, use the points from all the years in the table to do the interpolation. Next, use only four values from the years 1932–1952. Output the Neville table (the matrix $N_{i,j}$) for the second case.

(c) Apply your program to predict the number of points that US will get in 2002. First, use the points from all the years in the table. Next, use only the last five values from the years 1984-1998. Output the Neville table for the second case. Do you think this is a reliable prediction? Explain.

Solution:

(a) C program:

```
#include <stdio.h> /* for output */


/* function: neville
   -----------------
   Polynomial interpolation by Neville's algorithm
   n              - number of points
   x              - where to evaluate
   xk[n]          - nodes
   fk[n]          - function values
   nev[n], dif[n] - required internal storage
   verb           - verbosity flag
*/
double neville (int n, double x, double* xk, double* fk,
double* nev, double* dif, int verb)
{
  int i, k;

  for (i=0; i < n; i++) {
    nev[i] = fk[i];
    dif[i] = x-xk[i];
    if (verb) printf("%f ", nev[i]);
  }
  if (verb) printf("\n");
  for (k=1; k < n; k++) {
    for (i=n-1; i >= k; i--) {
      nev[i] += (nev[i]-nev[i-1])*dif[i]/(xk[i]-xk[i-k]);
      if (verb) printf("%f ", nev[i]);
    }
    if (verb) printf("\n");
  }

  return nev[n-1];
}

/* number of nodes */
#define NODES 18

/* main program */
int main (void)
{
  const int n = NODES;
  double years[] = { 24.,28.,32.,36.,48.,52.,56.,60.,64.,
  68.,72.,76.,80.,84.,88.,92.,94.,98.};
  double points[] = { 8., 15.,28.,6. ,21.,25.,14.,20.,11.,
  12.,16.,19.,28.,20.,11.,25.,30.,28.};
  double nev[NODES], dif[NODES], year1=44., year2=102., f;

  f = neville(n, year1, years, points, nev, dif, 0);
  printf("%f %f \n",year1,f);

  f = neville(4, year1, years+2, points+2, nev, dif, 1);
  printf("%f %f \n",year1,f);

  f = neville(n, year2, years, points, nev, dif, 0);
  printf("%f %f \n",year2,f);

  f = neville(5, year2, years+n-5, points+n-5, nev, dif, 1);
  printf("%f %f \n",year2,f);

  return 0;
}
```

(b) Using all the nodes, the number is $-311.110055 \approx -311$ points. This is obviously a wrong prediction. Using only four years, the number is $10.9 \approx 11$ points – a much more realistic number. The corresponding Neville table is

$$N = \begin{bmatrix} 28 & & & \\ 6 & 17 & & \\ 21 & 16 & 16.5 & \\ 25 & -38 & 2.5 & 10.9 \end{bmatrix}$$

(c) Using all the nodes, the number is $6105.626577 \approx 6106$ points: a bit too much for any Olympic games. Using only the last five nodes, the number is more reasonable: 17 points. The corresponding Neville table is

$$N = \begin{bmatrix} 20 & & & & \\ 11 & 26 & & & \\ 25 & 50 & 10 & & \\ 30 & 60 & 36.(6) & -0.(6) & \\ 28 & -20.5 & 160.625 & -62.5 & 17 \end{bmatrix}$$

The last prediction is not reliable. We can conclude that from the Neville table, where there is no sign of convergence. Using four points from years $1984-1994$ or $1988-1998$, the prediction is actually negative (-2/3 and -62.5 correspondingly).

Note: In 2002, the US won $3 \times 10 + 2 \times 13 + 11 = 67$ points at the winter Olympic games in Salt Lake City.