Answers to Homework 12: Systems of Linear Equations

1. (a) A vector norm $||\mathbf{x}||$ has the following properties

   i. $||\mathbf{x}|| \geq 0$ for all $\mathbf{x} \in \mathbb{R}^n$; $||\mathbf{x}|| = 0$ only if $\mathbf{x} = \mathbf{0}$

   ii. $||\alpha\,\mathbf{x}|| = |\alpha|\,||\mathbf{x}||$ for all $\alpha \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{R}^n$

   iii. $||\mathbf{x}+\mathbf{y}|| \leq ||\mathbf{x}|| + ||\mathbf{y}||$ for all $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$

   Prove that these properties are satisfied if a vector norm is defined as

$$||\mathbf{x}||_S = \left(\mathbf{x}^T\,\mathbf{S}\,\mathbf{x}\right)^{1/2}\ , \tag{1}$$

   where $\mathbf{S}$ is a symmetric positive definite matrix.

   Answer:

   The first property follows directly from the definition of positive definite matrices. The second property is verified directly:

$$||\alpha\,\mathbf{x}||_S = \left(\alpha^2\,\mathbf{x}^T\,\mathbf{S}\,\mathbf{x}\right)^{1/2} = |\alpha|\,\left(\mathbf{x}^T\,\mathbf{S}\,\mathbf{x}\right)^{1/2} = |\alpha|\,||\mathbf{x}||_S$$

   To verify the third property, let us examine the square of the norm

$$||\mathbf{x}+\mathbf{y}||_S^2 = (\mathbf{x}+\mathbf{y})^T\ \mathbf{S}\,(\mathbf{x}+\mathbf{y}) = \mathbf{x}^T\,\mathbf{S}\,\mathbf{x} + \mathbf{y}^T\,\mathbf{S}\,\mathbf{y} + 2\,\mathbf{x}^T\,\mathbf{S}\,\mathbf{y}$$

   According to the Cauchy-Buniakowski-Schwartz inequality (Theorem 7.3 in the textbook),

$$\mathbf{x}^T\,\mathbf{S}\,\mathbf{y} = \left(\mathbf{L}^T\,\mathbf{x}\right)^T\,\left(\mathbf{L}^T\,\mathbf{y}\right) \leq \left(\mathbf{x}^T\,\mathbf{S}\,\mathbf{x}\right)^{1/2}\,\left(\mathbf{y}^T\,\mathbf{S}\,\mathbf{y}\right)^{1/2}\ ,$$

   where $\mathbf{L}$ is the Cholesky factor of $\mathbf{S} = \mathbf{L}\,\mathbf{L}^T$. Therefore,

$$||\mathbf{x}+\mathbf{y}||_S^2 \leq \mathbf{x}^T\,\mathbf{S}\,\mathbf{x} + \mathbf{y}^T\,\mathbf{S}\,\mathbf{y} + 2\,\left(\mathbf{x}^T\,\mathbf{S}\,\mathbf{x}\right)^{1/2}\,\left(\mathbf{y}^T\,\mathbf{S}\,\mathbf{y}\right)^{1/2} = \left(||\mathbf{x}||_S + ||\mathbf{y}||_S\right)^2$$

   Both sides of the inequality are squares of positive numbers. Taking the square root, we arrive at the third property.

   (b) A matrix norm $||\mathbf{A}||$ has the following properties

   i. $||\mathbf{A}|| \geq 0$ for all $n \times n$ matrices $\mathbf{A}$; $||\mathbf{A}|| = 0$ only if $\mathbf{A} = \mathbf{0}$

   ii. $||\alpha\,\mathbf{A}|| = |\alpha|\,||\mathbf{A}||$

   iii. $||\mathbf{A}+\mathbf{B}|| \leq ||\mathbf{A}|| + ||\mathbf{B}||$

   iv. $||\mathbf{A}\,\mathbf{B}|| \leq ||\mathbf{A}||\,||\mathbf{B}||$

   Prove that these properties are satisfied for the *Frobenius* norm

$$||\mathbf{A}||_F = \left(\sum_{i=1}^{n}\sum_{j=1}^{n} |a_{ij}|^2\right)^{1/2} \tag{2}$$

Answer:

The first two properties follow directly from the definition. To prove the third property, consider the mapping

$$a_{ij} = x_{i+(j-1)n}$$

that associates matrix $\mathbf{A}$ with vector $\mathbf{x}$ of length n^2. Then

$$||\mathbf{A}||_F = \left(\mathbf{x}^T\,\mathbf{x}\right)^{1/2} = ||\mathbf{x}||_2$$

and the third property follows from the triangle inequality for the Euclidian vector norm. To prove the fourth property, let us examine the square of the norm

$$||\mathbf{AB}||_F^2 = \sum_{i=1}^{n}\sum_{j=1}^{n}\left|\sum_{k=1}^{n} a_{ik}\,b_{kj}\right|^2 .$$

According to the Cauchy-Buniakowski-Schwartz inequality,

$$\left|\sum_{k=1}^{n} a_{ik}\,b_{kj}\right|^2 \le \left(\sum_{k=1}^{n} a_{ik}^2\right)\left(\sum_{k=1}^{n} b_{kj}^2\right) .$$

Therefore,

$$||\mathbf{AB}||_F^2 \le \sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sum_{k=1}^{n} a_{ik}^2\right)\left(\sum_{k=1}^{n} b_{kj}^2\right) = \sum_{i=1}^{n}\sum_{k=1}^{n} a_{ik}^2\left(\sum_{j=1}^{n}\sum_{l=1}^{n} b_{lj}^2\right) = ||\mathbf{A}||_F^2\,||\mathbf{B}||_F^2 .$$

Taking the square root of both sides, we arrive at the fourth property.

2. Triangular matrix (**LU**) decomposition requires $\frac{n(n-1)(2n-1)}{6}$ multiplications. Triangular matrix inversion (**L** or **U**) requires $\frac{n(n-1)}{2}$ multiplications.

   (a) Find the number of multiplications necessary for solving two linear systems

   $$\mathbf{A}\,\mathbf{x} = \mathbf{b}_1 , \quad \mathbf{A}\,\mathbf{x} = \mathbf{b}_2 \tag{3}$$

   with the non-singular square matrix $\mathbf{A}$.

   Answer:

   We need to perform **LU** decomposition only once and perform L and U inversion twice for each new right-hand side. Therefore, the total number of multiplications is

   $$\frac{n(n-1)(2n-1)}{6} + 2n(n-1) = \frac{n(n-1)(2n+11)}{6}$$

   (b) Find the number of multiplications necessary for solving two linear systems

   $$\mathbf{A}_1\,\mathbf{x} = \mathbf{b} , \quad \mathbf{A}_2\,\mathbf{x} = \mathbf{b} , \tag{4}$$

   where $\mathbf{A}_1$ an $\mathbf{A}_2$ are non-singular square matrices that differ only by one element:

   $$\mathbf{A}_2 - \mathbf{A}_1 = \alpha\,\mathbf{e}_i\,\mathbf{e}_j^T , \tag{5}$$

where $\mathbf{e}_i$ is the i-th column of the identity matrix.

Hint: Recall the Sherman-Morrison formula

$$\left(\mathbf{A}+\mathbf{u}\mathbf{v}^T\right)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1+\mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}} \,. \tag{6}$$

Answer:

Let $y = \mathbf{A}_1^{-1}\mathbf{b}$ be the solution of the first system. multiplication. Let $z = \mathbf{A}_1^{-1}\mathbf{e}_i$. As we know from the previous problem, computation of $\mathbf{y}$ and $\mathbf{z}$ requires $\frac{n(n-1)(2n+11)}{6}$ multiplications. According to the Sherman-Morrison formula, the solution of the second system is

$$\mathbf{A}_2^{-1}\mathbf{b} = \left(\mathbf{A}^{-1} - \frac{\alpha\,\mathbf{A}^{-1}\mathbf{e}_i\,\mathbf{e}_j^T\,\mathbf{A}^{-1}}{1+\alpha\,\mathbf{e}_j^T\,\mathbf{A}^{-1}\mathbf{e_i}}\right)\mathbf{b} = \mathbf{y} - \mathbf{z}\,\frac{\alpha\,\mathbf{e}_j^T\,\mathbf{y}}{1+\alpha\,\mathbf{e}_j^T\,\mathbf{z}} = \mathbf{y} - \mathbf{z}\,\frac{y_j}{1/\alpha+z_j} \,.$$

This formula shows that only n additional multiplications are required to compute the solution of the second system once $\mathbf{y}$ and $\mathbf{z}$ are known. The total number of multiplications is

$$\frac{n(n-1)(2n+11)}{6} + n = \frac{n(n+5)(2n-1)}{6}$$

3. The following algorithm can be used for solving the equation

$$\mathbf{U}\mathbf{x} = \mathbf{b} \,, \tag{7}$$

where $\mathbf{U}$ is an upper triangular matrix:

```
UPPER TRIANGULAR(U, x)
1  for k ← n, n−1, ..., 1
2  do
3     for i ← k+1, k+2, ..., n
4     do
5        x_k ← x_k − u_{k,i} x_i
6     x_k ← x_k / u_{k,k}
```

The algorithm is initialized with the right-hand-side vector $\mathbf{b}$ and overwrites its elements with the elements of the solution vector $\mathbf{x}$. The analogous algorithm for a lower triangular matrix is

```
LOWER TRIANGULAR(L, x)
1  for k ← 1, 2, ..., n
2  do
3     for i ← 1, 2, ..., k−1
4     do
5        x_k ← x_k − l_{k,i} x_i
6     x_k ← x_k / l_{k,k}
```

Both algorithms access the elements of the corresponding matrices *by row* and use the inner loop to compute the dot product of two vectors. On many computers, the dot product algorithm

```
DOT PRODUCT(a, x, y)
1  for i ← 1, 2, ..., n
2  do
3      a ← a + x_i y_i
4
```

is less efficient than the scaled vector addition algorithm

```
ADD SCALED VECTOR(a, x, y)
1  for i ← 1, 2, ..., n
2  do
3      y_i ← y_i + a x_i
```

because the latter can be easily performed in parallel.

Modify the upper and lower triangular inversion algorithms so that they access the corresponding matrices *by column*. Show that this transforms the dot product algorithm in the inner loop to the scaled vector addition algorithm.

Answer:

```
UPPER TRIANGULAR 2(U, x)
1  for i ← n, n − 1, ..., 1
2  do
3      x_i ← x_i / u_{i,i}
4      for k ← 1, 2, ..., i − 1
5      do
6          x_k ← x_k − u_{k,i} x_i
```

```
UPPER TRIANGULAR 3(U, x)
1  for i ← n, n − 1, ..., 1
2  do
3      x_i ← x_i / u_{i,i}
4      ADD SCALED VECTOR(−x_i, u_{1,i}, ..., u_{i−1,i}, x_1, ..., x_{i−1})
```

```
LOWER TRIANGULAR 2(L, x)
1  for i ← 1, 2, ..., n
2  do
3      x_i ← x_i / l_{i,i}
4      for k ← i + 1, i + 2, ..., n
5      do
6          x_k ← x_k − l_{k,i} x_i
```

LOWER TRIANGULAR 3($\mathbf{L}, \mathbf{x}$)
1 **for** $i \leftarrow 1, 2, \ldots, n$
2 **do**
3 $\quad x_i \leftarrow x_i / l_{i,i}$
4 $\quad$ ADD SCALED VECTOR($-x_i, l_{i+1,i}, \ldots, l_{n,i}, x_{i+1}, \ldots, x_n$)

4. (Programming) In this assignment, we revisit the method of least-squares polynomial approximation. Consider a function $f(x)$, measured at a number of points $x_1, x_2, \ldots, x_n$ and approximated with the sum of Chebyshev polynomials

$$f(x_i) \approx \sum_{k=0}^{m} c_k \, T_k(x_i) \,, \quad i = 1, 2, \ldots, n \tag{8}$$

In the matrix form, the system of approximate equations is

$$\mathbf{f} \approx \mathbf{A}\,\mathbf{c} \,, \tag{9}$$

where $\mathbf{f}$ is the vector with elements $f(x_i)$, $\mathbf{c}$ is the vector with the elements c_k, and the $n \times (m+1)$ matrix $\mathbf{A}$ has the elements $a_{ik} = T_k(x_i)$. The method of least squares leads to the square system

$$\mathbf{A}^T \mathbf{A}\,\mathbf{c} = \mathbf{A}^T \mathbf{f} \,, \tag{10}$$

which can be solved for the unknown coefficient vector $\mathbf{c}$. The approximation algorithm consists of the following steps:

(a) Form the matrix $\mathbf{A}$.

CHEBYSHEV MATRIX($\mathbf{x}, \mathbf{A}$)
1 **for** $i \leftarrow 1, 2, \ldots, n$
2 **do**
3 $\quad a_{i,0} \leftarrow 1$
4 $\quad a_{i,1} \leftarrow x_i$
5 $\quad$ **for** $k \leftarrow 1, 2, \ldots, m-1$
6 $\quad$ **do**
7 $\quad\quad a_{i,k+1} \leftarrow 2\,x_i\,a_{i,k} - a_{i,k-1}$

(b) Form the normal matrix $\mathbf{C} = \mathbf{A}^T \mathbf{A}$.

NORMAL MATRIX($\mathbf{A}, \mathbf{C}$)
1 **for** $i \leftarrow 0, 1, 2, \ldots, m$
2 **do**
3 $\quad$ **for** $j \leftarrow 0, 1, 2, \ldots, i$
4 $\quad$ **do**
5 $\quad\quad c_{i,j} \leftarrow 0$
6 $\quad\quad$ **for** $k \leftarrow 1, 2, \ldots, n$
7 $\quad\quad$ **do**
8 $\quad\quad\quad c_{i,j} \leftarrow c_{i,j} + a_{k,i}\,a_{k,j}$

The algorithm fills only the lower triangular part of $\mathbf{C}$.

(c) Form the right-hand side $\mathbf{b} = \mathbf{A}^T \mathbf{f}$

RIGHT-HAND SIDE($\mathbf{A}, \mathbf{f}, \mathbf{b}$)

```
1  for k ← 0,1,2,...,m
2  do
3     b_k ← 0
4     for i ← 1,2,...,n
5     do
6        b_k ← b_k + a_{i,k} f_i
```

(d) Cholesky factorization of $\mathbf{C} = \mathbf{L}\mathbf{L}^T$

CHOLESKY($\mathbf{C}$)

```
1  for k ← 0,1,2,...,m
2  do
3     c_{k,k} = √c_{k,k}
4     for i ← k+1,k+2,...,m
5     do
6        c_{i,k} = c_{i,k}/c_{k,k}
7     for j ← k+1,k+2,...,m
8     do
9        for i ← j, j+1,...,m
10       do
11          c_{i,j} = c_{i,j} − c_{i,k} c_{j,k}
```

The algorithm overwrites the lower triangular part of $\mathbf{C}$ with $\mathbf{L}$.

(e) Upper and lower triangular inversion using the Cholesky factor $\mathbf{L}$ and the right-hand side $\mathbf{b}$. The output is the coefficient vector $\mathbf{c}$.

(f) At each point x, evaluate the approximation using the fast algorithm from Homework 7:

CHEBYSHEV SUM($x, \mathbf{c}$)

```
1  ĉ_1 ← 0
2  ĉ_0 ← c_n
3  for k ← n−1,n−2,...,0
4  do
5     t ← ĉ_1
6     ĉ_1 ← ĉ_0
7     ĉ_0 ← c_k + 2x ĉ_0 − t
8  return (ĉ_0 − x ĉ_1)
```

Using the data from Homework 5

y	24	28	32	36	48	52	56	60	64	68	72	76	80	84	88	92	94	98
p	8	15	28	6	21	25	14	20	11	12	16	19	28	20	11	25	30	28

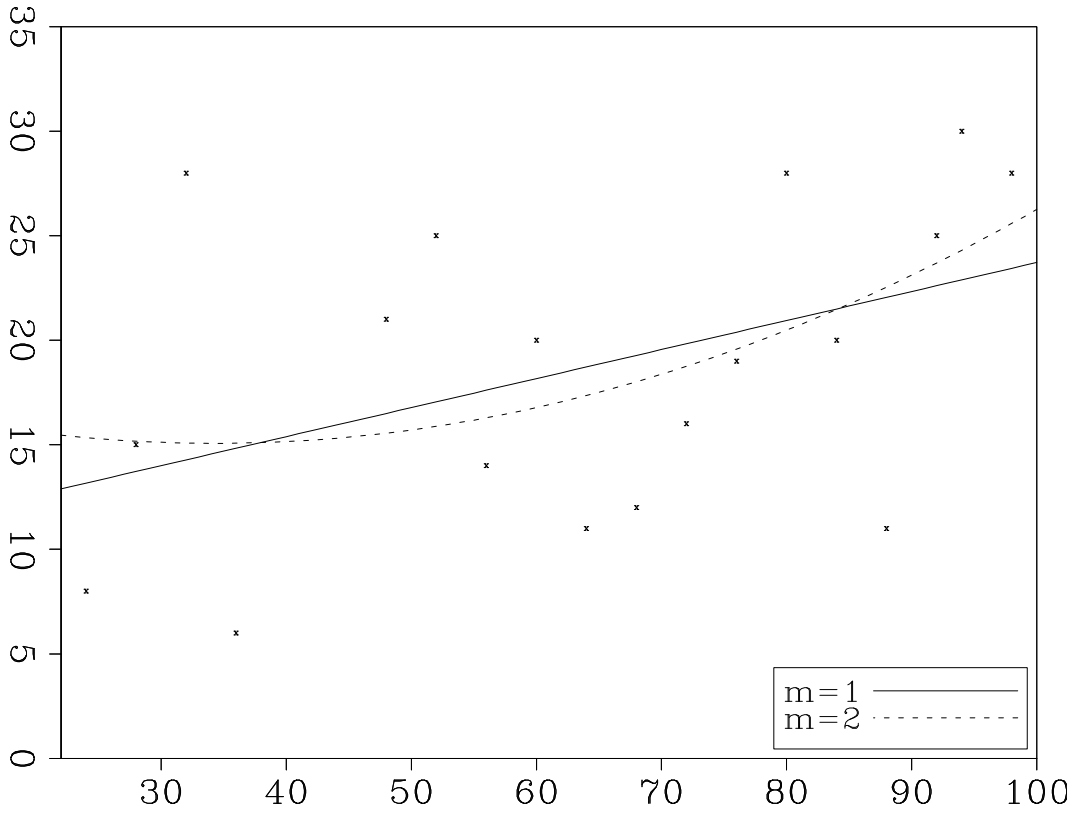
where y stands for the year, and p stands for the number of Olympic points, perform the steps above to approximate the dependence $p(y)$. Output the coefficient vector $\mathbf{c}$ and the Cholesky factor $\mathbf{L}$ for $m = 1, 2, 4, 8$ and plot the corresponding approximations in the interval $[22, 100]$ with the step size of one year.

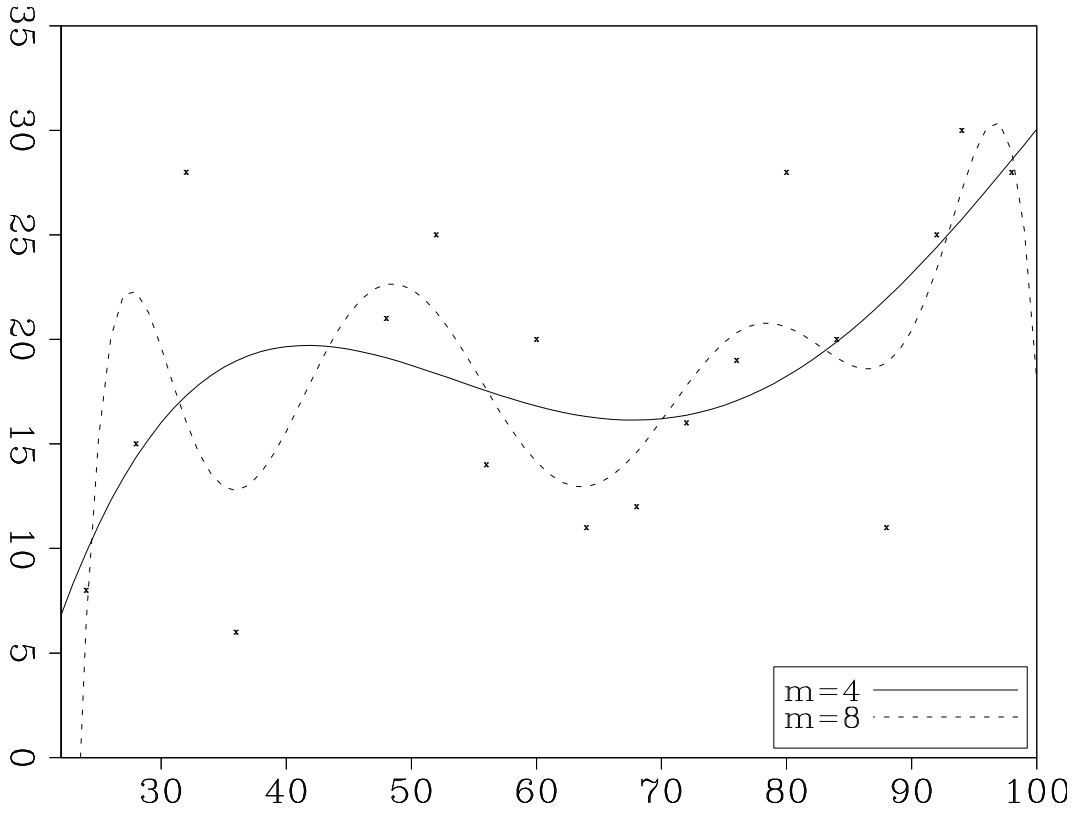
Answer:

$$
\begin{aligned}
\mathbf{c}_1^T &= \begin{bmatrix} 9.83783 & 0.138819 \end{bmatrix} \\
\mathbf{c}_2^T &= \begin{bmatrix} 18.1214 & -0.177767 & 0.00129571 \end{bmatrix} \\
\mathbf{c}_4^T &= \begin{bmatrix} -73.1649 & 6.21731 & -0.0729484 & 0.000347973 & -5.6431 \times 10^{-7} \end{bmatrix} \\
\mathbf{c}_8^T &= \begin{bmatrix} -13888.4 & 2221.86 & -74.9958 & 1.3957 & -0.0156843 \\
& 0.000109204 & -4.61121 \times 10^{-7} & 1.08217 \times 10^{-9} & -1.08315 \times 10^{-12} \end{bmatrix}
\end{aligned}
$$

$$
\mathbf{L} = \begin{bmatrix}
4.242641 \times 10^{0} & & & & \\
2.715290 \times 10^{2} & 9.736529 \times 10^{1} & & & \\
3.922038 \times 10^{4} & 2.378967 \times 10^{4} & 4.010749 \times 10^{3} & & \\
6.111846 \times 10^{6} & 4.732823 \times 10^{6} & 1.476421 \times 10^{6} & 1.419777 \times 10^{5} & \\
9.995063 \times 10^{8} & 8.919074 \times 10^{8} & 3.827343 \times 10^{8} & 7.002275 \times 10^{7} & \\
1.688678 \times 10^{11} & 1.653963 \times 10^{11} & 8.659300 \times 10^{10} & 2.253272 \times 10^{10} & \\
2.920551 \times 10^{13} & 3.055812 \times 10^{13} & 1.832904 \times 10^{13} & 6.021290 \times 10^{12} & \\
5.140707 \times 10^{15} & 5.650301 \times 10^{15} & 3.740640 \times 10^{15} & 1.454147 \times 10^{15} & \\
9.173213 \times 10^{17} & 1.047330 \times 10^{18} & 7.472325 \times 10^{17} & 3.302006 \times 10^{17} &
\end{bmatrix}
$$

$$
\begin{bmatrix}
5.663697 \times 10^{6} & & & & \\
3.450622 \times 10^{9} & 2.271413 \times 10^{8} & & & \\
1.308436 \times 10^{12} & 1.672057 \times 10^{11} & 7.717860 \times 10^{9} & & \\
3.987537 \times 10^{14} & 7.395130 \times 10^{13} & 6.663206 \times 10^{12} & 2.880291 \times 10^{11} & \\
1.072508 \times 10^{17} & 2.560097 \times 10^{16} & 3.371156 \times 10^{15} & 2.813037 \times 10^{14} & 1.076703 \times 10^{13}
\end{bmatrix}
$$


0
5
10
15
20
25
30
35
30
40
50
60
70
80
90
100
m=1
m=2


0
5
10
15
20
25
30
35
30
40
50
60
70
80
90
100
m=4
m=8

Solution:

C program:

```
#include <math.h>
#include <stdio.h>
#include <assert.h>
#include <stdlib.h>

/* Function: chebyshev_matrix
   -------------------------
   Form the matrix for Chebyshev approximation
   n       - number of data points
   m       - number of coefficients
   x[n]    - data coordinates
   a[n][m] - output Chebyshev matrix
*/
void chebyshev_matrix (int n, int m, const double x[], double **a)
{
    int i, k;

    for (i=0; i < n; i++) {
        a[i][0] = 1.;
        a[i][1] = x[i];
        for (k=1; k < m-1; k++) { /* use Chebyshev recursion */
            a[i][k+1] = 2.*x[i]*a[i][k] - a[i][k-1];
        }
    }
}

/* Function: normal_matrix
   ----------------------
   Form normal matrix C = A'A
   n, m - matrix dimensions
   a[n][m] - input
   c[m][m] - output
   Only the lower triangular portion of c is filled
*/
void normal_matrix (int n, int m, double** a, double** c)
{
    double p;
    int i, j, k;

    for (i=0; i < m; i++) {
        for (j=0; j <= i; j++) {
            p = 0.;
            for (k=0; k < n; k++) {
                p += a[k][i]*a[k][j];
            }
            c[i][j] = p;
        }
    }
}

/* Function: normal_matrix
   ----------------------
   Form right-hand side b = A'f
   a[n][m] - matrix
   f[n]    - input
```

```
   b[m]    - output
*/
void righthand_side (int n, int m, double** a, const double* f, double* b)
{
    int i, k;

    for (k=0; k < m; k++) {
        b[k] = 0.;
        for (i=0; i < n; i++) {
          b[k] += a[i][k]*f[i];
        }
    }
}

/* Function: cholesky
   ------------------
   Cholesky decomposition
   n - matrix dimensions
   c[n][n] - matrix (input is overwritten with output
   Only the lower triangular portion of c is accessed */
void cholesky (int n, double** c)
{
    int i,j,k;
    double p;

    for (k=0; k < n; k++) {
        p = c[k][k];
        assert (p > 0); /* check singularity */
        p = 1./sqrt(p);

        for (i=k; i < n; i++) {
            c[i][k] *= p;
        }
        for (j=k+1; j < n; j++) {
            for (i=j; i < n; i++) {
                c[i][j] -= c[i][k]*c[j][k];
            }
        }
    }
}

/* Function: lu_inversion
   ----------------------
   LU inversion: solves LL' x = b where L is lower triangular
   Algorithm uses vector dot products.
   n       - matrix dimensions
   l[n][n] - matrix
   x[n]    - right-hand side overwritten with the solution */
void lu_inversion (int n, double** l, double* x)
{
    int i, k;

    for (k=0; k < n; k++) {
        for (i=0; i < k; i++) {
            x[k] -= l[k][i]*x[i];
        }
        x[k] /= l[k][k];
    }
    for (k=n-1; k >= 0; k--) {
```

```
        for (i=k+1; i < n; i++) {
            x[k] -= l[i][k]*x[i];
        }
        x[k] /= l[k][k];
    }
}


/* Function: lu_inversion2
   ----------------------
   LU inversion: solves LL' x = b where L is lower triangular
   Alternative algorithm using scaled vector addition.
   n       - matrix dimensions
   l[n][n] - matrix
   x[n]    - right-hand side overwritten with the solution */
void lu_inversion2 (int n, double** l, double* x)
{
    int i, k;

    for (i=0; i < n; i++) {
        x[i] /= l[i][i];
        for (k=i+1; k < n; k++) {
            x[k] -= l[k][i]*x[i];
        }
    }
    for (i=n-1; i >= 0; i--) {
        x[i] /= l[i][i];
        for (k=0; k < i; k++) {
            x[k] -= l[i][k]*x[i];
        }
    }
}

/* Function: chebyshev_sum
   ----------------------
   Evaluates Chebyshev polynomial representation
   f(x) = sum(k=0,n-1) C_k*T_k(x)
   n    - number of coefficients
   x    - where to evaluate
   c[n] - coefficients
*/
double chebyshev_sum (int n, double x, const double c[])
{
    int k;
    double t, c0, c1;

    c1 = 0;
    c0 = c[n-1];
    for (k=n-2; k >= 0; k--) {
        t = c1;
        c1 = c0;
        c0 = c[k] + 2.*x*c0 - t;
    }

    return (c0 - x*c1);
}

/* Function: print_triangular
   -------------------------
```

```
   Prints a lower triangular matrix
   n       - matrix dimension
   a[n][n] - matrix */
void print_triangular (int n, double** a)
{
    int i, j;

    fprintf(stderr,"-------------\n");
    for (i=0; i < n; i++) {
        for (j=0; j <= i; j++) {
            fprintf(stderr,"%e ",a[i][j]);
        }
        fprintf(stderr,"\n");
    }
    fprintf(stderr,"-------------\n");
}

/* main program */
int main (void)
{
  double *a[18], *c[9], b[9], s;
  double x[] = {24,28,32,36,48,52,56,60,64,68,72,76,80,84,88,92,94,98};
  double f[] = { 8,15,28, 6,21,25,14,20,11,12,16,19,28,20,11,25,30,28};
  int i, k, n=18, m=9, mm[] = {2,3,5,9};

  /* Allocate space for Chebyshev matrix a */
  for (k=0; k < n; k++) {
    assert (a[k] = (double*) malloc(m*sizeof(double)));
  }

  /* Allocate space for normal matrix c: only lower triangular part */
  for (k=0; k < m; k++) {
    assert (c[k] = (double*) malloc((k+1)*sizeof(double)));
  }

  chebyshev_matrix(n,m,x,a); /* Form a */
  normal_matrix(n,m,a,c);    /* Form c */
  cholesky (m, c);           /* Cholesky decomposition of c */
  print_triangular (m,c);    /* Print the Cholesky factor */

  for (i=0; i < 4; i++) {
    m = mm[i];

    righthand_side(n,m,a,f,b); /* Form b */
    lu_inversion2 (m,c,b);     /* Find Chebyshev coefficients */
    for (k=0; k < m; k++) {
      fprintf(stderr,"%g ",b[k]);
    }
    fprintf(stderr,"\n");

    for (k=22; k <= 100; k++) {
      s = chebyshev_sum (m, (double) k, b); /* Evaluate function */
      printf("%f\n",s);
    }
  }

  exit(0);
}
```