Least Squares Linear Regression

Kevin Juan (kj89)

November 7th 2018

The Numerical Methods and Algorithms

The objective of this assignment was to implement least squares linear regression on a data set by solving a matrix equation using Gauss-Jordan elimination. Given a set of measured data points $\{x_i, y_i\}$ along with their associated measurement errors $\{\sigma_i\}$, we seek to find an optimal function, $f(x)$, that fits the data $\{x_i, y_i \pm \sigma_i\}$. Typically, $f(x) \to f(x, \vec{a})$, where $\vec{a}$ is a vector of adjustable fitting parameters with length m. We note that $m < N$ where N is denoted as the number of data points in the set. Our fitting function can be a linear combination of functions as shown below:

$$f(x, \vec{a}) = \sum_{k=1}^{m} a_k f_k(x) \tag{1}$$

The random nature of the error in the data suggests a probabilistic interpretation, which allows us to assume the data point is normally or Gaussian distributed, $X \sim \mathcal{N}(\mu, \sigma^2)$. The Central Limit Theorem says that the average properties of a large set of random variables becomes Gaussian. The probability density for $p(y_i|f(x_i))$ is shown below:

$$p(y_i|f(x_i)) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(y_i - f(x_i))^2}{2\sigma_i}} \tag{2}$$

The probability of the curve we seek to fit is given by the likelihood function, $\mathcal{L}$, assuming independence as shown below:

$$\mathcal{L} = \prod_{i=1}^{N} p(y_i) = \frac{1}{(2\pi)^{N/2}} \prod_{i=1}^{N} \frac{1}{\sigma_i} e^{-\frac{(y_i - f(x_i))^2}{2\sigma_i}} \tag{3}$$

The best fit curve is given when $\mathcal{L} = \max$ to give the most probable set of parameters $\vec{a}$. Due to the complexity of the likelihood function, we can transform it by taking the logarithm. Because the logarithm is a monotonic function, we can find the maximum likelihood by finding the maximum of the log-likelihood. Our log-likelihood function is shown below:

$$\ln \mathcal{L} = -\frac{N}{2} \ln 2\pi - \sum_{i=1}^{N} \ln \sigma_i - \frac{1}{2} \sum_{i=1}^{N} \left[\frac{y_i - f(y_i)}{\sigma_i} \right]^2 = \max \tag{4}$$

The first two terms of the log-likelihood function are independent of $\vec{a}$. We can take the derivative of $\ln \mathcal{L}$ to solve the maximization problem. A simplification allows us to get Chi-Square as shown below:

$$\chi^2 = \sum_{i=1}^{N} \left[\frac{y_i - f(y_i)}{\sigma_i} \right]^2 = \min \tag{5}$$

Because χ^2 varies with N, we define a reduced Chi-Square below:

$$\chi_r^2 = \frac{1}{N-m} \sum_{i=1}^{N} \left[\frac{y_i - f(y_i)}{\sigma_i} \right]^2 = \min \tag{6}$$

We typically look for $0.5 \lesssim \chi_r^2 \lesssim 2$. For $\chi_r^2 \lesssim 1$, $f(x, \vec{a}$ does not vary by more than σ_i on average. A $\chi_r^2 \ll 1$ means that our understanding of σ_i is incorrect, while a $\chi_r^2 \gg 1$ means that the $f(x)$ is incorrect.

To find our best fit function, we differentiate χ^2 with respect to a_l, where $l = 1, 2, ...m$. We thus get the following problem and its rearrangement:

$$\frac{\partial \chi^2}{\partial a_l} = 2 \sum_{i=1}^{N} \left[\frac{y_i - \sum_{k=1}^{m} a_k f_k(x_i)}{\sigma_i} \right] \left[-\frac{f_l(x_i) y_i}{\sigma_i} \right] \tag{7}$$

$$\sum_{k=1}^{m} a_k \left[\sum_{i=1}^{N} \frac{1}{\sigma_i} f_k(x_i) f_l(x_i) \right] = \sum_{i=1}^{N} \frac{f_l(x_i) y_i}{\sigma_i^2} \tag{8}$$

We see that this problem has the form $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} = \mathbf{F}_{lk}$, $\mathbf{x} = \mathbf{a}_k$, and $\mathbf{b} = \mathbf{b}_l$. The matrix $\mathbf{F}_{lk}$ is an $m \times m$ matrix and $\mathbf{a}_k$ and $\mathbf{b}_l$ are both vectors with length m. The matrix equation can be visualized below

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1m-1} & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2m-1} & a_{2m} \\ & & \vdots & & \\ a_{m1} & a_{m2} & \dots & a_{mm-1} & a_{mm} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{9}$$

The error in the data also impacts the error in the parameters a_k. Assuming independence, a change in every data point y_i of $\Delta y_i = \sigma_i$ gives a variance in parameter a_k of:

$$\sigma_{a_k}^2 = \sum_i \sigma_i^2 \left(\frac{\partial a_k}{\partial y_i} \right)^2 \tag{10}$$

It can be proven that the diagonals of $\mathbf{F}_{lk}^{-1}$, $\mathbf{F}_{kk}^{-1}$ contain the values for $\sigma_{a_k}^2$.

An effective way to find the solution to the matrix equation is to use the Gauss-Jordan elimination method. The algorithm takes in the matrix $\mathbf{F}_{lk}$ and vector $\mathbf{b}_l$ to return the solution vector $\mathbf{a}_k$ stored within $\mathbf{b}_l$. Upon exit of the algorithm, the matrix $\mathbf{F}_{lk}$ is the unit matrix, and is destroyed. The inverse matrix can be found using pivoting, which is implemented in the subroutine `gaussj` provided by Press et al[1]. The Gauss-Jordan elimination algorithm is shown below:

for $i = 1, 2, 3, ...N$ (loop over all rows)

$b_i \leftarrow b_i / a_{ii}$

$a_{ij} \leftarrow a_{ij} / a_{ii}$

for all $j \neq i$ (loop over all rows)

$b_j \leftarrow b_j - a_{ji} b_i$

$a_{jk} \leftarrow a_{jk} - a_{ji} a_{ik}$ for $k = N, ..., i$

end j

end i

More on matrix solving methods and routines can be found in §2.0-2.3 in Press et al[1]. More on least squres regression can be found in §15.0-15.4 in Press et al[1].

Data Set and Fit Function

The data set used for least squares regression was atmospheric CO_2 concentrations in ppm measured every month at La Jolla Pier, California from 1969 to 2007. The data was acquired from `http://cdiac.ornl.gov/ftp/trends/co2/ljo.dat`. The data set is plotted below:

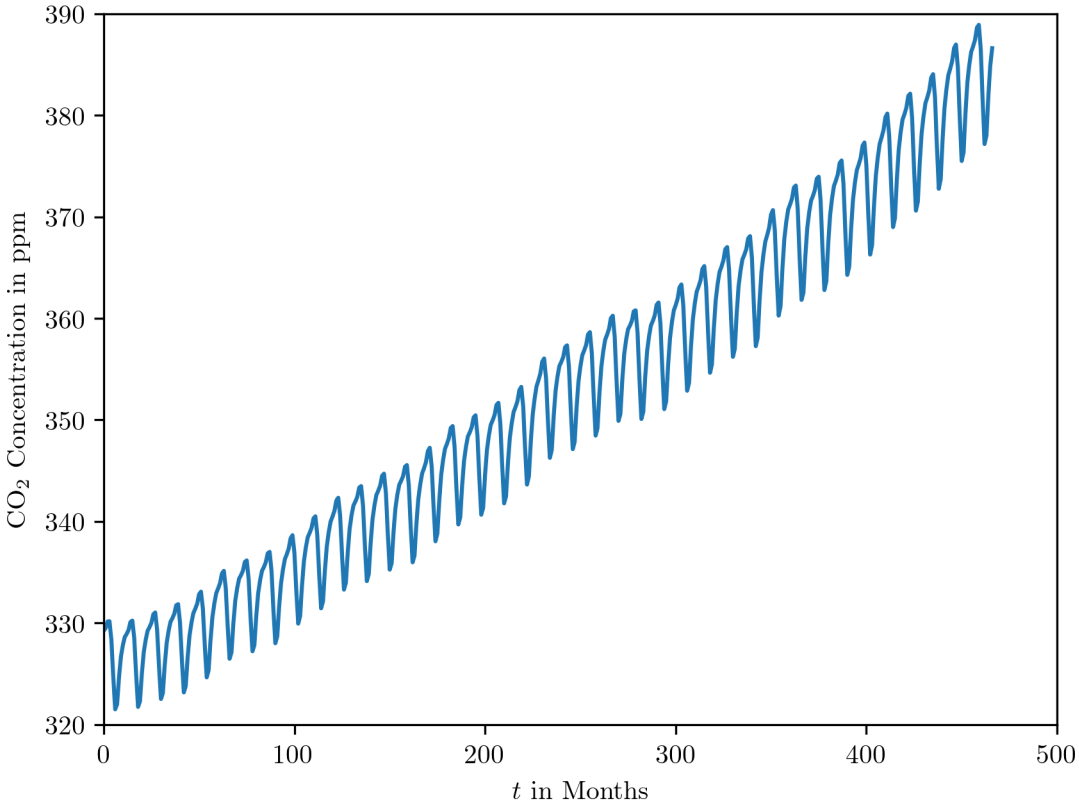

Figure 1: Original data set of monthly atmospheric CO_2 concentration in ppm at La Jolla Pier, CA from 1969 to 2007.

It is clear that there is a seasonal variation in CO_2 concentration that could be due to a number of factors like plant respiration. It was assumed that the data had an error of 0.20% in each y value. Based on the original data set, a fit function would contain some oscillatory terms with a period of 12 months, and would be quadratic in time. Due to the oscillatory nature and the uncertainty of such oscillations, both sine and cosine terms along with their first harmonics could be used as a fitting function. In this case, the first harmonics have periods of 6 months An additional fit function that could be tested would be one that does not include the first harmonics. The proposed 7 parameter and 5 parameter fitting functions are shown below:

$$f(t) = a_1 + a_2 t + a_3 t^2 + a_4 \sin\left(\frac{2\pi t}{12}\right) + a_5 \sin\left(\frac{2\pi t}{6}\right) + a_6 \cos\left(\frac{2\pi t}{12}\right) + a_7 \cos\left(\frac{2t}{6}\right) \quad (11)$$

$$f(t) = a_1 + a_2 t + a_3 t^2 + a_4 \sin\left(\frac{2\pi t}{12}\right) + a_5 \cos\left(\frac{2\pi t}{12}\right) \quad (12)$$

An additional polynomial fit could be fit to eliminate seasonal variations in CO_2 concentrations as shown below:

$$f(t) = a_1 + a_2 t + a_3 t^2 \tag{13}$$

Implementation and Results

A program was written to use the `gaussj` subroutine to solve a matrix equation to find optimal fitting parameters to Equations 11, 12, and 13. The following tables list the optimal parameters for each of these fits along with the corresponding errors.

Parameter	Value	Error
a_1	325.57	0.00851324
a_2	0.0929712	8.95864×10^{-7}
a_3	7.03427×10^{-5}	4.03403×10^{-12}
a_4	2.01576	0.00212294
a_5	-0.588851	0.00211501
a_6	3.88424	0.00210216
a_7	-1.9036	0.00210872

Table 1: Fitting parameters and their errors for Equation 11.

Parameter	Value	Error
a_1	325.541	0.00851232
a_2	0.0930399	8.95816×10^{-7}
a_3	7.03795×10^{-5}	4.03385×10^{-12}
a_4	2.01169	0.0021226
a_5	3.91067	0.00210177

Table 2: Fitting parameters and their errors for Equation 12.

Parameter	Value	Error
a_1	325.564	0.00851098
a_2	0.0927547	8.95735×10^{-7}
a_3	7.0145×10^{-5}	4.03367×10^{-12}

Table 3: Fitting parameters and their errors for Equation 13.

For the 7 parameter fit, a χ^2 of 1.55051 was calculated while a χ^2 value of 5.62282 was calculated for the 5 parameter fit. The following plots show the 7 parameter and 5 parameter fits against the original data set and the polynomial fit.

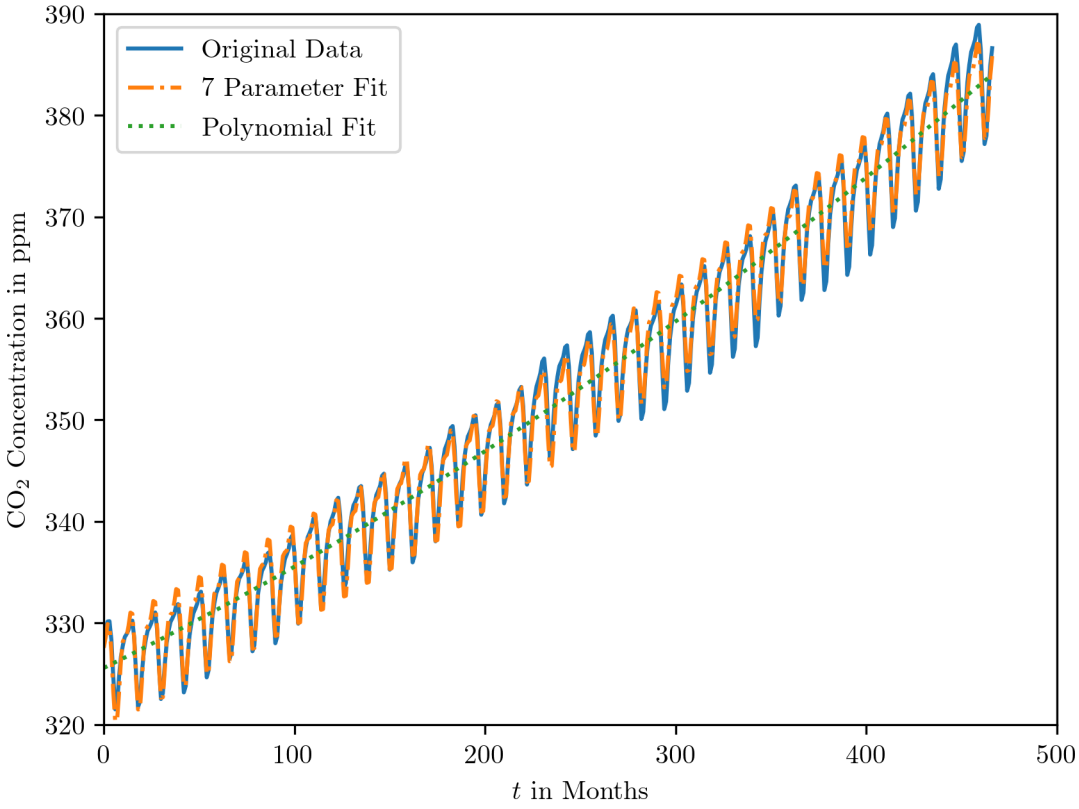

Figure 2: Plot showing the 7 parameter fit against the original data and the polynomial fit.

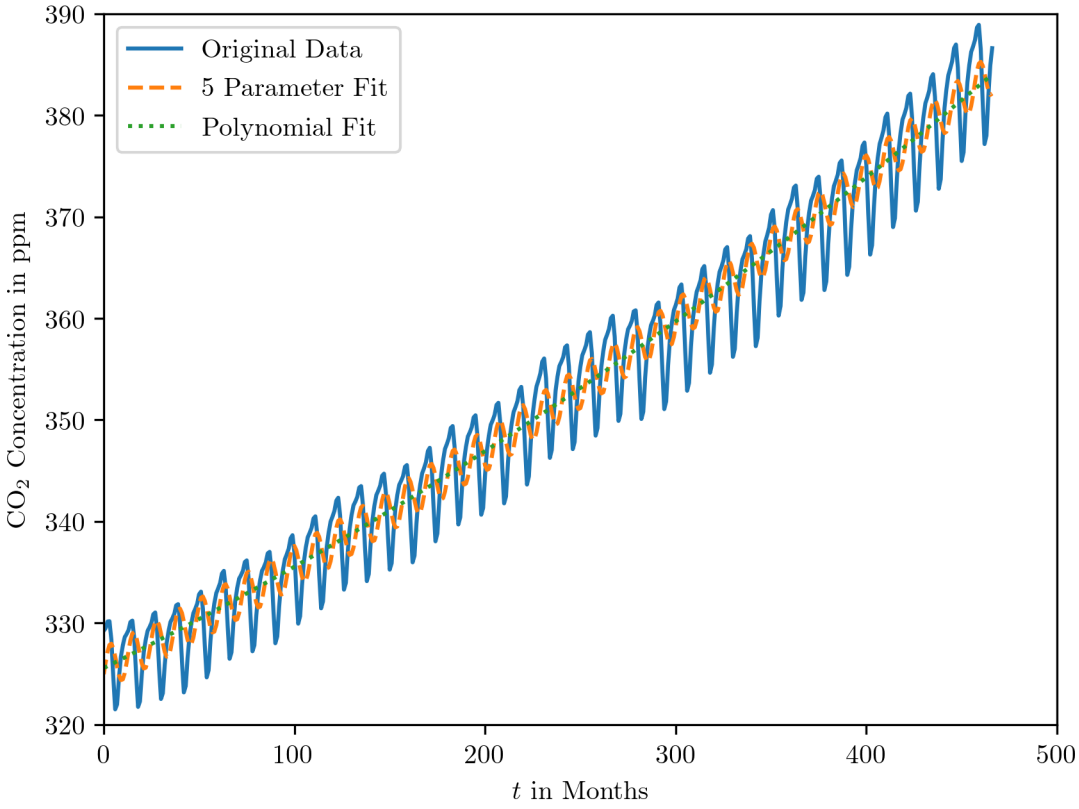

Figure 3: Plot showing the 5 parameter fit against the original data and the polynomial fit.

A plot of the absolute value of the residuals was also generated for both fits as shown below:

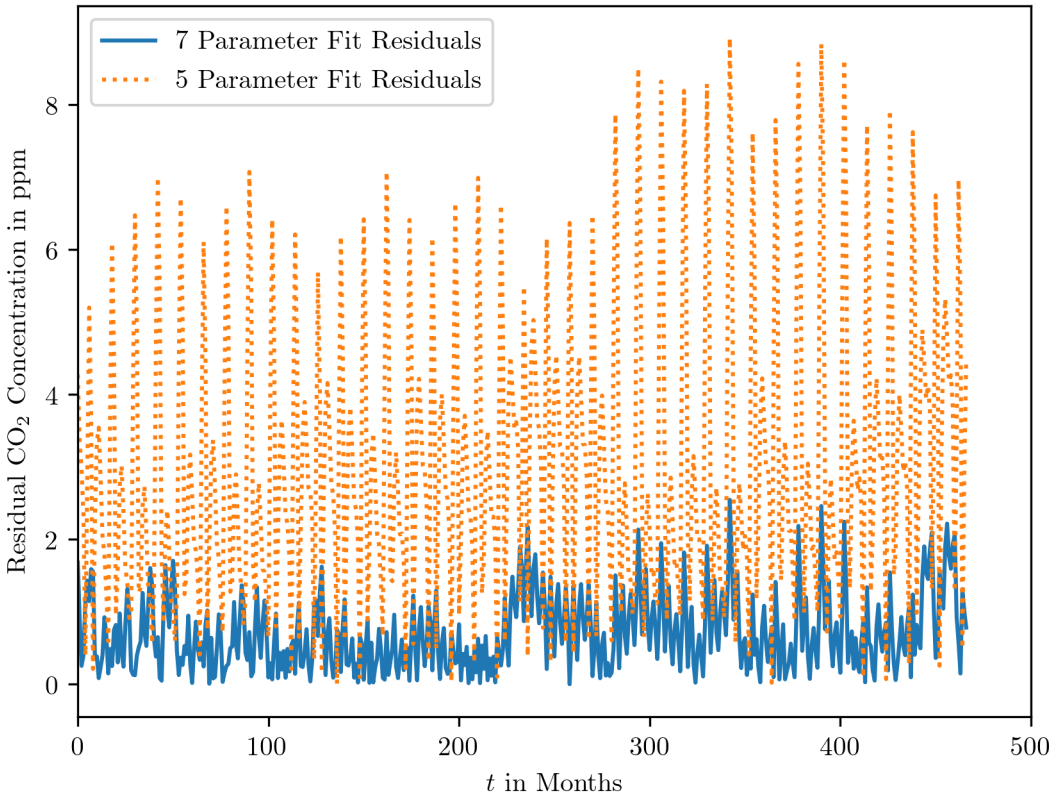

Figure 4: Plot of the absolute residuals for the 5 parameter and 7 parameter fits.

Based on the prior plots and the χ^2, we find tha tthe 7 parameter fit is superior to the 5 parameter fit.

Source Code

```
/* AEP 4380 Homework #8

Least squares curve fitting using Gauss-Jordan is tested
Data source: http://cdiac.ornl.gov/ftp/trends/co2/ljo.dat

Run on a core i7 using clang 1000.11.45.2 on macOS Mojave

Kevin Juan 7 November 2018
*/
```

```
#include <cstdlib>
#include <cmath>
#include <iostream> //  stream IO
#include <fstream>  //  stream file IO
#include <iomanip>  //  to format the output
#include <string>   // STD strings
#include <vector>   // STD vector class
#define ARRAYT_BOUNDS_CHECK
#include "arrayt.hpp" // to use arrays
#define SWAP(a, b)      \
    {                   \
        double temp(a); \
        (a) = (b);      \
        (b) = temp;     \
    }

using namespace std;

// seven parameter model
double model7(double t, int i)
{
    const static double pi = 4.0 * atan(1.0);
    if (i == 0)
    {
        return 1.0;
    }
    else if (i == 1)
    {
        return t;
    }
    else if (i == 2)
    {
        return t * t;
    }
    else if (i == 3)
    {
        return sin(2.0 * pi * t / 12.0);
    }
    else if (i == 4)
    {
        return sin(2.0 * pi * t / 6.0);
    }
    else if (i == 5)
    {
        return cos(2.0 * pi * t / 12.0);
```

```
    }
    else
    {
        return cos(2.0 * pi * t / 6.0);
    }
}

// five parameter model
double model5(double t, int i)
{
    const static double pi = 4.0 * atan(1.0);
    if (i == 0)
    {
        return 1.0;
    }
    else if (i == 1)
    {
        return t;
    }
    else if (i == 2)
    {
        return t * t;
    }
    else if (i == 3)
    {
        return sin(2.0 * pi * t / 12.0);
    }
    else
    {
        return cos(2.0 * pi * t / 12.0);
    }
}

// polynomial model
double modelPoly(double t, int i)
{
    if (i == 0)
    {
        return 1.0;
    }
    else if (i == 1)
    {
        return t;
    }
    else
```

```
    {
        return t * t;
    }
}

// nr3 gaussj
template <class T>
void gaussj(arrayt<T> &a, arrayt<T> &b)
{
    int i, icol, irow, j, k, l, ll, n = a.n1(), m = b.n2();
    double big, dum, pivinv;
    arrayt<T> indxc(n), indxr(n), ipiv(n);
    for (j = 0; j < n; j++)
        ipiv(j) = 0;
    for (i = 0; i < n; i++)
    {
        big = 0.0;
        for (j = 0; j < n; j++)
            if (ipiv(j) != 1)
                for (k = 0; k < n; k++)
                {
                    if (ipiv(k) == 0)
                    {
                        if (abs(a(j, k)) >= big)
                        {
                            big = abs(a(j, k));
                            irow = j;
                            icol = k;
                        }
                    }
                }
        ++(ipiv(icol));
        if (irow != icol)
        {
            for (l = 0; l < n; l++)
                SWAP(a(irow, l), a(icol, l));
            for (l = 0; l < m; l++)
                SWAP(b(irow, l), b(icol, l));
        }
        indxr(i) = irow;
        indxc(i) = icol;
        if (a(icol, icol) == 0.0)
            throw("gaussj: Singular Matrix");
        pivinv = 1.0 / a(icol, icol);
        a(icol, icol) = 1.0;
```

```
        for (l = 0; l < n; l++)
            a(icol, l) *= pivinv;
        for (l = 0; l < m; l++)
            b(icol, l) *= pivinv;
        for (ll = 0; ll < n; ll++)
            if (ll != icol)
            {
                dum = a(ll, icol);
                a(ll, icol) = 0.0;
                for (l = 0; l < n; l++)
                    a(ll, l) -= a(icol, l) * dum;
                for (l = 0; l < m; l++)
                    b(ll, l) -= b(icol, l) * dum;
            }
    }
    for (l = n - 1; l >= 0; l--)
    {
        if (indxr(l) != indxc(l))
            for (k = 0; k < n; k++)
                SWAP(a(k, indxr(l)), a(k, indxc(l)));
    }
}

// nr3 gauss j
template <class T>
void gaussj(arrayt<T> &a)
{
    arrayt<T> b(a.n1(), 0);
    gaussj(a, b);
}

int main()
{
    int i, j, npts, year, t, nval, l, k, nEqs7 = 7, nEqs5 = 5, nEqs3 = 3;
    double co2, ymin, ymax, sumF, sumB, sumChi, fit5ChiSq, fit7ChiSq;
    double fit5Val, fit7Val;
    // use dynamically sized container classes
    string cline;
    vector<double> x, y;

    ifstream fp; // input file stream
    fp.open("ljo.dat");
    if (fp.fail())
    {
        cout << "Can't open file." << endl;
```

```
    exit(0);
}
ofstream fpOrig;
fpOrig.open("original_data.dat");
if (fpOrig.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}

//--------- read data from file in complicted format -------
// skip first 16 lines
for (i = 0; i < 16; i++)
    getline(fp, cline); // read a whole line
t = 0;                  // time in months
npts = 0;               // number of data points
ymin = 1000.0;
ymax = -ymin;
for (i = 0; i < 70; i++)
{
    fp >> year;
    if (0 == i)
        nval = 11;
    else
        nval = 12; // line is short(?)
    for (j = 0; j < nval; j++)
    {
        fp >> co2;
        if (co2 > 0.0)
        {
            x.push_back(t);   // use auto sizing because we don't
            y.push_back(co2); // know how many elements there will be
            if (y[npts] > ymax)
                ymax = y[npts]; // x and y index like an array
            if (y[npts] < ymin)
                ymin = y[npts]; // could use co2 here also
            npts++;
        }
        fpOrig << setw(5) << t << setw(10) << co2 << endl;
        t += 1;
    }
    if (year >= 2007)
        break;          // end of file
    getline(fp, cline); // read rest of line
}
```

```
// initialize arrays
arrayt<double> sigmaSq(npts);
arrayt<double> F7(nEqs7, nEqs7);
arrayt<double> b7(nEqs7, nEqs7);
arrayt<double> F5(nEqs5, nEqs5);
arrayt<double> b5(nEqs5, nEqs5);
arrayt<double> F3(nEqs3, nEqs3);
arrayt<double> b3(nEqs3, nEqs3);

// calculate error values
for (i = 0; i < npts; i++)
{
    sigmaSq(i) = 0.002 * 0.002 * y[i] * y[i];
}

// initialize F_lk for 7 parameter
for (l = 0; l < nEqs7; l++)
{
    for (k = 0; k < nEqs7; k++)
    {
        sumF = 0.0;
        sumB = 0.0;
        for (i = 0; i < npts; i++)
        {
            sumF += model7(x[i], l) * model7(x[i], k) / sigmaSq(i);
            sumB += y[i] * model7(x[i], l) / sigmaSq(i);
        }
        F7(l, k) = sumF;
        if (k == 0)
        {
            b7(l, k) = sumB;
        }
        else
        {
            b7(l, k) = 0.0;
        }
    }
}

// initialize F_lk for 5 parameter
for (l = 0; l < nEqs5; l++)
{
    for (k = 0; k < nEqs5; k++)
    {
```

```
        sumF = 0.0;
        sumB = 0.0;
        for (i = 0; i < npts; i++)
        {
            sumF += model5(x[i], l) * model5(x[i], k) / sigmaSq(i);
            sumB += y[i] * model5(x[i], l) / sigmaSq(i);
        }
        F5(l, k) = sumF;
        if (k == 0)
        {
            b5(l, k) = sumB;
        }
        else
        {
            b5(l, k) = 0.0;
        }
    }
}

// initialize F_lk for polynomial fit
for (l = 0; l < nEqs3; l++)
{
    for (k = 0; k < nEqs3; k++)
    {
        sumF = 0.0;
        sumB = 0.0;
        for (i = 0; i < npts; i++)
        {
            sumF += modelPoly(x[i], l) * modelPoly(x[i], k) / sigmaSq(i);
            sumB += y[i] * modelPoly(x[i], l) / sigmaSq(i);
        }
        F3(l, k) = sumF;
        if (k == 0)
        {
            b3(l, k) = sumB;
        }
        else
        {
            b3(l, k) = 0.0;
        }
    }
}

// Gauss-Jordan elimination
gaussj(F7, b7);
```

```
gaussj(F5, b5);
gaussj(F3, b3);

// output 7 parameter error
for (i = 0; i < nEqs7; i++)
{
    for (j = 0; j < nEqs7; j++)
    {
        if (i == j)
        {
            cout << "Error in parameter a_" << i + 1 << " for the 7
            parameter fit is " << F7(i, j) << endl;
        }
    }
}

// output 5 parameter error
for (i = 0; i < nEqs5; i++)
{
    for (j = 0; j < nEqs5; j++)
    {
        if (i == j)
        {
            cout << "Error in parameter a_" << i + 1 << " for the 5
            parameter fit is " << F5(i, j) << endl;
        }
    }
}

// output 5 parameter error
for (i = 0; i < nEqs3; i++)
{
    for (j = 0; j < nEqs3; j++)
    {
        if (i == j)
        {
            cout << "Error in parameter a_" << i + 1 << " for the
            polynomial fit is " << F3(i, j) << endl;
        }
    }
}

// output 7 parameter fit
ofstream fpFit7;
fpFit7.open("fit7_data.dat");
```

```
if (fpFit7.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}
for (i = 0; i < nEqs7; i++)
{
    fpFit7 << b7(i, 0) << endl;
}

// output 5 parameter fit
ofstream fpFit5;
fpFit5.open("fit5_data.dat");
if (fpFit5.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}
for (i = 0; i < nEqs5; i++)
{
    fpFit5 << b5(i, 0) << endl;
}

// output polynomial fit
ofstream fpFitPoly;
fpFitPoly.open("fitPoly_data.dat");
if (fpFitPoly.fail())
{
    cout << "cannot open file" << endl;
    return (EXIT_SUCCESS);
}
for (i = 0; i < nEqs3; i++)
{
    fpFitPoly << b3(i, 0) << endl;
}

// chi-square calculation for 7 parameter
sumChi = 0.0;
for (i = 0; i < npts; i++)
{
    fit7Val = b7(0, 0) * model7(i, 0) + b7(1, 0) * model7(i, 1) + b7(2, 0)
    * model7(i, 2) + b7(3, 0) * model7(i, 3) + b7(4, 0) * model7(i, 4) +
    b7(5, 0) * model7(i, 5) + b7(6, 0) * model7(i, 6);
    sumChi += (y[i] - fit7Val) * (y[i] - fit7Val) / sigmaSq(i);
}
```

```
    sumChi /= (npts - nEqs7);
    cout << "Chi-Squared for 7 parameter fit: " << sumChi << endl;

    // chi-square calculation for 7 parameter
    sumChi = 0.0;
    for (i = 0; i < npts; i++)
    {
        fit5Val = b5(0, 0) * model5(i, 0) + b5(1, 0) * model5(i, 1) + b5(2, 0)
        * model5(i, 2) + b5(3, 0) * model5(i, 3) + b5(4, 0) * model5(i, 4);
        sumChi += (y[i] - fit5Val) * (y[i] - fit5Val) / sigmaSq(i);
    }
    sumChi /= (npts - nEqs5);
    cout << "Chi-Squared for 5 parameter fit: " << sumChi << endl;

    return (EXIT_SUCCESS);
}
```

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, editors. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK ; New York, 3rd ed edition, 2007. OCLC: ocn123285342.