AEP 4380 HW#7: Least Squares Curve Fitting

Gianfranco Grillo

November 9, 2016

1 Background

1.1 Curve Fitting Via Linear Least Squares

Given a data set obtained from experimental observations, we are often interested in learning the precise relationship between each of the data points. Assuming the relationship is a linear combination of a set of analytic functions, it is possible to find out the linear combination that best fits the data by performing a linear least squares fit. In essence, given a set of analytic functions relating the data to an independent variable, this method finds the weights that need to be given to each of these functions in order to minimize as much as possible the discrepancy between the observed data values and the theoretical values produced by the linear combination of the functions. The discrepancy between each data point and each point in the model is known as the residual. This residual is also dependent on the error inherent to each of the measurements: the greater the error, the less weight is given to the residual, and viceversa. In more precise terms, given raw data points (t_i, y_i) with an error for each y_i given by σ_i $(i = 1, 2, ..., N)$ (and assuming no error in the t_i variable), the "best fit" of a linear combination of fitting functions is defined as that fit which minimizes the total reduced chi-squared given by

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

with $f(t_i)$ corresponding to the linear combination of fitting functions,

$$f(t, \vec{a}) = \sum_{k=1}^{m} a_k f_k(t) \tag{2}$$

and N being the number of data points, and m the number of fitting functions. The least squares method of curve fitting thus consists in finding the appropriate parameters a_k in order to obtain the minimum value of χ_r^2 possible. In practice, this can be done by solving the following matrix equation for the a_k coefficients:

$$F\vec{a} = \vec{b} \tag{3}$$

where the vector $\vec{a}$, the F matrix elements F_{lk}, and the $\vec{b}$ vector elements b_l are given by

$$\vec{a} = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{bmatrix} \tag{4}$$

$$F_{lk} = \sum_{i=1}^{N} \frac{f_l(t_i) f_k(t_i)}{\sigma_i^2} \tag{5}$$

$$b_l = \sum_{i=1}^{N} \frac{y_i f_l(t_i)}{\sigma_i^2} \tag{6}$$

There is more than one way of numerically solving a matrix equation like (3). In this assignment, this is done via Gauss-Jordan elimination (with pivoting), which is described in section 2.1 of [1].

1.2 Fitting Atmospheric CO_2 Data

The webpage http://cdiac.ornl.gov/ftp/trends/co2/maunaloa.co2 contains monthly values of atmospheric CO_2 starting in 1958 and ending in 2008, as measured in Mauna Loa, Hawaii, by the Carbon Dioxide Research Group of the University of California. A plot for this data is shown in Figure 1. This assignment will attempt to fit this data via the least squares method, using a total of 7 fitting functions, given by

$$\begin{aligned} f_0 &= \sin\left(\frac{\pi t}{6}\right), \quad f_1 = \cos\left(\tfrac{\pi t}{6}\right), \quad f_2 = \sin\left(\frac{\pi t}{3}\right) \\ f_3 &= \cos\left(\frac{\pi t}{3}\right), \qquad f_4 = t^2, \qquad f_5 = t \\ f_6 &= 1 \end{aligned} \tag{7}$$

The independent variable t is taken as the number of months since the start of data collection in January of 1958. The first four fitting functions are used to fit for the seasonal variations, whereas the last three are the components of a quadratic function that is used to fit for the long term trend. An error of $\sigma_i = 0.002$ is assumed for all of the CO_2 values, whereas the error in the independent variable t is taken to be non-existent.

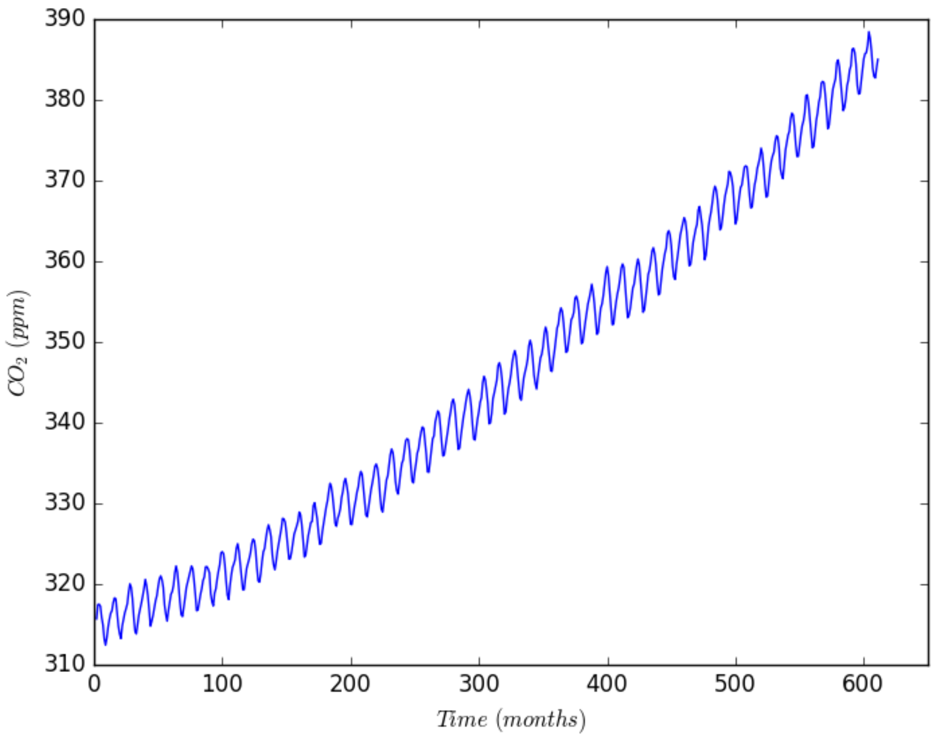

Figure 1: Atmospheric CO_2 vs time as measured in Hawaii

2 Results

2.1 Task 1

Task 1 consisted in performing the fit using the seven fitting functions and plotting it on top of the original data (Figure 2). The reduced chi-square value of the fit was (to 3 s. f.)$\chi_r^2 = 1.09$. The red line in Figure 2 shows the overall underlying trend and was produced by fitting the data only using the functions corresponding to the quadratic coefficients (f_4, f_5, and f_6). Figure 3 contains a scatter plot of the residuals for each point. Table 1 gives the value of the best fit parameters for the full least squares fit (the green curve in Figure 2), as well as their respective relative errors. All quantities are rounded to 3 significant figures.

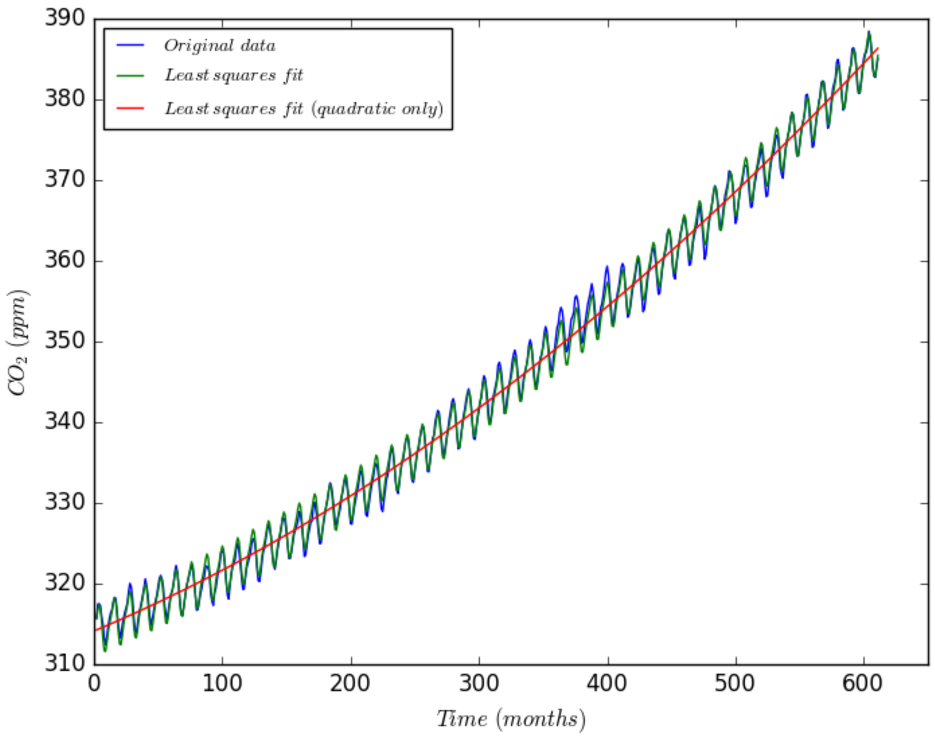

Figure 2: Atmospheric CO_2 vs time with fits

Fit parameter a_k	Value	Relative error
a_0	2.80	1.55×10^{-3}
a_1	-0.374	1.56×10^{-3}
a_2	-0.678	1.55×10^{-3}
a_3	0.384	1.56×10^{-3}
a_4	8.4×10^{-5}	1.02×10^{-12}
a_5	6.69×10^{-2}	3.91×10^{-7}
a_6	314	6.40×10^{-3}

Table 1: Best fit parameter values and error

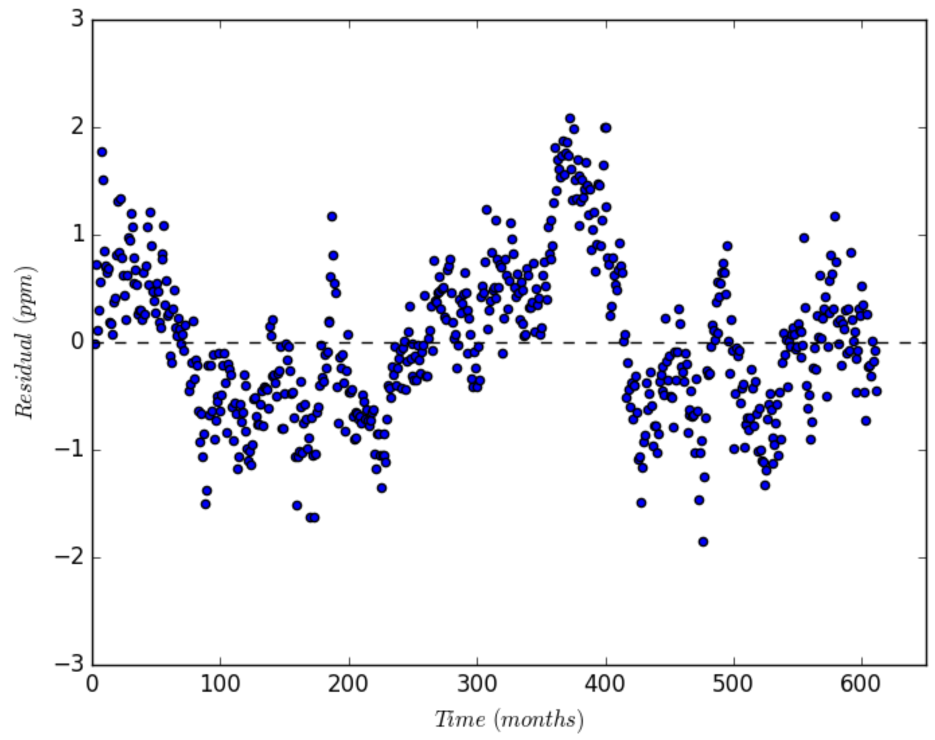

Figure 3: Residual vs time derived from original data and the least squares fit (7 parameters)

2.2 Task 2

The second task consisted in making the fit using five of the seven parameters by dropping f_2 and f_3, the parameters that are intended to account for the seasonal variations. Doing so gives a value of the reduced chi-square of $\chi_i^2 = 1.74$.

2.3 Task 3

Figure 3 shows that the residuals appear not to be random, and follow roughly a sine wave pattern with a period of about 300 months. So I added two more parameters to the fit, $f_7 = \cos(\frac{\pi t}{150})$ and $f_8 = \sin(\frac{\pi t}{150})$ in order to see whether this would improve the fit. It did: $\chi_r^2 = 0.649$ for this fit. Figure 4 shows the resulting residuals.

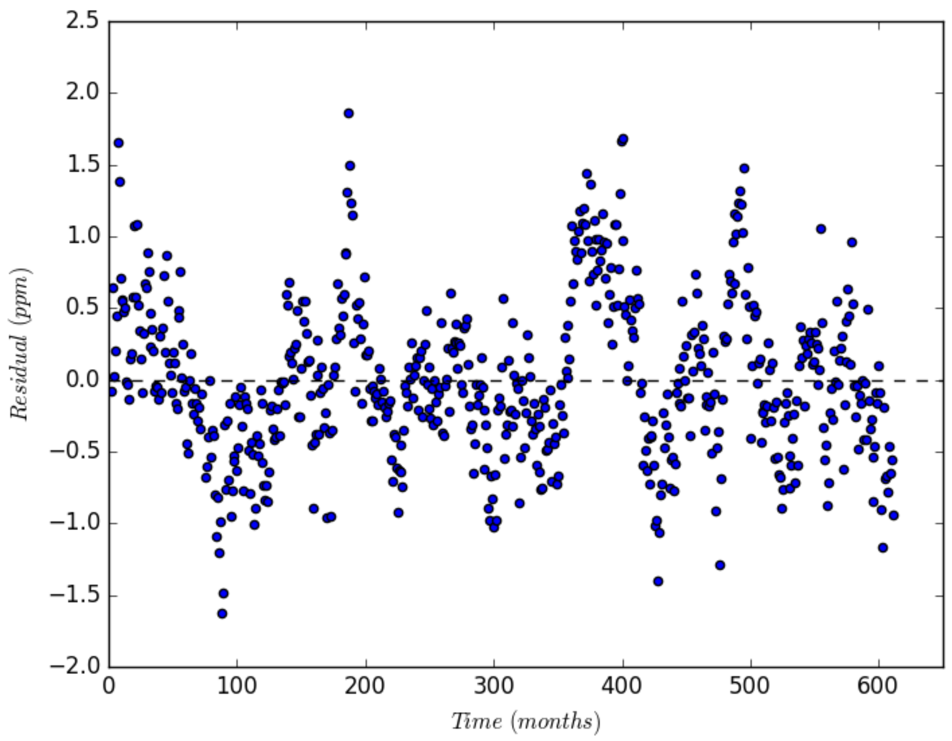

Figure 4: Residual vs time derived from original data and the least squares fit (9 parameters)

3 Analysis

The results show that the data can be successfully modeled with the fitting functions used, as the value for the reduced chi-squared in all three attempted fits falls within the range $0.5 < \chi_r^2 < 2.0$, which is the range expected for a typical good fit. The results also show that the seven parameter model with two 6 month period sine waves provides a better fit than the five parameter model without them, and the nine parameter model with the two 300 month period sine waves provides an even better model still. The first result is to be expected, but the second is a little surprising. What kind of process could be at work that has a 300 month period? It also looks like the residuals shown in Figure 4 could be fit even better of more oscillatory functions. It is possible that there might be some mechanisms at work here that are not obvious based on a cursory glance of the original data.

4 Source code

4.1 hw8a.cpp

```
/* AEP 4380 HW#8a
   Least Squares Curve Fitting
   Run on core i7 with g++ 5.4.0 (Ubuntu)
   Gianfranco Grillo 11/07/2016
*/

#include <cstdlib>
```

```
#include <cmath>

#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <vector>
#include "arrayt.hpp"
#include "nr3.h"
#define ARRAYT_BOUNDS_CHECK

using namespace std;

int main() {
    int i, npts = 607, npar = 7, j, k, t;
    double pi = 4.0 * atan(1.0), Flk, bl, chisqr;
    arrayt<int> tlist(npts);
    arrayt<double> co2list(npts), blist(npar), fit(npts), sqrerrlist(npts), reslist
        (npts);
    arrayt<double> Fmatrix(npar, npar), funcm(npts,npar);
    void readfile(arrayt<int>&, arrayt<double>&); // create t, co2, arrays from
        file
    void gaussj(arrayt<double>&, arrayt<double>&);
    ofstream fp, sp;
    fp.open("output1.dat");
    sp.open("output2.dat");
    readfile(tlist, co2list);
    for (i = 0; i < npts; i++) { // initialize function arrays
        t = tlist(i);
        funcm(i,0) = sin(pi*t/6.0);
        funcm(i,1) = cos(pi*t/6.0);
        funcm(i,2) = sin(pi*t/3.0);
        funcm(i,3) = cos(pi*t/3.0);
        funcm(i,4) = t*t;
        funcm(i,5) = t; // don't really need this but makes life easier
        funcm(i,6) = 1.0;
        sqrerrlist(i) = (0.002*co2list(i))*(0.002*co2list(i)); // create error
            array
    }
    for (i = 0; i < npar; i++) for (j = 0; j < i+1; j++) { // only need to
        calculate half of matrix
        Flk = 0;
        for (k = 0; k < npts; k++) {
            Flk = Flk + funcm(k,i)*funcm(k,j)/sqrerrlist(k);
        }
        Fmatrix(i,j) = Flk;
        if (i != j) {
            Fmatrix(j,i) = Flk;
        }
    }
    for (i = 0; i < npar; i++) { // generate b vector
        bl = 0;
        for (k = 0; k < npts; k++) {
            bl = bl + co2list(k)*funcm(k,i)/sqrerrlist(k);
        }
        blist(i) = bl;
    }
    gaussj(Fmatrix, blist);
    chisqr = 0.0;
    for (i = 0; i < npts; i++) {
        fit(i) = 0;
        for (j = 0; j < npar; j++) {
            fit(i) = fit(i) + blist(j)*funcm(i, j);
            }
        reslist(i) = co2list(i)-fit(i);
        chisqr = chisqr + reslist(i)*reslist(i)/sqrerrlist(i);
        fp << tlist(i) << setw(15) << co2list(i) << setw(15) << fit(i) << setw(15)
           << reslist(i) << endl;
    }
    chisqr = chisqr/(npts-npar);
    cout << chisqr << endl;
    for (i = 0; i < npar; i++) {
        sp << Fmatrix(i,i) << setw(15) << blist(i) << endl;
    }
    fp.close();
    return (EXIT_SUCCESS);
}

void readfile(arrayt<int>& tlist, arrayt<double>& co2list) {
    int i, j, npts, year, t, nval;
    double co2, ymin, ymax;
    string cline;
    vector<double> x, y;
    ifstream fp;
    fp.open("maunaloa.co2.txt");

    for (i=0; i<15; i++) getline(fp,cline);
    t = 0;
    npts = 0;
    ymin = 1000.0;
    ymax = -ymin;
    for (i=0; i<70; i++) {
        fp >> year;
        for (j=0; j<12; j++) {
            fp >> co2;
            if (co2 > 0.0) {
                x.push_back(t);
                y.push_back(co2);
                if (y[npts] > ymax) ymax = y[npts];
                if (y[npts] < ymin) ymin = y[npts];
                tlist(npts) = t;
                co2list(npts) = co2;
                npts++;
            }
            t += 1;
        }
        if (year >=2008) break;
        getline(fp, cline);
    }
    return;
}

void gaussj(arrayt<double> &a, arrayt<double> &b) {
    int i, icol, irow, j, k, l, ll, n=a.n1();
    double big, dum, pivinv;
    arrayt<int> 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));
            SWAP(b(irow), b(icol));
        }
        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;
        b(icol) *= 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;
                b(ll) -= b(icol)*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)));
    }
    return;
```

```
}
```

4.2 main() Variations

4.2.1 Quadratic Fit Only

```
int main() {
    int i, npts = 607, npar = 3, j, k, t;
    double pi = 4.0 * atan(1.0), Flk, bl, chisqr;
    arrayt<int> tlist(npts);
    arrayt<double> co2list(npts), blist(npar), fit(npts), sqrerrlist(npts), reslist
       (npts);
    arrayt<double> Fmatrix(npar, npar), funcm(npts,npar);
    void readfile(arrayt<int>&, arrayt<double>&); // create t, co2, arrays from
       file
    void gaussj(arrayt<double>&, arrayt<double>&);
    ofstream fp, sp;
    fp.open("output3.dat");
    sp.open("output4.dat");
    readfile(tlist, co2list);
    for (i = 0; i < npts; i++) { // initialize function arrays
        t = tlist(i);
        funcm(i,0) = t*t;
        funcm(i,1) = t; // don't really need this but makes life easier
        funcm(i,2) = 1.0;
        sqrerrlist(i) = (0.002*co2list(i))*(0.002*co2list(i)); // create error
            array
    }
    for (i = 0; i < npar; i++) for (j = 0; j < i+1; j++) { // only need to
       calculate half of matrix
        Flk = 0;
        for (k = 0; k < npts; k++) {
            Flk = Flk + funcm(k,i)*funcm(k,j)/sqrerrlist(k);
        }
        Fmatrix(i,j) = Flk;
        if (i != j) {
            Fmatrix(j,i) = Flk;
        }
    }
    for (i = 0; i < npar; i++) { // generate b vector
        bl = 0;
        for (k = 0; k < npts; k++) {
            bl = bl + co2list(k)*funcm(k,i)/sqrerrlist(k);
        }
        blist(i) = bl;
    }
    gaussj(Fmatrix, blist);
    chisqr = 0.0;
    for (i = 0; i < npts; i++) {
        fit(i) = 0;
        for (j = 0; j < npar; j++) {
            fit(i) = fit(i) + blist(j)*funcm(i, j);
            }
        reslist(i) = co2list(i)-fit(i);
        chisqr = chisqr + reslist(i)*reslist(i)/sqrerrlist(i);
        fp << tlist(i) << setw(15) << co2list(i) << setw(15) << fit(i) << setw(15)
           << reslist(i) << endl;
    }
    chisqr = chisqr/(npts-npar);
    cout << chisqr << endl;
    for (i = 0; i < npar; i++) {
        sp << Fmatrix(i,i) << setw(15) << blist(i) << endl;
    }
    fp.close();
    return (EXIT_SUCCESS);
}
```

4.2.2 5 Parameter Fit

```
int main() {
    int i, npts = 607, npar = 5, j, k, t;
    double pi = 4.0 * atan(1.0), Flk, bl, chisqr;
    arrayt<int> tlist(npts);
    arrayt<double> co2list(npts), blist(npar), fit(npts), sqrerrlist(npts), reslist
       (npts);
    arrayt<double> Fmatrix(npar, npar), funcm(npts,npar);
    void readfile(arrayt<int>&, arrayt<double>&); // create t, co2, arrays from
       file
    void gaussj(arrayt<double>&, arrayt<double>&);
    ofstream fp, sp;
    fp.open("output5.dat");
    sp.open("output6.dat");
    readfile(tlist, co2list);
    for (i = 0; i < npts; i++) { // initialize function arrays
        t = tlist(i);
        funcm(i,0) = sin(pi*t/6.0);
        funcm(i,1) = cos(pi*t/6.0);
        funcm(i,2) = t*t;
        funcm(i,3) = t; // don't really need this but makes life easier
        funcm(i,4) = 1.0;
        sqrerrlist(i) = (0.002*co2list(i))*(0.002*co2list(i)); // create error
           array
    }
    for (i = 0; i < npar; i++) for (j = 0; j < i+1; j++) { // only need to
       calculate half of matrix
        Flk = 0;
        for (k = 0; k < npts; k++) {
            Flk = Flk + funcm(k,i)*funcm(k,j)/sqrerrlist(k);
        }
        Fmatrix(i,j) = Flk;
        if (i != j) {
            Fmatrix(j,i) = Flk;
        }
    }
    for (i = 0; i < npar; i++) { // generate b vector
```

```
        bl = 0;
        for (k = 0; k < npts; k++) {
            bl = bl + co2list(k)*funcm(k,i)/sqrerrlist(k);
        }
        blist(i) = bl;
    }
    gaussj(Fmatrix, blist);
    chisqr = 0.0;
    for (i = 0; i < npts; i++) {
        fit(i) = 0;
        for (j = 0; j < npar; j++) {
            fit(i) = fit(i) + blist(j)*funcm(i, j);
            }
        reslist(i) = co2list(i)-fit(i);
        chisqr = chisqr + reslist(i)*reslist(i)/sqrerrlist(i);
        fp << tlist(i) << setw(15) << co2list(i) << setw(15) << fit(i) << setw(15)
           << reslist(i) << endl;
    }
    chisqr = chisqr/(npts-npar);
    cout << chisqr << endl;
    for (i = 0; i < npar; i++) {
        sp << Fmatrix(i,i) << setw(15) << blist(i) << endl;
    }
    fp.close();
    return (EXIT_SUCCESS);
}
```

4.2.3 9 Parameter Fit

```
int main() {
    int i, npts = 607, npar = 9, j, k, t;
    double pi = 4.0 * atan(1.0), Flk, bl, chisqr;
    arrayt<int> tlist(npts);
    arrayt<double> co2list(npts), blist(npar), fit(npts), sqrerrlist(npts), reslist
       (npts);
    arrayt<double> Fmatrix(npar, npar), funcm(npts,npar);
    void readfile(arrayt<int>&, arrayt<double>&); // create t, co2, arrays from
       file
    void gaussj(arrayt<double>&, arrayt<double>&);
    ofstream fp, sp;
    fp.open("output7.dat");
    sp.open("output8.dat");
    readfile(tlist, co2list);
    for (i = 0; i < npts; i++) { // initialize function arrays
        t = tlist(i);
        funcm(i,0) = sin(pi*t/6.0);
        funcm(i,1) = cos(pi*t/6.0);
        funcm(i,2) = sin(pi*t/3.0);
        funcm(i,3) = cos(pi*t/3.0);
        funcm(i,4) = t*t;
        funcm(i,5) = t; // don't really need this but makes life easier
        funcm(i,6) = 1.0;
        funcm(i,7) = cos(pi*t/150);
        funcm(i,8) = sin(pi*t/150);
        sqrerrlist(i) = (0.002*co2list(i))*(0.002*co2list(i)); // create error
            array
    }
    for (i = 0; i < npar; i++) for (j = 0; j < i+1; j++) { // only need to
        calculate half of matrix
        Flk = 0;
        for (k = 0; k < npts; k++) {
            Flk = Flk + funcm(k,i)*funcm(k,j)/sqrerrlist(k);
        }
        Fmatrix(i,j) = Flk;
        if (i != j) {
            Fmatrix(j,i) = Flk;
        }
    }
    for (i = 0; i < npar; i++) { // generate b vector
        bl = 0;
        for (k = 0; k < npts; k++) {
            bl = bl + co2list(k)*funcm(k,i)/sqrerrlist(k);
        }
        blist(i) = bl;
    }
    gaussj(Fmatrix, blist);
    chisqr = 0.0;
    for (i = 0; i < npts; i++) {
        fit(i) = 0;
        for (j = 0; j < npar; j++) {
            fit(i) = fit(i) + blist(j)*funcm(i, j);
            }
        reslist(i) = co2list(i)-fit(i);
        chisqr = chisqr + reslist(i)*reslist(i)/sqrerrlist(i);
        fp << tlist(i) << setw(15) << co2list(i) << setw(15) << fit(i) << setw(15)
           << reslist(i) << endl;
    }
    chisqr = chisqr/(npts-npar);
    cout << chisqr << endl;
    for (i = 0; i < npar; i++) {
        sp << Fmatrix(i,i) << setw(15) << blist(i) << endl;
    }
    fp.close();
    return (EXIT_SUCCESS);
}
```

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling and B. P. Flannery *Numerical Recipes, The Art of Scientific Computing, 3rd edit.*, Camb. Univ. Press 2007.