AEP 4380: Homework 8

Gregory Kaiser

November 15th, 2019

1 Problem Background and Solution Overview

The analysis of data trends sometimes requires more sophisticated analysis than finding a simple linear or exponential best fit line. Using a set of equations meant to model a set of data, parts of the data that have a specific functional form can be extracted. Each of these terms has its own physical significance and consequences on the overall trend.

Breaking a set of data into a sum of linear terms allows the calculation of a function which attempts to approximate the data. This estimate function is given by:

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

Where m is the number of fitting equations being used, a_k represents the coefficient of that particular term, and $f_k(t)$ represents one of the linear terms k evaluated at a particular time.

The reduced form of the chi-squared value takes into account the number of degrees of freedom that could account for error, and therefore divides by the difference between the number of points being approximated and the number of equations being used to approximate the data. It can give some sense of how well the best fit function fits the data, and is given as:

$$\chi_r^2 = \frac{1}{N-m} \sum_{i=1}^{N} \left(\frac{y(t_i) - f(t_i)}{\sigma(t_i)} \right)^2 \tag{2}$$

Where N is the number of data points in the set, $y(t_i)$ is the value of each point i, and $\sigma(t_i)$ is the approximate error at each point. Finding the form of $f(k,t)$ which fits the data set $y(i)$ the best is the same as minimizing this error estimate value.

This assignment finds a best fit function for the concentration of Carbon Dioxide (CO_2) as measured monthly from Baja California Sur, Mexico [3][4]. Using a parsing script provided [3], this data was plotted, and clearly has an overlying upward trend, with seasonal variations every 12 months, corresponding to an annual cycle:

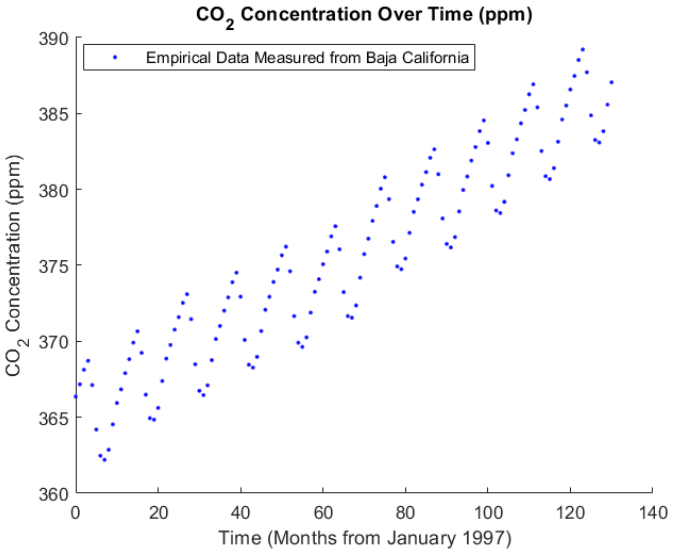

Figure 1: The raw data from Baja California Sur, Mexico. [4]

Assuming that the error at each point is approximately 0.1% and assuming that the seasonal variation can be approximated as a 12-month sinusoidal along with its first (6-month) harmonic, χ_r^2 is minimized to find a best fit line.

2 Solution Description

Minimizing the value of χ_r^2 boils down to solving the matrix equation

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

Where F is the NxN matrix

$$F_{lk} = \sum_{i=1}^{N} \frac{f_l(t_i) f_k(t_i)}{\sigma_i^2} \tag{4}$$

and b is the vector

$$b_l = \sum_{i=1}^{N} \frac{y(t_i) f_l(t_i)}{\sigma_i^2} \tag{5}$$

[3].

Using a given script for parsing the online data set [3], the data $y(t)$ is stored in memory. The error for each point is calculated by multiplying the data point by 0.001 corresponding to 0.1% error.

The function $f(t)$ was written as a helper function which returns a double when fed an index k for which function f_k to use and a time t at which to evaluate f_k. A switch case setup chooses the correct function to perform using the input, and $f(t)$ has seven terms corresponding to to following:

$$
\begin{aligned}
f_0(t) &= 1 \\
f_1(t) &= t \\
f_2(t) &= t^2 \\
f_3(t) &= cos(\frac{2\pi t}{12}) \\
f_4(t) &= sin(\frac{2\pi t}{12}) \\
f_5(t) &= cos(\frac{2\pi t}{6}) \\
f_6(t) &= sin(\frac{2\pi t}{6})
\end{aligned}
$$

F can then be generated by two for loops which step over all indices of its two dimensions. At each element, the sum over all times generates that array element's value following the formula from equation 4. Since F is symmetric, the loops only iterate through the upper diagonal elements. The array $\vec{b}$ is calculated similarly, but only looped over a single dimension, and uses the value of the data at $y(t_i)$.

Gauss-Jordan Elimination is used to solve the matrix equation 3. A function called gaussj() from Numerical Recipes [1] performs this matrix math in an efficient manner. Utilizing pivoting to avoid dividing by a zero element, the result of gaussj(F,$\vec{b}$) is to fill $\vec{b}$ with the matrix solution $\vec{a}$ and F with its inverse, F^{-1}. This function was adapted to use arrayt objects for familiarity and debugging purposes, and was tested with a simple example to verify its efficacy [2].

After calculating the solution $\vec{a}$, the actual function is formed using the formula in equation 1. While generating the fitting function, a separate set of data is also written to represent the best fit using only the first three components, which represent the constant, linear, and quadratic terms of the fit. This function with fewer terms is the trend of the data without seasonal variations. The residual plot is also generated easily from this point in the program.

The error for each term in the fitting function is given by:

$$\sigma_l = \sqrt{F^{-1}_{kk}} \tag{6}$$

Where l corresponds to the coefficient of the l^{th} function component.

3 Results and Interpretation

With a χ^2_r value of 1.17916, the following represents the seven-term best fit line generated using the described method.

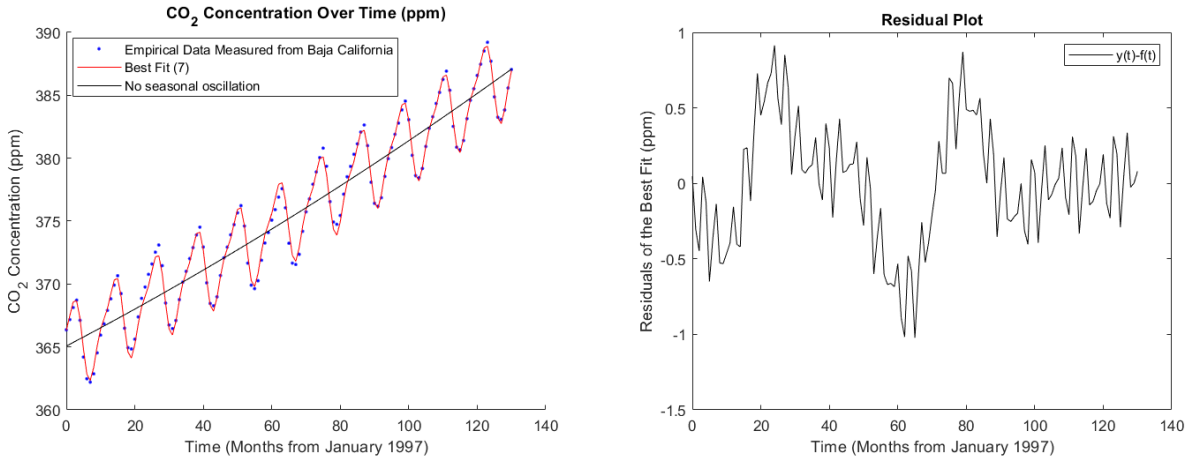

Figure 2: The seven-term best fit curve with non-oscillatory function and residual plot.

Since the value of χ_r^2 was between 0.5 and 2.0, this curve is considered a good fit for the data, and the fitting function does not deviate by more than 0.1% on average.

With a χ_r^2 value of 4.7563, the following represents the five-term best fit line generated using the described method. It neglects the first harmonic of the 12-month seasonal variation.

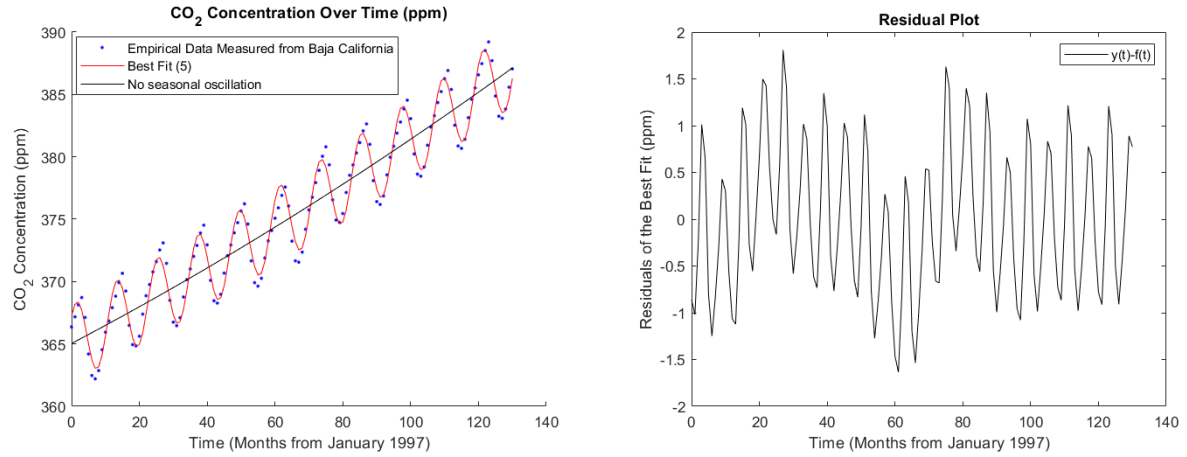

Figure 3: The five-term best fit curve with non-oscillatory function and residual plot.

Since the value of χ_r^2 for the 5 term fit was greater than 2, this curve is considered *not* a good fit for the data.

The seven-term approximation was much better than the five-term fit. This makes sense, since as you add more functional flexibility, the better you can tune the fitting function. Seeing which functions are required to get a good fit based on a glance at the data is a critical piece of using this method. By picking out the terms that one needs, a model of the system can be approximated, and important pieces of the trend can be extracted.

The residual plots should look like noise hovering closely around 0 to indicate that the residual has no functional form of its own. Figure 2 shows a residual plot that might still have some functional dependence: perhaps a slower, 60-month-period sin-wave. More data is needed to confirm that this

new term should be included in the fitting function, but it is possible that there is a larger-period variation in this data.

This table shows each term of the best fit function with its associated error:

Equation Index	Coefficient (7)	Error Parameters (7)	Coefficient (5)	Error Parameters (5)
0	365.07183	0.095258597	365.03678	0.095215733
1	0.14269703	0.0034232854	0.14308237	0.0034222797
2	0.00020490633	2.5660758e-005	0.00020648652	2.5654083e-005
3	2.173156	0.046432572	2.1825455	0.046427666
4	2.2434073	0.046455646	2.2407659	0.04645038
5	-0.93157652	0.046313849	N/A	N/A
6	-0.31806841	0.046487511	N/A	N/A

By extracting the parts of the best fit line that are constant, linear, and quadratic, one notices a small linear term on top of the quite strong 12-month period signal and constant offset. This indicates that the amount of CO_2 in the atmosphere near Baja California Sur, Mexico is increasing over time (and very slightly accelerating). One can clearly see why a more busy piece of data would require this method to extract the important overall trends, especially with data sets that have many dependant variables.

References

[1] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes, The Art of Scientific Computing*. (3rd edit.), Cambridge Univ. Press, 2007, (ISBN 978-0-521-88068- 8, QA297 .N866 2007) Online files accessed through Cornell's institutional subscriber license at http://numerical.recipes/routines/instbyfile.html.

[2] Kirkland, Earl. *Array Class Objects in C/C++ for Vectors and Matrices* AEP 4380 Fall 2019. https://courses.cit.cornell.edu/aep4380/secure/arrayt.hpp

[3] Kirkland, Earl. *Least Squares Curve Fitting.* AEP 4380 Fall 2019 Assignment 8. https://courses.cit.cornell.edu/aep4380/secure/hw08f19.pdf

[4] Keeling, R. F., S. C. Piper, A. F. Bollenbacher, and S. J. Walker. "Atmospheric CO2-curve values (ppmv) derived from flask air samples collected at Baja California Sur, Mexico." Carbon Dioxide Research Group. Scripps Institution of Oceanography (SIO). University of California. https://cdiac.ess-dive.lbl.gov/ftp/trends/co2/baj.dat

Source Code

```
/*  AEP 4380 Assignment #8
    Least Squares Curve Fitting

    Run on Windows core i7 with gcc version 8.2.0 (MinGW.org GCC-8.2.0-3)

    Gregory Kaiser November 15 2019

*/
#include <cstdlib>
#include <cmath>

#include <iostream>
#include <fstream>
#include <iomanip>

#include <string>//std strings
#include <vector>//std vector

//from class website -
//Kirkland, E: https://courses.cit.cornell.edu/aep4380/secure/arrayt.hpp
#include "arrayt.hpp"//for the matrix math stuff

using namespace std; //makes writing code easier

double pi = 4.0*atan(1.0);//pi

void testGauss();
void ghk_gaussj(arrayt<double>&, arrayt<double>&);
double f_co2(int, double);//helper function
//void ghk_gje(arrayt<double>&, arrayt<double>&); //gaussj without pivot

int main(){

    ofstream fp2; //output file for original data
    fp2.precision(9);

    fp2.open("hw8_1.dat");
    if(fp2.fail()){
        cout<<"cannotopenfile"<<endl;
        return(EXIT_SUCCESS);
    }

    ofstream fp3; //output file for best fit line
    fp3.precision(8);

    fp3.open("hw8_2.dat");
    if(fp3.fail()){
        cout<<"cannotopenfile"<<endl;
        return(EXIT_SUCCESS);
    }

    ofstream fp4; //output file for best fit errors
    fp4.precision(8);

    fp4.open("hw8_3.dat");
    if(fp4.fail()){
```

```
    cout<<"cannotopenfile"<<endl;
    return(EXIT_SUCCESS);
}

//---------------READING DATA EXAMPLE------------------------------
//Kirkland, E: https://courses.cit.cornell.edu/aep4380/secure/hw08f19.pdf
int i, j, k, npts, year, t, nval;
double co2, ymin, ymax;
// use dynamically sized container classes
string cline;
vector<double> x, y;
ifstream fp; // input file stream
fp.open("baj.dat.txt");
if( fp.fail() ) { cout << "Can't open file."<< endl; exit(0); }
//--------- read data from file in complicted format -------

// skip first 15 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;
    //cout << "year = " << year << endl;
    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++;
        }
        //cout << "t= " << t << ", co2= " << co2 << ", npts= " << npts << endl;
        fp2<<co2<<setw(15)<<t<<endl;
        t += 1;
    }
    if( year >= 2007 ) break; // end of file
    getline( fp, cline ); // read rest of line
}
//cout << "y.size() = " << y.size() << endl; // example of size(); should be same as npts
//cout << "Total number of points = " << npts << endl;
//cout << " with range " << ymin << " to " << ymax << endl;
j = 0;
for( i=0; i<x.size(); i++){
    if( fabs( i - x[i] ) > 0.1 ) {
        cout << "i, t[i] = " << i << ", " << x[i] << endl;
        j++;
    }
}
//cout << "number of bad points = " << j << endl;
//-------END READING DATA EXAMPLE----------------------------------------

//x and y contain the co2 data as a function of time (x=t, y=co2)

//-----------HW begin-------------------
```

```
int num_eqns = 7; //the number of fitting functions being used, 5 or 7
arrayt<double> F(num_eqns,num_eqns);
arrayt<double> b(num_eqns,1);
arrayt<double> sigma(npts);

//construct sigma from the co2 numbers in y
for(i=0;i<npts;i++){
    sigma(i) = 0.001*y[i];
    //error is 0.1% of the co2 concentration at a given y
}

//construct the matrix F using 2 nested for loops
//Fij=Fji so some steps can be taken towards efficiency
for(i=0;i<num_eqns;i++){
    for(j=i;j<num_eqns;j++){
        //calculate F_ij as a sum over every month with corresponding sigma
        F(i,j)=0;//zero the element before the sum
        for(k=0;k<npts;k++){
            //accumulate over all times for a specific element
            F(i,j)+=f_co2(i,x[k])*f_co2(j,x[k])/(sigma(k)*sigma(k));
        }
        F(j,i)=F(i,j);//symmetry
    }//end j
}//end i

//construct the b matrix using sigma and f_co2
for(i=0;i<num_eqns;i++){
    b(i,0)=0;//clear b before sum
    for(j=0;j<npts;j++){
        //sum a component of b over all points
        b(i,0)+=(y[j])*f_co2(i,x[j])/(sigma(j)*sigma(j));
    }
}

//solve the matrix equation Fa=b using the helper function
//for Gauss-Jordan Elimination
ghk_gaussj(F,b);
//b now contains a and F is now F^-1
double fitpoint;
double noseason;
double chi_sqr_red=0;
double error_now=0;
for(i=0;i<npts;i++){
    fitpoint=0;//the fit point for this month
    noseason=0;//the fit point without the seasonal oscillations
    for(j=0;j<num_eqns;j++){
        fitpoint+=b(j,0)*f_co2(j,x[i]);
        if(j<3){//j=0,1,2 are the const, lin, quad terms
            noseason+=b(j,0)*f_co2(j,x[i]);
        }
    }
    error_now = y[i]-fitpoint;
    chi_sqr_red += (error_now)*(error_now)/(sigma(i)*sigma(i));
    fp3<<fitpoint<<setw(15)<<error_now<<setw(15)<<noseason<<setw(15)<<i<<endl;
}
chi_sqr_red = chi_sqr_red/(npts-num_eqns);
cout<<"chi_sqr_red: "<<chi_sqr_red<<endl;

arrayt<double> errors(num_eqns);
```

```
    //calculate the error terms for the best fit parameters
    for(i=0;i<num_eqns;i++){
        errors(i) = sqrt(F(i,i));
        fp4<<b(i,0)<<setw(15)<<errors(i)<<setw(15)<<i<<endl;
    }

    //fp5.close();
    fp4.close();
    fp3.close();
    fp2.close();
    return(EXIT_SUCCESS);
} //end main

void testGauss(){
    cout<<"test: "<<endl;
    cout<<setw(15);
    int i,j;
    arrayt<double> tester(3,3);
    arrayt<double> sampleb(3,1);

    tester(0,0)=1;tester(0,1)=100;tester(0,2)=0;
    tester(1,0)=1;tester(1,1)=0;tester(1,2)=10;
    tester(2,0)=0;tester(2,1)=15;tester(2,2)=1;

    sampleb(0,0)=1;sampleb(1,0)=1;sampleb(2,0)=1;
    for(j=0;j<3;j++){
        for(i=0;i<3;i++){
            cout<<tester(i,j)<<setw(15);
        }cout<<endl;
    }cout<<endl;

    for(j=0;j<3;j++){
        cout<<sampleb(j,0)<<setw(15)<<endl;
    }cout<<endl;
    ghk_gaussj(tester,sampleb);
    for(j=0;j<3;j++){
        cout<<sampleb(j,0)<<setw(15)<<endl;
    }
}

//The function f which carries all of the problem specific fitting functions
//returns the value of the function at the given indpendent variable
//corresponding to the function given by the function index
double f_co2(int index, double months){
    //for this problem, the functions are:
    //f0 = 1
    //f1 = x
    //f2 = x^2
    //f3 = cos(x*2*pi/12) //corresponds to a 12 month (annual) period wave
    //f4 = sin(x*2*pi/12)
    //f5 = cos(x*2*pi/6) //corresponds to a 6 month (seasonal) period wave
    //f6 = sin(x*2*pi/6)
    switch(index){
        case 0: return 1;
        case 1: return months;
        case 2: return months*months;
        case 3: return cos(months*2.0*pi/12.0);
        case 4: return sin(months*2.0*pi/12.0);
        case 5: return cos(months*2.0*pi/6.0);
```

```
        case 6: return sin(months*2.0*pi/6.0);
        default: cout<<"Undefined Function Index"<<endl;
                 return(EXIT_SUCCESS);

    }
}

//modified gaussj from Num. Rec Press et. al. to use arrayt class for familiarity
//http://numerical.recipes/routines/instbyfile.html
void ghk_gaussj(arrayt<double> &a, arrayt<double> &b){
int i,icol,irow,j,k,l,ll,n=a.n1(),m=b.n2();
    //cout<<n<<setw(10)<<m<<setw(15)<<b.n2()<<endl;
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));
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)));
}
}

// //Gauss-Jordan Elimination for a given equation Ax=b
// void ghk_gje(arrayt<double> &A, arrayt<double> &b){
//     //solves for x if Ax=b
//     int i, j, k, num = b.n1();
```

```
//     for(i=0;i<num;i++){
//         b(i) = b(i)/A(i,i);
//         for(j=num-1;j>=i;j--){
//             A(i,j) = A(i,j)/A(i,i);
//         }
//         for(j=0;j<num;j++){
//             if(j!=i){//valid
//                 b(j) -= -A(j,i)*b(i);
//                 for(k=num-1;k>=i;k--){
//                     A(j,k) -= -A(j,i)*A(i,k);
//                 }
//             }
//         }//endj
//     }//end i
//     //now b is x and A is destroyed
// }
```