AEP4380 HW8
Least Squares Curve Fitting

Dan Girshovich

April 12, 2013

1 Overview

This document details the results of the included C++ program, which uses a reduced Chi-squared metric to fit a linear combination of functions to provided data. Specifically, the atmospheric CO2 data from the last 50 years is fit to a combination of polynomial and sinusoidal terms. Seasonal variations in the data can be removed using this method simply by discounting the sinusoidal terms.

2 Fitting Method

2.1 Overview

A linear fitting method seeks to find a function

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

such that the value

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

is minimized. Here, the m functions f_k are fitting functions, the m values a_k are fitting parameters, and the reduced Chi-squared value χ_r^2 is a function of the fit $f(t, \vec{a})$, the data $\vec{y}$, and the errors in the data $\vec{\sigma_i}$.

2.2 Matrix Formulation

Solving for the optimal fitting parameters $\vec{a}$ in the above equations is equivalent to solving for $\vec{a}$ in

$$\mathbf{F}\vec{a} = \vec{b}$$

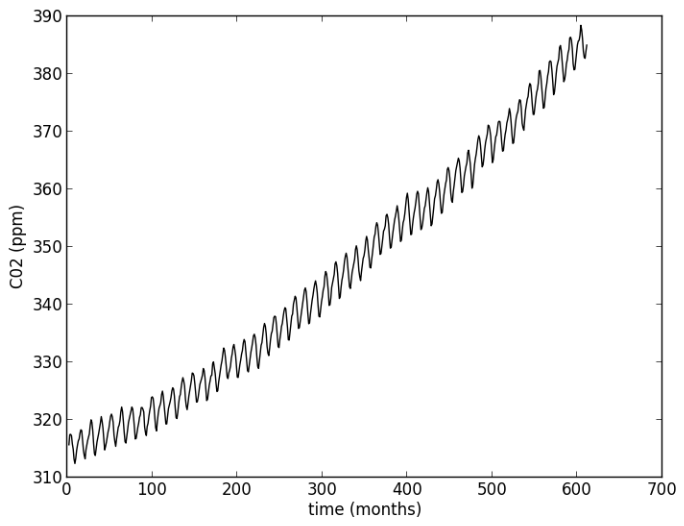

Figure 1: Atmospheric C02 measurements

where

$$F_{lk} = \sum_{i=1}^{N} \frac{f_l(t_i) f_k(t_i)}{\sigma_i^2} \text{ and } b_l = \sum_{i=1}^{N} \frac{y_i f_l(t_i)}{\sigma_i^2}.$$

$\mathbf{F}$ and $\vec{b}$ are calculated in `get_F` and `get_b`, respectively. `gaussj` from Numerical Recipes Chapter 2 [1] is then used to perform Gauss - Jordan elimination (with pivoting) to solve for $\vec{a}$ given $\mathbf{F}$ and $\vec{b}$. `gaussj` also produces $\mathbf{F^{-1}}$, which contains the *parameter* errors in its diagonals.

3 Fitting to Data

3.1 Carbon Dioxide Measurements

The CO2 data on `http://cdiac.ornl.gov/ftp/trends/co2/maunaloa.co2` is shown in Figure 1. Notice that there are short-term seasonal variations as well as a long-term upward trend. This data is read into the program through `load_arrays`.

3.2 Fitting Functions

A set of fitting function are linearly combined (with the fitting parameters) to create the fit. Looking at the data, it is evident that the long-term trend can be approximated with a (roughly linear) polynomial term and the short term trend can be approximated with sinusoidal terms. From this observation, the following fitting functions were chosen (t has units of months):

$$1,\ t,\ t^2,\ sin(kt),\ cos(kt),\ sin(\frac{kt}{2}), \text{ and } cos(\frac{kt}{2})$$

where $k = \frac{2\pi}{12}$, so the oscillating terms represent variations on both the 1 year and 6 month scale.

These functions are created in `get_fit_funcs`. This implementation allows the selection of a subset of fitting functions through the flag `first_harmonic`, which specifies whether the oscillating terms with the 6 month period are included. The function returns a `std:vector` of fitting functions that is used to solve for fitting parameters and analyze the fit by the remainder of the code.

3.3 Goodness of Fit Information

The fitting parameters are found using the matrix formulation described. This is implemented in `write_all_fit_data`, which uses `gaussj` to solve for $\vec{a}$. This function also extracts errors from $\mathbf{F_{-1}}$, calculates the final fit function, uses it to find χ_r^2, and outputs the results.

4 Results

4.1 Fits

The data was fit using the polynomial and sinusoidal basis functions given in the last section. The results are shown in Figure 2. The fit was also done without the seasonal variation terms included in the final fit function, and this is shown in Figure 3. The fit with seasonal variations is shown as an overlay of the original data in Figure 4. Finally, a residual plot between the full fit and the data is shown in Figure 5.

4.2 Goodness of Fit

For the full fit, $\chi_r^2 = 0.000174229$, which indicates a very good fit. The residual plot has low correlation, also indicating a good fit. However the residuals seem to show a long term oscillating term could be added to the fit function to further reduce correlation. The final fit parameters and their errors are shown below:

f	a	$error$
1	314.148	64.0772
t	0.0668684	0.00391019
t^2	8.40915e-05	1.02474e-08
$sin(kt)$	2.77709	15.5171
$cos(kt)$	-0.365962	15.5776
$sin(\frac{kt}{2})$	0.0197292	15.4916
$cos(\frac{kt}{2})$	0.0301944	15.6276

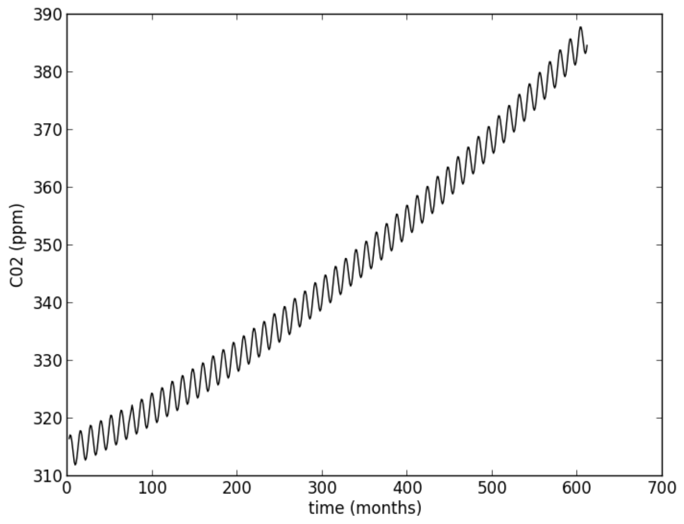

Figure 2: Fit using all fitting functions.

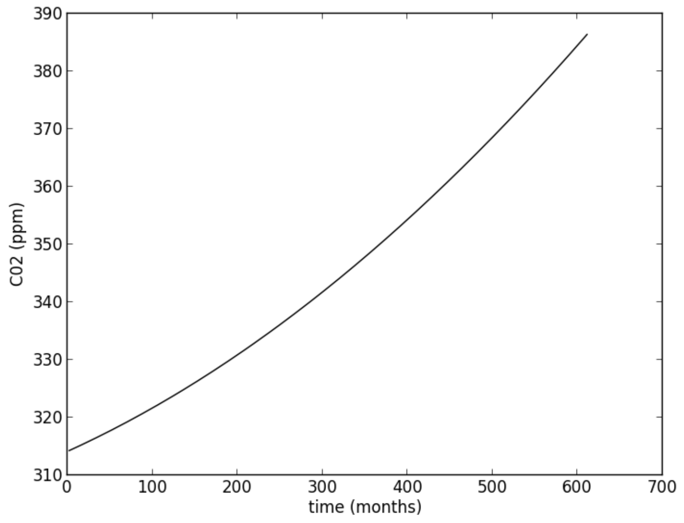

Figure 3: Fit with seasonal variations removed.

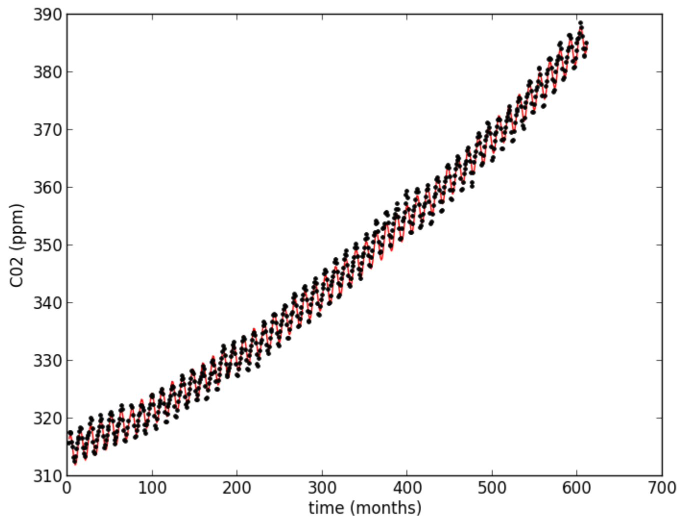

Figure 4: Fit as an overlay of the original data.

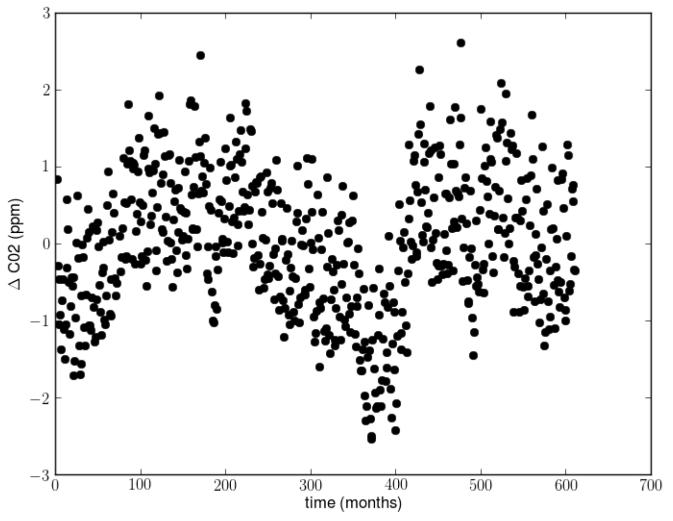

Figure 5: Residuals between full fit and original data.

4.3 Without First Harmonic

A fit without the 6 month periodic terms produces a fit with $\chi_r^2 = 0.000173787$, which indicates a slightly better fit than the full 7 parameter fit. This means that there is likely no oscillation in the data with a 6 month period. The fact that the fit parameters for these first harmonic terms is relatively small in the full fit also supports this claim.

4.4 Source Code

```
/*
AEP 4380 HW #8
Dan Girshovich
4/12/13
Least Squares Curve Fitting
Compile with: g++ -std=c++0x -O2 -o gen_data hw8.cpp
Tested on Mac OSX 10.8.2 with a Intel Core 2 Duo
*/

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include <functional>
#include "nr3.h"

using namespace std;

typedef function<double(double)> math_func;
const int N = 607; // max number of data points
const double sigma_perc =  0.2;
vector<int> ts = vector<int>(N);
vector<double> ys = vector<double>(N);
vector<double> sigmas = vector<double>(N);

// slightly modified given code to read from the data file
void load_arrays() {
    const int NCMAX = 200; // max number of characters per line
    char cline[N];
    double co2;
    FILE *fp;
    fp = fopen("maunaloa.co2", "r");
    if (NULL == fp) {
        printf("Can't open file.");
        exit(0);
    }
    for (int i = 0; i < 15; i++) fgets(cline, NCMAX, fp); // read a whole line
    int t = 0, npts = 0;
    int year;
    do {
        fscanf(fp, "%d", &year);
        for (int j = 0; j < 12; j++) {
            fscanf(fp, "%lf", &co2);
            if (co2 > 0.0) {
                ts[npts] = t;
                ys[npts++] = co2;
            }
            t++;
        }
        if (npts >= N) break;
        fscanf(fp, "%lf", &co2); // skipping average value
    } while (year < 2008);
}

// returns an array of fit functions
vector<math_func> get_fit_funcs(bool first_harmonic) {
    const double pi = 4.0 * atan(1.0);
    double k = 2 * pi / 12;
    vector<math_func> fs {
        [ = ](double t){return 1;},
        [ = ](double t){return t;},
        [ = ](double t){return t * t;},
        [ = ](double t){return sin(k * t);},
        [ = ](double t){return cos(k * t);},
    };
    vector<math_func> half_year_terms {
        [ = ](double t) {return sin((k * t) / 2);},
        [ = ](double t) {return cos((k * t) / 2);},
    };
    if (first_harmonic) {
        fs.insert(fs.end(), half_year_terms.begin(), half_year_terms.end());
    }
    return fs;
}

// returns the F matrix given fit functions
MatDoub_IO get_F(vector<math_func> fs) {
    int m = fs.size();
    MatDoub_IO F(m, m);
    for (int l = 0; l < m; l++) {
        for (int k = 0; k < m; k++) {
            double sum = 0.0;
            for (int i = 0; i < N; i++) {
                double y = ys[i];
                sum += fs[l](ts[i]) * fs[k](ts[i]) / (sigmas[i] * sigmas[i]);
            }
            F[l][k] = sum;
        }
    }
    return F;
}

// returns the b vector given fit functions
MatDoub_IO get_b(vector<math_func> fs) {
    int m = fs.size();
    MatDoub_IO b(m, 1);
    for (int l = 0; l < m; l++) {
        double sum = 0.0;
        for (int i = 0; i < N; i++) {
            double y = ys[i];
            sum += y * fs[l](ts[i]) / (sigmas[i] * sigmas[i]);
        }
        b[l][0] = sum;
    }
    return b;
}

// From Numerical Recipes Ch 2
void gaussj(MatDoub_IO &a, MatDoub_IO &b) {
    // see text
}

// From Numerical Recipes Ch 2
void gaussj(MatDoub_IO &a) {
    MatDoub b(a.nrows(), 0);
    gaussj(a, b);
}

void write1(string name, int suffix, function<double(int)> g, int max) {
    stringstream fname;
    fname << name << suffix << ".dat";
    ofstream of;
    of.open(fname.str().c_str());
    for (int i = 0; i < max; i++) {
        of << g(i) << endl;
    }
    of.close();
}

void write2(string name, int suffix, function<double(int)> g, int max) {
    stringstream fname;
    fname << name << suffix << ".dat";
    ofstream of;
    of.open(fname.str().c_str());
    for (int i = 0; i < max; i++) {
        of << ts[i] << "␣" << g(i) << endl;
    }
    of.close();
}

void write_all_fit_data(vector<math_func> fs, bool seasonal_var) {
    MatDoub_IO F = get_F(fs);
    MatDoub_IO b = get_b(fs);
    gaussj(F, b);
    // now b contains the solution vector (a), and F is (old F)^-1
    MatDoub &F_inv = F;
    MatDoub &a = b;
    int m = seasonal_var ? fs.size() : 3;
    math_func f = [ = , &a](double t) {
        double sum = 0.0;
        for (int k = 0; k < m; k++) {
            sum += a[k][0] * fs[k](t);
        }
        return sum;
    };
    auto get_params = [&a](int i) {
        return a[i][0];
    };
    write1("params", m, get_params, m);
    auto get_fit = [f](int i) {
        return f(ts[i]);
    };
    write2("fit", m, get_fit, N);
    auto get_error = [F_inv](int i) {
        return F_inv[i][i];
    };
    write1("error", m, get_error, m);
    auto get_chi_sq = [f, m](int j) {
        double sum = 0.0;
        for (int i = 0; i < N; i++) {
            double tmp = (ys[i] - f(ts[i])) / sigmas[i];
            sum += tmp * tmp;
        }
        return sum / (N - m);
    };
    write1("chi_sq", m, get_chi_sq, 1);
}

int main() {
    load_arrays();
    transform(ys.begin(), ys.end(), sigmas.begin(), [](double y) {
        return y * sigma_perc;
    });
    auto get_orig_data = [](int i) {
        return ys[i];
    };
    write2("orig", 0, get_orig_data, N);
    // with first harmonic
    vector<math_func> fs = get_fit_funcs(true);
    write_all_fit_data(fs, true);
    // no first harmonic
    vector<math_func> fs2 = get_fit_funcs(false);
    write_all_fit_data(fs2, true);
    // with seasonal variations removed after fit calculation
    write_all_fit_data(fs, false);
}
```

References

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