```
// regress3.c
// adopted from JavaScript code
// (http://www.wcrl.ars.usda.gov/cec/photo/a2graph.htm)
// does linear, log, exp, pow
// figures out which one produces best results (based on r2)

/*
What do about this one? pow slightly better than lin, yet
clearly should be lin. If get pow, and exponent == 1.0, then
turn into lin?

[pow: a=0.538997 b=1.000000 r2=1.0000000000000036]
[log: a=-10.445281 b=14.916592 r2=0.8307172732790469]
[exp: a=1.531064 b=0.095470 r2=0.8307172732403992]
[lin: a=0.000000 b=1.714286 r2=1.0000000000000004]
(r2 = 1.000000)
pow: y=1.714286x^1.000000
double f(double x) { return 1.714286 * pow(x, 1.000000); }

Need better versions of > == etc.: use DBL_EPSILON???
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <math.h>
#include <float.h>

   double sumx, sumx2, sumy, sumy2, sumxy;

#define MAX_VAL 10240

double x_values[MAX_VAL] = { 0.0 } ;
double y_values[MAX_VAL] = { 0.0 } ;
int numval = 0;
double log_x[MAX_VAL] = { 0.0 } ;
double log_y[MAX_VAL] = { 0.0 } ;

void stats()
{
        int i;
        printf("x\t\tlog(x)\t\ty\t\tlog(y)\n");
        for (i=0; i<numval; i++)
                printf("%f\t%f\t%f\t%f\n",
                        x_values[i], log_x[i],
                        y_values[i], log_y[i]);
        printf("\n");
        printf("sumx = %f\n", sumx);
        printf("sumy = %f\n", sumy);
        printf("sumx2 = %f\n", sumx2);
        printf("sumy2 = %f\n", sumy2);
        printf("sumxy = %f\n", sumxy);
}

void do_linear(double *x, double *y, int n)
{
        int w;
        sumx=sumx2=sumy=sumy2=sumxy=0;
        for (w=0;w<n;w++)
        {
                sumx += x[w];
                sumx2 += x[w]*x[w];
                sumy += y[w];
                sumy2 += y[w]*y[w];
                sumxy += x[w]*y[w];
        }
}

//log y=a+blnX
void do_log(double *x, double *y, int n)
{
```

```
        double lx;
        int w;
        sumx=sumx2=sumy=sumy2=sumxy=0;
        for (w=0;w<n;w++)
        {
                lx = log_x[w];
                sumx += lx;
                sumx2 += lx*lx;
                sumy += y[w];
                sumy2 += y[w]*y[w];
                sumxy += lx*y[w];
        }
}

//exponential y=ae^bX
void do_exp(double *x, double *y, int n)
{
        double ly;
        int w;
        sumx=sumx2=sumy=sumy2=sumxy=0;
        for (w=0;w<n;w++)
        {
                ly = log_y[w];
                sumx += x[w];
                sumx2 += x[w]*x[w];
                sumy += ly;
                sumy2 += ly*ly;
                sumxy += x[w]*ly;
        }
}

/*
Following gets compiler error if use lx, ly:

regress3.c
regress3.c(101) : fatal error C1001: INTERNAL COMPILER ERROR
        (compiler file '@(#)schedp5u.c:1.0', line 2281)
    Please choose the Technical Support command on the Visual Workbench
    Help menu, or open the Technical Support help file for more information
*/

//power y=ax^b
void do_pow(double *x, double *y, int n)
{
        int w;
        sumx=sumx2=sumy=sumy2=sumxy=0;
        for (w=0;w<n;w++)
        {
                sumx += log_x[w];
                sumx2 += log_x[w]*log_x[w];
                sumy += log_y[w];
                sumy2 += log_y[w]*log_y[w];
                sumxy += log_x[w]*log_y[w];
        }
}

double b1, a1, r2;

// switched order of lin and pow, so lin would come later and take
// precedence in case of a tie
char *regress_name[] = { "", "pow", "log", "exp", "lin" } ;

#define MIN_REGRESS         1

#define LINEAR_REGRESS      4
#define LOG_REGRESS         2
#define EXP_REGRESS         3
#define POW_REGRESS         1

#define MAX_REGRESS         4

#define BAD_REGRESS                     -1
```

```
int do_regress(int type, double *x, double *y, int n)
{
        switch (type)
        {
                case LINEAR_REGRESS: do_linear(x, y, n); break;
                case LOG_REGRESS: do_log(x, y, n); break;
                case EXP_REGRESS: do_exp(x, y, n); break;
                case POW_REGRESS: do_pow(x, y, n); break;
        }
    // stats();
        b1 = (n * sumxy - sumx * sumy) / (n * sumx2 - sumx * sumx);
        a1 = (sumy - b1 * sumx) / n;
        r2 = b1 * (n * sumxy - sumx * sumy) / (n * sumy2 - sumy * sumy);
        return type;
}

void fail(const char *s) { puts(s); exit(1); }

void setxyvals(void)
{
        double x, y;
        char buf[1024], *s, *xstr, *ystr;
        char *delim = " \t";
        int linenum = 0;
        while (s = gets(buf))
        {
                linenum++;
                while (isspace(*s)) s++;    // skip white space
                if (! *s) continue;
                if (*s == ';') continue;    // skip over comment lines
                if (*s == '#') continue;
                if (numval >= MAX_VAL)
                        fail("Too many data points");
                xstr = strtok(s, delim);
                ystr = strtok(NULL, delim);
                if ((! (xstr && ystr) || (*ystr == ';')))
                        fail("Missing data"); // TODO: tell them line number
                x = atof(xstr);
                y = atof(ystr);
                x_values[numval] = x;
                y_values[numval] = y;
                log_x[numval] = log(x);
                log_y[numval] = log(y);
        //printf("[%u] (%f,%f)\n", numval, x, y);
                numval++;
        }
}

double a1_weight = 1;
double b1_weight = 1;
int print_row = 0;

void test_predictions(int numval, double *x_values, double *y_values, int save_type, double save_a1, double
save_b1)
{
        int i;
        double x, y_pred, y_actual, y_diff, err, tot_err;

        tot_err = 0;

        save_a1 += a1_weight;
        save_b1 += b1_weight;

        for (i=0; i<numval; i++)
        {
                x = x_values[i];
                y_actual = y_values[i];
                switch (save_type)
                {
                        case LINEAR_REGRESS:    y_pred = save_a1 + (save_b1 * x); break;
                        case LOG_REGRESS:       y_pred = save_a1 + (save_b1 * log(x)); break; // TODO: also do
LOG10 // should be save_a1 * ???
                        case EXP_REGRESS:       save_a1 = exp(save_a1); y_pred = save_a1 * exp(save_a1 * save_b1);
```

```
break;
                        case POW_REGRESS:       y_pred = save_a1 * pow(x, save_b1); printf("%f <= %f * pow(%f,
%f)\n", y_pred, save_a1, x, save_b1); break;
                }
//              y_diff = fabs(y_actual - y_pred);
                y_diff = y_actual - y_pred; // maybe want to know if higher or lower
                err = y_diff / y_actual;
                if (print_row)
                        printf("%f\t%f\t%f\t%f\t%f\n", x, y_actual, y_pred, y_diff, err);
                tot_err += err; // maybe do something more subtle
        }

        printf("[%.04f, %.04f] [%.04f, %.04f] total_error = %0.4f\n", save_a1, save_b1, a1_weight, b1_weight,
tot_err);

        if (tot_err < 0)
        {
                a1_weight += 0.1;
                b1_weight -= 0.1;
        }
        else if (tot_err > 0)
        {
                a1_weight -= 0.1;
                b1_weight += 0.1;
        }
        else { puts("converged"); exit(0); }
}


#define LOG2LOG10       (1.0 / 0.43429448190325154)

main(int argc, char *argv[])
{
        double save_r2, save_a1, save_b1;
        int save_type = BAD_REGRESS;
        int i;
        setxyvals();
        save_r2 = 0;

        for (i=MIN_REGRESS; i<=MAX_REGRESS; i++)
        {
                do_regress(i, x_values, y_values, numval);
                printf("[%s: a=%f b=%f r2=%.16f]\n",
                       regress_name[i], a1, b1, r2);
                if ((r2 > 0) && (r2 <= 1.01)) // hack, to make sure that
                        if (r2 > save_r2)                       // r2 == 1.0 really takes
                        {                                                       // precedence
                                save_r2 = r2;
                                save_type = i;
                                save_a1 = a1;
                                save_b1 = b1;
                        }
        }

        if (save_type == BAD_REGRESS)
        {
                double d;
                int i;
                printf("Regressions didn't work\n");
                // see if all y are the same
                d = y_values[0];
                for (i=1; i<numval; i++)
                        if (d != y_values[i])
                                break;
                if (i==numval) // all y values the same
                        printf("y = %f (many->one)\n", d);
                // see if all x are the same
                d = x_values[0];
                for (i=1; i<numval; i++)
                        if (d != x_values[i])
                                break;
                if (i==numval) // all x values the same
                        printf("x = %f (one->many)\n", d);
```

```
                exit(0);
        }

        printf("(r2 = %f)\n", save_r2);

        printf("%s: ", regress_name[save_type]);

        switch (save_type)
        {
                case LINEAR_REGRESS:
                        printf("y=%f+%fx\n", save_a1, save_b1);
                        printf("double f(double x) { return %f + (%f * x); }\n", save_a1, save_b1);
                        break;
                case LOG_REGRESS:
                        printf("y=%f+%fln(x)\n", save_a1, save_b1);
                        printf("double f(double x) { return %f + (%f * log(x)); }\n", save_a1, save_b1);
                        save_b1 *= LOG2LOG10;
                        printf("double f(double x) { return %f + (%f * log10(x)); }\n", save_a1, save_b1);
                        break;
                case EXP_REGRESS:
                        save_a1 = exp(save_a1);
                        printf("y=%fln^(%fx)\n", save_a1, save_b1);
                        printf("double f(double x) { return %f * exp(%f * x); }\n", save_a1, save_b1);
                        break;
                case POW_REGRESS:
                        save_a1 = exp(save_a1);
                        printf("y=%fx^%f\n", save_a1, save_b1);
                        printf("double f(double x) { return %f * pow(x, %f); }\n", save_a1, save_b1);
                        break;
        }
        printf("\n");

#if 0
        printf("INITIAL:\n");
        print_row = 1;
        test_predictions(numval, x_values, y_values, save_type, save_a1, save_b1);
        printf("WEIGHT LOOP:\n");
        print_row = 0;
        for (i=1; i<20; i++)
                test_predictions(numval, x_values, y_values, save_type, save_a1, save_b1);
        printf("FINAL:\n");
        print_row = 1;
        test_predictions(numval, x_values, y_values, save_type, save_a1, save_b1);
#endif
}
```