Optimization of Continuous Problems

http://people.sc.fsu.edu/~jburkardt/isc/week11_lecture_20.pdf

.....

ISC3313:
Introduction to Scientific Computing with C++
Summer Semester 2011

.....

John Burkardt
Department of Scientific Computing
Florida State University

Last Modified: 17 July 2011

Optimization of Continuous Problems

- **Introduction**
- Fitting a Straight Line
- The Linear Least Squares Algorithm
- Minimizing by Fitting Parabolas
- Monte Carlo Minimization
- Using the BRENT Package
- Lab Exercise #10

INTRO: Schedule

Next Class:

- Optimization of Discrete Problems

Assignment:

- Today: in-class lab exercise #10
- Thursday: Programming Assignment #8 is due.

INTRO: Two Weeks Left For Projects!

Please remember that your project is due on Tuesday, August 2nd, at class time, 11am! This includes:

- a 5 minute oral presentation;
- a 3-5 page report to be turned in;
- a C++ program, to be turned in.

If you do not present your talk and turn in your report and programs on time, you will not receive a grade for the course.

INTRO: Optimization = Minimize or Maximize F(X)

This week, we consider the scientific computing topic of **optimization**. In computing, this term is generally used when we have one or more variables such as x, whose value varies over some range, and we have a function $f(x)$ which measures a cost, a profit, or some other quantity. Depending on the meaning of $f(x)$, we might seek an x which maximizes or minimizes this value.

If the variable x is a real number that can vary continuously throughout a range such as $a \leq x \leq b$, then we speak of a *continuous optimization problem*. If, instead, x can only take values from some finite set (or occasionally also an indexed infi sequence), we speak of a *discrete optimization problem*.

INTRO: Local and Global Minimizers

Today, we will concentrate on the continuous optimization problem. We will start with the most common such task, which is to fit a straight line to a set of data points.

If, instead of data points, we have a function $f(x)$, we can try to find the minimum of this function by estimating the shape of the graph using parabolas.

Whenever the graph of the function is flat (the derivative is zero), then the function might have what is called a local minimizer, that is, a value which is smaller than all its immediate neighbors.

Often, a function may have several local minimizers in an interval, but we probably are seeking a global minimizer, that is, a point where the function reaches its lowest value compared to all other points in the interval. This is a significantly harder problem.

INTRO: Monte Carlo or Brent For Global Minimizers

We will take a stab at solving the global minimizer problem for a wiggly function with lots of local minimizers by a sort of Monte Carlo approach.

Then we will look at a more sophisticated procedure, from a library by Richard Brent, for finding the zeros, local minimizers, or global minimizer of a function. Instead of writing such a procedure ourselves, we will concentrate on trying to understand how such a library can be used conveniently from our C++ programs.

Our in-class exercise will ask you to solve a simple problem using Richard Brent's library.

Optimization of Continuous Problems

- Introduction
- **Fitting a Straight Line**
- The Linear Least Squares Algorithm
- Minimizing by Fitting Parabolas
- Monte Carlo Minimization
- Using the BRENT Package
- Lab Exercise #10

LINEFIT: Problem #1 is Just Four Data Points

Our first example is completely artificial. We'll take four pairs of values (x, y) which don't lie on a straight line, and "wish" they did.

We might imagine that these four items of data are controlled by some unknown physical law, whose form is linear: $y = a * x + b$, or that the law is "mostly linear", with some lesser effects we can ignore for now.

It is also possible that our data has small errors in it, so that the fact that our points don't lie on a line is simply because we were inaccurate in our measurements.

Our main reason for this first data set is that it is so small that it will be easy to work out the results by hand.

LINEFIT: Data Values for Problem #1

problem1_data.txt:

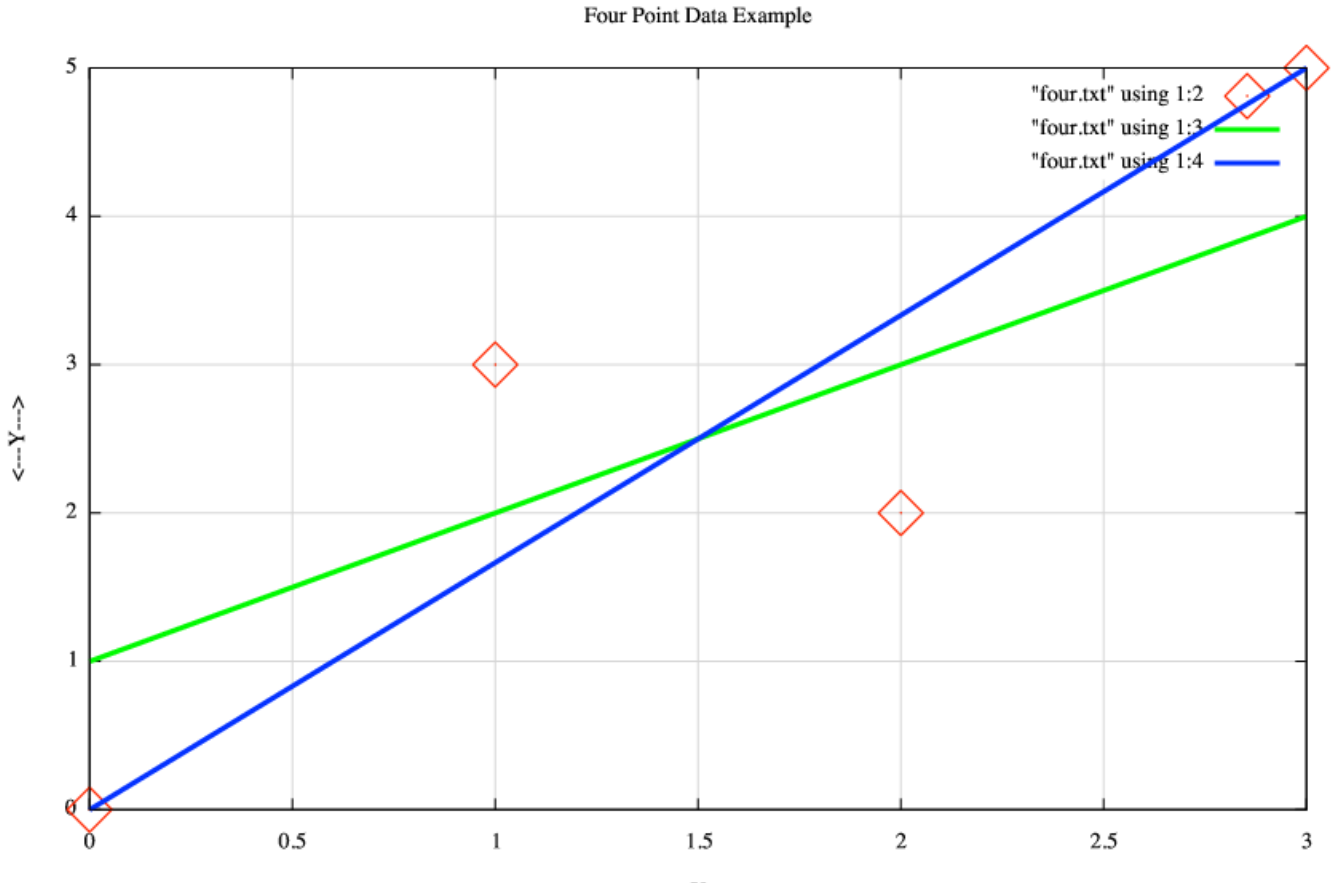
```
  X       Y
---    ------
0.0    0.0
1.0    3.0
2.0    1.0
3.0    5.0
```

Two reasonable candidates for approximating this data might be $y = x + 1$ and $y = \frac{5}{3}x$.

```
  X       Y       X+1      5/3 X
---    ------    ---      -----
0.0    0.0       1.0       0.0
1.0    3.0       2.0       1.66
2.0    1.0       3.0       3.33
3.0    5.0       4.0       5.0
```

LINEFIT: Graphing Candidate Lines for Problem #1

```
plot "problem1_data.txt" using 1:2 with points ps 4 pt 5,
     "problem1_data" using 1:3 with lines lw 3,
     "problem1_data" using 1:4 with lines lw 3
```



LINEFIT: Problem #2 Measures Dye Concentration

Dye was spilled into a lake, which is slowly drained by a stream.

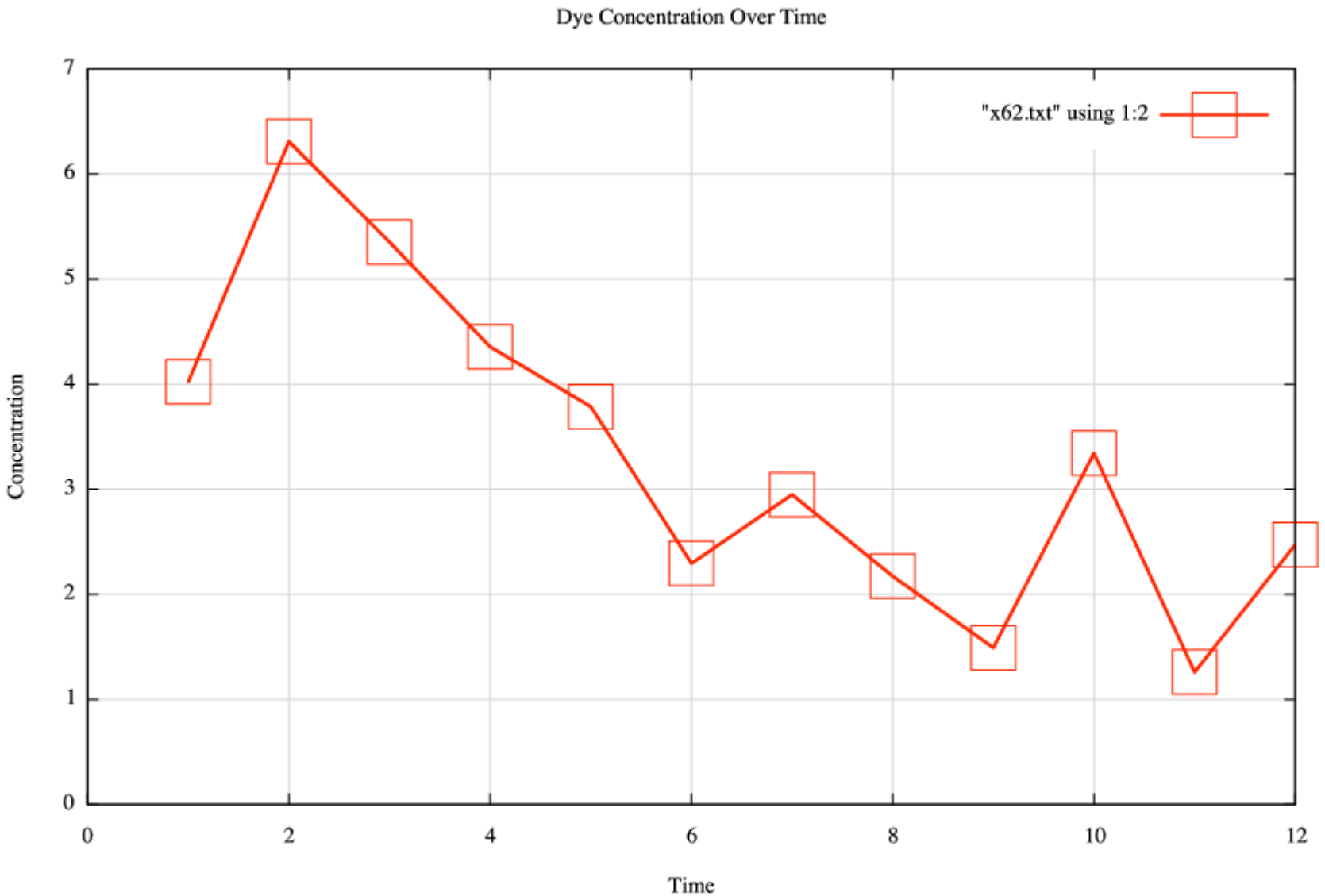
Measurements of the concentration of the dye were made daily for 12 days. A simple linear function $d = a * t + b$ is desired, both to try to estimate when the dye will be almost completely gone, and to be able to understand the rate at which the lake can clean itself in case of future problems.

LINEFIT: Data for Problem #2

T	D
1.0	4.0225
2.0	6.3095
3.0	5.3522
4.0	4.3553
5.0	3.7861
6.0	2.2947
7.0	2.9492
8.0	2.1732
9.0	1.4921
10.0	3.3424
11.0	1.2596
12.0	2.4732

LINEFIT: Plotting Data for Problem #2

plot "problem2_data.txt" using 1:2 with linespoints lw 3 pt 4 ps 5



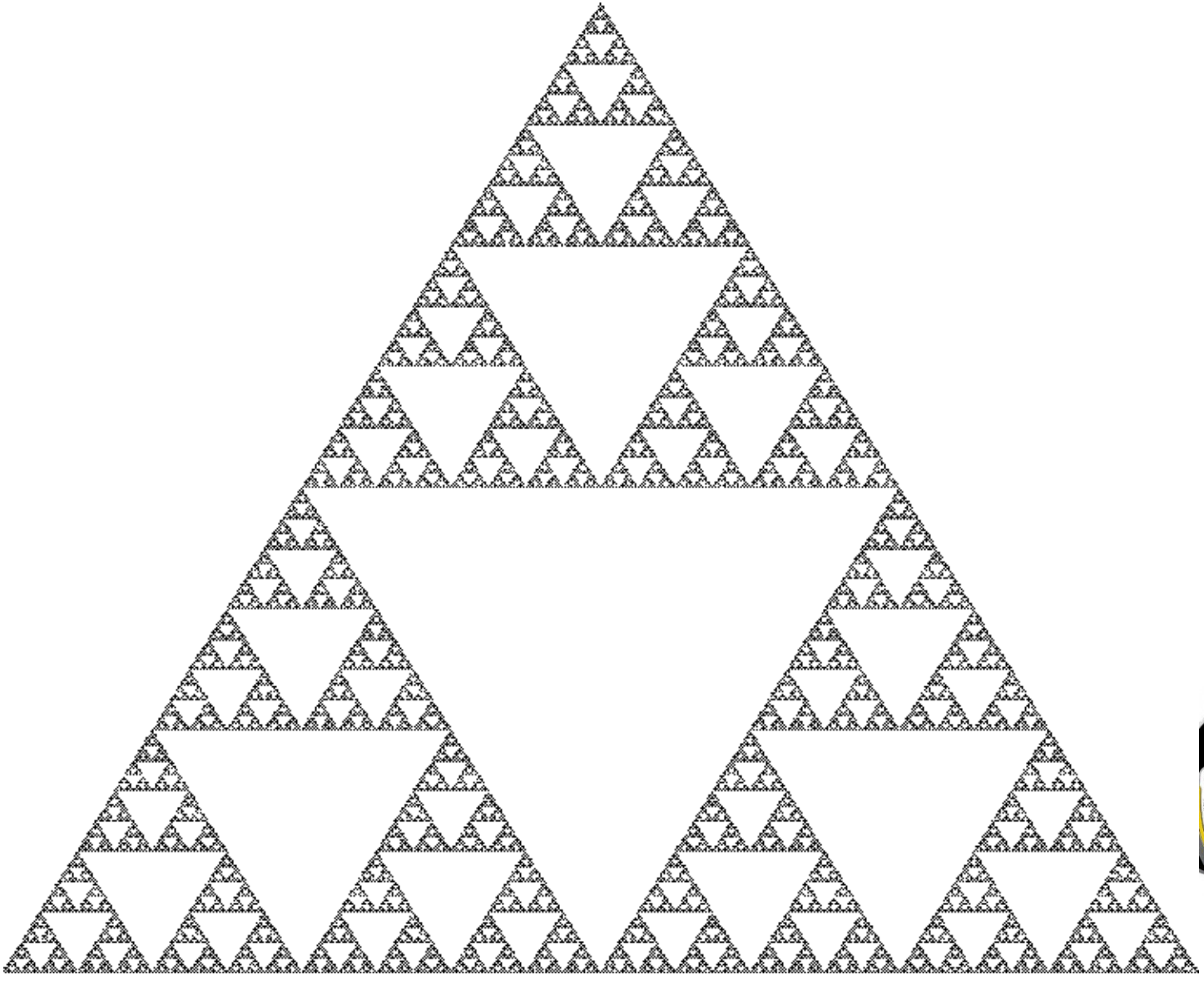
LINEFIT: Problem #3 Counts Filled in Image Boxes

An image consists of pixels, which are black or white.

We break the image into NxN squares, and count B(N), the squares which have at least one black pixel.

We do this letting the squares decrease in size from 512x512 down to 1x1.

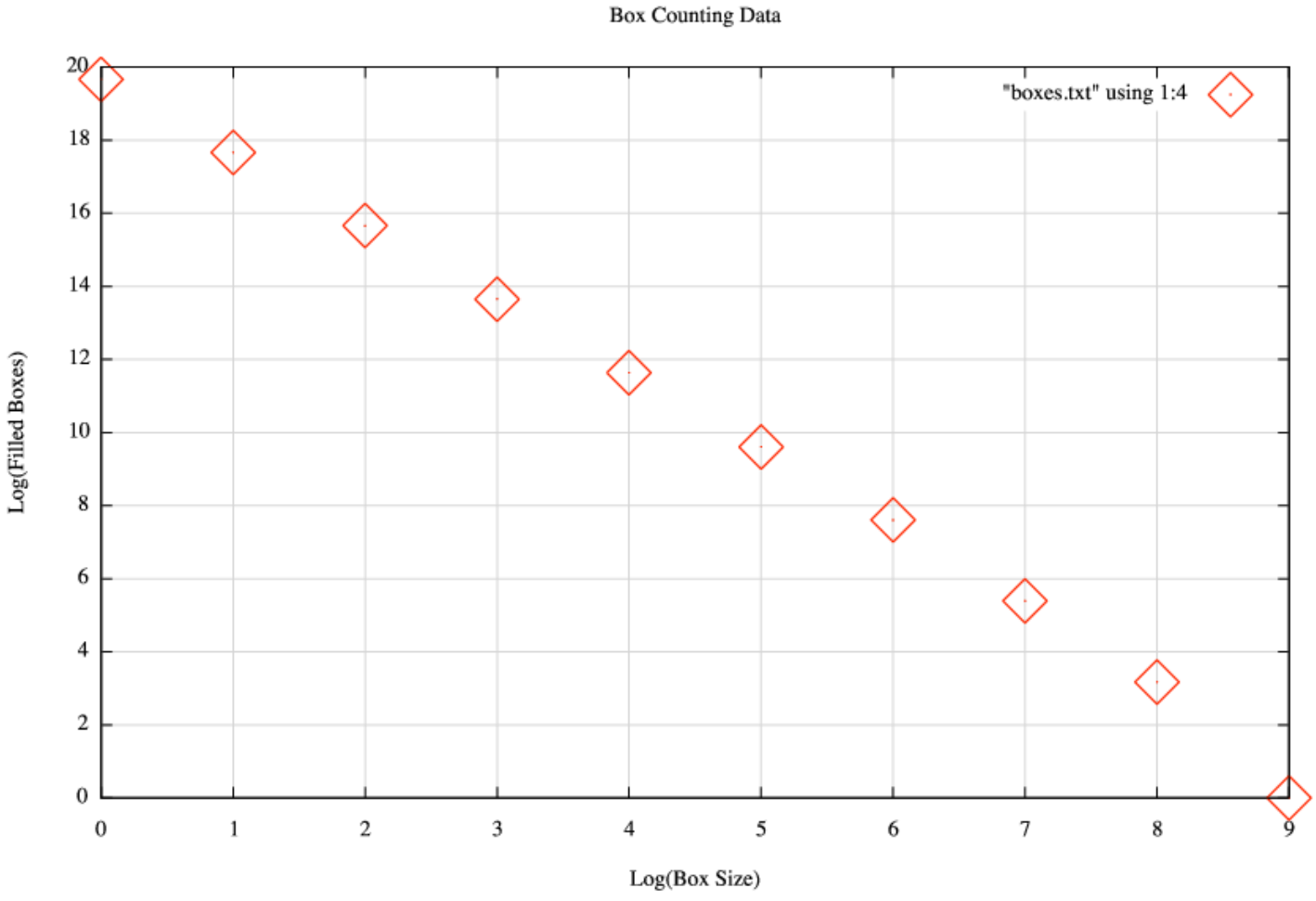
LINEFIT: The Image Used in Problem #3



LINEFIT: The Data for Problem #3

N	Log(N)	B(N)	Log(B(N))
1	0	830760	19.6641
2	1	207690	17.6641
4	2	51815	15.6611
8	3	12840	13.6484
16	4	3180	11.6348
32	5	780	9.60733
64	6	195	7.60733
128	7	42	5.39232
256	8	9	3.16993
512	9	1	0

LINEFIT: The Log/Log Plot of Problem #3



LINEFIT: The Linear Least Squares Problem

These problems seek a formula $y = a * x + b$ which will approximate the given data. If we can't reproduce the data exactly, then any formula can be regarded as an approximation. So how do choose a satisfactory one?

Given an approximation formula for our **n** items of data, (x_i, y_i), we can evaluate the error:

$$e_i = a * x + b - y_i$$

If we square each error, sum them, average, and take the square root, we have what is called the **RMS** or root-mean-square error. It's a single number, it's zero if the matching is perfect, it's small if the matching is good, and, because of the squaring, it prefers many small errors rather than a few big ones.

LINEFIT: The Linear Least Squares Problem

For our example data, let's set how the RMS error chooses between the two approximations $y = x + 1$ and $y = \frac{5}{3}x$:

x_i	y_i	x+1	e^2	5/3x	e^2
0	0	1	1	0	0
1	3	4	1	5/3	16/9
2	2	3	1	10/3	16/9
3	5	4	1	5	0
Sum			4		32/9
1/n			1		8/9
Sqrt			1		0.942

so the RMS error prefers the approximation $y = \frac{5}{3}x$.

Optimization of Continuous Problems

- Introduction
- Fitting a Straight Line
- **The Linear Least Squares Algorithm**
- Minimizing by Fitting Parabolas
- Monte Carlo Minimization
- Using the BRENT Package
- Lab Exercise #10

LLSQ: An Algorithm for the Line Fitting Problem

If we agree that one approximation is better than another if it has a lower RMS error for our data, then there is a solution to the line fitting problem, called the linear least squares algorithm.

While it can handle more complicated problems, in which we wish to fit a quadratic or cubic polynomial, for instance, the algorithm is particularly simple in the case where we are seeking a linear formula $y = a * x + b$, and we will concentrate on that question here.

LLSQ: Formulas for Slope and Intercept

So suppose we have n data items (x_i, y_i), and define averages:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$
$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

Then our slope and intercept for $y = a * x + b$ are:

$$a = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$
$$b = \bar{y} - a * \bar{x}$$

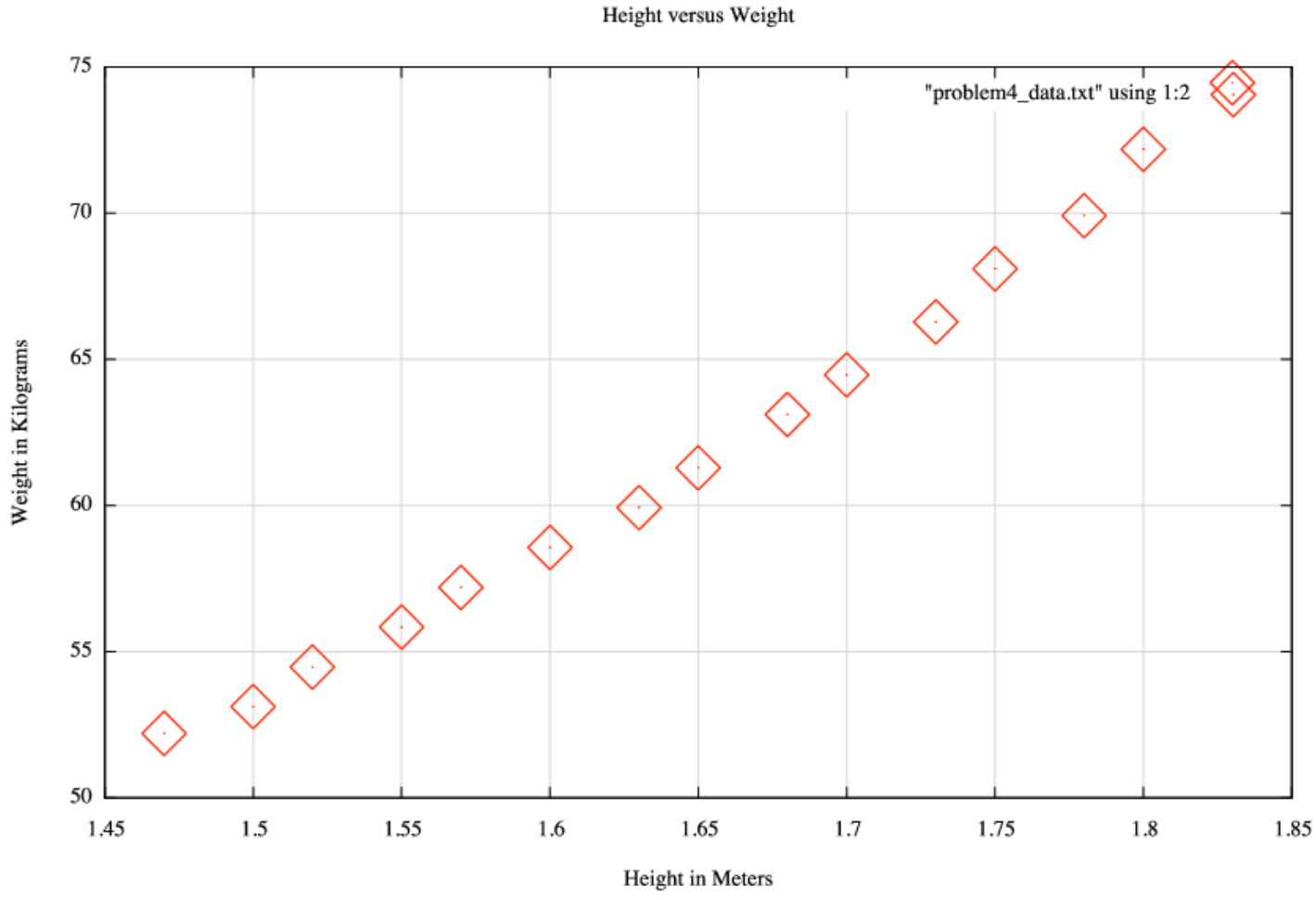
LLSQ: A Program for Slope and Intercept

llsq.cpp:

```
void llsq ( int n, double x[], double y[], double &a, double &b )
{
  double bot = 0.0, top = 0.0, xbar = 0.0, ybar = 0.0;
  int i;

  for ( i = 0; i < n; i++ )
  {
    xbar = xbar + x[i];
    ybar = ybar + y[i];
  }
  xbar = xbar / ( double ) n;
  ybar = ybar / ( double ) n;

  for ( i = 0; i < n; i++ )
  {
    top = top + ( x[i] - xbar ) * ( y[i] - ybar );
    bot = bot + ( x[i] - xbar ) * ( x[i] - xbar );
  }
  a = top / bot;
  b = ybar - a * xbar;
  return;
}
```

The peculiar definitions of **a** and **b** in the header for the function mean these variables are output quantities!

LLSQ: Data for Problem #4 to be Fitted by a Line



LLSQ: A Main Program to Call LLSQ

llsq_prb.cpp:

```
# include <cstdlib>
# include <iostream>
# include "llsq.hpp"    <-- Using llsq as a separate library.

using namespace std;

int main ( )
{
  double a, b;
  int n = 15;
  double x[15] = {
    1.47, 1.50, 1.52, 1.55, 1.57, 1.60, 1.63, 1.65, 1.68, 1.70,
    1.73, 1.75, 1.78, 1.80, 1.83 };
  double y[15] = {
    52.21, 53.12, 54.48, 55.84, 57.20, 58.57, 59.93, 61.29, 63.11, 64.47,
    66.28, 68.10, 69.92, 72.19, 74.46 };

  llsq ( n, x, y, a, b );  <-- The variables a and b are output from llsq!

  cout << "  Approximating line is y = " << a << " * x + " << b << "\n";

  return 0;
}
```

LLSQ: Using Two Files to Create an Executable Program

Remember that in C++, it is perfectly OK to build a program out of separate files. In this case, we will have the file **llsq.cpp** which contains the algorithm for solving the linear least squares problem we are interested in, and a second file, **llsq_prb.cpp**, a main program which sets up data for the problem.

To use two files in this way we must ensure that

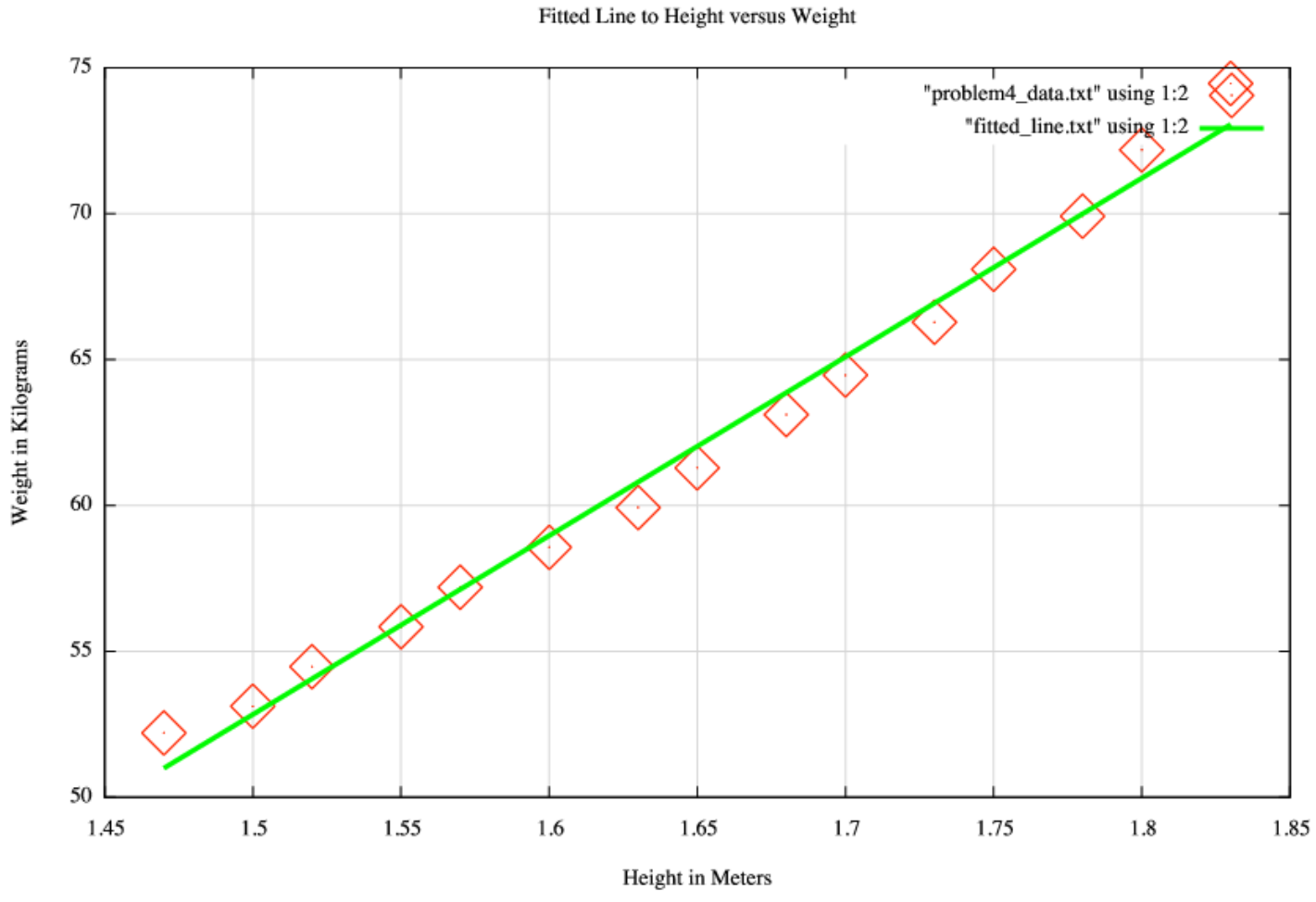
- the main program declares all functions used from the other file, or uses an include file if it is available;
- we compile the two files together to make one executable.

The simplest compile statement would be

```
g++ llsq_prb.cpp llsq.cpp
```

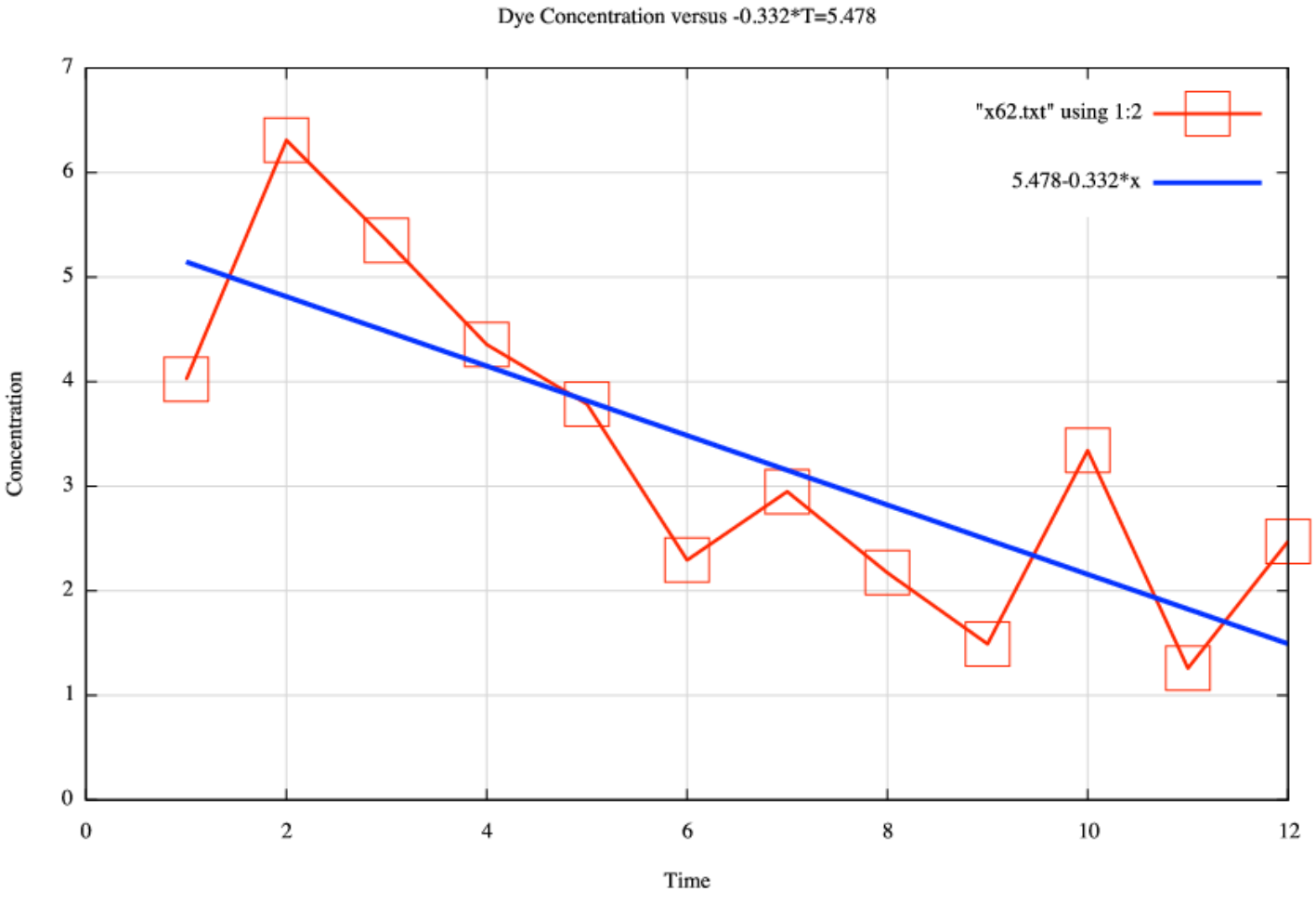
LLSQ: Results of Fitting a Line to Problem #4

$$y = 61.27 * x - 39.06$$



LLSQ: Results of Fitting a Line to Problem #2

The dye pollution problem



Optimization of Continuous Problems

- Introduction
- Fitting a Straight Line
- The Linear Least Squares Algorithm
- **Minimizing by Fitting Parabolas**
- Monte Carlo Minimization
- Using the BRENT Package
- Lab Exercise #10

PARABOLA: Seeking the Minimizer of a Function F(X)

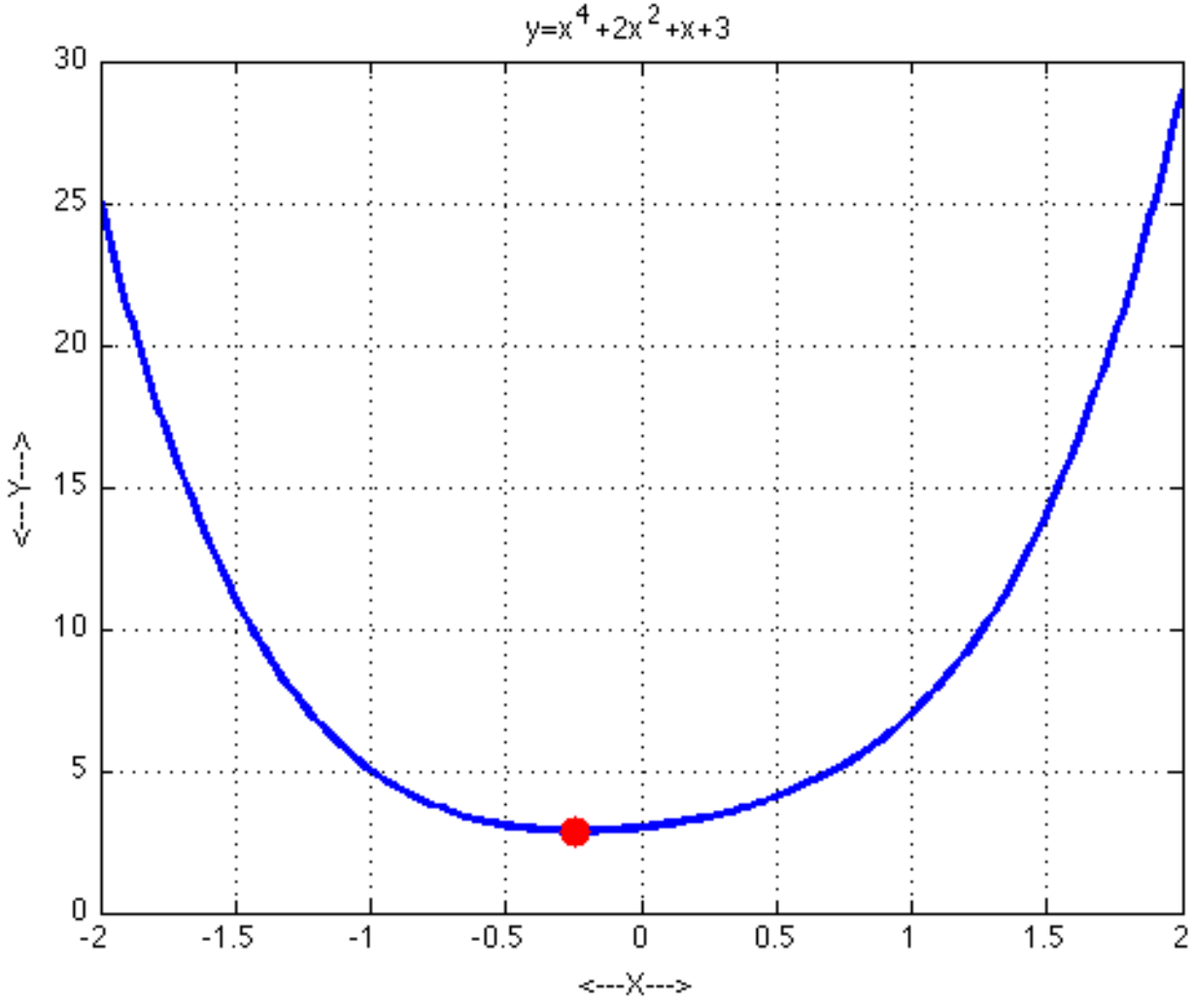
We chose the LLSQ algorithm to determine a straight line that was close to our data. We could think of the LLSQ algorithm as trying to minimize the RMS error associated with the choice of the values of the y-intercept b and slope a.

There are many important problems in mathematics, science, computing, and industry which can be regarded as variations of this problem, that is, choose a value for the variable x, or for several such variables, so that we

- **minimize** some quantity $f(x)$ representing error, or cost;
- or **maximize** some quantity $f(x)$ measuring profit or accuracy or reward.

We'll think of the first case, and we may call $f(x)$ the cost function.

PARABOLA: A Function F(X) for Problem 5



PARABOLA: Three Sample Points Suggest a Parabola

Our first idea should be to deal with the simplest problem, in which, over the range of x values we are interested in, the graph of the function looks like a valley.

Knowing two points on the graph is enough to draw a straight line, but knowing three points will give us a parabola. And if the graph is really valley-shaped, then our parabola is likely to have a minimum and to have it close to where the correct minimum is.

In other words, we are going to "model" the function $f(x)$ by a parabola, minimize the model function, and hope that that tells us something about the real problem.

PARABOLA: Finding the Fitting Parabola

To fit parabolas, we need to take sets of three data points (x_1, y_1), (x_2, y_2), (x_3, y_3) and determine the coefficients of the parabola $y = ax^2 + bx + c$ that passes through those points.

If we write the value of the parabola at each point, we get a system of linear equations for (a, b, c):

$$\begin{aligned} ax_1^2 + bx_1 + c &= y_1 \\ ax_2^2 + bx_2 + c &= y_2 \\ ax_3^2 + bx_3 + c &= y_3 \end{aligned}$$

which can be solved for (a, b, c):

$$\begin{pmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{pmatrix} * \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$$

PARABOLA: The Algorithm

The algorithm for minimizing $f(x)$ with parabolas is as follows:

We begin with three points x_1, x_2 and x_3;
we evaluate $y = f(x)$ to get y_1, y_2 and y_3.

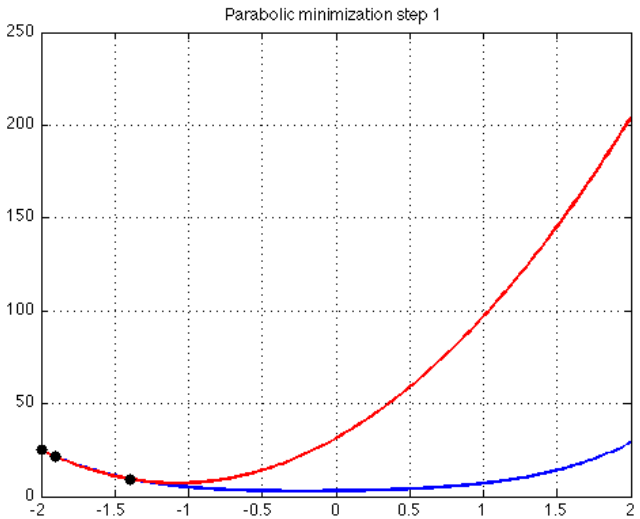
1. Determine the parabola $y = ax^2 + bx + c$ through (x_1, y_1), (x_2, y_2), (x_3, y_3);
2. This parabola is minimized at $x_4 = -\frac{b}{2a}$. Compute $y_4 = f(x_4)$;
3. Stop if you think you found the minimizer;
4. Stop if the iteration is not converging;
5. Otherwise, shift the data $(x_1 \leftarrow x_2, x_2 \leftarrow x_3, x_3 \leftarrow x_4)$ and take another step.

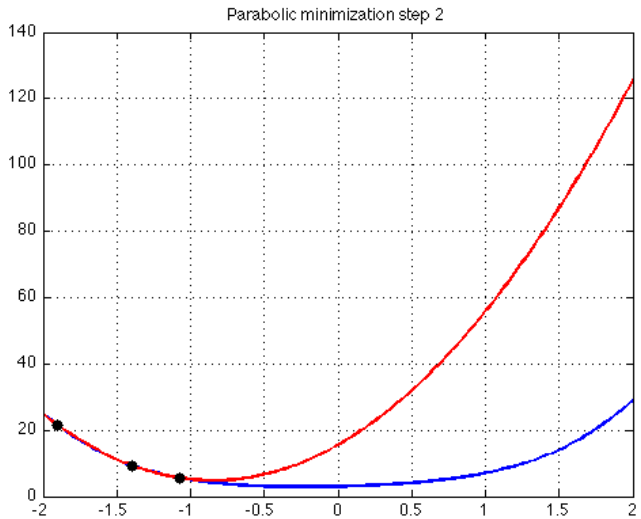
PARABOLA: Results for Problem 5

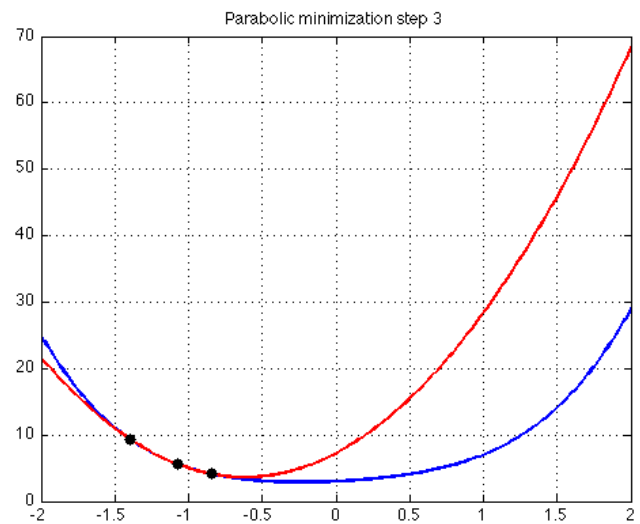
Using starting points -2, -1.9, and -1.4 for the function $y = x^4 + 2x^2 + x + 3$, the algorithm converges in about 12 steps:

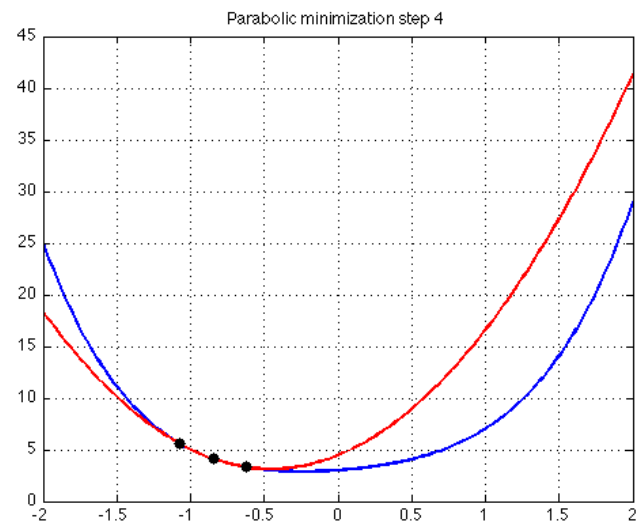
X	X-XOLD	P(X)	F(X)	|P-F|
-1.074364	0.325636	7.152810	5.566463	1.586350
-0.846867	0.227497	4.793781	4.101853	0.691928
-0.619298	0.227569	3.613444	3.294858	0.318586
-0.454061	0.165237	3.121355	3.000788	0.120567
-0.338809	0.115252	2.941053	2.903951	0.037102
-0.271005	0.067804	2.888553	2.881276	0.007276
-0.244393	0.026612	2.879320	2.878630	0.000689
-0.237723	6.66e-03	2.878520	2.878495	2.45e-05
-0.236796	9.27e-04	2.878493	2.878493	2.38e-07
-0.236735	6.15e-05	2.878493	2.878493	4.45e-1
-0.236733	1.64e-06	2.878493	2.878493	9.76e-14

PARABOLA: Step 1, 2, 3, 4

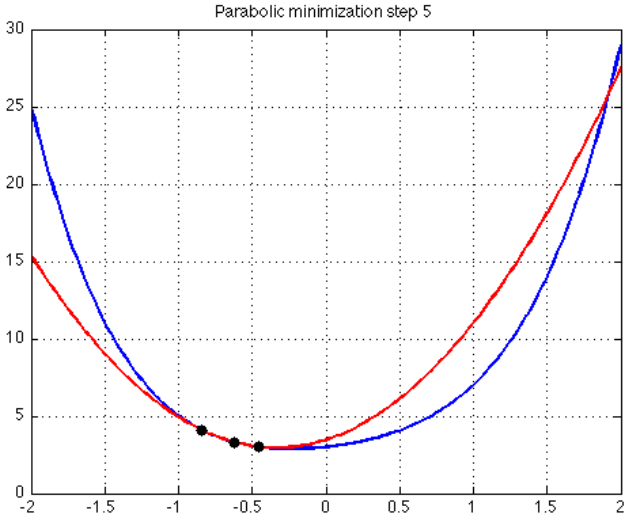


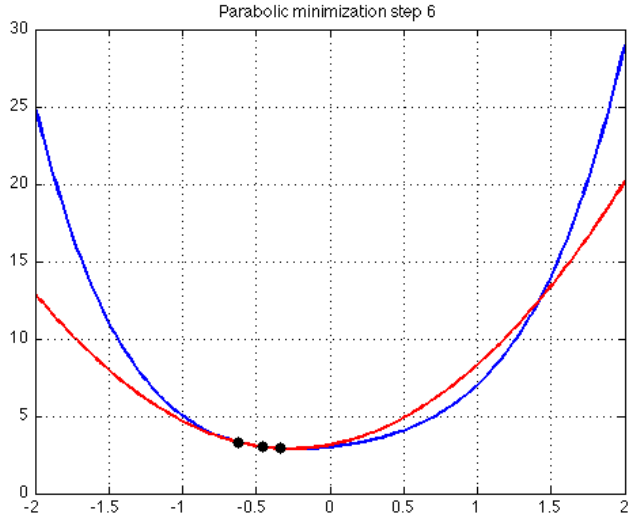


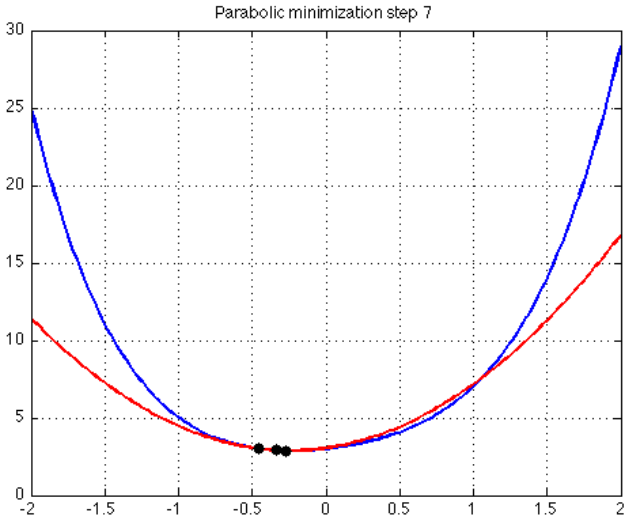


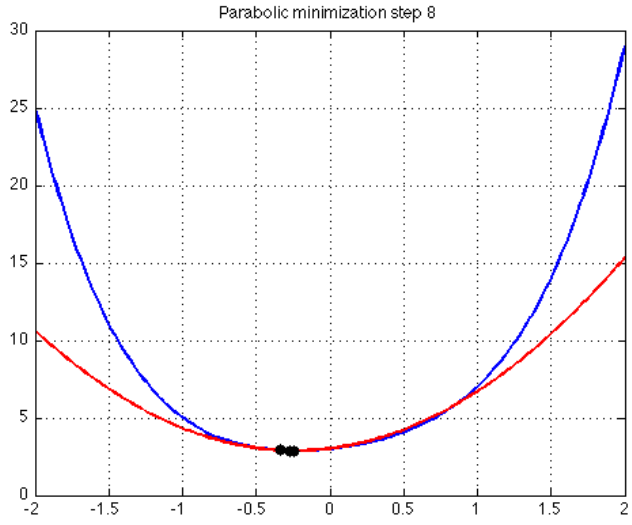


PARABOLA: Steps 5, 6, 7, 8









Optimization of Continuous Problems

- Introduction
- Fitting a Straight Line
- The Linear Least Squares Algorithm
- Minimizing by Fitting Parabolas
- **Monte Carlo Minimization**
- Using the BRENT Package
- Lab Exercise #10

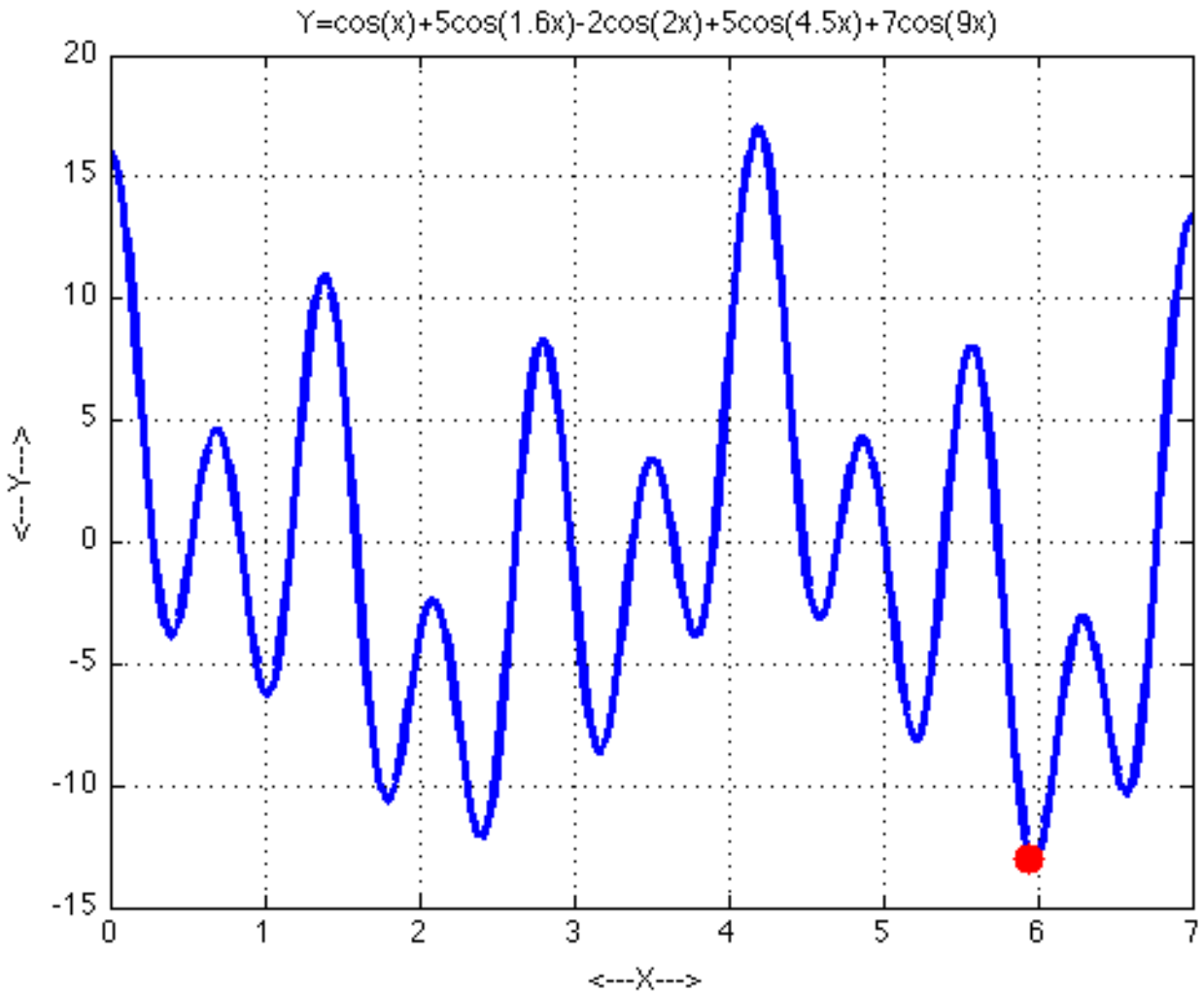
MONTE: Multiple Minimizers Make Difficulties

While we can spot the minimum on these graphs, we have to consider how a computer would do so; moreover, we have to realize that we used several hundred points to make the graph, and really only got one or two digits of accuracy in the location.

The problem becomes hard if the function f(x) is discontinuous, or not differentiable, or oscillatory (very wiggly).

Moreover, while we look at the whole graph, and see the big picture, a computer will concentrate on one part of the graph. If the graph has a local minimum, like a dent where a marble would come to rest, it might incorrectly halt there.

MONTE: A Function for Problem 6



MONTE: A Monte Carlo Approach

If we think the graph of our function has a number of local minimums within the interval $[a, b]$, then almost any minimizing algorithm is just as likely to get stuck at a local minimum as to find the best one.

This is another case where a Monte Carlo approach and help. If we assume that each starting point will most likely "roll downhill" to the nearest minimizer, then if we pick enough starting points at random, we have a good chance of finding the best minimizer... unless it is really trying to hide from us.

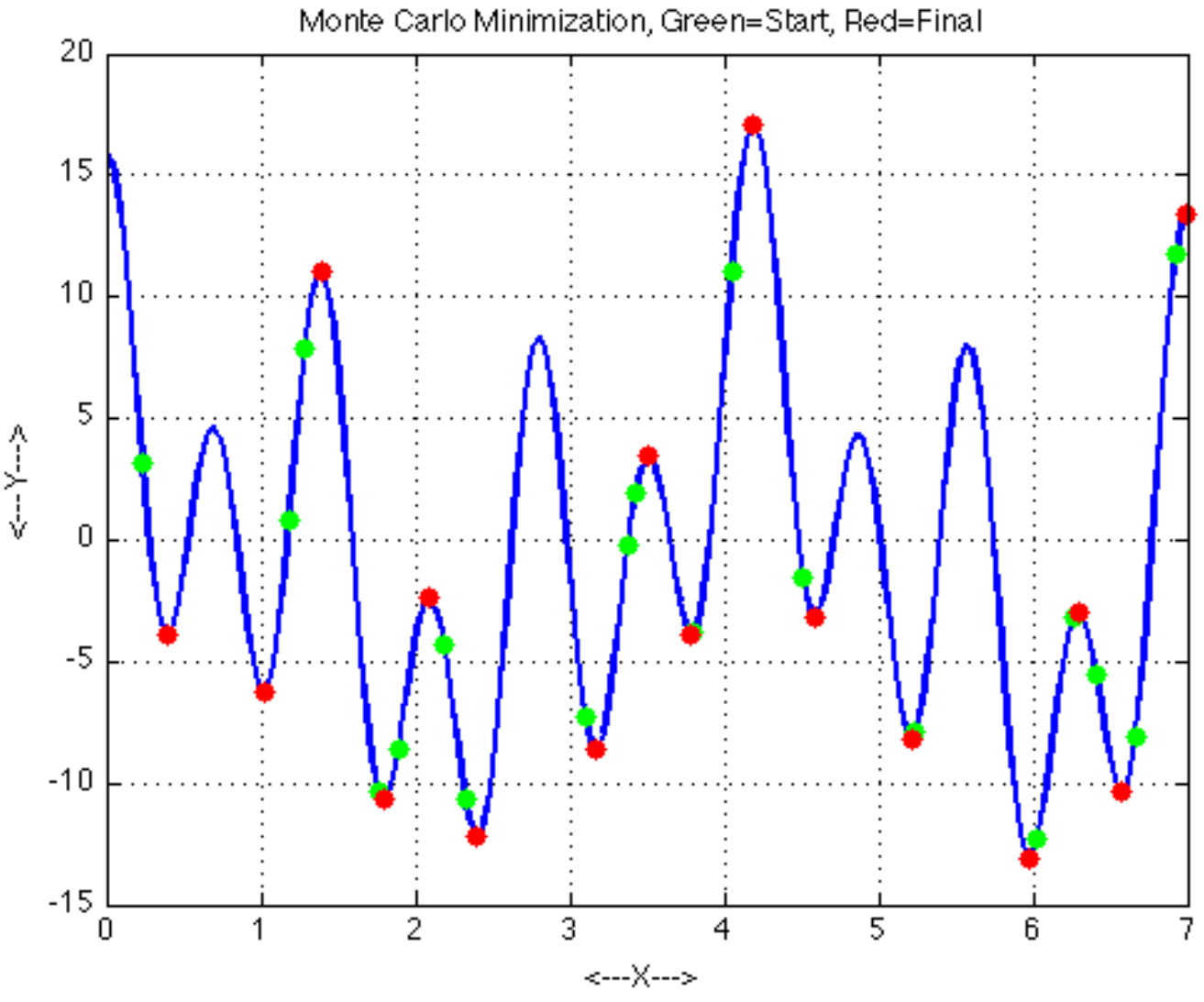
MONTE: Outline of an Algorithm

A simple procedure initializes $xbest = a$, $fbest = f(a)$, and then:

1. Choose x_1 at random in $[a, b]$, and choose x_2, x_3 "close" to x_1.
2. Call the parabolic minimizer function to compute *xmin*, *fmin* with these starting values, and a maximum of 15 steps.
3. If $fmin < fbest$, then $(xbest \leftarrow xmin, fbest \leftarrow fmin)$.
4. If results are satisfactory, quit.
5. Take another step, if we haven't reached our limit.

At the end, *xbest* contains our best estimate for the location of the minimizer, and *fbest* contains its value.

MONTE: Where We Started, What We Found



MONTE: Minimization Can Be Difficult

One thing this exercise points out is a flaw in the parabolic minimizer. As written, the parabolic minimizer takes the data points and finds a parabola through them, and then goes to ...the minimizer???...no, it goes to the point $x = -\frac{b}{2a}$, that is, the place where the derivative is zero, which might be a minimum or maximum.

And sure enough, if our starting Monte Carlo point is near a place on the graph that looks like a hill, rather than a valley, the parabolic minimizer marches uphill to the maximum. A smarter parabolic algorithm would realize this problem and march downhill in such a case.

Optimization of Continuous Problems

- Introduction
- Fitting a Straight Line
- The Linear Least Squares Algorithm
- Minimizing by Fitting Parabolas
- Monte Carlo Minimization
- **Using the BRENT Package**
- Lab Exercise #10

BRENT: A Minimization Library

The minimization problem is hard, especially if the function is not a simple valley shape, has many oscillations, and if it is important to try to find the lowest possible value, without being distracted by local minimums.

There are good algorithms to handle this problem, based on some of the ideas we have discussed, but using a more careful procedure to catch the special things that can happen. It is not necessary for the average user to understand how these algorithms work, but simply to know how to use them.

One example of such an algorithm is a set of minimization routines by a famous computer scientist and mathematician named Rich Brent. We will take a look at how to try to solve some of our minimization problems using software from his library.

BRENT: Using the Brent Library

The BRENT package is stored in a file called **brent.cpp**, and has an include file called **brent.hpp** which declares all the functions.

A user program must

- declare the BRENT functions by **#include "brent.hpp"**;
- initialize problem data;
- define $f(x)$ with a C++ function **f()**;
- call the appropriate function from the BRENT library.

Given $f(x)$, BRENT has procedures to seek a global minimizer, a local minimizer or a zero.

BRENT: Some Functions in the Brent Library

Some of the BRENT functions are:

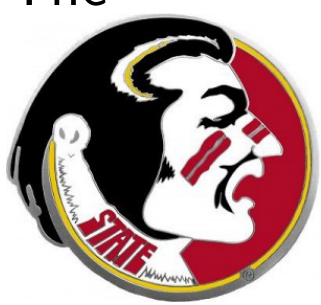
- double **glomin** (double a, double b, double c, double m, double e, double t, double f (double x), double &x) seeks a global minimizer;
- double **local_min** (double a, double b, double eps, double t, double f (double x), double &x) seeks a local minimizer;
- double **zero** (double a, double b, double t, double f (double x)) seeks a solution of $f(x) = 0$;

BRENT: Using GLOMIN() for Global Minimization

I need to supply to **glomin()** the following information:

- **a, b**: the left and right interval endpoints;
- **c**, an estimate for the solution, or just the midpoint;
- **m**, a bound for the second derivative;
- **e**, an error tolerance for f;
- **t**, an error tolerance for x;
- **f**, the name of the function that evaluates $f(x)$;

The return value of **glomin** is the minimum function value. The variable **x** returns the location where this minimum occurs.



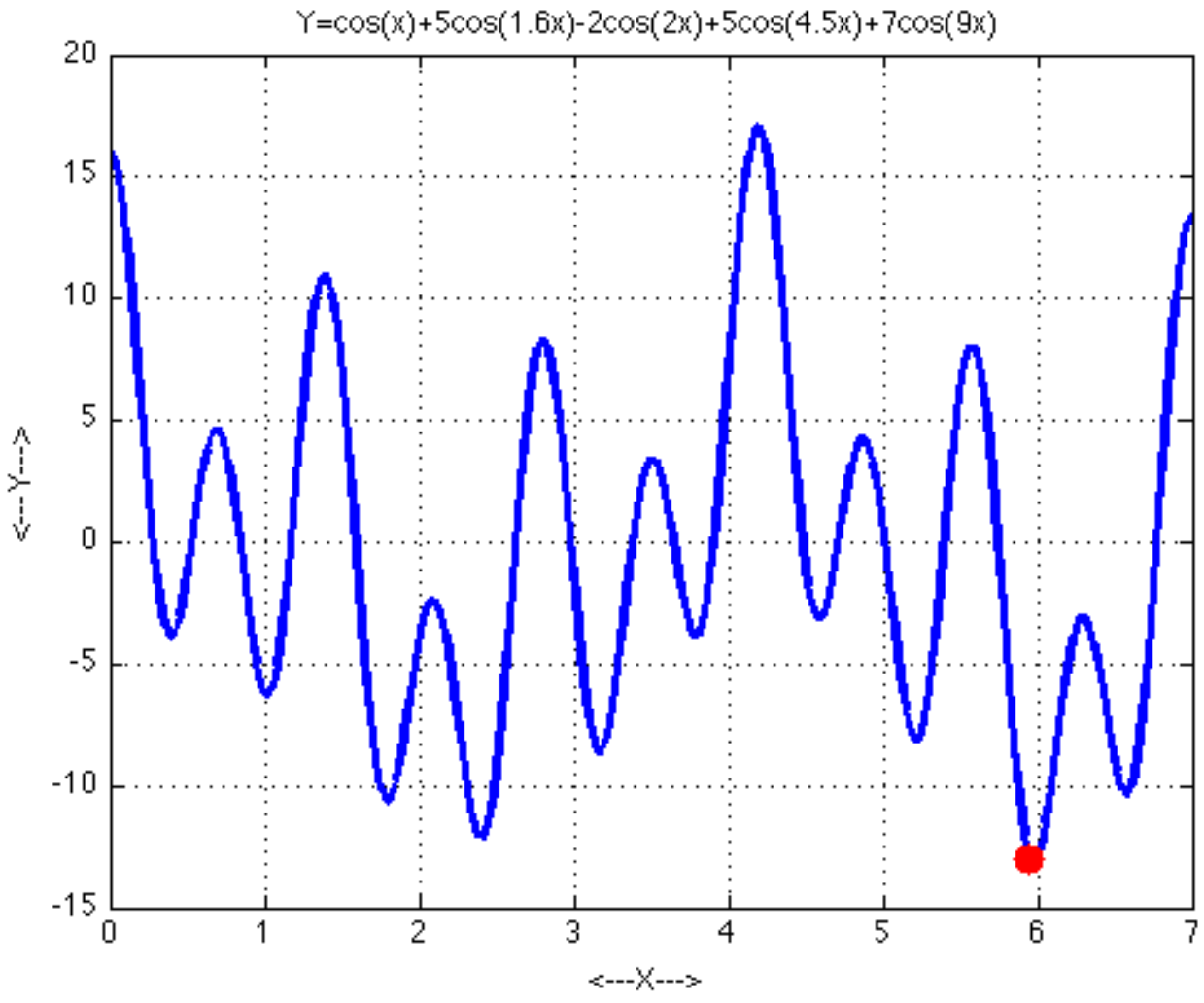
BRENT: A Main Program for Problem 6

brent_problem6.cpp:

```
# include <cstdlib>
# include <iostream>
# include <cmath>
# include "brent.hpp"
double f6 ( double x );
using namespace std;

int main ( )
{
  double a = 0.0, b = 7.0, c = 3.5, fmin, m = 101.0, e = 0.000001,
  t = 0.000001, xmin;

  fmin = glomin ( a, b, c, m, e, t, f6, xmin );

  cout << "\n";
  cout << "GLOMIN returns minimum F = " << fmin << " at X = " << xmin << "\n";

  return 0;
}
double f6 ( double x )
{
  double value;

  value =     cos (       x )
        + 5 * cos ( 1.6 * x )
        - 2 * cos ( 2.0 * x )
        + 5 * cos ( 4.5 * x )
        + 7 * cos ( 9.0 * x );

  return value;
}
```

BRENT: GLOMIN() Does Its Job

"GLOMIN returns minimum F = -12.7343 at x = 6.01134"



Optimization of Continuous Problems

- Introduction
- Fitting a Straight Line
- The Linear Least Squares Algorithm
- Minimizing by Fitting Parabolas
- Monte Carlo Minimization
- Using the BRENT Package
- **Lab Exercise #10**

EXERCISE: $f(x) = (x + \sin(x)) * \exp(-x^2)$

Consider the problem of finding a local minimizer within the interval [-5,1] for the function $f(x) = (x + \sin(x)) * \exp(-x^2)$.

To get an idea of what is going on, let's first sneak a peek with GNUPLOT. GNUPLOT can plot a formula as easily as it can plot data from a file.

```
gnuplot
  set xrange [-5:1]
  plot (x+sin(x)) * exp(-x*x) with lines
```

Do you see that there are two possible answers to the local minimum problem in this interval?

EXERCISE: Use LOCAL_MIN for a Local Minimizer

BRENT can solve this problem, using the function:

```
double local_min ( double a, double b, double t,
  double f ( double x ), double &x );
```

(Don't worry about the & in front of the **x** variable. That just means that **local_min()** can use it as an output variable.)

To use **local_min()**, you supply the following information:

- **a, b**: the left and right interval endpoints;
- **t**, an error tolerance;
- **f**, the name of the function that evaluates $f(x)$;

The return value of **local_min** is the minimum function value, and the variable **x** returns the location where this minimum occurs.

EXERCISE: Input Data for LOCAL_MIN

Write a program which uses the BRENT function **local_min()** to find a local minimizer of $f(x) = (x + \sin(x)) * \exp(-x^2)$.

Search within the interval $a = -5.0 \leq x \leq b = 1.0$.

Use an error tolerance of t = 0.000001;

Remember that your program doesn't need to include the actual text of the BRENT library, but it does need an "include" file.

When you compile your program, remember to list both your own program and the BRENT library.

Show your results to Detelina so you can get credit for the exercise!