MOVING AVERAGES

LAB 8

SECTION A

SUBMITTED BY:

BRIAN REBER

SUBMISSION DATE:

11/13/2009

Lab Problem
Our goal is to use the Wiimote to help us learn about and work with arrays. We want to learn about how to work with arrays, including how to use them in functions. We will also be learning about moving averages and their effects on data.

Analysis
The goal of this lab is to compute and print out the current acceleration values, the average acceleration for a user-specified amount of values prior, and the max and min values each time through the loop. To do this we will need an array, and need to get the user specified number from the command line. We will then fill the array and calculate the listed values by passing our array into different functions that will compute the values.

Inputs:
There will be data coming in via the Wiimote and Wiiwrap.

Outputs:
The output should look like this.

1.05, 1.98, .05, 1.00, 1.15, .98, 3.01, 2.65, 3.43,-2.43, -1.56,-3.05

We will have data that is comma separated. The first three numbers are the current value of acceleration in the x, y, and z directions. The next three are the average for the user-specified amount prior in the array. The next three are the maximum values recorded. The final three are the minimum recorded.

Design
Necessary Formulas:
- Average – (sum of all items) / (total number of items)

Algorithm:
1. First off, we will need to get from the command line the size of the buffer the user would like to use. This code was provided for us. If the user enters an invalid amount, we will end the program and have them retry.
2. Then we need to fill each array up to the specified amount. We will do this right in the main function. We do not want to use the update buffer function when we are initially filling the array because the update buffer function will run through the array and move each item up a spot and put each new value at the end.
3. After we initially fill up the array, we will then continue to fill it, but this time using the update buffer function. This way we will keep the amount of the array we actually use to the user specified amount.
4. Each time we call the update buffer function, we will also call the average function and find the average of the values in the array.
5. We also will call the max_min function each time to find the maximum and minimum value currently in the array.
6. We finally print out a comma-separated list of these values in the format that I listed above.

Testing
We ran into a few small problems when writing this program.

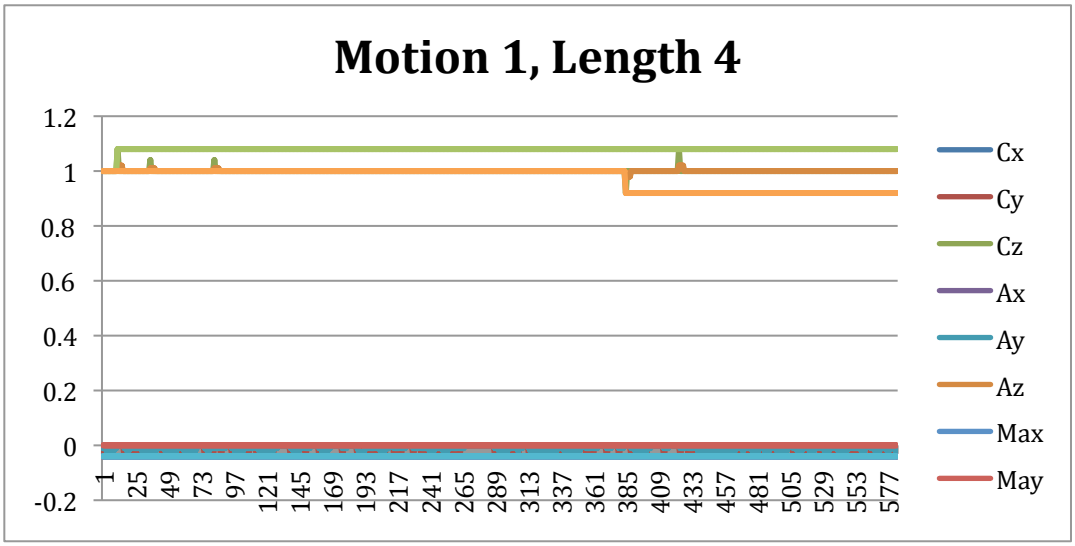
- One problem was a really stupid mistake. I forgot to put the & in when I was using the scanf statement. This was causing my program to crash. I thought that it was the Wiimote that was acting up, but then after a while of looking through what I had been working on, I finally figured it out.
- Another simple thing was that I somewhat misread the instructions. I originally wrote the max_min function to find the max and min values since the program

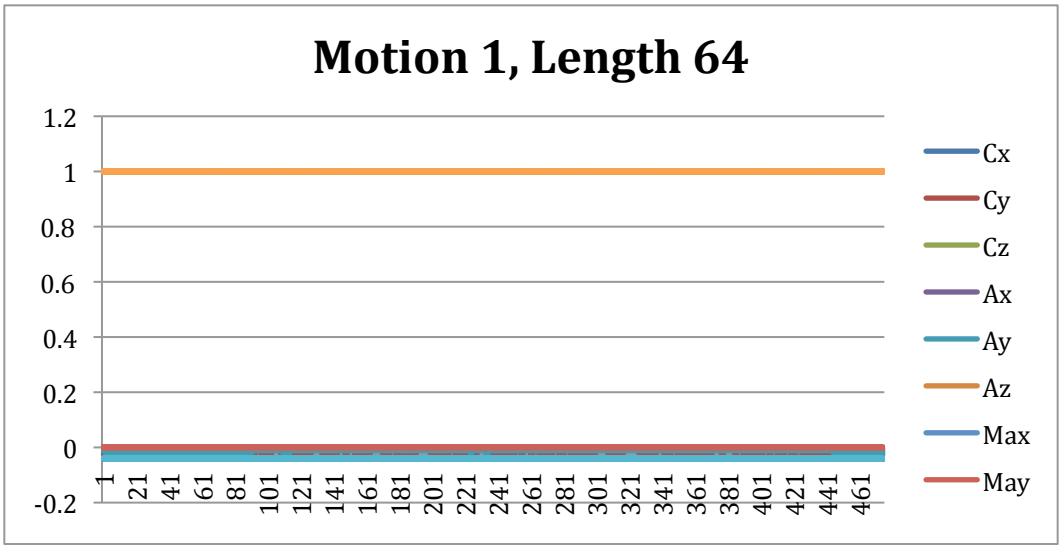
started. This is a useful value, but not the one that is asked for in the lab. So I just needed to change the program a little to find the max and min values that are currently in the array. This was a simple fix.

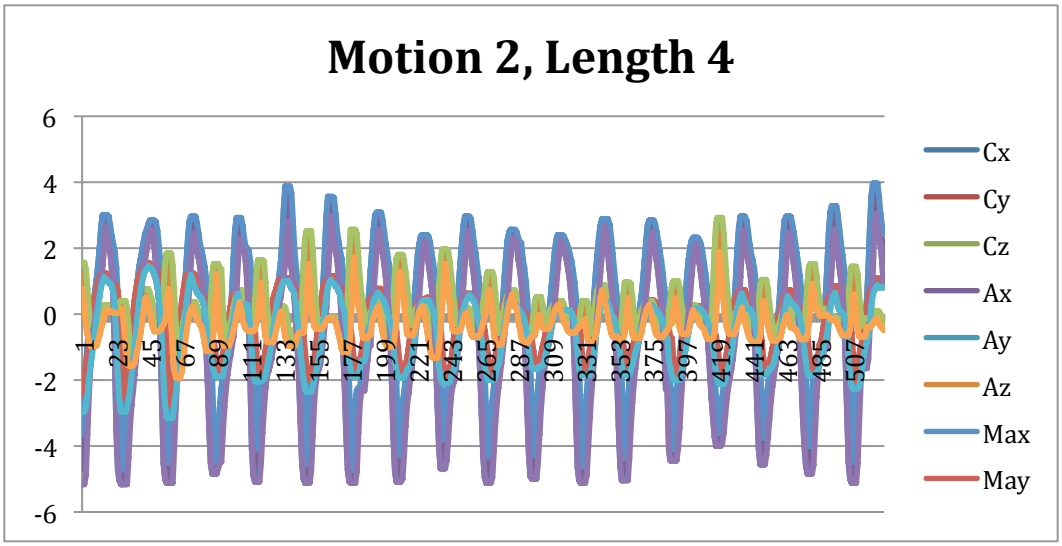
Comments
Lab Questions

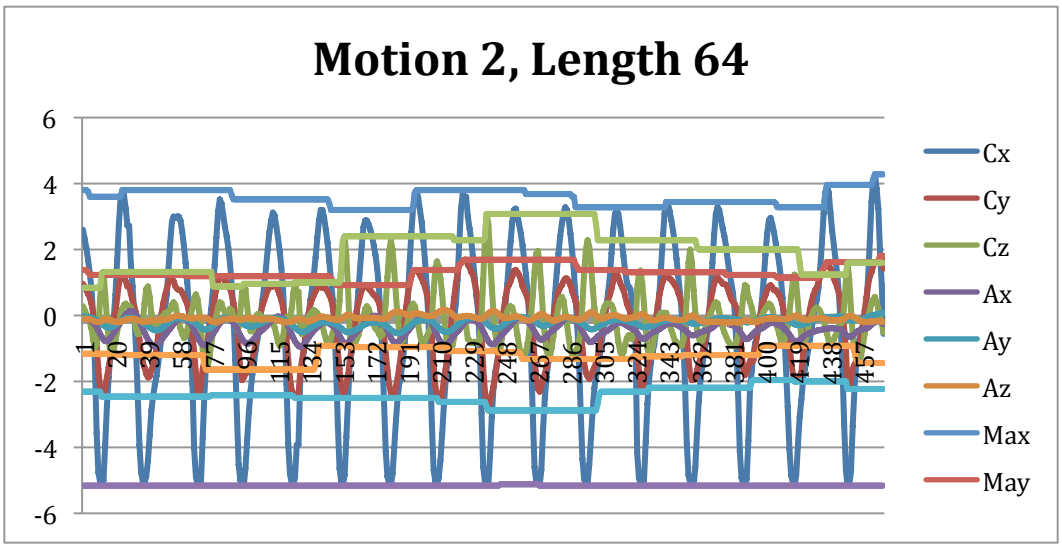
- When comparing the short and long window lengths, there are a few major differences. With the long window, it takes a while for the averages to change. This makes the graph a little smoother, and wider. With the short window, the changes come more rapidly. This makes the graph not as smooth and makes the frequency higher and the wavelength shorter (this is very prominent in motion 2). The reason the wavelength gets longer and the frequency gets lower when we go to a longer window is because there is more data. When you have 64 pieces of data, one really high value isn't going to make that big of a difference. Whereas when you only have 4 pieces of data in an array, one large value will make a much bigger difference.
- When comparing motion 2 and 3, it seems like it would be best to use a short window for the fast motion, and a long window for a slow motion. It seems to give us clearer averages and better-looking (easier to read) graphs. I am not sure if this is the best way, but I think that it is what I would do. It seems to make sense because when you have a slow motion, not much is going to change in a short time period, but with a fast motion, things will change quickly, meaning it would make sense to have a shorter graph.

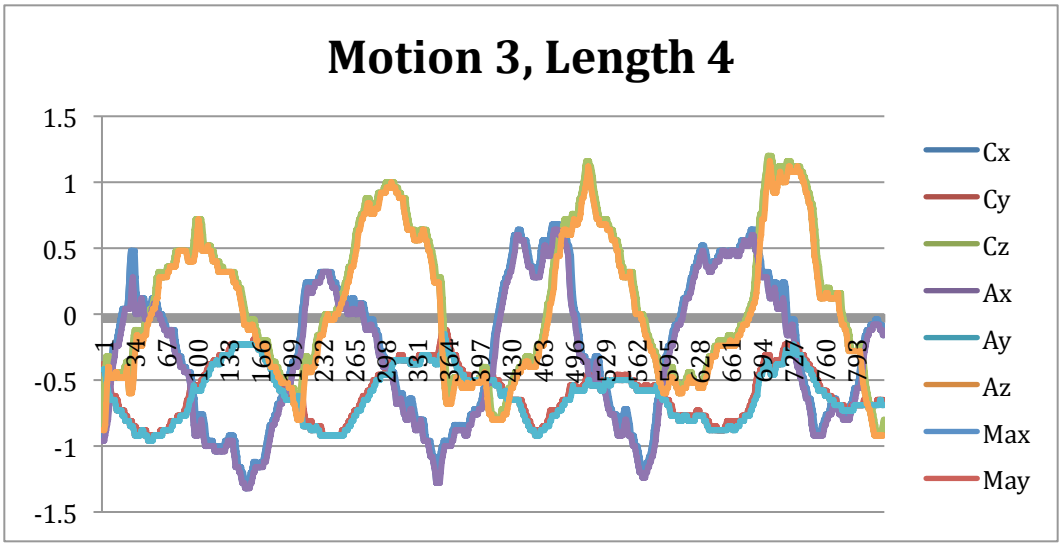
Other Comments
In general, I thought that this lab wasn't too difficult to figure out, once I understood what I was trying to do. Writing the code was pretty easy. The main problems were trying to figure out what the problem was asking us to do. This is all part of the top-down program design that we are learning about. I think it is a good thing that we are given a problem, and we are supposed to try and figure out what exactly the problem means, and then solve it. Figuring out what the problem means is a very helpful stage in solving the problem.

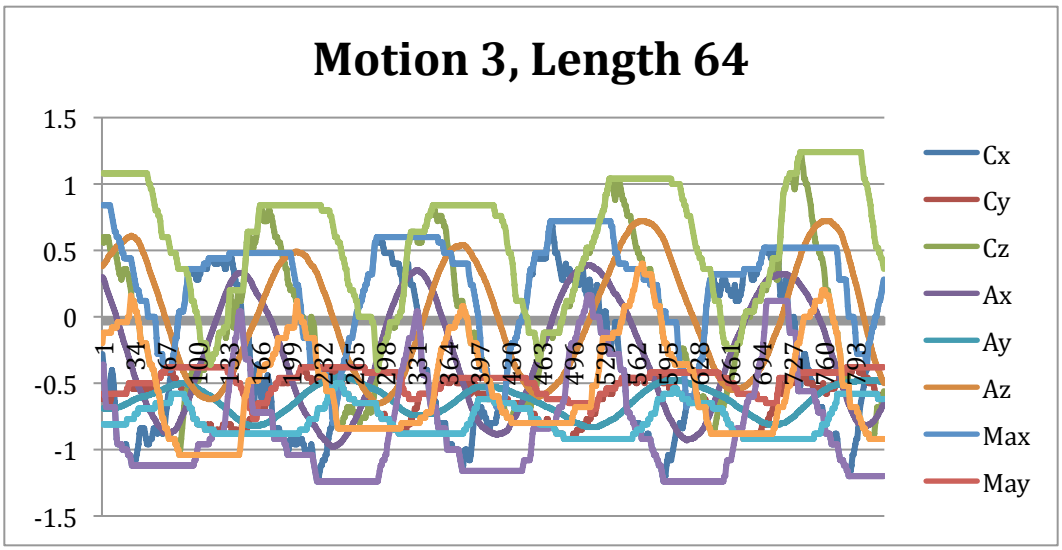



Motion 1, Length 64
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
1
21
41
61
81
101
121
141
161
181
201
221
241
261
281
301
321
341
361
381
401
421
441
461
Cx
Cy
Cz
Ax
Ay
Az
Max
May


Motion 2, Length 4
6
4
2
0
-2
-4
-6
1
23
45
67
89
111
133
155
177
199
221
243
265
287
309
331
353
375
397
419
441
463
485
507
Cx
Cy
Cz
Ax
Ay
Az
Max
May


Motion 2, Length 64
6
4
2
0
-2
-4
-6
1
20
39
58
77
96
115
134
153
172
191
210
229
248
267
286
305
324
343
362
381
400
419
438
457
Cx
Cy
Cz
Ax
Ay
Az
Max
May


Motion 3, Length 4
1.5
1
0.5
0
-0.5
-1
-1.5
1
34
67
100
133
166
199
232
265
298
331
364
397
430
463
496
529
562
595
628
661
694
727
760
793
Cx
Cy
Cz
Ax
Ay
Az
Max
May


Motion 3, Length 64
1.5
1
0.5
0
-0.5
-1
-1.5
1
34
67
100
133
166
199
232
265
298
331
364
397
430
463
496
529
562
595
628
661
694
727
760
793
Cx
Cy
Cz
Ax
Ay
Az
Max
May

Implementation

```
/* Lab 8 - Moving Average
 Brian Reber & Nathan Brinkman
 */

#include <stdio.h>


#define MAXPOINTS 10000

      // compute the average of the first num_items of buffer
      double avg(double buffer[], int num_items);

      //return the max and min of the first num_items of array
      void maxmin(double array[], int num_items, double* max, double* min);

      //shift length-1 elements of the buffer to the left and put the
      //new_item on the right.
      double updatebuffer(double buffer[], int length, double new_item);

int main(int argc, char* argv[]) {

      double x[MAXPOINTS], y[MAXPOINTS], z[MAXPOINTS];
      int lengthofavg = 0;
      int count = 0, t;
      float ax,ay,az;
      double max_x, min_x;
      double max_y, min_y;
      double max_z, min_z;


      int button_a, button_junk;

      //Read in the number from the command line
      if (argc>1)
        {
        sscanf(argv[1], "%d", &lengthofavg );
        printf("You entered a buffer length of %d\n", lengthofavg);
        }
      else {
        printf("Enter a length on the command line\n");
        return(-1);
        }
      if (lengthofavg < 1 || lengthofavg > MAXPOINTS)
        {
        printf("Invalid length\n");
        return(-1);
        }

      //Fill in the user specified number in the array
      //This is our initial time through so we need to populate the
      //array before we call the updatebuffer function
      for (count = 0; count < lengthofavg; count++)
      {
        scanf("%d,%f,%f,%f,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", &t, &ax, &ay, &az,

      &button_a,&button_junk,&button_junk,&button_junk,&button_junk,

      &button_junk,&button_junk,&button_junk,&button_junk,&button_junk,&button_junk);

        x[count] = ax;
        y[count] = ay;
        z[count] = az;
      }

      //We then run through until the a button has been pressed
      do
      {
        scanf("%d,%f,%f,%f,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d,%d", &t, &ax, &ay, &az,

      &button_a,&button_junk,&button_junk,&button_junk,&button_junk,

      &button_junk,&button_junk,&button_junk,&button_junk,&button_junk,&button_junk);
        //We update the array
```

```
        updatebuffer(x,lengthofavg,ax);
        updatebuffer(y,lengthofavg,ay);
        updatebuffer(z,lengthofavg,az);

        //We then compute the max and min
        maxmin(x, lengthofavg, &max_x, &min_x);
        maxmin(y, lengthofavg, &max_y, &min_y);
        maxmin(z, lengthofavg, &max_z, &min_z);

        //Finally printing it out
        printf("%4.2lf, %4.2lf, %4.2lf, %4.2lf, %4.2lf, %4.2lf, %4.2lf, %4.2lf, %4.2lf,
%4.2lf, %4.2lf, %4.2lf\n",
                ax,ay,az,avg(x, lengthofavg),avg(y, lengthofavg), avg(z, lengthofavg),
max_x, max_y, max_z, min_x, min_y, min_z);

      } while (!button_a);

      return 0;
}

/*
 param: buffer[] - this is the current array we want to find the average of
 param: num_items - this is the amount of items we need to scan through
 */
double avg(double buffer[], int num_items){
      int i;
      double avg = 0.0;
      for (i = 0; i < num_items; i++)
      {
        avg += buffer[i];
      }

      return avg/num_items;
}

/*
 param: array[] - this is the current array we want to find the max and min nums in of
 param: num_items - this is the amount of items we need to scan through
 param: double* max - this is the output param - we will use this to return the max
        value in the array.
 param: double* min - this is the output param - we will use this to return the min
        value in the array.
 */
void maxmin(double array[], int num_items, double* max, double* min){
      int i;
      double max_local = -50, min_local = 50;

      for (i = 0; i < num_items; i++)
      {
        if (array[i] > max_local) max_local = array[i];
        if (array[i] < min_local) min_local = array[i];
      }

      *max = max_local;
      *min = min_local;
}

/*
 param: buffer[] - this is the current array we want to update
 param: length - this is the amount of items we need to scan through
 param: new_item - this is the item we want to put at the end of the buffer array
 */
double updatebuffer(double buffer[], int length, double new_item){
      int i;
      for (i = 0; i < length-1; i++)
      {
        buffer[i] = buffer[i+1];
      }
      buffer[length-1] = new_item;
}
```