Digital Signal Processing

Lab 5 DSP Number Systems

Introduction

Lab Objectives

- Explore how several data types commonly used in DSP (byte, signed integer, long integer, float) affect memory usage and computation speed..
- Explore the effects of finite precision math on calculations.
- Measure the execution speed for addition and multiplication and compare how the speed changes using the various number data types.

Lab 5 Tips

- Download the procedure and open in Word
- Carefully read each section before working on them
- Copy code from text boxes. Use CTRL_A CTRL_C to get all of it!
- Do not use the MATLAB capture program to collect data.
  - Use the serial monitor and copy paste to excel

Lab 5 Tips

- Save time – Use the Excel tables are available for you in myCourses
- The lab report is to be in IEEE format

Lab Summary – Part 1

- Investigate what happens when one adds two numbers together of different data types
- Modify the code to loop through the values in the av1 and bv1 inputs and add them

```
// change declaration to byte, int, long, float
DATATYPE avl[16] = {100, 200, 20000, 100, 100, 100, 255, 255, 25500, 20000, 20000, 20000, 20000, 1000000000, 2000000000, 2000000000};
DATATYPE bvl[16] = {100, 200, 20000, 155, 156, 157, 100, 10,  25500, 12767, 12768, 12769, 12770, 1000000000, 147483649,  147483649};
```

- Run the code using different data types by defining the variable DATATYPE

```
#define DATATYPE long // change declaration to byte, int, long, float
```

- Think about what happens when adding various data types and the values you are adding

Lab Summary – Part 2

- Investigate Round off Error with floating point numbers
- Add 2 random numbers then subtract the same 2 numbers multiple times

```
for (int i = 1; i < NUM_CALC; i++)
{
  A = random_float() * scale_factor;
  B = random_float() * scale_factor;
  xv = xv+A;    // add A to x
  xv = xv+B;    // add B to x
  xv = xv-A;    // subtract A from x
  xv = xv-B;    // subtract B from x
} // for
```

```
scale_factor = 1.0; // start with 1.0
```

- Find the error from the desired value
- Scale the numbers differently and observe what happens
- Think about the reason for error. Why does the error change with different scale values

Lab Summary – Part 3

- Unexpected errors with floats
- Investigate what happens when one increments floating point numbers and long datatype numbers
- Think about:
  - What numbers can be represented exactly with floats?
  - What numbers can be represented exactly with longs?

Lab Summary – Part 4

- Finite precision effects with floats
  - Experiment with SINE wave and increasing angles
  - How can one improve the situation
  - Think about what you know about a sinewave and when it repeats. With what angle terms?

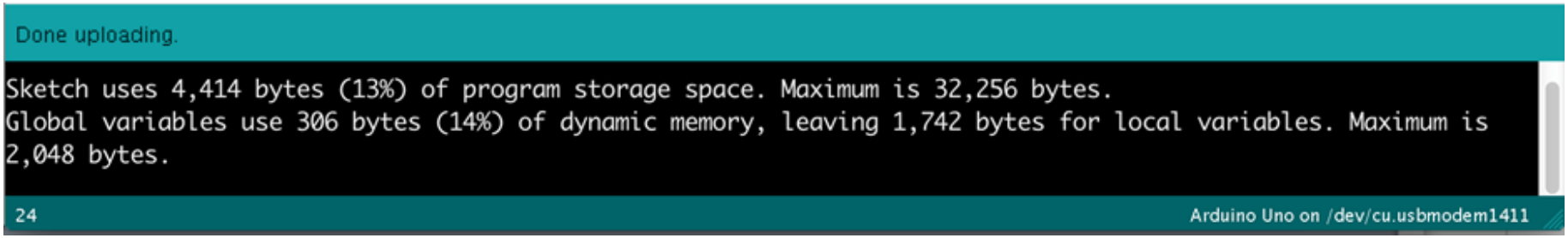
Lab Summary – Part 5

- Measure execution times using different data types
- Change the data types of ‘result’ and ‘xv[]’

- Note: Do not use the CAST function. Just change the data types of ‘result’ and ‘xv[]’

Lab Summary – Part 6

- Measure execution times while performing convolution
- Record the free memory used for data
- Record the execution times for different data types



Arduino Data Types

- BYTE – 8 bit unsigned integer

0 to 255

BYTE -- 8 Bits -- Unsigned	
Decimal	**Bit Pattern**
255	11111111
254	11111110
253	11111101
…	…
2	00000010
1	00000001
0	00000000

Arduino Data Types

- INT – 16 bit signed integer - 2's Complement

-32768 to 32767

INT -- 16 Bits 2's Complement	
Decimal	**Bit Pattern**
32767	0111111111111111
32766	0111111111111110
32765	0111111111111101
…	…
2	0000000000000010
1	0000000000000001
0	0000000000000000
-1	1111111111111111
-2	1111111111111110
-3	1111111111111101
…	…
-32766	1000000000000010
-32767	1000000000000001

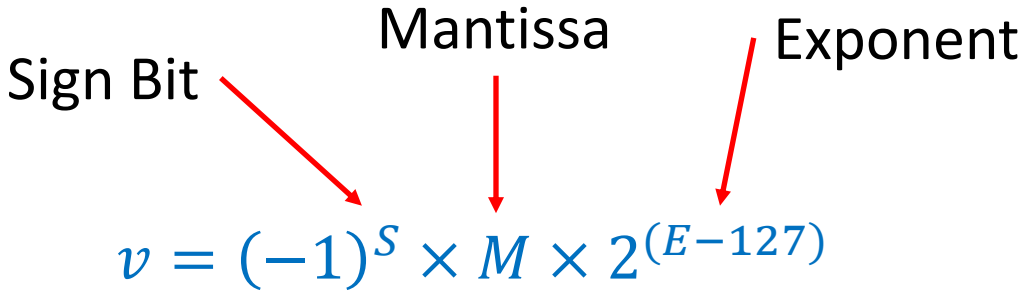
Arduino Data Types

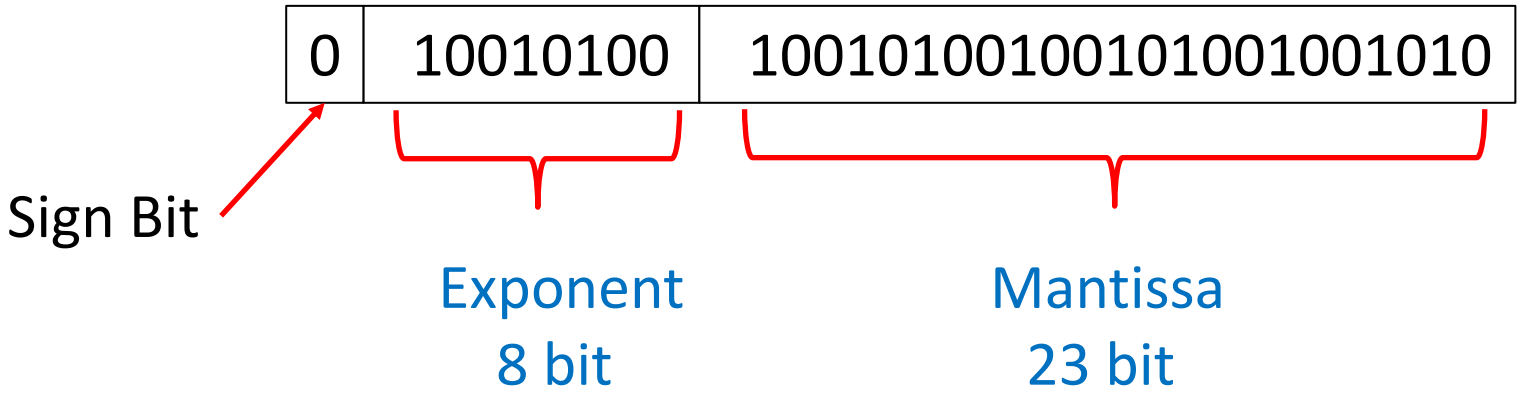
- LONG Integer - 32 bits 2's complement

LONG -- 32 Bits 2's Complement		
Decimal	**Bit Pattern**	**HEX**
2147483647	01111111111111111111111111111111	0x7FFFFFFF
2147483646	01111111111111111111111111111110	0x7FFFFFFE
2147483645	01111111111111111111111111111101	0x7FFFFFFD
…	…	…
2	00000000000000000000000000000010	0x00000002
1	00000000000000000000000000000001	0x00000001
0	00000000000000000000000000000000	0x00000000
-1	11111111111111111111111111111111	0xFFFFFFFF
-2	11111111111111111111111111111110	0xFFFFFFFE
-3	11111111111111111111111111111101	0xFFFFFFFD
…	…	…
-2147483646	10000000000000000000000000000010	0x80000002
-2147483647	10000000000000000000000000000001	0x80000001
-2147483648	10000000000000000000000000000000	0x80000000

Single Precision Floating Point

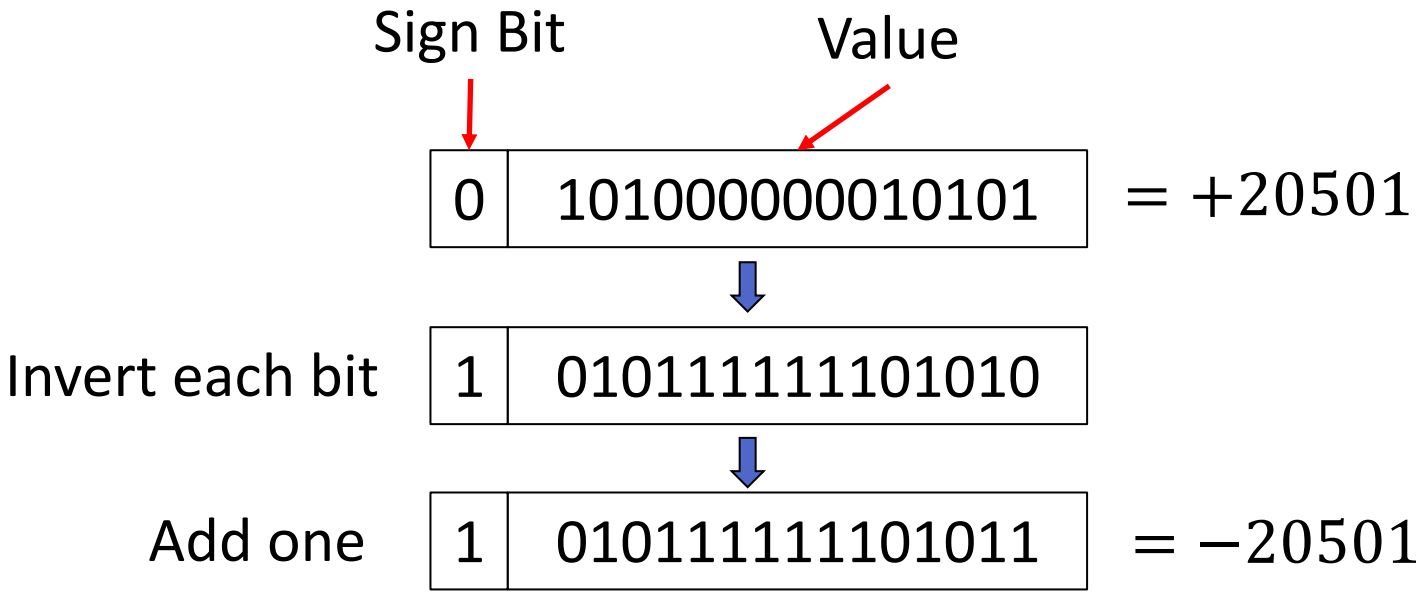
- The value represented by the number is





Finding the 2's Complement Positive Number Example

- Start with the number and "complement" each bit
- Add one to the number



2's Complement Math

- Adding 2's complement numbers is done simply with binary addition
  - 0+0 = 0, 0+1 = 1, 1+1 = 0 with a carry
- Add 3,489 + 23,732

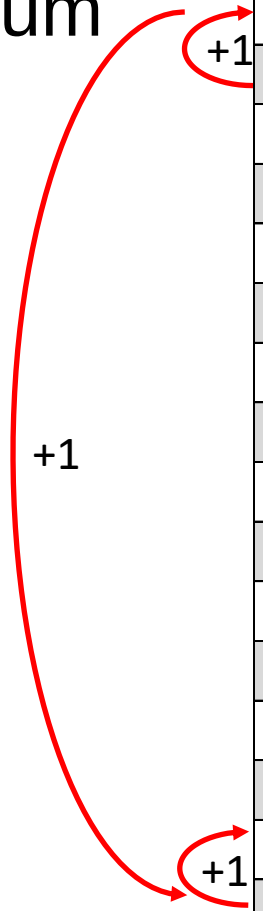
0	000110110100001	$= +3{,}489$
	+	
0	101110010110100	$= +23{,}732$
	=	
0	0110101001010101	$= +27{,}221$

Wrapping of Integer Values

- If I add two numbers and the sum is larger than the maximum value, the value will “wrap” around.

- Example 6+3 = -7

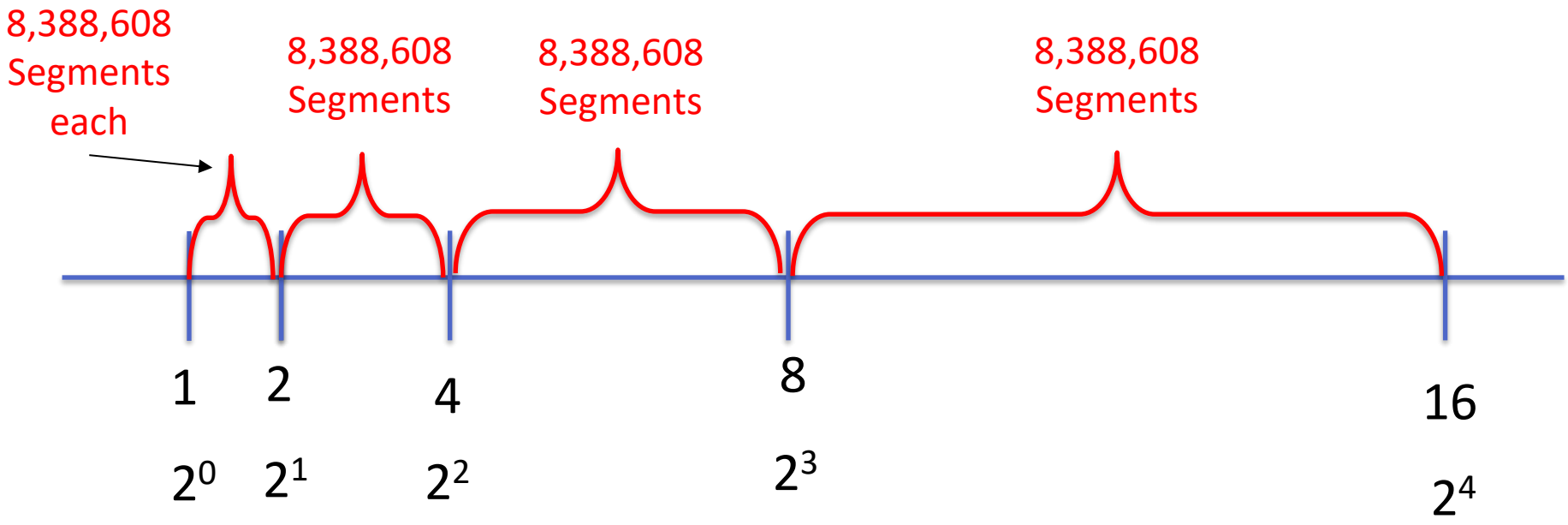
  6+1 = 7
  7+1 = -8
  -8+1 = -7



Two's Complement -- 4-Bits	
Decimal	Bit Pattern
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

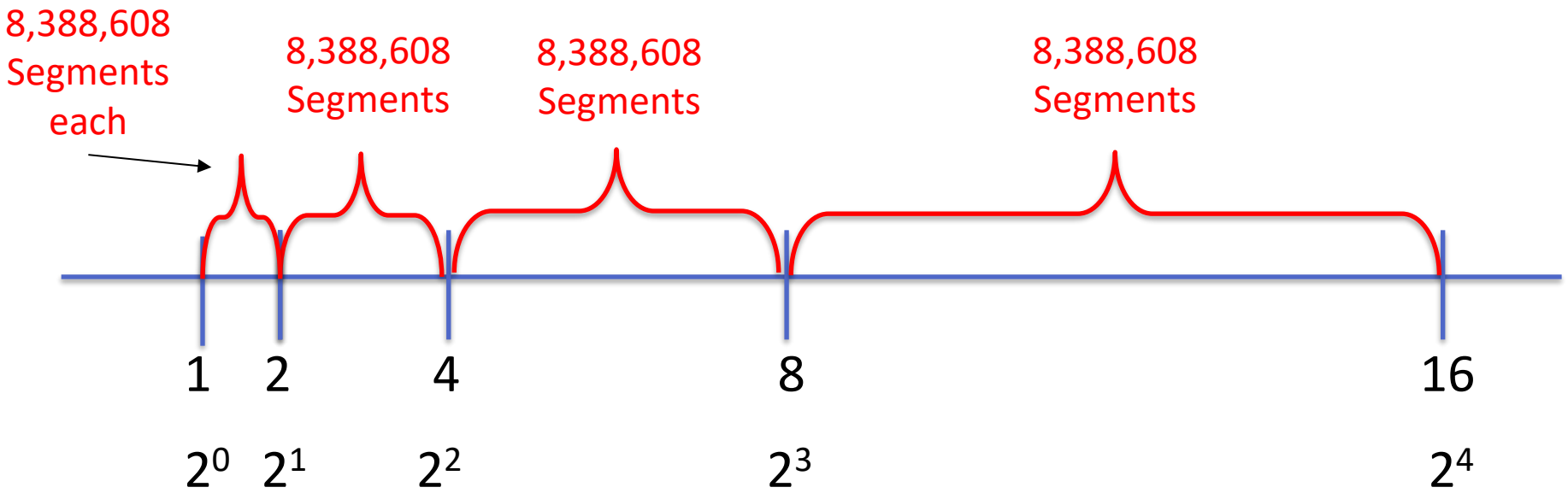
Floating Point Interval Between Numbers

- The mantissa has $2^{23} = 8{,}388{,}608$ unique values
- Each exponential interval is broken into 8,388,608 segments



Floating Point Interval Between Numbers

- The size of each segment becomes larger as the exponent increases.
- The gap between values increases as the exponent increases



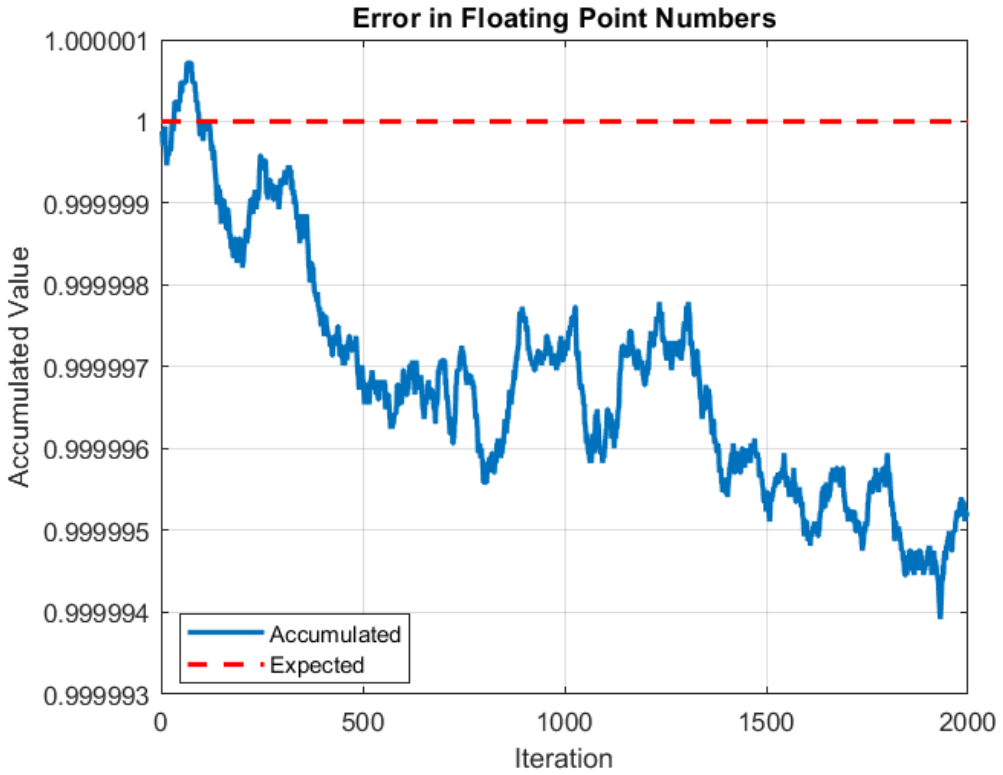
Error Accumulation Example

- Start with the value of 1
- Add a random value A
- Add a random value B
- Subtract A
- Subtract B
- Repeat 2000 times

```
x = single(1);

for i = 1:2000
  a = single( rand );
  b = single( rand );

 %  Add the two random values
  x = x + a;
  x = x + b;

  %  Subtract the two random values
  x = x - a;
  x = x - b;

end
```

Error Accumulation Example

- The value should always be 1
- There is error in each addition and subtraction
- And that error may accumulate depending on the sign



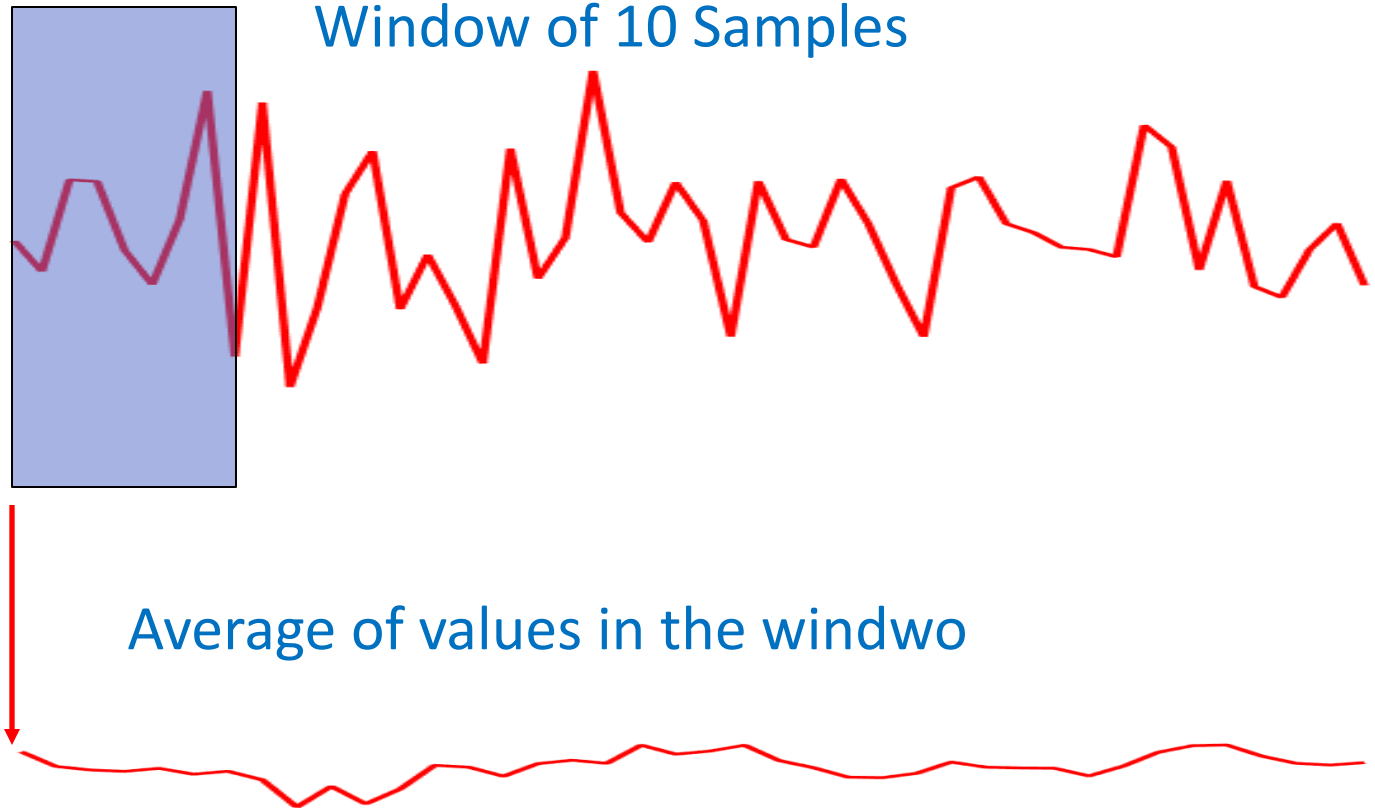
Execution Times for Different Data Types

- Some operations take longer with data type
- Explore the difference of multiplication between float and long
- Measure Execution Time of a convolution sum using addition and multiplication

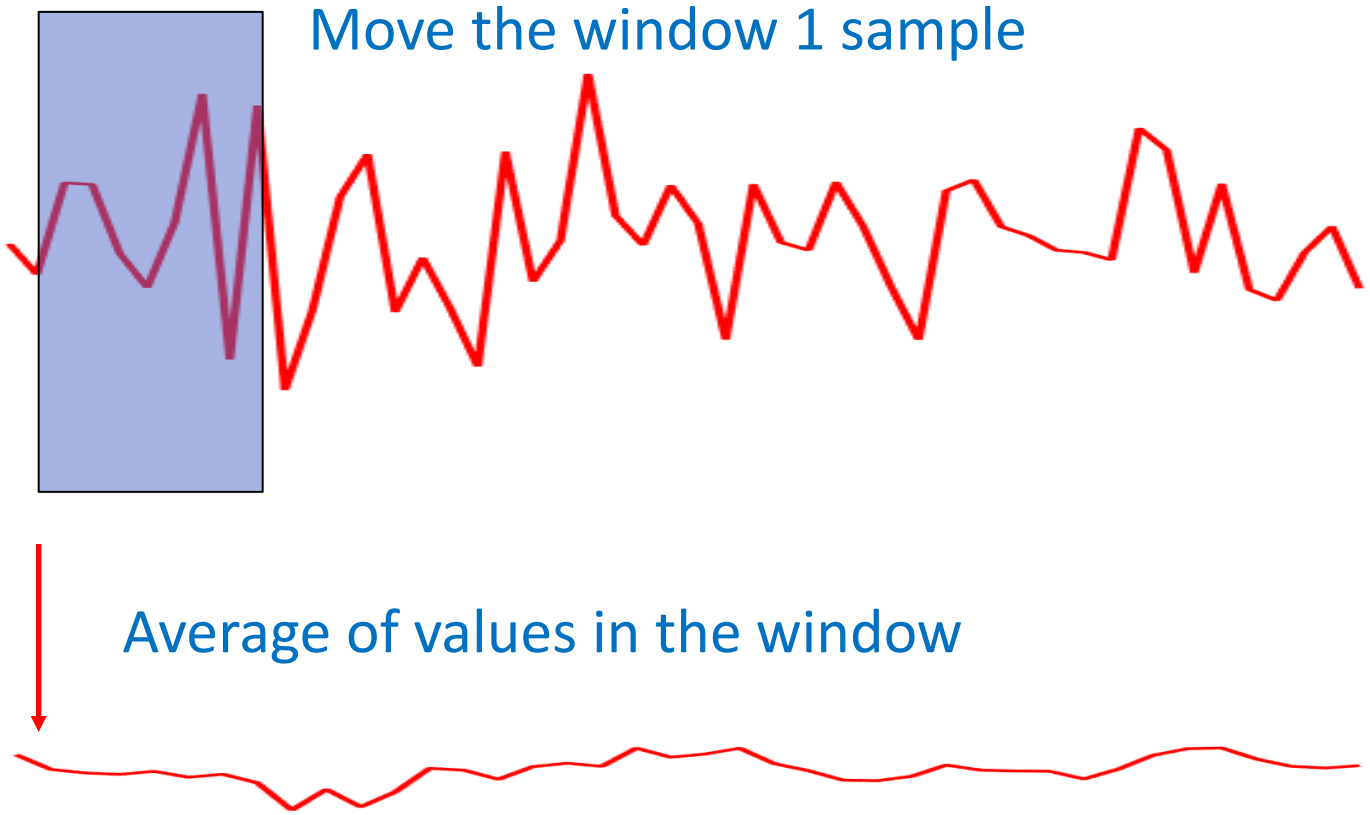
Moving Average Filter

- A moving average filter is a kind of lowpass filter
- Take n sequential samples and average them for the output sample
- Move to the right by one sample and average those n-samples

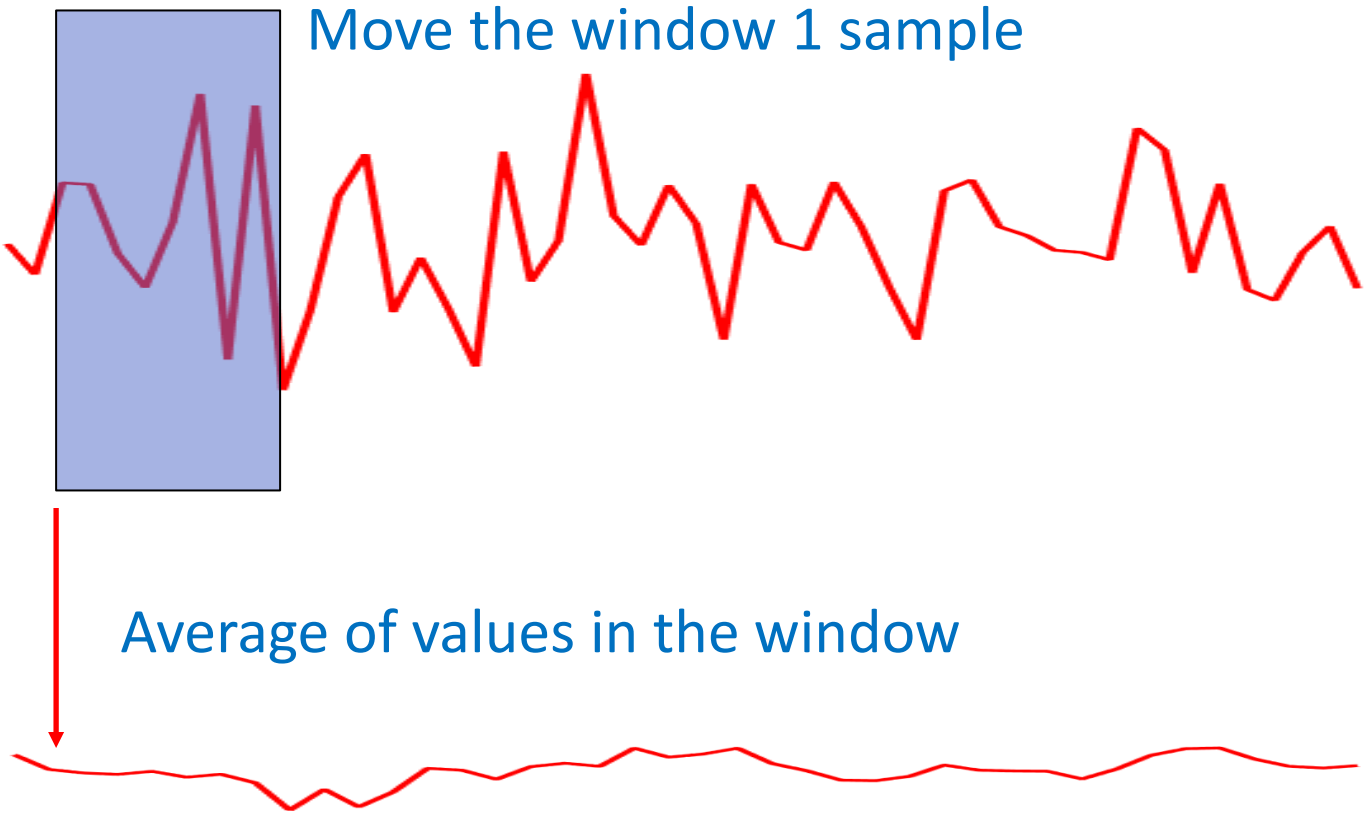
Moving Average Filter



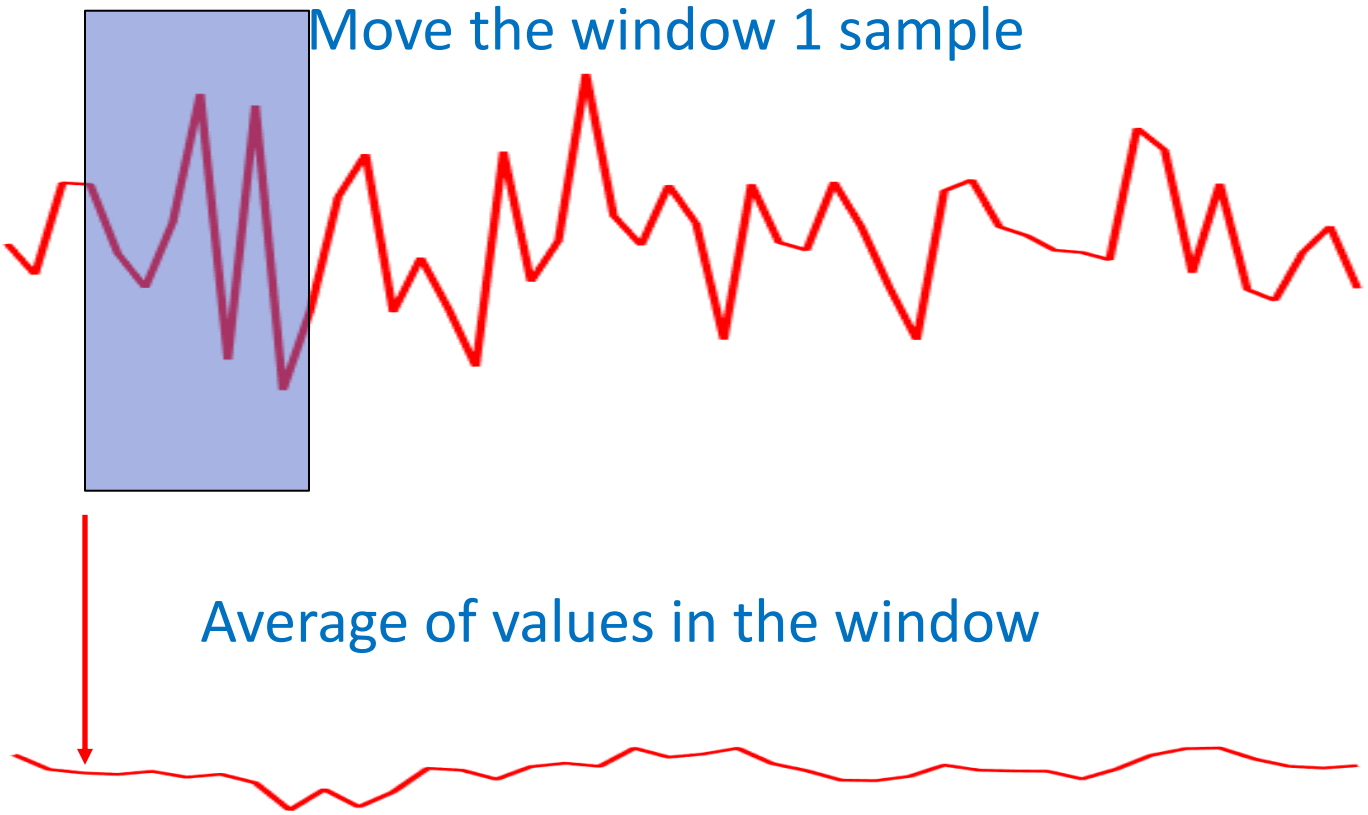
Moving Average Filter



Moving Average Filter

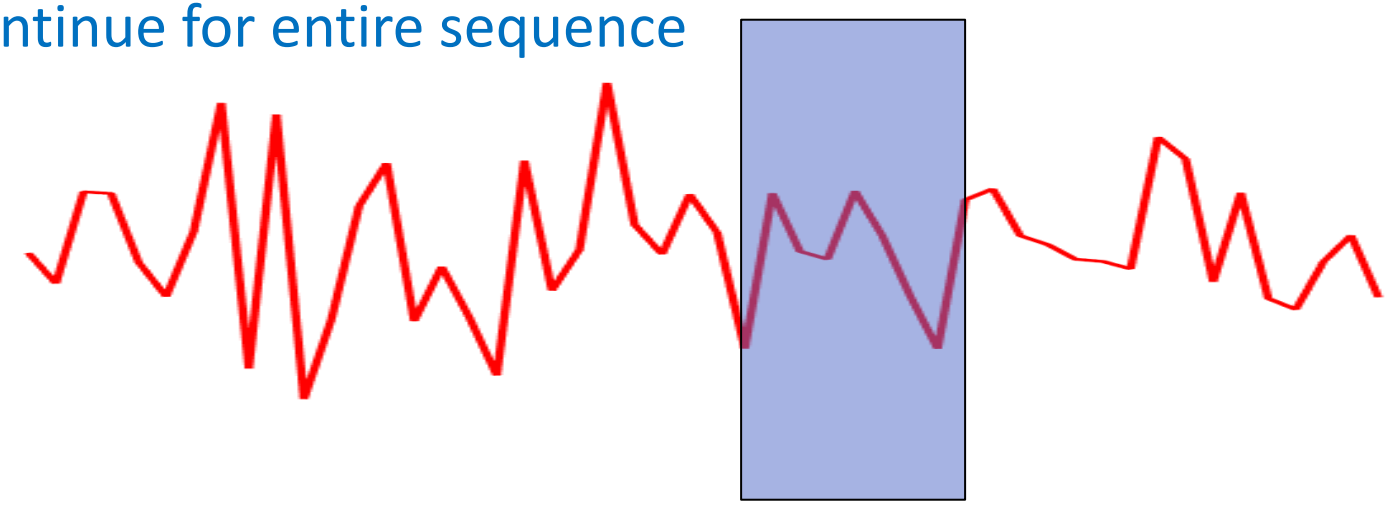


Moving Average Filter

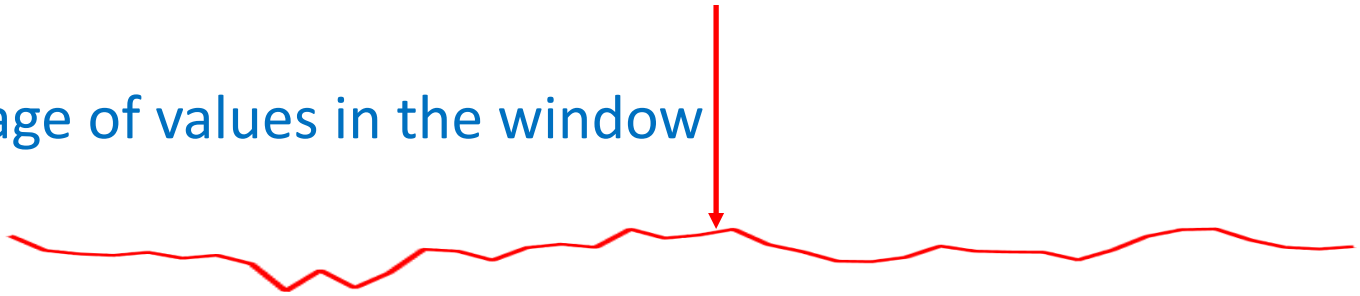


Moving Average Filter

Continue for entire sequence

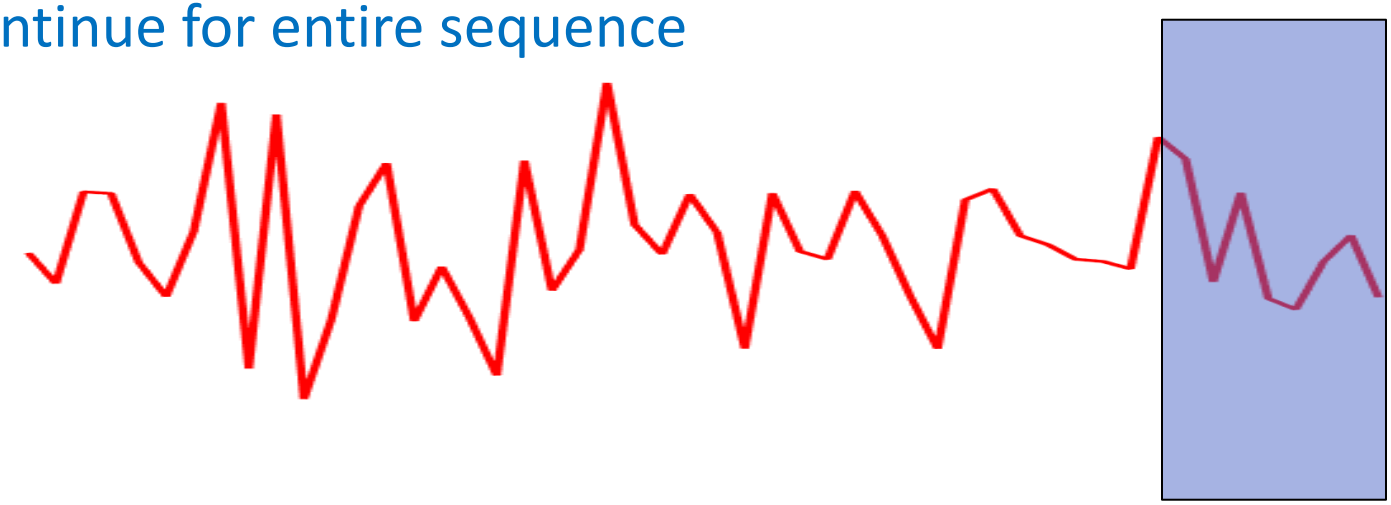


Average of values in the window



Moving Average Filter

Continue for entire sequence



Average of values in the window

C Code for MAV Filter

- Initialize the impulse response

```
Serial.println("\nh impulse response\nn\th[n]");

 for (int i = 0; i < IMP_RESP_LEN; i++)
 {
   h[i] = 1.0/IMP_RESP_LEN;
   Serial.print(i); Serial.print('\t');
   Serial.println(h[i],4);
 }
```

Loop over the length of the impulse response

Set each value in the impulse response to 1/length

Example IMP_RESP_LEN =5

$$h = \left[\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right]$$

C code for the MAV Filter

- Perform the convolution

```
// perform sum of products
 // start convolution only where data is valid
 // first IMP_RESP_LEN-1 datapoints are not valid

 startTime = micros();
 for (int k = IMP_RESP_LEN-1; k < DATA_LEN; k++){
   for (int i = 0; i < IMP_RESP_LEN; i++){
     yv[k] = yv[k]+h[i]*xv[k-i];
   }
 }
```

????? What's going on here ????

C code for the MAV Filter

```
for (int k = IMP_RESP_LEN-1; k < DATA_LEN; k++)
{
    for (int i = 0; i < IMP_RESP_LEN; i++)
          {
              yv[k] = yv[k]+h[i]*xv[k-i];
          }
 }
```

Inner Loop

Outer Loop

Impulse Response h[n]