



Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National
Laboratory
dbiersach@bnl.gov



```

1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Windows.Forms;
9
10 namespace SimpleEvents
11 {
12     public partial class Form1 : Form
13     {
14         Person person = new Person();
15
16         public Form1()
17         {
18             InitializeComponent();
19             person.FirstName = "Christian";
20             person.LastName = "Pano";
21         }
22
23         private void button1_Click(object sender, EventArgs e)
24         {
25             person.MainColor = textBox1.Text;
26         }
27     }
28 }
  
```

Session 18
Computational Biology,
Earth Science

Session Goals

- Gain an appreciation for a **suffix sort** to find the **longest repeated substring** (LRSS) in a sequence
 - Consider the use of this algorithm in the realm of **bioinformatics** and genetic sequence alignment
 - Consider interesting research questions that stem from using the algorithm to **explore DNA**
- Assess the value of considering in DNA, not just the single longest repeated substring, but also the most (and least) *frequently* repeated substrings of a given length

What is the Longest Repeated Substring?

input string

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	a	c	a	a	g	t	t	t	a	c	a	a	g	c

Step 1 - Form the Suffixes array

suffixes

0	a	a	c	a	a	g	t	t	t	a	c	a	a	g	c
1	a	c	a	a	g	t	t	t	a	c	a	a	g	c	
2	c	a	a	g	t	t	t	a	c	a	a	g	c		
3	a	a	g	t	t	t	a	c	a	a	g	c			
4	a	g	t	t	t	a	c	a	a	g	c				
5	g	t	t	t	a	c	a	a	g	c					
6	t	t	t	a	c	a	a	g	c						
7	t	t	a	c	a	a	g	c							
8	t	a	c	a	a	g	c								
9	a	c	a	a	g	c									
10	c	a	a	g	c										
11	a	a	g	c											
12	a	g	c												
13	g	c													
14	c														

Step 2 – Sort the Suffixes array

sorted suffixes

0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

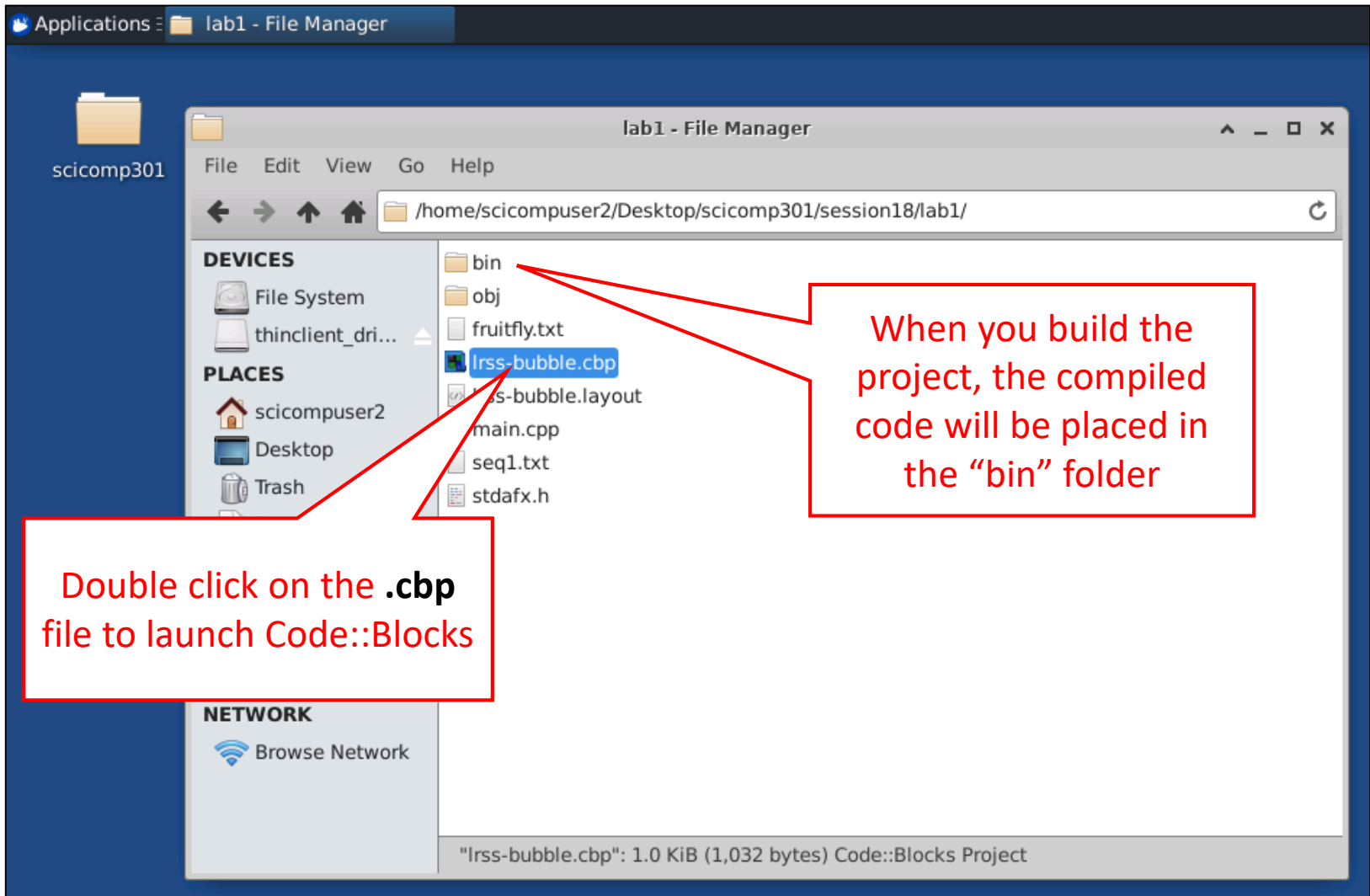
sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a a g c
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

Step 3 – Scan the Suffixes array

sorted suffixes	
0	a a c a a g t t t a c a a g c
11	a a g c
3	a a g t t t a c a a g c
9	a c a a g c
1	a c a a g t t t a c a a g c
12	a g c
4	a g t t t a c a
14	c
10	c a a g c
2	c a a g t t t a c a a g c
13	g c
5	g t t t a c a a g c
8	t a c a a g c
7	t t a c a a g c
6	t t t a c a a g c

longest repeated substring	
1	9
a a c a a g t t t a c a a g c	

Open Lab 1 – Longest Repeated Substring



View Lab 1 – Longest Repeated Substring

```
main.cpp [x]
95         longest = candidate;
96     }
97     return longest;
98 }
99
100
101 int main(int argc, char *argv[])
102 {
103     OpenDataFile(argc, argv);
104
105     LoadSequence();
106
107     boost::timer timer;
108
109     string longest = lrss(*seq);
110
111     cout << "The longest repeated substring in "
112          << "\"" << filename << "\" is: "
113          << longest << endl << endl;
114
115     cout.imbue(std::locale(""));
116     cout << "Total run time: "
117          << timer.elapsed() << "s" << endl;
118
119     CloseDataFile();
120
121     return 0;
122 }
```

Lab 1

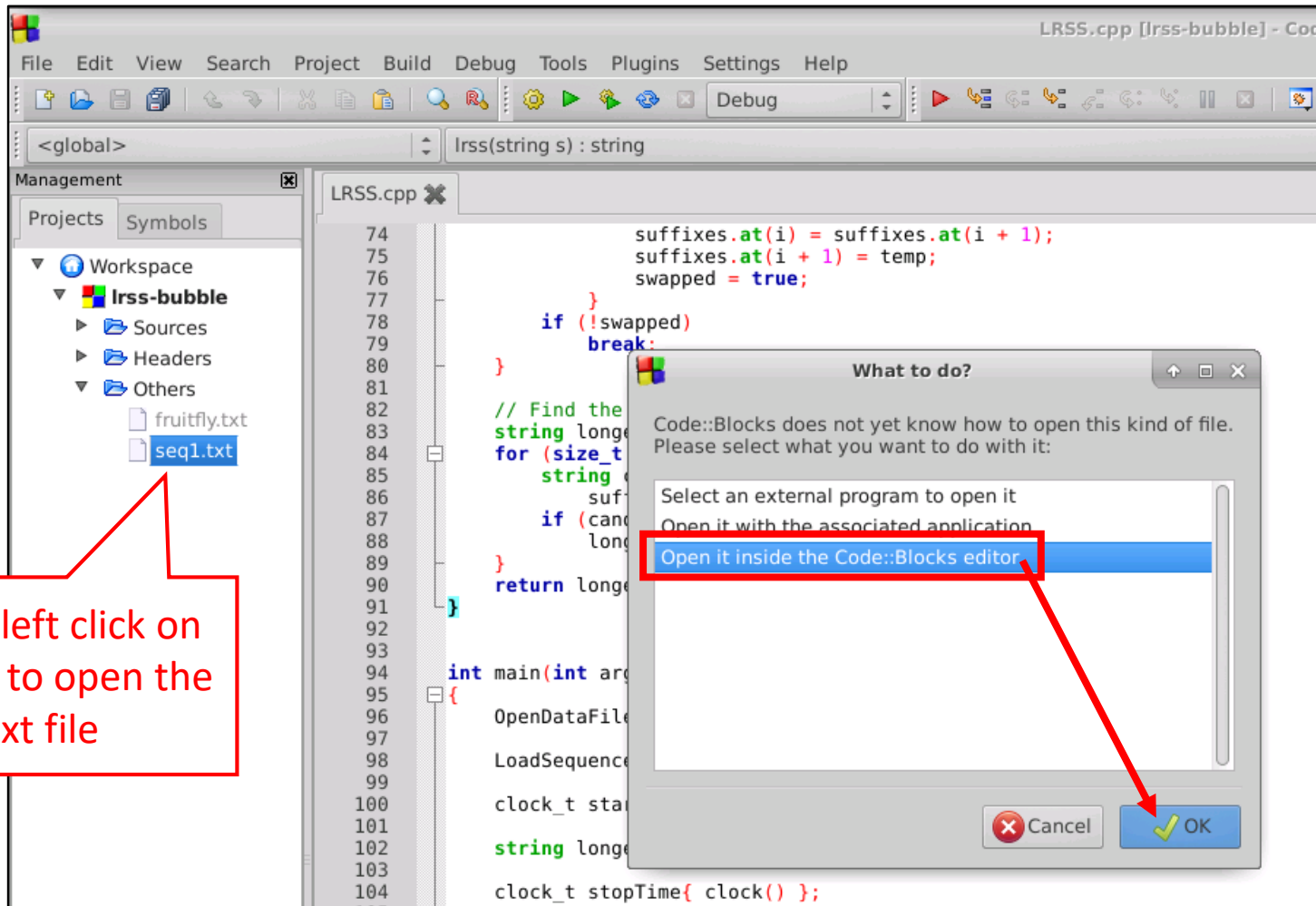
Longest Repeated Substring

```
string lrss(string s)
{
    // Create the suffix array
    vector<string> suffixes(s.size());
    for (size_t i{}; i < suffixes.size(); ++i)
        suffixes.at(i) = s.substr(i, s.size() - i);

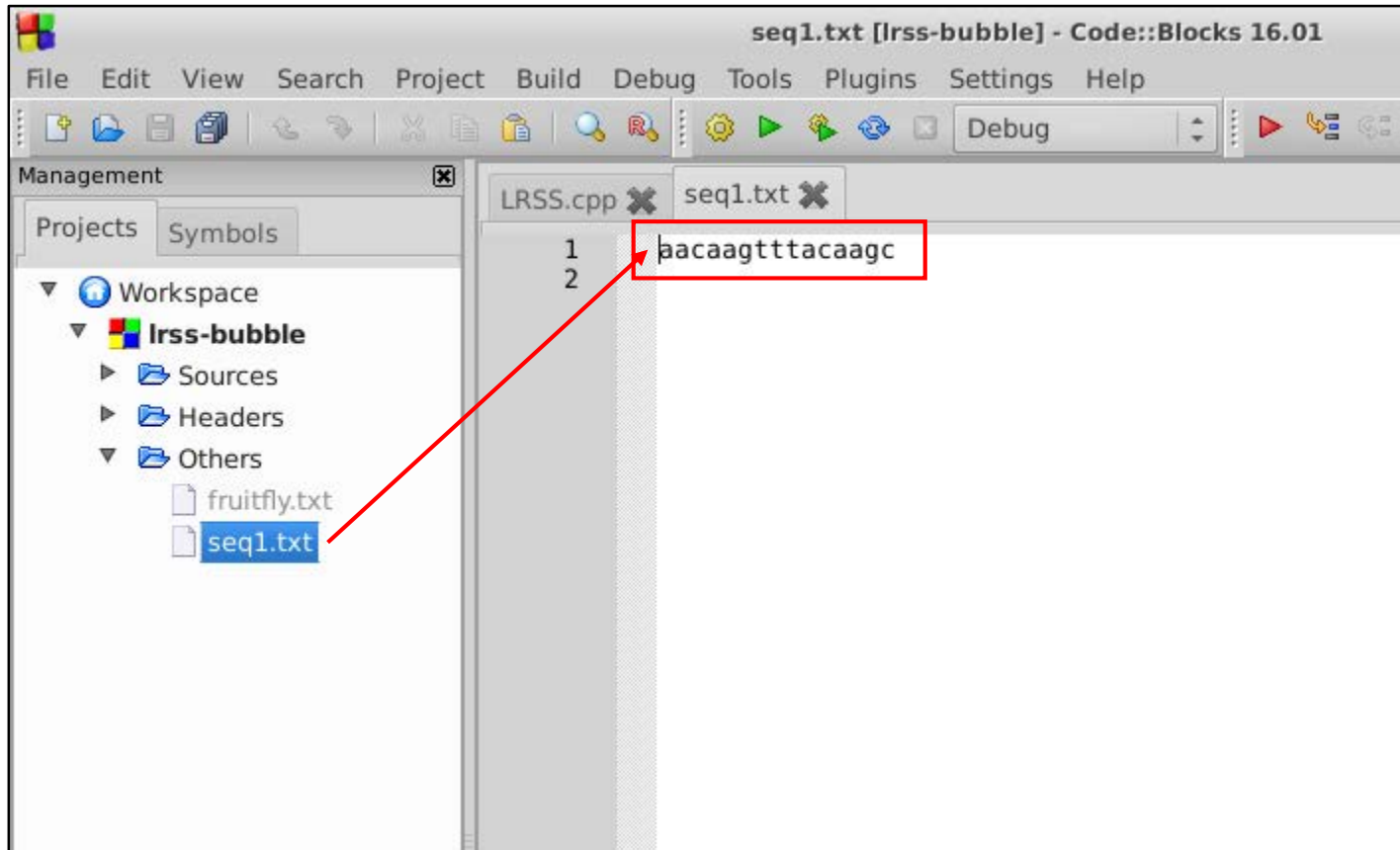
    // Bubble sort the suffix array
    while (true)
    {
        bool swapped = false;
        for (size_t i{}; i < suffixes.size() - 1; ++i)
            if (suffixes.at(i) > suffixes.at(i + 1))
            {
                string temp = suffixes.at(i);
                suffixes.at(i) = suffixes.at(i + 1);
                suffixes.at(i + 1) = temp;
                swapped = true;
            }
        if (!swapped)
            break;
    }

    // Find the longest repeated substring (lrss)
    string longest{};
    for (size_t i{}; i < suffixes.size() - 1; ++i)
    {
        string candidate = match(suffixes.at(i),
                                   suffixes.at(i + 1));
        if (candidate.size() > longest.size())
            longest = candidate;
    }
    return longest;
}
```

View Lab 1 – Longest Repeated Substring

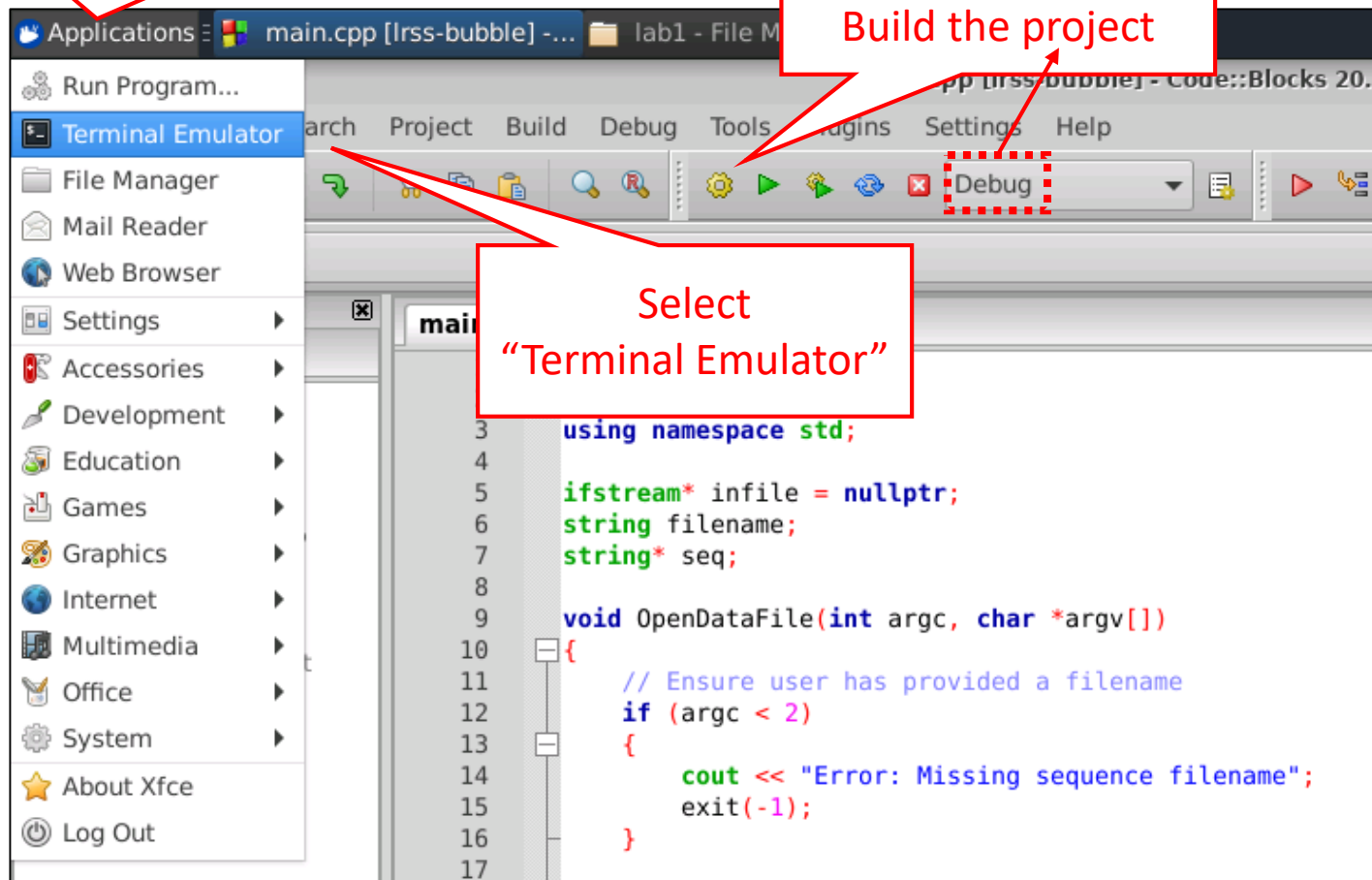


View Lab 1 – Longest Repeated Substring



Build Lab 1 – Longest Repeated Substring

Click on “Applications” button



Build Lab 1 – Longest Repeated Substring

scicompuser#/~/

Desktop/scicomp301/
session18/lab1/

seq1.txt

fruitfly.txt

bin/

Debug/

Release/

lrss-bubble

lrrs-bubble

In Linux only **folders** end with a forward slash /

A **parent** folder can be specified using the .. wildcard

The program **lrrs-bubble** is in the Debug/ folder

The **current** folder can be specified using the . wildcard

app name *data file path*
./lrrs-bubble ../../seq1.txt

Run Lab 1 – Longest Repeated Substring



The image shows a terminal window with the following content:

```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab1/bin/Debug
File Edit View Terminal Tabs Help
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab1/bin/Debug/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$ ./lr
ss-bubble ../../seq1.txt
The longest repeated substring in "../../seq1.txt" is: acaag
Total run time: 0.000226s
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$
```

A yellow highlight is under the output line: "The longest repeated substring in "../../seq1.txt" is: acaag".

A red box highlights the commands:

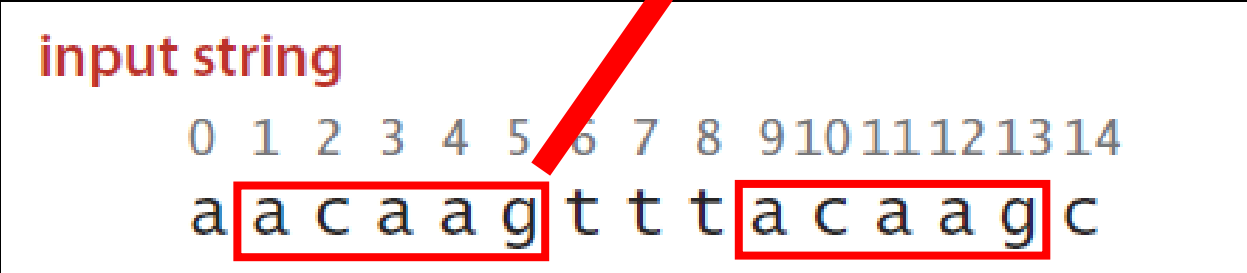
```
cd Desktop/scicomp301/session18/lab1/bin/Debug
./lrss-bubble ../../seq1.txt
```

A blue callout box contains the text:

Use **TAB** to autocomplete folder & file names
Press **ENTER** after typing each line

Check Lab 1 – Longest Repeated Substring

```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab1/bin/Debug
File Edit View Terminal Tabs Help
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab1/bin/Debug/
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$ ./lr
ss-bubble ../../seq1.txt
The longest repeated substring in "../../seq1.txt" is: acaag
Total run time: 0.000226s
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$
```

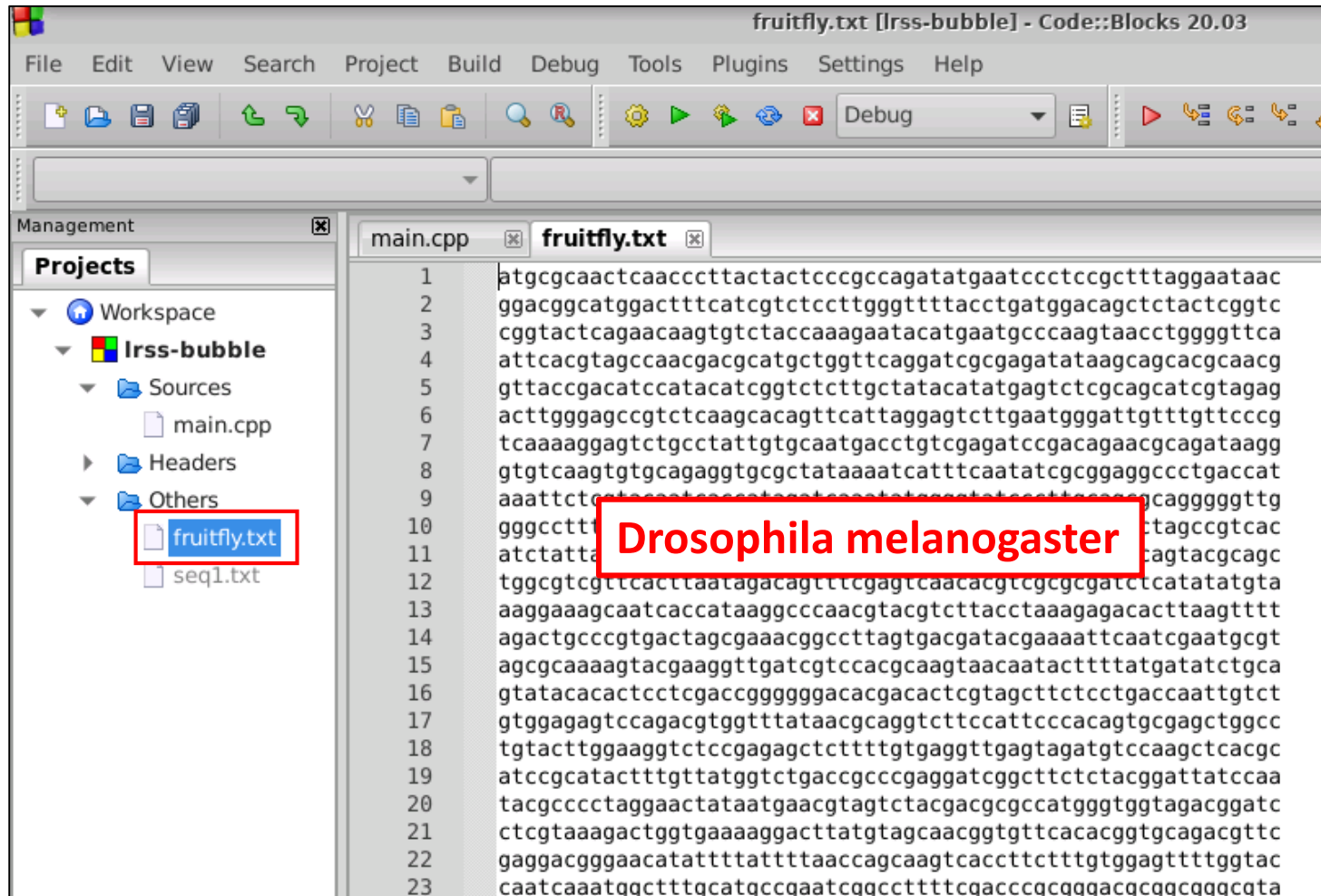


input string

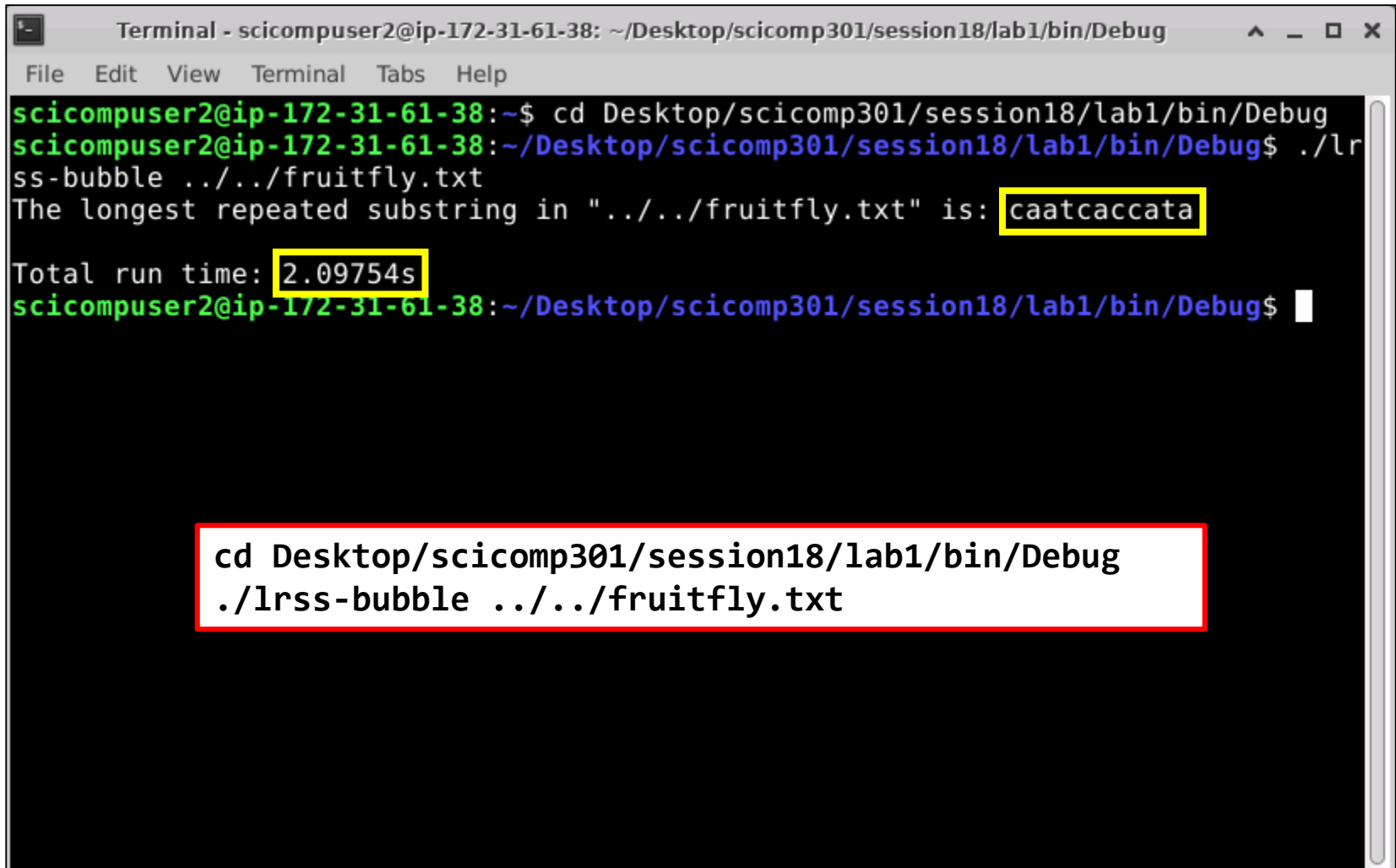
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
a	a	c	a	a	g	t	t	t	a	c	a	a	g	c

View Lab 1 – Longest Repeated Substring

<http://www.fruitfly.org/sequence/download.html>



Run Lab 1 – Longest Repeated Substring



A terminal window titled "Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab1/bin/Debug". The window contains the following text:

```
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab1/bin/Debug
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$ ./lr
ss-bubble ../../fruitfly.txt
The longest repeated substring in "../../fruitfly.txt" is: caatcaccata
Total run time: 2.09754s
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$
```

The output "caatcaccata" and the run time "2.09754s" are highlighted with yellow boxes. Below the terminal window, a red-bordered box contains the commands:


```
cd Desktop/scicomp301/session18/lab1/bin/Debug
./lrss-bubble ../../fruitfly.txt
```

<http://www.wolframalpha.com/input/?i=CAATCACCATA>

[illegible]


Exact matches to reference human genome: [Show genes](#)

chromosome 1 (80 matches)



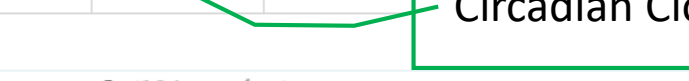
positions	7 890 455	15 992 109	19 226 283	...
genes	PER3	DDI2		

chromosome 2 (104 matches)



positions	3 127 827	4 091 837	6 442 141	...
genes	TSSC1	LOC100505594		

chromosome 3 (74 matches)



positions	375 420	4 304 431	4 806 486	...
genes	CHL1	SETMAR		

Circadian Clock

Tumor Suppressor

DNA Double Strand Repair

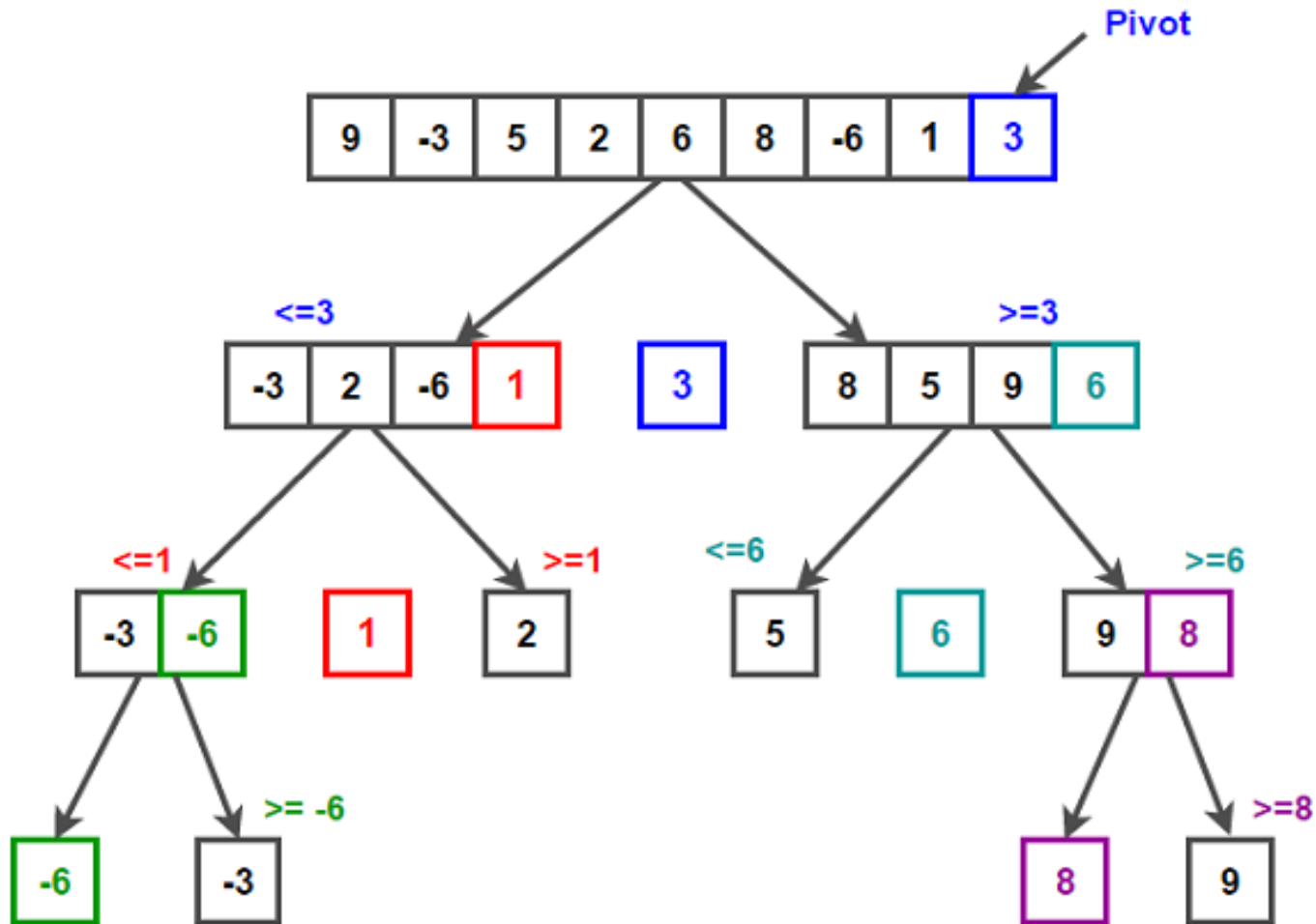
The Bubble Sort Algorithm

- The Bubble Sort is not very efficient because it can “move” only **one element** only **one slot** during **one pass**
 - The code spends a lot of time scanning through the array, mostly **reconfirming** that no swaps are needed between **adjacent** elements – little new information is gleamed from each pass
 - Therefore it wastes time constantly **rediscovering** that two elements are already in the right order – this is the culprit
- What if we could move an item with a **higher** value **farther** to the **end** of the array in **one big jump**, rather than only a single slot each pass?
 - And can we stop wasting time scanning through groups of elements we have **already proven to be in order**?

The Quicksort Algorithm

- Invented by **Tony Hoare** in 1959, **Quicksort** is an efficient sorting algorithm that uses a **divide & conquer** strategy to overcome the problems of Bubble Sort
 - On each pass, a portion of the vector is **split** according to a **pivot** value – those elements with a value \leq the **pivot** are swapped with those elements with a value $>$ the **pivot**
 - This act of partitioning separates the current section of the vector into a “**left**” portion having **smaller** values than the pivot, and a “**right**” portion having **larger** values than the pivot
- The **left** and **right** portions are then sorted (**conquered**) independently, and each portion is further split (**divided**) into smaller and smaller sized sub-portions, until the whole vector is eventually sorted

The Quicksort Algorithm



Open Lab 2 – LRSS Quicksort

Bubble Sort

```
// Bubble sort the suffix array
while (true)
{
    bool swapped = false;
    for (size_t i{}; i < suffixes.size() - 1; ++i)
        if (suffixes.at(i) > suffixes.at(i + 1))
        {
            string temp = suffixes.at(i);
            suffixes.at(i) = suffixes.at(i + 1);
            suffixes.at(i + 1) = temp;
            swapped = true;
        }
    if (!swapped)
        break;
}
```

Quicksort

```
template<typename T>
int partition(vector<T>& v, size_t left, size_t right)
{
    T pivot = v.at(right);
    size_t i = left;

    for (size_t j = left; j < right; ++j)
        if (v.at(j) <= pivot)
        {
            v.at(i).swap(v.at(j));
            i++;
        }

    v.at(right) = v.at(i);
    v.at(i) = pivot;

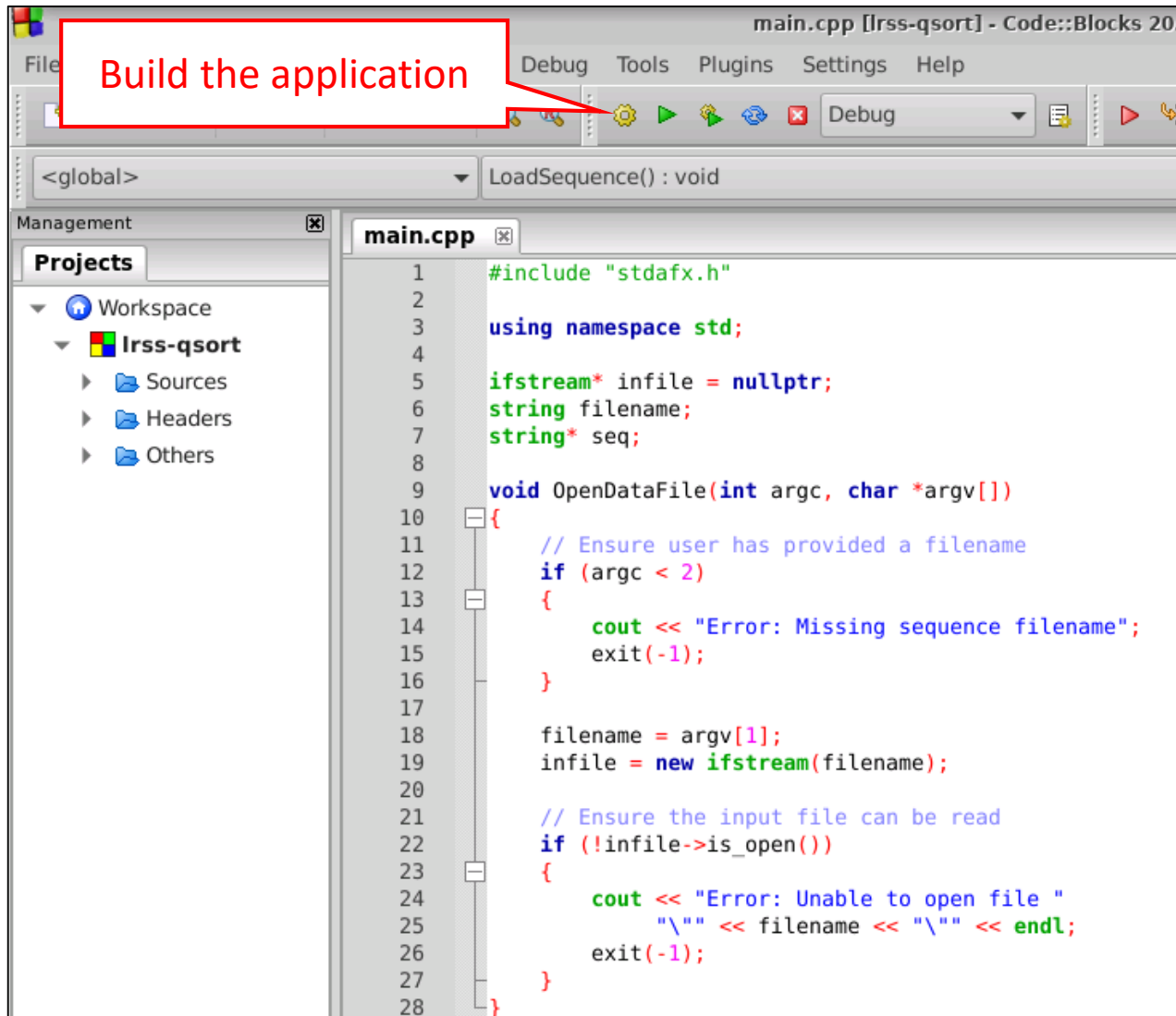
    return i;
}

template<typename T>
void quicksort(vector<T>& v, int left, int right)
{
    if (left < right)
    {
        int q = partition(v, left, right);
        quicksort(v, left, q - 1);
        quicksort(v, q + 1, right);
    }
}
```

Quicksort requires longer
and more complex code
than bubble sort...

Is it worth it?

Build Lab 2 – LRSS Quicksort



Run Lab 2 – LRSS Quicksort



A terminal window titled "Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab2/bin/Debug". The window contains the following text:

```
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab2/bin/Debug
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab2/bin/Debug$ ./lr
ss-qsort ../../fruitfly.txt
The longest repeated substring in "../../fruitfly.txt" is: caatcaccata
Total run time: 0.031733s
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab2/bin/Debug$
```

Below the terminal window, a red-bordered box contains the following commands:

```
cd Desktop/scicomp301/session18/lab2/bin/Debug
./lrss-qsort ../../fruitfly.txt
```

Check Lab 2 – LRSS Quicksort

Bubble Sort

```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab1/bin/Debug
File Edit View Terminal Tabs Help
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab1/bin/Debug
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$ ./lr
ss-bubble ../../fruitfly.txt
The longest repeated substring in "../../fruitfly.txt" is: caatcaccata
Total run time: 2.09754s
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab1/bin/Debug$
```

Quicksort

```
Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab2/bin/Debug
File Edit View Terminal Tabs Help
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab2/bin/Debug
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab2/bin/Debug$ ./lr
ss-qsort ../../fruitfly.txt
The longest repeated substring in "../../fruitfly.txt" is: caatcaccata
Total run time: 0.031733s
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab2/bin/Debug$
```

65x
faster!

The Role of Lecithin-Retinol Acyl Transferase in Keratinocyte Mechanics

Emily Peterson

Smithtown High School East

Project ID 2250

LRAT and Skin Cancer

- \downarrow Cell membrane \Rightarrow \uparrow cancer
- \downarrow Keratin \Rightarrow \downarrow Cell membrane
- \downarrow K5 \Rightarrow \downarrow Keratin
- \uparrow RA \Rightarrow \downarrow K5
- \uparrow Retinol \Rightarrow \uparrow RA
- \downarrow LRAT \Rightarrow \uparrow Retinol
- \downarrow LRAT \Rightarrow \uparrow skin cancer
- Cancer is invasive
- Cytoskeleton filaments
- K5 is a gene
- Retinoic Acid regulates K5
- Retinol is Vitamin A
- LRAT moderates Retinol
- LRAT controls cell stiffness?

LRAT

TCTGCTCCTCGGGCGGCCTTGAGCAGTGCCTAACGTTGAGCGTGAGGCTCGTGCTCCGGGTCTCGCGGGCGGCCGCCTCGGGCGTCGAGTCCCGGAG
ATTGGACAGACACCAGAGCCTGGGGACCGCGGAGTGACCGGTGGGGCTGGAGGGCGGCGCCGCGCCTTCTGGGGAGACGCGGAGGTATCAGGACCT
GGGCTATGCTCCTGATTTTATACAGACTGCCATGGCTCCAGATATAAAAGACCAAATAAAAAAGATAAGAATTGCTGGCAACATATGCACTAAACTTC
GTTTTGAAAAATCCCCTTGATGCCGATCACTGAGAAGTGATCCACAGGATGAAGAACCCCATGCTGGAGGTGGTGTCTTTACTACTGGAGAAGCTGC
TCCTCATCTCCAAC TTCACGCTCTTTAGTTGCGGGCGCCGCGGGCGAAGACAAAGGGAGGAACAGTTTTTATGAAACCAGCTCTTTCCACCGAGGCGA
CGTGCTGGAGGTGCCCCGACCCACCTGACCCACTATGGCATCTACCTAGGAGACAACCGTGTGCCCACATGATGCCCCGACATCCTGTTGGCCCTG
ACAGACGACATGGGGCGCACGCAGAAGGTGGTCTCCAACAAGCGTCTCATCCTGGGCGTTATTGTCAAAGTGGCCAGCATCCGCGTGGACACAGTGG
AGGACTTCGCCTACGGAGCTAACATCCTGGTCAATCACCTGGACGAGTCCCTCCAGAAAAAGGCACTGCTCAACGAGGAGGTGGCGCGGAGGGCTGA
AAAGCTGCTGGGCTTTACCCCTACAGCCTGCTGTGGAACAACTGCGAGCACTTCGTGACCTACTGCAGATATGGCACCCCGATCAGTCCCCAGTCC
GACAAGTTTTGTGAGACTGTGAAGATAATTATTCGTGATCAGAGAAGTGTTCTTGCTTCAGCAGTCTTGGGATTGGCGTCTATAGTCTGTACGGGCT
TGGTATCATACACTACCCCTTCTGCAATTTTTATTCCATTCTTCTATGGATGGCTGGCTAACTTCATACCCCATGTCAAGTGTGTATTCTGTAT
GTAAATATGTTTATATTTATAGAGCATCAATCAATATAAGCATTATTGAGAAAAATGTGACCGTAACACTGTGTTCTGGATAAAAAATGTGATTAGG
AATCACGCAAAGTGCTTACTGTGTAAGCCCAAGAACAAAGGCTTTCTGAATCTTCTCAGGCAGTTCAGATTTAAAGCACCATCCAAACCTTGAAAT
ACGACAGGGTGTGGTAGAATTAGCAATATGAGAAAACAGCCCTAAAATGATAGCCACAAGAGATTAATTGTGTTTTTTTTCTCCTGTAATCCT
TGTAAGTCAAGAATGTTTTATTGTGATTGGAAGACTTATATTGAGACCAGTAACCTTACTGTAAATTTACTTTGTTTCATTGAAAAACAAATTGATA
AACATATTAACCTGGAAGAATTTTTCTTTATTCAAATGAAAACATGTTTGATGACTGGTCAAAAAATAAGCTCATAATCTATTTTTTTCATGTAGTAT
ATAAGTCAAGAATGTTTTATTGTGATTGTAACCAATATTGGCAAATAGTACTTTAATGATGAAGTAAATGACCAGAAATTATAGAAATCTGTG
TTTTCTGTAAAAATAGCACTATAGTATCACTTGAACAATTTGATTTGGCTTTACTTACTAGGAAGCCTGGAATTCATTATTTTTTCTTTTATGT
GCCACTGTGGCTACTTTAAATCACTCTGAGAGGTAATGGATATAGGATTGAAGTTATGTGGGTATTTGGCATGTGTGTGTAATAAATATGTAGA
TAGTCACATATACACAGACTGAGAGATAAATTGTTCTTGATTGCTTTATTATCATCATACTAGTGTGTTTATTATAGAGTATCTGTAGAGGTGAATG
TAAAGTAAGTCCAATCTATTTTCTTATGTGATTGAATTTGTAGTGTTAACTTGCATATATGTTATTGGATGGGTTGTCTTTTAAAGCATTTACTAA
TGTAAGTCTGAAATTTTTTAAAGCCTTCAGATTTGTTTTCTAGTCACTTTTTTCCATATCATTCTAATTATAGTTTATATCCTTAAAGAAAGGATGC
CACAGTAGTATGTAAACCCAAACAAGTAGAACCCAAGCAAATAAAATTATTTAAATAAATTTAAAGTGGCTTAGTACTGCCAGTCATGTAAATTG
ATTCTGCTGAGGGTCTTATAAGAATTGAGATATAACAATGGTAAAACAAGCATTCAAGCACTTTTACAAAATTACCAAATCTTAAATGAAGCCAC
AGCTAGACTTGCATTTAGGTATTAATAATTGCTTTCTTAACTGTCAAGAATCACAAAATAACAAATCATATTATGAGTGAATATGGGGAGGGCGGGG
CCAATCAGTCAATGATAATCTGAACAAATTTTAAAGAGCAGATTTTAGATTAATAATGTTTTATCACCCTAATTTGCCACAACAACTCAGTATTT
AATTTTTCAAATTAATATTAATTTAAGTATTTTAAATAATTAACATTAATGGCAACACCATAGAATATAGGTGTTCTCTGGACCTATTC
TAACCACTTAAATTTATCTTAAGTATGCATACATAAAAGCAACCACTATGAGAACTACCGTGTAGTGGTTTTTCACTTACTGTATATTACCTTGT
AGGAATAGTTTAAAGGAAATTCATTTCTTAAAAATATAGTGTCTCAAATAATTAATTTTTTTGCAAACTTTAGTTATTACAGGCAGCAAAAACCACT
GTCTGAACTAAATCTGTGTTCAAAGATGAAGACCCCTATTAAAGCCAAGGACGTTCTTAAAGATTGGAAGTACATAATTAGTCTTGACTTACTT
CATTAAAGCAAGATTCAATTCCT

Open Lab 3 – Substring Frequency

```
void WriteSubstrings()
{
    cout << "Creating file \"
        << filename << \" ...\";

    *outfile << "length,count,freq,seq,proteins"
        << endl;

    map<string, int> table;

    for (int len{ 3 }; len < 19; ++len)
    {
        table.clear();
        for (size_t pos{}; pos <= seq->length() - len; ++pos)
        {
            string key = seq->substr(pos, len);
            auto p = table.find(key);
            if (p == table.end())
                table.insert(pair<string, int>(key, 1));
            else
                p->second++;
        }

        // Copy sorted map to an unsorted list
        vector<pair<string, int>> list;
        for (auto& p : table)
            list.push_back(p);

        WriteFreq(list, len, true, 5);
        WriteFreq(list, len, false, 5);
    }

    cout << " done!" << endl;
}
```

We sum the occurrences in which each substring of the given length appears

View Lab 3 – Substring Frequency

```
void WriteFreq(vector<pair<string, int>>& list, int len,
              bool topmost, size_t limit)
{
    string freqLabel;
    if (topmost)
    {
        freqLabel = "Most";
        // Sort list by decreasing substring frequency
        sort(list.begin(), list.end(),
             [](const pair<string, int> &a,
                const pair<string, int> &b)
             {
                 return a.second > b.second;
             });
    }
    else
    {
        freqLabel = "Least";
        // Sort list by increasing substring frequency
        sort(list.begin(), list.end(),
             [](const pair<string, int> &a,
                const pair<string, int> &b)
             {
                 return a.second < b.second;
             });
    }

    list.size() > limit ? limit : list.size();
    for (size_t row{}; row < limit; ++row)
    {
        *outfile << len << "," << list.at(row).second
                  << "," << freqLabel << "," << list.at(row).first
                  << "," << GetCodons(list.at(row).first) << endl;
    }
}
```

Instead of targeting **cout**, we can insert into a file stream to create a CSV file

DNA Codon Table

Standard genetic code

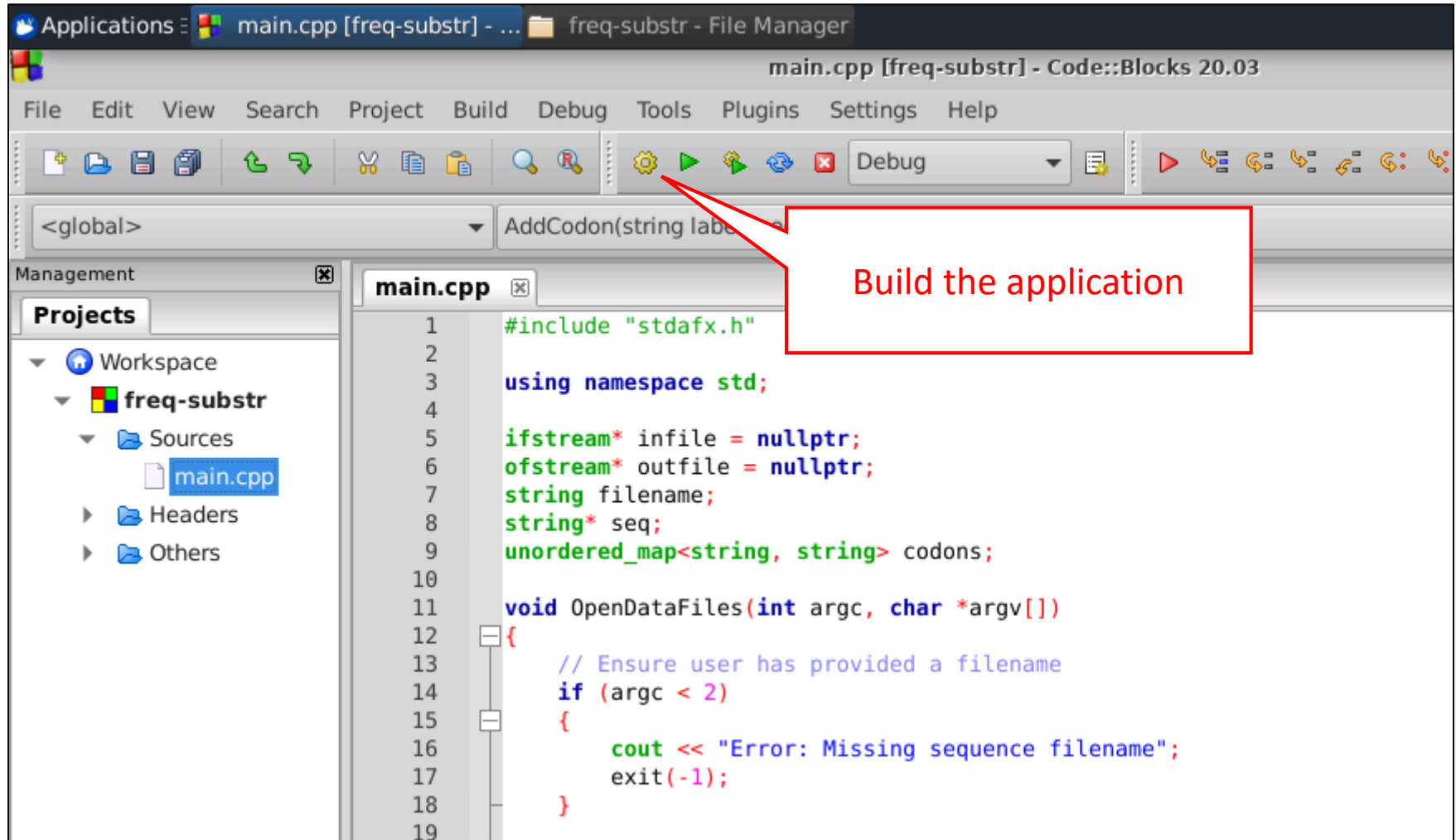
1st base	2nd base								3rd base
	T		C		A		G		
T	TTT	(Phe/F) Phenylalanine	TCT	(Ser/S) Serine	TAT	(Tyr/Y) Tyrosine	TGT	(Cys/C) Cysteine	T
	TTC		TCC		TAC		TGC		C
	TTA	(Leu/L) Leucine	TCA		TAA ^[B]	Stop (Ochre)	TGA ^[B]	Stop (Opal)	A
	TTG		TCG		TAG ^[B]	Stop (Amber)	TGG	(Trp/W) Tryptophan	G
C	CTT	(Leu/L) Leucine	CCT	(Pro/P) Proline	CAT	(His/H) Histidine	CGT	(Arg/R) Arginine	T
	CTC		CCC		CAC		CGC		C
	CTA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CTG		CCG		CAG		CGG		G
A	ATT	(Ile/I) Isoleucine	ACT	(Thr/T) Threonine	AAT	(Asn/N) Asparagine	AGT	(Ser/S) Serine	T
	ATC		ACC		AAC		AGC		C
	ATA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	ATG ^[A]	(Met/M) Methionine	ACG		AAG		AGG		G
G	GTT	(Val/V) Valine	GCT	(Ala/A) Alanine	GAT	(Asp/D) Aspartic acid	GGT	(Gly/G) Glycine	T
	GTC		GCC		GAC		GGC		C
	GTA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GTG		GCG		GAG		GGG		G

View Lab 3 – Substring Frequency

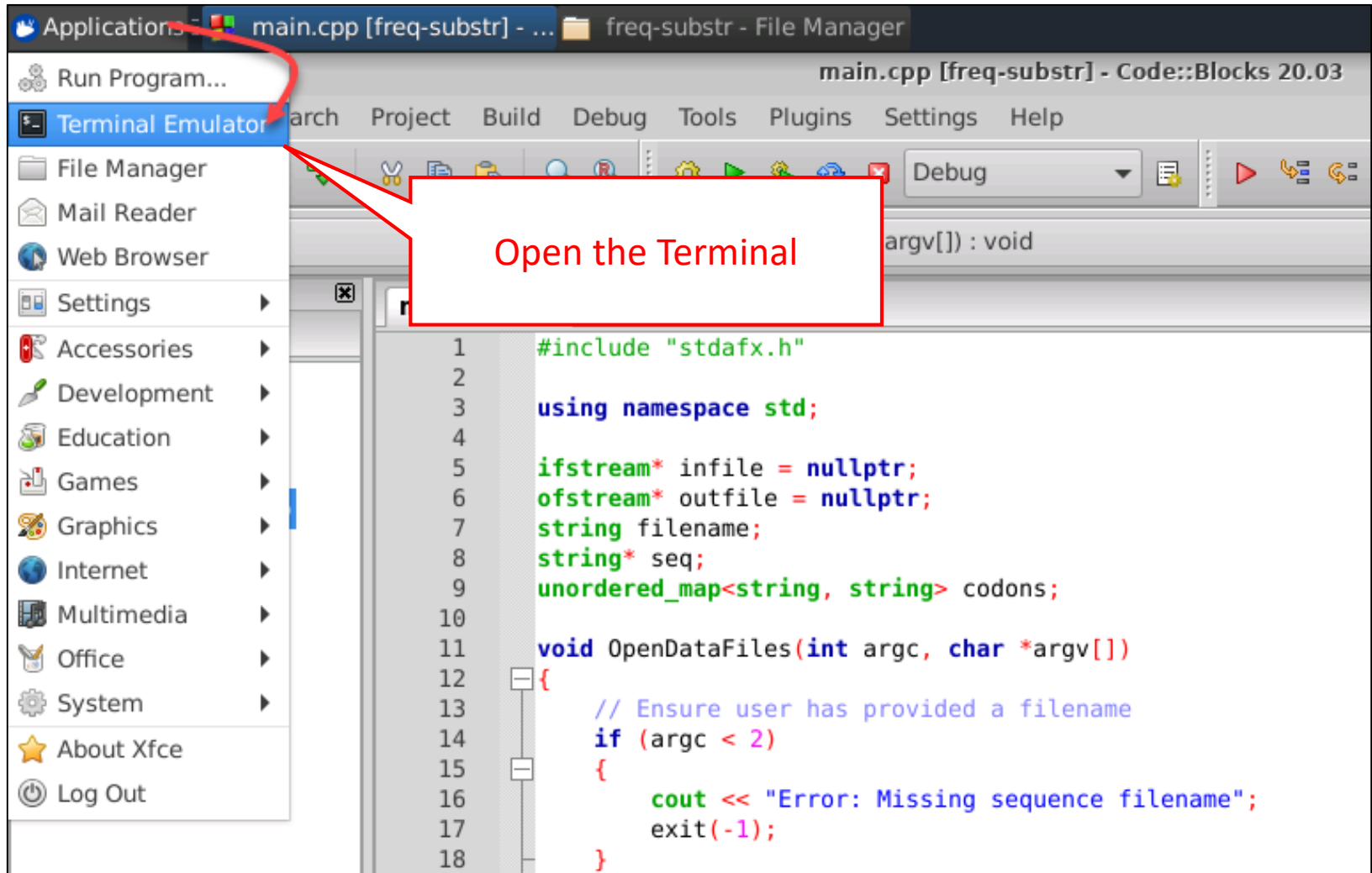
```
void AddCodon(string label, vector<string> seqs)
{
    for (auto& s : seqs)
        codons.insert(pair<string, string>(s, label));
}

void InitCodons()
{
    AddCodon("Ala", { "GCT", "GCA", "GCC", "GCG" }); // Alanine (Ala/A)
    AddCodon("Arg", { "CGT", "CGC", "CGA", "CGG", "AGA", "AGG" }); // Arginine (Arg/R)
    AddCodon("Asn", { "AAT", "AAC" }); // Asparagine (Asn/N)
    AddCodon("Asp", { "GAT", "GAC" }); // Aspartic Acid (Asp/D)
    AddCodon("Cys", { "TGT", "TGC" }); // Cysteine (Cys/C)
    AddCodon("Gln", { "CAA", "CAG" }); // Glutamine (Gln/Q)
    AddCodon("Glu", { "GAA", "GAG" }); // Glutamic Acid (Glu/E)
    AddCodon("Gly", { "GGT", "GGC", "GGA", "GGG" }); // Glycine (Gly/G)
    AddCodon("His", { "CAT", "CAC" }); // Histidine (His/H)
    AddCodon("Ile", { "ATT", "ATC", "ATA" }); // Isoleucine (Ile/I)
    AddCodon("Leu", { "TTA", "TTG", "CTT", "CTC", "CTA", "CTG" }); // Leucine (Leu/L)
    AddCodon("Lys", { "AAA", "AAG" }); // Lysine (Lys/K)
    AddCodon("Met/Start", { "ATG" }); // Methionine (Met / M) & Start
    AddCodon("Phe", { "TTT", "TTC" }); // Phenylalanine (Phe/F)
    AddCodon("Pro", { "CCT", "CCC", "CCA", "CCG" }); // Proline (Pro/P)
    AddCodon("Ser", { "TCT", "TCC", "TCA", "TCG", "AGT", "AGC" }); // Serine (Ser/S)
    AddCodon("Thr", { "ACT", "ACC", "ACA", "ACG" }); // Threonine (Thr / T)
    AddCodon("Trp", { "TGG" }); // Tryptophan (Trp/W)
    AddCodon("Tyr", { "TAT", "TAC" }); // Tyrosine (Tyr/Y)
    AddCodon("Val", { "GTT", "GTC", "GTA", "GTG" }); // Valine (Val/V)
    AddCodon("Stop", { "TAA", "TGA", "TAG" }); // Stop
}
```

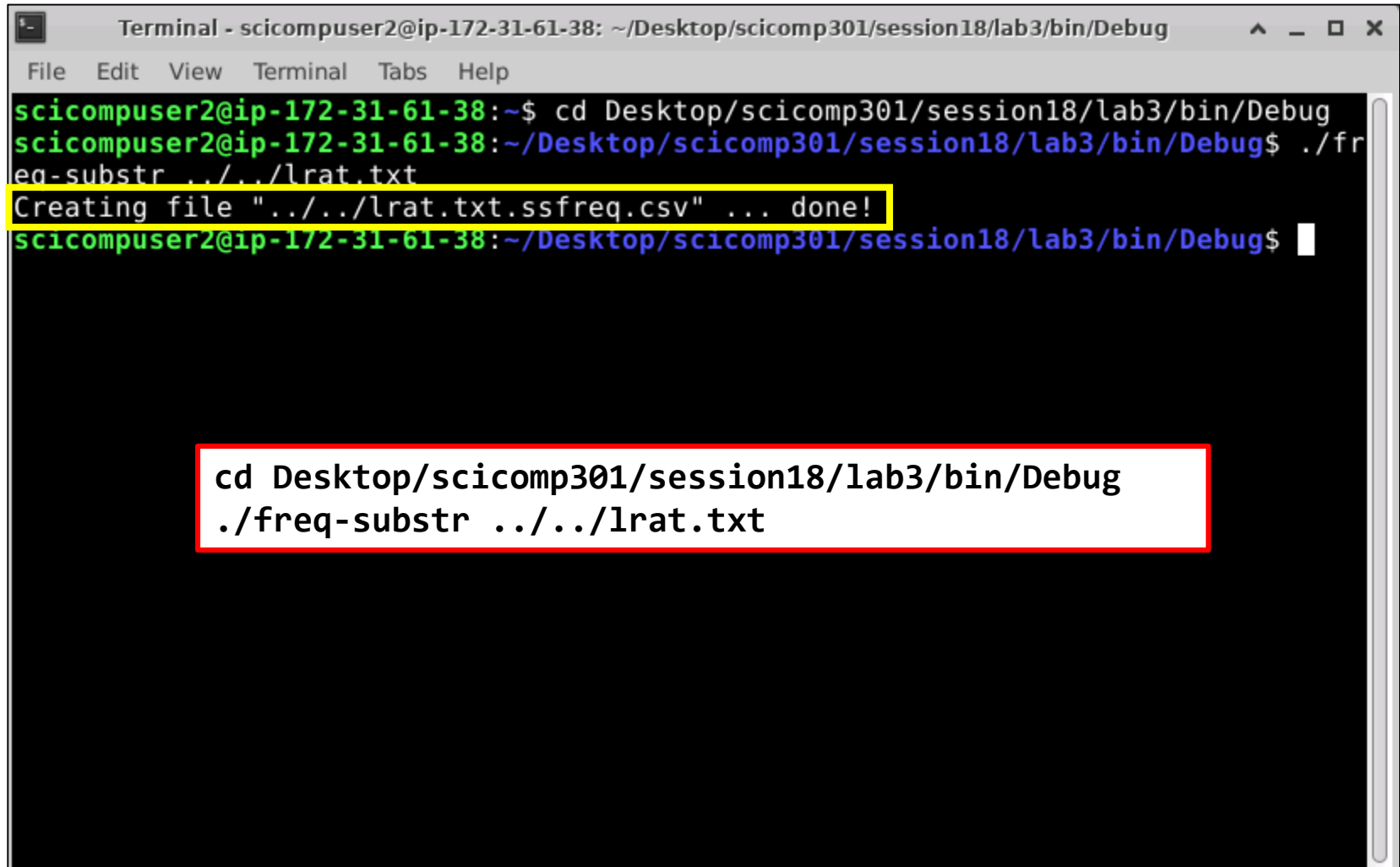
Build Lab 3 – Substring Frequency



Run Lab 3 – Substring Frequency



Check Lab 3 – Substring Frequency

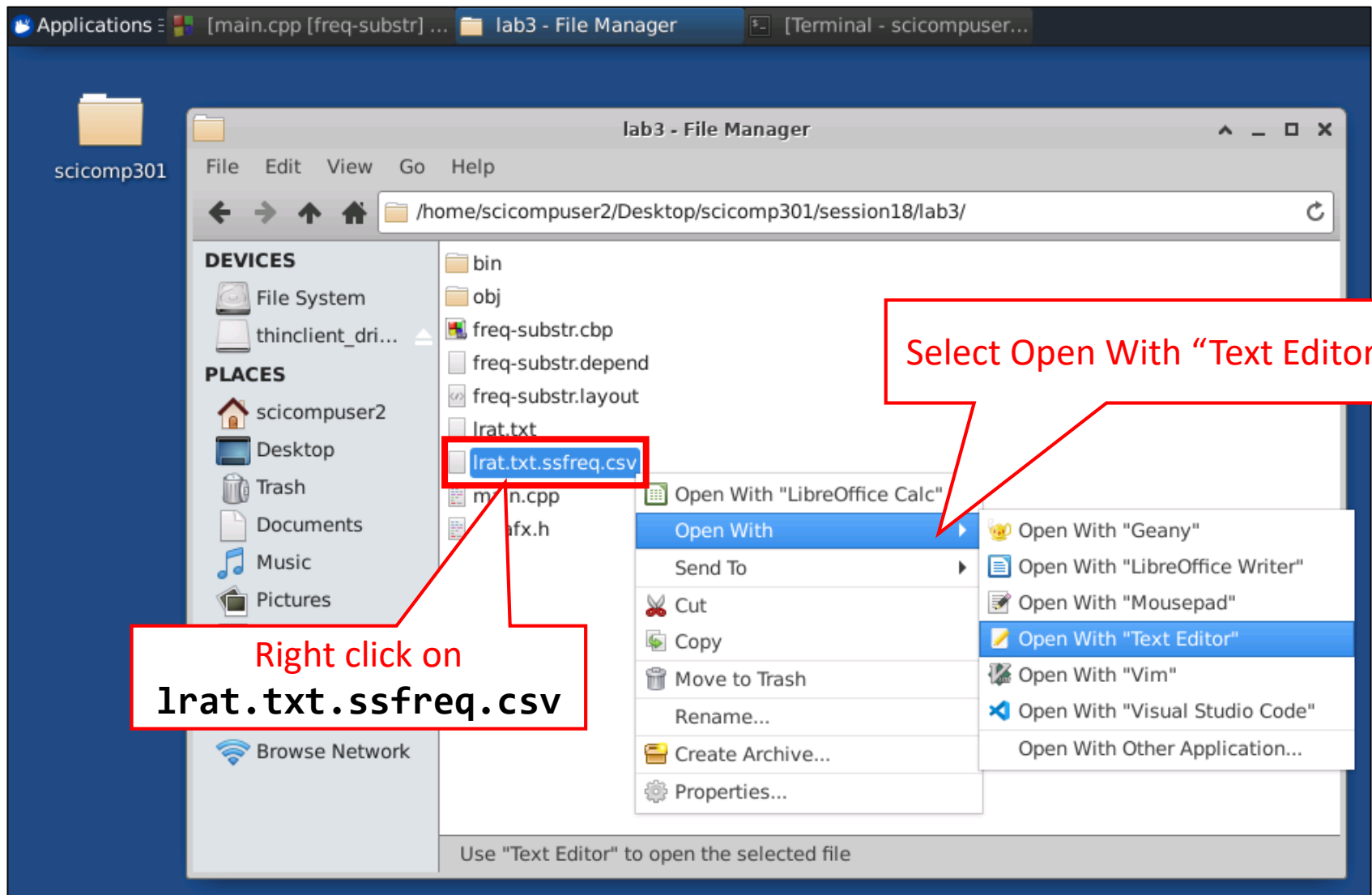
A terminal window titled "Terminal - scicompuser2@ip-172-31-61-38: ~/Desktop/scicomp301/session18/lab3/bin/Debug". The window has a menu bar with "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal shows the following commands and output:

```
scicompuser2@ip-172-31-61-38:~$ cd Desktop/scicomp301/session18/lab3/bin/Debug
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab3/bin/Debug$ ./freq-substr ../../lrat.txt
Creating file "../../lrat.txt.ssfreq.csv" ... done!
scicompuser2@ip-172-31-61-38:~/Desktop/scicomp301/session18/lab3/bin/Debug$
```

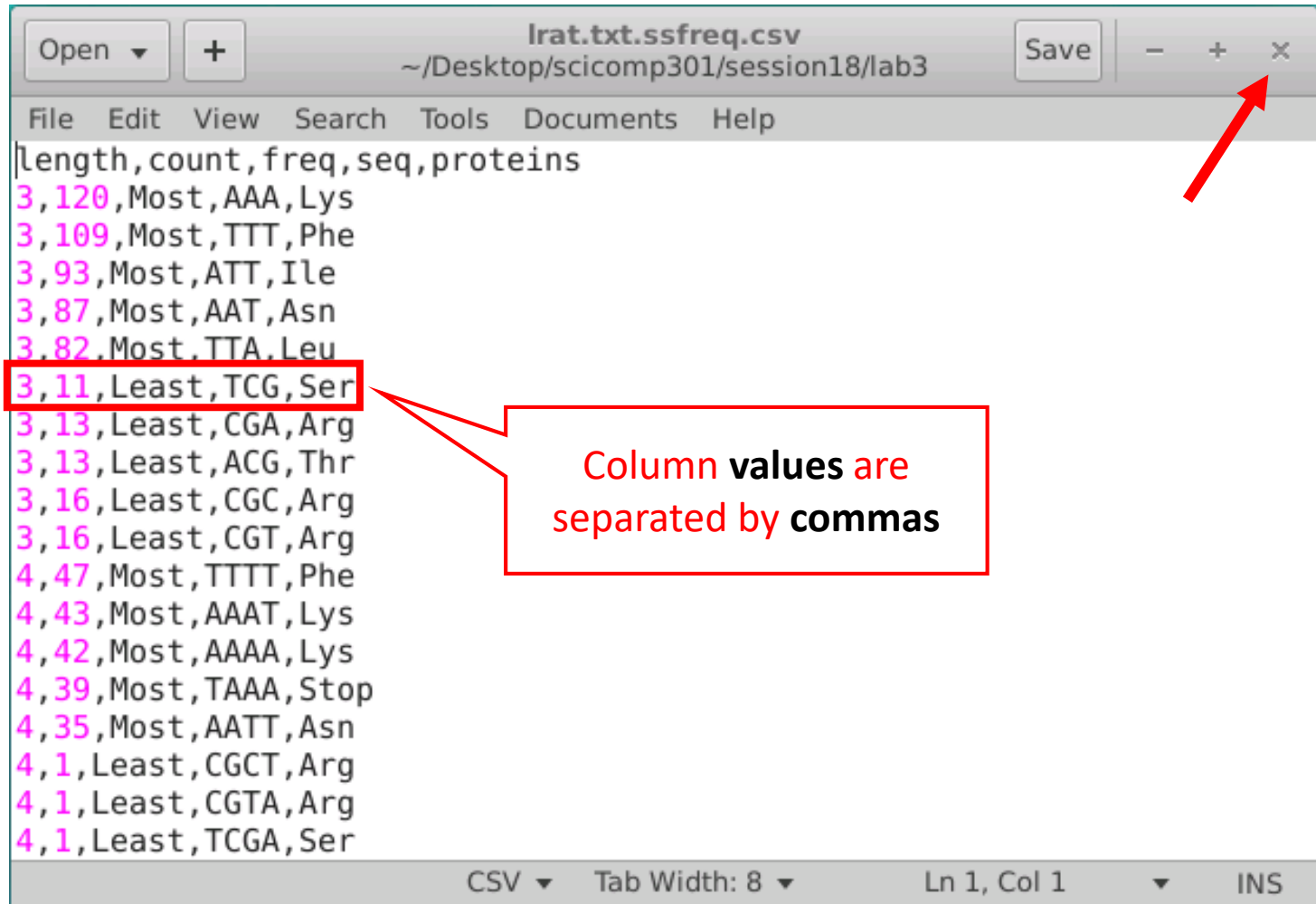
The output line "Creating file "../../lrat.txt.ssfreq.csv" ... done!" is highlighted with a yellow box. Below the terminal window, a red-bordered box contains the commands:

```
cd Desktop/scicomp301/session18/lab3/bin/Debug
./freq-substr ../../lrat.txt
```

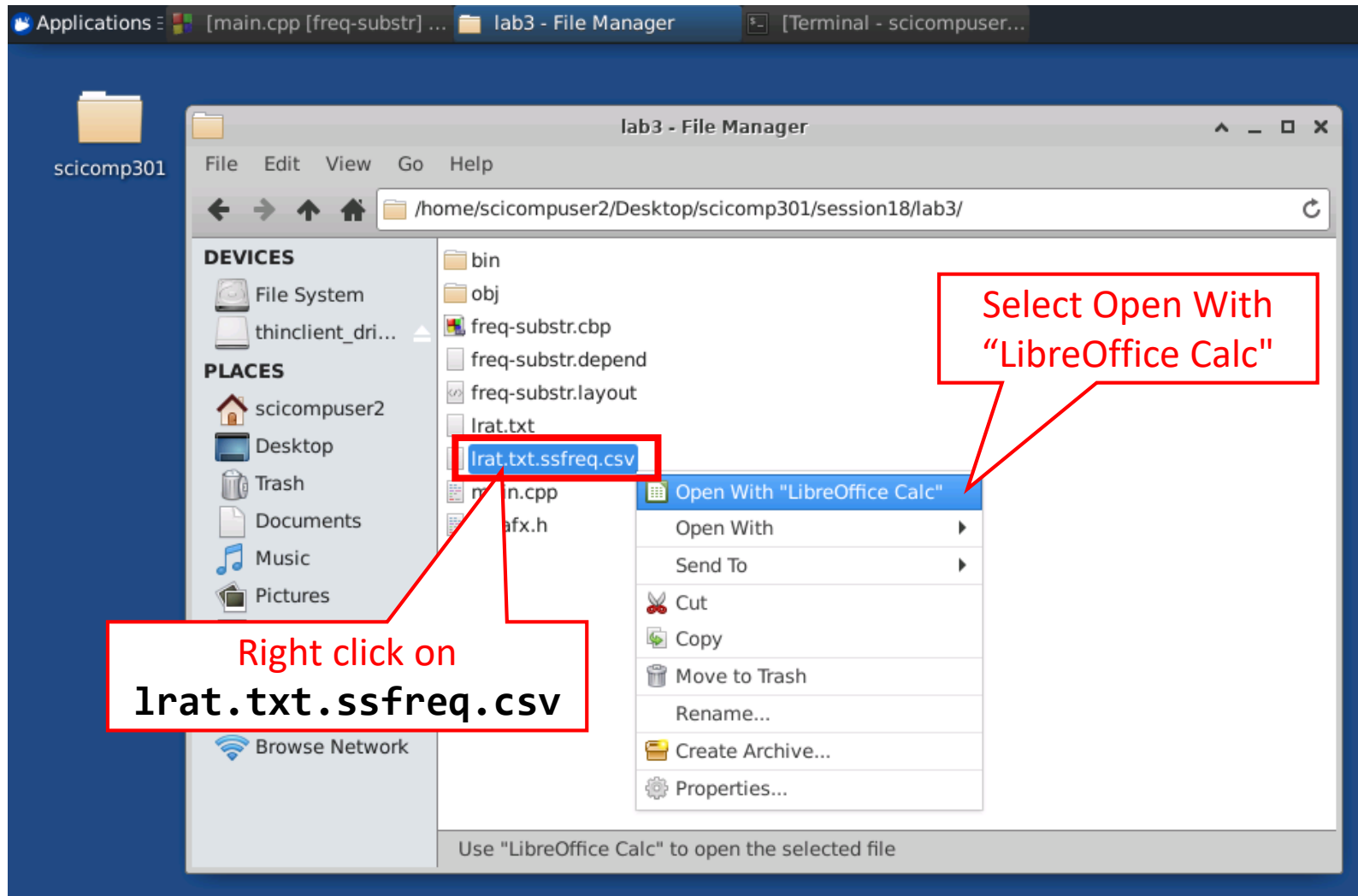
Check Lab 3 – Substring Frequency



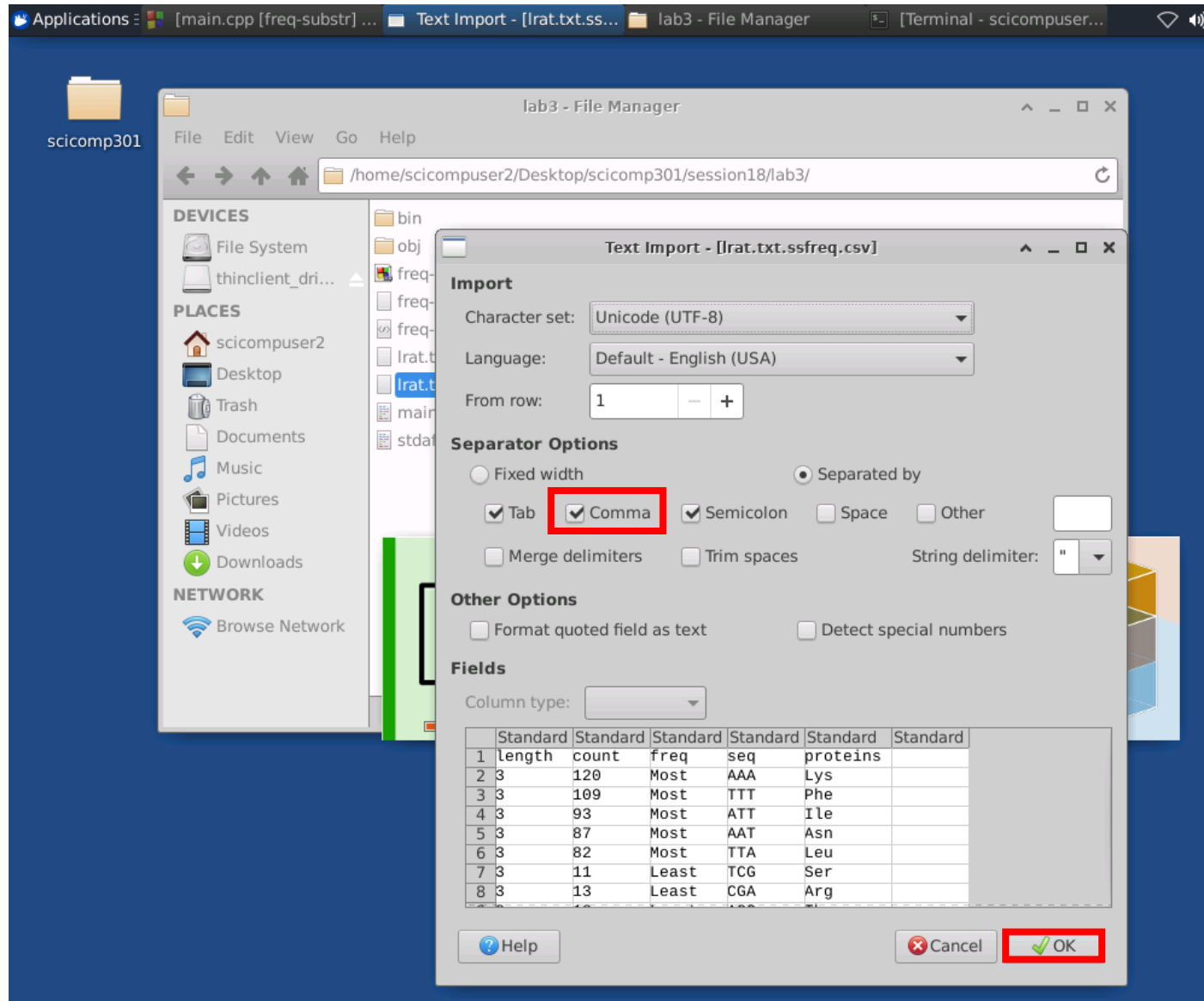
Comma Separated Value Files (.CSV)



Check Lab 3 – Substring Frequency



LibreOffice Calc \approx Microsoft Excel



Lab 3

Substring Frequency

	A	B	C	D	E	F	G	H	I
1	length	count	freq	seq	proteins				
2	3	120	Most	AAA	Lys				
3	3	109	Most	TTT	Phe				
4	3	93	Most	ATT	Ile				
5	3	87	Most	AAT	Asn				
6	3	82	Most	TTA	Leu				
7	3	11	Least	TCG	Ser				
8	3	13	Least	ACG	Thr				
9	3	13	Least	CGA	Arg				
10	3	16	Least	CGC					
11	3	16	Least	CGT					
12	4	47	Most	TTTT	Phe				
13	4	43	Most	AAAT	Lys				
14	4	42	Most	AAAA	Lys				
15	4	39	Most	TAAA	Stop				
16	4	35	Most	AATT	Asn				
17	4	1	Least	TCGA	Ser				
18	4	1	Least	CGAA	Arg				
19	4	1	Least	CGTA	Arg				
20	4	1	Least	CGCT	Arg				
21	4	1	Least	TGCG	Cys				
22	5	21	Most	TTTTT	Phe				
23	5	19	Most	TTAAA	Leu				
24	5	19	Most	TAAAA	Stop				
25	5	15	Most	AAATA	Lys				
26	5	15	Most	AAATT	Lys				
27	5	1	Least	CGGCG	Arg				
28	5	1	Least	TGCGA	Cys				
29	5	1	Least	TGCCT	Cys				
30	5	1	Least	TGCCG	Cys				
31	5	1	Least	GTTGC	Val				
32	6	11	Most	TTTTTT	Phe	Phe			
33	6	9	Most	TTAAAA	Leu	Lys			
34	6	8	Most	ATTAAA	Ile	Lys			
35	6	8	Most	AAATAA	Lys	Stop			
36	6	7	Most	TTTAAA	Phe	Lys			
37	6	1	Least	AATCAG	Asn	Gln			

Lists the 5 most frequent and 5 most infrequent substrings of a given length

Codons are mapped where possible

Check Lab 3 – Substring Frequency

17	1	Least	AGCAAGATTCAATTCCT	Ser, Lys, Ile, Gln, Phe					
17	1	Least	AGCAAGATTCAATTCCT	Ser, Lys, Ile, Gln, Phe					
18	1	Most	TCTGCTCCTCGGGCGGCC	Ser, Ala, Pro, Arg, Ala, Ala	FGG				
18	1	Most	CTGCTCCTCGGGCGGCCT	Leu, Leu, Leu, Gly, Arg, Pro	FGG				
18	1	Most	TGCTCCTCGGGCGGCCTT	Cys, Ser, Ser, Gly, Gly, Leu	FGG				
18	1	Most	GCTCCTCGGGCGGCCTTG	Ala, Pro, Arg, Ala, Ala, Leu	FGG				
18	1	Most	CTCCTCGGGCGGCCTTGA	Leu, Leu, Gly, Arg, Pro, Stop	FGG				
18	1	Least	ATTAAAGCAAGATTCAAT	Ile, Lys, Ala, Arg, Phe, Asn	LRAT				
18	1	Least	TTAAAGCAAGATTCAATT	Leu, Lys, Gln, Asp, Ser, Ile	LRAT				
18	1	Least	TAAAGCAAGATTCAATTC	Stop, Ser, Lys, Ile, Gln, Phe	LRAT				
18	1	Least	AAAGCAAGATTCAATTCC	Lys, Ala, Arg, Phe, Asn, Ser	LRAT	MME			
18	1	Least	AAGCAAGATTCAATTCCT	Lys, Gln, Asp, Ser, Ile, Pro	LRAT				
19	1	Most	TCTGCTCCTCGGGCGGCCT	Ser, Ala, Pro, Arg, Ala, Ala					
19	1	Most	CTGCTCCTCGGGCGGCCTT	Leu, Leu, Leu, Gly, Arg, Pro					
19	1	Most	TGCTCCTCGGGCGGCCTTG	Cys, Ser, Ser, Gly, Gly, Leu					

Dave Biersach:

This protein is important for blood clot formation (coagulation), which is needed to stop excessive bleeding after injury

Dave Biersach:

A common acute lymphocytic leukemia antigen

LRAT Genetic Homologs

FGG

From Wikipedia, the free encyclopedia

Fibrinogen gamma chain, also known as **FGG**, is a human gene found on Chromosome 4.

The protein encoded by this gene is the gamma component of fibrinogen, a blood-borne glycoprotein composed of three pairs of nonidentical polypeptide chains. Following vascular injury, fibrinogen is cleaved by thrombin to form fibrin which is the most abundant component of blood clots. In addition, various cleavage products of fibrinogen and fibrin regulate cell adhesion and spreading, display vasoconstrictor and chemotactic activities, and are mitogens for several cell types. Mutations in this gene lead to several disorders, including dysfibrinogenemia, hypofibrinogenemia and thrombophilia.^[1] Alternative splicing of the mRNA chain results in two transcript variants; the common γ A chain and the alternatively spliced γ' chain. Approximately 10% of the total plasma fibrinogen consists of γ A/ γ' fibrinogen, with <1% consisting of γ'/γ' fibrinogen. Increased and decreased levels of γ A/ γ' fibrinogen have been associated with CAD and DVT respectively.

FGG

GTGCACCTCCATCAGCGCCTATAAGAGTGAAGGGGAGTACGCGGAGTTCTCCTTCCCACTCAACCTTGGAGAGGAAAGCCTGCAGGGAGAGTTGAGATGGAAGGCAGAGAAGGCTCCTTCTTCCAGTCTTGGATCACCTTCTCCCTAAAGAACCAA
AGGTGTCTGTGCGAAGTCTACTAGCAACCCCAAGTTCAGCTGTCCGAAACGCTCCCACTCACCTTTCAGATACCCCAAGGTCCTCTTCAAGTTTGTGGTCTTGGCAACCTGACCTTGAATCTGAGGAGGTGAACCTG
GTGGTGATGAAAGTGAATCAGCCCGACAGCAACTTGAACCTGTGAGGTGATGGGACCCACTCACCAAGATGAGACTGATCTTGAAGCAGGAGAAATCAGGAGGCCAGGGTCTCCAGGCAGGAGAACTGATTCAGGTGACAGGCCCTGAAGCAGG
GGTGTGGCAATGTCTACTGAGTGAAGGTGAAGAGGTCAAGATGGACTCCAAGATCAGGTTTATCCAAAGGGTTGAATCCGGCAGCTGCACCACATCTGGACGCCAGAAGATGCTGTGAACACAGGGACAGGAACCTGCCTCTCTGGCCC
CTCTGGGCCCCCACCACCATCACCACCATTGAGTCTAGACACGTGATTAATTAAGGATCCCCGACCTCGACCTCTGGCTATAAAGGAAATTTATTTTCATTGCAATAGTGTGTGGAATTTTTGTGTCTCTCACTCGGAAGGACATATGGGAGGG
CAAATCATTGGTGCAGATCCCTCGGAGATCTCTAGCTAGAGGATCGATCCCCGCCCGGACGAACATAACCTGACTACGACATCTTGCCCTTCTTCGCGGGGAGTGCATGTAATCCCTTCAGTTGGTTGGTACAACTTGCCAACCTGGGCCCTGT
TCCACATGTGACACGGGGGGGACCAACACAAAGGGTTCTCTGACTGTAGTTGACATCCTTATAAATGGATGTGCACATTTGCCAACACTGAGTGGCTTTCATCTGGAGCAGACTTTCAGCTGTGTGGACTGCAACACAACTTGCCTTTATGTG
TAACTCTTGGCTGAAGCTCTTACACCAATGCTGGGGGACATGTACTCCAGGGGCCAGGAAGACTACGGGAGCTACACCAACGTCAATCAGAGGGGCCGTGTGAGTACCAGTAAGCGGACCTCAAGAGGGGCATTAGCAATAGTGTATAAGG
CCCCTTGTTAAACCTAAACGGGTAGCATATGCTTCCGGGTAGTAGTATATACTATCCAGACTAACCTTAATTAATGATATGTTTACCAACGGGAAGCATATGCTATCGAATTAGGGTATGTAAGGAGCAGCGATATCTCCCA
CCCCATGAGCTGTACGGTTTTATTATACATGGGTGAGGATCCACGAGGGTAGTGAACCAATTTAGTCAAGGGCAGTGGCTGAAGATCAAGGAGCGGGCAGTGAACCTCTCTGAATCTTCGCTGCTTCTTCAATTCCTCTGTTTAGCTAATAG
AATAACTGCTGAGTTGTGAACAGTAAGGTGTATGTGAGGTGCTCGAAACAAAGGTTTCAGGTGACGCCCCAGAAATAAATTTGACGGGGGTTCAGTGGTGGCATTGTGCTATGACACCAATAAACCCTCACAAACCCCTTGGGCAATAAATACT
AGTGTAGGAATGAAACATCTGAATATCTTAAACAATAGAAATCCATGGGTGGGGACAAGCGTAAGAGCTGGATGTCATCTCACAGCAATTTATGGCTATGGGCAACACATAATCTAGTGCAATATGATACTGGGGTTATTAAGATGTGTCCCA
GGCAGGGACCAAGACAGGTGAACCATGTTGTTACACTCTATTTGTAACAAGGGGAAAGAGAGTGGACGCCGACAGCAGCGGACTCCACTGGTTGTCTCTAACACCCCCGAAATTAACAGGGGCTCCACGCCAATGGGGCCATAAAACAAGACAAGT
GGCACTCTTTTTTTGAAATTTGGAGTGGGGGACGCGTCAGCCCCACAGCGGCCCTGCGGTTTTGGACTGTAAATAAGGGTGAATAAATTTGGCTGATTGTAACCCCGTAACCACTGCGGTCAAACCACTTGCCCCAAAACCACTAATGG
CACCCCGGGGAATACCTGCAATAAGTAGGTGGGCGGGCAAGATAGGGCGCGATGTGCTGCATCTGGAAGGACAAATACACACTTGCGCTGAGCGCAAGCAGAGGTTGTTGGTCTCATATACAGAGTGCCTGAGAGCAGTGGTGGGCTAAAT
GTGGCATGGGTAGCATATACTACCCCAATATCTGGATAGCATATGCTTCAATCTATATCTGGGTAGCATATGCTATCTTAATCTATATCTGGGTAGTATATGCTATCTTAATCTATATCTGGGTAGTATATGCTATCTTAATCTGGGT
AGCATAGGCTATCTAATCTATATCTGGGTAGCATATGCTATCTTAATCTATATCTGGGTAGTATATGCTATCTTAATCTGATCCGGGTAGCATATGCTATCTTAATAGAGATAGGGGTAGTATATGCTATCTTAATTTATATCTGGGTAGCATATA
CTACCCAAATATCTGGATAGCATATGCTATCTTAATCTATATCTGGGTAGCATATGCTATCTTAATCTATATCTGGGTAGCATATGCTATCTTAATCTATATCTGGGTAGTATATGCTATCTTAATTTATATCTGGGTAGCATATGCTATCTTAATTT
ATATCTGGGTAGCATATGCTATCTTAATCTATCTGGGTAGCATATGCTATCTTAATCTATATCTGGGTAGTATATGCTATCTTAATCTGATCCGGGTAGCATATGCTATCTCATGATAAGCTGTCAAACTAGAGAATTTCTTGAAGACGAAG
GGCTCTGATGATACGCTATTTTTATAGTTAATGTATGATAAATAAGTTTCTTAGAGCTCAGTGGGCACTTTTCGGGGAATATGTCGGCGGAACCCCTATTTGTTTTCTAAATACATTCAAATATGATCCGCTCATGAGACAAATCCGCTG
ATAAATGCTTCAATAATATTGAAAAAGGAAGAGTATGAGTATTAACATTTTCGGTGTGCGCCCTTATCCCTTTTTTGCGGCATTTTGCTTCTGTTTTGCTCACCAGAAACGCTGGTGAAGTAAAAGATGCTGAAGATCAGTTGGGTGCACGAG
TGGGTACATCGAAGTGGATCTCAACAGCGCGTAAGATCTTGAGAGTTTTCGCGCGAAGAACGTTTCCAATGATGAGCACTTTAAAGTTCTGCTATGTGGCGGGTATTAATCCCGTGTGACGCCGGGCAAGAGCAACTCGGTGCGCGCATACAC
TATTCTCAGAATGACTTGGTTGAGTATGACTTACCAGTACAGAAAAAGCATCTTCAGGGTAGCATGACAGTAAGAGAAATTTGACGTGGTCCGCAATACATGAGTATGCTGCGGCAACTTACTCTGACAACAGCTCGGAGGACCGAAGGAGCTAAC
CGCTTTTTTGCAACAACTGGGGATCATGTAACCTGCTTGTGCTTGGGAACCGAGCTGAATGAAGCATAACAAACGACGAGCTGACACCACGATGCTCGACGAATGGCAACAACTTGCGCAAACTATTAATGCGCAACTACTTACTCTAG
CTTCCGCGCAACAAATTAATGAGCTGGATGGAGGGCGGATAAAGTTGCAAGGCACTCTGCGCTCGGCCCTTTCGGCTGGCTGGTTTTGCTGATAAATCTGGAGCGGGTGGAGCTGGGCTCTGCGGTTATCATTTGACAGCTGGGGCGAGTGGTAAAT
CCCTCCGCTATCTGAGTTATCTACACGACGGGAGTCAGGCAACTATGGATGAGCAAAATAGACAGATCGTGAGTAAAGTGGCTCTGATTAAGCATGTGACGCAAGTTTACTCATATATATCTTTAGATGATTTAAATCTTCAATTT
TTAATTTAAAGGATCTAGGTGAAGATCCTTTTTGATAATCTCATGACCAAAATCCCTTAACGTGAGTTTTCGTTCCACTGAGCGTCAGACCCGTAAGAAAGATCAAAGGATCTCTTGTAGATCCTTTTTTCTGCGCGTAATCTGCTGCTTGA
CAAAAAAACCCGCTACACAGCGGTGGTTGTTTGGCGGATCAAGAGCTACCAACTCTTTTTCGGAAGGTAACCTGGCTTCAGCAGAGCGCAGATACCAAACTACTGTTCTTCTAGTGTAGCCGTAGTTAGGCCACCACTTCAAGAACTCTGTAGCACCG
CCTACATACCTCGCTCTGTAATCTGTTACAGTGGCTGCTGCCAGTGGCGATAAGTCGTGTTTACCGGTTGGACTCAAGAGTATGTTACCGGTAAGGCGCAGCGGTGGGCTGAACGGGGGGTTCGTGCACACAGCCAGCTTGGAGCGAAC
GACCTACACCGCAACTGAGATACCTACAGCTGAGCTATGAGAAAGCGCCACGCTTCCCGAAGGGAGAAAGGCGGACAGTATTCGGTAAGCGGCAGGGTGGAAACAGGAGAGCGCACGAGGGAGCTTCCAGGGGGAACCGCTGGTATCTTTATAGTC
CTGTGGGTTTTGCGCACCTCTGACTTGAGCGTGCATTTTTGTGATGCTCGTCAGGGGGCGGAGCTATGAAAAACGCCAGCAACCGCGCTTTTTACGGTTCTTGCGCTTTTGTGCGCTTTTGTCTCATAGTGTCTTCTCTCGCTTATCCCTGAT
TCTGTGGATAACCGTATTACCGCTTTGAGTGAGCTGATACCGCTCGCGCGACCGCAAGCGACGAGCGAGCTCAGTGAGCGAAGGAGGAGGAGCGCCAAATACGCAAAACCGCTCTCCCGCGCGTGGCGGCTTCAATTAATGACAGCTGGCA
CGACAGATTTTTCCGACTGGAAGACGGGGCAGTGAGCGCAACGCAATTAATGTAAGTACTGACTCACTTAGGACGACCCAGGGTTAGACTTTGAGTTTGTGTTGAGTGTGAGCGGTAACAAATTCACACAGGAACAGCTATG
ACCATGATTACGCCAAGCTCTAGCTAGAGGTGCACCAATTTCTCATGTTGACAGCTTATCATCGAGATCCGGGCAACGTTGTTGCCATTGCTGCAGGCGCAGAAGTGGTAGGTATGGAAGATCTATACATTGAATCAATATTGGCAATTAGCCATAT
TAGTCAATGGTTATATAGCATAAACTCAATTTGGCTATTGGCCATTGACGATATGCTATCTATATCAATATGATCAATTTATTTGGCTCATGTCCAATAGCCGCATGTGACATTTGATTATGACTAGTATTAAATGATCAATATTACGGGG
TCATTAGTTTACGCCATATGTTGGAATTTCCGGCTTACATAACTTCAGGTAAGTTGGCCGCTGCTGCTGACGCCCAACGACCCCGCCCTTACGCTCAATAGTAAAGTATGTTTCCCATAGTAAAGCCCAATAGGGAATTTCCATTGACGTCAATGGGT
GGAGTATTACGGTAAACTGCCCACTTGGCAGTACATCAAGTGTATCATATGCAAGTCCGCCCTTATGACGTCAATGACGGTAAATGGCCCGCTGGCATTATGCCAGTACATGACCTTACGGGACTTTCCTACTTGGCAGTACATCTACGTAT
TAGTCAATGCTATTACCATGGTGTGCGGTTTTGGCAGTACACCAATGGGCGTGGATAGCGGTTTACTCACGGGGATTTCAAGTCTCCACCCATTGACGTCAATGGGAGTTGTTTGGCACAAAATCAAGGGGACTTTCAAAATGTGCTAAT
AACCCCCGCCGTTGACGCAAAATGGGCGGTAGGCGGTACGGTGGGAGGCTATATAAGCAGAGCTCGTTTGTAGTGAACCTGCATCTCTCTCCGATCGCTGCTGCGAGGGCGAGCTGTTGGGCTCGCGGTTGAGGACAACTCTTCGGG
GTCTTTCCAGTACTCTTGGATCGGAACCCGTCGCGCTCCGAACGGTACTCCGCCACGAGGACCTGAGCAGTCCGATCGACCGGATCGGAAACCTCTCGAGAAAGGGCGTCAACCACTCAGATCGAAGTAGGCTGAGACAGCTGGCGGGG
GGCAGCGGGTGGCGGTGGGGTTGTTTCTGCGGAGGTGCTGCTGATGATGAATTAAGTAGGCGGTCTTGAGCAGCGCGGATGGTGGAGGTGAGGTGTGGCAGGCTTGAGATCCAGCTGTTGGGGTGAAGTACTCCTCTCAAAGCGGGCATTACTT
CTGCGCTAAGATGTGAGTTTCAAAAAAGGAGGAGGATTTGATATTACCTTGGCGCGCATCTGGCCATACACTTGAATGACATGACATCCATTTGCCTTCTCTCCACAGGTGTCCACTCCAGGTCAAGTTTAAACTGCGGCCGCCACCATGGTC
TGCTCCTGACCCCGGAACCTGTCTGTACTTCTAGCCCTGCTGTTCTGAGCAGCACTGTGCTGTTGTGGCACCCGGGACCACTGTGATCTGGCAGCAGAGATTCGCGAGCTACTGCCCAACCACTGTGGGATCGCCGACTTCTT
GAGCACCTATCAGACTAAGTGCACAAGGATCTGACAGGCTGGAAGATATCTTGCACAGGTGGAGAACAGACAGCAGGAAGTGAAGCAGCTGATCAAGGCCATCCAGCTGACCTACAACCCGACGAGAGCAGCAAGCCCAACATGATCGACGCG
CCACCTGAAGTCCCGGAAGATGCTGGAAGAGATGTAAGTACGAGGCGAGCATCTGACCCACGACAGCAGCATCTCGGATATCTGAGGAATCTTACAAACAGCAACAGGAAATCGTGAACCTGAAAGAAAGGTGGCCGAGCTGGAAGGCCAG
TGCCAGGAACCTTGCAAGGACACCGTGCAGATCTCACGACATCACCGGCAAGGACTGCGAGGATATCGCAACAAGGGCGCAAGCAGAGCGGCTGTACTTCAACGCCCTGGAAGGCCAACACGACAGTTCCTGGTGTACTGCGAGATCAGCGGAC
CGGCAACGGCTGGACCGTGTTCAGAAGCGGCTGGACGGCAGCTGGACTTCAAGAAGAATTTGGATTCAGTACAAAGAGGGCTTGGCCACCTGAGCCCTACGGGACCAACCGAGTTCTGGCTGGGCAACGAGAAGATCCACCTGATCAGCACCCAGA
GCGCATCCTTTACGCCGTCGCGGTGGAGCTGGAAGATTGGAACGGCGGACAGCAGCCGCGACTACGCCATGTTCAAAGTGGGACTGAAGCGGATAGTACCGGCTGACCTACGCCACTTTGCCGCGGAGATGCGCGGACGCTTGCAGCGG
TTCGACTCTCGGCGACGCCAGCGACAAGTTCTTACCAGGCCAACGGCATGCAAGTTCAGCAGCTGGGACAACGACAACGATAAGTTGAGGGGCACTGCGCGAGCAGGATGGCAGCGGTTGGTGTGATGAACAAGTGCACGCCGGGACCTGAA
CGCGGTGATACTACGAGGCGGCACTTACAGCAAGGCGACGCCCAACGCGTACGACAAAGGCACTTGGGCGCTTGGAAACCCGCTGGTACAGATGAAGAAACCACTGAAGATCATCCCTTTCAACAGACTGACCACTCGGCGAGGGCG
AGCAGCATCACCTGGGCGGAGCCAAACAGGTCCGGCTGAGCAGCTTCCGAGAGCAGAGTACGACAGCTGTACCCGAGGACGACCTGTGAGGCGCGCC

Trail of Discovery

Resistance of cancer cells to immune recognition and killing.

Lipinski B¹, Egyud LG.

Author information

Abstract


It is well recognized that, in order for a wound to heal, the fibrin clot must be eliminated by fibrinolytic enzymes. In certain instances, however, fibrin is ineffectively degraded or even not degraded. For example, in pregnancy, the placenta contains a layer of fibrin (Nitabuck's layer) which presents as 'self' to the immune system. Similar situations have been observed in many solid tumors. A hypothesis is presented according to which tumor cells can escape detection and attack by the immune system in most cancer patients. The tumor dons a 'coat' of the host's own protein on its cell surface. The coat is composed of fibrin and of a polymeric form of human serum albumin (HSA) which, by contrast to pure fibrin, is resistant to fibrinolytic degradation. Such a coated tumor appears as 'self' to the immune system, and thus is not detected as a tumor by the immune system (i.e. natural killer cells). When tumors are prepared for in vitro assays against drugs, they are routinely treated with proteolytic enzymes (e.g. pepsin, or chymotrypsin, etc.) which dissolve the protein coat, exposing the tumor cell surface to the drug. Thus, the in vivo existence of a coat on the tumor surface may explain why some drugs have little or no effect in vivo, while the same drugs are active in vitro.


Prior thought: Cancer evades phagocytosis
New thought: Cancer evades **fibrinolysis**


Applied Scientific Computing

[OUR SCIENCE](#) | [ABOUT](#) | [DEPARTMENTS](#) | [PARTNER WITH US](#) | [CAREERS](#) | [NEWS](#) | [FEEDBACK](#) | [DIRECTORY](#)

Search web or people...



 *a passion for discovery*





CONGRATULATIONS!

Brookhaven Lab - Mentored Students win awards at science competitions!




Six Brookhaven Lab-Mentored Students Garner Awards at Regeneron and Siemens Science Competitions

Six students who conducted their research under the direction of Brookhaven Lab mentors have been honored at two of the nation's top science competitions.


[Newsroom](#) | [Video & Streaming](#) | [Receive our weekly newsletter](#) | [Visiting the Lab](#)

Happening Now


[See all](#)



Turning Research Data into Scientific Discoveries



U.S. Department of Energy: Science for the People




Meet the Director: John Hill

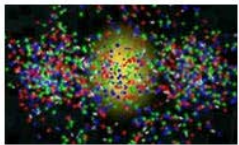
Our Mission

We advance fundamental research in nuclear and particle physics to gain a deeper understanding of matter, energy, space, and time; apply photon sciences and nanomaterials research to energy challenges of critical importance to the nation; and perform cross-disciplinary research on climate change, sustainable energy, and Earth's ecosystems.


Research Themes




Photon Sciences




QCD Matter



Energy Research



Physics of the Universe



Climate, Env. & Biosci.

[Research Programs Overview](#) | [Core Capabilities](#)

51

Applied Scientific Computing

Meet the Regeneron Scholars:

Finalist Emily Peterson: Smithtown High School East
Mentor: David Biersach, Information Technology Division
Title of Project: "Lecithin-Retinol Acyltransferase in Squamous Cell Carcinoma: The Relationship Between Oncology and Wound Repair"

Emily Peterson met Brookhaven Lab mentor David Biersach at a scientific computing seminar he was giving at her high school, where he learned of her research on skin cancer. Emily's research, conducted as a Simons Fellow under the guidance of Stony Brook University Professor Marcia Simon, focused on the possibility that a gene expression problem might inhibit the production of an enzyme responsible for strengthening cell walls. As cancer is invasive, skin cells with weak walls are more susceptible to becoming tumorous. Biersach showed Emily how a classic computer algorithm that looks for repeated substrings can be used in a novel way to determine if DNA sequences are likely to have important biological functions. This is accomplished by searching the human genome for other occurrences of these repeated sequences. They discovered that this enzyme's sequence also occurs in an enzyme involved in blood clotting. When blood clots form to heal a wound, the human body knows to leave the clot alone until the wound is fully repaired. After healing, a chemical signal triggers the body to break down the clot. The similarity in gene sequences that Peterson discovered suggests that cancer cells potentially use the same "don't bother me" signaling mechanism as blood clots, thus allowing the tumor to continue to grow in stealth mode. This collaboration is an excellent example of how students can apply skills in scientific computing directly to their research projects. Peterson hopes to continue her research by studying the enzyme's 3D atomic structure.



Finalist Emily Peterson [+ ENLARGE](#)

Applied Scientific Computing

**STUDENT SCIENCE**
A RESOURCE OF THE SOCIETY FOR SCIENCE & THE PUBLIC

Be a Champion for Science
Get your subscription to
Science News when you join.
[Join the Society](#)

Search Student Science... 

- Blogs and Resources
- Affiliated Fair Network
- Broadcom MASTERS
- Intel ISEF
- Regeneron STS
- Society Alumni



Regeneron STS 2017 Finalists

Congratulations to the 40 Regeneron Science Talent Search Finalists!

The Regeneron Science Talent Search (Regeneron STS) is the nation's most prestigious pre-college science competition. Science Talent Search alumni have made extraordinary contributions to science and hold more than 100 of the world's most coveted science and math honors, including the Nobel Prize and the National Medal of Science.

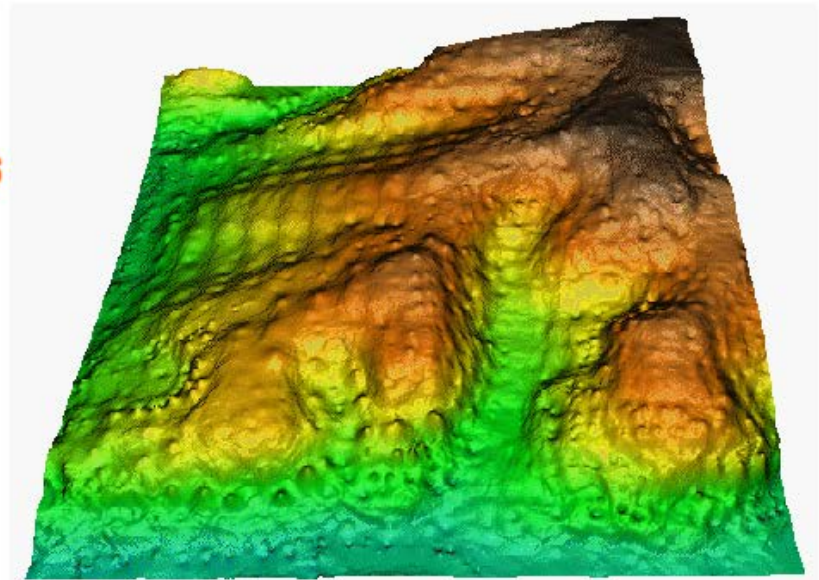
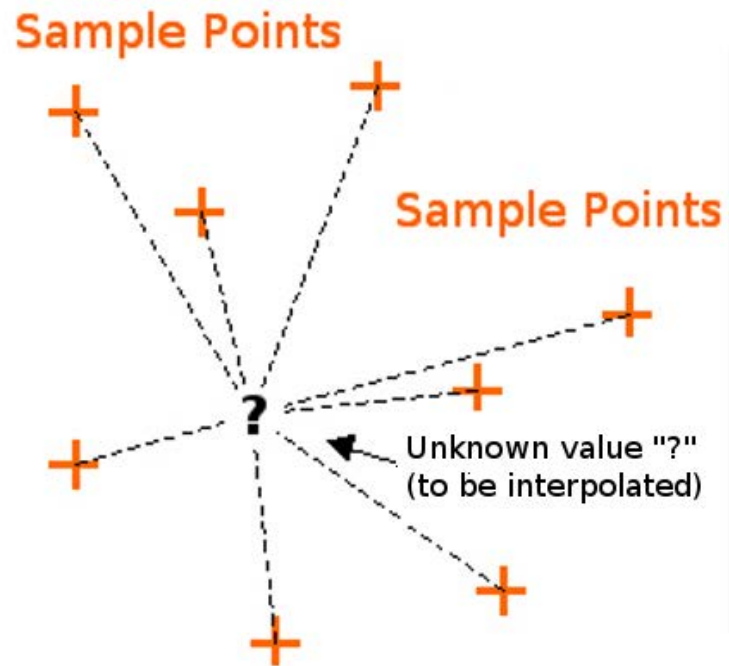
Trail of Discovery

- By considering not just the longest repeated substring, but also the five most (and five **least**) frequent substrings, Emily determined LRAT and FGK share common encodings
- FGK signals your body to **remove blood clots** after any damage is healed – it disposes healthy cells that are no longer needed
- Perhaps **cancer tricks your immune system** into thinking the cancerous cells are healthy and should not be disposed
- It may not be that the lack of LRAT results in a loss of cell membrane integrity (hence a greater chance of cancer spreading) – but there could be a link between something in LRAT and enabling cancer cell **“stealth mode”**

Session Goals

- Understand methods for **interpolating multi-dimensional data** taken from random sample locations
- Analyze the mathematics of the **Inverse Distance Weighting (IDW)** method
- Convert non-uniformly measured spatial data to a **regular conforming mesh**
- Develop approaches for estimating “**goodness of fit**” for predicted interpolated data points
- Use the **Root Mean Square Deviation (RMSD)** statistic as a measure of model accuracy

Interpolating Spatial Data



Interpolating Spatial Data

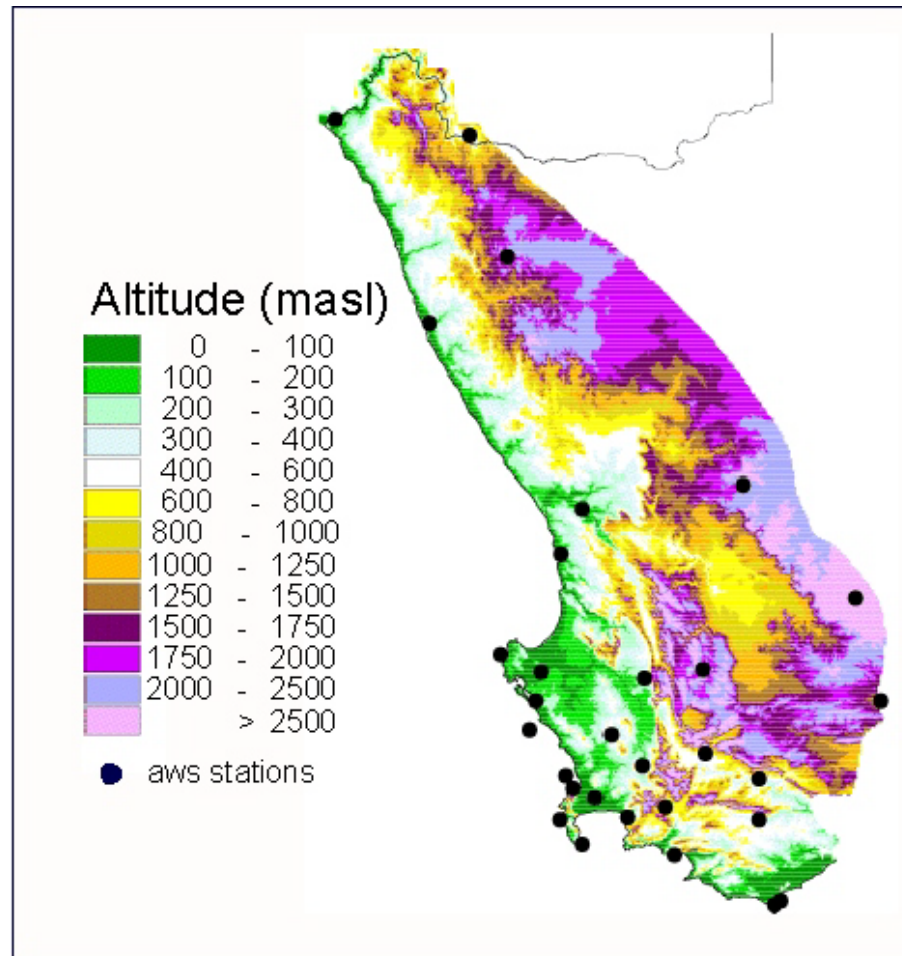
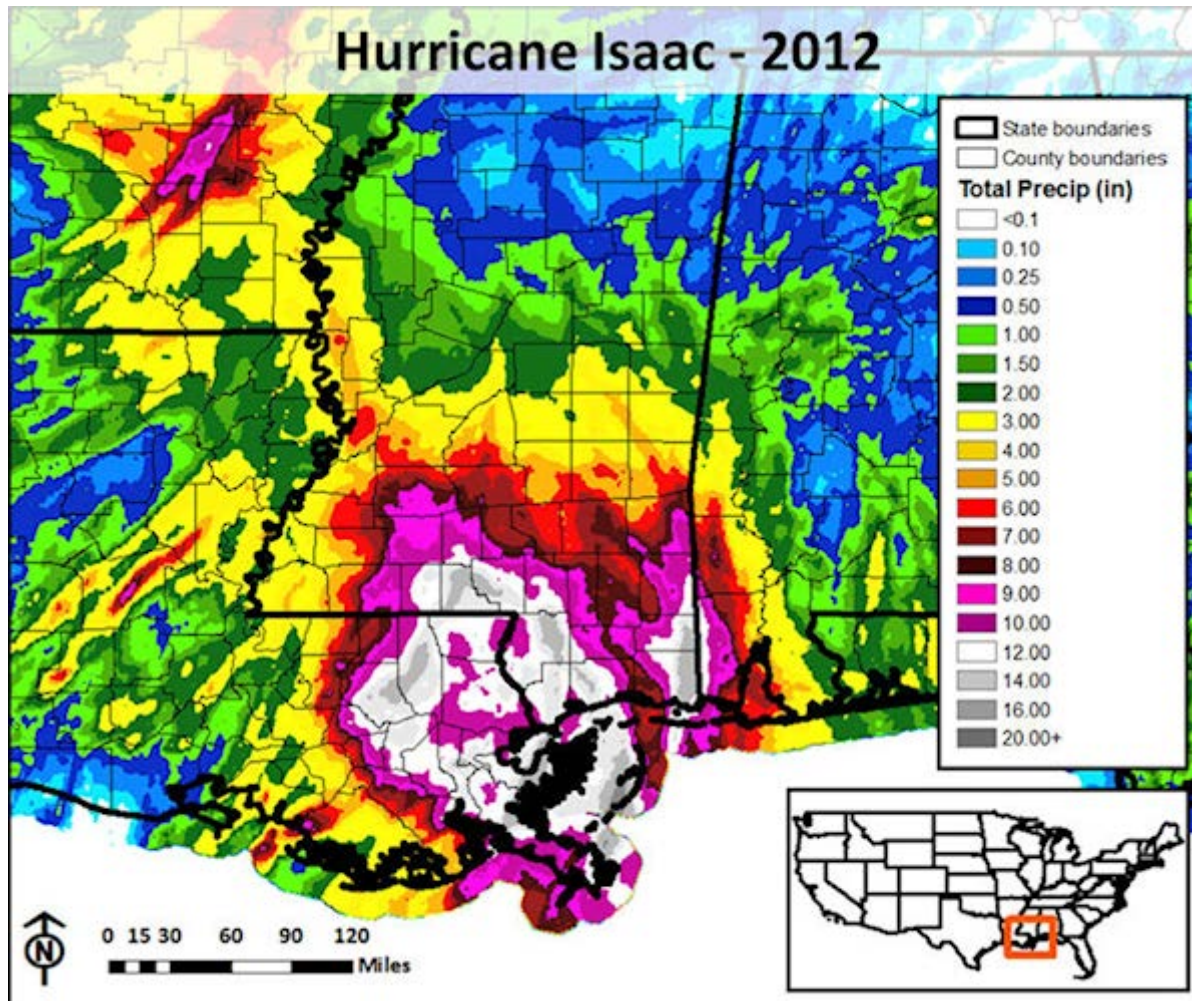
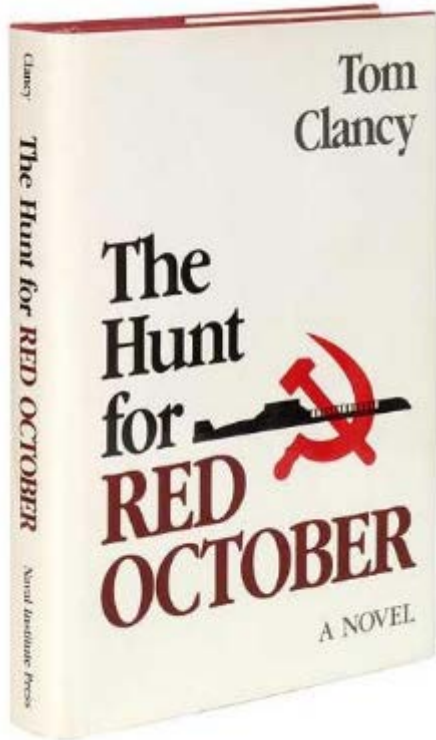


Figure 6 Altitude (200x200m) over the winter rainfall region of South Africa (after Directorate of Land Surveys and Information, 1996)

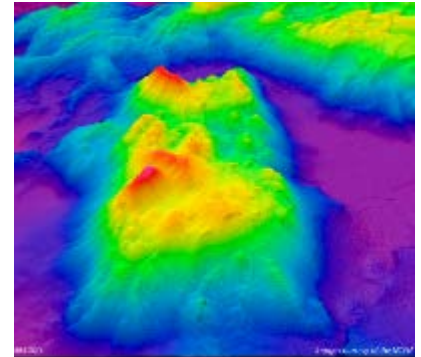
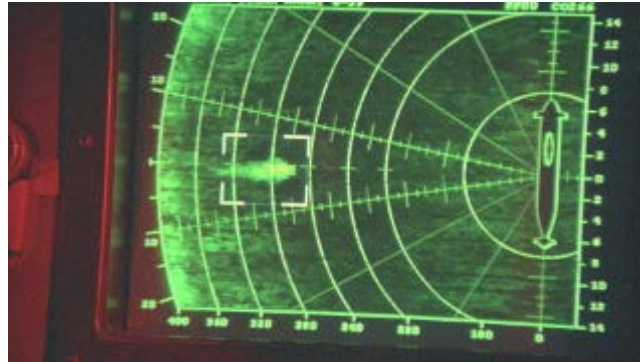
Interpolating Spatial Data



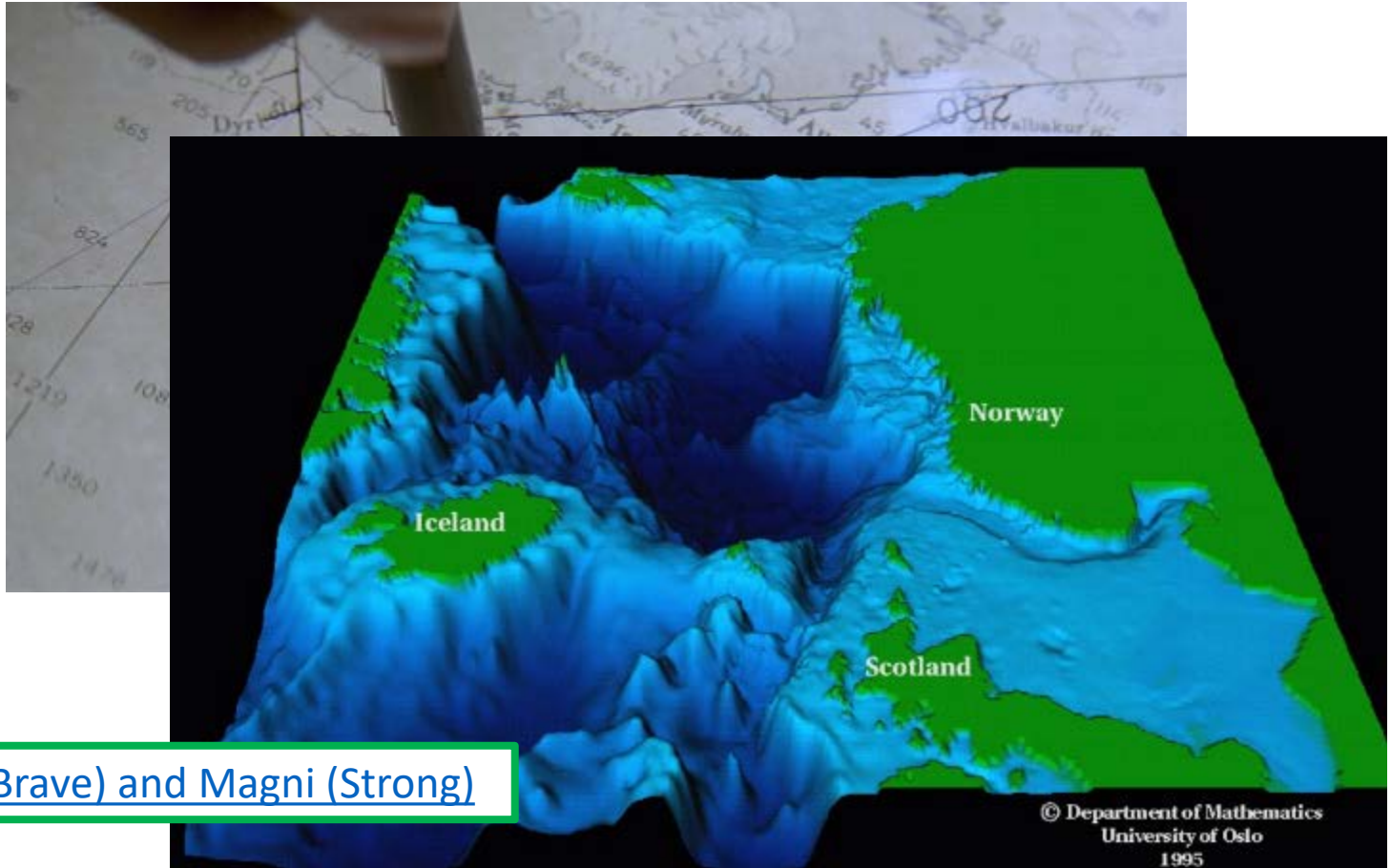
Interpolating Spatial Data



[Jonsey Reports](#)



Interpolating Spatial Data

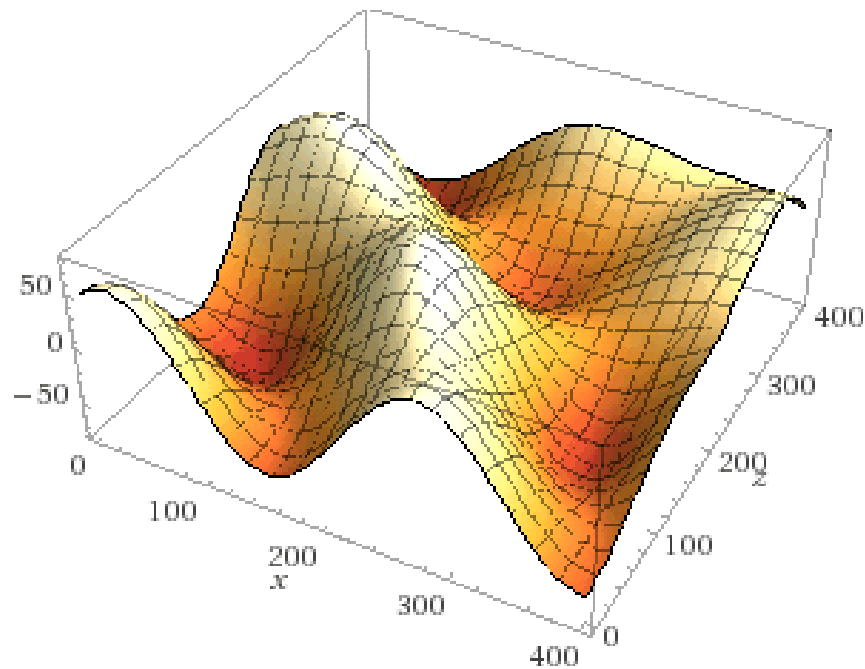


Modi (Brave) and Magni (Strong)

An “Actual” Ocean Floor

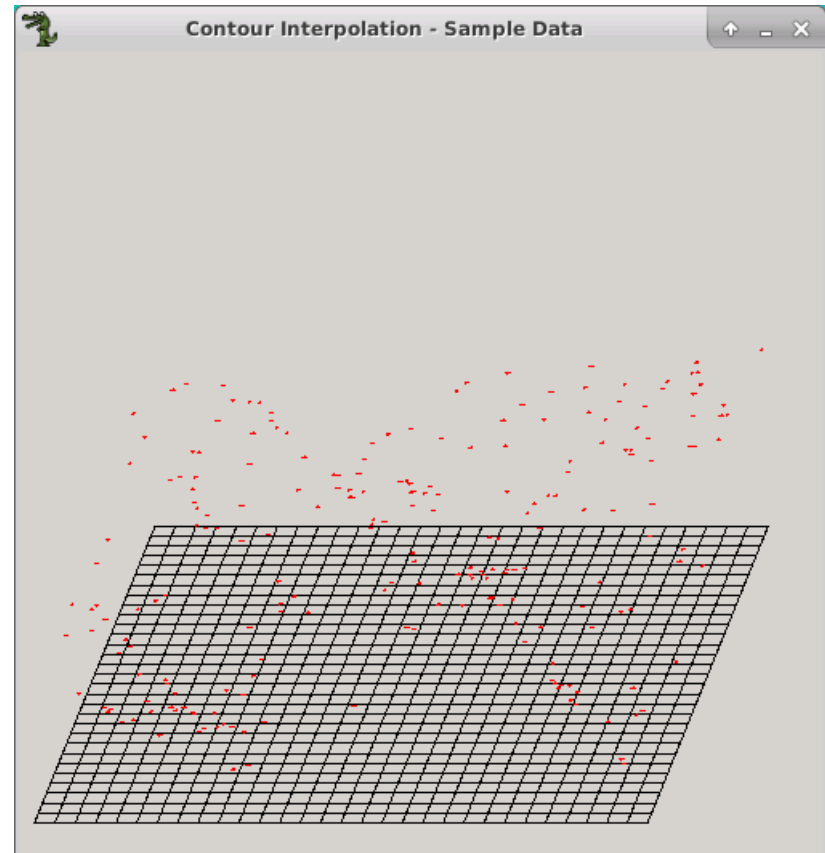
$$y = 30 \sin\left(\frac{x}{4}\right) \cos\left(\frac{z}{4}\right) + 50 \cos\left(\frac{\sqrt{x^2 + z^2}}{4}\right)$$

3D plot:



Sample Ocean Depth Soundings

- Ocean area is **400** units square, partitioned into grid of **30 x 30** intervals
- Depth samples were taken from **220 random** locations
- Floor reference grid has height $y = -80$
- Oblique projection



Sample Ocean Depth Soundings

```
void InitSamples()
{
    seed_seq seed{ 2017 };
    default_random_engine prng{ seed };
    uniform_int_distribution<int> dist{ 0, oceanSize };

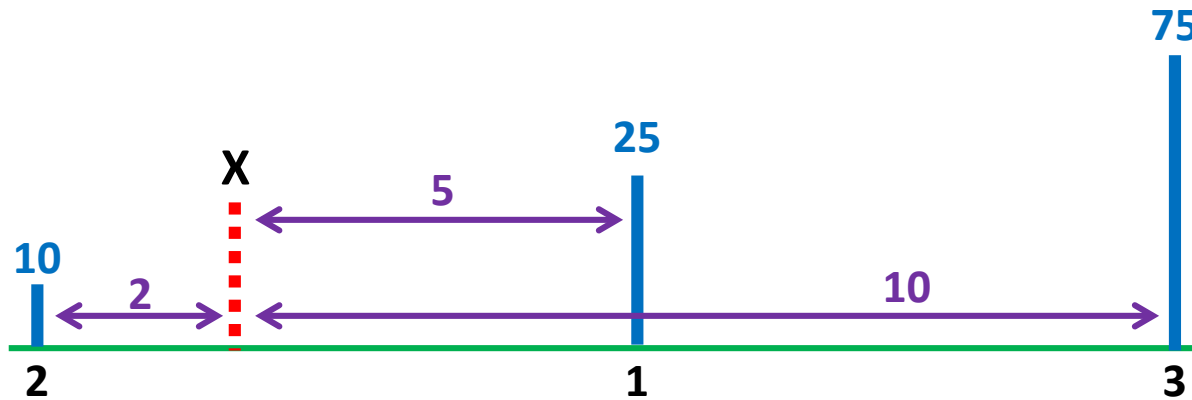
    // Generate random sample points
    for (size_t i{}; i < numSamples; ++i) {
        samples[i].x = dist(prng);
        samples[i].z = -dist(prng);
        samples[i].y = GetActHeight(samples[i].x, samples[i].z);
    }

    // Create a small marker at each sample point (a horizontal facet)
    for (size_t i{}; i < numSamples; ++i)
    {
        size_t v0 = vSamples.add(samples[i].x, samples[i].y, samples[i].z + 2);
        size_t v1 = vSamples.add(samples[i].x + 2, samples[i].y, samples[i].z);
        size_t v2 = vSamples.add(samples[i].x, samples[i].y, samples[i].z - 2);
        size_t v3 = vSamples.add(samples[i].x - 2, samples[i].y, samples[i].z);
        fSamples.add(&vSamples, { v0, v1, v2, v3 });
    }
}
```

The Inverse Distance Weighting (IDW) Method

- IDW is a type of deterministic method for **multivariate interpolation** of a set of *scattered* points
- The value assigned to *unknown* points are calculated from a **weighted average** of the values at the *known* points
- The theory is the farther away a known point is from the unknown point, the **less** that known distant point can contribute to the unknown height
- *Closer* known points contribute *more* to the unknown height than known points that are farther away
- Your contribution **is inverse** to your distance

The Inverse Distance Weighting (IDW) Method



Sample Index

Distance (d) from Sample to X

Sample Known Height

1	2	3
5	2	10
25	10	75

Weight = $1/d^p$ (set $p = 1$)

Sample Height • Weight

Totals			
0.2	0.5	0.1	0.8
5	5	7.5	17.5

$$\left(\frac{17.5}{0.8} \right) \rightarrow \text{Height at X} = 21.875$$

The Inverse Distance Weighting (IDW) Method

A general form of finding an interpolated value u at a given point x based on samples $u_i = u(x_i)$ for $i = 1, 2, \dots, N$ using IDW is an interpolating function:

$$u(\mathbf{x}) = \begin{cases} \frac{\sum_{i=1}^N w_i(\mathbf{x}) u_i}{\sum_{i=1}^N w_i(\mathbf{x})}, & \text{if } d(\mathbf{x}, \mathbf{x}_i) \neq 0 \text{ for all } i \\ u_i, & \text{if } d(\mathbf{x}, \mathbf{x}_i) = 0 \text{ for some } i \end{cases}$$

where

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$$

**The p value
indicates the
"power" of
the distance
penalty**

The Inverse Distance Weighting (IDW) Method

$$u(\mathbf{x}) = \begin{cases} \frac{\sum_{i=1}^N w_i(\mathbf{x}) u_i}{\sum_{i=1}^N w_i(\mathbf{x})}, & \text{if } d(\mathbf{x}, \mathbf{x}_i) \neq 0 \text{ for all } i \\ u_i, & \text{if } d(\mathbf{x}, \mathbf{x}_i) = 0 \text{ for some } i \end{cases}$$

where

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x}, \mathbf{x}_i)^p}$$

```
double GetEstHeight(double x, double z)
{
    double sumWeight = 0;
    double sumHeightWeight = 0;

    for (size_t n{}; n < numSamples; ++n) {
        double distance = sqrt(pow(x - samples[n].x, 2)
                                + pow(z - samples[n].z, 2));

        if (distance == 0) return samples[n].y;

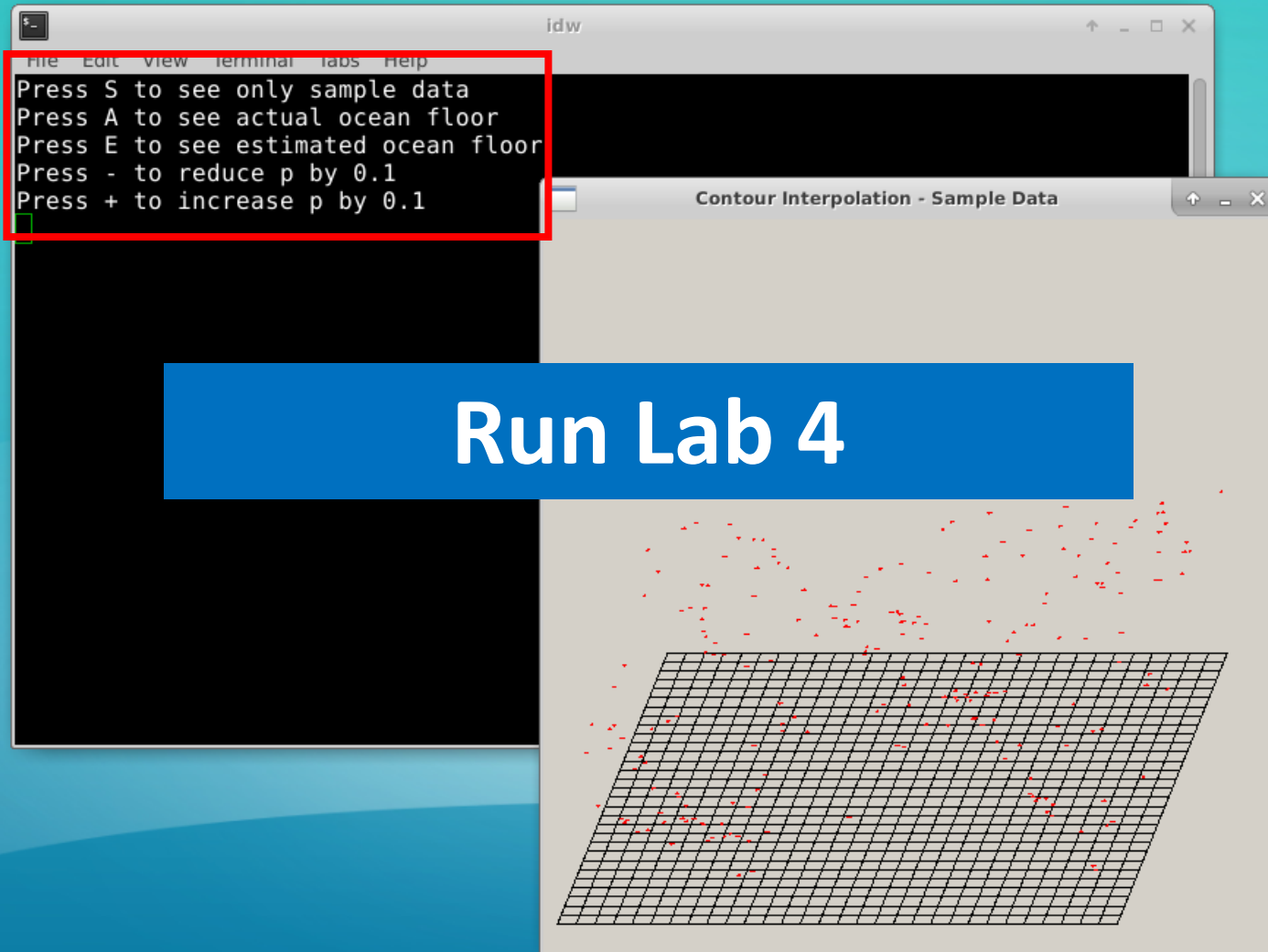
        double weight = 1 / pow(distance, p);

        sumWeight += weight;
        sumHeightWeight += samples[n].y * weight;
    }

    double height = sumHeightWeight / sumWeight;

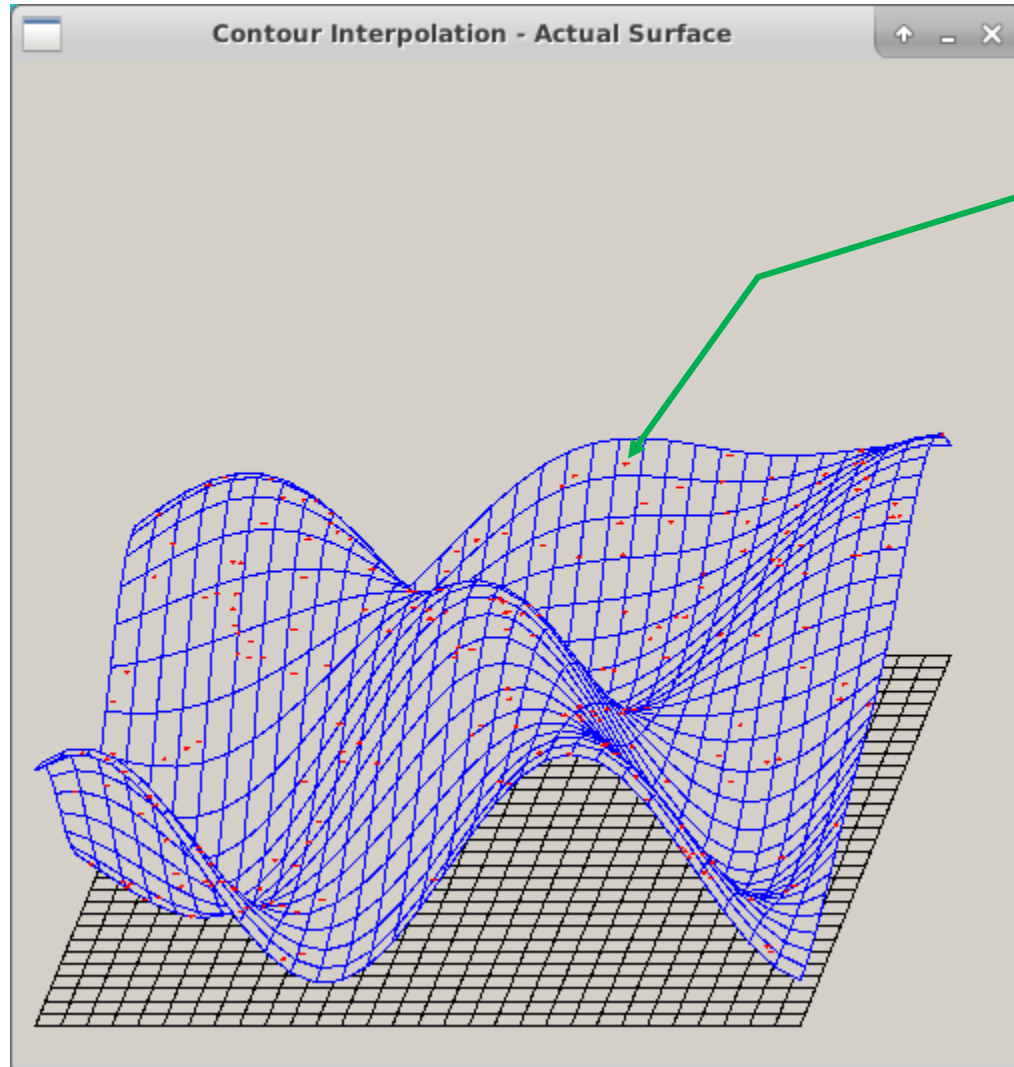
    return height;
}
```

Inverse Distance Weighting



Actual Ocean Floor

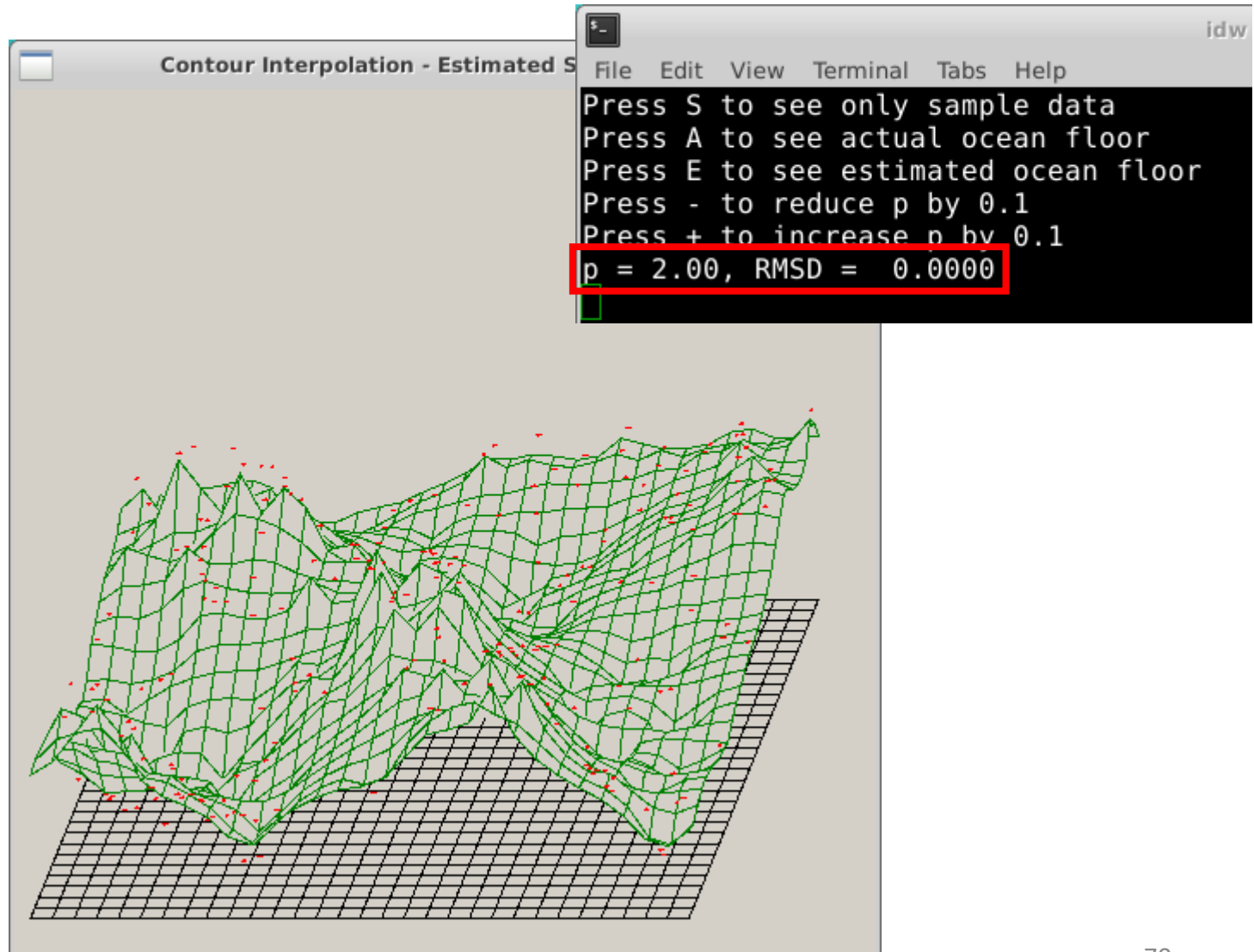
Press the **A**
key to see
the actual
ocean floor



The **red** "dots"
are the sample
data points

Estimated Ocean Floor

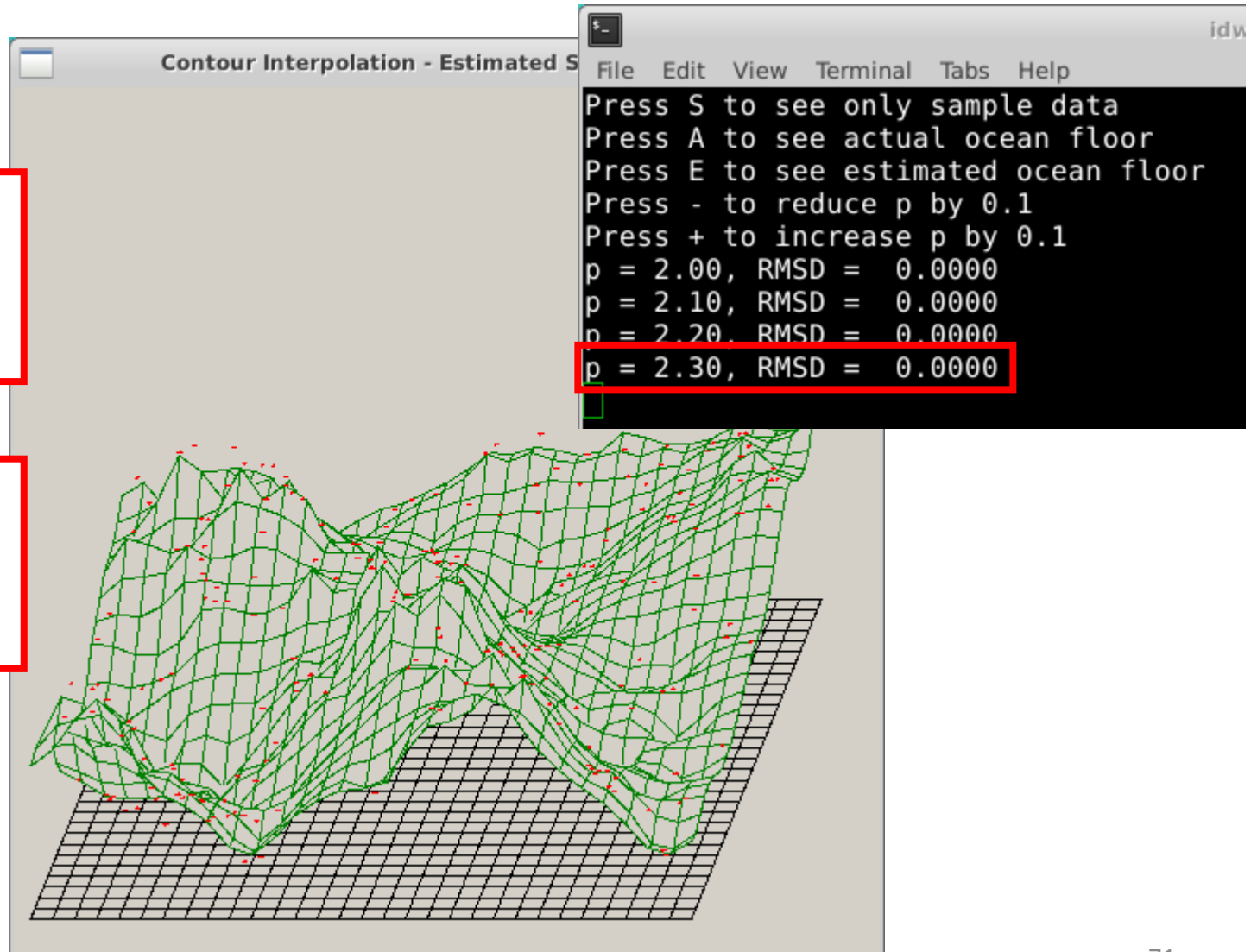
Press the **E**
key to see the
estimated
ocean floor



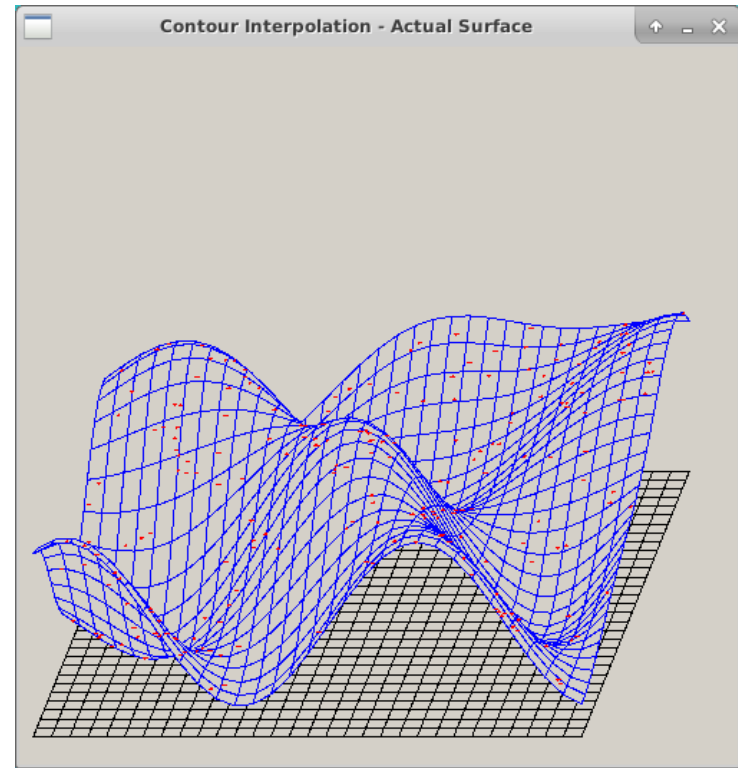
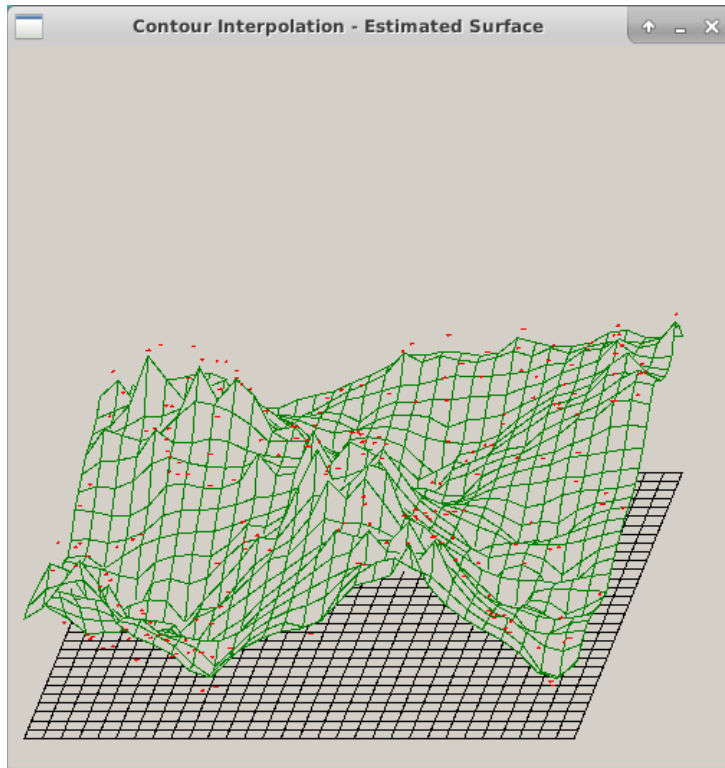
Estimated Ocean Floor

Press the **+** key
to increase p
(power) by 0.1

Press the **-** key
to decrease p
(power) by 0.1

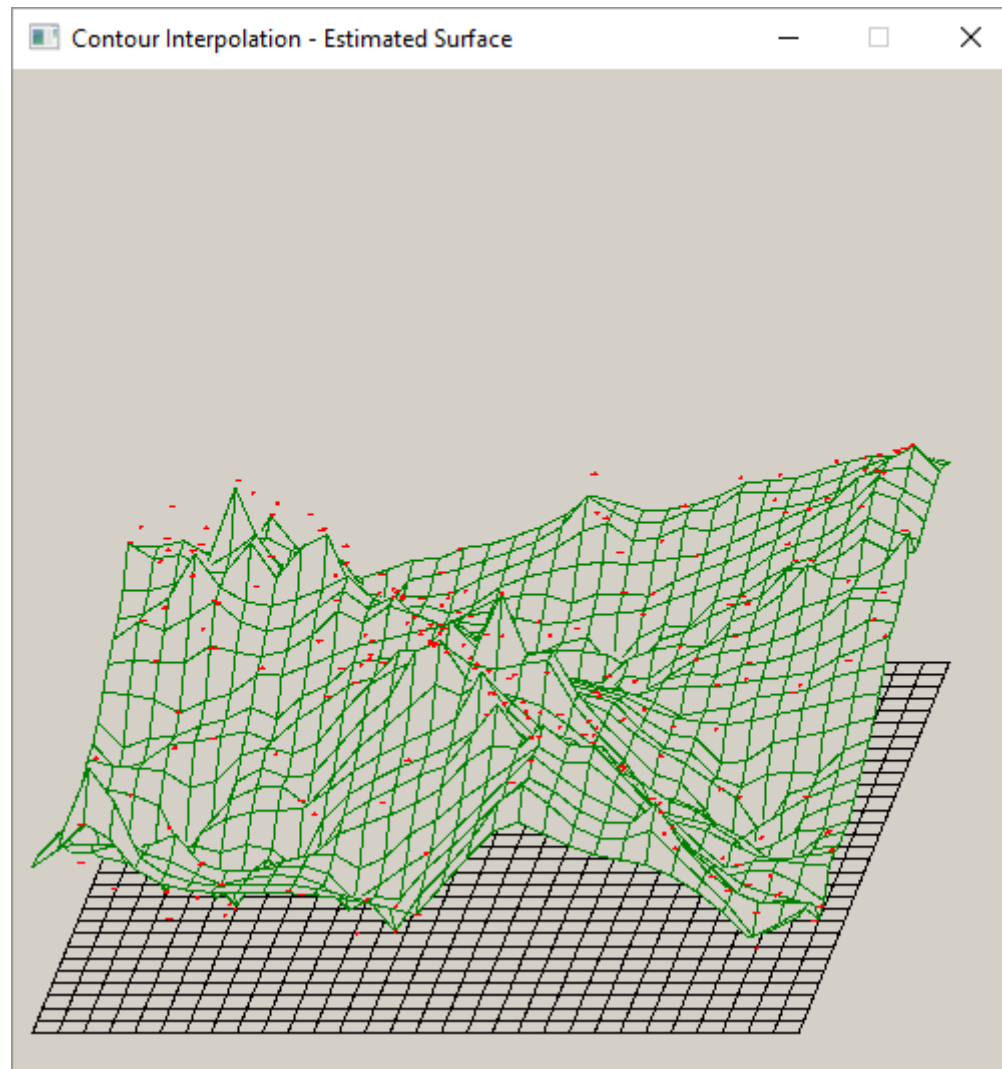


Estimated vs Actual ($p = 2.0$)

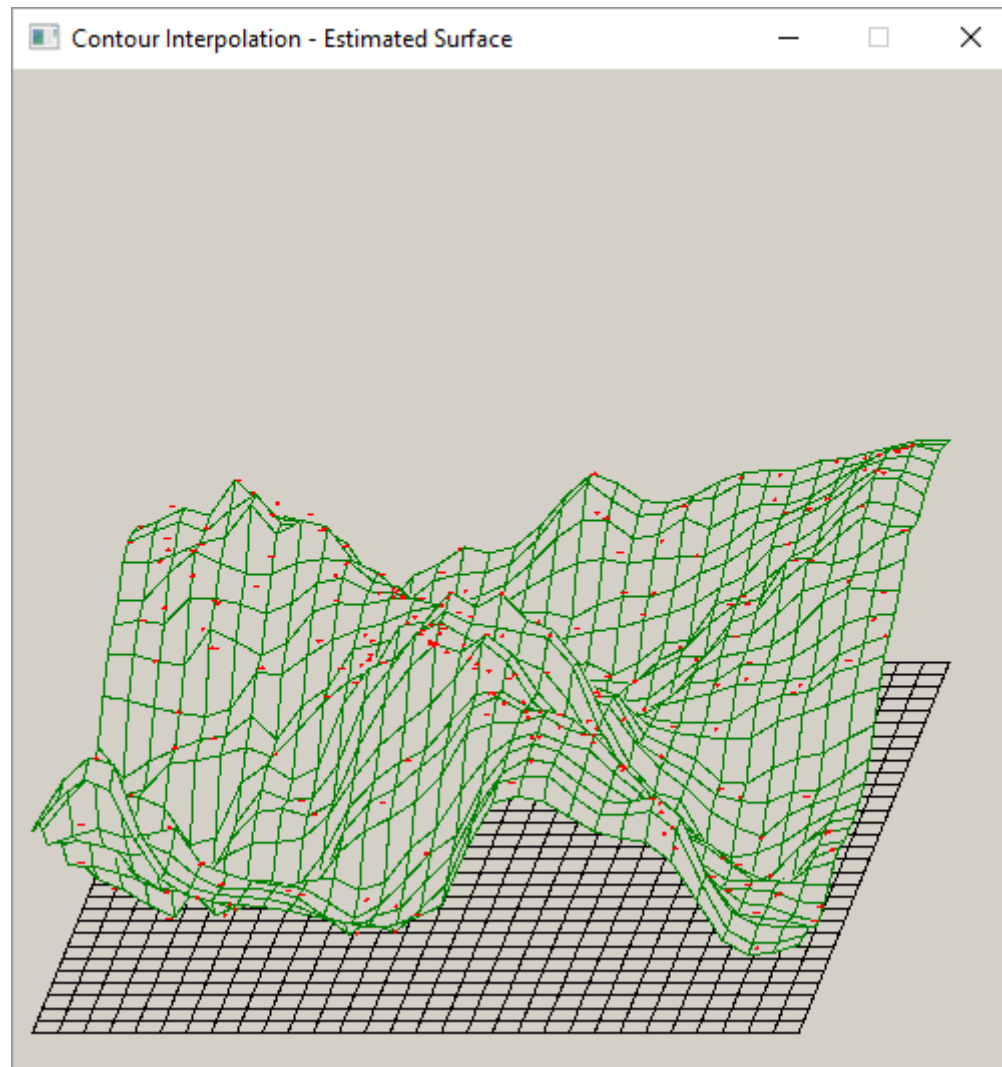


A first order approximation having only 24% of the world sampled (220 of 900 actual points)

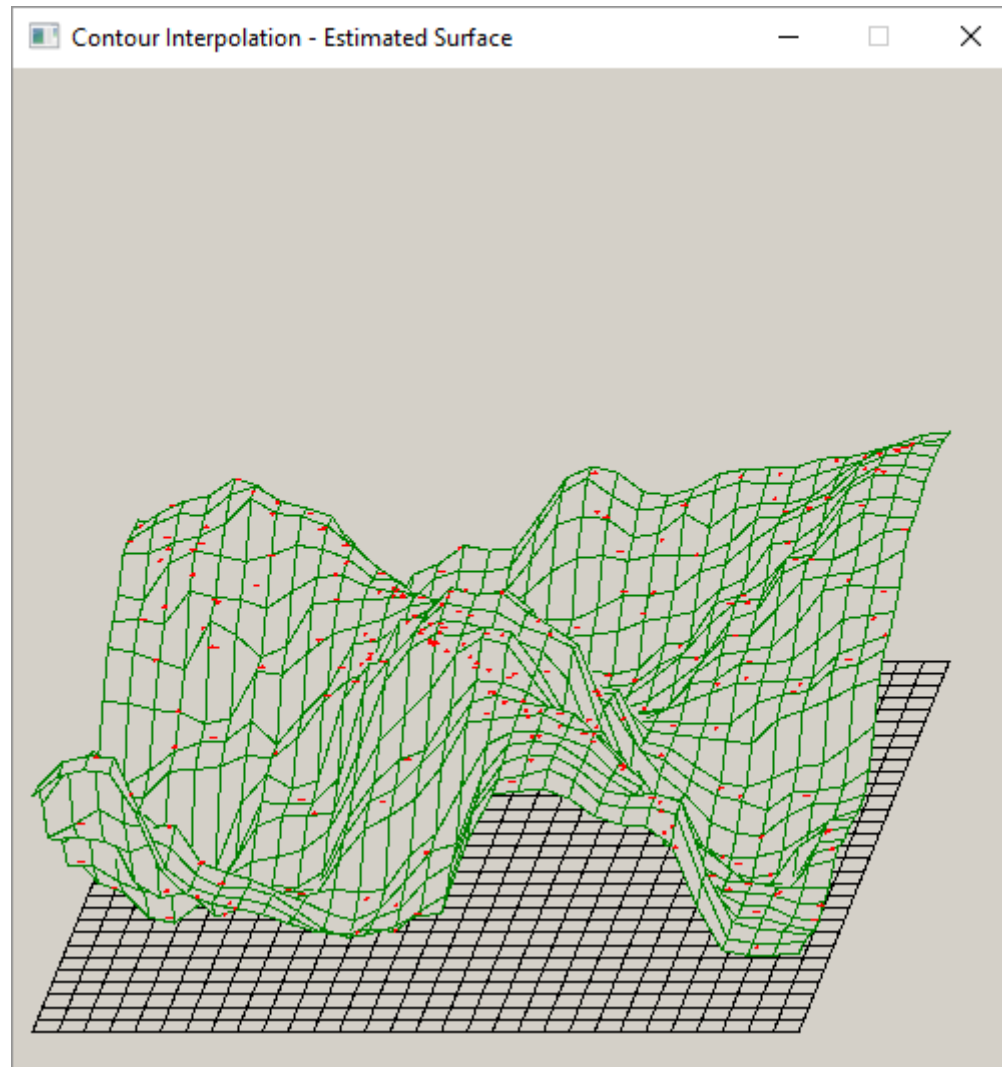
Estimate ($p = 2.0$)



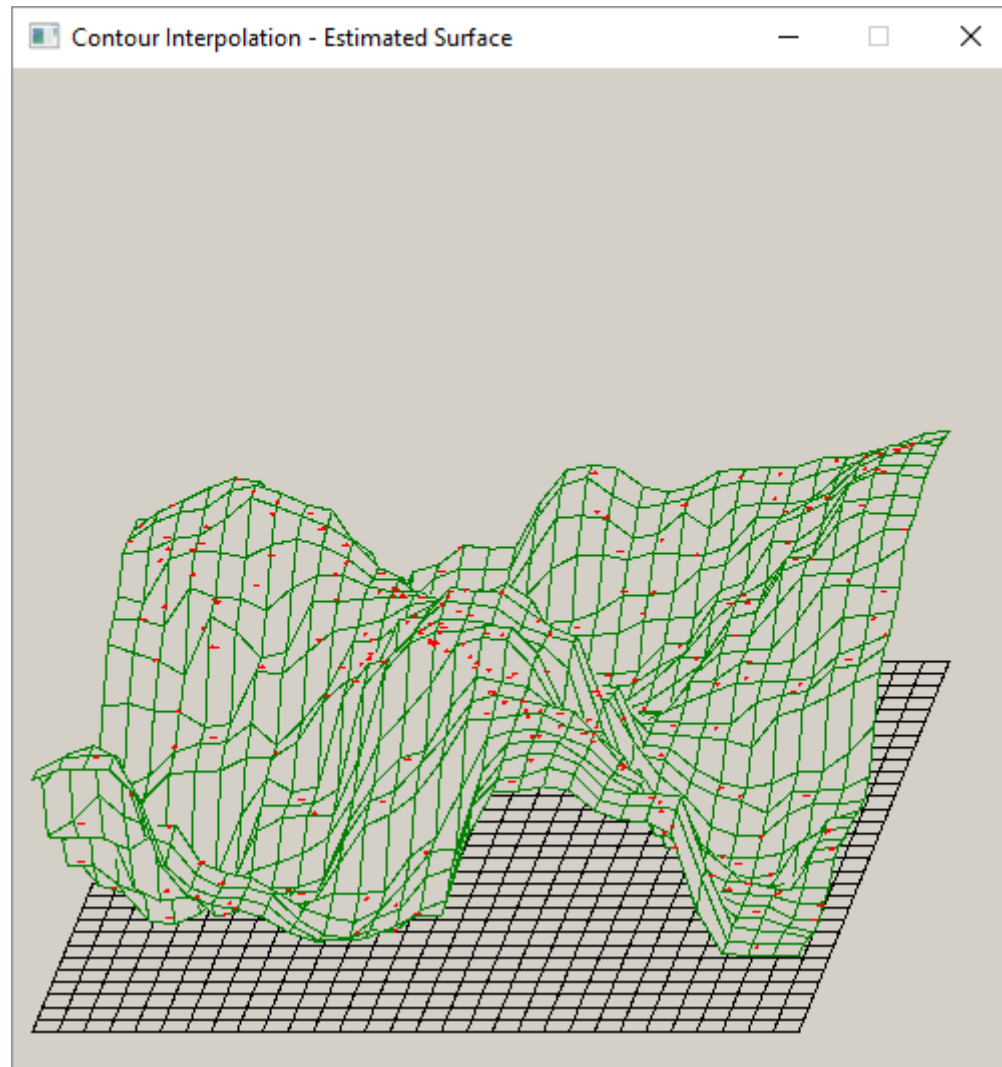
Estimate ($p = 3.0$)



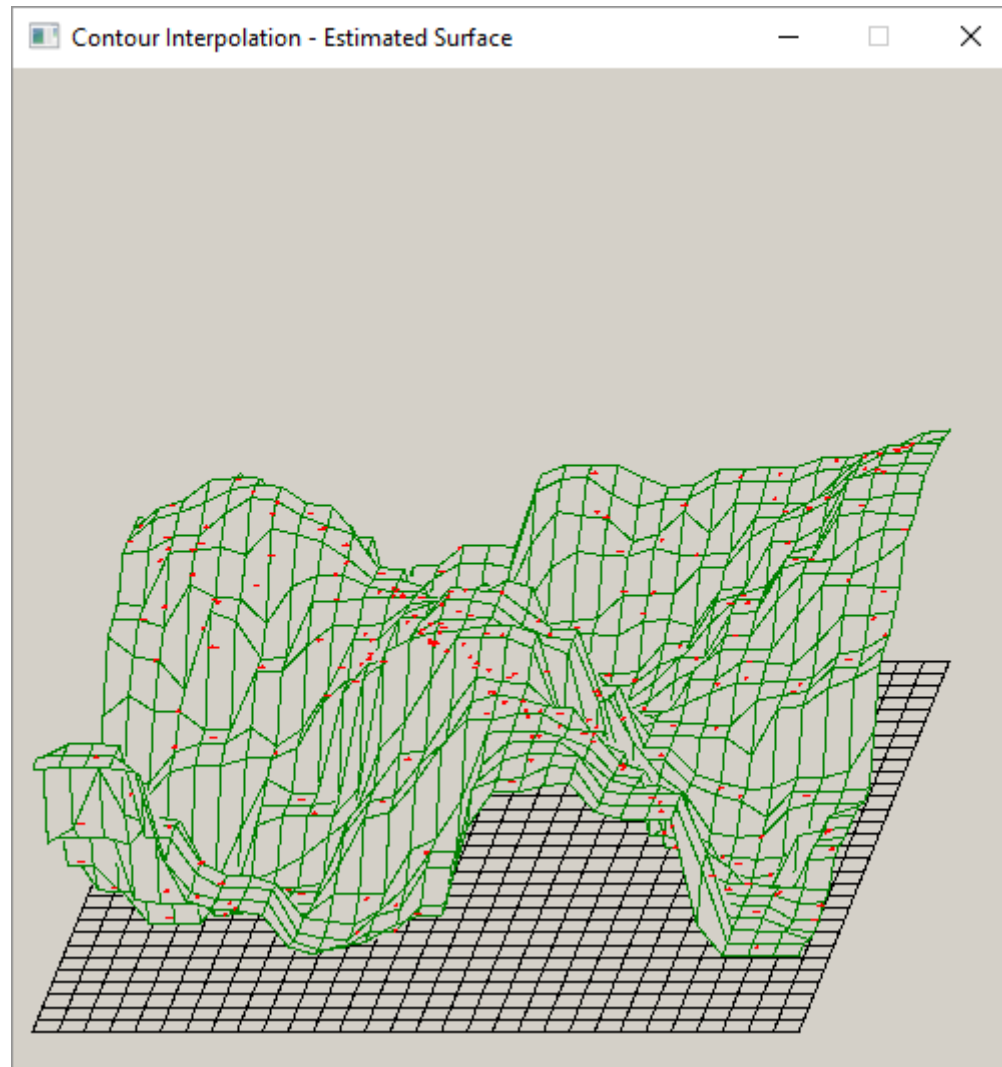
Estimate ($p = 4.0$)



Estimate ($p = 5.0$)



Estimate ($p = 9.0$)



Root Mean Square Deviation

- As we increase the **power** term **p**, is our model getting *better* or *worse* at predicting reality?
- The root-mean-square deviation (**RMSD**) is a statistic to measure the differences between values **predicted** by a model and the values actually **observed**

$$\text{RMSD} = \sqrt{\frac{\sum_{t=1}^n (\hat{y}_t - y)^2}{n}}.$$

- By averaging **the errors** (*actual* – *estimated*)² across **all sample points**, we calculate a comparative statistic that can help empirically determine the optimal **p** value because it will minimize the overall error of the model

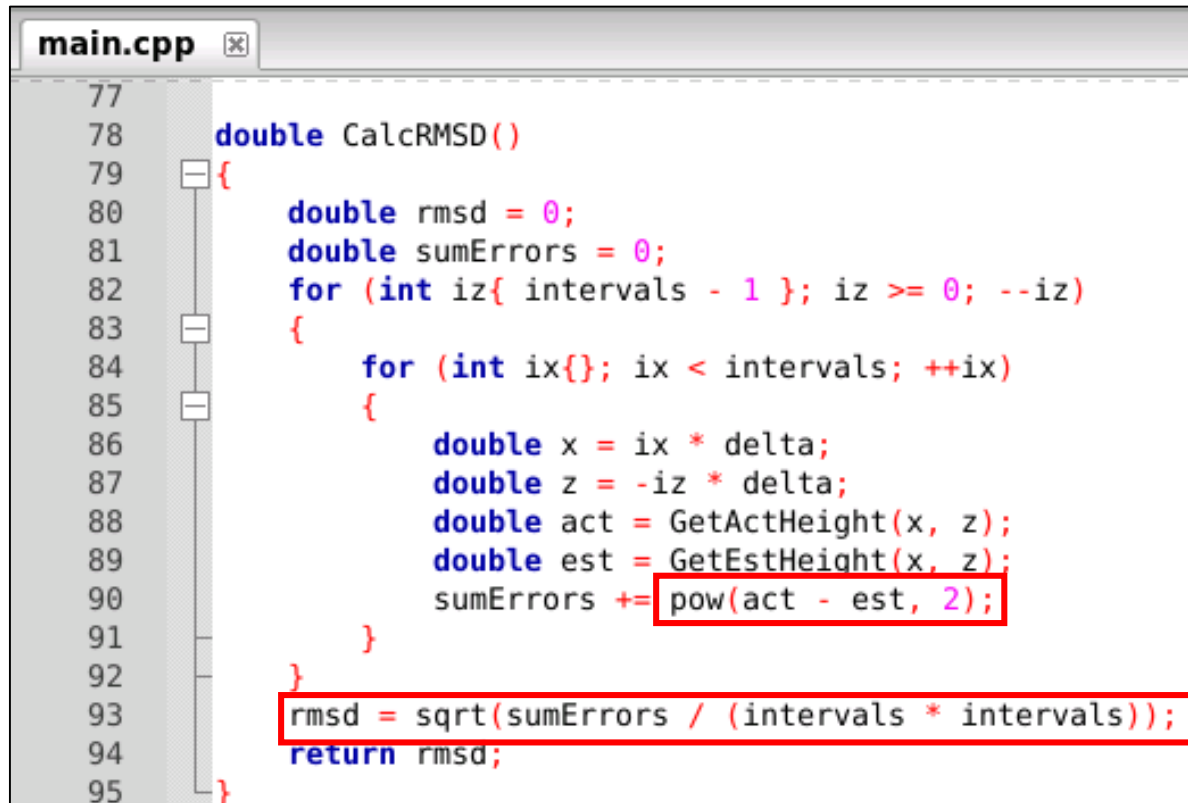
Edit Lab 4 - Calculate the **RMSD**

```
main.cpp
77
78 double CalcRMSD()
79 {
80     double rmsd = 0;
81     double sumErrors = 0;
82     for (int iz{ intervals - 1 }; iz >= 0; --iz)
83     {
84         for (int ix{}; ix < intervals; ++ix)
85         {
86             double x = ix * delta;
87             double z = -iz * delta;
88             double act = GetActHeight(x, z);
89             double est = GetEstHeight(x, z);
90             sumErrors += 0;
91         }
92     }
93     rmsd = 0;
94     return rmsd;
95 }
```

Fix the code to calculate **sumErrors** and **rmsd**



Edit Lab 4 - Calculate the RMSD

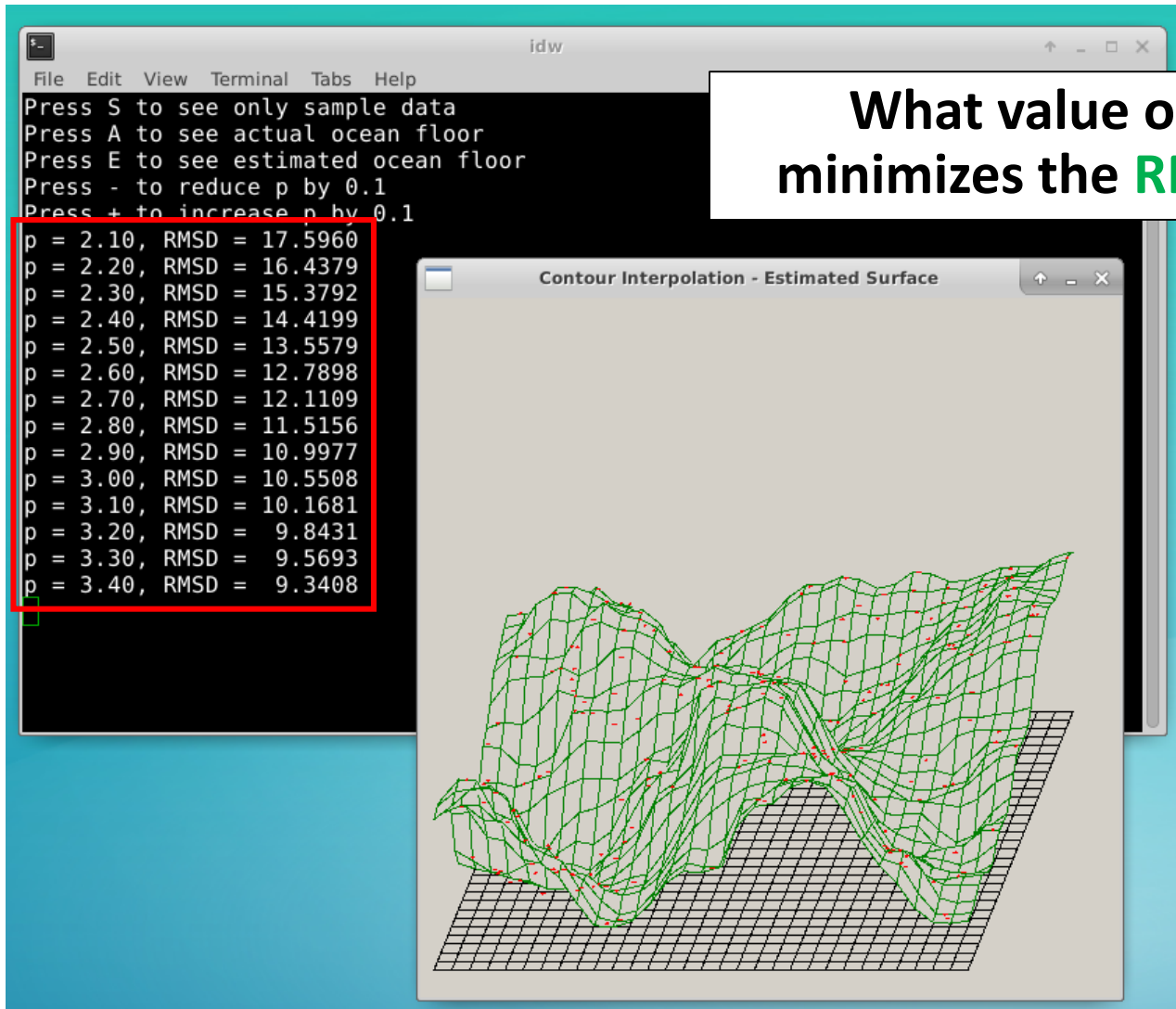


The screenshot shows a code editor window titled "main.cpp". The code defines a function `CalcRMSD()` that calculates the Root Mean Square Deviation (RMSD). The function iterates over a grid of points defined by `ix` and `iz` indices, comparing actual heights (`act`) with estimated heights (`est`). The squared differences are accumulated in `sumErrors`, and the final RMSD is calculated as the square root of the average squared error.

```
77
78 double CalcRMSD()
79 {
80     double rmsd = 0;
81     double sumErrors = 0;
82     for (int iz{ intervals - 1 }; iz >= 0; --iz)
83     {
84         for (int ix{}; ix < intervals; ++ix)
85         {
86             double x = ix * delta;
87             double z = -iz * delta;
88             double act = GetActHeight(x, z);
89             double est = GetEstHeight(x, z);
90             sumErrors += pow(act - est, 2);
91         }
92     }
93     rmsd = sqrt(sumErrors / (intervals * intervals));
94     return rmsd;
95 }
```

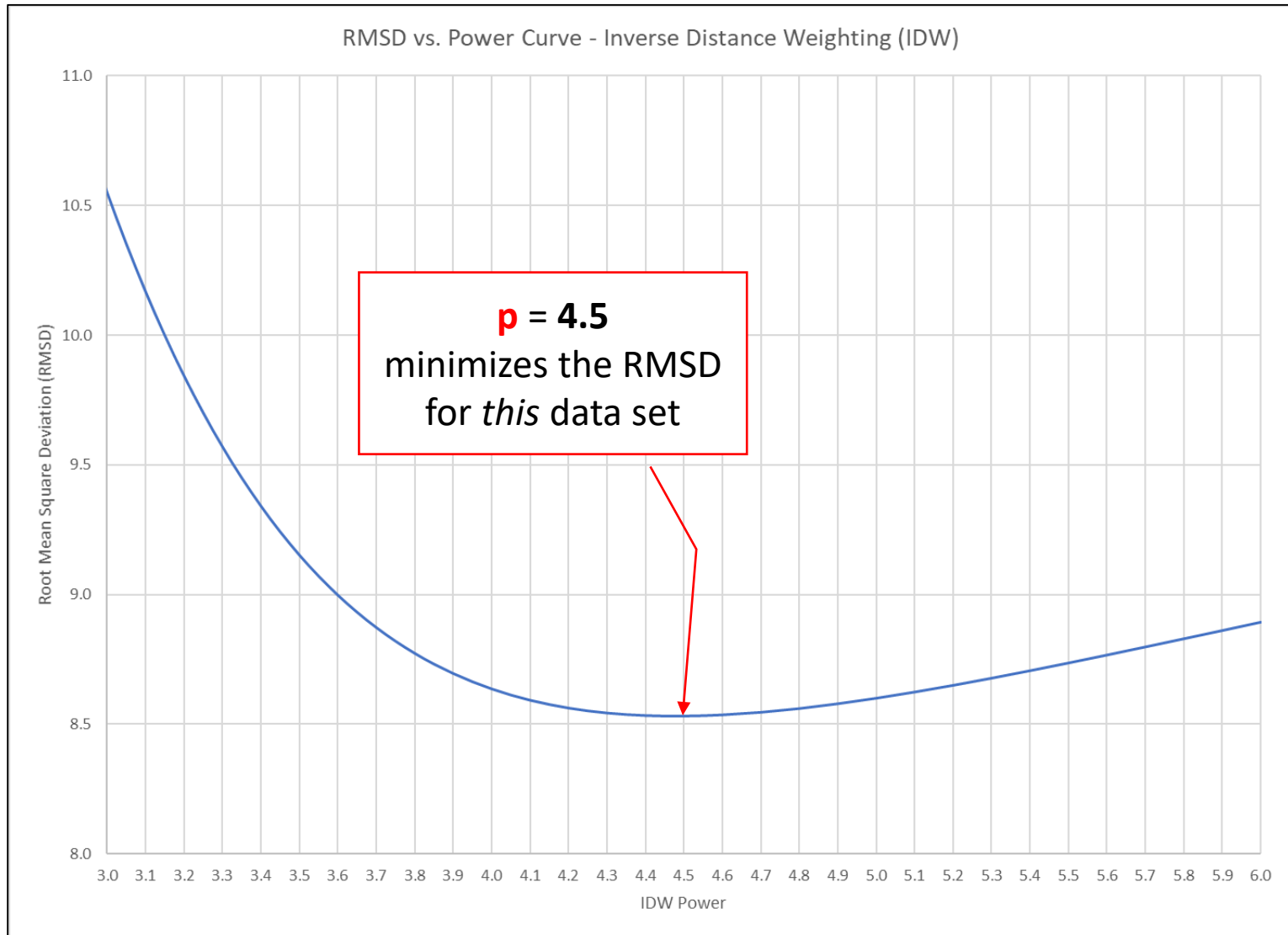
What value of **p** minimizes the RMSD?

Check Lab 4 - Calculate the RMSD



What value of **p** minimizes the **RMSD**?

Graphing the **RMSD**



Now you know...

- Identifying repeated sequences in DNA can lead to new understandings of genetic structure and purpose
 - **Suffix sort** is a very clever way to find the **longest repeating substring (LRSS)**
 - Sequence alignment tools (BLAST, etc.) look for which two sequences share a similar subsequence
 - Alternatively, LRSS looks inside a single sequence, to see what patterns might repeat ***within it***
- There could be biological significance of a **very long substring** that appears **only a few times** within the entire sequence

Now you know...

- The **Inverse Distance Weighting (IDW)** method can interpolate multi-dimensional data taken from random sample locations
 - How to convert non-uniformly measured spatial data into a **regular conforming mesh**
 - How to use **RMSD** as one metric to characterize the “goodness of fit” for predicted interpolated data points
- Scientists **rarely** enjoy the luxury of having too many data samples and must often interpolate to fill in the missing gaps