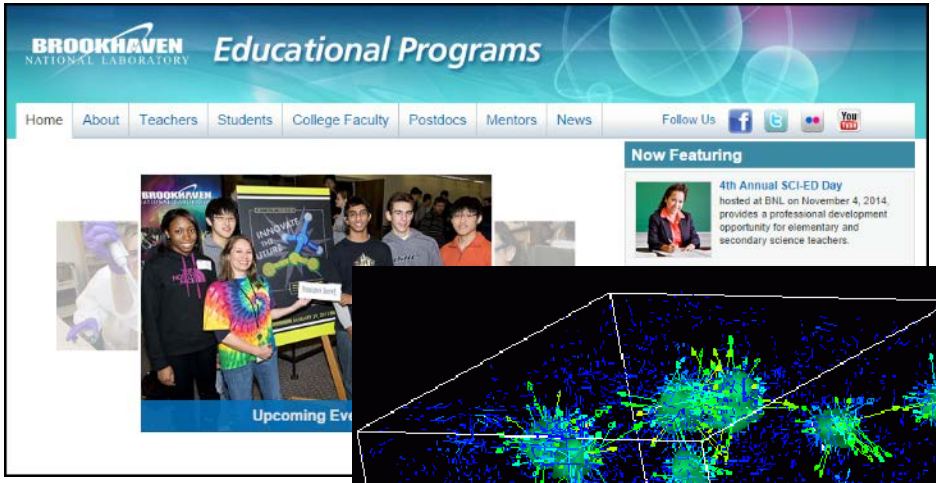
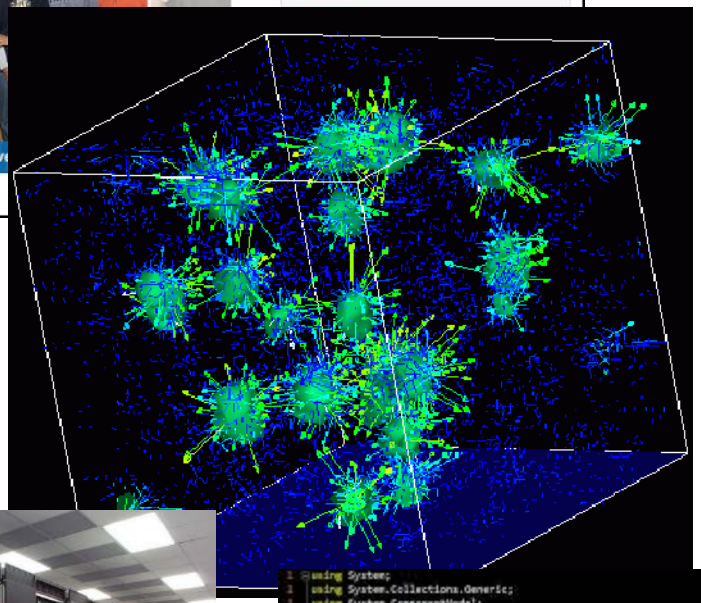
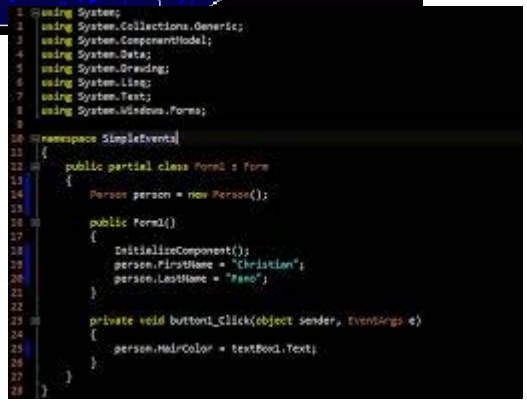






Survey of Scientific Computing (SciComp 301)

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

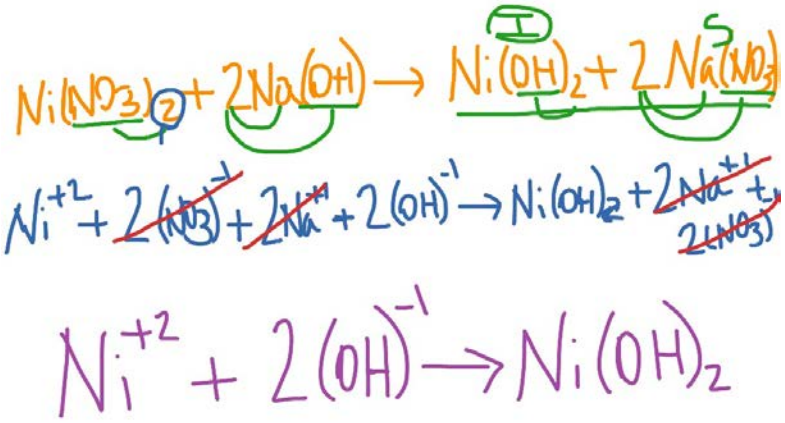
Session 17
Computational Chemistry, Clustering

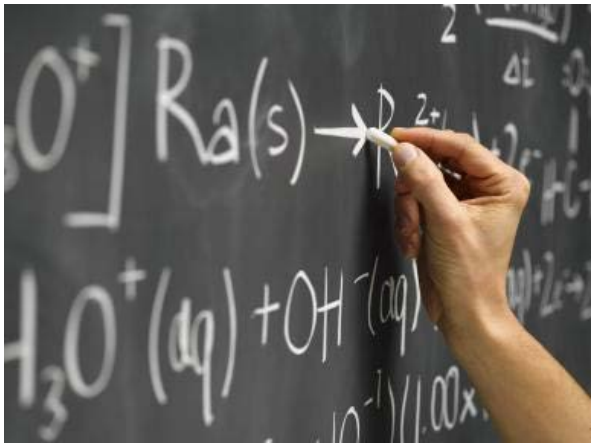
Session Goals

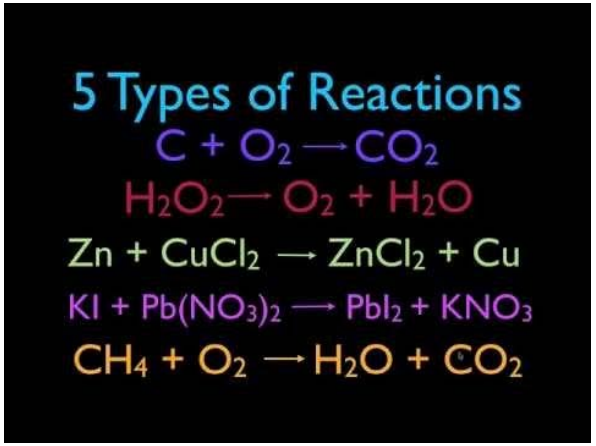
- Balance **ionic chemical equations** using linear programming
- Correlate “principle of atom conservation” (**POAC**) to the *minimization* of an objective function
- Represent molar & charge ratios as **linear constraints**
- Encode chemical equations as a matrix within a text file
- Implement **brute-force** searching using a **stack** data structure *instead* of using recursion or nested **for()** loops

Balancing Ionic Equations

- Can we write a program to balance ionic equations?







Balancing Ionic Equations

Step 1: Write reactants and products

$C_3H_8 + O_2 \rightarrow CO_2 + H_2O$

Step 2: Find one atom that occurs only in one substance on both sides

$C_3H_8 + O_2 \rightarrow CO_2 + H_2O$

Step 3: Find coeffi

Step ther unbalanced atom which occu only one substance

$1C_3H_8 + O_2 \rightarrow 3CO_2 + H_2O$

Step 5: Find coefficient to balanc s atom

$1C_3H_8 + O_2 \rightarrow$

Step 6: R

$\rightarrow 3CO_2 + 4H_2O$

$1C_3H_8 + 5O_2 \rightarrow 3CO_2 + 4H_2O$

Step 7: Make sure the coefficients are reduced to the smallest whole number values

$1C_3H_8 + 5O_2 \rightarrow 3CO_2 + 4H_2O$

Step 8: Check the balanced equation

$1C_3H_8 + 5O_2 \rightarrow 3CO_2 + 4H_2O$

3C, 8H, 10O 3C, 10O, 8H

Yikes - and this is not even ionic!

Chemistry is Optimization

- Consider a chemical formula as an **optimization** problem:

$$\mathrm{MnO_4^-} + \mathrm{H_2O_2} + \mathrm{H^+} \rightarrow \mathrm{Mn^{2+}} + \mathrm{O_2} + \mathrm{H_2O}$$

$$\boldsymbol{x_0}\mathrm{MnO_4^-} + \boldsymbol{x_1}\mathrm{H_2O_2} + \boldsymbol{x_2}\mathrm{H^+} \rightarrow \boldsymbol{x_3}\mathrm{Mn^{2+}} + \boldsymbol{x_4}\mathrm{O_2} + \boldsymbol{x_5}\mathrm{H_2O}$$

- Develop the **objective function** and the **constraints**:

$$obj\colon\ minimize \sum_{i=0}^{n} x_i$$

due to Principle of Atom Conservation (**POAC**)

Chemistry is Optimization

$$\boldsymbol{x_0}\mathrm{MnO_4^-} + \boldsymbol{x_1}\mathrm{H_2O_2} + \boldsymbol{x_2}\mathrm{H^+} \rightarrow \boldsymbol{x_3}\mathrm{Mn^{2+}} + \boldsymbol{x_4}\mathrm{O_2} + \boldsymbol{x_5}\mathrm{H_2O}$$

$$obj\colon\ minimize \sum_{i=0}^{n} x_i$$

$x_0 = x_3$	To balance the Manganese (Mn)
$4x_0 + 2x_1 = 2x_4 + x_5$	To balance the Oxygen (O)
$2x_1 + x_2 = 2x_5$	To balance the Hydrogen (H)
$-x_0 + x_2 = 2x_3$	To balance the charges

Chemistry is Optimization

- We have transformed solving a chemical equation into a solving a **linear optimization** problem
  - The **Simplex** method is a known good approach to solve linear programming problems in reasonable time
  - However managing **slack variables** and the **tableau** algorithm can be intimidating for novice programmers
- For small problems, we can just enumerate the ***search space*** (defined by all *permutations* of coefficients subject to the constraints) in order to find all of the possible solutions!

Encoding an Chemical Equation

- We can encode a chemical equation using a **2D matrix**
  - There will be a horizontal row for each **element** (atom)
  - We may need another row to encode the **ionic charges**
  - The **last row** will always contain the **maximum values** we want to consider for each unknown coefficient
- Each molecule is a term in the equation
  - There will be a vertical column for each **molecule** (or charged ion)
  - Every column is another **unknown** we need to resolve
- The matrix elements will be stored in a text (.txt) file
  - The first line will describe the **# of cols** and **# of rows**

Encoding Chemical Equation 1

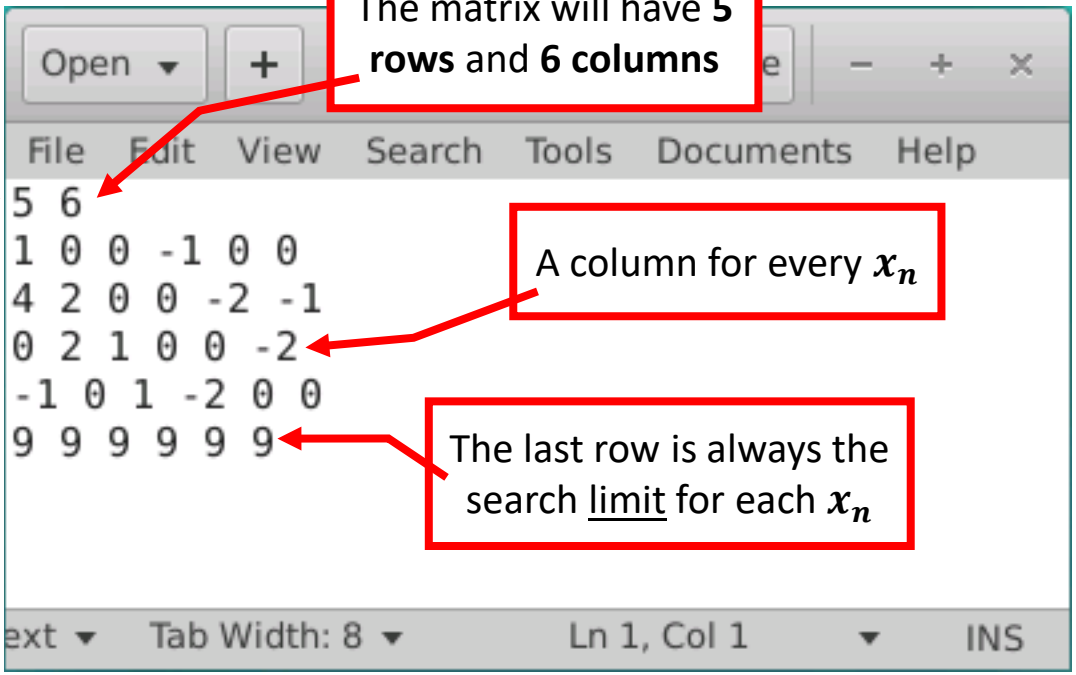
$$\mathrm{MnO_4^-} + \mathrm{H_2O_2} + \mathrm{H^+} \rightarrow \mathrm{Mn^{2+}} + \mathrm{O_2} + \mathrm{H_2O}$$

$$\boldsymbol{x_0}\mathrm{MnO_4^-} + \boldsymbol{x_1}\mathrm{H_2O_2} + \boldsymbol{x_2}\mathrm{H^+} \rightarrow \boldsymbol{x_3}\mathrm{Mn^{2+}} + \boldsymbol{x_4}\mathrm{O_2} + \boldsymbol{x_5}\mathrm{H_2O}$$

One row for every atom and a row for charges ***if needed***

The matrix will have **5 rows** and **6 columns**

Manganese (Mn)
Oxygen (O)
Hydrogen (H)
Charges (+/-)
Max Value



A column for every $\boldsymbol{x_n}$

The last row is always the search <u>limit</u> for each $\boldsymbol{x_n}$

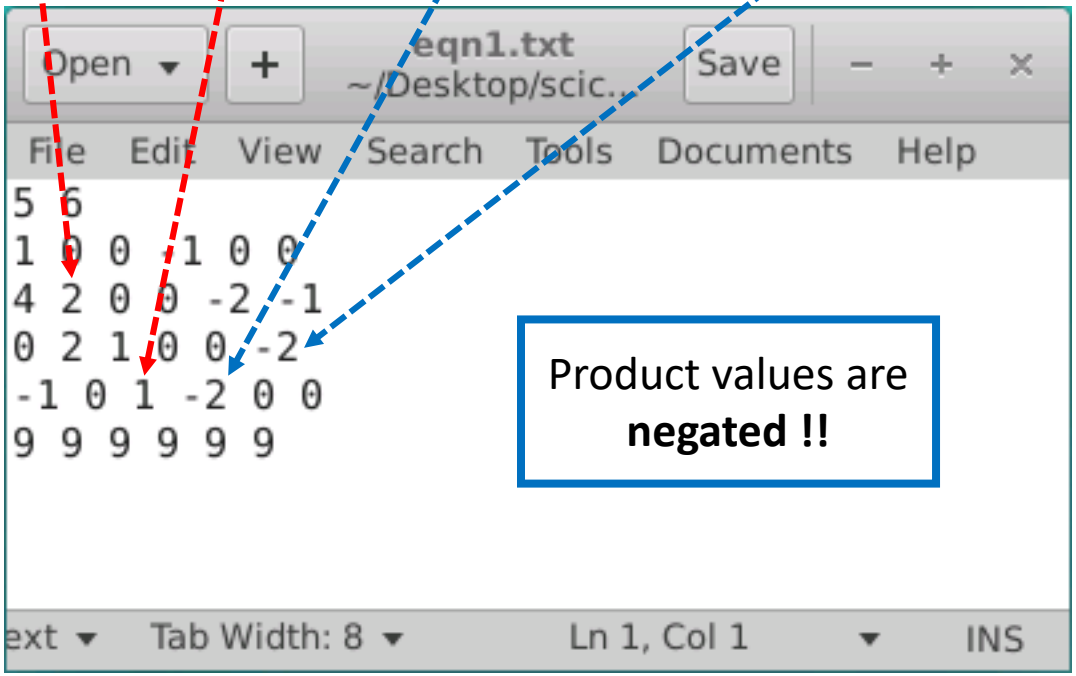
Encoding Chemical Equation 1

$$MnO_4^- + H_2O_2 + H^+ \rightarrow Mn^{2+} + O_2 + H_2O$$

$$x_0 MnO_4^- + x_1 H_2O_2 + x_2 H^+ \rightarrow x_3 Mn^{2+} + x_4 O_2 + x_5 H_2O$$

Reactant values are copied verbatim

Manganese (Mn)
Oxygen (O)
Hydrogen (H)
Charges (+/-)
Max Value



Product values are **negated !!**

Encoding Chemical Equation 1

- A potential **valid** set of unknown values will yield a **sum of zero** when convolved with each row (constraint) in the matrix

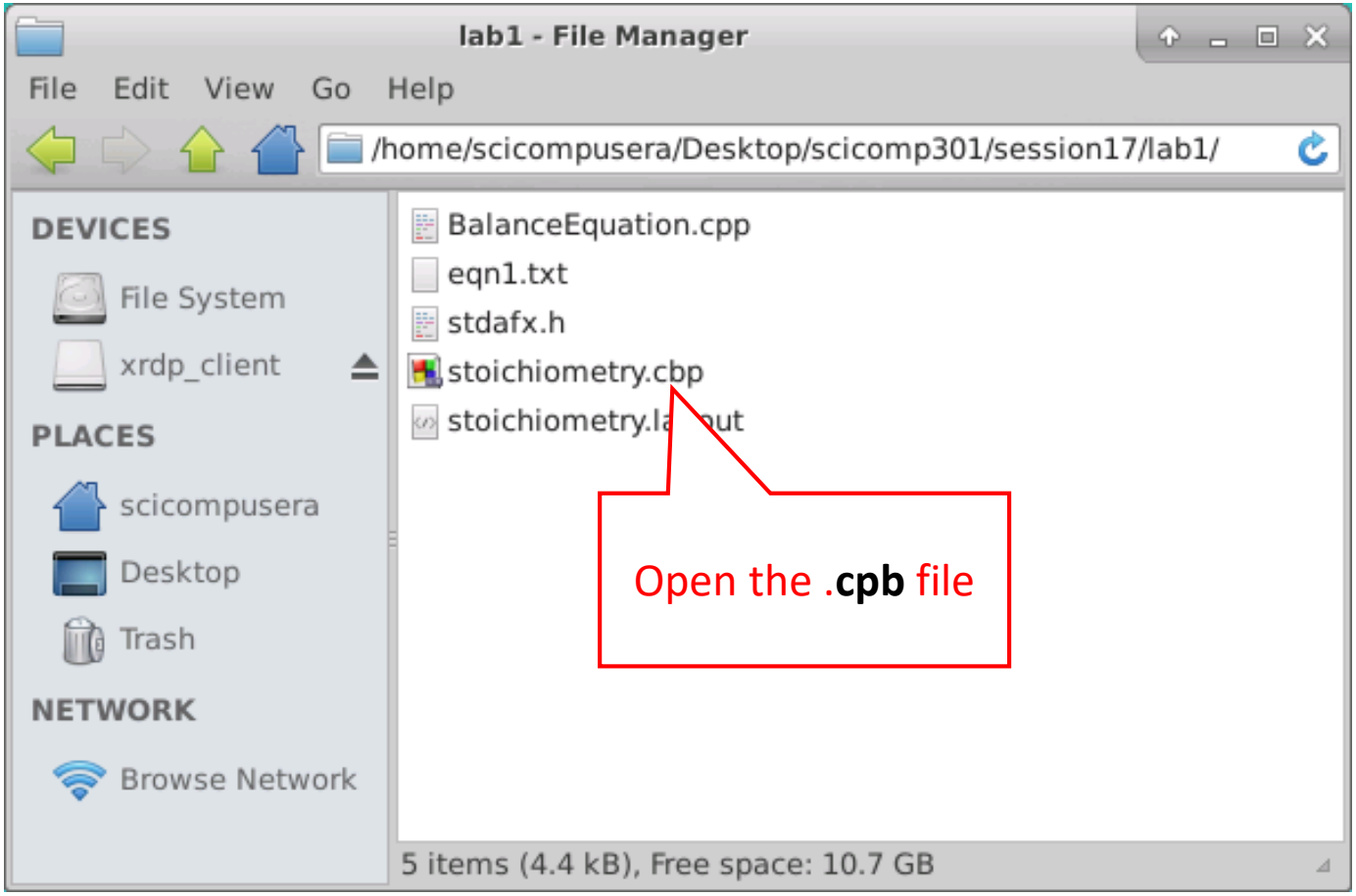
$$\boldsymbol{x_0}\mathrm{MnO_4^-} + \boldsymbol{x_1}\mathrm{H_2O_2} + \boldsymbol{x_2}\mathrm{H^+} \rightarrow \boldsymbol{x_3}\mathrm{Mn^{2+}} + \boldsymbol{x_4}\mathrm{O_2} + \boldsymbol{x_5}\mathrm{H_2O}$$

For the Oxygen atom these coefficients are taken from 3rd row in matrix

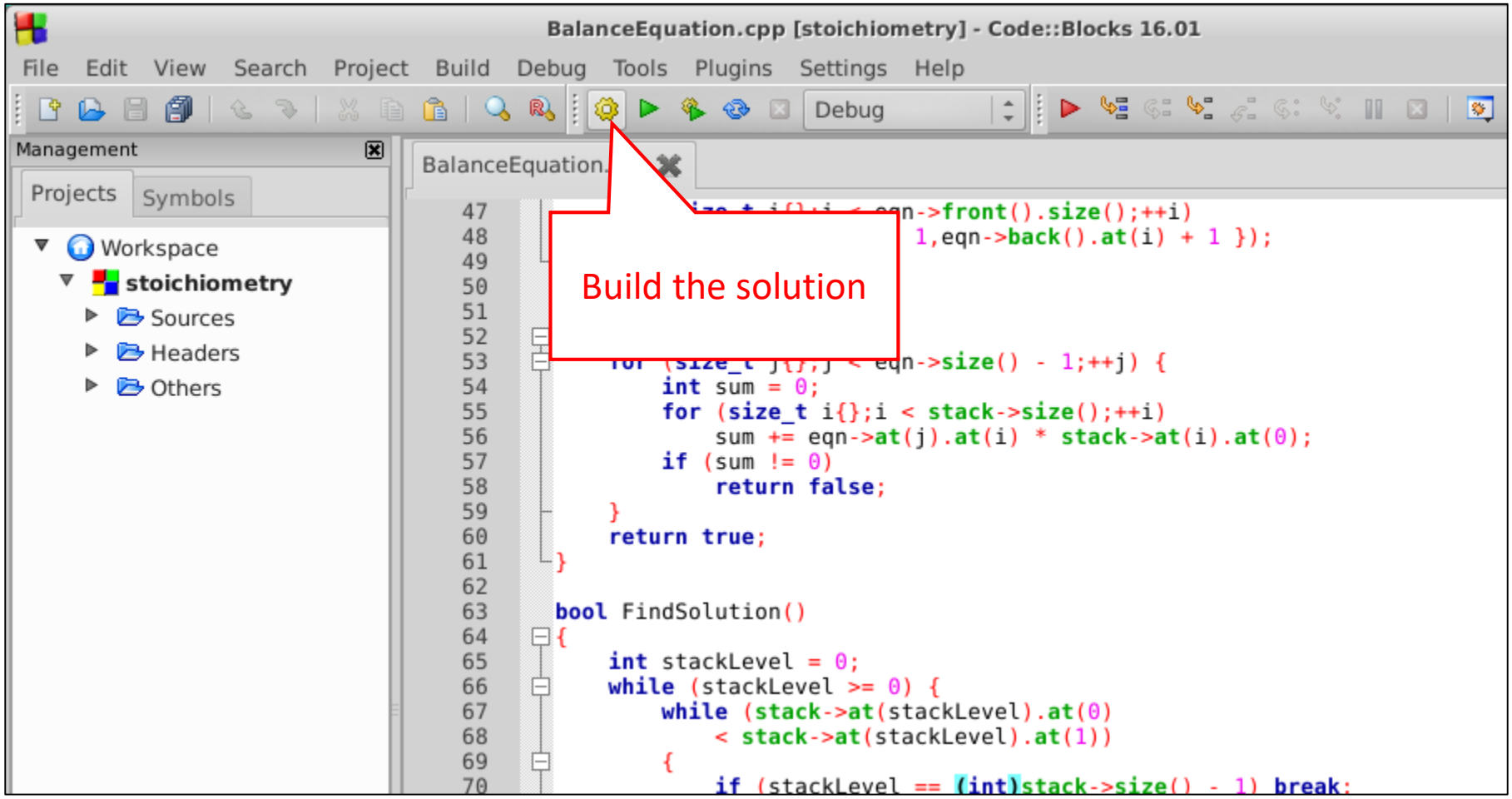
$$(\mathbf{4})\boldsymbol{x_0} + (\mathbf{2})\boldsymbol{x_1} + (\mathbf{0})\boldsymbol{x_2} + (\mathbf{0})\boldsymbol{x_3} + (\mathbf{-2})\boldsymbol{x_4} + (\mathbf{-1})\boldsymbol{x_5} = \boxed{\mathbf{0}}$$

- The current search values for each $\boldsymbol{x_n}$ are multiplied by each **column** coefficient in each **row** in the equation matrix – if the sum for **every row == 0**, then we have a *potential* solution!
- The final solution is where $\sum x_n$ has the lowest value (**POAC**)

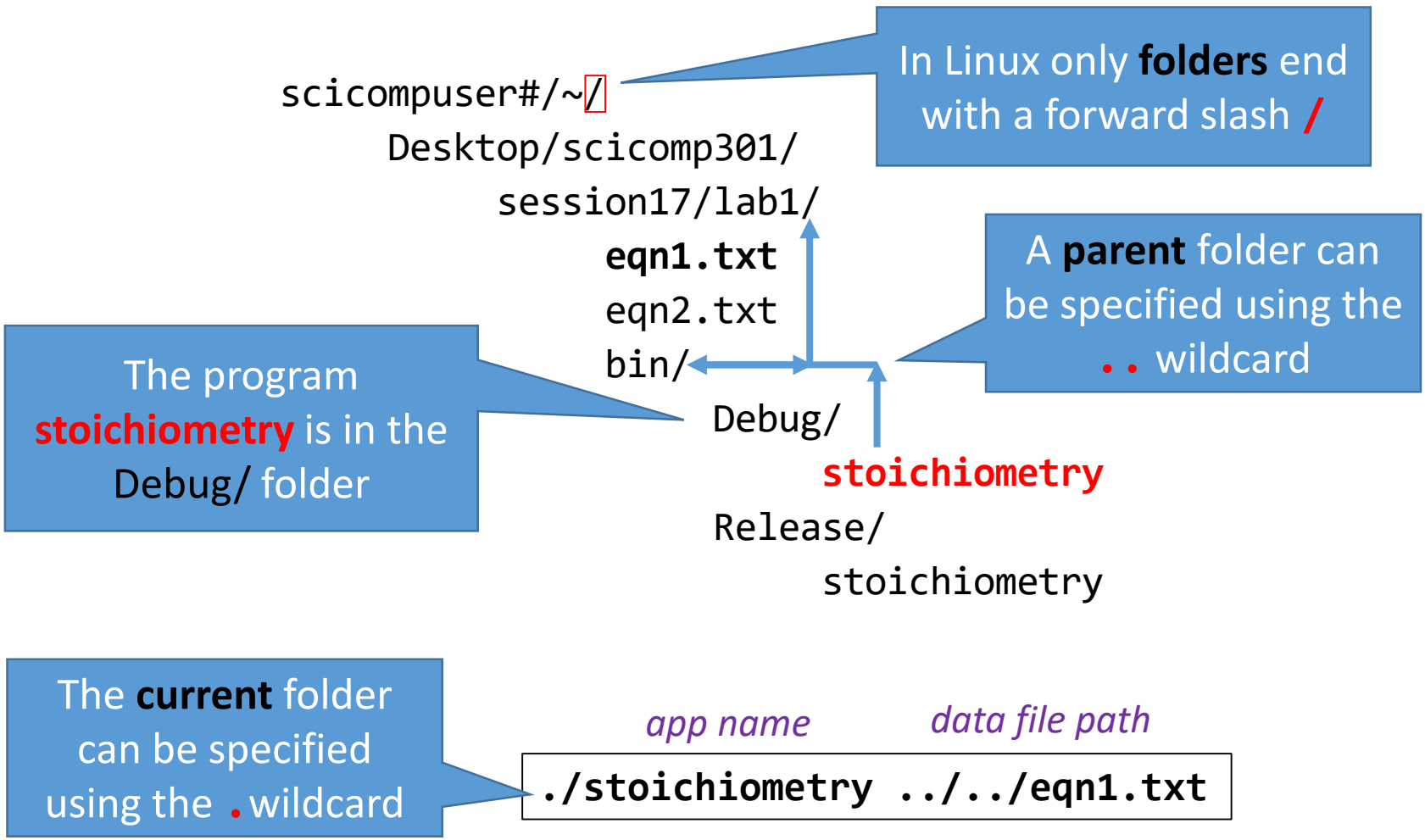
Open Lab 1 – Stoichiometry



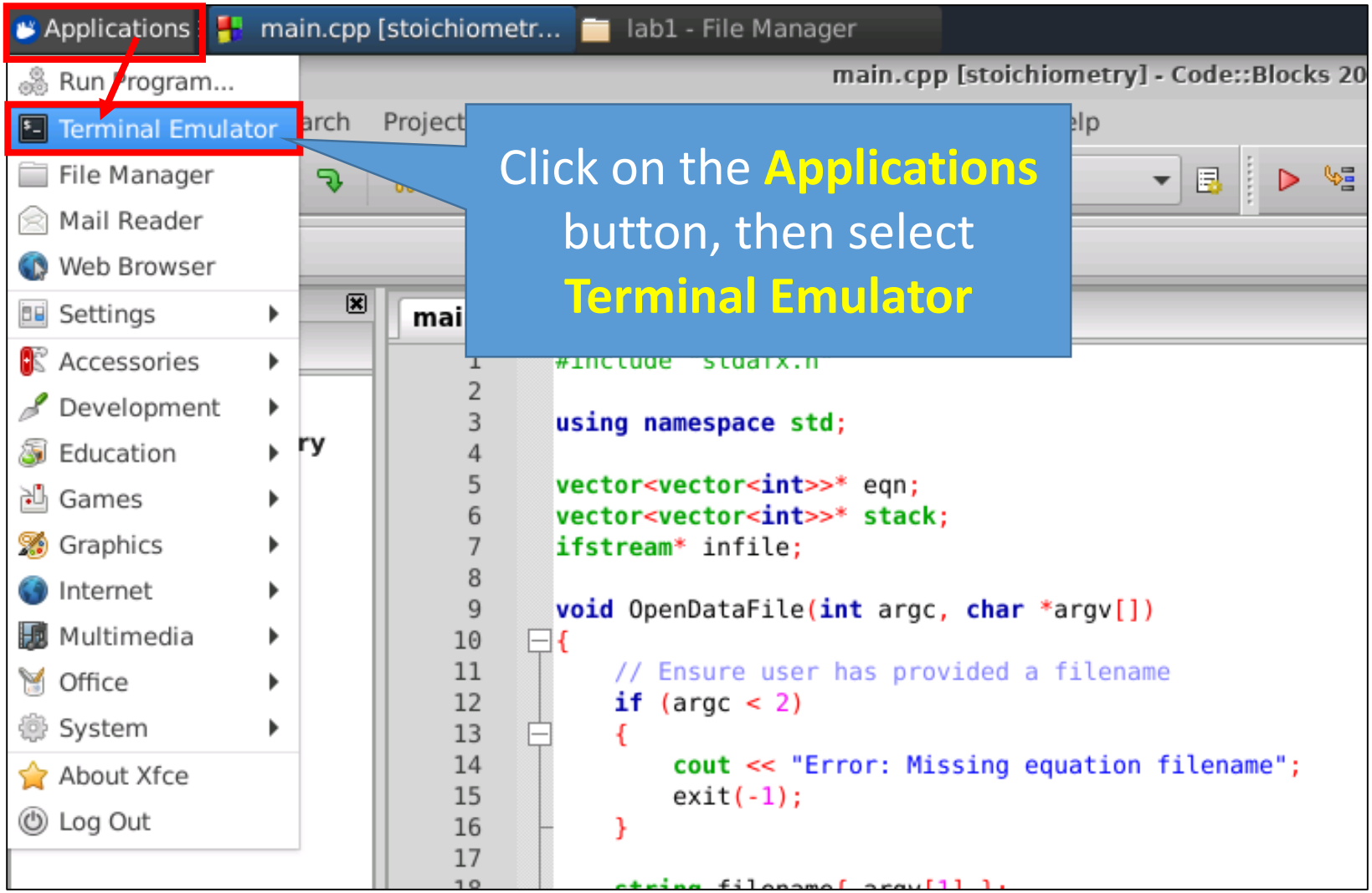
Build Lab 1 – Stoichiometry



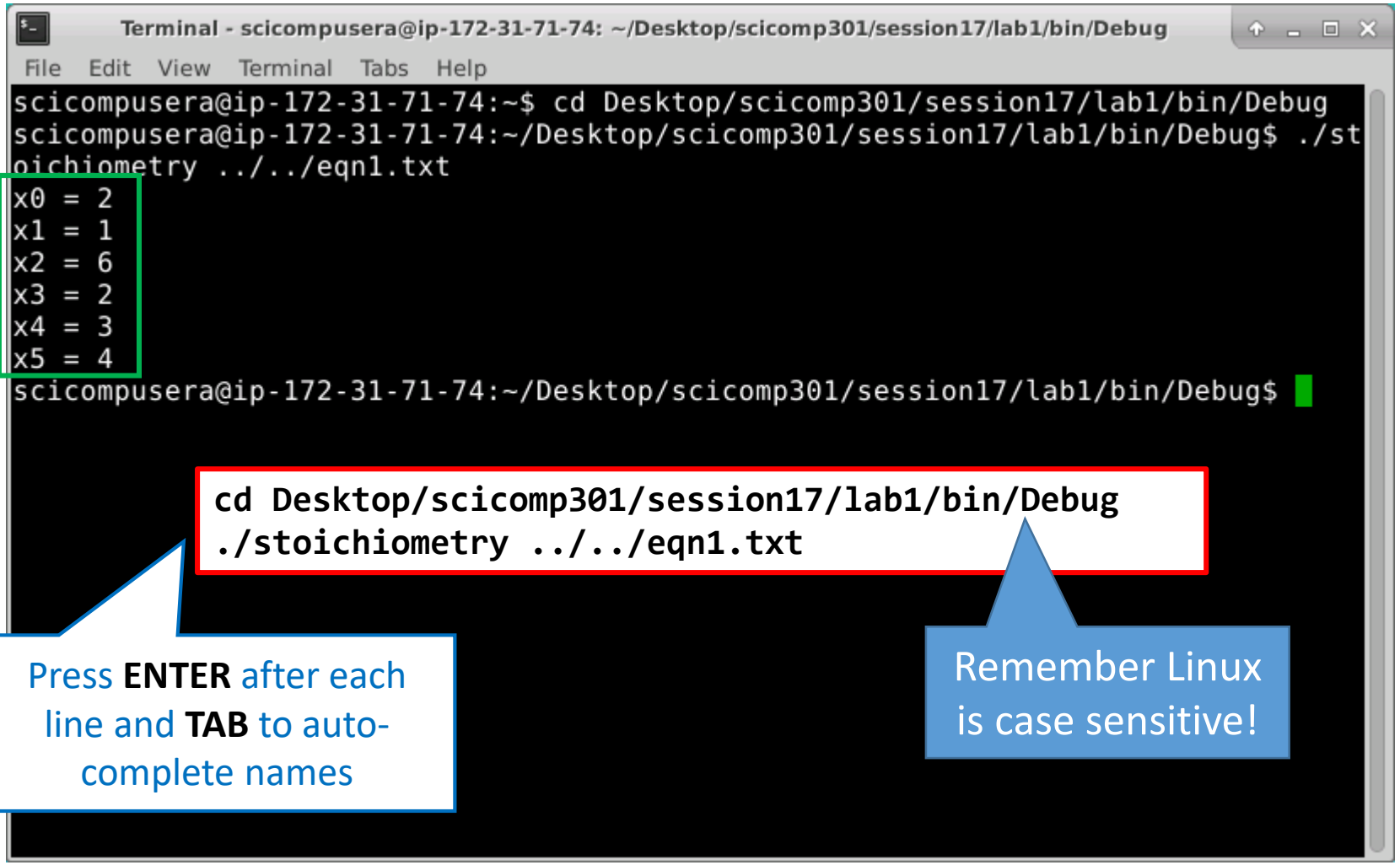
Project Folder & File Structure



Opening A Linux Terminal

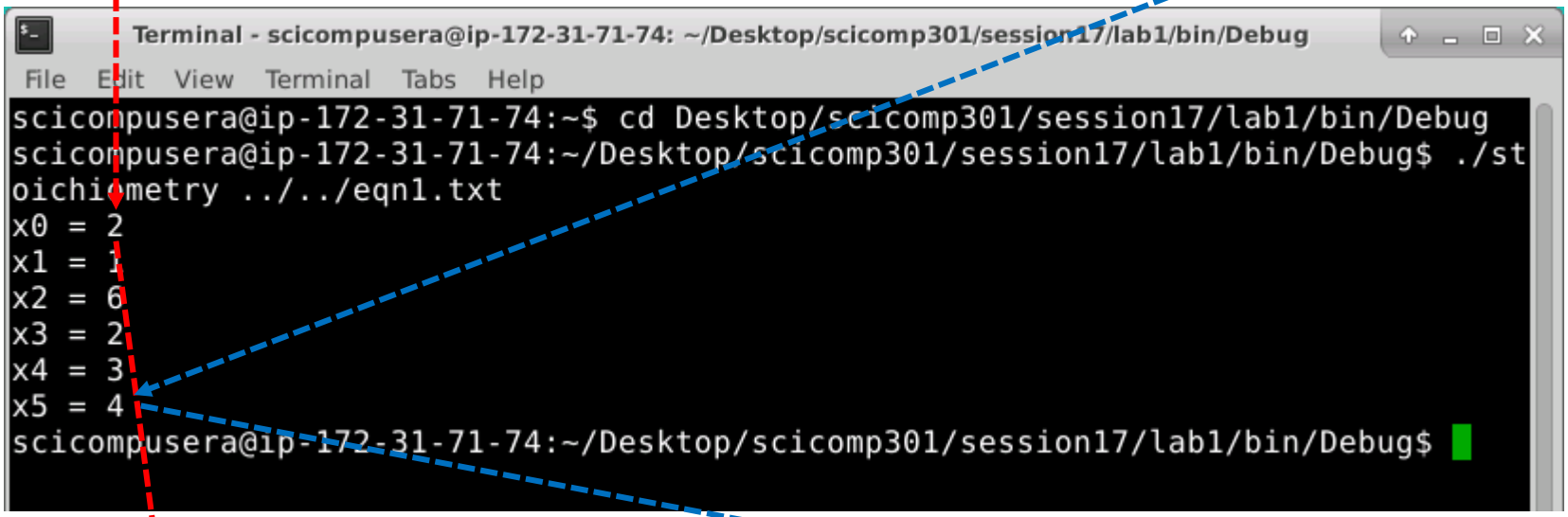


Run Lab 1 – Stoichiometry



Verify Chemical Equation 1

$$x_0 MnO_4^- + x_1 H_2O_2 + x_2 H^+ \rightarrow x_3 Mn^{2+} + x_4 O_2 + x_5 H_2O$$



$$2MnO_4^- + 1H_2O_2 + 6H^+ \rightarrow 2Mn^{2+} + 3O_2 + 4H_2O$$

Encoding Chemical Equation 2

$$\mathrm{Cr_2O_7^{2-}} + \mathrm{H^+} + \mathrm{H_2C_2O_4} \rightarrow \mathrm{Cr^{3+}} + \mathrm{H_2O} + \mathrm{CO_2}$$

$$\boldsymbol{x_0}\mathrm{Cr_2O_7^{2-}} + \boldsymbol{x_1}\mathrm{H^+} + \boldsymbol{x_2}\mathrm{H_2C_2O_4} \rightarrow \boldsymbol{x_3}\mathrm{Cr^{3+}} + \boldsymbol{x_4}\mathrm{H_2O} + \boldsymbol{x_5}\mathrm{CO_2}$$

A Row for Every Element

Matrix Size (Rows=6 Cols=6)

Six Rows:

1. Chromium (Cr)
2. Oxygen (O)
3. Hydrogen (H)
4. Carbon (C)
5. Charges
6. Max Search Value per Term

A Column for Every Molecule

Six Columns:

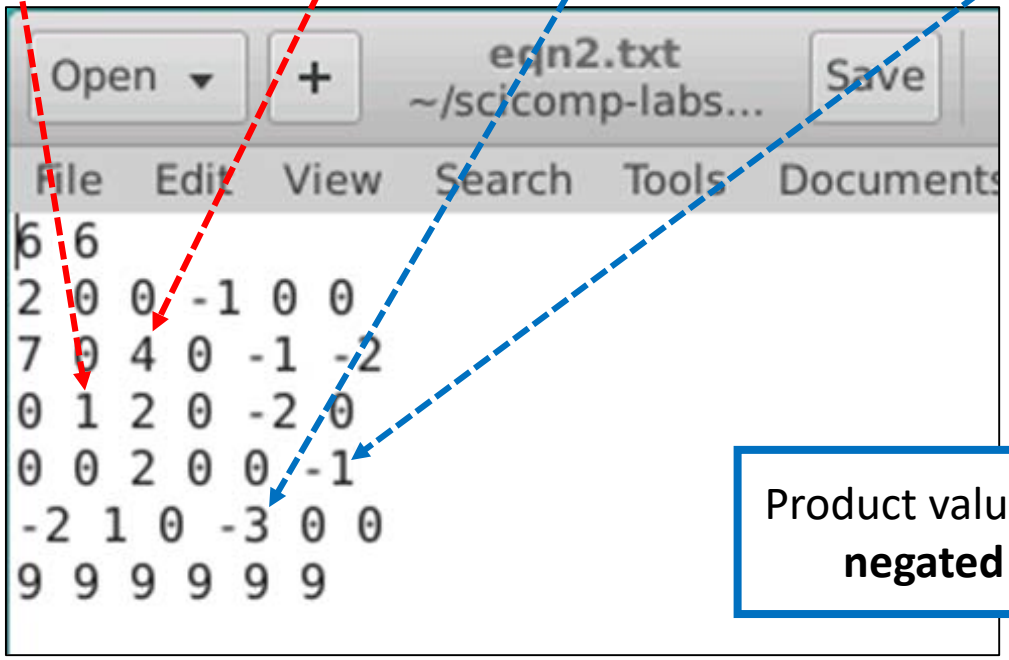
1. x_0
2. x_1
3. x_2
4. x_3
5. x_4
6. x_5

Encoding Chemical Equation 2

$$Cr_2O_7^{2-} + H^+ + H_2C_2O_4 \rightarrow Cr^{3+} + H_2O + CO_2$$

$$x_0 Cr_2O_7^{2-} + x_1 H^+ + x_2 H_2C_2O_4 \rightarrow x_3 Cr^{3+} + x_4 H_2O + x_5 CO_2$$

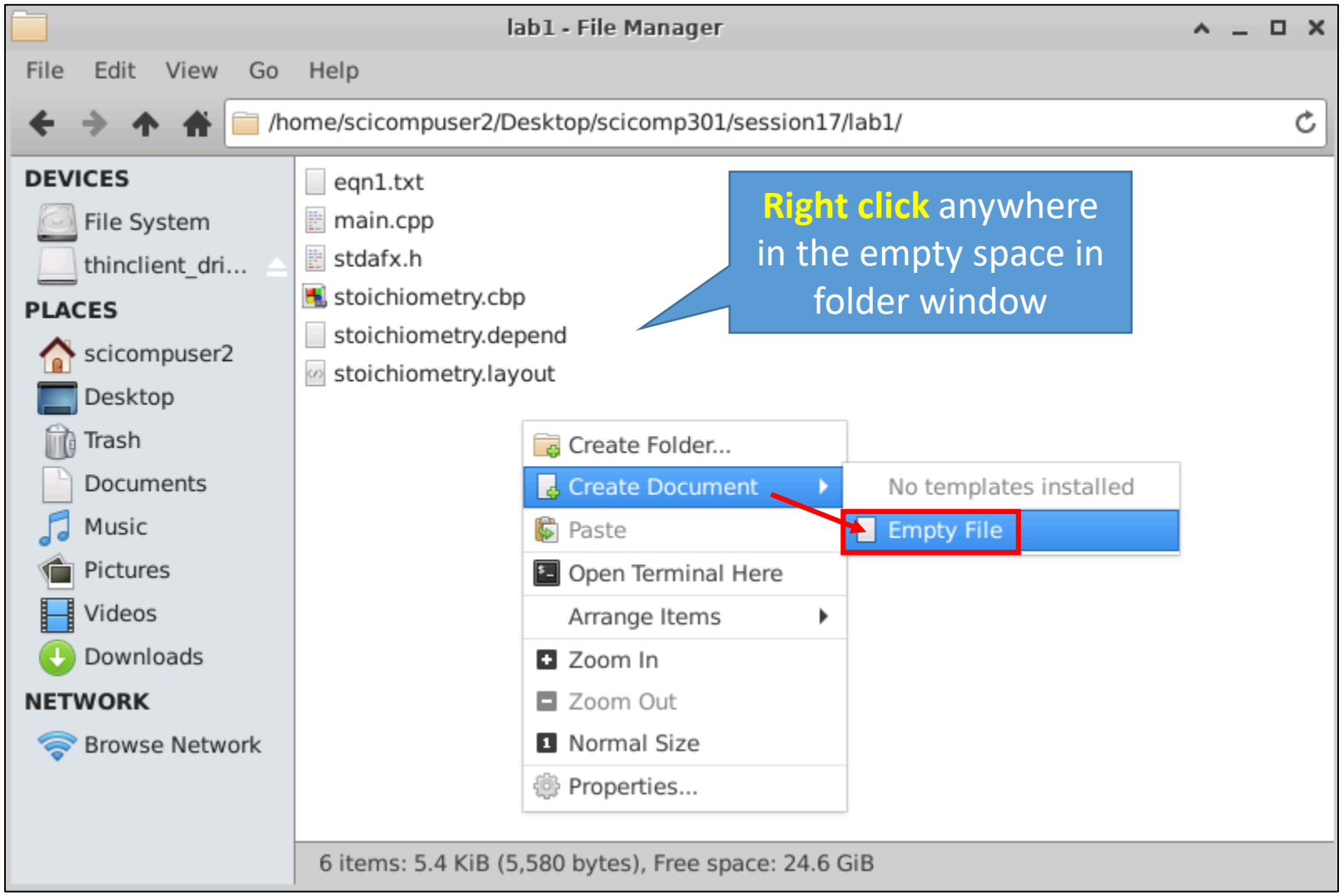
Reactant values are copied verbatim



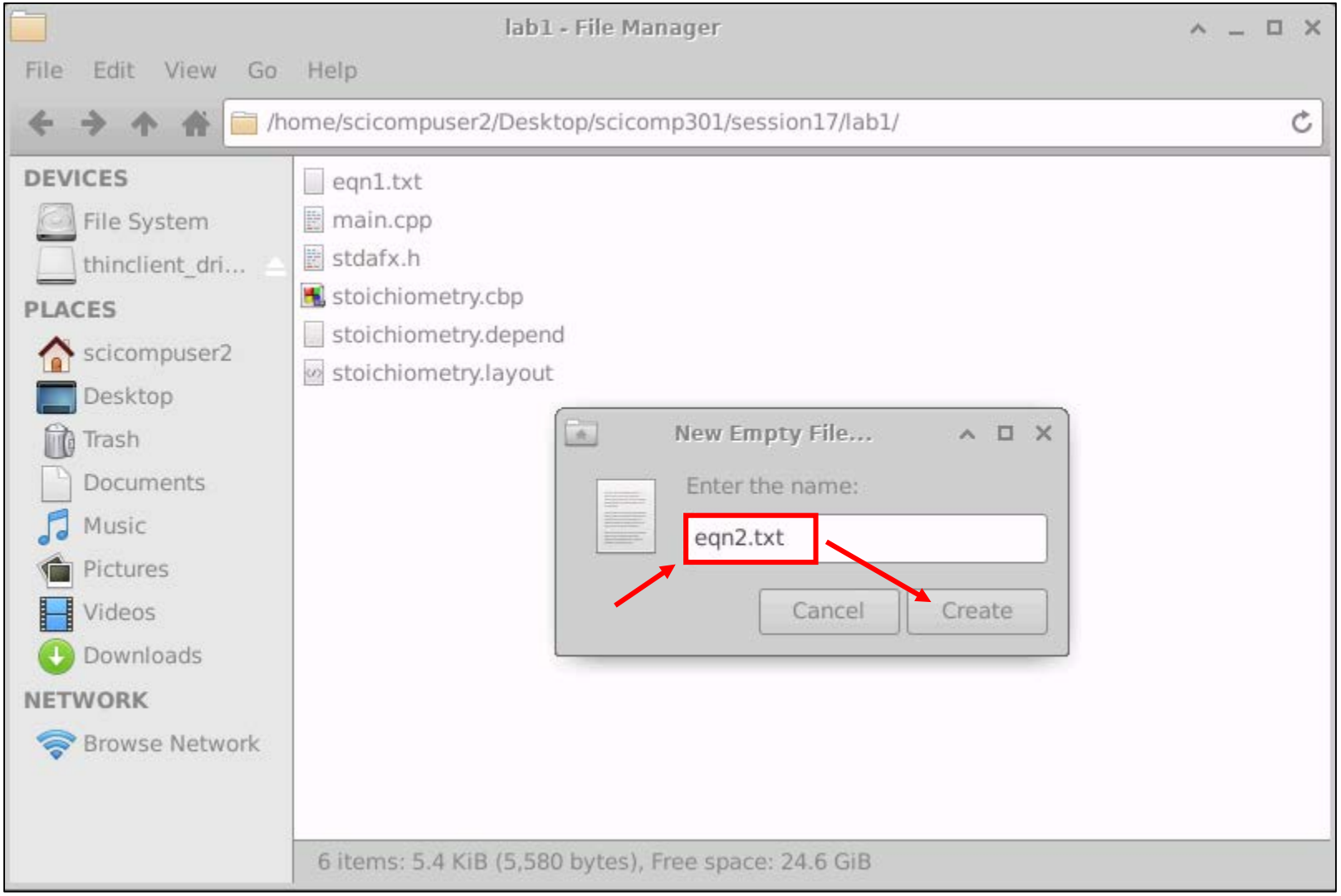
Chromium (Cr) →
Oxygen (O) →
Hydrogen (H) →
Carbon (C) →
Charges (+/-) →
Max Value →

Product values are **negated !!**

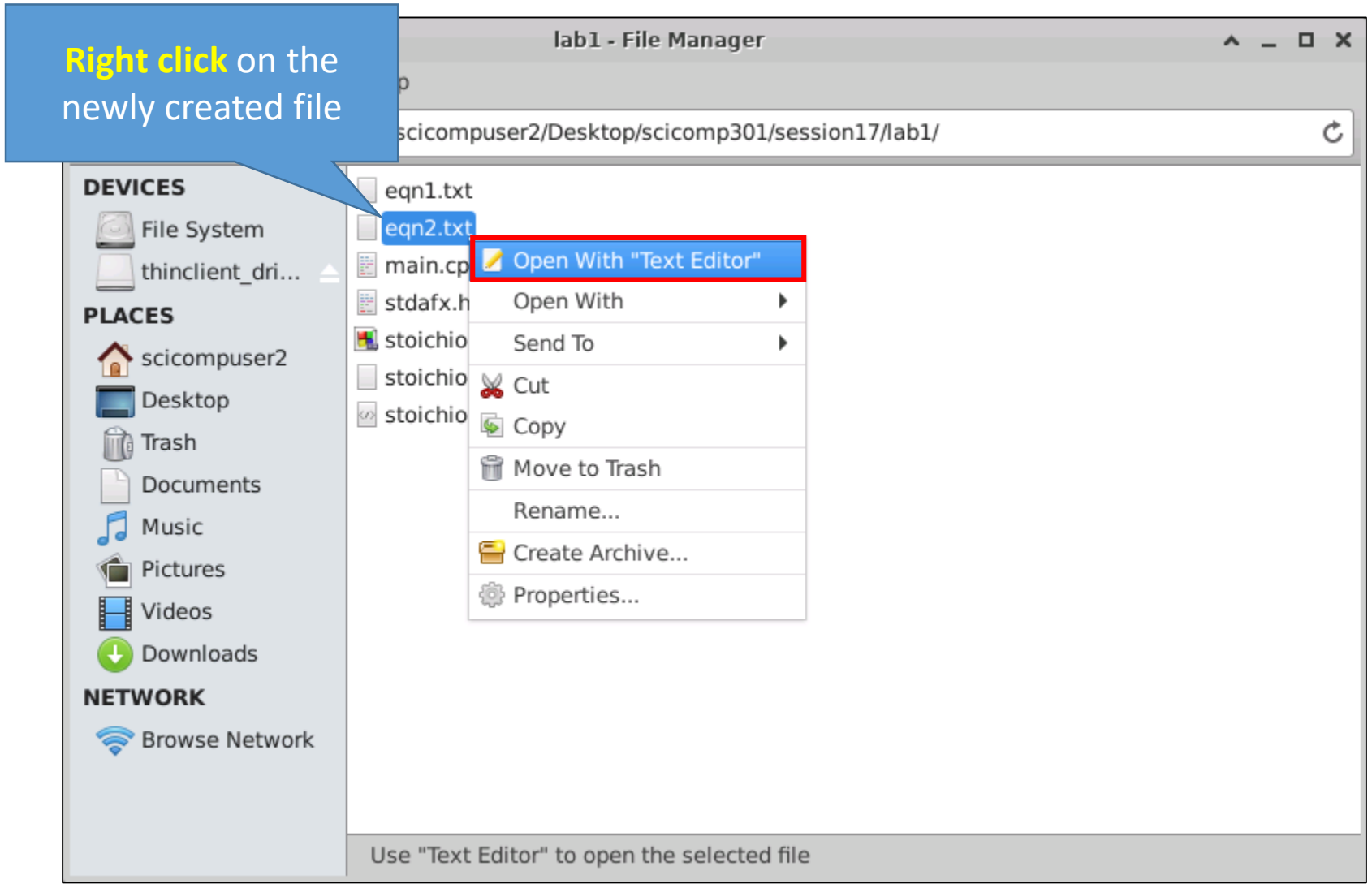
Encoding Chemical Equation 2



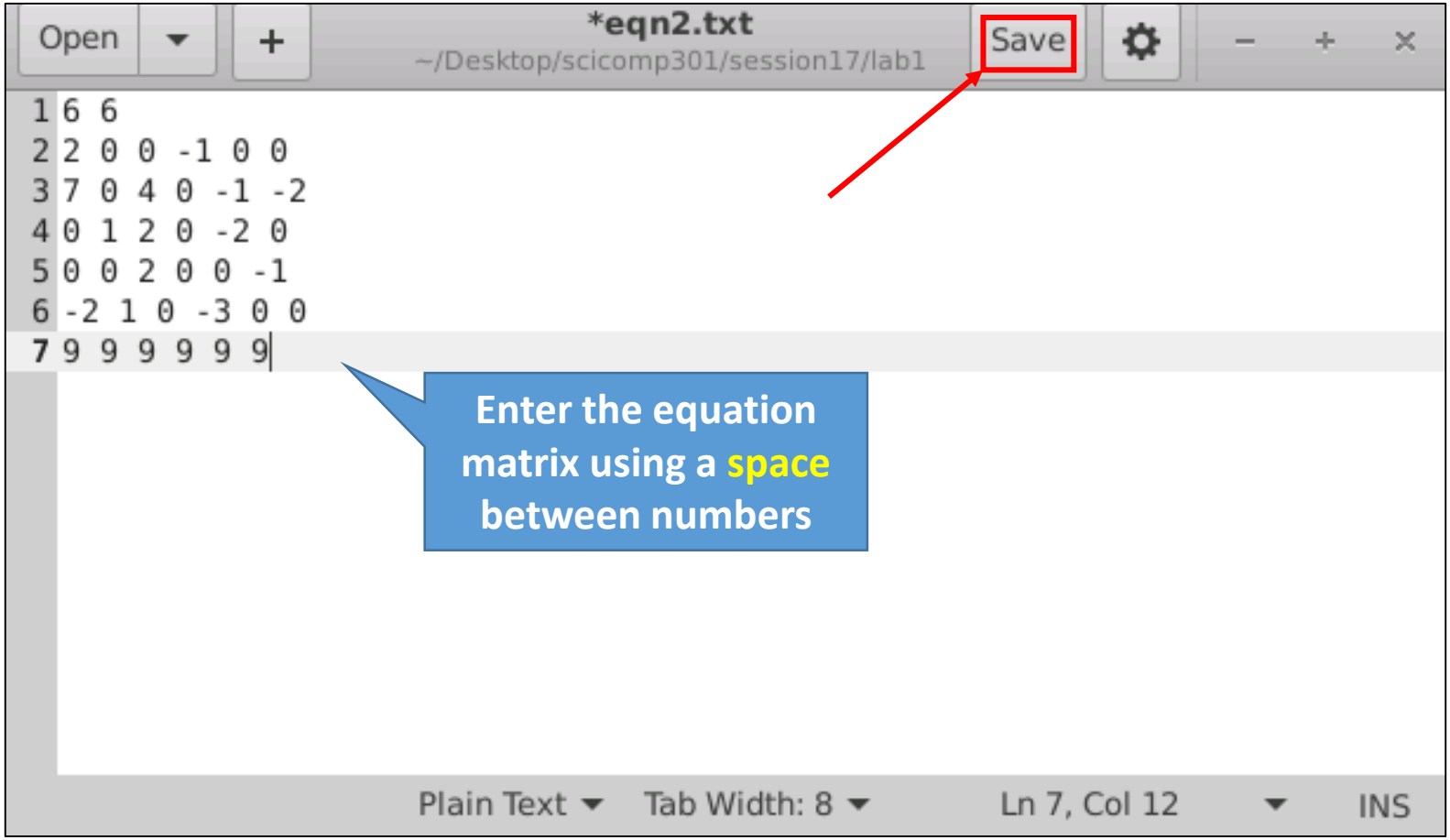
Encoding Chemical Equation 2



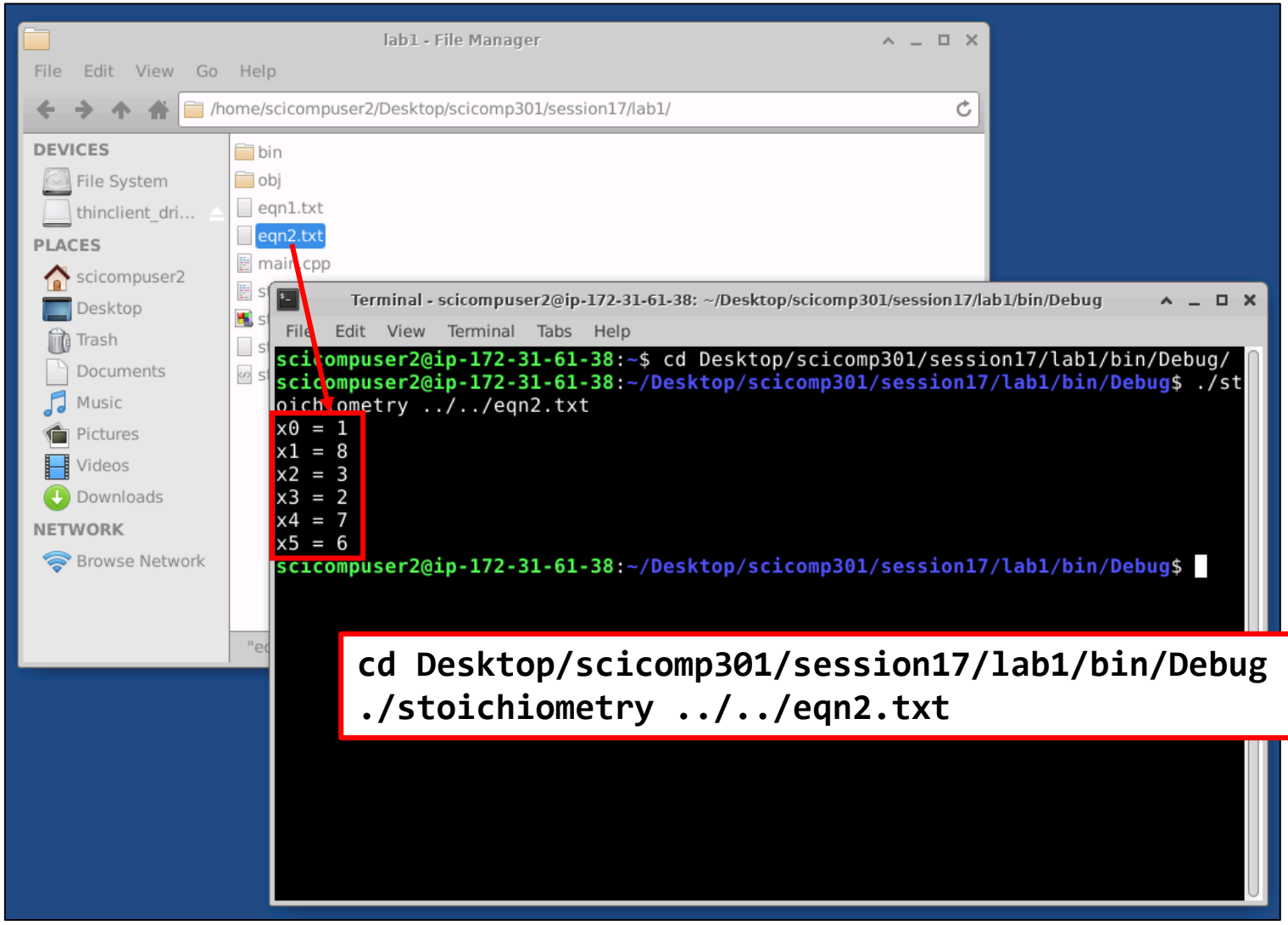
Encoding Chemical Equation 2



Encoding Chemical Equation 2

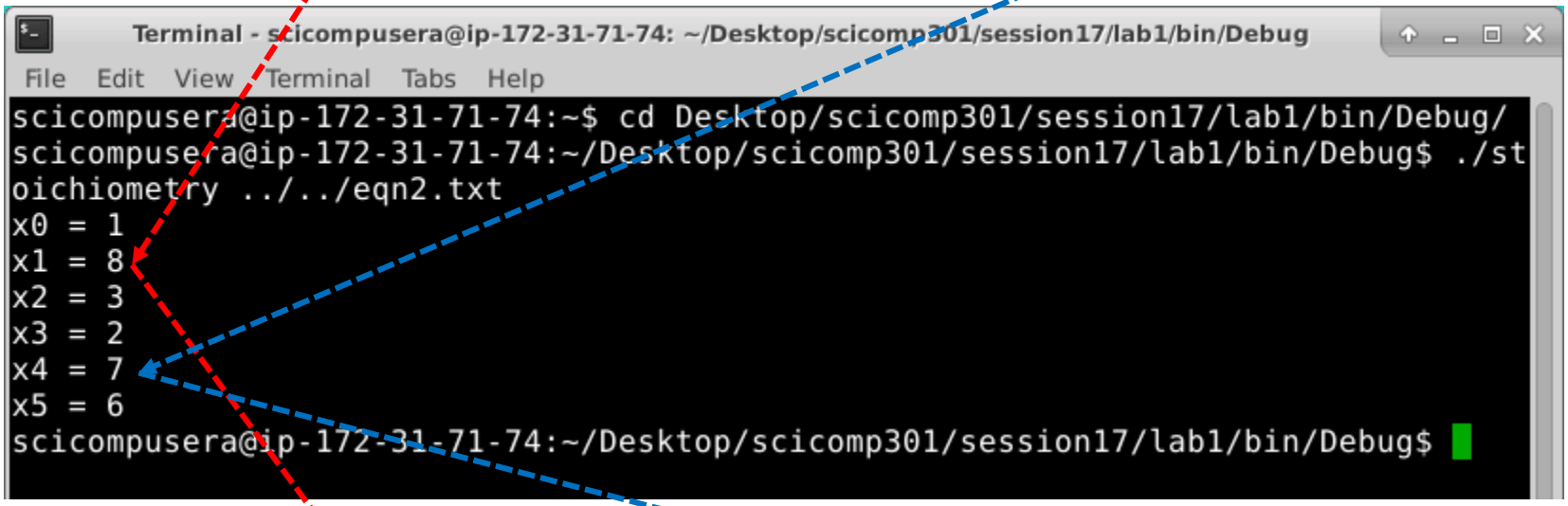


Balancing Chemical Equation 2



Verify Chemical Equation 2

$$x_0 Cr_2O_7^{2-} + x_1 H^+ + x_2 H_2C_2O_4 \rightarrow x_3 Cr^{3+} + x_4 H_2O + x_5 CO_2$$



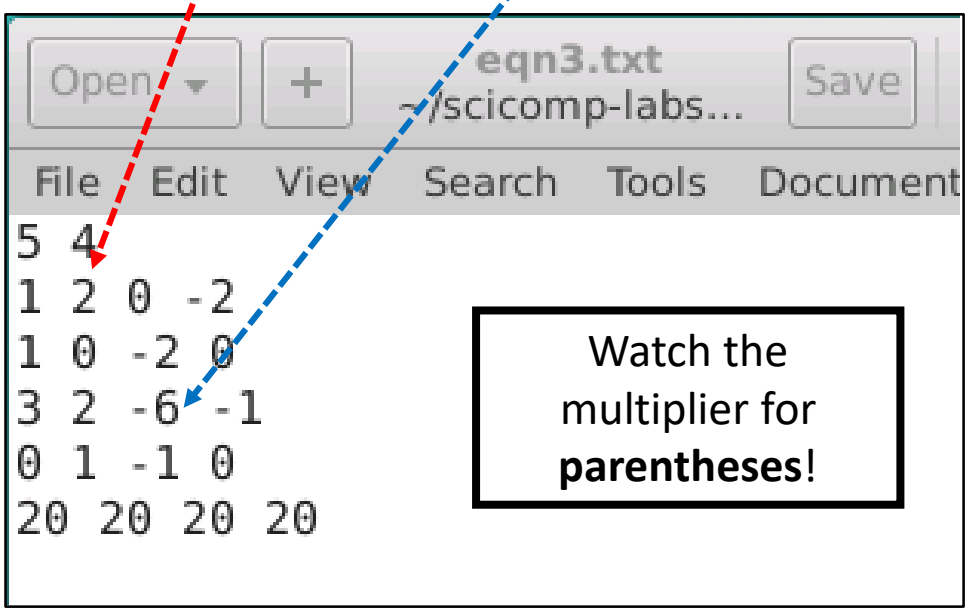
$$1Cr_2O_7^{2-} + 8H^+ + 3H_2C_2O_4 \rightarrow 2Cr^{3+} + 7H_2O + 6CO_2$$

Encoding Chemical Equation 3

$$HNO_3 + Ca(OH)_2 \rightarrow Ca(NO_3)_2 + H_2O$$

$$x_0 HNO_3 + x_1 Ca(OH)_2 \rightarrow x_2 Ca(NO_3)_2 + x_3 H_2O$$

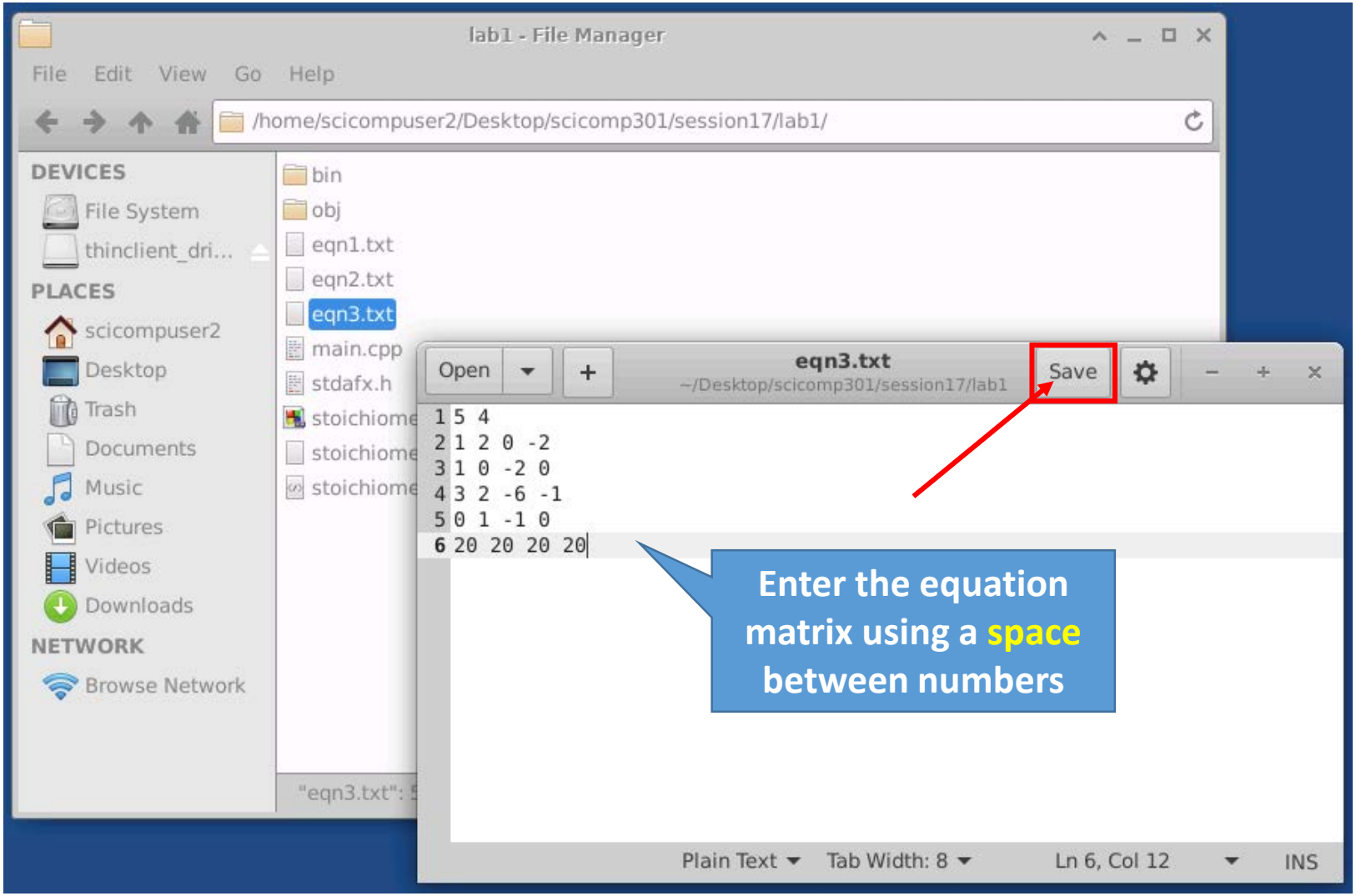
There is no row for **charges** in this equation



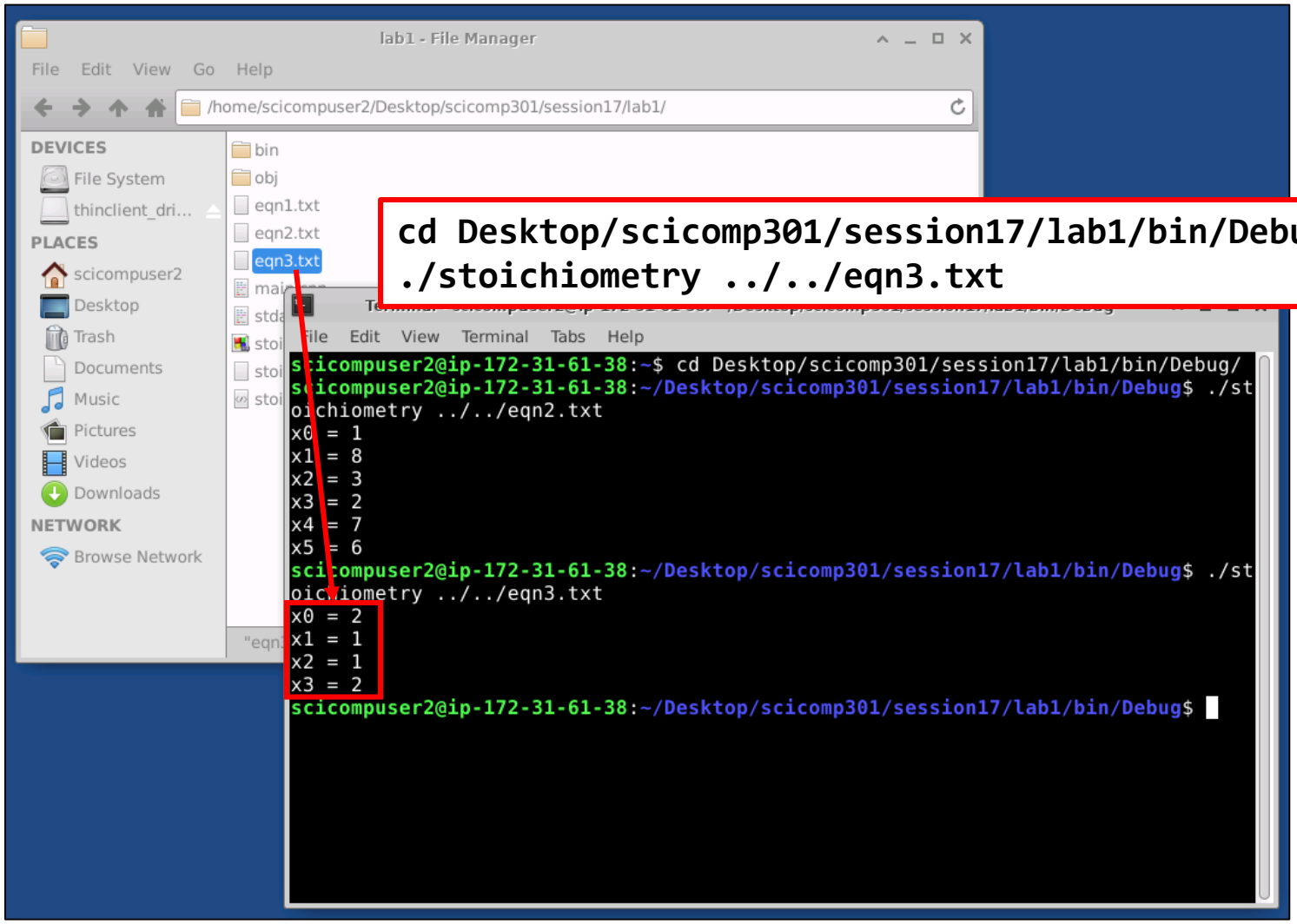
Hydrogen (H)
Nitrogen (N)
Oxygen (O)
Calcium (Ca)
Max Value

Watch the multiplier for **parentheses**!

Encoding Chemical Equation 3

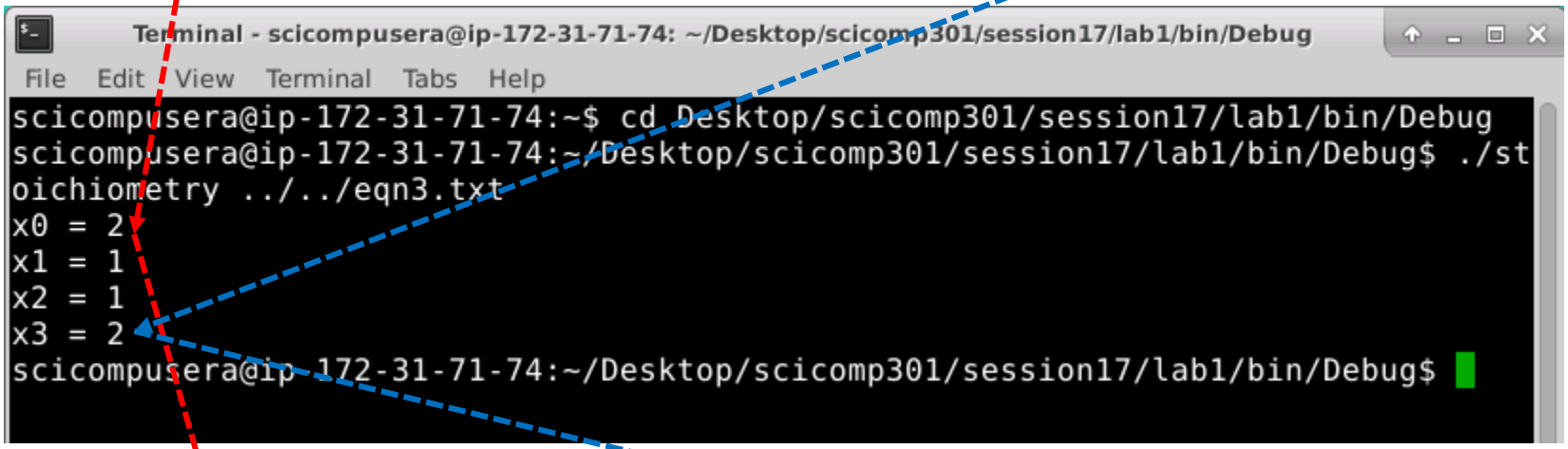


Balancing Chemical Equation 3



Verify Chemical Equation 3

$$x_0\mathrm{HNO_3} + x_1\mathrm{Ca(OH)_2} \rightarrow x_2\mathrm{Ca(NO_3)_2} + x_3\mathrm{H_2O}$$



$$\mathbf{2}\mathrm{HNO_3} + \mathbf{1}\mathrm{Ca(OH)_2} \rightarrow \mathbf{1}\mathrm{Ca(NO_3)_2} + \mathbf{2}\mathrm{H_2O}$$

Avoid Writing **Brittle** Code

```
static void NestedForLoops()
{
    for (int x0 = 0; x0 < 50; x0++)
    {
        for (int x1 = 0; x1 < 50; x1++)
        {
            for (int x2 = 0; x2 < 50; x2++)
            {
                for (int x3 = 0; x3 < 50; x3++)
                {
                    for (int x4 = 0; x4 < 50; x4++)
                    {
                        for (int x5 = 0; x5 < 50; x5++)
                        {
                            for (int x6 = 0; x6 < 50; x6++)
                            {
                                // Evaluate objective function
                            }
                        }
                    }
                }
            }
        }
    }
}
```

- We could use nested **for()** loops, but what if number of unknowns $(\boldsymbol{x_n})$ changes?
- Objective function would be evaluated $\mathbf{50^7}$ times =

$$\mathbf{781,250,000,000}$$

- No guarantee that molar coefficients $1 \leq x_n \leq 50$

$$P_2I_4 + P_4 + H_2O \rightarrow PH_4I + H_3PO_4$$

Use Data Structures vs. Brittle Code

```
vector<vector<int>>* stack;
```

Unknown	Index	Current Value	Max Value	
x_0	0	1	9	
x_1	1	1	9	
x_2	1	1	9	<--- stackLevel
x_3	2	1	9	
x_4	3	1	9	
x_5	4	1	9	

Our **stack** is a **vector of vectors**, with a row for **each unknown** and each row having **two** columns.
We also keep a pointer to the current ***stack level***

Use Data Structures vs. Brittle Code

Level	Range								
x_0	1	2	3	4	5	6	7	8	9
x1	1	2	3	4	5	6	7	8	9
x2	1	2	3	4	5	6	7	8	9
x3	1	2	3	4	5	6	7	8	9
x4	1	2	3	4	5	6	7	8	9
x5	1	2	3	4	5	6	7	8	9

- At each iteration, the current level moves to the next position in its range, and checks the constraints

Level	Range								
x_0	1	2	3	4	5	6	7	8	9
x1	1	2	3	4	5	6	7	8	9
x2	1	2	3	4	5	6	7	8	9
x3	1	2	3	4	5	6	7	8	9
x4	1	2	3	4	5	6	7	8	9
x5	1	2	3	4	5	6	7	8	9

- If the equation balances, but a new **minimum** atom count (sum) is found, display the current value for each unknown, and that sum becomes the new minimum

Use Data Structures vs. Brittle Code

Level	Range								
x_0	1	2	3	4	5	6	7	8	9
x1	1	2	3	4	5	6	7	8	9
x2	1	2	3	4	5	6	7	8	9
x3	1	2	3	4	5	6	7	8	9
x4	1	2	3	4	5	6	7	8	9
x5	1	2	3	4	5	6	7	8	9

Level	Range								
x_0	1	2	3	4	5	6	7	8	9
x1	1	2	3	4	5	6	7	8	9
x2	1	2	3	4	5	6	7	8	9
x3	1	2	3	4	5	6	7	8	9
x4	1	2	3	4	5	6	7	8	9
x5	1	2	3	4	5	6	7	8	9

Level	Range								
x_0	1	2	3	4	5	6	7	8	9
x1	1	2	3	4	5	6	7	8	9
x2	1	2	3	4	5	6	7	8	9
x3	1	2	3	4	5	6	7	8	9
x4	1	2	3	4	5	6	7	8	9
x5	1	2	3	4	5	6	7	8	9

- Just as a car odometer rolls over when a digit range reaches its limit, we increment the prior level's position and reset the current level to 1
- The program ends when every level in the stack has reached the **limit** of its *individually* assigned range

Use Data Structures vs. Brittle Code

```
bool FindSolution()
{
    int stackLevel = 0;
    while (stackLevel >= 0) {
        while (stack->at(stackLevel).at(0)
            < stack->at(stackLevel).at(1))
        {
            if (stackLevel == (int)stack->size() - 1) break;
            stackLevel++;
            stack->at(stackLevel).at(0) = 1;
        }

        if (IsSolution())
            return true;

        stack->at(stackLevel).at(0)++;
        while (stack->at(stackLevel).at(0)
            == stack->at(stackLevel).at(1))
        {
            stackLevel--;
            if (stackLevel < 0) break;
            stack->at(stackLevel).at(0)++;
        }
    }
    return false;
}
```

Find the lowest level in the stack which has not yet enumerated its full range

Invoke the objective function

Use the next value in this level's range

Walk up the stack, skipping all parent levels that have already completed their range

Using Data Structures vs. Brittle Code

```
void LoadEquation()
{
    int constraints, unknowns;
    *infile >> constraints >> unknowns;

    eqn = new vector<vector<int>>(constraints);
    vector<int> constraint(unknowns);

    for (auto& e : *eqn) {
        for (auto& c : constraint)
            *infile >> c;
        e = constraint;
    }
}
```

To read from a file use the stream insertion **>>** operator

A solution is valid only if the convolution of every matrix row with every stack row **== 0**

```
bool IsSolution()
{
    for (size_t j{};j < eqn->size() - 1;++j) {
        int sum = 0;
        for (size_t i{};i < stack->size();++i)
            sum += eqn->at(j).at(i) * stack->at(i).at(0);
        if (sum != 0)
            return false;
    }
    return true;
}
```

Balancing Ionic Equations

- Pick **one** of the following equations to balance using the `stoichiometry` application
- In the Lab 1 folder, **create** a text file that encodes your chosen equation using the corresponding filename

eqn4.txt: $\mathrm{MnO_4^- + Fe^{2+} + H^+ \rightarrow Mn^{2+} + Fe^{3+} + H_2O}$

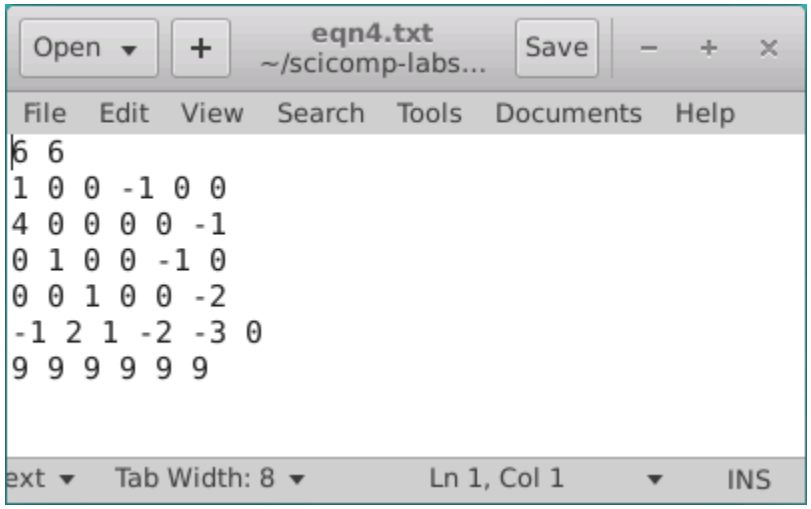
eqn5.txt: $\mathrm{C_8H_{18} + O_2 \rightarrow CO_2 + H_2O}$

eqn6.txt: $\mathrm{C_7H_6O_2 + O_2 \rightarrow CO_2 + H_2O}$

eqn7.txt: $\mathrm{P_2I_4 + P_4 + H_2O \rightarrow PH_4I + H_3PO_4}$

Balancing Chemical Equation 4

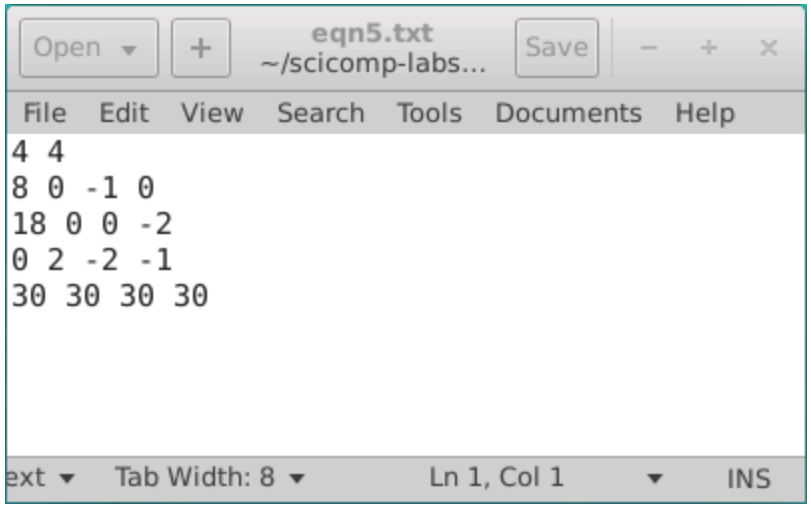
$$MnO_4^- + Fe^{2+} + H^+ \rightarrow Mn^{2+} + Fe^{3+} + H_2O$$



$$\mathbf{1}MnO_4^- + \mathbf{5}Fe^{2+} + \mathbf{8}H^+ \rightarrow \mathbf{1}Mn^{2+} + \mathbf{5}Fe^{3+} + \mathbf{4}H_2O$$

Balancing Chemical Equation 5

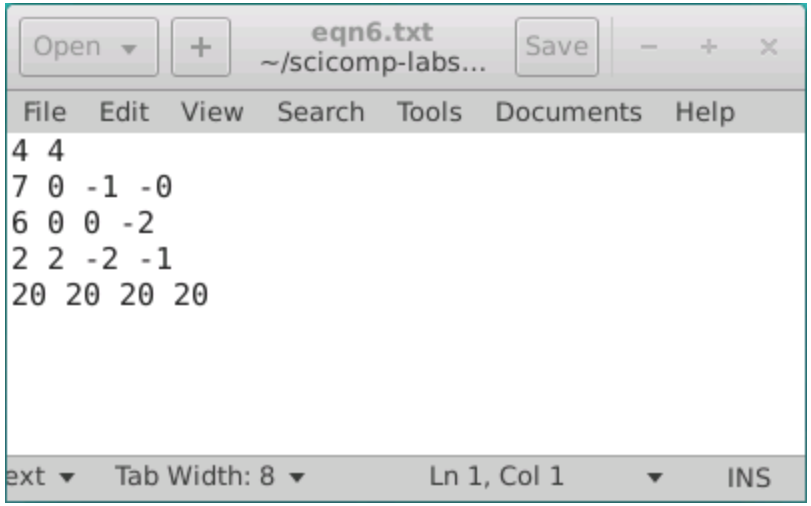
$$C_8H_{18} + O_2 \rightarrow CO_2 + H_2O$$



$$\mathbf{2}C_8H_{18} + \mathbf{25}O_2 \rightarrow \mathbf{16}CO_2 + \mathbf{18}H_2O$$

Balancing Chemical Equation 6

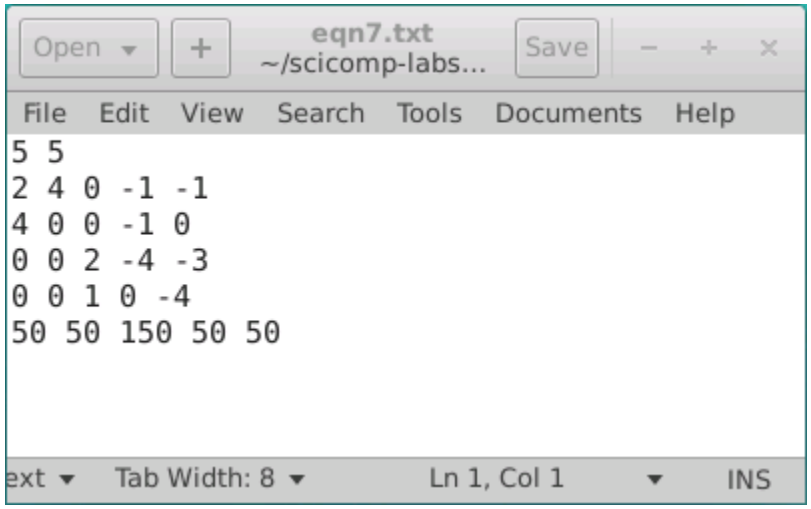
$$C_7H_6O_2 + O_2 \rightarrow CO_2 + H_2O$$



$$\mathbf{2}C_7H_6O_2 + \mathbf{15}O_2 \rightarrow \mathbf{14}CO_2 + \mathbf{6}H_2O$$

Balancing Chemical Equation 7

$$P_2I_4 + P_4 + H_2O \rightarrow PH_4I + H_3PO_4$$



$$\mathbf{10}P_2I_4 + \mathbf{13}P_4 + \mathbf{128}H_2O \rightarrow \mathbf{40}PH_4I + \mathbf{32}H_3PO_4$$

Diphosphorus tetraiodide

Now you know…

https://www.webqc.org/balance.php

- We can encode a chemical equation into a **matrix**, where the **rows** represent the element constraints for reactants and products and the **columns** represent the unknown coefficients of the molecules
- We can transform **balancing ionic equations** to a linear program, which can be solved using brute-force if the search space is sufficiently small
- Even relatively simple equations can exponentially increase the search space & using nested **for()** loops makes the code too **brittle**
- It is better to use the **Simplex** method, and normalize any rational solutions back to integers for the final answer

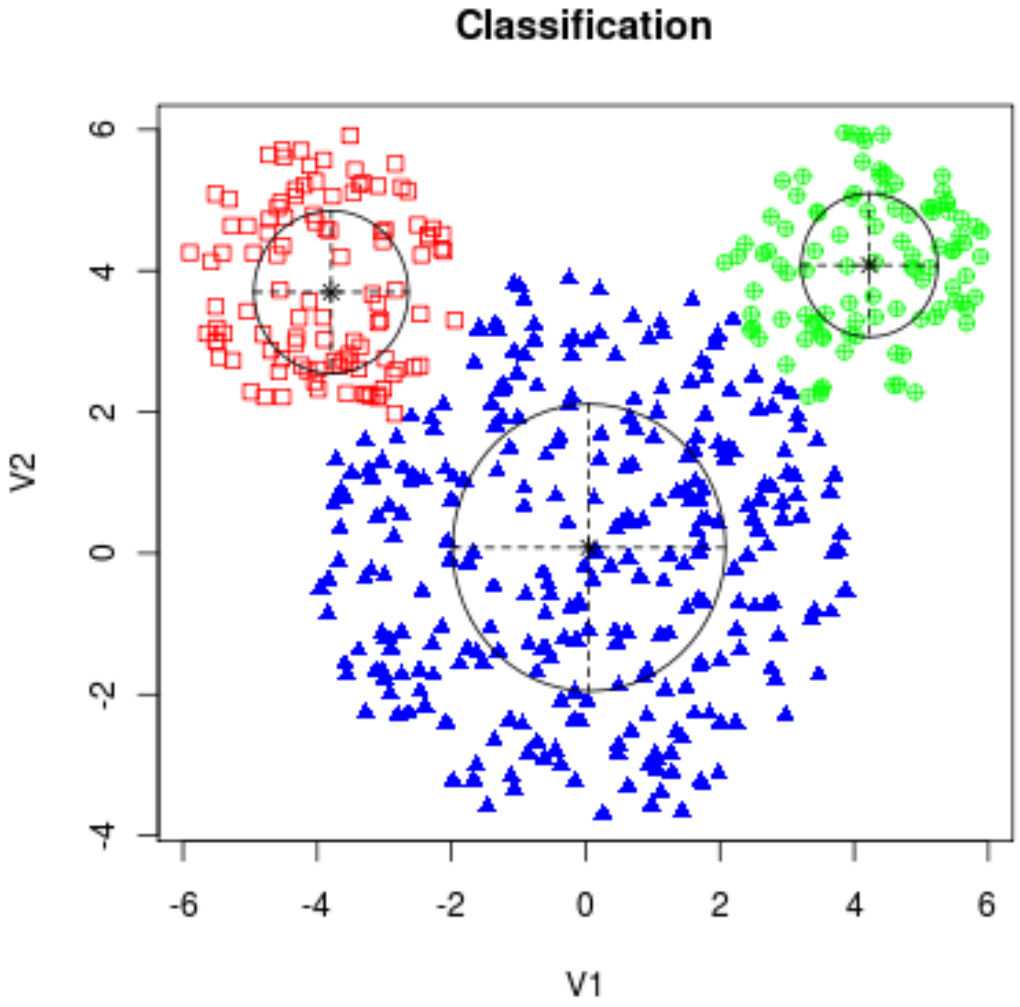
Session Goals

- Study an efficient algorithm for grouping similar **data sets**
- Gain an appreciation for **k-means Clustering** – an algorithm that iterates by **assigning data points to a cluster**
  - The goal is to programmatically assign data points that are “near” each other to the same **cluster** so we can perform group statistics on data that “belongs” together
  - Learn the limitations of k-means and visualize the dangers of “over fitting” data
  - Envision how we can use **variance** to programmatically identify **data outliers**

k-means Clustering

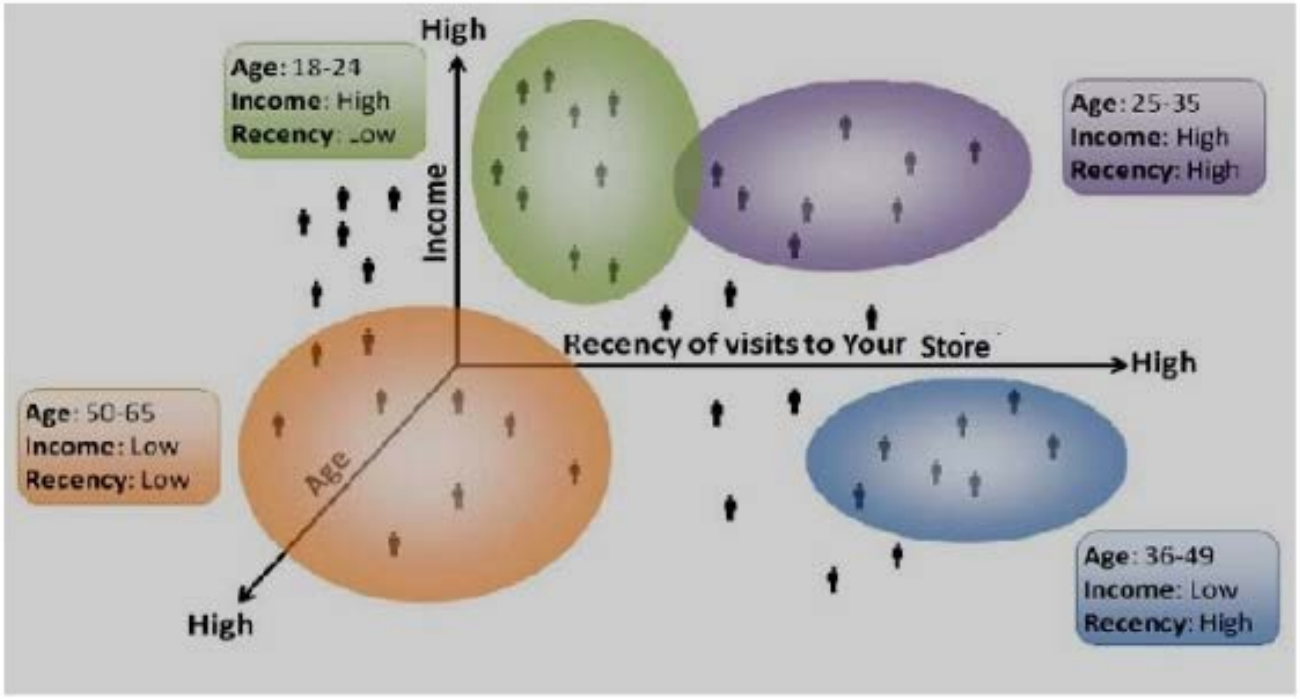
- *k*-means clustering aims to partition ***n*** observations into ***k*** clusters (where each cluster has its own $\boldsymbol{\mu}$ point)
- Each observation is assigned to the cluster whose $\boldsymbol{\mu}$ point **is closest** to that observation's point
- The problem is computationally difficult (NP-hard); however, there are efficient heuristic algorithms that are commonly employed and **converge quickly to a local optimum**
- The term "*k*-means" was first used by James MacQueen in 1967, though the idea goes back to Hugo Steinhaus in 1957
- A more efficient version was published Hartigan and Wong in **1979**

k-means Clustering



k-means Clustering

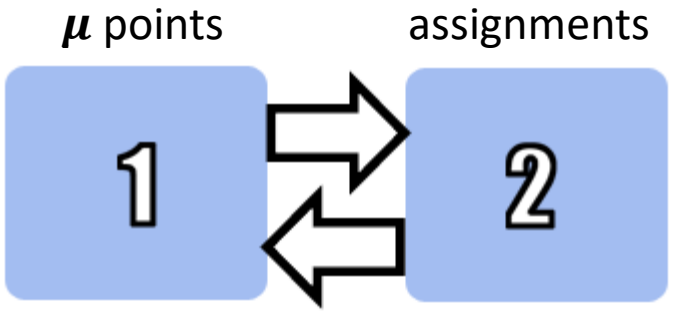
Example - Clusters using Age, Income & Recency



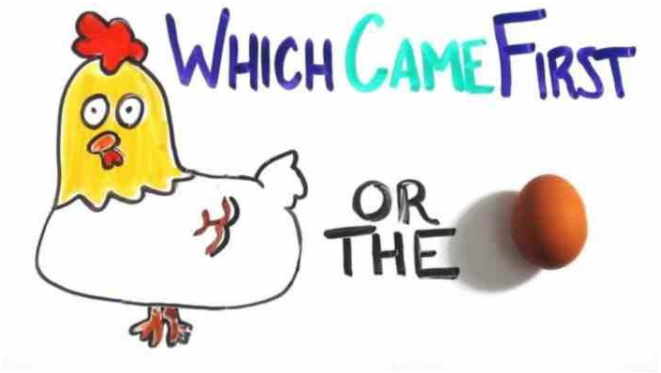
k-means Clustering

- A **one-to-many** mapping between **clusters** and **data points**
  - A **cluster** is a set of one or more unique data points
  - Every **data point** must be assigned to **only one** cluster at a time
  - Data points are initially assigned to clusters in a round-robin fashion
- The $\boldsymbol{\mu}$ point of each **cluster** is its geometric center (**centroid**) calculated by taking the average value (in each dimension) of ***only*** the **data points** assigned to that cluster
- Assignments *can* change the $\boldsymbol{\mu}$ points… which then *can* change the assignments… which then *can* change the $\boldsymbol{\mu}$ points… which then *can* change the assignments… which then *can*…

k-means Clustering



$$G_{\mu\nu} + \Lambda T_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}$$



Spacetime tells matter how to move;
matter tells spacetime how to curve.

— John Archibald Wheeler —

k-means Clustering

- The algorithm uses **two** phases per round:
  1. For each **cluster**, a new centroid ($\boldsymbol{\mu}$ point) is calculated using the ***current*** data point assignments

     If the new $\boldsymbol{\mu}$ ≠ current $\boldsymbol{\mu}$ then **move cluster** so current $\boldsymbol{\mu}$ = new $\boldsymbol{\mu}$
  2. For each **data point**, find the cluster which has the closest $\boldsymbol{\mu}$ point using Pythagorean distance formula

     If necessary, **reassign the data point** to the new closest cluster - subject to constraint that every cluster must have one data point
- The algorithm repeats phases #1 and #2 until no cluster moves and no data point is reassigned
- When the algorithm reaches steady-state the cluster has "**converged**"

Open Lab 2 – kMeansClustering.h

```
class DataPoint
{
public:
    DataPoint() = default;
    DataPoint(double new_x, double new_y);
    double x{};
    double y{};
    Cluster* c{nullptr};
};
```

Every datapoint belongs to just one cluster

```
class Cluster
{
public:
    Cluster(int index);
    double x{};
    double y{};
    string clr{};
    int population{};
    double mean_distance{};
};
```

View Lab 2 – kMeansClustering.h

```
class DataPoint
{
public:
    DataPoint() = default;
    DataPoint(double new_x, double new_y);
    double x{};
    double y{};
    Cluster* c{nullptr};
};
```

Every datapoint has an (x, y) Cartesian coordinate

The μ point (center) of a cluster is the average of the coordinates (x, y) of all the datapoints that belong to that cluster

```
class Cluster
{
public:
    Cluster(int ind
    double x{};
    double y{};
    string clr{};
    int population{};
    double mean_distance{};
};
```

Every cluster has a center (x, y) Cartesian coordinate

View Lab 2 – kMeansClustering.h

```
class DataPoint
{
public:
    DataPoint() = default;
    DataPoint(double new_x, double new_y);
    double x{};
    double y{};
    Cluster* c{nullptr};
};
```

Every cluster has a distinct Allegro color

```
class Cluster
{
public:
    Cluster(int index);
    double x{};
    double y{};
    string clr{};
    int population{};
    double mean_distance{};
};
```

View Lab 2 – kMeansClustering.h

```
class DataPoint
{
public:
    DataPoint() = default;
    DataPoint(double new_x, double new_y);
    double x{};
    double y{};
    Cluster* c{nullptr};
};
```

The number of datapoints belonging to a cluster is its population

```
class Cluster
{
public:
    Cluster(int index);
    double x{};
    double y{};
    string clr{};
    int population{};
    double mean_distance{};
};
```

Open Lab 2 – main.cpp

main.cpp | kMeansClustering.cpp | kMeansClustering.h

```
1   #include "stdafx.h"
2   #include "simplescreen.h"
3   #include "kMeansClustering.h"
4
5   using namespace std;
6
7   int num_clusters{3};
8   double mean_multiple{0};
9
10  vector<DataPoint*>* dataPoints{};
11  vector<Cluster*>* clusters{};
12  bool converged{};
13
14  double GetDistance(double x1, double y1, double x2, double y2 )
15  {
16      return sqrt(pow(x1-x2,2) + pow(y1-y2,2));
17  }
```

Global variables are declared outside the scope of any function

Even k-means uses the Pythagorean theorem!

View Lab 2 – main.cpp

main.cpp | kMeansClustering.cpp | kMeansClustering.h

```
13
14  double GetDistance(double x1, double y1, double x2,
15  {
16      return sqrt(pow(x1-x2,2) + pow(y1-y2,2));
17  }
18
19  void InitDataPoints()
20  {
21      dataPoints = new vector<DataPoint*>();
22      dataPoints->push_back(new DataPoint(23, 35));
23      dataPoints->push_back(new DataPoint(35, 18));
24      dataPoints->push_back(new DataPoint(14, 3));
25      dataPoints->push_back(new DataPoint(17, 6));
26      dataPoints->push_back(new DataPoint(38, 15));
27      dataPoints->push_back(new DataPoint(26, 41));
28      dataPoints->push_back(new DataPoint(27, 38));
29      dataPoints->push_back(new DataPoint(30, 35));
30      dataPoints->push_back(new DataPoint(19, 12));
31      dataPoints->push_back(new DataPoint(24, 32));
32      dataPoints->push_back(new DataPoint(41, 26));
        dataPoints->push_back(new DataPoint(38, 24));
        dataPoints->push_back(new DataPoint(36, 21));
        dataPoints->push_back(new DataPoint(30, 32));
        dataPoints->push_back(new DataPoint(17, 3));
        dataPoints->push_back(new DataPoint(12, 5));
        dataPoints->push_back(new DataPoint(24, 38));
        dataPoints->push_back(new DataPoint(14, 6));
        dataPoints->push_back(new DataPoint(27, 32));
41      dataPoints->push_back(new DataPoint(17, 9));
42      //dataPoints->push_back(new DataPoint(5, 40));
43  }
```

(x, y) coordinates of each data point

This point is a data outlier which we will use later

View Lab 2 – main.cpp

```
void InitClusters()
{
    clusters = new vector<Cluster*>();
    for (int i{}; i < num_clusters; ++i)
        clusters->push_back(new Cluster(i));
    // Assign each data point to an initial
    // cluster in a round-robin fashion
    for (size_t i{}; i < dataPoints->size(); ++i)
    {
        Cluster* c = clusters->at(i % clusters->size());
        dataPoints->at(i)->c = c;
        c->population++;
    }
}
```

Datapoints are assigned to an initial cluster in a **round-robin** fashion

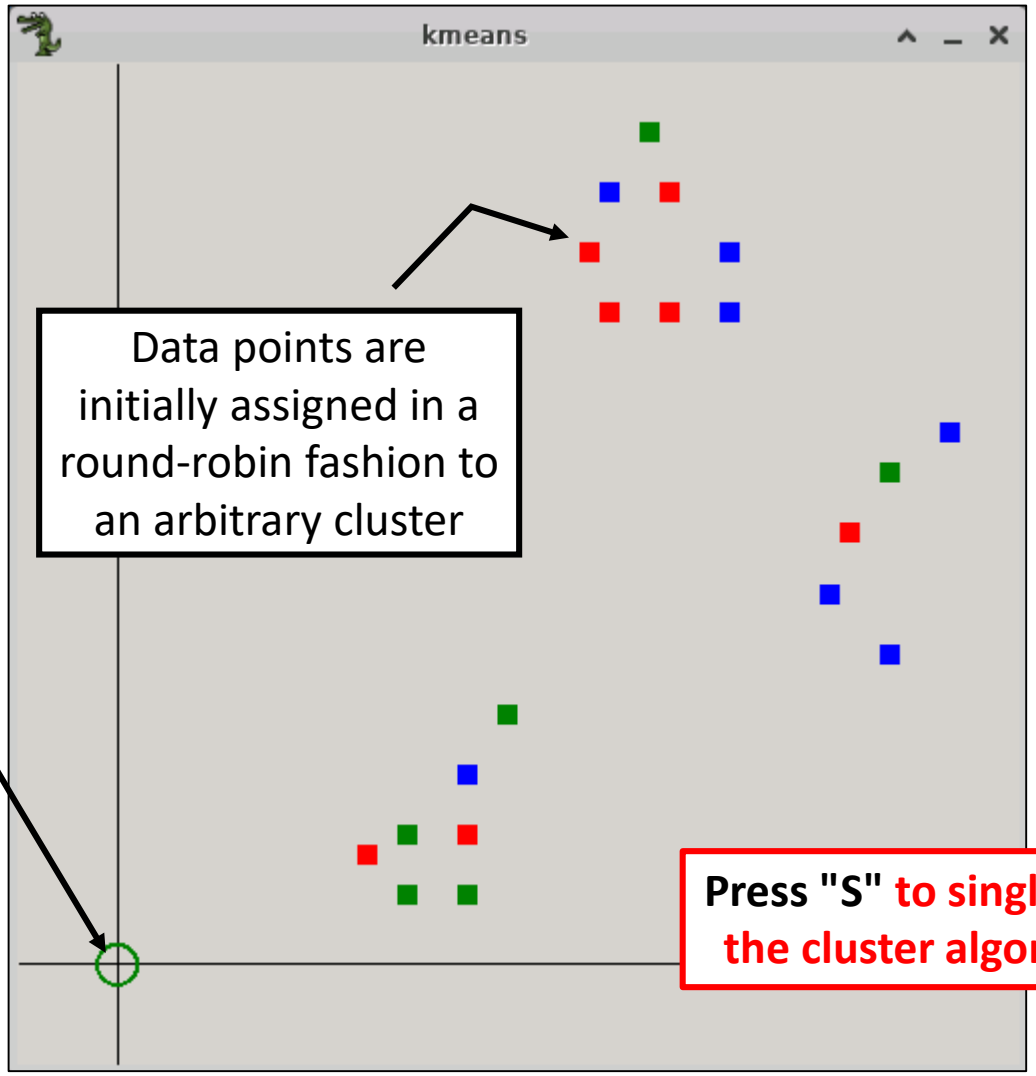
View Lab 2 – main.cpp

```
int main()
{
    SimpleScreen ss(draw, eventHandler
    ss.SetWorldRect(-5, -5, 45, 45);

    cout << "Press S to single step th
    cout << "Press Q to quit the appli

    InitDataPoints();

    InitClusters();

    ss.HandleEvents();

    return 0;
}
```

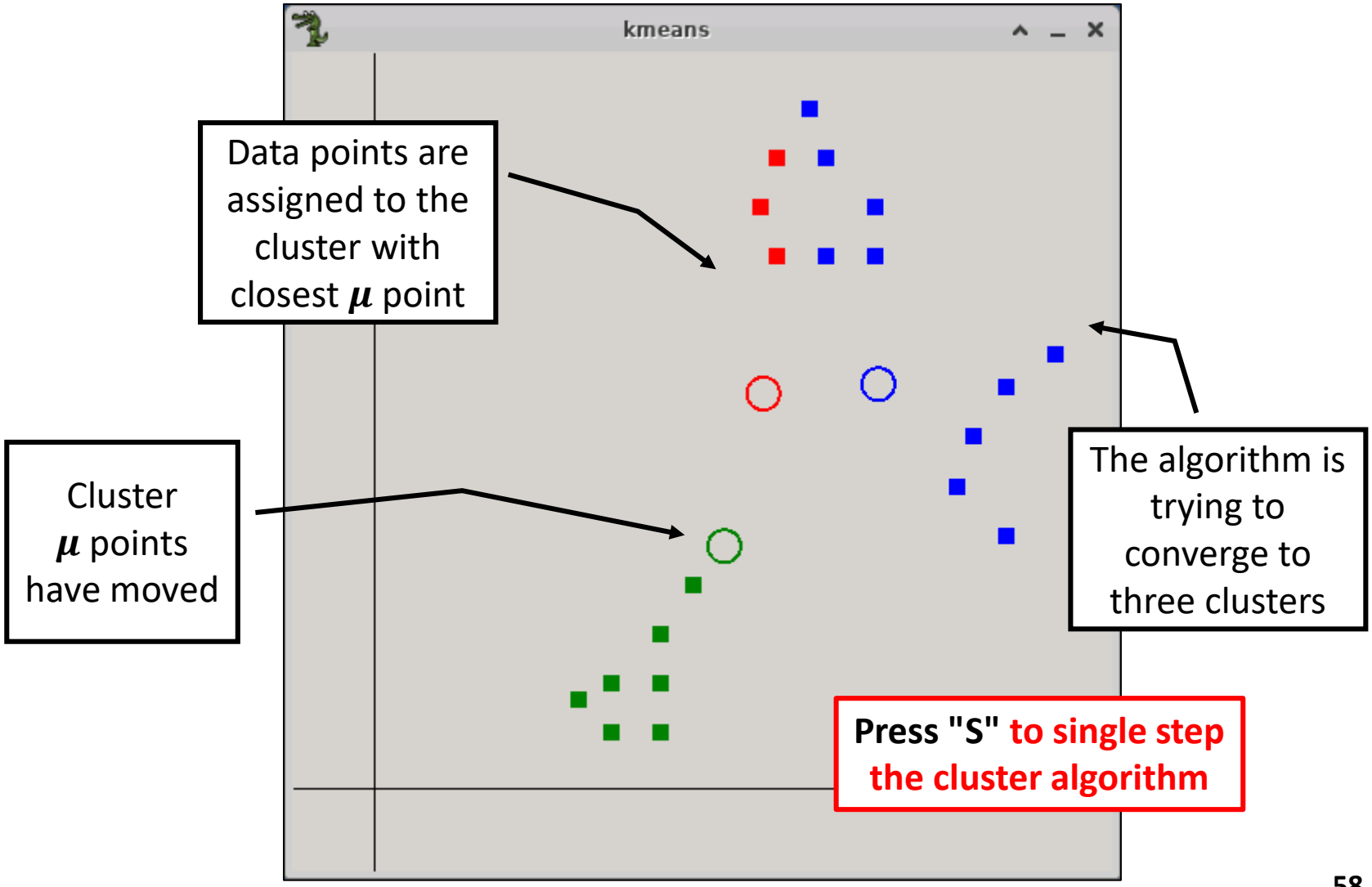
```
void draw(SimpleScreen& ss)
{
    ss.Clear();
    ss.DrawAxes();
    // Draw each data point
    for (auto dp : *dataPoints)
        ss.DrawRectangle(dp->c->clr, dp->x, dp->y, 1, 1, 0, true);
    // Draw each cluster's geometric center (the "mean")
    for (auto c : *clusters)
        ss.DrawCircle(c->x, c->y, 1, c->clr, 2);
    ss.Update();
}
```

```
void eventHandler(SimpleScreen& ss, ALLEGRO_EVENT& ev)
{
    if (ev.type == ALLEGRO_EVENT_KEY_CHAR)
    {
        switch (ev.keyboard.keycode)
        {
        case ALLEGRO_KEY_S:
            UpdateClusters();
            ss.Redraw();
            if (converged)
                cout << "Cluster has converged" << endl;
            break;
        case ALLEGRO_KEY_Q:
            ss.Exit();
            break;
        }
    }
}
```

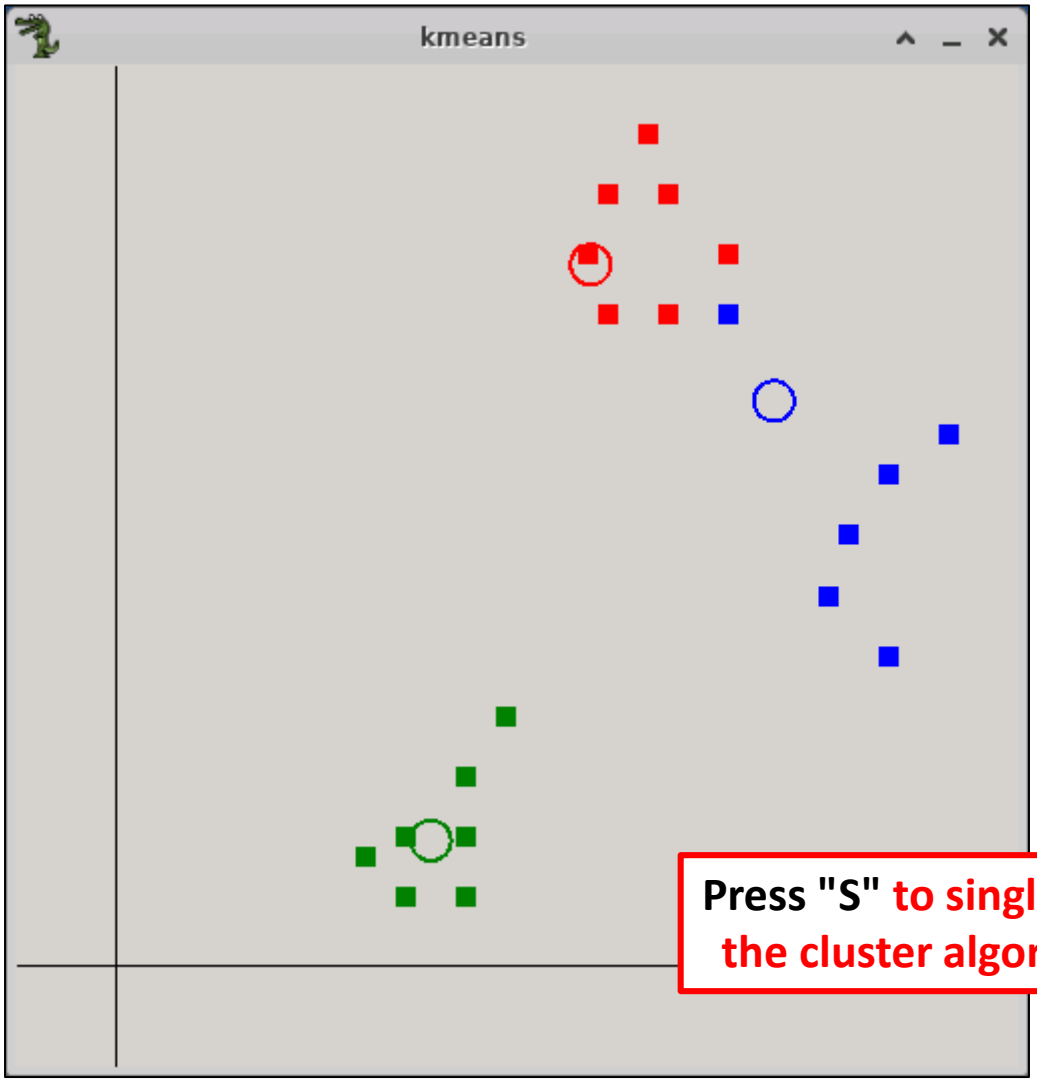
Run Lab 2 - Round 0



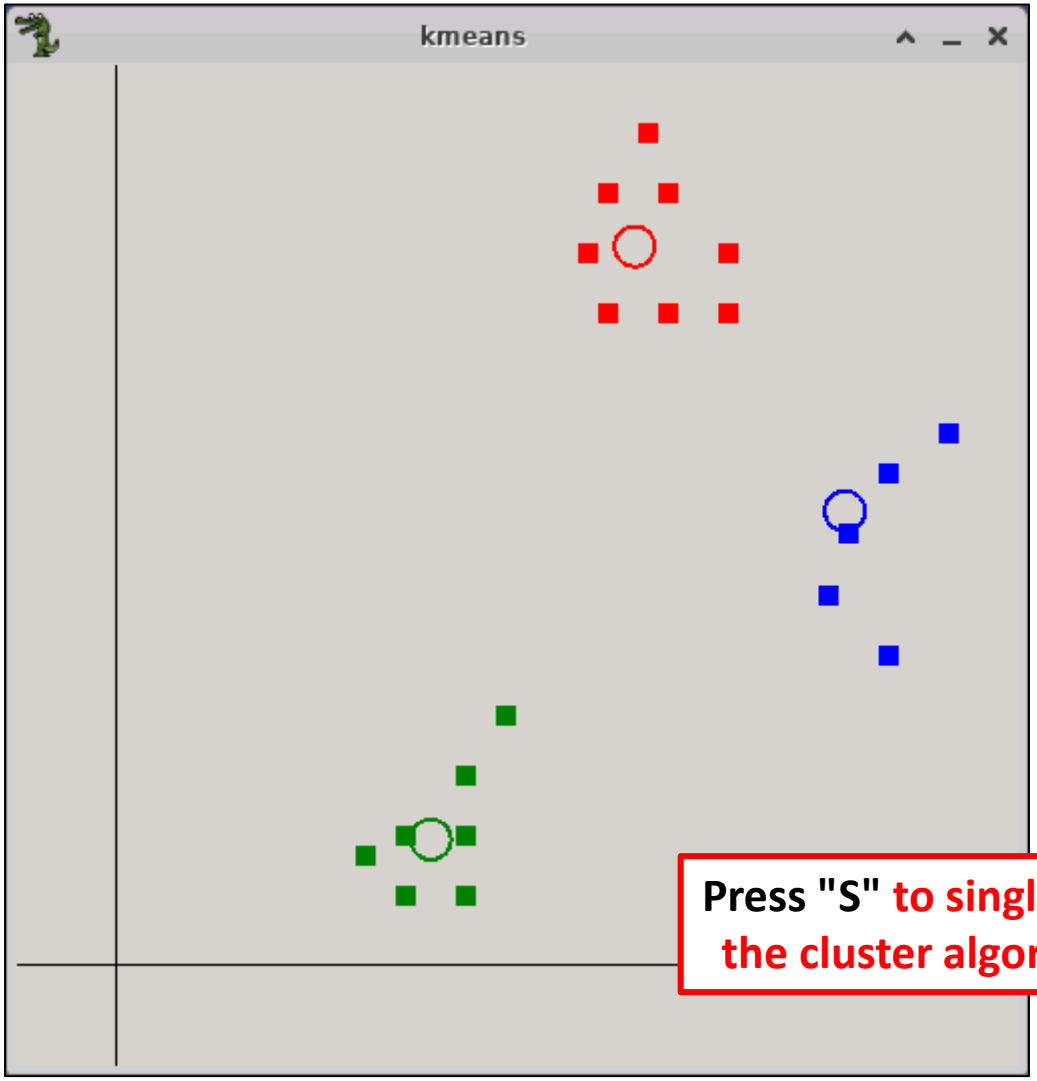
Check Lab 2 - Round 1



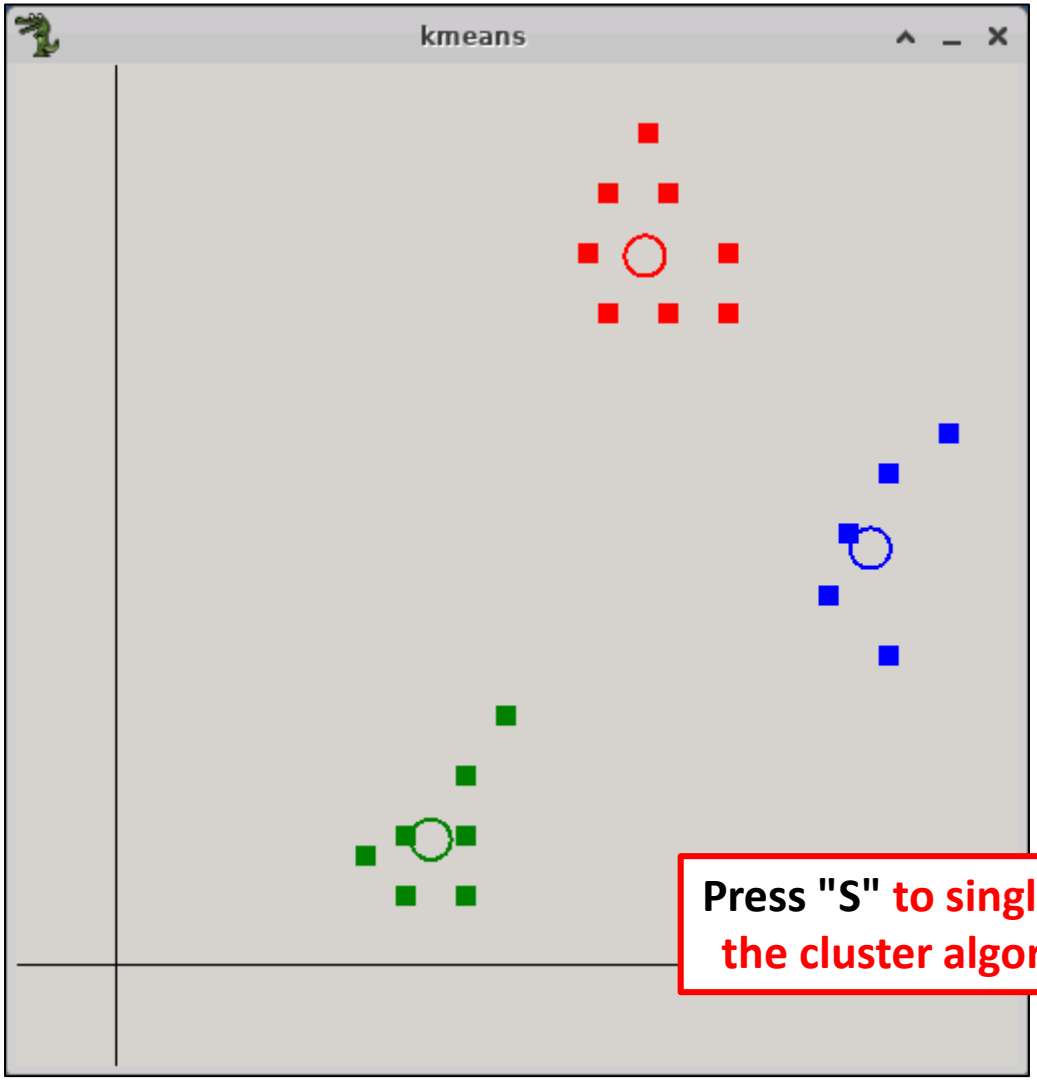
Check Lab 2 - Round 2



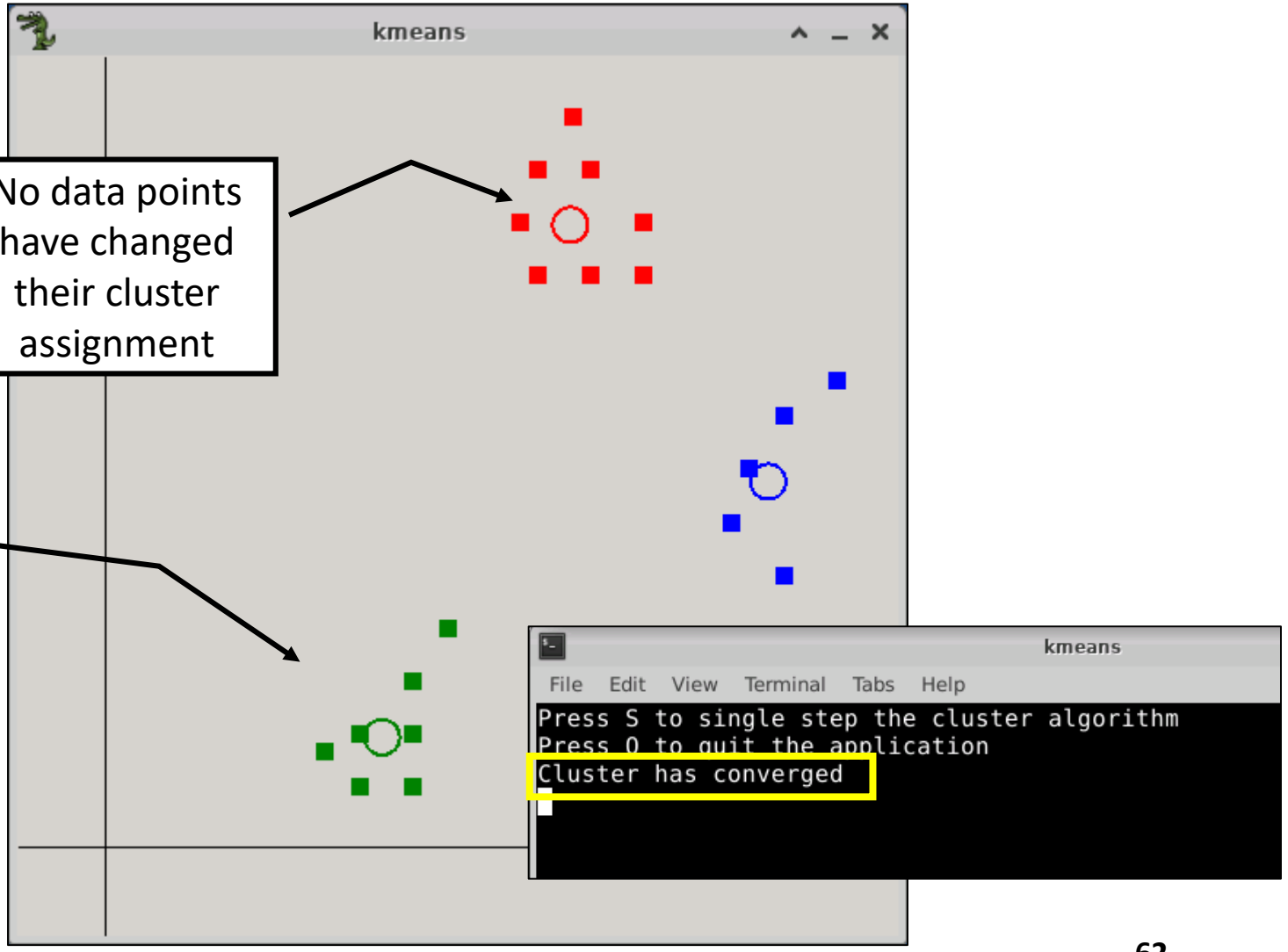
Check Lab 2 - Round 3



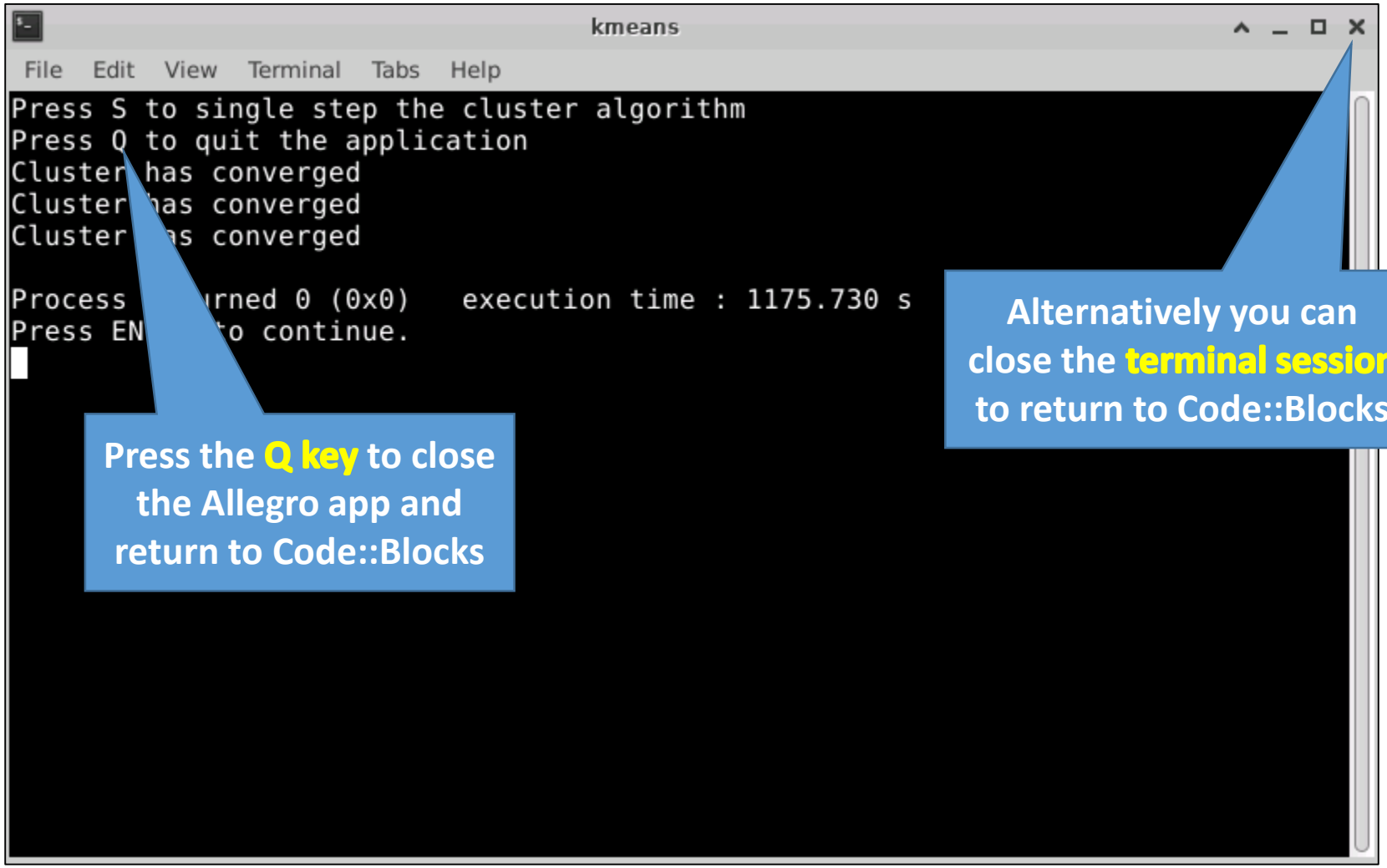
Check Lab 2 - Round 4



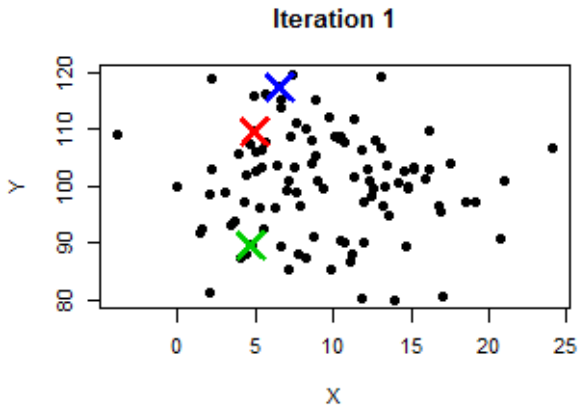
Check Lab 2 - Round 5

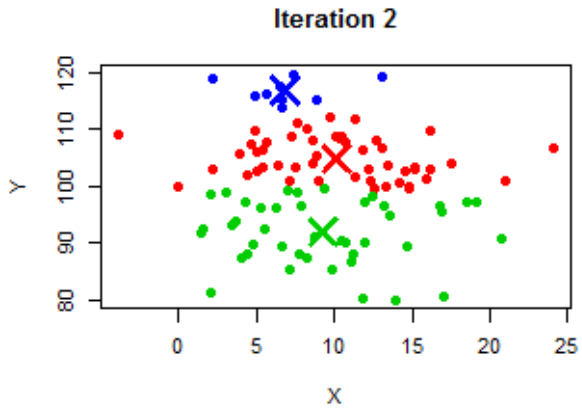


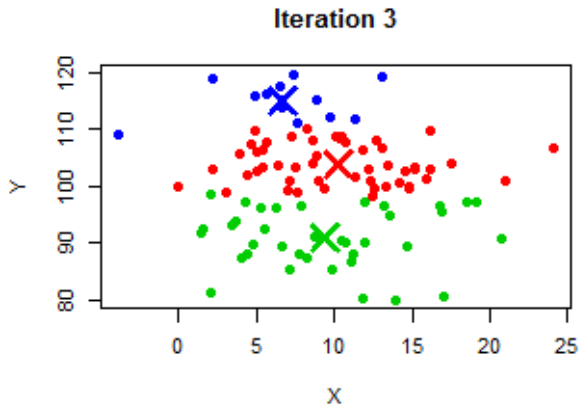
Quit Lab 2 - Round 5

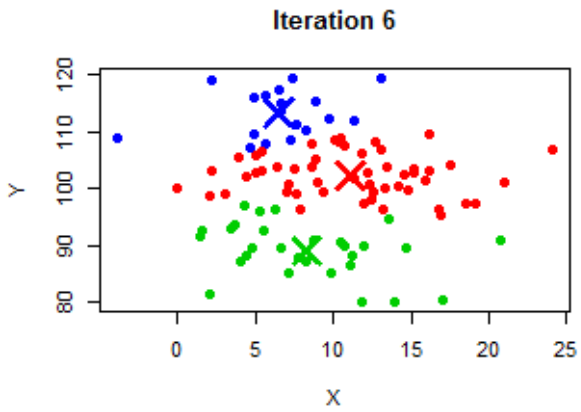


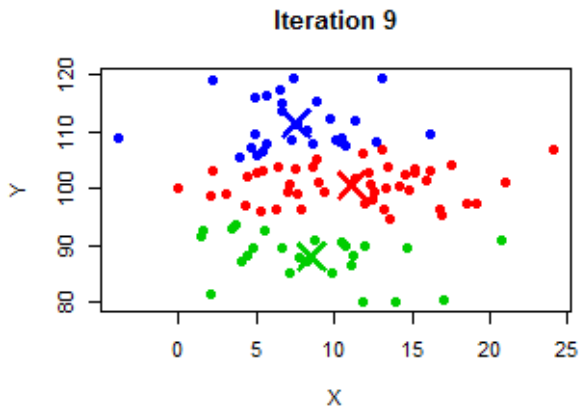
k-means Clustering

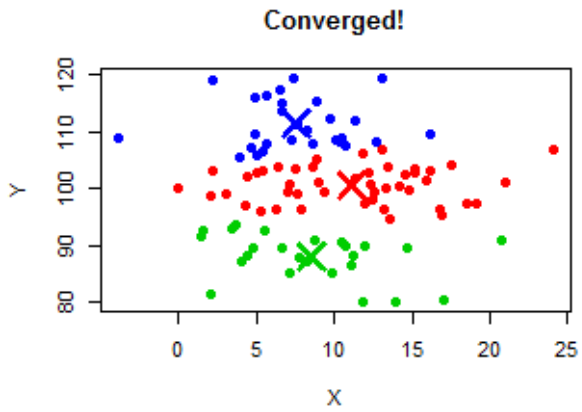




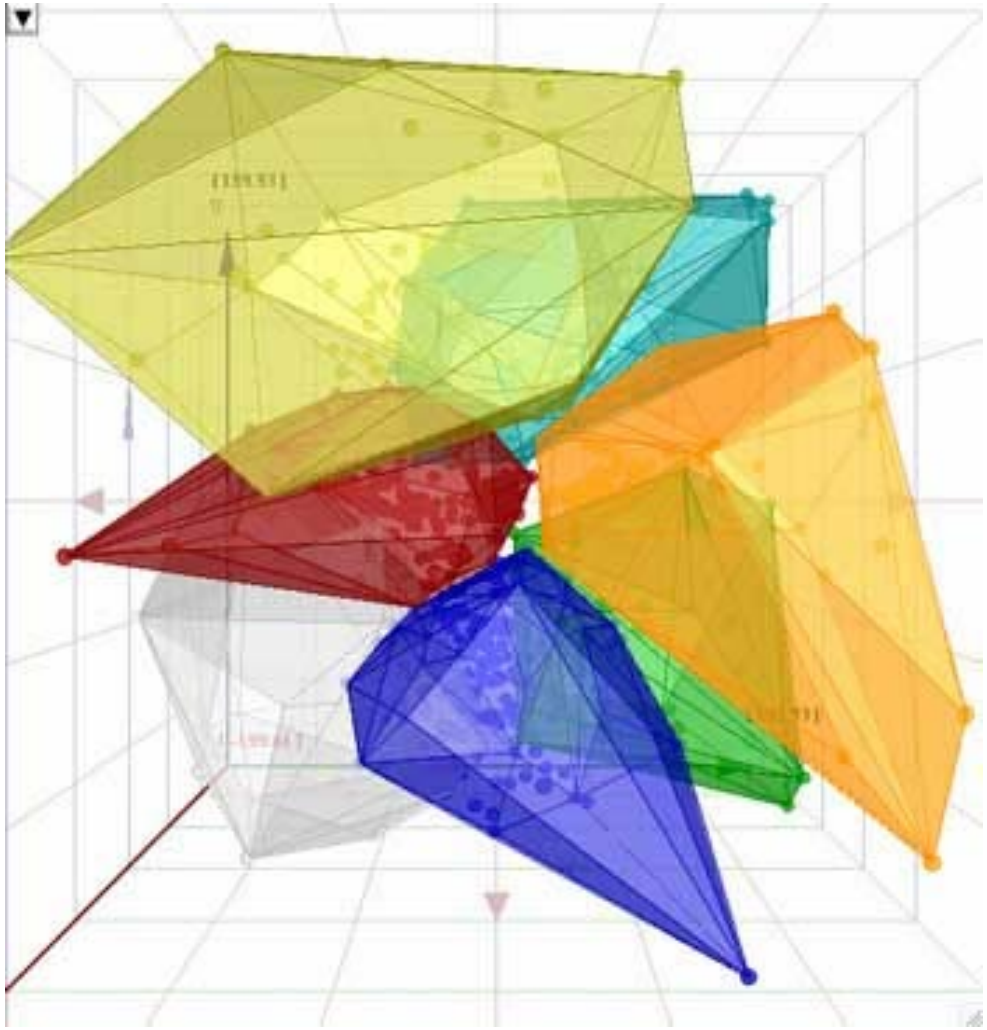








n-Dimensional k-means Clustering

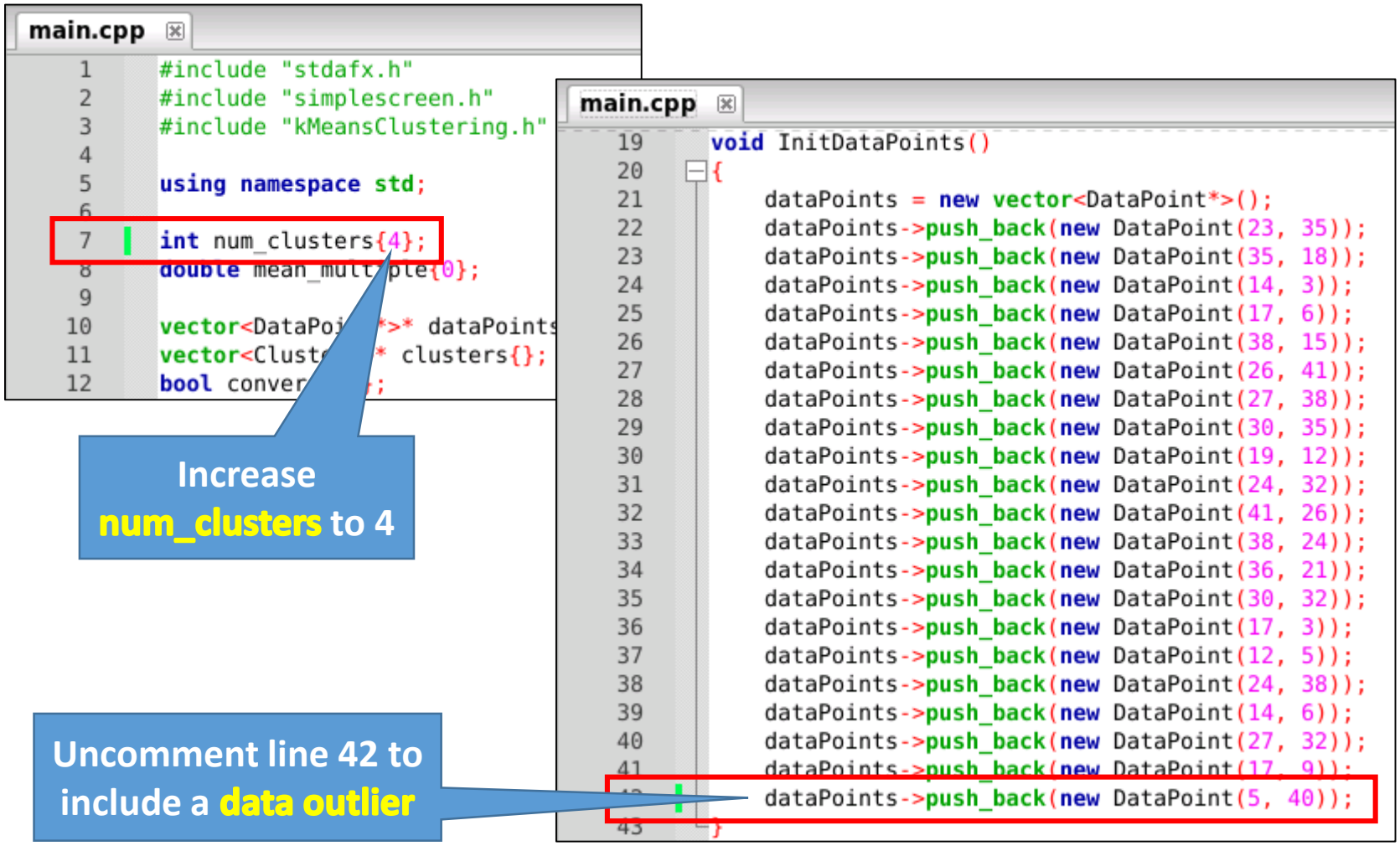


View Lab 2 – main.cpp

```
void UpdateClusters()
{
    // Phase I: Calculate the new geometric mean of each
    // cluster based upon current data point assignments
    bool phase1_change{false};
    for (auto c : *clusters)
    {
        DataPoint newCenter;
        double count{};
        for (auto dp : *dataPoints)
        {
            if (dp->c == c)
            {
                newCenter.x += dp->x;
                newCenter.y += dp->y;
                count++;
            }
        }
        newCenter.x /= count;
        newCenter.y /= count;
        // Move cluster center (mean) if necessary
        if (newCenter.x != c->x || newCenter.y != c->y)
        {
            c->x = newCenter.x;
            c->y = newCenter.y;
            phase1_change = true;
        }
    }
```

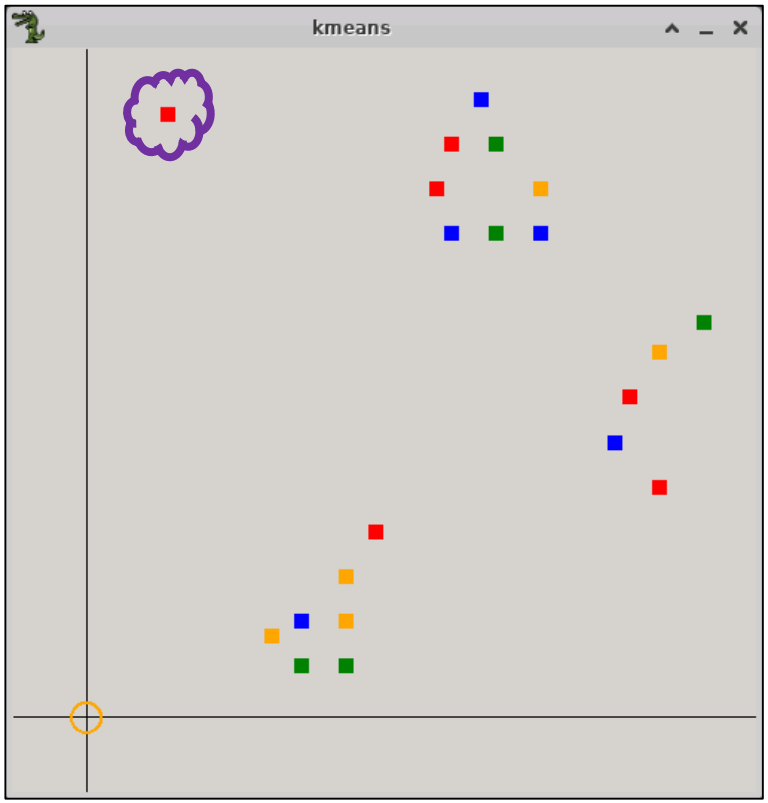
```
    // Phase II: Assign data points to nearest cluster
    bool phase2_change{false};
    for (auto dp : *dataPoints)
    {
        double distMin = numeric_limits<double>::max();
        Cluster* cNearest = nullptr;
        for (auto c : *clusters)
        {
            double dist = GetDistance(dp->x, dp->y, c->x, c->y);
            if (dist < distMin)
            {
                distMin = dist;
                cNearest = c;
            }
        }
        // If the nearest cluster to this point is NOT
        // the point's currently assigned cluster...
        if (cNearest != dp->c)
        {
            // Then reassign data point to the new cluster, but only
            // if this is not the last point the current cluster.
            // Reassignment must never result in an empty cluster
            if (dp->c->population > 1)
            {
                dp->c->population--;
                // Assign point to new (nearer) cluster
                dp->c = cNearest;
                dp->c->population++;
                phase2_change = true;
            }
        }
    }

    if (mean_multiple == 0 && !phase1_change && !phase2_change && !converged)
    {
        converged = true;
    }
```

Edit Lab 2 – main.cpp



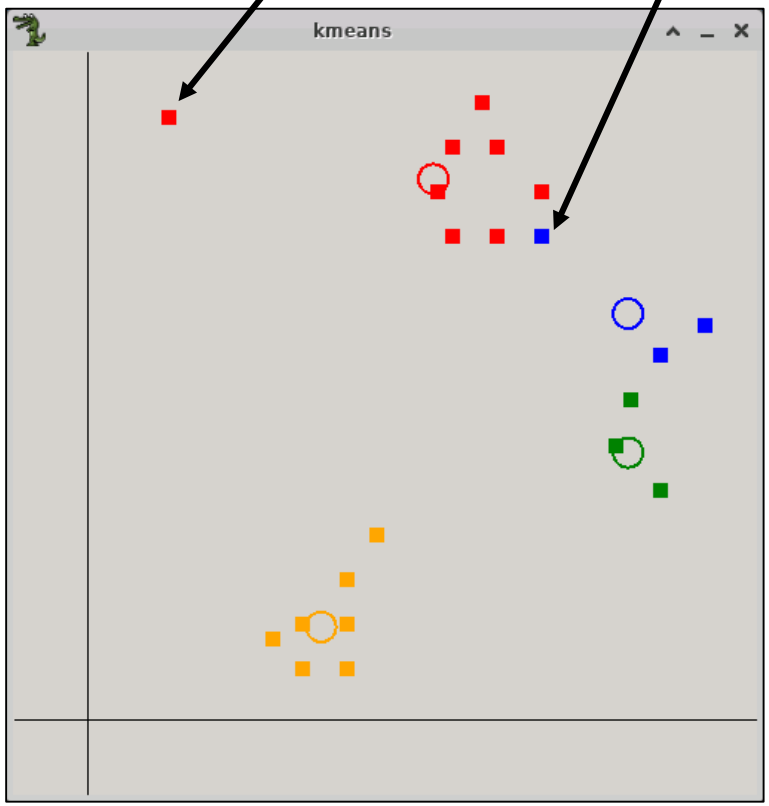
Run Lab 2 – Data Outlier

Press "S" to single step the cluster algorithm



Round 0

The outlier skews the red and blue clusters



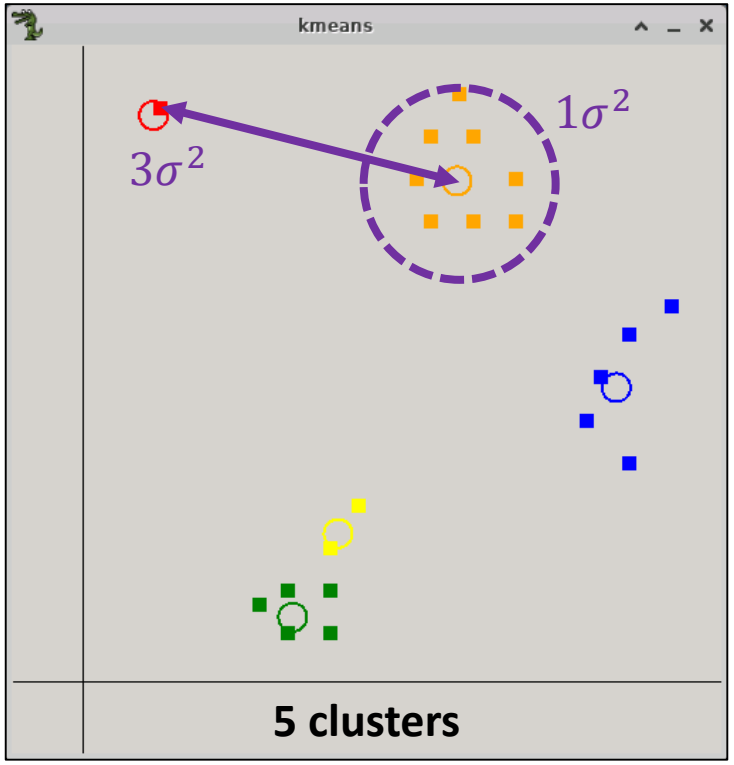
Round 3

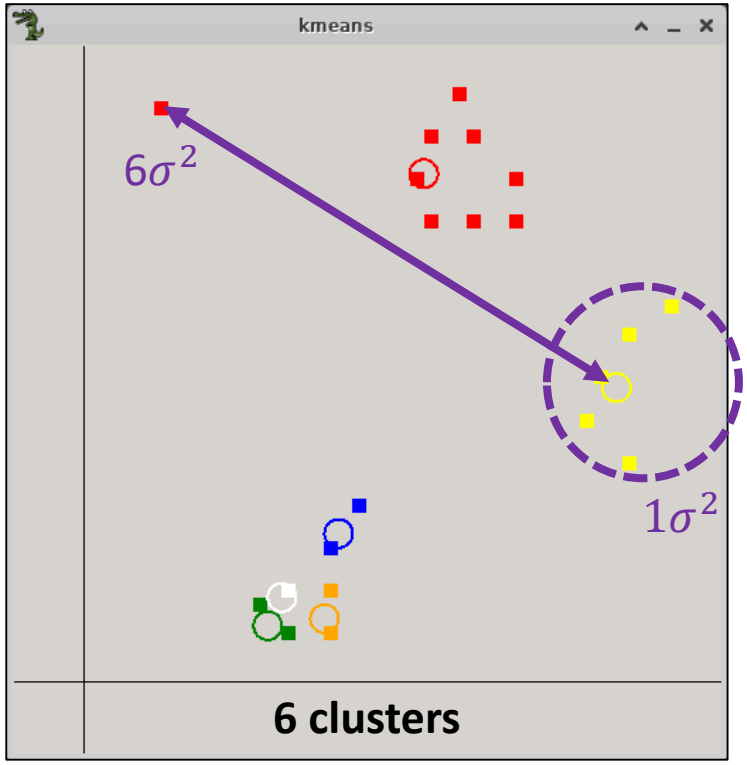
Edit Lab 2 – Research Questions

- Compare convergence with **num_clusters set** to 5 versus 6
- Is using more clusters a key to identifying data outliers?

Identifying Data Outliers

- Could distance **variance** ($\boldsymbol{\sigma^2}$) be used to gauge if a datapoint *reasonably* belongs to a given cluster?

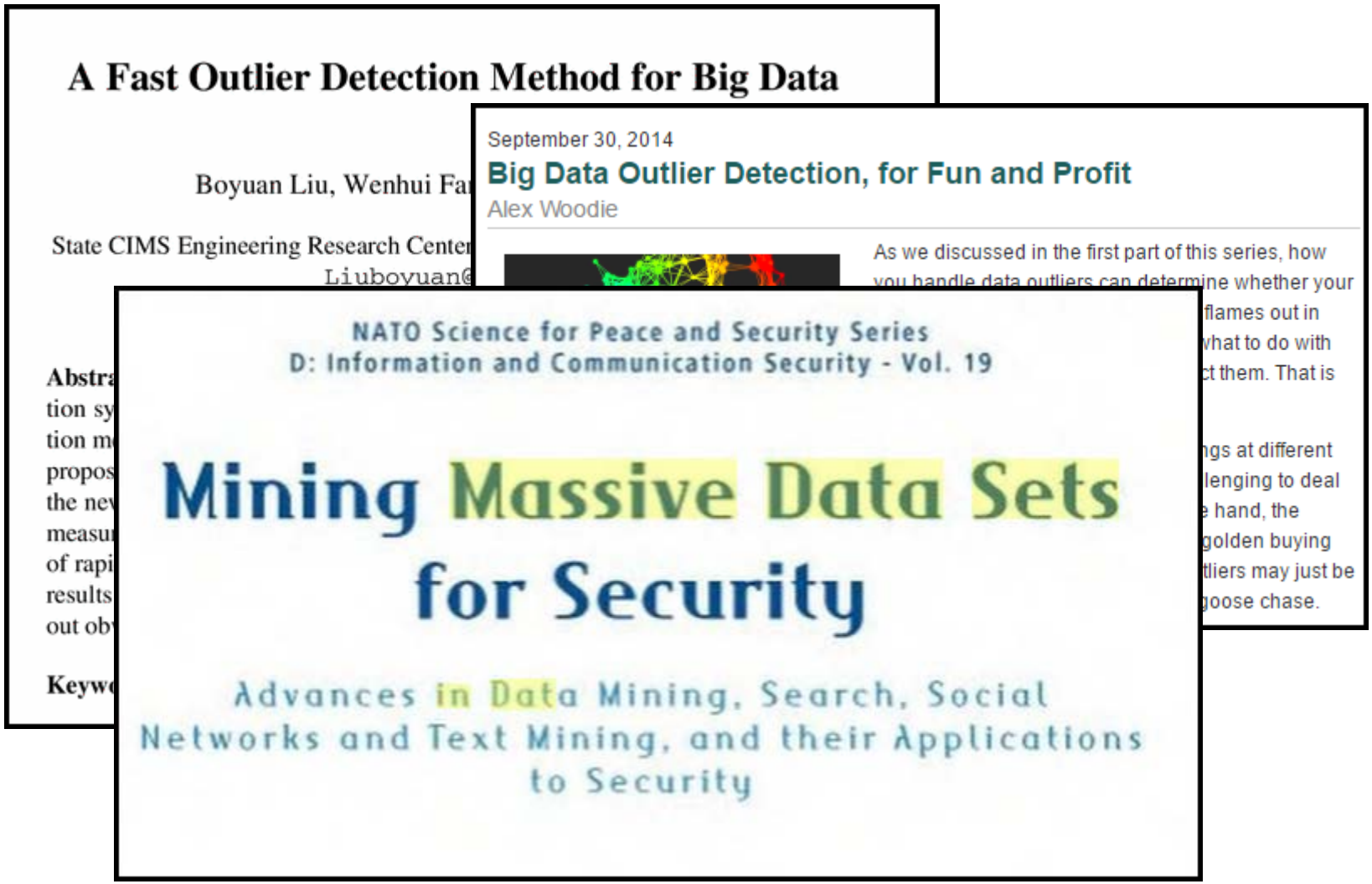




Identifying Data Outliers

- Once we have data partitioned into clusters, we can eliminate outliers by **evicting** any data points whose distance from its converged cluster's $\boldsymbol{\mu}$ point is *too large*
  - The data points should be distributed around the cluster's $\boldsymbol{\mu}$ point in a normal distribution. This means that **99.97%** of data points should be $\leq \boldsymbol{3\sigma}$ from the mean (geometric center) of the cluster
  - If a data point is $> \boldsymbol{3\sigma}$ distance from the $\boldsymbol{\mu}$ point of its cluster, it probably is an outlier and should be evicted
  - After we evict a data point, we iterate the algorithm all over again
- If a data point winds up getting evicted from ***all*** clusters, it is probably a **data capture error** or a **statistical anomaly**

Identifying Data Outliers



Now you know…

- We can use **k-means clustering** to categorize data points by its nearness to a similar group's statistical means
  - Data sets which are homogenous (have a more uniform distribution of values) **may not converge** to a meaningful set of clusters since the data lacks *sharpness*
  - We can use k-means **to identify data outliers** so we can avoid allowing those anomalies from skewing the main observations – this process is called evicting a point from the cluster
- We can extend k-means clustering to analyze **multi-dimensional data sets** to look for trends
  - This is exactly how Amazon and Netflix provide you purchase recommendations based upon your shared interests & tastes compared with other buyers – a **very effective** selling technique!