MONASH College

MCD4710

Introduction to algorithms and programming

Lecture 18

Gaussian Elimination

Overview

- Represent problem as system of linear equations
- Represent linear system in matrix/vector form
- Apply and implement backward substitution to solve upper-triangular systems
- Apply and implement forward elimination to transform general system into triangular form

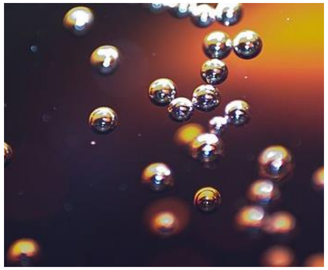
Linear systems of equations

Some Chemistry

ferric oxide
Fe_2O_3

carbon
C

iron
Fe



carbon dioxide
CO_2

Some Chemistry

ferric oxide

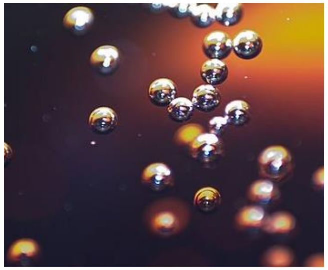
Fe_2O_3

carbon

C

iron

Fe



carbon dioxide

CO_2

Chemical notation

$$Fe_2O_3 + C \rightarrow Fe + CO_2$$

reactants

products

MONASH College

Some Chemistry

ferric oxide

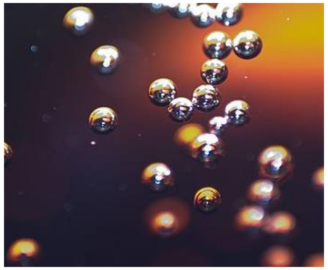
Fe_2O_3

carbon

C

iron

Fe



carbon dioxide

CO_2

Chemical notation

$$Fe_2O_3 + C \rightarrow Fe + CO_2$$

Count atoms

Atom	Reactants	Products
Fe	2	1
O	3	2
C	1	1

Numbers don't add up!

Want to *balance* chemical equation

Unbalanced equation

$$Fe_2O_3 + C \rightarrow Fe + CO_2$$

Balanced equation

$$2Fe_2O_3 + 3C \rightarrow 4Fe + 3CO_2$$

Count atoms

Atom	Reactants	Products
Fe	4	4
O	6	6
C	3	3

MONASH College

Let's balance equation *algorithmically*

General equation

$$a\mathrm{Fe_2O_3} + b\mathrm{C} \rightarrow c\mathrm{Fe} + d\mathrm{CO_2}$$

Conditions on solution

$$2a = c \qquad \text{(Fe)}$$

Let's balance equation *algorithmically*

General equation

$$aFe_2O_3 + bC \rightarrow cFe + dCO_2$$

Conditions on solution

$$2a = c \qquad \text{(Fe)}$$

$$3a = 2d \qquad \text{(O)}$$

Let's balance equation *algorithmically*

General equation

$$a\mathrm{Fe_2O_3} + b\mathrm{C} \rightarrow c\mathrm{Fe} + d\mathrm{CO_2}$$

Conditions on solution

$$2a = c \qquad (\mathrm{Fe})$$

$$3a = 2d \qquad (\mathrm{O})$$

$$b = d \qquad (\mathrm{C})$$

Let's balance equation *algorithmically*

General equation

$$a\mathrm{Fe_2O_3} + b\mathrm{C} \rightarrow c\mathrm{Fe} + d\mathrm{CO_2}$$

Conditions on solution

$$2a = c \qquad \text{(Fe)}$$

$$3a = 2d \qquad \text{(O)}$$

$$b = d \qquad \text{(C)}$$

Does that uniquely determine solution?

Let’s balance equation *algorithmically*

General equation

$$aFe_2O_3 + bC \rightarrow cFe + dCO_2$$

Conditions on solution

$$2a = c \qquad \text{(Fe)}$$
$$3a = 2d \qquad \text{(O)}$$
$$b = d \qquad \text{(C)}$$

$$4Fe_2O_3 + 6C \rightarrow 8Fe + 6CO_2$$

Atom	Reactants	Products
Fe	8	8
O	12	12
C	6	6

double quantities

$$2Fe_2O_3 + 3C \rightarrow 4Fe + 3CO_2$$

Atom	Reactants	Products
Fe	4	4
O	6	6
C	3	3

Let's balance equation *algorithmically*

General equation

$$aFe_2O_3 + bC \rightarrow cFe + dCO_2$$

Conditions on solution

$$2a = c \qquad \text{(Fe)}$$

$$3a = 2d \qquad \text{(O)}$$

$$b = d \qquad \text{(C)}$$

$$a = 4$$

$$4Fe_2O_3 + 6C \rightarrow 8Fe + 6CO_2$$

Atom	Reactants	Products
Fe	8	8
O	12	12
C	6	6

Fix quantities to fully determine solution

Reached a very general problem

<u>Solve Linear Systems</u>

Input: Set of n linear equations in n variables

Output: Assignment of values to variables satisfying all equations

How to represent this problem in Python?

Finding systematic representation

Solve Linear Systems

Input: Set of n linear equations in n variables

Output: Assignment of values to variables satisfying all equations

$$\begin{aligned} 2a &= c \\ 3a &= 2d \\ b &= d \\ a &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 2a - c &= 0 \\ 3a - 2d &= 0 \\ b - d &= 0 \\ a &= 4 \end{aligned} \quad \Rightarrow \quad \begin{aligned} 2a + 0b - 1c + 0d &= 0 \\ 3a + 0b + 0c - 2d &= 0 \\ 0a + 1b + 0c - 1d &= 0 \\ 1a + 0b + 0c + 0d &= 4 \end{aligned}$$

coefficient matrix $\boldsymbol{A}$

$$\begin{bmatrix} 2 & 0 & -1 & 0 \\ 3 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}$$

right hand side $\boldsymbol{b}$

solution vector $\boldsymbol{x}$

Finding systematic representation

<u>Solve Linear Systems</u>

Input: $n \times n$-matrix $\boldsymbol{A}$ of coefficients, n-dim vector $\boldsymbol{b}$

Output: n-dim vector $\boldsymbol{x}$ such that $\boldsymbol{Ax} = \boldsymbol{b}$

Python table

Python list

coefficient matrix $\boldsymbol{A}$

$$\begin{bmatrix} 2 & 0 & -1 & 0 \\ 3 & 0 & 0 & -2 \\ 0 & 1 & 0 & -1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \end{bmatrix}$$

right hand side $\boldsymbol{b}$

solution vector $\boldsymbol{x}$

Simple case: triangular system

Overall strategy: Transform and Conquer Paradigm

another input

or

problem input $\xrightarrow{\text{transform}}$ another representation $\xrightarrow{\text{conquer}}$ solution

or

another problem

What are easy to solve linear systems?

(Upper) triangular systems

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 6 \end{bmatrix}$$

called *upper triangular*, because only coefficients in main diagonal and above are non-zero

How can we solve such a system?

Back-substitution algorithm

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 6 \end{bmatrix}$$

$$2c = 6$$

$$c = 6/2$$

Back-substitution algorithm

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 6 \end{bmatrix}$$

$$b + 3c = 7$$

$$b = 7 - 9$$

Back-substitution algorithm

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ -2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 6 \end{bmatrix}$$

$$a + 2b + c = 2$$

$$a = 2 + 4 - 3$$

Back-substitution algorithm

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ -2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 6 \end{bmatrix}$$

Back-substitution in general

$$\begin{bmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} \\ 0 & u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & 0 & u_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

```
def solve_by_back_sub(u,b):
    n = len(b)
    x = n*[0]
    for i in range(n-1, -1, -1):
        # find value x[i]

    return x
```

Back-substitution algorithm

$$\begin{bmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} \\ 0 & u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & 0 & u_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$u_{i,i}x_i + \underbrace{(u_{i,i+1}x_{i+1} + \cdots + u_{i,n-1}x_{n-1})}_{s} = b_i$$

$$x_i = (b_i - s)/u_{i,i}$$

```
def solve_by_back_sub(u,b):
    n = len(b)
    x = n*[0]
    for i in range(n-1, -1, -1):
        # find value x[i]

    return x
```

Back-substitution algorithm

$$\begin{bmatrix} u_{0,0} & u_{0,1} & u_{0,2} & u_{0,3} \\ 0 & u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & 0 & u_{2,2} & u_{2,3} \\ 0 & 0 & 0 & u_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$u_{i,i}x_i + \underbrace{(u_{i,i+1}x_{i+1} + \cdots + u_{i,n-1}x_{n-1})}_{s} = b_i$$

$$x_i = (b_i - s)/u_{i,i}$$

```
def solve_by_back_sub(u,b):
    n = len(b)
    x = n*[0]
    for i in range(n-1, -1, -1):
        s = sum(x[j] * u[i][j] for j in range(n-1, i, -1))
        x[i] = (b[i] - s)/u[i][i]
    return x
```

Comprehension allows to directly express our formula

MONASH College

Back-substitution algorithm

```
def solve_by_back_sub(u,b):
    n = len(b)
    x = n*[0]
    for i in range(n-1, -1, -1):
        s = 0
        for j in range(n-1, i, -1):
            s += u[i][j] * x[j]
        x[i] = (b[i] - s)/u[i][i]
    return x
```

$$u_{i,i}x_i + \underbrace{(u_{i,i+1}x_{i+1} + \cdots + u_{i,n-1}x_{n-1})}_{s} = b_i$$

for-loop versus comprehension

```
def solve_by_back_sub(u,b):
    n = len(b)
    x = n*[0]
    for i in range(n-1, -1, -1):
        s = sum(x[j] * u[i][j] for j in range(n-1, i, -1))
        x[i] = (b[i] - s)/u[i][i]
    return x
```

Gaussian Elimination

Carl Friedrich Gauß
1777– 1855

general system → triangular system

$$\begin{bmatrix}1 & 2 & 1\\ -5 & -9 & -2\\ 2 & 3 & 1\end{bmatrix},\begin{bmatrix}2\\ -3\\ 3\end{bmatrix} \xrightarrow[\text{forward elimination}]{\text{transform}} \begin{bmatrix}1 & 2 & 1\\ 0 & 1 & 3\\ 0 & 0 & 2\end{bmatrix},\begin{bmatrix}2\\ 7\\ 6\end{bmatrix} \xrightarrow[\text{back substitution}]{\text{conquer}} \begin{bmatrix}3\\ 2\\ 3\end{bmatrix}$$

How to *eliminate* coefficients?

$$a + 2b + c = 2$$
$$-5a - 9b - 2c = -3$$

$$\Downarrow$$

$$-4a - 7b - c = -1$$

Observation:
Given equations imply (infinitely) many other equations

How to *eliminate* coefficients?

$$a + 2b + c = 2$$
$$-5a - 9b - 2c = -3$$

adding 5 times first equation to second yields equations where a-coefficient is *eliminated*

$$\Downarrow$$

$$b - 3c = 7$$

Observation:
Given equations imply (infinitely) many other equations

How to *eliminate* coefficients?

$$\begin{aligned} a + 2b + \ \ c &= 2 \\ -5a - 9b - 2c &= -3 \\ 2a + 3b + \ \ c &= 3 \end{aligned}$$

$$\Downarrow \ \Uparrow$$

Simpler system but same solutions

$$\begin{aligned} a + 2b + \ \ c &= 2 \\ b - 3c &= 7 \\ 2a + 3b + \ \ c &= 3 \end{aligned}$$

Observation:
Given equations imply (infinitely) many other equations

How to *eliminate* coefficients?

$$\begin{aligned} a + 2b + \ \ c &= 2 \\ -5a - 9b - 2c &= -3 \\ 2a + 3b + \ \ c &= 3 \end{aligned} \qquad \begin{bmatrix} 1 & 2 & 1 \\ -5 & -9 & -2 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 3 \end{bmatrix}$$

$$\Downarrow \ \Uparrow \qquad\qquad \Downarrow \ \Uparrow$$

$$\begin{aligned} a + 2b + \ \ c &= 2 \\ b - 3c &= 7 \\ 2a + 3b + \ \ c &= 3 \end{aligned} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & -3 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}$$

Observations:

- Solution *invariant* when adding multiple of row to another
- Exploit to get simpler but equivalent system

Iteratively eliminate below diagonal coefficients

simplify column 1

$$\begin{bmatrix} 1 & 2 & 1 \\ -5 & -9 & -2 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 3 \end{bmatrix}$$

$$\Updownarrow$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 2 & 3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 3 \end{bmatrix}$$

$$\Updownarrow$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ -1 \end{bmatrix}$$

simplify column 2

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & -1 & -1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ -1 \end{bmatrix}$$

$$\Updownarrow$$

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 2 \\ 7 \\ 6 \end{bmatrix}$$

General forward elimination

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & \boxed{a_{1,1}} & a_{1,2} & a_{1,3} \\ 0 & \boxed{a_{2,1}} & a_{2,2} & a_{2,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Subtract q times *pivot* row j from row i

How to choose q such that $a_{i,j}$ will be eliminated?

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        for i in range(j + 1, n):
            # eliminate entry a[i][j]


    return a, b
```

MONASH College

General forward elimination

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Subtract q times *pivot* row j from row i

$$q = a_{i,j}/a_{j,j}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & 0 & a_{2,2} - qa_{1,2} & a_{2,3} - qa_{1,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 - qb_1 \\ b_3 \end{bmatrix}$$

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        for i in range(j + 1, n):
            # eliminate entry a[i][j]


    return a, b
```

$$a_{i,m} - a_{j,m}\frac{a_{i,j}}{a_{j,j}}$$

$$a_{i,j} - a_{j,j}\frac{a_{i,j}}{a_{j,j}} = 0$$

General forward elimination

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a_{2,1} & a_{2,2} & a_{2,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Subtract q times *pivot* row j from row i

$$q = a_{i,j}/a_{j,j}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & 0 & a_{2,2} - q a_{1,2} & a_{2,3} - q a_{1,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 - q b_1 \\ b_3 \end{bmatrix}$$

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        for i in range(j + 1, n):
            q = a[i][j] / a[j][j]
            # subtract q*a[j] from a[i]
            # subtract q*b[j] from b[j]
    return a, b
```

MONASH College

General forward elimination

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & \boxed{a_{1,1}} & a_{1,2} & a_{1,3} \\ 0 & \boxed{a_{2,1}} & a_{2,2} & a_{2,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Subtract q times *pivot* row j from row i

$$q = a_{i,j}/a_{j,j}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ 0 & a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & 0 & a_{2,2} - q a_{1,2} & a_{2,3} - q a_{1,3} \\ 0 & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 - q b_1 \\ b_3 \end{bmatrix}$$

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        for i in range(j + 1, n):
            q = a[i][j] / a[j][j]
            a[i] = [a[i][m] - q*a[j][m] for m in range(n)]
            b[i] = b[i] - q*b[j]
    return a, b
```

One technical problem remains

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ -1 & -1 & 1 & 0 \\ 2 & 3 & -1 & 1 \\ -1 & 2 & -2 & 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 3 & -2 & 2 \end{bmatrix}, \begin{bmatrix} 1 \\ -2 \\ 1 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & -1 & -1 \\ 0 & 3 & -2 & 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -2 \\ 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 5 \end{bmatrix}, \begin{bmatrix} 1 \\ -2 \\ 1 \\ 7 \end{bmatrix}$$

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        for i in range(j + 1, n):
            q = a[i][j] / a[j][j]
            a[i] = [a[i][m] - q*a[j][m] for m in range(n)]
            b[i] = b[i] - q*b[j]
    return a, b
```

Can always flip equations; their order doesn't matter

Need to find pivot row

$$\begin{bmatrix}1 & 1 & 0 & 1\\ 0 & 0 & 1 & 1\\ 0 & 1 & -1 & -1\\ 0 & 3 & -2 & 2\end{bmatrix}, \begin{bmatrix}1\\ 1\\ -2\\ 1\end{bmatrix} \Leftrightarrow \begin{bmatrix}1 & 1 & 0 & 1\\ 0 & 1 & -1 & -1\\ 0 & 0 & 1 & 1\\ 0 & 3 & -2 & 2\end{bmatrix}, \begin{bmatrix}1\\ -2\\ 1\\ 1\end{bmatrix}$$

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        # find index k of valid pivot row
        # swap row j and k of system

        for i in range(j + 1, n):
            q = a[i][j] / a[j][j]
            a[i] = [a[i][m] - q*a[j][m] for m in range(n)]
            b[i] = b[i] - q*b[j]
    return a, b
```

Need to find pivot row

$$\begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & -1 & -1 \\ 0 & 3 & -2 & 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -2 \\ 1 \end{bmatrix} \Leftrightarrow \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & 1 & 1 \\ 0 & 3 & -2 & 2 \end{bmatrix}, \begin{bmatrix} 1 \\ -2 \\ 1 \\ 1 \end{bmatrix}$$

Assuming a non-zero pivot exists

```
def pivot_index(a,j):
    max_abs = abs(a[j][j])
    pindex = j
    for i in range(j+1,len(a)):
        if max_abs < abs(a[i][j]):
            pindex = i
    return pindex
```

```
def triangular(a, b):
    n = len(a)
    for j in range(n):
        k = pivot_index(a, j)
        a[j], a[k] = a[k], a[j]
        b[j], b[k] = b[k], b[j]
        for i in range(j + 1, len(a)):
            q = a[i][j] / a[j][j]
            a[i] = [a[i][m] - q*a[j][m] for m in range(n)]
            b[i] = b[i] - q*b[j]
    return a, b
```

Finishing touches

```
def solve_gauss_elim(a, b):
    u, c = triangular(a, b)
    return solve_backsub(u, c)
```

```
def triangular(a, b):
    u, c = deepcopy(a), deepcopy(b)
    n = len(u)
    for j in range(n):
        k = pivot_index(u, j)
        u[j], u[k] = u[k], u[j]
        c[j], c[k] = c[k], c[j]
        for i in range(j + 1, len(a)):
            q = u[i][j] / u[j][j]
            u[i] = [u[i][m] - q*u[j][m] for m in range(n)]
            c[i] = c[i] - q*c[j]
    return u, c
```

Operate on copies to not destroy input

Brief analysis

```
def solve_gauss_elim(a, b):
    u, c = triangular(a, b)
    return solve_backsub(u, c)
```

```
def triangular(a, b):
    u, c = deepcopy(a), deepcopy(b)
    n = len(u)
    for j in range(n):
        k = pivot_index(u, j)
        u[j], u[k] = u[k], u[j]
        c[j], c[k] = c[k], c[j]
        for i in range(j + 1, len(a)):
            q = u[i][j] / u[j][j]
            u[i] = [u[i][m] - q*u[j][m] for m in range(n)]
            c[i] = c[i] - q*c[j]
            #INV: u, c has same solution(s) as a, b
    return u, c
```

$O(n)$

$O(n^2)$

$O(n^3)$

What is the transform step in Gaussian elimination?

A. Forward elimination
B. Backward elimination
C. Forward substitution
D. Back substitution

1. Visit https://flux.qa
2. Log in using your Authcate details (not required if you're already logged in to Monash)
3. Touch the + symbol and enter the code: HHYBXU
4. Answer questions when they pop up.

What is the conquer step in Gaussian elimination?

A. Forward elimination
B. Backward elimination
C. Forward substitution
D. Back substitution

1. Visit https://flux.qa
2. Log in using your Authcate details (not required if you're already logged in to Monash)
3. Touch the + symbol and enter the code: HHYBXU
4. Answer questions when they pop up.

Summary

- Systems of linear equations can represent real-world problems with linear dependencies between variables
- Triangular systems can be solved by back-substitution
- General systems (with unique solution) can be transformed into triangular form via forward elimination
- Overall cubic $O(n^3)$ complexity

Important further questions

- What if we can’t find a pivot index?
- What if we have more unknowns than equations?
- What if we have more equations than unknowns?
- What if there is more than one solution?
- What if there is no solution?

Recommended reading

- Levitin, 6.2, Gaussian Elimination, p. 208
- Gilbert Strang, Textbook: Linear Algebra and its Applications

Coming Up

- Combinatorial Optimisation
- Greedy, Brute-force, and Backtracking