EMTH211-19S2 Engineering Linear Algebra and Statistics

Mark Hickman
Room 613 Erskine Building
Department of Mathematics & Statistics
University of Canterbury
mark.hickman@canterbury.ac.nz

Linear Systems

Problem

Solve a **linear** system of equations

$$A\,\mathbf{x} = \mathbf{b}.$$

Gaussian elimination – a refresher

Given a system of linear equations $A\mathbf{x} = b$, Gaussian elimination solves this system in **two** steps.

1. **Row Reduction:** Row operations are used to reduce the augmented matrix $[A \mid \mathbf{b}]$ to an **upper triangular** form; that is,

$$[A \mid \mathbf{b}] \longrightarrow \left[\begin{array}{cccccc|c} * & * & * & \cdots & * & * & * \\ 0 & * & * & \cdots & * & * & * \\ 0 & 0 & * & \cdots & * & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & * & * \end{array}\right].$$

2. **Back Substitution** is then used on the **row reduced** matrix to solve the system of equations.

Gaussian elimination – Row Reduction

- **Elementary row operations** are operations that **leave the solution unchanged**.
- Since a row of the augmented matrix corresponds to an equation in the system, there are three types of elementary row operations.
    - Interchange two rows

    $$R_i \leftrightarrow R_j.$$

    - Add a multiple α of one row to another row

    $$R_i \to R_i + \alpha R_j.$$

    - Multiply a row by a non-zero constant β

    $$R_i \to \beta R_i.$$

- It is the first two operations that we use in row reduction.

Gaussian elimination – Row Reduction

$$[A \mid \mathbf{b}] \longrightarrow \left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{32} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{array}\right]$$

- We wish to eliminate a_{21}. We can do this by

$$R_2 \to R_2 - \frac{a_{21}}{a_{11}} R_1 .$$

  This will update the entire second row.
- We now wish to eliminate a_{31}. Thus

$$R_3 \to R_3 - \frac{a_{31}}{a_{11}} R_1$$

- and so on down the first column.

Gaussian elimination – Row Reduction

$$[A \mid \mathbf{b}] \longrightarrow \left[\begin{array}{ccccc|c} \boxed{a_{11}} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & \tilde{a}_{22} & \tilde{a}_{23} & \cdots & \tilde{a}_{2n} & \tilde{b}_2 \\ 0 & \tilde{a}_{32} & \tilde{a}_{33} & \cdots & \tilde{a}_{3n} & \tilde{b}_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \end{array}\right]$$

- We wish to eliminate $\tilde{a}_{32}$. We do this by

$$R_3 \to R_3 - \frac{\tilde{a}_{32}}{\tilde{a}_{22}} R_2$$

- and so on down the second column.
- Repeating this procedure on the remaining columns of A, we eventually obtained the row reduced form of the augmented matrix.

Gaussian elimination – Row Reduction

- At each step an **entire** row of the augmented matrix is updated.
- When working on the j^{th} column, we add multiples of the j^{th} row to eliminate the entry.
- The elements on the diagonal are called **pivots**. If any pivot is zero we must use a row interchange to try to obtain a non-zero pivot.
- If it is not possible to get a non-zero pivot then the system **has either no solution or infinitely many solutions**.
- At the end of row reduction, the matrix has the form

$$\left[\begin{array}{cccccc|c} \boxed{*} & * & * & \cdots & * & * & * \\ 0 & \boxed{*} & * & \cdots & * & * & * \\ 0 & 0 & \boxed{*} & \cdots & * & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \boxed{*} & * & * \\ 0 & 0 & 0 & \cdots & 0 & \boxed{*} & * \end{array}\right] \qquad \boxed{*} \quad \text{denotes the pivots.}$$

Gaussian elimination – Back Substitution

- If **all** the pivots are non-zero, the system has an **unique** solution.
- After row reduction, the linear system has the form

$$\begin{bmatrix} \boxed{r_{11}} & * & * & \cdots & * & * \\ 0 & \boxed{r_{22}} & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \boxed{r_{n-1,n-1}} & * \\ 0 & 0 & 0 & \cdots & 0 & \boxed{r_{nn}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

- The last row gives

$$x_n = \frac{\beta_n}{r_{nn}}.$$

- The second last row gives

$$x_{n-1} = \frac{\beta_{n-1} + r_{n-1,n}\, x_n}{r_{n-1,n-1}} = \frac{r_{nn}\,\beta_{n-1} + r_{n-1,n}\,\beta_n}{r_{n-1,n-1}\, r_{nn}}$$

- and so on.

Gaussian elimination – What can go wrong?

- Consider the system

$$0.01x_1 + x_2 = 1$$
$$x_1 - x_2 = 0.$$

- Note that each coefficient and the right hand side is a two-digit number (for example, 0.01 is represented as 0.10×10^{-1})
- Exact Arithmetic:
  - Row Reduction:

$$\left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right] \to \left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 0 & -101 & -100 \end{array}\right]$$

  - Back Substitution:

$$\mathbf{x} = \begin{bmatrix} \frac{100}{101} \\ \frac{100}{101} \end{bmatrix} = \begin{bmatrix} 0.990099\ldots \\ 0.990099\ldots \end{bmatrix}.$$

- The best we could expect to obtain from a two-digit computer would be the closest two-digit numbers to the exact solution; namely

$$\mathbf{x} \approx \begin{bmatrix} 0.99 \\ 0.99 \end{bmatrix}.$$

- So what happens on our hypothetical two-digit computer?

Gaussian elimination – What can go wrong?

- Finite Precision Arithmetic: (on a two-digit computer)
  - Row Reduction:

$$\left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right] \xrightarrow{\text{two-digit}} \left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 0 & -100 & -100 \end{array}\right]$$

↑ closest Digit to 101

  since, to two digits,
  $-1 - 100 = -(0.10 \times 10^1) - (0.10 \times 10^3) = -0.10 \times 10^3 = -100.$
  - Back Substitution:

$$\mathbf{x} \approx \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

- While we still have a good approximation for x_2, $x_1 = 0$ which is a poor (to say the least) approximation for 0.99.
- However if we first do a row swap then the computation becomes
  - Row Reduction:

$$\left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0.01 & 1 & 1 \end{array}\right] \xrightarrow{\text{two-digit}} \left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0 & 1 & 1 \end{array}\right]$$

Row swap may improve accuracy of the matrix

  - Back Substitution:

$$\mathbf{x} \approx \begin{bmatrix} 1 \\ 1 \end{bmatrix}!$$

Gaussian elimination – The Moral

- This example demonstrates that, with finite precision arithmetic, **different pivots** may lead to **dramatically different answers** through Gaussian elimination. Some of these answers may be acceptable but some can be totally unacceptable.
- We need to develop a **pivot selection strategy** to avoid (if possible) bad answers!
- But first we need to understand what is happening in this example.

Gaussian elimination – Backward Error Analysis

- What system of equations do we need to solve with *exact* arithmetic in order to obtain the *same* row reduced matrix as we computed with finite precision arthmetic?
- Note that the first row of a matrix is unchanged by Gaussian elimination. In the second (that is, good) case, we have

$$\left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0.01 & 1 & 1 \end{array}\right] \xrightarrow{\text{two-digit}} \left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0 & 1 & 1 \end{array}\right] \xleftarrow[\text{exact}]{} \left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0.01 & 0.99 & 1 \end{array}\right]$$

- We see that the two systems are close:

$$\begin{aligned} x_1 - x_2 &= 0 \\ 0.01x_1 + x_2 &= 1 \end{aligned} \qquad\qquad \begin{aligned} x_1 - x_2 &= 0 \\ 0.01x_1 + 0.99x_2 &= 1 \end{aligned}$$

two-digit   exact

- Just to reiterate, the second system is the system that we would need to solve using exact arithmetic to obtain the same solution as solving the first system by two-digit arithmetic.
- Therefore the effect of using a two-digit computer to perform Gaussian elimination is to solve a slightly different (that is perturbed) system.

Gaussian elimination – Backward Error Analysis

- What about the first case which gave us such a bad answer?

$$\left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 1 & -1 & 0 \end{array}\right] \xrightarrow{\text{two-digit}} \left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 0 & -100 & -100 \end{array}\right] \xleftarrow[\text{exact}]{} \left[\begin{array}{cc|c} 0.01 & 1 & 1 \\ 1 & 0 & 0 \end{array}\right]$$

- In this case the two systems are not close:

$$\begin{aligned} 0.01x_1 + x_2 &= 1 \\ x_1 - x_2 &= 0 \end{aligned} \qquad\qquad \begin{aligned} 0.01x_1 + x_2 &= 1 \\ x_1 &= 0 \end{aligned}$$

two-digit   exact

- In both cases, the *computed* solution may be viewed as the *exact* solution to a *perturbed* system.
- This approach is called **backward error analysis**. The error in the computation is pushed back from the computation and placed on the data.
- It is an useful approach since frequently data is already inaccurate through experimental or modelling errors.

Gaussian elimination – Backward Error Analysis

- Ideally we would like that errors created by the computer solution would be no larger than the errors inherent in the data.
- Thus we would like the computer solution to be an exact solution to a **slightly perturbed** problem.
- However this example demonstrates that Gaussian elimination, *unmodified*, may result in the solution to a **greatly perturbed** problem.
- In both row reduction and backwards substitution we are dividing by pivots. Therefore if a pivot is small, any errors will be magnified.
- In the bad case, the pivot was 0.01. Therefore, each time we divide by the pivot, errors could be magnified by a factor of $1/0.01 = 100$ (and this does not allow for errors in the pivot itself).
- In the good case, the pivot was 1. Now errors are not magnified.
- This suggests the strategy **avoid small pivots**.

Gaussian elimination – Small Pivots

- But (and there is always a but!) what about the rescaled system

$$\begin{aligned} 10x_1 + 1000x_2 &= 1000 \\ x_1 - x_2 &= 0? \end{aligned}$$

- In this case we obtain

$$\left[\begin{array}{cc|c} 10 & 1000 & 1000 \\ 1 & -1 & 0 \end{array}\right] \xrightarrow{\text{two-digit}} \left[\begin{array}{cc|c} 10 & 1000 & 1000 \\ 0 & -100 & -100 \end{array}\right]$$

  the same bad result.
- Here the pivot is 10 so the bad result is not a consequence of a small pivot.
- If we do a row swap and use the *smaller* pivot, we obtain

$$\left[\begin{array}{cc|c} 1 & -1 & 0 \\ 10 & 1000 & 1000 \end{array}\right] \xrightarrow{\text{two-digit}} \left[\begin{array}{cc|c} 1 & -1 & 0 \\ 0 & 1000 & 1000 \end{array}\right]$$

  and thus the good result.

Gaussian elimination – Scaling

- However, in a *properly scaled* system, we should avoid small pivots.
- Ideally, the system should be scaled *before* it is given to the computer to solve.
  - We can scale an *equation* by multiplying by a non-zero constant. This is equivalent to an elementary row operation.
  - We can also scale the *variables* of a problem

$$x_i \to \alpha_i\, x_i$$

    where α_i are non-zero constants. This does not correspond to an elementary row operation (in fact, it is an elementary *column* operation).
- There is *no* strategy that will work in all situations. However one should use scaling, if possible, to avoid "large" and very small non-zero numbers in the system. The errors in the example above arose from adding a large number to a small number and the consequent loss of precision when using finite precision arithmetic.

Gaussian elimination – Scaling

- For example, consider

$$\begin{aligned} x_1 - 0.0001x_2 &= 0 \\ 100x_1 + x_2 &= 100. \end{aligned}$$

- One might
  - scale the second equation

$$\begin{aligned} x_1 - 0.0001x_2 &= 0 \\ x_1 + 0.01x_2 &= 1 \end{aligned}$$

  - and scale $x_2 \to 0.01x_2 = \tilde{x}_2$ say

$$\begin{aligned} x_1 - 0.01\tilde{x}_2 &= 0 \\ x_1 + \tilde{x}_2 &= 1. \end{aligned}$$

- In the original system, the coefficients varied by six orders of magnitude; in the scaled system, they varied by only two orders of magnitude.

Gaussian elimination – Partial Pivoting

- Again there is *no* pivoting strategy that will work in all situations.
- From a practical point of view, **partial pivoting** is the strategy most commonly used (Matlab uses it by default). It balances low computational overhead with accuracy of the final answer.
- With partial pivoting, **we swap rows so that the pivot always has the largest absolute value of all remaining entries in that column.**
- For example, in the third column, we search the boxed area for the element with largest absolute value. We then use a row operation to move this element to the pivot position.

$$\left[\begin{array}{cccccc|c} * & * & * & * & \cdots & * & * \\ 0 & * & * & * & \cdots & * & * \\ 0 & 0 & \boxed{*} & * & \cdots & * & * \\ 0 & 0 & * & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & * & * & \cdots & * & * \end{array}\right]$$

Gaussian elimination – Complete Pivoting

- An alternative is **complete pivoting**.
- In this case we search the submatrix

$$\left[\begin{array}{cccccc|c} * & * & * & * & \cdots & * & * \\ 0 & * & * & * & \cdots & * & * \\ 0 & 0 & * & * & \cdots & * & * \\ 0 & 0 & * & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & * & * & \cdots & * & * \end{array}\right]$$

  for the largest (in absolute value) entry. We then use row *and* column swaps to bring this element to the pivot position.
- Whilst this strategy is better than partial pivoting, it's use is rarely justified since the computational overhead is *significantly* higher.

Gaussian elimination – Example of Partial Pivoting

Consider the system whose augmented matrix is

$$\left[\begin{array}{cccc|c} 1 & 0 & 3 & -1 & 1 \\ 2 & 1 & 0 & 0 & 1 \\ -4 & 0 & -1 & 0 & 1 \\ -1 & 2 & 2 & 0 & -1 \end{array}\right].$$

- Column 1:
  - We want -4 moved to the pivot position so

$$R_1 \leftrightarrow R_3.$$

  - Gaussian elimination row operations

$$\begin{aligned} R_2 &\to R_2 + \tfrac{1}{2} R_1 \\ R_3 &\to R_3 + \tfrac{1}{4} R_1 \\ R_4 &\to R_4 - \tfrac{1}{4} R_1. \end{aligned}$$

Gaussian elimination – Example of Partial Pivoting

$$\left[\begin{array}{cccc|c} -4 & 0 & -1 & 0 & 1 \\ 0 & 2 & 2.25 & 0 & 0.75 \\ 0 & 0 & 2.75 & -1 & 1.25 \\ 0 & 1 & -0.5 & 0 & 1.5 \end{array}\right]$$

- Column 2:
  - We want 2 moved to the pivot position so

$$R_2 \leftrightarrow R_4.$$

  - Gaussian elimination row operation

$$R_4 \to R_4 - \tfrac{1}{2} R_2$$

    which completes the second column.

Gaussian elimination – Example of Partial Pivoting

$$\left[\begin{array}{cccc|c} -4 & 0 & -1 & 0 & 1 \\ 0 & 2 & 2.25 & 0 & 0.75 \\ 0 & 0 & 2.75 & -1 & 1.25 \\ 0 & 0 & -1.625 & 0 & 1.125 \end{array}\right]$$

- Column 3:
  - No row swaps required.
  - Gaussian elimination row operation

$$R_4 \to R_4 + \tfrac{1.625}{2.75} R_3$$

    which gives the row reduced form.

Gaussian elimination – Another Example

Solve the system whose augmented matrix is

$$\left[\begin{array}{cc|c} 1 & 2 & \frac{1}{3} \\ 2 & 4.01 & \frac{2}{3} \end{array}\right]$$

using a three-digit computer.

- Exact Solution is

$$\mathbf{x} = \begin{bmatrix} \frac{1}{3} \\ 0 \end{bmatrix}.$$

- With three-digit arithmetic, the augmented matrix becomes

$$\left[\begin{array}{cc|c} 1 & 2 & 0.333 \\ 2 & 4.01 & 0.667 \end{array}\right]$$

- To obtain a three-digit representation of this system, we are forced to introduce inaccuracies (albeit small) in the vector **b**.
- So what happens?

Gaussian elimination – Another Example

Gaussian elimination with partial pivoting gives

$$\left[\begin{array}{cc|c} 1 & 2 & 0.333 \\ 2 & 4.01 & 0.667 \end{array}\right] \to \left[\begin{array}{cc|c} 2 & 4.01 & 0.667 \\ 1 & 2 & 0.333 \end{array}\right] \xrightarrow{\text{three-digit}} \left[\begin{array}{cc|c} 2 & 4.01 & 0.667 \\ 0 & \color{red}{-0.005} & -0.0005 \end{array}\right]$$

- Note that the outcome of the row operation $R_2 \to R_2 - \frac{1}{2} R_1$ depends on the order in which the arithmetic operations are performed.
  - If we compute $\frac{1}{2}(2R_2 - R_1)$ we obtain

$$\frac{4 - 4.01}{2} = \frac{-0.01}{2} = -0.005$$

  - whereas if we compute $R_2 - \frac{1}{2} R_1$ we obtain

$$2 - \frac{4.01}{2} = 2 - 2.01 = -0.010.$$

- Finally the computed solution is

$$\mathbf{x} \approx \begin{bmatrix} 0.133 \\ 0.100 \end{bmatrix}.$$

GAUSSIAN ELIMINATION – ANOTHER EXAMPLE

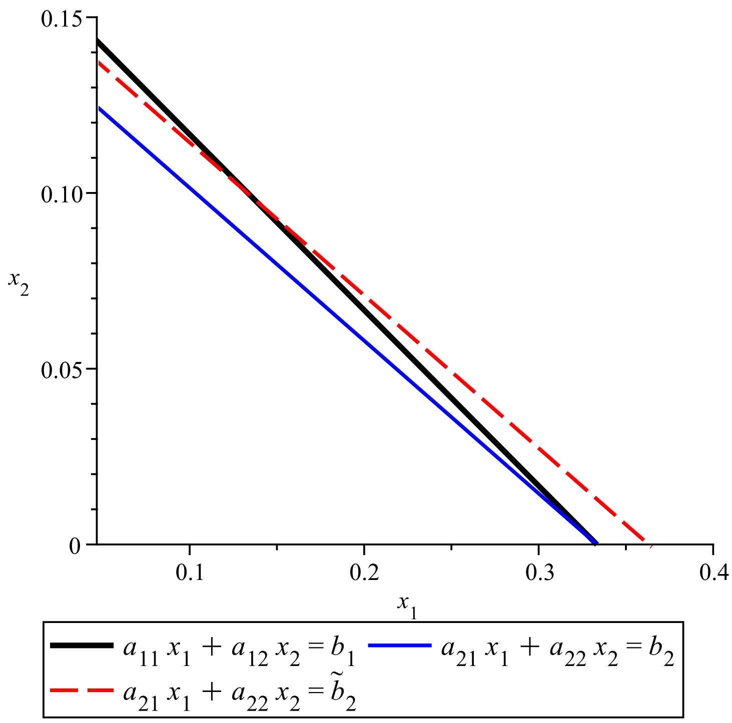
- Partial pivoting does not help in this case. The inaccuracies in the data are less than 0.1%. However

$$\begin{bmatrix} 0.133 \\ 0.100 \end{bmatrix} \qquad \qquad \begin{bmatrix} 0.333 \\ 0.000 \end{bmatrix}$$

computed solution — exact solution (to 3 significant figures)

- This is an example of an **ill-conditioned** system.
- Geometrically, this system represents two lines that are *almost parallel*.
- We will revisit ill-conditioned systems later in this course.

GAUSSIAN ELIMINATION – ILL-CONDITIONED SYSTEMS



How fast is fast?

- We have discussed accuracy issues for Gaussian elimination but what about speed? Is it a *fast* algorithm or do we need another algorithm?
- First we need a measure on how fast an algorithm is.
- Given that an algorithm is used on a computer (rather than by us), we measure the speed of a algorithm in terms of the number of *slow* operations that a computer has to perform.
- The slowest operations are the *floating point* operations; multiplication/division and addition/subtraction.
- Floating point operations are measured in terms of **flops**.
- Flops per second is *one* measure that is used to benchmark computer performance.
    - The fastest supercomputers today have performance in the 1 exaflop/sec (10^{18} flop/sec) range.
    - The fastest Intel processors have performance around 1 teraflop (10^{12} flop)/sec.
    - Graphic processing units (GPUs) deal mainly with floating points (flops) used on dedicated graphics cards are, for double precision floating point, in the 10 teraflop range.
- So what is the flop count for Gaussian elimination and should it concern us?

Gaussian elimination – flop count

- Two well-known (!) formulae that will help us in the flop count are

$$\sum_{i=1}^{n} i = \frac{1}{2}\left(n^2 + n\right)$$

$$\sum_{i=1}^{n} i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n$$

Gaussian elimination – flop count

- Row Reduction:
  - First Column: We use row operations of the form

$$R_i \to R_i - a_{i1} \times \frac{R_1}{a_{11}}.$$

  - We need to divide (that is multiply!) each element in R_1 by a_{11}. However we do not need to do this for the first element in this row and so there are $n\times$ to perform (remember though there are only $n-1$ remaining elements in the coefficient matrix, we also have the element b_1 in the augmented matrix). We update the first row so that the pivot element is now 1.
  - Each row we need to perform one addition and one multiplication. Again we do not need to do this for the first element since we *know* that this element is 0 by construction. Thus there are $n\times$ and $n+$ to perform.
  - Since there are $n-1$ rows we need to update we have a total count

$$[(n-1)n+n] = n^2 \times \text{ and } (n-1)n+$$

operations to perform. Thus $2n^2 - n$ flops are required.

Gaussian elimination – flop count

- Row Reduction:
  - Remaining Columns: The count is the same for remaining columns except that, for the j^{th} column, we are performing the operations on a $(n-j)\times(n-j+1)$ submatrix. Thus the flop count is

$$2(n-j+1)^2 - (n-j+1)$$

for $j = 2, \ldots, n$.
- The total flop count for row reduction is

$$\begin{aligned}\sum_{j=1}^{n}\left[2(n-j+1)^2-(n-j+1)\right] &= \sum_{i=1}^{n}\left[2i^2-i\right] \qquad (i=n-j+1)\\ &= \tfrac{2}{3}n^3+\tfrac{1}{2}n^2-\tfrac{1}{6}n\\ &\approx \tfrac{2}{3}n^3\end{aligned}$$

for large n. We are only interested in large n, so we write

$$\text{Row Reduction} = \mathcal{O}(\tfrac{2}{3}n^3).$$

$\mathcal{O}$ = order

Gaussian elimination – flop count

- After the above row reduction, the linear system has the form

$$\begin{bmatrix} 1 & * & * & \cdots & * & * \\ 0 & 1 & * & \cdots & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & * \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_{n-1} \\ \beta_n \end{bmatrix}$$

- The last row gives

$$x_n = \beta_n.$$

- The second last row gives

$$x_{n-1} = \beta_{n-1} - r_{n-1,n}\, x_n$$

- and so on.

Gaussian elimination – flop count

- Back Substitution: In the $(n-j+1)^{\text{th}}$ row (that is, the j^{th} row from the bottom) we need $(j-1)\times$ and $(j-1)+$ (check; note after the above row reduction, the pivot elements are all 1). Therefore the flop count is

$$\sum_{j=1}^{n} [2(j-1)] = n^2 - n$$

We write

$$\text{Back Substitution} = \mathcal{O}(n^2).$$

- The total flop count for Gaussian elimination *without pivoting* is

$$\text{Gaussian Elimination} = \mathcal{O}(\tfrac{2}{3}\, n^3)$$

since $n^2 << n^3$ for large n.

Gaussian elimination – flop count

- What about pivoting? It takes one addition to compare two numbers (computers compute the difference and then check the sign).
- Partial Pivoting: In the j^{th} column we need to compare $n-j+1$ elements to obtain the pivot. This involves $n-j$ comparisons. Therefore the flop count is

$$\sum_{j=1}^{n}\left[n-j\right]=\sum_{i=0}^{n-1} i=\tfrac{1}{2}n^2-\tfrac{1}{2}n$$

  or

$$\text{Partial Pivoting}=\mathcal{O}(\tfrac{1}{2}n^2)$$

Gaussian elimination – flop count

- Complete Pivoting: In the j^{th} column we need to compare $(n-j+1)^2$ elements to obtain the pivot. This involves $(n-j+1)^2-1$ comparisons. Therefore the flop count is

$$\sum_{j=1}^{n}\left[(n-j+1)^2-1\right]=\sum_{i=1}^{n}\left[i^2-1\right]=\tfrac{1}{3}n^3+\tfrac{1}{2}n^2-\tfrac{5}{6}n$$

  or

$$\text{Complete Pivoting}=\mathcal{O}(\tfrac{1}{3}n^3)$$

- Whereas partial pivoting does not increase the flop count significantly, complete pivoting does.

$$\text{Gaussian Elimination with partial pivoting}=\mathcal{O}(\tfrac{2}{3}n^3)$$

$$\text{Gaussian Elimination with complete pivoting}=\mathcal{O}(n^3)$$

Gaussian elimination – flop count

- As a comparison we have (check if you like)

$$\begin{aligned}\text{Gauss-Jordan Elimination} &= \mathcal{O}(n^3)\\ \text{Computing } A^{-1} &= \mathcal{O}(2n^3)\\ \text{Multiplying two } n \times n \text{ matrices} &= \mathcal{O}(2n^3)\end{aligned}$$

- A *generic* Gaussian elimination routine would take $\mathcal{O}(\frac{8}{3}n^3)$ to compute the inverse. The algorithm has to use the fact that there are many zeros in the identity matrix to obtain the above speed. Computing the inverse is the **least efficient** method of solving a system of linear equations.
- The fact that row reduction grows as *cube* of the number of variables is a very big issue. If we double the size of the problem, then the problem will take at least 8 times longer to solve.

Is speed an issue?

- These days most computer monitors have at least 10^6 pixels. Thus, in updating such a monitor, one could get a system of equations that have 10^6 variables! If we used Gaussian elimination, how long would it take?
- The flop count would be

$$0.67 \times 10^{18}.$$

- That is, on a

supercomputer	0.67 sec
state of the art graphics processor	67,000 sec (18 hours)
state of the art PC	670,000 sec (8 days)

- **Therefore Gauss elimination is NOT a fast algorithm.**
- More precisely, **generic** Gauss elimination is not fast. As seen in the first lab, it can be fast for certain problems.