Lecture 10

Gaussian Elimination with Pivoting

Owen L. Lewis

Department of Mathematics and Statistics
University of New Mexico

Sept. 19, 2024

Goals for today. . .

- Identify *why* our basic GE method is “naive”: identify where the errors come from?
  - division by zero, near-zero
- Propose strategies to eliminate the errors
  - partial pivoting, complete pivoting, scaled partial pivoting
- Investigate the cost: does pivoting cost too much?
- Try to answer “How *accurately* can we solve a system with or without pivoting?”
  - Analysis tools: norms, condition number, . . .

Gaussian Elimination Algorithm: Storing Multipliers

Listing 1: Forward Elimination

```
given A, b

for k = 1...n-1
   for i = k+1...n
      xmult = a_ik / a_kk
      a_ik = xmult
      for j = k+1...n
         a_ij = a_ij - (xmult) a_kj
      end
      b_i = b_i - (xmult) b_k
   end
end
```

We are storing the multipliers in the below diagonal entries (just being efficient).

Those entries will never be accessed during back-substitution!

Naive Gaussian Elimination Algorithm

- Forward Elimination
- + Backward substitution
- = Naive Gaussian Elimination

Example

GE_naive.m   GE_naive_test.m

Forward Elimination Cost?

What is the cost in converting from A to U?

Step k	Add	Multiply	Divide
1	$(n-1)^2$	$(n-1)^2$	$n-1$
2	$(n-2)^2$	$(n-2)^2$	$n-2$
$\vdots$			
n-1	1	1	1

or

add	$\sum_{j=1}^{n-1} j^2$
multiply	$\sum_{j=1}^{n-1} j^2$
divide	$\sum_{j=1}^{n-1} j$

Forward Elimination Cost?

add	$\sum_{j=1}^{n-1} j^2$
multiply	$\sum_{j=1}^{n-1} j^2$
divide	$\sum_{j=1}^{n-1} j$

We know $\sum_{j=1}^{p} j = \frac{p(p+1)}{2}$ and $\sum_{j=1}^{p} j^2 = \frac{p(p+1)(2p+1)}{6}$, so

add-subtracts	$\frac{n(n-1)(2n-1)}{6}$
multiply-divides	$\frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2} = \frac{n(n^2-1)}{3}$

Forward Elimination Cost?

add-subtracts	$\frac{n(n-1)(2n-1)}{6}$
multiply-divides	$\frac{n(n^2-1)}{3}$
add-subtract for b	$\frac{n(n-1)}{2}$
multiply-divides for b	$\frac{n(n-1)}{2}$

Back Substitution Cost

As before

add-subtract	$\frac{n(n-1)}{2}$
multiply-divides	$\frac{n(n+1)}{2}$

Naive Gaussian Elimination Cost

Combining the cost of forward elimination, updating b, and backward substitution gives

add-subtracts	$\frac{n(n-1)(2n-1)}{6} + \frac{n(n-1)}{2} + \frac{n(n-1)}{2} = \frac{n(n-1)(2n+5)}{3}$
multiply-divides	$\frac{n(n^2-1)}{3} + \frac{n(n-1)}{2} + \frac{n(n+1)}{2} = \frac{n(n^2+3n-1)}{3}$

So the total cost of add-subtract-multiply-divide is about

$$\frac{2}{3}n^3$$

$\Rightarrow$ double n results in a cost increase of a factor of 8

Why is this “naive”?

Example

$$A = \begin{bmatrix} 0 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Example

$$A = \begin{bmatrix} 1e-10 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Why is our basic GE “naive”?

Example

$$A = \begin{bmatrix} 0 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Divide by zero $\Longrightarrow$ Bad!

Example

$$A = \begin{bmatrix} 1e-10 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Adding numbers of vastly different sizes $\Longrightarrow$ Bad!

The Need for Pivoting

Solve:

$$A = \begin{bmatrix} 2 & 4 & -2 & -2 \\ 1 & 2 & 4 & -3 \\ -3 & -3 & 8 & -2 \\ -1 & 1 & 6 & -3 \end{bmatrix} \qquad b = \begin{bmatrix} -4 \\ 5 \\ 7 \\ 7 \end{bmatrix}$$

Note that there is nothing ”wrong” with this system. A is full rank. The solution exists and is unique.
Form the augmented system.

$$\left[\begin{array}{rrrr|r} 2 & 4 & -2 & -2 & -4 \\ 1 & 2 & 4 & -3 & 5 \\ -3 & -3 & 8 & -2 & 7 \\ -1 & 1 & 6 & -3 & 7 \end{array}\right]$$

The Need for Pivoting

Subtract $1/2$ times the first row from the second row,
add $3/2$ times the first row to the third row,
add $1/2$ times the first row to the fourth row.
The result of these operations is:

$$\left[\begin{array}{rrrr|r} 2 & 4 & -2 & -2 & -4 \\ 0 & 0 & 5 & -2 & 7 \\ 0 & 3 & 5 & -5 & 1 \\ 0 & 3 & 5 & -4 & 5 \end{array}\right]$$

The *next* stage of Gaussian elimination will not work because there is a zero in the *pivot* location, a_{22}.

The Need for Pivoting

Swap second and fourth rows of the augmented matrix.

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 3 & 5 & -4 & 5 \\ 0 & 3 & 5 & -5 & 1 \\ 0 & 0 & 5 & -2 & 7 \end{array}\right]$$

Continue with elimination: subtract (1 times) row 2 from row 3.

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 3 & 5 & -4 & 5 \\ 0 & 0 & 0 & -1 & -4 \\ 0 & 0 & 5 & -2 & 7 \end{array}\right]$$

The Need for Pivoting

Another zero has appear in the pivot position. Swap row 3 and row 4.

$$\left[\begin{array}{cccc|c} 2 & 4 & -2 & -2 & -4 \\ 0 & 3 & 5 & -4 & 5 \\ 0 & 0 & 5 & -2 & 7 \\ 0 & 0 & 0 & -1 & -4 \end{array}\right]$$

The augmented system is now ready for backward substitution.

Notes

Another example

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

As $\varepsilon \to 0$, $x_1, x_2 \to 1$.

Example

With Naive GE,

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & (1 - \frac{1}{\varepsilon}) \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 - \frac{1}{\varepsilon} \end{bmatrix}$$

Solving for x_1 and x_2 we get

$$x_2 = \frac{2 - 1/\varepsilon}{1 - 1/\varepsilon}$$

$$x_1 = \frac{1 - x_2}{\varepsilon}$$

For $\varepsilon \approx 10^{-20}$, $x_1 \approx 0$, $x_2 \approx 1$

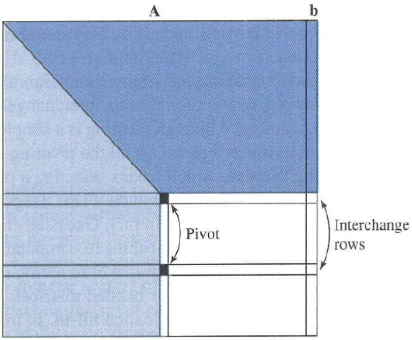
Pivoting Strategies

Partial Pivoting: Exchange only rows

- Exchanging rows does not affect the order of the x_i
- For increased numerical stability, make sure the largest possible pivot element is used. This requires searching in the partial column below the pivot element.
- Partial pivoting is usually sufficient.

Partial Pivoting

To avoid division by zero (small number), swap the row having the zero (small number) pivot with one of the rows below it.



To minimize the effect of roundoff, always choose the row that puts the largest pivot element on the diagonal, i.e., find i_p such that $|a_{i_p,i}| = \max(|a_{k,i}|)$ for $k = i, \ldots, n$

Partial Pivoting

Partial pivoting (swapping rows in a matrix) is equivalent to re-ordering equation:

$$
\begin{aligned}
&\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\
\Longrightarrow &\begin{cases} \varepsilon x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \\
\Longrightarrow &\begin{cases} x_1 + x_2 = 2 \\ \varepsilon x_1 + x_2 = 1 \end{cases} \\
\Longrightarrow &\begin{bmatrix} 1 & 1 \\ \varepsilon & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}
\end{aligned}
$$

The variable labels don't change. We haven't re-ordered our x's.

Notes

The Algorithm

Listing 2: (forward) GE with PP

```
for k = 1 to n-1
    rmax = 0
    for m = k to n
      r = |a_mk|
      if(r > rmax)
        rmax = r
        j = m
    end
    swap rows A(j,:) and A(k,:)
    swap elements b(j) and b(k)
    for i = k+1 to n
      xmult = a_ik / a_kk
      a_ik = xmult
      for j = k+1 to n
        a_ij = a_ij - xmult * a_kj
      end
      b_i = b_i - xmult * b_k
    end
  end
```

Partial Pivoting: Usually sufficient, but not always

- Partial pivoting is <u>usually</u> sufficient
- Consider

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

With Partial Pivoting, the first row is the pivot row:

$$\left[\begin{array}{cc|c} 2 & 2c & 2c \\ 0 & 1-c & 2-c \end{array}\right]$$

and for large c on a machine, $1-c \to -c$ and $2-c \to -c$:

$$\left[\begin{array}{cc|c} 2 & 2c & 2c \\ 0 & -c & -c \end{array}\right]$$

so that $x_1 = 0$ and $x_2 = 1$. For large c, exact is $x_2 \approx x_1 \approx 1$.

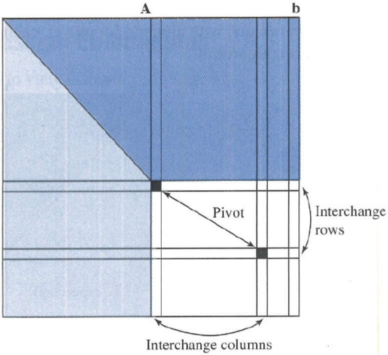
- The pivot is selected as the largest in the column, but it is small compared to its row, we still have issues.

More Pivoting Strategies

Full (or Complete) Pivoting: Exchange *both* rows and columns

- Column exchange requires changing the order of the x_i
- For increased numerical stability, make sure the largest possible pivot element is used. This requires searching in the pivot row, *and* in all rows below the pivot row, starting in the pivot column.
  That is, you search through a square submatrix of A for the largest element.
- Full pivoting is less susceptible to roundoff, but the increase in stability comes at a cost of more complex programming (not a problem if you use a library routine) and an increase in work associated with searching and data movement.

Full Pivoting



Partial Pivoting: but smarter

- Consider

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

- The pivot is selected as the largest in the column, but it should be the largest <u>relative</u> to its own row (of the things that haven't been pivots yet).

Scaled Partial Pivoting

We simulate full pivoting by using a scale with partial pivoting.

- pick pivot element as the largest entry in the column, but scale by the largest entry in each row, i.e., consider $\max_i |a_{i,k}/s_i|$ for finding the pivot in column k
- s_i is the largest entry in row i, so that we can "simulate" full pivoting by choosing the "largest" equation as the pivot row.

Notes

The Algorithm

Listing 3: (forward) GE with SPP (toy)

```
initialize s as maximum of each row
for k = 1 to n−1
    rmax = 0
    for m = k to n
      r = |a_mk / s(m)|
      if(r > rmax)
        rmax = r
        j = m
    end
    swap rows A(j,:) and A(k,:)
    swap elements b(j) and b(k)
    for i = k+1 to n
      xmult = a_ik / a_kk
      a_ik = xmult
      for j = k+1 to n
        a_ij = a_ij − xmult · a_kj
      end
      b_i = b_i − xmult · b_k
    end
  end
```

Scaled Partial Pivoting

We simulate full pivoting by using a scale with partial pivoting.

- We now have the bones of an algorithm, but swapping data in the matrix constantly is actually **terrible** for performance.
- do not swap, just keep track of the order of the pivot rows
- call this vector $\ell = [\ell_1, \ldots, \ell_n]$.

SPP Process

1. Determine a scale vector **s**. For each row
$$s_i = \max_{1 \leqslant j \leqslant n} |a_{ij}|$$
2. initialize $\ell = [\ell_1, \ldots, \ell_n] = [1, \ldots, n]$.
3. select row j to be the row with the largest ratio
$$\frac{|a_{\ell_i 1}|}{s_{\ell_i}} \qquad 1 \leqslant i \leqslant n$$
4. swap ℓ_j with ℓ_1 in ℓ
5. Now we need $n - 1$ multipliers for the first column:
$$m_{\ell_i,1} = \frac{a_{\ell_i 1}}{a_{\ell_1 1}}$$
6. So the index to the rows are being swapped, NOT the actual row vectors which would be expensive
7. finally use the multiplier $m_{\ell_i,1}$ times row ℓ_1 to subtract from rows ℓ_i for $2 \leqslant i \leqslant n$

SPP Process continued

1. For the second column in forward elimination, we select row j that yields the largest ratio of

$$\frac{|a_{\ell_i,2}|}{s_{\ell_i}} \qquad 2 \leqslant i \leqslant n$$

2. swap ℓ_j with ℓ_2 in ℓ
3. Now we need $n-2$ multipliers for the second column:

$$m_{\ell_i,2} = \frac{a_{\ell_i,2}}{a_{\ell_2 2}}$$

4. finally use the multiplier m_2 times row ℓ_2 to subtract from rows ℓ_i for $3 \leqslant i \leqslant n$
5. the process continues for row k

Note: scale factors are *not* updated

An Example

Consider... at beginning, $\ell = (1, 2, 3)^T$.

$$\begin{bmatrix} 2 & 4 & -2 \\ 1 & 3 & 4 \\ 5 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -1 \\ 2 \end{bmatrix},$$

$$\mathbf{s} = \begin{bmatrix} 4 \\ 4 \\ 5 \end{bmatrix}, \Rightarrow \left[\frac{|a_{i1}|}{s_i}\right] = \begin{bmatrix} 1/2 \\ 1/4 \\ 1 \end{bmatrix} \Longrightarrow \ell = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}.$$

Subtract $2/5$ of row 3 from row 1, and subtract $1/5$ of row 3 from row 2.

$$\begin{bmatrix} 0 & 3.2 & -2 \\ 0 & 2.6 & 4 \\ 5 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5.2 \\ -1.4 \\ 2 \end{bmatrix},$$

An Example

Now, $\ell = (3, 2, 1)^T$. Search for second pivot.

$$\begin{bmatrix} 0 & 3.2 & -2 \\ 0 & 2.6 & 4 \\ 5 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5.2 \\ -1.4 \\ 2 \end{bmatrix},$$

$$\mathbf{s} = \begin{bmatrix} 4 \\ 4 \\ 5 \end{bmatrix}, \Rightarrow \left[\frac{|a_{i2}|}{s_i} \right] = \begin{bmatrix} 4/5 \\ 13/20 \\ \times \end{bmatrix} \Longrightarrow \ell = \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}.$$

Subtract $13/16$ of row 1 from row 2.

$$\begin{bmatrix} 0 & 3.2 & -2 \\ 0 & 0 & 5.625 \\ 5 & 2 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5.2 \\ -5.625 \\ 2 \end{bmatrix},$$

Back Substitution. . . Sort of

1. Solve for x_n using last index ℓ_n:

$$a_{\ell_n n} x_n = b_{\ell_n} \;\Rightarrow\; x_n = \frac{b_{\ell_n}}{a_{\ell_n n}}$$

2. Solve for x_{n-1} using the second to last index ℓ_{n-1}:

$$x_{n-1} = \frac{1}{a_{\ell_{n-1} n-1}} \left(b_{\ell_{n-1}} - a_{\ell_{n-1} n} x_n \right)$$

The Algorithms

Listing 4: (forward) GE with SPP

```
Initialize ℓ = [1, ..., n]
Set s to be the max of rows
for k = 1 to n
   rmax = 0
   for m = k to n
      r = |a_{ℓ_i k} / s_{ℓ_i}|
      if(r > rmax)
         rmax = r
         j = m
   end
   swap ℓ_j and ℓ_k
   for i = k+1 to n
      xmult = a_{ℓ_i k} / a_{ℓ_k k}
      a_{ℓ_i k} = xmult
      for j = k+1 to n
         a_{ℓ_i j} = a_{ℓ_i j} − xmult · a_{ℓ_k j}
      end
   end
end
```

See *Gauss* algorithm on page 267 of handout

The Algorithms

Note: the multipliers are stored in the location $a_{\ell_i k}$ in the text

Listing 5: (back solve) GE with SPP

```
for k = 1 to n−1
   for i = k+1 to n
      b_{ℓ_i} = b_{ℓ_i} − a_{ℓ_i k} b_{ℓ_k}
   end
end
x_n = b_{ℓ_n} / a_{ℓ_n n}
for i = n−1 down to 1
   sum = b_{ℓ_i}
   for j = i+1 to n
      sum = sum − a_{ℓ_i j} x_j
   end
   x_i = sum / a_{l_i,i}
end
```

See *Solve* algorithm on page 269 of handout