MA 580; Gaussian Elimination

C. T. Kelley
NC State University
tim_kelley@ncsu.edu
Version of September 7, 2016

Master Chapters 1--7 of the Matlab book. NOW!!!
Read chapter 5 of the notes. Start on Chapter 9 of the Matlab book.

NCSU, Fall 2016
Part IVa: Gaussian Elimination
Simple Gaussian Elimination Revisited

For the second time we solve

(I) \[ 2x_1 + x_2 = 1 \]
(II) \[ x_1 + x_2 = 2 \]

This time we will record what we do.
Pivot element and multiplier

We plan to multiply (I) by .5 and subtract from (II).

Words:
- $a_{11}$ is the pivot element
- $l_{21} = .5 = a_{21}/a_{11}$ is the multiplier

I will tell you what $L$ is soon.
Solving the system

As before: the new (II) is

$$(\text{II})' x_2/2 = 1.5$$

Now I can **backsolve** the problem starting with $x_2 = 3$. Record the coefficient of $x_2$ in (II)' as $u_{22} = .5$. Now plug into (I) to get $x_1 = -1$. In matrix form, there’s something profound happening.
Suppose I put the multipliers in a unit lower triangular matrix $L$

$$L = \begin{pmatrix} 1 & 0 \\ l_{21} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ .5 & 1 \end{pmatrix}$$

and the coefficients of the backsolve in $U$

$$U = \begin{pmatrix} 2 & 1 \\ 0 & .5 \end{pmatrix}.$$

I’ve encoded all the work in the solve in $L$ and $U$. In fact

$$LU = \begin{pmatrix} 1 & 0 \\ .5 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 \\ 0 & .5 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = A$$
The Factor-Solve Paradigm

In real life, one separates the factorization $A = LU$ from the solve. First factor $A = LU$.

Then

- Solve $Lz = b$ (forward solve)
- Solve $Ux = z$ (back solve)

Put this together and

$$Ax = LUx = Lz = b.$$ 

So you can do several right hand side vectors with a single factorization.
The system for the forward solve looks like

\[
Lz = \begin{pmatrix}
1 & 0 & 0 & \ldots & 0 \\
l_{21} & 1 & 0 & \ldots & 0 \\
l_{31} & l_{32} & 1 & 0 & \ldots \\
\vdots & \vdots & \vdots & \ddots & 0 \\
l_{N1} & l_{N2} & l_{N3} & \ldots & 1
\end{pmatrix}
\begin{pmatrix}
z_1 \\
z_2 \\
z_3 \\
\vdots \\
z_N
\end{pmatrix} =
\begin{pmatrix}
b_1 \\
b_2 \\
b_3 \\
\vdots \\
b_N
\end{pmatrix}
\]

So \( z_1 = b_1 \), \( z_2 = b_2 - l_{21}z_1 \), ...
Algorithm for Lower Triangular Solve (forward)

Forward means solve for $z_i$ in order
Solve $Lz = b$ with unit triangular $L$

\begin{verbatim}
for $i = 1 : N$ do
    $z_i = b_i$
    for $j = 1 : i - 1$ do
        $z_i = z_i - l_{ij}z_j$
    end for
end for
\end{verbatim}

Bottom line: $z_i = b_i - \sum_{j=1}^{i-1} l_{ij}z_j, \ 1 \leq i \leq N.$

What if $l_{ii} \neq 1$?
Algorithm for Upper Triangular Solve (backward)

Backward means solve for $z_i$ in reverse order ($z_N$ first)

Solve $Ux = z$

for $i = N : 1$ do
  $x_i = z_i$
  for $j = i + 1 : N$ do
    $x_i = x_i - u_{ij}x_j$
  end for
  $x_i = x_i / u_{ii}$
end for

Bottom line: $x_i = \left( z_i - \sum_{j=i+1}^{N} u_{ij}x_j \right) / u_{ii}$, $N \geq i \geq 1$
Evaluating Cost

You can use several equivalent metrics for cost

- floating point operations
- floating point multiplications
- “flops”
  - an add + a multiply + some address computation

In MA 580, with one exception, adds and multiplies are paired and the three metrics give equal relative results.
Scalar products and matrix multiplies

\[ x^T y = \sum_{i=1}^{N} x_i y_i \]

\(N\) multiplies and \(N - 1\) adds.
Cost: \(N + O(1)\) leading order is the same for the adds/multiplies

Matrix-vector product:

\[(Ax)_i = \sum_{i=1}^{N} a_{ij} x_j \text{ for } j = 1 \ldots N\]

\(N^2\) multiplies and \(N^2 - N\) adds (\(N\) scalar products)
Cost: \(N^2 + O(N)\)
A trick

- Write the loops as nested sums and add 1 for each multiply.
- Replace the sums by integrals.
- Evaluate the integrals

You’ll be right to leading order.
Triangular Solve

We’ll do lower, upper is the same.

Algorithm:

\[
\text{for } i=1:N \text{ do } \\
\quad z_i = b_i - \sum_{j=1}^{i-1} l_{ij} \ast z_j \\
\text{end for}
\]

Sum:

\[
\sum_{i=1}^{N} \sum_{j=1}^{i-1} 1
\]

Integral (check it out)

\[
\int_1^N \int_1^{i-1} dj \, di = \frac{N^2}{2} + O(N)
\]
Simple Gaussian Elimination

Here are the steps

- Make a copy of $A$.
- Do Gaussian elimination as if one were solving an equation.
- Recover $L$ and $U$ from our (mangled) copy of $A$.

In the real world, we would overwrite $A$ with the data for the factors, saving storage.

Have I told you about the computer scientist who was low on memory?
The guy tells me he doesn’t have enough bytes.
So I bit him.
H. Youngman
Overwriting $A$ with $L$ and $U$

```latex
lu_simple A ← A

for $j = 1 : N$ do
    for $i = j + 1 : N$ do
        $a_{ij} = a_{ij} / a_{jj} \quad \{ \text{Compute the multipliers. Store them in the strict lower triangle of } A. \}$
        for $k = j + 1 : N$ do
            $a_{ik} = a_{ik} - a_{ij} a_{jk} \quad \{ \text{Do the elimination.} \}$
        end for
    end for
end for
```
Part IVa: Gaussian Elimination

Cost of LU factorization

\[
\sum_{j=1}^{N} \sum_{i=j+1}^{N} \sum_{k=j+1}^{N} 1 \approx \int_{1}^{N} \int_{j+1}^{N} \int_{j+1}^{N} dk \ di \ dj = \frac{N^3}{3} + O(N^2).
\]

So, cost of LU is \( \frac{N^3}{3} + O(N^2) \).
Recovering $L$ and $U$

$$(L, U) \leftarrow \text{recover}(A)$$

for $j = 1 : N$ do
  $l_{jj} = 1$
  for $i = 1 : j$ do
    $u_{ij} = a_{ij}$
  end for
  for $i = j + 1 : N$ do
    $l_{ij} = a_{ij}$
  end for
end for
The factorization costs $O(N^3)$ work
The upper and lower triangular solves cost $O(N^2)$ work
So . . .
  - the standard procedure is to separate them.
  - This makes it much more efficient to handle multiple right hand sides,
    as we will do in the nonlinear equations part.
Mission Accomplished?

Not exactly. Does

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

have an LU factorization?
No, but solving

$$Ax = \begin{pmatrix} x_2 \\ x_1 \end{pmatrix} = b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

is pretty easy. We’ve clearly missed something.
Partial pivoting aka column pivoting

The easy fix it to interchange the equations (rows) in the problem. Then you get \( A' = b' \).

\[
A' = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}
\]

and \( b' = (b_2, b_1)^T \).

Same answer, different problem.

Proposed algorithm: When you get a zero pivot, find something below it that is not zero, and swap those rows. Then make sure you don't forget what you did.
Are we there yet?

How about

$$A = \begin{pmatrix} .1 & 10 \\ 1 & 0 \end{pmatrix}$$

and $$b = (20, -1)^T$$.

Exact solution: $$(-1, 2.01)^T$$
Recall the Toy Floating Point Number System

Here’s a radix 10 system.

- Two digit mantissa
- Exponent range: -2:2
- Largest float = \( .99 \times 10^2 = 99 \)
- \( \text{eps}(0) = .10 \times 10^{-2} \)
- \( \text{eps}(1) = 1.1 - 1 = .1 \)
- Round to nearest.

\[
fl(x^*) = fl((-1, 2.01)^T) = (-1, 2)^T.
\]
How about some elimination?

Multiply first equation by 10 and subtract from second:

\[ \begin{align*}
0.1x + y & = 20 \\
-100y & = fl(-1 - 200) = \text{overflow!!} = \text{INF}.
\end{align*} \]

Better fix: find largest element in absolute value and swap rows.
Swap rows and it’s better.

\[ x = -1 \]
\[ 0.1x + y = 20 \]

Multiply first equation by 0.1 and subtract from second to get

\[ x = -1 \]
\[ 10y = 20 \]

which does the right thing.
What did we do right?

We **normalized** correctly by choosing the pivot element wisely.
Name of method: Gaussian Elimination with Partial Pivoting (GEPP)
aka: Gaussian Elimination with Column Pivoting
This is what the MATLAB backslash command does.

\[ x = A \backslash b; \]
GEPP

\texttt{lu\_gepp } \texttt{A } \leftarrow \texttt{A }

\begin{verbatim}
for \( j = 1 : N \) do 
    Find \( k \) so that \( a_{kj} = \max_{i \geq j} |a_{ij}| \)
    Swap rows \( j \) and \( k \); record the swap
    for \( i = j + 1 : N \) do 
        \( a_{ij} = a_{ij} / a_{jj} \) \{Compute the multipliers. Store them in the strict lower triangle of \( A \).\}
        for \( k = j + 1 : N \) do 
            \( a_{ik} = a_{ik} - a_{ij}a_{jk} \) \{Do the elimination.\}
        end for 
    end for 
end for
\end{verbatim}
Matlab for the elimination

One way (clarity)

```
for k=j+1:N
    a(i,k)=a(i,k)-a(i,j)*a(j,k);
end
```

Other way: **vectorize** (faster)

```
a(i,j+1:N) = a(i,j+1:N) - a(i,j)*a(j,j+1:N);
```

Even better: make \( i \) the inner loop index and vectorize that.

Why? Matlab (like fortran) stores arrays by columns.
Does this really make any difference?

Let's find out:
We will compute the multipliers first

```plaintext
for i = j+1:n
    a(i,j) = a(i,j)/a(j,j);
end
```

and then try four different things and compare to Matlab's `lu`. Matrix is `a = rand(1000,1000)`. 
Non-vectorized vs vectorized. \(i\) loop outside

Version 1

\[
\begin{align*}
\text{for } & i=j+1:n \\
& \text{for } k=j+1:n \\
& \quad a(i,k) = a(i,k) - a(i,j) \times a(j,k); \\
& \text{end} \\
& \text{end}
\end{align*}
\]

Version 2

\[
\begin{align*}
\text{for } & i=j+1:n \\
& a(i,j+1:n) = a(i,j+1:n) - a(i,j) \times a(j,j+1:n); \\
& \text{end}
\end{align*}
\]
Vectorized vs totally vectorized

Version 3

\[
\text{for } k=j+1:n \\
\quad a(j+1:n,k) = a(j+1:n,k) - a(j+1:n,j) \cdot a(j,k); \\
\text{end}
\]

and, now for something completely different

Version 4

\[
a(j+1:n,j+1:n) = a(j+1:n,j+1:n) - a(j+1:n,j) \cdot a(j,j+1:n) \\
; 
\]
Timings in seconds via Matlab `tic` and `toc`.

<table>
<thead>
<tr>
<th>Version 1</th>
<th>Version 2</th>
<th>Version 3</th>
<th>Version 4</th>
<th>Matlab lu</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>10</td>
<td>5</td>
<td>3</td>
<td>1.58e-02</td>
</tr>
</tbody>
</table>
Back to business: with GEPP you’ve stabilized $L$

By construction

- $|l_{ij}| \leq 1$
- So $\|L\|_\infty$, the maximum row sum $\leq N$
  which is more than small enough.

But you have **NOT** stabilized $U$. 
Incremental Cost of GEPP

- Swapping rows is index manipulation and copying. Neglect.
- Record swaps by permuting an integer vector. Neglect.
- A floating point comparison is a subtract followed by looking at the sign bit. Cost = one add.

Total number of adds:

\[ \sum_{j=1}^{N} (N - j) = \int_{1}^{N} (N - j) \, dj + O(N) = \frac{N^2}{2} + O(N). \]

Negligible compared to $N^3/3$ cost (in paired mult/adds) for the elimination.
GEPP as Matrix Factorization

If record the swap means build a matrix $P$ so that
- $P = I$ at the start
- You swap columns of $P$ as you swap rows of $A$.

Then

$$A = PLU$$

$P$ is a permutation matrix.
Permutation Matrices

- Let $P_{ij}$ be the identity with rows $i$ and $j$ swapped. $P_{ij}$ also swaps columns $i$ and $j$.
- The $P_{ij}$s do not commute.
- A permutation matrix is a product of some of the $P_{ij}$s.
- A permutation matrix $P$ is orthogonal. $P^T = P^{-1}$.
- A sequence of column swaps of $I$ is the transpose of a sequence of row swaps.
- $P_{ij}A$ swaps rows; $AP_{ij}$ swaps columns.
Some MATLAB: I

When you call Matlab’s $lu$ code, the permutation is wrapped in $l$.

```matlab
>> A=[1 2 3; 4 5 6; 7 8 9];
>> [l,u]=lu(A);
>> l
```

```
l =

  1.4286e-01   1.0000e+00  0
  5.7143e-01   5.0000e-01  1.0000e+00
  1.0000e+00   0            0
```

Note the use for short e format. Put format short e in your Matlab startup file.
I’ve put an explicit GEPP file on the moodle page `plu.m` for you to play with.

```matlab
>> A=[1 2 3; 4 5 6; 7 8 9];
>> [l, u, p]=plu(A);
>> [p p*l]
ans =
    0     1     0     1.4286e-01     1.0000e+00     0
    0     0     1     5.7143e-01     5.0000e-01     1.0000e+00
    1     0     0     1.0000e+00     0     0
```
Errors in PLU factorization

Recall: solve is $Lz = P^T b$ followed by $Ux = z$. Define

$$E = PLU - A, \quad r_L = Lz - P^T b, \quad r_U = Ux - z$$

Goal: estimate

$$\frac{\|x^* - x\|_\infty}{\|x\|_\infty}$$
Define some (hopefully small) errors

\[ \epsilon_A = \frac{\|E\|_\infty}{\|A\|_\infty}, \quad \epsilon_L = \frac{\|r_L\|_\infty}{\|L\|_\infty \|z\|_\infty}, \quad \epsilon_R = \frac{\|r_U\|_\infty}{\|U\|_\infty \|x\|_\infty}, \]

and the \textbf{growth factor}

\[ g = \frac{\|U\|_\infty}{\|A\|_\infty} \]

which measures element growth in the elimination.
Error Bound

\[
\frac{\|x^* - x\|_\infty}{\|x\|_\infty} \leq \kappa_\infty(A) [\epsilon_A + Ng(\epsilon_U(1 + \epsilon_L) + \epsilon_L)]
\]

The bad news here could be \(g\) and/or \(\kappa\). Errors in triangular solves are benign.
Proof in Chapter 3 of pink book.
Instability of GEPP and the evil matrix

Here’s the matrix that breaks GEPP

\[ A = \begin{pmatrix}
1 & 0 & \ldots & 0 & 0 & 1 \\
-1 & 1 & 0 & \ldots & 0 & 1 \\
-1 & -1 & 1 & 0 & 0 & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & 0 \\
-1 & \ldots & \ldots & \ldots & 1 & 1 \\
-1 & \ldots & \ldots & \ldots & -1 & 1 \\
\end{pmatrix} \]
Coefficients for the evil matrix

\[ a_{ij} = \begin{cases} 
1 & \text{if } i = j \text{ or } j = N \\
-1 & \text{if } i > j \\
0 & \text{otherwise.} 
\end{cases} \]
Example: N=5

\[ A = \begin{pmatrix}
1 & 0 & 0 & 0 & 1 \\
-1 & 1 & 0 & 0 & 1 \\
-1 & -1 & 1 & 0 & 1 \\
-1 & -1 & -1 & 1 & 1 \\
-1 & -1 & -1 & -1 & 1
\end{pmatrix} \]

No pivoting necessary. Just add row 1 to the other rows and ...
First step of GEPP

\[ A \leftarrow \begin{pmatrix}
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 2 \\
0 & -1 & 1 & 0 & 2 \\
0 & -1 & -1 & 1 & 2 \\
0 & -1 & -1 & -1 & 2
\end{pmatrix} \]

No pivoting necessary. Just add row 2 to the following rows and ...
Second step of GEPP

\[
\begin{pmatrix}
1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 2 \\
0 & 0 & 1 & 0 & 4 \\
0 & 0 & -1 & 1 & 4 \\
0 & 0 & -1 & -1 & 4
\end{pmatrix}
\]

No pivoting necessary. Just add row 3 to the following rows and...

Third step of GEPP

This is not looking too good.

$$\mathbf{A} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & -1 & 8 \end{pmatrix}$$

No pivoting necessary. Just add row 4 to the following rows and ...
Fourth step of GEPP

What a stinker!

$$\mathbf{A} \leftarrow \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 4 \\ 0 & 0 & 0 & 1 & 8 \\ 0 & 0 & 0 & 0 & 16 \end{pmatrix}$$

Looks like $\|\mathbf{U}\|_{\infty} = 2^{N-1}$. For this matrix $\|\mathbf{A}\|_{\infty} = N$, so the growth factor $g_{PP}$ for GEPP is kinda big . . .

$$g_{PP} = 2^{N-1}/N.$$  

GEPP is unstable for this problem, as you can (and should) check with Matlab’s $\texttt{lu}$ command.
If GEPP is unstable, why do we use it?

BECAUSE IT WORKS!

- The pathology of the example is never seen in practice
- Any fix, in the opinion of the whole world, costs too much, including the fixes we will propose in this course.
- This is our first example of an algorithm that can be unstable, but rarely or never is in practice.

How can you fail to love this stuff?
function a = evil(n)
% EVIL Create the n x n evil matrix to destabilize GEPP
% a = evil(n)
% a = eye(n,n);
for i = 1:n
    a(i,i) = 1;
    a(i,n) = 1;
    for j = 1:i-1
        a(i,j) = -1;
    end
end
Use Matlab’s `cond` command.

```matlab
>> a=evil(5);
>> kappa=cond(a,inf);
>> [l,u]=lu(a);
>> [kappa, norm(u,inf)]
```

```
ans =

    5   16
```

```matlab
>>
```
Look at $\kappa$ and $g_{pp}$ as $N$ increases.

```matlab
function stats = evil_study
% EVIL_STUDY Look at conditioning and growth factor
% records kappa and growth factor in stats array
%
dimensions=[10 20 50 100 500 1000];
stats=zeros(6,3);
for id=1:6
    stats(1,id)=dimensions(id);
    a=evil(dimensions(id));
    stats(id,2)=cond(a,2); % record kappa_2
    stats(id,3)=cond(a,inf); % record kappa_inf
    [l,u]=lu(a);
    stats(id,4)=norm(u,inf)/dimensions(id);
end
stats
```
Something’s fishy and poorly presented here

```matlab
>> evil_study;

stats =

  1.0000e+01  4.3850e+00  1.0000e+01  5.1200e+01
  2.0000e+01  8.8343e+00  2.0000e+01  2.6214e+04
  5.0000e+01  2.2306e+01  5.0000e+01  1.1259e+13
  1.0000e+02  4.4802e+01  1.0000e+02  6.3383e+27
  5.0000e+02  2.2486e+02  2.2397e+105  3.2734e+147
  1.0000e+03  4.4993e+02  3.3018e+271  5.3575e+297
```

Turn in a table like this and get zero.
### Presentation of results

<table>
<thead>
<tr>
<th>$N$</th>
<th>$\kappa_2$</th>
<th>$\kappa_\infty$</th>
<th>$g_{PP}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4.38e+00</td>
<td>1.00e+01</td>
<td>5.12e+01</td>
</tr>
<tr>
<td>20</td>
<td>8.83e+00</td>
<td>2.00e+01</td>
<td>2.62e+04</td>
</tr>
<tr>
<td>50</td>
<td>2.23e+01</td>
<td>5.00e+01</td>
<td>1.13e+13</td>
</tr>
<tr>
<td>100</td>
<td>4.48e+01</td>
<td>1.00e+02</td>
<td>6.34e+27</td>
</tr>
<tr>
<td>500</td>
<td>2.25e+02</td>
<td>2.24e+105</td>
<td>3.27e+147</td>
</tr>
<tr>
<td>1000</td>
<td>4.50e+02</td>
<td>3.30e+271</td>
<td>5.36e+297</td>
</tr>
</tbody>
</table>

Can this possibly be correct?
Note that
\[ \|x_2\| / \sqrt{N} \leq \|x\|_\infty \leq \|x\|_2. \]

Can you prove this?

So
\[ \|Ax\|_\infty \leq \|Ax\|_2 \leq \|A\|_2 \|x\|_2 \leq \sqrt{N} \|A\|_2 \|x\|_\infty. \]

Which implies that
\[ \|A\|_\infty \leq \sqrt{N} \|A\|_2 \text{ and so } \kappa_\infty(A) \leq N \kappa_2(A). \]

The $\kappa_\infty$ column in the table is wrong!
What happened?

- The $\ell^{\infty}$ condition estimator in Matlab uses the LU factorization.
- So the instability affects the estimate and it’s wrong.

Bottom line: the evil matrix is not badly conditioned.
Complete Pivoting

- We swapped rows to stabilize computation of $L$.
- How about swapping columns for $U$?

When done correctly, this is **complete pivoting**.
Complete Pivoting Algorithm

Pick the pivot element for column $j$ by making swapping

- Row $j$ and row $k$
- Column $j$ and column $l$

where

$$|a_{kl}| = \max_{m,n \geq j} |a_{mn}|$$

Contrast with partial pivoting, where you only search over the row index.
A ← A

for \( j = 1 : N \) do

Find \( k, l \) so that \( a_{kl} = \max_{m,n \geq j} |a_{mn}| \)

Swap rows \( j \) and \( k \), columns \( j \) and \( l \); record the swaps

for \( i = j + 1 : N \) do

\[ a_{ij} = a_{ij} / a_{jj} \] \{Compute the multipliers. Store them in the strict lower triangle of \( A \).\}

for \( k = j + 1 : N \) do

\[ a_{ik} = a_{ik} - a_{ij} a_{jk} \] \{Do the elimination.\}

end for

end for

end for
Incremental Cost

- At stage $j$ in the outer loop you do $(N - j)^2$ compares,
- which is equivalent to $(N - j)^2$ adds.
- So you doing

$$\int_{1}^{N} (N - j)^2 \, dj = \frac{N^3}{3} + O(N^2) \text{ extra adds}$$

- This is the example where adds and multiplies are not paired to leading order.

If adds and multiplies cost the same, net increase of 50%.
Not really worth it.
Here we get $A = P_L L U P_R$ where $P_L$ and $P_R$ are permutation matrices.

The solve goes like

- Multiply $Ax = b$ by $P_L^T$ to get $L U P_R x = P_L b \equiv b'$
- Solve $L z = b'$
- Solve $U w = z$
- Set $x = P_R^T w$

So ...
Just to make sure

\[ Ax = P_L L U P_R x = P_L L U P_R P_R^T w \]
\[ = P_L L U w = P_L L z \]
\[ = P_L b' = P_L P_L^T b = b \]
Here's a fact.

\[ g_{CP} \approx N^{0.5+\ln(N/4)} \]

which is much better than \( 2^{N-1}/N \).

The conjecture is that \( g_{CP} = O(N) \).

Prove that and get PhD and cool job right now!
Uniqueness of LU factorization: I

Put all pivots as part of $A$. Replace $A$ by $P_L^T A P_R^T$.
For unit lower triangular $L$ and upper triangular $U$, the equation

$$LU = A$$

is a system of $N^2$ equations in $N^2$ unknowns that is easy to solve.
To start, since $l_{11} = 1$, the equation for the first row is

$$u_{1j} = a_{1j}, \ 1 \leq j \leq N.$$

so the first row of $U$ is the first row of $A$. 
Uniqueness of LU factorization: II

For the second row, note that $l_{22} = 1$ and $l_{2k} = 0$ if $k > 2$.

\[ a_{2j} = \sum_{k=1:N} l_{2k} u_{kj} = \sum_{k=1:2} l_{2k} u_{kj} \]

\[ = l_{21} u_{1j} + l_{11} u_{2j} \]

So, for $1 \leq j \leq N$, \[ u_{2j} = a_{2j} - l_{21} u_{1j} \]

and so on.

I may well ask you to prove things like this in detail on an exam.
Integral Equation Example

We will use the trapezoid rule to solve

$$u(x) - \int_0^1 \sin(x - y)u(y) \, dy = f(x).$$

We will test the quality of the results with

- the method of manufactured solutions and
- a grid refinement study.
The method of manufactured solutions

- Pick $x^*$ and set $b = Ax^*$.
- Solve $Ax = b$ and obtain $\tilde{x}$.
- Compare $\tilde{x}$ to $x^*$. 
Here’s a function that examines a $10 \times 10$ matrix $A$.

```matlab
function ediff = testme(A)
% TESTME sees if the matlab backslash can solve an easy problem.

% xstar = ones(10,1);
bstar = A*xstar;
xtilde = A\bstar;
ediff = norm(xtilde - xstar,1);
```
Composite Trapezoid Rule

\[ \int_{0}^{1} u(x) \, dx \approx I_h(u) = \sum_{i=1}^{N} w_i u(x_i) \]

where

\[ h = 1/(N - 1); \quad x_i = (i - 1) \times h; \quad w_i = \begin{cases} \frac{h}{2} & i = 1 \text{ or } i = N \\ h & 2 \leq i \leq N - 1 \end{cases} \]

Theorem: If \( u \) is sufficiently smooth the composite trapezoid rule is second order accurate.

\[ I_h(u) - \int_{0}^{1} u(x) \, dx = O(h^2). \]
Approximate the integral equation

\[ u_i - \sum_{j=1}^{N} \int_{0}^{1} \sin(x_i - x_j) w_j u_j = f(x_i). \]

Theorem: The integral equation has a unique solution \( u^* \) for every continuous \( f \). The approximate problem has a unique solution \( u^h \) for each \( N \). \( u^* \) is as differentiable as \( f \). And if \( f \) is sufficiently smooth then

\[ \max_{1 \leq i \leq N} |u_i - u^*(x_i)| = O(h^2) \]
Building the Approximation

For a given $N$ and $h$ define

$$a_{ij}^h = \delta_{ij} - \sin(x_i - x_j)w_j \text{ and } f_i^h = f(x_i).$$

The approximate integral equation is $A^h u^h = f^h$.

Let $u_i^{h*} = u(x_i)$, where $u$ is the solution of the integral equation. The error estimate is

$$E_h \equiv \|u^h - u_i^{h*}\|_\infty = O(h^2).$$
If you knew $u^*$, you could compute $u^{*h}$ and therefore $E_h$. You’d expect 

$$E_{2h}/E_h \approx 4.$$ Why?

Let’s check that.
Method of Manufactured Solutions

Set $u^*(x) \equiv 1$. Then

$$f(x) = 1 - \int_0^1 \sin(x - y) \, dy = 1 - \cos(x - 1) + \cos(x)$$

Now for the computation ...
Cycle through

- $h = 1/(10 \times 2^p), 1 \leq p \leq 5$
  
  So $N = 21, 41, 81, \ldots, 321$.

- Loop over $p$ and print out $E(2h)/E(h)$
We’ll get the weights and nodes right.

```matlab
for p=1:5
    N=(10*2^p) + 1; h=1/(N-1);
    w=h*ones(N,1); w(1)=w(1)*.5; w(N) = w(N)*.5;

    % Make sure the spatial grid points are in a column vector.
    x=(0:N-1)*h; x=x';
    f=ones(N,1) -cos(x-1)+cos(x);
```
Building the matrix and solving the problem

```matlab
ustar = ones(N,1);
A = eye(N,N);
for j = 1:N
  for i = 1:N
    A(i,j) = A(i,j) - sin(h*(i-j))*w(j);
  end
end
utest = A\f;
EH(p,2) = norm(utest - ustar, inf);
EH(p,1) = N;
end % end of p loop
EH(2:5,3) = EH(1:4,2) ./ EH(2:5,2);
```
Tastefully presented output

<table>
<thead>
<tr>
<th>N</th>
<th>E(h)</th>
<th>E(2h)/E(h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>1.02e-04</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>2.56e-05</td>
<td>4.00098e+00</td>
</tr>
<tr>
<td>81</td>
<td>6.39e-06</td>
<td>4.00025e+00</td>
</tr>
<tr>
<td>161</td>
<td>1.60e-06</td>
<td>4.00006e+00</td>
</tr>
<tr>
<td>321</td>
<td>3.99e-07</td>
<td>4.00002e+00</td>
</tr>
</tbody>
</table>

Code is integral_example.m on Moodle page.
What have we done?

You just learned to solve any 2nd kind Fredholm integral equation

\[ u(x) - \int_0^1 k(x, y)u(y) \, dy = f(x). \]

If there's a unique solution \( u^* \) for any \( f \) then all you do is this.

- Build \( A^h \) and \( f^h \).
- Solve \( A^h u^h = f^h \) with Gaussian elimination.

And then

\[ \lim_{h \to 0} \max_{1 \leq i \leq N} |u_i - u^*(x_i)| = 0. \]