Solving Sets of Linear Equations

Examples of Applications: Systems of linear equations can arise

- in discretization of PDEs
- in force balance systems (e.g., structural mechanics)
- in the study steady-state properties of reactor systems
- in problems involving currents and voltages in resistor circuits and mass-spring systems
- Such systems really pop up in lots of places.

The steady-state heat equation's solution gives us the long time temperature in a plate whose sides are held at constant temperature. Discretizing (see Gerald and Wheatley p. 111) $u_{xx} + u_{yy} = 0$ gives:

$$\begin{pmatrix} -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & -4 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \end{pmatrix} = \begin{pmatrix} -50 \\ -50 \\ -150 \\ 0 \\ 0 \\ -100 \\ -50 \\ -50 \\ -150 \end{pmatrix}$$

For systems of just two equations in two unknowns, one can just graph the lines to find the intersection. The solution to a 2x2 system of linear equations is something you should have studied in Junior High School.

Matrix notation **A**. *Rows*, *columns*, a_{ij}. Size of matrix ixj. Adding matrices of the same size. Ex.

$$\mathbf{A} = \begin{pmatrix} 4 & 7 & -5 \\ -4 & 2 & 12 \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} 1 & 5 & 4 \\ 2 & -6 & 3 \end{pmatrix}$$

Find **A+B** and **A-B**.

To multiply two vectors (dot product), take the product of a row vector and a column vector. Ex.

$$\begin{pmatrix} 5 & -3 & 2 \end{pmatrix} * \begin{pmatrix} 6 \\ -1 \\ -4 \end{pmatrix}$$

Multiplication of matrices: Across rows and down columns – taking dot product of one row with one column. Thus, the number of columns of the first matrix must equal the number of rows of the second matrix.

$$\text{Let} \quad \mathbf{C} = \begin{pmatrix} 5 & -3 \\ 0 & 3 \\ 1 & 2 \end{pmatrix}$$

Find **AC** and **BC**. We cannot compute **AB**. Note that, in general $\mathbf{AB} \neq \mathbf{BA}$. Sometimes one exists and the other does not. Even when both exist, they are usually not equal. Multiplying a matrix by a constant, ex. 4**C**.

Writing a set of linear equations in matrix form: Write out for $a_{11}x_1 + ... + a_{1n}x_n = b_1$ etc. through $a_{n1}x_1 + ... + a_{nn}x_n = b_n$ as $\mathbf{A}x = b$.

$$\begin{array}{lcl}
a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n & = & b_1 \\
a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n & = & b_2 \\
\cdots & = & . \\
\cdots & = & . \\
\cdots & = & . \\
a_{n1}x_1 + a_{n2}x_2 + ... + a_{nn}x_n & = & b_n
\end{array}$$

Can be written:

$$\mathbf{A}x = b, \quad \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ \cdots \\ x_n \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ . \\ . \\ b_n \end{bmatrix}$$

Define *diagonal matrix*, *identity matrix*, *lower triangular matrix*, *upper triangular matrix*, *trdiagonal matrix.*

Demonstrate how to find a determinant: first for 2 by 2, then the following example going across the first row, then go down the first column. One can use any row or column.

Also do it by the definition, writing out all 24 terms and assigning them a plus or minus sign.

$$\det \begin{pmatrix} 3 & 0 & -1 & 2 \\ 4 & 1 & 3 & -2 \\ 0 & 2 & -1 & 3 \\ 1 & 0 & 1 & 4 \end{pmatrix}$$

For lower triangular or upper triangular or diagonal matrices, the determinant is just the product of the diagonal entries. Ex.

$$\det \begin{pmatrix} 4 & 0 & 0 \\ 6 & 3 & 0 \\ 1 & 2 & -2 \end{pmatrix}$$

Eigenvalues: Those values that make det ($\mathbf{A}$ - $\lambda\mathbf{I}$) $=$ 0. Find det $\mathbf{A}$ and the eigenvalues of $\mathbf{A}$ for

$$\mathbf{A} = \begin{pmatrix} 1 & 3 \\ 4 & 5 \end{pmatrix}$$

The inverse of a matrix is $\mathbf{A}^{-1}$ such that $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\,\mathbf{A} = \mathbf{I}$. Ex.

$$\begin{pmatrix} 3 & 1 \\ 5 & 2 \end{pmatrix}\begin{pmatrix} 2 & -1 \\ -5 & 3 \end{pmatrix}$$

We can solve $\mathbf{A}x = b$ by multiplying both sides on the left by $\mathbf{A}^{-1}$.

Gaussian Elimination

If the system is triangular, one can use forward or backsubstitution to solve. For example,

$$\begin{array}{rcrcrcr} 5x_1 & + & 3x_2 & - & 2x_3 & = & -3 \\ & & 6x_2 & + & x_3 & = & -1 \\ & & & & 2x_3 & = & 10 \end{array}$$

Consider the system of linear equations:

$$\begin{array}{rcrcrcr} 4x_1 & - & 2x_2 & + & x_3 & = & 15 \\ -3x_1 & - & x_2 & + & 4x_3 & = & 8 \\ x_1 & - & x_2 & + & 3x_3 & = & 13 \end{array}$$

Put the system in matrix form $[\mathbf{A}|b]$. This is called the augmented matrix.

The essense of elimination is to reduce the coefficient matrix to a triangular matrix and then use back-substitution to find the solution.

Elementary row operations:

1. multiply a row by a constant
2. add/subtract one row from another;
3. swap two rows.

Work through Gaussian Elimination for the above system, getting the system in upper triangular form.

Work through Gauss-Jordan Elimination (similar to Gaussian elimination, but entries above the diagonal are also turned to zero each time you use a new column) for the above system, getting the system in diagonal form.

Count operations for Gaussian elimination (and backsubstitution) and Gauss-Jordan (getting zeroes above the diagonal).

Gaussian elimination $O(n^3/3)$ operations $\sum_{i=1}^{n-1} i(i+2)$

At the start, we need to eliminate first entry of $n-1$ rows. To eliminate each one takes one division and n multiplies ($n-1$ on left and one on the right hand side) for $(n-1)(n+1)$ operations. To eliminate second column, the number of rows and columns have been reduced by one to get $(n-2)n$ etc. until we get $(1)(3)$.

Gauss-Jordan $O(n^3/2)$ operations $(n-1)\sum_{i=1}^{n}(i+1)$

For Gauss-Jordan, we eliminate entries above the diagonal as well as below. So for the first column, we again have $(n-1)(n+1)$ operations, but for the second column, we still need to deal with $n-1$ other rows, or $(n-1)n$ operations. Then $(n-1)(n-1)$, $(n-1)(n-2)$ and so forth down to $(2)(1)$.

For forward or back substitution, we have first - one division, then one division and 1 multiply, one division and 2 multplies up to one division and $n-1$ multiplies for a total of $O(\frac{n^2}{2})$ operations.

Some issues to note about Gaussian Elimination:

- Dividing by zero. There will be problems if you have a zero on the diagonal.
- Build up of errors due to small divisors leading to round-off error.
- Some systems are so badly behaved that we cannot avoid major growth of errors. Small changes in the coefficients or in the right hand side cause large changes in the solution. Such systems are called *ill-conditioned systems.*

Consider the following system of linear equations and solve using 3,4,5,6 and 7 significant digits (use chopping):

$$\begin{array}{rcl} 0.0003x_1 + 3x_2 & = & 2.0001 \\ x_1 + x_2 & = & 1 \end{array}$$

The exact answer is $x_1 = 1/3$ and $x_2 = 2/3$.

$$3: \left(\begin{array}{rr|l} 0.0003 & 3 & 2.00 \\ 1 & 1 & 1 \end{array}\right) \to \left(\begin{array}{rr|l} 0.0003 & 3 & 2.00 \\ 0 & -9980 & -6650 \end{array}\right) \Rightarrow \begin{array}{rcl} x_2 & = & 0.666 \\ x_1 & = & \frac{0.01}{0.0003} = 33.3 \end{array}$$

$$4: \left(\begin{array}{rr|l} 0.0003 & 3 & 2.000 \\ 1 & 1 & 1 \end{array}\right) \to \left(\begin{array}{rr|l} 0.0003 & 3 & 2.000 \\ 0 & -9998 & -6665 \end{array}\right) \Rightarrow \begin{array}{rcl} x_2 & = & 0.6666 \\ x_1 & = & \frac{2.-1.999}{0.0003} = 3.333 \end{array}$$

$$5: \left(\begin{array}{rr|l} 0.0003 & 3 & 2.0001 \\ 1 & 1 & 1 \end{array}\right) \to \left(\begin{array}{rr|l} 0.0003 & 3 & 2.0001 \\ 0 & -9998.9 & -6665.9 \end{array}\right) \Rightarrow \begin{array}{rcl} x_2 & = & 0.66666 \\ x_1 & = & \frac{2.0001-1.9999}{0.0003} = 0.66666 \end{array}$$

$$6: \left(\begin{array}{rr|l} 0.0003 & 3 & 2.0001 \\ 1 & 1 & 1 \end{array}\right) \to \left(\begin{array}{rr|l} 0.0003 & 3 & 2.0001 \\ 0 & -9998.99 & -6665.99 \end{array}\right) \Rightarrow \begin{array}{rcl} x_2 & = & 0.666666 \\ x_1 & = & \frac{0.00011}{0.0003} = 0.366666 \end{array}$$

$$7: \left(\begin{array}{rr|l} 0.0003 & 3 & 2.0001 \\ 1 & 1 & 1 \end{array}\right) \to \left(\begin{array}{rr|l} 0.0003 & 3 & 2.0001 \\ 0 & -9998.999 & -6665.999 \end{array}\right) \Rightarrow \begin{array}{rcl} x_2 & = & 0.6666666 \\ x_1 & = & \frac{0.000101}{0.0003} = 0.3366666 \end{array}$$

Significant Digits	x_1	x_2	x_1Exact	x_2Exact
3	33.3	0.666	$0.333\overline{3}$	$0.666\overline{6}$
4	3.333	0.6666	$0.333\overline{3}$	$0.666\overline{6}$
5	0.66666	0.66666	$0.333\overline{3}$	$0.666\overline{6}$
6	0.366666	0.666666	$0.333\overline{3}$	$0.666\overline{6}$
7	0.3366666	0.6666666	$0.333\overline{3}$	$0.666\overline{6}$

Work the problem switching rows.

$$\text{3digits}: \left(\begin{array}{rr|l} 1 & 1 & 1 \\ 0.0003 & 3 & 2.00 \end{array}\right) \to \left(\begin{array}{rr|l} 1 & 1 & 1 \\ 0 & 2.99 & 1.99 \end{array}\right) \Rightarrow \begin{array}{rcl} x_2 & = & 0.666 \\ x_1 & = & 0.334 \end{array}$$

For 5 or more digits this yields:

$$\left(\begin{array}{rr|l} 1 & 1 & 1 \\ 0.0003 & 3 & 2.0001 \end{array}\right) \to \left(\begin{array}{rr|l} 1 & 1 & 1 \\ 0 & 2.9997 & 1.9998 \end{array}\right)$$

Solution: Partial pivoting. Pivot the row (swap rows) with the largest value (in magnitude) in the column of interest such that this value is on the diagonal. By doing this, no row is multiplied by a number greater than 1 during elimination, so errors cannot grow very fast. (No worse than doubling). Just remember to keep the right hand side with its row. The solution to the above system becomes – with pivoting:

Significant Digits	x_1	x_2	x_1Exact	x_2Exact
3	0.334	0.666	$0.333\overline{3}$	$0.666\overline{6}$
4	0.3335	0.6666	$0.333\overline{3}$	$0.666\overline{6}$
5	0.33334	0.66666	$0.333\overline{3}$	$0.666\overline{6}$
6	0.333334	0.666666	$0.333\overline{3}$	$0.666\overline{6}$
7	0.3333334	0.6666666	$0.333\overline{3}$	$0.666\overline{6}$

A matrix that has entries no larger in magnitude than unity can still end up with large values after elimination. Consider (note that no partial pivoting is needed):

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 1 \\ -1 & -1 & 1 & 0 & 0 & 1 \\ -1 & -1 & -1 & 1 & 0 & 1 \\ -1 & -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & -1 & 1 \end{pmatrix}$$

Systems with complex coefficients or complex right hand sides can be broken up into real and imaginary parts. Say, let $x_j = c_j + id_j$, let $a_{kl} = e_{kl} + if_{kl}$ and let $b_j = g_j + ih_j$in $\mathbf{A}x = b$. Then equate the real parts of the equations and the imaginary parts of the equations. This yields twice as many equations in twice as many unknowns.

Scaling - It may also be useful to divide each row by its largest entry so that in some sense all the equations have similar scale. This makes the largest entry 1 in each equation at the start. Consider (chopping, using 3 significant digits) the system:

$$\begin{array}{rcrcl} 2x_1 & + & 100,000x_2 & = & 100,000 \\ x_1 & + & x_2 & = & 2 \end{array}$$

We obtain $x_1 = 0$ and $x_2 = 1$ instead of the correct solution $x_1 = 1.00002$ and $x_2 = 0.99998$. By scaling and pivoting, we get $x_1 = 1$ and $x_2 = 1$.

Whenever you solve a system of linear equations and obtain an answer, you can check your work by plugging in that solution and see if the right hand side matches. This is not fool-proof since an ill-conditioned system can give a small residual, $b - \mathbf{A}x$, when x is not very close to the true solution x_*. (We will discuss this further at a later time).

LU (Lower-Upper) Decomposition

This method is especially useful if we are solving for several right hand sides.

1. Place 1s on the diagonal of the **L** matrix.
2. Keep the multipliers from Gaussian elimination (with minus sign) in the relevant spot in the **L** matrix.
3. The **U** matrix is the matrix obtained at the end of Gaussian elimination.
4. Check that $\mathbf{LU} = \mathbf{A}$.
5. Once we have **L** and **U**, we let $y = \mathbf{U}x$, then $\mathbf{L}y = b$ and we can solve (using forward substitution) for y.
6. Then solve $\mathbf{U}x = y$ (back substitution) for x.

Work for the 3x3 system several pages back.

Since det **L** det **U** = det **LU** = det **A** and since det **L** = 1, we have det **A** = det **U**.

The number of operations for LU is the same as for Gaussian Elimination (except there is no right hand side $\sum_{i=1}^{n-1} i(i+1)$ for $O(n^3/3)$ operations
Forward and back-substitution are both needed for $2O(n^2/2)$ or $O(n^2)$ operations.

We cannot find an LU decomposition for the following matrix since there is a zero in the position to be divided by in at least one step (the first step). However, we can find the LU decomposition for a permuted matrix. The record keeping is more complicated since you must keep track of changes on the right-hand side as well.

$$\begin{array}{rcrcrcrcr} 0x_1 & + & 2x_2 & + & 0x_3 & + & 1x_4 & = & 0 \\ 2x_1 & + & 2x_2 & + & 3x_3 & + & 2x_4 & = & -2 \\ 4x_1 & - & 3x_2 & + & 0x_3 & + & 1x_4 & = & -7 \\ 6x_1 & + & 1x_2 & - & 6x_3 & - & 5x_4 & = & 6 \end{array}$$

Full pivoting involves moving the highest coefficient in the part of the matrix not yet eliminated to the diagonal in the row that is next in the elimination process. This involves switching rows *and* columns. It gets much more complicated.

Consider the number of operations for solving once we have the LU-decomposition (forward and backsubstitution each are $O(n^2/2)$ operations for a total of $O(n^2)$ operations).

We can solve $\mathbf{A}x = b$ whenever the determinant is non-zero.
LU should work whenever Gaussian Elimination or Gauss-Jordan works.

With pivoting, LU and Gaussian Elimination and Gauss-Jordan work whenever the determinant of $\mathbf{A}$ is non-zero.

Other direct methods include Cholesky's method (which uses $\mathbf{LL}^T$ for a symmetric, positive matrix) and Crout reduction. (We place the 1's on the diagonal of $\mathbf{U}$ as opposed to 1's on the diagonal of $\mathbf{L}$ in LU-decomposition. This involves solving for the first row of $\mathbf{L}$ followed by the first column of $\mathbf{U}$, followed by the second row of $\mathbf{L}$, followed by the second column of $\mathbf{U}$ and so on).

Count the number of operations needed to solve a tridiagonal system with Gaussian elimination and back-substitution.

- To eliminate the first column takes 3 operations (mult/divide. This includes the right hand side)
- To eliminate the second column also takes 3 operations, etc.
- The elimination of the second to last column takes 3 operations
- The total is $3(n-1)$ operations.
- The backsubstitution takes 1 operation for x_n but 2 for each of the other x values, for $\sim 2n$ operations.
- Thus, the total is about $5n$ operations.

It is always theoretically possible to avoid division by zero when a square set of equations (an $n \times n$ system) has a unique solution. There are several cases where there is not a unique solution. Examples include:

- There are fewer equations than unknowns: $x + y = 5$ has infinite number of solutions.
- The system can be inconsistent $x + y = 5$ and $x + y = 7$. Gaussian elimination would give (without roundoff) a row of zeroes or a row of zeroes and non-zero value(s) on the right.

Determinants to compute by the definition involves adding $n!$ terms, each of which is found by $n-1$ multiplications. Using Gaussian elimination, noting

1. swapping rows multiplies the determinant by -1

2. adding rows does not affect the determinant
3. multiplying a row by a constant multiplies the determinant by the same constant

is more efficient.

To find the inverse of $\mathbf{A}$, we can use Gaussian Elimination with n right hand sides each of which is a column of the identity matrix. I.e., we are solving $\mathbf{AX}$=$\mathbf{I}$. Work and count operations for:

$$\begin{pmatrix} 1 & -1 & 2 \\ 3 & 0 & 1 \\ 1 & 0 & 2 \end{pmatrix}$$

Meaning of Inverse: Since $\mathbf{A}^{-1}\mathbf{A} \ = \ \mathbf{I}$, we have

- $\mathbf{A}x \ = \ b$
- $\mathbf{A}^{-1}\mathbf{A}x \ = \ \mathbf{A}^{-1}b$
- So, $\mathbf{I}x \ = \ x \ = \ \mathbf{A}^{-1}b$ Thus, if we know the inverse of a matrix, we can understand how a change in any component of the right hand side b influences any given component of the solution x. We have $x_j \ = \ a_{j1}^{-1}b_1 + a_{j2}^{-1}b_2 + \ldots + a_{jn}^{-1}b_n$, where the subscript ij denotes the ij element in $\mathbf{A}^{-1}$.
- Another way to look at this is if we have the solutions to $\mathbf{A}x = \delta_i$, where δ_i is the standard normal unit vector, which is all zero except that the ith component is 1, and $\mathbf{y}_i$ is the solution for right hand side δ_i, then the solution to $\mathbf{A}x = b$ is $x = b_1\mathbf{y}_1 \ + \ b_2\mathbf{y}_2 \ + \ \ldots \ + \ b_n\mathbf{y}_n$.

Operations to compute the matrix inverse using Gauss-Jordan: First column takes $(n+1)(n-1)$ operations, the $(n+1)(n-1)$ operations etc. since for each column dealt with we have to add a new column on the right that had only zeroes added until this point. We need to deal with all n columns so we have $n(n+1)(n-1)$ or $n^3 - n$ operations to turn $\mathbf{A}$ into a diagonal matrix. Then divide by the n diagonals for n^3 operations.

Operations to compute the matrix inverse using Gaussian Elimination (LU): $O(n^3/3)$ operations to compute LU. Then solve for n right hand sides with forward and back substitution for $n * 2 * n^2/2$ or n^3 operations for a total of $4n^3/3$ operations. Taking into account zeroes on the right hand side gives $n * n^2/2$ for the backsubstitutions but only $1^2/2 + 2^2/2 + ... + n^2/2$ for the forward substitutions ,since we get all zeroes until the first 1 in the δ_i (right hand side) column, giving $\frac{n^3}{2} + \frac{n^3}{6} = \frac{2n^3}{3}$ operations once we have $\mathbf{L}$ and $\mathbf{U}$, for a total of n^3 operations to find the matrix inverse using LU-decomposition.

Compare operations for finding the inverse once and solving $x = \mathbf{A}^{-1}b$ for m right hand sides, vs. finding the LU decomposition and forward and back-substitution for m right hand sides. Once we have the inverse, then $\mathbf{A}^{-1}b$ takes n^2 operations: same as forward and back substitution once we have $\mathbf{L}$ and $\mathbf{U}$. So, obtaining the inverse doesn't save any work over LU. As a matter of fact, it is more work since obtaining the inverse takes more work than finding the LU-decomposition.

Tridiagonal matrix: When performing LU decomposition on a tridiagonal matrix, $\mathbf{L}$ only has entries on the diagonal and the subdiagonal, while $\mathbf{U}$ only has entries on the diagonal and superdiagonal. The number of operations is greatly reduced. Finding $\mathbf{L}$ and $\mathbf{U}$ takes only $2n$ operations. Forward and back-substitution each take $2n$ operations for a total of $6n$ operations.

Similarly, for banded matrices, we have a reduced number of operations in performing LU decomposition. The number of operations is related to how wide the bands are. If the band is of width w, the LU part must increase by a factor like w^2 and the forward and back-substitution by factors like w.

Matrix and Vector Norms

Norms are a measure of the size or largeness of a matrix or vector.

Vector norms

- One-norm: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
- Two-norm: $\|x\|_2 = (\sum_{i=1}^{n} x_i^2)^{1/2}$ Pythagorean length
- Infinity-norm: $\|x\|_\infty = \max_{1\leq i\leq n}|x_i|$ maximum magnitude norm
- Ex. 2.6 p. 156 Gerald and Wheatley

Matrix norms

- One-norm: $\|\mathbf{A}\|_1$ = maximum (absolute) column sum
- Infinity-norm: $\|\mathbf{A}\|_\infty$ = maximum (absolute) row sum
- From Gerald and Wheatley (p. 157) compute 1 and ∞ norm for

$$\begin{bmatrix} 5 & 9 \\ -2 & 1 \end{bmatrix} \quad \begin{bmatrix} 0.1 & 0 \\ 0.2 & -0.1 \end{bmatrix} \quad \begin{bmatrix} 5 & -5 & 7 \\ -4 & 2 & -4 \\ -7 & -4 & 5 \end{bmatrix}$$

Iterative Improvement

Once we have a computed solution, we can check $b - \mathbf{A}x$, which is called the residual, r. To improve our approximation of x, we can solve $\mathbf{A}e = r$ (use double precision for r and this computation) for the error, e. If e is small, our original answer was probably good. If $\frac{||e||}{||x||} < 10^{-p}$ then x is probably good to p digits.

Discuss example on bottom of page 159 Gerald and Wheatley.

$$\begin{bmatrix} 4.23 & -1.06 & 2.11 \\ -2.53 & 6.77 & 0.98 \\ 1.85 & -2.11 & -2.32 \end{bmatrix} \mathbf{x} \ = \ \begin{bmatrix} 5.28 \\ 5.22 \\ -2.58 \end{bmatrix}$$

has solution $\begin{bmatrix} 1.000 \\ 1.000 \\ 1.000 \end{bmatrix}$ with 3-digit chopping, the computed value for $\mathbf{x}$ is $\begin{bmatrix} 0.991 \\ 0.997 \\ 1.000 \end{bmatrix}$. Then **Ax -b**, the residual, **r**, is $\begin{bmatrix} 0.0349 \\ -0.00246 \\ 0.0103 \end{bmatrix}$ and the correction term **e** with 3-digit chopping for **Ae=r** becomes $\begin{bmatrix} 0.00822 \\ 0.00300 \\ -0.00000757 \end{bmatrix}$. We add our original **x** and **e** to get a better value for **x** of $\begin{bmatrix} 0.999 \\ 1.000 \\ 1.000 \end{bmatrix}$.

Ill-conditioned systems

Error due to round-off can be large when the system itself is very sensitive to effects of small errors in the matrix or right hand side b. Such a system is called *ill-conditioned.*

For example: The following system has solution (1,1)

$$\begin{aligned} 1.01x_1 + 0.99x_2 &= 2 \\ 0.99x_1 + 1.01x_2 &= 2 \end{aligned}$$

while the almost identical system which follows has solution (2,0)

$$\begin{aligned} 1.01x_1 + 0.99x_2 &= 2.02 \\ 0.99x_1 + 1.01x_2 &= 1.98 \end{aligned}$$

and the following system has solution (0,2)

$$\begin{array}{rcrcl} 1.01x_1 & + & 0.99x_2 & = & 1.98 \\ 0.99x_1 & + & 1.01x_2 & = & 2.02 \end{array}$$

For small changes in input, we get large changes in output. Here, we would get small residuals even when we have poor approximations to the solution.

Notice that for the first system in this section, if we found the approximate solution (2,0), we would find that $b - \mathbf{Ax} = \begin{pmatrix} 0.02 \\ -0.02 \end{pmatrix}$ with $\|b - \mathbf{Ax}\|_\infty = 0.02$. Such a small error would usually indicate a good solution. However, since the system is ill-conditioned, the solution turns out not to be good at all.

Another, more extreme example is

$$\begin{array}{rcrcl} 1x_1 & + & 2x_2 & = & 3 \\ 1.0001x_1 & + & 2x_2 & = & 3.0001 \end{array}$$

The exact solution is (1,1), but if we guess the solution is (3,0) we find $\|b-\mathbf{Ax}\|_\infty = 0.0002$.

One more example is:

$$\begin{array}{rcrcl} 1.002x_1 & + & 1x_2 & = & 2.002 \\ x_1 & + & 0.998x_2 & = & 1.998 \end{array}$$

The exact solution is (1,1), but if we guess change the right hand side to (2.0021, 1.998), the solution is (-23.95,26.00). For the approximation (1,1), $\|b - \mathbf{Ax}\|_\infty = 0.0001$.

Condition Number

Condition number $\equiv \|\mathbf{A}\|\|\mathbf{A}^{-1}\|$. This is hard to compute (since we need $\mathbf{A}^{-1}$) and for an ill-conditioned system, we cannot compute $\mathbf{A}^{-1}$ accurately. In general, we know if the condition number is large, we've got problems.

If y is the approximation to the exact solution of $\mathbf{A}x = b$,

$$r = b - \mathbf{A}y = \mathbf{A}x - \mathbf{A}y = \mathbf{A}(x-y) = \mathbf{A}e_1$$

We can solve for e_1. Then $\mathbf{A}(y+e_1) - b = r_1$. Then, we can solve $\mathbf{A}e_2 = r_1$. This is called *iterative improvement.* Use double precision computation. It is worthwhile to use LU to perform iterative improvement since we don't have to start all over with each new right hand side. If the system is ill-conditioned, we may still not obtain a very good answer.

Analysis of Error $\mathbf{A}e_1 = r$ and $e_1 = \mathbf{A}^{-1}r$. Taking norms gives $||e_1|| \leq ||\mathbf{A}^{-1}||||r||$.

Also, from $r = \mathbf{A}e_1$, we have $||r|| \leq ||\mathbf{A}||||e_1||$ so: $\frac{||r||}{||\mathbf{A}||} \leq ||e_1|| \leq ||\mathbf{A}^{-1}||||r||$.

Doing the same for $\mathbf{A}x = b$ and $x = \mathbf{A}^{-1}b$, we get $\frac{||b||}{||\mathbf{A}||} \leq ||x|| \leq ||\mathbf{A}^{-1}||||b||$.

Dividing the first by the second gives: $\frac{1}{||\mathbf{A}||||\mathbf{A}^{-1}||}\frac{||r||}{||b||} \leq \frac{||e_1||}{||x||} \leq ||\mathbf{A}||||\mathbf{A}^{-1}||\frac{||r||}{||b||}$

So when the condition number is large, we have no idea how big the (relative) error may be. It may be really small or really large or somewhere in between.

Let's find the condition numbers in the ill-conditioned problems above.

$$\mathbf{A} = \begin{pmatrix} 1.01 & 0.99 \\ 0.99 & 1.01 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^{-1} = \begin{pmatrix} 25.25 & -24.75 \\ -24.75 & 25.25 \end{pmatrix}$$

Using the 1 norm, we find $||\mathbf{A}||_1 = 2$ and $||\mathbf{A}^{-1}||_1 = 50$ so the condition number is 100. Using the ∞ norm, we find the same values. Thus, $\frac{||e||}{||x||} \leq ||\mathbf{A}||||\mathbf{A}^{-1}||\frac{||r||}{||b||}$ implies that the error, as a fraction of the true solution is less than 100 times the residual as a fraction of the true right hand side. So a 1% error in getting the right hand side (i.e., $\mathbf{A}y$ compared to b) may produce an answer with 100% error in x.

For the other matrix considered in the previous section, we find:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 1.0001 & 2 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^{-1} = \begin{pmatrix} -10000 & 10000 \\ 5000.5 & -5000 \end{pmatrix}$$

Using the 1 norm, we find $||\mathbf{A}||_1 = 4$ and $||\mathbf{A}^{-1}||_1 = 15000.5$ so the condition number is 60002. So a 1% error in getting the right hand side (i.e., $\mathbf{A}y$ compared to b) may produce an answer with 60000% error in x.

A similar analysis (pp 163-4 Gerald and Wheatley) can be performed to understand the influence of errors in $\mathbf{A}$ on the solution to $\mathbf{A}x = b$. We find: $\frac{||x-y||}{||y||} \leq \text{Condition number}\frac{||\mathbf{E}||}{||\mathbf{A}||}$. Here, once again, y is the computed solution while $\mathbf{E}$ is the error in matrix $\mathbf{A}$.

Iterative Methods for Solving Linear Equations

The spectral radius of a matrix $\rho(\mathbf{A}) = |\max \lambda|$ i.e, the magnitude of the largest eigenvalue.

Theorem: $\mathbf{A}$ is a convergent matrix $\Leftrightarrow\ \rho(\mathbf{A}) < 1\ \Leftrightarrow \lim_{n\to\infty} \mathbf{A}^n x = 0\ \forall x$

Example: Check the eigenvalues of

$$\begin{pmatrix} 1/4 & 1/2 \\ 1/2 & 1/8 \end{pmatrix}$$

Jacobi's Method Solve $\mathbf{A}x = b$ by $x^{(k+1)} = \mathbf{T}x^{(k)} + c$. It's like fixed point iteration. This method is especially useful for large systems with lots of zeroes.

$$\begin{array}{rcrcrcr} 3x_1 & + & 1x_2 & + & 1x_3 & = & 5 \\ 1x_1 & - & 4x_2 & + & 2x_3 & = & 8 \\ 2x_1 & - & 3x_2 & + & 6x_3 & = & 15 \end{array}$$

Use Jacobi's method to solve: perform 2 iterations, beginning with $\mathbf{x} = (0,0,0)$

- Write $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$ so $(\mathbf{L} + \mathbf{D} + \mathbf{U})x = b$.
- Then $\mathbf{D}x = b - (\mathbf{L} + \mathbf{U})x$.
- Then let $x^{(k+1)} = \mathbf{D}^{-1}(b - (\mathbf{L} + \mathbf{U})x^{(k)})$. (Of course, we need the elements of $\mathbf{D}$ to be non-zero).
- We can write this as: $x^{(k+1)} = \mathbf{D}^{-1}b - \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})x^{(k)}$
- So $\mathbf{T} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ and $c = \mathbf{D}^{-1}b$.

Use the Gauss-Seidel method, where we use updated values whenever possible. This is the same as leaving $\mathbf{L}$ on the left.

- $(\mathbf{L} + \mathbf{D})x^{(k+1)} = b - \mathbf{U}x^{(k)}$
- so $x^{(k+1)} = (\mathbf{L} + \mathbf{D})^{-1}(b - \mathbf{U}x^{(k)})$.
- Notice that $\mathbf{L} + \mathbf{D}$ is lower triangular so it is pretty easy to find the inverse.
- So $\mathbf{T} = (\mathbf{L} + \mathbf{D})^{-1}\mathbf{U}$ and $c = (\mathbf{L} + \mathbf{D})^{-1}b$.

Jacobi's method and Gauss-Seidel are best for large sparse systems (with non-zero diagonal elements). In these cases, clever storage methods that don't save the zeroes can be employed. Note that LU-decomposition or finding the inverse of a matrix does not, in general, lead to sparse $\mathbf{L}$ and $\mathbf{U}$ or sparse $\mathbf{A}^{-1}$. Even tridiagonal matrices can have full $\mathbf{A}^{-1}$, e.g,

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 \end{pmatrix} \Rightarrow \mathbf{A}^{-1} = \frac{1}{7}\begin{pmatrix} 6 & -5 & 4 & -3 & 2 & -1 \\ -5 & 10 & -8 & 6 & -4 & 2 \\ 4 & -8 & 12 & -9 & 6 & -3 \\ -3 & 6 & -9 & 12 & -8 & 4 \\ 2 & -4 & 6 & -8 & 10 & -5 \\ -1 & 2 & -3 & 4 & -5 & 6 \end{pmatrix}$$

There are linear systems for which Jacobi converges and Gauss-Seidel does not and vice versa. These methods might not converge (if the spectral radius of $\mathbf{T} \geq 1$) or might converge slowly (if the spectral radius of $\mathbf{T}$ is close to 1). (See the analysis below).

Convergence of Iterative Schemes

We consider methods of the form: $x^{(k+1)} = \mathbf{T}x^{(k)} + c$. If the spectral radius of $\mathbf{T}$ is less than 1, then $(I - \mathbf{T})^{-1}$ exists and $(I - \mathbf{T})^{-1} = I + \mathbf{T} + \mathbf{T}^2 + ...$

If the spectral radius of $\mathbf{T}$ is less than 1, then $x^{(k+1)} = \mathbf{T}x^{(k)} + c$ converges to the exact solution x.

Proof: Consider

$$\begin{aligned} x^{(k+1)} &= \mathbf{T}x^{(k)} + c \\ &= \mathbf{T}(\mathbf{T}x^{(k-1)} + c) + c \\ &= \mathbf{T}^2(x^{(k-1)}) + (\mathbf{T} + I)c \text{ and continue to get} \\ x^{(k+1)} &= \mathbf{T}^{k+1}x_0 + (I + \mathbf{T} + \mathbf{T}^2 + ... + \mathbf{T}^k)c \end{aligned}$$

As $k \to \infty, \mathbf{T}^{(k+1)} \to \mathbf{0}$ and the expression in parentheses $\to (I - \mathbf{T})^{-1}$.

$$\begin{aligned} \text{So, } x &= (I - \mathbf{T})^{-1}c \\ \text{yielding } (I - \mathbf{T})x &= c \\ \text{and } x &= \mathbf{T}x + c. \end{aligned}$$

I.e., the limit of the sequence $x^{(k)}$ is the solution to $x = \mathbf{T}x + c$.

If $\|x - x^{(k)}\| = \|\mathbf{T}x - \mathbf{T}x^{(k-1)}\| \leq \|\mathbf{T}\|\|x - x^{(k-1)}\| \leq \|\mathbf{T}\|^k\|x - x^{(0)}\|$, then the error goes to zero (first order convergence with constant the norm of $\mathbf{T}$) as k gets large, if the matrix $\mathbf{T}$ is convergent.

- So for Jacobi, we need $\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})$ to have eigenvalues less than 1 in absolute value
- for Gauss-Seidel, we need $(\mathbf{L}+\mathbf{D})^{-1}\mathbf{U}$ to have eigenvalues less than 1 in absolute value.
- Since convergence is linear (at rate of the largest eigenvalue), we can apply Aitken's method or Steffensen's method to speed up convergence.

If $\mathbf{A}$ is diagonally dominant, then both Gauss-Seidel and Jacobi's methods converge to the exact solution. (Diagonally dominant means that in each row the magnitude of the diagonal element is greater than the sum of the magnitudes of the elements in the rest of the row). For a diagonally dominant matrix, $\mathbf{A}$, the row sum of $\mathbf{T}$ (the ∞ norm) is less than 1, so by Gershgorin's theorem, all eigenvalues must have magnitude less than 1.

We may need to reorder the rows (the equations) to get Jacobi or Gauss-Seidel to converge. For example, consider the system:

$$\begin{array}{rcrcr} 11x_1 & + & 13x_2 & = & 286 \\ 11x_1 & - & 9x_2 & = & 99 \end{array}$$

We swap the rows to make the system diagonally dominant.
Swapping the rows to make $\mathbf{A}$ diagonally dominant will lead to convergence. However, sometimes even swapping rows won't help.

A variation on Gauss-Seidel and Jacobi is called *successive overrelaxation* or SOR. The idea is to take a combination of the old iterate and the new iterate. I.e.,

$$x_i^{new} = \lambda x_i^{new} + (1-\lambda)x_i^{old}.$$

When $\lambda = 1$, we have Gauss-Seidel. When $1 < \lambda < 2$, we have SOR. The best value of λ is problem specific (that means it depends on the particular problem). One would only use this method and spend the time finding the "best" λ if the system came up often, as occurs in solving certain partial differential equations numerically.

Power Method for Eigenvalues

This method helps find the largest eigenvalue (in absolute value) and its corresponding eigenvector for a matrix. The Power Method can be used to prove Jacobi or Gauss-Seidel will converge and which one ought to do better.

In general, it is difficult to solve for all eigenvalues and eigenvectors. One strategy might be to take $\mathbf{A} - \lambda\mathbf{I}$, use Gaussian Elimination and find the determinant for different values of λ. We might try to combine it with Bisection or Secant method to find an eigenvalue, but we would not know if this is the largest. It also would not provide the eigenvector.

The Power method is easy.

- We guess an eigenvector $\mathbf{x_0}$ (Often we use $(1,1,...)^T$).
- Multiply $\mathbf{Ax_0}$ and let $\mathbf{x_1} = \mathbf{Ax_0}/$(largest value in $\mathbf{Ax_0}$) (This way entries won't grow).
- Keep on repeating until $||\mathbf{x_{m+1}} - \mathbf{x_m}||$ is small.
- Then, $\mathbf{x_m}$ is the desired eigenvector and the eigenvalue is $\mathbf{Ax_{m+1}/x_m}$. All these quotients should be about the same since $\mathbf{Ax}^* = \lambda\mathbf{x}^*$.

Perform the Power Method on the matrices

$$\begin{pmatrix} 1/4 & 1/2 \\ 1/2 & 1/8 \end{pmatrix} \text{ and } \begin{pmatrix} 0 & 5 & 6 \\ 3 & 0 & 4 \\ 2 & 4 & 0 \end{pmatrix}$$

The first matrix has eigenvector (1.000,0.883) and eigenvalue of 0.691.
The second matrix has eigenvector (1.000,0.738,0.652) and eigenvalue of 7.598611.

Convergence of the Power Method for the case where the eigenvalue desired is distinct:

- Denote $|\lambda_1| > |\lambda_2| \geq ... \geq |\lambda_n|$. Write the initial guess as $c_1\mathbf{v_1} + c_2\mathbf{v_2} + ... + c_n\mathbf{v_n}$.
- Then $\mathbf{A}(c_1\mathbf{v_1} + c_2\mathbf{v_2} + ... + c_n\mathbf{v_n}) = c_1\lambda_1\mathbf{v_1} + c_2\lambda_2\mathbf{v_2} + ... + c_n\lambda_n\mathbf{v_n}$ and
- $\mathbf{A}^m(c_1\mathbf{v_1} + c_2\mathbf{v_2} + ... + c_n\mathbf{v_n}) = c_1\lambda_1^m\mathbf{v_1} + c_2\lambda_2^m\mathbf{v_2} + ... + c_n\lambda_n^m\mathbf{v_n}$
- Divide by λ_1^m and all terms except the first go to zero and we find $c_1\mathbf{v_1}$.
- Convergence is linear with constant $\left|\frac{\lambda_2}{\lambda_1}\right|$. Thus, we can speed up convergence using Aitken's Method or Steffensen's Method.

Inverse Power Method: Solve $\mathbf{Ax_1} = \mathbf{x_0}$. Then divide by the largest value in $\mathbf{x_1}$. Continuing for many steps should give the largest eigenvalue of $\mathbf{A}^{-1}$. So using the Power Method with the Inverse Power Method and taking the product of the results gives the condition number.

Nonlinear Systems of Equations

Fixed Point Iteration Methods

Consider the system of nonlinear equations:

$$\begin{aligned} x^2 + xy &= 10 \\ y + 3xy^2 &= 57 \end{aligned}$$

Here, we can solve for y in the first equation and substitute into the second equation to obtain an equation in one variable (x) and use a one-variable method to find x. Typically, this will not be possible or convenient for larger systems.

We can also use Seidel's method (which is really the same as Gauss-Seidel). In this case, we could let:

$$\begin{aligned} x_{i+1} &= \tfrac{10-x_i^2}{y_i} \\ y_{i+1} &= 57 - 3x_{i+1}y_i^2 \end{aligned}$$

However, this rearrangement leads to divergence of the iterates. On the other hand, the rearrangement

$$\begin{aligned} x_{i+1} &= \sqrt{10 - x_i y_i} \\ y_{i+1} &= \sqrt{\tfrac{57-y_i}{3x_{i+1}}} \end{aligned}$$

does lead to convergence when the initial guess is close to the root $(2,3)$.

If the nonlinear system is written in the form:

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

then Seidel's method will converge to a solution if $|f_x| + |g_x| < 1$ and $|f_y| + |g_y| < 1$ near the root and the initial guess is close enough to the root. Just like with fixed point iteration schemes, where we play with rearrangements in the hope that one will work, this method is not very efficient. It's more difficult to get such methods to work for a system, in general, than for a single equation.

Newton's Method for Systems

Recall Newton's method for solving $f(x) = 0$ is $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. Suppose we want to solve the system of nonlinear equations:

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

We perform a derivation similar to that of Newton's method for one equation:

$$\begin{aligned} f(x+\Delta x, y+\Delta y) &= f(x,y) + \Delta x f_x(x,y) + \Delta y f_y(x,y) + ... \\ g(x+\Delta x, y+\Delta y) &= g(x,y) + \Delta x g_x(x,y) + \Delta y g_y(x,y) + ... \end{aligned}$$

We are trying to find Δx and Δy such that $f(x+\Delta x, y+\Delta y) = 0$ and $g(x+\Delta x, y+\Delta y) = 0$. So we need to solve:

$$\begin{pmatrix} -f(x,y) \\ -g(x,y) \end{pmatrix} = \begin{pmatrix} f_x(x,y) & f_y(x,y) \\ g_x(x,y) & g_y(x,y) \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$$

$$\text{Thus,} \quad \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = \frac{1}{f_x g_y - g_x f_y} \begin{pmatrix} -g_y(x,y) & f_y(x,y) \\ g_x(x,y) & -f_x(x,y) \end{pmatrix} \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$$

For larger systems of equations, we would solve for $\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$ using Gaussian Elimination.

The matrix

$$\begin{pmatrix} f_x(x,y) & f_y(x,y) \\ g_x(x,y) & g_y(x,y) \end{pmatrix}$$

is called the Jacobian matrix, J, so Newton's method for systems can be written:

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} - J^{-1} \begin{pmatrix} f(x_n, y_n) \\ g(x_n, y_n) \end{pmatrix}$$

Consider: $\begin{aligned} x^2 + x - y^2 &= 1 \\ y - \sin x^2 &= 0 \end{aligned}$ Set up and work one step with $(x,y) = (0,0)$.

Pitfalls of Newton's Method for Systems

- We need a good starting guess. We usually can only find one on the basis of trial and error. It is difficult to graph functions of more than one variable to approximate a solution and obtain a reasonably good starting guess.
- Lots of (partial) derivatives are needed. One variation is to update the Jacobian only every few steps or to use numerical derivatives to approximate the Jacobian.
- Another variation is to consider $f^2 + g^2 + \ldots = 0$ and use methods to minimize a function of many variables, such as steepest descent or conjugate gradient.