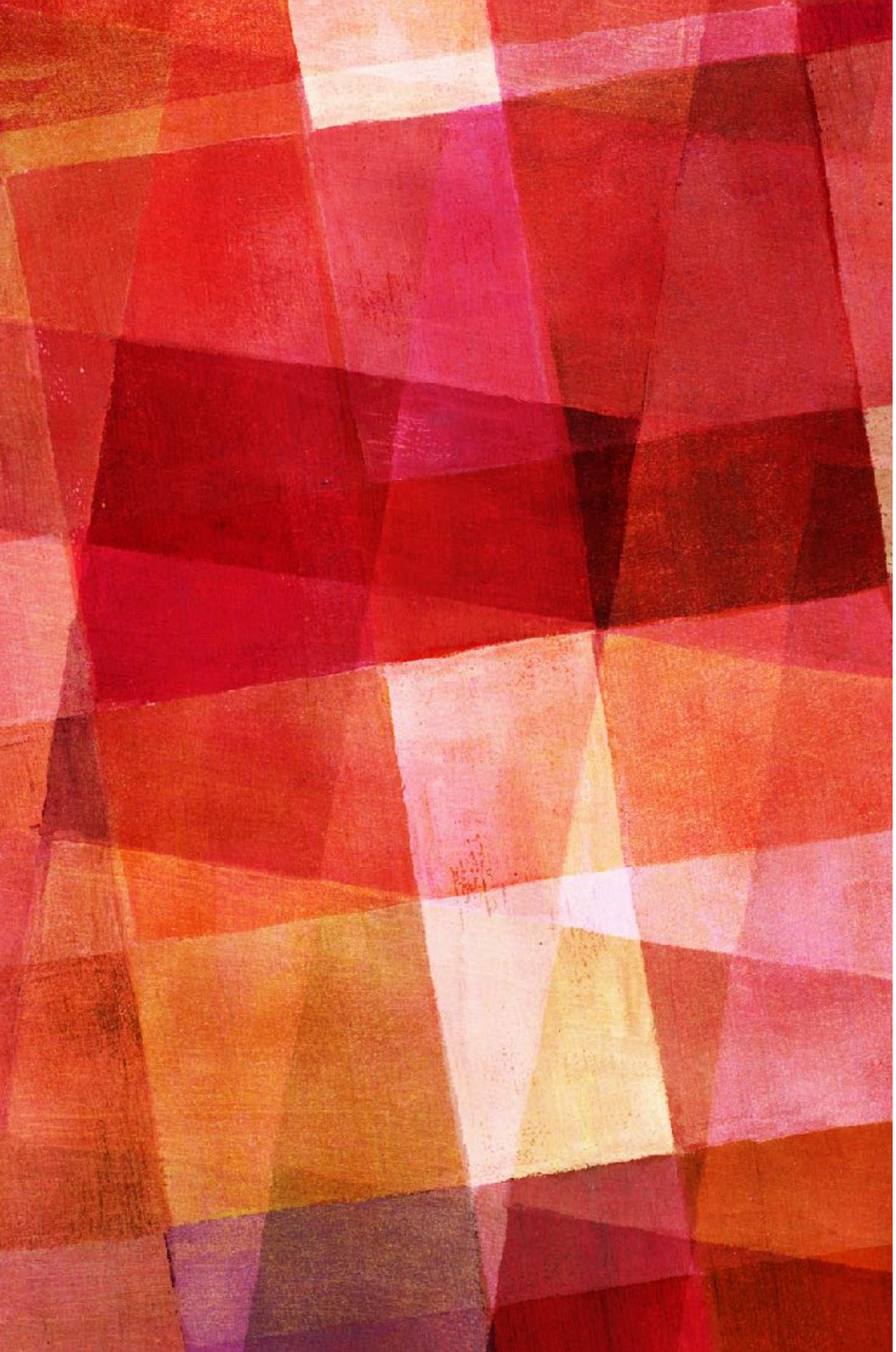
SYSTEMS OF LINEAR EQUATIONS

Ari Maller



LINEAR ALGEBRA

- Gaussian Elimination
- Backsubstitution
- LU Decomposition
- Eigen vectors and Eigen values

LINEAR EQUATIONS

- Systems of linear equations occur regularly in physics. If there are only a few equations we can readily solve them by hand, but if the number of equations becomes large it is much easier to solve them on a computer.
- A system of linear equations can be written as matrix multiplication $\mathbf{Ax} = \mathbf{v}$.
- One way to solve this equation is to invert the matrix $\mathbf{A}$ to get $\mathbf{x} = \mathbf{A}^{-1}\mathbf{v}$.
- Numerically this tends to be a complicated and inefficient process. There are other methods of solving this type of equation that are faster and more straightforward.

GAUSSIAN ELIMINATION

- Probably the most straightforward way to solve simultaneous equations is called Gaussian elimination. First though a few rules of linear algebra:

  1. If we multiply any row of the matrix **A** by any constant and the corresponding row of the vector **v** by the same constant the solution doesn't change.

  2. If we add to or subtract from any row of **A** a multiple of any other row and we do the same for the vector **v** then the solution doesn't change.

- Using these two rules we can apply the method of Gaussian elimination.

GAUSSIAN ELIMINATION

- Let's start with an example system of equations.

$$2w + x + 4y + z = -4$$
$$3w + 4x - y - z = 3$$
$$w - 4x + y + 5z = 9$$
$$2w - 2x + y + 3z = 7$$

- So writing as a matrix equation we would have:

$$\begin{bmatrix} 2 & 1 & 4 & 1 \\ 3 & 4 & -1 & -1 \\ 1 & 4 & 1 & 5 \\ 2 & 2 & 1 & 3 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -4 \\ 3 \\ 9 \\ 7 \end{bmatrix}$$

- which we can refer to as $\mathbf{Ax} = \mathbf{v}$.

- Gaussian elimination will rearrange A and v so that A is upper triangle, that is all zeros below the diagonal.

GAUSSIAN ELIMINATION

- First we divide the first row by the top-left element of the matrix (2 in this case). We must divide the top element in the vector for this to still be equal.

$$\begin{bmatrix} 1 & 0.5 & 2 & 0.5 \\ 3 & 4 & -1 & -1 \\ 1 & 4 & 1 & 5 \\ 2 & 2 & 1 & 3 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 3 \\ 9 \\ 7 \end{bmatrix}$$

- Next, the first element in the second row is a 3. If we subtract 3 times the first row this will make the first element of the second row 0.

$$\begin{bmatrix} 1 & 0.5 & 2 & 0.5 \\ 0 & 2.5 & -7 & -2.5 \\ 1 & 4 & 1 & 5 \\ 2 & 2 & 1 & 3 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 9 \\ 9 \\ 7 \end{bmatrix}$$

GAUSSIAN ELIMINATION

- Now we do a similar trick for the 3rd and 4th rows. Multiple the top row by the first element in those rows and subtract.

$$\begin{bmatrix} 1 & 0.5 & 2 & 0.5 \\ 0 & 2.5 & -7 & -2.5 \\ 0 & -4.5 & 1 & 4.5 \\ 0 & -3 & -3 & 2 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 9 \\ 11 \\ 11 \end{bmatrix}$$

- Now we move on to the second row and make the diagonal element equal to 1.

$$\begin{bmatrix} 1 & 0.5 & 2 & 0.5 \\ 0 & 1 & -2.8 & -1 \\ 0 & -4.5 & 1 & 4.5 \\ 0 & -3 & -3 & 2 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 3.6 \\ 11 \\ 11 \end{bmatrix}$$

GAUSSIAN ELIMINATION

- Then we can again make the off diagonal elements 0 by multiplying them by the second row and subtracting.

$$\begin{bmatrix} 1 & 0.5 & 2 & 0.5 \\ 0 & 1 & -2.8 & -1 \\ 0 & 0 & -13.6 & 0 \\ 0 & 0 & -11.4 & -1 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 3.6 \\ 27.2 \\ 21.8 \end{bmatrix}$$

- Repeating this procedure on the 3rd and then 4th row yields,

$$\begin{bmatrix} 1 & 0.5 & 2 & 0.5 \\ 0 & 1 & -2.8 & -1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ 3.6 \\ -2 \\ 1 \end{bmatrix}$$

- From this process we have created a new equation **A'x=v'** that has the same solution **x**. But now **A'** is upper triangular which means we can solve it by backsubstitution.

BACKSUBSTITUTION

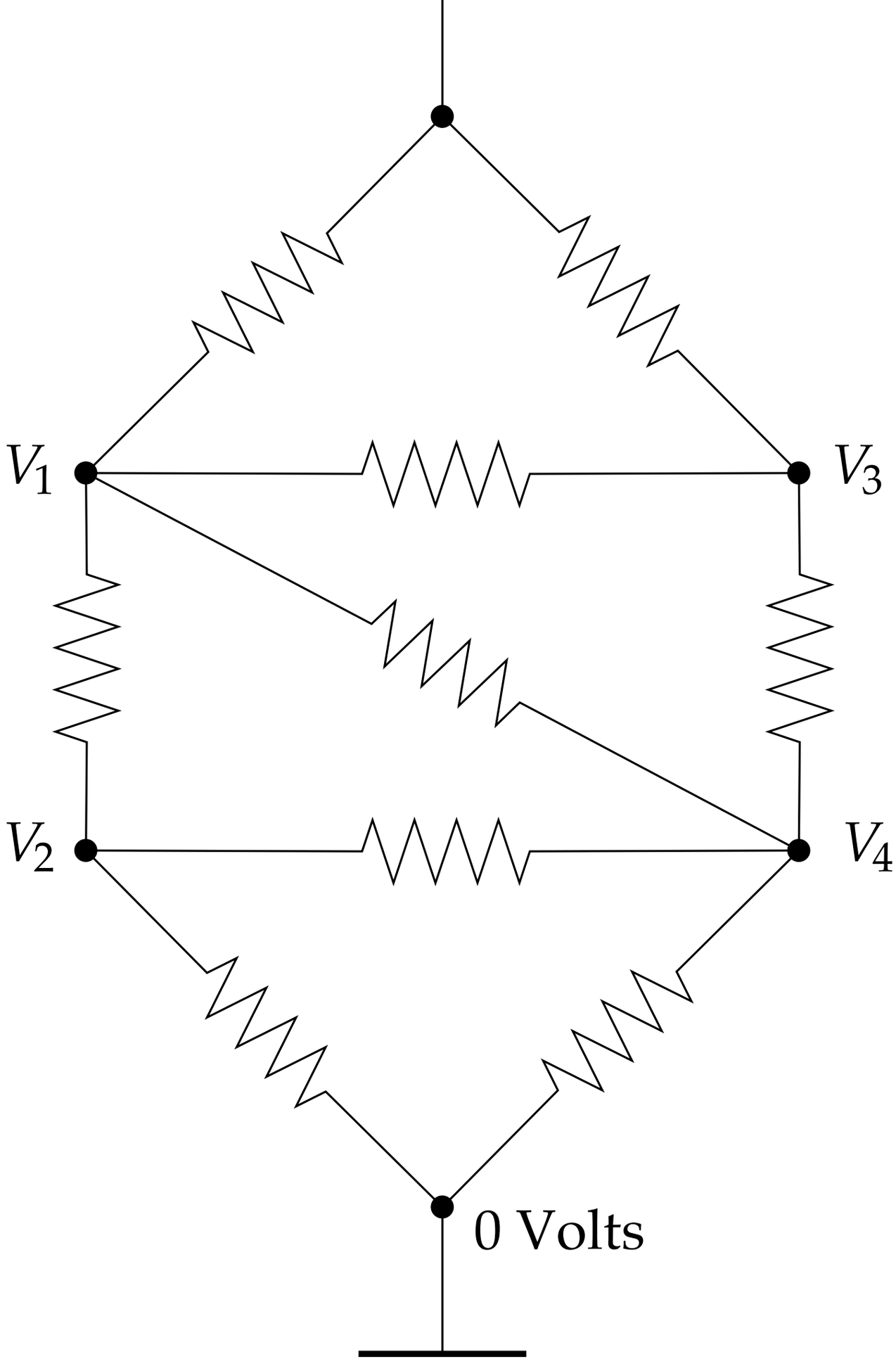
- Once we have a upper triangular matrix we can solve it rather simply by backsubstitution. If our linear equations are the following

$$\begin{bmatrix} 1 & a_{01} & a_{02} & a_{03} \\ 0 & 1 & a_{12} & a_{13} \\ 0 & 0 & 1 & a_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

- we can write them as the following equations:

$$\begin{aligned} w + a_{01}x + a_{02}y + a_{03}z &= v_0 \\ x + a_{12}y + a_{13}z &= v_1 \\ y + a_{23}z &= v_2 \\ z &= v_3 \end{aligned}$$

- from which we can see trivially that $z=v_3$. Then clearly we get $y=v_2-a_{23}v_3$ and then we can substitute in y and z to get x and w.



EXERCISE 6.1

- A circuit of resistors, shown to the left, all have the same resistance R. The voltage at the top is 5V, what are the voltages at the other points 1-4?
- Ohm' and Kirchhoff's Laws can be used to write an equation at each junction, for example at V_1:

$$\frac{V_1 - V_2}{R} + \frac{V_1 - V_3}{R} + \frac{V_1 - V_4}{R} + \frac{V_1 - V_+}{R} = 0$$

$$=> 4V_1 - V_2 - V_3 - V_4 = V_+$$

- write the other 3 equations and then turn them into matrix form and solve with Gaussian elimination.

GAUSSIAN ELIMINATION EXAMPLE CODE

```
import numpy as np

A = np.array([[ 2,  1,  4,  1 ], [ 3,  4, -1, -1 ], [ 1, -4,  1,  5 ], [ 2, -2,  1,  3 ]], dtype=float)
v = np.array([ -4, 3, 9, 7 ],dtype=float)
N = len(v)
# Gaussian elimination
for m in range(N):
    # Divide by the diagonal element
    div = A[m,m]
    A[m,:] /= div
    v[m] /= div
    # Now subtract from the lower rows
    for i in range(m+1,N):
        mult = A[i,m]
        A[i,:] -= mult*A[m,:]
        v[i] -= mult*v[m]

# Backsubstitution
x = np.empty(N,float)
for m in range(N-1,-1,-1):
    x[m] = v[m]
    for i in range(m+1,N):
        x[m] -= A[m,i]*x[i]
print(x)
```

PIVOTING

- What if we have a matrix that looks like:

$$\begin{bmatrix} 0 & 1 & 4 & 1 \\ 3 & 4 & -1 & -1 \\ 1 & -4 & 1 & 5 \\ 2 & -2 & 1 & 3 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -4 \\ 3 \\ 9 \\ 7 \end{bmatrix}$$

- The first element of the first row is zero, which means our Gaussian elimination won't work. The solution to this is actually rather simple, we can just swap the fist and second rows of the matrix and vector

$$\begin{bmatrix} 3 & 4 & -1 & -1 \\ 0 & 1 & 4 & 1 \\ 1 & -4 & 1 & 5 \\ 2 & -2 & 1 & 3 \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3 \\ -4 \\ 9 \\ 7 \end{bmatrix}$$

- and now Gaussian elimination will work. This swap is called pivoting.

LU DECOMPOSITION

- While Gaussian elimination is a perfectly fine method of solving systems of linear equations it is commonly used in a different form in physics calculations.
- That is because in physics we will often want to solve $\mathbf{Ax}=\mathbf{v}$ for the same $\mathbf{A}$ but many different $\mathbf{v}$. In that case it is wasteful to perform the Gaussian elimination each time, the matrix $\mathbf{A}$ will be the same every time.
- Instead we would rather perform the Gaussian elimination just once and then record the effect on the vector $\mathbf{v}$ in a way that allows us to perform it on many different vectors $\mathbf{v}$.
- A process that does this is called LU decomposition.

LU DECOMPOSITION

- Let's start with a matrix

$$A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

- we can write the steps of Gaussian elimination as a matrix

$$\frac{1}{a_{00}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -a_{10} & a_{00} & 0 & 0 \\ -a_{20} & 0 & a_{00} & 0 \\ -a_{30} & 0 & 0 & a_{00} \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{bmatrix}$$

- Let's call the matrix on the left $\mathbf{L_0}$ and notice that it is lower triangular. So we have $\mathbf{L_0\ A = B}$.

LU DECOMPOSITION

- Performing the next step of the Gaussian elimination on B

$$\frac{1}{b_{11}}\begin{bmatrix} b_{11} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -b_{21} & b_{11} & 0 \\ 0 & -b_{31} & 0 & b_{11} \end{bmatrix}\begin{bmatrix} 1 & b_{01} & b_{02} & b_{03} \\ 0 & b_{11} & b_{12} & b_{13} \\ 0 & b_{21} & b_{22} & b_{23} \\ 0 & b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} 1 & c_{01} & c_{02} & c_{03} \\ 0 & 1 & c_{12} & c_{13} \\ 0 & 0 & c_{22} & c_{23} \\ 0 & 0 & c_{32} & c_{33} \end{bmatrix}$$

- we can call this matrix on the left $\mathbf{L_1}$ and similarly define

$$L_2 = \frac{1}{c_{22}}\begin{bmatrix} c_{22} & 0 & 0 & 0 \\ 0 & c_{22} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -c_{32} & c_{22} \end{bmatrix} \qquad L_3 = \frac{1}{d_{33}}\begin{bmatrix} d_{33} & 0 & 0 & 0 \\ 0 & d_{33} & 0 & 0 \\ 0 & 0 & d_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- with these we can write our original equation as

$$L_3 L_2 L_1 L_0 A x = L_3 L_2 L_1 L_0 v$$

LU DECOMPOSITION

- This is the LU decomposition method, which is just the same as Gaussian decomposition, but we express the steps in terms of matrices. However, the difference is now with these matrices we can find the solution for many different vectors **v**.
- We can define two matrices as

$$L = L_0^{-1} L_1^{-1} L_2^{-1} L_3^{-1} \qquad U = L_3 L_2 L_1 L_0 A$$

- **U** is just what we multiply times **v** so it is upper triangular. It turns out L is lower triangular and **LU=A**. So our original equation can be expressed as

$$LUx = v$$

LU DECOMPOSITION

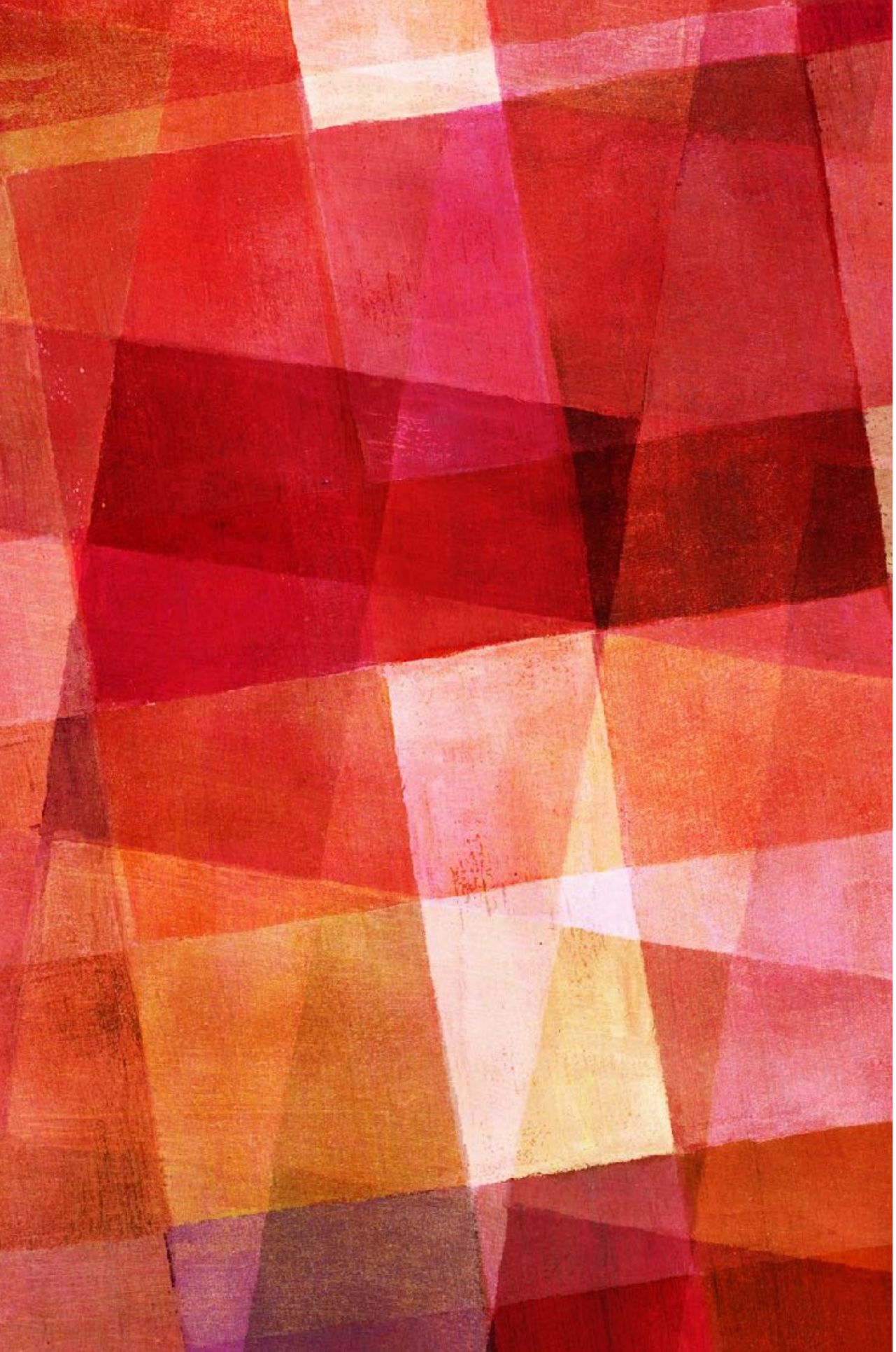
- So the LU decomposition method is to decompose **A** into two matrices which are lower and upper triangular.
- Once we have this decomposition we can solve the equation **Ax=v** for any **v**. Let's look at a 3×3 matrix for example

$$A = \begin{bmatrix} l_{00} & 0 & 0 \\ l_{10} & l_{11} & 0 \\ l_{20} & l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} u_{00} & u_{01} & u_{02} \\ 0 & u_{11} & u_{12} \\ 0 & 0 & u_{22} \end{bmatrix}$$

- if we define a new vector **y** by $\begin{bmatrix} u_{00} & u_{01} & u_{02} \\ 0 & u_{11} & u_{12} \\ 0 & 0 & u_{22} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$
- the **Ax=v** becomes

$$\begin{bmatrix} l_{00} & 0 & 0 \\ l_{10} & l_{11} & 0 \\ l_{20} & l_{21} & l_{22} \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \end{bmatrix}$$

- and we have two triangular equations that can be solved by back substitution.



EXERCISE 6.4

- Solve the resistor problem (6.1) using the numpy.linalg function solve which is an implementation of LU decomposition.
- Check that your answer agrees with your previous answer

INVERSE OF A MATRIX

- We can solve for the inverse of a matrix using the techniques we have already discussed. If we take our matrix equation to be $\mathbf{AX}=\mathbf{I}$ where $\mathbf{I}$ is the identity matrix, then $\mathbf{X} = \mathbf{A}^{-1}$ by definition.
- While previously we have taken x and v to be vectors, the methodology discussed will work just as well if they are matrices.
- In numpy this is coded as the inv function in numpy.linalg. $\mathbf{X}$=inv($\mathbf{A}$) will return the inverse of $\mathbf{A}$ as $\mathbf{X}$.

TRIDIAGONAL AND BANDED MATRICES

- In physics many matrices only have nonzero values near the diagonal. If only plus and minus one from the diagonal these are called tridiagonal matrices, if more than that the matrix is called banded, where the idea is that most elements are still zero.
- Equations with these types of matrices can be solved by the methods we have discussed; however, Gaussian elimination and backsubstition tend to be much more efficient that LU decomposition because of the large number of zeros.
- For a tridiagonal matrix you only have to subtract one row for each column and the resulting backsubstitution is particularly simple. This is important to remember if you are working with these types of matrices. The numpy function solve will probably take much longer than the Gaussian elimination code we have used in class.

EIGENVALUES AND EIGENVECTORS

- Another common matrix problem in physics is to solve for the eigenvalues and eigenvectors of a matrix.
- In physics the matrix is usually symmetric (or Hermitian if complex) so we will focus only on solving symmetric matrices. We are looking for a vector satisfying:

$$Av = \lambda v$$

- For an N×N matrix there are N eigenvectors $v_1 \ldots v_N$. We can consider the N eigenvectors to be columns of one N×N matrix we'll call **V**. Then the equation can be written as

$$AV = VD$$

- where **D** is a diagonal matrix whose elements are the corresponding eigenvalues λ_i.

QR ALGORITHM

- The most common way to solve the eigenvalue problem is to decompose the matrix **A** into an orthogonal matrix **Q** and an upper triangular matrix **R**. So

$$A = Q_1 R_1$$

- multiplying on the left by the transpose of $\mathbf{Q}_1$ (which is also the inverse since it is an orthogonal matrix).

$$Q_1^T A = Q_1^T Q_1 R_1 = R_1$$

- Now lets define a new matrix

$$A_1 = Q_1^T A Q_1$$

- this new matrix can also be decomposed into $A_1 = Q_2 R_2$

QR ALGORITHM

- Now lets define $\mathbf{A_2}$ as

$$A_2 = R_2 Q_2 = Q_2^T A_1 Q_2 = Q_2^T Q_1^T A Q_1 Q_2$$

- we can keep repeating this procedure till we get some $\mathbf{A_k}$ as

$$A_k = (Q_k^T ... Q_1^T) A (Q_1 ... Q_k)$$

- What is the point of this? It can be proven that if you do this procedure enough times the matrix $\mathbf{A_k}$ becomes diagonal. For each iteration the off diagonal elements become smaller and smaller. The matrix will become closer and closer to a diagonal matrix $\mathbf{D}$. Furthermore we can see that

$$V = \prod_{i=1}^{k} Q_i \quad \text{since} \quad D = A_k = V^T A V \quad => AV = DV$$

QR ALGORITHM

- The steps then to implement the algorithm are:

  1. Create an N×N matrix **V** initialized as the identity matrix and choose an accuracy for the off diagonal elements ε.

  2. Calculate the QR decomposition of **A**.

  3. Update A to the new value **A**=**RQ**.

  4. Check the off diagonal elements of **A**, if they are less than ε, then stop, otherwise go back to 2 and repeat.

- This algorithm in implemented in numpy.linalg as eigh. There is also a function eigvalsh if you only want the eigenvalues returned.

- An exercise on how to perform the QR decomposition is given in the book at exercise 6.8.

PYTHON PACKAGES

- Linear algebra methods are part of the numpy.linalg sub package.
- https://docs.scipy.org/doc/numpy/reference/routines.linalg.html