3.016

Lecture 7: Linear Algebra

Reading:
Kreyszig Sections: 13.1, 13.2, 13.5, 13.6 (pages602–606, 607–611, 623–626, 626–629)

3.016 Home

Uniqueness and Existence of Linear System Solutions

It would be useful to use the Mathematica Help Browser and open the link to Matrices & Linear Algebra in the Mathematics & Algorithms section. Look through the tutorials at the bottom on the linked page.

A linear system of m equations in n variables $(x_1, x_2, \ldots, x_n)$ takes the form

$$\begin{aligned} A_{11}x_1 + A_{12}x_2 + A_{13}x_3 + \ldots + A_{1n}x_n &= b_1 \\ A_{21}x_1 + A_{22}x_2 + A_{23}x_3 + \ldots + A_{2n}x_n &= b_2 \\ \vdots &= \vdots \\ A_{k1}x_1 + A_{k2}x_2 + A_{k3}x_3 + \ldots + A_{kn}x_n &= b_k \\ \vdots &= \vdots \\ A_{m1}x_1 + A_{m2}x_2 + A_{m3}x_3 + \ldots + A_{mn}x_n &= b_m \end{aligned} \tag{7-1}$$

Full Screen

Close

and is fundamental to models of many systems.

Quit

The coefficients, A_{ij}, form a matrix and such equations are often written in an "index" short-hand known as the Einstein summation convention:

$$A_{ji}x_i = b_j \qquad \text{(Einstein summation convention)} \tag{7-2}$$

©W. Craig Carter

where *if an index* (here i) *is repeated in any set of multiplied terms,* (here $A_{ji}x_i$) *then a summation over all values of that index is implied.* Note that, by multiplying and summing in Eq. 7-2, the repeated index i disappears from the right-hand-side. It can be said that the repeated index in "contracted" out of the equation and this idea is emphasized by writing Eq. 7-2 as

$$x_i A_{ij} = b_j \qquad \text{(Einstein summation convention)} \tag{7-3}$$

so that the "touching" index, i, is contracted out leaving a matching j-index on each side. In each case, Eqs. 7-2 and 7-3 represent m equations ($j = 1, 2, \ldots, m$) in the n variables ($i = 1, 2, \ldots, n$) that are contracted out in *each equation*. The summation convention is especially useful when the dimensions of the coefficient matrix is larger than two; for a linear elastic material, the elastic energy density can be written as:

$$E_{\text{elast}} = \frac{1}{2}\epsilon_{ij}C_{ijkl}\epsilon_{kl} = \frac{1}{2}\sigma_{pq}S_{pqrs}\sigma_{rs} \tag{7-4}$$

where C_{ijkl} and ϵ_{ij} are the fourth-rank *stiffness* tensor and second-rank elastic *strain* tensor; where S_{ijkl} and σ_{ij} are the fourth-rank compliance tensor and second-rank stress tensor;

In physical problems, the goal is typically to find the n x_i for a given m b_j in Eqs. 7-2, 7-3, or 7-1. This goal is conveniently represented in matrix-vector notation:

$$\underline{A}\vec{x} = \vec{b} \tag{7-5}$$

©W. Craig Carter

3.016

Solving Linear Sets of Equations

Demonstrations of several different ways to solve a set of linear equations for several variables. Here we will solve equations that we construct from matrices; in following examples, we will operate on the matrices directly.

Consider the set of equations

$x + 2y + z + t = a$
$-x + 4y - 2z = b$
$x + 3y + 4z + 5t = c$
$x + z + t = d$

We illustrate how to use a matrix representation to write these out and solve them...

Start with the matrix of coefficients of the variables, ***mymatrix****:*

```
mymatrix = {
   {1, 2, 1, 1},
   {-1, 4, -2, 0},
   {1, 3, 4, 5},
   {1, 0, 1, 1}};
mymatrix // MatrixForm
```
1

The system of equations will only have a unique solution if the determinant of ***mymatrix*** *is nonzero.*

```
Det[mymatrix]
```
2

Now define vectors for $\vec{x}$ and $\vec{b}$ in $\underline{A}\,\vec{x} = \vec{b}$

```
myx = {x, y, z, t};
```
3

```
myb = {a, b, c, d};
```
4

The left-hand side of the first equation will be

```
(mymatrix.myx)[[1]]
```
5

and the left-hand side of all four equations will be

```
lhs = mymatrix.myx;
lhs // MatrixForm
```
6

Now define an indexed variable ***linsys*** *with four entries, each being one of the equations in the system of interest:*

```
linsys[i_Integer] := lhs[[i]] == myb[[i]]
```
7

```
linsys[2]
```
8

Solving the set of equations for the unknowns $\vec{x}$

```
linsol = Solve[{linsys[1],
   linsys[2], linsys[3], linsys[4]}, myx]
```
9

1: This example is kind of backwards. We will take a matrix

$$\underline{A} = \begin{pmatrix} 1 & 2 & 1 & 1 \\ -1 & 4 & -2 & 0 \\ 1 & 2 & 4 & 5 \\ 1 & 0 & 1 & 1 \end{pmatrix} \text{ unknown vector } \vec{x} = \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix} \text{ and known vector } \vec{b} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix}$$

and extract four equations for input to `Solve` to obtain the solution to $\vec{x}$. Here, the coefficient matrix is a list of row-lists.

2: A unique solution will exist if the determinant, computed with `Det`, is non-zero.

3–4: These will be the left-hand- and right-hand-side vectors.

5: Matrix multiplication is indicated by the period (.). This will be the first of the equations.

6: `lhs` is a column-vector with four entries, and each entry is one of the lhs equations.

7–8: This function creates *logical equalities* for each corresponding entry on the left- and right-hand-sides. unknowns.

9: The function `Solve` is used on a system of equations (`{linsys[i]}` and variables.

3.016 Home

Full Screen

Close

Quit

©W. Craig Carter

Inverting Matrices or Just Solving for the Unknown Vector

Continuing the last example, it is much more compact to invert a matrix symbolically or numerically. If a matrix inverse is going to be used over and over again, it is probably best to compute and store the inverse once. However, if a one-time only solution for $\vec{x}$ in $\underline{A}\vec{x} = \vec{b}$ is needed, then computing the inverse is computationally less efficient than using an algorithm designed to solve for $\vec{x}$ directly. Here is an example of both methods.

Doing the same thing a different way, using Mathematica's ***LinearSolve*** *function:*

```
?LinearSolve
```
(1)

LinearSolve[m, b] finds an x which solves th
LinearSolve[m] generates a LinearSolveFun
which can be applied repeatedly to diffe

```
LinearSolve[mymatrix, myb]
```
(2)

And yet another way, based on $\vec{x} = \underline{A}^{-1}\underline{A}\vec{x} = \underline{A}^{-1}\vec{b}$

```
Inverse[mymatrix].myb // MatrixForm
```
(3)

$$\begin{pmatrix} \frac{a}{7}+\frac{b}{7}-\frac{2c}{7}+\frac{9d}{7} \\ \frac{a}{2}-\frac{d}{2} \\ \frac{13a}{14}-\frac{4b}{7}+\frac{c}{7}-\frac{23d}{14} \\ -\frac{15a}{14}+\frac{3b}{7}+\frac{c}{7}+\frac{19d}{14} \end{pmatrix}$$

And yet even another way, a very efficient LinearSolveFunction can be produced by LinearSolve. This function will operate on any rhs vector of the appropriate length. This would be an efficient way to find the numerical solution to a known matrix, but for many different rhs $\vec{b}$.

```
mymatrixsol = LinearSolve[mymatrix];
```
(4)

The result can be applied as a function calling a vector :

```
mymatrixsol[myb]
Simplify[mymatrixsol[myb]]
```
(5)

$$\left\{\frac{1}{7}(a+b-2c+9d),\ \frac{a-d}{2},\ \frac{1}{14}(13a-8b+2c-23d),\ \frac{1}{14}(-15a+6b+2c+19d)\right\}$$

1–2: `LinearSolve` can take two arguments, $\underline{A}$ and $\vec{b}$, and returns $\vec{x}$ that solves $\underline{A}\vec{x} = \vec{b}$. It will be noticibly faster than the following inversion method, especially for large matrices.

3: The matrix inverse is obtained with `Inverse` and a subsequent multiplication by the right-hand-side gives the solution.

4–5: Calling `LinearSolve` on a matrix alone, *returns an efficient function* that takes the unknown vector as an argument. Here we show the equivalence to item 3.

©W. Craig Carter

Uniqueness of solutions to the nonhomogeneous (heterogeneous) system

$$\underline{A}\vec{x} = \vec{b} \tag{7-6}$$

Uniqueness of solutions to the homogeneous system

$$\underline{A}\vec{x_o} = \vec{0} \tag{7-7}$$

Adding solutions from the nonhomogeneous and homogenous systems

You can add any solution to the homogeneous equation (if they exist, there are infinitely many of them) to any solution to the nonhomogeneous equation, and the result is still a solution to the nonhomogeneous equation.

$$\underline{A}(\vec{x} + \vec{x_o}) = \vec{b} \tag{7-8}$$

Determinants

©W. Craig Carter

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Determinants, Rank, and Nullity

Several examples of determinant calculations are provided to illustrate the properties of determinants. When a determinant vanishes (i.e., $\det \underline{A} = 0$), there is no solution to the inhomogeneous equation $\det \underline{A} = \vec{b}$, but there will be an infinity of solutions to $\det \underline{A} = 0$; the infinity of solutions can be characterized by solving for a number *rank* of the entries of $\vec{x}$ in terms of the *nullity* of other entries of $\vec{x}$

Create a matrix with one row as a linear combination of the others

```
myzeromatrix =
  {mymatrix[[1]], mymatrix[[2]],
   p * mymatrix[[1]] +
    q * mymatrix[[2]] + r * mymatrix[[4]],
   mymatrix[[4]]};
myzeromatrix // MatrixForm
```
1

$$\begin{pmatrix} 1 & 2 & 1 & 1 \\ -1 & 4 & -2 & 0 \\ p-q+r & 2p+4q & p-2q+r & p+r \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

```
Det[myzeromatrix]
```
2

```
LinearSolve[myzeromatrix, myb]
```
3

This was not expected to have a solution

```
MatrixRank[mymatrix]
MatrixRank[myzeromatrix]
```
4

```
NullSpace[mymatrix]
NullSpace[myzeromatrix]
```
5

*Try solving this inhomogeneous system of equations using **Solve**:*

```
zerolhs = myzeromatrix.myx
```
6

```
zerolinsys[i_Integer] :=
 zerolhs[[i]] == myb[[i]]
```
7

```
zerolinsolhet =
 Solve[Table[zerolinsys[i], {i, 4}], myx]
```
8

No solution, as expected, Let's solve the homogeneous problem:

```
zerolinsolhom = Solve[Table[zerolinsys[i] /.
    {a → 0, b → 0, c → 0, d → 0}, {i, 4}], myx]
```
9

```
{{y → 0, x → -2 t, z → t}}
```

1: A matrix is created where the third row is the sum of $p\times$first row, $q\times$second row, and $r\times$fourth row. In other words, one row is a linear combination of the others.

2: The determinant is computed with `Det`, and its value should reflect that the rows are not linearly independent.

3: An attempt to solve the linear inhomogeneous equation (here, using `LinearSolve`) should fail.

4: When the determinant is zero, there may still be some linearly-independent rows or columns. The rank gives the number of linearly-independent rows or columns and is computed with `MatrixRank`. Here, we compare the rank of the original matrix and the linearly-dependent one we created.

5: The *null space* of a matrix, $\underline{A}$, is a set of linearly-independent vectors that, if left-multiplied by $\underline{A}$, gives a zero vector. The nullity is how many linearly-independent vectors there are in the null space. Sometimes, vectors in the null space are called *killing vectors*. By comparing to the above, you will see examples of the *rank* + *nullity* = *dimension* rule for square matrices.

6–8: Here, an attempt to use `Solve` for the heterogeneous system with vanishing determinant is attempted, but of course it is bound to fail...

9: However, this is the solution to the singular homogeneous problem ($\underline{A}\vec{x} = \vec{0}$, where $\det \underline{A} = 0$. The solution is three (the rank) dimensional surface embedded in four dimensions (the rank plus the nullity). Notice that the solution is a multiple of the null space that we computed in item 5.

3.016 Home

Full Screen

Close

Quit

©W. Craig Carter

In example 07-1, it was stated (item 2) that a unique solution exists if the matrix's determinant was non-zero. The solution,

$$\vec{x} = \begin{pmatrix} \frac{2a+2b-4c+18d}{\det \underline{A}} \\ \frac{7a-7d}{\det \underline{A}} \\ \frac{13a-8b+2c-23d}{\det \underline{A}} \\ \frac{-15a+6b+2c+19d}{\det \underline{A}} \end{pmatrix} \tag{7-9}$$

indicates why this is the case and also illustrates the role that the determinant plays in the solution. Clearly, if the determinant vanishes, then the solution is undetermined unless $\vec{b}$ is a zero-vector $\vec{0} = (0, 0, 0, 0)$. Considering the *algebraic equation*, $ax = b$, the determinant plays the same role for matrices that the condition $a = 0$ plays for algebra: the inverse exists when $a \neq 0$ or $\det \underline{A} \neq 0$.

The determinant is only defined for square matrices; it derives from the elimination of the n unknown entries in $\vec{x}$ using all n equation (or rows) of

$$\underline{A}\vec{x} = 0 \tag{7-10}$$

For example, eliminating x and y from

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \text{gives the expression}$$

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \equiv a_{11}a_{22} - a_{12}a_{21} = 0 \tag{7-11}$$

and eliminating x, y, and z from

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{7-12}$$

gives the expression

$$\det \underline{A} \equiv a_{11}a_{22}a_{33} - a_{11}a_{32}a_{23} + a_{21}a_{32}a_{13} - a_{21}a_{12}a_{33} + a_{31}a_{12}a_{23} - a_{31}a_{22}a_{13} = 0$$

The following general and true statements about determinants are plausible given the above expressions:

3.016 Home
Full Screen
Close
Quit

©W. Craig Carter

- Each term in the determinant's sum us products of N terms—a term comes from each column.
- Each term is one of all possible the products of an entry from each column.
- There is a plus or minus in front each term in the sum, $(-1)^p$, where p is the number of *neighbor exchanges required to put the rows in order* in each term written as an ordered product of their columns (as in Eqs. 7-11 and 7-12).

These, and the observation that it is impossible to eliminate $\vec{x}$ in Eqs. 7-11 and 7-12 if the information in the rows is redundant (i.e., there is not enough information—or independent equations—to solve for the $\vec{x}$), yield the general properties of determinants that are illustrated in the following example.

©W. Craig Carter

Properties of Determinants and Numerical Approximations to Zero

Rules, corresponding to how det A changes when the columns of $\underline{A}$ are permuted or multiplied by a constant, are demonstrated.

```
rv[i_] :=
 rv[i] = Table[RandomReal[{-1, 1}], {j, 6}]
```
1

*Now use **rv** to make a 6 x 6 matrix, then find its determinant:*

```
RandMat = Table[rv[i], {i, 6}]
```
2

```
Det[RandMat]
```
3

Switching two rows changes the sign but not the magnitude of the determinant:

```
Det[{rv[2], rv[1], rv[3], rv[4], rv[5], rv[6]}]
```
4

Multiply one row by a constant and calculate determinant:

```
Det[{a*rv[2], rv[1],
  rv[3], rv[4], rv[5], rv[6]}]
```
5

Multiply two rows by a constant and calculate determinant:

```
Det[{a*rv[2], a*rv[1],
  rv[3], rv[4], rv[5], rv[6]}]
```
6

Multiply all rows by a constant and calculate determinant:

```
Det[
 a {rv[2], rv[1], rv[3], rv[4], rv[5], rv[6]}]
```
7

```
Clear[a, b, c, d, e]
LinDepVec = a*rv[1] + b*rv[2] +
  c*rv[3] + d*rv[4] + e*rv[5]
```
8

Example of numerical precision: this determinant should evaluate to zero...

```
Det[{rv[1], rv[2],
  rv[3], rv[4], rv[5], LinDepVec}]
```
9

$-4.85723\times10^{-17}\,a + 4.85723\times10^{-17}\,b + 4.16334\times10^{-17}\,c - 4.85723\times10^{-17}\,d - 1.38778\times10^{-17}\,e$

However, numerical precision does

```
Chop[Det[{rv[1], rv[2],
  rv[3], rv[4], rv[5], LinDepVec}]]
```
10

1–2: A matrix, *RandMat* , is created from rows with random real entries between -1 and 1.

3–4: This will demonstrate that switching neighboring rows of a matrix changes the sign of the determinant.

5–6: Multiplying *one* column of a matrix by a constant a, multiplies the matrix's determinant by *one factor of* a; multiplying two rows by a gives a factor of a^2. Multiplying every entry in the matrix by a changes its determinant by a^n.

7: Because the matrix has one linearly-dependent column, its determinant should vanish. This example demonstrates what happens with limited numerical precision operations on real numbers. The determinant is not zero, but could be considered *effectively zero*.

8: We create a row which is an arbitrary linear combination of the first five rows of `RandMat`.

9: This determinant should be zero. However, because the entries are numerical, differences which are smaller than the precision with which a number is stored, may make it difficult to distinguish between something that is *numerically zero* and one that is *precisely zero*. This is sometimes known as *round-off error*.

10: Problems with numerical imprecision can usually be alleviated with `Chop` which sets small magnitude numbers to zero.

3.016 Home

Full Screen

Close

Quit

©W. Craig Carter

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Determinants and the Order of Matrix Multiplication

Symbolic matrices are constructed to show examples of the rules $\det(\underline{AB}) = \det \underline{A} \det \underline{B}$ and $\underline{AB} \neq \underline{BA}$.

Creating a symbolic matrix

```
SymVec = {a, a, a, c, c, c};
```
1

```
Permuts = Permutations[SymVec]
Permuts // Dimensions
```
2

```
SymbMat = {
        Permuts[[1]],
        Permuts[[12]],
        Permuts[[6]],
        Permuts[[18]],
        Permuts[[17]],
        Permuts[[9]]};
SymbMat // MatrixForm
```
3

```
DetSymbMat = Simplify[Det[SymbMat]]
```
4

Creating a matrix of random rational numbers

```
RandomMat =
  Table[Table[RandomInteger[{-100, 100}]/RandomInteger[{1, 100}],
    {i, 6}], {j, 6}];
MatrixForm[RandomMat]
```
5

```
DetRandomMat = Det[RandomMat]
```
6

```
CheckA = Det[SymbMat.RandomMat] // Simplify
```
7

```
DetRandomMat * DetSymbMat == CheckA
```
8

Does the determinant of a product depend on the order of multiplication?

```
CheckB = Det[RandomMat.SymbMat] // Simplify
```
9

```
CheckA == CheckB
```
10

However, the product of two matrices depends on which matrix is on the left and which is on the right

```
(RandomMat.SymbMat - SymbMat.RandomMat) //
  Simplify // MatrixForm
```
11

1–3: Using `Permutations` to create all possible permutations of two sets of three identical objects for subsequent construction of a symbolic matrix, `SymbMat`, for demonstration purposes.

4: The symbolic matrix has a fairly simple determinant—it can only depend on two symbols and must be sixth-order.

5: A matrix with random rational numbers is created. . .

6: And, of course, its determinant is also a rational number.

7–10: This demonstrates that the determinant of a product is the product of determinants and is independent of the order of multiplication. . .

11: However, the result of multiplying two matrices *does* depend on the order of multiplication: $\underline{AB} \neq \underline{BA}$, in general.

Matrix multiplication is *non-commutative*: $\underline{AB} \neq \underline{BA}$ for most matrices. However, any two matrices for which the order of multiplication does not matter ($\underline{AB} = \underline{BA}$) are said to *commute*. Commutation is an important concept in quantum mechanics and crystallography.

Think about what commuting matrices means physically. If two linear transformations commute, then the order in which they are applied doesn't matter. In quantum mechanics, an operation is roughly equivalent to making an observation—commuting operators means that one measurement does not interfere with a commuting measurement. In crystallography, operations are associated with symmetry operations—if two symmetry operations commute, they are, in a sense, "orthogonal operations."

3.016 Home

Full Screen

Close

Quit

©W. Craig Carter

Vector Spaces

Consider the position vector

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{7-13}$$

The vectors $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$ can be used to generate any general position by suitable scalar multiplication and vector addition:

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = x \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + y \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} + z \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{7-14}$$

Thus, three dimensional real space is "spanned" by the three vectors: $(1, 0, 0)$, $(0, 1, 0)$, and $(0, 0, 1)$. These three vectors are candidates as "basis vectors for $\Re^3$."

Consider the vectors $(a, -a, 0)$, $(a, a, 0)$, and $(0, a, a)$ for real $a \neq 0$.

$$\vec{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{x + y}{2a} \begin{pmatrix} a \\ -a \\ 0 \end{pmatrix} + \frac{x - y}{2a} \begin{pmatrix} a \\ a \\ 0 \end{pmatrix} + \frac{x - y + 2z}{2a} \begin{pmatrix} 0 \\ a \\ a \end{pmatrix} \tag{7-15}$$

So $(a, -a, 0)$, $(a, a, 0)$, and $(0, a, a)$ for real $a \neq 0$ also are basis vectors and can be used to span $\Re^3$.

The idea of basis vectors and vector spaces comes up frequently in the mathematics of materials science. They can represent abstract concepts as well as being shown by the following two dimensional basis set:

©W. Craig Carter

basis vector 1 basis vector 2

1.0 + 1.0 =

1.0 + 0.5 =

1.0 + 0.2 =

0.1 + 1.0 =

0.5 + 0.7 =

0.0 + 1.0 =

Figure 7-2: A vector space for two-dimensional CsCl structures. Any combination of center-site concentration and corner-site concentration can be represented by the sum of two basis vectors (or basis lattice). The set of all grey-grey patterns is a vector space of patterns.

Linear Transformations

©W. Craig Carter

Visualization Example: Polyhedra

A simple octagon with different colored faces is transformed by operating on all of its vertices with a matrix. This example demonstrates how symmetry operations, like rotations reflections, can be represented as a matrix multiplication, and how to visualize the results of linear transformations generally.

We now demonstrate the use of matrix multiplication for manipulating an object, specifically an octohedron. The Octahedron is made up of eight polygons and the initial coordinates of the vertices were set to make a regular octahedron with its main diagonals parallel to axes x,y,z. The faces of the octahedron are colored so that rotations and other transformations can be easily tracked.

```
<< "PolyhedronOperations`"
Show[PolyhedronData["Octahedron"]]
```
1

Above, the color of the three dimensional object derives from the colors in the light sources. For example, note that there appears to be a blue light pointing down from the left. The lights stay fixed as we rotate the object. If Lighting → None, then the polyhedron's faces will appear to be black.

```
Show[PolyhedronData["Octahedron"],
 Lighting → None]
```
2

We can extract data from the Octahedron, and build our own with individually colored faces. We will need the individual colors to identify what happens to the polyhedron under linear transformaions.

```
PolyhedronData["Octahedron", "Faces"]
```
3

*The object **ColOct** is defined below to draw an octahedron and it invokes the **Polygon** function to draw the triangular faces by connecting three points at specific numerical coordinates that we obtain from the Octahedron data. Because we will turn off lighting, we will ask that each of the faces glow, using the **Glow** graphics directive*

```
octa = {p[1], p[2], p[3], p[4], p[5], p[6]} =
   PolyhedronData[
     "Octahedron", "Faces"][[1]];
colface[i_] := Glow[Hue[i / 8]] ;
ColOct =
  {{colface[0], Polygon[{p[4], p[5], p[6]}]},
   {colface[1], Polygon[{p[4], p[6], p[2]}]},
   {colface[2], Polygon[{p[4], p[2], p[1]}]},
   {colface[3], Polygon[{p[4], p[1], p[5]}]},
   {colface[4], Polygon[{p[5], p[1], p[3]}]},
   {colface[5], Polygon[{p[5], p[3], p[6]}]},
   {colface[6], Polygon[{p[3], p[1], p[2]}]},
   {colface[7], Polygon[{p[6], p[3], p[2]}]}};
```
4

```
Show[Graphics3D[ColOct], Lighting → None]
```
5

1: The package `PolyhedronOperations` contains `Graphics Objects` and other information such as vertex coordinates of many common polyhedra. This demonstrates how an `Octahedron` can be drawn on the screen. The color of the faces comes from the light sources. For example, there is a blue source behind your left shoulder; as you rotate the object the faces—oriented so that they reflect light from the blue source—will appear blue-ish. The color model and appearance is an advanced topic.

2: Setting `Lighting→None` removes the light sources and the octahedron will appear black. Our objective is to observe the effect of linear transformation on this object. To do this, will will want to identify each of the octahedron's faces by "painting" it.

3: We will build a custom octahedron from the Mm's version using `PolyhedronData`.

4: The data is extracted by grabbing the first part of `PolyhedronData` (i.e., `[[1]]`). We assign the name of the list *octa* , and name its elements `p[i]` in one step.
A function is defined and will be used to call `Glow` and `Hue` for each face. When the face glows and the lighting is off, the face will appear as the "glow color", independent of its orientation.
ColOct is a list of graphics-primitive lists: each element of the list uses the glow directive and then uses the points of the original octahedron to define `Polygons` in three dimensions.

5: We wrap *ColOct* inside `Graphics3D` and use `Show` with lighting off to visualize.

©W. Craig Carter

Linear Transformations: Matrix Operations on Polyhedra

A moderately sophisticated MATHEMATICA® function is defined to help visualize the effect of operating on each point of a polyhedron with a 3 × 3-matrix representing a symmetry operation.

```
transoct[tmat_, description_String] :=
 {ColOct /.
   {Polygon[{a_List, b_List, c_List}] →
     Polygon[{tmat.a, tmat.b, tmat.c}]},
  Text[Style[MatrixForm[tmat]], {0, 0, -.25}],
  Text[Style[description, Darker[Red]],
   {0, 0, .25}, Background → White]}
```
1

```
Show[Graphics3D[
  transoct[{{1, 0, 0}, {0, 1, 0}, {0, 0, -1}},
   "mirror-[001]"]], Lighting → None]
```
2

```
identity = IdentityMatrix[3];
rot90[001] = {{0, -1, 0}, {1, 0, 0}, {0, 0, 1}};
 ref[010] = {{1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
o[1, 1] = transoct[identity, "original"];
o[1, 2] = transoct[rot90[001], "90-[001]"];
o[1, 3] = transoct[ref[010], "m-[010]"];
o[2, 1] = transoct[ref[010].rot90[001],
   "90-[100] then m-[010]"];
o[2, 2] = transoct[rot90[001].ref[010],
   "m-[010] then 90-[100]"];
```
3

```
RotationTransform[Pi, {1, 1, 0}]
```
4

```
o[2, 3] = transoct[(0 1 0; 1 0 0; 0 0 -1), "180-[110]"];
```
5

```
sc[θ_, ϕ_] :=
 3 {Cos[θ] Sin[ϕ], Sin[θ] Sin[ϕ], Cos[ϕ]}
Manipulate[GraphicsGrid[
  Table[Show[Graphics3D[o[i, j]],
    Lighting → None, ViewPoint → sc[θ, ϕ],
    ImageSize → {200, 200},
    PlotRange → {{-1, 1}, {-1, 1}, {-1, 1}}],
   {j, 3}, {i, 2}]], {{θ, 2.1}, 0, 2 π},
 {{ϕ, -1.4}, -π / 2, π / 2}]
```
6

1: This is a moderately sophisticated example of rule usage inside of a function (*transoct*) definition: the pattern matches triangles (`Polygons` with three points) in a graphics primitive; names the points; and then multiplies a matrix by each of the points. The first argument to *transoct* is the matrix which will operate on the points; the second argument is an identifyer that will be used with `Text` to annonate the graphics.

2: This demonstrates the use of *transoct* : we get a rotate-able 3D object with floating text identifying the name of the operation and the matrix that performs the operation.

3: We will build an example that will visualize several symmetry steps simultaneously (say that fast outloud). We define matrices for *identity* , *rot90[001]* , and *ref[010]* , respectively, which leave the polyhedra's points unchanged, rotate counter-clockwise by 90° around the [001]-axis, and reflect through the origin in the direction of the [010]-axis.
We use these matrices to create new octahedra corresponding to combinations of symmetry operations.

4–5: It is not always straightforward to write down the matrix corresponding to an arbitrary symmetry operation. MATHEMATICA® has functions to help find many of them; here, we use `RotationTransform` to find the matrix corresponding to rotation by 180° around the [110]-axis.

6: This will display six of the octahedra with their annotated symmetry operations. `Manipulate` is used to change the viewpoint to someplace on a sphere of radius 3 (by changing the latitude angle, ϕ, and the longitude θ). A function to return a cartesian representation of the spherical coordinates is defined first and is used as the `ViewPoint` for each `Graphics3D`-object. `Table` iterates over the `o[i,j]` and passes its result to `GraphicsGrid`.

©W. Craig Carter

notebook (non-evaluated) pdf (evaluated, color) pdf (evaluated, b&w) html (evaluated)

Visualization Example: Invariant Symmetry Operations on Crystals

Each crystal's unit cell can be uniquely characterized by the symmetry operations (i.e., fixed rotation about an axis, reflection across a plane, and inversion through the origin) which leave the unit cell unchanged. The set of such symmetry operations define the *crystal point group*. There are only 32 point groups in three dimensions. In this example, we demonstrate invariant operations for an FCC cell.

```
corners = Flatten[Table[{i, j, k},
   {i, 0, 1}, {j, 0, 1}, {k, 0, 1}], 2]
faces = Join[Permutations[{0.5, 0.5, 0}],
  Permutations[{0.5, 0.5, 1}]]
fccsites = Join[faces, corners]
srad = √2 / 4;
FCC = Table[
  Sphere[fccsites[[i]], srad], {i, 1, 14}]
axes = { {RGBColor[1, 0, 0, .5],
    Cylinder[{{0, 0, 0}, {2, 0, 0}}, .05]},
   {RGBColor[0, 1, 0, .5],
    Cylinder[{{0, 0, 0}, {0, 2, 0}}, .05]},
   {RGBColor[0, 0, 1, .5],
    Cylinder[{{0, 0, 0}, {0, 0, 2}}, .05]}};
fccmodel = Translate[Join[FCC, axes],
  {-.5, -.5, -.5}]
Graphics3D[fccmodel]
```

1

```
bbox = 1.25 {{-1, 1}, {-1, 1}, {-1, 1}};
Manipulate[Grid[{{"original",
    "2π/3-[111]", "roto-inversion: 3̄"},
   {Graphics3D[fccmodel, PlotRange → bbox,
     ViewPoint → sc[θ, ϕ]],
    Graphics3D[Rotate[fccmodel, 2 π / 3,
      {1, 1, 1}], PlotRange → bbox,
     ViewPoint → sc[θ, ϕ]], Graphics3D[
     Rotate[GeometricTransformation[
       fccmodel, -IdentityMatrix[3]],
      2 π / 3, {1, 1, 1}], PlotRange → bbox,
     ViewPoint → sc[θ, ϕ]]}}],
 {{θ, 2.2}, 0, 2 π}, {{ϕ, -.6},
  -π / 2, π / 2}]
```

2

1: The first two commands define *faces* and *corners* which are the coordinates of the face-centered and corner lattice-sites. Note the use of `Flatten` in *corners* has the qualifier 2—it limits the scope of `Flatten` which would normally turn a list of lists into a (flat) single list. `Join` is used to collect the two coordinate lists together into *fccsites* . The atoms will be visualized with the `Sphere` graphics primitive and we use *srad* to set the radius of a close-packed FCC structure. *FCC* is a list of (a list of) graphics primitives for each of the fourteen spheres, and then three cylinders with `Opacity` and color are used to define the coordinate axes graphics: *axes* .
fccmodel is created by joining the spheres and the cylinders, and then using `Translate` on the resulting graphics primitives to put the center of the FCC cell at the origin.

2: `Translate` is an example of a function that operates directly on graphics primitives. We use related functions that also operate on graphics primitives, `Rotate` and `GeometricTransformation`, to illustrate how rotation by 120° about [111], and how inversion (multiplication by "minus the identity matrix") followed by the same rotation, are invariant symmetry operations for the FCC lattice.

3.016 Home

Full Screen

Close

Quit

©W. Craig Carter

Index

- ., 78
- annotation
  - example
    - in three-dimensional graphics, 89
- *axes*, 90
- basis vectors, 86
- `Chop`, 84
- coefficient matrix
  - form in Mathematica, 78
- *ColOct*, 88
- commutation
  - physical interpretation, 85
- commute, 85
- commuting matrices
  - physical interpretation, 85
- compliance tensor, 77
- computational efficiency
  - linear systems of equations, 79
- *corners*, 90
- crystal point group, 90
- `Det`, 78, 81
- determinants, 78
  - properties of, 82
- Einstein summation convention, 76
- elastic energy density, 77
- Example function
  - ColOct, 88
  - FCC, 90
  - RandMat, 84
  - axes, 90
  - corners, 90
  - faces, 90
  - fccmodel, 90
  - fccsites, 90
  - identity, 89
  - octa, 88
  - ref[010], 89
  - rot90[001], 89
  - srad, 90
  - transoct, 89
- *faces*, 90
- *FCC*, 90
- FCC crystal
  - visualization of, 90
- *fccmodel*, 90
- *fccsites*, 90
- `Flatten`, 90
- `GeometricTransformation`, 90
- `Glow`, 88
- graphics
  - reflection from default light sources, 88
- `Graphics3D`, 88, 89
- `GraphicsGrid`, 89

©W. Craig Carter

- Hue, 88

- *identity*, 89
- Inverse, 79

- Join, 90

- killing vectors, 81

- light sources, 88
- Lighting, 88
- linear equations
  - adding homogeneous solutions to the nonhomogeneous solutions, 80
  - existence of solutions, 76
- linear system of equations, 76
- linear systems of equations
  - computational efficiency, 79
- linear transformations, 87
  - visualization examples, 88
- linear vector spaces, 86
- LinearSolve, 79, 81
- logical equalities, 78

- Manipulate, 89
- Mathematica function
  - ., 78
  - Chop, 84
  - Det, 78, 81
  - Flatten, 90
  - GeometricTransformation, 90
  - Glow, 88
  - Graphics3D, 88, 89
  - GraphicsGrid, 89
  - Hue, 88
  - Inverse, 79
  - Join, 90
  - Lighting, 88
  - LinearSolve, 79, 81
  - Manipulate, 89
  - MatrixRank, 81
  - Octahedron, 88
  - Opacity, 90
  - Permutations, 85
  - Polygons, 88
  - Polygon, 89
  - PolyhedronData, 88
  - Rotate, 90
  - RotationTransform, 89
  - Show, 88
  - Solve, 78, 81
  - Sphere, 90
  - Table, 89
  - Text, 89
  - Translate, 90
  - ViewPoint, 89
- Mathematica package
  - PolyhedronOperations, 88
- MatrixRank, 81

- non-commutative, 85
- null space, 81
- nullity, 81
- numerical approximation to zero, 84
- numerical precision

©W. Craig Carter

demonstration of effects, 84

octa, 88
Octahedron, 88
Opacity, 90

Permutations, 85
Polygon, 89
Polygons, 88
PolyhedronData, 88
PolyhedronOperations, 88

RandMat, 84
random rational numbers
matrix of, 85
random real matrix
example, 84
rank, 81
ref[010], 89
rot90[001], 89
Rotate, 90
RotationTransform, 89
round-off error, 84
rules
example of usage to transform polyhedra, 89

Show, 88
solution to the singular homogeneous linear equation, 81
Solve, 78, 81
spanning set of vectors, 86
Sphere, 90
srad, 90
stiffness tensor, 77
strain, 77
stress, 77
summation convention
Einstein, 76
symmetry operations
visualization of, 89

Table, 89
Text, 89
three-dimensional graphics
adding text, example, 89
Translate, 90
transoct, 89

uniqueness of solutions for nonhomogeneous system of equations, 80

ViewPoint, 89
visualization of linear transformations, 88

3.016 Home

Full Screen

Close

Quit

©W. Craig Carter