# **Change of basis.**

Consider a vector ***a*** = [a1 a2 a3]T in **R3** with *coordinates* *a1* = 3, *a2* = 2, *a3* = 1 in standard *Cartesian* basis ***v1****,* ***v2****,* ***v3***. Since our choice of basis is arbitrary, and all that is required for the three basis vector is to be *independent*, we can represent ***a*** using a different basis, for example ***u1****,* ***u2****,* ***u3***. These new vectors do not need to be orthogonal to each other, nor of unit length: the only requirement is again that they be independent (no one of the 3 basis vectors can be obtained as a linear combination of the other 2).

We start by defining ***a*** in terms of the original Cartesian basis ***v1****,* ***v2****,* ***v3*** of unit vectors:

which shows clearly that ***av*** is a *linear combination* of the three basis vectors. In order to obtain the same result in matrix notation we can assemble the 3 *basis vectors* in a single matrix, the *identity* matrix ***I***:

a = [3;2;1]

I = eye(3)

a\_v = I\*a

Then, we define a new basis as the three vectors:

***u1*** = , ***u2*** = , ***u3*** =

u1 = [2;3;1], u2 = [0;1;1], u3 = [2 1 1]'

U = [2 0 2;3 1 1;1 1 1]

U = [u1 u2 u3]

Clearly the coordinates of the three vectors ***u1****,* ***u2****,* ***u3*** are defined in terms of the original ***v1****,* ***v2****,* ***v3*** Cartesian basis ***V*** *=* ***I***. Notice how ***u1****,* ***u2****,* ***u3*** represent a legitimate basis for **R3** because they are independent from each other. Our task is now to find a new representation of ***a****,* ***au*** in this ***U*** basis: this means we want to find a *linear combination* of ***u1****,* ***u2****,* ***u3*** that gives the original vector ***av***. In other words we want to find 3 coefficients *a1', a2', a3'* such that:

***u1*** ***u2 u3 av***

The three coefficients *a1', a2', a3'* are going to be the *coordinates* of ***a*** in the ***u1****,* ***u2****,* ***u3*** basis. This equation, which shows ***av*** as a linear combination of the columns of ***U****,* can be expressed

in matrix notation as:

***U au*** = ***av***

it has the general form: ***A x*** = ***b***

where ***A*** (matrix) and ***b*** (vector) are known and ***x*** (vector) is unknown. We recognize that the matrix notation is the representation of a system of 3 linear equations in 3 unknowns:

which also shows the traditional way to carry out *matrix multiplication* as separate *dot* products. We have learned in high school how to solve such a system. Essentially we are going to convert one of the 3 equations into an equation with only 1 unknown and back-substitute the value found for the 1st unknown in the other equations. The process is shown below step by step:

⇓ remove 1.5 of 1st eq. from 2nd equation

⇓ remove 0.5 of 1st eq. from 3rd equation

⇓ remove 1 of 2nd eq. from 3rd equation

*upper triangular matrix* ⇒ back substitution ⇓

back substitution ⇓

This iteration is known as *Gaussian elimination*. It's based on converting ***U*** into an *upper* (or *lower*) *triangular matrix*. Thus, the coordinates of ***a*** in the ***u1****,* ***u2****,* ***u3*** basis are:

. In fact:

***U*** basis ***au*** = ***a*** in ***av*** = ***a*** in

***U*** basis ***V*** basis (***I*** matrix)

a\_u = [0.5 -0.5 1]'

U\*a\_u

An important corollary of this result is that by *multiplying a vector by its own basis* we always obtain the *representation of the vector in the Cartesian orthogonal basis* (the *identity* matrix). Based on this result, one question comes immediately to mind:

*since there is a matrix (the* ***U*** *basis) that by multiplying* ***au*** *(the* ***a*** *vector in* ***U*** *basis) gives* ***av*** *(the* ***a*** *vector in the Cartesian* ***V*** *basis), is there a matrix, which we could call the ‘****inverse****’ of* ***U****, that performs the inverse operation?*

This is the operation that would give me *directly* ***au*** (the ***a*** vector in ***U*** basis), if I know ***U*** and ***av* (**the ***a*** vector in standard Cartesian ***V*** basis).

If we push Gaussian elimination to convert ***U*** to the identity matrix ***I*** (a process called *Gauss-Jordan elimination*), the same steps of elimination carried out on ***I*** yield the *inverse* of ***U***. In order to do this we first create an *augmented* matrix concatenating ***U*** and ***I***:

UI = [U I] ***U I***

UI(2,:) = UI(2,:)-1.5\*UI(1,:) ⇓ remove 1.5 of 1st row from 2nd row

UI(3,:) = UI(3,:)-0.5\*UI(1,:) ⇓ remove 0.5 of 1st row from 3rd row

UI(3,:) = UI(3,:)-1\*UI(2,:) ⇓ remove 1 of 2nd row from 3rd row

UI(1,:) = UI(1,:)-1\*UI(3,:) ⇓ remove 1 of 3rd row from 1st row

UI(2,:) = UI(2,:)+1\*UI(3,:) ⇓ remove -1 of 3rd row from 2nd row

UI([1 3],:) = UI([1 3],:)/2 ⇓ divide 1st and 3rd row by 2

***I = U/U I/U = 1/U = U-1***

This is the *row reduced echelon form* of the original augmented matrix, which we can also derive directly using MATLAB function *rref*. We can easily show that the component of the augmented matrix derived from the original ***I*** matrix is the *inverse* of ***U***. In fact:

UI = rref([U I])

invU = UI(:,4:6) , invU = UI(:,[4 5 6]), invU = inv(U), invU = U^-1

U\*invU , invU\*U

***U U-1 I******U-1 U***

and also:

***U-1 a*** in ***a*** in

***V*** basis ***U*** basis

a\_u = invU\*a\_v , a\_v = U\*a\_u

*Conclusion*: to represent ***av*** in the ***U*** basis (that is, to obtain ***au*** from ***av***)we *left* multiply ***av*** by the inverse of the ***U*** basis. This is the 1st of three very important operations we want to remember:

**CHANGE OF BASIS *U-1av*= *au***

since ***Uau*** = ***U(U-1av****)* = ***Iav* = *av*** it follows that: ***Uau* = *av***

This means that to *move* a vector from ***V***space to ***U*** space we *left multiply* that vector by the *inverse* of the ***U*** space basis, and to bring it back into ***V*** space we *left multiply* it by the ***U*** basis.

We will use the following 3-step strategy:

**1.** move from one space into another.

**2.** carry out a given operation in that space.

**3.** bring the result back in the original space.

in many important applications in computational biochemistry.

When we showed an example of Gaussian elimination we mentioned that this method is used to solve a system of linear equations of the general form ***Ax*** *=* ***b***. In particular, if ***b*** is a vector of 0s the system of equations is called *homogeneous*. In general, the solution can be found by multiplying both sides of the matrix equation by the inverse of ***A****,* ***A-1***:

However, calculation of the inverse requires *additional steps* that come after the solution of the system of equations has already been found by Gaussian elimination. Thus, although algebraically correct, there is almost never a need to calculate the inverse. In MATLAB, the solution of a system of linear equations is found directly by some variations of Gaussian elimination using the *mldivide* function: this function is usually called directly using the *backslash* operator:

a\_u = inv(U)\*a\_v

a\_u = mldivide(U,a\_v)

a\_u = U\a\_v

MATLAB has a second function, *linsolve*, which is faster than *mldivide* for very large systems of equations, but more difficult to use. In our very simple case, the result provided by the two function is identical, but *mldivide* is faster. In our course, we will typically use *mldivide*.

tic, a\_u = inv(U)\*a\_v, toc

tic, a\_u = mldivide(U,a\_v), toc

tic, a\_u = linsolve(U,a\_v), toc

It is important to remember that in all cases of *change of basis* the vector(s) (or any object to which those vectors are associated) **do not move**: it is the frame of reference (the *basis*) that changes. We see this clearly in a special case of *change of basis* that occurs when the new basis ***U*** is still *Cartesian* (basis vectors orthogonal to each other), but is *rotated* with respect to the original basis ***V***: this type of change of basis is called a *coordinate transformation*.

Very often we want to know directly what is ***au*** (***av*** in the new rotated basis ***U***), if we know the *angle(s)* of the rotation that is carried out on ***V*** to change it to ***U***. The situation is unchanged with respect to the case we just analyzed; the general rule remains:

***au = U-1av***

the only difference is that now we will calculate the matrix ***U-1*** from information about the rotation of the basis ***V***. The matrix ***U-1****,* which represents the inverse of some rotation of the original frame ***V***, is called a *transformation matrix*, and is traditionally represented with the letter ***Q***. A general method for formulating transformation matrices is based on the cosines of the angles between the axes of the two coordinate systems (***U*** and ***V***), i.e., the *direction cosines*. In the case of a 2-D basis it is easy to visualize one possible rotation by an angle θ.



The corresponding transformation matrix can be written as:

where the notation means the angle between and , and the ***au*** vector is obtained as . For example, if we were to apply a *clockwise* rotation of 30° (π/6) to the frame (imagining a 3rd ***v3*** axis of a righ-handed frame coming toward us, and observing the rotation from the origin looking outward along ***v3***), a 2-D vector ***av*** = [3 2]' would change according to:

a\_v = [3 2]'

theta = pi/6

Q = [cos(theta) sin(theta);-sin(theta) cos(theta)]

a\_u = Q\*a\_v

Since **,** based on the formulas for *change of basis*, the *transformation matrix* ***Q***is the *inverse* of the ***U*** basis.

which makes clear the geometric meaning of the *columns* of the transformation matrix ***Q*** as the *coordinates* ofthe ***V*** basisin the ***U*** basis.

Since ***Q*** *=* ***U-1*** it is easy to find the basis vectors of the rotated basis: they are given by the column vectors of the inverse of the *transformation* matrix: ***U*** *=* ***Q-1***.

U = inv(Q)

The general definition of **Q**, in 3-D, using *direction cosines* is:

or the equivalent:

which again shows the *columns* of ***Q*** as the *coordinates* of ***V*** in the ***U*** basis.

Representation of the Cartesian ***V*** basis in ***U*** basis does not change the length of its component vectors (all unit length), nor does it change the angle (90º) between them. Consider a typical transformation matrix ***Q*** in **R3**.

a = [3 5 1], b = [2 0 -6], c = cross(a,b)

u1 = a/norm(a), u2 = b/norm(b), u3 = c/norm(c)

I = eye(3); U = [u1;u2;u3]', U'\*U, U\*U'

Q = inv(U), Q'\*Q, Q\*Q'

While transformationmatricesin 3 dimensions are traditionally represented (particularly in the engineering literature) with the symbol ***Q***, transformation matrices are only a subset of all *orthogonal matrices* in ***n*** dimensions, which are always represented with the symbol ***Q***. Orthogonal matrices ***Q*** have important properties summarized in the following inset:

**1. They are *orthonormal***:

- all column vectors are unit vectors and orthogonal to each other

- all row vectors are unit vectors and orthogonal to each other

norm(Q(:,1)),norm(Q(:,2)),norm(Q(:,3)),norm(Q(1,:)),norm(Q(2,:)),norm(Q(3,:))

Q(:,1)' \*Q(:,2) , Q(:,1)' \*Q(:,3) , Q(:,2)' \*Q(:,3)

Q(1,:)\*Q(2,:)' , Q(1,:)\*Q(3,:)' , Q(2,:)\*Q(3,:)'

Q(1,:)\*Q(1,:)' , Q(2,:)\*Q(2,:)' , Q(3,:)\*Q(3,:)'

or simply

Q'\*Q, Q\*Q'

**2. They preserve the dot product of vectors**

- unitary transformation

a = [3 2 1]';b = [1 2 3]';

a'\*b

(Q\*a)'\*(Q\*b)

**3. The *transpose* of Q (every *column* becomes a *row*) is the same as the *inverse* of Q:**

Q' , inv(Q)

Q'\*Q, Q\*Q'

**4. The *determinant* of Q is ±1:**

det(Q)

**Similarity transformations.**

When a matrix ***T*** multiplies a vector ***a*** to produce another vector ***b***, it acts as a *function* or, as we say using the language of linear algebra, as a *matrix operator*:

***b*** = *f(****a****)* = ***Ta***.

We now ask the question: if we have a *matrix operator* ***Tv*** that carries out the operation:

***Tvav*** *=* ***bv***

in **V** space,

given ***au*** and ***bu*** as the representations of ***a*** and ***b*** in **U**space, what is the *matrix operator* ***Tu*** that carries out the equivalent operation:

***Tuau*** *=* ***bu***

in **U** space?

We know the following:

***Tvav*** *=* ***bv U-1av*** *=* ***au*** ***U-1bv*** *=* ***bu***

***Uau*** *=* ***av Ubu*** *=* ***bv***

We start from the first equation ***Tvav*** *=* ***bv*** , and observing that **(*UU-1*)**is the identity matrix ***I***, we can rewrite it as:

**(*UU-1*) *Tv* (*UU-1*) *av*** *=* ***bv***

and rearrange the parentheses:

***U*(*U-1TvU*)*U-1av*** *=* ***bv***

using ***U-1av*** *=* ***au*** we obtain:

***U* (*U-1TvU*)*au*** *=* ***bv***

and since ***Ubu*** *=* ***bv***, it follows that:

***U* (*U-1TvU*)*au*** *=* ***Ubu*** *=* ***bv***

Therefore:

**(*U-1TvU*) *au*** *=* ***bu***

and since ***Tuau*** *=* ***bu*** , it follows that:

**(*U-1TvU*) *au*** *=***(*Tu*) *au*** *=* ***bu******⇒ Tu*** *=* ***U-1TvU***

In this expression ***U*** can be any '*invertible*' matrix representing the basis of the **U** space.

***Tv*** and ***Tu*** are '**similar**' matrices in **V** and **U** space respectively, and the operation:

***U Tu U-1av*** *=* ***bv*** is called a'**similarity transformation**' of ***av*** to ***bv***.

| | |

| | |⎯⎯⎯⎯⎯⎯⇒ change of basis from **V** to **U** space

| |

| |⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⇒ carry out the operation ***a***  → ***b*** in **U** space

|

|⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⎯⇒ go back to the original **V** space

This operation exemplifies the 3-step strategy we already mentioned in the previous chapter:

**1.** move from one space into another.

**2.** carry out a given operation in that space.

**3.** bring the result back in the original space.

For example consider the vector ***a*** = [3 6] in the standard space **R2** that has as basis the identity matrix . We can rotate this vector (not the frame!) counter clockwise (looking out from the origin towards an imaginary ***z*** axis) by 30 degrees (π/6) by multiplying itby the *rotation* matrix (see SPECIAL TOPIC: intrinsic and extrinsic rotations):

***Rv*** = :

***Rv  a = b***

a = [3 6]';

R\_v = [cos(pi/6) sin(pi/6);-sin(pi/6) cos(pi/6)]; b = R\_v\*a

show\_frame('new',eye(3)); hold on

show\_vector([a;0],'magenta','a'); show\_vector([b;0],'yellow','b')

We could do the same operation in the **U** space that has as basis . First, we calculate the inverse of as . Then, we calculate the matrix ***Ru*,** which is*similar* to ***Rv***, but operates in **U** space.

***U-1 Rv URu***

Finally we carry out the *similarity transformation*:

***U Ru U-1av*** ***bv***

***au***

***bu***

U = [5 3;1 9]

R\_u = inv(U)\*R\_v\*U; R\_u = U\R\_v\*U % using the 'backslash' operator

b = U\*R\_u\*inv(U)\*a; b = U\*R\_u/U\*a % using the 'backslash' operator

The operation of similarity transformation becomes of extreme importance in those cases in which a linear transformation can be carried out more conveniently in a different space. We will encounter some very important biochemical applications of this operation.

**SPECIAL TOPIC: Intrinsic and extrinsic rotations.**

Rotation of a frame with respect to its original orientation by means of a *transformation matrix* ***Q*** is defined as an *intrinsic rotation*. We have seen that the general definition of a *transformation matrix* ***Q***, in 3-D, using *direction cosines* is:

or the equivalent:

which again shows the *columns* of ***Q*** as the *coordinates* of ***V*** in the ***U*** basis.

Since: and:

***Q*** = ***U-1*** = ***UT***  it follows that the rows of ***Q*** are the *coordinates* of ***U*** in the ***V*** basis:

We can visually check that both the columns and rows of ***Q*** represent a right-handed frame:

show\_frame('new',Q); show\_frame('new',Q')

In the case of 3-D rotations it is often convenient to split them as consecutive rotations around one of the principal axes. Clockwise rotations by θ around the ***v1*** (or ***x***), ***v2*** (or ***y***), ***v3*** (or ***z***) axis, looking outward from the origin along each axis of a right-handed frame, would be:

*intrinsic* rotation around x *intrinsic* rotation around y *intrinsic* rotation around z

Notice how the 2nd rotation around ***v2*** has the sign of the two sines transposed because in a right handed frame a clockwise rotation around ***z*** would bring the ***x*** axis onto the ***y*** axis, a clockwise rotation around ***x*** would bring the ***y*** axis onto the ***z*** axis, but a clockwise rotation around ***y*** would bring the ***z*** axis onto the ***x*** axis (and not the ***x*** axis onto the ***z*** axis).

Consecutive rotations of the frame around ***v1****,* ***v2****,* ***v3*** can be represented as a single *transformation* matrix ***Q*** that is the product of three independent *transformation* matrices. The key thing to understand is that each *successive* transformation matrix acts on a frame that has been rotated by the *previous* transformation matrix. Furthermore, the three elemental rotations can use three different *Euler* (from Leonhard Euler) angles defined as follows according to the modern *Tait-Bryan* convention:

* *α* (or ϕ) represents a rotation around the *x* axis leading to new y', z' axes
* *β* (or θ) represents a rotation around the *y'* axis leading to new x' and z'' axes
* *γ* (or ψ) represents a rotation around the *z''* axis.

a\_v = [3 2 1]'

phi = pi/6;theta = pi/3;psi = pi/10;

Q1 = [1 0 0; 0 cos(phi) sin(phi);0 -sin(phi) cos(phi)]

Q2 = [cos(theta) 0 -sin(theta);0 1 0;…

sin(theta) 0 cos(theta)]

Q3 = [cos(psi) sin(psi) 0;-sin(psi) cos(psi) 0;0 0 1]

a\_u = Q3\*Q2\*Q1\* a\_v

Q = Q3\*Q2\*Q1

a\_u = Q\*a\_v

These are also often referred to as *nautical*, or *cardan* (from Girolamo Cardano), or *yaw-pitch-roll* angles from the principal axes that describe the position of an airplane, a boat, or a car in space.

In many books you may still find the *traditional* definition of *Euler* angles as:

*α* (or ϕ) represents a rotation around the *z* axis leading to new x', y' axes

*β* (or θ) represents a rotation around the *x'* axis leading to new y'' and z' axes

*γ* (or ψ) represents a rotation around the *z'* axis.

**Rotation Matrices**

We have seen how any orientation can be achieved by composing three elemental rotations. These rotations can occur around the axes of a rotating frame, which is initially aligned with the fixed frame, and modifies its orientation after each rotation. These types of rotations are called intrinsic rotations and the associated matrices are called transformation matrices (**Q**). Sometimes it is convenient to attach a rotated frame to a rigid body, in which case, it is often called a *local* *coordinate system*. Alternatively, the rotations can occur about the axes of the fixed frame, in which case the rotations are called extrinsic rotations and the corresponding matrices are called rotation matrices (***R***). Thus, if we want to rotate a vector in the current space rather than representing it in a rotated space we multiply the vector by a *rotation* matrix ***R***. A *rotation* matrix is the *transpose* of the corresponding *transformation* matrix: for example, if we want to rotate a vector around the ***v3*** axis clockwise looking outward along the axis from the origin we will use a *rotation* matrix ***R*** that is the transpose of the *transformation* matrix ***Q*** that rotates the frame clockwise looking outward along the axis from the origin. The ***R*** matrices for rotations of vectors (or objects to which the vectors are attached) around the three axes of the standard Cartesian frame corresponding to the ***Q*** matrices for rotation of basis vectors in the 'same' direction are:

and splitting a 3-D rotations into consecutive clockwise rotations by θ around the ***v1*** (or ***x***), ***v2*** (or ***y***), ***v3*** (or ***z***) axis, looking outward from the origin along each axis of a right handed frame:

*extrinsic* rotation around x *extrinsic* rotation around y *extrinsic* rotation around z

It is very important to remember that since *rotation matrices* are the transpose of the corresponding *transformation matrices*, based on the transposition rule for products they are applied in reverse order when pre-multiplying a column vector. Thus, in general:

while:

a\_v = [3 2 1]'

phi = 30; theta = 60; psi = 18;

Q1 = [1 0 0; 0 cosd(phi) sind(phi);0 -sind(phi) cosd(phi)];

Q2 = [cosd(theta) 0 -sind(theta);0 1 0;sind(theta) 0 cosd(theta)];

Q3 = [cosd(psi) sind(psi) 0;-sind(psi) cosd(psi) 0;0 0 1];

Q = Q3\*Q2\*Q1;

R = Q';

R = Q1'\*Q2'\*Q3'; % transposition rule

R1 = Q1'; R2 = Q2'; R3 = Q3';

R = R1\*R2\*R3;

rot\_a\_v = R1\*R2\*R3\*a\_v

rot\_a\_v = R\*a\_v

However, because of the same transposition rule, when post-multiplying a row vector their individual transposes are used in standard order:

rot\_a\_v\_t = (R\*a\_v)'

rot\_a\_v\_t = a\_v'\*R3'\*R2'\*R1'

rot\_a\_v\_t = a\_v'\*Q3\*Q2\*Q1

which reveals that the *extrinsic* rotation of a vector in the same direction as the *intrinsic* rotation of the reference frame can also be obtained by *post-multiplying* the row vector by the same transformation matrix ***Q***. By the same token, the *intrinsic* rotation of the reference frame by the transformation matrix ***Q*** pre-multiplying a vector can be achieved also post-multiplying the vector by ***R = QT***:

a\_u = Q3\*Q2\*Q1\*a\_v

a\_u\_t = a\_v'\*R1\*R2\*R3

Clearly, rotationmatrices ***R*** have all the properties of orthogonal matrices ***Q***.

**Reflection matrices**

if ϕ= pi/2 then the first matrix represents a rotation by 90 degrees of a vector ***b***, while the second matrix produces a reflection of the vector about the 45 degrees (ϕ/2) line.

phi = pi/2

R\_rot = [cos(phi) -sin(phi);sin(phi) cos(phi)]

R\_ref = [cos(phi) sin(phi);sin(phi) -cos(phi)]

b = [1 0.5]'

b = b/norm(b)

b\_rot = R\_rot\*b

b\_ref = R\_ref\*b

show\_frame('new',eye(3));hold on

show\_vector([b;0],'k','b')

show\_vector([b\_rot;0],'m','b\\_rot')

show\_vector([b\_ref;0],'y','b\\_ref')

det(R\_ref)

Reflectionmatrices producewhat is defined as an*improper rotation*: they are a special subset of all orthogonal matrices ***Q*** as theycorrespond to a *left-handed basis***,** and their determinant is always -1.

**PRACTICE**

**1.** Use Gaussian elimination (function gauss\_elim\_step\_by\_step.m) to solve the matrix equation Ax = b, where:

A = [3 7 8;1 4 7;2 2 4];

b = [59 34 22]';

**2.** Find the rotation matrix that will rotate clockwise the vector ***a*** = by 13 degrees around **x**, 18 degrees around **y** and -30 around **z**. Write a function that can carry out this operation, then, compare your own function with the function 'R\_from\_Euler'. Be ready to discuss the code of this function and/or your own code.

**3.** Derive back the Euler angles from the R matrix found in the previous problem. Use the function 'Euler\_from\_R'. Be ready to discuss the code of this function.

**4.** Write a function that calculates the transformation matrix ***Q*** and the rotation matrix ***R*** for three consecutive rotations around **z**, **x**, and **y**.

**5.** Consider the linear transformation in **R3** represented by the matrix:

such that

a. Explain why this linear transformation has a special meaning in the space defined by the basis:

b. Show the corresponding ***U*** spaceproduct

c. Carry out the *similarity transformation* ***x******⇒ b*** in ***U*** space.

**6.** Consider the following transformation matrix:

Q = [0 1 0;

0 0 1;

1 0 0];

Any transformation matrix can be represented as a rotation around a single axis ***p*** by an angle ***θ***. Use the function 'p\_and\_theta\_from\_Q.m' to identify this axis and angle, and then the function 'Q\_from\_p\_and\_theta' to calculate back the original transformation matrix. You can also try the original algorithm developed by Euler to carry out these operations: they are provided as the functions 'p\_and\_theta\_from\_Q\_alt.m' and 'Q\_from\_p\_and\_theta\_alt'.

**SPECIAL TOPIC**: **Covariant and Contravariant bases.**

Consider 2 vectors represented in Cartesian basis and residing in the xy plane:

F= [3 1 0]'

a = [1.5 2 0]'

If ***F*** represents a *force* (a vector with a 'magnitude = length' and a 'direction') going through a *displacement* ***a***, then the dot product:

***FTa***

Fa = F'\*a

is a *scalar* representing the *work* carried out by the force. Vectors representing particular physical quantities are defined as ***tensors*** if operations associated with these vectors (e.g., their product) follow a characteristic law (based on the Einstein's *summation convention*) under coordinate transformation (change of basis): in this case the *force* and *displacement* vectors are considered as tensors of *1st order* or *rank 1*. Their *scalar* product is considered a *tensor* of *rank 0.*

Now we want to obtain the representation of ***F*** and ***a*** in a *NON-orthogonal* basis ***G***. First we generate 3 random vectors:

g1 = rand(3,1); g2 = rand(3,1); g3 = rand(3,1)

then we convert them to unit vectors:

g1 = g1/norm(g1); g2 = g2/norm(g2); g3 = g3/norm(g3); G = [g1 g2 g3]

Finally, we carry out a *change of basis* by multiplying ***F*** and ***a*** by the inverse of the ***G*** basis:

inv\_G = inv(G)

F\_g = inv\_G\*F

a\_g = inv\_G\*a

Here we recalculate the dot product to obtain a measure of the work carried out by the force ***F***:

Fa\_g = F\_g'\* a\_g

The calculated value of the work is now different! This is a surprising result because we expect the work to be a physical quantity that does not change when we change the coordinates frame of reference. The confusion is occurring because we are forgetting the complete correct definition of the dot product:

Fa\_g = (F\_g(1)\*g1 + F\_g(2)\*g2 + F\_g(3)\*g3)' \* (a\_g(1)\*g1 + a\_g(2)\*g2 + a\_g(3)\*g3)

which must be always applied when the reference basis is not orthogonal.

The need to multiply each element of the ***F*** and ***a*** vectors by the corresponding basis vector when the basis is not orthogonal can be eliminated by introducing the important distinction between *covariant* and *contravariant* bases.

We have learned that to represent a coordinates vector ***u*(Cartesian)** from the standard Cartesian orthogonal basisin a new basis ***G*** we multiply the components of the vector by the inverse of ***G*** as:

Since the components of vector ***a*** transform with the *inverse* of the matrix ***G***, these components are said to transform *contravariantly* under a change of basis. It is a convention of *tensor* notation that the *contravariant components* of ***a*(G)** be represented as *superscripts* *a1, a2, a3***,** while the basis vectors (which instead we define as *covariant*) are represented as *subscripts* ***g1****,* ***g2****,* ***g3***, such that the vector ***a*(G)** be represented as:

For every possible *covariant basis* ***G*** with basis vectors ***g1****,* ***g2****,* ***g3*** (notice the subscript), there exist a unique contravariant basis ***cvG*** whose basis vectors ***g1****,* ***g2****,* ***g3***(notice the superscript) are calculated as follows:

g\_cv1 = cross(g2,g3)/(g1'\*cross(g2,g3))

g\_cv2 = cross(g3,g1)/(g2'\*cross(g3,g1))

g\_cv3 = cross(g1,g2)/(g3'\*cross(g1,g2))

G\_cv = [g\_cv1 g\_cv2 g\_cv3]

We recall here that while the *dot* product between ***a*** and ***b*** is:

the *cross* product (represented by the symbol ) between ***a*** and ***b*** is:

where ***n*** is a unit vector perpendicular to both ***a*** and ***b*** directed as the *thumb* in the right hand rule.

*Covariant* and *contravariant* basis vectors are always bound by the following relationship:

where is the Kronecker delta (). This relationship can be represented as the following matrix product:

which clearly shows that the two bases are *orhogonal* to each other. Solving the corresponding system of 9 linear equations is an alternative way of deriving the contravariant basis from the covariant basis (and viceversa):

G\_cv = inv\_G'

show\_frame('new',eye(3))

show\_frame('add',G,'--r','--g',…

'--b','g1','g2','g3')

show\_frame('add',G\_cv,':r',':g',…

':b','g1\\_cv','g2\\_cv','g3\\_cv')

The figure on the side shows the relationship between the Cartesian ***E*** (continuous lines),covariant ***G*** (dashed lines),and contravariant ***G\_cv*** (dotted lines) bases discussed above.

From the above definitions it is also clear that the *contravariant* basis ***G\_cv*** is the inverse of the transpose (or equivalently the transpose of the inverse) of the *covariant* basis ***G*** and *vice versa*. This is the reason why the contravariant basis ***G\_cv*** is also called the *reciprocal basis* of ***G***. It also follows that:

As usual we obtain the representation of ***a*(Cartesian)** in the ***G\_cv*** basis by multiplying the vector by the basis inverse:

Here we follow the convention that the *covariant components* of ***a(G\_cv)*** be represented as *subscripts* *a1, a2, a3***,** while the *contravariant basis vectors* are represented as *superscripts* ***g1****,* ***g2****,* ***g3***. It is important to realize that the components of ***a(G\_cv)*** are defined as *covariant* because they *covary* with the base ***G***: in fact they can be obtained individually as the dot products of ***a*(Cartesian)** by the covariant basis vectors:

F\_g\_cv = [g1'\*F g2'\*F g3'\*F]'

a\_g\_cv = [g1'\*a g2'\*a g3'\*a]'

F\_g\_cv = G'\*F

a\_g\_cv = G'\*a

Likewise, the components of ***a*(G):**

are defined as *contravariant* because they *contravary* with the base ***G***:in fact they can alsobe obtained individually as the dot products of ***a*(Cartesian)** by the *contravariant basis vectors*:

F\_g = G\_cv'\*F

a\_g = G\_cv'\*a

or equivalently:

F\_g = inv\_G\*F

a\_g = inv\_G\*a

The importance of covariant and contravariant bases lies in the fact that if we express ***F*** in the covariant basis and ***a*** in the contravariant basis (or ***F*** in the contravariant and ***a*** in the covariant basis), then the dot product orhas the same value asthe dot product in Cartesian basis**.**

Fa = F'\*a

Fa = F\_g'\*a\_g\_cv

Fa = F\_g\_cv'\*a\_g

In other words, to calculate correctly the dot product of two vectors represented in a *non-orthogonal* basis ***G***, we multiply one vector by the *inverse* of ***G*** (***G-1***) and the other vector by the *transpose* of ***G*** (***GT***), and then we take their dot product. In this way the dot product of two vectors becomes invariant under coordinates transformation also in the case of non-orthogonal coordinates.

Notice that this rule is generally valid also for *orthogonal* bases, in which case the basis is given by an orthogonal transformation ***Q***, and the transpose is the same as the inverse.

**PRACTICE**

**1.** Calculate the contravariant basis ***G\_cv*** (consisting of the two column vectors ***g1*** and ***g2***) knowing that the covariant basis is:

**2.** A protein crystal has a primitive *triclinic* unit cell with the following dimensions in angstroms and degrees:

a = 45 ; b = 60 ; c = 71 ; alpha = 60 ; beta = 71 ; gamma = 49 ;

We want to find the *basis* for this non-orthogonal coordinates frame. We start by choosing ***a*** as colinear with ***e1*** of the Cartesian frame:

a\_basis\_vec = [a 0 0]'

Then we choose ***b*** by rotating by γ a vector of length 60 originally oriented along ***a***:

Q3 = [cosd(gamma) sind(gamma) 0;-sind(gamma) cosd(gamma) 0;0 0 1]

R3 = Q3'

b\_vec = [b 0 0]

b\_basis\_vec= R3\*b\_vec'

this is the same as:

b\_basis\_vec= [b\*cosd(gamma) b\*sind(gamma) 0]'

At this point we only need to find the 3rd basis vector **c**. We recall here that:

and that:

therefore:

c1 = c\*cosd(beta)

c2 = c\*(cosd(alpha) - cosd(beta)\*cosd(gamma))/sind(gamma)

c3 = c\*sqrt(1-(cosd(beta))^2 -(cosd(alpha) -cosd(beta)\*cosd(gamma))^2/(sind(gamma))^2)

c3 = sqrt(c\*c - c1\*c1 - c2\*c2)

c\_basis\_vec = [c1 c2 c3]

Thus, the basis ***P*** of the triclinic unit cell is :

P = [a\_basis\_vec b\_basis\_vec c\_basis\_vec]

acosd(a\_basis\_vec'\*b\_basis\_vec/(norm(a\_basis\_vec)\*norm(b\_basis\_vec)))

acosd(a\_basis\_vec'\*c\_basis\_vec/(norm(a\_basis\_vec)\*norm(c\_basis\_vec)))

acosd(b\_basis\_vec'\*c\_basis\_vec/(norm(b\_basis\_vec)\*norm(c\_basis\_vec)))

Knowing the ***P*** basis of the triclinic unit cell carry out the following:

a. Calculate the crystal coordinates (usually referred to as 'fractional coordinates') of the point having orthogonal coordinates x = 12, y = 31.0, z = 44.3. Remember that this is the same as representing the xyz vector in the P frame (change of basis). Try to understand the result by plotting in the same box the Cartesian frame, the ***P*** frame, and a test vector with x,y,z coordinates:

show\_frame('new',eye(3))

show\_frame('add',P,'--r','--g','--b','p1','p2','p3')

hold on

x = 12 ; y = 31 ; z = 44.3

test = [x y z]

plot3([0 test(1)]',[0 test(2)]',[0 test(3)]','-c','LineWidth',3)

xlim([0 60])

ylim([0 60])

zlim([0 60])

Can you plot the crystal space components of the test vector, and show that they do sum up to it?

b. Calculate the dimensions of the 'reciprocal space' unit cell of the crystal: this will be the *contravariant* basis of the *covariant* basis ***P***.

**3.** In an earlier earlier practice we calculated the contravariant basis ***G'*** (consisting of the two column vectors ***g1*** and ***g2***) knowing that:

1. The covariant basis is:

2. Covariant and contravariant basis vectors are always bound by the following relationship:

where is the Kronecker delta (). This relationship can be represented as the following matrix product:

Now, let's consider a slightly more difficult 3-dimensional problem in which the covariant basis **G** of the **G**  space is:

a. Calculate the contravariant basis **cvG**.

b. Consider the two particular matrices representing the *stress* and *strain* tensors in Cartesian coordinates:

**τ** is a matrix called the *stress tensor*. **Stress** (force/surface) is a [physical quantity](http://en.wikipedia.org/wiki/Physical_quantity) that expresses the internal [forces](http://en.wikipedia.org/wiki/Force) that neighbouring [molecules](http://en.wikipedia.org/wiki/Particle) of a [continuous material](http://en.wikipedia.org/wiki/Continuum_mechanics) exert on each other. For example, when a [liquid](http://en.wikipedia.org/wiki/Liquid) is under [pressure](http://en.wikipedia.org/wiki/Pressure), each small region of the liquid gets pushed inwards by all the surrounding molecules. This hydrostatic pressure is an example of *isotropic stress* because the forces acting on each small region of the liquid are the same in every direction.

**ε** is a matrixcalled the *strain tensor*. **Strain** is the infinitesimal deformation of a material (expressed as a pure number = change in length/original length) that generates the stress. Stress and strain are very important in the theory of elasticity (for example, a fibrous proteins like collagen behaves as an elastic material under small deformations).

The **strain energy density** (Estrain\_dens = work/volume = force x displacement/volume) associated with a small deformation of a unit volume of material at a certain point in space can be calculated as the ***double dot product*** (**:**) of the stress and strain tensors: . This product is calculated by summing all the products between the components of the two tensors with equal indices:

such that:

c. Calculate the *strain energy density* in Cartesian space and in **G** space. You will need to operate by *similarity transformation* to move the two tensors into the new space. Furthermore, recall that under coordinate transformation the double dot product follows the same rule as the dot product; thus you will need to represent the **τ** tensor in contravariant **cvG** basis and the **ε** tensor in covariant **vG** basis (or viceversa).