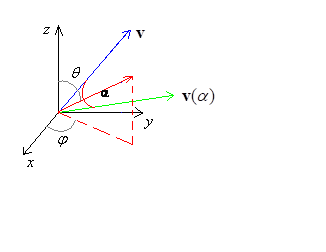
**Rotation Operator**

**Math Prelude on the Rotation matrix**

Let’s talk about spatial rotations of vectors (classical Cartesian vectors) for a second. Then we’ll segue into rotations of quantum mechanical wavevectors and operators. So consider a vector **v**. And suppose we rotate it through an angle **α**, meaning we rotate an amount (in radians) α about **α** direction, where,



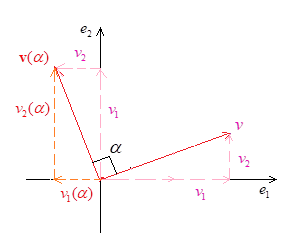
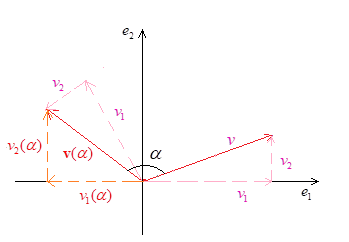
Then it will look something like **v**(α) below.



I’d like to work out the formula for calculating what **v**(α) is. The language of operators and basis kets is useful, so I’ll borrow QM notation. Let the unit vectors of our coordinate system be represented as |ei>, and an arbitrary vector be represented as |v>, and let the Rotation operator be designated (α). We can say that in general, our transformed vector can be written as:



which is to say, we can find the new vector by keeping the old components, and just rotating the basis vectors. It’s more pertinent to us to consider how to get a representation of the vector in terms of the old basis set. In other words, we want to know how *components* change, rather than how the *basis* *vectors* change. Just for visualization sake, here’s a few illustrations.

When we rotate a vector, **v**, with initial components v1, v2, by **α**, resulting in **v**(α), we want to know what the new components v1(α), v2(α) are, in terms of the original v1, v2. You can think of the original v1, v2 as ‘rotating’ with the vector, but of course these will no longer constitute the 1, 2 components anymore. But we do want to know what the new v1,2 are in terms of these old ones. For instance in the left diagram, where we rotate the vector by 90o, we have v1(α) = -v2, and v2(α) = v1. There are three equivalent ways to work this out. First, we can do:



where we have of course defined:



as the components of the rotated vector **v**(α) in the old basis. In the 4th line we note that since R(α) is a unitary operator (and really, it’s symmetric, because we don’t have any complex numbers here), its dagger (transpose) is its inverse. And the inverse of R(α) is just R(-α). So in this scenario, we see how the components of the vector change (in the old basis). And they change by projecting the present vector onto the basis rotated in the opposite direction. We can take one more step to put this in a format which makes it easier to think about it mentally.



Now we’ll explicitly presume we’re only dealing with real vectors, so we can take the \* off. Then we can say,



This formula says that vi(α) is obtained by taking the associated unit vector |ei>, rotating it the opposite way, finding its components in the old basis, and multiplying each component by the original components of the vector. This way requires and example, but often doing it like this is easier, at least in your head. Last, it would be nice to get a formula for vi(α) in terms of the matrix elements of the rotation operator:



Starting from the top of the last thing, we can write:



So we have:



And now I’d like to know what the matrix components, Rij(α), of the rotation operator are, for a general rotation. So physically, it is the matrix which rotates vectors by angle α counter clockwise about unit vector . It’s easiest to figure out the components by kind of working backwards. Let’s change coordinates to a system whose z-axis is aligned with the vector **α**. Let’s call this basis |enewi>. Then, inserting resolutions of identity:



The middle matrix is just a simple ‘rotation about the znew-axis’ matrix,



To get the U matrix, we just have to figure out the projection of the new basis vectors onto the old ones. The new basis vectors can be obtained from the old ones by (see diagram above) first rotating our original z-axis by θ about the original x-axis, and then rotating our new z-axis by φ about the original z-axis. The transformation/rotation matrix U would be:



Explicitly,



So putting it all together, we have:



This doesn’t simplify too much. Nonetheless, given this expression the components of **v**(α) may be determined from the **v** components as stated above:



So note that each of the individual components transforms as the *components* of a vector.

**Example**

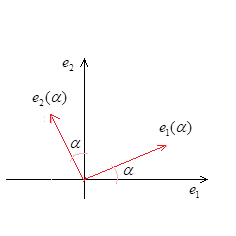
Gonna do a quick example utilizing these formulas. We start with an arbitrary vector **v** = v1**e**1 + v2**e**2, say, and rotate it, about the z (**e**3) axis say, and get the new vector **v**(α) = v1(α)**e**1 + v2(α)**e**2. So what are vi(α) in terms of vi? Of course we have the general formula. It’s



But we can do ourselves too. We have:



The rotated basis vectors are illustrated below:



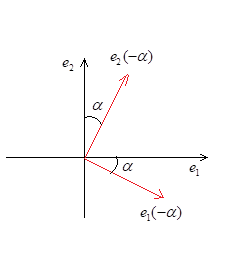
And so we have:



So this matches. And either way,



What if I just wanted to know how v2 changed, say? Could do this using the 2nd of the four highlighted formulas. Actually I’ll use it to get both components.



So we do,



Now on to QM….

**Quantum Mechanical Orbital Rotation Operator**

Let’s consider the rotation operator for small angles δ**α**. You will recall that we determined, when considering the Heisenberg formulation of quantum mechanics, that:



It follows that



So we have determined that the generator of spatial rotations is the orbital angular momentum operator **L**. Immediately we have that the full spatial rotation operator is:



So we have the following symmetry/conservation law:



Note that since we (assume) we have conservation of angular momentum, then this means the Hamiltonian of the universe must be invariant w/r to spatial rotations.

**Example**

Express the rotation operators D(α**i**), D(α**j**), D(α**k**) in the |ℓm> basis, where ℓ = 1. So to express these operators in the basis, we need to calculate:



where **L**x is the operator expressed in the Lz eigenbasis. From previous work we know this is (see Time-Independent folder, and Angular Momentum files):



Evaluating powers of this matrix a few times and we will see a pattern, namely:



and putting this in our expansion gives,



Now let’s work out the D(α**j**) rotation matrix. First, in the Lz basis, Ly looks like,



Working out powers of this matrix we have:



Filling these into the expansion we get:



Finally, the last rotation matrix is easy to evaluate:



Summarizing, the rotation matrices are:



Might observe that for 2π rotations, we do indeed get 1. What about π rotations? Then we have:



Say we did a π rotation of a spin up ket (1 0 0) about the x or y axis. Physically this would make it spin down (0 0 1). Is this what we get? Yes, well – (0 0 1). But phase shouldn’t matter per se´. Importantly, we get the same thing regardless of whether we do the rotation about the x or y axes.

**D(α) acting on kets**

Let’s look at the effect of the rotation operator on some others. Generally, operators which are invariant with respect to spatial rotations are called *isotropic*. Let’s look at its operation on kets. We will find:



This says that if we start with a position ket associated with the coordinate **r**, and we rotate it by **α**, we will end up with the ket associated with the coordinate **r**´ = **r**(α) (to use notation above). We could say that this is the definition of D, since it *is* how we constructed it in that Heisenberg Representation file in the first place. So we’ll take the first as given. Next, its action on |**p**> is to similarly rotate it by **α**. Apropos |ℓm>, we’re saying that suppose that we have a unit vector, , which we define as our z-axis. Then |ℓm> are the eigengets of the corresponding L2, **L**· operators (I’m writing what I might call Lz as **L**· instead so that the transformation properties of **L**· don’t get confused with those of Lz). Now say we rotate everything by **α**. Then goes to ´. And we’re saying that the simultaneous eigenkets of L2, **L**·**´** are just |ℓm´> = (α)|ℓm> (and we’re implicitly claiming that ℓ doesn’t change). Apropos the spin ket, we’re saying that the orbital rotation operator has no effect on spin, which makes sense, since the two operators commute. Alright, let’s look at the 2nd line then. We have:



Now the determinant of the rotation matrix is 1, which is evident since its transpose is its own inverse. This is because:



And so we get:



which indicates that the rotation operator acting on |**p**> rotates it in the same way it does **|r**>. Now let’s work on the angular momentum eigenket.



In order to interpret this expression we’ll make the following considerations. First note that since [L2, Lj] = 0, we have that [L2, **L**·**α**] = 0 → that [L2, e-i**L**·**α**/ћ] = 0. And this implies that e-i**L**·**α**/ћ |ℓm> is an eigenstate of L2 with eigenvalue ћ2ℓ(ℓ+1). So that justifies our saying that ℓ doesn’t change. Next we have to show that it is an eigenstate of the **L**·**´**operator. So this operator is the following guy,



where in the last line we use the fact that the inverse of the rotation matrix is the same as its transpose, and so inversing, i.e., taking α → -α, and transposing cancel each other out. From the next section, we’ll recognize this as being the operator we get via:



where the last line comes from fact that D’s are unitary operators, and so their inverses are their daggers (and inverses are obtained by α → -α). Okay, so now we say,



So there we go. But this analysis doesn’t resolve the phase of the new ket. Moving on to the spinors, of course the action of D(**α**) on |sm> is unity since it doesn’t act on that space. Generally speaking the action of the rotation operator on a wavefunction, in position space is:



where R-1(**α**) = R(-**α**) is the inverse rotation. And R-1(**α**)**r** is short for (R-1(**α**))mnxn(implicit summation over repeated indices).

**D†(α)…D(α) acting on operators**

As for the operators we will have (implicit summation over indices):

where R(**α**)ij is the 3D rotation matrix which rotates a vector about the angle **α**. The formulas on right are just the component-by-component versions of those on left. Just to be clear. The individual components of the vector operators transform as, well, components of vectors, not as vectors themselves. So for instance if we consider , and apply to it a 90o CCW rotation about the z axis, we will not get . Rather will become: -. This is because when we rotate a vector 90o, the new x-component, i.e. the vector’s new projection against the x-axis, will be minus its old y-component (see maybe that left-hand illustration on pg. 2). So the rotation operator tells us what the new x, y, z component of the vector will be (its new projections against the x,y,z axes), in terms of the initial x, y, z, components, after the rotation is applied. Well starting with the first we have:



The first commutator is:



The next order term would be:



continuing, it appears we get the infinite expansion,



That’s interesting but doesn’t really help. Let’s try a different approach. Consider:



So we have, like we would expect. Interestingly, this also tells us that the infinite series we generated above is equal to **r**′. This last procedure that we used is also applicable to the translation operator as well, and would’ve delivered our result more quickly. Let’s look at its action on **p**.



Working out the first commutator we have:



and so the second commutator would be:



so generally, we’d have:



which from our work before, we’ll recognize results in **p**′. which is **p** just rotated by **α**. Now let’s look at the angular momentum operator. This one is easy,



where **L**′ is the angular momentum operator (vector) rotated by **α**. Notice that it doesn’t equal **L**; this is because D(**α**) and **L** do not commute because the different components of **L** don’t commute with each other. This wasn’t a problem for the translation operator because the different components of **p** do commute with each other. Finally let’s consider spin,



So spatial rotations do not affect the spin. All of these results, you will observe, are intuitive based simply on classical grounds. 90% of the results following you can intuit just be thinking about what it means classically.

**General Rotation Operator**

We can generalize the rotation operator above to include spatial and spinor rotations. The general rotation operator would be:



So we have the following symmetry/conservation law:



We’d like to see what D’s action is on the other operators considered. It will be the same as the orbital rotation operator, except for the spin kets. So let’s analyze the effect of the spin rotation operator on spin states, as a means to get its effect on spin operators. Although we could work this out in general for any spin – fractional or otherwise, let’s just focus on the important case of the electron.

**Example**

We can fully express the rotation operator in this case, just as we did for ℓ = 1 above, and therefore explicitly see its effects on other kets.

So first,



Now,



Filling these in we get:



and for y-rotations we have:



and finally for rotations about the z-axis we have:



Now that we have the rotation matrices explicitly:



it is interesting to observe the following features peculiar to rotations of spinors. It is interesting to observe that we get -1, when α = 2π.



What about when α = π? Then we have:



**Operation of D on spin kets**

So we can repeat all the arguments we made for orbital angular momentum that when we operate D(α) on |sm>, we get:



which is to say that if we start with a state which is a simultaneous eigenstate of the 2, **S**· operators, and then rotate it by α, we end up with a state which the simultaneous eigenstate of the 2, **S**· operators. But there is something else we didn’t have before: the phase. We see from our explicit representation of the rotation operator above, that there will be an anomolous phase attached to the eigenstate. For instance, we saw that a 2π rotation about any axis multiplies the ket by (-1). This means that if we rotate a spinor through 2π, it rotates into itself, but with a phase factor of -1. In fact we have to do a 4π rotation to return it to its original phase. This anomalous behavior highlights the non-spatial nature of particles’ spin-states. And we can see that if we rotate a spin up ket by π radians about the x-axis, we get a spin down ket, but with an anomolous phase factor of

-i. And if we rotate a spin down eigenket π rad about the x-axis, we also get an anomolous factor of -i. Now if we do the same π rotation of a spin up eigenket about the y-axis, we get a spin down state with no anomolous phase factor, though if we rotate a spin down eigenket by π rad about the y-axis, we do get an anomolous factor of -1. So the phase seems to be path-dependent.

**D†(α)…D(α) acting on spin operator**

As for the operators we will have (implicit summation over indices), we have:

I think. For instance †(2π)z(2π) = -z.