Bset 1: Orientation and Particle Motion

In[1282]:=
```
SetDirectory@NotebookDirectory[];
<< "../MMA library.m"
```

Reflection

- This document lives in Mathematica, so reflection and notes are formatted green instead of my normal blue.
- Working in Mathematica was kind of nice, but I'm not sure I'll do it again. Being able to rapidly scan and insert images is a hard feature to lose.
- This Bset felt harder and longer than those in the Communications module, but that could simply be an artifact of my lack of familiarity with this content. As a result, I didn't finish and document my work to my normal standard, for which I apologize. I'm not currently worried about falling behind because I think I understand the core concepts sufficiently well to move on, but I still wish I had been able and willing to put a little more time in.

Euler Angle Review

```
With[{context = "s1`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

```
Global`
```

1. What's the Matrix?

```
yawMatrix = RotationMatrix[ψ, UnitVector[3, 3]];
pitchMatrix = RotationMatrix[θ, UnitVector[3, 2]];
rollMatrix = RotationMatrix[ϕ, UnitVector[3, 1]];
```

In[1046]:=
```
rollMatrix.yawMatrix.pitchMatrix//MatrixForm
```

Out[1046]//MatrixForm=
```
rollMatrix.yawMatrix.pitchMatrix
```

- I'm not 100% solid about why this is "roll*pitch*yaw" instead of "pitch*yaw*roll". Because the roll operation is applied last to the body, I want to multiply it *first* such that its effect isn't affected by the other transformations (if that made any sense).

```
With[{context = "s1`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

```
Global`
```

Build a Gimbal

Gimbal Lock

8. What is gimbal lock?

Gimbal lock is a situation in which two points that are adjacent (or arbitrarily close) on a sphere have dramatically different representations as angles pointing to them. This usually (always?) occurs at singularities, like the poles of a pan-tilt-yaw coordinate system. When a system is in gimbal lock, moving a small amount in one direction requires coordinated and long motion of multiple axes, which can cause componentwise interpolation to fail. This is particularly challenging when dealing with a physical system in which the axes cannot reorient arbitrarily quickly or when dealing with an animation environment in which smooth interpolation is a critical feature of the system.

9. Find it.

In 3-1-3 Euler angles, gimbal lock occurs whenever the second rotation is 0. In this case, the first and third axes are collinear.

In 3-2-1 angles, gimbal lock occurs when the second rotation is ±90 degrees, because that again puts the first and third rings into parallel alignment.

Particle Motion

I am a leaf on the wind. Watch how I soar.

In[1055]:=
```
With[{context = "s4`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[1055]=
```
Global`
```

- **As a result of being in Mathematica, some of the rote transformation steps aren't necessary for me to solve these types of problems (particularly converting second-order vector DE's to first-order scalar DE's). I know how to *do* them, but won't be doing so unless actually necessary.**
- **For any others wondering how I made this work with vector equations, note that I defined my forces as functions that only evaluate when fed a list of numbers. This prevents things from simplifying incorrectly before they should.**

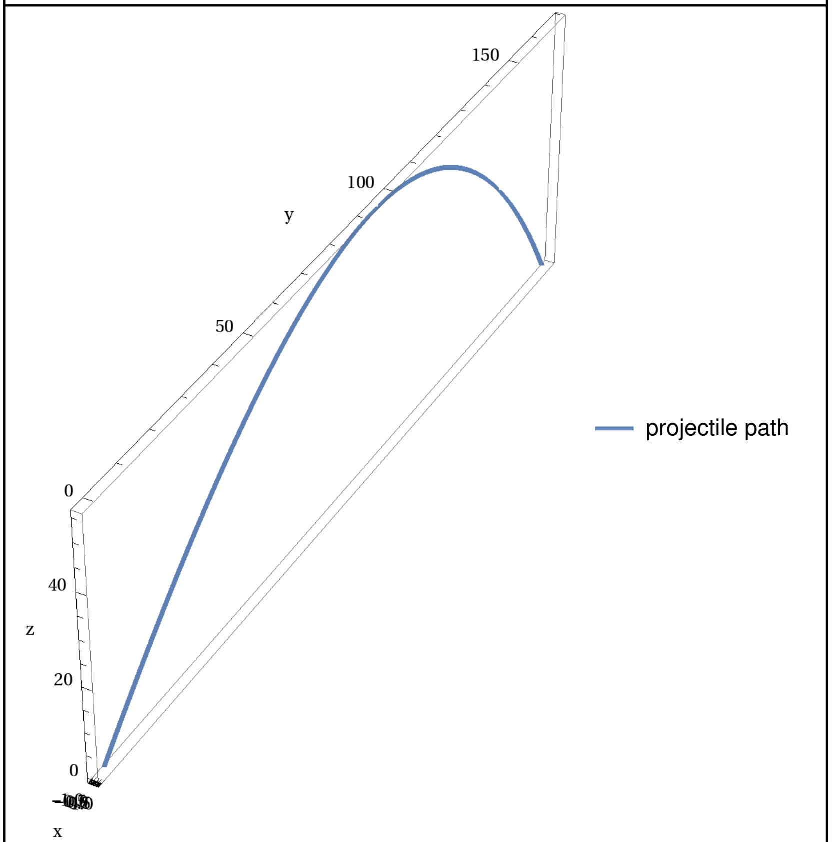
In[1233]:=
```
ClearAll@windyLeaf
windyLeaf[wind_] :=
 Module[{x, t, fg, fd, distance, end, simRange, diffeqs, events, initialConditions},

  (* Setup initial conditions *)
  initialConditions = {x[0] == {0, 0, 0},
    x'[0] == QuantityMagnitude@UnitConvert[ 130 mi/h , m/s ] {0, 1/Sqrt@2, 1/Sqrt@2}};

  With[
    (* Setup parameters *)
    {w = wind, m = 0.05, g = 9.8, cd = .25, d = 4.27/100, ρ = 1.29},

    (* define forces as function of velocity *)
    ClearAll[fg, fd];
    fg[v_?(VectorQ[#] &)] := {0, 0, -m g};
    fd[v_?(VectorQ[#] &)] := (-1/2 ρ cd (Pi d^2/4) Norm[v - w] (v - w));

    simRange = {t, 0, Infinity};

    diffeqs = {(fg[x'[t]] + fd[x'[t]] == m x''[t])};
    events = {WhenEvent[x[t][[3]] == 0,
       end = t;
       distance = Norm[x[t]];
       "StopIntegration"]};
   ]

sol = NDSolveValue[Join[initialConditions, diffeqs, events], x[t], simRange];
  Column[{
    Join[initialConditions, diffeqs] // TraditionalForm;

    Row[{"Distance travelled: ", distance, " meters"}],

    ParametricPlot3D[sol, {t, 0, end}, PlotLegends → {"projectile path"},
     AxesLabel → {"x", "y", "z"}, PlotRange → Full, ImageSize → Medium],
    With[{dsol = sol[[0]]'[t]},

     Plot [{sol[[1]], sol[[2]], sol[[3]], dsol[[1]], dsol[[2]], dsol[[3]]}, {t, 0, end},
      PlotLegends → {"x", "y", "z", "x'", "y'", "z'"}, ImageSize -> Medium]]},
   Frame → All, Alignment → Center]
 ]
```

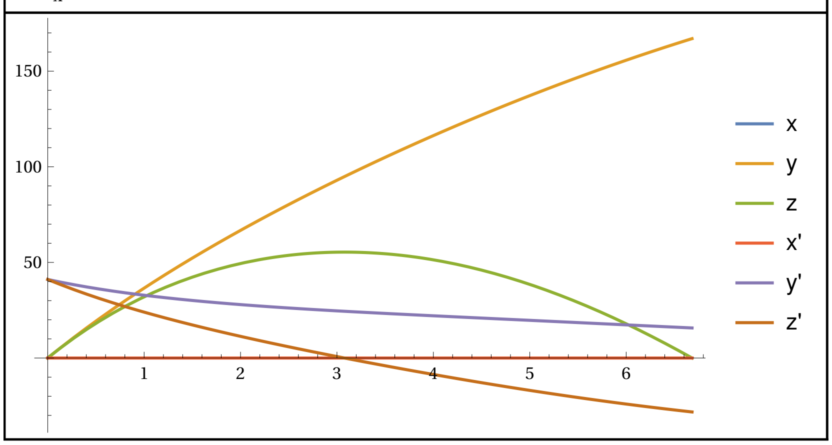
Results with no wind:

In[1235]:=
```
windyLeaf[{0, 0, 0}]
```

Out[1235]=

Distance travelled: 167.014 meters



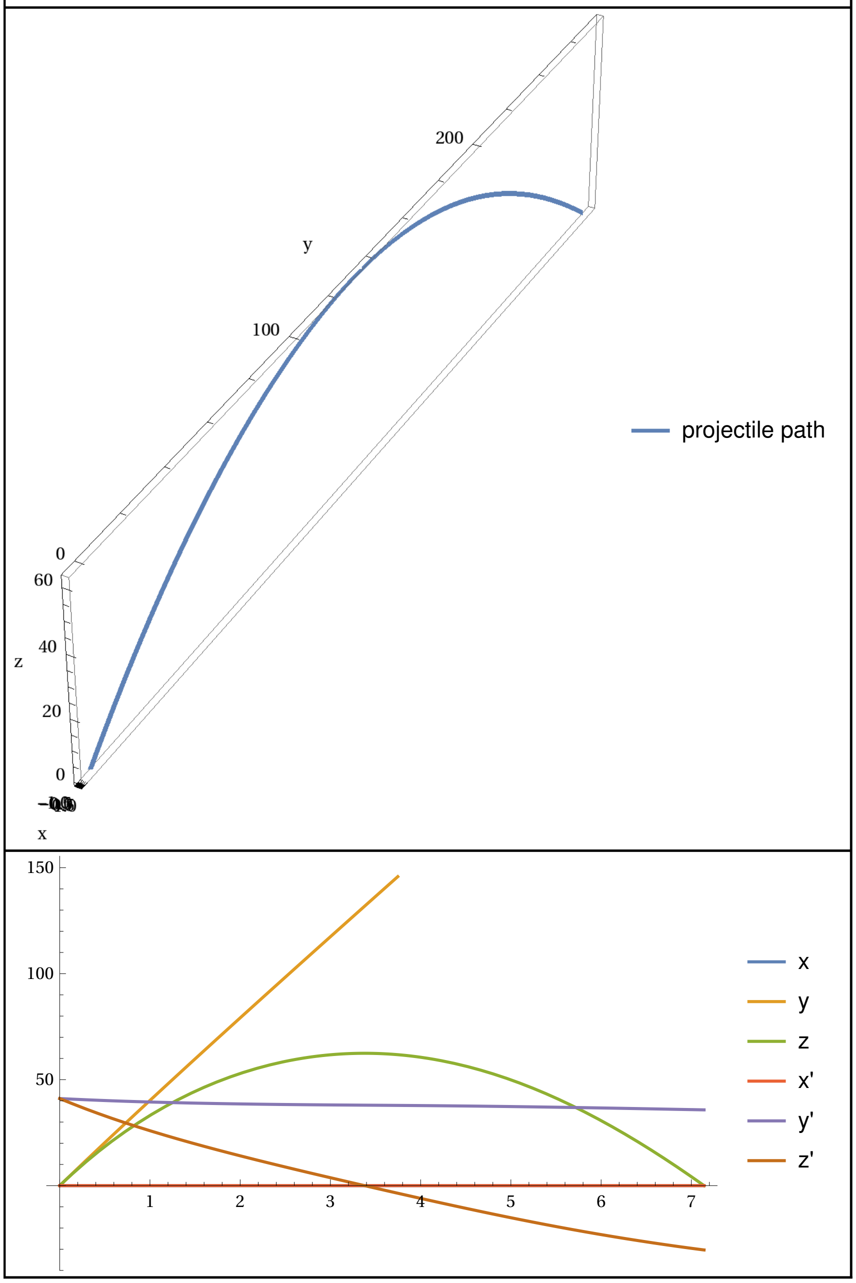


Results with an extreme tail wind

In[1242]:=
```
windyLeaf[{0, 30, 0}]
```

Out[1242]=

Distance travelled: 271.217 meters

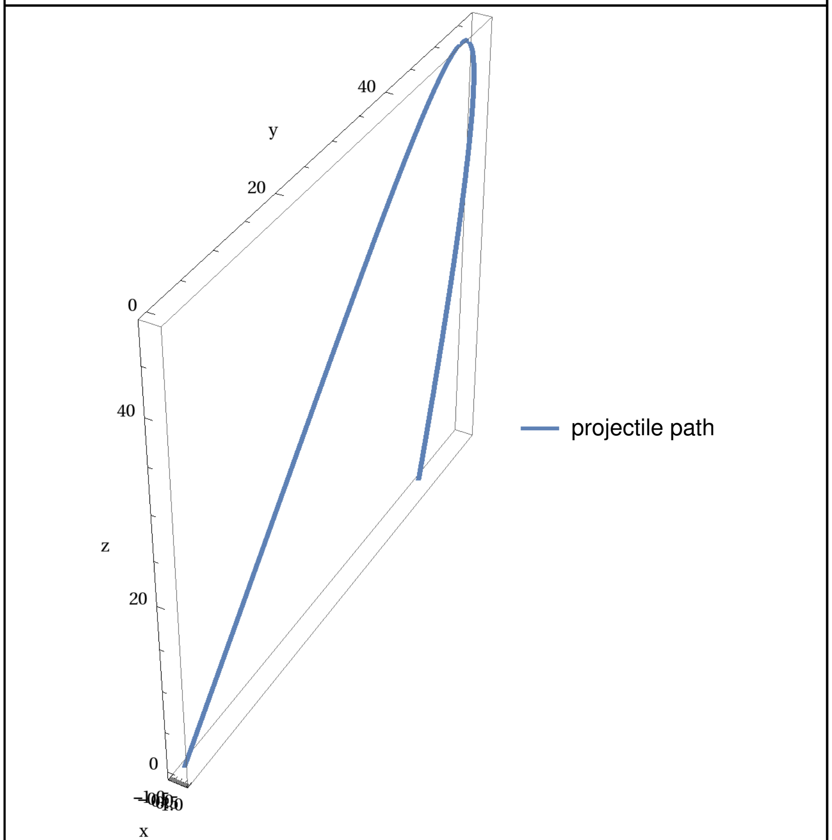


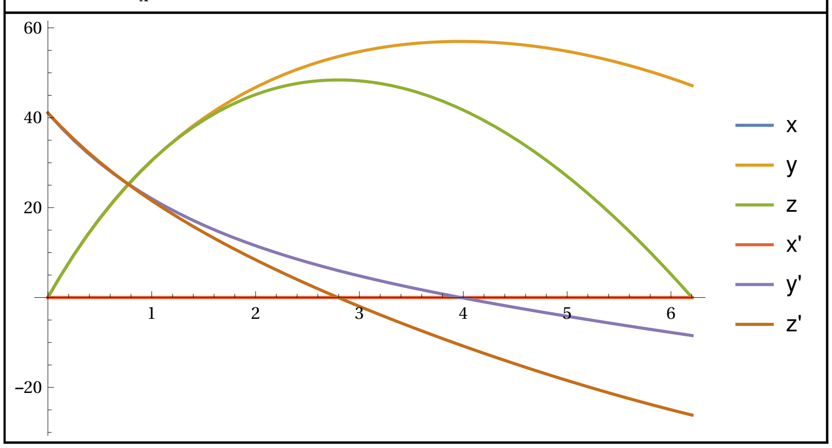
Results with an extreme headwind

In[1241]:=
```
windyLeaf[{0, -30, 0}]
```

Out[1241]=

Distance travelled: 47.1504 meters



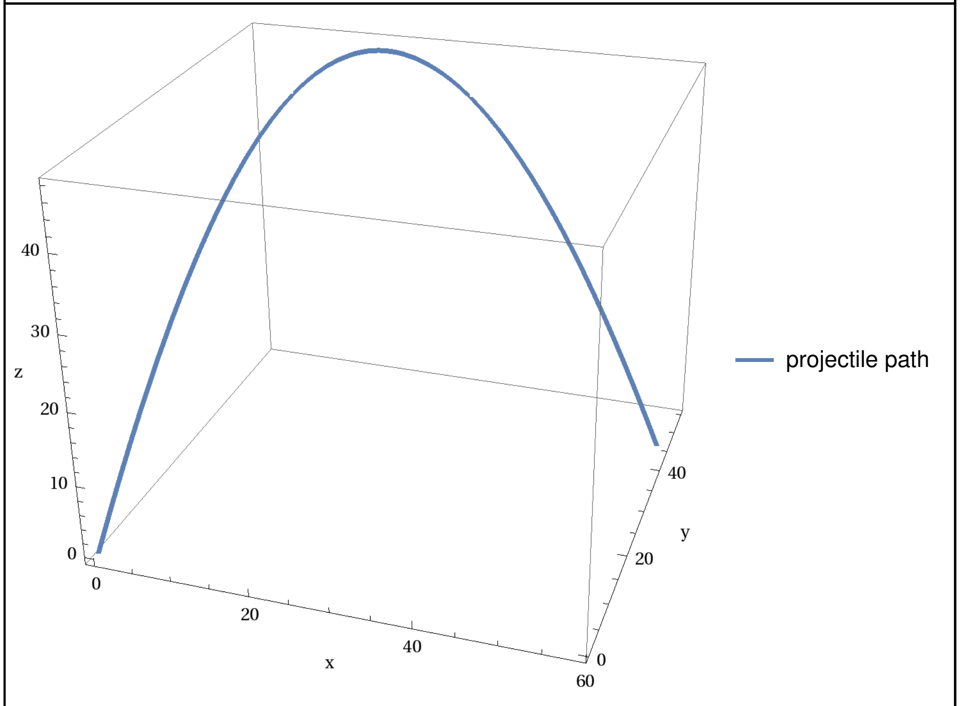


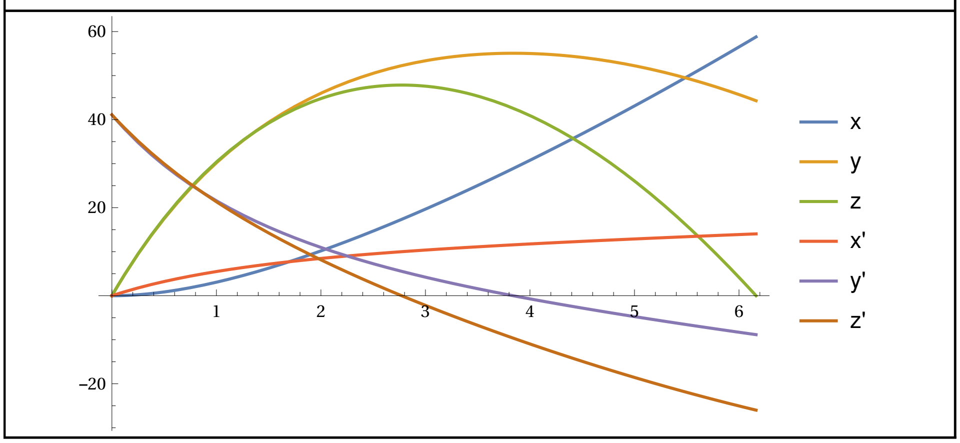
Results with headwind and crosswind

In[1244]:=
```
windyLeaf[{20, -30, 0}]
```

Out[1244]=

```
Distance travelled: 73.6063 meters
```



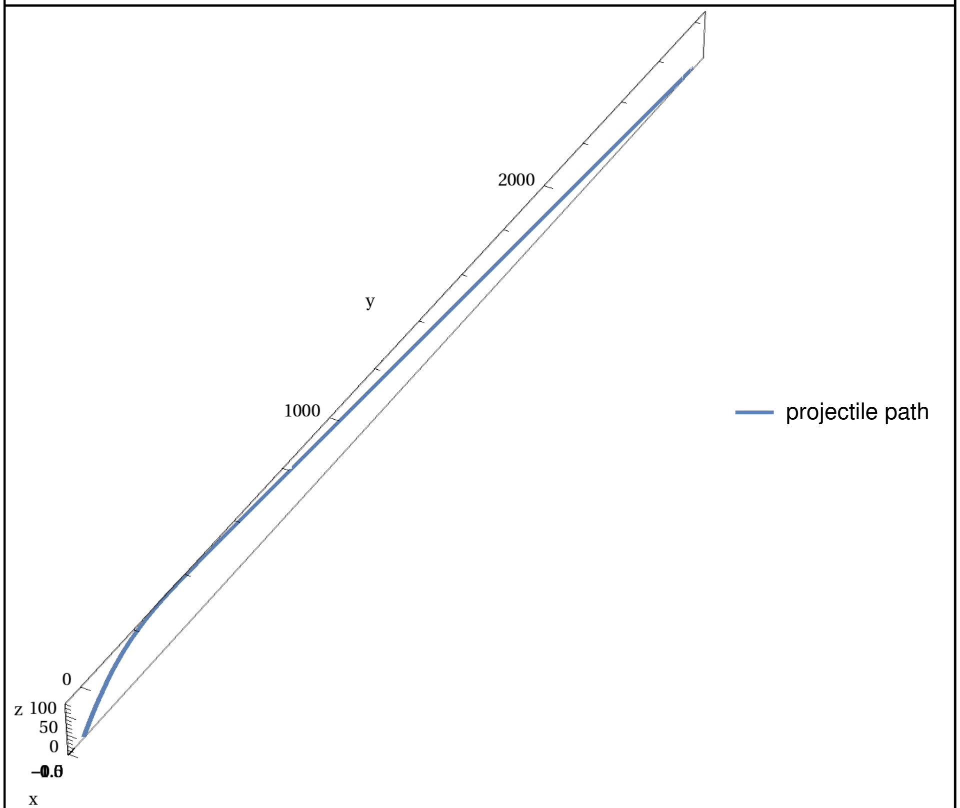


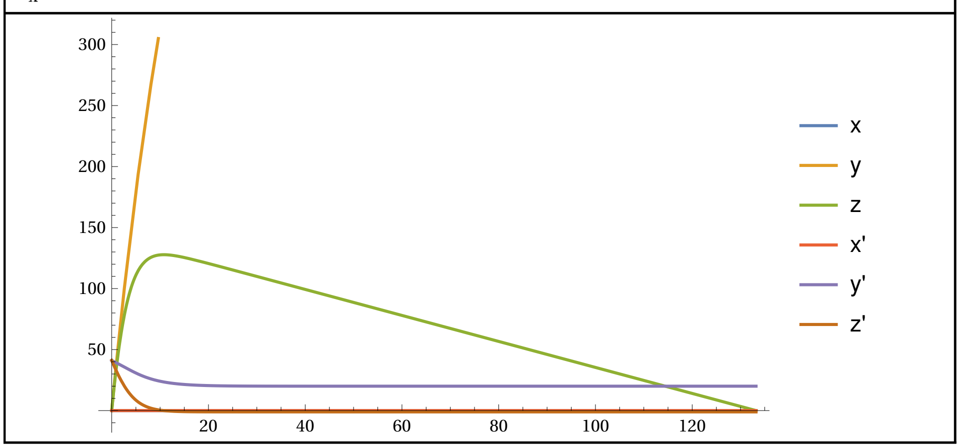
Results with strong updraft and tailwind

In[1250]:=
```
windyLeaf[{0, 20, 45}]
```

Out[1250]=

Distance travelled: 2797.81 meters





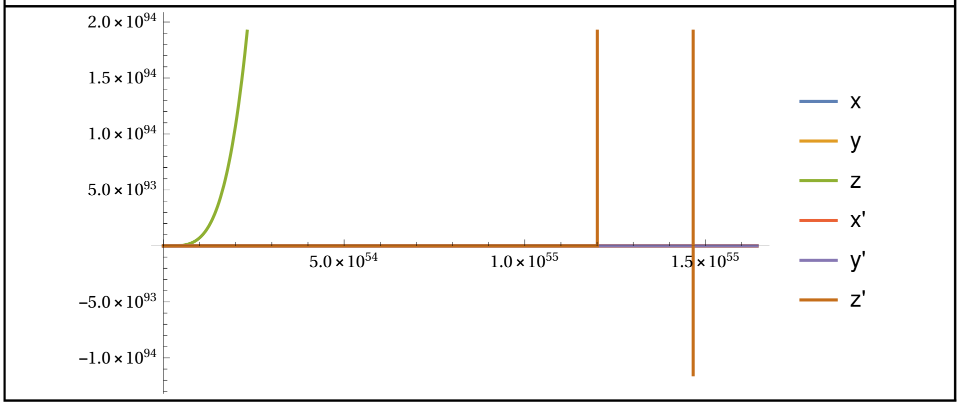
Results with extreme updraft

In[1255]:=
```
windyLeaf[{0, 0, 70}]
```

NDSolveValue: Event location failed to converge to the requested accuracy or precision within 100 iterations between t$149645 = 1.2009865208330457`*^55 and t$149645 = 1.651356431855972`*^55.

Out[1255]=

Distance travelled: 2.78642×10^{81} meters



◇ **At this point, I had spent at least three hours trying to get this working (and then trying to understand why it works). As a result, while I think I am capable of doing the validation, limiting cases, etc., I didn't write up anything to that effect. Sorry.**

In[1257]:=
```
With[{context = "s4`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[1257]= Global`

Cylindrical Coordinate Systems

In[1287]:=
```
With[{context = "s5`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[1287]= Global`

17. Explain the math

Because the direction of $\hat{e}_r$ depends on θ, the equation given *does* include three variables, r, z, and $\hat{e}_r$.

18. Derive acceleration

Step 1: Take the derivative

In[1324]:=
```
velocity = r'[t] * er[t] + r[t] * θ'[t] * eθ[t] + z'[t] * ez
accel = ∂t velocity // TraditionalForm
```

Out[1324]= er[t] r′[t] + ez z′[t] + eθ[t] r[t] θ′[t]

Out[1325]//TraditionalForm=

$$r(t)\,\mathrm{e}\theta'(t)\,\theta'(t) + \mathrm{e}\theta(t)\,r'(t)\,\theta'(t) + \mathrm{e}\theta(t)\,r(t)\,\theta''(t) + \mathrm{er}'(t)\,r'(t) + \mathrm{er}(t)\,r''(t) + \mathrm{ez}\,z''(t)$$

Step 2: Eliminate derivatives of unit vectors

In[1329]:=
```
replacements = <|er'[t] → θ'[t] eθ[t],
   eθ'[t] → -θ'[t] er[t]|>;
accel /. replacements;
Simplify@% // TraditionalForm
```

Out[1331]//TraditionalForm=

$$e\theta(t)\,(2\,r'(t)\,\theta'(t) + r(t)\,\theta''(t)) + \mathrm{er}(t)\left(r''(t) - r(t)\,\theta'(t)^2\right) + \mathrm{ez}\,z''(t)$$

As expected, this equals the value given in the problem set.

The Coriolis acceleration reflects the extent to which moving in a radial direction while rotating causes you to move ahead or behind relative to the rotation you would expect. This is relevant in hurricanes, or on a really good playground merry-go-round.

◇ **It is really cool to see the Coriolis Acceleration derived, thanks.**

19. Time derivatives of unit vectors

A constant magnitude vector can occupy only points on a sphere of fixed radius. Any continuous motion along the surface of that sphere must have direction tangent to the surface at that point. Because the radius of a sphere (to a given point) is always perpendicular to its tangent line at that point, the derivative of a moving vector must be orthogonal to it, implying that their dot product is zero.

In[1334]:=
```
With[{context = "s5`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[1334]= Global`

Spiral learning (Wheeee!)

In[1338]:=
```
With[{context = "s6`"}, If[Context[] ≠ context, Begin[context]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[1338]= Global`

20. Describe the helix

In[1353]:=

$$r(z\_) := \text{radius}$$

$$\theta(z\_) := \frac{z}{\text{radius}\tan(\beta)}$$

21. Estimate system parameters

age	10 years
body weight	71 lb (pounds)

In[1375]:=
```
radius = 1 m;
height = 30 ft;
β = 30°;
m = 71 lb;
```

◇ **I hit my time limit (and bedtime) before finishing this problem. If I get a chance later, I may complete it and re-upload the result. As a first guess, the equations will simplify by removing r(t), θ(t), r'(t), and θ'(t), leaving just two equations representing the vertical velocity and acceleration.**

In[1379]:=
```
With[{context = "s6`"}, If[Context[] == context, End[]]];
Dynamic[Refresh[Context[], UpdateInterval → 1]]
```

Out[1379]= Global`

Scratch Work

In[1380]:=
```
exportNotebookPDF[]
```