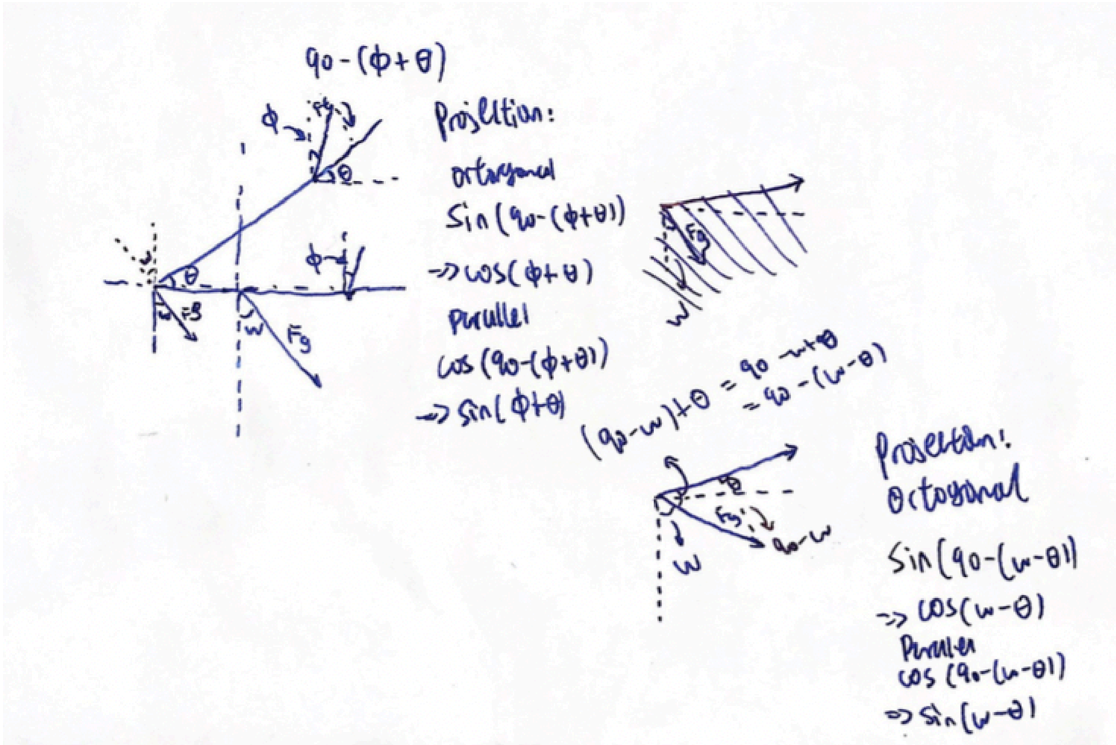
Let's draw a picture of this situation!



We first set up the basic assumptions and variables.

```
GRAV <- 9.8 # gravity (m/s^2)
MASS <- 1 # mass (kg)
I_CM <- 1/12 # roational inertia at the centre of gravity (kg m^2)
L1 <- 0.5 # distance from rotation point to CoM (m)
L2 <- 1 # distance from rotation point to tension (m)
PHI <- 0 # angle of Ft relative to floor (orthogonal) (rad)
FT <- 11 # tension force (N)
OMEGA <- 0 # angle of line orthogonal to floor relative to gravity (rad) (because shifted axis)
```

Additionally, we set the time interval and seed values for time and theta (distance from flat):

```
dt <- 0.00001
t_max <- 0.5

vx <- 0
vy <- 0

x <- L1 # initial x is L1 because that's the Rotation Point => CoM distance, and rot point is 0
y <- 0

theta <- 0
thetadot <- 0
time <- 0
```

Great. Let's start generating the table! We essentially write a for loop to appends to a few different vectors. Variables appended with `c` reflect the column vectors that we will put together.

```
cTime = NULL
cTheta = NULL
cDDTheta = NULL
cDTheta = NULL
cTorqueNet = NULL
cAccelX = NULL
cAccelY = NULL
cVelX = NULL
cVelY = NULL

cPosX = NULL
cPosY = NULL

cPosPX = NULL
cPosPY = NULL

cFFriction = NULL
cFNormal = NULL

# debugging values
cFNetY = NULL
cFTensionPhiComponent = NULL
cFGravityPhiComponent = NULL

cMuStatic = NULL
cKERot = NULL
cKETrans = NULL
```

Awesome. Let's now run a lovely little for loop to actually populate the values recursively.

```
for (i in 0:(t_max/dt)) {
    # We first populate the time column with the time, theta column with theta
    cTime[i] = time

    # Given the theta value, we calculate the net torque and set that
    I_ROT <- I_CM + MASS * L1^2 # we calculate I_ROT using
# the Parallel axis theorem

    torque <- L2 * FT * cos(theta + PHI) - L1 * MASS * GRAV * cos(theta - OMEGA)
    cTorqueNet[i] = torque
    # Now that we know the net torque, we could know how much the angular
    # acceleration is by just dividing out the rotational inertia
    thetadotdot <- torque/I_ROT
    cDDTheta[i] = thetadotdot
    # We could also multiply the theta acceleration by time to get the
    # velocity at that point
    thetadot <- dt*thetadotdot + thetadot
    cDTheta[i] = thetadot

    # we then tally the theta value
    theta <- dt*thetadot + theta
    cTheta[i] = theta

    # We could therefore component-ize the acceleration in theta, times
```

```
    # the length of the object until com, to figure the acceleratinos
    # of the com
    ax <- -1 * L1 * sin(theta) * thetadotdot
    cAccelX[i] = ax
    ay <- L1 * cos(theta) * thetadotdot
    cAccelY[i] = ay # @mark isn't sin and cos backwards?

    # "position prime": calculated positino
    cPosPX[i] = cos(theta)*L1
    cPosPY[i] = sin(theta)*L1

    # We also tally the components seperately for velocity
    vx <- ax*dt + vx
    vy <- ay*dt + vy

    # We finally tally the positions as well
    x <- vx*dt + x
    y <- vy*dt + y

    cPosX[i] = x
    cPosY[i] = y

    # Based on these accelerations, we therefore could calculate the relative
    # force of friction and normal force by subtracting the force in that direction
    # out of net
    ffriction <- FT*sin(PHI) + MASS*GRAV*sin(OMEGA)-MASS*ax
    fnormal <- MASS*ay-FT*cos(PHI)+MASS*GRAV*cos(OMEGA)

    cFNetY[i] = MASS*ay
    cFTensionPhiComponent[i] = FT*cos(PHI)
    cFGravityPhiComponent[i] = -MASS*GRAV*cos(OMEGA)

    cFFriction[i] = ffriction
    cFNormal[i] = fnormal

    # Then, we calculate the energies
    cKERot[i] = 0.5 * I_ROT * thetadot^2
    cKETrans[i] = 0.5 * MASS * (vx^2+vy^2)

    # Dividing the friction force by the normal force, of course, will result in
    # the (min?) friction coeff
    cMuStatic[i] = ffriction/fnormal

    # We incriment the time and also increment theta by multiplying the velocity
    # by dt to get change in the next increment
    time <- dt + time
}
```

We now put all of this together in a dataframe.

```
rotating_link <- data.frame(cTime,
    cTheta,
    cDTheta,
    cDDTheta,
```

```
    cTorqueNet,
    cAccelX,
    cAccelY,
    cPosX,
    cPosY,
    cPosPX,
    cPosPY,
    cFFriction,
    cFNormal,
    cMuStatic,
    cKERot,
    cKETrans)

names(rotating_link) <- c("time",
  "theta",
  "d.theta",
  "dd.theta",
  "net.torque",
  "accel.x",
  "accel.y",
  "pos.x",
  "pos.y",
  "pos.p.x",
  "pos.p.y",
  "friction.force",
  "normal.force",
  "friction.coeff",
  "ke.rot",
  "ke.trans")
```

Let's import some visualization tools, etc.

```
library(tidyverse)
```

Let's first see the head of this table:

```
head(rotating_link)
```

```
1e-05 4.16365149517594e-09 0.000277576766268729 13.8788383019864 4.62627943399546 -2.88933229236853e-08
6.9394191509932 0.5 2.08182574758797e-09 0.5 2.08182574758797e-09 5.50000002889332 7.21313970936437
0.762497365987937 1.28414768620341e-08 9.63110764652555e-09
2e-05 8.32730298348184e-09 0.000416365148830591 13.8788382561862 4.62627941872874 -5.77866456090008e-08
6.93941912809311 0.5 4.16365149174092e-09 0.5 4.16365149174092e-09 5.50000005778664 7.21313968646429
0.762497372414344 2.889332286012e-08 2.166999214509e-08
3e-05 1.38788382905364e-08 0.000555153530705451 13.878838187486 4.62627939582866 -9.63110754323193e-08
6.93941909374299 0.5 6.93941914526818e-09 0.5 6.93941914526818e-09 5.50000009631107 7.21313965211416
0.762497381386347 5.13659071091213e-08 3.8524430331841e-08
4e-05 2.08182574071794e-08 0.000693941911664307 13.8788380958856 4.62627936529521 -1.44466611996358e-07
6.93941904794282 0.5 1.04091287035897e-08 0.5 1.04091287035897e-08 5.50000014446661 7.21313960631399
0.762497392903945 8.02592294607188e-08 6.01944220955391e-08
5e-05 2.9145560321961e-08 0.000832730291478159 13.8788379813852 4.62627932712841 -2.02253254792593e-07
6.93941899069261 0.5 1.45727801609805e-08 0.5 1.45727801609805e-08 5.50000020225325 7.21313954906379
0.762497406967139 1.15573289724217e-07 8.66799672931625e-08
```

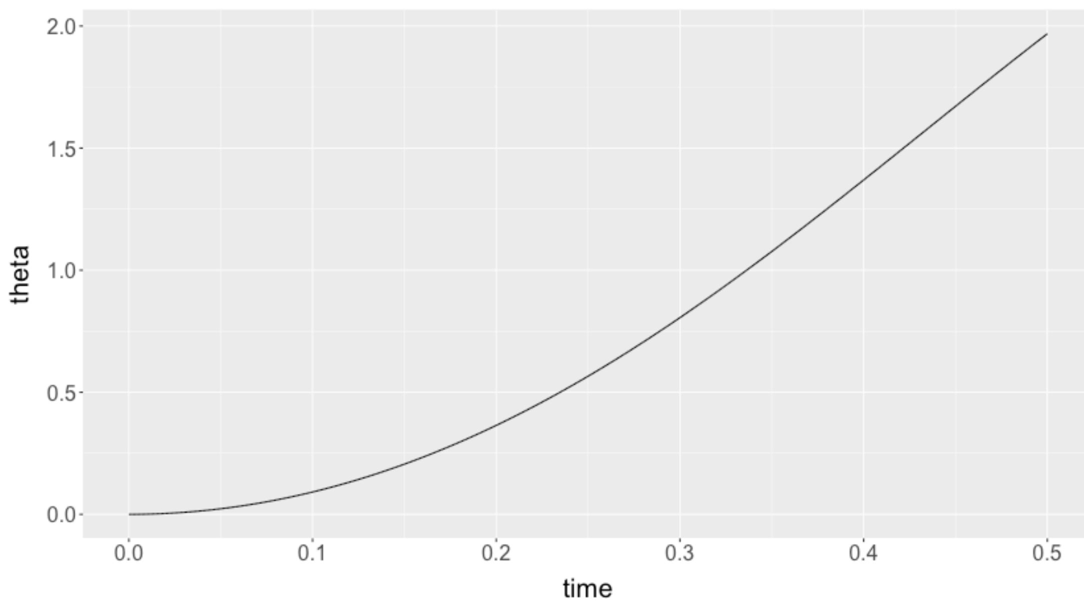
```
6e-05 3.88607470211411e-08 0.000971518669918007 13.8788378439847 4.62627928132824 -2.69671003201265e-07
6.93941892199236 0.5 1.94303735105705e-08 0.5 1.94303735105705e-08 5.500000269671 7.21313948036353
0.762497423575928 1.57308087666542e-07 1.17981065749907e-07
```

Before we start graphing, let's set a common graph theme.

```
default.theme <- theme(text = element_text(size=20), axis.title.y = element_text(margin = margin(t = 0,
```

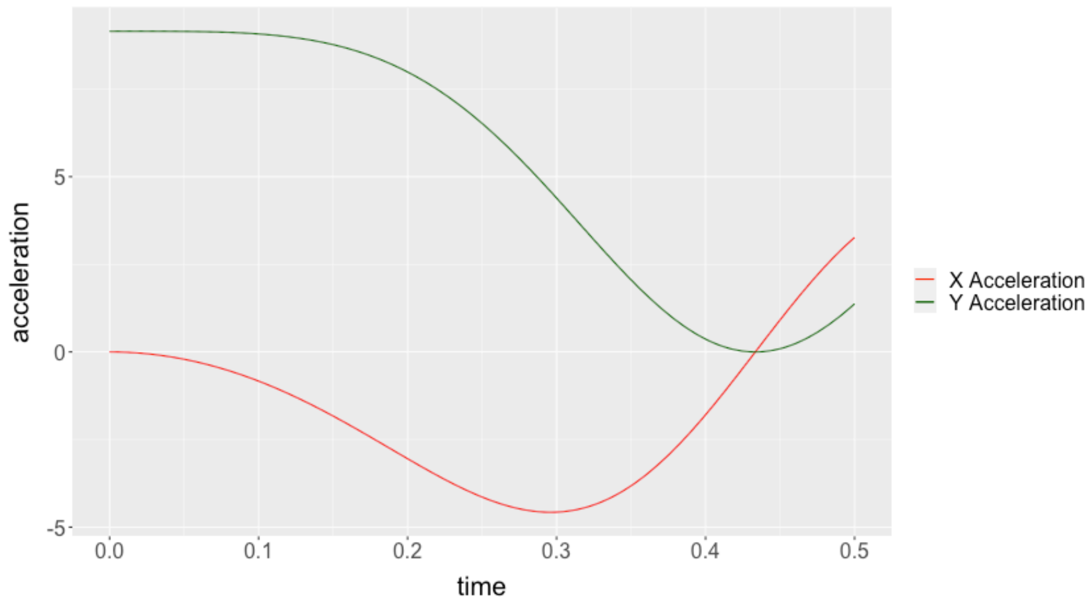
Cool! We could first graph a function for theta over time.

```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=theta)) + default.theme
```



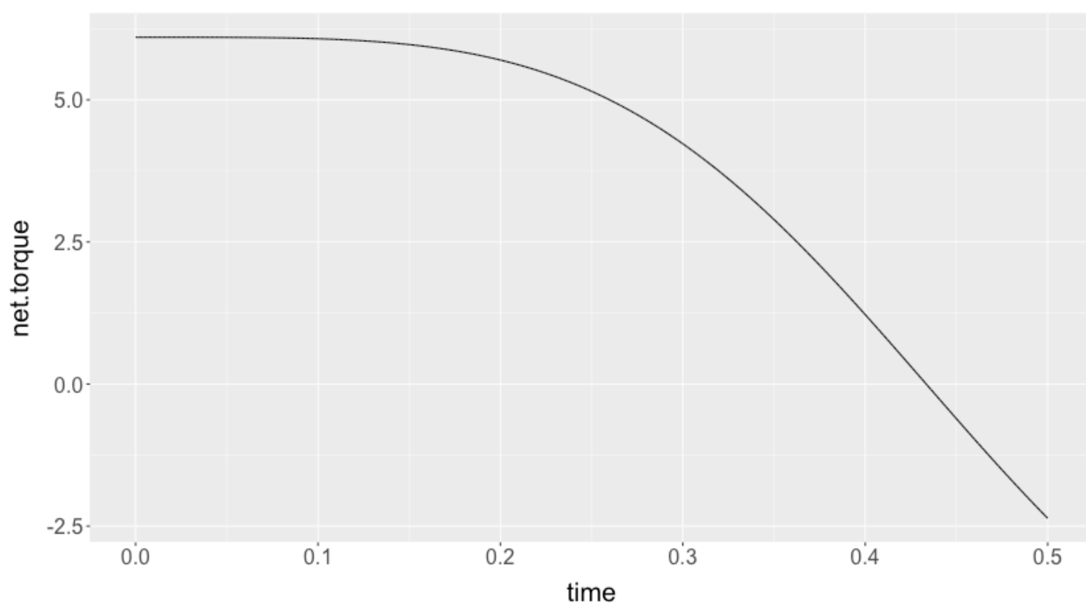
And, similarly, we will graph `ax` and `ay` on top of each other:

```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=accel.x, colour="X Acceleration")) + geom_line(aes
```



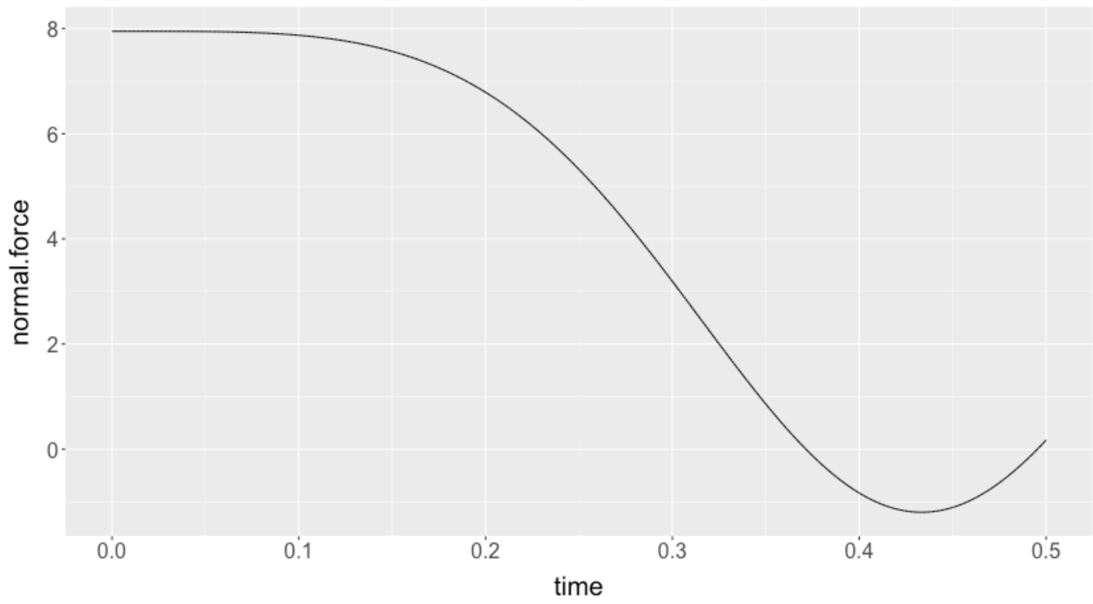
Let's also plot torque as well.

```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=net.torque)) + default.theme
```



And. **Most importantly!** Let's plot the normal force.

```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=normal.force)) + default.theme
```

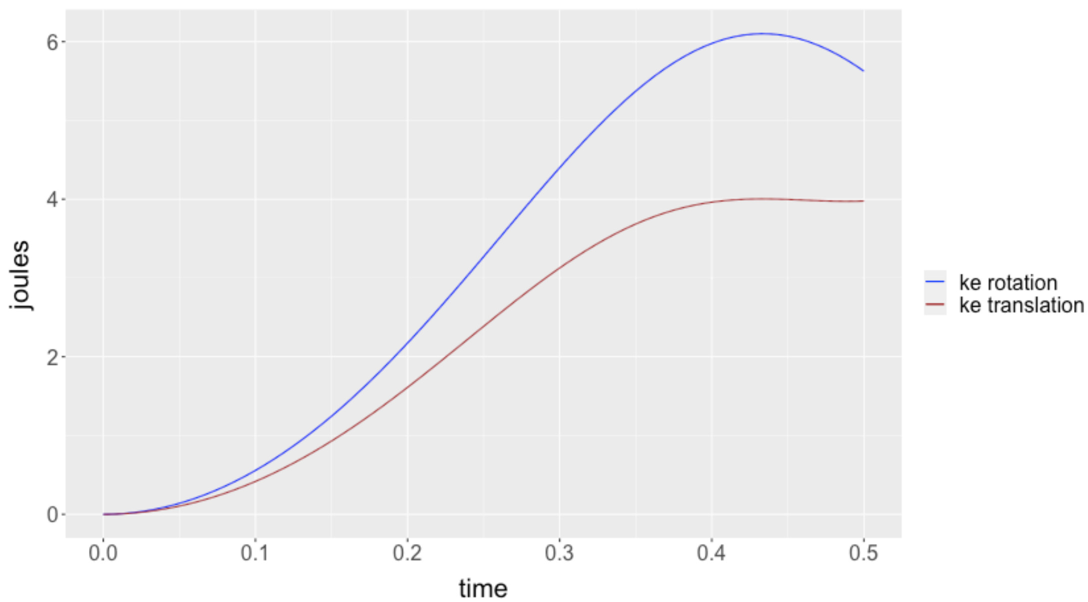


Obviously, after the normal force becomes negative, this graph stops being useful.

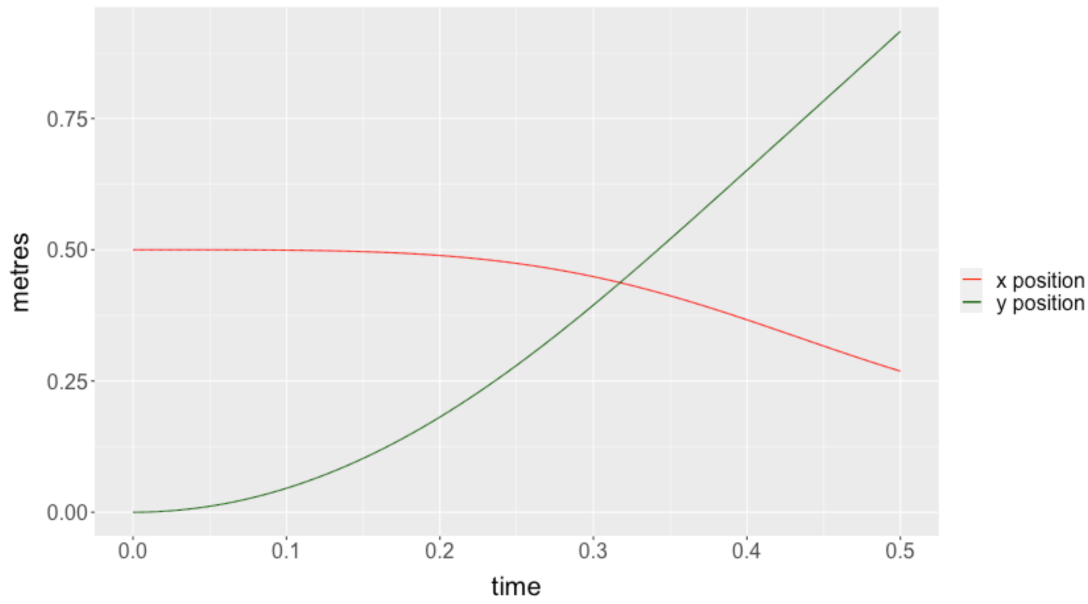
Theta dot atop theta:

We finally, plot KE rotation and translation

```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=ke.rot, colour="ke rotation")) + geom_line(aes(x=ti
```

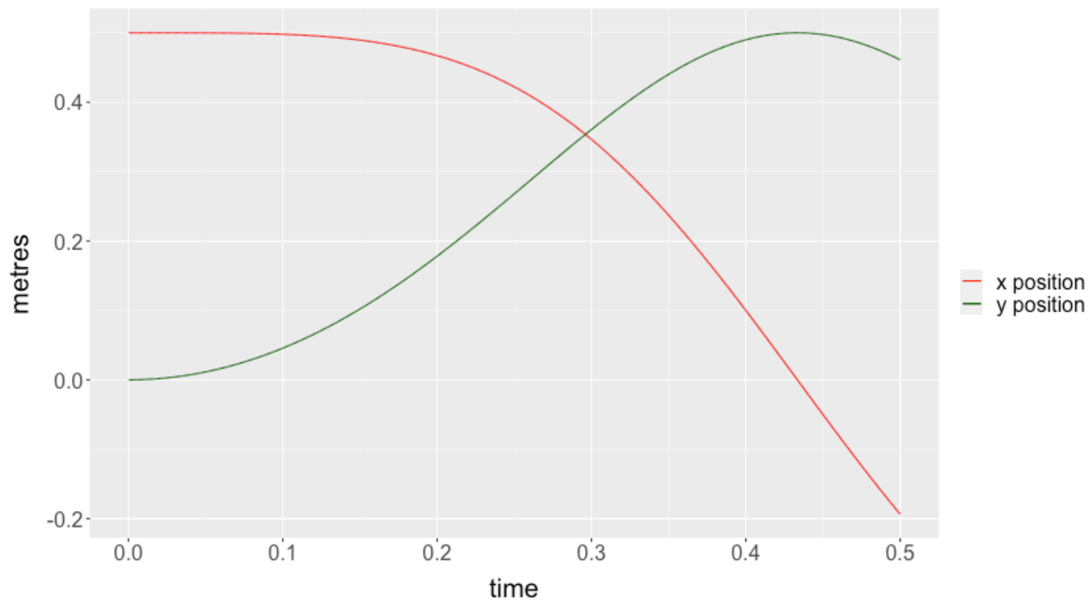



```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=pos.x, colour="x position")) + geom_line(aes(x=time
```

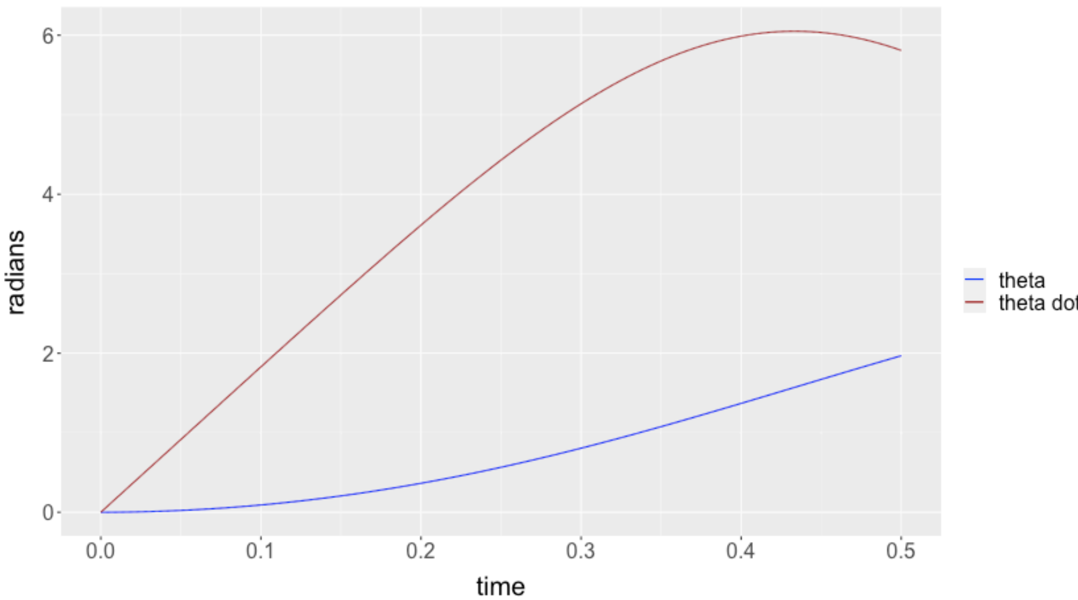


floor

```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=pos.p.x, colour="x position")) + geom_line(aes(x=t
```




```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=theta, colour="theta")) + geom_line(aes(x=time, y=
```




```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=dd.theta, colour="thetadd")) + scale_colour_manual
```