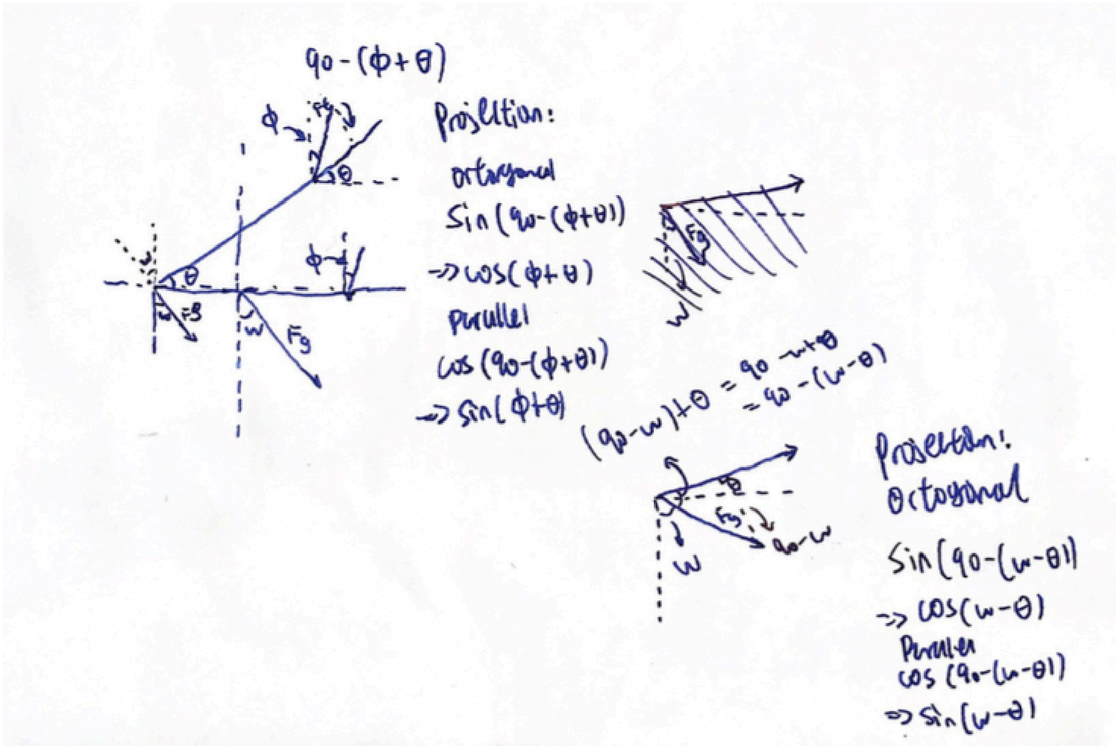
Let's draw a picture of this situation!




```
t, a = var("t a")
solve(t/cos(t) == a, t)
```

```
[t == a*cos(t)]
```

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.

```
t,b = var("theta b")
solve(sin(90+t) == cos(b), b)

[b == arccos(sin(theta + 90))]

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 5.49e-09 0.000366 18.3 6.1 -5.02335e-08 9.15 0.5 2.745e-09 0.5 2.745e-09 5.02335e-08
7.95 6.31867924528302e-09 2.2326e-08 1.67445e-08
```

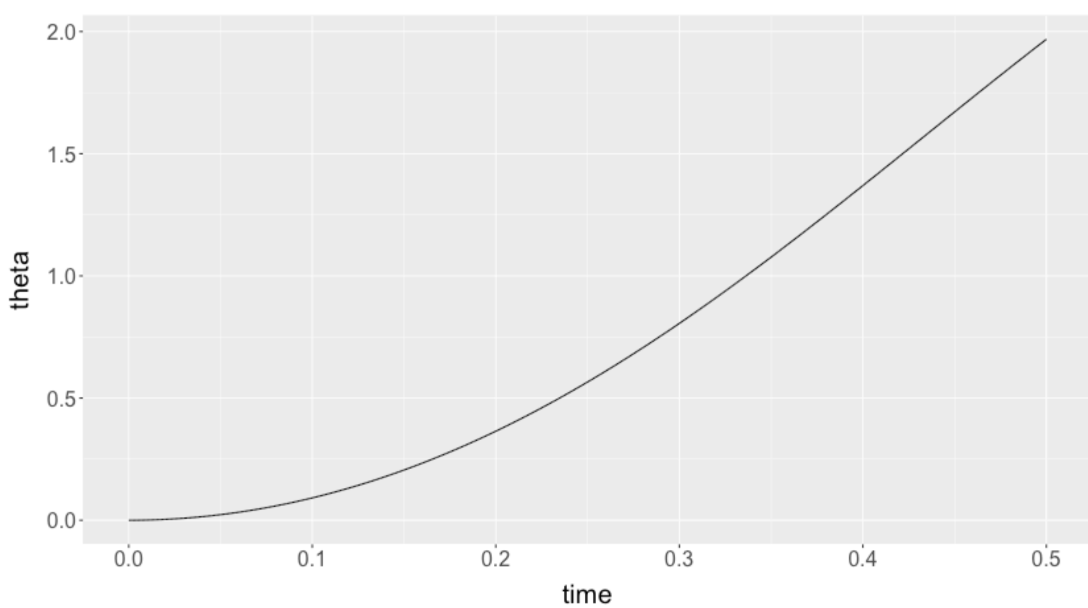
```
2e-05 1.098e-08 0.000549 18.3 6.1 -1.00467e-07 9.15 0.5 5.49e-09 0.5 5.49e-09 1.00467e-07 7.95
1.2637358490566e-08 5.02335e-08 3.7675125e-08
3e-05 1.83e-08 0.000732 18.3 6.1 -1.67445e-07 9.15 0.5 9.15e-09 0.5 9.15e-09 1.67445e-07 7.95
2.10622641509434e-08 8.9304e-08 6.6978e-08
4e-05 2.745e-08 0.000915 18.3 6.1 -2.511675e-07 9.15 0.5 1.3725e-08 0.5 1.3725e-08 2.511675e-07
7.95 3.15933962264151e-08 1.395375e-07 1.04653125e-07
5e-05 3.843e-08 0.001098 18.3 6.1 -3.516345e-07 9.14999999999999 0.5 1.9215e-08 0.5 1.9215e-08
3.516345e-07 7.94999999999999 4.42307547169812e-08 2.00934e-07 1.507005e-07
6e-05 5.124e-08 0.001281 18.3 6.09999999999999 -4.68845999999999e-07 9.14999999999998 0.5 2.562e-08
0.499999999999999 2.562e-08 4.68845999999999e-07 7.94999999999998 5.89743396226416e-08 2.734935e-07
2.05120125e-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,
```

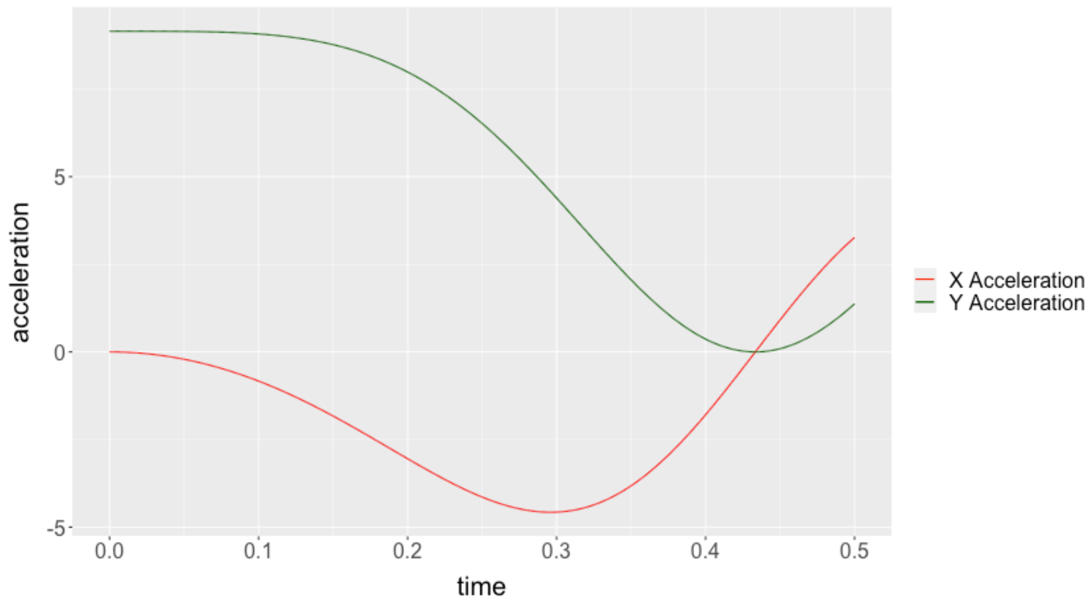
Cool! We could first graph a function for theta over time.

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



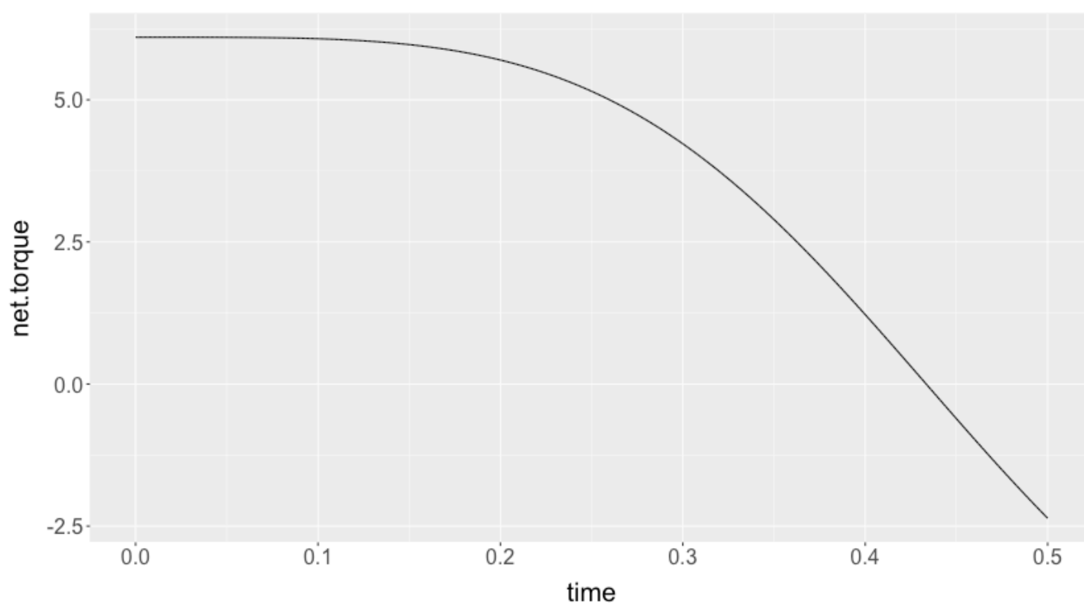
And, similarly, we will graph `ax` and `ay` on top of each other:

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



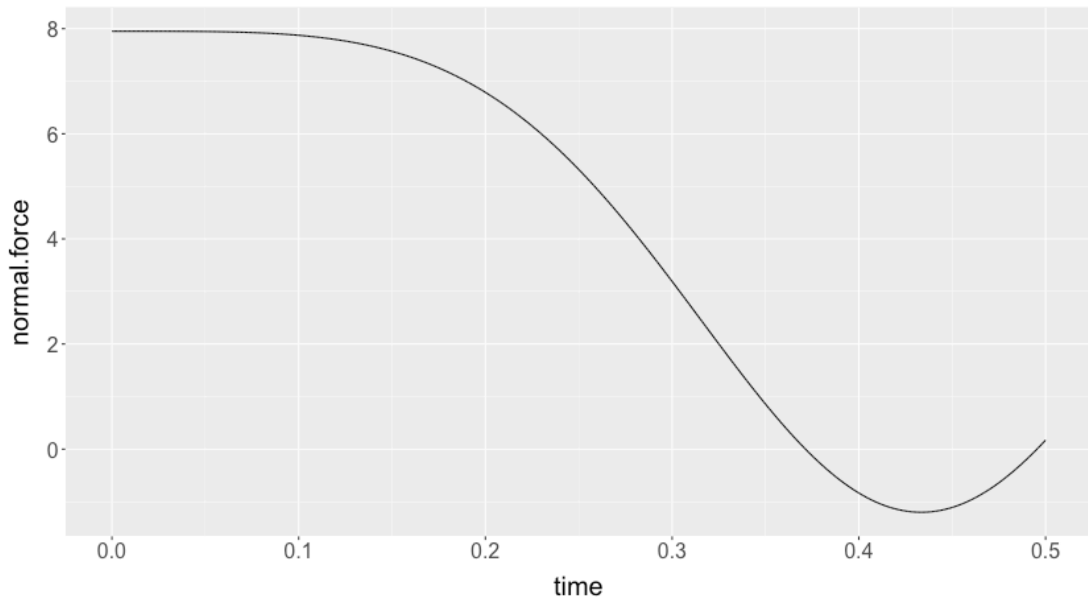
Let's also plot torque as well.

```
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
```

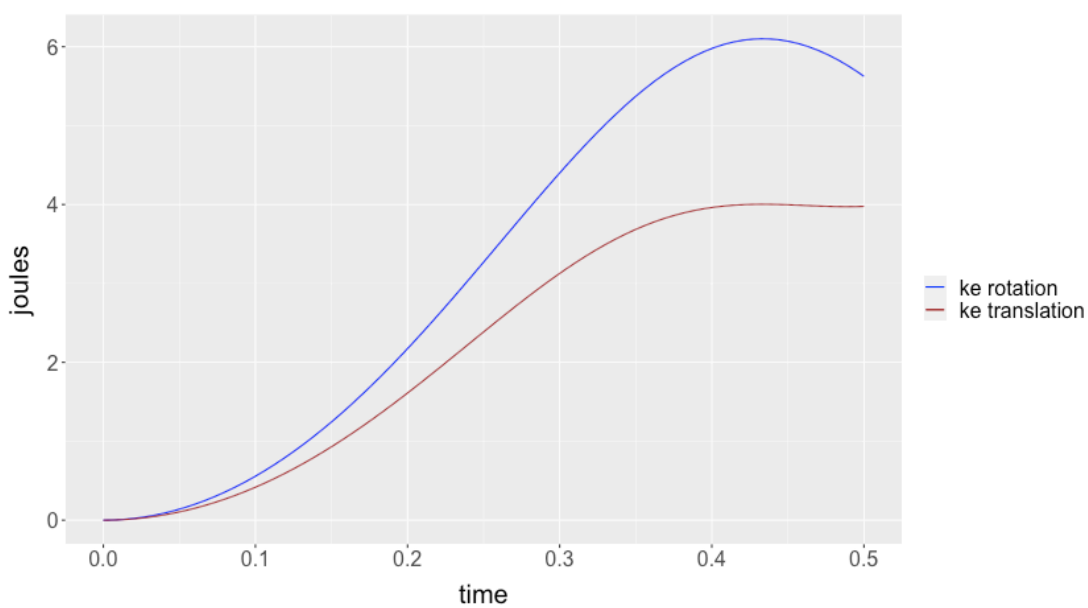


Obviously, after the normal force becomes negative, this graph stops being useful.

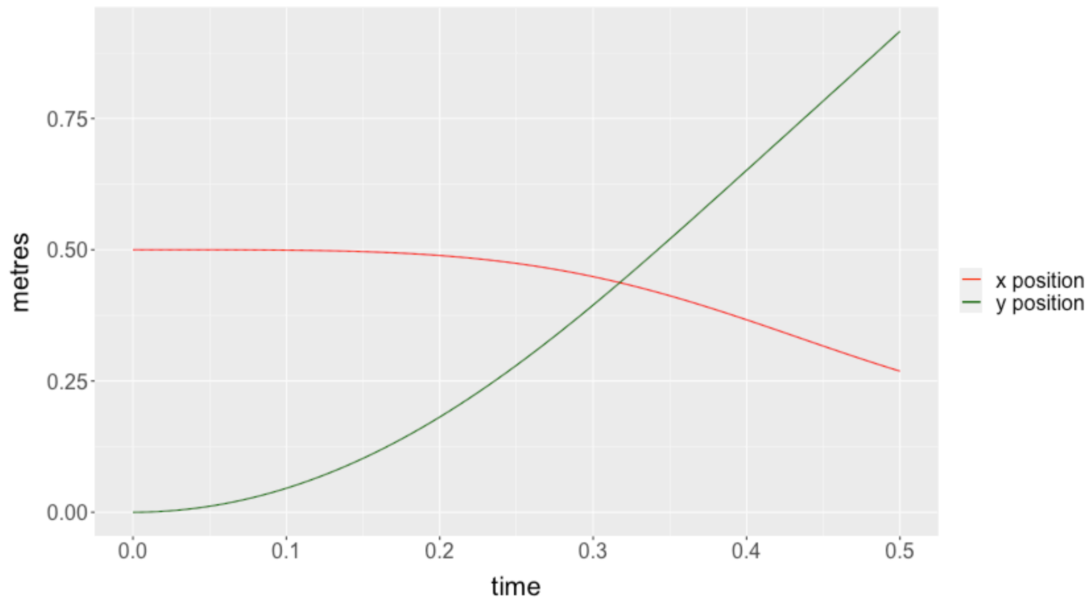
Theta dot atop theta:

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
```

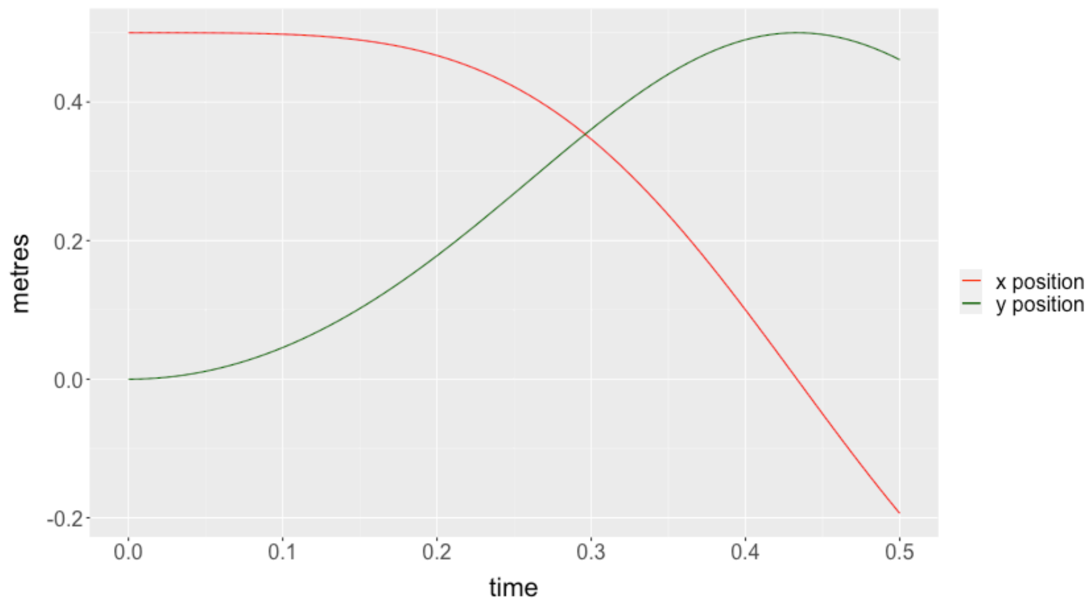



```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=pos.x, colour="x position")) + geom_line(aes(x=time
```

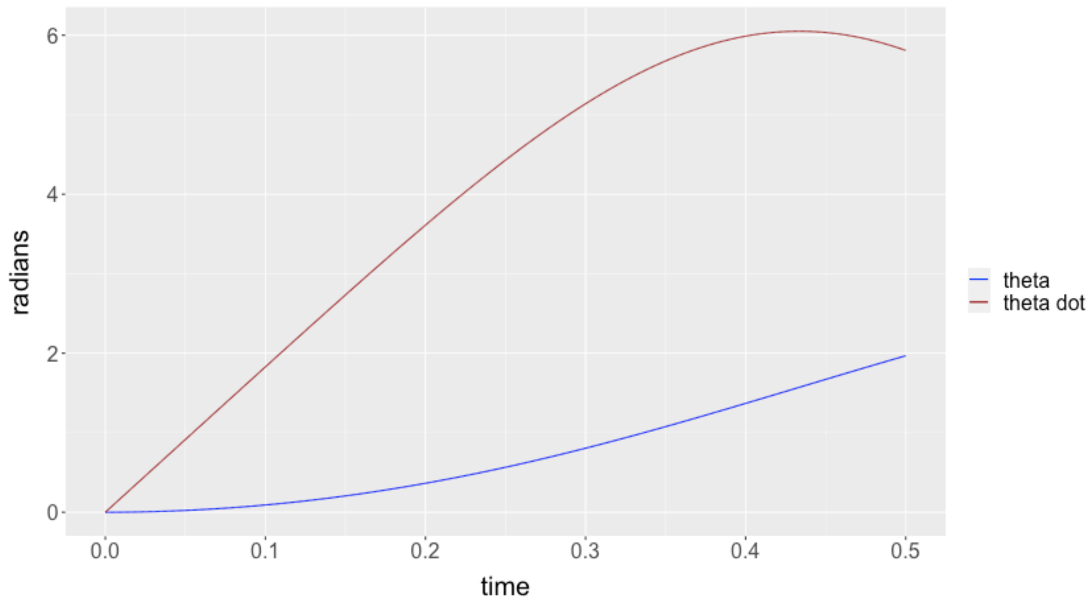


floor

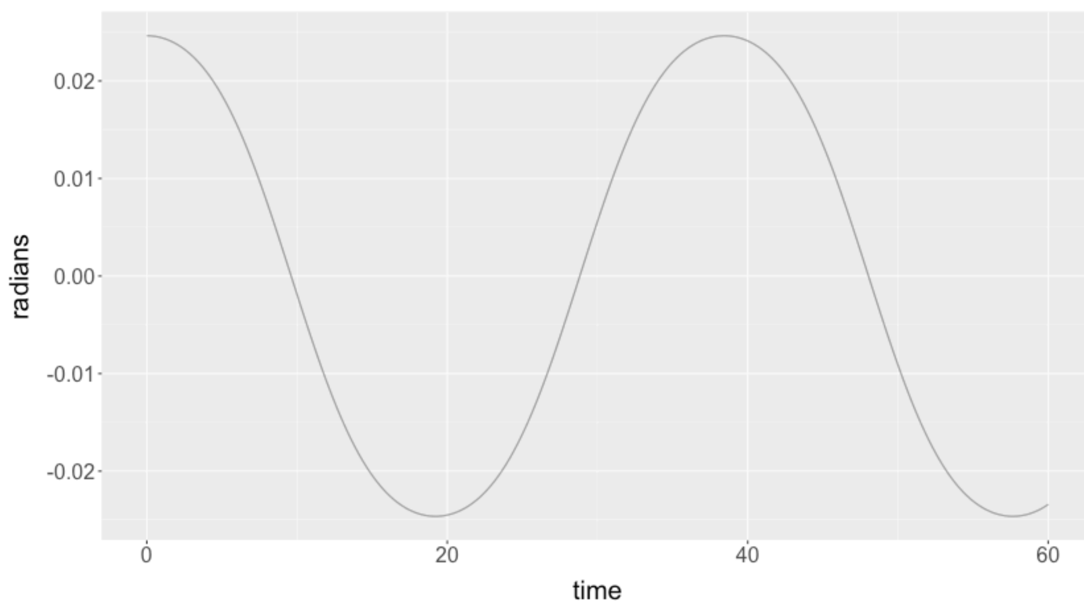
```
rotating_link %>% ggplot() + geom_line(aes(x=time, y=pos.p.x, colour="x position")) + geom_line(aes(x=t
```




```
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
```




```
write.csv(rotating_link, "./chainrot_table.csv", row.names = FALSE)
```