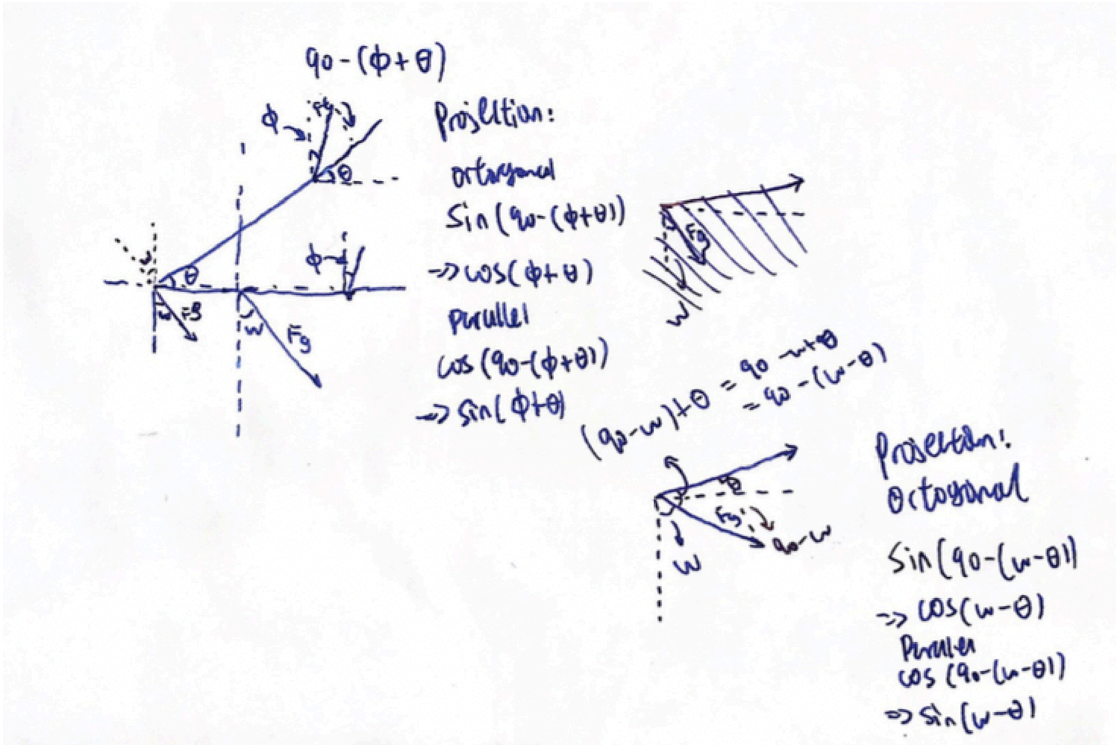
Let's draw a picture of this situation!



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 <- pi/6 # angle of Ft relative to floor (parallel) (rad)
FT <- 12 # tension force (N)
OMEGA <- 0.1 # 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 <- 0.0001
t_max <- 5

vx <- 0
vy <- 0

x <- 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
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*cos(theta))^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
    ax <- (-1*sin(theta)*thetadot^2+cos(theta)*thetadotdot)*L1
    cAccelX[i] = ax
```

```
    ## ay <- L1 * cos(theta) * thetadotdot
    cAccelY[i] = ay # @mark isn't sin and cos backwards?
    # 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,
    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",
  "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)
```

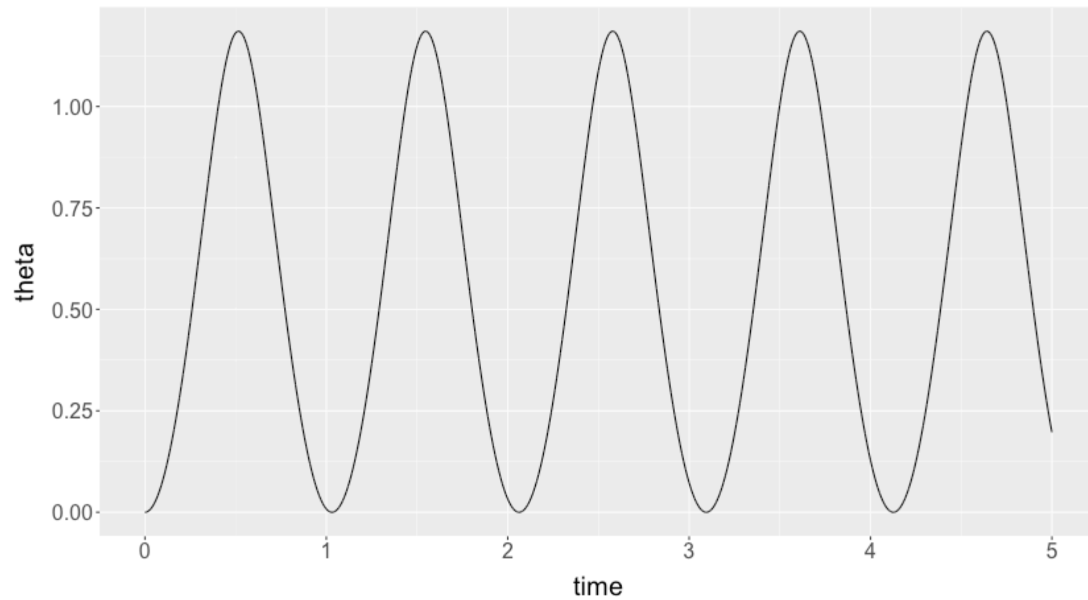
```
1e-04 1.65503533066528e-07 0.00331007033913572 16.5503500847044 5.51678336156803 -1.36957070625324e-06
8.27517504235209 -1.36957070625324e-14 2.48255283490049e-07 6.97836885270962 7.63391101666348
0.91412761263225 1.82609427500431e-06 1.36957070625325e-06
2e-04 4.965105669801e-07 0.00496510470321662 16.5503436408089 5.51678121360196 -4.1087102524066e-06
8.27517182040345 -6.84785166491308e-14 4.96510518650869e-07 6.97837159184917 7.63390779471484
0.914128357258976 4.10871078564987e-06 3.08153308923784e-06
3e-04 9.93021037301762e-07 0.00662013810071352 16.550333974969 5.51677799165225 -8.21741490575579e-06
8.27516698748041 -2.05435475293287e-13 8.27517423686492e-07 6.97837570055382 7.6339029617918
0.914129474199641 7.30437141208106e-06 5.47827855906406e-06
4e-04 1.65503484737311e-06 0.00827517020943236 16.5503210871885 5.51677369571816 -1.36956790672492e-05
8.2751605435829 -4.79349224609936e-13 1.24127593415794e-06 6.97838117881798 7.6338965178943
0.914130963454935 1.14130736658227e-05 8.55980524938151e-06
5e-04 2.48255186831635e-06 0.00993020070717963 16.5503049774726 5.51676832579872 -2.0543495271494e-05
8.27515248871082 -9.58697926641525e-13 1.73778596951651e-06 6.97838802663419 7.63388846302222
0.914132825025777 1.64348143474025e-05 1.23261107605995e-05
6e-04 3.47557193903431e-06 0.0115852292717624 16.550285645828 5.5167618818927 -2.87608541867632e-05
8.27514282286404 -1.72565517054075e-12 2.31704743310371e-06 6.9783962439931 7.63387879717543
0.91413505891332 2.23695895463475e-05 1.67771921598887e-05
```

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,
```

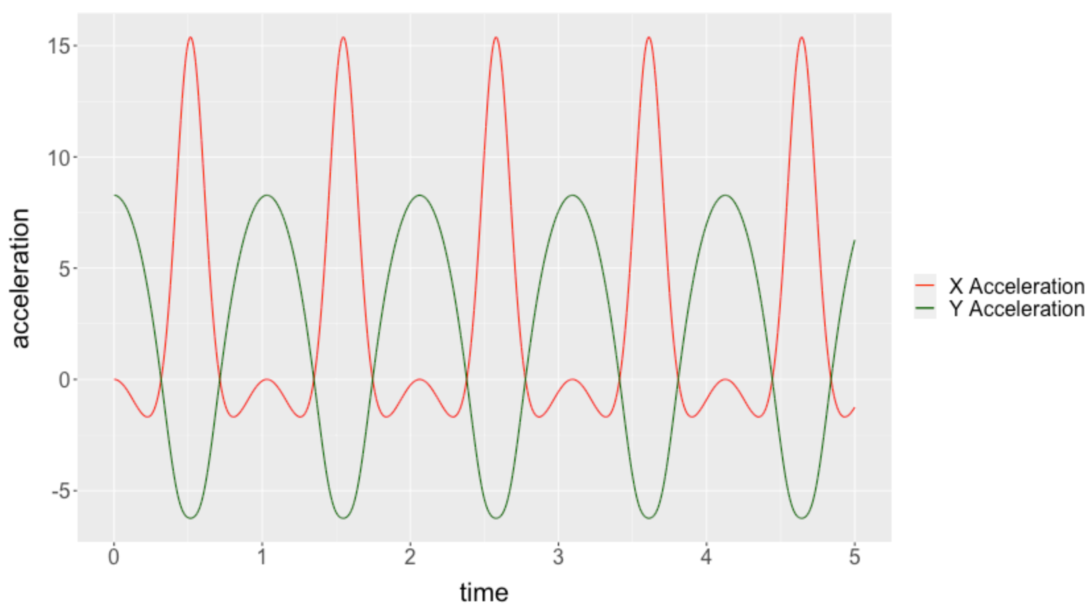
Cool! We could first graph a function for theta over time.

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



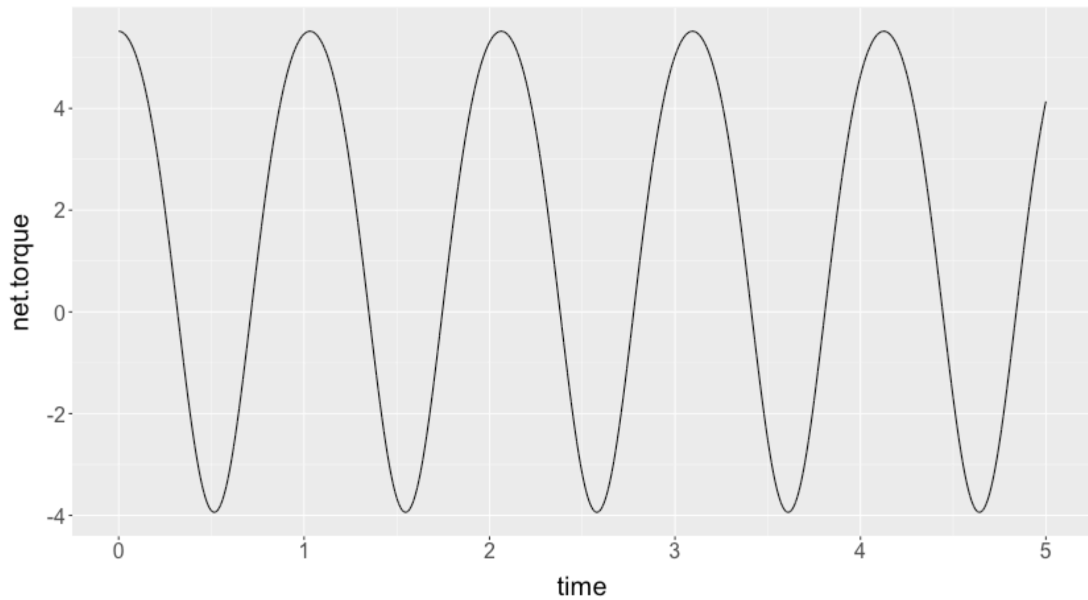
And, similarly, we will graph `ax` and `ay` on top of each other:

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



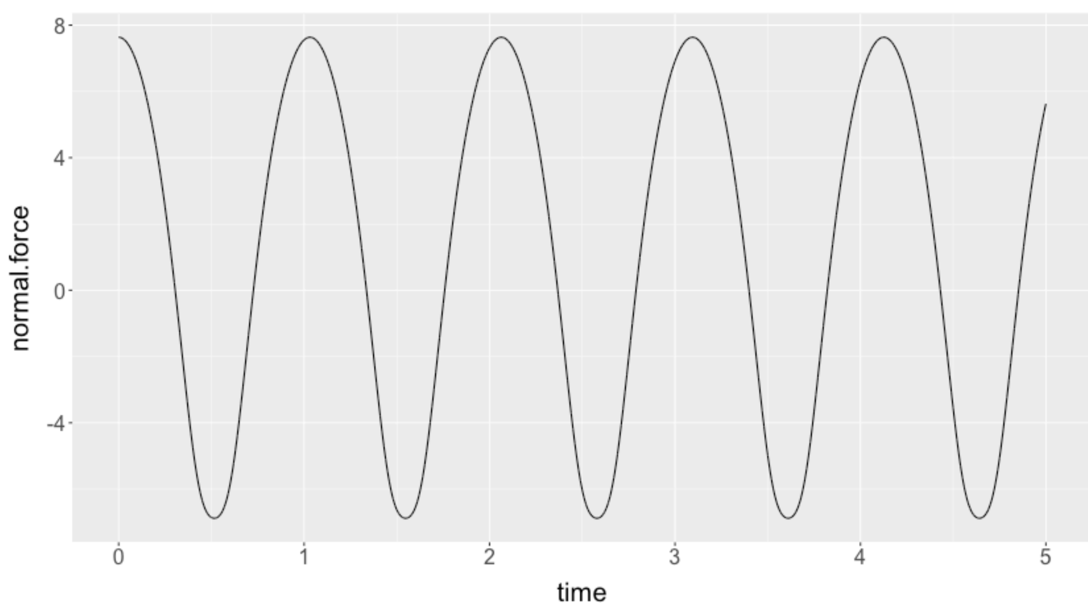
Let's also plot torque as well.

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



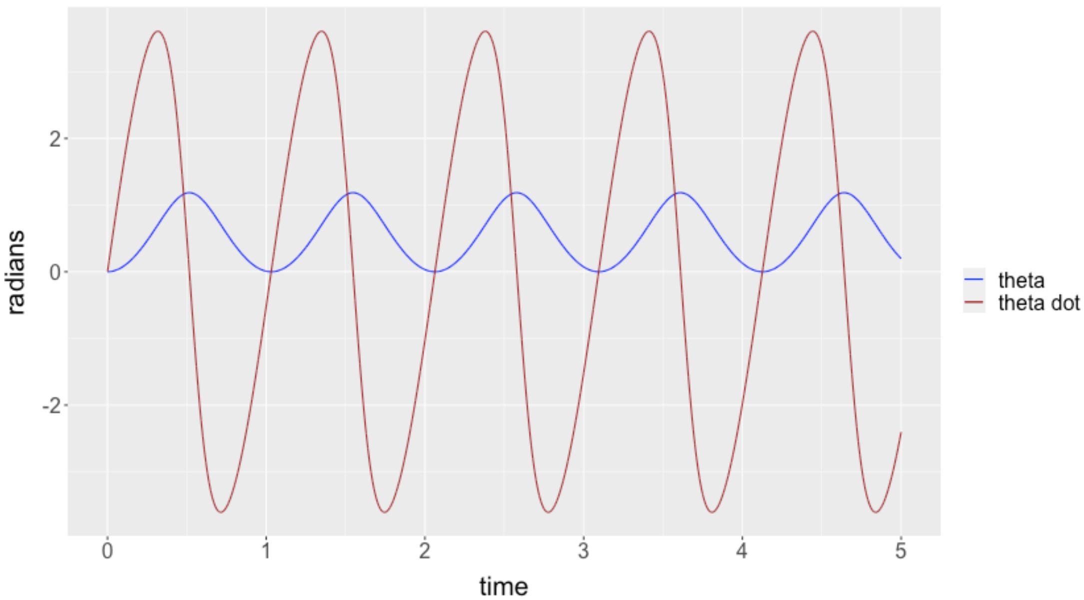
And. **Most importantly!** Let's plot the normal force.

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



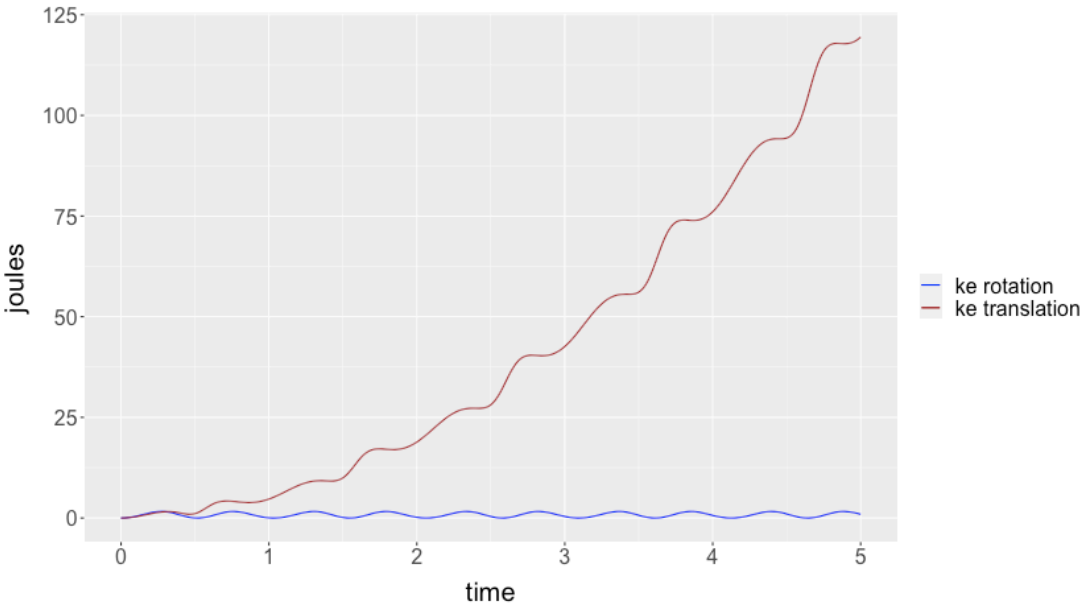
Obviously, after the normal force becomes negative, this graph stops being useful.
Theta dot atop theta:

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

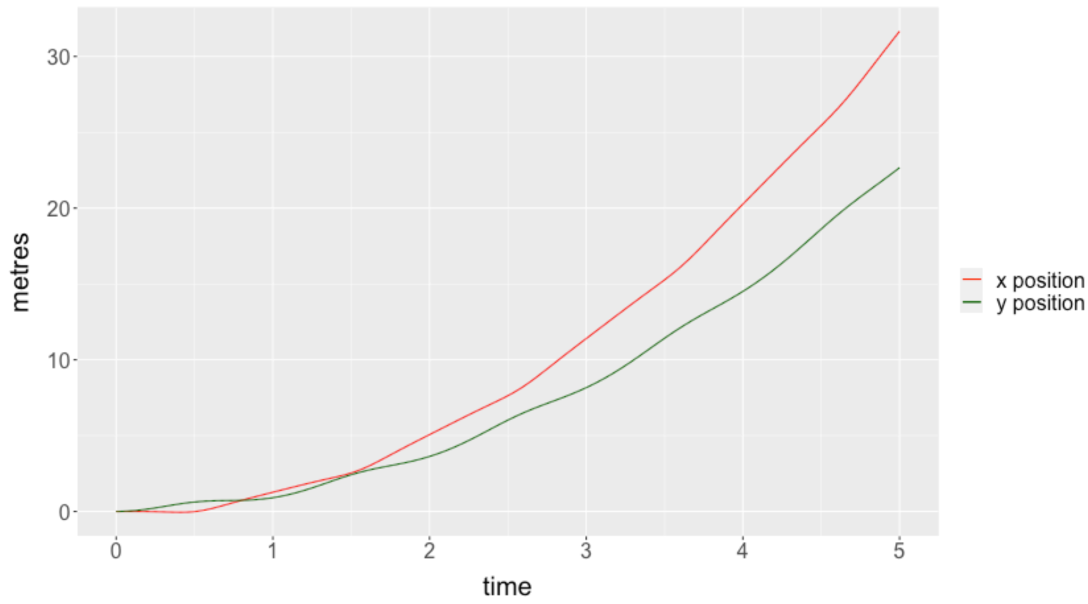


We finally, plot KE rotation and translation

```
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
```


30
20
10
0
metres
0
1
2
3
4
5
time
x position
y position