About this code

This code simulates sarcomere quick-release experiments based on the 4-state crossbridge model presented in "Dynamics of cross-bridge cycling, ATP hydrolysis, force generation, and deformation in cardiac muscle" Tewari et al. (2016) J Mol Cell Cardiol. 96:11-25,

This document supplements the mathematical formulation of the model of Tewari et al. (2016) and serves as a guide to understanding the associated computer code. For a complete description of the model, please refer to the original manuscript.

Disclaimer: This code is free to use, edit, reproduce as long as the source is cited.

Specifically, this code produces Figures 4 and 6 of the above paper (with slight modifications):

For additional documentation please visit http://virtualrat.org/publications/dynamics-cross-bridge-cycling (http://virtualrat.org/publications/dynamics-cross-bridge-cycling)

Original code for the model is Written in MATLAB by: Shivendra Tewari E-mail: TewariSG@gmail.com

The code was modified and rewritten in python by: Bahador Marzban E-mail: bmarzban@med.umich.edu

Overall structure of model

 title

F_{XB} is the active force generated by cross-bridges, F_1 and F_2 are passive forces associated with the muscle, and F is the applied force. Overall force balance for the model yields

$$F\left(t\right) = F_{XB}\left(t\right) + F_1\left(t\right) + F_2\left(t\right)$$

(1)

The model has no mass (initial terms).

Active and passive contributions to the force:

The model for active force (F_{XB}) force due to cross bridge is

$$F_{XB}\left(t\right) = \ k_{stiff,1}\left(\int_{-\infty}^{+\infty} sp_2\left(t,s\right)ds + \int_{-\infty}^{+\infty} sp_3\left(t,s\right)ds\right) + \ k_{stiff,2}\,\Delta r\int_{-\infty}^{+\infty} p_3\left(t,s\right)ds$$

(2)

where $k_{stiff,1}$ and $k_{stiff,2}$ are effective stiffness constants, Δr is the cross bridge strain associated with ratcheting deformation, and s is an independent variable representing strain in the population of attached cross-bridge states. Thus, the integrals in Equation (2) represent the following quantities

$$\frac{\int_{-\infty}^{+\infty} sp_2\left(t,s\right)ds}{\int_{-\infty}^{+\infty} p_2\left(t,s\right)ds} = \text{average strain of attached cross bridges in model state 2}$$

$$\frac{\int_{-\infty}^{+\infty} sp_3\left(t,s\right)ds}{\int_{-\infty}^{+\infty} p_3\left(t,s\right)ds} = \text{average strain of attached cross bridges in model state 3}$$

$$\int_{-\infty}^{+\infty} p_3\left(t,s\right)ds = \text{fraction of cross bridges in model state 3.}$$

The spring and dashpot (F_1 and η)) represent a linear Maxwell viscous model, governed by

$$\frac{dF_1}{dt} = K_{PE,1}\left(\frac{dL}{dt} - \frac{F_1}{\eta}\right)$$

(3)

==========

\ How to get to Equation (3):

Consider the F1 branch in Fig. 1 that includes the dashpot (damper) and the spring.

Since the dashpot and the spring are in series, the forces transmitted through both elements are equal

$F_1 = \ F_\eta = \ F_{K_{PE,1}}$. (i)

Total strain ΔL is equal to sum of the strains $\Delta L = \Delta L_{K_{PE,1}} + \Delta L_{\eta}$. (ii)

For the spring and the dashpot (viscous element) we can write the force-displacement and the force-rate of displacement equations as following:

$F_{K_{PE,1}} = K_{PE,1}\ \Delta L_{K_{PE,1}}$ (iii)

$F_{\eta} = \eta \frac{\mathrm{d}L_{\eta}}{dt}$ (iv)

Now we obtain the rate of the change in lengths (L):

(the derivative of Equation (ii) and (iii) w.r.t time (t))

$\frac{\mathrm{d}L}{dt} = \frac{\mathrm{d}L_{K_{PE,1}}}{dt} + \frac{\mathrm{d}L_{\eta}}{dt}$ (v)

$\frac{\mathrm{d}L_{K_{PE,1}}}{dt} = \frac{1}{K_{PE,1}} \frac{\mathrm{d}F_{K_{PE,1}}}{dt}$ (vi)

Recalling Equations (i), (iv), and (vi), we can rewrite Equation (v):

$\frac{\mathrm{d}L}{dt} = \frac{1}{K_{PE,1}} \frac{\mathrm{d}F_1}{dt} + \frac{F_1}{\eta}$ (vii)

Rearranging equation (vii) we get Equation (3):

$\frac{dF_1}{dt} = K_{PE,1} \left(\frac{dL}{dt} - \frac{F_1}{\eta} \right)$ (viii)

==========

F_2 is a nonlinear force:

$$F_2\ (L) = \begin{cases} \beta . K_{PE,1}\ \left[e^{(PExp_{titin}(L - SL_{rest}))} - 1 \right] & if\ L > SL_{rest} \\ -\beta . K_{PE,1}\ \left[e^{(PExp_{titin}(SL_{rest} - L))} - 1 \right] & if\ L < SL_{rest} \end{cases}$$

(4)

Where L is a sarcomere length.

To obtain a governing equation for length as a function of time, we rearrange Equation (3):

$$\frac{dL}{dt} = \frac{1}{K_{PE,1}} \left(\frac{dF_1}{dt} \right) + \frac{F_1}{\eta}$$

(5)

Substituting $F_1 = F - F_{XB} - F_2$ from Equation (1) we have

$$\frac{dL}{dt} = \frac{\dot{F} - \dot{F}_{XB} - \dot{F}_2}{K_{PE,1}} + \frac{F - F_{XB} - F_2}{\eta}.$$

(6)

To simulate muscle dynamics according to Equation (6), we need to calculate $\dot{F}_{XB}$ taking the time derivative of the Equation (2):

$$\dot{F}_{XB} = \frac{dF_{XB}}{dt} = k_{stiff,1}\frac{d}{dt}\left(\int_{-\infty}^{+\infty} sp_2\,(t,s)\,ds + \int_{-\infty}^{+\infty} sp_3\,(t,s)\,ds\right) + k_{stiff,2}\,\Delta r\frac{d}{dt}\int_{-\infty}^{+\infty} p_3\,(t,s)\,ds$$

(7)

Using the moment definitions, this equation is written

$$\dot{F}_{XB} = k_{stiff,1}\left(p_2^1 + p_3^1\right) + k_{stiff,2}\,\Delta r(p_3^0).$$

(8)

Expressions for p_3^0 , p_2^1 and p_3^1 are obtained from Equation (11) in the paper:

$$\frac{dp_2^1}{dt} = v\,p_2^0 + \widetilde{k_1}\left(p_1^1 - \alpha_1 p_1^2\right) - k_{-1}\left(p_2^1 + \alpha_1 p_2^2\right) - k_2\left(p_2^1 - \alpha_2 p_2^2\right) + k_{-2}\left(p_3^1\right).$$

$$\frac{dp_3^1}{dt} = v\,p_3^0 + k_2\left(p_2^1 - \alpha_2 p_2^2\right) - k_{-2}\left(p_3^1\right) - k_3\left(p_3^1 - \alpha_3 s_3^2 p_3^1 + 2\alpha_3 s_3 p_3^2\right)$$

$$\frac{dp_3^0}{dt} = k_2\left(p_2^0 - \alpha_2 p_2^1 + 0.5 * \alpha_2^2 p_2^2\right) - k_{-2}\left(p_3^0\right) - k_3\left(p_3^0 - \alpha_3 s_3^2 p_3^0 + 2\alpha_3 s_3 p_3^1 + p_3^2\right)$$

(9)

(Note that for $i > 2$ we assume p_k^i = 0.)

Substituting Equations (9) into Equation (8):

$$\dot{F}_{XB} = k_{stiff,1} * \left(v\,p_2^0 + v\,p_3^0 + \widetilde{k_1}\left(p_1^1 - \alpha_1 p_1^2\right) - k_{-1}\left(p_2^1 + \alpha_1 p_2^2\right) + k_3\left(p_3^1 - \alpha_3 s_3^2 p_3^1 + 2\alpha_3 s_3 p_3^2\right)\right)$$
$$+ k_{stiff,2}\,\Delta r\left(k_2\left(p_2^0 - \alpha_2 p_2^1 + 0.5 * \alpha_2^2 p_2^2\right) - k_{-2}\left(p_3^0\right) - k_3\left(p_3^0 - \alpha_3 s_3^2 p_3^0 + 2\alpha_3 s_3 p_3^1 + p_3^2\right)\right)$$

(10)

Recalling the velocity of sliding $\nu = \frac{dL}{dt}$.

We can cast the right-hand side of Equation (10) as having a velocity dependent term and velocity independent term:

$$\dot{F}_{XB} = A_{XB} + B_{XB}\frac{dL}{dt}$$

(11)

where the velocity-dependent term is

$$B_{XB} = k_{stiff,1}\left(p_2^0 + p_3^0\right).$$

(12)

Taking the derivative of F_2 with respect to time, we have

$$\dot{F}_2 = \frac{\partial F_2}{\partial L}\frac{dL}{dt} = (\beta.\ k_{PE,2})\,(PExp_{titin})\left[e^{(PExp_{titin}(L - SL_{rest}))}\right]\frac{dL}{dt}$$

13)

Next, substituting Equation (11) into the Equation (6) yields

$$\frac{dL}{dt} = \frac{1}{k_{PE,1}}\left(\dot{F} - A_{XB} - B_{XB}\frac{dL}{dt} - \frac{\partial F_2}{\partial L}\frac{dL}{dt}\right) + \frac{1}{\eta}(F - F_{XB} - F_2).$$

(14)

Rearranging we have

$$\frac{dL}{dt}\left(1+\frac{B_{XB}}{k_{PE,1}}+\frac{1}{k_{PE,1}}\frac{\partial F_2}{\partial L}\right)=\frac{1}{k_{PE,1}}\left(\dot{F}-A_{XB}\right)+\frac{1}{\eta}(F-F_{XB}-F_2)\,,$$

or

$$\frac{dL}{dt}=\frac{\frac{1}{k_{PE,1}}\left(\dot{F}-A_{XB}\right)+\frac{1}{\eta}(F-F_{XB}-F_2)}{\left(1+\frac{B_{XB}}{k_{PE,1}}+\frac{1}{k_{PE,1}}\frac{\partial F_2}{\partial L}\right)}.$$

(15)

Simulation of quick-release experiment:

To simulate the quick-release experiment, the internal states of the cross bridges and the sarcomere length kinetics are simulated by integrating Equation (15) in parallel with Equation (3)-(6) from the paper, governing the cross-bridge kinetics:

$$\frac{dP}{dt}=k_{np}N\left(t\right)-k_{pn}P\left(t\right)+\tilde{k}_d\hat{p}_1\left(t,s\right)-\ k_aP\left(t\right)+\tilde{k}_3e^{\alpha_3\left(s+s_3\right)^2}p_3\left(t,s\right)$$

$$\frac{\partial p_1}{\partial t}+\frac{dL}{dt}\frac{\partial p_1}{\partial s}=k_a\delta\left(s\right)P\left(t\right)-\tilde{k}_dp_1-\tilde{k}_1e^{-\alpha_1s}p_1+k_{-1}e^{+\alpha_1s}p_2$$

$$\frac{\partial p_2}{\partial t}+\frac{dL}{dt}\frac{\partial p_2}{\partial s}=\tilde{k}_1e^{-\alpha_1s}p_1-k_{-1}e^{+\alpha_1s}p_2-\ k_2e^{-\alpha_2s}p_2+\tilde{k}_{-2}p_3$$

$$\frac{\partial p_3}{\partial t}+\frac{dL}{dt}\frac{\partial p_3}{\partial s}=\ k_2e^{-\alpha_1s}p_2-\tilde{k}_{-2}p_3-\tilde{k}_3e^{\alpha_3\left(s+s_3\right)^2}p_3 \quad (16)$$

The initial state is obtained by holding the muscle at a fixed length L0 = 2.2 um, and integrating Equation (16) to reach a steady-state. Starting from this initial state, the applied force is reduced to a defined fraction of initial force, and the muscle dynamics simulated by integrating Equations (15) and (16).

The following is the plot for the steady state force and fixed length of sarcomere. (L0 = 2.2 um)

The magnitude of the steady state has been used as an initial condition.

title

Corrections on the original MATLAB code:

1. In the original code, p_3^2 in Equation (10) was multiplied by α_3 and the above derivation shows no coefficient for p_3^2. This typo, which negligible effects on the results, is fixed in the current (2019) distribution of the codes.
2. The Passive force subroutine has some typos for the passive force formulations. (Equations 2 and 13)
3. The force calculation formulation has been modified

Corrections to typos in the paper:

1. Equation (3) in the paper, should be (missing hat on $\hat{p}_1\ and\ \hat{p}_3$)

$$\frac{dP}{dt}=k_{np}N\left(t\right)-k_{pn}P\left(t\right)+\tilde{k}_d\hat{p}_1\left(t,s\right)-\ k_aP\left(t\right)+\tilde{k}_3e^{\alpha_3\left(s+s_3\right)^2}p_3\left(t,s\right)-\ldots$$

2. “F” in denominator of the LHS of the equation 12 of the paper should be changed to $\frac{dL}{dt}$. ==> $\frac{dL}{Ft}$ should be change to $\frac{dL}{dt}$
3. Page 23, Appendix D, line 11. “Panels C and D of Fig. D1” should be changed to “Panels B and D of Fig. D1”

Following are the code and results based on the above derivations

Importing the libraries that we are going to use in the program

In [1]:
```
import numpy as np
import time
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import math
```

Defining the heaviside function

In [2]:
```
def heav(x):
    return 1*(x>0)
```

Defining the passive force function based on the following formula

$$F_2\ (L) = \begin{cases} \beta.\, K_{PE,1}\ \left[e^{(PExp_{titin}(L-SL_{rest}))}-1\right] & if\ L > SL_{rest} \\ -\beta.\, K_{PE,1}\ \left[e^{(PExp_{titin}(SL_{rest}-L))}-1\right] & if\ L < SL_{rest} \end{cases}$$

and the derivative of the passive force according to equation (13):

In [3]:
```
def passiveForces(SL,SL_rest,kpe1):
    #The Passive Force Function has been corrected in the python version of th
e code.
    SLset = 2.2 # set Set sarcomere length, Units: um
    timrel = 0.1 # Time of sarcomere release, Units: ms
    # Passive Force formulation adopted from Rice etal Rice etal (Biophys J. 2
008 Sep;95(5):2368-90)
    # Function to calculate the passive forces of the muscle
    # The passive force is a function of the sarcomere length
    beta = 3.5 # um
    PCon_titin = kpe1*beta*0.002 #
    PExp_titin = 10 #(um-1)
    SL_collagen = 2.25 #(um)
    PCon_collagen = kpe1*beta*0.02 #
    PExp_collagen  = 70 #(um-1)

    # Passive forces: Trabeculae
    PF_titin = np.sign(SL-SL_rest)*PCon_titin* (math.exp(np.fabs(SL-SL_rest)*P
Exp_titin)-1)
    dPF_titin = PCon_titin*PExp_titin* math.exp(PExp_titin*np.abs(SL - SL_rest
))

    PF_collagen = heav(SL-SL_collagen)*PCon_collagen* (math.exp(PExp_collagen*
(SL-SL_collagen))-1)
    dPF_collagen = heav(SL-SL_collagen)*PCon_collagen*PExp_collagen* math.exp(
PExp_collagen*(SL-SL_collagen))

    PF = PF_titin + 1*PF_collagen #Trabeculae
    dPF = dPF_titin + dPF_collagen
    passive = [PF,dPF]
    return passive
```

Subroutine Model XB:

This subroutine used for the time integration

In [4]:
```
def Model_XB( y,t,TmpC,MgATP,Pi,MgADP,F,ode_solve):

    SLset = 2.2 # set Set sarcomere length, Units: um
    timrel = 0.1 # Time of sarcomere release, Units: ms
    ## Constants and parameters
    # Estimated parameters from Pi and ATP data (Average of N=21 GA runs)
    par = [4.5397e+02 ,  1.2521e+02 ,  4.1169e+01 ,  1.7553e+01 ,  1.5928e+02
,  1.5372e+00  , 8.7750e+01,   1.5137e+01 ,  1.0060e+01,   5.0247e+01 ,  9.938
3e-03,   4.0067e+00 ,  7.2899e+02  , 5.0129e-01 ,   1.1370e+03,   2.5464e+02,
1.9066e+04 ,  5.9698e-01]
    # Estimated Q10s from individual mouse Ref[20] in dynamic paper
    Q10s = [1.4382e+00 ,   3.0265e+00  , 1.0717e+00 ,  1.3403e+00 ,  1.4782e+0
0 ,  1.4413e+00]

    alpha1 = 1*par[7]
    alpha2 = 1*par[8]
    alpha3 = 1*par[9]
    s3 = par[10]
    K_Pi = par[11]
    K_T = par[17]
    K_D = 0.194 # MgADP dissociation constant from Yamashita etal (Circ Res. 1
994; 74:1027-33).
    g1 = (MgADP/K_D)/(1 + MgADP/K_D + MgATP/K_T) # is k-2_Bar in the dynamic p
aper. with out k-2

    g2 = (MgATP/K_T)/(1 + MgATP/K_T + MgADP/K_D) # is k3_bar in the dynamics p
aper. with out k3

    f1 = (Pi/K_Pi)/(1 + Pi/K_Pi) # is the k_d_bar in the dynamics paper. witho
ut kd

    f2 = 1/(1 + Pi/K_Pi) # is the k1_bar in the dynamic paper.

    # I think since the if the data we have is in other temprature. we can cal
culate all the
    # reaction rates from 17 C in the following. # for current input which is
    # Tmpc = 17. it will not make any changes to the reaction rates.


    kf = par[0]*Q10s[0]**((TmpC-17)/10) #ka in the paper
    kb = par[1]*f1*Q10s[0]**((TmpC-17)/10) #kd_bar in the paper
    k1 = par[2]*f2*Q10s[0]**((TmpC-17)/10)
    k_1 = par[3]*Q10s[0]**((TmpC-17)/10)
    k2 = par[4]*1*Q10s[1]**((TmpC-17)/10)
    k_2 = par[5]*1*Q10s[0]**((TmpC-17)/10)*g1
    k3 = par[6]*Q10s[0]**((TmpC-17)/10)*g2

    kpe1 = par[12]*Q10s[4]**((TmpC-17)/10)
    eta = par[13]*Q10s[2]**((TmpC-17)/10)
    kstiff1 = par[14]*Q10s[3]**((TmpC-17)/10)
    kpe2 = par[15]*Q10s[2]**((TmpC-17)/10)
    kstiff2 = par[16]*Q10s[5]**((TmpC-17)/10)

    SL_max = 2.4
    SL_min = 1.4
    SL_rest = 1.9 # (um)
```

```
    ## State Variables
    P1o = y[0]
    P1i = y[1]
    P1w = y[2]

    P2o = y[3]
    P2i = y[4]
    P2w = y[5]

    P3o = y[6]
    P3i = y[7]
    P3w = y[8]
    Pu = 1 - P1o - P2o - P3o
    SL = y[9]

    ## Stretch-sensitive rates
    f_alpha1o = (P1o - alpha1*P1i + 0.5*(alpha1*alpha1)*P1w)
    f_alpha1i = (P1i - alpha1*P1w)

    alpha0 = 1*alpha1
    f_alpha0o = (P2o + alpha0*P2i + 0.5*alpha0*alpha0*P2w)
    f_alpha0i = (P2i + alpha0*P2w)

    f_alpha2o = (P2o - alpha2*P2i + 0.5*(alpha2*alpha2)*P2w)
    f_alpha2i = (P2i - alpha2*P2w)

    alpha2b = 0;
    f_alphao = (P3o + alpha2b*P3i + 0.5*(alpha2b*alpha2b)*P3w)
    f_alphai = (P3i + alpha2b*P3w)

    f_alpha3o = (P3o + alpha3*(s3*s3*P3o + 2*s3*P3i + P3w))
    f_alpha3i = (P3i + alpha3*(s3*s3*P3i + 2*s3*P3w))

    ## Compute Active & Passive Force
    # Active Force
    dr = 0.01 # Power-stroke Size; Units: um # or 10 nm
    B_process = kstiff2*dr*P3o  # Force due to XB cycling
    C_process = kstiff1*(P2i+P3i) # Force due to stretching of XBs
    F_XB = (B_process + C_process)

    # Non-linear Passive force; Adopted from Rice etal (Biophys J. 2008 Sep;95
(5):2368-90)
    [F_passive,dFpassive] = passiveForces(SL,SL_rest,kpe2)


    f_myofibril = 0.45 # Percent Myofibril in Muscle from Palmer etal (Mol Cel
l Biochem. 2004 Aug;263(1-2):73-80)
    F_active_passive_effective = f_myofibril * (F_XB + F_passive)#

    Bxb = kstiff1*(P2o + P3o)
    Axb = heav(t-timrel)*(kstiff1*( k3*(P3i + alpha3*(P3i*s3**2 + 2*P3w*s3)) +
k1*(P1i - P1w*alpha1) - k_1*(P2i + P2w*alpha1)) + dr*kstiff2*(-P3o*k_2 - k3*(P
3o + alpha3*(P3o*s3**2 + 2*P3i*s3 )+ P3w) + k2*((P2w*alpha2**2)/2 - P2i*alpha2
+ P2o)))
    F_sum = heav(t-timrel)*(F - F_active_passive_effective)
    den = 1 + Bxb/kpe1 + dFpassive/kpe1
```

```
    Fdot = 0
    ## XB ODEs

    dSL = ((F_sum/eta) + (Fdot - Axb)/kpe1)/den*heav(SL-SL_min)*heav(SL_max-SL
)
    dP1o = kf*Pu   - kb*P1o - k1*f_alpha1o + k_1*f_alpha0o
    dP1i = 1*dSL*P1o - kb*P1i - k1*f_alpha1i + k_1*f_alpha0i
    dP1w = 2*dSL*P1i - kb*P1w - k1*P1w + k_1*P2w

    dP2o =         - k_1*f_alpha0o - k2*f_alpha2o + k_2*f_alphao + k1*f_alpha1
o
    dP2i = 1*dSL*P2o - k_1*f_alpha0i - k2*f_alpha2i + k_2*f_alphai + k1*f_alph
a1i
    dP2w = 2*dSL*P2i - k_1*P2w       - k2*P2w + k_2*P3w + k1*P1w

    dP3o =         + k2*f_alpha2o - k_2*f_alphao - k3*f_alpha3o
    dP3i = 1*dSL*P3o + k2*f_alpha2i - k_2*f_alphai - k3*f_alpha3i
    dP3w = 2*dSL*P3i + k2*P2w       - k_2*P3w - k3*P3w

    dYdT = [dP1o, dP1i, dP1w, dP2o, dP2i, dP2w, dP3o, dP3i, dP3w, dSL]
    if ode_solve==1:
        return dYdT
    else:
        return dSL,F_active_passive_effective

start = time.time()
SLset = 2.2 # set Set sarcomere length, Units: um
timrel = 0.1 # Time of sarcomere release, Units: ms
# Set temperature and initial SL
TmpC = 17
SL0 = 2.2 # Initial length of the sarcomere um

# Set metabolite concentrations,
MgATP = 5.0
MgADP = 0.0
Pi = 0.0
# Experimental conditions from Palmer etal J Mol Cell Cardiol. 2013 Apr;57:23-
31


init = np.zeros(10)
init[9] = SL0
Fmax= 37.9386
```

In [5]:
```
F_load = np.arange(0,1.05,0.05)* Fmax
```

In [6]:
```
mm = len(F_load)

vm = np.zeros((mm))
tspan = np.arange(0,0.4005,0.0005)

nn = len(tspan)
dSL = np.zeros((nn,mm))
SL_all = np.zeros((nn,mm))
Ftotal = np.zeros((nn,mm))
```

We define a time point that the speed should be calculated on the simulation - it depends on the time step that we define at tspan.

Time-points at which the average slope of sliding velocity is calculated i.e. between 40 to 50 ms after release

In [7]:
```
ts = 291
X = np.arange(ts-10,ts+11,1)
```

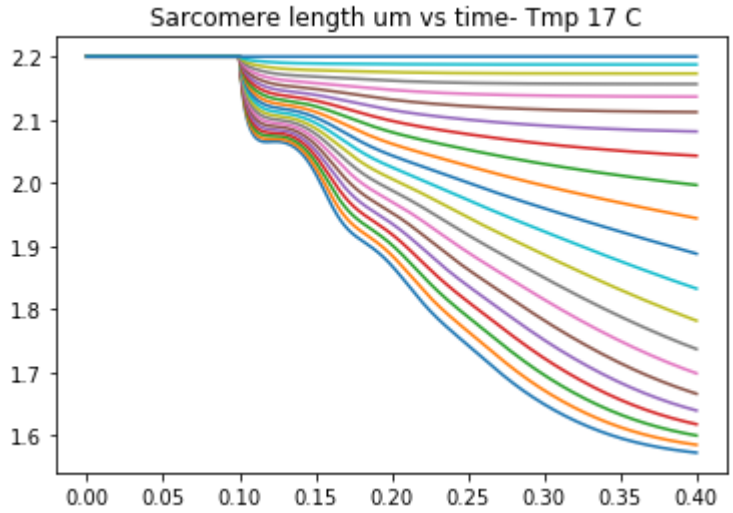
Ploting the model predictions for sarcomere length ande velocity for different afterloads

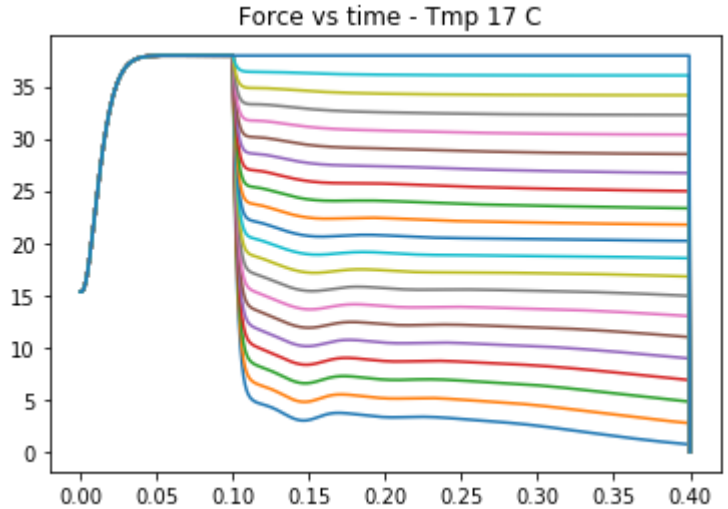
In [8]:
```
for j in range(1,mm +1):


    y = odeint(Model_XB,init,tspan,args = (TmpC,MgATP,Pi,MgADP,F_load[j-1],1))
    SL = y[:,9]
    for i in range (1,nn):
        dSL[i-1][j-1],Ftotal[i-1][j-1]= Model_XB(y[i-1,:],tspan[i-1],TmpC,MgAT
P,Pi,MgADP,F_load[j-1],0)

        vm[j-1] = np.mean(dSL[X-1,j-1]/SL0)

    Ftotal_load = Ftotal[:,j-1]
    plt.figure(3)
    plt.title(' Sarcomere length um vs time- Tmp 17 C')
    plt.plot(tspan,SL)
    #print (vm)
    #print (Ftotal.shape)
    #print (tspan)
    plt.figure(4)
    plt.plot(tspan,Ftotal_load)
    plt.title(' Force vs time - Tmp 17 C')
```





In [ ]:

Calculating the computational time

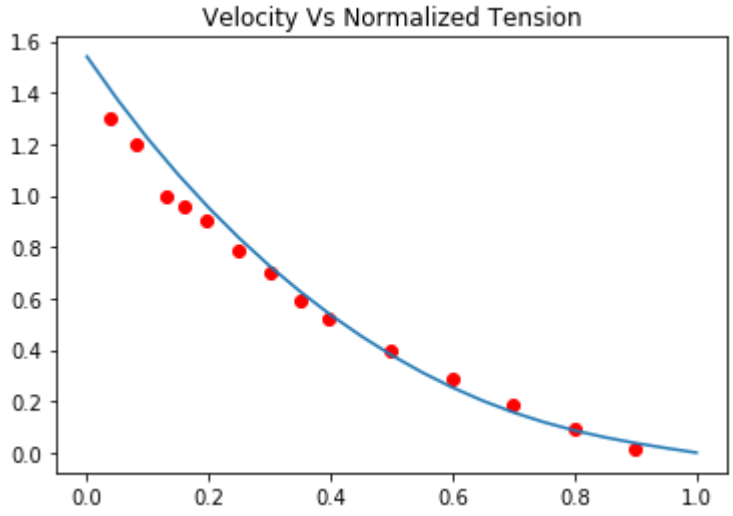
In [9]:
```
endtime = time.time()
print(endtime - start)
```

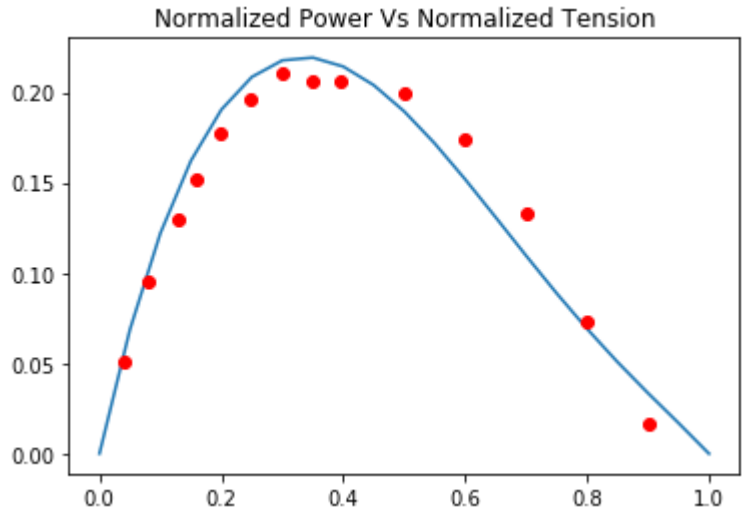
```
2.2018094062805176
```

Compare Model VS Experimental Data

Plots Data from Palmer et al J Mol Cell Cardiol. 2013 Apr;57:23-31

In [10]:
```
data_x = np.array([0.0391,    0.0797,    0.1300,    0.1580,    0.1970,    0.24
80,    0.3010,    0.3490,    0.3960,    0.4990,    0.5990,    0.7000,    0.800
0,    0.9000])
data_y = np.array([1.3000,    1.2000,    1.0000,    0.9600,    0.9000,    0.79
00,    0.7000,    0.5900,    0.5200,    0.4000,    0.2900,    0.1900,    0.091
0,    0.0180])
plt. figure(1)
plt.title('Velocity Vs Normalized Tension')
plt.plot(data_x,data_y,'ro')
plt.plot(F_load/Fmax, -vm)
#fig, axs = plt.figure(1)
#plt.axis('equal')
plt. figure(2)
plt.title('Normalized Power Vs Normalized Tension')
normal_power = data_y * data_x
plt.plot(F_load/Fmax, (-vm * F_load) / Fmax)
plt.plot(data_x,normal_power,'ro')
plt.show()
```





In [ ]:

In [ ]: