1 TKT 4150 Biomechanics: Problem set 9

For an optimal experience visit NTNU's jupyter-server and login with your NTNU credentials or visit tmpnb.org and upload "windkessels.ipynb", "Windkessel Schematics.jpg", and '55arteryNetwork_age19.csv' and '55arteryNetwork_age75.csv'. Then open windkessels.ipynb on that server to continue.

Introduction. This is an IPython notebook, for a more general introduction please see this introduction and guide where you can download a similar notebooks that are more instructional. This website gives a similar guide.

Save your work!

- If you are running this on tmpnb.org, remember to save your work by downloading the notebook via clicking `File->Download As` in the toolbar.

Exercise 1: Deriving the Windkessel Model

a) Use the principle of conservation of mass (i.e. volume for an incompressible material) to derive the differential equation for pressure corresponding to the 2 element Windkessel by assuming a compliant tube with inflow Q_1 such the $A(P) = CP$ where the distal (downstream) flow out of the tube Q_2 is driven across a fluid resistance R_p such that $P = R_p Q_2$. Q_2 is the flow out of compliant arteries over the peripheral (resistance) vessels.

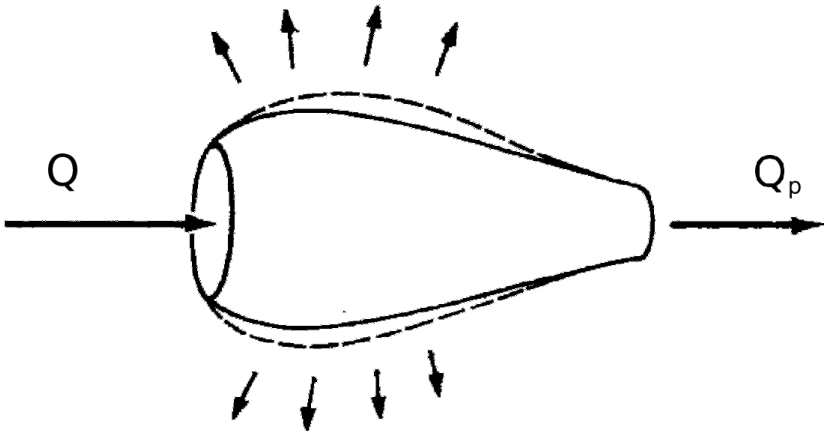

Figure 1: Diagram of mass balance in a section of compliant pipe. Q is Q_1, Q_p is Q_2.

Solution. Conservation of mass implies that $\dot{V} = Q_1 - Q_2$. $A(P)$ may be used to connect V to P as $V = l \times A$. Assuming l doesn't change significantly then $V(P) = lCP$. Thus $\dot{V} = lC\dot{P}$. Now the pressure P determines the flow out of the vessel Q_2 which is given by $Q_2 = \frac{P}{R_p}$. Thus $lC\dot{P} = Q_1 - \frac{P}{R_p}$

b) *Derivation of 4-element Windkessel Model via averaging of the 1-D Navier Stokes* Consider the governing equations for flow through a compliant tube

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0 \tag{1}$$

and

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}(\frac{Q^2}{A}) + \frac{A}{\rho}\frac{\partial P}{\partial x} + K_R\frac{Q}{A} = 0 \tag{2}$$

completed with the constitutive equation $P = \beta(\sqrt{A} - \sqrt{A_0})$

Show that the average values of P and Q, i.e. $\bar{P} = \frac{1}{l}\int_0^l P dx$, $\bar{Q} = \frac{1}{l}\int_0^l Q dx$ may be related as

$$k_1 l\frac{d\bar{P}}{dt} = Q_1 - Q_2 \tag{3}$$

and

$$\frac{d\bar{Q}}{dt}\frac{\rho l}{A_0} + \frac{K_R l\rho}{A_0^2}\bar{Q} = P(0) - P(l) \tag{4}$$

by integrating the partial differential equations from $x = 0$ to $x = l$. These equations are essentially the 4-element Windkessel equations. To complete the system assume a resistance, R_p, between $P(l)$ and venous pressure $P_v = 0$. K_R is the coefficient of viscous friction for a given velocity profile and A_0 is the characteristic cross sectional area.

Hint. To find $\frac{d\bar{P}}{dt}$ consider the chain rule applied to $\int_0^l \frac{\partial \bar{P}}{\partial t} dx$.

Assume the convective term $\frac{\partial}{\partial x}(\frac{Q^2}{A})$ is negligible and that spatial variations in A are much smoother than variations in P and Q, i.e. treat A as a constant over space. Dropping negligible terms we see $\int \frac{\partial Q}{\partial t} + \frac{A}{\rho}\frac{\partial P}{\partial x} + K_R\frac{Q}{A}dx = \frac{d\bar{Q}}{dt} + \frac{lA_0}{\rho}(P(x_2) - P(x_1)) + \frac{lK_R}{A_0}\bar{Q} \stackrel{=}{} 0$ which may be simplified to the solution given.

Solution. $P = \beta(\sqrt{A} - \sqrt{A_0})$ For the pressure consider $\int \frac{\partial P}{\partial t}dx = \int \frac{\beta}{2\sqrt{A}}\frac{\partial A}{\partial t}dx$. Assuming spatial variations in A are small we see that $\frac{d\bar{P}}{dt} = \frac{l\beta}{2\sqrt{A_0}}\frac{d\bar{A}}{dt}$.

Exercise 2: Frequency domain impedance analysis comparison of Windkessel models

Windkessel models are often used to interpret pressure and flow signals in the frequence domain. In this domain the relationship between pressure and flow is given as $\hat{p} = Z\hat{q}$ where $\hat{p}$ and $\hat{q}$ are the fourier transform of p and q. Impedances

are linear and thus may be added accordning to the formula $Z_{eq} = \sum Z_i$ for impedances in series, i.e. same flow through a sequence of elements, and $Z_{eq}^{-1} = \sum Z_i^{-1}$ for impedances in parallel, i.e. same pressure across a sequence of elements by split of flow. Capacitance has an impedance $Z_C = \frac{1}{j\omega C}$, resistance has impedance $Z_R = R$, and inertance has impedance $Z_L = j\omega L$ where $\omega = 2\pi f$ is the angular frequency.

a)

1. Derive the impedance for the 2-Element, 3-Element and 4-Element Windkessel models shown in Figure 2
2. How does the total resistance relate R in the 2-element Windkessel relate to the characteristic impedance and peripheral resistance in the other models?
3. Graph the input impedances over frequencies of inputs.
4. What interpretation can you make about the impedances at low and high frequencies? What implications does this have for relationship between flow and pressure in the cardiovascular system?

Answer.

- 2-Element windkessel $Z_{eq} = \frac{R}{1+j\omega RC}$
- 3-Element windkessel $Z_{eq} = R_c + \frac{R_p}{1+j\omega R_p C}$
- 4-Element series windkessel $Z_{eq} = R_c + j\omega L_c + \frac{R}{1+j\omega RC}$
- 4-Element parallel windkessel $Z_{eq} = \frac{j\omega R_c L_c}{R_c+j\omega L_c} + \frac{R}{1+j\omega RC}$

Total resistance is defined as the ratio between mean flow and mean pressure $R_{tot} = \frac{\bar{P}}{\bar{Q}}$, in the 3 and 4 element windkessels this is given by $R_c + R_p = \frac{\bar{P}}{\bar{Q}}$. You may also think of this as the steady state relation between pressure and flow. Thus inertial and deformation effects are not relevant. See figure at end. For all the impedance tends to R at $\omega = 0$ thus the constant flow relation is given by R. For the 2 and 3 element Windkessels at high frequencies Z_{eq} tend toward 0 thus sharp oscillations are dampened. In the series 4 element Windkessel the impedance tends towards ∞, thus high frequency oscillations are sustained. As models of the cardiovascular system the 4-element models might better represent sharp oscillations, but risk over emphasizing unphysiological features. The 2 and 3 element models likely over smooth the relationship between flow and pressure, but may be more robust and realistic when considering noisy input data.

b) Compare how well these relations fit some data. Below you will find implementation of the functions necessary to fit the impedance of the 2 element Windkessel model to a data set, e.g. './55arteryNetwork_age19.csv' and './55arteryNetwork_age75.csv'. Below you will find an example that estimates the compliance for the 2 element Windkessel.

Extend this example to estimate compliance, characteristic impedance and intertance for the windkessel modesl shown in Figure 2. Use this to compare compliance, characteristic aortic impediance, and characteristic aortic intertance between the 19 year old and the 75 year old.

```
#Intro
import numpy as np
import scipy as sp
import scipy.optimize
from scipy import io as sio
from matplotlib import pyplot as plt
unit_Pa_to_mmHg = 1/133.32
unit_m3_to_ml = 1e6

# This code may be used to import ./55arteryNetwork_age19.csv or ./55arteryNetwork_age75.csv
csv = np.genfromtxt ('./55arteryNetwork_age19.csv', delimiter=";", names=True)
p = csv["P_Pa"]*unit_Pa_to_mmHg
q = csv["Q_m3s"]*unit_m3_to_ml
t = csv["t_s"]
dt = dt = t[1]-t[0]
N = len(t)
T = N*dt
w = np.arange(N//2+1)*2*np.pi/T # Frequencies corresponding to an FFT of the data.

#WK2 Imp
def ZWK2(R,C,w):
    """ Calculate and return impedance of a 2elt WK
    Args:
        R Resistance
        C Compliance
        w the frequencies at which the impediance is calculated.
    """
    Z=R/(1+1j*w*R*C)
    return Z

#WK2 Est
#=============================================================================
# Guess a compliance value
#=============================================================================
R=np.mean(p)/np.mean(q) # Total arterial reistance is given by the ratio of
                        # average pressure to average flow. This is not always
                        # the same as the peripheral resistance. How can you
                        # modify this for R_c and R_p

C_guess=0.9


#---Compute the pressure pwk0 from impedance based on the guessed compliance value
Qfft=np.fft.rfft(q)
Z = ZWK2(R,C_guess,w)
pwk0 = np.fft.irfft(Qfft*Z,len(q))  #the len(q) has has to be specified when len(q) is odd

def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WKresidual(opt_args[0], R, p, q,t)
```

```
#=============================================================================
#  Estimate the compliance
#=============================================================================
import scipy.optimize
# res = scipy.optimize.minimize? #Uncomment this line to see the  help for scipy.optimize.minimize
# x0, the inital guess is a vector/list so for the 2 element Windkessel we only estimate C_guess,
# thus only 1 entry.
res = scipy.optimize.minimize(residual, [C_guess,])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Rwk2=R
Cwk2=res.x[0]
print("WK2", 'C=%f'%Cwk2)


Zeq_wk2 = ZWK2(Rwk2,Cwk2,w)
pwk2 = np.fft.irfft(Zeq_wk2*Qfft,len(q))

#WK2 Plot
plt.figure()
plt.title('Predicted pressures based on estimated compliances')
_=plt.plot(t,pwk0, label="C guess")
_=plt.plot(t,pwk2, label="Wk2 estimated fft")
_=plt.plot(t,p,'k', label="data")
_=plt.legend()
plt.xlabel("Time")
plt.ylabel("Pressure")
```

Solution. A Solution

```
#!/usr/bin/env python
#Intro
import numpy as np
import scipy as sp
import scipy.optimize
from scipy import io as sio
from matplotlib import pyplot as plt
unit_Pa_to_mmHg = 1/133.32
unit_m3_to_ml = 1e6

# This code may be used to import ./55arteryNetwork_age19.csv or ./55arteryNetwork_age75.csv
csv = np.genfromtxt ('./55arteryNetwork_age19.csv', delimiter=";", names=True)
p = csv["P_Pa"]*unit_Pa_to_mmHg
q = csv["Q_m3s"]*unit_m3_to_ml
t = csv["t_s"]
dt = dt = t[1]-t[0]
N = len(t)
T = N*dt
w = np.arange(N//2+1)*2*np.pi/T # Frequencies corresponding to an FFT of the data.

#End Intro
#WK2 Imp
def ZWK2(R,C,w):
    """ Calculate and return impedance of a 2elt WK
    Args:
        R Resistance
        C Compliance
        w the frequencies at which the impediance is calculated.
    """
    Z=R/(1+1j*w*R*C)
    return Z

#End WK2 Imp
```

```
def ZWK3(Z, R, C, w):
    """ Calculate and return impedance of a 2elt WK
    Args:
        Z Characteristic impedance
        R Peripheral Resistance
        C Compliance
        w the frequencies at which the impediance is calculated.
    """
    Zeq=Z + R/(1+1j*w*R*C)
    return Zeq

def ZWK4a(L,Z,R,C,w):
    """ Calculate and return impedance of a 2elt WK
    Args:
        L - characteristic intertance
        Z - characteristic impedance
        R - Peripheral Resistance
        C - Compliance
        w the frequencies at which the impediance is calculated.
    """
    Zeq= 1j*w*L +Z + R/(1+1j*w*R*C)
    return Zeq

def ZWK4b(L,Z,R,C,w):
    """ Calculate and return impedance of a 2elt WK
    Args:
        L - characteristic intertance
        Z - characteristic impedance
        R - Peripheral Resistance
        C - Compliance
        w the frequencies at which the impediance is calculated.
    """
    Zeq = 1j*w*Z*L/(1j*w*L +Z) + R/(1+1j*w*R*C)
    return Zeq

#WK2 Res
def WKresidual(C,R,p,q,t):
    """ Calculates error between Windkessel model and Data
        Args:
            C - compliance
            R - Resistance
            p - pressure measurements in time domain
            Qfft - Discrete fourier transform of flows corresponding to p
            w - frequencies of DFT
            t - times of p
        Returns:
            r : the sum of squared errors between pwk and p
    """
    N=len(q)
    dt=t[1]-t[0]
    T = N*dt;
    w = np.arange(N//2+1)*2*np.pi/T
    Qfft = np.fft.rfft(q)
    Pfft = np.fft.rfft(p)
    Zdata=Pfft/Qfft
    Z=ZWK2(R,C,w)
    Pwk_fft=Z*Qfft
    r=np.sum(np.abs(Pfft-Pwk_fft)**2)/N
    return r
#End WK2 Res

def WK3residual(Z, C, R_tot, p, q, t):
    """ Calculates error between Windkessel model and Data
```

```
        Args:
            Z - characteristic impedance
            C - compliance
            R_tot - Total Resistance
            p - pressure measurements in time domain
            Qfft - Discrete fourier transform of flows corresponding to p
            w - frequencies of DFT
            t - times of p
        Returns:
            r : the sum of squared errors between pwk and p
    """
    N=len(q)
    dt=t[1]-t[0]
    T = N*dt;
    w = np.arange(N//2+1)*2*np.pi/T
    Qfft = np.fft.rfft(q)
    Pfft = np.fft.rfft(p)
    Zdata=Pfft/Qfft
    Zeq=ZWK3(Z,R_tot-Z,C,w)
    Pwk_fft=Zeq*Qfft
    r=np.sum(np.abs(Pfft-Pwk_fft)**2)/N
    return r

def WK4aresidual(L,Z,C,R_tot,p,q,t):
    """ Calculates error between Windkessel model and Data
        Args:
            L - characteristic intertance
            Z - characteristic impedance
            C - compliance
            R_tot - Total Resistance
            p - pressure measurements in time domain
            Qfft - Discrete fourier transform of flows corresponding to p
            w - frequencies of DFT
            t - times of p
        Returns:
            r : the sum of squared errors between pwk and p
    """
    N=len(q)
    dt=t[1]-t[0]
    T = N*dt;
    w = np.arange(N//2+1)*2*np.pi/T
    Qfft = np.fft.rfft(q)
    Pfft = np.fft.rfft(p)
    Zdata=Pfft/Qfft
    Zeq=ZWK4a(L,Z, R_tot-Z,C,w)
    Pwk_fft=Zeq*Qfft
    r=np.sum(np.abs(Pfft-Pwk_fft)**2)/N
    return r

def WK4bresidual(L,Z,C,R_tot,p,q,t):
    """ Calculates error between Windkessel model and Data
        Args:
            L - characteristic intertance
            Z - characteristic impedance
            C - compliance
            R_tot - Total Resistance
            p - pressure measurements in time domain
            Qfft - Discrete fourier transform of flows corresponding to p
            w - frequencies of DFT
            t - times of p
        Returns:
            r : the sum of squared errors between pwk and p
    """
    N=len(q)
```

```
    dt=t[1]-t[0]
    T = N*dt;
    w = np.arange(N//2+1)*2*np.pi/T
    Qfft = np.fft.rfft(q)
    Pfft = np.fft.rfft(p)
    Zdata=Pfft/Qfft
    Zeq=ZWK4b(L,Z,R_tot-Z,C,w)
    Pwk_fft=Zeq*Qfft
    r=np.sum(np.abs(Pfft-Pwk_fft)**2)/N
    return r

#WK2 Est
#================================================================================
# Guess a compliance value
#================================================================================
R=np.mean(p)/np.mean(q) # Total arterial reistance is given by the ratio of
                        # average pressure to average flow. This is not always
                        # the same as the peripheral resistance. How can you
                        # modify this for R_c and R_p

C_guess=0.9


#---Compute the pressure pwk0 from impedance based on the guessed compliance value
Qfft=np.fft.rfft(q)
Z = ZWK2(R,C_guess,w)
pwk0 = np.fft.irfft(Qfft*Z,len(q))  #the len(q) has has to be specified when len(q) is odd

def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WKresidual(opt_args[0], R, p, q,t)

#================================================================================
#  Estimate the compliance
#================================================================================
import scipy.optimize
# res = scipy.optimize.minimize? #Uncomment this line to see the  help for scipy.optimize.minimize
# x0, the inital guess is a vector/list so for the 2 element Windkessel we only estimate C_guess,
# thus only 1 entry.
res = scipy.optimize.minimize(residual, [C_guess,])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Rwk2=R
Cwk2=res.x[0]
print("WK2", 'C=%f'%Cwk2)


Zeq_wk2 = ZWK2(Rwk2,Cwk2,w)
pwk2 = np.fft.irfft(Zeq_wk2*Qfft,len(q))

#WK3 Est
def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WK3residual(opt_args[0],opt_args[1], R, p, q,t)

Z_guess = 0.1*R
res = scipy.optimize.minimize(residual, [Z_guess, C_guess])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Zwk3=res.x[0]
Cwk3=res.x[1]
print('WK3', 'Z=%f'%Zwk3,'C=%f'%Cwk3)
```

```
Zeq_wk3 = ZWK3(Zwk3, R-Zwk3,Cwk3,w)
pwk3 = np.fft.irfft(Zeq_wk3*Qfft,len(q))

# WK4a

def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WK4aresidual(opt_args[0],opt_args[1],opt_args[2], R, p, q,t)

Z_guess = 0.1*R
L_guess = 0.1

res = scipy.optimize.minimize(residual, [L_guess, Z_guess, C_guess])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Lwk4a=res.x[0]
Zwk4a=res.x[1]
Cwk4a=res.x[2]
print('WK4a', 'L=%f'%Lwk4a,'Z=%f'%Zwk4a,'C=%f'%Cwk4a)


Zeq_wk4a = ZWK4a(Lwk4a, Zwk4a, R-Zwk4a,Cwk4a,w)
pwk4a = np.fft.irfft(Zeq_wk4a*Qfft,len(q))

# WK4b

def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WK4bresidual(opt_args[0],opt_args[1],opt_args[2], R, p, q,t)

Z_guess = 0.1*R
L_guess = 0.1

res = scipy.optimize.minimize(residual, [L_guess, Z_guess, C_guess])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Lwk4b=res.x[0]
Zwk4b=res.x[1]
Cwk4b=res.x[2]
print('WK4b', 'L=%f'%Lwk4b,'Z=%f'%Zwk4b,'C=%f'%Cwk4a)


Zeq_wk4b = ZWK4a(Lwk4a, Zwk4a, R-Zwk4a,Cwk4a,w)
pwk4b = np.fft.irfft(Zeq_wk4b*Qfft,len(q))


# WK4b

#=============================================================================
# Plot solutions
#=============================================================================

#WK2 Plot
plt.figure()
plt.title('Predicted pressures based on estimated compliances')
_=plt.plot(t,pwk0, label="C guess")
_=plt.plot(t,pwk2, label="Wk2 estimated fft")
_=plt.plot(t,p,'k', label="data")
_=plt.legend()
plt.xlabel("Time")
plt.ylabel("Pressure")
#End WK2 Plot

plt.figure("Pressure")
```

```
plt.title('Predicted pressures based on estimated compliances')
_=plt.plot(t,pwk0, label="C guess")
_=plt.plot(t,pwk2, label="Wk2 estimated fft")
_=plt.plot(t,pwk3, label="Wk3 estimated fft")
_=plt.plot(t,pwk4a, label="Wk4a estimated fft")
_=plt.plot(t,pwk4b, label="Wk4b estimated fft")
_=plt.plot(t,p,'k', label="data")
_=plt.legend()
plt.xlabel("Time")
plt.ylabel("Pressure")



plt.figure("Z_{eq}")
w_plt = np.linspace(0,10,100) #Not enough low frequency points in fft
plt.title('Predicted Impedances based on estimated parameters')
_=plt.plot(w_plt,np.abs(ZWK2(R,Cwk2,w_plt)), label="Wk2 estimated fft")
_=plt.plot(w_plt,np.abs(ZWK3(Zwk3, R-Zwk3,Cwk3,w_plt)), label="Wk3 estimated fft")
_=plt.plot(w_plt,np.abs(ZWK4a(Lwk4a, Zwk4a, R-Zwk4a,Cwk4a,w_plt)), label="Wk4a estimated fft")
_=plt.plot(w_plt,np.abs(ZWK4b(Lwk4b, Zwk4b, R-Zwk4b,Cwk4b,w_plt)), label="Wk4b estimated fft")
_=plt.legend()
plt.xlim(0,10)
plt.xlabel("Frequency [Hz]")
plt.ylabel("$|Z_{eq}|$")

# This code may be used to import ./55arteryNetwork_age19.csv or ./55arteryNetwork_age75.csv
csv = np.genfromtxt ('./55arteryNetwork_age75.csv', delimiter=";", names=True)
p = csv["P_Pa"]*unit_Pa_to_mmHg
q = csv["Q_m3s"]*unit_m3_to_ml
t = csv["t_s"]
dt = dt = t[1]-t[0]
N = len(t)
T = N*dt
w = np.arange(N//2+1)*2*np.pi/T # Frequencies corresponding to an FFT of the data.

R=np.mean(p)/np.mean(q) # Total arterial reistance is given by the ratio of
                        # average pressure to average flow. This is not always
                        # the same as the peripheral resistance. How can you
                        # modify this for R_c and R_p

C_guess=0.9


#---Compute the pressure pwk0 from impedance based on the guessed compliance value
Qfft=np.fft.rfft(q)
Z = ZWK2(R,C_guess,w)
pwk0 = np.fft.irfft(Qfft*Z,len(q))  #the len(q) has has to be specified when len(q) is odd

def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WKresidual(opt_args[0], R, p, q,t)

res = scipy.optimize.minimize(residual, [C_guess,])
Rwk2=R
Cwk2=res.x[0]
print("Older WK2", 'C=%f'%Cwk2)

Zeq_wk2 = ZWK2(Rwk2,Cwk2,w)
pwk2 = np.fft.irfft(Zeq_wk2*Qfft,len(q))

#WK3 Est
def residual(opt_args):
    """Transform WKresidual to a residual function of C,
```

```
    which is the only scalar argument"""
    return WK3residual(opt_args[0],opt_args[1], R, p, q,t)

Z_guess = 0.1*R
res = scipy.optimize.minimize(residual, [Z_guess, C_guess])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Zwk3=res.x[0]
Cwk3=res.x[1]
print('Older WK3', 'Z=%f'%Zwk3,'C=%f'%Cwk3)

Zeq_wk3 = ZWK3(Zwk3, R-Zwk3,Cwk3,w)
pwk3 = np.fft.irfft(Zeq_wk3*Qfft,len(q))

# WK4a
def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WK4aresidual(np.exp(opt_args[0]),opt_args[1],opt_args[2], R, p, q,t)

Z_guess = 0.1*R
L_guess = 0.1

res = scipy.optimize.minimize(residual, [np.log(L_guess), Z_guess, C_guess])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Lwk4a=np.exp(res.x[0])
Zwk4a=res.x[1]
Cwk4a=res.x[2]
print('Older WK4a', 'L=%f'%Lwk4a,'Z=%f'%Zwk4a,'C=%f'%Cwk4a)

Zeq_wk4a = ZWK4a(Lwk4a, Zwk4a, R-Zwk4a,Cwk4a,w)
pwk4a = np.fft.irfft(Zeq_wk4a*Qfft,len(q))

# WK4b
def residual(opt_args):
    """Transform WKresidual to a residual function of C,
    which is the only scalar argument"""
    return WK4bresidual(np.exp(opt_args[0]),opt_args[1],opt_args[2], R, p, q,t)

Z_guess = 0.1*R
L_guess = 0.05

res = scipy.optimize.minimize(residual, [np.log(L_guess), Z_guess, C_guess])
#res = scipy.optimize.minimize_scalar(lambda C: WKresidual(C, R, p, q,t))
Lwk4b=np.exp(res.x[0])
Zwk4b=res.x[1]
Cwk4b=res.x[2]
print('Older WK4b', 'L=%f'%Lwk4b,'Z=%f'%Zwk4b,'C=%f'%Cwk4a)

Zeq_wk4b = ZWK4a(Lwk4a, Zwk4a, R-Zwk4a,Cwk4a,w)
pwk4b = np.fft.irfft(Zeq_wk4b*Qfft,len(q))

#==============================================================================
# Plot solutions
#==============================================================================

plt.figure("Pressure")
plt.title('Predicted pressures based on estimated compliances')
_=plt.plot(t,pwk2, label="Older Wk2 estimated fft")
_=plt.plot(t,pwk3, label="Older Wk3 estimated fft")
_=plt.plot(t,pwk4a, label="Older Wk4a estimated fft")
_=plt.plot(t,pwk4b, label="Older Wk4b estimated fft")
_=plt.plot(t,p,'k--', label="Older data")
_=plt.legend()
plt.xlabel("Time")
```

```
plt.ylabel("Pressure")



plt.figure("Z_{eq}")
w_plt = np.linspace(0,10,100) #Not enough low frequency points in fft
plt.title('Predicted Impedances based on estimated parameters')
_=plt.plot(w_plt,np.abs(ZWK2(R,Cwk2,w_plt)), label="Older Wk2 estimated fft")
_=plt.plot(w_plt,np.abs(ZWK3(Zwk3, R-Zwk3,Cwk3,w_plt)), label="Older Wk3 estimated fft")
_=plt.plot(w_plt,np.abs(ZWK4a(Lwk4a, Zwk4a, R-Zwk4a,Cwk4a,w_plt)), label="Older Wk4a estimated fft")
_=plt.plot(w_plt,np.abs(ZWK4b(Lwk4b, Zwk4b, R-Zwk4b,Cwk4b,w_plt)), label="Older Wk4b estimated fft")
_=plt.legend()
```

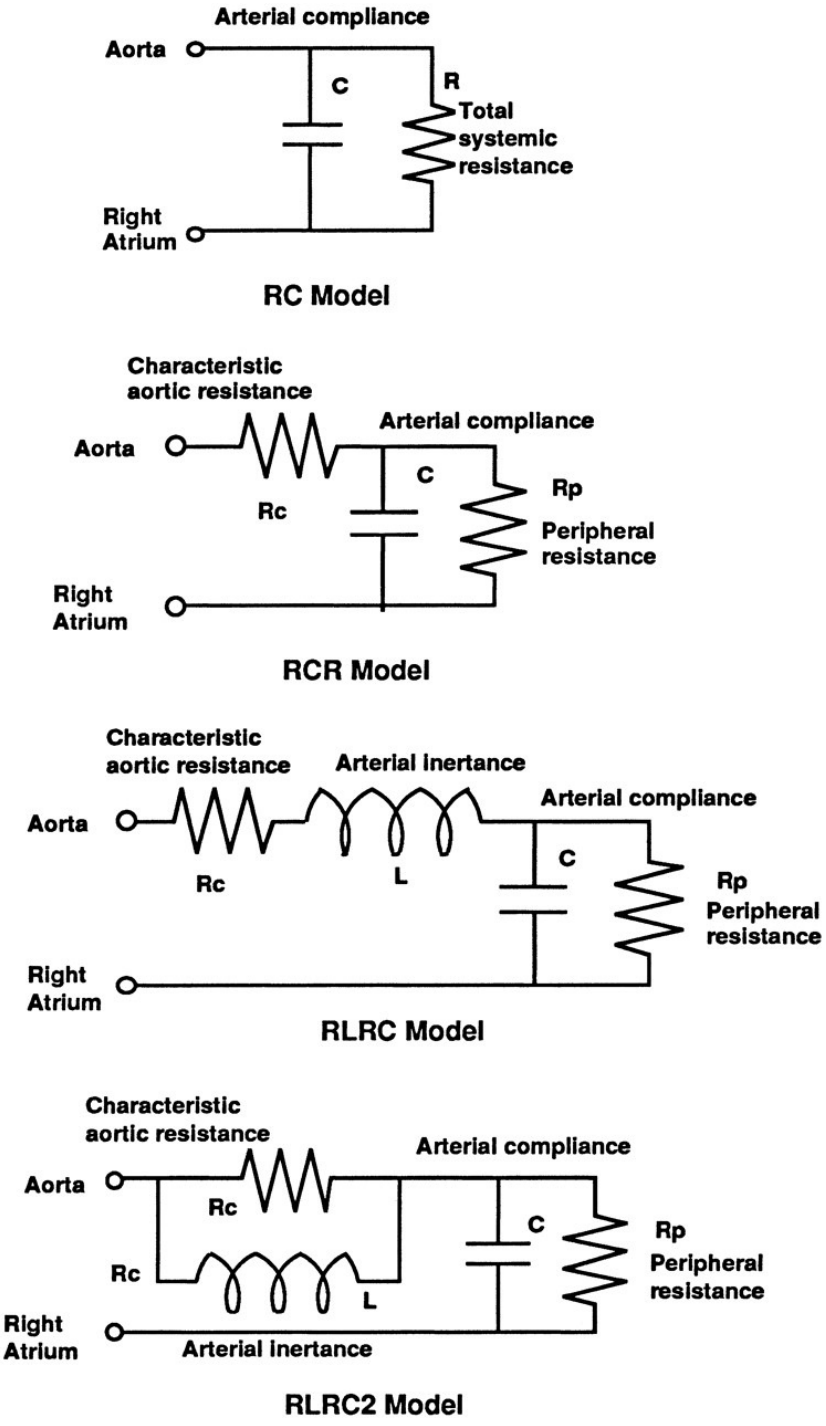

Figure 2: Diagrams of various Windkessel type models.