Dynamic (time-variant) model development of the FrED heater.

Various 1st order models are explored with delayed duty cycle schemes.

Uses impule testing data - "FrED_HeaterTest_DutyCyclePulse.csv"

J. Cuiffi, H. Wang - Penn State New Kensington

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

In [2]:
```
# Load impule data into dataframe
# Note: data collecteion started at an elevated heater temperature. Room temperature during
#       testing was approximately 20C
path_local = 'C:/Users/cuiff/Dropbox/Python Common Library/python-fred/data/Component Tests/'
data = pd.read_csv(path_local + 'FrED_HeaterTest_DutyCyclePulse.csv')
data
```

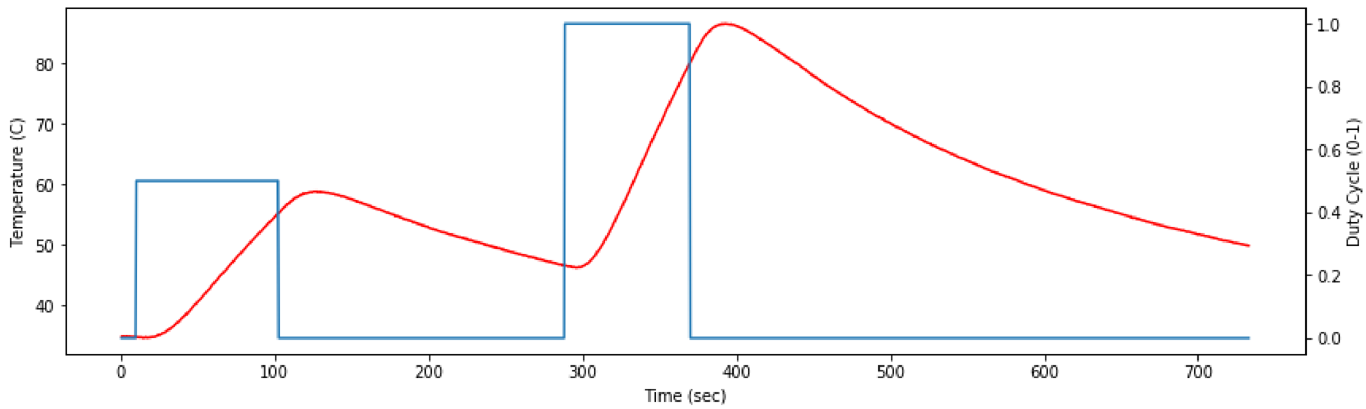
Out[2]:

	Run Time (sec)	Heater Duty (0-1)	Heater Actual (C)	Heater Current (mA)
0	0.495998	0.0	34.77	4.4
1	0.988060	0.0	34.87	5.0
2	1.489019	0.0	34.84	4.8
3	2.002396	0.0	34.84	4.8
4	2.486007	0.0	34.80	4.8
...	...	...	...	...
1461	731.048722	0.0	49.96	4.6
1462	731.556676	0.0	49.87	4.8
1463	732.049742	0.0	49.94	4.8
1464	732.550566	0.0	49.88	5.2
1465	733.049318	0.0	49.86	5.2

1466 rows × 4 columns

In [50]:
```
# Plot pulse response to duty cycle
fig, ax1 = plt.subplots()
fig.set_size_inches(14,4)
ax2 = ax1.twinx()
ax1.plot(data['Run Time (sec)'], data['Heater Actual (C)'], c='r')
ax2.plot(data['Run Time (sec)'], data['Heater Duty (0-1)'])
ax1.set_xlabel('Time (sec)')
ax1.set_ylabel('Temperature (C)')
ax2.set_ylabel('Duty Cycle (0-1)')
print('Heater Impulse Test')
```

```
Heater Impulse Test
```



Basic 1st order model:

$$\tau p \frac{dy}{dt} = -y(t) + kp * u(t - \theta p)$$

where y(t) is temepraturе and u(t) is duty cycle (with delay of θp)

Discretized (Euler) for finding next temperature point:

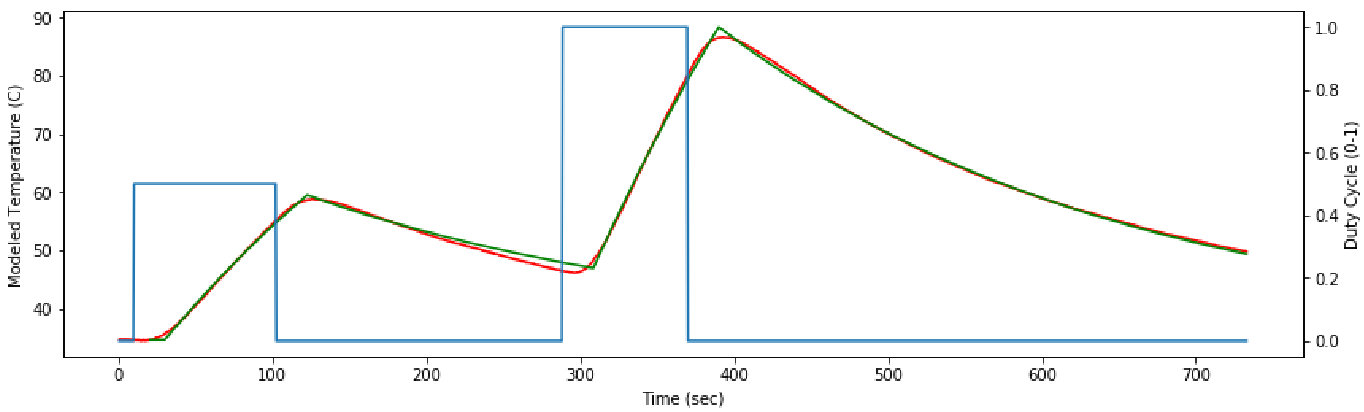
$$y(t) = (-y(t-1) + kp * u(t - \theta p)) \frac{\Delta T}{\tau p} + y(t-1)$$

Note: Does not use actual starting and ambient temperature, only changes - rebaseline after solve

In [93]:
```
# discretized 1st order model, must use delayed duty cycle as input, returns current temperature
def htr_temp_1stOrder(tempm1, duty, time, timem1, kp, taup):
    return ((-tempm1 + (kp * duty)) * (time - timem1) / taup) + tempm1
```

In [94]:
```
# Plot optimized parameters from H. Wang, kp = 168.5, taup = 265.5
# create 20 sec time offset in arrays
times = data['Run Time (sec)']
times = times[:-40]
dutys = data['Heater Duty (0-1)']
dutys = dutys[:-40]
Xdata = (times.to_numpy(), dutys.to_numpy())
# calculate temps, adding baseline temperature after
temps = temps = np.zeros(times.size)
for i in np.arange(times.size-1):
        temps[i+1] = htr_temp_1stOrder(temps[i], dutys[i], times[i+1], times[i], 168.5, 265.5)
temps += data['Heater Actual (C)'][40]
# Plot modeled response
fig, ax1 = plt.subplots()
fig.set_size_inches(14,4)
ax2 = ax1.twinx()
ax1.plot(data['Run Time (sec)'], data['Heater Actual (C)'], c='r')
ax1.plot(data['Run Time (sec)'][40:], temps, c='green')
ax2.plot(data['Run Time (sec)'], data['Heater Duty (0-1)'])
ax1.set_xlabel('Time (sec)')
ax1.set_ylabel('Modeled Temperature (C)')
ax2.set_ylabel('Duty Cycle (0-1)')
print('1st Order Model (green)')
```

```
1st Order Model (green)
```



Basic first principles, 1st order model:

$$mC\frac{dT(t)}{dt} = -hA(T(t) - Tamb) + Phtr(duty(t - delay))$$

where T is temeprature, mC is constant related to specific heat, hA is a constant related to covection loss, Tamb is ambient baseline temperature, Phtr is the power provided by the heating element, and duty is the heater duty cycle (0-1)

Note: mC, hA ($\tau \approx \frac{mC}{hA}$), and Phtr are to be optimized with assumed ambient of 20.0C, the delay due to geometrical and probe physics will be explored with a simple delay and a modified linear ramp of the duty cycle

Discretized (Euler) for finding next temperature point:

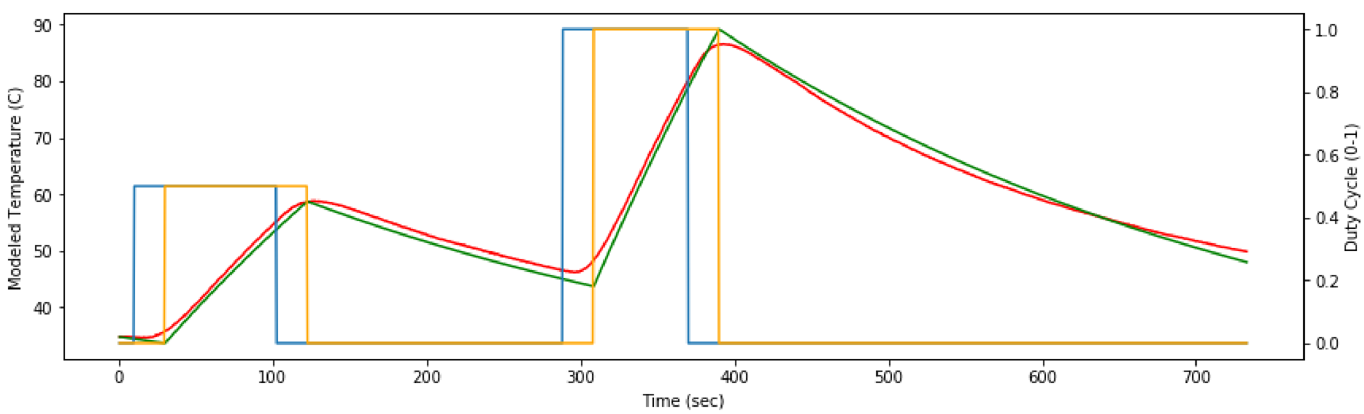
$$T(t) = (-hA(T(t-1) - Tamb) + Phtr(duty(t-1-delay)))\frac{\Delta T}{mC} + T(t-1)$$

In [120]:
```
# discretized 1st order model, must use delayed duty cycle as input, returns current temperature
# m1 indicates prior datapoint
def htr_temp_1stPrinc(tempm1, duty, time, timem1, mC, hA, Phtr):
    Tamb = 20.0
    return (( -hA * (tempm1 - Tamb) + Phtr * duty) * (time - timem1) / mC) + tempm1

# wrapper for using the curve_fit function, X is a 2D array of time, duty cycle (pre delayed),
# and a starting temperature (only a scalar, but sent as array to match size)
def htr_temp_1stPrinc_op(X, mC, hA, Phtr):
    times = Xdata[0]
    dutys = Xdata[1]
    temps = np.zeros(times.size)
    temps[0] = Xdata[2][0]
    for i in np.arange(temps.size-1):
        temps[i+1] = htr_temp_1stPrinc(temps[i], dutys[i], times[i+1], times[i], mC, hA, Phtr)
    return temps
```

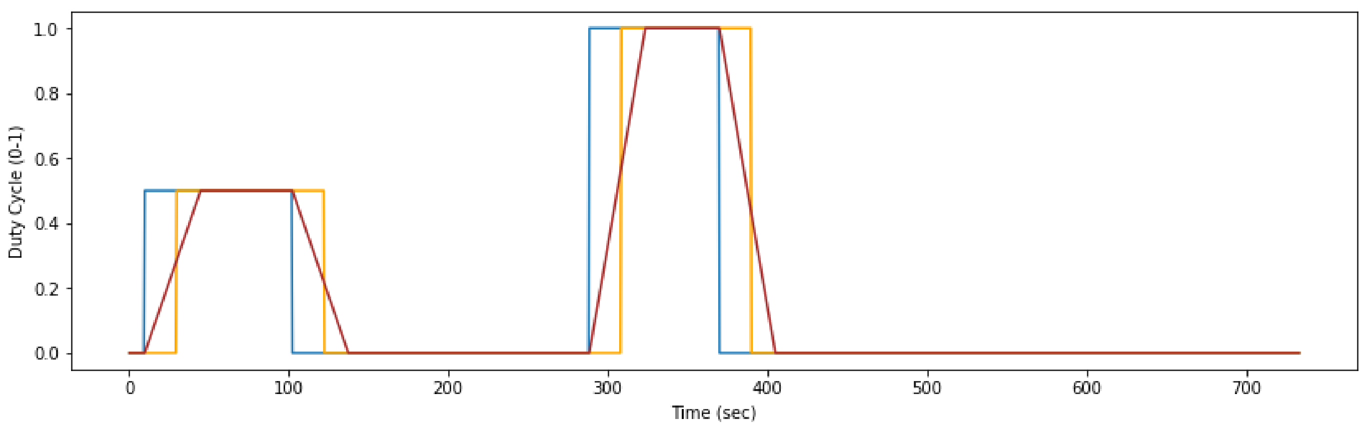
In [123]:
```
# optimize model paratmers of the 1st principles moodel and plot
# using simple time delayed duty cycle - 20sec delay
duty_delay = np.zeros(data['Heater Duty (0-1)'].size)
duty_delay[40:] = data['Heater Duty (0-1)'][:-40]
# prep an array to store the starting temperature
init_temps = np.zeros(data['Heater Duty (0-1)'].size)
init_temps[0] = data['Heater Actual (C)'][0]
# perform curve_fit optimization
Xdata = (data['Run Time (sec)'].to_numpy(), duty_delay, init_temps)
popt, pcov = curve_fit(htr_temp_1stPrinc_op, Xdata, data['Heater Actual (C)'].to_numpy())
print('mC = {0}'.format(popt[0]))
print('hA = {0}'.format(popt[1]))
print('Phtr = {0}'.format(popt[2]))
print('Tau = {0}'.format(popt[0]/popt[1]))
# calculate response
temps = htr_temp_1stPrinc_op(Xdata, popt[0], popt[1], popt[2])
# Plot modeled response
fig, ax1 = plt.subplots()
fig.set_size_inches(14,4)
ax2 = ax1.twinx()
ax1.plot(data['Run Time (sec)'], data['Heater Actual (C)'], c='r')
ax1.plot(data['Run Time (sec)'], temps, c='green')
ax2.plot(data['Run Time (sec)'], data['Heater Duty (0-1)'])
ax2.plot(data['Run Time (sec)'], duty_delay, c='orange')
ax1.set_xlabel('Time (sec)')
ax1.set_ylabel('Modeled Temperature (C)')
ax2.set_ylabel('Duty Cycle (0-1)')
print('1st Principles Model (green) with Offset Delay')
```

```
mC = 390.57962904019723
hA = 1.0301923358668996
Phtr = 266.62676211515185
Tau = 379.13272642581563
1st Principles Model (green) with Offset Delay
```



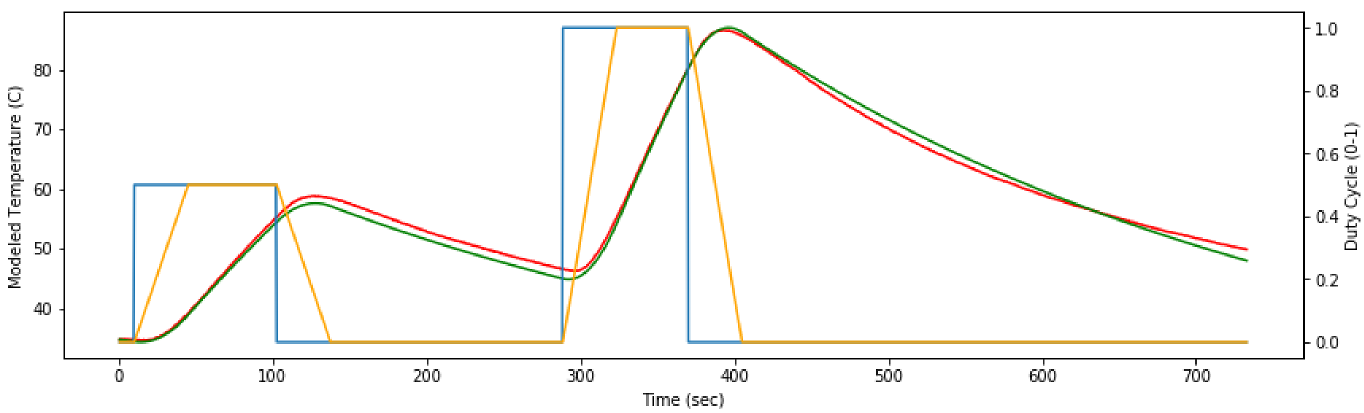
In [129]:
```python
# Create a ramped duty cycle delay to mimic heat transfer to the block/probe
duty_adj = np.zeros(data['Heater Duty (0-1)'].size)
target_duty = 0.0
last_duty = 0.0
point_delay = 70 # 2 x seconds
for i in np.arange(duty_adj.size-1):
    if target_duty != data['Heater Duty (0-1)'][i]:
        last_duty = target_duty
        target_duty = data['Heater Duty (0-1)'][i]
    if target_duty > last_duty:
        duty_adj[i+1] = duty_adj[i] + ((target_duty - last_duty) / point_delay)
        if duty_adj[i+1] > target_duty:
            duty_adj[i+1] = target_duty
    elif target_duty < last_duty:
        duty_adj[i+1] = duty_adj[i] + ((target_duty - last_duty) / point_delay)
        if duty_adj[i+1] < target_duty:
            duty_adj[i+1] = target_duty
# plot duty cycles
fig, ax1 = plt.subplots()
fig.set_size_inches(14,4)
ax1.plot(data['Run Time (sec)'], data['Heater Duty (0-1)'])
ax1.plot(data['Run Time (sec)'], duty_delay, c='orange')
ax1.plot(data['Run Time (sec)'], duty_adj, c='brown')
ax1.set_xlabel('Time (sec)')
ax1.set_ylabel('Duty Cycle (0-1)')
print('Modified Duty Cycles with Delay')
```

```
Modified Duty Cycles with Delay
```



In [130]:
```
# optimize model paratmers of the 1st principles moodel and plot
# using ramped time delayed duty cycle
# perform curve_fit optimization
Xdata = (data['Run Time (sec)'].to_numpy(), duty_adj, init_temps)
popt, pcov = curve_fit(htr_temp_1stPrinc_op, Xdata, data['Heater Actual (C)'].to_numpy())
print('mC = {0}'.format(popt[0]))
print('hA = {0}'.format(popt[1]))
print('Phtr = {0}'.format(popt[2]))
print('Tau = {0}'.format(popt[0]/popt[1]))
# calculate response
temps = htr_temp_1stPrinc_op(Xdata, popt[0], popt[1], popt[2])
# Plot modeled response
fig, ax1 = plt.subplots()
fig.set_size_inches(14,4)
ax2 = ax1.twinx()
ax1.plot(data['Run Time (sec)'], data['Heater Actual (C)'], c='r')
ax1.plot(data['Run Time (sec)'], temps, c='green')
ax2.plot(data['Run Time (sec)'], data['Heater Duty (0-1)'])
ax2.plot(data['Run Time (sec)'], duty_adj, c='orange')
ax1.set_xlabel('Time (sec)')
ax1.set_ylabel('Modeled Temperature (C)')
ax2.set_ylabel('Duty Cycle (0-1)')
print('1st Principles Model (green) with Ramped Delay')
```

```
mC = 26.73373406013599
hA = 0.07029247648681476
Phtr = 18.244396853779378
Tau = 380.321414129585
1st Principles Model (green) with Ramped Delay
```



In [22]:
```
# check current at pulse conditions
print('50% duty : ' + str(data[data['Heater Duty (0-1)'] == .5].mean()))
print('100% duty: ' + str(data[data['Heater Duty (0-1)'] == 1.0].mean()))
```

```
50% duty : Run Time (sec)          56.020046
Heater Duty (0-1)         0.500000
Heater Actual (C)        42.983243
Heater Current (mA)    3260.282162
dtype: float64
100% duty: Run Time (sec)         329.047551
Heater Duty (0-1)         1.000000
Heater Actual (C)        59.597607
Heater Current (mA)    6360.564417
dtype: float64
```

In [ ]: