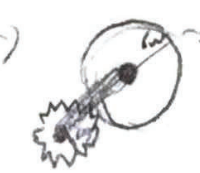
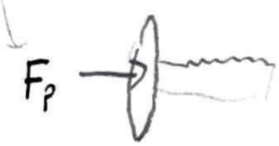
MEEN 2450 Ryan Dalby u0848407 HID02

Modelling Propulsion Force



$$P(t) = \text{constant} = P_o = 50.0\,psig\left(\frac{6894.76\,Pa}{1\,psig}\right) = 344738\,Pa$$



$F_p \rightarrow$

$\underline{\tau_g = \tau_w}$ $F_p = P_o A_{piston} = (344738\,Pa)(\pi(\overset{r_p}{10\times10^{-2}m})^2)$

$\tau_g = F_{gear}\, r_g = \tau_w = F_{wheel}\, r_{w}$ $F_p = F_{gear} = 10830.3\ N$

$(10830.3)(1\times10^{-2}m) = (F_{wheel})(2.5\times10^{-2}m)$

$\boxed{F_{wheel} = 4332.1\ N}$ $F_{wheel} = \dfrac{P_o\boxed{\pi(r_p)^2}r_g}{r_w}$

Modelling the Tram Motion

$m\dfrac{dx^2}{dt^2} = F_t - F_d - F_r$ → Steady state velocity means $\dfrac{dv}{dt} = 0$ thus $\dfrac{dx^2}{dt^2} = 0$

$0 = F_t - F_d - F_r$

$0 = F_t - \frac{1}{2}\rho A C_d V_{tr}^2 - C_r m g$

$\underline{\dfrac{P_o[\pi(r_p)^2]r_g}{r_w}} = F_t = F_{wheel}$ → Force on each wheel is same since connected to same axle t = cylinder thickness

$\rho_{air} = 1.205\ kg/m^3$ → Density of air approx. 20 °C dry air

$\underline{A = \pi(r_p + t_c)^2}$ $\underline{C_d = 0.82}$ → Coefficient of drag long cylinder

$\underline{V_{tr} = ?}$ $\underline{C_r = 0.03}$ $m = 5\,kg$ approx. $g = 9.81\,m/s^2$

↓ Axle weight + wheel weight + PVC weight + Piston weight

$$0 = \frac{P_o[\pi(r_p)^2]r_g}{r_w} - \frac{1}{2}\rho A C_d V_{tr}^2 - C_r m g$$

$$\boxed{V_{tr} = \sqrt{\frac{\left[\frac{P_o(\pi(r_p)^2)r_g}{r_w} - C_r m g\right](2)}{\rho A C_d}}}$$

ME EN 2450
Ryan Dalby u0848407
AID02

Modeling Train Motion:
Bisection Method- answer: 529.320240020752 m/s Iterations: 21 eT:5.738753580522209e-05%
Modified Secant Method: answer: 529.3199363664439 m/s Iterations: 14
eT:2.0660456783705307e-08%
Mullers Method- answer: 529.3197848462762 m/s Iterations: 9 eT:2.8604780857741632e-05%

Here we see that Muller's method converged in the least amount of iterations. The termination criteria for all methods was the approximate relative error. A closed form solution was then found so the true relative error eT could be found. Although eT isn't the lowest for Muller's method it went through fewer iterations. The value of eT is likely situationally dependent on how the method converges for different problems.

Propulsion System Design:
v was found using mullers method

	P0(Pa)	m(kg)	rp(m)	rg(m)	v(m/s)	iterations	True v(m/s)	True Relative Error(%)
0	137895.2	1	0.05	0.050	748.647368	12.0	748.647736	0.000049
1	137895.2	10	0.05	0.005	235.145315	13.0	235.145369	0.000023
2	275790.4	3	0.20	0.005	334.806790	13.0	334.806867	0.000023
3	413685.6	1	0.20	0.050	1296.781742	12.0	1296.782179	0.000034
4	413685.6	10	0.05	0.005	409.149350	13.0	409.149439	0.000022

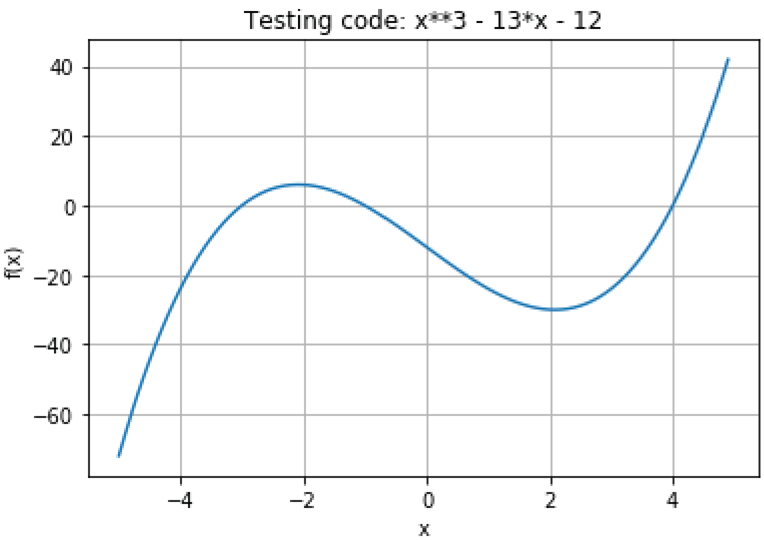
From this chart, we can see that the model we made is not very good since the velocity values are not realistic for a small train. Our numerical methods were verified by comparing to the closed form solution which had close values. We cannot validate our model though since the answers are very unrealistic thus the equations/and or assumptions to get the equations we used likely do not model the train motion very well.

To make this model we likely cannot assume steady-state conditions and we also must more accurately find mass and volume/shape of our train. We should also better analyze the piston system for how it will translate to a force and determine other losses of energy from the train moving.

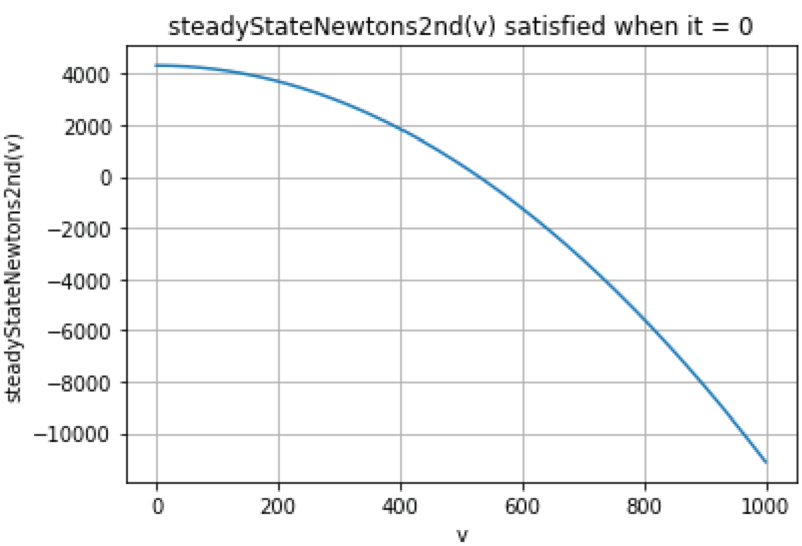
```
Python 3.6.5 |Anaconda, Inc.| (default, Mar 29 2018, 13:32:41) [MSC v.1900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 6.4.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/AID2/AID2.py',
wdir='C:/Users/hoops/OneDrive/Documents/School/ME EN 2450 Numerical Methods/AID2')
Test root finding methods:
Bisection Method:  (4.000007629394531, 17)
Modified Secant Method:  (-3.000000011054994, 13)
Mullers Method:  (-0.9999952830196684, 13)
```




```
Modeling Train Motion:
Bisection Method- answer: 529.320240020752 m/s Iterations: 21 eT:5.738753580522209e-05%
Modified Secant Method: answer: 529.3199363664439 m/s Iterations: 14 eT:2.0660456783705307e-08%
Mullers Method- answer: 529.3197848462762 m/s Iterations: 9 eT:2.8604780857741632e-05%
```




```
Propulsion System Design:
v was found using mullers method
     P0(Pa)  m(kg)  rp(m)  rg(m)       v(m/s)  iterations  True v(m/s)  True Relative Error(%)
0  137895.2      1   0.05  0.050   748.647368        12.0   748.647736                0.000049
```

1	137895.2	10	0.05	0.005	235.145315	13.0	235.145369	0.000023
2	275790.4	3	0.20	0.005	334.806790	13.0	334.806867	0.000023
3	413685.6	1	0.20	0.050	1296.781742	12.0	1296.782179	0.000034
4	413685.6	10	0.05	0.005	409.149350	13.0	409.149439	0.000022

In [2]:

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Feb 27 20:29:35 2019
AID 02
@author: Ryan Dalby
"""

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

def bisection(func, a, b, tol=1e-6, maxiter=10):
    """
    Given a function, bounds(a and b) for which a root exists between, a number of max ite
    will return the value of the root(either after max iterations or after tolerence has b
    """
    if(func(a)*func(b) >= 0): #Means that either a or b is a root or there is
        if(func(a) == 0):
            return a
        elif(func(b) == 0):
            return b
        else:
            raise ValueError("a and b do not bracket a root")

    currentA = a #the current a postion
    currentB = b #the current b postion
    lastC = 0 #last iteration's root esitmate
    i = 0 # loop counter
    while(i < maxiter):
        c = (currentA + currentB) / 2
        #here we move either a or b to c
        if(func(currentA)*func(c) > 0): #if the function at a and c are the same sign then
            currentA = c
        elif(func(currentA)*func(c) < 0): #if the function at a and c are different signs
            currentB = c
        else: #else the function at c is 0 and c is the root
            return (c, i)
        Ea = ((c - lastC) / c) * 100 #this is the current approximate relative error in pe
        lastC = c #set lastC for next loop
        i += 1 #set counter value for next loop

        if(abs(Ea) < tol):#before enetering loop again will check if we have reached the t
            return (lastC, i)

    raise RuntimeError("Root was not found within approximate relative error with the max

def modifiedSecant(f, x0, delta, tol, maxiter):
    """
    Given a function f, an initial guess for the root,a delta(step size), a tolerence(appr
    """
    i = 1
```

```
    x = x0
    lastX = x0
    while i < maxiter:
        x -= (delta*x*f(x)) / (f(x+x*delta) - f(x))
        eA = ((x - lastX) / x) * 100
        if abs(eA) < tol: #root found
            return (x, i)
        lastX = x
        i+=1

    raise RuntimeError("Root could not be found") #Fall through case where maxiters was re

def mullersMethod(f, xInit, h, tol, maxiter):
    """
    Given a function f, an initial guess for the root, a step size, a tolerence(approximat

    """
    #intial values of x0, x1, x2, offset left and right by h * initial guess
    x2 = xInit
    x1 = xInit + h * xInit
    x0 = xInit - h * xInit

    xLast = x2
    i = 1

    while i < maxiter:
        h0 = x1 - x0
        h1 = x2 - x1
        d0 = (f(x1) - f(x0)) / (h0)
        d1 = (f(x2) - f(x1)) / (h1)
        a = (d1 - d0) / (h1 + h0)
        b = a * h1 + d1
        c = f(x2)
        root = np.sqrt(b**2 - 4*a*c)

        if(np.abs(b + root) > np.abs(b - root)):
            denom = b + root
        else:
            denom = b - root

        xNew = x2 + (-2.0*c) / (b + denom)
        eA = ((xNew - xLast) / xNew) * 100
        if(eA < tol):
            return (xNew, i)
        xLast = xNew
        x0 = x1
        x1 = x2
        x2 = xNew
        i+=1
```

```python
    raise RuntimeError("Root could not be found") #Fall through case where maxiters was re

def steadyStateNewtons2nd(P0, m, rp, rg, v):
    """
    Returns value of steady state newtons that is satisfied when it is equal to 0
    """
    rho = 1.2 #density of dry air approx. at 20degC g/m^3
    rw = .025 #radius of train wheels in m
    Cr = .03 #rolling resistance coefficent
    g = 9.81 #gravitational constnt m/s^2
    Cd = 0.82 #coefficent of drag for a long cylinder
    Ft = (P0 * (np.pi * rp**2) * rg) / (rw) #force caused by the piston
    return Ft - 0.5 * rho * (np.pi * rp**2)*Cd * v**2 - Cr * m * g

def closedFormSteadyStateNewtons2nd(P0, m, rp, rg):
    """
    Returns value of veclocity from closed form solution
    """
    rho = 1.2 #density of dry air approx. at 20degC g/m^3
    rw = .025 #radius of train wheels in m
    Cr = .03 #rolling resistance coefficent
    g = 9.81 #gravitational constnt m/s^2
    Cd = 0.82 #coefficent of drag for a long cylinder
    Ft = (P0 * (np.pi * rp**2) * rg) / (rw) #force caused by the piston
    return np.sqrt(((Ft - Cr*m*g)*2)/(rho*(np.pi * rp**2)*Cd))

"""
Testing Code
"""
print("Test root finding methods:")
fTest= lambda x: x**3 - 13*x - 12
print("Bisection Method: ",bisection(fTest, 2, 7, .001, 100))
print("Modified Secant Method: ",modifiedSecant(fTest, 2, .01, .001, 100))
print("Mullers Method: ",mullersMethod(fTest, 2, .01, .001, 100))

xVals = np.arange(-5,5, .1)
plt.plot(xVals, fTest(xVals))
plt.grid()
plt.title("Testing code: x**3 - 13*x - 12")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.show()


"""
Modeling Train Motion
"""
print()
print("Modeling Train Motion:")
```

```
P0 = 344738 #Pa
m = 5 #kg
rp = .1 #radius of piston in m
rg = .01 #radius of gear in m

f = lambda v: steadyStateNewtons2nd(P0, m, rp, rg, v)
trueAns = closedFormSteadyStateNewtons2nd(P0, m, rp, rg) #found by hand calculations
bisectionAns = bisection(f, 0, 1000, .0001,100)
secAns = modifiedSecant(f, 1, .01, .0001, 100)
mulAns = mullersMethod(f, 1, .01, .0001, 100)

print("Bisection Method- answer: {} m/s Iterations: {} eT:{}%".format(*bisectionAns, np.ab
print("Modified Secant Method: answer: {} m/s Iterations: {} eT:{}%".format(*secAns, np.ab
print("Mullers Method- answer: {} m/s Iterations: {} eT:{}%".format(*mulAns, np.abs((trueA
vVals = np.arange(0,1000, .1)
plt.plot(vVals, f(vVals))
plt.grid()
plt.xlabel("v")
plt.ylabel("steadyStateNewtons2nd(v)")
plt.title("steadyStateNewtons2nd(v) satisfied when it = 0")
plt.show()

"""
Propulsion System Design
"""
print()
print("Propulsion System Design:")

P0ValsPsig = np.array([20, 20, 40, 60, 60]) #psig
P0Vals = P0ValsPsig  * 6894.76 #Pa
mVals = np.array([1, 10, 3, 1, 10]) #kg
rpVals = np.array([.05, .05, .2, .2, .05]) #m
rgVals = np.array([.05, .005, .005, .05, .005]) #m
vVals = np.empty_like(P0Vals)
iterations = np.empty_like(P0Vals)

for i in range(np.size(P0Vals)):
    fDesign = lambda v: steadyStateNewtons2nd(P0Vals[i], mVals[i], rpVals[i], rgVals[i], v
    vVals[i], iterations[i] = mullersMethod(fDesign, 100, .01, .0001, 100)

trueVVals = closedFormSteadyStateNewtons2nd(P0Vals, mVals, rpVals, rgVals) #found by hand
eT = abs((trueVVals - vVals)/(trueVVals))*100
testVals = pd.DataFrame({'P0(Pa)':P0Vals.tolist(), 'm(kg)':mVals.tolist(), 'rp(m)':rpVals.
                         'rg(m)':rgVals.tolist(), 'v(m/s)':vVals.tolist(), 'iterations':it
                         'True v(m/s)':trueVVals,'True Relative Error(%)':eT})

print("v was found using mullers method")
print(testVals.to_string())
```