Ay 190 Assignment 5

Root Finding

January 29, 2013

1 Eccentricity Anomality

1.1 Earth's orbit

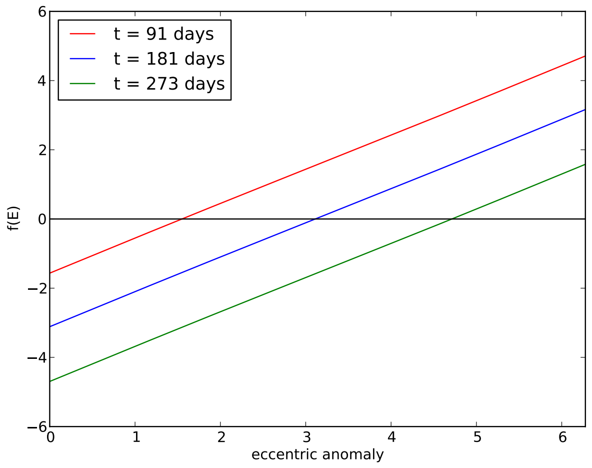

Figure 1: The function $f(E) = E - \omega t - e\sin(E)$, where e=0.0167 and $\omega = 2\pi/1year$, (i.e. Earth's orbit) for the times $t = 91$, 182, and 273 days. The roots of these functions give the eccentricity anomality at each of the given times.

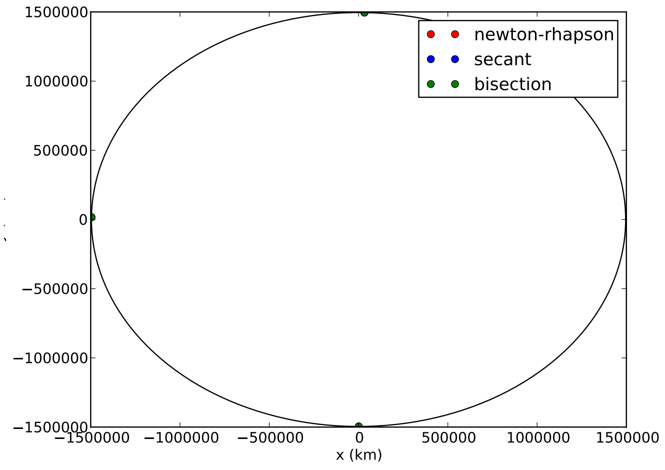

Figure 2: Position of Earth calcluated with various root finding methods at times t = 91, 181, and 273 days (going counterclockwise). The differences in the results from various root finding methods are so small the results from different methods are overplotted directly on top of each other.

1.2 Eccentric Earth orbit

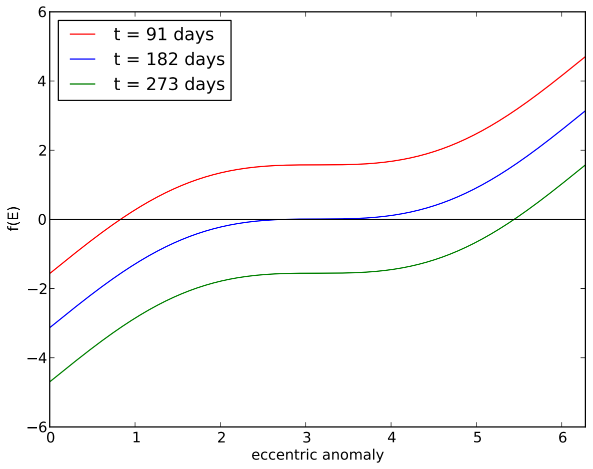

Figure 3: The function whose root to solve for at t = 91 days for the scenario in which Earth is in an eccentric orbit (e=0.99999).

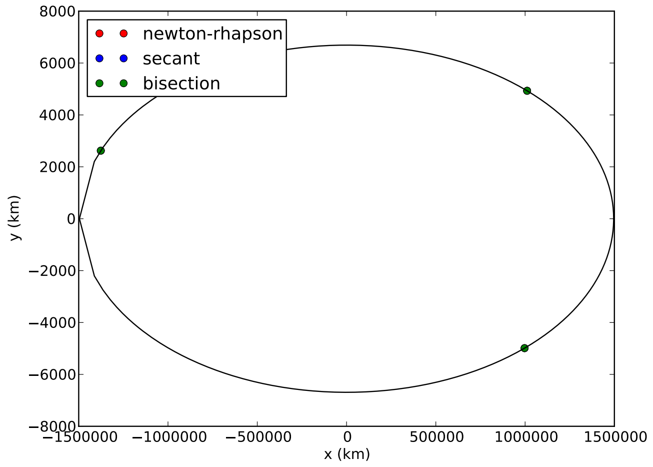

Figure 4: Results calcluated with various root finding methods for scenario in which Earth is in an eccentric orbit.

1.3 Comparison

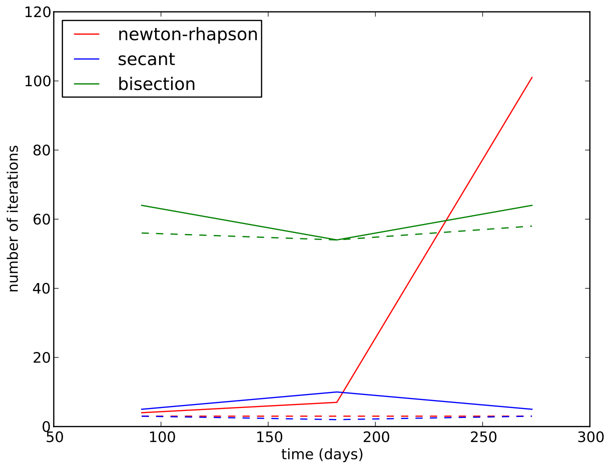

Figure 5: Comparison of the number of iterations required for convergence for the realistic and eccentric Earth orbits. The number of iterations required for the eccentric orbit is plotted with a solid line and the iterations required for the realistic orbit with a dashed line. The secant method performs the best, requiring 2-3 iterations for the realistic orbit and 5-10 iterations for the eccentric orbit. The number of iterations required using the bisection method was higher at around 54-58 and 54-64 iterations for the realistic and eccentric orbits, respectively (interestingly, no increase in iteraitons was required for t=181 d). The Newton-Rhapson method required only 3 iterations for the realistic orbit but 101 iterations for the eccentric orbit at t=273 d.

2 Polynomials with Multple Roots

To find all the roots of a given polynomial, one can find a root a_i of the given polynomial by any of the above root-finding methods, divide the polynomial by the corresponding factor $(x - a_i)$, and then recurse on the resulting reduced polnomial until the originla polynomial has been completely factored. An implementation of this algorithm is given below (also reproduced in Appendix A).

```
def find_all_roots(coeff0):
    """
    Computes all integer or rational roots of the given function
    f = sum([coeff[i]*x**i for i=0..len(coeff)]).
    """
    coeff=coeff0

    roots=[]
    while len(roots) < len(coeff0)-1:
        # find zero of polynomial
        f  = lambda x: sum([coeff[i] * x**i for i in range(len(coeff))])
        root,err,niter=secant(0.0,1.0,f,1e-15)
        roots.append(root)

        #synthetic division
        a=coeff[-1]
        for i in range(len(coeff)-2,-1,-1): coeff[i]=a=coeff[i]+a*root
        if not (abs(coeff[0]) <= 1e-10):
            print 'found false root x =',root,'! (',abs(coeff[0]),')'
            sys.exit()
        coeff=coeff[1:]

    return roots
```

This algorithm was tested on the function $f(x) = 3x^5 + 5x^4 - x^3$ (plotted below), which gave the correct roots 0 (multiplicity 3), 0.1804604217163715, and -1.8471270883830382.

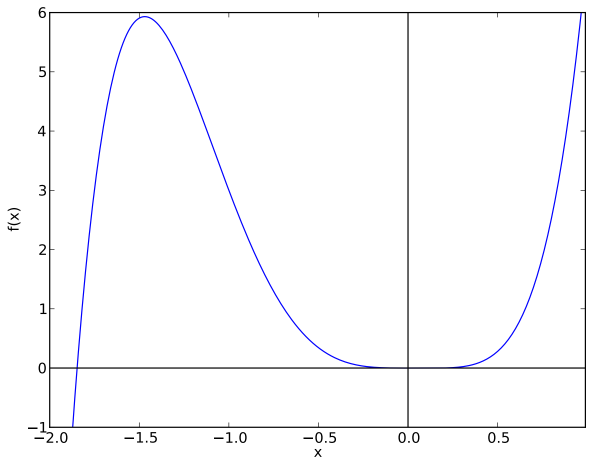

Figure 6: The polynomial $f(x) = 3x^5 + 5x^4 - x^3$, whose roots were computed using the above algorithm.

Appendices

The Python modules and scripts used in this assignment include

A The Root-Finding Routines: `findroot.py` (see pages 4-6)
B Script for Set 5: `set5.py` (see pages 7-9)

A The Root-Finding Routines: findroot.py

```
# findroot.py
# created 11/17/11 by stacy kim
#
# Contains a suite of routines to find roots that implement the methods
#    Newton-Raphson
#    secant
#    bisection.
#
#
# modified 1/26/13: removed support to output to file, instead returning array
# modified 1/28/13 to return number of iterations required to find solution
# modified 1/29/13: added method to find all rational roots to a polynomial


import sys
import math


# FILE OUTPUT ROUTINES ------------------------------------------------------------

files=None

def set_output(h,c,fns):
    """
    Enables output to files and sets file names for all 3 methods to those
    given in the array fns, in the order bisection, newton_raphson, secant.
    """
    global files

    # Open output files and write headers
    files=[0]*3

    for i in range(3):
        files[i]=open(fns[i],'w')
        files[i].write(fns[i])
        files[i].write('\nh={0},c={1},err=1e-4\n{2}{3}\n'.format(h,c,'x'.rjust(19),'precision'.rjust(19


def close_files():
    """Closes output files, if any."""
    if (files==None): return

    print "Finished writing output files."
    for i in range(3):
        files[i].close()


# ROOT-FINDING ROUTINES --------------------------------------------------------

def bisection(x1,x2,f,err):

    if (f(x1)*f(x2) > 0):
        print 'bisection: f(x1) and f(x2) must have opposite signs!\n' \
              '           f({0})={1},f({2})={3}'.format(x1,f(x1),x2,f(x2))
        sys.exit()

    if (x2==min(x1,x2)): x1,x2 = x2,x1

    count=0
    x=(x1+x2)/2.0
    while(abs(f(x)) > err):
        count+=1
        if (f(x)*f(x1)>0): x1=x
        else: x2=x
        x=(x1+x2)/2.0

        count+=1
        if (count>=990):
            print 'x1={0},x2={1},x={2}'.format(x1,x2,x)
            if (count==1000):
                print 'Failed to converge after 1000 iterations.'
                sys.exit()

    return x,count


def newton_raphson(x,f,df,err):
    count=0
    while(abs(f(x)) > err):
        x=x-f(x)/df(x)
        count+=1
    return x,abs(f(x)/df(x)),count


def secant(x2,x,f,err):
    count=0
    while(abs(f(x)) > err):
        x1=x2
        x2=x
        df=(f(x2)-f(x1))/(x2-x1)
        x=x2-f(x2)*(x2-x1)/(f(x2)-f(x1))
        count+=1

    df=(f(x2)-f(x))/(x2-x)
    return x,abs(f(x)/df),count
```

```
def find_all_roots(coeff0):
    """
    Computes all integer or rational roots of the given function
    f = sum([coeff[i]*x**i for i=0..len(coeff)]).
    """
    coeff=coeff0

    roots=[]
    while len(roots) < len(coeff0)-1:
        # find zero of polynomial
        f  = lambda x: sum([coeff[i] * x**i for i in range(len(coeff))])
        root,err,niter=secant(0.0,1.0,f,1e-15)
        roots.append(root)

        #synthetic division
        a=coeff[-1]
        for i in range(len(coeff)-2,-1,-1): coeff[i]=a=coeff[i]+a*root
        if not (abs(coeff[0]) <= 1e-10):
            print 'found false root x =',root,'! (',abs(coeff[0]),')'
            sys.exit()
        coeff=coeff[1:]

    return roots
```

B Script for Set 5

```
# set5.py
# created 1/23/13 by stacy kim

import numpy as np, math, sys
import matplotlib.pyplot as plt
from findroot import *

# EXERCISE 1 ------------------------------------------------------------------
# Root Finding: Eccentricity Anomality

# orbital parameters of Earth
e=0.0167
a=1.496e6 # km
b=a*math.sqrt(1-e*e)
P=365.25635
omega=2*math.pi/P # angular velocity
t=91.0

f  = lambda E: E-omega*t+e*math.sin(E)
df = lambda E: 1+e*math.cos(E)

# plot the function at t=91.0 d
EE=np.arange(0,2*math.pi,2*math.pi/1e3)
plt.plot(EE,[f(ee) for ee in EE],'r')
t=181.0
plt.plot(EE,[f(ee) for ee in EE],'b')
```

```
t=273.0
plt.plot(EE,[f(ee) for ee in EE],'g')
plt.legend(['t = 91 days','t = 181 days','t = 273 days'],loc='best')
plt.plot([0,2*math.pi],[0,0],'k')
plt.xlim([EE[0],EE[-1]])
plt.xlabel('eccentric anomaly')
plt.ylabel('f(E)')
plt.savefig('f(91d).pdf')
plt.show()

# solve for position of Earth at 3 timepoints
tt=[91.0, 182.0, 273.0]
E0,err0,cnt0=np.zeros([3,len(tt),3])
for i in range(len(tt)):
    t=tt[i]
    E0[i,0],err0[i,0],cnt0[i,0]=newton_raphson(i*math.pi/2,f,df,1e-10)
    E0[i,1],err0[i,1],cnt0[i,1]=secant(0.95*(i+1)*math.pi/2,1.05*(i+1)*math.pi/2,f,1e-10)
    E0[i,2],cnt0[i,2]=bisection(0.95*(i+1)*math.pi/2,1.05*(i+1)*math.pi/2,f,1e-10)

x=np.array([[a*math.cos(ee) for ee in time] for time in E0])
y=np.array([[b*math.sin(ee) for ee in time] for time in E0])

# solve for position of Earth at many timepoints
full_orbit=[]
for i in np.arange(0,P,P/1e3):
    t=i
    full_orbit.append(secant(-1,7,f,1e-10)[0])

full_x=[a*math.cos(ee) for ee in full_orbit]
full_y=[b*math.sin(ee) for ee in full_orbit]

# plot resulting positions
plt.plot(x[:,0],y[:,0],'ro',x[:,1],y[:,1],'bo',x[:,2],y[:,2],'go')
plt.legend(['newton-rhapson','secant','bisection'],loc='best')
plt.plot(full_x,full_y,'k')
plt.xlabel('x (km)')
plt.ylabel('y (km)')
plt.savefig('earth_pos.pdf')
plt.show()

# plot number of iterations required for solution
plt.plot(tt,cnt0[:,0],'r',tt,cnt0[:,1],'b',tt,cnt0[:,2],'g')
plt.legend(['newton-rhapson','secant','bisection'],loc='best')
plt.xlabel('time (days)')
plt.ylabel('number of iterations')
plt.savefig('niter_orig.pdf')
plt.show()



# HIGHLY ECCENTRIC EARTH ORBIT
e=0.99999
b=a*math.sqrt(1-e*e)
```

```
# plot the function at all t
EE=np.arange(0,2*math.pi,2*math.pi/1e3)
t=91.0
plt.plot(EE,[f(ee) for ee in EE],'r')
t=182.0
plt.plot(EE,[f(ee) for ee in EE],'b')
t=273.0
plt.plot(EE,[f(ee) for ee in EE],'g')
plt.legend(['t = 91 days','t = 182 days','t = 273 days'],loc='best')
plt.plot([0,2*math.pi],[0,0],'k')
plt.xlim([EE[0],EE[-1]])
plt.xlabel('eccentric anomaly')
plt.ylabel('f(E)')
plt.savefig('f(91d)_eccentric.pdf')
plt.show()

# solve for position of Earth at 3 timepoints
tt=[91.0, 182.0, 273.0]
E1,err1,cnt1=np.zeros([3,len(tt),3])
bounds=np.array([[0.0,1.0],[2.0,3.0],[5.0,6.0]])

for i in range(len(tt)):
    t=tt[i]
    E1[i,0],err1[i,0],cnt1[i,0]=newton_raphson(i*math.pi/2,f,df,1e-10)
    E1[i,1],err1[i,1],cnt1[i,1]=secant(bounds[i,0],bounds[i,1],f,1e-10)
    E1[i,2],cnt1[i,2]=bisection(bounds[i,0],bounds[i,1],f,1e-10)

x=np.array([[a*math.cos(ee) for ee in time] for time in E1])
y=np.array([[b*math.sin(ee) for ee in time] for time in E1])

# solve for position of Earth at many timepoints
full_orbit=[]
for i in np.arange(0,P,P/1e3):
    t=i
    full_orbit.append(secant(-1,7,f,1e-10)[0])
full_x=[a*math.cos(ee) for ee in full_orbit]
full_y=[b*math.sin(ee) for ee in full_orbit]

# plot resulting positions
plt.plot(x[:,0],y[:,0],'ro',x[:,1],y[:,1],'bo',x[:,2],y[:,2],'go')
plt.legend(['newton-rhapson','secant','bisection'],loc='best')
plt.plot(full_x,full_y,'k')
plt.xlabel('x (km)')
plt.ylabel('y (km)')
plt.savefig('earth_pos_eccentric.pdf')
plt.show()

# plot number of iterations required for solution
plt.plot(tt,cnt1[:,0],'r',tt,cnt1[:,1],'b',tt,cnt1[:,2],'g')
plt.plot(tt,cnt0[:,0],'r--',tt,cnt0[:,1],'b--',tt,cnt0[:,2],'g--')
plt.legend(['newton-rhapson','secant','bisection'],loc='best')
plt.xlabel('time (days)')
plt.ylabel('number of iterations')
plt.savefig('niter_orig_eccentric.pdf')
plt.show()

# print number of iterations required for solution (realisitc & eccentric orbits)
print 'iterations required'
print 't= 91d\t', cnt0[:,0],'\n      \t',cnt1[:,0]
print 't=181d\t', cnt0[:,1],'\n      \t',cnt1[:,1]
print 't=273d\t', cnt0[:,2],'\n      \t',cnt1[:,2]

# EXERCISE 2 ----------------------------------------------------------------
# Root Finding: Polynomials with Multiple Roots
coeff=[0,0,0,-1,5,3] # f = sum([coeff[i] * x**i for i=0..len(coeff)])
f = lambda x: sum([coeff[i] * x**i for i in range(len(coeff))])

# plot the function
x=np.arange(-2,1,0.01)
y=[f(xx) for xx in x]
plt.plot(x,y,[x[0],x[-1]],[0,0],'k',[0,0],[-1,6],'k')
plt.xlim([x[0],x[-1]])
plt.ylim([-1,6])
plt.xlabel('x')
plt.ylabel('f(x)')
plt.savefig('polynomial_plot.pdf')
plt.show()

# calclate all roots of the polynomial
print 'roots =',find_all_roots(coeff)
```