# Phys 20 Lab 6 - Root Finding

Chris Dudiak

June 1, 2013

## 1 Part 1 - Golden Ratio Convergence of Secant Method

Within a small distance $\epsilon$ of x, the function is approximately:

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + \epsilon^2 \frac{f''(x)}{2} + ...$$

Evaluating at $x_1$ and $x_2$ and plugging into the step relation for the Secant Method

$$f(x_3) = x_2 - f(x_2) \frac{x_2 - x_1}{f(x_2) - f(x_1)}$$

we get the following:

$$f(\epsilon_3) = \epsilon_2 - f(x_2) \frac{\epsilon_2 - \epsilon_1}{f(x_2) - f(x_1)}$$

When a trial solution $x_i$ differs from the true root by $\epsilon$ we get:

$$\epsilon_3 = \epsilon_2 - f(x_2+\epsilon) \frac{\epsilon_2 - \epsilon_1}{f(x_2+\epsilon_2) - f(x_1+\epsilon)}$$

Since $f(x_i)$ is zero, we get an approximation (neglecting higher order terms):

$$f(x_i+\epsilon_i) \approx \epsilon_2 f'(x_i) + \epsilon_i^2 \frac{f''(x_i)}{2}$$

Plugging this in, we get:

$$\epsilon_3 \approx \epsilon_2 - (\epsilon_2 f'(x_2) + \epsilon_2^2 \frac{f''(x_2)}{2}) \frac{\epsilon_2 - \epsilon}{\epsilon_2 f'(x_2) + \epsilon_2^2 \frac{f''(x_2)}{2} - \epsilon f'(x_1) + \epsilon^2 \frac{f''(x_1)}{2}}$$

Simplifying the denominator, we find:

$$\begin{aligned} f(x_2+\epsilon_2) - f(x_1+\epsilon) &\approx \epsilon_2 f'(x_2) + \epsilon_2^2 \frac{f''(x_2)}{2} - \epsilon f'(x_1) - \epsilon^2 \frac{f''(x_1)}{2} \\ &\approx f'(x_2) * (\epsilon_2 - \epsilon) * (1 + \frac{f''(x_2)}{2f'(x_2)}(\epsilon_2 + \epsilon)) \end{aligned}$$

Which gives:

$$\begin{aligned} \epsilon_3 &= \epsilon_2 - \frac{(\epsilon_2) * (\epsilon_2 - \epsilon) * (f'(x_2) + \epsilon_2 f''(x_2))}{f'(x_2) * (\epsilon_2 - \epsilon) * (1 + \frac{f''(x_2)}{2f'(x_2)}(\epsilon_2 + \epsilon))} \\ &= \epsilon_2 - \frac{(\epsilon_2) * (1 + \epsilon_2 \frac{f''(x_2)}{f'(x_2)})}{(1 + \frac{f''(x_2)}{2f'(x_2)}(\epsilon_2 + \epsilon))} \\ &= \frac{\epsilon_2(1 + \frac{f''(x_2)}{2f'(x_2)}(\epsilon_2 + \epsilon)) - (\epsilon_2)(1 + \epsilon_2 \frac{f''(x_2)}{f'(x_2)})}{(1 + \frac{f''(x_2)}{2f'(x_2)}(\epsilon_2 + \epsilon))} \\ &= \frac{\epsilon_2 \epsilon \frac{f''(x_2)}{2f'(x_2)}}{(1 + \frac{f''(x_2)}{2f'(x_2)}(\epsilon_2 + \epsilon))} \\ &\approx \epsilon_2 \epsilon \frac{f''(x_2)}{2f'(x_2)} \end{aligned}$$

Now, assume $\epsilon_{i+1} = C\epsilon_i^r$ for all i where C and r are constants independent of i. Plugging this into the recurrence, we find:

$$\begin{aligned} C\epsilon_2^r &\approx \epsilon_2 \epsilon \frac{f''(x_2)}{2f'(x_2)} \\ \epsilon_2^{r-1} &\approx \epsilon \frac{f''(x_2)}{2Cf'(x_2)} \\ \epsilon_2 &\approx \epsilon^{\frac{1}{r-1}} \frac{f''(x_2)}{2Cf'(x_2)}^{\frac{1}{r-1}} \end{aligned}$$

So $C = \frac{f''(x_2)}{2Cf'(x_2)}^{\frac{1}{r-1}}$ and $r = \frac{1}{r-1}$ from the assumption. Solving for the positive r, we find the convergence rate is:

$$\begin{aligned} r &= \frac{1}{r-1} \\ r^2 - r - 1 &= 0 \\ r &= \frac{1 + \sqrt{5}}{2} \end{aligned}$$

Which is the Golden Ratio.

## 2 Method Implementations

The following is the code for the three method implementations:

```
def bisection(f, x1, x2):
    acc = abs(x1 - x2)
    steps = []
    x0 = (x1 + x2) / 2.0
    while acc > precision:
        x0 = (x1 + x2) / 2.0
        guess = f(x0)
        if sign(guess) == sign(f(x1)):
            x1 = x0
        else:
            x2 = x0
        acc = abs(x1 - x2)
        steps.append(guess)
    return (x0, steps)

def newtonRaphson(f, fp, x1):
    guess = f(x1)
    steps = [guess]
    x2 = x1
    while abs(guess) > precision:
        x2 = x1 - guess / fp(x1)
        guess = f(x2)
        x1 = x2
        steps.append(guess)
    return (x2, steps)

def secant(f, x1, x2):
    guess = f(x2)
    prev = f(x1)
    steps = [guess]
    while abs(guess) > precision:
        next = x2 - guess * (x2 - x1) / (guess - prev)
        prev = guess
        guess = f(next)
        steps.append(guess)
        x1 = x2
        x2 = next
    return (x2, steps)
```

Using these methods, we plot the convergence rates of Sin(x) - .76:

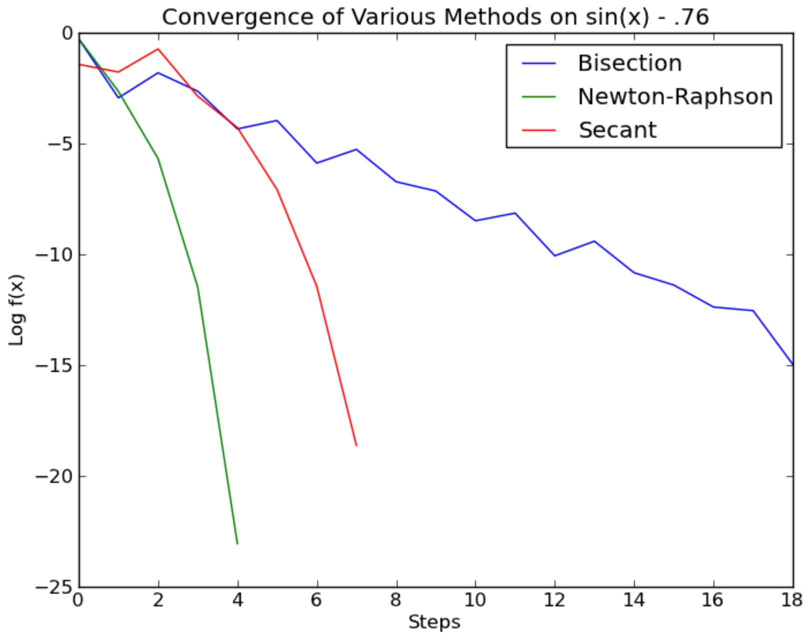



Figure 1: Convergence Rates of the Above Methods

We see that the bisection method is linear, Newton is quadratic, and Secant lies in the middle at the Golden Ratio.

# 3 Orbit

Using the Newton-Raphson Method (since the derivative is easy to calculate), we get the following orbital curve:

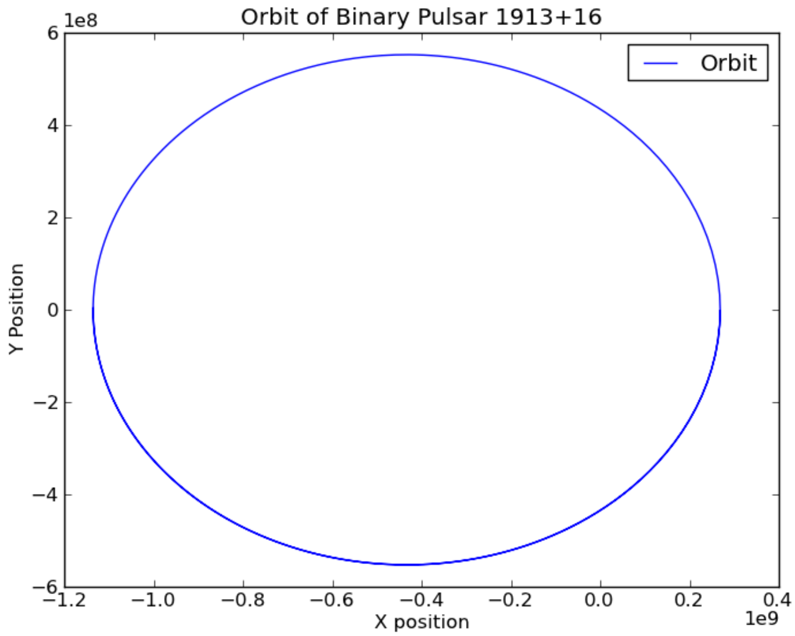



Figure 2: Orbit of the Hulse-Taylor Pulsar

# 4 Velocity Curve

By trial and error of varying the phase, we get the following velocity curve using a $\Delta$t of .001:

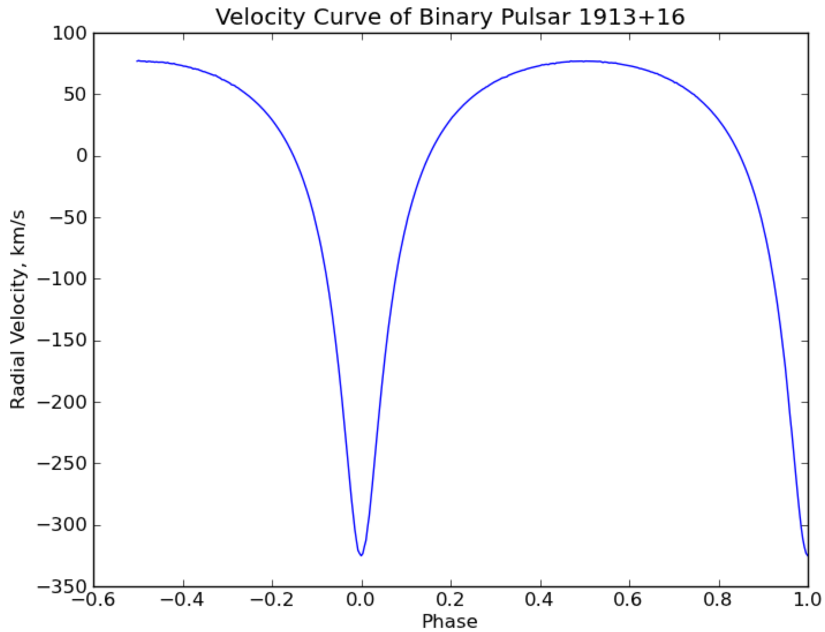



Figure 3: Velocity Curve of the Hulse-Taylor Pulsar

The above graph agrees with the 1975 diagram provided using a $\phi = -\frac{\pi}{2}$.

# 5 Orbit in Mathematica

Using Mathematica's FindRoot function, we get the following plot for the orbit:

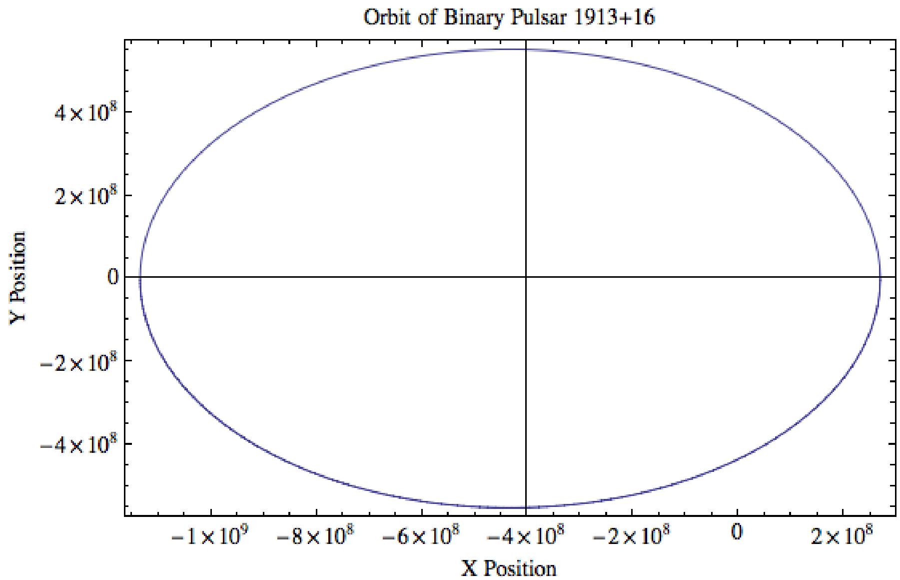



Figure 4: Orbit of the Hulse-Taylor Pulsar Using Mathematica

which agrees perfectly with the orbit found in Python using my Newton-Raphson function.

# 6 Code and Info

## 6.1 Output

The program outputs the following:

```
Running Methods on default function: f(x) = sin(x) - .76
Bisection Guess: 0.863314
Newton-Rpahson Guess: 0.863313
Secant Guess: 0.863313
Found a Qualitative Velocity Match at theta = -1.570796
Successfully Ran Script
```

## 6.2 Code

Code for this week's set in Python and Appended Mathematica Code as a PDF:

```python
#!/usr/bin/env python

# Chris Dudiak
# This program contains various methods to solve for the roots of functions.
```

```
import math
import numpy as np
import operator as op
import sys
import matplotlib.pyplot as plt

# global vars
precision = .00001

def roots(show):
    (zb, sb) = bisection(func, -math.pi / 2.0, math.pi / 2.0)
    (zn, sn) = newtonRaphson(func, funcP, 0)
    (zs, ss) = secant(func, -math.pi / 2.0, math.pi / 2.0)
    print "Bisection Guess: %f" % zb
    print "Newton-Rpahson Guess: %f" % zn
    print "Secant Guess: %f" % zs

    xB = range(len(sb))
    xN = range(len(sn))
    xS = range(len(ss))

    yB = map(lambda y: math.log(abs(y)), sb)
    yN = map(lambda y: math.log(abs(y)), sn)
    yS = map(lambda y: math.log(abs(y)), ss)

    # plot the convergence on a log-log plot
    fig = plt.figure()
    plt.plot(xB, yB, label="Bisection")
    plt.plot(xN, yN, label="Newton-Raphson")
    plt.plot(xS, yS, label="Secant")
    title = "Convergence of Various Methods on sin(x) - .76"
    plt.title(title)
    plt.xlabel("Steps")
    plt.ylabel("Log f(x)")
    plt.legend()
    if show:
        plt.show()
    fig.savefig("convergence.png")

    # Question 3 - Orbit
    e = 0.617139
    T = 27906.98161
    c = 299792458
    a = 2.34186 * c

    orbP = lambda x : T / (2 * math.pi) * (1 - e * np.cos(x))
```

```
xs = []
ys = []
deltaT = .001
xsDelta = []
ysDelta = []
ts = range(-14000, 28000, 100)
for t in ts:
    orb = lambda x : T / (2 * math.pi) * (x - e * np.sin(x)) - t
    orb2 = lambda x : T / (2 * math.pi) * (x - e * np.sin(x)) - (t + deltaT)
    (zero, _) = newtonRaphson(orb, orbP, t)
    (zero2, _) = newtonRaphson(orb2, orbP, t + deltaT)
    x = a * (np.cos(zero) - e)
    y = a * math.sqrt(1 - e**2) * np.sin(zero)
    xs.append(x)
    ys.append(y)

    xD = a * (np.cos(zero2) - e)
    yD = a * math.sqrt(1 - e**2) * np.sin(zero2)
    xsDelta.append(xD)
    ysDelta.append(yD)

# plot the orbit
fig = plt.figure()
plt.plot(xs, ys, label="Orbit")
title = "Orbit of Binary Pulsar 1913+16"
plt.title(title)
plt.xlabel("X position")
plt.ylabel("Y Position")
plt.legend()
if show:
    plt.show()
fig.savefig("orbit.png")

# Part 4 - Velocity Curve
th = -math.pi / 2.0
rads = []
for i in range(len(xs)):
    xp = (xsDelta[i] - xs[i]) / deltaT
    yp = (ysDelta[i] - ys[i]) / deltaT
    radV = np.dot([xp, yp], [np.cos(th), np.sin(th)]) / 1000.0
    rads.append(radV)

tTs = map(lambda t : float(t) / T, ts)
# plot the velocity curve
fig = plt.figure()
plt.plot(tTs, rads)
```

```
    title = "Velocity Curve of Binary Pulsar 1913+16"
    plt.title(title)
    plt.xlabel("Phase")
    plt.ylabel("Radial Velocity, km/s")
    if show:
        plt.show()
    fig.savefig("velocity.png")

    print "Found a Qualitative Velocity Match at theta = %f" % th
    print "Successfully Ran Script"

def func(x):
    return np.sin(x) - .76

def funcP(x):
    return np.cos(x)

# Takes a function and bounds x1,x2 to locate the zero by bisection.
# returns the tuple (x0, [steps taken])
def bisection(f, x1, x2):
    acc = abs(x1 - x2)
    steps = []
    x0 = (x1 + x2) / 2.0
    while acc > precision:
        x0 = (x1 + x2) / 2.0
        guess = f(x0)
        if sign(guess) == sign(f(x1)):
            x1 = x0
        else:
            x2 = x0
        acc = abs(x1 - x2)
        steps.append(guess)
    return (x0, steps)

# Takes a function, its derivative, and initial guess x1 to locate
# the zero by Newton-Raphson.
# returns the tuple (x0, [steps taken])
def newtonRaphson(f, fp, x1):
    guess = f(x1)
    steps = [guess]
    x2 = x1
    while abs(guess) > precision:
        x2 = x1 - guess / fp(x1)
        guess = f(x2)
        x1 = x2
        steps.append(guess)
```

```python
    return (x2, steps)

# Takes a function, two initial guesses x1,x2 to locate
# the zero by Secant method.
# returns the tuple (x0, [steps taken])
def secant(f, x1, x2):
    guess = f(x2)
    prev = f(x1)
    steps = [guess]
    while abs(guess) > precision:
        next = x2 - guess * (x2 - x1) / (guess - prev)
        prev = guess
        guess = f(next)
        steps.append(guess)
        x1 = x2
        x2 = next
    return (x2, steps)

# determines the sign of x
def sign(x):
    if x > 0:
        return 1
    elif x < 0:
        return -1
    else:
        return 0

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print "usage: %s bool_to_show_plot(0 or 1) " % sys.argv[0]
        sys.exit(1)
    print "Running Methods on default function: f(x) = sin(x) - .76"
    roots(int(sys.argv[1]))
```

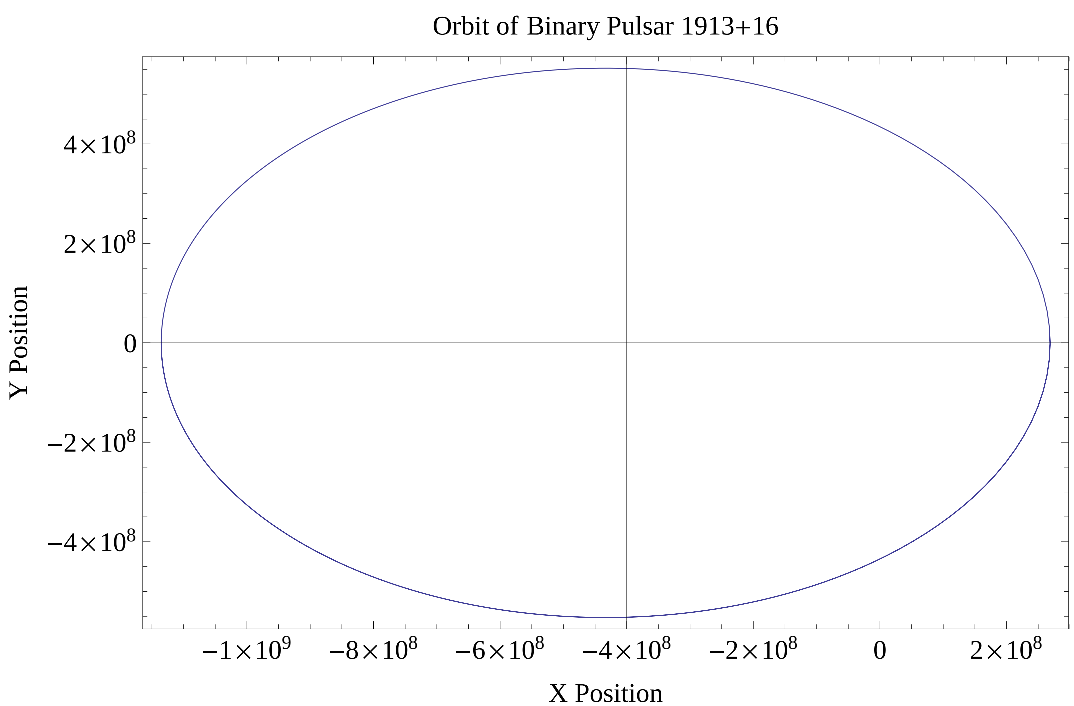
# Lab 6 - Roots

```
In[59]:= Needs["PlotLegends`"]
dir = NotebookDirectory[];
SetDirectory[dir];
```

Calculate the orbit for the pulsar

```
In[62]:= e = 0.617139;
T = 27906.98161;
c = 299792458;
a = 2.34186 * c;

In[66]:= ts = Table[t, {t, -14000, 28000, 100}];

In[67]:= eqs = Map[Function[t, T / (2 * Pi) * (x - e * Sin[x]) - t], ts];

In[68]:= zeros = Map[Function[f, FindRoot[f, {x, 0}]], eqs][[All, 1, 2]];

In[69]:= rs = Map[Function[z, a * (1 - e * Cos[z])], zeros];

In[70]:= xs =  Map[Function[z, a * (Cos[z] - e)], zeros];

In[71]:= ys =  Map[Function[z, a * (Sqrt[1 - e^2] * Sin[z])], zeros];

In[72]:= data = Transpose[{xs, ys}];

In[73]:= graph = ListPlot[data, Joined → True, AxesOrigin → {-4 * 10^8, 0}, Frame → True,
  FrameLabel → {{"Y Position", ""}, {"X Position", "Orbit of Binary Pulsar 1913+16"}},
  ImageSize → Large, LabelStyle → Larger ]
```

Out[73]=




```
In[74]:= Export["orbitMath.png", graph]

Out[74]= orbitMath.png
```