# Numerical Methods: Assignment #2

Hong Chenhui
drredthered.github.io

Zhejiang University — March 17, 2024

**Info:** Chinese version would (...may?) appear on my private blog afterwards, which may include not-so-academic comments on this matter. Click here for more details.

## 1 Problem

- **Question** Aerospace engineers sometimes compute the trajectories of projectiles like rockets. A related problem deals with the trajectory of a thrown ball. The trajectory of a ball is defined by the (x, y) coordinates, as displayed in Fig. P8.37. The trajectory can be modeled as.

$$y = x\tan(\theta) - \frac{1}{2}\frac{g}{v_0^2\cos^2(\theta)}x^2 + y_0 \tag{1}$$

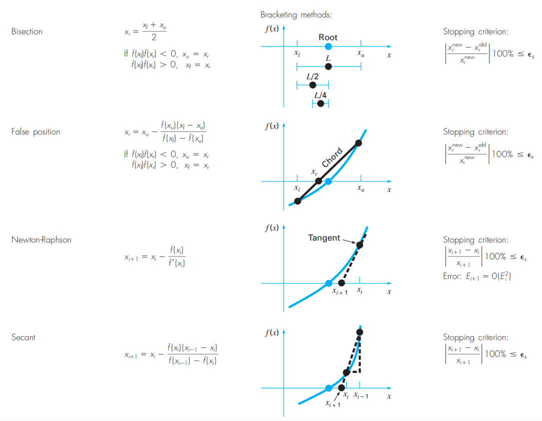
  Find the appropriate initial angle $\theta_0$, if the initial velocity is 30 m/s and the catcher's coordinates (x,y) is (90,1) m. Note that the ball leaves the thrower's hand at an elevation of $y_0 = 1.8$ m. Use a value of 9.81 $m/s^2$ for g and employ following methods to develop your initial guesses.

  – Bisection method
  – False position method
  – Fixed-point iteration
  – Newton-Raphson method
  – Secant method

### 1.1 Theoretical viewpoint

**Question**

Methods details except Fix-point iteration are omitted here, as they are not the focus of this assignment.



Question

It is necessarily to note that although the false-position method would seem to always be the bracketing method of preference, there are cases where it performs poorly, which illustrated its one-sidedness nature. Thus for this assignment, we would use the Modified False Position method as the bracketing method of preference. It would significantly improving the convergence rate as well as the stability of the method. Pseudocodes as follows:

1. begin iterating as normal false position method would do.
2. if one-sided for more than 2 times, divide the $f(x)$ value of the other side by 2.
3. end if the desired accuracy or the iteration limit is reached.

## 2 Implementation

$MATLABR2023a$ codes seen at the codes section as well as attached files.

1. With open method $\theta_0 = 0.85$ and any two-initials iteration $\theta_l = 0.8, \theta_u = 1.0$:

| | $\theta_0$ | iter | $\epsilon_a$ |
|---|---|---|---|
| Bisec | 0.899398457607283 | 48 | 3.703218574263763e-14 |
| FalPos | 0.899398457607284 | 12 | 0 |
| Fixpt | -1.566889411955219e+06 | 100 | 1.174552271115739e-04 |
| New-Raph | 0.899398457607283 | 7 | 0 |
| Secant | 0.899398457607284 | 11 | 1.234406191421253e-14 |

Significant poor result generated by Fixpt. Illustrated as below.

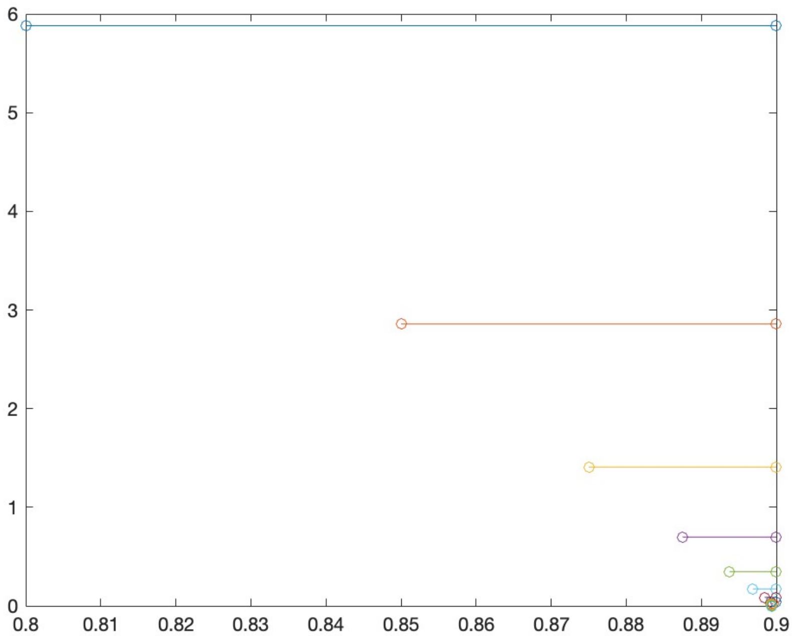



Fig 1. The convergence of Bisection method.

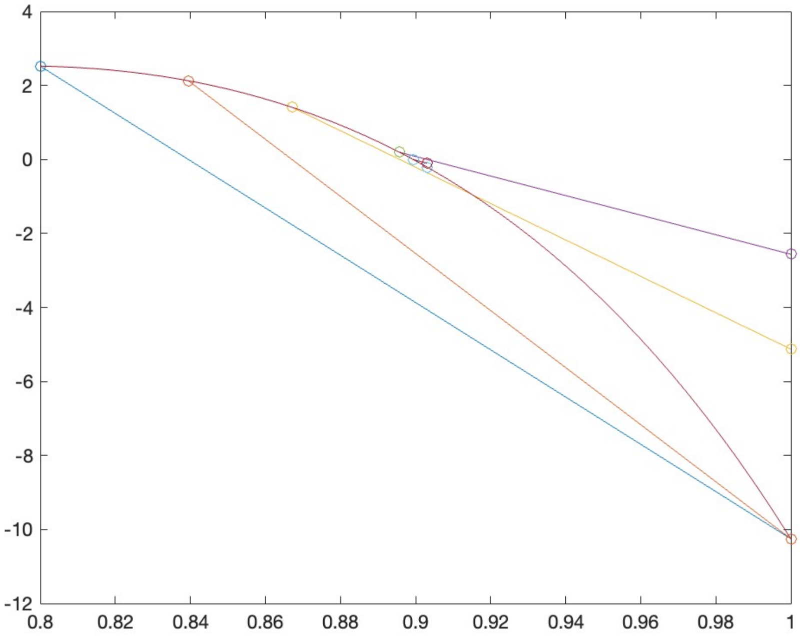



Fig 2. The convergence of False Position method.

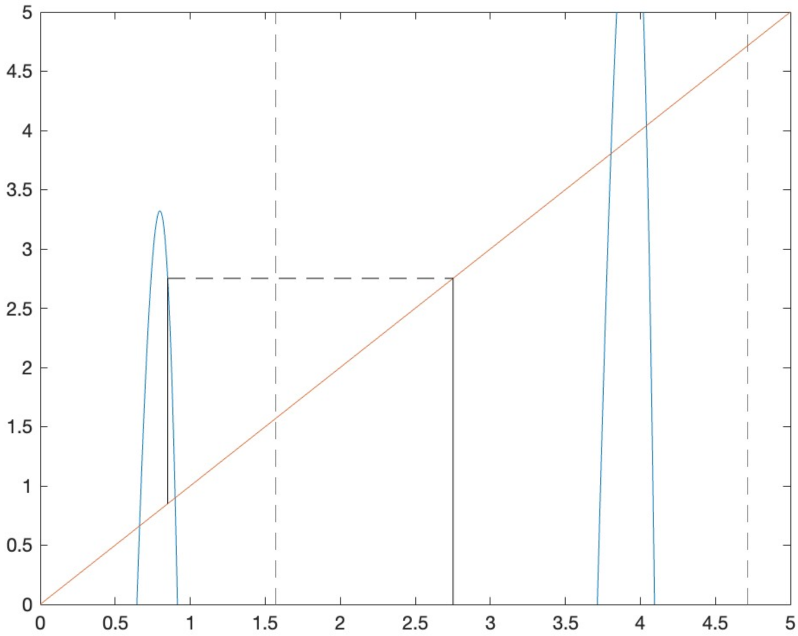



Fig 3. Fix-point iteration method never converge.

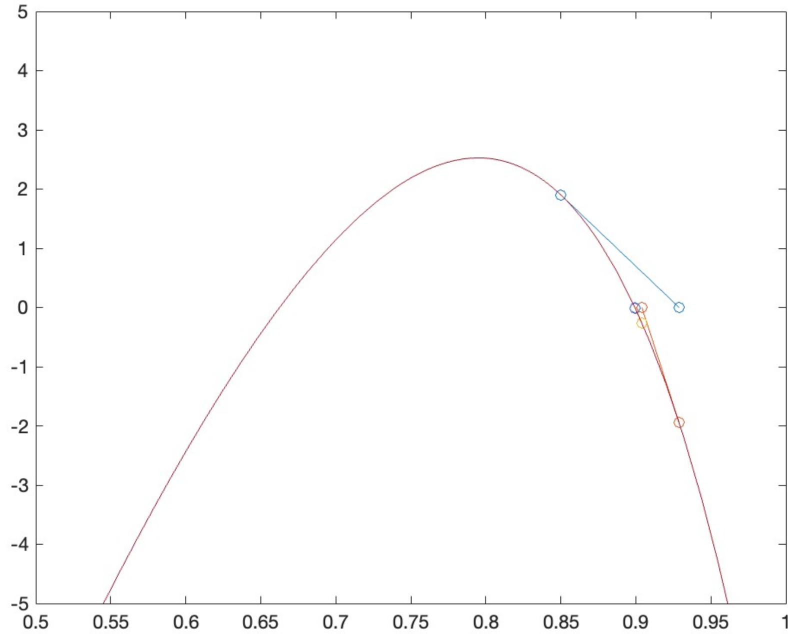



Fig 4. The convergence of Newton-Raphson method.

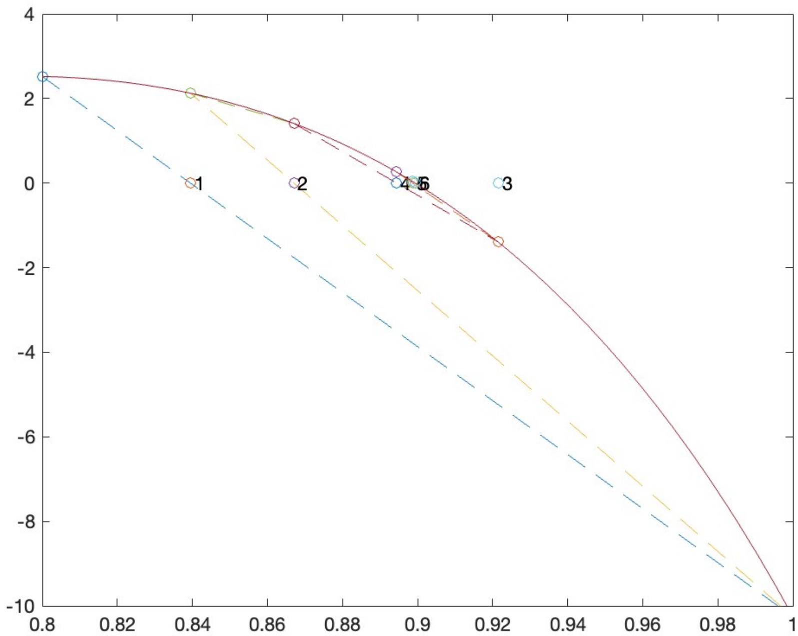



Fig 5. The convergence of Secant method.

2. With open method $\theta_0 = 0.60$ and any two-initials iteration $\theta_l = 0.6, \theta_u = 0.8$:

| | $\theta_0$ | iter | $\epsilon_a$ |
|---|---|---|---|
| Bisec | 0.662509214398280 | 49 | 2.513677547036482e-14 |
| FalPos | 0.662509214398280 | 9 | 3.351570062715310e-14 |
| Fixpt | -4.367188178140287e+05 | 100 | 0.004328076144322 |
| New-Raph | 0.662509214398281 | 7 | 1.675785031357653e-14 |
| Secant | 0.662509214398280 | 9 | 1.675785031357654e-14 |

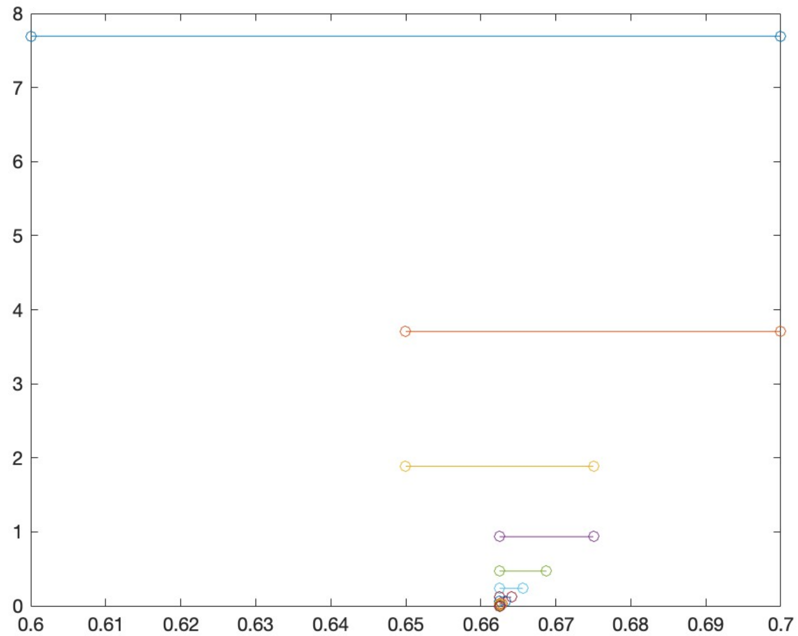



Fig 6. The convergence of Bisection method.

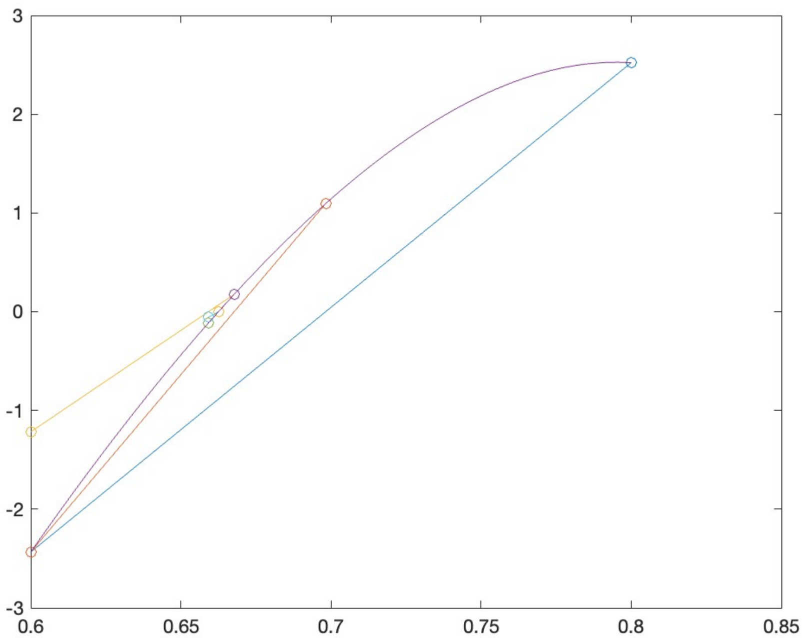



Fig 7. The convergence of False Position method.

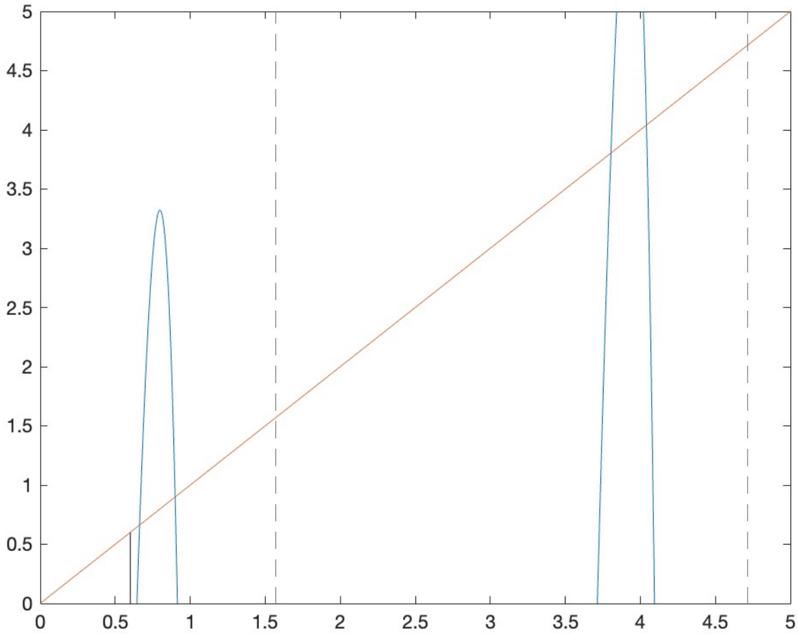



Fig 8. Fix-point iteration flying everywhere.

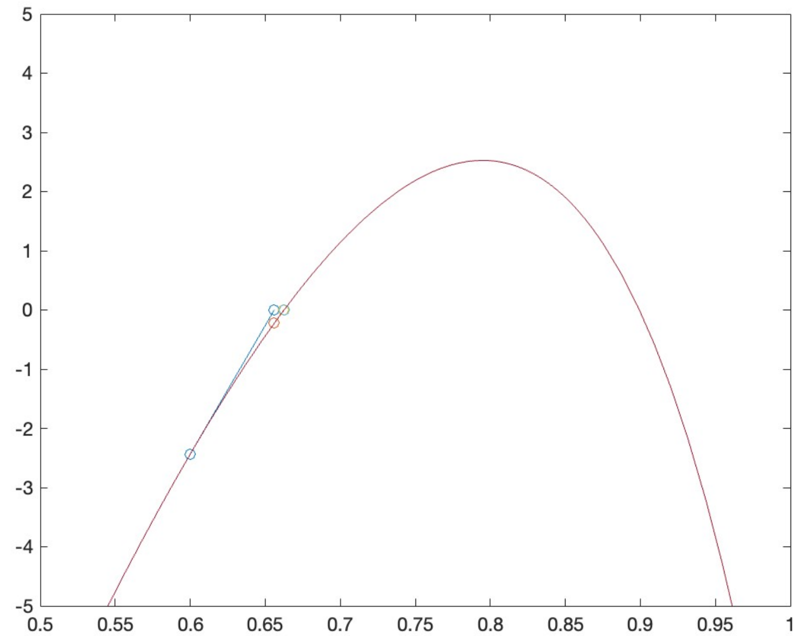



Fig 9. The convergence of Newton-Raphson method.

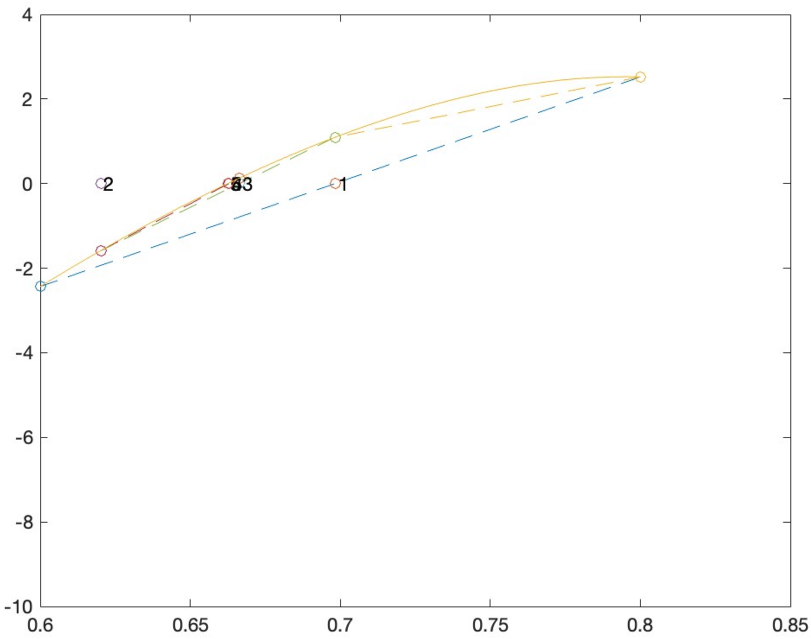



Fig 10. The convergence of Secant method.

# 3 Analysis

## 3.1 Worst day for Fix-point iteration

The reason behind the poor performance of Fix-point iteration is the fact that the slope of the function is greater than 1, which is a direct consequence of the fact that the function is not a contraction mapping. Which means, Fix-point iteration, due to its strict regulations, could be the least direct one, and the most sensitive one to the initial guess. This is the reason why the Fix-point iteration method is not recommended for this problem.

Yet, it does not necessarily mean that the method would obtain slower convergence rate than the other. For example, if we modified the original equation into:

$$\begin{aligned} y =& x\tan(\theta) - \frac{1}{2}\frac{g}{v_0^2\cos^2(\theta)}x^2 + y_0 \\ &= x\tan(\theta) - \frac{1}{2}\frac{g}{v_0^2}x^2(1+\tan^2(\theta)) + y_0 \\ &= -\frac{1}{2}\frac{g}{v_0^2}t^2 + xt + y_0 - \frac{1}{2}\frac{g}{v_0^2} \leftarrow (quadratic) \end{aligned} \tag{2}$$

Then using following iterations:

$$g_1(t) = t + \frac{1}{20}\left(-\frac{1}{2}\frac{g}{v_0^2}t^2 + xt + y_0 - \frac{1}{2}\frac{g}{v_0^2} - y\right) \tag{3}$$

$$g_2(t) = t + \frac{(-1)}{20}\left(-\frac{1}{2}\frac{g}{v_0^2}t^2 + xt + y_0 - \frac{1}{2}\frac{g}{v_0^2} - y\right) \tag{4}$$

Would one obtain

| | $\tan(\theta_0)$ | iter | $\epsilon_a$ | $\theta_0$ |
|---|---|---|---|---|
| Upper | 1.258602605626221 | 13 | 1.764215360213341e-14 | 0.899398457607284 |
| Lower | 0.780133378063891 | 13 | 1.423119502181116e-14 | 0.662509214398280 |

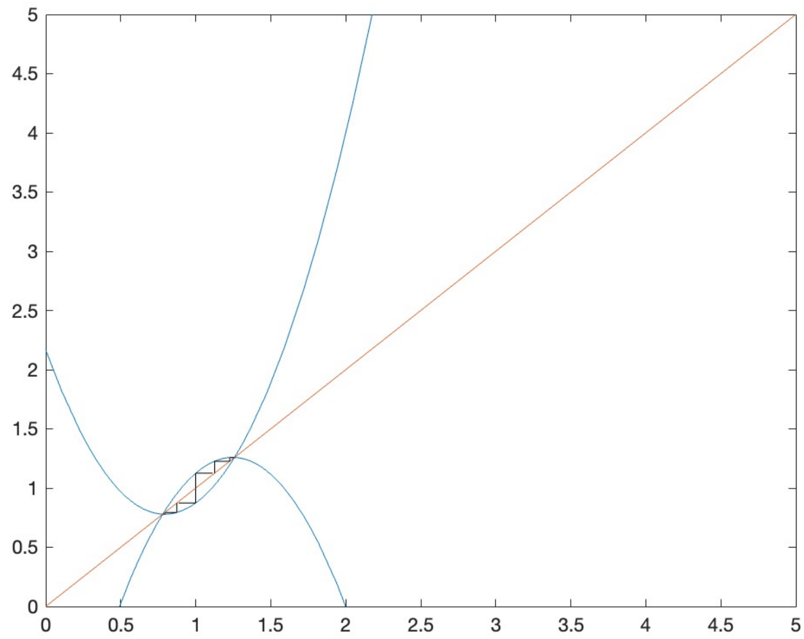



Fig. 11 The convergence of Fix-point iteration method with modified equation

Which is even better than the Bisection method in convergence rate————the latter needs almost 50 times to reach a satisfying result.

## 3.2 Comparison of the methods

As shown in the following graph.

| Method | Type | Guesses | Convergence | Stability | Programming | Comments |
|---|---|---|---|---|---|---|
| Direct | Analytical | — | — | — | | |
| Graphical | Visual | — | — | — | — | Imprecise |
| Bisection | Bracketing | 2 | Slow | Always | Easy | |
| False-position | Bracketing | 2 | Slow/medium | Always | Easy | |
| Modified FP | Bracketing | 2 | Medium | Always | Easy | |
| Fixed-point iteration | Open | 1 | Slow | Possibly divergent | Easy | |
| Newton-Raphson | Open | 1 | Fast | Possibly divergent | Easy | Requires evaluation of $f'(x)$ |
| Modified Newton-Raphson | Open | 1 | Fast (multiple), medium (single) | Possibly divergent | Easy | Requires evaluation of $f'(x)$ and $f''(x)$ |
| Secant | Open | 2 | Medium/fast | Possibly divergent | Easy | Initial guesses do not have to bracket the root |

## 3.3 Results in Degree

Thus we conclude:

$$\begin{aligned}\theta_0 &= 51.53173572^\circ \\ \theta_1 &= 37.95898187^\circ\end{aligned} \tag{5}$$

# 4 Codes

Matlab Codes are attached below and in the file. Also at my private blog: drredthered.github.io.

//script

RootFinding.m

```
%clearing script
clear;
clc;
%initialization
y = 1.0;
x = 90;
v0 = 30;
y0 = 1.8;
g = 9.81;
%root finding
f = @(theta) tan(theta).*x -
g .* x.^2./(2.*v0.^2.*cos(theta).^2) + y0 - y;

g = @(theta) theta + tan(theta).*x -
g .* x.^2./(2.*v0.^2.*cos(theta).^2) + y0 - y;

%for any two-point init. methods.
xl = 0.6;
xu = 0.8;
es = 0.5e-13;
imax = 100;

%for any open methods.
x0 = 0.6;

%bisection method
figure(1);
[theta0(1),iter(1),ea(1)] =
Bisection(xl,xu,es,imax,f);

%false position method modified
figure(2);
[theta0(2),iter(2),ea(2)] =
ModRegulaFalsi(xl,xu,es,imax,f);

%fixed-point method
figure(3);
[theta0(3),iter(3),ea(3)] = Fixpt(x0,es,imax,g);

%newton_raphson method
figure(4);
[theta0(4),iter(4),ea(4)] =
NewtonRaphson(x0,es,imax,f);

%secant method
figure(5);
[theta0(5),iter(5),ea(5)] =
SecantMethod(xl,xu,es,imax,f);

disp(theta0);
disp(iter);
disp(ea);
```

Bisection.m

```
function [x0,iter,ea] = Bisection(xl,xu,es,imax,f)

iter = 0;
fl = f(xl);
xr = xl;

while(1)
        xr = (xl+xu) / 2;
        fr = f(xr);
        iter = iter + 1;

        flag = fl*fr;
        if flag < 0
                xu = xr;
        elseif flag > 0
                xl = xr;
                fl = fr;
        else
                ea = 0;
        end

        if xr ~= 0
                ea = abs((xu-xl)/(xu+xl))*100;
        end

        X1(iter) = xl;
        X2(iter) = xu;
        Y(iter) = ea;

        if ea < es || iter >= imax, break,end
end

x0 = xr;

for i = 1:iter
        plot([X1(i),X2(i)],[Y(i),Y(i)],"-o");
hold on;
        end

end
```

```
ModRegulaFalsi.m

function [x0,iter,ea] = ModRegulaFalsi(xl,xu,es,imax,f)
    iter = 0;
    fl = f(xl);
    fu = f(xu);
    xr = xl;
    iu = 0;
    il = 0;

    plot([xl,xu],[fl,fu],"-o");
    hold on;

    while(1)
        xrold = xr;
        xr = xu - fu*(xl - xu)/(fl - fu);
        fr = f(xr);

        iter = iter + 1;
        if xr ~= 0
            ea = abs((xr - xrold)/ xr) * 100;
        end

        flag  = fl * fr;
        if flag < 0
            xu = xr;
            fu = f(xu);
            iu = 0;
            il = il +1;
            if il >= 2
                fl = fl / 2;
            end
        elseif flag > 0
            xl = xr;
            fl = f(xl);
            il = 0;
            iu = iu + 1;
            if iu >= 2
                fu = fu / 2;
            end
        else
            ea = 0;
        end

        plot([xl,xu],[fl,fu],"-o");

        if ea < es || iter >= imax, break, end
    end
    fplot(f,[0.6,0.8]);
    x0 = xr;
end
```

Fixpt.m

```
function [xr,iter,ea] = Fixpt(x0,es,imax,g)

%initialization
xrold = x0;
xr = g(xrold);

iter = 0;
ea = 0;

%plotting
fplot(g);
hold on
fplot(@(x)x);

%recurrence iteration
while(1)
        plot([xrold xrold], [xrold xr], 'k-')
        xlim([0,5])
        ylim([0,5])
        plot([xrold xr], [xr xr], 'k--')
        xlim([0,5])
        ylim([0,5])

        xrold = xr;
        xr = g(xrold);

        iter = iter + 1;
        if xr~= 0
                ea = abs((xrold - xr)/xr)*100;
        end
        if ( iter >= imax || ea < es ),break,end
end
%result
end
```

Fixpt.m

```
function [root, iterations, ea] =
NewtonRaphson( initial_guess, tolerance,
max_iteration, func )
    syms x;

    f = matlabFunction(sym(func));
    df = matlabFunction(diff(sym(func)));

    root = initial_guess;
    iterations = 0;
    ea = 100;

    while true
        iterations = iterations + 1;

        % Newton-Raphson iteration formula
        root_new = root - f(root) / df(root);

        if ea ~= 0
            ea = abs((root_new-root)/root_new)*100;
        end

        % Check for convergence
        if ea<tolerance || iterations>=max_iteration
            root = root_new;
            break;
        end

        plot([root,root_new],[f(root),0],"-o");
        hold on

        root = root_new;
    end

    fplot(f,[0.5,1]);
    ylim([-5,5]);

end
```

Fixpt.m

```
function [root, iterations, ea] =
SecantMethod(x0, x1, tolerance, max_iterations, func)
    syms x;

    f = matlabFunction(sym(func));

    % Initial values
    x_prev = x0;
    x_curr = x1;
    iterations = 0;
    ea = 100;

    while true
        iterations = iterations + 1;

        % Secant method formula
        x_next = x_curr - f(x_curr)
        * (x_curr - x_prev) / (f(x_curr) - f(x_prev));

        if ea ~= 0
            ea = abs((x_next - x_curr)/x_next)*100;
        end

        % Check for convergence
        if ea<tolerance || iterations>=max_iterations
            root = x_next;
            break;
        end

        plot([x_prev,x_curr],
                [f(x_prev),f(x_curr)],"--o");
        hold on
        plot(x_next,0,"-o");
        if iterations <= 6
            text(x_next+0.001,0.001,
                        num2str(iterations));
        end

        x_prev = x_curr;
        x_curr = x_next;
    end

    fplot(f,[0.6,0.8]);
    ylim([-10,4]);

end
```