```
generateFunction (generic function with 3 methods)
```

```
Enter cell code...
```

```
iterativeSolve (generic function with 1 method)
```

```julia
using LinearAlgebra ✓
```

```julia
using ForwardDiff ✓
```

```julia
using PlutoUI ✓
```

```
normalizeDegree (generic function with 1 method)
```

```julia
normalizeDegree(X) = mod(X, 2*pi).*(180/pi)
```

```
normalizeRad (generic function with 1 method)
```

```julia
normalizeRad(X) = mod(X, 2*pi)
```

How to use package

1. Define Vectors: `@vectors V1 V2 V3 V4`
2. Define Loop equation: `vLoop = V1 + V2 - V3 - V4`
3. Create Vector Loop: `VectorLoop(Input, Output1, Output2, Constraints)`
4. Store Known Values in a Dictionary: `Dict("θ1" => 2π)`
5. Solve

```
Enter cell code...
```

```
▸briVec(:(r26 * cos(θ26)), :(r26 * sin(θ26)))
```

```julia
@vectors V1 V2 V3 V4 Vp1 V5 V15 V6 V26
```

```
Enter cell code...
```

```
loopEq =
▸briVec(:(((r2 * cos(θ2) + r3 * cos(θ3)) + r4 * cos(θ4)) - r1 * cos(θ1)), :(((r2 * sin(θ2)
```

```julia
loopEq = V2 + V3 + V4 - V1
```

```
loopEq2 =
▸briVec(:(((((r2 * cos(θ2) + r26 * cos(θ26)) + r6 * cos(θ6)) + r5 * cos(θ5)) + r15 * cos(θ
```

```julia
loopEq2 = V2 + V26 + V6 + V5 + V15 - V1
```

```
v16 (generic function with 1 method)
```

```julia
function v16(θ2, θ3, knowns)
    r2 = knowns["r2"]
    r3 = knowns["r3"]
    x = r2 * cos(θ2) + (r3 / 2) * cos(θ3)
    y = r2 * sin(θ2) + (r3 / 2) * sin(θ3)
    r16 = sqrt(x^2 + y^2)
    θ16 = atan(y, x)
    return (r16, θ16)
end
```

```
▸[Vec(806.639, 1.63359, 1), Vec(401.0, 6.28318, 1), Vec(501.25, 4.82287, 1), Vec(501.25, 5.
```

```julia
# Define Vector of Vecs, used to plot the second vector loop
begin
    r16, θ16 = v16(θ2s[n], θ3s[n], knowns)
    Vecs2 = [Vec(r16, θ16), Vec(knowns["r6"], θ6s[n]), Vec(knowns["r5"], θ5s[n]),
    Vec(knowns["r15"], θ4s[n]), Vec(knowns["r1"], knowns["θ1"], -1)]
end
```

```
Enter cell code...
```

```
vLoop = ▸VectorLoop(:θ2, :θ3, :θ4, [])
```

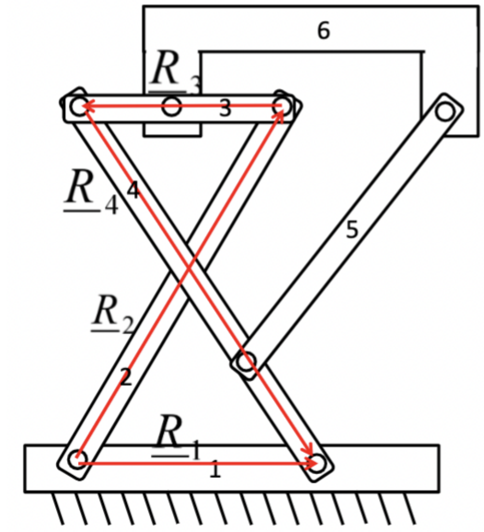
```julia
vLoop = VectorLoop(:θ2, :θ3, :θ4)
```

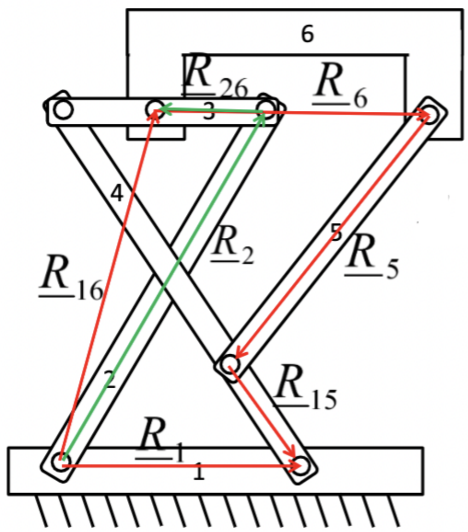
```
vLoop2 =
▸VectorLoop(:X, :θ5, :θ6, [:(θ15 = θ4), :(r26 = r3 / 2), :(θ26 = θ3), :(θ2 = X[1]), :(θ3 =
```

```julia
vLoop2 = VectorLoop(:X, :θ5, :θ6, [:(θ15 = θ4), :(r26 = r3/2), :(θ26 = θ3), :(θ2 =
X[1]), :(θ3 = X[2]), :(θ4 = X[3])])
```

```
Enter cell code...
```

```julia
# applyConstraint(loopEq2, vLoop2)
```





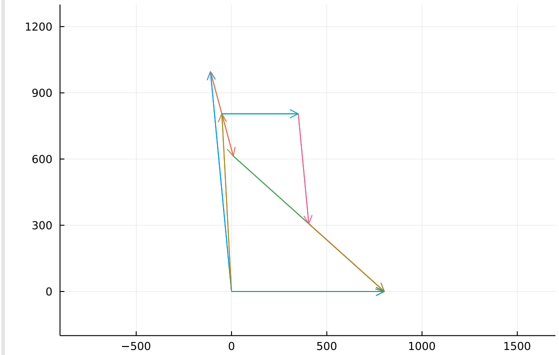

```
Enter cell code...
```

R5 = 501.25299806736604

```
R5= let
    θ15_min = (θ4s[end] - pi)
    r1 = knowns["r1"]
    r15 = knowns["r15"]
    p3x = r1 + r15*cos(θ15_min)
    p3y = r15*sin(θ15_min)

    p3min = (p3x, p3y)

    r6 = knowns["r6"]
    p1x = p1min[1]
    p1y = p1min[2]

    p2x = p1x + r6
    p2y = p1y

    y_R5 = p3y-p2y
    x_R5 = p3x - p2x
    mR5 = sqrt(y_R5^2 + x_R5^2)
    θr5 = normalizeRad(atan(y_R5, x_R5))*180/pi
    atan(y_R5, x_R5)*180/pi

    mR5
end
```

```
Enter cell code...
```

```
Enter cell code...
```




```julia
begin
    p = plotVecs(Vecs)
    plotVecs!(Vecs2, p)
    plot!(p, xlims=(-900, 1700), ylims=(-200,1300))
    # plotVecs(Vecs2)
end
```

```julia
Enter cell code...
```

```julia
@bind n Slider(length(inputs):-1:1)
```

p1min = ▸(-50.6157, 805.049)

```julia
p1min = (P1x[end], P1y[end])
```

p1max = ▸(797.676, 801.545)

```julia
p1max = (P1x[1], P1y[1])
```

▸(350.384, 805.047)

```julia
hP2(θ2s[n], θ3s[n], θ6s[n], knowns)
```

805.0470917217787

```julia
knowns["r2"]*sin(θ2s[n]) + knowns["r3"]/2*sin(θ3s[n]) + knowns["r6"]*sin(θ6s[n])
```

r15 = r1 * 0.625; r36 = r3 * 0.5; r5 = r1 * 0.625; r6 = r1 * 0.5;

```julia
md"""
r15 = r1 * 0.625;
r36 = r3 * 0.5;
r5 = r1 * 0.625;
r6 = r1 * 0.5;
"""
```

2.5

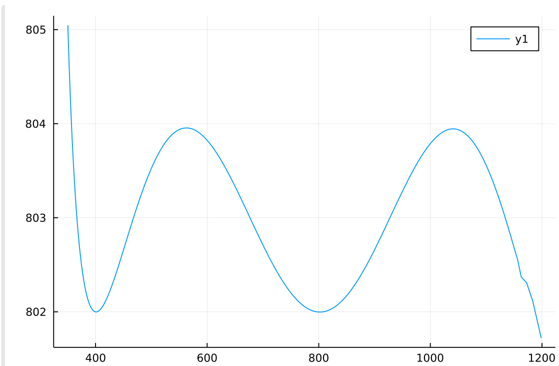
```julia
0.625*4
```

501.25299806736604

```julia
R5
```

knowns =
▸Dict("r1" ⇒ 802.0, "r2" ⇒ 1002.5, "θ1" ⇒ 0, "r15" ⇒ 501.25, "r5" ⇒ 501.25, "r6" ⇒ 40

```julia
knowns = Dict(
    "r1" => 4*a,
    "r2" => 5*a,
    "r3" => 2*a,
    "r4" => 5*a,
    "θ1" => 0,
    "r15" => 5*a/2,
    # "θ6" => 0,
    "r5" => 5*a/2,
    "r6" => 2*a,
)
```




```julia
plot(P2x, P2y)
```

▸[0.000441467, 6.09713e-5, 1.7115e-5, 0.000169897, 6.95915e-5, 5.61829e-5, 4.07554e-5, 3.11

```julia
θ6s
```

```julia
Enter cell code...
```

```julia
Enter cell code...
```

a = 200.5

```julia
a = 200.5
```

```
• Enter cell code...
```

```
• Enter cell code...
```

805.0

```
begin
    min_val = w
    max_val = min_val + 4
end
```

847.5988356585641

```
maximum(P1x[(P1y .> min_val) .& (P1y .< max_val)]) - minimum(P1x[(P1y .> min_val) .&
(P1y .< max_val)])
```

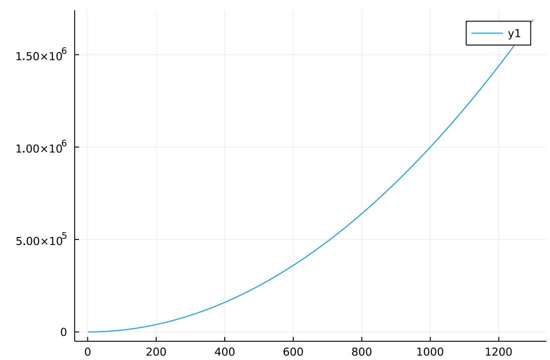
```
# θmin = θ2s[P1y[end:-1:1] .>min_val][1]
```

```
# θmax = θ2s[P1y[end:-1:1] .<max_val][end]
```

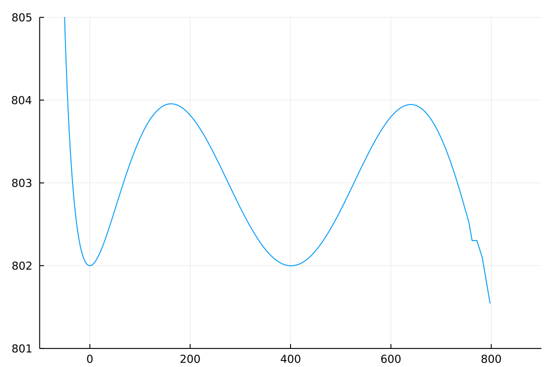
```
@bind w Slider(801:0.1:808)
```

1.683

```
begin
    θmin = 0.643
    θmax = 1.683
end
```




```
let
    xs = 1:1300
    ys = [x^2 for x in xs]
    plot(xs, ys)
end
```




```
begin
    p2 = plot(P1x, P1y, legend=false)
    # plot!(P2x, P2y)
    plot!(xlims=(-100,900), ylims=(min_val, max_val))
    # plot!(xlims=(-100,900), ylims=(790, 1500))

end
```

▸ [797.676, 782.251, 771.322, 761.912, 755.148, 748.705, 742.819, 737.319, 732.134, 727.209,

```
P1x
```

▸ [801.545, 802.095, 802.304, 802.303, 802.532, 802.665, 802.794, 802.907, 803.01, 803.104,

```
P1y
```

▸ [801.722, 802.119, 802.311, 802.371, 802.56, 802.688, 802.81, 802.92, 803.02, 803.112, 803

```
P2y
```

false

```
P1y ==P2y
```

foo (generic function with 1 method)

```
foo(x) = x+ 1
```

```
• Enter cell code...
```

bar (generic function with 1 method)

```
bar(x) = x^2
```

```
[[1.59387, 4.72134], [1.6657, 4.75106], [1.71532, 4.77183], [1.75758, 4.78964], [1.78662,
```

```
# Use newton method to solve for θ3 and θ4
begin
    y = []
    Xi = [π; 3.3π/4]
    for input in inputs
        h = (x) -> g(input, x)
        Xn = iterativeSolve(h, Xi, 0.05)
        Xi = normalizeRad.(Xn)
        push!(y, Xi)
    end
    y
end
```

```
[[3.78441, 0.000441467], [3.78633, 6.09713e-5], [3.78808, 1.7115e-5], [3.78975, 0.00016989
```

```
# Use newton method to solve for theta 5 and theta 6
begin
    y2 = []
    Xi_2 = [5π/4; 0]
    for input in inputs2
        h2 = (x) -> g2(input, x)
        Xn_2 = iterativeSolve(h2, Xi_2, 0.1)
        Xi_2 = normalizeRad.(Xn_2)
        push!(y2, Xi_2)
    end
    y2
end
```

```
Enter cell code...
```

```
θ5s =
[3.78441, 3.78633, 3.78808, 3.78975, 3.79153, 3.79327, 3.79502, 3.79677, 3.79852, 3.80027,
```

```
θ5s = [x[1] for x in y2]
```

```
θ6s =
[0.000441467, 6.09713e-5, 1.7115e-5, 0.000169897, 6.95915e-5, 5.61829e-5, 4.07554e-5, 3.11
```

```
θ6s = [x[2] for x in y2]
```

```
Enter cell code...
```

```
g (generic function with 1 method)
```

```
begin
    L2 = applyConstraint(loopEq, vLoop)
    L3 = applyConstraint(L2, knowns)
    exp = generateFunction(vLoop, L3)
    f = eval(exp)
    g(input, x) = f(input, x[1], x[2])
end
```

```
g2 (generic function with 1 method)
```

```
begin
    L2_2 = applyConstraint(loopEq2, vLoop2)
    L3_2 = applyConstraint(L2_2, knowns)
    exp2 = generateFunction(vLoop2, L3_2)
    f2 = eval(exp2)
    f2([1; 2; 3], 2, 3)
    inputs2 = collect(zip(θ2s, θ3s, θ4s))
    g2(input, x) = f2(input, x[1], x[2])
end
```

```
Enter cell code...
```

```
b = #3 (generic function with 1 method)
```

```
b = (x, y) -> begin x[1] + y[2] end
```

```
5
```

```
b([1;2], (3, 4))
```

```
Enter cell code...
```

```
Vec
```

```
plotVecs (generic function with 1 method)
```

```
function plotVecs(Vs::Vector{Vec})
    p = plot(arrow=true, legend=false)
    x, y = lineSegment(0, 0, Vs[1])
    if (Vs[1].sign > 0)
        point = [x[2], y[2]]
    else
        point = [x[1], y[1]]
    end
    plot!(p, x, y, arrow=true)
    for V in Vs[2:end]
        x, y = lineSegment(point[1], point[2], V)
        plot!(p, x, y, arrow=true)
        if (V.sign > 0)
            point = [x[2], y[2]]
        else
            point = [x[1], y[1]]
        end
    end
    return p
end
```

plotVecs! (generic function with 1 method)

```julia
function plotVecs!(Vs::Vector{Vec}, p::Plots.Plot)
    x, y = lineSegment(0, 0, Vs[1])
    if (Vs[1].sign > 0)
        point = [x[2], y[2]]
    else
        point = [x[1], y[1]]
    end
    plot!(p, x, y, arrow=true)
    for V in Vs[2:end]
        x, y = lineSegment(point[1], point[2], V)
        plot!(p, x, y, arrow=true)
        if (V.sign > 0)
            point = [x[2], y[2]]
        else
            point = [x[1], y[1]]
        end
    end
    return p
end
```

lineSegment (generic function with 1 method)

inputs =
[0.643, 0.644745, 0.64649, 0.648235, 0.64998, 0.651725, 0.65347, 0.655215, 0.65696, 0.6587

```julia
inputs = collect(θmin:0.001745:θmax)
```

θ3s =
[1.59387, 1.6657, 1.71532, 1.75758, 1.78662, 1.8142, 1.83906, 1.86208, 1.88359, 1.90387, 1

```julia
θ3s = [x[1] for x in y]
```

θ4s =
[4.72134, 4.75106, 4.77183, 4.78964, 4.8021, 4.81395, 4.82469, 4.83467, 4.84403, 4.85287,

```julia
θ4s = [x[2] for x in y]
```

```julia
Base.:-(V::Vec) = Vec(V.R, V.θ, V.sign*-1)
```

Vecs =
[Vec(1002.5, 1.68128, 1), Vec(401.0, 5.01585, 1), Vec(1002.5, 5.62438, 1), Vec(802.0, 0, -

```julia
Vecs = [Vec(knowns["r2"], θ2s[n]), Vec(knowns["r3"], θ3s[n]), Vec(knowns["r4"],
θ4s[n]), Vec(knowns["r1"], knowns["θ1"], -1)]
```

```julia
using Plots
```

θ2s =
[0.643, 0.644745, 0.64649, 0.648235, 0.64998, 0.651725, 0.65347, 0.655215, 0.65696, 0.6587

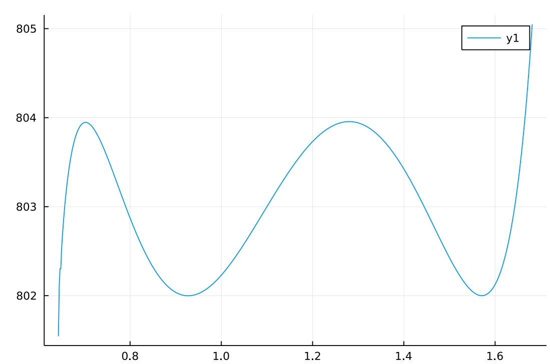
```julia
θ2s = inputs
```

596

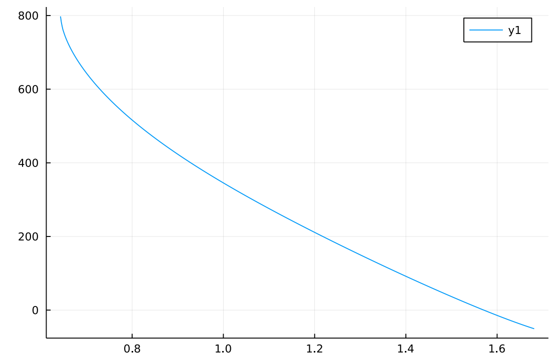
```julia
n
```

```julia
begin
    P1x = []
    P1y = []
    for i in 1:length(inputs)
        x_p, y_p = hP1(inputs[i], knowns, θ3s[i])
        push!(P1x, x_p)
        push!(P1y, y_p)
    end
    # P1x = P1x[end:-1:1]
    # P1y = P1y[end:-1:1]
end
```

```julia
begin
    P2x = []
    P2y = []
    for i in 1:length(inputs)
        x_p, y_p = hP2(inputs[i], θ3s[i], θ6s[i], knowns)
        push!(P2x, x_p)
        push!(P2y, y_p)
    end
    # P2x = P2x[end:-1:1]
    # P2y = P2y[end:-1:1]
end
```




```julia
plot(inputs, P1y)
```




```
plot(inputs, P1x)
```

848.2912464292209

```
maximum(P1x)-minimum(P1x)
```

```
Enter cell code...
```

Height of Point 1 vs angle

hP1 (generic function with 1 method)

```
function hP1(θ2, knowns, θ3)
    r2 = knowns["r2"]
    rp1 = knowns["r3"]/2
    y = r2*sin(θ2) + rp1*sin(θ3)
    x = r2*cos(θ2) + rp1*cos(θ3)
    return (x,y)
end
```

▸ Dict("r1" ⇒ 802.0, "r2" ⇒ 1002.5, "θ1" ⇒ 0, "r15" ⇒ 501.25, "r5" ⇒ 501.25, "r6" ⇒ 4(

```
knowns
```

hP2 (generic function with 1 method)

```
function hP2(θ2, θ3, θ6, knowns)
    r2 = knowns["r2"]
    rp1 = knowns["r3"]/2
    r6 = knowns["r6"]

    x = r2*cos(θ2) + rp1*cos(θ3) +r6*cos(θ6)
    y = r2*sin(θ2) + rp1*sin(θ3) +r6*sin(θ6)

    return (x, y)
end
```