In [ ]:

```
using Plots
using LinearAlgebra
using Interact
using Random

gr(size=(500,500));

Random.seed!(8765309);
```

In [ ]:

```
function quadratic_roots(alpha::Float64, beta::
    root1 = ((1.0 + 0.0im + beta - alpha * lamb
    root2 = ((1.0 + 0.0im + beta - alpha * lamb
    return (root1, root2);
end
```

In [ ]:

```
s_lambda = slider(0.1:0.1:5, label = "lambda",
s_alpha = slider(0.0:0.1:5, label = "alpha", va
s_beta = slider([0.5,0.9,0.99,0.999], label = "
theta = collect(0:0.01:1) .* (2 * pi);
plt = Interact.@map begin
    (root1, root2) = quadratic_roots(&s_alpha,
    scatter([real(root1), real(root2)], [imag(r
    plot!(sqrt(&s_beta) .* cos.(theta), sqrt(&s
end
wdg = Widget(["s_lambda" => s_lambda, "s_alpha"
@layout! wdg hbox(plt, vbox(:s_lambda, :s_alpha
```

Analysis of Momentum

In [ ]:

```
function gradient_descent(w0, A, alpha, niters)
    w = w0
    rv = zeros(niters)
    for i = 1:niters
        w = w - alpha * A * w
        rv[i] = norm(w)
```

```
    end
    return rv
end

function momentum_gd(w0, A, alpha, beta, niters
    w = w0
    wprev = w0
    rv = zeros(niters)
    for i = 1:niters
        (w, wprev) = (w - alpha * A * w + beta
        rv[i] = norm(w)
    end
    return rv
end
```

In [ ]:

```
# quadratic objective with many different eigen
A = diagm(0=>[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0
beta = ((sqrt(10) - 1)/(sqrt(10) + 1))^2;
alpha_mom = (2 + 2*beta) / (1 + 0.1);
alpha_gd = 2 / (1 + 0.1);
```

In [ ]:

```
w0 = randn(10);
```

In [ ]:

```
dists_gd = gradient_descent(w0, A, alpha_gd, 10
dists_gd_other_step_sizes = [gradient_descent(w
dists_mom = momentum_gd(w0, A, alpha_mom, beta,
```

In [ ]:

```
plot(dists_mom; label="momentum", yscale=:log10
plot!(dists_gd; label="optimal gd", yscale=:log
[plot!(dg; label="non-optimal gd", c="green", l
plot!()
```

In [ ]:

```
function nesterov_gd(w0, A, alpha, beta, niters
    w = w0
    v = w0
    rv = zeros(niters)
    for i = 1:niters
        wprev = w
```

```julia
            w = v - alpha * A * v
            v = w + beta * (w - wprev)
            rv[i] = norm(w)
        end
        return rv
end
```

In [ ]:
```julia
beta_nest = (sqrt(10) - 1)/(sqrt(10) + 1);
alpha_nest = 1;
dists_nest = momentum_gd(w0, A, alpha_nest, bet
```

In [ ]:
```julia
plot(dists_mom; label="polyak", yscale=:log10,
plot!(dists_gd; label="optimal gd", yscale=:log
plot!(dists_nest; label="nesterov", yscale=:log
```

In [ ]:
```

```

In [ ]:
```

```