jl-mason2014

October 21, 2024

[185]:
```
using ForwardDiff, Plots, NLsolve
```

Neglect electrostatic.
Use the parameter values from Mason 2014 paper.
Take $\phi = 0.6$.

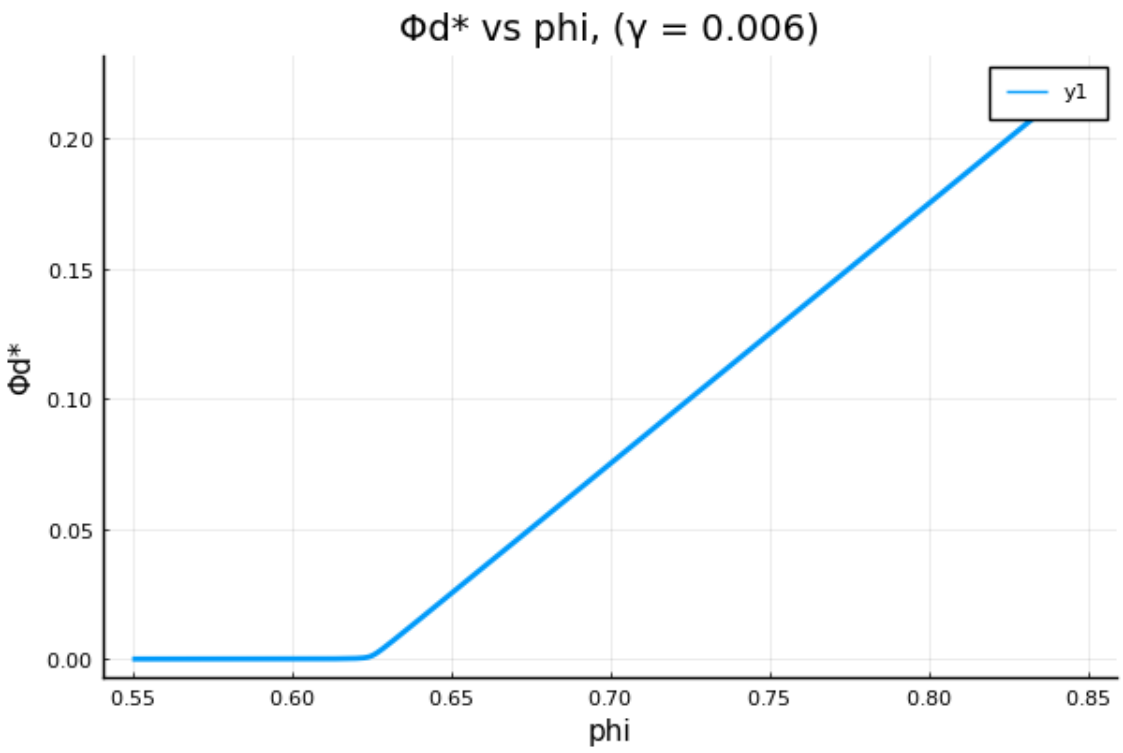
[186]:
```
alpha = 0.74  # Shear effect parameter
sigma = 9.8e-3  # Surface tension in J/m²
k_B = 1.38e-23  # Boltzmann constant in J/K
T = 298  # Temperature in K
a = 530e-9  # Droplet radius in meters
xi = 0.14  # Coupling parameter
phi_c = 0.625  # Critical volume fraction
```

```
0.625
```

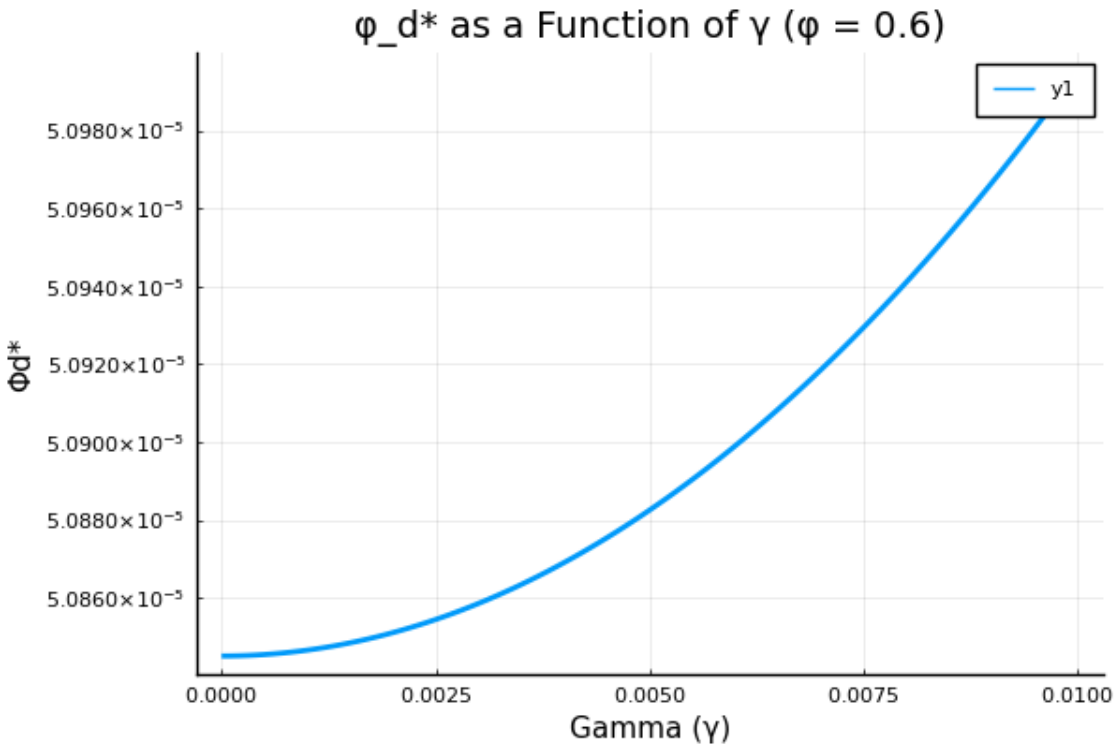
[187]:
```
phi_T_squared = (3 * k_B * T / a^3) / (2 * pi * xi * sigma / a)

function phid_star(phi, gamma)
    term1 = phi - (phi_c - alpha * gamma^2)
    return 0.5 * (term1 + sqrt(term1^2 + phi_T_squared))
end
```

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

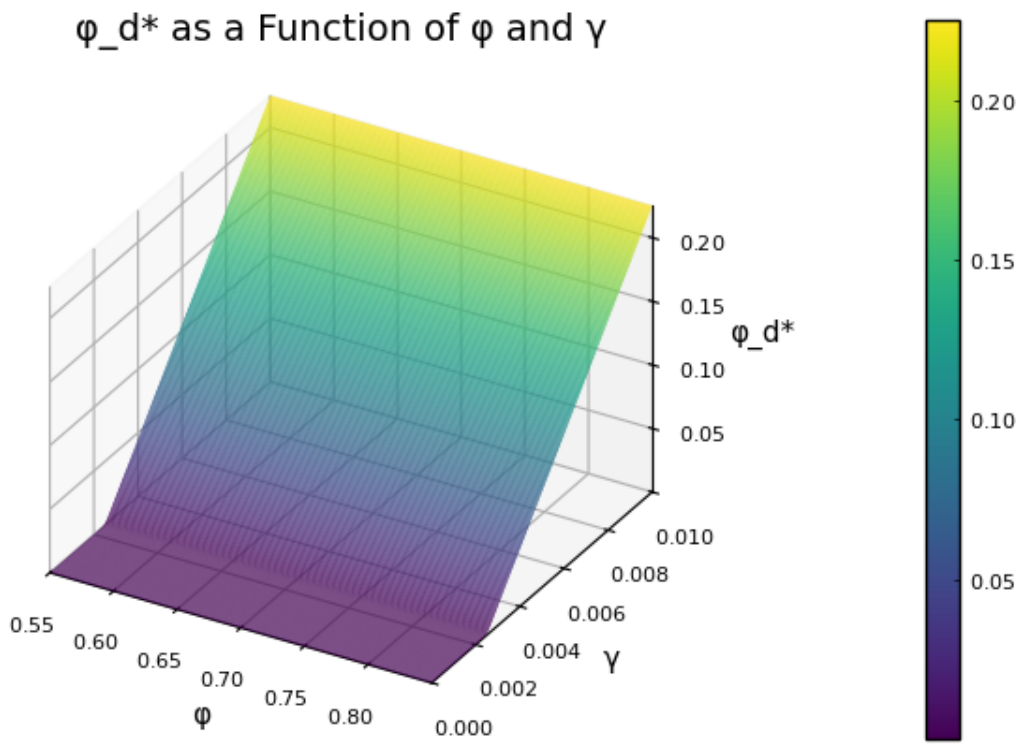
[188]:
```
phi_values = 0.55:0.0003:0.85
gamma_fixed = 0.006
phid_star_values = phid_star.(phi_values, gamma_fixed)
plot(phi_values, phid_star_values, xlabel="phi", ylabel="Φd*", title="Φd* vs␣
↪phi, ( = $gamma_fixed)", lw=2)
```



[189]:
```
gamma_values = 0:0.0001:0.01
phi_fixed =0.6
phid_star_values = phid_star.(phi_fixed, gamma_values)

plot(gamma_values, phid_star_values, xlabel="Gamma ( )", ylabel="Φd*",␣
 ↪title=" _d* as a Function of   ( = $phi_fixed)", lw=2)
```



[190]:
```
phi_values = range(0.55, 0.85, length=300)
gamma_values = range(0.0, 0.01, length=300)

phid_star_matrix = [phid_star(phi, gamma) for phi in phi_values, gamma in␣
 ↪gamma_values]

# Plotting _d* as a 3D surface plot as a function of  and
surface(phi_values, gamma_values, phid_star_matrix, xlabel=" ", ylabel=" ",␣
 ↪zlabel=" _d*", title=" _d* as a Function of  and ", c=:viridis)
```



[191]:
```
function F(phi, gamma)
    return 4 *   * a^2 * sigma * xi * phid_star(phi,gamma)^2 - 3 * k_B * T *␣
 ↪log(phi_c + phid_star(phi,gamma) - phi - alpha * gamma^2)
end
```

```
F (generic function with 2 methods)
```

[192]:
```
function osmotic_pressure(phi)
    dF_dphi = ForwardDiff.derivative(g -> F(g, 0), phi)
    return (phi^2 / ((4 / 3) *   * a^3)) * dF_dphi
end
```

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

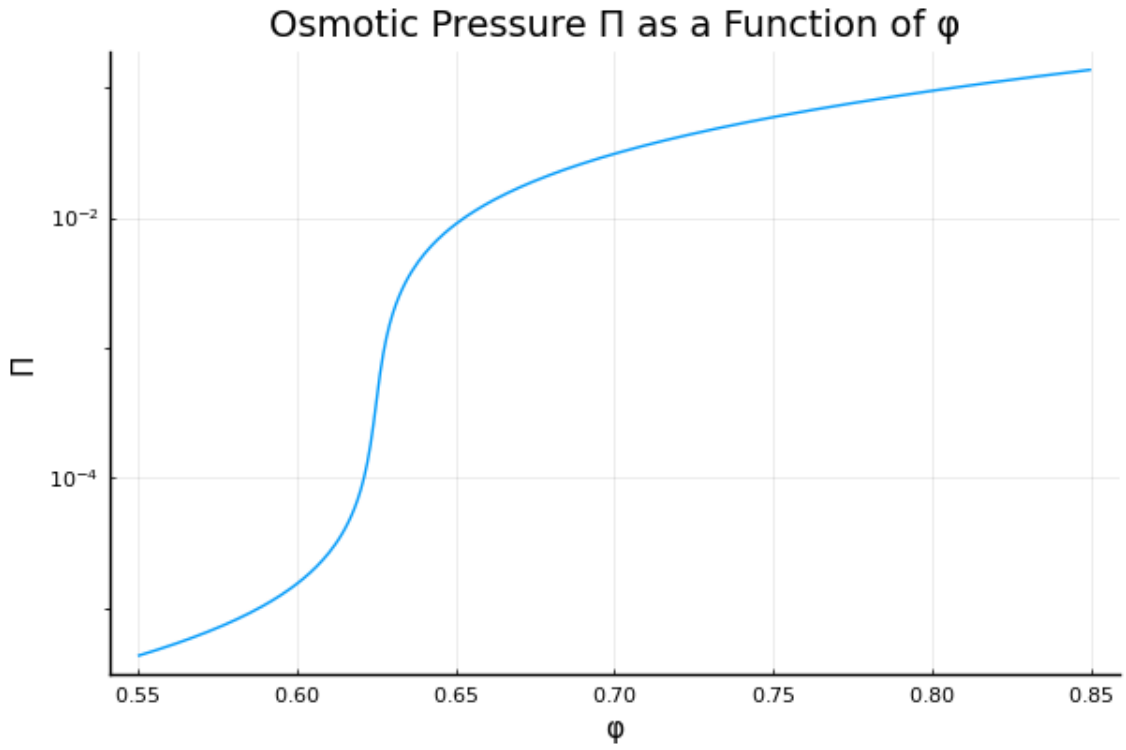
[193]:
```
phi_values = range(0.55, 0.85, length=300)
pi_values = [osmotic_pressure(phi) for phi in phi_values]
```

```
300-element Vector{Float64}:
    0.07977488659952206
    0.08115135498816604
    0.08256618512910183
    0.08402097954537907
    0.0855174312559691
    0.08705733025353747
    0.08864257054656678
```

```
0.0902751578240229
0.09195721780771027
0.09369100536537384

2389.1532394887586
2405.9245976851535
2422.755497618538
2439.646033398989
2456.5962991369543
2473.6063889432507
2490.6763969290423
2507.806417205848
2524.9965438855315
```

[194]:
```
plot(phi_values, (pi_values / sigma * a) , xlabel=" ", ylabel="Π",␣
↪title="Osmotic Pressure Π as a Function of  ", legend=false, yscale=:log)
```

```
Warning: scale log is unsupported with Plots.PyPlotBackend().
Choose from: [:identity, :ln, :log10, :log2]
@ Plots /Users/fanwei/.julia/packages/Plots/kLeqV/src/args.jl:1584
```

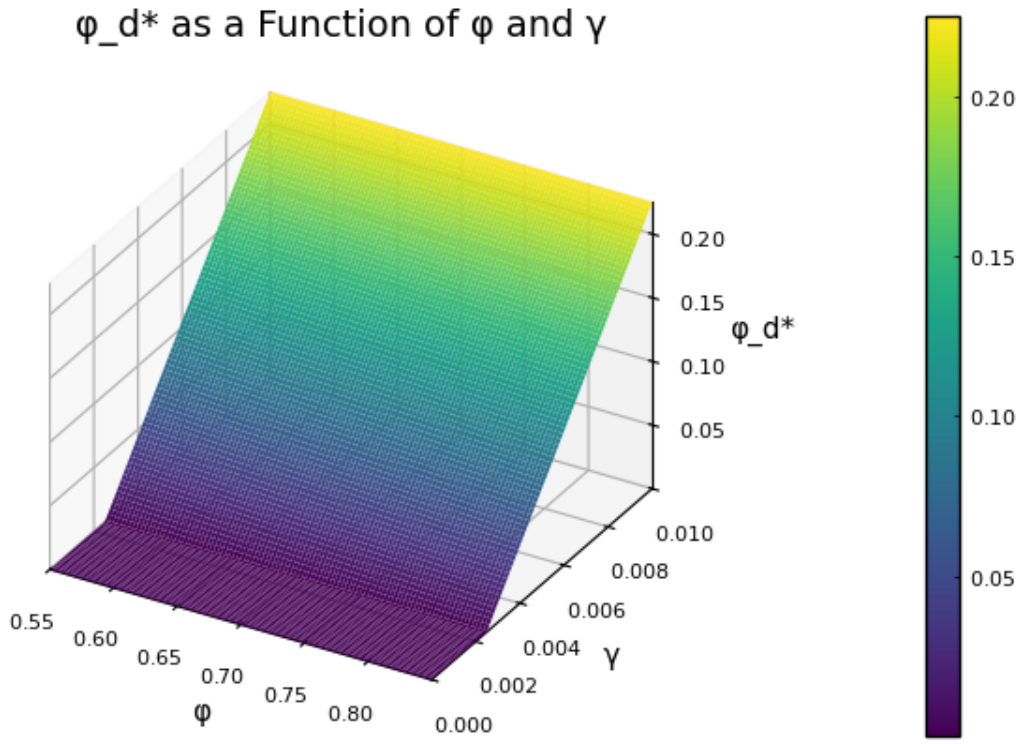


[195]:
```
# function phid_star(gamma)
#     term1 = phi - (phi_c - alpha * gamma^2)
#     return 0.5 * (term1 + sqrt(term1^2 + phi_T_squared))
```

```
# end

# gavec = 0:0.0001:0.01
# fvec = f.(gavec)
# dfvec = [ForwardDiff.derivative(ga -> f(ga),g) for g in gavec];

# p1 = plot(gavec, fvec)
# p2 = plot(gavec, dfvec)
# plot(p1,p2, layout = (1,2))
```

[196]:
```
function F(phi_d, phi, gamma)
    term = phi_c + phi_d - phi - alpha * gamma^2
    if term <= 0
        return Inf  # Return a large value to avoid invalid log
    end
    return 4 *   * a^2 * sigma * xi * phi_d^2 - 3 * k_B * T * log(term)
end

function find_phid_star(phi, gamma)
    f_prime(phi_d_vec) = ForwardDiff.derivative(phi_d -> F(phi_d, phi, gamma),␣
 ↪phi_d_vec[1])

    result = nlsolve(f_prime, [1e-15]; inplace=false)  # Initial guess without␣
 ↪bounds
    phi_d_star = result.zero[1]

    boundary_condition = phi_c - phi - alpha * gamma^2

    if boundary_condition > 0
        lower_bound = 0
    else
        lower_bound = phi + alpha * gamma^2 - phi_c
    end

    if phi_d_star < lower_bound
        return lower_bound
    end
    return phi_d_star
end

phi_values = range(0.55, 0.85, length=100)
gamma_values = range(0.0, 0.01, length=100)

phid_star_matrix = [find_phid_star(phi, gamma) for phi in phi_values, gamma in␣
 ↪gamma_values]

# Plotting _d* as a 3D surface plot as a function of  and
```

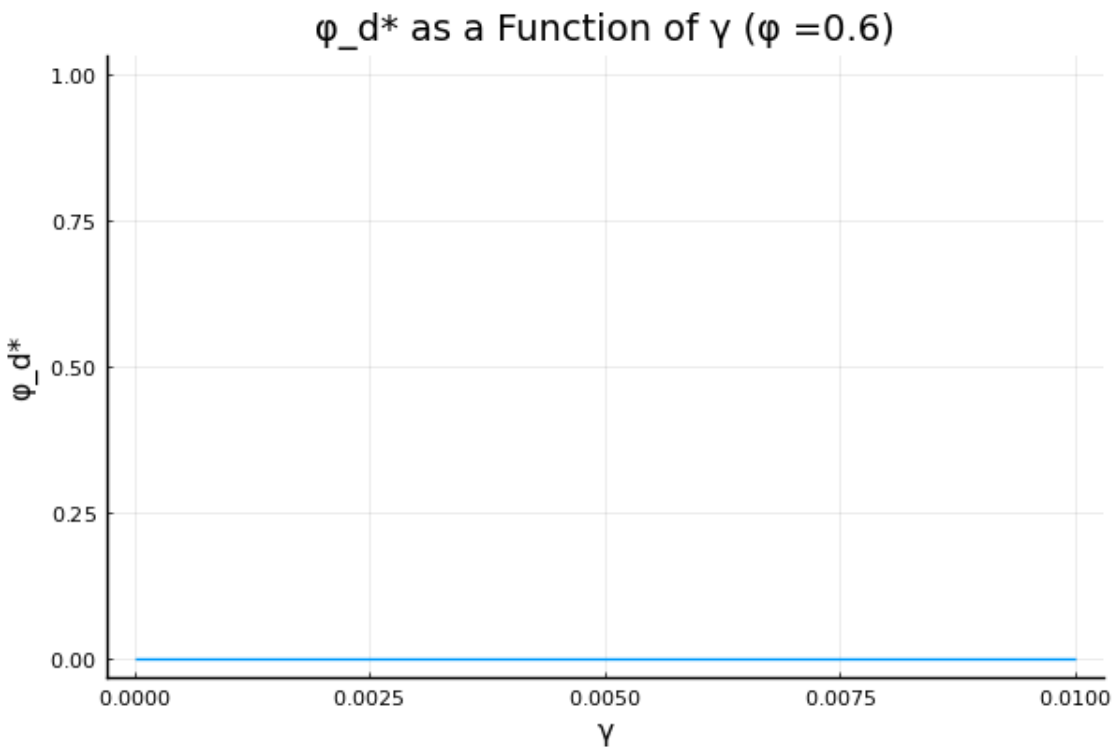
```
surface(phi_values, gamma_values, phid_star_matrix, xlabel=" ", ylabel=" ",␣
 ↪zlabel=" _d*", title=" _d* as a Function of   and  ", c=:viridis)

# Display the plot
plot!()
```



[197]:
```
find_phid_star(0.625, 0.001)
```

```
7.400000000545859e-7
```

[198]:
```
gamma_values = range(0.0, 0.01, length=50)
phi_fixed = 0.6

phid_star_values = [find_phid_star(phi_fixed, gamma) for gamma in gamma_values]

plot(gamma_values, phid_star_values, xlabel=" ", ylabel=" _d*", title=" _d* as a␣
 ↪Function of   (  =$phi_fixed)", legend=false)

plot!()
```

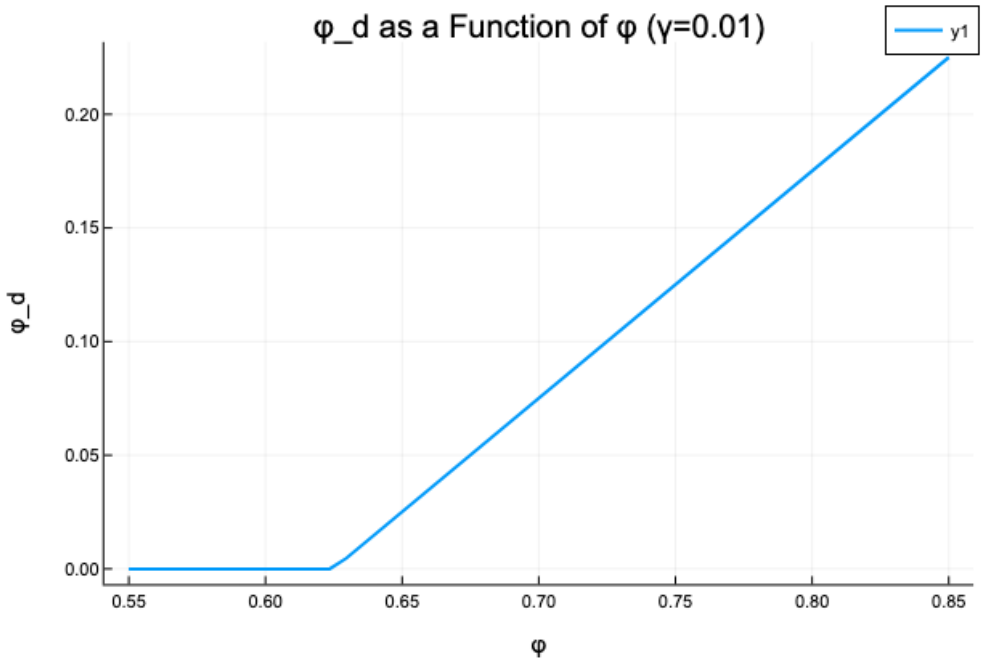


[217]:
```
function target_eqn(phi_d, phi, gamma)
    return 4 *   * a^2 * sigma * xi * phi_d^2 - 3 * k_B * T * log(phi_c + phi_d
 ↪+ phi - alpha * gamma^2)
end

function find_min_phi_d(phi, gamma)
    boundary_condition = phi_c - phi - alpha * gamma^2

    if boundary_condition > 0
        lower_bound = 0
    else
        lower_bound = phi + alpha * gamma^2 - phi_c
    end

    min_phi_d = lower_bound
    min_value = target_eqn(min_phi_d, phi, gamma)

    for phi_d in range(lower_bound, 0.35, length=50)
        current_value = target_eqn(phi_d, phi, gamma)
        if current_value < min_value
            min_value = current_value
            min_phi_d = phi_d
        end
```

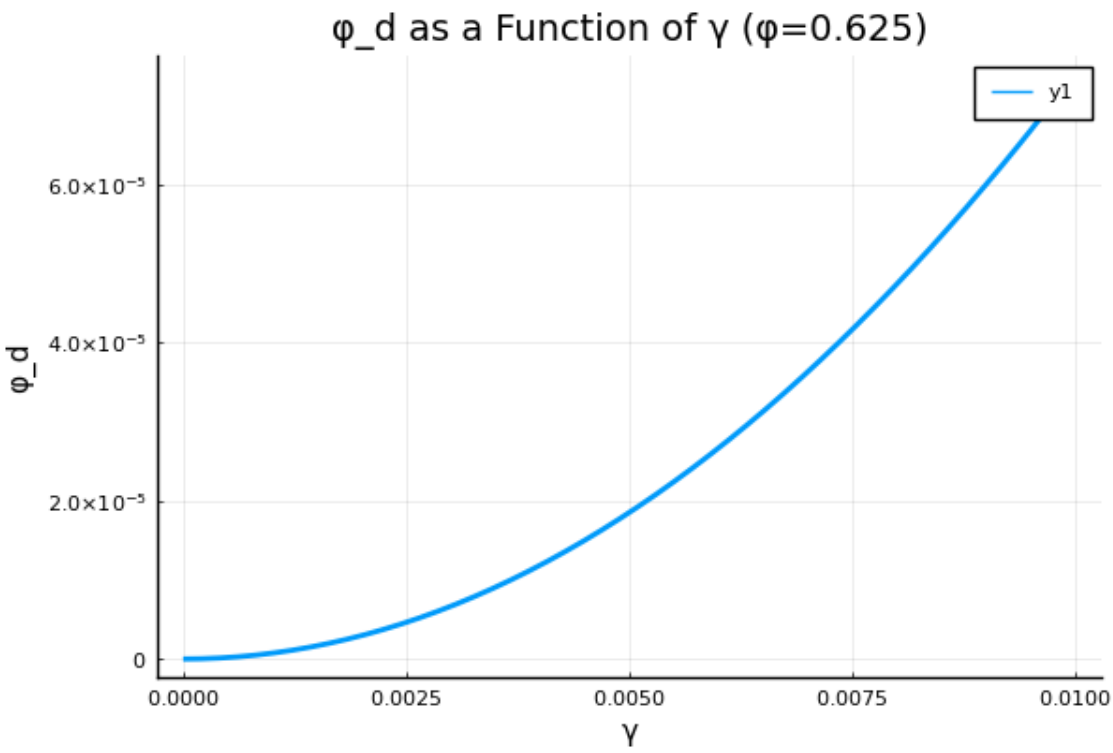
```
    end

    return min_phi_d, min_value
end


phi_values = range(0.55, 0.85, length=50)
gamma_values = range(0, 0.01, length=50)

phi_d_values = zeros(length(phi_values), length(gamma_values))

for i in 1:length(phi_values)
    for j in 1:length(gamma_values)
        phi = phi_values[i]
        gamma = gamma_values[j]
        phi_d, _ = find_min_phi_d(phi, gamma)
        phi_d_values[i, j] = phi_d
    end
end

phi_grid = repeat(phi_values', 1, length(gamma_values))
gamma_grid = repeat(gamma_values, 1, length(phi_values))
surface(phi_grid, gamma_grid, phi_d_values', xlabel=" ", ylabel=" ",␣
 ↪zlabel=" _d", title=" _d as a Function of   and  ")
```

```
attempt to save state beyond implementation limit
attempt to save state beyond implementation limit
attempt to save state beyond implementation limit
```

[212]:
```
gamma_fixed = 0.01

phi_values = range(0.55, 0.85, length=50)

phi_d_values = zeros(length(phi_values))

for i in 1:length(phi_values)
    phi = phi_values[i]
    phi_d, _ = find_min_phi_d(phi, gamma_fixed)
    phi_d_values[i] = phi_d
end

plot(phi_values, phi_d_values, xlabel=" ", ylabel=" _d", title=" _d as a␣
 ↪Function of   ( =$gamma_fixed)", lw=2)
```



[202]:
```
phi_fixed = 0.625

gamma_values = range(0, 0.01, length=50)

phi_d_values = zeros(length(gamma_values))

for i in 1:length(gamma_values)
    gamma = gamma_values[i]
    phi_d, _ = find_min_phi_d(phi_fixed, gamma)
    phi_d_values[i] = phi_d
end

plot(gamma_values, phi_d_values, xlabel=" ", ylabel=" _d", title=" _d as a␣
 ↪Function of   ( =$phi_fixed)", lw=2)
```


φ_d as a Function of γ (φ=0.625)
y1
6.0×10⁻⁵
4.0×10⁻⁵
2.0×10⁻⁵
0
φ_d
0.0000
0.0025
0.0050
0.0075
0.0100
γ