```julia
using LaTeXStrings
```

```julia
using Plots
```

```julia
using Markdown
```

```julia
using PlutoUI
```

```
GRBackend()
```

```julia
gr()
```

Lane-Emden Equation

$$\dfrac{\text{d}}{\text{d}\xi} (\xi^2 \dfrac{\text{d}\theta}{\text{d}\xi}) = -\xi^2\theta^n$$

```julia
md"""
Lane-Emden Equation

``
\dfrac{\text{d}}{\text{d}\xi} ( \xi^2 \dfrac{\text{d}\theta}{\text{d}\xi} ) = -\xi^2\theta^n
``
\
\

"""
```

Separation of variables

$$\dfrac{\text{d}y}{\text{d}\xi} = \dfrac{z}{\xi^2}$$

$$\dfrac{\text{d}z}{\text{d}\xi} = -\xi^2y^n$$

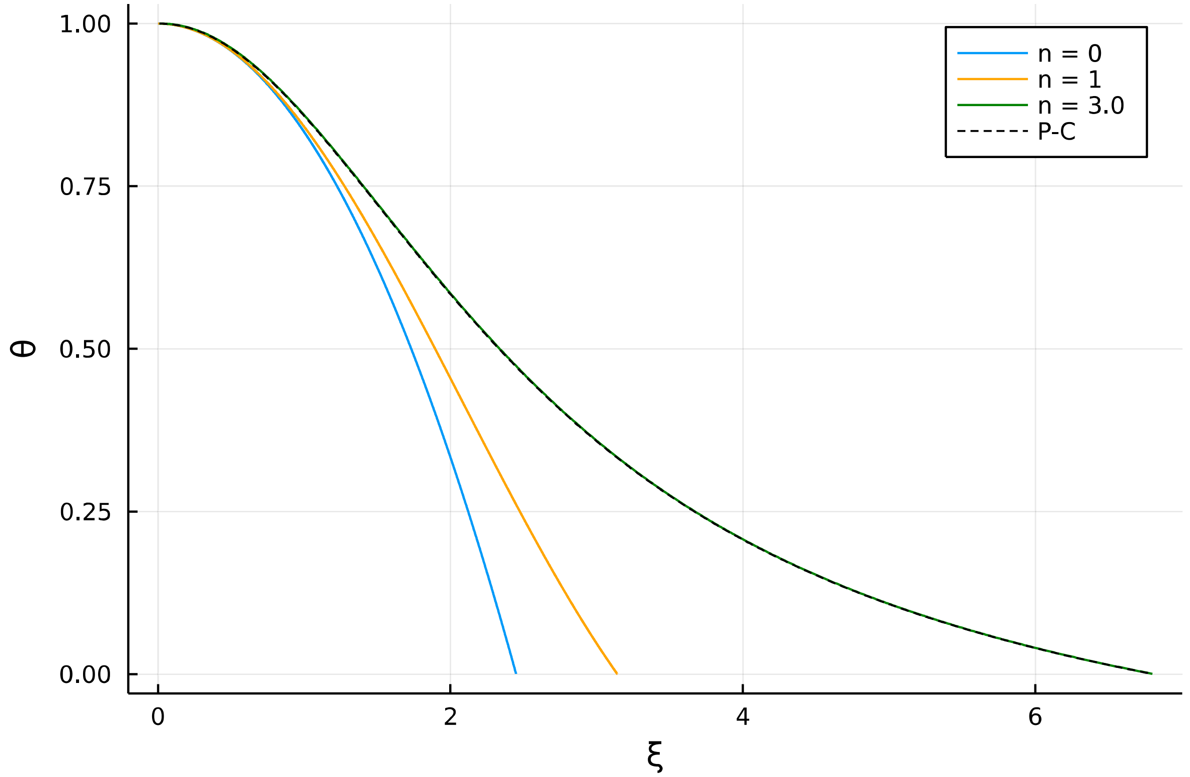
```
md"""
Separation of variables

``
\dfrac{\text{d}y}{\text{d}\xi}   =  \dfrac{z}{\xi^2}
``

``
\dfrac{\text{d}z}{\text{d}\xi}  = -\xi^2y^n
``
\
\
"""
```

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

```julia
function solveLaneEmden(log_delta_xi=-4, n=3)
    delta_xi = 10.0^log_delta_xi

    # Inner boundary condition
    y0 = 1 - delta_xi^2/6
    z0 = -delta_xi^3/3

    println([y0, z0])

    ys  = [y0]
    zs  = [z0]
    xis = [delta_xi]
    ycs = [y0]
    zcs = [z0]

    while true
        y  =  last(ys)
        z  =  last(zs)
        xi =  last(xis)
        yc =  last(ycs)
        zc =  last(zcs)

           ## Primitive method
        yi = y + delta_xi * z/xi^2
        zi = z + delta_xi * -xi^2*y^n

        ## Predictor-corrector technique
        xii = xi + delta_xi
        yci = yc + 1/2 * delta_xi * (z/xi^2 + zi/xii^2)
        zci = zc + 1/2 * delta_xi * (-xi^2*y^n - xi^2*yi^n)

        # Outer boundary condition
        if (yi < 1e-10 || yci < 1e-10)
            break
        end

        push!(xis, xii)
        push!(ys, yi)
        push!(zs, zi)
        push!(ycs, yci)
        push!(zcs, zci)

    end

    return (xis, ys, ycs)

end
```

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

```julia
function plotLaneEmden(log_delta_xi=-4, n=3)
    xis, ys, ycs = solveLaneEmden(log_delta_xi, n)

    xi2 = range(0,sqrt(6),step=1e-3)
    # @. will add . to every operator
    Plots.plot(xi2, 1 .- xi2.^2/6, label="n = 0")

    xi2 = range(0, pi, step=1e-3)
    Plots.plot!(xi2, sin.(xi2)./xi2, linecolor = :orange, label="n = 1")

    Plots.plot!(xis, ys,  linecolor = :green, label="n = $n")
    Plots.plot!(xis, ycs, linecolor = :black, linestyle = :dash, label="P-C")

    Plots.xlabel!("ξ")
    Plots.ylabel!("θ")

end
```



```julia
plotLaneEmden(log_delta_xi, n)
```

n: 3.0 log(Δξ): -2.0

```julia
md"n: $nslider $n
log(Δξ): $log_delta_xi_slider $log_delta_xi"
```

```julia
begin
nslider = @bind n Slider(0:0.01:4.99, default=3)
log_delta_xi_slider = @bind log_delta_xi Slider(-6:0.01:0.1, default=-2)
end;
```