Julia Notebook

Need Help?

- Learning: https://julialang.org/learning/
- Documentation: https://docs.julialang.org/
- Questions & Discussions:
  - https://discourse.julialang.org/
  - http://julialang.slack.com/
  - https://stackoverflow.com/questions/tagged/julia

If you ever ask for help or file an issue about Julia, you should generally provide the output of `versioninfo()` .

Checking the Installation

The `versioninfo()` function should print your Julia version and some other info about the system:

In [1]:

```
versioninfo()
```

```
Julia Version 1.6.0
Commit f9720dc2eb (2021-03-24 12:55 UTC)
Platform Info:
  OS: Linux (x86_64-pc-linux-gnu)
  CPU: Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz
  WORD_SIZE: 64
  LIBM: libopenlibm
  LLVM: libLLVM-11.0.1 (ORCJIT, ivybridge)
Environment:
  JULIA_NUM_THREADS = 4
```

In [2]:

```
using Plots, Optim, Flux, DiffEqFlux, DifferentialEquations, LaTeXStrings, DiffE
```

1D Heat Equation Model Problem

$$\frac{\partial^2 T}{\partial z^2} = \varepsilon(T, T_\infty)(T_\infty^4 - T^4) + 0.5(T_\infty - T)$$

Where:

$$\varepsilon(T, T_\infty) = -[1 + 5\sin(\frac{3\pi}{200}T) + \exp(0.02T)] \times 10^{-4}$$

Tutorials and documentation used heavily for this notebook:
https://diffeqflux.sciml.ai/dev/Flux/#Using-Flux-Chain-neural-networks-with-Flux.train!-1
https://diffeq.sciml.ai/v3.2.0/tutorials/bvp_example.html

In [3]:

```
function heat!(du,u,p,t)
        Tinf = p[1]
        T = u[1]
        dT = u[2]
        du[1] = dT
        du[2] = -(1.0+5.0*sin(3.0*pi*T/200.0)+exp(0.02*T))*10.0^(-4.0)*(Tinf^4.0
end

p = [50.0]
u0 = [p[1], 0.0]
tspan = (0.0,1.0)
dt = 0.1

function bc!(residual, u, p, t)
        residual[1] = u[1][1]
        residual[2] = u[end][1]
end
prob = TwoPointBVProblem(heat!,bc!,[p[1],0.0],tspan,p)
```

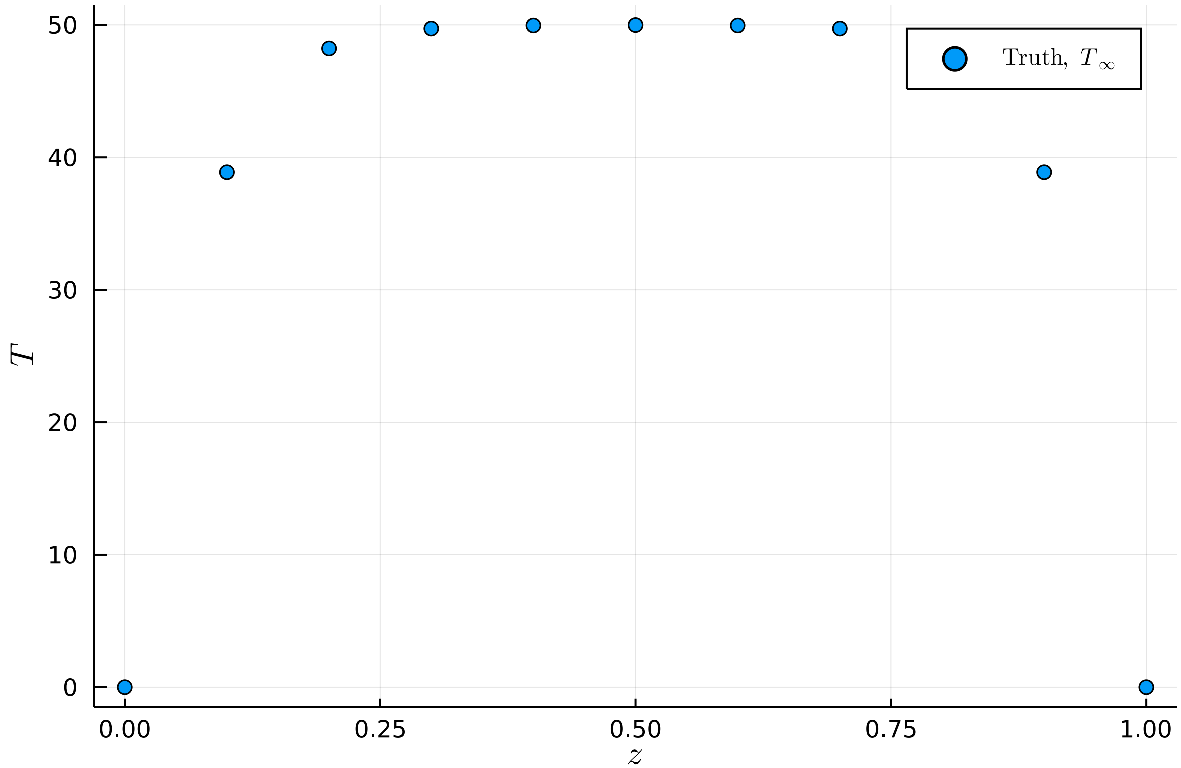
Out[3]:

```
BVProblem with uType Vector{Float64} and tType Float64. In-place: true
timespan: (0.0, 1.0)
u0: 2-element Vector{Float64}:
 50.0
  0.0
```

In [4]:

```
sol = solve(prob,MIRK4(),dt=dt)
scatter(sol.t,sol[1,:],label=L"\mathrm{Truth}, \ T_\infty",xlabel=L"z",ylabel=L'
```

Out[4]:



In [5]:

```
z = sol.t
truth = sol[1,:]
function predict_truth()
        solve(prob,MIRK4(),p=p,dt = dt)[1,:]
end
```

Out[5]: `predict_truth (generic function with 1 method)`

Imperfect Model

$\frac{\partial^2 T}{\partial z^2} = \epsilon_0 (T_\infty^4 - T^4)$

Where:

$\epsilon_0 = -10^{-4}$

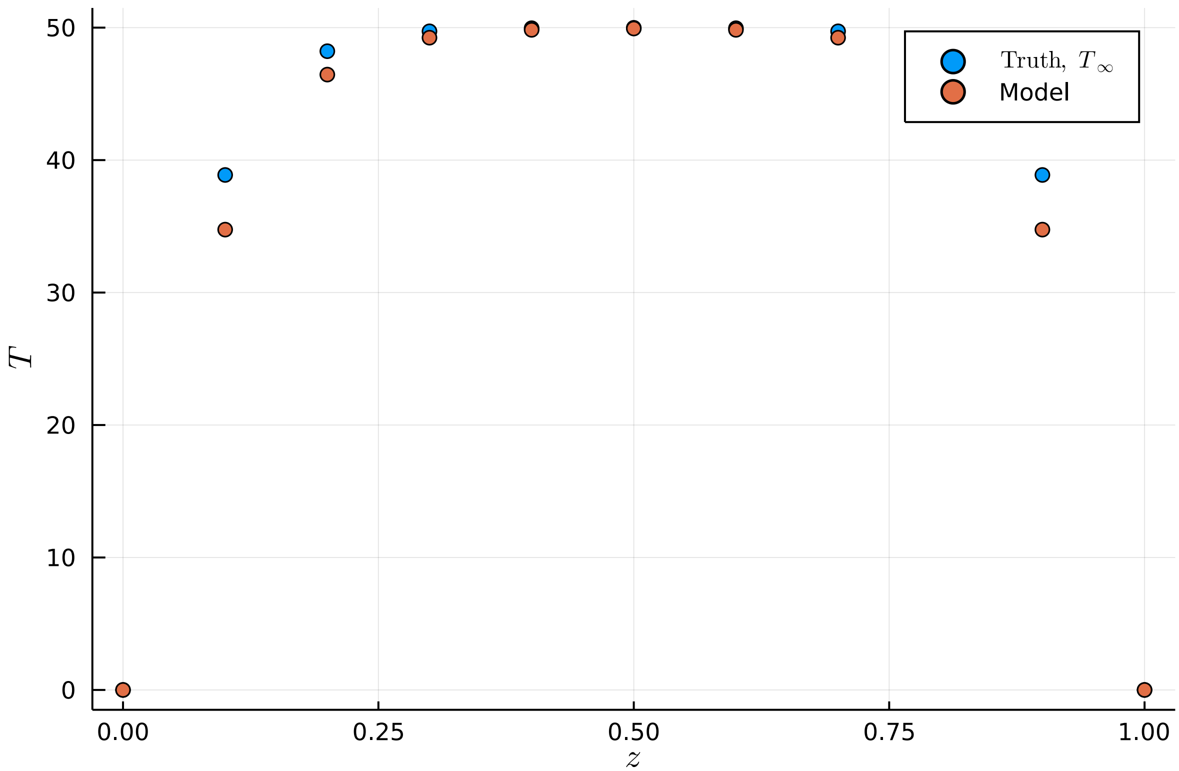
In [6]:

```
function model!(du,u,p,t)
        Tinf = p[1]
        T = u[1]
        dT = u[2]
        du[1] = dT
        du[2] = -5.0*10.0^(-4.0)*(Tinf^4.0-T^4.0)
end
pred = TwoPointBVProblem(model!,bc!,[p[1],0.0],tspan,p)
function predict_model()
        solve(pred,MIRK4(),p=p,dt = dt)[1,:]
end
model_prediction = predict_model()
```

Out[6]:
```
11-element Vector{Float64}:
  0.0
 34.75171808167662
 46.45491687856484
 49.240589942406764
 49.83382512154275
 49.93334547778014
 49.83382512154275
 49.240589942406764
 46.45491687856484
 34.751718081676614
  0.0
```

In [7]:
```
scatter!(sol.t,model_prediction,label="Model")
```

Out[7]:



Augment Imperfect Model

$$\frac{\partial^2 T}{\partial z^2} = \beta(T, T_\infty)\epsilon_0(T_\infty^4 - T^4)$$

Where:

$$\epsilon_0 = -5 \times 10^{-4}$$

And $\beta(T, T_\infty)$ is a function which augments the imperfect model and takes $T,\ T_\infty$ as inputs. In this application, we choose to find $\beta(T, T_\infty)$ using neural networks. Note that $\beta(T, T_\infty)$ has the following analytic solution:

$$\beta(T, T_\infty) = \frac{1}{\epsilon_0}\left[1 + 5\sin(\frac{3\pi}{200}T) + \exp(0.02T)\right] \times 10^{-4} + \frac{h}{\epsilon_0}\frac{T_\infty - T}{T_\infty^4 - T^4}$$

Initialize the neural network (using Julias Flux.ml) we use to augment our model, which takes two inputs and has a single output. We also use Flux.destructure to separate out the variables required for training (the weights).

In [8]:

```
nodes = 20
        dudt2 = Chain(x->[x[1],x[2]],
        Dense(2,nodes,tanh,initW = zeros, initb = zeros),
        #Dense(nodes,nodes,tanh,initW = zeros, initb = zeros), #Uncomment to add
        Dense(nodes,1, initW = zeros, initb = zeros))
  g,re = Flux.destructure(dudt2)
  re(g)([50.0,0.0])
```

Out[8]:

```
1-element Vector{Float64}:
 0.0
```

In [9]:

```
function aug_model(du,u,p,t)
        global re
  Tinf = p[3]
  g = p[4:end]
        T = u[1]
        dT = u[2]
        du[1] = dT
        du[2] = (1.0+re(g)([(T-Tinf)/Tinf,Tinf/50.0])[1])*-5.0*10.0^(-4.0)*(Tinf
end
```

Out[9]:

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

In [10]:

```
θ=[u0;p;g]
augmented = TwoPointBVProblem(aug_model,bc!,[p[1],0.0],tspan,θ)# sensealg=Forwar
```

Out[10]:

```
BVProblem with uType Vector{Float64} and tType Float64. In-place: true
timespan: (0.0, 1.0)
u0: 2-element Vector{Float64}:
 50.0
  0.0
```

In [11]:

```
function predict_n_ode(θ)
  #solve(pred,MIRK4(),p=p,dt = 0.01)[1,:]
  solve(augmented,MIRK4(),p=θ,dt=dt)[1,:]
end
predict_n_ode(θ)
```

Out[11]:

```
11-element Vector{Float64}:
  0.0
 34.75171808167662
 46.45491687856484
 49.240589942406764
 49.83382512154275
 49.93334547778014
 49.83382512154275
 49.240589942406764
 46.45491687856484
 34.751718081676614
  0.0
```

Define the loss function and try to get the gradient by using Zygote.gradient. Note that this currently is not implemented and returns an error.

In [12]:
```
function loss_n_ode(θ)
    pred = predict_n_ode(θ)
    loss = sum(abs2,truth .- pred)
    loss,pred
end
l, pred = loss_n_ode(θ)
loss(θ)

display(Zygote.gradient(loss,θ))
```

```
Continuous adjoint sensitivities are only supported for ODE/SDE/RODE problems.

Stacktrace:
  [1] error(s::String)
    @ Base ./error.jl:33
  [2] _adjoint_sensitivities(sol::ODESolution{Float64, 2, Vector{Vector{Float6
4}}, Nothing, Nothing, Vector{Float64}, Nothing, BVProblem{Vector{Float64}, Tupl
e{Float64, Float64}, true, Vector{Float64}, ODEFunction{true, typeof(aug_model),
LinearAlgebra.UniformScaling{Bool}, Nothing, Nothing, Nothing, Nothing, Nothing,
Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, typeof(SciMLBase.
DEFAULT_OBSERVED), Nothing}, SciMLBase.TwoPointBVPFunction{typeof(bc!)}, SciMLBa
se.StandardBVProblem, Base.Iterators.Pairs{Union{}, Union{}, Tuple{}, NamedTuple
{(), Tuple{}}}}, MIRK4, SciMLBase.LinearInterpolation{Vector{Float64}, Vector{Ve
ctor{Float64}}}, Nothing}, sensealg::InterpolatingAdjoint{0, true, Val{:centra
l}, Bool, Bool}, alg::MIRK4, g::DiffEqSensitivity.var"#df#168"{Matrix{Float64},
Vector{Float64}, Colon}, t::Vector{Float64}, dg::Nothing; abstol::Float64, relto
l::Float64, checkpoints::Vector{Float64}, corfunc_analytical::Nothing, callbac
k::Nothing, kwargs::Base.Iterators.Pairs{Symbol, Float64, Tuple{Symbol}, NamedTu
ple{(:dt,), Tuple{Float64}}})
    @ DiffEqSensitivity ~/.julia/packages/DiffEqSensitivity/Yn6oc/src/sensitivit
y_interface.jl:32
  [3] adjoint_sensitivities(::ODESolution{Float64, 2, Vector{Vector{Float64}}, N
othing, Nothing, Vector{Float64}, Nothing, BVProblem{Vector{Float64}, Tuple{Floa
t64, Float64}, true, Vector{Float64}, ODEFunction{true, typeof(aug_model), Linea
rAlgebra.UniformScaling{Bool}, Nothing, Nothing, Nothing, Nothing, Nothing, Noth
ing, Nothing, Nothing, Nothing, Nothing, Nothing, Nothing, typeof(SciMLBase.DEFA
ULT_OBSERVED), Nothing}, SciMLBase.TwoPointBVPFunction{typeof(bc!)}, SciMLBase.S
tandardBVProblem, Base.Iterators.Pairs{Union{}, Union{}, Tuple{}, NamedTuple{(),
Tuple{}}}}, MIRK4, SciMLBase.LinearInterpolation{Vector{Float64}, Vector{Vector
{Float64}}}, Nothing}, ::MIRK4, ::Vararg{Any, N} where N; sensealg::Interpolatin
gAdjoint{0, true, Val{:central}, Bool, Bool}, kwargs::Base.Iterators.Pairs{Symbo
l, Union{Nothing, Float64}, Tuple{Symbol, Symbol}, NamedTuple{(:callback, :dt),
Tuple{Nothing, Float64}}})
    @ DiffEqSensitivity ~/.julia/packages/DiffEqSensitivity/Yn6oc/src/sensitivit
y_interface.jl:6
  [4] (::DiffEqSensitivity.var"#adjoint_sensitivity_backpass#167"{Base.Iterator
s.Pairs{Symbol, Float64, Tuple{Symbol}, NamedTuple{(:dt,), Tuple{Float64}}}, MIR
K4, InterpolatingAdjoint{0, true, Val{:central}, Bool, Bool}, Vector{Float64}, V
ector{Float64}, Tuple{}, Colon, NamedTuple{(:dt,), Tuple{Float64}}})(Δ::Matrix{F
loat64})
    @ DiffEqSensitivity ~/.julia/packages/DiffEqSensitivity/Yn6oc/src/concrete_s
olve.jl:179
  [5] #98#back
    @ ~/.julia/packages/ZygoteRules/OjfTt/src/adjoint.jl:65 [inlined]
  [6] #178
    @ ~/.julia/packages/Zygote/RxTZu/src/lib/lib.jl:194 [inlined]
  [7] (::Zygote.var"#1686#back#180"{Zygote.var"#178#179"{Tuple{NTuple{6, Nothin
g}, Tuple{Nothing}}, DiffEqBase.var"#98#back#74"{DiffEqSensitivity.var"#adjoint_
sensitivity_backpass#167"{Base.Iterators.Pairs{Symbol, Float64, Tuple{Symbol}, N
amedTuple{(:dt,), Tuple{Float64}}}, MIRK4, InterpolatingAdjoint{0, true, Val{:ce
ntral}, Bool, Bool}, Vector{Float64}, Vector{Float64}, Tuple{}, Colon, NamedTupl
e{(:dt,), Tuple{Float64}}}}}})(Δ::Matrix{Float64})
    @ Zygote ~/.julia/packages/ZygoteRules/OjfTt/src/adjoint.jl:59
  [8] Pullback
```

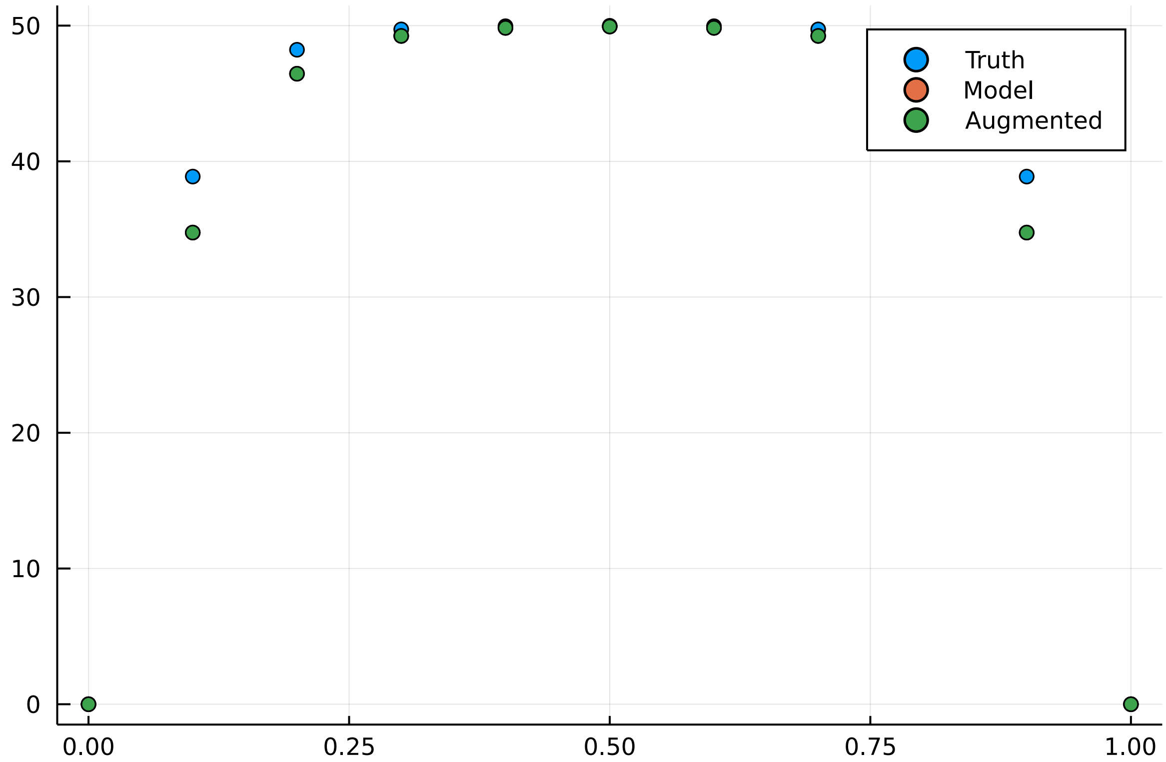
```
    @ ~/.julia/packages/DiffEqBase/qntkj/src/solve.jl:70 [inlined]
  [9] (::typeof(∂(#solve#57)))(Δ::Matrix{Float64})
    @ Zygote ~/.julia/packages/Zygote/RxTZu/src/compiler/interface2.jl:0
 [10] (::Zygote.var"#178#179"{Tuple{NTuple{6, Nothing}, Tuple{Nothing}}, typeof
(∂(#solve#57))})(Δ::Matrix{Float64})
    @ Zygote ~/.julia/packages/Zygote/RxTZu/src/lib/lib.jl:194
 [11] (::Zygote.var"#1686#back#180"{Zygote.var"#178#179"{Tuple{NTuple{6, Nothin
g}, Tuple{Nothing}}, typeof(∂(#solve#57))}})(Δ::Matrix{Float64})
    @ Zygote ~/.julia/packages/ZygoteRules/OjfTt/src/adjoint.jl:59
 [12] Pullback
    @ ~/.julia/packages/DiffEqBase/qntkj/src/solve.jl:68 [inlined]
 [13] (::typeof(∂(solve##kw)))(Δ::Matrix{Float64})
    @ Zygote ~/.julia/packages/Zygote/RxTZu/src/compiler/interface2.jl:0
 [14] Pullback
    @ ./In[11]:3 [inlined]
 [15] (::typeof(∂(predict_n_ode)))(Δ::Vector{Float64})
    @ Zygote ~/.julia/packages/Zygote/RxTZu/src/compiler/interface2.jl:0
 [16] Pullback (repeats 2 times)
    @ ./In[12]:7 [inlined]
 [17] (::Zygote.var"#41#42"{typeof(∂(loss))})(Δ::Float64)
    @ Zygote ~/.julia/packages/Zygote/RxTZu/src/compiler/interface.jl:41
 [18] gradient(f::Function, args::Vector{Float64})
    @ Zygote ~/.julia/packages/Zygote/RxTZu/src/compiler/interface.jl:59
 [19] top-level scope
    @ In[12]:13
 [20] eval
    @ ./boot.jl:360 [inlined]
 [21] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String,
filename::String)
    @ Base ./loading.jl:1094
```

Plot the results using the full physics equation, the imperfect model, and the augmented model before training. Note that the weights of the neural network are initialized to zero so the initial augmented model gives identical results to the model. (Neural network returns 0.0)

In [13]:

```
show_result = function (θ, l, pred)
        scatter(z,truth,label="Truth")
    scatter!(z,model_prediction,label="Model")
        scatter!(z,predict_n_ode(θ),label="Augmented")
end
show_result(θ, l, pred)
```

Out[13]:



Define a callback function required by the optimizer that returns the loss given the current weights of the neural network.

In [14]:
```
cb = function (θ,l,pred;doplot=false)
        display(l)
        #pl = plot(sol)
        return false
end
cb(θ,l,pred)
```

```
40.77726405588571
```

Out[14]:
```
false
```

Run the optimizer. Note that at the moment we are limited to finite differences for the gradient which is fairly inefficient for this problem.

In [15]:
```
#res = DiffEqFlux.sciml_train(loss_n_ode, θ, LBFGS(), cb = cb)
#res = DiffEqFlux.sciml_train(loss_n_ode, θ, ADAM(0.1), cb = cb, maxiters=100)

f = OptimizationFunction(loss, GalacticOptim.AutoFiniteDiff())
prob = OptimizationProblem(f,θ)
sol = solve(prob,BFGS())
```

Out[15]:
```
u: 84-element Vector{Float64}:
 50.0
  0.0
 49.99727534853881
  0.0
  0.0
  0.0
  0.0
  0.0
```

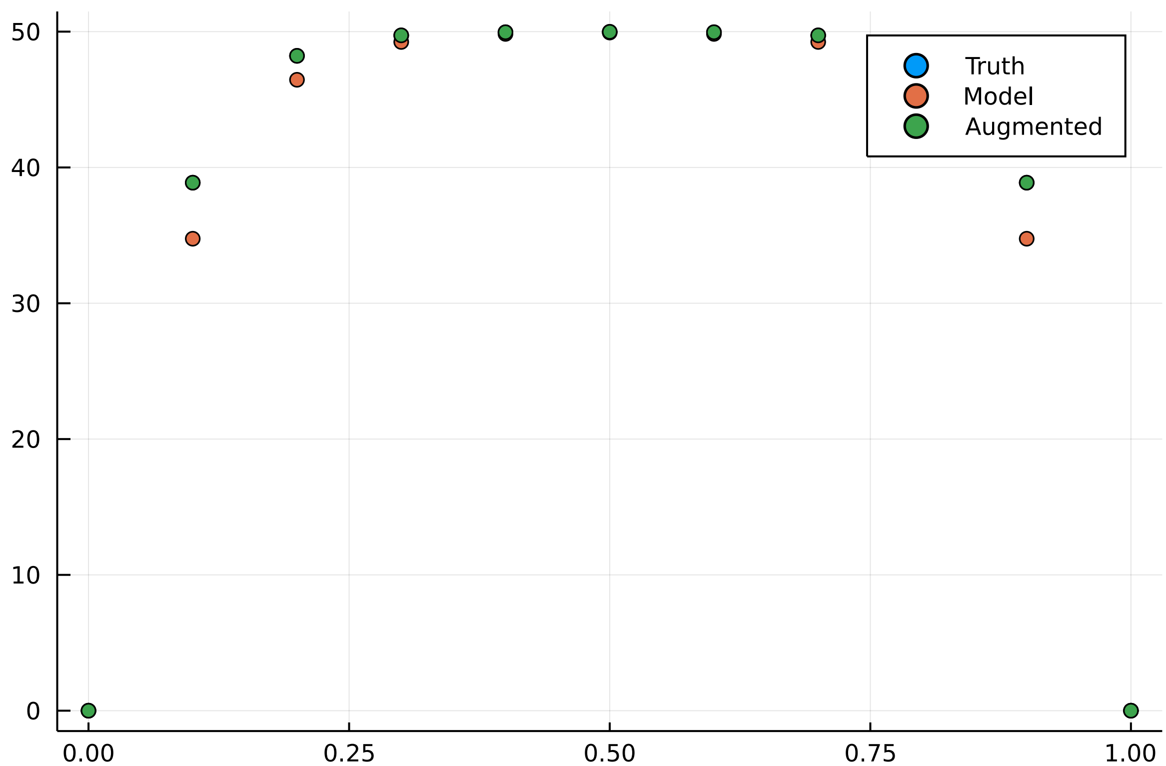
```
0.0
0.0
0.0
0.0
0.0
 ⋮
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.0
0.5029771808495614
```

Plot the results of the full physics equation (truth), the model, and the augmented model which is now using the optimal (trained) neural network weights.

In [16]:

```
#show_result(res.minimizer,loss_n_ode(res.minimizer)...)
show_result(sol.u,loss_n_ode(sol.u)...)
```

Out[16]:

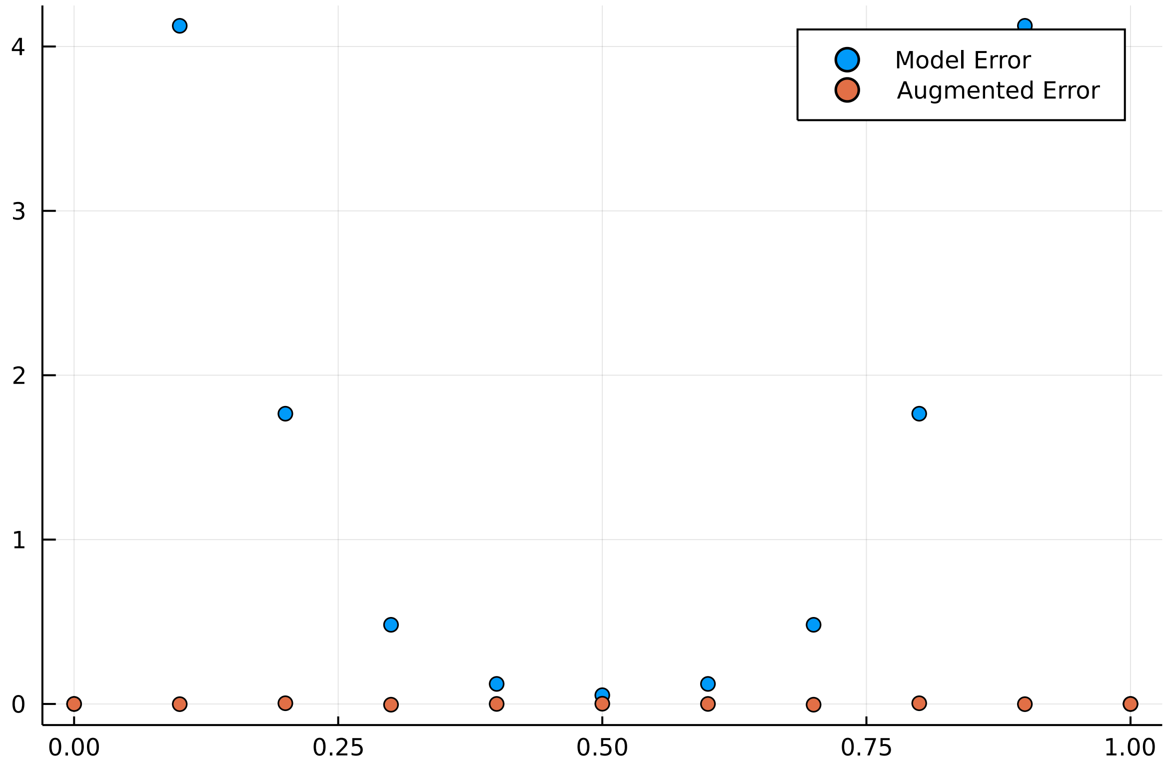


Also plot the error showing that the augmented model very nearly matches the full physics (truth) equation.

In [17]:

```
scatter(z,truth-predict_n_ode(θ),label="Model Error")
scatter!(z,truth-predict_n_ode(sol.u),label="Augmented Error")
```

Out[17]:


Model Error
Augmented Error
0
1
2
3
4
0.00
0.25
0.50
0.75
1.00