18.337 Final Project: Solving the Grad-Shafranov Equation w/ NeuralPDE.jl

Allen Wang

May 15, 2023

1 Background

Tokamaks are magnetic confinement fusion (MCF) devices that use large magnets to confine the plasma in a toroidal, donut-like, shape and have been the primary focus of fusion energy research for the past several decades. Plasmas are inherently highly unstable, thus fast real-time control algorithms are necessary to make adjustments to the magnets to maintain plasma stability. One necessary condition is to keep the plasma in an *Ideal Magnetohydrodynamic (MHD) Equilibrium*, that is, the plasma must satisfy the following system of partial differential equations in 3D space (PDEs):

$$\mathbf{J} \times \mathbf{B} = \nabla p \tag{1}$$

$$\nabla \times \mathbf{B} = \mu_0 \mathbf{J} \tag{2}$$

$$\nabla \cdot \mathbf{B} = 0 \tag{3}$$

where $\mathbf{J}$ is the current density, $\mathbf{B}$ is the magnetic field, and p is the pressure. Tokamaks are symmetric in the sense that every "slice of the donut" is the same. By leveraging this symmetry, Ideal MHD equilibria for tokamaks are described by the 2D Grad-Shafranov equation [3, 4]:

$$\frac{\partial^2 \psi(r,z)}{\partial r^2} - \frac{1}{r}\frac{\partial \psi(r,z)}{\partial r} + \frac{\partial^2 \psi(r,z)}{\partial z^2} = -\mu_0 r^2 \frac{dp(\psi)}{d\psi} - \frac{1}{2}\frac{d}{d\psi}(F^2(\psi)) \tag{4}$$

where ψ is a quantity known as the *poloidal magnetic flux* and is related to the magnetic field in the poloidal plane (see Figure 1 for a description), p is the pressure, and $F = rB_\theta$ where B_θ is a component of the magnetic field. Plasma control systems often need access to the full ψ contours for control tasks, e.g. controlling the shape of the ψ contours [1], however only ψ values at the edge can be directly measured and thus ψ everywhere else must be inferred, a problem known as *equilibrium reconstruction.* This is done with external measurements, or a priori information on, p, F, and boundary conditions on ψ and then solving the Grad-Shafranov equation. The challenging real-time constraints generally force compromises on real-time algorithms solving the Grad-Shafranov equation, and prior works have explored training neural networks on offline, higher fidelity, solutions to obtain better real-time results [5]. Instead of training on offline solutions to another solver, the goal of this work is to instead obtain a general real-time capable neural network solver for the Grad-Shafranov equation using the tools available in NeuralPDE.jl [6].

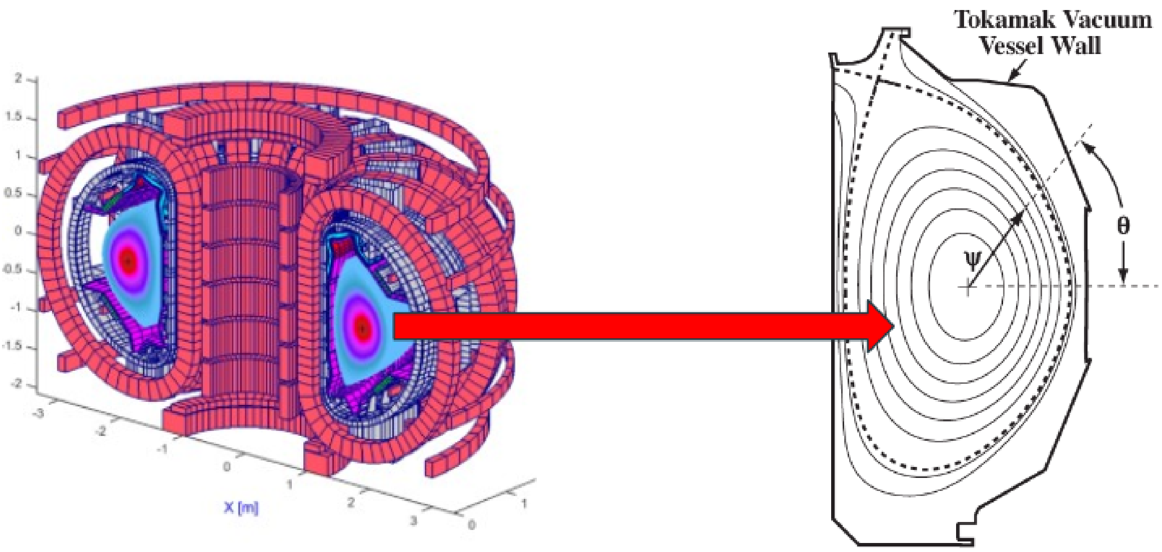


Figure 1: (Left) Cross-section of a tokamak showing magnets and plasma and (right) a "poloidal plane" showing contours of constant ψ.

2 Problem Setup

For an initial test problem, I tried to replicate example 5 in FreeGS, an open source Grad-Shafranov solver [2]. I obtained the p and F profiles for that example (Figure 2). The primary challenge of this problem relative to existing examples is that p and F must be constant with respect to normalized flux:

$$\psi_n(r,z) \equiv \frac{\psi(r,z)}{\max_{r,z}\psi(r,z)} \tag{5}$$

That is, the following quantities are fixed:

$$p(\psi_n) \quad F(\psi_n) \tag{6}$$

However, the Grad-Shafranov equation has the following:

$$\frac{dp(\psi)}{d\psi} \quad \frac{d}{d\psi}(F^2(\psi)) \tag{7}$$

To address this problem, I introduce a scale factor c and added the following additional loss term:

$$|| \frac{1}{\max_{r,z}\psi(r,z)} - c|| \tag{8}$$

in doing so, I identified a documentation inconsistency with the code that will now be fixed `https://github.com/SciML/NeuralPDE.jl/issues/679`. With this c parameter, we can

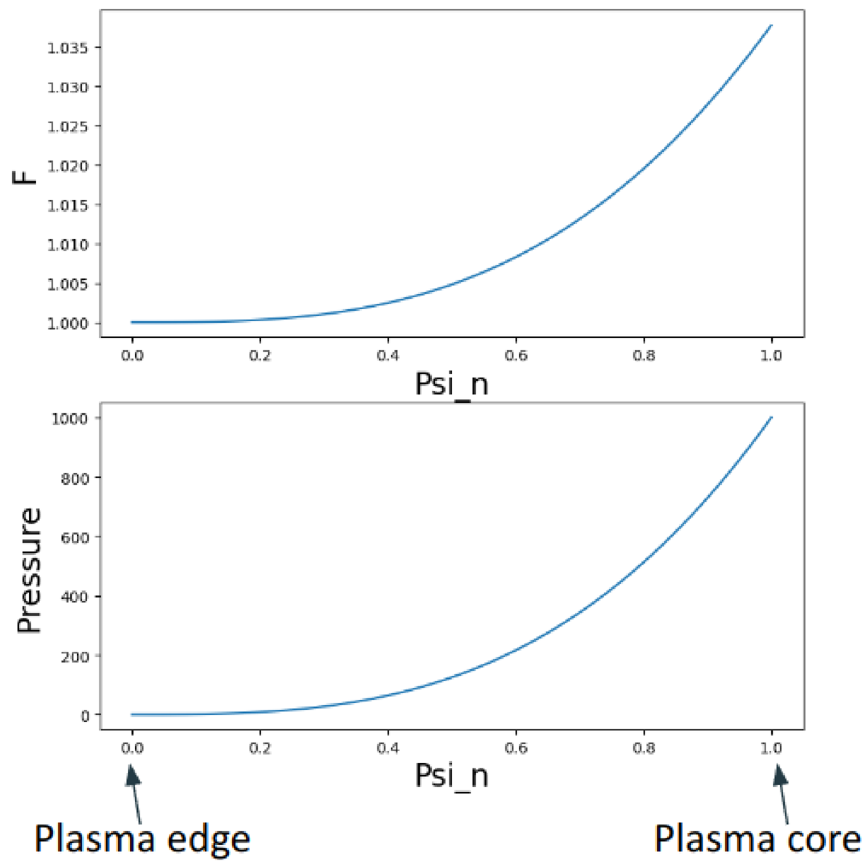


Figure 2: p and F profiles as functions of ψ_n for Example 5 of FreeGS

approximate the quantities in the Grad-Shafranov equation with the following functions of ψ_n:

$$\frac{dp(\psi)}{d\psi} \approx c\frac{dp(\psi_n)}{d\psi_n} \tag{9}$$

$$\frac{d}{d\psi}(F^2(\psi)) \approx 2c\frac{d}{d\psi_n}(F^2(\psi_n)) \tag{10}$$

3 Solving the Problem

The standard functionality of NeuralPDE.jl was used to solve the problem (source code attached). Typically, the loss would decrease to about 10^{-1} but then it would stop decreasing. In almost all solution attempts of the ψ function made, a banded structure such as the one shown in Figure 4 would occur. I tried quite a number of techniques to fix the problem to no avail; they include but are not limited to:

1. Using different learning rate schedules (e.g. 0.05 for 5000 iterations followed by 0.025 for 5000 then 0.01 for 5000, etc.)
2. Using BFGS in addition to Adam
3. Manually checked the underlying model with build_symbolic_loss_function, it looks correct
4. Introducing a convexity metric into the loss function by taking finite differences on ψ to include $\frac{d^2\psi}{dr^2}$ in the loss function
5. Both grid and quadrature training
6. Introducing $\frac{\partial\psi(r,z)}{\partial r}$ and $\frac{\partial\psi(r,z)}{\partial z}$ as additional dependent variables to solve for with boundary conditions relating them to $\psi(r,z)$
7. Using ψ_n as the dependent variable instead thus making the formulation look like $\frac{1}{c}\frac{\partial^2\psi_n}{\partial r^2} - \frac{1}{rc}\frac{\partial\psi_n}{\partial r} + \frac{1}{c}\frac{\partial^2\psi_n}{\partial z^2} = -\mu_0 r^2 c\frac{dp}{d\psi_n}(\psi_n) - 2c\frac{d}{d\psi_n}(F^2)$
8. Using different RNG seeds

4 Summary and Outlook

Solving the Grad-Shafranov equation faster would help enhance the abilities of plasma control systems as current real-time Grad-Shafranov solvers have to make compromises due to the extremely challenging real-time constraints. In this work, I tried exploring using NeuralPDE.jl to solve an example problem, but the solution always converges to a banded-structure that doesn't properly solve the problem. A number of different techniques were tried to no avail. Given that a decision has been made to rewrite NeuralPDE.jl (`https://github.com/SciML/NeuralPDE.jl/issues/687`), it is likely a good idea to wait for the new version before attempting further investigation.

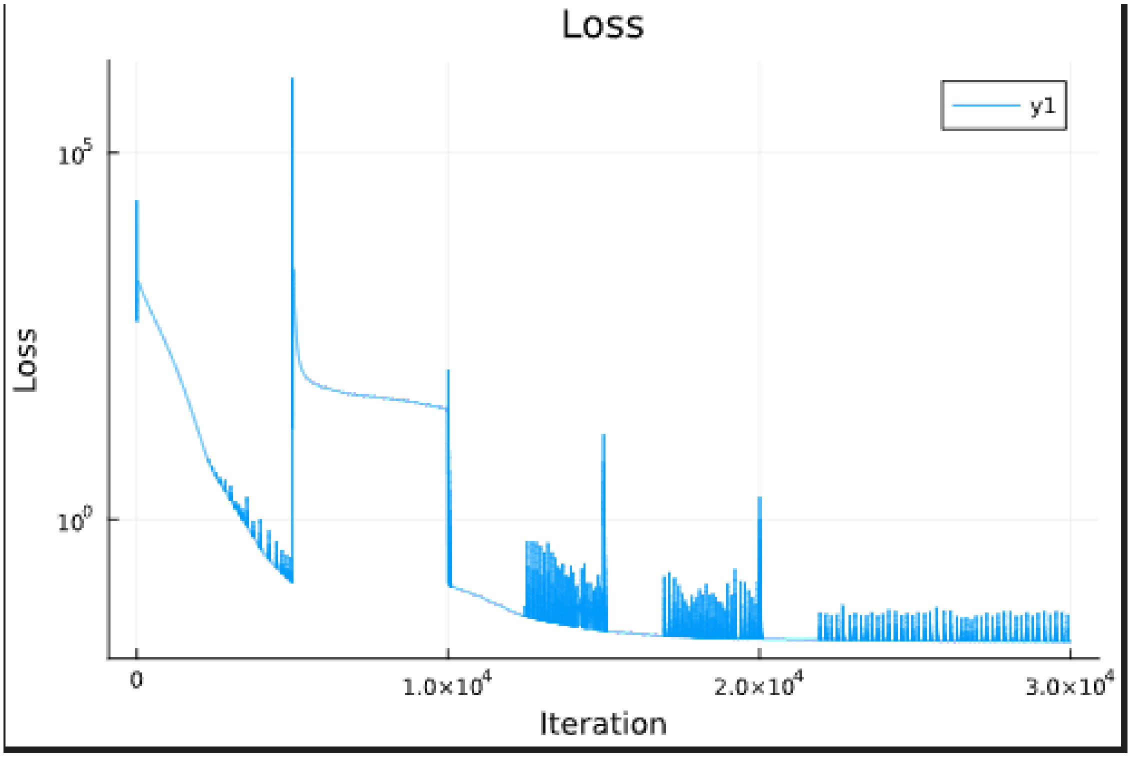


Figure 3: Typical loss curve. The large jump occurred upon restarting the optimization with a new learning rate.

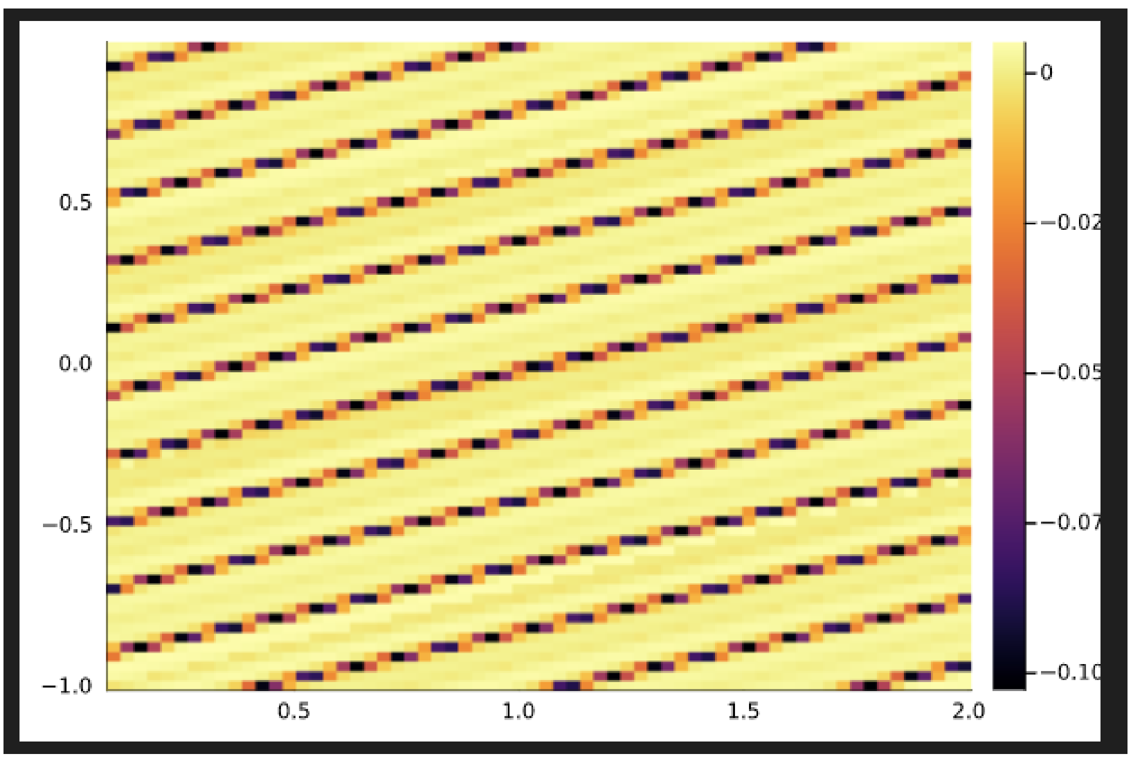


Figure 4: Typical plot of the ψ solution

References

[1] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.

[2] Ben Dudson. freegs. `https://github.com/freegs-plasma/freegs`, 2023.

[3] Harold Grad and Hanan Rubin. Hydromagnetic equilibria and force-free fields. *Journal of Nuclear Energy (1954)*, 7(3-4):284–285, 1958.

[4] Vitaly Dmitrijewitsch Shafranov. Equilibrium of a toroidal plasma in a magnetic field. *Journal of Nuclear Energy. Part C, Plasma Physics, Accelerators, Thermonuclear Research*, 5(4):251, 1963.

[5] JT Wai, MD Boyer, and E Kolemen. Neural net modeling of equilibria in nstx-u. *Nuclear Fusion*, 62(8):086042, 2022.

[6] Kirill Zubov, Zoe McCarthy, Yingbo Ma, Francesco Calisto, Valerio Pagliarino, Simone Azeglio, Luca Bottero, Emmanuel Luján, Valentin Sulzer, Ashutosh Bharambe, et al. Neuralpde: Automating physics-informed neural networks (pinns) with error approximations. *arXiv preprint arXiv:2107.09443*, 2021.

In [ ]:
```
using NeuralPDE, Lux, ModelingToolkit, Optimization, OptimizationOptimisers,
import ModelingToolkit: IntervalDomain, infimum, supremum
```

In [ ]:
```
vars = npzread("profiles.npz")
psis = vars["psis"]
psin = psis/(maximum(psis) - minimum(psis))

pprime_interp = LinearInterpolation(psin, vars["pprime"], extrapolation_bc=L
ffprime_interp = LinearInterpolation(psin, vars["ffprime"], extrapolation_bc


function pprime_interp_f(psi)
    return pprime_interp(psi)
end

function ffprime_interp_f(psi)
    return ffprime_interp(psi)
end
# Without these two lines, a function overload that accepts floats just does
pprime_interp_f(0.0)
ffprime_interp_f(0.5)

# Register the functions.
@register pprime_interp_f(psi)
@register ffprime_interp_f(psi)
```

In [ ]:
```
@parameters r, z, scale
@variables psi(..), Drpsi(..), Dzpsi(..)

Dr = Differential(r)
Drr = Differential(r)^2
Dz = Differential(z)
Dzz = Differential(z)^2
mu0 = 4.0 * pi * 1e-7

eq = Dr(Drpsi(r, z)) - 1.0/r * Drpsi(r, z) + Dz(Dzpsi(r, z)) ~
    -mu0 * r^2 * scale * pprime_interp_f(scale*psi(r, z)) - 2 * scale * ffpr

# Space and time domains
domains = [
    r ∈ IntervalDomain(0.1, 2.0),
    z ∈ IntervalDomain(-1.0, 1.0)
]

dx = 0.03
rs, zs = [infimum(d.domain):(dx):supremum(d.domain) for d in domains]

rs = Vector(rs)
zs = Vector(zs)

function additional_loss(phi, θ, p)
    c = p[1] # Scale factor.
    psi_eval = (r, z) -> phi[1]([r, z], θ[:psi])[1]
```

```
    # Evaluate the maximum and minimum at the z=0 plane.
    psis = psi_eval.(rs, 0.0)
    psi_max = maximum(psis)
    psi_min = minimum(psis)
    psi_diff = psis[2:end] - psis[1:end-1]
    psi_diff2 = psi_diff[2:end] - psi_diff[1:end-1]
    return 100.0 * abs2((1.0/psi_max - c))
end

# Boundary conditions
bcs = [
    psi(0.1, z) ~ 0.0, psi(2.0, z) ~ 0.0,
    psi(r, -1.0) ~ 0.0, psi(r, 1.0) ~ 0.0,
    Dr(psi(r, z)) ~ Drpsi(r, z),
    Dz(psi(z, z)) ~ Dzpsi(r, z),
]
```

$$\psi\left(0.1, z\right) = 0.0 \tag{1}$$
$$\psi\left(2.0, z\right) = 0.0 \tag{2}$$
$$\psi\left(r, -1.0\right) = 0.0 \tag{3}$$
$$\psi\left(r, 1.0\right) = 0.0 \tag{4}$$
$$\frac{\mathrm{d}}{\mathrm{d}r}\psi\left(r, z\right) = \mathrm{Drpsi}\left(r, z\right) \tag{5}$$
$$\frac{\mathrm{d}}{\mathrm{d}z}\psi\left(z, z\right) = \mathrm{Dzpsi}\left(r, z\right) \tag{6}$$

In [ ]:
```
# Neural network
dim = 2 # number of dimensions

vars = [psi(r, z), Drpsi(r, z), Dzpsi(r, z)]
chains = [Lux.Chain(Dense(dim, 16, Lux.σ), Dense(16, 16, Lux.σ), Dense(16, 1

# Seeding
rng = Random.default_rng()
Random.seed!(rng, 420)
for chain in chains
    Lux.setup(rng, chain) .|> gpu
end

discretization = PhysicsInformedNN(chains, QuadratureTraining(), additional_

@named pde_system = PDESystem(eq, bcs, domains, [r, z], vars, [scale], defau
prob = discretize(pde_system, discretization)
```

```
OptimizationProblem. In-place: true
u0: ComponentVector{Float64}(depvar = (psi = (layer_1 = (weight = [-0.5222790
837287903 0.2450590282678604; 0.24435700476169586 0.1965530663728714; … ; -0.
5399954915046692 0.31157130002975464; -0.020784961059689522 0.545043766498565
7], bias = [0.0; 0.0; … ; 0.0; 0.0;;]), layer_2 = (weight = [-0.2224248945713
0432 0.0797654464840889 … 0.3757952153682709 0.15557150542736053; -0.18783475
45862198 -0.31503376364707947 … -0.10993435978889465 -0.13735470175743103; …
; 0.3551483750343323 0.15315645933151245 … -0.28155404329299927 -0.0844416245
8181381; -0.2335842251777649 0.12314730882644653 … 0.06038163974881172 -0.131
67117536067963], bias = [0.0; 0.0; … ; 0.0; 0.0;;]), layer_3 = (weight = [0.3
340683579444885 -0.07777102291584015 … 0.45195043087005615 0.3676756918430328
4], bias = [0.0;;])), Drpsi = (layer_1 = (weight = [-0.054033536463975906 0.4
119073450565338; -0.441193163394928 -0.56633061170578; … ; 0.0294930413365364
07 -0.22794397175312042; 0.028729284182190895 -0.2146928608417511], bias =
[0.0; 0.0; … ; 0.0; 0.0;;]), layer_2 = (weight = [-0.13559329509735107 -0.111
10404878854752 … 0.07901331037282944 0.28638899326324463; -0.3807540535926819
-0.29612699151039124 … 0.40122872591018677 0.23254482448101044; … ; -0.269849
35998916626 -0.27580526471138 … -0.0014545239973813295 0.34738361835479736; -
0.40942221879959106 0.2895776629447937 … 0.403973788022995 -0.033536486327648
16], bias = [0.0; 0.0; … ; 0.0; 0.0;;]), layer_3 = (weight = [0.0520434975624
0845 0.58753901720047 … 0.27488917112350464 0.17206165194511414], bias = [0.
0;;])), Dzpsi = (layer_1 = (weight = [-0.24373255670070648 -0.563088119029998
8; -0.4694974422454834 -0.006693900562822819; … ; 0.04678152874112129 -0.2495
0501322746277; 0.16709747910499573 0.022105859592556953], bias = [0.0; 0.0; …
; 0.0; 0.0;;]), layer_2 = (weight = [0.10484801232814789 0.382907509803772 …
0.0547507144510746 -0.2697623670101166; 0.20850497484207153 -0.32748779654502
87 … -0.11751493811607361 0.19711211323738098; … ; 0.40632104873657227 0.1997
4587857723236 … 0.37682831287384033 0.024786941707134247; -0.2799744904041290
3 -0.4199541509151459 … -0.3284936547279358 0.2347298115491867], bias = [0.0;
0.0; … ; 0.0; 0.0;;]), layer_3 = (weight = [0.4409675896167755 0.010308898985
385895 … 0.25882217288017273 0.5142514109611511], bias = [0.0;;]))), p = [20.
0])
```

In [ ]:

```
losses = []

callback = function (p, l)
    append!(losses, l)
    if length(losses) % 100 == 0
        println("Current loss is: $l")
    end
    return false
end

res = Optimization.solve(prob, ADAM(0.05); callback = callback, maxiters = 5
prob = remake(prob, u0 = res.minimizer)
res = Optimization.solve(prob, ADAM(0.01); callback = callback, maxiters = 5
prob = remake(prob, u0 = res.minimizer)
res = Optimization.solve(prob, ADAM(0.005); callback = callback, maxiters =
prob = remake(prob, u0 = res.minimizer)
res = Optimization.solve(prob, ADAM(0.0025); callback = callback, maxiters =
prob = remake(prob, u0 = res.minimizer)
res = Optimization.solve(prob, ADAM(0.001); callback = callback, maxiters =
```

```
Current loss is: 1567.2871062299068
Current loss is: 1217.7537544693432
Current loss is: 971.8251341341283
Current loss is: 786.3138271898505
Current loss is: 635.9554596827132
Current loss is: 513.4599873308053
Current loss is: 415.6688275240066
Current loss is: 336.8634841365945
Current loss is: 272.2206355922599
Current loss is: 218.50839356196158
Current loss is: 173.80053955981552
Current loss is: 137.19241658151233
Current loss is: 107.56291184367564
Current loss is: 83.67378766225205
Current loss is: 64.10980552536346
Current loss is: 48.68108284964782
Current loss is: 36.65094443857832
Current loss is: 27.3208685724142
Current loss is: 20.192673361611085
Current loss is: 14.852184231093775
Current loss is: 10.955650919401425
Current loss is: 8.220537393353188
Current loss is: 6.318980645012798
Current loss is: 5.042626171485077
Current loss is: 4.196902899178233
Current loss is: 3.574120663051564
Current loss is: 3.1161841463789366
Current loss is: 2.5958471952084365
Current loss is: 2.2300623616365454
Current loss is: 1.9089600305195982
Current loss is: 1.6299247543280144
Current loss is: 1.38167938787669
Current loss is: 1.177167633622788
Current loss is: 1.0445700003521239
Current loss is: 0.8540514018276235
Current loss is: 0.7392336676575669
Current loss is: 0.6311855634546056
Current loss is: 0.5408887304032589
Current loss is: 0.4535290218758875
Current loss is: 0.42055048397092437
Current loss is: 0.33862438282111046
Current loss is: 0.30061087824358984
Current loss is: 0.27227765798681214
Current loss is: 0.24231701314294898
Current loss is: 0.2674130232112856
Current loss is: 0.19832879720728158
Current loss is: 0.186027444399698
Current loss is: 0.16480732643793644
Current loss is: 0.1502555524462741
Current loss is: 0.13810803114975262
Current loss is: 477.2731090278379
Current loss is: 167.20991048818695
Current loss is: 111.85772031665297
Current loss is: 92.37734871889991
Current loss is: 81.97089084590954
Current loss is: 75.68633405496972
```

```
Current loss is: 71.12604643850773
Current loss is: 67.82807185200319
Current loss is: 65.26088742868814
Current loss is: 63.182966001753115
Current loss is: 61.381089289397636
Current loss is: 59.748929886765346
Current loss is: 58.24480199813138
Current loss is: 56.84042336904375
Current loss is: 55.5053164272759
Current loss is: 54.26637919067126
Current loss is: 53.14280758795986
Current loss is: 52.14041882336386
Current loss is: 51.266313810735
Current loss is: 50.52649801060693
Current loss is: 49.88645673452924
Current loss is: 49.30582451432822
Current loss is: 48.77072993178967
Current loss is: 48.2652093844317
Current loss is: 47.76505230044961
Current loss is: 47.18350284455744
Current loss is: 46.65714963603809
Current loss is: 46.128175508455044
Current loss is: 45.58785574207938
Current loss is: 45.0472802697882
Current loss is: 44.49056557760164
Current loss is: 44.15630682246362
Current loss is: 43.35079830649707
Current loss is: 42.970670677324094
Current loss is: 42.16252727426117
Current loss is: 41.54557750393662
Current loss is: 40.914267812600706
Current loss is: 40.31835696123989
Current loss is: 39.60312362675139
Current loss is: 38.92523781376152
Current loss is: 38.231471945604454
Current loss is: 37.53330016514753
Current loss is: 36.80201494521708
Current loss is: 36.0852734564448
Current loss is: 35.32273396090085
Current loss is: 34.56867594280142
Current loss is: 33.80593305465536
Current loss is: 33.035081423935274
Current loss is: 32.26024182572252
Current loss is: 31.48115427727527
Current loss is: 0.12344239395966512
Current loss is: 0.11853315013803524
Current loss is: 0.11562243651685815
Current loss is: 0.112245967661906
Current loss is: 0.10856594212930225
Current loss is: 0.10474664097743025
Current loss is: 0.10085803487997487
Current loss is: 0.09697005416076802
Current loss is: 0.09312689701404687
Current loss is: 0.08933615305509238
Current loss is: 0.08565200352746986
Current loss is: 0.08205959112454897
```

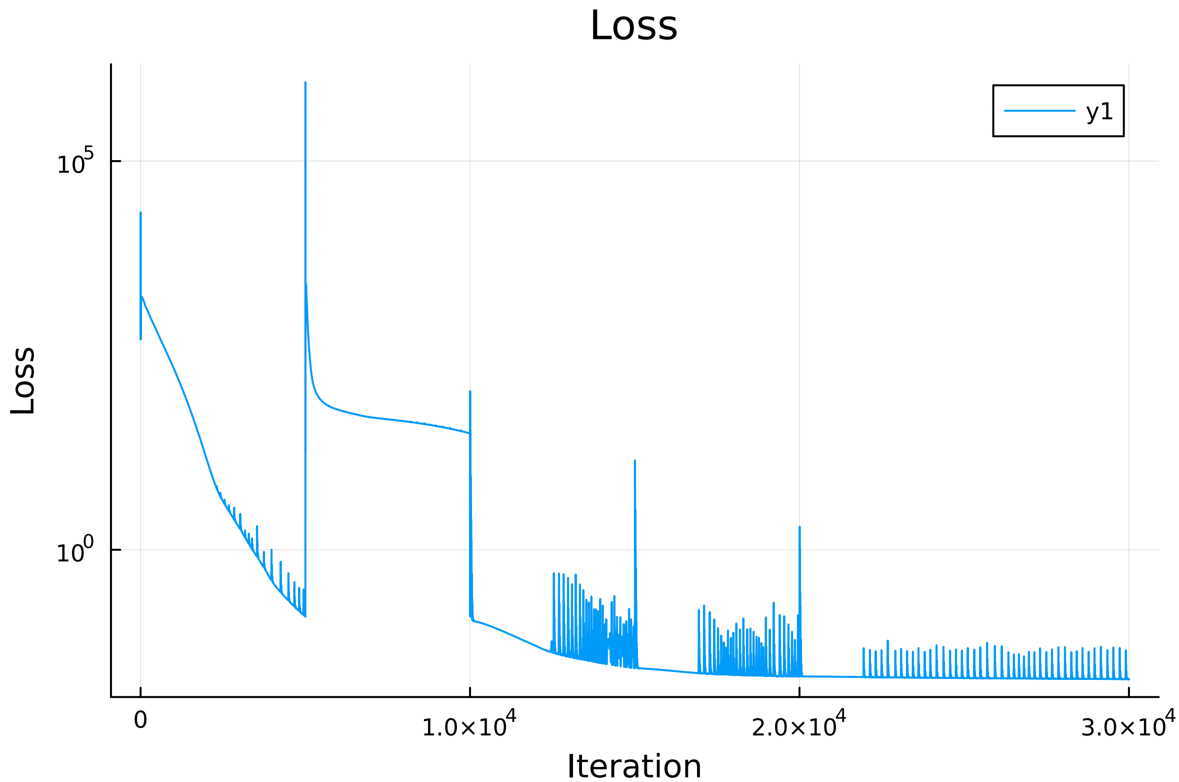
```
Current loss is: 0.0785832636248068
Current loss is: 0.07524366474529157
Current loss is: 0.07203246812293289
Current loss is: 0.06895690557314613
Current loss is: 0.06602287800122496
Current loss is: 0.06323420605968931
Current loss is: 0.06053728432950244
Current loss is: 0.05789007564787326
Current loss is: 0.055355824549804346
Current loss is: 0.0530440716341061
Current loss is: 0.05100279263883181
Current loss is: 0.049245241009386
Current loss is: 0.04783382153268712
Current loss is: 0.047601524273933556
Current loss is: 0.4958453997482113
Current loss is: 0.04408049788406373
Current loss is: 0.042975424014028575
Current loss is: 0.044308194534525634
Current loss is: 0.055555718021705974
Current loss is: 0.0747777559516299
Current loss is: 0.03962410350877459
Current loss is: 0.03952061770940227
Current loss is: 0.03988610817490543
Current loss is: 0.1123614247905583
Current loss is: 0.06124060036386572
Current loss is: 0.03837737279626972
Current loss is: 0.05123516558576773
Current loss is: 0.035794374624157674
Current loss is: 0.06432806674283648
Current loss is: 0.06159850251891897
Current loss is: 0.21303184851413276
Current loss is: 0.040033302647875225
Current loss is: 0.05129804056381746
Current loss is: 0.05050253495441753
Current loss is: 0.10035847430801714
Current loss is: 0.03562957207969218
Current loss is: 0.030898888945713868
Current loss is: 0.09025010975066136
Current loss is: 0.030169472469077533
Current loss is: 0.029776851660985974
Current loss is: 0.029598729088532564
Current loss is: 0.02940243240230665
Current loss is: 0.029190270000412893
Current loss is: 0.028965205860287085
Current loss is: 0.028731641569076684
Current loss is: 0.028491249837372484
Current loss is: 0.028246495841957864
Current loss is: 0.02799968978616812
Current loss is: 0.027753974189965343
Current loss is: 0.027509717036843646
Current loss is: 0.027268173897938723
Current loss is: 0.027030476405980514
Current loss is: 0.02679657263110278
Current loss is: 0.02656667095999005
Current loss is: 0.026342653938658544
Current loss is: 0.026126387249730182
```

```
Current loss is: 0.025915773927217958
Current loss is: 0.02646679015481902
Current loss is: 0.0581570395486786
Current loss is: 0.02544733601723343
Current loss is: 0.03264276517416379
Current loss is: 0.027315604719588823
Current loss is: 0.02518231058737364
Current loss is: 0.025166252599094614
Current loss is: 0.06374095801778032
Current loss is: 0.025327883929884824
Current loss is: 0.026446497654430665
Current loss is: 0.02589331537657051
Current loss is: 0.02539205491304284
Current loss is: 0.024446483358615937
Current loss is: 0.04963931180096867
Current loss is: 0.02648711538561362
Current loss is: 0.029560606089088254
Current loss is: 0.0436112909871682
Current loss is: 0.04098471552405097
Current loss is: 0.02710552910380637
Current loss is: 0.04234607072795166
Current loss is: 0.03437631664917621
Current loss is: 0.08651639529909869
Current loss is: 0.02387431708104958
Current loss is: 0.023722843974728215
Current loss is: 0.1450112815534097
Current loss is: 0.02363539283954744
Current loss is: 0.023628587593943533
Current loss is: 0.02538529156942475
Current loss is: 0.029577774570354015
Current loss is: 0.02546342671674828
Current loss is: 0.02427508117438649
Current loss is: 0.023536980346281593
Current loss is: 0.023445934646816418
Current loss is: 0.023431718470437013
Current loss is: 0.02341690222594708
Current loss is: 0.02340037041137554
Current loss is: 0.023382129385148907
Current loss is: 0.023362588234505148
Current loss is: 0.02334214649586923
Current loss is: 0.02332002648144195
Current loss is: 0.023297297882105623
Current loss is: 0.023273359846577107
Current loss is: 0.023248429657171794
Current loss is: 0.02322282762422
Current loss is: 0.023194722984578767
Current loss is: 0.023166911953231926
Current loss is: 0.023138010546541948
Current loss is: 0.02310828400543892
Current loss is: 0.023077090897505093
Current loss is: 0.023045667482288066
Current loss is: 0.023206886003762462
Current loss is: 0.022987673638717336
Current loss is: 0.02299085365200115
Current loss is: 0.023107252833889813
Current loss is: 0.02290813652815811
```

```
Current loss is: 0.032672311435033685
Current loss is: 0.02285556130673597
Current loss is: 0.02305499828964112
Current loss is: 0.022809942905112554
Current loss is: 0.023683671406962235
Current loss is: 0.022760357324779806
Current loss is: 0.02798607060043494
Current loss is: 0.022714510981985778
Current loss is: 0.022723862957439316
Current loss is: 0.02267252158531061
Current loss is: 0.022655819299035363
Current loss is: 0.023180515986667833
Current loss is: 0.022615738915579595
Current loss is: 0.045024627093115256
Current loss is: 0.022564582621796947
Current loss is: 0.023809693827513245
Current loss is: 0.022526513798241384
Current loss is: 0.022865058859969943
Current loss is: 0.022487136774049756
Current loss is: 0.023405540269622663
Current loss is: 0.02245033809849987
Current loss is: 0.023424966913418808
Current loss is: 0.022411996433323166
Current loss is: 0.022497340519671456
Current loss is: 0.0223750832706941
Current loss is: 0.022384630547681066
Current loss is: 0.024395846693324178
Current loss is: 0.022331828330905763
Current loss is: 0.02782205938891775
Current loss is: 0.022295131567034655
Current loss is: 0.025413879170308292
Current loss is: 0.02225759304916228
Current loss is: 0.02523179080236041
Current loss is: 0.022229579531831717
Current loss is: 0.022210153758341514
Current loss is: 0.02219917582790636
Current loss is: 0.022178288766957398
Current loss is: 0.022212072281368095
Current loss is: 0.02214808605474029
Current loss is: 0.022132398726998882
Current loss is: 0.024483994045063
Current loss is: 0.02211244196143023
Current loss is: 0.022433424429411805
Current loss is: 0.022498087272275112
Current loss is: 0.022057130616408134
Current loss is: 0.022386666809776766
Current loss is: 0.02202918539426679
Current loss is: 0.022054885775950987
Current loss is: 0.035668123388169416
Current loss is: 0.021995146444960527
Current loss is: 0.03180832247900964
Current loss is: 0.021960539056283125
Current loss is: 0.022651617282126762
Current loss is: 0.021936464626198107
Current loss is: 0.022345134969981577
Current loss is: 0.02191046486999173
```

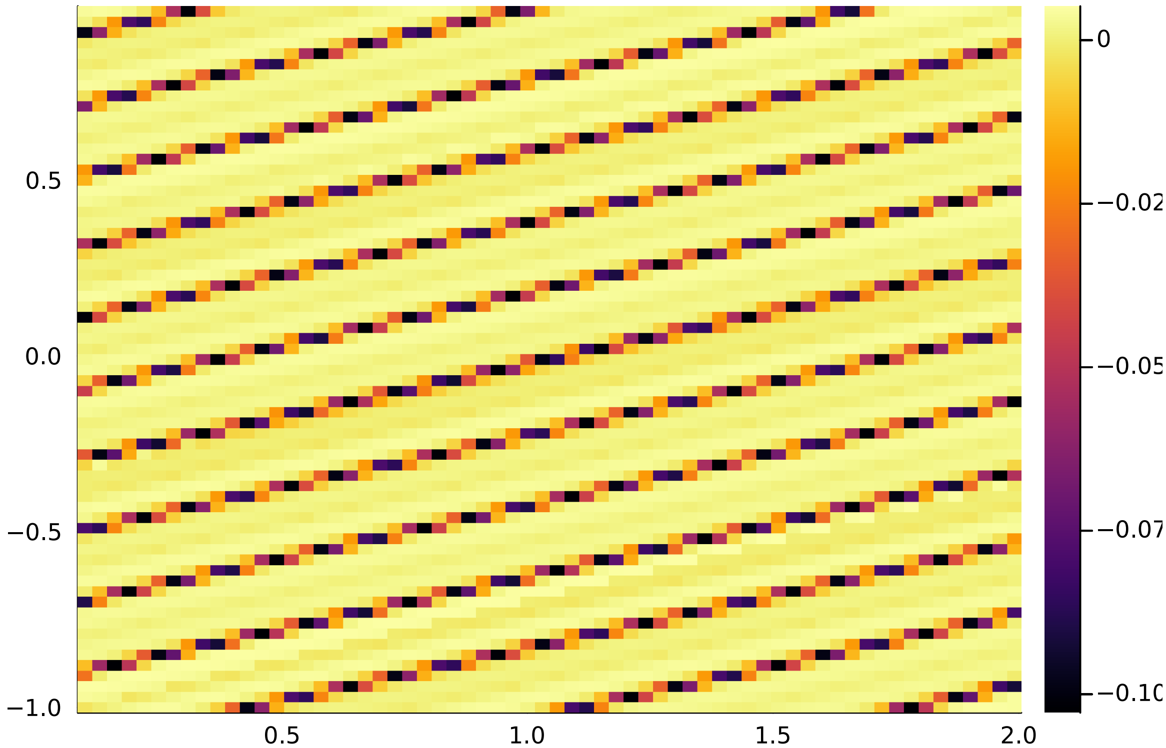
```
Current loss is: 0.0222789161188733
Current loss is: 0.021883969278565042
Current loss is: 0.021885103249248083
Current loss is: 0.021998532641358804
Current loss is: 0.021851261130678624
Current loss is: 0.032413200113537
Current loss is: 0.02182023575360416
Current loss is: 0.02394182307176615
Current loss is: 0.021795620552546274
Current loss is: 0.021799011928735172
Current loss is: 0.021770625257576427
Current loss is: 0.021863645637438197
Current loss is: 0.021746483966655822
Current loss is: 0.021750212677520846
Current loss is: 0.02172902856879842
Current loss is: 0.02173596182654484
Current loss is: 0.02172951799389933
Current loss is: 0.021688038827991887
Current loss is: 0.026741131057687745
Current loss is: 0.02166380540130919
u: ComponentVector{Float64}(depvar = (psi = (layer_1 = (weight = [-0.66923063
71595532 18.425410519170093; -0.1505369418382242 10.636163368045624; … ; -0.5
86136977955502 -23.59086974144638; -0.3662618904943838 -1.7641232836621157],
bias = [-0.09419355523421064; -0.15479595791973194; … ; -0.5807254249726345;
-0.8709768432798962;;]), layer_2 = (weight = [-0.5267117362365629 -0.14200824
502927392 … 0.4688551513618925 0.0009987090118028221; -1.3039552249468083 -0.
5299225558254037 … -0.6654225027223537 -0.4531270276039166; … ; 0.26884585130
121913 -0.04215633633587162 … -0.49848065931072516 -0.2686186907861078; -0.48
19546466307932 -0.05307944383447722 … 0.2153093393831614 -0.259214861063533
2], bias = [-0.15508410573880693; -0.7452803805176187; … ; -0.176674999963461
28; -0.11935418993473192;;]), layer_3 = (weight = [0.25234917616251584 -0.131
72307123914112 … 0.31210403676559034 0.2888075830376271], bias = [0.120955609
96395993;;])), Drpsi = (layer_1 = (weight = [-3.6168901635020156 8.4159801629
64662; 2.715488442973611 -9.267227788931022; … ; 2.9294238002406097 -2.753659
290095273; -0.16062441728819207 -6.250794692687454], bias = [0.86653820072721
91; 0.048129350946179296; … ; -1.0170044042922903; -0.22446265647946276;;]),
layer_2 = (weight = [-0.8061322582371894 -0.5849637646164765 … -0.54156242940
65029 -0.2721648024609249; -1.0355358607698848 -0.7070570573888194 … 0.094262
66567017963 0.8193864820421471; … ; -0.9819066901329443 -1.1419297846733103 …
-0.6359790201531179 0.9789629465632668; -0.9807419889600515 -0.19458286274863
64 … -0.06875234588772616 0.1615392066296994], bias = [-0.5825989704498492; -
0.19388345753405944; … ; -0.09150233396270939; -0.33963849164202264;;]), laye
r_3 = (weight = [0.08858100352038073 0.857961174371266 … 0.7715847272616732
0.235416235356368], bias = [0.08037131936706753;;])), Dzpsi = (layer_1 = (wei
ght = [-1.5147455944389892 -0.9721012504428336; -7.675801695482528 1.18219953
57165604; … ; 0.426627740068026 -1.7465701766752733; 1.346243295705222 -0.031
31400832701576], bias = [-0.843892418042364; -3.6060851209522338; … ; -0.7334
325045083282; 0.4588722659278727;;]), layer_2 = (weight = [-0.497510228940908
8 0.23837005287463442 … -0.7454879618445904 -1.1810210540753716; 1.8347052396
07666 -5.654690688226655 … 0.7371658155796983 0.7892749109900493; … ; -0.6406
427442465615 -1.8747289079264533 … 0.939462034107082 0.8368721176422634; -1.5
524493055412232 -1.0954504785697987 … -1.4708869212493991 0.01068866485382321
6], bias = [-0.7532080240770771; -1.4633917615384286; … ; 0.0177451719662897
8; 0.31534271328774127;;]), layer_3 = (weight = [0.7555879712439234 -1.510675
6183913719 … 0.9511512454855499 0.760118670228731], bias = [0.027574014937335
88;;]))), p = [-9.691406701475902])
```

In [ ]:
```
plot(losses, title="Loss", xlabel="Iteration", ylabel="Loss", yscale=:log10)
```



In [ ]:
```
phi = discretization.phi
scale_res = res.u[end]

psi_predict = reshape([phi[1]([r, z], res.u.depvar[:psi])[1] for r in rs for
psi_min = minimum(psi_predict)
psi_max = maximum(psi_predict)
println("Scale factor is: $scale_res")
println("Psi min is: $psi_min")
println("Psi max is: $psi_max")
heatmap(rs, zs, psi_predict)
```

```
Scale factor is: -9.691406701475902
Psi min is: -0.1028266564892037
Psi max is: 0.005196024798997223
```



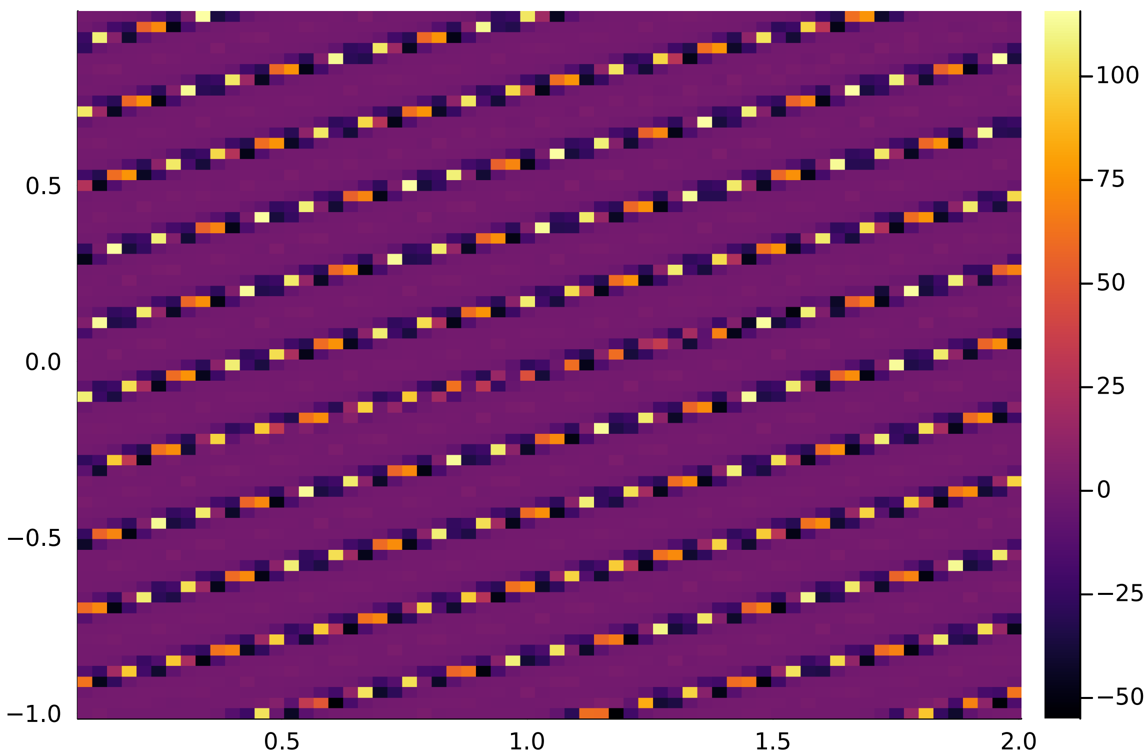
In [ ]:

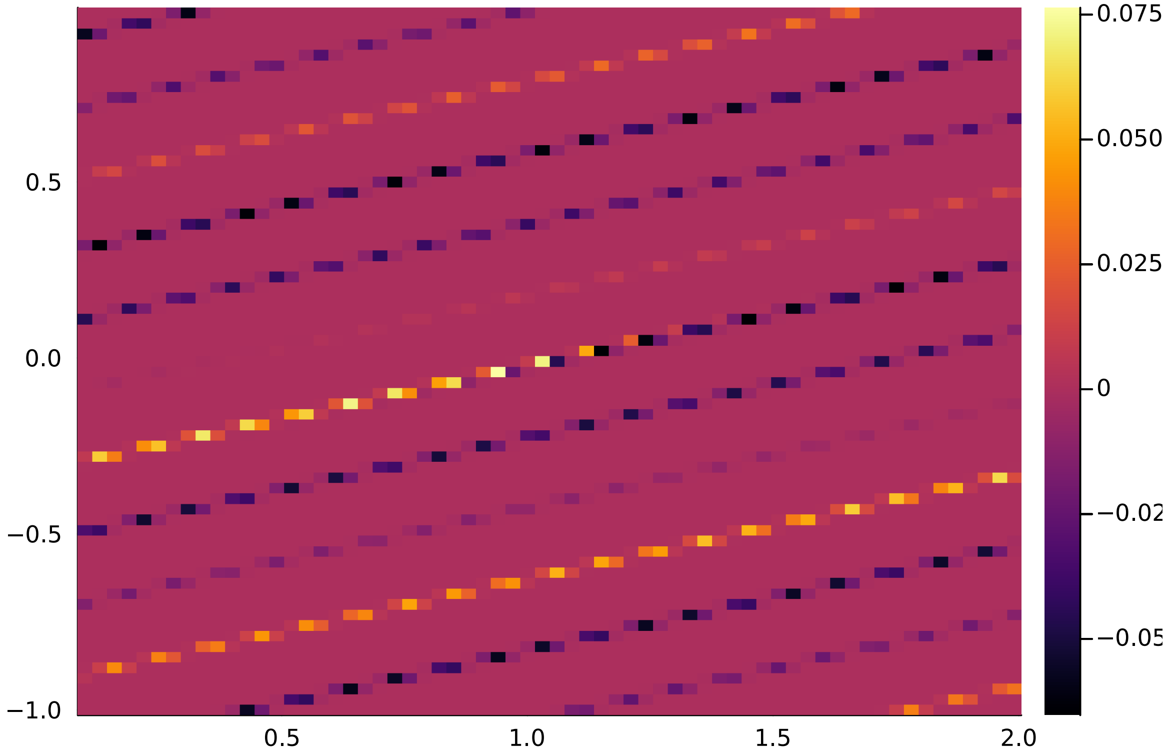
```
# Calculate the RHS.
p_term = reshape([-mu0 * r^2 * scale_res * pprime_interp_f(scale_res*phi[1](
ff_term = reshape([-0.58814188 * ffprime_interp_f(scale_res*phi[1]([r, z], r
rhs = p_term + ff_term

# Calculate the LHS
dz = (1.0/dx).*(psi_predict[2:end, :] .- psi_predict[1:end-1, :])
dr = (1.0/dx).*(psi_predict[:, 2:end] .- psi_predict[:, 1:end-1])


dz = vcat(zeros(1, size(dz, 2)), dz)
dr = hcat(zeros(size(dr, 1), 1), dr)


dzz = (1.0/dx).*(dz[2:end, :] .- dz[1:end-1, :])
drr = (1.0/dx).*(dr[:, 2:end] .- dr[:, 1:end-1])

dzz = vcat(zeros(1, size(dzz, 2)), dzz)
drr = hcat(zeros(size(drr, 1), 1), drr)

lhs = dzz .+ drr .- 1.0./rs .* dr
```

```
64×67 Matrix{Float64}:
  0.0          1.5041     -0.902782   …  -0.0332397   0.0394222   0.108373
  0.281751     0.402389   -0.418053       0.13112     0.24915     0.355347
 -0.00776092   0.331438    1.07975       -0.153555   -0.0932404  -0.0380712
 -0.00921045  -0.955254    2.59983       -0.189654   -0.127641   -0.0775582
 -0.0107754    0.565665    1.87555       -0.228537   -0.1552     -0.10828
 -0.0124629   -0.616758    1.10674    …  -0.28133    -0.179713   -0.133191
 -0.0142812   -0.591377   -0.421881      -0.366663   -0.205089   -0.154196
 -0.0162405   -0.558567    1.20661       -0.519368   -0.236941   -0.172912
 -0.0183533   -0.518467   -0.0905553     -0.807374   -0.284929   -0.191246
 -0.0206364   -0.470994   -0.118425      -1.36527    -0.367057   -0.212103
  ⋮                                   ⋱               ⋮
 -0.0639672    0.6268     -0.94748    …  -0.270207    1.77498    -0.0241417
 -0.0549472    0.834342   -0.834496      -0.863718    2.34919    -0.0350548
 -0.0469668    1.0144     -0.734414      -1.4526      2.91705    -0.0453139
 -0.039926     1.17007    -0.645403       1.85237    -0.411223    1.90613
 -0.0337316    1.30414    -0.566191      -1.50399    -1.001       2.47392
 -0.0282968    1.41913    -0.495776   …  -0.13697    -1.58569     3.03534
 -0.0235396    1.5173     -0.433294      -0.148214    1.97883    -0.554342
 -0.0193832    1.60066    -0.377962      -0.159025   -1.63827    -1.14036
 -0.0157553    1.67102    -0.329047      -0.169302   -0.138298   -1.72083
```

In [ ]:

```
display(heatmap(rs, zs, lhs))
display(heatmap(rs, zs, rhs))
```




0.5
0.0
−0.5
−1.0
0.5
1.0
1.5
2.0
0.075
0.050
0.025
0
−0.02
−0.05