

IMPORTS

In []: *#importing relevant libraries*

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit      #for fitting an exponential
import functools as fnt
from textwrap import wrap
```

CONSTANTS

In []: *#relevant constants*

```
pc = 3.086e18                      #cm
pi = np.pi

#galaxy specific constants (taken from SS21_ch11)
R = 10*(10**3)*pc                  #radius in cm
omega = 6.481*(10**(-18))          #angular velocity in s^-1
h = 0.5*(10**3)*pc                 #height in cm
eta_T = 10**26                      #Diffusion coefficient in cm^2 s^-1
to = h*h/eta_T                      #diffusion time in s
print('time stepping is normalised to to(s) = ', to)
```

time stepping is normalised to to(s) = 2.380849e+16

DIFFERENTIAL EQUATIONS

In []: *def Br_diff(t, Br, Bp, r_ind, dz, alpha):* #for b

```
dBr_dt = np.array(-1*(0)*(-3*Bp[0] + 4*Bp[1] - Bp[2])/(2*dz) + (2*Br[0]*alpha[r_ind]*Bp[1] - 2*Br[0]*alpha[r_ind]*Bp[2]))/(2*dz)
dBr_dt = np.append(dBr_dt, -1*alpha[r_ind]*np.array(Bp[2:] - Bp[:-2]))
dBr_dt = np.append(dBr_dt, -1*alpha[r_ind]*(3*Bp[-1] - 4*Bp[-2] + Bp[-3]))
return dBr_dt
```

def Bp_diff(t, Br, Bp, r_ind, dz, alpha, S, D, Ralpha, ao = True):

```
if ao:                                #for alpha-omega dynamo
    dBp_dt = np.array( D*S[r_ind]*Br[0] + (2*Bp[0] - 5*Bp[1] + 4*Bp[2])*alpha[r_ind] )
    dBp_dt = np.append(dBp_dt, D*S[r_ind]*np.array(Br[1:-1]) + np.array(D*S[r_ind]*Br[1:-1] - 2*Bp[1] + 5*Bp[2]))
    dBp_dt = np.append(dBp_dt, D*S[r_ind]*Br[-1] + (2*Bp[-1] - 5*Bp[-2])*alpha[r_ind])
else:                                   #for alpha-square omega dynamo
    dBp_dt = np.array( D*S[r_ind]*Br[0] + (Ralpha**2)*alpha[r_ind]*(-Bp[0] + Bp[1]) )
    dBp_dt = np.append(dBp_dt, D*S[r_ind]*np.array(Br[1:-1]) + (Ralpha**2)*alpha[r_ind]*(Bp[1] - Bp[2]))
    dBp_dt = np.append(dBp_dt, D*S[r_ind]*Br[-1] + (Ralpha**2)*alpha[r_ind]*(-Bp[-1] + Bp[-2]))
```

return dBp_dt

RUNGE KUTTA (4TH ORDER)

For Dirichlet Boundary Conditions

These boundary conditions specify that the time evolution of the boundary values is fixed to some number.

For Neumann Boundary Conditions

These boundary conditions specify that the spatial derivatives of the function values are some number at the boundary.

This Runge-Kutta code solves a general system of differential equations in two variables(since in our case we have Br and Bphi)

```
In [ ]: def rk4(F, G, X, Y, BX0, BXn, BY0, BYn, t, dt, dz, bc = 'dir'):
    #F is the function for the time evolution of X, in our case Br
    #G is the function for the time evolution of Y, in our case Bphi
    #X is the array of X values
    #Y is the array of Y values
    #t is the current time
    #dt is the time step
    #dz is the spatial step
    #bc is the boundary condition type
    #BX0, BY0, BXn, BYn are the boundary values

    #We know the time evolution of the borders
    #So we only need to solve for the interior

    #let len(X) = len(Y) = n
    #Assigning the first boundary value
    if bc == 'dir':
        X_new = np.array([BX0])
        Y_new = np.array([BY0])
    elif bc == 'neu':
        X_new = np.array([4*X[1]/3 - X[2]/3 - 2*dz*BX0/3])
        Y_new = np.array([4*Y[1]/3 - Y[2]/3 - 2*dz*BY0/3])

    #Solving the runge kutta coefficients
    k1 = F(t, X, Y)                                #len = n
    l1 = G(t, X, Y)                                #len = n

    k2 = F(t + dt/2, X + k1*dt/2, Y + l1*dt/2)      #len = n
    l2 = G(t + dt/2, X + k1*dt/2, Y + l1*dt/2)      #len = n

    k3 = F(t + dt/2, X + k2*dt/2, Y + l2*dt/2)      #len = n
    l3 = G(t + dt/2, X + k2*dt/2, Y + l2*dt/2)      #len = n

    k4 = F(t + dt, X + k3*dt, Y + l3*dt)            #len = n
    l4 = G(t + dt, X + k3*dt, Y + l3*dt)            #len = n

    #Assigning the interior values
    X_new = np.append(X_new, X[1:-1] + np.array(k1 + 2*k2 + 2*k3 + k4)[1:])
    Y_new = np.append(Y_new, Y[1:-1] + np.array(l1 + 2*l2 + 2*l3 + l4)[1:])

    #Assigning the last boundary value
    if bc == 'dir':
        X_new = np.append(X_new, BXn)
```

```

        Y_new = np.append(Y_new, BYn)
    elif bc == 'neu':
        X_new = np.append(X_new, 4*X[-2]/3 - X[-3]/3 + 2*dz*BXn/3)
        Y_new = np.append(Y_new, 4*Y[-2]/3 - Y[-3]/3 + 2*dz*BYn/3)

    return X_new, Y_new

```

In []: *#defining the grid and timestepping*

```

#grid parameters
z_res = 10**2                      #resolution
r_res = 10**1                        #resolution
z0 = -1                             #lower z/h limit
zn = 1                              #upper z/h limit
r0 = 0.001                          #lower r/R limit
rn = 1                              #upper r/R limit
dz = (zn-z0)/z_res                  #step size
dr = (rn-r0)/r_res                  #step size
z = np.linspace(z0, zn, z_res)      #normalised to scale height h
r = np.linspace(r0, rn, r_res)      #normalised to radius R

#time parameters
dt = 0.000199
t0 = 0

print('dt/to =', dt, 'and dz =', dz)
print('The solution is stable when 2*dt/(dz**2) =', 2*dt/(dz**2), '< 1')

```

dt/to = 0.000199 and dz = 0.02

The solution is stable when $2*dt/(dz^2) = 0.995 < 1$

In []: *#constants and other known variables (0 for the diffusion equation)*

```

l0 = 0.1*(10**2)*pc                #in cm
V0 = 2*10**7                       #in cm/s
omega0 = V0/R                       #in s^-1
alpha0 = (l0**2)*omega0/h          #normalised to  $l0^2 \omega_0/h$ 
S0 = -omega0                         #normalised to 1/r

alpha = np.repeat(1, r_res)
S = 1/r
D1 = -1
Ralpha = 1
r_ind = np.where(r >= 0.2)[0][0]   #Some dummy value, not used in al
                                    #Pick r where r/R around 0.2

```

#Initial seed fields

```

Br1 = np.cos(pi*z/2)
Bp1 = np.cos(pi*z/2)
Br2 = (np.cos(pi*z/2))**2 + (np.cos(3*pi*z/2))**2
Bp2 = (np.cos(pi*z/2))**2 + (np.cos(3*pi*z/2))**2
Br3 = np.sin(pi*z)
Bp3 = np.sin(pi*z)

```

#Dirichlet boundary conditions

```

Br_bc = [[0,0],[0,0],[0,0]]
Bp_bc = [[0,0],[0,0],[0,0]]

```

#Removing function dependence on known values by specifying them

```

Br_diff_eq = fnt.partial(Br_diff, r_ind = r_ind, dz = dz, alpha = alpha)

```

```
# Bp_diff_eq = []
D_array = [-1, -20, -14, -12, -10, -8, -6]
# for i in D_array:
#     Bp_diff_eq.append(fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha))
Bp_diff_eq1 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha)
Bp_diff_eq2 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha)
Bp_diff_eq3 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha)
Bp_diff_eq4 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha)
Bp_diff_eq5 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha)
Bp_diff_eq6 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha)
Bp_diff_eq7 = fnt.partial(Bp_diff, r_ind = r_ind, dz = dz, alpha = alpha),
```

In []: *#initialising the solutions*

```
#for D = -1
Br1_sol1 = np.array([Br1])
Bp1_sol1 = np.array([Bp1])

Br2_sol1 = np.array([Br2])
Bp2_sol1 = np.array([Bp2])

Br3_sol1 = np.array([Br3])
Bp3_sol1 = np.array([Bp3])

#for D = -20
Br1_sol2 = np.array([Br1])
Bp1_sol2 = np.array([Bp1])

Br2_sol2 = np.array([Br2])
Bp2_sol2 = np.array([Bp2])

Br3_sol2 = np.array([Br3])
Bp3_sol2 = np.array([Bp3])

#for D = -14
Br1_sol3 = np.array([Br1])
Bp1_sol3 = np.array([Bp1])

Br2_sol3 = np.array([Br2])
Bp2_sol3 = np.array([Bp2])

Br3_sol3 = np.array([Br3])
Bp3_sol3 = np.array([Bp3])

#for D = -12
Br1_sol4 = np.array([Br1])
Bp1_sol4 = np.array([Bp1])

Br2_sol4 = np.array([Br2])
Bp2_sol4 = np.array([Bp2])

Br3_sol4 = np.array([Br3])
Bp3_sol4 = np.array([Bp3])

#for D = -10
Br1_sol5 = np.array([Br1])
Bp1_sol5 = np.array([Bp1])

Br2_sol5 = np.array([Br2])
Bp2_sol5 = np.array([Bp2])
```

```

Br3_sol5 = np.array([Br3])
Bp3_sol5 = np.array([Bp3])

#for D = -8
Br1_sol6 = np.array([Br1])
Bp1_sol6 = np.array([Bp1])

Br2_sol6 = np.array([Br2])
Bp2_sol6 = np.array([Bp2])

Br3_sol6 = np.array([Br3])
Bp3_sol6 = np.array([Bp3])

#for D = -6
Br1_sol7 = np.array([Br1])
Bp1_sol7 = np.array([Bp1])

Br2_sol7 = np.array([Br2])
Bp2_sol7 = np.array([Bp2])

Br3_sol7 = np.array([Br3])
Bp3_sol7 = np.array([Bp3])

```

In []: *#initialising the time array*

```

t = np.array([t0])

steps= 3*10**4
for i in range(steps):

    #trying a small value
    #for D = -1
    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq1, Br1_sol1[-1], Bp1_sol1)
    Br1_sol1 = np.append(Br1_sol1, [Br_temp], axis = 0)
    Bp1_sol1 = np.append(Bp1_sol1, [Bp_temp], axis = 0)

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq1, Br2_sol1[-1], Bp2_sol1)
    Br2_sol1 = np.append(Br2_sol1, [Br_temp], axis = 0)
    Bp2_sol1 = np.append(Bp2_sol1, [Bp_temp], axis = 0)

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq1, Br3_sol1[-1], Bp3_sol1)
    Br3_sol1 = np.append(Br3_sol1, [Br_temp], axis = 0)
    Bp3_sol1 = np.append(Bp3_sol1, [Bp_temp], axis = 0)

    #tryign a large value
    #for D = -20
    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq2, Br1_sol2[-1], Bp1_sol2)
    Br1_sol2 = np.append(Br1_sol2, [Br_temp], axis = 0)
    Bp1_sol2 = np.append(Bp1_sol2, [Bp_temp], axis = 0)

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq2, Br2_sol2[-1], Bp2_sol2)
    Br2_sol2 = np.append(Br2_sol2, [Br_temp], axis = 0)
    Bp2_sol2 = np.append(Bp2_sol2, [Bp_temp], axis = 0)

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq2, Br3_sol2[-1], Bp3_sol2)
    Br3_sol2 = np.append(Br3_sol2, [Br_temp], axis = 0)
    Bp3_sol2 = np.append(Bp3_sol2, [Bp_temp], axis = 0)

```

```

#for D = -14
Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq3, Br1_sol3[-1], Bp1_sol
Br1_sol3 = np.append(Br1_sol3, [Br_temp], axis = 0)
Bp1_sol3 = np.append(Bp1_sol3, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq3, Br2_sol3[-1], Bp2_sol
Br2_sol3 = np.append(Br2_sol3, [Br_temp], axis = 0)
Bp2_sol3 = np.append(Bp2_sol3, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq3, Br3_sol3[-1], Bp3_sol
Br3_sol3 = np.append(Br3_sol3, [Br_temp], axis = 0)
Bp3_sol3 = np.append(Bp3_sol3, [Bp_temp], axis = 0)

#for D = -12
Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq4, Br1_sol4[-1], Bp1_sol
Br1_sol4 = np.append(Br1_sol4, [Br_temp], axis = 0)
Bp1_sol4 = np.append(Bp1_sol4, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq4, Br2_sol4[-1], Bp2_sol
Br2_sol4 = np.append(Br2_sol4, [Br_temp], axis = 0)
Bp2_sol4 = np.append(Bp2_sol4, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq4, Br3_sol4[-1], Bp3_sol
Br3_sol4 = np.append(Br3_sol4, [Br_temp], axis = 0)
Bp3_sol4 = np.append(Bp3_sol4, [Bp_temp], axis = 0)

#for D = -10
Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq5, Br1_sol5[-1], Bp1_sol
Br1_sol5 = np.append(Br1_sol5, [Br_temp], axis = 0)
Bp1_sol5 = np.append(Bp1_sol5, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq5, Br2_sol5[-1], Bp2_sol
Br2_sol5 = np.append(Br2_sol5, [Br_temp], axis = 0)
Bp2_sol5 = np.append(Bp2_sol5, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq5, Br3_sol5[-1], Bp3_sol
Br3_sol5 = np.append(Br3_sol5, [Br_temp], axis = 0)
Bp3_sol5 = np.append(Bp3_sol5, [Bp_temp], axis = 0)

#for D = -8
Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq6, Br1_sol6[-1], Bp1_sol
Br1_sol6 = np.append(Br1_sol6, [Br_temp], axis = 0)
Bp1_sol6 = np.append(Bp1_sol6, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq6, Br2_sol6[-1], Bp2_sol
Br2_sol6 = np.append(Br2_sol6, [Br_temp], axis = 0)
Bp2_sol6 = np.append(Bp2_sol6, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq6, Br3_sol6[-1], Bp3_sol
Br3_sol6 = np.append(Br3_sol6, [Br_temp], axis = 0)
Bp3_sol6 = np.append(Bp3_sol6, [Bp_temp], axis = 0)

#for D = -6
Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq7, Br1_sol7[-1], Bp1_sol
Br1_sol7 = np.append(Br1_sol7, [Br_temp], axis = 0)
Bp1_sol7 = np.append(Bp1_sol7, [Bp_temp], axis = 0)

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq7, Br2_sol7[-1], Bp2_sol
Br2_sol7 = np.append(Br2_sol7, [Br_temp], axis = 0)
Bp2_sol7 = np.append(Bp2_sol7, [Bp_temp], axis = 0)

```

```

Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq7, Br3_sol7[-1], Bp3_sol
Br3_sol7 = np.append(Br3_sol7, [Br_temp], axis = 0)
Bp3_sol7 = np.append(Bp3_sol7, [Bp_temp], axis = 0)

t = np.append(t, t[-1] + dt)

```

```

In [ ]: #saving the data
np.save('Br1_sols.npy', np.array([[Br1_sol1], [Br1_sol2], [Br1_sol3], [Br
np.save('Bp1_sols.npy', np.array([[Bp1_sol1], [Bp1_sol2], [Bp1_sol3], [Bp
np.save('Br2_sols.npy', np.array([[Br2_sol1], [Br2_sol2], [Br2_sol3], [Br
np.save('Bp2_sols.npy', np.array([[Bp2_sol1], [Bp2_sol2], [Bp2_sol3], [Bp
np.save('Br3_sols.npy', np.array([[Br3_sol1], [Br3_sol2], [Br3_sol3], [Br
np.save('Bp3_sols.npy', np.array([[Bp3_sol1], [Bp3_sol2], [Bp3_sol3], [Bp

```

```

In [ ]: fig, axs = plt.subplots(3, 2, figsize = (14, 21))
i = 1000
fig.suptitle("\n".join(wrap("Time evolution of Br for D=-1 (blue to red =
    "))

axs[0, 0].plot(z, Br1_sol1[0], color = 'red')
axs[0, 0].set_title('Seed 1: Dirichlet BC')
axs[0, 0].set(xlabel='z/h', ylabel='Br')

axs[0, 1].plot(z, Bp1_sol1[0], color = 'red')
axs[0, 1].set_title('Seed 1: Dirichlet BC')
axs[0, 1].set(xlabel='z/h', ylabel='Bp')

axs[1, 0].plot(z, Br2_sol1[0], color = 'red')
axs[1, 0].set_title('Seed 2: Dirichlet BC')
axs[1, 0].set(xlabel='z/h', ylabel='Br')

axs[1, 1].plot(z, Bp2_sol1[0], color = 'red')
axs[1, 1].set_title('Seed 2: Dirichlet BC')
axs[1, 1].set(xlabel='z/h', ylabel='Bp')

axs[2, 0].plot(z, Br3_sol1[0], color = 'red')
axs[2, 0].set_title('Seed 3: Dirichlet BC')
axs[2, 0].set(xlabel='z/h', ylabel='Br')

axs[2, 1].plot(z, Bp3_sol1[0], color = 'red')
axs[2, 1].set_title('Seed 3: Dirichlet BC')
axs[2, 1].set(xlabel='z/h', ylabel='Bp')

while i < steps:
    axs[0, 0].plot(z, Br1_sol1[i], color = (round(i/(2*steps), 5), 0.1, 0
    axs[0, 0].set_title('Seed 1: Dirichlet BC')
    axs[0, 0].set(xlabel='z/h', ylabel='Br')

    axs[0, 1].plot(z, Bp1_sol1[i], color = (round(i/(2*steps), 5), 0.1, 0
    axs[0, 1].set_title('Seed 1: Dirichlet BC')
    axs[0, 1].set(xlabel='z/h', ylabel='Bp')

    axs[1, 0].plot(z, Br2_sol1[i], color = (round(i/(2*steps), 5), 0.1, 0
    axs[1, 0].set_title('Seed 2: Dirichlet BC')
    axs[1, 0].set(xlabel='z/h', ylabel='Br')

    axs[1, 1].plot(z, Bp2_sol1[i], color = (round(i/(2*steps), 5), 0.1, 0
    axs[1, 1].set_title('Seed 2: Dirichlet BC')
    axs[1, 1].set(xlabel='z/h', ylabel='Bp')

```

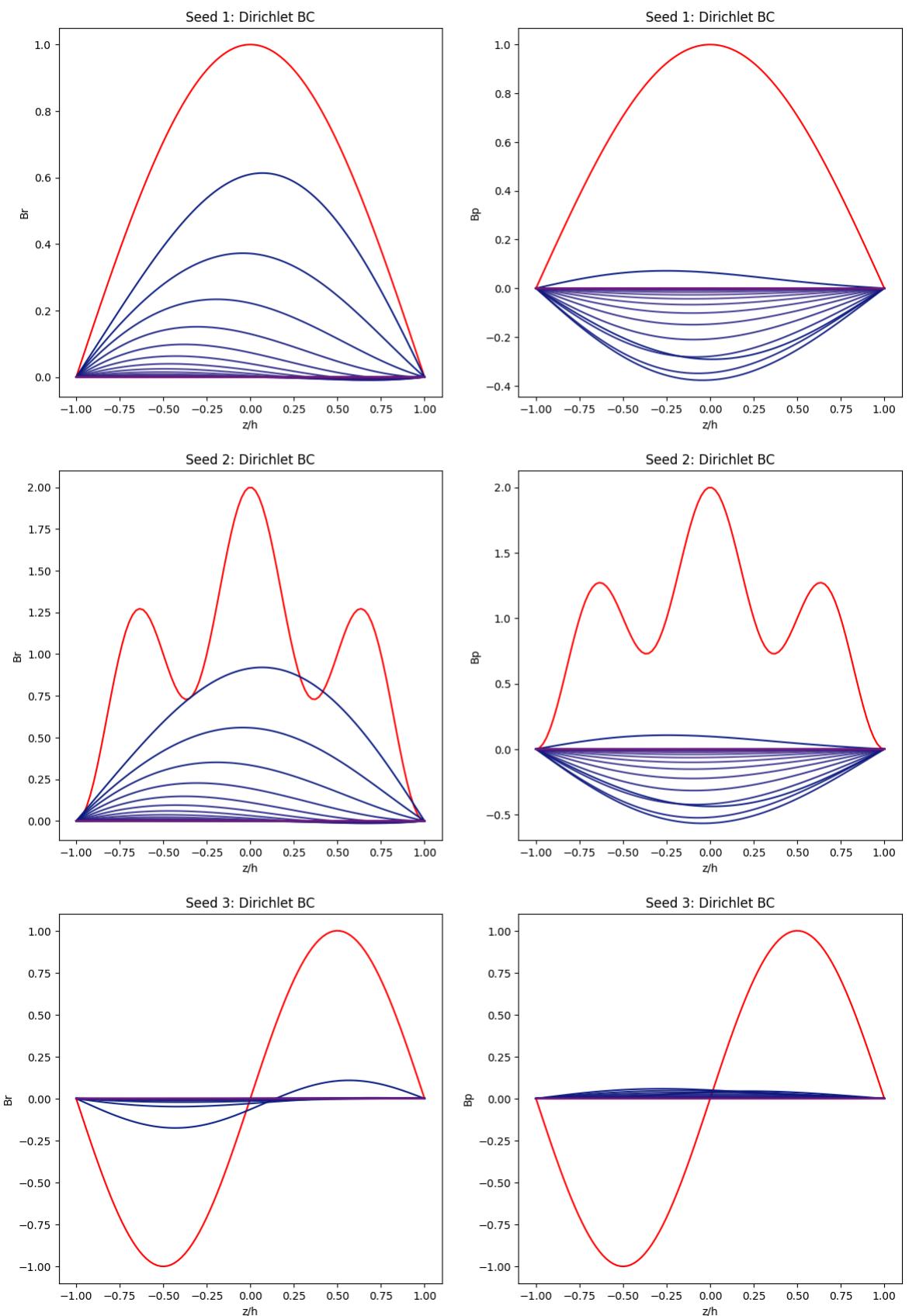
```
axs[2, 0].plot(z, Br3_sol1[i], color = (round(i/(2*steps), 5), 0.1, 0
axs[2, 0].set_title('Seed 3: Dirichlet BC')
axs[2, 0].set(xlabel='z/h', ylabel='Br')

axs[2, 1].plot(z, Bp3_sol1[i], color = (round(i/(2*steps), 5), 0.1, 0
axs[2, 1].set_title('Seed 3: Dirichlet BC')
axs[2, 1].set(xlabel='z/h', ylabel='Bp')

i += 1000

# for ax in axs.flat:
#     ax.set(xlabel='z/h', ylabel='Br')
```

Time evolution of Br for D=-1 (blue to red = early to late times, bright red = seed)



```
In [ ]: fig2, axs2 = plt.subplots(3, 2, figsize = (14, 21))
i = 1000
fig2.suptitle("\n".join(wrap("Time evolution of Br for D=-15 (blue to red"))
```

```

axs2[0, 0].plot(z, Br1_sol2[0], color = 'red')
axs2[0, 0].set_title('Seed 1: Dirichlet BC')
axs2[0, 0].set(xlabel='z/h', ylabel='Br')

axs2[0, 1].plot(z, Bp1_sol2[0], color = 'red')
axs2[0, 1].set_title('Seed 1: Dirichlet BC')
axs2[0, 1].set(xlabel='z/h', ylabel='Bp')

axs2[1, 0].plot(z, Br2_sol2[0], color = 'red')
axs2[1, 0].set_title('Seed 2: Dirichlet BC')
axs2[1, 0].set(xlabel='z/h', ylabel='Br')

axs2[1, 1].plot(z, Bp2_sol2[0], color = 'red')
axs2[1, 1].set_title('Seed 2: Dirichlet BC')
axs2[1, 1].set(xlabel='z/h', ylabel='Bp')

axs2[2, 0].plot(z, Br3_sol2[0], color = 'red')
axs2[2, 0].set_title('Seed 3: Dirichlet BC')
axs2[2, 0].set(xlabel='z/h', ylabel='Br')

axs2[2, 1].plot(z, Bp3_sol2[0], color = 'red')
axs2[2, 1].set_title('Seed 3: Dirichlet BC')
axs2[2, 1].set(xlabel='z/h', ylabel='Bp')

while i < steps:
    axs2[0, 0].plot(z, Br1_sol2[i], color = (round(i/(2*steps), 5), 0.1,
    axs2[0, 0].set_title('Seed 1: Dirichlet BC')
    axs2[0, 0].set(xlabel='z/h', ylabel='Br')

    axs2[0, 1].plot(z, Bp1_sol2[i], color = (round(i/(2*steps), 5), 0.1,
    axs2[0, 1].set_title('Seed 1: Dirichlet BC')
    axs2[0, 1].set(xlabel='z/h', ylabel='Bp')

    axs2[1, 0].plot(z, Br2_sol2[i], color = (round(i/(2*steps), 5), 0.1,
    axs2[1, 0].set_title('Seed 2: Dirichlet BC')
    axs2[1, 0].set(xlabel='z/h', ylabel='Br')

    axs2[1, 1].plot(z, Bp2_sol2[i], color = (round(i/(2*steps), 5), 0.1,
    axs2[1, 1].set_title('Seed 2: Dirichlet BC')
    axs2[1, 1].set(xlabel='z/h', ylabel='Bp')

    axs2[2, 0].plot(z, Br3_sol2[i], color = (round(i/(2*steps), 5), 0.1,
    axs2[2, 0].set_title('Seed 3: Dirichlet BC')
    axs2[2, 0].set(xlabel='z/h', ylabel='Br')

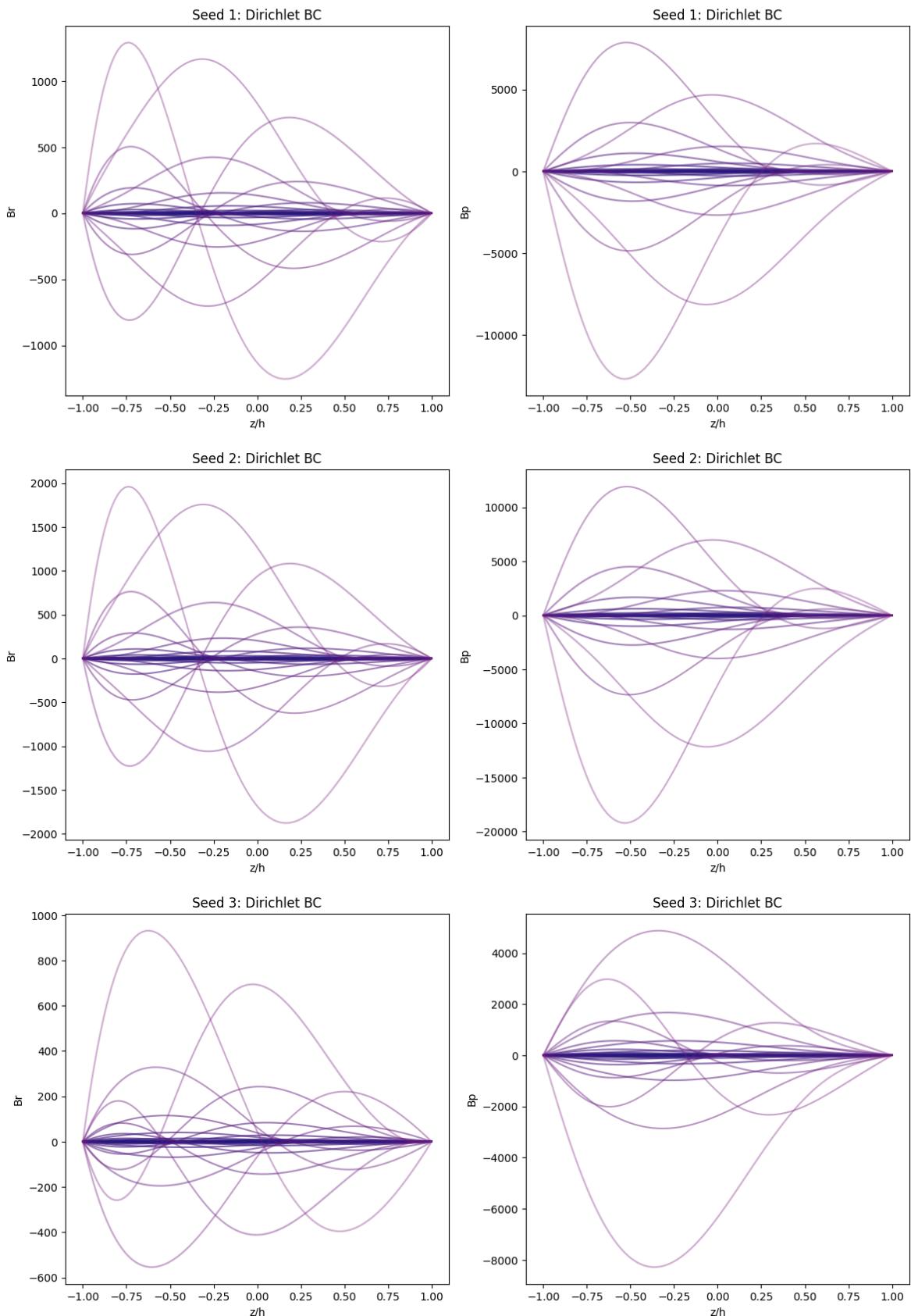
    axs2[2, 1].plot(z, Bp3_sol2[i], color = (round(i/(2*steps), 5), 0.1,
    axs2[2, 1].set_title('Seed 3: Dirichlet BC')
    axs2[2, 1].set(xlabel='z/h', ylabel='Bp')

    i += 1000

# for ax in axs2.flat:
#     ax.set(xlabel='z/h', ylabel='Br')

```

Time evolution of Br for D=-15 (blue to red =
early to late times, bright red = seed)



```
In [ ]: fig3, axs3 = plt.subplots(3, 2, figsize = (14, 21))
i = 1000
fig3.suptitle("\n".join(wrap("Time evolution of Br for D=-10 (blue to red = early to late times, bright red = seed)")))
```

```

axs3[0, 0].plot(z, Br1_sol3[0], color = 'red')
axs3[0, 0].set_title('Seed 1: Dirichlet BC')
axs3[0, 0].set(xlabel='z/h', ylabel='Br')

axs3[0, 1].plot(z, Bp1_sol3[0], color = 'red')
axs3[0, 1].set_title('Seed 1: Dirichlet BC')
axs3[0, 1].set(xlabel='z/h', ylabel='Bp')

axs3[1, 0].plot(z, Br2_sol3[0], color = 'red')
axs3[1, 0].set_title('Seed 2: Dirichlet BC')
axs3[1, 0].set(xlabel='z/h', ylabel='Br')

axs3[1, 1].plot(z, Bp2_sol3[0], color = 'red')
axs3[1, 1].set_title('Seed 2: Dirichlet BC')
axs3[1, 1].set(xlabel='z/h', ylabel='Bp')

axs3[2, 0].plot(z, Br3_sol3[0], color = 'red')
axs3[2, 0].set_title('Seed 3: Dirichlet BC')
axs3[2, 0].set(xlabel='z/h', ylabel='Br')

axs3[2, 1].plot(z, Bp3_sol3[0], color = 'red')
axs3[2, 1].set_title('Seed 3: Dirichlet BC')
axs3[2, 1].set(xlabel='z/h', ylabel='Bp')

while i < steps:
    axs3[0, 0].plot(z, Br1_sol3[i], color = (round(i/(2*steps), 5), 0.1,
    axs3[0, 0].set_title('Seed 1: Dirichlet BC')
    axs3[0, 0].set(xlabel='z/h', ylabel='Br')

    axs3[0, 1].plot(z, Bp1_sol3[i], color = (round(i/(2*steps), 5), 0.1,
    axs3[0, 1].set_title('Seed 1: Dirichlet BC')
    axs3[0, 1].set(xlabel='z/h', ylabel='Bp')

    axs3[1, 0].plot(z, Br2_sol3[i], color = (round(i/(2*steps), 5), 0.1,
    axs3[1, 0].set_title('Seed 2: Dirichlet BC')
    axs3[1, 0].set(xlabel='z/h', ylabel='Br')

    axs3[1, 1].plot(z, Bp2_sol3[i], color = (round(i/(2*steps), 5), 0.1,
    axs3[1, 1].set_title('Seed 2: Dirichlet BC')
    axs3[1, 1].set(xlabel='z/h', ylabel='Bp')

    axs3[2, 0].plot(z, Br3_sol3[i], color = (round(i/(2*steps), 5), 0.1,
    axs3[2, 0].set_title('Seed 3: Dirichlet BC')
    axs3[2, 0].set(xlabel='z/h', ylabel='Br')

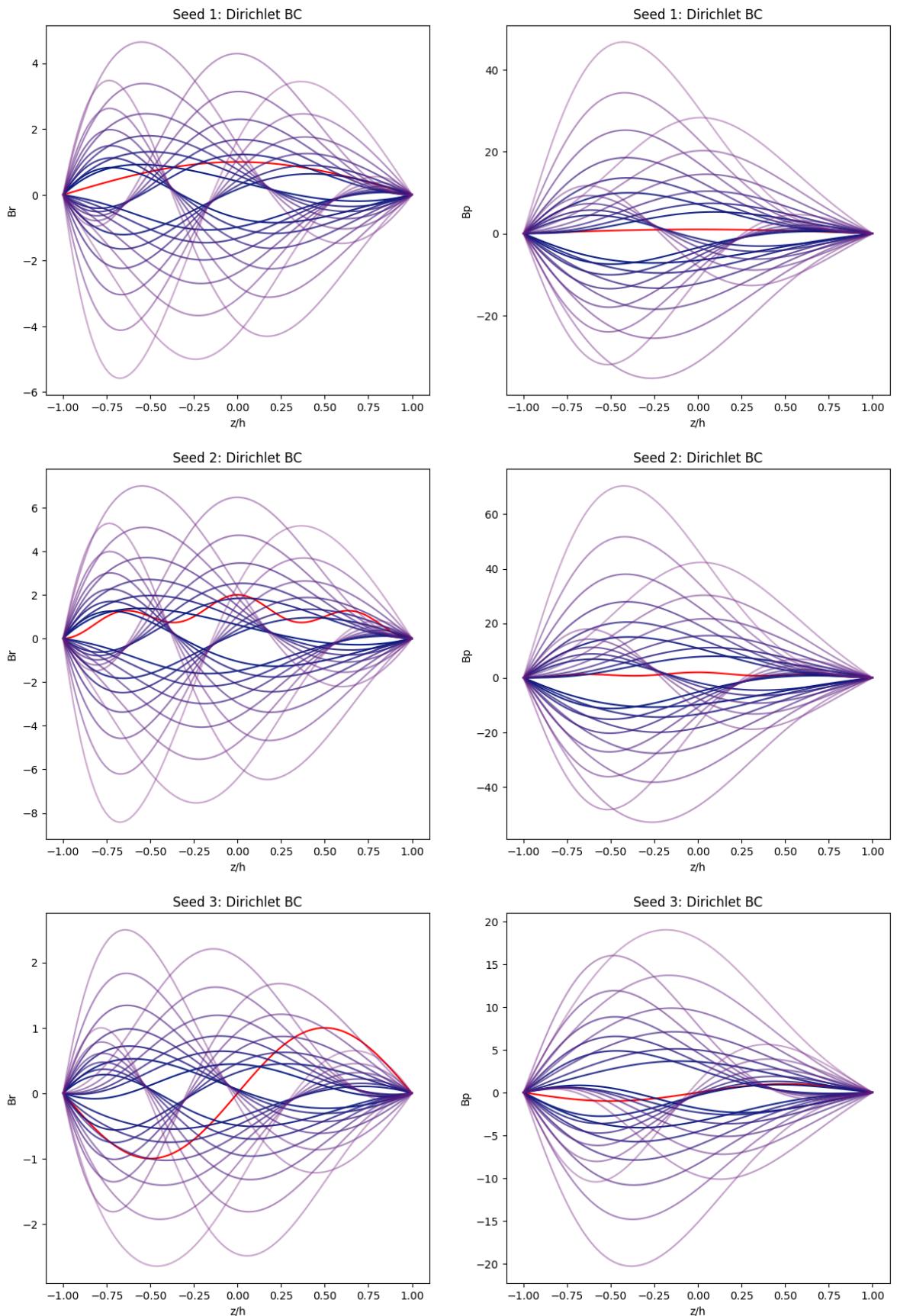
    axs3[2, 1].plot(z, Bp3_sol3[i], color = (round(i/(2*steps), 5), 0.1,
    axs3[2, 1].set_title('Seed 3: Dirichlet BC')
    axs3[2, 1].set(xlabel='z/h', ylabel='Bp')

    i += 1000

# for ax in axs3.flat:
#     ax.set(xlabel='z/h', ylabel='Br')

```

Time evolution of Br for D=-10 (blue to red = early to late times, bright red = seed)



```
In [ ]: fig4, axs4 = plt.subplots(3, 2, figsize = (14, 21))
i = 1000
```

```

fig4.suptitle("\n".join(wrap("Time evolution of Br for D=-8 (blue to red

axs4[0, 0].plot(z, Br1_sol4[0], color = 'red')
axs4[0, 0].set_title('Seed 1: Dirichlet BC')
axs4[0, 0].set(xlabel='z/h', ylabel='Br')

axs4[0, 1].plot(z, Bp1_sol4[0], color = 'red')
axs4[0, 1].set_title('Seed 1: Dirichlet BC')
axs4[0, 1].set(xlabel='z/h', ylabel='Bp')

axs4[1, 0].plot(z, Br2_sol4[0], color = 'red')
axs4[1, 0].set_title('Seed 2: Dirichlet BC')
axs4[1, 0].set(xlabel='z/h', ylabel='Br')

axs4[1, 1].plot(z, Bp2_sol4[0], color = 'red')
axs4[1, 1].set_title('Seed 2: Dirichlet BC')
axs4[1, 1].set(xlabel='z/h', ylabel='Bp')

axs4[2, 0].plot(z, Br3_sol4[0], color = 'red')
axs4[2, 0].set_title('Seed 3: Dirichlet BC')
axs4[2, 0].set(xlabel='z/h', ylabel='Br')

axs4[2, 1].plot(z, Bp3_sol4[0], color = 'red')
axs4[2, 1].set_title('Seed 3: Dirichlet BC')
axs4[2, 1].set(xlabel='z/h', ylabel='Bp')

while i < 3*10**4:
    axs4[0, 0].plot(z, Br1_sol4[i], color = (round(i/(2*steps), 5), 0.1,
    axs4[0, 0].set_title('Seed 1: Dirichlet BC')
    axs4[0, 0].set(xlabel='z/h', ylabel='Br')

    axs4[0, 1].plot(z, Bp1_sol4[i], color = (round(i/(2*steps), 5), 0.1,
    axs4[0, 1].set_title('Seed 1: Dirichlet BC')
    axs4[0, 1].set(xlabel='z/h', ylabel='Bp')

    axs4[1, 0].plot(z, Br2_sol4[i], color = (round(i/(2*steps), 5), 0.1,
    axs4[1, 0].set_title('Seed 2: Dirichlet BC')
    axs4[1, 0].set(xlabel='z/h', ylabel='Br')

    axs4[1, 1].plot(z, Bp2_sol4[i], color = (round(i/(2*steps), 5), 0.1,
    axs4[1, 1].set_title('Seed 2: Dirichlet BC')
    axs4[1, 1].set(xlabel='z/h', ylabel='Bp')

    axs4[2, 0].plot(z, Br3_sol4[i], color = (round(i/(2*steps), 5), 0.1,
    axs4[2, 0].set_title('Seed 3: Dirichlet BC')
    axs4[2, 0].set(xlabel='z/h', ylabel='Br')

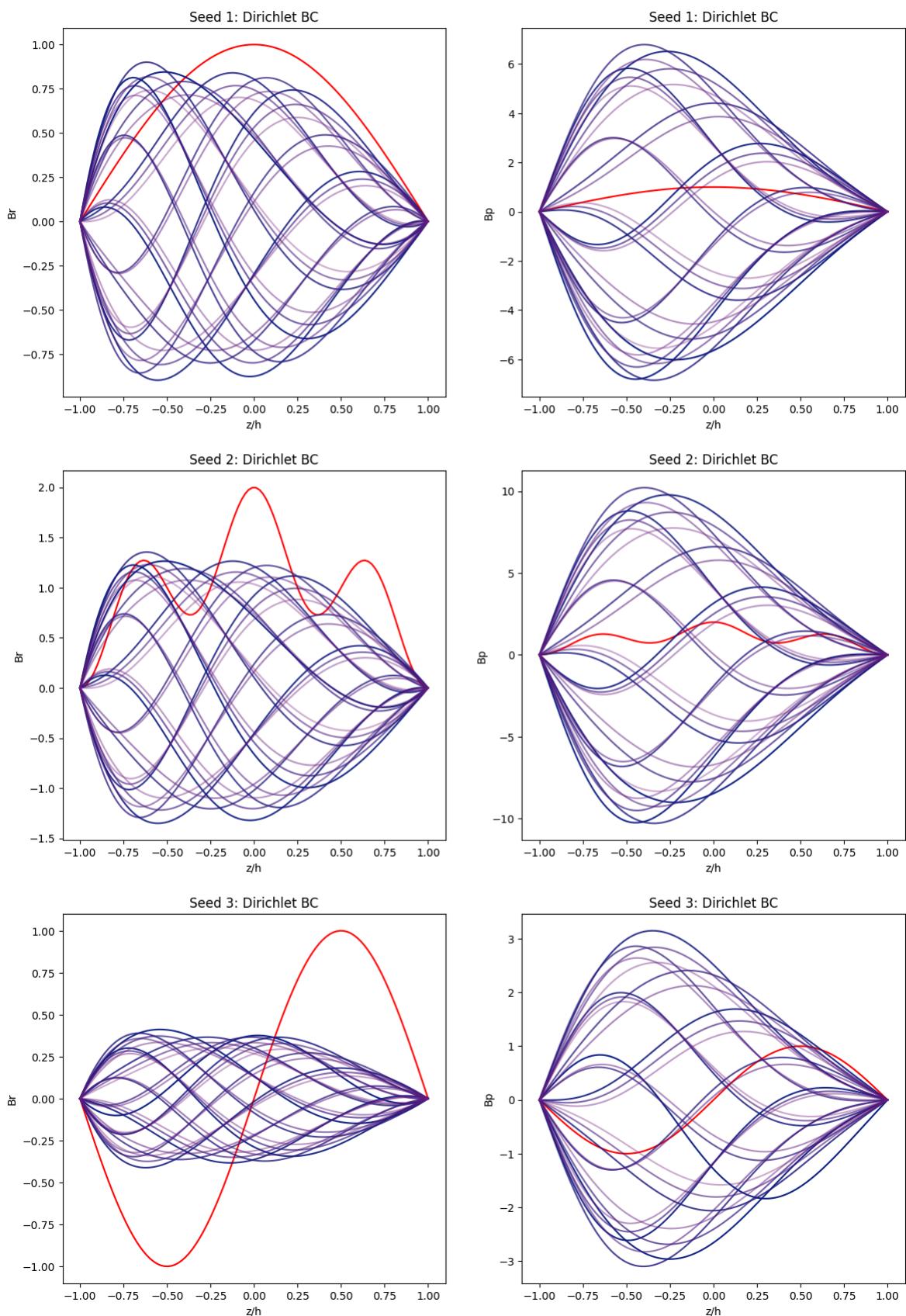
    axs4[2, 1].plot(z, Bp3_sol4[i], color = (round(i/(2*steps), 5), 0.1,
    axs4[2, 1].set_title('Seed 3: Dirichlet BC')
    axs4[2, 1].set(xlabel='z/h', ylabel='Bp')

    i += 1000

# for ax in axs4.flat:
#     ax.set(xlabel='z/h', ylabel='Br')

```

Time evolution of Br for D=-8 (blue to red = early to late times, bright red = seed)



initialising the solutions

```

Br1_sol5 = np.array([Br1]) Bp1_sol5 = np.array([Bp1])

Br2_sol5 = np.array([Br2]) Bp2_sol5 = np.array([Bp2])

Br3_sol5 = np.array([Br3]) Bp3_sol5 = np.array([Bp3])

t = np.array([t0])

steps= 6*10**4 for i in range(steps):

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq5,
    Br1_sol5[-1], Bp1_sol5[-1], Br1_0, Br1_n, Bp1_0, Bp1_n,
    t[-1], dt, dz, bc = 'dir')
    Br1_sol5 = np.append(Br1_sol5, [Br_temp], axis = 0)
    Bp1_sol5 = np.append(Bp1_sol5, [Bp_temp], axis = 0)

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq5,
    Br2_sol5[-1], Bp2_sol5[-1], Br1_0, Br1_n, Bp1_0, Bp1_n,
    t[-1], dt, dz, bc = 'dir')
    Br2_sol5 = np.append(Br2_sol5, [Br_temp], axis = 0)
    Bp2_sol5 = np.append(Bp2_sol5, [Bp_temp], axis = 0)

    Br_temp, Bp_temp = rk4(Br_diff_eq, Bp_diff_eq5,
    Br3_sol5[-1], Bp3_sol5[-1], Br1_0, Br1_n, Bp1_0, Bp1_n,
    t[-1], dt, dz, bc = 'dir')
    Br3_sol5 = np.append(Br3_sol5, [Br_temp], axis = 0)
    Bp3_sol5 = np.append(Bp3_sol5, [Bp_temp], axis = 0)

    t = np.append(t, t[-1] + dt)

```

```

In [ ]: fig5, axs5 = plt.subplots(3, 2, figsize = (14, 21))
i = 1000
fig5.suptitle("\n".join(wrap("Time evolution of Br for D=-1 (blue to red
seed 1) and Bp for D=-1 (red to blue seed 2) for different boundary conditions
BC1: Dirichlet BC, BC2: Neumann BC, BC3: Mixed BC"))
axs5[0, 0].plot(z, Br1_sol5[0], color = 'red')
axs5[0, 0].set_title('Seed 1: Dirichlet BC')
axs5[0, 0].set(xlabel='z/h', ylabel='Br')

axs5[0, 1].plot(z, Bp1_sol5[0], color = 'red')
axs5[0, 1].set_title('Seed 1: Dirichlet BC')
axs5[0, 1].set(xlabel='z/h', ylabel='Bp')

axs5[1, 0].plot(z, Br2_sol5[0], color = 'red')
axs5[1, 0].set_title('Seed 2: Dirichlet BC')
axs5[1, 0].set(xlabel='z/h', ylabel='Br')

axs5[1, 1].plot(z, Bp2_sol5[0], color = 'red')
axs5[1, 1].set_title('Seed 2: Dirichlet BC')
axs5[1, 1].set(xlabel='z/h', ylabel='Bp')

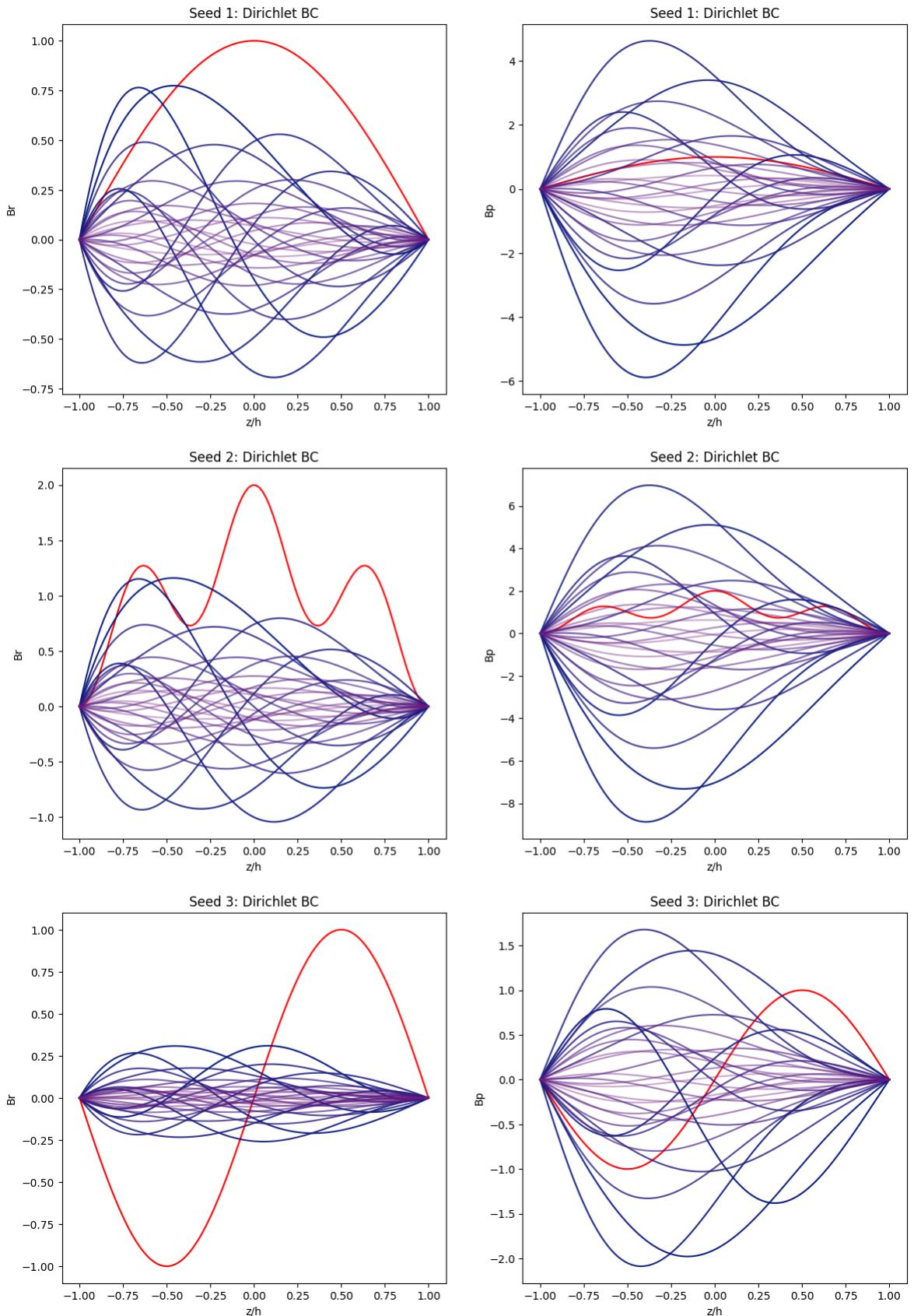
axs5[2, 0].plot(z, Br3_sol5[0], color = 'red')
axs5[2, 0].set_title('Seed 3: Dirichlet BC')
axs5[2, 0].set(xlabel='z/h', ylabel='Br')

axs5[2, 1].plot(z, Bp3_sol5[0], color = 'red')
axs5[2, 1].set_title('Seed 3: Dirichlet BC')
axs5[2, 1].set(xlabel='z/h', ylabel='Bp')

```

```
while i < steps:  
    axs5[0, 0].plot(z, Br1_sol5[i], color = (round(i/(2*steps), 5), 0.1,  
    axs5[0, 0].set_title('Seed 1: Dirichlet BC')  
    axs5[0, 0].set(xlabel='z/h', ylabel='Br')  
  
    axs5[0, 1].plot(z, Bp1_sol5[i], color = (round(i/(2*steps), 5), 0.1,  
    axs5[0, 1].set_title('Seed 1: Dirichlet BC')  
    axs5[0, 1].set(xlabel='z/h', ylabel='Bp')  
  
    axs5[1, 0].plot(z, Br2_sol5[i], color = (round(i/(2*steps), 5), 0.1,  
    axs5[1, 0].set_title('Seed 2: Dirichlet BC')  
    axs5[1, 0].set(xlabel='z/h', ylabel='Br')  
  
    axs5[1, 1].plot(z, Bp2_sol5[i], color = (round(i/(2*steps), 5), 0.1,  
    axs5[1, 1].set_title('Seed 2: Dirichlet BC')  
    axs5[1, 1].set(xlabel='z/h', ylabel='Bp')  
  
    axs5[2, 0].plot(z, Br3_sol5[i], color = (round(i/(2*steps), 5), 0.1,  
    axs5[2, 0].set_title('Seed 3: Dirichlet BC')  
    axs5[2, 0].set(xlabel='z/h', ylabel='Br')  
  
    axs5[2, 1].plot(z, Bp3_sol5[i], color = (round(i/(2*steps), 5), 0.1,  
    axs5[2, 1].set_title('Seed 3: Dirichlet BC')  
    axs5[2, 1].set(xlabel='z/h', ylabel='Bp')  
  
    i += 1000  
  
# for ax in axs5.flat:  
#     ax.set(xlabel='z/h', ylabel='Br')
```

Time evolution of Br for D=-1 (blue to red = early to late times, bright red = seed)



From the above simulations, I think the critical dynamo number lies somewhere between around to -12. We can

see that for D < -12, the fields get amplified.

```
In [ ]: def lin(t, A, b):
    return A + b*t

def fit_lin(data, t):
    #Fit exponential curve to the data
    popt, pcov = curve_fit(lin, t, data)

    return popt # Return parameters A and b of the fit

In [ ]: #The growth rate from the no-z approximation is given by equation 11.42 c
def gamma_noz(D):
    return (eta_T/(h**2))*(np.sqrt(-D) - 1)

#calculating the growth rate for no-z approximation
gamma_noz_array = gamma_noz(np.array(D_array))

#calculating the growth rate from the evolution
#I will calculate it from the 1st seed field only
Br1_evol1 = np.absolute(Br1_sol1[:, z_res//2])
Br1_evol2 = np.absolute(Br1_sol2[:, z_res//2])
Br1_evol3 = np.absolute(Br1_sol3[:, z_res//2])
Br1_evol4 = np.absolute(Br1_sol4[:, z_res//2])
Br1_evol5 = np.absolute(Br1_sol5[:, z_res//2])
Br1_evol6 = np.absolute(Br1_sol6[:, z_res//2])
Br1_evol7 = np.absolute(Br1_sol7[:, z_res//2])

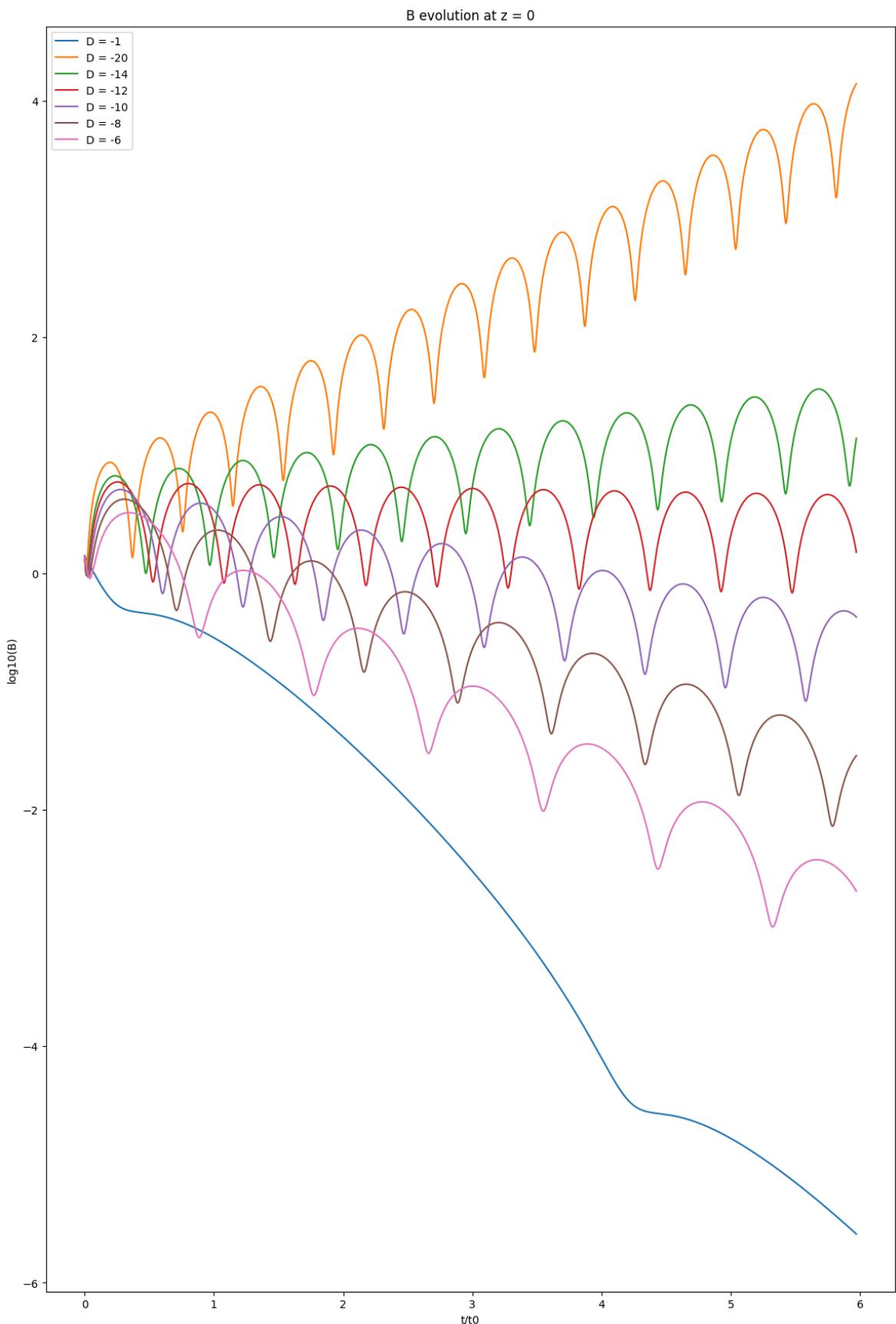
Bp1_evol1 = np.absolute(Bp1_sol1[:, z_res//2])
Bp1_evol2 = np.absolute(Bp1_sol2[:, z_res//2])
Bp1_evol3 = np.absolute(Bp1_sol3[:, z_res//2])
Bp1_evol4 = np.absolute(Bp1_sol4[:, z_res//2])
Bp1_evol5 = np.absolute(Bp1_sol5[:, z_res//2])
Bp1_evol6 = np.absolute(Bp1_sol6[:, z_res//2])
Bp1_evol7 = np.absolute(Bp1_sol7[:, z_res//2])

B_evol1 = np.sqrt(Br1_evol1**2 + Bp1_evol1**2)
B_evol2 = np.sqrt(Br1_evol2**2 + Bp1_evol2**2)
B_evol3 = np.sqrt(Br1_evol3**2 + Bp1_evol3**2)
B_evol4 = np.sqrt(Br1_evol4**2 + Bp1_evol4**2)
B_evol5 = np.sqrt(Br1_evol5**2 + Bp1_evol5**2)
B_evol6 = np.sqrt(Br1_evol6**2 + Bp1_evol6**2)
B_evol7 = np.sqrt(Br1_evol7**2 + Bp1_evol7**2)

#plotting the decay of the magnetic field

fig, axs = plt.subplots(figsize = (14, 21))
axs.plot(t, np.log10(B_evol1), label = 'D = -1')
axs.plot(t, np.log10(B_evol2), label = 'D = -20')
axs.plot(t, np.log10(B_evol3), label = 'D = -14')
axs.plot(t, np.log10(B_evol4), label = 'D = -12')
axs.plot(t, np.log10(B_evol5), label = 'D = -10')
axs.plot(t, np.log10(B_evol6), label = 'D = -8')
axs.plot(t, np.log10(B_evol7), label = 'D = -6')
axs.set_title('B evolution at z = 0')
axs.set_xlabel='t/t0', ylabel='log10(B)')
axs.legend()
```

Out[]: <matplotlib.legend.Legend at 0x71c76b37b7c0>



```
In [ ]: #fitting the linear curve
B_popt1 = fit_lin(np.log10(B_evol1), t)
B_popt2 = fit_lin(np.log10(B_evol2), t)
B_popt3 = fit_lin(np.log10(B_evol3), t)
B_popt4 = fit_lin(np.log10(B_evol4), t)
B_popt5 = fit_lin(np.log10(B_evol5), t)
B_popt6 = fit_lin(np.log10(B_evol6), t)
B_popt7 = fit_lin(np.log10(B_evol7), t)
```

```

print("Growth rate for D = -20 from simulation is", B_popt2[1]/to, "s^-1",
print("Growth rate for D = -14 from simulation is", B_popt3[1]/to, "s^-1",
print("Growth rate for D = -12 from simulation is", B_popt4[1]/to, "s^-1",
print("Growth rate for D = -10 from simulation is", B_popt5[1]/to, "s^-1",

```

Growth rate for D = -20 from simulation is 2.340505866822039e-17 s^-1, and growth rate from no-z approximation is 1.458360423109395e-16 s^-1
Growth rate for D = -14 from simulation is 5.549968093400047e-18 s^-1, and growth rate from no-z approximation is 1.1515461025768293e-16 s^-1
Growth rate for D = -12 from simulation is -6.172768383703154e-19 s^-1, and growth rate from no-z approximation is 1.0349676166517718e-16 s^-1
Growth rate for D = -10 from simulation is -7.527023695646718e-18 s^-1, and growth rate from no-z approximation is 9.081960511432601e-17 s^-1

So, I think the critical dynamo number in my case is somewhere around D = -12. We can see that for the no-z approximation case, the rate is about an order or two higher.

```

In [ ]: def pitch(Br, Bp):
    return 180*np.arctan(Bp/Br)/np.pi

i = 10000
fig6, axs6 = plt.subplots(3, 1, figsize = (7, 21))
fig6.suptitle("\n".join(wrap("Time evolution of pitch angle (deg) for D="))

axs6[0].plot(z, pitch(Br1_sol5[0], Bp1_sol5[0]), color = 'red')
axs6[0].set_title('Seed 1: Dirichlet BC')
axs6[0].set(xlabel='z/h', ylabel='Pitch angle')

axs6[1].plot(z, pitch(Br2_sol5[0], Bp2_sol5[0]), color = 'red')
axs6[1].set_title('Seed 2: Dirichlet BC')
axs6[1].set(xlabel='z/h', ylabel='Pitch angle')

axs6[2].plot(z, pitch(Br3_sol5[0], Bp3_sol5[0]), color = 'red')
axs6[2].set_title('Seed 3: Dirichlet BC')
axs6[2].set(xlabel='z/h', ylabel='Pitch angle')

while i < steps:
    axs6[0].plot(z, pitch(Br1_sol5[i], Bp1_sol5[i]), color = (round(i/(2*
    axs6[0].set_title('Seed 1: Dirichlet BC')
    axs6[0].set(xlabel='z/h', ylabel='Pitch angle')

    axs6[1].plot(z, pitch(Br2_sol5[i], Bp2_sol5[i]), color = (round(i/(2*
    axs6[1].set_title('Seed 2: Dirichlet BC')
    axs6[1].set(xlabel='z/h', ylabel='Pitch angle')

    axs6[2].plot(z, pitch(Br3_sol5[i], Bp3_sol5[i]), color = (round(i/(2*
    axs6[2].set_title('Seed 3: Dirichlet BC')
    axs6[2].set(xlabel='z/h', ylabel='Pitch angle'))

    i += 10000

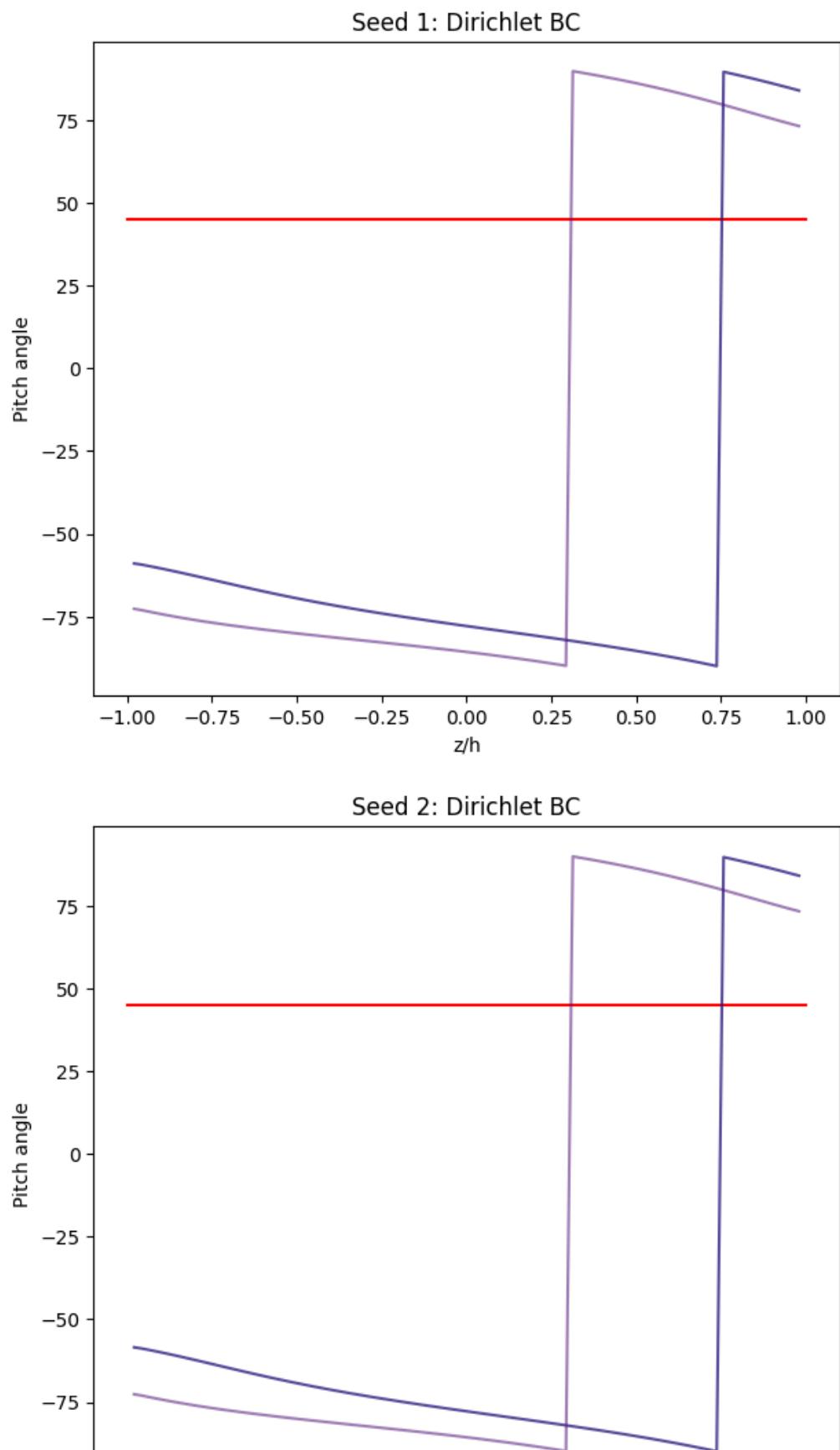
```

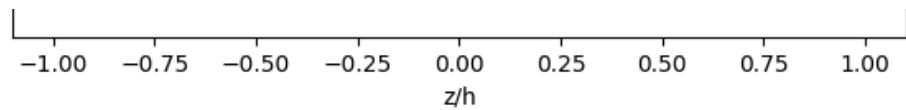
```

/tmp/ipykernel_33745/1949584569.py:2: RuntimeWarning: invalid value encountered
    return 180*np.arctan(Bp/Br)/np.pi

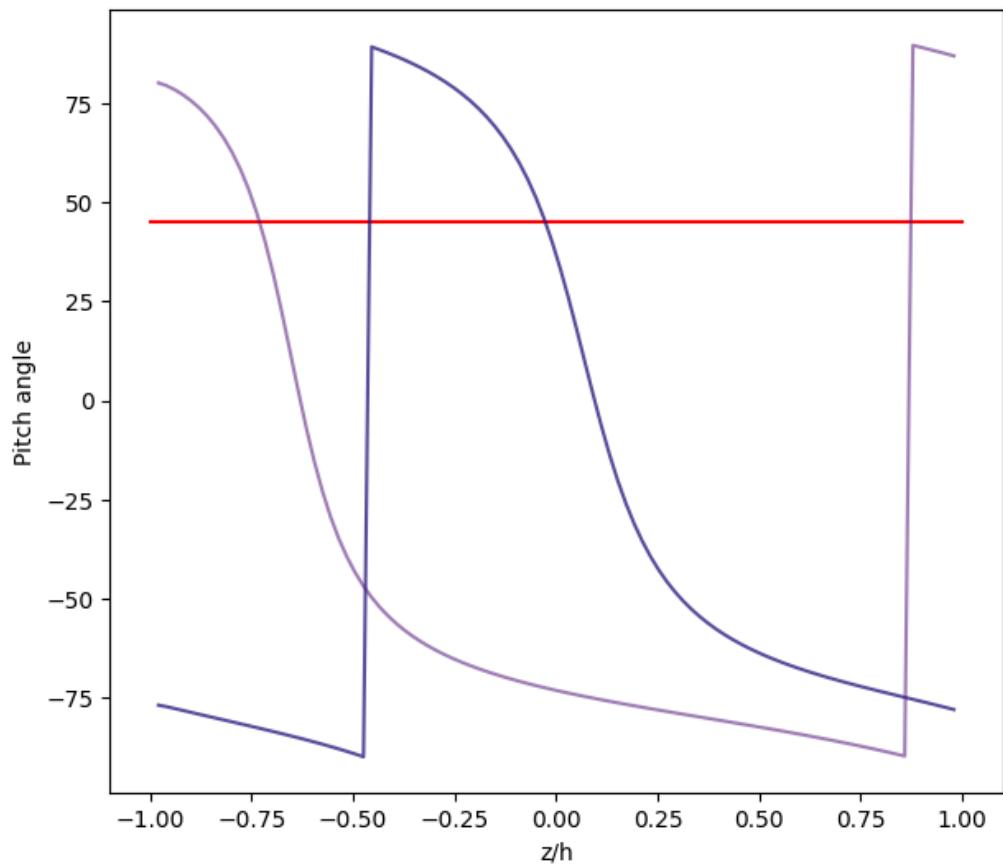
```

Time evolution of pitch angle (deg) for D=-6 (blue to red = early to late times, bright red = seed)





Seed 3: Dirichlet BC



```
In [ ]: i = 2000
fig6, axs6 = plt.subplots(3, 1, figsize = (7, 21))
fig6.suptitle("\n".join(wrap("Time evolution of pitch angle (deg) for D="))

axs6[0].plot(z, pitch(Br1_sol3[0], Bp1_sol3[0]), color = 'red')
axs6[0].set_title('Seed 1: Dirichlet BC')
axs6[0].set(xlabel='z/h', ylabel='Pitch angle')

axs6[1].plot(z, pitch(Br2_sol3[0], Bp2_sol3[0]), color = 'red')
axs6[1].set_title('Seed 2: Dirichlet BC')
axs6[1].set(xlabel='z/h', ylabel='Pitch angle')

axs6[2].plot(z, pitch(Br3_sol3[0], Bp3_sol3[0]), color = 'red')
axs6[2].set_title('Seed 3: Dirichlet BC')
axs6[2].set(xlabel='z/h', ylabel='Pitch angle')

while i < 3*10**4:
    axs6[0].plot(z, pitch(Br1_sol3[i], Bp1_sol3[i]), color = (round(i/(1*
    axs6[0].set_title('Seed 1: Dirichlet BC')
    axs6[0].set(xlabel='z/h', ylabel='Pitch angle')

    axs6[1].plot(z, pitch(Br2_sol3[i], Bp2_sol3[i]), color = (round(i/(1*
    axs6[1].set_title('Seed 2: Dirichlet BC')
    axs6[1].set(xlabel='z/h', ylabel='Pitch angle')

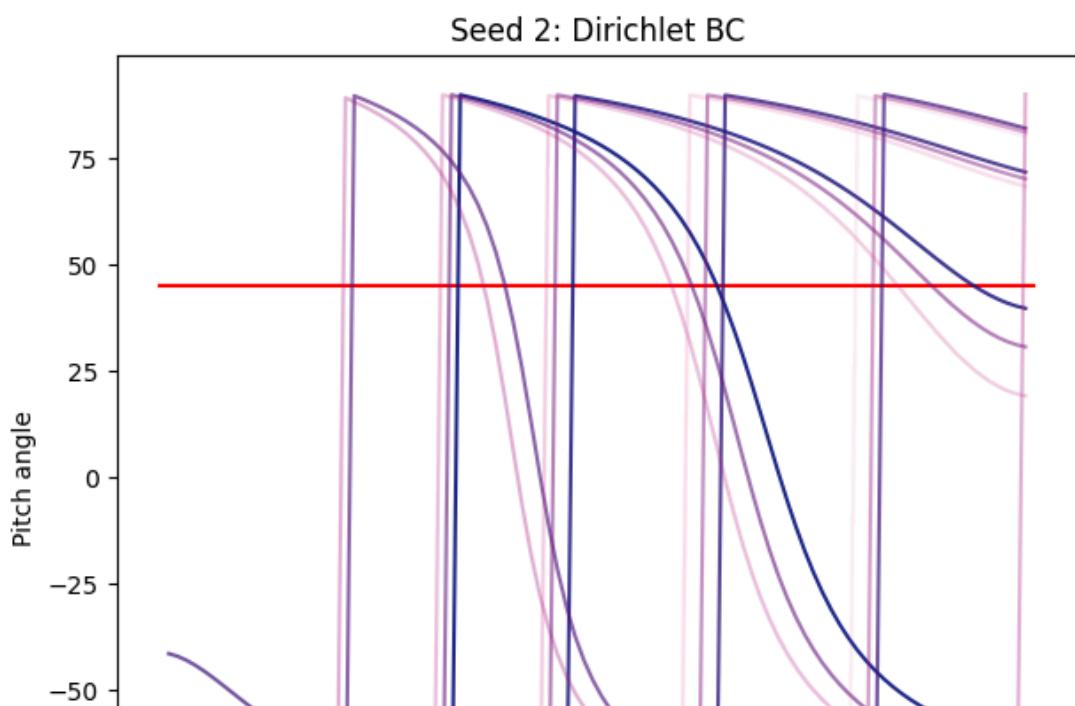
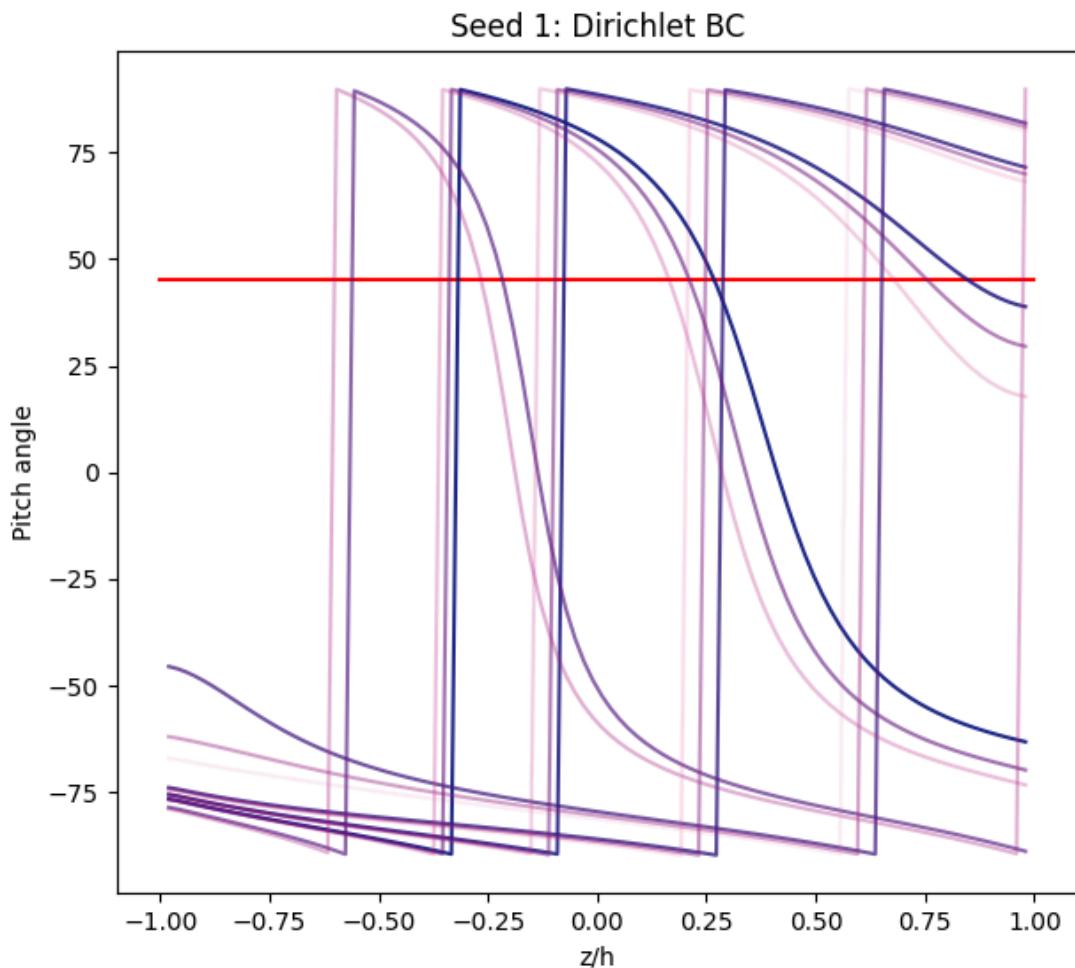
    axs6[2].plot(z, pitch(Br3_sol3[i], Bp3_sol3[i]), color = (round(i/(1*
```

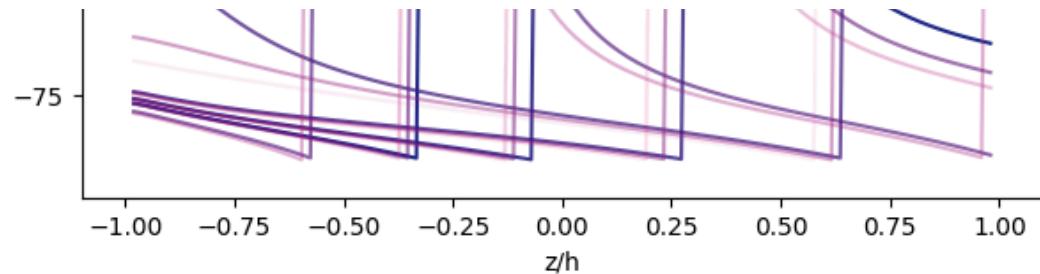
```
    axs6[2].set_title('Seed 3: Dirichlet BC')
    axs6[2].set(xlabel='z/h', ylabel='Pitch angle')

    i += 2000
```

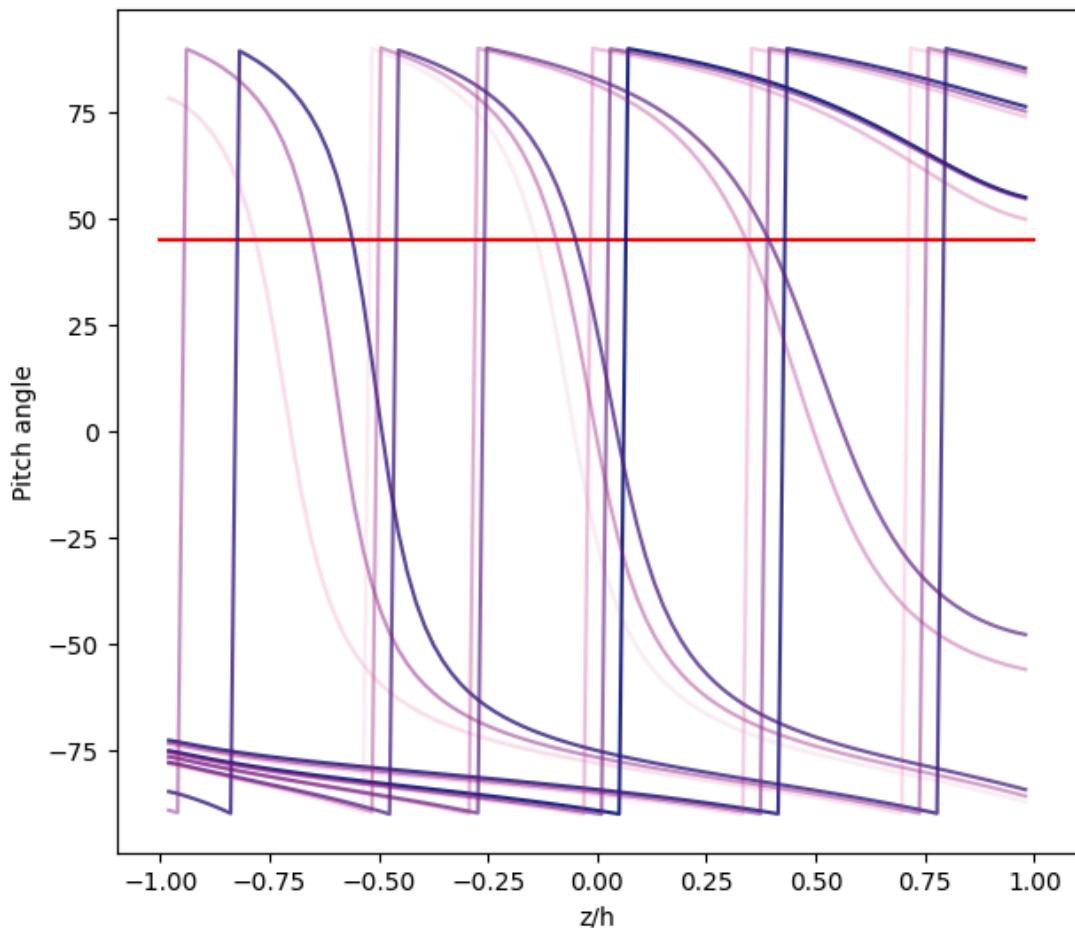
```
/tmp/ipykernel_33745/1302223438.py:2: RuntimeWarning: invalid value encountered in divide
    return 180*np.arctan(Bp/Br)/np.pi
```

Time evolution of pitch angle (deg) for D=-10
(blue to red = early to late times, bright red = seed)





Seed 3: Dirichlet BC



We can see that the pitch angle is not a constant, and that it oscillates with time, and also gets closer to 45 degrees in the case of $D = -6$. If we continue the simulation for longer, we should see the pitch angle slowly saturate to 45 degrees.

In the case of $D = -10$, since the field grows, the pitch angle doesn't necessarily have to saturate. We need to perform a longer simulation to see the effect clearly.

```
# Br_temp_D = np.array([])
# print(Br_temp_D.shape)
# Bp_temp_D = np.array([])
# for j in range(len(D_array)):
#     Br_temp_seed = np.array([])
#     Bp_temp_seed = np.array([])
#     Br_temp1, Bp_temp1 = rk4(Br_diff_eq, Bp_diff_eq[j],
```

```
Br_sols[:, :, -1, :][j][0][0], Bp_sols[:, :, -1, :][j][0][0],
Br_bc[0][0], Br_bc[0][1], Bp_bc[0][0], Bp_bc[0][1], t[-1],
dt, dz, bc = 'dir')
#     Br_temp2, Bp_temp2 = rk4(Br_diff_eq, Bp_diff_eq[j],
Br_sols[:, :, -1, :][j][1][0], Bp_sols[:, :, -1, :][j][1][0],
Br_bc[1][0], Br_bc[1][1], Bp_bc[1][0], Bp_bc[1][1], t[-1],
dt, dz, bc = 'dir')
#     Br_temp3, Bp_temp3 = rk4(Br_diff_eq, Bp_diff_eq[j],
Br_sols[:, :, -1, :][j][2][0], Bp_sols[:, :, -1, :][j][2][0],
Br_bc[2][0], Br_bc[2][1], Bp_bc[2][0], Bp_bc[2][1], t[-1],
dt, dz, bc = 'dir')
#     Br_temp_seed = np.repeat([[[[Br_temp1]]],
[[Br_temp2]], [[Br_temp3]]], 1, axis = 0)
#     Bp_temp_seed = np.repeat([[[[Bp_temp1]]],
[[Bp_temp2]], [[Bp_temp3]]], 1, axis = 0)
#     print(Br_temp_seed.shape)
#     Br_temp_D = np.append(Br_temp_D, [[[Br_temp1]],
[[Br_temp2]], [[Br_temp3]]], axis = 0)
#     # Bp_temp_D[j] = Bp_temp_seed
# print(Br_temp_D.shape)

#####
#####
```