BOUT++ Working Example: Sheath-Driven Instability In Straight Field Geometry

Brett Friedman

Collaborators: T.A. Carter, M.V. Umansky

History and Applications of the Conducting Wall Mode Instability

- H.L. Berk and G.V. Stupakov (1991) showed that a change from insulating to conducting endplates could bring a curvature-driven system from stable to unstable state.

- H.L. Berk, D.D. Ryutov, and Y.A. Tsidulko (1991) showed that conducting endplates in a straight field with an electron temperature gradient transverse to the field produces an instability. Work focused on mirror machine plasmas.

- H.L. Berk, R.H. Cohen, D.D. Ryutov, Y.A. Tsidulko, X.Q. Xu (1993) explained the physical mechanism for the instability. Work focused on tokamak SOL.

- Possible applicability to LAPD led to the model being implemented in BOUT++.

A BOUT++ Example Model: Three Field Conducting Wall Mode Fluid Instability*

Three-Field Model

$$\frac{\partial \varpi}{\partial t} = \nabla_{\|} j_{\|}$$

$$\frac{\partial v_{\|e}}{\partial t} = \nabla_{\|}\phi - 0.51\nu_{ei} v_{\|e}$$

$$\frac{\partial T_e}{\partial t} = -\mathbf{V_E}\cdot\nabla T_e$$

Linearized Parallel Sheath Boundary Conditions

$$j_{\|} = \pm e N_o C_s(\Lambda_1\phi + \Lambda_2 T_e)$$

Theoretical Values: $\Lambda_1 = 1, \Lambda_2 = log\sqrt{\frac{4\pi m_e}{m_i}}$

Transcendental Dispersion Relation

$$\frac{ik_{\|}L\ \tan(k_{\|}L)}{\omega + i\omega_{pe}^2/4\pi\sigma} = \frac{L\frac{m_e}{m_i}}{C_s}\left[\Lambda_1 - \Lambda_2\left(\frac{k_{\perp}C_s^2}{\omega\omega_{ci}L_T}\right)\right]$$

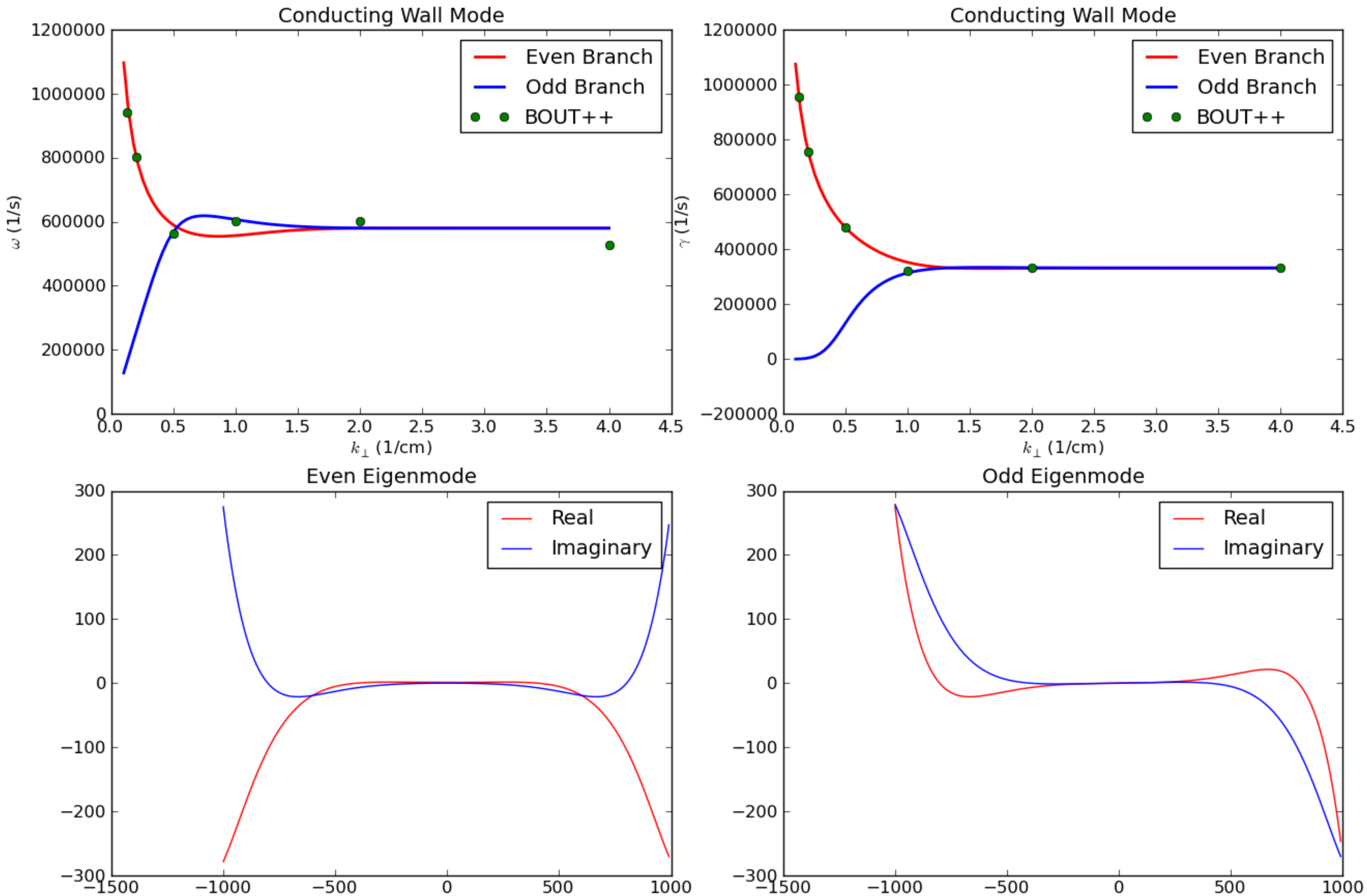
$$\frac{\omega^2}{\omega_{pe}^2} + i\frac{\omega}{4\pi\sigma} = \frac{V_A^2 k_{\|}^2}{c^2 k_{\perp}^2}$$

* M. Umansky notes (based on Berk et. al. papers)

Simple Test Case Comparison Between Numerical Dispersion Relation Solver and BOUT++

Slab geometry, flat density profile, exponential temperature profile

$$\Lambda_1 = 0, \Lambda_2 = 1$$



The BOUT++ Example Model: Simple Conducting Wall Mode Instability

Evolution equations

$$\frac{\partial \varpi}{\partial t} = -\frac{N_{i0}}{\frac{m_e}{m_i}0.51\nu_{ei}}\nabla_{\|}^2\phi$$

$$\frac{\partial T_e}{\partial t} = -\mathbf{v}_E \cdot \nabla T_{e0}$$

Laplace inversion of vorticity

$$\varpi = N_{i0}\nabla_{\perp}^2\phi$$

Sheath boundary conditions

$$\nabla_{\|}\phi = \pm\frac{m_e}{m_i}0.51\nu_{ei}\left(\Lambda_1\phi + \Lambda_2 T_e\right)$$

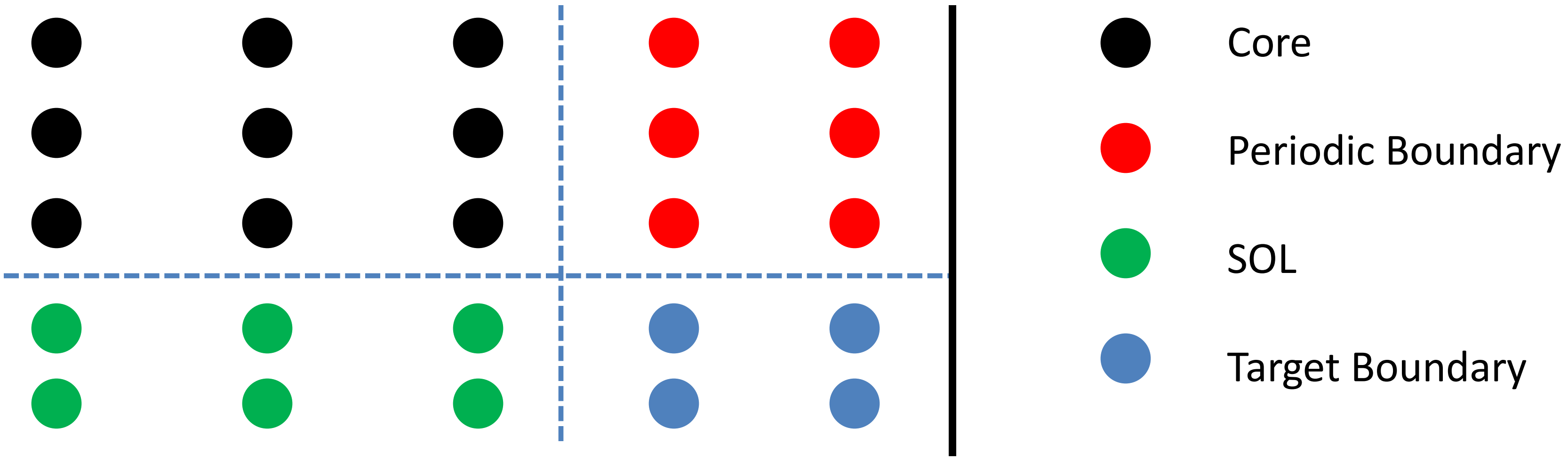
Target down/up

Other boundary conditions

$$\partial_x\varpi = \partial_x T_e = \partial_y\varpi = \partial_y T_e = 0$$

$$\partial_x\phi_{DC,IN} = \phi_{AC,IN} = \phi_{DC,OUT} = \partial_x\phi_{AC,OUT} = 0$$

Looping Over a Boundary Region: Setting the Parallel Gradient to a Value in the UpperY (Target) Region




```
// Boundary gradient to specified Field3D object
void bndry_yup_Grad_par(Field3D &var, const Field3D &value)
{

  RangeIter* xrup = mesh->iterateBndryUpperY();

  for(xrup->first(); !xrup->isDone(); xrup->next())
    for(int jy=mesh->yend+1; jy<mesh->ngy; jy++)
      for(int jz=0; jz<mesh->ngz; jz++) {

        var[xrup->ind][jy][jz] = var[xrup->ind][jy-1][jz]
+ mesh->dy[xrup->ind][jy]*sqrt(mesh->g_22[xrup->ind][jy])*value[xrup->ind][jy][jz];

    }

}
```

Let's look at the code,
run it,
and analyze it

Exercise: Here is the physics_run function. You do the rest.

```
int physics_run(BoutReal t)
{
  // Invert vorticity to get phi
  // Solves \nabla^2_\perp x + (1./c)*\nabla_perp c\cdot\nabla_\perp x + a x = b
  // Arguments are:   (b,   bit-field, a,    c)
  // Passing NULL -> missing term
  phi = invert_laplace(rho/Ni0, phi_flags, NULL, NULL);

  // Communicate variables
  mesh->communicate(comms);

  phi_sheath_bndryconds();

  jpar = -Ni0*ajpar;

  // Evolve rho, te, and ajpar
  ddt(rho) = mesh->Bxy*mesh->Bxy*Div_par_CtoL(jpar);

  ddt(te) = -vE_Grad(Te0, phi);

  //Must propagate phi boundaries into ajpar
  ddt(ajpar) = (1./fmei)*Grad_par_LtoC_wbc(phi) + 0.51*nu*jpar/Ni0;

  // Z filtering
  if(filter_z) {
    // Filter out all except filter_z_mode
    ddt(rho) = filter(ddt(rho), filter_z_mode);
    ddt(te) = filter(ddt(te), filter_z_mode);
    ddt(ajpar) = filter(ddt(ajpar), filter_z_mode);
  }

  return 0;
}
```

$$\mathrm{ajpar} = v_{\|e}$$

In BOUT.inp

[ajpar]

bndry_target = none

Write this function. Use one-sided derivatives in the boundaries.