DEVELOPER TUTORIAL

WESTON ORTIZ

OUTLINE

OUTLINE

Our developer tutorial topics

- Adding an Equation
- Types of Boundary Conditions in Goma
- Adding a Neumann/Robin Condition
- Adding a Material Property
- Adding a Viscosity Model
- Adding a Species Source
- Adding a Post Processing Card

ADDING AN EQUATION

AN EXAMPLE PROBLEM FOR A NEW EQUATION

Due to time constraints we will be looking at adding a simple equation

The Poisson Equation:

$$\Delta u = g$$

Domain is a unit square from (0,0) to (1,1)

Source is $g = -10exp(-((x-0.5)^2 + (y-0.5)^2)/0.02)$

BCs

- $n \cdot \nabla u = f = sin(5x)$ on bottom and top
- Dirichlet $u = 0$ on left and right

Borrowed from Fenics Docs Examples

A NEW EQUATION IN GOMA

Due to time constraints we will be looking at adding a simple equation

The Poisson Equation:

$$\Delta u = g$$

FORMULATION

Before we add equations to Goma we want to think about how we formulate the equations, as an example:

The Poisson Equation has two options a single equation

$$\Delta u = g$$

or a mixed form

$$\nabla \cdot \omega = g$$

$$\nabla q = \omega$$

CHOOSING A FORMULATION

When picking a formulation we might want to consider things like:

- If a formulation has advantageous boundary conditions
- A formulation supports a different solution procedure that is beneficial For simplicity of implementation we will choose the single equation form

$$\Delta u = g$$

WEIGHTED RESIDUAL

Equations in Goma are written in a weighted residual form

We should do this before we start our implementation so we understand what boundary conditions might be available and what we are going to be coding

WEIGHTED RESIDUAL FOR POISSON

First we put our equation in a residual form representing the error. Where $\bar{u}$ is a new variable

$$R(\bar{u}, x_i) = \Delta\bar{u} - g \neq 0$$

Then we can approximate the equation using a weighting function and integrating over the domain

$$\int_{\Omega} w(x_i) R(\bar{u}, x_i) dV = 0$$

WEIGHTED RESIDUAL FOR POISSON

Here we introduce our finite element basis function $\bar{u} = \sum_j \phi_j u_j$:

$$\int_\Omega w(x_i) \nabla \cdot \nabla \bar{u} - w(x_i) g \, dV = 0$$

FEM WEIGHTED RESIDUAL FOR POISSON

We interpolate our field variable using shape functions $\bar{u} = \sum_j \phi_i u_i$: And for Galerkin Finite element we use the same shape functions as weighting functions

$$\int_\Omega \phi_i \nabla \cdot \nabla \bar{u} - \phi_i g \, dV = 0$$

Goma does not support second derivatives using finite element shape functions, instead we integrate by parts

FEM WEIGHTED RESIDUAL FOR POISSON

Field variable using shape functions $\bar{u} = \sum_j \phi_j u_j, \nabla\bar{u} = \sum_j \nabla\phi_j u_j$:

$$\int_\Omega \phi_i \nabla \cdot \nabla\bar{u} - \phi_i g \, dV = 0$$

Use divergence theorem

$$\int v(\nabla \cdot u) dV = \int_{dV} v(n \cdot u) d\Gamma - \int \nabla v \cdot u dV$$

We get

$$-\int_\Omega \nabla\phi_i \cdot \nabla\bar{u} - \phi_i g \, dV + \int_{d\Omega} \phi_i n \cdot \nabla\bar{u} \, d\Gamma = 0$$

FEM WEAK FORM FOR POISSON, INTEGRATION BY PARTS

Now we have our weak form

$$-\int_{\Omega} \nabla\phi_i \cdot \nabla\bar{u} - \phi_i g \, dV + \int_{d\Omega} n \cdot \nabla\bar{u} \, d\Gamma = 0$$

If we omit the boundary term we have a natural boundary condition

$$-\int_{\Omega} \nabla\phi_i \cdot \nabla\bar{u} - \phi_i g \, dV = 0 \implies \int_{d\Omega} \phi_i n \cdot \nabla\bar{u} \, d\Gamma = 0$$

STEPS TO ADD AN EQUATION

Recipe with details available in `mm_fill_shell.c`

Abridged version in these slides

STEP 1: ADD MACROS FOR VARIABLE/EQUATION

`rf_fem_const.h`

Add macros for Poisson variable and equation

Usually we add to the end of the list of equations (might want to search and make sure names are available)

```
#define POISSON <num>
...
#define R_POISSON <num>
#define V_LAST <num+1>
```

STEP 2: ADJUST INITIALIZER LISTS

`mm_names.h`

Add initializations in `EQ_Name`, `Var_Name`, `Exo_Var_Names`, and `Var_Units`

These are important for I/O such as exodus reading/output and special boundary conditions such as GD_PARAB

Try to be consistent, we will choose:

```
"POISSON" "R_POISSON" "U"
```

STEP 3: ENTRIES FOR FIELD VARIABLES

`mm_as_structs.h`

Add entries for the following structs:

```
Element_Stiffness_Pointers // pointers into solution vector
Element_Variable_Pointers // pointers into "other" solutions (x_old)
Field_Variables // Values at current iteration u^n / grad u^n
Diet_Field_Variables // Values at other e.g. u^(n-1) / grad u^(n-1)
```

```
esp->u evp->u fv->u fv->grad_u[DIM] fv->d_grad_u_dmesh[DIM][DIM][MDE]
```

STEP 4: ALLOCATE SPACE

`mm_as_alloc.c`

We need to allocate space for

```
Element_Stiffness_Pointers
```

if the equation is enabled.

STEP 5: LOAD STIFFNESS POINTERS

`mm_fill_ptrs.c`

In `load_elem_dofptr()` add a new call to `load_varType_Interpolation_ptrs()` to populate `esp`

STEP 6: LOAD_VARIABLE

`bc_colloc.c`

In `load_variable()` add a case for your new variable

STEP 7A: EQUATION SPECIFICATION

`mm_input.c`

In `rd_eq_specs()` add `set_eqn()` call for your equation

In `rd_eq_specs()` add `set_var()` call for your variable

These are important for the equation section in goma

```
EQ = momentum1   Q1 U1 Q1        1.   1.   1.   1.   1.  0.
     equation       var          ms  adv  bnd  dif  src  por
```

We will pick `"poisson"` and `"u"`

STEP 7B: EQUATION SPECIFICATION

`mm_input.c`

Decide on equation term multipliers, for Poisson it probably makes sense to have diffusion, boundary, and source as we have 3 terms

```
1 EQ = momentum1   Q1 U1 Q1          1.   1.   1.   1.   1.  0.
2 EQ = poisson     Q1 U  Q1                    1.   1.   1.
3      equation       var          ms  adv  bnd  dif  src  por
```

$$-\int_{\Omega} \nabla\phi_i \cdot \nabla\bar{u} - \phi_i g \, dV + \int_{d\Omega} \phi_i n \cdot \nabla\bar{u} \, d\Gamma = 0$$

STEP 7C: EQUATION SPECIFICATION

`mm_input.c`

```
1 EQ = poisson     Q1 U  Q1                    1.   1.   1.
```

Read in your equation term multipliers in `rd_eq_specs()`

STEP 8: VARIABLE STRING TO INT

`mm_input_util.c`

Add conversion from variable string to your declared variable in `variable_string_to_int()`

```
"POISSON" -> POISSON
```

STEP 9: SET UP PD

`mm_prob_def.c`

Add lines to set member `e` based on etm `setup_pd()`

These are bitwise or's to enable terms, refer to many examples

STEP 10A: LOAD FIELD VARIABLES

`mm_fill_terms.c`

Fill your field variables using `esp load_fv()`

$$\bar{u} = \sum_j \phi_j u_j$$

STEP 10B: LOAD FIELD VARIABLES

`mm_fill_terms.c`

Fill your gradient field variables using `esp`

`load_fv_grad()`

$$\nabla \bar{u} = \sum_j \nabla \phi_j u_j$$

STEP 10C: LOAD FIELD VARIABLES

`mm_fill_terms.c`

Fill your mesh derivative field variables using `esp` `load_fv_mesh_derivs()`

$$\frac{\partial}{\partial d}\nabla\bar{u} = \sum_j \frac{\partial}{\partial d}\nabla\phi_j u_j$$

STEP 11: VARIABLE SENSITIVITY

`mm_flux.c`

Add your new variable to `load_fv_sens()`

STEP 12: INTERACTIONS

`mm_unknown_map.c`

1. In `set_interaction_masks()` add your new variable to any other variables it will have interactions with
2. Add a new case with interactions for your variable
3. Mesh displacements are almost universally expected unless moving mesh will never be used

STEP 12: ADD LSA MESH DERIV MODIFICATIONS

`ac_stability_util.c`

In `modify_fv_mesh_derivs_for_LSA_3D_of_2D()` add your gradient field variable if you added mesh derivatives for a gradient

STEP 13A: ADD A NEW ASSEMBLY FUNCTION FOR YOUR EQUATION

The majority of assembly functions are in `mm_fill_terms.c`

This file is quite bloated and we want to avoid adding to that so lets create new files `src/mm_fill_poisson.c` and `include/mm_fill_poisson.h`

STEP 13B: ADD A NEW ASSEMBLY FUNCTION FOR YOUR EQUATION

Add the new files to the `Makefile`

STEP 13C: ADD A NEW ASSEMBLY FUNCTION FOR YOUR EQUATION

Add a header guard to `mm_fill_poisson.h` to ensure the header is only loaded once

Add an initial prototype for our assembly

```
#ifndef GOMA_MM_FILL_POISSON
#define GOMA_MM_FILL_POISSON

extern int assemble_poisson(void);

#endif // GOMA_MM_FILL_POISSON
```

STEP 13D: ADD A NEW ASSEMBLY FUNCTION FOR YOUR EQUATION

Add assembly function in `mm_fill_poisson.c`

Include header files as needed

```
#include "goma.h" // catch all would prefer
                  // to use required headers instead
                  // but can take a while by hand
#include "mm_as.h" // ei, pd
#include "mm_as_structs.h" // field variables
#include "mm_fill_poisson.h"

int assemble_poisson(void) {

  return 0;
}
```

STEP 13E: ADD A NEW ASSEMBLY FUNCTION FOR YOUR EQUATION

Add Section for Residual assembly in `mm_fill_poisson.c`

```
int assemble_poisson(void) {

  if (af->Assemble_Residual) {

  }
```

$$R_i = R_i - (\nabla\phi_i \cdot \nabla\bar{u} - \phi_i g) g_{wt} d_{vol} h_3$$

$$g = -10exp(-((x-0.5)^2 + (y-0.5)^2)/0.02)$$

STEP 13F: ADD A NEW ASSEMBLY FUNCTION FOR YOUR EQUATION

Add Section for Jacobian assembly in `mm_fill_poisson.c`

```
if (af->Assemble_Jacobian) {
```

$$\bar{u} = \sum_j \phi_j u_j, \ \nabla \bar{u} = \sum_j \nabla \phi_j u_j$$

$$R_i = R_i - (\nabla \phi_i \cdot \nabla \bar{u} - \phi_i g) g_{wt} d_{vol} h_3$$

STEP 13G: ADD ASSEMBLY TO MATRIX FILL

Add a call to your assembly in `mm_fill.c`

```
        if( pde[R_POISSON] ) {
          err = assemble_poisson();
          EH( err, "assemble_poisson");
#ifdef CHECK_FINITE
          err = CHECKFINITE("assemble_poisson");
          if (err) return -1;
#endif
        }
```

TESTING WITH GD BCS

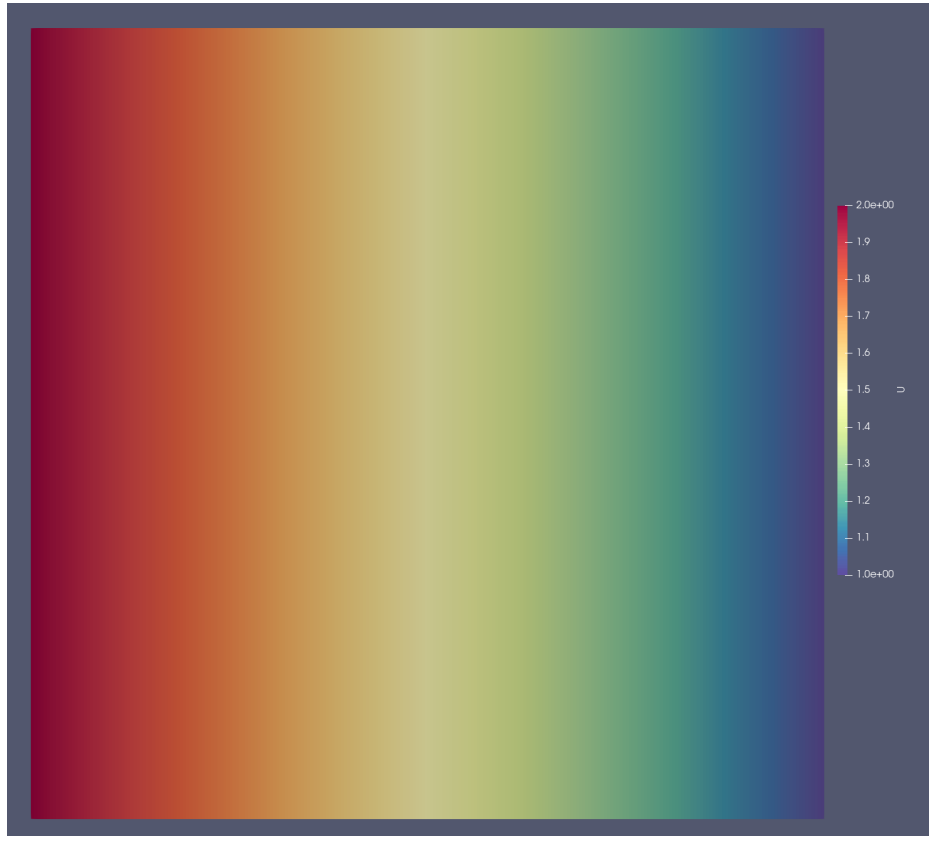
We can actually still test before adding boundary conditions as we have GD BCs which can act as Dirichlet conditions for us.

TEST 1: LAPLACE IN X DIRECTION

In a 1x1 box we use natural boundary conditions on top and bottom $n \cdot \nabla u = 0$

```
# Right SS 2 NS 2
BC = GD_LINEAR SS   2   R_POISSON 0 POISSON     0   -1.0  1.0
# Left SS 4 NS 4
BC = GD_LINEAR SS   4   R_POISSON 0 POISSON     0   -2.0  1.0
...
Number of EQ = -1
# disable source term
EQ = poisson Q1 U Q1                  1.0  1.0  0.0
END OF EQ
                            ms  adv  bnd  dif  src por
```

TEST 1: LAPLACE IN X DIRECTION

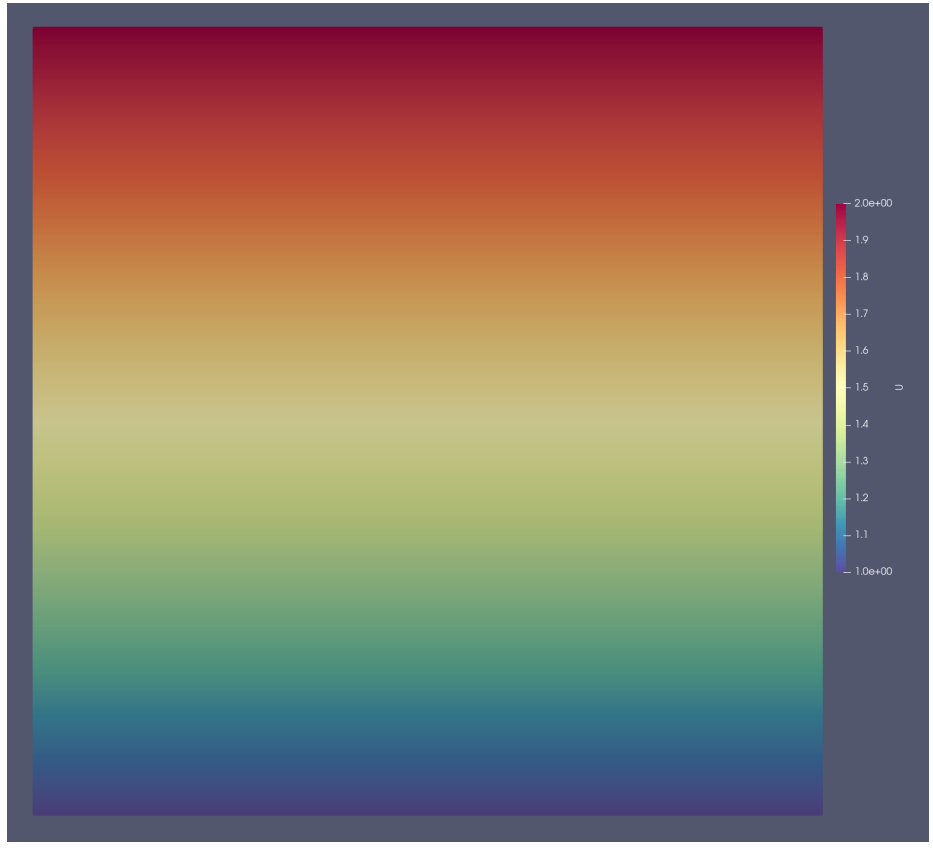


TEST 2: LAPLACE IN Y DIRECTION

In a 1x1 box we use natural boundary conditions on left and right $n \cdot \nabla u = 0$

```
# Top SS 1 NS 1
BC = GD_LINEAR SS   1   R_POISSON 0 POISSON     0   -1.0  1.0
# Top SS 3 NS 3
BC = GD_LINEAR SS   3   R_POISSON 0 POISSON     0   -2.0  1.0
...
Number of EQ = -1
# disable source term
EQ = poisson Q1 U Q1                  1.0  1.0  0.0
END OF EQ
                              ms  adv  bnd  dif  src por
```

TEST 2: LAPLACE IN Y DIRECTION

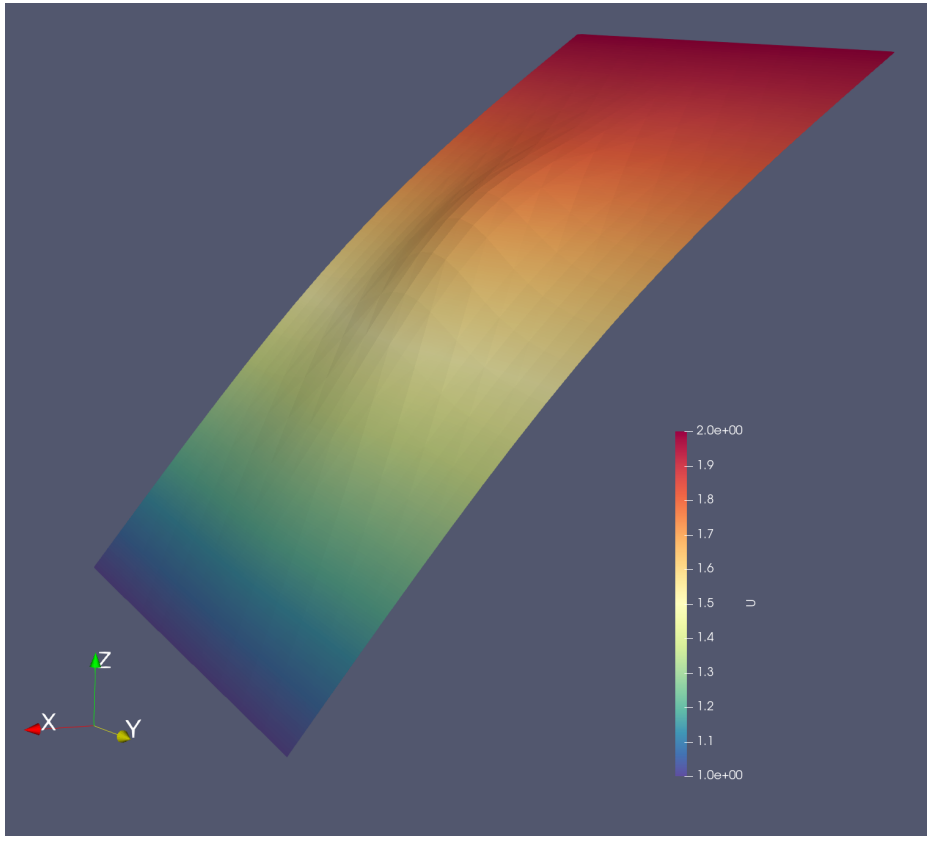


TEST 3: POISSON WITH DIRICHLET

In a 1x1 box we use natural boundary conditions on top and bottom $n \cdot \nabla u = 0$

```
# Right SS 2 NS 2
BC = GD_LINEAR SS   2   R_POISSON 0 POISSON     0   -1.0  1.0
# Left SS 4 NS 4
BC = GD_LINEAR SS   4   R_POISSON 0 POISSON     0   -2.0  1.0
...
Number of EQ = -1
# disable source term
EQ = poisson Q1 U Q1                  1.0  1.0  1.0
END OF EQ
                            ms  adv  bnd  dif  src por
```

TEST 3: POISSON WITH DIRICHLET



STEP 14: ADD NECESSARY BOUNDARY CONDITIONS

This will bring us into our next step in the training

We cannot run equations without boundary conditions unless they are satisfiable with natural boundary conditions

TYPES OF BOUNDARY CONDITIONS IN GOMA

TYPES OF BOUNDARY CONDITIONS IN GOMA

Goma contains a plethora of boundary conditions

- Dirichlet
- Neumann
- Robin
- Many Special cases.

EXAMPLES OF BCS: DIRICHLET

Dirichlet are usually pretty straight forward, we specify the value at a boundary or point

This will be the first BC we usually implement. Examples include:

- U,V,W: Momentum
- T: Energy

EXAMPLES OF BCS: NEUMANN

Neumann BCs are the second type of BCs and we specify the derivative vs. specifying the value.

These are very common in FEM as we usually have an "integration by parts" where the boundary term is in the form $n \cdot \nabla u$

The natural boundary condition is often a Neumann condition where we are specifying the derivative to be zero. Examples:

- QSIDE: Energy
- YFLUX_CONST: Species

EXAMPLES OF BCS: ROBIN

Robin BCs are the third type of BCs and are a combination of Dirichlet and Neumann

Often take a form $n \cdot \nabla u = h(u - u_{ref})$

Examples:

- QCONV: Energy
- YFLUX: Species

STRONG VS WEAK BCS

Goma has a notion of **Strong** and **Weak** boundary conditions

- **Strong**: is applied in a way such that we overwrite contributions of the equation, e.g. VELO_NORMAL, KINEMATIC

$$R_i = \int_{d\Omega} \text{BIG\_PENALTY}(\phi^j n \cdot v - g) = 0$$

- **Weak**: is instead the more typical BC where we are replacing the natural boundary condition with an integrated BC like Neumann or Robin

$$\int_{d\Omega} \phi_i n \cdot \nabla u = \int_{d\Omega} \phi_i f$$

SPECIAL CASES

There are a number of special cases in Goma

- **Embedded**: Applied on a diffuse interface (level-set) e.g. LS_CAPILLARY
- **Collocated**: Strong conditions at nodes, often Dirichlet-like, applied using collocation instead of Galerkin approach e.g. GD_PARAB, KINEMATIC_COLLOC, PLANE
- **Special**: BCs which don't fit in with normal methods, CAP_ENDFORCE
- **Rotated**: BCs which require equations like mesh and momentum to be rotated into normal and tangential coordinate systems, VELO_NORMAL

ADDING A DIRICHLET CONDITION

DIRICHLET CONDITION: STEP 1

Add new constant to `rf_bc_const.h`

```
#define POISSON_BC <uniq num>
```

DIRICHLET CONDITION: STEP 2

Add BC to Initializer list in `mm_names.h`

```
{ "POISSON", "POISSON_BC", DIRICHLET, POISSON_BC,
  R_POISSON, SCALAR, NO_ROT, {0,...0,1}, SINGLE_PHASE,
  DVI_SINGLE_PHASE_DB }
```

DIRICHLET CONDITION: STEP 3

Add BC to Dirichlet BC case in `mm_input_bc.c`

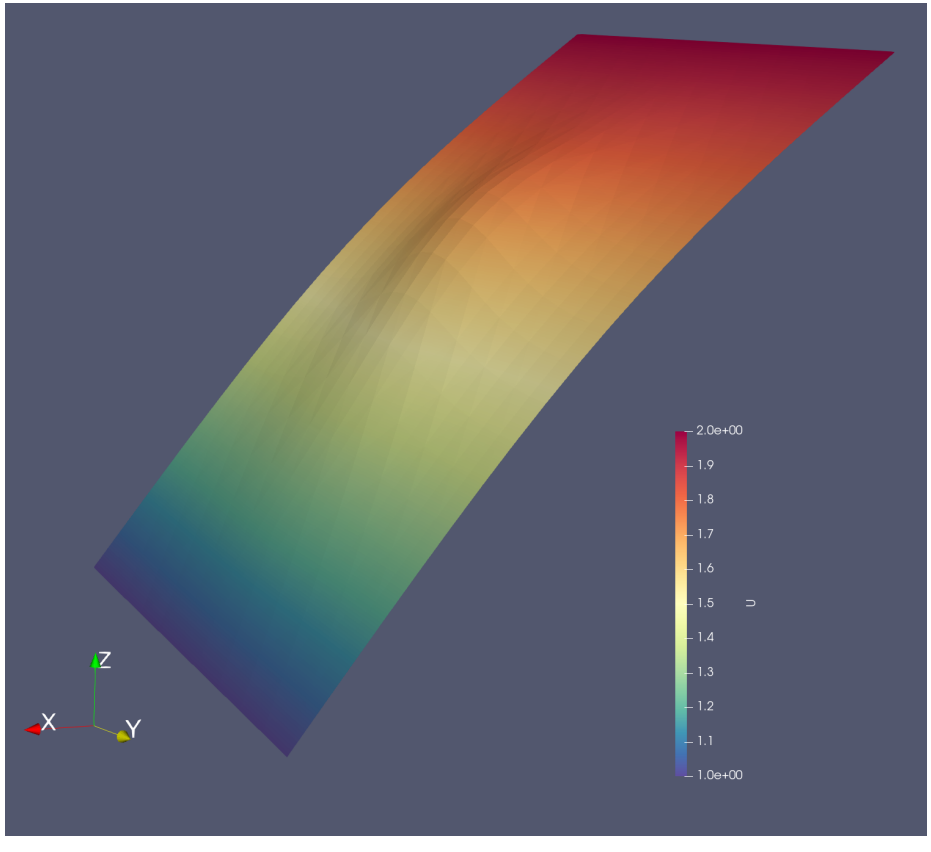
You can now use the Dirichlet BC

TEST DIRICHLET

We can modify our previous Poisson test to now use Dirichlet instead of GD

```
# Right SS 2 NS 2
#BC = GD_LINEAR SS   2   R_POISSON 0 POISSON     0   -1.0  1.0
BC = POISSON NS 2 1.0
# Left SS 4 NS 4
#BC = GD_LINEAR SS   4   R_POISSON 0 POISSON     0   -2.0  1.0
BC = POISSON NS 4 2.0
```

TEST DIRICHLET



ADDING A NEUMANN/ROBIN CONDITION

NEUMANN CONDITION STEP 1

Add new constant to `rf_bc_const.h`

```
#define POISSON_SIDE_SIN_BC <uniq num>
```

NEUMANN CONDITION STEP 2

Add to Initializer list in `mm_names.h`

```
#define POISSON_SIDE_SIN_BC <uniq num>
```

```
{ "POISSON_SIDE_SIN", "POISSON_SIDE_SIN_BC", WEAK_INT_SURF,
   POISSON_SIDE_SIN_BC, R_POISSON, SCALAR, NO_ROT, {0,...0,
   SINGLE_PHASE, DVI_SINGLE_PHASE_DB }
```

NEUMANN CONDITION STEP 3

Read input in `mm_input_bc.c`

We want to read five floats $\alpha, \beta, \gamma, \omega, \zeta$

$$n \cdot \nabla u = \alpha sin(\beta x + \gamma y + \omega z) + \zeta$$

NEUMANN CONDITION STEP 4

Make a new BC function in `mm_fill_poisson.c`

$$n \cdot \nabla u = \alpha sin(\beta x + \gamma y + \omega z) + \zeta$$

```
void
poisson_side_sin_bc(dbl func[DIM],
dbl d_func[DIM][MAX_VARIABLE_TYPES + MAX_CONC][MDE],
dbl alpha, dbl beta, dbl gamma, dbl omega, dbl zeta) {...}
```

NEUMANN CONDITION STEP 5

Add declaration to header `mm_fill_poisson.h`

```
void
poisson_side_sin_bc(dbl func[DIM],
dbl d_func[DIM][MAX_VARIABLE_TYPES + MAX_CONC][MDE],
dbl alpha, dbl beta, dbl gamma, dbl omega, dbl zeta);
```

NEUMANN CONDITION STEP 6

Add call to switch for BC in `bc_integ.c`

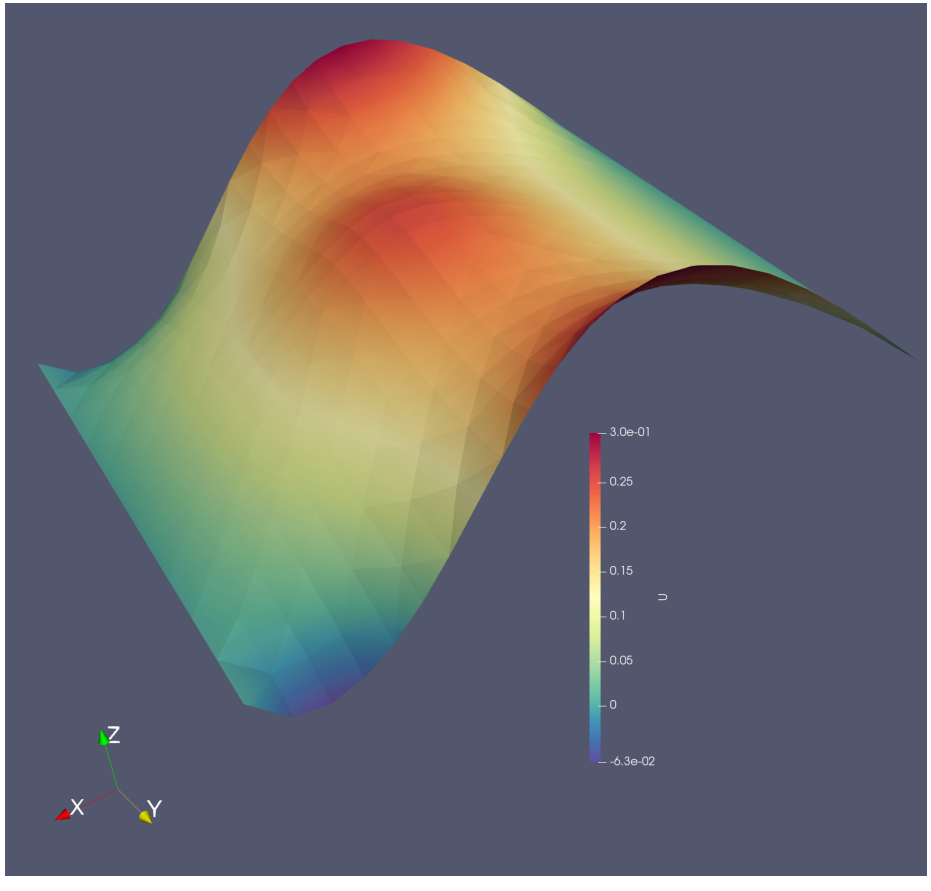
You can now use your Neumann BC

TEST NEUMANN CONDITION

Top and bottom we want $n \cdot \nabla = sin(5x)$, left and right $u = 0$

```
# Bottom SS 1 NS 1
BC = POISSON_SIDE_SIN SS 1 1.0 5.0 0.0 0.0 0.0
# Right SS 2 NS 2
BC = POISSON NS 2 0.0
# Top SS 3 NS 3
BC = POISSON_SIDE_SIN SS 3 1.0 5.0 0.0 0.0 0.0
# Left SS 4 NS 4
BC = POISSON NS 4 0.0
```

TEST NEUMANN CONDITION



NOTES ON ADDING OTHER BOUNDARY CONDITIONS

There are many examples, usually the first thing you should look for is a near-by example.

Adding a **Strong** condition should be the same as Neumann only now you are using **STRONG_INT_SURF** in `mm_names.h`, they are still called in `apply_integrated_bc()`

Other BCs should follow a similar process but may not be added to `bc_integ.c` but `bc_colloc.c` or one of the other BC files

ADDING A MATERIAL PROPERTY

ADDING A MATERIAL PROPERTY

Material properties are added so that we do not have to hard code values in our assemblies.

For our Poisson equation we will add a `PoissonSource` card which lets us read in either a constant (which would be useful to set to 0 for Laplace Equation) and our exponential source term

STEP 1: MATERIAL PROPERTY

`mm_mp_structs.h`

Add needed values to `struct Material_Properties`

```
dbl poisson_source;
int poissonSourceModel;
int len_u_poisson_source;
dbl *u_poisson_source;
```

STEP 2: MATERIAL PROPERTY

`dp_vif.c`

Add communication of the material properties to the various functions

There are many examples

If you have an allocation make sure that is communicated and allocated on all processors

STEP 3: MATERIAL PROPERTY

`mm_mp_const.h`

Add new constant for your model

```
// CONSTANT is 1 so we do not want to use that to different
#define POISSON_EXP 0
```

STEP 4A: MATERIAL PROPERTY

`mm_input_mp.c`

Read in your new material property

By default `look_for_mat_prop()` will look for CONSTANT properties so we only need to add special cases

STEP 4B: MATERIAL PROPERTY

mm_input_mp.c

Source is $g = \alpha exp(-((x-\beta)^2 + (y-\gamma)^2 + (z-\omega)^2))/\zeta)$ so we want 5 constants $\alpha, \beta, \gamma, \omega, \zeta$

```
if ( !strcmp(model_name, "EXP") )
  {
  poissonSourceModel = POISSON_EXP;
  model_read = 1;
  mat_ptr->poissonSourceModel = PoissonSourceModel;
  num_const = read_constants(imp, &(mat_ptr->u_poisson_source),
                             NO_SPECIES);
  if (num_const != 5) {
    EH(-1, "Expected 5 constants for Poisson Source = EXP");
  }
```

STEP 5: MATERIAL PROPERTY

Finally you should be able to integrate your new property into your function

Make sure to check the Model type that was read in

We will add a new function to `mm_fill_poisson.c`

```
//double poisson_source(POISSON_DEPENDENCE_STRUCT *d_source);
double poisson_source(void);
```

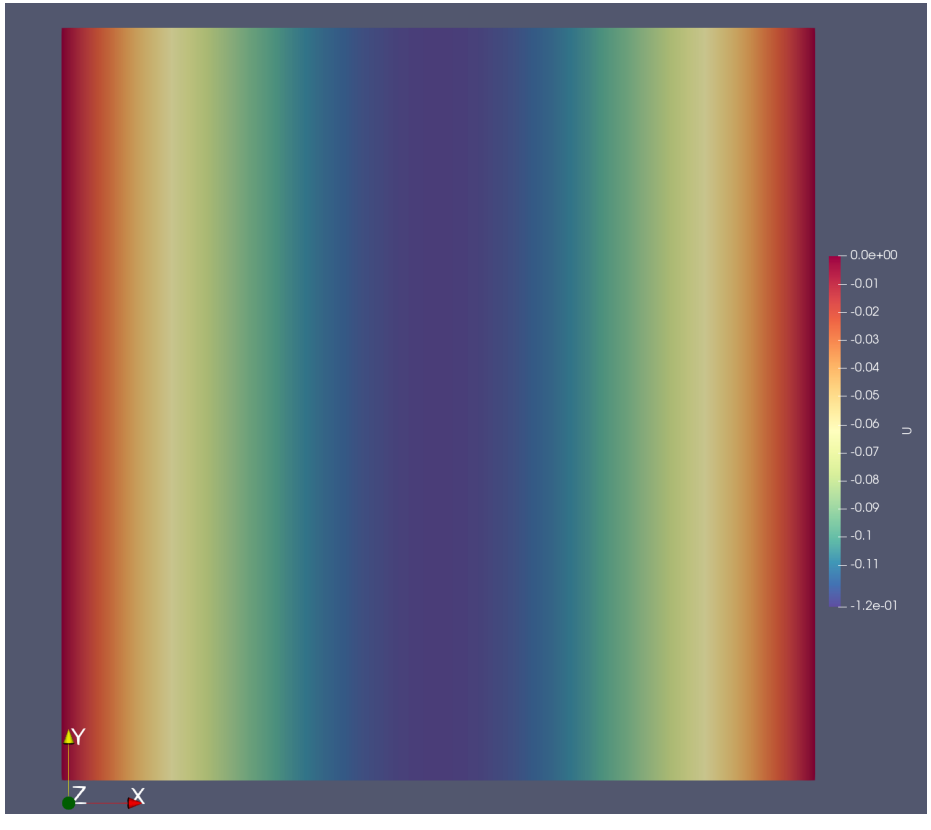
TEST MATERIAL PROPERTY

Test with a constant source

```
Poisson Source = CONSTANT 1.0
```

```
# Right
BC = POISSON NS 2 0.0
# Left
BC = POISSON NS 4 0.0
```

TEST MATERIAL PROPERTY



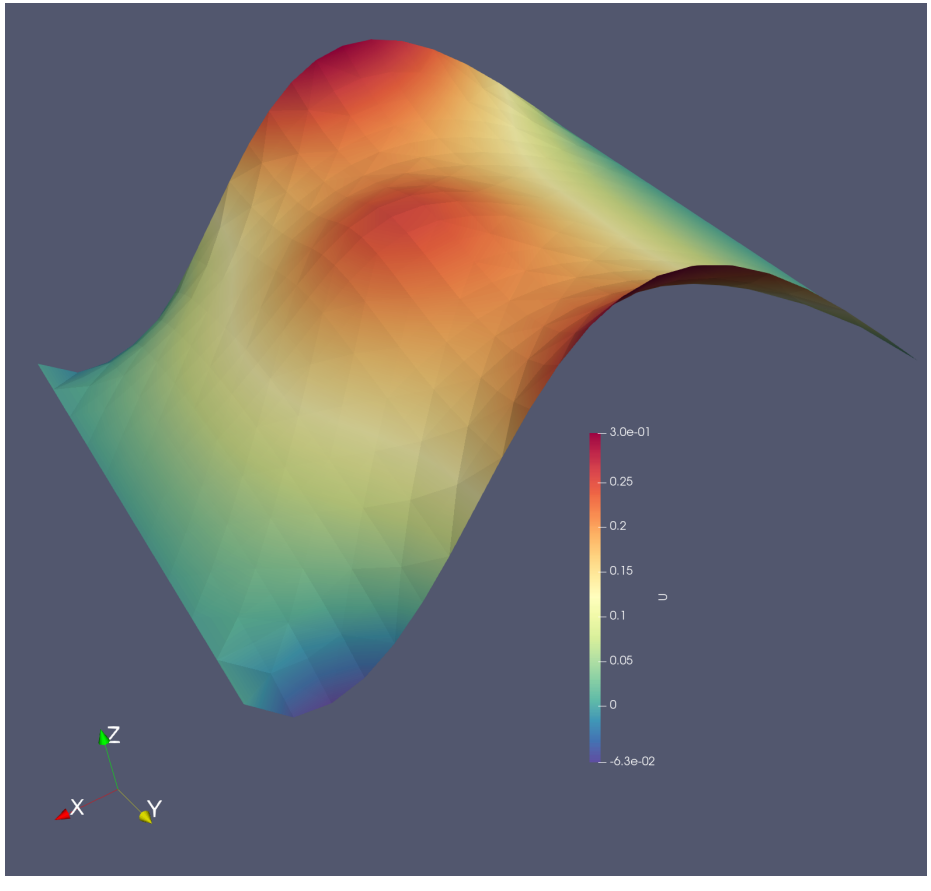
TEST 2 MATERIAL PROPERTY

Test with a constant source

```
Poisson Source = EXP -10.0 0.5 0.5 0.0 0.02
```

```
BC = POISSON_SIDE_SIN SS 1 1.0 5.0 0.0 0.0 0.0
BC = POISSON NS 2 0.0
BC = POISSON_SIDE_SIN SS 3 1.0 5.0 0.0 0.0 0.0
BC = POISSON NS 4 0.0
```

TEST 2 MATERIAL PROPERTY



ADDING A VISCOSITY MODEL

ADDING A VISCOSITY MODEL

Viscosity Models are typically added for Generalized Newtonian Fluids, though in some cases they could be added for Viscoelastic models with non-constant viscosity, or special cases where viscosity depends on more than the shear rate

Examples of models: Goma docs on Liquid Constitutive Equation

ADDING A VISCOSITY MODEL

For our example today we are going to add a 4-parameter Cross model

$$\mu(\dot{\gamma}) = \mu_{\text{inf}} + \frac{\mu_0 - \mu_{\text{inf}}}{1 + (\lambda\dot{\gamma})^{1-n}}$$

STEP 1: ADDING A VISCOSITY MODEL

`mm_mp_const.h`

Add a new define for your Constitutive Model

```
#define CROSS_VISCOSITY <uniq num>
```

STEP 2A: ADDING A VISCOSITY MODEL

`mm_input_mp.c`

Read in parameters for your new model

```
else if (!strcmp(model_name, "CROSS"))
```

STEP 2A: ADDING A VISCOSITY MODEL

`mm_input_mp.c`

Read in parameters for your new model, add to cases

$$\mu(\dot{\gamma}) = \mu_{\text{inf}} + \frac{\mu_0 - \mu_{\text{inf}}}{1 + (\lambda\dot{\gamma})^{1-n}}$$

```
"Low Rate Viscosity"
"Power Law Exponent"
"High Rate Viscosity"
"Time Constant"
```

STEP 3: ADDING A VISCOSITY MODEL

`mm_viscosity.c`

Add a new function to calculate your viscosity

```
dbl
cross_viscosity(struct Generalized_Newtonian *gn_local,
                dbl gamma_dot[DIM][DIM], /* strain rate */
                VISCOSITY_DEPENDENCE_STRUCT *d_mu )
```

$$\mu(\dot{\gamma}) = \mu_{\text{inf}} + \frac{\mu_0 - \mu_{\text{inf}}}{1 + (\lambda\dot{\gamma})^{1-n}}$$

STEP 4: ADDING A VISCOSITY MODEL

`mm_viscosity.c`

Add your viscosity to the `viscosity()` function

```
else if (gn_local->ConstitutiveEquation == CROSS_VISCOSITY)
  {
    mu = cross_viscosity(gn_local, gamma_dot, d_mu);
  }
```

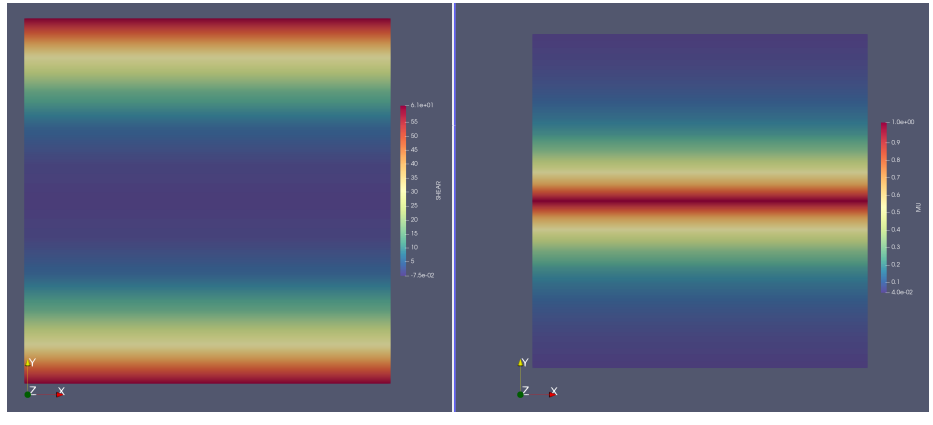
You should now be able to use your new model

TEST VISCOSITY MODEL

$$\mu(\dot{\gamma}) = \mu_{\text{inf}} + \frac{\mu_0 - \mu_{\text{inf}}}{1 + (\lambda\dot{\gamma})^{1-n}}$$

```
Liquid Constitutive Equation    = CROSS
Low Rate Viscosity              = CONSTANT      1.0
Power Law Exponent              = CONSTANT      0.33
High Rate Viscosity             = CONSTANT      1e-3
Time Constant                   = CONSTANT      2.0
```

TEST VISCOSITY MODEL



ADDING A SPECIES SOURCE

SPECIES EQUATIONS

Species equations are a special case in Goma

They offer a way to use any number of generic advection-diffusion-reaction equations in addition to offering some useful tools for managing concentrations

$$\frac{\partial c}{\partial t} + u \cdot \nabla c + \nabla \cdot D \nabla c = S$$

SPECIES SOURCE EXAMPLE

For our example we will add two sources for the Schnakenberg reactive model

$$\frac{\partial u}{\partial t} + w \cdot \nabla u - \nabla^2 u = \gamma(a - u + u^2 v)$$

$$\frac{\partial v}{\partial t} + w \cdot \nabla v - d\nabla^2 v = \gamma(b - u^2 v)$$

STEP 1: ADDING A SPECIES SOURCE

mm_mp_const.h

```
#define SCHNAKENBERG_U <uniq num>
#define SCHNAKENBERG_V <uniq num>
```

STEP 2: ADDING A SPECIES SOURCE

`mm_input_mp.c`

Read in species source parameters, we can read in γ, a for the u equation and b for the v equation.

$$\frac{\partial u}{\partial t} + w \cdot \nabla u - \nabla^2 u = \gamma(a - u + u^2 v)$$

$$\frac{\partial v}{\partial t} + w \cdot \nabla v - d\nabla^2 v = \gamma(b - u^2 v)$$

STEP 3: ADDING A SPECIES SOURCE

`mm_std_models.c`

Add new functions to calculate your species sources, should be quite a few examples available

```
double schnakenberg_u_source(int species_no, double *derivatives);
double schnakenberg_v_source(int species_no, double *derivatives);
```

STEP 4: ADDING A SPECIES SOURCE

`get_continuous_species_terms()`

Usually use another species source as an example

```
else if (mp-<SpeciesSourceModel[w]  == SCHNAKENBERG_U )
  {

  }
```

SCHNAKENBERG TEST PROBLEM

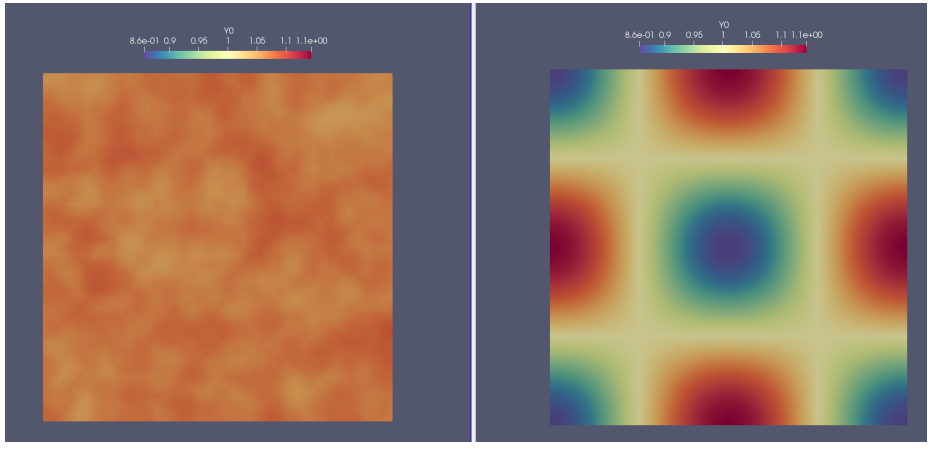
We've initialized a mesh with $a = 0.1, b = 0.9, d = 8.6676, \gamma = 230.82$:

$$(u, v) = \left(a + b + \epsilon(a + b), \frac{b}{(a + b)^2} + \epsilon \frac{b}{(a + b)^2} \right)$$

Where ϵ is a random perturbation of max 10% varying at nodes

Ref: https://www.sciencedirect.com/science/article/pii/S0307904X11008298

SCHNAKENBERG TEST PROBLEM

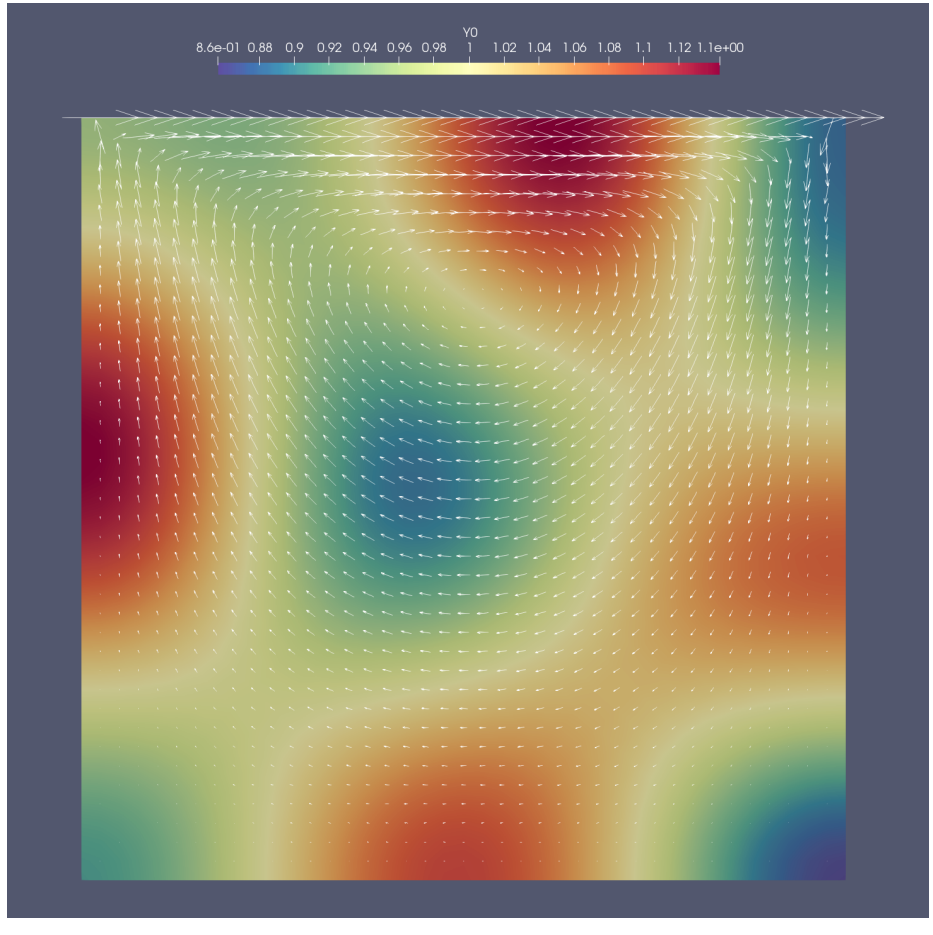


SCHNAKENBERG TEST PROBLEM

We can add flow with a lid driven cavity, using the previous solution as an initial guess

Now we can see that we get the advection for free as well when using species

SCHNAKENBERG TEST PROBLEM



ADDING A POST PROCESSING CARD

STEP 1: ADDING A POST PROCESSING CARD

`mm_post_def.h`

Add declaration of new variable

```
extern int TOTAL_SPECIES;
```

STEP 2: ADDING A POST PROCESSING CARD

`mm_post_proc.c`

Add instantiation of new variable

```
int TOTAL_SPECIES;
```

STEP 3: ADDING A POST PROCESSING CARD

`mm_post_proc.c`

Read input for new variable in

`rd_post_process_specs()`

```
iread = look_for_post_proc(ifp, "Total Species", &TOTAL_SPECIES, '\n');
```

STEP 4: ADDING A POST PROCESSING CARD

`mm_post_proc.c`

Setup variable in exodus II database

`load_nodal_tkn()`

```
if ( TOTAL_SPECIES != -1 && Num_Var_In_Type[MASS_FRACTION])
   {
      set_nv_tkud(rd, index, 0, 0, "TY","[1]", "TOTAL_SPECIES", FALSE);
      index++;
      TOTAL_SPECIES = index_post;
      index_post++;
```

STEP 5: ADDING A POST PROCESSING CARD

`dp_vif.c`

Add flag to distributed communication

```
ddd_add_member(n, &TOTAL_SPECIES, 1, MPI_INT);
```

STEP 5: ADDING A POST PROCESSING CARD

`mm_post_proc.c`

Add algorithm to calculate post proc value

`post_process_nodal()`

```
if (TOTAL_SPECIES != -1 && pd->e[MASS_FRACTION] ){
  double total = 0;
  for (int i = 0; i < pd->Num_Species; i++) {
    total += fv->c[i];
  }
  local_post[TOTAL_SPECIES] = total;
  local_lumped[TOTAL_SPECIES] = 1.;
}
```