ENERGY
www.csiro.au

Numbat

High-resolution simulations of density-driven convective mixing in porous media

Christopher Green

Report Number EPXXXXXX
June, 2018

CSIRO Energy
71 Normanby Road, Clayton VIC, 3168, Australia
Private Bag 10, Clayton South VIC, 3169, Australia
Telephone: +61 3 9545 2777
Fax: +61 3 9545 8380

Copyright and disclaimer

Important disclaimer

Contents

List of Figures

1 Numbat

Numbat is an open source MOOSE[1] application for high-resolution simulations of buoyancy-driven convection in porous media in both two and three dimensions.

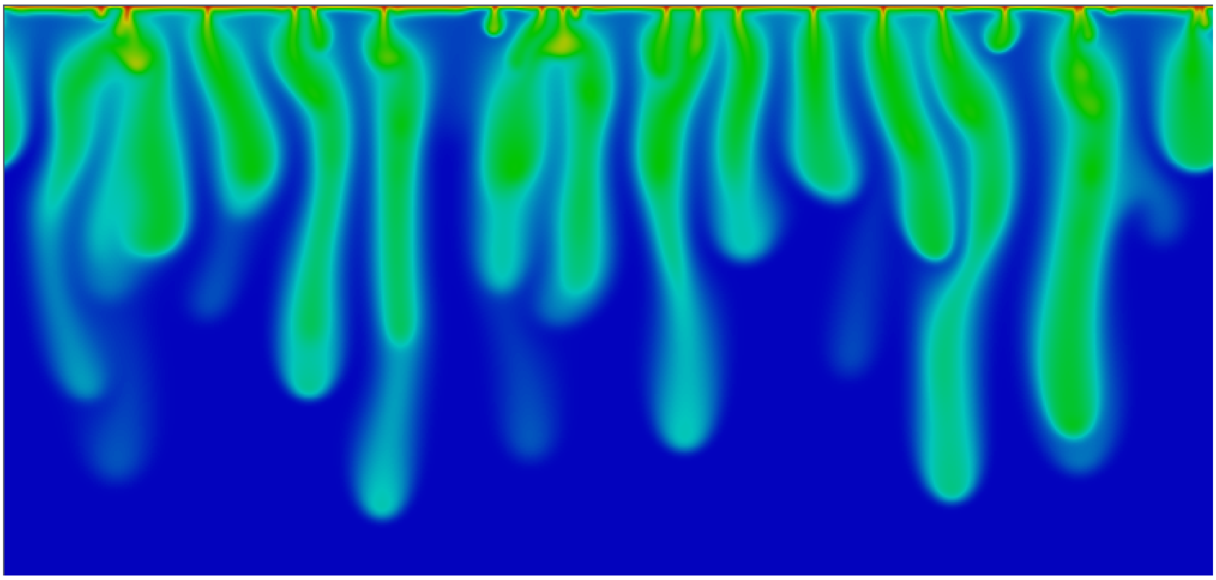

Figure 1.1: Density-driven convective mixing in porous media

As a MOOSE app, it provides access to powerful MOOSE features such as adaptive mesh refinement, hybrid parallelism, both continuous and discontinuous Galerkin methods, and much more, all wrapped in a simple interface.

Numat solves the coupled convection-diffusion and Darcy equations with the Boussinesq approximation using the finite element method.

Several formulations are available: from the full, dimensional governing equations, to a dimensionless streamfunction formulation.

Numbat is developed on GitHub[2] by CSIRO[3].

[1]www.mooseframework.org
[2]www.github.com/cpgr/numbat
[3]www.csiro.au

2 Introduction

Convective mixing of fluids in porous media is a physical process that manifests in complex flow patterns. In solutal convective mixing, the presence of a fluid component in the background fluid results in a density contrast between fluid containing the solute and fluid without. This density difference gives rise to a pressure gradient which drives motion.

Recently, a significant body of scientific literature has been concerned with density-driven convective mixing in porous media due to its applicability to geological storage of carbon dioxide, see Emami-Meybodi *et al.* (2015) for an extensive literature review. In the case where CO_2 is injected into a saline aquifer with a bounding cap rock, buoyancy drives vertical migration of the mobile CO_2 (CO_2 in the supercritical gas phase), which then spreads beneath the cap rock to form a thin, laterally extensive plume. In time, the gaseous CO_2 begins to dissolve into the local formation water, leading to a small increase in density of the saturated brine at the top of the aquifer of approximately 1%. Diffusion of the dissolved CO_2 allows further dissolution, a process that leads to a gravitational instability whereby a denser fluid lies atop a less dense one. After a sufficient period of time, vertical fluid motion is induced as vertical acceleration overcomes diffusion, and CO_2-rich water descends to the lower part of the reservoir.

The process of convective mixing can significantly increase the rate of dissolution of CO_2 in the formation water and hence reduce the amount of mobile CO_2. This can significantly reduce the risk of leakage into overlying aquifers, increasing the security of storage.

3 Background theory

3.1 Governing equations

Numbat implements the Boussinesq approximation to model density-driven convective mixing in porous media. To reduce the computational burden, only a single fluid phase is considered. This is a simplification to the actual physical process, where a gas phase may be present. This simplification is often used in practice, see Emami-Meybodi *et al.* (2015) for a discussion about the use of this simplifying assumption.

Note:

The more complicated two-phase model can be implemented using the `porous_flow` module

The governing equations for density-driven flow in porous media are Darcy's law

$$\mathbf{u} = -\frac{\mathbf{K}}{\mu}\left(\nabla P - \rho(c) g \hat{\mathbf{k}}\right), \tag{3.1}$$

where $\mathbf{u} = (u, v, w)$ is the velocity vector, $\mathbf{K}$ is permeability, μ is the fluid viscosity, P is the fluid pressure, $\rho(c)$ is the fluid density as a function of solute concentration c, g is gravity, and $\hat{\mathbf{k}}$ is the unit vector in the z direction.

The fluid velocity must also satisfy the continuity equation

$$\nabla \cdot \mathbf{u} = 0, \tag{3.2}$$

and the solute concentration is governed by the convection - diffusion equation

$$\phi \frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \phi D \nabla^2 c, \tag{3.3}$$

where ϕ is the porosity, t is time and D is the diffusivity.

Darcy's law and the convection-diffusion equations are coupled through the fluid density, which is given by

$$\rho(c) = \rho_0 + \frac{c}{c_0}\Delta\rho, \tag{3.4}$$

where c_0 is the equilibrium concentration, and $\Delta\rho$ is the increase in density of the fluid at equilibrium concentration.

The boundary conditions are

$$\begin{aligned} w &= 0, \quad z = 0, -H, \\ \frac{\partial c}{\partial z} &= 0, \quad z = -H, \\ c &= c_0, \quad z = 0, \end{aligned} \tag{3.5}$$

which correspond to impermeable boundary conditions at the top and bottom boundaries, given by $z = 0$ and $z = -H$, respectively, and a saturated condition at the top boundary.

Initially, there is no solute in the model

$$c = 0, \quad t = 0. \tag{3.6}$$

Numbat solves Eq. (3.1) and Eq. (3.3) with density coupled to concentration as in Eq. (3.4).

3.2 Dimensionless formulation

The governing equations can also be solved using a streamfunction formulation in 2D and a vector potential formulation in 3D. As a result, we shall consider the two cases separately.

3.2.1 2D solution

If we consider an anisotropic model, with vertical and horizontal permeabilities given by k_z and k_x, respectively, we can non-dimensionalise the governing equations in 2D following Ennis-King & Paterson (2005) Defining the anisotropy ratio γ as

$$\gamma = \frac{k_z}{k_x}, \tag{3.7}$$

we scale the variables using

$$\begin{gathered} x = \frac{\phi \mu D}{k_z \Delta \rho g \gamma^{1/2}} \hat{x}, \quad z = \frac{\phi \mu D}{k_z \Delta \rho g} \hat{z}, \quad u = \frac{k_z \Delta \rho g}{\mu \gamma^{1/2}} \hat{u}, \quad w = \frac{k_z \Delta \rho g}{\mu} \hat{w} \\ t = \left(\frac{\phi \mu}{k_z \Delta \rho g} \right)^2 \hat{t}, \quad c = c_0 \hat{c}, \quad P = \frac{\mu \phi D}{k_z} \hat{P}, \end{gathered} \tag{3.8}$$

where $\hat{x}$ refers to a dimensionless variable. The governing equations in dimensionless form are then

$$\mathbf{u} = -\left(\nabla P + c \hat{\mathbf{k}} \right), \tag{3.9}$$

$$\mathbf{u} = 0, \tag{3.10}$$

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \gamma \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial z^2}, \tag{3.11}$$

where we have dropped the hat on the dimensionless variables for brevity.

The dimensionless boundary conditions are

$$\begin{aligned} w &= 0, \quad z = 0, -Ra, \\ \frac{\partial c}{\partial z} &= 0, \quad z = -Ra, \\ c &= 1, \quad z = 0, \end{aligned} \tag{3.12}$$

where Ra is the Rayleigh number, defined as

$$Ra = \frac{k_z \Delta\rho g H}{\phi \mu D}. \tag{3.13}$$

In this form, the Rayleigh number only appears in the boundary conditions as the location of the lower boundary. Therefore, Ra can be interpreted in this formalism as a dimensionless model height, and can be varied in simulations by simply changing the height of the mesh.

Finally, the dimensionless initial condition is

$$c = 0, \quad t = 0. \tag{3.14}$$

For isotropic models, where $k_x = k_z$ and hence $\gamma = 1$, we recover the dimensionless equations given by Slim (2014)

The coupled governing equations must be solved numerically. To simplify the numerical analysis, we introduce the streamfunction $\psi(x, z, t)$ such that

$$u = -\frac{\partial \psi}{\partial z}, \quad w = \frac{\partial \psi}{\partial x}. \tag{3.15}$$

This definition satisfies the continuity equation, Eq. (3.10), immediately.

The pressure P is removed from Eq. (3.9) by taking the curl of both sides and noting that $\nabla \times \nabla P = 0$ for any P, to give

$$\nabla^2 \psi = -\frac{\partial c}{\partial x}, \tag{3.16}$$

where we have introduced the streamfunction ψ using Eq. (3.15).

The convection-diffusion equation, Eq. (3.11) becomes

$$\frac{\partial c}{\partial t} - \frac{\partial \psi}{\partial z}\frac{\partial c}{\partial x} + \frac{\partial \psi}{\partial x}\frac{\partial c}{\partial z} = \gamma \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial z}. \tag{3.17}$$

The boundary conditions become

$$\begin{aligned} \frac{\partial \psi}{\partial x} &= 0, \quad z = 0, -Ra, \\ \frac{\partial c}{\partial z} &= 0, \quad z = -Ra, \\ c &= 1, \quad z = 0, \end{aligned} \tag{3.18}$$

while the initial condition is still given by Eq. (3.14).

In two dimensions, Numbat solves Eq. (3.16) and Eq. (3.17).

3.2.2 3D solution

We now consider the case of a three-dimensional model. For simplicity, we consider the case where all lateral permeabilities are equal ($k_y = k_x$). The governing equations for the 3D model are identical to the 2D model. In dimensionless form, they are given by Eq. (3.9) to Eq. (3.11), with boundary conditions given by Eq. (3.12), and initial condition given by Eq. (3.14).

To solve these governing equations in 3D, a different approach must be used as the streamfunction ψ is not defined in three dimensions. Instead, we define a vector potential $\Psi = (\psi_x, \psi_y, \psi_z)$ such that

$$\mathbf{u} = \nabla \times \Psi. \tag{3.19}$$

It is important to note that the vector potential is only known up to the addition of the gradient of a scalar ζ as

$$\nabla \times (\Psi + \nabla\zeta) = \nabla \times \Psi \quad \forall\zeta, \tag{3.20}$$

as $\nabla \times \nabla\zeta = 0$ for any scalar ζ. This uncertainty is referred to as guage freedom, and is common in electrodynamics. Taking the curl of Eq. (3.9) and substituting Eq. (3.19), we have

$$\nabla(\nabla \cdot \Psi) - \nabla^2\Psi = \left(-\frac{\partial c}{\partial y}, \frac{\partial c}{\partial x}, 0\right), \tag{3.21}$$

where we have again used the fact that $\nabla \times \nabla P = 0$. If we choose $\nabla \cdot \Psi = 0$ to specify the guage condition, this simplifies to

$$\nabla^2\Psi = \left(\frac{\partial c}{\partial y}, -\frac{\partial c}{\partial x}, 0\right). \tag{3.22}$$

As shown in E & Liu (1997) $\nabla \cdot \Psi = 0$ is satisfied throughout the domain if

$$\begin{aligned} \psi_x = \psi_y &= 0, \quad z = 0, -Ra, \\ \frac{\partial \psi_z}{\partial z} &= 0, \quad z = 0, -Ra. \end{aligned} \tag{3.23}$$

The governing equations are then

$$\nabla^2\Psi = \left(\frac{\partial c}{\partial y}, -\frac{\partial c}{\partial x}, 0\right), \tag{3.24}$$

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \gamma\left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2}\right) + \frac{\partial^2 c}{\partial z^2}, \tag{3.25}$$

where the continuity is satisfied automatically because $\nabla \cdot (\nabla \times \Psi) = 0$ for any Ψ.

Finally, it is straightforward to show that $\psi_z = 0$ in order to satisfy $\nabla^2\psi_z = 0$ and $\frac{\partial \psi_z}{\partial z} = 0$, which means that the vector potential has only x and y components,

$$\Psi = (\psi_x, \psi_y, 0), \tag{3.26}$$

and therefore the fluid velocity $\mathbf{u} = (u, v, w)$ is

$$\mathbf{u} = \left(-\frac{\partial \psi_y}{\partial z}, \frac{\partial \psi_x}{\partial z}, \frac{\partial \psi_y}{\partial x} - \frac{\partial \psi_x}{\partial y}\right). \tag{3.27}$$

Note that if there is no y dependence, Eq. (3.24) and (3.25) reduce to

$$\begin{aligned} \nabla^2 \Psi &= \left(0, -\frac{\partial c}{\partial x}, 0\right), \\ \frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c &= \gamma \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial z^2}. \end{aligned} \tag{3.28}$$

It is simple to show that $\nabla^2 \psi_x = 0$ and $\psi_x = 0$ at $z = 0, -Ra$ are only satisfied if $\psi_x = 0$ in the entire domain. In this case, the governing equations reduce to the two-dimensional formulation, as expected.

In three dimensions, Numbat solves Eq. (3.24) and Eq. (3.25).

4 Installation instructions

To install Numbat, follow these simple instructions.

4.1 Install MOOSE

Numbat is based on the MOOSE framework, so the first step is to install MOOSE. For detailed installation instructions depending on your hardware, see www.mooseframework.com.

4.2 Clone Numbat

The next step is to clone the Numbat repository to your local machine.

In the following, it is assumed that MOOSE was installed to the directory `~/projects`. If MOOSE was installed to a different directory, the following instructions must be modified accordingly.

To clone Numbat, use the following commands

```
cd ~/projects
git clone https://github.com/cpgr/numbat.git
cd numbat
git checkout master
```

4.3 Compile Numbat

Next, compile Numbat using

```
make -jn
```

where *n* is the number of processing cores on the computer. If everything has gone well, Numbat should compile without error, producing a binary named `numbat-opt`.

4.4 Test Numbat

Finally, to test that the installation worked, the test suite can be run using

```
./run_tests -jn
```

where *n* is the number of processing cores on the computer. At this stage, all of the Numbat tests should have run successfully, and you are ready to run more complicated simulations, see the 2D examples and 3D examples for more details.

4.5 Keep up to date

New features and changes to Numbat may be implemented from time to time. To ensure that Numbat continues to work without issue, you should make sure that you update your installation periodically. This can be done using

```
git fetch origin
git rebase origin/master
make -jn
./run_tests -jn
```

5 Implementation details

Numerical simulations of solutal convection rely on seeding the gravitational instability to initiate the convective mixing process. Numerical roundoff will eventually initiate convective mixing, but can significantly overestimate the critical time for the onset of convection.

Instead, it is usual to seed the instability using a prescribed random perturbation in the model. In Numbat, this initial perturbation can be applied in several ways:

- In the dimensional formulation, a random noise sampled from a uniform distribution can be applied to the porosity;
- In the dimensionless streamfunction formulation, an initial perturbation to the diffusive concentration profile can be applied;
- A random fluctuation to the concentration boundary condition can be applied.

6 Input file syntax

The input file for a Numbat simulation is a simple, block-structured text file based on the input files for a plain MOOSE input file.

6.1 Essential input

Details of the minimum input file requirements are given below.

6.1.1 Mesh

All simulations must feature a mesh. For the basic model with a rectangular mesh, the built-in NumbatBiasedMesh can be used to create a suitable mesh. This is a type of GeneratedMesh that provides the option to refine the mesh near one boundary. The size of the initial element can be specified, after which the elements are progressively coarser, see NumbatBiasedMesh for details. This can be extremely useful in simulations of density-driven convection, where it is necessary to have a finer mesh in the vicinity of the boundary where the fingers form in order to capture the process adequately. Away from this region, the fingers grow and merge, so that a coarser mesh is sufficient to simulate them. Having a biased mesh such as that produced by NumbatBiasedMesh can minimise the number of elements necessary, reducing the overall computational demands.

In 2D, the input block looks like:

```
[Mesh]
  type = NumbatBiasedMesh
  dim = 2
  ymax = 1.5
  nx = 100
  ny = 50
  refined_edge = top
  refined_resolution = 0.001
[]
```

In 3D, the Mesh block would look like:

```
[Mesh]
  type = NumbatBiasedMesh
  dim = 3
  zmax = 1.5
  nx = 20
  ny = 20
  nz = 500
  refined_edge = front
  refined_resolution = 0.001
[]
```

Note:

In contrast to the normal GeneratedMesh provided by MOOSE, NumbatBiasedMesh renames the boundaries of the three dimensional mesh so that the boundaries `top` and `bottom` are at the extrema of the `z` axis.

6.1.2 Variables

The number and type of variables required depends on the chosen formulation. For the dimensional formulation, two nonlinear variables must be provided, representing the fluid pressure and solute concentration.

```
[Variables]
  [./concentration]
    initial_condition = 0
    scaling = 1e6
  [../]
  [./pressure]
    initial_condition = 1e6
  [../]
[]
```

For the dimensionless streamfunction formulation, the nonlinear variables for a 2D simulations are solute concentration and streamfunction:

```
[Variables]
  [./concentration]
    order = FIRST
    family = LAGRANGE
  [../]
  [./streamfunction]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
[]
```

In 3D, an additional streamfunction variable must also be added:

```
[Variables]
  [./concentration]
    order = FIRST
    family = LAGRANGE
    [./InitialCondition]
      type = NumbatPerturbationIC
      variable = concentration
      amplitude = 0.1
      seed = 1
    [../]
  [../]
  [./streamfunctionx]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
  [./streamfunctiony]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
[]
```

6.1.3 Materials

For the dimensional formulation, several material and fluid properties are required: porosity, permeability, fluid density and viscosity, and diffusivity. These can be added using the following Numbat materials:

```
[Materials]
  [./porosity]
    type = NumbatPorosity
    porosity = 0.3
    noise = noise
  [../]
  [./permeability]
    type = NumbatPermeability
    permeability = '1e-11 0 0 0 1e-11 0 0 0 1e-11'
  [../]
  [./diffusivity]
    type = NumbatDiffusivity
    diffusivity = 2e-9
  [../]
  [./density]
    type = NumbatDensity
    concentration = concentration
    zero_density = 995
    delta_density = 10.5
    saturated_concentration = 0.049306
  [../]
  [./viscosity]
    type = NumbatViscosity
    viscosity = 6e-4
  [../]
[]
```

Note:

No material properties are required in the dimensionless streamfunction formulation

6.1.4 Kernels

Four kernels are required for a dimensional model: NumbatTimeDerivative, NumbatDiffusion, NumbatConvection, and NumbatDarcy.

```
[Kernels]
  [./time]
    type = NumbatTimeDerivative
    variable = concentration
  [../]
  [./diffusion]
    type = NumbatDiffusion
    variable = concentration
  [../]
  [./convection]
    type = NumbatConvection
    variable = concentration
    pressure = pressure
  [../]
  [./darcy]
```

```
    type = NumbatDarcy
    variable = pressure
    concentration = concentration
  [../]
[]
```

For the dimensionless streamfunction formulation, four kernels are required for a 2D model: a NumbatDarcySF kernel, a NumbatDiffusionSF kernel, a NumbatConvectionSF kernel, and a TimeDerivative kernel.

```
[Kernels]
  [./Darcy]
    type = NumbatDarcySF
    variable = streamfunction
    concentration = concentration
  [../]
  [./Convection]
    type = NumbatConvectionSF
    variable = concentration
    streamfunction = streamfunction
  [../]
  [./Diffusion]
    type = NumbatDiffusionSF
    variable = concentration
  [../]
  [./TimeDerivative]
    type = TimeDerivative
    variable = concentration
  [../]
[]
```

For 3D models, an additional NumbatDarcySF kernel is required for the additional streamfunction variable. An example of the kernels block for a 3D isotropic model is

```
[Kernels]
  [./Darcy_x]
    type = NumbatDarcySF
    variable = streamfunctionx
    concentration = concentration
    component = x
  [../]
  [./Darcy_y]
    type = NumbatDarcySF
    variable = streamfunctiony
    concentration = concentration
    component = y
  [../]
  [./Convection]
    type = NumbatConvectionSF
    variable = concentration
    streamfunction = 'streamfunctionx streamfunctiony'
  [../]
  [./Diffusion]
    type = NumbatDiffusionSF
    variable = concentration
  [../]
  [./TimeDerivative]
```

```
    type = TimeDerivative
    variable = concentration
  [../]
[]
```

In the 3D case, it is important to note that the NumbatDarcySF kernel must specify the component that it applies to, and that the `streamfunction` keyword in the NumbatConvectionSF kernel must contain both streamfunction variables ordered by the `x` component then the `y` component.

Note:

For the streamfunction formulation, a TimeDerivative kernel is used, rather than a NumbatTimeDerivative kernel, as porosity has been scaled out of the problem.

6.1.5 Boundary conditions

Appropriate boundary conditions must be prescribed. Typically, these will be constant concentration at the top of the model domain, periodic boundary conditions on the lateral sides (to mimic an infinite model), and no-flow boundary conditions at the top and bottom surfaces.

In the 2D dimensional formulation, this can be achieved using the following input block:

```
[BCs]
  [./conctop]
    type = PresetBC
    variable = concentration
    boundary = top
    value = 0.049306
  [../]
  [./Periodic]
    [./x]
      variable = concentration
      auto_direction = x
    [../]
  [../]
[]
```

while in 3D

```
[BCs]
  [./conctop]
    type = PresetBC
    variable = concentration
    boundary = front
    value = 0.049306
  [../]
  [./Periodic]
    [./x]
      variable = concentration
      auto_direction = 'x y'
    [../]
  [../]
[]
```

In this case, the `conctop` boundary condition fixes the value of concentration at the top boundary, while the `Periodic` boundary condition enforces periodicity of concentration along the boundaries in the directions specified in the `auto_direction` parameter.

It is useful to note that a MOOSE GeneratedMesh provides descriptive names for the sides of the model (top, bottom, left, right) which can be referenced in the input file.

For the dimensionless streamfunction formulation, no-flow boundary conditions are prescribed on the top and bottom surfaces by holding the `streamfunction` variable constant (in this case 0).

```
[BCs]
  [./conctop]
    type = DirichletBC
    variable = concentration
    boundary = top
    value = 1.0
  [../]
  [./streamfuntop]
    type = DirichletBC
    variable = streamfunction
    boundary = top
    value = 0.0
  [../]
  [./streamfunbottom]
    type = DirichletBC
    variable = streamfunction
    boundary = bottom
    value = 0.0
  [../]
  [./Periodic]
    [./x]
      variable = 'concentration streamfunction'
      auto_direction = x
    [../]
  [../]
[]
```

6.1.6 Executioner

Each MOOSE simulation must use an `Executioner`, which provides parameters for the solve.

```
[Executioner]
  type = Transient
  l_max_its = 200
  end_time = 3e5
  solve_type = NEWTON
  petsc_options = -ksp_snes_ew
  petsc_options_iname = '-pc_type -sub_pc_type -ksp_atol'
  petsc_options_value = 'bjacobi ilu 1e-12'
  nl_abs_tol = 1e-10
  nl_max_its = 25
  dtmax = 2e3
  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 1
  [../]
[]
```

Executioners are a standard MOOSE feature that are well documented on the MOOSE, so no further detail is provided here.

6.1.7 Preconditioning

A default preconditioning block is used that provides all Jacobian entries to aid convergence.

```
[Preconditioning]
  [./smp]
    type = SMP
    full = true
  [../]
[]
```

This is a standard MOOSE feature that is documented on the MOOSE website, so no further detail is provided here.

6.1.8 Outputs

To provide ouptut from the simulation, an `Outputs` block must be specified. An example is

```
[Outputs]
  [./console]
    type = Console
    perf_log = true
    output_nonlinear = true
  [../]
  [./exodus]
    type = Exodus
    file_base = 2Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
  [./csvoutput]
    type = CSV
    file_base = 2Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
[]
```

There are a large number of output options available in MOOSE, see the MOOSE website for further details.

6.2 Action system

To avoid having to enter several of these input file blocks each time, and ensuring that the correct parameters are provided to each object in the correct order, Numbat provides some powerful actions that automatically add most of the required objects.

The NumbatAction adds all of the nonlinear variables, kernels, aux variables, aux kernels and postprocessors typically required in a dimensional Numbat simulation.

This action is called in the input file simply as

```
[Numbat]
  [./Dimensional]
  [../]
[]
```

The use of this action is exactly equivalent to the following input file syntax

```
[Variables]
  [./concentration]
    initial_condition = 0
  [../]
  [./pressure]
    initial_condition = 1e6
  [../]
[]

[AuxVariables]
  [./u]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./v]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]

[Kernels]
  [./time]
    type = NumbatTimeDerivative
    variable = concentration
  [../]
  [./diffusion]
    type = NumbatDiffusion
    variable = concentration
  [../]
  [./convection]
    type = NumbatConvection
    variable = concentration
    pressure = pressure
  [../]
  [./darcy]
    type = NumbatDarcy
    variable = pressure
    concentration = concentration
  [../]
[]

[AuxKernels]
  [./uAux]
    type = NumbatDarcyVelocity
    pressure = pressure
    variable = u
    component = x
  [../]
  [./vAux]
    type = NumbatDarcyVelocity
    pressure = pressure
    variable = v
    component = y
  [../]
[]
```

```
[BCs]
  [./conctop]
    type = DirichletBC
    variable = concentration
    boundary = top
    value = 1.0
  [../]
  [./Periodic]
    [./x]
      variable = 'concentration pressure'
      auto_direction = x
    [../]
  [../]
[]

[Postprocessors]
  [./boundary_flux]
    type = NumbatSideFlux
    variable = concentration
    boundary = top
  [../]
  [./total_mass]
    type = NumbatTotalMass
    variable = concentration
  [../]
[]
```

A specific value for the saturated boundary concentration can optionally be provided

```
[Numbat]
  [./Dimensional]
    boundary_concentration = 0.05
  [../]
[]
```

Similarly, the NumbatSFAction adds all of the nonlinear variables, kernels, aux variables, aux kernels and postprocessors typically required in a dimensionless Numbat simulation.

This action is called in the input file simply as

```
[Numbat]
  [./Dimensionless]
  [../]
[]
```

The use of this action is exactly equivalent to the following input file syntax for a 2D simulation.

```
[Variables]
  [./concentration]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
  [./streamfunction]
    order = FIRST
    family = LAGRANGE
    initial_condition = 0.0
  [../]
```

```
[]

[AuxVariables]
  [./u]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./v]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]

[Kernels]
  [./Darcy]
    type = NumbatDarcySF
    variable = streamfunction
    concentration = concentration
  [../]
  [./Convection]
    type = NumbatConvectionSF
    variable = concentration
    streamfunction = streamfunction
  [../]
  [./Diffusion]
    type = NumbatDiffusionSF
    variable = concentration
  [../]
  [./TimeDerivative]
    type = TimeDerivative
    variable = concentration
  [../]
[]

[AuxKernels]
  [./uAux]
    type = NumbatDarcyVelocitySF
    variable = u
    component = x
    streamfunction = streamfunction
  [../]
  [./vAux]
    type = NumbatDarcyVelocitySF
    variable = v
    component = y
    streamfunction = streamfunction
  [../]
[]

[BCs]
  [./conctop]
    type = DirichletBC
    variable = concentration
    boundary = top
    value = 1.0
  [../]
```

```
  [./streamfuntop]
    type = DirichletBC
    variable = streamfunction
    boundary = top
    value = 0.0
  [../]
  [./streamfunbottom]
    type = DirichletBC
    variable = streamfunction
    boundary = bottom
    value = 0.0
  [../]
  [./Periodic]
    [./x]
      variable = 'concentration streamfunction'
      auto_direction = x
    [../]
  [../]
[]

[Postprocessors]
  [./boundary_flux]
    type = NumbatSideFluxSF
    variable = concentration
    boundary = top
  [../]
  [./total_mass]
    type = NumbatTotalMassSF
    variable = concentration
  [../]
[]
```

The use of these actions is recommended for all users, as they reduce the possibility of input file errors.

6.3 Optional input

While the above required blocks will enable a Numbat simulation to run, there are a number of optional input blocks that will improve the simulations are increase the amount of results provided.

6.3.1 Mesh adaptivity

MOOSE features built-in mesh adaptivity that is extremely useful in Numbat simulations to reduce computational expense. This can be included using:

```
[Adaptivity]
  max_h_level = 1
  initial_marker = boxmarker
  initial_steps = 1
  marker = errormarker
  [./Indicators]
    [./gradjumpindicator]
      type = GradientJumpIndicator
      variable = concentration
    [../]
  [../]
```

```
  [./Markers]
    [./errormarker]
      type = ErrorToleranceMarker
      refine = 0.05
      indicator = gradjumpindicator
    [../]
    [./boxmarker]
      type = BoxMarker
      bottom_left = '0 0 -10'
      top_right = '500 500 0'
      inside = refine
      outside = dont_mark
    [../]
  [../]
[]
```

For details about mesh adaptivity, see the MOOSE website.

6.3.2 Initial condition

To seed the instability, a random perturbation to the initial concentration can be prescribed using the NumbatPerturbationIC initial condition.

```
[ICs]
  [./concentration]
    type = NumbatPerturbationIC
    variable = concentration
    amplitude = 0.1
    seed = 1
  [../]
[]
```

The NumbatPerturbationIC initial condition applies the diffusive concentration profile to the nodes for (scaled) time $t = 1$,

$$c_d(z, t = 1) = 1 + \mathrm{erf}(z/2), \tag{6.1}$$

for $z < 0$, where $\mathrm{erf(z)}$ is the error function.

A uniform random perturbation is then added to the diffusive concentration profile, where the amplitude of the perturbation is specified by the input file value `amplitude`.

6.3.3 Postprocessors

The flux over the top boundary or the total mass of solute in the model is of particular interest in many cases (especially convective mixing of CO_2). These can be calculated at each time step using the NumbatSideFlux and NumbatTotalMass Postprocessors.

```
[Postprocessors]
  [./boundaryfluxint]
    type = NumbatSideFlux
    variable = concentration
    boundary = top
  [../]
  [./mass]
    type = NumbatTotalMass
    variable = concentration
```

```
  [../]
[]
```

Versions of these Postprocessors for the dimensionless streamfunction formulation are also provided, see NumbatSideFluxSF and NumbatTotalMassSF for details.

Numbat also provides a simple Postprocessor to calculate the Rayleigh number for dimensional simulations, see NumbatRayleighNumber for details.

6.3.4 AuxKernels

The velocity components in the x and y directions (in 2D), and x, y, and z directions in 3D can be calculated using the auxiliary system. These velocity components are calculated using the streamfunction(s), see the governing equations for details.

In the 2D case, two auxiliary variables, u and w, can be defined for the horizontal and vertical velocity components, respectively.

Note:

Importantly, these auxiliary variables **must** have monomial shape functions (these are referred to as elemental variables, as the value is constant over each mesh element). This restriction is due to fact that the gradient of variables is undefined for nodal auxiliary variables (for example, those using linear Lagrange shape functions).

An example of the input syntax for the 2D case is

```
[AuxVariables]
  [./u]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./w]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]
```

For the 3D case, there is an additional horizontal velocity component (v), so the input syntax is

```
[AuxVariables]
  [./u]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./v]
    order = CONSTANT
    family = MONOMIAL
  [../]
  [./w]
    order = CONSTANT
    family = MONOMIAL
  [../]
[]
```

The velocity components are calculated by NumbatDarcyVelocity AuxKernels (or NumbatDarcyVelocitySF AuxKernels for the dimensionless streamfunction formulation). Each velocity component is computed by an AuxKernel.

For the 2D case, two AuxKernels are required:

```
[AuxKernels]
  [./uAux]
    type = NumbatDarcyVelocitySF
    variable = u
    component = x
    streamfunction = streamfunction
  [../]
  [./wAux]
    type = NumbatDarcyVelocitySF
    variable = w
    component = y
    streamfunction = streamfunction
  [../]
[]
```

while for 3D, three AuxKernels are necessary:

```
[AuxKernels]
  [./uAux]
    type = NumbatDarcyVelocitySF
    variable = u
    component = x
    streamfunction = 'streamfunctionx streamfunctiony'
  [../]
  [./vAux]
    type = NumbatDarcyVelocitySF
    variable = v
    component = y
    streamfunction = 'streamfunctionx streamfunctiony'
  [../]
  [./wAux]
    type = NumbatDarcyVelocitySF
    variable = w
    component = z
    streamfunction = 'streamfunctionx streamfunctiony'
  [../]
[]
```

Note:

For the 3D case, both streamfunction variables must be given, in the correct order (eg. x then y)

7 Running the Numbat application

7.1 Commandline

Most often, Numbat will be run from the commandline using

```
./numbat-opt -i input.i
```

Numbat (and all MOOSE applications) have a large number of commandline options available to the user. The complete list can be viewed using the `--help` option

```
./numbat-opt --help
```

7.1.1 Recovering

If you output checkpoint files (using `checkpoint = true` in your Outputs block) then the `--recover` option will allow you to continue a solve that died in the middle of the solve. This can allow you to recover a job that was killed because the power went out or your job ran out of time on the cluster you were using.

We recommend that all input files for large Numbat simulations enable checkpointing. This can be enabled using

```
[Outputs]
  [./console]
    type = Console
    perf_log = true
    output_nonlinear = true
  [../]
  [./csvoutput]
    type = CSV
    file_base = 2DSF
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
  [./checkpoint]
    type = Checkpoint
    num_files = 2
  [../]
[]
```

For all of the options available for checkpointing, see the MOOSE documentation.

If a long-running simulation does fail to complete, it can be recovered by

```
./numbat-opt --recover checkpoint_dir/XXXX -i input.i
```

where `checkpoint_dir` is the subdirectory where the checkpoint files are saved, and `XXXX` is the number of one of the available checkpoint files.

7.1.2 Overriding parameters

MOOSE provides a handy feature where any parameter in the input file can be overridden from the commandline, making it possible to script studies where only parameters are changed from simulation to simulation.

For example, assume that the anisotropy `gamma` of the porous medium is set as `1` in the input file

```
[Kernels]
  [./Darcy]
    type = NumbatDarcySF
    variable = streamfunction
    concentration = concentration
    gamma = 1
  [../]
[]
```

This value can be changed to `0.5` by running Numbat with the following commandline option

```
./numbat-opt -i input.i Kernels/Darcy/gamma=0.5
```

7.2 Graphical user interface (Peacock)

MOOSE provides a graphical user interface, Peacock, which can be used to both run simulations and create input files. Starting Peacock from within the base Numbat directory allows Peacock to extract the Numbat syntax, so that all Numbat objects are available in the menus.

> **Note:**
>
> It is not recommended to use Peacock to run very large models (e.g. three dimensional simulations)

8 Two-dimensional examples

Complete input files for 2D models using the dimensional and dimensionless streamfunction formulations are provided, for both isotropic and anisotropic porous media. These examples are provided in the Numbat *examples* folder.

8.1 Isotropic models

The first 2D examples are for an isotropic porous medium ($\gamma = 1$).

8.1.1 Input file

The complete input file for this problem is

```
# 2D density-driven convective mixing. Instability is seeded by small
   perturbation
# to porosity. Takes about 5 minutes to run using a single processor.

[Mesh]
  type = NumbatBiasedMesh
  dim = 2
  ymax = 1.5
  nx = 100
  ny = 50
  refined_edge = top
  refined_resolution = 0.001
[]

[Variables]
  [./concentration]
    initial_condition = 0
    scaling = 1e6
  [../]
  [./pressure]
    initial_condition = 1e6
  [../]
[]

[AuxVariables]
  [./noise]
    family = MONOMIAL
    order = CONSTANT
  [../]
[]

[ICs]
  [./noise]
    type = RandomIC
    variable = noise
    max = 0.003
    min = -0.003
  [../]
[]

[Materials]
  [./porosity]
    type = NumbatPorosity
```

```
    porosity = 0.3
    noise = noise
  [../]
  [./permeability]
    type = NumbatPermeability
    permeability = '1e-11 0 0 0 1e-11 0 0 0 1e-11'
  [../]
  [./diffusivity]
    type = NumbatDiffusivity
    diffusivity = 2e-9
  [../]
  [./density]
    type = NumbatDensity
    concentration = concentration
    zero_density = 995
    delta_density = 10.5
    saturated_concentration = 0.049306
  [../]
  [./viscosity]
    type = NumbatViscosity
    viscosity = 6e-4
  [../]
[]

[Kernels]
  [./time]
    type = NumbatTimeDerivative
    variable = concentration
  [../]
  [./diffusion]
    type = NumbatDiffusion
    variable = concentration
  [../]
  [./convection]
    type = NumbatConvection
    variable = concentration
    pressure = pressure
  [../]
  [./darcy]
    type = NumbatDarcy
    variable = pressure
    concentration = concentration
  [../]
[]

[BCs]
  [./conctop]
    type = PresetBC
    variable = concentration
    boundary = top
    value = 0.049306
  [../]
  [./Periodic]
    [./x]
      variable = concentration
      auto_direction = x
```

```
    [../]
  [../]
[]

[Preconditioning]
  [./smp]
    type = SMP
    full = true
  [../]
[]

[Executioner]
  type = Transient
  l_max_its = 200
  end_time = 3e5
  solve_type = NEWTON
  petsc_options = -ksp_snes_ew
  petsc_options_iname = '-pc_type -sub_pc_type -ksp_atol'
  petsc_options_value = 'bjacobi ilu 1e-12'
  nl_abs_tol = 1e-10
  nl_max_its = 25
  dtmax = 2e3
  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 1
  [../]
[]

[Postprocessors]
  [./boundaryfluxint]
    type = NumbatSideFlux
    variable = concentration
    boundary = top
  [../]
  [./mass]
    type = NumbatTotalMass
    variable = concentration
  [../]
[]

[Outputs]
  [./console]
    type = Console
    perf_log = true
    output_nonlinear = true
  [../]
  [./exodus]
    type = Exodus
    file_base = 2Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
  [./csvoutput]
    type = CSV
    file_base = 2Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
```

```
[]
```

8.1.2 Running the example

This example can be run on the commandline using

```
numbat-opt -i 2Dddc.i
```

Alternatively, this file can be run using the *Peacock* gui provided by MOOSE using

```
peacock -i 2Dddc.i
```

in the directory where the input file *2Dddc.i* resides.

8.1.3 Results

This 2D example should take only a few minutes to run to completion, producing a concentration profile similar to that presented in Figure 8.1, where several downwelling plumes of high concentration can be observed after 3528 s:

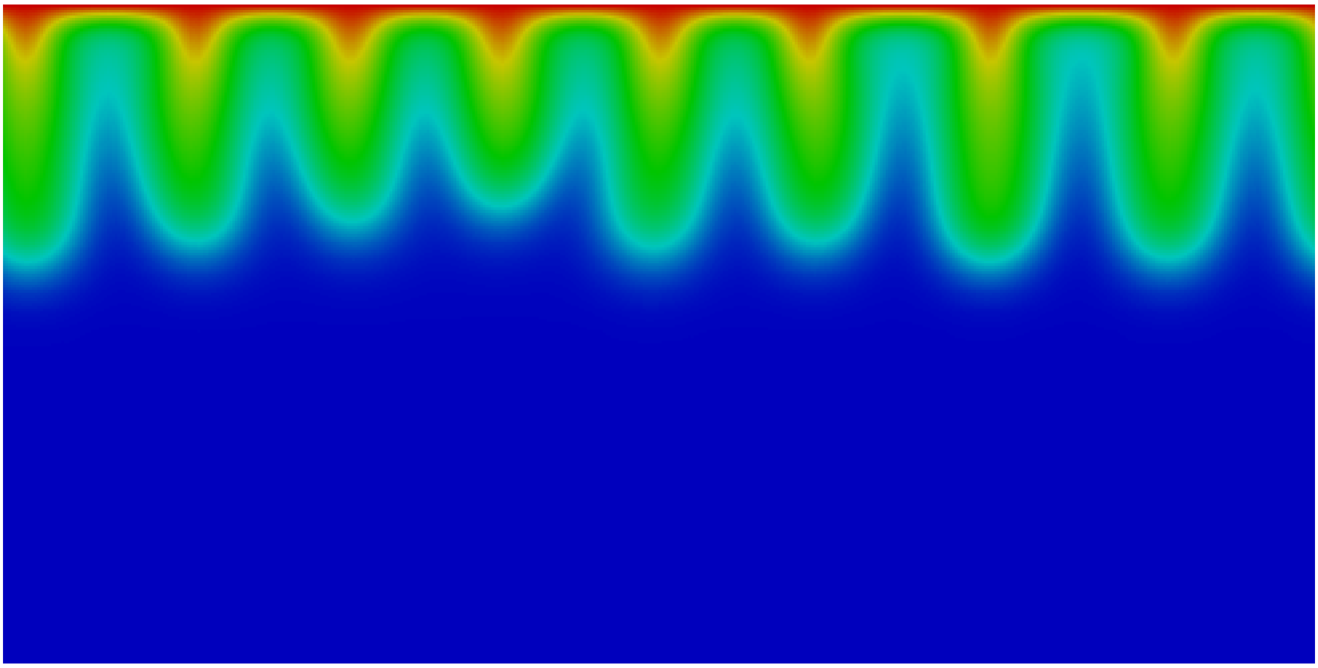

Figure 8.1: 2D concentration profile (t = 3528 s)

The flux per unit width over the top boundary is of particular interest in many cases (especially convective mixing of $\mathrm{CO_2}$). This is calculated using the *boundaryfluxint* postprocessor in the input file, and presented in 8.2.

Initially, the flux is purely diffusive, and scales as $1/\sqrt{(\pi t)}$, where t is time (shown as the dashed red line). After some time, the convective instability becomes sufficiently strong, at which point the flux across the top boundary rapidly increases (at a time of approximately 2000 seconds).

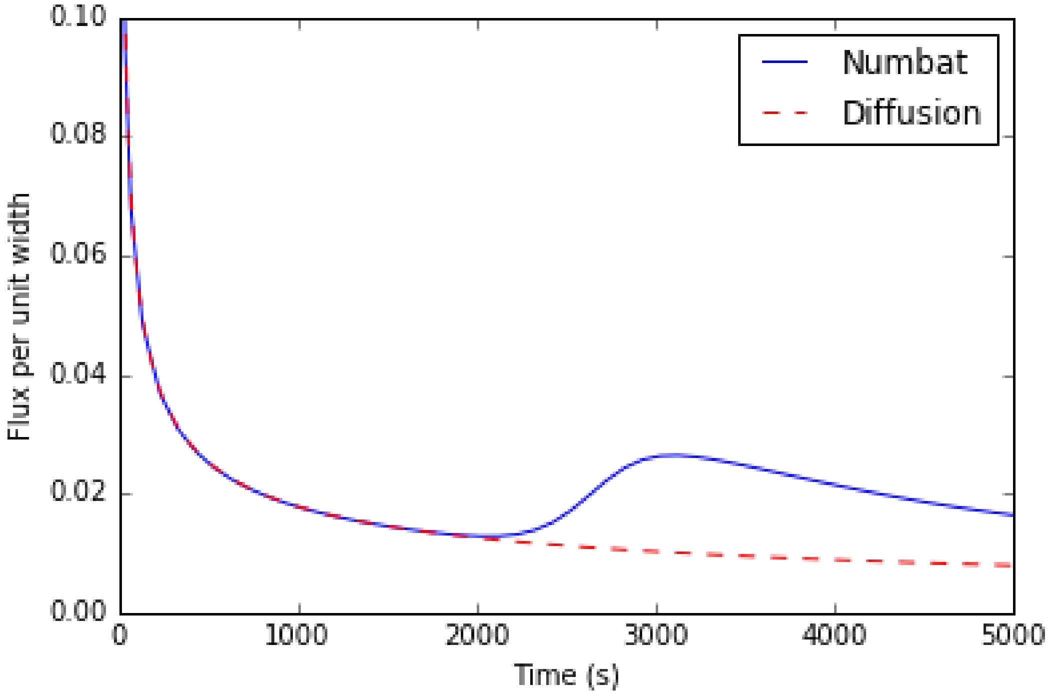

Figure 8.2: 2D flux across the top boundary

8.2 Anisotropic models

The second 2D example is for an anisotropic porous medium with $\gamma = 0.75$ (ie., the vertical permeability is three quarters of the horizontal permeability).

8.2.1 Input file

```
# 2D density-driven convective mixing. Instability is seeded by small
    perturbation
# to porosity. Permeability anisotropy is introduced with ky/kx = 0.5.
# Takes about 5 minutes to run using a single processor.

[Mesh]
  type = NumbatBiasedMesh
  dim = 2
  ymax = 1.5
  nx = 100
  ny = 50
  refined_edge = top
  refined_resolution = 0.001
[]

[Variables]
  [./concentration]
    initial_condition = 0
    scaling = 1e6
  [../]
  [./pressure]
    initial_condition = 1e6
```

```
  [../]
[]

[AuxVariables]
  [./noise]
    family = MONOMIAL
    order = CONSTANT
  [../]
[]

[ICs]
  [./noise]
    type = RandomIC
    variable = noise
    max = 0.003
    min = -0.003
  [../]
[]

[Materials]
  [./porosity]
    type = NumbatPorosity
    porosity = 0.3
    noise = noise
  [../]
  [./permeability]
    type = NumbatPermeability
    permeability = '1e-11 0 0 0 5e-12 0 0 0 1e-11'
  [../]
  [./diffusivity]
    type = NumbatDiffusivity
    diffusivity = 2e-9
  [../]
  [./density]
    type = NumbatDensity
    concentration = concentration
    zero_density = 995
    delta_density = 10.5
  [../]
  [./viscosity]
    type = NumbatViscosity
    viscosity = 6e-4
  [../]
[]

[Kernels]
  [./time]
    type = NumbatTimeDerivative
    variable = concentration
  [../]
  [./diffusion]
    type = NumbatDiffusion
    variable = concentration
  [../]
  [./convection]
    type = NumbatConvection
```

```
    variable = concentration
    pressure = pressure
  [../]
  [./darcy]
    type = NumbatDarcy
    variable = pressure
    concentration = concentration
  [../]
[]

[BCs]
  [./conctop]
    type = NumbatPerturbationBC
    variable = concentration
    boundary = top
    value = 1.0
  [../]
  [./Periodic]
    [./x]
      variable = concentration
      auto_direction = x
    [../]
  [../]
[]

[Preconditioning]
  [./smp]
    type = SMP
    full = true
  [../]
[]

[Executioner]
  type = Transient
  l_max_its = 200
  end_time = 5e5
  solve_type = NEWTON
  petsc_options = -ksp_snes_ew
  petsc_options_iname = '-ksp_atol'
  petsc_options_value = '1e-10'
  nl_abs_tol = 1e-9
  dtmax = 1e3
  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 1
  [../]
[]

[Postprocessors]
  [./boundaryfluxint]
    type = NumbatSideFlux
    variable = concentration
    boundary = top
  [../]
  [./mass]
    type = NumbatTotalMass
```

```
    variable = concentration
  [../]
[]

[Outputs]
  [./console]
    type = Console
    perf_log = true
    output_nonlinear = true
  [../]
  [./exodus]
    type = Exodus
    file_base = 2Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
  [./csvoutput]
    type = CSV
    file_base = 2Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
[]
```

Note that the permeability anisotropy is introduced in the kernels using the *gamma* and *anisotropic_tensor* input parameters.

8.2.2 Running the example

This example can be run on the commandline using

```
numbat-opt -i 2Dddc_anisotropic.i
```

Alternatively, this file can be run using the *Peacock* gui provided by MOOSE using

```
peacock -i 2Dddc_anisotropic.i
```

in the directory where the input file *2Dddc_anisotropic.i* resides.

8.2.3 Results

This 2D example should take only a few minutes to run to completion, producing a concentration profile similar to that presented in Figure 8.3, where several downwelling plumes of high concentration can be observed after 5000 s:

In comparison to the isotropic example (with $\gamma = 1$) presented in Figure 8.1, we note that the concentration profile in the anisotropic example has only reached a similar depth after 5000 s (compared to 3528 s). The effect of the reduced vertical permeability in the anisotropic example slows the convective transport.

This observation can be quantified by comparing the flux per unit width over the top boundary of both examples, see Figure 8.4.

The inclusion of permeability anisotropy delays the onset of convection in comparison to the isotropic example, from a time of approximately 2000 seconds in the isotropic example to approximately 3500 seconds in the anisotropic example.

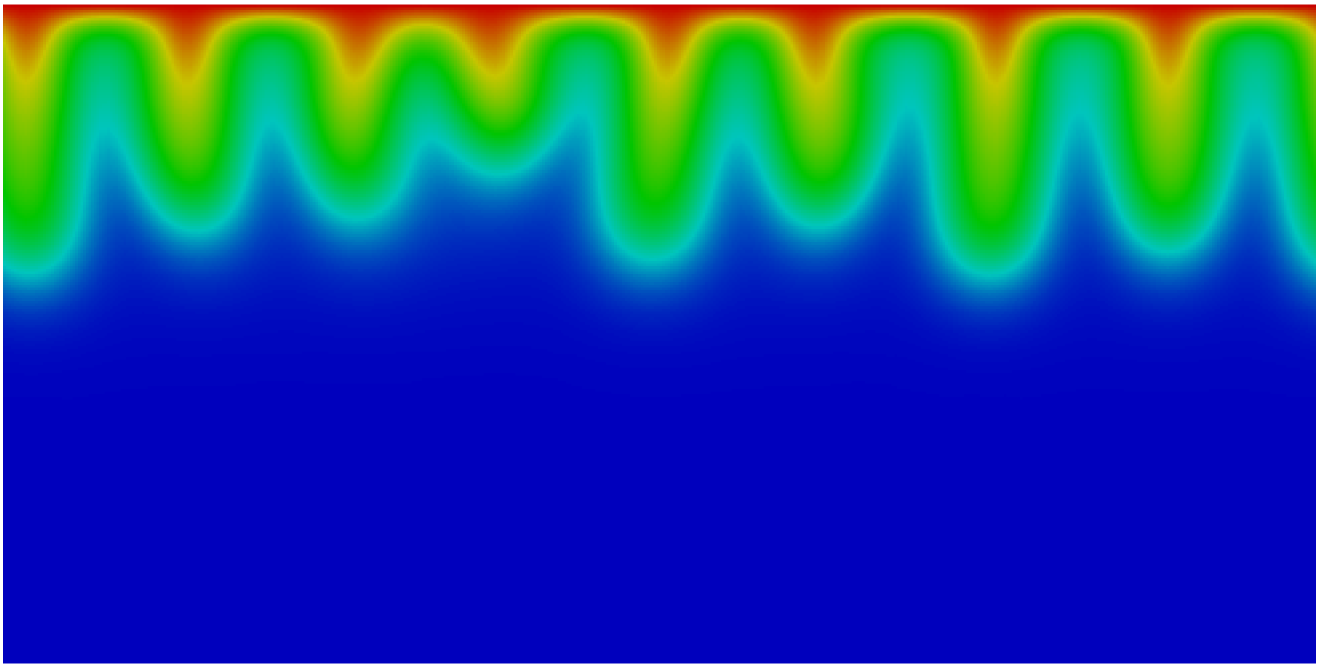

Figure 8.3: 2D concentration profile for $\gamma = 0.75$ (t = 5000 s)

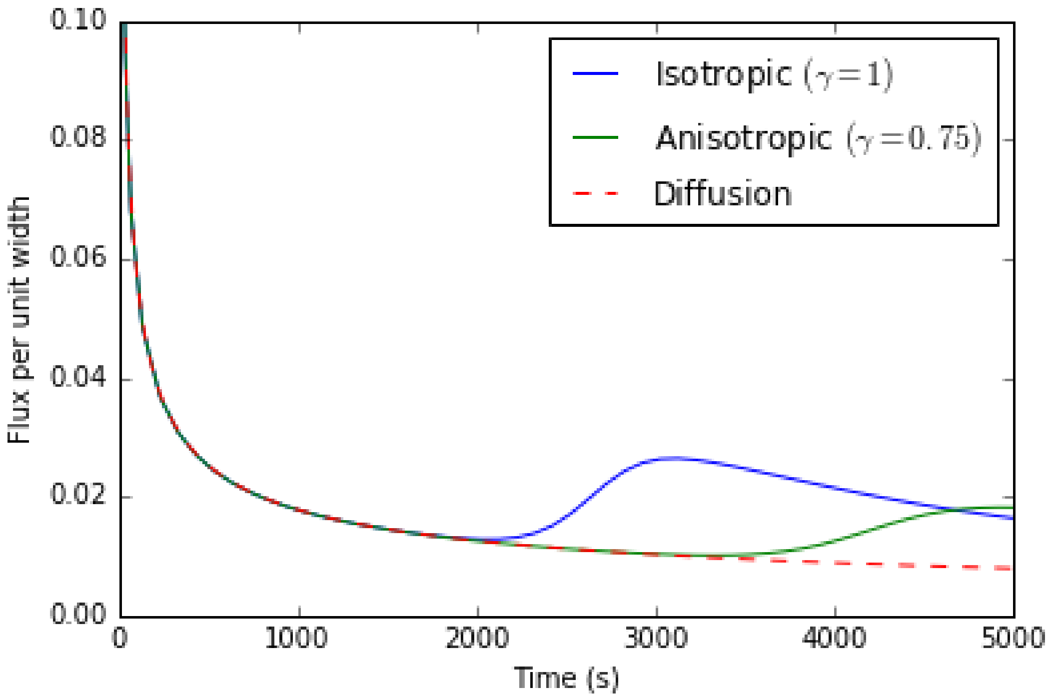

Figure 8.4: Comparison of the 2D flux across the top boundary

9 Three-dimensional examples

Complete input files for 2D models using the dimensional and dimensionless streamfunction formulations are provided, for both isotropic and anisotropic porous media. These examples are provided in the Numbat *examples* folder.

9.1 Isotropic models

The first examples are for an isotropic porous medium ($\gamma = 1$).

9.1.1 Input file

The complete input file for this problem is

```
# 3D density-driven convective mixing. Instability is seeded by small
   perturbation
# to porosity. Don't try this on a laptop!

[Mesh]
  type = NumbatBiasedMesh
  dim = 3
  zmax = 1.5
  nx = 20
  ny = 20
  nz = 500
  refined_edge = front
  refined_resolution = 0.001
[]

[Variables]
  [./concentration]
    initial_condition = 0
    scaling = 1e6
  [../]
  [./pressure]
    initial_condition = 1e6
  [../]
[]

[AuxVariables]
  [./noise]
    family = MONOMIAL
    order = CONSTANT
  [../]
[]

[ICs]
  [./noise]
    type = RandomIC
    variable = noise
    max = 0.003
    min = -0.003
  [../]
[]

[Materials]
  [./porosity]
```

```
    type = NumbatPorosity
    porosity = 0.3
    noise = noise
  [../]
  [./permeability]
    type = NumbatPermeability
    permeability = '1e-11 0 0 0 1e-11 0 0 0 1e-11'
  [../]
  [./diffusivity]
    type = NumbatDiffusivity
    diffusivity = 2e-9
  [../]
  [./density]
    type = NumbatDensity
    concentration = concentration
    zero_density = 995
    delta_density = 10.5
    saturated_concentration = 0.049306
  [../]
  [./viscosity]
    type = NumbatViscosity
    viscosity = 6e-4
  [../]
[]

[Kernels]
  [./time]
    type = NumbatTimeDerivative
    variable = concentration
  [../]
  [./diffusion]
    type = NumbatDiffusion
    variable = concentration
  [../]
  [./convection]
    type = NumbatConvection
    variable = concentration
    pressure = pressure
  [../]
  [./darcy]
    type = NumbatDarcy
    variable = pressure
    concentration = concentration
  [../]
[]

[BCs]
  [./conctop]
    type = PresetBC
    variable = concentration
    boundary = front
    value = 0.049306
  [../]
  [./Periodic]
    [./x]
      variable = concentration
```

```
      auto_direction = 'x y'
    [../]
  [../]
[]

[Preconditioning]
  [./smp]
    type = SMP
    full = true
  [../]
[]

[Executioner]
  type = Transient
  l_max_its = 200
  end_time = 3e5
  solve_type = NEWTON
  petsc_options = -ksp_snes_ew
  petsc_options_iname = '-pc_type -sub_pc_type -ksp_atol'
  petsc_options_value = 'bjacobi ilu 1e-12'
  nl_abs_tol = 1e-10
  nl_max_its = 25
  dtmax = 2e3
  [./TimeStepper]
    type = IterationAdaptiveDT
    dt = 1
  [../]
[]

[Postprocessors]
  [./boundaryfluxint]
    type = NumbatSideFlux
    variable = concentration
    boundary = top
  [../]
  [./mass]
    type = NumbatTotalMass
    variable = concentration
  [../]
[]

[Outputs]
  [./console]
    type = Console
    perf_log = true
    output_nonlinear = true
  [../]
  [./exodus]
    type = Exodus
    file_base = 3Dddc
    execute_on = 'INITIAL TIMESTEP_END'
  [../]
  [./csvoutput]
    type = CSV
    file_base = 3Dddc
    execute_on = 'INITIAL TIMESTEP_END'
```

```
  [../]
[]
```

9.1.2 Running the example

Note:

This example should **not** be run on a laptop or workstation due to the large computational requirements. Do **not** run this using the *Peacock* gui provided by MOOSE.

Examples of the total run times for this problem on a cluster are over 27 hours for a single processor down to only 30 minutes using 100 processors in parallel.

9.1.3 Results

This 3D example should produce a concentration profile similar to that presented in Figure 9.1, where several downwelling plumes of high concentration can be observed:

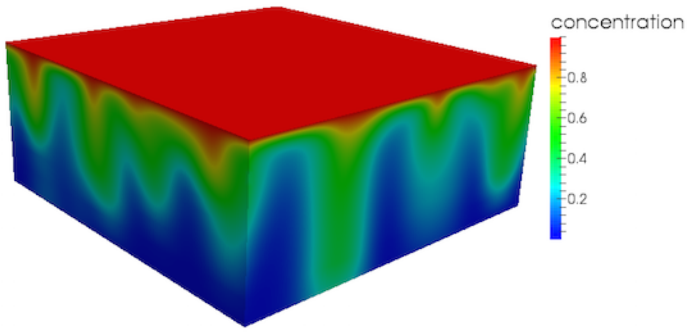

Figure 9.1: 3D concentration profile

Note that due to the random perturbation applied to the initial concentration profile, the geometry of the concentration profile obtained will differ from run to run.

The flux over the top surface is of particular interest in many cases (especially convective mixing of CO_2). This is calculated in this example file using the NumbatSideFlux in the input file, and presented in Figure 9.2.

Initially, the flux is purely diffusive, and scales as $1/\sqrt{(\pi t)}$, where t is time (shown as the dashed green line). After some time, the convective instability becomes sufficiently strong, at which point the flux across the top boundary rapidly increases (at a time of approximately 1,700 seconds). Also shown for comparison is the flux for the 2D example. It is apparent that the 3D model leads in a slower onset of convection (the time where the flux first increases from the diffusive rate).

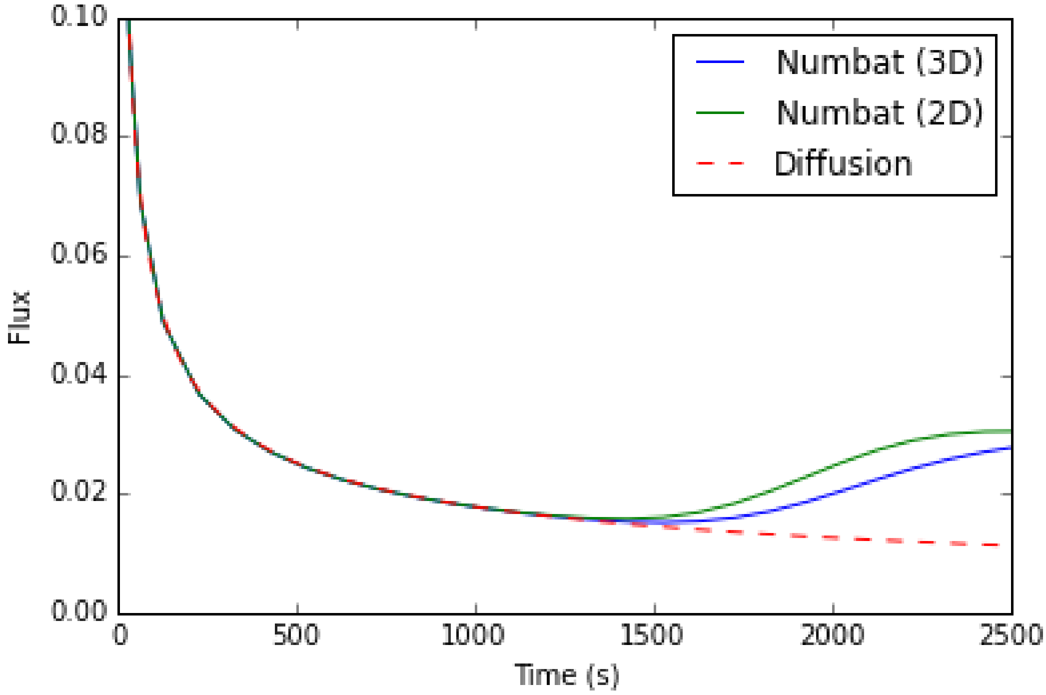

Figure 9.2: 3D flux across the top boundary

References

E, W., & Liu, J. G. 1997. Finite difference methods for 3D viscous incompressible flows in the vorticity-vector potential formulation on nonstaggered grids. *J. Comp. Phys.*, **138**, 57–82.

Emami-Meybodi, Hamid, Hassanzadeh, Hassan, Green, Christopher P., & Ennis-King, Jonathan. 2015. Convective dissolution of CO_2 in saline aquifers: Progress in modeling and experiments. *International Journal of Greenhouse Gas Control*, **40**, 238–266.

Ennis-King, J., & Paterson, L. 2005. Role of convective mixing in the long-term storage of carbon dioxide in deep saline aquifers. *SPE J.*, **10**, 349–356.

Slim, A.C. 2014. Solutal-convection regimes in a two-dimensional porous medium. *J. Fluid Mech.*, **741**, 461–491.

CONTACT US

t 1300 363 400
+61 3 9545 2176
e csiroenquiries@csiro.au
w www.csiro.au

WE DO THE EXTRAORDINARY EVERY DAY

We innovate for tomorrow and help improve today Ű for our customers, all Australians and the world.
Our innovations contribute billions of dollars to the Australian economy every year. As the largest patent holder in the nation, our vast wealth of intellectual property has led to more than 150 spin-off companies.
With more than 5,000 experts and a burning desire to get things done, we are AustraliaŠs catalyst for innovation.
WE IMAGINE. WE COLLABORATE. WE INNOVATE.

FOR FURTHER INFORMATION

CSIRO Energy
Chris Green
t +61 3 9545 8371
e chris.green@csiro.au
w www.csiro.au