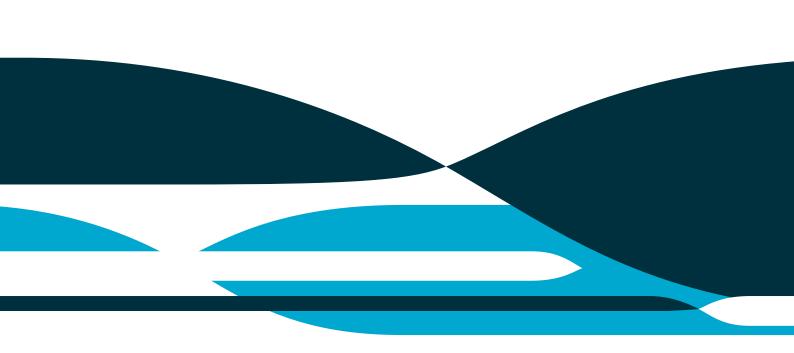


Numbat

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

Chris Green

Report EPXXXXX Version: ea0ecde 06/06/2017



CSIRO Energy Flagship 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

© 2016 CSIRO To the extent permitted by law, all rights are reserved and no part of this publication covered by copyright may be reproduced or copied in any form or by any means except with the written permission of CSIRO.

Important disclaimer

CSIRO advises that the information contained in this publication comprises general statements based on scientific research. The reader is advised and needs to be aware that such information may be incomplete or unable to be used in any specific situation. No reliance or actions must therefore be made on that information without seeking prior expert professional, scientific and technical advice. To the extent permitted by law, CSIRO (including its employees and consultants) excludes all liability to any person for any consequences, including but not limited to all losses, damages, costs, expenses and any other compensation, arising directly or indirectly from using this publication (in part or in whole) and any information or material contained in it.

Contents

| 1 | Nui | | 1 | | | | | |
|---|------|---|------------|--|--|--|--|--|
| | 1.1 | High-resolution simulations of density-driven convective mixing in porous media | 1 | | | | | |
| 2 | Inst | Installation instructions | | | | | | |
| | 2.1 | Install MOOSE | 2 | | | | | |
| | 2.2 | Clone Numbat | 2 | | | | | |
| | 2.3 | Compile Numbat | 2 | | | | | |
| | 2.4 | • | 2 | | | | | |
| | Bar | ckground theory | 3 | | | | | |
| | | | | | | | | |
| | 3.1 | | 3 | | | | | |
| | 3.2 | | 4 | | | | | |
| | 3.3 | 3D solution | 5 | | | | | |
| 4 | Inp | ut file syntax | 7 | | | | | |
| | 4.1 | Essential input | 7 | | | | | |
| | | | 7 | | | | | |
| | | | 8 | | | | | |
| | | | 8 | | | | | |
| | | | 9 | | | | | |
| | | • | L0 L2 | | | | | |
| | | | 12 [2 | | | | | |
| | | | L2 L2 | | | | | |
| | 4.2 | • | 13 | | | | | |
| | 4.2 | | L3 | | | | | |
| | | | L 4 | | | | | |
| | | | L4 | | | | | |
| 5 | 2D | examples | ۱7 | | | | | |
| 3 | 5.1 | · | - · L7 | | | | | |
| | J.1 | | L 1 L 7 | | | | | |
| | | | 20 | | | | | |
| | | | 21 | | | | | |
| | 5.2 | | 21 | | | | | |
| | 0.2 | | 21 | | | | | |
| | | | 26 | | | | | |
| | | | 26 | | | | | |
| 6 | 3D | example | 28 | | | | | |
| | | | 28 | | | | | |
| | | · | 32 | | | | | |
| | | | 32 | | | | | |
| 7 | Dof | ioroneos |) E | | | | | |

Numbat 1

Version: ea0ecde (06/06/2017)

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

Numbat is a massively-parallel code for high-resolution simulations of density-driven convective mixing in porous media built using the MOOSE framework.

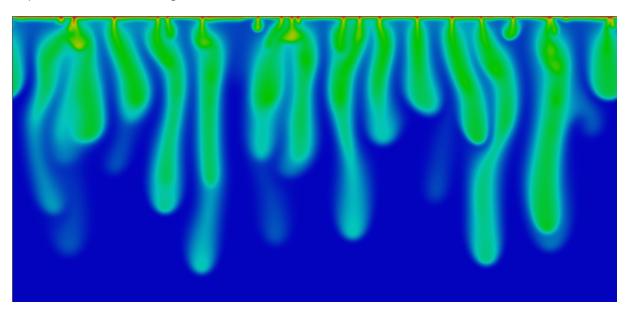


Figure 1.1: Density-driven convective mixing in a porous medium

2 Installation instructions

To install Numbat, follow these simple instructions.

2.1 **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.

Clone Numbat 2.2

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
```

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*.

2.4 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.

3 **Background theory**

3.1 **Governing equations**

Numbat implements the Boussinesq approximation to model density-driven convective mixing in porous media.

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

$$w = 0, \quad z = 0, -H,$$
 (3.5)

$$\frac{\partial c}{\partial z} = 0, \quad z = -H,\tag{3.6}$$

$$c = c_0, \quad z = 0,$$
 (3.7)

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

$$c = 0, \quad t = 0.$$
 (3.8)

The governing equations are 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 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 et. al (2005). Defining the anisotropy ratio γ as

$$\gamma = \frac{k_z}{k_x},\tag{3.9}$$

we scale the variables using

$$x = \frac{\phi\mu D}{k_z \Delta \rho g \gamma^{1/2}} \hat{x}, \quad z = \frac{\phi\mu D}{k_z \Delta \rho g} \hat{z}, \quad u = \frac{k_z \Delta \rho g}{\mu \gamma^{1/2}} \hat{u}, \quad w = \frac{k_z \Delta \rho g}{\mu} \hat{w}$$
$$t = \left(\frac{\phi\mu}{k_z \Delta \rho g}\right)^2 \hat{t}, \quad c = c_0 \hat{c}, \quad P = \frac{\mu \phi D}{k_z} \hat{P}, \tag{3.10}$$

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

$$\mathbf{u} = -\left(\nabla P + c\hat{\mathbf{k}}\right),\tag{3.11}$$

$$\mathbf{u} = 0, \tag{3.12}$$

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \gamma \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial z^2},\tag{3.13}$$

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

The dimensionless boundary conditions are

$$w = 0, \quad z = 0, -Ra,$$
 (3.14)

$$\frac{\partial c}{\partial z} = 0, \quad z = -Ra,\tag{3.15}$$

$$c = 1, \quad z = 0,$$
 (3.16)

where Ra is the Rayleigh number, defined as

$$Ra = \frac{k_z \Delta \rho g H}{\phi \mu D}.$$
 (3.17)

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

$$c = 0, \quad t = 0.$$
 (3.18)

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

$$u = -\frac{\partial \psi}{\partial z}, \quad w = \frac{\partial \psi}{\partial x}.$$
 (3.19)

This definition satisfies the continuity equation, Eq. (3.12), immediately.

The pressure P is removed from Eq. (3.11) by taking the curl of both sides and noting that $\nabla \times \nabla P = 0$ for any P, to give

$$\nabla^2 \psi = -\frac{\partial c}{\partial x},\tag{3.20}$$

where we have introduced the streamfunction ψ using Eq. (3.19).

The convection-diffusion equation, Eq. (3.13) becomes

$$\frac{\partial c}{\partial t} - \frac{\partial \psi}{\partial z} \frac{\partial c}{\partial x} + \frac{\partial \psi}{\partial x} \frac{\partial c}{\partial z} = \gamma \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial z}.$$
 (3.21)

The boundary conditions become

$$\frac{\partial \psi}{\partial x} = 0, \quad z = 0, -Ra, \tag{3.22}$$

$$\frac{\partial c}{\partial z} = 0, \quad z = -Ra,\tag{3.23}$$

$$c = 1, \quad z = 0,$$
 (3.24)

while the initial condition is still given by Eq. (3.18).

In two dimensions, Numbat solves Eq's. (3.20) and (3.21).

3.3 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's. (3.11) to (3.13), with boundary conditions given by Eq's. (3.14) to (3.16), and initial condition given by Eq. (3.18).

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

$$\mathbf{u} = \nabla \times \Psi. \tag{3.25}$$

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

$$\nabla \times (\Psi + \nabla \zeta) = \nabla \times \Psi \quad \forall \zeta, \tag{3.26}$$

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. (3.11) and substituting Eq. (3.25), we have

$$\nabla(\nabla \cdot \Psi) - \nabla^2 \Psi = \left(-\frac{\partial c}{\partial y}, \frac{\partial c}{\partial x}, 0 \right), \tag{3.27}$$

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

$$\nabla^2 \Psi = \left(\frac{\partial c}{\partial y}, -\frac{\partial c}{\partial x}, 0\right). \tag{3.28}$$

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

$$\psi_x = \psi_y = 0, \quad z = 0, -Ra, \quad \frac{\partial \psi_z}{\partial z} = 0,$$
 $z = 0, -Ra.$

The governing equations are then

$$\nabla^2 \Psi = \left(\frac{\partial c}{\partial y}, -\frac{\partial c}{\partial x}, 0\right),\tag{3.29}$$

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \gamma \left(\frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial y^2} \right) + \frac{\partial^2 c}{\partial z^2},\tag{3.30}$$

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,

$$\Psi = (\psi_x, \psi_y, 0), \tag{3.31}$$

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

$$\mathbf{u} = \left(-\frac{\partial \psi_y}{\partial z}, \frac{\partial \psi_x}{\partial z}, \frac{\partial \psi_y}{\partial x} - \frac{\partial \psi_x}{\partial y} \right). \tag{3.32}$$

Note that if there is no y dependence, Eq's. (3.29) and (3.30) reduce to

$$\nabla^2 \Psi = \left(0, -\frac{\partial c}{\partial x}, 0\right),\tag{3.33}$$

$$\frac{\partial c}{\partial t} + \mathbf{u} \cdot \nabla c = \gamma \frac{\partial^2 c}{\partial x^2} + \frac{\partial^2 c}{\partial z^2}.$$
(3.34)

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's. (3.29) and (3.30).

4 Input file syntax

The input file for a Numbat simulation is a simple, block-structured text file.

A working example of a 2D problem can be found at https://github.com/cpgr/numbat/blob/ master/examples/2D/2Dddc.i

A working example of a 3D problem can be found at https://github.com/cpgr/numbat/blob/ master/examples/3D/3Dddc.i

4.1 **Essential input**

Details of the minimum input file requirements are given below.

4.1.1 Mesh

All simulations must feature a mesh. For the basic model with a rectangular mesh, the built-in MOOSE GeneratedMesh can be used to create a suitable mesh. In 2D, the input block looks like:

```
[Mesh]
 type = GeneratedMesh
 dim = 2
 xmax = 1000
 ymin = -200
 ymax = 0
 nx = 80
 ny = 20
 bias_y = 0.7
```

This creates a 2D mesh from x=0 to x=1000 and y=-200 to y=0 with 80 elements in the x-direction and 20 elements in the y-direction. It is useful to have a mesh that is more refined at the top of the model, to accurately capture the initially small structure of the convective fingers. This is achieved using the built-in bias_y parameter.

In 3D, the Mesh block would look like:

```
[Mesh]
 type = GeneratedMesh
 dim = 3
 xmax = 200
 ymax = 200
 zmin = -200
 zmax = 0
 nx = 10
 ny = 10
 nz = 10
 bias_z = 0.7
```

Again, the mesh is refined at the top of the model using the the bias_z parameter.

4.1.2 Variables

For a 2D model, the simulation must have two variables: *concentration* and *streamfunction*. This can be implemented in the input file using the following code:

```
[Variables]
[./concentration]
order = FIRST
family = LAGRANGE
  [./InitialCondition]
    type = NumbatPerturbationIC
    variable = concentration
    amplitude = 0.01
    seed = 1
  [...]
[../]
[./streamfunction]
  order = FIRST
 family = LAGRANGE
  initial_condition = 0.0
[../]
```

Initial conditions can also be specified in the *Variables* block. In this case, the initial concentration is perturbed using a *NumbatPerturbationIC* to seed the instability.

For a 3D model, three variables are required: one *concentration* variable and two *streamfunction* variables corresponding to the x and y components. This can be implemented in the input file using:

```
[Variables]
  [./concentration]
  [../]
  [./streamfunctionx]
  [../]
  [./streamfunctiony]
  [../]
```

4.1.3 Kernels

Three kernels are required for a 2D model: a *NumbatDarcySF* kernel for the *streamfunction* variable, a *NumbatConvectionDiffusionSF* kernel for the *concentration* variable, and a *TimeDerivative* kernel also for the *concentration* variable. An example for an isotropic model is

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

```
[./ConvectionDiffusion]
   type = NumbatConvectionDiffusionSF
   variable = concentration
   streamfunction_variable = streamfunction
   coeff_tensor = '1 0 0 0 1 0 0 0 1'
  [../]
  [./TimeDerivative]
   type = TimeDerivative
   variable = concentration
 [../]
```

The coeff_tensor parameter in each convective diffusion kernel can be modified. The format of this For 3D models, an additional NumbatDarcySF kernel is required for the additional stream function variable. An example of the kernels block for a 3D isotropic model is

```
[Kernels]
  [./Darcy]
   type = NumbatDarcySF
   variable = streamfunctionx
   concentration_variable = concentration
   component = x
  [../]
  [./Darcy]
   type = NumbatDarcySF
   variable = streamfunctiony
   concentration_variable = concentration
   component = y
  [../]
  [./ConvectionDiffusion]
   type = NumbatConvectionDiffusionSF
   variable = concentration
   streamfunction_variable = 'streamfunctionx streamfunctiony'
   coeff_tensor = '1 0 0 0 1 0 0 0 1'
  [../]
  [./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_variable keyword in the NumbatConvectionDiffusionSF kernel must contain both streamfunction variables ordered by the x component then the y component.

4.1.4 Initial condition

To seed the instability, a random perturbation to the initial concentration is prescribed using the NumbatPerturbationIC initial condition. This can be applied in the Variables block in the input file (see above) using

```
[./InitialCondition]
  type = NumbatPerturbationIC
  variable = concentration
  amplitude = 0.01
  seed = 1
[../]
```

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

$$c_d(z, t = 1) = 1 + \operatorname{erf}(z/2),$$
 (4.1)

for z < 0, where 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*.

4.1.5 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 reservoir), and no-flow boundary conditions at the top and bottom surfaces.

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

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

In this case, the conctop boundary condition is a Dirichlet condition at the top of the model that fixes the value of concentration to unity. 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. No-flow boundary conditions are prescribed on the top and bottom surfaces by holding the streamfunction variable constant (in this case 0). Finally, periodic boundary conditions are applied by the periodic block, which specifies that both the concentration and streamfunction variables are periodic on boundaries in the x-direction.

A similar boundary condition block is used in 3D, except that no-flow boundaries must be imposed on both streamfunction variables, see below:

```
[BCs]
  [./conctop]
   type = DirichletBC
   variable = concentration
   boundary = front
   value = 1.0
  [../]
 [./streamfunxtop]
   type = DirichletBC
   variable = streamfunctionx
   boundary = front
   value = 0.0
  [../]
  [./streamfunxbottom]
   type = DirichletBC
   variable = streamfunctionx
   boundary = back
   value = 0.0
  [../]
  [./streamfunytop]
   type = DirichletBC
   variable = streamfunctiony
   boundary = front
   value = 0.0
  [../]
  [./streamfunybottom]
   type = DirichletBC
   variable = streamfunctiony
   boundary = back
   value = 0.0
  [../]
 [./Periodic]
   [./xy]
     variable = 'concentration streamfunctionx streamfunctiony'
     auto_direction = 'x y'
   [../]
```

```
[../]
```

4.1.6 Executioner

Each MOOSE simulation must use an *Executioner*, which provides parameters for the solve. In both 2D and 3D models, a transient *Executioner* is used, an example of which is presented below:

```
[Executioner]
 type = Transient
 scheme = bdf2
 dtmin = 0.1
 dtmax = 200
 end_time = 2000
 solve_type = PJFNK
 petsc_options_iname = '-ksp_type -pc_type -pc_sub_type'
 petsc_options_value = 'gmres asm ilu'
 [./TimeStepper]
   type = IterationAdaptiveDT
   dt = 1
   cutback_factor = 0.5
   growth_factor = 2
 [../]
```

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

4.1.7 Preconditioning

A default preconditioning block is used that provides all Jacobian entries to aid convergence. This is identical for both 2D and 3D models:

```
[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.

4.1.8 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 = filename
   execute_on = 'INITIAL TIMESTEP_END FINAL'
 [../]
[]
```

In this case, some output regarding the iterations is streamed to the console, while the results are provided in an Exodus file named filename.e. There are a large number of output options available in MOOSE, see the MOOSE website for further details.

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 rsults provided.

4.2.1 Mesh adaptivity

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

```
[Adaptivity]
 marker = combomarker
 \max_h = 2
 initial_marker = boxmarker
 initial\_steps = 1
 [./Indicators]
    [./gradjumpindicator]
      type = GradientJumpIndicator
      variable = concentration
    \lceil .../\rceil
  [../]
  [./Markers]
    [./errormarker]
      type = ErrorToleranceMarker
      coarsen = 0.0025
      refine = 0.005
      indicator = gradjumpindicator
    \lceil \dots / \rceil
    [./boxmarker]
      type = BoxMarker
      bottom_left = '0 -1.0 0'
      top_right = '1000 0 0'
      inside = refine
      outside = dont_mark
    [../]
    [./combomarker]
      type = ComboMarker
```

```
markers = 'boxmarker errormarker'
[../]
[../]
```

For details about mesh adaptivity, see the MOOSE website.

4.2.2 Flux at the top boundary

The flux over the top boundary is of particular interest in many cases (especially convective mixing of CO_2). This can be calculated at each time step using a *Postprocessor*:

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

The output of the *Postprocessor* can be saved to a *csv* file by including the following additional sub-block in the *Outputs* block:

```
[./csvoutput]
  type = CSV
  file_base = filename
  execute_on = 'INITIAL TIMESTEP_END FINAL'
[../]
```

4.2.3 Velocity components

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. Importantly, these auxiliary variables **must** have constant monomial shape functions (these are referred to as *elemental* variables, as the value is constant over each mesh element). This restriction is due to the gradient of the streamfunction variable(s) being 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 NumbatDarcyVelocitySF AuxKernels, one for each component. For the 2D case, the input syntax is

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

For the 3D case, three AuxKernels are required. Note that both streamfunction variables must be given, in the correct order (x then y). An example of the input syntax is

```
[AuxKernels]
 [./uAux]
   type = NumbatDarcyVelocitySF
   variable = u
   component = x
   streamfunction_variable = 'streamfunctionx streamfunctiony'
  [...]
```

```
[./vAux]
   type = NumbatDarcyVelocitySF
   variable = v
   component = y
   streamfunction_variable = 'streamfunctionx streamfunctiony'
 [../]
 [./wAux]
   type = NumbatDarcyVelocitySF
   variable = w
   component = z
   streamfunction_variable = 'streamfunctionx streamfunctiony'
 [../]
[]
```

2D examples

Isotropic models

The first 2D example is for an isotropic porous medium ($\gamma = 1$). A working example of this example can be found at https://github.com/cpgr/numbat/blob/master/examples/2D/isotropic/ 2Dddc_isotropic.i.

5.1.1 Input file

The complete input file for this problem is

```
[Mesh]
 type = GeneratedMesh
 dim = 2
 xmax = 1000
 ymin = -500
 ymax = 0
 nx = 80
 ny = 25
 bias_y = 0.9
[Adaptivity]
 marker = errormarker
 \max_h_{\text{level}} = 1
 [./Indicators]
   [./gradjumpindicator]
      type = GradientJumpIndicator
     variable = concentration
   \lceil \dots / \rceil
 [../]
 [./Markers]
    [./errormarker]
     type = ErrorToleranceMarker
     refine = 0.005
     indicator = gradjumpindicator
   [../]
 [../]
[Variables]
 [./concentration]
   order = FIRST
   family = LAGRANGE
    [./InitialCondition]
     type = NumbatPerturbationIC
     variable = concentration
     amplitude = 0.1
     seed = 1
    [../]
```

```
[...]
 [./streamfunction]
   order = FIRST
   family = LAGRANGE
   initial_condition = 0.0
 [../]
Π
[Kernels]
  [./Darcy]
   type = NumbatDarcySF
   variable = streamfunction
   concentration_variable = concentration
  [./TwoDConvectionDiffusion]
   type = NumbatConvectionDiffusionSF
   variable = concentration
   streamfunction_variable = streamfunction
  [...]
  [./TimeDerivative]
   type = TimeDerivative
   variable = concentration
 [../]
[AuxVariables]
  [./u]
   order = CONSTANT
   family = MONOMIAL
  [../]
 [./w]
   order = CONSTANT
   family = MONOMIAL
 [../]
[AuxKernels]
  [./uAux]
   type = NumbatDarcyVelocitySF
   variable = u
   component = x
   streamfunction_variable = streamfunction
  [../]
  [./wAux]
   type = NumbatDarcyVelocitySF
   variable = w
   component = y
   streamfunction_variable = 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
   [../]
  [../]
[]
[Executioner]
 type = Transient
 dtmax = 100
 end_time = 5000
 start_time = 1
 solve_type = PJFNK
 nl_abs_tol = 1e-10
 [./TimeStepper]
   type = IterationAdaptiveDT
   dt = 1
   cutback_factor = 0.5
   growth_factor = 2
  [.../]
  [./TimeIntegrator]
   type = LStableDirk2
  [...]
```

```
[Postprocessors]
  [./boundaryfluxint]
   type = NumbatSideFlux
   variable = concentration
   boundary = top
  [...]
  [./numdofs]
   type = NumDOFs
 [../]
[Preconditioning]
  [./smp]
   type = SMP
   full = true
   petsc_options = -snes_ksp_ew
   petsc_options_iname = '-pc_type -sub_pc_type -ksp_atol'
   petsc_options_value = 'asm lu 1e-12'
 [../]
[Outputs]
  [./console]
   type = Console
   perf_log = true
   output_nonlinear = true
  [../]
  [./exodus]
   type = Exodus
   file_base = 2Dddc_isotropic
   execute_on = 'INITIAL TIMESTEP_END'
  [...]
  [./csvoutput]
   type = CSV
   file_base = 2Dddc_isotropic
   execute_on = 'INITIAL TIMESTEP_END'
  [../]
```

5.1.2 Running the example

This example can be run on the commandline using

```
numbat-opt -i 2Dddc_isotropic.i
```

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

```
peacock -i 2Dddc_isotropic.i
```

in the directory where the input file 2Dddc_isotropic.i resides.

5.1.3 Results

This 2D example should take only a few minutes to run to completion, producing a concentration profile similar to that presented in Figure 5.1, where several downwelling plumes of high concentration can be observed after 3528 s:

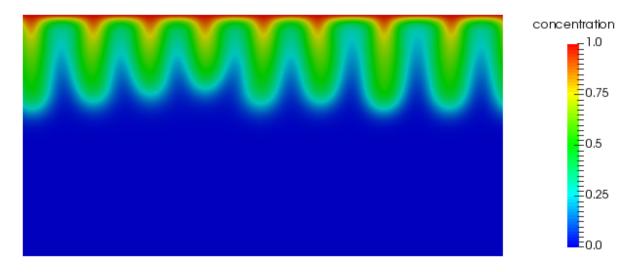


Figure 5.1: 2D concentration profile (t = 3528 s)

The flux per unit width over the top boundary is of particular interest in many cases (especially convective mixing of CO₂). This is calculated using the boundaryfluxint postprocessor in the input file, and presented in Figure 5.2.

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).

5.2 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). A working example of this example can be found at https://github.com/cpgr/numbat/blob/master/examples/2D/anisotropic/2Dddc_anisotropic. i.

5.2.1 Input file

The complete input file for this anisotropic example is

```
[Mesh]
 type = GeneratedMesh
 dim = 2
 xmax = 1000
 ymin = -500
 ymax = 0
 nx = 80
 ny = 25
 bias_y = 0.9
```

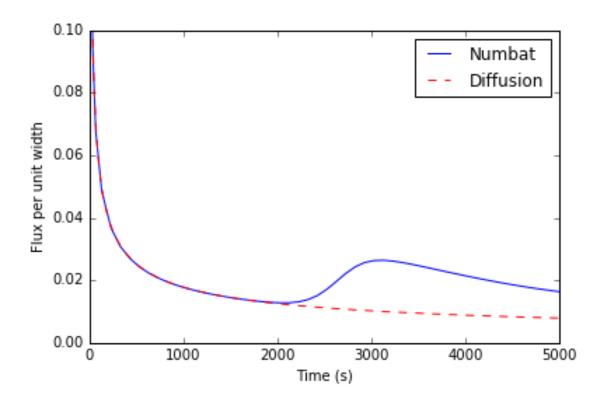


Figure 5.2: 2D flux across the top boundary

```
[]
[Adaptivity]
 marker = errormarker
 \max_h = 1
  [./Indicators]
   [./gradjumpindicator]
     type = GradientJumpIndicator
     variable = concentration
   [../]
  [../]
  [./Markers]
    [./errormarker]
     type = ErrorToleranceMarker
     refine = 0.005
     indicator = gradjumpindicator
   [../]
  [../]
[]
[Variables]
  [./concentration]
   order = FIRST
```

```
family = LAGRANGE
   [./InitialCondition]
     type = NumbatPerturbationIC
     variable = concentration
     amplitude = 0.1
     seed = 1
    [...]
  [../]
  [./streamfunction]
   order = FIRST
   family = LAGRANGE
   initial_condition = 0.0
  [../]
[Kernels]
  [./Darcy]
   type = NumbatDarcySF
   variable = streamfunction
   concentration_variable = concentration
   gamma = 0.75
  [...]
  [./ConvectionDiffusion]
   type = NumbatConvectionDiffusionSF
   variable = concentration
   streamfunction_variable = streamfunction
   anisotropic_tensor = '0.75 0 0 0 1 0 0 0 1'
  [../]
  [./TimeDerivative]
   type = TimeDerivative
   variable = concentration
 [../]
[AuxVariables]
 [./u]
   order = CONSTANT
   family = MONOMIAL
  [../]
  [./w]
   order = CONSTANT
   family = MONOMIAL
  [../]
[AuxKernels]
  [./uAux]
   type = NumbatDarcyVelocitySF
   variable = u
```

```
component = x
   streamfunction_variable = streamfunction
  [...]
  [./wAux]
   type = NumbatDarcyVelocitySF
   variable = w
   component = y
   streamfunction_variable = 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
   [../]
 [../]
[Executioner]
 type = Transient
 dtmax = 100
 end_time = 5000
 start_time = 1
 solve_type = PJFNK
 nl_abs_tol = 1e-10
 [./TimeStepper]
   type = IterationAdaptiveDT
   dt = 1
   cutback_factor = 0.5
```

```
growth_factor = 2
  [...]
  [./TimeIntegrator]
   type = LStableDirk2
 [../]
[]
[Postprocessors]
  [./boundaryfluxint]
   type = NumbatSideFlux
   variable = concentration
   boundary = top
  [../]
  [./numdofs]
   type = NumDOFs
 [../]
[Preconditioning]
  [./smp]
   type = SMP
   full = true
   petsc_options = -snes_ksp_ew
   petsc_options_iname = '-pc_type -sub_pc_type -ksp_atol'
   petsc_options_value = 'asm lu 1e-12'
 [../]
[Outputs]
  [./console]
   type = Console
   perf_log = true
   output_nonlinear = true
  \lceil \dots \rceil
  [./exodus]
   type = Exodus
   file_base = 2Dddc_anisotropic
   execute_on = 'INITIAL TIMESTEP_END'
  [../]
  [./csvoutput]
   type = CSV
   file_base = 2Dddc_anisotropic
   execute_on = 'INITIAL TIMESTEP_END'
  [../]
[]
```

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

5.2.2 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.

5.2.3 Results

This 2D example should take only a few minutes to run to completion, producing a concentration profile similar to that presented in Figure 5.3, where several downwelling plumes of high concentration can be observed after 5000 s:

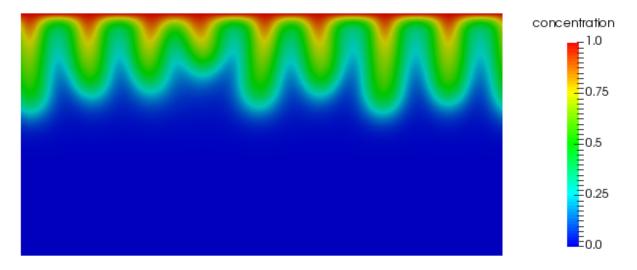


Figure 5.3: 2D concentration profile for $\gamma = 0.75$ (t = 5000 s)

In comparison to the isotropic example (with $\gamma=1$) presented in Figure 5.1, 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 5.4.

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.

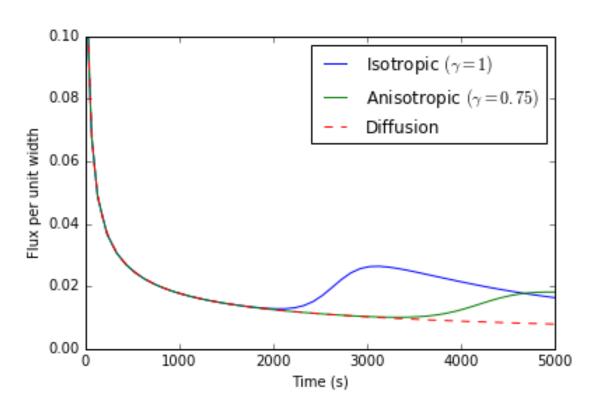


Figure 5.4: Comparison of the 2D flux across the top boundary

6 3D example

A working example of a 3D problem can be found at https://github.com/cpgr/numbat/blob/master/examples/3D/isotropic/3Dddc.i.

6.0.1 Input file

The complete input file for this problem is

```
[Mesh]
 type = GeneratedMesh
 dim = 3
 xmax = 500
 ymax = 500
 zmin = -200
 zmax = 0
 nx = 40
 ny = 40
 nz = 20
 bias_z = 0.7
[]
[Adaptivity]
 \max_h = 1
 initial_marker = boxmarker
 initial\_steps = 1
 marker = combomarker
 [./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
   [../]
   [./combomarker]
     type = ComboMarker
     markers = 'boxmarker errormarker'
   [../]
 [...]
```

```
[Variables]
 [./concentration]
   order = FIRST
   family = LAGRANGE
   [./InitialCondition]
     type = NumbatPerturbationIC
     variable = concentration
     amplitude = 0.02
     seed = 1
   [../]
 [...]
 [./streamfunctionx]
   order = FIRST
   family = LAGRANGE
   initial_condition = 0.0
 [../]
 [./streamfunctiony]
   order = FIRST
   family = LAGRANGE
   initial_condition = 0.0
 [../]
[Kernels]
  [./Darcy_x]
   type = NumbatDarcySF
   variable = streamfunctionx
   concentration_variable = concentration
   component = x
  [../]
 [./Darcy_y]
   type = NumbatDarcySF
   variable = streamfunctiony
   concentration_variable = concentration
   component = y
  [../]
  [./ConvectionDiffusion]
   type = NumbatConvectionDiffusionSF
   variable = concentration
   streamfunction_variable = 'streamfunctionx streamfunctiony'
   coeff_tensor = '1 0 0 0 1 0 0 0 1'
  [../]
 [./TimeDerivative]
   type = TimeDerivative
   variable = concentration
  [...]
```

```
[AuxVariables]
 [./u]
   order = CONSTANT
   family = MONOMIAL
 [./v]
   order = CONSTANT
   family = MONOMIAL
  [../]
  [./w]
   order = CONSTANT
   family = MONOMIAL
 [../]
[AuxKernels]
  [./uAux]
   type = NumbatDarcyVelocitySF
   variable = u
   component = x
   streamfunction_variable = 'streamfunctionx streamfunctiony'
  [../]
  [./vAux]
   type = NumbatDarcyVelocitySF
   variable = v
   component = y
   streamfunction_variable = 'streamfunctionx streamfunctiony'
 [../]
  [./wAux]
   type = NumbatDarcyVelocitySF
   variable = w
   component = z
   streamfunction_variable = 'streamfunctionx streamfunctiony'
 [../]
[BCs]
  [./conctop]
   type = DirichletBC
   variable = concentration
   boundary = front
   value = 1.0
 [../]
  [./streamfunxtop]
   type = DirichletBC
   variable = streamfunctionx
```

```
boundary = front
   value = 0.0
  [...]
  [./streamfunxbottom]
   type = DirichletBC
   variable = streamfunctionx
   boundary = back
   value = 0.0
  [../]
  [./streamfunytop]
   type = DirichletBC
   variable = streamfunctiony
   boundary = front
   value = 0.0
  [../]
  [./streamfunybottom]
   type = DirichletBC
   variable = streamfunctiony
   boundary = back
   value = 0.0
  [...]
  [./Periodic]
   [./xy]
     variable = 'concentration streamfunctionx streamfunctiony'
     auto_direction = 'x y'
   [../]
  [...]
[Preconditioning]
  [./smp]
   type = SMP
   full = true
 [../]
[Executioner]
 type = Transient
 dtmax = 100
 end_time = 2500
 start_time = 1
 solve_type = PJFNK
 nl_abs_tol = 1e-10
 petsc_options = -snes_ksp_ew
 [./TimeStepper]
   type = IterationAdaptiveDT
   dt = 1
   cutback_factor = 0.5
   growth_factor = 2
```

```
[../]
 [./TimeIntegrator]
   type = LStableDirk2
 [../]
[Postprocessors]
  [./boundaryfluxint]
   type = NumbatSideFlux
   variable = concentration
   boundary = front
 [../]
  [./numdofs]
   type = NumDOFs
 [...]
[Outputs]
 [./console]
   type = Console
   perf_log = true
   output_nonlinear = true
   output_linear = true
  [../]
 [./exodus]
   type = Exodus
   file_base = 3Dddc
   execute_on = 'INITIAL TIMESTEP_END'
  [../]
 [./csvoutput]
   type = CSV
   file_base = 3Dddc
   execute_on = 'INITIAL TIMESTEP_END'
 [../]
[]
```

6.0.2 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.

6.0.3 Results

This 3D example should produce a concentration profile similar to that presented in Figure 6.1, where several downwelling plumes of high concentration can be observed:

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

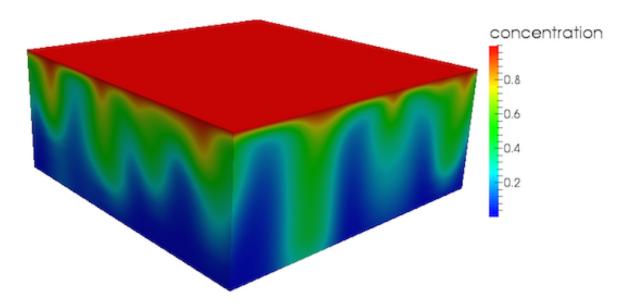


Figure 6.1: 3D concentration profile

 CO_2). This is calculated in this example file using the boundaryfluxint postprocessor in the input file, and presented in Figure 6.2.

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).

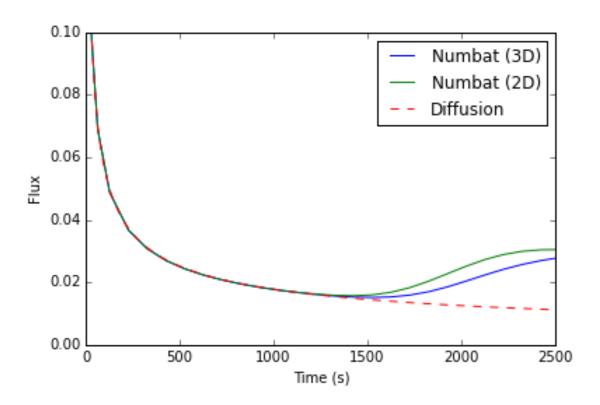


Figure 6.2: 3D flux across the top boundary

7 References

E, W. and Liu, J. G., Finite difference methods for 3D viscous incompressible flows in the vorticityvector potential formulation on nonstaggered grids, J. Comp. Phys., 138, 57-82 (1997)

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

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

CONTACT US

- 1300 363 400 $+61\ 3\ 9545\ 2176$
- enquiries@csiro.au
- w www.csiro.au

YOUR CSIRO

Australia is founding its future on science and innovation. Its national science agency, CSIRO, is a powerhouse of ideas, technologies and skills for building prosperity, growth, health and sustainability. It serves governments, industries, business and communities across the nation.

FOR FURTHER INFORMATION

CSIRO Energy Flagship

- Chris Green **t** +61 3 9545 8371
 - chris.green@csiro.au e
 - www.csiro.au/en/Research/EF