Ditto: Experimenting with Moose framework

Contents

[1 Mathematical Identities 3](#_Toc163729995)

[1.1. Continuity equation: 3](#_Toc163729996)

[1.2. Moose inner product notation: 3](#_Toc163729997)

[1.3. Integration by parts and divergence theorem 3](#_Toc163729998)

[2 How to create a the Ditto moose project 4](#_Toc163729999)

[3 Examples 5](#_Toc163730000)

[3.1. Diffusion (steady) 5](#_Toc163730001)

[3.2. ADDiffusion (steady) 7](#_Toc163730002)

[3.3. ADDiffusion (transient) 8](#_Toc163730003)

[3.4. Parameterized diffusivity 10](#_Toc163730004)

[4 Paraview notes 12](#_Toc163730005)

# Mathematical Identities

## Continuity equation:



where the first term is the rate of change of *c*, **j** is the flux and *R* the source/sink term for *c*.

## Moose inner product notation:





## Integration by parts and divergence theorem

Let *ψ* be a scalar variable and **u** a vector function. The divergence of their produce is:



By integrating over a domain Ω and rearranging we get:



The divergence theorem transforms a volume integral into a surface integral:



with being an outward normal vector on surface **. By combining eqs and we get:



# How to create a the Ditto moose project

1. Activate moose

conda activate moose

2. Generate the project files w/ stork

cd ~/projects

./moose/scripts/stork.sh Ditto

3. (Optional) Create GitHub repo called Ditto and push the project

cd ~/projects/Ditto

git remote add origin [git@github.com:[insert\_name]/Ditto.git](mailto:git@github.com:[insert_name]/Ditto.git)

git push -u origin main

4. Make the executable

make –j4 # e.g., with 4 processors

5. Run the tests

./run\_tests

6. Generate a problems folder

mkdir problems

# Examples

## Diffusion (steady)

We will generate a script for steady diffusion in 2D using the Dirichlet boundary conditions *u*left = 1 and *u*right = 0.

Let’s derive the weak form in inner product notation:

Strong form:



Multiply with a test function *ψ* and integrate over the domain



Integrate the first term by parts and apply the divergence theorem (eq. with **u** → ):



Expressing in the inner product notation:



We will set the domain to from -10, +10 along the *x* and *y* directions.

The problems is solved with the script ex01\_diffusion.i:

[Mesh]

type = GeneratedMesh

dim = 2

nx = 40

ny = 40

ymin = -10.0

ymax = 10.0

xmin = -10.0

xmax = 10.0

elem\_type = QUAD4

[]

[Variables]

[uu]

order = FIRST

family = LAGRANGE

[]

[]

[Kernels]

[diff]

type = Diffusion

variable = uu

[]

[]

[BCs]

[left\_uu]

type = DirichletBC

variable = uu

boundary = 'left'

value = 1

[]

[right\_uu]

type = DirichletBC

variable = uu

boundary = 'right'

value = 0

[]

[]

[Executioner]

type = Steady

solve\_type = 'PJFNK'

[]

[Outputs]

execute\_on = 'timestep\_end'

exodus = true

[]

We can run the script with the command:

./run.sh ex01\_diffusion.i

Paraview output:

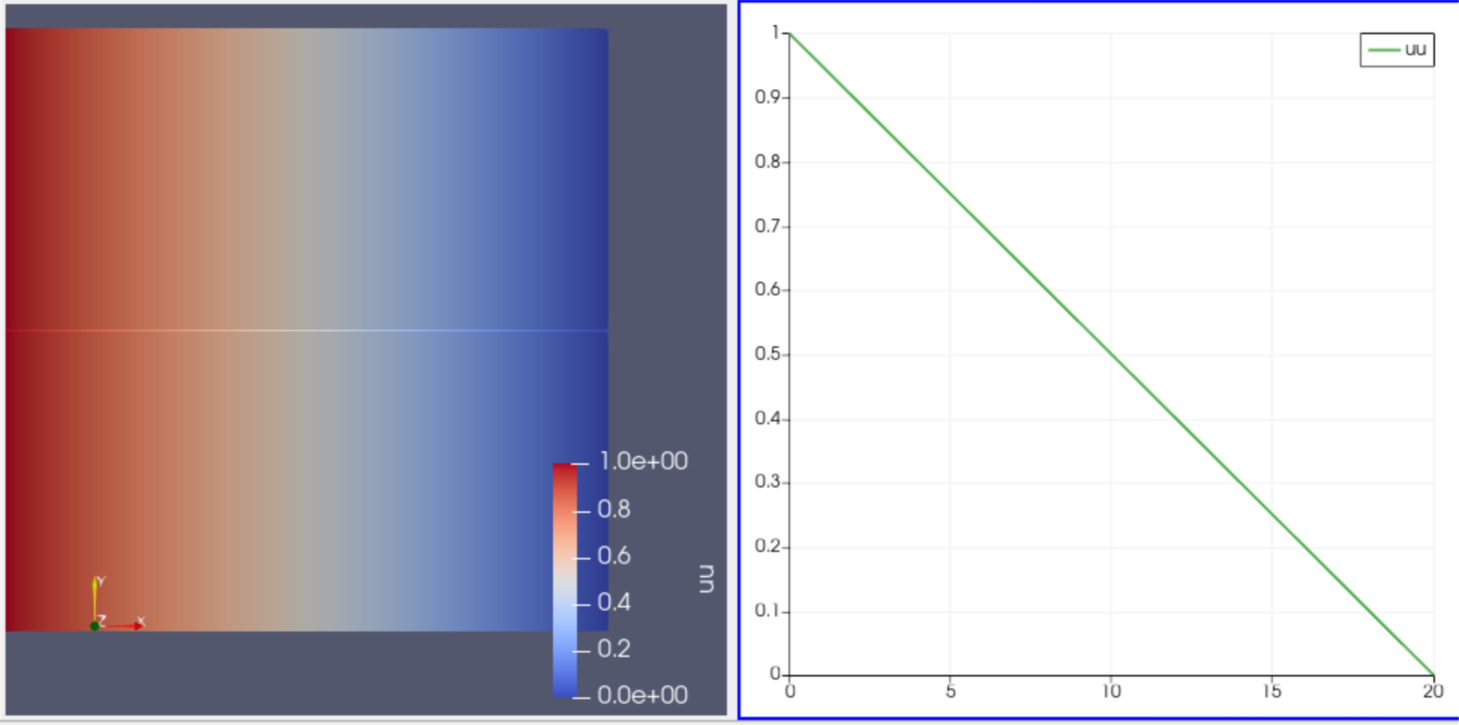
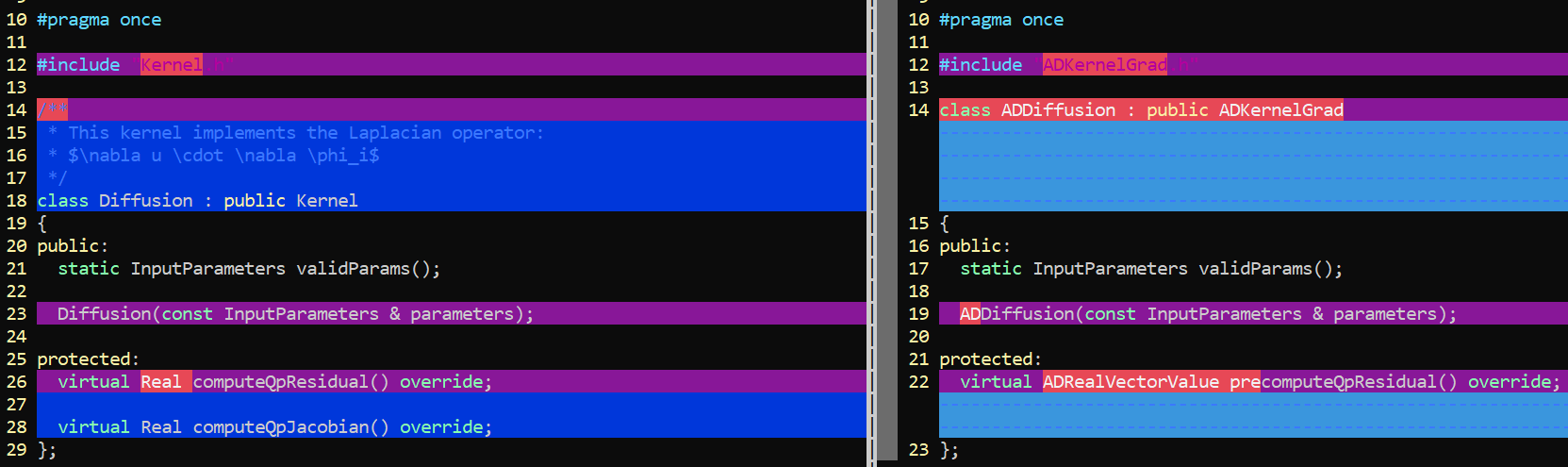


Figure 1. output from exodus file ex01\_diffusion\_out.e

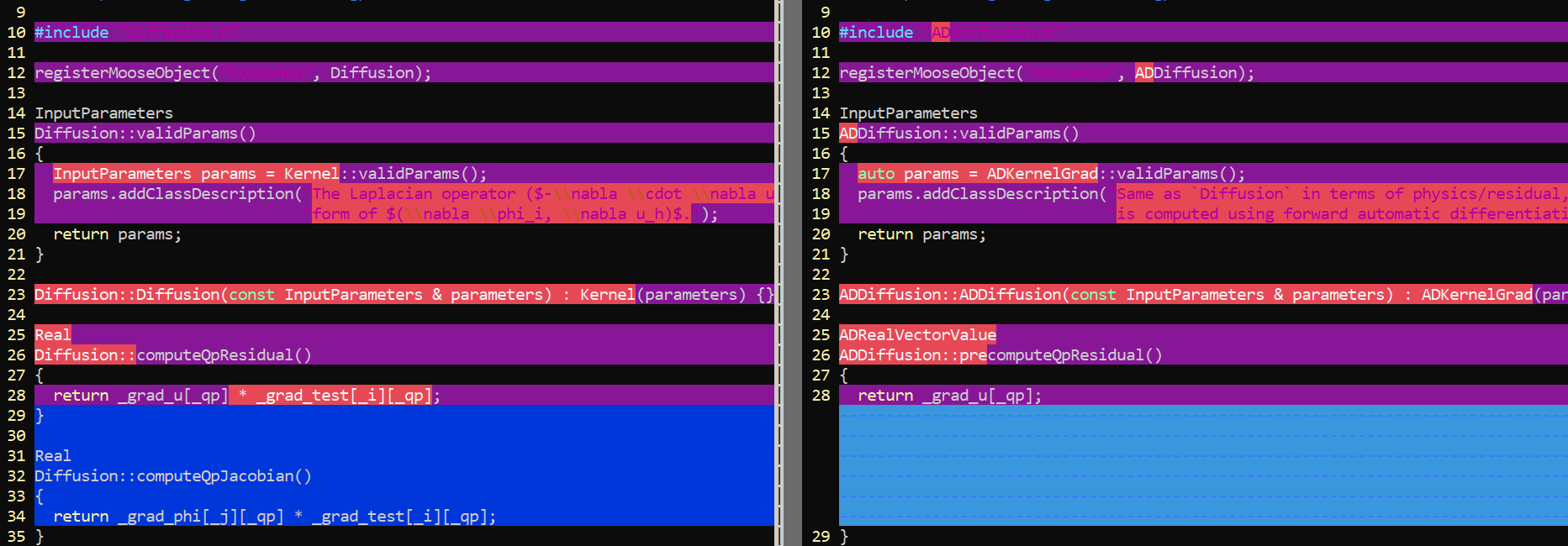
## ADDiffusion (steady)

Same with example 3.1 but we will use automatic differentiation.

The implementations of the kernels in path are shown below:









**Notes**

* In ADDiffusion the specification of the Jacobian is not required.
* The test function has been already multiplied in the precomputeQpResidual.

## ADDiffusion (transient)

Let’s derive the weak form in inner product notation:

Strong form:



Multiply with a test function *ψ* and integrate over the domain



Integrate the first term by parts and apply the divergence theorem (eq. with **u** → ):



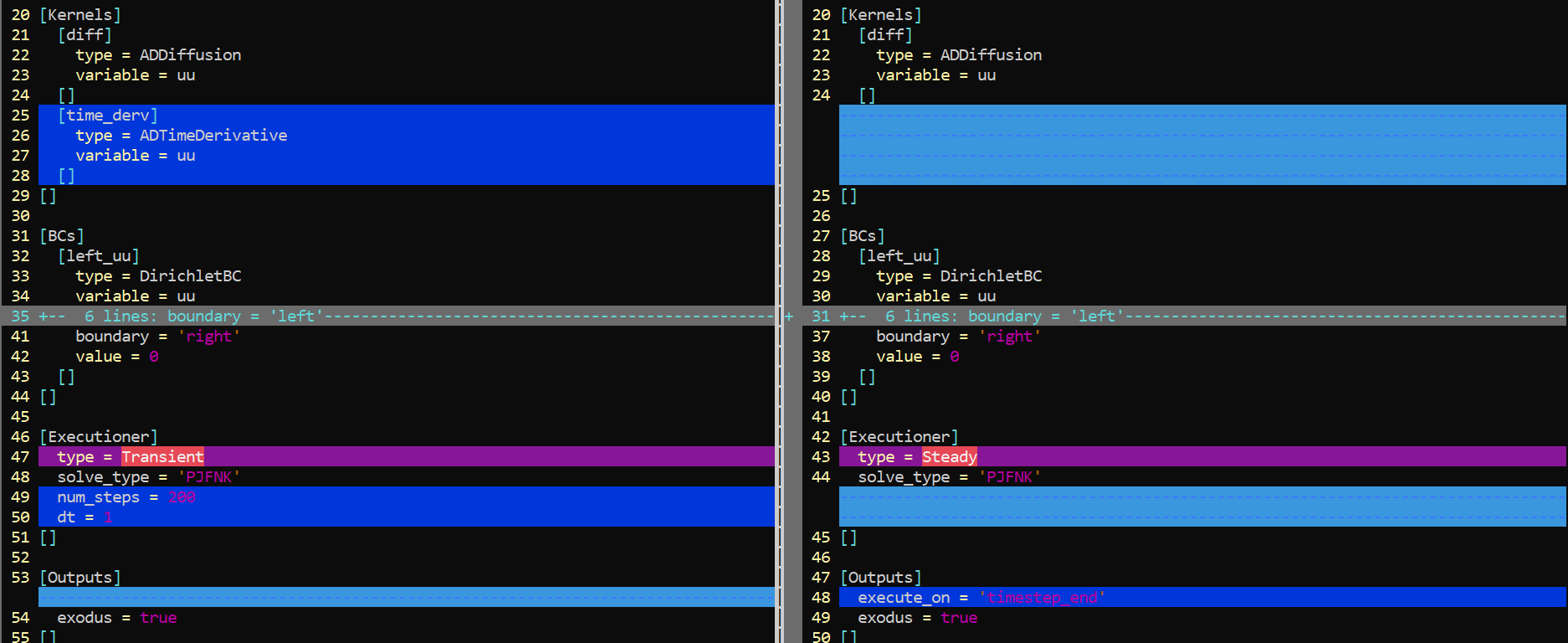
Expressing in the inner product notation:



We will set the domain to from -10, +10 along the *x* and *y* directions.

The kernel in invoked in example ex03\_transient.i.

Difference wrt example 3.2:



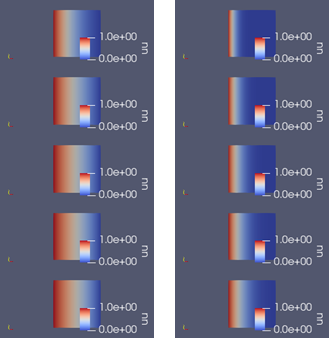


**Notes:**

* Add kernel ADTimeDerivative
* Change executioner to Transient

**Output:**

The evolution of the solution is shown in the left panel of **Figure 2**.



**Figure 2.** Diffused quantity *uu* in steps 0, 40, 80, 120, 160 and 200. Diffusion coefficient = (left) 1 and (right) 0.1.

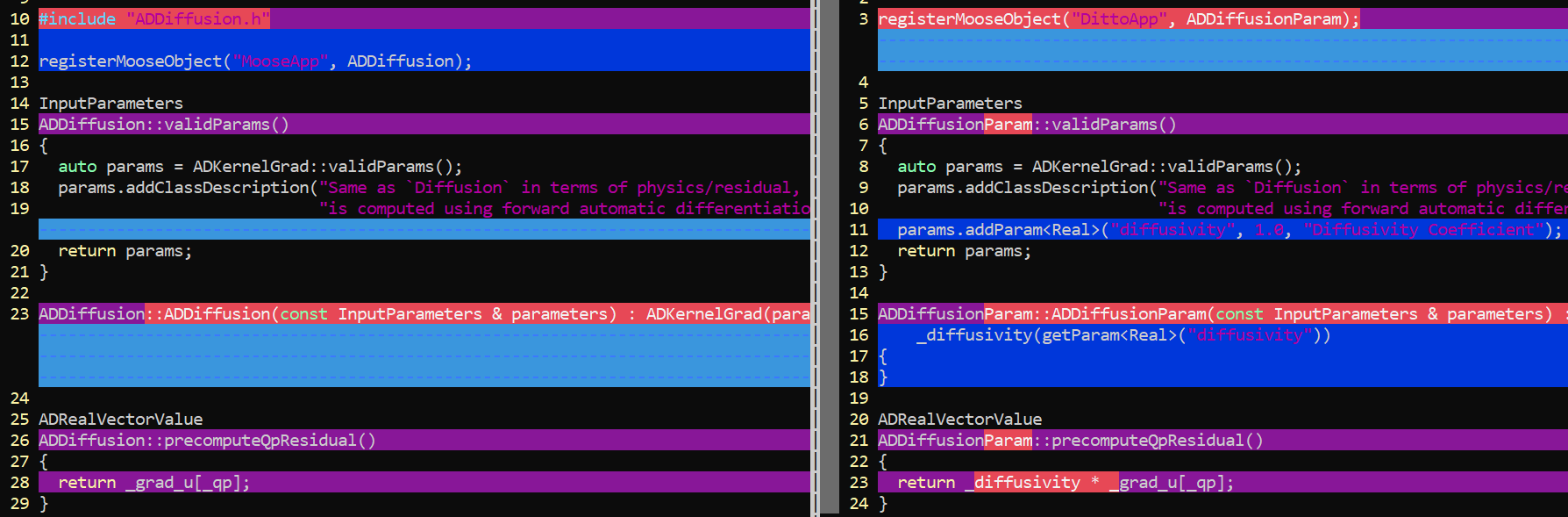
## Parameterized diffusivity

Same with example 3.3 but we will allow for varying the diffusion coefficient.

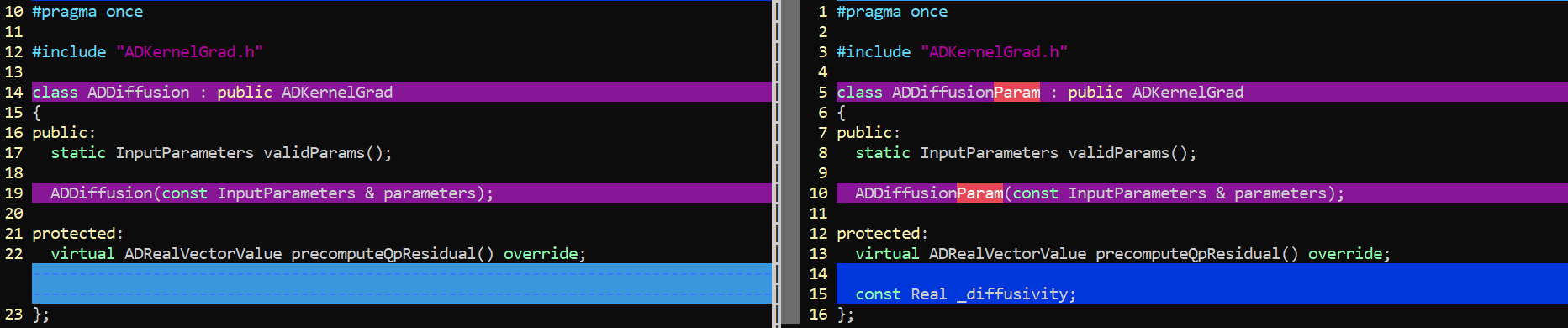
* The following files have been added in src/kernel/ADDiffusionParam.C and include/kernel/ADDiffusionParam.h
* The source was recompiled (Make –j6)

The files are similar with the default ADDiffusion kernels in moose framework with the exception that the diffusion coefficient has been made a parameter.

Comparisons are shown below:









**Output:**

The evolution of the solution is shown in the right panel of **Figure 2**.

# Paraview notes

TODO