

# PETSc

## Portable, Extensible Toolkit for Scientific Computation

Karl Rupp

`me@karlrupp.net`

Freelance Computational Scientist

Workshop on Modern Scientific Computing

DTU Compute, Technical University of Denmark

March 2-3, 2016



# Step 1: Setup

## Initialize Environment

```
$> git clone https://github.com/karlrupp/dtu2016_2.git  
  
$> cd dtu2016_2  
$> make  
$> ./pbratu
```

## Options

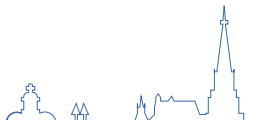
Provided via `petscrc`



### Poisson Equation

$$-\Delta u = 0$$

```
$> git checkout -f step-2  
$> make  
$> ./pbratu
```



## Step 3: Poisson Equation with Nonlinearity

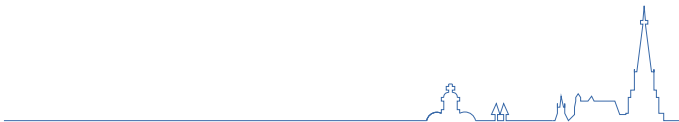
### Poisson Equation with Nonlinearity

$$-\Delta u - \lambda e^u = 0$$

```
$> git checkout -f step-3
$> make
$> ./pbratu
```

### Options

Provided via `petscrc`



## Preparation

Use the following options in `petscsrc`:

```
-da_grid_x 80  
-da_grid_y 80  
-ksp_max_it 500
```



## Preparation

Use the following options in `petscrc`:

```
-da_grid_x 80  
-da_grid_y 80  
-ksp_max_it 500
```

## Question

Determine critical parameter  $\lambda_{\text{crit}}$  (divergence)

Accuracy: One decimal after comma, e.g. 4.2



## Step 4: p-Bratu Equation

### p-Bratu Equation

$$-\nabla \cdot (\eta \nabla u) - \lambda e^u - f = 0$$
$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{p-2}{2}} \quad \gamma(u) = \frac{1}{2} |\nabla u|^2$$

```
$> git checkout -f step-4
$> make
$> ./pbratu
```

### Options

Provided via `petscrc`



## Step 5: Providing Jacobian

### p-Bratu Equation

$$-\nabla \cdot (\eta \nabla u) - \lambda e^u - f = 0$$
$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{p-2}{2}} \quad \gamma(u) = \frac{1}{2} |\nabla u|^2$$

```
$> git checkout -f step-5
$> make
$> ./pbratu
```

### Use 'simpler' Jacobians for Newton

Just use  $-\Delta w - e^u w$  (simplest)

More detail: Include  $\eta$ , but not  $\eta'$





## Play with Parameters

Check out the following options (one at a time, `-snes_monitor`):

```
-snes_mf  
-snes_fd  
-jtype 1 -myJ  
-jtype 2 -myJ
```

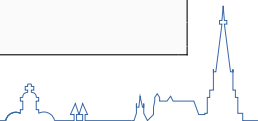
## Look at Scalability

Adjust

```
-da_grid_x 40  
-da_grid_y 40
```

Go up to (don't use `-snes_fd`)

```
-da_grid_x 160  
-da_grid_y 160
```



## Step 6: Full Implementation

### p-Bratu Equation

$$-\nabla \cdot (\eta \nabla u) - \lambda e^u - f = 0$$
$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{p-2}{2}} \quad \gamma(u) = \frac{1}{2} |\nabla u|^2$$

```
$> git checkout -f step-6
$> make
$> ./pbratu
```

### Two more implementations for Jacobian

- jtype 3: **Jacobian on 5-star stencil**
- jtype 4: **Full Jacobian**



## Preparation

Use the following options in `petscrc`:

```
-da_grid_x 320  
-da_grid_y 320  
-p 3  
-lambda 2
```

## Task

Find the fastest set of parameters (see `petscrc`)

Use `-log_summary`

Experiment with

- CPU vs. GPU

- Jacobian Matrices (`-jtype`)

- Linear solvers (KSP)

- Preconditioners (PC)



## PETSc can help You

solve algebraic and DAE problems in your application area  
rapidly develop efficient parallel code, can start from examples  
develop new solution methods and data structures  
debug and analyze performance  
advice on software design, solution algorithms, and performance

```
petsc-{users,dev,maint}@mcs.anl.gov
```

## You can help PETSc

report bugs and inconsistencies, or if you think there is a better way  
tell us if the documentation is inconsistent or unclear  
consider developing new algebraic methods as plugins, contribute if your  
idea works

