# Abstract

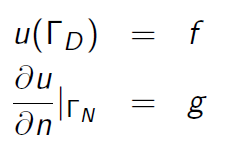
# Introduction

# Objectives

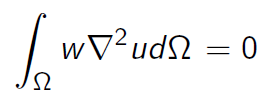
# Theory

## Derivation of Weak Form

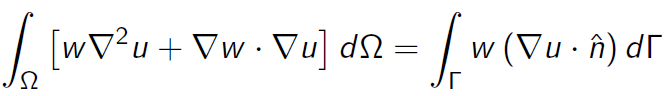
The strong form of the potential flow equation is given by (1) with boundary conditions (2):

(1)  
  (2)

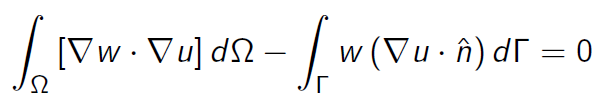
Multiplying by a weighting function and integrating over the domain yields (3):

 (3)

Using Green’s theorem and substitution of similar problem variables yields (4):

 (4)

To satisfy the potential equation, the term in (4) is zero; rearranging yields (5), the weak form of (1):

 (5)

## Stability of Method

For a non-dimensionalized unit mass, kinetic energy is given by (6), and for the whole domain by (7):

(6)

(7)

A perturbation to the potential flow *w*, adds additional kinetic energy to the domain, calculated in (8), (9), and (10):

(8)

(9)

(10)

The kinetic energy due to the perturbative field must be greater than zero, unless the perturbation is trivially zero. No such explicit imposition is placed on the term. Therefore minimizing the kinetic energy in the domain entails finding the configuration that sets , precisely the approach taken by FEM in (5).

## Choice of Basis Function

Figure 1 represents graphically the nodal linear hat functions: Green indicates lines in the x-y plane, red represents vertical lines (parallel to the z-axis), and blue indicates lines in a non-specific direction not in the x-y plane. All red lines are 1 unit tall, all nodes are on the x-y plane. All three elements in b) are the same identical element, with each basis function indicated for each node. The z-value of the function of a point on the plane defined by the two blue vectors indicates the value of the basis function from at a point in the x-y plane correlated with the projection of that point onto the x-y plane.

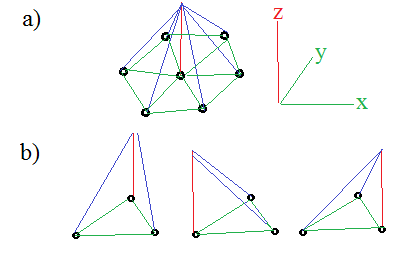
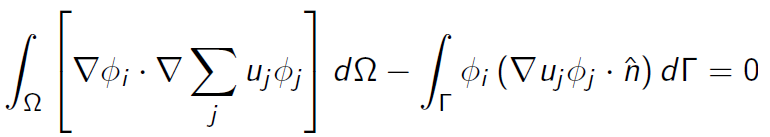


Figure : Graphical representation of linear hat functions

## Discretization of Weak Form

Equation (5) can be discretized in general by substituting in the appropriate functions for the weighting function and test function (11):

 (11)

If the basis function is described as a plane as indicated in Figure 1, its equation can be parameterized as ax+by+c = z. The divergence of this function is given by (12):

(12)

Therefore using (11) and (12) an individual node would be given by (13) for a node i and basis function from j:

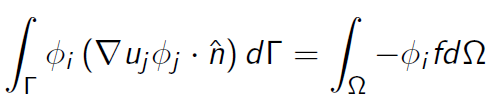
(13)

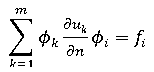
## Boundary Conditions

As given in the first equation of (1) the Dirichlet BC is an explicit setting of the value of *u* at that boundary node. This simply involves insertion of an explicit ‘1’ at Ai,i  and zeroing the rest of that row. This implicitly gives the relation as given in (14):

(14)

Neumann BCs involve more derivation, but less assembly work; At the boundary the discrete weak form from (11) with a BC ‘f’, is given by (15):

 (15)

where the weighting basis function for the BC on the boundary is 1. is precisely the Neumann BC. Examining at a particular node *i,* with‘m’ nodes connected by elements locally numbered 1-m (mappable to their actual global node numbers), using (15) and applying the relation in (13) yields (16):  (16)

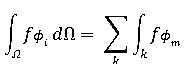
This is satisfied naturally by the FEM with no modification of the ‘A’ matrix required other than to insert the desired BC value at.

## Local Elemental Matrix Construction

Using the discrete form from (11) and applying the relation between the basis functions in (13) at an element with local node numbers 1-3 (which correspond to mappable global node numbers) yields the elemental matrix (17):

(17)

The RHS can be isolated from (15) and with some modifications can be adapted to an elemental form (18), where m indicates indexing by local node number:

 (18)

Modifying (18) gives the matrix value at a single node, using the area of a tetrahedron gives (19):

(19)

The basis function is a plane uniquely defined by the three points on its surface. This creates a system of equations that can be solved simultaneously. The solution for the basis function for local node 1 in matrix form is (20):

(20)

A set of equations exists for each of the three basis functions in the element. Each one can be solved for the ‘a’ and ‘b’ coefficients and used to calculate (17) with the gradients calculated using (13).

## Recovery of Global Matrix

The local elemental node numbers correspond to global node numbers in the ‘A’ matrix. Therefore the locally produced elemental matrix from (17) must be mapped to the correct ‘A’ rows. Rows of order are formed with their respective columns dictated by .

For instance suppose local i=1 node corresponds to 12, i=2 to 14, and i=3 to 27. Then the second row of (17) would be inserted into A(14,12), A(14,14), and A(14,27) moving column-wise from left to right. Likewise the other two rows would be inserted into the correct row dictated by .

One important consequence of this elemental stamping procedure is that if the surface isn’t homogenously tessellated, then some rows in the A matrix will contain more entries then others if they are more connected to other local nodes as all of them will have stamped their respective entries into that nodes row.

# Application Details

## General Solution Procedure

Generate immersed shape and nodes.

Generate the domain limits and node spacing

Perform a triangular meshing of the surface

for: each element

calculate connected node numbers and x-y locations

calculate triangular bounded area

calculate planar coefficients for elemental basis functions

calculate value of gradient dot products and assemble elemental matrix

calculate elemental forcing values

stamp elemental stiffness and forcing matrices into ‘A’ and ‘F’ matrices

end

Apply Neumann BCs

Apply Dirichlet BCs

Solve matrices for ‘u’ matrix

Write solution vector to file

## Implementation

## Convergence

# Appendix A

## Makefile Entry

Place after example KSP make entries in the KSP makefile (note tabs are contextual whitespace):

2DLJJB**:** 2DLJJB.o chkopts

-${CLINKER} -o 2DLJJB 2DLJJB.o ${PETSC\_KSP\_LIB}

${RM} 2DLJJB.o

## Source Code

/\* Bevan, Josh UMass Lowell 2013-2014

\*Developed with the help of Prof. Trelles, UMass Lowell

\*In partial satisfaction of directed study 22.602 Fall 2013

\*Solves the 2D Laplacian on an unstructured grid with KSP.\*/

static char help**[]** **=** "Solves the 2D Laplacian on an unstructured grid with KSP. Bevan 2014\n\n"**;**

#include <petscksp.h>

#undef \_\_FUNCT\_\_

#define \_\_FUNCT\_\_ "main"

int main**(**int argc**,**char **\*\***args**)**

**{**

Vec x**,** b**;** /\* approx solution, RHS\*/

Mat A**;** /\* linear system matrix \*/

KSP ksp**;** /\* linear solver context \*/

PC pc**;** /\* preconditioner context \*/

PetscReal norm**;** /\* norm of solution error \*/

PetscErrorCode ierr**;**

PetscInt n **=** 81**;** /\*Number of DOFs\*/

/\*For simplicity of use the mesh input file lengths need to be manually

\*input here JJBLeft[],JJBRight[],JJBElems[], and JJBNodes[]\*/

PetscInt row**[**9**],**col**[**9**],**its**,**elnd**[**3**],**JJBElems**[**384**],**iter**=**0**,**NbElems**,**NbVertices**;**

PetscInt numint**,**xx**=**0**,**yy**=**1**,**ie**,**rstart**,**rend**,**nlocal**,**NbBound**=**3**,**JJBLeft**[**3**],**JJBRight**[**3**];**

PetscScalar value**[**9**],**n1**[**2**],**n2**[**2**],**n3**[**2**],**gn1**[**2**],**gn2**[**2**],**gn3**[**2**],**A11**,**A22**,**A33**,**A12**,**A13**,**A23**;**

PetscScalar JJBNodes**[**162**],**numscal**,**TriArea**;**

PetscScalar LeftBC**[**3**],** RightBC**[**3**];** /\*Dirichlet BC values, top

\*and bottom domains implicitly Neumann=0\*/

PetscViewer viewer**;**

PetscInitialize**(&**argc**,&**args**,(**char**\*)**0**,**help**);**

ierr **=** PetscOptionsGetInt**(NULL,**"-sizer"**,&**n**,NULL);**CHKERRQ**(**ierr**);**

/\*----------------------------------------------------------------------\*/

/\* Create vectors. Create one vector and duplicate as needed.

\* The second argument to VecSetSizes() below causes PETSc to decide

\* how many elements per processor are assigned \*/

ierr **=** VecCreate**(**PETSC\_COMM\_WORLD**,&**x**);**CHKERRQ**(**ierr**);**

ierr **=** VecSetSizes**(**x**,**PETSC\_DECIDE**,**n**);**CHKERRQ**(**ierr**);**

ierr **=** VecSetFromOptions**(**x**);**CHKERRQ**(**ierr**);**

ierr **=** VecDuplicate**(**x**,&**b**);**CHKERRQ**(**ierr**);**

/\* Find start and end mesh points on each processor for the

\* interior of the mesh. This partitioning is based upon

\* the choices made for the vector in VecSetSizes()\*/

ierr **=** VecGetOwnershipRange**(**x**,&**rstart**,&**rend**);**CHKERRQ**(**ierr**);**

ierr **=** VecGetLocalSize**(**x**,&**nlocal**);**CHKERRQ**(**ierr**);**

/\*----------------------------------------------------------------------\*/

/\* Create matrix A with MatCreate(); matrix format can be

\* specified at runtime

\* Use nlocal as the local size of the matrix, this ensures it will

\* align with the vectors from above \*/

ierr **=** MatCreate**(**PETSC\_COMM\_WORLD**,&**A**);**CHKERRQ**(**ierr**);**

ierr **=** MatSetSizes**(**A**,**nlocal**,**nlocal**,**n**,**n**);**CHKERRQ**(**ierr**);**

ierr **=** MatSetFromOptions**(**A**);**CHKERRQ**(**ierr**);**

ierr **=** MatSetUp**(**A**);**CHKERRQ**(**ierr**);**

/\*----------------------------------------------------------------------\*/

/\*Read in mesh \*/

/\*Read in elem/node associations\*/

FILE **\***file **=** fopen**(**"JJBElems"**,** "r"**);**

**while(**fscanf**(**file**,** "%d"**,** **&**numint**)** **==** 1**)** **{**

JJBElems**[**iter**]** **=** numint**;**

iter**++;**

**}**

NbElems **=** iter**/**3**;**

fclose**(**file**);**

/\*Read in vertex coordinates\*/

FILE **\***file2 **=** fopen**(**"JJBNodes"**,** "r"**);**

iter**=**0**;**

**while(**fscanf**(**file2**,** "%lf"**,** **&**numscal**)** **==** 1**)** **{** /\*Note %lf for doubles to match PetscScalar \*/

JJBNodes**[**iter**]** **=** numscal**;**

iter**++;**

**}**

NbVertices **=** iter**/**2**;**

fclose**(**file2**);**

/\*Read in Left BC\*/

FILE **\***file3 **=** fopen**(**"JJBLeft"**,** "r"**);**

iter**=**0**;**

**while(**fscanf**(**file3**,** "%d"**,** **&**numint**)** **==** 1**)** **{**

JJBLeft**[**iter**]** **=** numint**-**1**;**

iter**++;**

**}**

NbBound **=** iter**;**

fclose**(**file3**);**

/\*Read in Right BC\*/

FILE **\***file4 **=** fopen**(**"JJBRight"**,** "r"**);**

iter**=**0**;**

**while(**fscanf**(**file4**,** "%d"**,** **&**numint**)** **==** 1**)** **{**

JJBRight**[**iter**]** **=** numint**-**1**;**

iter**++;**

**}**

fclose**(**file4**);**

/\*----------------------------------------------------------------------\*/

/\* Assemble matrix.

The linear system is distributed across the processors by

chunks of contiguous rows, which correspond to contiguous

sections of the mesh on which the problem is discretized.

For matrix assembly, each processor contributes entries for

the part that it owns locally. \*/

/\* From manual: The routine MatSetValuesBlocked() may offer much

\* better efficiency for users of block sparse formats

\* (MATSEQBAIJ and MATMPIBAIJ) \*/

**for** **(**ie**=**0**;**ie**<**NbElems**-**1**;**ie**+=**3**){**

ierr **=** PetscPrintf**(**PETSC\_COMM\_WORLD**,**"JJB: ie: %D\n"**,**ie**);**CHKERRQ**(**ierr**);**

/\*For each element find the associated node numbers \*/

elnd**[**0**]** **=** JJBElems**[**ie**]-**1**;**

elnd**[**1**]** **=** JJBElems**[**ie**+**1**]-**1**;**

elnd**[**2**]** **=** JJBElems**[**ie**+**2**]-**1**;**

/\* Determine element node locations \*/

n1**[**xx**]** **=** JJBNodes**[**2**\***elnd**[**0**]];**

n1**[**yy**]** **=** JJBNodes**[(**2**\***elnd**[**0**])+**1**];**

n2**[**xx**]** **=** JJBNodes**[**2**\***elnd**[**1**]];**

n2**[**yy**]** **=** JJBNodes**[(**2**\***elnd**[**1**])+**1**];**

n3**[**xx**]** **=** JJBNodes**[**2**\***elnd**[**2**]];**

n3**[**yy**]** **=** JJBNodes**[(**2**\***elnd**[**2**])+**1**];**

/\*Calculate bounded area\*/

TriArea **=** abs**((** **(**n1**[**xx**]\*(**n2**[**yy**]-**n3**[**yy**]))** **+** **(**n2**[**xx**]\*(**n3**[**yy**]-**n1**[**yy**]))** **+** **(**n3**[**xx**]\*(**n1**[**yy**]-**n2**[**yy**]))** **)/**2**);**

/\*Calculate gradients:

\* without the benefit of a low overhead linear solver these

\* were algebraicly solved. The equivalent Matlab code is:

Plane = [X0 Y0 1;...

X1 Y1 1;...

X2 Y2 1];

Gradient(:,0) = Plane\[1;0;0];

Gradient(:,1) = Plane\[0;1;0];

Gradient(:,2) = Plane\[0;0;1];

Gradient(3,:) = 0; %Remove unwanted c coeff \*/

**if** **(**n2**[**yy**]!=**n3**[**yy**]){**

gn1**[**xx**]** **=** **-**1**/(** **(** **(**n2**[**xx**]-**n3**[**xx**])** **\*** **(**n1**[**yy**]-**n3**[**yy**])** **/** **(**n2**[**yy**]-**n3**[**yy**])** **)** **-** **(**n1**[**xx**]-**n3**[**xx**])** **);**

**}else{**

gn1**[**xx**]** **=** 0**;}**

**if** **(**n2**[**xx**]!=**n3**[**xx**]){**

gn1**[**yy**]** **=** **-**1**/(** **(** **(**n2**[**yy**]-**n3**[**yy**])** **\*** **(**n1**[**xx**]-**n3**[**xx**])** **/** **(**n2**[**xx**]-**n3**[**xx**])** **)** **-** **(**n1**[**yy**]-**n3**[**yy**])** **);**

**}else{**

gn1**[**yy**]** **=** 0**;}**

**if** **(**n1**[**yy**]!=**n3**[**yy**]){**

gn2**[**xx**]** **=** **-**1**/(** **(** **(**n1**[**xx**]-**n3**[**xx**])** **\*** **(**n2**[**yy**]-**n3**[**yy**])** **/** **(**n1**[**yy**]-**n3**[**yy**])** **)** **-** **(**n2**[**xx**]-**n3**[**xx**])** **);**

**}else{**

gn2**[**xx**]** **=** 0**;}**

**if** **(**n1**[**xx**]!=**n3**[**xx**]){**

gn2**[**yy**]** **=** **-**1**/(** **(** **(**n1**[**yy**]-**n3**[**yy**])** **\*** **(**n2**[**xx**]-**n3**[**xx**])** **/** **(**n1**[**xx**]-**n3**[**xx**])** **)** **-** **(**n2**[**yy**]-**n3**[**yy**])** **);**

**}else{**

gn2**[**yy**]** **=** 0**;}**

**if** **(**n1**[**yy**]!=**n2**[**yy**]){**

gn3**[**xx**]** **=** **-**1**/(** **(** **(**n1**[**xx**]-**n2**[**xx**])** **\*** **(**n3**[**yy**]-**n2**[**yy**])** **/** **(**n1**[**yy**]-**n2**[**yy**])** **)** **-** **(**n3**[**xx**]-**n2**[**xx**])** **);**

**}else{**

gn3**[**xx**]** **=** 0**;}**

**if** **(**n1**[**xx**]!=**n2**[**xx**]){**

gn3**[**yy**]** **=** **-**1**/(** **(** **(**n1**[**yy**]-**n2**[**yy**])** **\*** **(**n3**[**xx**]-**n2**[**xx**])** **/** **(**n1**[**xx**]-**n2**[**xx**])** **)** **-** **(**n3**[**yy**]-**n2**[**yy**])** **);**

**}else{**

gn3**[**yy**]** **=** 0**;}**

/\*Calculate local elemental matrix\*/

A11 **=** TriArea **\*** **(**gn1**[**xx**]\***gn1**[**xx**]** **+** gn1**[**yy**]\***gn1**[**yy**]);**

A22 **=** TriArea **\*** **(**gn2**[**xx**]\***gn2**[**xx**]** **+** gn2**[**yy**]\***gn2**[**yy**]);**

A33 **=** TriArea **\*** **(**gn3**[**xx**]\***gn3**[**xx**]** **+** gn3**[**yy**]\***gn3**[**yy**]);**

A12 **=** TriArea **\*** **(**gn1**[**xx**]\***gn2**[**xx**]** **+** gn1**[**yy**]\***gn2**[**yy**]);**

A13 **=** TriArea **\*** **(**gn1**[**xx**]\***gn3**[**xx**]** **+** gn1**[**yy**]\***gn3**[**yy**]);**

A23 **=** TriArea **\*** **(**gn2**[**xx**]\***gn3**[**xx**]** **+** gn2**[**yy**]\***gn3**[**yy**]);**

/\*Create value array to be stamped\*/

value**[**0**]** **=** A11**;**

value**[**1**]** **=** A12**;**

value**[**2**]** **=** A13**;**

value**[**3**]** **=** A12**;**

value**[**4**]** **=** A22**;**

value**[**5**]** **=** A23**;**

value**[**6**]** **=** A13**;**

value**[**7**]** **=** A23**;**

value**[**8**]** **=** A33**;**

/\*Create global node index arrays for stamping\*/

row**[**0**]** **=** elnd**[**0**];**

row**[**1**]** **=** elnd**[**1**];**

row**[**2**]** **=** elnd**[**2**];**

row**[**3**]** **=** elnd**[**0**];**

row**[**4**]** **=** elnd**[**1**];**

row**[**5**]** **=** elnd**[**2**];**

row**[**6**]** **=** elnd**[**0**];**

row**[**7**]** **=** elnd**[**1**];**

row**[**8**]** **=** elnd**[**2**];**

/\*-\*/

col**[**0**]** **=** elnd**[**0**];**

col**[**1**]** **=** elnd**[**0**];**

col**[**2**]** **=** elnd**[**0**];**

col**[**3**]** **=** elnd**[**1**];**

col**[**4**]** **=** elnd**[**1**];**

col**[**5**]** **=** elnd**[**1**];**

col**[**6**]** **=** elnd**[**2**];**

col**[**7**]** **=** elnd**[**2**];**

col**[**8**]** **=** elnd**[**2**];**

/\* Stamp in local elemental matrix into global matrix of form: \*/

/\*

A(N1, N1) = A(N1, N1) + A\_elemental(1,1);

A(N2, N1) = A(N2, N1) + A\_elemental(2,1);

A(N3, N1) = A(N3, N1) + A\_elemental(3,1);

A(N1, N2) = A(N1, N2) + A\_elemental(1,2);

A(N2, N2) = A(N2, N2) + A\_elemental(2,2);

A(N3, N2) = A(N3, N2) + A\_elemental(3,2);

A(N1, N3) = A(N1, N3) + A\_elemental(1,3);

A(N2, N3) = A(N2, N3) + A\_elemental(2,3);

A(N3, N3) = A(N3, N3) + A\_elemental(3,3);\*/

ierr **=** MatSetValues**(**A**,**9**,**row**,**9**,**col**,**value**,**ADD\_VALUES**);**CHKERRQ**(**ierr**);**

**}**

/\* Assemble the matrix \*/

ierr **=** MatAssemblyBegin**(**A**,**MAT\_FINAL\_ASSEMBLY**);**CHKERRQ**(**ierr**);**

ierr **=** MatAssemblyEnd**(**A**,**MAT\_FINAL\_ASSEMBLY**);**CHKERRQ**(**ierr**);**

/\*----------------------------------------------------------------------\*/

/\* Apply BCs

\* MatZeroRows allows easy application of Dirichlet BCs by zeroing all entries

\* except the main diag (i.e. the self-reference needed to enforce BCs \*/

ierr **=** VecSet**(**b**,**0**);**CHKERRQ**(**ierr**);**

**for(**iter**=**0**;**iter**<**NbBound**;**iter**++){**

LeftBC**[**iter**]** **=** 2**;**

RightBC**[**iter**]** **=** 1**;**

**}**

ierr **=** MatZeroRows**(**A**,**NbBound**,**JJBLeft**,**1**,**0**,**0**);**CHKERRQ**(**ierr**);**

ierr **=** VecSetValues**(**b**,**NbBound**,**JJBLeft**,**LeftBC**,**INSERT\_VALUES**);**CHKERRQ**(**ierr**);**

ierr **=** MatZeroRows**(**A**,**NbBound**,**JJBRight**,**1**,**0**,**0**);**CHKERRQ**(**ierr**);**

ierr **=** VecSetValues**(**b**,**NbBound**,**JJBRight**,**RightBC**,**INSERT\_VALUES**);**CHKERRQ**(**ierr**);**

/\*----------------------------------------------------------------------\*/

ierr **=** KSPCreate**(**PETSC\_COMM\_WORLD**,&**ksp**);**CHKERRQ**(**ierr**);** /\* Create linear solver context \*/

/\* Set operators. A also serves as preconditioning matrix \*/

ierr **=** KSPSetOperators**(**ksp**,**A**,**A**,**DIFFERENT\_NONZERO\_PATTERN**);**CHKERRQ**(**ierr**);**

/\* Set linear solver defaults for problem.

Extract KSP and PC contexts from the KSP context,

to directly call any KSP and PC routines to set options \*/

ierr **=** KSPGetPC**(**ksp**,&**pc**);**CHKERRQ**(**ierr**);**

ierr **=** PCSetType**(**pc**,**PCJACOBI**);**CHKERRQ**(**ierr**);**

ierr **=** KSPSetTolerances**(**ksp**,**1.e-7**,**PETSC\_DEFAULT**,**PETSC\_DEFAULT**,**PETSC\_DEFAULT**);**CHKERRQ**(**ierr**);**

/\* Set runtime options,

-ksp\_type <type> -pc\_type <type> -ksp\_monitor -ksp\_rtol <rtol>

These options will override those specified above as long as

KSPSetFromOptions() is called \_after\_ any other customization

routines \*/

ierr **=** KSPSetFromOptions**(**ksp**);**CHKERRQ**(**ierr**);**

ierr **=** KSPSolve**(**ksp**,**b**,**x**);**CHKERRQ**(**ierr**);** /\* Solve linear system \*/

/\*----------------------------------------------------------------------\*/

ierr **=** KSPView**(**ksp**,**PETSC\_VIEWER\_STDOUT\_WORLD**);**CHKERRQ**(**ierr**);** /\* View solver info \*/

/\* Check the error\*/

ierr **=** KSPGetResidualNorm**(**ksp**,&**norm**);**CHKERRQ**(**ierr**);**

ierr **=** KSPGetIterationNumber**(**ksp**,&**its**);**CHKERRQ**(**ierr**);**

ierr **=** PetscPrintf**(**PETSC\_COMM\_WORLD**,**"JJB: Norm of error %G, Iterations %D\n"**,**norm**,**its**);**CHKERRQ**(**ierr**);**

/\* Output solution vector to file for external plotting\*/

ierr **=** PetscViewerASCIIOpen**(**PETSC\_COMM\_WORLD**,**"x.output"**,&**viewer**);**CHKERRQ**(**ierr**);**

ierr **=** VecView**(**x**,**viewer**);**CHKERRQ**(**ierr**);**

/\* Free work space \*/

ierr **=** VecDestroy**(&**x**);**CHKERRQ**(**ierr**);**

ierr **=** VecDestroy**(&**b**);**CHKERRQ**(**ierr**);**

ierr **=** MatDestroy**(&**A**);**CHKERRQ**(**ierr**);**

ierr **=** KSPDestroy**(&**ksp**);**CHKERRQ**(**ierr**);**

/\* PetscFinalize() before exiting program. Provides summary

\* and diagnostic information if certain runtime options

\* are chosen (e.g., -log\_summary) \*/

ierr **=** PetscFinalize**();**

**return** 0**;**

**}**

## Mesh Generation and Node Order Conditioning

%Bevan 2014

%Simple mesh generation wrapper that ultimately uses Delaunay()

%triangulation. Ideally a mesh generator generates node

%numbering such that neighbouring nodes are close to each other

%in value.

%This script also optionally allows "scrambling" of node numbers

%To remove any "nice" banding of the A matrix to investigate

%its effect on PETSc parallel overhead

Shape **=** 1**;** %[1 0045];

DomainSize **=** 4**;**

ref **=** 2**;**

powerRef **=** 2**;**%getDiscreteGeometry is a function written by Prof. Willis UMass Lowell

**[**TRItemp**,** Nodestemp**,** Toptemp**,** Bottomtemp**,** Lefttemp**,** Righttemp**,** InnerBoundarytemp**]** **=** getDiscreteGeometry**(**Shape**,** DomainSize**,** ref**,** powerRef**);**

%Allow ranomization of node order to investigate sparse pattern

Nodes **=** Nodestemp**;**

TRI **=** TRItemp**;**

Top **=** Toptemp**;**

Bottom **=**Bottomtemp**;**

Left **=** Lefttemp**;**

Right **=** Righttemp**;**

InnerBoundary **=** InnerBoundarytemp**;**

**if** false %True for ranomization, false to leave mesh alone

Choices **=** 1**:**length**(**Nodes**);**

**for** i**=**1**:**length**(**Nodes**)**

Chooser **=** max**(**1**,**round**(**rand**(**1**)\***length**(**Choices**)));**

Chosen **=** Choices**(**Chooser**);**

Nodes**(**i**,:)** **=** Nodestemp**(**Chosen**,:);**

TRI**(**TRItemp**==**Chosen**)** **=** i**;**

Top**(**Toptemp**==**Chosen**)** **=** i**;**

Bottom**(**Bottomtemp**==**Chosen**)** **=** i**;**

Left**(**Lefttemp**==**Chosen**)** **=** i**;**

Right**(**Righttemp**==**Chosen**)** **=** i**;**

InnerBoundary**(**InnerBoundarytemp**==**Chosen**)** **=** i**;**

Choices **=** Choices**([**1**:**Chooser**-**1 **,** Chooser**+**1**:end]);**

**end**

**end**

Nodes **=** reshape**(**Nodes**',**1**,**numel**(**Nodes**))';**

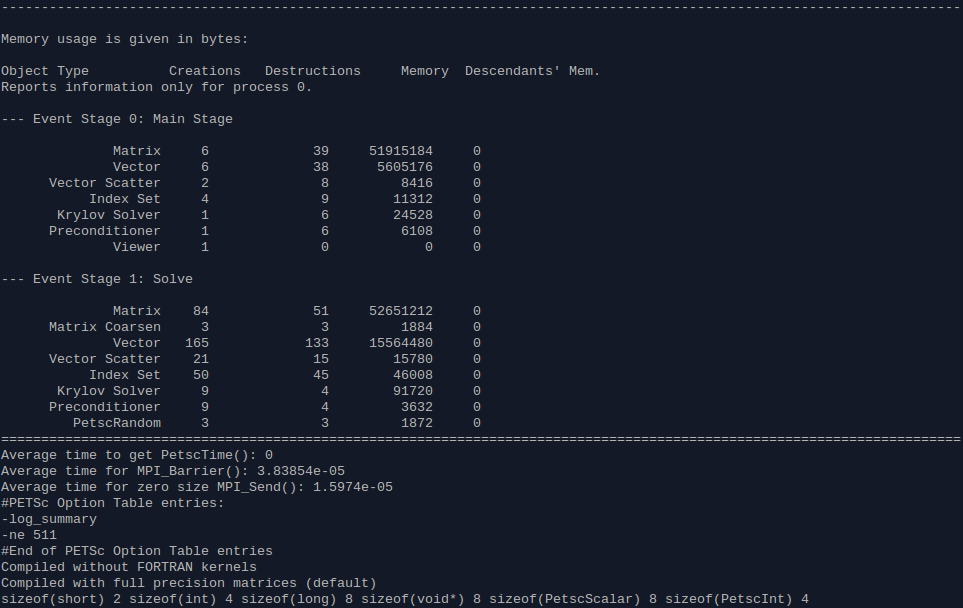
dlmwrite**(**'JJBElems'**,**reshape**(**TRI**',**1**,**numel**(**TRI**)),** 'delimiter'**,** '\n'**);**

dlmwrite**(**'JJBLeft'**,**Left**,** 'delimiter'**,** '\n'**);**

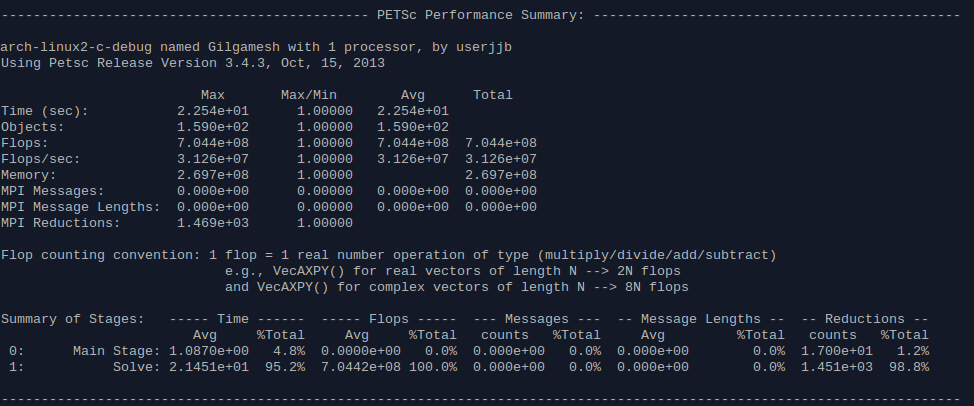
dlmwrite**(**'JJBRight'**,**Right**,** 'delimiter'**,** '\n'**);**

save**(**'JJBNodes'**,**'Nodes'**,**'-ascii'**,**'-double'**);**

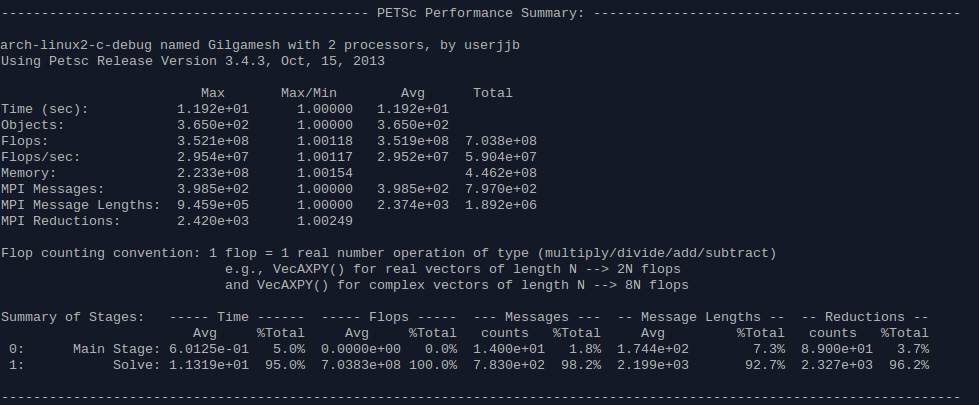
## C:\Users\jbevan\Documents\GitHub\PETSc-Directed-Study\Temp\2A.pngExample “-log\_summary” C:\Users\jbevan\Documents\GitHub\PETSc-Directed-Study\Temp\1A.png



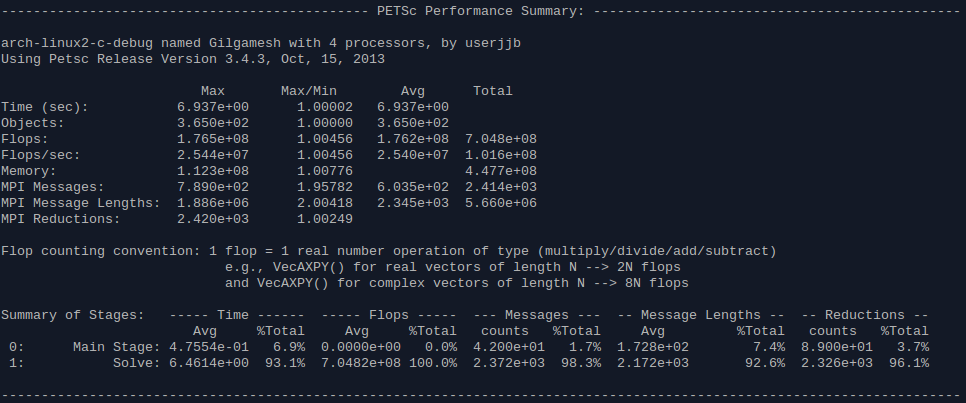
## Parallel Scalability Logs



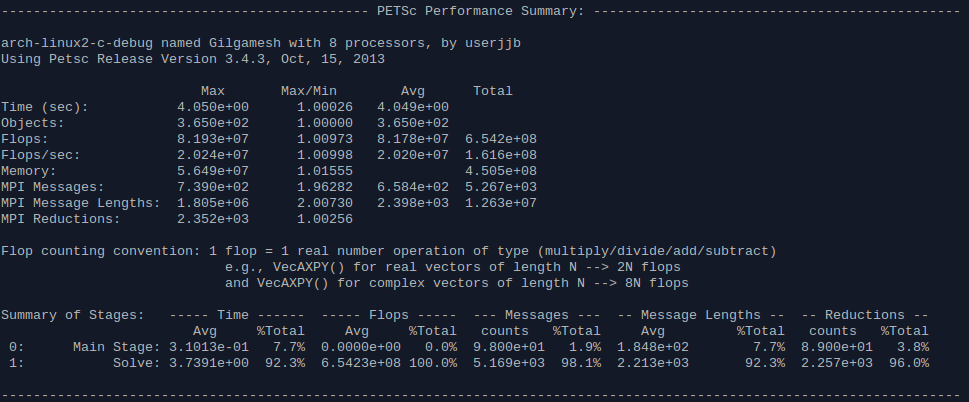
1 Processor



2 Processors



4 Processors



8 Processors