DPG Trilinos

Nate Roberts

July 14, 2011

Outline

- DPG as a method: mathematical details
- Implementation Goals
- Design Overview & User's Guide
- Selected Implementation Details
- Results
- Future Work

The DPG Method:

Introduction

Outline

DPG in brief: method and features

- *Discontinuous*: solution values can differ at element boundaries, with fluxes and traces that correspond to the difference.
- *Petrov*-Galerkin: Test and trial spaces may differ. Our test space is computed on the fly, element-locally.
- Formulate PDE as a first-order system (for stability).
- Integrate by parts to move derivatives to test functions, with flux and trace terms as additional unknowns.

DPG in brief: method and features

- Optimal test functions are computed on the fly by solving the local element problem

$$\begin{cases} \text{For basis element } u \in U, \text{find } Tu \in V \text{ such that:} \\ (Tu, v)_V = b(u, v), \quad \forall v \in V \end{cases}$$

- Key result: always delivers the *best approximation error* in the energy norm

$$\|u\|_E = \sup_{v \in V} \frac{b(u, v)}{\|v\|_V}.$$

Abstract Setting

Let U and V be real Hilbert spaces. Consider the problem

$$\text{Find } u \in U : b(u,v) = l(v) \quad \forall v \in V. \tag{1}$$

We assume that

1. $b(\cdot,\cdot)$ is *continuous*, i.e., for some real M,

$$|b(u,v)| \leq M \, ||u||_U \, ||v||_V \, ,$$

2. $b(\cdot,\cdot)$ is *weakly coercive*, that is, for some $\gamma > 0$,

$$\inf_{||u||_U=1} \sup_{||v||_V=1} b(u,v) > \gamma, \text{ and}$$

3. $b(\cdot,\cdot)$ is *non-degenerate*, that is,

$$\{v \in V : b(u,v) = 0 \quad \forall u \in U\} = \{0\}.$$

These conditions guarantee that the problem (1) has a unique solution provided that $l \in V'$, the dual of V.

Optimal Test Functions

Recall the Riesz Representation Theorem:
Let H be a Hilbert space and $L \in H^*$. Then there exists a unique $y \in H$ such that

$$Lx = (y, x) \quad \forall x \in H.$$

Note that:

- Our test space, V, is Hilbert.
- For any $u \in U$, $b(u, \cdot)$ is a functional on V.
- Therefore, the Riesz Representation Theorem guarantees the existence of $v \in V$ that represents $b(u, \cdot)$ within space V.

For $u \in U$, define Tu as the unique solution to

$$(Tu, v)_V = b(u, v) \quad \forall v \in V.$$

We call Tu an *optimal test function*.
(We call $T : U \to V$ the *optimal test function map*.)

Optimal Test Space for U_n

Take a finite-dimensional trial space $U_n \subset U$.

Define the *optimal test space* for U_n as $V_n = \text{span}\{Te_j : j = 1, \cdots, n\}$, where the e_j are a basis for U_n.

Then the resulting stiffness matrix is symmetric positive definite. We have:

$$b(e_j, Te_i) = (Te_j, Te_i)_V = (Te_i, Te_j)_V = b(e_i, Te_j).$$

Energy Norm

Define the *energy norm* on U by

$$\|u\|_E = \sup_{v \in V} \frac{b(u,v)}{\|v\|_V}.$$

- We can easily show that $\gamma \, ||u||_U \leq ||u||_E \leq M \, ||u||_U$—the energy norm is equivalent to the usual norm on U.
- The energy norm is the norm in which DPG guarantees the best approximation error.
- The energy norm depends on $(\cdot,\cdot)_V$, which we may *choose freely*.
- $\|u\|_E = \sup_{\|v\|_V=1} b(u,v) = \sup (Tu,v)_V = \frac{1}{\|Tu\|_V}(Tu,Tu)_V = \|Tu\|_V$. That is, the energy norm is generated by the inner product on V and the optimal test function map; i.e. $(u,u)_E \stackrel{\text{def}}{=} (Tu,Tu)_V$.

Optimality in the Energy Norm

Solve for $u_n \in U_n : b(u_n, v) = l(v) \forall v \in V_n$. Then the error is the best approximation error in the energy norm:

$$\|u - u_n\|_E = \inf_{w_n \in U_n} \|u - w_n\|_E$$

Practical Realization

1. Given BVP, develop mesh-dependent $b(\cdot,\cdot)$ with test space V that allows inter-element discontinuities (hence *Discontinuous* Petrov-Galerkin).
2. Choose trial space U_n.
3. Compute optimal test functions.

   Approximate T by $T_n : U_n \to \tilde{V}_n \subset V$. We use an enriched space of piecewise polynomials for $\tilde{V}_n$. Defining $t_j = T_n e_j$, we want to solve for t_j:

   $$(t_j, \tilde{e}_i)_V = b(e_j, \tilde{e}_i),$$

   where $\tilde{e}_i$ is the basis for $\tilde{V}_n$.
4. Use the optimal test functions to solve the problem on $U_n \times \tilde{V}_n$.

Poisson Formulation

We apply the DPG method to the Poisson problem in two dimensions. We begin with:

$$\nabla \cdot \nabla \phi = f.$$

We define $\psi = \nabla \phi$ and rewrite as a first-order system:

$$\nabla \phi - \psi = 0$$
$$\nabla \cdot \psi = f.$$

Poisson Formulation

Multiply by test functions and integrate:

$$\int_K \nabla\phi \cdot \mathbf{q} - \int_{\mathbf{K}} \psi \cdot \mathbf{q} = \mathbf{0}$$

$$\int_K \nabla \cdot \psi \, v = \int_K f v.$$

Integrate by parts, introducing flux $\widehat{\psi}_n$ and trace $\widehat{\phi}$ on the boundary:

$$-\int_K \phi \nabla \cdot \mathbf{q} - \int_{\mathbf{K}} \psi \cdot \mathbf{q} + \int_{\partial\mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n} = \mathbf{0}$$

$$-\int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n \, v = \int_K f v$$

Inner Product Choice 1: Mathematician's Norm

A simple choice for the test space norm: $\boldsymbol{q} \in H(\text{div})$ and $v \in H(\text{grad})$, so define

$$||(\boldsymbol{q}, v)||_V^2 \stackrel{\text{def}}{=} \int_\Omega \left((\nabla \cdot \boldsymbol{q})^2 + \boldsymbol{q} \cdot \boldsymbol{q} + \nabla v \cdot \nabla v + v^2 \right)$$

as the *mathematician's* norm on the test space.

Inner Product Choice 2: Optimal Test Norm

- We know from the analysis that DPG delivers the best solution in the *energy norm*, and that the choice of norm on the test space determines the energy norm.
- **Q:** Can we select the test space norm in such a way that the energy norm is exactly the norm of interest (L^2, in our case)?
- **A:** Yes, this is given in the abstract setting by

$$||v||_V \stackrel{\text{def}}{=} \sup_{||u||_U=1} b(u,v),$$

where $||\cdot||_U$ is the norm of interest. $||\cdot||_V$ is then called the *optimal test norm*. (Showing this is equivalent to showing the inf-sup condition for the bilinear form, i.e. proving that the variational formulation is well-posed.)

Optimal Test Norm: Derivation

Summing the equations that make up the bilinear form and grouping terms by trial space variables, we have

$$b(u,v) = \int_K (\psi \cdot (\mathbf{q} - \nabla \mathbf{v}) + \phi \nabla \cdot \mathbf{q}) + \int_{\partial K} (\widehat{\phi}\, \mathbf{q} \cdot \mathbf{n} + \widehat{\psi}_{\mathbf{n}}\, \mathbf{v})$$

Optimal Test Norm: Derivation

Applying the Cauchy-Schwarz inequality, we obtain

$$||(\boldsymbol{q}, v)||_V \stackrel{\text{def}}{=} \sup_{||u||=1} b(u, v)$$

$$= \sup_{||u||=1} \left(||\psi||_K^2 + ||\phi||_K^2 + \left|\left|\widehat{\phi}\right|\right|_{\partial K}^2 + \left|\left|\widehat{\boldsymbol{\psi}}_n\right|\right|_{\partial K}^2 \right)^{1/2} \cdot$$

$$\left(||\boldsymbol{q} - \nabla v||_K^2 + ||\nabla \cdot \boldsymbol{q}||_K^2 + ||\boldsymbol{q} \cdot \boldsymbol{n}||_{H^{-1/2}(\partial K)}^2 + ||v||_{H^{1/2}(\partial K)}^2 \right)^{1/2}$$

Since the L^2 norm

$||u|| = \left(||\psi||_K^2 + ||\phi||_K^2 + \left|\left|\widehat{\phi}\right|\right|_{\partial K}^2 + \left|\left|\widehat{\boldsymbol{\psi}}_n\right|\right|_{\partial K}^2 \right)^{1/2} = 1$, we have

$$||(\boldsymbol{q}, v)||_{V,opt}$$

$$= \left(||\boldsymbol{q} - \nabla v||_K^2 + ||\nabla \cdot \boldsymbol{q}||_K^2 + ||\boldsymbol{q} \cdot \boldsymbol{n}||_{H^{-1/2}(\partial K)}^2 + ||v||_{H^{1/2}(\partial K)}^2 \right)^{1/2}$$

Quasi-Optimal Test Norm

The optimal test norm for our Poisson formulation is given by

$$||(\boldsymbol{q}, v)||_{V,opt}$$
$$= \left(||\boldsymbol{q} - \nabla v||_K^2 + ||\nabla \cdot \boldsymbol{q}||_K^2 + ||\boldsymbol{q} \cdot \boldsymbol{n}||_{H^{-1/2}(\partial K)}^2 + ||v||_{H^{1/2}(\partial K)}^2 \right)^{1/2}$$

However, because of the boundary terms, this is not localizable, and therefore impractical for computation. We approximate the optimal test norm by a localizable norm, which we call the *quasi-optimal* test norm, given by

$$||(\boldsymbol{q}, v)||_{V,quasi} = \left(||\boldsymbol{q} - \nabla v||_K^2 + ||\nabla \cdot \boldsymbol{q}||_K^2 + \beta \left(||\boldsymbol{q} \cdot \boldsymbol{n}||_K^2 + ||v||_K^2 \right) \right)^{1/2},$$

where β is a parameter whose optimal value must be determined.

Implementation Goals

Immediate Implementation Goals

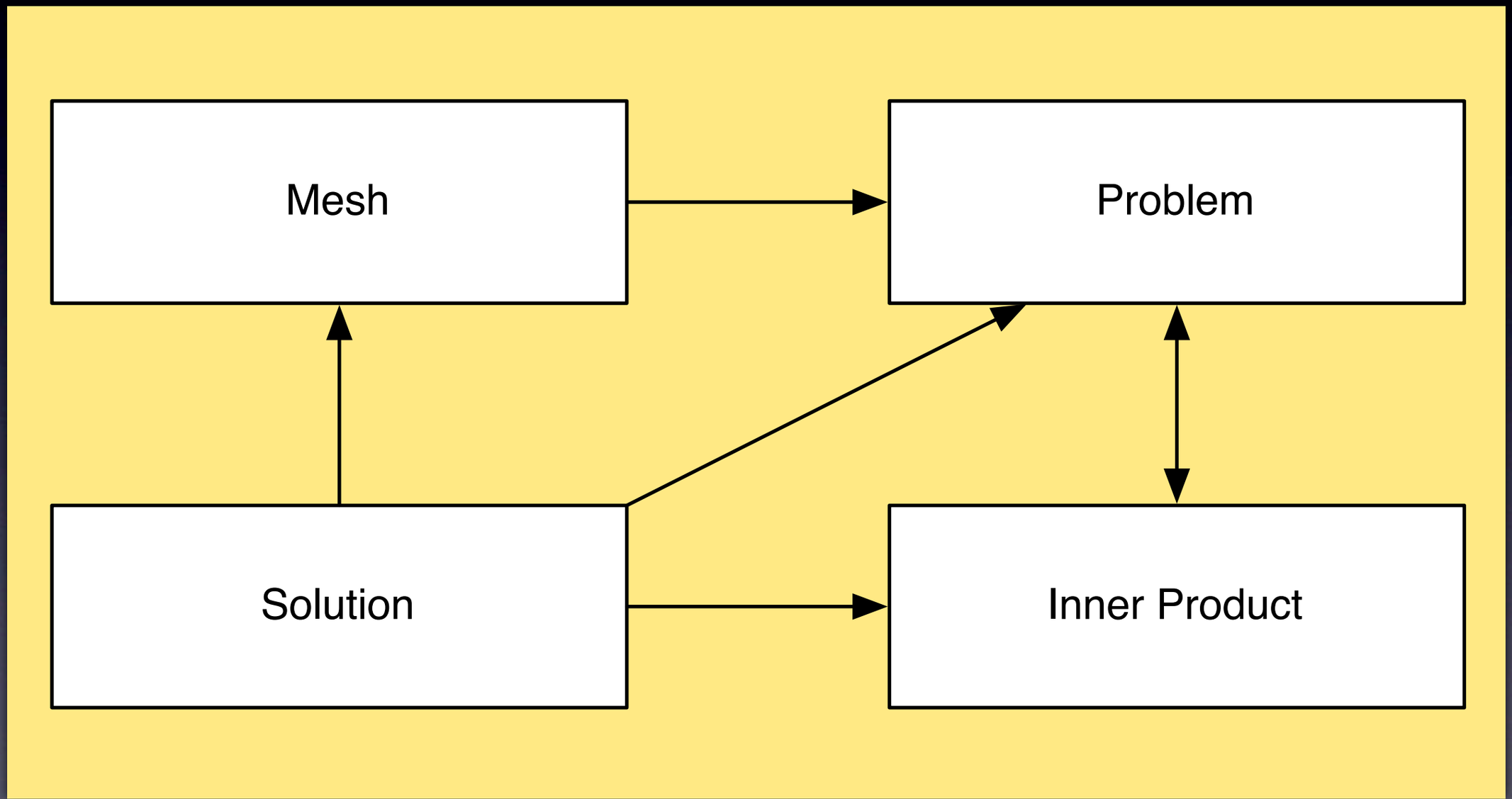
- implement last summer's 2D Stokes formulation, using Trilinos components
- Features required:
  - uniform rectangular meshes
  - mathematician's inner product
  - *optimal test inner product*
- attempt some local conservation experiments

Ultimate Implementation Goals

- support for arbitrary cell topologies (mostly there now, but untried)
- at least some parallelism
- adaptive 2D
- Newton-Raphson stepping for nonlinear problems (Navier-Stokes)
- adaptive 3D?

Design Overview & User's Guide

High-level Design



Problem

- BilinearForm: defines LHS of a first-order variational system in the continuous space
- RHS: defines the RHS of that same system
- BC: defines (Dirichlet) boundary conditions for the problem.

Poisson Example

Consider the Poisson problem:

$$\nabla \cdot \nabla \phi = f$$

Rewrite as 1st-order:

$$\psi - \nabla \phi = 0$$

$$\nabla \cdot \psi = f$$

Poisson Example

Multiply by test functions:

$$\psi \cdot \mathbf{q} - \nabla\phi \cdot \mathbf{q} = 0$$

$$\nabla \cdot \psi\, v = f v$$

Integrate by parts, introducing fluxes and traces:

$$-\int_K \psi \cdot \mathbf{q} - \int_{\mathbf{K}} \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n} = \mathbf{0}$$

$$-\int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n\, v = \int_K f v$$

Implementing a BilinearForm

1. Subclass `BilinearForm`
2. Populate `_trialIDs` and `_testIDs` vectors
3. Implement the following:
   - `trialIsBoundaryValue()`
   - `functionSpaceFor{Test|Trial}()`
   - `trialTestOperator()`
   - `applyBilinearFormData()`

Poisson Example: Trial & Test Variables

We have the following **trial** space variables:

$$\phi, \psi_1, \psi_2, \hat{\phi}, \hat{\psi}_n$$

and the **test** space variables:

$$v, \mathbf{q}$$

Populate trial and test IDs

```
PoissonBilinearForm::PoissonBilinearForm()
{
  _testIDs.push_back(Q_1);
  _testIDs.push_back(V_1);

  _trialIDs.push_back(PHI_HAT);
  _trialIDs.push_back(PSI_HAT_N);
  _trialIDs.push_back(PHI);
  _trialIDs.push_back(PSI_1);
  _trialIDs.push_back(PSI_2);
}
```

$$\phi, \psi_1, \psi_2, \hat{\phi}, \hat{\psi}_n$$

$$v, \mathbf{q}$$

Implement `trialIsBoundaryValue`

```
bool PoissonBilinearForm::trialIsBoundaryValue(int trialID) {
  if ((PHI_HAT==trialID) || (PSI_HAT_N==trialID)) {
    return true;
  } else {
    return false;
  }
}
```

$$-\int_K \psi \cdot \mathbf{q} - \int_{\mathbf{K}} \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n}$$

$$-\int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n\, v$$

Given a trialID:

- return bool indicating whether it is a boundary variable (trace or flux)

Implement `functionSpaceForTest`

```
EFunctionSpaceExtended functionSpaceForTest(int testID) {
  switch (testID) {
    case Q_1:
      return IntrepidExtendedTypes::FUNCTION_SPACE_HDIV;
    break;
    case V_1:
      return IntrepidExtendedTypes::FUNCTION_SPACE_HGRAD;
    break;
    default: throw "Error: unknown testID";
  }
}
```

Given a testID:

- return an enumeration value indicating the function space for the test function

Implement
funtionSpaceForTrial

```
EFunctionSpaceExtended
PoissonBilinearForm::functionSpaceForTrial(int trialID) {
  // Field variables, and fluxes, are all L2.
  // Traces (like PHI_HAT) are H1 if we use conforming traces.
  // For now, don't use conforming (like last summer).
  return IntrepidExtendedTypes::FUNCTION_SPACE_HVOL;
}
```

Given a trialID:

- return an enumeration value indicating the function space for the trial function

Implement `trialTestOperator`

```
bool trialTestOperator(
int trialID, int testID,
EOperatorExtended &trialOperator,
EOperatorExtended &testOperator
)
```

Given a trialID and a testID:

- return bool indicating whether they enter the bilinear form in nonzero way
- return operator (e.g. VALUE or GRAD) to apply to test and trial

Implement `trialTestOperator`

```
bool PoissonBilinearForm::trialTestOperator(…) {
  // being DPG, trialOperator will always be OPERATOR_VALUE
  trialOperator = IntrepidExtendedTypes::OPERATOR_VALUE;
  // unless we specify otherwise, trial and test don't interact:
  bool returnValue = false;
  switch (testID) {
    case Q_1:
      switch (trialID) {
        case PSI_1:
  …
```

$$-\int_K \psi \cdot \mathbf{q} - \int_{\mathbf{K}} \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n}$$

$$-\int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n \, v$$

Implement `trialTestOperator`

```
bool PoissonBilinearForm::trialTestOperator(…) {
  // being DPG, trialOperator will always be OPERATOR_VALUE
  trialOperator = IntrepidExtendedTypes::OPERATOR_VALUE;
  // unless we specify otherwise, trial and test don't interact:
  bool returnValue = false;
  switch (testID) {
    case Q_1:
      switch (trialID) {
        case PSI_1:
         returnValue = true;
         testOperator = IntrepidExtendedTypes::OPERATOR_X;
          break;
        case PSI_2:
          returnValue = true;
          testOperator = IntrepidExtendedTypes::OPERATOR_Y;
          break;
```

$$-\int_K \psi \cdot \mathbf{q} - \int_K \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n}$$

Implement `trialTestOperator`

```
bool PoissonBilinearForm::trialTestOperator(…) {
  …
  switch (testID) {
    case Q_1:
      switch (trialID) {
        …
        case PHI:
          returnValue = true;
          testOperator = IntrepidExtendedTypes::OPERATOR_DIV;
          break;
…
```

$$-\int_K \psi \cdot \mathbf{q} - \int_{\mathbf{K}} \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n}$$

Implement `trialTestOperator`

```
bool PoissonBilinearForm::trialTestOperator(…) {
  …
  switch (testID) {
    case Q_1:
      switch (trialID) {
        …
        case PHI_HAT:
          returnValue = true;
          testOperator = IntrepidExtendedTypes::OPERATOR_VALUE;
          break;
      default:
        break;
    }
    break;
```

$$-\int_K \psi \cdot \mathbf{q} - \int_{\mathbf{K}} \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n}$$

Implement `trialTestOperator`

```
bool PoissonBilinearForm::trialTestOperator(…) {
  …
  switch (testID) {
    …
    case V_1:
      switch (trialID) {
        …
        case PSI_1:
          returnValue = true;
          testOperator = IntrepidExtendedTypes::OPERATOR_DX;
          break;
        case PSI_2:
          returnValue = true;
          testOperator = IntrepidExtendedTypes::OPERATOR_DY;
          break;
        …
```

$$-\int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n \, v$$

Implement `trialTestOperator`

```
bool PoissonBilinearForm::trialTestOperator(…) {
  …
  switch (testID) {
    …
    case V_1:
      switch (trialID) {
        …
        case PSI_HAT_N:
          returnValue = true;
          testOperator = IntrepidExtendedTypes::OPERATOR_VALUE;
          break;
        default:
          break;
      }
    default:
      break;
  }
  return returnValue;
}
```

$$-\int_K \psi \cdot \nabla v + \int_{\partial K} \widehat{\psi}_n \, v$$

Implement
applyBilinearFormData

```
void applyBilinearFormData(int trialID, int testID,
     FieldContainer<double> &testTrialValuesAtPoints,
     FieldContainer<double> &points)
```

Given a trialID, testID, and spatial points:

- Apply material data from the bilinear form to the pointwise values of the test and trial functions (testTrialValuesAtPoints).

Implement
`applyBilinearFormData`

```
bool PoissonBilinearForm::applyBilinearFormData(…) {
switch (testID) {
    case Q_1:
      switch (trialID) {
        case PHI:
          // negate
          multiplyFCByWeight(testTrialValuesAtPoints,-1.0);
        break;
        …
```

$$-\int_K \psi \cdot \mathbf{q} - \int_{\mathbf{K}} \phi \nabla \cdot \mathbf{q} + \int_{\partial \mathbf{K}} \widehat{\phi}\, \mathbf{q} \cdot \mathbf{n}$$

Implementing a RHS

1. Subclass `RHS`
2. Implement the following:
   - `bool nonZeroRHS(testID)`
   - `void rhs(testID, &physicalPoints,&values)`

Implementing BCs

1. Subclass `BC`
2. Implement the following:

- `bool bcsImposed(varID)`
- `imposeBC(varID, FC<double> &physicalPoints, FC<double> &unitNormals, FC<double> &dirichletValues, FC<bool> &imposeHere)`
- `[bool singlePointBC(varID)]`

Implementing an InnerProduct

1. May not need to:
   `MathInnerProduct(bilinearForm);`
   `OptimalInnerProduct(bilinearForm);`

2. If you do need to, subclass DPGInnerProduct and implement:

   - `void operators(testID1, testID2, vector &testOp1,vector &testOp2)`
   - `void applyInnerProductData(FC<double> &testValues, testID1, testID2, operatorIndex,`

Building a Mesh

- currently, supports uniform rectangular meshes, but other meshes can easily fit into the data structures—mostly a matter of building appropriate constructors
- `Mesh(FC<double> &boundaryRect, int horizontalElements, int verticalElements, bilinearForm, int pTrial, int pTest);`

Solving

- `Solution solution(mesh, bc, rhs, ip);`
- `solution.solve();`
- Also have support for exact solutions, with built-in L2 comparison of exact versus computed solution (PoissonExactSolution built using Sacado).

Selected

Implementation Details

DofOrdering class

- Each element has multiple trial and test functions, each with potentially differing basis functions.
- DofOrdering allows the specification of the various basis functions and their ordering within the local stiffness matrix
- One DofOrdering for trial, one for test

DofOrdering class: sample methods

- `addEntry(varID, RCP<Basis> basis, basisRank, sideIndex = 0);`
- `RCP<Basis> getBasis(varID, sideIndex = 0);`
- `int getDofIndex(varID, basisDofOrdinal, sideIndex=0);`

Mesh Design: Considerations

- Intrepid provides efficient batch processing of cells
- This assumes that cells are alike in the basis functions they use as well as in topology
- In an adaptive mesh, not all cells will be alike

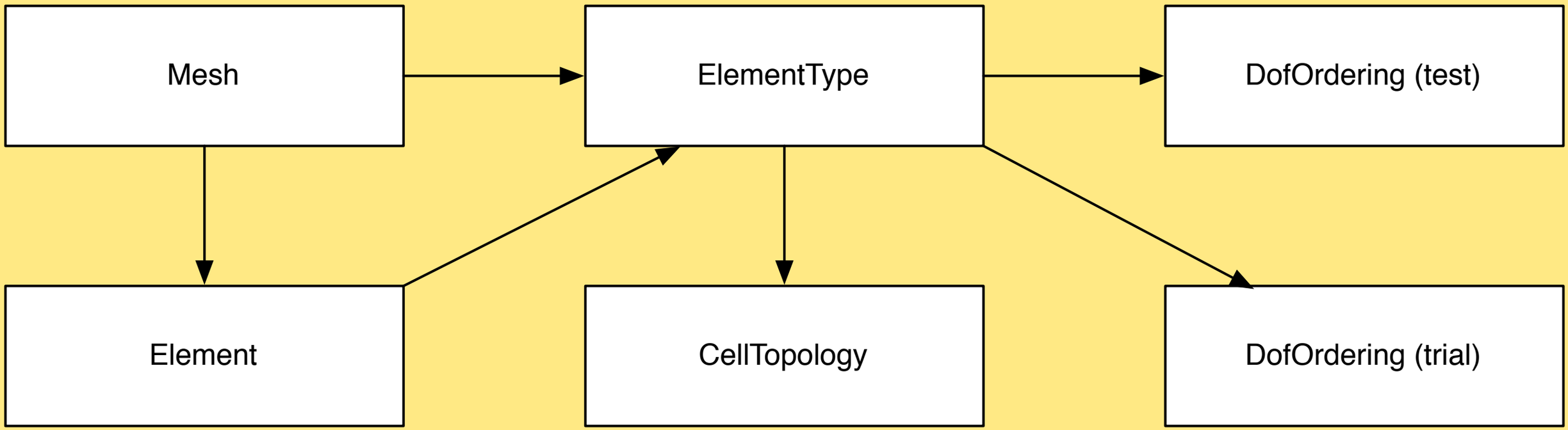
Mesh Design: ElementType

- Idea is to batch process elements by *ElementType*
- ElementType is defined by the tuple (cellTopo, trialOrdering, testOrdering)
- Mesh keeps list of cell nodes for each type

Mesh Design: ElementType

- To make storage and lookup of ElementTypes efficient, implement factories for ElementType, DofOrdering, and Basis
- Factories return RCPs, and guarantee uniqueness of the returned object

Mesh Design: Classes



Adaptivity Design: p-refinements

- When an element is refined (has p increased), it may disagree with its neighbors' trial and flux basis functions
- Strategy: simply take the finer choice at each edge, and use that as the basis

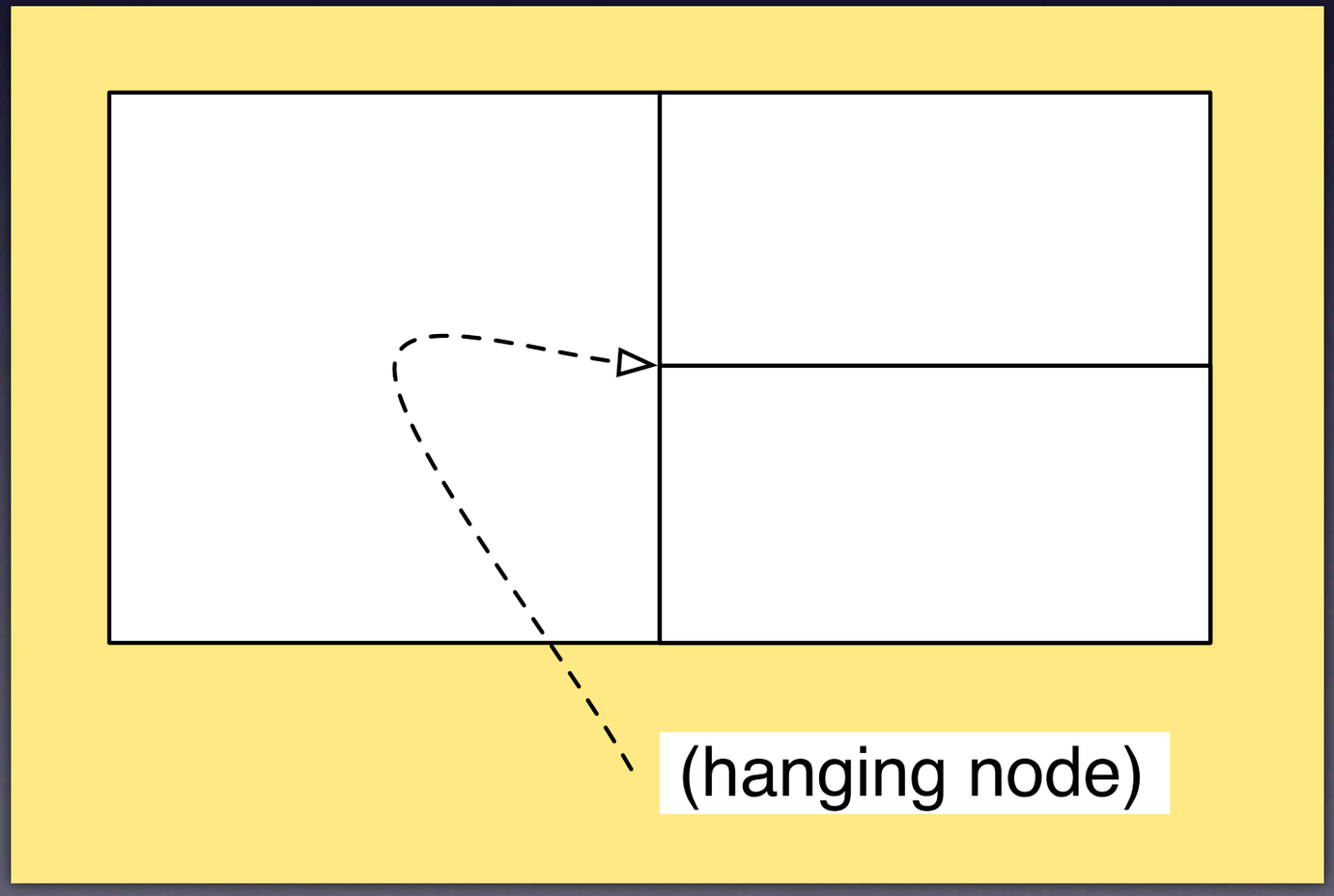
Adaptivity Design: p-refinements

3-step process to refine the mesh:

1. Identify cells to refine, and assign new DofOrderings
2. Identify their neighbors, and modify DofOrderings at shared sides
3. Rebuild mesh data structures

Adaptivity Design: h-refinements

The challenge with h-refinements is similar, and has to do with *hanging nodes*:



Adaptivity Design: h-refinements

- When an element is refined (has p increased), its neighbors may have extra subfaces/subedges
- Strategy: define the basis across the whole edge as the union of the two subedge bases, with support local to respective subedges

Adaptivity Design: h-refinements

- To support this, implement a MultiBasis, which is a subclass of Basis, and whose constructors take sets of Bases.

- This allows the side to appear "whole" to the unrefined element, with consequence that our integration routines do not need to treat hanging nodes specially

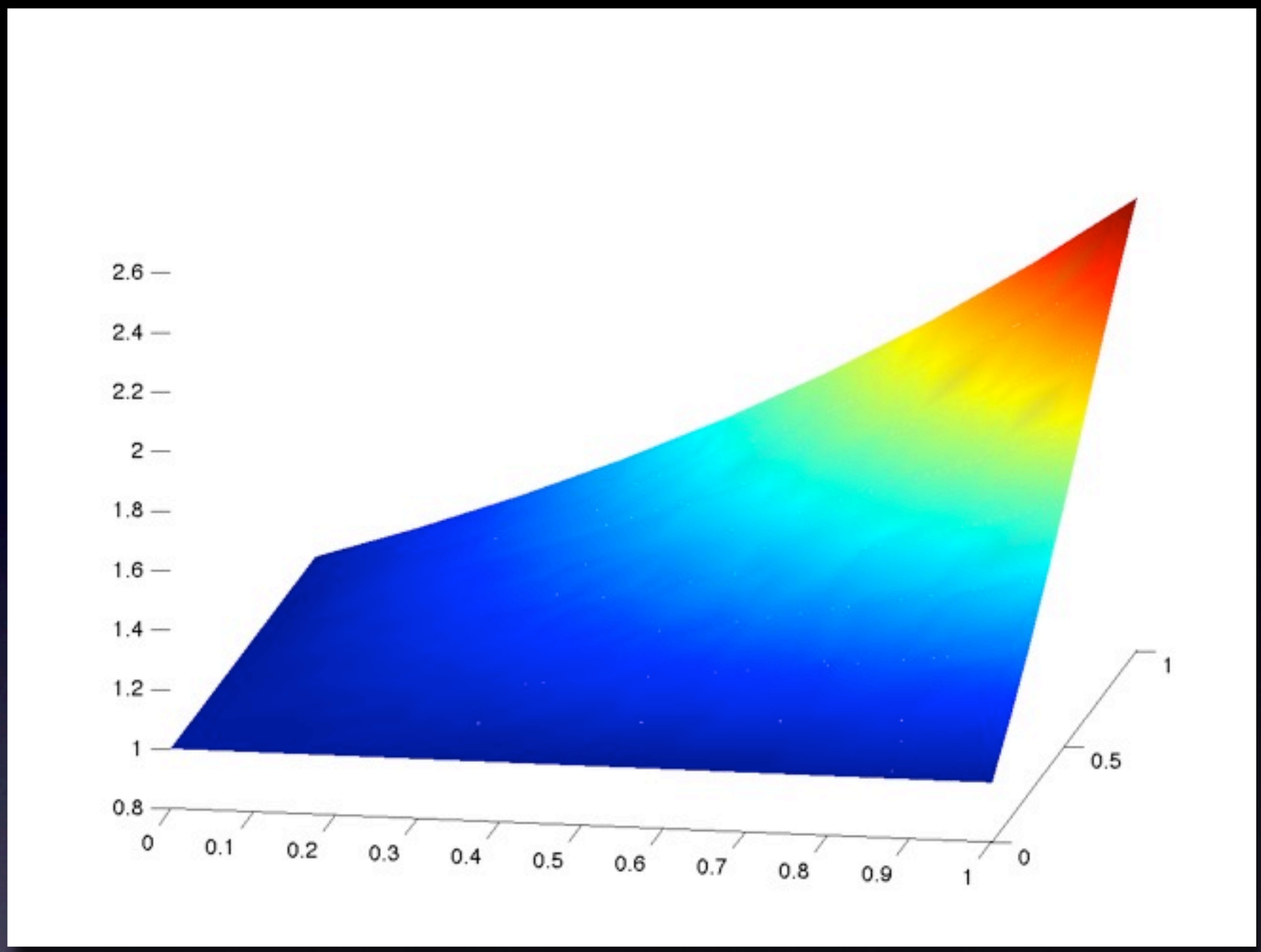
Results

Results

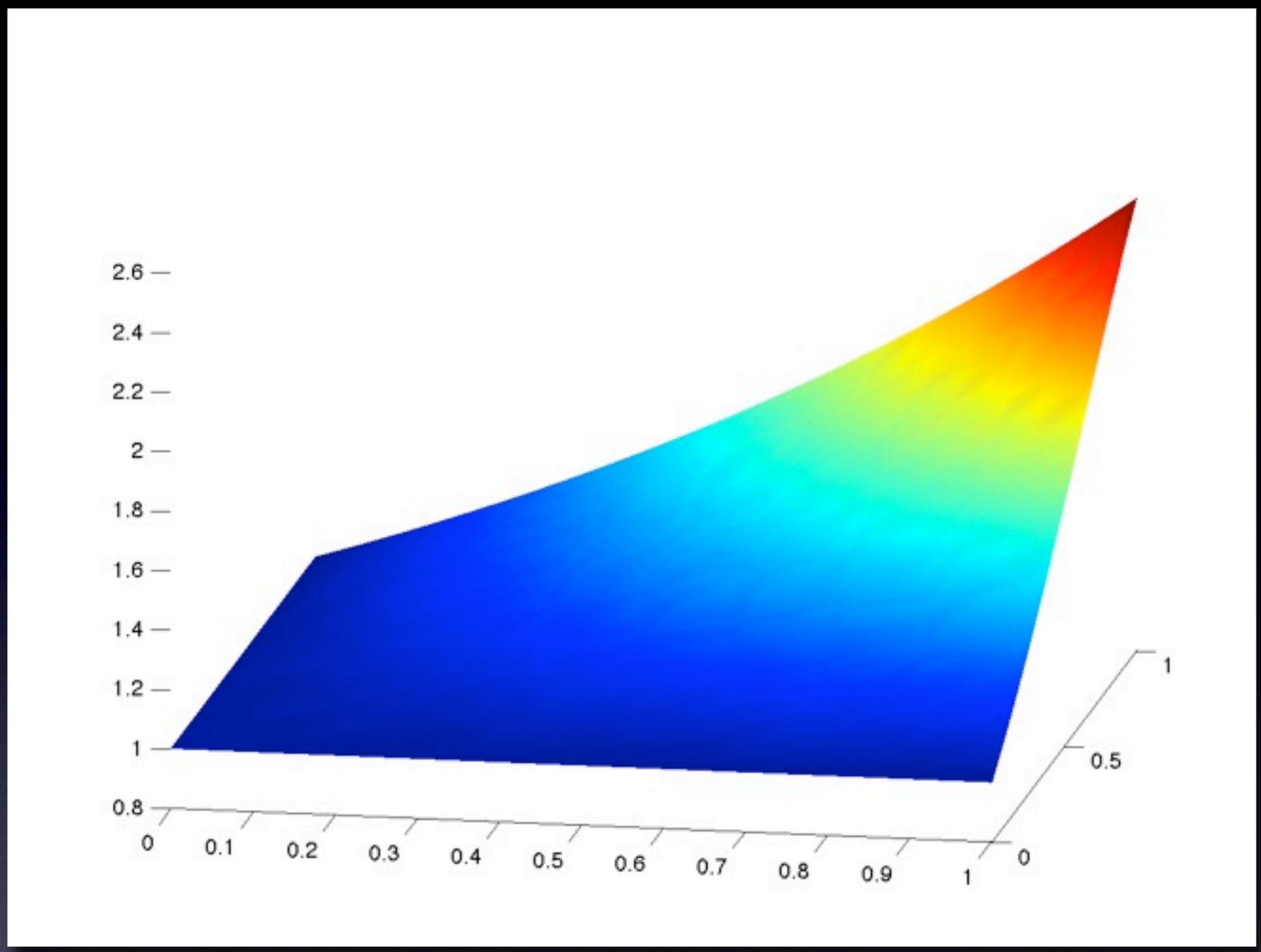
- Small proof-of-concept studies for Poisson:
  - vary p with fixed 8x8 mesh
    - h-convergence study with p=1
    - h-convergence study with p=2
- Each study used exact solution e^(x sin y), and Dirichlet conditions on both the trace and the flux (PSI_HAT_N and PHI_HAT) along the entire boundary

Study 1:

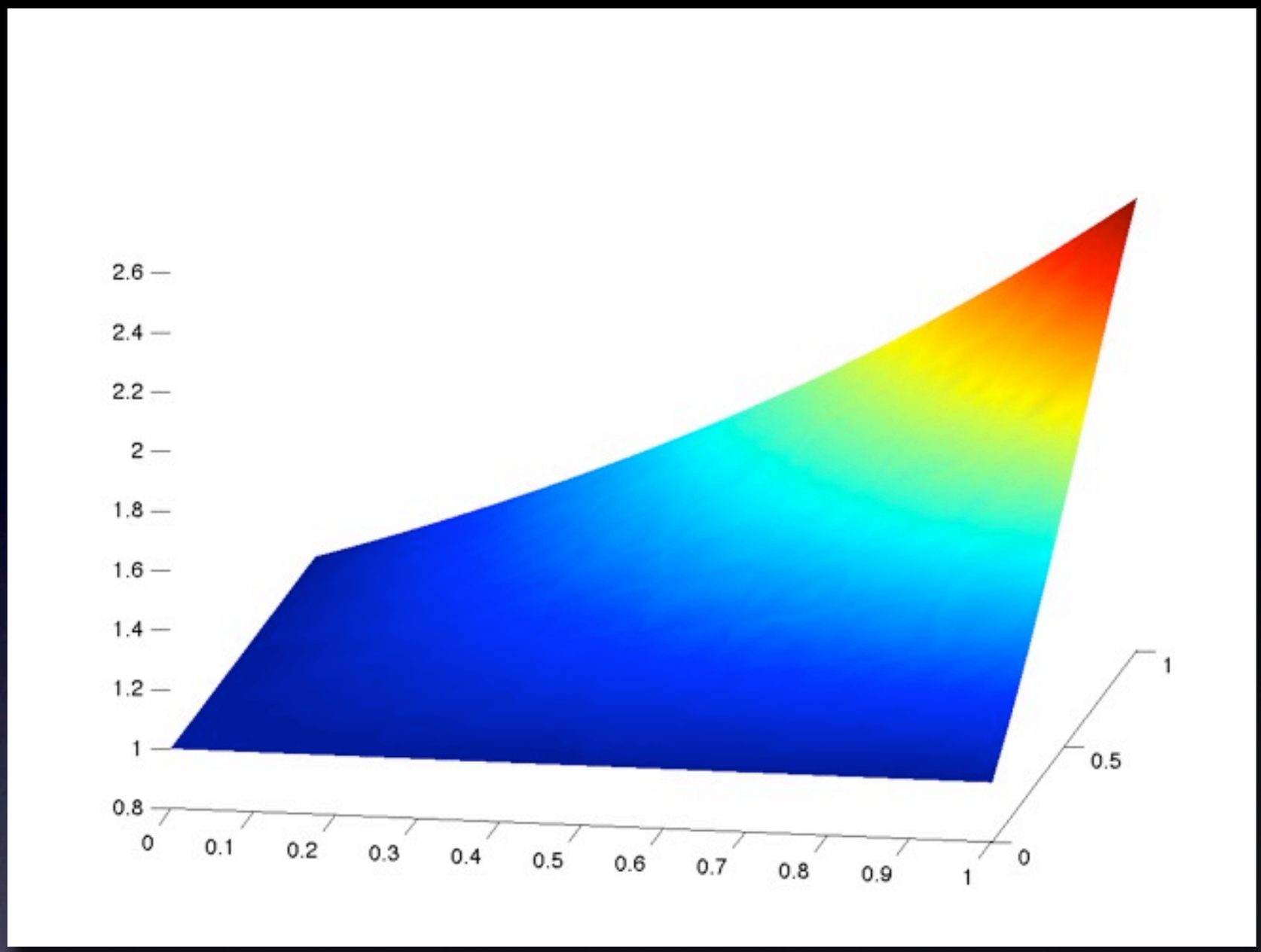
8x8 mesh, varying p



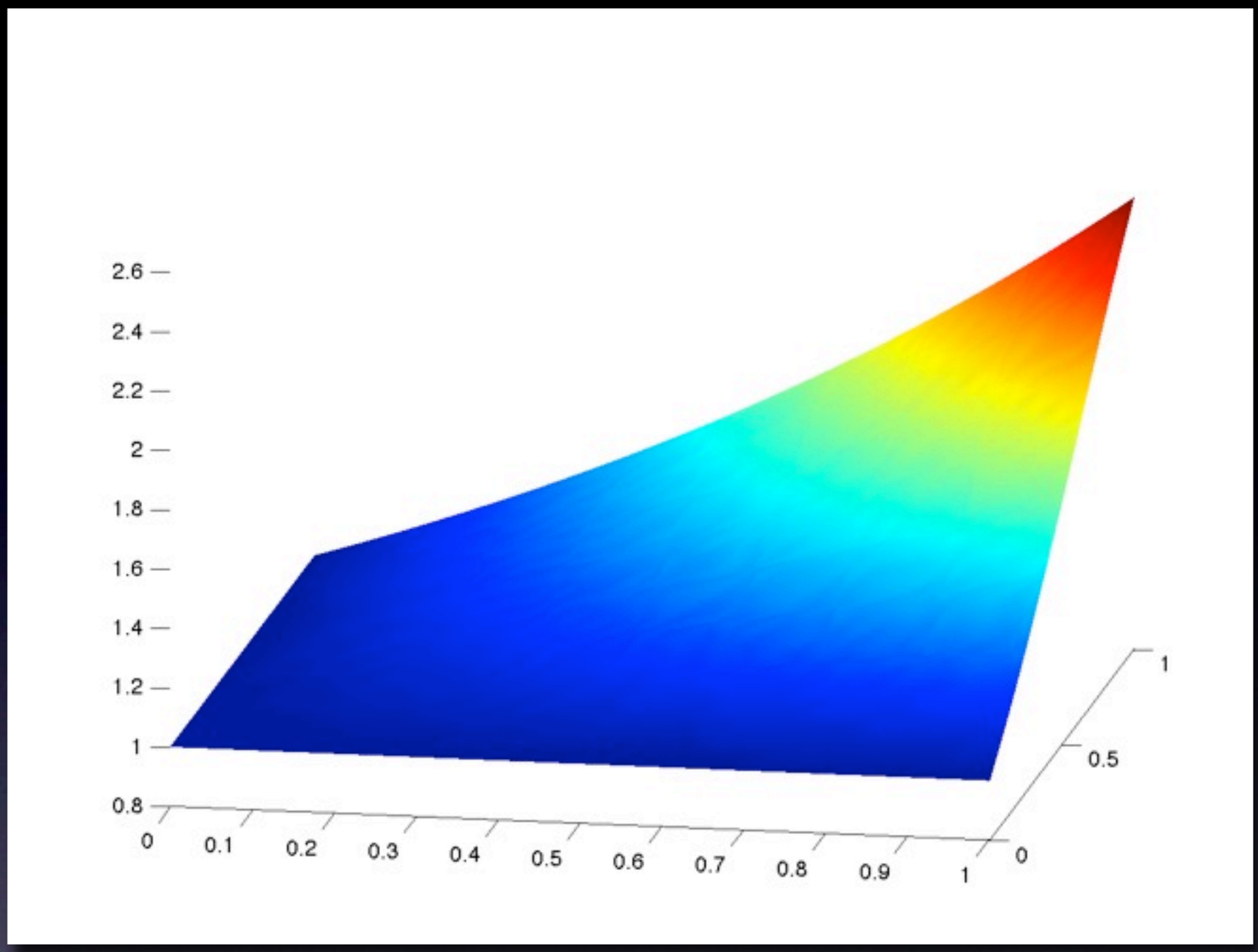
Poisson: p=1, 8x8 mesh



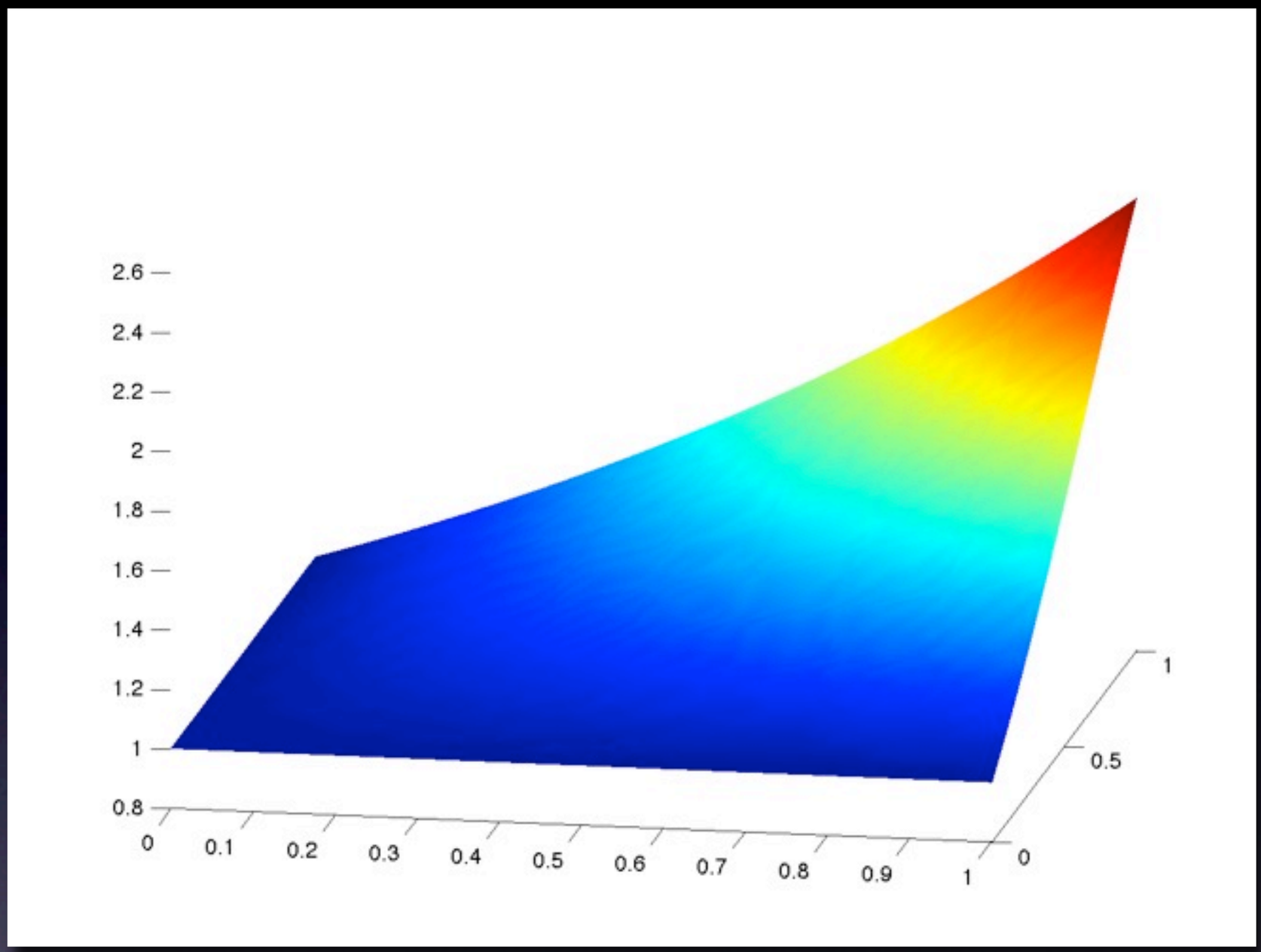
Poisson: p=2, 8x8 mesh



Poisson: p=3, 8x8 mesh



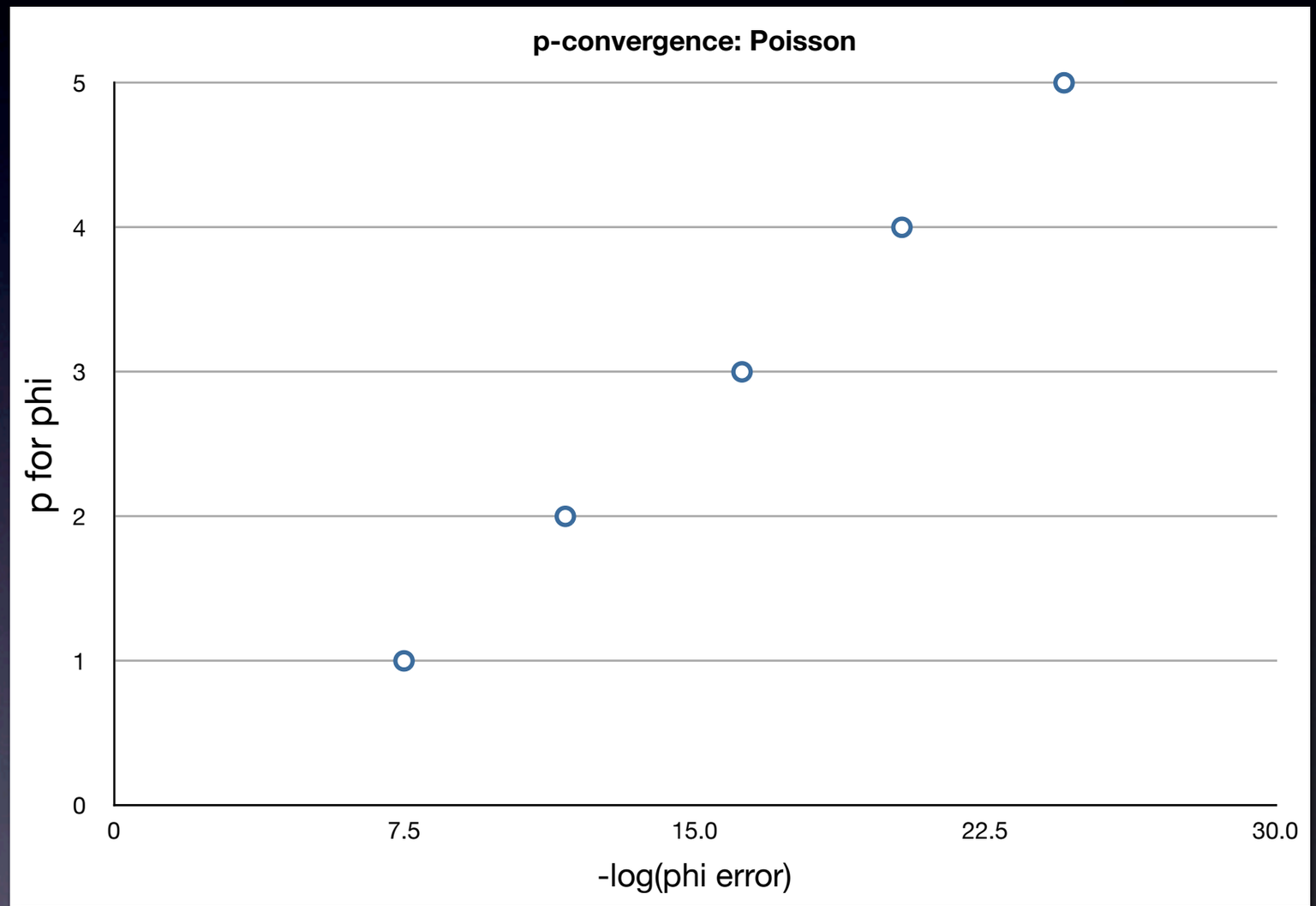
Poisson: p=4, 8x8 mesh



Poisson: p=5, 8x8 mesh

Poisson p-convergence

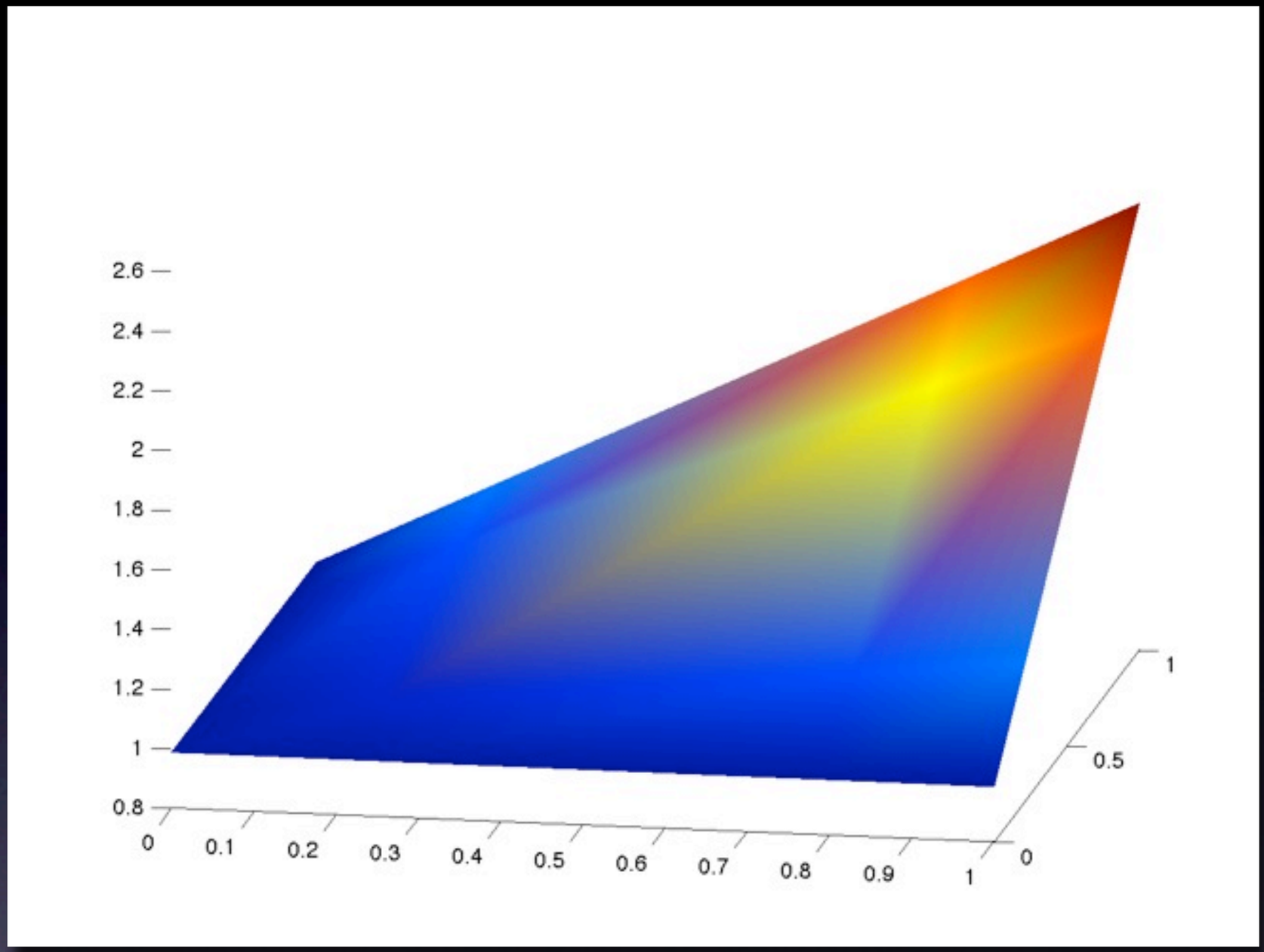
p for phi	Phi Error
1	0.000572
2	8.88E-06
3	9.27E-08
4	1.49E-09
5	2.26E-11



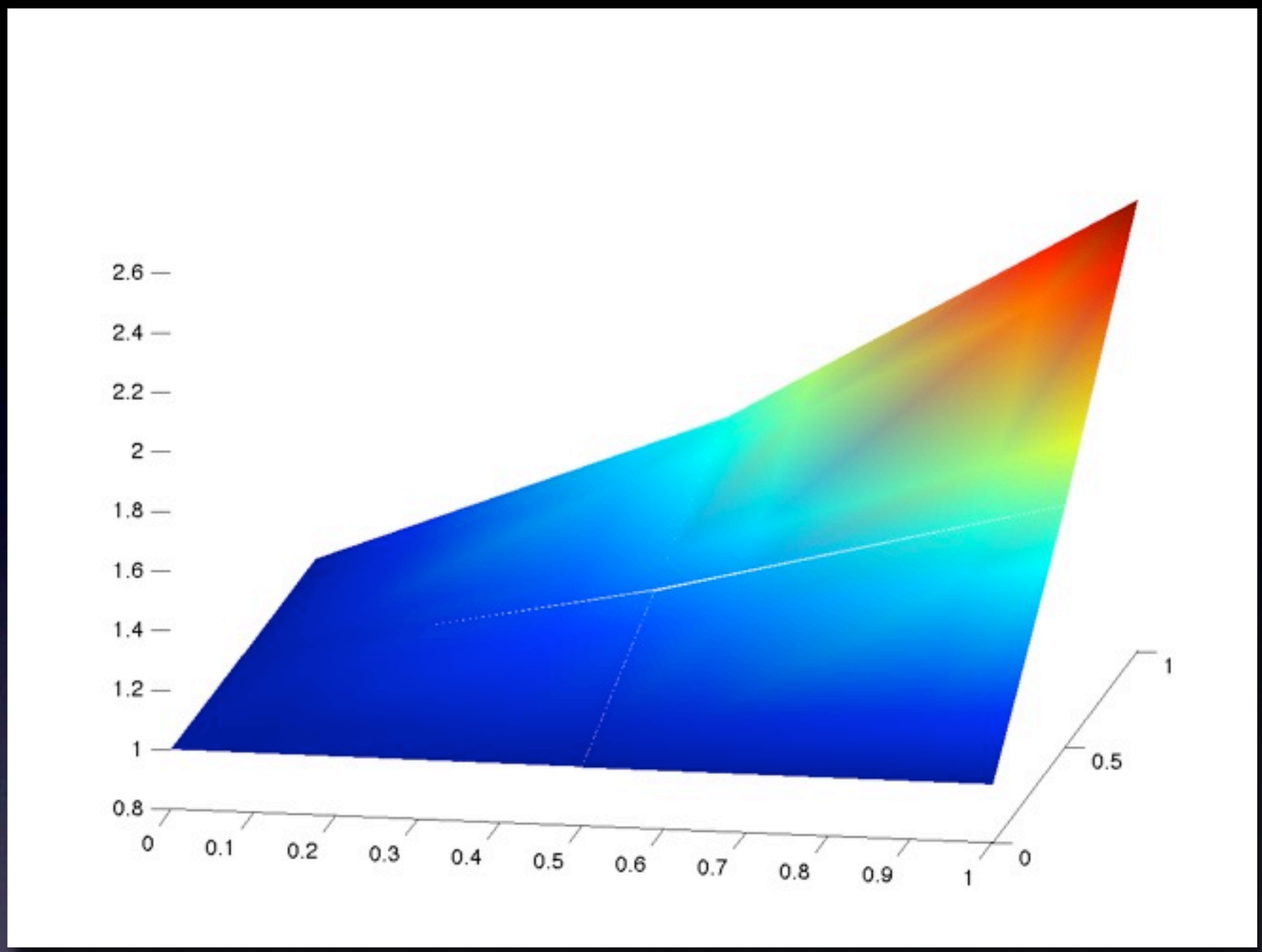
p+=2 for test functions; 8x8 mesh

Study 2:

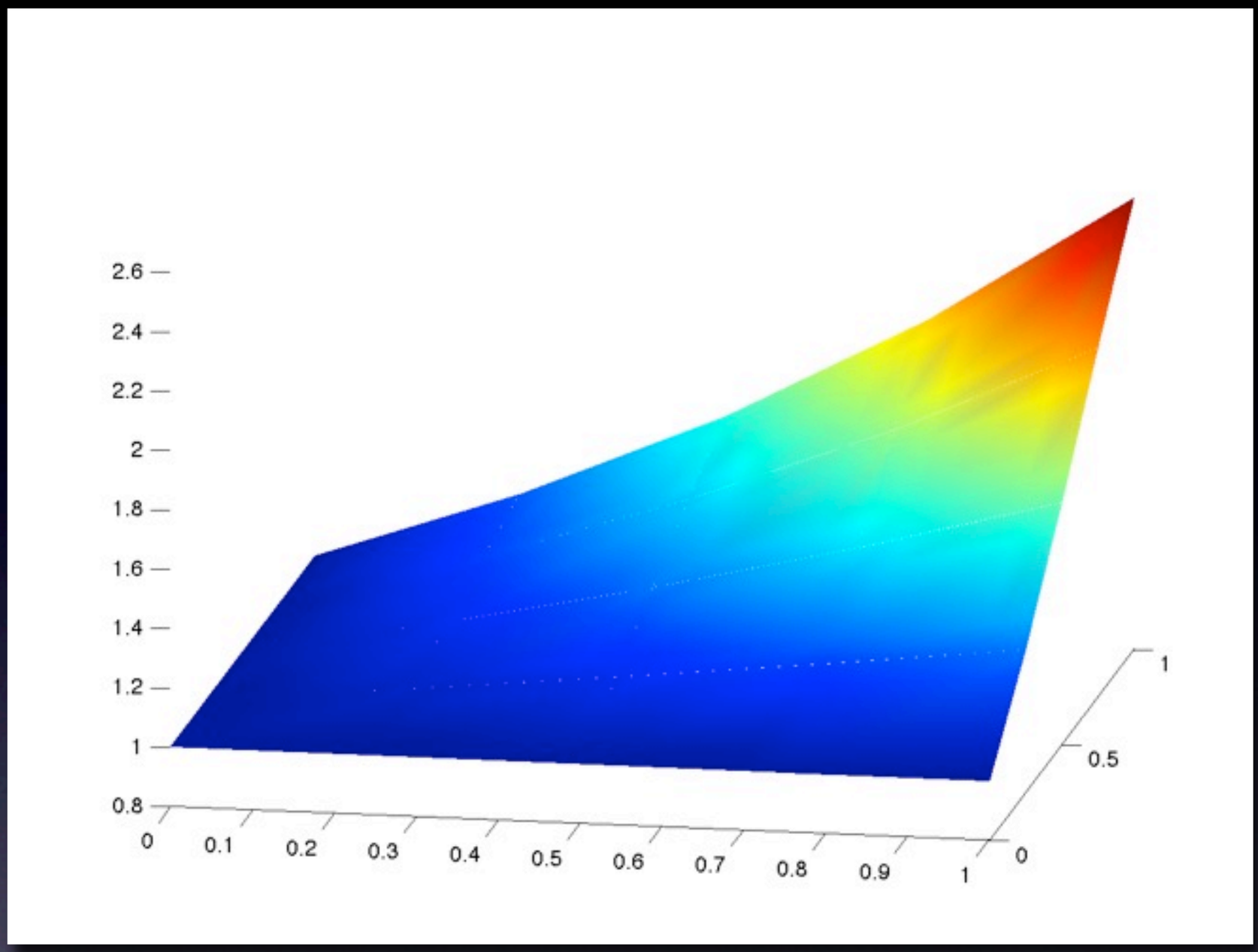
fix p=1, varying h



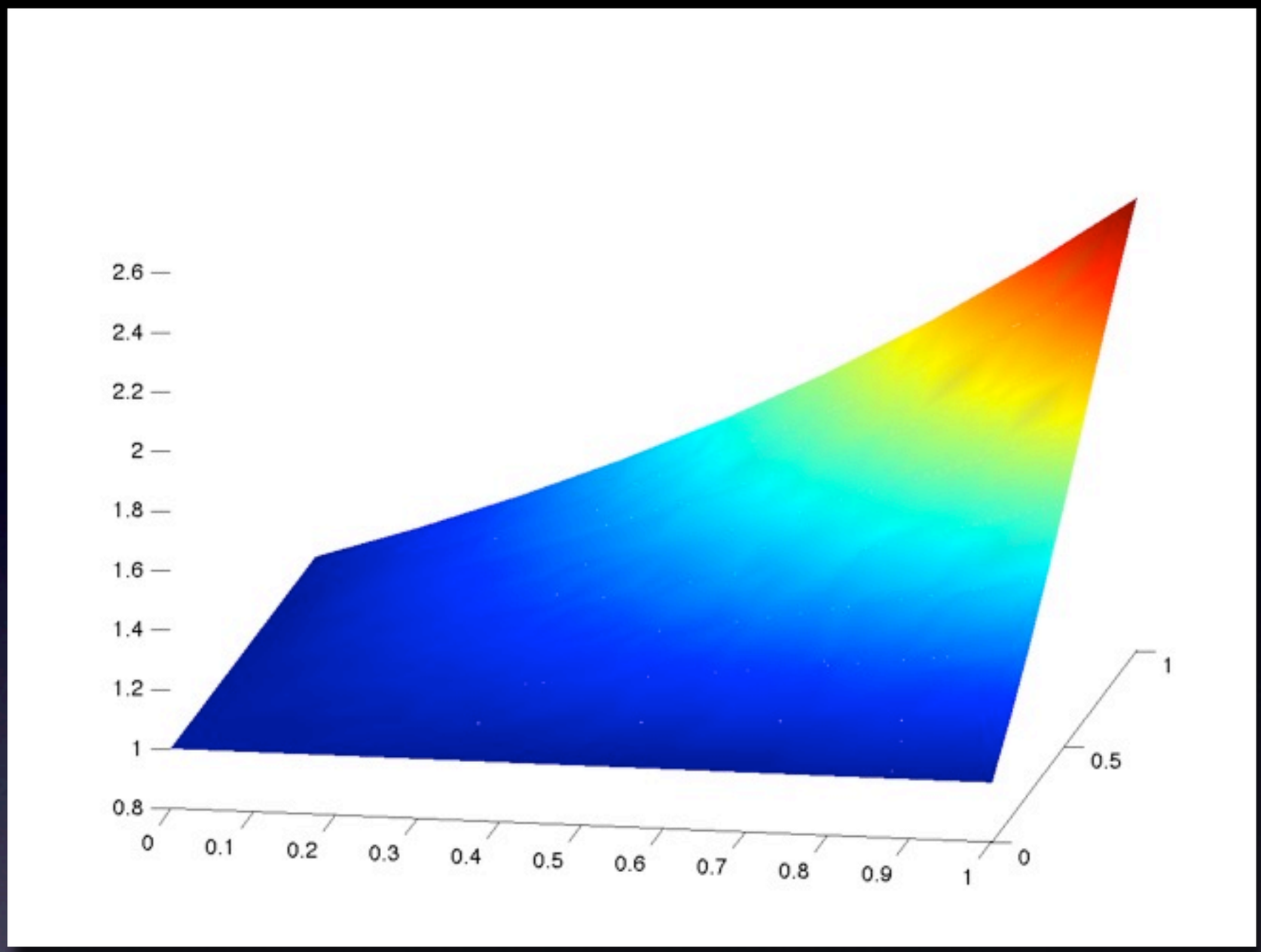
Poisson: p=1, 1x1 mesh



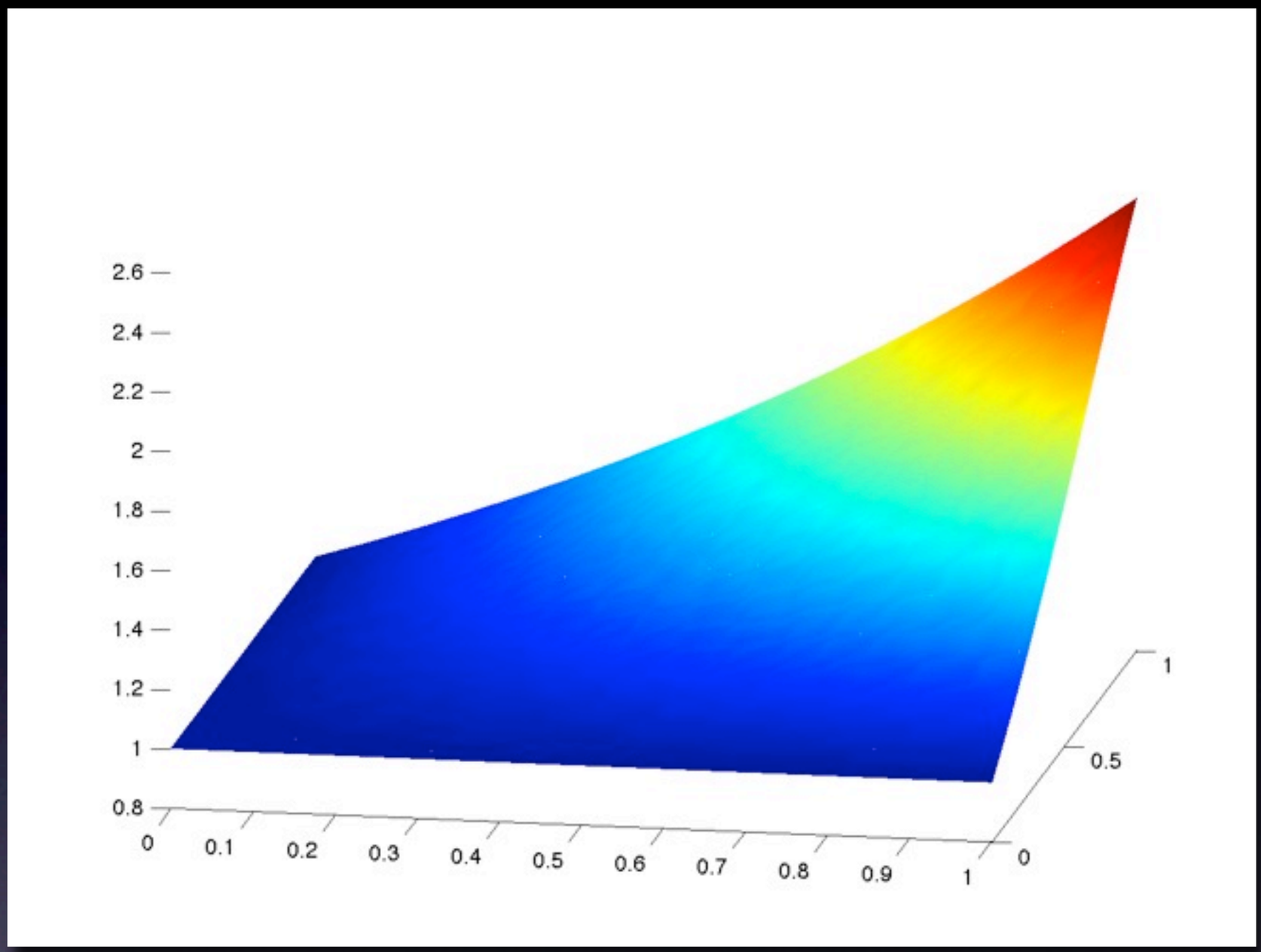
Poisson: p=1, 2x2 mesh



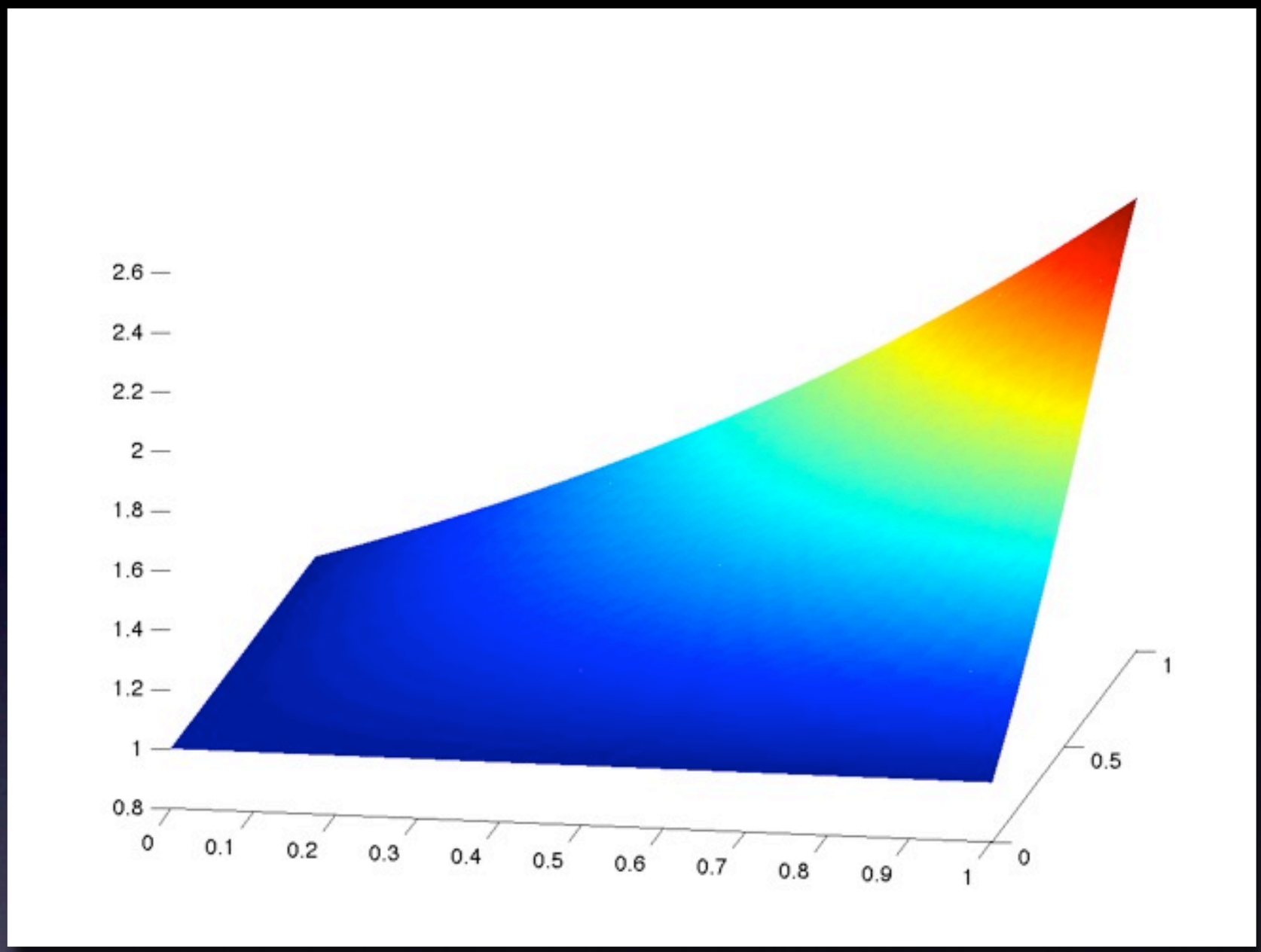
Poisson: p=1, 4x4 mesh



Poisson: p=1, 8x8 mesh



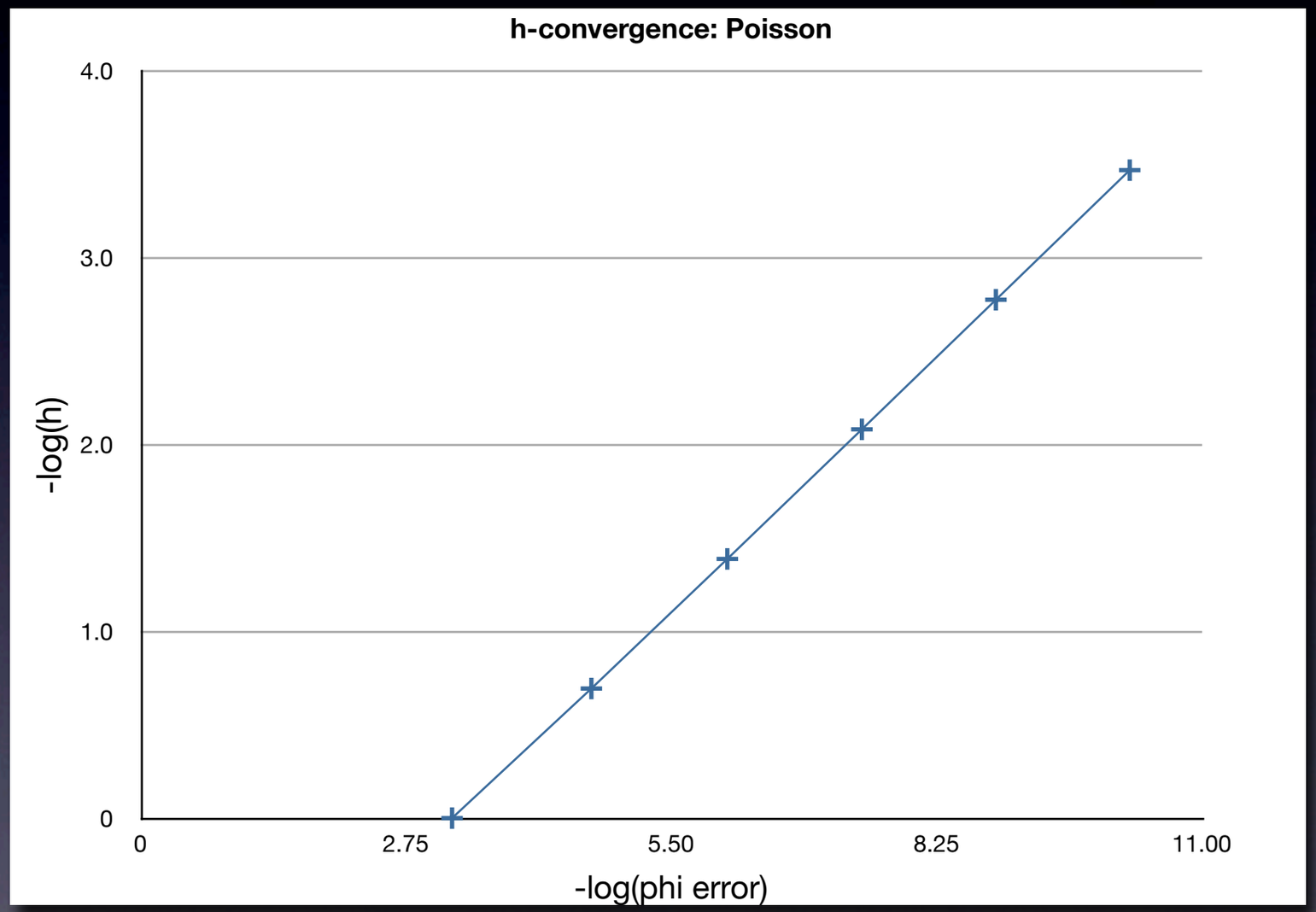
Poisson: p=1, 16x16 mesh



Poisson: p=1, 32x32 mesh

Poisson h-convergence

Mesh	Phi Error
1x1	0.040022
2x2	0.009445
4x4	0.002308
8x8	0.000572
16x16	0.000143
32x32	3.56E-05



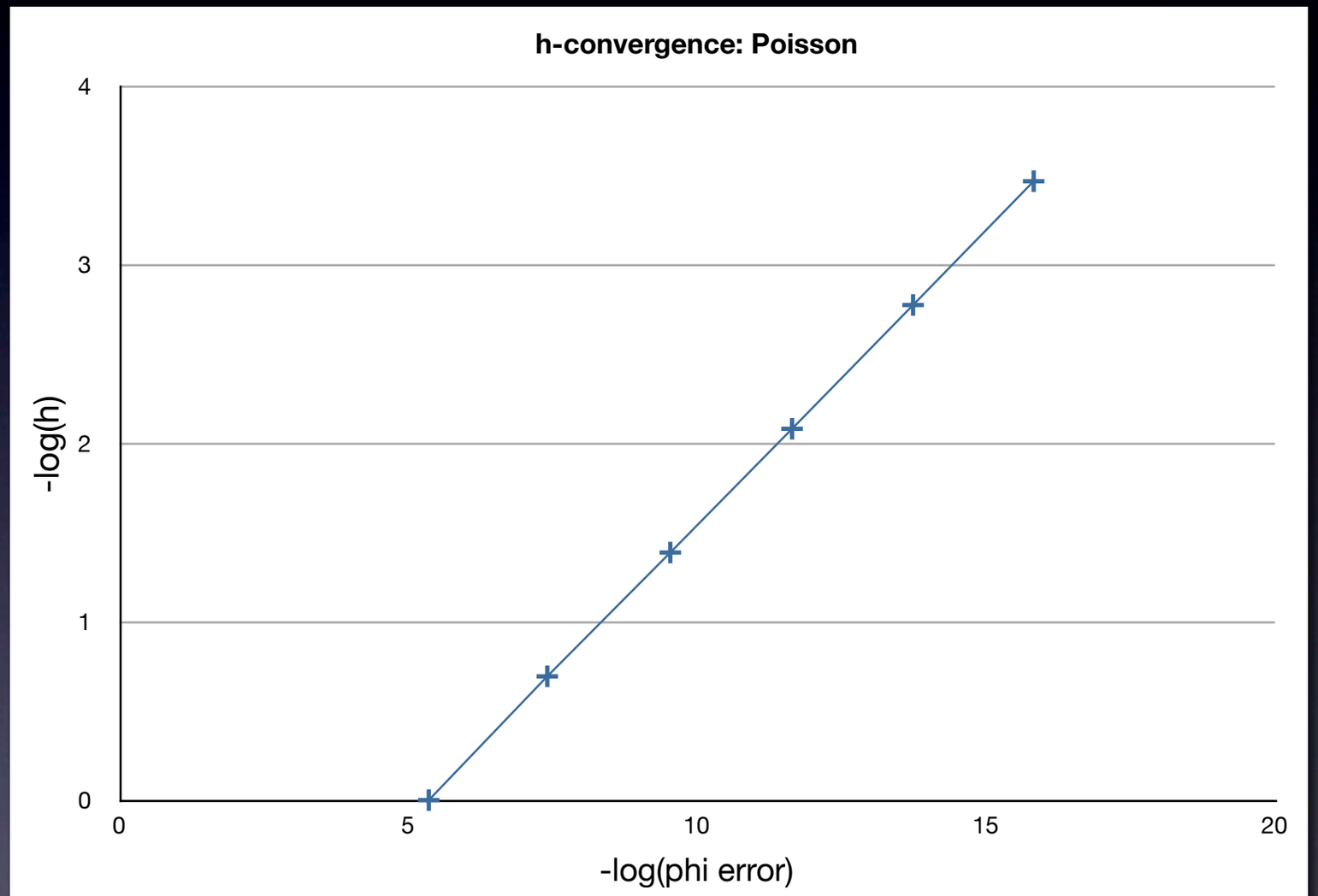
p=1 for trial variables, p=3 for test; average slope: 2.03

Study 3:

fix p=2, varying h

Poisson h-convergence

Mesh	Phi Error
1x1	0.004777
2x2	0.000615
4x4	7.31E-05
8x8	8.88E-06
16x16	1.09E-06
32x32	1.36E-07



p=2 for trial variables, p=4 for test; average slope: 3.02

Future Work

- Implement optimal test inner product
- implement last summer's 2D Stokes formulation, using Trilinos components, with and without conforming traces
- Actually implement h- and p-refinement
- Local conservation experiments
- Cavity-driven flow experiments
- Land-ice problem

Questions?