SEAMPLEX
Engineering made simple. But not simpler.

Evaluation of partial derivatives of pointwise node-based data over unstructured grids

Number
SP-WA-16-TN-9DB0

Rev.
A

Date
23-Nov-2016

Hash
d891638

Author
G. Theler jeremy@seamplex.com

Document type
Technical Note

Pages
6

Abstract

This brief note introduces a proposal to obtain the nodal values P_j of a partial derivative of a pointwise node-centered scalar field such as displacements u_j or temperatures T_j obtained with the finite-element method. We require that the nodal values of the derivative P_j interpolated with the shape functions $h_j(\mathbf{x})$ give the same internal product against each of the shape functions $h_j(\mathbf{x})$ than the derivative of the original field (say $\partial h/\partial x$) computed by using the derivative of the shape functions, say $\partial h_j/\partial x$. This requirement involves the mass matrix M, which we diagonalize to reduce the computational effort needed to obtain the nodal values P_j. We show a snippet of computational code as implemented in Seamplex' Fino.

Revision history

Rev.	Date	Author	
A	23-Nov-2016	gtheler	First issue

Contents

1 Introduction

It is often the case where a scalar field given as point-wise data over an unstructured grid needs to be differentiated over space. Illustrative cases are the solutions of partial differential equations, for example

- displacements u, v and w in a mechanical problem
- temperature T in a thermal problem

obtained by applying the finite element method over unstructured grids. In these cases, the secondary distributions

- strains ϵ and stresses σ
- heat fluxes q''

are also needed. They may be even more important than the actual base distributions.

Given a domain $U \in \mathbb{R}^3$ discretized with an unstructured grid with J nodes that define I elements, the set of nodal values say u_j and the set of shape functions $h_j(\mathbf{x})$ for $j = 1, \ldots, J$ a continuous function $u(\mathbf{x})$ is defined as

$$u(\mathbf{x}) = \sum_{j=1}^{J} h_j(\mathbf{x}) \cdot u_j \tag{1}$$

The shape functions $h_j(\mathbf{x})$ are continuous throughout the domain, but they are not necessarily differentiable at the node locations. The question that arises is how to compute another point-wise node-centered data over the same unstructured grid for these secondary distributions that involve the partial derivatives of the former. After some discussions between Seamplex and Matías Rivero from Barcelona Supercomputing Center (see figure 1), the following proposal is presented.

2 Proposal

First, we note that $u(\mathbf{x})$ as defined by equation (1) lives in the Sobolev space V_h defined by the J shape functions $h_j(\mathbf{x})$. We would like to compute the J nodal values P_j such that

$$P(\mathbf{x}) = \sum_{j=1}^{J} h_j(\mathbf{x}) \cdot P_j \tag{2}$$

is an approximation of a partial derivative of $u(\mathbf{x})$, say $\partial u/\partial x$, which we call $\epsilon(\mathbf{x})$. For this end, we require that the product of the approximation P against any function in $v_h(\mathbf{x}) \in V_h$ to be equal to the product of the derivative ϵ against the same function v_h:

$$\int_U P(\mathbf{x}) \cdot v_h(\mathbf{x})\, d^3\mathbf{x} = \int_U \epsilon(\mathbf{x}) \cdot v_h(\mathbf{x})\, d^3\mathbf{x} \qquad \forall v_h \in V_h \tag{3}$$

In particular, the shape functions $h_j(\mathbf{x})$ for $j = 1, \ldots, J$ are a basis of V_h. So in order for equation (4) to hold, and taking into account equation (2) we must have

$$\int_U \left(\sum_{j'=1}^{J} h_{j'}(\mathbf{x}) \cdot P_{j'} \right) \cdot h_j(\mathbf{x})\, d^3\mathbf{x} = \int_U \epsilon(\mathbf{x}) \cdot h_j(\mathbf{x})\, d^3\mathbf{x} \qquad \forall j = 1, \ldots, J \tag{4}$$

The nodal values P_j (which are our unknowns) are constants that do not depend on space, so we can re-arrange the integrand to arrive at

$$\sum_{j'=1}^{J} \left(\int_U h_j(\mathbf{x}) \cdot h_{j'}(\mathbf{x})\, d^3\mathbf{x} \right) \cdot P_{j'} = \int_U \epsilon(\mathbf{x}) \cdot h_j(\mathbf{x})\, d^3\mathbf{x} \qquad \forall j = 1, \ldots, J \tag{5}$$

The integral inside the parenthesis is the element $M_{jj'}$ of the mass matrix, and equation (5) is the linear problem

$$M \cdot \mathbf{P} = \boldsymbol{\epsilon}$$

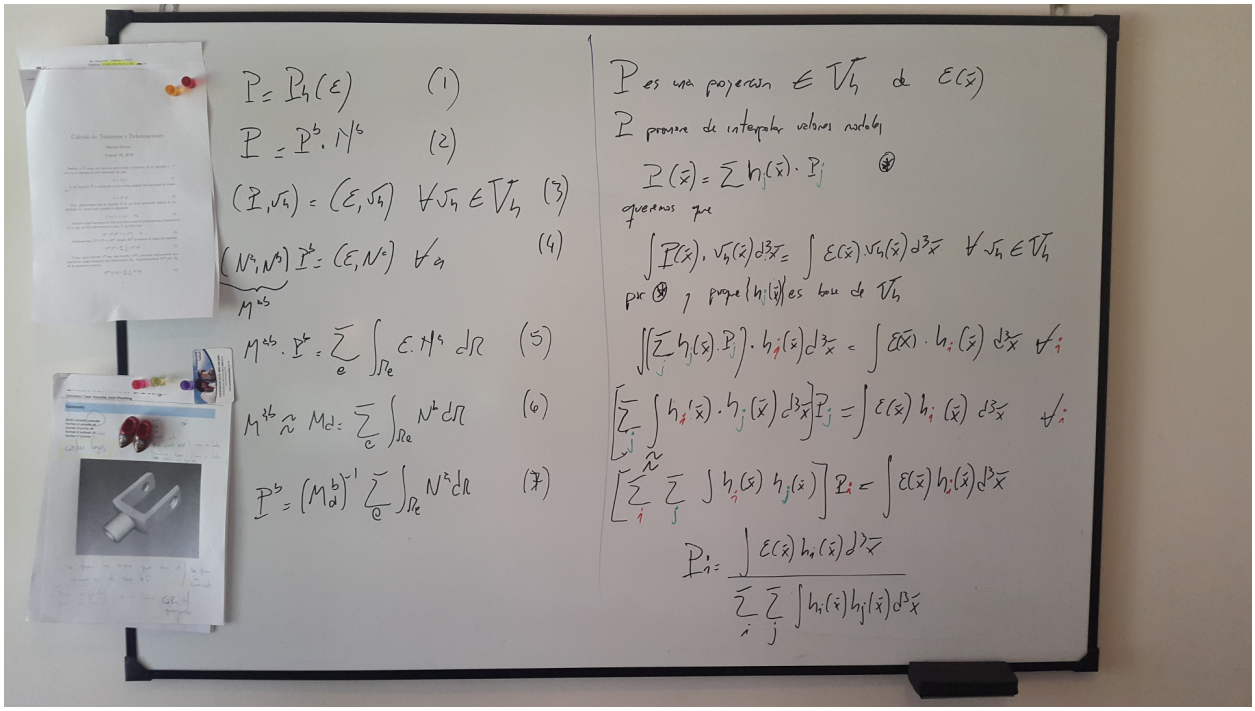

Figure 1: Our blackboard at Seamplex where the current proposal was derived. There is an extra summation sign that was detected after the picture was taken.

that ought to be solved in order to obtain the nodal values P_j. However, solving a linear problem just to obtain a derivative is a very expensive task we do not want to perform. But if we approximated the mass matrix M with a diagonal matrix $M^\star$ for example by summing all the elements of row j and storing it into the diagonal, i.e.

$$M^\star_{jj} = \sum_{j'=1}^{J} \left(\int_U h_j(\mathbf{x}) \cdot h_{j'}(\mathbf{x})\, d^3\mathbf{x} \right)$$

we would obtain the (approximated) nodal values P_j algebraically as

$$P_j \approx \frac{\displaystyle\int_U \epsilon(\mathbf{x}) \cdot h_j(\mathbf{x})\, d^3\mathbf{x}}{\displaystyle\sum_{j'=1}^{J} \int_U h_j(\mathbf{x}) \cdot h_{j'}(\mathbf{x})\, d^3\mathbf{x}} \tag{6}$$

If in particular $\epsilon(\mathbf{x})$ refers to the partial derivative $\partial u/\partial x$, then

$$\epsilon(\mathbf{x}) = \frac{\partial}{\partial x}\left[\sum_{j=1}^{J} h_j(\mathbf{x}) \cdot u_j\right] = \sum_{j=1}^{J} \frac{\partial h_j}{\partial x} \cdot u_j \tag{7}$$

Inserting equation (7) into (6), taking care of the dummy summation index and re-arranging factors, we obtain

$$P_j \approx \frac{\displaystyle\sum_{j'=1}^{J} \left(\int_U \frac{\partial h_{j'}}{\partial x} \cdot h_j(\mathbf{x})\, d^3\mathbf{x} \right) \cdot u_{j'}}{\displaystyle\sum_{j'=1}^{J} \int_U h_j(\mathbf{x}) \cdot h_{j'}(\mathbf{x})\, d^3\mathbf{x}}$$

2.1 Computational implementation

```
for (j = 0; j < fino.mesh->n_nodes; j++) {

  for (g = 0; g < fino.degrees; g++) {
    for (d = 0; d <fino.dimensions; d++) {
      fino.gradient[g][d]->data_value[j] = 0;
    }
  }

  M_jj = 0;  // lumped mass diagonal entry
  LL_FOREACH (fino.mesh->node[j].associated_elements, associated_element) {
    if (associated_element->element->type->dim == fino.dimensions) {

      // find the local index that corresponds to the global j index
      local_j = -1;
      for (local_jprime = 0; local_jprime < associated_element->element->type->nodes; local_jprime++) {
        if (associated_element->element->node[local_jprime]->id == j+1) {
          local_j = local_jprime;
          break;
        }
      }

      for (v = 0; v < associated_element->element->type->gauss[GAUSS_POINTS_CANONICAL].V; v++) {
        w_gauss = mesh_integration_weight(fino.mesh, associated_element->element, v);
        mesh_compute_x(associated_element->element, fino.mesh->fem.r, fino.mesh->fem.x);
        mesh_inverse(fino.mesh->bulk_dimensions, fino.mesh->fem.dxdr, fino.mesh->fem.drdx);
        mesh_compute_dhdx(associated_element->element, fino.mesh->fem.r, fino.mesh->fem.drdx, fino.mesh->fem.dhdx);

        for (local_jprime = 0; local_jprime < associated_element->element->type->nodes; local_jprime++) {
          M_jj += w_gauss * gsl_vector_get(fino.mesh->fem.h, local_jprime) * gsl_vector_get(fino.mesh->fem.h, local_j);

          for (g = 0; g < fino.degrees; g++) {
            petsc_call(VecGetValues(fino.phi, 1, &associated_element->element->node[local_jprime]->index[g], &xi));
            for (d = 0; d <fino.dimensions; d++) {
              fino.gradient[g][d]->data_value[j] += w_gauss * gsl_matrix_get(fino.mesh->fem.dhdx, local_jprime, d) * xi *  ↩
                    gsl_vector_get(fino.mesh->fem.h, local_j);
            }
          }
        }
      }
    }
  }

  for (g = 0; g < fino.degrees; g++) {
    for (d = 0; d <fino.dimensions; d++) {
      fino.gradient[g][d]->data_value[j] /= M_jj;
    }
  }
}
```