Correction to Laplacian

$$\mathbb{A}_e \sum_q N^e_{b,\xi_k}(\xi^q)\, \xi_{k,j}\, \xi_{l,j} \left[\sum_a N^e_{a,\xi_l}(\xi^q)\, u^e_a\right] \mathcal{D} w^q = 0$$

libCEED Operator

Symbol	math	math^T	code object
A	$\mathcal{E}^T B^T D B\, \mathcal{E}$	—	CeedOperator
$\mathcal{E}$	$u_A \rightarrow u^e_a$	$\mathbb{A}_e\, G^e_b = G_B$	CeedElemRestriction
B	$\sum_a N^e_{a,\xi_l}(\xi^q)\, u^e_a = u_{,\xi_l}(\xi^q)$	$\sum_q N^e_{b,\xi_k}\, G_k(\xi^q) = G^e_b$	Ceed Basis
D	$\xi_{k,j}\, \xi_{l,j}\, \mathcal{D} w^q\, u_{,\xi_l}(\xi^q) = G_k(\xi^q)$		CeedQFunction

Not exactly same as PHASTA definition

```c
CEED_QFUNCTION(laplacian)(void *ctx, const CeedInt Q, const CeedScalar *const *in,
                          CeedScalar *const *out) {
  // in[0] is gradient u, shape [2, nc=1, Q]
  // in[1] is quadrature data, size (3*Q)
  const CeedScalar (*du)[CEED_Q_VLA] = (const CeedScalar(*)[CEED_Q_VLA])in[0];
  const CeedScalar (*qd)[CEED_Q_VLA] = (const CeedScalar(*)[CEED_Q_VLA])in[1];

  // out[0] is output to multiply against gradient v, shape [2, nc=1, Q]
  CeedScalar (*dv)[CEED_Q_VLA] = (CeedScalar(*)[CEED_Q_VLA])out[0];

  // Quadrature point loop
  for (CeedInt q=0; q<Q; q++) {
    const CeedScalar du0 = du[0][q];
    const CeedScalar du1 = du[1][q];
    dv[0][q] = qd[0][q]*du0 + qd[2][q]*du1;
    dv[1][q] = qd[2][q]*du0 + qd[1][q]*du1;
  }

  return 0;
}
```

$du[l,q] = u_{,\xi_l}(\xi^q)$

$qd[m,q] = \underbrace{\xi_{k,j}\, \xi_{l,j}}_{\text{symmetric}}\, \mathcal{D} w^q$

$$\begin{bmatrix} 0 & 2 \\ 2 & 1 \end{bmatrix}$$

$dv[k,q] = G_k(\xi^q)$

$$\begin{bmatrix} dv_0 \\ dv_1 \end{bmatrix} = \begin{bmatrix} qd_0 & qd_2 \\ qd_2 & qd_1 \end{bmatrix} \begin{bmatrix} du_0 \\ du_1 \end{bmatrix}$$

```c
CEED_QFUNCTION(setup)(void *ctx, const CeedInt Q, const CeedScalar *const *in,
                      CeedScalar *const *out) {
  // At every quadrature point, compute qw/det(J).adj(J).adj(J)^T and store
  // the symmetric part of the result.

  // in[0] is Jacobians with shape [2, nc=2, Q]
  // in[1] is quadrature weights, size (Q)
  const CeedScalar (*J)[CEED_Q_VLA] = (const CeedScalar(*)[CEED_Q_VLA])in[0];
  const CeedScalar *qw = in[1];

  // out[0] is qdata, size [3, Q]
  CeedScalar (*qd)[CEED_Q_VLA] = (CeedScalar(*)[CEED_Q_VLA])out[0];

  // Quadrature point loop
  for (CeedInt q=0; q<Q; q++) {
    // J: 0 2   qd: 0 2   adj(J):  J22 -J12
    //    1 3       2 1           -J21  J11
    const CeedScalar J11 = J[0][q];
    const CeedScalar J21 = J[1][q];
    const CeedScalar J12 = J[2][q];
    const CeedScalar J22 = J[3][q];
    const CeedScalar w = qw[q] / (J11*J22 - J21*J12);
    qd[0][q] =   w * (J12*J12 + J22*J22);
    qd[2][q] =   w * (J11*J11 + J21*J21);
    qd[1][q] = - w * (J11*J12 + J21*J22);
  }

  return 0;
}
```

Recall $\underset{\sim}{x}(\underset{\sim}{\xi}) = \sum_a N_a^e(\underset{\sim}{\xi})\, \underset{\sim}{x}_a^e$

$J[i,j,q] = \sum_a N^e_{a,\xi_j}(\xi^q)\, x^e_{a_i} = \partial x / \partial \xi$

$qw[q] = w^q$

Aside: C vectors/arrays

C multi-index arrays can/are be represented as 1D arrays

Given X 2×3 : $\begin{bmatrix} X_{11} & X_{12} & X_{13} \\ X_{21} & X_{22} & X_{23} \end{bmatrix}$

In memory, C stores this as:

$$[X_{11}\ X_{12}\ X_{13}\ X_{21}\ X_{22}\ X_{23}]$$ row-major order

Express $X'[6]$ indexes equivalently:

$$X[i][j] = X'[i \cdot n_j + j]$$ (assumes index by zero)

where n_j is the stride of the j^{th} component