1 Lecture 1

Scientific libraries are necessary in order to communicate algorithmic advances effectively. Too much is left unspecified in papers, although papers are also necessary for proof convergence, accuracy, and performance models. Anything connected with performance measurements is unreviewable without running the code.

Libraries hide both hardware details (MPI) and implementation complexity (Krylov solvers). They also allow the accumulation of best practices without user input (default choice of classical Gram-Schmidt with selective reorthogonalization for parallel Krylov solvers). Users can also benefit from library improvements without any changes to their own code, such as better time step adaptivity in PETSc. Finally, extensibility is the hallmark of scientific code, since scientists are by definition attacking problems which have not been solved previously.

Historical Numerical Libraries

73 EISPACK, Brian Smith et.al.
79 BLAS, Lawson, Hanson, Kincaid and Krogh
90 LAPACK, many contributors
91 PETSc, Gropp and Smith
95 MPICH, Gropp and Lusk

This class will be centered around three programming projects, two of which are individual and one group-oriented. The individual projects will be approximately 21/2 weeks long, and the group project 31/2 weeks. They will cover finite element construction, optimization, and use in science and engineering.

Emphasize that C programming and PETSc use will be extremely important. If you are weak in these areas, the class will be oppressive.
2 Lecture 3

Any discussion of the finite element method should start with a search for solutions to partial differential equations. How do we solve them (or associated eigenvalue or optimization problems)? The first thing we would love to do is write down the solution function. This is almost never possible. However, we can do lots of things, such as infinite series and spectral methods, Green function integrals, conformal transformations, etc. Why don’t we just use these methods of the 18th and 19th centuries? I think the main objection is that general boundary geometry becomes very difficult. However, the study of variational problems leads to a new idea (weak solution) and a new method (Ritz method) which happens also to solve the problem of geometrically complex boundaries.

Ritz (1908) developed a method which is based on using an ansatz for the solution

\[ u(x) = \sum_{i} u_i \phi_i(x) \]  

(1)

so that the continuum problem reduces to a finite dimensional problem. The other ingredient, which arises naturally in variational problems, is the idea of a weak solution, namely that we look for residuals which are not pointwise zero, but rather orthogonal to a set of functions. Often we choose this set based on physical grounds. For example, the residual could be orthogonal to all low frequency functions, so that any deviation was higher frequency, meaning higher energy, and thus physically unlikely.

Finite elements arise from a particular choice of functions in which to expand. They have finite support, thus the finite, and that support coincides with cells into which the domain is divided, the elements. The original functions used by Courant and others were linear, or hat functions since they look like dunces hats in one dimension. Today continuous functions which are polynomial when restricted to a cell are popular, and often refered to as continuous Galerkin elements.

Introduce Ciarlet definition of finite element, namely that I have a triplet \((P, P', T)\) where \(P\) is a linear space from which we draw our approximate solution, \(P'\) is the dual space to \(P\), and \(T\) is the generic cell shape into which the domain \(\Omega\) is divided. What this downplays is that when computing this local definition is paired with conditions between cells. This opens up the possibility of non-conforming approximations, meaning discrete solution spaces which are not subspaces of the continuum approximation space.

Show the strong and weak forms Laplace and Stokes equations on the board.

Breakdown weak form computation into pieces. Introduce Jed form of residual and Jacobian.
3 Lecture 5

In FEM, we are interested in calculating things like the weak form of the residual

\[ \int_{\Omega} \psi F(u, \nabla u, x, t) \quad \psi \in P \] (2)

which can be broken down into cell integrals, and transformed to a reference cell,

\[ \sum_{T \in \Omega} \int_{T} \psi F(u, \nabla u, x, t) \quad \psi \in P \] (3)

\[ = \sum_{T \in \Omega} \int_{T_{ref}} \psi F(u, \nabla u, \xi, t) |J| \quad \psi \in P \] (4)

which can be approximated by quadrature

\[ \sum_{T \in \Omega} \sum_{q} \psi(x_q) F(u(x_q), \nabla u(x_q), x_q, t) \quad \psi \in P. \] (5)

where in all this we have

\[ u(x) = \sum_{i} u_i \phi_i(x) \quad \phi_i \in P. \] (6)

Thus it seems advantageous to have the basis function tabulated at the quadrature points, not only to evaluate \( \psi \), but also \( u \) and its derivatives.

Another operation we frequently want is the projection of a function onto our approximation space

\[ f(x) = \sum_{i} f_i \phi_i(x). \] (7)

This is trivial with an orthogonal basis,

\[ \int \phi_j f(x) = \int \phi_j \left( \sum_{i} f_i \phi_i(x) \right), \] (8)

\[ = \sum_{i} f_i \left( \int \phi_j \phi_i(x) \right), \] (9)

\[ = \sum_{i} f_i \delta_{ij}, \] (10)

\[ = f_j. \] (11)

However, orthogonal bases tend to produce ill-conditioned systems and also often have functions with global support. We are thus led to a biorthogonal system,

\[ n_j (\phi_i(x)) = \delta_{ij} \quad \phi_i \in P, n_j \in P'. \] (12)
Now we can get the \( i \)th finite element coefficient for a function just by acting with the \( i \)th dual basis vector,

\[
n_j(f(x)) = n_j \left( \sum_i f_i \phi_i(x) \right),
\]

(13)

\[
= \sum_i f_i n_j(\phi_i(x)),
\]

(14)

\[
= \sum_i f_i \delta_{ij},
\]

(15)

\[
= f_j,
\]

(16)

where we have used the linearity of the functional \( n_j \).

FIAT is an automated system for producing a biorthogonal, or \textit{nodal}, basis. The relation is obtained simply from the biorthogonality requirement. Suppose we already have another basis \( m_i \), usually called the \textit{modal} basis, and that we expand each \( \phi_i \) in terms of the \( m \) basis,

\[
n_j(\phi_i(x)) = \delta_{ij},
\]

(17)

\[
n_j \left( \sum_k p_k^i m_k \right) = \delta_{ij},
\]

(18)

\[
\sum_k p_k^i n_j(m_k) = \delta_{ij}.
\]

(19)

In matrix notation, this would be

\[
Vp = I
\]

(20)

where the matrix \( V \), called the \textit{Vandermonde} matrix, is defined as

\[
V_{jk} = n_j(m_k).
\]

(21)

Now we just invert \( V \) to get the coefficients \( p \) of the nodal basis we want in terms of the original \( m \) basis that we can easily write down. For example, it’s easy to write down a monomial basis, \( x^k \), for any space of polynomials. We could then use this algorithm to automatically produce the nodal basis. We usually do not use monomials because evaluation becomes unstable at high degree, but other stable bases, like the Dubiner basis used in FIAT, are available. Also notice that FIAT automates the production of local basis vectors for the primal and dual spaces, but does not have an abstraction for the “piecing together” of cells.

FIAT also fails to provide an encapsulation of the dual basis vectors; however, this is straightforward. When we represent the dual basis in our code, we can make use of the Riesz-Markov-Kakutani Representation Theorem (Riesz 1909; Markov 1938; Kakutani 1941; Wikipedia 2015) which says that any positive linear functional \( \psi \) on \( C_c(X) \), the space of continuous compactly supported
complex-valued functions on a locally compact Hausdorff space $X$, there is a unique regular Borel measure $\mu$ on $X$ such that

$$\psi(f) = \int_X f(x) d\mu(x).$$  \hfill (22)

Since we are operating in the purely discrete world of the computer, we will represent these measures by a quadrature rule

$$\int_X f(x) d\mu(x) \approx \sum_q w_q f(x_q).$$  \hfill (23)

Thus our dual space bases may all be stored internally as sets of quadrature rules supported on a cell and its boundary.

Sometimes, the specification of $P$ is not so straightforward. For example, we might say that $P$ is the space of all polynomials of degree $k + 2$ with divergences that are polynomials of degree $k$. It is not easy to write down a basis for such a space. Thus we start with the bigger space $\bar{P} = P_{k+2}$ for which we have a nice basis. Next we express the constraints as functionals. In our example above, we would use a functional like

$$l_i(m_j) = \int \mu_{i}^{k-1}(\nabla \cdot m_j)$$  \hfill (24)

where $\mu_i^{k-1}$ is a Legendre polynomial of degree $k - 1$ since this is orthogonal to all polynomials of lesser degree. Our space $P$ would now be the intersection of the null spaces of all the functionals $l_i$. This is easy to express using linear algebra. First we act with the functionals on our basis,

$$L_{ij} = l_i(m_j)$$  \hfill (25)

and then use the full SVD,

$$L = U\Sigma V.$$  \hfill (26)

The last columns of $V$ give a basis for the null space of $L$. We then feed this basis into our algorithm from before which produces the nodal basis $\phi$.

**Homework** Everyone must install the development branch of PETSc from the Git repository. It must be configured with MPI, SuperLU (so that we have parallel direct solves), Triangle and CTetGen (for unstructured meshing), p4est (for structured meshing), Metis and Parmetis (for mesh partitioning), and HDF5 (for output). If possible, I recommend using `ccache` during installation, e.g. `--with-cc=/Users/knepley/MacSoftware/bin/ccache,gcc,-Qunused-arguments`, so that re-compilation during development is not so slow.
**Quiz**  The Neumann problem for the Laplacian can be formulated,

\[ \Delta u = 0 \quad \int_{\Omega} u = 0, \quad (27) \]

where we in essence remove the constant function from the approximation space for \( u \). This is also exactly what we see for the \( P_1 \) element matrix

\[
\begin{pmatrix}
0.5 & 0 & -0.5 \\
0 & 0.5 & -0.5 \\
-0.5 & -0.5 & 1.0
\end{pmatrix}
\quad (28)
\]

which has the constant vector in its null space since all row sums vanish. However, if we treat the strong form of the equation (27) and solve by expanding in a polynomial basis, then the coefficients for the constant and linear basis functions are both free. What happened to the extra degree of freedom?


---

**4 Lecture 6**

```
./fem -log_view

./fem -dn_view
./fem -dn_view -dim 3
./fem -dn_view -simplex 0
./fem -dn_view -simplex 0 -dim 3

./fem -dn_petscsection_view
./fem -dn_petscsection_view -petscspace_order 1
./fem -dn_petscsection_view -petscspace_order 4 -dim 3
```

---

**5 Lecture 7**

The first programming assignment will be to construct your own finite element. The element will consist of a primal approximation space, a dual space to the primal space, and a prescription for piecing together the cell approximations. For simple continuous and discontinuous spaces, this prescription takes the form of an association between dual basis vectors and topological elements of the reference cell. An excellent resource is the FEniCS book.

We will employ the FIAT strategy for constructing a *nodal* basis, so the the primal space need only be able to evaluate basis functions and their derivatives at an arbitrary point in the reference cell. This is embodied in the `PetscSpaceEvaluate()` function. PETSc provides two default spaces: polynomials of a given degree, and delta functions supported on a set of points.

We have already noted that dual basis vectors can be encoded as quadrature rules, for which we use the `PetscQuadrature` object. The basic operation is to calculate the action of a dual basis vector on a function supported on the reference cell, embodied in `PetscDualSpaceApply()`. Since it must know about the reference cell, we set this using `PetscDualSpaceSetDM()`, and can create it using `PetscDualSpaceCreateReferenceCell()`. PETSc provides two default
dual spaces: point evaluation functionals at the Lagrange interpolation points, and an arbitrary set of user-defined quadratures.

The PetscFE class combines a PetscSpace (PetscFESetBasisSpace()), a PetscDualSpace (PetscFESetDualSpace()), and a cell and face PetscQuadrature (PetscFESetQuadrature(), PetscFESetFaceQuadrature()). The element uses the FIAT algorithm to produce a tabulation at the quadrature points, which we need for residual and Jacobian calculation, that can be accessed using PetscFEGetDefaultTabulation(). More generally, it can tabulate on any given set of point in the reference cell using PetscFEGetTabulation(). The main job of a PetscFE is to carry out integrals of weak forms, defined by pointwise functions, over cells, which it does using the methods

- PetscFEIntegrate(),
- PetscFEIntegrateResidual(),
- PetscFEIntegrateBdResidual(),
- PetscFEIntegrateJacobian(), and
- PetscFEIntegrateBdJacobian().

PETSc provides four default element types which handle different evaluation schemes: basic, composite, non-affine, and OpenCL. The basis scheme just loops over cells, which some tiling for cache efficiency. The composite class allows integration over composite cell groups, such as regularly refined cells. The non-affine evaluator uses higher order geometric information in the transformations. Finally, the OpenCL evaluator uses an OpenCL kernel to carry out the integration over blocks of cells. The PetscFECreateDefault() function provides a factory interface for the Lagrange elements using any of the evaluation schemes.

### 6 Lecture 8

We have already seen how the weak residual is defined by an integral over the domain Ω, which may be decomposed into integrals over the cells $T$. We now introduce a model of the cell integrals (Knepley et al. 2013),

$$
\int_T \psi F(u) = \int_T \psi f_0(u, \nabla u, x, t) + \nabla \psi \cdot \vec{f}_1(u, \nabla u, x, t)
$$

(29)

where we explicitly acknowledge that an integration by parts can be used for some terms in the residual. For inhomogeneous Neumann conditions, this will also produce a boundary integral which is not shown. The functions $f_0$ and $\vec{f}_1$ are defined pointwise on the reference cell, and are independent of the discretization. However, we must be able to reconstruct values of the solutions fields $u$ and its gradients $\nabla u$ at the evaluation points. Thus our interface with the user has been reduced to a pointwise definition of the integrand for the weak residual. Below we give a few examples.
**Poisson**  The Poisson equation,

\[-\Delta u = f, \tag{30}\]

produces the weak form

\[\langle \psi, -\Delta u - f \rangle = 0, \tag{31}\]
\[\langle \nabla \psi, \nabla u \rangle - \langle \psi, f \rangle = 0 \tag{32}\]

which gives us the definition

\[f_0 = -f, \tag{33}\]
\[\vec{f}_1 = \nabla u - \langle \psi, f \rangle. \tag{34}\]

We start by running

```plaintext
./ex12 -run_type test -bc_type dirichlet -interpolate 1 -petscspace_order 1
```

which shows the initial guess, \(L_2\) error, and initial residual. If we turn on FEM printing,

```plaintext
./ex12 -run_type test -bc_type dirichlet -interpolate 1 -petscspace_order 1 -dm_plex_print_fem 1
```

then it also displays the Jacobian. Increasing the print level

```plaintext
./ex12 -run_type test -bc_type dirichlet -interpolate 1 -petscspace_order 1 -dm_plex_print_fem 3
```

displays each cell integral for the residual and Jacobian, as well as the global matrix indices for each entry. We can refine the mesh in two different ways. If we use cell volume constraints,

```plaintext
./ex12 -run_type test -refinement_limit 0.0625 -bc_type dirichlet -interpolate 1 -petscspace_order 1 -dm_plex_print_fem 3
```

we can just double the number of cells, and get 5 degrees of freedom (dofs). Whereas regularly refining the mesh turns each triangle into four subtriangles,

```plaintext
./ex12 -run_type test -dm_refine 1 -bc_type dirichlet -interpolate 1 -petscspace_order 1 -dm_plex_print_fem 3
```

so that we have 9 dofs.

**Stokes**  The Stokes equation,

\[-\Delta v - \nabla p = f, \tag{35}\]
\[\nabla \cdot v = 0, \tag{36}\]

produces the weak form

\[\langle \nabla \psi_v, \nabla u \rangle + \langle \nabla \cdot \psi_v, p \rangle - \langle \psi_v, f \rangle = 0 \tag{37}\]
\[\langle \psi_p, \nabla \cdot v \rangle = 0 \tag{38}\]

which gives us the definition

\[f_0 = \begin{pmatrix} -f \\ \nabla \cdot v \end{pmatrix}, \tag{39}\]
\[\vec{f}_1 = \begin{pmatrix} \nabla u + pI \\ 0 \end{pmatrix}, \tag{40}\]

where \(I\) is the identity tensor. Note here that for multiple fields \(f_0\) is a vector and \(\vec{f}_1\) is a rank 2 tensor.
The single-fluid resistive MHD equations couple the equations of hy-
drodynamics and resistive Maxwells equations and may be written below in
conservation form

$$\frac{\partial u}{\partial t} + \nabla \cdot F(u) = \nabla \cdot F_d(u),$$  \hspace{1cm} (41)

where the fields are $u = (\rho, \rho \vec{v}, \vec{B}, e)$, $F$ is the flux,

$$F = \begin{pmatrix} \rho \vec{v} + \left( p + \frac{1}{2} \vec{B} \cdot \vec{B} \right) I \\ \vec{v} \vec{B} - \vec{B} \vec{v} \\ e + p + \frac{1}{2} \vec{B} \cdot \vec{B} \end{pmatrix} \vec{v} - \vec{B} (\vec{B} \cdot \vec{v})$$  \hspace{1cm} (42)

and $F_d$ the diffusive flux,

$$F_d = \begin{pmatrix} 0 \\ \frac{\eta}{Re \tau} \left( \nabla \vec{B} - \nabla \vec{B}^T \right) \\ \frac{1}{Re \tau} \cdot \vec{v} + \frac{\gamma}{\gamma - 1} \frac{\kappa}{Re \tau} \nabla T + \frac{\eta}{2} \left( \frac{1}{2} \nabla (\vec{B} \cdot \vec{B}) - \vec{B} \nabla \vec{B}^T \right) \end{pmatrix}$$  \hspace{1cm} (43)

In the above equations, $\rho$ is the density, $\vec{v}$ is the velocity, $\vec{B}$ is the magnetic field, $p$ and $T$ are the pressure and temperature, respectively, and $e$ is the total energy per unit volume of the plasma. The plasma properties are the resistivity $\eta$, the thermal conductivity $\kappa$, and the viscosity $\mu$, which have been normalized, respectively, by a reference resistivity $\eta_R$, a reference conductivity $\kappa_R$, and a reference viscosity $\mu_R$. The ratio of specific heats is denoted by $\gamma$ and taken to be 5/3 throughout this work. The non-dimensional parameters in the above equations are the Reynolds number, defined as $Re = \rho_0 U_0 L / \mu_R$, the Lundquist number, defined as $S = \mu_0 U_0 L / \eta_R$, and the Prandtl number, denoted by $Pr$, which is the ratio of momentum to thermal diffusivity. The non-dimensionalization is carried out using a characteristic tokamak length scale, $L$, and the Alfvén speed $U_0 = B_0 / \sqrt{\mu_0 \rho_0}$, where $B_0$, $\rho_0$, and $\mu_0$ are the characteristic strength of the magnetic field, a reference density and the permeability of free space, respectively. The equations are closed by the following equation of state

$$e = \frac{p}{\gamma - 1} + \frac{\rho}{2} \vec{v} \cdot \vec{v} + \frac{1}{2} \vec{B} \cdot \vec{B}$$  \hspace{1cm} (44)

and the stress tensor is related to the strain as

$$\tau = \mu \left( \nabla \vec{v} + \nabla \vec{v}^T \right) - \frac{2}{3} \mu I \nabla \cdot \vec{v}$$  \hspace{1cm} (45)

We must also have a divergence-free magnetic field at all times, $\nabla \cdot \vec{B} = 0$. 

Lecture 9

In order to calculate the Jacobian in our framework, we look at the expression for the residual again,

\[ F(u) = \int_T \psi f_0(u, \nabla u, x, t) + \nabla \psi \cdot \vec{f}_1(u, \nabla u, x, t) \]  

(46)

and differentiate it with respect to both the solution \( u \) and its gradient \( \nabla u \). In this, we treat \( u \) and \( \nabla u \) as independent variables, much as we do in the Lagrangian analysis of mechanical systems,

\[ \frac{\partial F(u)}{\partial u} = \int_T \psi f_0(u, \nabla u, x, t) + \nabla \psi \cdot \vec{f}_1(u, \nabla u, x, t) \]  

(47)

\[ = \int_T \psi \frac{\delta f_0(u, \nabla u, x, t)}{\delta u} \psi + \psi \frac{\delta f_0(u, \nabla u, x, t)}{\delta \nabla u} \cdot \nabla \psi \]  

(48)

\[ + \nabla \psi \cdot \frac{\delta \vec{f}_1(u, \nabla u, x, t)}{\delta u} \psi + \nabla \psi \cdot \frac{\delta \vec{f}_1(u, \nabla u, x, t)}{\delta \nabla u} \cdot \nabla \psi. \]  

(49)

However, we must understand what these derivatives with respect to the function \( u \), or functional derivatives, mean. You can appeal to a formal calculus of rules, but I think the easiest way to see this is to use a Taylor series approach. For example, we can calculate the derivative of \( x^2 \) using the chain rule, \( 2x \), but we can also derive it

\[ \frac{dx^2}{dx} = \lim_{\epsilon \to 0} \frac{(x + \epsilon)^2 - x^2}{\epsilon} = \lim_{\epsilon \to 0} 2x + \epsilon = 2x. \]  

(50)

We can do that same thing for these functional derivatives, for instance of the Laplacian

\[ Jw = \frac{\delta \Delta u}{\delta u} = \lim_{\epsilon \to 0} \frac{\Delta (u + \epsilon w) - \Delta u}{\epsilon} \]  

(51)

\[ = \lim_{\epsilon \to 0} \frac{\Delta w}{\epsilon} \]  

(52)

\[ = \Delta w \]  

(53)

which is an operational way to get to the Frechet derivative \( L \), which is the unique linear operator such that

\[ \lim_{w \to 0} \frac{||F(u + w) - F(u) - Lw||}{||w||} = 0. \]  

(55)

Let us now look at the more complex case of the \( p \)-Laplacian,

\[ -\nabla \cdot \left( \frac{\nabla u}{2} \right)^{p-2} \nabla u = f. \]  

(56)
We will use our operational definition of the derivative

\[
\frac{\delta f_1}{\delta \nabla u} w
\]

\[
= \lim_{\epsilon \to 0} \frac{1}{2\epsilon} \left( |\nabla(u + \epsilon w)|^{p-2} \nabla(u + \epsilon w) - |\nabla u|^{p-2} \nabla u \right)
\]

(57)

\[
= \lim_{\epsilon \to 0} \frac{1}{2\epsilon} \left( |\nabla(u + \epsilon w)|^{2(p/2-1)} \nabla(u + \epsilon w) - |\nabla u|^{p-2} \nabla u \right)
\]

(58)

\[
= \lim_{\epsilon \to 0} \frac{1}{2\epsilon} \left( \left( |\nabla u|^2 + 2\epsilon \nabla u \cdot \nabla w \right)^{p/2-1} \nabla(u + \epsilon w) - |\nabla u|^{p-2} \nabla u \right)
\]

(59)

\[
= \lim_{\epsilon \to 0} \frac{1}{2\epsilon} \left( \epsilon |\nabla u|^{p-2} \nabla w + 2\epsilon (\nabla u \cdot \nabla w)^{p/2-1} \nabla u \right)
\]

(60)

\[
= \frac{1}{2} |\nabla u|^{p-2} \nabla w + (\nabla u \cdot \nabla w)^{p/2-1} \nabla u
\]

(61)

If \( p = 4 \), we can write this in a nice form

\[
\left( \frac{1}{2} |\nabla u|^{p-2} I + \nabla u \otimes \nabla u \right) \cdot \nabla w,
\]

(62)

where we note that this applies for any input function \( w \). We can expand the \( w \) in our approximation basis and obtain a general form for the Jacobian matrix,

\[
J_{ij}(u) = \int_T \psi_i g_0(u, \nabla u, x, t) \psi_j + \psi_i g_1(u, \nabla u, x, t) \cdot \nabla \psi_j
\]

+ \[ \nabla \psi_i \cdot g_2(u, \nabla u, x, t) \psi_j + \nabla \psi_i \cdot g_3(u, \nabla u, x, t) \cdot \nabla \psi_j \]

(63)

and thus for our \( p \)-Laplacian example we have

\[
\hat{g}_3(u, \nabla u, x, t) = \frac{1}{2} |\nabla u|^{p-2} I + \nabla u \otimes \nabla u,
\]

(64)

or in index notation

\[
g_{3,ij} = \frac{1}{2} |\nabla u|^{p-2} \delta_{ij} + \partial u_i \partial u_j.
\]

(65)

(66)

### References


