Finite Element Modelling for Geosciences: Code Verification and the Method of Manufactured Solutions

Patrick Sanan
ETH Zurich
How do I know the code is working?

- As you’ve no doubt noticed, there are many, many ways to make mistakes in your implementation.
- How do you show that the code “works”?
- One way we’ve seen is to confirm that our finite element solution looks “close” to an exact solution that we know beforehand.
- But ..
  - What does “close” mean?
  - What if I don’t have an exact solution to compare with?
Verification
- Is your code solving the equations correctly?
- Does the code produce approximate solutions that converge at the correct rate?
- The subject of Chapter 9 in the course notes

Validation
- Does the code produce physically-meaningful results?
- Not covered in this course
Error estimates

- If we can construct an exact solution to the continuous problem, we can use it to verify our code.
- The FEM is well-known to converge, if implemented correctly!
- Convergence is expressed in terms of the grid spacing and an error norm:
  \[ \| u - u^h \| \]
- If an analytical expression is available, one can compute the solution at different spacings and examine how quickly it converges:
  \[ \| u - u^h \| \leq Ch^k \]
- The rate of convergence can be computed by fitting a straight line to a log-log plot of the grid spacing versus the error.
- Obvious issue: what if we don’t know \( u \)?
Computing errors

- The $L^2$ error

$$
\left( \int_\Omega (u^h(x, y) - u(x, y))^2 \, d\vec{x} \right)^{\frac{1}{2}}
$$

- An integral over the entire domain; compute it with quadrature!

- Sometimes one sees “superconvergence” if only comparing nodal values.

```matlab
err = sqrt(sum((T-T_exact).^2)) \% Not an L^2 norm!
```
The Method of Manufactured Solutions (MMS)

- General approach - works for most PDE.
- Intuition: choose the solution you want and adjust the forcing and boundary terms to match.
- Choose ("manufacture") your solution
- Plug it into your equations (physics + material properties)
- Extract the required boundary and forcing terms (CAS tools like SAGE/SymPy/MAPLE/Mathematica/etc. are helpful)
- Now you can use your FEM code to compute an approximate solution, and compare.

How to choose your manufactured solution
  - Make it complex enough
  - In particular, avoid low-order polynomials which might be exactly captured by your finite element basis functions
  - Good choices are combinations of sinusoidal or exponential functions
Assume we’re interested in approximate solutions to the Poisson equation (steady-state heat equation) on the unit square

\[ \nabla \cdot (\kappa(x, y) \nabla T) = f(x, y), \quad \kappa(x, y) = 1 + x \]

Note that we specified the coefficient structure, but not the boundary conditions or forcing.

Choose our own solution (depending on both \( x \) and \( y \), not just a polynomial)

\[ T_{\text{MMS}} \doteq \sin(x) \cos(2y) \]
Insert $T_{\text{MMS}}$ into our equation\(^1\) to obtain the forcing:

$$f(x, y) = \frac{\partial}{\partial x} \left( (1 + x) \frac{\partial T_{\text{MMS}}}{\partial x} \right) + \frac{\partial}{\partial y} \left( (1 + x) \frac{\partial T_{\text{MMS}}}{\partial y} \right)$$

$$= \cos(2y) \left( \cos(x) - 5(x + 1) \sin(x) \right)$$

Use $T_{\text{MMS}}$ to define boundary conditions, for example Dirichlet conditions:

$$T(x, 0) = T_{\text{MMS}}(x, 0), \quad T(x, 1) = T_{\text{MMS}}(x, 1), \quad \forall x \in [0, 1]$$

$$T(0, y) = T_{\text{MMS}}(0, y), \quad T(1, y) = T_{\text{MMS}}(1, y), \quad \forall y \in [0, 1]$$

---

\(^1\) I lazily just plugged this into Wolfram Alpha: $\text{Div}[(1+x)\cdot\text{Grad}[\sin(x)\cdot\cos(2*y)]]$
L2 error computation MATLAB code snippet

```matlab
L2_error_2 = 0;
for i_el = 1:el_tot
    nodes_el = element_node_map(:,i_el);
    coord_el = gcoord(:,nodes_el)';
    T_el = T(nodes_el);

    for i_pt = 1:pts_per_el
        jac = dNdxi(:,i_pt) * coord_el;
        det_jac = det(jac)

        coord_pt = N(i_pt,:) * coord_el;
        T_mms_pt = T_MMS(coord_pt(1), coord_pt(2));
        T_fe_pt = N(i_pt,:) * T_el;

        L2_error_2 = L2_error_2 + (T_mms_pt - T_fe_pt)^2 * det_jac * w(i_pt);
    end
L2_error = sqrt(L2_error_2)
```

Patrick Sanan (ETH Zurich)
1d to 2d 9 / 11
The expected convergence rate, from the course notes, is

\[
\left[ \int_{\Omega} (T - T^h)^2 \, d\vec{x} \right]^{\frac{1}{2}} \leq C h^{p+1}
\]

Our modified code shows slopes of approximately 2 and 3 on a log-log plot, so we are now much more confident that the implementation is correct.

Bilinear elements, \( p = 1 \)  
Biquadratic elements, \( p = 2 \)
If using MMS for something like the Stokes equations (Friday), you will perhaps run into the following issue.

As an example, let’s say that I have written a FE code to solve this (very) simple PDE:

\[
\frac{d}{dx} \left( k(x) \frac{du}{dx} \right) = 0, \quad u \in [0, 1], \quad u(0) = 1, u(1) = 2
\]

What issue might I run into if I apply the MMS?

I may have written my code assuming zero forcing!

So I may have to modify my weak form to take the additional forcing into account. Don’t forget to consider terms like

\[
F = \int_{\Omega} N^T f \, d\vec{x}
\]
If using MMS for something like the Stokes equations (Friday), you will perhaps run into the following issue:

As an example, let’s say that I have written a FE code to solve this (very) simple PDE:

\[
\frac{d}{dx} \left( k(x) \frac{du}{dx} \right) = 0, \quad u \in [0, 1], \quad u(0) = 1, \quad u(1) = 2
\]

What issue might I run into if I apply the MMS?

I may have written my code assuming zero forcing!

So I may have to **modify my weak form** to take the additional forcing into account. Don’t forget to consider terms like

\[
F = \int_{\Omega^e} N^T f \, d\vec{x}
\]