Overview

June 1, 2023

The goal is to discretize a weak equation for an unknown $u \in U$ of the form

$$\int_{\Omega} f\left(v, u, v_x, u_x\right)\, dx = 0 \quad \forall v \in V \tag{1}$$

where f is linear in the test functions v and v_x but may be nonlinear in u and its derivatives. In this introductory section we make no attempt to be fully general; our purpose is to convey the fundamental idea. Choose a finite dimensional approximating space $U^N \subset U$ and test function space $V^N \subset V$, with bases $\{\phi_j\}_{j=1}^{N}$ and $\{\psi_i\}_{i=1}^{N}$ for U and V respectively. We approximate the solution by

$$u\left(x\right) = \sum_{j=1}^{N} u_j \phi_j\left(x\right).$$

From [7] and [8] we know that the elements of the discrete residual $\mathbf{b}$ and Jacobian J in a linearization of 1 are computed as

$$b_i = \int_{\Omega} \frac{\partial f}{\partial v} \psi_i\, dx + \int_{\Omega} \frac{\partial f}{\partial v_x} \frac{\partial \psi_i}{\partial x}\, dx \tag{2}$$

and

$$J_{ij} = \int_{\Omega} \frac{\partial^2 f}{\partial v \partial u} \psi_i \phi_j\, dx + \int_{\Omega} \frac{\partial^2 f}{\partial v_x \partial u} \frac{\partial \psi_i}{\partial x} \phi_j\, dx + \int_{\Omega} \frac{\partial^2 f}{\partial v \partial u_x} \psi_i \frac{\partial \phi_j}{\partial x}\, dx + \int_{\Omega} \frac{\partial^2 f}{\partial v_x \partial u_x} \frac{\partial \psi_i}{\partial x} \frac{\partial \phi_j}{\partial x}\, dx. \tag{3}$$

One would then solve

$$J \delta \mathbf{u} = \mathbf{b}$$

for the Newton step $\delta \mathbf{u}$ in the coefficients u_j; if the PDE is linear then only one Newton step is needed.

The coefficients in the integrand of the residual are the partials of f with respect to the test function and its derivatives. The coefficients in the Jacobian are the mixed second partials of the f with respect to the test function, unknown function, and derivatives. Note that in the Jacobian we encounter only those second partials with one test function and one unknown function. Since f is linear in v there can be no nonzero second partials such as $\frac{\partial^2 f}{\partial v^2}$ or $\frac{\partial^2 f}{\partial v \partial v_x}$, and since at the top level we always need partials with respect to test functions we can ignore partials such as $\frac{\partial^2 f}{\partial u^2}$ and $\frac{\partial^2 f}{\partial u \partial u_x}$. First partials on the unknown

such as $\frac{\partial f}{\partial u}$ and $\frac{\partial f}{\partial u_x}$ won't be needed in the final residual calculation, but may be needed in intermediate steps in the computation of second partials such as $\frac{\partial^2 f}{\partial v \partial u}$.
Equations 2 and 3 connect the function f representing the PDE, the basis functions ψ_i and ϕ_i representing the discretization, and elements of the residual vector and Jacobian matrix in the linear system for the coefficients u_j (or, their Newton steps δu_j when the equation is nonlinear). The evaluation of basis functions and formation of a sparse matrix are the bread and butter of a finite element code (see, for example, [4] or [2]), and we don't discuss them further here. We concentrate on the evaluation of the coefficients, for which the key step is the evaluations of partials with respect to the test and unknown functions and their spatial derivatives.

1 Approaches to coding these ideas

1.1 Code generation (FEniCS)

The FEniCS project [6] uses compile-time automatic differentiation to generate source code to evaluate the required partial derivatives. and embed that code within integration loops that compute the required discrete matrix and vector entries. In principle, an optimizing compiler can turn the generated code into very efficient machine code. A disadvantage is that for complicated problems the generated code can use a great deal of memory. In his thesis [1] on a complex equation set from micromagnetics, Josh Engwer found frequent crashes from blowing out memory even on machines with 16-32 GB of RAM.

1.2 Runtime evaluation of an expression graph (Sundance)

The Sundance project [8] approaches the problem in a different way. Code generation is not used. Instead, an expression graph is built at runtime, and that graph is used for evaluation of all necessary partial derivatives. In a set of experiments with Poisson and Stokes problems with different basis orders, it was found in [8] that Sundance outperformed FEniCS in problem assembly speed by factors of 2-5, and for large problems FEniCS crashed.
So what's the catch? There are several:

- Sundance is very intricate C++ code that's difficult to maintain. Much of the complexity is in logic that could be done more easily and more clearly in Python without a significant performance hit; certain core numerical tasks should be still done in C/C++ called from Python via the `pybind` code binding system.
  - Much of the complexity came from using an extended chain rule that mixed spatial and functional differentiation. In the revision, spatial differentiation is handled separately as a preprocessing step. Hereafter, we assume that preprocessing has been done (I've already coded it).

- Sundance wasn't designed for vector-valued basis functions, and the complexity of adding them would require a substantial rewrite of the intricate C++ code. Better to start fresh. (Vector problems such as the Navier-Stokes and Maxwell equations could be solved, but only by using aggregated scalar basis functions rather than true vector bases. This isn't a problem for Navier-Stokes, but causes severe limitations for Maxwell.)
- Sundance has some bugs in its most general evaluation functions. These have shown up in a number of sufficiently complex PDEs.

1.2.1 Game plan

- Reworking of evaluation engine in Python to handle scalar, vector, tensor, and aggregate expressions
- Connection of evaluation engine to FEM assembly system (meshes, basis functions, DOF maps, discrete functions)
- Plans:
  - Short term
    - Expression analysis to identify evaluation requirements at each node
    - Pure Python evaluation; no calls to optimized C++
      - String evaluator: instead of doing numerical computation, evaluate to text strings so we can check the output. We can write to LaTeX string output for checking by eye, and Mathematica input form output for checking against Mathematica
  - Medium term:
    - High-performance evaluation for scalar expressions (all that's needed for Kohl's thesis). Can re-use some existing Sundance code for this. (Kohl/Lourdes/Harold)
    - Connection to FEM assembly system (meshes, basis functions, DOF maps, discrete functions) (Kohl/Lourdes)
    - Investigate design of high-performance evaluation of vector/tensor/aggregate expressions (Harold)
  - Longer term:
    - Kohl: Apply to problems in mathematical biology
    - Harold: Build a system capable of doing a full micromagnetics problem w/o blowing out memory

2 How it works

The first step is to understand how expressions are represented in graphs. A simple example is shown in figure 1; this is a graph representation of the expression

$$f = \frac{\partial v}{\partial x}\frac{\partial u}{\partial x} + \frac{\partial v}{\partial x}\sin(u) + vu.$$

Each node in the graph represents a quantity to be computed in the evaluation of f. The evaluation of f "flows upwards" as indicated in the arrows; the graph is therefore **directed**. The graph is *almost* a tree; it fails to be a tree because the "leafs", the variables v and u, are used in different evaluation paths. In more complex expressions, it's possible to other nodes to be used in different evaluation paths; this would be seen, for example, in an expression such as

$$\frac{x^2}{\sqrt{1+x^2}}$$

where the subexpression x^2 is used twice. Topologically, there are "loops" in the graph: for example,

$$E_1 - E_{10} - E_6 - E_7 - E_9 - E_1$$

forms a closed loop. However, once *direction* of evaluation is accounted for, we see that this loop doesn't form a *cycle*: information flows from E_1 to E_{10} via two different paths rather than "around the loop". This type of graph is called a **directed acyclic graph,** or **DAG** for short.

With an expression's DAG in hand, we can evaluate f and any derivative numerically by proceeding upwards through the graph. If only f itself is needed, then

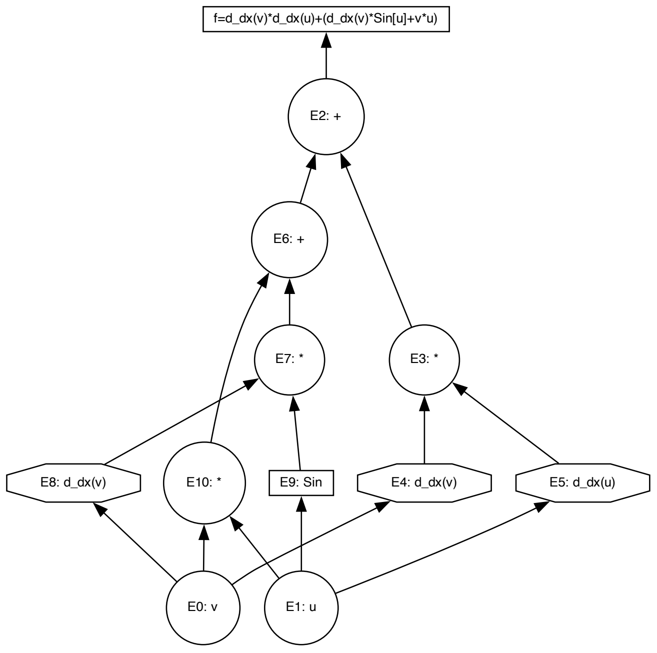

Figure 1:

2.1 Same graph, different contexts, different evaluation logic

We've seen that the same expression may be evaluated differently in different contexts. We may compute a residual only, a Jacobian only, or both. As a more complex example:

in a PDE-constrained optimization problem with Lagrangian $L(u,\lambda,\alpha) = F(u,\lambda,\alpha) - \lambda G(u,\alpha)$, in different stages of an optimization algorithm we might evaluate L with $\lambda = 0$ to obtain the objective function, and in the process of evaluating derivatives we will need to take variations with respect to λ to obtain a state equation for u given α, with respect to u to obtain an adjoint equation for λ given u and α , or with respect to α to obtain a design equation for α given λ and u.

It is therefore important to distinguish between an expression and its evaluation logic. Since one expression can have multiple, context-dependent, evaluation procedures, we will create for each context an **evaluator**, and store these in a dictionary that maps context to evalulator. For example,

```
contextToEvalMap = {
    context1 : evaluator1 ,
    context2 : evaluator2
}
```

2.2 Propagating derivatives

Let f be a node in the DAG that depends on N direct predecessor nodes $a_1, a_2, \cdots, a_N$. The "direct predecessors" are those nodes connected to node f by single edges pointing upwards to f. For example, in our graph 1, node E_7 has direct predecessors E_8 and E_9. Values at nodes are evaluated numerically by carrying out the specified operation on the results from their predecessor nodes. However, we also need derivatives: $\frac{\partial f}{\partial u}$, $\frac{\partial^2 f}{\partial u \partial v}$ and so on. These are propagated up from predecessors using the chain rule.

2.2.1 Notation for functional derivatives

We'll encounter derivatives of expressions with respect to functions and their spatial derivatives, such as

$$\frac{\partial f}{\partial u}, \quad \frac{\partial f}{\partial \frac{\partial u}{\partial y}}, \quad \frac{\partial f}{\partial \nabla \times \mathbf{v}} \quad \text{or} \quad \frac{\partial^2 f}{\partial v \partial \frac{\partial u}{\partial x}}.$$

If you've seen Lagrangian mechanics (see *e.g.* [5, 9, 3]), this idea of differentiating with respect to derivatives of functions will be familiar.

2.2.2 The chain rule for scalar-valued functions of scalar arguments and variables

The simplest case gives the main idea. Let f depend on N arguments $a_1, a_2, \cdots, a_N$, and consider differentiation with respect to M variables $u_1, u_2, \cdots, u_M$. Keep in mind that the differentiation variables u_1, etc, might be functions, spatial derivatives of functions, or design parameters. We'll use mid-range Greek letters λ, μ, ν for argument indices, and enumerate differentiation varables u, v, w explicitly. The first derivative of f is

$$\frac{\partial f}{\partial u} = \sum_{\lambda=1}^{N} \frac{\partial f}{\partial a_\lambda} \frac{\partial a_\lambda}{\partial u}.$$

The second derivative (mixed partial with respect to u and v) is

$$\frac{\partial^2 f}{\partial u \partial v} = \sum_{\lambda=1}^{N} \frac{\partial f}{\partial a_\lambda} \frac{\partial^2 a_\lambda}{\partial u \partial v} + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \frac{\partial^2 f}{\partial a_\lambda \partial a_\mu} \frac{\partial a_\lambda}{\partial u} \frac{\partial a_\mu}{\partial v}.$$

The first and second derivatives are sufficient for linear and nonlinear forward PDE problems. In sensitivity analysis (including direct sensitivities in gradient-based optimization) and eigenvalue problems, a third derivative is also needed. The chain rule for the third derivative is

$$\begin{aligned}\frac{\partial^3 f}{\partial u \partial v \partial w} = \sum_{\lambda=1}^{N} \frac{\partial f}{\partial a_\lambda} \frac{\partial^3 a_\lambda}{\partial u \partial v \partial w} + \\ + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \frac{\partial^2 f}{\partial a_\lambda \partial a_\mu} \left[\frac{\partial^2 a_\lambda}{\partial u \partial v} \frac{\partial a_\mu}{\partial w} + \frac{\partial^2 a_\lambda}{\partial u \partial w} \frac{\partial a_\mu}{\partial v} + \frac{\partial^2 a_\lambda}{\partial v \partial w} \frac{\partial a_\mu}{\partial u} \right] \\ + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \sum_{\nu=1}^{N} \frac{\partial^3 f}{\partial a_\lambda \partial a_\mu \partial a_\nu} \frac{\partial a_\lambda}{\partial u} \frac{\partial a_\mu}{\partial v} \frac{\partial a_\nu}{\partial w}.\end{aligned}$$

2.2.3 The chain rule for tensor-valued functions of tensor arguments and variables

Let $I, J, K, \cdots$ be integer multiindices $I = (i_1, i_2, \cdots, i_m)$ and so on. It may be that $\dim(I) \neq \dim(J)$. If $\dim(I) = 0, 1$, or 2 then an object indexed by I is a scalar, vector, or 2-tensor respectively. Tensors of higher order will be encountered. Let f_I, u_J, v_K, and w_K be tensor-valued quantities; f_I is a tensor of order $\dim(I)$, and so on. We're concerned with the multivariate, multi-argument chain rule for a tensor-valued function f of tensor-valued arguments $a_1, a_2, \cdots, a_N$ and tensor variables $u, v, \cdots$, written as

$$f_I \left(a_1 \left(u_J, v_K, \cdots \right), a_2 \left(u_J, v_K, \cdots \right), \cdots, a_n \left(u_J, v_K, \cdots \right) \right).$$

We'll use Greek letters $\lambda, \mu, \nu, \cdots$ for argument indices; these are natural numbers. The first partial of f with respect to u is

$$\frac{\partial f_I}{\partial u_J} = \sum_{\lambda=1}^{N} \frac{\partial f_I}{\partial a_\lambda} \frac{\partial a_\lambda}{\partial u_J}.$$

Let's unpack this. The result is a tensor of order $\dim(I) + \dim(J)$. For example, if f and u are scalars, then the result is a scalar; if one of f and u is a vector and the other a scalar, than the result is a vector; if f and u are vectors, the result is a 2-tensor (*i.e.*, a matrix), and so on. What about the tensor structure of the argument a_λ? It's irrelevant to this formula, because we interpret the operation

$$\frac{\partial f_I}{\partial a_\lambda} \frac{\partial a_\lambda}{\partial u_J} \qquad (4)$$

to represent contraction over all the indices of a in the tensors

$$\frac{\partial f_I}{\partial a_\lambda} \text{ and } \frac{\partial a_\lambda}{\partial u_J}.$$

Important: the index λ is **not** a tensor index, it's the integer index of the argument a in the list of arguments of f. The tensor indices of a_λ aren't shown here, and don't need to be since we understand implicitly that they're to be contracted away when performing the operation in equation 4.

Using the conventions described above, we next write out the chain rules for second and third partials,

$$\frac{\partial^2 f_I}{\partial u_J \partial v_K} = \sum_{\lambda=1}^{N} \frac{\partial f_I}{\partial a_\lambda} \frac{\partial^2 a_\lambda}{\partial u_J \partial v_K} + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \frac{\partial^2 f_I}{\partial a_\lambda \partial a_\mu} \frac{\partial a_\lambda}{\partial u_J} \frac{\partial a_\mu}{\partial v_K}$$

and

$$\begin{aligned}\frac{\partial^3 F_I}{\partial u_J \partial v_K \partial w_L} = & \sum_{\lambda=1}^{N} \frac{\partial f_I}{\partial a_\lambda} \frac{\partial^3 a_\lambda}{\partial u_J \partial v_K \partial w_L} + \\ & + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \frac{\partial^2 f_I}{\partial a_\lambda \partial a_\mu} \left[\frac{\partial^2 a_\lambda}{\partial u_J \partial v_K} \frac{\partial a_\mu}{\partial w_L} + \frac{\partial^2 a_\lambda}{\partial u_J \partial w_L} \frac{\partial a_\mu}{\partial v_K} + \frac{\partial^2 a_\lambda}{\partial v_K \partial w_L} \frac{\partial a_\mu}{\partial u_J} \right] \\ & + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \sum_{\nu=1}^{N} \frac{\partial^3 f_I}{\partial a_\lambda \partial a_\mu \partial a_\mu} \frac{\partial a_\lambda}{\partial u_J} \frac{\partial a_\mu}{\partial v_K} \frac{\partial a_\nu}{\partial w_L}\end{aligned}$$

2.3 What to compute (or not)

Given an integrand in a weak form, we need certain partial derivatives in order to form vectors and matrices with operations such as those in equations 2 and 3. Let g be an expression depending directly on N arguments $a_1, a_2, \cdots, a_N$. For purposes of this section the tensor structure of g, its arguments, and the variables u, v, etc are irrelevant, so we treat all variables as scalars.

Throughout this section, an expression being "zero" or "nonzero" refers to whether that expression is *structurally* zero or nonzero. For example, if $g(a_1, a_2) = a_1 a_2$, then $\frac{\partial g}{\partial a_1} = a_2$ which will be nonzero except when $a_2 = 0$; since it's zero only for special values of the arguments, we call it structurally nonzero. However, $\frac{\partial^2 g}{\partial a_1^2}$ is zero for all arguments, so we call it structurally zero.

Define the following sets.

- $A_d(g)$ is the set of nonzero d-th order partial derivatives *available* from g.
  - Example: if $g = uv + vu_x$, then $A_1(g) = \left\{\frac{\partial}{\partial u}, \frac{\partial}{\partial v}, \frac{\partial}{\partial u_x}\right\}$ and $A_2(g) = \left\{\frac{\partial^2}{\partial u \partial v}, \frac{\partial^2}{\partial u \partial v_x}\right\}$. Note that $A_1(g)$ doesn't contain $\frac{\partial}{\partial v_x}$, and $A_2(g)$ doesn't contain $\frac{\partial^2}{\partial v^2}$, $\frac{\partial^2}{\partial u^2}$, and so on.
  - We can write sets of partials compactly by simply listing the differentiation variables, for example, $\{u, v, u_x\}$ instead of $\left\{\frac{\partial}{\partial u}, \frac{\partial}{\partial v}, \frac{\partial}{\partial u_x}\right\}$, and $\{(u, v), (u, v_x)\}$ instead of $\left\{\frac{\partial^2}{\partial u \partial v}, \frac{\partial^2}{\partial u \partial v_x}\right\}$
- $R_d(g)$ is the set of nonzero d-th order partial derivatives *required* from g in order to compute whatever quantities are needed from g or its successors.

  - In calculation of a residual vector given an integrand f, no second partials are needed, so $R_2(f) = \emptyset$. With $f = uvq(x) + vp(x) + v_x u_x$, then $R_1(f) = \left\{\frac{\partial}{\partial v}, \frac{\partial}{\partial v_x}\right\}$. When computing a Jacobian, we'll have $R_2(f) = \left\{\frac{\partial^2}{\partial u \partial v}, \frac{\partial^2}{\partial u_x \partial v_x}\right\}$.

When determining A_d, we work from predecessor to successor (bottom to top) in the expression graph, ending up at the final expression to be computed. When determining R_d, we work from the final expression (where we know what quantities are required) backwards to the predecessors.

Some sets that will help in this process are

- $Q_d(g)$ is the set of d-tuples of integers $(\lambda_1, \lambda_2, \cdots, \lambda_d)$ such that the partial derivative
$$\frac{\partial^d g}{\partial a_{\lambda_1} \partial a_{\lambda_2} \cdots \partial a_{\lambda_d}}$$
is structurally nonzero. We don't reduce over permutations; for example, the pairs $(1,2)$ and $(2,1)$ are both counted. In general, if g depends on n arguments then $Q_1(g)$ will be $\{1, 2, \cdots, n\}$. Sets for higher-order derivatives will depend on the functional dependence of g upon its arguments.
  - Example: When g is a linear operator on the arguments, then $Q_d(g) = \emptyset$ for $d > 1$.
  - Example: When g is a product of two arguments, $g = a_1 a_2$, then $Q_2(g) = \{(1,2), (2,1)\}$ and $Q_d(g) = \emptyset$ for $d > 2$.
  - Example: When g is an unspecified nonlinear function of two arguments, then $Q_2(g) = \{(1,1), (1,2), (2,1), (2,2)\}$ and
  $$Q_3(g) = \{(1,1,1), (1,1,2), (1,2,1), (1,2,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2)\}.$$
  In this case, $Q_d(g)$ will contain all 2^d tuples with members 1 and 2. In a 3-argument nonlinear function, we'd have 3^d tuples, and so on.

  Let $Q_d(g, \ell)$ be the ℓ-th d-tuple in $Q_d(g, \ell)$, and $Q_d(g, \ell, j)$ be the j-th argument index in $Q_d(g, \ell)$. We'll write $D_{Q_d(g,\ell)} g$ for
$$\frac{\partial^d g}{\partial a_{Q_d(g,\ell,1)} \partial a_{Q_d(g,\ell,2)} \cdots \partial a_{Q(g,\ell,d)}}.$$

- Let p be a mixed partial of order m such as
$$\frac{\partial^m}{\partial u_1 \partial u_2 \cdots \partial u_m}.$$
We'll represent this compactly as the sorted m-tuple $(u_1, \cdots, u_m)$, and also with the notation D_p. Then the map $F(p, n)$ is defined as
$$F(p,n) = \begin{cases} \text{arrangements of } p \text{ into } n \text{ non-empty boxes} & 1 \le n \le m \\ \emptyset & \text{otherwise} \end{cases}$$
where here we *do* reduce over permutations.

- Example: $p = (u)$: $F(p,1) = \{(u)\}$
- Example: $p = (u,v)$: $F(p,1) = \{(u,v)\}$, $F(p,2) = \{((u),(v))\}$
- Example: $p = (u,v,w)$:
  * $F(p,1) = \{(u,v,w)\}$
  * $F(p,2) = \{((u),(v,w)),((u,v),(w)),((u,v),(w))\}$
  * $F(p,3) = \{((u),(v),(w))\}$

To identify a particular n-tuple in F, we write $F(p,n,k)$ for the k-th n-tuple of partials in $F(p,n)$.

- Example: with $p = (u,v,w)$, we have $F(p,2,1) = ((u),(v,w))$ and $F(p,2,3) = ((u,v),w)$.

Finally, $F(p,n,k,j)$ is the j-th partial in $F(p,n,k)$.

- Example: with $p = (u,v,w)$, we have $F(p,2,1,1) = (u)$ and $F(p,2,1,2) = (v,w)$. Then
$$D_{F(p,2,1,1)}g = \frac{\partial g}{\partial u}$$
and
$$D_{F(p,2,1,2)} = \frac{\partial^2 g}{\partial v \partial w}.$$

In the notation defined above, the chain rule is written out as

$$D_p g = \sum_{n=1}^{|p|} \sum_{\ell=1}^{|Q_n(g)|} D_{Q_n(g,\ell)}g \sum_{k=1}^{|F(p,n)|} \prod_{j=1}^{|F(p,n,k)|} D_{F(p,n,k,j)} a_{Q_n(g,\ell,j)}.$$

Pseudocode for straightforward evaluation is in algorithm 1.

2.3.1 Determination of available set $A_d(g)$

- Case $d = 0$: It's assumed that every node is structurally nonzero, so that $A_d(g) = \{I\}$ where I is the identity operator $Ig = g$, that is, the zeroth order partial of g.

- Case $d = 1$: Refer to the chain rule
$$\frac{\partial g}{\partial u} = \sum_{\lambda=1}^{n} \frac{\partial g}{\partial a_\lambda} \frac{\partial a_\lambda}{\partial u}.$$
The The quantity $\frac{\partial g}{\partial u}$ will be structurally nonzero if, for some λ, $\frac{\partial g}{\partial a_\lambda}$ is structurally nonzero and $\frac{\partial}{\partial u} \in A_1(a_\lambda)$. For first derivatives, we can always assume that $\frac{\partial g}{\partial a_\lambda}$ is structurally nonzero for each λ, so we then have
$$A_1(g) = \bigcup_{\lambda=1}^{n} A_1(a_\lambda), \tag{5}$$
or, equivalently,
$$A_1(g) = \bigcup_{\lambda \in Q_1(g)} A_1(a_\lambda).$$

Algorithm 1 Evaluation of chain rule

```
Initialize $D_p g \leftarrow 0$.
Find $F(p)$.
for $n = 1$ to $|p|$ do
    Find $Q_n(g)$
    for $q_\ell \in Q_n(g)$ do
        Compute $D_{q_\ell} g$
        Initialize $\mathrm{tmp}_1 \leftarrow 0$
        for $f_k \in F(p, n)$ do
            $\mathrm{tmp}_2 \leftarrow 1$
            for $f_{k,j} \in f_k$ do
                $\mathrm{tmp}_2 \leftarrow \mathrm{tmp}_2 * D_{f_{k,j}} a_{q_{\ell,j}}$
            end for
            $\mathrm{tmp}_1 \leftarrow \mathrm{tmp}_1 + \mathrm{tmp}_2$
        end for
        $D_p g \leftarrow D_p g + \mathrm{tmp}_1 * D_{q_\ell} g$
    end for
end for
Return $D_p g$
```

- Case $d = 2$: Refer again to the chain rule,

$$\frac{\partial^2 g}{\partial u \partial v} = \sum_{\lambda=1}^{n} \frac{\partial g}{\partial a_\lambda} \frac{\partial^2 a_\lambda}{\partial u \partial v} + \sum_{\lambda=1}^{n} \sum_{\mu=1}^{n} \frac{\partial^2 g}{\partial a_\lambda \partial a_\mu} \frac{\partial a_\lambda}{\partial u} \frac{\partial a_\mu}{\partial v}.$$

The first sum is dealt with as in the first-order case: it will be nonzero if for some λ we have $\frac{\partial^2 a_\lambda}{\partial u \partial v}$ nonzero. For the second sum, we find that

$$\frac{\partial^2 g}{\partial a_\lambda \partial a_\mu} \frac{\partial a_\lambda}{\partial u} \frac{\partial a_\mu}{\partial v}$$

is nonzero if $(\lambda, \mu) \in Q_1(g)$ and $\frac{\partial}{\partial u} \in A_1(a_\lambda)$ and $\frac{\partial}{\partial v} \in A_1(a_\mu)$. Turn these logical statements into set operations to find

$$A_2(g) = \left[\bigcup_{\lambda \in Q_1(g)} A_2(a_\lambda) \right] \bigcup \left[\bigcup_{(\lambda,\mu) \in Q_2(g)} A_1(a_\lambda) \otimes A_1(a_\mu) \right], \tag{6}$$

where $\otimes$ is the Cartesian product operator.

- Case $d = 3$: Proceed similarly to find

$$A_3(g) = \left[\bigcup_{\lambda=1}^{n} A_3(a_\lambda) \right] \bigcup \left[\bigcup_{(\lambda,\mu) \in Q_2(g)} A_1(a_\lambda) \otimes A_2(a_\mu) \right] \bigcup \left[\bigcup_{(\lambda,\mu,\nu) \in Q_3(g)} A_1(a_\lambda) \otimes A_1(a_\mu) \otimes A_1(a_\nu) \right]. \tag{7}$$

Notice that in all cases, A_d depends on A_{d-1}, A_{d-2}, and so on; therefore we must first compute A_1, then A_2, then A_3.

Now we consider some special cases.

- **A linear operation.** When $g\left(a_1, a_2, \cdots, a_n\right)$ is linear, we have $Q_1\left(g\right) = \left\{1, 2, \cdots, n\right\}$ as always, and then $Q_d\left(g\right) = \emptyset$ for all $d > 1$. Then, for all d we find
$$A_d\left(g\right) = \bigcup_{\lambda=1}^{n} A_d\left(a_\lambda\right).$$
- **A nonlinear unary operation**. Without further information about g we can only assume that $g^{(d)}\left(a\right)$ is structurally nonzero for all $d \geq 0$. Then $Q_d\left(g\right) = \left\{1\right\}$, and the available sets are
$$A_1\left(g\right) = A_1\left(a\right)$$
$$A_2\left(a\right) = A_2\left(a\right) \bigcup \left(A_1\left(a\right) \otimes A_1\left(a\right)\right)$$
$$A_3\left(a\right) = A_3\left(a\right) \bigcup \left(A_1\left(a\right) \otimes A_2\left(a\right)\right) \bigcup \left(A_1\left(a\right) \otimes A_1\left(a\right) \otimes A_1\left(a\right)\right).$$
- **A binary product.** For every binary product (whether scalar times scalar, dot product, cross product, Frobenius product, etc) we have
$$g = a_1 \cdot a_2$$
and thus $Q_1\left(g\right) = \left\{1, 2\right\}$, $Q_2\left(g\right) = \left\{\left(1,2\right), \left(2,1\right)\right\}$, and $Q_d\left(g\right) = \emptyset$ for all $d > 2$. Then
$$A_1\left(g\right) = A_1\left(a_1\right) \bigcup A_1\left(a_2\right)$$
$$A_2\left(g\right) = A_2\left(a_1\right) \bigcup A_2\left(a_2\right) \bigcup \left(A_1\left(a_1\right) \otimes A_1\left(a_2\right)\right)$$
$$A_3\left(g\right) = A_3\left(a_1\right) \bigcup A_3\left(a_2\right) \bigcup \left[A_1\left(a_1\right) \otimes A_2\left(a_2\right)\right] \bigcup \left[A_2\left(a_1\right) \otimes A_1\left(a_2\right)\right]$$
- **A quotient.** The operation is
$$g = \frac{a_1}{a_2}.$$
We have $Q_1\left(g\right) = \left\{1, 2\right\}$, $Q_2\left(g\right) = \left\{\left(1,2\right), \left(2,2\right)\right\}$, and $Q_3\left(g\right) = \left\{\left(1,2,2\right), \left(2,2,2\right)\right\}$. Then from equation 5 we have
$$A_1\left(g\right) = A_1\left(a_1\right) \bigcup A_1\left(a_2\right)$$
and from equation 6,
$$A_2\left(g\right) = A_2\left(a_1\right) \bigcup A_2\left(a_2\right) \bigcup \left(A_1\left(a_1\right) \otimes A_1\left(a_2\right)\right) \bigcup \left(A_1\left(a_2\right) \otimes A_1\left(a_2\right)\right).$$
We don't write out the expression for $A_3\left(g\right)$; it can be computed from equation 7

2.3.2 Determination of required set $R_d\left(g\right)$

It will help to start with some examples. Consider first a sum,
$$g\left(a_1, a_2\right) = a_1 + a_2,$$
and suppose that we need the partials listed in the set $R_1\left(g\right)$. The central question in this section is: given $R_1\left(g\right)$, which results are required from the arguments a_1 and a_2 to

produce the partials in $R_1(g)$. For the case of addition, here's the answer: if some partial p is required from g, and is available from an argument a_i, then it is required by a_i. Put more succinctly,

$$R_1(a_i) = R_1(g) \bigcap A_1(a_i) \quad i = 1, 2.$$

This will hold for all d:

$$R_d(a_i) = R_d(g) \bigcap A_d(a_i) \quad i = 1, 2.$$

Now consider a product:

$$g(a_1, a_2) = a_1 a_2$$

with first and second partials

$$\frac{\partial g}{\partial u} = a_1 \frac{\partial a_2}{\partial u} + a_2 \frac{\partial a_1}{\partial u}$$

$$\frac{\partial^2 g}{\partial u \partial v} = a_1 \frac{\partial^2 a_2}{\partial u \partial v} + \frac{\partial a_1}{\partial v}\frac{\partial a_2}{\partial u} + \frac{\partial a_1}{\partial v}\frac{\partial a_2}{\partial u} + a_2 \frac{\partial^2 a_2}{\partial u \partial v}.$$

Suppose $\frac{\partial g}{\partial u} \in R_1(g)$. If a_1 and $\frac{\partial a_2}{\partial u}$ are nonzero, then we need both; similarly for a_2 and $\frac{\partial a_1}{\partial u}$. Put in terms of our available sets:

- If $\frac{\partial a_2}{\partial u} \in A_1(a_2)$, then put $\frac{\partial a_2}{\partial u}$ into $R_1(a_2)$ and put a_1 into $R_0(a_1)$
- If $\frac{\partial a_1}{\partial u} \in A_1(a_1)$, then put $\frac{\partial a_1}{\partial u}$ into $R_1(a_1)$ and put a_2 into $R_0(a_2)$

For the second derivatives, we have: if $\frac{\partial^2 g}{\partial u \partial v} \in R_2(g)$, then

- If $\frac{\partial^2 a_2}{\partial u \partial v} \in A_2(a_2)$, then put $\frac{\partial^2 a_2}{\partial u \partial v}$ into $R_2(a_2)$ and put a_1 into $R_0(a_1)$
- If $\frac{\partial^2 a_1}{\partial u \partial v} \in A_2(a_1)$, then put $\frac{\partial^2 a_1}{\partial u \partial v}$ into $R_2(a_1)$ and put a_2 into $R_0(a_2)$
- If $\frac{\partial a_1}{\partial u} \in A_1(a_1)$ and $\frac{\partial a_2}{\partial v} \in A_1(a_2)$, then put $\frac{\partial a_1}{\partial u}$ into $R_1(a_1)$ and put $\frac{\partial a_2}{\partial v}$ into $R_2(a_2)$
- If $\frac{\partial a_1}{\partial v} \in A_1(a_1)$ and $\frac{\partial a_2}{\partial u} \in A_1(a_2)$, then put $\frac{\partial a_1}{\partial v}$ into $R_1(a_1)$ and put $\frac{\partial a_2}{\partial u}$ into $R_2(a_2)$

3 Mixing spatial and functional derivatives

Let D_α be a first-order spatial differential operator labeled by α. Suppose that g depends on arguments $a_1, a_2, \cdots, a_N$. For now, we make the restriction that g and its arguments don't depend on spatial derivatives of functions; in other words, we consider only first-order spatial derivatives. From the chain rule, we have

$$D_\alpha g = \sum_{\lambda=1}^{N} \frac{\partial g}{\partial a_\lambda} D_\alpha a_\lambda$$

$$= \sum_{\lambda \in Q_1(g)} \frac{\partial g}{\partial a_\lambda} D_\alpha a_\lambda.$$

Now differentiate with respect to a function u,

$$\frac{\partial}{\partial u} D_\alpha g = \sum_{\lambda=1}^{N} \frac{\partial g}{\partial a_\lambda} \frac{\partial}{\partial u} D_\alpha a_\lambda + \sum_{\lambda=1}^{N} \sum_{\mu=1}^{N} \frac{\partial^2 g}{\partial a_\lambda \partial a_\mu} D_\alpha a_\lambda \frac{\partial a_\mu}{\partial u}$$

$$= \sum_{\lambda \in Q_1(g)} \frac{\partial g}{\partial a_\lambda} \frac{\partial}{\partial u} D_\alpha a_\lambda + \sum_{(\lambda,\mu) \in Q_2(g)} \frac{\partial^2 g}{\partial a_\lambda \partial a_\mu} D_\alpha a_\lambda \frac{\partial a_\mu}{\partial u}.$$

We'll also differentiate with respect to $D_\beta u$. By assumption, we know that a_λ and $\frac{\partial g}{\partial a_\lambda}$ don't depend on any spatial derivatives of u, so we find

$$\frac{\partial}{\partial D_\beta u} D_\alpha g = \sum_{\lambda \in Q_1(g)} \frac{\partial g}{\partial a_\lambda} \frac{\partial}{\partial D_\beta u} D_\alpha a_\lambda.$$

3.1 Available sets

If $\frac{\partial g}{\partial u_j} \neq 0$ and

References

[1] J. Engwer, *Partitioned implicit Runge-Kutta timesteppers for micromagnetics with eddy currents*, PhD thesis, 2018.

[2] M. S. Gockenbach, *Understanding and implementing the finite element method*, vol. 97, Siam, 2006.

[3] L. Hand and J. Finch, *Analytical mechanics*, 2000.

[4] T. J. Hughes, *The finite element method: linear static and dynamic finite element analysis*, Courier Corporation, 2012.

[5] L. D. Landau and E. Lifschitz, *Mechanics*, vol. 1, CUP Archive, 1960.

[6] A. Logg, K.-A. Mardal, and G. Wells, *Automated solution of differential equations by the finite element method: The FEniCS book*, vol. 84, Springer Science & Business Media, 2012.

[7] K. Long, *Efficient discretization and differentiation of partial differential equations through automatic functional differentiation*, Automatic Differentiation, 2004 (2004).

[8] K. Long, R. Kirby, and B. van Bloemen Waanders, *Unified embedded parallel finite element computations via software-based fréchet differentiation*, SIAM Journal on Scientific Computing, 32 (2010), pp. 3323–3351.

[9] S. T. Thornton and J. B. Marion, *Classical dynamics of particles and systems*, Cengage Learning, 2021.