ANUGA Kinematic Viscosity Module

Lindon Roberts

November 24, 2010

1 Kinematic Viscosity Equation

The equation we consider is the 2-dimensional parabolic equation given by

$$\frac{\partial u}{\partial t} = (hu_x)_x + (hu_y)_y$$
$$\frac{\partial v}{\partial t} = (hv_x)_x + (hv_y)_y$$

where h is the stage height of the water, and u and v are the x- and y- velocities of the water.

1.1 Elliptic Approximation

Given some discretisation of the 2d domain into triangles $\{T_i\}_{i=1}^n$, and letting ∂T_i be the set of indices j such that triangle T_i and neighbourinng triangle T_j share an edge, we approximate as follows

$$\begin{aligned}\int_{T_i} \text{div}(u_x h, u_y h) dA &= \int_{\partial T_i} (u_x h, u_y h) \cdot \nu dS \\ &\approx \sum_{j \in \partial T_i} \ell_{ij} ((u_x h, u_y h) \cdot \nu) \\ &\approx \sum_{j \in \partial T_i} \ell_{ij} \left(\frac{h_i + h_j}{2}\right) (Du \cdot \nu) \\ &\approx \sum_{j \in \partial T_i} \ell_{ij} \left(\frac{h_i + h_j}{2}\right) \left(\frac{u_i - u_j}{d_{ij}}\right)\end{aligned}$$

where ℓ_{ij} is the length of the edge between T_i and T_j, and d_{ij} is the (straight-line) distance between the centroids of T_i and T_j, and ν is the outward normal.

We now have the option of whether to approximate the divergence at the centroid as the integral of the divergence over the whole triangle, or whether it should be the integral average of the divergence over the triangle, as given by

$$\frac{1}{|T_i|} \int_{T_i} \text{div}(u_x h, u_y h) dA \approx \sum_{j \in \partial T_i} \frac{\ell_{ij}}{|T_i|} \left(\frac{h_i + h_j}{2}\right) \left(\frac{u_i - u_j}{d_{ij}}\right)$$

Which approximation we use is governed by the boolean field `apply_triangle_areas`, which is set in the constructor (default value True). The module has the ability to solve the elliptic equation $\text{div}(u_x h, u_y h) = f$ for arbitrary right-hand side f also. This is implemented in the `cg_solve` method.

1.2 Parabolic Solver

To solve the full parabolic equation, we use Implicit Euler for timestepping. We will see below that the elliptic operator $A[h, u] \approx \text{div}(u_x h, u_y h)$ can be written in the form $A[h, u] = B(h)u + c(h)$ where $B(h)$ is a $n \times n$ matrix and $c(h)$ is a length n vector. Therefore our timestepping is given by

$$\frac{u^{n+1} - u^n}{\Delta t} = A[h^n, u^{n+1}] = B(h^n)u^{n+1} + c(h^n)$$

which rearranges to give

$$(I - \Delta t B(h^n))u^{n+1} = u^n + c(h^n)\Delta t$$

This is the equation we solve in the `parabolic_solve` method.

2 Setting up the matrix problem

Since the elliptic interaction is on all the n interior nodes by the interior nodes *and* the m boundary nodes, we actually get a $n \times (n+m)$ sparse matrix of interactions. Along the ith row, we have four nonzero entries. We define

$$g_{ij} = -\frac{\ell_{ij}}{d_{ij}} = g_i[edge]$$

where $edge = 0, 1, 2$ (the edge opposite the vertex numbered 0, 1 or 2), and T_j for $j \in \{0, \ldots, n+m-1\}$ is the triangle which neighbours T_i along T_i's edge number $edge$. If this edge is a boundary, then $j \geq n$ and we are interested in the boundary point at the midpoint of the edge. We enumerate the boundary edges $(i, edge)$ by first sorting by node i, then by $edge$.

We store the values $g_i[edge]$ in a $n \times 3$ matrix `geo_structure_values` in which the $(i, edge)$ entry is $g_i[edge]$. We also store another $n \times 3$ matrix `geo_structure_indices` in which the $(i, edge)$ entry is j. These matrices are computed during the initialisation process using the C extension module, as they are geometry-dependent only. We use these matrices and the stage heights h to define the elliptic operator $A[h, \cdot]$.

2.1 Applying Stage Heights

We are given a length n vector of stage heights h, from which we need to form the elliptic operator. As stated above, the operator is actually a $n \times (n+m)$ matrix, where we have the off-diagonal entries (in the appropriate places, as given by the matrix `geo_structure_values`, which maps $(i, edge)$ to j)

$$A_{ij} = \left(\frac{h_i + h_j}{2}\right) g_{ij}$$

and diagonal entries

$$A_{ii} = -\sum_{j \in \partial T_i} A_{ij}$$

We of course need to evaluate the stage height on the boundary edges, which we do using the evaluate method associated with each boundary class (the collection of boundary classes is stored in a dictionary with key $(i, edge)$). All the height values (the length n vector h and a length m vector of boundary values) is read into the C extension, along with the geometric structure matrices. The output from this is a tuple $(data, colind, rowptr)$, which are the three arrays corresponding to the CSR Sparse format for matrices (implemented in the ANUGA utilities).

Given that our operator will receive the uh and vh values as inputs, rather than u and v, we also create a matrix `stage_height_scaling` which is a diagonal matrix with entries $1/h[i]$ ($h_i > 0$) or 0 if $h_i = 0$ (in this case, the entire interaction should be zero, so this works). Thus applying this matrix to uh and vh will give us u and v respectively. Thus we construct this matrix during this process.

We now notice that we can only apply this operator matrix to a length $(n+m)$ vector of u or v values. The first n entries of this vector will be our input, but the last m entries are the values of u and v at the boundary, which we can evaluate already. If we write $A = [B|B']$ where B is a $n \times n$ matrix and B' is a $n \times m$ matrix, then the result of multiplyng the full vector of u (or v) by A is the same as $Bu + B'b$ where b is the m-vector of boundary values of u. Therefore we compute the length n vector $c(h) = B'b$, stored as `boundary_vector`, and computed in `build_boundary_vector`. Since we need $c(h)$ for u and v, we store this as a $n \times 2$ matrix. In the solving, we will only use one of these columns depending on whether or not we are solving for u or v. This is achieved by having the `qty_considered` value equal to 1 (u) or 2 (v).

3 Matrix Multiplication and Solving

3.1 Elliptic Multiplication

For the elliptic multiplication, we take in a length n vector (either uh or vh). We first need to change uh to u (or vh to v), which we do by multiplying by the `stage_heights_scaling` matrix (see above). If we are using the integral average formula rather than the integral formula (see the initial equations), we need to scale down each entry by the area of each triangle. We then extend this vector by zeros to make it of length $n+m$ (so we can multiply by A, but have the same effect as multiplying by B only),

and then multiply by A. We have now computed $B(h)u$. In general, if we just want to evaluate $A[h, u]$ (when `include_boundary` is True), then we need to add $c(h)$ to the result, so we do this. We do not want to do this when we are solving (see below), which is why we set it as an argument.

We then scale the result back from u to uh, in order to get consistency in the solver.

3.2 Parabolic Multiplication

This is almost identical to the elliptic case, except we calculate $(I - \Delta t B(h))u$ (note we do not include the boundary). We do not include the boundary term $c(h)$ because this method is only ever called during the parabolic solve. In the implementation, we assume that we get u (not uh) as the input. The scaling is done in the solver wrapper.

3.3 Solving

The solving is done with the ANUGA conjugate gradient solver. In the elliptic case, we read in a length n or $n \times 2$ right-hand side. If it is a vector, then we need to ensure we are considering the right quantity (see above, `qty_considered`). If it is a matrix, then solve for uh and vh using the two columns respectively. This is split up using the various `cg_solve` methods. We then solve the equation

$$B(h)u = f - c(h)$$

for u. This equivalently solves $A[h, u] = f$ for u (or v) and return either a vector of uh values or a matrix of uh and vh values (depending on the input). The elliptic solver is `cg_solve`.

For the parabolic solver, we assume that we are given both $(uh)^n$ and $(vh)^n$ as input. We then scale down to u^n and v^n (given that the most recent call of `apply_stage_heights` would have been for h^n), and solve the equation

$$(I - \Delta t B(h^n))u^{n+1} = u^n + c(h^n)\Delta t$$

Note that the elliptic solver returns uh, but the parabolic solver only returns u^{n+1} and v^{n+1} (the non-scaled values), since we do not have the values of h^{n+1} available.

4 Next Steps

The module is not yet complete. It functions as a stand-alone module, but is not yet fully integrated into the ANUGA code. The primary things that need to be addressed are

- Accessing the value of Δt from the domain input.
- Accessing the boundary values of h, uh and vh from the domain class. This may need to involve some fixing of the enumeration of the boundary sides used in the body of the code.
- Moving from accepting as input stage heights to just heights (stage heights + elevation).
- Simplify the code to take in a matrix with the h, uh and vh values in one go, apply stage heights and then perform a parabolic solve.
- Including the module as part of the evolve process for the ANUGA program.
- Setting up the compiler for Linux (all testing and development done on Windows).

5 More Information

For some more information, including sample usage, go to the Readme, or the test file.