1 Deriving SPH

The Navier-Stokes equations with gravity are

$$\rho \mathbf{a} = -\nabla p + \mu \nabla^2 \mathbf{v} + \rho \mathbf{g}.$$

The acceleration is the material derivative of velocity, and we usually take an Eulerian perspective and write this as

$$\mathbf{a} = \frac{D\mathbf{v}}{Dt} = \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}.$$

In smoothed particle hydrodynamics, though, we take a Lagrangian perspective, and actually associate computational particles with material points. This makes it easy to deal with the left-hand side of the Navier-Stokes equation.

To compute the spatial derivatives on the right hand side of the equation, we interpolate pressures and velocities at the material particles to get smoothed fields (hence the name). Then we differentiate the smoothed fields. For example, suppose we care about some scalar field $A(\mathbf{r})$. Each particle j has a mass m_j, a location $\mathbf{r}_j$, and a value $A_j = A(\mathbf{r}_j)$. Between particles, we write

$$A_S(\mathbf{r}) = \sum_j m_j \frac{A_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h), \tag{1}$$

where W is a smoothing kernel with radius h. The densities ρ_j that appear in (1) are themselves are computed using (1):

$$\rho_i = \rho_S(\mathbf{r}_i) = \sum_j m_j \frac{\rho_j}{\rho_j} W(\mathbf{r} - \mathbf{r}_j, h), = \sum_j m_j W(\mathbf{r} - \mathbf{r}_j, h).$$

Putting everything together, the SPH approximation computes field quantities at locations associated with computational particles. The governing equations for the particles and the associated quantities are then

$$\rho_i \mathbf{a}_i = \mathbf{f}_i^{\text{pressure}} + \mathbf{f}_i^{\text{viscosity}} + \rho_i \mathbf{g} \tag{2}$$

$$\mathbf{f}_i^{\text{pressure}} = -\sum_j m_j \frac{p_i + p_j}{2\rho_j} \nabla W(\mathbf{r}_i - \mathbf{r}_j, h) \tag{3}$$

$$\mathbf{f}_i^{\text{viscosity}} = \mu \sum_j m_j \frac{\mathbf{v}_j - \mathbf{v}_i}{\rho_j} \nabla^2 W(\mathbf{r}_i - \mathbf{r}_j, h), \tag{4}$$

where the pressure and viscous interaction terms have been symmetrized to ensure that particle i acts on particle j in the same way j acts on i.

To compute the pressure, we use the ideal gas equation of state

$$p_i = k(\rho_i - \rho_0). \tag{5}$$

Of course, this is not the right equation of state for a liquid! This equation is best regarded as a non-physical approximation that is legitimate as long as the artificial speed of sound is much greater than the velocities of interest in the problem (as is the case with the incompressible approximation that is more commonly used in other settings).

2 Smoothing kernels

Müller describes three radially symmetric different kernels for 3D simulation, each with the form

$$W(\mathbf{r}, h) = \frac{1}{Ch^d} \begin{cases} f(q), & 0 \leq q \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

where $q = \|bfr\|/h$ and $d = 3$ is the dimension. The kernels are based on the choices $f_{\text{poly6}}(q) = (1 - q^2)^3$ for general interpolation, $f_{\text{spiky}}(q) = (1 - q)^3$ for interpolating pressures, and $f_{\text{viscosity}}(q)$ for viscosity computations. The gradients are given by

$$\nabla W(\mathbf{r}, h) = \frac{\mathbf{r}}{Ch^{d+2}} \begin{cases} q^{-1} f'(q), & 0 \leq q \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

and the Laplacians are

$$\nabla^2 W(\mathbf{r}, h) = \frac{1}{Ch^{d+2}} \begin{cases} f''(q) + (d-1)q^{-1} f'(q), & 0 \leq q \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

The pressure kernel is designed with relatively steep gradients close to the origin to prevent the clustering of computational particles that occurs when pressures are interpolated with W_{poly6}. The viscosity kernel is designed so that the Laplacian will be positive definite, ensuring that we don't accidentally get negative viscous contributions that add energy to the system (and compromise stability). Specifically, the viscosity kernel is chosen so that the Laplacian will be proportional to $(1-q)$ (so that the kernel Laplacian will always be positive) and the kernel and its gradient vanish at $q = 1$. Because the Laplacian is different in two and three dimensions, these conditions lead to different functions $f_{\text{viscosity}}(q)$ in two and three dimensions. In two dimensions, the correct choice is

$$f_{\text{viscosity}}(q) = \frac{q^2}{4} - \frac{q^3}{9} - \frac{1}{6}\log(q) - \frac{5}{36}.$$

The normalizing constants in two dimensions are determined by

$$C = \int_{0^1} 2\pi q f(q)\, dq.$$

For our three functions, these constants are

$$\begin{aligned} C_{\text{poly6}} &= \pi/4 \\ C_{\text{spiky}} &= \pi/10 \\ C_{\text{viscosity}} &= \pi/40. \end{aligned}$$

3 Condensed interaction force expressions

Making things completely explicit for the cases we care about most, we have (for $0 \leq q \leq 1$)

$$W_{\text{poly6}}(\mathbf{r}, h) = \frac{4}{\pi h^8}(h^2 - r^2)^3 \tag{6}$$

$$\nabla W_{\text{poly6}}(\mathbf{r}, h) = -\frac{24\mathbf{r}}{\pi h^8}(h^2 - r^2)^2 \tag{7}$$

$$\nabla^2 W_{\text{poly6}}(\mathbf{r}, h) = -\frac{48}{\pi h^8}(h^2 - r^2)(h^2 - 3r^2) \tag{8}$$

$$\nabla W_{\text{spiky}}(\mathbf{r}, h) = -\frac{30}{\pi h^4}\frac{(1-q)^2}{q}\mathbf{r} \tag{9}$$

$$\nabla^2 W_{\text{viscosity}}(\mathbf{r}, h) = \frac{40}{\pi h^4}(1-q) \tag{10}$$

If we substitute (9), (10), and the equation of state (5) into (3) and (4), we have

$$\mathbf{f}_i^{\text{pressure}} = \frac{15k}{\pi h^4}\sum_{j \in N_i} m_j \frac{\rho_i + \rho_j - 2\rho_0}{\rho_j}\frac{(1-q_{ij})^2}{q_{ij}}\mathbf{r}_{ij}$$

$$\mathbf{f}_i^{\text{viscosity}} = \frac{40\mu}{\pi h^4}\sum_{j \in N_i} m_j \frac{\mathbf{v}_i - \mathbf{v}_j}{\rho_j}(1 - q_{ij})$$

where N_i is the set of particles within h of particle i and $q_{ij} = \|\mathbf{r}_{ij}\|/h$, $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$. Putting these terms together, we have

$$\mathbf{f}_i^{\text{pressure}} + \mathbf{f}_i^{\text{viscosity}} = \sum_{j \in N_i} \mathbf{f}_{ij}^{\text{interact}}$$

where

$$\mathbf{f}_{ij}^{\text{interact}} = \frac{m_j}{\pi h^4 \rho_j}(1 - q_{ij})\left[15k(\rho_i + \rho_j - 2\rho_0)\frac{(1-q_{ij})}{q_{ij}}\mathbf{r}_{ij} - 40\mu\mathbf{v}_{ij}\right],$$

and $\mathbf{v}_{ij} = \mathbf{v}_i - \mathbf{v}_j$. We then rewrite (2) as

$$\mathbf{a}_i = \frac{1}{\rho_i}\sum_{j \in N_i} \mathbf{f}_{ij}^{\text{interact}} + \mathbf{g}.$$

4 Sanity checks

I fairly regularly make typographical and copying errors when I do algebra and implement it in code. In order to stay sane when I actually write something somewhat complicated, I find it helpful to put together little test scripts to check my work numerically. For your edification, in this section I give my MATLAB test script corresponding to the derivation in these notes. The test script is done in MATLAB.

I begin by implementing the functions $f(q)$, the normalizing constants, and the kernel functions for each of the three kernels.

```
fp6 = @(q) (1-q^2)^3;
fsp = @(q) (1-q)^3;
fvi = @(q) q^2/4 - q^3/9 - log(q)/6 - 5/36;
Cp6 = pi/4;
Csp = pi/10;
Cvi = pi/40;
Wp6 = @(r,h) 1/Cp6/h/h * fp6( norm(r)/h );
Wsp = @(r,h) 1/Csp/h/h * fsp( norm(r)/h );
Wvi = @(r,h) 1/Cvi/h/h * fvi( norm(r)/h );
```

I computed the normalization constants analytically, but I'm prone to algebra mistakes when I compute integrals by hand. Let's check against MATLAB's `quad` function.

```
fprintf('Relerr for normalization constants:\n');
nerr_p6 = quad( @(q) 2*pi*q*fp6(q)/Cp6, 0,     1 ) - 1;
nerr_sp = quad( @(q) 2*pi*q*fsp(q)/Csp, 0,     1 ) - 1;
nerr_vi = quad( @(q) 2*pi*q*fvi(q)/Cvi, 1e-12, 1 ) - 1;
fprintf('  Cp6: %g\n', nerr_p6);
fprintf('  Csp: %g\n', nerr_sp);
fprintf('  Cvi: %g\n', nerr_vi);
```

Now check that I did the calculus right for the gradient and Laplacian of the W_{poly6} kernel and for the gradient of the pressure kernel.

```
h = rand(1);
r = rand(2,1)*h/4;
q = norm(r)/h;
r2 = r'*r;
h2 = h^2;
dr = norm(r)*1e-4;
gWp6_fd = fd_grad(@(r) Wp6(r,h), r, dr);
gWp6_ex = -24/pi/h^8*(h2-r2)^2*r;
gWsp_fd = fd_grad(@(r) Wsp(r,h),r,dr);
gWsp_ex = -(30/pi)/h^4 * (1-q)^2 / q * r;
lWp6_fd = fd_laplace(@(r) Wp6(r,h), r, dr);
lWp6_ex = -48/pi/h^8*(h2-r2)*(h2-3*r2);
fprintf('Check Wp6 kernel derivatives:\n');
fprintf('  grad Wp6:  %g\n', norm(gWp6_fd-gWp6_ex)/norm(gWp6_ex));
fprintf('  grad Wsp:  %g\n', norm(gWsp_fd-gWsp_ex)/norm(gWsp_ex));
fprintf('  laplacian: %g\n', (lWp6_fd-lWp6_ex)/lWp6_ex);
```

Now check that $f_{\mathrm{viscosity}}(q)$ satisfies the conditions that are supposed to define it:

$$f(1) = 0$$
$$f'(1) = 0$$
$$f''(q) + \frac{1}{q}f'(q) = 1 - q, \quad 0 < q < 1$$

The first two conditions we check directly; the last we check at a randomly chosen q.

```
q    = rand(1);
dq   = 1e-4*q;
fq   = fvi(q);
lffd = fd_laplace_radial(fvi,q,dq);
fprintf('Relerr for viscosity kernel checks:\n');
fprintf('  fvi (1): %g\n', fvi(1) );
fprintf('  dfvi(1): %g\n', fd_deriv(fvi,1,dq) );
fprintf('  Laplace: %g\n', fd_laplace_radial(fvi,q,dq)/(1-q)-1);
```

Now, let me check that I did the algebra right in getting the condensed formula for the interaction forces.

```
% Set up random parameter choices
r_ij = rand(2,1);
v_ij = rand(2,1);
k    = rand(1);
rho0 = rand(1);
rhoi = rand(1);
rhoj = rand(1);
mass = rand(1);
mu   = rand(1);
q    = norm(r_ij)/h;

% Compute pressures via equation of state
Pi = k*(rhoi-rho0);
Pj = k*(rhoj-rho0);

% Differentiate the kernels
Wsp_x  = -30/pi/h^4*(1-q)^2/q*r_ij;
LWvi   = 40/pi/h^4*(1-q);

% Do the straightforward computation
fpressure  = -mass*(Pi+Pj)/2/rhoj * Wsp_x;
fviscous   = -mu*mass*v_ij/rhoj * LWvi;
```

```
finteract1 = fpressure + fviscous;

% Do the computation based on my condensed formula
finteract2 = mass/pi/h^4/rhoj * (1-q) * ...
    ( 15*k*(rhoi+rhoj-2*rho0)*(1-q)/q * r_ij - ...
      40*mu * v_ij );

% Compare
fprintf('Relerr in interaction force check:\n');
fprintf('  fint:  %g\n', norm(finteract1-finteract2)/norm(finteract1));
```

Of course, all the above is supported by a number of little second-order accurate finite difference calculations.

```
function fp   = fd_deriv(f,r,h)
  fp = ( f(r+h)-f(r-h) )/2/h;

function fpp = fd_deriv2(f,r,h)
  fpp = ( f(r+h)-2*f(r)+f(r-h) )/h/h;

function del2f = fd_laplace_radial(f,r,h)
  del2f = fd_deriv2(f,r,h) + fd_deriv(f,r,h)/r;

function del2f = fd_laplace(f,r,h)
  e1 = [1; 0];
  e2 = [0; 1];
  del2f = (-4*f(r)+f(r+h*e1)+f(r+h*e2)+f(r-h*e1)+f(r-h*e2) )/h/h;

function gradf = fd_grad(f,r,h)
  e1 = [1; 0];
  e2 = [0; 1];
  gradf = [f(r+h*e1)-f(r-h*e1);
           f(r+h*e2)-f(r-h*e2)] / 2 / h;
```