*Research Programming Technical Social 1*

# Numerical approximation: friend or foe?

Ulf D. Schiller

Centre for Computational Science, University College London

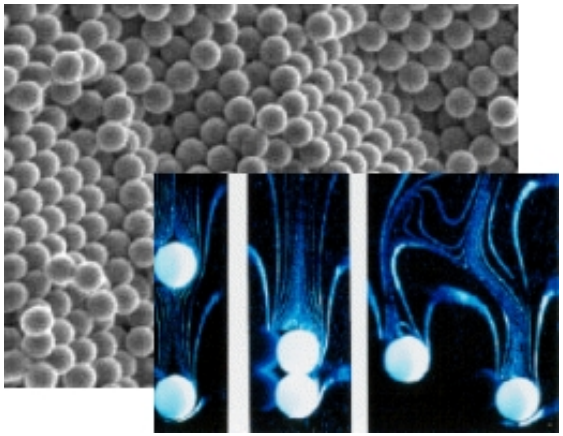
u.schiller@ucl.ac.uk

September 9th, 2015

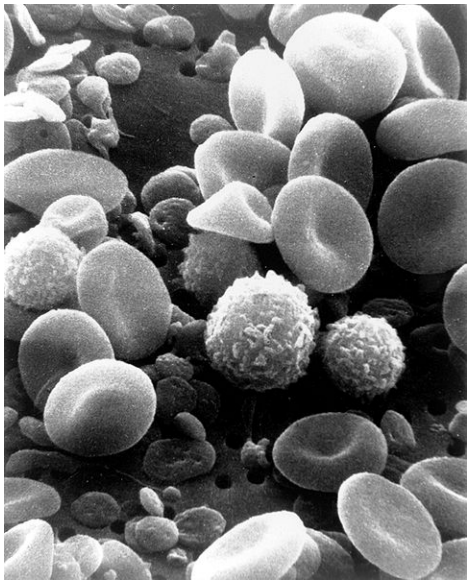
# Computational fluid dynamics: Flowing matter

Source: Wikipedia, GFDL

Source: Universiteit Utrecht



Source: Emory University



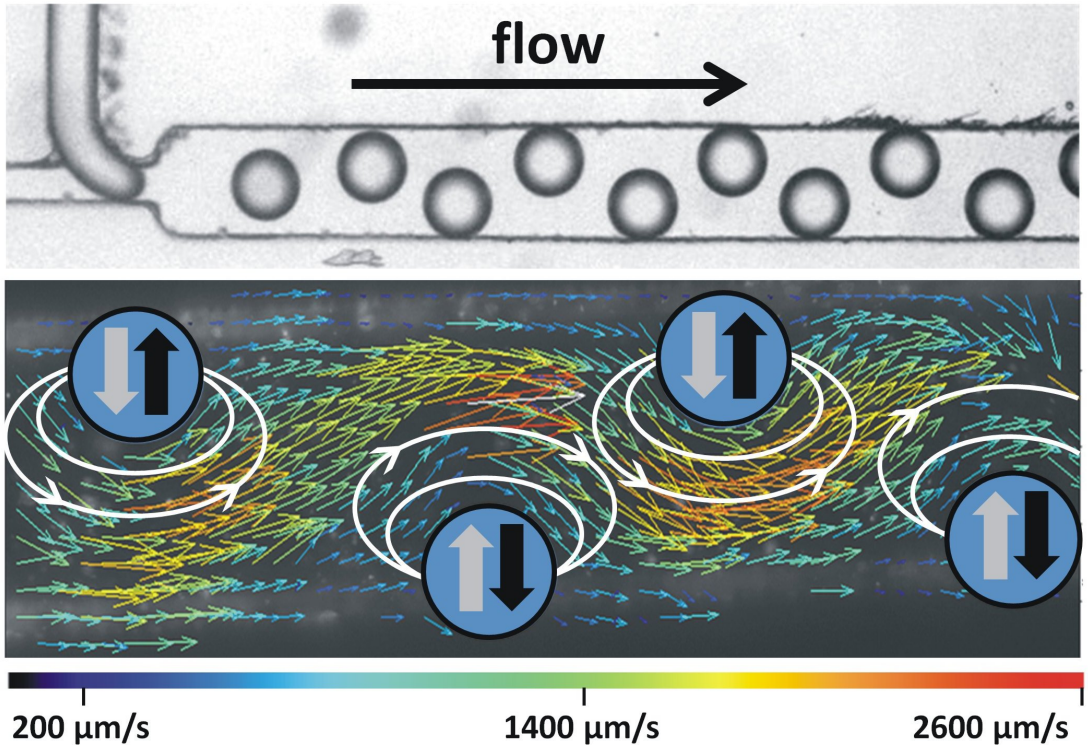
Source: Wikimedia

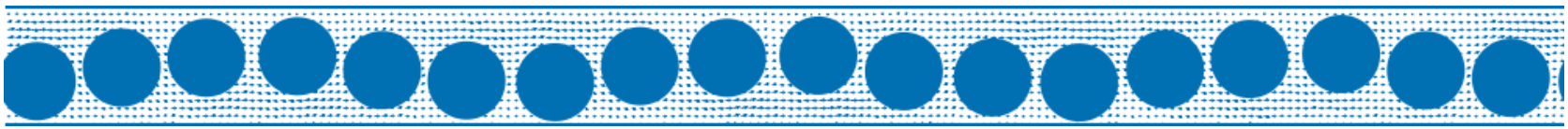
- Solutions, suspensions, emulsions: “contain” multiple length scales

$\rightarrow$ Motion of the solutes and flow of the solvent are both important

# Example: Microfluidic droplets

[Experiments by J.-B. Fleury / Seemann group]



[J.-B. Fleury, UDS, et al., *New J. Phys.* **16** 063029 (2014)]

# HemeLB: Simulation of large vascular networks

# Fluid dynamics: The Navier-Stokes equations

- Continuity equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

- Navier-Stokes equation

$$\frac{\partial (\rho \mathbf{u})}{\partial t} + \nabla \cdot \Pi = \rho \mathbf{f}$$

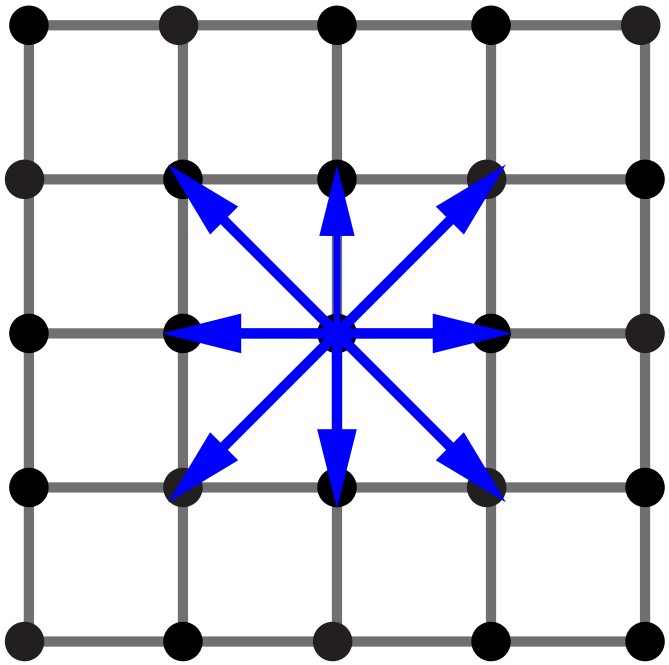
- Stress tensor

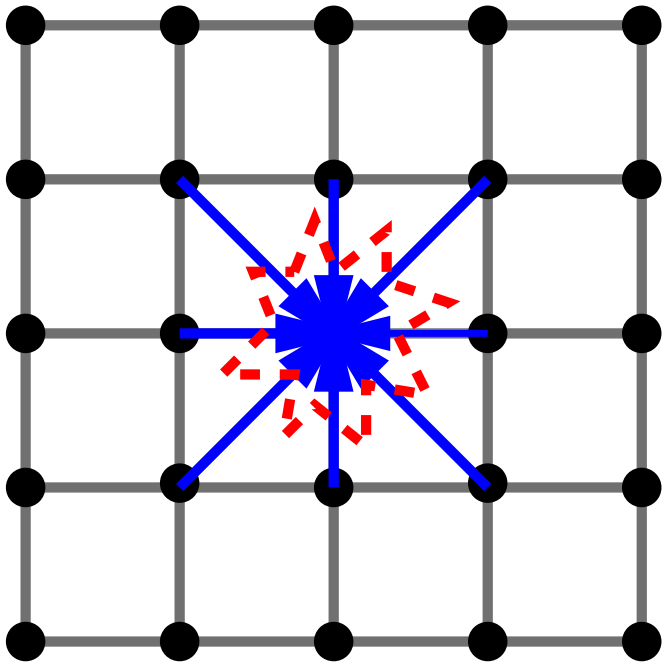
$$\Pi = \underbrace{\overbrace{\rho c_s^2}^{p} \mathrm{I} + \frac{\mathbf{j} \otimes \mathbf{j}}{\rho}}_{\sigma^{\mathrm{eq}}} + \underbrace{\eta : \left( \nabla \otimes \frac{\mathbf{j}}{\rho} \right)}_{\sigma^{\mathrm{visc}}} + \sigma^{\mathrm{fluct}}$$

- nonlinear partial differential equation

# Lattice Boltzmann

Historic origin: lattice gas automaton





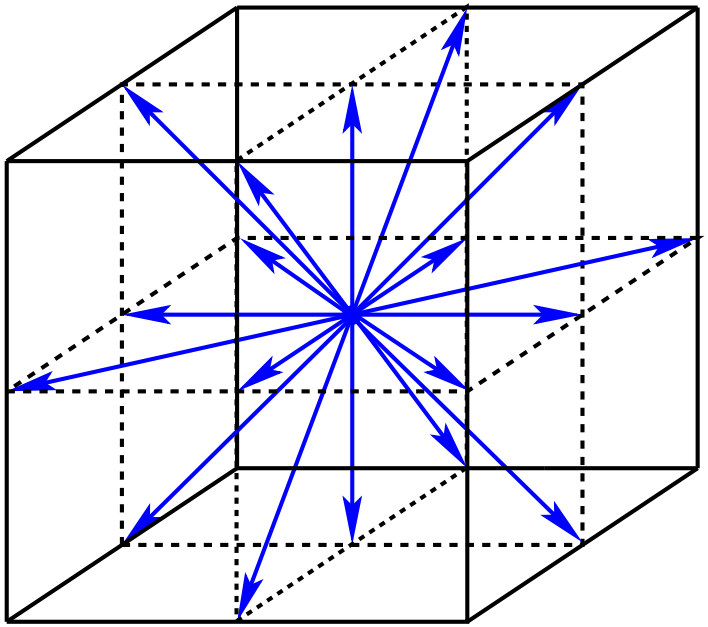
# The classical LB algorithm

1. streaming step: move $f_i^*(\mathbf{r}, t)$ along $\mathbf{c}_i$ to the next lattice site, increment $t$ by $h$

$$f_i(\mathbf{r} + h\mathbf{c}_i, t + h) = f_i^*(\mathbf{r}, t)$$

2. collision step: apply $\Lambda_{ij}$ and compute the post-collisional $f_i^*(\mathbf{r}, t)$ on every lattice site

$$f_i^*(\mathbf{r}, t) = f(\mathbf{r}, t) - \sum_j \Lambda_{ij} \left[ f_j(\mathbf{r}, t) - f_j^{\mathrm{eq}}(\rho, \mathbf{u}) \right]$$



D3Q19 lattice

# The D3Q19 model

Equilibrium distribution:

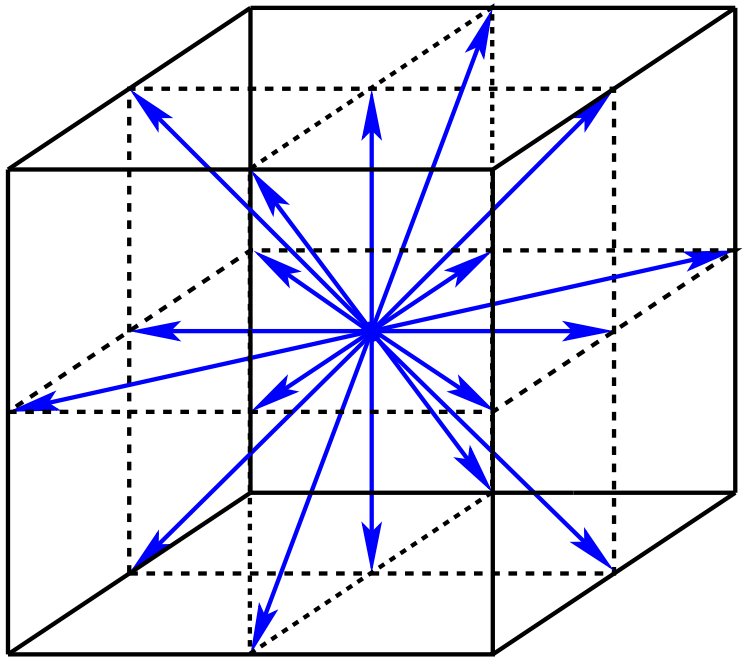
$$f_i^{\mathrm{eq}}(\rho, \mathbf{u}) = w_i \rho \left[ 1 + \frac{\mathbf{u} \cdot \mathbf{c}_i}{c_s^2} + \frac{\mathbf{uu} : (\mathbf{c}_i \mathbf{c}_i - c_s^2 \mathsf{I})}{2 c_s^4} \right]$$

Moments:

$$\sum_i f_i^{\mathrm{eq}} = \rho$$

$$\sum_i f_i^{\mathrm{eq}} \mathbf{c}_i = \rho \mathbf{u}$$

$$\sum_i f_i^{\mathrm{eq}} \mathbf{c}_i \mathbf{c}_i = \rho c_s^2 \mathsf{I} + \rho \mathbf{uu}$$



# Floating-point accuracy

- distribution

$$f_i = f_i^{\mathrm{eq}} + f_i^{\mathrm{neq}} = w_i \rho \left[ 1 + \frac{\mathbf{u} \cdot \mathbf{c}_i}{c_s^2} + \frac{\mathbf{uu} : (\mathbf{c}_i \mathbf{c}_i - c_s^2 \mathsf{I})}{2 c_s^4} \right] + \mathit{small}$$

- incompressible fluid: $\rho \approx \text{const.} = \rho_0$

$$f_i = w_i \rho_0 + \mathit{small}$$

- to avoid mixing large and small terms: store only the deviations

$$\tilde{f}_i = f_i - w_i \rho_0$$

- for single precision this can be crucial

# Some code examples

Structs (a.k.a. poor man's object orientation)

```
typedef const struct _LBmodel {
  /* dimensionality */
  const int n_dim;
  /* number of velocities */
  const int n_vel;
  /* velocity vectors */
  const double (*c)[NDIM];
  /* lattice weights */
  const void (*weights)(double *w, double cs2);
} LB_Model;
```

# Some code examples

## Precompiler pragmas

```
#define VELS(D,Q)     d##D##q##Q##_velocities
#define WEIGHTS(D,Q) d##D##q##Q##_weights
/* these macros are necessary to force prescan below */
/* because prescan does not occur for stringify and concat */
#define DnQm(D,Q,M)  { D, Q, VELS(D,Q), WEIGHTS(D,Q) }
```

## Changing the LB model becomes a one-liner

```
static const LB_Model lbmodel = DnQm(NDIM,NVEL);
```

# Data layout

## Collision optimized layout

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| (0,0,0) *i=0* | (0,0,0) *i=1* | (0,0,0) *i=2* | ... | (0,0,0) *i=18* | (0,0,1) *i=0* | (0,0,1) *i=1* | (0,0,1) *i=2* | ... | (0,0,1) *i=18* | (0,0,2) *i=0* | ... |
| (0,1,0) *i=0* | (0,1,0) *i=1* | (0,1,0) *i=2* | ... | (0,1,0) *i=18* | (0,1,1) *i=0* | (0,1,1) *i=1* | (0,1,1) *i=2* | ... | (0,1,1) *i=18* | (0,1,2) *i=0* | ... |
| ... | | | | | | | | | | | |
| (1,0,0) *i=0* | (1,0,0) *i=1* | (1,0,0) *i=2* | ... | (1,0,0) *i=18* | (1,0,1) *i=0* | (1,0,1) *i=1* | (1,0,1) *i=2* | ... | (1,0,1) *i=18* | (1,0,2) *i=0* | ... |
| ... | | | | | | | | | | | |

## Propagation optimized layout

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *i=0* (0,0,0) | *i=0* (0,0,1) | *i=0* (0,0,2) | ... | *i=0* (0,1,0) | *i=0* (0,1,1) | *i=0* (0,1,2) | ... | ... | *i=0* (1,0,0) | *i=0* (1,0,1) | *i=0* (1,0,2) | ... |
| *i=1* (0,0,0) | *i=1* (0,0,1) | *i=1* (0,0,2) | ... | *i=1* (0,1,0) | *i=1* (0,1,1) | *i=1* (0,1,2) | ... | ... | *i=1* (1,0,0) | *i=1* (1,0,1) | *i=1* (1,0,2) | ... |
| *i=2* (0,0,0) | *i=2* (0,0,1) | *i=2* (0,0,2) | ... | *i=2* (0,1,0) | *i=2* (0,1,1) | *i=2* (0,1,2) | ... | ... | *i=2* (1,0,0) | *i=2* (1,0,1) | *i=2* (1,0,2) | ... |
| ... | | | | | | | | | | | | |

# Some code examples

## Cache-efficient memory layout

```
/* temporary/secondary grid */
#define PFI *FI(NVEL,WGRID)
#define FI(i,x) fi[i][x]

static double *lbf = NULL;
static double PFI;
```

# Some code examples

```
static void lb_stream(double *f, double PFI, int x, int y) {

  int xc, xp, xm, xp2, xm2, xp4, xm4;
  xc = x%WGRID;
  xp  = (x+1)%WGRID; xm  = (x-1+WGRID)%WGRID;
  xp2 = (x+2)%WGRID; xm2 = (x-2+WGRID)%WGRID;
  xp4 = (x+4)%WGRID; xm4 = (x-4+WGRID)%WGRID;

  FI( 0, xc )[y]   = f[0];
  FI( 1, xp )[y]   = f[1];
  FI( 2, xm )[y]   = f[2];
  FI( 3, xc )[y+1] = f[3];
  FI( 4, xc )[y-1] = f[4];
  ...

}
```

https://gist.github.com/uschille/8f65dd40572b2d943409

# The lattice Boltzmann equation

- continuous Boltzmann equation

$$\left(\frac{\partial}{\partial t} + \mathbf{v}\cdot\frac{\partial}{\partial \mathbf{r}}\right) f(\mathbf{r},\mathbf{v},t) = -\Omega\left[f(\mathbf{r},\mathbf{v},t) - f^{\mathrm{eq}}(\mathbf{v})\right]$$

- discrete velocity model

$$\left(\frac{\partial}{\partial t} + \mathbf{c}_i\cdot\frac{\partial}{\partial \mathbf{r}}\right) f_i = -\sum_j \Omega_{ij}(f_j - f_j^{\mathrm{eq}})$$

- systematic discretization $\rightarrow$ lattice Boltzmann equation

$$\bar{f}_i(\mathbf{r}+h\mathbf{c}_i, t+h) - \bar{f}_i(\mathbf{r},t) = -\sum_j \Lambda_{ij}\left[\bar{f}_j(\mathbf{r},t) - f_j^{\mathrm{eq}}(\rho,\mathbf{u})\right]$$

- Caution: $\bar{f}_i$ are not the discrete $f_i$!

# Splitting of the discrete Boltzmann equation

- Discrete velocity model (now with force term)

$$\frac{\partial}{\partial t} f_i = -\mathbf{c}_i \cdot \frac{\partial}{\partial \mathbf{r}} f_i - \sum_j \Omega_{ij}(f_j - f_j^{\mathrm{eq}}) + G_i = (\mathcal{S} + \mathcal{C} + \mathcal{F})\, f_i$$

- Formal solution $\quad f_i(t+h) = \exp\left[h(\mathcal{S} + \mathcal{C} + \mathcal{F})\right]\ f_i(t)$
- Baker-Campbell-Hausdorff (for sufficiently differentiable operators)

$$\exp\left[h(\mathcal{S} + \mathcal{C} + \mathcal{F})\right] = e^{\frac{h}{2}(\mathcal{C}+\mathcal{F})} e^{h\mathcal{S}} e^{\frac{h}{2}(\mathcal{C}+\mathcal{F})} + \mathcal{O}(h^3) \approx \mathsf{C}^{\frac{1}{2}}\mathsf{S}\mathsf{C}^{\frac{1}{2}}$$

- Second-order in time lattice Boltzmann update

$$\mathbf{f}(t+h) = \mathsf{C}^{\frac{1}{2}}\mathsf{S}\mathsf{C}^{\frac{1}{2}}\,\mathbf{f}(t)$$

[P. Dellar, *Comp. Math. App.* **65** 129-141 (2013)]
[UDS, *Comp. Phys. Comm.* **185** 2586-2597 (2014)]

# Transformation to "real" quantities (mind the gap...)

- lattice Boltzmann equation

$$\bar{f}_i(\mathbf{r}+h\mathbf{c}_i, t+h) - \bar{f}_i(\mathbf{r}, t) = -\sum \Lambda_{ij}\left[\bar{f}_j(\mathbf{r}, t) - f_j^{\mathrm{eq}}(\rho, \mathbf{u})\right] + \sum (\delta_{ij} - \frac{1}{2}\Lambda_{ij}) g_j$$

- transformation

$$f_i = \bar{f}_i - \frac{1}{2}\sum \Lambda_{ij}\left(\bar{f}_j - f_j^{\mathrm{eq}}\right) + \frac{1}{2}\sum \left(\delta_{ij} - \frac{1}{2}\Lambda_{ij}\right) g_j = \frac{1}{2}(\bar{f}_i + \bar{f}_i^{*})$$

- macroscopic variables

$$\rho = \sum f_i = \sum \bar{f}_i$$

$$\rho\mathbf{u} = \sum f_i \mathbf{c}_i = \sum \bar{f}_i \mathbf{c}_i + \frac{h}{2}\mathbf{g}$$

$$\Pi^{\mathrm{neq}} = \sum (f_i - f_i^{\mathrm{eq}})\mathbf{c}_i\mathbf{c}_i = \frac{1}{2}\left(\bar{\Pi}^{\mathrm{neq}} + \bar{\Pi}^{\mathrm{neq},*}\right)$$

# Diffusive scaling (our foe?)

- lattice Boltzmann is second-order accurate in space *and* time
- grid spacing $a$, time step $h$

$$\text{speed of sound:} \quad c_s = \hat{c}_s \frac{a}{h}$$

$$\text{viscosity:} \quad \nu = \hat{c}_s^2 \left( \hat{\tau} - \frac{1}{2} \right) \frac{a^2}{h}$$

- linear stability requires $\hat{\tau} > 0.5$

$$\hat{\tau} = \frac{\nu}{\hat{c}_s^2} \frac{h}{a^2} + \frac{1}{2}$$

- grid refinement: $a$ smaller $\rightarrow$ $c_s$ smaller (not good...)

# Diffusive scaling (our foe?)

- Reynolds number, Mach number

$$Re = \frac{uL}{\nu} \qquad Ma = \frac{u}{c_s}$$

- diffusive scaling: $a \sim \epsilon L \quad h \sim \epsilon^2 T$

$$\frac{a^2}{h} = \text{const.} \qquad c \sim \frac{1}{\epsilon}\frac{L}{T} \underset{\epsilon \to 0}{\to} \infty$$

  $Ma \to 0$ at fixed Reynolds number

$\to$ This can be computationally expensive!

- N.b.: Diffusive scaling prevents us from simulating Knudsen effects!

# Cancellation of errors (our friend)

- Collisions

$$\frac{\partial}{\partial t} f_i = -\sum_j \Omega_{ij} (f_j - f_j^{\mathrm{eq}})$$

- Discrete collisions (Crank-Nicolson rule)

$$f_i^* = f_i - \sum_j \left[ \left( \mathsf{I} + \frac{h}{2}\Omega \right)^{-1} h\Omega \right]_{ij} \left( f_j - f_j^{\mathrm{eq}} \right) + \mathcal{O}((h/\tau)^3)$$

- The error $\mathcal{O}((h/\tau)^3)$ can cause non-linear instabilities!
- Why not exact solution?

$$f_i^* = f_i + \sum_j \left[ \exp(-h\Omega) - \mathsf{I} \right]_{ij} \left( f_j - f_j^{\mathrm{eq}} \right)$$

[Brownlee et al., *Phys. Rev. E* **75** 036711 (2007)]

[P. Dellar, *Comp. Math. App.* **65** 129-141 (2013)]

# Cancellation of errors (our friend)

- consider the behaviour of sinusoidal shear waves $\propto \exp(ikx)$
- omit nonlinear terms ($\mathbf{u} \cdot \nabla \mathbf{u} = 0$)

$$f_i^{\mathrm{eq}} = w_i \rho \left(1 + \frac{\mathbf{u} \cdot \mathbf{c}_i}{c_s^2}\right) = w_i \rho + \rho n_i$$

- lattice Boltzmann scheme becomes

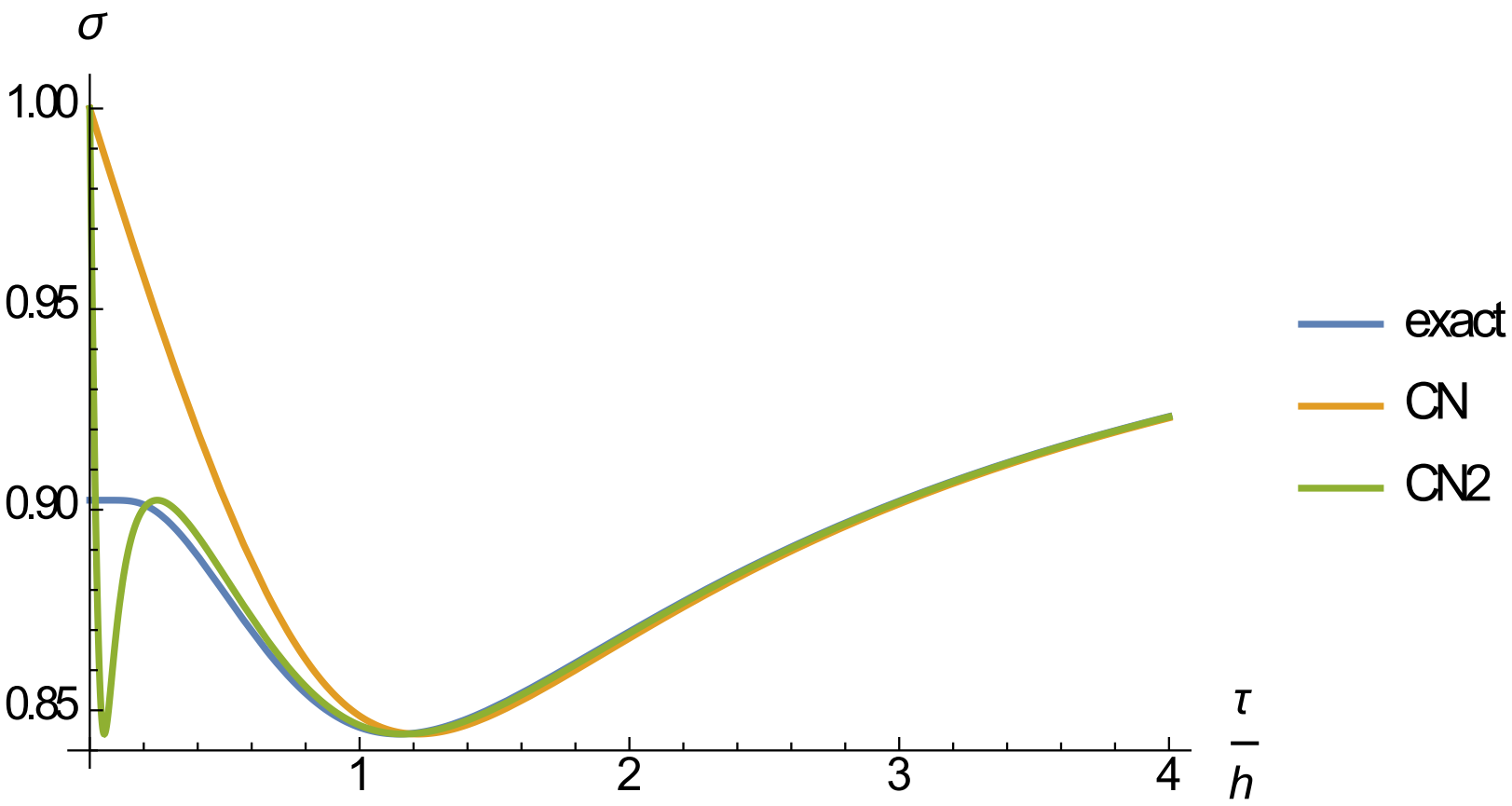
$$\sigma \exp\left(ikc_{ix}h\right) n_i = n_i - \frac{1}{\tau}\left(n_i - 3 w_i c_{iy} \sum c_{jy} n_j\right)$$

$\rightarrow$ matrix eigenvalue problem

$$\sigma = \exp\left(-c_s^2 k^2 \frac{\tau}{h}\right) \text{ as } k, \tau \rightarrow 0$$

[P. Dellar, *Comp. Math. App.* **65** 129-141 (2013)]

# Cancellation of errors (our friend)




$\rightarrow$ Navier-Stokes behaviour still recovered for $\tau/h \rightarrow 0$!

# Flying ice cube

- example of a numerical artefact in MD simulations
- constant temperature thermostat: velocity rescaling

$$v_i' = \sqrt{\frac{N k_B T}{\sum m_j v_j^2}} v_i$$

- energy is drained from high-frequency into low-frequency modes

$\rightarrow$ high linear momentum, no internal motion (unphysical)

[S. C. Harvey, R. K.-Z. Tan, and T. E. Cheatham III. *J. Comp. Chem.* **19**, 726-740 (1998)]

# Hot solvent/Cold solute

- another thermostat artefact
- truncation error from finite simulation time step
- algorithmic noise couples stronger to (light) solvent particles than to (heavy) solute particles

$\rightarrow$ solvent heats up, solute cools down (overall system temperature ok)

- N.b.: Can be rectified by using two thermostats, but that sometimes introduces large artefacts into the conformational dynamics.

[M. Lingenheil *et al.*. *J. Chem. Theory Comput.* **4**, 1293–1306 (2008)]

# Closing remarks

> *"But, as with education in general, simulation must be kept honest, because seeing is believing, and animated displays can be very convincing irrespective of their veracity."*
>
> D. C. RAPAPORT, THE ART OF MOLECULAR DYNAMICS SIMULATION

1. Conjecture: A bug in the code is always more likely than discovery of new physics.
2. Stipulation: Get the right answers for the right reasons!

*Nobody cares how fast you can compute the wrong answer.*

→ *Discussion session*

# How to validate and test (novel) simulation results?

## [ESPResSo-users] P3M broken in current develo

**From:**
**Subject:** [ESPResSo-users] P3M broken in current development code
**Date:** Mon, 04 Apr 2011 12:24:05 +0200
**User-agent:** Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.2.14) Gecko/201

Hello everybody!

At the end of last week we have noticed that the P3M code within the
*development* code of ESPResSo has been broken since 1st December 2010.
Unfortunately, our standard test code didn't find this, as it turned out
that the test itself is problematic.

# How to validate and test (novel) simulation results?

- We're looking for novel science – but how do we test it?
- What approaches are being used for validation?
- Standard protocols for (scientific) testing of
  - convergence?
  - equilibrium?
  - steady-state?
  - finite-size effects?
- Can this be automated?
- How to test the test? (To infinity and beyond?)