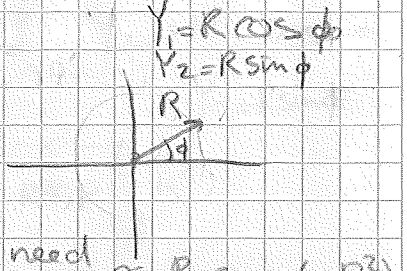
Intermezzo: drawing Gaussian random numbers
(need for prog 1.5, also topic of handin 3)
Random number generation - random int
- random float/double etc $[0,1)$

How to get another distribution?

Example: Gaussian (going to make 2 at once)

- draw 2 random $X_1, X_2 \in [0,1)$
- we want $\propto \exp(-Y_1^2 - Y_2^2)\, dY_1\, dY_2$
- go through $\phi = 2\pi X_1$ $R = f(X_2)$ ← we need



$$p_X(X)\,dX_1 = p_R(R)\,dR$$
$$= 1 \left|\frac{dX}{dR}\right| dR \quad \Rightarrow \quad \left|\frac{dX}{dR}\right| = 2R\exp(-R^2)$$
$$\propto R\exp(-R^2)$$

$$\Rightarrow X(R) = \exp(-R^2)$$
$$\Rightarrow R = [-\log X_2]^{1/2}$$

Resource scaling.

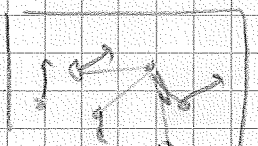
Look again at many-particle system (like Argon)

variables $r_1, r_2, \ldots, r_N$ } $O(N)$ variables
$v_1, v_2, \ldots, v_N$

memory: must store all this $O(N)$

cpu: - must update all this while running
- every particle interacts with every other $\Rightarrow O(N^2 T)$

$\ddot{y} = f(y)$ ← compute this



N large
T time steps

hand in on this for hard spheres, which is a little different

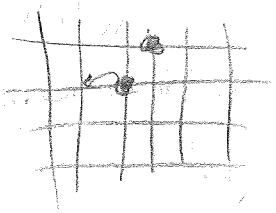
---

SCALING OF CPU / MEMORY is crucial
(because it's Stat Phys & N_A is so big)

Resource scaling example

growing population of N_{cell} cells on a grid (max 1 cell per point)

① multiply, move, die



R space

② at every time step pick a random cells and do ① N_{cell} times. (should be $O(N_{cell})$)

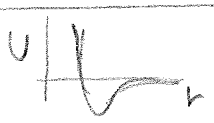
What limits the speed here? How should you store cells?

grid in memory
- good for checking space
- bad for random $O(N_{grid}/N_{cell})$

What would you do?

list of cells
- good for picking random cell
- flexible with boundaries
- bad for checking space $O(N_{cell}^2)$

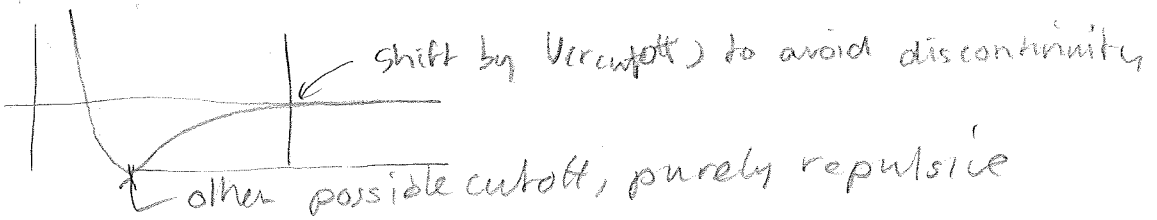
NEXT: interactions (main sink for CPU time)



- Already looked at Lennard-Jones $U_{LJ}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$

often used for weak-ish interactions, i.e Van der Waals

Falls off rapidly: cutoff, speeds things up



Weeks-Chandler-Andersen

$$U = \begin{cases} U_{LJ} + \epsilon & \text{if } r < 2^{1/6}\sigma \\ 0 & \text{otherwise} \end{cases}$$

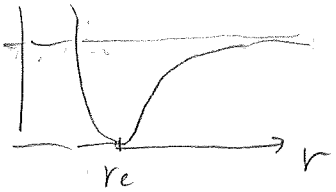
warning!

time step across cutoff has problem, because non-smooth ⇒ less accurate integration

$\dot{y} = f(y,t)$ $\quad y(t+h) = y(t) + hf(y,t) + h^2 f(y,t) + \cdots$

Taylor expansion

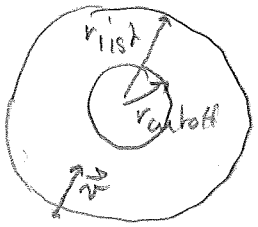
- other potentials, various cutoffs. Morse $U = U_0\{1 - e^{-a(r-r_e)}\}^2$



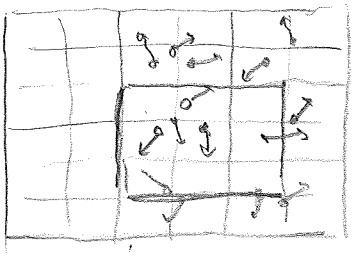
Cutoff r_{cutoff} helps speed things up

particle-particle means $O(N^2)$

- keep list of particles within r_{list} from eachother, ($r_{list} > r_{cutoff}$)
  update periodically $t \in \frac{r_{list} - r_{cutoff}}{v_{max}}$
  still $O(N^2)$ but faster.



- devide into cells



only particles within neighbouring cells can interact.

⇒ considerable bookkeeping

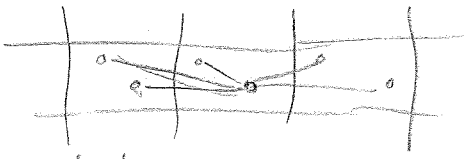
human time vs cpu time & memory

Long-range interactions $\propto \frac{1}{r}$ (electrostatic, gravity)

- cutoff no good
- minimum image convention ⇒ discontinuities

⇒ infinite sums again (screening anyway)

Ewald summation.

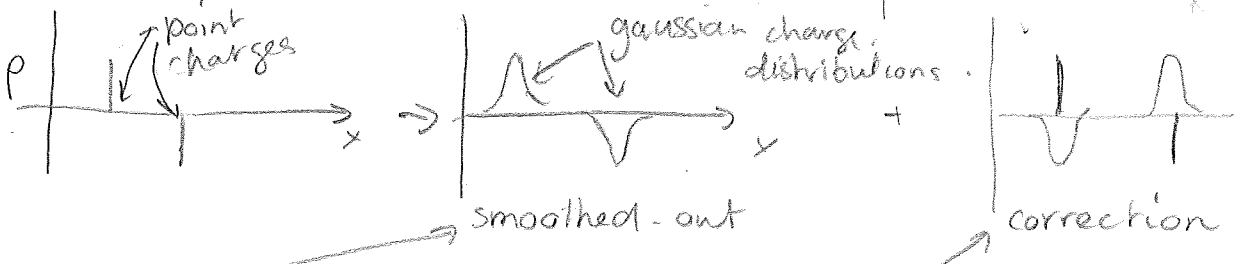


$$U(q,\vec{r}) = \sum_R \sum_j \frac{q q_j}{|\vec{r}_i - \vec{r}_j + R|}$$

← divergent in some points

↑ periodic displacement

not handy.



calculate potential U using Fourier transform

locally neutral; screened charges, short-range.

Warning: exclude self-interactions.

- can be extended to dipoles, etc.
- tricks to speed it up.