1. PARALLEL COMPUTING: CONCEPTS

1.1 Logical Organisation Elements [1, 45-51]

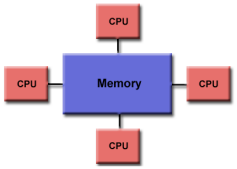
SISD: Single Instruction, Single Data
SIMD: Single Instruction, Multiple Data
MISD: Multiple Instruction, Single Data
MIMD: Multiple Instruction, Multiple Data

1.2 Terminology [1, 52-55]

Node	Standalone computer, comprised of multiple CPUs/cores, networked with other nodes
CPU / Socket / Processor / Core	CPUs with multiple cores are sometimes called sockets
Task	A program, a parallel program consists of multiple tasks
Pipelining	Breaking a task into steps
Shared Memory	Hardware: processors have direct access to common physical memory Software: parallel tasks have the same picture of memory
Symmetric Multi-Processor (SMP)	Multiple processors share a single address space and access to all resources
Distributed Memory	Hardware: network based memory access Software: tasks can only logically see local machine memory
Communications	Data exchange between parallel tasks
Synchronisation	Coordination of parallel tasks in real-time, usually involves waiting by at least one task, can therefore cause a parallel application's wall clock execution time to increase
Granularity	Ratio of computation to communication: - Coarse: large amounts of computational work are done between communication events - Fine: opposite
Observed Speedup	$\frac{\text{wall-clock time of serial execution}}{\text{wall-clock time of parallel execution}}$
Parallel Overhead	Amount of time required to coordinate parallel tasks
Massively Parallel	Hardware that comprises a given parallel system, having many processors
Embarrassingly Parallel	Solving many similar, but independent tasks simultaneously (little to no need for communication between the tasks)
Scalability	Parallel system's ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors

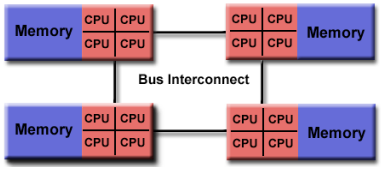
1.3 Shared Memory [1, 56-57]

UMA



Uniform Memory Access

NUMA



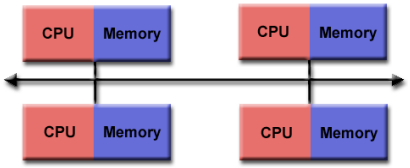
Non-Uniform Memory Access

UMA advantages:
- global address space provides a user-friendly programming perspective to memory
- data sharing between tasks is both fast and uniform

UMA disadvantages:
- lack of scalability between memory and CPUs
- it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors

NUMA disadvantages:
- not all processors have equal access time to all memories
- memory access across link is slower

1.4 Distributed Memory [1, 58-59]



- distributed memory systems require a communication network to connect inter-processor memory
- processors have their own local memory
- each processor operates independently, no cache coherency
- data communication is task of the programmer

Advantages:
- memory is scalable with the number of processors
- each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency
- cost effectiveness: can use commodity, off-the-shelf processors and networking

Disadvantages:
- programmer is responsible for many of the details associated with data communication between processors
- it may be difficult to map existing data structures, based on global memory, to this memory organisation
- slow

→ Hybrid distributed / shared memory

2. PROGRAMMING MODELS

2.1 Shared Memory (without threads) [1, 62]
- tasks share a common address space, which they read and write to asynchronously
- various mechanisms such as locks / semaphores may be used to control access to the shared memory

Advantage: notion of data ownership is lacking; no need to specify explicitly the communication of data between tasks
Disadvantage: difficult to understand and manage data locality

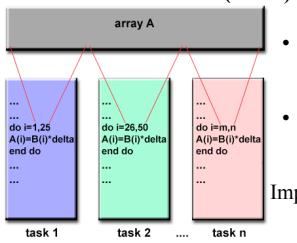
2.2 Shared Memory with threads [1, 63-64]
- a single process can have multiple, concurrent execution paths
- each thread has local data, but also shares the entire resources of the process
- threads communicate with each other through global memory → requires synchronisation constructs
- threads can come and go, but the process remains present

Implementations: POSIX threads, OpenMP

2.3 Distributed Memory / Message Passing [1, 65-66]
- tasks that use their own local memory during computation
- tasks exchange data through communications by sending and receiving messages

Implementation: MPI (Message Passing Interface)

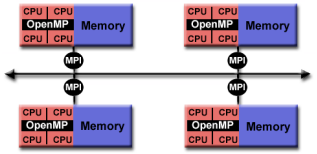
2.4 Data Parallel Model (GPUs) [1, 67-68]



- a set of tasks work collectively on the same data structure, each task works on a different partition of the same data structure
- tasks perform the same operation on their partition of work

Implementation: > Fortran 90 (F90)

2.5 Hybrid Model [1, 69]



- combines more than one of the previously described models
- common example: combination of MPI and OpenMP

2.6 Single Program Multiple Data (SPMD) [1, 70]
All taks execute their copy of the same program simultaneously. Thias program can be threads, message passing, data parallel or hybrid. All tasks may use different data. Probably the most commonly used parallel programming model for multi-node clusters.

2.7 Multiple Program Multiple Data (MPMD) [1, 71]
MPMD applications are not as common as SPMD applications, but may be better suited for certain types of problems.

3. DESIGNING PARALLEL PROGRAMS

3.1 Decomposition / Partitioning [1, 76-78]

→ break the problem into discrete „chunks“ of work

Domain Decomposition: each parallel task works on a portion of the data

Functional Decomposition: each task performs a portion of the overall work
(e.g. signal processing with multiple filters, climate modeling)

3.2 Communication [1, 79-81]

Cost of communications:
- inter-task communication virtually always implies overhead
- machine cycles and resources that could be used for computations are instead used to package and transmit data
- communications frequently require some type of synchronisation between tasks, which can result in tasks spending time waiting instead of doing work
- competing communication traffic can saturate the available network bandwidth, aggravating performanc

Latency vs. Bandwidth: sending many small packages can cause latency to dominate communication overheads.

synchronous communication: blocking, requires handshake
asynchronous communication: non-blocking

Scope of communications: Point-to-Point
Collective (e.g. broadcast)

3.3 Synchronisation [1, 82]

Barrier: Each task performs its work until it reaches the barrier. It then stops or blocks. When the last task reaches the barrier, all tasks are sychronised.

Lock / Semaphore: serialise access to global data or code section

Synchronous communication operation: e.g.: before a task can perform a send operation, it must first receive an acknowledgment from the receiving task.

3.4 Data Dependencies [1, 83]

A **dependence** exists between program statements when the order of statement execution affects the results of the program.
A **data dependence** results from multiple use of the same location(s) in storage by different tasks.

3.5 Load Balancing [1, 84-85]

→ distributing work among tasks so that all tasks are kept busy all of the time

Distribute the data evenly among the tasks if each task has to do similar work. (Exception: heterogeneous mix of machines with varying performance)

Certain classes of problems result in load imbalances even if data is evenly distributed among tasks:
- Sparse arrays - some tasks will have actual data to work on while others have mostly "zeros".
- Adaptive grid methods - some tasks may need to refine their mesh while others don't.
- N-body simulations - where some particles may migrate to/from their original task domain to another task's; where the particles owned by some tasks require more work than those owned by other tasks.

→ use scheduler – task pool approach (dynamic work assignment)

3.6 Granularity [1, 86]

= ratio of computation to communication

Fine-grain Parallelism: Relatively small amounts of computational work are done between communication events

Coarse-grain Parallelism: Relatively large amounts of computational work are done between communication/synchronization events

---

3.7 I/O [1, 87-88]

I/O operations are generally regarded as inhibitors to parallelism.

Rule #1: Reduce overall I/O as much as possible.

3.8 Scalability, Limits and Costs [1, 89-90]

Amdahl's Law: potential program speedup, $\text{speedup} = \frac{1}{1-P}$

where P denotes the fraction of code that can be parallelised

Introducing the number of processors N: $\text{speedup} = \frac{1}{\frac{P}{N} + S}$

where $S = 1 - P$ the serial fraction

Problems that increase the percentage of parallel time with their size are more scalable than problems with a fixed percentage of parallel time.

Text

$\frac{d}{dt} \Rightarrow$

---

4. PERFORMANCE EVALUATION

4.1 Basics [2a, 13]

Execution time on p processors: $T(p)$
- $T(1)$ is the best time on a single CPU core
- $pT(p) \geq T(1)$

Speedup: $S(p) = T(1)/T(p)$
- clearly $S(p) \leq p$
- $S(p)<1$ is possible, means slower than on a single core. One may be forced to use such a parallel code for memory reasons.

Efficiency: $E(p) = \frac{S(p)}{p} \leq 1$

4.2 Amdahl's Law [2a, 14]

$$T(p) = \left(s + \frac{1-s}{p}\right) T(1) + T_{\text{overhead}}(p)$$

where s denotes fraction of serial code

Maximum speedup:

$$S(p) = \frac{T(1)}{T(p)} = \frac{T(1)}{(s+(1-s)/p)T(1) + T_{\text{overhead}}(p)} \leq \frac{T(1)}{(s+(1-s)/p)T(1)} = \frac{1}{s+(1-s)/p} \leq \frac{1}{s}$$

→ If only 0.1% of the code is serial, it will never scale beyond 1000 cores.

4.3 Strong vs. Weak Scaling [2a, 15]

Strong scaling: keep the problem size constant as you increase the number of CPU cores N

Weak scaling: increase the problem size with N, can help beat Amdahl's law if the serial part does not depend on problem size

5. MONTE CARLO INTEGRATION

5.1 Integrating a Function [2a, 17-18]

Convert the integral to a discrete sum:

$$\int_a^b f(x)\,dx = \frac{b-a}{N}\sum_{i=1}^{N} f\left(a+i\frac{b-a}{N}\right)+O\left(\frac{1}{N}\right)$$

Higher order integrators

Trapezoidal rule:

$$\int_a^b f(x)\,dx = \frac{b-a}{N}\left(\frac{1}{2}f(a)+\sum_{i=1}^{N-1} f\left(a+i\frac{b-a}{N}\right)+\frac{1}{2}f(b)\right)+O\left(\frac{1}{N^2}\right)$$

Simpson's rule:

$$\int_a^b f(x)\,dx = \frac{b-a}{3N}\left(f(a)+\sum_{i=1}^{N-1}\left(3-(-1)^i\right)f\left(a+i\frac{b-a}{N}\right)+f(b)\right)+O\left(\frac{1}{N^4}\right)$$

Simpson rule with M points per dimension:

one dimension the error is $O(M^{-4})$

d dimensions we need $N=M^d$ points, the error is order $O(M^{-4})=O(N^{-4/d})$

→ An order-n scheme in 1 dimension is order-n/d in d dimensions!

→ Integration becomes extremely inefficient!

5.2 Monte Carlo Integration & Error Estimation [2a, 21-24]

$$\langle f\rangle = \frac{\int_\Omega f(\vec{x})\,d\vec{x}}{\int_\Omega d\vec{x}}$$

Instead of evaluating it at equally spaced points evaluate it at M points xi chosen randomly in Ω: $\langle f\rangle \approx \frac{1}{M}\sum_{i=1}^{M} f(\vec{x}_i)$

The error is statistical: $\Delta = \sqrt{\frac{\mathrm{Var}\,f}{M}} \propto M^{-1/2}$ $\qquad \mathrm{Var}\,f = \langle f^2\rangle - \langle f\rangle^2$

In d>8 dimensions Monte Carlo scales better than Simpson.

Expectation value: $\mathrm{E}[aX+bY] = a\,\mathrm{E}[X]+b\,\mathrm{E}[Y]$

Expectation value of the mean: $\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$

$$\mathrm{E}[\bar{X}] = \mathrm{E}\left[\frac{1}{N}\sum_{i=1}^{N} X_i\right] = \frac{1}{N}\sum_{i=1}^{N}\mathrm{E}[X_i] = \frac{1}{N}\sum_{i=1}^{N} E[X] = \mathrm{E}[X]$$

⇒ the mean is a true estimator

The sampling error is the rms (root mean square) deviation:

$$\begin{aligned}(\Delta X)^2 &= \mathrm{E}\left[(\bar{X}-\mathrm{E}[X])^2\right] = \mathrm{E}\left[\left(\frac{1}{N}\sum_{i=1}^{N}(X_i-\mathrm{E}[X])\right)^2\right]\\ &= \mathrm{E}\left[\frac{1}{N^2}\sum_{i,j=1}^{N}(X_i-\mathrm{E}[X])(X_j-\mathrm{E}[X])\right]\\ &= \frac{1}{N^2}\sum_{i,j=1}^{N}\left(\mathrm{E}[X_iX_j]-E[X]^2\right)\\ &= \frac{1}{N^2}\sum_{i=1}^{N}\left(E[X_i^2]-E[X]^2\right) = \frac{1}{N}\left(\mathrm{E}[X^2]-E[X]^2\right) = \frac{\mathrm{Var}\,X}{N}\end{aligned}$$

Samples are uncorrelated: $E[X_iX_j] = \mathrm{E}[X_i]\mathrm{E}[X_j]$ for $i \neq j$

Recipe for Monte Carlo integration and data analysis

1. Draw random points x and evaluate the function, to get random variables $X=f(x)$
2. Store the number, sum and sum of squares
   N, $\sum_{i=1}^{N} X_i$, $\sum_{i=1}^{N} X_i^2$
3. Calculate the mean as an estimate of the expectation value:
   $\mathrm{E}[X] \approx \bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$
4. Estimate the variance from the mean of squares and from it the error:
   $\mathrm{Var}\,X = \mathrm{E}[X^2]-\mathrm{E}[X] \approx \frac{N}{N-1}\left(\overline{X^2}-\bar{X}^2\right) = \frac{N}{N-1}\left(\frac{1}{N}\sum_{i=1}^{N} X_i^2-\bar{X}^2\right)$
   ⇒ $\Delta X \approx \sqrt{\frac{1}{N-1}\left(\overline{X^2}-\bar{X}^2\right)}$

5.3 Importance Sampling [2a, 25-26]

Sharply peaked functions

- in many cases a function is large only in a tiny region
- lots of time wasted in regions where the function is small
- the sampling error is large since the variance is large

→ Importance Sampling

Choose points not uniformly but with probability $p(x)$:

$$\langle f\rangle = \left\langle \frac{f}{p}\right\rangle_p := \frac{\int_\Omega \frac{f(\vec{x})}{p(\vec{x})}p(\vec{x})\,d\vec{x}}{\int_\Omega d\vec{x}}$$

The error is now determined by $\mathrm{Var}\frac{f}{p}$.

Find p similar to f and such that p-distributed random numbers are easily available.

6. MARKOV CHAIN MONTE CARLO

6.1 Monte Carlo for Physical Systems [3a, 4]

Evaluate phase space integral by importance sampling:

$$\langle A\rangle = \frac{\int_\Omega A(c)p(c)\,dc}{\int_\Omega p(c)\,dc} \quad \Rightarrow \langle A\rangle \approx \bar{A} = \frac{1}{M}\sum_{i=1}^{M} A_{c_i}$$

Pick configurations with the correct Boltzmann weight:

$$P[c] = \frac{p(c)}{Z} = \frac{\exp(-\beta E(c))}{Z}$$

6.2 Markov Chain Monte Carlo [3a, 7]

Instead of drawing independent samples c_i build a Markov chain:
$c_1 \to c_2 \to \ldots \to c_i \to c_{i+1} \to \ldots$

Transition probabilities $W_{x,y}$ for transition $x \to y$ need to satisfy:

- Normalisation: $\sum_y W_{x,y} = 1$
- Ergodicity: any configuration reachable from any other:
  $\forall x,y \exists n : \left(W^n\right)_{x,y} \neq 0$
- Balance: the distribution should be stationary:
  change in distribution in one step: $p_y^{(n+1)} = \sum_x W_{x,y}\,p_x^{(n)}$
  stationarity condition: $p_y^{(n+1)} = p_y^{(n)} \Rightarrow p_y = \sum_x W_{x,y}\,p_x$

  Detailed balance is sufficient but not necessary for balance: $\frac{W_{x,y}}{W_{y,x}} = \frac{p(y)}{p(x)}$

7. METROPOLIS ALGORITHM

- propose a change with an a-priori proposal rate $A_{x,y}$
- accept the proposal with a probability $P_{x,y}$
- the total transition rate is $W_{x,y}=A_{x,y}P_{x,y}$

$$P_{x,y}=\min\left[1,\frac{A_{y,x}p_y}{A_{x,y}p_x}\right]$$

7.1 Boltzmann Weight [3a, 9]

At a fixed temperatur T the average of a physical observable A can be calculated as a sum over all configurations c : $\langle A\rangle=\frac{1}{Z}\sum_c A_c\exp\left(-\beta E_c\right)$

c	configuration
E_c	energy of a configuration
A_c	value of the observable for a configuration
T	temperature
$\beta=\frac{1}{k_B T}$	inverse temperature
$Z=\sum_c \exp\left(-\beta E_c\right)$	partition function (normalisation)

This is ideal for importance sampling with the Boltzmann weight

$$p_c=\frac{1}{Z}\exp\left(-\beta E_c\right)$$

7.2 Sampling N-Body States [3a, 12-15]

Lennard-Jones potential: $V_{LJ}=4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12}-\left(\frac{\sigma}{r}\right)^{6}\right]=\epsilon\left[\left(\frac{r_m}{r}\right)^{12}-2\left(\frac{r_m}{r}\right)^{6}\right]$

Simple sampling:

- draw random configurations and calculate their energy and weight
- measure $\langle A\rangle\approx\frac{\sum_i A_i\exp\left(-\beta E_i\right)}{\sum_i \exp\left(-\beta E_i\right)}$
- problem: we will never hit a crystal that way

Importance sampling by Markov chains

Start from a suitable initial condition, e.g. a perfect crystal.
Then do the following updates:

- choose a random particle
- choose a random direction and distance, e.g. by Gaussian distribution with sensible parameters
- accept / reject with Boltzmann weight and Metropolis sampling
- measure $\langle A\rangle\approx\frac{1}{N}\sum_{i=1}^{N} A_i$

Equilibration

Starting from a random initial configuration it takes a while to reach the equilibrium distribution. The desired equilibrium distribution is a left eigenvector with eigenvalue 1. (This is just the balance condition.)

$$p_y=\sum_x p_x W_{x,y}$$

Convergence is controlled by the second largest eigenvalue

$$p_x(t)\to p(x)+O\left(\lambda_2^t\right)$$

We need to run the simulation for a while to equilibrate and only then start measuring, to be safe skip at least about 100 autocorrelation times.

7.3 Parallelisation of MC Simulations [3a, 16-17]

2 Strategies:

- parallelise the updates in a single simulation
- run multiple independent simulations in parallel

Scaling of MC simulations

Direct sampling:

- independent simulation (clone) on each core
- small overhead to set up the clones
- log(N) communication to collect the results
- near perfect scaling → perfectly parallel

Markov chain Monte Carlo:

- each clone needs to be equilibrated
- scaling and speedup limited by equilibration time

8. MONTE CARLO ERROR ANALYSIS

8.1 Error Estimation [3a, 23-24]

Monte Carlo error estimation: $(\Delta X)^2=\frac{\mathrm{Var}\,X}{N}$ (see above)

Used that samples are uncorrelated: $\mathrm{E}\left[X_i X_j\right]=\mathrm{E}\left[X_i\right]\mathrm{E}\left[X_j\right]$ for $i\neq j$

Now include correlations:

$$\begin{aligned}(\Delta X)^2 &= \frac{1}{N^2}\sum_{i,j=1}^{N}\left(\mathrm{E}\left[X_i X_j\right]-\mathrm{E}[X]^2\right)\\ &= \frac{\mathrm{Var}\,X}{N}+\frac{1}{N^2}\sum_{i\neq j}^{N}\left(\mathrm{E}\left[X_i X_j\right]-\mathrm{E}[X]^2\right)\\ &= \frac{\mathrm{Var}\,X}{N}+\frac{2}{N^2}\sum_{i=1}^{N}\sum_{t}\left(\mathrm{E}\left[X_i X_{i+t}\right]-\mathrm{E}[X]^2\right)\\ &= \frac{\mathrm{Var}\,X}{N}\left(1+2\tau_x\right)\end{aligned}$$

where we defined the integrated autocorrelation time as

$$\tau_x=\frac{\sum_t\left(\mathrm{E}\left[X_i X_{i+t}\right]-\mathrm{E}[X]^2\right)}{\mathrm{Var}\,X}$$

8.2 Binning Analysis [3a, 25]

Take averages of consecutive measurements: averages become less correlated and naive error estimates converge to real error.

$$A_i^{(l)}=\frac{1}{2}\left(A_{2i-1}^{(l-1)}+A_{2i}^{(l-1)}\right)$$

$$\Delta^{(l)}=\sqrt{\mathrm{Var}\,\frac{A^{(l)}}{M^{(l)}}}\quad\overset{l\to\infty}{\to}\quad\Delta=\sqrt{\left(1+2\tau_A\right)\mathrm{Var}\,\frac{A}{M}}\qquad ???$$

$$\tau_A=\lim_{l\to\infty}\frac{1}{2}\left(\frac{2^l\mathrm{Var}\,A^{(l)}}{\mathrm{Var}\,A^{(0)}}-1\right)$$

8.3 Correlated Quantities [3a, 27]

How do we calculate the errors of functions of correlated measurements?

Specific heat: $c_v=\frac{\langle E^2\rangle-\langle E\rangle^2}{k_B T^2}$

Expectation values of weighted samples in direct sampling:

$$\langle A\rangle=\frac{\left\langle\sum_c A_c\exp\left(-\beta E_c\right)\right\rangle}{\left\langle\sum_c \exp\left(-\beta E_c\right)\right\rangle}$$

8.4 Jackknife Analysis [3a, 28-29]

Evaluate the function on all and all but one segment.

$$U_0=f\left(\frac{1}{M}\sum_{i=1}^{M}X_i,\frac{1}{M}\sum_{i=1}^{M}Y_i\right)\qquad U_1=f\left(\frac{1}{M-1}\sum_{i=2}^{M}X_i,\frac{1}{M-1}\sum_{i=2}^{M}Y_i\right)$$

$$U_j=f\left(\frac{1}{M-1}\sum_{\substack{i=1\\ i\neq j}}^{M}X_i,\frac{1}{M-1}\sum_{\substack{i=1\\ i\neq j}}^{M}Y_i\right)$$

$$\langle U\rangle\approx U_0-(M-1)\left(\bar{U}-U_0\right)\qquad\bar{U}=\frac{1}{M}\sum_{i=1}^{M}U_i\qquad\Delta U=\sqrt{\frac{M-1}{M}\sum_{i=1}^{M}\left(U_i-\bar{U}\right)^2}$$

9. N-BODY PROBLEM

9.1 Dynamics of the N-Body Problem [6, 2-3]

Newton's equation of motion: $m_i \frac{d^2 \vec{x}_i}{dt^2} = \vec{F}_i = -\nabla_{\vec{x}_i} V(\vec{x}_1, \ldots, \vec{x}_N)$

$\nabla_{\vec{x}_i}$: gradient with respect to position of particle *i*

total energy: $E(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N) = E_{\text{kin}}(\vec{v}_1, \ldots, \vec{v}_N) + V(\vec{x}_1, \ldots, \vec{x}_N)$

kinetic energy: $E_{\text{kin}}(\vec{v}_1, \ldots, \vec{v}_N) = \sum_i \frac{m_i}{2} |\vec{v}_i|^2$

Interaction

$V(\vec{x}_1, \ldots, \vec{x}_N) = \frac{1}{2} \sum_{i \neq j} V_{ij}(r_{ij})$ where $r_{ij} = |\vec{x}_i - \vec{x}_j|$.

Coulomb: q_i is the charge of particle *i*

$$V_{ij}(r) = \frac{q_i q_j}{r}$$

Garvity: m_i is the mass of particle *i*

$$V_{ij}(r) = -G \frac{m_i m_j}{r}$$

Lennard Jones (van-der-Waals and hard-core)

$$V_{ij}(r) = 4 \varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r} \right)^{12} - \left(\frac{\sigma_{ij}}{r} \right)^{6} \right] = \varepsilon_{ij} \left[\left(\frac{r_{m,ij}}{r} \right)^{12} - \left(\frac{r_{m,ij}}{r} \right)^{6} \right]$$

9.2 Ergodicity [6, 4]

The long-time average of the dynamics is the same as the ensemble average:

$$\lim_{T \to \infty} \frac{1}{T} \int_0^T dt\, A(t) = \frac{\int d\vec{x}_1 \cdots d\vec{x}_N\, d\vec{v}_1 \cdots d\vec{v}_N\, A(\vec{x}_1, \ldots, \vec{x}_N)\, w(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N)}{\int d\vec{x}_1 \cdots d\vec{x}_N\, d\vec{v}_1 \cdots d\vec{v}_N\, w(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N)}$$

The weights depend on the ensemble.

9.3 Integration [6, 5-6]

Integration of the equations of motion conserves energy: microcanonical ensemble, where all states with the right energy have the same weight

$$w(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N) = \delta\left(E(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N) - E_{tot}\right)$$

At fixed temperature we get the canonical ensemble with Boltzmann weights

$$w(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N) = \exp\left(-\beta E(\vec{x}_1, \ldots, \vec{x}_N, \vec{v}_1, \ldots, \vec{v}_N)\right)$$

As $N \to \infty$ both ensembles give the same result.
The Boltzmann weight factorises into a potential and a kinetic part. This allows us to integrate out the motion in the canonical ensemble:

$$\langle A \rangle = \frac{\int d\vec{x}_1 \cdots d\vec{x}_N\, A(\vec{x}_1, \ldots, \vec{x}_N) \exp\left(-\beta V(\vec{x}_1, \ldots, \vec{x}_N)\right)}{\int d\vec{x}_1 \cdots d\vec{x}_N \exp\left(-\beta V(\vec{x}_1, \ldots, \vec{x}_N)\right)}$$

Newton's equation of motion is just a system of coupled ordinary differential equations:

$$\frac{d\vec{x}_i}{dt} = \vec{v}_i \,, \quad \frac{d\vec{v}_i}{dt} = \vec{a}_i = \frac{1}{m_i} \vec{F}_i = -\frac{1}{m_i} \nabla_{\vec{x}_i}(\vec{x}_1, \ldots, \vec{x}_N)$$

Verlet algorithm

$$\vec{x}_i(t + \Delta t) = \vec{x}_i(t) + \vec{v}_i(t)\, \Delta t + \frac{(\Delta t)^2}{2} \vec{a}_i(t) + O(\Delta t^3)$$

$$\vec{v}_i(t + \Delta t) = \vec{v}_i(t) + \frac{\Delta t}{2} \left(\vec{a}_i(t) + \vec{a}_i(t + \Delta t) \right) + O(\Delta t^2)$$

9.4 Initial Conditions [6, 7]

- Place the particles in an appropriate way. How depends on the problem at hand: random or in a crystal.
- calculate the potential energy $E_{\text{pot}} = V(\vec{x}_1, \ldots, \vec{x}_N)$
- choose random velocities, e.g. in a Gaussian distribution
- calculate the kinetic energy $E_{\text{kin}}(\vec{v}_1, \ldots, \vec{v}_N) = \sum_i \frac{m_i}{2} |\vec{v}_i|^2$
- rescale the velocities to obtain the desired energy

$$E_{\text{pot}} + \lambda^2 E_{\text{kin}} = E_{\text{tot}} \quad \Rightarrow \quad \lambda = \sqrt{\frac{E_{\text{tot}} - E_{\text{pot}}}{E_{\text{kin}}}} \,, \quad \vec{v}_i \leftarrow \lambda \vec{v}_i$$

- during the run rescale the velocities from time to time to correct roundoff errors

---

9.5 Constant Temperature [6, 8]

Temperature is just the mean kinetic energy. *G* is the number of kinetic degrees of freedom. *G=d* for point particles in d dimensions.

$$\langle E_{\text{kin}} \rangle = \frac{G}{2} k_B T$$

To change the ensemble from constant energy to constant temperature we use a „thermostat“, e.g. the Nosé-Hoover thermostat:

- add a friction term to the acceleration $m_i \frac{d\vec{v}_i}{dt} = \vec{F}_i - \eta \vec{v}_i$
- the friction term heats up or cools down the motion to keep the desired temperature

$$\frac{d\eta}{dt} = \frac{1}{m_s} \left(E_{\text{kin}} - \frac{G}{2} k_B T \right)$$

- choose m_s to have a fast relaxation without oscillations

10. N-BODY PROBLEM: BOUNDARIES

10.1 Boundary Conditions [6, 9]

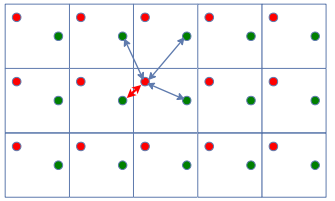
Open system with no boundaries:
- useful for bound systems (gravity, crystal)
- useless for gas, which will just fly apart

Hard walls:
- particles get reflected at the boundaries
- needs check whether particles cross the boundary

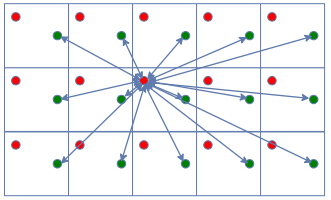
Periodic boundary conditions:
- particles exiting on one side reenter on the opposite
- needs check whether particles cross the boundary
- need to be careful in calculating distances
- advantage: no boundary effects (but still finite size effects)

10.2 Force Calculation with Periodic Boundaries [6, 10-11]

The System is infinitely repeated.



Short range forces:
pick nearest periodic image



Long range forces:
sum over all periodic images

10.3 Ewald Summation [6, 12]

Infinite sum for long-range forces, e.g. for Coulomb:

$$V_{ij}(\vec{r} = \vec{x}_i - \vec{x}_j) = \sum_{\vec{n}} \sum_i \frac{q_i q_j}{|\vec{r}_{\vec{n}} - \vec{r}|} \,,$$

$\vec{n}$ is an integer-valued vector enumerating the repeated images.
$\vec{r}_{\vec{n}}$ is the vector from the origin of the original cell to the image $\vec{n}$.

The sum converges very slowly and cannot easily be truncated.

Ewald summation: rewrite it as a sum of two rapidly converging sums:
- a short range part summes in real space
- a complementary long range part summed in Fourier space

---

10. N-BODY PROBLEM: BOUNDARIES (CONT.)

10.4 Short Range vs. Long Range [6, 13]

Long range forces require lots of calculation! Short range is much faster! A force is short range when it is finite range but also infinite range if it decays rapidly.

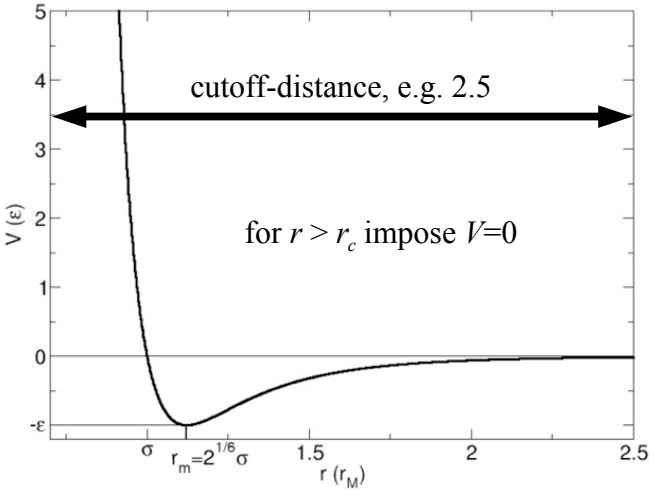
We want the truncation on the potential to be small: $\iiint_{|\vec{r}|>R_c} V(\vec{r})\,d\vec{r} \overset{R_c\to\infty}{\to} 0$

For power law decay this implies: $V(r)\propto r^{-a}$

$$\iiint_{|\vec{r}|>R_c} V(\vec{r})\,d\vec{r} \;\propto\; \int_{R_c}^{\infty} r^{-a} r^{d-1}\,dr \;\propto\; r^{d-a} \overset{R_c\to\infty}{\to} 0 \;\Rightarrow\; a>d$$

10.5 Truncation of the LJ Potential [6, 14-17]

Truncate to save computational time (more negligible interactions).



Truncation leads to some issues:

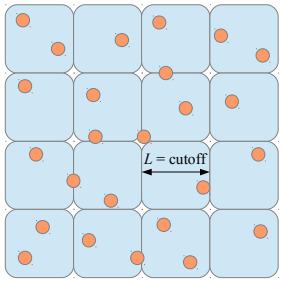
- potential truncated at e.g. $r_c=2.5\,\sigma$ (r_c is the cutoff-distance)
- offset in the potential is $V_{LJ}(r_c)=-\frac{1}{61.3}\varepsilon$ → jump discontinuity
- FIX: shift the potential upward, in order to cancel the offset:

$$V_{LJ_{\text{trunc}}}(r)=\begin{cases} V_{LJ}(r)-V_{LJ}(r_c) & \text{for } r\le r_c \\ 0 & \text{for } r>r_c \end{cases}$$

11. N-BODY PROBLEM: CELL LISTS

11.1 Domain Subdivision into Cells [6, 18]

- cell size L is the cutoff
- cells contain particles (or particle indices)
- particles spread into cells according to their position



11.2 Computing the Force on one Particle [6, 19]

Evaluation of the forces on a particle:
take into account only particles in the cell and its neighbours

Cost of computing N forces in 2D:

- $c = N/m$, m cells
- assume constant c
- complexity is $9cN$
- algorithmic improvement: $\frac{N^2}{9cN}=\frac{N}{9c}$

11.3 Implementation [6, 20-21]

Note:

- inverses are expensive
- calculating powers is expensive
- calculating square roots is very expensive

Implementation Suggestion A:

- For each cell create a container of particles in this cell.
- Iterate over all particles and put it into the cell it belongs to.

Implementation Suggestion B:

- Calculate the cell index for each particle and store it with the particles.
- Sort the particles by cell index.
- For each cell store a pointer or index to the first and last particle.

12. DIFFUSION

12.1 Brownian Motion [7, 3-6]

$\langle m u^2 \rangle = \frac{1}{2} k T$ Boltzmann constant: $k = 1.381 \cdot 10^{-23} \frac{\mathrm{J}}{\mathrm{K}} = \frac{\mathrm{kg\,m^2}}{\mathrm{K\,s^2}}$

m: Mass u: Velocity T: Temperatur [K]

12.2 Random Walk in 1D [7, 7-11]

Assumptions:
- Each particle steps to the left or to the right once every τ, moving with a velocity $\pm U$ a distance $\delta = \pm U \tau$ (τ, δ are constants)
- The probability of going to the left or to the right is ½. Successive steps are independent. The walk is not biased.
- Each particle moves independently of all other particles.

Consequences:
- Each particle goes nowhere on the average.
- Root-Mean-Square displacement is proportional to square root of time.

Dynamics: $x_i(n) = x_i(n-1) \pm \delta$

Mean displacement of particles after the *n*-th step:

$$\begin{aligned}\langle x(n) \rangle &= \frac{1}{N}\sum_{i=0}^{N} x_i(n) = \frac{1}{N}\sum_{i=0}^{N}\left(x_i(n-1) \pm \delta\right) = \frac{1}{N}\sum_{i=0}^{N} x_i(n-1) \\ &= \langle x(n-1) \rangle = \ldots = \langle x(0) \rangle = 0\end{aligned}$$

→ Particles spread symmetric about the origin.

Mean square displacement of particles after the *n*-th step:

$$\langle x^2(n) \rangle = \frac{1}{N}\sum_{i=0}^{N} x_i^2(n)$$

$$x_i^2(n) = \left[x_i(n-1) \pm \delta\right]^2 = x_i^2(n-1) \pm 2\delta x_i(n-1) + \delta^2$$

$$\begin{aligned}\langle x^2(n) \rangle &= \frac{1}{N}\sum_{i=0}^{N}\left[x_i^2(n-1) \pm 2\delta x_i(n-1) + \delta^2\right] \\ &= \langle x^2(n-1) \rangle + \delta^2 = \langle x^2(n-2) \rangle + 2\delta^2 = \ldots = \langle x^2(0) \rangle + n\delta^2 \\ &= n\delta^2\end{aligned}$$

The particles spread $\langle x^2(n) \rangle = \frac{\delta^2}{\tau} t$.

Define diffusion coefficient: $D = \frac{\delta^2}{2\tau}$ → $\langle x^2 \rangle = 2Dt$

12.3 Derivation of the Diffusion Equation [7, 12-18]

Let $\nu = f(x,t)$ be the number of particles per unit volume.

$$f(x, t+\tau) = f(x,t) + \tau \frac{\partial f}{\partial t}$$

Taylor expansion: $f(x+\Delta, t) = f(x,t) + \Delta \frac{\partial f}{\partial x} + \frac{\Delta^2}{2!}\frac{\partial^2 f}{\partial x^2} + \ldots$

Diffusion equation: $\frac{\partial f}{\partial t} = D \frac{\partial^2 f}{\partial x^2}$

13. SPARSE LINEAR ALGEBRA

13.1 Diffusion 1D [9, 2-3]

1D diffusion equation: $\frac{\partial f}{\partial t} = c \frac{\partial^2 f}{\partial x^2}$ Discretise space: $x_i = i \Delta x$

$$c \frac{\partial^2 f}{\partial x^2} \approx c \frac{f(x_{i+1}) + f(x_{i-1}) - 2f(x_i)}{(\Delta x)^2}, \quad \frac{\partial f}{\partial t} \approx \frac{f(x_i, t+\Delta t) - f(x_i, t)}{\Delta t}$$

Forward-Euler integrator:

$$f(x_i, t+\Delta t) \approx f(x_i, t) + \frac{c\Delta t}{(\Delta x)^2}\left[f(x_{i+1}, t) + f(x_{i-1}, t) - 2f(x_i, t)\right]$$

Rewriting as a matrix problem

Interpret the function at time t as a vector $f(t)$ with elements $f_i(t) = f(x_i, t)$.

$$f_i(t+\Delta t) = f_i(t) + \frac{c\Delta t}{(\Delta x)^2}\left[f_{i+1}(t) + f_{i-1}(t) - 2f_i(t)\right]$$

Fixed boundary conditions: $f_1(t+\Delta t) = f_1(t)$, $f_L(t+\Delta t) = f_L(t)$

Therewith, the following matrix equation are obtained:

$$f_i(t+\Delta t) = \sum_{j=1}^{N} M_{ij} f_j(t) \quad \rightarrow \quad \vec{f}(t+\Delta t) = M\vec{f}(t)$$

$$M = \begin{pmatrix} 1 & 0 & & & 0 \\ \frac{c\Delta t}{(\Delta x)^2} & 1 - \frac{2c\Delta t}{(\Delta x)^2} & \frac{c\Delta t}{(\Delta x)^2} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{c\Delta t}{(\Delta x)^2} & 1 - \frac{2c\Delta t}{(\Delta x)^2} & \frac{c\Delta t}{(\Delta x)^2} \\ 0 & & & 0 & 1 \end{pmatrix}$$

13.2 Diffusion 2D [9, 4-5]

2D diffusion equation: $\frac{\partial f}{\partial t} = c\Delta f = c\frac{\partial^2 f}{\partial x^2} + c\frac{\partial^2 f}{\partial y^2}$

Discretise space: $\vec{r}_{i,j} = (i\Delta x, j\Delta x)$

Finite difference equation:

$$\begin{aligned}f(\vec{r}_{i,j}, t+\Delta t) \approx f(\vec{r}_{i,j}, t) + \frac{c\Delta t}{(\Delta x)^2}\big[&f(\vec{r}_{i+1,j}, t) + f(\vec{r}_{i-1,j}, t) \\ &+ f(\vec{r}_{i,j+1}, t) + f(\vec{r}_{i,j-1}, t) - 4f(\vec{r}_{i,j}, t)\big]\end{aligned}$$

Rewriting as a matrix problem

On a L×L mesh use the indexing $f_{i+Lj}(t) = f(\vec{r}_{i,j}, t)$

Matrix equation: $\vec{f}(t+\Delta t) = M\vec{f}(t)$

M has nonzero entries only for $i - j = (0, \pm 1, \pm L)$.

13. SPARSE LINEAR ALGEBRA (CONT.)

13.3 Poisson Equation [9, 7-8]

Poisson equation: $\Delta\phi = f$

It relates the potential of gravity to the mass distribution:
- $\Delta\phi = 4\pi G\rho$
- $G = 6.673\cdot 10^{-11}\frac{\mathrm{m}^3}{\mathrm{kg\,s}^2}$ is the gravitational constant
- ρ is the mass density

It relates the electric potential to the cahrge distribution:
- $\Delta V = -\frac{\rho}{\varepsilon}$
- ε = permitivity of the medium $\left(8.85\cdot 10^{-12}\,\mathrm{F/m}\text{ in vacuum}\right)$
- ρ is the charge density

Put it on a mesh like the diffusion equation and write it as a matrix problem:
- $\sum_j M_{ij}\,\phi_j = 4\pi G\rho_j$
- $M\,\vec{\phi} = 4\pi G\vec{\rho}$

Solving the Poisson equation
The Poisson equation can be solved by iteration:

$$\frac{1}{(\Delta x)^2}\left[\phi(\vec{r}_{i+1,j})+\phi(\vec{r}_{i-1,j})+\phi(\vec{r}_{i,j+1})+\phi(\vec{r}_{i,j-1})-4\phi(\vec{r}_{i,j})\right] = 4\pi G\rho(\vec{r}_{i,j})$$

$$\Rightarrow\ \phi(\vec{r}_{i,j}) = \frac{1}{4}\left[\phi(\vec{r}_{i+1,j})+\phi(\vec{r}_{i-1,j})+\phi(\vec{r}_{i,j+1})+\phi(\vec{r}_{i,j-1})\right]-\pi G(\Delta x)^2\rho(\vec{r}_{i,j})$$

Start with random guess $\vec{\phi}^{(0)}$.
Iterate the fixed point equation: $\vec{\phi}^{(n+1)} = M\,\vec{\phi}^{(n)} - \pi G(\Delta x)^2\vec{\rho}$

Speed up convergence with successive overrelaxation (SOR):
$\vec{\phi}^{(n+1)} = (1-\alpha)\vec{\phi}^{(n)} + \alpha\left[M\,\vec{\phi}^{(n)} - \pi G(\Delta x)^2\vec{\rho}\right]$, where $1\le\alpha<2$

13.4 Sparse and Dense Matrices [9, 9-10]

Sparse vector of length N:
- $m << N$ non-zero entries
- vector operations with O(m) instead O(N) effort
- storage requirements are O(m) instead of O(N)

Sparse matrix of size $N\times N$:
- often m=O(N) or m=O(N log N) non-zero entries
- storage requirements are O(m) instead of O(N^2)

Operation	Dense matrix	Sparse matrix, O(N) non-zero entries
Matrix addition	O(N^2)	O(N)
Matrix-vector multiplication	O(N^2)	O(N)
Linear eq. solver	O(N^3)	O(N)
Calculate eigenvalues	O(N^2)	O(N)

13.5 Page Rank [9, 12-16]

The page rank matrix is used to rank web pages. It is a diffusion matrix on the graph of all web pages, mimicking a random surfer. The simplest version is:

- Pick one of the links on a page at random. Jump to a random page from all pages if there is no link on a page.
- The Matrix row for a given page s contains an entry $1/L(s)$ in every column corresponding to one of the $L(s)$ pages that it links to.
- Since all entries are positive and the row sums are 1, this is a Markov transition matrix.
- The equilibrium distribution gives the page rank. That is the largest left eigenvector of the matrix: $p_y = \sum_x W_{x,y}\,p_x \ \Leftrightarrow\ \vec{p}^{\,T} = \vec{p}^{\,T}W \ \Leftrightarrow\ \vec{p}\,W^T\vec{p}$

The power method
is the simplest iterative eigensolver. Just multiply the vector many times with the matrix. Algorithm:

(1) start with vector $y = z$, the initial guess
(2) for $k = 1, 2, \ldots$
(3) $\quad v = y/\|y\|_2$
(4) $\quad y = Av$
(5) $\quad \theta = v^*y$
(6) $\quad$ if $\|y-\theta v\|_2 \le \epsilon_M|\theta|$, stop
(7) end for
(8) accept $\lambda = \theta$ and $x = v$

Variants of page rank
The page rank matrices actually used are a bit more complicated:
- The "surfer" gets bored after a while:
  - with probability d the surfer follows a link
  - with probability 1-d the surfer randomly jumps to a new page
- This will raise the weight of not so popular pages.
- It means that all the zeroes get replaced by a small finite probability (1-d)/N

Making all zero entries finite makes the matrix dense:
$M = \mathrm{ConstantMatrix}\left((1-d)/N\right) + d\,W^T$

A better way is to incorporate it explicitly in the multiplication function:
$\vec{p}\,' = M\,\vec{p} = \mathrm{ConstantMatrix}\left((1-d)/N\right)\vec{p} + d\,W^T\vec{p}$
$= \mathrm{ConstantVector}\left((1-d)/N\right) + d\,W^T\vec{p}$
→ we perform a sparse matrix-vector product and add a constant vector

---

13.6 Sparse Matrix Problems [9, 17]

Many problems in CSE can be mapped to sparse matrix problems:
- Explicit time integrators: $\vec{f}(t+\Delta t) = M\,\vec{f}(t)$ (e.g. diffusion equation)
- Implicit time integrators: $M\,\vec{f}(t+\Delta t) = \vec{f}(t)$
- Solving PDEs: $M\,\vec{\phi} = 4\pi G\vec{\rho}$
- Sparse eigenproblems: $W^T\vec{p} = \lambda\,\vec{p}$ (e.g. equilibrium of diffusion, page rank)

13.7 Sparse Matrix Storage [9, 18-21]

Matrix-free:
just code the matrix-vector multiplication instead of storing the matrix

Packed band matrices with u upper and l lower subdiagonals:
store the diagonals only. a_{ij} stored in packed format in $p_{u+l+i-j,j}$

$$\begin{pmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ a_{31} & a_{32} & a_{33} & a_{34} & \\ & & a_{53} & a_{54} & a_{55} \end{pmatrix} \xrightarrow{\text{packed}} \begin{pmatrix} * & a_{12} & a_{23} & a_{34} & a_{45} \\ a_{11} & a_{22} & a_{33} & a_{44} & a_{55} \\ a_{21} & a_{32} & a_{43} & a_{54} & * \\ a_{31} & a_{42} & a_{53} & * & * \end{pmatrix}$$

Compressed storage formats
Dictionary of keys (DOK):
- an associative array mapping an index pair (i,j) to a value
- stored as a tree or hash map of non-zero values
- fast for iteratively building a matrix, slow access later

List of lists (LIL):
- stores one list per row, containing column index and value of the non-zero entries
- the "list" can be a linked list, an array or a vector, sorted by column index
- fast for iteratively building a matrix

Coordinate list (COO):
- a list of triples (column, row, value), sorted by column and row
- fast for iteratively building a matrix, slow access later

Compressed sparse row (CSR):
- stores the matrix in three arrays: column indices, values and row starts

	0	1	2	3	4
0	0	d	0	0	0
1	b	0	0	c	0
2	0	0	a	0	0
3	0	h	0	e	0
4	0	f	0	0	g

col_indices

0	1	2	3	4	5	6	7
1	0	3	2	1	3	1	4

data

d	b	c	a	h	e	f	g

row_starts

0	1	3	4	6	8
0	1	2	3	4	

Compressed sparse column (CSC): similar, but row indices and column starts

It makes sense to change matrix storage format to CSR or CSC after building the matrix if another format is used for efficient construction.

14. FOURIER SERIES

14.1 Fourier Series [14, 2-4]

For a continuous 2π periodic function $f(x)$:

$$f(x)=\sum_{k=-\infty}^{\infty}\hat{f}_k e^{ikx} \quad (-\pi\le x\le\pi) \quad \text{where} \quad \hat{f}_k=\frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-ikx}\,dx$$

Properties of Fourier Series:

- Orthogonal: allows us to easily solve for the Fourier coefficients
- Complete: able to represent all functions
- Computationally convenient: can compute the FS quickly using DFT and FFT
- Easy to compute derivatives in Fourier space

Derivatives

$$f'(x)=\sum_{k=-\infty}^{\infty} ik\hat{f}_k e^{ikx} \qquad f''(x)=\sum_{k=-\infty}^{\infty} -k^2\hat{f}_k e^{ikx}$$

14.2 FTs + PDEs [14, 8-17]

Example 1: Collocation for non-linear advection → Slides 8-10

Example 2: Diffusion equation → Slides 11-13

Example 3: Poisson equation → Slides 14-17

14.3 Discrete Fourier Transform (DFT) [14, 18-23]

The DFT allows a Fourier representation (with finite wave numbers) of a function only given on a finite number of grid points.

For a discretised 2π periodic function f_j defined at N points:

$$x_j=jh \quad \text{where} \quad h=\frac{2\pi}{N} \text{ and } j=0,1,\ldots,N-1$$

DFT: $f_j=\sum_{k=0}^{N-1}\hat{f}_k e^{ikx_j} \quad (j=0,\ldots,N-1)$

Discrete Orthogonality Property

$$\sum_{j=0}^{N-1} e^{ikx_j}e^{-ik'x_j}=\begin{cases}N & \text{if } k=k'+mN\\ 0 & \text{otherwise}\end{cases}$$

Computing Fourier coefficients

$$\hat{f}_k=\frac{1}{N}\sum_{j=0}^{N-1} f_j e^{-ikx_j} \quad \text{for } k=0,1,\ldots,N-1$$

For a discretised 2π periodic function f_j defined at N points:

$$f_j=\sum_{k=0}^{N-1}\hat{f}_k e^{ikx_j}=\sum_{k=0}^{N-1}\hat{f}_k e^{ik\frac{2\pi}{N}j}=\sum_{k=0}^{N-1}\hat{f}_k\omega^{jk}$$

where $\omega=e^{i\frac{2\pi}{N}} \quad j=0,1,\ldots,N-1$

Then the system to solve is:

$$\begin{bmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & \omega & \omega^2 & \cdots & \omega^{(N-1)}\\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(N-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \omega^{(N-1)} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)^2}\end{bmatrix}\begin{bmatrix}\hat{f}_0\\ \hat{f}_1\\ \vdots\\ \hat{f}_{N-1}\end{bmatrix}=\begin{bmatrix}f_0\\ f_1\\ \vdots\\ f_{N-1}\end{bmatrix}$$

$$A_N \qquad \hat{f} \quad = \quad \vec{f}$$

Conjugate and Inverse

$$\bar{A}_N=\begin{bmatrix}1 & 1 & 1 & \cdots & 1\\ 1 & \bar{\omega} & \bar{\omega}^2 & \cdots & \bar{\omega}^{(N-1)}\\ 1 & \bar{\omega}^2 & \bar{\omega}^4 & \cdots & \bar{\omega}^{2(N-1)}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & \bar{\omega}^{(N-1)} & \bar{\omega}^{2(N-1)} & \cdots & \bar{\omega}^{(N-1)^2}\end{bmatrix}$$

Using the Discrete Orthogonality Property: $\bar{A}_N A_N=N\,I$ and $A_N^{-1}=\frac{1}{N}\bar{A}_N$

So $\hat{f}=\frac{1}{N}\bar{A}_N\vec{f}$ and $\vec{f}=A_N\hat{f}$.

14.4 Fast Fourier Transform (FFT) [14, 24-37]

Example: Solving elliptic equations

Poisson equation: $\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}=Q(x,y) \qquad \phi=0 \text{ on } \partial\Omega$

Discretisation:

$$\frac{\phi_{i+1,j}-2\phi_{i,j}+\phi_{i-1,j}}{\Delta x^2}+\frac{\phi_{i,j+1}-2\phi_{i,j}+\phi_{i,j-1}}{\Delta y^2}=Q_{i,j} \qquad \begin{matrix}i=1,2,\ldots,M-1\\ j=1,2,\ldots,N-1\end{matrix}$$

Ansatz for the solution: $\phi_{i,j}=\sum_{k=1}^{M-1}\hat{\phi}_{kj}\sin\left(\frac{i\pi k}{M}\right)$ (using sine due to boun. cond.)

Expansion of the RHS: $Q_{i,j}=\sum_{k=1}^{M-1}\hat{Q}_{kj}\sin\left(\frac{i\pi k}{M}\right)$

Plug into the discretisation, simplify and equate coefficients:

$$\hat{\phi}_{k,j+1}+\left(\frac{\Delta y^2}{\Delta x^2}\left(2\cos\frac{\pi k}{M}-2\right)-2\right)\hat{\phi}_{k,j}+\hat{\phi}_{k,j-1}=\Delta y^2\hat{Q}_{k,j} \quad \text{for each } k$$

Algorithm:

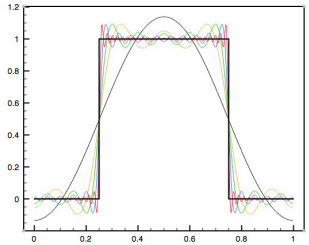
- For each $j=1,2,\ldots,N-1$ the RHS $Q_{i,j}$ is sine transformed to obtain
  $\hat{Q}_{k,j}$: $\hat{Q}_{k,j}=\frac{2}{M}\sum_{i=1}^{M-1}Q_{i,j}\sin\left(\frac{\pi k i}{M}\right) \qquad \begin{matrix}k=1,2,\ldots M-1\\ j=1,2,\ldots N-1\end{matrix}$
- Solve the tridiagonal system for each k:
  $\hat{\phi}_{k,j+1}+\left(\frac{\Delta y^2}{\Delta x^2}\left(2\cos\frac{\pi k}{M}-2\right)-2\right)\hat{\phi}_{k,j}+\hat{\phi}_{k,j-1}=\Delta y^2\hat{Q}_{k,j}$
- Get the $\phi_{i,j}$ from $\hat{\phi}_{k,j}$ using the inverse sine transform

Alternative: two dimensional DFT
→ Slides 33-34

Weaknesses of Fourier methods

- Periodic problems: boundary conditions and functions
- Only suitable for ODEs and PDEs with linear coefficients
- Gibb's phenomenon for discontinuous functions
- Aliasing for the DFT

Gibb's Phenomenon
Finite sum of the Fourier series gives an overshoot at the discontinuity.



15. RANDOM NUMBERS

Pseudo Random numbers are generated by an algorithm.

15.1 Linear Congruential Generators [2a, 29]

- are of the simple form $x_{n+1}=f(x_n)$
- good choice is the GGL generator: $x_{n+1}=(a x_n+c) \bmod m$ with a=16807, c=0, $m=2^{31}-1$
- quality depends sensitively on a, c, m
- the sequence repeats identically after $2^{31}-1$ iterations (with 500 mio. numbers/s that is just 4 s)

15.2 Lagged Fibonacci Generators (LFG) [2a, 30-33]

$x_n = x_{n-p} * x_{n-q} \bmod M$
Good choices for (p,q): (2281,1252), (9689,5502), (44497,23463)
$M=2^{32}$ or $M=2^{31}-1$ and addition or xor as operation

With $M=2^m$, LFG has a maximum cycle length of $(2^q-1)2^{m-1}$.
Theoretically possible cycle length: $2^{mq}-1$
There exist $2^{(q-1)(m-1)}$ cycles of maximal length.

16. NON-UNIFORM RANDOM NUMBERS

Given random numbers u in the interval [0,1[, you get a uniform distribution x in [a,b[with $x=a+(b-a)u$.

16.1 Non-uniform Distributions [2a, 38]

To get a random number x distributed with $f(x)$ in the interval [a,b[from a uniform random number u, calculate

$$F(y)=\int_a^y f(x)dx \ , \quad x=F^{-1}(u)$$

16.2 Normal Distribution [2a, 39]

$$f(x)=\frac{1}{\sqrt{2\pi}}\exp\left(-x^2\right)$$

Not easy to integrate in one dimension, but in two (Box-Muller):
$n_1=\sqrt{-2\ln(1-u_1)}\sin(2\pi u_2)$
$n_2=\sqrt{-2\ln(1-u_1)}\cos(2\pi u_2)$

17. LU DECOMPOSITION

16.1 Gaussian Elimination / LU Factorisation [10, 26-35]

Given: A Find: L, U, P, such that $PA = LR$, $A = P'LR$

Example

$$A=\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \qquad P=\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad L=[\,]$$

Swap largest coefficient of x to the first row:

$$U=\begin{bmatrix} \mathbf{-3} & -1 & 2 \\ \mathbf{2} & 1 & -1 \\ \mathbf{-2} & 1 & 2 \end{bmatrix} \qquad P=\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad L=[\,]$$

Compute $II=II-\frac{\mathbf{2}}{\mathbf{-3}}I$ and $III=III-\frac{\mathbf{-2}}{\mathbf{-3}}I$:

$$U=\begin{bmatrix} -3 & -1 & 2 \\ 0 & 1/3 & 1/3 \\ 0 & 5/3 & 2/3 \end{bmatrix} \qquad P=\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad L=\begin{bmatrix} 0 \\ -2/3 \\ 2/3 \end{bmatrix}$$

Swap largest coefficient of y to the second row:

$$U=\begin{bmatrix} -3 & -1 & 2 \\ 0 & 5/3 & 2/3 \\ 0 & 1/3 & 1/3 \end{bmatrix} \qquad P=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \qquad L=\begin{bmatrix} 0 \\ 2/3 \\ -2/3 \end{bmatrix}$$

Compute $III = III-\frac{1/3}{5/3}II = III-\frac{1}{5}II$:

$$U=\begin{bmatrix} -3 & -1 & 2 \\ 0 & 5/3 & 2/3 \\ 0 & 0 & 1/5 \end{bmatrix} \qquad P=\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \qquad L=\begin{bmatrix} 0 & 0 \\ 2/3 & 0 \\ -2/3 & 1/5 \end{bmatrix}$$

Expand the coefficient matrix L to a lower triangular matrix:

$$L=\begin{bmatrix} 1 & 0 & 0 \\ 2/3 & 1 & 0 \\ -2/3 & 1/5 & 1 \end{bmatrix}$$