Heidelberg Institute for Theoretical Studies
Schloss-Wolfsbrunnenweg 35
69118 Heidelberg

Interdisciplinary Center for Scientific Computing
Heidelberg University
INF 205
69120 Heidelberg

Zentrum für Astronomie, Universität Heidelberg
Astronomisches Recheninstitut
Mönchhofstr. 12-14
69120 Heidelberg

WS2020/21

Lecture Notes for MVComp1

Fundamentals of Simulation Methods

Prof. Dr. Volker Springel, extended by Prof. Dr. Frauke Gräter

November 2, 2020

Contents

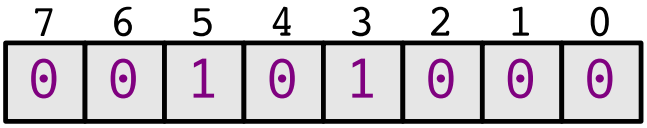
1 Issues of Floating Point Math

Numerical calculations on standard computers can sometimes lead to unexpected results due to complications such as overflow or round-off errors. It is important to be aware of the different things that can go wrong, and to adopt suitable precautions where possible. In this introductory section we recapitulate how standard computer languages handle arithmetic operations in order to get a more precise understanding of these issues.

1.1 Integer arithmetic

- Integers are whole numbers that computers represent in binary.
- The representable range depends on the available storage size of the chosen variable type.
- Negative numbers are represented as so-called two's complement.

The basic unit is the byte with 8 bits. We can number the bits from 0 (the 'least significant bit', or LSB) to 7 (the most significant bit, MSB). Example:



This number has the value

$$2^5 + 2^3 = 32 + 8 = 40. \tag{1.1}$$

The C-types for single bytes are:

type	range of values
`unsigned char`	$\{0, \ldots, 255\}$
`(signed) char`	$\{-128, -127, \ldots, 0, 1, \ldots, 127\}$

1.1.1 Two's complement for negative numbers:

- The MSB bit flags negative numbers.

- To change the sign of an integer number, one needs two carry out the following steps:
  1. invert all bits
  2. add +1 to the number

Example: Find the binary representation of -9.
start with +9: `00001001`
invert all bits: `11110110`
add +1: `11110111` that's it!
With the two's complement, no special treatment is necessary when adding negative numbers to positive numbers.

1.1.2 Common integer types in C

C-name	bits	range		
`char`	8	-128	...	$+127$
`short`	16	-32768	...	$+32767$
`int`	32	-2147463648	...	$+2147463647$
`long long`	64	-9.2×10^{18}	...	$+9.2 \times 10^{18}$
	n	-2^{n-1}	...	$+2^{n-1} - 1$

One can also use unsigned versions of these types (by adding `unsigned` in front), but it is recommended to do this only in exceptional cases because of the danger of nasty surprises in case a negative number is attempted to be stored in such a type.

1.1.3 Things to watch out for in integer arithmetic

The largest danger is **overflow**. Here is an example:

```
char a, b, c;
a = 100;
b = 5;
c = a * b;
```

When one tries to print out the variable c, one gets: -12 !
Why? `a * b = 500` has binary representation

```
....000111110100
```

This will now be truncated to just the 8 least significant bits, because the result is stored in a variable of type char. So one gets:

```
11110100
```

Because the most significant bit is set, this will be interpreted as a negative number. We can apply the two's complement to flip the sign and obtain the absolute value of the number. After the two's complement, we have

```
00001100
```

This corresponds to the value +12.

So when using integer variables, always be aware of their finite range. In the C-language, the type `int` is often sufficient as universal counting and indexing variable. But increasingly often, its range of about 2 billion may not be enough any more for large simulations. In this case, the use of 64-bit integer variables should be considered. (But using this for everything will cost performance and typically result in a waste of storage space.)

Another issue where some caution is in order is **integer division**. In most languages, this is defined to always truncate all dezimal places, i.e.

```
 8 /  3 =  2

-8 /  3 = -2

 8 / -3 = -2
```

Also watch out for the definition of the **modulus** operation in your language of choice. In C, the syntax for `8 modulus 3` is `8 % 3`, and the result is defined as `n modulus m = n - (n/m)*m`. Hence:

```
 8 %  3 =  2

-8 %  3 = -2

 8 % -3 =  2
```

Finally, another thing to watch out for are **implicit type conversions**. Suppose we have

```
char a = 85;
char b = 5;
int  c = a * b;
```

When we check the value of c we get the correct result 425, and not -87 that we would get due to overflow if the type of c would also be `char`. This happens because C does an automatic, implicit promotion of a and b to `signed int`, and only then carries out the arithmetic operation. But now contrast this with:

```
       int  a = 2000000000;
       int  b = 3;
long long  c = a * b;
```

When we now check the value of c, we get `1705032704`, which is equal to $6 \times 10^9 - 2^{32}$. Apparently, here our operation has overflown and was not rescued by implicit type conversion, because C will at most use `int` for this. To get the correct result, we have to force a type conversion ourselves, for example in the following form:

```
long long  c = a * ((long long)b);
```

This now yields the correct result of `6000000000`.

1.2 Floating point arithmetic

In general, floating point representations have a

- base β
- precision p (the 'number of digits')
- exponent e

For example, if $\beta = 10$ and $p = 4$, the representations of the two numbers 0.1 and 523 are

number	representation ($\beta = 10$, $p = 4$)
0.1	1.000×10^{-1}
523	5.230×10^{2}

Nowadays, floating point calculations on computers are most commonly done according to the IEEE-754 standard, using a binary base $\beta = 2$. In particular, single precision numbers use $\beta = 2$ and $p = 23(+1)$. This gives the representations

number	representation ($\beta = 2$)
0.1	$1.10011001100110011001101 \times 2^{-4}$
523	$1.00000101100000000000000 \times 2^{9}$

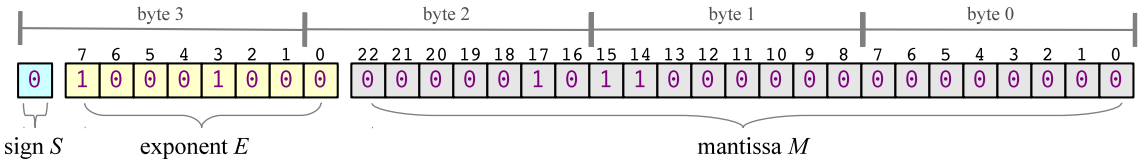
The first encoding has the value $(2^0 + 2^{-1} + 2^{-4} + 2^{-5} + 2^{-8} + \ldots) \times 2^{-4} \simeq 0.0996$, which differs from 0.1. In fact, 0.1 cannot be represented exactly in the binary format.

Some further notes:

- The representation is not unique – by shifting the exponent, one can arrive at other representations. This is addressed by *normalization*. Every number is stored as convention in the standard such that the leading digit before the dot is 1.
- If this is the case, then the "1" doesn't have to be stored anymore (since it's always there anyway), and one gains a bit of significance if this is exploited. Hence p is really 24 for single precision IEEE-754 numbers. However, representing zero requires a special solution in this case.

- Some numbers cannot be represented exactly – such as the 0.1 from the example.
- The IEEE standard defines a smallest and largest exponent. For single precision numbers this is $e_{\min} = -126$ and $e_{\max} = +127$. This restricts the range of representable numbers, i.e. there is a minimum and a maximum value that is possible, and values beyond this will cause an over- or underflow.

The storage scheme for single-precision IEEE-754 numbers is as follows:



The exponent is stored here in the 8 bits reserved for it not as a two-complement, but rather in a *biased* way, defined through

$$E = e + 127, \tag{1.2}$$

where E is stored as an unsigned byte. Because the range for e is restricted for regular floating point numbers, we have for them $1 \leq E \leq 254$. The values $E = 0$ and $E = 255$ fulfil a special purpose, see below.

The bits are set for the example of encoding the number 523 as a single precision floating point number. In binary representation, 523 can be written in normalized form as

$$1.000001011 \times 2^9 \tag{1.3}$$

The exponent is hence $e = 9$, such that $E = 136$. Because one of the exponent bits is moved to byte 2 in the encoding, byte 3 will then have the value 68. Byte 2 has the value 2, byte 1 the value 192, and byte 0 the value 0.

On little Endian computers (Intel, AMD chips), these bytes will be stored in memory from right to left, with the least significant byte first, i.e. in the sequence 0, 192, 2, 68. On big Endian machines (e.g. PowerPC architecture), the sequence is reversed.

In general, the value f of a IEEE floating point number is

$$f = (-1)^s \cdot \left(1 + \frac{M}{2^p}\right) \times 2^{E-b}, \tag{1.4}$$

where here s is the sign bit, M the integer number representing the mantissa, and E is the exponent. For single precision, we have $p = 23$ and the bias is $b = 127$.

The exponent values $E = 0$ and $E = 255$ are used to represent special values:

1. $E = 0$, $M = 0$: This is zero. Actually, one can have ± 0, depending on the sign bit.

2. $E = 255$, $M = 0$: By convention, this represents $\pm\infty$, depending on the sign bit.

3. $E = 255$, $M \neq 0$: This signals "not a number" (`NaN`), which can result as part of an invalid mathematical operation (division by 0, or square root of a negative number).

4. $E = 0$, $M \neq 0$: These are so-called *denormalized numbers*, because here no leading 1 in front of the dot can be assumed. The value of these numbers is given by

$$f = (-1)^s \cdot \frac{M}{2^p} \times 2^{-b+1} \tag{1.5}$$

Machine precision

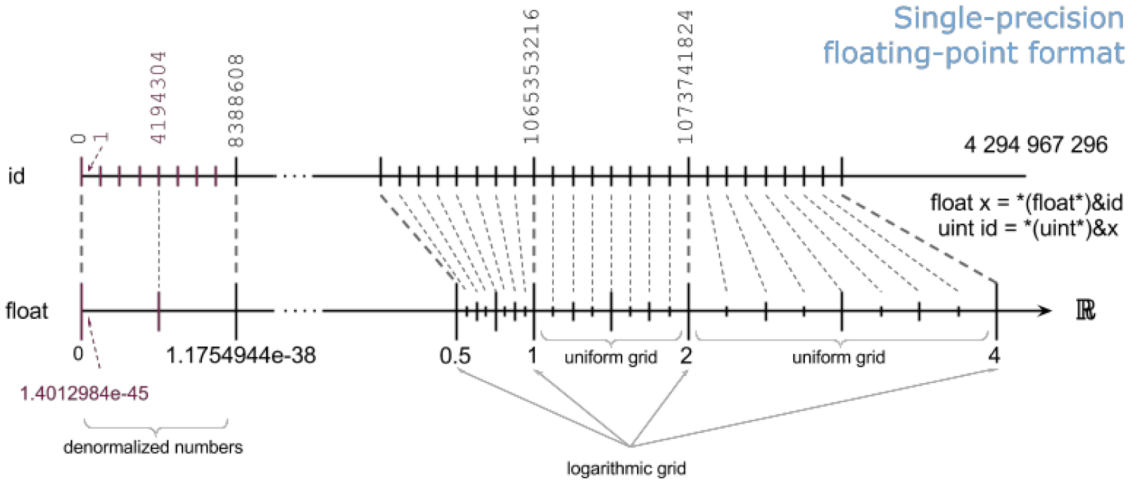
The smallest increment in the mantissa,

$$\epsilon_m = \frac{1}{2^p}. \tag{1.6}$$

is called machine precision. This can be coarsely interpreted as the smallest relative spacing between two floating point numbers that can still be distinguished by the representation scheme. For single precision this is $\epsilon_m \sim 1.19 \times 10^{-7}$.

1.2.1 The set of representable numbers

The set of floating point numbers is finite and subdivides the reals in a special kind of logarithmic grid. (figure below by Denis Yurin)



The largest representable single precision number is

$$f_{\max} = \left(1 + \frac{2^p - 1}{2^p}\right) \times 2^{127} \simeq 3.403 \times 10^{38}, \tag{1.7}$$

while the smallest positive normalized number is

$$f_{\min} = (1+0) \times 2^{-126} \simeq 1.175 \times 10^{-38}. \tag{1.8}$$

But actually, thanks to denormalized numbers, the smallest possible single precision value is

$$f_{\text{smallest}} = \frac{1}{2^p} \times 2^{-126} \simeq 1.4 \times 10^{-45}. \tag{1.9}$$

Due to the introduction of denormalized numbers, IEEE-754 guarantees that for $x \neq y$ one always has $x - y \neq 0$. This is meant to protect, for example, against the common bug-prone idiom:

```
if  x != y  then z = 1.0 / (x-y)
```

Notes on floating point arithmetic

- Any arithmetic operation's result is mapped to one of the numbers in the *finite* set of representable floating point numbers. This is called *rounding*. In total, there are about 4 billion different single precision numbers.

- The IEEE standard requires that the result of addition, subtraction, multiplication, and division is rounded exactly. This means that the result is as if the calculation was done exactly, and is then rounded to the *nearest* representable number.

- Any operation involving NaN will again yield NaN.

Some consequences and pitfalls

- It is possible that the result of $a + b$ is identical to a for $b \neq 0$. (In fact, this will typically happen when $|b| < \epsilon_m \cdot |a|$.)

- The law of associativity is not guaranteed (due to rounding errors). This means that $(a + b) + c$ does not necessarily evaluate to the same number as $a + (b + c)$.

- Even though $x/2.0$ and $0.5 * x$ are always the same, this is not true for $x/10.0$ and $0.1 * x$. This is because 0.1 is not exactly representable for base $\beta = 2$. Also, beware that optimizing compilers may change $x/10.0$ to $0.1 * x$ (since multiplication is much faster than division), but this can then change the result of the calculation and the semantics of the program.

- When numbers nearly cancel, a loss of precision (reduced number of significant digits) results. For example, consider $x = 10^8$, $y = 10^5$ and $z = -1 - 10^5$. Then:

```
x * y + x * z =  -1.0066 x 10^8

x * (y + z)   =  -1.0 x 10^8
```

Here the second result is correct, but the first one is 0.6% off.

- Comparing floating point numbers exactly is not a good idea as at least one of the two might not represented exactly as a float. A simple fix is to use a relative error that still is allowed.

Closer look at loss of precision

Let us more formally define the number of significant digits. Assume $x^\star$ is our floating point approximation to a number x, and likewise $y^\star$ for y. We would then call

$$\frac{x^\star - x}{x} \tag{1.10}$$

the relative error of the representation. One says that $x^\star$ approximates x to r significant digits if

$$|x^\star - x| < \frac{1}{2}10^{s-r+1}, \tag{1.11}$$

where s is the largest integer such that $10^s < |x|$ (i.e. the absolute error is at most 0.5 in the r-th significant digit of x). For example, $x^\star = 22/7 = 3.1428\ldots$ approximates π to three significant digits.

Cancellation of large numbers typically leads to a loss of significant digits and an explosion of the relative error. Example:

$$x^\star = 0.76545421 \tag{1.12}$$

$$y^\star = 0.76544200 \tag{1.13}$$

Both numbers are stored with 7 significant digits. But the difference

$$z^\star = x^\star - y^\star = 0.12210000 \times 10^{-4} \tag{1.14}$$

has only 3 significant digits in its approximation of z. Consequently, the relative error has become larger by a factor of 1000 or so!

The lesson is, if cancellation can be foreseen *avoid it if possible*. For example, the evaluation of

$$f(x) = 1 - \cos(x) \tag{1.15}$$

involves a cancellation for x near zero. Calculating instead

$$f(x) = \frac{\sin^2(x)}{1 + \cos(x)} \tag{1.16}$$

will yield a more accurate (but potentially more costly) result in this case.

1.2.2 Double and higher precision

Standard compliant double precision numbers use $p = 52(+1)$ bits for the mantissa, and 11 bits for the exponent which has the allowed range $\epsilon_{\max} = +1023$ and $\epsilon_{\min} = -1022$. (Btw: $|\epsilon_{\min}| < |\epsilon_{\max}|$ is adopted such that the inverse of the smallest representable doesn't overflow.) The storage footprint of a double is hence 64 bits, or 8 bytes. The largest and smallest positive representable numbers are

$$f_{\max} \simeq 1.8 \times 10^{308}, \tag{1.17}$$

$$f_{\min} \simeq 2.2 \times 10^{-308}. \tag{1.18}$$

And the machine precision is

$$\epsilon_m \simeq 2^{-52} = 2.2 \times 10^{-16}. \tag{1.19}$$

- Double precision has all the same principle pitfalls as single precision – but "much less".
- Recommendation: Use always double precision unless memory constraints force the use of single precision for some reason.
- But: Don't believe the use of double precision protects against inaccurate floating point results in all situations!

Quad-double precision

- This is a 128-bit floating point format in the IEEE standard based on a 16 byte presentation.
- It offers twice as many significant digits as plain double precision ($\sim$ 34 dezimal places) and an extended exponent range.
- Unfortunately, this is typically not supported in hardware by current processors, but it can be emulated in software by some compilers, in which case every floating point operation is between 2-10 times slower than a corresponding double precision operation.
- Note that `long double` will be accepted by C-compilers, but in many cases this will either be simply identical to 64-bit double precision values, or it will yield a 96-bit format. Sometimes, the software emulation of 128-bit accuracy can however be enabled through compiler flags.

Arbitrary precision

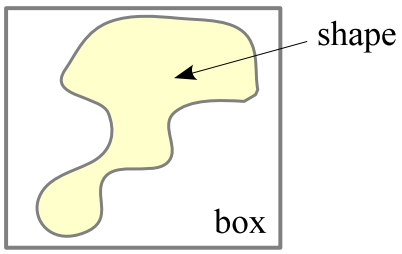
There are good floating point libraries that can be used for calculations in (nearly) arbitrary, user-defined precision. One very capable package is GMP, the 'GNU big number library' (`https://gmplib.org`).

2 Monte Carlo Techniques

The lecture will cover both stochastic and deterministic simulation methods. We start out with the stochastic branch, Monte Carlo techniques, which use random numbers to numerically explore the problem at hand. Questions that are handled by Monte Carlo methods range from protein folding and radiation hydrodynamics to financial markets and disease spread.

2.1 Monte Carlo Integration

So-called Monte Carlo integrations lie at the heart of many stochastic simulation methods. The basic idea can be intuitively understood with the "dartboard method" of integrating the area of an irregular domain.



This works as follows:

- Choose points randomly (i.e. uniformly) within the box.
- We know that the probability that a point hits inside the area is proportional to the ratio of the areas:

$$p(\text{dart hits inside area}) = \frac{A_{\text{shape}}}{A_{\text{box}}} \tag{2.1}$$

- We can now approximate this probability, and hence the area ratio, through the experimental result:

$$\frac{A_{\text{shape}}}{A_{\text{box}}} \simeq \frac{\#\text{hits in shape}}{\#\text{hits in box}} \tag{2.2}$$

  This is expected to become arbitrarily accurate as the number of trials goes to infinity.

Standard Monte Carlo integration

Let us now formalize this technique. We consider an integral in d-dimensions,

$$I = \int_V f(\mathbf{x})\,\mathrm{d}^d\mathbf{x}, \tag{2.3}$$

where V is a d-dimensional hypercube with (for simplicity) dimensions $[0,1]^d$. To compute this as a Monte Carlo integral, we do the following:

1. Generate N random vectors $0 \leq \mathbf{x}_i \leq 1$ with flat distribution (i.e. each component of the vector is drawn independently from a uniform distribution).

2. We compute
$$I_N = \frac{V}{N}\sum_i^N f(\mathbf{x}_i). \tag{2.4}$$
For $N \to \infty$, we then get $I_n \to I$.

3. The error of the result scales as $1/\sqrt{N}$, *independent* of the number of dimensions of the integral.

Especially the last point is quite remarkable – we'll later have to look at this in more detail. Before we do this, it is instructive to compare with the steps taken in standard integration techniques. In them, we divide each dimension in n regularly spaced points. The total number of points is hence $N = n^d$. Depending on the integration rule selected, the error will then scale as some power of $1/n$. For example, for the midpoint and trapezoidal rules, it will simply be $\propto 1/n^2$, and for Simpson's rule $\propto 1/n^4$.

If d is small, it is clear that the Monte Carlo integration has much larger errors than standard methods when the same N is used. However, the higher d becomes, the better Monte Carlos looks because then the standard method can spend comparatively fewer regular sampling points per dimension.

One can then for example ask: At what point is Monte Carlo as good as Simpson? Well, Simpson's error should scale as

$$\frac{1}{n^4} = \frac{1}{N^{4/d}}, \tag{2.5}$$

which starts to decline with N more weakly than Monte Carlo integration when $d \geq 8$. We hence clearly see that high dimensional problems are the regime where Monte Carlo integration becomes particularly interesting.

In fact, in some lattice simulations one has dimensions in the range $d = 10^6 - 10^{10}$. Here the only viable approach is to use Monte Carlo integration. In practice, standard methods fail already at much more moderate numbers of dimensions. For example, even with $d = 10$, putting down just a grid with $n = 10$ cells per dimension already yields a number of $N = 10^{10}$ grid points.

2.2 Error in Monte Carlo integration

Let $y_i = Vf(\mathbf{x}_i)$ be the value of the i-th function evaluation of our Monte Carlo integration. After N samples, we can thus write the approximation to the desired integral as

$$I_N = \frac{y_1 + y_2 + \ldots + y_N}{N} \tag{2.6}$$

The error of this quantity is *defined* as the width of its probability distribution $P_N(I_N)$, i.e.

$$\sigma_N^2 \equiv \left\langle I_N^2 \right\rangle - \left\langle I_N \right\rangle^2 . \tag{2.7}$$

Let's try to calculate this in order to get an idea of the size of this error. First, let's introduce the probability distribution of the y_i, denoted with $p(y)$. For it we have

$$\int p(y)\,\mathrm{d}y = 1, \tag{2.8}$$

$$\langle y \rangle = \int y\, p(y)\,\mathrm{d}y, \tag{2.9}$$

$$\left\langle y^2 \right\rangle = \int y^2\, p(y)\,\mathrm{d}y, \tag{2.10}$$

$$\sigma^2 = \left\langle y^2 \right\rangle - \left\langle y \right\rangle^2 . \tag{2.11}$$

We can then write

$$P_N(I_N) = \int \delta\left(I_N - \sum_i \frac{y_i}{N}\right) p(y_1)\, p(y_2) \cdots p(y_N)\,\mathrm{d}y_1\,\mathrm{d}y_1 \cdots \mathrm{d}y_N, \tag{2.12}$$

where the Dirac delta-function enforces that the average of the y_i is equal to I_N. We now take the Fourier transform of $p(y)$:

$$\hat{p}(k) = \int p(y)\,\mathrm{e}^{ik(y - \langle y \rangle)}\,\mathrm{d}y \tag{2.13}$$

Similarly, let's consider the Fourier transform of $P_N(I_N)$:

$$\begin{aligned}
\hat{P}_N(k) &= \int P_N(I_N)\,\mathrm{e}^{ik(I_N - \langle I_N \rangle)}\,\mathrm{d}I_N && (2.14)\\
&= \int p(y_1) \cdots p(y_N)\,\mathrm{e}^{i\frac{k}{N}(y_1 - \langle y_1 \rangle + y_2 - \langle y_2 \rangle + \ldots + y_N - \langle y_N \rangle)}\mathrm{d}y_1 \cdots \mathrm{d}y_N && (2.15)\\
&= \left[\hat{p}\left(\frac{k}{N}\right)\right]^N . && (2.16)
\end{aligned}$$

Here we made use of $\langle I_N \rangle = \langle y \rangle$. Now we expand $\hat{p}\left(\frac{k}{N}\right)$ in powers of k/N, in the limit of large N. We get

$$\begin{aligned} \hat{p}\left(\frac{k}{N}\right) &= \int p(p)\, \mathrm{e}^{i\frac{k}{N}(y-\langle y\rangle)}\, \mathrm{d}y & (2.17)\\ &= \int p(y)\left[1 + \frac{ik}{N}(y-\langle y\rangle) - \frac{k^2}{2N^2}(y-\langle y\rangle)^2 + \ldots\right] \mathrm{d}y & (2.18)\\ &= 1 - \frac{k^2\sigma^2}{2N^2} + \ldots & (2.19) \end{aligned}$$

Thus we find

$$\hat{P}_N(k) = \left[\hat{p}\left(\frac{k}{N}\right)\right]^N = \left(1 - \frac{k^2\sigma^2}{2N^2}\right)^N \simeq 1 - \frac{k^2\sigma^2}{2N} \simeq \mathrm{e}^{-\frac{k^2\sigma^2}{2N}}, \quad (2.20)$$

because N is very large. With this result in hand, we can now use it to calculate $P_N(I_N)$ through an inverse Fourier transform:

$$\begin{aligned} P_N(I_N) &= \frac{1}{2\pi}\int \mathrm{e}^{-ik(I_N-\langle I_N\rangle)}\, \hat{P}_N(k)\, \mathrm{d}k & (2.21)\\ &= \frac{1}{2\pi}\int \mathrm{e}^{-\frac{k^2\sigma^2}{2N} - ik(I_N-\langle I_N\rangle)}\, \mathrm{d}k & (2.22)\\ &= \frac{1}{2\pi}\mathrm{e}^{-\frac{1}{2}\frac{N}{\sigma^2}(I_N-\langle y\rangle)^2} \int \mathrm{e}^{-\frac{\sigma^2}{2N}\left(k + i(I_N-\langle y\rangle)\frac{N}{\sigma^2}\right)^2} \mathrm{d}k & (2.23) \end{aligned}$$

We now can do this integral by shifting k and recalling

$$\int \exp(-\alpha x^2)\, \mathrm{d}x = \sqrt{\frac{\pi}{\alpha}}, \quad (2.24)$$

obtaining

$$P_N(I_N) = \frac{\sqrt{N}}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2}\frac{N}{\sigma^2}(I_N - \langle y\rangle)^2\right). \quad (2.25)$$

This is a Gaussian with dispersion $\sigma_N = \sigma/\sqrt{N}$, *independent* of the shape of $p(y)$. What we just have derived is the *central limit theorem*! Independent of the detailed shape of $p(y)$, if we average enough of these distributions we will get a Gaussian.

Summary:

For standard Monte Carlo integration with N samples, the error is

$$\sigma_N = V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}} \quad (2.26)$$

where $\langle f\rangle$ and $\langle f^2\rangle$ are exact moments of the function we integrate, i.e.

$$\langle f\rangle \equiv \frac{1}{V}\int f(x)\mathrm{d}x = \frac{1}{V}\int y\, p(y)\mathrm{d}y. \quad (2.27)$$

In practice, we can estimate these moments from the Monte-Carlo samples themselves, i.e. we can estimate

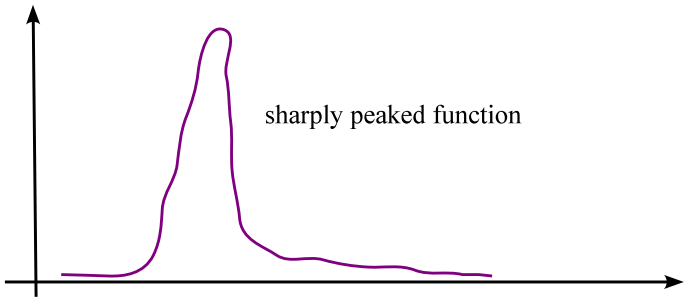
$$\langle f \rangle \simeq \frac{1}{N} \sum_i f(x_i), \tag{2.28}$$

$$\langle f^2 \rangle \simeq \frac{1}{N} \sum_i f^2(x_i), \tag{2.29}$$

and then use these moments to estimate the error.

2.3 Importance Sampling

One common problem in Monte Carlo integration is that often the integrand is very small on a dominant fraction of the integration volume. For example, if the integrand is sharply peaked, only points sampled close to the peak will give a significant contribution.



The idea of **importance sampling** is to choose the random points somehow preferentially around the peak, and putting less points where the integrand is small. This should be more efficient and help to reduce the error for a given number of points.

So let us assume we want to integrate

$$I = \int_V f(x)\,\mathrm{d}x, \tag{2.30}$$

and suppose we choose a distribution $p(x)$ which is close to the function $f(x)$, but which is simple enough so that it is possible to generate x-values from this distribution. We can then write:

$$I = \int_V p(x) \frac{f(x)}{p(x)}\,\mathrm{d}x. \tag{2.31}$$

Thus, if we sample points around a point x with probability $\mathrm{d}p = p(x)\,\mathrm{d}x$ (which is exactly the definition of sampling from the distribution $p(x)$), we simply obtain:

$$I = \lim_{N\to\infty} \frac{1}{N} \sum_i \frac{f(x_i)}{p(x_i)}. \tag{2.32}$$

Because f/p is flatter than f if the shape of p is similar to that of f, the variance of f/p will be smaller than the variance of f, i.e. we obtain a smaller error for given N. The ideal choice is $p(x) \propto f(x)$. This is often not possible in practice, but in fact possible in lattice Monte Carlo simulations, as we will see.

Example for importance sampling

Let's consider a simple 1D integration to demonstrate the concept of importance sampling. The integral we want to compute is:

$$I = \int_0^1 \left(x^{-1/3} + \frac{x}{10} \right) \mathrm{d}x. \tag{2.33}$$

This can be solved analytically and has the value $I = 31/20 \simeq 1.55$, so we don't really need Monte Carlo integration here, but for the sake of demonstrating the method we apply it anyway.

Doing this integral with standard Monte Carlo integration gives the error

$$\sigma_N = \sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}} \simeq \frac{0.85}{\sqrt{N}}. \tag{2.34}$$

Let's now try to do it with importance sampling, using the sampling probability

$$p(x) = \frac{2}{3} x^{-1/3} \tag{2.35}$$

over the interval $0 < x \leq 1$, based on the realization that $p(x)$ captures part of the shape of $f(x)$, while being simple enough to allow a creation of properly sampled points by direct inversion (see below). The new function to integrate is then $g = f/p$, and the width of the corresponding Monte Carlo error distribution function becomes

$$\sigma_N = \sqrt{\frac{\langle g^2 \rangle - \langle g \rangle^2}{N}} \simeq \frac{0.045}{\sqrt{N}}. \tag{2.36}$$

This is nearly 20 times better than obtained with plain sampling, hence a substantial gain in efficiency has been reached.

2.4 Random number generation

Good random number obviously play a central role in Monte Carlo techniques.

- Usually they are produced by *deterministic algorithms* leading to *pseudorandom numbers*.

- Such pseudorandom numbers need to be distinguished from "truly random" numbers generated by some physical process (like rolling the dice, radioactive decay, quantum transitions, etc.). Some modern CPUs include a hardware random number generator, based for example on coupled non-linear oscillators and additional sources of entropy. However, these generators are normally not used for Monte Carlo techniques, because
  - the sequence is not repeatable, making debugging difficult and preventing exact reproducibility
  - the generators are often slow
  - the quality of the distribution may not be perfect
  - the quality of the distribution may degrade with time, or correlate in subtle ways with environmental factors such as operating temperature, etc.
- There is value in having good random numbers. In 1950, the RAND corporation published a book entitled "1 million random digits", whose primary virtue is to contain no discernable information at all. This classic is available online (`http://www.rand.org/publications/classics/randomdigits`).

2.4.1 Pseudo-random numbers

Here we usually create an integer sequence that is then converted to a floating point number in the interval $[0, 1[$. The essential desirable properties of a good random number generator are:

- Repeatability: For the same seed, we want to obtain the same sequence of random numbers.
- Randomness: Good random numbers should
  - be uniformly and homogeneously distributed in the interval $[0, 1[$.
  - be independent of each other, i.e. show no correlations whatsoever (this is difficult and not true exactly for pseudo-random number generators)
- Speed: In modern applications, we may need billions of random numbers.
- Portability: We want the same results on different computer architectures.
- Long period: After a finite number of pseudo-random numbers, the sequence repeats. This period should be as large as possible.
- Insensitivity to seed: Neither the period nor the quality of the randomness should depend on the value of the seed, i.e. on where the sequence is started.

Linear congruential generators

The simplest pseudo-random number generators work with an integer mapping of the form

$$X_{i+1} = (aX_i + b) \mod m, \tag{2.37}$$

where a, b, and m are integers. The numbers X_i lie then in the range $[0, m-1]$ and can be mapped to floating point numbers in the unit interval by dividing with m. It is clear that such a generator can have a period of at most m. Examples for such *linear congruential generators* include:

- **ANSI-C:**

  $$a = 1103515245 \quad b = 12345 \quad m = 2^{31} \tag{2.38}$$

  The period here is quite small, just $m = 2^{31} \sim 2 \times 10^9$, which is quickly reached in modern computers. This is not good enough for serious Monte Carlo applications.

- **RAND generator in Matlab:**

  $$a = 16807 \quad b = 0 \quad m = 2^{31} - 1 \tag{2.39}$$

  Again, this has an uncomfortably short period.

- **UNIX drand48():**

  $$a = 25214903917 \quad b = 11 \quad m = 2^{48} \tag{2.40}$$

  This is starting to be somewhat usable, with a period of $2^{48} \simeq 2.8 \times 10^{14}$. Note however that the low order bits have shorter cycling times and show less randomness than they should, which is a common problem for all simple linear congruential random number generators.

- **NAG-generator:**

  $$a = 13^{13} \quad b = 0 \quad m = 2^{59} \tag{2.41}$$

  This has a very long period, but the low order bits are still not very good.

To get better random numbers, one needs to go to more complicated schemes than a simple integer mapping. One approach is to combine two or several linear congruential mappings. This is done for example in **ran2** of Numerical Recipes. This uses

$$X_{i+1} = (40014X_i) \mod 2147483563 \tag{2.42}$$

$$Y_{i+1} = (40692Y_i) \mod 2147483399 \tag{2.43}$$

$$Z_{i+1} = (X_i + Y_i) \mod 2147483563 \tag{2.44}$$

The Z_i are then mapped to floating point numbers. Here the period is extended to $\sim 10^{18}$.

Lagged Fibonacci generators

A refinement of this approach consists of using several integers to define the internal state of the generator. One then uses a prescription of the form

$$X_i = (X_{i-p} \odot X_{i-q}) \mod m \tag{2.45}$$

to create new integers, where p and q are the 'lags' (offsets to other past numbers), and $\odot$ is some arithmetic operation, for example addition, multiplication, etc., or also bitwise logical operations such as XOR. For large lags, the quality of these generators becomes very good.

For example, **RANLUX** is of this type, reaching a period 10^{171}. Another modern generator using this principle is the **Mersenne Twister**, also known as MT19937. This uses 624 internal 32-bit integers to describe its state, and abundantly employs XOR as well as other bit-shuffling and swapping operations. It's period is huge, a staggering $2^{19937} - 1$. This should be a pretty good choice for Monte Carlo! The Mersenne Twister is for example available in the GSL-library.[1]

2.5 Using random numbers

Random numbers are usually generated in the standard interval $[0, 1[$ with a uniform distribution. If that's what you need – fine. But often we need random numbers drawn from some other probability distribution function (e.g. a Gaussian). How is this done?

2.5.1 Exact inversion

Recall, the PDF satisfies $\int p(x)\,\mathrm{d}x = 1$ and $p(x) \geq 0$ for all x. Such probability distributions can be transformed to other distributions by observing conservation of probability:

$$p_1(x)\,\mathrm{d}x = p_2(y)\,\mathrm{d}y \tag{2.46}$$

where $y = y(x)$. This leads to the transformation rule

$$p_2(y) = p_1(x) \left| \frac{\mathrm{d}y}{\mathrm{d}x} \right|, \tag{2.47}$$

where here a modulus has been added to neutralize a possible sign change due to the mapping. This can now be used as follows: Suppose we know $p_1(x)$ (usually the distribution returned by our random number generator, in which case $p_1(x) = 1$) and we have a desired distribution $p_2(y)$, then we need to find the mapping $y = y(x)$ that transforms one into the other. We can obtain this by integrating the differential equation

$$\int_{-\infty}^{x} p_1(x')\,\mathrm{d}x' = \int_{-\infty}^{y} p_2(y')\,\mathrm{d}y'. \tag{2.48}$$

[1] `http://www.gnu.org/software/gsl`

This is simply saying that $P_1(x) = P_2(y)$, where $P_1(x)$ and $P_2(y)$ are the cumulative probability distribution functions of p_1 and p_2, respectively. Hence, we need to calculate

$$y = P_2^{-1}\left[P_1(x)\right]. \tag{2.49}$$

In case $p_1(x)$ is an ordinary random number generator, this can also be written as

$$x = \int_{-\infty}^{y} p_2(y')\,\mathrm{d}y'. \tag{2.50}$$

Unfortunately, the inversion cannot always be carried out algebraically, but if this is possible, this is the method of choice.

Example

Suppose you want to have random numbers from the distribution

$$p(y) = \frac{1}{4}y^3 \quad \text{for } y \in [0,2]. \tag{2.51}$$

The cumulative distribution is here

$$P(y) = \int_0^y \frac{y'^3}{4}\,\mathrm{d}y' = \frac{y^4}{16}. \tag{2.52}$$

Hence, we can draw random numbers uniformly from $x \in [0,1[$ and convert them to

$$y = (16x)^{1/4}, \tag{2.53}$$

which then sample our desired distribution function.

Important special cases

The Gaussian distribution

$$p(y) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right) \tag{2.54}$$

is often needed. The cumulative distribution is the error function, which can not be easily inverted without resorting to iterative (and hence comparatively expensive) methods.

There is however a simple trick, known as the Box-Muller method, that can circumvent this issue. Suppose we consider generating a 2D Gaussian distribution

$$p(x,y) = \frac{1}{2\pi} \exp\left(-\frac{x^2+y^2}{2}\right), \tag{2.55}$$

which is simply the product of two 1D-distributions. We can transform this to polar coordinates in the (x,y)-plane:

$$p(x,y)\,\mathrm{d}x\,\mathrm{d}y = \frac{1}{2\pi} \exp\left(-\frac{r^2}{2}\right) r\,\mathrm{d}r\,\mathrm{d}\phi. \tag{2.56}$$

Hence ϕ is uniformly distributed in $[0, 2\pi[$, i.e.

$$\phi = 2\pi \cdot X_1 \tag{2.57}$$

for some standard random number X_1 from the unit interval. For the radial coordinate we have on the other hand:

$$X_2 = \int_0^r r' \exp\left(-\frac{r'^2}{2}\right) \mathrm{d}r' \tag{2.58}$$

This can be integrated and inverted to yield

$$r = \sqrt{-2 \ln X_2}, \tag{2.59}$$

where X_2 is again a random number independently drawn from $[0, 1[$. Finally, we can calculate

$$x = r \cos\phi, \tag{2.60}$$
$$y = r \sin\phi, \tag{2.61}$$

which now yields two perfectly fine Gaussian distributed numbers x and y, which may both be used. This procedure hence always converts two random numbers from $[0, 1[$ to two independent Gaussian distributed numbers.

2.5.2 Rejection method

Assume that $p(x)$ is the desired random number distribution, and $f(x)$ is the distribution that we can create. If we have

$$p(x) \leq C \cdot f(x) \tag{2.62}$$

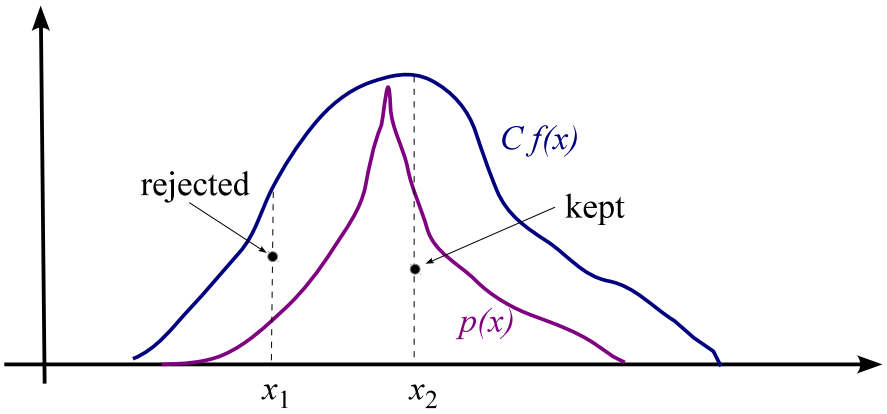
with some known constant C, then we can generate random numbers that sample $p(x)$ with the rejection method. This method works as follows:

1. Generate an x from $f(x)$.
2. Generate a y from a uniform distribution with the bounds $0 \leq y < C \cdot f(x)$.
3. If $y \leq p(x)$ return x as a sample value.
4. Otherwise, i.e. for $y > p(x)$ reject the trial value for x and repeat at step 1.

The probability $\mathrm{d}q$ to get a certain x within $\mathrm{d}x$ with this procedure is:

$$\mathrm{d}q = f(x)\mathrm{d}x \cdot \frac{p(x)}{Cf(x)} = \frac{1}{C}\, p(x)\, \mathrm{d}x \propto p(x)\, \mathrm{d}x. \tag{2.63}$$

Hence this will reproduce the desired probability distribution.



There are a number of advantaged of this approach, in particular, it works in any dimension, and $p(x)$ does not necessarily have to be normalized. The main disadvantage can lie in a low efficiency if the rejection rate is high. The latter is given by the complement to the acceptance rate, which is given by the area under $p(x)$ relative to the area under $Cf(x)$.

Example

Let's assume we want to distribute points uniformly on a sphere. The standard way is to use direct inversion. In 3D, this is still readily possible by the use of spherical polar coordinates. We have for the surface element

$$\sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi = \mathrm{d}\cos\theta \; \mathrm{d}\phi, \tag{2.64}$$

hence the distributions of $\mathrm{d}\cos\theta$ and $\mathrm{d}\phi$ are uniform over their range. Hence we can set

$$\cos\theta = 2\,u_1 - 1, \tag{2.65}$$

$$\phi = 2\pi u_2, \tag{2.66}$$

where u_1 and u_2 are standard uniform numbers. We can then calculate the coordinates as

$$x = \sin\theta \, \cos\phi, \tag{2.67}$$

$$x = \sin\theta \, \sin\phi, \tag{2.68}$$

$$z = \cos\theta. \tag{2.69}$$

This is fine, but cumbersome to generalize to hyperspheres in higher dimensions. A much simpler approach is to use rejection sampling. Suppose we draw three random numbers u_1, u_2, and u_3, which we then map to $[-1, 1]$ through $\tilde{u}_i = 2u_i - 1$. Now we calculate $r^2 = \tilde{u}_1^2 + \tilde{u}_2^2 + \tilde{u}_3^2$, and use the rejection method: We only keep the point if $r^2 \leq 1$, which effectively uniformly samples the inside of a sphere. If we then stretch the kept points as

$$x = \frac{\tilde{u}_1}{r}, \quad x = \frac{\tilde{u}_2}{r}, \quad z = \frac{\tilde{u}_3}{r}, \tag{2.70}$$

they are uniformly distributed on the unit sphere. This method works for any number of dimensions.

2.5.3 Sampling with a stochastic process

There are situations when neither direct inversion nor the rejection method can be readily used to sample from a given distribution function $p(x)$. In this case we can construct a sample of $p(x)$ through a stochastic process that has $p(x)$ as its equilibrium distribution.

We will accomplish this with a so-called *Markov process*, which generates a Markov chain. A Markov chain is a discrete sequence of states,

$$x_1 \xrightarrow{f} x_2 \xrightarrow{f} x_3 \xrightarrow{f} \ldots \xrightarrow{f} x_n, \tag{2.71}$$

where f is a Monte Carlo update operator. The characterizing property of a Markov process is that the transition probability from one state to the next state in the chain,

$$W_f(x \to x') = W_f(x'|x), \tag{2.72}$$

depends *only* on the current state, i.e. information about the history is not used at all. Note that f can here mediate a small update or an arbitrarily large one.

The transition probability has the natural properties

$$\int W_f(x \to x')\,\mathrm{d}x' = 1, \tag{2.73}$$

and $W_f(x \to x') \geq 0$.

We can also apply the transition probability to whole probability distributions, getting the new probability distribution after one transition:

$$p(x) \xrightarrow{f} p'(x') = \int p(x) W_f(x \to x')\,\mathrm{d}x. \tag{2.74}$$

We will now demand two properties of the update step that will turn the Markov process into a very powerful tool:

1. f must preserve $p_{\mathrm{eq}}(x)$ as an equilibrium distribution of the stochastic process, or in other words $p_{\mathrm{eq}}(x)$ must be a fix point of f. This requires

$$p_{\mathrm{eq}}(x') = \int p_{\mathrm{eq}}(x) W_f(x \to x')\,\mathrm{d}x. \tag{2.75}$$

2. Starting from any state x, repeated applications of f must be able to get arbitrarily close to any other state x'. This is called the ergodic property.

Two important results follow from these properties:

- Any ensemble of states approaches the equilibrium distribution if f is applied sufficiently often.
- The collection of states in a single Markov chain under the action of f approaches $p(x)$ as the number of steps goes to infinity.

Let us proof the first of these results. To this end, let $p(x)$ be the PDF of the initial ensemble, and $p_{\mathrm{eq}}(x)$ the equilibrium distribution. After applying f once, we obtain $p'(x')$. We now want to show that p' is closer to p_{eq} than p. To this end, we consider the norm

$$\begin{aligned} ||p' - p_{\mathrm{eq}}|| &\equiv \int |p'(x') - p_{\mathrm{eq}}(x')| \, \mathrm{d}x' && (2.76)\\ &= \int \mathrm{d}x' \left| \int \mathrm{d}x \, W_f(x \to x')(p(x) - p_{\mathrm{eq}}(x)) \right| && (2.77)\\ &\leq \int \mathrm{d}x' \int \mathrm{d}x \, W_f(x \to x') \, |p(x) - p_{\mathrm{eq}}(x)| && (2.78)\\ &= \int \mathrm{d}x \, |p(x) - p_{\mathrm{eq}}(x)| && (2.79)\\ &= ||p - p_{\mathrm{eq}}||. && (2.80) \end{aligned}$$

For the third line, we have basically used the triangle inequality, $|a + b| \leq |a| + |b|$. Thus, the difference between p and p_{eq} shrinks if f is applied. But, perhaps $||p - p_{\mathrm{eq}}||$ gets stuck at some finite value and doesn't really go to zero. This would mean that there must be another fix-point $\tilde{p}_{\mathrm{eq}}$ with $||\tilde{p}_{\mathrm{eq}} - p_{\mathrm{eq}}|| > 0$, and $||\tilde{p}'_{\mathrm{eq}} - p'_{\mathrm{eq}}|| = ||\tilde{p}_{\mathrm{eq}} - p_{\mathrm{eq}}||$.

However, the ergodicity property of the mapping f implies that there are some x' for which

$$\left| \int \mathrm{d}x \, W_f(x \to x')(\tilde{p}_{\mathrm{eq}}(x) - p_{\mathrm{eq}}(x)) \right| < \int \mathrm{d}x \, W_f(x \to x') \, |\tilde{p}_{\mathrm{eq}}(x) - p_{\mathrm{eq}}(x)| \, . \quad (2.81)$$

This is because if A is the set for which $\tilde{p}_{\mathrm{eq}}(x) - p_{\mathrm{eq}}(x) \leq 0$, and B the set for which $\tilde{p}_{\mathrm{eq}}(x) - p_{\mathrm{eq}}(x) > 0$, then there must be some x' from B and some x from A for which we have a non-zero $W_f(x \to x') > 0$, otherwise the two sets would be isolated from each other, violating the ergodic assumption. On the other hand, for the norm of $||\tilde{p}'_{\mathrm{eq}} - p'_{\mathrm{eq}}||$ we get

$$\begin{aligned} ||\tilde{p}'_{\mathrm{eq}} - p'_{\mathrm{eq}}|| &= \int \mathrm{d}x' \, |\tilde{p}'(x')_{\mathrm{eq}} - p'_{\mathrm{eq}}(x')| = \int \mathrm{d}x' \left| \int \mathrm{d}x \, W_f(x \to x')(\tilde{p}(x)_{\mathrm{eq}} - p_{\mathrm{eq}}(x)) \right| \\ &< \int \mathrm{d}x' \int \mathrm{d}x \, W_f(x \to x') \, |\tilde{p}(x)_{\mathrm{eq}} - p_{\mathrm{eq}}(x)| && (2.82)\\ &= \int \mathrm{d}x \, |\tilde{p}(x)_{\mathrm{eq}} - p_{\mathrm{eq}}(x)| = ||\tilde{p}_{\mathrm{eq}} - p_{\mathrm{eq}}||, && (2.83) \end{aligned}$$

where for establishing the $<$-sign we used equation (2.81). The conclusion reached here, $||\tilde{p}'_{\mathrm{eq}} - p'_{\mathrm{eq}}|| < ||\tilde{p}_{\mathrm{eq}} - p_{\mathrm{eq}}||$, contradicts the existence of two equilibrium distributions.

Detailed balance

Almost all of the commonly used update steps follow the **detailed balance condition**, i.e.:

$$p_{\mathrm{eq}}(x) \cdot W_f(x \to x') = p_{\mathrm{eq}}(x') \cdot W_f(x' \to x). \tag{2.84}$$

Here it is obvious and easy to show that $p_{\mathrm{eq}}(x)$ is a fix point under f, while for other choices of f this may still be the case but could be difficult to prove.

So detailed balance and ergodicity are already sufficient conditions to obtain a Markov chain that samples $p_{\mathrm{eq}}(x)$. But we still need to find a concrete realization of W_f.

2.5.4 The Metropolis-Hastings algorithm

The Metropolis-Hastings algorithm provides a simple and generic way for constructing a suitable transition operation. It works as follows:

1. When the currents state is x, propose a new state x' with a proposal probability $q(x \to x')$.

2. Calculate the Hasting's ratio

$$r = \min\left(1, \frac{p(x')\, q(x' \to x)}{p(x)\, q(x \to x')}\right), \tag{2.85}$$

   where the min-operation is used to restrict the value of r to the range $[0, 1]$.

3. Accept the proposed move with probability r (i.e. draw a random number $u \in [0, 1[$, and if its smaller than r, accept), in which case x' is made the next element in the Markov chain. Otherwise, the proposed state is *rejected*, and the old state is added (again) as an element in the Markov chain. This is sometimes called the Metropolis rejection step.

Does the Metropolis algorithm fulfill detailed balance? We can check this by working out the transition probability $W(x \to x')$, which is the product of the proposal probability of the new state and its acceptance probability:

$$W(x \to x') = q(x \to x') \cdot \frac{p(x')\, q(x' \to x)}{p(x)\, q(x \to x')} = \frac{p(x')}{p(x)}\, q(x' \to x). \tag{2.86}$$

Here we have assumed without loss of generality that the Hastings ratio is less than 1. In this case, the inverse transition is then simply given as

$$W(x' \to x) = q(x' \to x) \cdot 1 \tag{2.87}$$

Combining equations (2.86) and (2.87) we then verify the condition of detailed balance.

The proposal probability $q(x \to x')$ is fairly arbitrary – it only must be ergodic, i.e. all states must be reachable through successive applications of q, then the Monte Carlo Markov Chain (MCMC) created by the algorithm will eventually produce a fair sample of the target distribution function $p(x)$. This is a quite remarkable property.

Metropolis update

This is the special case in which the stochastic proposal operator is symmetric, i.e.

$$q(x \to x') = q(x' \to x). \tag{2.88}$$

Then the acceptance probability simply becomes

$$r = \min\left(1, \frac{p(x')}{p(x)}\right). \tag{2.89}$$

A proposed move to a state of higher probability is hence always accepted. (But one sometimes also moves to a proposed state of lower probability.)

The simplest form of such a symmetric update would be something like

$$q(x \to x') : \quad x' = x + e, \tag{2.90}$$

where e is distributed symmetrically around zero and is independent of x. For example, e could be drawn from a normal or uniform distribution of some prescribed width.

Example

Let's come back to our starting point, the generation of a sample from a distribution $p(x)$ with a stochastic process. For simplicity, we want to try this out on a simple Gaussian distribution, $p(x) \propto \exp(-x^2/2)$, using the Metropolis algorithm. As a proposal function, we could for example choose $q(x'|x) : \quad x' = x + (2u - 1)/10$, where u is drawn uniformly from $[0, 1[$. Then we could proceed like this:

1. Start with some x with $p(x) > 0$.
2. Draw u and calculate the proposal x'.
3. Now compute $r = \min\left[1, \exp(-x'^2/2)/\exp(-x^2/2)\right]$.
4. Draw another random number u' from $[0, 1[$ and take x' as new point in the chain if $u' \leq r$, otherwise take x as new point.
5. Repeat at step 2 until you believe you have enough points in the chain.

Note that there can be lots of repeated entries in the chain if the rejection rate is high. Also, you may not use the chain as a randomly sampled sequence from the underlying distribution. Subsequent entries in the change will be highly correlated with each other! Nevertheless, the collection of all the points in a single chain represent a proper sample from the distribution in the limit of an infinitely long chain, thanks to the ergodic property.

2.6 Monte Carlo simulations of lattice models

An important application of MCMC techniques lies in the thermodynamics of physical systems, for example solids described on a lattice. In this case, we may have a field $\phi_{\mathbf{x}}$ at each lattice site $\mathbf{x}$, whose dynamics is described by some Hamiltonian $H(\phi)$.

Assuming the canonical ensemble, the partition function is then given by

$$Z = \int \exp\left[-\frac{H(\phi)}{kT}\right] [\mathrm{d}\phi], \tag{2.91}$$

where $[\mathrm{d}\phi] = \mathrm{d}\phi_1\, \mathrm{d}\phi_2\, \mathrm{d}\phi_3 \cdots \mathrm{d}\phi_N$ is a short-hand notation for the differential volume element in the extremely high-dimensional space of all possible field configurations.

In practice, very often the task to compute the thermal average of some quantity A arises. This is given by

$$\langle A \rangle = \frac{1}{Z} \int A(\phi) \exp\left[-\frac{H(\phi)}{kT}\right] [\mathrm{d}\phi]. \tag{2.92}$$

Unfortunately, because of the high dimensionality, these integrals cannot be carried out with standard techniques. We hence would like to employ Monte Carlo integration combined with importance sampling in which the phase-space points of the system are chosen according to $p(\phi) \propto \exp\left[-\frac{H(\phi)}{kT}\right]$. For obtaining such a sample, we need a stochastic process, because neither direct inversion nor the rejection method are feasible.

For producing the required Markov chain, one can for example use the Metropolis-Hastings algorithm. If we symmetrically propose new states, then the acceptance probability will become

$$r = \min\left[1, \exp\left(-\frac{H(\phi') - H(\phi)}{kT}\right)\right], \tag{2.93}$$

still leaving many ways for how the proposals are generated.

Another possibility is to employ the so-called *heat bath*, or Gibbs sampler. This directly sets

$$W_f(\phi \to \phi') = C \exp\left(-\frac{H(\phi')}{kT}\right), \tag{2.94}$$

which even doesn't depend on the state ϕ at all. In practice, it is however not always trivial to invert this and actually use it.

In either case, once a sufficiently long Markov chain with sampled states has been constructed, we can use it to straightforwardly calculate alls sorts of thermal averages by replacing the integrals with averages of $A(\phi)$ at the sampled points.

Example: Ising Model

The Ising model is the simplest discrete spin model in which the field variable is $s_{\mathbf{x}} = \pm 1$, i.e. at each lattice site the spin either points up or down. The partition function of the system is

$$Z = \sum_{\{s_{\mathbf{x}}\}} \exp\left[-\beta\left(\frac{1}{2}\sum_{<\mathbf{x},\mathbf{y}>}(1 - s_{\mathbf{x}}s_{\mathbf{y}}) + B\sum_{\mathbf{x}} s_{\mathbf{x}}\right)\right]. \tag{2.95}$$

Here the first sum is over all possible spin configurations. The sum $< \mathbf{x}, \mathbf{y} >$ is only over pairs of nearest lattice sites; only their spin interaction is counted. Finally, B describes an external magnetic field, which one may also put to zero. $\beta = 1/T$ measures the temperature (we use a natural system of units here).

For $B = 0$, the Ising model shows 2nd-order phase transitions if the number of dimensions is larger than one. Then, below a certain critical temperature, spontaneous magnetization of the medium occurs. For $d = 2$, the transition temperature has been calculated analytically by Onsager, $\beta_c = \ln(1 + \sqrt{2})$, but for higher dimensions, analytic solutions are not known. Here one therefore needs to turn to Monte Carlo simulations.

The simplest approach is to use the Metropolis algorithm in which one selects a single spin $s_{\mathbf{x}}$ at lattice site $\mathbf{x}$ and proposes the opposite spin direction $s'_{\mathbf{x}}$. The selection of the lattice site can be done randomly, or in red-black ordering (or even type-writer ordering, but this is less efficient). One then computes the local energy functional $E_{\mathbf{x}}$ which involves all the terms in the interaction energy that involve the chosen spin. This gives rise to a change $\delta E_{\mathbf{x}} = E_{\mathbf{x}}(s'_{\mathbf{x}}) - E_{\mathbf{x}}(s_{\mathbf{x}})$ due to the spin flip. The acceptance probability is then given as

$$r = \min\left(1, \exp[-\beta\,\delta E_{\mathbf{x}}]\right), \tag{2.96}$$

and with this one can generate a long MC chain with different states of the system, which will eventually represent thermal equilibrium at the prescribed temperature.

For this simple spin system, one may also use the heat bath update, and choose the new spin direction of the selected site according to

$$p(s_{\mathbf{x}}) = \frac{\mathrm{e}^{-\beta E_{\mathbf{x}}(s_{\mathbf{x}}=+1)}}{\mathrm{e}^{-\beta E_{\mathbf{x}}(s_{\mathbf{x}}=+1)} + \mathrm{e}^{-\beta E_{\mathbf{x}}(s_{\mathbf{x}}=-1)}}. \tag{2.97}$$

Here the inversion is trivially possible; one simply draws a random number u from $[0, 1]$ and picks the $+1$ direction when $u < p(s_{\mathbf{x}})$ and spin equal to -1 otherwise.

Once a very long MCMC chain of the system in thermal equilibrium has been calculated, one can simply compute various thermodynamic quantities of interest by straightforward averaging (which really is Monte Carlo integration with importance sampling). For example:

- Mean magnetization: $M = \frac{1}{V}\sum_i s_i$
- Specific heat: $C_V = \frac{1}{V}\frac{\partial E}{\partial T} = \langle E^2\rangle - \langle E\rangle^2$
- Magnetic susceptibility: $\chi_M = \frac{1}{V}\frac{\partial M}{\partial T} = \langle M^2\rangle - \langle M\rangle^2$

2.7 Monte Carlo Markov Chains in parameter estimation

Another important application of MCMC techniques lies in parameter estimation, in particular in the context of *Baysian data analysis*. Often in physics, we want to use some experimental data

$$\mathbf{z} = (z_1, z_2, \ldots, z_n) \tag{2.98}$$

to infer a number of parameters

$$\theta = (\theta_1, \theta_2, \ldots, \theta_m) \tag{2.99}$$

which describe a physical model/theory. For example, observations of the microwave background radiation (where $\mathbf{z}$ might refer to the pixels with measured temperature fluctuations) are used to estimate parameters such as the mean density and expansion rate of the Universe.

This application of MCMC is not covered here as it is covered in MVComp2 (Computational Statistics).

3 Integration of ordinary differential equations

A large set of simulation methods are based on integrating *ordinary differential equations* (ODEs) numerically by discretization, every often to evolve a many-particle system in time. Just like Monte Carlo techniques discussed in the previous Chapter, this numerical solution to ODEs is an approximate technique. In contrast to MC, however, such integration schemes are deterministic.

We discuss in the following some basic methods for the integration of ODEs. These are relations between an unknown scalar or vector-valued function $\mathbf{y}(t)$ and its derivatives with respect to the dependent variable (t in this case – the following discussion associates this with 'time', but this could of course be also any other variable). Such equations hence formally take the form

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}, t), \tag{3.1}$$

and we seek the solution $\mathbf{y}(t)$, subject to boundary conditions.

Many simple dynamical problems can be written in this form, including ones that involve second or higher derivatives. This is done through a procedure called **reduction to 1st order**. One does this by adding the higher derivatives, or combinations of them, as further rows to the vector $\mathbf{y}$.

For example, consider a simple pendulum with the equation of motion

$$\ddot{q} = -\frac{g}{l}\sin(q), \tag{3.2}$$

where q is the angle with respect to the vertical. Now define $p \equiv q$, yielding a state vector

$$\mathbf{y} \equiv \begin{pmatrix} q \\ p \end{pmatrix}, \tag{3.3}$$

and a first order ODE of the form:

$$\frac{\mathrm{d}\mathbf{y}}{\mathrm{d}t} = \mathbf{f}(\mathbf{y}) = \begin{pmatrix} p \\ -\frac{g}{l}\sin(q) \end{pmatrix}. \tag{3.4}$$

A numerical approximation to the solution of an ODE is a set of values $\{y_0, y_1, y_2, \ldots\}$ at discrete times $\{t_0, t_1, t_2, \ldots\}$, obtained for certain boundary conditions. The most common boundary condition for ODEs is the **initial value problem** (IVP), where the state of $\mathbf{y}$ is known at the beginning of the integration interval. It is however

also possible to have mixed boundary conditions where $\mathbf{y}$ is partially known at both ends of the integration interval.

There are many different methods for obtaining a discrete solution of an ODE system (e.g. Press et al., 1992). We shall here discuss some of the most basic ones, restricting ourselves to the IVP, for simplicity.

3.1 Explicit Euler method

This solution method, sometimes also called "forward Euler", uses the iteration

$$y_{n+1} = y_n + f(y_n)\Delta t, \tag{3.5}$$

where y can also be a vector. Δt is the integration step.

- This approach is the simplest of all.
- The method is called *explicit* because y_{n+1} is computed with a right-hand-side that only depends on things that are already known.
- The stability of the method can be a sensitive function of the stepsize, and will in general only be reached for a sufficiently small step size.
- It is recommended to refrain from using this scheme in practice, since there are other methods that offer higher accuracy at the same or lower computational cost. The reason is that the Euler method is only *first order accurate*. To see this, note that the truncation error in a single step is of order $\mathcal{O}_s(\Delta t^2)$, which follows simply from a Taylor expansion. To simulate over time T, we need however $N_s = T/\Delta t$ steps, producing a total error that scales as $N_s\mathcal{O}_s(\Delta t^2) = \mathcal{O}_T(\Delta t)$.

We remark in passing that for a method to reach a global error that scales as $\mathcal{O}_T(\Delta t^n)$ (which is then called an "n^{th}-order accurate scheme), a local truncation error one order higher is required, i.e. $\mathcal{O}_s(\Delta t^{n+1})$.

3.2 Implicit Euler method

In a so-called "backwards Euler" scheme, one uses

$$y_{n+1} = y_n + f(y_{n+1})\Delta t, \tag{3.6}$$

which seemingly represents only a tiny change compared to the explicit scheme.

- This approach has excellent stability properties, and for some problems, is in fact essentially always stable even for extremely large timestep. Note however that the accuracy will usually become very bad when using such large steps.

- This stability property makes implicit Euler sometimes useful for *stiff equations* where the derivatives (suddenly) can become very large.
- The implicit equation for y_{n+1} that needs to be solved here corresponds in many practical applications to a non-linear equation that can be complicated to solve for y_{n+1}. Often, the root of the equation has to be found numerically, for example through an iterative technique.

Stability of the Euler method in an example

Let's look at a simple problem to investigate the stability of forward and backwards Euler. Suppose we have the ODE

$$\frac{\mathrm{d}y}{\mathrm{d}t} = -\alpha\, y, \tag{3.7}$$

with $\alpha > 0$ and $y(0) = y_0$. Here we of course know the analytic solution, given by $y(t) = y_0 \exp(-\alpha\, t)$.

What does *explicit* Euler give for this problem? We can work this out as

$$y_{n+1} = y_n + \dot{y}_n \Delta t = y_n - \alpha\, y_n \Delta t = y_n(1 - \alpha\, \Delta t). \tag{3.8}$$

Hence every step gives us a factor $(1 - \alpha\, \Delta t)$, and after n steps we have

$$y_n = (1 - \alpha\, \Delta t)^n\, y_0. \tag{3.9}$$

We see that for $0 < 1 - \alpha\, \Delta t < 1$ (i.e. equivalently for $\Delta t < 1/\alpha$) the y_n monotonically decline. This is acceptable. For $-1 < 1 - \alpha\, \Delta t < 0$ (i.e. equivalently for $1/\alpha < \Delta t < 2/\alpha$), the series oscillates but still declines overall, which is already somewhat problematic. But for $1 - \alpha\, \Delta t < -1$ (i.e. for $\Delta t > 2/\alpha$), the solution blows up and oscillates, which is as bad as it gets. So here the integration clearly becomes unstable for a too large timestep.

Now let's look at the *implicit* Euler instead. Here we have

$$y_{n+1} = y_n + \dot{y}_{n+1} \Delta t = y_n - \alpha\, y_{n+1} \Delta t, \tag{3.10}$$

yielding

$$y_{n+1} = \frac{y_n}{1 + \alpha\, \Delta t}, \tag{3.11}$$

which declines and is unconditionally stable for arbitrarily large timestep.

3.3 Implicit midpoint rule

If we use

$$y_{n+1} = y_n + f\left(\frac{y_n + y_{n+1}}{2}\right) \Delta t \tag{3.12}$$

we obtain the implicit midpoint rule. This is *second order accurate*, but still implicit, so difficult to use in practice. Interestingly, it is also time-symmetric, i.e. one can formally integrate backwards and recover exactly the same steps (modulo floating point round-off errors) as in a forward integration.

3.4 Runge-Kutta methods

The Runge-Kutta schemes form a whole class of versatile integration methods (e.g. Atkinson, 1978; Stoer & Bulirsch, 2002). Let's derive one of the simplest Runge-Kutta schemes.

1. We start from the exact solution,
$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} f\left(y(t)\right) \mathrm{d}t. \tag{3.13}$$
2. Next, we approximate the integral with the (implicit) trapezoidal rule:
$$y_{n+1} = y_n + \frac{f(y_n) + f(y_{n+1})}{2} \Delta t. \tag{3.14}$$
3. Runge proposed in 1895 to predict the unknown y_{n+1} on the right hand side by an Euler step, yielding a *2nd order accurate Runge-Kutta scheme*, sometimes also called predictor-corrector scheme:
$$\begin{aligned} k_1 &= f(y_n, t_n), & (3.15)\\ k_2 &= f(y_n + k_1 \Delta t, t_{n+1}), & (3.16)\\ y_{n+1} &= y_n + \frac{k_1 + k_2}{2} \Delta t. & (3.17) \end{aligned}$$
Here the step done with the derivate of equation (3.15) is called the 'predictor' and the one done with equation (3.16) is the corrector step.

Higher order Runge-Kutta schemes

A variety of further Runge-Kutta schemes of different order can be defined. Perhaps the most commonly used is the classical 4^{th}-order Runge-Kutta scheme:
$$\begin{aligned} k_1 &= f(y_n, t_n) & (3.18)\\ k_2 &= f\left(y_n + k_1 \frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right) & (3.19)\\ k_3 &= f\left(y_n + k_2 \frac{\Delta t}{2}, t_n + \frac{\Delta t}{2}\right) & (3.20)\\ k_4 &= f\left(y_n + k_3 \Delta t, t_n + \Delta t\right). & (3.21) \end{aligned}$$
These four function evaluations per step are then combined in a weighted fashion to carry out the actual update step:
$$y_{n+1} = y_n + \left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right) \Delta t + \mathcal{O}(\Delta t^5). \tag{3.22}$$
We note that the use of higher order schemes also entails more function evaluations per step, i.e. the individual steps become more complicated and expensive. Because of this, higher order schemes are not always better; they usually are up to some point, but sometimes even a simple second-order accurate scheme can be the best choice for certain problems.

3.5 Adaptive step sizes

One issue left open in our discussion thus far is the choice of the optimum integration step. How can we get an optimum compromise between accuracy, efficiency and stability?

To control the accuracy, we need a way to estimate the local integration error, and a scheme for adjusting the step size such that a prescribed desired maximum error is guaranteed. One useful idea for controlling this lies in solving a step of size Δt twice, once with Δt (case A), and once by doing two steps of size $\frac{\Delta t}{2}$ (case B). The difference between the results then yields an estimate of the truncation error.

For example, suppose we use a p-th order scheme (e.g. Runge-Kutta forth order), then we have for the two end states:

$$y_A - y(t_0 + \Delta t) = \alpha \cdot (\Delta t)^{p+1} + \mathcal{O}(\Delta t^{p+2}), \tag{3.23}$$

$$y_B - y(t_0 + \Delta t) = 2 \times \left[\alpha \cdot (\Delta t/2)^{p+1}\right] + \mathcal{O}(\Delta t^{p+2}). \tag{3.24}$$

This yields an error estimate of

$$\epsilon = |y_A - y_B| = \alpha \cdot (\Delta t)^{p+1} \cdot (1 - 2^{-p}). \tag{3.25}$$

Assuming we are given a local error bound ϵ_0 for each step, we can now construct an algorithm where this is respected in every integration step:

1. Take a step of size Δt (yielding y_A), and two of stepsize $\frac{\Delta t}{2}$ (yielding y_B). We can then get an error estimate as $y = |y_A - y_B|$.

2. If $\epsilon > \epsilon_0$, discard the step, halve the timestep, $\Delta t' = \frac{\Delta t}{2}$, and try again.

3. If $\epsilon \ll \epsilon_0$, keep y_B and double the step for the next step, $\Delta t' = 2\Delta t$.

4. Else if $\epsilon < \epsilon_0$, keep y_B and retain Δt for the next step.

When does it make sense to double the step in point 3? This should only be done if the estimated new error is still smaller than the error bound, i.e. for

$$\epsilon \cdot 2^{p+1} < \epsilon_0. \tag{3.26}$$

But there is another complication. So far, our error estimate has been entirely local, disregarding the fact that we might need to do many integration steps. If we want to guarantee a certain error globally, even when all the errors add up with the same sign, we have to lower ϵ_0 with smaller step size Δt, because we then need to do more steps!

We can for example account for this by setting

$$\epsilon_0 = \frac{\Delta t}{T} \epsilon_0^{\text{global}}, \tag{3.27}$$

where T is the total integrated time, and $\epsilon_0^{\text{global}}$ is our prescribed error bound for the whole integration. This means we loose effectively one power of Δt again, because of

$$\epsilon \simeq \alpha \cdot (1 - 2^{-p}) \cdot \Delta t^{p+1} < \frac{\Delta t}{T} \epsilon_0^{\text{global}}. \tag{3.28}$$

Instead of step doubling one can also use a continuous step adjustment. The idea is to scale the timestep such that the estimated error matches the desired error ϵ_0:

$$(\Delta t)^{\text{desired}} = (\Delta t) \cdot \left(\frac{\epsilon_0}{\epsilon}\right)^{1/(p+1)}, \tag{3.29}$$

where ϵ is the error if a step of size Δt is taken, and ϵ_0 is the error level that one wants. $(\Delta t)^{\text{desired}}$ is then an estimate of the timestep that would deliver this error level.

Use of this in practice requires that we obtain for each step also an estimate of the error ϵ that is made. Then we can use this expression to determine the timestep for the next step. Such a 'built-in' error estimate can be delivered by so-called embedded Runge-Kutta schemes, which compute it more cheaply than done with the step-doubling technique.

A typical algorithm for continuous adaptive step size control would then for example work as follows:

1. Advance the system for a step Δt and estimate the error ϵ of the step at the same time (we here assume a p-th order scheme with $\mathcal{O}(\epsilon) = \Delta t^{p+1}$).

2. Calculate the new step size as

$$(\Delta t)^{\text{new}} = \beta \cdot (\Delta t) \cdot \left(\frac{\epsilon_0}{\epsilon}\right)^{1/(p+1)} \tag{3.30}$$

   where $\beta \sim 0.9$ is an empirical "safety factor".

3. If $\epsilon < \epsilon_0$ accept the step taken in (1), otherwise discard it and try again with new step size.

The well-known Runge-Kutta-Fehlberg integration method is of this type.

3.6 The leapfrog

Suppose we have a second order differential equation of the type

$$\ddot{x} = f(x). \tag{3.31}$$

This could of course be brought into standard form, $\dot{\mathbf{y}} = \tilde{\mathbf{f}}(\mathbf{y})$, by defining something like $\mathbf{y} = (x, \dot{x})$ and $\tilde{\mathbf{f}} = (\dot{x}, f(x))$, followed by applying a Runge-Kutta scheme as introduced above.

However, there is also another approach in this case, which turns out to be particularly simple and interesting. Let's define $v \equiv \dot{x}$. Then the so-called Leapfrog integration scheme is the mapping $(x_n, v_n) \to (x_{n+1}, v_{n+1})$ defined as:

$$v_{n+\frac{1}{2}} = v_n + f(x_n)\frac{\Delta t}{2}, \tag{3.32}$$

$$x_{n+1} = x_n + v_{n+\frac{1}{2}}\,\Delta t, \tag{3.33}$$

$$v_{n+1} = v_{n+\frac{1}{2}} + f(x_{n+1})\frac{\Delta t}{2}. \tag{3.34}$$

- This scheme is 2nd-order accurate (proof through Taylor expansion).
- It requires only 1 evaluation of the right hand side per step (note that $f(x_{n+1})$ can be reused in the next step.
- The scheme can be written in a number of alternative ways, for example by combining the two half-steps of two subsequent steps. One then gets

  $$x_{n+1} = x_n + v_{n+\frac{1}{2}}\,\Delta t, \tag{3.35}$$

  $$v_{n+\frac{3}{2}} = v_{n+\frac{1}{2}} + f(x_{n+1})\,\Delta t. \tag{3.36}$$

  One here sees the time-centered nature of the formulation very clearly, and the interleaved advances of position and velocity give it the name leapfrog.

The performance of the leapfrog on certain problems is found to be surprisingly good, better than that of other schemes such as Runge-Kutta which have formally the same or even a better error order. This is illustrated in Figure 3.1 for the Kepler problem, i.e. the integration of the motion of a small point mass in the gravitational field of a large mass.

We see that the long-term evolution is entirely different. Unlike the RK schemes, the leapfrog does not build up a large energy error. So why is the leapfrog behaving here so much better than other 2nd order or even 4th order schemes?

3.7 Symplectic integrators

The reason for these beneficial properties lies in the fact that the leapfrog is a so-called symplectic method. These are structure-preserving integration methods (e.g. Saha & Tremaine, 1992; Hairer et al., 2002) that observe important special properties of Hamiltonian systems: Such systems have first conserved integrals (such as the energy), they also exhibit phase-space conservation as described by the Liouville theorem, and more generally, they preserve Poincare's integral invariants.

Symplectic transformations

- A linear map $F : \mathbb{R}^{2d} \to \mathbb{R}^{2d}$ is called symplectic if $\omega(F\xi, F\eta) = \omega(\xi, \eta)$ for all vectors $\xi, \eta \in \mathbb{R}^{2d}$, where w gives the area of the parallelgram spanned by the the two vectors.

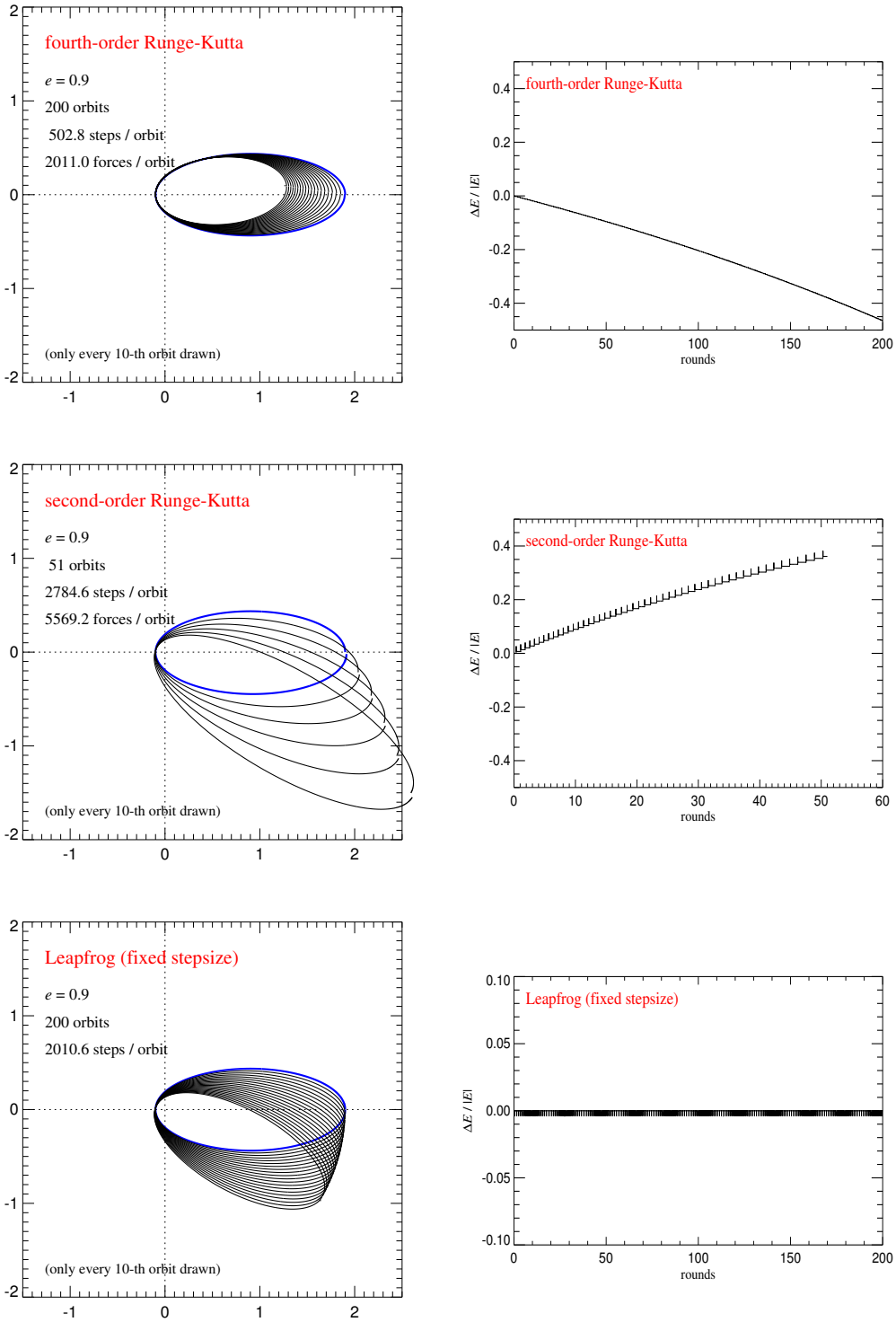


Figure 3.1: Kepler problem integrated with different integration schemes (Springel, 2005). The panels on top are for a 4th-order Runge Kutta scheme, the middle for a 2nd order Runge-Kutta, and the bottom for a 2nd-order leapfrog. The leapfrog does not show a secular drift of the total energy, and is hence much more suitable for long-term integration of this Hamiltonian system.

- A differentiable map $g : U \to \mathbb{R}^{2d}$ is called symplectic if its Jacobian matrix is everywhere symplectic, i.e. $\omega(g'\xi, g'\eta) = \omega(\xi, \eta)$.
- **Poincare's theorem** states that the time evolution generated by a Hamiltonian in phase-space is a symplectic transformation.

The above suggests that there is a close connection between exact solutions of Hamiltonians and symplectic transformations. Also, two consecutive symplectic transformations are again symplectic.

Separable Hamiltonians

Dynamical problems that are described by Hamiltonians of the form

$$H(p, q) = \frac{p^2}{2m} + U(q) \tag{3.37}$$

are quite common. These systems have separable Hamiltonians that can be written as

$$H(p, q) = H_{\text{kin}}(p) + H_{\text{pot}}(q). \tag{3.38}$$

Now we will allude to the general idea of *operator splitting* (Strang, 1968). Let's try to solve the two parts of the Hamiltonian individually:

1. For the part $H = H_{\text{kin}} = \frac{p^2}{2m}$, the equations of motion are

$$\dot{q} = \frac{\partial H}{\partial p} = \frac{p}{m}, \tag{3.39}$$

$$\dot{p} = -\frac{\partial H}{\partial q} = 0. \tag{3.40}$$

   These equations are straightforwardly solved and give

$$q_{n+1} = q_n + p_n \Delta t, \tag{3.41}$$
$$p_{n+1} = p_n. \tag{3.42}$$

   Note that this solution is exact for the given Hamiltonian, for arbitrarily long time intervals Δt. Given that it is a solution of a Hamiltonian, the solution constitutes a symplectic mapping.

2. The potential part, $H = H_{\text{pot}} = U(q)$, leads to the equations

$$\dot{q} = \frac{\partial H}{\partial p} = 0 \tag{3.43}$$

$$\dot{p} = -\frac{\partial H}{\partial q} = -\frac{\partial U}{\partial q}. \tag{3.44}$$

This is solved by

$$q_{n+1} = q_n, \tag{3.45}$$

$$p_{n+1} = p_n - \frac{\partial U}{\partial q}\Delta t. \tag{3.46}$$

Again, this is an exact solution independent of the size of Δt, and therefore a symplectic transformation.

Let's now introduce an operator $\varphi_{\Delta t}(H)$ that describes the mapping of phase-space under a Hamiltonian H that is evolved over a time interval Δt, then it is easy to see that the leapfrog is given by

$$\varphi_{\Delta t}(H) = \varphi_{\frac{\Delta t}{2}}(H_{\mathrm{pot}}) \circ \varphi_{\Delta t}(H_{\mathrm{kin}}) \circ \varphi_{\frac{\Delta t}{2}}(H_{\mathrm{pot}}) \tag{3.47}$$

for a separable Hamiltonian $H = H_{\mathrm{kin}} + H_{\mathrm{pot}}$.

- Since each individual step of the leapfrog is symplectic, the concatenation is also symplectic.
- In fact, the leapfrog generates the exact solution to a modified Hamiltonian H_{leap}, where $H_{\mathrm{leap}} = H + H_{\mathrm{err}}$. The difference lies in the 'error Hamiltonian' H_{err}, which is given by

  $$H_{\mathrm{err}} \propto \frac{\Delta t^2}{12}\left\{\{H_{\mathrm{kin}}, H_{\mathrm{pot}}\}, H_{\mathrm{kin}} + \frac{1}{2}H_{\mathrm{pot}}\right\} + \mathcal{O}(\Delta t^3), \tag{3.48}$$

  where the curly brackets are Poisson brackets (Goldstein, 1950). This can be demonstrated by expanding

  $$\mathrm{e}^{(H+H_{\mathrm{err}})\Delta t} = \mathrm{e}^{H_{\mathrm{pot}}\frac{\Delta t}{2}}\ \mathrm{e}^{H_{\mathrm{kin}}\Delta t}\ \mathrm{e}^{H_{\mathrm{pot}}\frac{\Delta t}{2}} \tag{3.49}$$

  with the help of the Baker-Campbell-Hausdorff formula (Campbell, 1897; Saha & Tremaine, 1992).
- This property explains the superior long-term stability of the integration of conservative systems with the leapfrog. Because it respects phase-space conservation, secular trends are largely absent, and the long-term energy error stays bounded and reasonably small.

4 Molecular Dynamics simulations

The aim of molecular dynamics (MD) simulations is to model a system in microscopic detail, over a physical length of time relevant for certain properties of interest. The systems that are studied may consist of, for example, individual atoms, molecules, macromolecules, proteins in a solvent, etc.

An important point is that one studies the molecules through *classical equations of motions*, based on an approximate representation of the inter-molecule and/or intra-molecular forces. The corresponding force laws may be empirically derived or in some cases can be motivated by quantum mechanical calculations.

Molecular dynamics simulations are intimately connected to statistical mechanics, and in fact, for interfering macroscopic properties, concepts of statistical mechanics need to be used. This in particular means that the precise *microstate* (given for example in terms of the positions and velocities of all atoms) of an MD simulation is unimportant, instead we are interested in *ensemble averages* of *macroscopic variables*, such as temperature, pressure, diffusion coefficient, etc.

In principle, carrying out a proper ensemble average would mean to average over many different simulations of the system. This is usually impossible. One can however resort to the *ergodic hypothesis* which postulates that the ensemble average is equal to the time average of a specific system. We can hence hope to accurately measure the macroscopic thermodynamic properties of a system by looking at a single realization for a long enough time and average our measurements over this time span.

4.1 Interaction potentials

Within the Born-Oppenheimer approximation, the total wave function ψ_{tot} is separated into the nuclear ψ_{n} and electronic wave function ψ_{e},

$$\psi_{\text{tot}}(R, r) = \psi_{\text{n}}(R)\psi_{\text{e}}(R; r), \tag{4.1}$$

where R and r are the coordinates and momenta of the nuclei and electrons, respectively. Thus, the electronic wave function ψ_{e} only parametrically depends on the position, not on the dynamics, of the nuclei. As a result of this approximation, Eq. 4.1 separates into two equations, a time-dependent Schrödinger equation for the motion of the nuclei, and a time-independent Schrödinger equation for the electronic dynamics.

The Born-Oppenheimer approximation allows to treat the electronic wave function of a system as a function of only the nuclear coordinates. Within this framework,

the calculation of electronic energies requires the solvation of the time-independent Schrödinger equation for the electrons. This, however, is prohibitive due to the large number of electrons in real-world molecular systems.

As a second approximation, therefore, a classical force field for the calculation of the potential energy of the system as a function of nuclear coordinates is derived, from fitting to the quantum-mechanical ground state energy and/or from experimental observables.

Within these approximations, in which electronic degrees of freedom of each atom are neglected, the potential energy for a system containing N atoms can be written down as

$$V(\mathbf{r}_1, \ldots, \mathbf{r}_N)) \; = \sum_{j>i}^{N} v_2(\mathbf{r}_i, \mathbf{r}_j) + \sum_{j>i}^{N} \sum_{k>j}^{N} v_3(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) + \ldots \tag{4.2}$$

where v_2 are pair-wise interaction potentials, v_3 are triplet potentials, etc. While purely atomistic systems are typically described by only pair potentials, $V(\mathbf{r}_1, \ldots, \mathbf{r}_N)$ of molecular systems includes most commonly also 3-body and 4-body terms, see below. A pair potential only depends on the pair separation, $v_2(\mathbf{r}_i, \mathbf{r}_j = v_2(|\mathbf{r}_i - \mathbf{r}_j|)$.

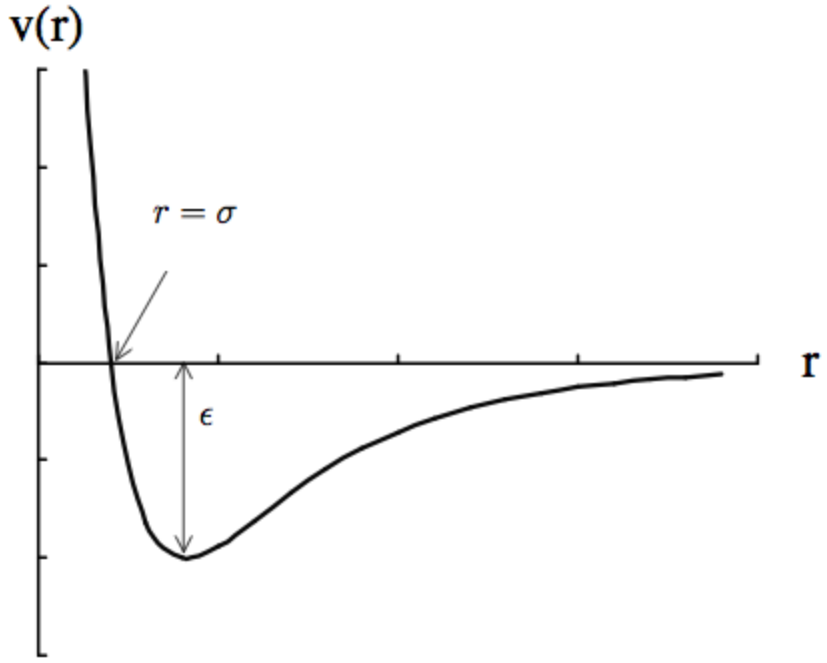
With the kinetic energy of $\sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i}$, we arrive at a conservative Hamiltonian system,

$$H = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m_i} + V(\mathbf{r}_1, \ldots, \mathbf{r}_N), \tag{4.3}$$

for which we can readily derive equations of motion in the standard way. Once the equations of motion are written down, we can integrate them as an ordinary N-body system, using, for example, the Leapfrog or Verlet integration schemes. While these integration schemes are of low-order, recall that they are symplectic, hence they have particularly good stability properties for long-term integrations of conservative systems.

4.1.1 Non-bonded interactions

Let us first consider the simplest case of a monatomic liquid such as argon. It features attractive forces at large distances due to London dispersion or 'van der Waals interactions', and strong repulsive forces at short distances due to the Pauli repulsion. London dispersion can be considered as an attraction between fluctuating dipoles, and scales with r_{ij}^{-6}, while the Pauli repulsion scales with $\frac{e^{-r_{ij}}}{r_{ij}}$.



A potential often used to approximately describe the interaction between pairs of atoms is the 12-6 Lennard-Jones potential:

$$v(r) = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{4.4}$$

Here ϵ characterizes the interaction strength, and σ the range. There is a minimum of the potential at $r_0 = 2^{1/6}\sigma \simeq 1.12\,\sigma$, with $v(r_0) = -\epsilon$. For distances below r_0, the force is (strongly) repulsive, mimicking the exponential dependency of the Pauli repulsion. For larger distances, it is attractive and approaches zero quite quickly.

For example, for argon suitable parameters to describe the potential are $\epsilon/k_\mathrm{B} = 120\,\mathrm{K}$ and $\sigma = 3.4 \times 10^{-8}\,\mathrm{cm}$.

One should bear in mind that MD simulations of monatomic systems typically neglect multi-body terms such as v_3 in Eq. 4.1. Up to 10 % of the lattice energy of solid state argon has been estimated to arise from non-additive non-pairwise terms. Despite of this significant contribution, these terms are rarely included in computer simulations, first because the pairwise approximation is reasonable, and secondly because triplet terms would render computer simulations in most cases too expensive. Hence, parameters for the Lennard-Jones potential are *effective* parameter taking any higher-order interactions into account.

We note that highly simplified pair potentials might be of interest for general investigations of liquids and comparison to theory. Examples are the hard-sphere potential

$$v^{\mathrm{HS}}(r_{ij}) = \begin{cases} \infty \text{ if } r_{ij} < \sigma \\ 0 \text{ if } r_{ij} \geq \sigma \end{cases} \tag{4.5}$$

or the soft-sphere potential

$$v^{\mathrm{SS}}(r_{ij}) = \epsilon(\sigma/r_{ij})^{\nu}, \tag{4.6}$$

with 'harder' potentials for higher ν.

For the long-range electrostatic attraction or repulsion of ions, a Coulombic interactions between the charges is added to the Lennard-Jones potential,

$$v^{\mathrm{qq}}(r_{ij}) = \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} \tag{4.7}$$

where q_i and q_i are the atomic charges, and ϵ_0 is the electrical permittivity of space.

4.1.2 Bonded interactions

When it comes to molecules, interactions within each molecule through chemical bonds need to be considered. The simplest way to describe a chemical bond is a harmonic potential. The two parameters, the spring constant and the equilibrium length, can be derived from spectroscopy and X-ray structures, respectively. This applies analogously to angles within a set of three atoms connected by bonds.

Torsional rotations around molecular bonds are very critical as they involve barriers in the range of $k_{\mathrm{B}}T$, with k_{B} the Boltzmann constant, and T the (laboratory or body) temperature, and thus are heavily sampled. They can be described among others by a functional form put forward by Ryckaert and Bellemans Ryckaert & Bellemans (1975)

$$v_{\mathrm{dih}}(\phi_{ijkl}) = \sum_{n=0}^{5} C_n(cos(\psi))^n, \tag{4.8}$$

where $\psi = \phi - 180°$, and i, j, k and l are the atoms involved in the dihedral, and C_n are constants in kJ mol^{-1}.

4.1.3 Example for a molecular force field

Fig. 4.1 illustrates the components of a force field comprising bonded interactions, namely energy terms for bonds, v_{b}, angles, v_{a}, proper and improper dihedrals, v_{dih}, and additional non-bonded interactions, specifically non-polar Lennard-Jones interactions, v_{LJ}, and electrostatic interactions between partial charges of the atoms,

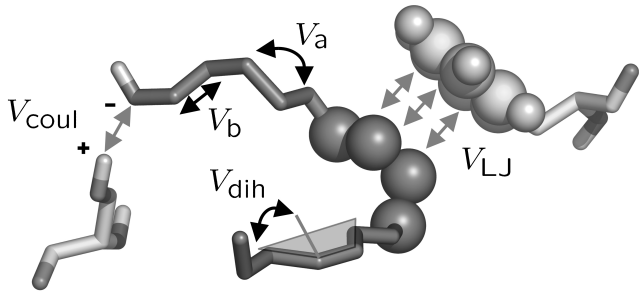


Figure 4.1: Scheme to illustrate the components of a typical mechanical force field as used for the simulations of macromolecules such as proteins. Black arrows: bonded interactions: v_b, bond-stretching potential, v_a, angle-bending potential, v_{dih}, dihedral (out-of-plane) potential. Grey arrows: non-bonded interactions: v_{Coul}, Coulomb potential, v_{LJ}, Lennard-Jones potential.

v_{Coul}. It is given by

$$\begin{aligned} v(R) &= v_b + v_a + v_{dih} + v_{Coul} + v_{LJ} \\ &= \sum_{bonds} \frac{k_i}{2}(l_i - l_{i,0})^2 \\ &+ \sum_{angles} \frac{k_i}{2}(\theta_i - \theta_{i,0})^2 \\ &+ \sum_{dihedrals} \sum_{n=0}^{5} C_n(cos(\psi))^n \\ &+ \sum_{atoms\ i} \sum_{atoms\ j>i} 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}}. \end{aligned} \tag{4.9}$$

The force field parameters for bonded interactions comprise the equilibrium bond length $l_{i,0}$ and angle $\theta_{i,0}$, the respective force constants k_i, and the Ryckaert-Bellemans constants C_n for the dihedrals. Non-bonded interactions are parametrized in terms of partial charges q_i for Coulombic interactions, and the parameters ϵ_{ij} and σ_{ij}, defining the depth and width of the Lennard-Jones potential, respectively. Parameters are derived by fitting to experimental thermodynamic or structural quantities, and/or to higher (quantum) level calculations.

4.2 Statistical mechanics aspects

Usually, MD simulations are first evolved to reach a certain dynamical equilibrium state in which any memory of the initial conditions has been completely erased. One is then often interested in time averages $\overline{A}$ of some macroscopic quantity $A(\Gamma)$, which itself can be calculated in terms of the microstate Γ of the system. Formally,

we are interested in the quantity

$$\overline{A} = \lim_{\tau\to\infty} \frac{1}{\tau} \int_{t_0}^{t_0+\tau} A(\Gamma(t))\,\mathrm{d}t. \tag{4.10}$$

In a typical MD simulation, this average is simply carried out as an average over a finite number of time steps:

$$\overline{A} \simeq \frac{1}{N_{\text{steps}}} \sum_{k=1}^{N_{\text{steps}}} A(\Gamma_k) \tag{4.11}$$

Alternatively, one might be interested in simulating the system (far) away from equilibrium. In such non-equilibrium simulations, a perturbation is added to the Hamiltonian,

$$H^{\text{NE}} = H + A(\mathbf{r}, \mathbf{p}) \cdot F(t), \tag{4.12}$$

with $F(t)$ as the time-dependent applied field, and A($\mathbf{r}$, $\mathbf{p}$) as a function of the positions and momenta of all particles. A simple example is the generation of shear flow to measure the shear viscosity of a fluid. In this case, at each MD step, an external force is applied along the x-coordinate to each atom, the magnitude of which depends on the y-position of the atom.

The nature of the thermodynamic ensemble decides on some of the macroscopic quantities that can be calculated. The ensemble in turn is defined by the choice of fixed macroscopic parameters (NpT, NVT etc). The next sections describe the techniques to fix the temperature and the pressure, the two quantities also very often fixed in a corresponding laboratory experiment.

4.2.1 Temperature and pressure adjustment

If we define the instantaneous kinetic temperature $\mathcal{T}$ of our N-body MD-system as

$$\frac{3}{2} k_{\text{B}} \mathcal{T} = \left\langle \frac{m\mathbf{v}_i^2}{2} \right\rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{m\mathbf{v}_i^2}{2}, \tag{4.13}$$

then $\mathcal{T}$ averaged over many microstates produced by an MD simulation corresponds to the temperature of the system, i.e.

$$T = \frac{1}{N_{\text{steps}}} \sum_{k} \mathcal{T}(\Gamma_k) \tag{4.14}$$

In molecular dynamics simulations of a micro-canonical NVE ensemble, one follows the classical equations of motion for a system of given particle number N, volume V and total energy E. The averages one computes are hence micro-canonical ensemble averages. Here, every microstate (which all have the same energy, by definition) is accessible with equal probability.

Note that specifying the number density N/V combined with the temperature also specifies the pressure $P = (N/V)k_{\mathrm{B}}T$. This already fully determines the thermodynamic state of a pure liquid. Of course, to get as close as possible to the thermodynamic limit, one needs to try to make N as large as possible, limited only by the computational cost.

Also, one should aim to run sufficiently long to allow proper statistical sampling of all possible particle trajectories, which is needed to justify the use of the ergodic theorem. Depending on what one wants to measure, the required time interval can be quite different. For example, measuring a diffusion constant will require more simulation time than just measuring the average temperature.

However, experimentally much more accessible are systems in which the temperature is constant, not the energy. These NVT ensembles represent the *canonical ensemble*. In the canonical ensemble, each microstate of energy E has the probability

$$p = \frac{1}{Z}\,\mathrm{e}^{-\frac{E}{k_{\mathrm{B}}T}}, \tag{4.15}$$

of occurring, and the expectation value of a macroscopic variable A is given by

$$\langle A\rangle = \frac{1}{Z}\int \mathrm{d}\Gamma\, A(\Gamma)\exp\left(-\frac{E(\Gamma)}{k_{\mathrm{B}}T}\right), \tag{4.16}$$

which is the classical Boltzmann-Gibbs average. The partition function

$$Z = \int \mathrm{d}\Gamma\, \exp\left(-\frac{E}{k_{\mathrm{B}}T}\right) \tag{4.17}$$

acts essentially as a normalization factor.

How do we control the temperature of our MD system? This is of central importance as the temperature is often a key thermodynamic control variable of a MD system. If the current (instantaneous) temperature $\mathcal{T}$ is very different from the desired target temperature T, we can rescale all velocities according to

$$\mathbf{v}_i' = \sqrt{\frac{T}{\mathcal{T}}}\mathbf{v}_i. \tag{4.18}$$

Note that this will not automatically lead to a thermodynamic equilibrium distribution of the velocities, where for each Cartesian component we expect a Gaussian, i.e.

$$p(v_x) = \frac{1}{\sqrt{2\pi}\sigma}\,\exp\left(-\frac{v_x^2}{2\sigma^2}\right) \tag{4.19}$$

with $\sigma^2 = k_{\mathrm{B}}T/m$. Simple rescaling of velocities instead leads to some residual distortions in the distribution function, which do not exactly reflect thermodynamic equilibrium when this procedure is applied.

This can be mitigated by the use of the Nose-Hoover 'thermostat'. Here one adds a thermostat variable as an additional degree of freedom, and uses it to drive the

velocities to the desired temperature through a suitable friction/antifriction term in the equations of motion. For example, we may use:

$$m_i \dot{\mathbf{v}}_i = \mathbf{F}_i - 2\alpha \mathbf{v}_i \tag{4.20}$$

$$\frac{\mathrm{d}\alpha}{\mathrm{d}t} = \frac{1}{\tau_E}\left(\sum_i \frac{1}{2} m_i \mathbf{v}_i^2 - \frac{3}{2} N k_\mathrm{B} T\right) \tag{4.21}$$

where τ_E is a parameter that controls the timescale over which the temperature is regulated and α is the thermostat variable.

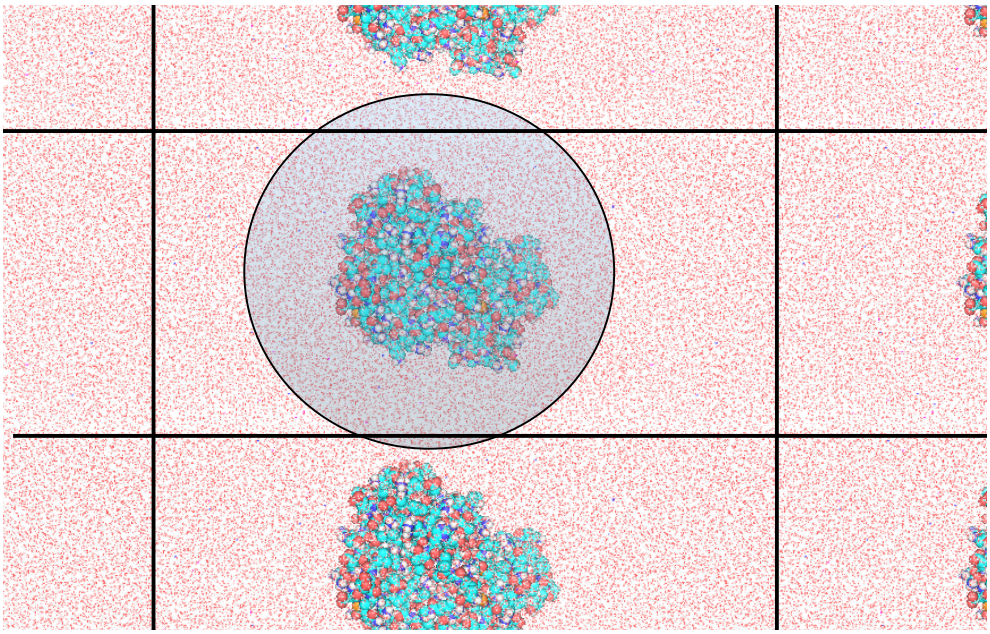
Also important is the NpT-ensemble, in which the volume of the system is adjusted such that the pressure of the system is kept constant. Again, a direct rescaling the coordinates does not result in a thermodynamically defined ensemble as it violates Newtonian mechanics. A solution is the Parrinello-Rahman 'barostat', which analogously to the Nose-Hoover thermostat is an extended ensemble method. The box vectors are subject to an equation of motion, and the equation of motion is coupled to this.

Molecular dynamics simulations can also be used to approximate other types of ensembles. Of particular importance is the *grand canonical ensemble* (μVT-Ensemble) in which the chemical potential μ instead of the particle number is held constant.

4.3 Practical aspects

4.3.1 Boundary conditions

As simulated MD systems only contain particle numbers that are very much smaller than those in any macroscopic sample, surface effects would easily spoil any attempt to approximate the continuum limit if they cannot be efficiently suppressed. For this reason, one usually adopts periodic boundary conditions, because this largely eliminates surface effects in a simple way. The system is periodically replicated in all three Cartesian coordinates. Particles with coordinates beyond the limits of the simulation box will be 'moved' into the simulation system on the other side.



For a fluid of Lennard Jones particles, the box length should be a multiple of σ, such that the interaction of a particle with its periodic image is negligibly small. If the potential is long-ranged, e.g. $v(r) \sim r^{-1}$, periodic boundary conditions can induce errors in particular of processes/observables with large long-range contributions. These include phonons in solids, light scattering factors, or phase transitions.

4.3.2 Initial conditions

In case of a liquid or gas, the initial conditions might not require particular care as their details should anyway be quickly forgotten in a proper MD simulation as it transits to equilibrium. One then usually sets up the initial particle positions on a simple regular grid. However, particles in solids typically have relaxation time scales much longer than the simulation time scale, and are typically placed according to an experimental X-ray structure. Also, care must be taken to start with reasonable molecular bond lengths within each molecule. Otherwise, forces at the first MD steps are extraordinarily high, eventually causing instabilities.

To speed up the settling to thermodynamic equilibrium, it is helpful to draw random initial velocities from an appropriate Maxwellian distribution at the temperature of choice.

4.3.3 Finite range interactions

Formally, calculating all the pairwise forces for N molecules is a $\mathcal{O}(N^2)$ problem, which would rather seriously limit the system sizes that can be studied. However, at $r = 3\sigma$, the Lennard-Jones potential has dropped already to about $v(r) \simeq -0.005\epsilon$, and this low binding energy indicates the weak forces that act at distances this large and beyond. It is hence clear that forces at large distances become negligible, and

the full N^2-interactions do not all have to be calculated. One therefore commonly introduces a cut-off radius into the potential, e.g. in the form of a hard cut-off:

$$v_c(r) = \begin{cases} v(r) & \text{for } r \leq r_c, \\ 0 & \text{otherwise.} \end{cases} \tag{4.22}$$

As an alternative to a hard cut-off at r_c one may also modify the potential such that it smoothly drop to zero, for example in the form

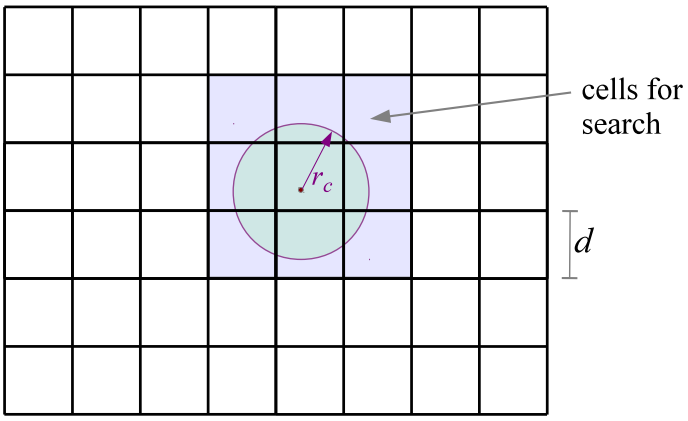
$$v_c(r) = \begin{cases} v(r) - v(r_c) - v'(r_c)(r - r_c) & \text{for } r \leq r_c, \\ 0 & \text{otherwise.} \end{cases} \tag{4.23}$$

Note that the latter form modifies the potential everywhere, which can introduce a small bias in the results.

But independently of how exactly it is carried out, the most important feature of the cut-off is that it reduces the force calculation problem from an unwieldy $\mathcal{O}(N^2)$ to one of order $\mathcal{O}(N)$, provided one has a fast (ideally zeroth-order) method of finding the neighbors that are actually participating in the interactions. Two methods can be used for this:

Search grids

The simplest approach is to use a Cartesian search grid with cell size $d \geq r_c$ overlaid over the system. In a first step, one bins all the particles onto this grid, using for example link-lists as a book-keeping device to organize lists of particles contained in every single cell.



Then, in the calculation of the forces for a given particle, one only considers the particles in the same search grid cell as the target particle, as well as the 26 surrounding cells (in three dimensions). Since we have $d \geq r_c$, this search will already guarantee that we definitely find all interacting neighbors with distance up to r_c, independent of where our target point lies with respect to the target cell.

We note that a search grid with $d = r_c$ still leads to many in principle superfluous look-ups of potential interaction neighbors. This is because one effectively checks all

particles in a volume of $(3d)^3 = (3r_c)^3$, while contributing to the interactions is only the spherical volume $(4\pi/3)r_c^3$ around the target particle. This sphere covers hence only 15.5% of the region that is actually searched. Assuming a roughly uniform distribution of the particles, one then needs to carry out ~ 6 times more distance computations than really used later for the interactions. By using a somewhat finer search grid, one can reduce this overhead somewhat and increase the efficiency of the neighbor search further.

Range search with a tree

A generalization of the search grid idea is to use a *search tree*. Here one considers a hierarchical grid in which the cell size from level to level differs by a factor of 2. This is simply the oct-tree we considered in the gravity calculation with a 'tree-code', except that we do not need the multipole moments here.

The interaction neighbors out to a distance r_c can then simply be found by performing a special tree walk: A tree node is opened when it has a geometric overlap with the search region (which is a sphere or box of size r_c around the target position), otherwise the walk across the branch is terminated. An advantage of this method is that it works well also for variable r_c; in fact, the search region is allowed to be widely different from particle to particle, but the method always works with nearly constant efficiency. In most MD simulations, the particle density is fairly constant, and r_c has a global value, too, hence search trees are not needed. However, in smoothed particle hydrodynamics (SPH), this is different. Here one needs, just like in MD, interaction neighbors out to certain distance, but these distances may vary widely in space and time.

Neighbour lists

For simulations with a fairly homogeneous distribution of particles, neighbour list searches are very efficient, and work equally well for MD and Monte Carlo simulations (see next Chapter). Let us consider a sphere with a cut-off radius r_c around a particle i, e.g. within a Lennard Jones fluid. We now define a larger sphere with radius r_l around the same particle i. A list ist constructed that contains for each particle i all 'neighbours', i.e. all particles within the larger sphere, i.e. within r_l. Now, at the first MD step, the neighbours list is constructed, and used in a number of subsequent MD steps. After some τ_l, the neighbour list is newly constructed. Parameters for this Verlet type of neighbour list are r_c and τ, which are chosen such that particles within the short layer between r_c and r_l do only very rarely move out of the sphere with r_c within the time scale τ. The longer τ the larger should be the buffer. The speed-up for a homogeneous system with as little as 500 particles can be already in the range of a factor of 2-3. Typical values for molecular systems are $r_c = 1\,\mathrm{nm}$, r_l=1.5 nm, with a τ of for example 10 integration steps.

4.3.4 Long-range interactions

Lennard-Jones interactions decay with $1/r$ to the power of 6, i.e. relatively rapidly (see above). Electrostatic interactions, instead, scale with $1/r$, and neglecting long-range forces gives rise to serious errors. Thus, beyond a cut-off r_c, long-range electrostatic forces are commonly still considered, though through computationally cheaper approximative means.

One option is the description by solutions of the Poisson equation. For obtaining the forces in this case, the same techniques as in the gravitational dynamics can be applied. They are discussed in Chapter 5ff. If PME is used for the long-range interaction, also the short-range interactions are modified accordingly. For the Coulombic interactions, e.g., this gives

$$V^{\text{short}}(r) = \mathrm{c}\frac{(\beta r_{ij})}{r_{ij}} q_i q_j \tag{4.24}$$

where β determines the extent to which the potential is calculated as a physical space sum or a Fourier space sum.

4.3.5 Time integration

A very popular scheme for MD simulations is the so-called Verlet time integrator Verlet (1967). This is given by

$$\mathbf{r}_i^{(n+1)} = 2\,\mathbf{r}_i^{(n)} - \mathbf{r}_i^{(n-1)} + \mathbf{a}_i^{(n)}(\Delta t)^2. \tag{4.25}$$

Note that this can in principle work without storing the velocities explicitly, but one needs two copies of the old positions. If one also wants to know the velocities, these are computed in this scheme from

$$\mathbf{v}_i^{(n)} = \frac{1}{2\Delta t}\left(\mathbf{r}_i^{(n+1)} - \mathbf{r}_i^{(n-1)}\right). \tag{4.26}$$

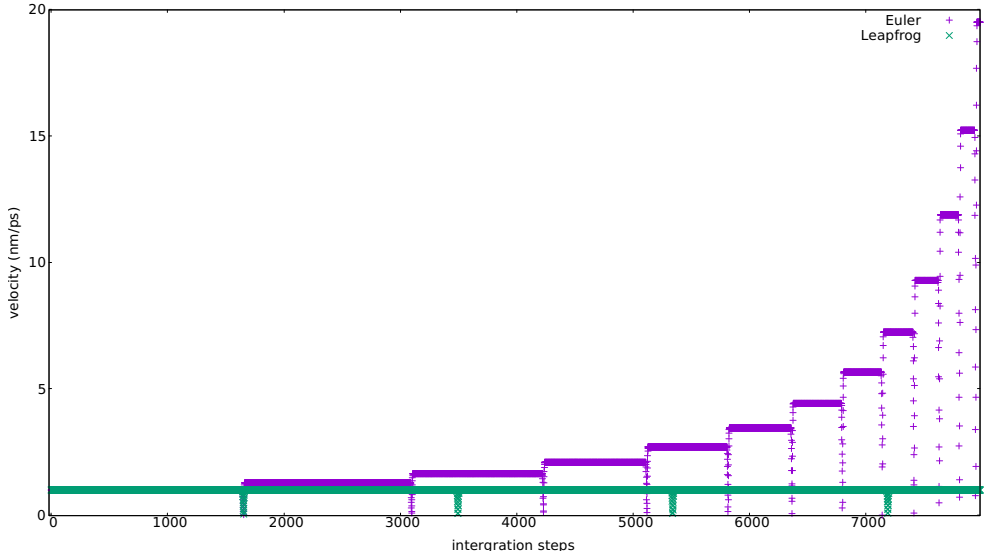
The Verlet scheme may look a bit strange at first, but actually it is really *identical* to the ordinary Leapfrog. The latter can be written as

$$\mathbf{v}_i^{(n+1/2)} = \mathbf{v}_i^{(n)} + \mathbf{a}_i^{(n)}\frac{\Delta t}{2}, \tag{4.27}$$

$$\mathbf{r}_i^{(n+1)} = \mathbf{r}_i^{(n)} + \mathbf{v}_i^{(n+1/2)}\Delta t, \tag{4.28}$$

$$\mathbf{v}_i^{(n+1)} = \mathbf{v}_i^{(n+1/2)} + \mathbf{a}_i^{(n+1)}\frac{\Delta t}{2}. \tag{4.29}$$

If we now plug in equation (4.27) into equation (4.28), consider the same equation a second time for index n instead of $n+1$, and subtract the two from each other, we obtain the Verlet scheme (4.25). This also implies that the Verlet scheme, just like leapfrog, is a second-order method, time-centered, time-reversible, and symplectic (see Chapter 2). The Euler scheme comes with the same computational costs but smaller numerical stability for Newton's equations of motion.



When using Euler but not Verlet, velocities diverge quickly for a set of non-interacting particles in a box with repulsive walls, even if the time steps is chosen smaller than the residence time of particles within the repulsive potential.

The Verlet (or leapfrog) scheme does not handle velocities very satisfactorily. A variance, the velocity-verlet scheme Swope et al. (1982) calculates positions and velocities at each time t,

$$\mathbf{r}_i^{(n+1)} = \mathbf{r}_i^{(n)} + \mathbf{v}_i^{(n)}\Delta t + \frac{\Delta t^2}{2}\mathbf{a}_i^{(n)}, \tag{4.30}$$

$$\mathbf{v}_i^{(n+1)} = \mathbf{v}_i^{(n)} + [\mathbf{a}_i^{(n)} + \mathbf{a}_i^{(n+1)}]\frac{\Delta t}{2}. \tag{4.31}$$

Time steps of MD simulations are chosen such that they are smaller than the fastest motions of the simulated molecular system. Vibrational frequencies

$$\nu = \frac{1}{2\pi}\sqrt{\frac{k}{\mu}} \tag{4.32}$$

where k is the spring constant and μ the reduced mass of a bond, are highest for bonds involving the lightest atoms, that is, hydrogen. Accordingly, MD simulations with chemical bonds to hydrogen, as is the case for water and virtually all organic matter, can only be stably run with a time step of 1 fs or lower. For many applications, however, the intramolecular dynamics of atoms relative to one another is not of interest (and anyways classically not well described). A pragmatic solution is fixing the high frequency interactions such as hydrogen bonds, allowing time steps of $\sim$2 fs. To further reduce the time step, other degrees of freedom can be neglected. A first obvious candidate is the rotation around bonds with three identical atoms, such as methyl (CH_3) groups, or the modelling of more than one atom as one particle, which is called coarse-graining and very problem-dependent.

4.4 Integrating other equations of motion: Langevin and Brownian Dynamics

A stochastic alternative to Newton's equations of motion are Langevin dynamics. In Langevin dynamics, friction and random forces are added to the systematic forces due to the potential energy between particles. One can consider it as a heat bath (analogous to the temperature coupling schemes discussed above), but now here with *random* collisions with the bath.

$$m\ddot{\mathbf{x}}(t) = \nabla V(\mathbf{x}(t)) - \gamma m\dot{\mathbf{x}}(t) + \mathbf{R}(t) \tag{4.33}$$

with γ as the damping constant or collision parameter, and $\mathbf{R}(t)$ a random force vector, which averages to zero over time and follows a stationary Gaussian process. The higher γ, the less the dynamics is governed by the inertia and the more diffusive (Brownian) it is. The stochastic forces typically mimic collisions with solvent molecules (e.g. water around a biomolecule), which are not explicitly considered as particles.

A generalized Verlet scheme can be used to integrate Langevin dynamics.

$$\mathbf{v}_i^{(n+1/2)} = \mathbf{v}_i^{(n)} + m^{-1}\frac{\Delta t}{2}[-\nabla V(\mathbf{r}_i^{(n)} - \gamma m\mathbf{v}_i^{(n)} + \mathbf{R}^{(n)}], \tag{4.34}$$

$$\mathbf{r}_i^{(n+1)} = \mathbf{r}_i^{(n)} + \mathbf{v}_i^{(n+1/2)}\Delta t, \tag{4.35}$$

$$\mathbf{v}_i^{(n+1)} = \mathbf{v}_i^{(n+1/2)} + m^{-1}\frac{\Delta t}{2}[-\nabla V(\mathbf{r}_i^{(n+1)} - \gamma m\mathbf{v}_i^{(n+1)} + \mathbf{R}^{(n+1)}]. \tag{4.36}$$

It reduces to velocity-Verlet (see above) when γ is zero. It is, however, only applicable in the regime of small γ. The superscript of $\mathbf{R}$ simply implies that it is chosen randomly at each integration step.

At high friction, i.e the overdamped regime, inertia can be ignored, $m\ddot{\mathbf{x}}(t) = 0$, resulting in Brownian dynamics of the form

$$\dot{\mathbf{x}}(t) = \frac{\mathbf{D}}{k_\mathrm{B}T}\nabla V(\mathbf{x}(t)) + \mathbf{R}(t) \tag{4.37}$$

$\mathbf{D}$ is a diffusion tensor, or $k_\mathrm{B}T\mathbf{Z}^{-1}$, with $\mathbf{Z}$ as the friction tensor. An integration scheme by Ermak and McCammon is

$$\mathbf{r}_i^{(n+1)} = \mathbf{r}_i^{(n)} + \frac{\Delta t}{k_B T}\mathbf{D}^n\nabla V(\mathbf{r}_i^{(n)}) + \mathbf{R}^{(n)}. \tag{4.38}$$

A few comments with regard to $\mathbf{R}^{(n)}$: It averages to zero over time, $\langle\mathbf{R}^{(n)}\rangle = 0$. and has a variance linearly propertional to $\mathbf{D}^n$, $\langle(\mathbf{R}^{(n)}(\mathbf{R}^{(m)})^T\rangle = 2\mathbf{D}^n\Delta t$. Let's consider a simple example, a particle moving in one dimension in absence of a systematic force, and random numbers r_n being drawn from a uniform distribution in the interval [-0.5, 0.5]. Then the integration algorithm simplifies to

$$\mathbf{r}_i^{(n+1)} = \mathbf{r}_i^{(n)} + (24D\Delta t)^{\frac{1}{2}}r_n \tag{4.39}$$

Note that in BD, for that the variance of the random walk scales linearly in time, the random displacements scale with the square-root of the time-step. The time step for integrating BD is typically much larger than in MD or LD. The definition of the diffusion tensor defines the hydrodynamics of the system.

5 The particle-mesh technique

Direct summation of particle-particle forces scales with $\mathcal{O}(N^2)$. An important approach to accelerate the force calculation for an N-body system lies in the use of an auxiliary mesh, which allows very efficient summation in the reciprocal space.

The slowly converging direct sum is split into two terms, a direct summation over short-range interactions and a summation in reciprocal space of long-range interactions.

$$V = V^{\text{short}} + V^{\text{long}} \tag{5.1}$$

Among the methods to efficiently compute V^{long}, we will first concentrate on the so-called particle-mesh (PM) technique (White et al., 1983; Klypin & Shandarin, 1983) that was originally pioneered in plasma physics (Hockney & Eastwood, 1988). In the last Chapter, we will introduce the particle-mesh Ewald technique, a method that originates from Ewald summation (Ewald, 1921) and is often superior to PM in case of periodic boundary conditions. PM techniques, with or without Ewald, are widely used and have been applied in simulations of molecules, galaxies or, among the first applications, of plasmas. Both procedures in general involve four steps:

1. Construction of a density field ρ on a suitable mesh.

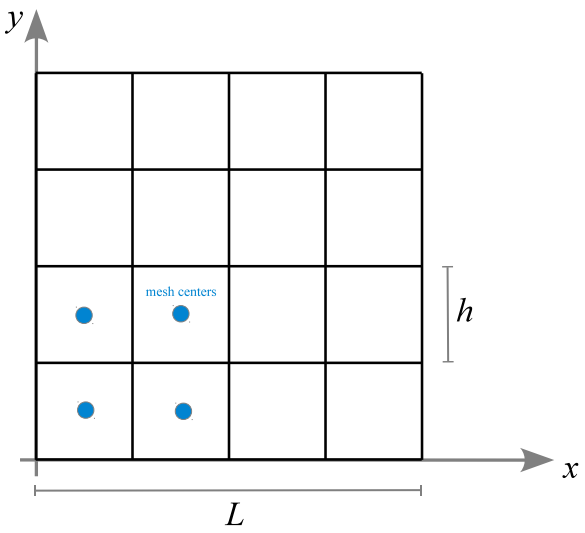
2. Computation of the potential on the mesh by solving the Poisson equation.

3. Calculation of the force field from the potential.

4. Calculation of the forces at the original particle positions.

We shall now discuss these four steps in turn.

5.1 Mass/charge assignment



We want to put N particles with mass m_i and coordinates $\mathbf{x}_i$ $(i = 1, 2, \ldots, N)$ onto a mesh with uniform spacing $h = L/N_\mathrm{g}$. For simplicity, we will assume a cubical calculational domain with extension L and a number of N_g grid cells per dimension. Let $\{\mathbf{x_p}\}$ denote the set of discrete cell-centers, with $\mathbf{p} = (p_x, p_y, p_z)$ being a suitable integer index $(0 \leq p_{x,y,z} < N_g)$. Note that one may equally well identify the $\{\mathbf{x_p}\}$ with the lower left corner of a mesh cell, if this is more practical.

We associate a shape function $S(\mathbf{x})$ with each particle, normalized according to

$$\int S(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1. \tag{5.2}$$

To each mesh-cell, we then assign the fraction $W_\mathbf{p}(\mathbf{x}_i)$ of particle i's mass that falls into the cell indexed by $\mathbf{p}$. This is given by the overlap of the mesh cell with the shape function, namely:

$$W_\mathbf{p}(\mathbf{x}_i) = \int_{\mathbf{x_p}-\frac{\mathbf{h}}{2}}^{\mathbf{x_p}+\frac{\mathbf{h}}{2}} S(\mathbf{x}_i - \mathbf{x})\,\mathrm{d}\mathbf{x} \tag{5.3}$$

The integration extends here over the cubical cell $\mathbf{p}$. By introducing the top-hat function

$$\Pi(\mathbf{x}) = \begin{cases} 1 & \text{for } |\mathbf{x}| \leq \frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{5.4}$$

we can extend the integration boundaries to all space and write instead:

$$W_\mathbf{p}(\mathbf{x}_i) = \int \Pi\left(\frac{\mathbf{x} - \mathbf{x_p}}{h}\right) S(\mathbf{x}_i - \mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{5.5}$$

Note that this also shows that the assignment function W is a convolution of Π with S. The full density in grid cell $\mathbf{p}$ is then given by

$$\rho_{\mathbf{p}} = \frac{1}{h^3} \sum_{i=1}^{N} m_i W_{\mathbf{p}}(\mathbf{x}_i). \tag{5.6}$$

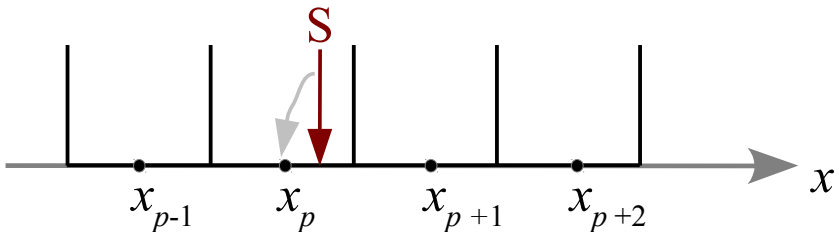
These general formula evidently depend on the specific choice one makes for the shape function $S(\mathbf{x})$. We discuss below a few of the most commonly employed low-order assignment schemes.

5.1.1 Nearest grid point (NGP) assignment

The simplest possible choice for S is a Dirac δ-function. One then gets:

$$W_{\mathbf{p}}(\mathbf{x}_i) = \int \Pi\left(\frac{\mathbf{x} - \mathbf{x_p}}{h}\right) \delta(\mathbf{x}_i - \mathbf{x})\, \mathrm{d}\mathbf{x} = \Pi\left(\frac{\mathbf{x}_i - \mathbf{x_p}}{h}\right). \tag{5.7}$$

In other words, this means that $W_{\mathbf{p}}$ is either 1 (if the coordinate of particle i lies inside the cell), or otherwise it is zero. Consequently, the mass of particle i is fully assigned to exactly one cell – the nearest grid point. The sketch below illustrates this further.



5.1.2 Clouds-in-cell (CIC) assignment

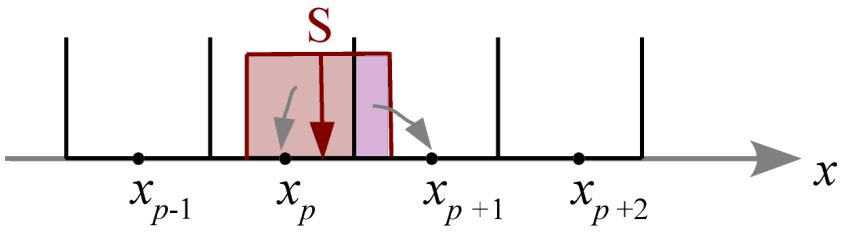
Here one adopts as shape function

$$S(\mathbf{x}) = \frac{1}{h^3} \Pi\left(\frac{\mathbf{x}}{h}\right), \tag{5.8}$$

which is the same cubical 'cloud' shape as that of individual mesh cells. The assignment function is

$$W_{\mathbf{p}}(\mathbf{x}_i) = \int \Pi\left(\frac{\mathbf{x} - \mathbf{x_p}}{h}\right) \frac{1}{h^3} \Pi\left(\frac{\mathbf{x}_i - \mathbf{x}}{h}\right) \mathrm{d}\mathbf{x}, \tag{5.9}$$

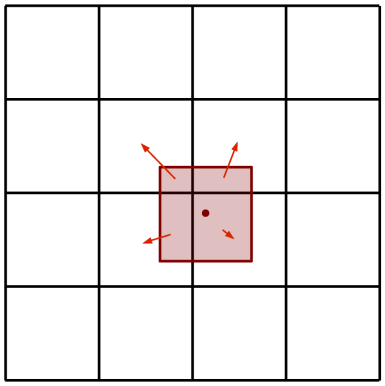
which only has a non-zero (and then constant) integrand if the cubes centred on $\mathbf{x}_i$ and $\mathbf{x_p}$ overlap. How can this overlap be calculated? The 1D sketch below can help to make this clear.



Recall that for one of the dimensions we have $x_p = (p_x+1/2)h$ for $p \in \{0, 1, 2, \ldots, N-1\}$. for a given particle coordinate x_i we may first calculate a 'floating point index' by inverting this relation, yielding $p_f = x_i/h - 1/2$. The index of the left cell of the two cells with some overlap is then given by $p = \lfloor p_f \rfloor$, where the brackets denote the integer floor, i.e. the largest integer not larger than p_f. We may then further define $p^\star \equiv p_f - p$, which is a number between 0 and 1. From the sketch, we see that the length of the overlap of the particle's cloud with the cell p is $h - hp^\star$, hence the assignment function at cell p takes on the value $W_p = 1 - m^\star$ for this location of the particle, whereas the assignment function for the neighboring cell $p + 1$ will take on the value $W_{p+1} = m^\star$.

These considerations readily generalize to 2D and 3D. For example, in 2D, we first assign to the y_i-coordinate of point i a 'floating point index' $q_f = y_i/h - 1/2$. We can then use this to compute a cell index as the integer floor $q = \lfloor q_f \rfloor$, and a fractional contribution $q^\star = q_f - q$. We then obtain the following weights for the assignment of a particle's mass to the four cells its 'cloud' touches in 2D (as sketched):

$$W_{p,q} = (1 - p^\star)(1 - q^\star) \tag{5.10}$$
$$W_{p+1,q} = p^\star(1 - q^\star) \tag{5.11}$$
$$W_{p,q+1} = (1 - p^\star)q^\star \tag{5.12}$$
$$W_{p+1,q+1} = p^\star q^\star \tag{5.13}$$



In the corresponding 3D case, each particle contributes to the weight functions of 8 cells, or in other words, it is spread over 8 cells.

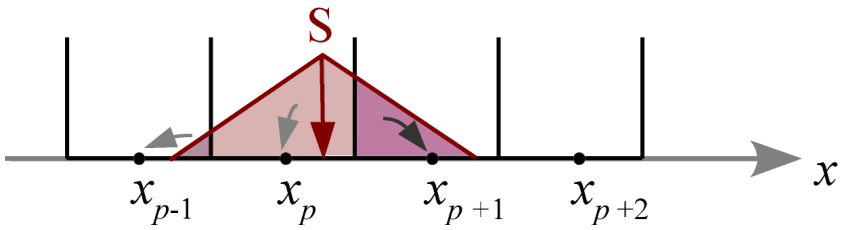
5.1.3 Triangular shaped clouds (TSC) assignment

One can construct a systematic sequence of ever higher-order shape functions by adding more convolutions with the top-hat kernel. For example, the next higher order (in 3D) is given by

$$W_{\mathbf{p}}(\mathbf{x}_i) = \int \Pi\left(\frac{\mathbf{x}-\mathbf{x_p}}{h}\right) \frac{1}{h^3}\Pi\left(\frac{\mathbf{x}_i-\mathbf{x}-\mathbf{x}'}{h}\right) \frac{1}{h^3}\Pi\left(\frac{\mathbf{x}'}{h}\right) \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{x}' \quad (5.14)$$

$$= \frac{1}{h^6}\int \Pi\left(\frac{\mathbf{x}-\mathbf{x_p}}{h}\right) \Pi\left(\frac{\mathbf{x}_i-\mathbf{x}}{h}\right) \Pi\left(\frac{\mathbf{x}'-\mathbf{x}}{h}\right) \mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{x}'. \quad (5.15)$$

This still has a simple geometric interpretation. If one pictures the kernel shape as a triangle with total base length $2h$, then the fraction assigned to a certain cell is given by the area of overlap of this triangle with the cell of interest. The triangle will now in general touch 3 cells per dimension, making an evaluation correspondingly more expensive. In 3D, 27 cells are touched for every particle.



What's then the advantage of using TSC over CIC, if any? Or should one stick with NGP? The assignment schemes differ in the smoothness and differentiability of the reconstructed density field. In particular, for NGP, the assigned density and hence the resulting force jump discontinuously when a particle crosses a cell boundary. The resulting force law will then at best be piece-wise constant.

In contrast, the CIC scheme produces a force that is piece-wise linear and continuous, but has a first derivative that jumps. Here the information where a particle is inside a certain cell is not completely lost, unlike in NGP.

Finally, TSC is yet smoother, and also the first derivative of the force is continuous. Which of these schemes is the preferred choice is ultimately problem-dependent. In most cases, CIC and TSC are quite good options, providing sufficient accuracy with still reasonably small (and hence computationally cheap) assignment kernels. The latter get invariably bigger and bigger for higher-order assignment schemes, which not only is computationally ever more costly but also invokes additional communication overhead in certain parallelization schemes.

5.2 Solving for the gravitational potential

Once the density field is obtained, one would like to solve Poisson's equation

$$\nabla^2\Phi = 4\pi G\,\rho \quad (5.16)$$

Table 5.1: Commonly used shape functions.

Name	Cloud shape $S(x)$	# of cells used	assignment function shape
NGP	$\delta(x)$	1^d	Π
CIC	$\frac{1}{h^d}\Pi\left(\frac{\mathbf{x}}{h}\right)$	2^d	$\Pi \star \Pi$
TSC	$\frac{1}{h^d}\Pi\left(\frac{\mathbf{x}}{h}\right) \star \frac{1}{h^d}\Pi\left(\frac{\mathbf{x}}{h}\right)$	3^d	$\Pi \star \Pi \star \Pi$

and obtain the gravitational potential discretized on the same mesh. There are primarily two methods that are in widespread use for this:

First, there are Fourier-transform based methods which exploit the fact that the potential can be viewed as a convolution of a Green's function with the density field. In Fourier-space, one can then exploit the convolution theorem and cast the computationally expensive convolution into a cheap algebraic multiplication. Due to the importance of this approach, we will discuss it extensively in the next chapter.

Second, there are so-called iterative solvers for Poisson's equation which yield a solution directly in real-space. Simple versions of such iterative solvers use Jacobi or Gauss-Seidel iteration, more complicated ones employ a sophisticated multi-grid approach to speed up convergence. We shall discuss these methods in chapter 7.

5.3 Calculation of the forces

Let's assume for the moment that we already obtained the gravitational potential Φ on the mesh, with one of the methods mentioned above. We would then like to get the acceleration field from

$$\mathbf{a} = -\nabla\Phi. \tag{5.17}$$

One can achieve this by calculating a numerical derivative of the potential by *finite differencing*. For example, the simplest estimate of the force in the x-direction would be

$$a_x(i,j,k) = -\frac{\Phi(i+1,j,k) - \Phi(i-1,j,k)}{2h}, \tag{5.18}$$

where $\mathbf{p} = (i,j,k)$ is a cell index. The truncation error of this expression is $\mathcal{O}(h^2)$, hence the estimate of the derivative is second-order accurate.

Alternatively, one can use *larger stencils* to obtain a more accurate finite difference approximation of the derivative, at greater computational cost. For example, the 4-point expression

$$a_x(i,j,k) = -\frac{1}{2h}\left\{\frac{4}{3}\left[\Phi(i+1,j,k) - \Phi(i-1,j,k)\right] - \frac{1}{6}\left[\Phi(i+2,j,k) - \Phi(i-2,j,k)\right]\right\} \tag{5.19}$$

can be used, which has a truncation error of $\mathcal{O}(h^4)$ (which can be simply proven through Taylor expansion).

For the y- and z-dimensions, corresponding formulae where j or k are varied and the other cell coordinates are held fixed can be used. Whether a second- or fourth-order discretization formula is used depends again on the question which compromise between accuracy and speed one considers best for a given problem. In many collisionless systems, the residual truncation error of a second-order finite difference approximation of the force will be negligible compared to other errors inherent in the simulation scheme, hence the second-order formula would be sufficient in this case.

5.4 Interpolating from the mesh to the particles

Once we have the force field on a mesh, we are not yet fully done. We actually desire the forces at the particle coordinates of the N-body system, not at the coordinates of the mesh cells of our auxiliary computational grid. We are hence left with the problem of interpolating the forces from the mesh to the particle coordinates.

Recall that we defined the density field in terms of mass assignment functions, of the form

$$\rho_{\mathbf{p}} = \frac{1}{h^3} \sum_i W_{\mathbf{p}}(\mathbf{x}_i) = \frac{1}{h^3} \sum_i W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}). \tag{5.20}$$

Here we introduced in the last expression an alternative notation for the weight assignment function.

Assume that we have computed the acceleration field on the grid, $\{\mathbf{a}_{\mathbf{p}}\}$. It turns out to be very important to *use the same* assignment kernel as used in the density construction also for the force interpolation, i.e. the force at coordinate $\mathbf{x}$ for a mass m needs to be computed as

$$\mathbf{F}(\mathbf{x}) = m \sum_{\mathbf{p}} \mathbf{a}_{\mathbf{p}} W(\mathbf{x} - \mathbf{x}_{\mathbf{p}}), \tag{5.21}$$

where W denotes the assignment function used for computing the density field on the mesh. This requirement results from the desire to have a vanishing *self-force*, as well as pairwise antisymmetric forces between every particle pair. The self-force is the force that a particle would feel if just it alone would be present in the system. If numerically this force would be calculated with a non-zero value, the particle would accelerate all by itself, violating momentum conservation. Likewise, for two particles, we would like to have that the forces they mutually exert on each other are equal in magnitude and opposite in direction to each other, such that momentum conservation is manifest.

We now prove that using the same kernels for the mass assignment and force interpolation protects against these numerical artefacts. We start by noting that the acceleration field at a mesh point $\mathbf{p}$ is a linear response to the mass at another mesh point $\mathbf{p}'$ (this can be trivially seen when Fourier techniques are used to solve

the Poisson equation). We can hence express the field as

$$\mathbf{a}_{\mathbf{p}} = \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}')\, h^3 \rho_{\mathbf{p}'} \tag{5.22}$$

with a Green's function $\mathbf{d}(\mathbf{p}, \mathbf{p}')$. This vector-valued Green's function for the force is antisymmetric, i.e. it changes sign when the two points in the argument are swapped. Note that $h^3\rho_{\mathbf{p}'}$ is simply the mass contained in mesh cell $\mathbf{p}'$.

We can now calculate the self-force resulting from the density assignment and interpolation steps:

$$\begin{aligned}
\mathbf{F}_{\text{self}}(\mathbf{x}_i) &= m_i \sum_{\mathbf{p}} W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) \mathbf{a}_{\mathbf{p}} && (5.23)\\
&= m_i \sum_{\mathbf{p}} W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}') h^3 \rho_{\mathbf{p}'} && (5.24)\\
&= m_i \sum_{\mathbf{p}} W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}') m_i W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}'}) && (5.25)\\
&= m_i^2 \sum_{\mathbf{p},\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}') W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}}) W(\mathbf{x}_i - \mathbf{x}_{\mathbf{p}'}) && (5.26)\\
&= 0. && (5.27)
\end{aligned}$$

Here we have started out with the interpolation from the mesh-based acceleration field, and then inserted the expansion of the latter as as convolution over the density field of the mesh. Finally, we put in the density contribution created by the particle i at a mesh cell $\mathbf{p}'$. We then see that the double sum vanishes because of the antisymmetry of $\mathbf{d}$ and the symmetry of the kernel product under exchange of $\mathbf{p}$ and $\mathbf{p}'$. Note that this however only works because the kernels used for force interpolation and density assignment are actually equal – it would have not worked out if they would be different, which brings us back to the point emphasized above.

Now let's turn to the force antisymmetry. The force exerted on a particle 1 of mass m_1 at location $\mathbf{x}_1$ due to a particle 2 of mass m_2 at location $\mathbf{x}_2$ is given by

$$\begin{aligned}
\mathbf{F}_{12} &= m_1 \mathbf{a}(\mathbf{x}_1) = m_1 \sum_{\mathbf{p}} W(\mathbf{x}_1 - \mathbf{x}_{\mathbf{p}}) \mathbf{a}_{\mathbf{p}} && (5.28)\\
&= m_1 \sum_{\mathbf{p}} W(\mathbf{x}_1 - \mathbf{x}_{\mathbf{p}}) \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}') h^3 \rho_{\mathbf{p}'} && (5.29)\\
&= m_1 \sum_{\mathbf{p}} W(\mathbf{x}_1 - \mathbf{x}_{\mathbf{p}}) \sum_{\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}') m_2 W(\mathbf{x}_2 - \mathbf{x}_{\mathbf{p}'}) && (5.30)\\
&= m_1 m_2 \sum_{\mathbf{p},\mathbf{p}'} \mathbf{d}(\mathbf{p}, \mathbf{p}') W(\mathbf{x}_1 - \mathbf{x}_{\mathbf{p}}) W(\mathbf{x}_2 - \mathbf{x}_{\mathbf{p}'}) && (5.31)
\end{aligned}$$

Likewise we obtain for the force experienced by particle 2 due to particle 1:

$$\mathbf{F}_{21} = m_1 m_2 \sum_{\mathbf{p}',\mathbf{p}} \mathbf{d}(\mathbf{p}, \mathbf{p}') W(\mathbf{x}_2 - \mathbf{x}_{\mathbf{p}}) W(\mathbf{x}_1 - \mathbf{x}_{\mathbf{p}'}) \tag{5.32}$$

We may swap the summation indices through relabeling and exploit the antisymmetry of $\mathbf{d}$, obtaining:

$$\mathbf{F}_{21} = -m_1 m_2 \sum_{\mathbf{p}',\mathbf{p}} \mathbf{d}(\mathbf{p},\mathbf{p}') W(\mathbf{x}_1 - \mathbf{x}_{\mathbf{p}}) W(\mathbf{x}_2 - \mathbf{x}_{\mathbf{p}'}) \tag{5.33}$$

Hence we have $\mathbf{F}_{12} + \mathbf{F}_{21} = 0$, independent of where the points are located on the mesh.

6 Force calculation with Fourier transform techniques

Fourier transforms provide a powerful tool for solving certain partial differential equations.

6.1 Ewald summation

The total electrostatic energy of N particles in the unit cell and their periodic images is given by

$$\Phi = \frac{1}{2} \sum_{n} \sum_{i}^{N} \sum_{i}^{N} \frac{q_i q_j}{r_{i,j,n}}, \tag{6.1}$$

where n is the three-dimensional box index vector, and $r_{i,j,n}$ is the actual distance between two charges and not the minimum distance to any image. This sum is conditionally convergent, but converges very slowly.

Ewald summation was first introduced to calculate the electrostatic energy, or Madelung constant, of crystals. The Ewald summation itself does not make use of a mesh but is a summation between the point charges themselves, though now in reciprocal (Fourier) space (see next sections).

6.1.1 1D ionic solid as simple toy example

Let's consider an infinite chain of ions along x with alternating charge $q = \pm 1$ separated by distance $d = 1$. We now want to compute the electrostatic potential at point $\mathbf{r} = (x, y)$,

$$\Phi(\mathbf{r}) = \sum_{n=-\infty}^{\infty} \frac{(-1)^n}{\sqrt{(x-n)^2 + y^2}}. \tag{6.2}$$

The series is convergent and can in principle be used, but is practically *very* slow. And here comes the trick: when a sum is slowly varying in real space, it is typically decaying rapidly in Fourier space. We will take the function $\Phi(\mathbf{r})$ and split it into a local and distant term, $\Phi^{\text{local}}(\mathbf{r})$ and $\Phi^{\text{distant}}(\mathbf{r})$. The latter can be Fourier-transformed into $\tilde{\Phi}^{\text{distant}}(\mathbf{k})$ which decays rapidly in $\mathbf{k}$. This procedure is called Ewald summation.

One could in principle split the total potential $\phi(r) = \frac{1}{r}$ into a short-ranged and long-ranged part, but this would destroy the smoothness of the potential. Instead,

a window function is used to smoothly but rapidly switch from the short to the long-ranged part with increasing distance r.

$$\phi^{\text{short}}(r) = W(r)\frac{1}{r} \tag{6.3}$$

and

$$\phi^{\text{distant}}(r) = [1 - W(r)]\frac{1}{r} \tag{6.4}$$

Now, let's concentrate on the distant term and its summation in Fourier space.

$$\Phi^{\text{distant}}(\mathbf{r}) = \sum_n \phi^{\text{long}}(|\mathbf{r} - \mathbf{r}_n|) = C\sum_\nu \tilde{\phi}^{\text{long}}(\nu) \tag{6.5}$$

with C indicating a prefactor to ensure equality with the real space expression.

An appropriate window function is the error function, which is the integral over a Gaussian up to a value x,

$$\text{erf}(x) = \frac{2}{\sqrt{(\pi)}}\int_0^x e^{-t^2}dt \tag{6.6}$$

$\text{erf}(x)$ is zero at $x = o$ but rapidly changes to 1 for infinite x. Now, the window function in Ewald summation is $1 - \text{erf}(r)$, or $\text{erfc}(x)$, the complementary error function.

This results in a long-range term to the single-ion potential of

$$P_y(x) = \phi^{\text{long}}(r) = \frac{1}{r}\text{erf}(r) = \frac{2}{\sqrt{\pi}}\int_0^1 e^{-r^2u^2}du \tag{6.7}$$

and its Fourier transform (in x not y, with $r^2 = x^2 + y^2$)

$$\tilde{P}_y(k) = \frac{1}{\pi}\int_0^1 \frac{1}{u}e^{-\frac{k^2}{4u^2} - y^2u^2}du. \tag{6.8}$$

While this is not a closed form, it is obvious that the function rapidly decays with increasing k for any y. The sum over the whole chain of oppositely charged ions, the Ewald sum, finally reads

$$\Phi^{\text{distant}}(x, y) = 4\pi \sum_{m=1, m \text{ odd}}^{\infty} \cos(m\pi x)\tilde{P}_y(m\pi). \tag{6.9}$$

To demonstrate the nature of slow versus fast convergence of the brute-force direct summation versus Ewald summation, let's consider the potential at $\mathbf{r} = (0.25, 0.25)$. for an accuracy up to the 6th digit, we need ∼800,000 terms for the direct summation, but only 5 direct and 1 distant summation terms of the Ewald sum!

With Ewald summation, one can calculate the energy of an ion U_i in an ionic crystal. For a simple ionic crystal such as NaCl (cubic lattice), this is given (here now in form of a direct summation) by

$$U_i = \frac{M_a}{R}\frac{q^2}{4\pi\epsilon} \tag{6.10}$$

where M_a is the Madelung's constant.

$$M_a = \sum_i \sum_j \text{sign}(q_i q_j)\frac{R}{r_{ij}} \tag{6.11}$$

Now, for the three-dimensional case mentioned above, with charged particles in a simulation system with periodic images, with

$$V = \frac{1}{2}\sum_n \sum_i^N \sum_i^N \frac{q_i q_j}{r_{i,j,n}}, \tag{6.12}$$

Ewald summation is done by the decomposition also used for the simple case above:

$$\Phi = V_{dir} + V_{rec} + V_0 \tag{6.13}$$

$$\Phi_{dir} = \frac{f}{2}\sum_{i,j}^N \sum_{n_x}\sum_{n_y}\sum_{n_z} q_i q_j \frac{erfc(\alpha r_{i.j.n})}{r_{i.j.\mathbf{n}}} \tag{6.14}$$

$$\Phi_{rec} = \frac{f}{2\pi V}\sum_{i,j}^N q_i q_j \sum_{m_x}\sum_{m_y}\sum_{m_z} \frac{exp(-(\pi\mathbf{m}/\alpha)^2 + 2\pi i\mathbf{m}(\mathbf{r}_i - \mathbf{r}_j))}{\mathbf{m}^2} \tag{6.15}$$

$$\Phi_0 = -\frac{f\alpha}{\sqrt{\pi}}\sum_u^N q_i^2 \tag{6.16}$$

α is a parameter that determines the relative weight of the direct and reciprocal sums. The calculation of the Ewald sum is too slow but can be sped up enormously by using Fast Fourier transforms (see below).

6.1.2 Particle Mesh Ewald

Ewald and PM techniques have in common that the short-ranged potential is summed up in real space, while the long-range potential is calculated in reciprocal space. However, in contrast to the original Ewald summation, particle-mesh Ewald introduces for the long-range part a mesh, just as described above, and uses the discrete density on this mesh for the Ewald summation. PME is significantly more efficient than the Ewald summation. PME differs from the PM technique described above in a number of aspects, among others with respect to the interpolation scheme, or shape function. PME as originally proposed by Darden (Darden et al., 1993) uses (polynomial) spline interpolation.

6.2 Convolution problems

In this subsection we shall consider the particularly important example of using Fourier transforms to solve Poisson's equation, but we note that the basic technique can be used in similar form also for other systems of equations.

Suppose we want to solve Poisson's equation,

$$\nabla^2\Phi = 4\pi G\rho \tag{6.17}$$

for a given mass density distribution ρ. The following considerations similarly apply to the Poisson's equation for electrostatics,

$$\nabla^2\Phi = \frac{\rho}{\epsilon_0} \tag{6.18}$$

for a charge density ρ.

Actually, we can readily write down a solution for a non-periodic space, since we know the Newtonian potential of a point mass, and the equation is linear. The potential is simply a linear superposition of contributions from individual mass elements, which in the continuum can be written as the integration:

$$\Phi(\mathbf{x}) = -\int G\,\frac{\rho(\mathbf{x}')\,\mathrm{d}\mathbf{x}'}{|\mathbf{x}-\mathbf{x}'|}. \tag{6.19}$$

This is recognized to be a convolution integral of the form

$$\Phi(\mathbf{x}) = \int g(\mathbf{x}-\mathbf{x}')\,\rho(\mathbf{x}')\,\mathrm{d}\mathbf{x}', \tag{6.20}$$

where

$$g(\mathbf{x}) = -\frac{G}{|\mathbf{x}|} \tag{6.21}$$

is the *Green's function* of Newtonian gravity. The convolution may also be formally written as:

$$\Phi = g \star \rho. \tag{6.22}$$

We now may recall the *convolution theorem*, which says that the Fourier transform of the convolution of two functions is equal to the product of the individual Fourier transforms of the two functions, i.e.

$$\mathcal{F}(f \star g) = \mathcal{F}(f)\cdot\mathcal{F}(g), \tag{6.23}$$

where $\mathcal{F}$ denotes the Fourier transform and f and g are the two functions. A convolution in real space can hence be transformed to a much simpler, point-by-point multiplication in Fourier space.

There are many problems where this can be exploited to arrive at efficient calculational schemes, for example in solving Poisson's equation for a given density field. Here the central idea is to compute the potential through:

$$\Phi = \mathcal{F}^{-1}\left[\mathcal{F}(g)\cdot\mathcal{F}(\rho)\right], \tag{6.24}$$

i.e. in Fourier space, with $\hat{\Phi}(\mathbf{k}) \equiv \mathcal{F}(\Phi)$, we have the simple equation

$$\hat{\Phi}(\mathbf{k}) = \hat{g}(\mathbf{k}) \cdot \hat{\rho}(\mathbf{k}). \tag{6.25}$$

But how do we solve this in practice?

Let's first assume that we have *periodic boundary conditions* with a box of size L in each dimension. The continuous $\rho(\mathbf{x})$ can in this case be written as a Fourier series of the form

$$\rho(\mathbf{x}) = \sum_{\mathbf{k}} \rho_{\mathbf{k}} \, \mathrm{e}^{i\mathbf{k}\mathbf{x}}, \tag{6.26}$$

where the sum over the $\mathbf{k}$-vectors extends over a discrete spectrum of wave vectors, with

$$\mathbf{k} \in \frac{2\pi}{L} \begin{pmatrix} n_1 \\ n_2 \\ n_3 \end{pmatrix}, \tag{6.27}$$

where n_1, n_2, n_3 are all positive and negative integer numbers. The allowed modes in $\mathbf{k}$ hence form an infinitely extended Cartesian grid with spacing $2\pi/L$ – because of the periodicity condition, only these waves 'fit' into the box. (For a real field such as ρ, there is also a reality constraint of the form $\rho_{\mathbf{k}} = \rho^{\star}_{-\mathbf{k}}$, hence here the modes are not all independent.) The Fourier coefficients can be calculated as

$$\rho_{\mathbf{k}} = \frac{1}{L^3} \int_V \rho(\mathbf{x}) \, \mathrm{e}^{-i\mathbf{k}\mathbf{x}} \mathrm{d}\mathbf{x}, \tag{6.28}$$

where the integration is over one instance of the periodic box.

More generally, the periodic Fourier series features the following orthogonality and closure relationships:

$$\frac{1}{L^3} \int \mathrm{d}\mathbf{x} \, \mathrm{e}^{i(\mathbf{k}-\mathbf{k}')} = \delta_{\mathbf{k},\mathbf{k}'} \tag{6.29}$$

$$\frac{1}{L^3} \sum_{\mathbf{k}} \mathrm{e}^{i\mathbf{k}\mathbf{x}} = \delta(\mathbf{x}), \tag{6.30}$$

where the first relation gives a Kronecker delta, the second a Dirac δ-function.

Let's now look at the Poisson equation again and replace the potential and the density field with their corresponding Fourier series:

$$\nabla^2 \left(\sum_{\mathbf{k}} \Phi_{\mathbf{k}} \, \mathrm{e}^{i\mathbf{k}\mathbf{x}} \right) = 4\pi G \left(\sum_{\mathbf{k}} \rho_{\mathbf{k}} \, \mathrm{e}^{i\mathbf{k}\mathbf{x}} \right) \tag{6.31}$$

We see that we can easily carry out the spatial derivate on the left hand side, yielding:

$$\sum_{\mathbf{k}} \left(-\mathbf{k}^2 \Phi_{\mathbf{k}} \right) \, \mathrm{e}^{i\mathbf{k}\mathbf{x}} = 4\pi G \sum_{\mathbf{k}} \rho_{\mathbf{k}} \, \mathrm{e}^{i\mathbf{k}\mathbf{x}} \tag{6.32}$$

The equality must hold for each of the Fourier modes separately, hence we infer

$$\Phi_{\mathbf{k}} = -\frac{4\pi G}{\mathbf{k}^2}\rho_{\mathbf{k}}. \tag{6.33}$$

Comparing with equation (6.25), this means we have identified the Green's function of the Poisson equation for Newtonian gravity in a periodic space as

$$g_{\mathbf{k}} = -\frac{4\pi G}{\mathbf{k}^2}, \tag{6.34}$$

or of the Poisson's equation for electrostatics as

$$g_{\mathbf{k}} = -\frac{1}{\epsilon_0}\frac{1}{\mathbf{k}^2}, \tag{6.35}$$

6.3 The discrete Fourier transform (DFT)

The above considerations were still for a continuos density field. On a computer, we will usually only have a discretized version of the field $\rho(\mathbf{x})$, defined at a set of (e.g. mass or charge) points. Assuming we have N equally spaced points per dimension, the $\mathbf{x}$ positions may only take on the discrete positions

$$\mathbf{x}_{\mathbf{p}} = \frac{L}{N}\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \quad \text{where } p_1, p_2, p_3 \in \{0, 1, \ldots, N-1\}. \tag{6.36}$$

With the replacement $\mathrm{d}^3\mathbf{x} \to (L/N)^3$, we can cast the Fourier integral (6.28) into a discrete sum:

$$\rho_{\mathbf{k}} = \frac{1}{N^3}\sum_{\mathbf{p}} \rho_{\mathbf{p}}\, \mathrm{e}^{-i\mathbf{k}\mathbf{x}_{\mathbf{p}}}. \tag{6.37}$$

Because of the periodicity and the finite number of density values that is summed over, it turns out that this also restricts the number of $\mathbf{k}$ values that give different answers – shifting $\mathbf{k}$ in any of the dimensions by N times the fundamental mode $2\pi/L$ gives again the same result. We may then for example select as primary set of $\mathbf{k}$-modes the values

$$\mathbf{k}_{\mathbf{l}} = \frac{2\pi}{L}\begin{pmatrix} l_1 \\ l_2 \\ l_3 \end{pmatrix} \quad \text{where } l_1, l_2, l_3 \in \{0, 1, \ldots, N-1\}, \tag{6.38}$$

and the construction of ρ through the Fourier series becomes a finite sum over these N^3 modes. We have now arrived at the *discrete Fourier transform* (DFT), which can equally well be written as:

$$\hat{\rho}_{\mathbf{l}} = \frac{1}{N^3}\sum_{\mathbf{p}} \rho_{\mathbf{p}}\, \mathrm{e}^{-i\frac{2\pi}{N}\mathbf{l}\mathbf{p}} \tag{6.39}$$

$$\rho_{\mathbf{p}} = \sum_{\mathbf{l}} \hat{\rho}_{\mathbf{l}} \, \mathrm{e}^{i \frac{2\pi}{N} \mathbf{l}\mathbf{p}} \tag{6.40}$$

Here are some notes about different aspects of the Fourier pair defined by these relations:

- The two transformations are an invertible linear mapping of a set of N^3 (or N in 1D) complex values $\rho_{\mathbf{p}}$ to N^3 complex values $\hat{\rho}_{\mathbf{l}}$, and vice versa.

- To label the frequency values, $\mathbf{k} = (2\pi/L) \cdot \mathbf{l}$, one often conventionally uses the set $l \in \{-N/2, \ldots, -1, 0, 1, \ldots, \frac{N}{2} - 1\}$ instead of $l \in \{0, 1, \ldots, N-1\}$, which is always possible because shifting l by multiples of N does not change anything because this yields only a 2π phase factor. With this shift, the occurrence of both negative and positive frequencies is made more explicit, and they are arranged quasi-symmetrically in a box in $\mathbf{k}$-space centered on $\mathbf{k} = (0, 0, 0)$. The box extends out to
$$k_{\max} = \frac{N}{2} \frac{2\pi}{L}, \tag{6.41}$$
which is the so-called Nyquist frequency (e.g. Diniz et al., 2002). Adding waves beyond the Nyquist frequency in a reconstruction of ρ on a given grid would add redundant information that could not be unambiguously recovered again from the discretized density field. (Instead, the power in these waves would be erroneously mapped to lower frequencies – this is called *aliasing*, see also the so-called *sampling theorem* and think of the *wagon wheel effect*.)

- Plancherel's theorem relates the quadratic norm of the transform pair, namely
$$\sum_{\mathbf{p}} |\rho_{\mathbf{p}}|^2 = N^3 \sum_{\mathbf{l}} |\hat{\rho}_{\mathbf{l}}|^2. \tag{6.42}$$
This is equally true for the inner product of two different functions, which equals the inner product of the two Fourier transforms (Parseval's theorem). This implies that computations of this sort can be either done in the direct or in the Fourier space, solely depending on what is easier.

- The $1/N^3$ normalization factor could equally well be placed in front of the Fourier series instead of the Fourier transform, or one may split it symmetrically and introduce a factor $1/\sqrt{N^3}$ in front of both. This is just a matter of convention, and both alternative conventions are sometimes used.

- In fact, many computer libraries for the DFT will omit the factor N completely and leave it up to the user to introduce it where needed. Commonly, the DFT library functions define as forward transform of a set of N complex numbers x_j, with $j \in \{0, \ldots, N-1\}$, the set of N complex numbers:
$$y_k = \sum_{j=0}^{N-1} x_j \, \mathrm{e}^{-i \frac{2\pi}{N} j \cdot k}. \tag{6.43}$$

The backwards transform is then defined as

$$y_k = \sum_{j=0}^{N-1} x_j \, \mathrm{e}^{i\frac{2\pi}{N} j\cdot k}. \tag{6.44}$$

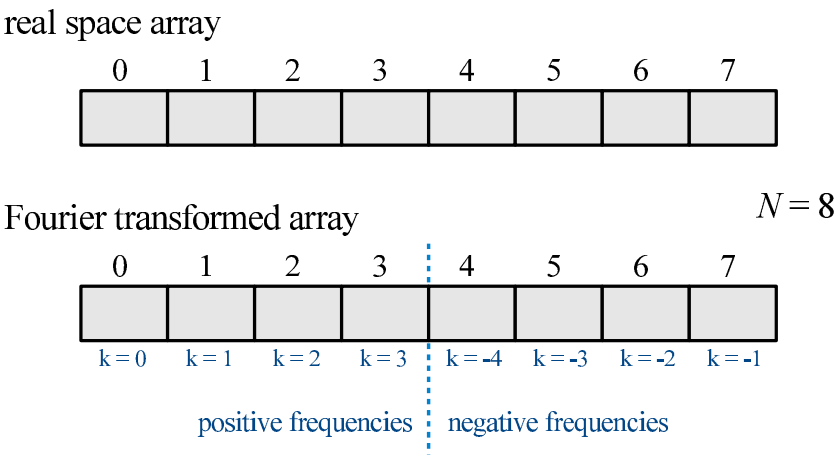
This form of writing the Fourier transform is now nicely symmetric, with the *only difference* between forward and backward transforms being a sign in the exponential function. However, in this case we have that $\mathcal{F}^{-1}(\mathcal{F}(\mathbf{x})) = N\mathbf{x}$, i.e. to get back to the original input vector $\mathbf{x}$ one must eventually divide by N. Note that the multi-dimensional transforms are simply Cartesian products of one-dimensional transforms, i.e. those are obtained as straightforward generalizations of the one-dimensional definition.

- Computing the DFT of N numbers has in principal a computational cost of order $\mathcal{O}(N^2)$. This is because for each of the N numbers one has to calculate N terms and sum them up. Fortunately, in 1965, the *Fast Fourier Transform* (FFT) algorithm (Cooley & Tukey, 1965) has been (re)discovered (Gauss had already known it in principle). This method for calculating the DFT breaks down the problem recursively into smaller and smaller blocks. It turn out that this divide and conquer strategy can reduce the computational cost to $\mathcal{O}(N \log N)$, which is a very significant difference. The result of the FFT algorithm is mathematically identical to the DFT. But actually, in practice the FFT is even better than a direct computation of the DFT, because as an aside the FFT also reduces the round-off error that would otherwise be incurred. It is ultimately only because of the existence of the FFT algorithm that Fourier methods are so widely used in numerical calculations and applicable to even very large problem sizes.

6.4 Storage conventions for the DFT

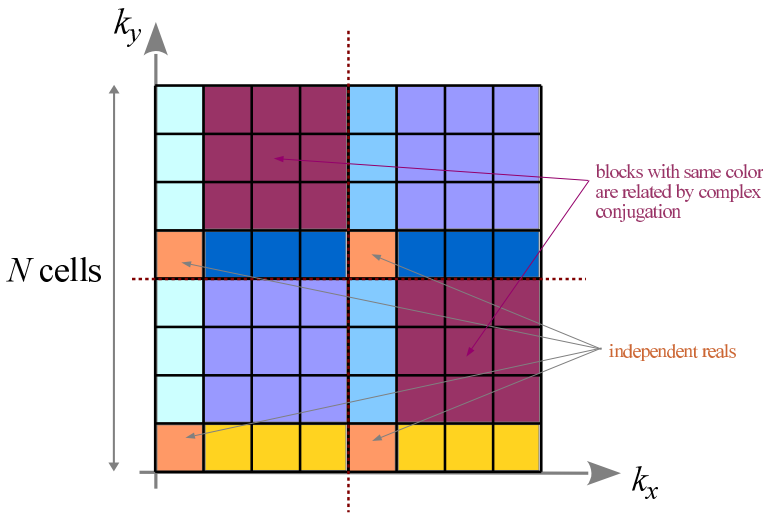
Most numerical libraries for computing the FFT store both the original field and its Fourier transform as simple arrays indexed by $k \in \{0, \ldots, N-1\}$. The negative frequencies will then be stored in the upper half of the array, consistent with what one obtains by subtracting N from the linear index.

An example in 1D for $N = 8$ may help to make this clear:



Correspondingly, in 2D, the grid of real-space values is mapped to a grid of k-space values of the same dimensions. Again, negative frequencies seem to be stored 'backwards', with the smallest negative frequency having the largest linear index, and the most negative frequency appearing as first value past the middle of the mesh. But this is again fully consistent with the translational invariance in k-space with respect to shifts of the indices by multiples of N.

Finally, when we have a real real-space field (such as the physical density), the discrete Fourier transform fulfills a reality constraint of the form $\hat{\rho}_{\mathbf{k}} = \hat{\rho}^{\star}_{-\mathbf{k}}$. This implies a set of relations between the complex values that make up the Fourier transform of ρ, reducing the number of values that can be chosen arbitrarily. How is this manifested in the discrete case? Consider the following sketch, in which regions of like color are related to each other by the reality constraint. Note that $k_x = N/2$ indices are aliased to themselves under complex conjugation, i.e. negating this gives $k_x = -N/2$, but since N can be added, this mode really maps again to $k_x = N/2$. Nevertheless, for the yellow regions there are always different partner cells when one considers the corresponding $-\mathbf{k}$ cell. Only for the red cells, this is not the case; those are mapped to themselves and are hence real due to the reality constraint.



If we now count how many independent numbers we have in the Fourier transformed grid of a 2D real field, we find

$$2\left(\frac{N}{2}-1\right)^2 \times 2 \;+\; 4\left(\frac{N}{2}-1\right) \times 2 \;+\; 4 \times 1 \tag{6.45}$$

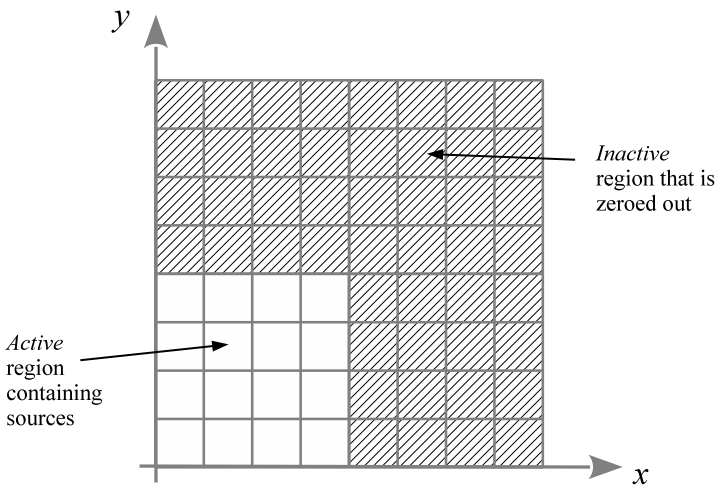
The first term accounts for the two square-shaped regions that have different mirrored regions. Those contain $(\frac{N}{2}-1)^2$ complex numbers, each with two independent real and imaginary values. Then there are 4 different sections of rows and columns that are related to each other by mirroring in k-space. Those contain $\frac{N}{2}-1$ complex numbers each. Finally, there are 4 independent cells that are real and hence account for one independent value each. Reassuringly, the sum of equation (6.45) works out to N^2, which is the result we expect: the number of independent values in Fourier space must be exactly equal to the N^2 real values we started out with, otherwise we would not expect a strictly reversible transformation.

6.5 Non-periodic problems with 'zero padding'

Can we use the FFT/DFT techniques discussed above also to calculate non-periodic force fields? At first, this may seem impossible since the DFT is intrinsically periodic. However, through the so-called zero-padding trick one can circumvent this limitation.

Let's discuss the procedure based on a 2D example (it works also in 1D/3D, of course):

1. We need to arrange our mesh such that the source distribution lives only in one quarter of the mesh, the rest of the density field needs to be zeroed out. Schematically we hence have the following situation:



2. We now set up our desired real-space Green's function, i.e. the response of a mass/charge at the origin. The Green's function for the whole mesh is set-up as $g_{N-i,j} = g_{i,N-j} = g_{N-i,N-j} = g_{i,j}$ where $0 \leq i, j \leq N/2$. This is equivalent to defining g everywhere on the mesh and using as relevant distance the distance to the *nearest periodic image* of the origin. Note that by replicating g with the condition of periodicity, the periodically extended mesh then effectively yields a Green's function that is nicely symmetric around the origin.

3. We now want to carry out the real-space convolution

$$\phi = g \star \rho \tag{6.46}$$

   by using the definition of the discrete, periodic convolution

$$\Phi_{\mathbf{p}} = \sum_{\mathbf{n}} g_{\mathbf{p}-\mathbf{n}} \, \rho_{\mathbf{n}}, \tag{6.47}$$

   where both g and ρ are treated as periodic fields for which adding multiples of N to the indices does not change anything. We see that this sum indeed yields the correct result for the non-periodic potential in the quarter of the mesh that contains our source distribution. This is because the Green's function 'sees' only one copy of the source distribution in this sector; the zero-padded region is big enough to prevent any cross-talk from the (existing) periodic images of the source distribution. This is different in the other three quadrants of the mesh. Here we obtain incorrect potential values that are basically useless and need to be discarded.

4. Given that equation (6.47) yields the correct result in the region of the mesh covered by the sources, we may now just as well use periodic FFTs in the usual way to carry out this convolution quickly! The only downside of this procedure is that it features an enlarged cost in terms of CPU and memory usage. Because we have to effectively double the mesh-size compared to the corresponding periodic problem, the cost goes up by a factor of 4 in 2D, and by a factor of 8 in 3D.

5. We note that James (1977) proposed an ingenious trick that allows a more efficient treatment of isolated source distributions. Through suitably determined correction masses on the boundaries, the memory and CPU cost can be reduced compared to the zero-padding approach described above.

6.5.1 Simple example for zero padding

7 Iterative solvers and the multigrid technique

7.1 The Poisson equation as a linear system of equations

Let's return to the problem of solving the Poisson equation,

$$\nabla^2 \Phi = 4\pi G \rho, \tag{7.1}$$

and consider first the one-dimensional problem, i.e.

$$\frac{\partial^2 \Phi}{\partial x^2} = 4\pi G \rho(x). \tag{7.2}$$

The spatial derivative on the left hand-side can be approximated as

$$\left(\frac{\partial^2 \Phi}{\partial x^2}\right)_i \simeq \frac{\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}}{h^2}, \tag{7.3}$$

where we have assumed that Φ is discretized with N points on a regular mesh with spacing h, and i is the cell index. This means that we have the equations

$$\frac{\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}}{h^2} = 4\pi G \rho_i. \tag{7.4}$$

There are N of these equations, for the N unknowns Φ_i, with $i \in \{0, 1, \ldots, N-1\}$. This means we should in principle be able to solve this algebraically! In other words, the system of equations can be rewritten as a standard linear set of equations, in the form

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{7.5}$$

with a vector of unknowns, $\mathbf{x} = \mathbf{\Phi}$, and a right hand side $\mathbf{b} = 4\pi G\, h^2 \rho$. In the 1D case, the matrix $\mathbf{A}$ (assuming periodic boundary conditions) is explicitly given as

$$\mathbf{A} = \begin{pmatrix} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \cdots & & & \\ & & & 1 & -2 & 1 \\ 1 & & & & 1 & -2 \end{pmatrix} \tag{7.6}$$

Solving equation (7.5) directly constitutes a matrix inversion that can in principle be carried out by LU-decomposition or Gauss elimination with pivoting (e.g. Press et al., 1992). However, the computational cost of these procedures is of order $\mathcal{O}(N^3)$, meaning that it becomes extremely costly, and sooner than later infeasible, already for problems of small to moderate size.

But, if we are satisfied with an approximate solution, then we can turn to iterative solvers that are much faster.

7.2 Jacobi iteration

Suppose we decompose the matrix $\mathbf{A}$ as

$$\mathbf{A} = \mathbf{D} - (\mathbf{L} + \mathbf{U}), \tag{7.7}$$

where $\mathbf{D}$ is the diagonal part, $\mathbf{L}$ is the (negative) lower diagonal part and $\mathbf{U}$ is the upper diagonal part. Then we have

$$[\mathbf{D} - (\mathbf{L} + \mathbf{U})]\,\mathbf{x} = \mathbf{b}, \tag{7.8}$$

and from this

$$\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} + \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}. \tag{7.9}$$

We can use this to define an iterative sequence of vectors $\mathbf{x}^n$:

$$\mathbf{x}^{(n+1)} = \mathbf{D}^{-1}\mathbf{b} + \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^{(n)}. \tag{7.10}$$

This is called Jacobi iteration (e.g. Saad, 2003). Note that $\mathbf{D}^{-1}$ is trivially obtained because $\mathbf{D}$ is diagonal. i.e. here $(\mathbf{D}^{-1})_{ii} = 1/\mathbf{A}_{ii}$.

The scheme converges if and only if the so-called convergence matrix

$$\mathbf{M} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U}) \tag{7.11}$$

has only eigenvalues that are less than 1, or in other words, that the spectral radius $\rho_s(\mathbf{M})$ fullfils

$$\rho_s(\mathbf{M}) \equiv \max_i |\lambda_i| < 1. \tag{7.12}$$

We can easily derive this condition by considering the error vector of the iteration. At step n it is defined as

$$\mathbf{e}^{(n)} \equiv \mathbf{x}_{\text{exact}} - \mathbf{x}^{(n)}, \tag{7.13}$$

where $\mathbf{x}_{\text{exact}}$ is the exact solution. We can use this to write the error at step $n+1$ of the iteration as

$$\mathbf{e}^{(n+1)} = \mathbf{x}_{\text{exact}} - \mathbf{x}^{(n+1)} = \mathbf{x}_{\text{exact}} - \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(n)} = \mathbf{M}\mathbf{x}_{\text{exact}} - \mathbf{M}\mathbf{x}^{(n)} = \mathbf{M}\mathbf{e}^{(n)} \tag{7.14}$$

Hence we find

$$\mathbf{e}^{(n)} = \mathbf{M}^n\mathbf{e}^{(0)}. \tag{7.15}$$

This implies $|\mathbf{e}^{(n)}| \leq [\rho_s(\mathbf{M})]^n |\mathbf{e}^{(0)}|$, and hence convergence if the spectral radius is smaller than 1.

For completeness, we state the Jacobi iteration rule for the Poisson equation in 3D when a simple 2-point stencil is used in each dimension for estimating the corresponding derivatives:

$$\Phi_{i,j,k}^{(n+1)} = \frac{1}{6}\left(\Phi_{i+1,j,k}^{(n)} + \Phi_{i-1,j,k}^{(n)} + \Phi_{i,j+1,k}^{(n)} + \Phi_{i,j-1,k}^{(n)} + \Phi_{i,j,k+1}^{(n)} + \Phi_{i,j,k-1}^{(n)} - 4\pi G h^2 \rho_{i,j,k}\right) \tag{7.16}$$

7.3 Gauss-Seidel iteration

The central idea of Gauss-Seidel iteration is to use the updated values, as soon as they become available, for computing further updated values. We can formalize this as follows. Adopting the same decomposition of $\mathbf{A}$ as before, we can write

$$(\mathbf{D} - \mathbf{L})\mathbf{x} = \mathbf{U}\mathbf{x} + \mathbf{b}, \tag{7.17}$$

from which we obtain

$$\mathbf{x} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x} + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{b}, \tag{7.18}$$

suggesting the iteration rule

$$\mathbf{x}^{(n+1)} = (\mathbf{D} - \mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(n)} + (\mathbf{D} - \mathbf{L})^{-1}\mathbf{b}. \tag{7.19}$$

This seems at first problematic, because we can't easily compute $(\mathbf{D} - \mathbf{L})^{-1}$. But we can modify the last equation as follows:

$$\mathbf{D}\mathbf{x}^{(n+1)} = \mathbf{U}\mathbf{x}^{(n)} + \mathbf{L}\mathbf{x}^{(n+1)} + \mathbf{b}. \tag{7.20}$$

From which we get the alternative form:

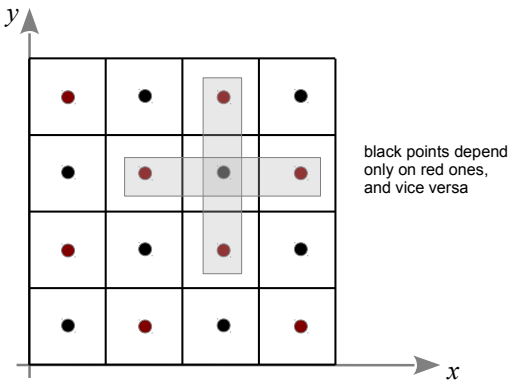
$$\mathbf{x}^{(n+1)} = \mathbf{D}^{-1}\mathbf{U}\mathbf{x}^{(n)} + \mathbf{D}^{-1}\mathbf{L}\mathbf{x}^{(n+1)} + \mathbf{D}^{-1}\mathbf{b}. \tag{7.21}$$

Again, this may seem of little help because it looks like $\mathbf{x}^{(n+1)}$ would only be implicitly given. However, if we start computing the new elements in the first row $i = 1$ of this matrix equation, we see that no values of $\mathbf{x}^{(n+1)}$ are actually needed, because $\mathbf{L}$ has only elements below the diagonal. For the same reason, if we then proceed with the second row $i = 2$, then with $i = 3$, etc., only elements of $\mathbf{x}^{(n+1)}$ from rows above the current one are needed. So we can calculate things in this order without problem and make use of the already updated values. It turns out that this speeds up the convergence quite a bit, with one Gauss-Seidel step often being close to two Jacobi steps.

7.3.1 Red black ordering

A problematic point about Gauss-Seidel is that the equations have to be solved in a specific sequential order, meaning that this part cannot be parallelized. Also, the result will in general depend on which element is selected to be the first. To overcome this problem, one can sometimes use so-called red-black ordering, which effectively is a compromise between Jacobi and Gauss-Seidel.

Certain update rules, such as that for the Poisson equation, allow a decomposition of the cells into disjoint sets whose update rules depend only on cells from other sets. For example, for the Poisson equation, this is the case for a chess-board like pattern of 'red' and 'black' cells.



One can then first update all the black points (which rely only on the red points), followed by an update of all the red points (which rely only on the black ones). In the second of these two half-steps, one can then use the updated values from the first half-step, making it intuitively clear why such a scheme can almost double the convergence rate relative to Jacobi.

7.4 The multigrid technique

Iterative solvers like Jacobi or Gauss-Seidel often converge quite slowly, in fact, the convergence seems to "stall" after a few steps and proceeds only anemically. One also sees that high-frequency errors in the solution are damped out quickly by the iterations, but long-wavelength errors die out much more slowly. Intuitively this is not unexpected: In every iteration, only neighboring points communicate, so the information travels only by one cell (or more generally, one stencil length) per iteration. And for convergence, it has to propagate back and forth over the whole domain a few times.

Idea: By going to a coarser mesh, we may be able to compute an improved initial guess which may help to speed up the convergence on the fine grid (Brandt, 1977). Note that on the coarser mesh, the relaxation will be computationally cheaper (since there are only 1/8 as many points in 3D, or 1/4 in 2D), and the convergence rate should be faster, too, because the perturbation is there less smooth and effectively on a smaller scale relative to the coarser grid.

So schematically, we for example might imagine an iteration scheme where we first iterate the problem $\mathbf{Ax} = \mathbf{b}$ on a mesh with cells $4h$, i.e. for times coarser than the fine mesh. Once we have a solution there, we continue to iterate it on a mesh coarsened with cell size $2h$, and only finally we iterate to solution on the fine mesh h.

A couple of questions immediately come up when we want to work out the details of this basic idea:

1. How to get from a coarse solution to a guess on a finer grid?
2. How to solve $\mathbf{Ax} = \mathbf{b}$ on the coarser mesh?
3. What if there is still an error left with long wavelength on the fine grid?

We clearly need mappings from a finer grid to a coarser one, and vice versa! This is the most important issue to solve.

7.4.1 Prolongation and restriction operations

Coarse-to-fine: This transition is an interpolation step, or in the language of multigrid methods (Briggs et al., 2000), it is called *prolongation*. Let $\mathbf{x}^{(h)}$ be a vector defined on a mesh $\Omega^{(h)}$ with N cells and spacing h, covering our computational domain. Similarly, let $\mathbf{x}^{(2h)}$ be a vector living on a coarser mesh $\Omega^{(2h)}$ with twice the spacing and half as many points per dimension. We now define a linear interpolation operator $\mathbf{I}_{2h}^{h}$ that maps points from the coarser to the fine mesh, as follows:

$$\mathbf{I}_{2h}^{h}\mathbf{x}^{(2h)} = \mathbf{x}^{(h)}. \tag{7.22}$$

A simple example in 1D would be the following:

$$\mathbf{I}_{2h}^{h}: \quad \begin{aligned} x_{2i}^{(h)} &= x_i^{(2h)} \\ x_{2i+1}^{(h)} &= \tfrac{1}{2}(x_i^{(2h)} + x_{i+1}^{(2h)}) \end{aligned} \qquad \text{for } 0 \leq i < \tfrac{N}{2} \tag{7.23}$$

Here, every second point is simply injected from the coarse to the fine mesh, and the intermediate points are linearly interpolated from the neighboring points, which here is a simple arithmetic average.

Fine-to-coarse: The converse mapping represents a smoothing operation, or a *restriction* in multigrid-language. We can define the restriction operator as

$$\mathbf{I}_{h}^{2h}\mathbf{x}^{(h)} = \mathbf{x}^{(2h)}, \tag{7.24}$$

which hence takes a vector defined on the fine grid $\Omega^{(h)}$ to one that lives on the coarse grid $\Omega^{(2h)}$. Again, lets give a simple example in 1D:

$$\mathbf{I}_h^{2h}: \quad x_i^{(2h)} = \frac{x_{2i-1}^{(h)} + 2x_{2i}^{(h)} + x_{2i+1}^{(h)}}{4} \quad \text{for } 0 \leq i < \tfrac{N}{2} \tag{7.25}$$

This evidently is a smoothing operation with a simple 3-point stencil.

One usually chooses these two operators such that the transpose of one is proportional to the other, i.e. they are related as follows:

$$\mathbf{I}_h^{2h} = c\,[\mathbf{I}_{2h}^h]^{\mathrm{T}} \tag{7.26}$$

were c is a real number.

In a shorter notation, the above prolongation operator can be written as

$$\text{1D-prolongation, } \mathbf{I}_{2h}^h: \quad \left]\ \frac{1}{2}\ \ 1\ \ \frac{1}{2}\ \right[\tag{7.27}$$

which means that every coarse point is added with these weights to three points of the fine grid. The fine-grid points accessed with weight 1/2 will get contributions from two coarse grid points. Similarly, the restriction operator can be written with the short-hand notation

$$\text{1D-restriction, } \mathbf{I}_h^{2h}: \quad \left[\ \frac{1}{4}\ \ \frac{1}{2}\ \ \frac{1}{4}\ \right] \tag{7.28}$$

This expresses that every coarse grid point is a weighted sum of three fine grid points.

For reference, we also state the corresponding low-order prolongation and restriction operators in 2D:

$$\text{2D-prolongation, } \mathbf{I}_{2h}^h: \quad \left]\ \begin{matrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{matrix}\ \right[\tag{7.29}$$

$$\text{2D-restriction, } \mathbf{I}_h^{2h}: \quad \left[\ \begin{matrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{matrix}\ \right] \tag{7.30}$$

7.4.2 The multigrid V-cycle

The error vector plays an important role in the multigrid approach. It is defined as

$$\mathbf{e} \equiv \mathbf{x}_{\text{exact}} - \tilde{\mathbf{x}}, \tag{7.31}$$

where $\mathbf{x}_{\text{exact}}$ is the exact solution, and $\tilde{\mathbf{x}}$ the (current) approximate solution.

Another important concept is the *residual*, defined as

$$\mathbf{r} \equiv \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}. \tag{7.32}$$

Note that error and residual are solutions of the linear system, i.e. we have

$$\mathbf{A}\mathbf{e} = \mathbf{r}. \tag{7.33}$$

Coarse-grid correction scheme: We now define a function,

$$\tilde{\mathbf{x}}'^{(h)} = \mathrm{CG}(\tilde{\mathbf{x}}^{(h)}, \mathbf{b}^{(h)}), \tag{7.34}$$

that is supposed to return an improved solution for the problem $\mathbf{A}^{(h)}\mathbf{x}^{(h)} = \mathbf{b}^{(h)}$ on grid level h, based on some starting guess $\tilde{\mathbf{x}}^{(h)}$ and a right hand side $\mathbf{b}^{(h)}$. This so-called *coarse grid correction* proceeds along the following steps:

1. Carry out a relaxation step on h (for example by using one Gauss-Seidel or one Jacobi iteration).

2. Compute the residual: $\mathbf{r}^{(h)} = \mathbf{b}^{(h)} - \mathbf{A}^{(h)}\tilde{\mathbf{x}}^{(h)}$.

3. Restrict the residual to a coarser mesh: $\mathbf{r}^{(2h)} = \mathbf{I}_h^{2h}\,\mathbf{r}^{(h)}$.

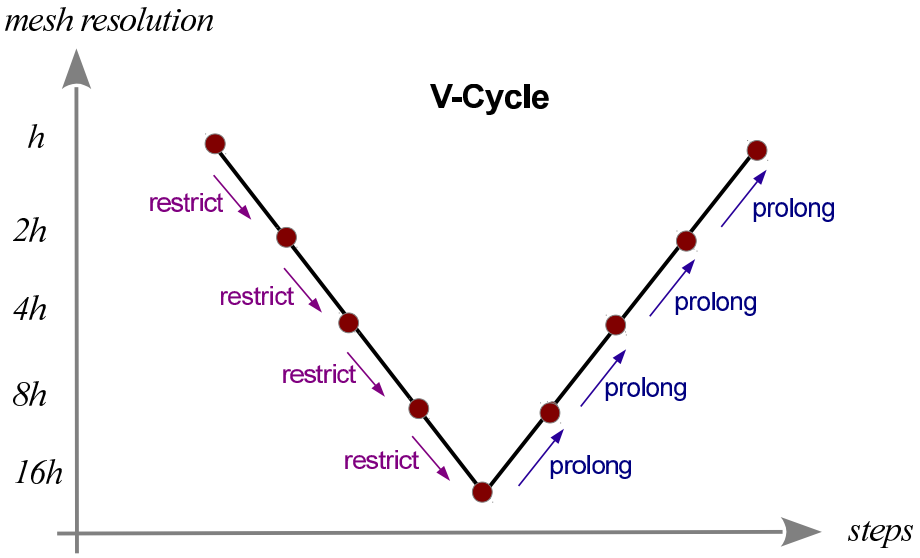
4. Solve $\mathbf{A}^{(2h)}\mathbf{e}^{(2h)} = \mathbf{r}^{(2h)}$ on the coarsened mesh, with $\tilde{\mathbf{e}}^{(2h)} = 0$ as initial guess.

5. Prolong the obtained error $\mathbf{e}^{(2h)}$ to the finer mesh, $\mathbf{e}^{(h)} = \mathbf{I}_{2h}^{h}\,\mathbf{e}^{(2h)}$, and use it to correct the current solution on the fine grid: $\tilde{\mathbf{x}}'^{(h)} = \tilde{\mathbf{x}}^{(h)} + \mathbf{e}^{(h)}$.

6. Carry out a further relaxation step on the fine mesh h.

How do we carry out step 4 in this scheme? We can use recursion! Because what we have to do in step 4 is exactly the function CG(.,.) is defined to do. We however then also need a stopping condition for the recursion, which is simply a prescription that tells us under which conditions we should skip steps 2 to 5 in the above scheme. We can do this by simply saying that further coarsening of the problem should stop once we have reached a minimum number of cells N. At this point we either just do the relaxation steps, or we solve the remaining problem exactly.

V-Cycle: When the coarse grid correction scheme is recursively called, we arrive at the following schematic diagram for how the iteration progresses, which is called a V-cycle:



One finds that the V-cycle rather dramatically speeds up the convergence rate of simple iterative solvers for linear systems of equations. It is easy to show that the computational cost of one V-cycle is of order $\mathcal{O}(N_{\mathrm{grid}})$, where N_{grid} is the number of grid cells on the fine mesh. A convergence to truncation error (i.e. machine precision) requires several V-cycles and involves a computational cost of order $\mathcal{O}(N_{\mathrm{grid}} \log N_{\mathrm{grid}})$. For the Poisson equation, this is the same cost scaling as one gets with FFT-based methods. In practice, good implementations of the two schemes should roughly be equally fast. In cosmology, a multigrid solver is for example used by the MLAPM (Knebe et al., 2001) and RAMSES codes (Teyssier, 2002). An interesting advantage of multigrid is that it requires less data communication when parallelized on distributed memory machines.

One problem we haven't addressed yet is how one finds the operator $\mathbf{A}^{(2h)}$ required on the coarse mesh. The two most commonly used options for this are:

- Direct coarse grid approximation: Here one simply uses the same discrete equations on the coarse grid as on the fine grid, just scaled by the grid resolution h as needed. In this case, the stencil of the matrix does not change.

- Galerkin coarse grid approximation: Here one defines the coarse operator as

$$\mathbf{A}^{(2h)} = \mathbf{I}_h^{2h}\, \mathbf{A}^{(h)}\, \mathbf{I}_{2h}^{h}, \tag{7.35}$$

  which is formally the most consistent way of defining $\mathbf{A}^{(2h)}$, and in this sense optimal. However, computing the matrix in this way can be a bit cumbersome, and it may involve a growing size of the stencil, which then leads to an enlarged computational cost.

7.4.3 The full multigrid method

The V-cycle scheme discussed thus far still relies on an initial guess for the solution, and if this guess is bad, one has to do more V-cycles to reach satisfactory

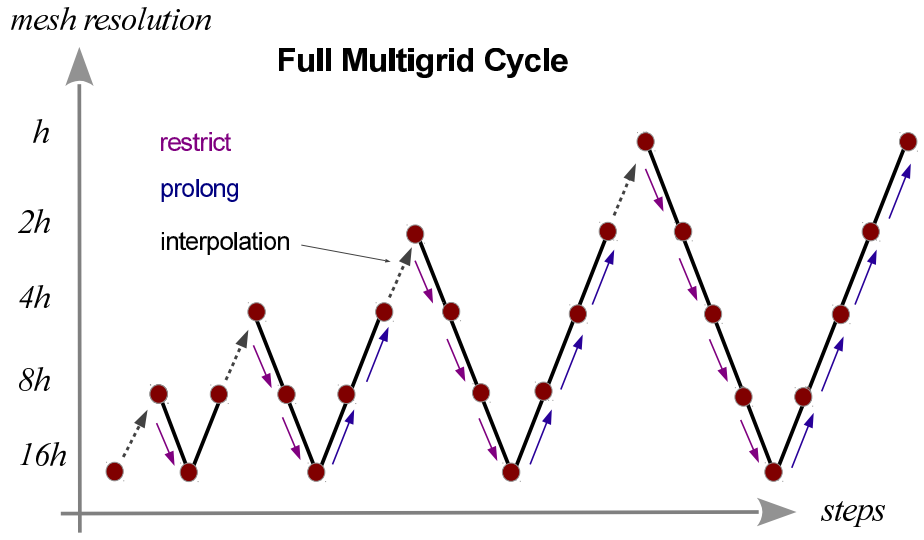
convergence.

This raises the question on how one may get a good guess. If one is dealing with the task of recurrently having to solve the same problem over and over again, with only small changes from solution to solution, as will often be the case in simulation problems, one may be able to simply use the solution from the previous timestep as a guess. In all other cases, one can allude to the following idea: Let's get a good guess by solving the problem on a coarser grid first, and then interpolate the coarse solution to the fine grid as a starting guess.

But at the coarser grid, one is then again confronted with the task to solve the problem without a starting guess. Well, we can then simply recursively apply the idea again, and delegate the finding of a good guess to a yet coarser grid, etc. This then yields the **Full Multigrid Cycle:**

1. Initialize the right hand side on all grid levels, $\mathbf{b}^{(h)}$, $\mathbf{b}^{(2h)}$, $\mathbf{b}^{(4h)}$, ..., $\mathbf{b}^{(H)}$, down to some coarsest level H.
2. Solve the problem (exactly) on the coarsest level H.
3. Given a solution on level i with spacing $2h$, map it to the next level $i+1$ with spacing h and obtain the initial guess $\tilde{\mathbf{x}}^{(h)} = I_{2h}^{h}\, \mathbf{x}^{(2h)}$.
4. Use this starting guess to solve the problem on the level $i+1$ with one V-cycle.
5. Repeat Step 3 until the finest level is reached.

We arrive at the scheme depicted in the sketch.



The computational cost of such a full multigrid cycle is still of order the number of mesh cells, as before.

8 Tree algorithms

Once we have discretized a collisionless fluid in terms of an N-body system, two questions come up:

1. How do we integrate the equations of motion in time?

2. How do we compute the right hand side of the equations of motion, i.e. the gravitational forces?

For the first point, we can simply use one of our ODE integration schemes, preferably a symplectic one since we are dealing with a Hamiltonian system. The second point seems also straightforward at first, as the accelerations (forces) can be readily calculated through *direct summation*:

$$\ddot{\mathbf{x}}_i = -G \sum_{j=1}^{N} \frac{m_j}{[(\mathbf{x}_i - \mathbf{x}_j)^2 + \epsilon^2]^{3/2}} (\mathbf{x}_i - \mathbf{x}_j). \tag{8.1}$$

This calculation is *exact*, but for each of the N equations we have to calculate a sum with N partial forces, yielding a computational cost of order $\mathcal{O}(N^2)$. This quickly becomes prohibitive for large N, and causes a conflict with our urgent need to have a large N!

Perhaps a simple example is in order to show how bad the N^2 scaling really is in practice. Suppose you can do $N = 10^6$ in a month of computer time, which is close to the maximum that one may want to do in practice. A particle number of $N = 10^{10}$ would then already take of order 10 million years.

We hence need faster, approximative force calculation schemes. We shall discuss three different possibilities:

- Hierarchical multipole methods ("tree-algorithms")

- Fourier-transform based methods ("particle-mesh algorithms")

- Iterative solvers for Poisson's equation ("relaxation methods", multigrid-methods)

Various combinations of these approaches may also be used, and sometimes they are also applied together with direct summation on small scales. The latter may also be accelerated with special-purpose hardware (e.g. the GRAPE board), or with graphics processing units (GPUs) that are used as fast number-crunshers.

8.1 Multipole expansion

The central idea is here to use the multipole expansion of a distant group of particles to describe its gravity (Barnes & Hut, 1986), instead of summing up the forces from all individual particles.

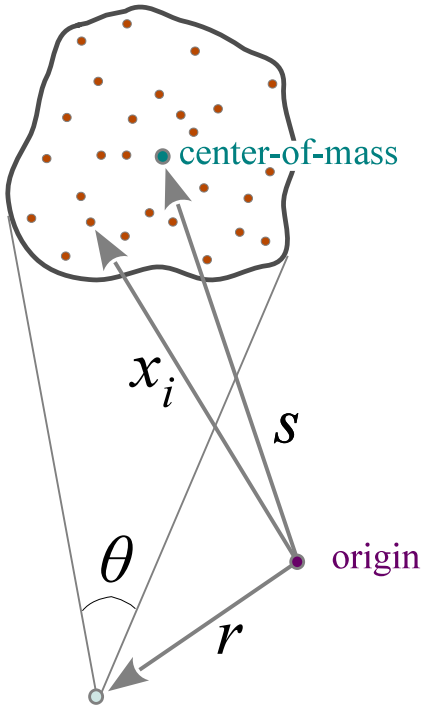


Figure 8.1: Multipole expansion for a group of distant particles. Provided the reference point $\mathbf{r}$ is sufficiently far away, the particles are seen under a small opening angle θ, and the field created by the particle group can be approximated by the monopole term at its center of mass, augmented with higher order multipole corrections if desired.

The potential of the group is given by

$$\Phi(\mathbf{r}) = -G \sum_i \frac{m_i}{|\mathbf{r} - \mathbf{x}_i|}, \tag{8.2}$$

which we can re-write as

$$\Phi(\mathbf{r}) = -G \sum_i \frac{m_i}{|\mathbf{r} - \mathbf{s} + \mathbf{s} - \mathbf{x}_i|}. \tag{8.3}$$

Now we expand the denominator assuming $|\mathbf{x}_i - \mathbf{s}| \ll |\mathbf{r} - \mathbf{s}|$, which will be the case provided the *opening angle* θ under which the group is seen is sufficiently small, see the sketch of Figure 8.1. We can then use the Taylor expansion

$$\frac{1}{|\mathbf{y} + \mathbf{s} - \mathbf{x}_i|} = \frac{1}{|\mathbf{y}|} - \frac{\mathbf{y} \cdot (\mathbf{s} - \mathbf{x}_i)}{|\mathbf{y}|^3} + \frac{1}{2} \frac{\mathbf{y}^T \left[3(\mathbf{s} - \mathbf{x}_i)(\mathbf{s} - \mathbf{x}_i)^T - (\mathbf{s} - \mathbf{x}_i)^2 \right] \mathbf{y}}{|\mathbf{y}|^5} + \dots, \tag{8.4}$$

where we introduced $\mathbf{y} \equiv \mathbf{r} - \mathbf{s}$ as a short-cut. The first term on the right hand side gives rise to the monopole moment, the second to the dipole moment, and the third to the quadrupole moment. If desired, one can continue the expansion to ever higher order terms.

These multipole moments then become properties of the group of particles:

$$\text{monopole:} \quad M = \sum_i m_i \tag{8.5}$$

$$\text{quadrupole:} \quad Q_{ij} = \sum_k m_k \left[3(\mathbf{s} - \mathbf{x}_k)_i (\mathbf{s} - \mathbf{x}_k)_j - \delta_{ij} (\mathbf{s} - \mathbf{x}_k)^2\right] \tag{8.6}$$

The dipole vanishes, because we have done the expansion relative to the center-of-mass, defined as

$$\mathbf{s} = \frac{1}{M} \sum_i m_i \mathbf{x}_i. \tag{8.7}$$

If we restrict ourselves to terms of up to quadrupole order, we hence arrive at the expansion

$$\Phi(\mathbf{r}) = -G \left(\frac{M}{|\mathbf{y}|} + \frac{1}{2} \frac{\mathbf{y}^T \mathbf{Q} \mathbf{y}}{|\mathbf{y}|^5} \right); \quad \mathbf{y} = \mathbf{r} - \mathbf{s}, \tag{8.8}$$

from which also the force can be readily obtained through differentiation. Recall that we expect the expansion to be accurate if

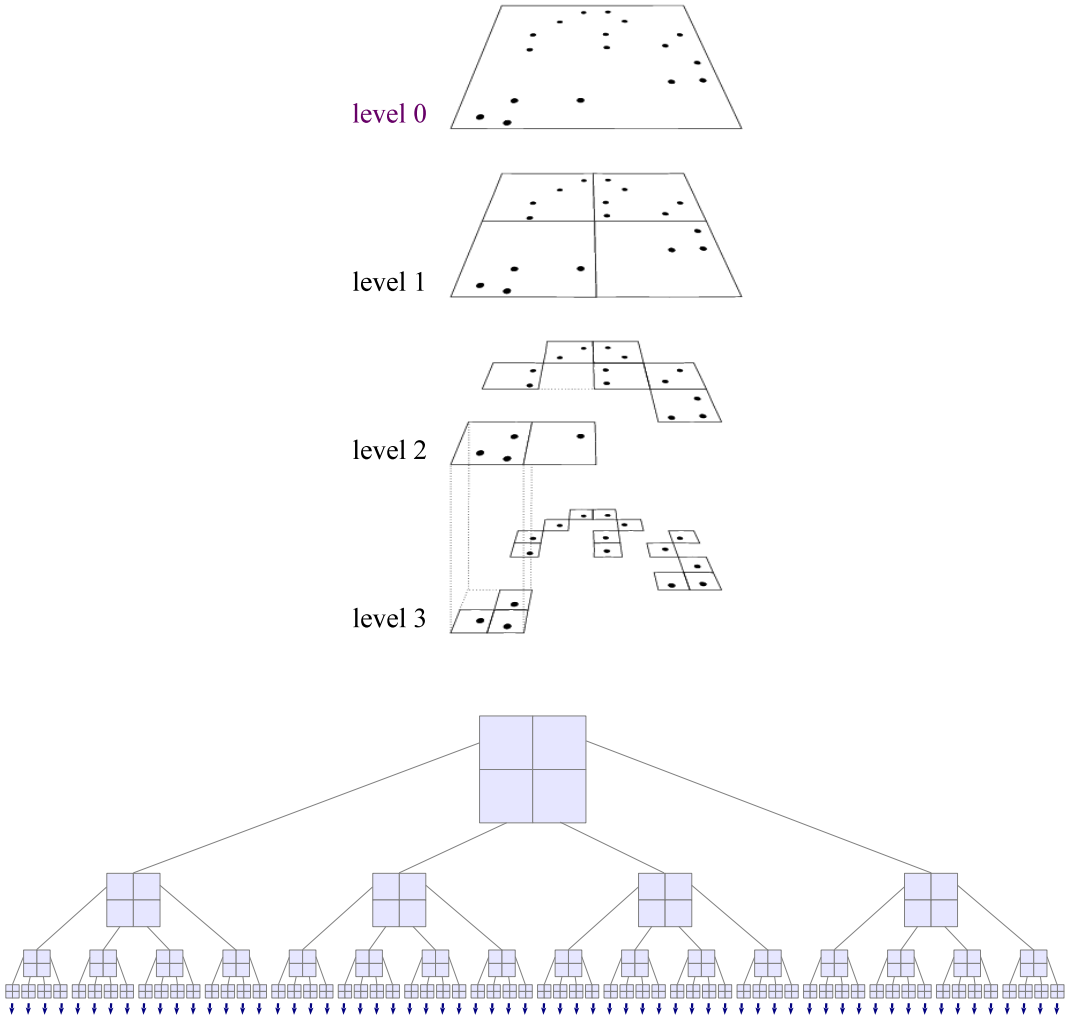
$$\theta \simeq \frac{\langle |\mathbf{x}_i - \mathbf{s}| \rangle}{|\mathbf{y}|} \simeq \frac{l}{y} \ll 1, \tag{8.9}$$

where l is the radius of the group.

8.2 Hierarchical grouping

Tree algorithms are based on a hierarchical grouping of the particles, and for each group, one then pre-computes the multipole moments for later use in approximations of the force due to distant groups. Usually, the hierarchy of groups is organized with the help of a tree-like data structure, hence the name "tree algorithms".

There are different strategies for defining the groups. In the popular Barnes & Hut (1986) oct-tree, one starts out with a cube that contains all the particles. This cube is then subdivided into 8 sub-cubes of half the size in each spatial dimension. One continues with this refinement recursively until each subnode contains only a single particle. Empty nodes (sub-cubes) need not be stored. Here is schematic sketch how this can look like in two dimensions (where one has a 'quad-tree'):



The sketch at the top shows the topological organization of the tree.

- We note that the oct-tree is not the only possible grouping strategy. Sometimes also so-called kd-trees (Stadel, 2001), or other binary trees are used where subdivisions are done along alternating spatial axes.

- An important property of such hierarchical, tree-based groupings is that they are geometrically fully flexible and adjust to any clustering state the particles may have. They are hence automatically adaptive.

- Also, there is no significant slow-down when severe clustering starts.

- The simplest way to construct the hierarchical grouping is to sequentially insert particles into the tree, and then to compute the multipole moments recursively.

8.3 Tree walk

The force calculation with the tree then proceeds by *walking the tree*. Starting at the root node, one checks for every node whether the opening angle under which it is seen is smaller than a prescribed tolerance angle θ_c (see also Salmon & Warren, 1994). If this is the case, the multipole expansion of the node can be accepted, and the corresponding partial force is evaluated and added to an accumulation of the total force. The tree walk along this branch of the tree can then be stopped. Otherwise, one must open the tree node and consider all its sub-nodes in turn.

The resulting force is then approximate in nature by construction, but the overall size of the error can be conveniently controlled by the tolerance opening angle θ_c. If one makes this smaller, more nodes will have to be opened. This will make the residual force errors smaller, but at the price of a higher computational cost. In the limit of $\theta_c \to 0$ one gets back to the expensive direct summation force.

An interesting variant of this approach to walk the tree is obtained by not only expanding the potential on the source side into a multipole expansion, but also around the target coordinate. This can yield a substantial additional acceleration and results in so-called fast multipole methods (FFM). The FALCON code of Dehnen (2000, 2002) employs this approach. A further advantage of the FFM formulation is that force anti-symmetry is manifest, so that momentum conservation to machine precision can be achieved. Unfortunately, the speed advantages of FFM compared to ordinary tree codes are significantly alleviated once individual time-step schemes are considered. Also, FFM is more difficult to parallelize efficiently on distributed memory machines.

Cost of the tree-based force computations

How do we expect the total cost of the tree algorithm to scale with particle number N? For simplicity, let's consider a sphere of size R containing N particles that are approximately homogeneously distributed. The mean particle spacing of these particles will then be

$$d = \left[\frac{(4\pi/3)R^3}{N}\right]^{1/3}. \tag{8.10}$$

We now want to estimate the number of nodes that we need for calculating the force on a central particle in the middle of the sphere. We can identify the computational cost with the number of interaction terms that are needed. Since the used nodes must tessellate the sphere, their number can be estimated as

$$N_{\rm nodes} = \int_d^R \frac{4\pi r^2\,{\rm d}r}{l^3(r)}, \tag{8.11}$$

where $l(r)$ is the expected node size at distance r, and d is the characteristic distance of the nearest particle. Since we expect the nodes to be close to their maximum

allowed size, we can set $l \simeq \theta_c r$. We then obtain

$$N_{\rm nodes} = \frac{4\pi}{\theta_c^3} \ln \frac{R}{d} \propto \frac{\ln N}{\theta_c^3}. \tag{8.12}$$

The total computational cost for a calculation of the forces for all particles is therefore expected to scale as $\mathcal{O}(N \ln N)$. This is a very significant improvement compared with the N^2-scaling of direct summation.

We may also try to estimate the expected typical force errors. If we keep only monopoles, the error in the force per unit mass from one node should roughly be of the order of the truncation error, i.e. about

$$\Delta F_{\rm node} \sim \frac{G M_{\rm node}}{r^2} \theta^2. \tag{8.13}$$

The errors from multipole nodes will add up in quadrature, hence

$$(\Delta F_{\rm tot})^2 \sim N_{\rm node} (\Delta F_{\rm node})^2 = N_{\rm node} \left(\frac{G M_{\rm node}}{r^2} \theta^2 \right)^2 \propto \frac{\theta^4}{N_{\rm node}} \propto \theta^7. \tag{8.14}$$

The force error for a scheme with monopoles only therefore scales as $(\Delta F_{\rm tot}) \propto \theta^{3.5}$, roughly inversely as the invested computational cost. A much more detailed analysis of the performance characteristics of tree codes can be found, for example, in Hernquist (1987).

References

Atkinson, K. (1978), *An introduction to numerical analysis*, Wiley

Balsara, D. S., Rumpf, T., Dumbser, M., Munz, C.-D. (2009), *Efficient, high accuracy ADER-WENO schemes for hydrodynamics and divergence-free magnetohydrodynamics*, Journal of Computational Physics, 228, 2480

Barnes, J., Hut, P. (1986), *A Hierarchical O(NlogN) Force-Calculation Algorithm*, Nature, 324, 446

Bauer, A., Springel, V. (2012), *Subsonic turbulence in smoothed particle hydrodynamics and moving-mesh simulations*, MNRAS, 423, 2558

Bertone, G., Hooper, D., Silk, J. (2005), *Particle dark matter: evidence, candidates and constraints*, Physics Reports, 405, 279

Binney, J., Tremaine, S. (1987), *Galactic dynamics*, Princeton University Press

Binney, J., Tremaine, S. (2008), *Galactic Dynamics: Second Edition*, Princeton University Press

Brandt, A. (1977), *Multi-Level Adaptive Solutions to Boundary-Value Problems*, Mathematics of Computation, 31, 333

Briggs, W. L., Henson, V. E., McCormick, S. F. (2000), *A Multigrid Tutorial*, EngineeringPro collection, Society for Industrial and Applied Mathematics (SIAM, Philadelphia)

Campbell, J. E. (1897), *On a law of combination of operators bearing on the theory of continuous transformation groups*, Proc Lond Math Soc, 28, 381

Chandrasekhar, S. (1943), *Dynamical Friction. I. General Considerations: the Coefficient of Dynamical Friction.*, ApJ, 97, 255

Cooley, J. W., Tukey, J. W. (1965), *An algorithm for the machine calculation of complex Fourier series*, Math. Comp., 19, 297

Courant, R., Friedrichs, K., Lewy, H. (1928), *Über die partiellen Differenzengleichungen der mathematischen Physik*, Mathematische Annalen, 100, 32

Darden, T., York, D., Pedersen, L. (1993), *Particle mesh Ewald: An Nlog(N) method for Ewald sums in large systems*, The Journal of Chemical Physics, 98(12), 10089

Dehnen, W. (2000), *A Very Fast and Momentum-conserving Tree Code*, ApJL, 536, L39

Dehnen, W. (2002), *A Hierarchical O(N) Force Calculation Algorithm*, Journal of Computational Physics, 179, 27

Diniz, P., da Silva, E., Netto, S. (2002), *Digital Signal Processing: System Analysis and Design*, Cambridge University Press

Eckart, C. (1960), *Variation Principles of Hydrodynamics*, Physics of Fluids, 3, 421

Ewald, P. P. (1921), *Die Berechnung optischer und elektrostatischer Gitterpotentiale*, Ann. Phys., 64, 253287

Field, G. B. (1965), *Thermal Instability.*, ApJ, 142, 531

Gingold, R. A., Monaghan, J. J. (1977), *Smoothed particle hydrodynamics - Theory and application to non-spherical stars*, MNRAS, 181, 375

Goldstein, H. (1950), *Classical mechanics*, Addison-Wesley

Hairer, E., Lubich, C., Wanner, G. (2002), *Geometric numerical integration*, Springer Series in Computational Mathematics, Springer, Berlin

Harten, A., Lax, P. D., Van Leer, B. (1983), *On upstream differencing and Godunov-type schemes for hyperbolic conservation laws*, SIAM Review, 25, 35

Hernquist, L. (1987), *Performance characteristics of tree codes*, ApJS, 64, 715

Hockney, R. W., Eastwood, J. W. (1988), *Computer simulation using particles*, Bristol: Hilger

James, R. A. (1977), *The Solution of Poisson's Equation for Isolated Source Distributions*, Journal of Computational Physics, 25, 71

Kirkwood, J. G. (1946), *The Statistical Mechanical Theory of Transport Processes I. General Theory*, Journal of Chemical Physics, 14, 180

Klypin, A. A., Shandarin, S. F. (1983), *Three-dimensional numerical model of the formation of large-scale structure in the Universe*, MNRAS, 204, 891

Knebe, A., Green, A., Binney, J. (2001), *Multi-level adaptive particle mesh (MLAPM): a c code for cosmological simulations*, MNRAS, 325, 845

Kolmogorov, A. N. (1941), *Dissipation of Energy in the Locally Isotropic Turbulence*, Proceedings of the USSR Academy of Sciences, 32, 16

Landau, L. D., Lifshitz, E. M. (1959), *Fluid mechanics*, Course of theoretical physics, Oxford: Pergamon Press

LeVeque, R. J. (2002), *Finite volume methods for hyperbolic systems*, Cambridge University Press

Lucy, L. B. (1977), *A numerical approach to the testing of the fission hypothesis*, AJ, 82, 1013

Miyoshi, T., Kusano, K. (2005), *A multi-state HLL approximate Riemann solver for ideal magnetohydrodynamics*, Journal of Computational Physics, 208, 315

Mo, H., van den Bosch, F. C., White, S. (2010), *Galaxy Formation and Evolution*, Cambridge University Press

Monaghan, J. J. (1992), *Smoothed particle hydrodynamics*, ARA&A, 30, 543

Ollivier-Gooch, C. F. (1997), *Quasi-ENO Schemes for Unstructured Meshes Based on Unlimited Data-Dependent Least-Squares Reconstruction*, Journal of Computational Physics, 133, 6

Pope, S. B. (2000), *Turbulent Flows*, Cambridge University Press

Press, W. H., Teukolsky, S. A., Vetterling, W. T., Flannery, B. P. (1992), *Numerical recipes in C. The art of scientific computing*, Cambridge: University Press, 1992, 2nd ed.

Price, D. J. (2012), *Smoothed Particle Hydrodynamics: Things I Wish My Mother Taught Me*, in *Advances in Computational Astrophysics: Methods, Tools, and Outcome*, edited by R. Capuzzo-Dolcetta, M. Limongi, A. Tornambè, volume 453 of *Astronomical Society of the Pacific Conference Series*, 249

Pringle, J. E., King, A. (2007), *Astrophysical Flows*, Cambridge University Press

Rankine, W. J. M. (1870), *On the thermodynamic theory of waves of finite longitudinal disturbances*, Philosophical Transactions of the Royal Society of London, 160, 277288

Rusanov, V. V. (1961), *Calculation of interaction of non-steady shock waves with obstacles*, J. Comput. Math. Phys. USSR, 1, 267

Ryckaert, J.-P., Bellemans, A. (1975), *Molecular dynamics of liquid n-butane near its boiling point*, Chemical Physics Letters, 30(1), 123

Saad, Y. (2003), *Iterative Methods for Sparse Linear Systems: Second Edition*, Society for Industrial and Applied Mathematics

Saha, P., Tremaine, S. (1992), *Symplectic integrators for solar system dynamics*, AJ, 104, 1633

Salmon, J. K., Warren, M. S. (1994), *Skeletons from the treecode closet*, J. Comp. Phys., 111, 136

Schaal, K., Bauer, A., Chandrashekar, P., Pakmor, R., Klingenberg, C., Springel, V. (2015), *Astrophysical hydrodynamics with a high-order discontinuous Galerkin scheme and adaptive mesh refinement*, MNRAS, 453, 4278

Shu, F. H. (1992), *The physics of astrophysics. Volume II: Gas dynamics.*, University Science Books, Mill Valley, CA

Springel, V. (2005), *The cosmological simulation code GADGET-2*, MNRAS, 364, 1105

Springel, V. (2010), *Smoothed Particle Hydrodynamics in Astrophysics*, ARA&A, 48, 391

Springel, V., Hernquist, L. (2002), *Cosmological smoothed particle hydrodynamics simulations: the entropy equation*, MNRAS, 333, 649

Stadel, J. G. (2001), *Cosmological N-body simulations and their analysis*, Ph.D. thesis, University of Washington

Stoer, J., Bulirsch, R. (2002), *Introduction to Numerical Analysis*, Texts in Applied Mathematics, Springer

Stone, J. M., Gardiner, T. A., Teuben, P., Hawley, J. F., Simon, J. B. (2008), *Athena: A New Code for Astrophysical MHD*, ApJS, 178, 137

Strang, G. (1968), *On the Construction and Comparison of Difference Schemes*, SIAM J. Numer. Anal., 5, 506

Swope, W. C., Andersen, H. C., Berens, P. H., Wilson, K. R. (1982), *A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters*, The Journal of Chemical Physics, 76(1), 637

Teyssier, R. (2002), *Cosmological hydrodynamics with adaptive mesh refinement. A new high resolution code called RAMSES*, A&A, 385, 337

Toro, E. (1997), *Riemann solvers and numerical methods for fluid dynamics*, Springer

van Leer, B. (1984), *On the Relation Between the Upwind-Differencing Schemes of Godunov, Enguist, Osher and Roe*, SIAM J. Sci. Stat. Comput., 5, 1

van Leer, B. (2006), *Upwind and High-Resolution Methods for Compressible Flow: From Donor Cell to Residual-Distribution Schemes*, Communications in Computational Physics, 1, 192

Verlet, L. (1967), *Computer "Experiments" on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules*, Phys. Rev., 159, 98

White, S. D. M., Frenk, C. S., Davis, M. (1983), *Clustering in a neutrino-dominated universe*, ApJL, 274, L1