Computational physics

Dynamics of finite quantum systems

Lecture notes

(WORK IN PROGESS)

Armin Scrinzi

Version 2014

(2011 contributions by Francesco Alaimo and Urs Waldmann)

July 14, 2014

Contents

Chapter 1

Introduction

1.1 Where is computational physics used?

March 4 issue of Physical Review Letters

26 out of 47 publications exclusively or essentially rely on large scale computations!

In every single section:

- General Physics: Statistical, Quantum Mechanics, Quantum Information, etc.
- Gravitation and Astrophysics
- Elementary Particles and Fields
- Atomic, Molecular, and Optical Physics
- Nonlinear Dynamics, Fluid Dynamics, Classical Optics, etc.
- Plasma and Beam Physics
- Condensed Matter: Structure, etc.
- Condensed Matter: Electronic Properties, etc.
- Soft Matter, Biological, and Interdisciplinary Physics

1.2 How are computers used?

Rather incomplete list

- Fluid dynamics, plasma simulations: particle-in-cell (PIC) codes
- Precision calculations of atomic and molecular properties: Hartree Fock (HF), Configuration Interaction (CI)
- Solid state electronic structure: density functional theory (DFT), dynamical mean field theory (DMF)
- "Soft matter": molecular dynamics (MD), multi-scale techniques
- Solution of model systems: e.g. Gross-Pitaevsky (GP), Ising problem, Fermi liquid
- Transport in solids and nano-structures: Greens-function techniques
- Large statistical systems: Monte-Carlo techniques
- Qauntum Chromo Dynamics (QCD): e.g. lattice gauge theory
- etc. ...

1.3 Analytical and computational physics

1.3.1 Skills and techniques

Sorted by importance:

- The best possible intuitive understanding of the physics.
- Clear physical modelling.
- Mathematical understanding of the equations: existence of solutions, operators, eigenvalues, stability, well-posedness, conserved quantities(norm, energy, currents, symmetries).
- Visualization
- Knowledge of numerical techniques: eigenproblems, linear algebra, random numbers, approximation techniques, integration, differentiation, matrix structure, computational complexity, complex calculus, iterative methods, random numbers.
- Programming: data-structures, object oriented programming, re-usable code, documentation, code management.
- Awareness of existing software: LAPACK, FFTW, integration routines, special functions
- Understanding of computer architecture (for large-scale problems): data-locality, communication overhead, CPU vs. memory, parallelism

1.3.2 Computational and analytical techniques

- Computational and analytical work are closely entangled: large part of modern theory is for computational evaluation.
- Computational physics continues where analytical techniques reach their limits.
- Computational techniques can bridge gaps, i.e. we return to analysis after we have explored using computational means.
- We cannot just "solve" and equation on the computer in the same sense as we solve it analytically.
- Instead of wondering how large a given effect might be, we can often very quickly get computational estimates that save us a lot of handwaving and fruitless discussions.
- Solving a physics problem on the computer is hardly ever simple.
- Understanding compuational results often is not much easier than understanding experimental results.
- "The Schrödinger equation is useless" (...but is is beautiful and it is true).
- Certain systems (e.g. plasmas) can essentially only be "simulated"
- (Good) computational methods are exact: we can get solutions to a given mathematical problem to in principle any pre-set accurcy. Computational physics is not "fuzzy".
- The best computation is the one that takes maximal advantage of all analytic theory in reach.
- Good computation requires good programming.

1.3.3 Python

- Easy to use
- Object oriented
- Many packages available — graphics, linear algebra, ...
- Free and open source
- Platform independent

Problems

- not fast
- hides important technical issues

For real programming, need to use C++ in addition.

Chapter 2

Single-particle stationary Schrödinger equation

2.1 The 1-d Harmonic oscillator on a grid

$$[-\frac{1}{2}\partial_x^2 + \frac{1}{2}x^2]\Psi_n(x) = E_n\Psi_n(x) \tag{2.1}$$

Energies: $E_n = n + 0.5, \quad n = 0, 1, 2, 3, \ldots$
Eigenfunctions: $\Psi_n(x) = H_n(x)e^{-x^2/2}/N_n, \quad N_n = \pi^{-1/4}(2^n N!)^{-1}$

2.1.1 Finite difference schemes

Want to know $\Psi(x_i) := \Psi_i, i = 0, 1, \ldots, N-1$ on an equidistant grid of points $x_0 = -L/2$, $x_{N-1} = L/2$ with distance between grid points $h = L/(N-1)$.

(First) derivative

$$\Psi'(x_i) = \lim_{h\downarrow 0} \frac{\Psi(x_i) - \Psi(x_i - h)}{h} \approx \frac{\Psi(x_i) - \Psi(x_i - h)}{h} \tag{2.2}$$

How accurate: Taylor series $\Psi(x) = \sum_{n=0}^{\infty} \Psi^{(n)}(x_i)\frac{(x-x_i)^n}{n!}$

$$\frac{\Psi(x_i) - \Psi(x_i - h)}{h} = \Psi^{(1)} + \sum_{n=2}^{\infty} \Psi^{(n)}(x_i)\frac{(-h)^{n-1}}{n!} \tag{2.3}$$

Assume $\Psi^{(n)}$ do not grow too much: $\Psi^{(n)} \leq Cn!$:

$$\Rightarrow \mathcal{E}(h) := \left|\frac{\Psi(x_i) - \Psi(x_i - h)}{h} - \partial_x\Psi(x_i)\right| < C\frac{h}{1-h} \approx Ch \text{ for small } h. \tag{2.4}$$

Problem 1.1: Compute the error $\mathcal{E}(h)$ of the first order finite difference scheme

$$\partial_x\Psi(x) = \frac{\Psi(x+h) - \Psi(x)}{h} + \mathcal{E}(h) \tag{2.5}$$

for the Gaussian function $\exp(-x^2)$ and for $1/x$ at $x = 0, 1$.

In matrix-vector notation:

$$\begin{pmatrix} \Psi'_0 \\ \Psi'_1 \\ \Psi'_2 \\ \vdots \\ \Psi'_{N-2} \\ \Psi'_{N-1} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & \cdots & -1 & 1 & 0 \\ 0 & & \cdots & & -1 & 1 \end{pmatrix} \begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_{N-2} \\ \Psi_{N-1} \end{pmatrix} \tag{2.6}$$

$$\vec{\Psi}' = \widehat{D}_1 \vec{\Psi} \tag{2.7}$$

Note: "Inaccuracy" at the beginning of the intervals implies boundary condition $\Psi_{-1} = 0$
Note: Matrix $\widehat{D}$ is not anti-symmetric (but ∂_x is anti-hermitian)

Alternative:

$$\Psi'_i = \frac{1}{h}(\Psi_{i+1} - \Psi_i) \tag{2.8}$$

equivalent in the limit $h \downarrow 0$.

$$\begin{pmatrix} \Psi'_0 \\ \Psi'_1 \\ \Psi'_2 \\ \vdots \\ \Psi'_{N-2} \\ \Psi'_{N-1} \end{pmatrix} = \frac{1}{h} \begin{pmatrix} -1 & 1 & 0 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ 0 & 0 & -1 & 1 & \dots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & \cdots & 0 & -1 & 1 \\ 0 & & \cdots & & 0 & -1 \end{pmatrix} \begin{pmatrix} \Psi_0 \\ \Psi_1 \\ \Psi_2 \\ \vdots \\ \Psi_{N-2} \\ \Psi_{N-1} \end{pmatrix} \tag{2.9}$$

$$\vec{\Psi}' = -\widehat{D}_1^T \vec{\Psi} \tag{2.10}$$

Note: "Inaccuracy" at the end of the intervals Implies boundary condition $\Psi_N = 0$

Problem 1.2:

The finite difference scheme

$$\Psi'_i = \frac{\Psi_{i+1} - \Psi_{i-1}}{2h} + \mathcal{E}(h) \tag{2.11}$$

is an anti-hermitian approximation of the derivative. Give the error estimate $\mathcal{E}(h)$ of the finite-difference scheme. Write the scheme as a matrix $\widehat{D}_2$.

We can construct the second derivative as

$$\partial_x^2 \approx -\widehat{D}_1^T \widehat{D}_1 = -\frac{1}{h^2} \begin{pmatrix} 2 & -1 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & \cdots & -1 & 2 & -1 \\ 0 & & \cdots & & -1 & 2 \end{pmatrix} \tag{2.12}$$

or

$$(-\partial_x^2\Psi)_i = \frac{2\Psi_i - \Psi_{i-1} - \Psi_{i+1}}{h^2} \tag{2.13}$$

Note: Is hermitian by construction: $(A^TA)^T = A^T(A^T)^T) = A^TA$
Note: One verifies the error $\mathcal{E}(h) = \mathcal{O}(h^3)$. The error is in third order, although we started from second order error in the first derivative. we are lucky in this particular case, it is not a general procedure: with approximations $\partial_x \approx \widehat{D}$, the approximation $-\partial_x^2 \approx \widehat{D}^T\widehat{D}$ is in general *not* higher order than $\widehat{D}$.

Approximate Schrödinger equation

$$\frac{1}{2}[\widehat{D}_1^T\widehat{D}_1 + \widehat{V}]\vec{\Psi}^{(n)} = \vec{\Psi}^{(n)}\tilde{E}^{(n)} \tag{2.14}$$

with

$$\widehat{V} = \begin{pmatrix} x_0^2 & 0 & 0 & 0 & \dots & 0 \\ 0 & x_1^2 & 0 & 0 & \dots & 0 \\ 0 & 0 & x_2^2 & 0 & \dots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & \cdots & 0 & x_{N-2}^2 & 0 \\ 0 & & \cdots & & 0 & x_{N-1}^2 \end{pmatrix} \tag{2.15}$$

This finite matrix eigenvalue problem can be solved by standard software. One obtains approximate eigenvalues $\tilde{E}^{(n)}$ and the corresponding eigenvectors $\vec{\Psi}^{(n)}$.

2.1.2 Mathematics

Remember some facts from linear algebra (or check the Golub/vanLoan for it):

Unitary matrix

$\widehat{U}^\dagger\widehat{U} = \widehat{U}\widehat{U}^\dagger = \mathbf{1} \Rightarrow U^\dagger = U^{-1}$

Spectral representation of a matrix

Let A be a hermitian $N \times N$ matrix

$$\widehat{A}^\dagger = \widehat{A} : (\widehat{A}_{ij})^* = \widehat{A}_{ji} \tag{2.16}$$

Then there exists a unitary matrix $\widehat{U}$ and a diagonal matrix

$$(d_A)_{mn} := \delta_{mn}a_n \tag{2.17}$$

such that

$$\widehat{A}\widehat{U} = \widehat{U}d_A. \tag{2.18}$$

In index notation

$$\sum \widehat{A}_{ij}\widehat{U}_{jn} = \widehat{U}_{jn}a_n \tag{2.19}$$

The columns $\vec{U}^{(n)} : (\vec{U}^{(n)})_j = \widehat{U}_{jn}$ are the eigenvectors of $\widehat{A}$ to the eigenvalue a_n.

It follows, that $\widehat{A}$ can be represented using its eigenvalues and eigenvectors

$$\widehat{A} = \widehat{U}d_A\widehat{U}^\dagger = \sum_{n=0}^{N-1} \vec{U}^{(n)}a_n(\vec{U}^{(n)})^\dagger = \sum_{n=0}^{N-1} |n\rangle a_n\langle n| \tag{2.20}$$

The last form is the common quantum-mechanical form of writing the spectral representation.

Problem 1.3: Show the anti-hermiticity of the derivative operator

$$\int dx\chi^*(x)\partial_x\psi(x). \tag{2.21}$$

What are the conditions for this to hold?

Problem 1.4: Give the argument why a hermitian matrix has strictly real eigenvalues.

Problem 1.5: A hermitian matrix is called "positive" if its lowest eigenvalue is > 0. We define the $N \times N$ "overlap matrix" $\widehat{S}$ for a set of square-integrable functions $B = \{\phi_0(x), \phi_1(x), \ldots \phi_{N-1}(x)\}$ as

$$\widehat{S}_{ij} = \int dx\phi_i^*(x)\phi_j(x) \tag{2.22}$$

Prove that the overlap matrix is positive, when B is linearly independent. Show that $\widehat{S}$ is singular, when B is linearly dependent.

```
#! /usr/bin/env python

# an elementary finite difference solver

import numpy as np          # standard module for array handling
import scipy.linalg as la   # standard module for linear algebra

# (1) choose number of grid points, volume...
ng=330    # number of grid points
L=10.     # size of simulation box

# (2) select the system specifications
def potential(x): return x*x/2. # potential

# (3) set up derived quantities
h=L/float(ng)  # grid spacing
qh=0.5/(h*h)

# (4) set up operators
kin=np.zeros((ng,ng)) # kinetic energy matrix
for i in range(ng):
    kin[i,i]=2.*qh
    if i>0:
        kin[i,i-1]=-qh
        kin[i-1,i]=-qh
    if i<ng-1:
        kin[i+1,i]=-qh
        kin[i,i+1]=-qh
pot=np.zeros((ng,ng)) # potential energy matrix
for i in range(ng): pot[i,i]=potential(-L/2+float(i)*h)

# (5) solve eigenproblem
(val,vec)=la.eig(kin+pot)

# (6) show results
print np.sort(val.real)[:10]
```

Problem 1.6: Download and install Python. Write a program that prints "hello world!". Download the "numpy" and "scipy" packages. Write a finite difference code for the harmonic oscillator (copy from lecture). Plot the errors of the lowest three eigenvalues as a function of number of grid points at fixed box-size L and as a function of box size L. Make sure to disentangle the two errors.

2.1.3 Higher order finite difference schemes

The explicitly symmetric forms of the FD 2nd derivative (accurate to order $2n$) can be given by writing the coefficients to starting from the diagonal to the right $\widehat{D}^{(2)}_{ij}$, $i \leq j$ and using $(\widehat{D}^{(2)})^T = \widehat{D}^{(2)}$. Here the schemes up to order 8:

```
'd2_o2': (-2.,1.),
'd2_o4': (-5./2.,4./3.,-1./12.),
'd2_o6': (-49./18.,3./2.,-3./20.,1./90.),
'd2_o8': (-205./72.,8./5.,-1./5.,8./315.,-1./560.)
```

Problem 1.7: Construct general finite-difference schemes for the first derivative of arbitrary order $p-1$. By "order $= p-1$" one means that the scheme returns the *exact* first derivative for polynomials of degree up to $p-1$, i.e. $\mathcal{E}(h) = \mathcal{O}(h^p)$.
Hint: It is sufficient to construct a scheme $d_0, d_1, \ldots, d_{p-1}$ that is exact for the monomials $x^i, i = 0, 1, \ldots, p-1$.

$$\sum_{j=0}^{p-1} x_j^i d_j = i x_{j_0}^{i-1}, i = 0, 1, \ldots, p-1, j_0 \in 0, 1, \ldots, p-1 \tag{2.23}$$

which is a system of linear equations of the form

$$\widehat{X}\vec{d} = \vec{r}, \quad (\vec{d})_j = d_j, \quad (\vec{r})_i = i x_{j_0}^{i-1}. \tag{2.24}$$

You will obtain different schemes, depending on the choice of j_0, e.g. in the center or nearer to the edges of the range of x_j.

Use python linear solvers to solve this, try to convert the resulting floating numbers d_i to fractions. Verify the numbers for $p-1 = 2, 4$ analytically.

2.2 Basis set representations

2.2.1 Taylor series expansion

Assume the solution is analytic

$$\Psi(x) = \sum_j a_j x^j \text{ on interval } [-L/2, L/2] \tag{2.25}$$

Insert into Schrödinger equation and solve for a_i and eigenvalues.

$$\frac{1}{2}\sum_{j=0}^{N-1}(-j(j-1)x^{j-2}+x^{j+2})a_j = \sum_{j=0}^{N-1} x^j a_j \tilde{E} = 0 \tag{2.26}$$

Multiply from the left by x^i and integrate over $[-L/2, L/2]$:

$$\sum_j [\widehat{T}_{ij}+\widehat{V}_{ij}]a_j = \sum_j \widehat{S}_{ij} a_j \tilde{E} \tag{2.27}$$

with

$$\widehat{S}_{ij} := \int_{-L/2}^{L/2} dx x^{i+j} \tag{2.28}$$

$$\widehat{T}_{ij} := -\frac{1}{2}j(j-1)\int_{-L/2}^{L/2} dx x^{i+j-2} \tag{2.29}$$

$$\widehat{V}_{ij} := \frac{1}{2}\int_{-L/2}^{L/2} dx x^{i+j+2} \tag{2.30}$$

$$\tag{2.31}$$

Note: Kinetic energy term is badly asymmetric: We will obtain *complex* eigenvalues. Replace by a symmetric form (partial integration, ignore surface terms)

$$\widehat{T}_{ij} := \frac{1}{2}ji\int_{-L/2}^{L/2} dx x^{i+j-2} \tag{2.32}$$

Solve finite (generalized) matrix eigenvalue problem

$$(\widehat{T}+\widehat{V})\vec{a} = \widehat{S}\vec{a}\tilde{E} \tag{2.33}$$

How? — Use standard software. The problem can be immediately transformed to a standard eigenvalue problem:

$$\widehat{S}^{-1}(\widehat{T}+\widehat{V})\vec{a} = \vec{a}\tilde{E} \tag{2.34}$$

Note: $\widehat{S}$ is invertible as the monomials x^j are linearly independent However, the matrix $\widehat{S}^{-1}(\widehat{T}+\widehat{V})$ is not symmetric, which has important numerical drawbacks. A better way is to construct an explicitly symmetric form as follows: write

$$\widehat{S} = \widehat{R}^T\widehat{R} \tag{2.35}$$

with real, invertible matrices $\widehat{R}$ and re-write the eigenvalue problem as

$$\widehat{R}^{-1T}(\widehat{T}+\widehat{V})\widehat{R}^{-1}\vec{b} = \vec{b}\tilde{E}, \quad \vec{b} = \widehat{R}\vec{a}. \tag{2.36}$$

The lhs. matrix is explicitly symmetric (if $\widehat{T}+\widehat{V}$ is symmetric). This path is standard, where the so-called Cholesky-decomposition is used for constructing triangular $\widehat{R}$. The exact working of this is not of concern for us here. A different (although not as numerically efficient) way of decomposing $\widehat{S}$ shown in the following problem:

Problem 2.8: Use the spectral representation of the overlap matrix $\widehat{S}$ and the fact that it is positive (i.e. all eigenvalues are > 0) to construct $\widehat{R} : \widehat{S} = \widehat{R}^T\widehat{R}$.

With $\vec{b} := \widehat{R}\vec{a}$:

$$(\widehat{R}^T)^{-1}(\widehat{T}+\widehat{V})\widehat{R}^{-1}\vec{b}^{(n)} = \vec{b}^{(n)}\tilde{E}^{(n)} \tag{2.37}$$

Note: All eigenvalues $\tilde{E}^{(n)}$ will be real as $(\widehat{R}^T)^{-1}(\widehat{T}+\widehat{V})\widehat{R}$ is hermitian

2.2.2 Expansion into general polynomials

The Taylor series ansatz is the most straight forward but numerically very instable method of parameterizing a continuous wave function by a finite set of discrete numbers.

For very short expansions ($N \lesssim 10$) , the Taylor series does give improvements compared to the FD scheme with the same number of points, but at larger N it quickly breaks down.

We have not discussed boundary conditions for the ansatz (we will see soon that this needs more attention), but this is not the problem. The true problem is that mathematically the the overlap matrix $\widehat{S}$ is invertible, however, numerically this inversion is extremely unstable.

Condition number

A formal derivation of the importance of the condition number can be found in Golub/vanLoan, chap. 2.7. Here some discussion to give you a feeling for what happens:

Errors in the representation of matrices and vectors:

$$(\widehat{A}+\epsilon\widehat{F})x(\epsilon) = \vec{b}+\epsilon\vec{f} \tag{2.38}$$

Approximate spectral representation (1st order perturbation theory of quantum mechanics)

$$\widehat{A}+\epsilon\widehat{F} \approx \widehat{U}_A d_\epsilon \widehat{U}_A^\dagger \tag{2.39}$$

For notational simplicity, assume $\widehat{U}_A$ is real.

$$(d_\epsilon)_{mn} = \delta_{mn}(a_n+\epsilon f_n), \quad f_n := \vec{U}^{(n)}\cdot F\vec{U}^{(n)}) \tag{2.40}$$

Solution of the linear system

$$x(\epsilon) = \sum_n \vec{U}^{(n)}\frac{1}{a_n+\epsilon f_n}(\vec{U}^{(n)}\cdot\vec{b}+\epsilon\vec{U}^{(n)}\cdot\vec{f}) \tag{2.41}$$

Clearly, if $a_n + \epsilon f_n \approx 0$, very large errors will result. In general, if the matrix $\widehat{F}$ is random, i.e. not related to $\widehat{A}$, the f_n will all be of comparable magnitude $f_n \approx f$. The error in the largest eigenvalue a_n is necessarily given by machine precision, typically

$$\frac{\epsilon f}{\max_n(a_n)} \gtrsim 10^{-14} \tag{2.42}$$

That means that the relative error for the smallest a_n will be

$$\frac{\epsilon f}{\min_n(|a_n|)} \gtrsim 10^{-14} \frac{\max_n(|a_n|)}{\min_n(|a_n|)} \tag{2.43}$$

Clearly, when the "condition number" $\kappa(A)$ is large

$$\kappa(A) := \frac{\max_n(|a_n|)}{\min_n(|a_n|)} \sim 10^{14}, \tag{2.44}$$

$1/(a_n + \epsilon f)$ can become singular and matrix inversion breaks down completely, at smaller ratios severe errors occur. Note that the matrix norm $||A|| := \max_n(|a_n|)$ we can also write

$$\kappa(A) := ||A||||A^{-1}|| \tag{2.45}$$

Problem 2.9: Numerically compute the condition number for the overlap matrix $\widehat{S}$ of the monomials $\{1, x, x^2, \dots x^{N-1}\}$ in the interval $[-1, 1]$. At which N is condition number $\sim 10^{14}$ reached?

This problem of "near linear dependency" or "ill-conditioning" of the overlap can be avoided by not expanding the solution in the monomials $1, x, x^2 \dots x^{N-1}$, but rather in some different set of polynomials $P_0(x), P_1(x), \dots, P_{N-1}(x)$ of maximal order $N-1$. Mathematically, these polynomials span the same space of solutions.

All polynomials are alike

Any set of N linearly independent polynomials $p_n(x)$ of maximal degree $N-1$ is related to any other set of linearly independent polynomials $q_n(x)$ of maximal degree $N-1$ by an invertible (non-singular) linear $N \times N$ transformation $\widehat{M} : q_n(x) = \sum_{m=0}^{N-1} p_m(x)\widehat{M}_{mn}$ and $p_m(x) = \sum_{n=0}^{N-1} q_n(x)(\widehat{M}^{-1})_{nm}$. We can also write this in the form of a row-vector with a matrix:

$$\vec{q}(x) = \vec{p}(x)\widehat{M}, \quad \vec{p}(x) = \vec{q}(x)\widehat{M}^{-1} \tag{2.46}$$

$$\vec{q}(x) := \begin{pmatrix} q_0(x) \\ q_1(x) \\ q_2(x) \\ \vdots \\ q_{N-2}(x) \\ q_{N-1}(x) \end{pmatrix}^T \qquad \vec{p}(x) := \begin{pmatrix} p_0(x) \\ p_1(x) \\ p_2(x) \\ \vdots \\ p_{N-2}(x) \\ p_{N-1}(x) \end{pmatrix}^T \tag{2.47}$$

Let $\vec{a}$ be the solution to the eigenproblem using a expansion in $p_m(x)$ as:

$$(\widehat{T}^{(p)} + \widehat{V}^{(p)})\vec{a} = \widehat{S}^{(p)}\vec{a}\tilde{E}, \tag{2.48}$$

where $(\widehat{S}^{(p)})_{mn} = \langle p_m | p_n \rangle$ and $(\widehat{S}^{(q)})_{mn} = \langle q_m | q_n \rangle$ with the wave function:

$$\Psi(x) = \sum_n p_n(x) a_n := \vec{p}(x) \cdot \vec{a}. \tag{2.49}$$

Then the eigenproblem in expansion $q_n(x)$ has the matrices:

$$\widehat{T}^{(q)} = \widehat{M}^T \widehat{T}^{(p)} \widehat{M}, \quad \widehat{V}^{(q)} = \widehat{M}^T \widehat{V}^{(p)} \widehat{M}, \quad \widehat{S}^{(q)} = \widehat{M}^T \widehat{S}^{(p)} \widehat{M}. \tag{2.50}$$

and takes the form

$$(\widehat{T}^{(q)} + \widehat{V}^{(q)})\vec{b} = \widehat{S}^{(q)}\vec{b}\tilde{E}. \tag{2.51}$$

One immediately sees that:

$$\vec{a} = \widehat{M}\vec{b}, \qquad \vec{b} = \widehat{M}^{-1}\vec{a} \tag{2.52}$$

and the resulting approximation to the exact eigensolution should be identical in both expansions:

$$\Psi^{(q)} = \sum_m q_m(x) b_m = \vec{q}(x) \cdot \vec{b} = (\vec{p}\widehat{M}) \cdot (\widehat{M}^{-1}\vec{a}) = \vec{p} \cdot \widehat{M}\widehat{M}^{-1}\vec{a} = \vec{p}(x) \cdot \vec{a}. \tag{2.53}$$

Any difference when using different polynomials in expansions lies only in numerical stability and computational convenience. Mathematically all such expansions are exactly equivalent.

2.2.3 Orthogonal polynomials, special case: Legendre polynomials

(see Gradshteyn/Ryzhik: Tables of Integrals, Series, and Products)

We will choose polynomials such that the overlap matrix is easily invertible. In the ideal case the overlap matrix is diagonal. A prominent example of such orthogonal polynomials are the Legendre polynomials with the following recurrence formulas:

$$P_0(x) = 1 \tag{2.54}$$

$$P_1(x) = x \tag{2.55}$$

$$P_{n+1}(x) = \frac{1}{n+1}\left[(2n+1)xP_n(x) - nP_{n-1}(x)\right]. \tag{2.56}$$

$P_{n+1}(x)$ can be evaluated efficiently with $P_0(x)$ and $P_1(x)$.

The Legendre polynomials obey the orthogonal relation:

$$\int_{-1}^{1} dx P_n(x) P_m(x) = \delta_{mn} \frac{2}{2n+1}. \tag{2.57}$$

We can readily also obtain a recurrence formula for the *derivatives* of these polynomials:

$$\partial_x P_0 = 0 \tag{2.58}$$

$$\partial_x P_1 = 1 \tag{2.59}$$

$$\partial_x P_{n+1}(x) = \frac{1}{n+1}\left[(2n+1)(x\partial_x P_n(x) + P_n(x)) - n\partial_x P_{n-1}(x)\right] \tag{2.60}$$

With Legendre polynomials one has not much more effort than with monomials and the advantage is that one gets no errors when inverting matrices.

Example:

$$\widehat{S} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{5} \end{pmatrix}, \widehat{S}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 5 \end{pmatrix} \Rightarrow \text{no errors!} \tag{2.61}$$

Problem 2.10: The Chebyshev polynomials $T_n(x)$ of degrees $n = 0, 1, \ldots, N-1$ are defined by the orthogonality relations

$$\int_{-1}^{1} dx \frac{T_m(x)T_m(x)}{\sqrt{1-x^2}} = \begin{cases} 0 & : m \neq n \\ \pi & : m = n = 0 \\ \pi/2 & : else \end{cases} \tag{2.62}$$

Write a code for evaluating $T_m(x)$ (find the 2-point recurrence relations in the literature or — probably much more quickly — on the web). Verify the correctness of the code by computing the integrals.
Note: For crude integration, you can use $\int_a^b f(x)dx \approx h\sum_{n=0}^{N-1} f(a+nh)$, $h = (b-a)/N$. Much more accurate and efficient is Gauss quadrature. Gauss-Legendre will give reasonable results, but far from exact (see problem below). Try using "Gauss-Chebyshev" quadrature instead.

Gauss-Legendre quadrature

We need to compute the integrals:

$$\widehat{T}_{mn} = \frac{1}{2}\langle \partial_x Q_m | \partial_x Q_n \rangle = \frac{1}{2}\int_{-\frac{L}{2}}^{\frac{L}{2}} dx Q'_m(x) Q'_n(x) \tag{2.63}$$

$$\widehat{V}_{mn} = \frac{1}{2}\langle Q_m | x^2 | Q_n \rangle = \frac{1}{2}\int_{-\frac{L}{2}}^{\frac{L}{2}} dx Q_m(x) x^2 Q_n(x) \tag{2.64}$$

The Legendre polynomials are orthogonal on the interval $[-1, 1]$. But we are interested in the interval $[-L/2, L/2]$. That is why we scale to the interval $[-L/2, L/2]$:

$$Q_n(x) =: P_n(\frac{2x}{L}), \qquad x \in [-\frac{L}{2}, \frac{L}{2}] \tag{2.65}$$

$$\Rightarrow \int_{-\frac{L}{2}}^{\frac{L}{2}} dx Q_m(x) Q_n(x) = \int_{-\frac{L}{2}}^{\frac{L}{2}} dx P_m(\frac{2x}{L}) P_n(\frac{2x}{L}) = \int_{-1}^{1} dy \frac{L}{2} P_m(y) P_n(y) = \frac{L}{2}\delta_{mn}\frac{2}{2n+1}. \tag{2.66}$$

Although $\widehat{T}_{mn}, \widehat{V}_{mn}$ and similar integrals can be evaluated in closed form, it is simpler, usually faster, usually more stable, and more versatile to use Gauss-Legendre quadrature for the evaluation.

The following is exact for any polynomial R_{2n-1} of degree $2n-1$

$$\int_{-1}^{1} dx R_{2n-1}(x) = \sum_{i=0}^{N-1} R_{2n-1}(x_i) w_i, \qquad \text{for any } n \leq N, \tag{2.67}$$

where x_i and w_i are nodes and weight of the n-point Gauss-Legendre quadrature formula respectively. The weights w_i can be calculated numerically. For an adequate method see Module M4: Numerical Methods, SS2009, M. Kerscher. The Gauss-Legendre quadrature is more efficient because we need to evaluate only half of the points. For one-dimensional integration, that seems a minor difference. However in, say, 3 dimensions, the reduction is $2^3 = 8$, i.e. almost a factor of 10!

Problem 2.11: Use Gauss-Legendre quadrature to compute the integral

$$\int_0^1 e^{-x} dx. \tag{2.68}$$

For this particular example, which kind of convergence do you expect? Plot the error of the numerical integral vs. the number of quadrature points to verify your expectations. Now try to compute the integral

$$\int_0^1 \sqrt{x} dx. \tag{2.69}$$

How does the error change with the number of quadrature points? Point out the reason for the two different convergence behaviors in the two cases.

Hint: As Gauss-Legendre quadrature is for integration over the interval [-1,1], map into that interval by a change of variables. However, an N-point quadrature is *exact* only up to polynomial degree $2N-1$. Compare the error of the numerical approximation with the error when truncating the Taylor series at degrees $2N-1$.

Note: G auss-Legendre quadrature points x_i and weights w_i, $i = 0, 1, \ldots, N-1$ are returned by the function "p_roots(N)" from the scipy-module "scipy.special.orthogonal".

2.2.4 General properties of orthogonal polynomials

Metric, weight function $v(x)$

Two polynomials Φ and χ are orthogonal with respect to some Hilbert space (over an interval $[a, b]$):

$$\langle \Phi | \chi \rangle = \int_a^b dx\, \Phi(x) \chi(x) \cdot v(x), \tag{2.70}$$

where $v(x)$ is the weight function. Because of the positivity of the norm ($||\Phi|| > 0 \forall \Phi \neq 0$) $v(x) > 0$ almost everywhere.
Example: Polar coordinates:

$$\int_0^\infty dr\, r^2 \Phi(r) \chi(r), \tag{2.71}$$

where $a = 0$, $b = \infty$ and r^2 can be associated with the weight function $v(x)$.

So far we had the case: $v(x) = 1$, $a = -1$, $b = 1$. Now we can construct a set of orthogonal polynomials for any $v(x)$:

$$\langle Q_m | Q_n \rangle = \int_a^b dx\, v(x) Q_m(x) Q_n(x) = N_n \delta_{mn} \tag{2.72}$$

provided

$$\int_a^b dx\, x^n v(x) < \infty \quad \forall n \tag{2.73}$$

Explicit construction (Schmidt orthonormalization):

$$\begin{aligned}
Q_0^{(v)}(x) &:= 1 && (2.74)\\
\tilde{Q}_1^{(v)}(x) &:= a_1 x + b_1 && (2.75)\\
|Q_1^{(v)}\rangle &= |\tilde{Q}_1^{(v)}\rangle - |Q_0^{(v)}\rangle\langle Q_0^{(v)}|\tilde{Q}_1^{(v)}\rangle = a_1 x + b_1 - \int_a^b dx\, (a_1 x + b_1) v(x) && (2.76)\\
|Q_{n+1}^{(v)}\rangle &= x|Q_n^{(v)}\rangle - \sum_{m=0}^{n} \frac{|Q_m^{(v)}\rangle\langle Q_m^{(v)}|x|Q_n^{(v)}\rangle}{\langle Q_m^{(v)}|Q_m^{(v)}\rangle} := x|Q_n^{(v)}\rangle - \mathbf{P}_n^{(v)} x |Q_n^{(v)}\rangle, && (2.77)
\end{aligned}$$

where $\mathbf{P}_n^{(v)}$ is the projection operator.

Recurrence relations

For any set of orthogonal polynomials there exists a (unique) three terms recurrence relation:

$$Q_n(x) = (a_n x + b_n) Q_{n-1}(x) + c_n Q_{n-2}(x) \quad a_n,\ b_n \text{ and } c_n \text{ depend on } n \text{ and } v(x) \tag{2.78}$$

Orthogonal polynomials and Gauss quadrature

For any set of ON polynomials on interval $[a, b]$ with weight function $v(x)$ we have a quadrature rule $\{x_i^{(v,n)}, w_i^{(v,n)}, i = 0, 1, \ldots, n-1\}$ such that the integral for any polynomial $P_m(x)$ with degree $m \leq 2n - 1$ is *exact*:

$$\int_a^b dx\, v(x) P_m(x) = \sum_{i=0}^{n-1} P(x_i^{(v,n)}) w_i^{(v,n)} \quad \forall m \leq 2n - 1 \tag{2.79}$$

The base points are the zeros of the orthogonal polynomial of degree n. Proof and construction of weights in "LMU — Numerik für Physiker".

Note: Quadrature rules for singular functions, if character and position of singularity is known.
Note: Quadratures for a weight function $v(x)$ converge rapidly for $\int v(x) f(x)$, if the higher derivatives of $f(x)$ are well bounded, e.g. $f^{(n)}(x) < C$, C independent of n

Note: It is important to stress the fact that only n points are needed to compute the integral of a polynomial whose order can be as high as $2n - 1$.

2.2.5 Object oriented programming

We will consider Legendre, Laguerre, and other polynomials as “objects”. To define the object we will make use of the concept of class. Quoting Wikipedia:*Object-oriented programming (OOP) is a programming paradigm using ”objects” — data structures consisting of data fields and methods together with their interactions — to design applications and computer programs. Programming techniques may include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance. Many modern programming languages now support OOP.*

Some of the most important concepts about Object oriented programming are listed in what follows:

- **modularity:** our “foo.py” files should contain well-defined units of functionality with well-defined and limited input and output. In each “module” only few functions and data should be accessible from outside the module (by calls to methods and classes of the module).

  Note: “foo” is often used as a placeholder for “some name”. Its etymology is confusing.

- **inheritance:** e.g. LegendrePolynomial inherits the methods “test()” and “val()” from OrthogonolPolynomial, i.e. any OrthogonalPolynomial has those methods; similarly OrthogonalPolynomial inherits “plot()” from MyFunction, i.e. any MyFunction can be plotted.

- **encapsulation:** is closely related to modularity and the “class”: keep tight control of what goes in and out of a module

- “Polymorphism” and “messaging” are not manifestly at work in these first examples.

Computational Style

Here there are some useful tips needed for programming in a professional way:

- Document! Write the documentation first, then the code!
- Introduce test for (almost) everything;
- Do type conversion explicitly where it can be critical;
- Keep input in a single place in the code
- Never duplicate code or other information — write modules or subroutines instead, use templates
- Never duplicate code, except possibly for speed.
- Never ever put any constants into the body of the code: physical constants like $\hbar$, speed of light, units conversions or numerical control parameters like threshold “epsilons”, maximum number of iterations, maximally allowed problem sizes, etc. physical constants can be put into a single module, algorithm parameters can be in the header of the module containing the algorithm.

- In case constructs, never leave default cases undefined.
- Never assume that inputs can only be certain form or range of values — always check!
- Never use smart tricks (unless they are absolutely needed for speed or similar reasons): they are difficult to understand later, they may be compiler dependent. If you do, DOCUMENT in great detail.
- Strictly adhere to standards.

2.2.6 Objects in Python: class for orthogonal polynomials

orthopol.py

```
#! /usr/bin/env python
import numpy as np
import matplotlib.pyplot as plt
import scipy.special.orthogonal as so
import myfunction as mf

class OrthogonalPolynomial(mf.MyFunction):
    """
    values and derivatives of orthogonal polynomials
    mf.MyFunction has method "plot()"
    """
    def val(self,q,n=0):
        """values and derivatives up to order n
        """
        # values and derivatives
        #  p(x,n) = [a(n)+b(n)*x] *  p(x,n-1)                 + c(n)* p(x,n-2)
        # p'(x,n) = [a(n)+b(n)*x] * p'(x,n-1) + b(n)*p(x,n-1) + c(n)*p'(x,n-2)
        v=np.zeros((n+1))
        d=np.zeros((n+1))
        v[0]=1.
        d[0]=0.
        if n>0:
            for i in range(1,n+1):
                v[i]=(self.a(i)+self.b(i)*q)*v[i-1]                +self.c(i)*v[i-2]
                d[i]=(self.a(i)+self.b(i)*q)*d[i-1]+self.b(i)*v[i-1]+self.c(i)*d[i-2]
        return v,d

    def test(self,n):
        """check orthogonality up to order n
        """
        # get the quadrature rule
        x,w=self.quadrature(n+1)

        # print it - should be diagonal
        np.set_printoptions(precision=5,suppress=True)

        # compute the overlap matrix
        o=np.zeros((n+1,n+1))
        for j in range(len(x)):
            v=self.val(x[j],n)[0]
            for i in range(n+1):
                o[i,:]=o[i,:]+v[i]*v[:]*w[j]
        print o

        # plot the functions
        self.plot(-1.,1.,n+1)
        plt.show()

class LegendrePolynomial(OrthogonalPolynomial):
    """recurrence coefficients and quadrature
    inherits from OrthogonalPolynomial:
    val()... return values and derivatives
    plot().. plot on reasonable interval
    """
    def a(self,i): return 0.
    def b(self,i): return (2.*float(i)-1.)/float(i)
    def c(self,i): return (-float(i)+1.)/float(i)
    def quadrature(self,n): return so.p_roots(n)

class LaguerrePolynomial(OrthogonalPolynomial):
    """recurrence coefficients and quadrature
    """
    def a(self,i): return (2*float(i)-1.)/float(i)
    def b(self,i): return -1./float(i)
    def c(self,i): return (-float(i)+1.)/float(i)
    def quadrature(self,n): return so.la_roots(n,0)

class ChebyshevPolynomial(OrthogonalPolynomial):
    """recurrence coefficients and quadrature
    """
    def a(self,i): return 0.
    def b(self,i):
        if i==1: return 1. # Chebyshevs are normalized inconsistently
        else: return 2.
    def c(self,i): return -1.
    def quadrature(self,n): return so.t_roots(n)
```

myfunction.py

"MyFunction" is a very simple class for now, but we will use it to set up other basis functions than orthogonal polynomials:

```
import numpy as np
import matplotlib.pyplot as plt
class MyFunction:
    # draw the functions on the interval
    def plot(self,x0=None,x1=None,n=1,pts=100):
        """Plot all functions on element"""
        if x0 is None: x0=self.x0
        if x1 is None: x1=self.x1
        if hasattr(self,'n'):
            if self.n is not None: n=self.n

        x=np.linspace(x0,x1,pts)
        f=np.zeros((len(x),n))
        for i in range(len(x)): f[i,:]=self.val(x[i],n-1)[0]
        for i in range(n): plt.plot(x,f[:,i])
```

Problem 2.12: (Teamwork) Using `orthopol.py` as a model and also importing it, create a module with the base class `BasisFunction` and the derived classes `LaguerreExpon` and `LegendreScaled` with the following properties and structure:

- Document what you plan to do in the module using """ ...""" and # comments
- `BasisFunction` refers to a fixed set of n basis functions $\Phi_0, \Phi_1, \dots, \Phi_{n-1}$.
- `BasisFunction` inherits all properties from `MyFunction` (download from web page or get it from Git repository).
- it has a method, `overlap()`, which returns the overlap matrix $\widehat{S} : \widehat{S}_{ij} = \int dx \Phi_i(x)\Phi_j(x)$ computed by a quadrature rule.
- `LaguerreExpon` specifies BasisFunction to be of the form $L_n(2x/k)\exp(-x/k)$
- it has a method `quadrature` that returns the suitable quadrature rule for these functions
- it has a method `val` that evaluates values and derivatives of the functions.
- `LegendreScaled` performs the same functions for the scaled Legendre polynomials $P_n(2\frac{x-x0}{x1-x0} - 1)$

2.3 Rayleigh-Ritz variational principle

Let $\mathbf{A}$ be a self-adjoint (i.e. hermitian) operator on a Hilbert space $\mathcal{H}$ with a domain $D(\mathbf{A})$ (i.e. functions on which $\mathbf{A}$ is defined) which is bounded from below (i.e. its lowest eigenvalue $a_0 > -\infty$). Let $a_0 \leq a_1 \leq \ldots \leq a_{N-1}$ denote the sorted sequence of the eigenvalues of $\mathbf{A}$. Let $\{\phi_0, \phi_1, \ldots, \phi_{N-1}\} \subset D(\mathbf{A})$ be any set of (linearly independent) functions from $D(\mathbf{A})$. Denote by $\widehat{A}$ and $\widehat{S}$ the operator and overlap *matrices*

$$(\widehat{A})_{ij} = \langle \phi_i | \mathbf{A} | \phi_j \rangle, \quad (\widehat{S})_{ij} = \langle \phi_i | \phi_j \rangle. \tag{2.80}$$

Then the sorted eigenvalues $\tilde{a}_0 \leq \tilde{a}_1 \leq \ldots \leq \tilde{a}_{N-1}$ of the matrix eigenvalue problem

$$\widehat{A}\vec{c}^{(n)} = \widehat{S}\vec{c}^{(n)}\tilde{a}_n \text{ for } n = 0, 1, \ldots, N-1. \tag{2.81}$$

is a sequence of *upper bounds* of the exact eigenvalues:

$$a_n \leq \tilde{a}_n \text{ for } n = 0, 1, \ldots, N-1 \tag{2.82}$$

Note that adding more functions will always lower the bound $\tilde{a}_n$, i.e.

$$a_n \leq \tilde{a}_n^{(\infty)} \leq \ldots \leq \tilde{a}_n^{(N+1)} \leq \tilde{a}_n^{(N)}. \tag{2.83}$$

The proof of this is not very hard and should have been part of the “Mathematische Quantenmechanik”. It is a direct consequence of the “Minimax principle” for the eigenvalues of a semi-bounded operator. For operators on infinite dimensional spaces it is more math than what I want to bring in these lectures. The finite-dimensional version is essentially the same and shows the idea.

Minimax principle

(see Golub/vanLoan, chapter 8)
Let S_k be a k-dimensional subspace of S_N. The minimax principle says that

$$a_{k-1} = \min_{\dim(S)=k} \max_{\phi \in S} \frac{\langle \phi | \widehat{A} | \phi \rangle}{\langle \phi | \phi \rangle}, \quad S \subset \mathcal{H}. \tag{2.84}$$

where everything is defined as above.
Proof:(For finite dimensional Hilbert-space $\mathcal{H}$: dim($\mathcal{H}$)= N)

- Pick $S_0 = \text{span}\,\{\phi_0, \phi_1, \ldots, \phi_{k-1}\}$ where ϕ_i is the eigenfunction belonging to the i-th eigenvalue. Then we have $\max_{\phi \in S_0} \frac{\langle \phi | \widehat{A} | \phi \rangle}{\langle \phi | \phi \rangle} = a_{k-1}$, from which it follows that $a_{k-1} \geq \min_{\dim(S)=k} \max_{\phi \in S} \frac{\langle \phi | \widehat{A} | \phi \rangle}{\langle \phi | \phi \rangle}$.
- For the opposite inequality we consider T = span $\{\phi_{k-1}, \phi_k, \ldots, \phi_{N-1}\}$ and we pick an arbitrary $S \subset D(\widehat{A})$ such that dim(S) = k. As T is $N-k+1$-dimensional, it must intersect with S, which means that there exists $\phi_* = \sum_{l=k-1}^{N-1} c_l \phi_l \in S \cap T$ such that:

$$\max_{\phi \in S} \frac{\langle \phi | \widehat{A} | \phi \rangle}{\langle \phi | \phi \rangle} \geq \frac{\langle \phi_* | \widehat{A} | \phi_* \rangle}{\langle \phi_* | \phi_* \rangle} \geq a_{k-1} \tag{2.85}$$

The last inequality follows from the fact that $\widehat{A}\phi_* = \sum_{l=k-1}^{N-1} a_l c_l \phi_l$ and therefore

$$\langle \phi_* | \widehat{A} | \phi_* \rangle = \sum_{l=k-1}^{N-1} \sum_{m=k-1}^{N-1} a_l c_l c_m^* \langle \phi_l | \phi_n \rangle = \sum_{l=k-1}^{N-1} a_l |c_l|^2, \tag{2.86}$$

assuming without loss of generality orthonormal eigenvectors ϕ_l. We see that $\langle \phi_* | \widehat{A} | \phi_* \rangle / \langle \phi_* | \phi_* \rangle$ is an average over all eigenvalues $a_{k-1} \leq a_k \leq \ldots$, which cannot be lower than its lowest value.

- Since the inequality

$$a_{k-1} \leq \max_{\phi \in S} \frac{\langle \phi | \widehat{A} | \phi \rangle}{\langle \phi | \phi \rangle} \tag{2.87}$$

is valid for any S with $\dim S = k$, we will also have

$$a_{k-1} \leq \min_{\dim(S)=k} \max_{\phi \in S} \frac{\langle \phi | \widehat{A} | \phi \rangle}{\langle \phi | \phi \rangle} \tag{2.88}$$

which completes the proof.

2.4 Boundary conditions

The Laplace operator is not well-defined on the functions we used so far: on the boundary, the functions discontinuously (and non-differentiably) drop to 0. One consequence is that one is easily lead to non-hermitian matrices. We need to define the set of expansion functions (or "basis functions"), where the second derivative matrix can be properly defined.

Domain of an operator and a quadratic form

One of the key concepts for differential operators is the "domain", i.e. a proper selection of functions on which the differential operator can be defined. While this may appear a minor technical issue that one is inclined to solve on the fly, it is in fact the core of any (linear) partial differential equation. It is intimately related to questions like conservation of probability (in quantum mechanics) or energy (in Maxwell's equations). In particular when truncating space for computations, doing coordinate transformations, using analyticity arguments, or using wild beasts like δ-"potentials" one needs to check how these manipulations affect the definition of the domain of the operator. Here we essentially by-pass these questions except for the most immediate issues.

A minimal requirement for the definition of an operator $\mathbf{A}$ on Hilbert space $\mathcal{H}$ is that the result of applying it to a function from its domain must be in $\mathcal{H}$:

$$\Psi \in D(\mathbf{A}) : \mathbf{A}\Psi \in \mathcal{H} : ||\mathbf{A}\Psi||^2 = \langle \mathbf{A}\Psi | \mathbf{A}\Psi \rangle < \infty. \tag{2.89}$$

If $\mathbf{A}\Psi$ formally contains a δ-function, $\Psi \notin D(\mathbf{A})$.

Luckily, the condition about the "domain" can be relaxed somewhat: it is sufficient, that

$$\langle \Psi | \mathbf{A}\Psi \rangle < \infty, \tag{2.90}$$

which usually can be defined even if $\mathbf{A}\Psi$ contains a δ-function. More generally, we can operate with a set of functions ϕ_m such that $\langle\phi_n|\mathbf{A}|\phi_m\rangle$ and $\langle\phi_n|\phi_m\rangle$ exist $\forall m, n$. This space is larger than $D(\mathbf{A})$, it is the "closure" of $D(\mathbf{A})$ with the norm $||\Psi||_1 = \langle\Psi|\widehat{A}|\Psi\rangle + \langle\Psi|\Psi\rangle$. It is called the "domain of the quadratic form" associated with $\mathbf{A}$. For the Laplacian, it is the "first Sobolev space".

In the numerical examples above we have fallen below the exact harmonic oscillator ground state of 0.5, not because of numerical error, but because even this relaxed condition is violated. The scalar product of the Hilbert space of the 1d-harmonic oscillator extends over the whole real axis:

$$\langle\Phi_i|\partial_x^2|\Phi_j\rangle = \int_{-\infty}^{\infty} dx \Phi_i^*(x)\partial_x^2\Phi_j(x) \tag{2.91}$$

Restricting the integration to finite intervals $[-L/2, L/2]$ amounts to setting the $\Phi_i \equiv 0$ outside, creating a discontinuity at the boundaries. Then ∂_x^2 ceases to give any well-defined result.

Problem 4.13: Let $\mathbf{H}$ be a semi-bounded self-adjoint operator with eigenvectors $E_i \leq E_j$ for $i < j$ in the Hilbert space $\mathcal{H}$. Let $S_M = \{\phi_0, \phi_1, \dots, \phi_{M-1}\}$ be a set of M linearly independent vectors from $\mathcal{H}$. Let $\widehat{H}_{ij} = \langle\phi_i|\mathbf{H}|\phi_j\rangle$ and $\widehat{S}_{ij} = \langle\phi_i|\phi_j\rangle$ be the matrices in the generalized eigenproblem

$$\widehat{H}\vec{q}_i = \widehat{S}\vec{q}_i\tilde{E}_i. \tag{2.92}$$

Show that the upper bounding property $E_i < \tilde{E}_i^{(M)}, i = 0, 1, \dots, M-1$ follows from the minimax principle

$$E_{k-1} = \min_{\dim(S)=k} \max_{\phi\in S} \frac{\langle\phi|\mathbf{H}|\phi\rangle}{\langle\phi|\phi\rangle}, \quad S \subset \mathcal{H}. \tag{2.93}$$

Show that, as a direct consequence, the approximate energies always decrease when one adds more functions: Let $\tilde{E}_n^{(N)}$, $n \leq N$ be the nth approximate eigenvalue obtained with $S_M \supset S_N = \{\Phi_0, \Phi_1, \dots, \Phi_{N-1}\}$ and $\tilde{E}_n^{(M)}$, $M > N$ the nth eigenvalue obtained with an extended set $S_N \subset S_M = \{\Phi_0, \Phi_1, \dots, \Phi_{M-1}\}$. Then

$$\tilde{E}_n^{(M)} \leq \tilde{E}_n^{(N)} \tag{2.94}$$

Dirichlet boundary conditions

Note that if we require the functions Φ_i to satisfy the so-called Dirichlet boundary conditions, i.e. $\Phi_i(-L/2) = \Phi_i(L/2) = 0 \forall i$, the integral can still be defined: even when the first derivative is not differentiable, the second derivative just results in δ-like singularities at $x = -L/2$ and $x = L/2$. The function $\Phi_i^*(x)$ will be zero at those points and there is no contribution to the integral. With such a boundary condition, we are also guaranteed to obtain a hermitian kinetic energy matrix $\widehat{T}$, as can be easily seen by partial integration.

Example: Apply ∂_x^2 to functions with a discontinuous 1st derivative:
Let $f(x)$ and its first two derivatives be as in figure 2.1. We have that $f''(x) = -2\delta(x) + \delta(x-1) + \delta(x+1)$ makes sense in a distributional sense and can be integrated over. This example shows what was expressed above: a function which vanishes at the boundaries has a second derivative that can be integrated over, thanks to the special property of the δ function.

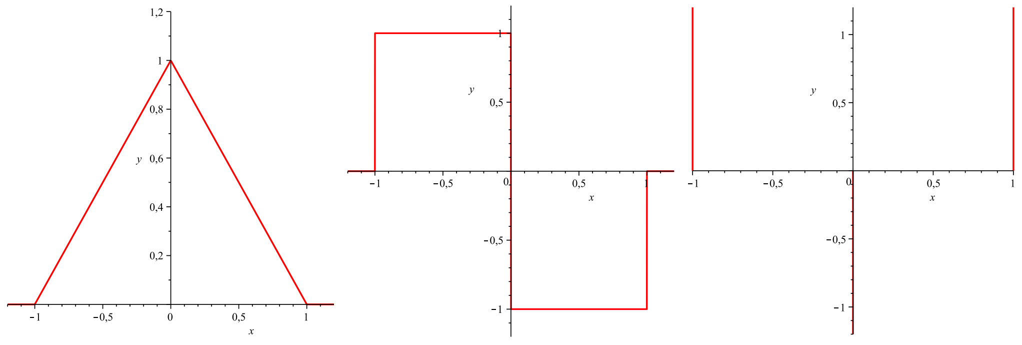
(a) $f(x)$ (b) $f^{'}(x)$ (c) $f^{''}(x)$

Figure 2.1: $f(x)$ and its first two derivatives

2.4.1 Implementation of Dirichlet boundary conditions

A suitable set of functions for defining matrix elements of ∂_x^2 with Dirichlet boundary conditions can be implemented starting from nearly any set of differentiable functions.

The polynomials that we have previously chosen, the (scaled) Legendre polynomials, are such that $P_{2n}(-1) = P_{2n}(1) = 1$ and $P_{2n+1}(-1) = P_{2n+1}(1) = 1$ with $n = 0, 1, \dots, N-1$, whereas we need a function that goes to zero at the boundaries, i.e. we need to choose polynomials $B_n(x)$ which are such that $B_n(-1) = B_n(1) = 0 \ \forall n$. We will choose a new set of polynomials:

$$B_n(x) = \begin{cases} P_{n+2}(x) - P_0(x) \text{ for } n \text{ even} \\ P_{n+2}(x) - P_1(x) \text{ for } n \text{ odd} \end{cases} \quad n = 0, 1, \dots, N-1 \tag{2.95}$$

(This is even simpler than it appears, remember: $P_0(x) \equiv 1, P_1(x) = x$.)
Note: The choice of the polynomials is not unique. Clearly any non-singular transformation $\vec{B}' = \vec{B} \cdot M$, with M non singular, has the same property.

Note: It is useful to know how to implement Dirichlet boundary conditions for Finite Difference schemes (FD). To this end we can consider, without loss of generality, the first order FD, and we introduce into this scheme two imaginary grid points ψ_{-1} and ψ_N and set their values to zero. We see that this is equivalent to the truncation of the 2nd order FD matrices that we used in the previous chapter.

Problem 4.14: (Teamwork) Given an (almost) arbitrary set of functions $\Phi_i(x)$ on the interval $[a, b]$. Give an algorithm to construct a linear combination of these functions

$$\vec{B}(x) = \vec{\Phi}(x)\widehat{M} \tag{2.96}$$

with $B_0(a) = 1$, $B_1(b) = 1$, and $B_m(a) = B_m(b) = 0$ everywhere else. What are the minimal requirements on the Φ_i's for such a construction to be possible?

Hint: The construction is not unique. One possibility is to first construct two functions B_0 and B_1

from any pair of functions Φ_{i_0}, Φ_{i_1}. This requires invertibility of the matrix

$$C = \begin{pmatrix} \Phi_{i_0}(a) & \Phi_{i_1}(a) \\ \Phi_{i_0}(b) & \Phi_{i_1}(b) \end{pmatrix} \tag{2.97}$$

All other functions can then be constructed as

$$B_m = \Phi_m + B_0 a_0 + B_1 a_1 \tag{2.98}$$

with suitable a_0, a_1.

Problem 4.15: (Teamwork) Implement the above algorithm in Python: write a class `FiniteElement` that is derived from class `BasisFunction` above. The constructor takes an arbitrary object of class `BasisFunction` and creates a new object of transformed functions.

2.5 Managing your codes: Git

The idea of code management systems like "Git" is that somewhere there exists a repository, where one can store and retrieve code. The system keeps track of every change that is applied to the code (who does this change, when, etc,...) and it is also possible to access all the previous versions of your code. Another feature that makes the system useful is that it gives you a warning in case that there is a conflict. A practical example of conflict is that two developers try to fix the same bug in the program at the same time: in this case they will receive a warning that will allow them to compare the two versions and then choose which version to finally "commit" to the repository.

At present, Git is possibly the most advanced of such code management systems. An older system that you may encounter is SVN ("subversion"), with similar, slightly less advanced functionality and command structure.

Problem 5.16: (Teamwork) Learn how to use the Git version control system:

1. Development teams: sign up for one of the "teams".

2. Together, decide for a name of the team.

3. Download and install git.

4. clone the computational physics repository

   ```
   prompt>git clone ssh://User.Name@git.physik.uni-muenchen.de:/pub/scm/CompPhys2014.git
   prompt>cd CompPhys2014
   ```

5. Create and checkout a new branch "team_name"

   ```
   prompt>git checkout -b team_name
   ```

6. Modify the file readme.txt to include your team's and members' names and commit the changes using

```
prompt>git add readme.txt
prompt>git commit readme.txt -m "Added team and member names to readme.txt"
```

7. Inspect the history of the branch by "git log".
8. Inspect the difference between the original and the newly committed changes ("git diff master").
9. Try to add your newly written module "finiteelement" (previous problem) *to your team branch* (not to the master branch).
10. Push your team's branch to the server using

```
prompt>git push origin team_name
```

Use git documentation (man git on linux) or online references to find out how to exactly do all this. Example reference: http://git-scm.com/book/en/

2.6 How to solve the eigenvalue problem

(Golub, van Loan)
Let us go back to the harmonic oscillator (H.O.). We learned two different ways to implement it:

- Finite difference (FD) schemes
- Basis set representations.

The FD schemes are simple, but not very accurate for fixed N, while the basis set representations can be very accurate. But do we get the "better" performance of the basis set representations for free? The answer is: not necessarily. In general it is "expensive" to solve the eigenvalue problem with the basis set representations, because $\widehat{T}$ and $\widehat{V}$ are full matrices. Yet, we can get very high accuracies. In turn, the FD scheme is "cheaper", but limited in the accuracy that can be reached. To understand the performance criteria for both approaches, we have to look at the fundamental problem of how the eigenvalue problem is solved for matrices.

The idea of the eigenvalue problem is to solve:

$$\det(\widehat{A} - \lambda \widehat{I}) = 0, \quad (\widehat{I})_{nm} := \delta_{nm}. \tag{2.99}$$

$\det(\widehat{A} - \lambda \widehat{I})$ is a polynomial in λ (the so-called characteristic polynomial) of degree $N-1$ for a $N \times N$ matrix $\widehat{A}$. We know that the zeros of a general real polynomial cannot be written in the form a finite number of root-expressions (Abel-Ruffini theorem). It is therefore plausible that the eigenvalues of a matrix cannot be computed in a finite number of algebraic operations — all eigensolvers are iterative.

2.6.1 Power method

The basic idea of this method is extremely simple: assume we have a diagonal $N \times N$ matrix

$$(d_A)_m n = \delta_{mn} a_m \tag{2.100}$$

and we choose the indices such that $|a_0| \leq |a_1| \leq \ldots$. Then in $(d_A)^n$, the largest in magnitude eigenvalue a_{N-1} will dominate all others. If we apply $(d_A)^k$ to any vector $\vec{c}$ that has a non-vanishing $N-1$-component c_{N-1}, then $\vec{q}_k = (d_A)^k \vec{c}$ will be a vector that is dominated by its component $\vec{q}^k_{N-1}$, i.e. nearly proportional to the eigenvector $(0, 0, \ldots, 1)$. With increasing power k, the error decreases $\sim (|a_{N-2}|/|a_{N-1}|)^k$. In case of degenerate largest eigenvalues, one at least knows that the vector is from the degenerate subspace.

The same happens for general matrices $\widehat{A}$, as one readily sees by using the spectral representation:

$$\widehat{A} = \widehat{U} \hat{d}_A \widehat{U}^\dagger = \sum_{n=0}^{N-1} \vec{U}_n a_n \vec{U}_n^\dagger \tag{2.101}$$

Assume that we pick some (almost arbitrary) initial vector $\vec{q}^{(0)} = \sum_{i=0}^{N-1} \vec{U}_i c_i$ and compute (note that $\widehat{U}\widehat{U}^\dagger = 1 \Leftrightarrow \vec{U}_n^\dagger \vec{U}_m = \delta_{mn}$):

$$\vec{q}^{(k)} := (\widehat{A})^k \vec{q}^{(0)} = \underbrace{(\widehat{U} \hat{d}_A \widehat{U}^\dagger) \cdot (\widehat{U} \hat{d}_A \widehat{U}^\dagger) \cdot \ldots}_{\text{k times}} \cdot \vec{q}_0 = \widehat{U} (\hat{d}_A)^k \widehat{U}^\dagger \vec{q}_0 = \widehat{U} (\hat{d}_A)^k \widehat{U}^\dagger \sum_{n=0}^{N-1} \vec{U}_n c_n \tag{2.102}$$

$$= \sum_{n,\beta,\gamma,\alpha} U_{\alpha\beta} \delta_{\beta\gamma} (a_\gamma)^k \delta_{\gamma n} c_n = \sum_{n,\alpha} U_{\alpha n} (a_n)^k c_n = \sum_{n=0}^{N-1} \vec{U}_n (a_n)^k c_n. \tag{2.103}$$

With $|a_0| \leq |a_1| \leq \ldots \leq |a_{N-1}|$ the eigenvector for largest eigenvalue dominates:

$$\frac{(\vec{q}^{(k)})_{N-1}}{(\vec{q}^{(k)})_{N-2}} \sim \left(\frac{a_{N-1}}{a_{N-2}}\right)^k \frac{c_{N-1}}{c_{N-2}} \tag{2.104}$$

In practice we do not need to use the spectral representation. We compute $(\widehat{A})^k \vec{v}$ and obtain $(a_{N-1})^k \vec{U}_{N-1} + \mathcal{O}(\left(\frac{a_{N-1}}{a_{N-2}}\right)^k)$. The advantages of the power method are that it is straightforward, easy to exploit for sparse matrices, and that it can rapidly converge for the largest (few) $|a_n|$. The disadvantage is that it works only for the eigenvalue(s) with largest $|a_n|$.

If we sort the eigenvalues of a matrix $\widehat{A}$ such that $a_0 \leq a_1 \leq \ldots \leq a_{N-1}$ (not $|a_n|$!), we see we can determine the lowest (few) $a_0, a_1, \ldots$ as well as the highest $\ldots, a_{N-2}, a_{N-1}$ by the power method: suppose $|a_0| < |a_{N-1}|$, then direct application of the power method will provide a_{N-1}. We can then apply the power method to $\tilde{A} := \widehat{A} - a_{N-1} \hat{I}$ to obtain the value of $\tilde{a}_0 = a_0 - a_{N-1}$.

2.6.2 Operations count, scaling of an algorithm, sparse matrices

The computational cost of the power method depends the number of iterations needed for convergence, which in turn somewhat depends on the choice of the starting vector, but mostly on the ratio

$|a_{N-1}|/|a_{N-2}|$. For the choice of the starting vector we can exploit knowledge of the solution vector, if we have any such knowledge. In many cases random choice will be the best we can do. The eigenvalue structure cannot be changed. The actual number of operations needed — "operations count" in addition depends on the number of non-zero matrix elements. For a "full" matrix $\widehat{A}$, i.e. all $\widehat{A}_{ij} \neq 0$, performing a matrix-vector multiplication $\widehat{A}\vec{c}$ requires N^2 multiplications (which is the operation that takes the most CPU time). Clearly, each matrix element counts and must be multiplied onto some component of the vector:

```
for m in range(N):
    for n in range(N):
        ac[m]=ac[m]+a[m,n]*c[n]
```

Operations count

The total number of floating point operations needed is the *operations count* of an algorithm. Usually, additions are much faster than multiplications which in turn are much faster than divisions. One should avoid doing large number of division (which can be often achieved by pre-computing the inverses outside a loop). Multiplications are at the core of most algorithms and their number determines the performance.

The operations count is what ultimately determines how long your code will run until completion. In this example, we see that the time grows unpleasantly quickly, namely quadratically, with N^2. One says, an algorithm *scales* $\sim \mathcal{O}(N^2)$. Even worse, many important algorithms in linear algebra scale with the 3rd power $\mathcal{O}(N^3)$: linear system solving, matrix inversion, eigenproblem solving, and, most prominently, matrix-matrix multiplication.

One topic that we will discuss later in more detail is that apart from operations count, in present computers the access to memory can be a second, important bottleneck for performance.

A rule of thumb

If you have composed an algorithm consisting of nested loops and you find in the innermost loop more than a single multiplication, chances are that you can move that multiplication outside the innermost loop and thus safe almost a factor 2 in computation time.

Sparse matrices

These scalings hold for *full* matrices. When a matrix is *sparse*, i.e. when it has many matrix elements $\widehat{A}_{ij} = 0$, the operations count can drop significantly. Most obviously, for matrix-vector multiplication:

```
for m in range(N):
    for n in range(N):
        if a[m,n]==0: continue  # skip multiplication
        ac[m]=ac[m]+a[m,n]*c[n]
```

The will be only as many multiplications as there are non-zero matrix elements. Even this primitive implementation can give large gains, as the check "a[m,n]==0" is much faster than multiplication.

Only when the fraction of zeros to non-zeros in the matrix becomes comparable to the ratio of CPU times for multiplication to comparison, the algorithm must be improved.

In operations other than matrix-vector multiplications, taking advantage of sparsity can be more difficult. This notably includes many linear system solving algorithms or eigensolvers. We will come back to this later.

Banded matrices

A very important class of sparse matrices are *banded* matrices where

$$\widehat{A}_{ij} = 0 \text{ for } |i-j| > b \tag{2.105}$$

with *band width* b. With this extra knowledge about matrix structure, we can (slightly) improve the algorithm for matrix-vector multiplication

```
for m in range(N):
    for n in range(max(0,m-b),min(m+b,N)):
        ac[m]=ac[m]+a[m,n]*c[n]
```

A total number of $\leq (2b+1)N$ multiplications are performed. The algorithm is $\mathcal{O}(bN)$ (small multiplicative factors like ~ 2 are omitted in describing the scaling of an algorithm.) We do not need to check our matrix elements as we know ahead of time that they are zero and we save these operations, which will result in only a minor gain in time. The gains can be much more significant for the $\mathcal{O}(N^3)$ linear algebra operations mentioned above, many $\mathcal{O}(b^2N)$ algorithms are available for banded matrices. It is good to remind oneself of the numbers: with, for example, $N = 10^4$ and $b = 100$, an algorithm for a banded matrix may be $N^2/b^2 = 10000$ times faster than the algorithm for full matrices, for example one second vs. 3 hours!

Tensor products

Suppose, a matrix $\widehat{A}$ is the tensor product of two matrices, $\widehat{A} = \widehat{B} \otimes \widehat{C}$, with dimensions $M \times M$ and $N \times N$, respectively. The dimension of $\widehat{A}$ is then $I \times I$ with $I = MN$ and the matrix elements are

$$(\widehat{A})_{ii'} = (\widehat{B})_{mm'}(\widehat{C})_{nn'}, \quad i = m + Mn,\ i' = m' + Mn'. \tag{2.106}$$

Clearly, while $\widehat{A}$ has I^2 elements, it really contains only the information of the $M^2 + N^2$ matrix elements of $\widehat{B}$ and $\widehat{C}$. A closer look shows that this can be used to dramatically reduce the operations count for the matrix-vector multiplication $\widehat{A}\vec{x}$, by the sequence operations

$$y_i = y_{mn} = \sum_{m'=0}^{M-1} (\widehat{B})_{mm'} x_{m'n} \tag{2.107}$$

$$z_i = z_{mn} = \sum_{n'=0}^{N-1} (\widehat{C})_{nn'} y_{mn'} \tag{2.108}$$

Problem 6.17:

- What is the operations count of the procedure above? Compare it with the operations count of the naive application $\widehat{A}\vec{c}$.
- Write a Python class "Tensor" that implements the above procedure and compare the actual timing to the naive implementation. Use sufficiently large matrices to see reasonable times. On Linux, you can use the "time" command. Does it scale as expected?
- Try to cast this procedure into the form of a triple matrix product

$$\widehat{Z} = \widehat{B}\widehat{X}\widehat{C}^T. \tag{2.109}$$

- Apart from writing loops, e.g. `for m in range(M):...`, one can also use the `numpy` matrix-matrix multiplication. Find and use it, compare the performance to a loop implementation.

Problem 6.18: Frequently, there arise operators in tensor product form, e.g. the single-particle operators in few- and many-particle systems. Upon discretization in appropriate bases, matrices in the product space arise that are tensor products of matrices in the factor spaces. E.g. the Hamiltonian matrix for a system of two non-interacting particles has the form

$$\widehat{H} = \widehat{H}_1 \otimes \mathbf{1} + \mathbf{1} \otimes \widehat{H}_2 \tag{2.110}$$

($\widehat{H}_1$ and $\widehat{H}_2$ will be identical, if the two particles are identical). For $\vec{c}$ from the (discretization of) the tensor product space $\mathcal{H}_1 \otimes \mathcal{H}_2$, let the size of the discretizations of $\mathcal{H}_1$ and $\mathcal{H}_2$ be N_1 and N_2, respectively, and therefore the total size of the discretization $N = N_1N_2$. Show that the matrix-vector multiplication $\widehat{H}\vec{c}$ can be implemented with operations count $\sim N_1^2N_2 + N_1N_2^2$ (assuming full matrices). Write a Python class "Tensor" that implements the tensor product algebra and tensor-vector multiplication.

2.6.3 Krylov subspace

The power method for eigenvalues is the most primitive representative of an important class of methods that use the "Krylov subspace"

$$\mathcal{K}^{(N,\vec{q}_0)} = \text{span}(\vec{q}_n := (A)^n\vec{q}_0, n = 0, 1, \ldots, N-1) \tag{2.111}$$

For the power method, although all vector are computed, only the last one $\vec{q}_{N-1}$ is eventually used for computing the eigenvalue. Many modern methods try to take maximal advantage of the of the complete Krylov subspace and build an approximate solution $\vec{x}$ as a linear combination of all Krylov vectors $\vec{q}_n$:

$$\vec{x} = \sum_{n=0}^{N-1} c_n\vec{q}_n \tag{2.112}$$

The advantage of Krylov methods it that their computations count is dominated by matrix-vector multiplications, which can be implemented efficiently for sparse matrices, i.e. matrices where many matrix element are =0. We will encounter some examples below.

Krylov subspace method are reminiscent of polynomial approximations, as any vector from the Krylov space can be written as a polynomial of the matrix applied to some starting vector $\vec{q}_0$:

$$\mathcal{K}^{(N,\vec{q}_0)} \ni \vec{x} = \sum_{n=0}^{N-1} c_n \vec{q}_n = \left(\sum_{n=0}^{N-1} c_n \widehat{A}^n \right) \vec{q}_0 \tag{2.113}$$

2.6.4 List of algorithms for eigenproblems

In general, for full matrices, the operations count is $\mathcal{O}(N^3)$. For banded matrices

$$\widehat{M}_{ij} = 0 \quad \text{for } |i-j| > b \tag{2.114}$$

the operations count is $\mathcal{O}(N(b+1)^2)$. Note that a banded matrix is a sparse matrix with non-zero elements concentrated only along certain diagonal bands. The parameter b is defined as the number of off-diagonal elements different from zero, therefore for a diagonal matrix we will have $b = 0$ and for the second order finite difference method b will equal 1 (second order: with only one off-diagonal one gets $\mathcal{E}(h) = \mathcal{O}(h^3)$).

What follows is a list of the more common algorithms used to solve eigenvalue problems:

- **QR:** It reduces the matrix into a tridiagonal form and then apply the power method to this modified matrix.
  Properties: Universal and robust (LAPACK)
  Cost: $4N^3/3$ for reduction, tri-diagonal eigenvalues $\mathcal{O}(N)$, eigenvectors $\mathcal{O}(N^2)$

- **Inverse iteration:** Iterate $\vec{c}^{(n+1)} = (\widehat{A} - \lambda_0 \mathbf{1})^{-1} \vec{c}^{(n)}$ for guess value λ_0
  Properties: Get single or a few eigenvalues (and eigenvectors) near λ_0
  Cost: $2N^3/3$ for LU factorization ($\mathcal{O}(Nb^2)$ for banded) + $\mathcal{O}(N^2)/\mathcal{O}(Nb)$ per eigenvector.

- **Jacobi:** Systematically reduce the off-diagonal matrix elements by Jacobi rotation.
  Properties: Not based on the power method, iterative, quadratic convergence, can be fast on certain matrices, nicely parallelizable.
  Cost: $\mathcal{O}(I \times N^3)$, where I can become large because of the slow convergence.

- **Lanczos:** Krylov subspace method , smart construction of an orthogonal Krylov space basis.
  Properties: Iterative, numerically quite unstable $\rightarrow$ needs careful coding and lots of "tricks".
  Cost: Best for sparse matrices or tensor products (e.g.ARPACK).

Problem 6.19: The symmetric Lanczos algorithm is a method for finding a basis of vectors $|i\rangle, i = 0, 1, \ldots, M-1$, with respect to which a symmetric $N \times N$ matrix $\widehat{A}$, $N \geq M$ is tri-diagonal

$$\widehat{T}_{ij} := \langle i|\widehat{A}|j\rangle = 0 \text{ for } |i-j| > 1 \tag{2.115}$$

One can get approximations to the eigenvalues of $\widehat{A}$ by diagonalizing the tri-diagonal $M \times M$ matrix $\widehat{T}$.

- Find the algorithm on the web.
- Explain why the Lanczos method for approximating eigenvalues can be considered a Krylov subspace method.
- Set up a random matrix with random elements, but enforce symmetry: $\widehat{A}_{ij} = \widehat{A}_{ji}$ for arbitrary N. (Use function numpy.random.random.)
- Implement the Lanczos algorithm for $\widehat{A}$ in python.
- For small N, say $N = 5$, check whether your $\widehat{T}$ is indeed tri-diagonal by computing the matrix elements $\langle i|\widehat{A}|j\rangle$.
- Verify by using $N = M = 8$ and showing that the eigenvalues of $\widehat{A}$ and $\widehat{T}$ agree. (Use linalg.eigvals for computing the eigenvalues of $\widehat{A}$ and $\widehat{T}$.)
- Increase N and investigate at which point the Lanczos algorithm breaks down.

Problem 6.20: (Teamwork) Implement the inverse iteration algorithm for a *single* eigenvalue a nearest to λ_0:

$$(\widehat{A} - \lambda_0 \mathbf{1})^{-1} \vec{b}^{(k)} = \vec{c}^{(k-1)}, \quad \vec{c}^{(k)} = \vec{b}^{(k)}/||\vec{b}^{(k)}||, \quad \tilde{a}^{(k)} = (\vec{c}^{(k)})^{\dagger} \widehat{A} \vec{c}^{(k)} \tag{2.116}$$

Use "krylov.py" (see website) as a model. For easier identification, call your code "inversiter.py". The code should have the following features:

- It must be documented!
- Give a desired accuracy ϵ and iterate only until the accuracy is reached.
- The following parameters should be read from the command line: N=matrix dimension, λ_0=guess eigenvalue, ϵ=desired accuracy
- Use the LU algorithm (linalg.lu_factor and linalg.lu_solve) to solve the system of linear equations.
- The inverse iteration result should be compared to the linalg.eig(...) result.
- Use the module mytimer.py to compare the execution times and trace which part of the code uses how much time.

Power method for several eigenvectors and values

Problem 6.21: Explore the "Krylov subspace" $\{\vec{q}^{(i)} := A^i\vec{q}, i = 0, 1, \ldots, K-1\}$: $K^{(K,M)} = \text{span}(\vec{q}^{(K-M)}, \vec{q}^{(K-M+1)}, \ldots, \vec{q}^{(K-1)})$ will contain reasonable approximations to eigenvectors for the M highest eigenvalues. Choose a random $N \times N$ matrix (i.e. a symmetric matrix with random entries $\widehat{A}_{ij} = \widehat{A}_{ji} \in [-1, 1]$) of size $N = 100$. Compute its eigenvalues and vectors. Compare to the $M = 10$ approximations to eigenvalues obtained by the generalized eigenproblem

$$\widehat{A}^{(K)}\vec{p}^{(m)} = \widehat{S}^{(K)}\vec{p}^{(m)}\tilde{a}_m^{(K)} \tag{2.117}$$

where

$$\widehat{A}_{ij}^{(K)} = \vec{q}^{(K-M+i)} \cdot \widehat{A}\vec{q}^{(K-M+j)} \tag{2.118}$$

and analogously $\widehat{S}^{(K)}$. Plot the converges $\tilde{a}_m^{(K)} \to a_m$ as a function of K. Explain the observed behavior. What is the operations count for obtaining these? How does compute time compare to the compute time needed for a direct solution by numpy.eig()?

We have a matrix $\widehat{A}$, which can be written as $\widehat{A} = \widehat{U}\hat{d}_A\widehat{U}^\dagger$. We want to find an approximation for the eigenvalues of $\widehat{A}$, which we denote by a_i. To this end we consider the $N \times M$ matrix $\hat{Q}^{(0)} = (\vec{q}_0^{(0)}, \cdots, \vec{q}_{m-1}^{(0)})$, where the $\vec{q}_i^{(0)}$ are linearly independent and we also consider the following iterative scheme:

$$\begin{aligned} \widehat{T}^{(k)} &= \widehat{A}\hat{Q}^{(k-1)} \\ \hat{Q}^{(k)} &\equiv \text{orthogonalize}(\widehat{T}^{(k)}) \end{aligned} \qquad (2.119)\text{span}$$

We observe that, since we obtained $\vec{q}$ applying a basis transformation to $\vec{t}$, we remain in the same space, i.e. $\text{span}(\vec{q}_0^{(k)}, \cdots, \vec{q}_{M-1}^{(k)}) = \text{span}(\vec{t}_0^{(k)}, \cdots, \vec{t}_{M-1}^{(k)})$. Given these definitions and this iterative scheme it is possible to see that the deviation of $\vec{q}_m^{(k)}$ from a linear combination of eigenvectors of $\widehat{A}$ (represented by $\vec{U}_n$, i.e. the columns of the matrix $\widehat{U}$) is equal to $(\frac{a_{N-m-1}}{a_{N-m}})^k$, i.e. it can be done arbitrary small by increasing k. The eigenvalues are assumed to be sorted by $|a_k| \leq |a_l|$ for $k < l$. In fact we have:

$$\vec{t}_m^{(k)} = \widehat{A}^k\vec{q}_m^{(0)} = \widehat{U}\widehat{d}_A^k\widehat{U}^\dagger\vec{q}_m^{(0)} = \widehat{U}\widehat{d}_A^k\widehat{U}^\dagger \underbrace{\sum_n \vec{U}_n c_{nm}}_{=\vec{q}_m^{(0)}} = (\widehat{U}\widehat{d}_A^k\widehat{U}^\dagger\widehat{U}\widehat{c})_m = (\widehat{U}\widehat{d}_A^k\widehat{c})_m \tag{2.120}$$

the m highest vectors will be enhanced relative to the lower ones by the factor indicated above.

Note: In this form the procedure only illustrates the principle. It is not used in practice. It requires a certain caution if the matrix contains degenerate eigenvalues, and in some cases, it may not work.

2.7 Linear system solving — the LU decomposition

A standard way of solving a system of linear equations of the form

$$\widehat{A}\vec{x} = \vec{c} \tag{2.121}$$

is by the so-called "LU-decomposition". This is the fact that any $M \times N$ matrix $\widehat{A}$ can be written as two matrices $\widehat{L}$ and $\widehat{U}$, which are lower and upper triangular, respectively

$$\widehat{A} = \widehat{L}\widehat{U}. \tag{2.122}$$

which allows writing the solution

$$\vec{x} = \widehat{U}^{-1}\widehat{L}^{-1} \tag{2.123}$$

We solve twice but with triangular matrices

$$\widehat{U}\vec{x} = \vec{y}, \qquad \widehat{L}[\widehat{U}\vec{x}] = \widehat{L}\vec{y} = \vec{c} \tag{2.124}$$

The gain is that solving a system of linear equations with triangular matrices is easy. Operations count for the decomposition is $\mathcal{O}(N^3)$ (full square matrices) and system solving requires $2 \times N^2/2$ operations. The method is particularly useful if several systems with the same matrix $\widehat{A}$ but different "right hand sides" (rhs) $\vec{c}_r, r = 0, 1, \dots, R-1$ are to be solved.

The decomposition can be chosen such that all diagonal values of $\widehat{L}$ are $\equiv 1$ (or, alternatively, all diagonal values of $\widehat{U}$) and therefore need not to be stored. The LU decomposition can then stored in the storage of the original matrix $\widehat{A}$.

Pivoting

For numerical reasons, which we do not discuss here, it is important to sort the indices of the matrix in such a way, that the "pivot" elements are as large as possible. This can be done on both sides of matrix

$$\widehat{P}\widehat{A}\widehat{Q} = \widehat{L}_f\widehat{U}_f \tag{2.125}$$

("full pivoting") with the "permutation matrices" $\widehat{P}$ and $\widehat{Q}$

$$(\widehat{P})_{i,j} = \delta_{i,p(j)}, \quad , (\widehat{Q})_{k,l} = \delta_{k,q(l)}, \quad , \tag{2.126}$$

where p and q are permutations of $j = 0, 1, \dots, M-1$ and $l = 0, 1, \dots, N-1$, respectively. Somewhat simpler is "partial pivoting"

$$\widehat{P}\widehat{A} = \widehat{L}_p\widehat{U}_p. \tag{2.127}$$

Note that the operations count for $\widehat{P}\widehat{A}$ is near negligible as all entries of $\widehat{P}$ are either 1 or 0 and no multiplications are required.

Pivoting greatly increases numerical stability of the LU-decomposition.

LU-decomposition for banded matrices

For banded matrices, the factors $\widehat{L}$ and $\widehat{U}$ can be made to be *banded*. Operations count for decomposition is $\mathcal{O}(b^2N)$, for system solving it is $\mathcal{O}(bN)$.

2.7.1 Timing of your code: how bad is N^3?

We have seen that the operations count for the eigenvalues problem goes usually as $O(N^3)$. We want to understand what does this mean and especially we would like to see how advantageous it is to have a banded matrix, for which we have seen that the operations count $\sim O(Nb^2)$. On the SVN depository trunk "mytimer.py" it can be found a program that has been written exactly to this scope. On this program one needs to define a static variable (we remind that a static variable is a variable that is kept between function calls), but this concept does not (still?) exist in Python and therefore one is obliged to use slightly alternative methods.

In any case this program is of particular importance because it contains some general ideas that are worth to be stressed:

- The importance of timing inside a program.
- How to keep variables that are inside a function between function calls.
- How to properly use tables.

2.8 Local basis sets

2.8.1 Comparison basis sets vs. grid methods

The main features of **grid methods** are:

- Banded matrix $\Rightarrow \sim O(Nb^2)$.
- Low accuracy, given by h^p where p is the order of the method and h is the step-size.
- It works only because the Schrödinger equation is often formulated as a PDE (partial differential equation), with all local operators: derivatives and multiplication operators.

The main features of the (global) **basis set** are:

- Full matrix $\Rightarrow \sim O(N^3)$.
- High accuracy for smooth solutions.
- Space of approximate solutions is well-defined: the space spanned by the basis functions is a subspace of the Hilbert space.
- (General) basis set methods do *not* make use of the locality of the SE!

2.8.2 Local basis sets — finite elements

We will now construct local basis sets that exploit the *locality* of the operators. The main idea behind this method is the splitting of the interval into small pieces (not necessarily of equal size). Then one needs to take M functions h for each interval and to approximate the solution to the differential equation as a linear combination of this M functions. Let ψ be the exact solution and $\tilde{\psi}$ an approximate solution. We will have:

$$\psi(x) \simeq \tilde{\psi} = \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} h_{mn}(x) a_{mn} \tag{2.128}$$

where h_{mn} is such that $h_{mn}(x) = 0$ for $x \notin [x_n, x_{n+1}]$, $m = 0...M-1$. The functions have their support on the **finite element** $[x_n, x_{n+1}]$ If, in particular, the functions h_{mn} are polynomials, we will have a piecewise *polynomial* approximation of $\psi(x)$.

Since we usually only deal with multiplication operators (e.g. potentials) and differential operators, discretization in terms of the localized functions leads to N disconnected problems, each of the size $M \times M$. With what we learned about operations counts, numerical handling such a system is much more efficient with scalings like, e.g. for the eigenproblem, NM^3 instead of N^3M^3.

The problem here is that $\tilde{\Psi}(x)$ is not continuous at x_n, $n = 1, 2, ..., N-1$ and therefore we cannot consistently define matrix elements, e.g. of the Laplacian $-\Delta$, for them. It would be too easy after all: we cannot approximate one large problem, where in course of time every point may impact on every other point in the problem, into many smaller sets of problems on subsets of the total points, that are group wise totally unaware of each other.

As discussed earlier, the minimal requirement is that the functions are everywhere continuous and at least differentiable almost everywhere (form domain, 1st Sobolev space). That is why we need to impose a continuity condition:

$$\lim_{\epsilon \to 0} [\tilde{\Psi}(x_n - \epsilon) - \tilde{\Psi}(x_n + \epsilon)] = 0. \tag{2.129}$$

We can easily see that

$$\tilde{\Psi}(x_n + \epsilon) = \sum_{n'=0}^{N-1} \sum_{m=0}^{M-1} h_{mn'}(x_n + \epsilon) a_{mn'} = \sum_{m=0}^{M-1} h_{mn}(x_n + \epsilon) a_{mn} \tag{2.130}$$

because $h_{mn'}(x_n + \epsilon) \equiv 0$ for $x_n + \epsilon \notin [x_{n'}, x_{n'+1}]$, i.e. for $n' \neq n$. Similarly

$$\tilde{\Psi}(x_n - \epsilon) = \sum_{m=0}^{M-1} h_{mn-1}(x_n - \epsilon) a_{mn-1}. \tag{2.131}$$

The **continuity condition** becomes

$$\lim_{\epsilon \to 0} [\tilde{\Psi}(x_n - \epsilon) - \tilde{\Psi}(x_n + \epsilon)] = \sum_{m=0}^{M-1} (h_{mn-1}(x_n) a_{mn-1} - h_{mn}(x_n) a_{mn}) = 0. \tag{2.132}$$

The condition at x_n can be now be written as

$$\vec{c}_n \cdot \vec{a} = 0 \tag{2.133}$$

with

$$\vec{c}_n = \begin{pmatrix} \vdots \\ 0 \\ 0 \\ h_{0,n-1}(x_n) \\ h_{1,n-1}(x_n) \\ \vdots \\ h_{M-1,n-1}(x_n) \\ -h_{0,n}(x_n) \\ -h_{1,n}(x_n) \\ \vdots \\ -h_{M-1,n}(x_n) \\ 0 \\ 0 \\ \vdots \end{pmatrix}, \qquad \vec{a} = \begin{pmatrix} \vdots \\ a_{M-2,n-2} \\ a_{M-1,n-2} \\ a_{0,n-1} \\ a_{1,n-1} \\ \vdots \\ a_{M-1,n-1} \\ a_{0,n} \\ a_{1,n} \\ \vdots \\ a_{M-1,n} \\ a_{0,n+1} \\ a_{1,n+1} \\ \vdots \end{pmatrix}, \qquad \mathrm{len}(\vec{a}) = N \times M. \tag{2.134}$$

This condition at the point x_n is a "linear constraint" on $\vec{a}$: we require, that the $\vec{a}$ are orthogonal to all $\vec{c}_n$. For each x_n we need one $\vec{c}_n$ and the total space of allowed coefficient vectors is a the subspace of $\mathbb{C}^{MN}$ of the vectors that are orthogonal to all $\vec{c}_n$:

$$\vec{a} \in \left\{\vec{x} \in \mathbb{C}^{NM} | \vec{c}_n \cdot \vec{x} = 0, \forall n = 0, 1, \ldots, N-1.\right\} \tag{2.135}$$

The constraint onto a subspace can always be formulated as a projection, represented by the matrix $\widehat{Q}$, such that

$$\vec{a} = \widehat{Q}\vec{x} \tag{2.136}$$

is an allowed coefficient vector for any $\vec{c} \in \mathbb{C}^{NM}$. The explicit form of these projectors is left as an exercise.

We will bring the constraint to a simple and numerically efficient form. We can make transformation $\widehat{T}_n$ of the basis within each element $[x_n, x_{n+1}]$ (compare the corresponding problem above):

$$f_{mn}(x) = \sum_{m'=0}^{M-1} \widehat{T}_{mm'} h_{m'n}(x) \tag{2.137}$$

such that

$$f_{0n}(x_n) = 1 = f_{M-1,n}(x_{n+1}) \tag{2.138}$$
$$f_{mn}(x_n) = 0 = f_{mn}(x_{n+1}) \qquad \text{for all other cases} \tag{2.139}$$

If $\widehat{T}$ is non-singular (invertible) we have $\text{span}(f_{mn}) = \text{span}(h_{mn})$. Let now $\vec{c}_n^{(f)}$ be the constraint vector with respect to the basis f:

$$\vec{c}_n^{(f)} = \begin{pmatrix} \vdots \\ 0 \\ 1 \\ -1 \\ 0 \\ \vdots \end{pmatrix}. \tag{2.140}$$

And then the continuity condition reads:

$$\vec{c}_n^{(f)}\vec{a} = a_{M-1,n-1} - a_{0n} = 0 \Leftrightarrow \boxed{a_{M-1,n-1} \equiv a_{0n}}. \tag{2.141}$$

With these conditions at each x_n, $\tilde{\Psi}(x)$ is a good ansatz for a variational calculation.

Let us now set up the matrices $\widehat{T}$ and $\widehat{V}$ with the local basis f:

$$(\widehat{T})_{mn,m'n'} = 0 \quad \text{for} \quad n \neq n' \tag{2.142}$$

$$(\widehat{T}_n)_{mm'} = \langle f_{mn}| - \frac{1}{2}\partial_x^2|f_{m'n}\rangle = -\frac{1}{2}\int_{-\frac{L}{2}}^{\frac{L}{2}} dx\, (\partial_x f_{mn}(x))(\partial_x f_{m'n}(x)) \tag{2.143}$$

$$(\widehat{V}_n)_{mm'} = \langle f_{mn}|V(x)|f_{m'n}\rangle = \int_{-\frac{L}{2}}^{\frac{L}{2}} dx\, f_{mn}(x)V(x)f_{m'n}(x). \tag{2.144}$$

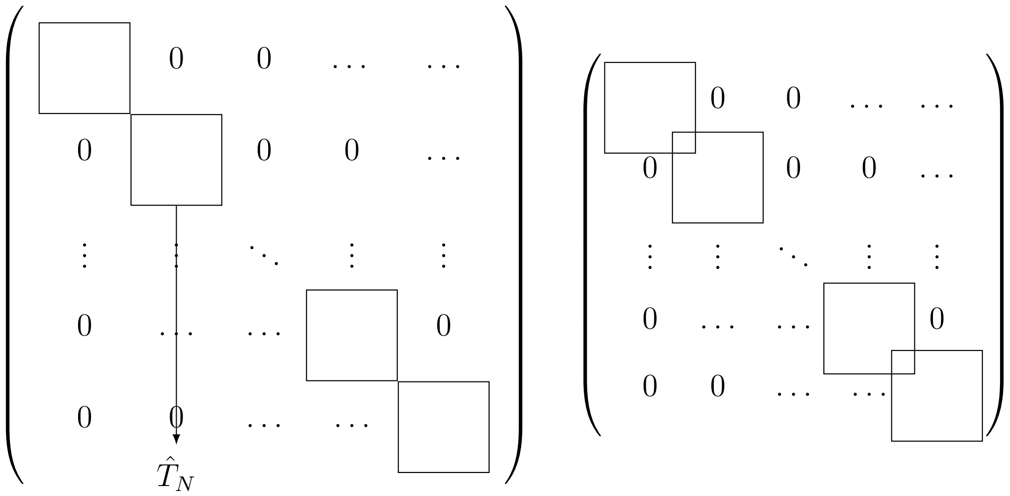

Figure 2.2: Block scheme of matrices in the f_{mn}-basis: without constraints(left) and after imposing continuity contstraints (right)

Without the continuity condition, one obtains a block scheme of the FEM matrices shown on the left of Figure 2.2. As, with our special basis f_{mn} the linear constraint for x_n reduces to identifying the index $i = Mn$ with the neighboring $i' = Mn + 1$, we obtain the matrix in the constraint space by identifying the corresponding rows and columns. The total matrix size shrinks by the number of constraints $N - 1$ and the matrix blocks overlap at their corners (Figure 2.2, right side). We get a sparse (i.e. lots of zeros in the matrix) and banded (with band width b) matrix which is divided into blocks (the $m \times M$ matrices $\widehat{T}_n$).

This is the basic idea of the *finite element method* (**FEM**). There are a few alternative local basis sets, as we will see later, but we used FEM to show the key features of the local basis sets.

In practice, we can, just add the overlapping blocks into the same large matrix. The matrix element of (local) operator $\mathbf{A}$ at the n'th corner are then

$$\int_{x_{n-1}}^{x_n} dx f_{M-1,n-1}(x)\mathbf{A}f_{M-1,n-1} + \int_{x_n}^{x_{n+1}} dx f_{0,n}(x)\mathbf{A}f_{0,n} = \tag{2.145}$$

$$\int_{x_{n-1}}^{x_{n+1}} dx[f_{M-1,n-1}(x) + f_{0,n}(x)]\mathbf{A}[f_{M-1,n-1}(x) + f_{0,n}(x)]. \tag{2.146}$$

(The cross-terms have no overlap). We see that the procedure can also be understood as replacing the non-zero boundary functions at $n = 1, \dots N - 1$ by pair-wise by "bridge-functions"

$$b_n(x) := f_{M-1,n-1}(x) + f_{0,n}(x). \tag{2.147}$$

2.8.3 Technical remark: block-matrices

It is possible to split the $I \times J$ matrix $\widehat{A}$ whose elements are given by:

$$(\widehat{A})_{ij}, \quad i = 0, 1, \dots, I - 1, \quad j = 0, 1, \dots, J - 1 \tag{2.148}$$

into blocks

$$\widehat{A} = \begin{pmatrix} \widehat{A}_{KM} & \widehat{A}_{KN} \\ \widehat{A}_{LM} & \widehat{A}_{LN} \end{pmatrix} \tag{2.149}$$

with $K + L = I$, $M + N = J$ and

$$\begin{aligned}
(\widehat{A}_{KM})_{ij} &= \widehat{A}_{ij}, \quad i = 0, 1, \dots, K - 1, \quad j = 0, 1, \dots, M - 1 && (2.150)\\
(\widehat{A}_{KN})_{ij} &= \widehat{A}_{ij}, \quad i = 0, 1, \dots, K - 1, \quad j = M + (0, 1, \dots, N - 1) && (2.151)\\
(\widehat{A}_{LM})_{ij} &= \widehat{A}_{ij}, \quad i = K + (0, 1, \dots, L - 1), \quad j = 0, 1, \dots, M - 1 && (2.152)\\
(\widehat{A}_{LN})_{ij} &= \widehat{A}_{ij}, \quad i = K + (0, 1, \dots, L - 1), \quad j = M + (0, 1, \dots, N - 1) && (2.153)
\end{aligned}$$

To see how multiplication rule works for block matrices let us introduce the $J \times R$ matrix $\widehat{B}$. Let the blocking of its left hand indices match the blocking of the right hand indices of $\widehat{A}$: $J = M + N$. Its right hand blocking $R = P + Q$ does not need to match any blocking of $\widehat{A}$:

$$\widehat{B} = \begin{pmatrix} \widehat{B}_{MP} & \widehat{B}_{MQ} \\ \widehat{B}_{NP} & \widehat{B}_{NQ} \end{pmatrix} \tag{2.154}$$

Multiplication rules (almost) like for a 2×2 matrix.

$$\widehat{C} := \widehat{A}\widehat{B} = \begin{pmatrix} \widehat{A}_{KM}\widehat{B}_{MP} + \widehat{A}_{KN}\widehat{B}_{NP} & \widehat{A}_{KM}\widehat{B}_{MQ} + \widehat{A}_{KN}\widehat{B}_{NQ} \\ \widehat{A}_{LM}\widehat{B}_{MP} + \widehat{A}_{LN}\widehat{B}_{NP} & \widehat{A}_{LM}\widehat{B}_{MQ} + \widehat{A}_{LN}\widehat{B}_{NQ} \end{pmatrix} \tag{2.155}$$

This can be easily verified by restricting the index ranges i, r to the subranges K, L and M, N, respectively, and by splitting the summations over j:

$$(\widehat{C})_{ir} = \sum_{j=0}^{J-1} (\widehat{A})_{ij}(\widehat{B})_{jr} = \sum_{j=0}^{M-1} (\widehat{A})_{ij}(\widehat{B})_{jr} + \sum_{j=M}^{M+N-1} (\widehat{A})_{ij}(\widehat{B})_{jr} \tag{2.156}$$

Formally we can write

$$\widehat{C}_{UV} = \widehat{A}_{UM}\widehat{B}_{MV} + \widehat{A}_{UN}\widehat{B}_{NV} = \sum_{W=\{M,N\}} \widehat{A}_{UW}\widehat{B}_{WV}, \quad U = K, L, \quad V = P, Q \tag{2.157}$$

Note: Difference to ordinary numbers: matrices do not commute $\widehat{A}_{UW}\widehat{B}_{WV} \neq \widehat{B}_{WV}\widehat{A}_{UW}$, matrices have a transpose $A^T \neq A$ in general, so we can not (always) treat block matrices as ordinary numbers.

Note: Many algorithms that work for ordinary number matrices can be generalized to block-matrices (e.g. LU decomposition). The block-algorithms are often more efficient computationally, sometimes more stable and often better readable.

Transpose of a block matrix

$$\begin{pmatrix} \widehat{A} & \widehat{B} \\ \widehat{C} & \widehat{D} \end{pmatrix}^T = \begin{pmatrix} \widehat{A}^T & \widehat{C}^T \\ \widehat{B}^T & \widehat{D}^T \end{pmatrix} \tag{2.158}$$

All of this, of course, works for any blocking of a matrix. Only the summation index $J = M + N$ must be blocked in the same way for both matrices.

Transformation matrix to finite element functions

A nice application of the formalism developed for block matrices is given by the finite elements method. Let's suppose we have only two points x_0 and x_1 and M functions per interval. We will have the functions $h_m(x_0)$, $h_m(x_1)$, $m = 0..M-1$. Let's now introduce the $2 \times M$ matrix $\widehat{B}$, whose elements are given by $(\widehat{B})_{bm} = h_m(x_b)$, $b = 0, 1$. We will now split this matrix into blocks and we will show how to obtain a transformation for the finite element basis functions.

Now with $I = 2$ (no blocking on the first index) and blocking $J =: M = 2 + K$ (here subscripts indicate the sizes of the individual blocks) we can write:

$$\widehat{B} = \begin{pmatrix} (\widehat{B}_0)_{22} & (\widehat{B}_1)_{2K} \end{pmatrix} \tag{2.159}$$

$$\widehat{T} = \begin{pmatrix} (\widehat{B}_0^{-1})_{22} & -(\widehat{B}_0^{-1}\widehat{B}_1)_{2K} \\ 0_{K2} & \mathbf{1}_{KK} \end{pmatrix} \tag{2.160}$$

where 0_{K2} denotes a $K \times 2$ matrix of zeros, $\mathbf{1}_{KK}$ the $K \times K$ identity matrix. For the transformation to finite element basis functions :

$$\widehat{B}\widehat{T} = \Big((\widehat{B}_0 \widehat{B}_0^{-1})_{22} + (\widehat{B}_1)_{2K} 0_{K2}, \quad -(\widehat{B}_0 \widehat{B}_0^{-1})_{22} \widehat{B}_1 + (\widehat{B}_1)_{2K} \mathbf{1}_{KK} \Big) = \big(\mathbf{1}_{22}, \quad 0_{2K} \big) \tag{2.161}$$

Note: Matrix $\widehat{T}$ is not unique.

2.8.4 Implementation of the FEM

Now we are interested in how to put the FEM into code. As we said we can write the single blocks (the $M \times M$ matrices $\widehat{T}_n$) of the matrix $\widehat{T}$ as:

$$(\widehat{T}_n)_{mm'} = \langle f_{mn} | -\frac{1}{2}\partial_x^2 | f_{m'n} \rangle. \tag{2.162}$$

In Python we get the matrix $\widehat{T}$ by the following code:

$$\widehat{T}[i0 : i1, i0 : i1] \mathrel{+}= \widehat{T}_n, \tag{2.163}$$

where the += operation is defined in the following way:

$$\widehat{T}[...] = \widehat{T}[...] + \widehat{T}_n. \tag{2.164}$$

We use this operation because, as said before, there is an overlap of the single blocks in the matrix $\widehat{T}$. $i0$ and $i1$ depend on n and give the position of the $M \times M$ matrix $\widehat{T}_n$ (single block) in the big matrix $\widehat{T}$. For the $\widehat{S}$ matrix we do the same:

$$(\widehat{S}_n)_{mm'} = \langle f_{mn} | f_{m'n} \rangle, \quad \widehat{S}[i0 : i1, i0 : i1] \mathrel{+}= \widehat{S}_n. \tag{2.165}$$

Now we have to compute the matrices $\widehat{T}_n$ and $\widehat{S}_n$. We can do that by using the quadrature. If we do this for the matrix $\widehat{S}_n$ we get the following result:

$$(\widehat{S}_n)_{mm'} = \sum_{i=0}^{P-1} f_{mn}(x_i) f_{m'n}(x_i) w_i = \sum_i v_{im} w_i v_{im'} = \widehat{V}^T \widehat{d}_w \widehat{V}, \quad v_{im} = f_{mn}(x_i), \tag{2.166}$$

where P is the number of quadrature points, x_i are the quadrature points and w_i the quadrature weights. It is now easy to do it for the matrix $\widehat{T}$. The result is:

$$\widehat{T} = \frac{1}{2} \widehat{D}^T \widehat{d}_w \widehat{D}, \quad (\widehat{D})_{im} := \partial_x f_{mn}(x_i). \tag{2.167}$$

So we need to compute the matrices $\widehat{V}$ and $\widehat{D}$ defined as:

$$(\widehat{V})_{im} = f_{mn}(x_i) \tag{2.168}$$

$$(\widehat{D})_{im} = \frac{\partial}{\partial x} f_{mn}(x_i) \tag{2.169}$$

$$\tag{2.170}$$

This calculation can be looked up in the file "basisfunction.py".

Note: We must stress the fact that specifying the Jacobian when a change of variable for an integral is performed is not enough, also boundary conditions play an important role. To illustrate better this concept let's consider the integral $\int_{-\infty}^{+\infty} f(x)g(x)\,dx$: in polar coordinates the radial component will be: $\int_0^{+\infty} r^2\phi(r)\chi(r)\,dr = \int_0^{\infty} U(r)V(r)\ dr$ where we defined U and V such that: $r\phi(r) = U(r)$ and $r\chi(r) = V(r)$. The last expression shows that if we use the functions U and V we will have a Jacobian $= 1$ and boundary conditions $U(0) = 0$, $V(0) = 0$, whereas if we decide to work with the functions ϕ and χ we will have a Jacobian equal to r^2 and the boundary conditions $\phi(0)$ and $\chi(0)$ can be finite.

2.8.5 The "BasisFunction" object

In addition to what was already introduced in the exercises, for every set of BasisFunction we can compute the matrices

```
|         ...int j(q)dq v^* v
d|d       ...int j(q)dq dv^* dv
|1/q^2|   ...int j(q)dq v^* 1/q^2 v
|1/q|     ...int j(q)dq v^* 1/q v
```

etc.. Note that the object potential(q) is defined elsewhere.

Problem 8.22: Extend BasisFunction by an instance "FDGrid" to incorporated the 2nd order finite difference scheme into the BasisFunction on the same level as LegendreScaled and LaguerreExpon. Extend your own solution (preferred) or use the model solution provided on the SVN repository trunk.

For assembling the overall matrices from the matrices on each interval (as a limiting case there may also be just a single interval) we define the

"Axis" object

Problem 8.23: An axis is an array of elements, including the case of a single element. It contains all information about the discretization of a single coordinate axis. Write this class with the basic properties

```
name... x,r,etc. (determines boundary conditions, jacobian,
                 maximal lower and upper boundaries)
n ..... number of discretization coefficients
lb..... actual lower boundary of the axis
ub..... actual upper boundary of the axis
e ..... array of one or several finite elements or a single set of BasisFunction
```

and a method

```
matrix(string)  ...return a discretized matrix as a numpy matrix
```

where “string” defines which matrix elements are formed. It is advisable to use the same “string” definitions as in basisfunction.py. Use the class to solve the Schrödinger equation of the (3d) harmonic oscillator and the hydrogen atom in polar coordinates and for angular momentum $L = 0$. (All needed codes, exept for axis.py, are available on the SVN repository.)

Later more will be added, like overlap_inv(), and there will be a couple of auxiliary routines like show() for plotting, read() for reading the axis definitions from file, and things like type(), which returns the axis type 'fem','scaledlegendre','fdgrid', etc.

Note: Avoid information doubling: i.e. we could just define a variable self.type, which just carries the string that the method type() returns. However, we then always need to make sure that the information there corresponds to the actual type. The method type() just looks at the actual type of the data “self.e” and determines the type, so that no error is possible.

Note: This principle is violated, for convenience, by the variable self.n.

2.8.6 Locality vs. orthogonality

We have no problem with orthogonality when we do not require continuity. The Legendre polynomials, for example, are orthogonal (but not continuous). But if we want to implement differential operators, continuity is the minimum requirement that we need. What we can do to get continuity is to construct a new function on which we impose the continuity condition $a_{10} \equiv a_{01}$. Effectively we have a single basis function that has both elements as a support:

$$\tilde{h}_{01,10} = h_{01} + h_{10}. \tag{2.171}$$

There are two points of view to look at this:

- We impose the continuity condition on $\vec{a}$.
- We merge two functions, i.e. h_{01} and h_{10}, into one single new basis function $\tilde{h}_{01,10}$.

Without continuity condition the blocks are independent. We can make each block orthogonal by a similarity transformation and end up with a nice diagonal matrix. Because of the continuity condition we move the blocks up and thus the single blocks are not independent. This kind of overlap matrix we can bring to an almost diagonal matrix. This diagonal matrix is orthogonal except for the ■ and shown in Figure **??**.

2.8.7 Other local basis functions: B-splines, wavelets

B-splines

A different method for obtaining piece-wise local approximations for a solution are B-splines. In general, a piece-wise polynomial approximation can be written as a sum of po $b_n(x)$ of degree $M-1$

$$\Psi(x) \approx \sum_{m=0}^{N-1} b_m(x) a_m, \tag{2.172}$$

where each b_m has support only on a finite interval. One needs to make sure that sufficient continuity is guaranteed. In FEM we did this by imposing a constraint on the coefficients. In B-splines, one uses for the $b_m(x)$ functions that go smoothly to zero at their interval boundaris and one makes sure that the intervals overlap.

One constructs such $b_m(x)$ by starting from a single polynomial of degree $M-1$ and shifting (and optionally scaling) it across the x-axis. Let $b_0(x)$ be a polynomial on an interval $[0, s]$ with boundary conditions $b_0(0) = b_0(s) = 0$. Outside the interval the $b_0(x)$ is equal to zero. We now shift the polynomial by intervals s/M:

$$b_i(x) = b_0\left(x - i\frac{s}{M}\right), \tag{2.173}$$

see Figurefig4.

The first $M-1$ of these functions all share the interval $[\frac{s(M-1)}{M}, s]$. On that interval, the linear combination

$$\Psi(x) \approx \sum_{m=0}^{M-1} b_m(x)a_m, \quad , x \in [\frac{s(M-1)}{M}, s] \tag{2.174}$$

is again a polynomial of degree $M-1$, parametrized by M coefficients. It is easy to see that all b_m are linearly independent, therfore the parametrization of $\Psi(x)$ is an arbitray polynomial. This holds for any s/M-sized interval, where M of the b_m overlap. The end of the simulations box requires some speciat treatment. In the simplest case, say for an axis $[-L/2, L/2]$ one may just slowly let the approximation degrade to the point such that for the first and last sub-intervals only the single $b_0(x)$ and $b_{N-1}(x)$, respectivly, are non-zero.

We see that B-splines are very similar to FEM. One difference is that the non analyticity occurs at every point where one $b_m(x)$ reaches its interval boundary. In contrast, in FEM, M basis functions share the interval and non-analyticities only occur at the boundaries of this joint interval.

The main technical difference to FEM is that the matrices become **evenly banded** with band-width $M-1$ in B-splines. FEM has also band-width $M-1$, but the matrix has extra zeros within the band (**block-like structure**). Both methods are largely used and there are minor, mostly technical differences. FEM may be more flexible because the method is formulated for general functions and not only for polynomials.

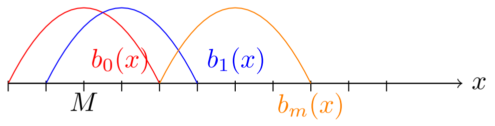

Figure 2.3: B-splines

Let us have a look at the overlap matrix of the B-splines. It holds that

$$\langle b_i | b_j \rangle = 0 \text{ for } |i-j| \geq M. \tag{2.175}$$

That means that the overlap matrix is a band matrix which is shown in Figure 2.4.

Problem 8.24: Implement simple 3rd and 5th order B-spline schemes as follows:

- Choose $B_n^{(0)}(x) = x^n$

Wavelets

The idea of the wavelets is that you take one wavelet (like B-splines). The wavelet can be chosen local. Then you shift it (like B-splines) across the axis. Again we have one initial basis wavelet $w_0(x)$. By shifting we get the other wavelets: $w_M = w_0(x - s)$. But we do not only shift the wavelets (like uniform B-splines), but also scale some initial basis function:

$$w_{sl}(x) = 2^s w_{00}(2^s x - l), \tag{2.176}$$

where l sets the wavelet and s tells us how large the wavelet is.

The advantage is that we get high local resolution where we need it because we can approximate smooth parts by a few wavelets and oscillating parts by increasingly more scaled wavelets. Presently, it is little used in practice for solving differential equations of physics. But it is very important in signal analysis and data compression (”harmonic analysis”). We will discuss this later more in detail when we discuss Fourier representations and signal analysis.

2.8.8 Comparison FEM (or other local basis functions) with FD

Problem 8.25: Show that any finite element basis $f_{mn}(x)$ can be transformed such that the overlap matrix elements take the following shape (for the notation, see lecture notes):

$$(S_n)_{mm'} = \langle f_{mn}|f_{m'n}\rangle = d_{mn}\delta_{mm'} + o_n\delta_{m+M-1,m'} + o_n\delta_{m-M+1,m'} \tag{2.177}$$

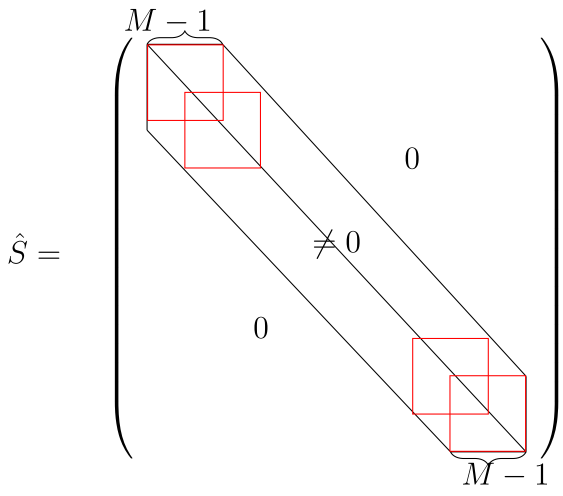

Figure 2.4: Overlap matrix with B-splines. Red square shows how the matrix S looks like for the FEM.

i.e. it is diagonal except for one off-diagonal element that connects the first function of the element to the last. Find an algorithm to do this and implement it in python. Is this construction unique? Or is there still freedom that could be used, e.g. to get get other matrices sparser?
Hint: Construct f_{0n} and f_{M-1n} such that they are orthogonal to all remaining f's. Then orthogonalize the "inner functions" $m = 1, \dots M-2$.

Problem 8.26: Show that overlap matrix can be written in the form

$$\widehat{S} = \widehat{Q}\widehat{S}_0\widehat{Q}, \tag{2.178}$$

where $\widehat{Q} = 1 - \widehat{P}$ projects any vector $\vec{a}$ onto a vector that obeys the all continuity conditions $\vec{a} \cdot \vec{c}_n = 0$ for $n = 1, \dots N-1$. Give the explicit form of $\widehat{P}$.

Problem 8.27: (This may be a tough one...) The inverse $\widehat{S}^{-1}\widehat{S} = \widehat{Q}$ on the space of vectors $\vec{a}$ that obey the continuity conditions $\widehat{Q}\vec{a} = \vec{a}$ (from which follows $\widehat{P}\vec{a} = 0$) can be written as

$$\widehat{S}^{-1} = \widehat{Q}\widehat{S}_0^{-1}\widehat{Q} - \widehat{Q}\widehat{S}_0^{-1}\widehat{P}(\widehat{P}\widehat{S}_0^{-1}\widehat{P})^{-1}\widehat{P}\widehat{S}_0^{-1}\widehat{Q}. \tag{2.179}$$

How large are the matrices that need to be inverted? Assuming N elements and the M functions on each element chosen in the "almost diagonal" form discussed above, what is the operations count for this algorithm? What is the cost of applying $\widehat{Q}$, $\widehat{P}$ and the inverses? Prove the identity. Implement the procedure.
Hint: For the prove write $\widehat{Q} = \mathbf{1} - \widehat{P}$, then it is just matrix algebra to show that $\widehat{S}\widehat{S}^{-1} = (\mathbf{1} - \widehat{P}) = \widehat{Q}$.

Note: As $\widehat{Q}$ is a projector, the matrix $\widehat{S} = \widehat{Q}\widehat{S}_0\widehat{Q}$ does not have an inverse on the complete space of $\vec{a}$'s. However, when restricted to the space of vectors that obey the continuum conditions $\widehat{Q}\vec{a}$ (the "range" of $\widehat{Q}$), an inverse can be defined as "$\widehat{S}^{-1}\widehat{S} = \widehat{Q}$" where $\widehat{Q} \neq \mathbf{1}$ is the projector.

Note: I have never seen this identity in this particular form in literature, but it is closely related to the "Woodbury" formula for updating inverses of matrices, which in turn is member of a whole family of low-rank updates for large matrices.

There is no orthogonal FEM basis

Suppose we have transformed our basis set to the "almost diagonal" form discussed above. Now we can try to proceed and construct a fully orthogonal basis with the boundary conditions as defined above. Therefore we must not linearly combine $f_{0n}(x)$ with $f_{M-1,n}(x)$. We can combine either of them with functions $f_{im}, i = 1, M-2$, but that will introduce non-vanishing overlaps $(\widehat{S}_n)_{01} = \langle f_{0n} | f_{in} \rangle$. So, up to orthogonal transformations between the "inner" $f_{in}(x)$ the "almost diagonal" representation is unique.

Now let us assume we DO have a local orthogonal continuous basis. Then, by our standard construction, we can transform it to the "almost diagonal" form. But we just showed above that no invertible such transformation exists. Therefore there exists no local continuous orthogonal basis.

The only requirement for the construction is that the B_0 matrix of boundary values of at least two of the functions is invertible. However, that condition just means that the values on one element boundary can parameterized independently of the values at the other boundary. This clearly is a prerequisite for any local basis set: we must not have an a priori fixed relation between values at two arbitrary points of our solution.

This does it for the FEM, similarly for the B-splines. I believe, it is general for any local basis, but still hoping for confirmation from the mathematical side.

2.8.9 DVR - "Discrete Variable Representations"

Choose a grid on some interval $[a, b]$ such that the x_i's are the quadrature points of some quadrature rule (e.g. orthogonal polynomials). The M^{th} order polynomial is exactly defined if we know its values at x_i. The ansatz that we make is the following:

$$\Psi(x) \approx P_{M-1}(x), \quad a_i = P_{M-1}(x_i), \tag{2.180}$$

where x_i are the quadrature points of some Gauss-quadrature. This is the parameterization of our wave function $\Psi(x)$ and uniquely defines $\Psi(x)$. The gain of the DVR is that we can easily do integrals:

$$\int |\Psi|^2 \, dx = \sum_{i=0}^{M-1} |a_i|^2 w_i. \tag{2.181}$$

It is also easy to compute the overlap integral on an interval $[a, b]$:

$$\int_a^b dx \, v(x) \Psi^*(x) \Psi(x) = \sum_{i=0}^{M-1} w_i |\Psi(x_i)|^2, \tag{2.182}$$

where $v(x)$ is the weight belonging to the orthogonal polynomials. The integrals are **exact** if $\Psi(x)$ is a polynomial of degree $M-1$ (i.e. oder M). Suppose now that, for example, we compute a potential x^2 on an interval $[a, b]$:

$$\int_a^b dx \, v(x) |\Psi(x)|^2 x^2 \approx \sum_{i=0}^{M-1} w_i |\Psi(x_i)|^2 x_i^2. \tag{2.183}$$

The polynomial $\Psi(x)$ will be of degree $M-1$, i.e. $\Psi(x)^2 x^2$ is of degree $2(M-1)+2 = 2M$. The integral is not anymore exact and we make a (small) **quadrature error**. Suppose that we choose $M = 100$. Loosing one point may not make a big difference if $\Psi(x)$ is smooth. So the error is small and can usually be neglected for practical purposes. However, we do lose rigorous upper bounding property of our variational ansatz. Unfortunately the representation of **differential operators** will in general be represented by **full** matrices.

Assume that we have a set of orthogonal polynomials $Q_n, n = 0, 1, \ldots, N-1$ and the corresponding N-point quadrature rule $(x_i, w_i), i = 0, 1, \ldots, N-1$. Then our wave function can be represented by:

$$\Psi_{\vec{b}}^{(Q)}(x) = \sum_{n=0}^{N-1} Q_n(x) b_n = \vec{Q}(x) \cdot \vec{b}. \tag{2.184}$$

Then the corresponding DVR representation in terms of $\vec{a}^T = (a_0, a_1, \ldots, a_{M-1})$ simply is:

$$\Psi_{\vec{b}}^{(Q)}(x_i) = \sum_{n=0}^{N-1} Q_n(x_i) b_n = \vec{Q}(x_i) \cdot \vec{b} = a_i \tag{2.185}$$

or

$$\vec{a} = \widehat{Q}\vec{b}, \quad \widehat{Q}_{in} = Q_n(x_i). \tag{2.186}$$

It is easy to see that $\widehat{Q}$ is invertible and just generates a similarity transformation, i.e. a change of basis.

Let us now define some **new polynomials**:

$$P_i(x) := \sum_{n=0}^{N-1} w_i (\widehat{Q})_{in} Q_n(x), \quad \text{or} \quad \vec{P}(x) = \widehat{d}_w \widehat{Q} \vec{Q}(x), \tag{2.187}$$

where $(\widehat{d}_w)_{ij} = \delta_{ij} w_i$.

Note: In general, the degree of $P_i(x)$ is $> i-1$, i.e. for example P_i is a polynomial of degree $N-1$. With these new polynomials the wave function can be written in the following way:

$$\Psi(x) = \sum_{n=0}^{N-1} P_i(x) a_i =: \vec{P}(x) \cdot \vec{a}. \tag{2.188}$$

The **overlap matrix of the** $P_i(x)$ on an interval $[a, b]$ is given by:

$$\int_a^b dx\, v(x) P_i(x) P_j(x) = \sum_{k=0}^{N-1} w_k P_i(x_k) P_j(x_k) = \sum_{k=0}^{N-1} w_k \delta_{ik} \delta_{jk} = w_i \delta_{ij}. \tag{2.189}$$

In the last step we used that $\Psi(x_j) = a_j$ for x_j being a quadrature point. This means that the polynomials P_i are orthogonal and thus $P_i(x_j) = \delta_{ij}$. The Lagrange polynomials obey these properties. So the new basis are the Lagrange polynomials for the points x_i.

FE-DVR

The idea is to pick for the FE polynomials the Lagrange polynomials on a quadrature grid where two points are fixed at the interval boundaries (**Lobatto quadrature**). This is a FE basis with:

$$\begin{aligned} L_0(x_0) =& L_{M-1}(x_{M-1}) = 1 \\ L_m(x_0) =& L_m(x_{M-1}) = 0 \quad \forall m \neq 0, M-1 \end{aligned} \qquad \text{(2.190)span}$$

The overlap matrix will now look like:

$$\sum_{i=0}^{M-1} w_i L_m(x_i) L_n(x_i) = \delta_{mn} w_n \tag{2.191}$$

We have orthogonal polynomials for the FEM, but this contradicts all the things we said above about the fact that continuity requirements destroy orthogonality. The catch is that if we specify M points in the Lobatto quadrature we obtain an accuracy only up to degree $(2M-3)$. This happens because by fixing two points (those at the interval boundary) we lose two degrees of freedom.

2.9 Higher (two-)dimensional problems

The Hamiltonian of the harmonic oscillator in 2 dimensions is

$$H(x_1, x_2) = \sum_{i=1}^{2} -\frac{1}{2}(\partial_x^2)_i + \frac{1}{2}x_i^2 = H(1) + H(2) \tag{2.192}$$

More precisely

$$H^{(2)} = H^{(1)} \otimes \mathbf{1} + \mathbf{1} \otimes H^{(1)} \tag{2.193}$$

where $\otimes$ represents the tensor product.

The tensor product structure of the operator can be exploited computationally, if we choose the discretization of the two-dimensional Hilbert space $\mathcal{H}$ in product form, i.e.:

$$b_i^{(2)}(x_1, x_2) = b_{m_i}^{(1)}(x_1)b_{n_i}^{(1)}(x_2) = b_{m_i}^{(1)}(x_1) \otimes b_{n_i}^{(1)}(x_2) \tag{2.194}$$

In this basis, the Hamiltonian matrix has tensor product form

$$(\widehat{H}^{(2)})_{ij} = \langle b_i^{(2)}|H^{(2)}|b_j^{(2)}\rangle = \langle b_{m_i} \otimes b_{n_i}|H^{(1)} \otimes \mathbf{1} + \mathbf{1} \otimes H^{(1)}|b_{m_j} \otimes b_{n_j}\rangle \tag{2.195}$$

$$= \langle b_{m_i}|H^{(1)}|b_{m_j}\rangle\langle b_{n_i}|b_{n_j}\rangle + \langle b_{m_i}|b_{m_j}\rangle\langle b_{n_i}|H^{(1)}|b_{n_j}\rangle. \tag{2.196}$$

In compact notation, $\widehat{H}^{(2)}$ can be written as the sum of two tensor products

$$\widehat{H}^{(2)} = \widehat{H}^{(1)} \otimes \widehat{S}^{(1)} + \widehat{S}^{(1)} \otimes \widehat{H}^{(1)} \tag{2.197}$$

with $(\widehat{H}^{(1)})_{km} = \langle b_k^{(1)}|H^{(1)}|b_m^{(1)}\rangle$. Similarly for $\widehat{S}^{(2)} = \widehat{S}^{(1)} \otimes \widehat{S}^{(1)}$

2.9.1 Tensor product

Tensor product space

As a brief reminder, the tensor product space $\mathcal{H} = \mathcal{H}_1 \otimes \mathcal{H}_2$ of two Hilbert spaces $\mathcal{H}_1$ and $\mathcal{H}_2$ is the space spanned by all linear combinations of tensor products of functions $\phi_i \in \mathcal{H}_1$ and $\chi_j \in \mathcal{H}_2$:

$$\mathcal{H} \ni \Psi = \sum_{ij} c_{ij}\phi_i \otimes \chi_j \tag{2.198}$$

The scalar product for single tensor products in $\mathcal{H}$

$$\langle \phi_i \otimes \chi_j|\phi_k \otimes \chi_l\rangle = \langle \phi_i|\phi_k\rangle\langle \chi_j|\chi_l\rangle \tag{2.199}$$

and for general functions $\Psi, \Phi \in \mathcal{H}$ by using the anti-bilinearity of the scalar product

$$\langle \Psi|\Phi\rangle = \sum_{ij,kl} c_{ij}^* c_{kl}\langle \phi_i \otimes \chi_j|\phi_k \otimes \chi_l\rangle \tag{2.200}$$

Tensor product of operators

Let the operators be $A_1 : \mathcal{H}_1 \to \mathcal{H}_1$ and $A_2 : \mathcal{H}_2 \to \mathcal{H}_2$, then the tensor product of $T = A_1 \otimes A_2$ acts on any tensor product $\phi \otimes \chi$ as

$$(A_1 \otimes A_2)(\phi \otimes \chi) = (A_1\phi) \otimes (A_2\chi) \tag{2.201}$$

The action on general $\Psi \in \mathcal{H}$ is defined by linearity, similarly as in the case of the scalar product.

Tensor product of two matrices

$$\left(\widehat{A} \otimes \widehat{B}\right)_{kl,mn} = (\widehat{A})_{km}(\widehat{B})_{ln}, \quad i = 0, ..KL, j = 0..MN \tag{2.202}$$

The new $KL \times MN$ matrix is very large, so we see that things get quite complicated in more dimensions as problem size grows exponentially with the number of dimensions (*curse of dimensionality*).

Straightforward implementation

The tensor class in the program tensor.py represents a tensor operator as a list of tensor products of factor matrices and scalar multiplicative factors:

$$T = \sum_i c_i \widehat{A}_i \otimes \widehat{B}_i \tag{2.203}$$

In the python program we can use *operator overloading* for notational clarity use in general algorithms. when defining `__add__`. If we let T, S be tensors then

$$T + S = \sum_{i=0}^{I-1} c_i \widehat{A}_i \otimes \widehat{B}_i + \sum_{j=0}^{J-1} d_j \widehat{C}_j \otimes \widehat{D}_j = \sum_{k=0}^{I+J-1} e_k \widehat{U}_k \otimes \widehat{V}_k \tag{2.204}$$

and this operation is implemented by writing `T+S` wich is equivalent to the function `T.__add__(S)`

2.9.2 Polar coordinates in 2 dimensions

Problem 9.28: Plane waves and trigonometric functions are well-suited as basisfunction for periodic problems. Let $p_n(x) = \exp[2\pi i x(n - N/2)/L]$, $n = 0, 1, \ldots, N-1$, N even, and let $\Psi(x)$ be a periodic function $\Psi(x) = \Psi(x + L)$. Then the basis set expansion

$$\Psi(x) \approx \sum_{n=0}^{N-1} p_n(x) c_n = \vec{p}(x) \cdot \vec{c}, \quad c_n = \frac{1}{L} \int_0^L dx p_n^*(x) \Psi(x) \tag{2.205}$$

is the (truncated) Fourier series.

- Show that the points $x_j := jL/N$, $j = 0, 1, \ldots, N-1$ and weights $w_j \equiv L/N$ provide a quadrature rule
$$\int_0^L dx f(x) \approx \sum_{j=0}^{N-1} w_j f(x_j) \tag{2.206}$$
which is exact for the $2N-1$ dimensional space of functions
$$f_{\vec{c}}(x) = \sum_{m=-N+1}^{N-1} e^{2i\pi xm/L} c_m. \tag{2.207}$$

  Hint: It is sufficient to show that
$$\int_0^L dx e^{2i\pi xm/L} = \sum_{j=0}^{N-1} w_j e^{2i\pi x_j m/L} = \delta_{0m} L. \tag{2.208}$$

  Note: This means in particular that all overlaps $\langle p_n | p_m \rangle$ can be evaluated exactly with N points of this quadrature.

  Note: Close analogy, but also difference to Gauss quadratures: an N-point Gauss quadrature is exact on the polynomials up to degree $2N-1$, which is a $2N$ dimensional function space (one more!).

- Give the matrix representation of $-i\partial_x$ in this basis.

- Assume N even and construct a purely real basis of sines and cosines
$$q_n(x) = \begin{cases} (p_{n/2}(x) + p_{N-n/2}(x))/2 & \text{for } 0 < n \text{ even} \\ (p_{(n+1)/2}(x) - p_{N-(n+1)/2)}(x))/2i & \text{for } 0 < n \text{ odd} \end{cases} \tag{2.209}$$
and $q_0 \equiv 1$.

- Give the matrix representation of $-i\partial_x$ in the sine-cosine basis.

Problem 9.29: Implement 2-dimensional polar coordinates, for which the kinetic energy term has the form
$$-\frac{1}{2}\frac{1}{\rho}\partial_\rho \rho \partial_\rho - \frac{1}{2}\frac{1}{\rho^2}\partial_\phi^2 \tag{2.210}$$
and solve the 2d harmonic oscillator in these coordinates. For the implementation, go through the following steps:

1. In "axis.py", introduce two new axis types "rho" on $[0, \infty)$ and "phi" on $[0, 2\pi]$ with the appropriate boundary conditions and Jacobians.
2. In "axis.py", add an axis kind=='trigon'.

3. In “basisfunction.py” introduce a new basis set “CosSin” defined as follows

$$b_n(\phi) = \begin{cases} \cos(\pi n \phi) & \text{for } n \text{ even} \\ \sin(\pi(n+1)\phi) & \text{for } n \text{ odd} \end{cases} \tag{2.211}$$

4. Use as the “quadrature rule” for the trigonometric functions the rule established above.

5. Extend “se2d.py”: define axes “arho” and “aphi”, set up the factors of the tensor product, and put together the total Hamiltonian as

$$\frac{1}{2}[(\widehat{D}^{(\rho)} + \widehat{P}^{(\rho)}) \otimes \widehat{S}^{(\phi)} + \widehat{Q}^{(\rho)} \otimes \widehat{D}^{(\phi)}] \tag{2.212}$$

($\widehat{D}, \widehat{P}, \widehat{Q}, \widehat{S}$ denoting derivative, potential, $1/\rho^2$, and overlap matrices, respectively.

How does sorting of the coordinates ρ, ϕ or ϕ, ρ affect the matrix band width?

2.9.3 An atom in external fields

We will now discuss the problem of an atom in external fields. It is a simple, but in general not solvable problem. We will consider an static external electric field and thus have to discuss the linear and quadratic Stark effect.

Static electric field

The Hamiltonian in the case of an atom in a static external field reads as follows:

$$H_{\vec{\mathcal{E}}} = -\frac{1}{2}\frac{1}{r}\partial_r^2 r + \frac{1}{2r^2}\mathbf{L}^2 - \frac{1}{r} - \vec{\mathcal{E}} \cdot \vec{r} \tag{2.213}$$

We want to calculate the **eigenvalues** of this problem. For simplicity we choose the coordinate system in such a way that the direction of the field is the z-direction, i.e. $\vec{\mathcal{E}} = (0, 0, \mathcal{E})$. Thus $\vec{\mathcal{E}} \cdot \vec{r}$ becomes $\mathcal{E}z$ with $z = \cos\Theta r$. The **discretization** of the Hamiltonian is:

$$\widehat{H} = \widehat{T} + \widehat{V} + \mathcal{E}\widehat{D}, \tag{2.214}$$

where $\widehat{D}$ is the dipole interaction. The kinetic energy term can be written in the following form:

$$\widehat{T} = \widehat{T}^{(r)} \otimes \mathbf{1}^{(\Omega)} + \widehat{Q}^{(r)} \otimes \widehat{L^2}^{(\Omega)}. \tag{2.215}$$

While the potential and the dipole interaction can be written as:

$$\widehat{V} = \widehat{V}^{(r)} \otimes \mathbf{1}^{(\Omega)}, \tag{2.216}$$

$$\widehat{D} = \widehat{D}^{(r)} \otimes \widehat{D}^{(\Omega)}. \tag{2.217}$$

We use the standard notation of the polar coordinates, where $\Omega = (\Theta, \varphi)$. As the **basis** we choose for the radial coordinate (r) the FE and for the angular coordinates (Ω) spherical harmonies Y_{lm}. As the spherical harmonics are eigenfunctions of $\mathbf{L}_z$ and $\mathbf{L}^2$, the Laplacian becomes diagonal with respect to the discretization in l, m.

To get the matrix of the dipole interaction $\widehat{D}$ we compute:

$$\langle i|z|j\rangle = \langle b_{n_i}(r)|\langle Y_{l_i m_i}(\Omega)|r\cos\Theta|b_{n_j}(r)\rangle Y_{l_j m_j}(\Omega)\rangle = \langle b_{n_i}(r)|r|b_{n_j}(r)\rangle\langle Y_{l_i m_i}(\Omega)|\cos\Theta|Y_{l_j m_j}(\Omega)\rangle. \tag{2.218}$$

We can compute the angular part by numerical integration, but there are **relations for the spherical harmonies** that make everything easier. It turns out that $\cos\Theta Y_{lm}$ is again a linear combination of only two spherical harmonics

$$\cos\Theta Y_{lm}(\Theta,\varphi) = \left[\frac{(l+1+m)(l+1-m)}{(2l+1)(2l+3)}\right]^{\frac{1}{2}} Y_{l+1,m} + \left[\frac{(l+m)(l-m)}{(2l+1)(2l-1)}\right]^{\frac{1}{2}} Y_{l-1,m}. \tag{2.219}$$

Here we can see the dipole selection rule. The Y_{lm} can change the azimuthal quantum number only by one, the m-quantum number remains unchanged. This is good for our computation because we will get a very sparse matrix. The elements of the angular part of the sparse matrix are given by the following equations:

$$\langle Y_{l-1,m}|\cos\Theta|Y_{l,m}\rangle = \left[\frac{(l+1+m)(l+1-m)}{(2l+1)(2l+3)}\right]^{\frac{1}{2}} \underbrace{\langle Y_{l-1,m}|Y_{l+1,m}\rangle}_{0} + \left[\frac{(l+m)(l-m)}{(2l+1)(2l-1)}\right]^{\frac{1}{2}} \underbrace{\langle Y_{l-1,m}|Y_{l-1,m}\rangle}_{1} \tag{2.220}$$

and thus the angular part of the matrix for the dipole interaction reads:

$$(\widehat{D}^{(\Omega)})_{lm,l'm'} = \langle Y_{lm}|\cos\theta|Y_{l'm'}\rangle = \delta_{mm'}[\delta_{l+1,l'} + \delta_{l,l'+1}]d_{lm} \tag{2.221}$$

with

$$d_{lm} = \frac{(l+m)(l-m)}{(2l+1)(2l-1)}. \tag{2.222}$$

That is why we get a matrix with respect to l and l':

$$D^{(\Omega)} = \begin{pmatrix} 0 & d_0 & 0 & 0 & \dots & 0 \\ d_0^* & 0 & d_1 & 0 & \dots & 0 \\ 0 & d_1^* & 0 & d_2 & \dots & \vdots \\ \vdots & & \ddots & \ddots & \ddots & 0 \\ 0 & & \cdots & \ddots & 0 & \ddots \\ 0 & & \cdots & & \ddots & 0 \end{pmatrix}. \tag{2.223}$$

Note: Fill in the lower diagonal using hermiticity.

2.9.4 Soft skills: input and output

For studying the behavior of a physical system, for checking numerical stability and convergence, or for debugging, one typically runs the same program with many different inputs. It is advisable to keep track of the results of the many different runs.

Here there is one very practical way of doing this:

- Get the input parameters from a file, say "foo.inp".
- For repeated runs, edit that file, i.e. keep it open in an editor.
- For running, save the file and run the program.
- For each run, the program creates a new subdirectory, say "foo/00xx/", where the run-number "00xx" is incremented automatically.
- The current input file is copied into "foo/00xx/inp", all output is written to a single or several files "foo/00xx/outfile1", "foo/00xx/timing", etc.etc.
- In addition, the subdirectory may contain information about which code version was run. If you are connected to svn, you may just redirect the output of "svn info" and "svn diff" to a file, say, "foo/00xx/version_info"

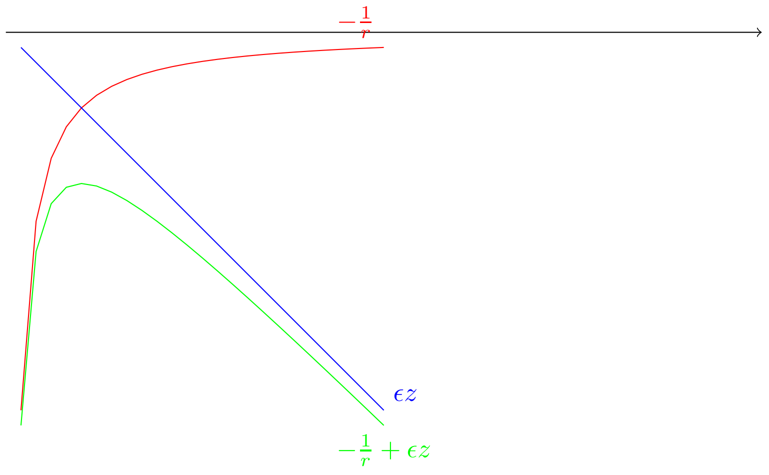

Figure 2.5: The green line shows Coulomb potential + dipole interaction energy. In particular the bigger is x, the lowest the energy. At large x, arbitrarily low potential energies and therefore also arbitrarily low total energies can be realized: the system atom+static dipole field has not ground state energy.

Linear and quadratic Stark effect

A brief reminder about the Stark effect: In presence of an external electric field, the spectral lines of atoms and molecules shift and split. The *quadratic Stark effect* appears in second order perturbation theory as

$$E(\mathcal{E}) = E_0 - \frac{\alpha}{2}\mathcal{E}^2. \tag{2.224}$$

For the ground state, α always positive and the ground state energy is *lowered* by the external static field.

The *linear Stark effect* of the hydrogen atom is a case of a degenerate perturbation theory. The external field lifts the degeneracy of the angular momentum states. In the simplest case of the degenerate $n = 2$, $l = 0, 1$ states one has

$$E_2 \pm \beta\mathcal{E} \tag{2.225}$$

where $E_2 = -1/8$ is the unperturbed energy. It is important to note that the Hamiltonian for the hydrogen atom in an external field does not have a lower bound and has a strictly continuous spectrum.

Problem 9.30: Trace the Stark-shifts and splittings of the $n = 1, 2$ states of hydrogen as a function of the electric field $\mathcal{E}$. For small fields, you can just use the existing "hfield.py". For larger fields, bound states become increasingly embedded in the continuum states. One strategy to find them is to increase the field in small steps and trace the evolution of the energies.

Identify the range of validity of the linear and quadratic regimes for the Stark effect.

Can we rigrously apply the minmax principle? What is the meaning of the discrete eigenvalues that we obtain from our numerical approximation?

2.9.5 More general potentials - integrations in 1 and more dimensions

Let us consider the 1-d Morse potential (see figure 2.6)

$$V(r) = D_e[1 - e^{\alpha(r-r_e)}]^2 \tag{2.226}$$

In order to compute the matrix element $\widehat{V}_{ij} = \int_0^\infty r^2 V(r)h_i(r)h_j(r)\,dr$ we have different possibilities:

- **Analytical integration:** is possible only sometimes. Further this option has the disadvantages that we must perform all the calculations again if a different potential is chosen and that the numerical evaluation of the analytical expression can be numerically unstable. It can be a good choice in higher dimensions.
- **Recurrence relations:** We will see more about that later, for the moment it is important to know that they require a some caution concerning numerical stability.
- **Quadratures:** This method is exact for polynomials times the respective weight function. It still gives a good approximation for smooth functions (functions with a convergent Taylor series). Some problems may arise when the function $V(r)$ we want to integrate is not smooth in

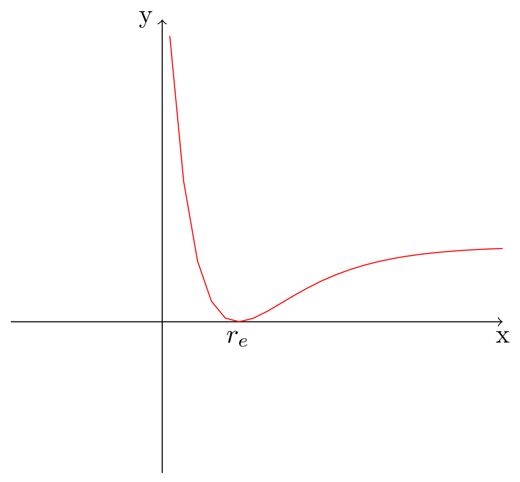

Figure 2.6: Morse Potential

certain regions: compare the problem in the section on orthogonal polynomials and quadratures. In order to apply the method efficiently in cases with non-smooth integrands one can create a specialized quadrature rule for a given positive definite weight function. For example, integrable singularities like $V(r) \sim \frac{1}{\sqrt{r}}$ can be computed by Gauss-Jacobi quadrature. The integrals $\int_0^\infty \frac{dr}{\sqrt{r}} P_n(r)$ exist and orthogonal (Jacobi) polynomials can be constructed. As a rule, Gauss quadratures, when applicable, are very efficient.

- **Adaptive integration:** by the recursive algorithm below.
- **Monte Carlo integration:** Of great importance in some areas of physics. When little is known about the integrand a good choice. In high dimensions, often the only viable method. Good for obtaining low accuracy estimates, slow convergence $\sim N^{-1/2}$ in the number of function evaluations N. A seperate chapter will be devoted to such methods.

2.9.6 Recursive algorithms

Recursive integration in one dimension

We will now look at recursive integration in more details, also because it is a nice prototype for recursive algorithms.

The idea behind recursive integration is to take a quadrature rule on an interval $[a, b]$ and check the accuracy of our result. Let's suppose we choose to work with a Gauss-Legendre quadrature; we will have:

$$f(x) = \sum_{i=0}^{2N-1} f^{(i)}(x_0) \frac{(x-x_0)^i}{i!} + O(\frac{f^{(2N)}(q)(b-a)^{2N}}{(2N)!}) \tag{2.227}$$

where $q \in [a, b]$. In the next step we will have to check the accuracy of our result and in order to do that we split the interval $[a, b]$ into two parts: $[a, \frac{a+b}{2}] \cup [\frac{a+b}{2}, b]$ and we compare the results that we obtain on the full interval with those that we obtain on the split intervals. Then we repeat this process until we reach the desired accuracy. If the functions we work with are smooth enough, we will usually manage to keep the error under control. Especially we will have an error which shrinks extremely fast with the interval size, and that is the reason why we halt the interval every time.

To sum up, the general strategy behind recursive integration is given by the following scheme:

- Get some quadrature rule with accuracy $\mathcal{O}((a-b)^p)$ on interval $[a,b]$.
- Check accuracy of the integral by comparing two integrals on split interval.
- If desired accuracy reached return result
- Iterate on sub-intervals: $[a,(a+b)/2]$ and $[(a+b)/2,b]$.

Failure of relative accuracy check

The natural stopping criterion for recursive integration is the comparison of the relative error with the accuracy we want to work with. However there are certain pathological situations in which one must take a certain caution, since this stopping criterion is never fulfilled and a typical example is given by the case of cancellation error.

Example: Suppose we want to calculate the following integral:

$$\int_0^{2\pi} \sin(x)\,dx. \tag{2.228}$$

We know that the analytical result is zero, but a computer will give ϵ_3 as an answer. If we split the interval of integration in two parts we will have:

$$\int_0^{\pi} sin(x)\,dx = 2 \simeq 2+\epsilon_1 \qquad \int_{\pi}^{2\pi} sin(x)\,dx = -2 \simeq -2+\epsilon_2. \tag{2.229}$$

The relative error is $\frac{\epsilon_1+\epsilon_2}{\epsilon_3} \simeq 1$ if ϵ_i is machine precision. So in this case, the algorithm will not stop base on relative error only. We need an additional stopping criterion, namely the absolute accuracy: if the value of an integral changes by less than an some absolute error, we consider the calculation converged. For giving meaning to this, we need some outside information which absolut error we will consider as small.

Quadrature in higher dimensions

The general higher-dimensional integral over a rectangular volume can be defined by a recursion of 1-d integrations:

$$F_0 = \int_{V_0 \subset R^n} d^{(n)}x f(x_0,x_1,\ldots,x_{n-1}) = \int_{a_0}^{b_0} dx_0 \int_{a_1}^{b_1} dx_1 \ldots \int_{a_{n-1}}^{b_{n-1}} dx_{n-1} f(x_0,x_1,\ldots,x_{n-1}) \tag{2.230}$$

$$= \int_{a_0}^{b_0} dx_0 F_1(x_0;a_1,b_1,\ldots,a_{n-1},b_{n-1}) \tag{2.231}$$

where

$$F_1(x_0; a_1, b_1, \ldots, a_{n-1}, b_{n-1}) = \int_{V_1 \subset R^{n-1}} dx_1 \ldots dx_{n-1} df(x_0; x_1, \ldots, x_{n-1}) \tag{2.232}$$

The general recursion is defined as

$$F_i(x_0 \ldots x_{i-1}; a_i, b_i, \ldots, a_{n-1}, b_{n-1}) = \int_{a_i}^{b_i} dx_i F_{i+1}(x_0 \ldots x_{i-1}; x_i; a_{i+1}, b_{i+1}, \ldots, a_{n-1}, b_{n-1}) \tag{2.233}$$

with the truncation point

$$F_n(x_0, x_1, \ldots, x_{n-1}) = f(x_0, x_1, \ldots, x_{n-1}) \tag{2.234}$$

Can be easily extended to general volumes.

Problem 9.31: Recursive algorithms: write a n-dimensional recursive integration algorithm `recursive_integral.py` by extending `recursive_integral_1d.py`. Use `integral_nd.py` as the basic integrator.

Hint: Describe the n-dimensional volume as in `integral_nd.py` by the interval boundaries in each dimension. The only difficulty is to systematically split this volume into 2^n subvolumes. An elegant way of doing this is by using the binary representation of $m \in 0, 1, \ldots, 2^n - 1$ to indicate where the lower corners of each sub-volume are located, e.g. for $n = 2$: m=0=(00) - lower left, m=1=(01) - lower right, m=2=(10) - upper left, m=3=(11) - upper right. One way of obtaining all bit values of an integer $M < 2^n$ is

```
m=M
for p in range(n):
    bit[p] = m%2
    m=m/2
```

2.9.7 1d "Helium" atom

Problem 9.32: The "1-d Helium" model has the following Hamiltonian

$$H^{(2)}(x_1, x_2) = H^{(1)}(x_1) + H^{(1)}(x_2) + \frac{1}{\sqrt{(x_1 - x_2)^2 + 0.5}} \tag{2.235}$$

with the single particle Hamiltonian

$$H^{(1)}(x) = -\frac{1}{2}\partial_x^2 - \frac{2}{\sqrt{x^2 + 0.5}} \tag{2.236}$$

- Compute the ground and first few excited states of $H^{(1)}$ in a high order finite element basis. Make sure the result is accurate. (Which remarkable number do you obtain for the ground state energy?) For the integrals, use high order quadratures and make sure they are accurate.

- Compute ground and first few excited states of $H^{(2)}$. For the integrals, use the (new) function "interaction" in "basisfunction.py". You can also use the current "potential.py", which includes "cou1d(Z,a)": $-Z/(x^2+a)$.
- Verify your code using harmonic oscillator potentials instead of the ''1-d Coulomb" potentials.
- Get an estimate of the accuracy of your calculation by looking at the convergence.
- Plot the particle density $|\Psi(x_1, x_2)|^2$ of ground and first few excited states.

Chapter 3

2-particle systems

3.1 Exponential growth of problem size

The size of the problem grows with increasing (spatial) dimensions. In an one dimensional problem we have a wave function $\Psi_{\vec{a}}(x)$ where the a_i's are the expansion coefficients. The number of expansion coefficients can be associated with poins of a grid (e.g. the quadrature points of a DVR method). If we now suppose that we have a_i, $i = 0, 1, ..., 9$ (i.e. discretized with ten discretization coefficients) $\widehat{H}$ is only a 10×10 matrix, which, in for a one-dimensional problem, can give reasonable results.

If we have a problem that includes a single particle in *three dimensions* (e.g. hydrogen atom) we have a discretization with ten points on each coordinate (e.g. x, y, z). That gives us $10^3 = 1000$ grid points and thus 1000 a_i's. The discretized operators will be 1000×1000 matrices.

If we now have a two particles system in three dimensions (e.g. helium) we get 10^6 points and $\widehat{H}$ is a $10^6 \times 10^6$ m matrix. If full, its storage alone requires $10^{12} \times 8$ bytes, which is around ten terabytes. The operations count is correspondingly high. Even with the rather poor discretization of each coordinat, doing such a calculation is nearly ruled out. The exponantial scaling N^d for d dimensions and N grid points per dimension is known in numerics as **the curse of dimensions**: very clearly, a direct approach this must fail quickly. The curse of dimensions is nothing that is specific to numerics. It simply reflects the enormous amount of information that can be contained in a multi-dimensional system. We find it in all few- and many-body systems. It is fair to say that a singificant part of physics is about handling this complexity: solid-state theory, statistical physics, quantum theory in general, quantum chemistry, etc.

We have to be smart solving our problem. One general way of being smart is to reduce the dimensionality of the problem by exploiting symmetries (rotational, translational, . . .). An example is the hydrogen atom (without field) mentioned above: the separation of angular symmetry reduces the originally 3-dimensional problem to a one-dimensional one. Also using roational symmetry, the helium atom can be reduced from a six dimensional to a three dimensional problem.

3.2 2 interacting particles in a harmonic potential

As an example where dimensions can be reduced we will discuss two interacting particles in a harmonic potential. The Hamiltonian of the problem can be written as:

$$H^{(2)}(\vec{r}_1,\vec{r}_2) = H^{(1)}(\vec{r}_1) \otimes \mathbf{1} + \mathbf{1} \otimes H^{(1)}(\vec{r}_2) + V_{ee}(\vec{r}_1,\vec{r}_2), \tag{3.1}$$

where

$$H^{(1)} := -\frac{1}{2}\Delta + \frac{1}{2}r^2, \tag{3.2}$$

where Δ is the Laplace operator. The interaction does NOT have tensor product form and thus does not factorize (i.e. cannot be written as $\sum_{i=1}^{M} U_i(\vec{r}_1)W_i(\vec{r}_2)$), but as an "interaction" it has some extra properties, namely that it only depends on the distance between the particles:

$$V_{ee}(\vec{r}_1,\vec{r}_2) = V(|\vec{r}_1 - \vec{r}_2|). \tag{3.3}$$

This can be exploited by making a smart choice for the coordinates:

$$\vec{r}_\pm = \sqrt{\frac{1}{2}}(\vec{r}_1 \pm \vec{r}_2). \tag{3.4}$$

The change of coordinates gives us a tensor product form of the potential and a complete factorization of the six dimensional problem into two three dimensional problems which are both roatitionally symmetric:

$$H^{(2)}(\vec{r}_+,\vec{r}_-) = H^{(+)} \otimes 1 + \mathbf{1} \otimes H^{(-)}, \tag{3.5}$$

where

$$H^{(+)} = -\frac{1}{2}\Delta_+ \frac{1}{2}r_+^2 \tag{3.6}$$

and

$$H^{(-)} = -\frac{1}{2}\Delta_- + \frac{1}{2}r_-^2 + V_{ee}(\sqrt{2}r_-). \tag{3.7}$$

$H^{(+)}$ is trivially solvable and $H^{(-)}$ is easily numerically solvable. Thus we solved the problem by a chnage of coordinates. We see that it is important to choose a "good" coordinate system. But note that it is not always clear what a good coordinate system is.

Note: We can also couple to a field an keep the separability, i.e. $\vec{\mathcal{E}} \cdot (\vec{r}_1 + \vec{r}_2) = \sqrt{2}\vec{\mathcal{E}} \cdot \vec{r}_+$

3.3 Helium

Let us now go back to the helium atom. We assume a fixed nucleus and an infinite nuclear mass. Then the Hamiltonian has the same structure as the two interacting particles discussed above:

$$H^{(2)}(\vec{r}_1,\vec{r}_2) = H^{(1)} \otimes \mathbf{1} + \mathbf{1} \otimes H^{(1)} + V_{ee}(\vec{r}_1,\vec{r}_2) \tag{3.8}$$

where

$$H^{(1)} = -\frac{1}{2}\Delta - \frac{2}{r}. \tag{3.9}$$

The Coulomb repulsion of the electrons does NOT have tensor product form:

$$V_{ee}(\vec{r}_1, \vec{r}_2) = \frac{1}{|\vec{r}_1 - \vec{r}_2|}. \tag{3.10}$$

The transformation in an other coordinate system (as discussed above) does not work here (try it if you have time). So it is not possible to do a factorization. To calculate the eigenstates of helium we do **not** have a Hamiltonian $H^{(2)}$ which is just a product of three dimensional functions. It is a six dimensional object. In this case things get difficult because we cannot cheat (by reducing dimensions) anymore. But the choice of the coordinates does matter nevertheless. There is no "best" choice for helium (or any other system), but several criteria (with decreasing importance):

- **Suitability to the physical situation**: Bound states, doubly excited states or ionization processes, for example, can favour different coordinates. A poor choice here may exclude a practical solution, even if "all coordinates are equal". In a highly accurate **bound state** the electrons will get close to each other a lot of times. Therefor the **relative** motion is very important. The **coordinates** that we will choose here describe the **distance** between the particles with the coordinates $\vec{r}_1, \vec{r}_2, \vec{r}_3 = |\vec{r}_1 - \vec{r}_2|$ and the Euler angles α, β, γ. Thus we shift the problem to the question how the triangle shown in Figure 3.1 is set in space. If we want an

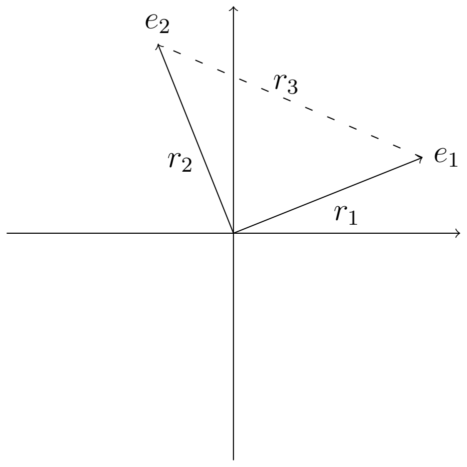

Figure 3.1: Bound states

  efficient representation of the ion and the free electron in an ionizing state of helium, a good choice are independent particle coordinates as the particles are **largely** independent.

- Implementation of symmetries (rotations, reflections, particle exchange,...) and sparsity structure.

- Numerical efficiency: How many discretization points are needed? (Closely related to previous point.)

- Numerical stability: Which basis functions can be used? The choice r_1, r_2 and r_3 is bad for a scattering problem, for example. As shown in Figure 3.2 a tiny difference in $r_3 - r_2$ leads to a large change in the angle θ in e_2. So we need a lot of functions in the area where θ changes. (Related to possible discretizations in a coordinate systems.)

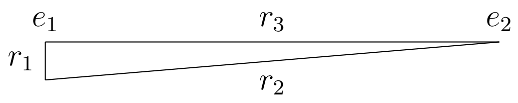

Figure 3.2: Electron scattering

- Possibility to (efficiently) compute observables (How hard are the integrals to do?). Observables are usually defined (and simple) in the coordinates $(\vec{r}_1, \vec{r}_2)$.

- Computational clarity and coding convenience.

independent particle (cartesian)	$\vec{r}_1, \vec{r}_2$	bound states	Gaussian bases
independent particle (polar)	r_i, θ_i, ϕ_i	bound and scattering	Slater basis, grid methods
interparticle	$r_1, r_2, \lvert\vec{r}_1 - \vec{r}_2\rvert$, Euler angles	bound states	"explicitly correlated" bases
Jacobi	$\vec{r}_1 \pm \vec{r}_2$	certain doubly excited states	
"perimetric"	u,v,w (see below)	bound	easy integrals, sparse matrices!
hyperspherical	$\sqrt{r_1^2 + r_2^2} + 5(!)$ "angles"	states near threshold	numerically difficult: singularities

3.3.1 Symmetries

For rotationally symmetric situations, the original 3-dimensional 2-electron system can be reduced to a set of independent 3-dimensional systems by exploiting angular symmetry. Further, minor, reductions in problems size can be achieved by using parity and exchange symmetries.

Angular symmetry

Suppose that the Hamiltonian commutes with $\vec{L}^2$ and $\vec{L}_z$. Then we can write any eigenstate Ψ_i as:

$$\Psi_i(\vec{r}_1, \vec{r}_2) = \sum_{n=0}^{L} G_n^{(LM+)}(\alpha, \beta, \gamma)\Phi_{i,n}(r_1, r_2, \vec{r}_1 \cdot \vec{r}_2), \tag{3.11}$$

where $\Phi_{i,n}(r_1, r_2, \vec{r}_1 \cdot \vec{r}_2)$ is rotationally invariant. $G_n^{(LM+)}$ are the angular factors with Euler angles α, β, γ. The angular factors fulfill the follwoing relations:

$$\mathbf{L}^2 G_n^{(LM+)} = L(L+1)G_n^{(LM+)}, \tag{3.12}$$
$$\mathbf{L}_z G_n^{(LM+)} = M G_n^{(LM+)}. \tag{3.13}$$

If we choose $L = 0$, we can set $G_{n=0}^{(00+)} \equiv 1$.

The space spanned by the $G_n^{(LM+)}$, $M = -L, \dots, L$, $n = 0, \dots, L$ is invariant under rotations. In addition, it gives an *irreducible representation* of the rotation group: any two functions in the space are connected by a rotation. It has no invariant subspaces, except the space itself and the empty space. This group theoretical argument shows that there are no further "tricks" to reduce the space based solely on rotational symmetry.

Parity

In **single particle** problems the parity P often is neglected, as in that case parity implied by angular momentum: $P = (-1)^L$. In **multiparticle** problems this does not hold. We can use parity to further reduce the size of our basis by fixing it to $P = +1$ or $P = -1$. In the representation of the rotations by the $G_n^{(L,M+)}$ this is already included: there are $L + 1$ factors with "natural parity" $P = (-1)^L$, and another set of L factors with "unnatural parity" $P = -(-1)^L$.

Note that any *irreducible* representation of spatial rotations *must* have definite parity, as rotations do not change parity. If there are two functions with different parity in the representation, they cannot be transformed into each other by rotatations and therefore the representation cannot be irreducible.

Exchange symmetry

We can decompose any function into eigenfunctions of exchange:

$$\Phi_{i,n}^{(\pm)}(r_1, r_2, \vec{r}_1 \cdot \vec{r}_2) = \pm\Phi_{i,n}^{(\pm)}(r_2, r_1, \vec{r}_2 \cdot \vec{r}_1) \tag{3.14}$$

Electrons are fermions which are anti-symmetric under exchange. So can we choose, both, $\Phi_{i,n}^{(+)}$ and $\Phi_{i,n}^{(-)}$? Yes, because of the **spin** of the particles. A spin state is anti-symmmetric $(-)$ for a spin singlet (i.e. $|S=0,S_z=0\rangle = 1/\sqrt{2}(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle))$ and symmetric (+) for a spin triplet (i.e. $|S=1,S_z=0,\pm 1\rangle = 1/\sqrt{2}(|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle), |\uparrow\uparrow\rangle, |\downarrow\downarrow\rangle)$. The total singlet and triplet wave functions including the spin factors

$$|S=0,S_z=0\rangle\Phi^{(+)}_{i,n}, \qquad |S=1,S_z\rangle\Phi^{(-)}_{i,n} \tag{3.15}$$

are anti-symmteric. As we neglect spin forces, we can solve separately for the spatial parts to obtaine the spatial parts of the singlet (+) and triplet (-) eigenstates.

3.3.2 Implementation

Kinetic energy operator for $L=0$

We express the operators in r_1, r_2 and r_3 (no derivation given):

$$\frac{1}{r_1}, \frac{1}{r_2}, \frac{1}{|\vec{r}_1 - \vec{r}_2|} = \frac{1}{r_3} \tag{3.16}$$

$$(\Delta_1 + \Delta_2)_{L=0} = \frac{4}{r_3}\partial_{r_3} + 2\partial^2_{r_3} + \sum_{i=1,2} \frac{2}{r_i}\partial_{r_i} + \partial^2_{r_i} + \frac{r_3^2 - (-1)^i(r_1^2 - r_2^2)}{r_i r_3}\partial_{r_i}\partial_{r_3} \tag{3.17}$$

Cusp conditions

On general mathematical grounds, we know the form of the solution near the Coulomb singularities:

$$\left(\frac{\partial \Phi}{\partial r_i}\right)_{r_i=0} = \mu_i q_i \Phi(r_i = 0), \tag{3.18}$$

where μ_i is the reduced mass and q_i the product of the charges for the particles at the end of the coordinate r_i. It is difficult to approximate this by smooth functions. That is why for high accuracy, the behavior easily representable by the basis or, for extremely accurate results, it may even be explicitly built into the basis. This is the most important reason for using interparticle coordinates.

"Explicitly correlated basis"

Egyl Hylleraas performed the first quantum mechanical calculations of the Helium bound state energies in the 1920'ies, using the follwing basis on the r_1, r_2, r_3-coordinates:

$$|i\rangle = |m_i, n_i, k_i\rangle = r_1^{m_i} r_2^{n_i} r_3^{k_i} e^{-\alpha r_1 - \beta r_2} \tag{3.19}$$

This is a choice, but is a good one: it turns out to work well. One motivation for this choice is that this basis simulates the exponential decay of the quantum mechanical wave function. We should now guess/optimize α, β: the optimization criteria is to get the lowest possible energy (lowest upper bound to the energy).

This basis is also called "explicitly correlated basis", which means that $r_3 = |r_2 - r_1|$ explicitly appears in the basis function. Further this basis can reproduce very well the cusp at $r_3 = 0$.

Note: The basis has one important problem: for larger basis sizes it develops near *linear dependencies* (compare our initial example using monomials). Can be controlled by computing in higher accuracy (16 byte $\sim$ 31 decimal digits), or by replacing the monomials r^m by Laguerre polynomials $L_m(2\alpha r)$

Note: This is NOT a product basis: the integration boundaries of the functions are interconnected. More abstractly, the Hilbert space defined by

$$H^{(3)} := \{\Phi(r_1, r_2, r_3) | \int_0^\infty dr_1 \int_0^\infty dr_2 \int_{|r_1-r_2|}^{r_1+r_2} dr_3 r_1 r_2 r_3 |\Phi(r_1, r_2, r_3)|^2 < \infty\} \tag{3.20}$$

is not just the product of 3 Hilbert spaces

$$H_i^{(1)} = \{\phi(r_i) | \int_0^\infty dr_i r_i |\phi(r_i)|^2 < \infty\}, \quad i = 1, 2, 3 : \tag{3.21}$$

$$H^{(3)} \neq H_1^{(1)} \otimes H_1^{(1)} \otimes H_1^{(1)} \tag{3.22}$$

As the scalar product on $H^{(3)}$ is not given by products of scalar products of the factor functions. Therefore, *in these coordinates*, there exists no product basis and the Hamiltonian cannot be written as (finite) sum of tensor products.

Integrations

All integrals will have the general form

$$J_{m,n,k} := \int_0^\infty dr_1 \int_0^\infty dr_2 \int_{|r_1-r_2|}^{r_1+r_2} dr_3 r_1^m r_2^n r_3^k e^{-2\alpha r_1 - 2\beta r_2} \tag{3.23}$$

The integration over r_3 gives polynomials in r_1, r_2 times $e^{-2\alpha r_1} e^{-2\beta r_2}$ and therefore this integral has a closed form: a sum of rational functions in the variables α and β. The problem is that this closed form has many positive and negative terms and its evaluation is numerically unstable due to cancellation errors.

The solution to this problem is given by recursive integration.The general recurrence scheme is:

$$J_{m,n,k} = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_{|r_1-r_2|}^{r_1+r_2} dr_3 r_1^m r_2^n r_3^k e^{-2\alpha r_1 - 2\beta r_2} \tag{3.24}$$

$$J_{m,n,k} = \frac{1}{2\alpha + 2\beta} \left(A_{m,n,k} + m J_{m-1,n,k} + n J_{m,n-1,k}\right) \tag{3.25}$$

and

$$\begin{aligned} A_{m,n,k} =& \frac{1}{2\alpha + 2\beta} [m A_{m-1,n,k} + n A_{m,n-1,k} + 2k A_{m,n,k-1} \\ & + 2(2\alpha)^{-(m+k+1)} \delta_{0n} (m+k)! + 2(2\beta)^{-(n+k+1)} \delta_{0m} (n+k)!] \end{aligned} \tag{3.26}$$

This recurrence scheme contains only positive terms, i.e. there are no cancellation erros, which are usually the main source of error.

Note: The integrations can also be done quite efficiently just numerically, as we have exact quadratures. This is the method of choice when we replace the monomials r_1^m by Laguerre polynomials $L_m(\alpha r_1)$

Example: Let's calculate some matrix elements:

$$\langle i|\frac{1}{r_1}|j\rangle = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_{|r_1-r_2|}^{r_1+r_2} dr_3 r_1 r_2 r_3 \frac{1}{r_1} e^{-2\alpha r_1 - 2\beta r_2} r_1^{m_i+m_j} r_2^{n_i+n_j} r_3^{k_i+k_j} \tag{3.27}$$

$$\langle i|\frac{1}{r_3}\partial_{r_3}|j\rangle = \int_0^\infty dr_1 \int_0^\infty dr_2 \int_{|r_1-r_2|}^{r_1+r_2} dr_3 r_1 r_2 r_3 e^{-\alpha r_1 - \beta r_2} r_1^{m_i} r_2^{n_i} r_3^{k_i} \left(\frac{1}{r_3}\partial_{r_3}\right) [r_1^{m_j} r_2^{n_j} r_3^{k_j} e^{-\alpha r_1 - \beta r_2}] \tag{3.28}$$

Implementation of exchange symmetry

We want to construct a basis function $|i_\pm\rangle$ with the following symmetry property:

$$P_x|i_\pm\rangle = \pm|i_\pm\rangle \tag{3.29}$$

where P_x is the exchange operator:

$$P_x|i\rangle = P_x\left[r_1^{m_i} r_2^{n_i} r_3^{k_i} e^{-\alpha r_1 - \beta r_2}\right] = \left[r_1^{n_i} r_2^{m_i} r_3^{k_i} e^{-\beta r_1 - \alpha r_2}\right] \tag{3.30}$$

If we define the projector on (anti-)symmetric function:

$$P_\pm = \frac{1}{2}[1 \pm P_x] \tag{3.31}$$

the basis:

$$|i_\pm\rangle := P_\pm|i\rangle = \frac{1}{2}(1 \pm P_x)|i\rangle \tag{3.32}$$

will have the symmetry property that we want.

Note: This works for any basis coordinate system.

Let's now compute some matrix element of this new basis. For the overlap matrix we will have:

$$\langle i_\pm|j_\pm\rangle = \langle i|P_\pm P_\pm|j\rangle = \langle i|P_\pm|j\rangle = \frac{1}{2}[\langle i|j\rangle \pm \langle i|P_x|j\rangle] \tag{3.33}$$

$$\langle i_-|j_+\rangle = 0 \text{ because } P_+P_- = 0 \tag{3.34}$$

For the Hamiltonian:

$$\langle i_\pm|H|j_\pm\rangle = \langle i|P_\pm H P_\pm|j\rangle = \langle i|H P_\pm P_\pm|j\rangle = \langle i|H P_\pm|j\rangle == \frac{1}{2}[\langle i|H|j\rangle \pm \langle i|H P_x|j\rangle] \tag{3.35}$$

where we used the fact that $[H, P_x] = [H, P_\pm] = 0$

Problem 3.33: Does the hydrogen atom form a negative ion? It does, i.e. there is a bound state consisting of a proton and two electrons. Proof this by using `he_inter.py` to determine an upper bound for the system's ground state energy. By studying the convergence of the upper bound, try to give an accuracy estimate for your result. Can you find more than one bound state?
Hint: For finding the ionic bound state, it is important to choose good exponents in the basis. On physics grounds, try to guess values for α and β: one electron may be considered bound like in the neutral H atom, implying an exponent 1. Clearly, the extra electron will be very loosely bound, i.e. the wave function should admit at least one electron to be very far. Translate this into a value for one exponent and then manually search to improve the value.

Problem 3.34: Setting up the matrix in `he_inter.py` is slow. Use the timer routines `mytimer.py` to locate the timing problem. Suggest ways how to speed up the code. Implement your solution.

Problem 3.35: When all masses must be considered finite, after splitting center-of-mass motion, the three-body Coulomb Hamiltonian is

$$H = -\frac{1}{2\mu_1}\Delta_{13} - \frac{1}{2\mu_2}\Delta_{23} - \frac{1}{m_3}\vec{\nabla}_{13}\cdot\vec{\nabla}_{23} + \frac{Z_1 Z_3}{r_{13}} + \frac{Z_1 Z_2}{r_{12}} + \frac{Z_2 Z_3}{r_{23}}, \tag{3.36}$$

where $\vec{r}_{ij} = \vec{r}_i - \vec{r}_j$ and $\Delta_{ij}, \vec{\nabla}_{ij}$ are Laplacian and Nabla w.r.t. to the corresponding $\vec{r}_{ij}$. Z_i is the charge of the i'th particle. $\mu_i^{-1} = m_i^{-1} + m_3^{-1}$ are the standard reduced masses. The additional mass-polarization $\vec{\nabla}_{13}\cdot\vec{\nabla}_{23}$ term must be taken into account, when m_3 is no longer infinite relative to the masses m_1, m_2.

Derive an expression for the mass-polarization term in terms of the inter-particle distances r_{ij} and the partial derivates $\partial_{r_{ij}}$ if all angular dependencies can be neglected.
Hint: Calculate $\vec{\nabla}_{13}\cdot\vec{\nabla}_{13} f(r_{13}) g(r_{23}) h(r_{12})$ and use $\vec{r}_{13}\cdot\vec{r}_{23} = r_{13}^2 + r_{23}^2 - r_{12}^2$

Implement the additional terms into `he_inter.py` and compute the effect of the finite mass of the Helium-nucleus (=α-particle) on the ground state energy: $m_\alpha \approx 7294 m_e$.

What happens to the ground state energy as the mass ratio m_α/m_e decreases? Assess the convergence of your results. For which mass ratios is the system bound at all?

Solution: 3.35: For convenience, we change notation $\vec{r}_{i3} \to \vec{r}_i, i = 1,2$, $\vec{r}_{12} = \vec{r}_2 - \vec{r}_1 =: \vec{r}_3$. We have

$$\begin{aligned}
\vec{\nabla}_i r_i &= \frac{\vec{r}_i}{r_i}, \quad i = 1,2 && (3.37)\\
\vec{\nabla}_1 |r_3| &= \vec{\nabla}_1[(\vec{r}_1 - \vec{r}_2)^2]^{1/2} = \frac{(\vec{r}_1 - \vec{r}_2)}{r_3} = -\frac{\vec{r}_3}{r_3} && (3.38)\\
\vec{\nabla}_2 |r_3| &= \frac{(\vec{r}_2 - \vec{r}_1)}{r_3} = \frac{\vec{r}_3}{r_3} && (3.39)
\end{aligned}$$

Now

$$\begin{aligned}
\vec{\nabla}_2 f(r_1)g(r_2)h(r_3) &= f(r_1)\frac{\vec{r}_2}{r_2}\partial_2 g(r_2)h(r_3) + f(r_1)g(r_2)\frac{\vec{r}_3}{r_3}\partial_3 h(r_3) && (3.40)\\
&= \left[\frac{\vec{r}_2}{r_2}\partial_2 + \frac{\vec{r}_3}{r_3}\partial_3\right] f(r_1)g(r_2)h(r_3) && (3.41)\\
\vec{\nabla}_1 f(r_1)g(r_2)h(r_3) &= \left[\frac{\vec{r}_1}{r_1}\partial_1 - \frac{\vec{r}_3}{r_3}\partial_3\right] f(r_1)g(r_2)h(r_3) && (3.42)
\end{aligned}$$

Using this, one can compute the action of $\vec{\nabla}_3 \cdot \vec{\nabla}_1$.

$$\begin{aligned}
&\vec{\nabla}_1 \cdot \vec{\nabla}_2 = && (3.43)\\
&= \vec{\nabla}_1 \cdot \left(\frac{\vec{r}_2}{r_2}\partial_2 + \frac{\vec{r}_3}{r_3}\partial_3\right) && (3.44)\\
&= \frac{\vec{r}_2}{r_2}\vec{\nabla}_1\partial_2 + \left(\vec{\nabla}_1 \cdot \vec{r}_3\right)\frac{1}{r_3}\partial_3 + \vec{r}_3 \cdot \left(\vec{\nabla}_1 \frac{1}{r_3}\right)\partial_3 + \vec{r}_3 \cdot \frac{1}{r_3}\vec{\nabla}_1\partial_3 && (3.45)\\
&= \frac{\vec{r}_2}{r_2}\left(\frac{\vec{r}_1}{r_1}\partial_1 - \frac{\vec{r}_3}{r_3}\partial_3\right)\partial_2 - 3\frac{1}{r_3}\partial_3 + \vec{r}_3 \cdot \frac{\vec{r}_3}{r_3}\frac{1}{r_3^2}\partial_3 + \frac{\vec{r}_3}{r_3}\left(\frac{\vec{r}_1}{r_1}\partial_1 - \frac{\vec{r}_3}{r_3}\partial_3\right)\partial_3 && (3.46)\\
&= \frac{\vec{r}_2\vec{r}_1}{r_2 r_1}\partial_2\partial_1 - \frac{\vec{r}_2\vec{r}_3}{r_2 r_3}\partial_3\partial_2 - 2\frac{1}{r_3}\partial_3 + \frac{1}{r_3}\partial_3 + \frac{\vec{r}_3\vec{r}_1}{r_3 r_1}\partial_1\partial_3 - \partial_3\partial_3 && (3.47)\\
& && (3.48)
\end{aligned}$$

Note that the above expression is invariant under exchange $1 \leftrightarrow 2$, as $\vec{r}_3 = \vec{r}_2 - \vec{r}_1$ changes sign. Now we use

$$\begin{aligned}
\vec{r}_1 \cdot \vec{r}_2 &= \frac{1}{2}[r_1^2 + r_2^2 - (\vec{r}_2 - \vec{r}_1)^2] = \frac{1}{2}[r_1^2 + r_2^2 - r_3^2] && (3.49)\\
\vec{r}_1 \cdot \vec{r}_3 &= \vec{r}_1 \cdot (\vec{r}_2 - \vec{r}_1) = \vec{r}_1 \cdot \vec{r}_2 - r_1^2 = \frac{1}{2}[r_2^2 - r_1^2 - r_3^2] && (3.50)\\
\vec{r}_2 \cdot \vec{r}_3 &= \vec{r}_2 \cdot (\vec{r}_2 - \vec{r}_1) = -\frac{1}{2}[r_1^2 - r_2^2 - r_3^2] && (3.51)\\
& && (3.52)
\end{aligned}$$

and procede to

$$= \frac{r_1^2 + r_2^2 - r_3^2}{2r_2 r_1}\partial_2\partial_1 + \frac{r_1^2 - r_2^2 - r_3^2}{2r_2 r_3}\partial_3\partial_2 - \frac{2}{r_3}\partial_3 + \frac{r_2^2 - r_1^2 - r_3^2}{2r_1 r_3}\partial_1\partial_3 - \partial_3\partial_3. \qquad (3.53)$$

We have obtained a total of 11 terms for the mass-polarization. Note that all of these terms except for the $\partial_1\partial_2$-term already appear in the (symmetric) Laplacian. The implementation only requires changing a few factors and adding the $\partial_1\partial_2$-term.

Problem 3.36: To assess the convergence of the results, try (1) to optimize the coefficients

α, β in the exponentials. At fixed powers, systematically try different α, β, starting from the default values 1,1. In general, the best (=lowest energy) exponents will depend on the mass ratios and also at the specific state (ground, excited) that you are looking at. Try to optimize for $He : m_3 = \infty$ and the muonic molecular ion $pp\mu$ (consisting of two protons and a muon ($m_\mu \approx 207 m_e$, charge=-1). What do the "optimal values" tell you about the geometry of the two systems?

Problem 3.37: To decide, whether a system is bound, you need to compare its energy to the energy of a system combining two constitutents with two different charges, in case of He the He^+ ion, in case of $pp\mu$ the muonic hydrogen atom $p\mu$. Remember that the ground state energy of any two-body Coulomb system is $-0.5(Z_1 Z_2)^2(1/m_1 + 1/m_2)$. You should find that $pp\mu$ is bound. Also, the positronium ion $e^+e^-e^-$ is bound. Try to prove it using this code.

Problem 3.38: To obtain higher angular momentum states $L > 0$, we need to allow angular dependence in the basis. The recipe is simple: just multiply the rotationally invariant basis functions that we used so far by functions that have the correct transformation behavior under rotations. As may be known, vectors (dipoles) rotate like $L = 1$ objects. In fact, we have two vectors available $\vec{r}_1$ and $\vec{r}_2$. We can define a new basis, twice as large as the $L = 0$ basis, by

$$|m_i, n_i, n_i\rangle = |i\rangle \to \{\vec{r}_1|i\rangle, \vec{r}_2|i\rangle\}. \tag{3.54}$$

Return to the case of $m_3 = \infty$ to compute matrix elements for the kinetic energy with the $L = 1$-basis. Use the same approach as for the mass-polarization terms to get the explict expressions in terms of partial derivatives and powers. In addition you now have angles $\cos\theta_1$ and $\cos\theta_2$ and angular integrations. These are all easy to perform. Compute the lowest $L = 1$ energy (for crosschecks: -2.12384 in atomic units).

Solution: 3.38: We remember

$$\Delta\vec{r} = \vec{r}\Delta + [\Delta, \vec{r}] = \vec{r}\Delta + \vec{\nabla}\cdot[\vec{\nabla}, \vec{r}] + [\vec{\nabla}, \vec{r}]\vec{\nabla} = \vec{r}\Delta + 2\vec{\nabla} \tag{3.55}$$

to find

$$\langle \vec{r}_1 b|\Delta_1 \vec{r}_1 c\rangle \quad = \quad \langle b|r_1^2\Delta_1 + 2\vec{r}_1\cdot\vec{\nabla}_1|c\rangle \tag{3.56}$$

We use our previous results to get for functions $c = f(r_1)g(r_2)h(r_3)$:

$$\begin{aligned} \vec{r}_1 \cdot \vec{\nabla}_1 &= r_1\partial_1 - \frac{\vec{r}_1\vec{r}_3}{r_3}\partial_3 && (3.57)\\ \vec{r}_2 \cdot \vec{\nabla}_1 &= \frac{\vec{r}_2\cdot\vec{r}_1}{r_1}\partial_1 - \frac{\vec{r}_1\vec{r}_3}{r_3}\partial_3 && (3.58)\end{aligned}$$

and the corresponding $1 \leftrightarrow 2$ terms. In more general notation, the kinetic energy matrix elements become

$$\langle \vec{r}_c a|\Delta_{c''}\vec{r}_{c'} b\rangle = \langle a|\vec{r}_c\cdot\vec{r}_c' \left[\Delta_{c'} + \frac{2}{r_c'}\partial_{c'}\right] + (-1)^{c'}\frac{2\vec{r}_c\vec{r}_3}{r_3}\partial_3|b\rangle \tag{3.59}$$

It is clear that we can compute all matrix elements in this way. One possibility of implementation is to set up the matrices in blocks

$$\widehat{H}^{(cc')}_{ij} = \langle \vec{r}_c b_i | H \vec{r}_{c'} b_j \rangle, \quad c, c' = 1, 2 \tag{3.60}$$

and list the required re-definition of powers and exponents. Specifically, we have the blocks

$$\begin{aligned}
\widehat{H}^{(11)}_{ij} &= \langle b_i | \vec{r}_1 \cdot \vec{r}_1 \widehat{H}_{L=0} - r_1 \partial_1 + \frac{\vec{r}_1 \vec{r}_3}{r_3} \partial_3 | b_j \rangle && (3.61)\\
\widehat{H}^{(22)}_{ij} &= \langle b_i | \vec{r}_2 \cdot \vec{r}_2 \widehat{H}_{L=0} - r_2 \partial_2 - \frac{\vec{r}_2 \vec{r}_3}{r_3} \partial_3 | b_j \rangle && (3.62)\\
\widehat{H}^{(12)}_{ij} &= \langle b_i | \vec{r}_1 \cdot \vec{r}_2 \widehat{H}_{L=0} - \frac{\vec{r}_1 \vec{r}_2}{r_2} \partial_2 - \frac{\vec{r}_1 \vec{r}_3}{r_3} \partial_3 | b_j \rangle && (3.63)\\
\widehat{H}^{(21)}_{ij} &= \widehat{H}^{(12)}_{ij} && (3.64)
\end{aligned}$$

and for the overlap matrix

$$\widehat{S}^{(cc')}_{ij} = \langle b_i | \vec{r}_c \vec{r}_{c'} | b_j \rangle \tag{3.65}$$

Implementation is most convenient by first defining a few functions that allow multiplication and addition of the "InterOper" objects, then write the operators as given above (check the example code for how to do this). You may verify your own solutions against this.

When setting up the matrices, one has to pay attention to the fact that $1 \leftrightarrow 2$-exchange affects also the vectors $\vec{r}_1 \leftrightarrow \vec{r}_2$. Present implementation of exchange cannot handle this (can be modified). The solution is, *not* to impose exchange symmetry, which results in returning both, singlet and triplet states in a single calculation (but diagonalizing an almost twice-as-large matrix).

Another pitfall when moving from the old code is that the "matrix(...)"-method was assuming symmetric matrices always. However, this is not the case for the off-diagonal blocks, which are related by

$$\widehat{H}^{(12)} = [\widehat{H}^{(21)}]^T, \tag{3.66}$$

but $\widehat{H}^{(12)} \neq [\widehat{H}^{(12)}]^T$.

A good way to implement exchange is to work with angular factors that have well-defined exchange symmetry, i.e. $\vec{r}_\pm := \vec{r}_2 \pm \vec{r}_1$. There will be no matrix elements between the $\vec{r}_\pm$ blocks and we compute a smaller matrix to begin with ($s = \pm 1$)

$$\begin{aligned}
\widehat{H}^{(s)}_{ij} &= \langle b_i | \vec{r}_s \cdot \vec{r}_s \widehat{H}_{L=0} - (\vec{r}_2 + s\vec{r}_1)(\vec{\nabla}_2 + s\vec{\nabla}_1) | b_j \rangle && (3.67)\\
&= \langle b_i | \vec{r}_s \cdot \vec{r}_s \widehat{H}_{L=0} - \vec{r}_2 \vec{\nabla}_2 - \vec{r}_1 \vec{\nabla}_1 - s\vec{r}_2 \vec{\nabla}_1 - s\vec{r}_1 \vec{\nabla}_2) | b_j \rangle && (3.68)\\
&= \langle b_i | \vec{r}_s \cdot \vec{r}_s \widehat{H}_{L=0} - r_2 \partial_2 - r_1 \partial_1 - s\vec{r}_1 \vec{r}_2 (\frac{1}{r_1} \partial_1 + \frac{1}{r_2} \partial_2) - (1-s) r_3 \partial_3 | b_j \rangle && (3.69)\\
& && (3.70)
\end{aligned}$$

Combine the L=0 and L=1 bases. Include the dipole coupling to the field into the Hamiltonian

$$H = H_0 + \mathcal{E}(z_{13} + z_{12}), \tag{3.71}$$

where H_0 consists of blocks of the Hamiltonians for L=0 and L=1. Note that the dipole operaror is non-zero only for matrix elements between $L = 0$ and $L = 1$ states (dipole selection rules). For weak fields $\mathcal{E} \lesssim 0.1$, compute the energy shifts. Use complex scaling to estimate the decay-width of the states.

Hint: For complex scaling you multiply the individual terms in the Hamiltonian matrix by complex phases $\eta = \exp i\theta$ as follows

$$\widehat{H}_\eta = \eta^{-2}\widehat{T} + \eta^{-1}\widehat{V} + \eta\widehat{D} \tag{3.72}$$

Problem 3.39: Form an approximate picture of the various states you have obtained: compute expectation values of r_1, r_2, r_3, which tell you the approximate shape of the triangle formed by the particles (if there were anything like an exact position in quantum mechanics). Also calculate their square and then the variances $(\langle r_i^2\rangle - \langle r_i\rangle^2)^{1/2}$, which tells you to which extent one can speak of a well-defined distance at all. You should find that with decreasing binding energy, the particles are farther apart. In excited states, one constituent tends to wander away from the other two. Show this for the He ground and excited states, for $pp\mu$, and also for the $L = 1$ states, if you did obtain them.

Summary of the code

- The basis is generated as a list of singel basis funtions `InterBas`: this knows the powers and exponents of the basis function. On this level, the desired exchange symmetry needs to be taken into consideration: if the we will symmetrized our operator, only basis functions that are not related to a basis function already in the list can be added to the list. Otherwise, the basis would be linearly dependent after symmetrization (and the inverse of the overlap would not exist).

- An operator in interparticle coordinates `InterOper` is a set of separate terms `InterTerm`. A term consists of a factor, powers of r_i , and derivatives ∂_{r_i}. E.g.

$$-2\frac{1}{r_2}\partial_{r_1}^2 \ldots [-2., 0, -1, 0, 2, 0, 0] \tag{3.73}$$

  In operator setup, we can automatically supplement exchange symmetric terms. This reduces the chance for errors by typos.

- A table of integrals is set up an extended as needed. We start from a zero-size table.

- Setting up the matrix is by (1) first applying the `InterTerm` to the basis, which results in a list of basis functions and then adding all the integrals corresponding to the basis functions from the list.

3.3.3 Perimetric coordinates

We can introduce the so-called perimetric coordinates:

$$\begin{aligned} u &= -r_1 + r_2 + r_3 && (3.74) \\ v &= r_1 - r_2 + r_3 && (3.75) \\ w &= r_1 + r_2 - r_3. && (3.76) \end{aligned}$$

In this new coordinates the whole Hamiltonian can be written by tensor products:

$$H^{(3)}(u, v, w) = H_u^{(1)} \otimes H_v^{(1)} \otimes H_w^{(1)}, \tag{3.77}$$

where $H_q^{(1)} = \{\Phi(q) | \int_0^\infty dq\, f(q)|\Phi(q)|^2 < \infty\}$. These coordinates are computationally great because we can use a product basis, the integration is a product of three one dimensional integrations (i.e. $\int_0^\infty dr_1 \int_0^\infty dr_2 \int_{|r_1-r_2|}^{r_1+r_2} dr_3\, r_1 r_2 r_3 \rightarrow \int_0^\infty du \int_0^\infty dv \int_0^\infty dw\, f(u, v, w)$) and we get sparse matrices. **BUT** unfortunately the cusps cannot be modelled accurately in perimetric coordinates. Let us, for example, look at the following situation:

$$r_3 = 0 \Leftrightarrow u + v = 0. \tag{3.78}$$

As shown in figure 3.3 it is difficult to locate the cusp in perimetric coordinates. Furthermore it is difficult to model this area with a smooth function and thus to describe the asymptotic behaviour. We get a **slow convergence** here.

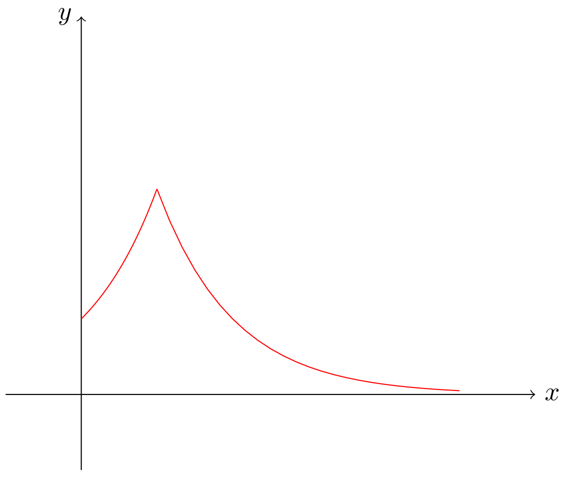

Figure 3.3: This kind of cusp is particularly difficult to model with a smooth function.

3.3.4 He in $\vec{r}_1, \vec{r}_2$ coordinates (independent particle coordinates)

Now we will discuss coordinates which can be used as a basis for pretty much everything. We will discuss it as a general procedure which is extendable to several particles.

Hamiltonian

In this coordinates the Hamiltonian of a helium atom is:

$$\mathbf{H}^{(2)}(\vec{r}_1,\vec{r}_2) = \mathbf{H}^{(1)}(\vec{r}_1)\otimes \mathbf{1} + \mathbf{1}\otimes \mathbf{H}^{(1)}(\vec{r}_2) + V(|\vec{r}_1-\vec{r}_2|) \tag{3.79}$$

with

$$\mathbf{H}^{(1)}(\vec{r}_i) = -\frac{1}{2}\frac{1}{r_i}\partial^2_{r_i} r_i \otimes \mathbf{1}^{(\theta_i,\varphi_i)} + \frac{1}{2}\frac{1}{r_i^2}\otimes (\mathbf{L}^{(1)})^2 - \frac{Z}{r_i}\otimes \mathbf{1}, \tag{3.80}$$

where we use polar coordinates for each particle (i.e. $(x_i, y_i, z_i)\to(\varphi_i,\theta_i,r_i)$). We will discuss the Coulomb potential later in detail. Now we concentrate on the other parts.

Basis

The two-particle space is the tensor product space of the one-particle spaces:

$$\mathcal{H}^{(2)}(\vec{r}_1,\vec{r}_2) = \mathcal{H}^{(1)}(\vec{r}_1)\otimes\mathcal{H}^{(1)}(\vec{r}_2), \tag{3.81}$$

where the one-partile space is $\mathcal{H}^{(1)}(\vec{r}) = \{\Phi(\vec{r})|\langle\Phi|\Phi\rangle < \infty\}$. A basis in this two-particle space is a tensor product of bases of one-particle spaces:

$$|i\rangle = \underbrace{|l_i^{(1)}, m_i^{(1)}, k_i^{(1)}\rangle}_{\text{first particle}} \otimes \underbrace{|l_i^{(2)}, m_i^{(2)}, k_i^{(2)}\rangle}_{\text{second particle}}. \tag{3.82}$$

We use the spherical harmonics basis:

$$|l,m,k\rangle = Y_{lm}(\theta,\varphi)h_k(r), \tag{3.83}$$

where $h_k(r)$ is some basis on the radial coordinates. So our choice is some product basis.

Angular momentum

We know that:

$$[\mathcal{H}^{(2)}, (\vec{L}^{(2)})^2] = [\mathcal{H}^{(2)}, L_z^{(2)}] = 0, \tag{3.84}$$

where $\vec{L}^{(2)} = \vec{L}^{(1)}\otimes\mathbf{1} + \mathbf{1}\otimes\vec{L}^{(1)}$ is the two particles angular momentum. We can now expand into the basis of the eigenfunctions of $\vec{L}$ and L_z:

$$|L,M,l_1,l_2,k,k'\rangle = |L,M,l_1,l_2\rangle\otimes h_k(r_1)h_{k'}(r_2). \tag{3.85}$$

This total angular momentum eigenbasis is NOT a product basis. It requires the following linear combinations of functions from the product basis (project onto a space with fixed total angular momentum L, M):

$$|L,M,l_1,l_2\rangle = \underbrace{\sum_{l_1,l_2,m_1,m_2}[|l_1,m_1\rangle\otimes|l_2,m_2\rangle][\langle l_1,m_1|\otimes\langle l_2,m_2|]}_{=\mathbf{1}}\,|L,M=m_1+m_2,l_1,l_2\rangle \tag{3.86}$$

$$=: \sum_{m_1,m_2}|l_1m_1\rangle\otimes|l_2m_2\rangle\underbrace{\langle l_1,m_1,l_2m_2|L,M,l_1,l_2\rangle}_{\text{Clebsch-Gordan}}. \tag{3.87}$$

We can write the projector onto the L, M space:

$$P_{LM} = \sum_{l_1,l_2} |L, M, l_1, l_2\rangle\langle L, M, l_1, l_2| \tag{3.88}$$

and construct the new basis as for the exchange operator (above):

$$|i\rangle_{LM} := P_{LM}|i\rangle. \tag{3.89}$$

Parity

If we think back to the coordinates r_1, r_2, r_3 we know that parity is in the factor G_{l0}. Here we only select $|L, M, l_1, l_2\rangle$ with:

- $l_1 + l_2 + L$ even: “Natural parity”

- $l_1 + l_2 + L$ odd: “Unnatural parity”.

For interparticle coordinates, the representation of the rotation group for a two-particle system by the functions $G_n^{LM\pm}$

Problem 3.40: One choice of the functions $G_n^{LM\pm}$ used in the inter-particle coordinate system is

$$G_n^{LM\pm} = |LMl_1l_2\rangle \tag{3.90}$$

with $l_1 = n$ and $l_2 = L - l_1$ or $l_2 = L - l_1 + 1$. Show that the two choices for l_2 correspond to the two different parities $(-1)^L$ and $(-1)^{L+1}$. Give explicit functional dependence of $|LMl_1l_2\rangle$ on the polar angles $\theta_1, \phi_1, \theta_2, \phi_2$ in terms of spherical harmonics and Clebsch-Gordan coefficients.

Exchange symmetry

The exchange symmetry is the same as above:

$$|L, M, l_1, l_2, k, k'\rangle_{\pm} = \frac{1}{2}\left[|L, M, l_1, l_2, k, k'\rangle \pm |L, M, l_2, l_1, k', k\rangle\right] = P_{\pm}|L, M, l_1, l_2, k, k'\rangle. \tag{3.91}$$

Interaction integrals

It is not easy to do the interaction (e.g. $1/|\vec{r}_1 - \vec{r}_2|$) integrals. A convenient way is using the multipole expansion:

$$\frac{1}{|\vec{r}_1 - \vec{r}_2|} = \sum_{l=0}^{\infty} \frac{4\pi}{2l+1} \sum_{m=-l}^{l} Y_{lm}^*(\theta_1, \varphi_1) Y_{lm}(\theta_2, \varphi_2) \frac{r_<^l}{r_>^{l+1}} \tag{3.92}$$

with

$$r_< := \min(r_1, r_2), \qquad r_> := \max(r_1, r_2). \tag{3.93}$$

This almost looks like an (infinite) sum of tensor products. The only thing that causes trouble is $r_<^l/r_>^{l+1}$. Acutally, for $h_k(r_1)$ and $h_{k'}(r_2)$ whose support do not overlap, it still has the form of a (trensor) product, say $r_1 \otimes \frac{1}{r_2}$ if $r_1 < r_2$ If, for example, the supports of h_k and $h_{k'}$ do not overlap, the integrals can be brought to tensor product form.

Problem 3.41: Multipole integrals(1): Write a code to compute the radial integrals arising in the multipole expansion

$$J_l(i,j,m,n) = \int dr_1 r_1^2 \int dr_2 r_2^2 f_i(r_1) g_j(r_2) \frac{r_<^l}{r_>^{l+1}} f_m(r_1) g_n(r_2) \tag{3.94}$$

for two sets of finite element functions $f_i, i \in 0,1,\dots,I-1$ and $g_j, j \in 0,1,\dots,J-1$ that have no overlap. Try to exploit symmetries to reduce the number of calculations. Think about possibilities to test the correctness of your results.
Hint: Easy, for example the 2-dimensional recursive integration algorithm will do the work. Note that the integrator can do all integrals in one shot, if the intgrand returns an array of values.

Problem 3.42: Multipole integrals(2): Do the same as above, but for identical elements on r_1 and r_2. The recursive algorithm will take long time at moderate accuracies (if it converges at all, try it!). Find a better way of doing these integrals.
Hint: The coordinate transformation $r_1, r_2 \to (r_1 \pm r_2)/\sqrt{2}$ helps a lot. Write an integration routine for these coordinates or (better) adapt `integral_nd.py` for non-rectangular integration volumes.

Matrix elements of the Coulomb interaction

A single term in the sum is (using $\Omega_i = (\theta_i, \varphi_i)$):

$$\begin{aligned}
M_{lm,ij} &= \frac{4\pi}{2l+1} \int d\Omega_1 \int d\Omega_2 Y^*_{l_i m_i}(\Omega_1) Y^*_{l'_i m'_i}(\Omega_2) Y^*_{lm}(\Omega_1) Y_{lm}(\Omega_2) Y_{l_j m_j}(\Omega_1) Y_{l'_j m'_j}(\Omega_2) && (3.95)\\
&\quad \times \int_0^\infty dr_1 r_1^2 \int_0^\infty dr_2 r_2^2 h_{k_i}(r_1) h_{k'_i}(r_2) \frac{r_<^l}{r_>^{l+1}} h_{k_j}(r_1) h_{k'_j}(r_2) && (3.96)\\
&= \frac{4\pi}{2l+1} \int d\Omega_1 Y^*_{l_i m_i}(\Omega_1) Y^*_{lm}(\Omega_1) Y_{l_j m_j}(\Omega_1) && (3.97)\\
&\quad \times \int d\Omega_2 Y^*_{l'_i m'_i}(\Omega_2) Y_{lm}(\Omega_2) Y_{l'_j m'_j}(\Omega_2) && (3.98)\\
&\quad \times \int_0^\infty dr_1 r_1^2 \int_0^\infty dr_2 r_2^2 h_{k_i}(r_1) h_{k'_i}(r_2) \frac{r_<^l}{r_>^{l+1}} h_{k_j}(r_1) h_{k'_j}(r_2) && (3.99)
\end{aligned}$$

The angular integrals factorize and can be easily calculated (analytically, by numerical integration, or finding existing software subroutines). For the integration over the radial coordinates, there is a variety of strategies. The simplest is to employ the two-dimensional recursive integration used before in the one dimensional Helium model. In the present case convergence of this routine will not be rapid, as there is a non-anlyticity at the line $r_1 = r_2$. Another strategy is to isolate the singularity and perform the integral as (see also figure 3.4 and 3.5):

$$\int_0^\infty dr_1 \int_0^\infty dr_2 = \int_0^\infty dr_1 \int_0^{r_1} dr_2 + \int_0^\infty dr_1 \int_{r_1}^\infty dr_2 \tag{3.100}$$

which can be done rapidly using numerical quadratures.

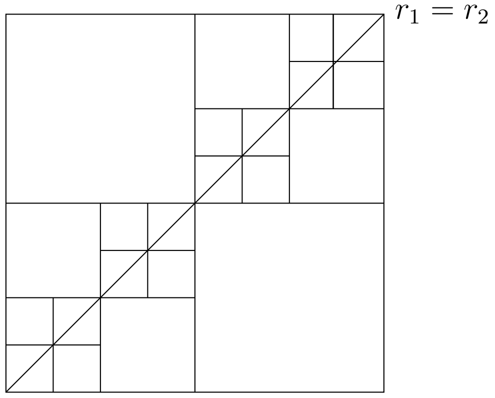

Figure 3.4: If we apply a quadrature to a function which is not smooth for $r_1 = r_2$ we will have to iterate the quadrature many times in the vicinity of the singularity, until we will have intervals which are so small that can be regareded as a zero measure set. In the other regions we will deal with huge matrices that can be written in a nice form (see also figure 3.5).

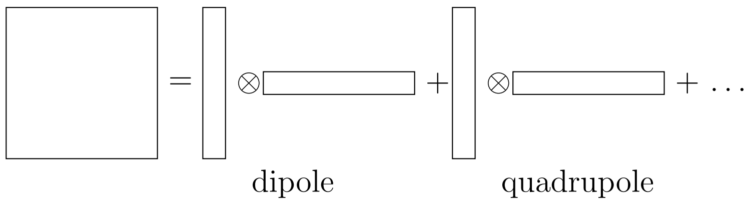

Figure 3.5: The huge matrix in figure 3.4 can be written in this nice form using the so called low-rank approximation for a matrix.

3.3.5 Multipole expansions: tensor product approximations

Form is specific for the Coulomb potential, the general principle can be applied to most typical interactions in physics: the interaction becomes smoother at large separations $|\vec{r}_1 - \vec{r}_2|$. At these distances, we need very few terms to approximate the "structure" of the interaction (e.g. monopole + dipole + quadrupole). In matrix language, this is a "tensor product approximation" to the operator.

3.3.6 Low rank approximation

The tensor product approximation to multiplication operators is a generalization of a very fundamental technique for approximating matrices: the low rank approximation.

Rank of a matrix

The "rank" of a matrix $M \times N$ matrix $\widehat{A}$ is the dimension of the "range" of the matrix, i.e. dimension of the the space spanned by all vectors obtained by applying the matrix:

$$\text{range}(\widehat{\text{A}}) = \text{span}(\widehat{A}v \in R^M | v \in R^N) \tag{3.101}$$

Note that rank$\widehat{A} \leq \min(M, N)$. The rank can be very small, including, trivally, rank 0 for the zero-matrix, or rank 1 for the tensor product of two vectors $\widehat{A} = \vec{u} \otimes \vec{v}^T = |u\rangle\langle v|$.

Singular value decomposition (SVD)

Any $M \times N$ matrix can be written as

$$\widehat{A} = \widehat{U}\Sigma\widehat{V}^\dagger \tag{3.102}$$

where $\widehat{U}$ is $M \times M$ and unitary and $\widehat{V}$ is $N \times N$ and unitary. Σ is $M \times N$ and "diagonal" in the sense $\Sigma_{ij} = \delta_{ij}\sigma_{\min(i,j)}$, $i \in 0, 1, \ldots, M-1$, $j \in 0, 1, \ldots, M-1$.

The spectral decomposition is a special case of SVD that exist only for square matrices with $\widehat{A}\widehat{A}^\dagger = \widehat{A}^\dagger\widehat{A}$ (therefore also for hermitian matrices $\widehat{A} = \widehat{A}^\dagger$). For the spectral decomposition $\widehat{U} = \widehat{V}$, for hermitian matrices Σ is real.

Singular value decomposition algorithms are provided by standard linear algebra packages (e.g. LAPACK). Sometimes, e.g. when comparing differnt algorithms for SVD, it is good to remember, that the SVD is not unique: we can multiply the diagonal elements of Σ and correspondingly $\widehat{U}$ and $\widehat{V}$ by arbitrary phases without changing the general SVD structure.

Approximating a matrix

Often, like in the multipole expansion of the Coulomb potential, there is very little structure in a matrix, even though it appears as a full matrix. A trivial example is a 1-dimensional projector which is a rank 1 matrix.

$$P_u = |u\rangle\langle u|, \quad \langle u|u\rangle = 1 \tag{3.103}$$

If written in an arbitrary basis, it will be represented by a full matrix

$$P_u = \begin{pmatrix} u_0u_0^* & u_0u_1^* & \ldots & u_0u_{N-1}^* \\ u_1u_0^* & u_1u_1^* & \ldots & u_1u_{N-1}^* \\ \vdots & \vdots & \ddots & \vdots \\ u_{N-1}u_0^* & u_{N-1}u_1^* & \ldots & u_{N-1}u_{N-1}^* \end{pmatrix} \tag{3.104}$$

i.e. N^2 numbers, although, actually, contains only N relevant different numbers $u_0, u_1, \ldots u_{N-1}$

The singular value decomposition is the tool to detect such structures and exploit them for physical insight or numerical efficiency. Assume rank$A = K$ and the columns of $\widehat{U}$ and $\widehat{V}$ are sorted such that $\Sigma_{ii} = 0$ for $i > K$. Then, clearly

$$\widehat{A} = \sum_i \widehat{U}_K \Sigma_K \widehat{V}_k^\dagger \tag{3.105}$$

where Σ_K is $K \times K$, $\widehat{U}$ is $M \times K$ with only the comlums up to K, and similarly $\widehat{V}_K$.

Even when $\widehat{A}$ has full rank rank$(\widehat{A}) = \min(M,N)$ mathematically, many Σ_{ii} may be very small and neglecting them will not cause significant error. E.g. no multipole term is excatly=0, but infinitly many will be arbitrarily small. Now select the largest elements $|\Sigma_{ii}| > \epsilon$ for $i > K^{(\epsilon)} \leq$ rank$(\widehat{A})$. Then one obtains a "low rank approximation"

$$\widehat{A} \approx \widehat{A}_\epsilon = \sum_i \widehat{U}_{K^{(\epsilon)}} \Sigma_{K^{(\epsilon)}} \widehat{V}_{K^{(\epsilon)}}^\dagger \tag{3.106}$$

Example: SVD of interaction potentials

Suppose you have, on two grids x_i and y_j

$$\widehat{V}_{ij} := V(|x_i - y_j|) \tag{3.107}$$

then a low rank approximation is

$$\widehat{V} \approx \sum_{k=0}^{M-1} \widehat{U}_k \otimes \widehat{W}_k^T \tag{3.108}$$

where $\widehat{U}_k$ are $I \times 1$ matrices and $\widehat{W}_k$ are $J \times 1$ matrices and $M \leq$ rank$\widehat{V}$. For $M =$ rank$\widehat{V}$ the sum becomes exact.

Problem 3.43: Study low rank approximations to multiplication operators.

Hierarchical matrices

Low rank approximation is not a useful strategy for all matrices: the unit matrix has full rank, at best nothing is gained by attempting a low rank approximation.

In cases where all the trouble comes from some small region (e.g. near $\vec{r}_1 = \vec{r}_2$), on can split the matrix into a small local part (e.g. the diagonal), which is trivial to apply, and the rest, where low rank approximations are efficient. A systematic procedure for that are the "hierarchical" or H-matrix approximations.

3.3.7 Implementation - managing the complexity

Representation of the wave function

- $|i\rangle$ — `BasTwo`

$$\psi(r_1, r_2) = \sum_{l_1,l_2} \sum_{L,M} \sum_{i,j} D_{L,M}^{l_1 l_2}(\theta_1, \phi_1; \theta_2, \phi_2) h_i(r_1) h_j(r_2) a_{ij}^{LMl_1l_2} \tag{3.109}$$

$$(L^{(2)})^2 D_{L,M} = L(L+1)D_{L,M} \tag{3.110}$$

$$L_z^{(2)} D_{L,M} = M D_{L,M} \tag{3.111}$$

$$D_{L,M}^{l_1 l_2} = \sum_{m_1,m_2} c_{l_1 m_1 l_2 m_2}^{LM} Y_{l_1 m_1}(\theta_1,\phi_1) Y_{l_2 m_2}(\theta_2,\phi_2) \tag{3.112}$$

where $c_{l_1 m_1 l_2 m_2}^{LM}$ are the Clebsch-Gordan coefficients.

$$\frac{1}{|r_1 - r_2|} = \sum_{l,m} \frac{4\pi}{2l+1} Y_{lm}^*(\theta_1,\phi_1) Y_{lm}(\theta_2,\phi_2) \frac{r_<^l}{r_>^{l+1}} \tag{3.113}$$

Representation of the wave function

- `BasTwo`

  The class `BasTwo` is a list of operator momentum blocks associated with $D_{L,M}^{l_1,l_2}$.

- `BasTwoAngle`

  In The class `BasTwoAngle` we define the $D_{L,M}^{l_1,l_2}$ in terms of the Clebsch Gordan coefficients $c_{l_1 m_1 l_2 m_2}^{LM}$.

- `BasTwoRadial`

  This class is a pair of discretization axis for r_1 and r_2.

Setting up the matrix

- `Gaunt factors`:

$$G(l_i, l, l_j, m_i, m, m_j) = \int d\Omega Y_{l_i m_i}^* Y_{lm} Y_{l_j m_j} \tag{3.114}$$

  and consequently

$$\int d\Omega Y_{l_i m_i}^* Y_{lm}^* Y_{l_j m_j} = G^*(l_j, l, l_i, m_j, m, m_i). \tag{3.115}$$

Problem 3.44: Using the definitions of the associated Legendre functions given in the script, implement LegendreAssoc. Implement a quadrature rule that delivers exact overlap matrix elements for (products of) associated Legendre functions with the sum of m's an even number. Using this quadrature, write a subroutine "GauntCoeff(l1,l2,l3,m1,m2,m3)" for computing the "Gaunt coefficients" arising in the multipole expansion:

$$\int d\Omega Y_{l_1 m_1}^*(\Omega) Y_{l_2 m_2}(\Omega) Y_{l_3 m_3}(\Omega) \tag{3.116}$$

Hint: For LegendreAssoc, start from LegendrePolynomial and modify it. For the quadrature, observe that all integrals where there appear odd powers of $(1-x^2)^{1/2}$ give =0 because of the ϕ-integration.

Problem 3.45: Find python code for the Clebsch-Gordan coefficients from the web. Incorporate a test routine into the software you found by verifying the following identies of the Clebsch-Gordan coefficients for a sufficiently large range of angular momenta (say, $L, l_1, l_2 \sim 5$).

$$\sum_{m_1,m_2} \langle L, M|l_1, m_1, l_2, m_2\rangle\langle l_1, m_1, l_2, m_2|L', M'\rangle = \delta_{JJ'}\delta_{MM'} \tag{3.117}$$

$$\sum_{L,M}\langle l_1, m_1', l_2, m_2'|L, M\rangle\langle L, M|l_1, m_1, l_2, m_2\rangle = \delta_{m_1m_1'}\delta_{m_2m_2'} \tag{3.118}$$

3.4 Mathemagics: complex scaling

Sometimes, under the influence of some perturbation, bound states disolve into a continuum. Still there is good reason to believe that they leave some trace and can be described as approximate bound states that decay.

3.4.1 H atom in a static electric field

In presence of an (arbitrarily small) static electric field, there are no exact square integrable eigenfunctions and thus there are no rigorous bound state for a potential $-1/r - \epsilon z$: the physical reason is that *tunneling* is always possible!

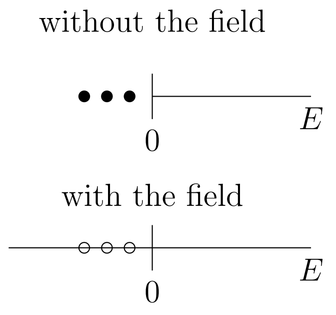

Figure 3.6: Eigenvalues for the hydrogen atom without an electric field (above) and with a static electric dipole field (below). The empty circles mark the positions of previous eigenvalues.

The exact spectrum (wihout a field) of a hydrogen atom can be seen in Figure 3.6. But what happens if we add a field? No matter how small the field is, the spectrum becomes **all** the real axis (i.e. continuous, see Figure 3.6). Now the following question arises: What happens to our bound states? Are there some traces of them in the continuous spectrum? One would expect that they turn into "resonances", i.e. poles in the scattering (S-) matrix. But these resonances do not appear at **real** images. They appear upon analytical continuation of the S-matrix to complex energies. So we can also ask: What should be the properties of those S-matrices? But let us first look again at the potential of the hydrogen atom in a field. We can see an oscillation of the wave function with decreasing amplitude which is a trace of our bound state. But the wave function is **not** normalizable and thus it has **not** an isolated point in the spectrum.

As we saw in both examples a “perturbation” let a bound state decay into the continuum (external field or Coulomb interaction). We can make them visible by “complex scaling”. As we said before, the bound states do not completely disappear, they leave traces as *poles of the S-matrix* in the complex plane (at complex values of the energy). We can also *continue analytically the Hamiltonian operator*. The new operator has *complex* eigenvalues $W = E - i\Gamma/2$, exactly for the location where the S-matrix has the poles.

Note: In this method, some rigorous mathematical proofs exist, mostly for the hydrogen atom and some model potentials. Many results have no rigorous proof (yet), but there is ample numerical evidence that they indeed hold.

3.4.2 Real scaling of the coordinates

The real scaling of the coordinates is like changing the units in which we measure our results. Thus it clearly cannot change the physics. The scaling transformation can be made a unitary operator:

$$\vec{r} \to \alpha\vec{r} = e^{\lambda}\vec{r}, \quad \alpha \in (0, \infty). \tag{3.119}$$

Define:

$$U_\lambda \Psi(\vec{r}) := e^{-3\lambda/2}\Psi(e^{\lambda}\vec{r}). \tag{3.120}$$

This definition conserves the norm:

$$||U_\lambda\Psi||^2 = \int d^{(3)}r e^{-3\lambda}|\Psi(e^{\lambda}\vec{r})|^2 \overset{\vec{r}e^{\lambda}=:\vec{q}}{=} \int d^{(3)}q|\Psi(\vec{q})|^2 = ||\Psi||^2 \tag{3.121}$$

and has a trivial inverse $U_\lambda^{-1} = U_{-\lambda}$. Thus U is unitary. Furthermore we define the **real scaled** Hamiltonian:

$$\mathbf{H}_\lambda := U_\lambda \mathbf{H} U_\lambda^*. \tag{3.122}$$

What does this mean for the eigenvalues of $\mathbf{H}_\lambda$? The answer is that the unitary transformations leave the spectrum invariant (i.e. $\sigma(\mathbf{H}_\lambda) \equiv \sigma(\mathbf{H}_{\lambda=0})$). All $\mathbf{H}_\lambda$ have the same spectrum, i.e. the same bound state spectrum and the same continuous spectrum. The reasons for this are the following:

- A unitary transformation does not change the spectrum of any operator.
- The physics **remains unchanged** by scaling transformations.

Let us, for example, look at the hydrogen atom with a scaled operator:

$$\begin{aligned}\mathbf{H}_\lambda \quad &= -\frac{1}{2}\left(\frac{1}{r^2}\partial_r r^2 \partial_r\right)e^{-2\lambda} - \frac{1}{e^{\lambda}r} + \frac{l(l+1)}{2r^2e^{2\lambda}} &(3.123)\\ &= e^{-2\lambda}T + e^{-\lambda}V = -e^{-2\lambda}\frac{1}{2}\Delta - e^{-\lambda}\frac{1}{r}, &(3.124)\end{aligned}$$

where we just replaced $\vec{r} \to e^{\lambda}\vec{r}$ and thus $\partial_r \to \frac{\partial}{\partial(e^{\lambda}r)} = \frac{1}{e^{\lambda}}\frac{\partial}{\partial r}$.

Eigenfunctions and eigenvalues (bound states)

Let ψ_i be an eigenfunction of the Hamiltonian $\mathbf{H}_{\lambda=0}$. Then $\mathbf{H}_0\psi_i = E_i\psi_i$. From this we get:

$$U_\lambda \mathbf{H}_0 \underbrace{U_\lambda^* U_\lambda}_{\mathbf{1}} \psi_i = U_\lambda \mathbf{H}_0 \psi_i = E_i U_\lambda \psi_i =: E_i \psi_{i,\lambda}. \tag{3.125}$$

So we have:

$$\mathbf{H}_\lambda \psi_{i,\lambda} = E_i \psi_{i,\lambda} \tag{3.126}$$

and see that E_i is independent of λ like for example in the hydrogen atom. In this case ψ_i is proportional to the **Laguerre polynomials** $\times e^{-kr}, k = 1/n$. The derivatives are purely algebraic operations. Thus the eigenfunction property is conserved.

3.4.3 Complex scaling of the coordinates

Let us now take a bold step. We analytically continue $\mathbf{H}_\lambda$ to complex values of λ. For simplicity we first only look at the imaginary values $\lambda \to i\theta$, $\theta > 0$. Therefor, as a minimum requirement, we must be able to analytically continue the potential, i.e. define $V(e^{i\theta}\vec{r})$. The complex scaled Hamiltonian reads:

$$\mathbf{H}_\theta := -e^{-2i\theta}\frac{1}{2}\Delta - e^{-i\theta}\frac{1}{r}. \tag{3.127}$$

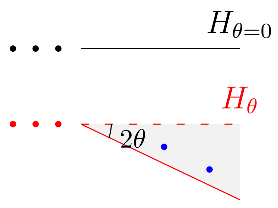

Figure 3.7: Comparison of the spectrum of $H_{\theta=0}$ (above) with the spectrum of H_θ (below, red). The blue dots represent the resonance states, which give the pole of the S-matrix.

Let's now study the spectrum of H_θ. It is possible to distinguish between three different kinds of eigenvalues (see also figure 3.7):

- For bound states the specturm of H_θ is the same as the spectrum of $H_{\theta=0}$, i.e. the bound states spectrum is independent of θ;
- In order to understand what happens at the continuous spectrum we will look at the free particle:

  $$\Delta \to \Delta_\theta = e^{-2i\theta}\Delta_0. \tag{3.128}$$

  The spectrum of $H_{\theta=0}$ is given by $[0,\infty)$ whereas the spectrum of H_θ is given by $e^{-2i\theta}[0,\infty)$ and is therefore complex (a complex spectrum was expected, since H_θ is not an hermitian operator). Therefore we have that the continuous spectrum is rotated into the lower complex plane by an angle 2θ;

- in the shaded region in figure 3.7 there can be new true (normalizable) eigenstates with complex eigenvalues $W = E - \frac{i\Gamma}{2}$, $\Gamma \geq 0$. These states are known as "resonance states" with resonance energies E and resonance width Γ. Further the position of these complex eigenvalues is independent of the angle θ.

These last eigenvalues are the most interesting because they give the poles of the S-matrix. If we let $|W\rangle$ be the state corresponding to W we will have that $H_\theta|W\rangle = W|W\rangle$ and:

$$i\frac{d}{dt}|W\rangle = W|W\rangle; \qquad W = E - \frac{i\Gamma}{2}. \tag{3.129}$$

If we take for simplicity $E = 0$:

$$i\frac{d}{dt}|W\rangle = E - \frac{i\Gamma}{2}|W\rangle \Rightarrow |W,t\rangle = e^{-\frac{t\Gamma}{2}}|W,0\rangle \tag{3.130}$$

and therefore the probability of finding $|W,t\rangle$ in $|W,0\rangle$ is $|\langle W,0|W,t\rangle|^2 \sim e^{-\Gamma t}$.

3.4.4 Doubly excited states of Helium

Imagine He^+ ion in some excited state:

Suppose we have an electron e_1^* that is in the excited state $n_1 = 2$. Then the energy is $-Z^2/2n^2 = -4/8 = -1/2$. Suppose furthermore that another electron e_2^* is in some arbitrary excited state n_2. If e_1^* and e_2^* do not interact then we have a bound state because nothing will change. But if we now "switch" on the Coulomb interaction, the electrons e_1^* and e_2^* will exchange energy. (Note that if e_2^* is far away from e_1^* it will only "see" a single charge.) In this case the electron e_1^* could deexite and give its energy to e_2^*. What is the total energy in this case? The total energy $e_1^* + e_2^*$ for e.g. $n_2 = 2$ is $-1/2 - 1/2 = -1$. Then the total energy after the deexitation of e_1^* (now $n_1 = 1$) is $-2 + E_2 = -1 \Rightarrow E_2 = 1 > 0$. This is an unbound state! We can see this, for example, in the Feschbach-resonance, core hole states and the Auger effect. So if it was not for the Coulomb interaction we would have a bound state. The question also here is: What are the traces of the doubly excited state in the continuum spectrum?

Problem 4.46: Doubly excited states of helium: the lowest three doubly excited states of Helium with L=0 have energies ~ -0.777, -0.621, and -0.589. Implement complex scaling for `he_inter.py`. Find those states in the calculations. What are their widths Γ? How accurate are your results for the widths and for the doubly excited state energies? Translate the widths into lifetimes $2\pi/\Gamma$ (1 atomic unit of time $\approx 24 \times 10^{-18}s$).
Hint: For implementing, split the definition `ham` of the Hamiltonian into kinetic energy and potential, generate the matrices and add them with their correct complex scaling factors.
Hint: For choosing the exponents, remember that the doubly excited states will be more extended than the ground state, i.e. they will be better approximated by choosing somewhat smaller exponents.

Hint: For estimating the accuracy of your results (1) vary the basis size, (2) vary the exponents, and (3) vary the scaling angle. The result is at most accurate to the extend that it does not depend on any of these changes (within "reasonable" ranges of variations).

3.4.5 Energy shifts and decay of an atom in a static electric field

Complex scaling allows computing Stark-shifts and tunneling decay width without resorting to perturbation theory. Strictly speaking, complex scaling as laid out above lacks mathematical foundation for the Stark effect (the dipole interaction is not "dilation analytic"). Remember, however, that also perturbation theory for the Stark effect diverges. Still meaningfull results can be extracted from high order perturbation theour using further mathematical tricks. Naive application of complex scaling to the Stark effect gives correct answers, whose accuracy is only limited by the usual discretization errors. "Correctness" is here defiend two-fold: it agrees with the results from 18th(!) order perturbation theory and with results from observing computing tunneling rates with time-dependent perturbation theory.

Problem 4.47: Repeat the calculation of the Stark effect, now using complex scaling. Extend `hfield_simple.py` to include complex scaling (use the example `he_inter.py`). If your basis is good enough, you can identify the "true" eigensolutions by the fact that they are independent of the scaling angles. Compare your results with the previous results without using complex scaling. In the imaginary parts of the energies you have additional information about the decay width. Plot the life-times $2\pi/\Gamma$ of the the first few states as a function of field strength (logarithmic axis recommended, one atomic unit of time is $\approx 24 \times 10^{-18}s$). How do life-times depend on field strength? Interprete the result in terms of a Gamov factor for tunneling.

Chapter 4

The time-dependent Schrödinger equation

If we know the complete set of eigenfunctions of a Hamiltonian matrix, we can very easily construct the resulting time-evolution using the spectral representation of the time-evolution operator

$$U(t_0,t) = \exp(-i(t-t_0)\mathbf{H}) = V\exp[-i(t-t_0)d_H]V^\dagger, \tag{4.1}$$

where V is composed of the eigenfunctions of H and d_H is the diagonal matrix of its eigenvalues.

In practice, this can rarely be done. If $H(t)$ itself depends on time, the time-evolution operator can still be formally written in closed form using the time-ordering symbol "T":

$$U(t_0,t) = T\exp[-i\int_{t_0}^{t} d\tau H(\tau)], \tag{4.2}$$

but this notation is highly symbolic, with T standing in for an infinite sum of nested integrals of commutators. It can rarely be turned into a useful computational scheme, although such methods exist and will be discussed later.

Usually, the explicit construction of the time-propagation operator is expensive (if feasible at all). This reflects the fact that it is the answer to "everything", i.e. the time-evolution of any conceivable state of the system. Almost always, our ambition is much more modest: we want the time-evolution of one specific initial state. This restraint in ambition results in much reduced computational challenge: solving the ordinary differential equation:

$$i\frac{d}{dt}|\Psi(t)\rangle = H(t)|\Psi(t)\rangle, \quad |\Psi(t_0)\rangle = |\Psi_0\rangle. \tag{4.3}$$

Problem 0.48: Setting an initial state: express an arbitrary function $\Psi_0(x)$ in a basis set

1.
$$\Psi(x) \approx \tilde{\Psi}(x) = \sum_i f_i(x)c_i. \tag{4.4}$$

Show that the coefficient vector $\vec{c}$ given by

$$\vec{c} = \widehat{S}^{-1}\vec{b}, \quad b_j = \langle f_j|\Psi\rangle \tag{4.5}$$

is the optimal approximation in the sense of a Galerkin criterion, i.e. the error is orthogonal to the subspace spanned by the basis.

2. Implement the above for the finite element basis.

3. Plot the $\tilde{\Psi}$, Ψ and visualize the error in a meaningful way.

4.1 Discretization

Suppose we decide to discretize our wave function by a finite set of linear time-dependent parameters $c_i(t)$, collected in a vector $\vec{c}(t)$:

$$|\Phi_{\vec{c}(t)}\rangle \approx |\Psi(t)\rangle. \tag{4.6}$$

Now we look for a solution $\vec{c}(t)$ such that $\Phi_{\vec{c}(t)}$ is "closest" to $\Psi(t)$ on some time interval $[t_o, t]$. But we still have to define what we mean by "closest".

Time-global optimality

In time we may choose a time interval $[t_0, t]$ and choose a distance between Φ and Ψ, e.g. $||\cdot||$, the norm in Hilbert space. We then search for the minimum:

$$\min_{\vec{c}(t)} \langle ||\Phi_{\vec{c}(t)} - \Psi(t)||\rangle_{[t_0,t]} \tag{4.7}$$

in the space of all coefficient-functions $c_i(t)$, where $\langle\cdot\rangle_{[t_0,t]}$ is a measure of the distance between two paths in time. This is probably what we would like best. Its practical use is limited for two reasons: (1) It adds an extra dimension to the problem ($\rightarrow$ curse of dimension $\rightarrow$ *much* bigger problem than, e.g., just finding eigenstates) and (2) for time, there is no similarly simple principle of optimization as we have with the minimax principle for space. Remember that the minimax principle is limited to operators with a lowest eigenvalue, but time extends to the infinite past (e.g. $i\partial/\partial t$ is not bounded).

Time-local optimality

We want to solve the Schrödinger equation "as good as we can" under the constraint that $\Phi_{\vec{c}(t)}$ has the form $\Phi = \sum_i |i\rangle c_i$, i.e.:

$$i\frac{\partial}{\partial t}|\Phi_{\vec{c}(t)}\rangle - H(t)|\Phi_{\vec{c}(t)}\rangle \overset{!}{=} 0. \tag{4.8}$$

As we can represent only a subset of the full Hilbert space, the equation cannot usually be fulfilled exactly. What we can get is

$$i\frac{\partial}{\partial t}|\Phi_{\vec{c}(t)}\rangle - H(t)|\Phi_{\vec{c}(t)}\rangle = |\chi_\varepsilon\rangle. \tag{4.9}$$

Then we demand that the error that we make by inserting our approximate solution into the TDSE at each time t is the "unavoidable error" given a certain discretization, i.e. a certain basis $|i\rangle$ or a certain subspace $\mathcal{K} \subset \mathcal{H}$. By unavoidable we mean that the error is nothing that we can represent by our basis, so we can also not "bring it to the other side" to further reduce it: $|\chi_\varepsilon\rangle \notin \mathcal{K} \subset \mathcal{H}$. This means that the error is *orthogonal* to $\mathcal{K}$:

$$\langle \Phi_{\vec{d}} | \chi_\varepsilon \rangle = \langle \Phi_{\vec{d}} | i\frac{d}{dt} - \mathbf{H} | \Phi_{\vec{c}(t)} \rangle = 0 \qquad \forall \vec{d}. \tag{4.10}$$

In mathematics, this kind of minimal principle is called a "Galerkin method". In physics, we call it the "Dirac-Frenkel" variational principle.

Problem 1.49: Show that the Dirac-Frenkel variational principle can also be formulated as a minimum principle:

$$F := \langle \Phi_{\vec{c}(t)} | i\frac{d}{dt} - \mathbf{H}(t) | \Phi_{\vec{c}(t)} \rangle. \tag{4.11}$$

has its minimal for any time t. Show further that it implies the following equations for the time-dependence of the coefficients $\vec{c}(t)$

$$i\widehat{S}\frac{d}{dt}\vec{c}(t) = \widehat{H}\vec{c}(t), \tag{4.12}$$

where $\widehat{S}$ and $\widehat{H}$ are overlap and Hamiltonian matrix, respectively.
Hint: A necessary condition at the minimum of the function $F(\vec{d})$ is $\partial_{d_i} F(\vec{d}) = 0 \, \forall i$.

4.2 Solving the discretized TDSE

4.2.1 Solving the ODE

Simplest case: linear, time independent equation. Orthonormal basis $\widehat{S} = \mathbf{1}$

$$i\frac{d}{dt}\vec{c} = \widehat{H}\vec{c} \tag{4.13}$$

with initial condition $\vec{c}(0) = \vec{c}_0$. Formal solution by matrix exponentiation

$$\vec{c}(t) = \exp(-it\widehat{H})\vec{c}_0 \tag{4.14}$$

4.2.2 Exponentiation of a matrix $\widehat{A}$

Spectral representation

A matrix $\widehat{H}$ has a spectral representation if it can be written as $\widehat{H} = \widehat{W}\hat{d}\widehat{W}^\dagger$ where $\widehat{W}$ is a unitary matrix and $\hat{d}$ is diagonal.
If we use this spectral decomposition for $\widehat{H}$ we will have:

$$(\widehat{H})_{mn} = \sum_{k,l} \widehat{W}_{mk}\hat{d}_{kl}\widehat{W}^\dagger_{ln} = \sum_l (\vec{W}_l)_m E_l (\vec{W}_l^\dagger)_n \sim \sum_l |E_l\rangle E_l \langle E_l| \tag{4.15}$$

$$\vec{c}(t) = \widehat{W}\exp(-it\hat{d})\widehat{W}^\dagger\vec{c}_0; \qquad (e^{-i\hat{d}t})_{mn} = \delta_{mn}e^{-iE_n t} \tag{4.16}$$

Power series

Functions of $\widehat{A}$:

$$e^{\widehat{A}} = \sum_{n=0}^{\infty} \frac{\widehat{A}^n}{n!} \simeq \sum_{n=0}^{M-1} \frac{\widehat{A}^n}{n!}; \qquad e^{\widehat{A}}\vec{c} \simeq \sum_{n=0}^{M-1} \frac{\widehat{A}^n}{n!}\vec{c} \tag{4.17}$$

A linear combination of the vectors is given by $\chi_n = \vec{c}, \widehat{A}\vec{c}, \widehat{A}^2\vec{c}, \dots$ which spans the Krylov subspace. For large n we will obtain: $\vec{A}^n\vec{c} \simeq$ eigenvector associated to the largest eigenvalue. The vectors χ_n become near linearly dependent for large value of n. We get a poor representation of the solution due to numerical problems.

Chebyshev expansion

A valid alternative is given by an expansion of the exponential into orthogonal polynomials:

$$e^x \approx \sum_{n=0}^{N-1} a_n T_n(x) \tag{4.18}$$

where T_n is some set of orthogonal polynomials, say, Chebyshev (as they are used in practice) and a_n are the expansion coefficients. Like the power series, this is a Krylov method, but it is more stable numerically: we construct specific linear combination of the Krylov vectors rather then $\widehat{A}^n\vec{c}$.

For applying the N-th order approximate exponential to a vector, we only need $N-1$ matrix-vector multiplications:

$$e^{\widehat{A}}\vec{c} \approx \sum_{n=0}^{N-1} a_n T_n(\widehat{A})\vec{c} \tag{4.19}$$

Remember that any orthogonal polynomial of any algebraic object is given by three-point recurrence relations, where each recurrence requires only a single multiplication:

$$T_n(\widehat{A}) = (a_n\widehat{A} + b_n)T_{n-1}(\widehat{A}) + T_{n-2}(\widehat{A}). \tag{4.20}$$

The coefficients a_n can be determined by:

$$\int_a^b dx T_m(x) exp(x) = \int_a^b \sum_n T_m(x) a_n T_n(x) dx \tag{4.21}$$

so that one gets the equations:

$$\vec{b} = \widehat{S}\vec{a} \qquad \vec{a} = \widehat{S}^{-1}\vec{b} \qquad b_m = \int_a^b dx T_m(x) e^x \tag{4.22}$$

In order to evaluate $T_n(\widehat{A})\vec{\chi}$ one can use the recurrence relation for the orthogonal polynomials:

$$T_n(\widehat{A})\vec{\chi} = (a_n + b_n\widehat{A})T_{n-1}(\widehat{A})\vec{\chi} + c_n T_{n-1}(\widehat{A})\vec{\chi} \tag{4.23}$$

Therefore to determine $T_n(\widehat{A})\vec{\chi}$ only n matrix-vector multiplications are required, which is exactly the same cost we had before, but now it is more stable.

Problem 2.50: Write a matrix-exponentiation using general orthogonal polynomials T_n in the form

$$\exp(\widehat{A})\vec{c} \approx \sum_{n=0}^{N-1} a_n T_n(\widehat{A})\vec{c} \tag{4.24}$$

Make the code independent of the specific choice of orthogonal polynomials, along the lines given in `orthopol.py`. Compare the performance of these methods for given N with the classical Runge-Kutta method implemented in `odesolver.py`.
Hint: The factors a_n are defined in the same way as for functions of real numbers:

$$\exp(x) \approx \sum_{n=0}^{N-1} a_n T_n(x) \tag{4.25}$$

from which they can be determined by using the linear independence of the polynomials. In practice, the factors a_n should be obtained using numerical quadratures during setup of the method. The vectors $\vec{c}_n := T_n(\widehat{A})\vec{c}$ by recurrence starting from $\vec{c}_0$. For recurrence, you may try to adapt `orthopol.py`.

Hint: You may take the `class RungeKutta` in `odesolver.py` as a model for your code.

4.2.3 Runge-Kutta methods

$$\frac{d}{dt}y(t) = f(y(t), t) \tag{4.26}$$

is the general ODE. Given $\vec{y}(t) =: y_0$, we want to obtain an approximation $y_1 \simeq y(t+h)$ for step size h.

Here is the recipe for a general RK method:

- Compute s auxiliary vectors $\vec{k}_i, i = 0, 1, \ldots, s-1$:

$$\vec{k}_i = \vec{f}(\vec{y}_0 + h\sum_{j=0}^{s-1} a_{ij}\vec{k}_j, t + hc_i) \tag{4.27}$$

- Obtain the approximate solution $y_1 = y_0 + h\sum_j b_j\vec{k}_j$

The numbers b_i, c_j, a_{ij} define a particular Runge-Kutta (RK) method. The common way to write all these terms is a the Butcher tableau (see table 4.1). The number s of support vectors $\vec{k}_i$ and for a given RK method is called the *stage* of the method.

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \dots & a_{1s} \\ c_2 & a_{21} & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & \vdots \\ c_s & a_{s1} & \dots & \dots & a_{ss} \\ \hline & b_1 & b_2 & \dots & b_s \end{array}$$

Table 4.1: Butcher tableau for a RK method. If the method is explicit only the lower triangle is unequal zero.

Example: Butcher tableau for the "classical" RK method (stage = 4)

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Note, the the classical "RK4" method only requires storing a *single* auxiliary vector $\vec{k}_j$

4.2.4 Explicit vs. implicit RK methods

There is an important distinction between two classes of RK methods:

- **Explicit RK method:** the matrix if the a_{ij} is lower triangular with diagonal =0, i.e. $a_{ij} = 0$ for $j \geq i$:

$$\vec{k}_i = \vec{f}(\vec{y}_0 + h \sum_{j=0}^{i-1} a_{ij} \vec{k}_j, t_0 + hc_i). \tag{4.28}$$

  Explicit methods are very easy to apply: one can construct $\vec{k}_i$ recursively as each $\vec{k}_i$ depends only on the $\vec{k}_j$ that have been calculated previously. We need s evaluations the derivatives only. For linear problems

$$\vec{f}(\vec{y}, t) = \widehat{H} \vec{y} \tag{4.29}$$

  the approximation is a linear combination of the Krylov vectors $\widehat{H}^n \vec{y}_0, n = 0, 1, \dots, s-1$: explicit Runge-Kutta is a Krylov subspace method.

- **Implicit RK method:** $\exists j \geq i : a_{ij} \neq 0$: requires solving the *implicit equation* for the $\vec{k}_i$

$$\vec{k}_i = \vec{f}(\vec{y}_0 + h \sum_{j=0}^{s-1} a_{ij} \vec{k}_j, t_0 + hc_i) \tag{4.30}$$

  Implicit methods are usually much harder to implement than explicit methods, as the system of equations needs to be solved. However, explicit methods in general suffer from instability (see discussion below), which is why — depending on the problem at hand — one may decide to use implicit methods.

Problem 2.51: Write a class `RungeKutta` that implements general explicit Runge-Kutta methods. Use a table of Butcher tableaus which contains the specifications for the different methods. As a minimum, implement the "classical" 4th order Runge-Kutta, a second order RK scheme, and the Euler forward step (or more, if you enjoy it). The class should have the following methods:

- The `__init__` constructor to select and set up the method.
- The single time-step `step`: $y(x) \to y(x+h)$ using a function `derivative(y,x)`.
- A `test` to test on the examples
$$\partial_x y = \lambda y \tag{4.31}$$
and on the classical pendulum equations
$$(\dot{x}, \dot{p}) = (p, -x) \tag{4.32}$$
- A `__str__` method to display the method (useful for debugging).

Make sure the class works for arbitrary objects, if only the derivatives form a vector space, i.e. can be added and can be multiplied by a scalar.

Let M be some mapping that generates an approximation $M[y_{n-1}, y_n, t, h] = y_n \approx y(t+nh)$ to the solution of the ODE with initial condition $y_n = y(t+(n-1)h)$.

4.2.5 Existence of a solution, consistency, convergence

Lipschitz condition

There exists L such that and G some subset of R^2

$$|f(y,t) - f(\tilde{y},t)| \leq L|y-\tilde{y}| \qquad \forall (y,t) \in G \tag{4.33}$$

This holds in particular for functions that are differentiable on a compact and convex region. L is then the maximum of the ∂_y-derivative on that set

$$L = \max_{(y,t)\in G} |\partial_y f(y,t)| \tag{4.34}$$

Consistency of a method

A method is *consistent* if, with decreasing step size, the error that we make in a single step goes to zero faster than the step size, i.e.:

$$\lim_{h\to 0} \frac{1}{h}\left[y(t+h) - M(y(t),t,h)\right] = 0 \tag{4.35}$$

This is a minimal requirement for a meaningful method: the error calculated to the step size must decrease with the step size. If it does not, integrating over a fixed interval by performing many steps one after the other will tend to add up to a constant overall error, no matter how small the individual step size.

Consistency order or just "order" of a method

A single step is accurate to order p in the step size h if:

$$[y(t+h) - M(y(t), t, h)] = \mathcal{O}(h^{p+1}). \tag{4.36}$$

Convergence of a method

Convergence requires that the accumulated error of the procedure over a finite interval $[t_0, t_1]$ can be made arbitrarily small.

To approximate the solution $y(t)$ across $[t_0, t_1]$ we compute solutions y_n on a sufficiently dense grid of $t_n = t_0 + nh$, $h = (t_1 - t_0)/N$ in the interval, recursively by:

$$y(t+nh) \approx y_n = M[y_{n-1}, t_{n-1}, h]. \tag{4.37}$$

The procedure is called *convergent* if the deviation of the $(y_0, y_1, \ldots, y_N)$ from $(y(t_0), y(t_0 + h), \ldots, y(\underbrace{t_0 + Nh}_{t_1}))$ goes to zero:

$$\lim_{h\to 0} \max_j |y_j - y(t_0 + jh)| = 0. \tag{4.38}$$

Convergence order q

How quickly, i.e. to which order in the step size, does a convergent method approach the exact result?

$$\max_j |y_j - y(t_0 + jh)| = \mathcal{O}(h^{q+1}) \tag{4.39}$$

We say the (single step) method has *convergence* order q.

Because of the accumulation of errors, the convergence order is typically smaller than the consistency order. Assuming random accumulation of errors, the errors can be expected to partially cancel so that $q \approx p - r$, where $r \approx 0.5$.

Accumulation of errors and lower limit to step size

The errors discussed so far are method related and would occur also in infinite precision calculations. However, mathematically, any method with consistency order $p > 1$ can be forced to convergence by just choosing step size h small enough.

However, *numerical error* due to finite precision may bring the method to a halt: for a finite interval, one needs

$$N = \frac{t_n - t_0}{h} \tag{4.40}$$

steps. At one point, the *numerical* errors in a single step will exceed the error of M on the same step and set a limit to the single step error: suppose the step h is so small, that the relative change of the *true* solution $y(t+h) - y(t) \sim 10^{14}$, i.e. approaches machine precision. Such a change cannot be correctly represented with 14-digit accuracy and random errors of the same size as the true change will increase: we cannot improve accuracy.

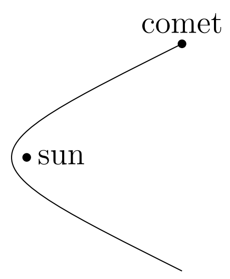

Figure 4.1: Trajectory of a comet approaching the sun. The trajectory is nearly a straight line when the comet is far from the sun, whereas it varies rapidly when the comet is close to the sun: a case for step size control!

Further reduction of h leads to an *increase* of the error on a finite interval $[t_0, t_1]$.
If you encounter this problem, likely reasons are (in this sequence)

- Over-ambitious discretization (see discussion of stability below)
- Bad ODE solver the demands too small step size h, you may need to use implicit solver.

4.2.6 Step size control

The motivation for step-size control can be understood by looking at a simple example from classical mechanics. Let's consider the trajectory of a comet that passes close by the sun. As can be also seen from figure 4.1 we can have very large steps when it is far from the sun and it has an almost straight trajectory, whereas we need a dense grid near the sun where it rapidly changes speed and direction.

Using step-size control we can obtain:

- gain in time;
- predictable/controllable accuracy.

A simple method for step size control is to compare larger and smaller steps and use the known consistency error to estimate the actual error. (see figure 4.2).

Single/double step size control

If we assume that the used method M has consistency of order p we can estimate the error from the difference of results $y_2^{(1)} - y_1^{(2)}$ obtained with two different step-sizes:
The scheme works as follows:

- Error in first h-step: $\delta_1 := y(h) - y_1^{(1)} = y(h) - M[y(0), t, h] =: C_1[h]h^{p+1}$
  (C_1 also depends on $y_1, y(0), t$, which we drop for notational simplicity).
- Error after 2nd h-step: $\delta_2 := y(2h) - y_2^{(1)} = \delta_1 + C_2[h]h^{p+1} = (C_1[h] + C_2[h])h^{p+1}$

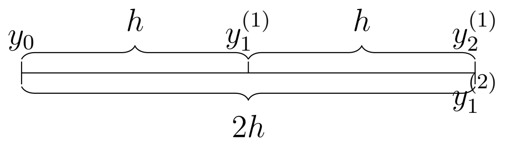

Figure 4.2: In order to estimate the error in a step-size control method one can compare results obtained with two different step sizes.

- Error in double-size $2h$-step: $\delta_3 := y(2h) - y_1^{(2)} = y(2h) - M[y(0), t, 2h] = C_3[h](2h)^{p+1}$
- *Assume* the C's constant: $C := C_1 \approx C_2 \approx C_3$
- from the difference $|\delta_3 - \delta_2| = |y_1^{(2)} - y_2^{(1)}| = |2Ch^{p+1} - C(2h)^{p+1}| = |\delta_2(2^p - 1)|$ we get the error estimate: $\delta_2 = |y(2h) - y_2^{(1)}| = |y_1^{(2)} - y_2^{(1)}|/(2^p - 1)$
- estimate step size γh for desired error ϵ:

$$\frac{(\gamma h)^{p+1}}{h^{p+1}} = \frac{\epsilon}{\delta_2} = \frac{(2^p - 1)\epsilon}{|y_1^{(2)} - y_2^{(1)}|} \tag{4.41}$$

  from which we obtain the correction factor γ:

$$\gamma = \left[\frac{(2^p - 1)\epsilon}{|y_1^{(2)} - y_2^{(1)}|}\right]^{1/(p+1)} \tag{4.42}$$

  (γ is a factor which scales the original step-size).
- get a new step size $h \longrightarrow \gamma h$.
- If $\gamma > 1 \Rightarrow \delta_2 < \epsilon$: accept $y_2^{(1)}$ as good approximation to $y(2h)$
  else: reject, start over from $y_0 = y(0)$ with new, reduced step size.
- **Important:** In practice one uses $h \longrightarrow \gamma hs$ where s is a safety factor < 1 introduced to avoid rejecting too many steps.

Note: The overhead/extra cost is 50 % as the $y_1^{(2)}$ value is used only for estimate.

Problem 2.52: Solve the ODE

$$i\frac{d}{dt}\vec{c} = \widehat{H}\vec{c} \tag{4.43}$$

where $\widehat{H}$ is a $N \times N$ random real symmetric matrix by exponentiating $\widehat{H}$ to obtain

$$\vec{c}(t) = \exp[-it\widehat{H}]\vec{c}(0) \tag{4.44}$$

in the spectral representation of $\widehat{H}$. Avoid using indices of matrices and eigenvectors as far as possible, use matrix notation instead and `numpy.dot` for matrix-matrix multiplications.

Problem 2.53: Implement the double-step error control algorithm (see notes) in the class `OdeSolver`. As the single step method, use any of the single-step ODE methods provided in `OdeSingle`. Verify your control using the equation

$$i\frac{d}{dt}\vec{c} = \widehat{H}\vec{c} \tag{4.45}$$

where $\widehat{H}$ is a $N \times N$ random real symmetric matrix. As the comparison "exact" solution you may use the matrix exponential in the previous example. You may use `odesolver.py` as a starting point (see `@classmethod test` and `def step` for code stubs).

Problem 2.54: Set up a solver for the ODE resulting from a basis set discretization

$$\frac{d}{dt}\vec{c} = -i\widehat{S}^{-1}\widehat{H}\vec{c}. \tag{4.46}$$

Implement this using `odesolver.py`. Use a model system of your choice (wave equation, harmonic oscillator, classical Kepler problem) to verify the correctness of your ODE solve by comparing to known solution.
Hint: The simple 1d wave equation translates the initial state by constant velocity, the average postion of a quantum mechanical operator follow exactly the classical motion, closed orbitals of the Kepler problem are ellipsis with known parameters.

Extrapolation methods

We have made the simplest assumption about the dependence of C on h, namely $C \equiv$const across the step size. If we assume that C is not constant, but some analytic function of h: $C(h)$, we can extrapolate results obtained with several step sizes to further improve the results. The simplest of these methods is "Richardson extrapolation" another method based on the idea of analytical continuation is Burlisch-Stoer.

Embedded methods

For reducing the overhead in step size control one can use so-called embedded RK methods. In these methods one forms different linear combintations of the same set of support vectors $\vec{k}_j$ to obtain two different approximations to the solution, typically of different consistency order:

$$\vec{y}_n^{(p)} = h\sum_j b_j^{(p)}\vec{k}_j \quad \text{and} \quad \vec{y}_n^{(p-1)} = h\sum_j b_j^{(p-1)}\vec{k}_j \tag{4.47}$$

Here as an example the Butcher tableau for a RK Fehlberg embedded method with orders p=4 and p=5 (RKF45):

0						
1/4	1/4					
3/8	3/32	9/32				
12/13	1932/2197	-7200/2197	7296/2197			
1	439/216	-8	3680/513	—845/4104		
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	
p=4	25/216	0	1408/2565	2197/4104	-1/5	0
p=5	16/135	0	6656/12825	28561/56430	-9/50	2/55

The only relevant overhead of the method is to form the two alternative linear combinations, which is usually negligible compared to computing the derivative. However, the method is less robust and only applicable in the particular context of a method. In contrast, double-step controle only requires a well defined consitency order of the method.

Problem 2.55: Assume that the ratio of errors coefficients $C^{(p)}[h]$ in the two alternative results of RKF45 is constant. Base a step-size control on this assumption.

4.2.7 Stability

Let us look at the simple linear equation

$$\frac{d}{dt}y = \lambda y. \tag{4.48}$$

with the known exact solution $y(t) = e^{\lambda t}y(0)$.

Assume for now that the method depends linearly on the initial condition

$$M[y_{n-1}, y_n, t, h] = F(\lambda h)y_{n-1} \tag{4.49}$$

Then

$$y_N = F(\lambda h)^N y_0. \tag{4.50}$$

For $\lambda > 0$ we expect as a minimum requirement

$$|y_N| < |y_0|, \tag{4.51}$$

which implies

$$|F(\lambda h)| < 1. \tag{4.52}$$

The **stability region** of a method is the set of those $\mu \in \mathbb{C}$ such that $F(\mu) < 1$.

Why and when stability matters

Let us assume $\frac{d}{dt}\vec{y} = \widehat{A}\vec{y}$ with $\widehat{A}$ having the following spectral representation: $\widehat{A} = \widehat{W}\hat{d}_A\widehat{W}^\dagger$. We want to solve the ODE in the basis of eigenvectors: let $\vec{z} = \widehat{W}^\dagger\vec{y}$; then $\frac{d}{dt}\vec{z} = \hat{d}_A\vec{z}$. This can be written as a set of decoupled equations:

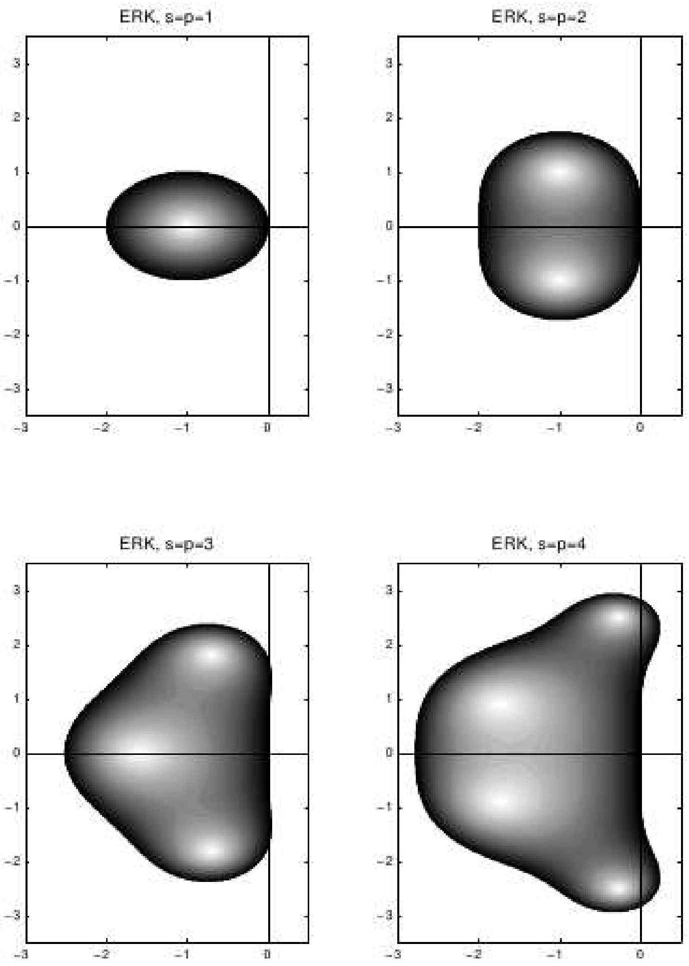

Figure 4.3: Stability regions of explicit Runge-Kutta methods s=p=1,. . . 4.

$$\frac{d}{dt}z_0 \quad = \quad a_0 z_0 \tag{4.53}$$

$$\vdots \tag{4.54}$$

$$\frac{d}{dt}z_{M-1} \quad = \quad a_{M-1} z_{M-1}. \tag{4.55}$$

If all a_n are in the stability region, none of the components will grow exponentially. Assume that the initial vector is $z_0 = (1, 2, 3, 0, \varepsilon_n, 0)^T$, where ε_n is some noise (numerical error). The ε-component will never grow, if a_n is in the stability region. The error remains more or less constant. If not, the component will grow *exponentially* and the solution will break down.

Stability has nothing to do with accuracy. A method that is perfectly stable can produce completely wrong results, if step-sizes are choosen too large. Stability just aserts that these wrong results are not due to (exponential) growth of numerical noise. It assures that tiny numerical errors cannot build up and explode exponentially.

Stability and discretization

In practice, the required stability region is closely linked to discretization. If discretized system has very large eigenvalues, small step sizes are required in explicit methods. If those high eigenvalues reflect rapid change of your actual system, there is probably little you can do about it on the computational front (you may re-consider your modelling, though).

However, often, such high eigenvalues are not of physical interest and arise due to a over-ambitious discretization. For example, very fine grid spacings Δx allow the representations of very high momenta $p_{\max} \sim 2\pi/\Delta x$ and the corresponding high kinetic energies $\sim p_{\max}^2/2$. As a result, you pay a double penalty for small Δx: (1) you need $\propto 1/\Delta x$ more points and (2) you must make $\propto 1/\Delta x^2$ smaller time-steps, if you are using an explicit method, total third-power penalty. Clearly, you should make your grid as fine as needed for representing the dynamics of your system, but *absolutly avoid* too fine grids.

Avoiding unneeded high eigenvalues is a simple advice that can be hard to follow. In that case, implicit, unconditionally stable methods are the remedy.

On the other hand, if high eigenvalues *do* play an actual role in the physics, then switching to a more stable method will not help: while your solution will not explode, it will likely just be incorrect. Stability is only about avoiding exponential growth: it allows you to keep high frequency noise at a low level by “brushing over it”. If you are actually interested in some high frequency-structure, you need to probe it with adequate sampling, i.e. whith evaluation of your derivative at small step sizes $h \sim 2\pi/E_{\max}$.

Summing it up: stability is an important *technical* property. If you have problems with stability, first check whether you can avoid them by a more careful discretization, before you resort to the cumbersome implementation of an implicit method.

Problem 2.56: Investigate the stability range for different methods.

Stability analysis for general ODEs

For linear ODEs and methods we have a linear connection between $\vec{y}_0$ and $\vec{y}_1$:

$$\vec{y}_1 = \widehat{M}\vec{y}_0 \tag{4.56}$$

Then we only need to compute or estimate the largest (in magnitude) eigenvalues of $\widehat{M}$. Clearly, stability of the solution will depend on time, if the ODE is explicitly time-dependent.

For the general non-linear case one locally linearizes the method around some point $\vec{y}_0$

$$\vec{y}_1 \approx \widehat{M}[\vec{y}_0, t]\vec{y}_0 \quad \text{for} \quad |\vec{y}_1 - \vec{y}_0|\,\text{small}. \tag{4.57}$$

Stability is then defined only locally for a given $\vec{y}_0$.

"(Absolute) stability region"

There is an entertaining multitude of names for different aspects of stability, see, e.g.
`http://ode-math.com/StabiltyRegionDefinitions/StabilityRegionDefinitions.html`
Here we remember to keep the problem in mind and that, if we have two methods that are comparably easy to apply and have similar order, we will opt for the one with higher stability. Or, if stability is our key problem, we will use the method with the largest possible stability range.

Note: Implict Runge-Kutta methods are *absolutely stable*, i.e. stable for all negative, real λ

4.2.8 Crank-Nicolson

The Crank Nicolson method is *absolutely stable*, i.e. it is stable for all $\mu \in (-\infty, 0)$. Its consistency and convergence order are 2 and 1, respectively and it is an *implicit* Runge-Kutta method with the following tableau:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

In the case of a linear, time-constant derivative function $f(\vec{y}, t) = \widehat{A}\vec{y}$:

$$(1 - \frac{h}{2}\widehat{A})^{-1}(1 + \frac{h}{2}\widehat{A})\vec{y}_{n-1} = \vec{y}_n. \tag{4.58}$$

Expanding $(1 - \frac{h}{2}\widehat{A})^{-1} = \sum_{n=0}^{\infty}\left(\frac{h}{2}\widehat{A}\right)^n$, one sees that it approximates the exponential $\exp(h\widehat{A})$ to the order $(h\widehat{A})^2$. For using Crank-Nicolson, we need to solve the linear equation for $\vec{y}_n$:

$$(1 - \frac{h}{2}\widehat{A})\vec{y}_n = \vec{z} \quad \text{with} \quad z = (1 + \frac{h}{2}\widehat{A})\vec{y}_{n-1}. \tag{4.59}$$

The method is best applied when solving $1 - h\widehat{A}/2$ is "easy", i.e. for example for banded, tridiagonal (second order FD scheme) matrices $\widehat{A}$. For time-constant $\widehat{A}$ with band-width b, the LU-factorization $\mathcal{O}(Nb^2)$ can be done during setup, solving is then $\mathcal{O}(Nb)$.

Problem 2.57: Implement the Crank-Nicolson method for general linear ODEs

$$\frac{d}{dt}\vec{y} = \widehat{A}\vec{y} \tag{4.60}$$

and investigate its consistency and convergence using $\frac{d}{dt}y = \lambda y$. Write your own code from scratch or add it as a new single step method to `odesingle.py`.

Problem 2.58: Analyze the stability of several single step methods: classical Runge-Kutta, midpoint-rule (see `odesingle.py` for the Butcher tableaus) and Crank-Nicolson by studying analytically $F(\mu)$ for the standard ODE $\frac{d}{dt}y = \lambda y$. Next implement a general numerical test in `odesolver.py` modeled after the "convergence" and "consistency" tests. Confirm you analytical findings.

4.2.9 Split step (split operator) methods

We use this mostly in the linear case of the TDSE, i.e. $id\vec{y}/dt = \widehat{H}\vec{y}$. So we need:

$$U(t) = e^{-it\widehat{H}}. \tag{4.61}$$

Let $\widehat{H} = \widehat{A} + \widehat{B}$, where we can easily form $e^{-it\widehat{A}}$ and $e^{-it\widehat{B}}$. Suppose that we know the spectral representations of $\widehat{A}$ and $\widehat{B}$:

$$\widehat{A} = \widehat{W}_A \hat{d}_A \widehat{W}_A^\dagger, \qquad \widehat{B} = \widehat{W}_B \hat{d}_B \widehat{W}_B^\dagger \tag{4.62}$$

and therefore

$$e^{-i\widehat{X}} = \widehat{W}_X e^{-it\hat{d}_X} \widehat{W}_X^\dagger, \quad \widehat{X} = \widehat{A}, \widehat{B}. \tag{4.63}$$

If $[\widehat{A}, \widehat{B}] \neq 0$

$$e^{-it\widehat{H}} \neq e^{-it\widehat{A}} e^{-it\widehat{B}} \neq e^{-it\widehat{B}} e^{-it\widehat{A}}. \tag{4.64}$$

But

$$e^{-it\widehat{H}} = e^{-it\widehat{A}/2} e^{-it\widehat{B}} e^{-it\widehat{A}/2} + \mathcal{O}(t^3), \tag{4.65}$$

which has consistency order 2 and is unconditionally stable. With $\vec{y} \to \vec{z} = \widehat{W}_A^\dagger \vec{y}$ in the spectral representation of $\widehat{A}$:

$$\vec{z}_n = \underbrace{e^{-it\hat{d}_A/2}}_{\text{easy}} \underbrace{\widehat{W}_{AB}}_{\text{hard}} \underbrace{e^{-it\hat{d}_B}}_{\text{easy}} \underbrace{\widehat{W}_{AB}^\dagger}_{\text{hard}} \underbrace{e^{-it\hat{d}_A/2}}_{\text{easy}} \vec{z}_{n-1}, \tag{4.66}$$

where $\widehat{W}_{AB}$ is a $N \times N$ matrix.

In general, $\widehat{W}_{AB}$ is a huge matrix, with a cost of $\mathcal{O}(N^2)$ operations to apply. In some cases, it can be a much smaller matrix or some other efficient possibility to transform from the spectral representation of $\widehat{A}$ to that of $\widehat{B}$.

Example: Suppose that $\widehat{A} = p^2/2$ and $\widehat{B} = V(x)$. Discretize the momentum and kinetic energy on an equidistant grid in x-space. Then $\widehat{B}$ is diagonal and $p^2/2$ is diagonal on the corresponding k-grid. Then we do the transformation between position and momentum representation by FFT and get a order of $\mathcal{O}(N \log N)$ instead of $\mathcal{O}(N^2)$. Suppose that $N = 1000$ then $log_2 N = 10$. Instead of a 1000×1000 matrix-vector multiplication with operations count 1000^2 we get the 1000×10 operation of FFT. This is a crucial difference (factor 100).

For *time-dependent problems*, i.e. $f(\vec{y}, t)$ explicitly depends on t, the Runge-Kutta method is formulated in full generality. The *split step* method assumes time-constant $\widehat{A}, \widehat{B}$. For time-dependent $\widehat{A}$, taking $\widehat{A}(t_0)$ of $t_0 = t + h/2$ (in the middle of the time-step interval) leads to consistency order 2 also for the time-dependent problem.

Problem 2.59: Grid methods: Discretize $\Psi_\pm(x_i)$ on an equidistant grid $x_n = -L + n\frac{2L}{N}$, $n = 0, 1, \ldots, N-1$. Use periodic boundary conditions $\Psi(x_N) = \Psi(x_0)$. Implement derivatives by 2nd and 4th order finite difference schemes. Check convergence and accuracy. Compare performance to the FE implementation.

Trotter-Suzuki expansion

Split step is unconditionally stable (if the exponentiations can be done exactly), but it has consistency order of only 2. The basic idea of split step can be generalized to repeated exponentiations with the goal of increasing consistency order. This type of expansions is called Trotter-Suzuki.

Magnus expansion

For time-dependent problems, often the so-called Magnus expansion can be useful. Approximate the abstract time-ordered exponential $T[\exp \int_{t_0}^{t} \ldots]$ up to $\mathcal{O}(|t_0 - t|^n)$ by a simple exponential of a new operator $\Omega^{(n)}$:

$$U(t_0, t) := T \exp\left[-i \int_{t_0}^{t} \widehat{H}(\tau) d\tau\right] = \exp[-i\Omega^{(n)}(t_0, t)] + \mathcal{O}(|t - t_0|^{n+1}) \tag{4.67}$$

One finds

$$\Omega^{(2)}(t, t_0) = \int_{t_0}^{t} dt_1 \widehat{H}(t_1) \tag{4.68}$$

$$\Omega^{(4)}(t, t_0) = \Omega(t_0, t)^{(2)} + \frac{1}{2} \int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \left[\widehat{H}(t_1), \widehat{H}(t_2)\right] \tag{4.69}$$

$$\Omega^{(6)}(t, t_0) = \Omega(t_0, t)^{(4)} + \frac{1}{2} \int_{t_0}^{t} dt_1 \int_{t_0}^{t_1} dt_2 \int_{t_0}^{t_2} dt_3 \left[\widehat{H}(t_1), \left[\widehat{H}(t_2), \widehat{H}(t_3)\right]\right] + \left[\widehat{H}(t_3), \left[\widehat{H}(t_2), \widehat{H}(t_1)\right]\right] \tag{4.70}$$

$$\vdots \tag{4.71}$$

This is less scary than what it looks when the time-dependence has the simple form, as often in physics:

$$\widehat{H}(t) = \widehat{A} + f(t)\widehat{B} : \tag{4.72}$$

only integrals over $f(t)$ will be needed. The commutator can applied explicitly

$$[\widehat{H}(t_1),\widehat{H}(t_2)]\vec{c}=[f(t_1)-f(t_2)][\widehat{A},\widehat{B}]\vec{c}=[f(t_1)-f(t_2)](\widehat{A}\widehat{B}-\widehat{B}\widehat{A})\vec{c}. \tag{4.73}$$

Often, the commutator takes very simple forms like when the time-dependent part of the Hamiltonian is the dipole operator

$$[-\frac{1}{2}\Delta+V(\vec{r}),\vec{r}]=-\frac{1}{2}\vec{\nabla}. \tag{4.74}$$

The same principle can also be applied in classicals dynamics (symplectic integrators, to be discussed later). We see that Magnus $\Omega^{(2)}$ slightly improves split-step for time-dependent problems, but $\Omega^{(4)}$ crucially increases consistencey order.

Non-linear problems

Non-linear equations can be tackled by a similiar strategy as time-dependent problems: do a piece-wise linearization, i.e. assume that, on short time-intervals, only changes up to the first order matter. An example for applying this stratege is given below, solving the “Non-linear Schrödinger equation”, an equation that arrises in optics and also for the description of Bose-Einstein condensates. With that strategy, e.g. the split-step method again gives consistency order 2.

4.2.10 A TDSE solver

(Implemented in 1d only)

- Read and set up your discretization (class Axis)
- Set up your operator (class QMOperator)
- Create your initial wave function $\psi(t=0)$ (class WaveFunction): e.g. lowest eigenstate (from QMOperator), gaussian wave packet (method of class WaveFunction),. . .
- read your time-grid, i.e. the points in time where you want to see the solution.
- set up your solver (class OdeSolver): classical runge kutta, others...
- step through the time grid and store the solutions
- use the solutions $\psi(t)$: e.g. display (should be a method of WaveFunction, not implemented yet)

Improvements in 1d:

- Right now, the operator is converted to a full matrix and that matrix is multiplied on the coefficient vector: operations count $N\times N$ vs. $N\times b$ if band structure is exploited.
- Also, the inverse overlap matrix is computed explicitly (always a full matrix). Better is to store its LU decomposition and apply: again $N\times N$ vs. $N\times b$

Improvements for 2d:

- Extend QMOperator and WaveFunction must be extended to 2d by including 2 axes.
- If explicit solvers are used, matrix vector multiplication is crucial: exploit to the maximal possible extend the tensor product form of the matrices: MUCH lower operations count.

Improve performance: multiplying vectors by tensor products

Let us use double indexing for the tensor product matrices: $\widehat{A}$ and $\widehat{B}$ are $M \times M$ and $N \times N$, respectively.

$$\widehat{T} = \widehat{A} \otimes \widehat{B} \tag{4.75}$$

means

$$(\widehat{T})_{mn,m'n'} = A_{mm'} B_{nn'} \tag{4.76}$$

Let $\vec{c}$ be a vector in the tensor product space $R^M \otimes R^N$ with its components labelled by $c_{m'n'}$. Then

$$(\widehat{T}\vec{c})_{mn} = \sum_{m'n'} (\widehat{T})_{mn,m'n'} c_{m'n'} = \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} (\widehat{T})_{mn,m'n'} c_{m'n'} \tag{4.77}$$

requires $(MN)^2$ floating multiplications and additions. However, the special shape of $\widehat{T}$ as a tensor product allows the same operator with a much lower operations count:

$$\sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} (\widehat{T})_{mn,m'n'} c_{m'n'} = \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} \widehat{A}_{mm'} \widehat{B}_{n,n'} c_{m'n'} = \sum_{m'=0}^{M-1} \widehat{A}_{mm'} \sum_{n'=0}^{N-1} \widehat{B}_{n,n'} c_{m'n'} \tag{4.78}$$

which can be writen of two sets of small matrix-matrix mutliplications: re-arrange $\vec{c}$ as a $M \times N$ matrix $\widehat{C} : c_{m'n'} = (\widehat{C})_{m'n'}$

$$\widehat{X} = \widehat{C}\widehat{B}^T \tag{4.79}$$

with operations count MN^2 and

$$\widehat{Y} = \widehat{A}\widehat{X}, \quad (\widehat{Y})_{mn} = (\widehat{T}\vec{c})_{mn} \tag{4.80}$$

with operations count M^2N, giving at total operations count $M^2N + MN^2 \ll (MN)^2$ if $M \gg 1$ and $N \gg 1$.

Except for conceptional clarity, this crucial advantage of tensor products is why we introduced them in the first place.

4.2.11 The Finite-Difference Time-Domain (FDTD) method

A very large fraction of modern simulations of Maxwell's equation is done using the so-called "Finite Difference Time-Domain" method. Its key properties are

- Simple setup of the discretization of Maxwell's equation

- Consistency order 2 in space and time
- Stability of time-propagation
- Easy inclusion of material properties, like susceptibility
- Good parallelizability

Main drawback is a potentially rather small time-step and grid spacing needed for the representation of high frequencies. Time-step and spatial resolution are linked by the so-called "Courant-condition", basically saying that in units where the speed of light is =1, $\Delta t \leq \Delta X$: if we decrease grid spacing for better spatial resolution, we need to correspondingly decrease the the time-step. For the Maxwell equations in three spatial dimension, the computing time therefore grows as ΔX^{-4}, the memory grows as ΔX^{-3}.

We will discuss here only the situation where symmetry allows to reduce the problem to a single dimension, i.e. plane wave solution. We will excite these plane waves from a time-dependent current. Maxwell's equations are, in suitable units

$$\vec{\nabla} \times \vec{E} = -\frac{d}{dt}\vec{B}, \qquad \vec{\nabla} \cdot \vec{E} = \rho \tag{4.81}$$

$$\vec{\nabla} \times \vec{B} = \frac{d}{dt}\vec{E} + \vec{j}, \qquad \vec{\nabla} \cdot \vec{B} = 0, \tag{4.82}$$

If we assume that our system is translationally invariant in y and z-directions, and if we are interested in solutions that share this symmetry (remember that a symmetric system does allow asymmetric solutions), we can decide for polarization direction of $\vec{E} = (0, E(x,t), 0)$ and correspondingly $\vec{B} = (0, 0, B(x,t)$ and solve the equations

$$\begin{aligned} \partial_x E &= -\partial_t B && (4.83) \\ -\partial_x B &= \partial_t E + J(x,t). && (4.84) \end{aligned}$$

We will use a forced external current $J(x,t)$ as an "antenna" emitting the radiation that propagates. For now, we assume the propagation occurs in vacuum, without dispersion or other effects of matter.

As a side-remark, note that the equation can be written in the form

$$\begin{pmatrix} \partial_x & 0 \\ 0 & -\partial_x \end{pmatrix} \begin{pmatrix} E \\ B \end{pmatrix} = \frac{d}{dt} \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} E \\ B \end{pmatrix} + \begin{pmatrix} J \\ 0 \end{pmatrix} \tag{4.85}$$

or

$$\widehat{D}\vec{\Phi} = \frac{d}{dt}\widehat{S}\vec{\Phi} + \vec{J}, \tag{4.86}$$

essentially the same structure that we have been dealing with much of the time. The 3-dimensional Maxwell equations can be written in the same form with 6-component vectors $\vec{\Phi} = (\vec{E}, \vec{B})$. In principle, we can use any combination of discretization (in space) and solvers (in time) to numerically solve the equation.

Indeed, FDTD is a special discretization/solver pair. There is one peculiarity about the matrix $\widehat{S}$: while so far we had it mostly positive definite, $\widehat{S}$ is indefinite, but, of course, invertible. More

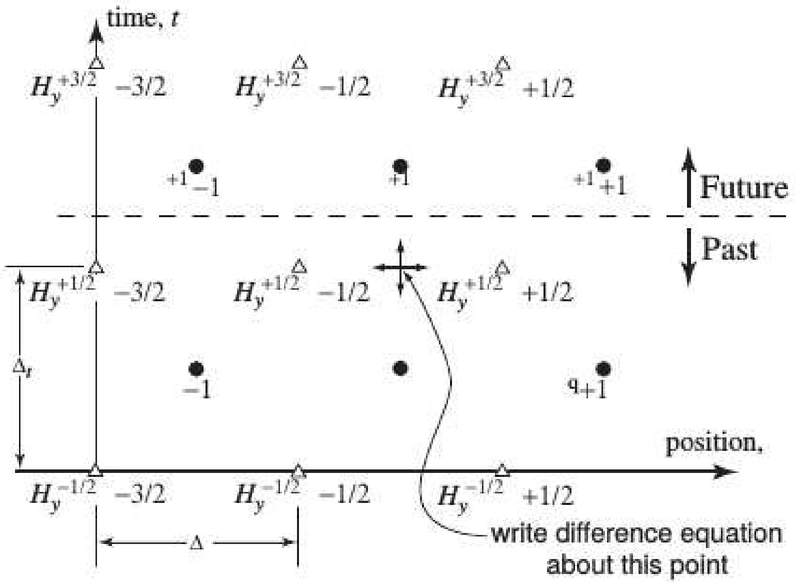

Figure 4.4: Illustation of the FDTD "staggered" space-time grid. Triangles carry the magnetic field, bullets the electric field. (from "Understanding the FDTD Method" by John B. Schneider, available online at the time of writing).

precisely, it is a "symplectic" matrix that we will briefly encounter when discussing classical mechanics simulations, see discussion there. Symplectic $\widehat{S}$ allow a so-called symplectic time-integrators. The simplest of these is the "leap-frog" algorithm, which serves as the time-integrator in FDTD. For the spatial discretization, an equidistant grid with 2nd order FD is used. The only little extra trick is the following: for computing the time-derivative $\partial_t B(x_i, t)$ we need (in 2nd order FD) only $E(x_{i-1}, t)$ and $E(x_{i+1}, t)$, for $\partial_t E(x_{i\pm1}, t)$ we only need $B(x_i, t), B(x_{i+2}, t)$ and $B(x_i, t), B(x_{i-2}, t)$, respectively. The even and the odd grid points never mix! It is just two completely separate problems, and we need to solve only one of them, which reduces the number of grid points by a factor of 2. From now on, we write $E(x_{2i}, t_{2n}) = E_{in}$ and $B(x_{2i+1}, t_{2n+1}) = B_{in}$.

The "leap-frog" procedure for time-integration in this case reads

$$B_{i,n+1} = B_{i,n} + \Delta t(E_{i+1,n} - E_{i-1,n}) + J_{i,n} \tag{4.87}$$

$$E_{i,n+1} = E_{i,n} + \Delta t(B_{i+1,n+1} - B_{i-1,n+1}) \tag{4.88}$$

In the original grid, each grid point x_i carries E and B. After omission of all $E(x_i, t_n)$ at, say, odd i, and corresponding omission of all $B(x_i, t_n)$ at the even i we have twice the spacing. The leap-frog method can be interpreted in a similar way. As a resutlt, we have two space-time grids offset relative to each other by 1/2 grid spacing and time-step: in FDTD jargon, "staggered" grids.

Problem 2.60: Set up a FDTD code. For the grid, choose periodic boundary conditions. Initially all fields = 0. Use a time-dependent current a the origin $J_{in} = \delta_{i0} \sin^2 \pi n\Delta t/T \sin(\omega n\Delta t)$. Plot the current and fields as functions of time. Verify your result by comparing to the analytical solution. How are grid-spacing and frequency ω related. What happens for grid spacings $\Delta X \gtrsim 2\pi/\omega$? What happens when $\Delta t \gtrsim \Delta X$? (Remember, our "speed of light" is =1).

Bibliography

[1] G. Reider, “Photonik”, (1997), Springer, Wien/New York.

[2] M. Born and E. Wolf, “Principles of Optics” (1999), Cambridge University Press, Cambridge.

Chapter 5

Monte Carlo methods

5.1 Introduction

If a system is too complex to "solve" it, i.e. produce the complete desired information about it with well-defined accuracy, we can try to map it partially, by systematically probing it. In order to obtain unbiased information from such a probing, we take the probes "randomly", i.e. without any assumptions about the system. This is the most abstract description of Monte Carlo methods. The name indeed refers to the city of Monte Carlo — synomymous with gambling — and the idea of completely random sequencies of numbers that should occur in, e.g., roulette.

Provided we have a source of "random" numbers, sanity of results, accuracy estimates, and convergence can be tested using the probability theory. The definition of randomness is a negative one: as set of numbers in a certain interval without any structure. The number of conceivable "structures" is unlimited, but actual random numbers are can only be tested for a finite set of structures: in its general form, the concept remains logically incomplete. This is reflected in computational practice: scrutinize numbers for a set of possible hidden structures until you are convinced that there are no structures that would compromise your result.

The randomness needs to be adhered only to the extent that indeed we do not have *a priori* knowledge of our system. Where we *do* have such knowledge, we must make all possible efforts to take advantage of it. For example, we may approximately know where a funktion is large and therefore will contribute most to an integral (even if we cannot give the exact form of the integral). Or we may know a criterion for the regions in phase-space that are more important than others. We then can introduce biased probing based on such additional knowledge and take the bias propertly into account. Much of the art of Monte-Carlo methods is in the skillful use of knowledge about your problem.

We will discuss two examples for the application of Monte-Carlo methods:

- Monte-Carlo integration
- The Metropolis algorithm and its derivatives to solve the Ising model: $H = \sum_{i,j=0}^{N-1} J_{ij}s_i s_j + B\sum_{i=0}^{N-1} s_i,\ s_i = \pm 1$. The Ising model describes a set of spins interacting with each other and with an external magnetic field B. Loosely speaking it is the harmonic oscillator of thermodynamics: there are several exact results known, against which we can test our numerics (or any

other method). The Hilbert space $\mathcal{H}$ for such a system is $\mathcal{H} = \underbrace{S_2 \otimes \ldots \otimes S_2}_{\text{N times}}$ where S_2 is the Hilbert space for one particle. As s_i can only assume two different values the total space has dimension $\dim(\mathcal{H}) = 2^N$.

5.2 Monte Carlo (MC) integration

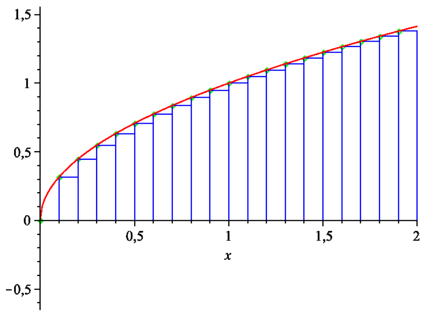

Figure 5.1: Graphical definition of a definite integral. The x-axis is divided into small parts of length h_i and all the boxes that are obtained using such a procedure are added up in the end.

The most basic (and intuitive) definition of integral is the Riemann integral (figure 5.1). The step to follow in order to "obtain" an integral are:

- add up the areas of the boxes;
- take box width to zero;
- hope it converges.

This can be formalized saying that the value of the integral I is given by $I = \sum_i h_i f(x_i)$.

For a fixed width h we have:

$$I = \frac{b-a}{N} \sum_{i=0}^{N-1} f(x_i) \overset{N\to\infty}{\longrightarrow} \int_a^b f(x)dx \tag{5.1}$$

A Monte Carlo integral over the interval $[a, b]$ is defined as:

$$\lim_{N\to\infty} \frac{b-a}{N} \sum_{i=0}^{N-1} f(x_i) \tag{5.2}$$

where x_i are *randomly* chosen points. The fact that x_i is chosen randomly is particular important: if there is any structure in the choice of x_i one risks to get irregular convergence.

The basic problem can be seen already when one applies the approach to the definite integral of a simple function of a single variable, e.g. $\int_0^{100} dt \sin 2\pi t$: if we choose t_i randomly, but only from a limited set of, say, integer numbers, we will get the correct result =0. If we happen to get it from the set $n + 1/4$, we obtain an incorrect result. This may appear to be a silly example, but we must remember, that any number on the computer has a finite number of digits, the number on a computer map uniquely onto the integer numbers: it is just a matter of resolution until we, at a finite number digits, can run into accuracy problems of this kind.

Error of MC integration

Even with "true" random numbers, the error of any MC integration remains finite and decreases comparatively slowly as $\sim N^{-1/2}$ with the number N of probes taken.

Let our desired integral be

$$I = \int_V dx f(x). \tag{5.3}$$

To see what determines the error, let us assume that we probe a function by N points. The individual function value at a random point x_i is itself a random number f_i. Similarly, the integral estimate $I_N(x_0, x_1, \dots, x_{N-1}) = V/N \sum_{i=0}^{N-1} f(x_i)$ is a random variable. Although we have no easy access to the distribution $\rho(f_i)$ of the $f(x_i)$, by the central limit theorem we know that the distribution of $\rho(I_N) = \prod_{i=0}^{N-1} \rho(f_i)$ approach a Gaussian with expectation value I. The variance of I_N is

$$\sigma^2(I_N) = \frac{1}{V^N} \int_V dx_0 \dots \int_V dx_{N-1} \left[\left(\frac{V}{N} \sum_{i=0}^{N-1} f(x_i) \right) - I \right]^2 = \frac{V}{N} \int_V dx \, (f(x) - \langle f \rangle)^2 = \frac{V}{N} \sigma^2(f), \tag{5.4}$$

where we have written

$$I = \frac{V}{N} \sum_{i=0}^{N-1} \frac{I}{V} = \frac{V}{N} \sum_{i=0}^{N-1} \langle f \rangle. \tag{5.5}$$

As the distribution of the I_N is approximately Gaussian, the $\pm\sigma(I_N)$ define the 68.2% confidence interval for our estimate. As one expects, the confidence interval narrows $\propto N^{-1/2}$. It is important to observe that it is proportional to the variance $\sigma(f)$ of the integrand f. Clearly, the error also grows with the volume V that needs to be sampled.

If the variance σ_f is small, reasonably accurate values of the integral can be obtained quickly (extreme case: constant function). Of course we do not know σ_f, but we can estimate it by approximating $\int_V f^2(x) dx$ within the same MC simulation.

Note that the *rate of convergence* is always the same $1/\sqrt{N}$, i.e. rather slow: for 10 times better accuracy, you need 100 times more function evaluations.

5.2.1 Reasons to use MC integration

The convergence $\sim N^{-1/2}$ appears rather miserable compared to, e.g., the exponential convergence of Gauss quadratures with the number of quadrature points. However, there are no assumptions about differntiability of the integrand, as in high order quadratures. Most importantly, convergence of MC integration is independent of dimensions of the integral, only depending on its variance.

Assume a quadrature that converges $\sim h^k$ in one dimension. With a total of N points in d dimension, we have $\sim N^{-d}$ points for each coordinate and a convergence $\sim N^{-k/d}$ for the quadrature. So unless we can use very high order quadratures, the convergence of MC integration does not look that bad. But high order quadratures are rarely applicable for irregular and poorly differentiable integrands.

Another reason is its conceptual simplicity, at least when taken in its simplest form given here.

Comparison of MC integration with Gauss quadratures

This comparison can only be performed, when you make some assumptions about the structure of your integrand.

Case 1 - "smooth" integral - extreme case**:** All higher derivatives are uniformly bounded, i.e. there exists a constant C such that all higher partial derivatives are bounded by it:

$$\left[\prod_{i\in T}\partial_{x_i}\right] f(x_0,x_1,\ldots,x_{M-1}) \le C \tag{5.6}$$

where all possible partial derivatives T are an N-tuple:

$$T = (\underbrace{0,0,\ldots,0}_{n_0\text{ times}},\underbrace{1,1,\ldots,1}_{n_1\text{ times}},\ldots\underbrace{M-1,M-1,\ldots,M-1}_{n_{M-1}\text{ times}}), \qquad \sum_{i=0}^{M-1} n_i = N. \tag{5.7}$$

Note that, like in the 1-d case, sometimes we may succeed to extract non-analytic behavior (say, $\sim (x)^\alpha$ for α not a natural number) by choosing a proper weight function for the Gauss quadrature.

This is the case, where we can increase the *order* of a Gauss quadrature to (exponentially) increase its accuracy. To see this, think of a Taylor series that breaks off at the N'total order.

In this case, if any accuracy is needed at all and if the function has any variance, the exponential convergence of the quadrature will be hard to beat.

Case 2: The higher derivatives of the integrand are not bounded. One can only use a fixed order quadrature and obtains a convergence which is proportional to $\sim h^p$. But decreasing h in M dimensions, means the number of function evaluations increases by $N \propto h^{-M}$. Comparing the two errors, we find

$$\delta I_{MC} \propto N^{-1/2}, \qquad \delta I_{quad} \propto h^p \propto N^{-p/M} \quad \text{for}\, p > 2M. \tag{5.8}$$

For moderate dimensions M, depending on our integrand, we should always be able to improve the *convergence behavior* for the quadrature to beat Monte Carlo

Case 3 - **very** irregular integration volume V**:** Doing multi-dimensional integrals by quadratures becomes difficult, when the integral volume is not a rectangle. (To get a **high order** for the quadrature you need **high** oder approximation for the **integral boundaries**. This can be very difficult for highly irregular shapes.) One can try a coordinate transformation to map V onto a rectangle. For example, when integrating over a sphere of radius R, the polar coordinates map it into the rectangle $(r,\theta,\phi) \in [0,R]\times[-\pi/2,\pi/2]\times[0,2\pi]$. But exactly that will not be possible in general. This is where high order quadratures become difficult to construct and where Monte Carlo has the decisive edge.

Problem 2.61: Create a class for Monte-Carlo integration in one dimension. Include a test routine to prove its correctness by comparison of the Monte-Carlo result with the exact integral. Plot the MC value as N increases, show that the errors tend to decrease as $N^{-1/2}$. Clearly, you will not obtain exactly the a decrease by $N^{-1/2}$. For the test routine, find a criterion to accept or reject the decrease as correct or reject as incorrect.

Problem 2.62: Keeping the number of integration points N fixed, repeatedly compute the integral and show how it scatters around the true value. Sort the results into a histogram, i.e. the counts that you find in intervals around the true value. For large N (and sufficiently narrow bins of the histogram), the distribution should resemble a Gaussian. Fit a Gaussian to the distribution that you obtain by manually adjusting its width. Compare the width of your distribution to the variance of your function with respect to the integral.

5.3 Random numbers

The use of probabilistic reasoning requires strictly random numbers. If there are some systematic patterns in the x_i we use, the error estimates and convergence patterns are not valid. A somewhat silly example would be probing of a periodic function, say $\sin x$ with numbers that are spaced at period 2π: clearly, the result can be anything between $\pm L$, if L is the integration range.

5.3.1 What is random? — Test your random numbers!

Randomness is defined negatively: there is no pattern of any kind in the sequence of a set of random numbers. To really obtain random number one must:

- use a physical process that we believe is random;
- do not use random numbers, but use numbers that are generated by an algorithm. The algorithm is “the pattern”.

Suppose we want to have random numbers evenly distributed in [0,1]. Evenly distributed means if we count how many numbers fall into any bin of fixed size $[a, a+h] \in [0,1]$, for sufficiently many numbers, any such bin will contain the same number of numbers. Clearly, the numbers $i/N, i = 0, 1, \ldots, N-1$ are evenly distributed, but have a very simple pattern. Next you need to systematically exclude any possible pattern from your set of numbers to decide whether they are “random”. As (probably) there is an infinite number of possible “patterns”, randomness in that sense is not a operationally well-defined concept.

In particular, any set of numbers generated by an *algorithm*, is by definition NOT random: the algorithm is its “pattern”. There are attempts to use physical processes that we believe are “random”, to generated random numbers. That is fine (if the process *really* is “random”), but not practical: it also means that two runs of a code will never produce the exactly same result which is a nightmare for debugging!

The solution are **pseudo-random numbers** that are generated by some algorithm and thus **not** random by definition. But nevertheless in practice we prefer them over truly random numbers because they give **reproducible results**. that must obey certain criteria, like, truly independent numbers, when binned, must be Poisson-distributed, or, the difference between two subsequent numbers must be random, or…

A suite of tests that the random numbers should pass, has been worked out by different places. A good reference in such matters is NIST — the US "National Institute of Standards". A test suit including their and other tests is provided under Gnu Public Licence (GPL) as the "**dieharder**" test suit (google it!).

Problem 3.63: Download the `dieharder` test suite for random generators. Select a few of the tests and discuss their idea. Find critical discussion of the selected tests: where would a given test fail to detect lack of "randomness"?

Here are some examples of the provided tests (see documentation, quotes from Wikipedia):

- **Birthday spacing test**: Choose a random number of points on a large interval. The spacings between the points should be asymptotically exponentially distributed. The name is based on the birthday paradox.
- **Overlapping permutations**: Analyze sequences of five consecutive random numbers. The 120 possible orderings should occur with statistically equal probability.
- Ranks of matrices: Select some number of bits from some number of random numbers to form a matrix over {0,1}, then determine the rank of the matrix. Count the ranks.
- Count the 1s: Count the 1 bits in each of either successive or chosen bytes. Convert the counts to "letters", and count the occurrences of five-letter "words".
- Parking lot test: Randomly place unit circles in a 100 x 100 square. If the circle overlaps an existing one, try again. After 12,000 tries, the number of successfully "parked" circles should follow a certain normal distribution.
- Minimum distance test: Randomly place 8,000 points in a 10,000 x 10,000 square, then find the minimum distance between the pairs. The square of this distance should be exponentially distributed with a certain mean.
- etc…

Note: If your "random" numbers produce a highly unlikely result, you should be suspicious of your numbers generated.

5.3.2 Pseudo-random Number Generator (RNG)

LCG — Linear Congruence Generators

Although operating systems, compilers or languages, all provide random number generators, it is a good idea to have your own, such that your algorithm works the same way under all circumstances.

One way to get a linear pseudo-random number generator is to consider the following general sequence:

$$x_{n+1} = ax_n + c \mod M \tag{5.9}$$

with x_n, a, c and M being integers. This produces uniformly distributed numbers in the interval $0, 1, \dots, M-1$ and all numbers occur if and only if:

1. c and M do not have common factors (except for 1).
2. for any factor q of M, $a \mod q = 1$.
3. if $a \mod 4 = 1$, then $M \mod 4 = 0$

These are **minimal requirements**, but they do **not guarantee** "good" random numbers.

Problem 3.64: Historically, a famous example for a bad random number generator was provided by "randu" by IBM, where a sequence of x_n was generated by the prescription:

$$x_{n+1} = 65539 x_n \mod 2^{31}. \tag{5.10}$$

Run this through the "dieharder" test suit to see how it performs.

Also, plot subsequent triples of the numbers (scaled into [0,1]) as dots in 3d into $[0,1]\times[0,1]\times[0,1]$, and you will see a nice pattern. (Note that it violates condition 3.)

Run the `numpy.random.rand` generator through `dieharder`. Inteprete the results.

5.3.3 Multiple Recursive Generators

A decent class of algorithms are "Multiple Recursive Generators" (MRG) defined as:

$$x_n = \left(\sum_{i=1}^{K} a_i x_{n-i}\right) \mod M. \tag{5.11}$$

Note: Note that you generate random numbers in the interval $[0, M-1]$. To get random numbers in the interval $[0,1)$ just perform: $x_n/M = y_n \in [0,1)$. Instead of just using one preceding value, the pseudo random number is generated from the K pseudo random numbers preceding it.

Problem 3.65: Test the MRG:

$$x_n = (177786 x_{n-1} + 64654 x_{n-5}) \mod (2^3 1 - 1) \tag{5.12}$$

using the "dieharder" test suit.

5.3.4 Speeding up convergence

While the fundamental convergence $\propto N^{-1/2}$ cannot be changed, we can make an effort to reduce the proportionality factor, essentially the variance of the integrand.

Variance reduction

Find a comparison function $g(x)$ such that the variance $\sigma_{f-g}^2 \ll \sigma_f^2$ and that the integral

$$\int_a^b dx g(x) = J \tag{5.13}$$

can be calculated efficiently.

Importance sampling

Find a (non-negative) function $0 < w(x)$ such that in

$$\int_a^b dx w(x) \frac{f(x)}{w(x)} =: \int_a^b dx w(x) g(x). \tag{5.14}$$

the integrand $g(x) = f(x)/w(x)$ should have little variance (i.e. $\sigma_g \ll \sigma_f$). Then, rather than using uniformly distributed points, generate random points x according to the "probability density" $w(x)$. Let $x_i^{(w)}$ be random numbers distributed according to $w(x)$, then

$$\int dx w(x) g(x) \approx \frac{W}{N} \sum_{j=0}^{N-1} g(x_j^{(w)}). \tag{5.15}$$

Note that the integration volume V must be re-normalized to $W = \int_a^b dx w(x)$: this is easy to see, assuming the case $w(x) = f(x), g \equiv 1$. As the random variables $g_j = g(x_j^{(w)})$ have small variance, the sum will converge more rapidly. The fact that we have omitted $w(x)$ from the summands is compensated by the fact that $x_j^{(w)}$ appear more frequently at values x where $w(x)$ is large. In that way, the function is probed, where it is important: for reducing variance, $w(x)$ should have large values where $f(x)$ has large values.

5.3.5 Distributions according to a probability density function

Distribution by inversion

The example above of speeding up integration by importance sampling also tells us how to generate random numbers in an interval $[a, b]$ according to a preset (positive definite) probability density $w(x)$. We can simply change integration variable

$$\int_a^b dx w(x) g(x) = \int_0^W dy(x) g(x(y)) \tag{5.16}$$

with

$$dy(x) = w(x)dx, \quad dy/dx = w(x), \quad y(x) = \int_a^x w(x')dx', \quad W = y(b). \tag{5.17}$$

As $w(x) > 0$, the function $y(x)$ is invertible. In this new variable y, the MC approximation to the integral is

$$\int_0^W dy g(x(y)) \approx \frac{W}{N} \sum_{j=0}^{N-1} g(x(y_j)). \tag{5.18}$$

We identify the random points according to a distribution $w(x)$ as

$$x_j^{(w)} = x(y_j), \tag{5.19}$$

where the y_j are equally distributed in $[0, 1]$.

Problem 3.66: Let $w(\xi)$ be a probability density on $[a, b]$ and define $W(x) := \int_a^x d\xi w(\xi)$. Express the graphical arguments given in the lecture in terms of calculus to show that the $W^{-1}(y_i)$ are distributed according to the probability density $w(x)$, if y_i are equally distributed on $[0, 1]$.

Apart from this proof by simple calculus, it may be useful to also have a geometric picture of why one obtains the desired distribution (see figure 5.2): Draw the function $W(x)$. As $0 < w(x) = \partial_x W(x)$, finite interval Δy on the y-axis is mapped into a finite interval Δx on the x-axis. The ratio of the width of the intervals is

$$\Delta y \approx \partial_x W(x) \Delta x = w(x) \Delta x \tag{5.20}$$

So, the interval Δy is mapped onto an interval Δx, which shrinks with increasing derivative $\partial_x W(x) = w(x)$. If we have points y_i with a constant density on the y-axis, the density on the intervals Δx on the x-axis is enhanced proportional to $w(x)$.

Example: (piecewise) linear distribution function on $[a, b] = [0, 1]$:

$$w(\xi) = 8\xi/5 \text{ for } \xi < 1/2, (16\xi - 4)/5 \text{ for } \xi > 1/2 \tag{5.21}$$

$$W(x) = 8x^2/10 \text{ for } x < 1/2, (8x^2 - 4x + 1)/5 \text{ for } x > 1/2 \tag{5.22}$$

The equation $W(x) = y$ can be solved for any y in $[0, 1]$.

von Neumann rejection

Suppose you have $w(x)$ on $[a, b]$. You want to generate more points when the value of the function is higher. Consider the following scheme (see also figure 5.3):

- put a box around $w(x)$;
- randomly select points (x_i,y_i) from that box;
- use x_i only if $y_i \leq w(x_i)$;
- x_i will be distributed according to $w(x)$.

The numbers of points in a given bin $[x_0, x_0 + \Delta x] \sim$ surface under $w(x)$.

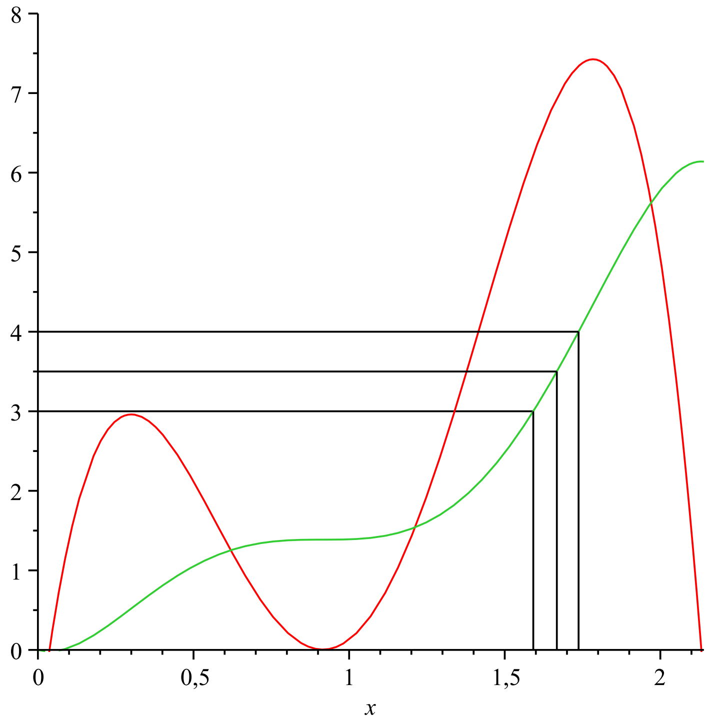

Figure 5.2: Graphical representation of the inversion method. The red curve shows a (not normalized distribution $w(x)$ and the green curve represents its primitive $W(x)$. It can easily be seen that the points on the y-axis are mapped into a smaller interval into the x-axis.

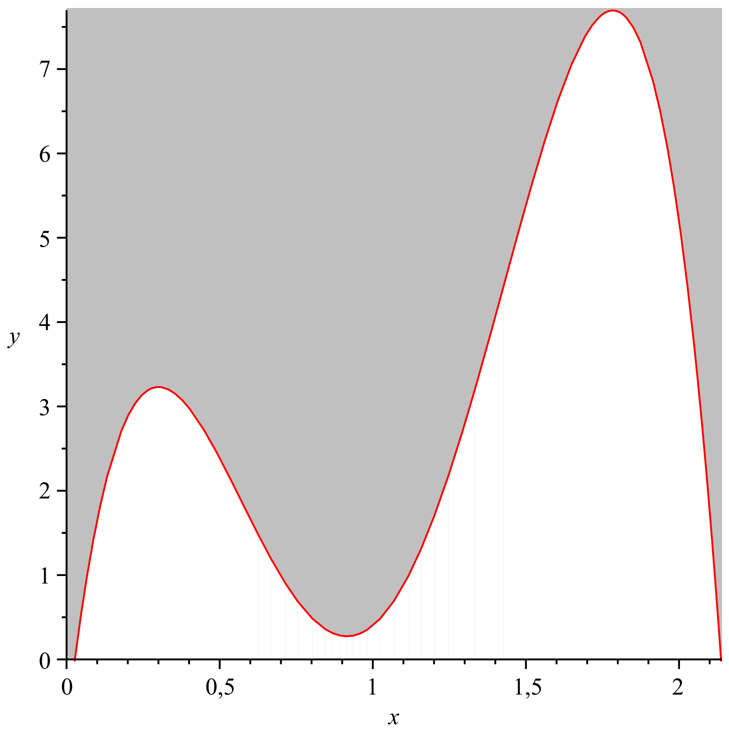

Figure 5.3: In the Von Neumann rejection method we put a box around a distribution (represented by a red curve) and, after we randomly select some points, we throw away all the points that are in the grey region. The remaining points will be distributed according to the desired distribution.

Combination of von Neumann and inversion methods — importance sampling reloaded

Suppose you have some probablity distribution $f(x)$, whose integral you can either not form or not invert. But assume also, that you know another distribution function $g(x)$ for which you can generate a distribution easily (by inversion), and for which $f(x) \leq g(x)$. (Remark: for $g(x) > f(x)$ is not a *probability* distribution in the strict sense, as $\int dx g(x) > \int dx f(x) = 1$.)

Then you can generate a distribution according to $f(x)$ as follows

- generate x according to distribution g.
- generate a random number y in $[0, g(x)]$
- if $y \leq f(x)$ accept x as your random point x_i
- else reject

The thus generated x_i will be distributed according to $f(x)$.

Problem 3.67: Discuss the gain of combining importance sampling with von Neumann rejection. Assign a cost (in terms of CPU time) of ϕ for the evaluation of $f(x)$, a cost ρ for generating the equally distributed random numbers y, and cost γ for evaluating the $W^{-1}(y) = x$. Express the gain in CPU time over direct von Neumann rejection in terms of ϕ, ρ, γ and the exact integrals over $f(x)$ and the importance sampling function $g(x)$. As an example, compute the integral $\int_0^1 dx \frac{1}{\sqrt{1-x^2}}$ by combining von Neumann rejection and importance sampling with the sampling function $g(x) = \frac{1}{\sqrt{1-x}}$.

5.3.6 Learning by doing

So far, we have used the function values that we evaluated solely to throw them into integral-pot and then forget them. By this, we have been throwing away precious information! After all, as we probe the function, we get it to know, we see where it tends to be large, we see where it has large variance. This knowledge is all all in some statistical, approximate sense. One can use this experience to guide the further MC process. For example, one can make one random run through the function. In the process, one notes the areas where one obtains large values or intervals where the values add up to large numbers. In the subsequent run, one probes these areas more accurately.

One can realize this with a recursive splitting of the integration volume (as we did with quadratures), estimate the variances. Note that, if the function has any kind of smoothness, the variances will drop with decreasing sample volumes. One can set up a histogram guess of the function and do importance sampling based on the previous experience. There is much room for creativity. It is important to always make sure that one ultimately converges to the correct limit. This is comparativley easy to prove in many cases. It is much harder to get an algorithm to probe a system in finite time. We will next discuss a wide-spread class of algorithms that "learns from experience" and tries to do this with increasing efficiency.

5.4 Ising model (1d)

The Ising model is a toy model for ferromagnetism (see figure 5.4). Its dynamics is defined by the Hamiltonian energy for a given spin configuration $\vec{S} = (s_0, s_1, \dots, s_{N-1})$ is given by

$$H(\vec{S}) = \frac{1}{2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} J_{ij} s_i s_j - B \sum_{i=0}^{N-1} s_i \tag{5.23}$$

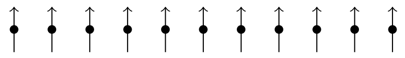

Figure 5.4: Chain of aligned spin in 1d. This would be the ground state of the Ising model for $J < 0$.

J_{ij}... interaction, determines the topology of the model. Simplest case $J_{ij} = J\delta_{|i-j|,1}$ (only nearest neighbors interaction) $s_i = \pm 1$... spin states.

Problem: dimension of the Hilbert space grows very rapidly $\sim 2^N$. In the limit $N \to \infty$ there are closed solutions.

"The harmonic oscillator of staticstical physics"

Ground state

For $J < 0$: all spins aligned with B: $s_i = 1$ $(B > 0)$ For $J > 0$: alignement is energetically costly.

5.4.1 Finite temperatures $kT > 0$

For given temperature kT the occupation prabability for a spin configuration $\vec{S}$ is given by the Boltzmann factor:

$$P(\vec{S}) \sim \exp[-H(\vec{S})/kT]. \tag{5.24}$$

Average values of energy and magnetization

For the energy we have:

$$\langle E \rangle = \sum_{\vec{S}} H(\vec{S}) P(\vec{S}) = \frac{\sum_{\vec{S}} H(\vec{S}) e^{-\frac{H(\vec{S})}{kt}}}{\sum_{\vec{S}} e^{-\frac{H(\vec{S})}{kt}}} \tag{5.25}$$

The average magnetization s can be calculated in a similar way and for $J = -\epsilon, \epsilon > 0$ we have:

$$\langle s \rangle = \frac{\sinh(B/kT)}{\sqrt{\cosh^2(B/kT) - (1 - e^{-4\epsilon/kT})}} \tag{5.26}$$

Correlation function

$$\langle s_i s_j \rangle = \exp(-|i-j| / \ln \tanh J/kT) \tag{5.27}$$

5.4.2 The naive approach: randomly probe the configuration space

In the end, what we want to know are high-dimensional discrete sums, say the average magnetization

$$\langle s \rangle = \sum_{\vec{S} \in \mathcal{S}} s_0 \exp[-H(\vec{S})/kT] / \sum_{\vec{S} \in \mathcal{S}} \exp[-H(\vec{S})/kT] \tag{5.28}$$

We have selected s_0 as one representative of the spins. If we use *periodic boundary conditions* $s_N \equiv s_0$, indeed any spin spin is representative for any other. For a smaller number of particles, we can proceed by brute force and sum over all 2^n configurations. For large numbers, this is out of question. At the same time, we can expect many vectors $\vec{S}$ to be similar in the sense that they have about the same energy $H(\vec{S})$ and similar contribution to the sums. Therefore, it will be sufficient to probe the space by Monte-Carlo methods without actually probing every single state. The problem is that, depending on temperature, certain configurations give the main contribution. Say, at very low temperatures, the exponent will strongly favor ground state, with almost all spins aligned (paramagnetic case $J < 0$) and just a few spins flipped. Most random states that we may generate will give no contribution and/or would be rejected in a von Neumann rejection algorithm. A way out of this dilemma is to start from some point in the integration volume and randomly move through it in such a way as to visit all regions with probability proportional to the Boltzmann factor. This is achieved by the so-called “Metropolis algorithm”.

5.4.3 The Metropolis algorithm

Here is a recipe how to construct a set of N spin configurations $\vec{S}^{(n)}, n = 0, 1, \ldots, N-1$ such that they are distrbuted with probability

$$P(\vec{S}) = \exp[-H(\vec{S})/kT] \tag{5.29}$$

where $P(\vec{S})$ is defined on all states $\vec{S}$ of configuration space.

Select some initial spin configuration $\vec{S}^{(0)} = (s_0, s_1, \ldots, s_{N-1})$, and compute its energy $E_0 = H(\vec{S}^{(0)})$. Now iterate the Metropolis update M times

1. Randomly select one spin s_i of the configuration $\vec{S}^{(n)}$.
2. Flip $s_i \to -s_i$ to obtain $\vec{S}^{(n)} \to \vec{S}^{(*)}$
3. Compute the energy $E_* = H(\vec{S}^{(*)})$ for $\vec{S}^{(*)}$
4. If $E_* \le E_n$, accept new configuration: $\vec{S}^{(n+1)} = \vec{S}^{(*)}$, $E_{n+1} = E_*$
5. Else accept new configuration with probability $\exp[-(E_* - E_n)/kT]$
6. if finally recjected, duplicate old configuration: $\vec{S}^{(n+1)} = \vec{S}^{(n)}$, $E_{n+1} = E_n$
7. compute $E_{n+1} = H(\vec{S}^{(n+1)})$

This procedure produces a sequence of $\vec{S}^{(n)}$, where each configuration appears with probability $\propto e^{-\frac{H(\vec{S}_n)}{kT}}$

Implementation of point 5: For $E_* > E_n$ we have that $\exp[-(E_* - E_n)/kT] \in [0,1]$. Therefore to implement 5 we should pick a random number $r \in [0,1]$ and if $r < \exp[-(E_* - E_n)/kT]$ we accept the move, else we reject it.

Why does it work?

[This reasoning is a minor reformulation of the original argument by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller, Journal of Chemical Physics 21, 1887 (1953).]

To follow the reasoning, it may help if you consider our spin flipping procedure a "radom walk" through an N-dimensional space (configuration space).

First, it is easy to see, that in the limit where we take the number of Metropolis updates $M \to \infty$ we will cover the whole space: by spin flipping, we can obviously reach any state in the space, and there is a finite probability for any state $\vec{S}^{(*)}$ to be accepted into our sample. Therefore, for sufficiently large M it becomes extremely unlikely for any state *never* to appear. That likelihood goes to 0 for $M \to \infty$. In general this property is called **ergodicity**.

Second, we need to show that in the long run, the number of times that a state $\vec{S}_k$ is visited is $C\exp(-E_k/kT)$ (C being the proper normalization constant). This we will do by showing that, as long as we have *not* visited all sites according to their Boltzmann factors, it is more likely to visit the under-represented sites than the over-represented ones.

We label all possible configurations as $\vec{S}_k$, $k = 0, 1, \ldots, 2^N - 1$. This is to be distinguished from the seqence of configurations generated by the algorithm $\vec{S}^{(m)}$, $m = 0, 1, \ldots, M-1$, which will in general not contain all configurations $\vec{S}_k$ and where a single $\vec{S}_k$ may appear repeatedly.

Let P_{kl} be the probability that a single spin flip takes a given configuration $\vec{S}_k$ into $\vec{S}_l$. That is understood after step 1 but before making any further selection of steps 2 and up. Then it is easy to see that $P_{kl} = P_{lk}$: if $\vec{S}_k$ and $\vec{S}_l$ are connected by flipping, say, spin s_i, as the chance of randomly just selecting that s_i out of $s_0, s_1, \ldots, s_{N-1}$ is $1/N$ for both, $\vec{S}_k$ and $\vec{S}_l$. For definiteness, let $E_k \leq E_l$. Let v_k and v_l be the number of times that the two configurations are visited. Then the average number of times that we go from the v_l visits of $\vec{S}_l$ to $\vec{S}_l$ is just $j_{l\to k} = v_l P_{lk}$ (as $E_k \leq E_l$ we will certainly accept the move). We have selected the notation "$j_{l\to k}$" to suggest an average "flux" from l to k. On the other hand, the average number of times we proceed from the v_k visits at $\vec{S}_k$ to $\vec{S}_l$ is $j_{k\to l} = v_k P_{kl}\exp(-(E_l - E_k)/kT$. Only for $v_x \propto \exp(-E_x/kT), x = k, l$ there is no net "flux" $j_{k\to l} + j_{l\to k} = 0$. In all other cases, our procedure will take us more often to the under-represented sites: our prescription generates a random walk, whose stationary distribution is the Boltzmann-distribution.

For this conclusion, $P_{kl} = P_{lk}$ is essential. The generalization of this property is called **detailed balance**.

Problems and limitations

The above arguments only show that in the limit of infinitely many steps, we will arrive at the equilibrium (Boltzmann) distribution. It does not say anything about how quickly the limit is reached.

If the space is covered very slowly, i.e. if the random walk remains in the vicinity of any initial point for many steps, the algorithm will converge very slowly. This can be caused by too small steps or too large space. It can also be caused by "barriers" separating one part of the configuration space from the other.

Example: double-well potential with large barrier between the wells.

A way out of such situations can be the following generalization of the Metropolis algorithm, which allows tuning the step size by selecting different step procedures, not just the single spin flip.

Problem 4.68: Use `metropolis.py` from the repository and speed it up by avoiding exponentiations and full energy evaluations inside the metropolis loop. Also, think how *integer* random numbers in a large integer range could be used for further speedup. (Unfortunately, this cannot be implemented in Python as there seem to be no fast integer random numbers.)

Problem 4.69: Extend `metropolis.py` or your new faster version to compute the two-point correlation function $\langle s_i s_{i+n} \rangle$ for a range of n. Verify that for large M the correct statistical reduction $\sim 1/\sqrt{M}$ of the error is found. Compare results for large N with the known limit $N \to \infty$.

5.4.4 Generalization: Metropolis-Hastings Algorithm

The idea can be generalized to any distribution $p(\vec{c})$ for a vector $\vec{c}$ in space $\mathcal{K}$, if we use a "random walk" prescription $\vec{c}^{(n)} \to \vec{c}^{(n+1)}$, where $\vec{c}^{(n+1)}$ is selected according to some transition probability $Q(\vec{c} \to \vec{c}')$. The prescription should fulfill the two conditions:

- **ergodicity**: selecting $\vec{c}^{(m)}, m = 1, ..., \infty$ by the prescription $Q(\vec{c}^{(n)} \to \vec{c}^{(n+1)})$ covers the whole space: $\lim_{N\to\infty} \min_m |\vec{c}^{(m)} - \vec{d}| = 0 \quad \forall \vec{d} \in \mathcal{K}$
- **detailed balance**: let $p(\vec{c})$ be the equilibrium distribution, then:

$$p(\vec{c})Q(\vec{c} \to \vec{b}) = p(\vec{b})Q(\vec{b} \to \vec{c}). \tag{5.30}$$

The "detailed balance" condition is sufficient, but not necessary: it can be relaxed to a "balance" condition:

$$\int db^n \, p(\vec{c})Q(\vec{c} \to \vec{b}) = \int db^n \, p(\vec{b})Q(\vec{b} \to \vec{c}) \tag{5.31}$$

i.e. in equilibrium, the "outgoing" probability flux away from a given state $\vec{c}$ to *all* other states $\vec{b}$ is equal to the "incoming" flux from all states into $\vec{c}$. To understand the difference to "detailed balance", you can imagine an equilibrium situation of three equally occupied state vectors with a probability "ring flux" $\vec{c} \to \vec{d} \to \vec{e} \to \vec{c}$: detailed balance is violiated for the pairs $\vec{c}, \vec{d}$ and $\vec{d}, \vec{e}$, as the flux only goes one way, but balance obviously is fulfilled.

Example: Detailed balance for the Boltzmann distribution $p(\vec{c})$ and Q of the original Metropolis algorithm fulfilled as:

$$e^{-E_n/kT} Q(\vec{S}^{(n)} \to \vec{S}^{(m)}) = e^{-E_m/kT} Q(\vec{S}^{(m)} \to \vec{S}^{(n)}) \tag{5.32}$$

with Q:

$$\text{for } E_n < E_m : Q(\vec{S}^{(n)} \to \vec{S}^{(m)}) = \frac{1}{q} e^{-(E_m - E_n)} \text{ and } Q(\vec{S}^{(m)} \to \vec{S}^{(n)}) = \frac{1}{q}, \tag{5.33}$$

where q is the number of neighbours just one spin-flip away and similarly for $E_m < E_n$.

5.4.5 Wang-Landau sampling (WLS)

Ref: Fugao Wang and D.P. Landau, Phys. Rev. Lett. **86**, 2050 (2001).

Sample the density of states rather than the configurations. Again assuming a Boltzmann distribution and in addition that the quantities that we want to average over are smooth functions energy, the density of states is a useful concept. For example, the internal energy as a function of temperature for system depending continuously on its coordinate vector $\vec{x} = (x_0, x_1, \ldots, x_{N-1})$ is

$$\int d^N x \, H(\vec{x}) e^{-H(\vec{x})/kT} / \int d^N x \, e^{-H(\vec{x})/kT}. \tag{5.34}$$

As the integrand does not explicitly depend on $\vec{x}$, it is useful to write $d^N x = dE g(E) d\Omega$. In the simple case of a N free particles $H(\vec{P}) = (\vec{P} \cdot \vec{P})/2 =: |P|^2/2$, where $\vec{P} = (p_x^{(0)}, p_y^{(0)}, p_z^{(0)}, p_x^{(1)}, p_y^{(1)}, p_z^{(1)}, \ldots)$ the **density of states** $g(E)$ is easily deduced from $d^{(3N)}p = d|P||P|^{(3N-1)} d\Omega = dE E^{(3N-2)/2} d\Omega$, i.e. $g(E) \propto E^{(3N-2)/2}$. In more general cases, $g(E)$ is exactly what we need to know to compute the expectation value for a system's energy at a given temperature. Interesting quantities like specific heat can be deduced.

The Wang-Landau algorithm directly aims at computing $g(E)$. The general idea is to divide the total energy range into bins and randomly walk through these bins such that in the end one has visited each bin equally often. In the process we will record how much the system "likes" to go to each of these bins, i.e. the energies of how many states fall into each bin. Here is how it works step by step (using the example of the Ising model):

1. Divide your total range of energies into bins $[E_i, E_{i+1}] = [E_i, E_i + \Delta E]$:

2. Set up two histograms G_i and H_i, where G_i will approximate $g(E_i)$ and H_i will count how many times the E_i-bin is visited.

3. Initialize $\vec{S}^{(0)}$ to an arbitrary configuration, set all $G_i \equiv 1$ and $H_i \equiv 0$, calculate $E_0 = H(\vec{S}^{(0)})$

4. Pick some (empirical) factor f ($f = 2.7$ appears to be a good choice).

5. Random walk loop:

6. From the current configuration $\vec{S}^{(n)}$ with energy E_n, get $\vec{S}^{(*)}$ by a random spin flip as in the Metropolis algorithm and calculate $E_* = H(\vec{S}^{(*)})$

7. Accept the new configuration $\vec{S}^{(*)}$ with probability

$$P(E_n \to E_*) = \min \left[1, \frac{G_n}{G_*} \right] \tag{5.35}$$

where G_* is the G_i-value of the bin, into which E_* falls. (Same type of rule as in the Metropolis algorithm, but using G_i instead of the Boltzmann distribution).

8. For whichever state $\vec{S}^{(n+1)}$ is accepted, increment the bin-count $H_i \to H_i + 1$ and multiply the $G_i \to fG_i$.

9. Every once in a while, check the flatness of H_i.

10. If histogram is "sufficiently flat", e.g.

$$\frac{|H_{\max} - H_{\min}|}{H_{\max} + H_{\min}} < 0.2 \tag{5.36}$$

rescale the factor $f \to \sqrt{f}$. Keep G_i but reset $H_i \equiv 1$. The factor 0.2 is an empirically reasonable choice.

11. if $f \sim 1$ stop, else start over.

After this procedure, G_i contains an approximation to the density of states, within tolerances determined by the choice of bins, limits on f, flatness-criterion.

Why does WLS work?

Assuming a random walk, in step (6) the probability for randomly selecting a state $\vec{S}^{(*)}$ with energy E_* in the energy interval $E_i, E_i + \delta E$ is just $\sim g(E_i)\Delta E$. We may assume that the rate for going from $E_n \to E_{n+1}$ by a random spin flip proportional to the density of states near E_{n+1}: $R(E_n \to E_{n+1}) = g(E_{n+1})\Delta E$.

By the rejection in step (7), this probability is counter-acted by the probability penalty of the selection criterion, which favors sites that have not been visted much so far.

In the beginning the penalty f for prior visits to a site is very high, ensuring that we quickly cover the whole energy range.

A flat histogramm H_i indicates that we have visited all sites equally often. That means that we have reached an approximate "detailed balance situation" for the energy bins, assume for definiteness $G_n < G_{n+1}$ and denote by $P(E_n \to E_{n+1})$ the acceptance probability,

$$R(E_n \to E_{n+1})P(E_n \to E_{n+1}) = R(E_n \to E_{n+1})\frac{G_n}{G_{n+1}} = R(E_{n+1} \to E_n)\underbrace{P(E_{n+1} \to E_n)}_{=1} \tag{5.37}$$

or

$$\Delta E g(E_{n+1})\frac{G_n}{G_{n+1}} = \Delta E g(E_n) \tag{5.38}$$

or

$$\frac{G_n}{G_{n+1}} = \frac{g(E_n)}{g(E_{n+1})}. \tag{5.39}$$

The initially high penalty f ensures fast coverage of all energies, but it also exagerates statistical fluctuations. Its successive reduction allows fine-tuning of the rougher patterns established in earlier rounds.

Ultimately, one is driven to a situation, where one visits all energies equally often (flat histogram). This makes sure that one does not miss important energy regions. In the process, one records how much the system "likes" to go to certain energy regions, which is, assuming that all states are equally likely (i.e. there is no additional bias for any configuration), proportional the density of states.

Note that, for purposes other than energy, states within an energy bin may be very different. So if your observable is not just a function of energy, the density of states is not a useful quantity.

A technical note

Repeated multiplications by f will quickly lead to floating overflows. This can be avoided by working with $\log(G_i)$ instead.

It values $f \approx 1 + 10^{-8}$ can be achieved and provide a usefull break-off level for the procedure.

Chapter 6

Classical systems and chaos

6.1 The Kepler problem

6.1.1 Reduce to 1st order ODE / Hamiltonian formalism

Choose center-of-mass coordinates and reduced mass = 1 (use appropriate units) then the Kepler problem is

$$\ddot{\vec{r}} = -\frac{\vec{r}}{r^3}. \tag{6.1}$$

With $\dot{\vec{r}} = \vec{p}$

$$\frac{d}{dt}\begin{pmatrix}\vec{r}\\ \vec{p}\end{pmatrix} = \begin{pmatrix}\vec{p}\\ -\vec{\nabla}\frac{1}{r}\end{pmatrix} \tag{6.2}$$

Introducing the phase space coordinates $u := (\vec{x}, \vec{p})$, we can also write

$$\frac{d}{dt}u = \widehat{J}\nabla_u[\frac{\vec{p}^2}{2} + V(\vec{r})] \tag{6.3}$$

with

$$\widehat{J} := \begin{pmatrix}0 & \widehat{I}\\ -\widehat{I} & 0\end{pmatrix} \quad \text{and} \quad \nabla_u := \begin{pmatrix}\vec{\nabla}_r\\ \vec{\nabla}_p\end{pmatrix} \tag{6.4}$$

and the 3×3 identity matrix $\widehat{I}$. For negative energies solutions are the Kepler orbits with period

$$T = 2\pi a^3, \tag{6.5}$$

where $a = 1/(2|E|)$ is the major axis of the ellipsis. We can put any ODE solver to test by computing $|\vec{r}(t_0) - \vec{r}(t_0 + nT)|$.

Problem 1.70: Use the existing ODE solver routine to compute $\vec{p}(t_0)$. Also, investigate the stability of the ODE along the path, i.e. get the eigenvalues of the Jacobian matrix as the solution evolves. Check the relation to the step size.

We observe that although we can control the error, it grows severely for long-time propagation, it grows by a power law or worse! In the present example, the main part of the error is due to incorrect periodicity. This can be easily corrected, which strongly reduces the error. But still, the solution drifts away from the exact one.

6.2 Symplectic integrators

If we want to study qualitative features of the long-time behavior of a system, we need methods that maintain these qualitative features. In quantum mechanics, after discretizing the system, we can construct methods that are strictly unitary: for example the split-step methods, but also Crank-Nicolson. Note that unitary transformations leave the eigenvalue spectrum invariant and remember that eigenvalues can be considered as the infinite time limit of the system - "states" of the system, be they bound or scattering. Even if our discretized version of the TDSE only qualitatively reproduces the spectrum of the operator, the long-time behavior will remain "qualitatively" close to the original system.

6.2.1 Leap frog propagator

The simplest example of a symplectic integrator is the "leap frog" method. With step size $x_i \approx x(ih)$ and $p_{i-1/2} \approx p((i-1/2)h)$ the method is

$$\begin{aligned} x_i &= x_{i-1} + hp_{i-1/2} && (6.6) \\ p_{i+1/2} &= p_{i-1/2} + hF(x_i) && (6.7) \\ & && (6.8) \end{aligned}$$

where $F(x_i)$ is the force at point x_i.

To see that this procedure is symplectic (and what exactly this means), we use a more general formulation below, where we will write a whole class of operators (including leap frog) in fancy forms like

$$u_i = e^{hL_T/2} e^{hL_H} e^{hL_T/2} u_{i-1} \tag{6.9}$$

Problem 2.71: Implement the leap frog method and reconsider the evolution of the error at long times.

6.2.2 Symplectic metric

Canonical transformations of a Hamiltonian system play the same role in classical mechanics as unitary transformations do in quantum mechanics: they leave the dynamical equations formally intact. More precisely, they leave the Poisson brackets invariant (quantum mechanics: the commutators). As the Hamiltonian equations are given as Poisson brackets with the Hamiltonian function, they also leave the Hamiltonian equations invariant. Classical time-evolution itself is a canonical transformation (as quantum time-evolution is unitary).

If you have been exposed to the concept, you know that canonical transformations leave the canonical quadratic form

$$\omega = dx \wedge dp \tag{6.10}$$

invariant. These quadratic forms define the "symplectic metric" in phase-space (more precisely: the tangent space of phase-space), which is exactly the metric given by our matrix $\widehat{J}$. Note that this metric is not applied to points in phase space, but to gradients ∇_u: the "tangent space" of phase-space.

Let $A(\vec{x}, \vec{p}) = A(u)$ and $B(\vec{x}, \vec{p}) = B(u)$ be to functions on (many-particle) phase space. Then the gradient $\vec{\nabla}_u A(u) = \vec{X}_A(u)$ define vector fields, i.e. define a vector for each point in phase space.

The Poission brackets can be written as a (pseudo-)scalar product between the vectors $X_A(u)$ adn $X_B(u)$ at each phase-space point u:

$$\{A, B\} = \nabla_u A \cdot \widehat{J} \nabla_u B = \vec{X}_A \cdot \widehat{J} \vec{X}_B \tag{6.11}$$

So if we have some (x,p)-coordinate transformation $S : A \to S(A)$ such that

$$\nabla_u A \cdot \widehat{J} \nabla_u B = \nabla_u S(A) \cdot \widehat{J} \nabla_u S(B), \tag{6.12}$$

then we know that the dynamical structure, equations, energies are unaffected.

The poisson bracket between any two observables $A(u)$ and $B(u)$ is invariant under a map S, if the poisson brackets for all phase space coordinates $\{u_i, u_j\}$ are invariant. First, the transformation for any observable A is induced by the transformation of the coordinates themselves

$$S(A)(u) := A(S(u)) \tag{6.13}$$

where we define the transformation for each phase space point separately

$$S(u)_l = S(u_l), \tag{6.14}$$

think of a time evolution

$$S_t(u_l(0)) = u_l(t). \tag{6.15}$$

Then

$$\{S(A), S(B)\} = \nabla_u A(S(u)) \cdot \widehat{J} \nabla_u B(S(u)) \sum_{kl} (\partial_k A) \nabla_u S(u_l) \cdot \widehat{J} \nabla_u S(u_k)(\partial_k B) \tag{6.16}$$

The Poisson bracket is invariant when

$$\nabla_u S(u_l) \cdot \widehat{J} \nabla_u S(u_k) = \nabla_u u_l \cdot \widehat{J} \nabla_u u_k = (\widehat{J})_{kl} \quad \forall k, l \tag{6.17}$$

6.2.3 Split step propagators

A large class of symplectic integrators are split-step type. For that we assume that the Hamiltonian and with it the corresponding Lie derivatives can be split into two parts

$$H = T + V, \qquad L_H = L_T + L_V. \tag{6.18}$$

The most naive way to approximate the time-evolution is

$$e^{hL_{\mathcal{H}}} \approx e^{hL_T} e^{hL_V} \tag{6.19}$$

As each factor on the right hand side is symplectic, the whole procedure is symplectic. However, consistency order $p = 1$ only.

As in the split-step for quantum operators, the symmetrized form

$$e^{hL_{\mathcal{H}}} \approx e^{hL_T/2} e^{hL_V} e^{hL_T/2} \tag{6.20}$$

is consistency order $p = 2$.

With the Lie derivative, these expressions look somewhat formal. Clearly, we will use the linearized version where the operation turns into a matrix exponentiation

$$e^{hL_H} u = e^{hL_H(u)} u, \tag{6.21}$$

where the Lie matrix is evaluated at the phase space point to which the exponential is applied, for example

$$e^{hL_T} e^{hL_V} u = e^{hL_T(w)} e^{hL_V(u)} u \quad \text{with} \quad w = e^{hL_V(u)} u. \tag{6.22}$$

For generalization to higher consistency order one uses repeated piecewise linear approximations. For $p = 4, 6, \ldots$ this leads to

$$e^{hL_{\mathcal{H}}} = \prod_{n=1}^{p} e^{a_n h L_T} e^{b_n h L_V} + \mathcal{O}(h^p + 1) \tag{6.23}$$

The coefficients a_n and b_n can be obtained by comparing the series expansions of left and right hand sides up to a given order in h. For $p = 4$ we have

$$\vec{a} = \frac{1}{2(2 - 2^{1/3})}(1, 1 - 2^{1/3}, 1 - 2^{1/3}, 1), \quad \vec{b} = \frac{1}{2 - 2^{1/3}}(1, -2^{1/3}, 1, 0) \tag{6.24}$$

Generally speaking this could be not terribly useful for large phase space dimension, as we need to do matrix exponentiations. If we assume that T, like ordinary kinetic energy, depends only on $\vec{p}$ and V, like an ordinary potential, depends only on $\vec{r}$, we see that

$$\exp(hL_T(w)) = 1 + hL_T(w) \tag{6.25}$$

as all higher powers $L_T^n = 0$, $n > 1$, and similarly for L_V. With this, in leap-frog we recognize the symmetric $p = 2$ split step symplectic integrator.

6.2.4 The merit of symplectic integrators

The key importance of symplectic integrators is that they preserve the general structure of classical mechanics. One important result of general theory is that, if we have a differentiable *symplectic* (=canonical) time evolution S_t there exists a Hamiltonian function H_D that generates it:

$$u(t) = S_t(u(0)) \Leftrightarrow S_t = \exp[tL_{H_D}] \tag{6.26}$$

Hamilton's equations are a consequence. Conversely, if the scheme is *not* symplectic, there is cannot be any Hamiltonian that would generate it.

We try to construct our numerical scheme such that it produces a symplectic evolution S_t and that the corresponding H_D is not too different from the original H. In practice, H_D can rarely be determined. We may hope that the (unknown) H_D is but a "slightly perturbed" version of H. This is relevant for studying chaotic behavior, as the KAM (Kolmogorov, Arnol'd, Moser) theorem says, that sufficiently small perturbations of the Hamiltonian leave the global (topological) properties of the solution unchanged. In particular, quasis-periodic motions remain quasi-periodic. If we want to investigate a given system for "stable orbits", a symplectic numerical scheme has a chance to uncover them.

Note the close analogy to quantum mechanics: if we have a differentiable *unitary* time-evolution U_t, there exists a (selfadjoint) Hamiltonian operator that generates it:

$$\Psi(t) = U_t \Psi(0) \Leftrightarrow U_t = \exp[-itH] \tag{6.27}$$

From this point of view, the TDSE is a trivial consequence. The difference between quantum and classical mechanics is not in the dynamics, it is how measurements ("observables") work.

6.2.5 Performance

Apart from general considerations, it is interesting to compare the performance of a symplectic integrator with, for example, a simple Runge-Kutta 4. The 4th order symplectic integrator given above requires 3 potential evaluations (the expensive part of the integrator).

In a comparison between the standard RK4 and the 4th order symplectic integrator for the Kepler problem, RK4 clearly wins: at the same number of steps, the error with RK4 is lower by about a factor 20. We may count into that that the symplectic solver requires only 3 potential evaluations. As this is the time-crititcal part of the calculation, the comparision is somewhat improved, but still RK4 outperforms the symplectic solver by a factor 10 in accuracy at comparable CPU times. In terms of computation times this means about a factor of 2.

The use of the symplectic integrator lies in its respecting more fundamental properties of the Hamiltonian system. An example that shows this is the Henon-Heiles Hamiltonian derived for the motion of stars in a Galaxy disk of the form

$$H(q_1, q_2, p_1, p_2) = \frac{1}{2}(p_1^2 + p_2^2 + q_1^2 + q_2^2) + q_1 q_2^2 - \frac{1}{3} q_2^2 \tag{6.28}$$

For studying the long-time evolution of the system in 4-dimensional phase-space, we use the two-dimensional "Poincare-map", i.e. the points in the q_2, p_2-plane where $q_1 = 0$. The p_1 value is then fixed by energy conservation.

6.2.6 Restricted 3-body problem

Hamiltonian

$$H = \frac{1}{2}\vec{p}^2 - \kappa \left(\frac{m}{\vec{x}_m(t) - \vec{x}} + \frac{\mu}{\vec{x}_\mu(t) - \vec{x}} \right) \tag{6.29}$$

Here a large mass m and a second, smaller mass μ rotate around each other at constant angular velocity. In this field a (very small) mass = 1 moves.

The motion if the light mass is instable for $\mu/m > 0.04$, otherwise it may be stable. In the solar system indeed the system sun-Jupiter-Trojans forms such an approximate triangle.

Jupiter mass = 0.00095 Solar mass

6.3 Classical trajectory Monte-Carlo

The differences between classical and quantum mechanics is smaller than what appears at first glance. For example, the dynamics of the harmonic oscillator is the same in both regimes in the sense that expectation values of position and momentum evolve in the same way. This somewhat more generally holds for Hamiltonians that are polynomial up to degree 2 in both, position and momentum. In an approximate sense, this holds for a much larger class of systems.

Classical trajectory Monte Carlo methods are based on the idea that we can approximate quantum dynamics by their statistical aspects. This excludes "entanglement" effects, as they appear, e.g., in Bell's inequalities. We assume that quantum uncertainty is nothing but a lack of knowledge and pretend to cover up this lack of knowledge by unbiased random probing of the system.

A typical situation is scattering from a quantum system, a simple example being proton scattering from the hydrogen atom $H^+ + H(1s)$. Physical questions are final momenta, final atomic states, possible transfer of the electron to the projectile. Considered as a classical system, this is a 3-body Coulomb scattering problem. The dynamical equations are not difficult to solve (even in Python). The key tasks are the choice of initial conditions and the interpretation of the final state.

For the initial conditions, the description of the incoming proton at a given momentum and a statistical distribution of impact parameters is straight forward. In contrast, the Hydrogen ground state is a truely quantum state and care is required when modelling the state into a distribution in phase space. For the ground state, we might use the Wigner distribution, which happens to be positive. Already for the first excited state, this approach breaks down in principle, as the Wigner distribution becomes negative close to the nucleus and cannot be interpreted as a probability distribution. One may instead use approaches like the Husimi-distribution, which averages the Wigner distribution over phase-space volumes of size $\hbar$ and is always positive. In this way, we replace quantum uncertainty with a reasonable, but nontheless arbitrary statistical uncertainty. The choice of initial conditions is one of the often secret ingredients of classical trajectory Monte Carlo simulations and always involves sustantial approximations. Another problem is the Pauli exclusion principle, which cannot be rigorously mapped into a classical model.

As the result of the classical propagation one obtains the phase-space distribution for all particles at some large time. There may be two particles comparatively close at a relative energy that is bound and one particle far away. The far away particle can be again easily associated with a click in a detector at some momentum. The bound system should be mapped back onto the quantum system. Different quantum states would correspond to different inelastic scattering "channels". Such a mapping can hardly be made unique and one must content oneself with recording the (continuous) distributions of energies and angular momenta.

While the previous MC methods were all exact the limit $N \to \infty$, classical trajectory MC is inaccurate by construction. There are two reasons to still use it: it allows the formation of imagination of what “really” happens in the process, “which particles go where a which momentum”. Strictly, such ideas are meaningless. However, many processes that appear quantum, are classical in essence and the pictures *are* adequate. The second reason is more mondane: while quantum calculations may be exact in principle, in practice achievable accuracies may be very bad and subjected to methodological artefacts. MC may give a more plausible estimate of physical observables like cross sections.

6.4 Parallelization

There is an advantage of many Monte-Carlo methods: it is just doing the same task over and over again with different inputs and accumulating the results in the end. The individual tasks can be accomplished without any coordination with the other tasks. This is ideal for parallelization and contributes to the popularity of MC methods at an age, where computer speed is mostly increased by building more rather than faster CPUs.

Chapter 7

High performance and parallel computing

7.1 LAPACK — solving your linear algebra problems

From the manual:
"LAPACK is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision."

Mathematical background: Golub, van Loan: "Matrix Computations"

7.1.1 Structure of the LAPACK package

7.1.2 LAPACK subroutine names

The LAPACK subroutines used to solve an eigenvalue problem have a name which has the following form:

`tmacmp[x]`:

- `t`: data type `s,d,c,z` for float (4 byte real, $\sim$ 7 decimal digits), double (8 byte real, $\sim$ 14 decimal digits), complex (8 byte, $2\times\sim$ 7 decimal digits), and double complex (16 byte, $2\times\sim$ 14 decimal digits)
- `ma`: matrix type `ge,gb,gt,sy,po,pb,he,hb,`... for general,general banded, general tri-diagonal, symmetric, positive, positive banded, hermitian, hermitian banded etc.
- `cmp` (two or three letters): what to compute: `ev,sv,svd,trf`... for eigenproblem, linear solver, singular value decomposition, LU factorization, etc.
- `x`: driver routines distinguish between "simple" and "expert" levels, `x` indicates expert level with many more parameters.

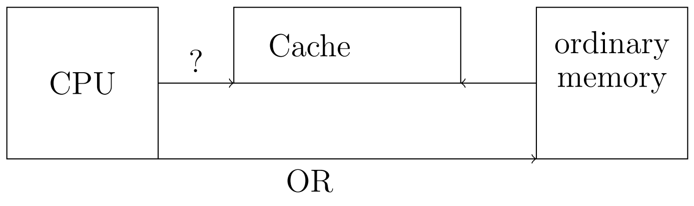

Figure 7.1

Example 1: `zhbevx` solves the 16-byte complex banded hermitian eigenvalue problem, expert version.
Example 2: The command lapack.dsbev()" solves a double symmetric banded eigenvalue problem.

BLAS

Lapack relies on BLAS — the Basic Linear Algebra Subroutines. This has three levels of operations:

- Level 1: vector-vector, e.g. "axpy"
$$\vec{y} \leftarrow \alpha\vec{x} + \vec{y}; \tag{7.1}$$
- Level 2: vector-matrix like
$$\vec{y} \leftarrow \widehat{A}\vec{x} + \vec{y}; \tag{7.2}$$
- Level 3: matrix-matrix:
$$\widehat{C} \leftarrow \alpha\widehat{A}\widehat{B} + \beta\widehat{C} \tag{7.3}$$

BLAS is where the main optimization happens. Some compiler-vendors offer their own implementations of BLAS.

ATLAS

"Automatically Tuned Linear Algebra Software": during installation, the software explores your machine — memory speeds, cache-sizes and cache-speeds, etc. — to produce optimal code for one specific machine. Often outperforms even BLAS implementations provided by hardware and compiler vendors. Somewhat tiresome to install, but may pay off, when you do big tasks on some specific machine. Actually, should be installed by the system administrator.

Note: The cache is the fast part of the memory and it plays an important role in this optimization procedure. The cache is the place of the computer where you store data you are working with. So the CPU looks for data into the cache and if it doesn't find them there, it goes to look for them into the ordinary memory (see also figure 7.1) (this last search might be long). Not finding needed data in the cache is called "cache miss".

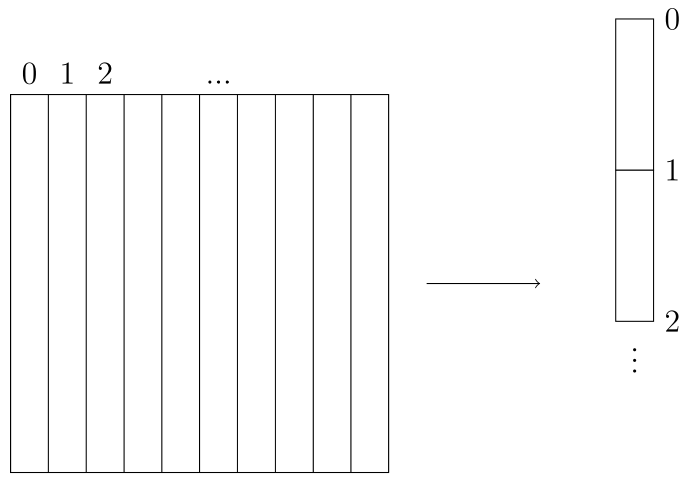

Figure 7.2: How a full matrix is stored into the memory of a computer.

7.1.3 Why is LAPACK fast?

- Algorithms are specified for details of the problem (full, banded, positive,...);
- Access patterns to data are local (as many operations on the same set of data as possible);
- Blocking, adjustment to specific hardware.

7.1.4 LAPACK matrix storage

Standard ways of storage (incomplete list):

- **Full**: Full matrices are stored column-wise (Fortran tradition) as a long string (see figure 7.2). This means that to make a move along a row is extremely long and therefore one should try to remain as long as possible along the same column. The position in linear storage is given by $A_{ij} : i + L_A j$ where L_A is the leading dimension of storage for A (i.e. $L_A \geq N_r$ where N_r is the number of row of A);
- **Upper packed**: for full but symmetric or hermitian matrices: Only the upper triangle is stored, the other half is inferred. In linear storage, $\widehat{A}_{ij}$ is at position $i + j(j+1)/2$
- **Upper banded**: for a symmetric or hermitian band matrix, only the diagonal and the super-diagonals need to be stored. The sub-diagonals can be obtained by transposition (and conjugation, in case of hermitian). The matrix elements of a matrix $\widehat{A}$ are stored in a 2-dimensional array $B : \widehat{A}_{i,j} = B[L_B{+}i{-}j, j]$. In linear storage for the B array this is position $L_B{+}i{-}j{+}L_B j$, with L_B being the leading dimension of the array B. Clearly, $L_B \geq 1+$ (number of super-diagonals of $\widehat{A}$) (see figure 7.3).

Matrix interface: define an abstract object matrix:

- where storage is (automatically) arranged according to structure;
- where linear algebra operations are implemented by some fast package (LAPACK).

Fourier Transform: FFTW.
Parallel linear algebra: PETSc.

Problem 1.72: Implement a method `solve()` into the module `gmatrix.py` that solves

$$\widehat{A}\vec{x} = \vec{b} \tag{7.4}$$

for *general* full and for symmetric banded real matrices $\widehat{A}$. Implement the method using the corresponding LAPACK routines from `scipy.linalg.flapack`. Test and compare performance of the two routines.
Hint: The arguments of the LAPACK routines are poorly documented in `scipy`. You get a list of functions and their arguments by `python>>>> help(scipy.linalg.flapack)`. Compare the arguments with the arguments documented on the LAPACK homepage to make the correct selections.

7.2 Some general advice

A hierarchy for solving physics problems on a computer:

1. Physics, not mathematics.
2. Mathematics, not algorithms.
3. Algorithms, not implementation.

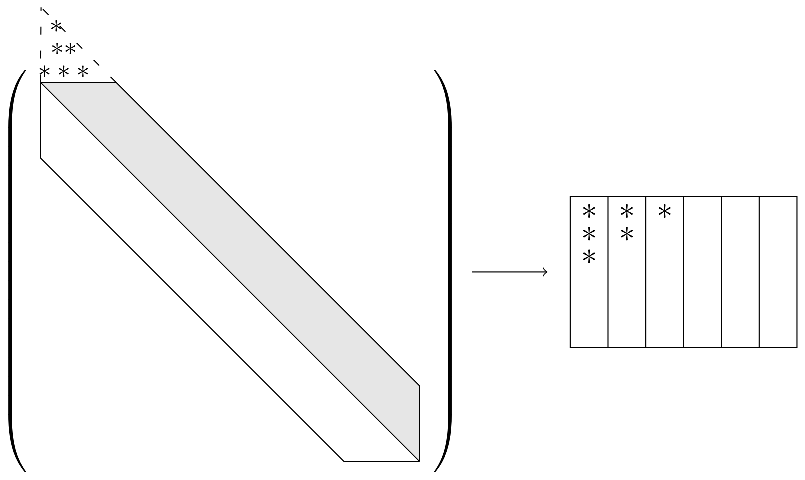

Figure 7.3: How a banded matrix is stored into the memory of a computer. The shaded region is the only part which is actually stored together with the $*$ which are some imaginary points which have been introduced in order to make everything symmetric.

4. Implementation clarity, not speed or elegancy.
5. Language independent implementation, not specific tricks.
6. Use standard language syntax and libraries.
7. General algorithms, not hardware-specific.
8. Solutions for specific types of computer architectures.
9. Isolate machine-specific algorithms or pieces of code in a single module with well defined interfaces.

7.3 Improve CPU performance

Moore's law, statement published in "Cramming more components onto integrated circuits", in "Electronics", vol. 38 (1965): "The complexity for minimum component costs has increased at a rate of roughly a factor of two per year... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer.

Key technologies: Conjugated Metal Oxide Semiconductors (CMOS), photolithography and excimer lasers. Still going on with rather basic physics research (junctionless transistors, inclusion of quantum effects . . .).

Limits for transistor density

- Structure size: photolithography wave length $\sim 200nm$ can produce structures as small as 30 nm.
- Heat-dissipation
- Granularity due to low electron numbers: statistical behavior rather than smooth "currents" and "charges". Finally, quantum effects.

How to convert number of transistors into speed?

Switch-times of transistors:

Performance: if density increases by a factor f, the linear distances between components decrease by only $\sqrt{f}$. I.e. communication time / component increases.

For taking advantage of many transistors we need *parallelism.*

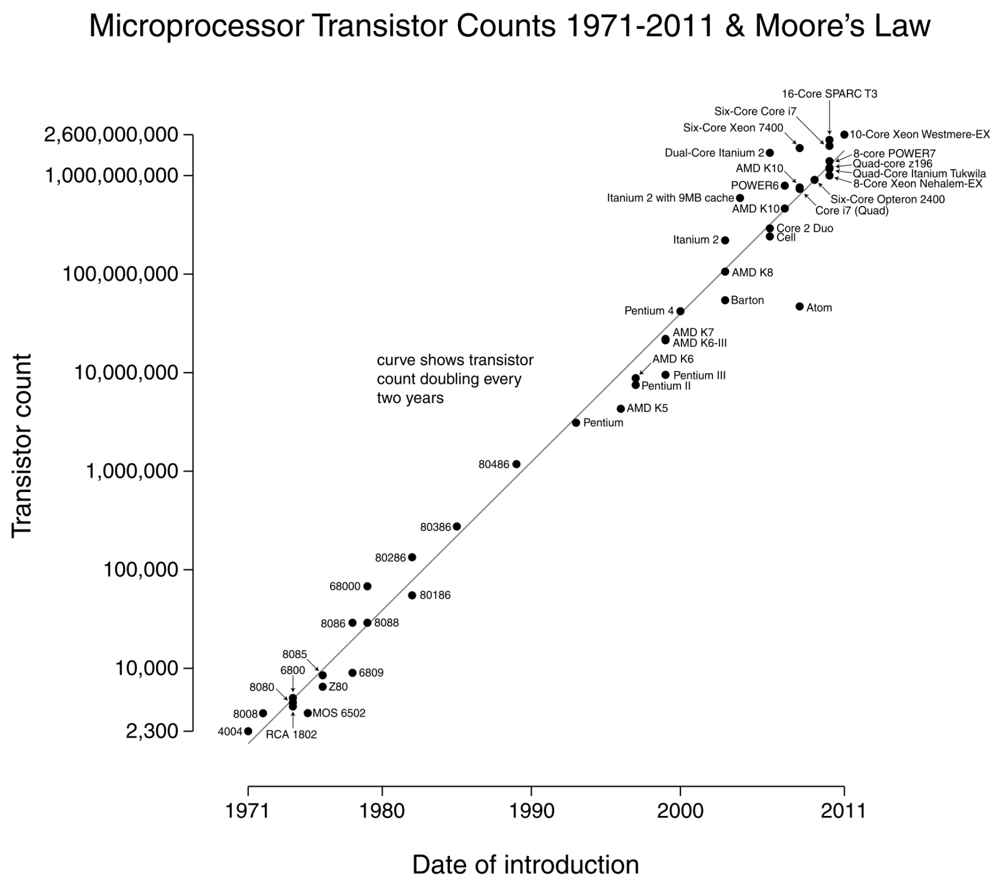

Figure 7.4: Moore's law: exponential growth of the number of components per chip at a rate of doubling every two years during a period of 40 years (1971-2011): $[2.6 \times 10^9/2300]^{1/20} \approx 2$

7.4 Parallelism

Idea: We have an exponentially growing number of transistors — probably organized into an also exponentially growing number of CPUs — let them all work on our problem.

Of little use, if our problem is a single hard problem requiring many steps one after the other. (A group of people is firing a grill at the Flaucher: as fast as the single person who knows best how to do it).

Can be fully exploited, if we actually have a large number of small problems. A perfect example of transforming a big seemingly single problem — the computation of a high-dimensional integral — into very many and almost independent small ones is Monte Carlo integration. (A group of people is moving a pile of dirt: large speedup if they do not get into each other's way and if the pile is not too small.)

Types of parallel operations:

- SIMD — single instruction multiple data, e.g. multiply a set of numbers by a scalar.
- MIMD — multiple instructions multiple data: let independent processes run and synchronize, e.g. Monte Carlo

Issues:

- synchronization: the CPUs' actions will in general depend on each other
- communication: the amount of information that needs to be communicated between CPUs
- load balancing: to take full advantage of all CPUs, they must be kept equally busy
- algorithms and debugging: the logical interdependencies become complex and confusing surprisingly quickly!

Comment on loadbalancing

When moving a pile of dirt, we can uses two strategies: (1) split the pile and alot a piece of the work to everybody (2) Let everybody take from the pile until the pile is gone.

(1) looks like a bad strategy (except as an incentive for lazy individuals): work will be finished only at the pace of the slower. However, if the workers were glued to the ground and could access only a limited range of the pile, it would be the only way to go. This is a bit the situation with distributed memory machines, where data cannot easily be moved to different CPUs. (2) balances the load according to every individuals capabilities: if all CPUs have the same access to all the memory, that is easy to do.

Still (1) may not be so bad, for example, when calculating all elements of a vector: IF each element is equally hard to calculate and IF all tasks are hosted on equally strong hardware. If either of these conditions is violated (2) is better (somewhat harder to implement).

Granularity

The amount of data and the time for which a single CPU can operate independently without need to communicate is the "granularity" of an algorithm:

"Coarse grain": hardly any communication needed, simple final assembly of the results: put all into a histogram, add them all up (MC), etc. Can be done over "slow" communication, e.g. "SETI@home (Search for extraterrestrial intelligence at home)"

"Medium grain": Large independent parts of the code, say subroutines doing different tasks.

"Fine grain": time-propagation of a PDE: at each step, continuity conditions need to be communicated between processors or a matrix-vector multiplication needs to be performed.

7.4.1 Possible speed-ups: Amdahl's law

Significant gain only, if serial parts of the code (setup, final processing of results), play a small role.

p-number of CPUs, T_1-execution time on a single CPU, T_p-execution time of parallel code. The speedup is

$$S_p = \frac{T_1}{T_p} \tag{7.5}$$

Let f be the fraction of time that the code spends in potentially parallelizable algorithms, and the respective times are parallel parts $T_p = fT_1/p$ and sequential parts $T_s = (1-f)T_1$ Then the there is a simple upper limit to the obtainable speedup (Amdahl's law)

$$S_p = \frac{T_1}{T_s + T_p} = \frac{1}{1 - f + f/p} \tag{7.6}$$

Further parallelization becomes inefficient once we reach $p \sim \frac{f}{1-f}$ processors. As we may have 1000's of processors available, we better keep f very very close to 1.

"Scaling" of a parallel code

If $S_p = a \times p$ we speak of "linear scaling", more or less the ideal case, in particular if $a \lesssim 1$.

Note that one sometimes can get "super-linear" scaling $a > 1$: One can get better than linear scaling, if by splitting the problem into smaller pieces, memory access gets faster. For the expert: cache-misses are reduced.

Communication time

Another obvious limit is the extra time needed for communication. Parameters going into that time:

Latency: how long it takes to initiate communication: Infiniband: $\gtrsim 1\mu s$

Signaling-rate Bit/s of (useful) data transmitted: Infiniband: $10 \sim 100 GBit/s$

Infiniband: currently wide spread technology for distributed memory machines (see below).

Number of connects needed, topology,

Suppose you have a perfectly parallel code and suppose you have a fixed amount of data that needs to be transmitted at any time step between any two "neighboring" processors, say, the boundary values of two finite elements belonging to two different CPUs, then $T_c = cp$. Then the speedup will be

$$S_p \sim \frac{T_1}{T_1/p + c} \tag{7.7}$$

Parallelization is efficient as long as $T_1 \lesssim pc$. Very fine grain parallelism is doomed:

Intel Core7 (2011): 159,000 MIPS (Mega instructions per second): if you are communicating with a latency of $1\mu s$, you need to do quite a few instructions before communicating for gaining from parallelism.

7.4.2 Shared vs. distributed memory

Much of communication is about accessing the same data by different CPUs. Therefore the physical arrangement of memory is important.

SMP - symmetric multi-processor machine

several CPUs sharing memory, using the same (broad band) bus. All memory locations are equally accessible to all CPUs

Multi-Core

"SMP on a single chip"

Distributed Computing

Distributed memory machines + fast network (e.g. Infiniband). Workhorses of present parallel computing

Example: "Beowulf" cluster: connect commodity machines by decent (e.g. gigabit) network.

7.4.3 Parallel algorithm: examples

Vector scalar product

Matrix-matrix product

A good example for the difficulties and potential gains of parallel algorithm is $N \times N$ matrix multiplication.

$$\widehat{C} = \widehat{A} \cdot \widehat{B} \tag{7.8}$$

Floating point operations count is $O(N^3)$. We first assume that the matrix elements are evenly distributed across the P tasks, i.e. each task should produce N^2/P result matrix elements.

The details about *how* the elements are distributed are important. Let us now assume column-wise distribution of the matrix: a range of *columns* with N^2/P matrix elements in total reside in a given task τ_i. To produce any column of $\widehat{C}$, all columns from the other $P-1$ tasks must be communicated to τ_i. This involves latency t_l and data transfer time t_t, total communication time for task i

$$T_{column} = (P-1)(t_l + t_t N^2/P) \tag{7.9}$$

This is the communication time to compute the N^2/P matrix elements of the result that reside in one task.

It turns out we can save a lot of time by distributing the matrix not column-wise, but block-wise. The main reason for advantage of this algorithm is that we can then *broadcast* pieces of matrix to all tasks rather than individually copying them to each task. Broadcasting is a $\log N$ type of algorithm.

Label the tasks τ_{ij} in the form of a $\sqrt{P} \times \sqrt{P}$ matrix with indices. Split the matrices into equal dimension blocks $\widehat{A}_{ij}$, $\widehat{B}_{ij}$, $\widehat{C}_{ij}$, each residing in one τ_{ij}.

Here is how the algorithm for computing one block $\widehat{C}_{mn}$ works

1. Initialize $\widehat{C}_{mn} = 0$
2. loop $k = 0, \sqrt{P} - 1$
   - Broadcast $\widehat{A}_{m,m+k}$ to all tasks $t_{m,j}, j = 0, 1, \ldots, \sqrt{P} - 1$ in row m. The index $m + k$ is to be understood as $m + k \mod \sqrt{P}$. Communication time is $(\log \sqrt{P})(t_l + t_t N^2/P)$. The task can (ideally) be performed within each m-row independently.
   - Copy $\widehat{B}_{m+k,n}$ to task τ_{mn}: communication time is $t_l + t_t N^2/P$. Can (ideally) be performed for all mn-blocks in parallel, without interference.
   - In each task τ_{mn} add $\widehat{C}_{mn} {+}{=} \widehat{A}_{m,m+k}\widehat{B}_{m+k,n}$ (perfectly parallel, no communication needed).

The communication time to compute the matrix elements residing in task τ_{mn} is

$$T_{2d} = \sqrt{P}(\log(\sqrt{P}) + 1)(t_l + t_t N^2/P) \approx \frac{\sqrt{P}\log P}{2}(t_l + t_t N^2/P) \tag{7.10}$$

This is a very significant gain on a machine with, say, $P = 1024$ CPUs: total communication time reduces by a factor $[32 \times 10/2]/[1024 - 1] \approx 0.15 \approx 1/7$.

Note that is only communication time, not the time of actual floating operations. Communication and floating operations have different scaling: communication is N^2, floating is N^3. The algorithm is only relevant, when communication overheads are larger or comparable in size to matrix operations. This may well be that case, when the individual blocks are just $\sim 100 \times 100$.

Let us assume $t_l = 1\mu s$, $t_t = 1$ complex number$/ns$, 10^{12} FLOPS and $N = 100$. Then compute time/block $\sim 100^3/10^{12}s = 1\mu s$. Communication time per block $10^4 \times 10^{-9} = 10\mu s$.

Many such tricks are built into standard software like ScaLAPACK, the “scalable LAPACK”, ore PETSc, wich is largely based on ScaLAPACK. Therefore, again, it is advisable to turn to standard packages where possible.

7.5 OpenMP

API (Application Programmer Interface) defined for Fortran (1997) and C (1998).

Controlled by compiler directives.
"Spontaneous" forking (creation of a multiple threads) and joining.
For shared memory machines: i.e. random access to all data of a coded is needed.
See codes `openmp.cpp` and `omp_dot.cpp` in "trunk/cpp" repository.
Compilers, as a rule, understand OpenMP directives. Compile with some flag, e.g.

```
linux>g++ -fopenmp openmp.cpp
linux>a.out
```

7.6 MPI - message passing interface

A demon in the background starts several jobs and controlles communication between them. The jobs may be running on quite different machines. Freely available package "MPICH2".
See codes `mpi.cpp` and `mpi_dot.cpp` in "trunk/cpp" repository.
Usage:

```
linux> mpic++ mpi_dot.cpp # compile and link script
linux> mpirun -np 2 a.out # run np=2 jobs
```

The compile / load scripts "mpic++", "mpif90", and the run script "mpirun" are created when installing MPICH2

Note: There exists a python interface to MPI: mpi4py

Finite-element time-propagation

7.6.1 Existing software

Writing parallel software is hard work in general. Computational methods sometimes are designed with the primary concern for the method to scale well upon parallelization (e.g. PIC - "Particle In Cell" for plasma simulations). However, when you encounter the need to parallelize, check available software first.

Unfortunately, unlike single processore linear algebra problems, there is no final software package for solving all standard tasks. Yet, there is an increasing number of good candidates out there, for example scaLAPACK, PETSc — "Portable, Extensible Toolkit for Scientific Computation". If you need to solve, e.g., a large eigenproblem in parallale, first turn to one of these packages.

FFT needs complicated communation patterns ("dont't try this at home"): there exists an MPI implementation of FFTW.

Note: There exists a python interface to PETSc: petsc4py

Chapter 8

Group theory on the computer: Young tableaus

As an exercise we will compute all irreducible representations of the permutation group on the space of n spins. In rare cases, this is useful by itself. In passing, it illustrates the meaning of "irreducible representation" in a direct way, teaches the correct use and interpretation of Young tableaus, and gives an exercise in decomposing linear space into a direct sum of subspaces.

- Define a Young *diagram* by specifying the total number of spins `n` and the difference between first and second row `m`.
- Get the dimension of the representation `nr` by the hook formula
- Fully define the Young *tableau* by specifying the number of unpaired spins s. Note: the z-component of the total Spin will be S_z =`s`/2.
- Fill in the young tablau with `1`'s for spin up and `0`'s for spin down according to the rules and parameters `n,m,s`. The sequence of zero's and ones gives a binary number k.
- Initialize the component vector `a[k]=1`, i.e. a single vector
- Obtain the standard vector corresponding to the Young tableau: symmetrize with respect to the rows, then anti-symmetrize for each column
- loop until `nr` orthogonal vectors are generated:
- permute to next permutation, orthogonalize vector relative to previous vectors
- if new vector is linearly independent (i.e. non-zero after orthogonalization) accept as new basis vector of irrep.

As tests we will

- Check whether dimensions `nr` for all `m,s` add up to 2^n, the complete spin space.

- Whether the orthonormal basis vectors span the complete Hilbert space.

$$1 = \sum_{m,s} \sum_{i} |b_i^{(m,s)}\rangle\langle b_i^{(m,s)}| \tag{8.1}$$

- Check spin eigenvector property