introduction to computational physics

basic concepts and approaches

frank zirkelbach

frank.zirkelbach@physik.uni-augsburg.de

experimantal physics IV - university of augsburg

outline

- motivation

outline

- motivation
- history of computing hardware/software

outline

- motivation
- history of computing hardware/software
- warning

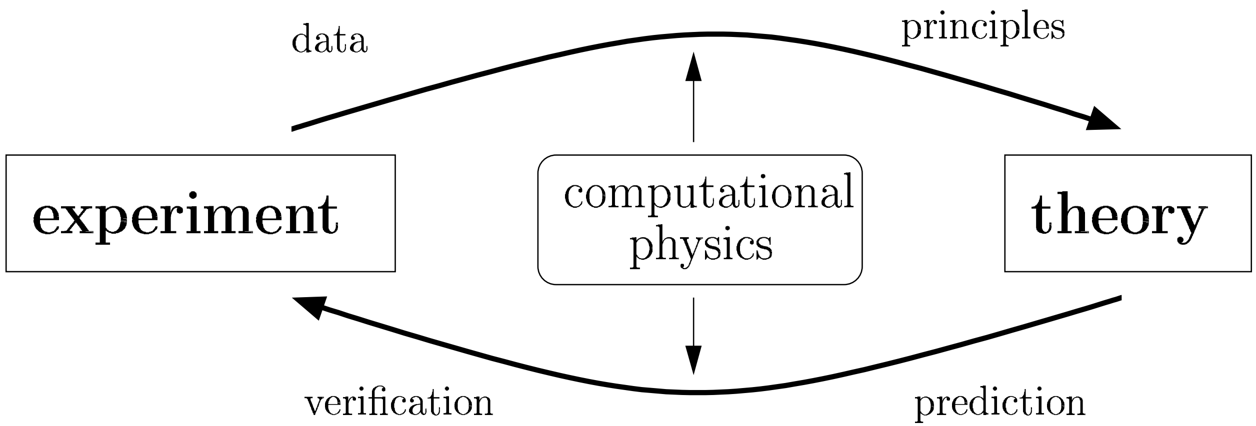
outline

- motivation
- history of computing hardware/software
- warning
- computational techniques

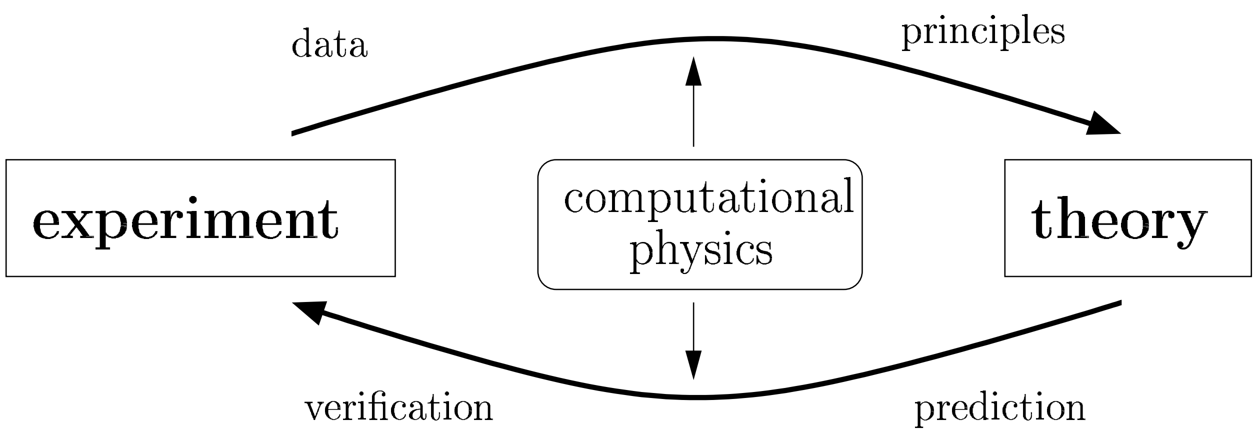
outline

- motivation
- history of computing hardware/software
- warning
- computational techniques
- summary

motivation


data
principles
experiment
computational physics
theory
verification
prediction

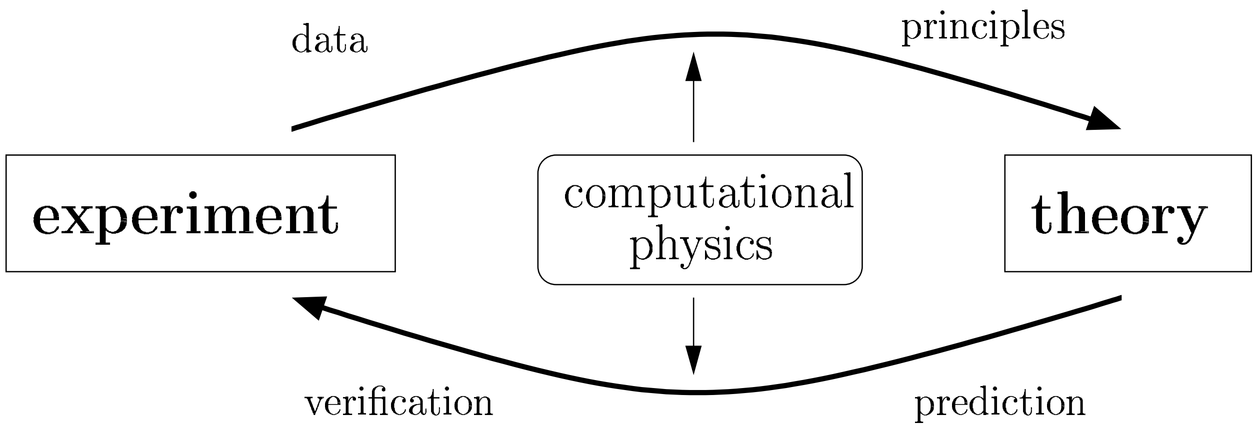
motivation



challenge:

- precise mathematical theory
- often: solving theory's equations ab-initio is not realistic
- only a few models can be solved exactly

motivation

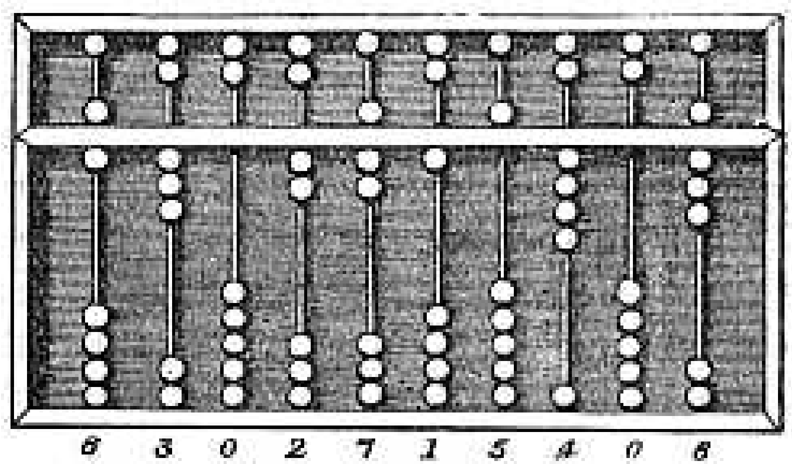


challenge:

- precise mathematical theory
- often: solving theory's equations ab-initio is not realistic
- only a few models can be solved exactly

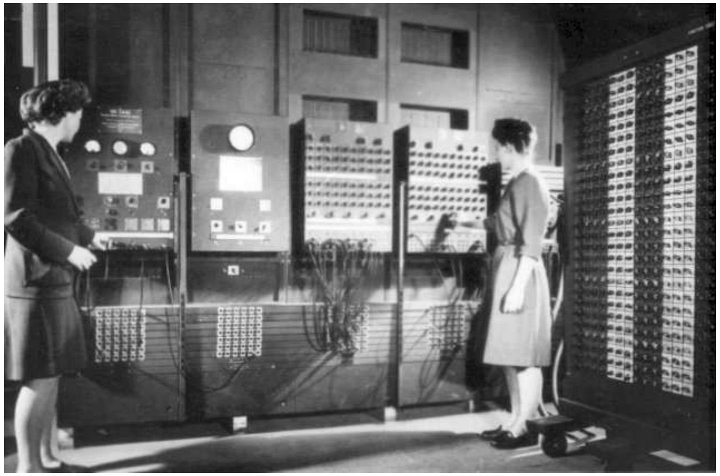
$\Rightarrow$ study and implementation of numerical algorithms

history of computing hardware



- $3000\,bc$: abacus - first calculating device

history of computing hardware

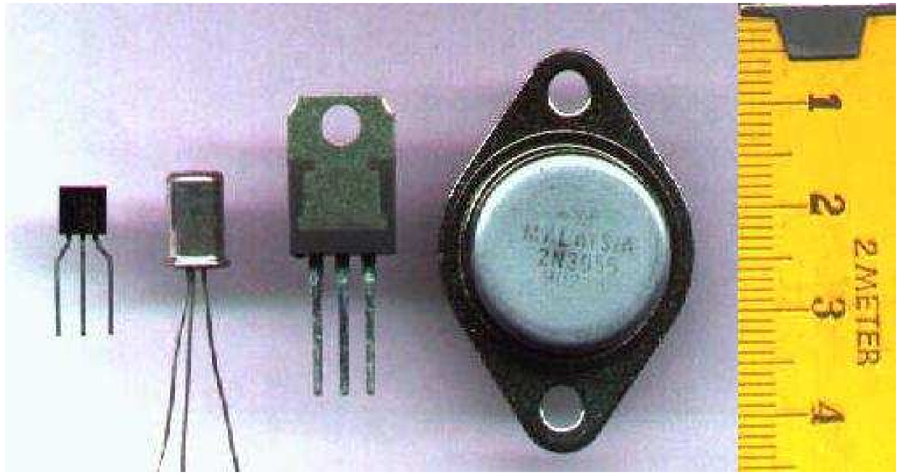


- $3000\,bc$: abacus - first calculating device
- 1945: eniac - electrical digital computer

history of computing hardware

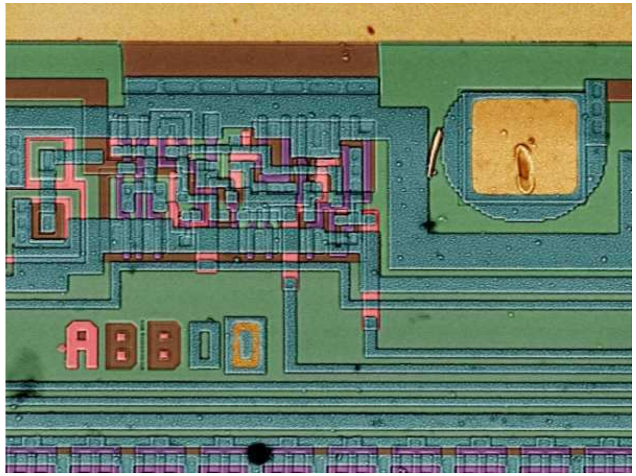
- $3000\,bc$: abacus - first calculating device
- 1945: eniac - electrical digital computer
- $1938/41$: z1/3 - featuring memory and programmability

history of computing hardware

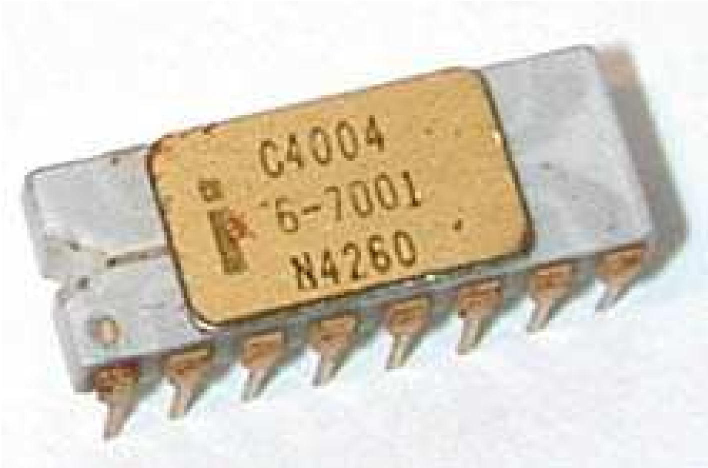
- $3000\,bc$: abacus - first calculating device
- 1945: eniac - electrical digital computer
- $1938/41$: z1/3 - featuring memory and programmability
- 1960: pdp-1 - transistor based computers

history of computing hardware



- $3000\,bc$: abacus - first calculating device
- 1945: eniac - electrical digital computer
- $1938/41$: z1/3 - featuring memory and programmability
- 1960: pdp-1 - transistor based computers
- 1964: pdp-8 - integrated circuit computers

history of computing hardware



- 1970: intel 4004 - first single chip μ-processor

history of computing hardware

- 1970: intel 4004 - first single chip μ-processor
- $1977/85$: cray1/2 - vector supercomputer

history of computing hardware

- 1970: intel 4004 - first single chip μ-processor
- $1977/85$: cray1/2 - vector supercomputer
- $1977/82/85$: 6502/6510/m68k - first pc

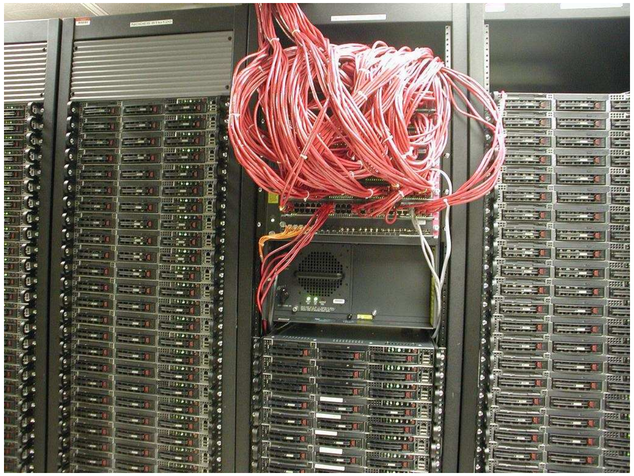
history of computing hardware

- 1970: intel 4004 - first single chip μ-processor
- $1977/85$: cray1/2 - vector supercomputer
- $1977/82/85$: 6502/6510/m68k - first pc
- $1978/82/85$: 8086/80286/80386

history of computing hardware

- 1970: intel 4004 - first single chip μ-processor
- $1977/85$: cray1/2 - vector supercomputer
- $1977/82/85$: 6502/6510/m68k - first pc
- $1978/82/85$: 8086/80286/80386
- 1985: mips - first risc design

history of computing hardware



- 1970: intel 4004 - first single chip μ-processor
- $1977/85$: cray1/2 - vector supercomputer
- $1977/82/85$: 6502/6510/m68k - first pc
- $1978/82/85$: 8086/80286/80386
- 1985: mips - first risc design
- $1990/2000$: massive parallel computing

history of computing software

- 1946: plankalkül - high-level programming language

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system
- 1969: unix - multics port to pdp-8, pdp-11/20

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system
- 1969: unix - multics port to pdp-8, pdp-11/20
- 1972: c programming language - thompson, ritchie

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system
- 1969: unix - multics port to pdp-8, pdp-11/20
- 1972: c programming language - thompson, ritchie
- 1978/84/85: apple os/atari, amiga os/mac os

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system
- 1969: unix - multics port to pdp-8, pdp-11/20
- 1972: c programming language - thompson, ritchie
- 1978/84/85: apple os/atari, amiga os/mac os
- 1981/85/92/95: ms-dos/windows 1.0/3.x/95

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system
- 1969: unix - multics port to pdp-8, pdp-11/20
- 1972: c programming language - thompson, ritchie
- 1978/84/85: apple os/atari, amiga os/mac os
- 1981/85/92/95: ms-dos/windows 1.0/3.x/95
- 1983: gnu project - unix-like free software development

history of computing software

- 1946: plankalkül - high-level programming language
- 1950: assembler - translating instruction mnemonics
- 1954: fortran - formula translation
- 1963: basic - beginner's all purpose symbolic instruction code
- 1964: os/360 - batch processing operating system
- 1969: unix - multics port to pdp-8, pdp-11/20
- 1972: c programming language - thompson, ritchie
- 1978/84/85: apple os/atari, amiga os/mac os
- 1981/85/92/95: ms-dos/windows 1.0/3.x/95
- 1983: gnu project - unix-like free software development
- 1991: linux - open-source kernel

warning - numerical errors

- machine accuracy ϵ_m

warning - numerical errors

- machine accuracy ϵ_m
  - ieee 64-bit floating point format: $v = -1^s 2^{-e} m$

s:	signe	1 bit
m:	mantissa	52 bit
e:	exponent	11 bit

warning - numerical errors

- machine accuracy ϵ_m
  - ieee 64-bit floating point format: $v = -1^s 2^{-e} m$

s:	signe	1 bit
m:	mantissa	52 bit
e:	exponent	11 bit

  - ϵ_m: smallest floating point with $1 + \epsilon_m \neq 1$
    $\epsilon_m \approx 2.22 \times 10^{-16}$ (roundoff error)

warning - numerical errors

- machine accuracy ϵ_m
  - ieee 64-bit floating point format: $v = -1^s 2^{-e} m$

s:	signe	1 bit
m:	mantissa	52 bit
e:	exponent	11 bit

  - ϵ_m: smallest floating point with $1 + \epsilon_m \neq 1$
    $\epsilon_m \approx 2.22 \times 10^{-16}$ (roundoff error)
- truncation error ϵ_t

warning - numerical errors

- machine accuracy ϵ_m
  - ieee 64-bit floating point format: $v = -1^s 2^{-e} m$

s:	signe	1 bit
m:	mantissa	52 bit
e:	exponent	11 bit

  - ϵ_m: smallest floating point with $1 + \epsilon_m \neq 1$
    $\epsilon_m \approx 2.22 \times 10^{-16}$ (roundoff error)
- truncation error ϵ_t
  - discrete approximation of continuous quantity

warning - numerical errors

- machine accuracy ϵ_m
  - ieee 64-bit floating point format: $v = -1^s 2^{-e} m$

s:	signe	1 bit
m:	mantissa	52 bit
e:	exponent	11 bit

  - ϵ_m: smallest floating point with $1 + \epsilon_m \neq 1$
    $\epsilon_m \approx 2.22 \times 10^{-16}$ (roundoff error)
- truncation error ϵ_t
  - discrete approximation of continuous quantity
  - persists even on hypothetical perfect computer ($\epsilon_m = 0$)

warning - numerical errors

- machine accuracy ϵ_m
  - ieee 64-bit floating point format: $v = -1^s 2^{-e} m$

s:	signe	1 bit
m:	mantissa	52 bit
e:	exponent	11 bit

  - ϵ_m: smallest floating point with $1 + \epsilon_m \neq 1$
    $\epsilon_m \approx 2.22 \times 10^{-16}$ (roundoff error)
- truncation error ϵ_t
  - discrete approximation of continuous quantity
  - persists even on hypothetical perfect computer ($\epsilon_m = 0$)
  - machine independent, characteristic of used algorithm

warning - recursive functions

- avoid recursive functions!

warning - recursive functions

- avoid recursive functions!

```
int fak(n) {
    if(n==1) return 1;
    else return n*fak(n-1);
}
```

warning - recursive functions

- avoid recursive functions!

```
int fak(n) {
    if(n==1) return 1;
    else return n*fak(n-1);
}
```

- better:

warning - recursive functions

- avoid recursive functions!

```
int fak(n) {
    if(n==1) return 1;
    else return n*fak(n-1);
}
```

- better:

```
int fak(n) {
    int help=1;
    while(n>1) {
      help=help*n;
      n=n-1;
    }
return help;
}
```

computational techniques

- rough discretization
- solution of linear algebraic equations
- interpolation and extrapolation
- integration of functions
- evaluation of (special) functions
- monte carlo methods
- eigensystems
- spectral applications
- modeling of data
- ordinary differential equations
- two point boundary value problems
- partial differential equations

---

http://www.nr.com/

first steps: rough discretization

- example: homogenous field of force $\vec{F} = (0, -mg)$

  equation of motion: $\vec{F} = m\vec{a} = m\frac{d^2\vec{r}}{dt^2}$

  initial condition: $\vec{r}(t=0) = \vec{r_0} = (x_0, y_0)$

  $\frac{d\vec{r}}{dt}|_{t=0} = (v_{x_0}, v_{y_0})$

first steps: rough discretization

- example: homogenous field of force $\vec{F} = (0, -mg)$

  equation of motion: $\vec{F} = m\vec{a} = m\frac{d^2\vec{r}}{dt^2}$

  initial condition: $\vec{r}(t=0) = \vec{r_0} = (x_0, y_0)$

  $\frac{d\vec{r}}{dt}|_{t=0} = (v_{x_0}, v_{y_0})$

- algorithm using discretized time ($T = N\tau$):

  $x^1 = x_0;\quad y^1 = y_0;$

  $v_x^1 = v_{x_0};\quad v_y^1 = v_{y_0};$

  loop:

  $x^2 = x^1 + \tau v_x^1;\quad y^2 = y^1 + \tau v_y^1;$

  $v_x^2 = v_x^1;\quad v_y^2 = v_y^1 - g\tau;$

  $x^1 = x^2;\quad y^1 = y^2$

  $v_x^1 = v_x^2;\quad v_y^1 = v_y^2;$

first steps: rough discretization

- example: homogenous field of force $\vec{F} = (0, -mg)$

  equation of motion: $\vec{F} = m\vec{a} = m\frac{d^2\vec{r}}{dt^2}$

  initial condition: $\vec{r}(t=0) = \vec{r_0} = (x_0, y_0)$

  $\frac{d\vec{r}}{dt}|_{t=0} = (v_{x_0}, v_{y_0})$

- algorithm using discretized time ($T = N\tau$):

  $x^1 = x_0; \quad y^1 = y_0;$

  $v_x^1 = v_{x_0}; \quad v_y^1 = v_{y_0};$

  loop:

  $x^2 = x^1 + \tau v_x^1; \quad y^2 = y^1 + \tau v_y^1;$

  $v_x^2 = v_x^1; \quad v_y^2 = v_y^1 - g\tau;$

  $x^1 = x^2; \quad y^1 = y^2$

  $v_x^1 = v_x^2; \quad v_y^1 = v_y^2;$

- euler's method for solving o.d.e.

euler's method: error estimation

- truncation error ϵ_t:

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t, x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t, x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:
  - every floating point step: error of $O(\epsilon_m)$

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:
  - every floating point step: error of $O(\epsilon_m)$
  - $O(\tau^{-1})$ steps $\Rightarrow$ error of $O(\frac{\epsilon_m}{\tau})$

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:
  - every floating point step: error of $O(\epsilon_m)$
  - $O(\tau^{-1})$ steps $\Rightarrow$ error of $O(\frac{\epsilon_m}{\tau})$
- optimum:

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:
  - every floating point step: error of $O(\epsilon_m)$
  - $O(\tau^{-1})$ steps $\Rightarrow$ error of $O(\frac{\epsilon_m}{\tau})$
- optimum:
  - $\epsilon \sim \frac{\epsilon_m}{\tau} + \tau$

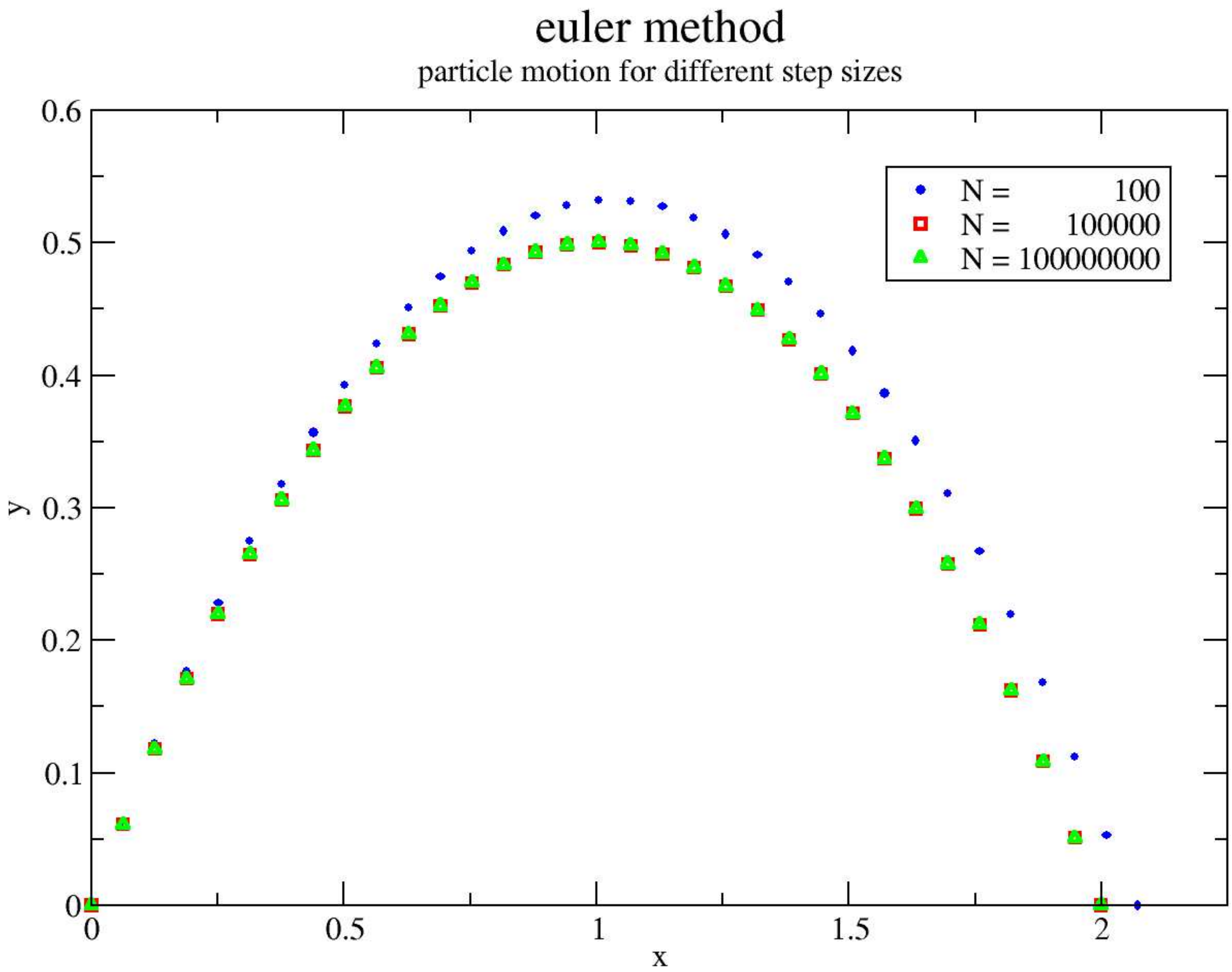
euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:
  - every floating point step: error of $O(\epsilon_m)$
  - $O(\tau^{-1})$ steps $\Rightarrow$ error of $O(\frac{\epsilon_m}{\tau})$
- optimum:
  - $\epsilon \sim \frac{\epsilon_m}{\tau} + \tau$
  - 64-bit: $\epsilon_m \sim 10^{-16} \Rightarrow \tau \sim 10^{-8}$

euler's method: error estimation

- truncation error ϵ_t:
  - $x_{t+\tau} = x(t) + v(t,x)\tau + O(\tau^2)$
  - period T: $O(\tau^{-1})$ steps $\Rightarrow \epsilon_t \sim O(\tau)$
- machine accuracy:
  - every floating point step: error of $O(\epsilon_m)$
  - $O(\tau^{-1})$ steps $\Rightarrow$ error of $O(\frac{\epsilon_m}{\tau})$
- optimum:
  - $\epsilon \sim \frac{\epsilon_m}{\tau} + \tau$
  - 64-bit: $\epsilon_m \sim 10^{-16} \Rightarrow \tau \sim 10^{-8}$
  - 32-bit: $\epsilon_m = 1.19 \times 10^{-7} \Rightarrow \tau \sim 3 \times 10^{-4}$

euler's method: accuracy



monte carlo methods

- algorithms for solving computational problems using random numbers

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:
  - monte carlo integration

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:
  - monte carlo integration
  - metropolis algorithm

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:
  - monte carlo integration
  - metropolis algorithm
  - simulated annealing

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:
  - monte carlo integration
  - metropolis algorithm
  - simulated annealing
- advantages:

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:
  - monte carlo integration
  - metropolis algorithm
  - simulated annealing
- advantages:
  - more efficient than other methods

monte carlo methods

- algorithms for solving computational problems using random numbers
- deterministic pseudo-random sequences
- applications:
  - monte carlo integration
  - metropolis algorithm
  - simulated annealing
- advantages:
  - more efficient than other methods
  - no need fo simplifying assumptions

random number generator

linear congruential generator:

- $I_{j+1} = (aI_j + c)\, mod\, m$
  a: multiplier, c: increment
  m: modulus, I_0: seed

random number generator

linear congruential generator:

- $I_{j+1} = (aI_j + c)\, mod\, m$
  a: multiplier, c: increment
  m: modulus, I_0: seed
- minimal standard by park and miller:
  $a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647, \quad c = 0$

random number generator

linear congruential generator:

- $I_{j+1} = (aI_j + c)\, mod\, m$
  a: multiplier, c: increment
  m: modulus, I_0: seed
- minimal standard by park and miller:
  $a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647, \quad c = 0$
- always seed the rng

random number generator

linear congruential generator:

- $I_{j+1} = (aI_j + c)\, mod\, m$
  a: multiplier, c: increment
  m: modulus, I_0: seed
- minimal standard by park and miller:
  $a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647, \quad c = 0$
- always seed the rng

$\Rightarrow$ sequence of integers $\in [0, m[$

random number generator

linear congruential generator:

- $I_{j+1} = (aI_j + c)\, mod\, m$
  a: multiplier, c: increment
  m: modulus, I_0: seed
- minimal standard by park and miller:
  $a = 7^5 = 16807, \quad m = 2^{31} - 1 = 2147483647, \quad c = 0$
- always seed the rng

$\Rightarrow$ sequence of integers $\in [0, m[$
division by modulus $\Rightarrow$ uniform deviates :

$$p(x)dx = \begin{cases} dx & 0 \leq x < 1 \\ 0 & \text{sonst} \end{cases}$$

special deviates

- transformation method:

special deviates

- transformation method:
  - arbitrary probability distribution $\rho(y)$

special deviates

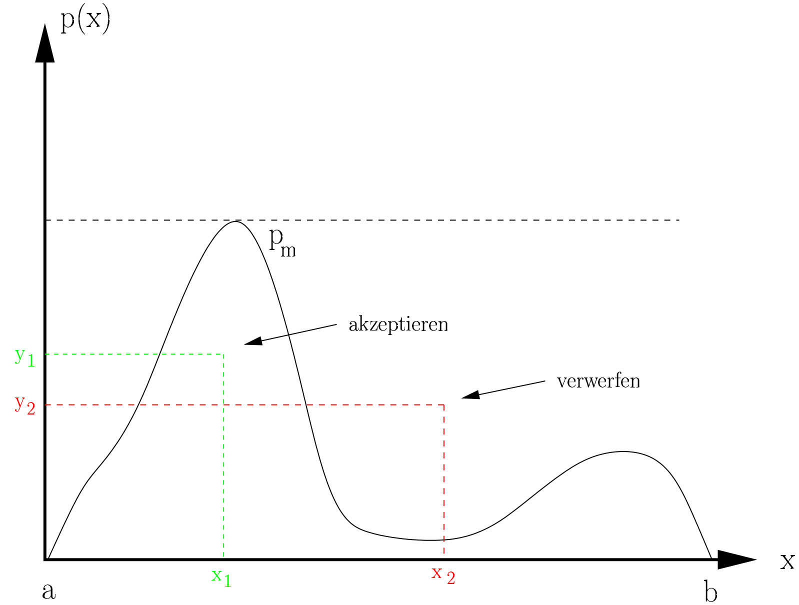
- transformation method:
  - arbitrary probability distribution $\rho(y)$
  - trafo: $p(x)dx = \rho(y)dy \Rightarrow x = \int_{-\infty}^{y} \rho(y)dy$

special deviates

- transformation method:
  - arbitrary probability distribution $\rho(y)$
  - trafo: $p(x)dx = \rho(y)dy \Rightarrow x = \int_{-\infty}^{y} \rho(y)dy$
  - get inverse of $x(y) \Rightarrow y(x)$

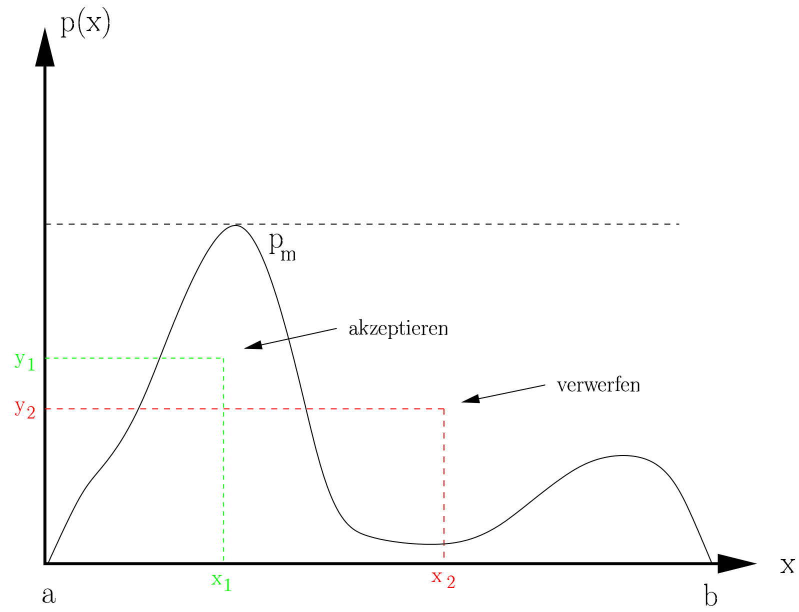
special deviates

- transformation method:
  - arbitrary probability distribution $\rho(y)$
  - trafo: $p(x)dx = \rho(y)dy \Rightarrow x = \int_{-\infty}^{y} \rho(y)dy$
  - get inverse of $x(y) \Rightarrow y(x)$
- rejection method:



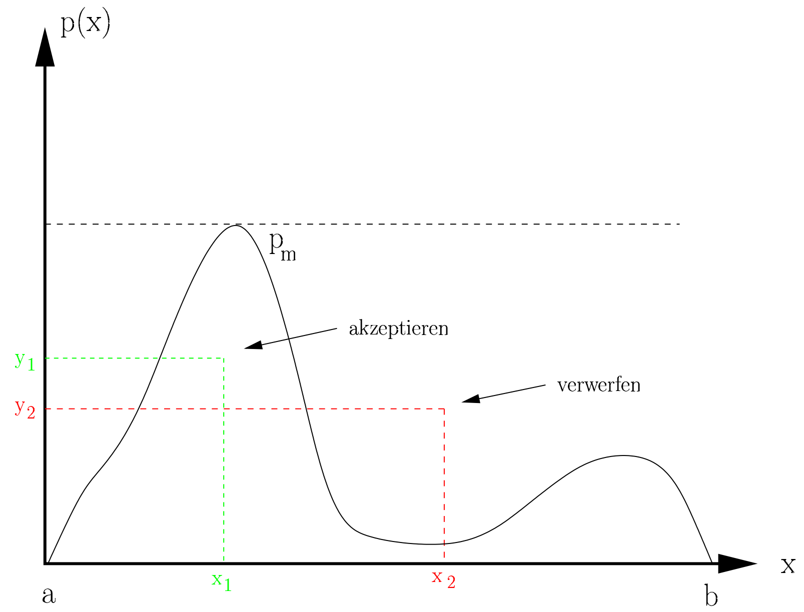
special deviates

- transformation method:
  - arbitrary probability distribution $\rho(y)$
  - trafo: $p(x)dx = \rho(y)dy \Rightarrow x = \int_{-\infty}^{y} \rho(y)dy$
  - get inverse of $x(y) \Rightarrow y(x)$
- rejection method:
  - $p(x) \in [a, b]$ mit
    $p(x) \geq 0 \quad \forall x \in [a, b]$



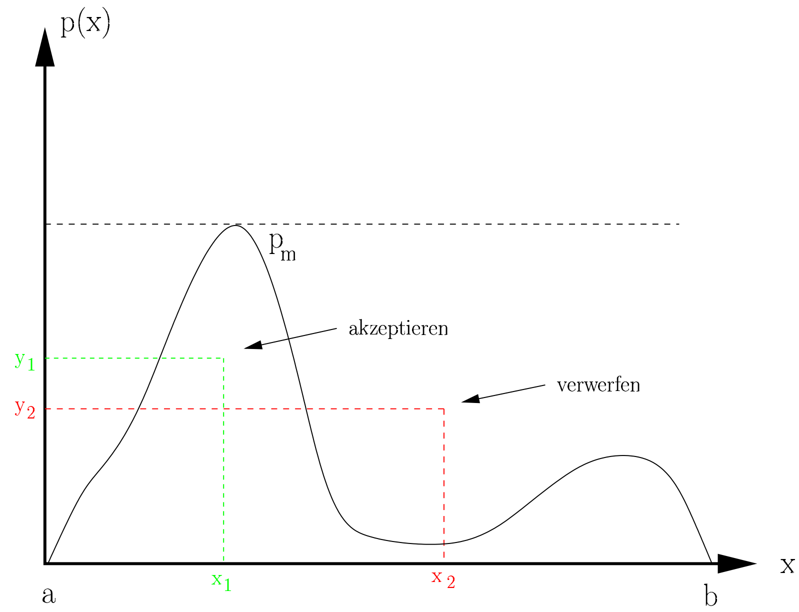
special deviates

- transformation method:
  - arbitrary probability distribution $\rho(y)$
  - trafo: $p(x)dx = \rho(y)dy \Rightarrow x = \int_{-\infty}^{y} \rho(y)dy$
  - get inverse of $x(y) \Rightarrow y(x)$
- rejection method:
  - $p(x) \in [a, b]$ mit
    $p(x) \geq 0 \quad \forall x \in [a, b]$
  - uniformly distributed
    $x \in [a, b]$ und
    $y \in [0, p_m]$



special deviates

- transformation method:
  - arbitrary probability distribution $\rho(y)$
  - trafo: $p(x)dx = \rho(y)dy \Rightarrow x = \int_{-\infty}^{y} \rho(y)dy$
  - get inverse of $x(y) \Rightarrow y(x)$
- rejection method:
  - $p(x) \in [a, b]$ mit
    $p(x) \geq 0 \quad \forall x \in [a, b]$
  - uniformly distributed
    $x \in [a, b]$ und
    $y \in [0, p_m]$
  - if $y \leq p(x)$ use x, else reject x



monte carlo integration

basics:

- $I = \int_\Omega f d\Omega$

monte carlo integration

basics:

- $I = \int_\Omega f d\Omega$
- instead of regular x_i, choose them at random

monte carlo integration

basics:

- $I = \int_\Omega f d\Omega$
- instead of regular x_i, choose them at random
- $I \approx \Omega < f > \pm \Omega \sqrt{\frac{<f^2> - <f>^2}{N}}$

  $< f >= \frac{1}{N} \sum_{i=1}^{N} f(\vec{x_i})$

  $< f^2 >= \frac{1}{N} \sum_{i=1}^{N} f^2(\vec{x_i})$

monte carlo integration

basics:

- $I = \int_\Omega f d\Omega$
- instead of regular x_i, choose them at random
- $I \approx \Omega < f > \pm \Omega \sqrt{\frac{<f^2> - <f>^2}{N}}$

  $< f >= \frac{1}{N} \sum_{i=1}^{N} f(\vec{x_i})$

  $< f^2 >= \frac{1}{N} \sum_{i=1}^{N} f^2(\vec{x_i})$

example: gambling for π

monte carlo integration

basics:

- $I = \int_\Omega f d\Omega$
- instead of regular x_i, choose them at random
- $I \approx \Omega < f > \pm \Omega \sqrt{\frac{<f^2> - <f>^2}{N}}$

  $< f >= \frac{1}{N} \sum_{i=1}^{N} f(\vec{x_i})$

  $< f^2 >= \frac{1}{N} \sum_{i=1}^{N} f^2(\vec{x_i})$

example: gambling for π

$$\pi = \int_{-1}^{1} \int_{-1}^{1} p(x,y) dx dy \approx \frac{4}{N} \sum_{i=1}^{N} p(x_i, y_i)$$

$$\text{with } p(x,y) = \begin{cases} 1 & x^2 + y^2 \leq 1 \\ 0 & \text{sonst} \end{cases}$$

metropolis algorithm

ising model:

- d-dimensional periodic lattice

metropolis algorithm

ising model:

- d-dimensional periodic lattice
- two possible states for magnetic moment at site i:
  $\mu_i = \mu S_i \qquad S_i = \pm 1 \quad \forall i$

metropolis algorithm

ising model:

- d-dimensional periodic lattice
- two possible states for magnetic moment at site i:
  $\mu_i = \mu S_i \qquad S_i = \pm 1 \quad \forall i$
- nearest neighbors moments interact,
  interaction strength $\frac{J_{ij}}{\mu^2}$

metropolis algorithm

ising model:

- d-dimensional periodic lattice
- two possible states for magnetic moment at site i:
  $\mu_i = \mu S_i \qquad S_i = \pm 1 \quad \forall i$
- nearest neighbors moments interact,
  interaction strength $\frac{J_{ij}}{\mu^2}$

$\Rightarrow$ hamiltonian: $H = -\sum_{(i,j)} J_{ij} S_i S_j$

metropolis algorithm

ising model:

- d-dimensional periodic lattice
- two possible states for magnetic moment at site i:
  $\mu_i = \mu S_i \qquad S_i = \pm 1 \quad \forall i$
- nearest neighbors moments interact,
  interaction strength $\frac{J_{ij}}{\mu^2}$

$\Rightarrow$ hamiltonian: $H = -\sum_{(i,j)} J_{ij} S_i S_j$

partition function:

$$Z = \sum_{i=1}^{N} e^{\frac{-E_i}{k_B T}} = Tr(e^{-\beta H})$$

metropolis algorithm

- importance sampling:
  $< A >= \sum_i p_i A_i \approx \frac{1}{N} \sum_{i=1}^{N} A_i$, with

$$p_i = \frac{e^{-\beta E_i}}{Z}$$

metropolis algorithm

- importance sampling:
  $< A >= \sum_i p_i A_i \approx \frac{1}{N} \sum_{i=1}^{N} A_i$, with

  $$p_i = \frac{e^{-\beta E_i}}{Z}$$

- detailed balance

  sufficient condition for equilibrium:

  $$W(A \to B)p(A) = W(B \to A)p(B)$$

  $\Rightarrow \frac{W(A \to B)}{W(B \to A)} = \frac{p(B)}{p(A)} = e^{\frac{-\Delta E}{k_B T}}$

  with $\Delta E = E(B) - E(A)$

metropolis algorithm

- choose W:

$$W(A \to B) = \begin{cases} e^{-\beta \Delta E} & : \Delta E > 0 \\ 1 & : \Delta E < 0 \end{cases}$$

metropolis algorithm

- choose W:

$$W(A \to B) = \begin{cases} e^{-\beta \Delta E} & : \Delta E > 0 \\ 1 & : \Delta E < 0 \end{cases}$$

- algorithm:

metropolis algorithm

- choose W:

$$W(A \to B) = \begin{cases} e^{-\beta \Delta E} & : \Delta E > 0 \\ 1 & : \Delta E < 0 \end{cases}$$

- algorithm:
  - visit every lattice site

metropolis algorithm

- choose W:

$$W(A \to B) = \begin{cases} e^{-\beta \Delta E} & : \Delta E > 0 \\ 1 & : \Delta E < 0 \end{cases}$$

- algorithm:
  - visit every lattice site
  - calculate ΔE for spin flip

metropolis algorithm

- choose W:

$$W(A \to B) = \begin{cases} e^{-\beta \Delta E} & : \Delta E > 0 \\ 1 & : \Delta E < 0 \end{cases}$$

- algorithm:
  - visit every lattice site
  - calculate ΔE for spin flip
  - flip spin if $r \leq e^{\frac{-\Delta E}{k_B T}}$

summary

- importance of computational physics
- things to keep in mind when doing computational physics
- euler's method for solving o.d.e.
- introduction to monte carlo methods