

# Formidabelark FYS4150

## Innhold

|     |                                  |   |
|-----|----------------------------------|---|
| 1.1 | Lineær algebra og sånn . . . . . | 1 |
| 1.2 | ODE . . . . .                    | 2 |
| 1.3 | PDE . . . . .                    | 3 |
| 1.4 | Monte Carlo-metoder . . . . .    | 4 |
| 1.5 | Numerisk integrasjon . . . . .   | 4 |
| 1.6 | Statistisk fysikk . . . . .      | 5 |

Hei

## 1.1 Lineær algebra og sånn

**Diskretisering** av  $-\frac{d^2 u(x)}{dx^2} = f(x, u(x))$ . Bruk def. av derivert, det gir.  $u_i'' \approx (u_{i+1} - 2u_i + u_{i-1})/h^2$ . Kan skrives som tridiagonal matrise

$$\mathbf{A} = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \cdots & \cdots & \cdots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix}$$

Gir likninga  $\mathbf{A}\mathbf{u} = \mathbf{f}(\mathbf{u})$ . Som vi kan skrive som  $a_i u_{i-1} + b_i u_i + c_i u_{i+1} = f_i$ . Først forover substitusjon,

```
btemp = b[1];
u[1] = f[1]/btemp;
for(i=2 ; i <= n ; i++) {
    temp[i] = c[i-1]/btemp;
    btemp = b[i]-a[i]*temp[i];
    u[i] = (f[i] - a[i]*u[i-1])/btemp;
```

deretter bakover substitusjon

```
for(i=n-1 ; i >= 1 ; i--) {
    u[i] -= temp[i+1]*u[i+1];
```

Lurt å plote feil fra denne metoden som  $\epsilon_i = \log(|v_i - u_i|/|u_i|)$ , hvor  $u_i$  er den analytiske løsninga i punkt  $i$ . **FLOPS** for de forskjellige metodene

- Radredusiering  $2n^3/n$
- LU-dekomp.  $2n^3/3$

- Tridiagonal løser  $8n$

- QR  $4n^3/3$

**LU-dekomponering** Likninga  $\mathbf{Ax} = \mathbf{w}$ , kan skrives som  $\mathbf{Ax} \equiv \mathbf{LUx} = \mathbf{w}$ . Dette kan regnes i to steg

$\mathbf{Ly} = \mathbf{w}$ ;  $\mathbf{Ux} = \mathbf{y}$ , hvor vi har  $\mathbf{y} = \mathbf{Ux} = \mathbf{L}^{-1}\mathbf{w}$ .

**Jacobis metode** Likninga  $\hat{A}\mathbf{x} = \mathbf{b}$  løses ved å bruke  $\mathbf{x}^{(k+1)} = \hat{D}^{-1}(\mathbf{b} - (\hat{L} + \hat{U})\mathbf{x}^{(k)})$ , hvor  $D$  er en diagonal matrise og  $L$  og  $U$  er nedre og øvre triangulære matriser,  $\mathbf{x}^{(k)}$  er et gjett på løsninga. Hvis matrisa  $A$  er positiv definit eller dominant på diagonalen kan man vise at denne metoden alltid vil konvergere. Vi implementerer metoden på denne måten

- Definer en toleranse for når ikke-diagonale elementer er null
- Sammenlikn største ikke-diagonale element med toleransen, hvis større enn toleranse må man fortsette å iterere
- Velg største ikke-diag. element og regn ut rotasjonsvinkelen etter den
- Foreta rotasjonen
- Fortsett til maks element i  $A \leq \epsilon$

**Eigenverdier og ‘similarity’transformering** Man kan finne eigenverdiene av  $A$  ved å bruke

$\mathbf{B} = \mathbf{S}^T \mathbf{A} \mathbf{S}$ ,  $\mathbf{S}^T \mathbf{S} = \mathbf{S}^{-1} \mathbf{S} = \mathbf{I}$ . Gjør man det mange nok ganger ender man opp med

$\mathbf{S}_1^T \dots \mathbf{S}_1^T \mathbf{A} \mathbf{S}_1 \dots \mathbf{S}_N = \mathbf{D}$ , hvor  $D$  har eigenverdiene til  $A$  på diagonalen. Eigenverdiene endres ikke av denne transformasjonen:  $\mathbf{Ax} = \lambda \mathbf{x} \Rightarrow (\mathbf{S}^T \mathbf{A} \mathbf{S})(\mathbf{S}^T \mathbf{x}) = \lambda \mathbf{S}^T \mathbf{x}$ , men vi ser at egenvektorene endres!  $\rightarrow$  må rotere egenvektormatrisa motsatt vei for å få de “ekte” egenvektorene (prosjekt 1?).

(Jacobis metode for egenverdier) Vi setter

$$\mathbf{S} = \begin{pmatrix} 1 & 0 & \dots & 0 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 0 & \dots \\ 0 & 0 & \dots & \cos \theta & 0 & \dots & 0 & \sin \theta \\ 0 & 0 & \dots & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & 1 & \dots \\ 0 & 0 & \dots & -\sin \theta & 0 & \dots & 0 & \cos \theta \end{pmatrix}$$

Algoritmisk: 1. Velg toleranse  $\epsilon \sim 10^{-8}$ . 2. Lag en while-løkke som går så lenge  $\max(a_{ij}^2) \geq \epsilon$  for  $i \neq j$ . 3. Bruk det største ikke-diag. elementet  $|a_{kl}| = \max |a_{ij}|$  ( $i \neq j$ ) til å regne ut  $\tau = (a_{ll} - a_{kk})/2a_{kl}$ , hvor  $\tan \theta = -\tau \pm \sqrt{1 + \tau^2}$ . Velg den løsningen som gir minste rotasjon for å forhindre at andre elementer blir forskjøvet langt fra null. 4. Gjennomfør likhetstransformasjonen  $\mathbf{B} = \mathbf{S}(\mathbf{k}, \mathbf{l}, \theta)^T \mathbf{A} \mathbf{B}(\mathbf{k}, \mathbf{l}, \theta)$ , kan gjøres på en lur måte. 5. Fortsett til  $\max(a_{ij}^2) \leq \epsilon$ . Krever  $3n^2 - 5n^2$  rotasjoner som hver krever  $4n$  operasjoner  $\rightarrow 12n^3 - 15n^3$  FLOPS. ULEMPE: uvisst hvor mange iterasjoner man trenger. Fröbeniusnormen er alltid bevart ved ortogonale transformasjoner, dette sikrer konvergens for denne metoden.

**Householders algoritme** er bedre enn Jacobis metode! Starter med  $\mathbf{S} = \mathbf{S}_1 \mathbf{S}_2 \dots \mathbf{S}_{n-2}$ , der  $S$  er en orthogonal matrise. Man lar den virke på hver side av  $A$  og vi ender opp med

$$\mathbf{S}^T \mathbf{A} \mathbf{S} = \begin{pmatrix} a_{11} & e_1 & 0 & 0 & \dots & 0 & 0 \\ e_1 & a'_{22} & e_2 & 0 & \dots & 0 & 0 \\ 0 & e_2 & a''_{33} & e_3 & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & \dots & \dots & \dots & \dots & a_{n-2}^{(n-1)} & e_{n-1} \\ 0 & \dots & \dots & \dots & \dots & e_{n-1} & a_{nn}^{(n-1)} \end{pmatrix}.$$

Vi skriver rotasjonsmatrisa som

$$\mathbf{S}_1 = \begin{pmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & \mathbf{P} \end{pmatrix},$$

der  $\mathbf{P} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^T$ .  $\mathbf{u}$  er en vektor vi må finne.

$$\mathbf{S}_1^T \mathbf{A} \mathbf{S}_1 = \begin{pmatrix} a_{11} & (\mathbf{P}\mathbf{v})^T \\ \mathbf{P}\mathbf{v} & \mathbf{A}' \end{pmatrix}.$$

$\mathbf{v}$  er vektoren med elementene i første rad og kolonne i  $A$ . Vi må også ha at  $(\mathbf{P}\mathbf{v})^T = (k, 0, \dots)$ . Vi har også bruk for

$$(\mathbf{P}\mathbf{v})^T \mathbf{P}\mathbf{v} = k^2 = \mathbf{v}^T \mathbf{v} = |\mathbf{v}|^2 = \sum_{i=2}^n a_{i1}^2,$$

som gir oss at  $k = \pm v$ .

$$\mathbf{P}\mathbf{v} = \mathbf{v} - 2\mathbf{u}(\mathbf{u}^T \mathbf{v}) = k\mathbf{e}, \quad (1)$$

Som vi kan skrive som  $\mathbf{v} - k\mathbf{e} = 2\mathbf{u}(\mathbf{u}^T \mathbf{v})$ , opphører vi dette i andre får vi  $2(\mathbf{u}^T \mathbf{v})^2 = (v^2 \pm a_{21}v)$ , (NB: velg fortegnet som gir størst verdi for å unngå numerisk tap) som så settes inn i

$$\mathbf{u} = \frac{\mathbf{v} - k\mathbf{e}}{2(\mathbf{u}^T \mathbf{v})}.$$

Dette settes så inn i  $P$  og vi kan rotere ferdig. Denne prosessen gjentas  $(n-1)$  ganger for en  $n \times n$ -matrise.  $<3$

**Lanczos algoritme** brukes på symmetriske egenverdiproblemer. Vi har matrisa  $A$ . Algoritmen genererer en sekvens med reelle tridiagonale matriser  $T_k$  med dim.  $k \times k$  ( $k \leq n$ ), s.a. ekstremalegenverdiene til  $T_k$  blir bedre og bedre estimater av egenverdiene til  $A$ . Metoden bruker en likhetstranf.  $T = Q^T A Q$ .

**Splineinterpolasjon** er en metode for å interpolere mellom datapunkter. Mest brukt er kubiske spliner, altså polynomer av grad tre som binder sammen datapunktene. Har man et intervall  $[x_0, x_n]$  med datapunkter å interpolere så har vi  $n$  polynomer av typen  $s_i(x) = a_{i0} + a_{i1}x + a_{i2}x^2 + a_{i3}x^3$ . Antar at de deriverte er kontinuerlige, s.a.  $s'_{i-1}(x_i) = s'_i(x_i)$  og  $s''_{i-1}(x_i) = s''_i(x_i)$ , dette gir oss  $4n$  koeff. å finne og  $4n-2$  likn. å løse. Vi setter  $s''_i(x_i) \equiv f_i$  og  $s''_i(x_{i+1}) \equiv f_{i+1}$ , trekker vi en rett linje mellom  $f_i$  og  $f_{i+1}$  så får vi uttrykket

$$s''_i(x) = \frac{f_i}{x_{i+1} - x_i}(x_{i+1} - x) + \frac{f_{i+1}}{x_{i+1} - x_i}(x - x_i),$$

som kan integreres to ganger, da får vi et tredjeordens polynom hvor vi kan finne integrasjonskonstantene ved å bruke  $s_i(x_i) = y_i$  og  $s_i(x_{i+1}) = y_{i+1}$ , uttrykket vi ender opp med kan så skrives om ved å bruke kontinuitetsbetingelsene og vi vil ende opp med en tridiagonal matrise som vi kan løse. Voilà, vi har interpolert.

## 1.2 ODE

Essensiell ting å huske: "finite difference"-metoder og gaussisk kvadratur hvor steglengde varierer og punkter vektet forskjellig.

**Omskrivning** av ODE. Newtons 2. lov (fjærsystem):  $m\ddot{x} = -kx$ . Setter  $x(t) \equiv y^{(1)}(t)$  og  $v(t) \equiv y^{(2)}(t)$ . Det gir oss

$$m\dot{y}^{(2)}(t) = -ky^{(1)}(t) \quad \dot{y}^{(1)}(t) = y^{(2)}(t).$$

**Eulers metode** Taylorutvikling gir

$x_{i+1} = x(t = t_i + h) = x(t_i) + hx'(t_i) + O(h^2)$ , hvor  $x'(t) = v(t)$ . Total feil blir, siden man summerer over alle steg  $N = (b - a)/h$ ,  $NO(h^2) \approx O(h)$ .

**Runge-Kutta** (2. orden) Definer  $\frac{dy}{dt} = f(t, y)$ ,

$$y(t) = \int f(t, y) dt, \quad y_{i+1} = y_i + \int_{t_i}^{t_{i+1}} f(t, y) dt.$$

Taylorutvikler  $f(t, y)$  om midtpunktet til

integrasjonsintervallet,  $t_i + h/2$ , vi får

$$\int_{t_i}^{t_{i+1}} f(t, y) dt \approx hf(t_{i+1/2}, y_{i+1/2}) + O(h^3)$$

som gir  $y_{i+1} = y_i + hf(t_{i+1/2}, y_{i+1/2}) + O(h^3)$ . Vi vet ikke  $y_{i+1/2}$ , den er

$$y_{(i+1/2)} = y_i + \frac{h}{2} \frac{dy}{dt} = y(t_i) + \frac{h}{2} f(t_i, y_i)$$

Vi ender opp med  $k_1 = hf(t_i, y_i)$  og

$k_2 = hf(t_{i+1/2}, y_i + k_1/2)$  som gir

$y_{i+1} \approx y_i + k_2 + O(h^3)$ . Man regner m.a.o. et par mellomsteg for å oppnå et bedre estimat. Fjerde ordens RK bruker

$k_1 = hf(t_i, y_i)$ ,  $k_2 = hf(t_i + h/2, y_i + k_1/2)$ ,  $k_3 = hf(t_i + h/2, y_i + k_2/2)$ ,  $k_4 = hf(t_i + h, y_i + k_3)$  som gir  $y_{i+1} = y_i + \frac{1}{6} (k_1 + 2k_2 + 2k_3 + k_4)$ . Feil:  $O(h^4)$ .

Geometrisk tolkning:  $k_1$  beregner stigninga i  $t_i$ ,  $k_2$  beregner stigninga i  $t_{i+1/2}$ ,  $k_3$  beregner stigninga i  $t_{i+1/2}$  ved hjelp av  $k_2$ ,  $k_4$  gjør et anslag på stigninga i  $t_{i+1}$ .

**Adaptive metoder** steglengden varier ettersom hvor mye funksjonen endrer seg, man sjekker om estimert verdi er innafor en viss toleranse og man avgjør så om steglengden skal øke eller avta i neste iterasjon.

**Predictor-corrector** Betrakt  $dy/dt = f(t, y)$  1. Regn ut stigninga ved  $t_i$ , i.e.  $k_1 = f(t_i, y_i)$ . 2. Anslå løsning:  $y_{i+1} \approx y(t_i) + hk_1$  (Eulers metode). 3. Bruk anslaget til å regne ut stigninga ved  $t_{i+1}$ ,  $k_2 = f(t_{i+1}, y_{i+1})$ . 4.

Korriger anslaget for løsning

$$y_{i+1} \approx y(t_i) + (k_1 + k_2)h/2.$$

**Løse pendelsystem**

- Choose the initial position and speed, with the most common choice  $v(t = 0) = 0$  and some fixed value for the position.
- Choose the method you wish to employ in solving the problem.
- Subdivide the time interval  $[t_i, t_f]$  into a grid with step size  $h = (t_f - t_i)/N$ , where  $N$  is the number of mesh points.

- Calculate now the total energy given by  $E_0 = \frac{1}{2}kx(t = 0)^2 = \frac{1}{2}k$ .
- The Runge-Kutta method is used to obtain  $x_{i+1}$  and  $v_{i+1}$  starting from the previous values  $x_i$  and  $v_i$ .
- When we have computed  $x(v)_{i+1}$  we upgrade  $t_{i+1} = t_i + h$ .
- This iterative process continues till we reach the maximum time  $t_f$ .
- The results are checked against the exact solution. Furthermore, one has to check the stability of the numerical solution against the chosen number of mesh points  $N$ .

## HVORDAN SJEKKE FEIL NÅR MAN LØSER ODE

Man kan sjekke forskjell i energi til systemet mellom numerisk og analytisk løsning, den vil øke ettersom tida går, spesielt ved dårlig oppløsning.

## 1.3 PDE

**DIFFUSJONSlikninga 1D**  $\nabla^2 u(x, t) = \frac{\partial u(x, t)}{\partial t}$  kan løses på flere måter. Først **eksplicit** metode, vi bruker forward Euler og den ender opp som en matrisemultiplikasjon. Vi diskretiserer og vår

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}.$$

som gir oss  $u_{i,j+1} = \alpha u_{i-1,j} + (1 - 2\alpha)u_{i,j} + \alpha u_{i+1,j}$ . Som essensielt er  $V_{j+1} = \mathbf{A}V_j$ . Utdrag av kode:

```
u(0) = unew(0) = u(n) = unew(n) = 0.0;
for (int i = 1; i < n; i++) {
    x = i*step;
    // initial condition
    u(i) = func(x);
    // initialise the new vector
    unew(i) = 0;
}
// Time integration
for (int t = 1; t <= tsteps; t++) {
    for (int i = 1; i < n; i++) {
        // Discretized diff eq
        unew(i) = alpha * u(i-1)
        + (1 - 2*alpha)*u(i) + alpha*u(i+1);
    }
}
```

Stabilitetsbetingelse:  $\Delta t/\Delta x^2 \leq 1/2$  finner vi fra spektralradien  $\rho(\mathbf{A}) = \max \left\{ |\lambda| : \det(\mathbf{A} - \lambda \hat{I}) = 0 \right\}$ .

For den **implisitte** metoden bruker vi backward Euler,

$$u_t \approx \frac{u(x_i, t_j) - u(x_i, t_j - \Delta t)}{\Delta t},$$

og ender opp med

$$u_{i,j-1} = -\alpha u_{i-1,j} + (1 - 2\alpha)u_{i,j} - \alpha u_{i+1,j}, \text{ i.e.}$$

$\mathbf{A}V_j = V_{j-1}$ . Denne løses med den tridiagonale løseren vår. Stabil for alle tids- og posisjonssteg. Til slutt har vi **Crank-Nicolson**

$$\begin{aligned} & \frac{\theta}{\Delta x^2} (u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) + \\ & \frac{1-\theta}{\Delta x^2} (u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}) = \\ & \frac{1}{\Delta t} (u_{i,j} - u_{i,j-1}), \end{aligned}$$

velger vi  $\theta = 1/2$  får vi CN-metoden. Vi ser at vi kan skrive  $(2\hat{I} + \alpha\hat{B})V_j = (2\hat{I} - \alpha\hat{B})V_{j-1}$ . Altså først gjøre den eksplisitte metoden, deretter bruke resultatet derfra til å løse en tridiagonal likning. NB: Husk at vi må ha  $\alpha = (\Delta t + \Delta t/2)/\Delta x^2$ .

**TRUNKERINGSFEIL:** verdt å merke seg at det halve tidssteget man bruker i CN gjør at trunkeringsfeilen i tid er forskjellig fra den implisitte metoden. For å finne de bruker vi

$$u(x + \Delta x, t) = u(x, t) + \frac{\partial u(x, t)}{\partial x} \Delta x + \frac{\partial^2 u(x, t)}{2\partial x^2} \Delta x^2 + \mathcal{O}(\Delta x^3), \quad (2)$$

$$u(x - \Delta x, t) = u(x, t) - \frac{\partial u(x, t)}{\partial x} \Delta x + \frac{\partial^2 u(x, t)}{2\partial x^2} \Delta x^2 + \mathcal{O}(\Delta x^3), \quad (3)$$

$$u(x, t + \Delta t) = u(x, t) + \frac{\partial u(x, t)}{\partial t} \Delta t + \mathcal{O}(\Delta t^2), \quad (4)$$

$$u(x, t - \Delta t) = u(x, t) - \frac{\partial u(x, t)}{\partial t} \Delta t + \mathcal{O}(\Delta t^2), \quad (5)$$

og setter dette inn i de forskjellige metodene. Ta tid og posisjon hver for seg.

## 1.4 Monte Carlo-metoder

**Tilfeldige tall** genereres av en funksjon som baserer seg på modulodivisjon, i.e. man deler et tall på et annet og svaret er resten. Kalles Linear congruential relations og ser ut som dette

$N_i = (aN_{i-1} + c) \text{MOD} M$ , men tallet som returneres er  $x_i = N_i/M$  for å sikre at det er mellom 0 og 1.  $M$  er perioden til funksjonen og bør være så stort som mulig,  $N_0$  er såkornet.  $a$  og  $c$  velges på en "lur måte". **Varians**  $\sigma_X^2 = \langle x^2 \rangle - \langle x \rangle^2$ . Standardavviket går som  $\sigma \sim 1/\sqrt{N}$ . Feil for tradisjonelle metoder hvor man int. over en  $d$ -dim. hyperkube med sider  $L$  (inneh.

$N = (L/h)^d \text{int.punkter}$ ) går som  $N^{-k/d}$ . MC er uavh. av dimensjonen til int. **SUPERBRA**. Vi liker å se på variansen som funksjon av ant. datapunkter ved numerisk integrasjon:  $\sigma_N^2 = \sigma_X^2/N$ .

**Numerisk integrasjon** Endrer grenser ved å bruke  $y = a + (b - a)x$ . Vi velger oss en PDF  $p(x)$  som vi putter inn i integralet.

$$I = \int_a^b F(x)dx = \int_a^b p(x) \frac{F(x)}{p(x)} dx = \int_{\tilde{a}}^{\tilde{b}} \frac{F(x(y))}{p(x(y))} dy.$$

Hvor vi har  $dy/dx = p(x)$  og integralet kan så skrives

$$I \approx \frac{1}{N} \sum_{i=1}^N \frac{F(x(y_i))}{p(x(y_i))},$$

hvor  $y_i \in [0, 1]$

**Importance sampling** løs  $p(y)dy = dx$  for  $x$ ,  $x(y) = \int_0^y p(y')dy'$ , (husk å løse for  $y(x)$ ) da kan man trekke uniforme tall  $x$  og distribuere de etter ønsket PDF, t.d. eksponentialdist.:  $p(y) = \exp(-y)$ , eller for den uniforme distribusjonen (som vi jo trekker tall fra):  $y(x) = a + (b - a)x$ , hvor  $a$  og  $b$  er de opprinnelige integrasjonsgrensene.

**Brute force**-integrering: husk å gange med volumelementet som kommer fra Jacobideterminanten  $\prod_{i=1}^d (b_i - a_i)$ , hvor  $a_i$  og  $b_i$  er integrasjonsgrensene for de forskjellige dimensjonene (i tilfelle flerdim. integral). **Akseptering/avvisning** kan brukes i stedet for importance sampling. Man trekker da et tall, sjekker om det er inne i intervallet vi er interessert i og bruker det om det er innafor.

## 1.5 Numerisk integrasjon

**Newton-Cotes** inneh. trapes-, rektangel- og Simpsons metode. Kalles også "equal step-method." Filosofi: diskretisere int.intervall med  $N$  punkter for en polynomisk integrand med dim. maks  $N - 1$ . Man tilnærmer integranden med et polynom.

**Gaussisk kvadratur** Grunnidé for alle integrasjonsmetoder:

$$I = \int_a^b f(x)dx \approx \sum_{i=1}^N \omega_i f(x_i),$$

GK går ut på å velge en ortogonal basis av polynomer og et sett integrasjonspunkter som vektes forskjellig. Kan skrive integranden  $f(x)$  som produkt av vektfunksjonen og en glatt funksjon,  $W(x)g(x)$ . Integrasjonspkt. er nullpkt. til de valgte ortogonale punktene av grad  $N$ . Vektene finner vi fra en invers

matrise. Vi rep. integranden med et polynom av grad  $2N - 1$  siden vi har  $2N$  likn.,  $N$  for int.pkt. og  $N$  for vektene.

**Trapesmetoden** 1. Velg ant. int.pkt. og fikser steglengden, 2. Regn ut  $f(a)$  og  $f(b)$ , og gang disse mde  $h/2$ . 3. Loop over  $n = 1 \rightarrow n - 1$ , summer opp  $f(a + h) + f(a + 2h) + \dots + f(b - h)$ . 4. Gang hele summen med  $h$  og legg til det du regnet ut i 2.

### Rektangelmetoden

$I = \int_a^b f(x)dx \approx h \sum_{i=1}^N f(x_{i-1/2})$ , m.a.o. man diskretiserer funksjonen i  $N$  rektangler, evaluerer funksjonen i midtpunktet i hvert rektangel og ganger med steglengden  $h$ .

### Simpsons metode

$$\int_{-h}^{+h} f(x)dx = \frac{h}{3} (f_h + 4f_0 + f_{-h}) + O(h^5),$$

**Bytte av intervall**  $t = (b - a)x/2 + (b + a)/2$ , husk å regne ut  $dt$ .

**Verlet-metoden** finner den ved å Taylorutvikle et kobla likningssett.

$$\begin{aligned} x(t + h) &= x(t) + hx^1(t) + (h^2/2)x^2(t) + O(h^3) \\ x_{i+1} &= 2x_i - x_{i-1} + h^2 x_i^{(2)} + O(h^4) \end{aligned}$$

Trunkeringsfeil i fjerde potens siden alle odde ledd kansellerer. Finner hastigheten ved

$$x^1(t) = \frac{x_{i+1} - x_{i-1}}{2h} + O(h^2)$$

Dette utgjør Verlet. Skriver man om Taylorutviklinga får man LEapfrog, bl.a. ved å bruke halve tidssteg.

$$\begin{aligned} x(t + h) &= x(t) + h(x^1(t) + (h/2)x^2(t)) + O(h^3) \\ x(t + h/2) &= (x^1(t) + (h/2)x^2(t)) + O(h^2) \\ x(t + h) &= x(t) + h + x^1(t + h/2) + O(h^3) \\ \Rightarrow x^1(t + h) &= x^1(t + h/2) + (h/2)x^2(t + h) + O(h^2) \end{aligned}$$

## 1.6 Statistisk fysikk

### Metropolisalgoritmen for Isingmodellen

- Generer starttilstand ved å plassere deg ved et tilfeldig spinn
- Generer prøvetilstand, *i.e.* snu ett spinn og beregn energidiff. (fem mulige verdier for 2D)
- Hvis prøvetilstand har negativ energidiff., så godta den nye tilstanden (energien senkes)
- Hvis ikke godtatt: generer tilfeldig tall  $r$ , sammenlikn med  $w = \exp(-\beta \Delta E)$  og godta hvis  $r \leq w$

- Oppdater forventningsverdier *etc.*
- Gjenta til likevekt er nådd, en MC-iterasjon er gjort ved en summasjon over alle spinnene.

For å finne når likevekt er nådd så kan man regne ut korrelasjonsfunksjonen for f.eks. magnetiseringa. Hvis det er likevekt så vil kun være fluktuasjoner som gjør at magnetiseringa endrer seg og det er ingen korrelasjon.

**Susceptibilitet m.m.**  $\chi = (\langle M^2 \rangle - \langle M \rangle^2)/(k_B T)$ , varmekap.  $C_V = (\langle E^2 \rangle - \langle E \rangle^2)/(k_B T^2)$ . Kan være lurt å dele disse kvantitetene på antall spinn så man kan sammenligne verdiene for forskjellige gitterstørrelser.

$$\sigma_E^2 = \langle E^2 \rangle - \langle E \rangle^2 = \frac{1}{Z} \sum_{i=1}^M E_i^2 e^{-\beta E_i} - \left( \frac{1}{Z} \sum_{i=1}^M E_i e^{-\beta E_i} \right)^2$$

$$\langle E \rangle = -\partial \ln Z / \partial \beta. \quad C_V = (\partial^2 \ln Z / \partial \beta^2) / (k_B T^2).$$

### Partisjonsfunksjon for 2D-Isingmodell

$$\begin{aligned} Z &= 2 \exp(8\beta J) + 2 \exp(-8\beta J) + 12 \exp(-\beta \cdot 0) = \\ &= 4 \cosh(8\beta J) + 12. \end{aligned}$$

**Markovkjeder** har tre viktige egenskaper i dens overgangsmatrise: avhenger kun av avstanden mellom to punkter  $i - j$  i rommet (homogenitet), isotropisk siden den ikke endres når man går fra  $(i, j)$  til  $(-i, -j)$ , homogen i tid siden den kun avhenger av start- og sluttiden.

**Detaljert balanse** er et krav man innfører for å unngå sykliske løsninger, *i.e.* at den gjentar seg selv, det er  $W(j \rightarrow i)w_j = W(i \rightarrow j)w_j$ . Vi kan skrive om dette ved å bruke overgangssannsynligheten  $T$  og aksepteringssanns.  $A$ ,  $(T_{j \rightarrow i} A_{j \rightarrow i}) / (T_{i \rightarrow j} A_{i \rightarrow j}) = w_i / w_j$ , Bruker vi så Boltzmanndistribusjonen,  $w_i = \exp(-\beta E_i) / Z$ , så får vi  $w_i / w_j = \exp(\beta(E_j - E_i))$ . Systemet vårt når en Boltzmanndistribuert likevekt. Hurra.

**Spektralradie** Først eksplisitt forover-Euler, da har vi  $\hat{A} = \mathbf{1} + \alpha \hat{B}$ . Egenverdiene til identitetsmatrisa er 1, mens vi for  $\hat{B}$  bør faktisk regne det ut så vi er helt sikre (selv om vi vet at en *reell* tridiagonal matrise er positiv definit). Vi kan skrive

$$b_{i,j} = 2\delta_{i,j} - \delta_{i+1,j} - \delta_{i-1,j}.$$

Eigenverdilikninga er gitt som

$$\hat{B}\hat{v} = \lambda\hat{v}$$

$$\begin{aligned}
\Rightarrow (\hat{B}\hat{v})_i &= \lambda_i \hat{v}_i \\
&= \sum_{j=1}^n (2\delta_{i,j} - \delta_{i+1,j} - \delta_{i-1,j}) v_j \\
&= 2v_i - v_{i+1} - v_{i-1} = \lambda_i v_i.
\end{aligned}$$

Vi kan så velge en basis,  $\sin(\beta\theta)$ , å uttrykke egenvektorene i slik at vi får

$$2\sin(i\theta) - \sin(i+1\theta) - \sin(i-1\theta) = \lambda_i \sin(i\theta).$$

Bruker vi så identiteten

$\sin(x+y) + \sin(x-y) = 2\cos(x)\sin(y)$  så kan vi forenkle uttrykket over til,

$$\begin{aligned}
2(1 - \cos(i\theta))\sin(i\theta) &= \lambda_i \sin(i\theta) \\
\lambda_i &= 2(1 - \cos(i\theta)).
\end{aligned}$$

Eigenverdiene for  $\hat{A}$  blir da  $\Gamma = 1 - 2\alpha(1 - \cos(i\theta))$

$$\Rightarrow -1 < 1 - 2\alpha(1 - \cos(i\theta)) < 1 \Rightarrow \alpha < \frac{1}{2},$$

hvilket impliserer at vi har kravet,

$$\frac{\Delta t}{\Delta x^2} < \frac{1}{2},$$

som betyr at vi ikke kan velge tids- og lengdesteg som vi vil og fremdeles få et konvergerende resultat.

**HUSK Å SKRIVE ALT DU GJØR I OPPGAVEN!**

| Metode         | Trunkering   | Stabilitetskrav                          |
|----------------|--|--|
| Eksplisitt     | $\mathcal{O}(\Delta x^2)$ og $\mathcal{O}(\Delta t)$   | $\Delta t \leq \Delta x^2/2$             |
| Implisitt      | $\mathcal{O}(\Delta x^2)$ og $\mathcal{O}(\Delta t)$   | $\forall \quad \Delta t$ og $\Delta x^2$ |
| Crank-Nicolson | $\mathcal{O}(\Delta x^2)$ og $\mathcal{O}(\Delta t^2)$ | $\forall \quad \Delta t$ og $\Delta x^2$ |

Tabell 1: Vi ser her trunkeringsfeil og stabilitetskrav for de tre forskjellige metodene. Vi noterer oss at Crank-Nicolson ser ut til å være den beste metoden.

| Vektfunk.                  | Intervall                 | Polynom   |
|----------------------------|---------------------------|-----------|
| $W(x) = 1$                 | $x \in [-1, 1]$           | Legendre  |
| $W(x) = \exp(-x^2)$        | $x \in (-\infty, \infty)$ | Hermite   |
| $W(x) = x^\alpha \exp(-x)$ | $x \in [0, \infty)$       | Laguerre  |
| $W(x) = 1/\sqrt{1-x^2}$    | $x \in [-1, 1]$           | Chebyshev |