Formidabelark FYS4150

Innhold

1.1 Eksamen 2014

ALGORITMER ER VIKTIGE **Oppgave 1** Egenverdier, "similaritytransformations", diskretisering av diff.likn., Jacobis metode/algoritme, Householders algoritme.

Oppgave 2PDE+linalg, diffusjonslikn., eksplisitt/implisitt, trunkeringsfeil, tridiagonal løser, FLOPS

Oppgave 3 ODE, omskrivning til et sett koblede likn., Eulers algoritme(???), feilestimat, Runge-Kutta, geometrisk tolkning av RK, enhetstesting

1.2 Eksamen 2013

Oppgave 1 ODE, andreordens til to førsteordens koblede likn., Eulers algoritme, Runge-Kutta med feilestimat, geometrisk tolkning av RK.

Oppgave 2 PDE, diffusjon, diskretiser, eksplisitt/implisitt, trunkeringsfeil, tridiagonal løser, FLOPS

Oppgave 3 Numerisk integrasjon, Gaussisk kvadratur, Legendre-polynomer, Laguerre-pol., MC-metoder, brute force og importance sampling.

1.3 Eksamen 2012

Oppgave 1 Linalg, Gaussisk eliminasjon, LU dekomp., diskretisering, FLOPS, enhetstesting

Oppgave 2 Metropolisalgoritmen, antakelser for den, enhetstesting,, markovkjeder,

1.4 Eksamen 2011

Oppgave 1 ODE, omskriving til to koblede, Eulers algoritme, Runge-Kutta, feilestimat

Oppgave 2 Numerisk integrasjon, trapesregelen, newton-cotes, gaussisk kvadratur, legendre-polynom, laguerre-polynom, MC-integrasjon, importance sampling.

Oppgave 3 Linalg., egenverdier, similarity transforms, diskretisering, jacobis metode,

1.5 Lineær algebra og sånn

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

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

Gir likninga $\mathbf{Au} = \mathbf{f}(\mathbf{u})$. Som vi kan skrive som $a_iu_{i-1} + b_iu_i + c_iu_{i+1} = f_i$,. 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];
```

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 $\mathbf{y} = \mathbf{Ux} = \mathbf{L^{-1}w}$.
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 definitt eller dominant på diagonalen kan man vise at denne metoden alltid vil konvergere.
Egenverdier og 'similarity'transformering Man kan finne egenverdiene av A ved å bruke $\mathbf{B} = \mathbf{S}^T\mathbf{AS}$, $\mathbf{S}^T\mathbf{S} = \mathbf{S}^{-1}\mathbf{S} = \mathbf{I}$. Gjør man det mange nok ganger ender man opp med $\mathbf{S}_N^T \dots \mathbf{S}_1^T\mathbf{AS}_1 \dots \mathbf{S}_N = \mathbf{D}$, hvor D har egenverdiene til A på diagonalen. Egenverdiene endres ikke av denne transformasjonen: $\mathbf{Ax} = \lambda\mathbf{x} \Rightarrow (\mathbf{S}^T\mathbf{AS})(\mathbf{S}^T\mathbf{x}) = \lambda\mathbf{S}^T\mathbf{x}$, men vi ser at egenvektorene endres! → må rotere egenvektormatrisa motsatt vei for å få de "ekte" egenvektorene (prosjekt 1?).
(Jacobis metode for egenverdier) Vi setter

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

Denne matrisa roterer vår matrise A slik at ett av de ikkediagonale elementene i A blir satt til null. Vi velger alltid det største elemtentet til å være lik null slik at metoden konvergerer raskest mulig. Må ha en while-test og if-tester for å sjekke at metoden konvergerer og stanse den når vi er nærme nok en diagonal matrise. ULEMPE: uvisst hvor mange iterasjoner man trenger.
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 ortogonal matrise. Man lar den virke på hver side av A og vi ender opp med

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

Vi skriver rotasjonsmatrisa som

$$\mathbf{S_1} = \begin{pmatrix} 1 & \mathbf{0^T} \\ \mathbf{0} & \mathbf{P} \end{pmatrix},$$

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

$$\mathbf{S}_1^T\mathbf{AS}_1 = \begin{pmatrix} a_{11} & (\mathbf{Pv})^T \\ \mathbf{Pv} & \mathbf{A}' \end{pmatrix},$$

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

Som vi kan skrive som $\mathbf{v} - k\mathbf{e} = 2\mathbf{u}(\mathbf{u}^T\mathbf{v})$, opphøyer vi dette i andre får vi $2(\mathbf{u}^T\mathbf{v})^2 = (v^2 \pm a_{21}v)$, 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

1.6 ODE

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

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

1.7 PDE

DIFFUSJONSlikninga 1D $\nabla^2 u(x,t) = \frac{\partial u(x,t)}{\partial t}$ kan løses på flere måter. Først **ekpsplisitt** 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);
      //  intitialise 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
      }
```

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

$$\frac{\theta}{\Delta x^2}\left(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}\right) + \frac{1-\theta}{\Delta x^2}\left(u_{i+1,j-1} - 2u_{i,j-1} + u_{i-1,j-1}\right) = \frac{1}{\Delta t}\left(u_{i,j} - u_{i,j-1}\right), \tag{2}$$

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

1.8 Monte Carlo-metoder

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. $N = (L/h)^d$int.punkter) går som $N^{-k/d}$. MC er uavh. av dimensjonen til int. SUPERBRA. **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 $y_i \in [0,1]$

1.9 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 dermed et sett integrasjonspunkter som vektes forskjellig. Kan skrive integranden $f(x)$ som produkt av vektfunksjonen og en glatt funksjon, $W(x)g(x)$.
Legendre-polynom REferer til Rottmann
Trapesmetoden
HUSK Å SKRIVE ALT DU GJØR I OPPGAVEN!