STEFFEN HAUG

Numeriske Metodar

Prosjekt 3

I. Kvadraturmetodar

i. Adaptiv Simpson-kvadratur

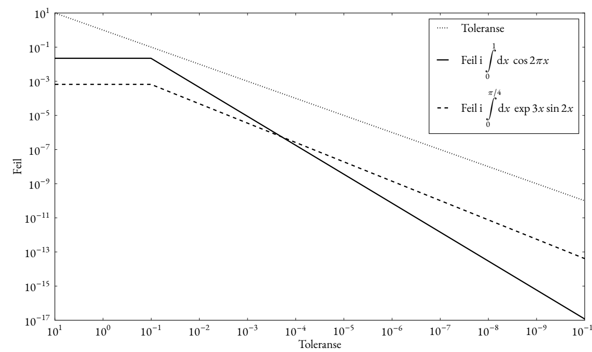
Listing 1: *Adaptiv Simpson-kvadratur*

```
def S(f, a, b):
    return (f(a) + 4 * f((a + b) / 2) + f(b))  \
          * (b - a) / 6

def asm_quad(f, a, b, tol=1e-5):
    I0 = S(f, a, b)
    c = (a + b) / 2

    I = S(f, a, c) + S(f, c, b)
    err = abs(I - I0) / 15

    if err < tol:
        return I + (I - I0) / 15
    else:
        return asm_quad(f, a, c, tol=tol/2)  \
             + asm_quad(f, c, b, tol=tol/2)
```

ii. Romberg-kvadratur

Listing 2: *Romberg-kvadratur*

```
def romberg(f, a, b, MAX_ITER=100, tol=1e-5):
    R = np.zeros((2, MAX_ITER))
    Rp, Rn = 0, 1

    h = b - a
    R[Rp, 0] = 0.5 * h * (f(a) + f(b))

    for n in range(1, MAX_ITER):
        h = h * 0.5
        L = np.linspace(a + h, b - h, 1 << n - 1)
        R[Rn, 0] = R[Rp, 0]/2 + np.sum(f(L))*h

        for k in range(1, n + 1):
            E = (R[Rn, k - 1] - R[Rp, k - 1])  \
              / ((1 << 2 * k) - 1)
            R[Rn, k] = R[Rn, k - 1] + E

        Rp, Rn = Rn, Rp

        if abs(E) < tol: break

    return R[Rp, n]
```



Figur 1: *Feil i adaptiv Simpson-kvadratur. Ein ser at feilen er grensa ovanfrå av toleransen.*

II. Simulasjon av fri, stiv lekam

Vi ønsker å simulere ein *fri, stiv lekam*, med andre ord ein lekam som ikkje lar seg deformere, som roterer fritt i rommet frå ein gitt start-tilstand.

Fri rotasjon betyr at netto påført dreiemoment er null. At lekamen ikkje lar seg deformere medfører at treighetsmomentet er konstant. (massen flyttar seg ikkje relativt til rotasjons-aksen) Dette betyr at normen til dreieimpulsen, og rotasjonsenergien er bevart. [1] Ingenting hindrar dreieimpulsen i å endre retning.

i. Definisjon av problemet

Differensiallikninga [1, Namn på symbola er endra for å passe vår oppgåvetekst]

$$T\frac{\partial \omega}{\partial t} + \omega \times T\omega = M \tag{1}$$

skildrar rotasjonen til eit stivt legeme. Her er M påført dreiemoment, T treighetsmoment, og ω vinkelfart. Per antagelse er $M = 0$. Innfør substitusjonen

$$\mathbf{m} = \mathrm{T}\omega$$
$$\implies \frac{\partial \mathbf{m}}{\partial t} = \mathrm{T}\frac{\mathrm{d}\omega}{\mathrm{d}t} \quad \text{og} \quad \omega = \mathrm{T}^{-1}\mathbf{m}$$

i (1), trekk frå $\mathrm{d}\mathbf{m}/\mathrm{d}t$ på begge sider, og snu kryssproduktet for å få like forteikn. Samtidig innfører vi notasjonen $\dot{f} = \mathrm{d}f/\mathrm{d}t$.

$$\dot{\mathbf{m}} = \mathbf{m} \times \mathrm{T}^{-1}\mathbf{m} \tag{1*}$$

Bevaringslovene nevnt til å byrje med gir oss:

$$\gamma = \|\mathbf{m}\|^2 = \mathbf{m}(t) \cdot \mathbf{m}(t) \tag{2}$$
$$E = \frac{1}{2}\mathbf{m}(t) \cdot \mathrm{T}^{-1}\mathbf{m}(t) \tag{3}$$

Der (2) er ei sfære med radius $\|\mathbf{m}\|$, og (3) er ei ellipse. Sidan både γ og E er konstantar er løysingane $\mathbf{m}$ begrensa til å sitje på skjæringa mellom flatene.

Anta $\mathrm{T} = \mathrm{diag}(\mathrm{I}_1, \mathrm{I}_2, \mathrm{I}_3)$. Ettersom T er diagonal er $\mathrm{T}^{-1} = \mathrm{diag}(1/\mathrm{I}_1, 1/\mathrm{I}_2, 1/\mathrm{I}_3)$. Skriv $\mathbf{m} = (x \ \ y \ \ z)$, og skriv ut (1*) på komponentform:

$$\underbrace{\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}}_{\dot{\mathbf{m}}} = \underbrace{\begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}}_{\mathbf{m}} \times \left[\underbrace{\begin{pmatrix} 1/\mathrm{I}_1 & 0 & 0 \\ 0 & 1/\mathrm{I}_2 & 0 \\ 0 & 0 & 1/\mathrm{I}_3 \end{pmatrix}}_{\mathrm{T}^{-1}} \underbrace{\begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}}_{\mathbf{m}} \right]$$

Rekn ut kryssproduktet:

$$\underbrace{\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{z}(t) \end{pmatrix}}_{\dot{\mathbf{m}}} = \underbrace{\begin{pmatrix} Ay(t)z(t) \\ Bx(t)z(t) \\ Cx(t)y(t) \end{pmatrix}}_{f(t,\mathbf{m})} \tag{4}$$

der A, B og C er konstantar

$$A = 1/I_3 - 1/I_2$$
$$B = 1/I_1 - 1/I_3$$
$$C = 1/I_2 - 1/I_1$$

ii. Implisitt Runge-Kutta midtpunkt-metode

Vi ønsker å løyse differensiallikningar av sorten

$$\dot{y} = f(t, y)$$

numerisk, ved hjelp av Runge-Kutta metoden

$$y_{n+1} = y_n + hf\left(t_n + \frac{h}{2}, \frac{1}{2}(y_n + y_{n+1})\right), \tag{5}$$

kalt *implisitt midtpunktmetode*, fordi y_{n+1} avheng av eit estimat for $y_{n+1/2}$ (derav implisitt) for å estimere tangenten i midpunktet mellom t_n og t_{n+1}. Substituer

$$u = \frac{1}{2}(y_n + y_{n+1}) \implies y_{n+1} = 2u - y_n$$

for å forenkle notasjonen litt. Dette gir likningssystemet

$$2u - y_n = y_n + hf(t + h/2, u)$$
$$\implies u = y_n + \frac{h}{2}f(t + h/2, u)$$
$$\implies y_n + \frac{h}{2}f(t + h/2, u) - u = 0$$

som må løysast med omsyn til u for kvart tidssteg. Gitt ein verdi for u reknar vi ut y_{n+1}:

$$y_{n+1} = y_n + hf(t + h/2, u) \tag{6}$$

Anvending på problemet

Eitt steg gjenstår før vi kan implementere løysaren: Vi er nøtt å velgje ein måte å løyse det implisitte steget. Frå (1*) får vi, med $\mathbf{u} = (x\ y\ z)$, likninga

$$\mathbf{F}(\mathbf{u}) = \begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} + \frac{h}{2}\begin{pmatrix} Ayz \\ Bxz \\ Cxy \end{pmatrix} - \begin{pmatrix} x \\ y \\ z \end{pmatrix} = 0. \tag{7}$$

Vi innlemmer $h/2$ i konstantane A, B og C. Til slutt sit vi att med systemet

$$\begin{cases} x_n + \hat{A}yz - x = 0 \\ y_n + \hat{B}xz - y = 0 \\ z_n + \hat{C}xy - z = 0 \end{cases}$$

der x_n, y_n, z_n, samt $\hat{A}$, $\hat{B}$ og $\hat{C}$ er kjende konstantar. Jacobian-matrisa lar seg enkelt rekne ut ved hjelp av symbolske verktøy, til dømes `sympy` i Python. Vi har

$$\mathcal{J}_{\mathbf{F}} = \begin{pmatrix} -1 & \hat{A}z & \hat{A}y \\ \hat{B}z & -1 & \hat{B}x \\ \hat{C}y & \hat{C}x & -1 \end{pmatrix}.$$

Altso kan vi finne $\mathbf{u} = (x\ \ y\ \ z)$ med å bruke Newtons metode:

$$\mathbf{u} \longleftarrow \mathbf{u} - \mathcal{J}_{\mathbf{F}}^{-1}\mathbf{F}(\mathbf{u})$$

som burde konvergere i løpet av svært få iterasjonar dersom vi brukar førre iterasjon som startpunkt, og relativt kort skrittlengde.

iii. Implementasjon av RK-metoden

Vi brukar eit symbolsk verktøy for å rekne ut (7) i forkant; koden for dette er ikkje spesielt vanskelig, men den er stygg, så eg har klipt den ut. Den kan finnast i notebooken `rk3d.pynb`.

Listing 3: *Implisitt Runge-Kutta midtpunktmetode.*

```python
def RK(f, y0, t):

# (sympy-junk removed from here)
# F = F(u), Ji = inv(Jac(F))

y = np.empty((len(t), 3))
y[0] = y0

for n in range(len(t) - 1):
    h = t[n + 1] - t[n]

    u = newton(
        lambda u:  F(u, y[n], t[n] + h/2, h),
        lambda u: Ji(u, y[n], t[n] + h/2, h),
        y[n]
    )

    m = np.array(f(t[n] + h/2, u), dtype=float)

    y[n + 1] = y[n] + h * m

return y.T
```

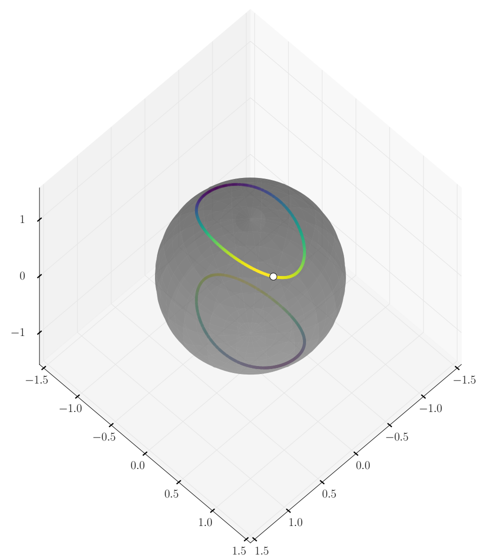
I metoden inngår Newtons metode; denne har vi implementert i ei tidlegare øving, men legg den ved for kompletthets skuld:

Listing 4: *Newtons metode frå tidlegare øving.*

```python
def newton(F, Ji, u0, tol=1e-10, maxiter=10):
    u = u0
    for _ in range(maxiter):
        u = u - Ji(u).dot(F(u))

        if linalg.norm(F(u)) < tol:
            break

    return u
```

Den eine haka ved implementasjonen er at differensiallikninga vi vil løyse, for det første, må vera tredimensjonell, og for det andre må vera definert berre ved hjelp av funksjonar som er kompatible med `sympy`.

Desse vala er gjort med vilje; det er enklare å ha eit eksplisitt uttrykk for Jacobian-matrisa ved kvart tidssteg, (som ein får ved å rekne den ut i forkant) enn å forsøke å estimere den numerisk. Når ein har bestemt seg for å bruke symbolske metodar i staden for tilnærmingar er det enklare å behandle uttrykk der dimensjonen er kjend. `sym-py` *har* funksjonalitet for å lage vektorar med symbol, så det er (i prinsippet) mogleg, men element i desse vektorane får obskure navn (som "`_dummy_123`") som gjer feilsøking til eit mareritt.



FIGUR 2: *Løysing av* (1*) *med initialverdiar* $T = \operatorname{diag}(1, 2, 5)$ *og* $m_0 = (2\ 5\ 7)$. *Tidsrommet er* $[t = 0,\ t = 250]$, *med* $N = 500$ *steg med lik steglengde. Motsette punkt på sfæra er farga likt,* m_0 *er farga kvitt.*

REFERANSAR

[1] J. R. Lien og G. Løvhøiden, *Generell fysikk for universiteter og høgskoler* (2015).