Numerisk Integrasjon

Anne Kværnø

March 12, 2018

1 Problemstilling

Vi skal altså finne en numerisk tilnærmelse til integralet

$$I(a,b) = \int_a^b f(x)dx$$

for en gitt funksjon $f(x)$.

Teknikken vi skal diskutere kalles *numeriske kvadratur*, disse er på formen

$$Q(a,b) = \sum_{i=0}^{n} w_i f(x_i)$$

der punktene x_i kalles noder og w_i vekter. Og vi ønsker altså at

$$I(a,b) \approx Q(a,b).$$

Trapesregelen, midtpunkt-regelen og Simpsons regel er alle kjente eksempler på slike formler.

I dette notatet skal vi se mer generelt på hvordan slike kvadraturformler kan konstrueres, hvordan man kan finne et estimat for feilen i kvadratueret, og hvordan dette kan settes sammen til en algoritme som gir et tilnærmelse til løsningen med en nøyaktighet som er mindre eller omtrent lik en oppgitt toleranse.

2 Kvadraturformler basert på polynominterpolasjon.

Velg et sett med noder x_i, $i = 0, \ldots, n$ på intervallet $[a,b]$. Interpolasjonspolynomet til $f(x)$ gjennom disse nodene er gitt ved

$$p_n(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x)$$

der *kardinalfunksjonene* $\ell_i(x)$ er gitt ved

$$\ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j},$$

se notatet om polynominterpolasjon. Da kan vi bruke integralet av $p_n(x)$ som en tilnærmelse til integralet av $f(x)$, dvs

$$I(a,b) = \int_a^b f(x)dx \approx \int_a^b p_n(x)dx = \sum_{i=0}^{n} f(x_i) \int_a^b \ell_i(x)dx = \sum_{i=0}^{n} w_i f(x_i) = Q(a,b)$$

Definisjon: Presisjonsgrad Et numerisk kvadratur har presisjonsgrad d dersom $Q(a,b) = I(a,b)$ for alle polynomer av grad d eller mindre.

Eksempler

1. Trapesmetoden er basert på nodene (a,b), og blir

   $$T(a,b) = \frac{b-a}{2}(f(a) + f(b)).$$

   Presisjonsgraden finner vi ved:

   $$f = 1, \qquad I(a,b) = b - a, \qquad T(a,b) = b - a, \tag{1}$$

   $$f = x, \qquad I(a,b) = \frac{b^2 - a^2}{2}, \qquad T(a,b) = \frac{b^2 - a^2}{2}, \tag{2}$$

   $$f = x^2, \qquad I(a,b) = \frac{b^3 - a^3}{3}, \qquad T(a,b) = \frac{(b-a)(b^2 + a^2)}{2}, \tag{3}$$

   så trapesmetoden har presisjonsgrad 1.

2. På intervallet $[-1,1]$, velg noder $\pm\sqrt{(}3)/3$. Dette gir kvadraturformelen

   $$Q(-1,1) = f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right).$$

   Denne har har også presisjonsgrad 3 (vis det selv).

Jo høyere presisjonsgrad, jo bedre kvadratur. Alle kvadraturformler basert på polynominterpolasjon i $n+1$ noder vil automatisk ha presisjonsgrad minst n, men bedre resultat er som vi har sett mulig.

I praksis vil vi utvikle slike formler på et standardinterval $[-1,1]$ for deretter å flytte disse over til intervallet $[a,b]$. Eller vi bruker *sammensatte formler*: Partisjoner intervallet $[a,b]$ i N passende delintervaller

$$a = X_0 < X_1 < \cdots < X_N = b$$

og anvend en kvadraturformel på hvert delintervall:

$$\int_a^b f(x) = \sum_{k=0}^{N-1} \int_{X_k}^{X_{k+1}} f(x)dx \approx \sum_{k=0}^{N-1} Q(X_k, X_{k+1}).$$

Oppsummert: 1. Velg noder på standardintervallet $[-1,1]$. 2. Finn kvadraturformelen $Q(-1,1)$ ved å integrere interpolasjonspolynomet. 3. Flytt dette til et tilfeldig intervall $[a,b] \;\Rightarrow\; Q(a,b)$. 4. Forutsatt at feilen $E(a,b) = I(a,b) - Q(a,b) \approx C(b-a)^p$ for en gitt p er det også mulig å finne et feilestimat. Det kan brukes til å konstruere en algoritme som automatisk gir en passende partisjonering av intervallet.

La oss gå gjennom hele denne prosessen med et velkjent eksempel:

2.1 Simpsons metode

Standardintervall [-1,1]: Vi starter med å finne en tilnærmelse til $\int_{-1}^{1} f(t)dt$. Velg nodene $t_0 = -1$, $t_1 = 0$ og $t_2 = 1$. Dette gir oss kardinalfunksjonene

$$\ell_0 = \frac{1}{2}(t^2 - t), \qquad \ell_1(t) = 1 - t^2, \qquad \ell_2(t) = \frac{1}{2}(t^2 + t).$$

som igjen gir vektene

$$w_0 = \int_{-1}^{1} \ell_0(t)dt = \frac{1}{3}, \qquad w_1 = \int_{-1}^{1} \ell_1(t)dt = \frac{4}{3}, \qquad w_2 = \int_{-1}^{1} \ell_2(t)dt = \frac{1}{3}$$

slik at

$$\int_{-1}^{1} f(t)dt \approx \int_{-1}^{1} p_2(t)dt = \sum_{i=0}^{2} w_i f(t_i) = \frac{1}{3}\left[\, f(-1) + 4f(0) + f(1) \,\right].$$

Simpsons metode har presisjonsgrad 3 (sjekk det selv).

Eksempel:

$$\int_{-1}^{1} \cos\left(\frac{\pi t}{2}\right) dt = \frac{4}{\pi} \approx \frac{1}{3}\left[\cos(-\pi/2) + 4\cos(0) + \cos(\pi/2)\right] = \frac{4}{3}.$$

Over til intervallet [a,b]:

Resultatet fra standardintervallet flyttes til et intervall $[a, b]$ ved transformasjonen

$$x = \frac{b-a}{2}t + \frac{b+a}{2}.$$

Husk at da blir $dx = (b-a)/2\, dt$.

Det betyr at

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}t + \frac{b+a}{2}\right) dt \approx \frac{b-a}{6}\left[\, f(a) + 4f\left(\frac{b+a}{2}\right) + f(b)\,\right].$$

Så Simpsons formel over et interval $[a, b]$ er

$$S(a,b) = \frac{b-a}{6}\,(\, f(a) + 4f(c) + f(b)\,), \qquad c = \frac{b+a}{2}.$$

(Det er vanlig praksis å bruke S i stedet for Q i dette tilfellet.)

Sammensatt Simpsons metode

Del intervallet $[a, b]$ inn i $2m$ like store intervaller av lengde $ h = (b-a)/(2m)$. La $x_j = a + jh$, $i = 0, \cdots, 2m$, og anvend Simpsons formel på hvert delintervall $[x_{2j}, x_{2j+2})]$. Det resulterer i

$$\int_a^b f(x)dx = \sum_{j=0}^{m-1} \int_{x_{2j}}^{x_{2j+2}} f(x)dx \approx \sum_{j=0}^{m} S(x_{2j}, x_{2j+2}) \tag{4}$$

$$= \sum_{j=0}^{m-1} \frac{h}{3}\left[f(x_{2j}) + 4f(x_{2j+1}) + f(x_{2j+2})\right] \tag{5}$$

$$= \frac{h}{3}\left[f(x_0) + 4\sum_{j=0}^{m-1} f(x_{2j+1}) + 2\sum_{j=1}^{m-1} f(x_{2j}) + f(x_{2m})\right] = S_m(a,b). \tag{6}$$

Den sammensatte formelen, her kalt $S_m(a, b)$, er kjent som Simpsons regel fra Matematikk 1.

2.1.1 Implementasjon av sammensatt Simpsons metode

```
In [1]: %matplotlib inline
        from numpy import *
        from matplotlib.pyplot import *
        from math import factorial
        newparams = {'figure.figsize': (8.0, 4.0), 'axes.grid': True,
                     'lines.markersize': 8, 'lines.linewidth': 2,
                     'font.size': 14}
        rcParams.update(newparams)
```

```
In [2]: def simpson(f, a, b, m=10):
        # Finner en tilnærmelse til integralet av funksjonen f
        # over intervallet [a,b] ved bruk av Simpsons metode med m delintervaller.
            n = 2*m
            x_noder = linspace(a, b, n+1)     # Jevnt fordelte noder fra a til b
            h = (b-a)/n
            S1 = f(x_noder[0]) + f(x_noder[n])
            S2 = sum(f(x_noder[1:n:2]))         # S2 = f(x_1)+f(x_3)+...+f(x_m)
            S3 = sum(f(x_noder[2:n-1:2]))       # S3 = f(x_2)+f(x_4)+...+f(x_{m-1})
            S = h*(S1 + 4*S2 + 2*S3)/3
            return S
```

3 Testing

Test om koden er korrekt: Simpsons metode vil integrere 2.gradspolynomer eksakt. f.eks. $f(x) = 4x^3 + x^2 + 2x - 1$. Vis at $\int_{-1}^{2} f(x)dx = 18$. Sjekk om koden gir rett svar.

```
In [3]: # Definerer funksjonen
        def f(x):
            return 4*x**3+x**2+2*x-1

        I_eksakt = 18.0

        # Bruker Simpsons sammensatte metode for å tilnærme integralet
        S = simpson(f, -1, 2, m=1)
        feil = I_eksakt-S
        print('I = {:.8f},  S = {:.8f},  feil = {:.3e}'.format(I_eksakt, S, feil))
```

```
I = 18.00000000,  S = 18.00000000,  feil = 0.000e+00
```

Numeriske eksperimenter

1. Finn en tilnærmelse til $\int_0^{\pi/2} \cos(x)dx$. Bruk $m = 1, 2, 4, 8$. Finn eksperimentelt hvor stor m må være for at feilen skal bli mindre enn 10^{-6}.
2. Gjenta eksperimentet med integralet

$$\int_0^8 \frac{1}{1+16x^2}dx.$$

(Eksakt løsning er $\arctan(32)/4$.)

3.0.1 Feilanalyse

Bruk Taylorutviklinger rundt midtpunktet $c = (b+a)/2$. La $h = (b-a)/2$. Dette gir

$$\int_a^b f(x)dx = \int_{-h}^{h} f(c+s)ds = \int_{-h}^{h} f(c) + sf'(c) + \frac{1}{2}s^2f''(c) + \frac{1}{6}s^3f'''(c) + \frac{1}{24}s^4f^{(4)}(c) + \cdots \quad (7)$$

$$= 2hf(c) + \frac{h^3}{3}f''(c) + \frac{h^5}{60}f^{(4)}(c) + \cdots . \quad (8)$$

Hvert andre ledd forsvinner pga. symmetri.

Tilsvarende for kvadraturet $S(a,b)$:

$$S(a,b) = \frac{h}{3}\left(f(c-h) + 4f(c) + f(c+h)\right) \quad (9)$$

$$= \frac{h}{3}\Big(f(c) - hf'(c) + \frac{1}{2}h^2f''(c) - \frac{1}{6}h^3f'''(c) + \frac{1}{24}h^4f^{(4)}(c) + \cdots \quad (10)$$

$$+ 4f(c) \quad (11)$$

$$+ f(c) + hf'(c) + \frac{1}{2}h^2f''(c) + \frac{1}{6}h^3f'''(c) + \frac{1}{24}h^5f^{(4)}(c) + \cdots\Big) \quad (12)$$

$$= 2hf(c) + \frac{h^3}{3}f''(c) + \frac{h^5}{36}f^{(4)}(c) + \cdots \quad (13)$$

Vi har da følgende uttrykk for feilen:

$$E(a,b) = \int_a^b f(x)dx - S(a,b) = -\frac{h^5}{90}f^{(4)}(c) + \cdots = -\frac{(b-a)^5}{2^5 \cdot 90}f^{(4)}(c) + \cdots$$

Dette gir en rimelig ide om størrelsen på feilen, under forusetningen av at $b-a$ er relativt liten slik at høyere ordens ledd kan ignoreres.

Det er mulig, men ikke helt trivielt å vise følgende presise resultat:
Setning (feil for Simpsons metode)

La $f(x) \in C^4[a,b]$. Da fins en $\xi \in (a,b)$ slik at

$$E(a,b) = \int_a^b f(x)dx - \frac{b-a}{6}\left[\, f(a) + 4f\left(\frac{b+a}{2}\right) + f(b) \,\right] = -\frac{(b-a)^5}{2880} f^{(4)}(\xi).$$

Dette kan vi bruke for å finne et uttrykk for feilen i sammensatt Simpsons formel.

$$\int_a^b f(x)dx - S_m(a,b) = \sum_{j=0}^{m-1} \left(\int_{x_{2j}}^{x_{2j+2}} f(x)dx - \frac{h}{3}\left[f(x_{2j}) + 4f(x_{2j+1}) + f(x_{2j+2}) \right] \right) \tag{14}$$

$$= \sum_{j=0}^{m-1} -\frac{(2h)^5}{2880} f^{(4)}(\xi_j) \tag{15}$$

der $\xi_j \in (x_{2j}, x_{2j+2})$. Vi kan nå bruke Section **??** til å vise at det fins en $\xi \in (a,b)$ slik at

$$\sum_{j=0}^{m-1} f^{(4)}(\xi_j) = m f^{(4)}(\xi).$$

Ved å bruke $2mh = b - a$ får vi følgende uttrykk for feilen:

$$\int_a^b f(x)dx - S_m(a,b) = -\frac{(b-a)h^4}{180} f^{(4)}(\xi).$$

3.0.2 Feilestimat

Er det mulig å finne en tilnærmelse til feilen $I(a,b) - S(a,b)$, uten å kjenne $f^{(4)}(x)$?

Anta at intervallet (a,b) er så lite at $f^{(4)}(x)$ er tilnærmet konstant over intervallet. La $H = b - a$.

Da har vi følgende:

$$I(a,b) - S_1(a,b) \approx CH^5, \tag{16}$$

$$I(a,b) - S_2(a,b) \approx 2C\left(\frac{H}{2}\right)^5. \tag{17}$$

der $I(a,b) = \int_a^b f(x)dx$, og $S_m(a,b)$ er sammensatt Simpsons formel anvendt på m like store delintervaller av $[a,b]$, og $C = f^{(4)}(\xi)/2880$, der ξ er et eller annet punkt i (a,b). Hvilket spiller ingen rolle siden forutsetningen her er at $f^{(4)}(x)$ er nesten konstant.

Ta differensen mellom disse to:

$$S_2(a,b) - S_1(a,b) \approx \frac{15}{16}CH^5 \qquad \Rightarrow \qquad CH^5 \approx \frac{16}{15}(S_2(a,b) - S_1(a,b)).$$

Sett dette inn i uttrykket over:

$$E_1(a,b) = I(a,b) - S_1(a,b) \approx \frac{16}{15}(\, S_2(a,b) - S_1(a,b)\,) = \mathcal{E}_1(a,b), \tag{18}$$

$$E_2(a,b) = I(a,b) - S_2(a,b) \approx \frac{1}{15}(\, S_2(a,b) - S_1(a,b)\,) = \mathcal{E}_2(a,b). \tag{19}$$

Siden feilen i $S_2(a,b)$ er omtrent $1/16$ av feilen i $S_1(a,b)$, og begge uansett må regnes ut for å finne feilestimatene, så vil vi bruke $S_2(a,b)$ som tilnærmelse. Denne kan gjøres enda bedre ved å legge til feilestimatet.

Eksempel

Finn en tilnærmelse til integralet $\int_0^1 \cos(x)dx = \sin(1)$ ved hjelp av Simpsons formel over et og to delintervaller. Beregn feilestimatene $\mathcal{E}_m$, $m = 1,2$ og sammenlign den med virkelig feilen.

Vi får

$$S_1(0,1) = \frac{1}{6}\big[\cos(0.0) + 4\cos(0.5) + \cos(1.0)\big] = 0.8417720923 \tag{20}$$

$$S_2(0,1) = \frac{1}{12}\big[\cos(0.0) + 4\cos(0.25) + 2\cos(0.5) + 4\cos(0.75) + \cos(1.0)\big] = 0.8414893826 \tag{21}$$

Den eksakte feilen og feilestimatet blir

$$E_1(0,1) = \sin(1) - S_1(0,1) = -3.011 \cdot 10^{-4}, \quad \mathcal{E}_1(0,1) = \frac{16}{15}(S2 - S1) = -3.016 \cdot 10^{-4}, \tag{22}$$

$$E_2(0,1) = \sin(1) - S_2(0,1) = -1.840 \cdot 10^{-5}, \quad \mathcal{E}_2(0,1) = \frac{1}{16}(S2 - S1) = -1.885 \cdot 10^{-5}. \tag{23}$$

I dette tilfellet er det et meget godt samsvar mellom feilestimatet og målt feil. Vi får en enda bedre tilnærmelse ved å bruke

$$Q = S_2(0,1) + \mathcal{E}_2(0,1) = 0.8414705353607151$$

som har en feil $\sin(1) - Q = 4.4945 \cdot 10^{-7}$. Det gir et mye mer nøyaktig resultat uten ekstra arbeid.

3.0.3 Implementasjon av Simpsons metode med feilestimat

Funksjonen `simpson_basis` regner ut en tilnærmelse til integralet

$$\int_a^b f(x)dx$$

ved hjelp av Simpsons metode, inkludert et feilestimat.

```
In [4]: def simpson_basis(f, a, b):
            # Regner ut en tilnærmelse til integralet av f over [a,b].
            # Inn: Funksjonen f, og integrasjonsintervallet [a,b]
            # Ut: S_1(a,b), S_2(a,b) og feilestimatet for S_2(a,b).

            # Finner nodene
            c = 0.5*(a+b)
            d = 0.5*(a+c)
            e = 0.5*(c+b)

            # Regner ut S1=S_1(a,b), S2=S_2(a,b) og feilestimatet for S2
            H = b-a
            S1 = H*(f(a)+4*f(c)+f(b))/6
            S2 = 0.5*H*(f(a)+4*f(d)+2*f(c)+4*f(e)+f(b))/6
            feilestimat = (S2-S1)/15
            return S1, S2, feilestimat
```

Testing Test implementasjonen på eksempelet over:

$$\int_0^1 \cos(x)dx.$$

og kontroller at svarene blir de samme som tidligere.

```
In [5]: def f(x):
            return cos(x)

        a, b = 0, 1
        I_eksakt = sin(1)           # Eksakt løsning
```

```
        S1, S2, feilestimat = simpson_basis(f, a, b)

        print('S1 = {:.8f}, S2 = {:.8f}'.format(S1, S2))
        print('Feil i S2 = {:.3e},  Feilestimat for S2 = {:.3e}'.format(I_eksakt-S2, feilestimat
```

```
S1 = 0.84177209, S2 = 0.84148938
Feil i S2 = -1.840e-05,  Feilestimat for S2 = -1.885e-05
```

Numerisk eksperiment Gitt det ubestemte integralet (med ubestemt løsning):

$$\int \frac{1}{1+16x^2}dx = \frac{\arctan(4x)}{4}.$$

1. Bruk `simpson_basis` for å finne en tilnærmelse til integralet over intervallet $[0,8]$. Skriv ut $S_2(0,8)$, feilestimatet $\mathcal{E}_2(0,8)$ og målt feil $E_2(0,8)$. Er feilestimatet pålitelig? 2. Gjenta eksperimentet over intervallene $[0,1]$ og $[4,8]$. Legg merke til forskjellen i målt feil over de to intervallene. 3. Gjenta eksperimentet igjen over intervallet $[0,0.1]$

```
In [6]: def runge(x):
            return 1/(1+16*x**2)

        a, b = 0, 0.1
        I_eksakt = 0.25*(arctan(4*b)-arctan(4*a))           # Eksakt løsning

        S1, S2, feilestimat = simpson_basis(runge, a, b)

        print('S1 = {:.8f}, S2 = {:.8f}'.format(S1, S2))
        print('Feil i S2 = {:.3e},  Feilestimat for S2 = {:.3e}'.format(
               I_eksakt-S2, feilestimat))
```

```
S1 = 0.09513705, S2 = 0.09512722
Feil i S2 = -6.282e-07,  Feilestimat for S2 = -6.550e-07
```

Observasjoner av eksperimentet: 1. Over intervallet $[0,8]$: Stor feil, dårlig feilestimat. 2. Over intervallet $[0,1]$: Stor feil, dårlig feilestimat. Over intervallet $[4,8]$: Liten feil, greit feilestimat. 3. Over intervallet $[0,0.1]$, liten feil, greit feilestimat.

Forklaring

Fra Section **??** har vi

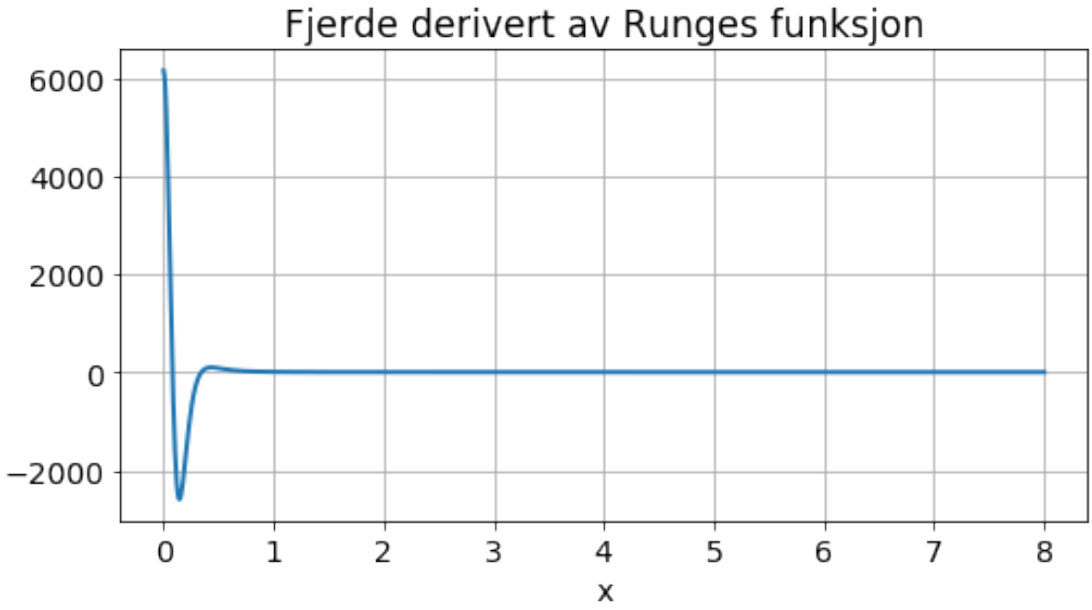
$$E(a,b) = -\frac{(b-a)^5}{2880}f^{(4)}(\xi). \tag{1}$$

Så vi må forvente at feilen er stor når $f^{(4)}(x)$ er stor. Videre antok vi i utledningen av feilestimatet at $f^{(4)}(x)$ var nogenlunde konstant over intervallet.

Så det kan være en god grunn til å ta en titt på den 4. deriverte av vår funksjon:

$$f(x) = \frac{1}{1+16x^2} \qquad \Rightarrow \qquad f^{(4)}(x) = 6144\frac{1280x^4 - 160x^2 + 1}{(1-16x^2)^5}$$

```
In [7]: # Lager et plott av den 4. deriverte av 1/(1+16x^2)
        def d4f(x):
            return 6144*(1280*x**4-160*x**2+1)/((1+16*x**2)**5)

        x = linspace(0, 8, 1001)
        plot(x, d4f(x))
        xlabel('x')
        title('Fjerde derivert av Runges funksjon');
```



De er dermed ingen overraskelse at feilen er stor og feilestimatet svikter over intervaller som inkluderer $[0, 1]$. Dette intervallet må partisjoneres i langt mindre biter for at vi skal få et meningsfyllt resultat.

Men hvordan kan intervallet partisjoneres når $f^{(4)}(x)$ ikke er kjent?

3.0.4 Adaptiv integrasjon

Gitt at en grunnleggende integrasjonsrutine som gir en tilnærmelse $Q(a,b)$ med et feilestimat $\mathcal{E}(a,b)$.

Vi ønsker å finne en partisjon av intervallet $[a,b]$

$$a = X_0 < X_1 \cdots < X_m = b$$

slik at

$$|\mathcal{E}(X_j, X_{j+1})| \approx \frac{X_{k+1} - X_k}{b-a}\text{Tol}$$

der Tol er en toleranse gitt av brukeren. I så fall vil

$$\text{Samlet feil} \approx \sum_{j=0}^{m-1} \mathcal{E}(X_k, X_{k+1}) \leq \text{Tol}.$$

Adaptiv integrasjon Gitt f, a, b og Tol. * Beregn $Q(a,b)$ og $\mathcal{E}(a,b)$. * **if** $|\mathcal{E}(a,b)| \leq$ Tol: * Godkjenn resultatet, returner $Q(a,b) + \mathcal{E}(a,b)$. * **else**: * La $c = (a+b)/2$, og prosessen på delintervallene $[a,c]$ og $[c,b]$, med toleranse Tol/2. * De godkjente resultatene fra hvert delintervall summeres underveis.

Implementasjon Den adaptive algoritmen er implementert med Simpsons metode som basismetode. `simpson_basic` er lett omskrevet fra funksjonen over, den returnerer bare $S2$ og feilestimatet, det er det eneste som behøves i den adaptive algortimen.

`simpson_adaptive` er en rekursiv funksjon (funksjon som kaller seg selv). For å unngå at den gjør så i det uendelige, er det lagt til en ekstra variabel `level` som øker med 1 for hver gang funskjonen kalles av seg selv, og en maksverdi for denne.

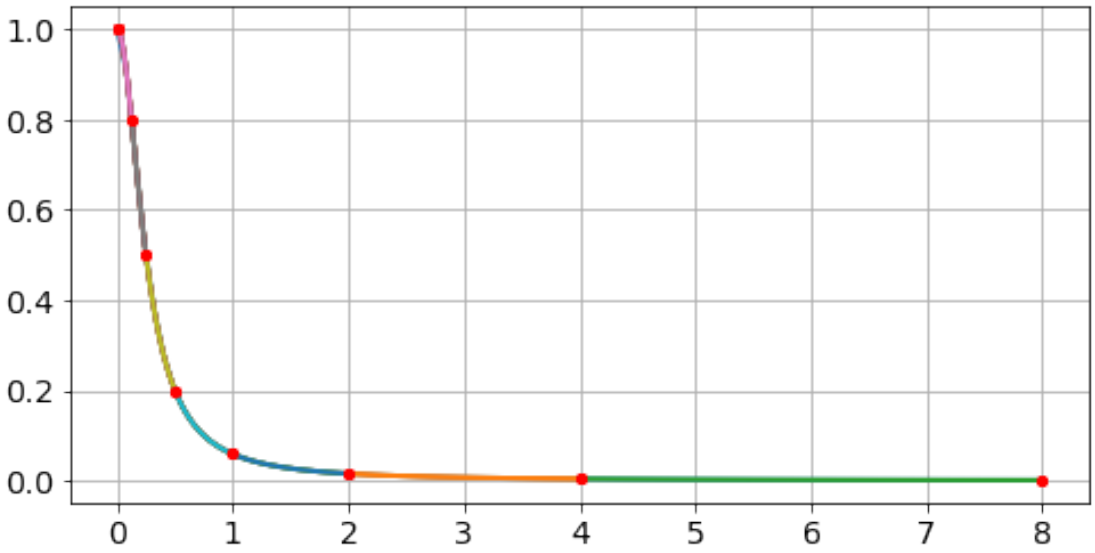
```
In [8]: def simpson_basis(f, a, b):
            # Regner ut en tilnærmelse til integralet av f over [a,b].
            # Inn: Funksjonen f, og integrasjonsintervallet [a,b]
            # Ut: S_1(a,b), S_2(a,b) og feilestimatet for S_2(a,b).

            # Finner nodene
            c = 0.5*(a+b)
            d = 0.5*(a+c)
```

```
    e = 0.5*(c+b)

    # Regner ut S1=S_1(a,b), S2=S_2(a,b) og feilestimatet for S2
    H = b-a
    S1 = H*(f(a)+4*f(c)+f(b))/6
    S2 = 0.5*H*(f(a)+4*f(d)+2*f(c)+4*f(e)+f(b))/6
    feilestimat = (S2-S1)/15
    return S2, feilestimat

def simpson_adaptive(f, a, b, tol = 1.e-6, level = 0, maks_level=15):
    # Finner en tilnærmelse til integralet av funksjonen
    # f over intervallet [a,b], gitt toleransen tol.

    Q, feil_estimat = simpson_basis(f, a, b)

    # ---------------------------------------------------
    # Utskrift og plott for demonstrasjonens skyld.
    if level == 0:
        print(' l   a           b         feil_est   tol')
        print('==============================================')
    print('{:2d}   {:.6f}   {:.6f}   {:.2e}   {:.2e}'.format(
            level, a, b, abs(feil_estimat), tol))

    x = linspace(a, b, 101)
    plot(x, f(x), [a, b], [f(a), f(b)], '.r')
    # ---------------------------------------------------

    if level >= maks_level:
        print('Maksimalt antall nivåer nådd')
        return S2

    if abs(feil_estimat) < tol:
        resultat = S2 + feil_estimat     # Godkjent, returnerer forbedret approx.
    else:
        # Deler intervallet i to, og bruker metoden på
        # hvert delintervall.
        c = 0.5*(b+a)
        resultat_venstre = simpson_adaptive(f, a, c, tol = 0.5*tol,
                                            level = level+1)
        resultat_hoyre   = simpson_adaptive(f, c, b, tol = 0.5*tol,
                                            level = level+1)
        resultat = resultat_hoyre + resultat_venstre
    return resultat
```

Eksempel Bruk adaptive Simpson til å finne en tilnærmelse til integralet $\int_0^5 1/(1+16x^2)dx$ med ulike toleranser, f.eks. Tol=$10^{-3}, 10^{-5}, 10^{-7}$. Sammenlign den numeriske løsningen med den eksakte.

```
In [9]: def runge(x):
            return 1/(1+(4*x)**2)

        a, b = 0, 8
        I_eksakt = 0.25*(arctan(4*b)-arctan(4*a))

        tol = 1.e-3
        resultat = simpson_adaptive(runge, a, b, tol=tol)

        # Utskrift
        print('\nNumerisk løsning = {:8f}, eksakt løsning = {:8f}'
              .format(resultat, I_eksakt))
        feil = I_eksakt - resultat
        print('\nToleranse = {:.1e}, feil = {:.3e}'.format(tol, abs(feil)))
```

```
 l   a           b         feil_est   tol
==============================================
 0   0.000000   8.000000   4.25e-02   1.00e-03
 1   0.000000   4.000000   1.85e-02   5.00e-04
 2   0.000000   2.000000   5.11e-03   2.50e-04
 3   0.000000   1.000000   7.84e-04   1.25e-04
 4   0.000000   0.500000   6.41e-04   6.25e-05
 5   0.000000   0.250000   3.43e-05   3.13e-05
 6   0.000000   0.125000   1.21e-06   1.56e-05
 6   0.125000   0.250000   1.31e-06   1.56e-05
 5   0.250000   0.500000   7.82e-07   3.13e-05
 4   0.500000   1.000000   1.45e-05   6.25e-05
 3   1.000000   2.000000   1.40e-05   1.25e-04
 2   2.000000   4.000000   8.29e-06   2.50e-04
 1   4.000000   8.000000   4.33e-06   5.00e-04

Numerisk løsning = 0.665849, eksakt løsning = 0.384889

Toleranse = 1.0e-03, feil = 2.810e-01
```



3.1 Andre kvadraturformler

Simpsons metode er bare et eksempel på kvadraturformler, og alt som er gjennomgått for Simpsons metode kan også gjøres for andre.

La meg til sist nevne noen vanlige klasser av metoder:

Newton-Cotes metoder Basert på jevnt fordelte noder over intervallet, se her for noen eksempler. Trapes og Simpson er eksempler på lukkede Newton-Cotes metoder, midtpunkt-metoden et eksemp på en åpen Newton-Cotes metode.

Gauss-Legendre kvadratur: Velg nodene som nullpunktene i polynomet

$$L_m(t) = \frac{d^m}{dt^m}(t^2 - 1)^m.$$

Disse har presisjonsgrad $d = 2m - 1$, og er det beste som kan oppnås med m noder. Midtpunktmetoden er et eksempel på et Gauss-Legendre kvadratur med $m = 1$.

4 Appendix

Den generaliserte middelverdisetningen

La $f \in C[a,b]$. Gitt k noder x_i i $[a,b]$ og k positive vekter $w_i > 0$, $i = 1,\ldots,k$. Da fins en $\eta \in [\min x_i, \max x_i]$ slik at

$$\sum_{i=1}^{n} w_i f(x_i) = f(\eta) \sum_{i=1}^{k} w_i.$$