Miniprojekt 1 – TANA21

Malcolm Vigren
malvi108@student.liu.se

Frans Skarman
frask812@student.liu.se

October 11, 2018

Innehållsförteckning

1 Introduktion

Numerisk integration är en viktig del i många beräkningar. Det är ofta svårt eller omöjligt att integrera godtyckliga funktioner och datamängder analytiskt, särskilt på datorer. Numerisk integration är dock inte exakt i det generella fallet, vilket innebär att approximationer måste göras. En av dessa metoder är trapetsmetoden, som approximerar integralen med hjälp av förstagradspolynom. Denna studie siktar på att utvärdera hur väl denna metod presterar, både noggrannhetsmässigt och beräkningskomplexitetsmässigt.

1.1 Uppgift

Syftet med denna undersökning är att utvärdera trapetsmetoden för numerisk beräkning av integraler. Noggrannhetsordningen samt den aritmetiska komplexiteten hos metoden ska utvärderas.

2 Teori

Trapetsregeln approximerar integralen genom att anpassa förstagradspolynom till funktionen som ska integreras, och beräkna arean under dessa. Detta kan beräknas med formeln

$$\int_{x_0}^{x_n} f(x)dx \approx T(h) = \frac{h}{2}(f(x_0) + 2\sum_{k=1}^{n-1} f(x_k) + f(x_n)) \tag{1}$$

där h är en steglängd, enligt $h = x_n - x_{n-1}$. Denna formel bör ha aritmetisk komplexitet $O(h)$, och noggranhetsordning $p = 2$.

2.1 Lösning

För att utvärdera noggranhetsordningen beräknades ett antal integraler analytiskt. Sedan beräknades det numeriska värdet av samma integraler med trapetsmetoden och olika steglängder. För beräkningen av noggranhetsordning användes steglängderna $h_i = \frac{1}{10^i}, i \in 1, 2, \ldots 5$. För att beräkna felet vid varje steglängd h_i beräknades differensen mellan de numeriskt bestämda värdena och de analytiskt bestämda värdena.

För att sedan beräkna noggranhetsordningen användes Ekvation 2, där R_t är felet som funktion av steglängden och c är en konstant.

$$\frac{|R_t(h_{i+1})|}{|R_t(h_i)|} = \frac{ch_{i+1}^p}{ch_i^p} = 10^p \Leftrightarrow p \approx \log_{10}\left(\frac{|R_t(h_{i+1})|}{|R_t(h_i)|}\right) \tag{2}$$

Följande funktioner användes för evaluering av noggranhetsordningen:

- Exponentialfunktionen e^x med primitiv funktion e^x, under intervallet $[0, 1]$
- Förstagradspolynomet $x + 1$ med primitiv funktion $\frac{x^2}{2} + x$, under intervallet $[0, 1]$
- Andragradspolynomet $x^2 + 2x + 1$ med primitiv funktion $\frac{x^3}{3} + x^2 + x$, under intervallet $[0, 1]$
- 100-gradspolynomet x^{100} med primitiv funktion $\frac{x^{101}}{101}$, under intervallet $[0, 1]$
- Funktionen $\frac{4}{1+x^2}$ med primitiv funktion $4\arctan(x)$, under intervallet $[0, 1]$
- Funktionen $\sqrt{x}$ med primitiv funktion $\frac{2x\sqrt{x}}{3}$, under intervallet $[0, 1]$
- Den periodiska funktionen $\sin^2(x)$ med primitiv funktion $\frac{1}{4}\sin(2x)$, under intervallet $[0, \pi]$

På ett liknande sätt beräknades metodens aritmetiska komplexitet. Tiden det tog att köra algoritmen vid olika steglängder bestämdes genom att utföra 100 iterationer av beräkningen och mäta tiden det tog. Precis som för noggranhetsordningen kan aritmetiska komplexiteten beräknas med Ekvation 3 där T betecknar exekveringstiden vid en viss steglängd. Här användes $h_i = \frac{1}{2^i}, i \in 20, 21, \ldots 25$.

$$\frac{|T(h_{i+1})|}{|T_t(h_i)|} = \frac{ch_{i+1}^p}{ch_i^p} = 2^p \Leftrightarrow p \approx \log_2\left(\frac{|T(h_{i+1})|}{|T_t(h_i)|}\right) \tag{3}$$

Följande funktioner användes för evaluering av noggranhetsordningen:

- Exponentialfunktionen e^x
- Andragradspolynomet $x^2 + 2x + 1$
- Funktionen $\frac{4}{1+x^2}$

3 MATLAB-kod

Följande kodlistning innehåller funktionen trapezoid som integrerar med trapetsmetoden.

```
1 function result = trapezoid(samples, step)
2     result = samples(1);
3     result = result + sum(samples(2:end-1) * 2);
4     result = result + samples(end);
5
6     result = step * result / 2;
7 end
```

I Bilaga A listas även koden som användes för evaluering av felet vid integration samt den aritmetiska komplexiteten av algoritmen.

4 Resultat

I detta avsnitt presenteras resultatet av undersökningen.

4.1 Noggranhetsordning

Tabell 1 ger värdet av testintegralerna uträknat med olika steglängd. I samma tabell ger också det analytiskt beräknade värdet av varje integral. Tabell 2 ger felet vid olika steglängder.

I Tabell 2 ges också den resulterande uppskattningen av noggrannhetsordningen p.

$\mathbf{1/h}$	$\mathbf{e^x}$	$\mathbf{x+1}$	$\mathbf{x^2+2x+1}$	$\mathbf{x^{100}}$	$\frac{\mathbf{4}}{\mathbf{1+x^2}}$	$\sqrt{\mathbf{x}}$	$\sin^{\mathbf{2}}(\mathbf{x})$
10^1	1.720	1.500	2.335	$5.000 \cdot 10^{-2}$	3.140	0.6605	1.571
10^2	1.718	1.500	2.333	$1.072 \cdot 10^{-2}$	3.142	0.6665	1.571
10^3	1.718	1.500	2.333	$9.909 \cdot 10^{-3}$	3.142	0.6667	1.571
10^4	1.718	1.500	2.333	$9.901 \cdot 10^{-3}$	3.142	0.6667	1.571
10^5	1.718	1.500	2.333	$9.901 \cdot 10^{-3}$	3.142	0.6667	1.571
Analytiskt	$e-1$	$\frac{3}{2}$	$\frac{7}{3}$	$\frac{1}{101}$	π	$\frac{2}{3}$	$\frac{\pi}{2}$

Tabell 1: Värdet av de olika integralerna uträknat med olika steglängd

$\mathbf{1/h}$	$\mathbf{e^x}$	$\mathbf{x+1}$	$\mathbf{x^2+2x+1}$	$\mathbf{x^{100}}$
10^1	$1.432 \cdot 10^{-3}$	0	$1.667 \cdot 10^{-3}$	$4.010 \cdot 10^{-2}$
10^2	$1.432 \cdot 10^{-5}$	$-2.220 \cdot 10^{-16}$	$1.667 \cdot 10^{-5}$	$8.202 \cdot 10^{-4}$
10^3	$1.432 \cdot 10^{-7}$	0	$1.667 \cdot 10^{-7}$	$8.332 \cdot 10^{-6}$
10^4	$1.432 \cdot 10^{-9}$	$-4.441 \cdot 10^{-16}$	$1.667 \cdot 10^{-9}$	$8.333 \cdot 10^{-8}$
10^5	$1.432 \cdot 10^{-11}$	$2.220 \cdot 10^{-16}$	$1.666 \cdot 10^{-11}$	$8.333 \cdot 10^{-10}$
p	2.00	Inf	2.00	1.94

$\mathbf{1/h}$	$\frac{\mathbf{4}}{\mathbf{1+x^2}}$	$\sqrt{\mathbf{x}}$	$\sin^{\mathbf{2}}(\mathbf{x})$
10^1	$-1.667 \cdot 10^{-3}$	$-6.157 \cdot 10^{-3}$	$2.220 \cdot 10^{-16}$
10^2	$-1.667 \cdot 10^{-5}$	$-2.037 \cdot 10^{-4}$	$4.441 \cdot 10^{-16}$
10^3	$-1.667 \cdot 10^{-7}$	$-6.532 \cdot 10^{-6}$	$6.661 \cdot 10^{-16}$
10^4	$-1.667 \cdot 10^{-9}$	$-2.075 \cdot 10^{-7}$	$7.772 \cdot 10^{-15}$
10^5	$-1.668 \cdot 10^{-11}$	$-6.570 \cdot 10^{-9}$	$7.994 \cdot 10^{-15}$
p	2.00	1.49	-0.25

Tabell 2: Fel vid för olika funktioner vid olika steglängder, samt potensen p

4.2 Aritmetisk komplexitet

I Tabell 4.2 presenteras exekveringstiderna för funktionerna med olika steglängder, samt dess beräknade aritmetiska komplexitet.

$\mathbf{1/h}$	$\mathbf{e^x}$	$\mathbf{x^2+2x+1}$	$\frac{\mathbf{4}}{\mathbf{1+x^2}}$
2^{20}	$1.377 \cdot 10^{-1}$s	$6.240 \cdot 10^{-2}$s	$6.049 \cdot 10^{-2}$s
2^{21}	$2.640 \cdot 10^{-1}$s	$1.521 \cdot 10^{-1}$s	$1.674 \cdot 10^{-1}$s
2^{22}	$6.366 \cdot 10^{-1}$s	$4.482 \cdot 10^{-1}$s	$4.611 \cdot 10^{-1}$s
2^{23}	$1.437 \cdot 10^{-1}$s	1.150s	1.148s
2^{24}	$2.853 \cdot 10^{-1}$s	2.357s	2.519s
2^{25}	$5.614 \cdot 10^{-1}$s	4.530s	4.967s
p	1.11	1.20	1.13

Tabell 3: Tid i sekunder för testerna av aritmetisk komplexitet, samt potensen p

5 Diskussion

De erhållna resultaten stämmer mestadels överens med vad som förväntades. De flesta av de experimentellt beräknade noggranhetsordningarna p var runt 2, vilket stämmer överens med teorin. I fallet med förstagradspolynomet och den periodiska funktionen blev det dock oändligt respektive -0.25, vilket beror på att dessa gav inte särskilt stora fel vid integrationen. De små eller nollvärda felen leder till nolldivision vid beräkningen av noggranhetsordningen, vilket orsakar de opålitliga noggranhetsordningarna.

Resultatet vid integrering av $\sqrt{x}$ är intressant. Felen var något större än för de andra funtionerna, och gav $p = 1.49$ istället för 2. Detta skulle kunna bero på att derivatan av $\sqrt{x}$ är odefinierad för $x = 0$, vilket leder till fel i uppskattningen av lutningen vid den första "trapetsen". I vilket fall tyder detta på att denna metod är mindre lämpad för integration av kvadratrötter.

Även den aritmetiska komplexiteten stämmer väl överrens med det teorin, som säger att den borde vara 1. Detta är mer tydligt med korta steglängder, då for-looparna tar en obetydlig tid att köra förhållande till integrationerna, vilket är varför mycket korta steglängder har använts här.

Bilagor

A MATLAB-kod för beräkning av fel och aritmetisk komplexitet

```
1
2  calculate_errors(@(x) exp(x), exp(1) - 1, 'exp', 0, 1);
3  calculate_errors(@(x) x + 1, 3/2, '1-degree', 0, 1);
4  calculate_errors(@(x) x.^2 + 2.*x + 1, 7/3, '2-degree', 0, 1);
5  calculate_errors(@(x) x.^100, 1/101, '100-degree', 0, 1);
6  calculate_errors(@(x) 4./(1+x.^2), pi, 'atan', 0, 1);
7  calculate_errors(@(x) sqrt(x), 2/3, 'sqrt', 0, 1);
8  calculate_errors(@(x) sin(x).^2, pi/2, 'periodic', 0, pi);
9
10 fprintf('\n\n\n');
11
12 arithmetic_complexity(@(x) exp(x), 'exp', 0, 1);
13 arithmetic_complexity(@(x) x.^2 + 2.*x + 1, '2-degree', 0, 1);
14 arithmetic_complexity(@(x) 4./(1+x.^2), 'atan', 0, 1);
15
16 function difference = calculate_errors(fun, expected, msg, start_x, end_x)
17     errors = [];
18     result = [];
19     for step_amount = [1:5]
20         step_size = 1/(10 ^ step_amount) * (end_x - start_x);
21         x = [0: step_size:  end_x];
22         vals = fun(x);
23         integrated = trapezoid(vals, step_size);
24         difference = integrated - expected;
25         errors = [errors difference];
26         result = [result integrated];
27     end
28
29     h_values = [];
30     for i = [2:length(errors)]
31         h_values = [h_values, errors(i - 1) / errors(i)];
32     end
33
34     fprintf('%s: ', msg);
35     fprintf('%.3d, ', errors);
36     fprintf('potens: %.2f, ', log10(mean(h_values)));
37     fprintf("\n");
38     fprintf('    values:')
39     fprintf('%.3d   ', result)
40     fprintf("\n");
41 end
42
43
44 function nothing = arithmetic_complexity(fun, msg, start_x, end_x)
45     result = [];
46     step_range = (1./(2.^[20:25])).*(end_x - start_x);
47     for step_size = step_range
48         tic;
49         for runs = [1:10]
50             x = [start_x: step_size:  end_x];
51             vals = fun(x);
52             integrated = trapezoid(vals, step_size);
53         end
54         result = [result toc];
55     end
56
57     % figure
58     % plot([20:25], log2(result));
59     % xlabel('Step size 1/2^x')
60     % ylabel('log2(time)')
61
```

```
62      h_values = [];
63      for i = [2:length(result)]
64          h_values = [h_values, result(i) / result(i - 1)];
65      end
66
67      fprintf('%s: ', msg);
68      fprintf('%.3d, ', result);
69      fprintf('potens: %.2f, ', log2(mean(h_values)));
70      fprintf("\n");
71  end
```