TANA21: Projektrapport

Numerisk linjär algebra

Namn	**Personnummer**	**Epostaddress**
Martin Söderén	900929-1098	marso329@student.liu.se
Alexander Yngve	930320-6651	aleyn573@student.liu.se

1 Inledning

Denna rapport behandlar hur man kan lösa polynom approximationer med hjälp av MATLAB och Newtons metod för approximation och vilka resultat man kan förvänta sig.

2 Uppgift

Projektets uppgift är att skapa två MATLAB-funktioner, mypolyfit som bestämmer ett interpolationspolynom av lägsta möjliga grad och mypolyval som som evaluerar det beräknade polynomet. Därefter ska den förstnämnda funktionens korrekthet kontrolleras och dess tidskomplexitet bestämmas.

Frågor att besvara:

- Räknar koderna rätt?
- Är kodernas aritmetiska komplexitet som förväntad?

3 Teori

Enligt sats 5.2.1 i kursboken så går det att bestämma ett entydigt polynom av grad $\leq$ n om det finns n+1 skilda punkter.

Genom att ansätta ett interpolationspolynom kan man skapa ett linjärt ekvationssystem. Detta kan man sätta upp i matrisform och då blir koefficientmatrisen triangulerad vilket leder till att denna kan lösas med antingen framåtsubstitution eller bakåtsubstitution. Detta ger tidskomplexiteten $\mathcal{O}(n^2)$ enligt kursbok kapitel 5.4.

$$f[x_i] = f(x_i)$$

$$f[x_1, x_1....., x_{k+1}] = \frac{f[x_2, x_3, ..., x_{k+1}] - f[x_1, x_2, ..., x_k]}{x_{k+1} - x_1}$$

Den k:te dividerade differensen av f

För utvärdering av polynom så används Horners schema som beskrivs i kursboken avsnitt 4.6. Fördeled med Horners är att den beräknar polynomet rekursivt och använder hälften som många multiplikationer jämfört med med om man evaluerar polynomet normalt.

4 Metod

Funktionerna implementerades enligt metoden beskriven i avsnitt 5.4 i kursboken. Därefter testades *mypolyfit* med koden i figur 1.

```
%clf;
for i = [5 10 20 40 80 160 320 640 1280]
    N = i;
    x = 1:N;
    y = randi([-20,20], 1, N);

    i
    tic;
    P = mypolyfit(x, y);
    toc;

    px = 1:0.1:N;
    py = mypolyval(P, 1:0.1:N);

    %plot(x, y, '*', px, py);
    %pause;
end
```

Figur 1 – Testkod

5 Kod

Funktionen *mypolyfit* (figur 2) bestämmer ett interpolationspolynom av lägsta möjliga grad och *mypolyval* (figur 3) evaluerar interpolationspolynomet.

```
function [n] = mypolyfit(x,y)
N = length(x)-1;
temp = zeros(N + 1,N + 1);
temp(1:N + 1,1) = y';
% Create difference table
for k=2:N+1
    for m=1:N+2-k
        temp(m,k) = (temp(m + 1,k - 1) - temp(m,k - 1))/(x(m + k - 1)- x(m));
    end
end
% First row in difference table (def 5.4.1)
a = temp(1,:);
n = a(N+1);
% Nested multiplication for coefficients
for k = N:-1:1
n = [n a(k)] - [0 n*x(k)];
end
```

Figur 2 – *mypolyfit*

```
function [ p ] = mypolyval( P, X )
n=length(P)-1;
p=P(1);
for i=1:n
    p=p.*X+P(i+1);
end
```

Figur 3 – *mypolyval*

6 Validering

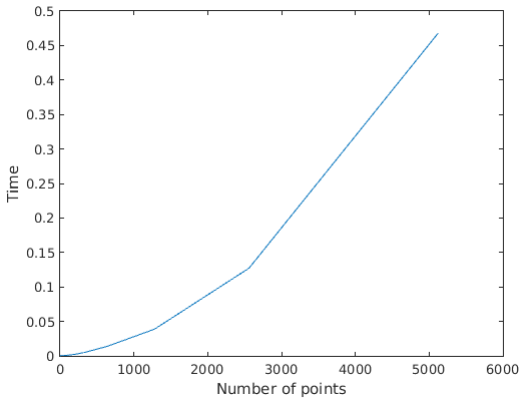
Om man väljer att plotta figurer i koden i figur 1 så ser man att alla punkter från polynomet ligger på den ursprungliga linjen. Vilket leder till att man kan anta att koden räknar rätt.

7 Resultat

Resultatet av testet beskrivet i avsnitt 4 återges i tabell 1.

Datapunkter	**Tid (s)**
5	0.000160
10	0.000244
20	0.000448
40	0.000787
80	0.000683
160	0.001644
320	0.005802
640	0.015361
1280	0.060299

Tabell 1 – Resultat



Figur 4 – Tiden plottad.

8 Diskussion

Vi har kört testkoden många gånger och inte fått ett fel någon gång så vi antar att koden räknar rätt. Vad gäller tidskomplexiteten så är datan väldigt ostabil vilket leder till att det är svårt att avgöra vilken tidskomplexitet den har men den verkar vara $\mathcal{O}(n^2)$ vilket är som förväntat.