TANA23 Matematiska algoritmer och modeller

Laboration 2. Ekvationer och Interpolation

1 Introduktion

Om vi vill använda en dator för att utföra matematiska beräkningar så måste vi acceptera beräkningsfel eftersom datorn arbetar med begränsad noggranhet. Syftet med denna laboration är att bli bekant med problem som uppstår på grund av att datorn inte kan räkna helt exakt.

Redovisning sker genom att en kort rapport skrivs där svaret på alla frågor finns med. Program skrivna i Python skall bifogas. Tänk på att programmen skall skickas på ett format som gör att jag kan testköra dem. Enklast är att bara lägga in all kod i en textfil med lite kommentarer angående vilken uppgift som ett visst kodstycke löser. Bifoga även grafer. Den färdiga rapporten skall skickas med epost till `fredrik.berntsson@liu.se`. Var noga med att skriva *TANA23 Lab1* i rubriken.

2 Ekvationslösning

I denna uppgift skall vi beräkna lösningen till ekvationen

$$f(x) = \sqrt{1+x}\mathrm{e}^{x/2} - 2\sin(2x)(x+x^2) = 0.$$

med hjälp av funktionen `fsolve` som finns i paketet `scipy.optimize`. Vi skall även undersöka dess konvergenshastighet. Vi är intresserade av roten $x^* \approx 1.35$.

Uppgift 2.1 Rita en graf över funktionen $f(x)$ på intervallet $[0, 3]$. Bekräftar grafen att det finns en rot $x^* \approx 1.35$. **Redovisa grafen**.

Tips Vi kan skapa en vektor med jämt utspridda x-värden på intervallet med hjälp av funktionen `linspace` från paketet `numpy`. Beräknar vi sedan en vektor `f` med motsvarande funktionsvärden så kan vi plotta genom att skriva

```
>>> import matplotlib.pyplot as pp
>>> pp.plot(x,f)
>>> pp.show()
```

Vi skall nu skapa en Python funktion som implemneterar uttrycket ovan. För att kunna följa vilka funktionsanrop som görs av `fsolve` så lägger vi till en rad där aktuellt x-värde och dessutom $f(x)$ skrivs ut. Lämplig Python kod kan exempelvis vara

```
def funk( x ):
    import numpy as np
    f=np.sqrt(1+x)*np.exp(x/2)-2*np.sin(2*x)*(x+x**2)
    print('x=',x,'  f(x)=',f)
    return f
```

Uppgift 2.2 Lös ekvationen $f(x) = 0$ med hjälp av `fsolve`. Använd start gissningen $x_0 = 1.35$. Hur många funktionsberäkningar behövs? Ange approximationen $\bar{x} \approx x^*$ och funktionsvärdet $f(\bar{x})$.

Uppgift 2.3 Kopiera de x-värden där funktionen $f(x)$ beräknats då ekvationen löstes av `fsolve` till en vektor `x`. Antag att det sista x-värdet är exakt och beräkna felet för övriga iterationer x_k. Ser detta ut som kvadratisk konvergens? Förklara din slutsats. **Redovisa koden**

3 Beräkning av Division

Då en ny processor skall byggas måste man först bestämma vilka operationer som skall implementeras i hårdvara och vilka som kan implementeras med mjukvara. Färre instruktioner betyder att processorn blir enklare till sin konstruktion. En grundläggande instruktion som ibland lämnas utan hårdvarustöd är division, alltså beräkningen $z := x/y$. I denna övning skall vi visa hur en effektiv mjukvaruimplementation av division kan tänkas se ut. Problemet är väldigt enkelt men samtidigt är det viktigt att få en så snabb, och pålitlig, kod som möjligt. Vi måste också förvissa oss om att resultatet är noggrannt.

Uppgift 3.1 Problemet att beräkna $z = 1/y$ kan formuleras som en icke-linjär ekvation,

$$f(z) = y - \frac{1}{z} = 0.$$

Tillämpa Newton-Raphsons metod på ekvationen $f(z) = 0$ och härled motsvarande iterationsformel $z_{k+1} = \phi(z_k)$. Du måste förenkla funktionen $\phi(z)$ på ett sådant sätt att ingen division ingår i uttrycket.

Kommentar En tanke vore att försöka beräkna $z = x/y$ direkt men då går det inte att undvika divisioner i den iterationsformel man får efter att ha tillämpat Newton-Raphsons metod. Pröva gärna!

Uppgift 3.2 Eftersom datorn använder ett talsystem där reella tal representeras som

$$y = \pm(1.f)_2 2^e,$$

och $(2^e)^{-1} = 2^{-e}$ behöver vi endast beräkna divisionen $1/y$ för $1 \leq y < 2$. Detta betyder att roten z^* till ekvationen $f(z) = 0$ alltid satisfierar $\frac{1}{2} < z^* \leq 1$. Vi väljer att använda start gissningen $z_0 = \frac{3}{4}$. Uppskatta det initiala felet $|z^* - z_0|$.

Uppgift 3.3 Konvergensanalysen för Newton-Raphsons metod visar att

$$|z^* - z_{k+1}| = \frac{\phi''(\eta)}{2}|z^* - z_k|^2,$$

där η ligger i intervallet (z_k, z). Utnyttja detta för att bestämma det minsta antal iterationer som krävs för att $|z^* - z_k| \leq \mu$, där μ är maskinkonstanten, skall gälla. **Redovisa beräkningarna**.

Tips Hitta maximum av $|\phi''(\eta)|$ på intervallet $\frac{1}{2} < \eta \leq 1$.

Uppgift 3.4 Utnyttja dina resultat ovan för att skriva en funktion

```
>>>  z = Division( x , y )
```

Funktionen skall lösa problemet i två steg. Först beräknas $1/y$ och sedan multipliceras $z = x \cdot (1/y)$. Du får förenkla problemet genom att förutsätta att $1 \leq y < 2$.

Uppgift 3.5 Utnyttja din funktion för att beräkna $z = 1.32/y$, för ett stort antal y-värden i intervallet $1 \leq y < 2$. Jämför resultatet från din funktion med Pythons division $1.32/y$. Plotta absolutbeloppet av skillnaden mellan de två beräkningarna och kommentera resulatet. **Redovisa grafen**.

Kommentar Det är möjligt att vi får snabbare konvergens för vissa y-värden men genom att välja antalet iterationer från det teoretiska resonemanget ovan kan vi undvika villkorssatser i koden. Det gör att en division alltid beräknas precis lika snabbt. Förutsägbar tidsåtgång är viktigt.

Uppgift 3.6 Det sista vi skall göra är att välja ett bättre startvärde z_0. Vi delar upp intervallet $[1, 2)$ i 8 lika stora delintervall $y_k \leq z \leq y_{k+1}$, $k = 0, 1, 2, \ldots, 7$, där $y_k = 1 + k/8$. Vi väljer sedan en lämplig startgissning z_0 beroende på vilket y_k som ligger närmast vårt y-värde. Startgissningar $z_k = 1/y_k$ lagras i en tabell. Visa först att det nu räcker med $n = 4$ Newton-Raphson iterationer. Vi kan alltså använda mer minne (för att larga en tabell) och spara beräkningar. Komplettera din kod så att detta implementeras. Upprepa sedan jämförelsen med Pythons divisions uträkning. **Redovisa både grafen och din funktion**.

Tips Det finns en funktion `round()` i Python som kan användas för att beräkna ett det index k som gör att $|y_k - y|$ minimeras. Gör ett variabelbyte så att alla delintervall får längden 1. Här får givetvis ingen division användas.

4 Polynominterpolation med Tillämpning

I tillämpningar hittar man ofta funktioner som är väldigt beräkningskrävande. Dock behöver man inte alltid särskilt hög noggranhet. I denna övning skall vi implementera $f(x) = \arctan(x)$, på intervallet $1 \leq x \leq 2$, med ett absolut fel $|\Delta f| \leq 10^{-6}$, genom att interpolera en tabell med polynom av olika gradtal.

Uppgift 4.1 Polynominterpolation finns implementerat i paketet `numpy`. Vi har särskilt två funktioner `polyfit` och `polyval`. Vi vill hitta ett andragradspolynom $p_2(x)$ som interpolerar följande tabell

x	0.9	1.1	1.2
$f(x)$	0.4710	0.2452	0.2385

och skriver

```
import numpy as np
x=[0.9 , 1.1 , 1.2]
f=[0.4710 , 0.2452 , 0.2385]
p2=np.polyfit(x,f,2)
```

Vill vi beräkna $p_2(x)$, för exempelvis $x = 1.15$, skriver vi `np.polyval(p2,1.15)`. Vill vi istället plotta både punkter och polynom på intervallet $0.9 \leq x \leq 1.2$ skriver vi

```
n=100
xx=np.linspace(0.9,1.2,n)
pvals=np.polyval(p2,xx)
import matplotlib.pyplot as pp
pp.plot(x,f,'x')
pp.plot(xx,pvals)
pp.show()
```

Pröva detta.

Nu skall vi studera problemet med att approximera $\arctan(x)$ på aktuellt intervall.

Uppgift 4.2 Först väljer vi att approximera $f(x) = \arctan(x)$, på intervallet $1 \leq x \leq 2$, med hjälp av förstagradspolynom. Vi väljer $n + 2$ stycken interpolations punkter $x_k = 1 + (k - 1)/(n - 1)$, $k = 0, 1, 2, \ldots, n, n + 1$, och beräknar $y_k = \arctan(x_k)$. För att beräkna en approximation av $\arctan(x)$ för att visst x-värde hittar vi ett index k sådant att $x_k \leq x < x_{k+1}$. Vi beräknar sedan ett interpolerande polynom $p_1(x)$ och approximerar $\arctan(x) \approx p_1(x)$.

Implementera ovanstående metod i en funktion `ApproxArctan` som tar ett x-värde som inargument och beräknar motsvarande funktionsvärde.

Tips Du kan välja ett värde n först i din funktion och skapa interpolationspunkterna x_k med `x=np.linspace(1-1/(n-1),2+1/(n-1),n+2)`. Beräkna sedan tabellvärdena med `y=np.arctan(x)`. Det är givetvis vansinnigt att beräkna funktionen $\arctan(x)$ n gånger som ett delmoment i ett Python program som skall beräkna $\arctan(x)$ en gång effektivt men det ignorerar vi för tillfället. Notera dessutom att vi har två interpolationspunkter utanför aktuellt intervall. Dessa fyller ingen funktion ännu men underlättar senare.

Uppgift 4.3 Välj att använda $n = 5$ tabellvärden. Beräkna $\arctan(x)$ både med `numpy` och med din funktion ovan för 1000 jämntutspridda tal på intervallet $1 \leq x \leq 2$. Plotta absolutbeloppet av skillnaden mellan dem. Hur stort är det maximala felet i din approximation? **Redovisa grafen**.

Uppgift 4.4 Då linjär interpolation används på varje delintervall $x_k < x < x_{k+1}$ så gäller att

$$R_T = p_1(x) - f(x) = \frac{f^{(2)}(\eta(x))}{2!}(x - x_k)(x - x_{k+1}) \quad x_k \leq \eta(x) \leq x_{k+1}.$$

För att hitta en övre gräns för trunkeringsfelet som beror på antalet interpolationspunkter n så skall vi göra följande: Först låt $f(x) = \arctan(x)$ och beräkna $f^{(2)}(x)$. Vi hittar sedan

$$\max_{1 \leq x \leq 2} |f^{(2)}(x)|.$$

Det andra vi måste göra är att maximera uttrycket

$$g(x) = |(x - x_k)(x - x_{k+1})|, \quad x_k \leq \eta(x) \leq x_{k+1}.$$

För att kunna arbeta med uttrycket måste vi göra oss oberoende av interpolationspunkterna x_k. Vi inför därför en steglängd $h = x_{k+1} - x_k$, och gör ett variabelbyte $z = (x - x_k)/h$. Då skall vi istället maximera

$$g(z) = h^2|z(z - 1)|, \quad 0 \leq z \leq 1.$$

Detta problem går att lösa genom funktionsundersökning. Problemet underlättas genom att resonera kring symmetri för $g(z)$.

Slutligen, använd resultaten ovan och hitta det värde n som gör att $|R_T| \leq 10^{-6}$. **Redovisa dina beräkningar**

Tips Denna uppgift kan lösas analytiskt med kunskaper från Envariabelanalysen men *Wolfram Alpha* kan hitta rätt enkla uttryck för derivator av $\arctan(x)$. Det går att utnyttja uttrycket för $f^{(3)}(x)$ för att inse att maximum, för $f^{(2)}(x)$, fås för $x = 1$.

Uppgift 4.5 Använd det värde på n du fick i föregående uppgift och beräkna återigen $\arctan(x)$ både med `numpy` och med din funktion ovan för 1000 jämntutspridda tal på intervallet $1 \leq x \leq 2$. Plotta absolutbeloppet av skillnaden mellan dem. Hur stort är det maximala felet i din approximation nu? **Redovisa grafen**.

Vi ser att det krävs en rätt stor tabell för att få tillräcklig noggranhet. Det beror till stor del på att felet vid linjär interpolation beror på h^2. Det gör att tabell storleken beror på $\sqrt{\varepsilon}$, där ε är det absoluta fel som efterfrågas. Ett sätt att minska tabellstorleken är att välja en metod med högre noggranhetsordning. Det skall vi titta på nu.

Uppgift 4.6 Kubisk interpolation innebär att interpolations punkter x_{k-1}, x_k, x_{k+1}, och x_{k+2} används för att approximera funktionen $f(x)$ i intervallet $x_k < x < x_{k+1}$. Vi behöver alltså fyra interpolationspunkter för att skapa ett tredjegradspolynom $p_3(x)$. Ändra i din Python funktion `ApproxArctan` så att linjär interpolation byts mot kubisk. Välj även nu $n = 5$ och plotta skillnaden mellan din funktion och den från `numpy`. Hur stort blir det maximala absoluta felet i approximationen? **Redovisa grafen**

Tips Här ser vi poängen med att ha två extra interpolationspunkter utanför intervallet $1 \leq x \leq 2$. Dessa behövs då x tillhör något av intervallen $[x_1, x_2]$ eller $[x_{n-1}, x_n]$.

Uppgift 4.7 Då kubisk interpolation används gäller att

$$R_T = p_3(x) - f(x) = \frac{f^{(4)}(\eta(x))}{4!}(x - x_{k-1})(x - x_k)(x - x_{k+1})(x - x_{k+2}) \quad x_{k-1} \leq \eta(x) \leq x_{k+2}.$$

En övregräns för $|R_T|$ kan hittas på precis samma sätt som tidigare. Räkningarna blir dock mer komplicerade. Istället kan vi inse att resultatet kommer att bli en felfräns av typen

$$|R_T| \leq Ch^4,$$

där C är en konstant. Vi kan då välja exempelvis $n = 5$ och beräkna det maximala absoluta felet som fås (då exempelvis 1000 jämnt utspridda punkter testas) och utnyttja detta för att hitta ett experimentellt värde på konstanten C. Detta C värde används sedan för att bestämma ett h sådant att $|R_T| \leq 10^{-6}$. Genomför detta och hitta en lämplig tabell storlek n. Verifiera även att det absoluta felet blir tillräckligt litet genom att beräkna, och plotta, det absoluta felet i approximationen av $\arctan(x)$ för 1000st jämnt utspridda punkter. Hur stort blir det maximala felet på intervallet $[1, 2]$? **Redovisa dina beräkningar och grafen**

Uppgift 4.8 Det sista vi skall göra är att lösa problemet med att $\arctan(x)$ måste beräknas för interpolationspunkterna då funktionen anropas. Lösningen är helt enkelt att skriva ut de aktuella funktionsvärdena $\arctan(x_k)$ med Python och sedan kopiera in rätt siffervärden i koden. **Redovisa koden**.