TMV136 Matematisk analys i en variabel E, 2010

MATLAB — NUMERISK INTEGRATION, KVADRATUR

Beskrivning och mål.

Den här laborationen har som syfte att lära ut några av Matlabs färdiga funktioner för numerisk integration (kvadratur) samt att genom att uppmana laboranten att skriva en egen sådan funktion få denne att förstå hur dessa fungerar. Genomförd laboration bör också ge en insikt i vikten av att välja *rätt* metod för den aktuella funktionen. I övrigt skall förmågan att använda Matlab i allmänhet, och i synnerhet att läsa hjälptext, fördjupas. Mer specifikt är målet att du ska kunna:

- använda Matlabs inbyggda kvadraturfunktioner.
- kunna förstå viktiga delar av hjälptexten i dessa.
- därifrån dra slutsatser om vilken funktion du lämpligen bör använda på den aktuella integralen.
- beskriva hur enkla kvadraturmetoder (i synnerhet Simpsons regel) fungerar och
- skriva program som använder sig av dessa för att approximera värdet på en integral.
- redogöra för de problem som uppstår när man försöker lösa generaliserade integraler med kvadratur och
- kunna redogöra för hur man överkommer dessa problem.
- skriva `for`-loopar.
- skriva funktionsfiler med korrekta in- och utargument samt helt behärska funktionsanrop.
- redogöra för vinkelfrekvensens inverkan på grafen till en sinusfunktion.
- redogöra för medelvärdets och standardavvikelsens inverkan på normalfördelningskurvan.
- diskutera begreppet effektivitet hos ett program i termer av tid, minne och operationer med handledaren.

Arbeta två och två och diskutera er fram till lösningar. Se till att båda har förstått vad ni gör. Spara alla filer på bådas användarkonton och kommentera programmen på egen hand efter laborationstillfället för att repetera. Det kan vara lämpligt att den som har minst programmeringsvana sitter vid tangentbordet, eller att ni turas om.

Redovisa genom att visa html-dokument genererat av Matlabs `publish` samt den aktuella Matlab-koden för handledaren. Deklarera alla matematiska funktioner som explicita Matlab-funktioner, dvs i funktions*filer*. Redovisning sker under andra halvan av laboration två samt den första delen av laboration tre. Vi kommer göra hårdare bedömningar under det senare tillfället. Den som inte blir godkänd kommer att få lämna in skriftliga kompletteringar. Avsnitt 1.3 behöver ni bara ha förstått vad probelmet är och lösa Uppgift 1.3.1 A och utföra variabelsubstitutionen i C.

Ni har två laborationstillfällen på er och vi beräknar att det kommer att gå åt minst lika mycket tid utöver detta för att klara det hela. Fredrik svarar på frågor även när det inte är laborationstid.

Date: 2010–11–09, Fredrik Lindgren, Matematiska vetenskaper, Chalmers tekniska högskola.

1. Uppgifter

1.1. **Matlabs inbyggda kvadraturfunktioner.** Vi vill alltså beräkna olika integraler numeriskt.

1.1.1. *Inledande uppgift.* Börja med att leta efter lämpliga funktioner för det genom att skriva till exempel

```
lookfor quadrature
```

eller

```
lookfor integral
```

i kommandofönstret. Av de funktioner man då får utskrivna kan man nog gissa sig till att `quad` är den mest basala. Läs dess hjälptext och försöka att beräkna följande integraler numeriskt.

$$\int_0^2 (1+x^2)\,dx \tag{1}$$

$$\int_0^1 e^{-x}\,dx \tag{2}$$

$$\int_0^1 \frac{1}{1+x^2}\,dx. \tag{3}$$

Hur många korrekta decimaler (`format long`) får du? Hur lång tid tar beräkningen (`tic, toc`)?

I slutet av hjälptexten finns ett stycke med rubriken *Notes*. Där beskrivs när fyra olika Matlabfunktioner bör användas. Den fjärde, `quadv` är ointressant för oss men försök förstå när de andra tre är bra. Ordet *smooth* (*glatt* på svenska) betyder här att derivatorna är förhållandevis små[1], att funktionen är relativt flack. Funktionen

$$f(t) = \cos(\omega t) \tag{4}$$

är i den bemärkelsen glatt när när ω är litet men inte när det är stort. Den är också starkt oscillerande för stora ω.

1.1.2. *Jämförelse av olika kvadraturprogram.*

A Rita grafen till funktionen (4) för $\omega = 1, 10, 100$ och 1000 i samma figur och se till att du kan förklara vad du ser i termer av begreppet 'glatt'. Se till att du har tillräcklig upplösning, se Laboration 2 i förra kursen!

B Vilken av funktionerna `quad, quadl` respektive `quadgk` borde ge bäst resultat för små respektive stora värden på ω om man får tro *Notes*? Vilken borde ge sämst resultat?

C Skriv en `for`-loop som för $\omega = 2^k$, $k = 0, 1, \ldots, 15$ beräknar integralen

$$I = \int_0^{\pi/4} \cos(\omega t)\,dt \tag{5}$$

med hjälp av de tre olika kvadraturmetoderna och sparar resultaten i tre olika vektorer. Plotta dessa som punkter som funktion av k i en och samma figur och jämför med de riktiga värdena på integralerna. Det kan vara vettigt att plotta dessa värden som punkter och inte som linjer. Blev det som man kunde ana? Ta dessutom tid på varje metod för sig och plotta tidsåtgången som funktion av k för alla metoderna. Vad är det du ser i figuren?

D Ändra toleransen `TOL` för `quad` och `quadl` till säg 10^{-14}. Vad blir skillnaden? Kunde man förutse det genom att läsa *Notes*?

E Integralen

$$\int_0^1 \frac{1}{x^\alpha}\,dx, \quad 0 < \alpha < 1$$

är singulär i $x = 0$. Vilken av funktionerna borde fungera bäst för att beräkna den integralen? Ta tid och testa om det stämmer! Singulariteten blir värre när α närmar sig ett. Jämför med det analytiskt erhållna värdet!

[1] Glatt betyder väldigt olika saker i olika sammanhang och man bör ge akt på vad författaren eller föreläsaren menar när ordet används.

1.2. **Egna kvadraturmetoder.** För att få en bild av hur integrationsmetoder fungerar och för att öva oss i att skriva program ska vi skriva våra egna kvadraturfunktioner. I [Adams, Kap 6.6-6.8] beskrivs några algortitmer i detalj (trapetsregeln, mittpunktsregeln, Simpsons regel samt Romberg-integration/Richardson-extrapolation). En förfinad variant av Simpsons regel används i `quad`. Funktionen `quadgk` använder en variant av Gauss metod som nämns i slutet av kapitel 6.8 i [Adams]. Vi tittar närnare på Simpsons metod [Adams, sid.376] som innebär att man approximerar integralen

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

genom att dela upp intervallet $[a, b]$ på ett *jämnt* antal, n, delintervall. Det görs genom att man sätter $h = (b-a)/n$ och sedan $x_0 = a$ och $x_i = a + ih$ för $i = 0, 1, 2, \ldots, n$. Därefter räknas funktionsvärdena ut i dessa punkter, dvs $y_i = f(x_i)$ varvid summan

$$S_n = \frac{h}{3}\left(y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \ldots + 2y_{n-2} + 4y_{n-1} + y_n\right)$$

evalueras. Vad som har hänt är att funktionsvärdena har multiplicerats med en vikt, hw_i, innan de summeras ihop där

$$w_i = \begin{cases} \dfrac{1}{3} \text{ om } i \in \{0, n\}, \\ \dfrac{2}{3} \text{ om } i \text{ är jämn och} \\ \dfrac{4}{3} \text{ om } i \text{ är udda.} \end{cases}$$

1.2.1. *Simpsonkvadratur med for-loop.* Skriv en *funktionsfil* `myquad1.m` som tar fyra inargument där de tre första är samma som i t.ex. `quad` och det fjärde är det jämna antalet intervall som vi vill dela upp integrationsområdet i. Programmet ska sedan på något sätt skapa en vektor på formen

$$wy = (w_0 y_0, w_1 y_1, \ldots, w_{n-1} y_{n-1}, w_n y_n) \tag{6}$$

för att sedan summera ihop alla dessa tal i en `for`-loop. Funktionen ska returnera det approximativa värdet på integralen. Testa på funktionerna (1)-(3). Notera att din funktion i vissa fall blir bättre!

1.2.2. *Matlabs `sum`.* Spara om filen från föregående uppgift som `myquad2.m` och ersätt `for`-loopen med Matlabs `sum`. Vilken går snabbast, `myquad1` eller `myquad2`? Testa med `tic` och `toc`.

1.2.3. *Jämför med `quad`.* Beräkna integralen

$$\int_0^{\pi/4} \sin(x)\,dx$$

för hand och med `quad`. Hur många korrekta decimaler får du? Hur många funktionsevalueringar gör `quad`? Hur många evalueringar måste du göra med din egen `myquad2` för att få lika många decimalers noggranhet? Skillnaden ligger i adaptiviteteten hos `quad`. Den skapar korta intervall där felet hotar att bli stort och större där det inte är så stort. På så vis hushåller den med antalet uträkningar.

1.2.4. *Ett annat sätt...* Vad gör följande Matlabkod?

```
h=(b-a)/n;
x=a:h:b;
y=f(x);
w=2/3*ones(size(y));
w([1,end])=1/3;
w(2:2:end-1)=4/3;
I=dot(y,w);
I=I*h; %Varför bör man multiplicera med h här och inte förr?
```

Här är f ett godtyckligt funktionshandtag till en funktion $f : \mathbb{R} \to \mathbb{R}$.

1.3. **Indefinita integraler och dylikt.**

1.3.1. *Oändliga intervall och normalfördelningar.* Betrakta funktionen $f(x) = e^{-x^2}$. Den här funktionen har ingen primitiv funktion i termer av andra enkla funktioner. Däremot kan man visa att den generaliserade integralen

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$$

varvid det naturligtvis följer att

$$\frac{1}{\sqrt{\pi}}\int_{-\infty}^{\infty} e^{-x^2}\,dx = 1.$$

A Visa med hjälp av variabelsubstitution att

$$\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{\infty} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx = 1.$$

för alla $\mu \in \mathbb{R}$, $\sigma > 0$.

Funktionen $g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$ är en extremt viktig funktion inom sannolikhetsteorin. Här vill man ofta beräkna integraler i stil med

$$\frac{1}{\sqrt{2\pi\sigma^2}}\int_{-\infty}^{b} e^{-\frac{(x-\mu)^2}{2\sigma^2}}\,dx \tag{7}$$

som till exempel är sannolikheten att ett normalfördelat slumptal har ett värde mindre än b.

B Rita grafen till funktionen g, först med $\mu = 0$ och $\sigma = 0, 1, 1, 10$ i samma figur och sedan med $\sigma = 1$ och $\mu = -5, 0, 5$ i en annan figur.

Eftersom ingen känd primitiv funktion finns i detta fall är man tvungen att beräkna detta numeriskt. Här har vi ytterligare en komplikation, nämligen att integralen ska utföras över ett väldigt stort område. Hur ska man göra? Standardknepet är att utföra en variabelsubstitution så att området får ändlig längd. En klassisk sådan är

$$x = \tan(t). \tag{8}$$

Den är dock inte så bra för numeriska beräkningar då man måste beräkna massa trigonemetriska funktioner. En bättre variant kan vara att sätta $x = t/(1-t)$ för positiva x och $x = t/(t-1)$ för negativa.

C Utför variabelsubstitutionen (8) och beräkna integralen (7) för $\mu = 0$, $\sigma = 1$ och $b = 1/2$ med den snabbaste av dina egna kvadraturprogram samt lämplig färdig Matlab-funktion. Plotta gärna den nya integranden och jämför med graferna från uppgift B. Jämför också med resultatet som Matlabs `normcdf` (NORMal Cumulative Distribution Function) ger.

1.3.2. *Att häva singulariteter.* Om man genom ett variablebyte kan få bort en singularitet innan man utför den numeriska beräkningen så är det bra. Ett annat knep framgår av följande exempel.

Antag att man vill beräkna $\int_0^{\pi/2} 1/\sqrt{sin(x)}\,dx$. Vi kan få bort singulariteten genom att skriva

$$\int_0^{\pi/2} \frac{1}{\sqrt{sin(x)}}\,dx = \int_0^{\pi/2} \frac{1}{\sqrt{sin(x)}} - \frac{1}{\sqrt{x}}\,dx + \int_0^{\pi/2} \frac{1}{\sqrt{x}}\,dx$$

där den sista integralen är analytiskt lösbar och den första går mot noll då x går mot noll (varför?).

2. Tips om numeriska experiment

I den här laborationen så ska ni vid flera tillfällen utföra numeriska experiment. Så här kan det se ut. Antag att vi vill testa hur lång tid det tar att summera respektive räkna ut produkten av N slumpade tal. Man kan tänka sig att det tar olika lång tid beroende på vilka tal vi har och omdatorn jobbar med något annat etcetera. Därför gör vi det många gånger och tittar sedan på till exempel medelvärdet.

```
M=1000; %Antal experiment
N=10000; %Antal slumptal
Summa=zeros(1,M); %Vektor för att spara de olika summorna.
Produkt=Summa;    % " " ... de olika produkterna.
SumTid=Summa;     % Vektorer för att spara motsvarande tider i
ProdTid=Summa;

for k=1:M
   z=randn(1,N); %Vi slumpar ut N gaussiskt fördelade slumptal
   tic %Vi börjar en tidtagning
   Summa(k)=sum(z); %Vi summerar talen
   SumTid(k)=toc;   % Vi mäter tiden och sparar på plats k i SumTid
   tic              % Vi nollställer klockan
   Produkt(k)=prod(z);  %Vi beräknar produkten av talen
   ProdTid(k)=toc;      % Vi mäter tiden
end
medSumTid=mean(SumTid);   %Räknar ut medelvärdet av tidsåtgången
medProdTid=mean(ProdTid); % "   " ...
figure(1),clf %Plottar alla tiderna samt medlvärdena som lodräta sträck:
plot(1:M,SumTid,'*b',1:M,ProdTid,'ok',...
    [1,M],medSumTid*[1,1],'b',[1,M],medProdTid*[1,1],'k')
```

3. Quads hjälptext med kommentarer

Nedna följer hjälptexten för funktionen `quad` med kommentarer.

```
 QUAD   Numerically evaluate integral, adaptive Simpson quadrature.
    Q = QUAD(FUN,A,B) tries to approximate the integral of scalar-valued
    function FUN from A to B to within an error of 1.e-6 using recursive
    adaptive Simpson quadrature. FUN is a function handle. The function
    Y=FUN(X) should accept a vector argument X and return a vector result
    Y, the integrand evaluated at each element of X.
```

Ovan är det absolut enklaste exemplet. Notera den förbestämda feltoleransen och att FUN måste vara vektoriserad. Nedan framgår hur man ändrar toleransen TOL. Avsnittet med TRACE intresserar oss inte nu.

```
    Q = QUAD(FUN,A,B,TOL) uses an absolute error tolerance of TOL
    instead of the default, which is 1.e-6.  Larger values of TOL
    result in fewer function evaluations and faster computation,
    but less accurate results.  The QUAD function in MATLAB 5.3 used
    a less reliable algorithm and a default tolerance of 1.e-3.

    Q = QUAD(FUN,A,B,TOL,TRACE) with non-zero TRACE shows the values
    of [fcnt a b-a Q] during the recursion. Use [] as a placeholder to
    obtain the default value of TOL.
```

Det går att säga åt `quad` att hålla räkningen på hur många gånger FUN anropas. Det är bara att ange ett extra utargument som framgår nedan.

```
[Q,FCNT] = QUAD(...) returns the number of function evaluations.

Use array operators .*, ./ and .^ in the definition of FUN
so that it can be evaluated with a vector argument.
```

Under rubriken *Notes* så jämförs `quad` med några av Matlabs andra funktioner för numerisk kvadratur. Det är en mycket viktig ruta.

```
Notes:
QUAD may be most efficient for low accuracies with nonsmooth
integrands.
QUADL may be more efficient than QUAD at higher accuracies
with smooth integrands.
QUADGK may be most efficient for oscillatory integrands and any smooth
integrand at high accuracies. It supports infinite intervals and can
handle moderate singularities at the endpoints. It also supports
contour integration along piecewise linear paths.
QUADV vectorizes QUAD for array-valued FUN.
```

Första exemplet är väl inte några större problem, men notera hur man gör om man vill integrera en funktion som har deklarerats med flera inparametrar som vi vill hålla konstanta under integrationen i det andra exemplet.

```
Example:
   Q = quad(@myfun,0,2);
where myfun.m is the M-file function:
   %------------------%
   function y = myfun(x)
   y = 1./(x.^3-2*x-5);
   %------------------%

or, use a parameter for the constant:
   Q = quad(@(x)myfun2(x,5),0,2);
where myfun2 is the M-file function:
   %----------------------%
   function y = myfun2(x,c)
   y = 1./(x.^3-2*x-c);
   %----------------------%

Class support for inputs A, B, and the output of FUN:
   float: double, single

See also quadv, quadl, quadgk, quad2d, dblquad, triplequad,
                               trapz, function_handle
```

Referenser

[Adams] Robert A. Adams, *Calculus - A Complete Course*, Sjunde upplagan, Pearson, New York.