Wetenschappelijk Rekenen

Examen - Derde bachelor informatica

Oefeningen – 27 augustus 2012

1. Bepaal de 6-punts Newton-Cotes formule $N_6(f)$ van het open type ter benadering van

$$I(f) = \int_{a-h}^{a+h} f(t)dt$$

met als stapgrootte $\frac{2h}{7}$. Bepaal eveneens de macht m en de constante C in de uitdrukking

$$I(f) - N_6(f) = h^{m+1} C f^{(m)}(\xi)$$

met $\xi \in [a-h, a+h]$.

Oplossing: Aangezien we interesse hebben in de 6-punts Newton-Cotes formule van het open type, weten we dat we de eindpunten $a-h$ en $a+h$ niet gebruiken en dat we de punten binnen dit interval equidistant kiezen. Bovendien weten we uit de opgave dat de stapgrootte $\frac{2h}{7}$ is. We bekomen dus dat we 6 punten in het interval $[a-h, a+h]$ beschouwen, namelijk de punten $a-h+1(\frac{2h}{7})$, $a-h+2(\frac{2h}{7})$, $\ldots a-h+6(\frac{2h}{7})$.

Met Maple kunnen we eenvoudig de kwadratuurformule $N_6(f)$ bepalen door eerst de polynomiale interpolant door deze punten te bepalen en die vervolgens te integreren:

```
restart:
with(CurveFitting):
p:=unapply(PolynomialInterpolation([seq(subs(y=(a-h)+c*(2*h/7),[y,f(y)]),c=1..6)], t),t);
N6:=simplify(integrate(p(t),t=a-h..a+h));
```

We krijgen dan als resultaat dat

$$N_6(f) = \frac{2h}{1440}\left(611f\left(a-\left(\frac{5}{7}\right)h\right) - 453f\left(a-\left(\frac{3}{7}\right)h\right) + 562f\left(a-\left(\frac{1}{7}\right)h\right) \right.$$
$$\left. +562f\left(a+\left(\frac{1}{7}\right)h\right) - 453f\left(a+\left(\frac{3}{7}\right)h\right) + 611f\left(a+\left(\frac{5}{7}\right)h\right)\right)$$

We weten dat de interpolatie zeker exact is voor veeltermen van graad hoogstens 5, dus verwachten we pas fouten vanaf veeltermen van minstens de 6^{de} graad. Daarom proberen we nu de integraal van de veelterm t^6 te benaderen met de kwadratuurformule door middel van de code:

```
f:=t->t^6;
If:=int(f(t),t=a-h..a+h);
N6:=simplify(integrate(p(t),t=a-h..a+h));
simplify(If-N6);
```

waaruit we de output krijgen dat

$$\frac{24032}{352947}h^7$$

waaruit volgt dat $m = 6$ en $C = \dfrac{24032}{352947 \cdot 6!} = \dfrac{1502}{15883065}$.

2. Een student heeft de volgende code geschreven:

```
function [waarde,vector] = examen1112_02(matrix)
    [rows, cols] = size(matrix);
    if rows ~= cols
        disp('geen n x n matrix');
        return;
    end
    precisie = eps;
    oud = rand(rows,1);
    stap = 0;
    stoppen = false;
    verhouding = 0;
    while (stoppen == false)
       stap = stap + 1;
       nieuw = matrix * oud;
       huidig = max(abs(nieuw)) / max(abs(oud));
       if (verhouding - huidig) < precisie
            stoppen = true;
       end
       verhouding = huidig;
       oud = nieuw;
    end
    h = matrix * oud ./ oud;
    % geef de dominante resultaten terug
    waarde = sign(h(1));
    vector = oud;
end
```

De student merkt echter op dat zijn resultaten niet overeen komen met de dominante oplossing die je, samen met de andere resultaten, kan terugvinden met het commando `eig(matrix)`. Na enig zoeken blijkt dat de student twee (2) fouten in zijn code heeft gemaakt en dat de student een inzichtsfout maakt bij het interpreteren van de resultaten.

Corrigeer om te beginnen de code en verklaar de inzichtsfout die de student maakt, namelijk waarom `vector` anders is bij het controleren van zijn werk met behulp van de matrix `A = [3 1 ; 1 3]`. Denk vervolgens na over twee manieren waarop je de hierboven vermelde code kan uitbreiden om deze beter te maken (een manier om minder fouten te maken en een manier om in ditzelfde algoritme met minder iteraties tot de oplossing te komen). Implementeer beide manieren op basis van de hierboven vermelde code en bewaar je code als `functie_beter.m` . Bewaar ook de gecorrigeerde code in `examen1112_02.m` !

Oplossing: De eerste fout die de student maakt, is dat er bij het bepalen van de bekomen precisie geen rekening wordt gehouden met de absolute waarde. Daardoor zal elke `(verhouding - huidig)` die een negatief getal oplevert ertoe leiden dat we stoppen. Verder wordt de eigenwaarde steeds nauwkeuriger berekend en bewaard in de variabele `verhouding`. Wanneer je op het einde het resultaat terug geeft aan de gebruiker moet je dan ook niet enkel rekening houden met het teken van de eigenwaarde, maar uiteraard ook met de waarde van de eigenwaarde! Zo krijgen we de twee correcties:

```
if abs(verhouding - huidig) < precisie
```

en

```
waarde = sign(h(1)) * verhouding;
```

We kunnen dit algoritme op twee manieren verbeteren. Om overflow/underflow te voorkomen, kunnen we onze resultaten normaliseren. De genormaliseerde versie van dit algoritme wordt onder andere beschreven in algoritme 4.2 in het boek op pagina 175. We vervangen

```
nieuw = matrix * oud;
huidig = max(abs(nieuw)) / max(abs(oud));
```

door de nieuwe code

```
huidig_zonder_norm = matrix * oud;
huidig = norm(huidig_zonder_norm,inf);
nieuw = huidig_zonder_norm/huidig;
```

Daarnaast bepalen we nu op een vrij domme manier, namelijk gewoon door een willekeurge initiële gok, de eigenwaarde die overeenkomt met de eigenvector. Een betere keuze is om te werken met het Rayleigh quotiënt, wat een betere benadering geeft voor de eigenwaarde en waardoor we met minder iteraties tot een oplossing kunnen komen. We vervangen

```
huidig = norm(huidig_zonder_norm,inf);
```

door de nieuwe code

```
huidig = oud'*matrix*oud/(oud'*oud);
```

Kijken we naar de teller `stap`, dan zien we dat we met deze aanpassing er inderdaad in slagen om de dominante eigenvector en bijbehorende eigenwaarde te berekenen met ongeveer de helft van de iteraties voor de matrix `A = [3 1; 1 3]` . De volledige gecorrigeerde en geoptimaliseerde code is:

```
function [waarde,vector] = examen1112_02_best(matrix)
    [rows, cols] = size(matrix);
    if rows ~= cols
        disp('geen n x n matrix');
        return;
    end
    precisie = eps;
    oud = rand(rows,1);
    stap = 0;
    stoppen = false;
    verhouding = 0;
    while (stoppen == false)
       stap = stap + 1;
       huidig_zonder_norm = matrix * oud;
       huidig = oud'*matrix*oud/(oud'*oud);
       nieuw = huidig_zonder_norm/huidig;
       if abs(verhouding - huidig) < precisie
            stoppen = true;
       end
       verhouding = huidig;
       oud = nieuw;
    end
    h = matrix * oud ./ oud;
    % geef de dominante resultaten terug
    waarde = sign(h(1)) * verhouding;
    vector = oud;
end
```

3. In de oefeningen hebben we een aantal implementaties bekeken voor het oplossen van stelsels van ODEs. Zo hebben we in de bestanden `voorbeeld1.m` en `methodeEuler.m` de Eulermethode besproken en in de bestanden `voorbeeld2.m` en `methodeAchterwaartseEuler.m` de Achterwaartse-Euler-methode.

   Eveneens in de oefeningen hebben we het gehad over Runge-Kutta methodes, die een volledige familie beschrijven van methodes voor het oplossen van stelsels van ODEs. Eén

van deze methodes is gedefinieerd als

$$k_1 = f(t_n, y_n) \qquad k_2 = f(t_n + \frac{2}{3}h, y_n + \frac{2}{3}hk_1) \qquad y_{n+1} - y_n = h\left(\frac{1}{4}k_1 + \frac{3}{4}k_2\right)$$

Maak een nieuw bestand `voorbeeld_vraag3.m` en `methodeRungeKutta.m` aan (uiteraard gebaseerd op de hiervoor vermelde bestanden!) waarin je deze methode implementeert. Denk goed na over wat je zoal moet aanpassen, en leg uit waarom je bij de implementatie begint van oftewel `methodeEuler.m` oftewel `methodeAchterwaartseEuler.m` .

Werkt deze methode beter of slechter voor het in `voorbeeld1.m` en `voorbeeld2.m` beschreven probleem? Maak hiervoor duidelijke plots met stapgrootte $h = \frac{1}{2}$ en $h = \frac{1}{16}$ en vermeld hoe je kan zien welke methode beter is. Kan je dit theoretisch verklaren?

Oplossing: We kunnen eenvoudig controleren dat de opgegeven Runge Kutta methode een expliciete methode is. We starten dus best vanaf de code gegeven in `methodeEuler.m`. We krijgen de `methodeRungeKutta.m` met de volgende code:

```
function [x,y] = methodeRungeKutta(rhs,h,xspan,y0);
x=[xspan(1)];
y=y0;
x0=x(1);
y0=y(:,1);
xend=xspan(2);
while x0 < xend-h/2
   y1=rungeKutta(rhs,x0,y0,h);
   y=[y,y1];
      y0=y1;
   x0=x0+h;
   x=[x,x0];
end;

function out=rungeKutta(rhs,x,y,h);
k1 = feval(rhs, x, y);
k2 = feval(rhs, x + (2/3)*h, y + (2/3) * h * k1);
out = y+h*((1/4)*k1 + (3/4)*k2);
```

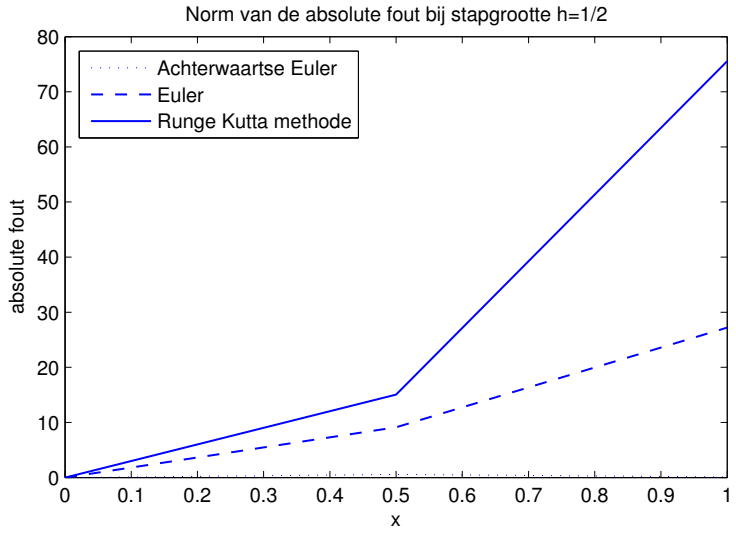
waarbij de grootste verandering – alsook de enige fundamentele code die je zelf hebt moeten aanpassen – de functie `rungeKutta(...)` is. Verder moeten we in `voorbeeld_vraag3.m` één regel aanpassen, namelijk:

```
[x,y] = methodeRungeKutta(@probleem23,h,xspan,y0);
```

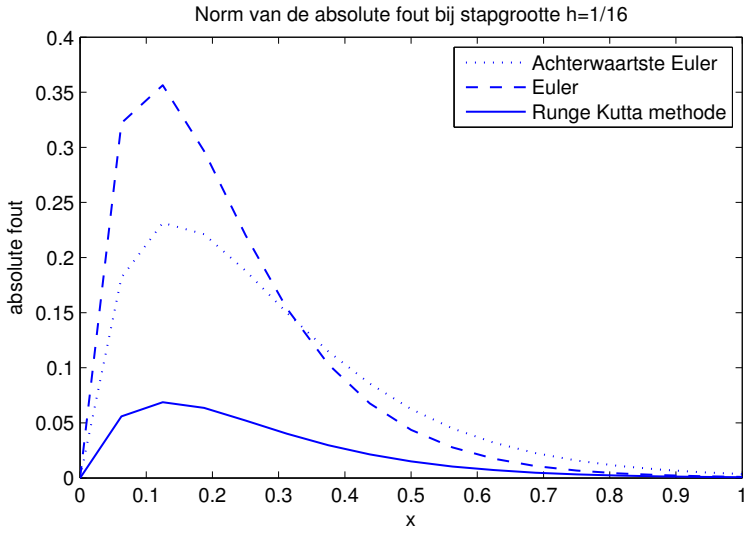
Deze code plot voor ons automatisch belangrijke info, waaronder de norm van de absolute fout. Op basis van deze informatie kunnen we goed nagaan welke methode nu beter of slechter is. Als we gebruik maken van de ODE zoals beschreven in de `voorbeeld1.m`, namelijk de ODE:

$$\begin{pmatrix} y_1' \\ y_2' \end{pmatrix} = \begin{pmatrix} y_2 \\ y_2 \cdot \begin{pmatrix} \frac{y_2 - 1}{y_1} \end{pmatrix} \end{pmatrix} \text{ met } \begin{pmatrix} y_1(0) \\ y_2(0) \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \\ -3 \end{pmatrix}$$

in het interval $[0, 1]$, dan vinden we voor een kleine stapgrootte, namelijk $h = \frac{1}{2}$, dat de Runge Kutta methode die we net geïmplementeerd hebben, slechter presteert dan zowel de Euler als de achterwaartste Euler methode.



Hieruit kunnen we echter niet besluiten dat de Runge Kutta methode slecht is. Integendeel, wanneer we de stapgrootte verkleinen, zien we heel snel dat de Runge Kutta methode veel betere prestaties neerzet. Nemen we bijvoorbeeld een stapgrootte van $h = \frac{1}{16}$, dan zien we dat de Runge Kutta methode beter is dan de (achterwaartse) Euler methode.



De reden waarom deze Runge-Kutta methode beter werkt, is omdat deze methode tweede-orde nauwkeurig is, terwijl de (achterwaartse) Euler methode slechts eerste-orde nauwkeurig is. Uiteraard, omdat deze Runge Kutta methode een expliciete methode is, zal deze Runge Kutta methode niet goed werken bij stijve ODE problemen, terwijl de achterwaartse Euler methode wél goed zal presteren bij dergelijke problemen.

4. De Curiosity rover, die momenteel op Mars rondrijdt, heeft de afgelopen dagen elke middag en elke nacht de temperatuur op Mars gemeten in graden Celsius:

dag	-21	-18	-19	-20	-24
nacht	-41	-43	-47	-42	-51

Welke dag- en nachttemperatuur mogen we de volgende dag verwachten? Beschrijf en probeer minstens drie methodes om tot een goed antwoord te komen. Vermeld ook waarom elke methode een goede methode is en/of waarom je verwacht dat de methode tot betere resultaten zal leiden dan de vorige methode die je gebruikt hebt.

Als we op het einde van de volgende dag te weten komen dat de dagtempereratuur $-22°C$ en de nachttemperatuur $-47°C$ was, welke methode was dan het beste en had je verwacht dat deze methode het beste was? Waarom?

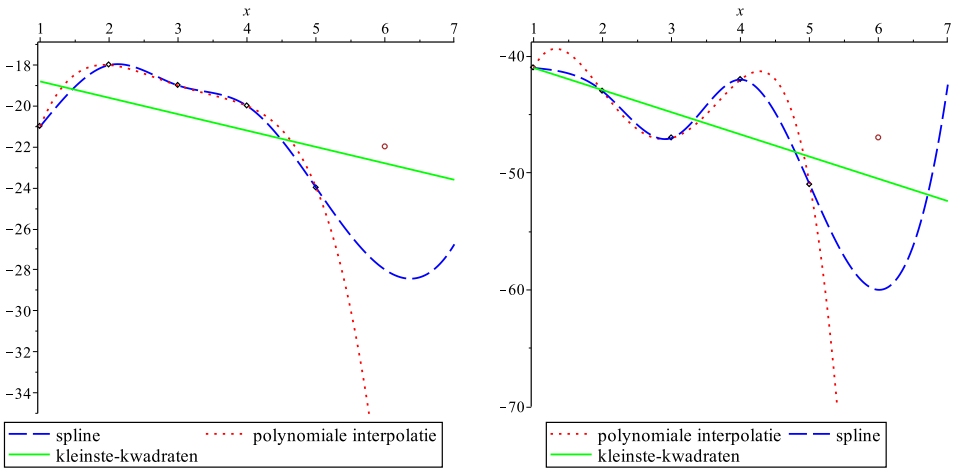
Oplossing: Deze oefening kan je zowel in MATLAB als in Maple oplossen, en beide manieren (of een combinatie van beide manieren) zijn natuurlijk even goed. Hieronder vind je de code waarmee je de oplossing in Maple kan uitwerken. Om te beginnen moeten we al opmerken dat de beste methode bepaald zal worden door de manier waarom je de data interpreteert. Zo kan je de data van de dag en de nacht als twee afzonderlijke verzamelingen beschouwen, of je kan via interpolatie proberen om alle datapunten in één keer te interpoleren. De meest betrouwbare resultaten mag je logischerwijs verwachten als je de dagen en de nachten afzonderlijk bekijkt.

Willen we weten welke temperatuur het de volgende dag/nacht wordt, dan kunnen we gebruik maken van de kleinste-kwadratenmethode, wat ons een lijn oplevert tussen de datapunten door (we kunnen uiteraard ook een hogere graad kiezen), of we kunnen gebruik maken van interpolatie. In het laatste geval kunnen we een onderscheid maken tussen polynomiale interpolatie, waarbij we één functie gebruiken om door alle waardes te interpoleren, het gebruik van splines en/of het gebruik van een cubic Hermite. Wij kiezen ervoor om via de kleinste-kwadratenmethode een rechte te bepalen die zo nauwkeurig mogelijk de punten benadert, een interpolatie met één veelterm en een interpolatie met behulp van splines. Met de volgende Maple code bekom je de resultaten:

```
reset:
with(CurveFitting):
with(plots):
points := [[1,-21],[2,-18],[3,-19],[4,-20],[5,-24]]:
LeastSquares(points, x):
p1 := plot(%,x=1..7, color = green, linestyle=1):
PolynomialInterpolation(points, x):
p2 := plot(%, x=1..7, color = red, linestyle=2):
Spline(points, x):
p3 := plot(%, x=1..7, color = blue, linestyle=3):
p4 := plot(points,style=point, color=black):
p5 := plot([[6,-47]], style=point, color=brown, symbol=circle):
display({p1, p2, p3, p4,p5});
```

Met deze code stel je achtereenvolgens de vergelijkingen op voor de kleinste-kwadratenmethode, polyomiale interpolatie en spline interpolatie en plot je vervolgens de punten en de vergelijkingen op één grafiek. Je krijgt:

Uit deze resultaten blijkt dat, zoals verwacht, de polynomiale interpolatie niet goed is. Dit komt omdat de graad van de functie snel te groot wordt en de functie dan ook wild begint te schommelen buiten het bereik $[1, 5]$. De spline interpolatie doet het een stuk



(a) dag (b) nacht

Figuur 1: plots met resultaten

beter. Hoewel we met de spline de temperatuur van de volgende dag niet goed kunnen bepalen, zien we wel dat onze gok vermoedelijk beter zal zijn voor de daaropvolgende dag. Helaas zal ook de spline interpolatie daarna sterk beginnen schommelen. De beste benadering voor de temperatuur van de volgende dag lijkt de kleinste-kwadratenmethode te zijn. Dit is een resultaat dat we konden verwachten omdat de datapunten onderling weinig schommelen en er slechts een kleine band bestaat tussen de hoogste en de laagste dag/nacht temperatuur tussen dewelke we naar alle waarschijnlijkheid de temperatuur van de volgende dag zullen vinden. Omdat de kleinste-kwadratenmethode net resulteert in een lijn die door deze band loopt, hadden we inderdaad alle redenen om te verwachten dat deze methode de beste ging zijn.

Schrijf jouw oplossingen neer op papier. Plaats de bestanden die je eventueel gemaakt hebt om de vragen op te lossen op Minerva. Dit doe je door

- te surfen naar http://indianio/
- in te loggen met jouw Minerva-gebruikersnaam en -wachtwoord
- jouw bestanden te zippen tot 1 enkel bestand en up te loaden.

Zorg ervoor dat ik uit de naamgeving kan afleiden bij welke vraag elk bestand hoort.

Nog enkele tips :

- schrijf proper : het is de beste manier om te slijmen.
- probeer je antwoorden bondig en correct te formuleren.
- heb je bij Maple en/of MATLAB moeite met (de syntax van) bepaalde commando's, aarzel dan niet hulp te vragen.