Informatica
Numerieke wiskunde
Prof. Koen Van Den Abeele
Andries Vansweevelt

Verslag project numerieke wiskunde

Academiejaar 2017-2018

Thomas Bamelis en Michiel Provoost

Inleiding

Dit verslag behandeld de vragen en antwoorden gesteld in het eind-project numerieke wiskunde 2017. De hoofdvraag in dit project is hoe wortels/nulpunten van een willekeurige veelterm kunnen gevonden worden met de methode van Newton-Raphson en de methode van Bairstow. Verder behandelt dit verslag de performantie en fouten op beide algoritmen en sluit af met deze met elkaar te vergelijken. Beide methoden werden geïmplementeerd in matlab en getest op 10 gegeven veeltermen, met complexe en reële nulpunten. Verder werden verschillende opgedragen ease-of-use features geïmplementeerd, zoals een figuur plotten indien enkel de coëfficiënten gegeven.
Ook word kort aandacht besteed aan een analyse van een bepaald geval, waarin bekeken wordt welke startwaarden voor beide methoden tot welke (of geen) nulpunten leiden.

1 Newton-Raphson

Dit hoofdstuk bespreekt de implementatie van Newton-Raphson, waarin onder andere de methode van Horner, samen met een analyse van de performantie en fouten.

1.1 Evaluatie veelterm en zijn afgeleide m.b.v. het Hornerschema

Hier wordt opdracht 1 besproken.

Met het uitgebreide Hornerschema, besproken in [1], kan een veelterm geëvalueerd in een gegeven punt. Daarna kan met een deel van de oplossing verder gewerkt worden om 1-per-1 de afgeleiden van de veelterm te evalueren in hetzelfde punt.

1.1.1 Implementatie

Dit werd als volgt geïmplementeerd:

```
1  function y = my_polyval( p, x, m)
2  %my_polyval Geeft de functiewaarde in c van de eerste m afgeleiden van p
3  %terug, via het uitgebreid schema van Horner.
4  %
5  %   Signatuur: y = my_polyval( p, x, m)
6  %
7  %   @param p
8  %       De veelterm die geevalueerd zal worden, voorgesteld door zijn
9  %       coefficienten als vector, met de hoogste graad term als eerste
10 %       element enzovoort. Dit is een 1 x n vector.
11 %
12 %   @param x
13 %       Het punt waarin de functies zullen geevalueerd worden.
14 %
15 %   @param m
16 %       Het aantal keer dat p moet afgeleid worden.
17 %
18 %   @return y
19 %       De vector die de functiewaarden in x van de 0'de tot de m'de
```

```
20 %        afgeleide van p bevat.
21 %
22
23
24
25
26 graadStartPlusEen = size( p );
27 graadStartPlusEen = graadStartPlusEen(2);
28
29
30 y = zeros( m + 1, 1);
31
32 % p gebruiken, want minder geheugen gebruikt (call by value)
33
34
35 for afgeleide = 1:m+1
36
37     % 1ste niet kopieren want die zit al in p.
38
39     for coefficient = 2:(graadStartPlusEen - afgeleide+1)
40
41         %De coefficienten aanpassen. Alle ellementen van p voor
42         %p(coefficient) zijn al aangepast, dus de 2de term
43         %in onderstaande formule ook.
44
45         p(coefficient) = p(coefficient) + p(coefficient - 1)*x;
46
47     end
48
49     %De functiewaarde in x van deze afgeleide berekenen en opslaan.
50     %y begint bij 1 en eindigt bij m + 1, dus 1 verder zetten.
51
52     y(afgeleide) = factorial(afgeleide-1) * p(graadStartPlusEen - afgeleide
           +1);
53
54
55
56 end
57 end
```

Zoals in [1] wordt aangetoond, worden de m eerste afgeleid (inclusief $f^{(0)}$) gevonden, door m keer het schema van Horner toe te passen op p waarna de functiewaarde van i'de afgeleide gegeven wordt door $k!b_{n-k}^{k+1}$. Merk op dat in de 'afgeleide' iteratie, de 'afgeleide' - 1 'ste afgeleide behandeld wordt om beter matlabs vector nummering in te kunnen spelen.
Het 1'ste element van p, dus de coefficient van de hoogste graad term, wordt nooit aangepast omdat volgens Horner $a_0 = b_0$. Daarna worden de rest van de coefficienten aangepast volgens $b_i = a_i + b_{i-1} * x$. Er wordt rekening gehouden met het feit dat de graad van de veelterm bij iedere iteratie van de afgeleide met 1 verlaagd wordt.

Een bijkomende opdracht was om zo weinig mogelijk geheugen te gebruiken. Om hier mee om te gaan werd in de vector p zelf gewerkt. Dit is geen enkel probleem omdat je nooit de oude vorige elementen nodig hebt en omdat matlab call-by-value is. Dit wil zeggen dat matlab p kopieert en de kopie meegeeft aan de functie waar mee gewerkt mag worden zonder dat de originele p wordt aangepast. Het enige

geheugen dat extra wordt aangemaakt, is het geheugen gebruikt om de functie waarden terug te geven een de oproeper van de functie. Dit wil zeggen dat het geheugengebruik van de functie van orde $\Theta(n+m)$ is, met n de graad + 1 van p en m het aantal keer dat afgeleid moet worden.

1.1.2 Fouten

De stabiliteit van deze methode komt op 2 manieren in gedrang:

- In lijn 45, als p(coefficient) en p(coefficient - 1)*x bijna even groot zijn en tegengesteld teken hebben.
- In lijn 52, als de 'afgeleide' afgeleide zeer groot word, zullen afrondingsfouten groter worden door de faculteit.

1.2 Een nulpunt vinden met Newton-Raphson

Hier wordt opdracht 2 besproken.

Indien een willekeurig veelterm gegeven wordt en een startwaarde, kan via de formule van Newton-Raphson ($x_{i+1} = x_i + \frac{p(x_i)}{p'(x_i)}$) een nieuwe x gevonden worden, die dichter bij het nulpunt ligt als er convergentie optreedt.

1.2.1 Implementatie

```
1  function w = newtonraphson( p, start, tol)
2  %newtonraphson Geeft het nulpunt van p terug die gevonden word via de
3  %startwaarde start met fouten tolerantie tol (standaard 10^-6),
4  %via de methode van Newton-Raphson.
5  %
6  %   Signatuur: w = newtonraphson( p, start, tol)
7  %
8  %   @param p
9  %       De veelterm die geevalueerd zal worden, voorgesteld door zijn
10 %       coefficienten als vector, met de hoogste graad term als eerste
11 %       element enzovoort. Dit is een 1 x n vector.
12 %
13 %   @param start
14 %       De startwaarde nodig om Newton-Raphson aan te vatten.
15 %
16 %   @param tol
17 %       De tolerantie van de fout, indien niet meegegeven 10^-6.
18 %
19 %   @return w
20 %       Het nulpunt gevonden aan de hand van de starwaarde binnen de
21 %       tolerantie tol indien de fouttolerantie van de gevonden waarden
22 %       kleiner of gelijk is aan tol, anders NaN.
23 %
24 %
25 %
26
27
28 % p mag geen 0de graads zijn.
29 s = size(p);
30 s = s(2);
31 if s <= 1
32     disp("Functie moet minstens graad 1 zijn!");
33     w = NaN;
```

```
34      return
35  end
36
37  %Check of tol gegeven is.
38
39  if nargin == 2
40      tol = 10^(-6);
41  end
42
43  % Initialiseer de startwaarden.
44
45  vorigeX = start;
46  huidigeX = start;
47
48  % De fout benadering initialiseren zodat deze in de while geraakt.
49
50  foutAprox = tol + 1;
51
52  iterator = 0;
53  while foutAprox > tol && iterator < 10000
54
55      vorigeX = huidigeX;
56
57      % Bereken p(vorigeX) en p'(vorigeX).
58
59      [ px, afgpx] = my_polyval( p, vorigeX, 1);
60
61
62      % Stabiliteit beschermen tegen een kleine afgeleide.
63
64      if afgpx < 1
65
66          stabV = 100;
67
68      else
69
70          stabV = 1;
71
72      end
73
74
75      % px gedeeld door afgpx
76
77      deling = px / ( afgpx * stabV );
78      deling = deling * stabV;
79
80      % Bereken de huidigeX
81
82      huidigeX = vorigeX - deling;
83
84
85      foutAprox = abs(huidigeX - vorigeX);
86      iterator = iterator + 1;
87  end
88
```

```
89 % Checken of de fout benadering kleiner is dan tol.
90 if abs(huidigeX - vorigeX) > tol
91     w = NaN;
92     return
93 end
94
95 w = huidigeX;
96
97 end
```

Het iteratief berekenen van de volgende x, wordt gestopt op 2 manieren:

1. Door de 'zwakke' stopwaarde, als de nieuwe x en de vorige x in absolute waarde kleiner zijn dan de meegegeven tolerantie.
   In dit geval wordt de functie aanroep als succesvol gezien.

2. Na 10000 iteraties wordt de lus verplicht afgebroken, om niet in een oneindige lus terecht te komen.

Bij iedere iteratie wordt de 'zwakke' fout opnieuw berekent en gecheckt of aan de voorwaarde voldaan wordt. Op het einde van het algoritme wordt nog een laatste check gedaan om te zien of er voldaan is aan de meegegeven tolerantie. Indien wel wordt de gevonden benadering van het nulpunt meegegeven, anders NaN. Er werd nog een extra geïmplementeerd om de stabiliteit te bevorderen die in de volgende sectie besproken wordt.

1.2.2 Fouten

De grootste fout wordt verkregen door een kleine $p^{'}(x_i)$ in de berekening van een nieuwe x, door de deling in lijn 77. Om hiermee om te gaan wordt indien $p^{'}(x_i)$ kleiner dan 1 wordt, in de deling een $p^{'}(x_i)$ vermenigvuldigt wordt met 100 om het verderzetten van de fout op $p^{'}(x_i)$ te verzachten. Hierna wordt de volledige quotiënt opnieuw vermenigvuldigt met 100. De 100 kan groter genomen worden, maar dan worden de afrondingsfouten voor een grote $p(x)_i$ ook groter.

1.3 Deflatie met Horner

Hier wordt opdracht 3 besproken.

Nadat een nulpunt met Newton-Raphson gevonden wordt, kan de (x - gevondenNulpunt) factor weggedeeld worden met het alom bekende schema van Horner. Er moest ook voorzien worden dat als het gegeven nulpunt complex is, zijn complex toegevoegde ook weggedeeld wordt.

1.3.1 Implementatie

```
1  function q = horner( p, x)
2  %horner Voert een deflatie stap uit op p zodat p een graad lager word en
3  %door de factor met nulpunt x, en indien x complex is ook zijn complex
4  %toegvoegde, weg te delen via de methode van Horner.
5  %
6  %   Signatuur: q = horner( p, x)
7  %
8  %   @param p
9  %       De veelterm die de deflatie ondergaat, voorgesteld door zijn
10 %       coefficienten als vector, met de hoogste graad term als eerste
11 %       element enzovoort. Dit is een 1 x n vector.
12 %
13 %   @param x
14 %       Het nulpunt die zal weggedeeld worden in p.
```

```
15 %
16 %    @result q
17 %        De veelterm die het resultaat is van de deflatie van p, voorgesteld
18 %        door zijn coefficienten als vector, met de hoogste graad term als
19 %        eerste element enzovoort. Dit is een 1 x n vector.
20 %
21
22
23 % Initialiseer q.
24 grootteP = size(p);
25 grootteP = grootteP(2);
26 q = zeros(1, (grootteP - 1));
27
28 %Stop als p te klein is.
29
30 if grootteP <= 1
31    disp("Graad p is te laag!");
32    q = NaN;
33    return
34 end
35
36 if grootteP <= 2 && not( isreal(x) )
37    disp("Graad p is te laag voor een complex getal!");
38    q = NaN;
39    return
40 end
41
42 % Coefficienten van de hoogste graad termen van p en q zijn gelijk.
43
44 q(1) = p(1);
45
46 % Start en eindwaarde for loop elk 1 hoger voor gemak in matlab (rest niet
47 % nodig).
48 for coefficient = 2:(grootteP - 1)
49
50    q(coefficient) = p(coefficient) + q(coefficient - 1)*x;
51
52 end
53
54
55
56 % Complex toegevoegde ook wegdelen indien x complex is.
57 if not( isreal(x) )
58
59     x = conj( x );
60
61     % Kopieer q om verder te kunnen werken ermee.
62
63     a = q;
64
65     % q nog een graad lager initialiseren.
66
67     q = zeros( 1, grootteP - 2);
68
69     % Coefficienten van de hoogste graad termen van p en q zijn gelijk.
```

```
70
71      q(1) = a(1);
72
73      % Start en eindwaarde for loop elk 1 hoger voor gemak in matlab (rest
            niet
74      % nodig).;
75      for coefficient = 2:(grootteP - 2)
76
77        q(coefficient) = a(coefficient) + q(coefficient - 1)*x;
78
79      end
80
81  end
82
83
84  end
```

De implementatie is gelijk aan die van sectie 1.1 met 1 iteratie, waarbij de eerste n-1 coëfficiënten worden teruggegeven na de iteratie. Dit stelt dus een veelterm van 1 graad lager dan p voor. Dit geld echter enkel als het nulpunt reëel is. In dit geval word een veelterm die 2 graden lager is dan p teruggegeven, waaruit beide nulpunten op dezelfde manier zijn weggedeeld. Tijdens het wegdelen van het complex toegevoegde wordt dus met een input van 1 graad lager een output van 2 graden lager gegeven.

1.3.2 Fouten

De stabiliteit van deze methode lijd onder dezelfde problemen als 1.1:

- In lijn 50, als p(coefficient) en q(coefficient - 1)*x bijna even groot zijn en tegengesteld teken hebben.
- In lijn 77, als a(coefficient) en q(coefficient - 1)*x bijna even groot zijn en tegengesteld teken hebben.

1.4 Alle nulpunten vinden

Hier wordt opdracht 3 besproken.

Met de hiervoor besproken methoden werd zoals gevraagd een implementatie geschreven die alle nulpunten berekent van een gegeven veelterm. Deze methode vraagt om een veelterm, een startwaarde en een tolerantie die 10^{-6} is indien niet meegegeven. Er wordt een nulpunt bepaald met de methode van Newton-Raphson met de gegeven startwaarde, waarna dit nulpunt weggedeeld wordt met de methode van Horner. Er wordt met de nieuwe veelterm die door Horner werd teruggegeven verder gewerkt en er wordt opnieuw een nulpunt met Newton-Raphson bepaald met als startwaarde het vorige gevonden nulpunt. Dit wordt gedaan tot alle nulpunten van de veelterm gevonden worden.
Als Newton-Raphson niet convergeert, en dus NaN wordt teruggegeven, wordt de huidige functie geplot en om een nieuw startpunt gevraagd aan de gebruiker.
Uiteindelijk wordt een vector met alle nulpunten van p teruggegeven.

1.4.1 Implementatie

```
1  function ws = newtonraphsondef( p, start, tol)
2  %newtonraphsondef Geeft alle nulpunten van de veelterm p terug met fouten
3  %tolerantie tol (standaard 10^-6).
4  %
5  %   Signatuur: ws = newtonraphsondef( p, start, tol)
6  %
7  %   @param p
8  %       De veelterm die geevalueerd zal worden, voorgesteld door zijn
```

```
9  %       coefficienten als vector, met de hoogste graad term als eerste
10 %       element enzovoort. Dit is een 1 x n vector.
11 %
12 %   @param start
13 %       De startwaarde nodig om Newton-Raphson aan te vatten.
14 %
15 %   @param tol
16 %       De tolerantie van de fout, indien niet meegegeven 10^-6
17 %
18 %   @return ws
19 %       de nulpunten gevonden aan de hand van de starwaarde binnen de
20 %       tolerantie tol indien de fouttolerantie van de gevonden waarden
21 %       kleiner of gelijk is aan tol, anders NaN.
22 %
23
24
25 %ZORG DAT JE ALLES VIND, DOE DIT DOOR NIEUWE STARTWAARDEN TE DOEN INGEVEN
26 %INDIEN NIET ALLES WERD GEVONDEN, DOE DIT ZOALS AANGEGEVEN IN DE OPGAVE
27
28 %zeg dat ze desnoods een complexe startwaarde moeten geven,
29 %want enkel zo kun je complexe nulpunten vinden.
30
31
32 % p mag geen 0de graads zijn.
33
34 grootteP = size(p);
35 grootteP = grootteP(2);
36 if grootteP <= 1
37     disp("Functie moet minstens graad 1 zijn!");
38     ws = NaN;
39     return
40 end
41
42 % Check of tol gegeven is.
43
44 if nargin == 2
45     tol = 10^(-6);
46 end
47
48 % Initialiseer ws.
49
50 ws = zeros( 1, grootteP - 1);
51
52 % Maak de arrays aan om te plotten indien geen nulpunt gevonden word.
53 plotGrootte = 5000;
54 startPlot = -500;
55 eindPlot = 500;
56 X = zeros( plotGrootte, 1);
57 Y = zeros( plotGrootte, 1);
58 stapGrootte = (abs(startPlot) + abs(eindPlot)) / plotGrootte;
59
60 % Itereer aan de hand van het aantal gevonden nulpunten.
61
62 nbGevondenNulpunten = 0;
63
```

```
64  while nbGevondenNulpunten ~= grootteP - 1
65
66
67      huidigNulpunt = newtonraphson( p, start, tol);
68
69      if ~isnan(huidigNulpunt)
70          %Nulpunt gevonden binnen de tolerantie:
71          %Deel nulpunt weg en voeg nulpunt toe aan ws
72
73          nbGevondenNulpunten = nbGevondenNulpunten + 1 ;
74
75
76          p = horner( p, huidigNulpunt);
77
78
79          ws(nbGevondenNulpunten) = huidigNulpunt;
80
81          %Indien het nulpunt complex is werd zijn toegevoegde ook
82          %weggedeelt:
83
84          if not( isreal(huidigNulpunt) )
85              nbGevondenNulpunten = nbGevondenNulpunten + 1 ;
86              ws(nbGevondenNulpunten) = conj(huidigNulpunt);
87          end
88
89      else
90          % Geen nulpunt gevonden, laad nieuwe startwaarde
91
92          % Maak arrays om te plotten.
93
94          xWaarde = startPlot;
95
96          huidigeGrootteP = size(p);
97          huidigeGrootteP = huidigeGrootteP(2);
98
99          % Bereken alle koppels om te plotten.
100         for i = 1:plotGrootte
101
102             yWaarde = 0;
103             exponent = 0;
104
105             % Bereken de y waarde.
106             % Begin achteraan de lijst, dus met de constante.
107             for j = huidigeGrootteP:-1:1
108
109                 yWaarde = yWaarde + p(j) * xWaarde^(exponent);
110
111                 exponent = exponent + 1;
112
113             end
114
115             X(i) = xWaarde;
116             Y(i) = yWaarde;
117
118             xWaarde = xWaarde + stapGrootte;
```

```
119             end
120 
121             % Plot de functie.
122 
123             plot(X,Y);
124 
125             % Vraag om juiste input
126 
127             input_hf = str2double(input('Geen nulpunt gevonden, geef nieuwe (
                    misschien complexe) startwaarde: ','s'));
128             while isnan(input_hf) || fix(input_hf) ~= input_hf
129                 input_hf = str2double(input('Geef een getal in: ','s'));
130             end
131 
132             start = input_hf;
133 
134         end
135 
136 
137 end
138 end
```

1.4.2 Fouten

Omdat deze functie op zich niet veel rekenwerk bevat, maar dit wordt doorgegeven aan de opgeroepen functie, kan hier niet veel over de fout gepraat worden.
Bij het plotten van de functie zijn kleine fouten niet zo belangrijk, omdat dit op een grote schaal gebeurt en omdat een plot sowieso nooit als betrouwbaar mag gezien worden.

2 Bairstow

3 Convplot

Bekijk welke punten niet tot convergentie leiden. Zoek ook patronen, en plaatsen waar precies willekeurig een nulpunt optreed.

4 Newton-Raphson VS Bairstow

Volgens die mail van Andries convergeert Bairstow altijd, dus das een groot voordeel maar kijk nog eens verder

Besluit

Afsluitende tekst

Referenties

[1] Adhemar Bultheel. *Inleiding tot de numerieke wiskunde.* Acco, 2006.