Matlab examenvragen – oefening 5

Implementatie (vraag 1,2,3)

```
function [eval, optellingen, vermenigvuldigingen] = dubbelhorner(p,c,d)

% p(x) is de polynoom
% als je het algoritme zelf wilt testen, gebruik 'syms x' in je command
% window om aan te geven dat x een variabele is en typ dan gewoon 'p(x)
=
% ... ', anders werkt 'coeffs' niet.

% c+di is het complex getal waarin de functie geëvalueerd wordt.

a = coeffs(p,'All'); % coeffs geeft een uitdrukking (GEEN vector) terug
met de coëfficiënten bij oplopende graad, 'All' zorgt ervoor dat ook de
coëfficiënten '0' worden teruggegeven
a = a(1); % hier mag je een random getal invullen om een vector te
verkrijgen

n = length(a)-1; % -1 aangezien de graad van de constante term 0 is

b = zeros(1,n+2);
b(1) = 0;
b(2) = a(1);

rho = 2*c;
mu = -(c^2 + d^2);

optellingen = 1; % afkomstig van mu
vermenigvuldigingen = 3; % 1 van rho en 2 van mu (de kwadraten)

for k = 3:n+2
    b(k) = a(k-1) + rho*b(k-1) + mu*b(k-2); % a(k-1) ipv a(k) omdat de
'a_0' index 1 heeft
    optellingen = optellingen + 2;
    vermenigvuldigingen = vermenigvuldigingen + 2;
end

C = b(n+2) - c*b(n+1);
D = d*b(n+1);

optellingen = optellingen + 1;
vermenigvuldigingen = vermenigvuldigingen + 2;

eval = C + D*1i; % 1i ipv i, suggestie door matlab om het 'robuuster'
te maken

end
```

Vraag 4

```
>> syms x
>> p(x) = x^7+3*x^4+4*x-3;
>> p(2-i) % we evalueren de polynoom volgens het algoritme dat in
Matlab aanwezig is

ans =

- 294 - 47i

>> dubbelhorner(p,2,-1) % we gebruiken ons eigen algoritme

ans =

  -2.9400e+02 - 4.7000e+01i % we zien dat hetzelfde resultaat bekomen
wordt
```

Vraag 5

```
>> [~,optellingen,vermenigvuldigingen] = dubbelhorner(p,2,-1)

optellingen =

    16

vermenigvuldigingen =

    19
```

Het aantal optellingen bedraagt 16 (2*7+2) en het aantal vermenigvuldigingen 19 (2*7+5). In het algemeen zal het aantal bewerkingen gelijk zijn aan (2n+2)O + (2n+5)V. Dit wijkt lichtjes af van het theoretisch verwachte aantal bewerkingen (2n+2)O + (2n+4)V.

Enerzijds is dit omwille van het feit dat men in de for-loop bij k = 3 een optelling maakt met het getal '0'. In theorie zouden we deze optelling (en vermenigvuldiging met mu) niet doen, op die manier is er al 1 optelling en 1 vermenigvuldiging meer.

Anderzijds geven we het resultaat terug door C en D afzonderlijk te berekenen, dit vraagt 1O + 2V. In theorie voeren we hiervoor de volgende bewerking uit: $p(c) = b_{n-1} * (c + di – rho) + b_n$. Dit vraagt 2O + 2V. Op die manier hebben we in ons algoritme 1 optelling minder.

Uiteindelijk geven deze twee effecten samen aanleiding tot 1 extra vermenigvuldiging in het geïmplementeerde algoritme ten opzichte van de theoretsche waarden.