Praktische Foutenschatters d.m.v. Nulregels

Peter Coppens, Alexander Boucquey, Simon Dirckx, Cédric Picron

1 Trapezium regel

We gebruiken de samengestelde regel beschreven in (Numerical approximation of integrals). Om dan de trapezium regel te implementeren, nemen we $w_j = \{1, 1\}$, $x_j = \{-1, 1\}$.

Deze implementatie geeft de relatieve errors:
f_1 : 0.123355
f_2 : 0.032950
f_3 : 0.033711
f_4 : 0.033461.

De grootte van deze fouten kan intuitief verklaard worden aan de hand van figuur 1.

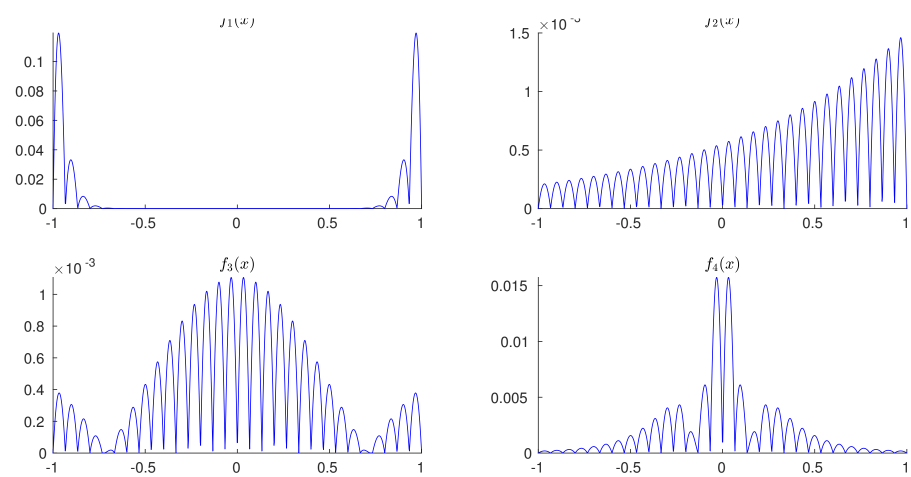

Figure 1: Error between interpolation by trapezium rule and the actual function.

2 Nulregels

We berekenen de nulregels a.d.h.v de momentvergelijking op p237. We gaan dus niet uit van symmetrische regels (aangezien dit niet gevraagd is in de opgave). Een overgang naar deze zou echter wel de nauwkeurigheid verhogen. We berekenen nulregelse gebaseerd op $n+1 = 31$ punten. Deze willen we even sterk maken als de trapezium regel. We doen dit aan de hand van de MATLAB code in Appendix B. De input is dan de vector berekend door de code in Appendix A.

De berekende errors e_j worden weergegeven in figuur 2. De errors E_j na het verwijderen van het fase-effect worden getoond in figuur 3.

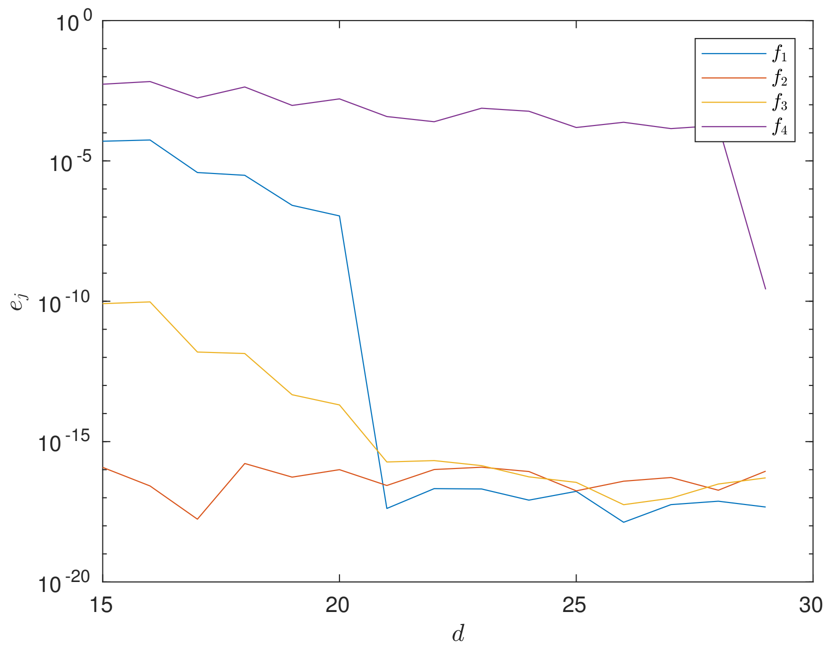

Figure 2: e_j in functie van de graad van de nulregel.

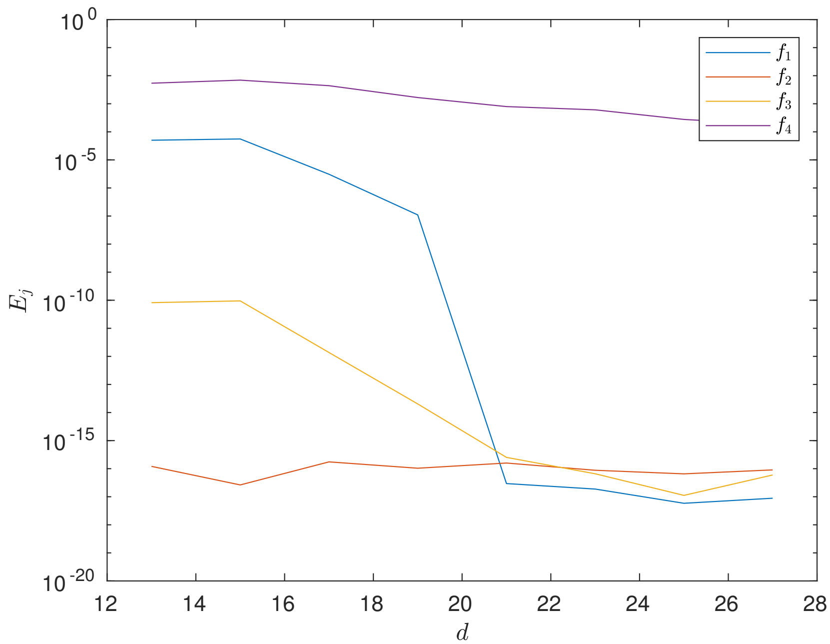

Figure 3: E_j in functie van de graad van de nulregel.

3 Reductie factoren

Bij het berekenen van de factoren zullen we de errors e_j, die rond machine precisie liggen, uitsluiten. Ze bestaan namelijk hoofdzakelijk uit ruis van de berekeningen. We beschouwen een error als ruis als: $e_j < 10^{-10}$, in dit geval kiezen we $r_j = 10^{-10}$ (in de rest van de berekeningen zouden deze dan niet gebruikt moeten worden, maar dit maakt de plots wel duidelijker).
Figuur 4 toont twee plots. De bovenste is van de reductie factoren r_j voor het verwijderen van de fase effecten. De onderste is die van R_j na het verwijderen van de fase effecten.

Merk op dat de lage waarden die afwisselend optreden bij e_j zijn uitgevlakt bij E_j. Dit wordt dan ook zichtbaar als men r_j en R_j vergelijkt.

We zien sterke convergentie bij f_1 en f_3. Ze zitten echter in het ruis gebied, waardoor dit niet te bevestigen valt aan de hand van de reductie factoren. f_1 gaat duidelijk naar het asymptotisch gebied, voordat hij eveneens in het ruis gebied terecht komt. f_4 blijft de hele tijd zwak asymptotisch.

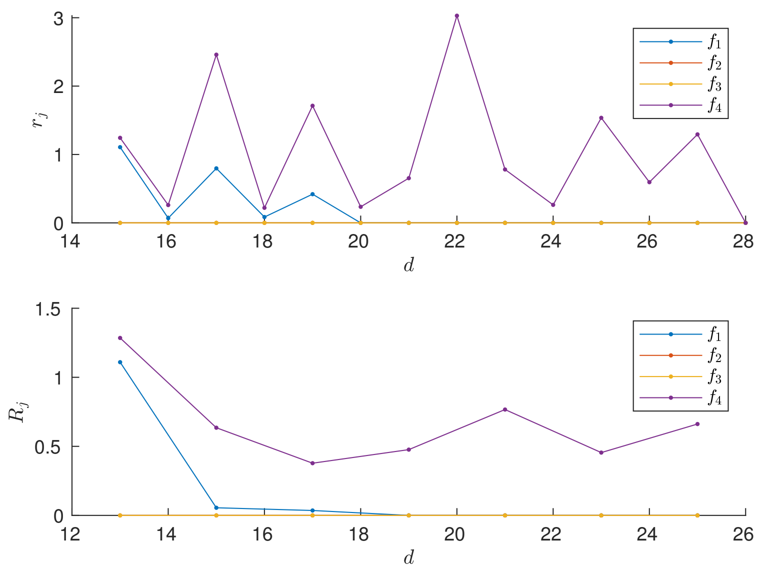

Figure 4: Reductie factoren, met en zonder fase effect.

4 Foutenschatting

We kunnen de nulregels, bepaald in de eerdere secties niet gebruiken om de fout op de samengestelde trapezium regel te bepalen. Dit is namelijk geen interpolerende regel van $n + 1 = 31$ punten. De theorie gaat ervan uit dat dit wel het geval is. Voor de samengestelde trapezium regel kunnen we maar één nulregel bepalen: $n = [-0.5, 0.5]$. Deze moet dan toegepast worden op elk deelinterval waar we de normale trapezium regel op toepassen om een lokale fouten schatting te krijgen. Dit doen we aan de hand van het eerste algoithm in Sectie 6 van de tekst over nulregels. Dit is uitgewerkt in MATLAB code in Appendix C. We bekomen de schatting:
$f_1 :$ 0.0333
$f_2 :$ 0.5614
$f_3 :$ 0.0211
$f_4 :$ 0.0314
De werkelijke fout is:
$f_1 :$ 0.0117
$f_2 :$ 0.0774
$f_3 :$ 0.0504
$f_4 :$ 0.0222
Merk op dat de schatting een redelijke bovengrens is voor f_1 en f_4, een extreme bovengrens voor f_2 en onder de fout zit voor f_3. Dit zijn geen geweldige resultaten.

Appendices

A Code om de samengestelde trapezium regel te berekenen.

```
function q = ctrap(k, f)
    h = 2/k;
    q = (h/2)*[1, 2*ones(1, k-2), 1];

    if exist('f','var')
        x = -1 + h*(0:k);
        xf = linspace(-1, 1, 1000);
        plot(xf, abs(f(xf)-interp1(x,f(x),xf)), 'b-');
    end
end
```

B Code om de nulregels te berekenen

```
function U = null_moment(q, K)
    % calculate orthogonal equally strong nullrules for
    % the quadrature rule defined by the vector of weights q.
    k = length(q)-1;
    x = -1 + (2/k)*(0:k);

    V = flipud(vander(x)');
    U = zeros(k+1, k);
    for m = 1:K
        % Calculate next nullrule
        V = V(1:end-1,:);
        NS = null(V);
        nv = zeros(size(NS(:, 1)));
        for l = 1:size(NS, 2)
            if norm(U(:, 1:m-1)*(U(:, 1:m-1)\NS(:, l)) - NS(:, l)) > 10^-3
                nv = NS(:, l);
                break;
            end
        end

        % Orthogonalize
        for i = 1:m-1
            r = sum(nv.*U(:,i));
            nv = nv - r*U(:,i);
        end

        % Make equally strong
        U(:,m) = norm(q)*nv/norm(nv);
    end
    U = U'; % null vectors are stored on the rows
end
```

C Code voor QAG algorithm

```matlab
%% Testfunctions
f1 = @(x) x.^20;
f2 = @(x) exp(x);
f3 = @(x) exp(-x.^2);
f4 = @(x) 1./(1+16*x.^2);
f = {f1, f2, f3, f4};

%% Error estimate
n = [-0.5, 0.5];
q = [0.5, 0.5];
xh = linspace(-1, 1, 31);
h = xh(2)-xh(1);
Et = zeros(4, 1);
for j = 1:4
    for i = 1:length(xh)-1
        fh = @(x) f{j}(0.5*(x*(xh(i)+xh(i+1)) + xh(i+1)-xh(i)));
        Ih = @(x) I{j}(0.5*(x*(xh(i)+xh(i+1)) + xh(i+1)-xh(i)));
        e1 = apply_rule(n, fh)*0.5*h;
        ft = apply_rule(q, fh);
        et = apply_rule(q, @(x) abs(fh(x) - ft))*0.5*h;
        r1 = abs(e1)/et;
        if r1 > 1/200
            Et(j) = Et(j) + et;
        else
            Et(j) = Et(j) + (200^(1.5))*(r1^(1/2))*abs(e1);
        end
    end
end

disp(Et);

function q = apply_rule(w, f)
    % apply quadrature rule or null rule to f where f is a function handle
    % and w is the row vector representation of the rule.
    k = length(w)-1;
    fx = f(-1 + (2/k)*(0:k));
    q = w*fx';
end
```