Numerieke Modellering & Benadering Practicum

Dennis Debree and Jonas Bertels

KU Leuven Departement Computerwetenschappen

12 april 2019

1 Bivariate Veelterm Benadering

1.1 Inleiding

Voor het vak numerieke modellering en benadering aan de KU Leuven werd de opdracht gegeven een bivariate veelterm benadering te implementeren in MATLAB. Deze benadering werd dan gebruikt om enkele 2 dimensionale oppervlakken te benaderen. Dit verslag geeft een visueel overzicht van de benaderingen en hun oppervlak en van de nauwkeurigheid van de benadering afhankelijk van de graad van de benaderende veelterm.

1.2 De basisfunctie

```
function C = kkb(x, y, F, m, n)
Afull = fliplr(vander(x));
Bfull = fliplr(vander(y));
C =  Bfull(:, 1:n)\F*pinv((Afull(:, 1:m)))';
end
```

Deze functie creëert eerst de A en B Vandermonde matrices om hieruit de x en y waarden te halen. Daarna wordt uit deze A en B matrices de F matrix berekent door eerst de graad van de A en B matrices te verkorten tot m en n en hierna C te bereken door de formule $C = B^{+}F(A^{+})^{T}$. Deze formule werd als volgt afgeleid:
We proberen het volgende te minimaliseren:

$$C = B^{+}F(A^{+})^{T} \tag{1}$$

Om dit te doen moet $\Gamma c = f$ of $\Gamma^{+}\vec{(}F) = C$ en aangezien $\Gamma = A \otimes B$ geldt:

$$\text{vec}(C) = (A \otimes B)^{+}\text{vec}(F) = (A^{+} \otimes B^{+})\text{vec}(F) = \text{vec}(B^{+}F(A^{+})^{T}) \tag{2}$$

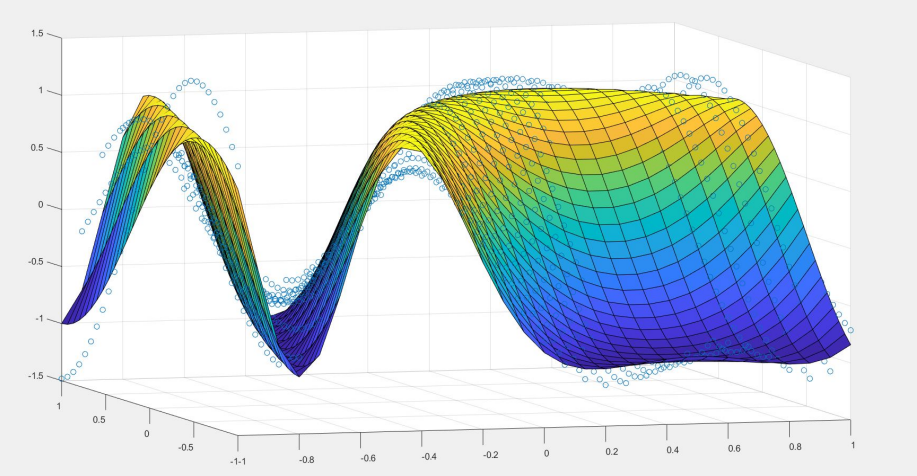
(Omdat we weten dat:)

$$(A \otimes B)^{+} = A^{+} \otimes B^{+} \tag{3}$$

$$(A \otimes B)\text{vec}(F) = \text{vec}(BFA^{T}) \tag{4}$$

Oorspronkelijk werd (2) gebruikt in plaats van (1) omdat we dachten dat (2) ons toeliet om slechts 1 inverse matrix uit te rekenen in plaats van 2 en zo

Figuur 1: Functiebenadering van $\sin((2x-1)^2+2y)$



de hoeveelheid benodigd werk doet dalen. Dit was natuurlijk fout omdat $A \otimes B$ dimensies heeft die het product zijn van de dimensies van de A en B matrices. Toen de functiebenadering van de Etna moest gegenereerd worden bleek al snel dat het gebruik van (2) helemaal geen optimalisatie was in termen van geheugen. Het Kronecker product zorgde er namelijk voor dat matrices zo groot werden dat ze niet meer in het geheugen pasten, en dat bij kleinere waarden van m en n het verschillende minuten duurde voordat de matrix gegenereerd werd (dit is logisch want bij een $m = n = 25$ is $\dim(A) = 1425 \times 25$ en $\dim(B) = 1425 \times 25$ waardoor het Kronecker product een dimensie van 2030625×625 had).

1.3 2 functiebenaderingen

We gebruiken nu de kkb functie om enkele 2-D oppervlakken te benaderen. De gegeven functie polyval laat ons toe aan de hand van het resultaat van onze kkb-functie de benaderde waarden voor x en y te evalueren. Op Figuur 1 en Figuur 2 is de functie voorgesteld als een oppervlak en de benaderde waarden als punten (de blauwe cirkels) die boven of onder dit oppervlak verschijnen.

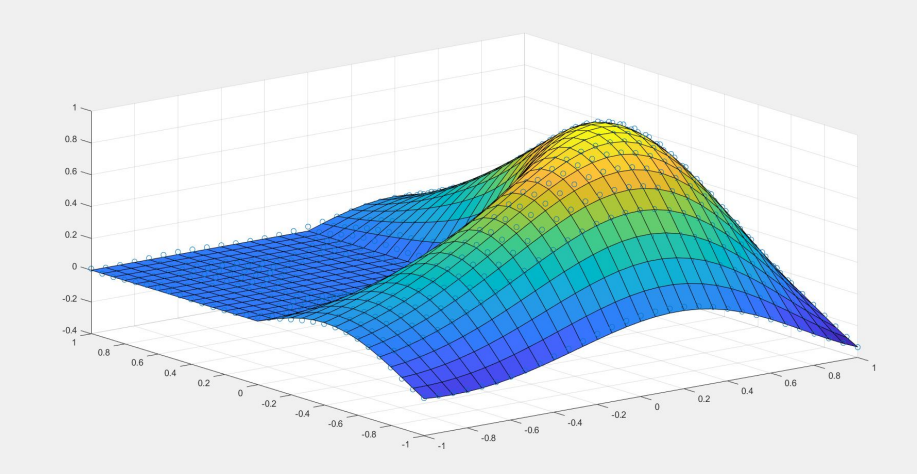
1.4 De kostfunctie afhankelijk van de graad van de benaderende veelterm

De volgende stap is het effect bepalen van de graad van de benaderende veelterm op de nauwkeurigheid van de benadering. Om dit te doen gebruikten we de volgende formule: $\sum_{i=1}^{M} \sum_{j=1}^{N} (f_{ij} - z(x_i, y_j))^2$. Het resultaat is te zien op figuur 3.

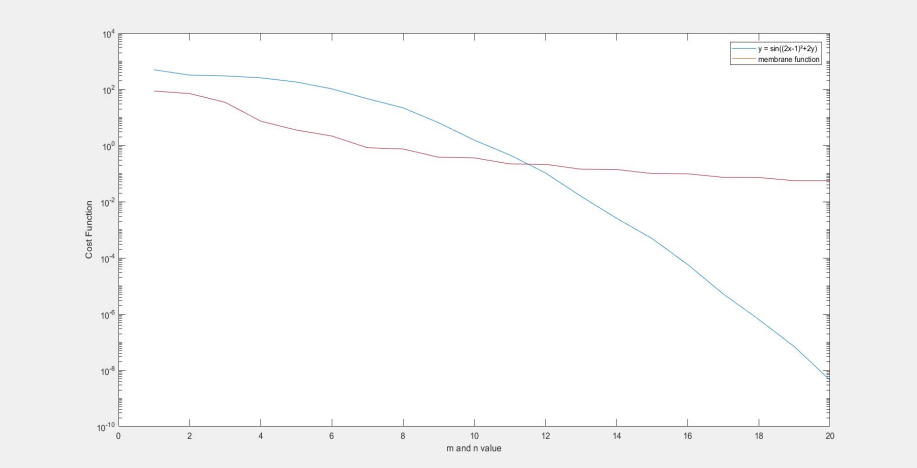
1.5 Veeltermbenadering Etna

De veeltermbenadering van de Etna werd op praktisch dezelfde manier geïmplementeerd als de functiebenaderingen in oefening 1.2. Zoals al eerder vermeld kwam het door deze functiebenadering dat we begrepen dat onze oorspronkelijk implementatie van kkb niet optimaal was.

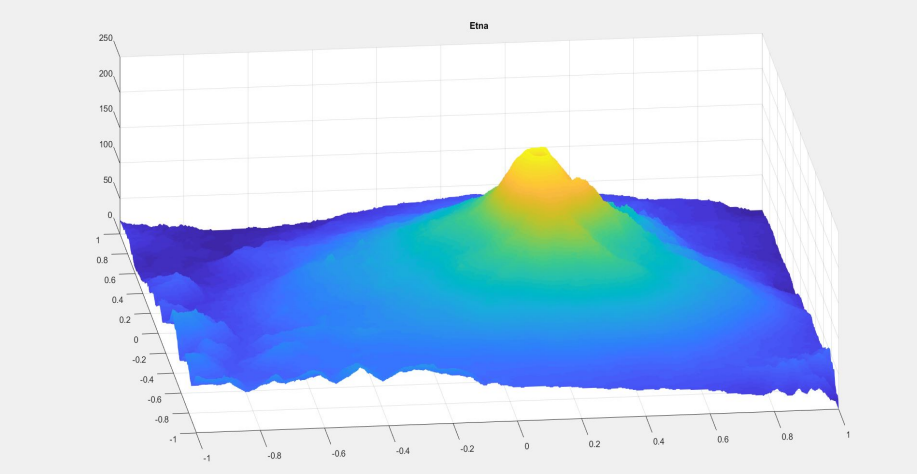
Figuur 2: Functiebenadering van de membraan functie



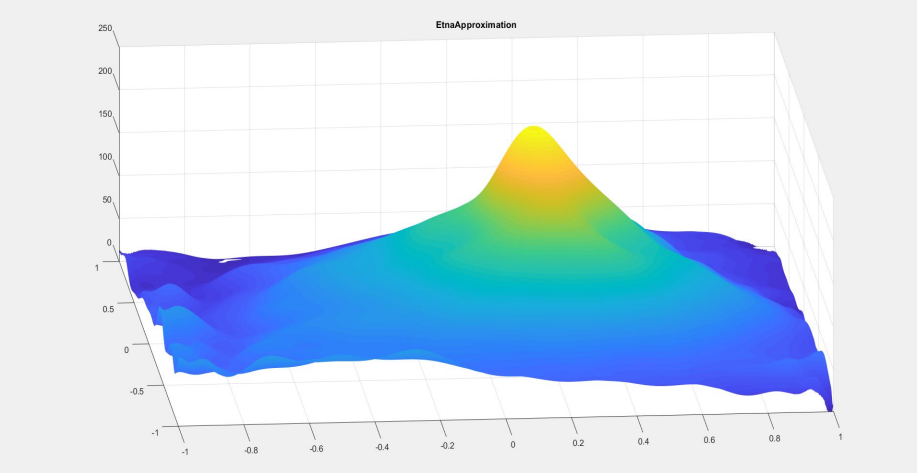
Figuur 3: Kostfunctie als functie van n=m



Figuur 4: De Etna volgens ASTER Global Digital Elevation Map



Figuur 5: Functiebenadering van de Etna met $m = n = 25$



2 Benadering via splines

Deze opgave vroeg ons om een Matlab-functie te schrijven die een optimale interpolerende kubische spline-benadering maakt en dan deze spline evalueert in de gegeven functiewaarden. Met deze functie is het dan de bedoeling om enkele functies of krommen te benaderen. De resultaten en gebruikte werkwijzen worden in de volgende paragrafen besproken.

2.1 Deel 1

Als gegevens om de spline te bepalen worden alle abscissen gegeven van de te bepalen functie of krommen (x_0 tot x_n) samen met de alle functiewaarden behalve de laatste (f_0 tot f_{n-1}). De laatste functiewaarde moet volgens de definitie van periodieke spline gelijk zijn aan de eerste ($f_n = f_0$).

Voor het efficiënt oplossen van krommen waarbij meerdere splines met dezelfde abscissen bepaald moeten worden, mag de lijst van functiewaarden ook een matrix zijn, waarbij elke rij in deze matrix één set van functiewaarden bepaald.

Met deze gegevens kan dan volgens de theorie uit paragraaf 5.2.3 uit de cursus de matrix A en vector b (of matrix met in iedere kolom de vector horende bij een set gegeven functiewaarden) bepaald worden. Deze kunnen dan gebruikt worden om het stelsel $As = b$ op te lossen naar s, wat een vector (of matrix met in iedere kolom de vector horende bij een set functiewaarden, in dezelfde volgorde als in matrix b) geeft met de waarden s''_0 tot s''_{n-1}. Uit de definitie van periodieke splines kunnen we bepalen dat s''_n gelijk moet zijn aan s''_0.

Hiermee bepalen we dan als laatste de constanten c_1 en c_2 voor iedere 3de-graads polynoom p_1 tot p_n in de spline, waardoor we een vector (of matrix, met zelfde structuur als s) met alle c_1's en c_2's bekomen. Hiermee zijn alle onbekenden van onze spline benadering(en) bepaald.

Voor het evalueren van de spline-benadering gebruiken we het kenmerk van deze benadering dat iedere bepaalde 3de-graads polynoom p_i slechts geldig is binnen het interval $[x_{i-1}, x_i]$. De abscissen voor de te bepalen functiewaarden worden gegeven in een vector t. Het volstaat dus voor iedere waarde t_j het interval te zoeken waarbij $x_{i-1} \leq t_j \leq x_i$ en dan de polynoom horende bij dit interval te evalueren in t_j. Indien meerdere splines benaderd werden, gebeurt deze evaluatie ook voor iedere spline op hetzelfde moment (eg. We berekenen de functiewaarde in t_j voor iedere benaderde spline in 1 Matlab operatie, gezien Matlab ons toestaat operaties rechtstreeks op vectoren uit te voeren).

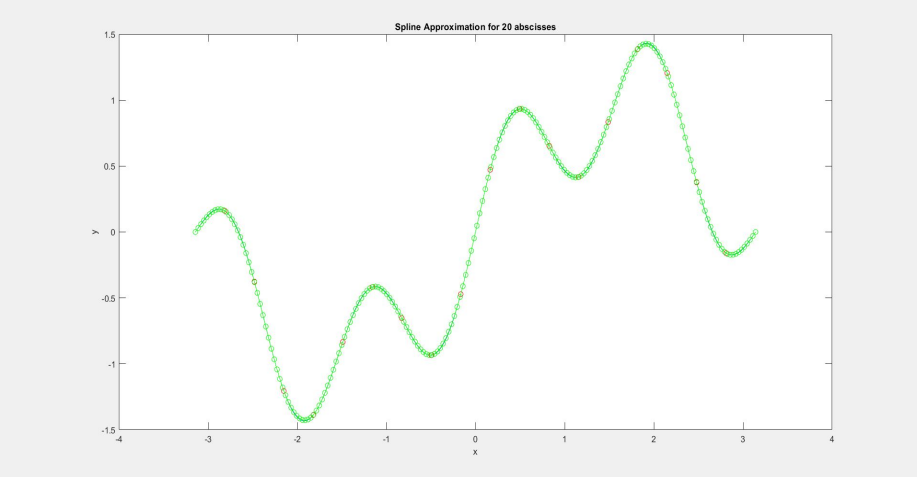
2.2 Deel 2

We gebruiken nu de functie die hierboven beschreven werd om de functie

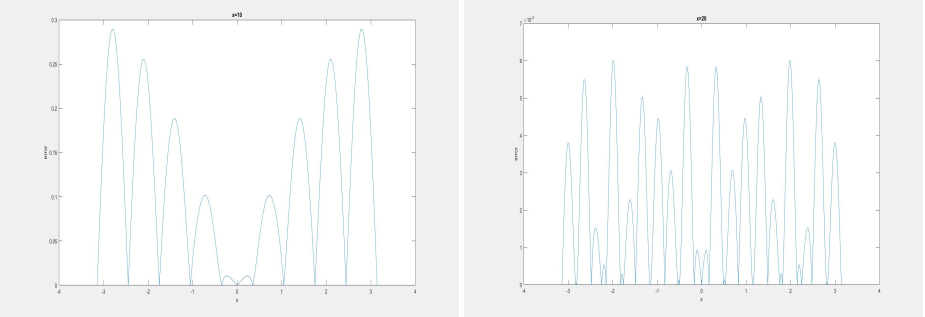
$$f(x) = \sin(x) + \frac{\sin(4x)}{2} \tag{5}$$

te bepalen.
Figuur 6 toont de benadering met 20 equidistant gekozen abscissen. We zien hier dat voor 20 abscissen de functie vrij goed benaderd lijkt te zijn. De rode

Figuur 6: Spline Benadering met 20 abscissen



Figuur 7: Absolute fout spline benadering



punten zijn de gegeven functiewaarden, de groene de punten geëvalueerd op de spline.

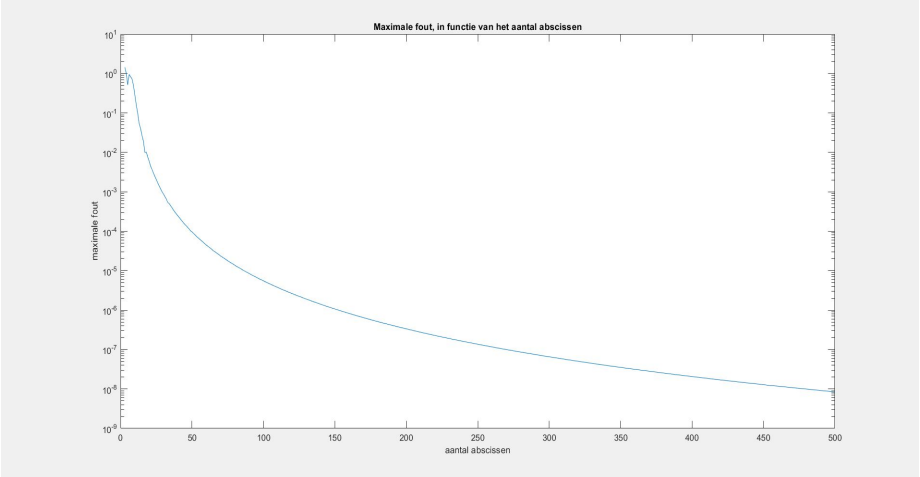
Figuur 7 toont de absolute fout tussen de gegeven functie en de spline benadering voor aantal abscissen gelijk aan 10 en 20 (opnieuw equidistant gekozen). We zien duidelijk dat de fouten verkleinen maar steeds oscilleren tussen de vastgelegde punten (dit zijn de nulpunten). Dit is te verwachten met een veeltermbenadering.

Figuur 8 toont de maximale fout in de benadering bij een gegeven aantal abscissen, de error wordt logaritmisch geplot. We zien dat de fout in het begin heel sterk afneemt, maar dan afzwakt. Een foutloze benadering kan nooit bekomen worden, benaderen tot op machineprecisie nauwkeurig is misschien mogelijk maar zou zeer veel rekenwerk vergen.

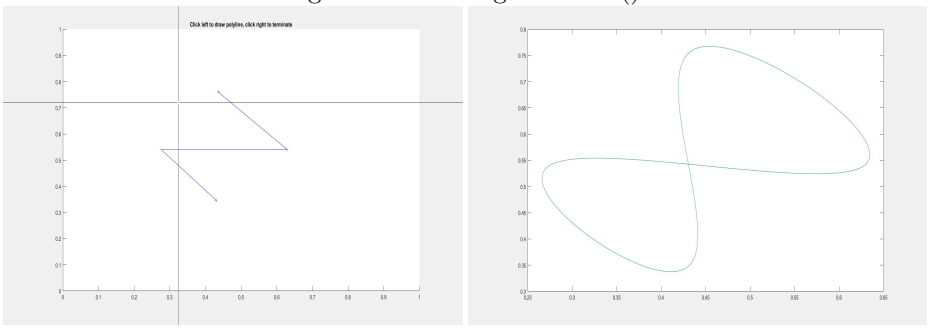
2.3 Deel 3 en Deel 4

In deel 3 krijgen we x en y simpelweg door click() en evalueren we de x en y waarden samen. Als we op 4 willekeurige plaatsen klikken krijgen we figuur 9. We kunnen dan voor deel 4 bijvoorbeeld de striptekenfiguur Sidonia tekenen met deze functie. (Zie figuur 10). Om deze 2D spline-functies te evalueren hebben

Figuur 8: Maximale fout in functie van het aantal abscissen



Figuur 9: Tekening met click()



we gewoon aangenomen dat de abscissen (de t-waarden) equidistant zijn want we hebben geen functievoorschrift van de te benaderen functie.

Figuur 10: Sidonia getekend via Splines