Prof. Dr. Dirk Lebiedz
Dipl.-Math. oec. Klaus Stolle
Institut für Numerische Mathematik
Universität Ulm

WiMa-Praktikum II (Numerik)
Wintersemester 2016/2017

25. Januar 2017

# Projekt 4 - Adaptive Gitterverfeinerung

Im letzten Proekt haben wir Triangulierungen und ihre Datenstruktur genauer betrachtet sowie als erste Möglichkeit einer Verfeinerung eine gegebene Triangulierung gleichmäßig verfeinert. Gleichmäßige Verfeinerungen haben den Nachteil, dass auch an Stellen des Gebietes verfeinert wird, an denen eine gröbere Triangulierung durchaus ausreichend wäre. Daher werden in der Praxis häufig adaptive Verfeinerungen angewendet. Hier wird die Triangulierung gezielt an Stellen verfeinert, an denen eine höhere Auflösung gewünscht wird.

## 1 Adaptive Netz-Verfeinerung

Um ein Netz aus Dreiecken adaptiv verfeinern zu können, werden Fehler-Indikatoren benötigt. Diese geben für jedes Element eine Abschätzung für den Fehler an. Grob gesprochen werden dann diejenigen Elemente verfeinert, auf denen der Fehler am größten ist. Wir wollen in diesem Projekt verschiedene Strategien zur Markierung der zu verfeinernden Elemente betrachten: Zu einer Triangulierung $\mathcal{T}$ sei der Vektor $\eta = (\eta_T)$, der für jedes Element $T \in \mathcal{T}$ den Fehlerindikator $\eta_T$ enthält, gegeben.

- Strategie 1 (Absolutes Kriterium):
  Verfeinere alle Dreiecke $T \in \mathcal{T}$ für die gilt: $\eta_T \geq \theta$, $\theta \in \mathbb{R}$.
- Strategie 2 (Relatives Kriterium):
  Verfeinere alle Dreiecke $T \in \mathcal{T}$ für die gilt: $\eta_T \geq \theta \max\limits_{T' \in \mathcal{T}} \eta_{T'}$, $\theta \in [0, 1]$.
- Strategie 3 (Dörfler-Marking/Bulk-Chasing):
  Suche eine minimale Menge $\mathcal{M} \subset \mathcal{T}$ für die gilt: $\theta \sum_{T \in \mathcal{T}} \eta_T^2 < \sum_{T \in \mathcal{M}} \eta_T^2$, $\theta \in [0, 1]$. Verfeinere dann alle $T \in \mathcal{M}$.

Es gibt nun drei Möglichkeiten, ein Element zu verfeinern: die sogenannte Rot-, Grün- oder Blau-Verfeinerung, siehe Abbildung 1.

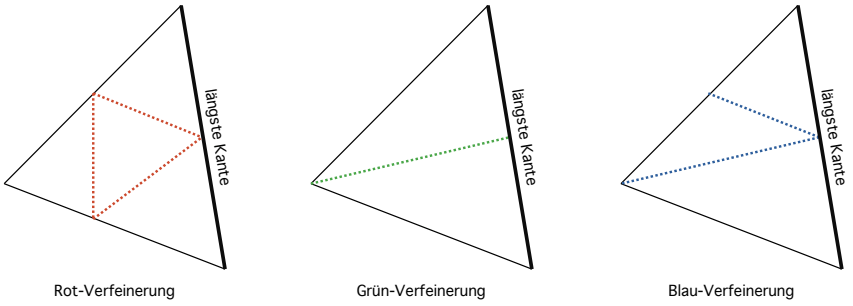



Abbildung 1: Rot-, Grün- und Blau-Verfeinerung

Um ein verfeinertes Netz zu erhalten, das immer noch zulässig ist (d.h. keine hängenden Knoten usw.), kann man bei der Verfeinerung wie folgt vorgehen:

1. Markiere bei allen markierten Elementen alle drei Kanten.

2. Markiere bei jedem Element, das mindestens eine markierte Kante hat, auch die längste Kante (falls diese nicht bereits markiert ist).
   Beachte: Wiederhole diesen Schritt so lange, bis keine neuen Kanten mehr markiert worden sind.
3. Der Mittelpunkt jeder markierten Kante wird ein neuer Knoten.
4. Falls alle Kanten eines Elements markiert sind: Verfeinere dieses Element rot.
5. Falls zwei Kanten eines Elements markiert sind: Verfeinere dieses Element blau.
   Beachte: Verwende die längste Kante als „Referenzkante“.
6. Falls nur eine Kante eines Elements markiert ist: Verfeinere dieses Element grün.

# 2 Aufgabenstellung

In diesem Projekt werden wir mithilfe eines adaptiven Algorithmus die Gitterverfeinerung implementieren. Üblicherweise folgt man dabei dem Schema

**SOLVE** → **ESTIMATE** → **MARK** → **REFINE**,

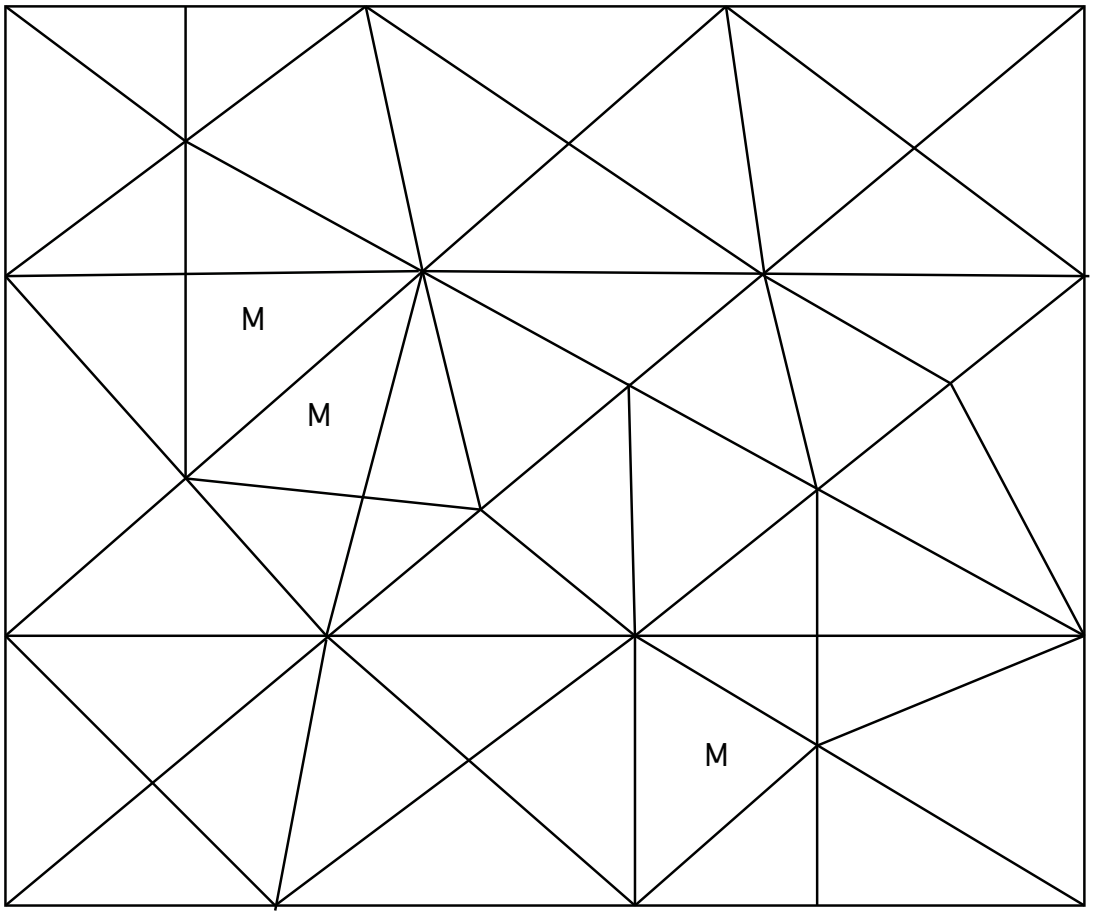
was so lange wiederholt wird, bis der Fehlerschätzer im **ESTIMATE**-Schritt klein genug ist. Wir betrachten hier aber nun keine partielle Differentialgleichung, sondern einfache Bilddaten, so dass der **SOLVE**-Schritt wegfällt.

Die Aufgabe ist nun, die adaptive Netzverfeinerung zu programmieren.

Laden Sie dazu das Material von der Homepage herunter und führen Sie folgende Arbeitsschritte durch.

## 2.1 Theorie verstehen

Zeichnen Sie in das folgende Bild eine mögliche Verfeinerung der mit „M“ markierten Elemente ein.




Schreiben Sie stichpunktartig auf, wie man bei der Verfeinerung vorgehen könnte.

## 2.2 ESTIMATE: Fehler berechnen

Eine Funktion `computeEtaR`, welche für alle Elemente die maximale Differenz zwischen den Farbwerten (in Graustufen) berechnet, ist bereits gegeben.

## 2.3 MARK: Elemente markieren

Schreiben Sie Funktionen `markElements_absolute`, `markElements_max` und `markElements_Doerfler`, welche die obigen Strategien umsetzen, um zu einem gegebenen Fehlerindikator `etaR` die zu verfeinernden Elemente zu markieren.

## 2.4 Einige Hilfs-Matrizen aufbauen

Wir gehen davon aus, dass wie üblich alle Elemente, Kantenstücke und Knoten durchnummeriert sind (die sogenannte *globale* Nummerierung). Zum Verfeinern einzelner Elemente ist es jetzt beispielsweise hilfreich zu wissen, welche Kanten gerade das $i$-te Element bilden. Dafür brauchen wir die folgenden Hilfs-Datenstrukturen, welche aus den Strukturen `coordinates`, `elements` und `boundary` erzeugt werden (wir unterscheiden hier diesmal nicht zwischen Dirichlet- und Neumann-Rand, weil wir ja Bilder betrachten):

- `element2edges`: Gibt die globalen Kanten-Indizes der lokalen 3 Kanten eines Elements an. Dabei soll `element2edges(i,l)` die Nummer der Kante zwischen den Knoten `elements(i,k)` und `elements(i,(k mod 3)+1)` angeben.
- `edge2nodes`: Gibt die Indizes der zu einer Kante gehörenden Knoten an. Die Zeile `edge2nodes(i,:)` enthält also gerade die Nummern $j, k$ der Knoten an der $i$-ten Kante.
- `boundary2edges`: Gibt die globalen Kanten-Indizes der Rand-Kanten an, `boundary2edges(i)` ist also die Nummer der $i$-ten Kante des Randes.

Schreiben Sie dazu eine Funktion

```
[edge2nodes,element2edges,boundary2edges] = provideGeometricData(elements,boundary).
```

## 2.5 REFINE: Verfeinerung durchführen

Mithilfe der obigen Matrizen kann nun die eigentliche Verfeinerung durchgeführt werden. Schreiben Sie dazu eine Funktion

```
[coordinates,newElements,newBoundary] = refineRGB(coordinates,elements,boundary).
```

## 2.6 Hauptprogramm

Das Hauptprogramm ist im Wesentlichen vorgegeben. Vollziehen Sie nach, was dort passiert, und probieren Sie die verschiedenen Verfeinerungs-Strategien (und auch Bilder) aus.