
BACHELORARBEIT

FEM-FORTRAN90 PROGRAMM IN C++ ERWEITERN, LAUFZEITMESSUNGEN UND DOXYGEN MIT
L^AT_EX DOKUMENTATION EINFÜHRUNG

BEARBEITET VON

CHEN YOYU

MATRNR: 1351627

STUDIENGANG: MASCHINENBAU-INFORMATIK

ZEITRAUM: 20.08.2018 BIS 19.08.2018

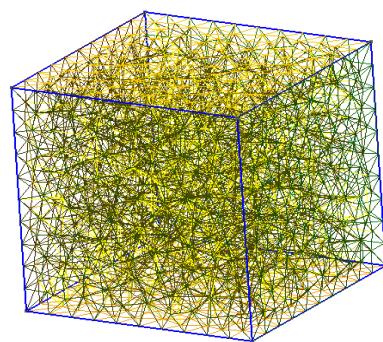
ERSTPRÜFER:

PROF. DR. PIEPK WOLFGANG

ZWEITPRÜFER:

PROF. DR. HENTSCHEL-RAPSCH CLAUS

*Hochschule Hannover
Fakultät II
Maschinenbau und Bioverfahrenstechnik*



06.09.2018

Inhaltsverzeichnis

1 Einleitung	2
2 Gmsh Einführung	3
2.1 Einleitung	3
2.2 Installation	3
2.3 Zeichnen in Gmsh-GUI	3
2.4 Vernetzen in Gmsh-GUI	7
2.5 Import aus CAD-Datei	8
2.6 Format der Netzdatei	10
3 Problemstellung	12
3.1 Wärmeleitungsproblem	12
4 Aufbau des Gleichungssystems	14
4.1 Linearer Ansatz	14
4.2 Intergration für das Tetraeder	16
4.3 Lokale Steifigkeitsmatrix	16
4.4 Lokale rechte Seite \vec{b}	20
4.5 Beispiel Geometrie	21
4.6 Die Globale Steifigkeitsmatrix	22
4.7 Die globale rechte Seite \vec{b}	23
4.8 Gleichungssystem mit Randbedingung	24
5 Gleichungssystem Optimierung	25
5.1 Warum optimiert man das Gleichungssystem ?	25
5.2 Optimierung	25
6 Das numerische Verfahren BiCGSTAB-Verfahren	26
6.1 Projektionsmethoden und Krylov-Unterraum-Verfahren	26
6.2 Warum wird das BiCGSTAB-Verfahren gewählt?	27
6.3 Algorithmische Schritte	27
7 Doxygen Einführung	29
7.1 Einleitung	29
7.2 Installation	29
7.3 Benutzen der Doxywizard	29
8 Versionsverwaltung durch Eclipse(IDE)	39
8.1 Einleitung des Eclipse(IDE)	39
8.2 Installation	39
8.3 Versionsverwaltung	39

8.4	Import eines Makefile-Projekts	39
8.5	Konfiguration des Makfiles	41
8.6	Git Repository erzeugen	43
8.7	Version pflichten	44
9	Ergebnis und Zeitmessung	48
9.1	Ergebnis	48
9.2	Zeitmessung und Vergleich mit unterschiedlichen Versionen	49
10	Fazit und Ausblick	52
10.1	Fazit	52
10.2	Ausblick	52
A	Anhang	54
A.1	Das C++ Programm	54
A.1.1	Hauptprogramm <i>FEM_OBJ.cpp</i>	54
A.1.2	Unterprogramm Netz dimensionieren <i>dim.cpp</i>	54
A.1.3	Unterprogramm Netz einlesen <i>input.cpp</i>	55
A.1.4	Unterprogramm Randpunkte sortieren <i>randpunkte.cpp</i>	56
A.1.5	Unterprogramm Steifigkeitsmatrix berechnen <i>SteifigkeitsMatrix.cpp</i>	58
A.1.6	Unterprogramm Gleichungssystem optimieren <i>matopt.cpp</i>	60
A.1.7	Unterprogramm BiCGSTAB Verfahren <i>bicgstab_solve.cpp</i>	61
A.1.8	Unterprogramm Skalarprodukt <i>dot_product.cpp</i>	63
A.1.9	Unterprogramm Matrix multipliziert mit einem Vektor <i>matmulvkt.cpp</i>	64
A.1.10	Unterprogramm ein Konstant multipliziert mit einem Vektor <i>constmulvkt.cpp</i>	64
A.1.11	Unterprogramm ein Vektor plus ein Vektor <i>vktplus.cpp</i>	65
A.1.12	Unterprogramm ein Vektor minus ein Vektor <i>vktminus.cpp</i>	65
A.1.13	Unterprogramm Gnuplot Skript <i>cg_Tetraed.dem</i>	66
A.1.14	Include File <i>funktions.h</i>	66
A.1.15	Include File <i>GLS.h</i>	67
A.1.16	Include File <i>Netzdim.h</i>	67
A.1.17	Include File <i>Netz.h</i>	67
A.1.18	Include File <i>OptGLS.h</i>	67
A.1.19	Include File <i>RandPkt.h</i>	68
A.2	Doxygen Dokumentation Datei	69

1 Einleitung

Die Finite-Elemente-Methode(FEM) ist ein numerisches Verfahren, welches bei unterschiedlichen physikalischen Aufgabenstellungen wie z.B. Strukturanalyse-, Wärmeleitungs-, Fluidströmung-Probleme, Festigkeitsuntersuchung, elektromagnetische Potentialberechnung, ... angewendet wird.

In dieser Arbeit wird das Wärmeleitungsproblem mit Hilfe der Finiten-Elemente-Methode (hinfür FEM genannt) untersucht. Die Wärmeleitungsgleichung (vom Diffusionsgleichungs-Typ) ist eine partielle Differentialgleichung. Sie beschreibt den Zusammenhang zwischen der räumlichen und zeitlichen Änderung der Temperatur in einem Körper und eignet sich zur Berechnung instationärer Temperaturfelder. Hier wird nur die räumliche Änderung untersucht.

Die FEM stellt ein Werkzeug zur Verfügung, welches numerische Lösungen eines komplexen Systems aus Differentialgleichungen berechnen läßt. Das Berechnungsgebiet (z.B. ein Festkörper mit Rand) wird in endlich viele kleine¹ Teilgebiete einfacher Form aufgeteilt, z.B in viele Tetraeder oder Hexaeder. Sie sind sogenannte "finite Elemente", wenn noch die Ansatzfunktion darauf definiert wird, d.h. eine komplexe Geometrie wird in viele kleine einfache geometrische Körper zerlegt und somit vernetzt. Eine Geometrie wird durch eine Netz-Software vernetzt, normalerweise benutzt man folgende Software: ICEM, ElmerFEM, Gmsh, MeshLab usw.

Das physikalische Verhalten der Lösung lässt sich aufgrund der einfachen Geometrie der Netzelemente mit bekannten Ansatzfunktionen untersuchen. Jedes Element besitzt einen Anteil im linearen Gleichungssystem und diese einfachen Einzelement-Gleichungen, die für jedes finite Element die Lösung approximieren, werden dann zu einem großen linearen Gleichungssystem zusammen assembliert, um das gesamte Problem zu lösen.

Das lineare Gleichungssystem wird danach durch numerische Verfahren gelöst z.B. Jacobi-Verfahren, Bi-Lanczos-Algorithmus, BiCG-Verfahren usw. Hier wird das Wärmeleitungsproblem mit einer bestimmten Geometrie durch Programme in den Programmiersprachen FORTRAN90 und C++ programmiert, gelöst und Laufzeitmessungen durchgeführt.

Ein Programm besteht aus dem Quellcode und den dazugehörigen Kommentaren. Mit einem bestimmten Format können alle Kommentare in eine Dokumentation übernommen werden. Dies gelingt durch das Software-Werkzeug *Doxxygen*, mit dessen Hilfe der Quellcode des Programms gut erklärt, d.h. dokumentiert werden kann.

Software-Engineering bedeutet, daß ein fertiges Programm weiterentwickelt wird und somit sich fortlaufend verändert. In diesem Prozess erhält man natürlich im Laufe der Zeit viele Versionen für das ganze Projekt. Zum Zweck der Versionspflege wird ein Werkzeug vorgeschlagen, welches die Software verwaltet und mit dem man das Programm pflegen kann. Dazu wird hier in dieser Arbeit die Eclipse-IDE als Software vorgestellt und benutzt, so daß die Eclipse-IDE eine gute Lösung für das Problem der Versionspflege liefert.

¹Dieses Konzept hat sich bewährt, z.B. beim Differenzieren in einem Punkt, wo man ebenfalls in einer kleinen Umgebung des Punktes den Grenzwert des Differentialquotienten berechnet und in dieser kleinen Umgebung etwas Krummes (die Kurve) durch etwas Gerades (die Tangente) approximieren kann.

2 Gmsh Einführung

Vor dem Lösen des Wärmeleitungsproblems benötigt man erst ein Input-Netz-File. In diesem Kapitel wird erklärt, wie man durch **Gmsh** [1] ein Netz-File erzeugt.

2.1 Einleitung

Gmsh ist ein Finite-Elemente-Netzgenerator, der von Christophe Geuzaine und Jean-François [1] entwickelt wurde. Das Ziel besteht darin, ein schnelles, leichtes und benutzerfreundliches Vernetzungswerkzeug mit parametrischer Eingabe und erweiterten Visualisierungsfunktionen bereitzustellen. Gmsh besteht aus 4 Modulen: geometry, mesh, solver und post-processing. Die Spezifikation der Eingabe erfolgt entweder über die grafische Benutzeroberfläche oder in einer ASCII Textdatei mit der eigenen Skriptsprache von Gmsh (.geo-Datei). Um eine komplexe Geometrie zu vernetzen, muß auch eine CAD-Datei importiert werden können.

2.2 Installation

Unter Ubuntu kann man durch folgendes Kommando Gmsh installieren:

- `sudo apt-get install gmsh`

2.3 Zeichnen in Gmsh-GUI

Eine einfache Geometrie, z.B. ein Würfel oder ein Tetraeder, kann in der Gmsh-GUI vernetzt werden z.B. durch Tetraeder. Mit folgendem Kommando wird eine grafische Benutzeroberfläche geöffnet:

- `gmsh`

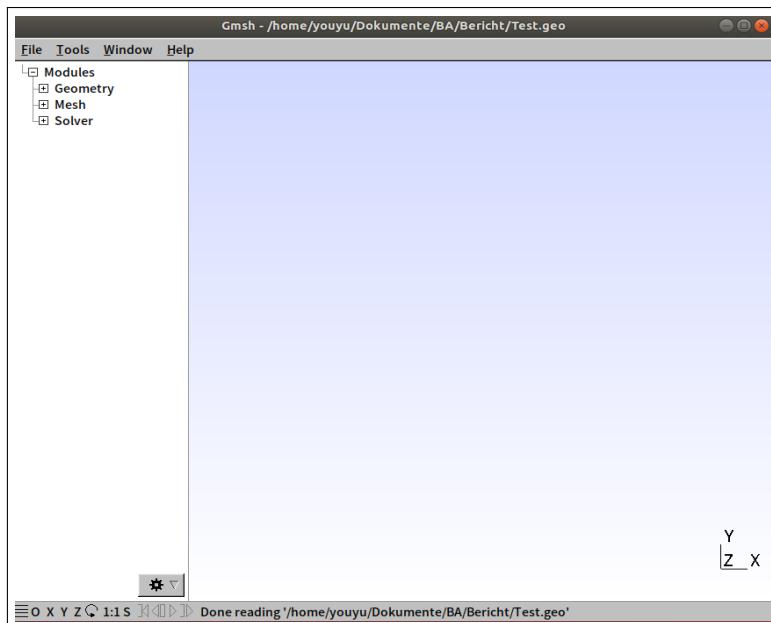


Abbildung 1: Gmsh Start

Bei dieser Version gibt es nur 3 Module nämlich "Geometry", "Mesh" und "Solver", die an der linken Seite des Fensters stehen. Die rechte Seite ist die Bearbeitungsfläche. Hier wird ein Tetraeder als Beispiel Geometrie bezeichnet(4 Punkte, 6 Kanten, 4 Flächen, 1 Volumen). Normalerweise wird eine einfache 3D-Geometrie in der folgenden Reihenfolge erzeugt:

Punkte → Linien → Flächen → Volumen

Einmal linke Maustaste in "Geometry" → "Elementary entitie" → "Add" klicken. Da stehen die erwähnten Grund-Komponenten zur Verfügung.

1. Zuerst muß man "Point" anklicken, um einen Punkt zu bezeichnen.

Jeder Punkt hat 4 Parameter: XYZ-Koordinatensystem und die Maschenweite eines Netzelementes welches an diesem Punkt angeheftet ist. 4 Punkte werden erzeugt: (0,0,0), (1,0,0), (0,1,0), (0,0,1)
Die Netzweite ist jeweils mit 0.1 gewählt worden. Im folgenden Fenster sind die Parameter einzugeben und mit der "Add" Taste zu erzeugen.

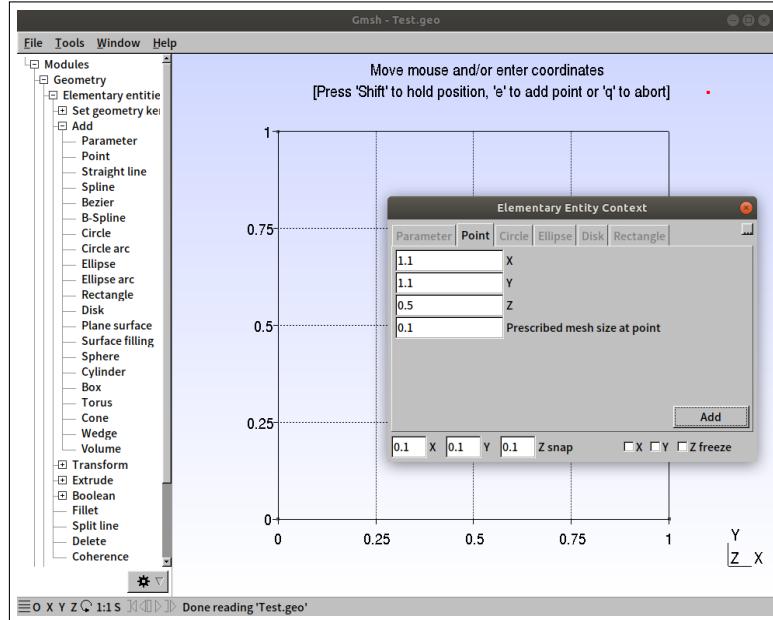


Abbildung 2: Gmsh Punkt

Jeweiliges Eingeben eines 'e' sorgt für die Abspeicherung der Eingabe, wenn man das zusätzliche Fenster nicht benutzt. Danach schließen des "Elementary Entity Context" Fensters durch drücken der Taste 'q'². Dann sind die Punkte fertig erzeugt.

²Kein Mausclick, wenn man das zusätzliche Fenster nicht benutzt.

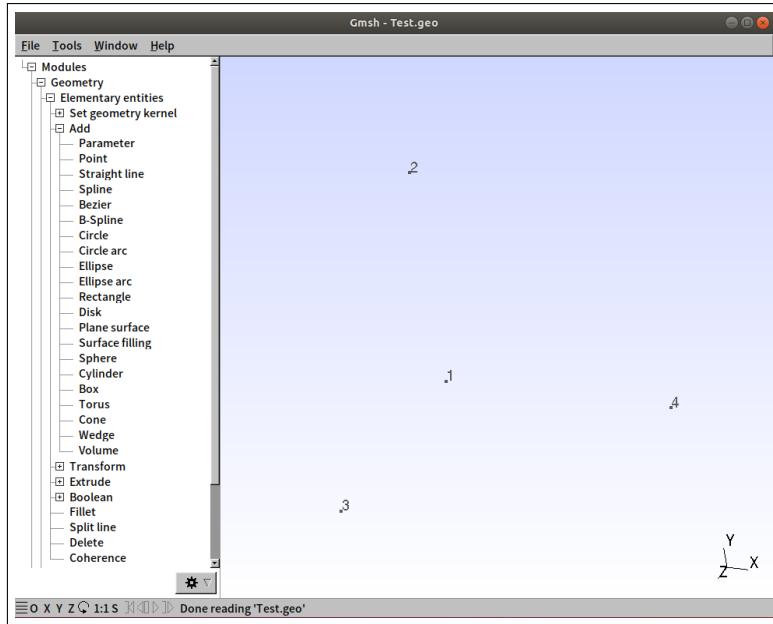


Abbildung 3: 4 Punkte

2. Dann die Taste **"Straight line"** mit einem Mausclick betätigen, um die Linien zu bezeichnen.

Eine Linie besitzt einen Anfangspunkt und einen Endpunkt, daher werden die 4 Punkte nacheinander mit einem Mausclick verbunden. Jeweilige Tasten-Eingaben eines 'e' sorgt für die Abspeicherung der Eingabe. Wenn die Punkte fertig verbunden sind, betätigen der Taste 'q' (kein Mausclick).

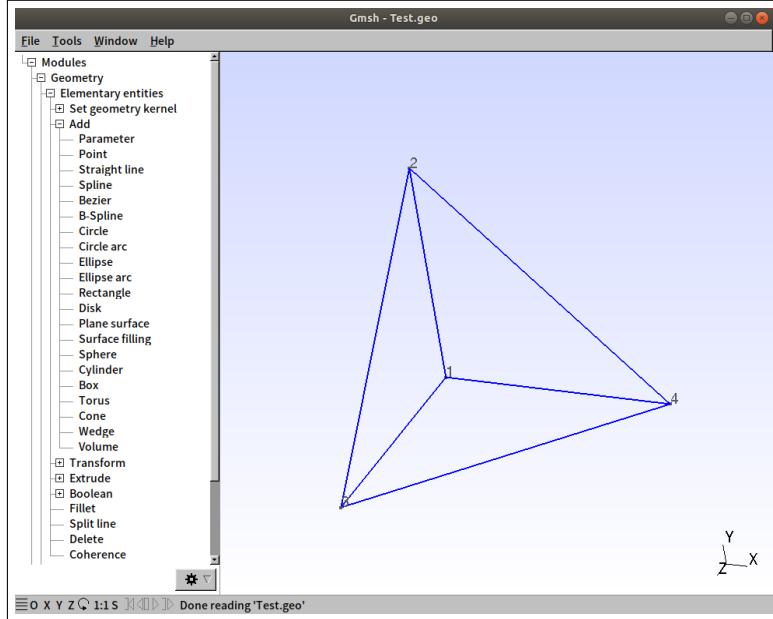


Abbildung 4: Gmsh Linie

3. Dann klicken das **"Plane surface"**, um die Fläche zu bezeichnen

Jede Fläche des Tetraeders ist ein Dreieck, daher müssen 3 Linien ausgewählt werden, um die entsprechende Fläche zu erzeugen. Wenn 3 Linien fertig ausgewählt sind, betätigen der Taste 'e', um das Abspeichern zu bewirken.

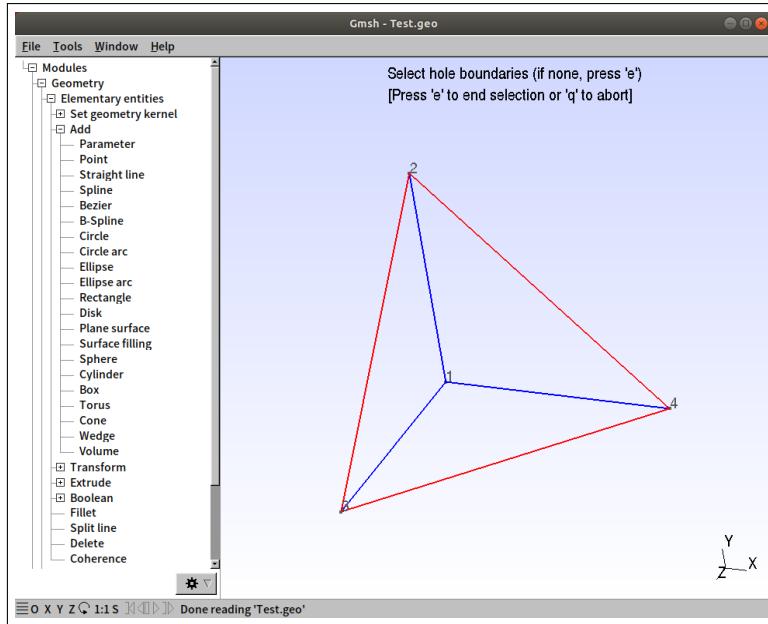


Abbildung 5: Gmsh Fläche

Die anderen 3 Flächen des Tetraeders werden mit den gleichen Schritten definiert.

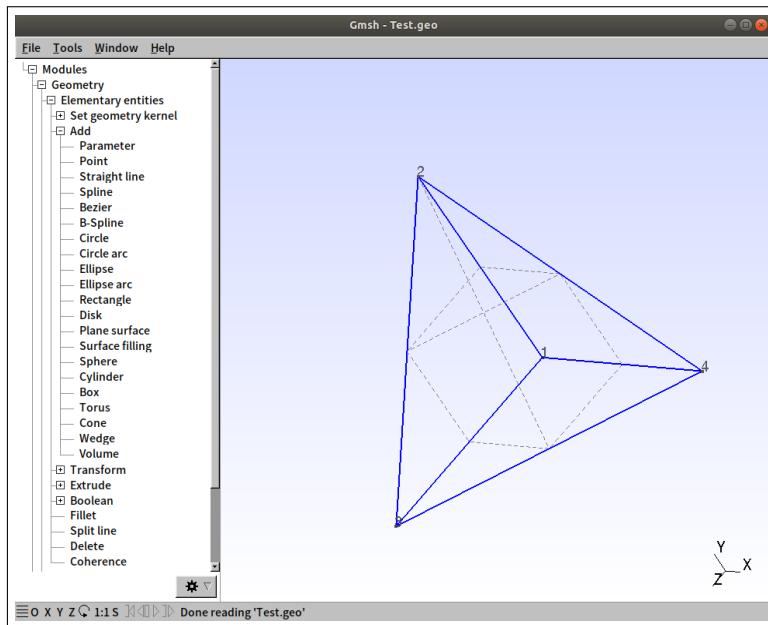


Abbildung 6: Gmsh 4 Flächen

4. Am Ende betätigen der Taste **”Volum”**, um das Volumen zu definieren

Wählen irgendeiner Fläche : **”Plane”** mit der linken Maustaste anwählen.

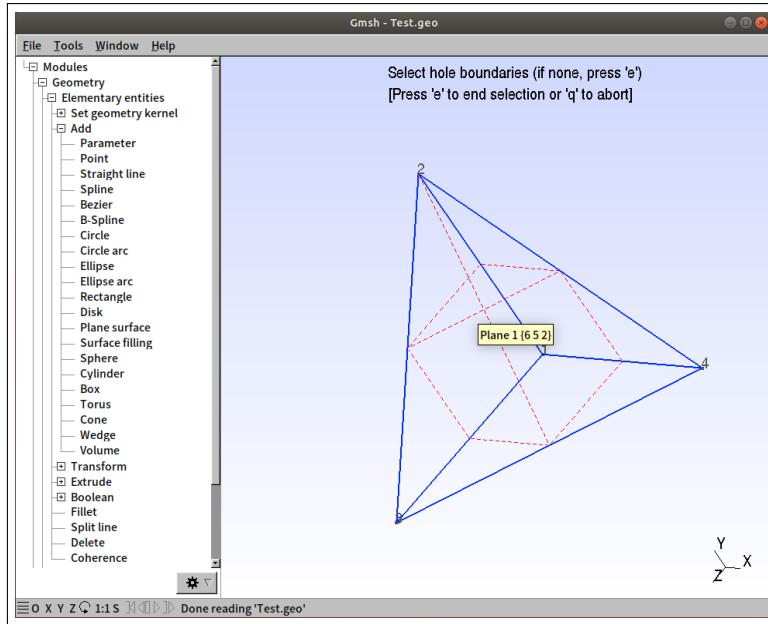


Abbildung 7: Gmsh Volumen auswählen

Jeweiliges Eingeben eines 'e' sorgt für die Abspeicherung der Eingabe. Dann klicken der Taste 'q', das Volumen wird abgespeichert.

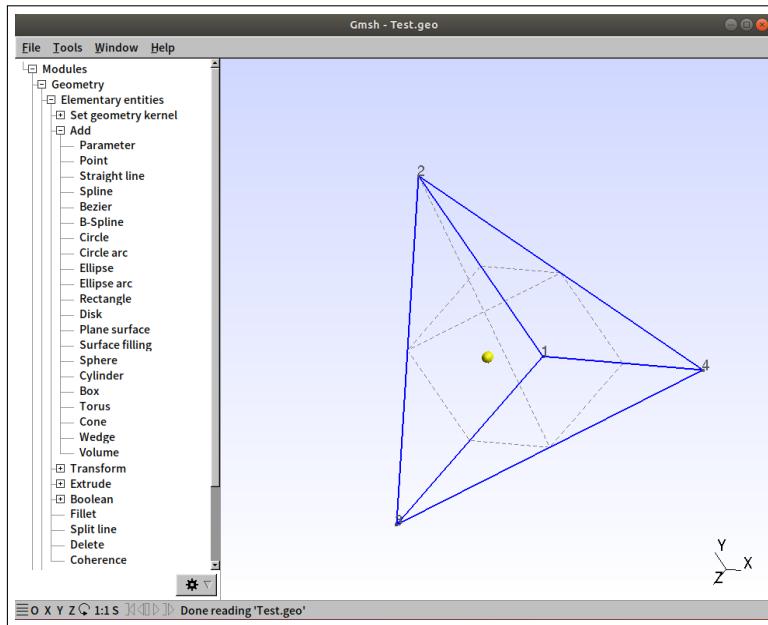


Abbildung 8: Gmsh Volumen erzeugen

Wenn Randbedingungen berücksichtigt werden sollen muß das **"Volum"** zusätzlich mit Hilfe der **"Physical Surface"** zu einem **"Physical Volum"** erklärt werden!

2.4 Vernetzen in Gmsh-GUI

Eine fertige Geometrie aus Gmsh-GUI kann als ein Gmsh-Skript gespeichert werden, welches man auch durch folgendes Kommando aufrufen kann:

- `gmsh Skriptname.geo`

Einmal linke Maustaste in "Mesh" → "3D" klicken, das Tetraeder wird automatisch vernetzt.

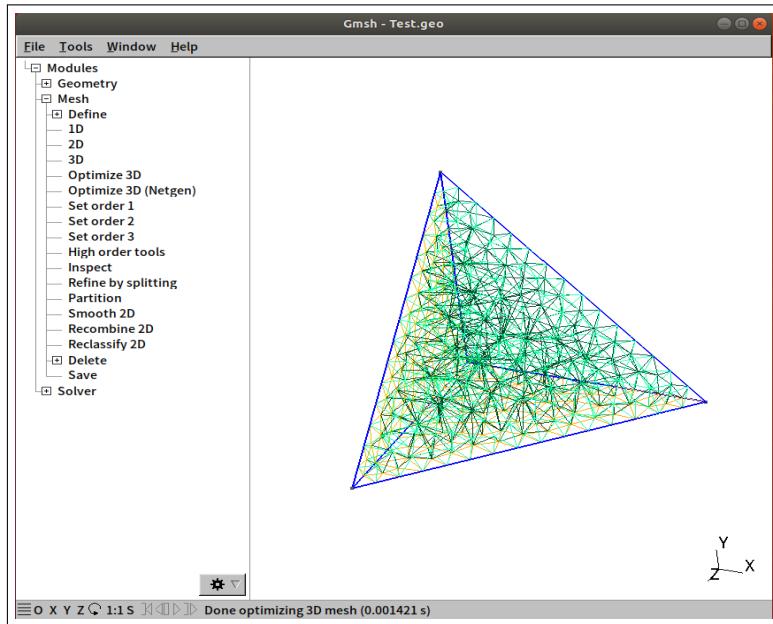


Abbildung 9: Gmsh Vernetzen

Das Netz ist als eine ASCII Textdatei(.msh) abzuspeichern, welche ein eigenes Format hat.

2.5 Import aus CAD-Datei

Eine komplexe Geometrie ist nur extrem aufwendig mit der Gmsh-GUI zu konstruieren. Daher kann die Geometrie auch aus einer CAD-Datei importiert werden, z.B. im STEP-Format ".stp" – oder IGES-Format ".igs" – Datei. Die folgende Abbildung zeigt eine ganz einfache Welle, die durch das CAD-Programm *Solid Edge* erzeugt wurde. Die Geometrie wurde von *Solid Edge* als *Part3.stp* Datei herausgeschrieben.



Abbildung 10: Welle

Zuerst den Menupunkt **”File”** → **”open”** → **”Part3.stp”** im Gmsh-GUI anwählen. In Gmsh werden automatisch alle Komponenten (Punkte, Kurven, Flächen, Volumen usw.) definiert.

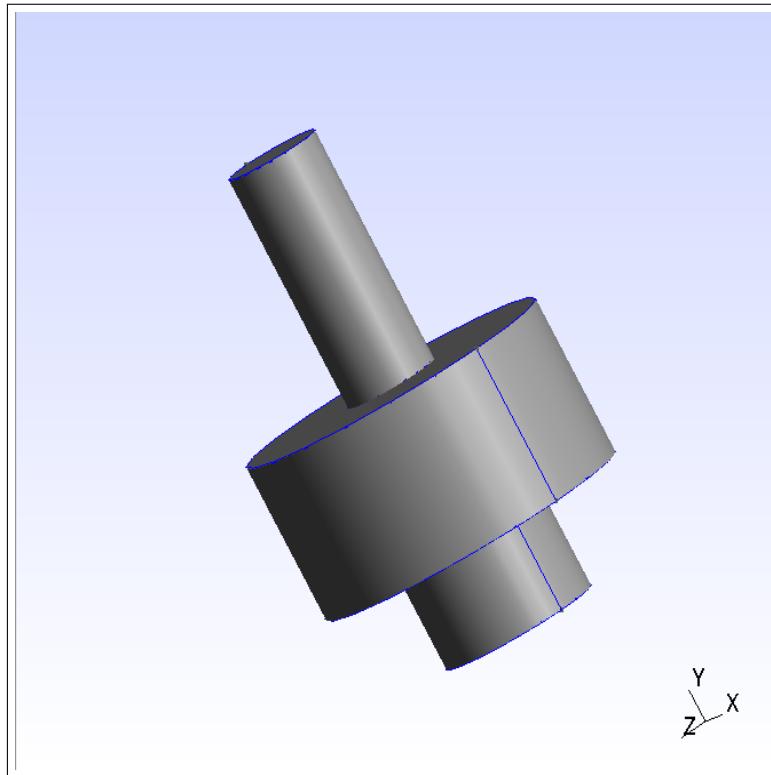


Abbildung 11: Welle im Gmsh

Klicken **”Tools”** → **”Options”**, dann ist die Größe des Elements einzustellen.

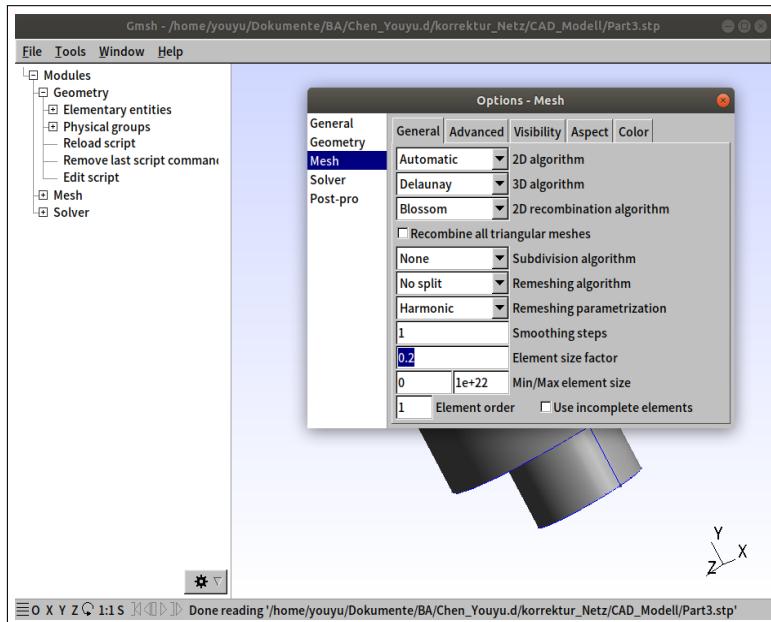


Abbildung 12: Einstellung der Elementgröße

Am Ende wird die Geometrie durch **”Mesh”** → **”3D”** Gmsh vernetzt. Das Netz wird in der folgenden Abbildung gezeigt:

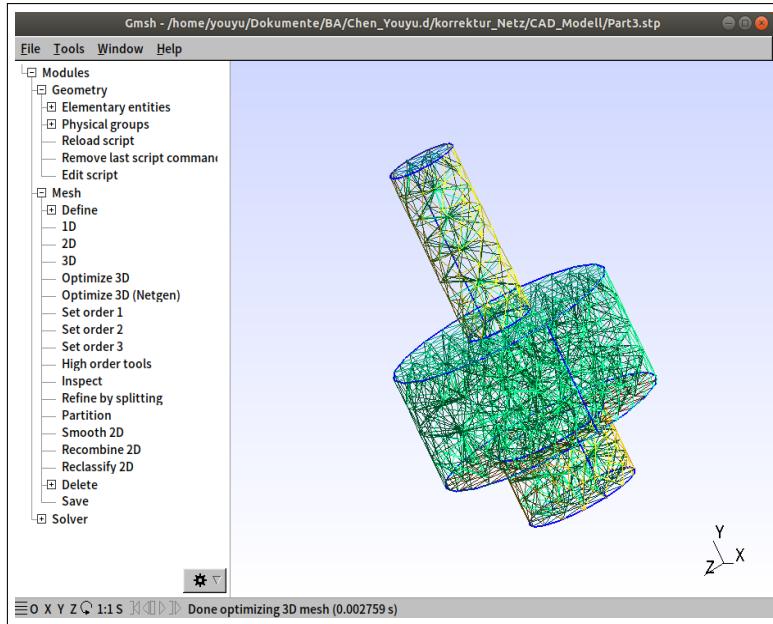


Abbildung 13: Netz der Welle

2.6 Format der Netzdatei

Das Netzdatei besitzt ein eigenes Gmsh spezifisches Format, das teilt sich in drei Teile ein: Definieren der Format-Version(**\$MeshFormat-\$EndMeshFormat**), Knoten (**\$Nodes-\$EndNodes**) und Elemente(**\$Elements-\$EndElements**) im Netz.

```

$MeshFormat
2.0 Datei-Typ Daten-Größe
$EndMeshFormat
$Nodes
Knotenanzahl
Knotennummer x-Koord. y-Koord. z-Koord.
...
$EndNodes
$Elements
Elementanzahl
Elementnummer Elementtyp EtikettNummer Etikett Knotennummerliste
...
$EndElements
```

Hier:

Datei-typ ist eine Ganzzahl und gleich Null in der ASCII Datei.

Daten-Größe ist eine Ganzzahl und gleich die Größe, die eine reelle Zahl benutzt(normalerweise Daten-Größe = sizeof(double)).

Knotenanzahl ist die Anzahl der gesamten Knoten im Netz

Knotennummer ist die Nummer(der Index) der n-ten Knoten im Netz.

x-Koord. y-Koord. z-Koord. sind die Koordinaten (3 reelle Zahlen) des n-ten Knotens im Netz.

Elementanzahl ist die Anzahl der gesamten Elemente im Netz.

Elementnummer ist die Nummer(der Index) des n-ten Elements im Netz.

Elementtyp definiert den geometrischen Typ des n-ten Elements:

- 1 Linie(2 Knoten)
- 2 Dreiecke(3 Knoten)
- 3 Vierecke(4 Knoten)
- 4 Tetraeder(4 Knoten)
- 5 Hexaeder(8 Knoten)
- 6 Prisma(6 Knoten)
- 7 Pyramide(5 Knoten)
- 8 Linie 2. Ordnung(3 Knoten)
- 9 Dreiecke 2. Ordnung(6 Knoten)
- 11 Tetraeder 2. Ordnung(10 Knoten)
- 15 Punkt(1 Knoten)

Etikettensummer ist die Nummer des Etiketts des n-ten Elements. Standardmäßig erzeugt Gmsh das Netz mit zwei Etiketten (engl. Tags) (Das Programm liest die Dateien mit einer Zufallszahl für der Etiketten ein: siehe unten. Sie werden hier nicht benötigt.).

Etikett ist eine Ganzzahl. Standardmäßig ist das erste Etikett die Nummer der physikalischen Einheit, zu der das Element gehört; Das zweite Etikett ist die Nummer der geometrischen Einheit, zu der das Element gehört; Das dritte Etikett ist die Nummer einer Netzpartition, zu der das Element gehört.

Knotennummerliste ist die Liste der Knotennummern des n-ten Elements (getrennt mit einem Leerzeichen, ohne Komma). Die Knoten werden in Gmsh automatisch geordnet.

3 Problemstellung

3.1 Wärmeleitungsproblem

Im folgenden wird ein kubisches Bauteil betrachtet, welches durch viele bezüglich der Numerierung gleichstrukturierte Tetraederelemente eingeteilt wird.

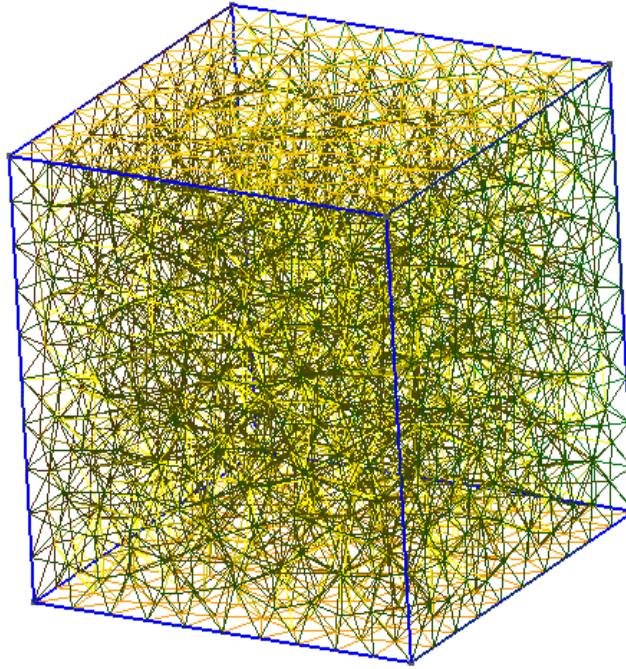


Abbildung 14: Beispiel Geometrie

Es soll für diesen 3-dimensionalen Würfel ein Wärmeleitungsproblem mit Randwertvorgaben

$$-D[\Delta u(\vec{r})] = q(\vec{r}) \Leftrightarrow -D\left[\frac{\partial^2 u(x, y, z)}{\partial x^2} + \frac{\partial^2 u(x, y, z)}{\partial y^2} + \frac{\partial^2 u(x, y, z)}{\partial z^2}\right] = q(x, y, z)$$

und den Parametern im Ansatz $\Rightarrow D = 1, q(x, y, z) = 4$ gelöst werden.

- *Dirichletsche Randbedingung* : $u(s) = \varphi(s)$ auf dem Rand ∂V ist entlang des Bogens s eine Funktion $\varphi(s)$ vorgegeben, also zum Beispiel eine Temperaturbelegung am Rand.
- *Neumannsche Randbedingung* : $\frac{\partial u}{\partial \vec{n}} = 0$, sie wird benutzt, wenn Symmetrien mit einbezogen werden sollen. Gibt es bei einem Bauteil eine Symmetrielinie, dann ist normal zur Randlinie links und rechts davon die Temperatur gleich und somit konstant, also ist die Ableitung senkrecht zum Rand 0.
- *Cauchysche Randbedingung* : Dies ist die allgemeinste Randbedingung, in der die beiden vorherigen zusammen kombiniert werden können $\frac{\partial u}{\partial \vec{n}} + \alpha(s)u = \gamma(s)$.

Hier in dieser Arbeit wird eine *Dirichletsche Randbedingung* verwendet, und die gesuchte Funktion für die Randbedingung wird vorgegeben zu

$$u(x, y, z) = 20 - 2 \cdot y^2 + x^3 \cdot y - x \cdot y^3 + z^3 \cdot x - z \cdot x^3, \quad (1)$$

woraus die obige Vorgabe $q(x, y, z) = 4$ folgt. Damit ist die Aufgabe als Randwertproblem vollständig gestellt. Mit diesem Randwert wird ein lineares Gleichungssystem erzeugt:

$$\underline{\underline{A}} \cdot \vec{x} = \vec{b}$$

Hier:

Die linke Seite $\underline{\underline{A}}$ wird als Steifigkeitsmatrix bezeichnet, die rechte Seite \vec{b} heißt Lastvektor und folgt aus den Randbedingungen, \vec{x} ist der Vektor der unbekannten Temperatur in jeden Punkt des Netzes. Im folgenden Kapitel wird erklärt, wie die Steifigkeitsmatrix zusammengebaut wird und dann die rechte Seite unter Berücksichtigung der Randbedingung entsteht:

$$\underline{\underline{S}} \cdot \vec{u} = \vec{b}$$

Das lineare Gleichungssystem ist durch numerische Verfahren lösbar, wie z.B: die Conjugate–Gradient–Methode, BiCG-Methode usw.

4 Aufbau des Gleichungssystems

4.1 Linearer Ansatz

Es wird ein linearer Ansatz in der Form

$$u(x, y, z) = \sum_i c_i \phi_i = c_1 \phi_1 + c_2 \phi_2 + c_3 \phi_3 + c_4 \phi_4 = c_1 + c_2 x + c_3 y + c_4 z = (c_1, c_2, c_3, c_4) \cdot \begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} = \underline{c} \cdot \vec{r}$$

für u gewählt. Als Grundelement wird das Tetraederelement betrachtet. Ein schiefwinkliges Tetraederelement in kartesischen Koordinaten (x, y, z) wird in den Einheitseraeder im (ξ, η, ζ) -Koordinatensystem transformiert. Damit wird der Ansatz im neuen Koordinatensystem zu

$$u(\xi, \eta, \zeta) = \alpha_1 \cdot 1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \zeta$$

Jedes Tetraeder-Element besitzt 4 Punkte P_1, P_2, P_3, P_4 . Dabei muß beachtet werden, daß für jedes Tetraederelement die Numerierung entgegen dem Uhrzeigersinn erhalten bleibt. Für diese Elementknotennummern werden jetzt die (x_k, y_k, z_k) angegeben.

$$\begin{aligned} P_1 &= (x_1, y_1, z_1) \\ P_2 &= (x_2, y_2, z_2) \\ P_3 &= (x_3, y_3, z_3) \\ P_4 &= (x_4, y_4, z_4) \end{aligned}$$

Hierbei wird als Funktionenbasis folgendes vorgegeben: $\phi_1(\vec{r}) = 1, \phi_2(\vec{r}) = x, \phi_3(\vec{r}) = y, \phi_4(\vec{r}) = z$. Die Koordinatentransformation vom kartesischen Koordinatensystem ins schiefwinklige³ erfolgt durch

$$\vec{r} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} (x_2 - x_1) & (x_3 - x_1) & (x_4 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) & (y_4 - y_1) \\ (z_2 - z_1) & (z_3 - z_1) & (z_4 - z_1) \end{bmatrix} \begin{bmatrix} \xi \\ \eta \\ \zeta \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} (x_2 - x_1)\xi + (x_3 - x_1)\eta + (x_4 - x_1)\zeta \\ (y_2 - y_1)\xi + (y_3 - y_1)\eta + (y_4 - y_1)\zeta \\ (z_2 - z_1)\xi + (z_3 - z_1)\eta + (z_4 - z_1)\zeta \end{bmatrix}$$

Mit Hilfe der partiellen Ableitungen wird die Jacobimatrix ermittelt. Zu diesem Zweck werden die Variablen in der folgenden Form definiert: $x_i - x_j = x_{ij}, y_i - y_j = y_{ij}, z_i - z_j = z_{ij}$
Zuerst wird die Jacobimatrix der Koordinatentransformation ermittelt:

$$\underline{\mathbf{J}} = \begin{bmatrix} x_{21} & y_{21} & z_{41} \\ x_{31} & y_{31} & z_{31} \\ x_{41} & y_{41} & z_{41} \end{bmatrix}$$

Im C++ Programm werden diese wie folgt einprogrammiert:

```

x21=n.P[ n.T[ti+1][1]-1 ][0] - n.P[ n.T[ti+1][0]-1 ][0]; /// \f$x_2-x_1\f$ 
y21=n.P[ n.T[ti+1][1]-1 ][1] - n.P[ n.T[ti+1][0]-1 ][1]; /// \f$y_2-y_1\f$ 
z21=n.P[ n.T[ti+1][1]-1 ][2] - n.P[ n.T[ti+1][0]-1 ][2]; /// \f$z_2-z_1\f$ 

x31=n.P[ n.T[ti+1][2]-1 ][0] - n.P[ n.T[ti+1][0]-1 ][0]; /// \f$x_3-x_1\f$ 
y31=n.P[ n.T[ti+1][2]-1 ][1] - n.P[ n.T[ti+1][0]-1 ][1]; /// \f$y_3-y_1\f$ 
z31=n.P[ n.T[ti+1][2]-1 ][2] - n.P[ n.T[ti+1][0]-1 ][2]; /// \f$z_3-z_1\f$ 

x41=n.P[ n.T[ti+1][3]-1 ][0] - n.P[ n.T[ti+1][0]-1 ][0]; /// \f$x_4-x_1\f$ 
y41=n.P[ n.T[ti+1][3]-1 ][1] - n.P[ n.T[ti+1][0]-1 ][1]; /// \f$y_4-y_1\f$ 
z41=n.P[ n.T[ti+1][3]-1 ][2] - n.P[ n.T[ti+1][0]-1 ][2]; /// \f$z_4-z_1\f$ 
```

³Oder, wenn man einen quadratischen Ansatz wählt ins krummlinige Koordinatensystem.

Und die Determinante der Jakobimatrix folgt zu:

$$\det(\underline{\mathbf{J}}) = x_{21}y_{31}z_{41} + x_{31}y_{41}z_{21} + x_{41}y_{21}z_{31} - x_{41}y_{31}z_{21} - x_{31}y_{21}z_{41} - x_{21}y_{41}z_{31}$$

Was im C++ Programm zu

```
det_Phi=x21*y31*z41+x31*y41*z21+x41*y21*z31 - x41*y31*z21-x31*y21*z41-x21*y41*z31;
```

führt. Danach wird die Inverse der Jacobimatrix ermittelt, da man für die Transformation der Differeniale $\xi = \xi(x, y, z), \eta = \eta(x, y, z), \zeta = \zeta(x, y, z)$ benötigt:

$$\underline{\mathbf{J}}^{-1} = \frac{1}{\det(\underline{\mathbf{J}})} \begin{bmatrix} Pinv_{11} & Pinv_{12} & Pinv_{13} \\ Pinv_{21} & Pinv_{22} & Pinv_{23} \\ Pinv_{31} & Pinv_{32} & Pinv_{33} \end{bmatrix}$$

Hier sind:

$$\begin{aligned} \varphi_{11} &:= Pinv_{11} = y_{31} \cdot z_{41} - y_{41} \cdot z_{31} \\ \varphi_{12} &:= Pinv_{12} = x_{41} \cdot z_{31} - x_{31} \cdot z_{41} \\ \varphi_{13} &:= Pinv_{13} = x_{31} \cdot y_{41} - x_{41} \cdot y_{31} \\ \varphi_{21} &:= Pinv_{21} = y_{41} \cdot z_{21} - y_{21} \cdot z_{41} \\ \varphi_{22} &:= Pinv_{22} = x_{21} \cdot z_{41} - x_{41} \cdot z_{21} \\ \varphi_{23} &:= Pinv_{23} = x_{41} \cdot y_{21} - x_{21} \cdot y_{41} \\ \varphi_{31} &:= Pinv_{31} = y_{21} \cdot z_{31} - y_{31} \cdot z_{21} \\ \varphi_{32} &:= Pinv_{32} = x_{31} \cdot z_{21} - x_{21} \cdot z_{31} \\ \varphi_{33} &:= Pinv_{33} = x_{21} \cdot y_{31} - x_{31} \cdot y_{21} \end{aligned}$$

Im C++ Programm:

```
one_by_Phi = 1.0 / det_Phi;
Pinv11 = (y31 * z41 - y41 * z31);
Pinv12 = (x41 * z31 - x31 * z41);
Pinv13 = (x31 * y41 - x41 * y31);
Pinv21 = (y41 * z21 - y21 * z41);
Pinv22 = (x21 * z41 - x41 * z21);
Pinv23 = (x41 * y21 - x21 * y41);
Pinv31 = (y21 * z31 - y31 * z21);
Pinv32 = (x31 * z21 - x21 * z31);
Pinv33 = (x21 * y31 - x31 * y21);
```

Der lineare Ansatz für die gesuchte Lösungsfunktion wird eingesetzt mit :

$$\begin{aligned} u(x, y, z) &= c_1\phi_1 + c_2\phi_2 + c_3\phi_3 + c_4\phi_4 = c_1 + c_2x + c_3y + c_4z \\ u(\xi, \eta, \zeta) &= \alpha_1 \cdot 1 + \alpha_2\xi + \alpha_3\eta + \alpha_4\zeta \end{aligned}$$

und wegen der Koordinatentransformation $\xi = \xi(x, y, z), \eta = \eta(x, y, z), \zeta = \zeta(x, y, z)$ und der damit zu berücksichtigenden Kettenregel folgt $u_\xi = \frac{\partial u(\xi, \eta, \zeta)}{\partial \xi} = \alpha_2$, $u_\eta = \frac{\partial u(\xi, \eta, \zeta)}{\partial \eta} = \alpha_3$, $u_\zeta = \frac{\partial u(\xi, \eta, \zeta)}{\partial \zeta} = \alpha_4$

Die später in Steifigkeitsmatrix benötigten Transformationen $(x, y, z) \rightarrow (\xi, \eta, \zeta)$ der Ableitungsoperatoren ergeben sich somit zu:

$$\begin{aligned} \frac{\partial u(x, y, z)}{\partial x} &= \frac{\partial u(\xi, \eta, \zeta)}{\partial \xi} \cdot \frac{\partial u(x, y, z)}{\partial x} + \frac{\partial u(\xi, \eta, \zeta)}{\partial \eta} \cdot \frac{\partial u(x, y, z)}{\partial x} + \frac{\partial u(\xi, \eta, \zeta)}{\partial \zeta} \cdot \frac{\partial u(x, y, z)}{\partial x} \\ &= \color{red}{\alpha_2} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{11} + \color{blue}{\alpha_3} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{21} + \color{green}{\alpha_4} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{31} \\ \frac{\partial u(x, y, z)}{\partial y} &= \frac{\partial u(\xi, \eta, \zeta)}{\partial \xi} \cdot \frac{\partial u(x, y, z)}{\partial y} + \frac{\partial u(\xi, \eta, \zeta)}{\partial \eta} \cdot \frac{\partial u(x, y, z)}{\partial y} + \frac{\partial u(\xi, \eta, \zeta)}{\partial \zeta} \cdot \frac{\partial u(x, y, z)}{\partial y} \\ &= \color{red}{\alpha_2} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{12} + \color{blue}{\alpha_3} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{22} + \color{green}{\alpha_4} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{32} \\ \frac{\partial u(x, y, z)}{\partial z} &= \frac{\partial u(\xi, \eta, \zeta)}{\partial \xi} \cdot \frac{\partial u(x, y, z)}{\partial z} + \frac{\partial u(\xi, \eta, \zeta)}{\partial \eta} \cdot \frac{\partial u(x, y, z)}{\partial z} + \frac{\partial u(\xi, \eta, \zeta)}{\partial \zeta} \cdot \frac{\partial u(x, y, z)}{\partial z} \\ &= \color{red}{\alpha_2} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{13} + \color{blue}{\alpha_3} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{23} + \color{green}{\alpha_4} \cdot \frac{1}{\det(\underline{\mathbf{J}})} \varphi_{33} \end{aligned}$$

Damit man nicht für jedes Element die Integrale berechnen muß, wird die Koordinatentransformation durchgeführt und einmal integriert und dieses Integrationsergebnis dann jedes lokale Element des globalen Netzes zurücktransformiert. Damit ist nur noch die Intergration über das Einheitstetraederelement erforderlich.

4.2 Intergration für das Tetraeder

Die Integration des Einheitstetraeder wird über die Funktion $1 - \xi$ und $1 - \xi - \eta$ berechnet:

$$\begin{aligned} \text{Integration} &= \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} 1 d\zeta d\eta d\xi \\ &= \int_0^1 \int_0^{1-\xi} 1 - \xi - \eta d\eta d\xi \\ &= \int_0^1 1 - \xi - \xi(1 - \xi) - 0.5(1 - \xi)^2 d\xi \\ &= \int_0^1 0.5 - \xi + 0.5\xi^2 d\xi \\ &= \frac{1}{6} \end{aligned}$$

4.3 Lokale Steifigkeitsmatrix

Die stationäre Wärmeleitungsgleichung lautet :

$$-D\vec{\nabla}_{\vec{r}} \cdot [\vec{\nabla}_{\vec{r}} u(\vec{r})] = q(\vec{r}) \Leftrightarrow -D\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}\right) u(x, y, z) = q(x, y, z). \quad (2)$$

Multiplikation dieser Gleichung mit einer Testfunktion und dann die Ausführung einer Volumenintegration führen auf die Form, die es ermöglicht sie Steifigkeitsmatrix zu identifizieren. Die Anwendung des Gauß'schen-Satzes⁴ führt auf die Integralgleichung der Wärmeleitungsgleichung :

$$\int_V D \left\{ \left(\frac{\partial u(\vec{r})}{\partial x} \right)^2 + \left(\frac{\partial u(\vec{r})}{\partial y} \right)^2 + \left(\frac{\partial u(\vec{r})}{\partial z} \right)^2 \right\} - q(\vec{r}) \cdot u(\vec{r}) d^3 \vec{r} = 0$$

Daraus folgt die Form der Steifigkeitsmatrix, die es zu berechnen gilt :

$$S_{ji} = D \left\{ \int_V \vec{\nabla}_{\vec{r}} \phi_j(\vec{r}) \cdot \vec{\nabla}_{\vec{r}} \phi_i(\vec{r}) d^3 \vec{r} + \oint_{\partial V} \phi_j(\vec{r}) \alpha(s) \phi_i(\vec{r}) ds - \oint_{\partial V} \phi_j(\vec{r}) \gamma(s) ds \right\}$$

Der Rand teilt sich normalerweise in mehrere Teile ein, für die unterschiedliche Randbedingungen gegeben sein können. Es gilt $\partial V = \partial V_1 + \partial V_2$. Hier werden ausschließlich Dirichletsche-Randbedingungen vorgegeben, so daß $\alpha(s) = 0$ gesetzt wird, da keine Neumannsche-Randbedingung berücksichtigt werden sollen. Mit der Dirichletschen-Randbedingung ist dann nur $\gamma(s)$ vorzugeben, was mit Gl. (1) auch durchgeführt wird. Die Steifigkeitsmatrix dann mit

$$S_{ji} = D \int_V \vec{\nabla}_{\vec{r}} \phi_j(\vec{r}) \cdot \vec{\nabla}_{\vec{r}} \phi_i(\vec{r}) d^3 \vec{r}$$

berechnet werden, was dann zur Definition

$$S_{ji} = \left(\frac{\partial \phi_j(x, y, z)}{\partial x} \right)^2 + \left(\frac{\partial \phi_j(x, y, z)}{\partial y} \right)^2 + \left(\frac{\partial \phi_j(x, y, z)}{\partial z} \right)^2$$

Führt. Die einzusetzenden transformierten Differentiale aus Kapitel 4.1 ergeben dann :

$$\frac{\partial \phi_j(x, y, z)}{\partial x}^2 = \frac{1}{\det(\underline{\underline{\mathbf{J}}})^2} \cdot (\color{red}{\alpha_2^2} \varphi_{11}^2 + \color{blue}{\alpha_3^2} \varphi_{21}^2 + \color{green}{\alpha_4^2} \varphi_{31}^2 + \color{magenta}{2\alpha_2\alpha_3} \varphi_{11}\varphi_{21} + \color{cyan}{2\alpha_2\alpha_4} \varphi_{11} \varphi_{31} + \color{yellow}{2\alpha_3\alpha_4} \varphi_{21}\varphi_{31})$$

⁴Das ist die mehrdimensionale partielle Integrationsregel.

$$\frac{\partial \phi_j(x, y, z)}{\partial y}^2 = \frac{1}{\det(\underline{\underline{\mathbf{J}}})^2} \cdot (\alpha_2^2 \varphi_{12}^2 + \alpha_3^2 \varphi_{22}^2 + \alpha_4^2 \varphi_{32}^2 + 2\alpha_2 \alpha_3 \varphi_{12} \varphi_{22} + 2\alpha_2 \alpha_4 \varphi_{12} \varphi_{32} + 2\alpha_3 \alpha_4 \varphi_{22} \varphi_{32})$$

$$\frac{\partial \phi_j(x, y, z)}{\partial z}^2 = \frac{1}{\det(\underline{\underline{\mathbf{J}}})^2} \cdot (\alpha_2^2 \varphi_{13}^2 + \alpha_3^2 \varphi_{23}^2 + \alpha_4^2 \varphi_{33}^2 + 2\alpha_2 \alpha_3 \varphi_{13} \varphi_{23} + 2\alpha_2 \alpha_4 \varphi_{13} \varphi_{33} + 2\alpha_3 \alpha_4 \varphi_{23} \varphi_{33})$$

Um die Berechnung der lokalen Steifigkeitsmatrix zu vereinfachen werden folgende Variablendefinitionen (a, b, c, d, e, f) gewählt:

$$a = \frac{(y_{31}z_{41} - y_{41}z_{31})^2 + (x_{41}z_{31} - x_{31}z_{41})^2 + (x_{31}y_{41} - x_{41}y_{31})^2}{\det(\underline{\underline{\mathbf{J}}})}$$

$$b = \frac{(y_{41}z_{21} - y_{21}z_{41})^2 + (x_{21}z_{41} - x_{41}z_{21})^2 + (x_{41}y_{21} - x_{21}y_{41})^2}{\det(\underline{\underline{\mathbf{J}}})}$$

$$c = \frac{(y_{31}z_{41} - y_{41}z_{31})^2 + (x_{41}z_{31} - x_{31}z_{41})^2 + (x_{31}y_{41} - x_{41}y_{31})^2}{\det(\underline{\underline{\mathbf{J}}})}$$

$$d = \frac{(y_{31}z_{41} - y_{41}z_{31}) \cdot (y_{41}z_{21} - y_{21}z_{41}) + (x_{41}z_{31} - x_{31}z_{41}) \cdot (x_{21}z_{41} - x_{41}z_{21}) + (x_{31}y_{41} - x_{41}y_{31}) \cdot (x_{41}y_{21} - x_{21}y_{41})}{\det(\underline{\underline{\mathbf{J}}})}$$

$$e = \frac{(y_{31}z_{41} - y_{41}z_{31}) \cdot (y_{21}z_{31} - y_{31}z_{21}) + (x_{41}z_{31} - x_{31}z_{41}) \cdot (x_{31}z_{21} - x_{21}z_{31}) + (x_{31}y_{41} - x_{41}y_{31}) \cdot (x_{21}y_{31} - x_{31}y_{21})}{\det(\underline{\underline{\mathbf{J}}})}$$

$$f = \frac{(y_{31}z_{41} - y_{41}z_{31}) \cdot (y_{41}z_{21} - y_{21}z_{41}) + (x_{41}z_{31} - x_{31}z_{41}) \cdot (x_{21}z_{41} - x_{41}z_{21}) + (x_{31}y_{41} - x_{41}y_{31}) \cdot (x_{41}y_{21} - x_{21}y_{41})}{\det(\underline{\underline{\mathbf{J}}})}$$

Hier jetzt die genauen Definitionen dieser Variablen :

$$a = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{11}^2 + Pinv_{12}^2 + Pinv_{13}^2)$$

$$b = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{21}^2 + Pinv_{22}^2 + Pinv_{23}^2)$$

$$c = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{31}^2 + Pinv_{32}^2 + Pinv_{33}^2)$$

$$d = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{11}Pinv_{21} + Pinv_{12}Pinv_{22} + Pinv_{13}Pinv_{23})$$

$$e = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{11}Pinv_{31} + Pinv_{12}Pinv_{32} + Pinv_{13}Pinv_{33})$$

$$f = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{21}Pinv_{31} + Pinv_{22}Pinv_{32} + Pinv_{23}Pinv_{33})$$

Insgesamt erhält man das Zwischenergebniss mit den abkürzenden Definitionen:

$$I = \frac{\alpha_2^2}{6} \cdot a + \frac{\alpha_3^2}{6} \cdot b + \frac{\alpha_4^2}{6} \cdot c + \frac{2\alpha_2 \alpha_3}{6} \cdot d + \frac{2\alpha_2 \alpha_4}{6} \cdot e + \frac{2\alpha_3 \alpha_4}{6} \cdot f$$

$$\begin{aligned}
& \Rightarrow (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \cdot \frac{a}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} + (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \cdot \frac{b}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} \\
& + (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \cdot \frac{c}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} + (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \cdot \frac{d}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} \\
& + (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \cdot \frac{e}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} + (\alpha_1, \alpha_2, \alpha_3, \alpha_4) \cdot \frac{f}{6} \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix}
\end{aligned}$$

Mit Hilfe dieser Matrix wird die Steifigkeitsmatrix für jedes lokale Element aufgestellt werden können.

$$\begin{aligned}
\underline{\underline{S}}_1' &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \underline{\underline{S}}_2' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \underline{\underline{S}}_3' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
\underline{\underline{S}}_4' &= \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \underline{\underline{S}}_5' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad \underline{\underline{S}}_6' = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}
\end{aligned}$$

Das Koordinatensystem ist jetzt noch im lokalen Einheitskoordinatensystem(ξ, η, ζ). Da die Lösungsfunktion aus benachbarten finiten Elementen linear sind, und am Knoten denselben Wert haben, folgt damit zwingend, dass sie einen stetigen Übergang an den benachbarten Tetraederkanten heben. Wegen die Interpolationsbedingungen muss $u_i = (x_i, y_i, z_i) = 1$ gelten, falls $i = k$ in $P_k(x_k, y_k, z_k)$ gleich der Knotennummer im Einheitstetraeder ist. Daher muss noch ins globale Koordinatensystem(x, y, z) rücktransformiert werden.

$$\begin{aligned}
u_1(0,0,0) &= \alpha_1 & \Rightarrow \alpha_1 &= u_1 \\
u_2(1,0,0) &= \alpha_1 + \alpha_2 & \Rightarrow \alpha_2 &= -u_1 + u_2 \\
u_3(0,1,0) &= \alpha_1 + 0 + \alpha_3 & \Rightarrow \alpha_3 &= -u_1 + u_3 \\
u_4(0,0,1) &= \alpha_1 + 0 + 0 + \alpha_4 & \Rightarrow \alpha_4 &= -u_1 + u_4
\end{aligned}$$

Dies in Matrixschreibweise, um $\vec{\alpha} = \underline{\underline{A}} \cdot \vec{\mathbf{u}}$ definieren zu können

$$\begin{aligned}
\vec{\mathbf{u}} &= \underline{\underline{A}}^{-1} \cdot \vec{\alpha} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} \\
&\Rightarrow \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \vec{\alpha} = \underline{\underline{A}} \cdot \vec{\mathbf{u}}
\end{aligned}$$

Die Integrale sind jetzt in Abhängigkeit von den Knotenvariablen u_i darzustellen.

$$I = \vec{\alpha}^T (a\underline{\underline{S}}_1' + b\underline{\underline{S}}_2' + c\underline{\underline{S}}_3' + d\underline{\underline{S}}_4' + e\underline{\underline{S}}_5' + f\underline{\underline{S}}_6') \vec{\alpha} = (\underline{\underline{A}} \cdot \vec{\mathbf{u}})^T \cdot (a\underline{\underline{S}}_1' + b\underline{\underline{S}}_2' + c\underline{\underline{S}}_3' + d\underline{\underline{S}}_4' + e\underline{\underline{S}}_5' + f\underline{\underline{S}}_6') \cdot \underline{\underline{A}} \cdot \vec{\mathbf{u}}$$

$$\Rightarrow \vec{\mathbf{u}}^T \cdot \underline{\underline{A}} \cdot (a\underline{\underline{S}}_1' + b\underline{\underline{S}}_2' + c\underline{\underline{S}}_3' + d\underline{\underline{S}}_4' + e\underline{\underline{S}}_5' + f\underline{\underline{S}}_6') \cdot \underline{\underline{A}} \cdot \vec{\mathbf{u}}$$

Die lokale Steifigkeitsmatrix pro Element lautet somit:

$$\begin{aligned}
&= \underline{\underline{\mathbf{A}}}^T \{a\underline{\underline{S}}_1' + b\underline{\underline{S}}_2' + c\underline{\underline{S}}_3' + d\underline{\underline{S}}_4' + e\underline{\underline{S}}_5' + f\underline{\underline{S}}_6'\} \underline{\underline{\mathbf{A}}} \\
&= \frac{a}{6} \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \frac{b}{6} \begin{pmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \frac{c}{6} \begin{pmatrix} 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \\
&+ \frac{d}{6} \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \frac{e}{6} \begin{pmatrix} 2 & -1 & 0 & -1 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 \end{pmatrix} + \frac{f}{6} \begin{pmatrix} 2 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 \end{pmatrix} \\
&= \{a\underline{\underline{S}}_1 + b\underline{\underline{S}}_2 + c\underline{\underline{S}}_3 + d\underline{\underline{S}}_4 + e\underline{\underline{S}}_5 + f\underline{\underline{S}}_6\}
\end{aligned}$$

Damit wird nun das Integral für das e-te Element festgestellt:

$$\begin{aligned}
I^{(e)} &= \int_{V^{(e)}} u_x^2 + u_y^2 + u_z^2 dx dy dz = (\vec{\mathbf{u}}^{(e)})^T \cdot \{a\underline{\underline{S}}_1' + b\underline{\underline{S}}_2' + c\underline{\underline{S}}_3' + d\underline{\underline{S}}_4' + e\underline{\underline{S}}_5' + f\underline{\underline{S}}_6'\} \cdot \vec{\mathbf{u}}^{(e)} \\
&= (\vec{\mathbf{u}}^{(e)})^T \cdot \frac{1}{6} \begin{pmatrix} a+b+c+2d+2e+2f & -a-d-e & -b-d-f & -c-e-f \\ -a-e-d & a & d & e \\ -b-d-f & d & b & f \\ -c-e-f & e & f & c \end{pmatrix} \vec{\mathbf{u}}^{(e)}
\end{aligned}$$

Die obere Herleitung wird in der folgenden Tabellen zusammengefasst:

Tabelle 1: Variablen der lokalen Steifigkeitsmatrix

Elementknoten im lokalen Koord.
$P_1 = (x_1, y_1, z_1)$
$P_2 = (x_2, y_2, z_2)$
$P_3 = (x_3, y_3, z_3)$
$P_4 = (x_4, y_4, z_4)$
$\det(\underline{\underline{\mathbf{J}}}) = x_{21}y_{31}z_{41} + x_{31}y_{41}z_{21} + x_{41}y_{21}z_{31} - x_{41}y_{31}z_{21} - x_{31}y_{21}z_{41} - x_{21}y_{41}z_{31}$
$a = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{11}^2 + Pinv_{12}^2 + Pinv_{13}^2)$
$b = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{21}^2 + Pinv_{22}^2 + Pinv_{23}^2)$
$c = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{31}^2 + Pinv_{32}^2 + Pinv_{33}^2)$
$d = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{11}Pinv_{21} + Pinv_{12}P_{22} + Pinv_{13}Pinv_{23})$
$e = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{11}Pinv_{31} + Pinv_{12}P_{32} + Pinv_{13}Pinv_{33})$
$f = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} (Pinv_{21}Pinv_{31} + Pinv_{22}P_{32} + Pinv_{23}Pinv_{33})$

Im C++ Programm:

```

one_by_Phi = 1.0 / det_Phi;
Pinv11 = (y31 * z41 - y41 * z31);
Pinv12 = (x41 * z31 - x31 * z41);
Pinv13 = (x31 * y41 - x41 * y31);
Pinv21 = (y41 * z21 - y21 * z41);
Pinv22 = (x21 * z41 - x41 * z21);
Pinv23 = (x41 * y21 - x21 * y41);
Pinv31 = (y21 * z31 - y31 * z21);
Pinv32 = (x31 * z21 - x21 * z31);
Pinv33 = (x21 * y31 - x31 * y21);
a = Pinv11*Pinv11 + Pinv12*Pinv12 + Pinv13*Pinv13;
b = Pinv21*Pinv21 + Pinv22*Pinv22 + Pinv23*Pinv23;
c = Pinv31*Pinv31 + Pinv32*Pinv32 + Pinv33*Pinv33;
d = Pinv11*Pinv21 + Pinv12*Pinv22 + Pinv13*Pinv23;
e = Pinv11*Pinv31 + Pinv12*Pinv32 + Pinv13*Pinv33;
f = Pinv21*Pinv31 + Pinv22*Pinv32 + Pinv23*Pinv33;

```

Jede lokale Steifigkeitsmatrix wird mit den Variablen a, b, c, d, e, f berechnet:

$$\underline{\underline{S}}_{\text{lokal}} = \frac{1}{\det(\underline{\underline{\mathbf{J}}})} \begin{bmatrix} a + b + c + 2 \cdot (d + e + f) & -a - d - e & -b - d - f & -c - e - f \\ -a - d - e & a & d & e \\ -b - d - f & d & b & f \\ -c - e - f & e & f & c \end{bmatrix}$$

Im C++ Programm:

```
Slocal[0][0]=a+b+c+2.0*(d+e+f);
Slocal[0][1]=-a-d-e;
Slocal[0][2]=-b-d-f;
Slocal[0][3]=-c-e-f;
Slocal[1][0]=Slocal[0][1];
Slocal[2][0]=Slocal[0][2];
Slocal[3][0]=Slocal[0][3];
Slocal[1][1]=a;
Slocal[1][2]=d;
Slocal[1][3]=e;
Slocal[2][1]=d;
Slocal[2][2]=b;
Slocal[2][3]=f;
Slocal[3][1]=e;
Slocal[3][2]=f;
Slocal[3][3]=c;
for(int i=0;i<4;i++){
    for(int j=0;j<4;j++){
        Slocal[i][j]=Slocal[i][j]*one_by_Phi; // Faktor 1/det multiplizieren
    }
}
```

4.4 Lokale rechte Seite \vec{b}

Die rechte Seite folgt dann aus der Wärmeleitungsgleichung zu:

$$\int_{V^{(e)}} q(\vec{r}) \cdot u(\vec{r}) d^3 r = \int_{V^{(e)}} q(\alpha_1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \zeta) \det(\underline{\underline{\mathbf{J}}}) d\zeta d\eta d\xi$$

$$\Rightarrow b_i = \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} \phi_j(\xi, \eta, \zeta) \cdot q(\xi, \eta, \zeta) \cdot \det(\underline{\underline{\mathbf{J}}}) d\zeta d\eta d\xi \rightarrow (i = 1, 2, 3, 4)$$

Hier ist $q(\xi, \eta, \zeta)$ wegen der Randwertvorgabe in Gl. (1) als eine Konstante vorzugeben und gleich 4:

$$\begin{aligned}
b_1 &= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} 1 - \xi - \eta - \zeta d\zeta d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \frac{1}{2} (1 - \xi - \eta)^2 d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \frac{1}{6} (1 - \xi)^3 d\xi \\
&= \frac{\det(\underline{\underline{\mathbf{J}}})}{6}
\end{aligned}$$

$$\begin{aligned}
b_2 &= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} \xi d\zeta d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \xi - \xi^2 - \xi\eta d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \frac{1}{2} \xi (1 - \xi)^2 d\xi \\
&= \frac{\det(\underline{\underline{\mathbf{J}}})}{6}
\end{aligned}$$

$$\begin{aligned}
b_3 &= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} \eta d\zeta d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \eta - \xi\eta - \eta^2 d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \frac{1}{6} (1 - \xi)^3 d\xi \\
&= \frac{\det(\underline{\underline{\mathbf{J}}})}{6}
\end{aligned}$$

$$\begin{aligned}
b_4 &= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \int_0^{1-\xi-\eta} \zeta d\zeta d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \int_0^{1-\xi} \frac{1}{2} (1 - \xi - \eta)^2 d\eta d\xi \\
&= 4 \cdot \det(\underline{\underline{\mathbf{J}}}) \int_0^1 \frac{1}{6} (1 - \xi)^3 d\xi \\
&= \frac{\det(\underline{\underline{\mathbf{J}}})}{6}
\end{aligned}$$

Daher folgt das Ergebnis für die lokale rechte Seite $\vec{\mathbf{b}}$

$$\vec{\mathbf{b}}_{lokal} = \frac{\det(\underline{\underline{\mathbf{J}}})}{6} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

4.5 Beispiel Geometrie

Zur Aufstellung der Gesamtsteifigkeitsmatrix ($\in \mathbb{R}^{n \cdot n}$ n ist Anzahl aller Punkten) werden die jeweiligen lokalen Steifigkeitsmatrizen zur bisherigen Gesamtsteifigkeitsmatrix dazuaddiert, und zwar genau an

den Addressen, die das Feld mit den globalen Knotennummern jedem Tetraederelement zuordnet. Die Steifigkeitsmatrix ist ein symmetrisches Matrix sowohl lokal wie auch global.

In der folgenden Abbildung (15) wird eine einfache Geometrie als Beispiel gezeigt und anhand dieser einfachen Anordnung wird vorgeführt wie die Gesamtsteifigkeitsmatrix berechnet wird.

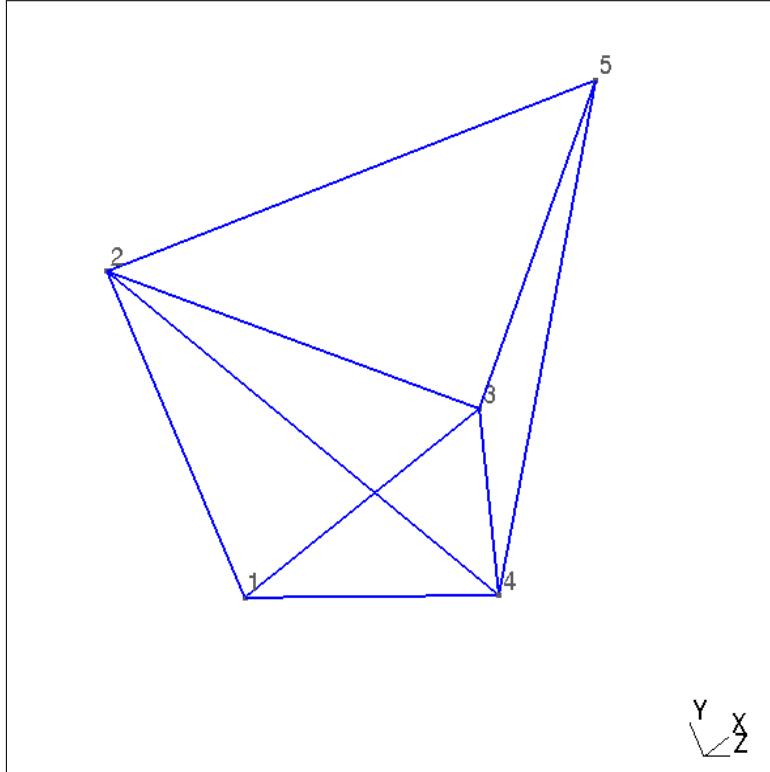


Abbildung 15: Kubische Geometrie mit 2 finiten Tetraederelementen mit Elementnumerierung und Knotennumerierung

4.6 Die Globale Steifigkeitsmatrix

Diese Geometrie besteht aus 2 Tetraederelementen (5 Knoten), deswegen besitzt diese Struktur zwei lokale Steifigkeitsmatrizen:

- Tetraederelement 1: Knoten 1, 2, 3, 4
- Tetraederelement 2: Knoten 2, 3, 4, 5

$$\text{das lokale Matrixelement 1: } \begin{bmatrix} s_{11}^{(l)} & s_{12}^{(l)} & s_{13}^{(l)} & s_{14}^{(l)} \\ s_{21}^{(l)} & s_{22}^{(l)} & s_{23}^{(l)} & s_{24}^{(l)} \\ s_{31}^{(l)} & s_{32}^{(l)} & s_{33}^{(l)} & s_{34}^{(l)} \\ s_{41}^{(l)} & s_{42}^{(l)} & s_{43}^{(l)} & s_{44}^{(l)} \end{bmatrix} \text{ an die globale Adresse: } \begin{bmatrix} s_{11}^{(1)} & s_{12}^{(1)} & s_{13}^{(1)} & s_{14}^{(1)} \\ s_{21}^{(1)} & s_{22}^{(1)} & s_{23}^{(1)} & s_{24}^{(1)} \\ s_{31}^{(1)} & s_{32}^{(1)} & s_{33}^{(1)} & s_{34}^{(1)} \\ s_{41}^{(1)} & s_{42}^{(1)} & s_{43}^{(1)} & s_{44}^{(1)} \end{bmatrix}$$

$$\text{das lokale Matrixelement 2: } \begin{bmatrix} s_{11}^{(l)} & s_{12}^{(l)} & s_{13}^{(l)} & s_{14}^{(l)} \\ s_{21}^{(l)} & s_{22}^{(l)} & s_{23}^{(l)} & s_{24}^{(l)} \\ s_{31}^{(l)} & s_{32}^{(l)} & s_{33}^{(l)} & s_{34}^{(l)} \\ s_{41}^{(l)} & s_{42}^{(l)} & s_{43}^{(l)} & s_{44}^{(l)} \end{bmatrix} \text{ an die globale Adresse: } \begin{bmatrix} s_{22}^{(2)} & s_{23}^{(2)} & s_{24}^{(2)} & s_{25}^{(2)} \\ s_{32}^{(2)} & s_{33}^{(2)} & s_{34}^{(2)} & s_{35}^{(2)} \\ s_{42}^{(2)} & s_{43}^{(2)} & s_{44}^{(2)} & s_{45}^{(2)} \\ s_{52}^{(2)} & s_{53}^{(2)} & s_{54}^{(2)} & s_{55}^{(2)} \end{bmatrix}$$

Die globale Steifigkeitsmatrix assembliert sich in der folgenden Weise aus den einzelnen lokalen

$$\text{Steifigkeitsmatrizen: } \begin{bmatrix} s_{11}^{(1)} & s_{12}^{(1)} & s_{13}^{(1)} & s_{14}^{(1)} & 0 \\ s_{21}^{(1)} & s_{22}^{(1)} & s_{23}^{(1)} & s_{24}^{(1)} & 0 \\ s_{31}^{(1)} & s_{32}^{(1)} & s_{33}^{(1)} & s_{34}^{(1)} & 0 \\ s_{41}^{(1)} & s_{42}^{(1)} & s_{43}^{(1)} & s_{44}^{(1)} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & s_{22}^{(2)} & s_{23}^{(2)} & s_{24}^{(2)} & s_{25}^{(2)} \\ 0 & s_{32}^{(2)} & s_{33}^{(2)} & s_{34}^{(2)} & s_{35}^{(2)} \\ 0 & s_{42}^{(2)} & s_{43}^{(2)} & s_{44}^{(2)} & s_{45}^{(2)} \\ 0 & s_{52}^{(2)} & s_{53}^{(2)} & s_{54}^{(2)} & s_{55}^{(2)} \end{bmatrix} =$$

$$\begin{bmatrix} s_{11}^{(1)} & & & & \\ s_{21}^{(1)} & s_{22}^{(1)} + s_{11}^{(2)} & & & \\ s_{31}^{(1)} & s_{32}^{(1)} + s_{21}^{(2)} & s_{33}^{(1)} + s_{22}^{(2)} & & \\ s_{41}^{(1)} & s_{42}^{(1)} + s_{31}^{(2)} & s_{43}^{(1)} + s_{32}^{(2)} & s_{44}^{(1)} + s_{33}^{(2)} & \\ 0 & s_{41}^{(2)} & s_{42}^{(2)} & s_{43}^{(2)} & s_{44}^{(2)} \end{bmatrix}$$

Hier ist nur der erzeugende Teil der Matrix dargestellt. Der restliche Teil wird durch Symmetrioperatoren $s_{rl} = s_{lr}$ generiert.

4.7 Die globale rechte Seite \vec{b}

Die rechte Seite \vec{b} berechnet sich übrigens mit denselben Argumenten:

$$\text{das lokale rechte Seite 1: } \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ b_3^{(l)} \\ b_4^{(l)} \end{bmatrix} \text{ an die globale Adresse: } \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \\ b_4^{(1)} \end{bmatrix}$$

$$\text{das lokale rechte Seite 2: } \begin{bmatrix} b_1^{(l)} \\ b_2^{(l)} \\ b_3^{(l)} \\ b_4^{(l)} \end{bmatrix} \text{ an die globale Adresse: } \begin{bmatrix} b_2^{(2)} \\ b_3^{(2)} \\ b_4^{(2)} \\ b_5^{(2)} \end{bmatrix}$$

Die globale rechte Seite assembliert sich in der folgenden Weise aus der einzelnen Lokale:

$$\begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} + b_1^{(2)} \\ b_3^{(1)} + b_2^{(2)} \\ b_4^{(1)} + b_3^{(2)} \\ b_4^{(2)} \end{bmatrix}$$

Im C++ Program:

```
for(int i=0;i<4;i++){
blocal[i]=1.0;
}
for(int i=0;i<4;i++){
l=n.T[ti+1][i]-1;
for(int j=0;j<i+1;j++){
r=n.T[ti+1][j]-1;
g.Sglobal[1][r]=g.Sglobal[1][r]+Slocal[i][j];
g.Sglobal[r][1]=g.Sglobal[1][r];
}
g.bglobal[1]=g.bglobal[1]+det_Phi*blocal[i];
}
```

Dieses Gleichungssystem ist nicht die ganze Wahrheit. Man muss noch den Faktor von der Tetraeder-Integration (der Faktor ist gleich $\frac{1}{6}$) betrachten. Die rechte Seite muss multipliziert mit $\frac{q}{24}$ und $\frac{1}{6}$ auf der Steifigkeitsmatrix.

Im C++ Programm:

```

q=4.0;
for(int i=0;i<Ndim;i++){
g.bglobal[i]=q/24.0*g.bglobal[i];
}
for(int i=0;i<Ndim;i++){
for(int j=0;j<Ndim;j++){
g.Sglobal[i][j]=1.0/6.0*g.Sglobal[i][j];
}
}

```

4.8 Gleichungssystem mit Randbedingung

Bis jetzt wurden die Randbedingungen noch nicht berücksichtigt. Die Randwerte werden in folgender Weise in der linken Seite verarbeitet:

1. Ist für die k -te Knotenvariable die Temperatur (der Randwert) u_k vorgegeben, so wird das u_k -fache der k -ten Spalte von \underline{S} vom Lastvektor \vec{b} abgezogen.
2. Dann wird die k -te Zeile und Spalte von \underline{S} mit 0 belegt,
3. das k -te Diagonalelement von \underline{S} mit 1 und
4. das k -te Komponente von \vec{b} gleich u_k gesetzt.

Im C++ Programm:

```

x=n.P[ki-1][0];
y=n.P[ki-1][1];
z=n.P[ki-1][2];
/// kubische Lösung: \n
/// f$urand=20-2*y^2+x^3*y-x*y^3+z^3*x-z*x^3\f$
urand=20.0-2.0*y*y+pow(x,3)*y-x*pow(y,3)+pow(z,3)*x-z*pow(x,3);
for(int j=0;j<n.dim.n_tetr_pkt;j++){
/// Hier werden alle Komponenten belegt!
g.bglobal[j]=g.bglobal[j]-g.Sglobal[j][ki-1]*urand;
g.Sglobal[j][ki-1]=0.0;
g.Sglobal[ki-1][j]=0.0;
}
g.Sglobal[ki-1][ki-1]=1.0;

```

Wegen Schritt 4 in der Aufzählung werden offensichtlich alle Knoten-Komponenten verändert, die die k Knoten nicht enthalten. Daher muss \vec{b} am Rand noch zurückgesetzt werden.

Im C++ Programm:

```

x=n.P[ki-1][0];
y=n.P[ki-1][1];
z=n.P[ki-1][2];
urand=20.0-2.0*y*y+pow(x,3)*y-x*pow(y,3)+pow(z,3)*x-z*pow(x,3);
/// Hier wird nur die ki-te Komponente belegt!
g.bglobal[ki-1]=urand;

```

Jetzt ist das Gleichungssystem $\underline{S} \cdot \vec{u} = \vec{b}$ fertig assembliert, danach wird das BiCG-Verfahren verwendet, um es zu lösen.

5 Gleichungssystem Optimierung

5.1 Warum optimiert man das Gleichungssystem ?

Das Gleichungssystem ist jetzt fertig assembled: $\underline{S} \cdot \vec{u} = \vec{b}$. Das Ergebnis zeigt die Temperatur an den Knoten. Einige Knoten sind auf dem Rand und die anderen liegen im inneren der Geometrie, d.h. die Lösungen an den Randpunkten sind schon bekannt wegen der vorgegebenen *Dirichletsche Randbedingung*, also u_{rand} sind bekannt. Die Randpunkte besetzen einen großen Teil im ganzen linearen Gleichungssystem, aber das Programm braucht nur die Lösungen der inneren Punkten zu berechnen. Daher ist dieses Gleichungssystem optimierbar, um die Rechenzeit zu verkürzen und den verbrauchten Hauptspeicherplatz zu verkleinern.

5.2 Optimierung

Hier ein Beispiel, dass die Dimension 4 hat (für eine 4×4 -Steifigkeitsmatrix, für den Lösungsvektor \vec{u} und die rechte Seite \vec{b} 1×4). Die roten Elemente sind die Randpunkte und die schwarzen sind die inneren Punkte.

$$\begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Wegen der bekannten Randtemperatur u_1 und u_2 kann man alle roten Elemente weg löschen. Die verkürzte Dimension ist 2 (für Steifigkeitsmatrix 2×2 , für den Lösungsvektor \vec{u} und die rechte Seite \vec{b} 1×2), die rechte Seite \vec{b} muss neu berechnet werden:

$$\vec{b}_{neu} = \begin{bmatrix} b_3^{(neu)} \\ b_4^{(neu)} \end{bmatrix} = \begin{bmatrix} b_3 \\ b_4 \end{bmatrix} - \begin{bmatrix} s_{31} & s_{32} \\ s_{41} & s_{42} \end{bmatrix} \cdot \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

Jetzt wird das Gleichungssystem $\underline{s}_{neu} \cdot \vec{u}_{neu} = \vec{b}_{neu}$ fertig assembled,

$$\Rightarrow \begin{bmatrix} s_{33} & s_{34} \\ s_{43} & s_{44} \end{bmatrix} \cdot \begin{bmatrix} u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b_3^{(neu)} \\ b_4^{(neu)} \end{bmatrix}$$

Im C++ Programm:

```

for(int i=0;i<ndim.n_tetr_pkt;i++){
    if(RP.InnerePkt[i]<0){
        optg.x[i]=g.bglobal[i]; // die Temperatur am Rand ist schon bekannt
        j=j+1;
    }
    else{
        optg.bmR[m]=g.bglobal[i];
        for(int q=0;q<ndim.n_tetr_pkt;q++){
            tmp[m][q]=g.Sglobal[i][q];
        }
        // Innere Punkte neu nummerieren ==> Punktematrix verkürzen
        optg.PmR[m][0]=n.P[i][0];
        optg.PmR[m][1]=n.P[i][1];
        optg.PmR[m][2]=n.P[i][2];
        m++;
    }
}
tmp2=matmulvkt(tmp,optg.x,RP.pmRdim,ndim.n_tetr_pkt,ndim.n_tetr_pkt);
optg.bmR=vktminus(optg.bmR,tmp2,RP.pmRdim,RP.pmRdim);

```

danach wird das BiCG-Verfahren verwendet, um die Gleichung zu lösen.

6 Das numerische Verfahren BiCGSTAB-Verfahren

In diesem Kapitel geht es um das BiCGSTAB-Verfahren, das ein iteratives numerisches Verfahren zur approximativen Lösung eines unsymmetrischen linearen Gleichungssystems ist: $\underline{\underline{A}} \cdot \vec{x} = \vec{b}, \underline{\underline{A}} \in \mathbb{R}^{n \times n}$. Dieses Verfahren ist eine Variante der BiCG(Biconjugat-Gradienten-Methode) und hat eine schnellere und glattere Konvergenz als das normale BiCG-Verfahren sowie andere Varianten z.B die CGS(Conjugate-Gradient-Squared-Methode).

6.1 Projektionsmethoden und Krylov-Unterraum-Verfahren

Alle oben erwähnten Verfahren sowie sämtliche Iterationsverfahren zur Lösung linearer Gleichungssysteme basieren auf der Projektionsmethode des Krylov-Unterraum-Verfahrens. Das Prinzip der Projektionsmethoden liegt darin, dass ausgehend von einer Anfangsnäherung \vec{x}_0 eine approximative Lösung \vec{x}_m mit Hilfe der beiden m -dimensionalen Unterräume \vec{K}_m und \vec{L}_m des \mathbb{R}^n bestimmt werden. Die approximative Lösungen $\vec{x}_m \in \vec{x}_0 + \vec{K}_m$ ist durch diese Verfahren so zu berechnen und unter Berücksichtigung der Bedingung:

$$(\vec{b} - \underline{\underline{A}} \cdot \vec{x}_m) \perp \vec{L}_m \quad (3)$$

wobei die Orthogonalitätsbedingung über das euklidische⁵ Skalarprodukt definiert ist.

$$\vec{x} \perp \vec{y} \Leftrightarrow (\vec{x}, \vec{y})_2 = 0$$

Wenn man dabei $\vec{K}_m = \vec{L}_m$ wählt, so nennt man dies eine *Galerkin-Bedingung* bzw. ist $\vec{K}_m \neq \vec{L}_m$, so spricht man von einer *schießen Projektionsmethode* und es wird dann als *Petrov-Galerkin-Bedingung* bezeichnet. Ein spezifisches Projektionsverfahren ergibt durch die Wahl von \vec{L}_m , wobei sich durch die folgende Definition, der sogenannte **Krylov-Unterraum**:

$$\vec{K}_m = \vec{K}_m(\underline{\underline{A}}, \vec{r}_0) = \text{span}\{\vec{r}_0, \underline{\underline{A}} \cdot \vec{r}_0, \dots, \underline{\underline{A}}^{m-1} \cdot \vec{r}_0\}$$

mit $\vec{r}_0 = \vec{b} - \underline{\underline{A}} \cdot \vec{x}_0$ ergibt. Da \vec{K}_m von m Vektoren aufgespannt wird, ist die Dimension dieses Raumes höchstens gleich m . Diese Eigenschaften werden zur Erzeugung vieler Algorithmen benutzt. Ein gutes Beispiel ist das Gauß-Seidel-Verfahren:

$$x_i^{(m+1)} = -\frac{1}{a_{ii}} \left[\sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} + \sum_{j=i+1}^n a_{ij} x_j^{(m)} - b_i \right] \Leftrightarrow \vec{x}^{(m+1)} = \underline{\underline{D}}^{-1} \cdot [\vec{b} - \underline{\underline{L}} \cdot \vec{x}^{(m+1)} - \underline{\underline{R}} \cdot \vec{x}^{(m)}]$$

Gauß-Seidel-Verfahren kann auch als orthogonale Projektionsmethode mit

$\vec{x}_0 = (\vec{x}_{m+1,1}, \dots, x_{m+1,i-1}, 0, x_{m,i+1}, \dots, x_{m,n})^T$ und den eindimensionalen Räumen $\vec{K} = \vec{L} = \text{span}\{\vec{e}_i\}$ interpretiert werden⁶, wobei:

- $\vec{b}_i - (\underline{\underline{A}} \vec{x})_i = 0$ mit $\vec{x} \in \vec{x}_0 + \vec{K}$
- $\vec{b} - \underline{\underline{A}} \vec{x} \perp \vec{L}$ mit $\vec{x} \in \vec{x}_0 + \vec{K}$

Die **Krylov-Unterraum-Methoden** ermitteln die Nährungslösung $\vec{x}_m \in \vec{x}_0 + \vec{K}_m$ an die gesuchte Lösung $\underline{\underline{A}}^{-1} \cdot \vec{b}$ z.B. unter der Orthogonalitätsbedingung, wobei bei jeder Iteration die Dimension des Unterraums um Eins hoch-inkrementiert wird. Wenn die auftretenden Rundungsfehler vernachlässigt werden, würden diese Methode nach dem Iterationsverlauf die exakte Lösung herausbekommen. Das heißt, man braucht nur einen Anfangsvektor \vec{x}_0 und das Residuum \vec{r}_0 einzusetzen und damit initialisieren, wobei in der Iterationsschleife dann $\vec{r}_m = \underline{\underline{A}} \vec{x}_m - \vec{b}$ wird. Und dann nach dem m Iterationsschritten wird der Raum \vec{K}_m durch Verwenden des Verfahrens aufgespannt. Mit Hilfe eines Prüfvariablen im Programm kontrolliert das Verfahren die Genauigkeit des Lösungsvektors bzw. die Anzahl des Iterationsschritte.

Die beiden bekanntesten **Krylov-Unterraum-Methoden** sind:

⁵ $(\vec{x}, \vec{y})_2 = \vec{x} \cdot \vec{y} := \sum_i x_i y_i$ mit $x_i, y_i \in \mathbb{R}$

⁶ Mit \vec{e}_i der kanonischen Basis.

- **Konjugierte Gradienten–Verfahren** (=CG–Verfahren), das von Hestenes und Stiefel im Jahr 1952 hergeleitet wurde
- die **GMRES–Methode** von Saad und Schulz aus dem Jahr 1986

Aus dem CG–Verfahren können dann viele iterative Varianten (1. BiCG, CGS, BiCGStab, ... oder 2. GMRES, QMR, TFQMR, ..., siehe [3] S.142 ff.) als Iterationsalgorithmen hergeleitet werden. Spezielle Lösungsverfahren ergeben sich durch die geeignete Wahl des Raumes \tilde{L}_m , sowie durch Ausnutzen von speziellen Eigenschaften der Matrix $\underline{\underline{A}}$, wodurch das Verfahren beschleunigt werden kann, aber auch seine Anwendbarkeit unter Umständen einschränkt.

6.2 Warum wird das BiCGSTAB–Verfahren gewählt?

Das bikonjugierte⁷ stabilisierte Gradientenverfahren wird kurz als BiCGStab–Verfahren bezeichnet, das auf den CGS– und BiCG–Verfahren basiert und von Van der Vorst[10] im Jahr 1992 hergeleitet wurde. [3]: Das Verfahren kann das unregelmäßige Konvergenzverhalten des CGS–Verfahrens vermeiden und kann die Oszillation durch eindimensionale Residuenminimierung verkleinern. Es konvergiert im Großteil der Fälle ähnlich schnell wie das CGS–Verfahren und kann als Kombination des BiCG–Verfahrens mit der wiederholten Ausnutzung des GMRES–Verfahrens[3] interpretiert werden. Zumindest lokal wird ein Residuenvektor minimiert, was die Konvergenz wesentlich glättet. Es ist gut geeignet für FEM–Wärmeleitungs-Probleme mit **Konvektionsterm**, denn in diesem Fall, wird die Matrix und das Gleichungssystem dann wegen der Konvektion zu einem nichtsymmetrischen linearen Gleichungssystem. Ohne Konvektion bleibt das FEM–Wärmeleitungs–Probleme positiv–definit und symmetrisch, also CG geeignet.

6.3 Algorithmische Schritte

Um ein lineares System zu löseng beginnt das BiCGSTAB mit einem Start–Vektor \vec{x}_0 und dann berechnet sich mit der folgenden Schritten:

1. Setze $\vec{p} = \vec{b} - \underline{\underline{A}} \cdot \vec{x}$
2. Setze $\vec{b} = \vec{b} - \underline{\underline{A}} \cdot \vec{x}$
3. Setze $\vec{r}_0 = \vec{r}$
4. Setze $\rho = \vec{r} \cdot \vec{r}$
5. For–Schleife laufen($k = 1, 2, 3, \dots$) bis $\vec{r} \cdot \vec{r} < \epsilon$ (*sogenannter Proof*)
6. $\vec{v} = \underline{\underline{A}} \cdot \vec{p}$
7. $\alpha = \rho / (\vec{v} \cdot \vec{r}_0)$
8. $\vec{s} = \vec{r} - \alpha \cdot \vec{v}$
9. $\vec{t} = \underline{\underline{A}} \cdot \vec{s}$
10. $\omega = \frac{\vec{t} \cdot \vec{s}}{\vec{t} \cdot \vec{t}}$
11. $\vec{x}_{k+1} = \vec{x}_k + \alpha \cdot \vec{p} + \omega * \vec{s}$
12. $\vec{r} = \vec{s} - \omega \cdot \vec{t}$
13. ρ_k speichern
14. $\rho_{k+1} = \vec{r} \cdot \vec{r}_0$
15. $\beta = \alpha / \omega \cdot \rho_{k+1} / \rho_k$

⁷Durch die Bi–Konjugiertheit können auch nicht–symmetrische lineare Gleichungssysteme mit dem CG–Verfahren gelöst werden.

$$16. \vec{p} = \vec{r} + \beta \cdot (\vec{p} - \omega \cdot \vec{v})$$

17. Ende der For-Schleife

Im C++ Programm:

```

while(dot_product(r,r,ndim,ndim) > eps){
    v=matmulvkt(a,p,ndim,ndim,ndim);
    alpha=rho/dot_product(v,r0,ndim,ndim);
    s=vktminus(r,constmulvkt(alpha,v,ndim),ndim,ndim);
    t=matmulvkt(a,s,ndim,ndim,ndim);
    omega=dot_product(t,s,ndim,ndim)/dot_product(t,t,ndim,ndim);
    x=vktplus(x,constmulvkt(alpha,p,ndim),ndim,ndim);
    x=vktplus(x,constmulvkt(omega,s,ndim),ndim,ndim);
    r=vktminus(s,constmulvkt(omega,t,ndim),ndim,ndim);
    rho_alt=rho;
    rho=dot_product(r,r0,ndim,ndim);
    beta=alpha/omega*rho/rho_alt;
    tmp1=vktminus(p,constmulvkt(omega,v,ndim),ndim,ndim);
    tmp2=constmulvkt(beta,tmp1,ndim);
    p=vktplus(r,tmp2,ndim,ndim);
    it++;
    cout.setf(ios::scientific);
    cout<<"BiCGStab_solve : it = "<<it<<" proof = "<<setprecision(8)<<dot_product(r,r,ndim,ndim)<<endl;
}

```

7 Doxygen Einführung

7.1 Einleitung

Doxygen ist ein freies Software-Dokumentationswerkzeug unter der GNU General Public License. Durch spezielle Kommentare im Quellcode können Softwareentwickler Erläuterung zu Programmelementen definieren, aus denen Doxygen eine übersichtliche Dokumentation erstellt. Es ist auch möglich, den Zusammenhang zwischen jeweiligen Unterprogrammen durch eine Grafik zu erklären. Doxygen kann nicht nur HTML sondern LaTeX automatisch erzeugen. Als grafische Benutzeroberfläche dient der sogenannte Doxywizard, mit dem die Einstellungen der Ausgabe-Files sehr einfach und schnell vorgenommen werden können. Folgende Programmiersprachen wird unterstützt: C++, C, Objective-C, Java, Fortran, Python und IDL.

7.2 Installation

Doxygen benutzt das GraphViz-Paket, um die Grafik zu bezeichnen, deswegen werden 2 Pakete unter Ubuntu in folgender Weise installiert.

- *sudo apt-get install doxygen*
- *sudo apt-get install graphviz*

7.3 Benutzen der Doxywizard

Durch folgendes Kommando wird eine Benutzeroberfläche für Doxygen aufgerufen:

- *doxywizard*

Folgende Abbildung zeigt die Einstellungen des Ausgabe-Files.

1. einmal linken Maustastenklick in "Project", dann ist Konfiguration des Projekts einzustellen.

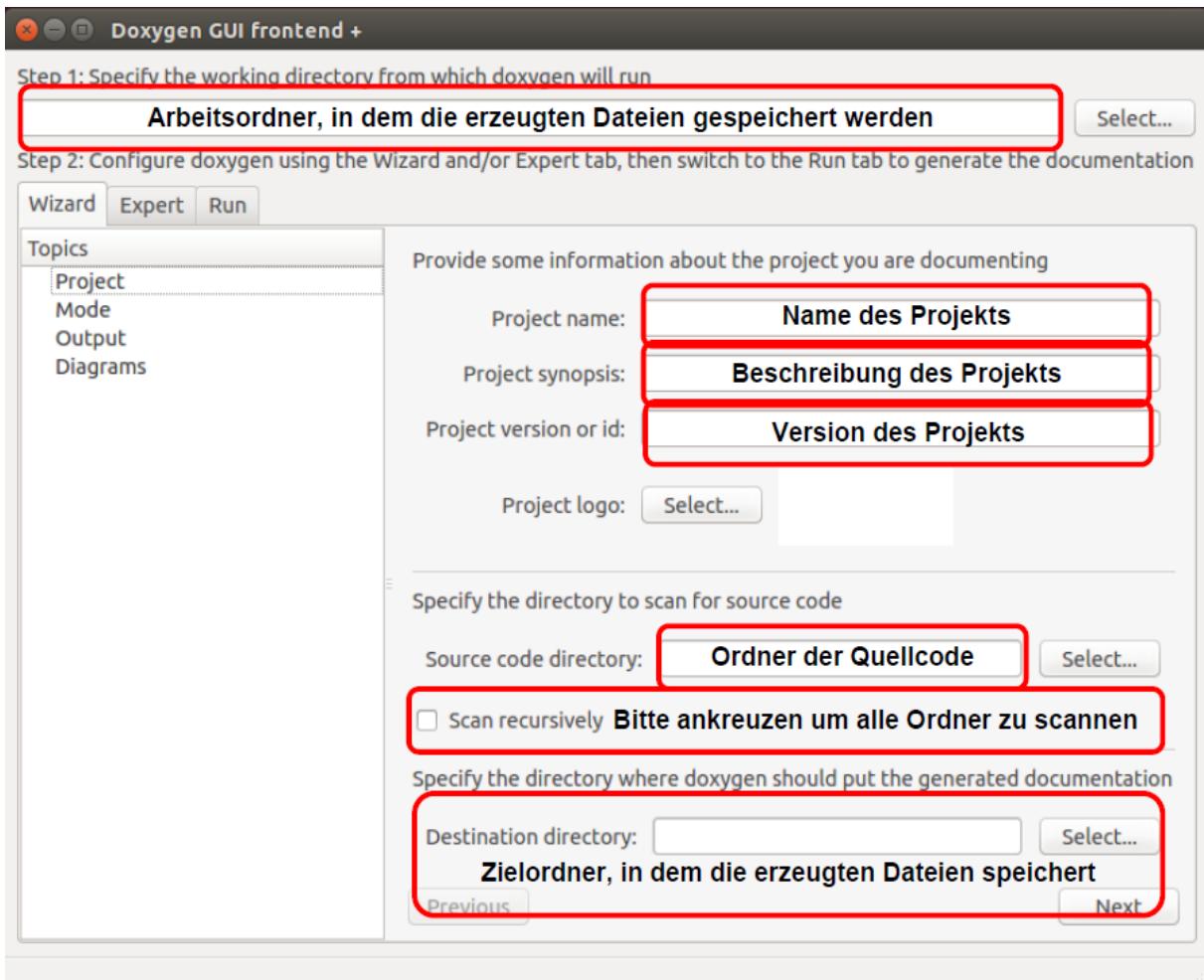


Abbildung 16: Schritt1 "Projekt"

2. Wenn alle Optionen für "Project" richtig eingestellt sind, Klick den "Next"-Button in dem "Mode"-Menu.

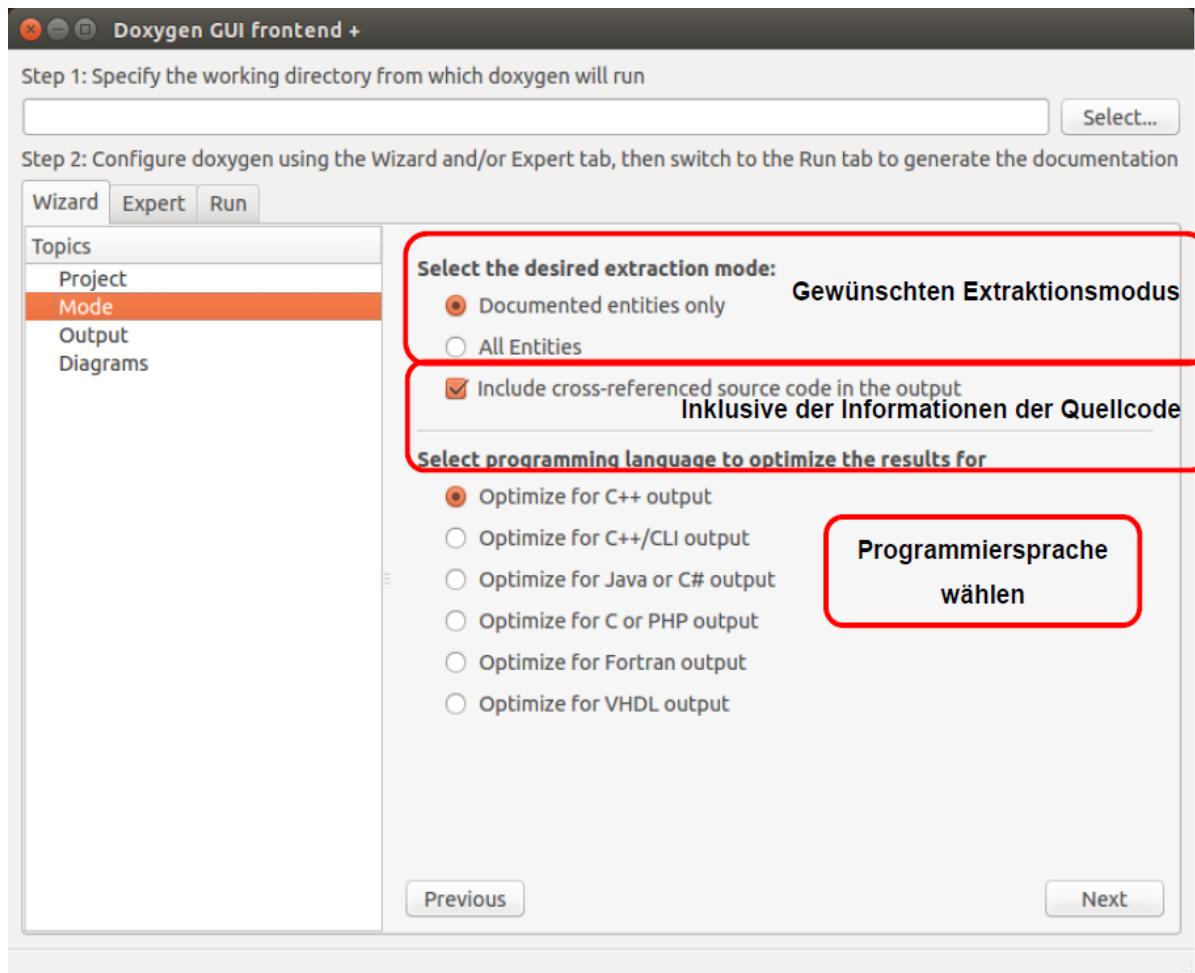


Abbildung 17: Schritt2 "Mode"

3. Wenn alle Optionen für "Mode" richtig eingestellt wurden, Klick den "Next"-Button im "Output"-Menu. Hier wird nur LaTeX eingestellt.

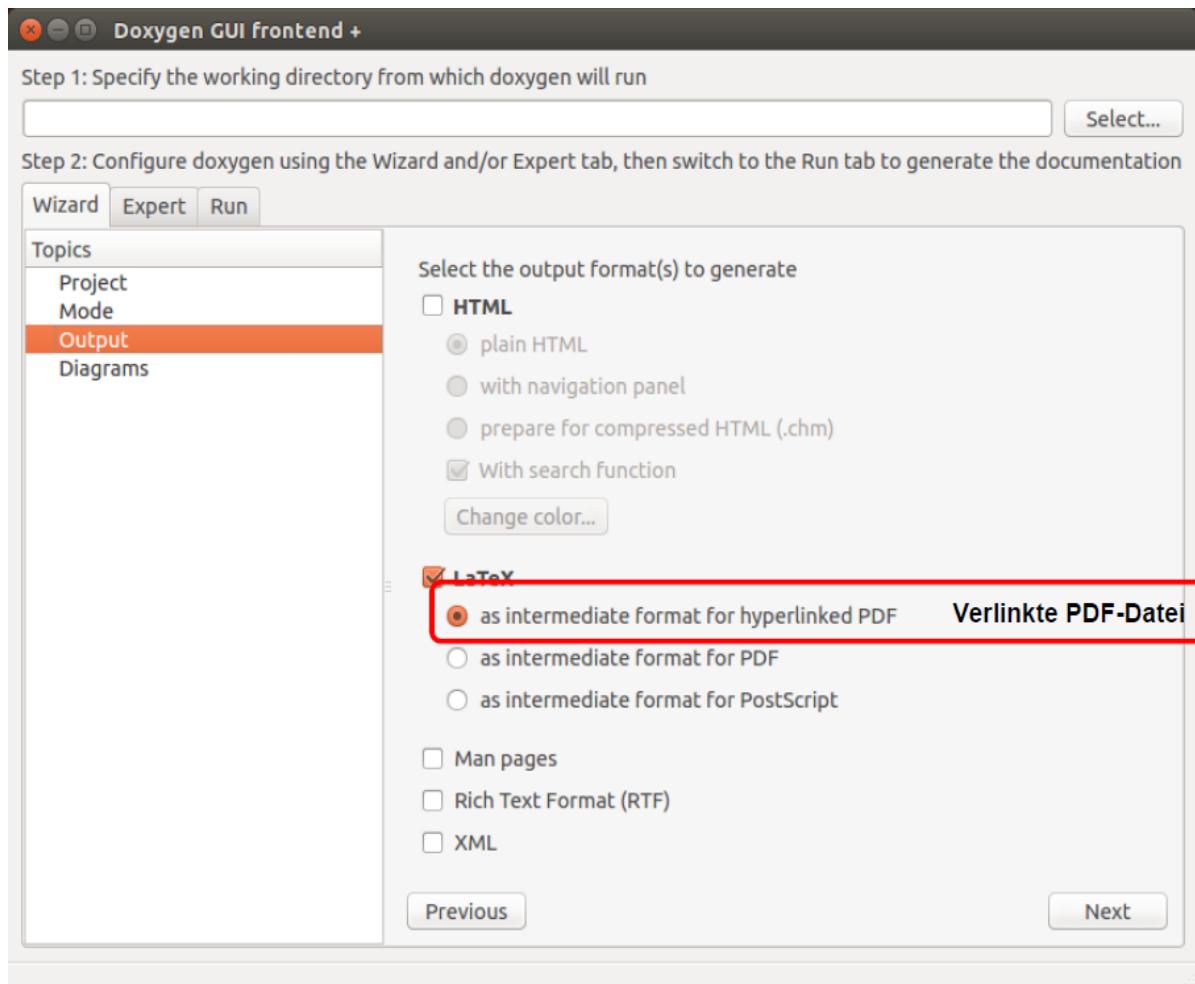


Abbildung 18: Schritt3 "Output"

4. Wenn alle Optionen für "Output" richtig eingestellt wurden, Klick den "Next"-Button im "Diagrams"-Menu. Hier wird das GraphViz-Paket benutzt, um verschiedene Grafiken zeichnen lassen zu können, z.B. Aufrufbäume.

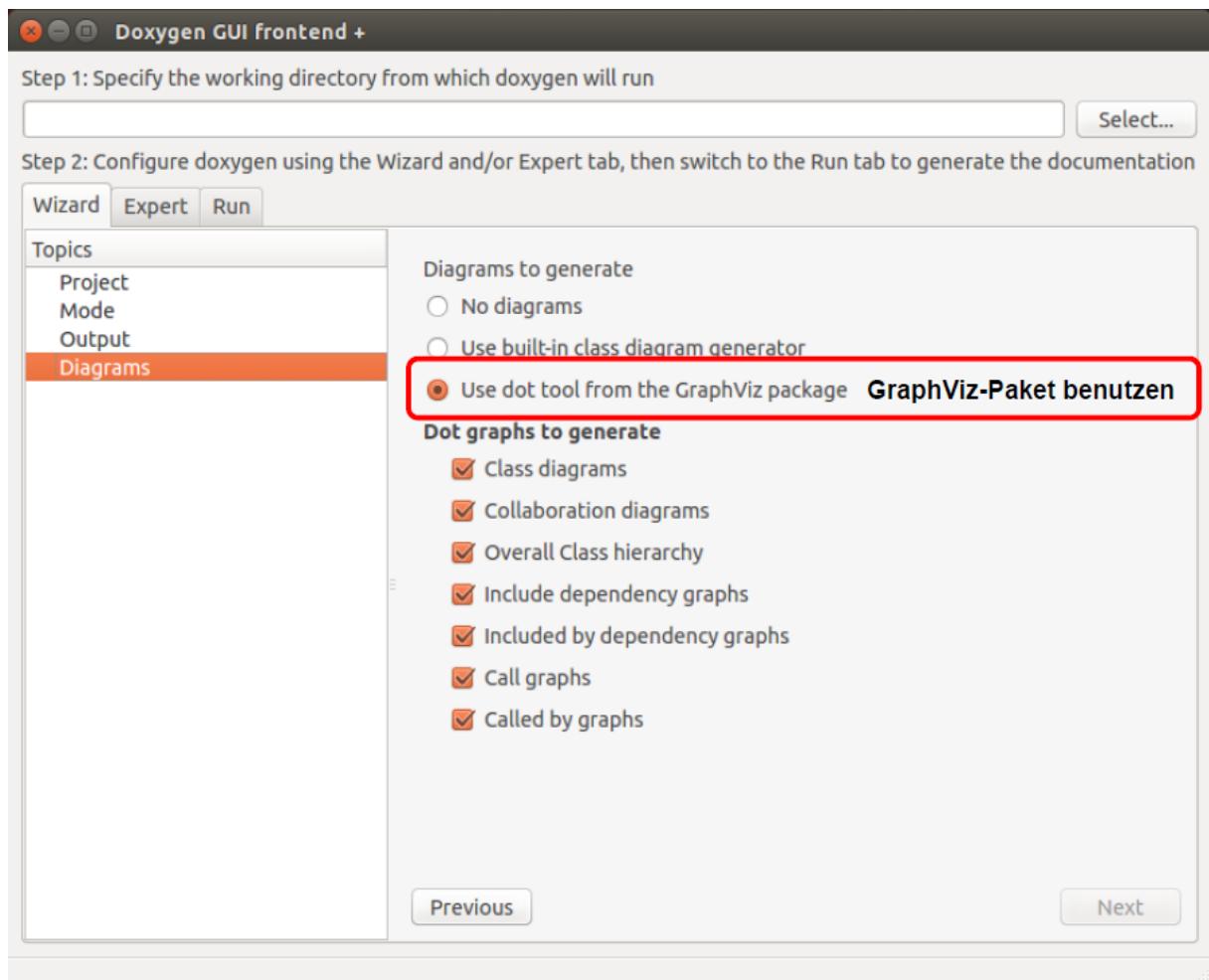


Abbildung 19: Schritt4 "Diagrams"

5. In "Wizard" sind nur allgemeine konfigurationen vorzunehmen, und mehr Optionen wählen zu können, müssen in "Expert" die Einstellungen durchgeführt werden.

- "Project":

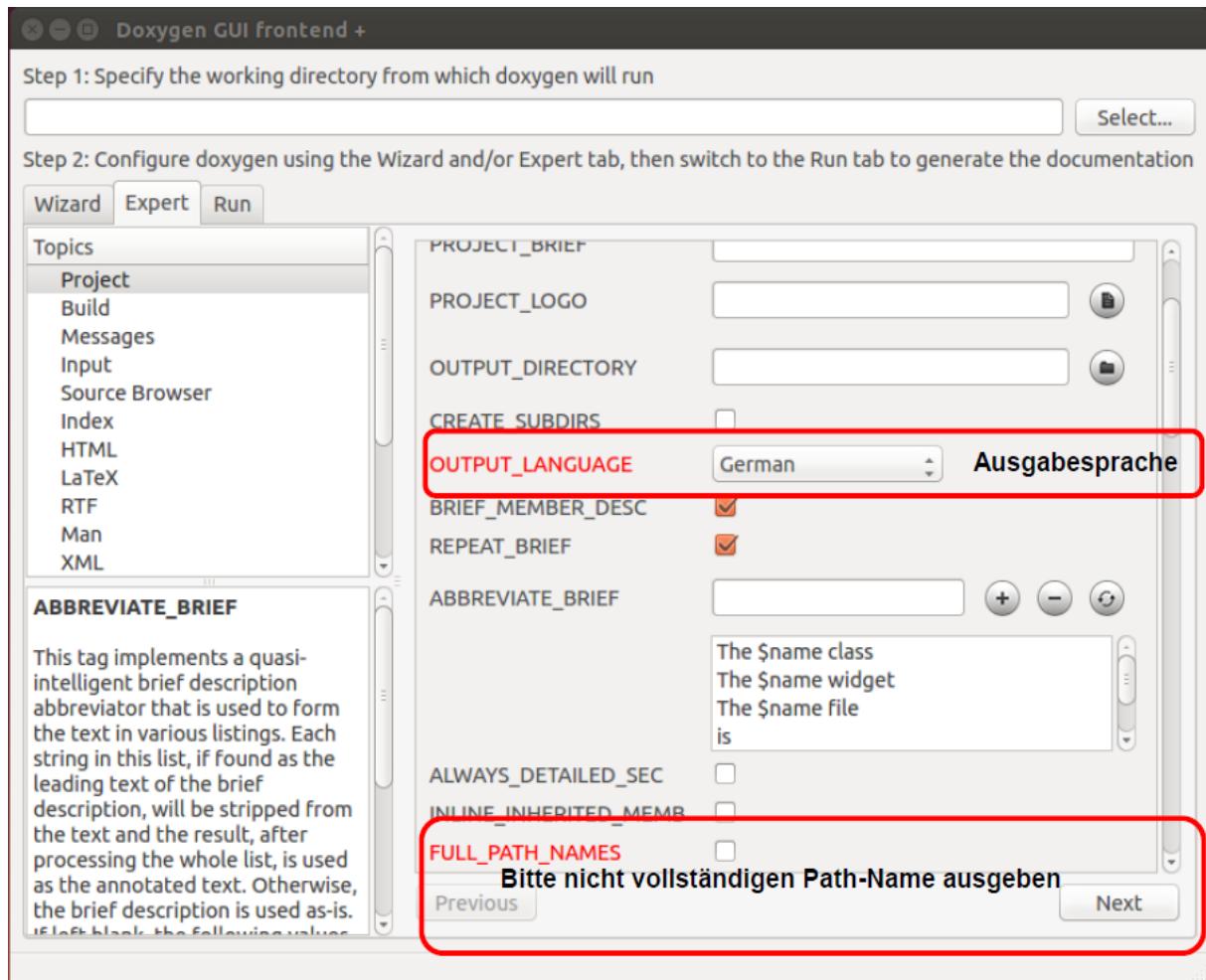


Abbildung 20: Schritt5.1 "Expert"

Wenn der vollständige Pfad-Name angekreuzt wird, sieht die Liste desjeweiligen Programms ziemlich lang aus. z.B. die Auflistung der Dateien:

1.1 Auflistung der Dateien	
Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:	
bicgstab_solve.cpp	Das BiCG-Verfahren
constmulvkt.cpp	Ein Unterprogramm für konstant*Vektor
dim_tetr.cpp	Anzahl des Tetraeder bestimmen
dim_tetr_pkt.cpp	Anzahl der Punkten bestimmen
dot_product.cpp	Skalarprodukt
FEM.cpp	Hauptprogramm FEM_Tetraeder
Flaechen_RB.cpp	Flächen Randbedingung
funktions.h	Funktionen declaration
matmulvkt.cpp	Ein Unterprogramm für Matrix*Vektor
randpunkte_InnerePkt.cpp	Innere Punktenummer bestimmen
randpunkte_PNummerF.cpp	Randpunkte bestimmen
RBdimension.cpp	Anzahl der Randpunkte bestimmen
SteifigkeitsMatrix.cpp	Steifigkeitsmatrix berechnen
tetr_input.cpp	Tetraeder Element einlesen
tetr_pkt_input.cpp	Koordinatensystem aller Punkten einlesen

Abbildung 21: der richtige Path-Name

1.1 Auflistung der Dateien	
Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:	
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ bicgstab_solve.cpp	Das BiCG-Verfahren
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ constmulvkt.cpp	Ein Unterprogramm für konstant*Vektor
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ dim_tetr.cpp	Anzahl des Tetraeder bestimmen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ dim_tetr_pkt.cpp	Anzahl der Punkten bestimmen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ dot_product.cpp	Skalarprodukt
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ FEM.cpp	Hauptprogramm FEM_Tetraeder
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ Flaechen_RB.cpp	Flächen Randbedingung
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ funktions.h	Funktionen declaration
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ matmulvkt.cpp	Ein Unterprogramm für Matrix*Vektor
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ randpunkte_InnerePkt.cpp	Innere Punktenummer bestimmen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ randpunkte_PNummerF.cpp	Randpunkte bestimmen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ RBdimension.cpp	Anzahl der Randpunkte bestimmen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ SteifigkeitsMatrix.cpp	Steifigkeitsmatrix berechnen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ tetr_input.cpp	Tetraeder Element einlesen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ tetr_pkt_input.cpp	Koordinatensystem aller Punkten einlesen
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ vktminus.cpp	Ein Unterprogramm für $\vec{a} - \vec{b}$
/home/god-cyy/Dokuments/BA/Chen_Youyu.d/CPP_FEM.d/ vkplus.cpp	Ein Unterprogramm für $\vec{a} + \vec{b}$

Abbildung 22: der falsche Path-Name

- In "Build":

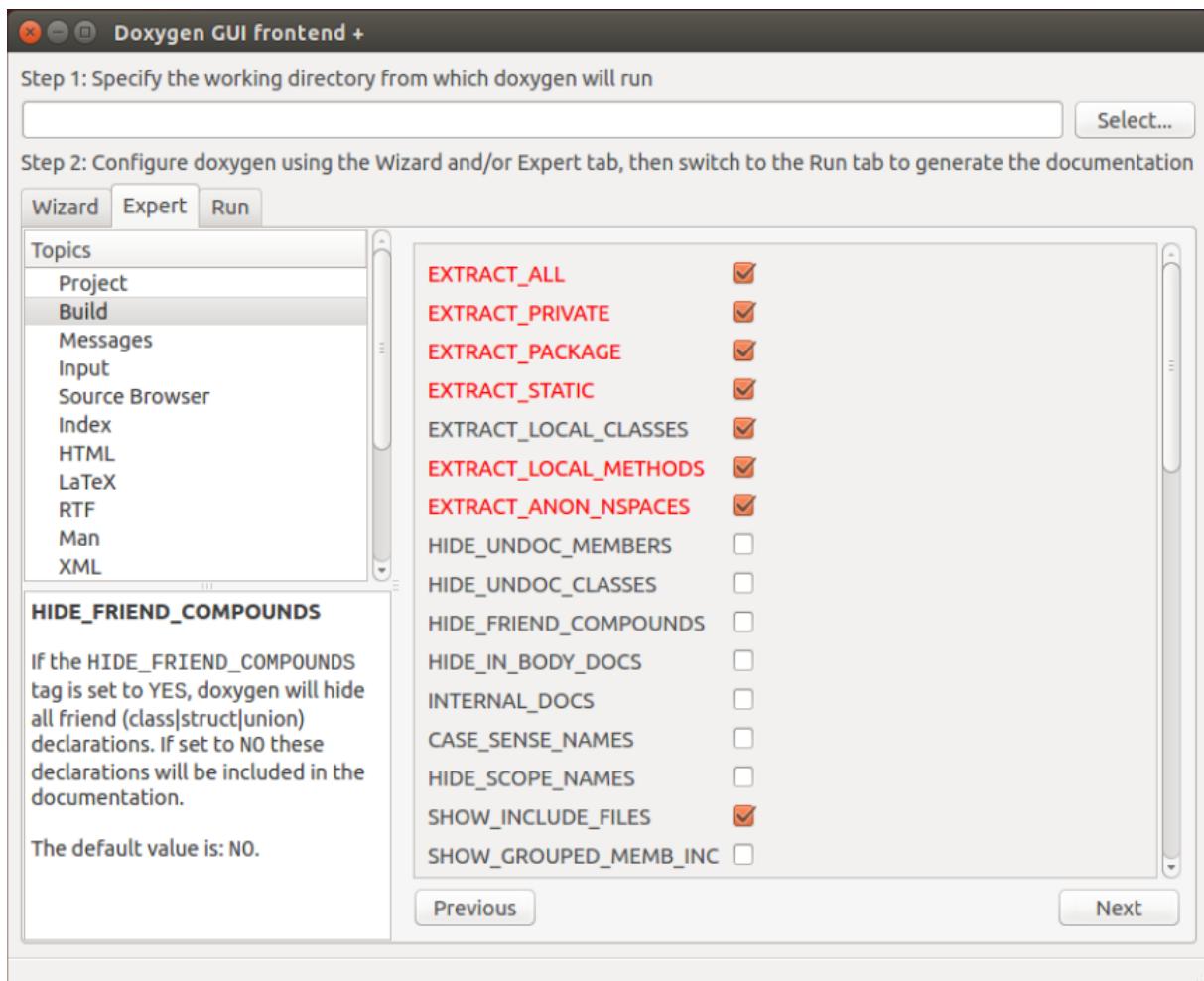


Abbildung 23: Schritt5.2 "Expert"

6. Jetzt werden die Ausgabedateien richtig konfiguriert. Einmal rechte Maustastenklick in "Run doxygen", dann werden die LaTeX-Dateien automatisch erzeugt.
- Die Laufinformation ist auch zu speichern, wenn man "Save log..." klickt.

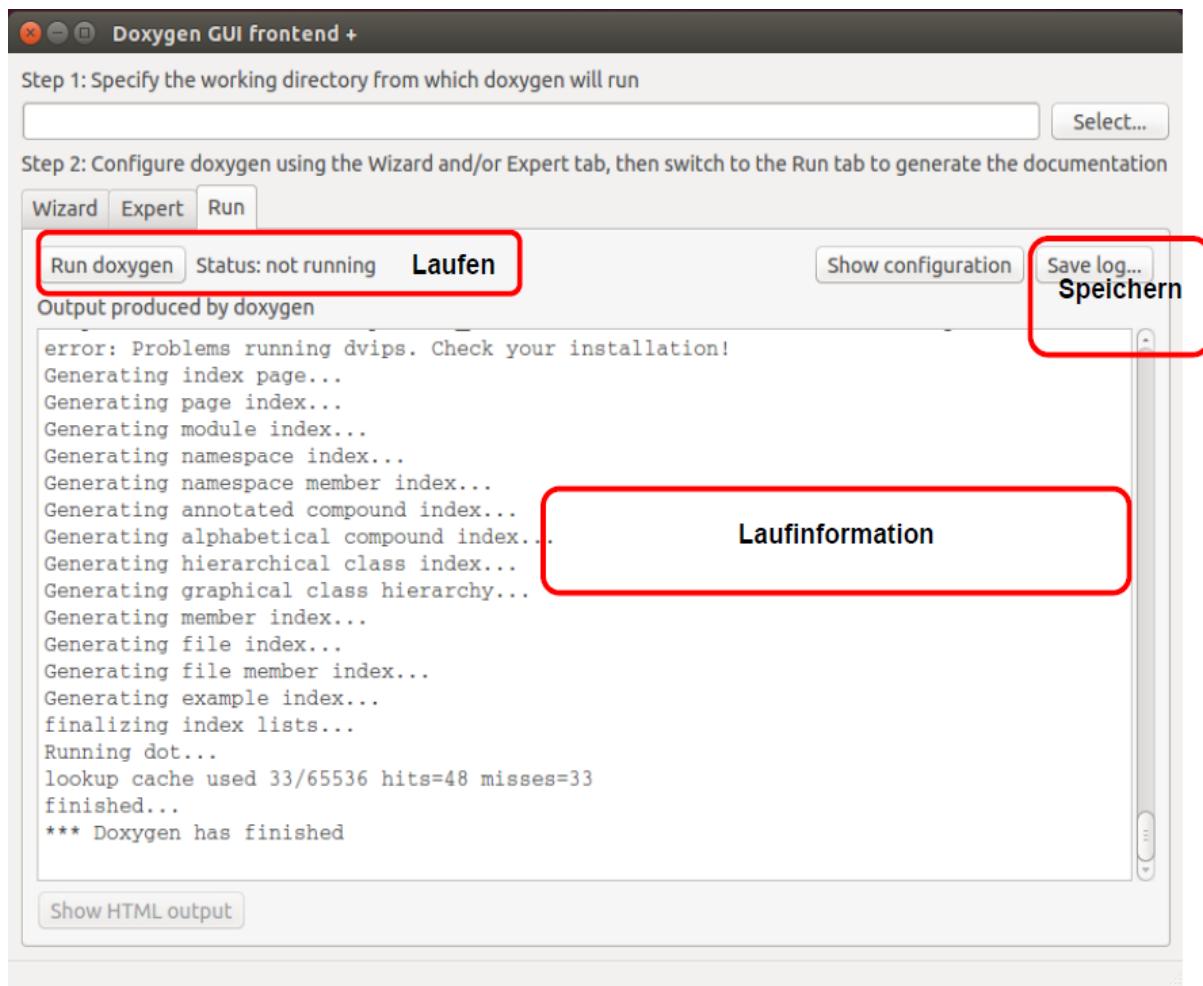


Abbildung 24: Schritt6.1 "Run"

- Alle Konfiguration ist auch zu sehen, wenn man "Show configuration" klickt.

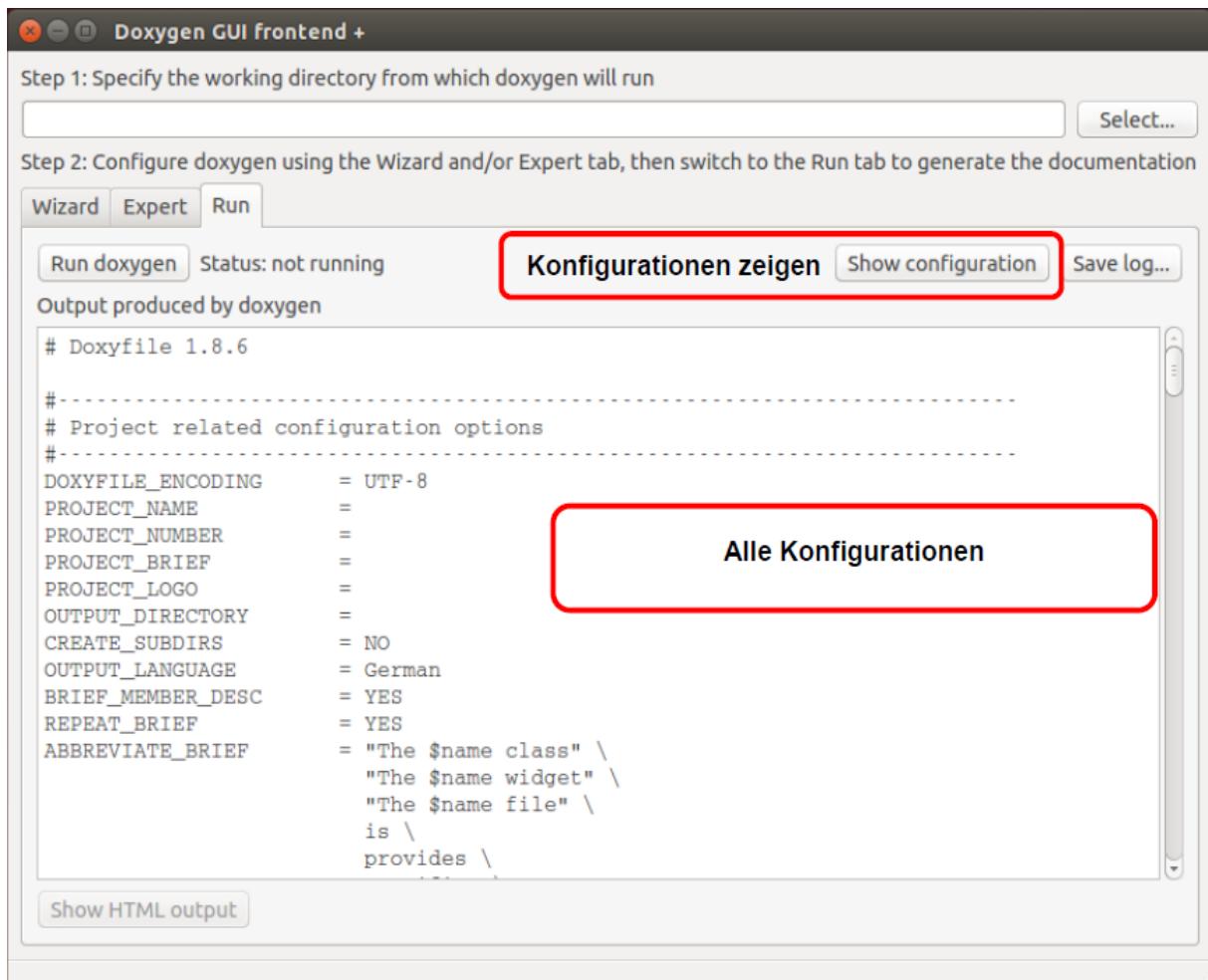


Abbildung 25: Schritt6.2 "Run"

- Wenn Doxygen fertig ist mit seiner Arbeit, liegt ein Ordner "latex" im Zielordner, in der "*.tex" Dateien und Grafiken zu finden sind. Alle LaTeX-Dateien werden durch einen Makefile zusammengefaßt und die Erzeugung durch L^AT_EXdadurch gesteuert.

Durch folgendes Kommando lässt sich die PDF-Datei erzeugen:

- make*

Wenn die PDF-Dateien nicht mehr gebraucht werden, sind sie auch lösbar:

- make clean*

Der Standardname der PDF-Datei heißt "refman.pdf". Sehen ein Beispiel Dokumentation Datei im Anhang [A.2](#)

8 Versionsverwaltung durch Eclipse(IDE)

8.1 Einleitung des Eclipse(IDE)

Eclipse ist ein quelloffenes freies Programmierwerkzeug zur Entwicklung von Software verschiedenster Art. Ursprünglich wurde Eclipse als integrierte Entwicklungsumgebung(IDE) für die Programmiersprache Java genutzt, aber mittlerweile wird es wegen seiner Erweiterbarkeit auch für viele andere Entwicklungsaufgaben (z.B. Programmiersprache C++, PHP, Python) durch Plugins eingesetzt. Es gibt eine Vielzahl sowohl quelloffener als auch kommerzieller Erweiterungen. Eclipse ist selbst basiert auf Java-Technik, aber ab Version 3.0 auf dem OSGi-Framework Equinox.

8.2 Installation

Das aktuelle Eclipse kann aus der offiziellen Webseite von Eclipse heruntergeladen werden. Zuerst öffnet man den Installer: *eclipse-inst*, und wählt das benötigte Plugin: Hier wird das IDE für C++ installiert:

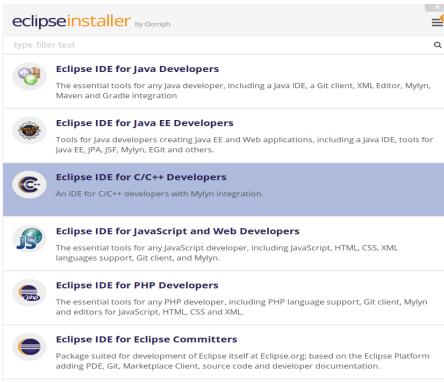


Abbildung 26: Eclipse Installation

8.3 Versionsverwaltung

Eine Versionsverwaltung ist ein System, das zum Kontrollieren von Änderungen an Dateien verwendet wird. Alle Versionen werden in einem Archiv mit Zeitstempel und Benutzerkennung gespeichert und können später zurückgesetzt oder wiederhergestellt werden. Typischerweise wird dieses System in der Software-Entwicklung verwendet, um Quelltexte zu pflegen. Das Eclipse-System bietet eine gute Unterstützung für das Versionsverwaltungssystem Git. Diese Unterstützung wird vom EGit-Projekt durch eine Reihe von Plugins durchgeführt. Eclipse verwendet die JGit-Bibliothek, um die Git-Befehle auszuführen. JGit ist eine Bibliothek, die die Git-Funktionalität in Java implementiert. Im aktuellen Eclipse werden diese Git-Plugins inklusiv installiert, bzw. man kann sie direkt benutzen.

8.4 Import eines Makefile-Projekts

Als Beispiel wird jetzt ein C++ Programm mit Makefile importiert, um die kommenden Versionen zu pflegen. Diese Version ist das FEM-Programme, welches im Folgenden untersucht wird und das die einfache Geometrie eines Würfels benutzt, um seine Berechnungen durchführen zu können. Danach wird eine verallgemeinerte Geometrie eingeführt, z.B. eine aus einem CAD-System importierte Datei mit komplizierterer Geometrie. Zuerst einmal linke Maustastenklick in **"File" → "Import" und "Existing Code as Makefile Projekt"** wählen, dann anklicken des **"Next"-Button** nach dem zweiten Schritt:

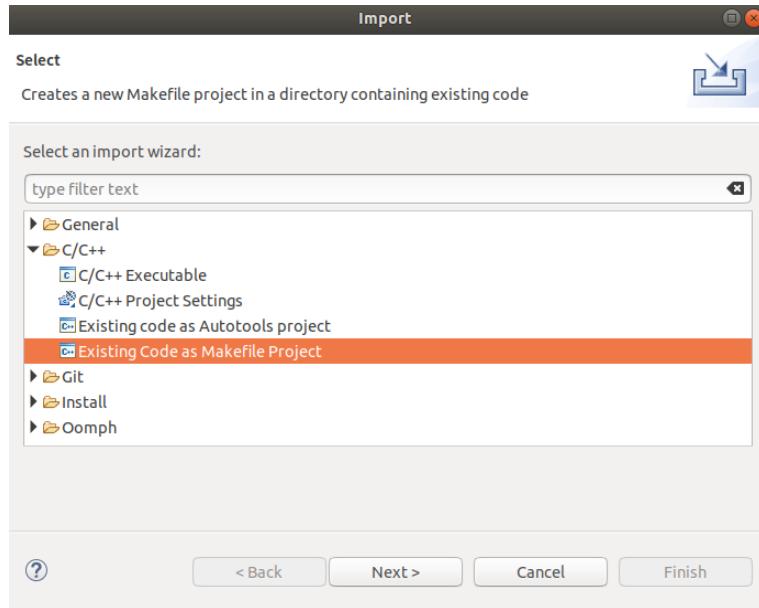


Abbildung 27: Import eines Projekts

Als zweites wird der Quellcode gewählt und der Name des Projekts definiert:

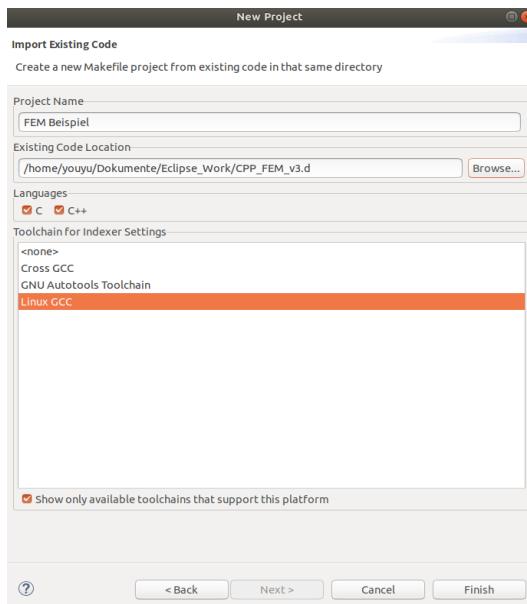


Abbildung 28: Projekt Name

Am Ende klicken des "Finish"-Buttons. Dateien des Projekts "**FEM_Beispiel**" sind an der linken Seite dargestellt:



Abbildung 29: Dateien des Projekts

8.5 Konfiguration des Makfiles

Das Projekt wird durch einen Makefile gesteuert, der *FEM.mk* heißt. Im Terminal wird das folgende Kommando ausgeführt:

- *make -f FEM.mk*

Das wird auch in Eclipse konfiguriert.

Zuerst einmal linken Maustastenklick in "Projekt" → "Properties"

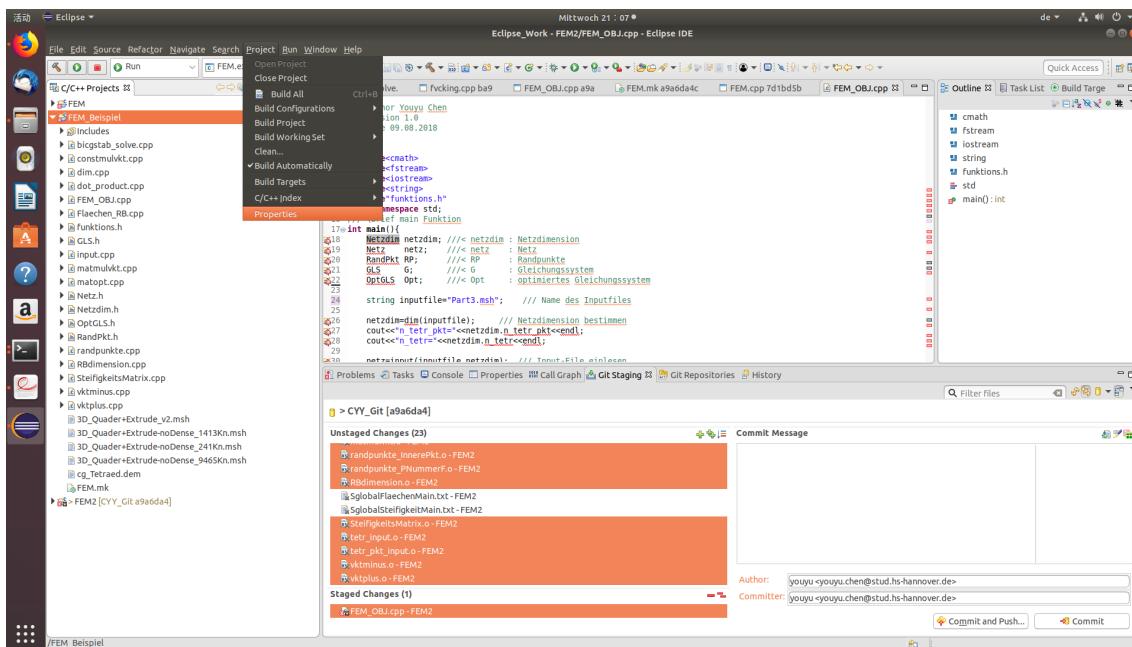


Abbildung 30: Eingenschaften des Projekts

Hier wird nicht das Standard Build-Kommando sondern *make -f FEM.mk* benutzt.

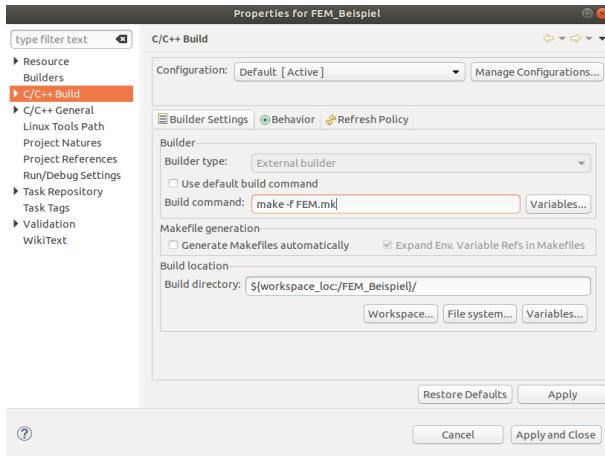


Abbildung 31: Make Kommando1

Dann anklicken von **”Behavior”**, und löschen durch **”all”**. Am Ende anklicken von **”Apply”**.

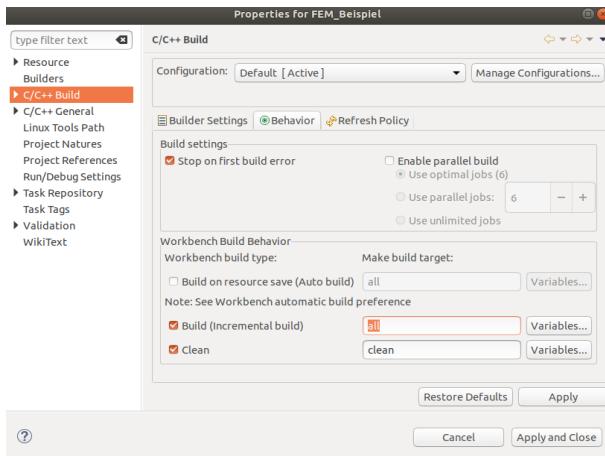


Abbildung 32: Make Kommando2

Das Makefile wird jetzt fertig eingestellt, und man kann in **”Projekt”** → **”Build Projekt”** oder **”clean”** durchführen. Die Informationen über Fehlermeldungen werden im Console-Fenster dargestellt.

```

Problems Tasks Console Properties Call Graph Git Staging Git Repositories History
CDT Build Console [FEM_Beispiel]
23:03:00 **** Build of configuration Default for project FEM_Beispiel ****
make -f FEM.mk
g++ -c -O3 FEM_OBJ.cpp
g++ -c -O3 dim.cpp
g++ -c -O3 input.cpp
g++ -c -O3 R8dimension.cpp
g++ -c -O3 randpunkte.cpp
g++ -c -O3 SteifigkeitsMatrix.cpp
g++ -c -O3 Flaechen_RB.cpp
g++ -c -O3 bicgstab_solve.cpp
bicgstab_solve.cpp: In function 'int bicgstab solve(Netzdim, Netz, RandPkt, OptGLS)':
bicgstab_solve.cpp:147:8: warning: ignoring return value of 'int system(const char*)', declared with attribute warn_unused_result [-Wunused-result]
        system("gnuplot cg_Tetraed.dem");
        ^
g++ -c -O3 matmulvkt.cpp
g++ -c -O3 dot_product.cpp
g++ -c -O3 vktminus.cpp
g++ -c -O3 vktplus.cpp
g++ -c -O3 constmulvkt.cpp
g++ -c -O3 matopt.cpp
FEM.mk : Try to link the executable !
g++ FEM_OBJ.o dim.o input.R8dimension.o randpunkte.o SteifigkeitsMatrix.o Flaechen_RB.o bicgstab_solve.o matmulvkt.o dot_product.o vktminus.o constm
23:03:04 Build Finished. 0 errors, 1 warnings. (took 4s.278ms)

```

Abbildung 33: Make in Console

Die ausführbare Programm als binäre Datei kann mit rechtem Maustastenklick **”Run as”** → **”Local**

C/C++ Application” gestartet werden.

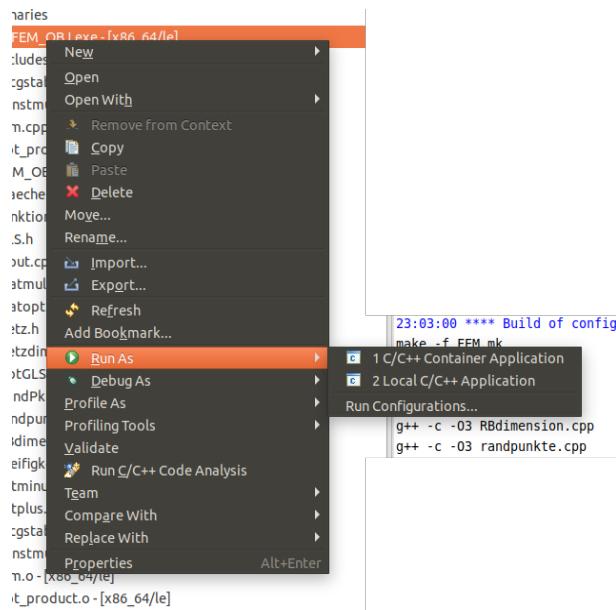


Abbildung 34: ausführen

8.6 Git Repository erzeugen

Jetzt ist das Git Repository zu erzeugen. Zuerst einmal rechte Maustastenklick am Projekt und nach "Team" → "Share Projekt".

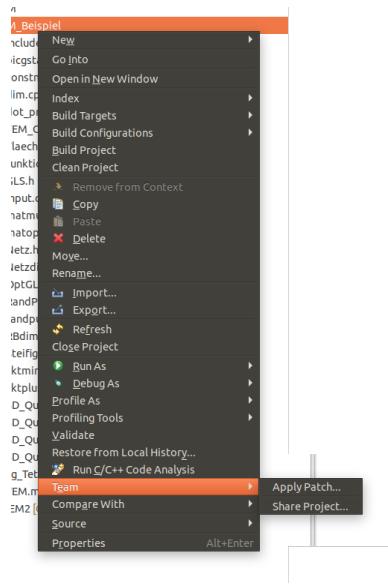


Abbildung 35: Git Repository1

Dann kreuzen von "Use or create repository in parent folder of projekt" an.

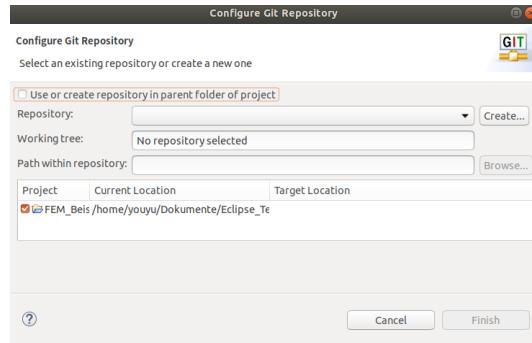


Abbildung 36: Git Repository2

Danach linken Maustastenklick in "Create Repository",

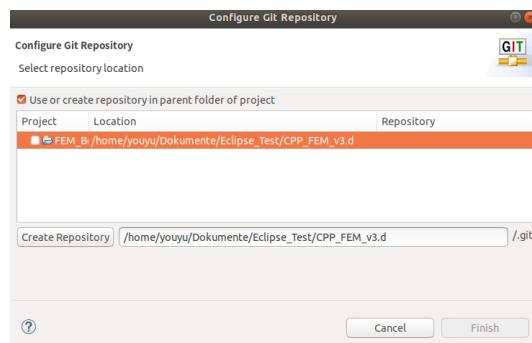


Abbildung 37: Git Repository3

ein Git Repository wird fertig unter dem Projekt–Ordner erzeugt.

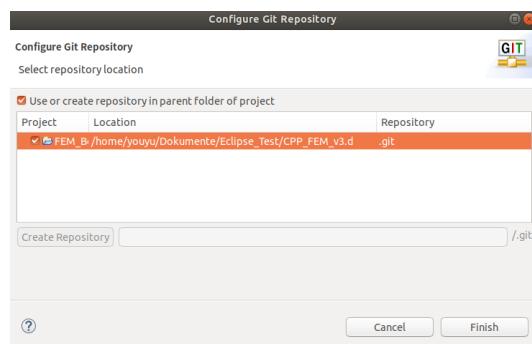


Abbildung 38: Git Repository4

8.7 Version pflichten

Jetzt wieder rechte Maustatsenklick am Projekt und nach "Team" gehen, dort gibt es viele neue Option für das Git Repository. Und anklicken von "Commit". Dann kommt man in ein *Git Staging* Fenster an der rechten Seite. Im "Commit Message"-Fenster kann man die Informationen der neuen Version hineinschreiben.

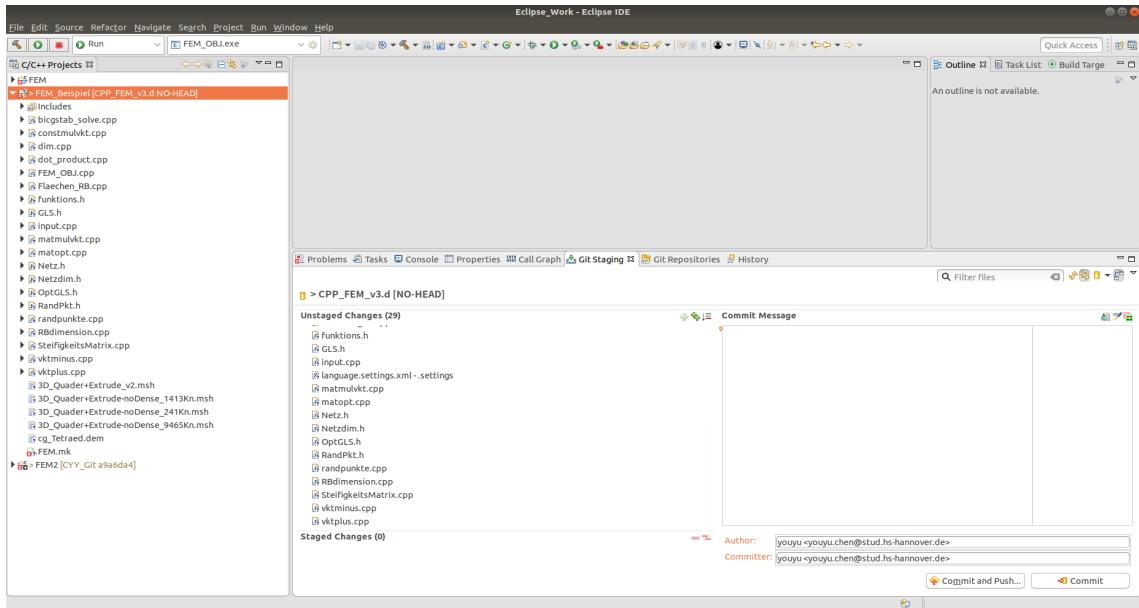


Abbildung 39: Git Repository5

Und dann verlegt man alle Dateien von *"Unstaged Changes"* in *"Staged Changes"*, um die originale Version zu erzeugen. Einmal linke Maustastenklick in **"Commit"**, diese Version wird als *"Master"* gespeichert.

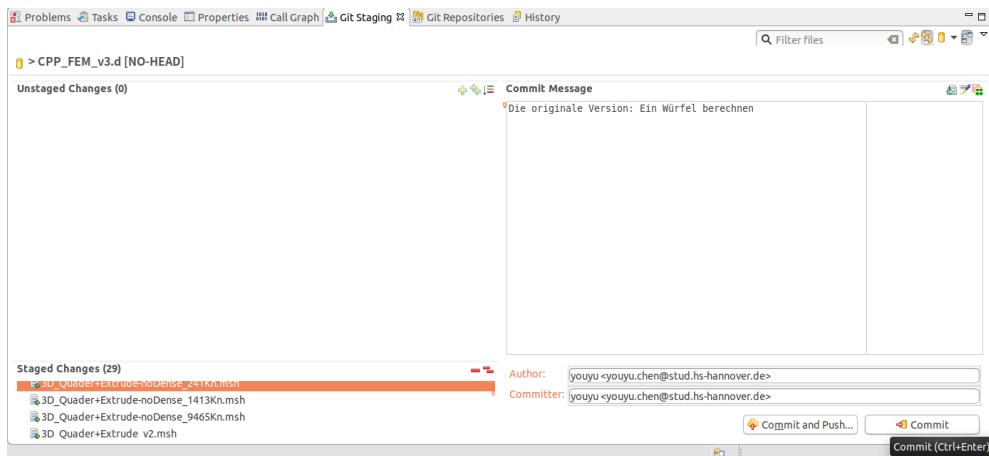


Abbildung 40: Git Repository6

Nochmal rechte Maustastenklick am Projekt und nach **"Team"** → **"Show in History"** gehen, dann werden die verschiedenen Versionen zeilenweise dargestellt.

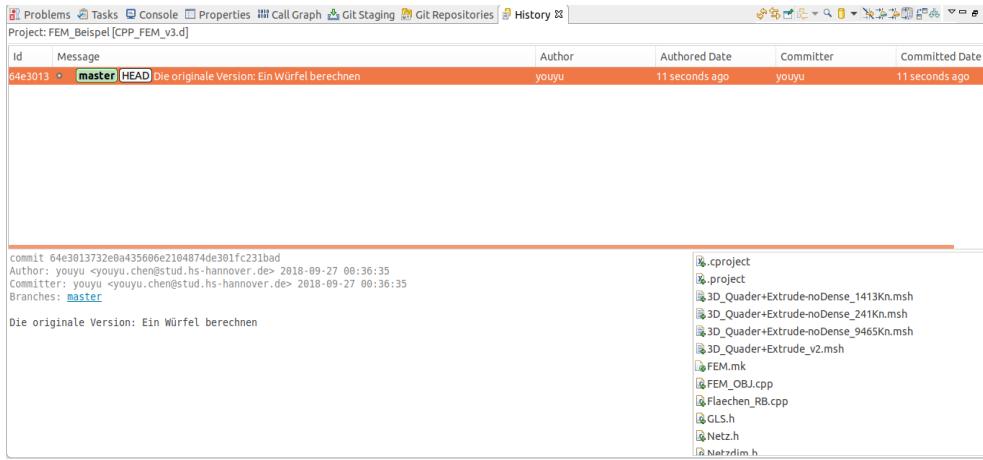


Abbildung 41: Git Repository7

Jetzt wird am Projekt etwas gearbeitet, um eine allgemeinere Geometrie zu berechnen. Der folgende Quellcode wird geändert: "FEM.mk", "FEM_OBJ.cpp", "Flächen_RB.cpp", "funktion.h", "RandPkt.h", "randpunkte.cpp". Eclipse erkennt automatisch die wichtigen Änderungen z.B. die Änderung des Quellcodes, deswegen liegen diese Änderungen dann direkt in "Staged Changes". Hier wird die alte Input-Datei(3D-Quader+Extrude_v2*.msh) gelöscht und die neue(Part3*.msh) eingefügt. Eclipse erkennt diese Änderungen auch, aber die Dateien liegen in "Unstaged Changes" und man kann sie aber von Hand nach "Staged Changes" legen.

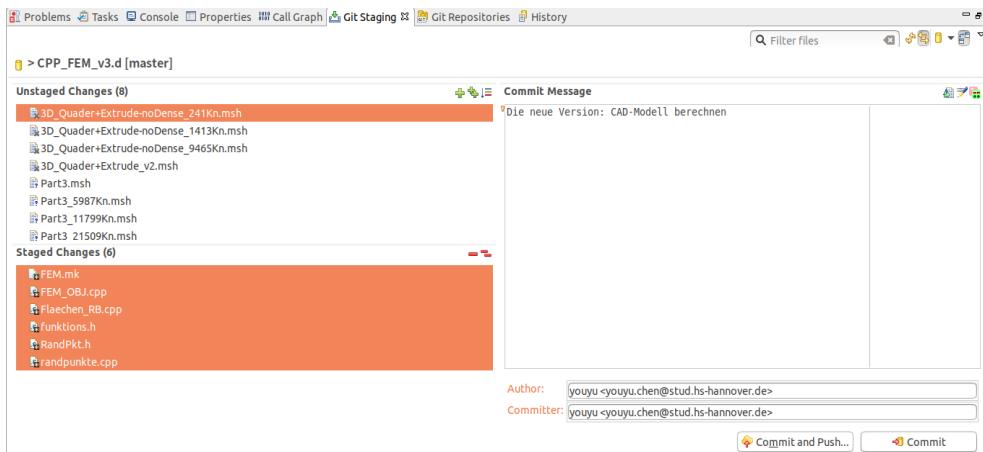


Abbildung 42: Git Repository8

Neue Informationen über die Version lassen sich im "Commit Message"-Fenster hineinschreiben. Am Ende einmal linke Maustastenklick an "Commit", die neue Version wird im Git Repository hochgeladen, das kann man in "History" sehen. Die Änderungen liegen an der rechten unteren Seite. Gelöschte Dateien werden durch ein schwarzes Kreuz markiert, neue eingefügte Dateien werden durch ein grünes Plus markiert, geänderter Quellcode wird direkt hineingelegt.

The screenshot shows a Git repository interface with the following details:

Commit History:

ID	Message	Author	Authored Date	Committer	Committed Date
f5ef605e5b6fc44ab03888492cb7c115d4e39dc	master [HEAD] Die neue Version: CAD-Modell berechnen	youyu	70 seconds ago	youyu	70 seconds ago
3af38b0	Die originale Version: Ein Würfel berechnen	youyu	7 minutes ago	youyu	7 minutes ago

Commit Details:

```

commit f5ef605e5b6fc44ab03888492cb7c115d4e39dc
Author: youyu <youyu.chen@stud.hs-hannover.de> 2018-09-27 00:50:00
Committer: youyu <youyu.chen@stud.hs-hannover.de> 2018-09-27 00:50:00
Parent: 3af38b05e4c84d4aa6e1e87adacc6786520ab579 (Die originale Version: Ein Würfel berechnen)
Branches: master

Die neue Version: CAD-Modell berechnen
  
```

File List:

- 3D_Quader+Extrude-noDense_1413Kn.msh
- 3D_Quader+Extrude-noDense_241Kn.msh
- 3D_Quader+Extrude-noDense_9465Kn.msh
- 3D_Quader+Extrude_v2.msh
- FEM.mk
- FEM_OBJ.cpp
- Flaechen_RB.cpp
- Part3.msh
- Part3_11799Kn.msh
- Part3_21509Kn.msh
- Part3_5987Kn.msh
- PandPkr.h

Abbildung 43: Git Repository9

9 Ergebnis und Zeitmessung

9.1 Ergebnis

Das BiCGSTAB-Verfahren ergibt eine approximative Lösung, die natürlich Fehler besitzt. Der Fehler ist an den inneren Punkten zu messen, da am Rand die Temperatur vorgegeben wird, ist dort der Fehler 0. Mit der Funktion

$$u(x, y, z) = 20 - 2 \cdot y^2 + x^3 \cdot y - x \cdot y^3 + z^3 \cdot x - z \cdot x^3$$

wird die Lösungsfunktion vorgegeben und das Programm damit überprüft, ob es bei der Randtemperaturvorgabe, die inneren Temperaturen korrekt berechnen kann. Im Vergleich zu der approximativen Lösung des C++-Programms, ist der Fehler dieses Algorithmus zu berechnen mit :

```

for(int i=0;i<Ndim.n_tetr_pkt;i++){
    if(RP.InnerePkt[i] < 0){
        // Lösung am Rand ausgeben
        xnetz=N.P[i][0];
        ynetz=N.P[i][1];
        znetz=N.P[i][2];
        u=20.0-2.0*pow(ynetz,2)+pow(xnetz,3)*ynetz-xnetz*pow(ynetz,3)+\
            pow(znetz,3)*xnetz-znetz*pow(xnetz,3);
        delta=u-optg.x[i];
        fin<<i<<" "<<xnetz<<" "<<ynetz<<" "<<znetz<<" "<<optg.x[i]<<" "<<u<<" "<<delta<<endl;
    }
    else{
        // Lösung in der inneren Punkte ausgeben
        xnetz=PTetr[j][0];
        ynetz=PTetr[j][1];
        znetz=PTetr[j][2];
        u=20.0-2.0*pow(ynetz,2)+pow(xnetz,3)*ynetz-xnetz*pow(ynetz,3)+\
            pow(znetz,3)*xnetz-znetz*pow(xnetz,3);
        delta=u-x[j];
        fin<<i<<" "<<xnetz<<" "<<ynetz<<" "<<znetz<<" "<<x[j]<<" "<<u<<" "<<delta<<endl;
        j++;
    }
}

```

Mit der Verfeinerung des Netzes wird der Fehler verkleinert und im Idealfall zu Null konvergieren. Die Lösungen und damit der Fehler an jedem Punkt wird durch *Gnuplot* mit einem Diagramm dargestellt.

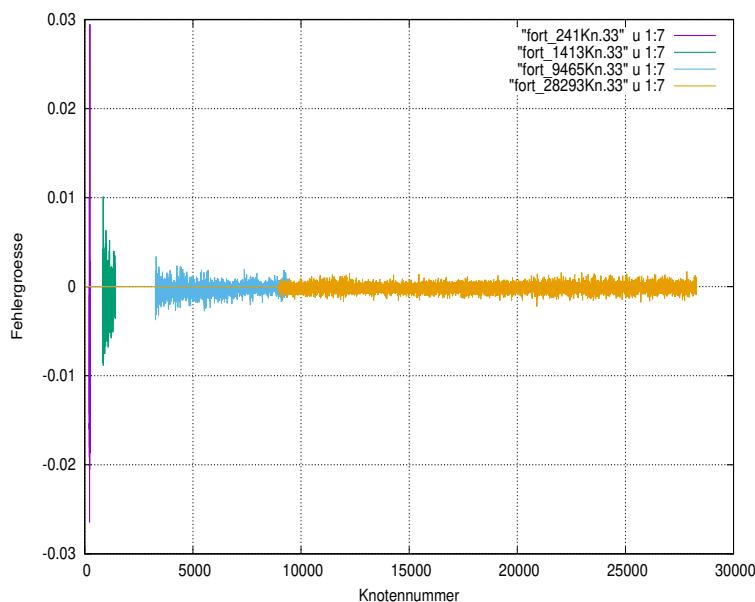


Abbildung 44: Fehler mit der Lösung eines Würfels für 241, 1413, 9465 und 28293 Knoten.

Dieses Diagramm 44 zeigt 4 Lösungen eines einfachen Würfels. Die Netze besitzen unterschiedliche Knotenzahl (jeweils 241, 1413, 9465, 28293 Knoten im Netz). Je größer die Anzahl der Knoten ist, desto kleiner wird der Fehler, d.h. die Genauigkeit erhöht sich mit dem dichteren Netz. Das C++ Programm kann auch mit einer allgemeinen Geometrie rechnen z.B. die im Kapitel ?? "Gmsh Einführung" gezeigte Welle. Der Fehler konvergiert im Idealfall zu Null.

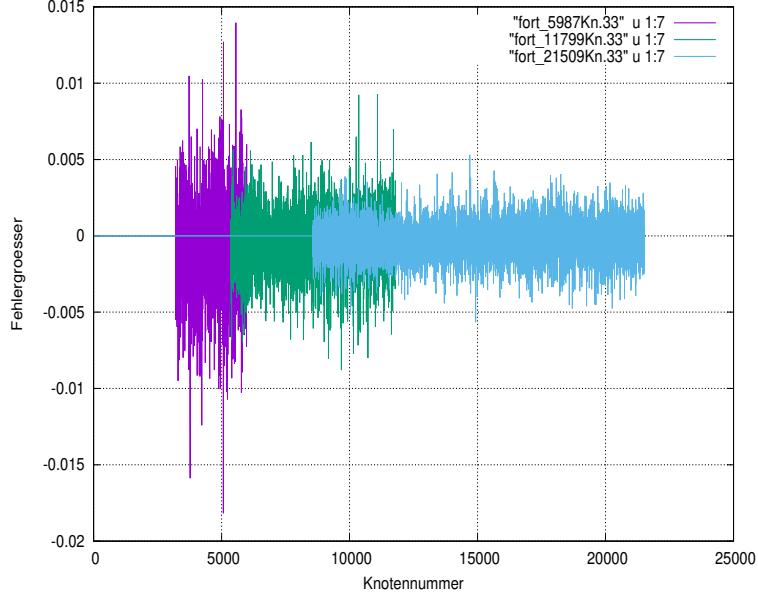


Abbildung 45: Fehler mit der Lösung einer Welle

9.2 Zeitmessung und Vergleich mit unterschiedlichen Versionen

Das C++ Programm wird ausgehend vom FORTRAN90-Programm erweitert, die Programmiersprache FORTRAN90/95 hat eine Optimierungsmöglichkeit für lineare Algebra Berechnungen, es ermöglicht nämlich die Matrixmultiplikation und das Skalarprodukt mit in FORTRAN90/95 definierten und aufrufbaren intrinsischen Funktionen auszuführen. Die beiden Funktionen werden direkt aufgerufen in dem FORTRAN90/95-Quellcode: *matmul()* und *dot_product()*, aber im C++ muss man diese Funktionen als Unterprogramm ausprogrammieren, um einen realistischen Vergleich zu ermöglichen. Daher haben die beiden Versionen einen sehr großen Unterschied. Um die beiden Programmiersprachen zu verglichen wird die Matrixmultiplikation und das Skalarprodukt auch zusätzlich als Unterprogramm in der FORTRAN90/95-Version ausprogrammiert.

Wie im Kapitel 5.1 "Gleichungssystem Optimierung" dargestellt, wird die Laufzeit sehr reduziert ohne Miteinbeziehung der Randpunkte in die Berechnung.

Betriebsumgebung:

- CPU: AMD Ryzen 5 1600X, 3.6GHz, 6 physikalische Kerne=12 virtuelle Kerne
- Compiler: GNU g++
- Compiler Option: -O3(Optimierungsstufe 3)
- Betriebssystem: Ubuntu 18.0

In der folgenden Tabelle werden die unterschiedlichen Laufzeiten gezeigt jeweils im C++ und FORTRAN mit 5 unterschiedlich großen Problemen:

Tabelle 2: Laufzeit in der FORTRAN-Version

Probe(Knotenanzahl)	Laufzeit V1(s)	Laufzeit V2(s)	Laufzeit V3(s)	Laufzeit V4(s)
241	6.84998930E-04	1.46199949E-03	1.67001039E-04	1.74999237E-04
1413	9.29810107E-02	5.65932035E-03	8.55699182E-03	2.40360051E-02
9465	8.37235737E+00	1.20530487E+02	3.27240181E+00	3.88815117E+01
23573	8.61457443E+01	1.31957812E+03	4.24411087E+01	6.13997620E+02
28293	1.39295898E+02	2.52778198E+03	4.98716583E+01	7.13806946E+02

Wobei:

- Version 1: *matmul()* und *dot_product()* nicht ausprogrammiert
- Version 2: *matmul()* und *dot_product()* ausprogrammiert
- Version 3: *matmul()* und *dot_product()* nicht ausprogrammiert und mit der Optimierung des Gleichungssystems
- Version 4: *matmul()* und *dot_product()* ausprogrammiert und mit der Optimierung des Gleichungssystems

Tabelle 3: Laufzeit in der C++-Version

Probe(Knotenanzahl)	Laufzeit V1(s)	Laufzeit V2(s)
241	2.63400000E-03	1.32000000E-04
1413	1.46437000E-01	2.50640000E-02
9465	1.22022990E+01	5.44957400E+00
23573	1.32038597E+02	6.38482500E+01
28293	1.98718498E+02	7.85788030E+01

wobei:

- Version 1: ohne der Optimierung des Gleichungssystems
- Version 2: mit der Optimierung des Gleichungssystems

Aus den zwei Tabellen ist zu ersehen, dass FORTRAN90/95 einen großen Vorteil bei der Berechnung eines großen linearen Gleichungssystems hat. Wenn die beiden Versionen unter gleichen Bedingung betrieben werden, läuft C++ schneller als FORTRAN. Außerdem ist es ersichtlich laut dieser Tabellen, dass die Optimierung des Gleichungssystems mehr als die Hälfte der Laufzeit reduziert.

Neben dem GNU GCC Compiler gibt es auch andere Compiler: *ifort*(für FORTRAN90) und *icc*(für C++). Diese Compiler stellt der Prozessorhersteller INTEL zur Verfügung, daher sind diese INTEL-Compiler für INTEL-Prozessoren optimiert. Hier werden die Programme C++ Version 2 und FORTRAN Version 4 als Probe genommen, um die Laufzeit in der INTEL-Umgebung zu vergleichen.

- CPU: Intel Xeon-Prozessor, 2.4 GHz, 8 physikalische Kerne=16 virtuelle Kerne
- Compiler : *icc* und *ifort*
- Compiler Option: -O3(Optimierungsstufe 3), -mkl(Mathe-Kernel-Library), -SSE4.2(Intel Xeon Vektorprozessoreinheit)
- Betriebssystem: Ubuntu und Suse

Tabelle 4: Laufzeit in der INTEL-Umgebung

Probe(Knotenanzahl)	Ubuntu	Suse			
		icc -O3	icc -O3 -xSSE4.2 - fast	ifort -O3 -mkl -xSSE4.2 - fast	ifort -O3 -mkl -xSSE4.2 - fast -fopenmp
241	2.57000000E-04	1.95000000E-04	1.54000000E-04	9.9999458E-04	1.0000020E-03
1413	4.22060000E-02	2.07580000E-02	2.22020000E-02	2.0999998E-02	2.0999998E-02
9465	8.23191900E+00	5.73119400E+00	5.56148800E+00	5.8050000E+00	5.7330000E+00
23573	1.07964235E+02	7.63073300E+01	6.93339940E+01	7.7725010E+01	7.7769000E+01
28293	1.31925299E+02	1.02195083E+02	8.98182430E+01	1.1283600E+02	9.6399161E+01

Dabei gibt es zwei zusätzliche Optionen:

- -O3: Optimierungsstufe 3
- -xSSE4.2: Vektorisierungseinheit mit einbeziehen
- -fast: kann die Laufzeit verbessern
- -fopenmp: aktivieren den Parallelizer um basierende auf den OpenMP*-Direktiven Multi-Threaded-Code zu generieren

Durch diese Tabelle wird es deutlich, dass der vom INTEL-Compiler übersetzte Code schneller ist als der vom GNU-Compiler kompilierte Code. Denn der Hersteller INTEL verbessert die Übersetzung in Maschinensprache besser für eigene Prozessoren mit für INTEL bekannten CPU-Eigenschaften.

Die OpenMP-Parallelisierung ist noch nicht optimal genug. Dies wird verursacht z.B. durch eine zu feingranulare Parallelisierung auf Skalarprodukt- oder Matrixmultiplikationsebene, wobei auch da noch nicht alle Teile im BiCGStab-Algorithmus durch die von Hand Parallelisierung erfaßt wurden. Ähnliches gilt für die INTEL-Compiler-Optionen für die Optimierung mit Compiler-Schaltern, da ist bei weitem noch nicht alles ausgereizt.

10 Fazit und Ausblick

10.1 Fazit

Bevor man ein Produkt einsetzt, muss man seine Betriebsumgebung analysieren und simulieren. Simulationstechnik spielt heutzutage eine immer wichtigere Rolle im Maschinenbau. Durch die FEM ergibt sich eine gute Lösung, die in vielen Maschinenbaubereichen angewendet werden kann, zum Beispiel : Festigkeits-Analyse, Wärmeleitungsprobleme, Schwingungsanalysen usw. Die Kerntheorie der FEM ist, dass eine komplexes Physikmodell in viele kleinere Teil-Modelle, nämlich "finite Element" zerlegt werden kann, und jedes einzelne finite Element wird mit einem Ansatz behandelt und dann wieder zum eigentlichen großem Modell assembliert, dann kann das Problem durch numerische Verfahren gelöst werden. Wärmeleitungsprobleme sind ganz normal im Maschinenbau z.B. beim Diesel-Motor eines Auto. Diese Arbeit behandelt eine Vorgehensweise mit der die Wärmeleitung in einem Bauteil berechnet werden kann. Dies wird mit Hilfe von 3 Werkzeugen realisiert:

- CAD-Software: um die Geometrie des Bauteils zu zeichnen
- Gmsh: um das Bauteil zu vernetzen
- ein C++ Programm: um das Problem zu bearbeiten

Das Input-Format für den Vernetzer `gmsh` muß ".stp" oder ".igs" sein⁸ damit `gmsh` dies erkennen und bearbeiten kann. Das C++ Programm hat Beschränkungen in der Elementanzahl, allein wegen des beschränkten Hauptspeichers dürfen nicht zu viele Knoten in das Modell hineingelegt werden. Deswegen ist es sehr wichtig, das Programm zu verbessern. Eine weitere Möglichkeit ist es, dass Programm durch den INTEL-Compiler im Quellcode optimieren zu lassen. Die andere Möglichkeit ist, dass man den Algorithmus selbst optimieren kann. Unter diesem Aspekt ist die Berechnung der Randpunkte im FEM-Modell unnötig, daher wird das Gleichungssystem nur für innere Punkten aufgestellt und gelöst, die Randpunkte wurden entfernt.

Ein großes Software Projekt wird nicht direkt von einer Person fertig programmiert, sondern schrittweise bearbeitet und vor allem von mehreren Mitarbeitern gleichzeitig verändert. Dazu ist es benötigt die Versionen zu verwalten und die die Beschreibung des Programm in Kommentaren zusammen zu packen. Eclipse-IDE und Doxygen liefern hiefür eine gute Unterstützungen. Eclipse-IDE kann die unterschiedliche Versionen eines Projekts verwalten, und Doxygen kann eine Dokumentation der Kommentare im Programm erzeugen. Mit Hilfe der Eclipse- und Doxygen-Programme wird die Teamarbeit eines Software Projekts erleichtert, die Kontrolle durch den Projektleiter verbessert und es ist besser möglich unterschiedliche Versionen zu pflegen.

10.2 Ausblick

Bei dem FORTRAN90/95- oder C++-Programm wird nur eine *Dirichletsche Randbedingung* benutzt, es gibt aber noch andere z.B. *Neumannsche Randbedingung* und *Cauchysche Randbedingung*. Man kann also die Programme noch mit anderen Randbedingungen erweitern. Das Programm basiert auf der Linux Umgebung, und man kann somit weiter an den Aufbau einer grafischen Benutzer Oberfläche denken. Schlussendlich kann das Programm auch unter MS-Windows oder MacOS funktionieren.

In manchen Situationen wird die Wärmeleitung nur auf ein Teilgebiet eines Bauteils konzentriert, dazu wird es in diesem Gebiete dichter vernetzt. Durch Gmsh gibt es eine Möglichkeit, dass das Bauteil um einen Punkt der Oberfläche herum dichter vernetzt werden kann. Aber der Punkt muss an der Oberfläche liegen, ansonst kann Gmsh das nicht schaffen. Das Problem wird in der Zukunft vielleicht durch andere Mesh Software realisiert.

⁸".stp" : Step-Format, ".igs" : Iges-Format.

Abbildungsverzeichnis

1	Gmsh Start	3
2	Gmsh Punkt	4
3	4 Punkte	5
4	Gmsh Linie	5
5	Gmsh Fläche	6
6	Gmsh 4 Flächen	6
7	Gmsh Volumen auswählen	7
8	Gmsh Volumen erzeugen	7
9	Gmsh Vernetzen	8
10	Welle	8
11	Welle im Gmsh	9
12	Einstellung der Elementgröße	9
13	Netz der Welle	10
14	Beispiel Geometrie	12
15	Kubische Geometrie mit 2 finiten Tetraederelementen mit Elementnumerierung und Knotennumerierung	22
16	Schritt1 "Projekt"	30
17	Schritt2 "Mode"	31
18	Schritt3 "Output"	32
19	Schritt4 "Diagrams"	33
20	Schritt5.1 "Expert"	34
21	der richtige Path-Name	35
22	der falsche Path-Name	35
23	Schritt5.2 "Expert"	36
24	Schritt6.1 "Run"	37
25	Schritt6.2 "Run"	38
26	Eclipse Installation	39
27	Import eines Projekts	40
28	Projekt Name	40
29	Dateien des Projekts	41
30	Eigenschaften des Projekts	41
31	Make Kommando1	42
32	Make Kommando2	42
33	Make in Console	42
34	ausführen	43
35	Git Repository1	43
36	Git Repository2	44
37	Git Repository3	44

38	Git Repository4	44
39	Git Repository5	45
40	Git Repository6	45
41	Git Repository7	46
42	Git Repository8	46
43	Git Repository9	47
44	Fehler mit der Lösung eines Würfels für 241, 1413, 9465 und 28293 Knoten.	48
45	Fehler mit der Lösung einer Welle	49

Tabellenverzeichnis

1	Variablen der lokalen Steifigkeitsmatrix	19
2	Laufzeit in der FORTRAN-Version	50
3	Laufzeit in der C++-Version	50
4	Laufzeit in der INTEL-Umgebung	51

A Anhang

A.1 Das C++ Programm

A.1.1 Hauptprogramm *FEM_OBJ.cpp*

```
/*
 * \file FEM.cpp
 * \brief Hauptprogramm FEM_Tetraeder
 *
 * \author Youyu Chen
 * \version 1.0
 * \date 09.08.2018
 */

#include<cmath>
#include<fstream>
#include<iostream>
#include<string>
#include"funktions.h"
using namespace std;
/// \brief main Funktion
int main(){
    Netzdim netzdim; //Netzdimension
    Netz netz; //Netz
    RandPkt RP; //Randpunkte
    GLS G; //Gleichungssystem
    OptGLS Opt; //optimiertes Gleichungssystem

    string inputfile="Part3.msh"; //Name des Inputfiles

    netzdim=dim(inputfile); //Netzdimension bestimmen
    cout<<"n_tetr_pkt="<<netzdim.n_tetr_pkt<<endl;
    cout<<"n_tetr="<<netzdim.n_tetr<<endl;

    netz=input(inputfile,netzdim); //Input-File einlesen

    RP=randpunkte(inputfile,netzdim, netz);
    cout<<"*****";
    cout<<"main : Randpunktnummern abgespeichert ! *";
    cout<<"*****";
    cout<<"pmRdim = "<<RP.pmRdim<<endl;

    G=SteifigkeitsMatrix(netzdim,netz); //globale Steifigkeismatrix und Rechte Seite b berechnen

    G=Flaechen_RB(netzdim, netz, RP, G);

    Opt=matopt(netzdim, netz, RP, G); //Matrix verkuerzen
    Opt.xmR=new double[RP.pmRdim]; // optimierte unbekante Vektor x dimensionieren
    bicgstab_solve(netzdim,netz,RP,Opt); //BiCG Verfahren aufrufen
    return 0;
}
```

A.1.2 Unterprogramm Netz dimensionieren *dim.cpp*

```
/*
 * \file dim.cpp
 * \brief Anzahl der Punkten und Tetraederelement bestimmen
 *
 * Ein *.msh File wird als Input eingelesen
 *
 * \author Youyu Chen
 * \version 1.0
```

```

* \date 09.08.2018
*/

#include<iostream>
#include<string>
#include<fstream>
#include"funktions.h"
using namespace std;
/// \brief Funktion um die Anzahl der Punkten zu bestimmen
/// \param &str Name des Inputfiles
/// \return geben die Klasse Netzdim(Netzdimension) zurück
Netzdim dim(string &str){
    int j; ///

```

A.1.3 Unterprogramm Netz einlesen *input.cpp*

```

/*
* \file input.cpp
* \brief Punkte und Tetraederelement einlesen
*
* Ein "*.msh" File wird als Input eingelesen
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include"funktions.h"
using namespace std;
/// \brief Funktion um alle Punkte einzulesen

```

```

/// \param &str Name der Input-File
/// \param netzdim Die Klasse Netzdime(Netzdimension)
Netz input(string &str,Netzdim netzdim){
Netz netz;
int idummy1=0; ///< idummy1: Element Index
int idummy2=0; ///< idummy2: Element Typ
int idummy3=0; ///< idummy3: Nummer der tags
int idummy4=0; ///< idummy4: <tags>
int idummy5=0; ///< idummy5: Knoten Nummer-List
int dim_element;
int j=0;
netz.P = new double*[netzdim.n_tetr_pkt]; ///< **p: Punktfeld dimensionieren
for (int i = 0; i < netzdim.n_tetr_pkt; i++)
{
netz.P[i] = new double[3];
}
netz.T=new int*[netzdim.n_tetr+1];
for(int i=0;i<netzdim.n_tetr+1;i++){
netz.T[i]=new int[4];
}
netz.T[0][0]=1;
netz.T[0][1]=2;
netz.T[0][2]=3;
netz.T[0][3]=4;
int idummy=0; ///< idummy: Punktenummer
double a,b,c,d; ///< a,b,c : XYZ-Koordinatenssystem wenn Punkte einlesen
//< a,b,c,d : 4 Knotennummer wenn Tetraeder einlesen
fstream fin(str.c_str());
if(fin){
string s;
for(int i=1;i<=5;i++){
getline(fin,s);
}
for(int i=0;i<netzdim.n_tetr_pkt;i++){
fin>>idummy>>a>>b>>c;getline(fin,s); // XY-Koordinatenssystem einlesen
netz.P[i][0]=a;
netz.P[i][1]=b;
netz.P[i][2]=c;
}
getline(fin,s);
getline(fin,s);
fin>>dim_element; // Anzahle aller Elemente einlesen
getline(fin,s);
for(int i=1;i<dim_element+1;i++){
fin>>idummy1>>idummy2>>idummy3>>idummy4>>idummy5;
if(idummy2==4){
j++;
fin>>a>>b>>c>>d; // 4 Knotennummer einlesen
netz.T[j][0]=a;
netz.T[j][1]=b;
netz.T[j][2]=c;
netz.T[j][3]=d;
}
getline(fin,s);
}
}
else
cerr<<"Fail"<<endl; // Fehlernmeldung
fin.close();
return netz;
}

```

A.1.4 Unterprogramm Randpunkte sortieren *randpunkte.cpp*

```

/*
 * \file randpunkte_PNummerF.cpp
 * \brief Randpunkte bestimmen

```

```

/*
 * \author Youyu Chen
 * \version 1.0
 * \date 09.08.2018
 */
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion um alle Randpunkte zu bestimmen \n
/// diese Funktion ist gleich wie randpunkte_InnerePkt() \n
/// der Unterschied dazwischen ist nur die Rückgabe
/// \param n_tetr_pkt Anzahl aller Punkten
/// \param NRanddim Dimension des Randpunkts
/// \param ***p Punkte mit Koordinatenssystem
/// \return geben die Klasse RandPkt(Randpunkte) zurück
RandPkt randpunkte(string &str, Netzdim ndim, Netz n){
RandPkt RP;
int dim_element;
int idummy1=0; //idummy1: Element Index
int idummy2=0; //idummy2: Element Typ
int idummy3=0; //idummy3: Nummer der tags
int idummy4=0; //idummy4: <tags>
int idummy5=0; //idummy5: Knoten Nummer-List
int a,b,c;
RP.pmRdim=0;
RP.InnerePkt=new int[ndim.n_tetr_pkt];
for(int i=0;i<ndim.n_tetr_pkt;i++)
RP.InnerePkt[i]=i+1;
fstream fin(str.c_str());
string s;
for(int i=0;i<ndim.n_tetr_pkt+7;i++){
getline(fin,s);
}
fin>>dim_element;getline(fin,s);
cout<<"s = "<<s<<"dim_element = "<<dim_element<<endl;
for(int i=0;i<dim_element;i++){
fin>>idummy1>>idummy2>>idummy3>>idummy4>>idummy5;
if(idummy2==15){
fin>>a;getline(fin,s);
if(RP.InnerePkt[a-1]>0)
RP.InnerePkt[a-1]=-1*RP.InnerePkt[a-1];
}
else if(idummy2==1){
fin>>a>>b;getline(fin,s);
if(RP.InnerePkt[a-1]>0)
RP.InnerePkt[a-1]=-1*RP.InnerePkt[a-1];
if(RP.InnerePkt[b-1]>0)
RP.InnerePkt[b-1]=-1*RP.InnerePkt[b-1];
}
else if(idummy2==2){
fin>>a>>b>>c;getline(fin,s);
if(RP.InnerePkt[a-1]>0)
RP.InnerePkt[a-1]=-1*RP.InnerePkt[a-1];
if(RP.InnerePkt[b-1]>0)
RP.InnerePkt[b-1]=-1*RP.InnerePkt[b-1];
if(RP.InnerePkt[c-1]>0)
RP.InnerePkt[c-1]=-1*RP.InnerePkt[c-1];
}
else
getline(fin,s);
}
ofstream fio;
fio.open("InnerePkt.txt",ios::out | ios::trunc);
for(int i=0;i<ndim.n_tetr_pkt;i++){
if(RP.InnerePkt[i]<0)
RP.pmRdim++;
fio<<i<<" "<<RP.InnerePkt[i]<<endl;
}
RP.pmRdim=ndim.n_tetr_pkt-RP.pmRdim;

```

```

    return RP;
}

```

A.1.5 Unterprogramm Steifigkeitsmatrix berechnen *SteifigkeitsMatrix.cpp*

```

/*
* \file SteifigkeitsMatrix.cpp
* \brief Steifigkeitsmatrix berechnen
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion um die Steifigkeitsmatrix zu berechnen
/// \param n_tetr_pkt Anzahl aller Punkte
/// \param n_tetr Anzahl der Tetraeder
/// \param **T Tetraeder Element
/// \param ***P Punkte mit Koordinatensystem
/// \return geben die Klasse GLS(Gleichungssystem) zurück
GLS SteifigkeitsMatrix(Netzdim ndim,Netz n){
    GLS g;
    g.bglobal=new double[ndim.n_tetr_pkt]; // rechte Seite \f$ \vec{b} \f$ dimensionieren
    g.Sglobal=new double*[ndim.n_tetr_pkt]; // Steifigkeitsmatrix dimensionieren
    for(int i=0;i<ndim.n_tetr_pkt;i++){
        g.Sglobal[i]=new double[ndim.n_tetr_pkt];
    }
    int Ndim=ndim.n_tetr_pkt; // Ndim: Dimension
    int ti;
    int l;
    int r;
    double det_Phi; // Determinante \f$ \phi \f$
    double one_by_Phi; // \f$ \frac{1}{\det \phi} \f$
    double a,b,c,d,e,f;
    double x21,x31,x41;
    double y21,y31,y41;
    double z21,z31,z41;
    double Pinv11,Pinv12,Pinv13;
    double Pinv21,Pinv22,Pinv23;
    double Pinv31,Pinv32,Pinv33;
    double q;
    double Slocal[4][4]; // locale Steifigkeitsmatrix
    double blocal[4]; // locale b-Matrix
    /// Slocal[4][4] initialisieren
    for(int i=0;i<Ndim;i++){
        for(int j=0;j<Ndim;j++){
            g.Sglobal[i][j]=0.0;
        }
    }
    /// blocal[4] initialisieren
    for(int j=0;j<Ndim;j++){
        g.bglobal[j]=0.0;
    }
    det_Phi=0.0;
    q=0.0;
    for(ti=0;ti<ndim.n_tetr;ti++){
        x21=n.P[ n.T[ti+1][1]-1 ][0] - n.P[ n.T[ti+1][0]-1 ][0]; // \f$ x_2 - x_1 \f$
        y21=n.P[ n.T[ti+1][1]-1 ][1] - n.P[ n.T[ti+1][0]-1 ][1]; // \f$ y_2 - y_1 \f$
        z21=n.P[ n.T[ti+1][1]-1 ][2] - n.P[ n.T[ti+1][0]-1 ][2]; // \f$ z_2 - z_1 \f$

        x31=n.P[ n.T[ti+1][2]-1 ][0] - n.P[ n.T[ti+1][0]-1 ][0]; // \f$ x_3 - x_1 \f$
        y31=n.P[ n.T[ti+1][2]-1 ][1] - n.P[ n.T[ti+1][0]-1 ][1]; // \f$ y_3 - y_1 \f$
        z31=n.P[ n.T[ti+1][2]-1 ][2] - n.P[ n.T[ti+1][0]-1 ][2]; // \f$ z_3 - z_1 \f$

        x41=n.P[ n.T[ti+1][3]-1 ][0] - n.P[ n.T[ti+1][0]-1 ][0]; // \f$ x_4 - x_1 \f$
        y41=n.P[ n.T[ti+1][3]-1 ][1] - n.P[ n.T[ti+1][0]-1 ][1]; // \f$ y_4 - y_1 \f$
```

```

z41=n.P[ n.T[ti+1][3]-1 ][2] - n.P[ n.T[ti+1][0]-1 ][2]; // \f$z_4-z_1\f$

det_Phi=x21*y31*z41+x31*y41*z21+x41*y21*z31 - x41*y31*z21-x31*y21*z41-x21*y41*z31;
if(det_Phi < 0.0){
cout<<"SteifigkeitsMatrix : det_Phi= "<<det_Phi<<endl;
cout<<"SteifigkeitsMatrix : Tetr.Nr= "<<ti<<endl;
cout<<"SteifigkeitsMatrix : negative Orientierung"<<endl;
cout<<"SteifigkeitsMatrix : ===== Exit"<<endl;
}
one_by_Phi = 1.0 / det_Phi;
Pinv11 = (y31 * z41 - y41 * z31);
Pinv12 = (x41 * z31 - x31 * z41);
Pinv13 = (x31 * y41 - x41 * y31);
Pinv21 = (y41 * z21 - y21 * z41);
Pinv22 = (x21 * z41 - x41 * z21);
Pinv23 = (x41 * y21 - x21 * y41);
Pinv31 = (y21 * z31 - y31 * z21);
Pinv32 = (x31 * z21 - x21 * z31);
Pinv33 = (x21 * y31 - x31 * y21);
a = Pinv11*Pinv11 + Pinv12*Pinv12 + Pinv13*Pinv13;
b = Pinv21*Pinv21 + Pinv22*Pinv22 + Pinv23*Pinv23;
c = Pinv31*Pinv31 + Pinv32*Pinv32 + Pinv33*Pinv33;
d = Pinv11*Pinv21 + Pinv12*Pinv22 + Pinv13*Pinv23;
e = Pinv11*Pinv31 + Pinv12*Pinv32 + Pinv13*Pinv33;
f = Pinv21*Pinv31 + Pinv22*Pinv32 + Pinv23*Pinv33;
Slocal[0][0]=a+b+c+2.0*(d+e+f);
Slocal[0][1]=-a-d-e;
Slocal[0][2]=-b-d-f;
Slocal[0][3]=-c-e-f;
Slocal[1][0]=Slocal[0][1];
Slocal[2][0]=Slocal[0][2];
Slocal[3][0]=Slocal[0][3];
Slocal[1][1]=a;
Slocal[1][2]=d;
Slocal[1][3]=e;
Slocal[2][1]=d;
Slocal[2][2]=b;
Slocal[2][3]=f;
Slocal[3][1]=e;
Slocal[3][2]=f;
Slocal[3][3]=c;
for(int i=0;i<4;i++){
for(int j=0;j<4;j++){
Slocal[i][j]=Slocal[i][j]*one_by_Phi; // vermeintlicher Fehler <== ELMER
}
}
for(int i=0;i<4;i++){
blocal[i]=1.0;
}
for(int i=0;i<4;i++){
l=n.T[ti+1][i]-1;
for(int j=0;j<i+1;j++){
r=n.T[ti+1][j]-1;
g.Sglobal[1][r]=g.Sglobal[1][r]+Slocal[i][j];
g.Sglobal[r][1]=g.Sglobal[1][r];
}
g.bglobal[1]=g.bglobal[1]+det_Phi*blocal[i];
}
}
cout<<" SteifigkeitsMatrix : Kubisch.Loesungsfunktionsvorgabe "<<endl;
q=4.0; // \f$u(x,y,z) = 20 - 2*y^2 + x^3*y - x*y^3 + z^3*x - z*x^3 \f$

for(int i=0;i<Ndim;i++){
/// Rechte Seite * q=4 und die 1/6 der Steifigkeitsmatrix auf die andere Seite
g.bglobal[i]=q/24.0*g.bglobal[i];
}
for(int i=0;i<Ndim;i++){
for(int j=0;j<Ndim;j++){
g.Sglobal[i][j]=1.0/6.0*g.Sglobal[i][j];
}
}
return g;
}

```

A.1.6 Unterprogramm Gleichungssystem optimieren *matopt.cpp*

```
/*
 * \file matopt.cpp
 * \brief Steifigkeitsmatrix und rechte Seite b verkürzen
 *
 * \author Youyu Chen
 * \version 1.0
 * \date 09.08.2018
 */

#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include<time.h>
#include<stdlib.h>
#include<limits>
#include <iomanip>
#include"functions.h"
using namespace std;
/// \brief Funktion um die Matrix zu verkürzen
/// \param ndim Dimension des Netzes
/// \param n Netz
/// \param RP Randpunkte
/// \param GLS Gleichungssystem
/// \return geben die Klasse OptGLS(Optimierte Gleichungssystem) zurück
OptGLS matopt(Netzdim ndim, Netz n, RandPkt RP, GLS g){
OptGLS optg; //;< optg: optimiertes Gleichungssystem

int j=0;
int m=0;
int r=0;

clock_t st0,ste; //;< st0,ste : Zeit Messung für die Optimierung
double cpu_sekunden; //;< cpu_sekunden : Zeitspanne der Optimierung

double **tmp=new double *[RP.pmRdim]; //;< **tmp : Zwischen Matrix
for(int i=0;i<RP.pmRdim;i++){
tmp[i]=new double[ndim.n_tetr_pkt];
}
double *tmp2=new double [RP.pmRdim]; //;< **tmp2: Zwischen Matrix

optg.PmR=new double *[RP.pmRdim]; //;< optimierte Punktematrix PmR
for(int i=0;i<RP.pmRdim;i++){
optg.PmR[i]=new double[3];
}
optg.bmR=new double [RP.pmRdim]; //;< optimierte rechte Seite \f$ \overrightarrow{bmR}\f$
optg.SmR=new double *[RP.pmRdim]; //;< optimierte Steifigkeitsmatrix SmR
for(int i=0;i<RP.pmRdim;i++){
optg.SmR[i]=new double[RP.pmRdim];
}
optg.x=new double[ndim.n_tetr_pkt]; //;< unbekannte Vektor \f$ \vec{x} \f$
for(int i=0;i<ndim.n_tetr_pkt;i++){
optg.x[i]=0.0; //;< \f$ \vec{x} \f$ initialisieren
}
j=0;
m=0;
cout<<" Matrix verkürzen....." << endl;
st0=clock();
for(int i=0;i<ndim.n_tetr_pkt;i++){
if(RP.InnerePkt[i]<0){
optg.x[i]=g.bglobal[i]; //;< die Temperatur am Rand ist schon bekannt
j=j+1;
}
else{
optg.bmR[m]=g.bglobal[i];
for(int q=0;q<ndim.n_tetr_pkt;q++){
tmp[m][q]=g.Sglobal[i][q];
}
}
}
}
```

```

/// Innere Punkte neu nummerieren ==> Punktematrix verkürzen
optg.PmR[m][0]=n.P[i][0];
optg.PmR[m][1]=n.P[i][1];
optg.PmR[m][2]=n.P[i][2];
m++;
}
}
tmp2=matmulvkt(tmp,optg.x,RP.pmRdim,ndim.n_tetr_pkt,ndim.n_tetr_pkt);
optg.bmR=vktminus(optg.bmR,tmp2,RP.pmRdim,RP.pmRdim);
m=0;
r=0;
for(int i=0;i<ndim.n_tetr_pkt;i++){
for(j=0;j<ndim.n_tetr_pkt;j++){
if(RP.InnerePkt[i] > 0 && RP.InnerePkt[j] > 0){
optg.SmR[m][r]=g.Sglobal[RP.InnerePkt[i]-1][RP.InnerePkt[j]-1];
r++;
if(r>=RP.pmRdim){
r=0;
m++;
}
}
}
}
ste=clock();
cpu_sekunden=(double)(ste-st0)/CLOCKS_PER_SEC; // Zeitdauer der Optimierung
cout<<" ==> Matrix optimiert : Dazu wurden "<<cpu_sekunden<<" CPU--Sekunden benoetigt."<<endl;
return optg;
}

```

A.1.7 Unterprogramm BiCGSTAB Verfahren *bicgstab_solve.cpp*

```

/*
* \file bicgstab_solve.cpp
* \brief Das BiCG-Verfahren
*
*
* Ein iteratives numerisches Verfahren
* zur approximativen Lösung eines linearen Gleichungssystems:
* \f$ A \vec{x} = \vec{b}, A \in \mathbb{R}^{n \times n} \f$
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include<time.h>
#include<stdlib.h>
#include<limits>
#include <iomanip>
#include "funktions.h"

using namespace std;
/// \brief Funktion des BiCG-Verfahrens
/// \param ndim Anzahl der Matrix und des Vektors
/// \param **a 2-Dimension Matrix(linke Seite)
/// \param *b Vektor(rechte Seite)
/// \param *x unbekannter Vektor
/// \param **Ptetr Punktnummern im Tetraeder
/// \return Keine Rückgabe, aber ein Gnuplot-Skript wird aufgerufen um den Fehler zu sehen
int bicgstab_solve(Netzdim Ndim, Netz N, RandPkt RP, OptGLS optg){
double alpha; // alpha: Reelle Zahl \f$ \alpha \f$
double beta; // beta: Reelle Zahl \f$ \beta \f$
double delta; // delta: Fehler \f$ \Delta \f$
double omega; // omega: Reelle Zahl \f$ \omega \f$
double rho; // rho: Reelle Zahl \f$ \rho \f$
double rho_alt; // rho_alt: Reelle Zahl \f$ \rho_{alt} \f$
double eps; // eps: Genauigkei \f$ \epsilon \f$
double u; // u: Standard Lösung

```

```

double xnetz; ///< xnetz: X-Koordinaten
double ynetz; ///< ynetz: Y-Koordinaten
double znetz; ///< znetz: Z-Koordinaten

int ndim=RP.pmRdim; ///< ndim : optimierte Dimension
double **a=optg.SmR; ///< **a : Linke Seite Matrix
double *b=optg.bmR; ///< *b : Rechte Seite Vektor
double *x=optg.xmR; ///< *x : Unbekannter Vektor
double **PTetr=optg.PmR; ///< **PTetr : optimierte Punkte

double *p=new double[ndim]; ///< *p: Vektor \f$ \vec p\f$
double *r=new double[ndim]; ///< *r: Vektor \f$ \vec r\f$
double *r0=new double[ndim]; ///< *r0: Vektor \f$ \vec {r_0}\f$
double *v=new double[ndim]; ///< *v: Vektor \f$ \vec v\f$
double *s=new double[ndim]; ///< *s: Vektor \f$ \vec s\f$
double *t=new double[ndim]; ///< *t: Vektor \f$ \vec t\f$

double *tmp1=new double [ndim]; ///< *tmp1: Zwischen Vektor \f$ \overrightarrow{tmp1}\f$
double *tmp2=new double [ndim]; ///< *tmp2: Zwischen Vektor \f$ \overrightarrow{tmp2}\f$

int it; //< it: Zähler für Iterationsschleife
clock_t st0,ste; //< st0,ste: Zeit Messung des Verfahrens
double cpu_sekunden; //< cpu_sekunden: Zeitdauer des Verfahrens
cout<<" BiCGStab_solve : ndim= "<<ndim<<endl;
eps=numeric_limits<double>::epsilon(); //> Genauigkeit der double Zahl
eps=eps*0.01;
eps=eps*0.01;
cout<<" BiCGStab_solve : relative machine precision epsilon eps= "<<eps<<endl;
cout<<" BiCGStab_solve : Vor Conjugate-Gradient-Algorithmus"<<endl;
/// BiCG-Verfahren anfangen:
st0=clock();
/// Initialisieren \f$ \vec x=3.0\f$
for(int i=0;i<ndim;i++){
x[i]=3.0;
}
p=vktminus(b,matmulvkt(a,x,ndim,ndim),ndim,ndim); //> Setze \f$ \vec p=\vec b-A*\vec x\f$
r=vktminus(b,matmulvkt(a,x,ndim,ndim),ndim,ndim); //> Setze \f$ \vec b=\vec b-A*\vec x\f$
r0=r; //> Setze \f$ \vec r_0=\vec r\f$
rho=dot_product(r,r,ndim,ndim); //> Setze \f$ \rho = \vec r \cdot \vec r\f$
it=0;
cout<<" BiCGStab_solve : Vor der Iterationsschleife"<<endl;
/// Iterationsschleife anfangen:
while(dot_product(r,r,ndim,ndim) > eps){ //> proof \f$ \vec r \cdot \vec r > \epsilon ?\f$
v=matmulvkt(a,p,ndim,ndim,ndim); //> \f$ \vec v=A*\vec p\f$
alpha=rho/dot_product(v,r0,ndim,ndim); //> \f$ \alpha = \rho / (\vec v \cdot \vec r_0)\f$
s=vktminus(r,constmulvkt(alpha,v,ndim),ndim,ndim); //> \f$ \vec s=\vec r - \alpha \vec v\f$
t=matmulvkt(a,s,ndim,ndim,ndim); //> \f$ \vec t = A*\vec s\f$
//> \f$ \omega = \frac{\vec v \cdot \vec r}{\vec v \cdot \vec r_0}\f$
omega=dot_product(t,s,ndim,ndim)/dot_product(t,t,ndim,ndim);
x=vktplus(x,constmulvkt(alpha,p,ndim),ndim,ndim);
//> \f$ \vec x_{k+1} = \vec x_k + \alpha \vec p + \omega \vec s\f$
x=vktplus(x,constmulvkt(omega,s,ndim),ndim,ndim);
//> \f$ \vec r = \vec s - \omega \vec v\f$
r=vktminus(s,constmulvkt(omega,t,ndim),ndim,ndim);
rho_alt=rho; //> \f$ \rho_{k+1} \quad \text{speichern}\f$
rho=dot_product(r,r0,ndim,ndim); //> \f$ \rho_{k+1} = \vec r \cdot \vec r_0\f$
beta=alpha/omega*rho_rho_alt; //> \f$ \beta = \alpha / \omega \rho_{k+1}\f$
tmp1=vktminus(p,constmulvkt(omega,v,ndim),ndim,ndim);
tmp2=constmulvkt(beta,tmp1,ndim);
p=vktplus(r,tmp2,ndim,ndim); //> \f$ \vec p = \vec r + \beta (\vec p - \omega \vec v)\f$
it++;
cout.setf(ios::scientific);
cout<<"BiCGStab_solve : it = "<<it<<" proof = "<<setprecision(8)<<dot_product(r,r,ndim,ndim)<<endl;
}
cout<<"BiCGStab_solve : Nach der Iterationsschleife"<<endl;
ste=clock();
cpu_sekunden=(double)(ste-st0)/CLOCKS_PER_SEC; //> Zeitdauer des Verfahrens
cout<<"BiCGStab_solve : Nach Conjugate-Gradient-Algorithmus"<<endl;
if(it>ndim){
cout<<"BiCGStab_solve : Nicht konvergiert --> it ="<<it<<endl;
}
else{
cout<<"BiCGStab_solve : konvergiert --> it ="<<it<<endl;
}

```

```

cout<<" BiCGStab_solve : Dazu wurden  "<<cpu_sekunden<<" CPU--Sekunden benoetigt."<<endl;
ofstream fin;
fin.open("fort.33",ios::out | ios::trunc); // Die Lösung wird in "fort.33" ausgegeben
int j=0;

for(int i=0;i<Ndim.n_tetr_pkt;i++){
if(RP.InnerePkt[i] < 0){
/// Lösung am Rand ausgeben
xnetz=N.P[i][0];
ynetz=N.P[i][1];
znetz=N.P[i][2];
u=20.0-2.0*pow(ynetz,2)+pow(xnetz,3)*ynetz-xnetz*pow(ynetz,3)+\
    pow(znetz,3)*xnetz-znetz*pow(xnetz,3);
delta=u-optg.x[i];
fin<<i<<" "<<xnetz<<" "<<ynetz<<" "<<znetz<<" "<<optg.x[i]<<" "<<u<<" "<<delta<<endl;
}
else{
/// Lösung in der inneren Punkte ausgeben
xnetz=PTetr[j][0];
ynetz=PTetr[j][1];
znetz=PTetr[j][2];
u=20.0-2.0*pow(ynetz,2)+pow(xnetz,3)*ynetz-xnetz*pow(ynetz,3)+\
    pow(znetz,3)*xnetz-znetz*pow(xnetz,3);
delta=u-x[j];
fin<<i<<" "<<xnetz<<" "<<ynetz<<" "<<znetz<<" "<<x[j]<<" "<<u<<" "<<delta<<endl;
j++;
}

}
fin.close();
/// Gnuplot-Skript "cg_Tetraed.dem" aufrufen
system("gnuplot cg_Tetraed.dem");
return 0;
}

```

A.1.8 Unterprogramm Skalarprodukt *dot_product.cpp*

```

/*
* \file dot_product.cpp
* \brief Skalarprodukt
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/

#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion Skalarprodukt
/// \param *a Vektor a
/// \param *b Vektor b
/// \param arow Dimension des Vektors a
/// \param brow Dimension des Vektors b
/// \return geben die Lösung  $\vec{a} \cdot \vec{b}$  zurück
double dot_product(double *a,double *b,int arow,int brow){
double out=0; // out: Die Lösung
if(arow==brow){
for(int i=0;i<arow;i++){
out+=a[i]*b[i]; //  $\sum_{i=1}^n a_i b_i$ 
}
}
else
cerr<<" Vecor arow!= brow"<<endl;
return out;
}

```

A.1.9 Unterprogramm Matrix multipliziert mit einem Vektor *matmulvkt.cpp*

```
/*
* \file matmulvkt.cpp
* \brief Ein Unterprogramm für Matrix*Vektor
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion Matrix*Vektor \n
/// Anzahl des Matrix-Spalts soll gleich die Vektor-Dimension sein
/// \param **a Matrix a
/// \param *b Vektor b
/// \param arow Anzahl der Zeile a
/// \param acol Anzahl des Spalts a
/// \param brow Vektor-Dimension
/// \return geben einen neuen Vektor zurück
double *matmulvkt(double **a,double *b,int arow,int acol,int brow){
    double *out=new double[arow]; /////< *out: der Lösung-Vektor
    for(int i=0;i<arow;i++){
        out[i]=0.0;
    }
    if(acol==brow){
        for(int i=0;i<arow;i++){
            for(int j=0;j<acol;j++){
                out[i]+=a[i][j]*b[j]; //\f$ \overrightarrow{out}=A*\vec{b}\f$
            }
        }
    }
    else
        cerr<<"acol!=brow"<<endl; // Fehlernachricht
    return out;
}
```

A.1.10 Unterprogramm ein Konstant multipliziert mit einem Vektor *constmulvkt.cpp*

```
/*
* \file constmulvkt.cpp
* \brief Ein Unterprogramm für konstant*Vektor
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion Konstant*Vektor
/// \param a ein Konstant
/// \param *b Vektor
/// \param brow Dimension des Vektors
/// \return geben den Lösung Vektor zurück
double *constmulvkt(double a,double *b,int brow){
    double *out=new double[brow]; /////< *out: der Lösung-Vektor
    for(int i=0;i<brow;i++){
        out[i]=a*b[i]; //\f$ out_i=a*b_i \f$
    }
}
```

```

    return out;
}

```

A.1.11 Unterprogramm ein Vektor plus ein Vektor *vktplus.cpp*

```

/*
* \file vktplus.cpp
* \brief Ein Unterprogramm für  $\vec{a} + \vec{b}$ 
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion  $\vec{a} + \vec{b}$ 
///
/// Dimension  $\vec{a}$  soll gleich wie  $\vec{b}$  sein
/// \param *a Vektor
/// \param *b Vektor
/// \param arow Dimension des Vektors a
/// \param brow Dimension des Vektors b
/// \return geben den Lösung Vektor zurück
/// \see vktminus()
double *vktplus(double *a,double *b,int arow,int brow){
double *out=new double [arow]; // < *out: der Lösung-Vektor
if(arow==brow){
for(int i=0;i<arow;i++){
out[i]=a[i]+b[i]; //  $out_i = a_i + b_i$ 
}
}
else
cerr<<"arow!=brow"<<endl; // Fehlermeldung
return out;
}

```

A.1.12 Unterprogramm ein Vektor minus ein Vektor *vktminus.cpp*

```

/*
* \file vktminus.cpp
* \brief Ein Unterprogramm für  $\vec{a} - \vec{b}$ 
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<iostream>
#include<string>
#include<fstream>
#include<cmath>
#include"funktions.h"
using namespace std;
/// \brief Funktion  $\vec{a} - \vec{b}$ 
///
/// Dimension  $\vec{a}$  soll gleich wie  $\vec{b}$  sein
/// \param *a Vektor
/// \param *b Vektor
/// \param arow Dimension des Vektors a
/// \param brow Dimension des Vektors b
/// \return geben den Lösung Vektor zurück
/// \see vktplus()
double *vktminus(double *a,double *b,int arow,int brow){
double *out=new double [arow]; // < *out: der Lösung-Vektor
if(arow==brow){

```

```

for(int i=0;i<arow;i++){
out[i]=a[i]-b[i]; //\f$ out_i=a_i*b_i \f$
}
}
else
cerr<<"arow!=brow"<<endl; // Fehlermeldung
return out;
}

```

A.1.13 Unterprogramm Gnuplot Skript *cg_Tetraed.dem*

```

set term x11
#set term epslatex color "Times-Roman" 14
#
#set parametric
#set style data linespoints
#
set xlabel "$x\$ \$\\rightarrowtail\$"
set ylabel "$y\$ \$\\rightarrowtail\$"
#set xlabel "x -->"
#set ylabel "y -->"
#
set samples 100
set grid
#
unset key
#
plot "fort.33" using 1:5 with lines , "fort.33" using 1:6 with lines
pause -1 "Hit return to continue"
plot "fort.33" using 1:7 with lines
pause -1 "Hit return to continue"
#plot "fort.33" using 1:7 with lines, "924Kn_Ergebn.d/fort.33" using 1:7 with lines, "470Kn_Ergebn.d/fort.33" using 1:7 with
#pause -1 "Hit return to continue"

```

A.1.14 Include File *funktions.h*

```

/*
* \file funktions.h
* \brief Funktionen declaration
*
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/
#include<string>
#include<iostream>
#include"Netzdim.h"
#include"Netz.h"
#include"RandPkt.h"
#include"GLS.h"
#include"OptGLS.h"
using namespace std;
Netzdim dim(string &str);
Netz input(string &str,Netzdim netzdim);
RandPkt randpunkte(string &str, Netzdim ndim, Netz n);
GLS SteifigkeitsMatrix(Netzdim ndim,Netz n);
GLS Flaechen_RB( Netzdim ndim, Netz n, RandPkt RP, GLS g);
OptGLS matopt(Netzdim ndim, Netz n, RandPkt RP, GLS g);
int bicgstab_solve(Netzdim Ndim, Netz N, RandPkt RP, OptGLS optg);
double *matmulvkt(double **a,double *b,int arow,int acol,int brow);
double dot_product(double *a,double *b,int arow,int brow);
double *vktminus(double *a,double *b,int arow,int brow);
double *vktplus(double *a,double *b,int arow,int brow);
double *constmulvkt(double a,double *b,int brow);

```

A.1.15 Include File *GLS.h*

```
/*
 * \file GLS.h
 * \brief Klasse Gleichungssystem
 *
 *
 * \author Youyu Chen
 * \version 1.0
 * \date 09.08.2018
 */

#include<iostream>
using namespace std;
class GLS{
public:
    int ndim;           ///ndim      : Dimension
    double *bglobal;   ///*bglobal : globale rechte Seite  $\vec{b}$ 
    double **Sglobal;  ///**Sglobal: globale Steifigkeitsmatrix
};
```

A.1.16 Include File *Netzdim.h*

```
/*
 * \file Netzdim.h
 * \brief Klasse Dimension eines Netzes
 *
 *
 * \author Youyu Chen
 * \version 1.0
 * \date 09.08.2018
 */

#include<iostream>
using namespace std;
class Netzdim{
public:
    int n_tetr_pkt;  ///n_tetr_pkt : Anzahl aller Punkte
    int n_tetr;      ///n_tetr     : Anzahl der Tetraederelement
};
```

A.1.17 Include File *Netz.h*

```
/*
 * \file Netz.h
 * \brief Klasse Netz
 *
 *
 * \author Youyu Chen
 * \version 1.0
 * \date 09.08.2018
 */

#include<iostream>
using namespace std;
class Netz{
public:
    double **P;  ///**P: Koordinatensystem der Punkte
    int    **T;  ///**T: Tetraederelement(besteh aus 4 Knotennummer)
};
```

A.1.18 Include File *OptGLS.h*

```
/*!
```

```

* \file OptGLS.h
* \brief Klasse das optimierte Gleichungssystem
*
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/

```

```

#include<iostream>
using namespace std;
class OptGLS{
    public:
        int ndim;      ///< ndim : Dimension
        double *bmR;   ///< *bmR : optimierte \f$ \vec b \f$
        double **SmR;  ///< **SmR : optimierte Steifigkeitsmatrix
        double *x;     ///< *x : unbekannter Vektor \f$ \vec x \f$
        double *xmR;   ///< *xmR : optimierter unbekannter Vektor \f$ \vec x \f$
        double **PmR;  ///< **PmR : neu nummerierte innere Punkte
};

```

A.1.19 Include File *RandPkt.h*

```

/*!
* \file RandPkt.h
* \brief Klasse Randpunkte
*
*
* \author Youyu Chen
* \version 1.0
* \date 09.08.2018
*/

```

```

#include<iostream>
using namespace std;
class RandPkt{
    public:
        int *InnerePkt; //;< *InnerePkt : Innere Punkte
        int pmRdim;     //;< pmRdim : Anzahle aller innere Punkte
};

```

A.2 Doxygen Dokumentation Datei

FEM_Tetraeder

Erzeugt von Doxygen 1.8.13

Inhaltsverzeichnis

1 Datei-Verzeichnis	1
1.1 Auflistung der Dateien	1
2 Datei-Dokumentation	3
2.1 bicgstab_solve.cpp-Dateireferenz	3
2.1.1 Ausführliche Beschreibung	4
2.1.2 Dokumentation der Funktionen	4
2.1.2.1 bicgstab_solve()	4
2.2 FEM_OBJ.cpp-Dateireferenz	7
2.2.1 Ausführliche Beschreibung	7
2.2.2 Dokumentation der Funktionen	8
2.2.2.1 main()	8
2.3 funktions.h-Dateireferenz	9
2.3.1 Ausführliche Beschreibung	10
2.3.2 Dokumentation der Funktionen	11
2.3.2.1 bicgstab_solve()	11
2.3.2.2 constmulvkt()	14
2.3.2.3 dim()	14
2.3.2.4 dot_product()	14
2.3.2.5 Flaechen_RB()	15
2.3.2.6 input()	15
2.3.2.7 matmulvkt()	16
2.3.2.8 matopt()	16
2.3.2.9 randpunkte()	17
2.3.2.10 SteifigkeitsMatrix()	17
2.3.2.11 vktminus()	18
2.3.2.12 vktplus()	18
2.4 matmulvkt.cpp-Dateireferenz	18
2.4.1 Ausführliche Beschreibung	19
2.4.2 Dokumentation der Funktionen	19
2.4.2.1 matmulvkt()	19
Index	21

Kapitel 1

Datei-Verzeichnis

1.1 Auflistung der Dateien

Hier folgt die Aufzählung aller Dateien mit einer Kurzbeschreibung:

bicgstab_solve.cpp	Das BiCG-Verfahren	3
FEM_OBJ.cpp	Hauptprogramm FEM_Tetraeder	7
funktions.h	Funktionen declaration	9
matmulvkt.cpp	Ein Unterprogramm für Matrix*Vektor	18

Kapitel 2

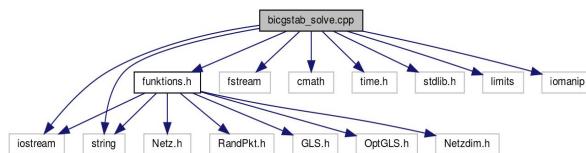
Datei-Dokumentation

2.1 bicgstab_solve.cpp-Dateireferenz

Das BiCG-Verfahren.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include <time.h>
#include <stdlib.h>
#include <limits>
#include <iomanip>
#include "funktions.h"
#include "bicgstab_solve.cpp"
```

Include-Abhängigkeitsdiagramm für bicgstab_solve.cpp:



Funktionen

- int **bicgstab_solve** (Netzdim Ndim, Netz N, RandPkt RP, OptGLS optg)

Funktion des BiCG-Verfahrens.

2.1.1 Ausführliche Beschreibung

Das BiCG-Verfahren.

Ein iteratives numerisches Verfahren zur approximativen Lösung eines linearen Gleichungssystems: $A * \vec{x} = \vec{b}$, $A \in \mathbb{R}^{n*n}$

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.1.2 Dokumentation der Funktionen

2.1.2.1 bicgstab_solve()

```
int bicgstab_solve (
    Netzdim Ndim,
    Netz N,
    RandPkt RP,
    OptGLS optg )
```

Funktion des BiCG-Verfahrens.

Parameter

<i>ndim</i>	Anzahl der Matrix und des Vektors
<i>**a</i>	2-Dimension Matrix(linke Seite)
<i>*b</i>	Vektor(rechte Seite)
<i>*x</i>	unbekannter Vektor
<i>**PTetr</i>	Punktnummern im Tetraeder

Rückgabe

Keine Rückgabe, aber ein Gnuplot-Skript wird aufgerufen um den Fehler zu sehen

< alpha: Reelle Zahl α

< beta: Reelle Zahl β

```
< delta: Fehler  $\Delta$ 
< omega: Reelle Zahl  $\omega$ 
< rho: Reelle Zahl  $\rho$ 
< rho_alt: Reelle Zahl  $\rho_{alt}$ 
< eps: Genauigkeit  $\epsilon$ 
< u: Standard Lösung
< xnetz: X-Koordinaten
< ynetz: Y-Koordinaten
< znetz: Z-Koordinaten
< ndim : optimierte Dimension
< **a : Linke Seite Matrix
< *b : Rechte Seite Vektor
< *x : Unbekannter Vektor
< **PTetr : optimierte Punkte
< *p: Vektor  $\vec{p}$ 
< *r: Vektor  $\vec{r}$ 
< *r0: Vektor  $\vec{r}_0$ 
< *v: Vektor  $\vec{v}$ 
< *s: Vektor  $\vec{s}$ 
< *t: Vektor  $\vec{t}$ 
< *tmp1: Zwischen Vektor  $\vec{tmp1}$ 
< *tmp2: Zwischen Vektor  $\vec{tmp2}$ 
< it: Zähler für Iterationsschleife
< st0,ste: Zeit Messung des Verfahrens
< cpu_sekunden: Zeitdauer des Verfahrens
Genauigkeit der double Zahl
BiCG-Verfahren anfangen:
Initialisieren  $\vec{x} = 3.0$ 
Setze  $\vec{p} = \vec{b} - A * \vec{x}$ 
Setze  $\vec{b} = \vec{b} - A * \vec{x}$ 
```

Erzeugt von Doxygen

Setze $\vec{r}_0 = \vec{r}$

Setze $\rho = \vec{r} \cdot \vec{r}$

Iterationsschleife anfangen:

proof $\vec{r} \cdot \vec{r} > \epsilon?$

$\vec{v} = A * \vec{p}$

$\alpha = \rho / (\vec{v} \cdot \vec{r}_0)$

$\vec{s} = \vec{r} - \alpha * \vec{v}$

$\vec{t} = A * \vec{s}$

$\omega = \frac{\vec{t} \cdot \vec{s}}{\vec{t} \cdot \vec{t}}$

$\vec{x}_{k+1} = \vec{x}_k + \alpha * \vec{p} + \omega * \vec{s}$

$\vec{r} = \vec{s} - \omega * \vec{t}$

ρ_k speichern

$\rho_{k+1} = \vec{r} \cdot \vec{r}_0$

$\beta = \alpha / \omega * \rho_{k+1} / \rho_k$

$\vec{p} = \vec{r} + \beta * (\vec{p} - \omega * \vec{v})$

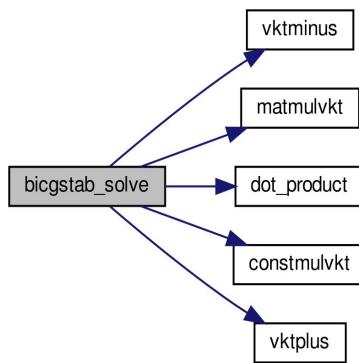
Zeitdauer des Verfahrens

Die Lösung wird in "fort.33" ausgegeben

Lösung am Rand ausgeben

Lösung in den inneren Punkte ausgeben

Gnuplot-Skript "cg_Tetraed дем" aufrufen Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



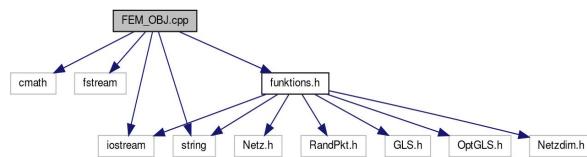
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.2 FEM_OBJ.cpp-Dateireferenz

Hauptprogramm FEM_Tetraeder.

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
#include "funktions.h"
Include-Abhängigkeitsdiagramm für FEM_OBJ.cpp:
```



Funktionen

- int `main ()`
main Funktion

2.2.1 Ausführliche Beschreibung

Hauptprogramm FEM_Tetraeder.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.2.2 Dokumentation der Funktionen

2.2.2.1 main()

int main ()

main Funktion

< netzdim : Netzdimension

< netz : Netz

< RP : Randpunkte

< G : Gleichungssystem

< Opt : optimiertes Gleichungssystem

Name des Inputfiles

Netzdimension bestimmen

Input-File einlesen

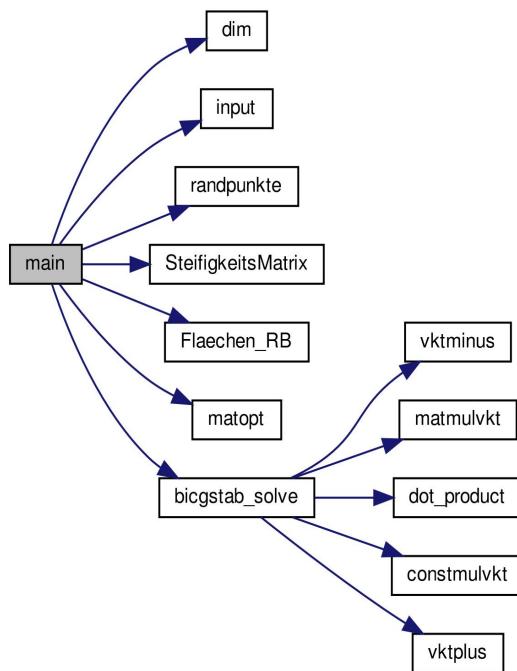
globale Steifigkeitsmatrix und Rechteseite b berechnen

Matrix verkuerzen

optimierte unbekante Vektor x dimensionieren

Erzeugt von Doxygen

BiCG Verfahren aufrufen Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



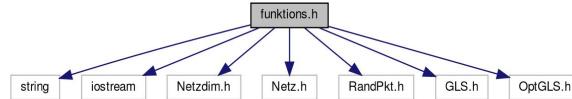
2.3 funktions.h-Dateireferenz

Funktionen declaration.

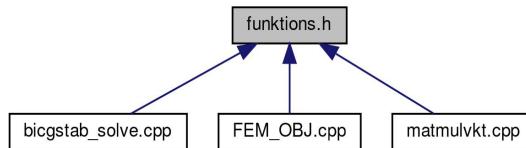
```
#include <string>
#include <iostream>
#include "Netzdim.h"
#include "Netz.h"
#include "RandPkt.h"
#include "GLS.h"
#include "OptGLS.h"
```

Erzeugt von Doxygen

Include-Abhängigkeitsdiagramm für funktions.h:



Dieser Graph zeigt, welche Datei direkt oder indirekt diese Datei enthält:



Funktionen

- Netzdim **dim** (string &str)
- Netz **input** (string &str, Netzdim netzdim)
- RandPkt **randpunkte** (string &str, Netzdim ndim, Netz n)
- GLS **SteifigkeitsMatrix** (Netzdim ndim, Netz n)
- GLS **Flaechen_RB** (Netzdim ndim, Netz n, RandPkt RP, GLS g)
- OptGLS **matopt** (Netzdim ndim, Netz n, RandPkt RP, GLS g)
- int **bicgstab_solve** (Netzdim Ndim, Netz N, RandPkt RP, OptGLS optg)

Funktion des BiCG-Verfahrens.
- double * **matmulvkt** (double **a, double *b, int arow, int acol, int brow)

*Funktion Matrix*Vektor
Anzahl des Matrix-Spalts soll gleich die Vektor-Dimension sein.*
- double **dot_product** (double *a, double *b, int arow, int brow)
- double * **vktminus** (double *a, double *b, int arow, int brow)
- double * **vktplus** (double *a, double *b, int arow, int brow)
- double * **constmulvkt** (double a, double *b, int brow)

2.3.1 Ausführliche Beschreibung

Funktionen declaration.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.3.2 Dokumentation der Funktionen

2.3.2.1 bicgstab_solve()

```
int bicgstab_solve (
    Netzdim Ndim,
    Netz N,
    RandPkt RP,
    OptGLS optg )
```

Funktion des BiCG-Verfahrens.

Parameter

<i>ndim</i>	Anzahl der Matrix und des Vektors
<i>**a</i>	2-Dimension Matrix(linke Seite)
<i>*b</i>	Vektor(rechte Seite)
<i>*x</i>	unbekannter Vektor
<i>**PTetr</i>	Punktnummern im Tetraeder

Rückgabe

Keine Rückgabe, aber ein Gnuplot-Skript wird aufgerufen um den Fehler zu sehen

< alpha: Reelle Zahl α

< beta: Reelle Zahl β

< delta: Fehler Δ

< omega: Reelle Zahl ω

< rho: Reelle Zahl ρ

< rho_alt: Reelle Zahl ρ_{alt}

< eps: Genauigkeit ϵ

< u: Standard Lösung

< xnetz: X-Koordinaten

Erzeugt von Doxygen

```

< ynetz: Y-Koordinaten
< znetz: Z-Koordinaten
< ndim : optimierte Dimension
< **a : Linke Seite Matrix
< *b : Rechte Seite Vektor
< *x : Unbekannter Vektor
< **PTetr : optimierte Punkte
< *p: Vektor  $\vec{p}$ 
< *r: Vektor  $\vec{r}$ 
< *r0: Vektor  $\vec{r}_0$ 
< *v: Vektor  $\vec{v}$ 
< *s: Vektor  $\vec{s}$ 
< *t: Vektor  $\vec{t}$ 
< *tmp1: Zwischen Vektor  $\overrightarrow{\text{tmp1}}$ 
< *tmp2: Zwischen Vektor  $\overrightarrow{\text{tmp2}}$ 
< it: Zähler für Iterationsschleife
< st0,ste: Zeit Messung des Verfahrens
< cpu_sekunden: Zeitdauer des Verfahrens
Genauigkeit der double Zahl

BiCG-Verfahren anfangen:

Initialisieren  $\vec{x} = 3.0$ 
Setze  $\vec{p} = \vec{b} - A * \vec{x}$ 
Setze  $\vec{b} = \vec{b} - A * \vec{x}$ 
Setze  $\vec{r}_0 = \vec{r}$ 
Setze  $\rho = \vec{r} \cdot \vec{r}$ 

Iterationsschleife anfangen:

proof  $\vec{r} \cdot \vec{r} > \epsilon^2$ 
 $\vec{v} = A * \vec{p}$ 
 $\alpha = \rho / (\vec{v} \cdot \vec{r}_0)$ 
 $\vec{s} = \vec{r} - \alpha * \vec{v}$ 

```

$$\vec{t} = A * \vec{s}$$

$$\omega = \frac{\vec{r} \cdot \vec{s}}{\vec{t} \cdot \vec{t}}$$

$$\vec{x}_{k+1} = \vec{x}_k + \alpha * \vec{p} + \omega * \vec{s}$$

$$\vec{r} = \vec{s} - \omega * \vec{t}$$

ρ_k speichern

$$\rho_{k+1} = \vec{r} \cdot \vec{r}_0$$

$$\beta = \alpha / \omega * \rho_{k+1} / \rho_k$$

$$\vec{p} = \vec{r} + \beta * (\vec{p} - \omega * \vec{v})$$

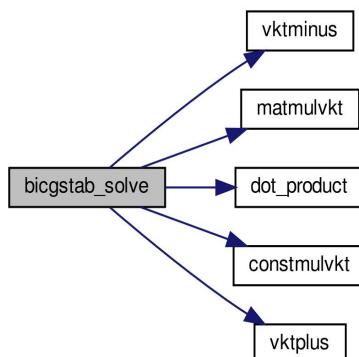
Zeitdauer des Verfahrens

Die Lösung wird in "fort.33" ausgegeben

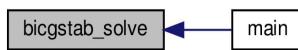
Lösung am Rand ausgeben

Lösung in den inneren Punkte ausgeben

Gnuplot-Skript "cg_Tetraed дем" aufrufen Hier ist ein Graph, der zeigt, was diese Funktion aufruft:



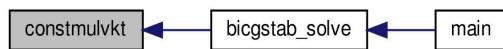
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.2 constmulvkt()

```
double* constmulvkt (
    double a,
    double * b,
    int brow )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.3 dim()

```
Netzdim dim (
    string & str )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.4 dot_product()

```
double dot_product (
    double * a,
    double * b,
    int arow,
    int brow )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.5 Flaechen_RB()

```
GLS Flaechen_RB (
    Netzdim ndim,
    Netz n,
    RandPkt RP,
    GLS g )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.6 input()

```
Netz input (
    string & str,
    Netzdim netzdim )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.7 matmulvkt()

```
double* matmulvkt (
    double ** a,
    double * b,
    int arow,
    int acol,
    int brow )
```

Funktion Matrix*Vektor

Anzahl des Matrix-Spalts soll gleich die Vektor-Dimension sein.

Parameter

<i>**a</i>	Matrix a
<i>*b</i>	Vektor b
<i>arow</i>	Anzahl der Zeile a
<i>acol</i>	Anzahl des Spalts a
<i>brow</i>	Vektor-Dimension

Rückgabe

geben einen neuen Vektor zurück

< **out*: der Lösung-Vektor

$$\overrightarrow{out} = A * \vec{b}$$

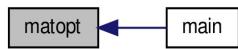
Fehlermeldung Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.8 matopt()

```
OptGLS matopt (
    Netzdim ndim,
    Netz n,
    RandPkt RP,
    GLS g )
```

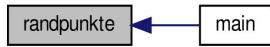
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.9 randpunkte()

```
RandPkt randpunkte (
    string & str,
    Netzdim ndim,
    Netz n )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.10 SteifigkeitsMatrix()

```
GLS SteifigkeitsMatrix (
    Netzdim ndim,
    Netz n )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



2.3.2.11 vktminus()

```
double* vktminus (
    double * a,
    double * b,
    int arow,
    int brow )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

**2.3.2.12 vktplus()**

```
double* vktplus (
    double * a,
    double * b,
    int arow,
    int brow )
```

Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

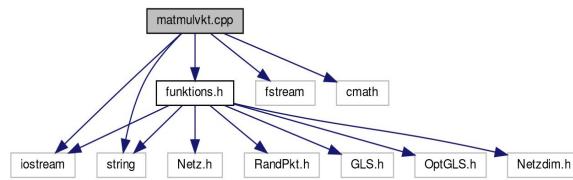


2.4 matmulvkt.cpp-Dateireferenz

Ein Unterprogramm für Matrix*Vektor.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
```

```
#include "funktions.h"
Include-Abhängigkeitsdiagramm für matmulvkt.cpp:
```



Funktionen

- double * matmulvkt (double **a, double *b, int arow, int acol, int brow)

*Funktion Matrix*Vektor
Anzahl des Matrix-Spalts soll gleich die Vektor-Dimension sein.*

2.4.1 Ausführliche Beschreibung

Ein Unterprogramm für Matrix*Vektor.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.4.2 Dokumentation der Funktionen

2.4.2.1 matmulvkt()

```
double* matmulvkt (
    double ** a,
    double * b,
    int arow,
    int acol,
    int brow )
```

*Funktion Matrix*Vektor
Anzahl des Matrix-Spalts soll gleich die Vektor-Dimension sein.*

Erzeugt von Doxygen

Parameter

<i>**a</i>	Matrix a
<i>*b</i>	Vektor b
<i>arow</i>	Anzahl der Zeile a
<i>acol</i>	Anzahl des Spalts a
<i>brow</i>	Vektor-Dimension

Rückgabe

geben einen neuen Vektor zurück

< **out*: der Lösung-Vektor

$$\overrightarrow{out} = A * \vec{b}$$

Fehlermeldung Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:



Index

```
bicgstab_solve
    bicgstab_solve.cpp, 4
    funktions.h, 11
bicgstab_solve.cpp, 3
    bicgstab_solve, 4

constmulvkt
    funktions.h, 13

dim
    funktions.h, 14
dot_product
    funktions.h, 14

FEM_OBJ.cpp, 7
    main, 8
Flaechen_RB
    funktions.h, 15
funktions.h, 9
    bicgstab_solve, 11
    constmulvkt, 13
    dim, 14
    dot_product, 14
    Flaechen_RB, 15
    input, 15
    matmulvkt, 15
    matopt, 16
    randpunkte, 17
    SteifigkeitsMatrix, 17
    vktminus, 17
    vktplus, 18

input
    funktions.h, 15

main
    FEM_OBJ.cpp, 8
matmulvkt
    funktions.h, 15
    matmulvkt.cpp, 19
matmulvkt.cpp, 18
    matmulvkt, 19
matopt
    funktions.h, 16

randpunkte
    funktions.h, 17

SteifigkeitsMatrix
    funktions.h, 17
```

References

- [1] C. Geuzane, J.F. Remacle : *Gmsh: a three-dimensional finite-element mesh generator with built-in pre- and post-processing facilities*, International Journal for numerical Methods in Engineering, Volume 79, Issue 11, pages 1309–1331, 2009
- [2] LAPACK is a software package provided by Univ. of Tennessee; Univ. of California, Berkeley; Univ. of Colorado Denver; and NAG Ltd..
- [3] Andreas Maister : *Numerik linearer Gleichungssysteme*, Springer–Spektrum, 2015, e–Book
- [4] <http://www.netlib.org/>
- [5] Wikipedia: Gnuplot; <https://de.wikipedia.org/wiki/Gnuplot>, Internetpräsenz, (Besuch am ???.201???.2018)
- [6] Gnuplot: gnuplot homepage; <http://www.gnuplot.info/>, Internetpräsenz, (Besuch am ???.2018)
- [7] Intel: Intel(R) Parallel Studio XE 2017; <https://software.intel.com/en-us/intel-parallel-studio-xe>, Internetpräsenz, (Besuch am 22.3.2017) [Für ifort und Math-Kernel-Library.]
- [8] GNU: Welcome to the home of GNU Fortran; <https://gcc.gnu.org/fortran/>, Internetpräsenz, (Besuch am ???.2018)
- [9] <http://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf>
- [10] H.A.van der Vorst : BI-CGSTAB : A fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems, SIAM.J.Sci.Stat. Comput., 13: 631–644, 1992
- [11] Wikipedia: Wall-clock time; https://en.wikipedia.org/wiki/Wall-clock_time, Internetpräsenz, (Besuch am ???.2018)
- [12] Wikipedia: Speedup; <https://de.wikipedia.org/wiki/Speedup>, Internetpräsenz, (Besuch am ???.2018)
- [13] Wikipedia: CPU time; https://en.wikipedia.org/wiki/CPU_time, Internetpräsenz, (Besuch am ???.2018)
- [14] Eclipse Git Tutorial; <https://http://www.vogella.com/tutorials/EclipseGit/article.html>, Lars Vogel (c) 2009, 2017 vogella GmbH version 4.3, 28.11.2017
- [15] Quarteroni Sacco Saleri:*Numerische Mathematik 1*, Springer–Spektrum, 2002