FEM Tetraeder

Erzeugt von Doxygen 1.8.6

Fre Aug 17 2018 16:02:12

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 3

2.1.2 Dokumentation der Funktionen 4

2.1.2.1 bicgstab_solve 4

2.2 constmulvkt.cpp-Dateireferenz 6

2.2.1 Ausführliche Beschreibung 7

2.2.2 Dokumentation der Funktionen 7

2.2.2.1 constmulvkt 7

2.3 dim_tetr.cpp-Dateireferenz 8

2.3.1 Ausführliche Beschreibung 9

2.3.2 Dokumentation der Funktionen 9

2.3.2.1 dim_tetr 9

2.4 dim_tetr_pkt.cpp-Dateireferenz 10

2.4.1 Ausführliche Beschreibung 10

2.4.2 Dokumentation der Funktionen 11

2.4.2.1 dim_tetr_pkt 11

2.5 dot_product.cpp-Dateireferenz 11

2.5.1 Ausführliche Beschreibung 12

2.5.2 Dokumentation der Funktionen 12

2.5.2.1 dot_product 12

2.6 FEM.cpp-Dateireferenz 13

2.6.1 Ausführliche Beschreibung 14

2.6.2 Dokumentation der Funktionen 14

2.6.2.1 main 14

2.7 Flaechen_RB.cpp-Dateireferenz 15

2.7.1 Ausführliche Beschreibung 16

2.7.2 Dokumentation der Funktionen 17

2.7.2.1 Flaechen_RB . . . 17
2.8 funktions.h-Dateireferenz . . . 18
2.8.1 Ausführliche Beschreibung . . . 19
2.8.2 Dokumentation der Funktionen . . . 19
2.8.2.1 bicgstab_solve . . . 19
2.8.2.2 constmulvkt . . . 22
2.8.2.3 dim_tetr . . . 22
2.8.2.4 dim_tetr_pkt . . . 23
2.8.2.5 dot_product . . . 23
2.8.2.6 Flaechen_RB . . . 24
2.8.2.7 matmulvkt . . . 25
2.8.2.8 randpunkte_InnerePkt . . . 26
2.8.2.9 randpunkte_PNummerF . . . 27
2.8.2.10 RBdimension . . . 28
2.8.2.11 SteifigkeitsMatrix . . . 29
2.8.2.12 tetr_input . . . 31
2.8.2.13 tetr_pkt_input . . . 31
2.8.2.14 vktminus . . . 32
2.8.2.15 vktplus . . . 32
2.9 matmulvkt.cpp-Dateireferenz . . . 33
2.9.1 Ausführliche Beschreibung . . . 34
2.9.2 Dokumentation der Funktionen . . . 34
2.9.2.1 matmulvkt . . . 34
2.10 randpunkte_InnerePkt.cpp-Dateireferenz . . . 35
2.10.1 Ausführliche Beschreibung . . . 36
2.10.2 Dokumentation der Funktionen . . . 36
2.10.2.1 randpunkte_InnerePkt . . . 36
2.11 randpunkte_PNummerF.cpp-Dateireferenz . . . 37
2.11.1 Ausführliche Beschreibung . . . 38
2.11.2 Dokumentation der Funktionen . . . 38
2.11.2.1 randpunkte_PNummerF . . . 38
2.12 RBdimension.cpp-Dateireferenz . . . 39
2.12.1 Ausführliche Beschreibung . . . 40
2.12.2 Dokumentation der Funktionen . . . 40
2.12.2.1 RBdimension . . . 40
2.13 SteifigkeitsMatrix.cpp-Dateireferenz . . . 41
2.13.1 Ausführliche Beschreibung . . . 41
2.13.2 Dokumentation der Funktionen . . . 42
2.13.2.1 SteifigkeitsMatrix . . . 42
2.14 tetr_input.cpp-Dateireferenz . . . 43

2.14.1 Ausführliche Beschreibung . . . 44
2.14.2 Dokumentation der Funktionen . . . 44
2.14.2.1 tetr_input . . . 44
2.15 tetr_pkt_input.cpp-Dateireferenz . . . 45
2.15.1 Ausführliche Beschreibung . . . 45
2.15.2 Dokumentation der Funktionen . . . 45
2.15.2.1 tetr_pkt_input . . . 45
2.16 vktminus.cpp-Dateireferenz . . . 46
2.16.1 Ausführliche Beschreibung . . . 47
2.16.2 Dokumentation der Funktionen . . . 47
2.16.2.1 vktminus . . . 47
2.17 vktplus.cpp-Dateireferenz . . . 48
2.17.1 Ausführliche Beschreibung . . . 48
2.17.2 Dokumentation der Funktionen . . . 48
2.17.2.1 vktplus . . . 48

Index **51**

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

constmulvkt.cpp
: Ein Unterprogramm für konstant∗Vektor . . . 6

dim_tetr.cpp
: Anzahl des Tetraeder bestimmen . . . 8

dim_tetr_pkt.cpp
: Anzahl der Punkten bestimmen . . . 10

dot_product.cpp
: Skalarprodukt . . . 11

FEM.cpp
: Hauptprogramm FEM_Tetraeder . . . 13

Flaechen_RB.cpp
: Flächen Randbedingung . . . 15

funktions.h
: Funktionen declaration . . . 18

matmulvkt.cpp
: Ein Unterprogramm für Matrix∗Vektor . . . 33

randpunkte_InnerePkt.cpp
: Innere Punktenummer bestimmen . . . 35

randpunkte_PNummerF.cpp
: Randpunkte bestimmen . . . 37

RBdimension.cpp
: Anzahl der Randpunkte bestimmen . . . 39

SteifigkeitsMatrix.cpp
: Steifigkeitsmatrix berechnen . . . 41

tetr_input.cpp
: Tetraeder Element einlesen . . . 43

tetr_pkt_input.cpp
: Koordinatenssystem aller Punkten einlesen . . . 45

vktminus.cpp
: Ein Unterprogramm für $\vec{a}-\vec{b}$ . . . 46

vktplus.cpp
: Ein Unterprogramm für $\vec{a}+\vec{b}$ . . . 48

Kapitel 2

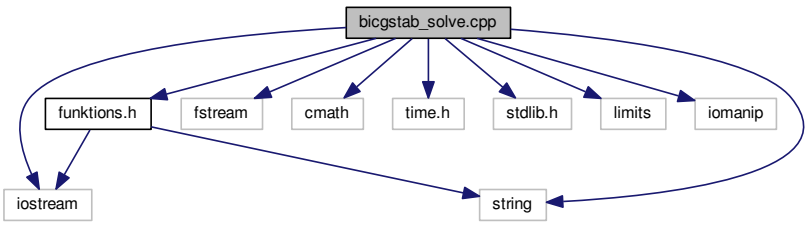
Datei-Dokumentation

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-Abhängigkeitsdiagramm für bicgstab_solve.cpp:



Funktionen

- void bicgstab_solve (int ndim, double ∗∗a, double ∗b, double ∗x, double ∗∗PTetr)

  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 void bicgstab_solve (int *ndim,* double ∗∗ *a,* double ∗ *b,* double ∗ *x,* double ∗∗ *PTetr*)

Funktion des BiCG-Verfahrens.

Parameter

ndim	Anzahl der Matrix und des Vektors
∗∗*a*	2-Dimension Matrix(linke Seite)
∗*b*	Vektor(rechte Seite)
∗*x*	unbekanter Vektor
∗∗*PTetr*	Punktnummern im Tetraeder

Rückgabe

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

double variables declaration:

< alpha: Reelle Zahl α

< beta: Reele Zahl β

< delta: Fehler Δ

< omega: Reelle Zahl ω

< rho: Reelle Zahl ρ

< rho_alt: Reelle Zahl ρ_{alt}

< eps: Genauigkei ε

< u: Standard Lösung

< xnetz: X-Koordinaten

< ynetz: Y-Koordinaten

< znetz: Z-Koordinaten

< ∗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 $\overrightarrow{tmp1}$

$<$ ∗tmp2: Zwischen Vektor $\overrightarrow{tmp2}$

$<$ it: Zähler für Iterationsschleife

$<$ st0,ste: Zeit Messung des Verfahrens

$<$ cpu_sekunden: Zeitdauer des Verfahrens

Genauigkeit der double Zahl

BiCG-Verfahren anfangen:

Initialisiren $\vec{x} = 3.0$

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 $\vec{r} \cdot \vec{r} > \varepsilon$?

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

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

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

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

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

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

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

$\rho_k \quad speichern$

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

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

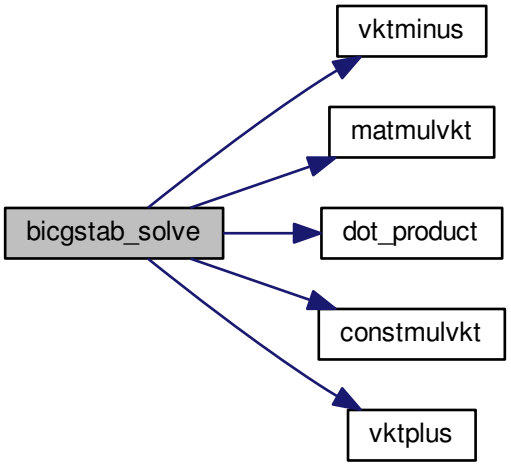
$\vec{p} = \vec{r} + \beta * (\vec{p} - \omega * \vec{v})$

Zeitdauer des Verfahrens

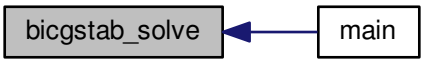
Die Lösung wird in "fort.33" ausgegeben

Gnuplot-Skript "cg_Tetraed.dem" aufrufen

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



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

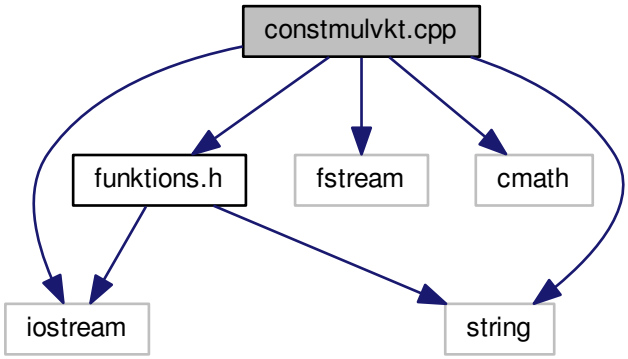


2.2 constmulvkt.cpp-Dateireferenz

Ein Unterprogramm für konstant∗Vektor.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für constmulvkt.cpp:



Funktionen

- double ∗ constmulvkt (double a, double ∗b, int brow)

  Funktion Konstant∗Vektor.

2.2.1 Ausführliche Beschreibung

Ein Unterprogramm für konstant∗Vektor.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.2.2 Dokumentation der Funktionen

2.2.2.1 double∗ constmulvkt (double *a*, double ∗ *b*, int *brow*)

Funktion Konstant∗Vektor.

Parameter

a	ein Konstant

$*b$	Vektor
brow	Dimension des Vektors

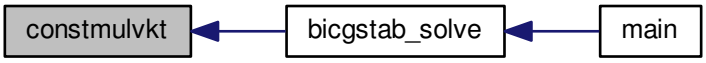
Rückgabe

geben den Lösung Vektor zurück

< *out: der Lösung-Vektor

$out_i = a * b_i$

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

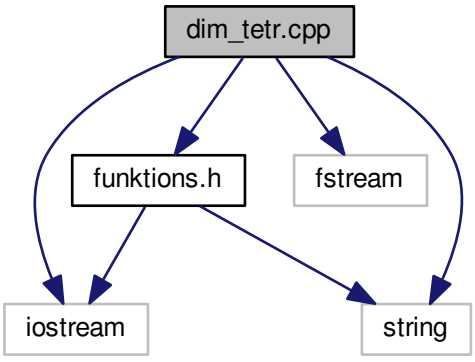


2.3 dim_tetr.cpp-Dateireferenz

Anzahl des Tetraeder bestimmen.

```
#include <iostream>
#include <string>
#include <fstream>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für dim_tetr.cpp:



Funktionen

- int dim_tetr (string &str, int n_tetr_pkt)

  Funktion um die Anzahl des Tetraeders zu bestimmen.

2.3.1 Ausführliche Beschreibung

Anzahl des Tetraeder bestimmen. Ein "∗.msh" File wird als Input eingelesen

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.3.2 Dokumentation der Funktionen

2.3.2.1 int dim_tetr (string & *str,* int *n_tetr_pkt*)

Funktion um die Anzahl des Tetraeders zu bestimmen.

Parameter

&str	Name des Inputfiles
n_tetr_pkt	Anzahl aller Punkten

Rückgabe

geben die Anzahl des Tetraeders zurück

< dim: Zähler des Tetraeders

< a: 1.Spalt des Files

< b: 2.Spalt des Files

Das File besteht aus 2 Teilen:

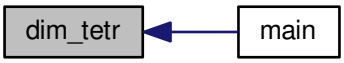
1: Alle Punkte mit XYZ-Koordinaten

2: Alle Element mit Punktnummer

lesen das File bis zu Anfang des Elements ein

1. Spalt ist die Nummer des Elements
2. Spalt ist der Typ des Elements
   Typ Nummer:
   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)

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

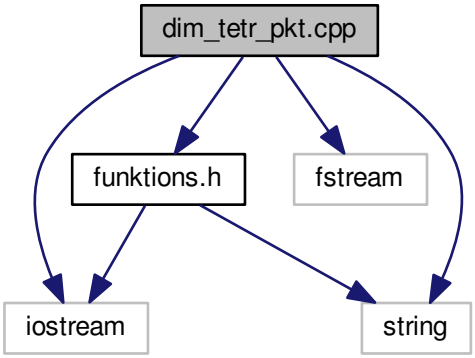


2.4 dim_tetr_pkt.cpp-Dateireferenz

Anzahl der Punkten bestimmen.

```
#include <iostream>
#include <string>
#include <fstream>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für dim_tetr_pkt.cpp:



Funktionen

- int dim_tetr_pkt (string &str)

  Funktion um die Anzahl der Punkten zu bestimmen.

2.4.1 Ausführliche Beschreibung

Anzahl der Punkten bestimmen. Ein "∗.msh" File wird als Input eingelesen

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.4.2 Dokumentation der Funktionen

2.4.2.1 int dim_tetr_pkt (string & *str*)

Funktion um die Anzahl der Punkten zu bestimmen.

Parameter

&str	Name des Inputfiles

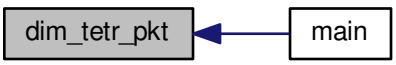
Rückgabe

geben die Anzahl der Punkten

< dim: Zähler der Punkten

Die Anzahl steht in die 5.Zeile des Files, kann direkt eingelesen werden

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

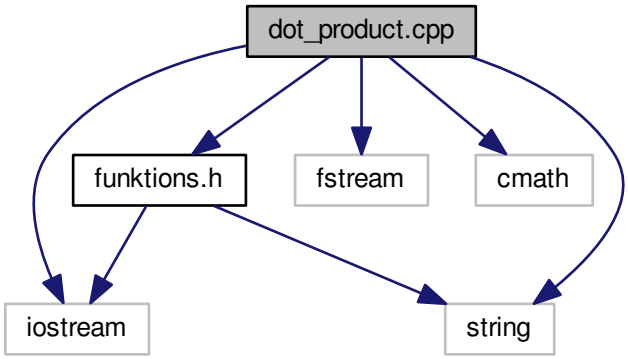


2.5 dot_product.cpp-Dateireferenz

Skalarprodukt.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für dot_product.cpp:



Funktionen

- double dot_product (double ∗a, double ∗b, int arow, int brow)
  Funktion Skalarprodukt.

2.5.1 Ausführliche Beschreibung

Skalarprodukt.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.5.2 Dokumentation der Funktionen

2.5.2.1 double dot_product (double ∗ *a,* double ∗ *b,* int *arow,* int *brow*)

Funktion Skalarprodukt.

Parameter

∗*a*	Vektor a

∗b	Vektor b
arow	Dimension des Vektors a
brow	Dimension des Vektors b

Rückgabe

geben die Lösung $\vec{a} \cdot \vec{b}$ zurück

< out: Die Lösung

$\sum_{i=1}^{n} a_i * b_i$

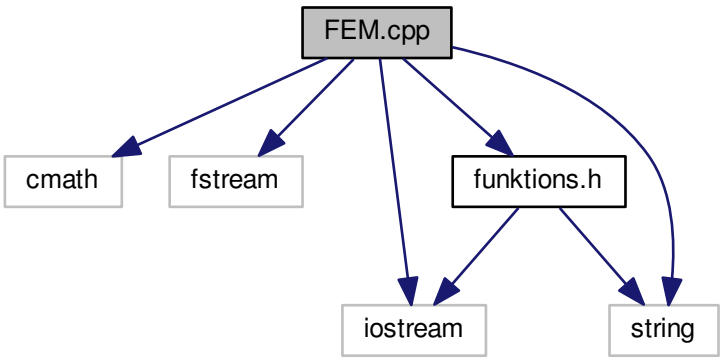
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

2.6 FEM.cpp-Dateireferenz

Hauptprogramm FEM_Tetraeder.

```
#include <cmath>
#include <fstream>
#include <iostream>
#include <string>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für FEM.cpp:



Funktionen

- int main ()

main Funktion

2.6.1 Ausführliche Beschreibung

Hauptprogramm FEM_Tetraeder.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.6.2 Dokumentation der Funktionen

2.6.2.1 int main ( )

main Funktion

< n_tetr_pkt: Anzahl der Punkten

< n_tetr : Anzahl des Tetraeders

< **P: Koordinatenssystem jeder Punkts(n_tetr_pkt Zeilen, 3 Spalte)

< **T: Tetraeder Element(besteht aus 4 Knoten ==>n_tetr Zeilen, 4 Spalte)

< NRanddim: Randpunktdimensionen

< **PNummerF: Randpunkt(NRanddim Zeilen, 6 Spalte)

< **InnerePkt: Innere Punkte(n_tetr_pkt Zeilen)

< **global: globale Steifigkeitsmatrix und b-Matrix zusammen:

1. 1 bis n_tetr_pkt Spalte ===> Steifigkeitsmatrix
2. n_tetr_pkt Spalt ===> b-Matrix(rechte Seite)

Name des Inputfiles

Anzahl der Punkten bestimmen

Anzahl des Tetraeders bestimmen

< *bglobal : b-Matrix dimensionieren(n_tetr_pkt Zeilen)

< **Sglobal: Steifigkeitsmatrix dimensionieren (n_tetr_pkt Zeilen, n_tetr_pkt Spalte)

< *x: Temperaturverteilung dimensionieren(n_tetr_pkt Zeilen)

Punkte einlesen

Tetraeder einlesen

Randpunktedimension bestimmen

Randpunkte bestimmen

Innerepunkt bestimmen

globale Steifigkeitsmatrix und b-Matrix berechnen

globale SteifigkeitsMatrix wird in " SglobalSteifigkeitMain.txt" ausgegeben

globale b-Matrix wird in " bglobalSteifigkeitMain.txt" ausgegeben

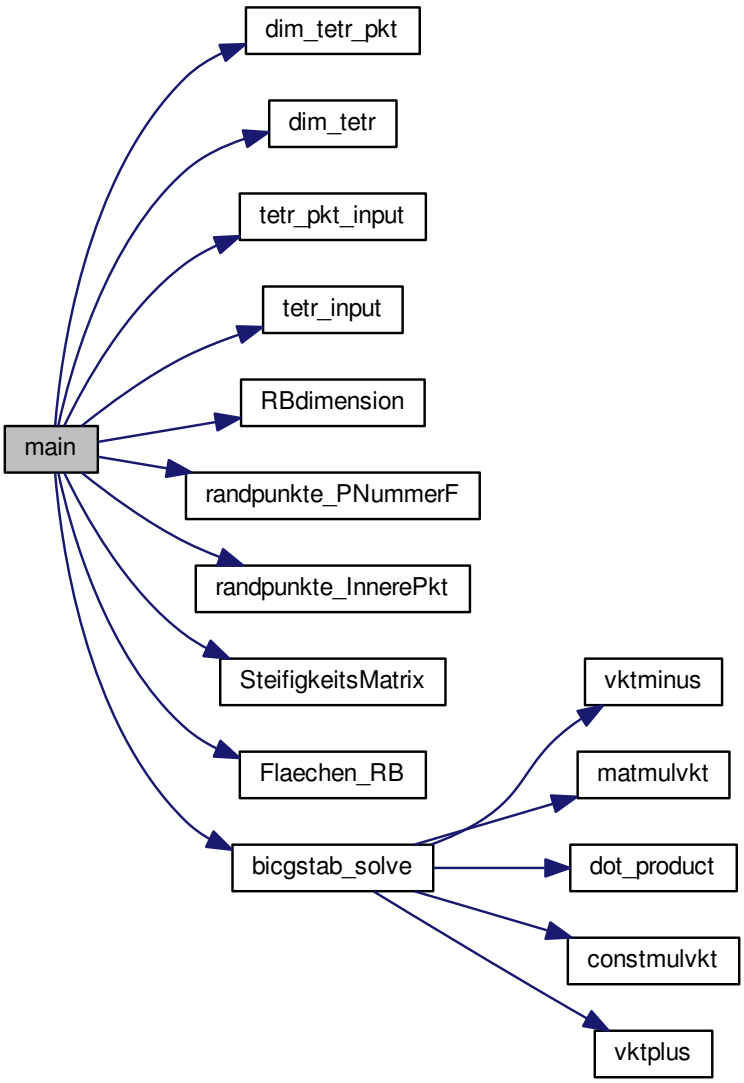
globale SteifigkeitsMatrix mit RB wird in " SglobalFlaechenMain.txt" ausgegeben

globale b-Matrix mit RB wird in " bglobalFlaechenMain.txt" ausgegeben

Temperaturverteilung initialisieren

Das BiCG-Verfahren aufrufen

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

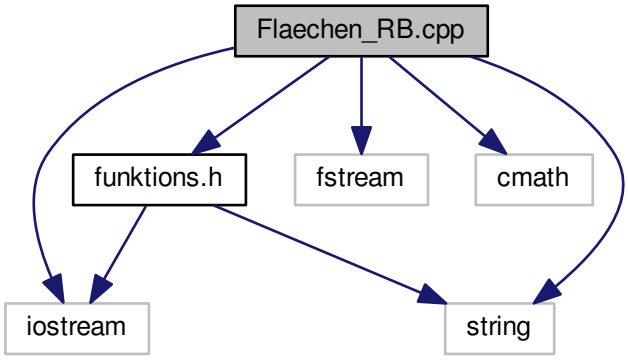


2.7 Flaechen_RB.cpp-Dateireferenz

Flächen Randbedingung.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für Flaechen_RB.cpp:



Funktionen

- double ∗∗ Flaechen_RB (int NRanddim, int n_tetr_pkt, int ∗∗PNummerF, double ∗∗P, double ∗bglobal, int ∗InnerePkt, double ∗∗Sglobal)

  Funktion um Flächen Randbedingung zu berechnen
  Multiplikation der Steifigkeitsmatrix mit T_k in der ki_ten Spalte
  Randbedingung an der k. Randflaeche
  (k=1 : linke Tetraederflaeche)
  (k=2 : untere Tetraederflaeche)
  (k=3 : vordere Tetraederflaeche)
  (k=4 : rechte Tetraederflaeche)

2.7.1 Ausführliche Beschreibung

Flächen Randbedingung.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.7.2 Dokumentation der Funktionen

2.7.2.1 double** Flaechen_RB (int *NRanddim,* int *n_tetr_pkt,* int ** *PNummerF,* double ** *P,* double * *bglobal,* int * *InnerePkt,* double ** *Sglobal*)

Funktion um Flächen Randbedingung zu berechnen

Multiplikation der Steifigkeitsmatrix mit T_k in der ki_ten Spalte

Randbedingung an der k. Randflaeche

(k=1 : linke Tetraederflaeche)

(k=2 : untere Tetraederflaeche)

(k=3 : vordere Tetraederflaeche)

(k=4 : rechte Tetraederflaeche)

Parameter

NRanddim	Anzahl der Randpunkten
n_tetr_pkt	Anzahl aller Punkten
***PNummerF*	Randpunkte
***P*	Alle Punkte
**bglobal*	globale b-Matrix
**InnerePkt*	Innere Punkte
***Sglobal*	globale Steifigkeitsmatrix

Rückgabe

geben die neue globale Matrix(Sglobal und bglobal) zurück

$<$ **global: globale Matrix dimensionieren

$<$ urand: kubische Lösung

$<$ x, y, z: XYZ-Kooridinaten

$<$ f1zaehler: Punkte auf Randflächen F_1

$<$ f2zaehler: Punkte auf Randflächen F_2

$<$ f3zaehler: Punkte auf Randflächen F_3

$<$ f4zaehler: Punkte auf Randflächen F_4

$<$ f5zaehler: Punkte auf Randflächen F_5

$<$ f6zaehler: Punkte auf Randflächen F_6

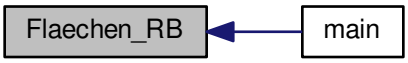
kubische Lösung:

$urand = 20 - 2 * y^2 + x^3 * y - x * y^3 + z^3 * x - z * x^3$

Hier werden alle Komponenten belegt!

Hier wird nur die ki-te Komponente belegt!

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

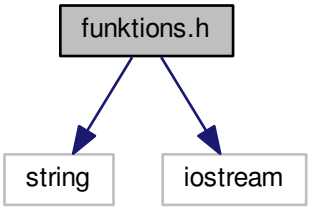


2.8 funktions.h-Dateireferenz

Funktionen declaration.

```
#include <string>
#include <iostream>
```

Include-Abhängigkeitsdiagramm für funktions.h:



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

Funktionen

- int dim_tetr_pkt (string &str)

  Funktion um die Anzahl der Punkten zu bestimmen.
- int dim_tetr (string &str, int n_tetr_pkt)

  Funktion um die Anzahl des Tetraeders zu bestimmen.
- double ∗∗ tetr_pkt_input (string &str, int n_tetr_pkt)

  Funktion um alle Punkte einzulesen.
- int ∗∗ tetr_input (string &str, int n_tetr_pkt, int n_tetr)

  Funktion um die Tetraeder Element einzulesen.
- int RBdimension (int n_tetr_pkt, double ∗∗p)

  Funktion um Anzahl der Randpunkte zu bestimmen.

- int ∗∗ randpunkte_PNummerF (int n_tetr_pkt, int NRanddim, double ∗∗p)

  Funktion um alle Randpunkte zu bestimmen
  diese Funktion ist gleich wie randpunkte_InnerePkt()
  der Unterschied dazwischen ist nur die Rückgabe.
- int ∗ randpunkte_InnerePkt (int n_tetr_pkt, int NRanddim, double ∗∗p)

  Funktion um innere Punktenummer zu bestimmen.
- double ∗∗ SteifigkeitsMatrix (int n_tetr_pkt, int n_tetr, int ∗∗T, double ∗∗P)

  Funktion um die Steifigkeismatrix zu berechnen.
- double ∗∗ Flaechen_RB (int NRanddim, int n_tetr_pkt, int ∗∗PNummerF, double ∗∗P, double ∗bglobal, int ∗InnerePkt, double ∗∗Sglobal)

  Funktion um Flächen Randbedingung zu berechnen
  Multiplikation der Steifigkeitsmatrix mit T_k in der ki_ten Spalte
  Randbedingung an der k. Randflaeche
  (k=1 : linke Tetraederflaeche)
  (k=2 : untere Tetraederflaeche)
  (k=3 : vordere Tetraederflaeche)
  (k=4 : rechte Tetraederflaeche)
- void bicgstab_solve (int ndim, double ∗∗a, double ∗b, double ∗x, double ∗∗PTetr)

  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)

  Funktion Skalarprodukt.
- double ∗ vktminus (double ∗a, double ∗b, int arow, int brow)

  Funktion $\vec{a}-\vec{b}$.
- double ∗ vktplus (double ∗a, double ∗b, int arow, int brow)

  Funktion $\vec{a}+\vec{b}$.
- double ∗ constmulvkt (double a, double ∗b, int brow)

  Funktion Konstant∗Vektor.

2.8.1 Ausführliche Beschreibung

Funktionen declaration.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.8.2 Dokumentation der Funktionen

2.8.2.1 void bicgstab_solve (int *ndim*, double ∗∗ *a*, double ∗ *b*, double ∗ *x*, double ∗∗ *PTetr*)

Funktion des BiCG-Verfahrens.

Parameter

ndim	Anzahl der Matrix und des Vektors
$**a$	2-Dimension Matrix(linke Seite)
$*b$	Vektor(rechte Seite)
$*x$	unbekanter Vektor
$**PTetr$	Punktnummern im Tetraeder

Rückgabe

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

double variables declaration:

$<$ alpha: Reelle Zahl α

$<$ beta: Reele Zahl β

$<$ delta: Fehler Δ

$<$ omega: Reelle Zahl ω

$<$ rho: Reelle Zahl ρ

$<$ rho_alt: Reelle Zahl ρ_{alt}

$<$ eps: Genauigkei ε

$<$ u: Standard Lösung

$<$ xnetz: X-Koordinaten

$<$ ynetz: Y-Koordinaten

$<$ znetz: Z-Koordinaten

$<$ $*$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 $\overrightarrow{tmp1}$

$<$ $*$tmp2: Zwischen Vektor $\overrightarrow{tmp2}$

$<$ it: Zähler für Iterationsschleife

$<$ st0,ste: Zeit Messung des Verfahrens

$<$ cpu_sekunden: Zeitdauer des Verfahrens

Genauigkeit der double Zahl

BiCG-Verfahren anfangen:

Initialisiren $\vec{x} = 3.0$

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 $\vec{r} \cdot \vec{r} > \varepsilon$?

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

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

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

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

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

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

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

$\rho_k \quad speichern$

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

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

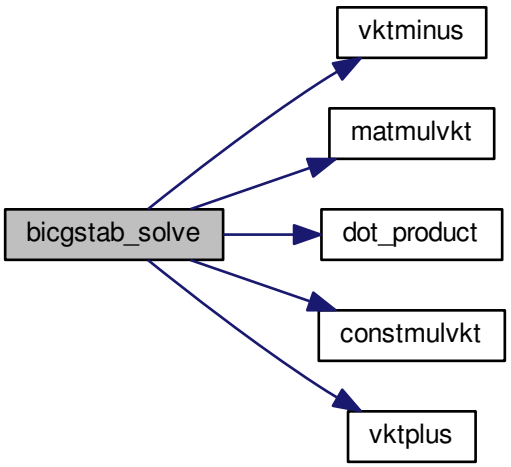
$\vec{p} = \vec{r} + \beta * (\vec{p} - \omega * \vec{v})$

Zeitdauer des Verfahrens

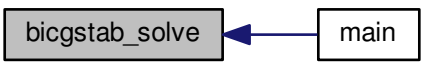
Die Lösung wird in "fort.33" ausgegeben

Gnuplot-Skript "cg_Tetraed.dem" aufrufen

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



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



2.8.2.2 double∗ constmulvkt (double *a,* double ∗ *b,* int *brow*)

Funktion Konstant∗Vektor.

Parameter

a	ein Konstant
∗*b*	Vektor
brow	Dimension des Vektors

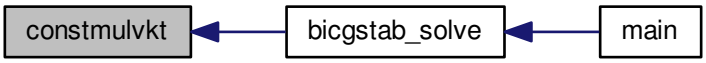
Rückgabe

geben den Lösung Vektor zurück

< ∗out: der Lösung-Vektor

$out_i = a * b_i$

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



2.8.2.3 int dim_tetr (string & *str,* int *n_tetr_pkt*)

Funktion um die Anzahl des Tetraeders zu bestimmen.

Parameter

&str	Name des Inputfiles
n_tetr_pkt	Anzahl aller Punkten

Rückgabe

geben die Anzahl des Tetraeders zurück

< dim: Zähler des Tetraeders

< a: 1.Spalt des Files

< b: 2.Spalt des Files

Das File besteht aus 2 Teilen:

1: Alle Punkte mit XYZ-Koordinaten

2: Alle Element mit Punktnummer

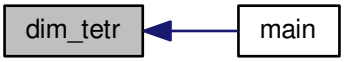
lesen das File bis zu Anfang des Elements ein

1. Spalt ist die Nummer des Elements
2. Spalt ist der Typ des Elements

   Typ Nummer:

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)

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



2.8.2.4 int dim_tetr_pkt (string & *str*)

Funktion um die Anzahl der Punkten zu bestimmen.

Parameter

&str	Name des Inputfiles

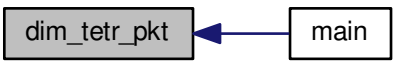
Rückgabe

geben die Anzahl der Punkten

< dim: Zähler der Punkten

Die Anzahl steht in die 5.Zeile des Files, kann direkt eingelesen werden

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



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

Funktion Skalarprodukt.

Parameter

$*a$	Vektor a
$*b$	Vektor b
arow	Dimension des Vektors a
brow	Dimension des Vektors b

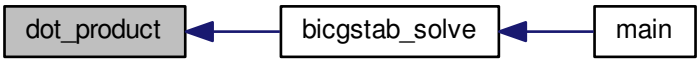
Rückgabe

geben die Lösung $\vec{a} \cdot \vec{b}$ zurück

$<$ out: Die Lösung

$\sum_{i=1}^{n} a_i * b_i$

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



2.8.2.6 double$**$ Flaechen_RB (int *NRanddim,* int *n_tetr_pkt,* int $**$ *PNummerF,* double $**$ *P,* double $*$ *bglobal,* int $*$ *InnerePkt,* double $**$ *Sglobal*)

Funktion um Flächen Randbedingung zu berechnen

Multiplikation der Steifigkeitsmatrix mit T_k in der ki_ten Spalte

Randbedingung an der k. Randflaeche

(k=1 : linke Tetraederflaeche)

(k=2 : untere Tetraederflaeche)

(k=3 : vordere Tetraederflaeche)

(k=4 : rechte Tetraederflaeche)

Parameter

NRanddim	Anzahl der Randpunkten
n_tetr_pkt	Anzahl aller Punkten
$**$*PNummerF*	Randpunkte
$**$*P*	Alle Punkte
$*$*bglobal*	globale b-Matrix
$*$*InnerePkt*	Innere Punkte
$**$*Sglobal*	globale Steifigkeitsmatrix

Rückgabe

geben die neue globale Matrix(Sglobal und bglobal) zurück

$<$ $**$global: globale Matrix dimensionieren

$<$ urand: kubische Lösung

< x, y, z: XYZ-Kooridinaten

< f1zaehler: Punkte auf Randflächen F_1

< f2zaehler: Punkte auf Randflächen F_2

< f3zaehler: Punkte auf Randflächen F_3

< f4zaehler: Punkte auf Randflächen F_4

< f5zaehler: Punkte auf Randflächen F_5

< f6zaehler: Punkte auf Randflächen F_6

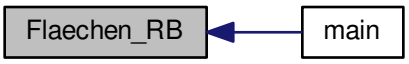
kubische Lösung:

$urand = 20 - 2 * y^2 + x^3 * y - x * y^3 + z^3 * x - z * x^3$

Hier werden alle Komponenten belegt!

Hier wird nur die ki-te Komponente belegt!

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



2.8.2.7 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

**a	Matrix a
*b	Vektor b
arow	Anzahl der Zeile a
acol	Anzahl des Spalts a
brow	Vektor-Dimension

Rückgabe

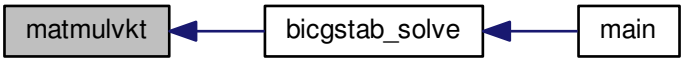
geben einen neuen Vektor zurück

< ∗out: der Lösung-Vektor

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

Fehlermeldung

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



2.8.2.8 int∗ randpunkte_InnerePkt (int *n_tetr_pkt*, int *NRanddim*, double ∗∗ *p*)

Funktion um innere Punktenummer zu bestimmen.

Parameter

n_tetr_pkt	Anzahl aller Punkten
NRanddim	Dimension des Randpunkts
∗∗*p*	Punkte mit Koordinatenssystem

Rückgabe

geben die innere Punktenummer zurück

< ∗∗pnummerf: Nummerfeld der Randpunkt dimensionieren

< ∗innerepkt: Innerepunkte dimensionieren

< f1zaehler: Punkte auf Randflächen F_1

< f2zaehler: Punkte auf Randflächen F_2

< f3zaehler: Punkte auf Randflächen F_3

< f4zaehler: Punkte auf Randflächen F_4

< f5zaehler: Punkte auf Randflächen F_5

< f6zaehler: Punkte auf Randflächen F_6

< eps: Genauigkeit Vergliech zu Null

< xmax: Größte Zahl in X-Richtung

< xmin: kleinste Zahl in X-Richtung

< ymax: Größte Zahl in Y-Richtung

< ymin: kleinste Zahl in Y-Richtung

< zmax: Größte Zahl in Z-Richtung

< zmin: kleinste Zahl in Z-Richtung

Randflächen suchen

Punkte auf Randflächen F_1(ymax)

Punkte auf Randflächen F_2(ymin)

Punkte auf Randflächen F_3(xmax)

Punkte auf Randflächen F_4(xmin)

Punkte auf Randflächen F_5(zmax)

Punkte auf Randflächen F_6(zmin)

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

2.8.2.9 int** randpunkte_PNummerF (int *n_tetr_pkt*, int *NRanddim*, double ** *p*)

Funktion um alle Randpunkte zu bestimmen

diese Funktion ist gleich wie randpunkte_InnerePkt()

der Unterschied dazwischen ist nur die Rückgabe.

Parameter

n_tetr_pkt	Anzahl aller Punkten
NRanddim	Dimension des Randpunkts
***p*	Punkte mit Koordinatenssystem

Rückgabe

geben die Randpunkte(auf 6 Flächen) zurück

Siehe auch

randpunkte_InnerePkt()

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

2.8.2.10 int RBdimension (int *n_tetr_pkt,* double ∗∗ *p*)

Funktion um Anzahl der Randpunkte zu bestimmen.

Parameter

n_tetr_pkt	Anzahl aller Punkte
$**p$	Punkte mit Koordinatenssystem

Rückgabe

geben die Anzahl der Randpunkte zurück

$<$ NRanddim: Anzahl der Randpunkt

$<$ f1zaehler: Punkte auf Randflächen F_1

$<$ f2zaehler: Punkte auf Randflächen F_2

$<$ f3zaehler: Punkte auf Randflächen F_3

$<$ f4zaehler: Punkte auf Randflächen F_4

$<$ f5zaehler: Punkte auf Randflächen F_5

$<$ f6zaehler: Punkte auf Randflächen F_6

$<$ eps: Genauigkeit

$<$ xmax: Größte Zahl in X-Richtung

$<$ xmin: kleinste Zahl in X-Richtung

$<$ ymax: Größte Zahl in Y-Richtung

$<$ ymin: kleinste Zahl in Y-Richtung

$<$ zmax: Größte Zahl in Z-Richtung

$<$ zmin: kleinste Zahl in Z-Richtung

Randflächen suchen

Punkte auf Randflächen F_1(ymax)

Punkte auf Randflächen F_2(ymin)

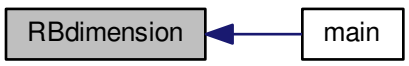
Punkte auf Randflächen F_3(xmax)

Punkte auf Randflächen F_4(xmin)

Punkte auf Randflächen F_5(zmax)

Punkte auf Randflächen F_6(zmin)

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



2.8.2.11 double$**$ SteifigkeitsMatrix (int *n_tetr_pkt,* int *n_tetr,* int $**$ *T,* double $**$ *P*)

Funktion um die Steifigkeismatrix zu berechnen.

Parameter

n_tetr_pkt	Anzahl aller Punkte
n_tetr	Anzahl der Tetraeder
$**T$	Tetraeder Element
$**P$	Punkte mit Koordinatenssystem

Rückgabe

geben die globale Steifigkeitsmatrix und b-Matrix zurück

< ∗bglobal: rechte Seite b dimensionieren

< ∗∗Sglobal: Steifigkeitsmatrix dimensionieren

< ∗∗global: globale Steifigkeitsmatrix und b-Matrix zusammen:

1. 1 bis n_tetr_pkt Spalte ===> Steifigkeitsmatrix
2. n_tetr_pkt Spalt ===> b-Matrix(rechte Seite)

< Ndim: Dimension

< det_Phi: Determinante ϕ

< one_by_Phi: $\frac{1}{det\phi}$

< Slocal[4][4]: locale Steifigkeitsmatrix

< blocal[4] : locale b-Matrix

Slocal[4][4] initialisieren

blocal[4] initialisieren

$x_2 - x_1$

$y_2 - y_1$

$z_2 - z_1$

$x_3 - x_1$

$y_3 - y_1$

$z_3 - z_1$

$x_4 - x_1$

$y_4 - y_1$

$z_4 - z_1$

vermeintlicher Fehler <== ELMER

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

Rechte Seite ∗ q=4 und die 1/6 der Steifigkeitsmatrix auf die andere Seite

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

2.8.2.12 int** tetr_input (string & *str,* int *n_tetr_pkt,* int *n_tetr*)

Funktion um die Tetraeder Element einzulesen.

Parameter

&str	Name des Inputfiles
n_tetr_pkt	Anzahl aller Punkten
n_tetr	Anzahl des Tetraeder

< **t: Tetraederfeld dimensionieren

< idummy1: Element Index

< idummy2: Element Typ

< idummy3: Nummer der tags

< idummy4: <tags>

< idummy5: Knoten Nummer-List

< dim_element: Anzahl aller Elemente

< a,b,c,d: 4 Knoten

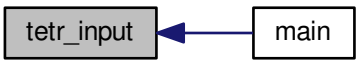
Lesen das File bis zu die Zeile ein,die zu Element gehört

Typnummer des Tetraeders ist gleich 4 ==> idummy2 == 4 ?

4 Knoten einlesen

Fehlermeldung

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



2.8.2.13 double** tetr_pkt_input (string & *str,* int *n_tetr_pkt*)

Funktion um alle Punkte einzulesen.

< **p: Punktefeld dimensionieren

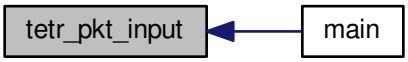
< idummy: Punktenummer

< a,b,c: XYZ-Koordinatenssystem

XYZ-Knoten einlesen

Fehlermeldung

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



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

Funktion $\vec{a} - \vec{b}$.

Dimension $\vec{a}$ soll gleich wie $\vec{b}$ sein

Parameter

*a	Vektor
*b	Vektor
arow	Dimension des Vektors a
brow	Dimension des Vektors b

Rückgabe

geben den Lösung Vektor zurück

Siehe auch

vktplus()

< *out: der Lösung-Vektor

$out_i = a_i * b_i$

Fehlermeldung

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

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

Funktion $\vec{a} + \vec{b}$.

Dimension $\vec{a}$ soll gleich wie $\vec{b}$ sein

Parameter

$*a$	Vektor
$*b$	Vektor
arow	Dimension des Vektors a
brow	Dimension des Vektors b

Rückgabe

geben den Lösung Vektor zurück

Siehe auch

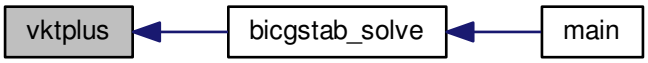
vktminus()

$<$ $*$out: der Lösung-Vektor

$out_i = a_i * b_i$

Fehlermeldung

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

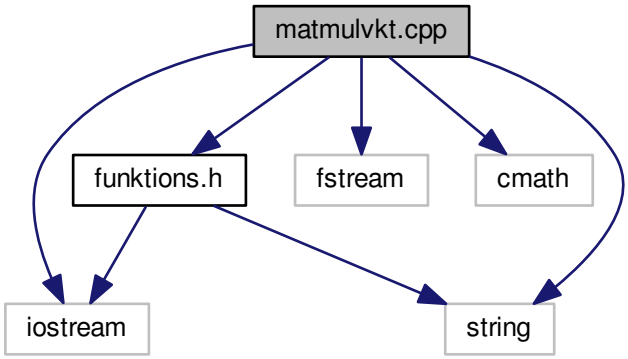


2.9 matmulvkt.cpp-Dateireferenz

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.

2.9.1 Ausführliche Beschreibung

Ein Unterprogramm für Matrix∗Vektor.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.9.2 Dokumentation der Funktionen

2.9.2.1 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

$**a$	Matrix a
$*b$	Vektor b
arow	Anzahl der Zeile a
acol	Anzahl des Spalts a
brow	Vektor-Dimension

Rückgabe

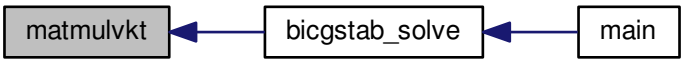
geben einen neuen Vektor zurück

< ∗out: der Lösung-Vektor

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

Fehlermeldung

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

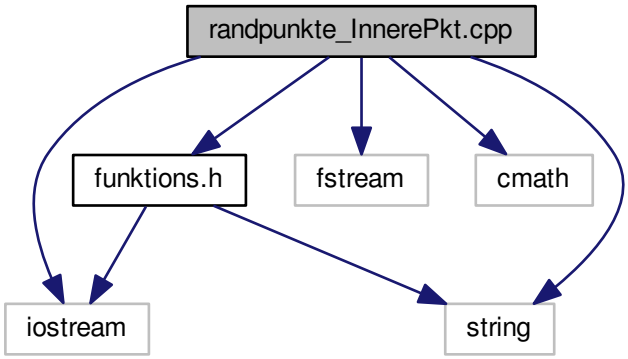


2.10 randpunkte_InnerePkt.cpp-Dateireferenz

Innere Punktenummer bestimmen.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für randpunkte_InnerePkt.cpp:



Funktionen

- int ∗ randpunkte_InnerePkt (int n_tetr_pkt, int NRanddim, double ∗∗p)

  Funktion um innere Punktenummer zu bestimmen.

2.10.1 Ausführliche Beschreibung

Innere Punktenummer bestimmen.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.10.2 Dokumentation der Funktionen

2.10.2.1 int∗ randpunkte_InnerePkt (int *n_tetr_pkt*, int *NRanddim*, double ∗∗ *p*)

Funktion um innere Punktenummer zu bestimmen.

Parameter

n_tetr_pkt	Anzahl aller Punkten

NRanddim	Dimension des Randpunkts
$**p$	Punkte mit Koordinatenssystem

Rückgabe

geben die innere Punktenummer zurück

< $**$pnummerf: Nummerfeld der Randpunkt dimensionieren

< $*$innerepkt: Innerepunkte dimensionieren

< f1zaehler: Punkte auf Randflächen F_1

< f2zaehler: Punkte auf Randflächen F_2

< f3zaehler: Punkte auf Randflächen F_3

< f4zaehler: Punkte auf Randflächen F_4

< f5zaehler: Punkte auf Randflächen F_5

< f6zaehler: Punkte auf Randflächen F_6

< eps: Genauigkeit Vergliech zu Null

< xmax: Größte Zahl in X-Richtung

< xmin: kleinste Zahl in X-Richtung

< ymax: Größte Zahl in Y-Richtung

< ymin: kleinste Zahl in Y-Richtung

< zmax: Größte Zahl in Z-Richtung

< zmin: kleinste Zahl in Z-Richtung

Randflächen suchen

Punkte auf Randflächen F_1(ymax)

Punkte auf Randflächen F_2(ymin)

Punkte auf Randflächen F_3(xmax)

Punkte auf Randflächen F_4(xmin)

Punkte auf Randflächen F_5(zmax)

Punkte auf Randflächen F_6(zmin)

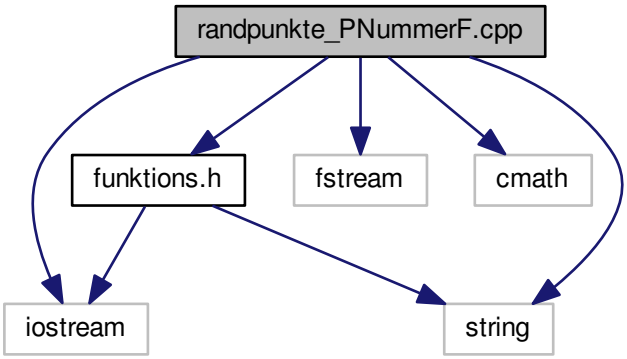
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

2.11 randpunkte_PNummerF.cpp-Dateireferenz

Randpunkte bestimmen.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für randpunkte_PNummerF.cpp:



Funktionen

- int ∗∗ randpunkte_PNummerF (int n_tetr_pkt, int NRanddim, double ∗∗p)

  Funktion um alle Randpunkte zu bestimmen
  diese Funktion ist gleich wie randpunkte_InnerePkt()
  der Unterschied dazwischen ist nur die Rückgabe.

2.11.1 Ausführliche Beschreibung

Randpunkte bestimmen.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.11.2 Dokumentation der Funktionen

2.11.2.1 int∗∗ randpunkte_PNummerF (int *n_tetr_pkt*, int *NRanddim*, double ∗∗ *p*)

Funktion um alle Randpunkte zu bestimmen

diese Funktion ist gleich wie randpunkte_InnerePkt()

der Unterschied dazwischen ist nur die Rückgabe.

Parameter

n_tetr_pkt	Anzahl aller Punkten
NRanddim	Dimension des Randpunkts
∗∗p	Punkte mit Koordinatenssystem

Rückgabe

geben die Randpunkte(auf 6 Flächen) zurück

Siehe auch

randpunkte_InnerePkt()

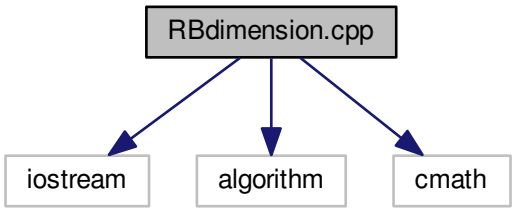
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

2.12 RBdimension.cpp-Dateireferenz

Anzahl der Randpunkte bestimmen.

```
#include <iostream>
#include <algorithm>
#include <cmath>
```

Include-Abhängigkeitsdiagramm für RBdimension.cpp:



Funktionen

- int RBdimension (int n_tetr_pkt, double ∗∗p)

  Funktion um Anzahl der Randpunkte zu bestimmen.

2.12.1 Ausführliche Beschreibung

Anzahl der Randpunkte bestimmen.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.12.2 Dokumentation der Funktionen

2.12.2.1 int RBdimension (int *n_tetr_pkt*, double ∗∗ *p*)

Funktion um Anzahl der Randpunkte zu bestimmen.

Parameter

n_tetr_pkt	Anzahl aller Punkte
∗∗*p*	Punkte mit Koordinatenssystem

Rückgabe

geben die Anzahl der Randpunkte zurück

< NRanddim: Anzahl der Randpunkt

< f1zaehler: Punkte auf Randflächen F_1

< f2zaehler: Punkte auf Randflächen F_2

< f3zaehler: Punkte auf Randflächen F_3

< f4zaehler: Punkte auf Randflächen F_4

< f5zaehler: Punkte auf Randflächen F_5

< f6zaehler: Punkte auf Randflächen F_6

< eps: Genauigkeit

< xmax: Größte Zahl in X-Richtung

< xmin: kleinste Zahl in X-Richtung

< ymax: Größte Zahl in Y-Richtung

< ymin: kleinste Zahl in Y-Richtung

< zmax: Größte Zahl in Z-Richtung

< zmin: kleinste Zahl in Z-Richtung

Randflächen suchen

Punkte auf Randflächen F_1(ymax)

Punkte auf Randflächen F_2(ymin)

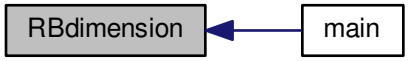
Punkte auf Randflächen F_3(xmax)

Punkte auf Randflächen F_4(xmin)

Punkte auf Randflächen F_5(zmax)

Punkte auf Randflächen F_6(zmin)

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

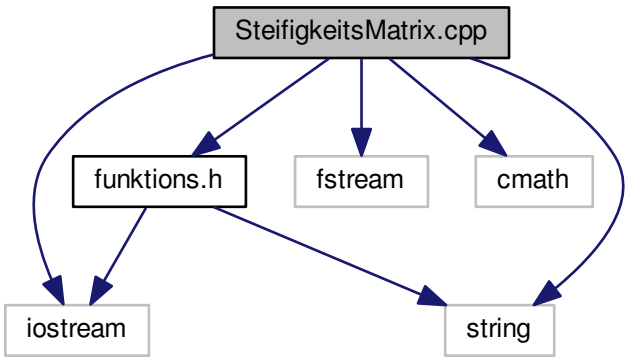


2.13 SteifigkeitsMatrix.cpp-Dateireferenz

Steifigkeitsmatrix berechnen.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für SteifigkeitsMatrix.cpp:



Funktionen

- double ∗∗ SteifigkeitsMatrix (int n_tetr_pkt, int n_tetr, int ∗∗T, double ∗∗P)

  Funktion um die Steifigkeismatrix zu berechnen.

2.13.1 Ausführliche Beschreibung

Steifigkeitsmatrix berechnen.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.13.2 Dokumentation der Funktionen

2.13.2.1 double** SteifigkeitsMatrix (int *n_tetr_pkt*, int *n_tetr*, int ** *T*, double ** *P*)

Funktion um die Steifigkeismatrix zu berechnen.

Parameter

n_tetr_pkt	Anzahl aller Punkte
n_tetr	Anzahl der Tetraeder
***T*	Tetraeder Element
***P*	Punkte mit Koordinatenssystem

Rückgabe

geben die globale Steifigkeitsmatrix und b-Matrix zurück

< *bglobal: rechte Seite b dimensionieren

< **Sglobal: Steifigkeitsmatrix dimensionieren

< **global: globale Steifigkeitsmatrix und b-Matrix zusammen:

1. 1 bis n_tetr_pkt Spalte ===> Steifigkeitsmatrix
2. n_tetr_pkt Spalt ===> b-Matrix(rechte Seite)

< Ndim: Dimension

< det_Phi: Determinante ϕ

< one_by_Phi: $\frac{1}{det\phi}$

< Slocal[4][4]: locale Steifigkeitsmatrix

< blocal[4] : locale b-Matrix

Slocal[4][4] initialisieren

blocal[4] initialisieren

$x_2 - x_1$

$y_2 - y_1$

$z_2 - z_1$

$x_3 - x_1$

$y_3 - y_1$

$z_3 - z_1$

$x_4 - x_1$

$y_4 - y_1$

$z_4 - z_1$

vermeintlicher Fehler <== ELMER

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

Rechte Seite $*$ q=4 und die 1/6 der Steifigkeitsmatrix auf die andere Seite

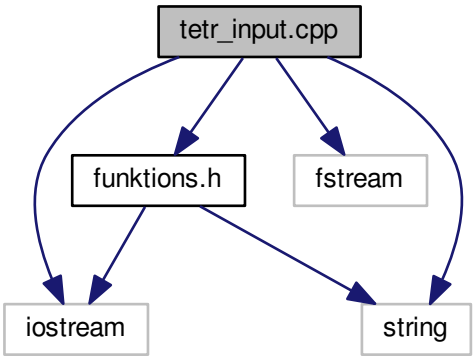
Hier ist ein Graph der zeigt, wo diese Funktion aufgerufen wird:

2.14 tetr_input.cpp-Dateireferenz

Tetraeder Element einlesen.

```
#include <iostream>
#include <string>
#include <fstream>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für tetr_input.cpp:



Funktionen

- int $**$ tetr_input (string &str, int n_tetr_pkt, int n_tetr)

  Funktion um die Tetraeder Element einzulesen.

2.14.1 Ausführliche Beschreibung

Tetraeder Element einlesen. Tetraeder Element besteht aus 4 Knoten.

Dieses Unterprogramm kann Tetraeder Element aus alle Element sortieren.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.14.2 Dokumentation der Funktionen

2.14.2.1 int∗∗ tetr_input (string & *str,* int *n_tetr_pkt,* int *n_tetr*)

Funktion um die Tetraeder Element einzulesen.

Parameter

&str	Name des Inputfiles
n_tetr_pkt	Anzahl aller Punkten
n_tetr	Anzahl des Tetraeder

< ∗∗t: Tetraederfeld dimensionieren

< idummy1: Element Index

< idummy2: Element Typ

< idummy3: Nummer der tags

< idummy4: <tags>

< idummy5: Knoten Nummer-List

< dim_element: Anzahl aller Elemente

< a,b,c,d: 4 Knoten

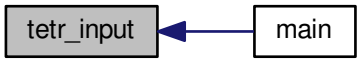
Lesen das File bis zu die Zeile ein,die zu Element gehört

Typnummer des Tetraeders ist gleich 4 ==> idummy2 == 4 ?

4 Knoten einlesen

Fehlermeldung

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

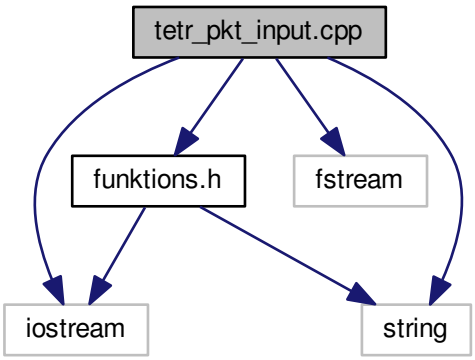


2.15 tetr_pkt_input.cpp-Dateireferenz

Koordinatenssystem aller Punkten einlesen.

```
#include <iostream>
#include <string>
#include <fstream>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für tetr_pkt_input.cpp:



Funktionen

- double ∗∗ tetr_pkt_input (string &str, int n_tetr_pkt)

  Funktion um alle Punkte einzulesen.

2.15.1 Ausführliche Beschreibung

Koordinatenssystem aller Punkten einlesen.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.15.2 Dokumentation der Funktionen

2.15.2.1 double∗∗ tetr_pkt_input (string & *str,* int *n_tetr_pkt*)

Funktion um alle Punkte einzulesen.

< **p: Punktefeld dimensionieren

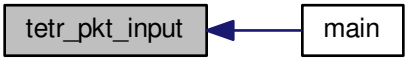
< idummy: Punktenummer

< a,b,c: XYZ-Koordinatenssystem

XYZ-Knoten einlesen

Fehlermeldung

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

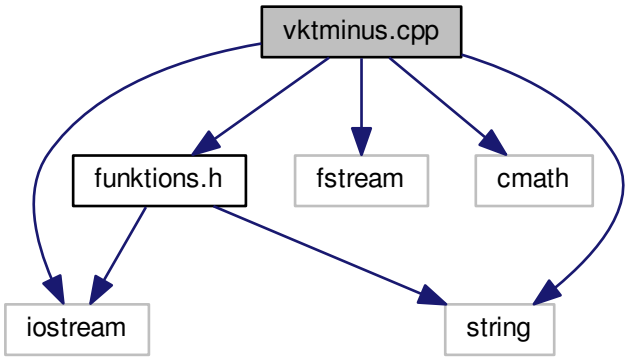


2.16 vktminus.cpp-Dateireferenz

Ein Unterprogramm für $\vec{a} - \vec{b}$.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für vktminus.cpp:



Funktionen

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

  Funktion $\vec{a} - \vec{b}$.

2.16.1 Ausführliche Beschreibung

Ein Unterprogramm für $\vec{a} - \vec{b}$.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.16.2 Dokumentation der Funktionen

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

Funktion $\vec{a} - \vec{b}$.

Dimension $\vec{a}$ soll gleich wie $\vec{b}$ sein

Parameter

*a	Vektor
*b	Vektor
arow	Dimension des Vektors a
brow	Dimension des Vektors b

Rückgabe

geben den Lösung Vektor zurück

Siehe auch

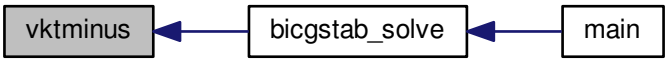
vktplus()

< *out: der Lösung-Vektor

$out_i = a_i * b_i$

Fehlermeldung

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

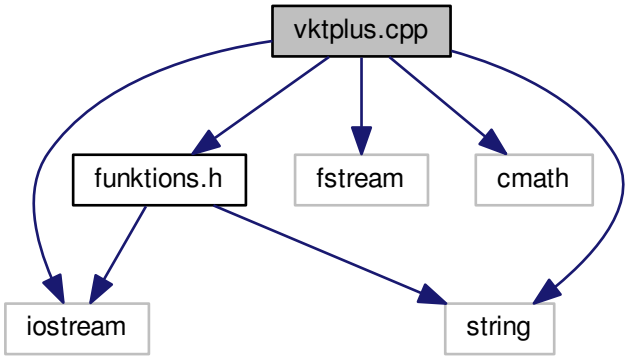


2.17 vktplus.cpp-Dateireferenz

Ein Unterprogramm für $\vec{a}+\vec{b}$.

```
#include <iostream>
#include <string>
#include <fstream>
#include <cmath>
#include "funktions.h"
```

Include-Abhängigkeitsdiagramm für vktplus.cpp:



Funktionen

- double ∗ vktplus (double ∗a, double ∗b, int arow, int brow)
  Funktion $\vec{a}+\vec{b}$.

2.17.1 Ausführliche Beschreibung

Ein Unterprogramm für $\vec{a}+\vec{b}$.

Autor

Youyu Chen

Version

1.0

Datum

09.08.2018

2.17.2 Dokumentation der Funktionen

2.17.2.1 double∗ vktplus (double ∗ *a*, double ∗ *b*, int *arow*, int *brow*)

Funktion $\vec{a}+\vec{b}$.

Dimension $\vec{a}$ soll gleich wie $\vec{b}$ sein

Parameter

$*a$	Vektor
$*b$	Vektor
arow	Dimension des Vektors a
brow	Dimension des Vektors b

Rückgabe

geben den Lösung Vektor zurück

Siehe auch

vktminus()

$<$ $*$out: der Lösung-Vektor

$out_i = a_i * b_i$

Fehlermeldung

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

Index

bicgstab_solve
    bicgstab_solve.cpp, 4
    funktions.h, 19
bicgstab_solve.cpp, 3
    bicgstab_solve, 4

constmulvkt
    constmulvkt.cpp, 7
    funktions.h, 21
constmulvkt.cpp, 6
    constmulvkt, 7

dim_tetr
    dim_tetr.cpp, 9
    funktions.h, 22
dim_tetr.cpp, 8
    dim_tetr, 9
dim_tetr_pkt
    dim_tetr_pkt.cpp, 11
    funktions.h, 23
dim_tetr_pkt.cpp, 10
    dim_tetr_pkt, 11
dot_product
    dot_product.cpp, 12
    funktions.h, 23
dot_product.cpp, 11
    dot_product, 12

FEM.cpp, 13
    main, 14
Flaechen_RB
    Flaechen_RB.cpp, 17
    funktions.h, 24
Flaechen_RB.cpp, 15
    Flaechen_RB, 17
funktions.h, 18
    bicgstab_solve, 19
    constmulvkt, 21
    dim_tetr, 22
    dim_tetr_pkt, 23
    dot_product, 23
    Flaechen_RB, 24
    matmulvkt, 25
    RBdimension, 27
    randpunkte_InnerePkt, 26
    randpunkte_PNummerF, 27
    SteifigkeitsMatrix, 29
    tetr_input, 30
    tetr_pkt_input, 31
    vktminus, 32
    vktplus, 32

main
    FEM.cpp, 14
matmulvkt
    funktions.h, 25
    matmulvkt.cpp, 34
matmulvkt.cpp, 33
    matmulvkt, 34

RBdimension
    funktions.h, 27
    RBdimension.cpp, 40
RBdimension.cpp, 39
    RBdimension, 40
randpunkte_InnerePkt
    funktions.h, 26
    randpunkte_InnerePkt.cpp, 36
randpunkte_InnerePkt.cpp, 35
    randpunkte_InnerePkt, 36
randpunkte_PNummerF
    funktions.h, 27
    randpunkte_PNummerF.cpp, 38
randpunkte_PNummerF.cpp, 37
    randpunkte_PNummerF, 38

SteifigkeitsMatrix
    funktions.h, 29
    SteifigkeitsMatrix.cpp, 42
SteifigkeitsMatrix.cpp, 41
    SteifigkeitsMatrix, 42

tetr_input
    funktions.h, 30
    tetr_input.cpp, 44
tetr_input.cpp, 43
    tetr_input, 44
tetr_pkt_input
    funktions.h, 31
    tetr_pkt_input.cpp, 45
tetr_pkt_input.cpp, 45
    tetr_pkt_input, 45

vktminus
    funktions.h, 32
    vktminus.cpp, 47
vktminus.cpp, 46
    vktminus, 47
vktplus
    funktions.h, 32
    vktplus.cpp, 48

vktplus.cpp, 48
    vktplus, 48