HANA Numerik SEP Projekt - Heissluftballon

Autoren

Dominique Alguacil

Selim Arslan

Kaya Ercihan

Institution

ZHAW School of Engineering

Dozent

Simon Stingelin

Abgabedatum

13.01.2023

1 Inhaltsverzeichnis

2 Aufgabe1

2.1 Einleitung

In Aufgabe 1 ist das erste Ziel die schwachen Gleichungen aus den starken Gleichungen zu errechnen. Schwache Gleichungen sind eine Verallgemeinerung der starken Gleichungen, um numerische Algorithmen mit machbarem Rechenaufwand auf die schwachen Gleichungen anwenden zu können. Die schwachen Gleichungen beziehen sich in unserem Beispiel auf das Temperaturfeld (2.a.1), auf das Druckfeld (2.a.2) und auf das Strömungsfeld (2.a.3).

2.2 Schwache Gleichungen

2.2.1 Gleichung 2.a.1 für Temperaturfeld

Für das Temperaturfeld haben wir die schwache Gleichung wie folgt errechnet:

$-div(\kappa * \nabla T) + \vec{u} * \nabla T = 0$
$\vec{u}(x) * \nabla T * v(x) = v(x) * div(\kappa * \nabla T)$ // $\vec{u}(x) \in V,\ v(x) \in V,\ x \in \Omega$
Ziel $\vec{u}(x) \rightarrow$ Strömungsfeld berechnen
Integrieren
$\int_\Omega \vec{u}(x) * \nabla T * v(x) * dx = \kappa * \int_\Omega div(\kappa * \nabla T) * v(x) * dx$ // κ nur in spezifischen Fall 1-Dimensional

Allgemeiner Divergenz-Satz: $\int_{\partial\Omega} n * \vec{w}(x) * dA = \int_\Omega div\left(\vec{w}(x)\right) * dx$ // $\vec{w}(x) \in V$

Wir wählen $v(x)$, sodass $v(x) = 0$ auf $\partial\Omega$
$\int_\Omega div\left(\kappa * \nabla v(x)\right)dx = \int_\Omega div(\kappa \nabla T) * v(x) + \kappa * \nabla T * \nabla v(x)dx = \int_{\partial\Omega} \kappa * \nabla T * v(x)dA$
$\int_\Omega \kappa * \nabla T * v(x)dx = -\int_\Omega \kappa * \nabla T * \nabla v(x)dx + \int_{\partial\Omega} \kappa * \nabla T * v(x)dA$]
Nicht auf ganz Ω sondern nur auf den Rand, wegen des Outer (deswegen statt $\partial\Omega$ $reduziert\ sich\ auf\ \Gamma_{Outer}$ aufgrund der Robin-Randbedingung.

$\int_\Omega \vec{u}(x) * \nabla T * v(x)dx = -\int_\Omega \kappa * \nabla T * \nabla v(x)dx + \int_{\Gamma_{\text{Outer}}} \kappa * \nabla T * v(x)dA \quad für\ alle\ v\ in\ V$

2.2.2 Gleichung 2.a.2 für Druckfeld

Für das Druckfeld haben wir die schwache Gleichung wie folgt errechnet:

$$\nu * \Delta\vec{u} = -\frac{1}{\rho} * \nabla p - \vec{g} * \alpha * (T - T_0) = 0$$

$$\rightarrow -\nu * div\big(grad(\vec{u})\big) + \frac{1}{\rho} * grad(p) + \vec{g} * \alpha * (T - T_0) = 0$$

$$\rightarrow -\nu * \begin{pmatrix} div\big(grad(u_x)\big) \\ div\left(grad\big(u_y\big)\right) \end{pmatrix} + \frac{1}{\rho} * \begin{pmatrix} \partial_x p \\ \partial_y p \end{pmatrix} + \begin{pmatrix} g_x \\ g_y \end{pmatrix} * \alpha * (T - T_0) = 0$$

Daraus haben wir zwei Gleichungen gemacht. Jeweils eine Gleichung für die erste und zweite Zeile.
Zeile 1:

$$-\nu * div\left(grad\big(u(x)\big)\right) + \frac{1}{\rho} * \partial_x p + g_x * \alpha * (T - T_0) = 0$$

Eine Testfunktion q(x) einfügen:

$$\int_\Omega -\nu * div(\nabla u_x) * q(x) + \frac{1}{\rho} * (\partial_x p) * q(x) + g(x) * \alpha * (T - T_0) * q(x) dx = 0 \quad (1.1)$$

$$\int_\Omega -\nu * div(\nabla u_x) * q(x) dx \rightarrow In\ Erinnerung\ behalten$$

$$\int_\Omega div\big(\nabla u_x * q(x)\big) dx = \int_\Omega div(\nabla u_x) * q(x) + \nabla u_x * \nabla q(x) dx = \int_{\partial\Omega} n * \nabla u_x * q(x) dA = 0$$

Divergenz-Satz → Aufgrund des Divergenz-Satzes (partiell integrieren, Kettenregel) wurde so umgeformt

$\int_\Omega div(\nabla u_x) * q(x) dx = \int_\Omega -\nabla u_x * \nabla q(x) dx$ // $* -\nu$ auf beiden Seiten um Term zu erreichen

$-\nu * \int_\Omega div(\nabla u_x) * q(x) dx = \int_\Omega \nu * \nabla u_x * \nabla q(x) dx$

Einfügen in 1.1

$$\int_\Omega \nu * \nabla u_x * \nabla q(x) dx + \frac{1}{\rho} * -p * \partial_x\big(q(x)\big) + g_x * \alpha * (T - T_0) * q(x) dx = 0$$

Zeile2:
Wir nehmen eine neue Testfunktion k(x) und machen alles wie in «Zeile 1» → Wir finden die schwache Gleichung:

$$\int_\Omega \nu * \nabla u_y * \nabla k(x)\, dx + \frac{1}{\rho} * -p * \partial_y\big(k(x)\big) + g_y * \alpha * (T - T_0) * k(x) dx = 0$$

→ Schwache Gleichung für den Druck

2.2.3 Gleichung 2.a.3

Für das Strömungsfeld haben wir die schwache Gleichung wie folgt errechnet:

$$div(\vec{u}) = 0$$

Wir wählen eine Testfunktion t(x)

$\rightarrow div(\vec{u}) * t(x) = 0$

$\rightarrow \int_\Omega div(\vec{u}) * t(x) dx = 0$

Ansatz: $div() = \nabla \rightarrow \nabla\vec{v} = \partial x * v_x + \partial y * v_y$; $\rightarrow \nabla skalar = grad(skalar) = \begin{pmatrix} \partial_x s \\ \partial_y s \end{pmatrix}$;

$\rightarrow \nabla * \nabla = \Delta = Laplace$

$\int_\Omega div\big(\vec{u} * t(x)\big) dx$ //Divergenz entsprich Ableitung → Kettenregel, deswegen wird so umgeformt

Divergenzsatz: $\int_{\partial\Omega} n * \vec{u} * t(x) dA = 0$

// Weil der Rand 0 ist → Dirichlet → Wird nur über Rand integriert → Multplikation = 0

$\int_\Omega div(\vec{u}) * t(x) + \vec{u} * \nabla t(x) dx$

$\int_\Omega div(\vec{u}) * t(x) dx = -\int_\Omega \vec{u} * \nabla t(x) dx = 0$

2.2.4 Zusammenstellung der schwachen Gleichungen als Gleichungssystem

$$\int_\Omega \vec{u}(x) * \nabla T * v(x)dx = -\int_\Omega \kappa * \nabla T * \nabla v(x)dx + \int_{\Gamma_{\text{Outer}}} \kappa * \nabla T * v(x)dA$$

$$\int_\Omega \nu * \nabla u_x * \nabla q(x)dx + \frac{1}{\rho} * -p * \partial_x\big(q(x)\big) + g_x * \alpha * (T - T_0) * q(x)dx = 0$$

$$\int_\Omega \nu * \nabla u_y * \nabla k(x)\,dx + \frac{1}{\rho} * -p * \partial_y\big(k(x)\big) + g_y * \alpha * (T - T_0) * k(x)dx = 0$$

$$\int_\Omega div(\vec{u}) * t(x)dx = -\int_\Omega \vec{u} * \nabla t(x)dx = 0$$

2.2.5 Gleichungen in Code

Diese Gleichungen sind alle im Stokes-Flow und im Navier-Stokes-Flow enthalten. Beispielsweise kann die Gleichung 2.a.3 im Stokes Flow folgendermassen vorgefunden werden: $a+= div\_u \cdot q \cdot r \cdot dx$. Durch diese $+=$ Operation wird jede der von uns ebenfalls ausgerechneten Schwachen Gleichungen zur Bilinearform dazugerechnet. Somit können wir also im Stokes-Flow und im Navier-Stokes-Flow verifizieren, ob unsere schwachen Gleichungen korrekt sind.

Im Folgenden ist eine Gegenüberstellung von Python-Code zu Formel im Stokes-Flow zu sehen:

Code	Formel
BilinearForm(X, symmetric = False) u* grad(T)* S) * r * dx #Konvektion kappa * grad(T) * grad(S) Bilinearform beinhaltet die Randbedingung: gamma *(T- Tamb)*S*r*ds('outer')	$\int_\Omega \vec{u}(x) * \nabla T * v(x) dx =$ $-\int_\Omega \kappa * \nabla T * \nabla v(x) dx$ $+\int_{\Gamma_{Outer}} \kappa * \nabla T * v(x) dA$
Für X-Komponente: nu* grad(ur)* grad(vr) #Stokes-Flow x, innere Reibung 1/ rho * p * div_v #Druckterm g * alpha * grad(T)[1] * vz #Auftriebskraft	$\int_\Omega \nu * \nabla u_x * \nabla q(x) dx$ $+\frac{1}{\rho} * -p * \partial_x(q(x))$ $+g_x * \alpha * (T - T_0) * q(x) dx$
Für Y-Komponente: nu * grad(uz) * grad(vz) #Stokes-Flow y, innere Reibung 1/ rho * p * div_v #Druckterm g * alpha * grad(T)[1] * vz #Auftriebskraft	$\int_\Omega \nu * \nabla u_y * \nabla k(x) dx$ $+\frac{1}{\rho} * -p * \partial_y(k(x))$ $+g_y * \alpha * (T - T_0) * k(x) dx$
div_u * q *r * dx #Inkompressible	$\int_\Omega div(\vec{u}) * t(x) dx = 0$

Im Folgenden ist eine Gegenüberstellung von Python-Code zu Formeln im Navier-Stokes-Flow zu sehen:

Code	Formel
Rest ist wie in Stokes-Flow.	Rest ist wie in Stokes-Flow.
u*(grad(ur)*vr+grad(uz)*vz	$\int_\Omega \nu * \nabla u_{x,y} * \nabla q(x) dx$

3 Aufgabe 2

3.1 Einleitung

Für Aufgabe 2 haben Python-Funktionen für Stokes-Flow, Navier-Stokes-Flow Newton erhalten (Listing 5-7). Diese Funktionen haben wir umstrukturiert und angewendet. Dabei ist der Parameter «a» die Bilinearaform, welche aus dem Stokes-Flow hervorgeht und «gfx» die Grid-Function. Weiter beinhaltet der Stokes-Flow das Strömungsfeld, die Testfunktion «q», den Robin-Randwert (gamma), den Ortsvektor (r), die dynamische Viskosität (nu), diese wiederum beinhaltet die kinetische Viskosität (mu), die Dichte (rho), den Wärmeausdehnungskoeffizienten (alpha), Wärmeleitfähigkeit (kappa) und den konkatenierten Raum (X).

3.2 Definitionen im Stokes-Flow

In dieser Aufgabe bezeichnet das Strömungsfeld $\vec{u}$ den Strom der Wärme in zweidimensionaler Form. Die Testfunktion q und sind alle Basisfunktionen des Raumes. Der Robin-Randwert ist der Wärmestrom gegen aussen. Der Ortsvektor (r) ist der Ausgangspunkt für beispielsweise das Strömungsfeld $\vec{u}$. Die dynamische Viskosität ist ein Mass für die Zähflüssigkeit eines Fluids oder in unserem Fall der Luft [1]. Die kinetische Viskosität ist die dynamische Viskosität geteilt durch die Dichte (rho) [1]. Der Wärmeausdehnungskoeffizient ist wie folgt definiert:

$\frac{1}{T}$ $[T \coloneqq Temperatur\ in\ Kelvin]$.

Der konkatenierte Raum (X) ist ein FE-Space und bezeichnet den Zusammenschluss der Räume W, Vr, Vz und Q. Der Raum W definiert das H1-Space für die Temperatur (T).

Der Raum Vr definiert den H1-Space für die radiale Richtung der Strömung. Im Vr wird eine Dirichlet-Randbedingung auf die Symmetrieachse, auf den äusseren und den inneren Rand gelegt.
Die Bedeutung einer Dirichlet-Randbedingung ist, dass die definierten Randbedingungen Null sind.
Der Raum Vz definiert den H1-Space für die Z-Richtung der Strömung. Der Raum Q definiert den H1-Space für den Druck. In Q muss die Dimension um eins reduziert werden, sodass der Lagrange-Parameter und das Strömungsfeld übereinstimmen.

3.3 Newton-Iteration

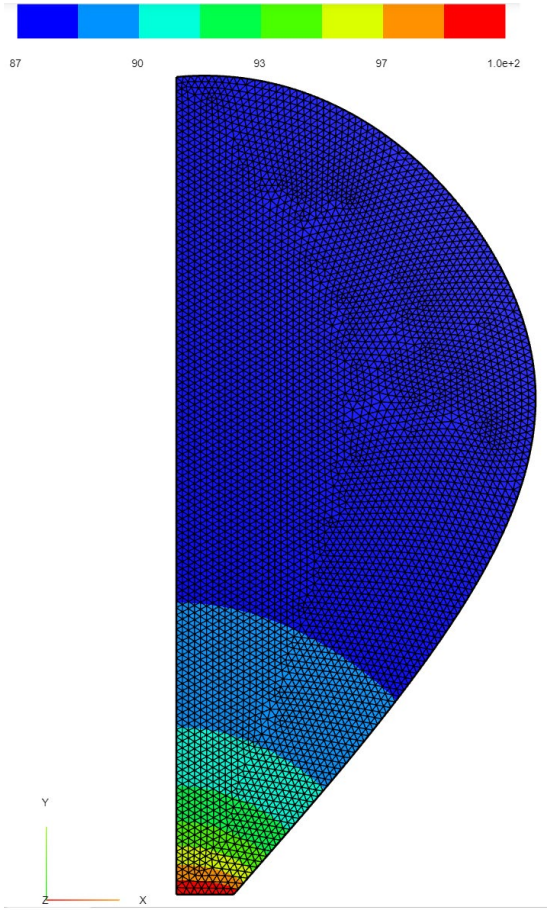
Diese Newton-Iteration ist für den N-Dimensionalen und nicht für den 1-dimensionalen Raum gebaut. Aus diesem Grund konnten wir nicht die bereits in der Höheren Mathematik für Informatiker verwendete Funktion kopieren.

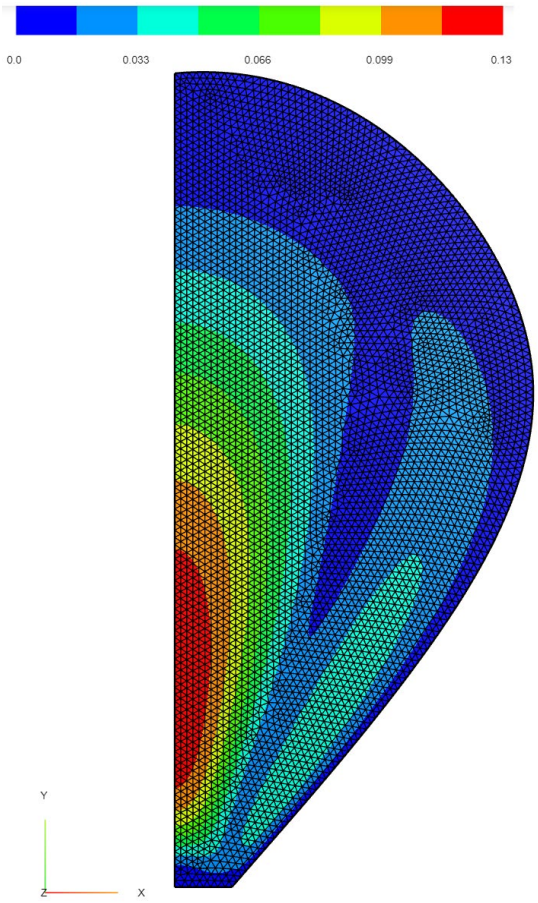
Wir verwenden hier NGSolve-Funktionen. Mit der Bilinearform und der NGSolve-Funktion «Apply» wird die Funktion $F(u_N)$ erzeugt. Diese Funktion heisst schwache Gleichung. Danach wird mit der Bilinearform und dem Vektor der «Gridfunction», über die Funktion «AssembleLinearization» die Jacobi-Matrix durch Ableitung von jeder einzelnen Komponente erzeugt. NGSolve kann nichtlineare Bilinearformen behandeln. Anschliessend wird das lineare Gleichungssystem für die Freiheitsgrade gelöst, welche die unbekannten Variablen darstellen und in einem Vektor zusammengefasst sind. Mit jedem weiteren Schritt wird der Vektor aktualisiert. Weiter wird überprüft, ob wir die maximale Anzahl an Iterationen erreicht wird oder die Toleranz unterschritten haben. Tritt eine der Bedingungen ein, brechen wir die Iteration ab.

Die Freiheitsgrade sind die unbekannten Parameter, die nicht bereits durch die Dirichlet-Randbedingungen der H1-Räume, definiert sind. Der Dirichlet-Rand ist bei den einzelnen H1-Räumen absichtlich so gewählt worden um exakt die richtige Anzahl an Freiheitsgraden, die mit der Anzahl an Gleichungen korrelieren muss, zu definieren. Wenn mehr Freiheitsgrade als Gleichungen existieren ist zu wenig Information vorhanden, um Resultate für die Freiheitsgrade zu erhalten.

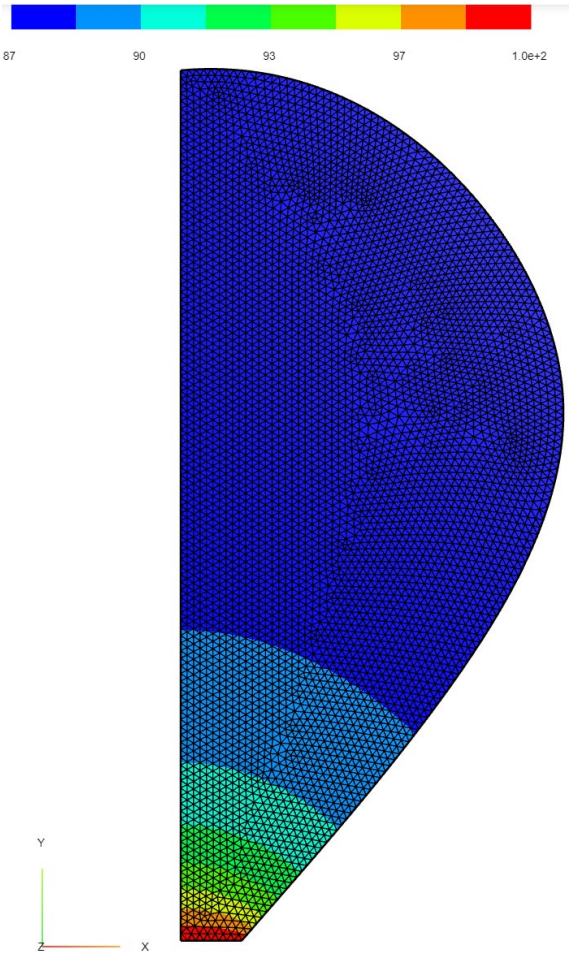
3.4 Aufgabe 2.1

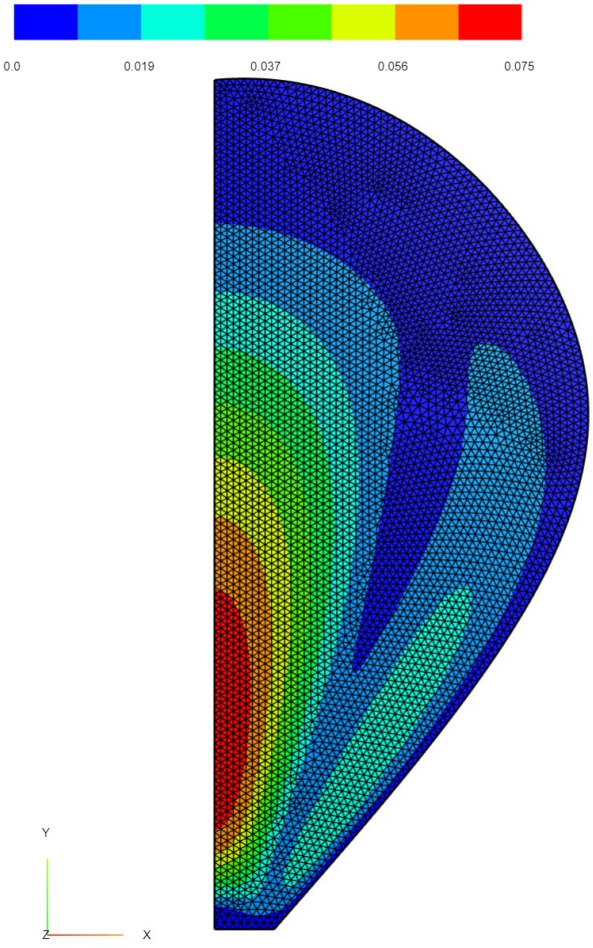
In dieser Teilaufgabe wird der Stokes-Flow ausgeführt. Am unteren Rand des Finiten dreidimensionalen Elements wird die Temperatur auf 100°C gesetzt. Dann wird durch die im Stokes-Flow generierte Bilinearform und die Grid-Funktion die Newton-Iteration durchgeführt. Danach wird die Lösung des Stokes-Flows, der Temperatur und der Geschwindigkeit geplottet. Die Temperatur (links) und das Strömungsfeld (rechts) sehen folgendermassen aus:





Nachfolgend wird die Temperatur, die aus der Gridfunction aus der Newton-Iteration hervorgeht (links) und das Strömungsfeld rechts geplottet:





3.5 Aufgabe 2.2

In dieser Aufgabe geht es darum zu zeigen, dass die kinetische Viskosität und die Viskosität der Luft divergieren, wenn diese aneinander angeglichen werden. Um dies zu zeigen, soll der Navier-Stokes-Flow eingesetzt werden. Zu Beginn haben wir eine sehr generöse kinetische Viskosität. Diese wird nun schrittweise von 31 bis zur Viskosität der Luft (1) reduziert. Anschliessend wird diese reduzierte kinetische Viskosität gesetzt und die Newton-Iteration mit der Bilinearform aus dem Navier-Stokes-Flow und der Gridfunction durchiteriert.

Da die Diskretisierung in dem H1-Raum bei unserem Dozenten mit «orderinner=Order+1» gewählt wurde, kann man auch nicht stetige Funktionen im L2 rechnen. Für die Strömung muss man die Order also um 1 erhöhen, dann konvergiert es für die Strömung. Diese Diskretisierung können wir so aber nicht auf dem System der ZHAW ausführen. Es wurde vereinbart, dass die das «orderinner = Order+1» weggelassen werden kann.

Momentan erhalten wir trotzdem noch ein «no convergence» als Resultat. Dies ist aber laut Absprache mit Herrn Stingelin in Ordnung. Wir haben die Toleranz von $5 * 10^{-9}$ auf $1 * 10^{-8}$ erhöht. Zusätzlich haben wir noch die Skalierung der Geometrie von 0.1 auf 0.075 geändert, damit die Iteration numerisch stabiler werden. Dieses Vorgehen hat die Konvergenz von der vierten Iteration von der kinetischen Viskosität 31.0 auf eine Divergenz bei der fünfundzwanzigsten Iteration von der kinetischen Viskosität 2.747919143876338 verbessert.

Durch weiteres Verbessern wäre eine Konvergenz mit der kinetischen Viskosität von 1.0 möglich. Dies ist aber unter Absprache mit Herrn Stingelin nicht notwendig.

4 Aufgabe 3

4.1 Einführung

Die Lösung befindet sich in einem polaren Koordinatensystem, wobei diese Aufgabenstellung sich im Spezialfall der axialsymmetrischen Lösung befindet. Das bedeutet, dass wir ein dreidimensionales Problem als zweidimensionales Problem lösen können. Dies ist viel effizienter, da durch das Wegfallen einer Dimension auch eine Variable entfällt. Folglich sind weniger Gleichungen zu lösen. Diese Lösung ist möglich, weil in jedem Winkel (phi) die Lösung exakt gleich ist und somit phi weggelassen beziehungsweise auf null gesetzt werden kann. Das polare Koordinatensystem setzt sich aus den folgenden Koordinaten zusammen:

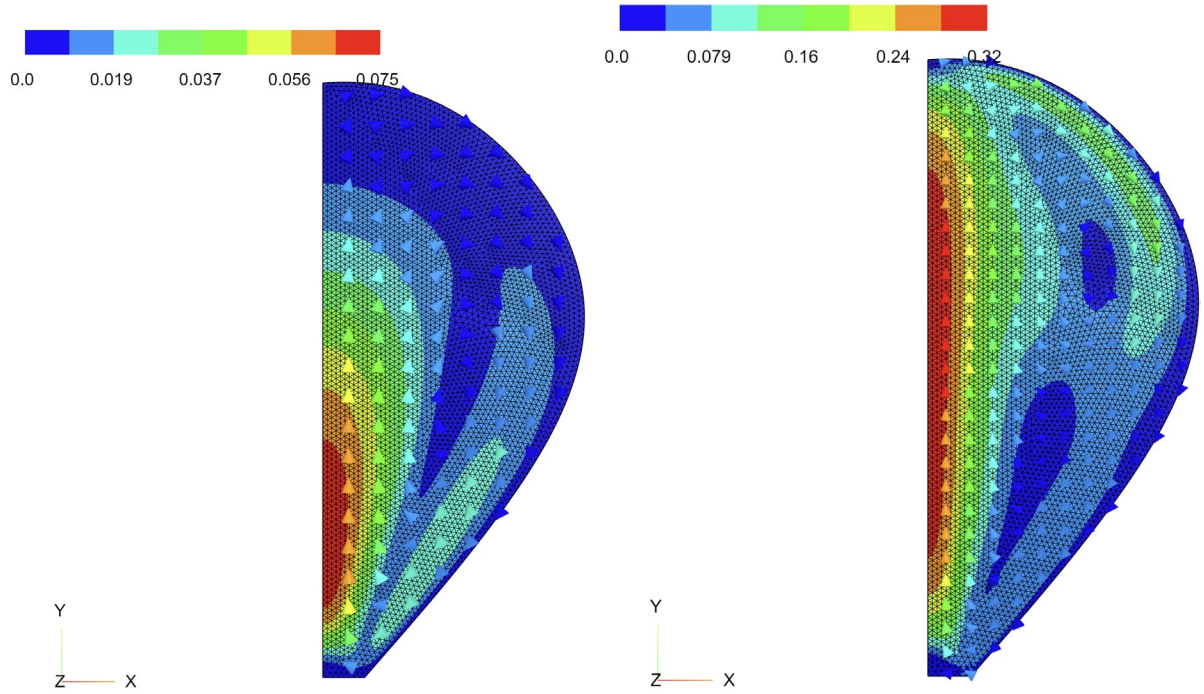
$$Koordinaten = \begin{pmatrix} r \\ phi \\ z \end{pmatrix}$$

4.2 Dimensionsreduktion

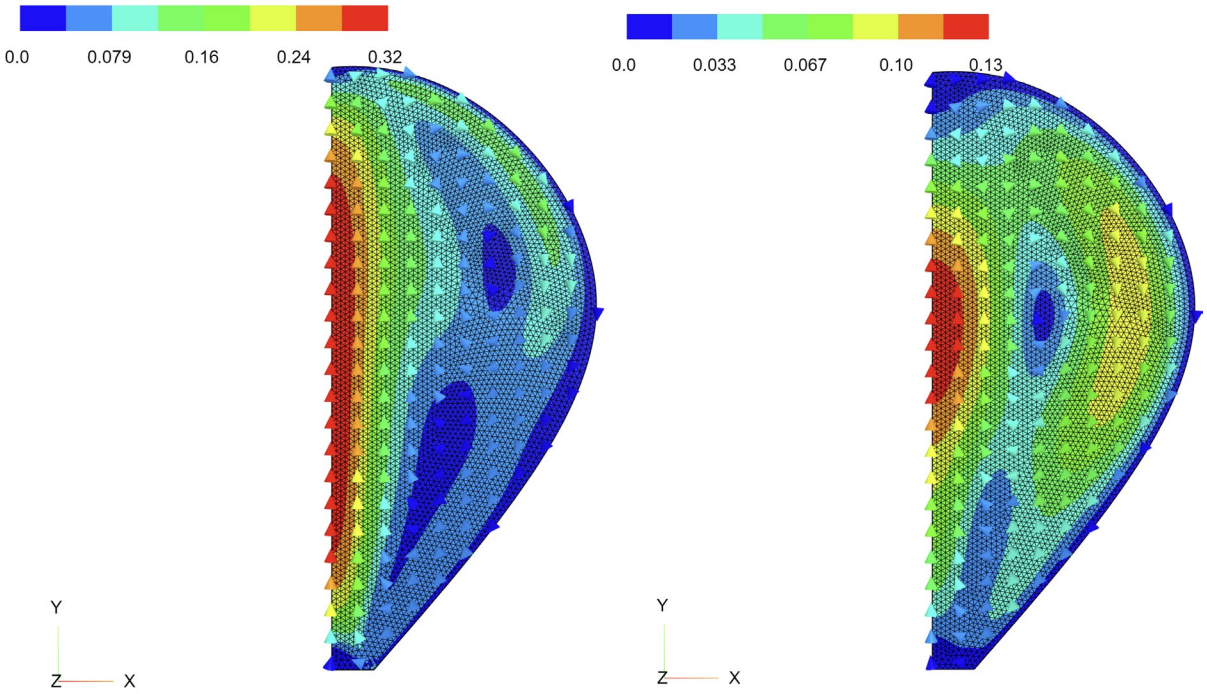
Um eine Dimension zu reduzieren, wird in dieser Aufgabenstellung die r-Komponente (also phi) aus dem Stokes- und dem Navier-Stokes-Flow der Python-Funktion herausgelöscht.

4.3 Resultate

Nach der veränderten Dimension haben wir zwei neue Python-Funktionen («stokesFlow2d» und «navierStokesFlow2d») erstellt, die wir verwenden, um die Newton-Iterationen zu iterieren. Der Stokes- und der Navier-Stokes-Flow wird, nachdem sie durch die Newton-Iteration gelöst wurden, geplottet. In den nachfolgenden zwei Abbildungen wird das Strömungsfeld vom Heissluftballon mittels Stokes-Flow abgebildet. Nachfolgend ist das Strömungsfeld abgebildet, links dreidimensional und rechts zweidimensional.



Nachfolgend dieselbe Abbildung wie oben nur mit dem Navier-Stockes-Flow.



5 Aufgabe 4

5.1 Einführung

Für die Aufgabe 4 wurde die Masse, wie vorgegeben, auf 200 Kilogramm, der atmosphärische Druck auf 1013.25 mBar und die Umgebungstemperatur T_{amb} auf 0 °C angenommen.

Die im Python-Source-Code gegebene Variable «rhoT» ist die temperaturabhängige Funktion der Dichte und die Variablen «massHotAir» und «massColdAir» sind die Massen der Luft innerhalb und ausserhalb des Ballons.

5.2 Auftriebskraft

Die Auftriebskraft lässt sich durch folgende Formel berechnen $F_{Auftrieb} = -\rho * g * V$, wobei "$\rho$" die Dichte des Mediums, «g» die Erdanziehungskraft (9.81$\frac{m}{s^2}$) und V das Volumen darstellt, berechnet.
Da «massHotAir» und «massHotAir» durch die Formel «Masse = Volumen * Dichte» gegeben ist, musste nur noch die Erdanziehungskraft multipliziert werden. Das Resultat dieser Überlegung war die folgende Gleichung:
$F_{Auftrieb} = massColdAir * g$

5.3 Höhe

Die Originalhöhe wurde aufgrund des gegeben Python-Source-Code (2.85 (Original-Höhe) * 0.075 (Skalierungsfaktor)) mit 0.21375 angenommen.
Da wir wissen, dass folgendes gegeben ist:

$$alpha = \sqrt[3]{\frac{masse * g}{massColdAir - massHotAir * g}} = \sqrt[3]{\frac{masse}{massColdAir - massHotAir}}$$

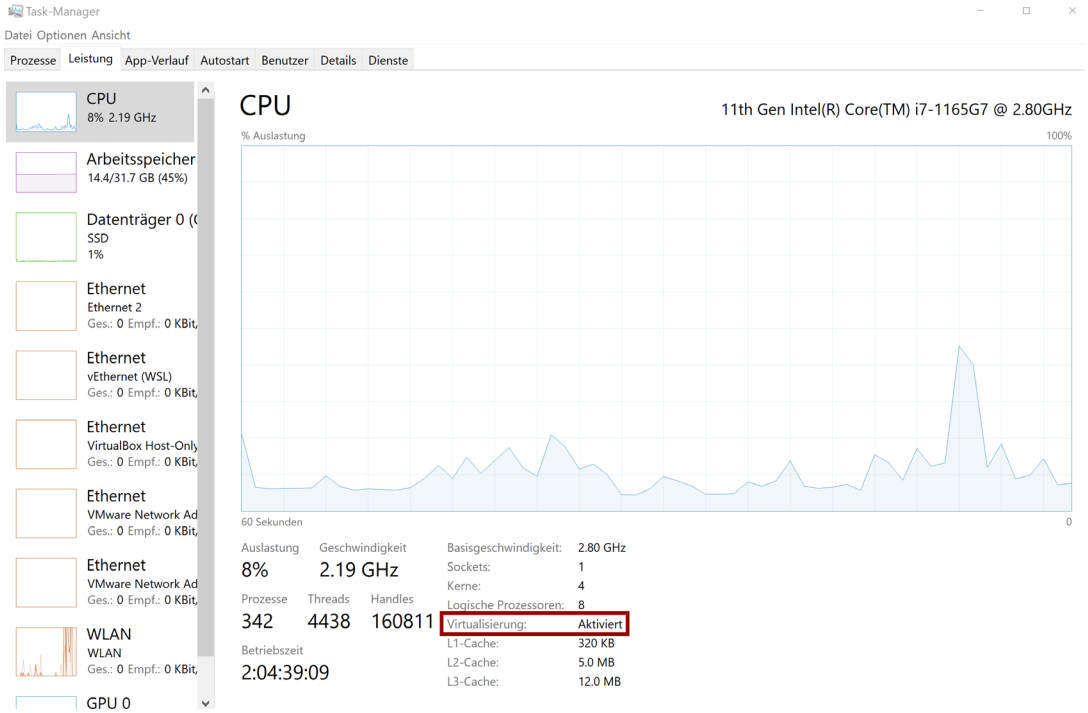
Das ist die Masse, die wir haben, dividiert durch die Masse, die wir effektiv haben.
Damit kann die neue Höhe berechnet werden.

$$h_{neu} = h_{original} * alpha$$

6 Docker

Dies ist eine Anleitung für den Einsatz von Docker im HANA-Kurs. Falls NGSolve lokal auf dem Windows-Rechner verwendet werden soll, kann dies folgendermassen gemacht werden:

1. Als erstes muss im Task-Manager geprüft werden, ob die Virtualisierung aktiviert ist.



   Falls die Virtualisierung inaktiv ist, soll die Anleitung in «Virtualisierung aktivieren» im Kapitel 6.1 durchgegangen werden.
2. Docker herunterladen: https://www.docker.com/
3. Nach der Docker-Installation kann auf dem als Administrator ausgeführtem CMD mittels folgendem Command das Docker-Image heruntergeladen werden:
   «docker pull ercihan/netgen»
4. Nun kann ein Docker-Container mittels folgendem Command erzeugt werden:
   «C:\Windows\system32>**docker run -it --memory="8g" ercihan/netgen**
   Ausgabe:
   `<string>:1: DeprecationWarning: The distutils package is deprecated and slated for removal in Python 3.12. Use setuptools or check PEP 632 for potential alternatives`
   `<string>:1: DeprecationWarning: The distutils.sysconfig module is deprecated, use sysconfig instead`
   `root@71c85cf6592b:/#»`
   Die «DeprecationWarnings» können so weit ignoriert werden, sie haben keinen Einfluss auf das korrekte Funktionieren der Umgebung für unsere Zwecke in diesem Modul. Wie man sieht, ist man bereits in der Bash des Containers eingeloggt. Nun kann der Container beispielsweise in Visual Studio Code verwendet werden, indem man sich im VSCode zum Container verbindet und auf dessen Python-Umgebung das Jupyter-Notebook ausführt.
5. Falls zu einem späteren Zeitpunkt erneut auf die Konsole zugegriffen werden soll, funktioniert das mit folgendem Command:
   «`docker exec -it containerName bash`»

6.1 Virtualisierung aktivieren

Wenn die Virtualisierung nicht aktiv ist, muss diese im BIOS/UEFI aktiviert werden.

1. Um in das BIOS zu gelangen, muss der Computer neu gestartet werden.
2. Beim Starten des PC ins BIOS zu gelangen, muss eine Herstellerspezifische Taste gedrückt werden. Normalerweise ist es die F1-, F2-, F3-, F10-, F12-, ESC- oder DEL-Taste.
3. Wo die Virtualisierung eingestellt werden kann, variiert von Hersteller zu Hersteller. Meistens kann die Virtualisierung unter dem «Avanced»-Tab aktiviert werden.
4. Speichern Sie die Einstellungen und starten Sie denn Computer neu.

7 Literaturverzeichnis

[1] H. Kuchlin, Taschenbuch der Physik, Leipzig: Fachbuchverlag Leipzig, 1996.

8 Abkürzungsverzeichnis

Abkürzung	Beschreibung
FE	Finites Element
Stokes-Flow	Gesamte nichtlineare schwache Gleichung
Trial-Funktion	Platzhalter für unsere Lösung. Sämtliche Basisfunktionen, sodass es die Bilinearform ergibt. Sobald die Trial-Funktion in die Gleichung hinzugefügt wurde, ergibt sich daraus eine Matrix. Ergo ist dies eine Linearform.

Anhang

Heissluftballon

December 22, 2022

[1]:
```
from netgen.meshing import Mesh as NGMesh # Vorsicht es gibt Mesh auch in␣
↪ngsolve!
from netgen.meshing import MeshPoint, Pnt, Element1D, Element0D
from ngsolve import *
from ngsolve.webgui import Draw
import numpy as np
import scipy as sp

def NewtonSteps (a, gfx, nMax=10, tol=1e-9):
    '''
    gfx = Grid function
    The grid function is an approximation of the solution of the boundary value␣
↪problem.
    This grid function is used for discretising the boundary value problem on␣
↪the local fine grid.
    '''
    X = gfx.space #Will be defined as a 'raum'
    # vectors for newton steps
    res = gfx.vec.CreateVector()
    du = gfx.vec.CreateVector()

    for it in range (nMax):
        print('Iteration ', it)
        with TaskManager (): #NGSolve supports shared memory parallelization␣
↪using its built-in TaskManager
            # From here on our code
            #F(u_n) // Chapter 12.1
            a.Apply(gfx.vec, res) #res=Residuum

            #F'(u_n) // -> Jacobi-Matrix
            a.AssembleLinearization(gfx.vec) #Bills the second variation -> the␣
↪jacobi matrix -> linearisation point is the vector
            #Can be seen in the Gelfand-equation exercise ; Linearisation /␣
↪derivation of each component will be billed
            inv = a.mat.Inverse(X.FreeDofs())#Feedofs=Freiheitsgrade -> alles␣
↪ohne Dirchichletrand #inverse() = lineares Gleichungssystem loesen
```

```
            # F'(u_n) du = F(u_n)
            du.data = inv * res

            # update iteration
            # u_n+1 = u_n - du
            gfx.vec.data -= du

            #stopping criteria
            stopcritval = sqrt(abs(InnerProduct(du,res)))
            print("Stopcritvalue: " + str(stopcritval))
            if stopcritval < tol:
                break
    if stopcritval < tol:
        return 1 #gfx is global return here means if it worked or not
    else:
        return 0

'''
Trial-Function = Platzhalten für unsere Loesung -> fuer alle TrialFunktionen in␣
 ↪Space -> Saemtliche Basisfunktionen sodass Bilinearform ergibt
                Sobald TrialFunktion in Gleichung drin ist, dann ergibt das␣
 ↪eine Matrix eine Linearform -> Dieskretisiert bei Bilinearform immer eine␣
 ↪Matrix
Test-Function = Funktionen die man dranmultipliziert -> Saemtliche␣
 ↪Basisfunktionen
Bilinear-Form = Schwache Gleichung in unserem Fall -> Gesamte nichtlineare␣
 ↪schwache Gleichung, sodass alles gleich 0 ist.
Freiheitsgrade = Unbekannte linear-Faktoren von unseren Grid-Function (Mesh)␣
 ↪oder Loesung die wir suchen -> man kann 9 Freiheitsgrade haben aber nur␣
 ↪einen Unbekannten wenn die anderen 8 sind durch Rand-bedingung bereits␣
 ↪gegeben (bspw. Dirichletrandbedingung)
                bei Freedofs -> ist es durch den konkatnierten Raum und deren␣
 ↪definierten Dirchletraendern so zugeschnitten, damit wir die richtige Anzahl␣
 ↪an unbekannten Vars kriegen
ur = Trial-Function of Testfunction vr
vr = Testfunction of Trial-Function ur
uz = Trial-Function of Testfunction vz
vz = Testfunction of Trial-Function uz
p = Trial-Function of Testfunction q
q = Testfunction of Trial-Function p
T = Trial-Function of Testfunction S / or Temperature
S = Testfunction of Trial-Function T
r = Ortsvektor
z = Ortsvektor
X = Concatenated space (W,Vr ,Vz ,Q)
rho = Dichte
```

```
alpha = 'warmeausdehnungskoefficient' = 1/T
kappa = 0.0262W/(mK) = 'Waermeleitfaehigkeit'
g = 9.81m/s^2
gamma = Robin-Randwert -> Wie gross ist der Waermestrom gegen aussen -> im Text␣
 ↪ist es betta
Tamb = Umgebungstemperatur
dx = Volumenelement / Integrator
nu = mu / rho ; mu = scaleMu * 18.2e-6 / np.sqrt(293.15) * sqrt(273.15 + T) ;␣
 ↪-> dynamische viskositaet / mu=kinetische viskositaet
'''
def stokesFlow(ur, r, vr, T, X, nu, alpha, uz, vz, q, p, rho, g, gamma, Tamb,␣
 ↪S, dx, kappa):
    # Stokes flow
    div_u = ur/r+ grad(ur)[0]+ grad(uz)[1]
    div_v = vr/r+ grad(vr)[0]+ grad(vz)[1]

    u = CoefficientFunction ((ur, uz))#For vectorial usage of ur and uz
    conv = u* grad(T)

    a = BilinearForm(X, symmetric = False)
    a += (kappa * grad(T) * grad(S)+ conv * S) * r * dx
    a += gamma *(T- Tamb)*S*r*ds('outer')#Randintegral
    a += (nu*ur/r **2* vr + nu* grad(ur)* grad(vr) + nu * grad(uz ) * grad(vz)␣
 ↪+ 1/ rho * p * div_v + g * alpha * grad(T)[1] * vz)*r*dx
    a += div_u * q *r * dx
    return a, u, div_u, div_v, conv

def navierStokesFlow(ur, r, vr, T, X, nu, alpha, uz, vz, q, p, rho, g, gamma,␣
 ↪Tamb, S, dx, kappa, div_v, div_u):
    #Navier-Stokesflow
    aNS = BilinearForm(X, symmetric=False)
    aNS += (kappa * grad(T) * grad(S) + conv * S) * r * dx
    aNS += gamma * (T - Tamb) * S * r * ds('outer')
    aNS += (nu * ur / r**2 * vr + nu * grad(ur) * grad(vr) + nu * grad(uz) *␣
 ↪grad(vz) + u * (grad(ur) * vr + grad(uz) * vz) + 1 / rho * p * div_v + g *␣
 ↪alpha * grad(T)[1] * vz) * r * dx
    aNS += div_u * q * r * dx
    return aNS
```

```
importing NGSolve-6.2.2204
```

[2]:
```
from netgen . occ import *
from netgen . webgui import Draw as DrawGeo

# Geometry
# wir skalieren die Geometrie um die Rechenbarkeit zu gewaehren
scale = 0.075
```

```
geom = MoveTo (0 ,0)
geom = geom.LineTo(.2 ,0)
geom = geom.Spline([(0.85 ,0.7875) ,(1.25 ,1.8) ,(0.9 ,2.5) ,(0 ,2.85)])
geom = geom.Close().Face()
geom.faces.name = 'hotair'
geom.edges.name = 'outer'
geom.edges.Min(X).name = 'sym'
geom.edges.Min(Y).name = 'inlet'
geom = geom.Scale(gp_Pnt((0 ,0 ,0)), scale)
geo = OCCGeometry(geom, dim =2)

# Mesh
mesh = Mesh (geo.GenerateMesh(maxh = scale * 0.025))
mesh.Curve(3)# curved elements on the boundary
```

[2]:
```
<ngsolve.comp.Mesh at 0x7f62116c4860>
```

[3]:
```
############################################################
####### Finite element Methods - Space definition
############################################################
order = 2
W = H1(mesh , order=order , dirichlet ="inlet")
Vr = H1(mesh , order=order , dirichlet ="sym|outer|inlet")##orderinner=order+1,/
 ↪/0 auf symettrieachse auf aeusserem Rand und innen dran ist sie auch 0
Vz = H1(mesh , order =order , dirichlet ="outer|inlet")## orderinner=order+1,
Q = H1(mesh , order=order-1) #Lagrange parameter und stroemungfeld matchen ->␣
 ↪deswegen 1 ordnung tiefer
X = FESpace ([W,Vr ,Vz ,Q])#W = FE-Space Temperatur, Vr = Radiale Richtung fuer␣
 ↪Stroemung, Vz = Z-Richtung fuer Stroemung, Q = Fuer Druck (Fe-Space)
T, ur, uz,p = X.TrialFunction()
S, vr,vz,q = X.TestFunction()
r = x #x = Coeffient function from ngsolve
z = y #y = Coeffient function from ngsolve
############################################################
############################################################
############################################################
```

[4]:
```
T0_d = {"sym" : 0, "outer" : 0, "inlet" : 100}
T0 = CoefficientFunction([T0_d[b] for b in mesh.GetBoundaries()])
```

[5]:
```
############################################################
####### Material Parameters
############################################################
kappa = Parameter(0.0262)
scaleMu = Parameter(30)
mu = scaleMu * 18.2e-6 / np.sqrt(293.15) * sqrt(273.15 + T)
rho = 1013.25e-3 * 1e5 / (287.058 * (273.15 + T))
```

```
nu = mu / rho
g = 9.81
alpha = 1 / (273.15 + T)
gamma = Parameter(1e-3)
Tamb = Parameter(0)

#### For exercise 4
p0 = 1013.25e-3
############################################################
############################################################
############################################################
```

[6]:
```
### My definitions
gfx = GridFunction(X) #Produktraum -> Konkatenierung der Raeume
############################

############################################################
####### Grid function
############################################################
Temp = CoefficientFunction (gfx.components[0])
velocity = CoefficientFunction (gfx.components[1:3])
pressure = CoefficientFunction (gfx.components[3])
gfx.components[0].Set(100,definedon=mesh.Boundaries('inlet'))
############################################################
############################################################
############################################################
```

[7]:
```
############################################################
####### Implementation of exercise 2.1
############################################################
(a, u, div_u, div_v, conv) = stokesFlow(ur, r, vr, T, X, nu, alpha, uz, vz, q,␣
 ↪p, rho, g, gamma, Tamb, S, dx, kappa)
gfx.components[0].Set(T0,BND)
gfxVecData = NewtonSteps(a, gfx)
```

```
Iteration  0
Stopcritvalue: 4.420478670416403
Iteration  1
Stopcritvalue: 0.0041056078838375655
Iteration  2
Stopcritvalue: 1.2445977774604225e-05
Iteration  3
Stopcritvalue: 6.012965523273329e-09
Iteration  4
Stopcritvalue: 1.7849277634557295e-09
Iteration  5
Stopcritvalue: 2.8888032803068165e-09
```

```
Iteration  6
Stopcritvalue: 2.2510672586898335e-09
Iteration  7
Stopcritvalue: 1.7392375464506542e-09
Iteration  8
Stopcritvalue: 3.880593926838614e-09
Iteration  9
Stopcritvalue: 2.700109383074952e-09
```

[8]:
```
############################################################
####### Implementation of exercise 2.1
############################################################

###### Solution of stokesflow
Draw (Temp, mesh, 'T')
Draw ( velocity, mesh, 'v')
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

[8]:
```
BaseWebGuiScene
```

[9]:
```
############################################################
####### Implementation of exercise 2.1
############################################################

Draw(gfx.components[0])
velocity = CoefficientFunction(gfx.components[1:3])
Draw(velocity, mesh, "velocity")#Stroemungsfeld
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

[9]:
```
BaseWebGuiScene
```

[10]:
```
############################################################
####### Implementation of exercise 2.2
############################################################
#oderinnerOorder+1 (Diskretisierung an der liegt das) im H1 -> so kann auch␣
 ↪nicht stetig mit L2 gerechnet werden -> fuer die Stroemung 1 order erhoehen␣
 ↪dann konvergiert fuer 1
aNS = navierStokesFlow(ur, r, vr, T, X, nu, alpha, uz, vz, q, p, rho, g, gamma,␣
 ↪Tamb, S, dx, kappa, div_v, div_u)
```

```
gfx.components[0].Set(T0,BND)

# iterativ for Navier - Stokes flow
scaleMus = 1+ np. linspace (0 ,30**(1/2) ,30)[:: -1]**2
print("Das ist die Schrittweise Skalierung: " + str(scaleMus))
for s in scaleMus :
    print(s)
    scaleMu.Set(s)
    ret = NewtonSteps(aNS, gfx, nMax=25, tol=1e-8)
    if ret < 1:
        print('no convergence')
        break
```

```
Das ist die Schrittweise Skalierung: [31.         28.9667063  27.00475624
25.11414982 23.29488704 21.5469679
 19.87039239 18.26516052 16.73127229 15.26872771 13.87752675 12.55766944
 11.30915577 10.13198573  9.02615933  7.99167658  7.02853746  6.13674197
  5.31629013  4.56718193  3.88941736  3.28299643  2.74791914  2.28418549
  1.89179548  1.57074911  1.32104637  1.14268728  1.03567182  1.        ]
31.0
Iteration  0
Stopcritvalue: 4.4214976901022105
Iteration  1
Stopcritvalue: 0.0011086238296268691
Iteration  2
Stopcritvalue: 2.5110854599814182e-05
Iteration  3
Stopcritvalue: 1.22516147937401e-07
Iteration  4
Stopcritvalue: 9.852264169072667e-10
28.966706302021404
Iteration  0
Stopcritvalue: 5.4079296983125255e-05
Iteration  1
Stopcritvalue: 4.0050500046707186e-07
Iteration  2
Stopcritvalue: 1.338011796819819e-09
27.00475624256837
Iteration  0
Stopcritvalue: 5.765246154099228e-05
Iteration  1
Stopcritvalue: 5.155738467700775e-07
Iteration  2
Stopcritvalue: 2.139725026651298e-09
25.114149821640904
Iteration  0
Stopcritvalue: 6.152022514471869e-05
```

```
Iteration  1
Stopcritvalue: 6.693172247715492e-07
Iteration  2
Stopcritvalue: 1.2974232215664809e-09
23.294887039239
Iteration  0
Stopcritvalue: 6.568163663880807e-05
Iteration  1
Stopcritvalue: 8.69383607570998e-07
Iteration  2
Stopcritvalue: 9.434102899575584e-10
21.546967895362663
Iteration  0
Stopcritvalue: 7.014900744888384e-05
Iteration  1
Stopcritvalue: 1.1279005066964982e-06
Iteration  2
Stopcritvalue: 3.481650003040383e-10
19.87039239001189
Iteration  0
Stopcritvalue: 7.490490662273546e-05
Iteration  1
Stopcritvalue: 1.4545138574961794e-06
Iteration  2
Stopcritvalue: 7.315687117652817e-10
18.26516052318668
Iteration  0
Stopcritvalue: 7.993512501640081e-05
Iteration  1
Stopcritvalue: 1.8527429048349992e-06
Iteration  2
Stopcritvalue: 8.11043226848298e-10
16.73127229488704
Iteration  0
Stopcritvalue: 8.522358609380812e-05
Iteration  1
Stopcritvalue: 2.313852417381104e-06
Iteration  2
Stopcritvalue: 1.430058143775387e-09
15.268727705112962
Iteration  0
Stopcritvalue: 9.076337857824874e-05
Iteration  1
Stopcritvalue: 2.811519094357667e-06
Iteration  2
Stopcritvalue: 4.206691077168533e-09
13.877526753864448
Iteration  0
```

```
Stopcritvalue: 9.656277086183918e-05
Iteration  1
Stopcritvalue: 3.302243275091296e-06
Iteration  2
Stopcritvalue: 3.858706069855737e-09
12.557669441141499
Iteration  0
Stopcritvalue: 0.00010265751226394015
Iteration  1
Stopcritvalue: 3.7358018592681892e-06
Iteration  2
Stopcritvalue: 1.1892700826547392e-08
Iteration  3
Stopcritvalue: 2.4939591517989686e-09
11.309155766944114
Iteration  0
Stopcritvalue: 0.00010912038168210932
Iteration  1
Stopcritvalue: 4.095187062774253e-06
Iteration  2
Stopcritvalue: 7.439166700530901e-09
10.131985731272295
Iteration  0
Stopcritvalue: 0.0001160111289392172
Iteration  1
Stopcritvalue: 4.401705870893827e-06
Iteration  2
Stopcritvalue: 1.18963285132114e-08
Iteration  3
Stopcritvalue: 3.711841946949208e-09
9.02615933412604
Iteration  0
Stopcritvalue: 0.00012349930348844 88
Iteration  1
Stopcritvalue: 4.764360032911093e-06
Iteration  2
Stopcritvalue: 6.295705428858056e-08
Iteration  3
Stopcritvalue: 5.7685908878466295e-09
7.991676575505351
Iteration  0
Stopcritvalue: 0.0001316215621680035
Iteration  1
Stopcritvalue: 5.276163610024758e-06
Iteration  2
Stopcritvalue: 9.747057605502238e-08
Iteration  3
Stopcritvalue: 8.56083791294692e-09
```

```
7.028537455410226
Iteration  0
Stopcritvalue: 0.0001406460123479317
Iteration  1
Stopcritvalue: 6.091048838193541e-06
Iteration  2
Stopcritvalue: 5.70844925443373e-08
Iteration  3
Stopcritvalue: 1.0420722214083419e-08
Iteration  4
Stopcritvalue: 4.24881905347862e-09
6.136741973840666
Iteration  0
Stopcritvalue: 0.00015072843284234314
Iteration  1
Stopcritvalue: 7.155822706743699e-06
Iteration  2
Stopcritvalue: 2.6570581312006276e-08
Iteration  3
Stopcritvalue: 2.1081952981308823e-08
Iteration  4
Stopcritvalue: 8.430675727059361e-09
5.31629013079667
Iteration  0
Stopcritvalue: 0.00016200682312438315
Iteration  1
Stopcritvalue: 8.367130401860362e-06
Iteration  2
Stopcritvalue: 2.9082746432919606e-08
Iteration  3
Stopcritvalue: 2.0458741179313303e-08
Iteration  4
Stopcritvalue: 2.6297964170031766e-08
Iteration  5
Stopcritvalue: 8.909520839821366e-09
4.5671819262782405
Iteration  0
Stopcritvalue: 0.00017449794582630895
Iteration  1
Stopcritvalue: 9.727701909183523e-06
Iteration  2
Stopcritvalue: 1.581663293922209e-07
Iteration  3
Stopcritvalue: 1.394240855447834e-07
Iteration  4
Stopcritvalue: 6.192952140181971e-08
Iteration  5
Stopcritvalue: 3.5333410416113204e-07
```

```
Iteration  6
Stopcritvalue: 6.398276437392989e-07
Iteration  7
Stopcritvalue: 1.5891998398674636e-07
Iteration  8
Stopcritvalue: 9.071276305141814e-08
Iteration  9
Stopcritvalue: 5.216456942800537e-08
Iteration  10
Stopcritvalue: 2.073322045970722e-08
Iteration  11
Stopcritvalue: 3.078335255865036e-08
Iteration  12
Stopcritvalue: 5.947276661050992e-09
3.8894173602853748
Iteration  0
Stopcritvalue: 0.00018796105362798736
Iteration  1
Stopcritvalue: 1.142690635988205e-05
Iteration  2
Stopcritvalue: 5.487393568628898e-08
Iteration  3
Stopcritvalue: 3.760640798269e-09
3.2829964328180736
Iteration  0
Stopcritvalue: 0.00020161176494497374
Iteration  1
Stopcritvalue: 1.364744322429451e-05
Iteration  2
Stopcritvalue: 4.320037702196251e-08
Iteration  3
Stopcritvalue: 8.626980611308641e-09
2.747919143876338
Iteration  0
Stopcritvalue: 0.00021412877009996747
Iteration  1
Stopcritvalue: 1.582071136572986e-05
Iteration  2
Stopcritvalue: 1.4780657036978089e-07
Iteration  3
Stopcritvalue: 2.5248545644206005e-08
Iteration  4
Stopcritvalue: 5.379117607312979e-07
Iteration  5
Stopcritvalue: 2.2547628735522175e-05
Iteration  6
Stopcritvalue: 4.870005169906122e-07
Iteration  7
```

```
Stopcritvalue: 1.832818466486345e-07
Iteration  8
Stopcritvalue: 2.0193537351912737e-07
Iteration  9
Stopcritvalue: 5.1633494153164946e-08
Iteration  10
Stopcritvalue: 2.7839585598792713e-08
Iteration  11
Stopcritvalue: 1.6107133662439394e-07
Iteration  12
Stopcritvalue: 4.5590017062094914e-07
Iteration  13
Stopcritvalue: 2.7799109725452007e-08
Iteration  14
Stopcritvalue: 3.5676075317619854e-08
Iteration  15
Stopcritvalue: 3.9217264151747797e-08
Iteration  16
Stopcritvalue: 4.0260701403779964e-08
Iteration  17
Stopcritvalue: 4.255702649840839e-08
Iteration  18
Stopcritvalue: 9.70369554770416e-08
Iteration  19
Stopcritvalue: 2.1137606871458151e-07
Iteration  20
Stopcritvalue: 2.9854831007540725e-08
Iteration  21
Stopcritvalue: 1.0094283453499957e-07
Iteration  22
Stopcritvalue: 1.8743233975615578e-07
Iteration  23
Stopcritvalue: 4.982296661173144e-08
Iteration  24
Stopcritvalue: 6.842318058672986e-08
no convergence
```

[11]:
```
############################################################
####### Plot of exercise 2.2
############################################################
Draw(gfx.components[0])
velocity = CoefficientFunction(gfx.components[1:3])
Draw(velocity, mesh, "velocity")#Stroemungsfeld
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

[11]:
```
BaseWebGuiScene
```

[12]:
```
############################################################
####### Definitions of exercise 3
############################################################
T, ux, uy,p = X.TrialFunction()
S, vx,vy,q = X.TestFunction()

def stokesFlow2d(ux, vx, T, X, nu, alpha, uy, vy, q, p, rho, g, gamma, Tamb, S,␣
 ↪dx, kappa):
    # Stokes flow
    div_u = grad(ux)[0]+ grad(uy)[1]
    div_v = grad(vx)[0]+ grad(vy)[1]

    u = CoefficientFunction ((ux, uy))#For vectorial usage of ur and uz
    conv = u* grad(T)

    a = BilinearForm(X, symmetric = False)
    a += (kappa * grad(T) * grad(S)+ conv * S) * dx
    a += gamma *(T- Tamb)*S*ds('outer')
    a += (nu* grad(ux)* grad(vx) + nu * grad(uy) * grad(vy) + 1/ rho * p *␣
 ↪div_v + g * alpha * grad(T)[1] * vy)*dx #bis zu dieser Zeile mit Dozent␣
 ↪besprochen
    a += div_u * q * dx
    return a, u, div_u, div_v, conv

def navierStokesFlow2d(ux, vx, T, X, nu, alpha, uy, vy, q, p, rho, g, gamma,␣
 ↪Tamb, S, dx, kappa, div_v, div_u):
    #Navier-Stokesflow
    aNS = BilinearForm(X, symmetric=False) #ebensproblem in x, y -> r␣
 ↪rausbringen -> z wirds y und r ist das x koordinate
    aNS += (kappa * grad(T) * grad(S) + conv * S) * dx
    aNS += gamma * (T - Tamb) * S * ds('outer')
    aNS += (nu * grad(ux) * grad(vx) + nu * grad(uy) * grad(vy) + u * (grad(ux)␣
 ↪* vx + grad(uy) * vy) + 1 / rho * p * div_v + g * alpha * grad(T)[1] * vy) *␣
 ↪dx
    aNS += div_u * q * dx
    return aNS
#Rotationssymettrische (axisymetric solution in English) loesung heisst in␣
 ↪jedem Winkel phi ist die Loesung genau gleich -> Vorteil nur ein 2D Problem␣
 ↪loesen kriegen aber die Loesung fuer ein 3D Problem -> FE kann man das␣
 ↪machen aber mit finiten Volumen nicht
```

[13]:
```
############################################################
####### Implementation of exercise 3 - Stokes-Flow
############################################################
(a, u, div_u, div_v, conv) = stokesFlow2d(ux, vx, T, X, nu, alpha, uy, vy, q,␣
 ↪p, rho, g, gamma, Tamb, S, dx, kappa)
gfu = GridFunction(X)
gfu.components[0].Set(T0,BND)
gfuVecData = NewtonSteps(a, gfu)
velocity = CoefficientFunction(gfx.components[1:3])
Draw(velocity, mesh, "velocity")#Stroemungsfeld
```

```
Iteration  0
Stopcritvalue: 49.52082977450688
Iteration  1
Stopcritvalue: 0.24413504797120011
Iteration  2
Stopcritvalue: 0.00264389135846286
Iteration  3
Stopcritvalue: 3.6389119679334467e-06
Iteration  4
Stopcritvalue: 2.3725853418445997e-11
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

[13]:
```
BaseWebGuiScene
```

[14]:
```
############################################################
####### Implementation exercise 3 - Navier-Stokes-Flow
############################################################
#oderinnerOorder+1 (Diskretisierung an der liegt das) im H1 -> so kann auch␣
 ↪nicht stetig mit L2 gerechnet werden -> fuer die Stroemung 1 order erhoehen␣
 ↪dann konvergiert fuer 1
aNS = navierStokesFlow2d(ux, vx, T, X, nu, alpha, uy, vy, q, p, rho, g, gamma,␣
 ↪Tamb, S, dx, kappa, div_v, div_u)
gfk = GridFunction(X)
gfk.components[0].Set(T0,BND)

# iterativ for Navier - Stokes flow
scaleMus = 5+ np. linspace (0 ,30**(1/2) ,30)[:: -1]**2
print("Das ist die Schrittweise Skalierung: " + str(scaleMus))
for s in scaleMus :
    print(s)
    scaleMu.Set(s)
    ret = NewtonSteps(aNS, gfk, nMax=25, tol=1e-8)
    if ret < 1:
        print('no convergence')
```

```
        break
```

```
Das ist die Schrittweise Skalierung: [35.         32.9667063  31.00475624
29.11414982 27.29488704 25.5469679
 23.87039239 22.26516052 20.73127229 19.26872771 17.87752675 16.55766944
 15.30915577 14.13198573 13.02615933 11.99167658 11.02853746 10.13674197
  9.31629013  8.56718193  7.88941736  7.28299643  6.74791914  6.28418549
  5.89179548  5.57074911  5.32104637  5.14268728  5.03567182  5.        ]
35.0
Iteration  0
Stopcritvalue: 49.621616868615064
Iteration  1
Stopcritvalue: 0.019785811537210162
Iteration  2
Stopcritvalue: 3.925351835445179e-05
Iteration  3
Stopcritvalue: 1.7549801657267164e-08
Iteration  4
Stopcritvalue: 2.1044467185675377e-11
32.96670630202141
Iteration  0
Stopcritvalue: 0.00012267426321214015
Iteration  1
Stopcritvalue: 1.1806636704782536e-07
Iteration  2
Stopcritvalue: 2.618350695808991e-11
31.00475624256837
Iteration  0
Stopcritvalue: 0.00012962542287915863
Iteration  1
Stopcritvalue: 1.4517185596317234e-07
Iteration  2
Stopcritvalue: 5.0723954101789735e-12
29.114149821640904
Iteration  0
Stopcritvalue: 0.0001371025551809839
Iteration  1
Stopcritvalue: 1.7954498865175577e-07
Iteration  2
Stopcritvalue: 5.595147858995814e-12
27.294887039239
Iteration  0
Stopcritvalue: 0.00014514582530839932
Iteration  1
Stopcritvalue: 2.234125421064547e-07
Iteration  2
Stopcritvalue: 1.0706877250252582e-11
```

```
25.546967895362663
Iteration  0
Stopcritvalue: 0.00015379555060363246
Iteration  1
Stopcritvalue: 2.797698937529638e-07
Iteration  2
Stopcritvalue: 2.863369758859233e-11
23.87039239001189
Iteration  0
Stopcritvalue: 0.00016309051473931968
Iteration  1
Stopcritvalue: 3.5264806712316987e-07
Iteration  2
Stopcritvalue: 1.556645269530207e-11
22.26516052318668
Iteration  0
Stopcritvalue: 0.0001730653792463333
Iteration  1
Stopcritvalue: 4.4750845683719056e-07
Iteration  2
Stopcritvalue: 2.606603254300257e-11
20.73127229488704
Iteration  0
Stopcritvalue: 0.0001837467529628204
Iteration  1
Stopcritvalue: 5.71736548199072e-07
Iteration  2
Stopcritvalue: 1.0786886722353422e-11
19.268727705112962
Iteration  0
Stopcritvalue: 0.0001951473088133804
Iteration  1
Stopcritvalue: 7.352144364109539e-07
Iteration  2
Stopcritvalue: 1.901725744094125e-11
17.877526753864448
Iteration  0
Stopcritvalue: 0.0002072571777696387
Iteration  1
Stopcritvalue: 9.510901167934373e-07
Iteration  2
Stopcritvalue: 3.3576457387434886e-11
16.5576694411415
Iteration  0
Stopcritvalue: 0.00022003148002833737
Iteration  1
Stopcritvalue: 1.236112881837237e-06
Iteration  2
```

```
Stopcritvalue: 5.964633541246876e-11
15.309155766944114
Iteration  0
Stopcritvalue: 0.00023337287034307525
Iteration  1
Stopcritvalue: 1.6103335909118796e-06
Iteration  2
Stopcritvalue: 1.0299525824321506e-10
14.131985731272295
Iteration  0
Stopcritvalue: 0.000247107816579445
Iteration  1
Stopcritvalue: 2.094479372010623e-06
Iteration  2
Stopcritvalue: 1.6586091707937748e-10
13.02615933412604
Iteration  0
Stopcritvalue: 0.00026095837696169243
Iteration  1
Stopcritvalue: 2.7027684848673497e-06
Iteration  2
Stopcritvalue: 2.4396839396460936e-10
11.991676575505352
Iteration  0
Stopcritvalue: 0.00027451163720333406
Iteration  1
Stopcritvalue: 3.4276813097930643e-06
Iteration  2
Stopcritvalue: 3.534742108910583e-10
11.028537455410227
Iteration  0
Stopcritvalue: 0.00028720163674010 67
Iteration  1
Stopcritvalue: 4.216809814350232e-06
Iteration  2
Stopcritvalue: 6.498720155172861e-10
10.136741973840666
Iteration  0
Stopcritvalue: 0.00029831915522196 04
Iteration  1
Stopcritvalue: 4.951511823223569e-06
Iteration  2
Stopcritvalue: 1.401998027712904e-09
9.31629013079667
Iteration  0
Stopcritvalue: 0.00030706417716478215
Iteration  1
Stopcritvalue: 5.456058224208502e-06
```

```
Iteration  2
Stopcritvalue: 2.5259162015761297e-09
8.567181926278241
Iteration  0
Stopcritvalue: 0.00031262259473686086
Iteration  1
Stopcritvalue: 5.567591917032233e-06
Iteration  2
Stopcritvalue: 3.4353641637956638e-09
7.889417360285375
Iteration  0
Stopcritvalue: 0.00031421004163564593
Iteration  1
Stopcritvalue: 5.2482598860611665e-06
Iteration  2
Stopcritvalue: 3.4816813540440893e-09
7.282996432818074
Iteration  0
Stopcritvalue: 0.0003110359258497745
Iteration  1
Stopcritvalue: 4.645509418376292e-06
Iteration  2
Stopcritvalue: 2.709824181696201e-09
6.747919143876338
Iteration  0
Stopcritvalue: 0.00030222329041325015
Iteration  1
Stopcritvalue: 4.00229860254447e-06
Iteration  2
Stopcritvalue: 1.7701915313987592e-09
6.284185493460166
Iteration  0
Stopcritvalue: 0.0002867912827398901
Iteration  1
Stopcritvalue: 3.4518624979988362e-06
Iteration  2
Stopcritvalue: 1.1265003984948184e-09
5.89179548156956
Iteration  0
Stopcritvalue: 0.0002637712332832199
Iteration  1
Stopcritvalue: 2.9297854998461545e-06
Iteration  2
Stopcritvalue: 7.807001273855242e-10
5.570749108204518
Iteration  0
Stopcritvalue: 0.00023243382209792615
Iteration  1
```

```
Stopcritvalue: 2.324592855456017e-06
Iteration  2
Stopcritvalue: 5.202151864442961e-10
5.321046373365042
Iteration  0
Stopcritvalue: 0.00019254793215707722
Iteration  1
Stopcritvalue: 1.626077965778223e-06
Iteration  2
Stopcritvalue: 2.6784317906441305e-10
5.142687277051129
Iteration  0
Stopcritvalue: 0.0001445969560097406
Iteration  1
Stopcritvalue: 9.263889985714972e-07
Iteration  2
Stopcritvalue: 8.18825757245744e-11
5.035671819262783
Iteration  0
Stopcritvalue: 8.989589333994947e-05
Iteration  1
Stopcritvalue: 3.5856212425589657e-07
Iteration  2
Stopcritvalue: 2.295978591702246e-11
5.0
Iteration  0
Stopcritvalue: 3.056436145317217e-05
Iteration  1
Stopcritvalue: 4.119833604214128e-08
Iteration  2
Stopcritvalue: 4.1309045177851944e-11
```

[15]:
```
############################################################
####### Plot exercise 3 - Navier-Stokes-Flow
############################################################
velocity = CoefficientFunction(gfk.components[1:3])
Draw(velocity, mesh, "velocity")#Stroemungsfeld
```

```
WebGuiWidget(value={'gui_settings': {}, 'ngsolve_version': '6.2.2204',␣
 ↪'mesh_dim': 2, 'order2d': 2, 'order3d':…
```

[15]:
```
BaseWebGuiScene
```

[16]:
```
############################################################
####### First bullet point - exercise 4
############################################################
rhoT = 1013.25e-3*1e5/(287.058*(273.15+Temp)) #Temperaturabhaengige Funktion␣
 ↪der Dichte
```

```
#10^5 -> heisst bar in pascal umrechnen ; rhoT[kg/m^3]
rho0 = 1013.25e-3*1e5/(287.058*273.15)

massHotAir = 2*np.pi*Integrate(rhoT*r, mesh) #Masse bei warmer Luft
massColdAir = 2*np.pi*Integrate(rho0*r, mesh) #Masse bei kalte Luft

FAuftrieb=(massColdAir-massHotAir)*g
print(g)
print(massHotAir)
print(massColdAir)
print("Die Auftriebskraft lautet: " + str(FAuftrieb) + " N")
```

```
9.81
0.003334695866043888
0.004436405617570045
Die Auftriebskraft lautet: 0.010807772662471597 N
```

[17]:
```
############################################################
####### Second bullet point - exercise 4
############################################################
#FAuftrieb*hoeheGesucht/hoeheOrig = FG = (200kg+masseHotAir)*g
masse = 200
hOrig = 0.21375 #Original Hoehe -> 2.85(Original-Hoehe * 0.
 ↪075(Skalierungsfaktor))
alpha = (masse / (massColdAir-massHotAir))**(1/3) #masse die wir heben moechten␣
 ↪durch die masse die wir effektiv heben
print("Der Skalierungsfaktor lautet: " + str(alpha))
hNeu = hOrig*alpha
print("Die neue Höhe lautet: " + str(hNeu) + " Meter")
```

```
Der Skalierungsfaktor lautet: 56.622312291184095
Die neue Höhe lautet: 12.1030192522406 Meter
```