



Masterarbeit

Querschnittswerte von polygonalen dünnwandigen Querschnitten mit 3D- Grafik der Wölbfunktion – Realisierung in Python

zusätzliche Aufgabe:

Grafische Programmerweiterungen im C#-Code für Steifigkeits-
und Massenwerte von Windturbinen-Rotorblättern

Masterstudiengang Maschinenbau

eingereicht am 22.11.2022 – Termin der tatsächlichen Abgabe

von

Heinrich Vent

Matrikel-Nr.: 219 100 105

Bearbeitungszeitraum: 20 Wochen

Gutachter:

Prof. Dr. rer. nat. habil. Uwe Ritschel

Lehrstuhl für Windenergietechnik

Justus-von-Liebig-Weg 2

18059 Rostock

Zweitgutachter:

PD Dr.-Ing. habil. Evgeni Stanoev

Lehrstuhl für Windenergietechnik

Justus-von-Liebig-Weg 2

18059 Rostock

Inhaltsverzeichnis

1. Einleitung.....	20
2. Kenntnisstand.....	23
2.1 Querschnittswerte für Zug/Druck, Biegung und Schub.....	24
2.1.1 Aufbau des Modells	24
2.1.2 Grundlegende Flächenmomente	27
2.1.3 Hauptträgheitsmomente und -achsen	29
2.1.4 Schubflächen	31
2.2 Saint-Venantsche Torsion am dünnwandigen Querschnitt.....	33
2.2.1 Bedingungen und Ziele der Saint-Venantschen Torsion	33
2.2.2 Erweiterung des Modells	35
2.2.3 Dünnwandiger geschlossener einzelliger Querschnitt	37
2.2.4 Dünnwandiger geschlossener mehrzelliger Querschnitt	41
2.2.5 Dünnwandige offene Querschnittsteile	45
2.3 Wölbkrafttorsion am dünnwandigen Querschnitt.....	46
2.3.1 Bedingungen und Ziele der Wölbkrafttorsion	46
2.3.2 FE-Lösung der Wölbfunktion	49
2.3.3 Wölb-Flächenmomente	54
2.3.4 Schubmittelpunkt	55
2.3.5 Einheitsverwölbung und Wölbträgheitsmoment	58
2.3.6 Saint-Venantsche Torsion mit Wölbkrafttorsion	59
3. Material und Methoden.....	60
3.1 Matlab-Programm „QUEBER_Version2016“.....	61
3.1.1 Übersicht Programmbestandteile	61
3.1.2 Eingabedatei	62
3.1.3 Programmausgaben	63
3.2 C#-Programm „Thesis_Interface“.....	65
3.2.1 Vergleich mit „QUEBER_Version2016“	65
3.2.2 2D-Grafik einer Wölbfunktion in C#	66
3.3 Aufbau neues Programm.....	68
3.3.1 Verwendete Python-Installation und IDE	68
3.3.2 Funktionales Design	68
3.3.3 Data Design	70
3.3.4 Konzepterstellung	74
4. Ergebnisse.....	75

4.1 Programmbeschreibung.....	75
4.2 Grafiken.....	78
4.3 Ergebnisvalidierung.....	82
5. Diskussion.....	92
6. Zusammenfassung und Ausblick.....	94
7. Literaturverzeichnis.....	95
7.1 Bücher.....	95
7.2 Universitätsschriften.....	96
7.3 Internet-Quellen.....	97
8. Anhang.....	102
8.1 Grundlagen und Zielsetzungen.....	103
8.1.1 Grundprinzipien der Finite-Elemente-Methode	103
8.1.2 Grundprinzipien der Elastostatik	107
8.2 Relevante Aspekte von Python.....	115
8.2.1 Grundlagen der Programmierung	115
8.2.2 Einleitung zu Python	120
8.2.3 Operatoren, Funktionen und Anweisungen in Python	123
8.2.4 Datentypen in Python	131
8.2.5 Objektorientierte Programmierung in Python	140
8.2.6 Numerik mit Package NumPy	143
8.2.7 Datenvisualisierung mit Package Matplotlib	147
8.2.8 Graphical User Interface mit Package PYSimpleGUI	150
8.2.9 Dokumentation mit Package MkDocs	152
8.2.10 PEP 8 - Styleguide für Python	155
8.2.11 Grundlagen des Software-Designs	162
8.2.12 Debugging, Errors und Exceptions in Python	174
8.3 Anhänge zu Material und Methoden.....	180
8.3.1 Berechnungsablauf „QUEBER_Version2016“	180
8.3.2 Ausgabedatei von „QUEBER_Version2016“	191
8.3.3 Quellcode der Programmerweiterung in C#	195
9. Eidesstattliche Versicherung.....	199

Abbildungsverzeichnis

Abbildung 1: Schematische Darstellung der geometrischen Größen zur Definition des i -ten Elements des Querschnitts.....	24
Abbildung 2: Darstellung von Ursprungs-KS (x, y, z) und FSP-KS (x, y, z) für einen beliebig geformten Querschnitt, mit beispielhaft eingetragener infinitesimaler Teilfläche dA (Hofstetter 2013), S. 160.....	25
Abbildung 3: Beispiel der Einteilung in Elemente bei einem Z-Profil, zulässige Ungenauigkeit in der Modellierung der Ecken bzw. Elementübergänge (rechts) (Linke 2015), S. 76.....	26
Abbildung 4: Qualitative Darstellung von FSP-KS, HTA-KS und HTA-Winkel $\varphi_{1,2}$	29
Abbildung 5: Einzelliger Querschnitt mit Polstrahl $r(s)$ KS (x, y, z) im SMP M (Linke 2015), S. 100.....	33
Abbildung 6: Beispiele für wölbfreie dünnwandige geschlossene Querschnitte mit einheitlichem Polstrahl r_\perp (Linke 2015), S. 126.....	34
Abbildung 7: Beispiele für wölbfreie dünnwandige offene Querschnitte mit SMP im Schnittpunkt aller Profilmittellinien (Linke 2015), S. 132.....	34
Abbildung 8: Belastungen am Balken mit Bezug zu FSP und SMP im betrachteten Querschnitt am Beispiel eines L-Profiles (Linke 2015), S. 91.....	35
Abbildung 9: Beispiel-Querschnitt aus Rechteck-Elementen mit einzellig geschlossenen Profilteilen und einem offenen Profilteil (Stanoev 2016).....	36
Abbildung 10: Zusammensetzung der Schubspannung im Querschnitt an der Stelle s (Stanoev 2013), S. 10.....	37
Abbildung 11: Infinitesimales Elemen der Umfangslänge ds mit geometrischen Größen (bezogen auf SMP M) und Kraft dF (Linke 2015), S. 100, bearb.....	38
Abbildung 12: Dünnwandiger geschlossener zweizelliger Querschnitt mit Torsionsmomenten und Schubflüssen der Zellen (Linke 2015), S. 106.....	41
Abbildung 13: Dünnwandiger geschlossener zweizelliger Querschnitt mit Verdrehung um SMP (Linke 2015), S. 108.....	42
Abbildung 14: Bezogene Schubflüsse in den Zellen eines dünnwandigen geschlossenen mehrzelligen Querschnitts mit gedachten Schnitten (Stanoev 2013), S. 17.....	43
Abbildung 15: Entkoppelte Schnittgrößen im HTA-KS eines verwölbten Querschnitts; Beispiel eines C-Trägers mit Wirkungslinien von Kräften und Momenten in M bzw. S (Mittelstedt 2021), S. 308-309, bearb.....	46

Abbildung 16: Schematische Darstellung von Gabellagerung eines tordierten Stabes (Stanoev 2013), S. 22.....	47
Abbildung 17: Verwölbung und Wölbnormalspannungen am Beispiel eines einseitig eingespannten I-Trägers mit Torsionsbelastung (Mittelstedt 2021), S. 290.....	47
Abbildung 18: Primäre und sekundäre Schubspannungen am Beispiel eines einseitig eingespannten I-Trägers mit Torsionsbelastung (Mittelstedt 2021), S. 291.....	48
Abbildung 19: Wölbnormalspannungen durch Flanschbiegemomente am Beispiel eines einseitig eingespannten I-Trägers mit Torsionsbelastung (Mittelstedt 2021), S. 291.....	48
Abbildung 20: Geometrische Größen am Element eines FE-Modells zur Bestimmung der Wölbfunktion (Stanoev 2013), S. 24, bearb.....	49
Abbildung 21: Geometrische Größen zur SMP-Bestimmung eines beliebig geformten dünnwandigen Querschnitts (Linke 2015), S. 163, bearb.....	56
Abbildung 22: Geometrische Größen zur Umrechnung zwischen Polstrahlen r_{ID} und r_{TM} (Mittelstedt 2021), S. 241.....	57
Abbildung 23: 2D-Grafik mit Knoten und Elementen sowie Wölbfunktion bezogen auf M in „QUEBER_Version2016“, (Stanoev 2016) / (Wunderlich k.A.).....	63
Abbildung 24: 3D-Grafik mit Knoten und Elementen sowie Wölbfunktion bezogen auf M in „QUEBER_Version2016“, (Stanoev 2016) / (Wunderlich k.A.).....	64
Abbildung 25: 2D-Darstellung eines WEA-Rotorblatt-Querschnitts „bladeod200.txt“ samt Wölbfunktion im C#-Programm „Thesis_Interface“ (Nan k.A.).....	67
Abbildung 26: Eingangsdaten, FSP und Flächenmomente, Dehn-, Biege- und Schubsteifigkeiten im DataDesign Python-Programm,vgl.(Stanoev 2016).....	70
Abbildung 27: Wölbfunktion, bez. auf O , Torsionsträgheitsmoment und -steifigkeit im Data Design des Python-Programms, vgl. (Stanoev 2016).....	71
Abbildung 28: Wölbfunktion, bez. auf S und M , SMP und Wölbträgheitsmoment im Data Design des Python-Programms, vgl. (Stanoev 2016).....	72
Abbildung 29: Abstrahierte Darstellung von Datenpaketen im Data Design des Python-Programms, vgl. (Stanoev 2016).....	73
Abbildung 30: Klassendiagramm eines Konzepts mit OOP für neues Programm, vgl. (Tech with Tim 2020) / (Werkle 2008), S. 555.....	74
Abbildung 31: Eingabefenster für das Python-Programm „Thin-Walled Cross Section Properties“.....	75
Abbildung 32: Functional Design Teil 1 für das Python-Programm „Thin-Walled Cross Section Properties“.....	76

Abbildung 33: Functional Design Teil 2 für das Python-Programm „Thin-Walled Cross Section Properties“.....	77
Abbildung 34: 2D-Grafik mit Wölbfunction OM für „Profil-Wunderlich.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Wunderlich k.A.).....	78
Abbildung 35: 3D-Grafik mit Wölbfunction OM für „Profil-Wunderlich.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Wunderlich k.A.).....	79
Abbildung 36: 2D-Grafik mit Wölbfunction OMS für „3kasten_Spkt.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Stanoev k.A. a).....	80
Abbildung 37: 3D-Grafik mit Wölbfunction OMM für „3kasten_Spkt.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Stanoev k.A. a).....	81
Abbildung 38: Fehlermeldung und Quellcode für Skalierung von Achsen in 2D- und 3D-Grafiken des Python-Programms.....	92
Abbildung 39: Arbeitsschritte zur Verwendung eines FEM-Programms im Entwicklungsprozess eines Tragwerks (Selke 2013), S. 236.....	105
Abbildung 40: Beidseitig gelagerter Biegebalken nach Theorie 1. , 2. und 3. Ordnung (Spura 2019), S. 2, bearb.....	107
Abbildung 41: Kraft- und Verformungsgrößen der Elastostatik mit Umrechnungsmöglichkeiten (Spura 2019), S. 3.....	108
Abbildung 42: Schnittgrößen am Teilstück dx eines querkraftbelasteten Balkens (Spura 2019), S. 7.....	109
Abbildung 43: Verformung eines Balkens infolge von Biegebelastung (Euler-Bernoulli-Balken) (Spura 2019), S. 10.....	110
Abbildung 44: Gegenüberstellung von (a) realer Schubspannungsverteilung und Verwölbung und (b) Annahme des Ebenbleibens des Querschnitts ohne Verwölbung mit mittlerer Schubspannung (Spura 2019), S. 11.....	111
Abbildung 45: Kombination von (a) Euler-Bernoulli-Balken unter Biegung mit (b) Schubbelastung zu (c) Timoshenko-Balken (Spura 2019), S. 12.....	112
Abbildung 46: Unbehinderte Verwölbung im Gabellager und Verwölbung bei Wölbbehinderung am Beispiel eines I-Trägers (Nasdala 2015), S. 153.....	113
Abbildung 47: Beispiel für einen Mehrfeldbalken mit Unterbrechungen der Stetigkeit durch Gelenk, Zwischenlager und Einzelkraft (Mittelstedt 2021), S. 200	114
Abbildung 48: Vergleich Laufzeit über Entwicklungsaufwand für höhere Programmiersprache (Python) und niedere Programmiersprache (C), (Natt 2020), S. 3.....	116
Abbildung 49: Schematischer Aufbau von for- und while-Schleife, (Woyand 2019), S. 73, S. 77.....	118

Abbildung 50: Platz 1-19 des TIOBE Index für August 2022 mit Veränderung zum August 2021 (TIOBE 2022).....	121
Abbildung 51: Arithmetische Operatoren in Python 3 (Woyand 2019), S. 45....	123
Abbildung 52: Wichtige mathematische Funktionen aus Modul <i>math</i> (Woyand 2019), S. 47.....	123
Abbildung 53: Reservierte Schlüsselwörter in Python (Woyand 2019), S. 50...124	124
Abbildung 54: Kombinierte Zuweisungsoperatoren in Python (Woyand 2019), S. 54.....	124
Abbildung 55: Beispiefunktion in Python mit erklärendem Kommentar, Berechnungsanweisung und Rückgabe des berechneten Wertes (Woyand 2019), S. 55.....	126
Abbildung 56: Berechnung der Fakultät eines Parameters (<i>n</i>) mithilfe von Rekursion in Python (Woyand 2019), S. 115.....	126
Abbildung 57: Auswahl wichtiger Formatelemente in Python (Woyand 2019), S. 64.....	127
Abbildung 58: Beispiele für die Verwendung von Formatelementen in der <i>print()</i> -Funktion von Python (Woyand 2019), S. 65.....	127
Abbildung 59: Beispiel zum Lesen einer Textdatei mit <i>open()</i> und <i>.readlines()</i> in Python (Woyand 2019), S. 120.....	128
Abbildung 60: Beispiel zum Schreiben einer Texdatei mit <i>open()</i> und <i>.write()</i> (Woyand 2019), S. 123.....	128
Abbildung 61: Schemata von Programmverzweigungen in Python mit <i>if</i> -, <i>else</i> - und <i>elif</i> -Anweisungen (Woyand 2019), S. 67-69.....	129
Abbildung 62: Beispiel für <i>if</i> -Anweisung mit <i>elif</i> - und <i>else</i> -Klausel in Python (Woyand 2019), S. 68.....	129
Abbildung 63: Vergleichsoperatoren in Python mit Beispiel und Ergebnis (Woyand 2019), S. 71.....	129
Abbildung 64: Gegenüberstellung der Operatoren <i>is</i> und <i>==</i> in Python (Schäfer 2019), S. 15.....	129
Abbildung 65: Logische Operatoren in Python mit Beispiel und Ergebnis (Woyand 2019), S. 71.....	130
Abbildung 66: <i>for</i> -Schleife in Python mit Beispielen, u.a. Verwendung der <i>range()</i> -Funktion (Woyand 2019), S. 72-74, bearb.....	130
Abbildung 67: <i>while</i> -Schleife mit Beispiel in Python (Woyand 2019), S. 77.....	130
Abbildung 68: Eigenschaften numerischer Datentypen in Python (Schäfer 2019), S. 17.....	131
Abbildung 69: Verschiedene (Exponential)Schreibweisen einer Gleitpunktzahl in Python (Woyand 2019), S. 33.....	131

Abbildung 70: Syntax zur Beschreibung komplexer Zahlen in Python (Woyand 2019), S. 34.....	132
Abbildung 71: Beispiel für Verwendung von String-Methode <code>string.split()</code> in Python (Woyand 2019), S. 40.....	133
Abbildung 72: Operationen an sequentiellen Datentypen (Listen, Tuples, Strings) in Python (Schäfer 2019), S. 17.....	133
Abbildung 73: Beispiele für Listen und Verwendung in <i>if</i> -Anweisung und <i>for</i> -Schleife mit <i>not in-</i> und <i>in</i> -Operator in Python (Woyand 2019), S. 85-86.....	134
Abbildung 74: Beispiele zur Syntax von Tuples in Python, mit und ohne runde Klammern (Woyand 2019), S. 95.....	135
Abbildung 75: Mengenoperationen an zwei Sets (Mengen) in Python (Woyand 2019), S. 97.....	136
Abbildung 76: Beispiel von Sets (Mengen) und Mengenoperationen in Python (Woyand 2019), S. 97.....	136
Abbildung 77: Beispiel für Schlüsselwert – Datenwert – Paare eines Dictionaries in Python (Woyand 2019), S. 98.....	137
Abbildung 78: Beispiele für geordnete Datentypen (sequentielle Objekttypen) in Python – Strings, Listen und Tuples (Woyand 2019), S. 105.....	137
Abbildung 79: Beispiele von Slicing in Python (Woyand 2019), S. 106.....	137
Abbildung 80: Beispiele für List Comprehension in Python (Woyand 2019), S. 108.....	138
Abbildung 81: Beispiele für List Comprehension mit <i>if</i> -Anweisung in Python (Woyand 2019), S. 108.....	138
Abbildung 82: Beispiel für Verwendung von <i>iter</i> -Funktion und Methode <code>next()</code> in Python (Woyand 2019), S. 109.....	139
Abbildung 83: Beispiel für die Zusammenführung von zwei Listen mit <code>zip()</code> -Funktion in Python (Woyand 2019), S. 110.....	139
Abbildung 84: Schema zum Aufbau einer Klasse mit Beispiel in Python (Woyand 2019), S. 145, bearb.....	140
Abbildung 85: Beispiel für die Verwendung eines Konstruktors zur Wertzuweisung der Attribute <i>x</i> und <i>y</i> eines Objektes der Klasse <code>vektor()</code> (Woyand 2019), S. 151.....	141
Abbildung 86: Beispiel der Überladung des Operators <code>+</code> für Objekte der Klasse <code>vektor()</code> in Python (Woyand 2019), S. 154.....	142
Abbildung 87: Methoden zum Überladen von Operatoren in Python (Woyand 2019), S. 156.....	142
Abbildung 88: Gegenüberstellung von <code>sin()</code> -Funktion aus <i>math</i> -Modul und <i>numpy</i> für Liste und <code>numpy.array</code> in Python (Woyand 2019), S. 174.....	143

Abbildung 89: Beispiel für Erzeugung von Matrix mit <code>array()</code> aus NumPy in Python (Woyand 2019), S. 175.....	144
Abbildung 90: Beispiele für Erzeugung von Standardmatrizen mit NumPy-Funktionen <code>zeros()</code> und <code>ones()</code> sowie Vektoren mit <code>arange()</code> und <code>linspace()</code> aus NumPy in Python (Woyand 2019), S. 176.....	144
Abbildung 91: Beispiele für Array-Operationen aus NumPy in Python (Woyand 2019), S. 176-178.....	145
Abbildung 92: Beispiele für Lösung eines linearen Gleichungssystems und Inverse einer Matrix mit <code>numpy.linalg</code> in Python (Woyand 2019), S. 178-179...	145
Abbildung 93: Beispiel für Schreibweisen beim Slicing eindimensionaler Arrays aus NumPy in Python (Klein 2019), S. 57-58.....	146
Abbildung 94: Beispiel für Plot einer Sinus-Funktion mit Matplotlib in Python (Matplotlib 2022a).....	147
Abbildung 95: Grundlegende Objekte und Methoden einer Grafik mit Matplotlib in Python (Matplotlib 2022d).....	148
Abbildung 96: Beispiel für objektorientiert programmierten 2D-Linien-Plot mit Matplotlib in Python (Matplotlib 2022d).....	149
Abbildung 97: Beispiel für GUI mit widgets (Text, Input, FileBrowse, Button) mit PySimpleGUI in Python (erstellt mit PyCharm Community 2022).....	151
Abbildung 98: Dateien und Ordner eines neu erstellten Projekts mit MkDocs (Christie 2014a).....	152
Abbildung 99: Beispiel für einzeiligen docstring in Python (Goodger 2022).....	153
Abbildung 100: Beispiel für mehrzeiligen docstring in Python (Goodger 2022). 153	
Abbildung 101: Beispiel für docstring-Ausgabe mit <code>__doc__</code> in Python (Slatkin 2020), S. 451.....	154
Abbildung 102: Beispiele für Einrückung bei mehrzeiligen Anweisungen nach PEP 8 in Python (van Rossum 2022).....	155
Abbildung 103: Beispiele für Position von schließenden Klammern von Listen nach PEP 8 in Python (van Rossum 2022).....	156
Abbildung 104: Beispiel für Zeilenumbruch mit Backslash nach PEP 8 in Python (van Rossum 2022).....	156
Abbildung 105: Beispiel für Zeilenumbruch in Klammerausdruck vor Operatoren nach PEP 8 in Python (van Rossum 2022).....	156
Abbildung 106: Beispiele für <code>import</code> -Anweisungen nach PEP 8 in Python (van Rossum 2022).....	156
Abbildung 107: Beispiele für Leerzeichen nach Komma oder Semikolon nach PEP 8 in Python (van Rossum 2022).....	157

Abbildung 108: Beispiel für Leerzeichen um Operatoren nach PEP 8 in Python (van Rossum 2022).....	157
Abbildung 109: Beispiele für Leerzeichen in Schlüsselwort-Parametern von Funktionen nach PEP 8 in Python (van Rossum 2022).....	158
Abbildung 110: Beispiel für trailing comma und runde Klammern bei Erstellung von Tuple mit einem Wert nach PEP 8 in Python (van Rossum 2022).....	158
Abbildung 111: Beispiel für inline comment nach PEP 8 in Python (van Rossum 2022).....	158
Abbildung 112: Allgemeine Stile der Namensgebung (naming styles) (van Rossum 2022).....	159
Abbildung 113: Beispiel für Vergleich mit singleton <i>None</i> nach PEP 8 in Python (van Rossum 2022).....	160
Abbildung 114: Beispiel für Verwendung von einzeiliger definierter Funktion statt Lambda-Ausdruck nach PEP 8 in Python (van Rossum 2022).....	160
Abbildung 115: Beispiel für Rückgabewerte von Funktionen nach PEP 8 in Python (van Rossum 2022).....	160
Abbildung 116: Beispiel für Präfix-Prüfung nach PEP 8 in Python (van Rossum 2022).....	161
Abbildung 117: Beispiel für Objekttyp-Vergleich nach PEP 8 in Python (van Rossum 2022).....	161
Abbildung 118: Beispiel für Prüfung auf Inhalt bei sequentiellen Daten nach PEP 8 in Python (van Rossum 2022).....	161
Abbildung 119: Beispiele für Booleschen Wert in Bedingung nach PEP 8 in Python (van Rossum 2022).....	161
Abbildung 120: Einteilung von Dokumentation für Software nach Schritten eines Software-Prozesses.....	164
Abbildung 121: Einteilung von Bedienungsanleitungen für Software nach Inhalt	164
Abbildung 122: Phasen eines Software Development Life Cycle (SDLC) im Waterfall-Modell nach Winston Royce.....	165
Abbildung 123: Übersetzung eines Problems in Klassen mit Attributen und Methoden (links) und associations zwischen Klassen (rechts) am Beispiel einer Kursbelegung an einer Schule (Tech With Tim 2020), 34:30.....	170
Abbildung 124: Konstruktor einer Klasse <i>Person</i> mit Parametern, Attributen und Prüfung des Datentyps eines Parameters (<i>address</i>) am Beispiel einer Kursbelegung an einer Schule (Tech With Tim 2020), 14:30.....	172

Abbildung 125: Konstruktor einer Subklasse <i>Student(Person)</i> mit <i>super()</i> -Konstruktor am Beispiel einer Kursbelegung an einer Schule (Tech With Tim 2020), 19:39.....	173
Abbildung 126: Syntax-Varianten zur Ausgabe von Werten einfachen Datentyps mit <i>print()</i> in Python (Slatkin 2020), S. 404.....	174
Abbildung 127: Beispiel für Debugging mit <i>repr()</i> in Python (Slatkin 2020), S. 406	174
Abbildung 128: Übersicht Debugging mit <i>print()</i> in Python (Wellesley 2022)....	175
Abbildung 129: Übersicht Debugging durch Aufteilung und Auskommentierung von Ausdrücken in Python (Wellesley 2022).....	176
Abbildung 130: Beispiel für neue <i>Exception</i> -Subklasse in Python (TechVidvan 2022a).....	178
Abbildung 131: Beispiel für Fehlermeldung im Fall einer selbsterstellten exception in Python (TechVidvan 2022a).....	178
Abbildung 132: Beispiel für <i>try / except</i> – Anweisung mit selbstdefinierter Fehlermeldung im Fall einer exception in Python (TechVidvan 2022a).....	178
Abbildung 133: Beispiel für <i>try / except</i> – Anweisungen mit mehreren <i>except</i> -Anweisungsblöcken in Python (TechVidvan 2022a).....	179
Abbildung 134: Beispiel für <i>try / except</i> – Anweisungen mit <i>finally</i> -Anweisung in Python (TechVidvan 2022a).....	179
Abbildung 135: Beispiel für <i>try / except</i> – Anweisungen mit anschließender <i>else</i> -Anweisung in Python (TechVidvan 2022a).....	179
Abbildung 136: Legende mit Bedeutung von Symbolen in den Flowcharts für „QUEBER_Version2016“, vgl. (Wikipedia 2022c).....	180
Abbildung 137: Flowchart zum „Vorbereiten der Eingabedaten“ (Lesen aus Textdatei und Sortieren) in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	181
Abbildung 138: Flowchart zur Berechnung von grundlegenden Flächenmomenten und FSP in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	182
Abbildung 139: Flowchart Berechnung Verdrehwinkel und Hauptträgheitsachsen bzw. -radianen in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	183
Abbildung 140: Flowchart zur Berechnung der Polstrahle aller Elemente in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	184
Abbildung 141: Flowchart zur Berechnung der Elementlastvektoren und des Systemlastvektors in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	185
Abbildung 142: Flowchart zur Berechnung der Elementlastvektoren und des Systemlastvektors in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	186

Abbildung 143: Flowchart zur Berechnung des Torsionsträgheitsmoments aus offenen und geschlossenen Anteilen in „QUEBER_Version2016“, vgl. (Stanoev 2016).....	187
Abbildung 144: Flowchart zur Berechnung von Normierungskonstante und Wölfunktion bezogen auf S in „QUEBER_Version2016“, vgl. (Stanoev 2016).188	
Abbildung 145: Flowchart zur Berechnung des SMP M in „QUEBER_Version2016“, vgl. (Stanoev 2016).....189	
Abbildung 146: Flowchart zur Berechnung der Wölfunktion bezogen auf M und des Wölbträgheitsmoments in „QUEBER_Version2016“, vgl. (Stanoev 2016)..190	

Tabellenverzeichnis

Tabelle 1 Ausgabegrößen des Berechnungsprogramms.....	20
Tabelle 2 Eingabegrößen des Berechnungsprogramms.....	21
Tabelle 3 Funktionales Design des Python-Programms, vgl. (Stanoev 2016).....	69
Tabelle 4 Vergleich der Berechnungsergebnisse „QUEBER_Version2016“ und Python-Programm mit Eingabedatei „Profil_Wunderlich.txt“ (Wunderlich k.A.)...	82
Tabelle 5 Vergleich der Berechnungsergebnisse „QUEBER_Version2016“ und Python-Programm mit Eingabedatei „3kasten_Spkt.txt“ (Stanoev k.A. a).....	84
Tabelle 6 Vergleich der Berechnungsergebnisse „QUEBER_Version2016“ und Python-Programm mit Eingabedatei „FR-01-VAR1neu.txt“ (Peters k.A. b).....	86
Tabelle 7 Vor- und Nachteile des Waterfall-Modells nach Winston Royce (JavaTpoint 2021b).....	166
Tabelle 8 Vor- und Nachteile von modularity im Software-Design.....	167

Abkürzungsverzeichnis

Maschinenbau

FEM Finite Elemente Methode
FSP Flächenschwerpunkt
HTA Hauptträgheitsachse(n)
KS Koordinatensystem
SMP Schubmittelpunkt
WEA Windenergieanlage
WT Windturbine

Programmierung

API Application programming interface (Programmierschnittstelle)
GUI Graphical User Interface
IDE Integrated Development Environment
IDLE Integrated Development and Learning Environment
OOP Objektorientierte Programmierung (Object-oriented programming)
SDD Software Design Document
SDLC Software Design Life Cycle
SLOC Source lines of code
SRS Software Requirement Specification

Formelzeichenverzeichnis

Formelzeichen		Dimension	Bezeichnung
Programm	Arbeit /	Länge (L)	kein Wert (-)
Literatur		Masse (M)	dimensionslos ([])
		Zeit (T)	Matrix [m,n] ; Dim. der Einträge
(1 , 2)	(1,2)	(L , L)	Koordinaten im Hauptträgheitsachsen-KS (HTA-KS)
	$a = \omega_0$	L^2	Koeffizient a im Ansatz der Wölbfunction $\omega(y,z)$, Normierungskonstante ω_0
	$a_y = z_M$	L	Koeffizienten a_y und a_z im Ansatz der Wölbfunction $\omega(y,z)$,
	$a_z = -y_M$		Koordinaten des SMP M ($-a_z = y_M$, $a_y = z_M$) im FSP-KS
A	A	L^2	Querschnittsfläche, Flächenmoment 0. Grades
A_i	A_i	L^2	Fläche des i -ten Elements, Elementfläche
	A_s	L^2	Von der Profilmittellinie eingeschlossene Fläche
	A_m		
A_qy, A_qz	A_{qy}, A_{qz}	L^2	Schubfläche, Querschubfläche, bezogen auf Achsen des Ursprungs-KS bzw. FSP-KS
	A_{Sy}, A_{Sz}		
A_wwM	$I_{\omega\omega M}, I_{\omega\omega}$	L^6	Wölbträgheitsmoment, Wölbflächenmoment 2. Ordnung, bezogen auf SMP M
	$A_{\omega\omega M}$		
	$A_{\omega\omega}$		
	$I_{\tilde{\omega}} = A_{\tilde{\omega}}$		Wölb-Flächenmomente
	$I_{y\tilde{\omega}} = A_{y\tilde{\omega}}$		
	$I_{z\tilde{\omega}} = A_{z\tilde{\omega}}$		
A_y, A_z	$S_{\bar{y}}, S_{\bar{z}}$	L^3	Statisches Moment, Flächenmoment 1. Grades, bezogen auf Ursprungs-KS
	$A_{\bar{y}}, A_{\bar{z}}$		

A_yy, A_zz	$I_{\bar{y}\bar{y}}, I_{\bar{z}\bar{z}}$	L^4	Flächenträgheitsmoment, axiales Flächenmoment 2. Grades, bezogen auf Ursprungs-KS
A_yyS, A_zzS	I_{yy}, I_{zz} A_{yy}, A_{zz}	L^4	Flächenträgheitsmoment, bezogen auf FSP-KS
A_yz	$I_{\bar{y}\bar{z}}$ $A_{\bar{y}\bar{z}}$	L^4	Deviationsmoment, biaxiales Flächenmoment 2. Grades, bezogen auf Ursprungs-KS
A_yzS	I_{yz} A_{yz}	L^4	Deviationsmoment, bezogen auf FSP-KS
DICKE	[NM , 1] ; L		Liste der Dicke t_i der Elemente
	$EI_{\omega\omega M}$		Verwölbungssteifigkeit,
	$EI_{\omega\omega} EI_{\omega}$		Wölbsteifigkeit, bezogen auf SMP M
	$EA_{\omega\omega M}$		
	F	$M L T^{-2}$	Kraft / Äußere Kraft
G	G	$M L^{-1} T^{-2}$	Schubmodul
	GA_{qy}	$M L T^{-2}$	Schubsteifigkeit,
	$\kappa_y GA$	/	Querschubsteifigkeit, bezogen auf Achse des FSP-KS
	GA_{qz}		
	$\kappa_z GA$		
i1, i2	I_1, I_2	L^4	Hauptträgheitsmoment
i1, i2	i_1, i_2	L	Hauptträgheitsradius
IT	I_T	L^4	Torsionsträgheitsmoment
	$J_{1,2}$	$M L^2$	Massenträgheitsmomente, bezogen auf Hauptachsen
Ka	$k_{a,i}$	Ø	Nummer des Anfangsknotens des i -ten Elements
Ke	$k_{e,i}$	Ø	Nummer des Endknotens des i -ten Elements
KNOKO	[NK, 2] ; L		Liste der Knotenpunkt-Koordinaten
	k_i	[2, 2] ; $M L^{-1} T^{-2}$	Elementsteifigkeitsmatrix für Wölfunktion

	K	Systemsteifigkeitsmatrix für Wölfunktion	
L	l_i	L	Länge des i -ten Elements
LANGE		[NM, 1] ; L	Liste der Länge l_i der Elemente
M_{SMP}	(L, L)		Koordinatenursprung des SMP-KS, Drillruhepunkt, Querkraftmittelpunkt
M_x	$M \text{ L}^2 \text{T}^{-2}$		Torsionsmoment um Schwerpunktlinie x
M_{xp}	$M \text{ L}^2 \text{T}^{-2}$		Primäre Torsion, Saint-Venantsche Torsion
M_{xs}	$M \text{ L}^2 \text{T}^{-2}$		Sekundäre Torsion, Wölbkrafttorsion
M_ω	$M \text{ L}^2 \text{T}^{-2}$		Wölbmoment, Bimoment, Wölbmoment
M_{by}, M_{bz}	$M \text{ L}^2 \text{T}^{-2}$		Biegemomente
N	$M \text{ L} \text{T}^{-2}$		Normalkraft (in x-Richtung)
N_c	Ø		Gesamtanzahl der Zellen eines geschlossenen Querschnitts
NK	N_k	Ø	Gesamtanzahl der Knotenpunkte
NM	N_m	Ø	Gesamtanzahl der Elemente
O	(L, L)		Koordinatenursprung des Ursprungs-KS
p_i	[2, 1] ; $M \text{ L} \text{T}^{-2}$		Elementlastvektor für Wölfunktion
\underline{P}	[NK, 1] ; $M \text{ L} \text{T}^{-2}$		Systemlastvektor für Wölfunktion
Q	$M \text{ L} \text{T}^{-2}$		Querkraft
$r_{t,i}, r_{\perp i}$	L		Polstrahl / Hebel des i -ten Elements
$r_{tM,i}$	L		Polstrahl bezogen auf SMP M
$r_{tO,i}$	L		Polstrahl bezogen auf Ursprung O
RT	r_t , r_\perp	[NM , 1] ; L	Liste der Polstrahle / Hebel der Elemente

S	(L, L)	Koordinatenursprung des FSP-KS	
FSP			
S_i	(L, L)	FSP des i -ten Elements	
FSP_i			
s	- bzw. L	Umfangsrichtung bzw. Koordinate in Umfangsrichtung, Umlaufkoordinate	
STAB	[NM , 2] ; []	Anfangs- und Endpunkte der Elemente in Kastenprofilen	
STAB_ OFF		Anfangs- und Endpunkte der Stab- elemente in offenen Profilteilen	
t	t_i	L	Dicke des i -ten Elements
	T		Torsionsmoment, bezogen auf SMP
	T_s	q M T ⁻²	(Bredtscher) Schubfluss
	V		Vektor für die Werte der Verwölbung $\tilde{\omega}_{s,k}$ in den Knotenpunkten
	W_T	L ³	Torsionswiderstandsmoment
x		L	Koordinaten im FSP-KS in Richtung der Schwerpunktlinie x (x-Achse, Drillruheachse)
(y , z)	(\bar{y} , \bar{z})	(L, L)	Koordinaten im Ursprungs-KS
(y_s , z_s)	(y , z)	(L, L)	Koordinaten im FSP-KS
(Ya, Za)	($\bar{y}_{a,i}$, $\bar{z}_{a,i}$)	(L , L)	Koordinaten des Anfangspunkts $k_{a,i}$ des i -ten Elements im Ursprungs-KS
(Ye, Ze)	($\bar{y}_{e,i}$, $\bar{z}_{e,i}$)	(L , L)	Koordinaten des Endpunkts $k_{e,i}$ des i -ten Elements im Ursprungs-KS
(Ym, Zm)	(\bar{y}_M , \bar{z}_M)	(L , L)	Schubmittelpunkt im Ursprungs-KS
(YmS, ZmS)	(y_M , z_M)	(L , L)	Schubmittelpunkt im FSP-KS
	(a_y , a_z)		
	(e_y , e_z)		
(Ys, Zs)	(\bar{y}_s , \bar{z}_s)	(L , L)	Flächenschwerpunkt, geometrischer Schwerpunkt im Ursprungs-KS
	α_i	Rad / °	Steigung des i -ten Elements

	∂	-	Partielle Ableitung einer Größe
	δ	-	Infinitesimale Änderung einer Größe
	d	-	Differenz / Änderung einer Größe
Phi	$\varphi_{1,2}$	Rad	Hauptträgheitsachsenwinkel, HTA-Winkel
	φ_0, φ		
	φ_x	Rad	Verdrehung des Querschnitts
	φ'_x	Rad / L	Verdrillung / Verwindung des Querschnitts
TS	ϑ	Rad	Verdrillung / Verwindung des Querschnitts
	ϑ'		
TS	Φ	L^2	Bezogener Schubfluss, Torsionsfunktion, konstanter bezogener Schubfluss
	K_y, K_z	0	Schubkorrekturfaktor, Querschubzahl, bezogen auf Achse des FSP-KS
	σ_{xx}	$M L^{-1} T^{-2}$	Normalspannung in x-Richtung
	$\sigma_{\omega\omega}$	$M L^{-1} T^{-2}$	Wölbnormalspannungen in x- Richtung (Biegespannungen durch Wölbkrafttorsion)
	$\tau(s)$	$M L^{-1} T^{-2}$	Schubspannung an der Stelle s der Umlaufkoordinate s
	$\tau_{s,p}(s)$	$M L^{-1} T^{-2}$	Primäre Schubspannungen in Umfangsrichtung s (durch Saint- Venant'sche Torsion)
	$\tau_{xs}^0(s)$		
	$\tau_{xs}^1(s)$		
	$\tau_{s,s}(s)$	$M L^{-1} T^{-2}$	Sekundäre Schubspannungen in Umfangsrichtung s (durch Wölbkrafttorsion)
	ω	L^2	Verwölbung
	$\omega(s)$	L^2	Wölbfunction
	$\tilde{\omega}_i(s_i)$	L^2	Verwölbungsfunktion eines i-ten Elements
	$\tilde{\omega}_{,s}(s)$	L	Ableitung Wölbfunction in Umfangsrichtung s

OM	ω	L^2	Wölbfunktion, bezogen auf Ursprungs-KS / O
OMS	ω_s	L^2	Wölbfunktion (Einheitsverwölbung), bezogen auf FSP S
OMM	ω_M	L^2	Wölbfunktion (Einheitsverwölbung), bezogen auf SMP M

Indize Programm	Arbeit / Literatur	Bezeichnung / Größe
1, 2	1, 2	Bezogen auf HTA-KS
	b	Biegung
i	i	Nummer eines Elements
k	k	Nummer eines Knotens
m , _m, M	M	Bezogen auf SMP M
	p, q	Nummer der Zelle des geschlossenen Querschnitts
$_s, s$	y, z	Bezogen auf FSP S, bezogen auf Achse(n) des FSP-KS
$_S, S$	S	
$_y, _z, O$	\bar{y}, \bar{z}, O	Bezogen auf Ursprungs-KS
t	, \perp	Senkrecht zur Bezugsgerade (zu Umfangsrichtung s_i)

1. Einleitung

Es gibt vielseitige Möglichkeiten, mithilfe von kommerzieller oder freier Software numerische Berechnungen im Bereich der Strukturmechanik durchzuführen. Dabei stehen fertige FEM-Programme wie Ansys, RFEM oder Marc Mentat zur Verfügung. Außerdem enthalten CAD-Programme wie Inventor oder FreeCAD einige Funktionalitäten zur Auslegung der modellierten Bauteile.

Programmiersprachen, insbesondere Matlab, C# und Python, bieten Möglichkeiten zur eigenen Erstellung von numerischen Berechnungsalgorithmen. Der Vorteil eigenhändig entwickelter Software besteht in der individuellen Gestaltung und Kontrolle des Programmaufbaus und der passenden Formatierung der Eingabe- und Ausgabedaten. Zudem fördert das Programmieren das tiefe Verständnis der Berechnungsabläufe. Schließlich erlaubt es auch eine bewusste Optimierung der Ergebnisqualität und ein zielgerichtetes Fehlermanagement direkt in der Software.

Kommerzielle Software überzeugt mit einem großen Funktionsumfang sowie Softwarewartung und -updates. Allerdings bedeutet dies auch einen hohen Einarbeitungs- und Weiterbildungsaufwand sowie regelmäßige Kosten für die Lizenzen. Außerdem sind die numerischen bzw. strukturmechanischen Hintergründe für den Anwender nicht immer ersichtlich und nachvollziehbar.

In Zeiten des wachsenden Spezialisierungsgrades von Ingenieurstätigkeiten bietet selbstständig erstellte FEM-Software eine attraktive Möglichkeit zur Lösung der technischen Aufgabenstellungen.

Diese Masterarbeit liefert ein in Python geschriebenes Programm zur Berechnung von Werten für einen Bauteilquerschnitt, siehe Tabelle 1.

Tabelle 1 Ausgabegrößen des Berechnungsprogramms

Bezeichnung / Größe	Formelzeichen im Programm
<u>Werte im Ursprungs-KS</u>	
• geometrischer Schwerpunkt	Ys, Zs
• Querschnittsfläche	A
• Schubfläche	Aqy, Aqz
• Statisches Moment	A_y, A_z
• Flächenträgheitsmoment	A_yy, A_zz
• Deviationsmoment	A_yz
• Schubmittelpunkt	Ym, Zm
<u>Werte im FSP-KS</u>	

• Flächenträgheitsmoment	A_yyS, A_zzS
• Deviationsmoment	A_yzS
• Verdrehwinkel	Phi
• Hauptträgheitsmoment	I1, I2
• Hauptträgheitsradius	i1, i2
• Torsionsträgheitsmoment	IT
• Schubmittelpunkt	YmS, ZmS
<u>Werte der Wölfunktion</u>	
• Wölbträgheitsmoment	A_wwM
• Wölfunktion, bezogen auf Ursprungs-KS	OM
• Wölfunktion, bezogen auf FSP-KS	OMS
• Wölfunktion, bezogen auf SMP-KS	OMM
<u>Sonstige</u>	
• Bezugener Schubfluss	TS

Die benötigten Eingabewerte sind allesamt Informationen über die Geometrie des Querschnitts, siehe Tabelle 2.

Tabelle 2 Eingabegrößen des Berechnungsprogramms

Bezeichnung / Größe	Formelzeichen im Programm
• Anzahl der Knotenpunkte	NK
• Anzahl der Elemente	NM
• Knotenpunkt-Koordinaten	KNOKO
• Anfangs- und Endpunkte der Elemente	STAB
• Dicke der Elemente	DICKE

Wie im Titel der Masterarbeit beschrieben funktioniert das Programm für alle polygonalen dünnwandigen Querschnitte mit offener und/oder geschlossener Geometrie. Eine Grundannahme ist ein einheitliches isotropes Material.

Zur ersten Plausibilitätsprüfung und Visualisierung der Berechnungsergebnisse besteht ein Teil der Aufgabe aus dem Erstellen einer 2D- und einer 3D-Grafik. Sie beinhalten die Querschnittsgeometrie im Ausgangskoordinatensystem, den geometrischen Schwerpunkt und den Schubmittelpunkt. Außerdem erscheint in

der 3D-Grafik die Wölbfunction bezogen auf den Schubmittelpunkt.

Grundlage und Vorbild für die Arbeit ist das Matlab MuPad Notebook „QUEBER_Version2016“ (Stanoev 2016). Die Bezeichnung der verwendeten Größen, die Zielsetzung, der Funktionsumfang und die Art der Grafiken ist größtenteils identisch.

Das C# Programm „Thesis_Interface“ (Samlaf-Adams 2020) / (Vent 2022) mit einer Querschnittswerteberechnung im Falle von inhomogenen Werkstoffeigenschaften bei Rotorblättern von WEA spielt ebenfalls eine große Rolle bei dem Software-Design. Dies gilt insbesondere in Bezug auf die objektorientierte Programmierung. Diese spielt in Matlab keine zentrale Rolle, wohingegen Python ebenfalls diese Möglichkeit der Strukturierung von Methoden und Funktionen bietet. Daher besteht eine Teilaufgabe aus der Erweiterung einer 2D-Grafik in „Thesis_Interface“ um die Wölbfunction.

Somit vereint das neue Programm in Python den objektorientierten Programmierstil und das Software-Design von C# mit den umfangreichen Bibliotheken und Funktionen für Numerik und grafische Datenvisualisierung von Matlab. In Python gibt es für letzteres die Packages Numpy und Matplotlib (NumPy 2022) / (Matplotlib 2022).

Das Skript zur Vorlesung „Dünnwandige Stabsysteme“ (Stanoev 2013) bildet eine wichtige Grundlage für die strukturmechanischen Hintergründe der Arbeit.

Die berechneten Werte ermöglichen den nächsten Schritt in der statischen und/oder dynamischen Auslegung des Bauteils. So benötigt die Berechnung der Torsions- und Biegeigenmoden die Flächenträgheitsmomente zur Erstellung der Steifigkeitsmatrix. Wenn die aufgebrachten Lasten bekannt sind, bildet die Steifigkeitsmatrix auch den Ausgangspunkt für die Berechnung von Schnitt- und Verschiebungsgrößen der Elemente. Das ermöglicht schließlich die Dimensionierung des Querschnitts und die Auswahl eines Werkstoffs. Die Auslegung von Bauteilverbindungen baut auch auf diesen Schritten auf.

2. Kenntnisstand

Die Masterarbeit basiert auf Theorien der Festigkeitslehre und Strukturmechanik (siehe Anhang 8.1.2). Dazu kommen erforderliche Kenntnisse über die Programmiersprache Python (siehe Anhang 8.2) sowie Aufbau und Strukturierung von (FEM-) Berechnungsprogrammen im Allgemeinen (siehe Anhang 8.1.1).

2.1 Querschnittswerte für Zug/Druck, Biegung und Schub

2.1.1 Aufbau des Modells

Der Gesamtquerschnitt besteht aus mehreren beliebig angeordneten N_m Rechteck-Elementen. Diese sind über den Anfangsknoten $k_{a,i}$ und den Endknoten $k_{e,i}$ sowie die Dicke t_i in ihrer Geometrie vollständig definiert. Der Anfangsknoten $k_{a,i}$ besitzt die Koordinaten $(\bar{y}_{a,i}, \bar{z}_{a,i})$ und der Endknoten $k_{e,i}$ die Koordinaten $(\bar{y}_{e,i}, \bar{z}_{e,i})$. Die Koordinaten beziehen sich auf das Ursprungs-Koordinatensystem $\bar{x}, \bar{y}, \bar{z}$ (Ursprungs-KS) mit dem Koordinatenursprung O . Abbildung 1 stellt diese und die nachfolgend genannten Größen grafisch dar.

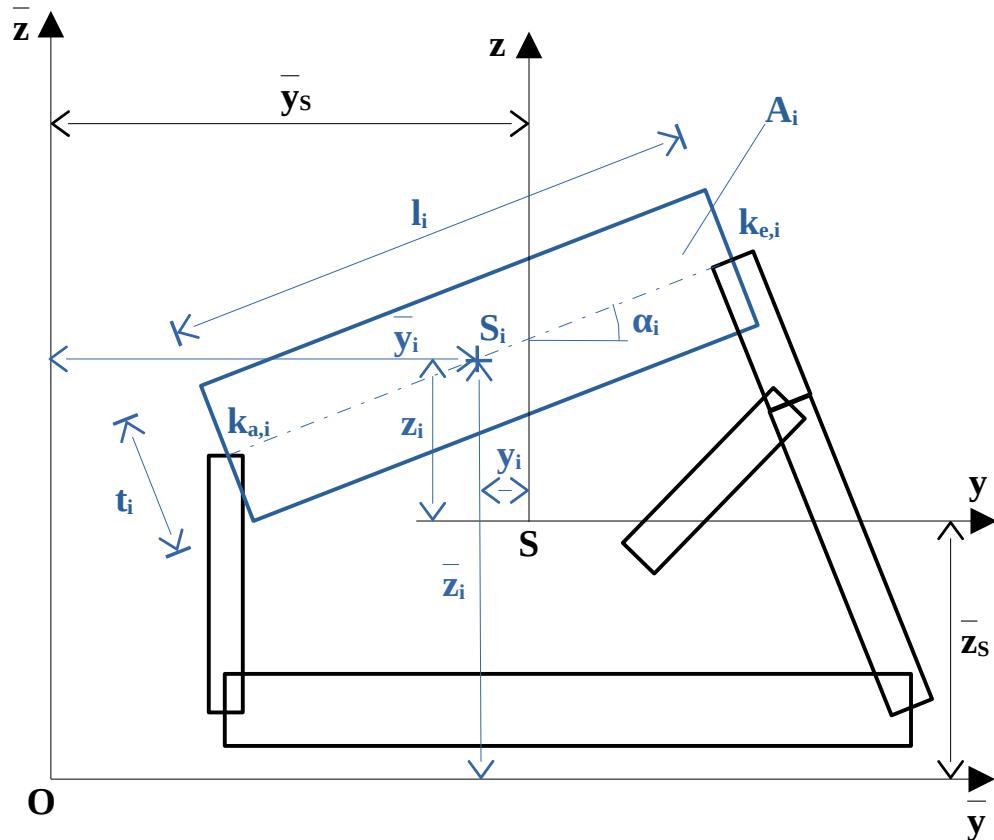


Abbildung 1: Schematische Darstellung der geometrischen Größen zur Definition des i -ten Elements des Querschnitts

Die Darstellung zeigt das Flächenschwerpunkt-KS x, y, z (FSP-KS). Die Ermittlung des Flächenschwerpunktes S (FSP) des gesamten Querschnitts benötigt den Flächenschwerpunkt S_i und die Fläche A_i aller N_m Elemente. Die Koordinaten des FSP im Ursprungs-KS sind (\bar{y}_s, \bar{z}_s) .

Dazu wird zunächst die Länge l_i des i -ten Elements bestimmt. Nach Satz des Pythagoras entsteht Gleichung 2.1 aus den Koordinaten des Anfangsknotens $k_{a,i}$ und des Endknotens $k_{e,i}$.

$$l_i = \sqrt{(\bar{y}_{e,i} - \bar{y}_{a,i})^2 + (\bar{z}_{e,i} - \bar{z}_{a,i})^2} \quad (\text{Gleichung 2.1})$$

Die Koordinaten $(\bar{y}_{Si}, \bar{z}_{Si})$ des i -ten Element-FSP S_i benötigen für die Berechnung ebenfalls die Koordinaten von $k_{a,i}$ und $k_{e,i}$, siehe Gleichung 2.2 / Gleichung 2.3.

$$\bar{y}_{Si} = \frac{1}{2}(\bar{y}_{a,i} + \bar{y}_{e,i}) \quad (\text{Gleichung 2.2})$$

$$\bar{z}_{Si} = \frac{1}{2}(\bar{z}_{a,i} + \bar{z}_{e,i}) \quad (\text{Gleichung 2.3})$$

(Petersen 2013), S. 1202 / (Stanoev 2016)

Das beschriebene Modell basiert auf einigen für die numerische Berechnung günstigen Vereinfachungen. So erfordert die exakte Bestimmung der Querschnittswerte eine Einteilung des Querschnitts in unendlich viele infinitesimale Teilstücke, wie es Abbildung 2 zeigt. Das verwendete Modell sucht daher den Kompromiss zwischen Rechengenauigkeit und Rechenaufwand.

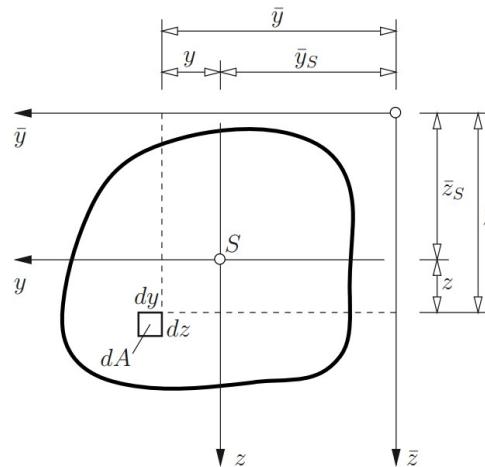


Abbildung 2: Darstellung von Ursprungs-KS ($\bar{x}, \bar{y}, \bar{z}$) und FSP-KS (x, y, z) für einen beliebig geformten Querschnitt, mit beispielhaft eingetragener infinitesimaler Teilfläche dA (Hofstetter 2013), S. 160

Die Beschränkung auf Rechteck-Elemente macht die Berechnung der Länge l_i , der Elementfläche A_i und des Element-FSP $(\bar{y}_{Si}, \bar{z}_{Si})$ wie gezeigt sehr einfach.

Für dünnwandige Profile, mit $t \ll$ sonstige Profilabmessungen (hier a), ist eine teilweise Überlappung bzw. Aussparung an den Elementübergängen zulässig. Die berechneten Flächenmomente stimmen mit guter Genauigkeit. Die Mittellinien der Elemente liegen dabei auf der Profilmittellinie des originalen Querschnitts, siehe Abbildung 3. (Linke 2015), S. 75-78

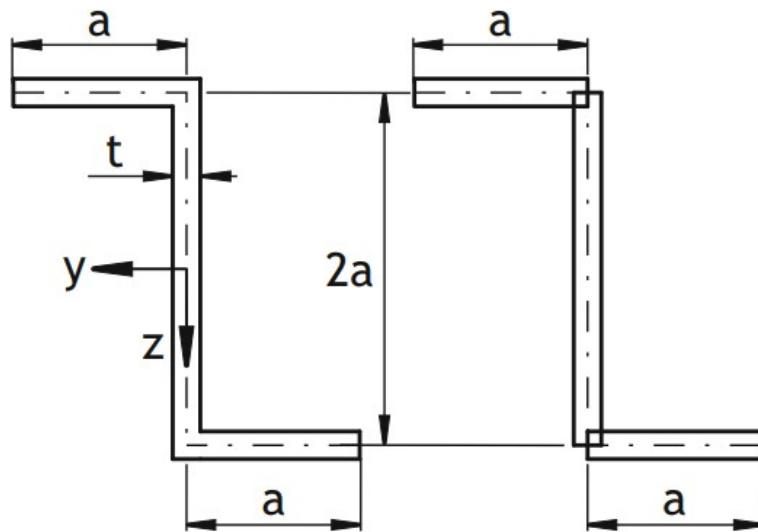


Abbildung 3: Beispiel der Einteilung in Elemente bei einem Z-Profil, zulässige Ungenauigkeit in der Modellierung der Ecken bzw. Elementübergänge (rechts) (Linke 2015), S. 76

2.1.2 Grundlegende Flächenmomente

Gleichung 2.4 beschreibt die Berechnung der Querschnittsfläche A für den Gesamtquerschnitt aus N_m Rechteck-Elementen mit den Querschnittsflächen A_i . Die Ausgangsdaten sind die Dicke t_i und die Länge l_i des i -ten Elements.

$$A = \int_A dA = \sum_i^{N_m} A_i = \sum_i^{N_m} t_i l_i \quad (\text{Gleichung 2.4})$$

Die statischen Momente $S_{\bar{y}}$ bzw. $S_{\bar{z}}$ des Gesamtquerschnitts werden durch die Summe der statischen Momente der Elemente $S_{\bar{y},i}$ bzw. $S_{\bar{z},i}$ beschrieben, wie in Gleichung 2.5 und Gleichung 2.6 zu sehen. Dabei bezieht sich das statische Moment auf eine Achse des Ursprungs-KS, \bar{y} oder \bar{z} . Die Parameter \bar{y}_{Si} und \bar{z}_{Si} geben den Abstand des FSP S_i des i -ten Elements zur Bezugssachse an.

$$S_{\bar{y}} = \int_A \bar{y} dA = \sum_{i=1}^{N_m} S_{\bar{y},i} = \sum_{i=1}^{N_m} \bar{y}_{Si} A_i \quad (\text{Gleichung 2.5})$$

$$S_{\bar{z}} = \int_A \bar{z} dA = \sum_{i=1}^{N_m} S_{\bar{z},i} = \sum_{i=1}^{N_m} \bar{z}_{Si} A_i \quad (\text{Gleichung 2.6})$$

Analog zum statischen Moment beschreiben Gleichung 2.7 und Gleichung 2.8 das Flächenträgheitsmoment $I_{\bar{y}\bar{y}}$ bzw. $I_{\bar{z}\bar{z}}$ des Gesamtquerschnitts im Ursprungs-KS. $I_{\bar{y}\bar{y},i}$ und $I_{\bar{z}\bar{z},i}$ sind die Flächenträgheitsmomente des i -ten Elements.

$$I_{\bar{y}\bar{y}} = \int_A \bar{y}^2 dA = \sum_{i=1}^{N_m} I_{\bar{y}\bar{y},i} = \sum_{i=1}^{N_m} \bar{y}_{Si}^2 A_i \quad (\text{Gleichung 2.7})$$

$$I_{\bar{z}\bar{z}} = \int_A \bar{z}^2 dA = \sum_{i=1}^{N_m} I_{\bar{z}\bar{z},i} = \sum_{i=1}^{N_m} \bar{z}_{Si}^2 A_i \quad (\text{Gleichung 2.8})$$

Gleichung 2.9 zeigt die Berechnung des Deviationsmoments $I_{\bar{y}\bar{z}}$ (gleich $I_{\bar{z}\bar{y}}$).

$$I_{\bar{y}\bar{z}} = I_{\bar{z}\bar{y}} = \int_A \bar{y} \bar{z} dA = \sum_{i=1}^{N_m} I_{\bar{y}\bar{z},i} = \sum_{i=1}^{N_m} \bar{y}_{Si} \bar{z}_{Si} A_i \quad (\text{Gleichung 2.9})$$

Der Flächenschwerpunkt FSP (\bar{y}_s , \bar{z}_s) im Ursprungs-KS ergibt sich aus Gleichung 2.10 und Gleichung 2.11 mit den statischen Momenten $S_{\bar{y}}$ und $S_{\bar{z}}$.

$$\bar{y}_S = \frac{S_{\bar{z}}}{A} \quad (\text{Gleichung 2.10})$$

$$\bar{z}_S = \frac{S_{\bar{y}}}{A} \quad (\text{Gleichung 2.11})$$

Da die Koordinaten des FSP im FSP-KS gleich null sind, müssen auch die statischen Momente S_y und S_z im FSP-KS gleich null sein, siehe Gleichung 2.12 und Gleichung 2.13.

$$S_y = z_S A = 0 \cdot A = 0 \quad (\text{Gleichung 2.12})$$

$$S_z = y_S A = 0 \cdot A = 0 \quad (\text{Gleichung 2.13})$$

Die Koordinaten im FSP-KS und im Ursprungs-KS haben die in Gleichung 2.14 beschriebene Beziehung zueinander. Dieser Zusammenhang führt zu Gleichung 2.15, Gleichung 2.16 und Gleichung 2.17. Die drei Gleichungen beschreiben die Berechnung der Flächenträgheitsmomente I_{yy} und I_{zz} sowie des Deviationsmomentes I_{yz} mit Bezug auf das FSP-KS. Der Satz des Steiner als Ansatz führt zu den selben Gleichungen. Diese Umrechnung auf den FSP wird als Erste Querschnittsnormierung bezeichnet. Die nicht wie $N_{\bar{x}}$ exzentrisch angreifende Normalkraft N_x (Wirkungslinie ist x -Achse) verursacht keine zusätzlichen Anteile an den Biegemomenten M_{by} und M_{bz} .

$$x = \bar{x} \quad y = \bar{y} - \bar{y}_S \quad z = \bar{z} - \bar{z}_S \quad (\text{Gleichung 2.14})$$

$$I_{yy} = I_{\bar{y}\bar{y}} - \bar{z}_S^2 A \quad (\text{Gleichung 2.15})$$

$$I_{zz} = I_{\bar{z}\bar{z}} - \bar{y}_S^2 A \quad (\text{Gleichung 2.16})$$

$$I_{yz} = I_{\bar{y}\bar{z}} - \bar{y}_S^2 \bar{z}_S^2 A \quad (\text{Gleichung 2.17})$$

(Mittelstedt 2021), S. 169-177, S. 192-194

2.1.3 Hauptträgheitsmomente und -achsen

Die folgenden Berechnungen umfassen die Zweite Querschnittsnormierung. Sie sind für unsymmetrische Querschnitte erforderlich, da hier die beiden Hauptträgheitsachsen (HTA) 1 und 2 nicht mit den Achsen y und z des FSP-KS zusammenfallen. Ziel ist eine weitere Vereinfachung der Schnittgrößenberechnung durch Änderung des Bezugssystems auf das HTA-KS. Hier sind Normalkraft N_x sowie die Biegemomente M_{b1} und M_{b2} rechentechnisch durch das Verschwinden des Deviationsmomentes I_{12} vollständig voneinander entkoppelt.

Dazu wird das FSP-KS um den Hauptträgheitsachsenwinkel (HTA-Winkel) $\varphi_{1,2}$ um die x -Achse gedreht. So entstehen die beiden HTA 1 und 2. Die x -Achse im HTA-KS ist somit identisch mit der x -Achse aus dem FSP-KS. Die beiden Eigenschaften des HTA-KS sind in Gleichung 2.18 beschrieben. Abbildung 4 verdeutlicht den qualitativen Zusammenhang zwischen FSP-KS und HTA-KS.

$$x_{(FSP)} = x_{(HTA)} \quad I_{12} = 0 \quad (\text{Gleichung 2.18})$$

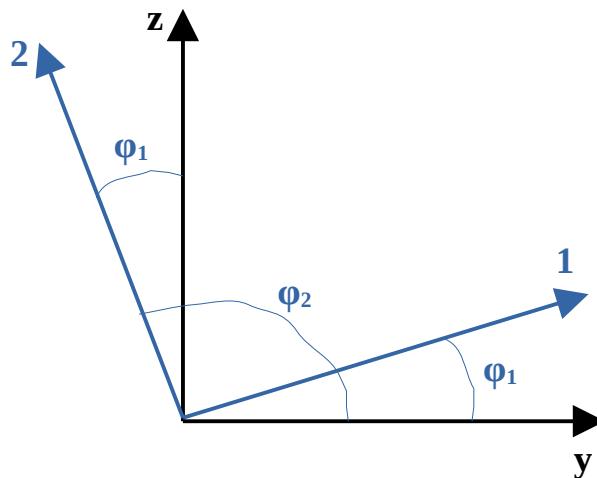


Abbildung 4: Qualitative Darstellung von FSP-KS, HTA-KS und HTA-Winkel $\varphi_{1,2}$

Die Berechnung der Hauptträgheitsmomente I_1 und I_2 erfolgt wie in Gleichung 2.19 beschrieben. Ein Hauptträgheitsmoment ist dabei das maximale (i.d.R. I_1) und das andere (i.d.R. I_2) das minimale Flächenträgheitsmoment um die x -Achse.

$$I_{1,2} = \frac{I_{yy} + I_{zz}}{2} \pm \sqrt{\left(\frac{I_{yy} - I_{zz}}{2}\right)^2 + I_{yz}^2}, \text{ i.d.R. } I_1 > I_2 \quad (\text{Gleichung 2.19})$$

Der HTA-Winkel $\varphi_{1,2}$ folgt aus Gleichung 2.20. Für $\varphi_2 = \varphi_{1,2}$ vertauschen sich die Hauptträgheitsmomente I_1 und I_2 im Wert gegenüber $\varphi_1 = \varphi_{1,2}$.

$$\tan(2\varphi_{1,2}) = \frac{2I_{yz}}{I_{zz} - I_{yy}} \quad \varphi_1 = \varphi_{1,2} \quad \varphi_2 = \varphi_1 + \frac{\pi}{2} \quad (\text{Gleichung 2.20})$$

Die Transformationsmatrix dient zur Umrechnung von Größen aus dem FSP-KS in das HTA-KS. Beispielhaft zeigt Gleichung 2.21 die Umrechnung von Koordinaten. Gleiches gilt für die Biegemomente M_{b1} , M_{b2} aus M_{by} , M_{bz} .

$$\begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{bmatrix} \cos(\varphi_{1,2}) & \sin(\varphi_{1,2}) \\ -\sin(\varphi_{1,2}) & \cos(\varphi_{1,2}) \end{bmatrix} \begin{pmatrix} y \\ z \end{pmatrix} \quad (\text{Gleichung 2.21})$$

Die Trägheitsradien i_1 und i_2 dienen zur Berechnung der Massenträgheitsmomente J_1 und J_2 um die jeweilige Hauptträgheitsachse, siehe Gleichung 2.22.

$$i_{1,2} = \sqrt{\frac{I_{1,2}}{A}} \quad J_{1,2} = m i_{1,2}^2 \quad (\text{Gleichung 2.22})$$

(Wikipedia, 2022b)

(Hofstetter 2013), S. 163-164 / (Mittelstedt 2021), S. 182-184

2.1.4 Schubflächen

Berechnungen bei Querkraftbelastung gehen vereinfacht vom Ebenbleiben des Querschnitts aus. Weiterhin gilt die Annahme von konstant im Querschnitt verteilten Schubspannungen (statt parabolischer Verteilung). Die Balkentheorie nach Timoshenko beinhaltet Schubspannungen und Scherung. Sie benötigt daher Schubkorrekturfaktoren (Querschubzahlen) κ_y und κ_z zur Berücksichtigung der genannten Vereinfachungen, siehe Gleichung 2.23. Schubspannungen sind hierbei auf fiktive richtungsabhängige Schubflächen (Querschubflächen) A_{qy} (A_{sy}) und A_{qz} (A_{sz}) bezogen. Sie ermöglichen die Ermittlung von richtungsabhängigen Schubsteifigkeiten (Querschubsteifigkeiten) $\kappa_y GA = GA_{qy}$ und $\kappa_z GA = GA_{qz}$ (vgl. Anhang 8.1.2). Standardwerte für Rechtecke aus Stahl sind $\kappa = 5/6 = 0,833$ bzw. $\kappa = 0,85$ (wenn an KS-Achsen ausgerichtet). (Nasdala 2015), S. 144-145

$$A_{qy} = \kappa_y A \quad A_{qz} = \kappa_z A \quad (\text{Gleichung 2.23})$$

Die mittlere Schubspannung τ_m im Querschnitt ergibt sich mit Gleichung 2.24.

$$\tau_m = \frac{Q_y}{A_{qy}} = \frac{Q_z}{A_{qz}} = G \gamma_m \quad (\text{Gleichung 2.24})$$

Ein Ansatz zur Berechnung von Schubkorrekturfaktoren liefert eine Formänderungsenergie im Querschnitt mit infinitesimaler Länge dx durch Querkraft Q_y bzw. Q_z . Das Ergebnis ist Gleichung 2.25.

$$\kappa_z = \frac{Q_z^2}{A \int_A \tau_m^2 dA} \quad \kappa_y = \frac{Q_y^2}{A \int_A \tau_m^2 dA} \quad (\text{Gleichung 2.25})$$

(Linke 2015), S. 182-183

Eine weitere Schreibweise für Gleichung 2.25 steht in Gleichung 2.26. (Frenzel 2022)

$$\kappa_y = \frac{I_{\bar{z}\bar{z}}^2}{A \int_{\bar{y}_o}^{\bar{y}_u} \left(\frac{S_{\bar{z}}^2}{b(\bar{y})} \right) d\bar{y}} \quad \kappa_z = \frac{I_{\bar{y}\bar{y}}^2}{A \int_{\bar{z}_o}^{\bar{z}_u} \left(\frac{S_{\bar{y}}^2}{b(\bar{z})} \right) d\bar{z}} \quad (\text{Gleichung 2.26})$$

Eine Vereinfachung für einen dünnwandigen Querschnitt aus beliebig angeordneten rechteckigen Elementen liefert Gleichung 2.27. Dabei sind $A_{qy,i}$ und $A_{qz,i}$ die Schubflächen eines i -ten Elements.

$$A_{qy} = \sum_{i=1}^{N_m} A_{qy,i} = \sum_{i=1}^{N_m} t_i l_i |\cos \alpha_i| \quad (\text{Gleichung 2.27})$$

$$A_{qz} = \sum_{i=1}^{N_m} A_{qz,i} = \sum_{i=1}^{N_m} t_i l_i |\sin \alpha_i|$$

Die Ausdrücke $\cos \alpha_i$ und $\sin \alpha_i$ beschreiben Gleichung 2.28 und Gleichung 2.29 mithilfe der Koordinaten von Anfangs- und Endknoten $k_{a,i}$ ($\bar{y}_{a,i}$, $\bar{z}_{a,i}$) und $k_{e,i}$ ($\bar{y}_{e,i}$, $\bar{z}_{e,i}$) eines i -ten Elements.

$$\cos \alpha_i = \frac{\bar{y}_{e,i} - \bar{y}_{a,i}}{l_i} \quad (\text{Gleichung 2.28})$$

$$\sin \alpha_i = \frac{\bar{z}_{e,i} - \bar{z}_{a,i}}{l_i} \quad (\text{Gleichung 2.29})$$

(Stanoev 2016)

2.2 Saint-Venantsche Torsion am dünnwandigen Querschnitt

2.2.1 Bedingungen und Ziele der Saint-Venantschen Torsion

Ziel ist die Ermittlung der Torsionssteifigkeit $G I_T$ und der Verdrehung φ_x des Querschnitts. Dazu dienen die Berechnungen des Torsionsträgheitsmomentes I_T und der Verdrillung φ'_x sowie die Festlegung der Materialeigenschaften mit dem Schubmodul G . (Linke 2015), S. 96

Die Theorie der Saint-Venantschen Torsion gilt, wenn keine Wölbspansungen auftreten. Das ist der Fall, wenn ein wölbfreier Querschnitt vorliegt oder wenn die Verwölbung des Balkens in Richtung der x -Achse vollständig stattfindet (z.B. durch eine Gabellagerung). (Linke 2015), S. 89 / (Stanoev 2013), S. 10, S. 22

Als weitere Bedingung treten in Umfangsrichtung s keine Normalspannungen auf. (Linke 2015), S. 100

Abbildung 5 zeigt einen geschlossenen einzelligen Querschnitt mit dem SMP M , der Umfangrichtung/Umlaufrichtung bzw. Umlaufkordinate s , dem Polstrahl $r(s)$, der Dicke $t(s)$ und der Profilmittellinie. Der Polstrahl $r(s)$ liegt zwischen SMP und Profilmittellinie.

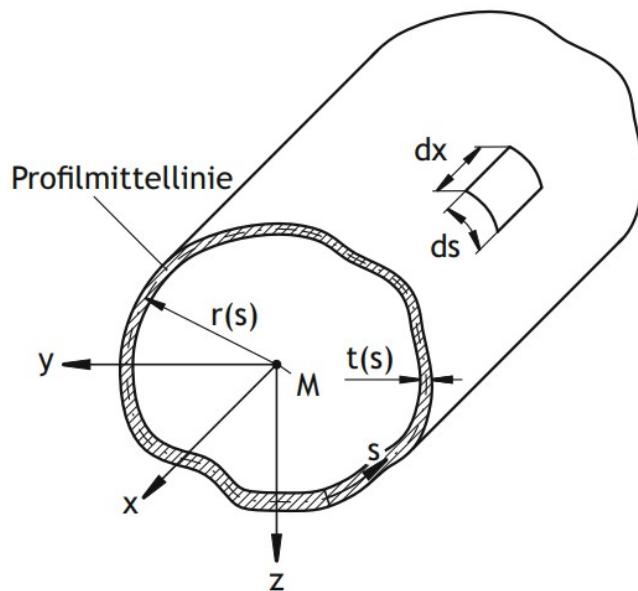


Abbildung 5: Einzelliger Querschnitt mit Polstrahl $r(s)$ KS (x, y, z) im SMP M (Linke 2015), S. 100

Eine Kategorisierung der Querschnitte erfolgt in einzellige und mehrzellige Querschnitte. Die Torsion von offenen und geschlossenen Profilteilen weist zudem Unterschiede auf. Offene Profilteile unterliegen einer separaten Betrachtung. (Linke 2015), S. 99, S.105 / (Stanoev 2013), S. 11

Es gibt einige offensichtliche Fälle von Wölf freiheit bei dünnwandigen Querschnitten. Wölf freiheit tritt auf, wenn alle Elemente des Querschnitts den gleichen Polstrahl r_{tM} aufweisen, siehe Abbildung 6. (Linke 2015), S. 126

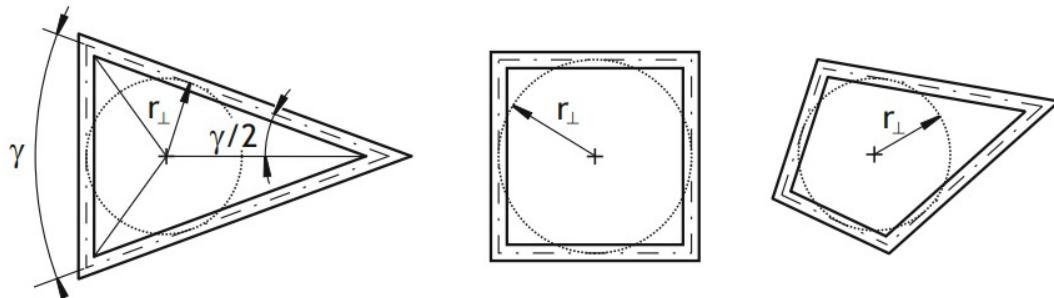


Abbildung 6: Beispiele für wölf freie dünnwandige geschlossene Querschnitte mit einheitlichem Polstrahl r_{\perp} (Linke 2015), S. 126

Offene Querschnitte sind wölf frei, wenn sich die Profilmittellinien aller Elemente im SMP des Querschnitts schneiden oder treffen, siehe Abbildung 7. Der Polstrahl $r_{tM,i}$ der Elemente bezogen auf den SMP ist 0. (Linke 2015), S. 132

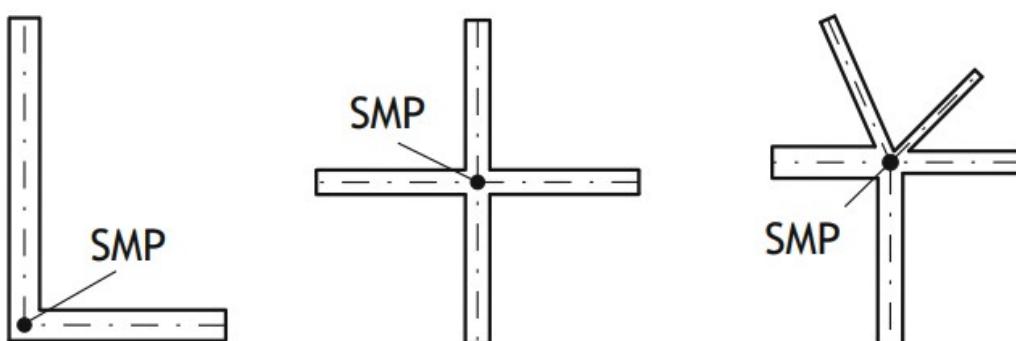


Abbildung 7: Beispiele für wölf freie dünnwandige offene Querschnitte mit SMP im Schnittpunkt aller Profilmittellinien (Linke 2015), S. 132

2.2.2 Erweiterung des Modells

Zur Vereinfachung der nachfolgenden Berechnungen geht die Wirkungslinie des Torsionsmoments T durch den SMP M , siehe Abbildung 8. Die Querkräfte Q_y und Q_z sind gleich den äußeren Kräften F_y und F_z . Das Torsionsmoment M_x um die x -Achse des FSP-KS entsteht aus diesen mit den Hebelarmen a und b , siehe Gleichung 2.30.

$$M_x = -b F_y - a F_z \quad Q_y = F_y \quad Q_z = F_z \quad (\text{Gleichung 2.30})$$

Die Berechnung des Torsionsmoments T benötigt weiterhin die Koordinaten (e_y , e_z) des SMP im FSP-KS. Die Formelzeichen (a_y , a_z) bzw. (y_M , z_M) - wie im Weiteren verwendet - sind ebenfalls möglich. Gleichung 2.31 fasst die Berechnung von T aus den genannten Größen zusammen.

$$T = M_x + Q_y e_z - Q_z e_y = M_x + Q_y z_M - Q_z y_M \quad (\text{Gleichung 2.31})$$

(Linke 2015), S. 90-91

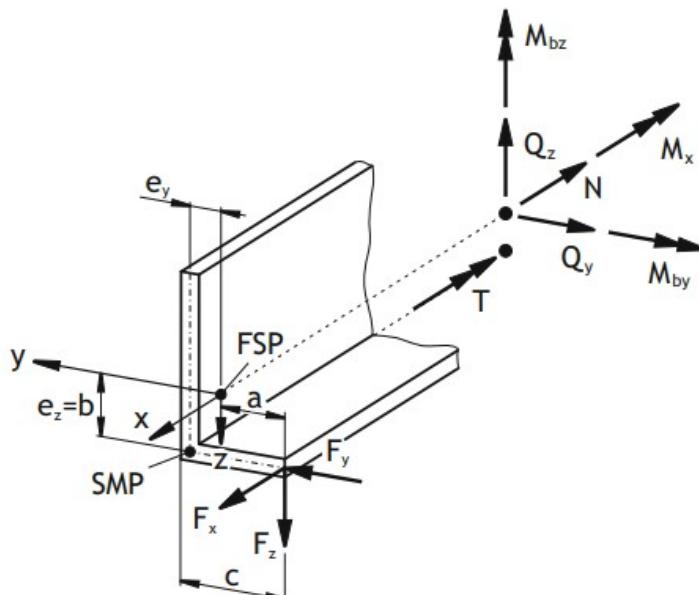


Abbildung 8: Belastungen am Balken mit Bezug zu FSP und SMP im betrachteten Querschnitt am Beispiel eines L-Profil (Linke 2015), S. 91

Abbildung 9 zeigt die erforderlichen Größen für die Berechnung von Torsioneigenschaften des Querschnitts. Die bereits aus Unterkapitel 2.1.1 bekannten Größen am i -ten Element sind die Länge l_i , Dicke t_i , Anfangsknoten $k_{a,i}$ mit $(\bar{y}_{a,i}, \bar{z}_{a,i})$, Endknoten $k_{e,i}$ mit $(\bar{y}_{e,i}, \bar{z}_{e,i})$, Elementfläche A_i und Steigung α_i . Der Schubmittelpunkt (SMP) M des Gesamtquerschnitts hat die Koordinaten (y_M, z_M) . Die Berechnung erfolgt in Unterkapitel 2.3.4. In Umfangrichtung s auf der

Profilmittellinie erscheint die Koordinate s . Die Abbildung zeigt einen offenen Profilteil aus einem Element und einen geschlossenen Profilteil aus 5 Elementen. Zur Veranschaulichung ist die Dicke t_i der Elemente, besonders des beschrifteten Elements, sehr groß gewählt im Verhältnis zur Länge. Die Abbildung stellt dennoch schematisch einen dünnwandigen Querschnitt dar.

Der Polstrahl $r_{tM,i}$ des i -ten Elements liegt senkrecht zur Profilmittellinie. Er beschreibt den Abstand zwischen Profilmittellinie und SMP M . (Linke 2015), S. 100

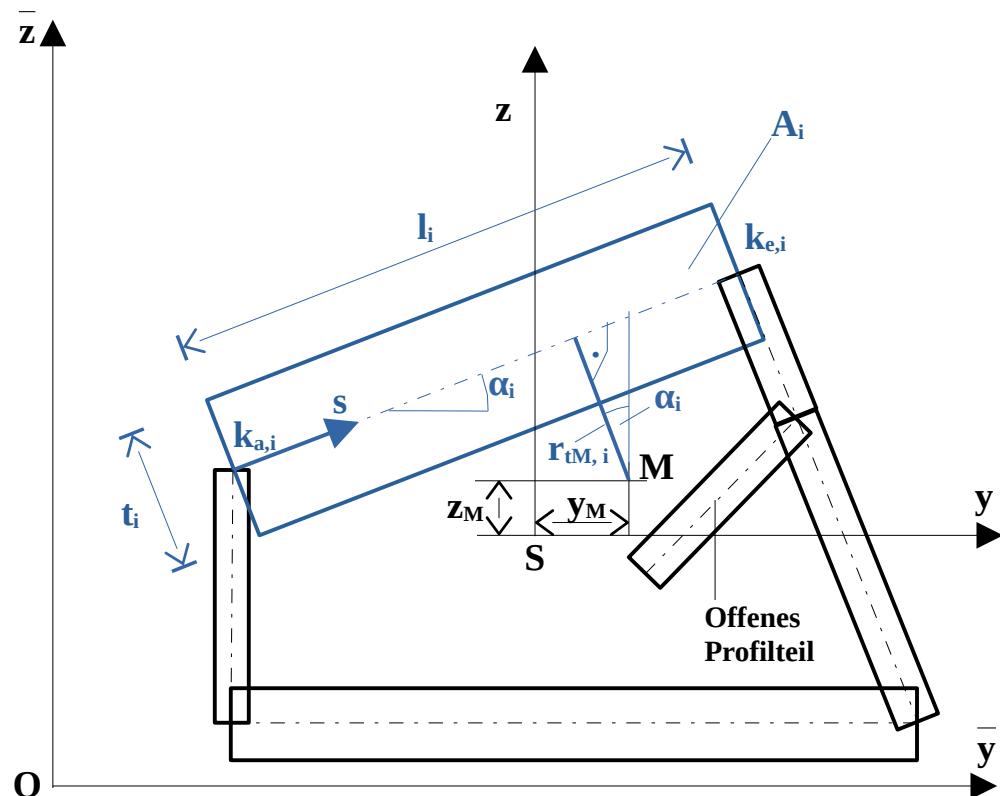


Abbildung 9: Beispiel-Querschnitt aus Rechteck-Elementen mit einzellig geschlossenen Profilteilen und einem offenen Profilteil (Stanoev 2016)

(Stanoev 2013), S.12 / (Stanoev 2016)

2.2.3 Dünnwandiger geschlossener einzelliger Querschnitt

Die Schubspannungen sind bei dünnwandigen geschlossenen Querschnitten – idealisiert betrachtet – konstant über die Dicke t der Elemente (für $t(s) \ll r(s)$). Sie verlaufen in Richtung der Profilmittellinie (bei kleinen Änderungen von $t(s)$). (Linke 2015), S.99

Abbildung 10 zeigt eine genaue Schubspannungsverteilung an der Stelle s des Querschnitts. Hierbei ist die Schubspannung τ_{xs} nach Gleichung 2.32 die Summe aus τ_{xs}^0 und τ_{xs}^1 . Vereinfacht ist $\tau(s) = \tau_{xs}^0$, und τ_{xs}^1 spielt keine Rolle.

$$\tau_{xs} = \tau_{xs}^0 + \tau_{xs}^1 \quad \tau(s) \approx \tau_{xs}^0 \quad (\text{Gleichung 2.32})$$

(Stanoev 2013), S. 10

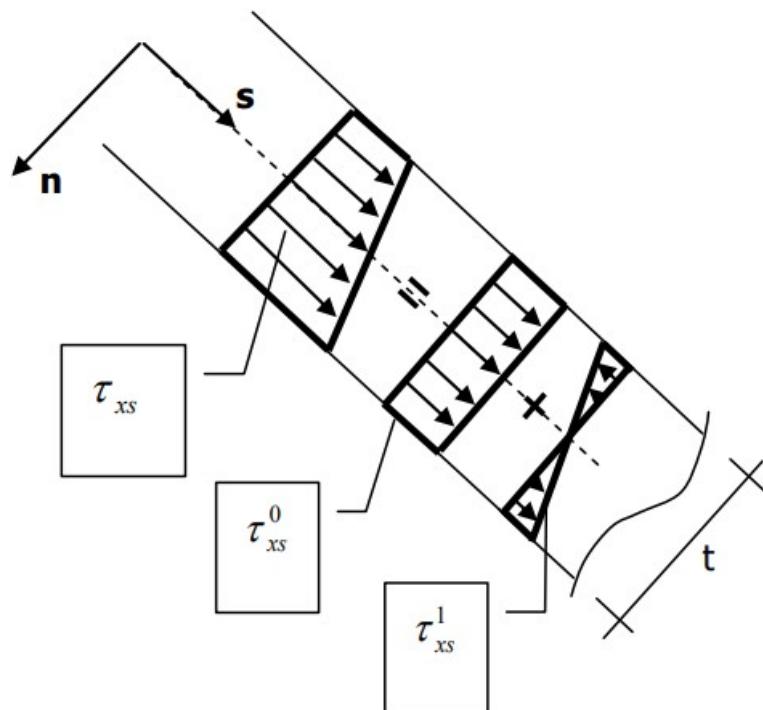


Abbildung 10: Zusammensetzung der Schubspannung im Querschnitt an der Stelle s (Stanoev 2013), S. 10

Der Schubfluss (Bredtsche Schubfluss) T_s „[...] stellt die über der Dicke aufsummierten Schubspannungen dar.“ Das verdeutlicht Gleichung 2.33. Er ist an allen Stellen des Querschnitts konstant.

$$T_s = \tau(s) t(s) \quad (\text{Gleichung 2.33})$$

(Linke 2015), S. 100

Abbildung 11 veranschaulicht weitere geometrische und mechanische Zusammenhänge an einem (hier infinitesimalen) Element des Querschnitts. Die aus der Torsion T resultierende Elementkraft dF hängt linear von der Länge ds des Elements in Umfangsrichtung s ab, siehe Gleichung 2.34. dF wirkt in Umfangsrichtung s des Querschnitts. Das Elementtorsionsmoment dT steht im linearen Zusammenhang mit der Länge des Polstrahls $r_{tM,i}$ des Elements, siehe Gleichung 2.35. Das Torsionsmoment T des Gesamtquerschnitts ist ein Umlaufintegral bzw. Umfangsintegral über alle Elemente, siehe Gleichung 2.36.

$$dF = T_s ds \quad (\text{Gleichung 2.34})$$

$$dT = r_{tM} dF = r_{tM} T_s ds \quad (\text{Gleichung 2.35})$$

$$T = \oint dT = T_s \oint r_{tM} ds \quad (\text{Gleichung 2.36})$$

Die vom Element eingeschlossene Fläche dA_s ist ein Dreieck, siehe Gleichung 2.37. Die von der Profilmittellinie eingeschlossene Fläche A_s ist in der Berechnung ein Umlaufintegral, siehe Gleichung 2.38.

$$dA_s = \frac{1}{2} r_{tM} ds \quad (\text{Gleichung 2.37})$$

$$A_s = \oint dA_s = \frac{1}{2} \oint r_{tM} ds \quad (\text{Gleichung 2.38})$$

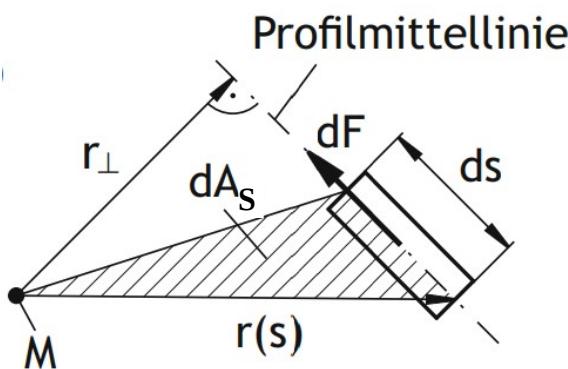


Abbildung 11: Infinitesimales Element der Umfangslänge ds mit geometrischen Größen (bezogen auf SMP M) und Kraft dF (Linke 2015), S. 100, bearb.

Wenn der SMP M und/oder die Länge des Polstrahls r_{tM} unbekannt sind, ist die Gleichung 2.39 eine Möglichkeit zur Ermittlung von A_s aus den Knotenpunkt-Koordinaten der N_m Elemente. Die Koordinaten des Anfangsknotens $k_{a,i}$ eines Elementes i sind $(\bar{y}_{a,i}, \bar{z}_{a,i})$ und die des Endknotens $k_{e,i}$ sind $(\bar{y}_{e,i}, \bar{z}_{e,i})$.

$$A_S = \frac{1}{2} \sum_{i=1}^{N_m} (\bar{y}_{a,i} \bar{z}_{e,i} - \bar{y}_{e,i} \bar{z}_{a,i}) \quad (\text{Gleichung 2.39})$$

(Stanoev k.A.), S. 2

Es besteht für das Torsionsmoment T aus Gleichung 2.36 mithilfe von Gleichung 2.38 der in Gleichung 2.40 beschriebene Zusammenhang.

$$T = 2 A_S T_s \quad (\text{Gleichung 2.40})$$

Durch Substituieren des Schubflusses T_s in Gleichung 2.40 mit Gleichung 2.33 und Umstellen entsteht Gleichung 2.41. Das ist die *1. Bredtsche Formel*. Die maximale Schubspannung τ_{\max} tritt bei der Umlaufkoordinate s mit der kleinsten Dicke t_{\min} auf. So ist die Berechnung des Torsionswiderstandsmoments W_T möglich, siehe Gleichung 2.42.

$$\tau(s) = \frac{T}{2 A_S t(s)} \quad (\text{Gleichung 2.41})$$

$$\tau_{\max} = \frac{T}{2 A_S t_{\min}} = \frac{T}{W_T} \quad W_T = 2 A_S t_{\min} \quad (\text{Gleichung 2.42})$$

Mit diesen Zwischenschritten ist jetzt die Berechnung des Torsionsträgheitsmomentes I_T des Querschnitts möglich (*2. Bredtsche Formel*), siehe Gleichung 2.43. Daraus entsteht Gleichung 2.44 für die Verdrillung φ'_x des Querschnitts. Gleichung 2.45 definiert die zusätzliche Verdrehung $d\varphi_x$ für einen Abschnitt bzw. ein Feld des Balkens der Länge dx mit konstantem Torsionsmoment T , konstanter Torsionssteifigkeit $G I_T$ und gleicher x -Achse.

$$I_T = \frac{4 A_S^2}{\oint \frac{ds}{t(s)}} \quad (\text{Gleichung 2.43})$$

$$\varphi'_x = \frac{T}{G I_T} \quad (\text{Gleichung 2.44})$$

$$d\varphi_x = \varphi'_x dx \quad (\text{Gleichung 2.45})$$

(Linke 2015), S. 100-105 / (Petersen 2013), S. 140-141

Eine wichtige geometrische Größe ist der bezogene Schubfluss Φ (Torsionsfunktion, konstanter bezogener Schubfluss), siehe Gleichung 2.46. Einsetzen von Φ in Gleichung 2.43 (oder Umstellen von Gleichung 2.46) ergibt für das Torsionsträgheitsmoment I_T die Gleichung 2.47.

$$\Phi = \frac{2 A_S}{\oint \frac{ds}{t(s)}} = \frac{I_T}{2 A_S} = \frac{T_s}{G \varphi'_x} \quad (\text{Gleichung 2.46})$$

$$I_T = 2 \Phi A_S = \Phi^2 \oint \frac{ds}{t(s)} = \Phi^2 \sum_{i=1}^{N_m} \frac{l_i}{t_i} \quad (\text{Gleichung 2.47})$$

(Stanoev 2013). S. 16-18, S. 21 / (Mittelstedt 2021), S. 263

2.2.4 Dünnwandiger geschlossener mehrzelliger Querschnitt

Mithilfe des Superpositionsprinzip ist eine Aufteilung des Torsionsmoments T in die Torsionsmomente T_p der jeweils p -ten Zelle des Querschnitts möglich, siehe Gleichung 2.48. T_p verursacht in der p -ten Zelle den Schubfluss T_{Sp} mit der von der Profilmittellinie eingeschlossenen Fläche der Zelle A_{Sp} , siehe Gleichung 2.49. Die Zellenanzahl ist dabei N_c .

$$T = \sum_{p=1}^{N_c} T_p \quad (\text{Gleichung 2.48})$$

$$T_p = 2 A_{Sp} T_{Sp} \quad (\text{Gleichung 2.49})$$

Zur Veranschaulichung dient der in Abbildung 12 gezeigte zweizellige Querschnitt. In der Abbildung steht q_p für T_{Sp} . a bzw. $2a$ sind beispielhafte Maße für die Seitenlängen der Zellen. Die Zellen haben jeweils eine eigene Umfangsrichtung bzw. Umlaufkoordinate s_p . Die Zellenanzahl N_c beträgt hier 2.

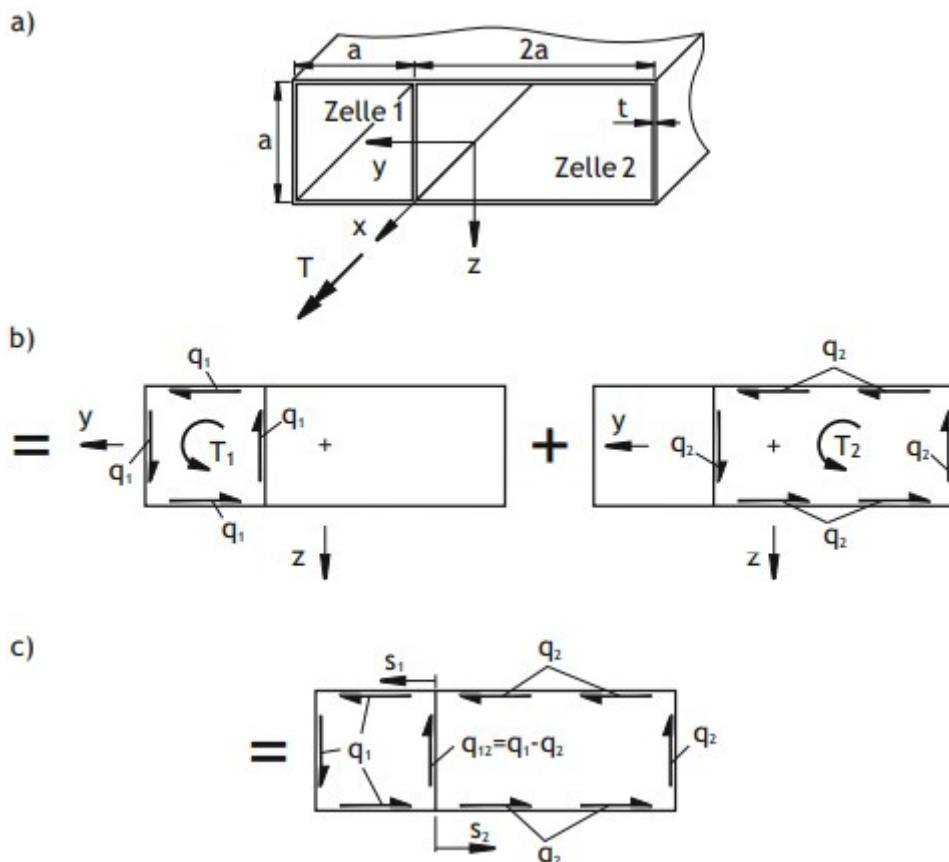


Abbildung 12: Dünnwandiger geschlossener zweizelliger Querschnitt mit Torsionsmomenten und Schubflüssen der Zellen (Linke 2015), S. 106

Dabei überlagern sich die Schubflüsse T_{Sp} in den zu zwei Zellen gehörigen Elementen bzw. Profilteilen. Der Drehsinn der Torsionsmomente T_p der Zellen ist im Vorzeichen zu beachten. Er wird als in Umfangrichtung s_p positiv definiert. So ergibt der in Gleichung 2.50 beschriebene allgemeine Zusammenhang für das in Abbildung 12 gezeigte Beispiel die Gleichung 2.51. Der Schubfluss T_{S2} zeigt hier für die Berechnung von T_{S12} ebenfalls in Umfangsrichtung s_1 .

$$T_{Spq} = T_{Sp} + T_{Sq} \quad (\text{Gleichung 2.50})$$

$$T_{S12} = T_{S1} + (-T_{S2}) \quad (\text{Gleichung 2.51})$$

Die Querschnittsform bleibt bei einer Verdrehung φ_x erhalten. Daher ist die Verdrehung φ_{xi} und die Verdrillung φ'_{xi} in allen Zellen gleich den Werten im gesamten Querschnitt, siehe Gleichung 2.52 und Gleichung 2.53. Diese Annahme ist ersichtlich in Abbildung 13. Hier steht ϑ für φ_x und ϑ_p für φ_{xp} . M und M_p sind SMP des Gesamtquerschnitts bzw. einer p -ten Zelle.

$$\varphi_x = \varphi_{x1} = \varphi_{xp} = \dots = \varphi_{xNc} \quad (\text{Gleichung 2.52})$$

$$\varphi'_{x1} = \varphi'_{xp} = \dots = \varphi'_{xNc} \quad (\text{Gleichung 2.53})$$

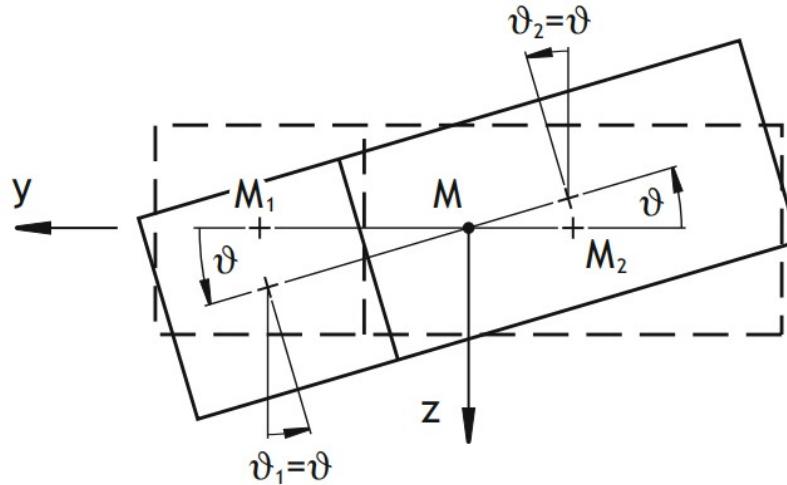


Abbildung 13: Dünnwandiger geschlossener zweizelliger Querschnitt mit Verdrehung um SMP (Linke 2015), S. 108

Gleichung 2.54 definiert die Verdrillung φ'_{xp} der p -ten Zelle in Abhängigkeit der Geometrie (A_{Sp} , $t(s)$, ds), des Werkstoffs (G) und der Belastung (T_{Sp} , T_{Sq} aus T_p und A_{Sp} , siehe Gleichung 2.48 und Gleichung 2.49). Die Summe umfasst alle gemeinsamen Elemente der p -ten Zelle mit der bzw. den angrenzenden Zelle(n) mit der laufenden Nummer p . Diese reduziert das Umlaufintegral über alle Elemente der p -ten Zelle in der Klammer. Die Gleichung gilt für gegenläufige Umfangsrichtungen s_p und s_q im gemeinsamen Querschnittsteil s_{pq} .

$$\varphi'_{xp} = \frac{1}{2GA_{Sp}} \left(\oint_p \frac{T_{Sp}}{t(s)} ds - \sum_{q \neq p} \int_{pq} \frac{T_{Sq}}{t(s)} ds \right) \quad (\text{Gleichung 2.54})$$

Für N_c Zellen entstehen mit Gleichung 2.54 und Gleichung 2.49, eingesetzt in Gleichung 2.48, $N_c + 1$ Gleichungen. Demgegenüber stehen ebenfalls $N_c + 1$ Unbekannte (Verdrillung φ'_x und N_c Schubflüsse T_{Sp}). Dieses lineare Gleichungssystem ist eindeutig lösbar. Das Torsionsträgheitsmoment I_T bzw. die Torsionsteifigkeit GI_T des Querschnitts dienen als zusätzliche Möglichkeit zur Berechnung der Verdrillung φ'_x , siehe Gleichung 2.44.

(Linke 2015), S. 105-110

Über den bezogenen Schubfluss Φ_p der p -ten Zelle ist die Berechnung des Torsionsträgheitsmoments I_{Tp} der Zelle möglich. Abbildung 14 zeigt drei Zellen eines dünnwandigen geschlossenen Querschnitts. Die Zellen sind für die Berechnung gedanklich aufgetrennt. So entsteht ein einziger offener Querschnitt. Die Nummern der Zellen sind in diesem Beispiel $p - 1$, p und $p + 1$. Die Pfeile zeigen die jeweilige Umfangsrichtung s_p , s_{p-1} und s_{p+1} an.

Die Berechnung des bezogenen Schubflusses Φ_p der p -ten Zelle ist damit nach Gleichung 2.55 möglich. Für jede Zelle entsteht eine Gleichung. Bei N_c Zellen ist das ein lineares Gleichungssystem mit N_c Gleichungen und N_c Unbekannten (den N_c bezogenen Schubflüssen Φ_p der Zellen).

$$\Phi_p \oint_p \frac{ds}{t(s)} \pm \sum_{q=1, q \neq p}^{N_c-1} \Phi_q \int_{pq} \frac{ds}{t(s)} = 2A_{Sp} \quad (\text{Gleichung 2.55})$$

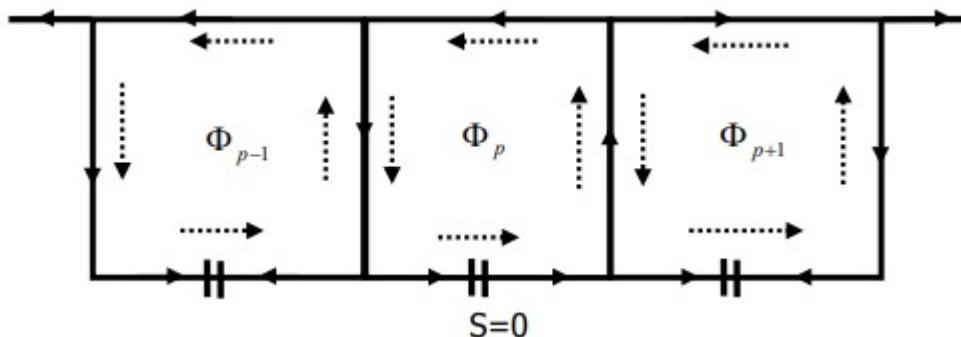


Abbildung 14: Bezugene Schubflüsse in den Zellen eines dünnwandigen geschlossenen mehrzelligen Querschnitts mit gedachten Schnitten (Stanoev 2013), S. 17

Aus den Betrachtungen für dünnwandige geschlossene einzellige Querschnitte mit Gleichung 2.43, Gleichung 2.46 und Gleichung 2.47 folgt die Berechnung von I_T für dünnwandige geschlossene mehrzellige Querschnitte. Das Torsionsmoment I_T des Gesamtquerschnitts besteht aus einer additiven Zusammenfassung der Torsionsmomente I_{Tp} der Teilquerschnitte bzw. wie hier der N_c Zellen, siehe Gleichung 2.56.

$$I_T = \sum_{p=1}^{N_c} I_{Tp} = 2 \sum_{p=1}^{N_c} \Phi_p A_{Sp} = \sum_{p=1}^{N_c} \Phi_p^2 \oint \frac{ds}{t(s)} \quad (\text{Gleichung 2.56})$$

(Stanoev 2013), S. 17 / (Mittelstedt 2021), S. 269-271

2.2.5 Dünnwandige offene Querschnittsteile

Die gedankliche Auftrennung an einer beliebigen Stelle führt bei einem offenen Querschnittsteil zur Entstehung von zwei neuen Teilquerschnitten. Die aus einer Torsionsbelastung resultierende Schubspannungsverteilung ist linear über die Dicke t_i der Elemente verteilt. Die Schubspannungs-Maxima liegen in den Randfasern der Elemente. Die Profilmittellinie ist schubspannungsfrei, vergleiche mit τ_{xs}^1 in Abbildung 10 aus Unterkapitel 2.2.3.

Das Torsionsträgheitsmoment $I_{T,offen}$ der offenen Querschnittsteile ist die Summe der Torsionsträgheitsmomente $I_{T,offen,i}$ der offenen Elemente. Für kombinierte Querschnitte aus offenen und geschlossenen Querschnittsteilen gilt die Berechnung für alle N_m Elemente, da der lineare Schubspannungsverlauf in der Berechnung für dünnwandige geschlossene Querschnitte nicht enthalten ist. Das Ergebnis wird mit dem Torsionsträgheitsmoment I_T der geschlossenen Querschnittsteile addiert. Daher folgt Gleichung 2.57 für beliebige dünnwandige Querschnitte.

$$I_{T,offen} = \sum_{i=1}^{N_m} I_{T,offen,i} = \frac{1}{3} \sum_{i=1}^{N_m} l_i t_i^3 \quad (\text{Gleichung 2.57})$$

Die maximale Schubspannung aus dem linearen Schubspannungsanteil $\tau_{max,offen}$ tritt im Element mit der maximalen Dicke t_{max} in der Randfaser auf, siehe Gleichung 2.58. Darin steht als Option das Torsionswiderstandsmoment $W_{T,offen}$.

$$\tau_{max,offen} = \frac{T}{I_{T,offen}} t_{max} = \frac{T}{W_{T,offen}} \quad W_{T,offen} = \frac{I_{T,offen}}{t_{max}} \quad (\text{Gleichung 2.58})$$

Im Vergleich zu einem dünnwandigen geschlossenen Querschnitt hat ein geometrisch ansonsten identischer dünnwandiger offener Querschnitt ein deutlich geringeres Torsionsträgheitsmoment I_T (bei $t(s) \ll r(s)$). Dafür ist die maximale Schubspannung τ_{max} deutlich größer. Das verdeutlichen Gleichung 2.59 und Gleichung 2.60 am Beispiel eines Kreisringquerschnitts mit Radius r .

$$\frac{I_{T,geschlossen}}{I_{T,offen}} = 3 \left(\frac{r}{t} \right)^2 \quad (\text{Gleichung 2.59})$$

$$\frac{\tau_{max,geschlossen}}{\tau_{max,offen}} = \frac{t}{3r} \quad (\text{Gleichung 2.60})$$

2.3 Wölbkrafttorsion am dünnwandigen Querschnitt

2.3.1 Bedingungen und Ziele der Wölbkrafttorsion

Die Wölbkrafttorsion heißt auch Biegetorsion 1. Ordnung. Sie dient als Erweiterung der Saint-Venantschen Torsion. Ein Ebenbleiben des Querschnitts bzw. eine freie Verwölbung – wie bei der Saint-Venantschen Torsion angenommen – sind nicht gegeben. „Praxisrelevante Querschnitte sind i. Allg. nicht wölf frei, so dass aufgrund der Belastung und/oder Lagerungsbedingungen Verwölbungsbehinderungen und entsprechende zusätzliche Spannungszustände hervorgerufen werden können.“ Querschnitte, die keine Wölbkrafttorsion aufweisen, sind in Unterkapitel 2.2.1 beschrieben, vgl. insbesondere Abbildung 6 und Abbildung 7. Für sie gilt ausschließlich die Theorie der Saint-Venantschen Torsion. (Mittelstedt 2021), S. 289

Ziel ist die Ermittlung des Wölbträgheitsmoments $I_{\omega\omega M}$ ($I_{\omega\omega}$, bezogen auf SMP M , Wölbflächenmoment 2. Ordnung) bzw. der Verwölbungssteifigkeit $EI_{\omega\omega M}$ ($EI_{\omega\omega}$) als Querschnittswert. Damit ist die Berechnung von zusätzlich im Querschnitt auftretender sekundärer Torsion M_{xs} (Wölbkrafttorsion) bzw. eines Wölbmoments M_{ω} (Wölbmoment, Bimoment) möglich, siehe Abbildung 15. Daraus resultieren zusätzliche Beanspruchungen in Form von Wölbnormalspannungen $\sigma_{\omega\omega}$ (als Biegespannungen) und sekundären Schubspannungen $\tau_{s,s}(s)$ im Querschnitt. (Linke 2015), S. 143 / (Mittelstedt 2021), S. 290-291, S. 308-309 / (Nasdala 2015), S. 154

$$\begin{pmatrix} N_x \\ M_y \\ -M_z \\ M_{\omega} \end{pmatrix} = E \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & I_{yy} & 0 & 0 \\ 0 & 0 & I_{zz} & 0 \\ 0 & 0 & 0 & I_{\omega\omega} \end{pmatrix} \begin{pmatrix} u' \\ -w'' \\ -v'' \\ -\vartheta'' \end{pmatrix}$$

$$M_{\omega} = -EI_{\omega\omega}\vartheta''$$

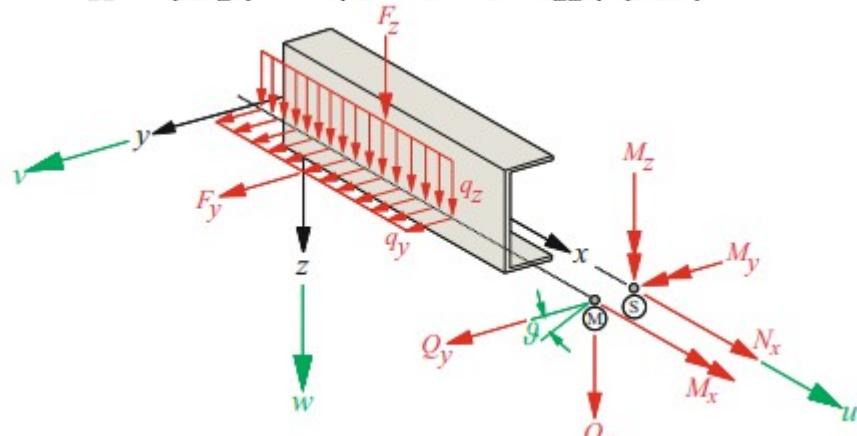


Abbildung 15: Entkoppelte Schnittgrößen im HTA-KS eines verwölbten Querschnitts; Beispiel eines C-Trägers mit Wirkungslinien von Kräften und Momenten in M bzw. S (Mittelstedt 2021), S. 308-309, bearb.

Wölbnormalspannungen $\sigma_{\omega\omega}$ haben allgemein eine behinderte Verwölbung ω eines Querschnitts als Ursache. „Das Entstehen von Wölbspannungen als Folge einer behinderten Verwölbung tritt jedoch nicht nur an Einspannstellen auf, sondern kann auch die Folge von Krafteinleitungen oder starken Änderungen des Querschnitts sein.“ (Mittelstedt 2021), S. 290 Eine ungehinderte (freie) Verwölbung eines Endquerschnitts bei Torsion ist mithilfe einer Gabellagerung möglich, siehe Abbildung 16 (vgl. Abbildung 46 in Anhang 8.1.2). (Stanoev 2013), S. 22

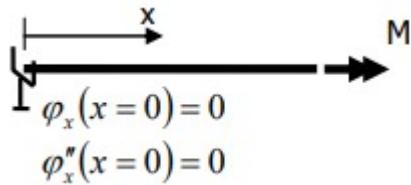


Abbildung 16: Schematische Darstellung von Gabellagerung eines tordierten Stabes (Stanoev 2013), S. 22

Im Beispiel eines einseitig eingespannten I-Trägers ist ein äußeres Torsionsmoment M_T gleich der Schnittgröße M_x in allen Querschnitten. Zwei Kräfte F in den Flanschen stellen das Torsionsmoment M_x dar. Sie verursachen eine entgegengesetzte Ausbiegung v und Längsverschiebung u der Flansche, siehe Abbildung 17. Weiterhin kommt es zu einer Verdrehung φ_x um die SMP-Achse (Drillruheachse). Im Querschnitt der Einspannung ist die Verwölbung vollständig behindert und verursacht Wölbnormalspannungen $\sigma_{\omega\omega}$. Am freien Ende hingegen führt freie Verwölbung des Querschnitts zu den genannten Verformungen.

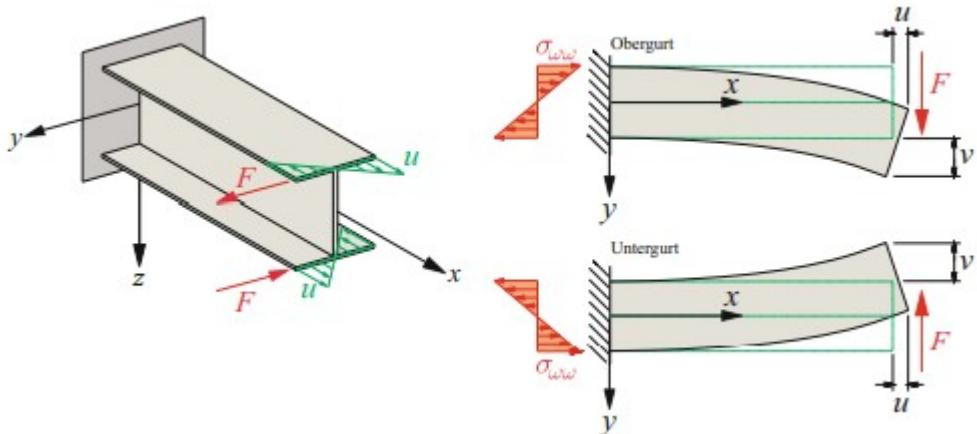


Abbildung 17: Verwölbung und Wölbnormalspannungen am Beispiel eines einseitig eingespannten I-Trägers mit Torsionsbelastung (Mittelstedt 2021), S. 290

Allgemein gilt eine Aufteilung in primäre Torsion M_{xp} (Saint-Venantsche Torsion) und sekundäre Torsion M_{xs} (Wölbkrafttorsion), siehe Gleichung 2.61.

$$M_x = M_{xp} + M_{xs} \quad (\text{Gleichung 2.61})$$

Gleiches gilt für die daraus resultierenden primären Schubspannungen $\tau_{s,p}(s)$ und sekundären Schubspannungen $\tau_{s,s}(s)$. $\tau_{s,p}(s)$ ist linear über die Dicke des Flansches verteilt (vgl. Abbildung 10 in Unterkapitel 2.2.3 mit $\tau^1_{xs}(s)$ und $\tau^0_{xs}(s)$). $\tau_{s,s}(s)$ verläuft konstant über die Dicke und quadratisch über die Länge des Flansches, siehe Abbildung 18 .

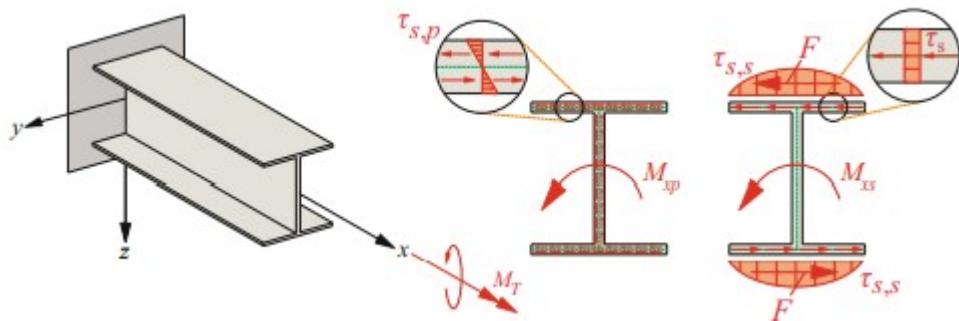


Abbildung 18: Primäre und sekundäre Schubspannungen am Beispiel eines einseitig eingespannten I-Trägers mit Torsionsbelastung (Mittelstedt 2021), S. 291

Im Beispiel sind entgegengesetzte Flanschbiegemomente M_{Fl} eine Erklärung für Wölbnormalspannungen $\sigma_{\omega\omega}$ mit linearem Verlauf über die Länge der Flansche, siehe Abbildung 19. Sie entstehen durch das Torsionsmoment M_T .

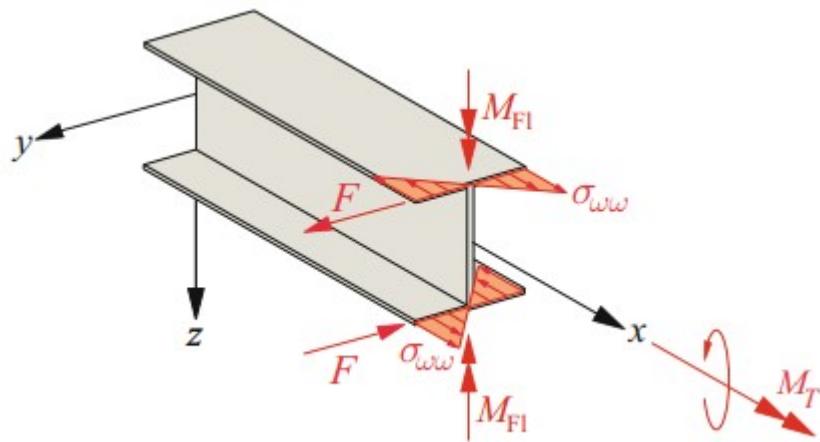


Abbildung 19: Wölbnormalspannungen durch Flanschbiegemomente am Beispiel eines einseitig eingespannten I-Trägers mit Torsionsbelastung (Mittelstedt 2021), S. 291

2.3.2 FE-Lösung der Wölbfunktion

Das FE-Modell für reine Saint-Venantsche Torsion aus Abbildung 9 in Unterkapitel 2.2.2 wird mit einem Polstrahl $r_{tO,i}$ für ein i -tes Element erweitert. Der Polstrahl bezieht sich allgemein auf den Koordinatenursprung eines beliebig gewählten KS. Für eine Berechnung mithilfe trigonometrischer Beziehungen ist die Verwendung des Ursprungs-KS mit O sinnvoll, siehe Gleichung 2.62. Eine vereinfachte Schreibweise liefert Gleichung 2.63.

$$r_{tO,i} = \bar{y}_{a,i} \cos \alpha_i + \bar{z}_{a,i} \sin \alpha_i \quad (\text{Gleichung 2.62})$$

$$r_{tO,i} = (\bar{y}_{a,i}(\bar{z}_{e,i} - \bar{z}_{a,i}) + \bar{z}_{a,i}(\bar{y}_{a,i} - \bar{y}_{e,i})) \frac{1}{l_i}$$

$$r_{tO,i} = (\bar{y}_{a,i} \bar{z}_{e,i} - \bar{y}_{e,i} \bar{z}_{a,i}) \frac{1}{l_i} \quad (\text{Gleichung 2.63})$$

(Stanoev 2013), S. 13, S. 25 / (Stanoev 2016)

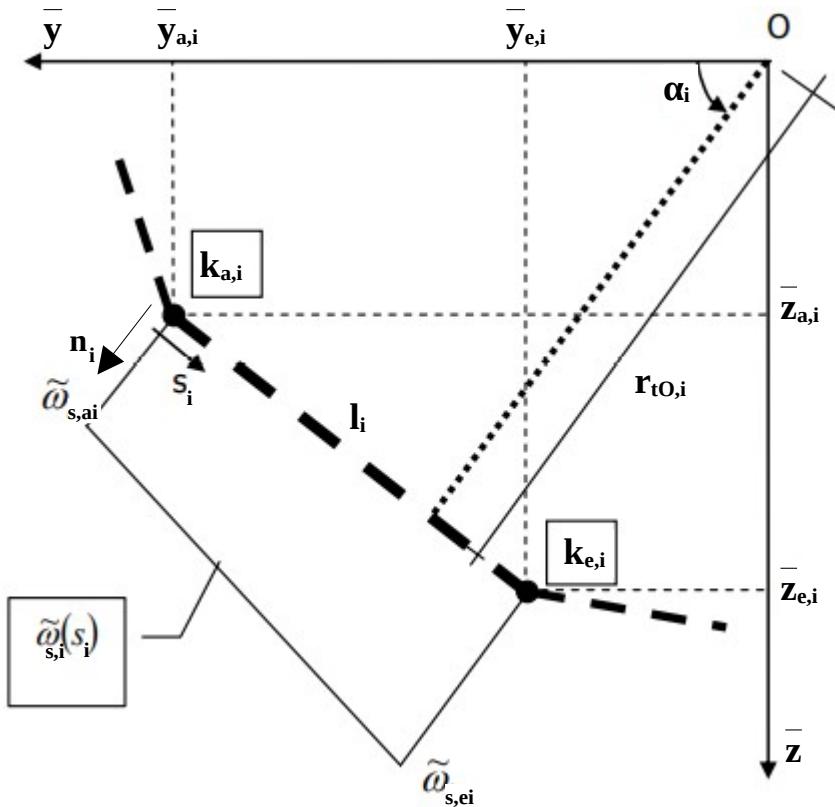


Abbildung 20: Geometrische Größen am Element eines FE-Modells zur Bestimmung der Wölbfunktion (Stanoev 2013), S. 24, bearb.

Mit dem FE-Modell ist die Bestimmung der Wölbfunktion $\omega(s)$ eines dünnwandigen Querschnitts möglich. Sie ist – wie die Schubspannungen – in dünnwandigen Querschnitten näherungsweise konstant über die Dicke. Die

Berechnung unterscheidet nicht zwischen offenen und geschlossenen Querschnitten. Das Prinzip der virtuellen Arbeit für die primären Schubspannungen dient als Ansatz. (Stanoev 2013), S. 13, S. 22 Schubspannungen in Dickenrichtung sind bei dünnwandigen Querschnitten vernachlässigbar. Daher lassen sich primäre Schubspannungen $\tau_{s,p}(s) \approx \tau^0_{xs}(s)$ mit Gleichung 2.64 in die Richtungskomponenten $\tau_{xy}(s)$ und $\tau_{xz}(s)$ zerlegen.

$$\tau_{xy} = \tau_{xs} \frac{\partial y}{\partial s} \quad \tau_{xz} = \tau_{xs} \frac{\partial z}{\partial s} \quad (\text{Gleichung 2.64})$$

Mit den Gleitungen y_{xy} und y_{xz} ist eine virtuelle Arbeit δW an einer infinitesimalen Teilfläche des Querschnitts dA darstellbar, siehe Gleichung 2.65.

$$-\delta W = \int (\delta y_{xy} \tau_{xy} + \delta y_{xz} \tau_{xz}) dA = 0 \quad (\text{Gleichung 2.65})$$

Infinitesimale Gleitungen (δy_{xy} , δy_{xz}) sind näherungsweise gleich den Ableitungen infinitesimaler Verwölbungen ($\delta \tilde{\omega}_{,y}$, $\delta \tilde{\omega}_{,z}$), siehe Gleichung 2.66.

$$\delta y_{xy} \approx \delta \tilde{\omega}_{,y} \quad \delta y_{xz} \approx \delta \tilde{\omega}_{,z} \quad (\text{Gleichung 2.66})$$

$$-\delta W = \int (\delta \tilde{\omega}_{,y} \tau_{xy} + \delta \tilde{\omega}_{,z} \tau_{xz}) dA = 0$$

Die Terme aus Gleichung 2.64 ersetzen die Schubspannungen τ_{xy} und τ_{xz} in Gleichung 2.66, siehe Gleichung 2.67.

$$-\delta W = \int (\delta \tilde{\omega}_{,y} \frac{\partial y}{\partial s} + \delta \tilde{\omega}_{,z} \frac{\partial z}{\partial s}) \tau_{xs} dA = 0 \quad (\text{Gleichung 2.67})$$

Dieser Zusammenhang ergibt im Umlauf-KS mit Koordinaten (s, n) und Ableitung infinitesimaler Verwölbung $\delta \tilde{\omega}_{,s}$ in Umfangsrichtung s die Gleichung 2.68.

$$-\delta W = \int_s \int_n \delta \tilde{\omega}_{,s} \tau_{xs} dn ds = 0 \quad (\text{Gleichung 2.68})$$

Ein äquivalenter Ausdruck (Gleichung 2.69) ersetzt die Schubspannung τ_{xs} , siehe Gleichung 2.70.

$$\tau_{xs} = G \gamma_{xs} = G (-\tilde{\omega}_{,s} + r_t) \varphi'_x \quad (\text{Gleichung 2.69})$$

$$-\delta W = \int_s \int_n \delta \tilde{\omega}_{,s} G (-\tilde{\omega}_{,s} + r_t) \varphi'_x dn ds = 0 \quad (\text{Gleichung 2.70})$$

Für einen Querschnitt aus N_m Elementen mit jeweiliger Dicke t_i und Länge l_i in Umfangsrichtung s_i gilt Gleichung 2.71 (ähnlich zu Gleichung 2.70).

$$\sum_{i=1}^{N_m} t_i \left(\int_0^{l_i} \delta \gamma_{xs,i} \tau_{xs,i} ds \right) = \sum_{i=1}^{N_m} t_i \left(\int_0^{l_i} \delta \tilde{\omega}_{s,i} G_i(\tilde{\omega}_{s,i} - r_{tO,i}) ds_i \right) \varphi'_x \quad \begin{array}{l} \text{(Gleic} \\ \text{hung} \\ \text{2.71)} \end{array}$$

...=0

Ein Umstellen der Gleichung und Einsetzen von $\varphi'_x = 1$ ergibt Gleichung 2.72.

$$\sum_{i=1}^{N_m} t_i \left(\int_0^{l_i} \delta \tilde{\omega}_{s,i} G_i \tilde{\omega}_{s,i} ds_i \right) = \sum_{i=1}^{N_m} t_i \left(\int_0^{l_i} \delta \tilde{\omega}_{s,i} G_i r_{tO,i} ds_i \right) \quad \begin{array}{l} \text{(Gleichu} \\ \text{ng 2.72)} \end{array}$$

(Stanoev 2013), S. 23

Vektor- und Matrixschreibweise ermöglicht die numerische Berechnung von Gleichung 2.72. Dazu ist zunächst die Formulierung von Elementsteifigkeitsmatrix k_i , Elementlastvektor p_i und Elementverwölbungsvektor v_i nötig. Diese ermöglichen die Assemblierung von Systemsteifigkeitsmatrix K , Systemlastvektor P und Systemverwölbungsvektor V . Schließlich folgt die Berechnung der Verwölbung $\tilde{\omega}_{s,k}$ in allen N_k Knoten aus dem Vektor V .

Die als näherungsweise linear angenommene Verwölbungsfunktion eines i -ten Elements $\tilde{\omega}_{s,i}(s_i)$ berechnet sich mithilfe der Verwölbung an Anfangs- und Endknoten ($\tilde{\omega}_{s,ai}$ und $\tilde{\omega}_{s,ei}$), siehe Gleichung 2.73, vgl. Abbildung 20.

$$\tilde{\omega}_{s,i}(s_i) = \tilde{\omega}_{s,ai} + \frac{\tilde{\omega}_{s,ei} - \tilde{\omega}_{s,ai}}{l_i} s_i = \tilde{\omega}_{s,ai} + \tilde{\omega}_{s,i} s_i \quad \begin{array}{l} \text{(Gleichung 2.73)} \end{array}$$

Der Term für die Ableitung der Wölbungsfunktion eines i -ten Elements $\tilde{\omega}_{s,i}$ lässt sich in Vektorschreibweise nach Gleichung 2.74 ausdrücken. Analog dazu folgt Gleichung 2.75 für die Ableitung infinitesimaler Verwölbung am i -ten Element $\delta \tilde{\omega}_{s,i}$.

$$\tilde{\omega}_{s,i} = \frac{\tilde{\omega}_{s,ei} - \tilde{\omega}_{s,ai}}{l_i} = \begin{bmatrix} -\frac{1}{l_i} & \frac{1}{l_i} \end{bmatrix} \cdot \begin{bmatrix} \tilde{\omega}_{s,ai} \\ \tilde{\omega}_{s,ei} \end{bmatrix} = B_i v_i \quad \begin{array}{l} \text{(Gleichung 2.74)} \end{array}$$

$$\delta \tilde{\omega}_{s,i} = \frac{\delta \tilde{\omega}_{s,ei} - \delta \tilde{\omega}_{s,ai}}{l_i} = \begin{bmatrix} -\frac{1}{l_i} & \frac{1}{l_i} \end{bmatrix} \cdot \begin{bmatrix} \delta \tilde{\omega}_{s,ai} \\ \delta \tilde{\omega}_{s,ei} \end{bmatrix} = B_i \delta v_i \quad \begin{array}{l} \text{(Gleichung} \\ \text{2.75)} \end{array}$$

Da die Ableitung der Wölbungsfunktion eines i -ten Elements $\tilde{\omega}_{s,i}$ konstant über die Länge l_i des Elements ist, gilt dies auch für Schubspannung $\tau_{xs,i}$ und Schubfluss $T_{s,i}$ im i -ten Element, siehe Gleichung 2.76 (vgl. Gleichung 2.69) und Gleichung 2.77 (vgl. Gleichung 2.33 in Unterkapitel 2.2.3).

$$\tau_{xs,i} = G_i \gamma_{xs,i} = -G_i (\tilde{\omega}_{s,i} - r_{tO,i}) \varphi'_x \quad (\text{Gleichung 2.76})$$

$$T_{s,i} = \tau_{xs,i} t_i = -G_i t_i (\tilde{\omega}_{s,i} - r_{tO,i}) \varphi'_x \quad (\text{Gleichung 2.77})$$

Eine Rechnung in Einheitsgrößen setzt $\varphi'_x = 1$ vorraus, siehe Gleichung 2.78 mit Einheitsschubspannung $\tilde{\tau}_{xs,i}$ und Gleichung 2.79 mit Einheitsschubfluss $\tilde{T}_{s,i}$.

$$\tilde{\tau}_{xs,i} = -G_i (\tilde{\omega}_{s,i} - r_{tO,i}) = G_i (r_{tO,i} - \underline{B}_i \underline{v}_i) \quad (\text{Gleichung 2.78})$$

$$\tilde{T}_{s,i} = -G_i t_i (\tilde{\omega}_{s,i} - r_{tO,i}) = G_i t_i (r_{tO,i} - \underline{B}_i \underline{v}_i) \quad (\text{Gleichung 2.79})$$

In nachfolgender Gleichung 2.80 für ein i -tes Element ersetzen die Ausdrücke mit \underline{B}_i und \underline{v}_i sowie $\delta \underline{v}_i$ aus Gleichung 2.74 bzw. Gleichung 2.75 die Variablen $\tilde{\omega}_{s,i}$ und $\delta \tilde{\omega}_{s,i}$ in Gleichung 2.72.

$$\delta \underline{v}_i^T \int_0^{l_i} \underline{B}_i^T G_i t_i \underline{B}_i ds_i \underline{v}_i = \delta \underline{v}_i^T \int_0^{l_i} \underline{B}_i^T G_i t_i r_{tO,i} ds_i \quad (\text{Gleichung 2.80})$$

Eine Elementsteifigkeitsmatrix \underline{k}_i ist nach Gleichung 2.81 definiert.

$$\underline{k}_i = \int_0^{l_i} \underline{B}_i^T G_i t_i \underline{B}_i ds_i = \begin{bmatrix} \frac{G_i t_i}{l_i} & \frac{-G_i t_i}{l_i} \\ \frac{-G_i t_i}{l_i} & \frac{G_i t_i}{l_i} \end{bmatrix} \quad (\text{Gleichung 2.81})$$

Ein Elementlastvektor \underline{p}_i ist nach Gleichung 2.82 definiert.

$$\underline{p}_i = \int_0^{l_i} \underline{B}_i^T G_i t_i r_{tO,i} ds_i = \begin{bmatrix} -G_i t_i r_{tO,i} \\ G_i t_i r_{tO,i} \end{bmatrix} \quad (\text{Gleichung 2.82})$$

Damit ist eine weitere Umformung von Gleichung 2.80 für ein i -tes Element durch Zusammenfassen von Termen möglich, siehe Gleichung 2.83.

$$\delta \underline{v}_i^T \underline{k}_i \underline{v}_i = \delta \underline{v}_i^T \underline{p}_i \quad (\text{Gleichung 2.83})$$

Ein Vektor \underline{V} fasst die Werte der Verwölbung $\tilde{\omega}_k$ in allen N_k Knotenpunkten zusammen. Damit lässt sich Gleichung 2.72 für das Gesamtsystem mit Gleichung 2.80 für ein i -tes Element wie in Gleichung 2.84 gezeigt umschreiben.

$$\delta \underline{V}^T \sum_i^{N_m} t_i \left(\int_0^{l_i} \underline{B}_i^T G_i t_i \underline{B}_i ds_i \right) \underline{V} = \delta \underline{V}^T \sum_i^{N_m} t_i \left(\int_0^{l_i} \underline{B}_i^T G_i t_i r_{tO,i} ds_i \right) \quad (\text{Gleichung 2.84})$$

Schließlich folgt die Assemblierung von Systemsteifigkeitsmatrix \underline{K} aus den Elementsteifigkeitsmatrizen \underline{k}_i und Systemlastvektor \underline{P} aus den Elementlastvektoren \underline{p}_i aller N_m Elemente. Daraus entsteht analog zu Gleichung 2.83 für ein i -tes Element aus Gleichung 2.84 die folgende Gleichung 2.85.

$$\delta \underline{V}^T \underline{K} \underline{V} = \delta \underline{V}^T \underline{P} \quad (\text{Gleichung 2.85})$$

(Stanoev 2013), S. 24-26 / (Stanoev 2016)

2.3.3 Wölb-Flächenmomente

Die Wölb-Flächenmomente $I_{\tilde{\omega}}(A_{\tilde{\omega}})$, $I_{y\tilde{\omega}}(A_{y\tilde{\omega}})$ und $I_{z\tilde{\omega}}(A_{z\tilde{\omega}})$ berechnen sich nach Gleichung 2.86, Gleichung 2.87 und Gleichung 2.88.

$$I_{\tilde{\omega}} = A_{\tilde{\omega}} = \int \int_{y \ z} \tilde{\omega}_s dy dz = \int_A \tilde{\omega}_s dA \quad (\text{Gleichung 2.86})$$

$$I_{y\tilde{\omega}} = A_{y\tilde{\omega}} = \int \int_{y \ z} y \tilde{\omega}_s dy dz = \int_A y \tilde{\omega}_s dA \quad (\text{Gleichung 2.87})$$

$$I_{z\tilde{\omega}} = A_{z\tilde{\omega}} = \int \int_{y \ z} z \tilde{\omega}_s dy dz = \int_A z \tilde{\omega}_s dA \quad (\text{Gleichung 2.88})$$

Eine äquivalente Formulierung für das in Unterkapitel 2.3.2 beschriebene FE-Modell liefern Gleichung 2.89 und Gleichung 2.90. Der \underline{V} -Vektor aus Unterkapitel 2.3.2 liefert benötigte Werte in den Knotenpunkten $k_{a,i}$ und $k_{e,i}$ eines i -ten Elements zur Beschreibung der Wölfunktion $\tilde{\omega}_s$.

$$I_{y\tilde{\omega}} = \frac{1}{6} \sum_{i=1}^{N_m} (y_{a,i}(2\tilde{\omega}_{s,ai} + \tilde{\omega}_{s,ei}) + y_{e,i}(\tilde{\omega}_{s,ai} + 2\tilde{\omega}_{s,ei})) t_i l_i \quad (\text{Gleichung 2.89})$$

$$I_{z\tilde{\omega}} = \frac{1}{6} \sum_{i=1}^{N_m} (z_{a,i}(2\tilde{\omega}_{s,ai} + \tilde{\omega}_{s,ei}) + z_{e,i}(\tilde{\omega}_{s,ai} + 2\tilde{\omega}_{s,ei})) t_i l_i \quad (\text{Gleichung 2.90})$$

Gleichung 2.91 beschreibt für ein FE-Modell das Wölbflächenmodell $I_{\tilde{\omega}}$.

$$I_{\tilde{\omega}} = \sum_{i=1}^{N_m} \left(\frac{\tilde{\omega}_{s,ai} + \tilde{\omega}_{s,ei}}{2} \right) t_i l_i \quad (\text{Gleichung 2.91})$$

(Stanoev 2013), S. 30-31 / (Stanoev 2016) / (Stanoev k.A.)

2.3.4 Schubmittelpunkt

Ziel eines Bezugs von Wölb-Flächenmomenten auf den Schubmittelpunkt eines Querschnitts ist die entkoppelte Betrachtung von Wölbnormalspannungen bei behinderter Verwölbung. Sie verursachen somit keine Normalkraft und/oder Biegemomente im Querschnitt. Die Koordinaten des Schubmittelpunkts M (*SMP*) beruhen auf den Koeffizienten a_y und a_z im Ansatz der Wölfunktion $\omega(y, z)$, siehe Gleichung 2.92. (y, z) beziehen sich auf das FSP-KS des Querschnitts.

$$\omega(y, z) = a + a_y y + a_z z + \tilde{\omega}(y, z) \quad (\text{Gleichung 2.92})$$

Im FSP-KS eines Querschnitts sind die statischen Momente $S_y (A_y)$ und $S_z (A_z)$ null, siehe Unterkapitel 2.1.2. Es gilt das Gleichungssystem in Gleichung 2.93.

$$\begin{bmatrix} I & 0 & 0 \\ 0 & I_{yy} & I_{yz} \\ 0 & I_{yz} & I_{zz} \end{bmatrix} \cdot \begin{bmatrix} a \\ a_y \\ a_z \end{bmatrix} = - \begin{bmatrix} I_{\tilde{\omega}} \\ I_{y\tilde{\omega}} \\ I_{z\tilde{\omega}} \end{bmatrix} \quad (\text{Gleichung 2.93})$$

Für die Koeffizienten a_y und a_z gelten somit Gleichung 2.94 und Gleichung 2.95.

$$a_y = \frac{-I_{zz} I_{y\tilde{\omega}} + I_{yz} I_{z\tilde{\omega}}}{I_{zz} I_{yy} - I_{yz}^2} = z_M \quad (\text{Gleichung 2.94})$$

$$a_z = \frac{I_{yz} I_{y\tilde{\omega}} - I_{yy} I_{z\tilde{\omega}}}{I_{zz} I_{yy} - I_{yz}^2} = -y_M \quad (\text{Gleichung 2.95})$$

Wenn FSP-KS = HTA-KS mit $I_{yz} = 0$ und $\varphi_1 = 0$ gilt, vereinfachen sich die Gleichungen zu Gleichung 2.96 und Gleichung 2.97.

$$a_y = \frac{-I_{y\tilde{\omega}}}{I_{yy}} = z_M \quad (\text{Gleichung 2.96})$$

$$a_z = \frac{-I_{z\tilde{\omega}}}{I_{zz}} = -y_M \quad (\text{Gleichung 2.97})$$

Die Koordinaten des SMP M eines Querschnitts sind (y_M, z_M) .

(Stanoev 2013), S. 30-32 / (Stanoev 2016)

Im SMP angreifende Querkräfte verursachen keine zusätzliche Torsion eines Querschnitts. Der SMP ist damit Querkraftmittelpunkt. Bei reiner Torsionsbelastung eines Querschnitts erfährt dieser eine Drehung φ_x um seinen SMP bzw. seine SMP-Achse. Daher kommt eine weitere Bezeichnung eines SMP als Drillruhepunkt D bzw. einer SMP-Achse als Drillruheachse. (Stanoev 2013), S. 33-35

Ein SMP liegt bei symmetrischen Querschnitten immer auf allen Symmetriechsen. Bei rotationssymmetrischen Querschnitten entspricht der SMP daher dem Drehpunkt. (Linke 2015), S. 162

Eine allgemeine Erklärung für einen SMP liefert Gleichung 2.98. Es besteht ein Gleichgewicht im SMP zwischen inneren und äußeren Torsionsmomenten (Schnittgrößen – vgl. Gleichung 2.35 und Gleichung 2.36 in Unterkapitel 2.2.3 und Belastungen). e_{yM} und e_{zM} bezeichnen dabei die Hebelarme der Querkräfte Q_z und Q_y um den gesuchten SMP. (Linke 2015), S. 163

$$T = \int_T dT = \int r_{tM}(s) T_s(s) ds = Q_z e_{yM} = Q_y e_{zM} \quad (\text{Gleichung 2.98})$$

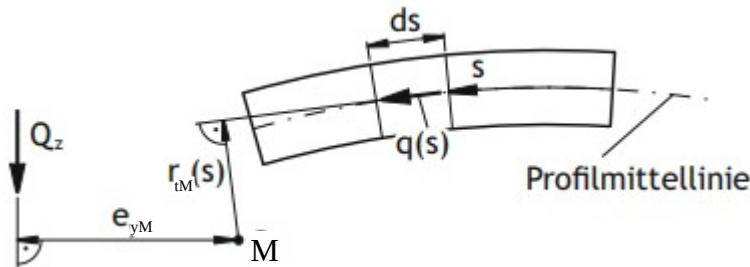


Abbildung 21: Geometrische Größen zur SMP-Bestimmung eines beliebig geformten dünnwandigen Querschnitts (Linke 2015), S. 163, bearb.

Mit den Koordinaten (y_M, z_M) eines SMP im FSP-KS ist eine Umrechnung zwischen den Polstrahlen r_{tO} und r_{tM} möglich, siehe Gleichung 2.99. Die Vorgehensweise ist analog auch für andere Polstrahle mit unterschiedlichen Bezugspunkten möglich, wie in Abbildung 22 und Gleichung 2.100 gezeigt. Dort hat r_{tD} den Bezugspunkt D mit den Koordinaten (y_D, z_D) im FSP-KS. (Mittelstedt 2021), S. 241, S. 303

$$r_{tM} = r_{tO} - (y_M + \bar{y}_s) \cos \alpha - (z_M + \bar{z}_s) \sin \alpha \quad (\text{Gleichung 2.99})$$

$$r_{tM} = r_{tO} - \bar{y}_M \cos \alpha - \bar{z}_M \sin \alpha$$

$$r_{tM} = r_{tD} - (y_M - y_D) \cos \alpha - (z_M - z_D) \sin \alpha \quad (\text{Gleichung 2.100})$$

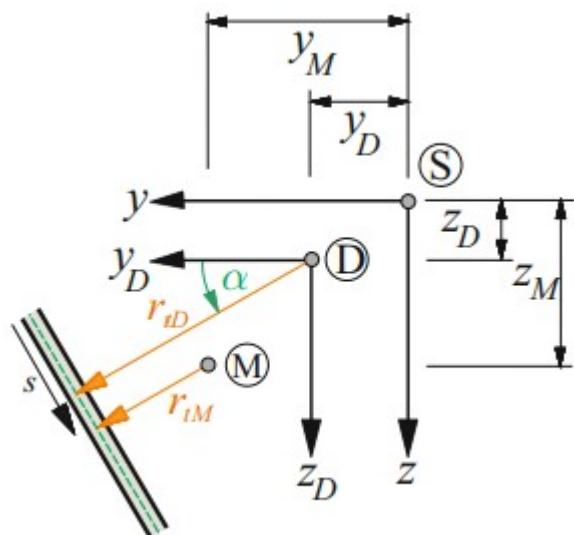


Abbildung 22: Geometrische Größen zur Umrechnung zwischen Polstrahlen r_{tD} und r_{tM} (Mittelstedt 2021), S. 241

2.3.5 Einheitsverwölbung und Wölbträgheitsmoment

Gleichung 2.92 aus Unterkapitel 2.3.4 stellt die Einheitsverwölbung $\omega_M(\bar{y}, \bar{z})$ bezogen auf den SMP $M(\bar{y}_M, \bar{z}_M)$ dar. Eine andere Schreibweise liefert Gleichung 2.101. Diese lineare Transformation aus $\omega(\bar{y}, \bar{z})$ ermöglicht einen Bezug der Verdrehung φ_x auf die SMP-Achse eines Querschnitts. Die Beschreibung von $\omega(\bar{y}, \bar{z})$ erfolgt mithilfe des in Unterkapitel 2.3.2 ermittelten \underline{V} -Vektors. Der Koeffizient a im Ansatz der Wölbfunktion $\omega(\bar{y}, \bar{z})$ entspricht dabei dem Anfangswert der Wölbfunktion (Integrationskonstante) ω_0 , siehe Gleichung 2.102.

$$\omega(\bar{y}, \bar{z}) = \omega_M(\bar{y}, \bar{z}) = \omega_0 + \bar{z}_M \bar{y} - \bar{y}_M \bar{z} + \tilde{\omega}(\bar{y}, \bar{z}) \quad (\text{Gleichung 2.101})$$

$$a = \omega_0 = \frac{-A_{\tilde{\omega}}}{A} \quad (\text{Gleichung 2.102})$$

Analog zu Gleichung 2.101 ist die Berechnung einer Einheitsverwölbung $\omega_S(\bar{y}, \bar{z})$ bezogen auf den FSP S möglich, siehe Gleichung 2.103.

$$\omega_S(\bar{y}, \bar{z}) = \omega_0 + \bar{z}_S \bar{y} - \bar{y}_S \bar{z} + \tilde{\omega}(\bar{y}, \bar{z}) \quad (\text{Gleichung 2.103})$$

Eine Umrechnung von $\omega_S(\bar{y}, \bar{z})$ zu $\omega_M(\bar{y}, \bar{z})$ erfolgt mit Gleichung 2.104.

$$\omega_M(\bar{y}, \bar{z}) = \omega_S(\bar{y}, \bar{z}) + z_M y - y_M z \quad (\text{Gleichung 2.104})$$

Mithilfe der Einheitsverwölbung $\omega_M(\bar{y}, \bar{z})$ ist die Berechnung des Wölbträgheitsmoments $I_{\omega\omega M}$ (auch $A_{\omega\omega M}$) des Querschnitts möglich, siehe Gleichung 2.105. Eine Formulierung im FE-Modell nach Abbildung 20 aus Unterkapitel 2.3.2 liefert Gleichung 2.106.

$$I_{\omega\omega M} = \int \omega_M^2 dA = \int \tilde{\omega} \omega_M dA \quad (\text{Gleichung 2.105})$$

$$I_{\omega\omega M} = \frac{1}{3} \sum_{i=1}^{N_m} (\omega_{M,ai}^2 + \omega_{M,ei}^2 + \omega_{M,ai} \omega_{M,ei}) t_i l_i \quad (\text{Gleichung 2.106})$$

2.3.6 Saint-Venantsche Torsion mit Wölbkrafttorsion

Wenn Wölbkrafttorsion / sekundäre Torsion in einem Querschnitt auftritt, ist das in der Berechnung des Torsionsträgheitsmomentes I_T zu berücksichtigen. Mithilfe von Gleichung 2.77 aus Unterkapitel 2.3.2 und Gleichung 2.46 und Gleichung 2.47 aus Unterkapitel 2.2.3 ist eine angepasste Formulierung von I_T eines geschlossenen dünnwandigen Querschnitts möglich, siehe Gleichung 2.107.

$$I_{T,geschl} = \sum_i^{N_m} r_{tO,i} t_i l_i (r_{tO,i} - \underline{B}_i v_i) \quad (\text{Gleichung 2.107})$$

Dazu kommt additiv ein Anteil für offene dünnwandige Querschnittsteile wie in Gleichung 2.57 aus Unterkapitel 2.2.5. Diese Gleichung unterscheidet nicht zwischen verwölbten und unverwölbten Querschnitten, siehe Gleichung 2.108.

$$I_{T,offen} = \frac{1}{3} \sum_i^{N_m} t_i^3 l_i \quad (\text{Gleichung 2.108})$$

$$I_T = I_{T,geschl} + I_{T,offen}$$

(Stanoev 2016)

3. Material und Methoden

Dieses Kapitel beinhaltet Informationen zu den Programmen „QUEBER_Version2016“ (Stanoev 2016) und „Thesis_Interface“ (Samlaf-Adams 2020) / (Vent 2022). Danach folgt ein Konzept für das neue Python-Programm.

3.1 Matlab-Programm „QUEBER_Version2016“

Das Programm „QUEBER_Version2016“ gilt für beliebige offene und/oder geschlossene ein- oder mehrzellige polygonale dünnwandige Querschnitte („Geometrische Werte dünnwandiger Querschnitte“) mit einheitlichem Schubmodul G . Berechnungsablauf und grafische Darstellungen sind Hauptvorlage für das neue Python-Programm. (Stanoev 2016)

3.1.1 Übersicht Programmbestandteile

Im folgenden stehen die im Programm verwendeten Überschriften.

„Vorbereiten der Eingabedaten [...]“

Einlesen der Profildaten aus einer Quelldatei [...]

Sortieren der Daten [...]

Berechnung der Querschnittswerte [...]“

Berechnen des Schwerpunktes und der Flächenträgheitsmomente [...]

Berechnung des Verdrehwinkels und der Hauptträgheitsachsen und Hauptträgheitsradien [...]

Berechnung Polstrahl zum finiten Element [...]

Prozedur zur Assemblierung der Systemsteifigkeitsmatrix/
Gesamtsteifigkeitsmatrix SS und des „rechte Seite“-Vektors B [...]

Berechnung der Werte der Wölbfunction OM in den Knotenpunkten $OMEG$ [...]

Lösung des Gleichungssystems $SS * OMEG = B$ [...]

Berechnung des St. Venantschen Torsionsträgheitsmomentes It [...]

Berechnung Normierungskonstante OM_0 [...]

Wölbkordinaten bezogen auf den Schwerpunkt S [...]

Berechnung Schubmittelpunkt-Koordinaten [...]

Wölbkoordinaten bezogen auf den SM-Punkt M [...]

Wölbträgheitsmoment berechnen A_{omomM} [...]

Vorbereiten und Erstellen der Ausgabedaten [...]“

Darstellen des Profils mit Verwölbungen in 2D und 3D [...]

Plotten der Querschnittskontur [...]

Plotten der 2D-Ansicht mit Verwölbung bzgl. KS-Ursprung 0 [...]

Plotten der 2D-Ansicht mit Verwölbung bzgl. Schwerpunkt S [...]

Plotten der 2D-Ansicht mit Verwölbung bzgl. Schubmittelpunkt M [...]

Vorbereiten der 3D-Plots [...]

Plotten der 3D-Ansicht mit Verwölbung bzgl. KS-Ursprung 0 [...]

Plotten der 3D-Ansicht mit Verwölbung bzgl. Schwerpunkt S [...]

Plotten der 3D-Ansicht mit Verwölbung bzgl. Schubmittelpunkt M “ (Stanoev 2016)

Eine genaue Darstellung von Eingabeverarbeitung und Berechnungsablauf liefert Anhang 8.3.1.

3.1.2 Eingabedatei

Eingabedaten sind Anzahl von Knoten N_k und Elementen N_m , Knotenkoordinaten im Ursprungs-KS (\bar{y}_i , \bar{z}_i) sowie Dicke t_i und Nummer von Anfangs- und Endknoten $k_{a,i}$ / $k_{e,i}$. Reihenfolge der Daten und Anordnung erfolgt wie im Zitat gezeigt.

„LV_DwSS__Woelbfunktion

Profil_Wunderlich

Anzahl Knoten : Abschnitte

9 8

Knotenkoordinaten Y Z

-6.00000 0.00000

[...]

10.00000 20.00000

Anfangspunkt : Endpunkt : Dicke

1 2 1.20

[...]

7 9 1.40 „

(Wunderlich k.A.)

3.1.3 Programmausgaben

Das Programm erstellt ein Textdokument „Ausgabefile“ mit dem zu Programmbeginn initialisierten Dateipfad, siehe Abbildung 137 (Anhang 8.3.1). Die ausgegebenen Querschnittswerte stehen in Tabelle 1 in 1. Einleitung. Im Textkopf sind Informationen zur Institution, zum Berechnungsprogramm und zum/zur Anwender/in vorhanden. Danach folgt eine Replikation der Eingabedaten und danach die Berechnungsergebnisse, siehe Anhang 8.3.2. (Stanoev 2016)

Ergebnisse sind zusätzlich in Form von 2D- und 3D-Graphiken der Querschnittskontur mit Wölfunktion – bezogen auf O, S bzw. M – im Matlab MuPad Notebook verfügbar. Ein Beispiel einer 2D-Grafik mit Wölfunktion / Einheitsverwölbung ω_M ist in Abbildung 23 dargestellt.

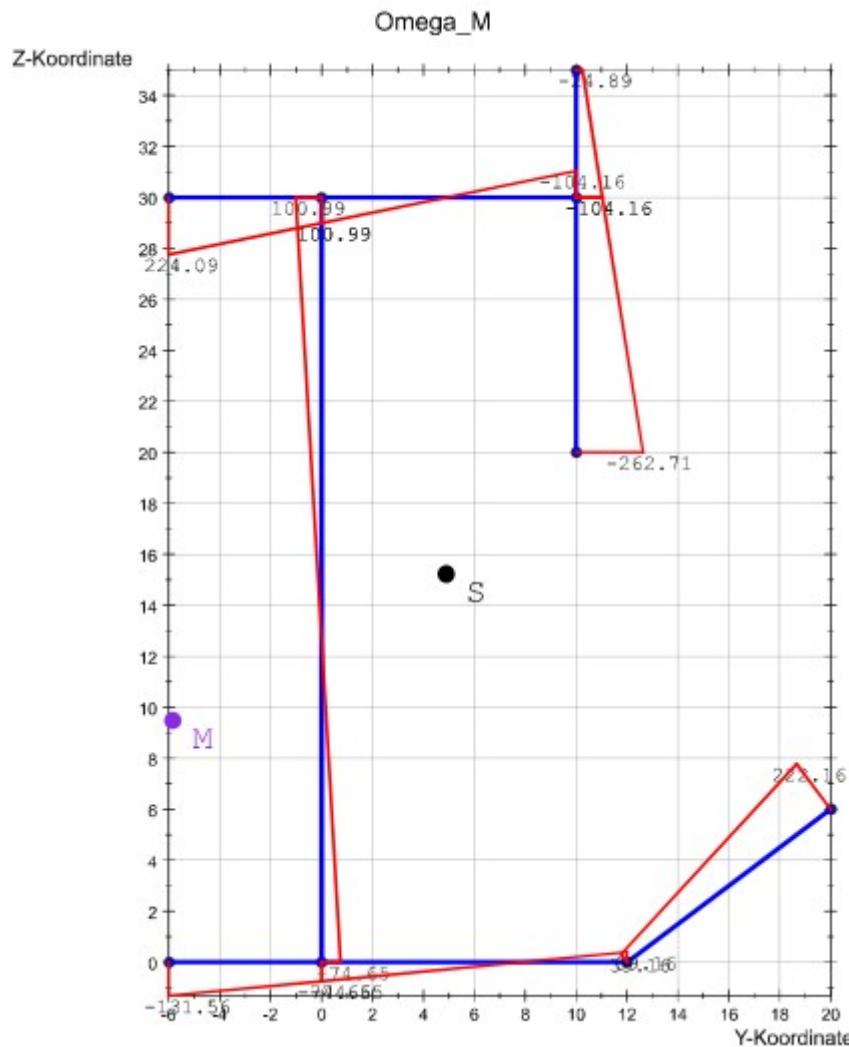


Abbildung 23: 2D-Grafik mit Knoten und Elementen sowie Wölfunktion bezogen auf M in „QUEBER_Version2016“, (Stanoev 2016) / (Wunderlich k.A.)

Die in Abbildung 23 dargestellten Informationen sind ebenfalls in 3D verfügbar, siehe Abbildung 24. Gleiches gilt für die Wölbfunktionen mit Bezug auf S und O .

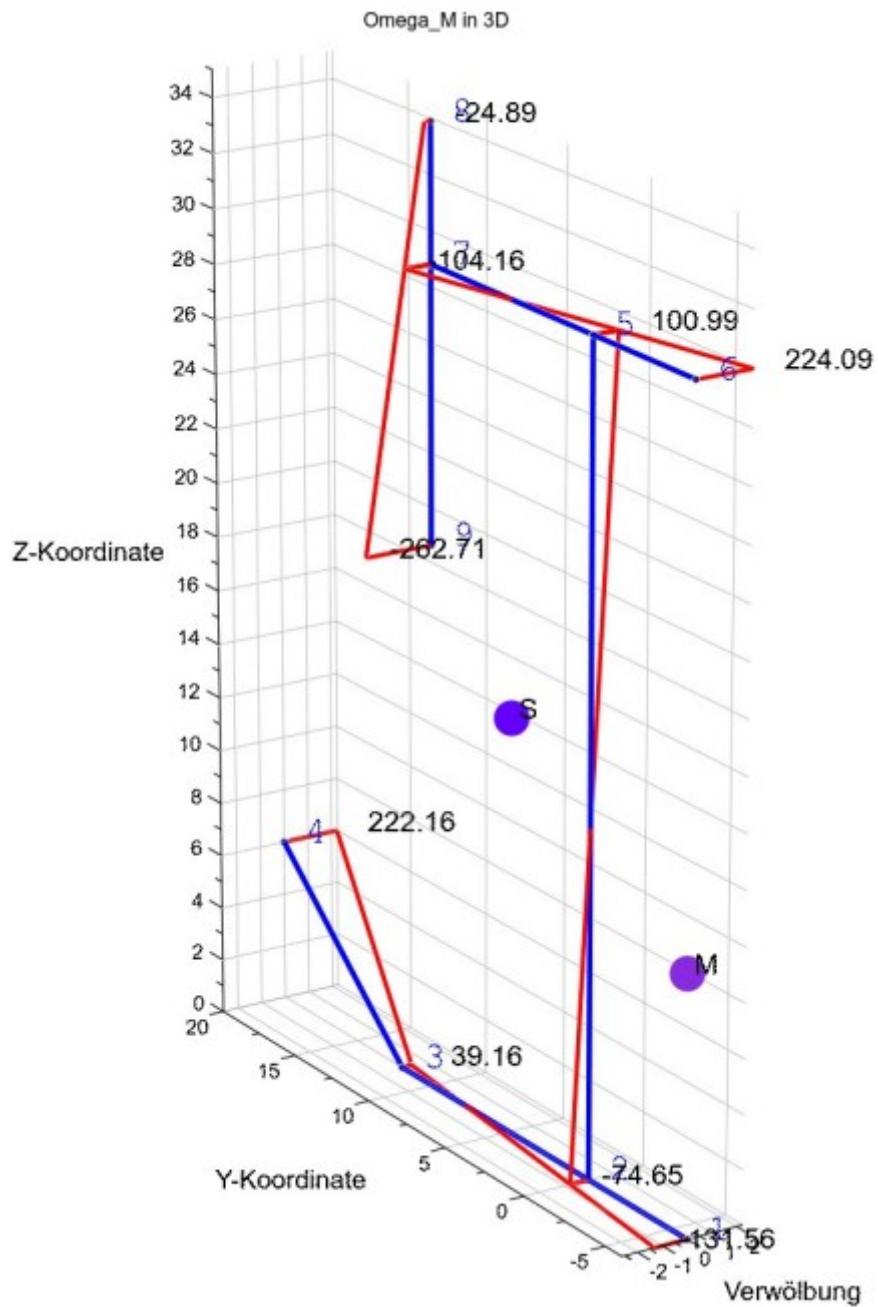


Abbildung 24: 3D-Grafik mit Knoten und Elementen sowie Wölbfunktion bezogen auf M in „QUEBER_Version2016“, (Stanoev 2016) / (Wunderlich k.A.)

(Stanoev 2016)

3.2 C#-Programm „Thesis_Interface“

Das Programm „Thesis_Interface“ für Rotorblätter von WEA wird in den Quellen (Samlafö-Adams 2020) und (Vent 2022) näher beschrieben. Es beinhaltet u.a. eine Berechnung von Querschnittswerten für Querschnitte aus Verbundwerkstoffen. Diese haben im Querschnitt variable Materialeigenschaften. Die Berechnung findet sich im Menüauswahlpunkt *Blade Cross-Section* bzw. in der Klasse *Blade_Cross_section*. (Samlafö-Adams 2020), „Thesis_Interface“, *Blade.cs*, Z. 885-1624

3.2.1 Vergleich mit „QUEBER_Version2016“

Formatierung und Inhalt der Eingabedatei sind prinzipiell wie im Matlab-Programm „QUEBER_Version2016“ aufgebaut, vgl. Unterkapitel 3.1.2. Der Unterschied besteht in zusätzlichen Angaben zu Materialeigenschaften, z.B. als Spalten mit mittlerem *E*-Modul und mittlerem Schubmodul *G* nach den Knotenkoordinaten und der Dicke. Die Dicke ist an Knoten angegeben und nicht wie bei „QUEBER_Version2016“ für Elemente. (Nan k.A.) / (Wunderlich k.A.)

Der Berechnungsablauf von *Blade Cross-Section* entspricht – samt Variablennamen und Gleichungen – zu großen Teilen dem Berechnungsablauf von „QUEBER_Version2016“, siehe Unterkapitel 3.1.1 bzw. Anhang 8.3.1. *Blade Cross-Section* beinhaltet weiterhin die Berechnung von Steifigkeiten aus Materialeigenschaften und Flächenmomenten. (Samlafö-Adams 2020), „Thesis_Interface“, *Blade.cs*, Z. 1538-1583

Es gibt ggf. weitere Programmiersprachen-spezifische Unterschiede, wie z.B. die funktionale und objektorientierte Anordnung von Funktionsdefinitionen und -aufrufen innerhalb einer Klasse *Blade_Cross_section* bei „Thesis_Interface“. Demgegenüber steht ein vorwiegend prozeduraler (und ansatzweise funktionaler) Programmierstil in „QUEBER_Version2016“, vgl. auch Anhang 8.2.1. (Samlafö-Adams 2020), „Thesis_Interface“, *Blade.cs*, Z. 885-952

3.2.2 2D-Grafik einer Wölbfunction in C#

Die Umsetzung in C# erfolgt mithilfe des Namespace `System.Windows.Forms.DataVisualization.Charting`, siehe Anhang 8.3.3 sowie (Vent 2022). Die Berechnung der Koordinaten der Wölbwerte an den Knoten erfolgt wie in (Stanoev 2016), „QUEBER_Version2016“, *proc OM_VEC()*, vgl. Abbildung 23 in Unterkapitel 3.1.3.

Abbildung 25 zeigt die mögliche grafische Aufbereitung von ausgewählten Querschnittswerten mithilfe von Windows Forms in C#. Die 2D-Grafik stellt den Rotorblattquerschnitt einer WEA aus der Inputdatei „bladeod200.txt“ (Nan k.A.) dar. Zudem erscheinen die Werte der Wölbfunction als Punkte und Linie über den Knoten bzw. Elementen des Querschnitts. Die Eintragung von Maximal- und Minimalwert `omm_max` und `omm_min` dient der besseren Einordnung und Plausibilitätsprüfung der Berechnungsergebnisse. Die Wölbfunction in der Grafik bezieht sich auf den SMP M .

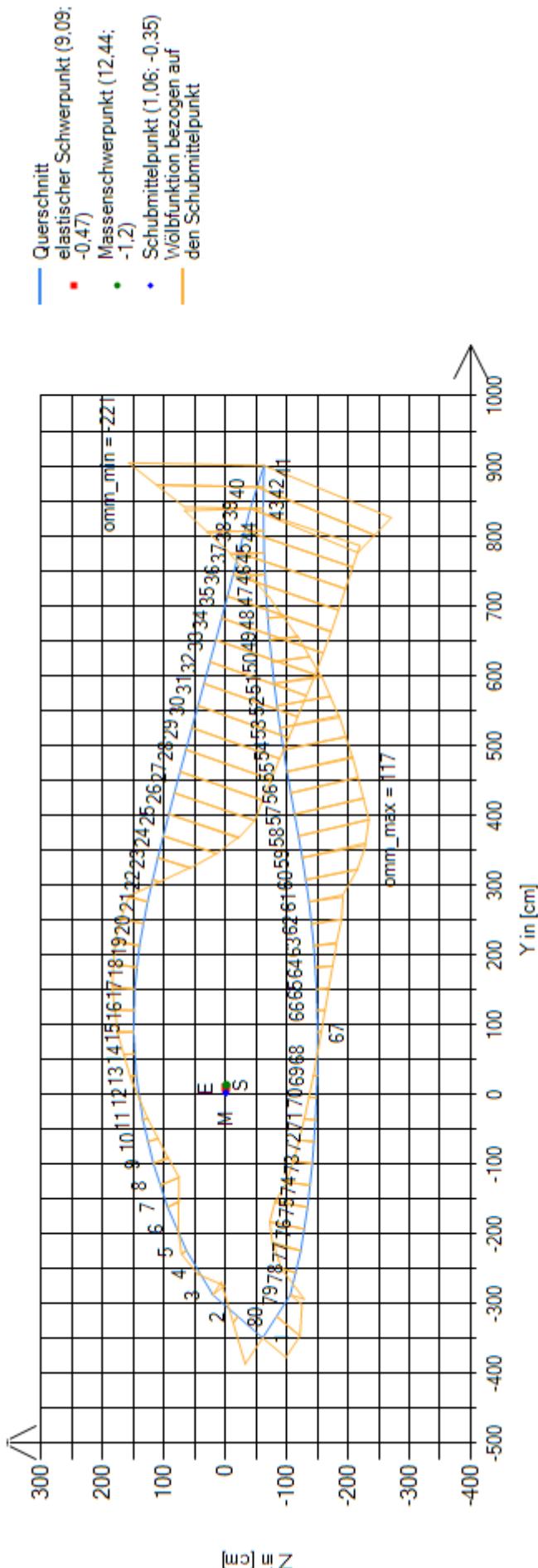


Abbildung 25: 2D-Darstellung eines WEA-Rotorblatt-Querschnitts „bladeod200.txt“ samt Wölbfunktion im C#-Programm „Thesis_Interface“ (Nan k.A.)

3.3 Aufbau neues Programm

Das neue Python-Programm enthält Gleichungen und Formelzeichen aus (Stanoev 2016), „QUEBER_Version2016“ bzw. (Samlaf-Adams 2020), „Thesis_Interface“, *Blade_Cross_section*. Die Planung erfolgt für ein Neodesign Top-Down. Dazu dienen functional abstraction und data abstraction als Darstellungsform für benötigte Programm-Bestandteile, vgl. (JavaTpoint 2021c) / (Tech with Tim 2021) bzw. Anhang 8.2.11.

3.3.1 Verwendete Python-Installation und IDE

Es kommt eine Python-Distribution von Anaconda zum Einsatz, vgl. Anhang 8.2.2 bzw. (Anaconda 2022). Diese enthält zum Zeitpunkt der Installation (10.10.2022) u.a. Python (Version 3.9.12), die IDE Spyder (Version 5.3.3) und pip. Numpy (Version 1.21.5), Matplotlib (Version 3.5.1) und SciPy (Version 1.7.3) sind auch mitinstalliert. Das ebenfalls enthaltene Jupiter Notebook (ähnlich einem Matlab Mupad Notebook) kommt nicht zum Einsatz. Spyder ist für die Erstellung eines unabhängig ausführbaren Programms optimal.

Eine Installation weiterer benötigter Packages wie z.B. PySimpleGUI erfolgt in einer jetzt verfügbaren Anaconda prompt:

„`conda install -c conda-forge pysimplegui`“ (Anaconda 2022a)

3.3.2 Funktionales Design

Functional abstraction beinhaltet eine Zusammenfassung von Anweisungen zu Funktionseinheiten. Diese sind in Tabelle 3 ersichtlich. Die Einteilung orientiert sich am Aufbau von Kapitel 2 bzw. dem in Unterkapitel 3.1.1 gezeigten Aufbau von „QUEBER_Version2016“. (JavaTpoint 2021c)

Tabelle 3 Funktionales Design des Python-Programms, vgl. (Stanoev 2016)

Bezeichnung der Funktionseinheit	Erzeugte Daten
Eingabe	
• Wähle Eingabefile (GUI)	Pfad Eingabefile
• Wähle ggf. Optionen (GUI)	z.B. GMOD, skalfak
• Starte Berechnung (GUI)	-
• Lese Eingabefile	String Eingabefile
• Sortiere Daten	NK, NM, KNOKO, STAB, DICKE
Datenverarbeitung	
• Berechne Flächenmomente	A, A_y, A_z, A_qy, A_qz, A_yy, A_zz, A_yz
• Berechne FSP	Ys, Zs
• Berechne Flächenmomente bez. auf S	A_yyS, A_zzS, A_yzS
• Berechne Hauptträgheitswerte	Phi, Phi_grad, I1, I2, i1, i2
• Berechne Polstrahle	RT
• Assembliere Systemlastvektor	B
• Assembliere Systemsteifigkeitsvektor	SS
• Berechnung V-Vektor	OMEG
• Berechne Torsionsträgheitsmoment	IT, TS
• Berechne Normierungskonstante	OM_0
• Berechne Wölbkoord. bez. auf S	OMS, OMS_skal
• Berechne Koordinaten des SMP	Ym, Zm, YmS, ZmS
• Berechne Wölbkoord. bez. auf M	OMM, OMM_skal
• Berechne Wölbträgheitsmoment	A_wwM
• (Berechne Steifigkeiten) – optional, bei Eingabe von variablem E-Modul / Schubmodul in Elementen/Knoten ist elementweise Berechnung nötig	EA, EA_y, EA_z, GA_qy, GA_qz, EA_yy, EA_zz, EA_yz, EA_yyS, EA_zzS, EA_yzS, EI1, EI2, GIT, EA_wwM
Ausgabe	
• Erzeuge Textdatei	Ausgabefile
• Erzeuge 2D-Ansichten mit Wölbfunktion bez. auf O, S, M	2D-Grafik (mit OM_skal, OMS_skal, OMM_skal)
• Erzeuge 3D-Ansichten mit Wölbfunktion bez. auf O, S, M	3D-Grafik (mit OM, OMS, OMM)

3.3.3 Data Design

Aufbauend auf dem funktionalem Design in Tabelle 3 in Unterkapitel 3.2.1 bzw. den Flowcharts in Anhang 8.3.1 folgt eine erweiterte Darstellung mit Zusammenhängen von Daten (cohesion and coupling, Gemeinsamkeiten und Abhängigkeiten), vgl. (Tech with Tim 2021).

Abbildung 26 zeigt Eingangsdaten, Variablen für FSP und Flächenmomente sowie optional (in Klammern) *E*-Modul, Schubmodul und Dehn-, Biege- und Schubsteifigkeiten, vgl. Unterkapitel 2.1 und (Stanoev 2016).

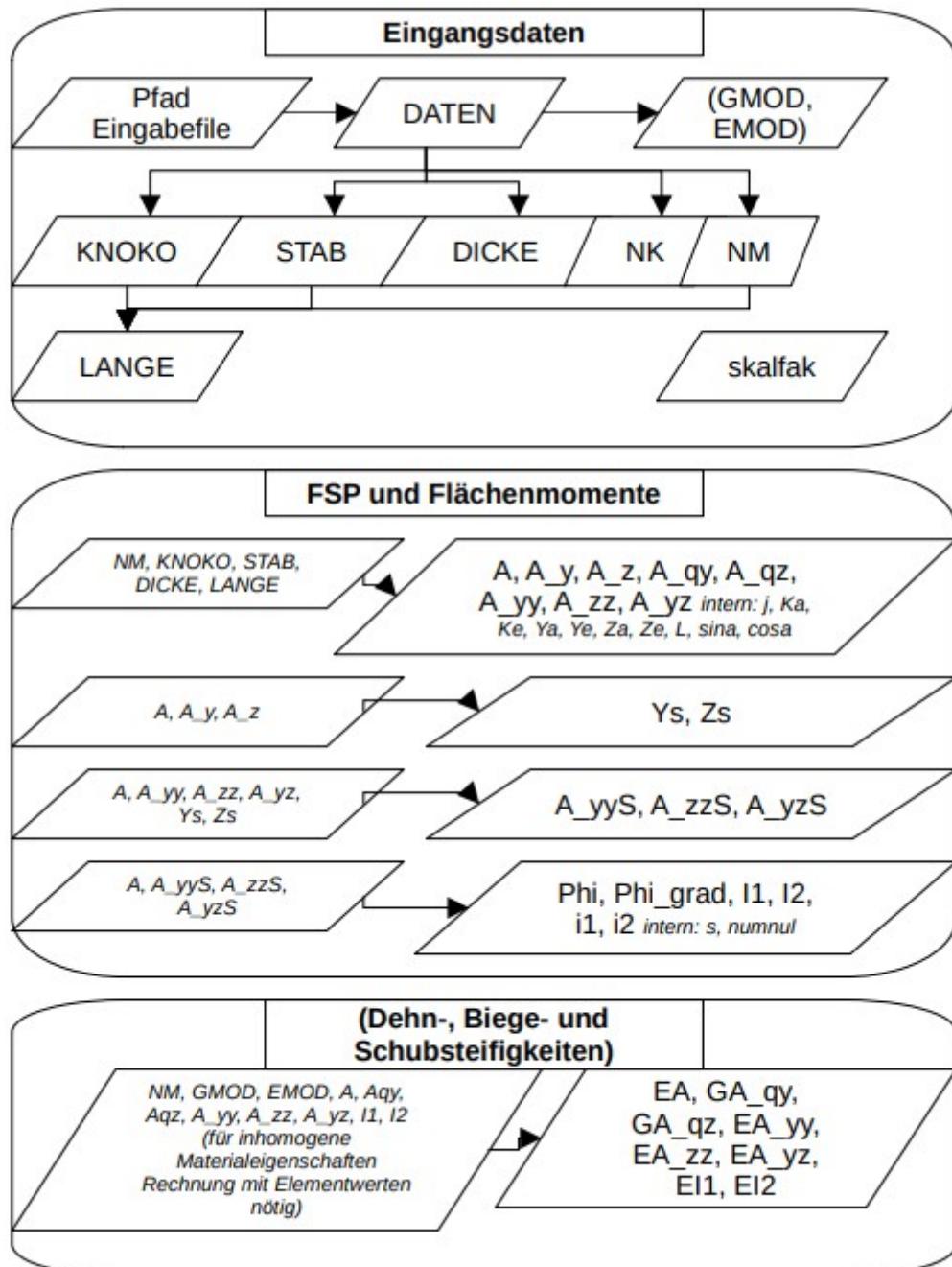


Abbildung 26: Eingangsdaten, FSP und Flächenmomente, Dehn-, Biege- und Schubsteifigkeiten im DataDesign Python-Programm,vgl.(Stanoev 2016)

In Abbildung 27 erscheinen benötigte Variablen zur Berechnung der Wölbfunction, bezogen auf O , und des Torsionsträgheitsmoments des Querschnitts, vgl. Unterkapitel 2.3. Optional ist eine Berechnung der Torsionssteifigkeit bei bekannten Schubmodul(en) der Elemente möglich. (Stanoev 2016)

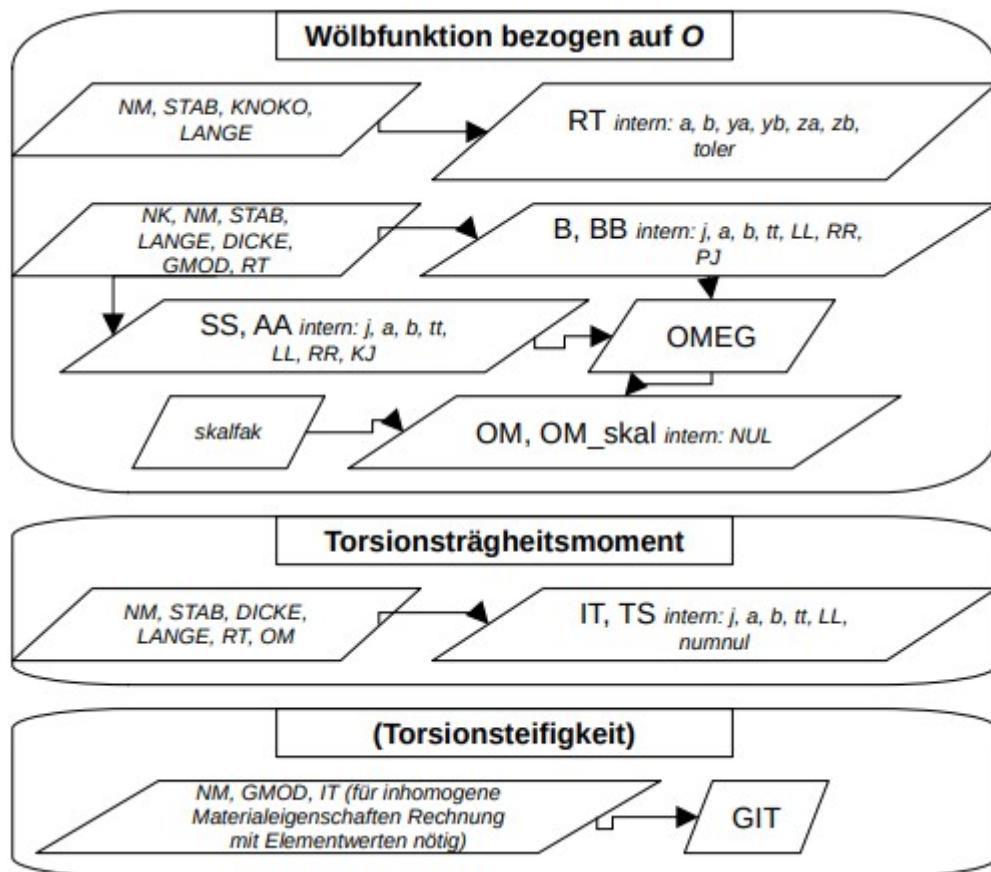


Abbildung 27: Wölbfunction, bez. auf O , Torsionsträgheitsmoment und -steifigkeit im Data Design des Python-Programms, vgl. (Stanoev 2016)

Abbildung 28 zeigt die letzten Daten im Programm. Diese sind Wölbfunktion, bezogen auf S und M , Koordinaten von SMP und Wölbträgheitsmoment, vgl. Unterkapitel 2.3. (Stanoev 2016)

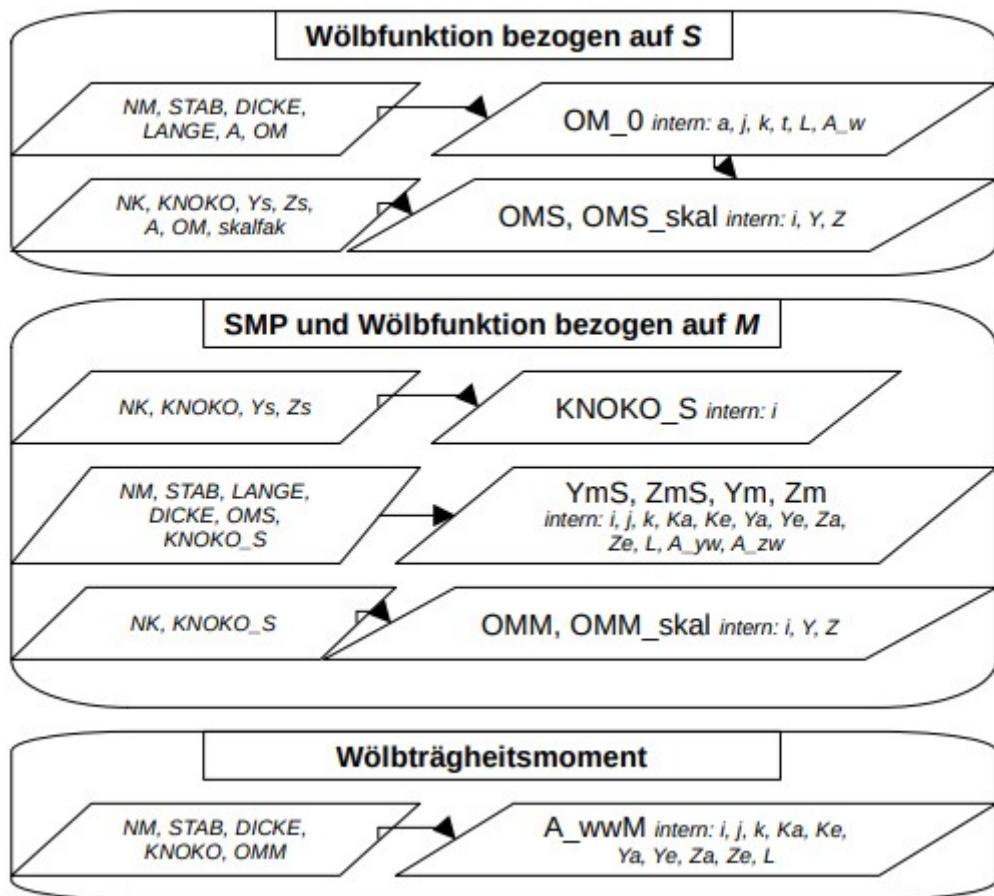


Abbildung 28: Wölbfunktion, bez. auf S und M , SMP und Wölbträgheitsmoment im Data Design des Python-Programms, vgl. (Stanoev 2016)

Schließlich erfolgt eine Zusammenfassung von Daten zu Datenpaketen mit hohem Abstrahierungsgrad aus den Darstellungen in Abbildung 26, Abbildung 27 und Abbildung 28. Das dient der Übersichtlichkeit, siehe Abbildung 29.

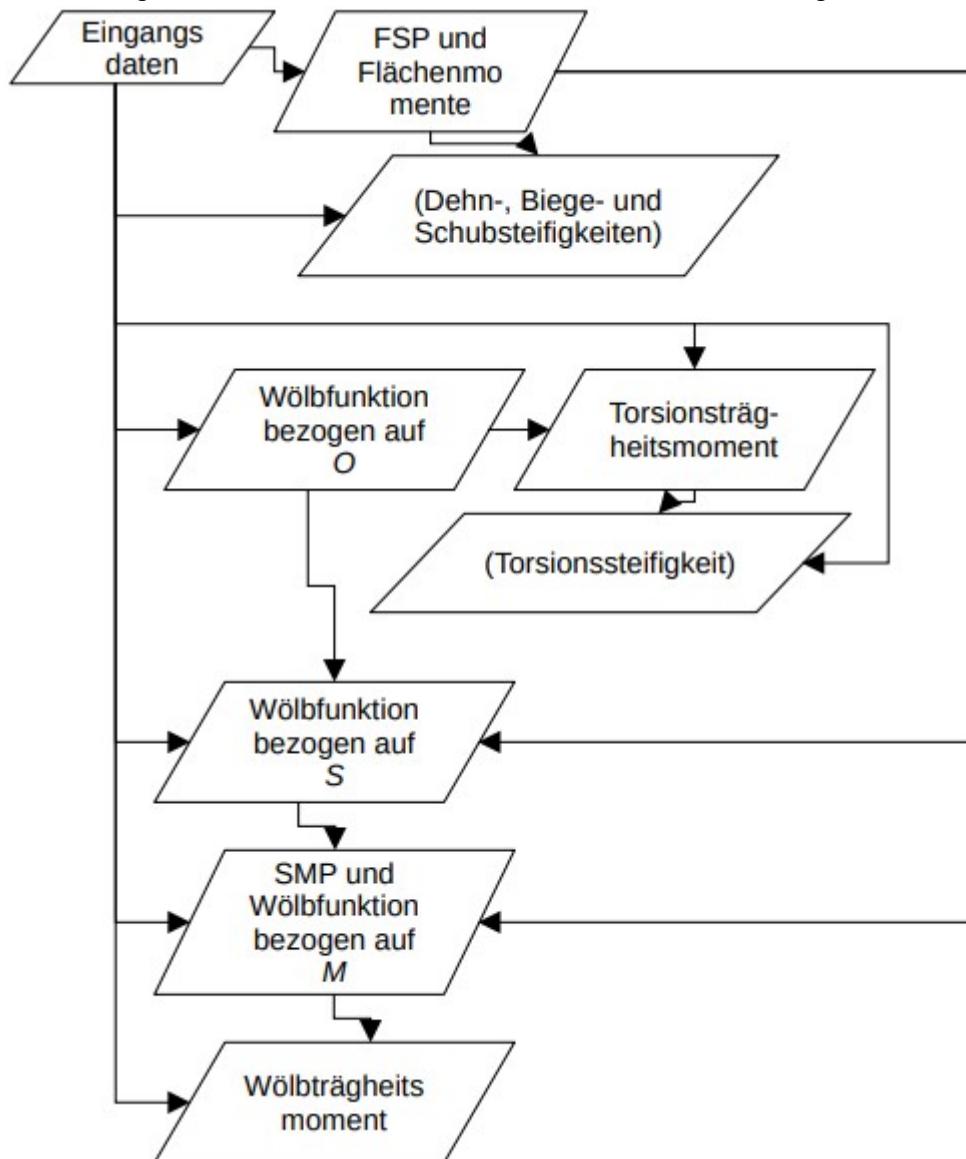


Abbildung 29: Abstrahierte Darstellung von Datenpaketen im Data Design des Python-Programms, vgl. (Stanoev 2016)

3.3.4 Konzepterstellung

Zwei Varianten ergeben sich aus den bisherigen Betrachtungen:

Es besteht die Möglichkeit einer Trennung von Daten und Funktionen / Methoden in mehrere Klassen (z.B. Querschnitt, Element, Knoten), vgl. Ende des Anhangs 8.1.1 bzw. (Werkle 2008), S. 555. Es besteht eine einfache Möglichkeit von scalability, d.h. weitere features wie z.B. Berechnung von Eigenmoden oder Hinzufügen neuer Elementtypen sind effizient möglich („Design for flexibility“), vgl. Anhang 8.2.11 bzw. (Tech with Tim 2021).

Ein mögliches Klassendiagramm sieht wie in Abbildung 30 aus. Die Zahlen (bzw. * für beliebig viele) an den Assoziationslinien stehen für die mögliche Anzahl der an einer Assoziation beteiligten Instanzen dieser Klasse, vgl. Abbildung 123 in Anhang 8.2.11 bzw. (Tech with Tim 2020).

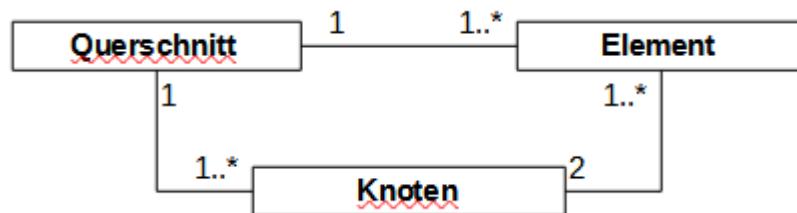


Abbildung 30: Klassendiagramm eines Konzepts mit OOP für neues Programm, vgl. (Tech with Tim 2020) / (Werkle 2008), S. 555

Es ist weiterhin eine Zusammenfassung von Daten in Vektoren und Matrizen in einer vorwiegend prozeduralen und funktionalen Variante möglich, vgl. (Stanoev 2016), „QUEBER_Version2016“ bzw. Unterkapitel 3.1.1 und Anhang 8.3.1. Der Programmaufbau richtet sich in dieser Variante nach den in Unterkapiteln 3.3.2 und 3.3.3 beschriebenen funktionalen bzw. datenbezogenen Unterteilungen.

Da Einfachheit ein wichtiges Design-Kriterium in Python ist, wird in der Erstellung eines neuen Python-Programms diese Variante bevorzugt, (vgl. „Zen of Python“: “Simple is better than complex.“ / „Readability counts.“ / „If the implementation is hard to explain, it's a bad idea.“ / „If the implementation is easy to explain, it may be a good idea.“ , siehe Ende von Anhang 8.2.11 bzw. (Peters 2004)).

4. Ergebnisse

4.1 Programmbeschreibung

Das neue Python-Programm beinhaltet eine Berechnung von geometrischen Querschnittswerten (auch bei unbekannten Materialeigenschaften) wie in Tabelle 1 in 1. Einleitung beschrieben. Eine Berechnung von Steifigkeitswerten ist nicht enthalten.

Bei Programmstart erscheint ein Fenster zur Eingabe eines Dateipfads, siehe Abbildung 31. Bei Klick des „Solve“-Buttons beginnt als erstes die Berechnung, vgl. Abbildung 29 in Unterkapitel 3.3.3. Lesbare Eingabedateien entsprechen der Formatierung wie für „QUEBER_Version2016“ (vgl. Unterkapitel 3.1.2 bzw. (Wunderlich k.A.)). Sie enthalten die in Tabelle 2 in 1. Einleitung genannten Daten. Kommentare nach Werten wie in „FR-01-VAR1neu.txt“ (Peters k.A. b) führen zu einem Programmabsturz und sind daher zu vermeiden bzw. zu entfernen.



Abbildung 31: Eingabefenster für das Python-Programm „Thin-Walled Cross Section Properties“

Weiterhin speichert das Programm Ergebnisse in einer neu erstellten Textdatei ab (ähnlich zu Ausgabedatei von „QUEBER_Version2016“ in Anhang 8.3.2). Dazu dient der Eingabedateipfad und der Dateiname der Eingabedatei. Dieser Dateiname wird mit einer Endung versehen („_results“) und im selben Verzeichnis abgespeichert.

Weiterhin öffnet sich ein zweites Fenster „Graph Plot Menu“. Darin sind Möglichkeiten zur grafischen Ausgabe von Querschnittsgeometrie und zusätzlichen Daten in 2D und 3D enthalten. Zusätzliche Daten sind FSP S, SMP M sowie Wölfunktionen ($\omega(\bar{y}, \bar{z})$, $\omega_s(\bar{y}, \bar{z})$ bzw. $\omega_M(\bar{y}, \bar{z})$) bezogen auf O, S bzw. M.

Abbildung 32 zeigt Module und Top-Level-Funktionen (bzw. deren Aufrufe) aus dem neuen Python-Programm. Zunächst erscheinen das Start-Modul „Main“ mit Aufrufen von Top-Level-Funktionen sowie das Modul „auxiliary_variables“ zur Berechnung von benötigten Zwischenwerten.

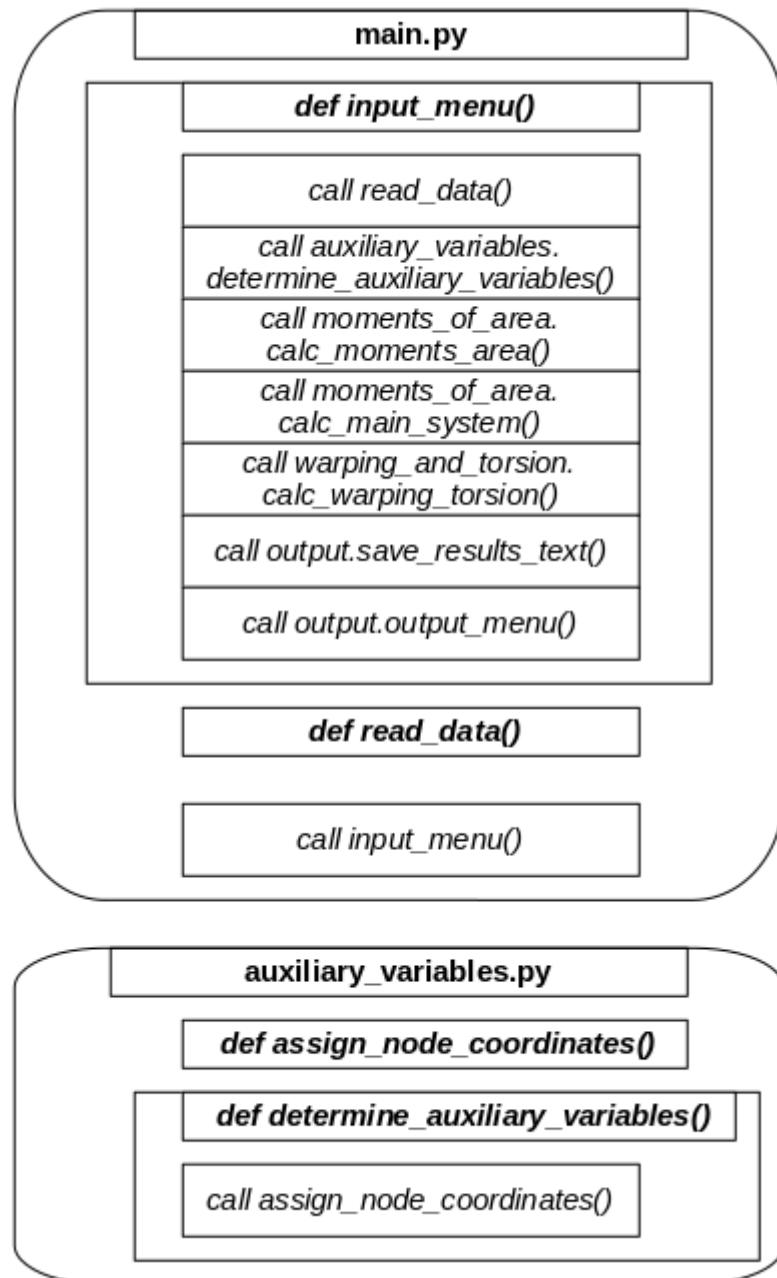


Abbildung 32: Functional Design Teil 1 für das Python-Programm „Thin-Walled Cross Section Properties“

Weiterhin folgen die Module „moments_of_area“, „warping_and_torsion“ sowie „output“, siehe Abbildung 33.

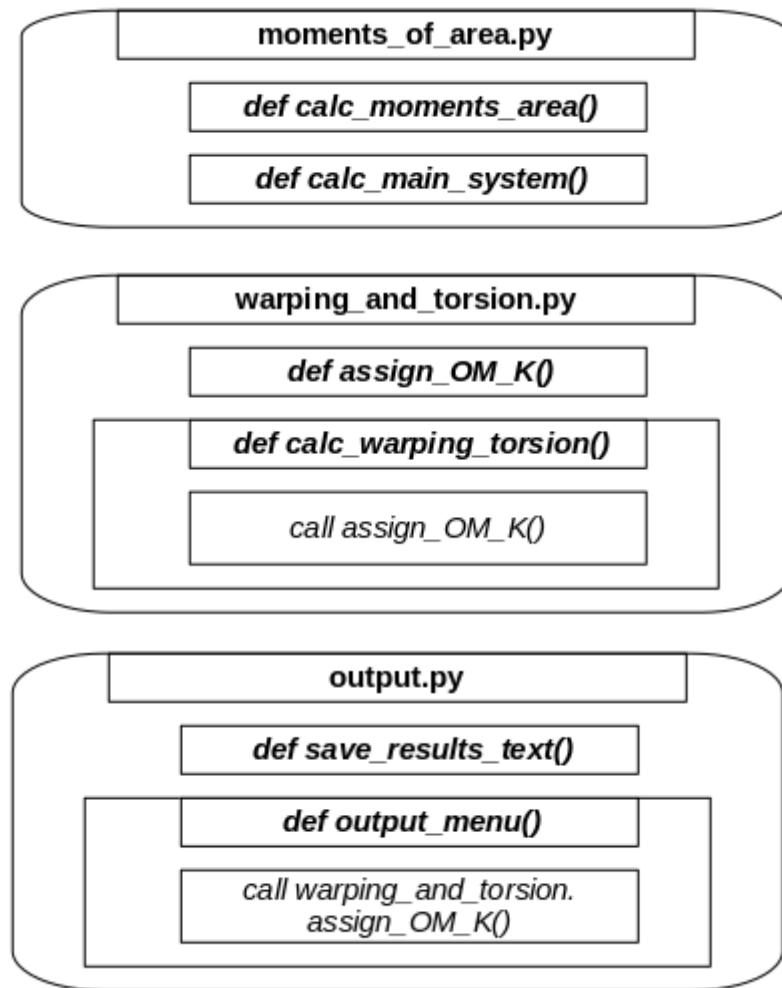


Abbildung 33: Functional Design Teil 2 für das Python-Programm „Thin-Walled Cross Section Properties“

4.2 Grafiken

Die grafische Ausgabe erfolgt mithilfe einer zusätzlichen GUI („Graph Plot Menu“). Diese dient als Träger für Matplotlib-Funktionen. Vorbild dafür ist (Github 2022). Ein Speichern bzw. Anpassen der angezeigten Grafik ist mithilfe einer eingebauten Toolbar möglich.

Als Beispiel für eine 2D-Grafik zeigt Abbildung 34 Ergebnisse von „Profil-Wunderlich.txt“ (Wunderlich k.A.).

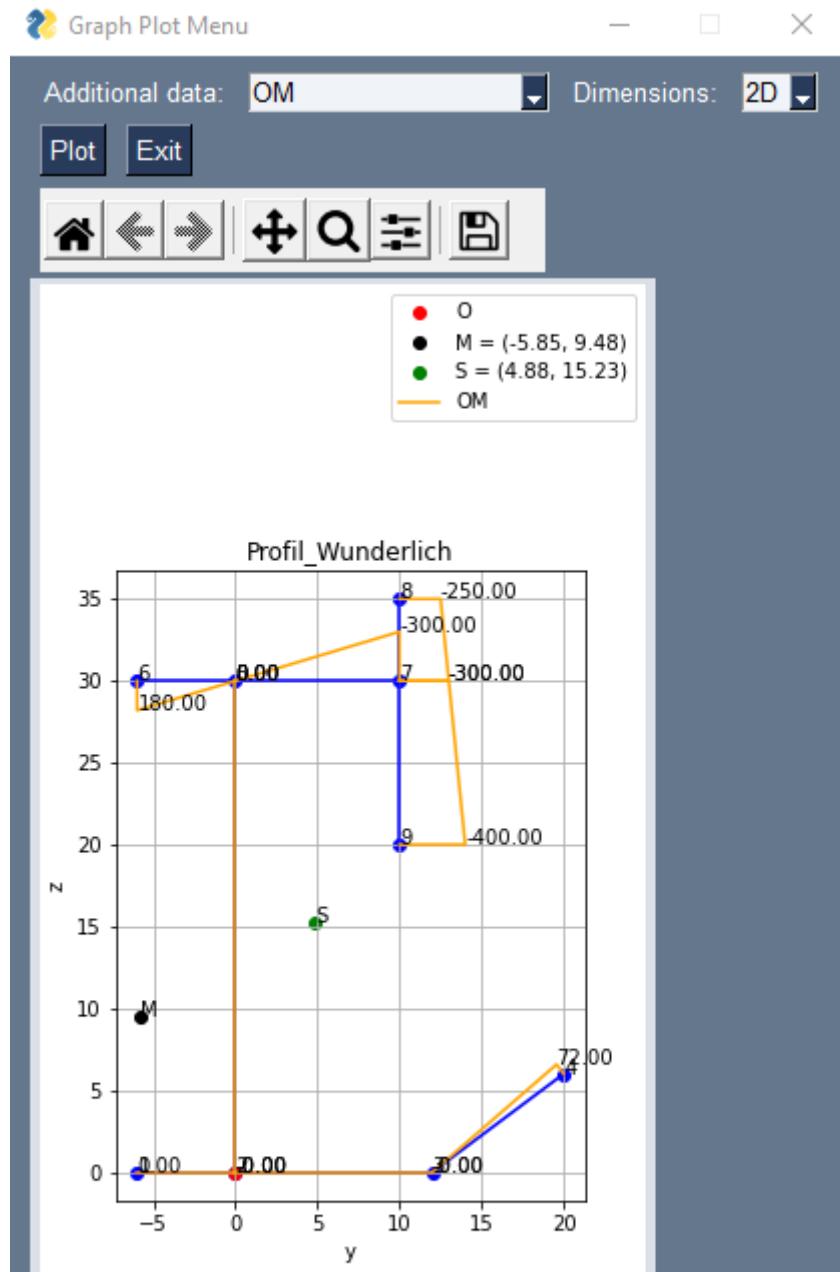


Abbildung 34: 2D-Grafik mit Wölfelfunktion OM für „Profil-Wunderlich.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Wunderlich k.A.)

3D-Grafiken sind ebenfalls möglich. Ein Beispiel zeigt Abbildung 35, ebenfalls mit der Eingabedatei „Profil-Wunderlich.txt“ (Wunderlich k.A.).

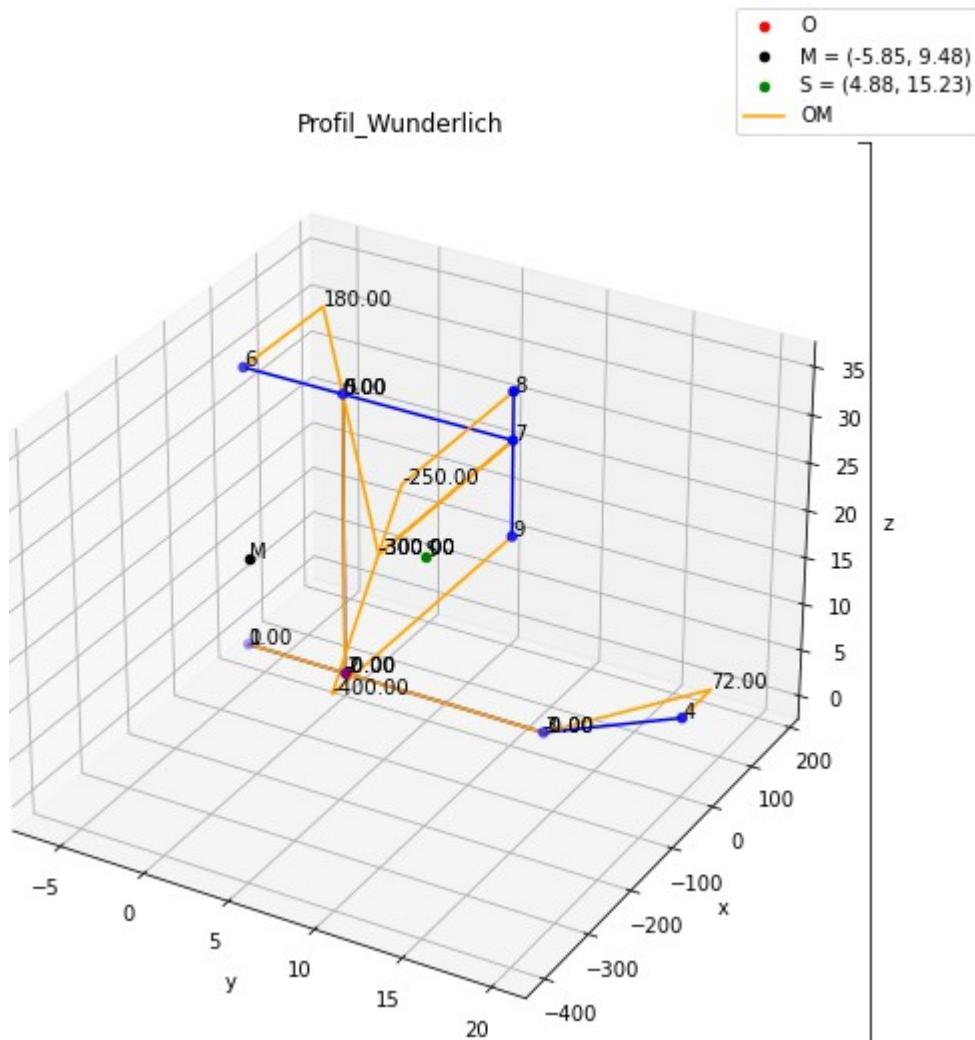


Abbildung 35: 3D-Grafik mit Wölfunktion OM für „Profil-Wunderlich.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Wunderlich k.A.)

Eine Darstellung der Wölbfunction bezogen auf S ($\omega_s(\bar{y}, \bar{z})$) liefert Abbildung 36.

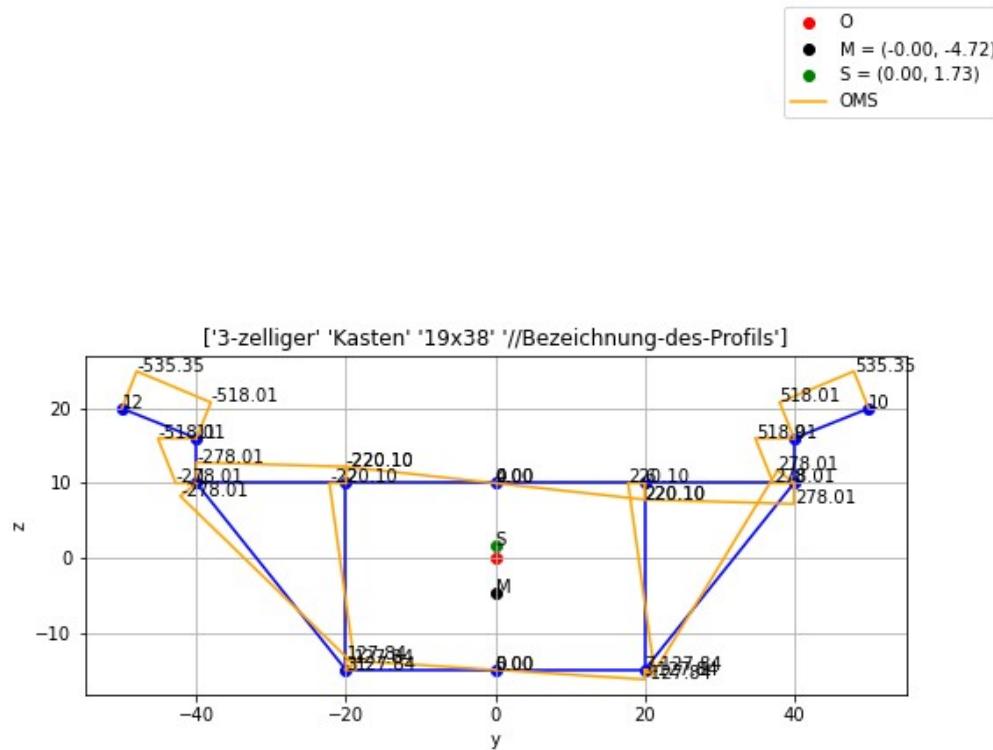


Abbildung 36: 2D-Grafik mit Wölbfunction OMS für „3kasten_Spkt.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Stanoev k.A. a)

Weiterhin zeigt Abbildung 37 in 3D den Querschnitt aus „3kasten_Spkt.txt“ (Stanoev k.A. a) mit Wölfunktion bezogen auf $M(\omega_M(\bar{y}, \bar{z}))$.

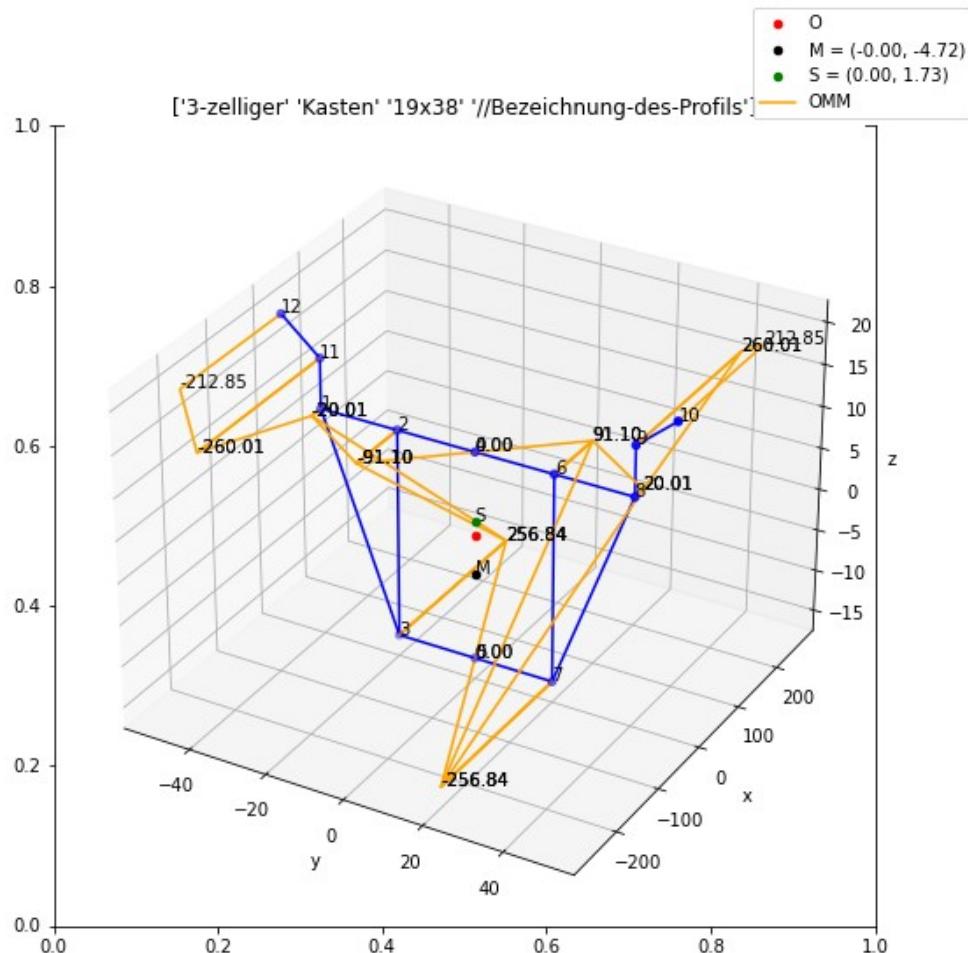


Abbildung 37: 3D-Grafik mit Wölfunktion OMM für „3kasten_Spkt.txt“ im „Graph Plot Menu“ des Python-Programms, vgl. (Stanoev k.A. a)

4.3 Ergebnisvalidierung

Es ist eine Validierung der Berechnungsergebnisse erforderlich. Dazu stehen in Tabelle 4 Berechnungsergebnisse von „QUEBER_Version2016“ und dem neuen Python-Programm gegenüber. Die Reihenfolge der Daten folgt einer Ausgabedatei (vgl. Anhang 8.3.2). Für die Eingabedatei „Profil_Wunderlich.txt“ stimmen alle Ausgabewerte sehr genau überein.

Tabelle 4 Vergleich der Berechnungsergebnisse „QUEBER_Version2016“ und Python-Programm mit Eingabedatei „Profil_Wunderlich.txt“ (Wunderlich k.A.)

Größe	Python-Programm	„QUEBER Version2016“	Abweichung [%]
Ys	4.88455	4.88455	0.00
Zs	15.23171	15.2317	0.00
A	123.00	123.0	0.00
A_qy	62.80	62.8	0.00
A_qz	66.60	66.6	0.00
A_y	600.80	600.8	0.00
A_z	1873.50	1873.5	0.00
A_yy	7871.20	7871.2	0.00
A_zz	48507.00	48507.0	0.00
A_yz	7843.00	7843.0	0.00
A_yyS	4936.56	4936.56	0.00
A_zzS	19970.40	19970.4	0.00
A_yzS	-1308.21	-1308.21	0.00
Phi_grad	4.93631	4.93631	0.00
I1	20083.38	20083.4	0.00
I2	4823.57	4823.57	0.00
i1	12.77809	12.7781	0.00
i2	6.26227	6.26227	0.00
IT	81.832	81.832	0.00
Ym	-5.85488	-5.85488	0.00
Zm	9.48444	9.48444	0.00
YmS	-10.73943	-10.7394	0.00
ZmS	-5.74726	-5.74726	0.00
A_wwM	1369984.11	1369980.0	0.0003
OM (1)	0.0000	0	0.00

OM (2)	-0.0000	6.21725e-14	0.00
OM (3)	-0.0000	9.14824e-14	0.00
OM (4)	72.0000	72.0	0.00
OM (5)	0.0000	3.48166e-13	0.00
OM (6)	180.0000	180.0	0.00
OM (7)	-300.0000	-300.0	0.00
OM (8)	-250.0000	-250.0	0.00
OM (9)	-400.0000	-400.0	0.00
OMS (1)	-30.5366	-30.5366	0.00
OMS (2)	60.8537	60.8537	0.00
OMS (3)	243.6341	243.634	0.00
OMS (4)	408.1805	408.18	0.00
OMS (5)	-85.6829	-85.6829	0.00
OMS (6)	2.9268	2.92683	0.00
OMS (7)	-233.3659	-233.366	0.00
OMS (8)	-207.7886	-207.789	0.00
OMS (9)	-284.5203	-284.52	0.00
OMM (1)	-131.5601	-131.56	0.00
OMM (2)	-74.6535	-74.6535	0.00
OMM (3)	39.1599	39.1599	0.00
OMM (4)	222.1647	222.165	0.00
OMM (5)	100.9930	100.993	0.00
OMM (6)	224.0863	224.086	0.00
OMM (7)	-104.1626	-104.163	0.00
OMM (8)	-24.8881	-24.8881	0.00
OMM (9)	-262.7114	-262.711	0.00
TS (1)	0.00000	-	0.00
TS (2)	0.00000	-	0.00
TS (3)	0.00000	-	0.00
TS (4)	0.00000	-	0.00
TS (5)	0.00000	-	0.00
TS (6)	0.00000	-	0.00
TS (7)	0.00000	-	0.00
TS (8)	0.00000	-	0.00
TS (9)	0.00000	-	0.00

Die in Tabelle 5 gezeigten Werte mit der Eingabedatei „3kasten_Spkt.txt“ stimmen ebenfalls sehr genau überein. Hier zeigt sich die korrekte Berechnung der bezogenen Schubflüsse in den Elementen (TS(1)-TS(14)). Diese sind in diesem Beispiel im Gegensatz zu „Profil_Wunderlich.txt“ ungleich null. Grund dafür ist die geschlossene Form dieses Querschnitts.

Tabelle 5 Vergleich der Berechnungsergebnisse „QUEBER_Version2016“ und Python-Programm mit Eingabedatei „3kasten_Spkt.txt“ (Stanoev k.A. a)

Größe	Python-Programm	„QUEBER Version2016“	Abweichung [%]
Ys	0.00000	5.66491e-18	0.00
Zs	1.73375	1.73375	0.00
A	156.79	156.786	0.00
A_qy	98.00	98.0	0.00
A_qz	75.00	75.0	0.00
A_y	0.00	8.88178e-16	0.00
A_z	271.83	271.827	0.00
A_yy	105647.58	105648.0	0.00
A_zz	18036.86	18036.9	0.00
A_yz	0.00	0.0	0.00
A_yyS	105647.58	105648.0	0.00
A_zzS	17565.58	17565.6	0.00
A_yzS	0.00	-1.53988e-15	0.00
Phi_grad	0.00000	0.0	0.00
I1	105647.58	105648.0	0.00
I2	17565.58	17565.6	0.00
i1	25.95830	25.9583	0.00
i2	10.58468	10.5847	0.00
IT	29842.819	29842.8	0.00
Ym	-0.00000	2.09673e-15	0.00
Zm	-4.71632	-4.71632	0.00
YmS	-0.00000	2.09107e-15	0.00
ZmS	-6.45007	-6.45007	0.00
A_wwM	2672491.90	2672490.0	0.00

OM (1)	0.0000	0	0.00
OM (2)	23.2353	23.2353	0.00
OM (3)	371.1789	371.179	0.00
OM (4)	208.6642	208.664	0.00
OM (5)	208.6642	208.664	0.00
OM (6)	394.0931	394.093	0.00
OM (7)	46.1496	46.1496	0.00
OM (8)	417.3285	417.328	0.00
OM (9)	657.3285	657.328	0.00
OM (10)	657.3285	657.328	0.00
OM (11)	-240.0000	-240.0	0.00
OM (12)	-240.0000	-240.0	0.00
OMS (1)	-278.0140	-278.014	0.00
OMS (2)	-220.1038	-220.104	0.00
OMS (3)	127.8398	127.84	0.00
OMS (4)	0.0000	1.79412e-13	0.00
OMS (5)	0.0000	-3.19744e-14	0.00
OMS (6)	220.1038	220.104	0.00
OMS (7)	-127.8398	-127.84	0.00
OMS (8)	278.0140	278.014	0.00
OMS (9)	518.0140	518.014	0.00
OMS (10)	535.3515	535.351	0.00
OMS (11)	-518.0140	-518.014	0.00
OMS (12)	-535.3515	-535.351	0.00
OMM (1)	-20.0113	-20.0113	0.00
OMM (2)	-91.1024	-91.1024	0.00
OMM (3)	256.8412	256.841	0.00
OMM (4)	0.0000	1.62163e-13	0.00
OMM (5)	-0.0000	3.05348e-15	0.00
OMM (6)	91.1024	91.1024	0.00
OMM (7)	-256.8412	-256.841	0.00
OMM (8)	20.0113	20.0113	0.00
OMM (9)	260.0113	260.011	0.00
OMM (10)	212.8480	212.848	0.00
OMM (11)	-260.0113	-260.011	0.00
OMM (12)	-212.8480	-212.848	0.00

TS (1)	-6.69706	-6.69706	0.00
TS (2)	6.69706	6.69706	0.00
TS (3)	4.86581	4.86581	0.00
TS (4)	-11.56287	-11.5629	0.00
TS (5)	11.56287	11.5629	0.00
TS (6)	-11.56287	-11.5629	0.00
TS (7)	11.56287	11.5629	0.00
TS (8)	-4.86581	-4.86581	0.00
TS (9)	-6.69706	-6.69706	0.00
TS (10)	-6.69706	-6.69706	0.00
TS (11)	0.00000	0	0.00
TS (12)	0.00000	0	0.00
TS (13)	0.00000	0	0.00
TS(14)	0.00000	0	0.00

„FR-01-VAR1neu.txt“ (Stanoev k.A. b) funktioniert im Matlab MuPad Notebook „QUEBER_Version2016“. Für das Python-Programm ist ein Entfernen von Kommentaren nach Zeilen mit Werten nötig. Die Ergebnisse stehen in Tabelle 6 gegenüber. Die Abweichungen aller Werte liegen unter 0.00%. Für eine kürzere Darstellung sind einige Werte nicht eingetragen (mit ... gekennzeichnet).

Tabelle 6 Vergleich der Berechnungsergebnisse „QUEBER_Version2016“ und Python-Programm mit Eingabedatei „FR-01-VAR1neu.txt“ (Peters k.A. b)

Größe	Python-Programm	„QUEBER _Version2016“	Abweichung [%]
Ys	0.00000	-8.23044e-17	0.00
Zs	366.52784	366.528	0.00
A	3453.24	3453.24	0.00
A_qy	2332.14	2332.14	0.00
A_qz	1449.24	1449.24	0.00
A_y	0.00	-2.84217e-13	0.00
A_z	1265709.92	1265710.0	0.00
A_yy	189533027.65	189533000.0	0.00
A_zz	525022443.81	525022000.0	0.00
A_yz	0.00	-0.000000000232831	0.00

A_yyS	189533027.65	189533000.0	0.00
A_zzS	61104518.67	61104500.0	0.00
A_yzS	0.00	-0.000000000128657	0.00
Phi_grad	0.00000	0.0	0.00
I1	189533027.65	189533000.0	0.00
I2	61104518.67	61104500.0	0.00
i1	234.27656	234.277	0.00
i2	133.02187	133.022	0.00
IT	126215267.514	126215000.0	0.00
Ym	-0.00000	3.21895e-13	0.00
Zm	302.35045	302.35	0.00
YmS	-0.00000	3.21977e-13	0.00
ZmS	-64.17739	-64.1774	0.00
A_wwM	310298266738.45	3.10298e+11	0.00
OM (1)	0.0000	0	0.00
OM (2)	-10778.2649	-10778.3	0.00
OM (3)	-18872.9115	-18872.9	0.00
OM (4)	-37696.0117	-37696.0	0.00
OM (5)	-43156.5224	-43156.5	0.00
OM (6)	-50069.0994	-50069.1	0.00
OM (7)	-56114.0679	-56114.1	0.00
OM (8)	-62664.5822	-62664.6	0.00
OM (9)	-68231.8362	-68231.8	0.00
OM (10)	-72654.6691	-72654.7	0.00
OM (11)	-78248.9788	-78249.0	0.00
OM (12)	-82070.5365	-82070.5	0.00
...
OM (21)	-109819.8449	-109820.0	0.00
OM (22)	-110829.5796	-110830.0	0.00
...
OM (31)	10778.2649	10778.3	0.00
OM (32)	18872.9115	18872.9	0.00
...
OM (41)	82070.5365	82070.5	0.00
OM (42)	86679.6849	86679.7	0.00
...

OM (51)	110829.5796	110830.0	0.00
OM (52)	111590.0014	111590.0	0.00
...
OM (61)	-0.0000	4.66649e-12	0.00
OM (62)	-48800.9241	-48800.9	0.00
OM (63)	50648.5827	50648.6	0.00
OM (64)	-0.0000	0.000000000129965	0.00
OM (65)	-50648.5827	-50648.6	0.00
OM (66)	55448.5827	55448.6	0.00
OM (67)	-0.0000	0.0000000000135408	0.00
OM (68)	-55448.5827	-55448.6	0.00
OMS (1)	0.0000	-6.59758e-11	0.00
OMS (2)	-332.2214	-332.221	0.00
OMS (3)	-546.5194	-546.519	0.00
OMS (4)	-1043.2274	-1043.23	0.00
OMS (5)	-1152.4316	-1152.43	0.00
OMS (6)	-1174.2851	-1174.29	0.00
OMS (7)	-1134.8915	-1134.89	0.00
OMS (8)	-538.1128	-538.113	0.00
OMS (9)	88.9537	88.9537	0.00
OMS (10)	650.8994	650.899	0.00
OMS (11)	1544.1325	1544.13	0.00
OMS (12)	2267.5201	2267.52	0.00
...
OMS (21)	12307.2323	12307.2	0.00
OMS (22)	13643.2758	13643.3	0.00
...
OMS (31)	332.2214	332.221	0.00
OMS (32)	546.5194	546.519	0.00
...
OMS (41)	-2267.5201	-2267.52	0.00
OMS (42)	-3302.9004	-3302.9	0.00
...
OMS (51)	-13643.2758	-13643.3	0.00
OMS (52)	-15228.6322	-15228.6	0.00

...
OMS (61)	0.0000	-6.13031e-11	0.00
OMS (62)	6178.2523	6178.25	0.00
OMS (63)	-4330.5937	-4330.59	0.00
OMS (64)	-0.0000	6.40176e-11	0.00
OMS (65)	4330.5937	4330.59	0.00
OMS (66)	469.4063	469.406	0.00
OMS (67)	-0.0000	6.94577e-11	0.00
OMS (68)	-469.4063	-469.406	0.00
OMM (1)	0.0000	-1.1558e-11	0.00
OMM (2)	-2161.2771	-2161.28	0.00
OMM (3)	-3755.3890	-3755.39	0.00
OMM (4)	-7460.9667	-7460.97	0.00
OMM (5)	-8507.1608	-8507.16	0.00
OMM (6)	-9735.5494	-9735.55	0.00
OMM (7)	-10761.5005	-10761.5	0.00
OMM (8)	-11416.1810	-11416.2	0.00
OMM (9)	-11873.7124	-11873.7	0.00
OMM (10)	-12184.5792	-12184.6	0.00
OMM (11)	-12427.2860	-12427.3	0.00
OMM (12)	-12499.6981	-12499.7	0.00
...
OMM (21)	-9076.6751	-9076.68	0.00
OMM (22)	-8151.3670	-8151.37	0.00
...
OMM (31)	2161.2771	2161.28	0.00
OMM (32)	3755.3890	3755.39	0.00
...
OMM (41)	12499.6981	12499.7	0.00
OMM (42)	12452.6496	12452.6	0.00
...
OMM (51)	8151.3670	8151.37	0.00
OMM (52)	6976.7459	6976.75	0.00
...
OMM (61)	0.0000	-3.10336e-11	0.00

OMM (62)	-3448.3567	-3448.36	0.00
OMM (63)	5296.0153	5296.02	0.00
OMM (64)	0.0000	8.80226e-12	0.00
OMM (65)	-5296.0153	-5296.02	0.00
OMM (66)	10096.0153	10096.0	0.00
OMM (67)	-0.0000	2.45456e-11	0.00
OMM (68)	-10096.0153	-10096.0	0.00
TS (1)	252.95863	252.959	0.00
...
TS (6)	252.95863	252.959	0.00
TS (7)	220.48677	220.487	0.00
...
TS(15)	220.48677	220.487	0.00
TS(16)	300.51417	300.514	0.00
...
TS(28)	300.51417	300.514	0.00
TS(29)	0.00000	0	0.00
TS(30)	300.51417	300.514	0.00
...
TS(33)	300.51417	300.514	0.00
TS(34)	0.00000	0	0.00
TS(35)	300.51417	300.514	0.00
...
TS(47)	300.51417	300.514	0.00
TS(48)	220.48677	220.487	0.00
...
TS(55)	220.48677	220.487	0.00
TS(57)	252.95863	252.959	0.00
...
TS(62)	252.95863	252.959	0.00
TS(63)	-80.02740	-80.0274	0.00
TS(64)	-47.55554	-47.5555	0.00
TS(65)	-47.55554	-47.5555	0.00
TS(66)	-80.02740	-80.0274	0.00
TS(67)	-32.47186	-32.4719	0.00

TS(68)	0.00000	0	0.00
TS(69)	32.47186	32.4719	0.00
TS(70)	0.00000	0	0.00
TS(71)	0.00000	0	0.00
TS(72)	0.00000	0	0.00

5. Diskussion

Die Berechnungsgenauigkeit stimmt sehr gut mit dem Matlab MuPad Notebook „QUEBER_Version2016“ überein, vgl. Unterkapitel 4.3. Vektor- und Matrizenrechnung mit NumPy in Python ist somit für diese Aufgabenstellung sehr gut geeignet.

Ggf. ist eine Anwendung von *NUM_NULL* auf weitere Berechnungsergebnisse im Python-Skript möglich. Bzw. verschiedene Berechnungen erfordern eventuell eigene Grenzen für das Nullsetzen von Werten. Dabei ist eine Abhängigkeit von *NUM_NULL* zu der Größenordnung z.B. von Knotenkoordinaten vorstellbar.

Im „Graph Plot Menu“ des Programms besteht keine Möglichkeit zur Anpassung von Fenstergröße und Skalierung von Wölbwerten. Die Werte sind jetzt manuell im Modul *output.py*, Funktion *output_menu* mit den Werten *WINDOW_SIZE*, *ASPECT_RATIO*, *SCALE_FACTOR_OM* anpassbar (Zeilen 190-192).

Bei Grafiken sind Werte der Wölfunktion für jedes angrenzende Element einmal eingetragen. Dabei erscheint die Grafik -besonders in 2D- oft unübersichtlich, vgl. Abbildung 36 in Unterkapitel 4.2. Ein Wert pro Knoten reicht daher aus. Eine Implementierung einer solchen Funktionalität ist nicht vorhanden.

3D-Grafiken haben im Python-Programm nicht die gleiche Achsenkalierung. Es ist keine automatische Skalierung wie bei 2D-Grafiken möglich, siehe Abbildung 38. Eine mögliche Lösung besteht in einer Anpassung der Wertebereiche mit Parametern wie im C#-Sharp-Programm „Thesis_Interface“, vgl. Abbildung 25 in Unterkapitel 3.2.2. Dort ist allerdings mit dem verwendeten Vorgehen keine exakte Gleichskalierung der Achsen erreicht worden, nur eine Näherung.

```
NotImplementedError: Axes3D
currently only supports the aspect
argument 'auto'. You passed in 1.
```

```
def configure_2D_figure(fig):
    plt.gca().set_aspect('equal', adjustable='box')
```

Abbildung 38: Fehlermeldung und Quellcode für Skalierung von Achsen in 2D- und 3D-Grafiken des Python-Programms

Zum Einlesen einer anderen Eingabedatei ist nach Ausführen des Programms mit *Solve*-Button ein Neustart des Programms erforderlich. Ansonsten sind die neuen Daten nicht im Programm verfügbar.

Der Dateipfad der Ausgabedatei ist automatisch im Verzeichnis der Eingabedatei. Der Dateiname der Ausgabedatei wird ebenfalls automatisch erzeugt. Hier bietet sich ein Einfügen von Optionen an.

Im Programm sind keine Kommentare bzw. docstrings vorhanden. Damit ist keine automatische Erstellung einer Dokumentation möglich, vgl. Anhang 8.2.9. Dafür ist der Quellcode übersichtlich.

Es wird vorwiegend ein funktionaler Programmierstil verwendet und keine Unterteilung in Klassen (vgl. Anhang 8.2.1 und 8.2.5). In Python sind allerdings alle Daten und Bestandteile objektorientiert angelegt (z.B. Datentypen wie `str` oder `np.array`)

Das Programm verwendet möglichst keine side effects. Effekte von Funktionen sind (bis auf wenige Ausnahmen, z.B. Plots in Grafiken) durch Rückgabewerte erkennbar. Die Übergabe von verwendeten Werten als Parameter macht diese unabhängig vom Namespace des Funktionsaufrufs. Ein Stück weit sind docstrings bzw. Kommentare somit nicht erforderlich, da Effekte und Parameter in der Funktionsdefinition klar erkennbar sind.

Weiterhin besteht die Möglichkeit, das Programm als Jupyter Notebook oder als ausführbare exe.Datei zu konzipieren. Bisher ist ein in einer IDE wie z.B. Spyder ausführbare Version vorhanden. Möglichkeiten zur Erstellung von exe.Dateien aus Python-Skripten liefern auto-py-to-exe (Python 2022b) und pyinstaller (Andrade 2021). Für Programme aus mehreren Modulen ist die Anwendung mithilfe von auto-py-to-exe oder pyinstaller für mich nicht klar ersichtlich.

Im Programm ist als bevorzugte Sprache englisch gewählt. Einige Formelzeichen sind zur Wahrung der Übereinstimmung mit „QUEBER_Version2016“ in deutsch beibehalten (z.B. DICKE, STAB).

6. Zusammenfassung und Ausblick

Diese Masterarbeit umfasst Design, Anfertigung und Dokumentation eines Python-Programms. Dieses heißt „Thin-Walled Cross Section Properties“. Eingangsdaten definieren eindeutig Geometrie und Lage eines Querschnitts aus Knoten und Elementen. Koordinaten im beliebig gewählten Ursprungs-KS definieren einen Knoten. Anfangs- und Endknoten sowie Dicke definieren ein Element. Damit ist eine Berechnung und Ausgabe von Querschnittswerten möglich. Diese gelten für dünnwandige Querschnitte. Berechnete Querschnittswerte sind Flächenmomente 0., 1. und 2. Ordnung, Werte im Hauptachsensystem sowie Koordinaten von Schubmittelpunkt (SMP) M und Flächenschwerpunkt (FSP) S . Flächenmomente werden auch in Bezug auf M und S berechnet. Weiterhin beinhaltet das Python-Programm eine Berechnung von Wölfunktionen (ebenfalls im Ursprungs-KS, FSP-KS und SMP-KS). Eine grafische Ausgabe von Geometrie und Wölfunktionen ist in 2D und 3D möglich. Weiterhin gibt das Python-Programm eine Textdatei mit Berechnungsergebnissen zurück. Einheiten sind nicht angegeben. Daher erfolgen Berechnungen mit Dimensionen von Größen. Eingabe- und AusgabefORMAT sowie Berechnungsablauf entsprechen dem Matlab MuPad Notebook „QUEBER_Version2016“ (Stanoev 2016).

Es bestehen einige Erweiterungsmöglichkeiten, z.B. zusätzliche Funktionen bzw. Optionen zum Einlesen weiterer Eingabeformate. Damit ist eventuell eine Berechnung mit Eingabedateien wie in C#-Programm „Thesis_Interface“ (Samlaf-Adams 2020) möglich. Wenn Materialeigenschaften angegeben sind, ist nach Hinzufügen einiger Funktionen auch eine Ausgabe von Steifigkeitswerten des Querschnitts möglich. Damit besteht die Option, nach Berechnung von Querschnittswerten mehrerer Querschnitte eine Berechnung von Eigenmoden wie in „Thesis_Interface“ anzufügen. Ein solches Vorgehen ist automatisierbar für mehrere Eingabedateien. Dafür bietet sich eventuell eine Anpassung von Eingabe- und AusgabefORMAT des Python-Programms an. Bei größerem Umfang des Programms ist eine automatisierte Dokumentation sinnvoll. Für Grafiken sind Optionen denkbar, z.B. zum Einstellen der Skalierung von Wölfunktionen.

Anhänge:

- Python-Programm „Thin-Walled Cross Section Properties“
- C#-Programm „Thesis_Interface“ (mit Implementierung)

7. Literaturverzeichnis

7.1 Bücher

- Hofstetter, G. / Mang, H. (2013), Festigkeitslehre. 4. Auflage, Berlin.
- Klein, B. (2019), Numerisches Python – Arbeiten mit NumPy, Matplotlib und Pandas. 1. Auflage, München.
- Linke, M. / Nast, E. (2015), Festigkeitslehre für den Leichtbau – Ein Lehrbuch zur Technischen Mechanik. 1. Auflage, Berlin.
- Mittelstedt, C. (2021), Rechenmethoden des Leichtbaus – Grundlagen, Stäbe und Balken, Energiemethoden. 1. Auflage, Berlin.
- Nasdala, L. (2015), FEM-Formelsammlung Statik und Dynamik – Hintergrundinformationen, Tipps und Tricks. 3. Auflage, Wiesbaden.
- Natt, O. (2020), Physik mit Python – Simulationen, Visualisierungen und Animationen von Anfang an. 1. Auflage, Berlin.
- Öchsner, A. (2021), Classical Beam Theories of Structural Mechanics. 1. Auflage, Cham, Schweiz.
- Petersen, C. (2013), Stahlbau – Grundlagen der Berechnung und baulichen Ausbildung von Stahlbauten. 4. Auflage, Berlin.
- Schäfer, C. (2019), Schnellstart Python – Ein Einstieg ins Programmieren für MINT-Studierende. 1. Auflage, Wiesbaden.
- Selke, P. (2013), Höhere Festigkeitslehre – Grundlagen und Anwendung. 1. Auflage. München.
- Slatkin, B. (2020), Effektiv Python programmieren – 90 Wege für bessere Python-Programme. 2. Auflage, Frechen.
- Spura, C. (2019), Einführung in die Balkentheorie nach Timoshenko und Euler-Bernoulli. 1. Auflage. Wiesbaden.
- Steyer, R. (2018), Programmierung in Python – Ein kompakter Einstieg für die Praxis. 1. Auflage, Wiesbaden.
- Werkle, H. (2008), Finite Elemente in der Baustatik – Statik und Dynamik der Stab- und Flächenträgerwerke. 3. Auflage, Wiesbaden.
- Woyand, H. (2019), Python für Ingenieure und Naturwissenschaftler – Einführung in die Programmierung, mathematische Anwendungen und Visualisierungen, 3. Auflage, München.

7.2 Universitätsschriften

- Nan Li, (k.A.), Eingabedatei für „Blade Cross-Section“-Berechnung im C-Sharp-Programm „Thesis Interface“. Textdatei “bladeod200.txt” Universität Rostock, Fakultät MSF, Lehrstuhl für Windenergietechnik.
- Peters, H. (k.A.), Eingabedatei für Matlab MuPad Notebook “QUEBER_Version2016”. Textdatei “FR-01-VAR1neu.txt” Universität Rostock, Fakultät MSF.
- Samlafo-Adams, A. (2020), Master Thesis - Enhancements in the computer tool RBCsharp for calculating the natural vibration of wind turbine rotor blades with cross section data preprocessor (inklusive C# Programm: “Thesis_Interface”). Masterarbeit Universität Rostock, Fakultät MSF, Lehrstuhl für Windenergietechnik.
- Stanoev, E. (2013), Dünnewandige Stabsysteme. Skript Vorlesung “Skript_5-6_DwSS.pdf” Universität Rostock, Fakultät MSF, Lehrstuhl für Schiffstechnische Konstruktionen.
- Stanoev, E. / Schwarz, S. / Zhengkun Liu / Nan Li (2016), COMPUTERPROGRAMM QUEBER: Geometrische Werte dünnwandiger Querschnitte – Berechnung der Querschnittswerte eines offenen / (mehrzelligen) geschlossenen Profils (ohne Eingabe der Zellen). Matlab MuPad Notebook “QUEBER_Version2016” Universität Rostock, Fakultät MSF, Lehrstuhl für Windenergietechnik.
- Stanoev, E. (k.A.), Projekt Querschnittswerteprogramm – PM 02 Bauinformatik, Master BIW. Word-Dokumente “Querschnittswerte.doc” Universität Rostock, Agrar- und Umweltwissenschaftliche Fakultät.
- Stanoev, E. (k.A. a), Eingabedatei für Matlab MuPad Notebook „QUEBER_Version2016“. Textdatei “3kasten_Spkt.txt” Universität Rostock, Fakultät MSF.
- Vent, H. (2022), Grafische 2D-Darstellung der Rotorblatt-Querschnitte einer WEA (inklusive Programmerweiterung (QuerschnittGraph2D) in C# Programm “Thesis_Interface”). Studienarbeit Universität Rostock, Fakultät MSF, Lehrstuhl für Windenergietechnik.
- Wunderlich (k.A.), Eingabedatei für Matlab MuPad Notebook “QUEBER_Version2016” (Stanoev 2016). Textdatei “Profil_Wunderlich.txt” Universität Rostock, Fakultät MSF.

7.3 Internet-Quellen

- Amos, D. (2020), Object-Oriented Programming (OOP) in Python 3, <https://realpython.com/python3-object-oriented-programming/>, 08.08.2022
- Anaconda Inc. (2022), Anaconda Inc. Homepage, <https://www.anaconda.com>, 09.08.2022.
- Anaconda Inc. (2022a), Pysimplegui :: Anaconda.org, <https://anaconda.org/conda-forge/pysimplegui>, 10.10.2022.
- Andrade, F. (2021), How to Easily Convert a Python Script to an Executable File (.exe) | by Frank Andrade | Towards Data Science, 19.11.2022.
- Bhalla, H. (2018), Why Python is called Dynamically Typed – GeeksforGeeks, <https://www.geeksforgeeks.org/why-python-is-called-dynamically-typed/>, 10.08.2022.
- Christie, T. / MkDocs Team (2014), MkDocs – Homepage, <https://www.mkdocs.org>, 16.08.2022.
- Christie, T. / MkDocs Team (2014a), Getting Started – MkDocs, <https://www.mkdocs.org/getting-started/>, 16.08.2022.
- Codeacademy (2022), Codeacademy Homepage, <https://www.codecademy.com>, 09.08.2022.
- Cone, M. (2022), Markdown Guide – Homepage, <https://www.markdownguide.org>, 16.08.2022.
- Enthought Inc. (2022), Enthought Inc. Homepage, <https://www.enthought.com>, 09.08.2022.
- Enthought Inc. (2022a), Enthought Downloads – Enthought Deployment Manager (EDM), <https://assets.enthought.com/downloads/>, 09.08.2022.
- Finxter, C. (2022), Finxter – Create Your Thriving Coding Business, Subscribed Newsletter „Python Freelancer Course“, <https://blog.finxter.com>, 10.08.2022.
- Frenzel, S. (2022), Schubflächenberechnung mit DUENQ | Dlubal Software, <https://www.dlubal.com/de/support-und-schulungen/support/knowledge-base/001405>, 04.09.2022.
- Github Inc. (2019), Github – PyCraftDeveloper/Documentation-Builder at pythonawesome.com, <https://github.com/PycraftDeveloper/Documentation-Builder?ref=pythonawesome.com>, 17.08.2022.
- Github Inc. (2022), Github – PySimpleGUI/DemoPrograms/Demo_Matplotlib_EMBEDDED_Toolbar.py, https://github.com/PySimpleGUI/PySimpleGUI/blob/master/DemoPrograms/Demo_Matplotlib_EMBEDDED_Toolbar.py, 15.11.2022.

- Goodger, D. / van Rossum, G. (2022), PEP 257 – Docstring Conventions, <https://peps.python.org/pep-0257/>, 16.08.2022.
- Hammerly, A. (2017), 4 Programming Paradigms In 40 Minutes – RubyConf 2017, <https://www.youtube.com/watch?v=cgVVZMfLjEI>, 08.08.2022.
- IPython development team (k.A.), Jupiter and the future of IPython – IPython Homepage, <https://ipython.org>, 09.08.2022.
- JavaTpoint (2021), Software Engineering Tutorial – javatpoint, <https://www.javatpoint.com/software-engineering-tutorial>, 19.08.2022.
- JavaTpoint (2021a), Software Processes – javatpoint, <https://www.javatpoint.com/software-processes>, 19.08.2022.
- JavaTpoint (2021b), Waterfall Model (Software Engineering) – javatpoint, <https://www.javatpoint.com/software-engineering-waterfall-model>, 19.08.2022.
- JavaTpoint (2021c), Software Engineering | Software Design Principles – javatpoint, <https://www.javatpoint.com/software-engineering-software-design-principles#:~:text=Following%20are%20the%20principles%20of%20Software%20Design%201,3%20Modularity.%20...%204%20Strategy%20of%20Design.%20>, 19.08.2022.
- JetBrains s.r.o. (2022), JetBrains for Education – Keep Evolving, <https://www.jetbrains.com/education/>, 09.08.2022.
- JetBrains s.r.o. (2022a), PyCharm – the Python IDE for Professional Developers by JetBrains, <https://www.jetbrains.com/pycharm/>, 09.08.2022.
- Jones, D. (2021), Simplify Python GUI Development With PySimpleGUI – Real Python, <https://realpython.com/courses/simplify-gui-dev-pysimplegui/>, 17.08.2022.
- The Matplotlib development team (2022), Matplotlib Homepage – Visualization with Python, <http://matplotlib.org>, 06.07.2022.
- The Matplotlib development team (2022a), Getting Started – Matplotlib 3.5.3 documentation, https://matplotlib.org/stable/users/getting_started/index.html#where-to-go-next, 16.08.2022.
- The Matplotlib development team (2022b), Plot Types – Matplotlib 3.5.3 documentation, https://matplotlib.org/stable/plot_types/index.html, 16.08.2022.
- The Matplotlib development team (2022c), Matplotlib cheatsheets – Visualization with Python, <https://matplotlib.org/cheatsheets/>, 16.08.2022.
- The Matplotlib development team (2022d), Basis Usage – Matplotlib 3.5.3 documentation, <https://matplotlib.org/stable/tutorials/introductory/usage.html#sphx-glr-tutorials-introductory-usage-py>, 16.08.2022.

- The Matplotlib development team (2022e), Embedding in Tk – Matplotlib 3.5.3 documentation, https://matplotlib.org/stable/gallery/user_interfaces/embedding_in_tk_sgskip.html#sphx-glr-gallery-user-interfaces-embedding-in-tk-sgskip-py, 16.08.2022.
- NumPy (2022), NumPy Homepage, <https://numpy.org>, 06.07.2022.
- OpenTechSchool (2014), Python for beginners, https://opentechschool.github.io/python-beginners/en/getting_started.html#what-is-python-exactly, 09.08.2022.
- Peter, T. (2004), PEP 20 – The Zen of Python, <https://peps.python.org/pep-0020/> 10.08.2022.
- Project Jupyter Community (2022), Project Jupyter Homepage, <https://jupyter.org> 10.08.2022.
- PyPA (2020), Python Packaging User Guide – Tutorials – Installing Packages, <https://packaging.python.org/en/latest/tutorials/installing-packages/>, 09.08.2022.
- PySimple GUI Tech LLC (2022), PySimpleGUI – Homepage, <https://www.pysimplegui.org/en/latest/>, 17.08.2022.
- PySimpleGUI Tech LLC (2022a), PySimpleGUI – PyPI, <https://pypi.org/project/PySimpleGUI/>, 17.08.2022.
- Python Software Foundation (2022), Offizielle Website zur Programmiersprache Python, <https://www.python.org>, 07.08.2022.
- Python Software Foundation (2022a), autopep8 – PyPi, <https://pypi.org/project/autopep8/>, 15.08.2022.
- Python Software Foundation (2022b), auto-py-to-exe PyPi, 19.11.2022.
- pythontutorial.net (2022), A Basic Guide to Python Boolean Data Types, Falsy and Truthy Values, <https://www.pythontutorial.net/python-basics/python-boolean/#:~:text=To%20represent%20true%20and%20false%2C%20Python%20provides%20you,data%20type%20has%20two%20values%3A%20True%20and%20False.>, 10.08.2022.
- Python Wiki (2012), Why is Python a dynamic language and also a strongly typed language – Python Wiki, <https://wiki.python.org/moin/Why%20is%20Python%20a%20dynamic%20language%20and%20also%20a%20strongly%20typed%20language>, 10.08.2022.
- Raybaut, P. et al.(2021), WinPython – The easiest way to run Python, Spyder with SciPy and friends out of the box on any Windows PC, without installing anything!, <https://winpython.github.io>, 09.08.2022.
- Read the Docs, Inc. & contributors (2022), Home | Read the Docs, <https://readthedocs.org>, 20.08.2022.

- Real Python (2022), Real Python – Homepage, <https://realpython.com>, 09.08.2022.
- Refsnes Data (2022), Python Tutorial – Python Home - w3schools.com, <https://www.w3schools.com/python/default.asp>, 12.08.2022.
- The SciPy Community (2022), PEP 8 and SciPy – SciPy v1.9.0 Manual, <https://docs.scipy.org/doc/scipy/dev/contributor/pep8.html>, 15.08.2022.
- the Sphinx developers (2022), Welcome - Sphinx documentation, <https://www.sphinx-doc.org/en/master/#>, 20.08.2022.
- Spyder Website Contributors (2022), Spyder IDE Homepage, <https://www.spyder-ide.org>, 09.0.2022.
- Stapleton, I. (2016), Flake8: Your Tool For Style Guide Enforcement – flake8 5.0.4 documentation, <https://flake8.pycqa.org/en/latest/>, 15.08.2022.
- TechVidvan (2022), Errors and Exceptions in Python – TechVidvan, <https://techvidvan.com/tutorials/errors-and-exceptions-in-python/#:~:text=Exceptions%20in%20python%20are%20the%20errors%20that%20occur,displays%20a%20runtime%20exception%20message%20on%20the%20screen.>, 20.08.2022.
- TechVidvan (2022a), Python Exception Handling – Learn errors and exceptions in Python - TechVidvan, <https://techvidvan.com/tutorials/errors-and-exceptions-in-python/#:~:text=Exceptions%20in%20python%20are%20the%20errors%20that%20occur,displays%20a%20runtime%20exception%20message%20on%20the%20screen.>, 20.08.2022.
- Tech With Tim (2020), Software Design Tutorial #1 – Software Engineering & Software Architecture, <https://www.youtube.com/watch?v=FLtqAi7WNBY>, 19.8.2022.
- Tech With Tim (2020a), Software Design Tutorial #2 – Implementing Our Design, <https://www.youtube.com/watch?v=6thjSbJcoUc>, 19.08.2022.
- Tech With Tim (2021), 10 Design Principles For Software Engineers, <https://www.youtube.com/watch?v=XQzEo1qag4A>, 19.08.2022.
- TIOBE Software BV (2022), TIOBE Index for August 2022 – August Headline: Python going through the roof, <https://www.tiobe.com/tiobe-index/>, 09.08.2022.
- van Rossum, G. / Warsaw, B. / Coghlan, N. (2022), PEP 8 - Style Guide for Python Code, <https://peps.python.org/pep-0008/>, 07.08.2022.
- Wellesley College (2022), CS111 – Debugging Techniques, pdf-Dokument, <https://cs111.wellesley.edu/content/reference/debugging.pdf>, 20.08.2022.
- Wikipedia (2022b), Trägheitsradius, <https://de.wikipedia.org/wiki/Trägheitsradius>, 18.07.2022.

Wikipedia (2022c), Programmablaufplan,
<https://de.wikipedia.org/wiki/Programmablaufplan>, 27.09.2022.

8. Anhang

8.1 Grundlagen und Zielsetzungen

8.1.1 Grundprinzipien der Finite-Elemente-Methode

Eine infinitesimale mathematische Betrachtung des Systems ist hauptsächlich auf einfache Geometrien anwendbar. Das führt zu einer exakten Lösung mithilfe einer unendlichen Anzahl von Elementen bzw. von Differentialgleichungen. Dieses Prinzip behandelt stetige Probleme. Für komplexe Geometrien und Aufgaben stößt diese Vorgehensweise an ihre Grenzen.

„Wir haben gelernt, dass es nicht möglich ist, die stark verknüpften Zusammenhänge der Festigkeitslehre als Ganzes zu erfassen, geschweige denn zu lösen. Ein Weg zur Lösung besteht in der Diskretisierung, der Verwendung einer endlichen Zahl genau definierter Elemente des zu untersuchenden Systems. Das bezeichnet man dann als diskretes Problem.“ (Selke 2013), S. 231
Der diskrete Ansatz liefert Approximationen der Lösung. Die Geometrie besteht für eine möglichst genaue Lösung aus möglichst vielen kleinen einfachen Elementen. Gemeinsame Knoten verbinden die Elemente.

Die Finite-Elemente-Methode (FEM) basiert auf dem Prinzip der virtuellen Verrückung. Für die bereits vereinfachte Geometrie entsteht eine nochmals approximierte Lösung der Differentialgleichungen. Knotenkräfte F_i stellen die Belastungen dar. Sie führen zu einer Verformung des Systems. Ansatzfunktionen für die Verrückungen der Knoten $u_i = f(F_i)$ beschreiben die Verformung. Dazu bilden die zusammengefassten Elementsteifigkeiten eine Gesamtsteifigkeit des Systems ab. Die Verzerrungen und damit die Spannungen (aus Materialgesetzen) in den Elementen folgen daraus.

Ein lineares Gleichungssystem ersetzt die Differentialgleichungen. Die Linearisierung erfolgt z.B. mit einer Taylor-Entwicklung.

(Selke 2013), S. 231-233

Die meisten Prozesse und damit auch Computerprogramme bestehen aus drei grundlegenden Teilen: Eingabe – Verarbeitung – Ausgabe / Preprozessor – Solver – Postprozessor. Ein FEM-Programm umfasst analog dazu die Schritte „Modellieren – Berechnen – Ergebnisdarstellung“. (Selke 2013), S. 235

Die Modellierung (Preprozessor) erfolgt in einer grafischen Benutzeroberfläche oder mithilfe einer Eingabedatei (z.B. Textdatei). Das reale System wird diskretisiert und Elemente und Knoten definiert. In diesem Schritt ist der/die

Anwender/in aktiv. Eine Haupteinflussmöglichkeit auf die Qualität der Berechnung ist die Wahl geeigneter Elementtypen und -größen sowie die Auswahl der zu den Elementtypen und Belastungsarten passenden Ansatzfunktion(en). Die erforderlichen Daten für eine FEM-Berechnung sind:

- Koordinaten der Knoten
- Lagerart der Knoten (z.B. feste Einspannung oder Drehgelenk)
- Auflagerbedingungen (Erhöhung der Lagerwertigkeit an ausgewählten Knoten)
- Geometrie der Elemente (Elementtyp, Maße)
- Materialeigenschaften der Elemente (E -Modul, Schubmodul G , Dichte ρ)
- Belastungen (als Knotenkräfte)
- Lastfallüberlagerungsvorschrift
- Bemessungskennwerte (Sicherheitsfaktoren, Belastbarkeit des Materials)

Eine FEM-Berechnung umfasst die - vom Computer ausgeführten - folgenden Schritte:

- „Aufbau der Elementsteifigkeitsmatrizen und Elementlastvektoren
- Aufbau der Systemsteifigkeitsmatrix und des Lastvektors
- Lösung des Gleichungssystems $[u_i = f(F_i)]$, Anm. des Verfassers]
- Ermittlung der Elementspannungen bzw. -schnittgrößen“ (Werkle 2008), S. 551

Die Ergebnisdarstellung erfolgt in Form einer Datei (z.B. Textdatei) oder einer grafischen Ausgabe, z.B von Verformungen oder Spannungen. Dabei prüft der/die Anwender/in die Plausibilität der Ergebnisse und Qualität von Modellbildung und Berechnung nach der Ergebnisausgabe.

(Selke 2013), S. 235-239 / (Werkle 2008), S. 551-552

Die Masterarbeit ordnet sich als Teil des Preprozessors mit der Definition von Knoten und Elementen ein. Belastungen und Lagerung sind nicht berücksichtigt. Die berechneten Querschnittswerte dienen dem Aufbau der Element- und Systemsteifigkeitsmatrizen im Solver.

Abbildung 39 stellt die Einbindung eines FEM-Programms in die Entwicklung bzw. Auslegung am Beispiel eines Tragwerks dar.

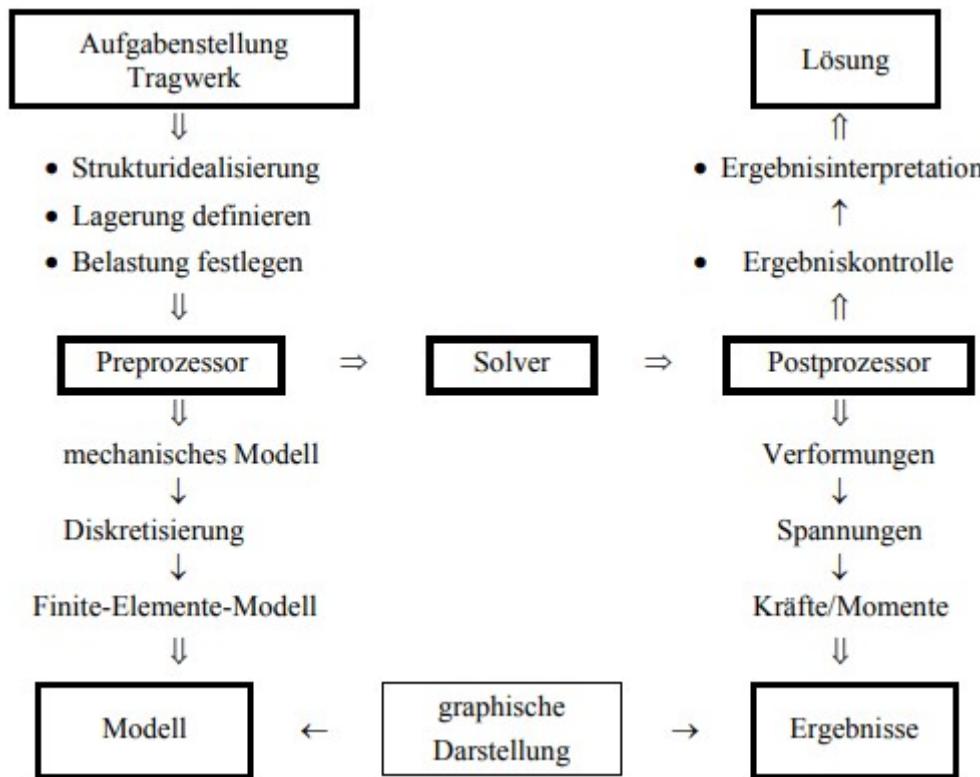


Abbildung 39: Arbeitsschritte zur Verwendung eines FEM-Programms im Entwicklungsprozess eines Tragwerks (Selke 2013), S. 236

Es gibt zwei grundlegende Programmier-Paradigmen in FEM-Programmen. Das Matlab-Programm „QUEBER_Version2016“ (Stanoev 2016) verwendet die prozedurale Ausführung der Berechnungsschritte. Das C#-Programm „Thesis_Interface“ (Samlaf-Adams 2020) / (Vent 2022) nutzt die Vorteile der objektorientierten Programmierung (OOP). Diese liegen in der besseren Verständlichkeit und Ordnung umfangreicher Programme.

Die zentrale Idee der OOP ist die Verwendung von Klassen. Eine Klasse bietet Raum für die allgemeine Definition von Daten und Methoden (Funktionen in einer Klasse). Sie funktioniert ähnlich wie ein Template als Formvorlage. Bei der Erzeugung eines Objektes stehen erstellte Klassen als Objekttyp zur Verfügung. Die Eigenschaften eines Objektes entsprechen zunächst denen seiner Klasse. Es ist eine Zuweisung von objektspezifischen Werten möglich. Ein Objekt verfügt ebenfalls über Methoden seiner Klasse.

Es folgt ein einfaches Beispiel für ein objektorientiertes FEM-Programm:

Klasse Element

- Daten zu Geometrie und Materialeigenschaften
- Methode Erzeuge_Element
- Methode Ermittle_Steifigkeitsmatrix
- Methode Ermittle_Spannungsmatrix

Klasse Gesamtsystem

- Systemsteifigkeitsmatrix
- Lastvektor
- Knotenverschiebungsgrößen
- Methode Erzeuge_Systemsteifigkeitsmatrix
- Methode Erzeuge_Systemlastvektor
- Methode Addiere_Elementsteifigkeitsmatrix
- Methode Addiere_Last
- Methode Berücksichtige_Lagerbedingungen
- Methode Löse_Gleichungssystem
- Methode Gib_Knotenverschiebungen_aus

Der Programmablauf des Beispiels sieht folgendermaßen aus:

Preprozessor

1. Erzeugung Objekt der Klasse Gesamtsystem
2. Klasse Element -→ Methode Erzeuge_Element (für jedes Element einmal ausführen)
3. alle Element-Objekte -→ Zuweisung von Eingabedaten zu Geometrie und Materialeigenschaften
4. Gesamtsystem-Objekt → Methode Berücksichtige_Lagerbedingungen
5. Gesamtsystem-Objekt → Methode Erzeuge_Systemlastvektor
6. Gesamtsystem-Objekt → Methode Addiere_Last

Solver

7. alle Element-Objekte → Methode Ermittle_Steifigkeitsmatrix
8. Gesamtsystem-Objekt → Methode Erzeuge_Systemsteifigkeitsmatrix
9. Gesamtsystem-Objekt → Methode Addiere_Elementsteifigkeitsmatrix
10. Gesamtsystem-Objekt → Methode Löse_Gleichungssystem
11. Gesamtsystem-Objekt → Methode Gib_Knotenverschiebungen_aus
12. alle Element-Objekte → Methode Ermittle_Spannungsmatrix

Postprozessor

Datenausgabe in Form von Grafiken und/oder Tabellen/Textdateien

(Werkle 2008), S. 555

8.1.2 Grundprinzipien der Elastostatik

Die berechneten Querschnittswerte beziehen sich z.B. auf einen Stab oder einen Balken. Die Geometrie ist dabei nicht das Unterscheidungsmerkmal. Der Unterschied besteht in der aufgebrachten Belastung. Auf einen Stab wirken ausschließlich Normalkräfte N (Zug/Druck) in der Schwerpunktslinie x des Stabes. Sie rufen als einzige Beanspruchung Normalsspannungen σ_{xx} hervor. Die benötigten Querschnittswerte sind die Querschnittsfläche A und der Elastizitätsmodul E des Materials. (Mittelstedt 2021), S. 202-203

Ein Balken erfährt als Belastungen Normalkräfte (Zug/Druck) N , Biegemomente M_{by} und M_{bz} sowie Querkräfte Q_y und Q_z . Entscheidende Querschnittswerte in der Berechnung sind Querschnittsfäche A , Flächenträgheitsmomente I_{yy} und I_{zz} bzw. Hauptträgheitsmomente I_1 und I_2 , dazu Elastizitätsmodul E , Schubmodul G und Schubkorrekturfaktoren κ_y und κ_z . Sie führen zu Dehnsteifigkeit EA , Biegesteifigkeiten EI_{yy} und EI_{zz} bzw. EI_1 und EI_2 sowie den Schubsteifigkeiten $\kappa_y GA = GA_{qy}$ und $\kappa_z GA = GA_{qz}$ des Querschnitts. Dabei sind $A_{qy} = \kappa_y A$ und $A_{qz} = \kappa_z A$ (A_{sy} und A_{sz}) die richtungsabhängigen Schubflächen des Querschnitts.

Die Balkentheorie unterscheidet die Aufstellung der Gleichgewichtsbedingungen in die Theorien 1., 2. und 3. Ordnung, siehe Abbildung 40. Die am häufigsten eingesetzte Theorie 1. Ordnung gilt am unverformten Balken. Die Verformungen (hier Durchbiegung w) sind klein gegenüber den Querschnittsabmessungen (hier Höhe h). Es treten nur lineare Betrachtungen auf. Für die Theorien 2. und 3. Ordnung gelten die Gleichgewichtsbedingungen am verformten Balken und die Verformungen sind größer gegenüber den Querschnittsabmessungen. Die Theorie 2. Ordnung findet v.a. bei Stabilitätsnachweisen und Knicken Verwendung. Es treten eventuell nichtlineare Betrachtungen auf. Die Theorie 3. Ordnung ist bei extremen Verformungen – z.B. Seilnetzen – anwendbar und enthält nichtlineare Betrachtungen des Balkens.

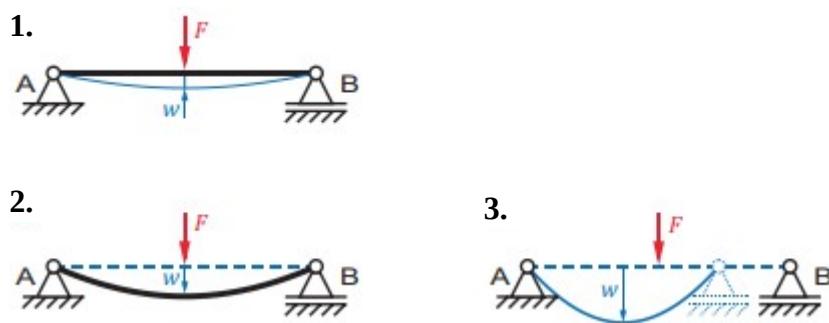


Abbildung 40: Beidseitig gelagerter Biegebalken nach Theorie 1., 2. und 3. Ordnung (Spura 2019), S. 2, bearb.

Der dargestellte allgemeine Berechnungsablauf in der Elastostatik bildet u.a. eine Grundlage für FEM-Berechnungen, siehe Abbildung 41. Eine Berechnung ist für statisch bestimmte und unbestimmte Systeme möglich.

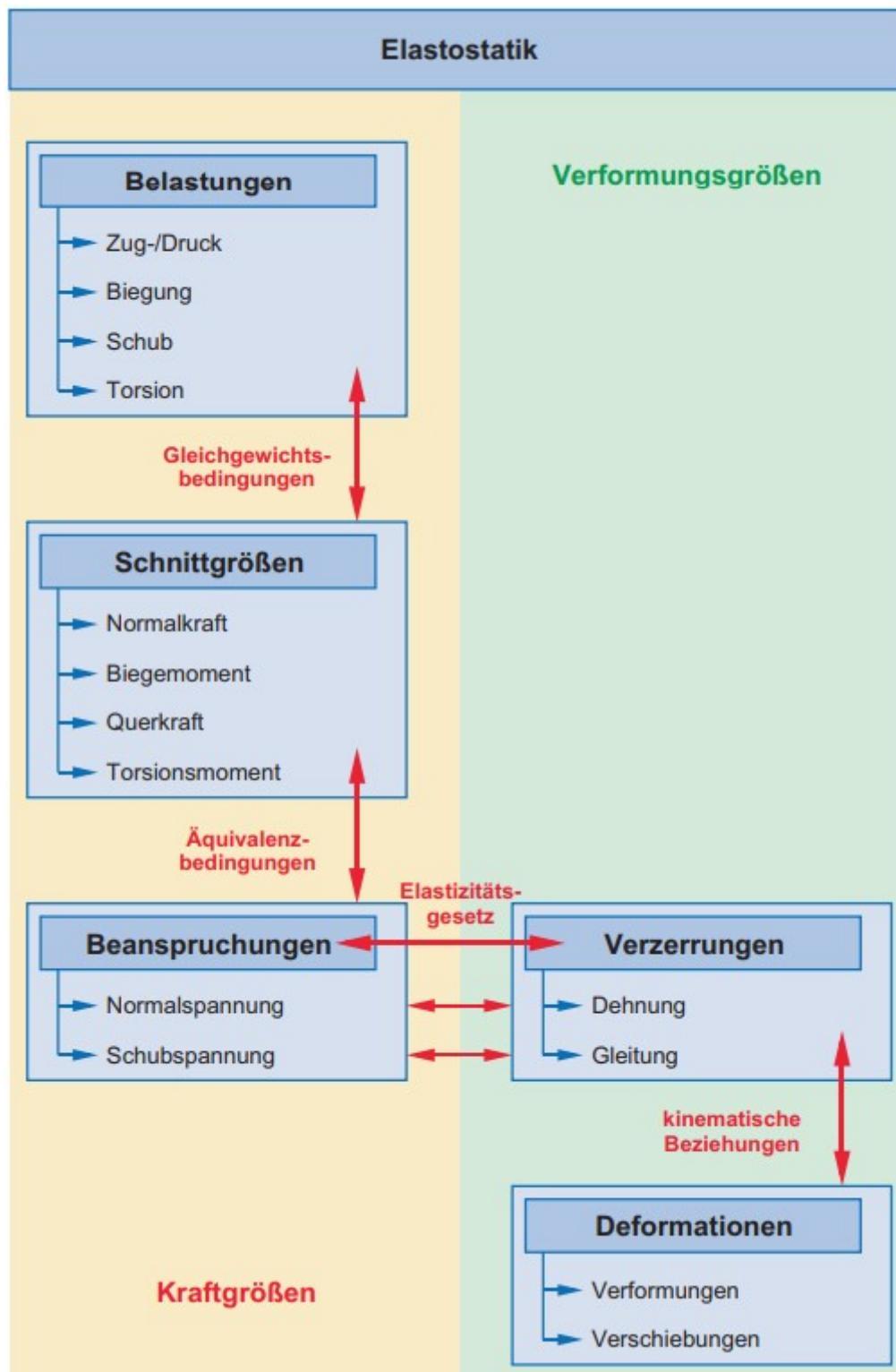


Abbildung 41: Kraft- und Verformungsgrößen der Elastostatik mit Umrechnungsmöglichkeiten (Spura 2019), S. 3

Die grundlegenden Belastungsarten in der Elastostatik sind Zug/Druck N , Biegung M_y , Schub Q_z und Torsion M_x bzw. T . Sie treten in Form von Einzelkräften und -momenten, Streckenlasten, Punkt-, Flächen- und Volumenkräften auf.

Gleichgewichtsbedingungen ergeben die Schnittgrößen (innere Kräfte und Momente) in einem Querschnitt des Bauteils. Zur Veranschaulichung dient ein mit einer Streckenlast $q_z(x)$ belasteter Balken in Abbildung 42. Zwei wichtige Gleichgewichtsbedingungen für die Schnittgrößen Querkraftschub $Q_z(x)$ und Biegemoment $M_y(x)$ stehen in Gleichung 8.1 und Gleichung 8.2.

$$Q'_z(x) = -q_z(x) \quad (\text{Gleichung 8.1})$$

$$M'_y(x) = Q_z(x) \quad (\text{Gleichung 8.2})$$

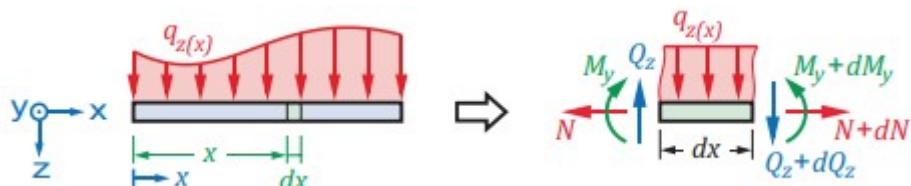


Abbildung 42: Schnittgrößen am Teilstück dx eines querkraftbelasteten Balkens
(Spura 2019), S. 7

Äquivalenzbedingungen drücken die Schnittgrößen eines Querschnitts als Beanspruchungen (Spannungen) aus. Die Schnittgrößen Querkraftschub $Q_z(x)$ und Biegemoment $M_y(x)$ sind mit der mittleren Schubspannung $\tau_m(x)$ bzw. der Normalspannung $\sigma_x(z)$ durch die beiden Äquivalenzbedingungen in Gleichung 8.3 und Gleichung 8.4 verbunden.

$$M_y(x) = \int z \sigma_x(z) dA \quad (\text{Gleichung 8.3})$$

$$Q_z(x) = \tau_m(x) \kappa A = \tau_m(x) A_s \quad (\text{Gleichung 8.4})$$

Mit dem Elastizitätsgesetz und Elastizitäts- bzw. Schubmodul E und G folgen aus den Beanspruchungen die Verzerrungen (Dehnung, Gleitung). Das Elastizitätsgesetz beinhaltet Gleichung 8.5 für die Dehnung $\varepsilon(x,z)$ und Gleichung 8.6 für die mittlere Gleitung $y_m(x)$.

$$\sigma(x,z) = E \varepsilon(x,z) \quad (\text{Gleichung 8.5})$$

$$\tau_m(x) = G y_m(x) \quad (\text{Gleichung 8.6})$$

Diese sind über kinematische Beziehungen mit den Deformationen (Verformungen, Verschiebungen) verknüpft, siehe Gleichung 8.7 und Gleichung 8.8. Sie beinhalten die Verdrehung $\psi(x)$ für die Drehung des Querschnitts zur Schwerpunktlinie x durch Biegung und die Steigung / Neigung $w'(x)$ der Schwerpunktlinie selbst.

$$\varepsilon(x, z) = \psi'(x)z \quad (\text{Gleichung 8.7})$$

$$\gamma_m(x) = \psi(x) + w'(x) \quad (\text{Gleichung 8.8})$$

Abbildung 43 zeigt einen ausschließlich mit einem Biegemoment M_{by} belasteten Balken. Die Querschnitte bleiben eben zur Schwerpunktlinie x . Der Querschnitt $A(x)$ erfährt eine Verdrehung $\psi(x)$, eine Durchbiegung $w(x)$ und eine Neigung $w'(x)$. Daraus resultiert die Verschiebung $u(x, z)$ eines Betrachtungspunktes im Querschnitt, siehe Gleichung 8.9. Die Verschiebung ist somit abhängig von der Dehnung $\varepsilon(x, z)$, siehe Gleichung 8.10. Ein Balken unter reiner Biegung gilt als Euler-Bernoulli-Balken.

$$u(x, z) = \psi(x)z = -w'(x)z \quad (\text{Gleichung 8.9})$$

$$\varepsilon(x, z) = u'(x, z) \quad (\text{Gleichung 8.10})$$

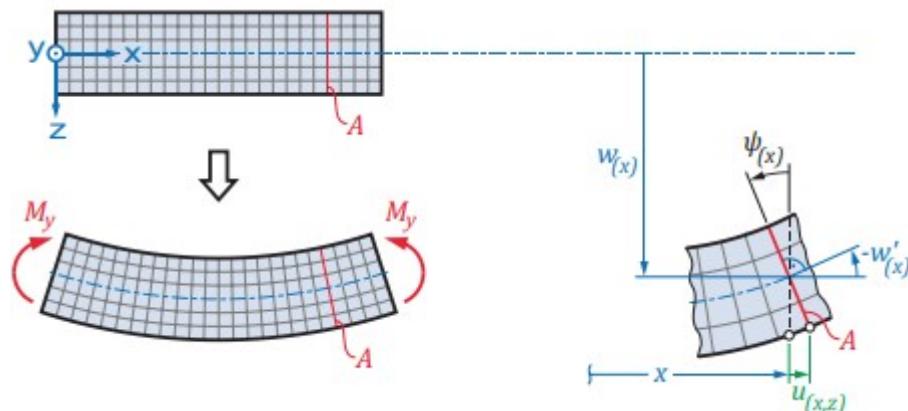


Abbildung 43: Verformung eines Balkens infolge von Biebelastung (Euler-Bernoulli-Balken) (Spura 2019), S. 10

Ein Balken erfährt ausschließlich eine Schubbelastung, siehe Abbildung 44. Die freien Oberflächen bleiben schubspannungsfrei. Daher stellt sich ein nichtlinearer Schubspannungsverlauf im Querschnitt ein (a). Dieser führt zu einer Verwölbung. Eine Vereinfachung der Berechnung liefert (b) mit im Querschnitt konstanten Werten für die mittlere Schubspannung $\tau_m(x)$ und die mittlere Gleitung $y_m(x)$. Der Querschnitt bleibt vereinfacht wölf frei und hat die Schubflächen A_{qy} und A_{qz} .

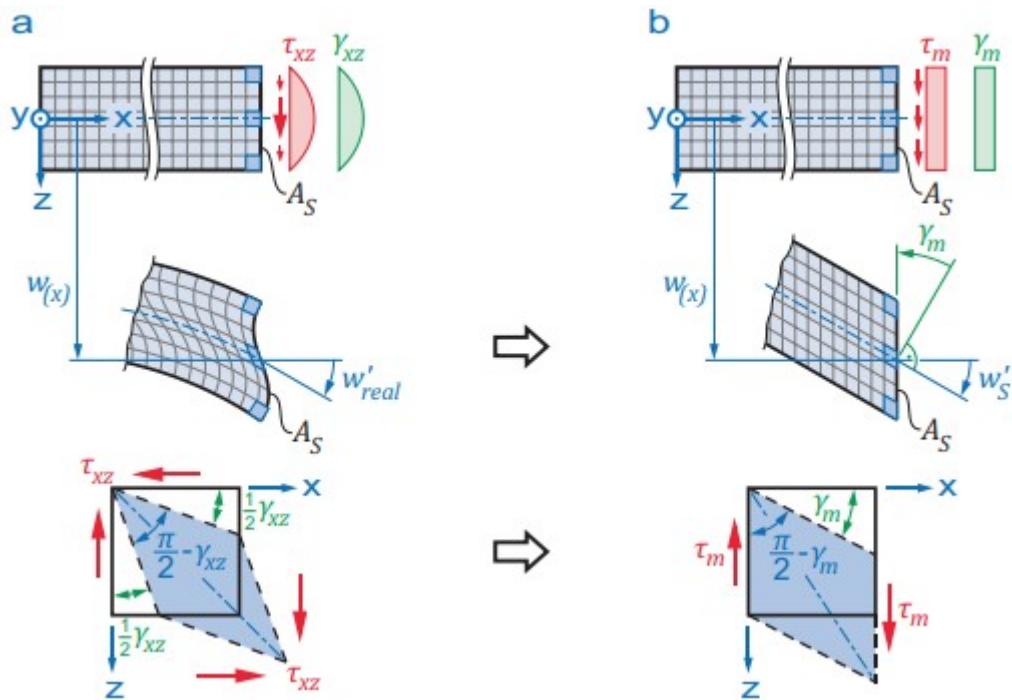


Abbildung 44: Gegenüberstellung von (a) realer Schubspannungsverteilung und Verwölbung und (b) Annahme des Ebenbleibens des Querschnitts ohne Verwölbung mit mittlerer Schubspannung (Spura 2019), S. 11

Die Balkentheorie gilt unter folgenden Annahmen:

- schlanker Balken, d.h. Balkenlänge $>>$ Querschnittsabmessungen ($I > 5b$, $I > 5h$)
- Schwerpunktlinie x und Querschnitt sind konstant bzw. nahezu konstant über gesamte Balkenlänge I (damit auch Biege- und Schubsteifigkeit)
- zweidimensionale Belastung (Kräfte in x - z -Ebene, Momente um y -Achse)
- kleine Verformungen ($v, w \ll h, b$ bzw. $v, w \ll I / 500$)
- Durchbiegung w in Richtung der z -Achse
- keine Torsion um Schwerpunktlinie x
- Normal- und Schubspannungen unabhängig voneinander
- Ebenbleiben der Querschnitte, keine Verwölbung
- Balken vor Belastung spannungsfrei
- isotropes linear-elastisches Materialverhalten

Die Balkentheorie unterscheidet zwischen Euler-Bernoulli-Balken (a) für reine Biegung (vgl. Abbildung 43) und Timoshenko-Balken (c) für kombinierte Biege- und Schubbelastung. Die in Abbildung 44 gezeigte Vereinfachung durch Wölf freiheit (b) gilt auch für den Timoshenko-Balken, siehe Abbildung 45. Die beiden Spannungsarten sind dabei unabhängig voneinander in ihrer Wirkung.

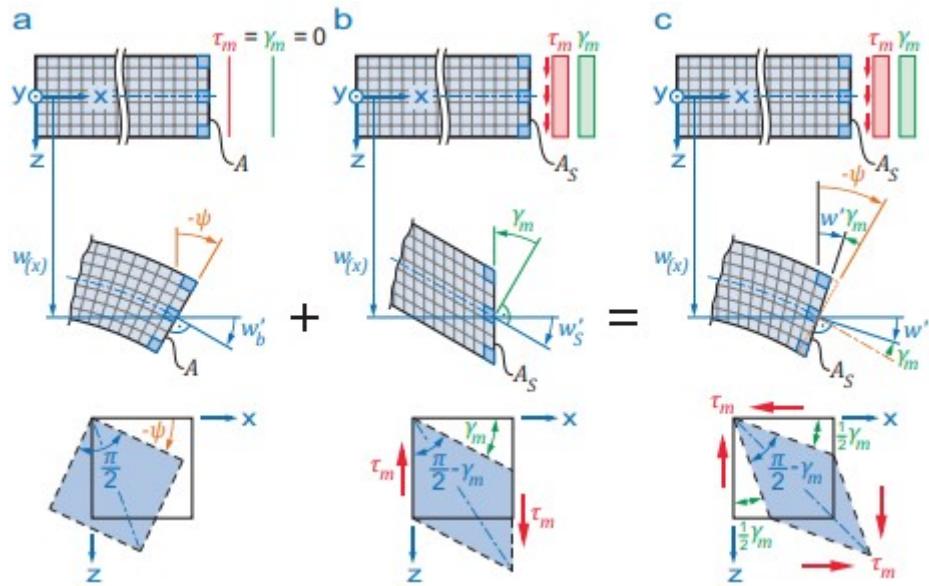


Abbildung 45: Kombination von (a) Euler-Bernoulli-Balken unter Biegung mit (b) Schubbelastung zu (c) Timoshenko-Balken (Spura 2019), S. 12

(Spura 2019), S. 1-5, S. 10-13

Der Euler-Bernoulli-Balken - kurz Bernoulli-Balken - ist ein Sonderfall des Timoshenko-Balkens. Er ist schubsteif, d.h. die Schubsteifigkeiten $\kappa_y GA = GA_{qy}$ und $\kappa_z GA = GA_{qz}$ betragen theoretisch unendlich. Schub bzw. Scherung spielen somit keine Rolle. Die Schwerpunktlinie x verformt sich bei Belastung. Die Querschnitte bleiben in ihrer jeweiligen Ebene senkrecht zur Schwerpunktlinie (1. Benoulli'sche Hypothese) und sind wölf frei (2. Benoulli'sche Hypothese). Der Euler-Bernoulli-Balken ist nur für schlanke Balken anwendbar: Balkenhöhe h zu Balkenlänge $l \ll 1 / 10$ (besser $\ll 1 / 15$).

Die Zusammenhänge für den Euler-Bernoulli-Balken zwischen Biegesteifigkeit EI_{yy} und verschiedenen Belastungen stehen in den nachfolgenden Gleichungen. Sie gelten auch für über die Balkenlänge veränderliche Biegesteifigkeiten $EI_{yy}(x)$.

$$[EI_{yy}(x) w_b''(x)]'' = q_y(x) \quad (\text{Streckenlast}) \quad \text{(Gleichung 8.11)}$$

$$[EI_{yy}(x) w_b''(x)]' = -Q_y(x) \quad (\text{Querkraft}) \quad \text{(Gleichung 8.12)}$$

$$EI_{yy}(x) w_b''(x) = -M_y(x) \quad (\text{Biegemoment}) \quad \text{(Gleichung 8.13)}$$

$$w_b'(x) = -\psi(x) \quad (\text{Neigung}) \quad \text{(Gleichung 8.14)}$$

$$w_b(x) \quad (\text{Biegelinie}) \quad \text{(Gleichung 8.15)}$$

(Nasdala 2015), S. 145-146 / (Spura 2019), S. 17-18

Der Timoshenko-Balken ist schubweich, d.h. Schubspannungen gehen in die Betrachtung ein. Bei Belastung bleibt der Querschnitt in sich eben. Er verdreht sich (schert bzw. neigt) um die Schwerpunktlinie x . Mittlere Schubspannung τ_m und mittlere Gleitung / Scherung y_m sind im Querschnitt konstant. Der Timoshenko-Balken gilt unter der Bedingung: Balkenhöhe h zu Balkenlänge $l \ll 1 / 8$ (besser $\ll 1 / 10$).

Zusätzlich zu den Gleichungen des Euler-Bernoulli-Balkens für Biegung gelten die folgenden Gleichungen für Schub. Beide Biegelinien überlagern sich additiv.

$$[GA_s(x)w'(x)]' = -q_y(x) \quad (\text{Streckenlast}) \quad \text{(Gleichung 8.16)}$$

$$GA_s(x)w_s'(x) = Q_y(x) \quad (\text{Querkraft}) \quad \text{(Gleichung 8.17)}$$

$$w_s(x) \quad (\text{zusätzliche Biegelinie durch Schub}) \quad \text{(Gleichung 8.18)}$$

$$w(x) = w_b(x) + w_s(x) \quad (\text{kombinierte Biegelinie}) \quad \text{(Gleichung 8.19)}$$

(Nasdala 2015), S. 149 / (Öchsner 2021), S. 79 / (Spura 2019), S. 7, S. 14-15

Die Ansätze zur Torsionsberechnung unterteilen sich in Saint Venantsche Torsion (Unterkapitel 2.2) und Wölbkrafttorsion (Unterkapitel 2.3). Saint Venantsche Torsion geht von unbehinderter Verwölbung (Verformung in x -Richtung aufgrund Torsionsmoment M_x bzw. T) des Querschnitts aus, wie es z.B. bei einer Gabellagerung der Fall ist, siehe Abbildung 46. Ansonsten ermittelt die Theorie der Wölbkrafttorsion zusätzliche Normalspannungen, die die Torsion bei Wölbbehinderung im Querschnitt verursacht. Die Wölbkrafttorsion ist insbesondere bei offenen Querschnitten, deren Profilmittellinien sich nicht in einem Punkt schneiden, relevant (wie z.B. dem I-Profil). Bei Vollquerschnitten und Kreisringquerschnitten spielt sie keine Rolle und bei dünnwandigen geschlossenen Querschnitten eine geringe. (Nasdala 2015), S. 153

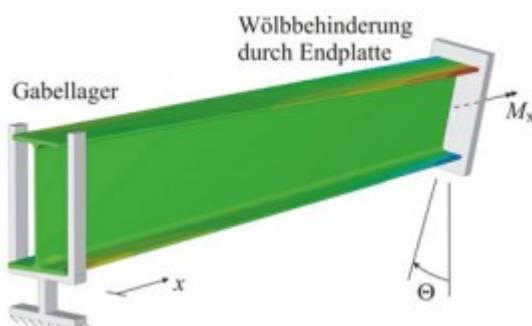


Abbildung 46: Unbehinderte Verwölbung im Gabellager und Verwölbung bei Wölbbehinderung am Beispiel eines I-Trägers (Nasdala 2015), S. 153

Bei linear-elastischem Verhalten des Systems gilt das Superpositionsprinzip für Beanspruchungen. So addieren sich z.B. die durch eine Normalkraft N verursachten Normalspannungen mit den durch Wölbkrafttorsion des Torsionsmomentes M_x verursachten Normalspannungen zur Gesamtnormalspannung des Querschnitts. Gleichermaßen gilt für die Verformungen (wie in Gleichung 8.19 für Biege- und Schubbelastung gezeigt). (Linke 2015), S. 191
Die Flächenschwerpunkte M (FSP) aller Querschnitte liegen auf einer Geraden, der Schwerpunktlinie x . Im einfachsten Fall haben diese Querschnitte auch identische Querschnittswerte. Das ist der einfachste zu berechnende Balken in Bezug auf die Geometrie.

Wenn alle Belastungen und Verformungen stetig im gesamten Balken sind, liegt ein Einfeldbalken vor. Ansonsten behandelt die Theorie den Balken als Mehrfeldbalken. Das ist z.B. in Abbildung 47 der Fall. Gelenke, Lager und Einzelkräfte bzw. Einzelmomente unterbrechen die Stetigkeit der aufgestellten Gleichungen. Dann sind für jedes Feld separate Gleichungen nötig. Zu den Randbedingungen kommen Übergangsbedingungen zur Formulierung der geometrischen Abhängigkeiten zwischen den Feldern.

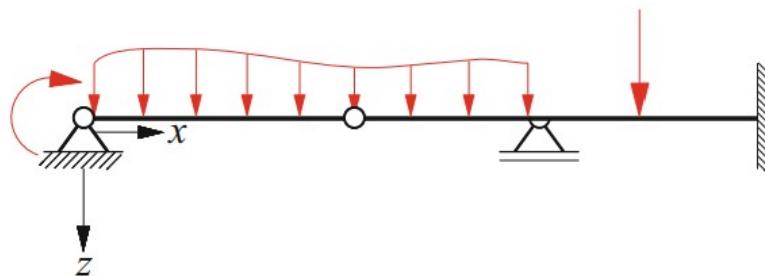


Abbildung 47: Beispiel für einen Mehrfeldbalken mit Unterbrechungen der Stetigkeit durch Gelenk, Zwischenlager und Einzelkraft (Mittelstedt 2021), S. 200

(Linke 2015), S. 71 / (Mittelstedt 2021), S. 200

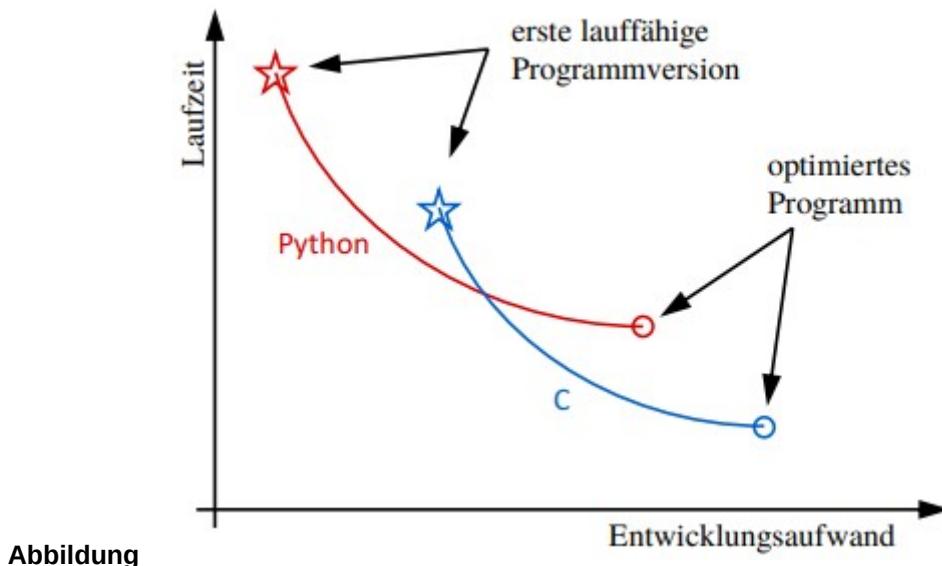
8.2 Relevante Aspekte von Python

8.2.1 Grundlagen der Programmierung

„Unter numerischer Programmierung versteht man das Gebiet der Informatik und Mathematik, in dem es um Approximationsalgorithmen geht, d.h. die numerische Approximation von mathematischen Problemen oder numerischer Analysis. In anderen Worten: Probleme mit stetigen Variablen.“ (Klein 2019), S. 5

Abbildung 48 zeigt einen Vor- und Nachteil von höheren Programmiersprachen wie z.B. Python, Matlab oder C# gegenüber niederen Programmiersprachen wie z.B. C oder Fortran. Die Einteilung hängt von der Komplexität und dem Abstrahierungsgrad der Programmiersprache ab. Diese sind z.B. bei Python sehr groß. Dadurch wird die Einarbeitung und Anwendung vereinfacht, da viele Prozesse vorgefertigt sind und automatisch ablaufen. Der Preis dafür liegt in der meist längeren Laufzeit des Programms. In dieser Kategorie sind niedere Programmiersprachen also von Vorteil. Deswegen kommt für umfangreiche numerische Berechnungen oft die Programmiersprache C zum Einsatz. Python bietet zur Optimierung der Laufzeit Einbindungen von in C geschriebenen Programmerweiterungen an, siehe z.B. Package NumPy in Unterkapitel 8.2.6. Eine weitere Einteilung von Programmiersprachen erfolgt in kompilierte und interpretierte Programmiersprachen. Kompilieren übersetzt direkt in Maschinencode (z.B. C/C++), wohingegen z.B. Python und Matlab einen zur Laufzeit arbeitenden Interpreter benötigen.

(Natt 2020), S. 3-4



Abbildung

48: Vergleich Laufzeit über Entwicklungsaufwand für höhere Programmiersprache (Python) und niedere Programmiersprache (C), (Natt 2020), S. 3

Grundsätzlich gibt es vier Paradigmen zum Formulieren von Problemen in einer Programmiersprache. Das sind die prozedurale, logische, funktionale und objektorientierte Programmierung.

Die prozedurale Programmierung wird im Matlab MuPad Notebook "QUEBER_Version2016" (Stanoev 2016) deutlich. Eine zeilenweise Abarbeitung des Programms ermöglicht einen übersichtlichen und nachvollziehbaren Berechnungsablauf.

Die logische Programmierung arbeitet hauptsächlich mit logischen Vergleichsoperatoren (z.B. und, oder, nicht-und) und benutzt vorrangig die booleschen Werte wahr und falsch (boolean values *True* and *False*).

Die funktionale Programmierung umfasst die Definition und (mehrfache) Verwendung von Funktionen. Eine Funktion liefert für übergebene Eingangsdaten mit Anweisung(en) erzeugte Ausgangsdaten. Damit sind wiederkehrende Schritte mit wenigen SLOC (Source lines of code) darstellbar. Zudem sind Funktionen einfach in andere Programmteile/Programme übertragbar.

Die objektorientierte Programmierung (OOP) schließlich führt die Verwendung von Klassen und Instanzen dieser Klassen (Objekten) ein. Eine Klasse umfasst Eigenschaften / Attribute (Daten) und Verhaltensweisen / Funktionen (Methoden) z.B. zum Verändern und Ausgeben dieser Daten oder zum Erstellen von neuen Instanzen dieser Klasse. Eine Klasse ist damit eine Art Template oder Blueprint für die Erstellung (Instanziierung) der Instanzen (Objekte) dieser Klasse. Objekte

sind somit Datenpakete (Verbunde, Verbundstrukturen). Nach / während der Instanziierung eines Objektes ist die Zuweisung von Daten möglich. Danach stehen die Änderung (wenn erlaubt) von Daten sowie die Ausführung der Methoden des Objektes zur Verfügung.

Somit unterscheidet sich die OOP von der funktionalen Programmierung. Eine Funktion liefert bei gleichen Eingangsdaten immer die gleichen Ergebnisse. Methoden eines Objektes beziehen sich auf veränderliche Daten des Objektes. So liefern sie in Abhängigkeit der bisherigen Programmschritte unterschiedliche Ergebnisse. Die OOP ist daher für die numerische Abbildung mechanischer Modelle gut geeignet.

(Amos 2020) / (Hammerly 2017) / (Natt 2020), S. 346-347 / (Woyand 2019), S. 141-147

Viele Programmiersprachen beinhalten vorgefertigte Klassen, Funktionen / Methoden und Operatoren. Zu den Operatoren gehören grundlegende Arithmetik-Operatoren wie z.B. für Addition, Subtraktion, Multiplikation und Division. Dazu kommen logische Operatoren und Vergleichsoperatoren wie z.B. Ist-gleich, Oder, Ist-nicht-gleich und Ist-größer-als. Mathematische Funktionen wie z.B. trigonometrische Funktionen sind ggf. auch vorhanden. Es besteht die Möglichkeit der Erstellung von Variablen und der Zuweisung von Werten zu diesen. (Woyand 2019), S. 4-9, S. 45-47

Die Daten benötigen verschiedene Datentypen. Diese sind grob in Ganzzahl (Integer), Gleitpunktzahl (Float), Zeichenkette (String) und boolesche Werte (Boolean) unterteilt. (Woyand 2019), S. 31-36

Eine Funktion (bzw. Methode als Teil einer Klasse / eines Objektes) wird einmal definiert und dann beliebig oft ausgeführt (initialisiert). Neben der Abbildung mathematischer Funktionen ist z.B. das Sortieren, Bewegen oder (graphische) Ausgeben von Daten möglich (side effects). Es besteht die Möglichkeit zur Übergabe von Parametern (Eingangswerte der Funktion). Es gibt Funktionen mit und ohne Rückgabewert. In jedem Fall wird ein Anweisungsblock ausgeführt. Thematisch verwandte Funktionen sind ggf. in Bibliotheken / Modulen / Packages zusammengefasst. (Natt 2020), S. 35 / (Woyand 2019), S. 54-59, S. 112

Ein Programm benötigt eventuell Eingabedaten vom / von der Anwender/in. Es sind oft auch Schnittstellen vorhanden zum Einlesen von Daten (z.B. aus einer Textdatei). Zusätzlich gibt es ggf. Möglichkeiten zur Datenausgabe (visuell, digital oder als Datei). (Woyand 2019), S. 63, S. 119-120, S. 123

Neben einer sequentiellen Abarbeitung (Haupteigenschaft der prozeduralen Programmierung) gibt es die Ausführung von Programmteilen unter einer Bedingung. Diese Programmverzweigungen funktionieren mit if- und else-Anweisungen. Wenn die geprüfte Bedingung wahr (*True*) ist, führt das Programm Anweisungen aus. Ist die Bedingung nicht erfüllt (*False*), geht das Programm zum nachfolgenden Schritt weiter. Eine Verknüpfung von mehreren Bedingungen mithilfe von logischen Operatoren ist möglich. (Woyand 2019), S. 66-69, S. 71

Programmschleifen (Wiederholungsanweisungen, Schleifen, Loops) steuern bzw. regeln die wiederholte Ausführung bestimmter Anweisungen. In vielen Programmiersprachen vorhandene Programmschleifen sind for-Schleife und while-Schleife. Eine for-Schleife führt für jede Variable in einer Liste bestimmte Anweisungen aus. Eine while-Schleife führt bestimmte Anweisungen nach Prüfung einer Bedingung solange aus, bis die Bedingung nicht erfüllt ist, siehe Abbildung 49. (Woyand 2019), S. 72-73, S. 77

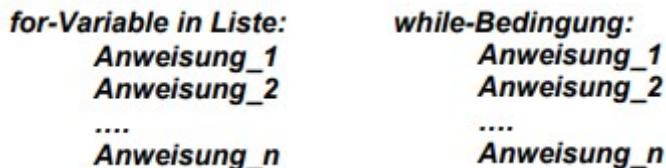


Abbildung 49: Schematischer Aufbau von for- und while-Schleife, (Woyand 2019), S. 73, S. 77

Für die numerische Programmierung ist eine „geordnete Zusammenfassung beliebiger Objekte“ wichtig. Allgemein steht dazu eine Liste zur Verfügung. Wenn die Elemente einer Liste ebenfalls aus Listen jeweils gleicher Länge bestehen entsteht eine Matrix. Die Art und Weise der Speicherplatznutzung und die Möglichkeiten einer Liste sind nicht in allen Programmiersprachen gleich. Python hat z.B. Listen, Sets, Tuples und Dictionaries für verschiedene Anwendungsbereiche. (Woyand 2019), S. 85

Bei einer Programmiersprache bietet sich eine Unterteilung der Merkmale in Syntax und Konzepte an. Die Konzepte sind ggf. entscheidend für die Verwendungsmöglichkeiten des Programms. Die Syntax dient hauptsächlich der Ergonomie für den/die Programmier/in. Das verdeutlicht ein Zitat aus einem Newsletter über Programmierung:

“Learning a programming language is like learning a natural language — it becomes easier the more languages you've already learned.

There are two things underlying every single language: concepts and syntax. Concepts are hard. Syntax is easy. And you can directly build upon the concepts from an old language when studying a new language.

Therefore, you should focus on the concepts!" (Finxter 2022)

8.2.2 Einleitung zu Python

Die Python Software Foundation repräsentiert auf einer Website (Python 2022) die Programmiersprache Python. Dort sind u.a. eine Dokumentation sowie Links zu Hilfeforen und Lernmöglichkeiten vorhanden. Zudem steht die neueste Version von Python zur Verfügung (Stand 09.08.2022: Python 3.10.6) sowie PyPI (Python Package Index) für Programmerweiterungen. (Python 2022)

Guido van Rossum gilt als der Erfinder von Python Anfang der 1990er Jahre. Ziel der Programmiersprache ist Einfachheit in Bezug auf Erlernbarkeit, Lesbarkeit, Erweiterbarkeit (mit Packages) und Einbettbarkeit in andere Sprachen. Neben der neuesten Version Python 3 gibt es Python 2. Beide Versionen sind miteinander nicht kompatibel. Deshalb liegt der Hauptfokus weiterer Betrachtungen auf Python 3. Für naturwissenschaftliche Anwendungen bzw. numerische Programmierung sind besonders die Packages NumPy, SciPy und Matplotlib interessant. Data Science und Machine Learning sind u.a. mit dem Package Pandas möglich. (Schäfer 2019), S. 1-2

Es sind wissenschaftliche Python-Distributionen verfügbar. Sie unterscheiden sich in den mitgelieferten Packages bzw. der Eignung für verschiedene Betriebssysteme. Unter Windows gibt es verschiedene Distributionen, siehe (Anaconda 2022), (Enthought 2022(a)) und (Raybaut 2021). (Natt 2020), S. 7-8

Python ist als universal einsetzbare Programmiersprache konzipiert, z.B. für Systemadministration, Websites und zur numerischen Programmierung. Matlab und R sind auf Machine Learning, Data Science und numerische Programmierung spezialisiert. Mit den bereits genannten Packages NumPy, SciPy, Matplotlib und Pandas entspricht der Funktionsumfang von Python in etwa dem von Matlab. (Klein 2019), S. 6-7

Abbildung 50 zeigt ein aktuelles Rating der Popularität von Programmiersprachen. Dabei fließt als Faktor die Anzahl der Suchanfragen in verschiedenen Suchmaschinen in die Berechnung ein. Zusätzlich berücksichtigt der TIOBE Index die Anzahl darin ausgebildeter Ingenieure, Kurse und Drittanbieter. Der Index dient somit als grober Anhaltspunkt für die derzeitige Häufigkeit in der Verwendung der Programmiersprachen. Python, C, Java und C++ sind die Sprachen mit einem Rating von über 10%. Auch R, Matlab und Fortran als wichtige Sprachen für numerische Programmierung sowie C# sind in den Top 20 enthalten. Es gibt auch die Platzierung von 1987 bis heute. Dabei belegt C immer den ersten oder zweiten Platz und C# ist über die Jahre immer in den Top 10. Python hingegen erfreut sich besonders in den letzten 4-5 Jahren seit etwa 2018 stark wachsender Beliebtheit. (TIOBE 2022)

Programming Language	Ratings	Change
 Python	15.42%	+3.56%
 C	14.59%	+2.03%
 Java	12.40%	+1.96%
 C++	10.17%	+2.81%
 C#	5.59%	+0.45%
 Visual Basic	4.99%	+0.33%
 JavaScript	2.33%	-0.61%
 Assembly language	2.17%	+0.14%
 SQL	1.70%	+0.23%
 PHP	1.39%	-0.80%
 Swift	1.27%	+0.30%
 Classic Visual Basic	1.27%	+0.04%
 Delphi/Object Pascal	1.22%	+0.60%
 Objective-C	1.22%	+0.61%
 Go	0.98%	+0.08%
 R	0.92%	-0.13%
 MATLAB	0.90%	-0.08%
 Ruby	0.82%	-0.18%
 Fortran	0.81%	-0.32%

Abbildung 50: Platz 1-19 des TIOBE Index für August 2022 mit Veränderung zum August 2021 (TIOBE 2022)

Eine Erweiterung der Python-Installation ist mit pip möglich. Dieses ist ggf. schon in der Python-Installation enthalten und installiert Packages aus PyPI und GitHub. (PyPA 2020)

Je nach Installationsart und Version von Python ist zur Ausführung von pip unterschiedliche Syntax in der command line nötig.

- *python3 -m pip install packagename*
- *pip install packagename*
- *pip3 install packagename*
- *python3 -m pip3 install packagename*

(Slatkin 2020), S. 443-444

Python bietet die direkte Ausführung von Anweisungen in der Python-Shell an. Das eignet sich zur schnellen Überprüfung von einfachen Programmierideen. Für umfangreiche Programme ist die Erstellung eines Python-Skripts sinnvoll. (Woyand 2019), S. 1, S. 12

Zum ergonomischen und einfachen Arbeiten bietet sich die Installation einer Integrated Development Environment (IDE) an. Mit der Python-Version kommt IDLE (Integrated Development and Learning Environment). JetBrains liefert die IDE / IDLE PyCharm als kommerzielle Version (für akademische Nutzung kostenlos) und Community-Version (eingeschränkter Umfang). Die IDE Spyder (Scientific Python Development Environment) ist kostenlos. Die Python-Distributionen (Anaconda 2022) und WinPython (Raybaut 2021) beinhalten Spyder. Schließlich stehen die ebenfalls kostenlosen Notebook Interfaces Jupyter Notebook und JupyterLab zur Verfügung. Sowohl Spyder als auch Jupyter Notebook / JupyterLab verfügen über eine Schnittstelle zu IPython. (JetBrains 2022a) / (Jupyter 2022) / (Natt 2020), S. 10 / (Schäfer 2019), S. 7-8 / (Spyder 2022)

IPython ist ein Interactive Shell und eine Erweiterung für IDEs. Es beinhaltet u.a. Namens vervollständigung und Unterstützung von Packages zur Erstellung von Graphical User Interfaces (GUI). IPython eignet sich gut als Ergänzung für NumPy, SciPy und Matplotlib. (IPython k.A.) / (Schäfer 2019), S. 7

Es bestehen verschiedene, teilweise kostenlose Möglichkeiten zum angeleiteten Erlernen von Python. Diese sind z.B. in den Websites (Codeacademy 2022), (JetBrains 2022), (OpenTechSchool 2014), (Refsnes Data 2022) und (Real Python 2022) vorhanden.

Python selbst ist auch für kommerzielle Nutzung kostenlos. Python-Programme laufen ohne Änderung auf den Betriebssystemen Windows, Mac Os und Linux. (Woyand 2019), S. 1

8.2.3 Operatoren, Funktionen und Anweisungen in Python

Zur Verknüpfung von Werten und/oder Objekten sind in Python - wie in vielen anderen Programmiersprachen - Operatoren enthalten. Arithmetische Operatoren sind – wenn nicht explizit genannt – für Gleitpunktzahlen und Ganzzahlen gültig bzw. einer Mischung beider Datentypen, siehe Abbildung 51. Die Rangfolge in der Abbildung gibt auch die Rangfolge bei der Ausführung an. Sie kann durch die Verwendung von Klammern verändert werden. (Woyand 2019), S 45

Operator	Bedeutung	Beispiele
$**$	Potenzierung	$-3^{**}4$ $2^{**}0.5$
$+ -$	Positives und negatives Vorzeichen	$+12.0$ -7
$* /$	Multiplikation und Division	$12.3*4$ $7/3$
$+ -$	Addition und Subtraktion	$12.3+4$ $7-3$
$//$	Ganzzahlige Division	$7//3$
$\%$	Modulo-Operator (Ermittlung des Restes bei ganzzahliger Division)	$7\%3$

Abbildung 51: Arithmetische Operatoren in Python 3 (Woyand 2019), S. 45

Python bietet einige mathematische Standardfunktionen. Diese befinden sich im Modul *math*. Mit der Import-Anweisung (z.B. `from math import sin, cos, tan` oder `import math`) sind diese im Programm verfügbar. Das Modul *math* enthält u.a. die in Abbildung 52 gezeigten Funktionen.

Funktionsname	Bedeutung	Kommentar
<code>sin(x)</code>	Sinus	x muss im Bogenmaß angegeben werden.
<code>cos(x)</code>	Cosinus	x muss im Bogenmaß angegeben werden.
<code>tan(x)</code>	Tangens	x muss im Bogenmaß angegeben werden.
<code>asin(x)</code>	Arcussinus	$-1 \leq x \leq 1$
<code>acos(x)</code>	Arcuscosinus	$-1 \leq x \leq 1$
<code>atan(x)</code>	Arcustangens	
<code>atan2(x,y)</code>	Arcustangens von x/y	
<code>sinh(x)</code>	Sinushyperbolicus	
<code>cosh(x)</code>	Cosinushyperbolicus	
<code>tanh(x)</code>	Tangenshyperbolicus	
<code>exp(x)</code>	Eulersche Funktion	
<code>log(x)</code>	Natürlicher Logarithmus	$x > 0$
<code>log10(x)</code>	Logarithmus zur Basis 10	
<code>pow(x,y)</code>	Potenzierung (wie $x^{**}y$)	
<code>radians(x)</code>	Bogenmaß eines Winkels	
<code>sqrt(x)</code>	Quadratwurzel	$x \geq 0$

Abbildung 52: Wichtige mathematische Funktionen aus Modul *math* (Woyand 2019), S. 47

Die übergebenen Werte der Parameter müssen rationale Zahlen sein, also Gleitpunktzahlen und/oder Ganzzahlen. (Woyand 2019), S. 46-47

Die Benennung von Variablen unterliegt einigen Regeln, Einschränkungen und Empfehlungen. Variablen können Klein- und Großbuchstaben enthalten, Ziffern und Unterstriche (underscore). Ein Variablenname darf nicht mit einer Ziffer beginnen. Umlaute sind zu vermeiden wegen der Beschränkung auf bestimmte Sprachräume. Python enthält reservierte Wörter (Schlüsselwörter), siehe Abbildung 53. Diese dürfen nicht als Variablenname verwendet werden. (Woyand 2019), S. 50

and	assert	break	class
continue	def	del	elif
else	except	exec	finally
for	from	global	if
import	in	is	lambda
not	or	pass	print
raise	return	try	while
with			

Abbildung 53: Reservierte Schlüsselwörter in Python (Woyand 2019), S. 50

Im Kontext mit Variablen ist die Zuweisung eine wichtige Anweisung. Sie weist einer Variablen einen Wert zu. Dabei ersetzt der zuletzt zugewiesene Wert den davor gültigen Wert. Die Zuweisung erfolgt mithilfe eines Zuweisungsoperators, siehe Abbildung 54. Der einfachste Zuweisungsoperator ist `=`. Der Variablenname steht auf der linken Seite des Zuweisungsoperators, der zugewiesene Wert auf der rechten. Bei einer Zuweisung wird erst der Wert der rechten Seite ermittelt und dieser dann dem Variablenamen auf der linken Seite zugewiesen. Bei Verwendung des Variablenamens in einer anderen Anweisung wird dort der aktuelle Wert der Variable eingesetzt. (Woyand 2019), S. 50-54

Zuweisungsoperator	Beispiel	Abkürzung für
<code>+=</code>	<code>i += 1</code>	<code>i = i + 1</code>
<code>-=</code>	<code>i -= 1</code>	<code>i = i - 1</code>
<code>*=</code>	<code>i *= 3</code>	<code>i = i * 3</code>
<code>/=</code>	<code>i /= 3</code>	<code>i = i / 3</code>
<code>%=</code>	<code>i %= 3</code>	<code>i = i % 3</code>
<code>**=</code>	<code>i **= 3</code>	<code>i = i ** 3</code>
<code>//=</code>	<code>i //= 3</code>	<code>i = i // 3</code>

Abbildung 54: Kombinierte Zuweisungsoperatoren in Python (Woyand 2019), S. 54

In Python gilt: „[...] Variablen sind Objekte eines bestimmten Typs.“ Der Typ der Variable entspricht dabei dem Datentyp des zuletzt zugewiesenen Wertes. (Schäfer 2019), S. 13 Ein wichtiges Konzept in Python ist die dynamische

Typisierung. Der Variablenotyp wird nicht bei Erstellung einer Variablen deklariert (wie es die statische Typisierung z.B. in C oder C+ macht). Python ist somit eine deklarationsfreie Programmiersprache. Die Typzuweisung findet während der Ausführung / Laufzeit des Programms (at runtime) im Interpreter statt. (Bhalla 2018) / (Schäfer 2019), S. 13 / (Woyand 2019), S. 52

In Python ist auch das Konzept der starken Typisierung (strong typing, strongly typed) wichtig. Der Interpreter führt nur für den jeweiligen Datentyp zulässige Operationen mit den Werten der Variablen aus (z.B. Arithmetik für Gleitpunktzahlen (floating number, float) und Ganzzahlen (integer, int)). Eine schwach typisierte Programmiersprache wie z.B. JavaScript führt auch unzulässige / ungewollte Operationen aus, wie z.B. die Addition von Zeichenketten (strings) und Ganzzahlen (integer). Ein Vorteil der Kombination von dynamischer und starker Typisierung ist das schnelle und einfache Schreiben des Quellcodees. Als Nachteil der dynamischen Typisierung treten gegebenenfalls in der Laufzeit des Programms Fehler wegen unzulässigen Operationen oder Funktionsaufrufen mit Werten auf. Die starke Typisierung kompensiert das ein Stück weit, da Widersprüche zu Fehlermeldungen führen und dem/der Programmier/in damit bewusst und sichtbar sind. (Python Wiki 2012)

„Python ist eine sogenannte strukturierte Programmiersprache. Das bedeutet, dass der Programmcode in Blöcke unterteilt werden kann. Um die einzelnen Blöcke zu kennzeichnen, werden in Python Einrückungen (indentations) verwendet. Das bedeutet zum einen, dass Python keine Schlüsselwörter oder Klammern zur Definition von Blöcken wie andere Sprachen benötigt, und zum anderen, dass der Code automatisch übersichtlicher und lesbarer wird.“ Die empfohlene Einrückungstiefe beträgt 4 Leerzeichen oder ein Tabulatorzeichen. Dabei sind automatisch von der IDE vorgenommene Einrückungen zu bevorzugen (wegen Risiko des Flüchtigkeitsfehlers). (Schäfer 2019), S. 9-10

Die Definition einer Funktion erfolgt mit dem Schlüsselwort *def*. Dann folgt der Funktionsname, direkt dahinter runde Klammern - ggf. mit Parametern - (*Parameter_1, Parameter_2,..., Parameter_n*) und schließlich ein Doppelpunkt : . In der nächsten Zeile beginnt der (jeweils gleichweit) eingerückte Anweisungsblock der Funktion. Es gibt Funktionen mit und ohne Rückgabewert. Die Rückgabe erfolgt am Ende des Anweisungsblocks mit dem Schlüsselwort *return* und dem Ausdruck oder Variablennamen des zurückgegebenen Werts, siehe Abbildung 55. (Woyand 2019), S. 56-59

```

def dist(xa,ya,xb,yb):          #2
    """dist - Berechnet den Abstand
    zweier Punkte. Übergabeparameter
    sind xa,ya (erster Punkt) und xb,yb
    (zweiter Punkt)"""
    abstand = sqrt((xa-xb)**2+(ya-yb)**2)  #3
    return abstand                  #4

```

Abbildung 55: Beispielfunktion in Python mit erklärendem Kommentar, Berechnungsanweisung und Rückgabe des berechneten Wertes (Woyand 2019), S. 55

Funktionen enthalten zusätzlich zu Pflichtparametern (Positions-Parametern) ggf. optionale Parameter (Schlüsselwort-Parameter). Schlüsselwort-Parameter haben voreingestellte Werte (Default-Werte). Diese sind in bei der Definition einer Funktion angegeben. Sie stehen in der Parameteraufzählung hinter den Pflichtparametern und müssen bei Verwendung der Funktion nicht angegeben werden. Ein Beispiel ist: *def Funktion(Pflichtparameter_1, Pflichtparameter_2, Schluesselwort-Parameter=Wert); Variable_1 = Funktion(Wert_1, Wert_2); Variable_2 = Funktion(Wert_1, Wert_2, Wert_3)* (Woyand 2019), S. 110-112

Funktionen haben den Vorteil der Wiederverwendbarkeit. Python bietet die Möglichkeit, Funktionen in einer Moduldatei mit der Endung .py abzuspeichern (im selben Verzeichnis wie die Datei des Programms). Mit der Anweisung *from Modul import Funktion* bzw. *import Modul* ist/sind die Funktion(en) des genannten Moduls im Programm verfügbar. Dabei wird im Verzeichnis eine kompilierte Moduldatei (Endung .pyc) erzeugt. Wenn Änderungen an der ursprünglichen Moduldatei stattfinden, ist diese kompilierte Version vor Neuausführung des Programms zu löschen. (Woyand 2019), S. 112-113

Bei Funktionen besteht die Möglichkeit von Rekursion. Dabei ruft sich eine Funktion selbst auf (rekursive Funktion). Eine rekursive Funktion benötigt daher eine Abbruchbedingung. Eine mögliche Verwendung ist die Berechnung der Fakultät eines Werts, siehe Abbildung 56. (Woyand 2019), S. 114-115

```

def faku2(n):
    print("n=",n)
    if n==1:
        p = 1
    else:
        p = n*faku2(n-1)
    return p

print(faku2(1))
print(faku2(5))

```

Abbildung 56: Berechnung der Fakultät eines Parameters (*n*) mithilfe von Rekursion in Python (Woyand 2019), S. 115

„Funktionen verfügen über einen eigenen eigenen Namensraum [Namespace, Anm. d. Verf.].“ Das bedeutet: Eine in einem Namensraum initialisierte Variable gilt ausschließlich in diesem. Eine im globalen Namensraum des Hauptprogramms initialisierte Variable (globale Variable) ist für alle Funktionen und Methoden verfügbar. Eine in einem dem Hauptprogramm untergeordneten Namensraum initialisierte Variable gilt nur in diesem Namensraum. (Woyand 2019), S. 116-117

Python verfügt über die beiden Standardfunktionen *input()* und *print()*. Diese sind vorrangig in der Python-Shell nützlich für Eingabe und Ausgabe. *input()* liest einen beliebigen String ein und gibt diesen als String zurück. *print()* nimmt beliebig viele Überabeparameter beliebigen Typs und gibt sie als einen String in der Konsole aus. Bei der Ausgabe von Werten erweisen sich Formatelemente als hilfreich, siehe Abbildung 57. Diese Formatelemente sind Platzhalter für Variablen. Sie werden mithilfe von - einem %-Zeichen nachgestellten - Ziffern in Feldgröße und Anzahl der Nachkommastellen genauer definiert oder in Exponentialschreibweise dargestellt, siehe Abbildung 58. (Woyand 2019), S. 63-65

Formatelement	Ausgabe von
%i (%d)	Ganzzahlen (int)
%f	Gleitpunktzahlen (float)
%e (%E)	Gleitpunktzahlen (Ausgabe im Exponentialformat)
%s	Zeichenketten (Strings)

Abbildung 57: Auswahl wichtiger Formatelemente in Python (Woyand 2019), S. 64

```
>>> x = 1200.8788
>>> print("Das Ergebnis ist: %8.3f" % x) #1
Das Ergebnis ist: 1200.879
>>> print("Das Ergebnis ist: %10.3E" % x) #2
Das Ergebnis ist: 1.201E+03
>>> print("Das Ergebnis ist: %10.3e" % x) #3
Das Ergebnis ist: 1.201e+03

>>> y = 125
>>> print("Die Zahl ist: %4i" % y) #4
Die Zahl ist: 125
>>> print("Die Zahl ist: %7d" % y) #5
Die Zahl ist: 125
>>> einheit = "km/h"
>>> print("Die Geschwindigkeit ist %i %s" % (y,einheit)) #6
Die Geschwindigkeit ist 125 km/h
>>>
```

Abbildung 58: Beispiele für die Verwendung von Formatelementen in der *print()*-Funktion von Python (Woyand 2019), S. 65

Im ingenieurtechnischen Kontext ist das Lesen und Schreiben von (Text-)Dateien – z.B. mit Daten in Tabellen- oder Listenform – wichtig. Beides benutzt für Textdateien (.txt) die Standardfunktion `open()`, siehe Abbildung 59 zum Lesen (mit Parameter “`r`”). Der Parameter “`w`” gilt zum Schreiben und “`a`” zum Lesen und Schreiben. Zum Entfernen von nicht benötigten Zeichen - z.B. Leerzeichen, Tabulatoren (`\t`) oder Zeilenumbrüche (`\n`) - gibt es die Methode `.strip()`. `open()` legt eine Liste unter dem zugewiesenen Variablenamen an. Jede Textzeile ist ein String-Element in der Liste. Die Liste verfügt über Methoden zur Aufbereitung der Daten, z.B. `readlines()` oder `.close()` zum Schließen der Textdatei. (Woyand 2019), S. 119-123

```
datei = open("staedte.txt", "r")
inhalt = datei.readlines()
for i in inhalt:
    print(i)
datei.close()
```

Abbildung 59: Beispiel zum Lesen einer Textdatei mit `open()` und `.readlines()` in Python (Woyand 2019), S. 120

Das Schreiben in eine Textdatei benötigt in `open()` den Parameter “`w`” oder “`a`”. Die Methode `.write()` der erstellten Liste ermöglicht das Schreiben in die Textdatei. Hierbei sind eventuell Formatelemente nützlich, siehe Abbildung 60. (Woyand 2019), S. 123

```
dat = open("einmaleins.txt", "w")
dat.write("Kleines Einmaleins\n\n")
for i in range(1,11,1):
    for j in range(1,11,1):
        dat.write("%10d% (j*i))
        dat.write("\n")
dat.close()
```

Abbildung 60: Beispiel zum Schreiben einer Texdatei mit `open()` und `.write()` (Woyand 2019), S. 123

Zur Erstellung von Programmverzweigungen bietet Python `if`-, `else`- und `elif`-Anweisungen. Der Anweisungsblock ist eingerückt, siehe Abbildung 61. Schlüsselwort (hier `if` oder `elif`) und Bedingung im Code sind mit einem Leerzeichen getrennt, siehe Abbildung 62. Die sog. `else`-Klausel der `if`-Anweisung (auch vollständige Alternative) wird bei Nicht-Erfüllung der Bedingung(en) ausgeführt. Sie ist genauso weit wie die zugehörige `if`-Anweisung eingerückt. Schließlich gibt es die `elif`-Klausel mit eigener Bedingung als Variante der `else`-Klausel. (Woyand 2019), S. 66-69

```

if Bedingung:
    Anweisung_1
    Anweisung_2
    ...
    Anweisung_n

if Bedingung:
    Anweisungsblock_1
else:
    Anweisungsblock_2

if Bedingung_1:
    Anweisungsblock_1
elif Bedingung_2:
    Anweisungsblock_2
elif Bedingung_3:
    ....
else:
    Anweisungsblock_n

```

Abbildung 61: Schemata von Programmverzweigungen in Python mit *if*-, *else*- und *elif*-Anweisungen (Woyand 2019), S. 67-69

```

x = float(input("Bitte x eingeben: "))
if x <= 0.0:                                #1
    y = 0.0
elif 0.0 < x < 1.0:                         #2
    y = x
else:
    y = 1.0
print(y)

```

Abbildung 62: Beispiel für *if*-Anweisung mit *elif*- und *else*-Klausel in Python (Woyand 2019), S. 68

Bedingungen verwenden Vergleichsoperatoren und haben entweder den booleschen Wert *True* oder *False* (Groß- und Kleinschreibung der Buchstaben ist wichtig), siehe Abbildung 63. (Woyand 2019), S. 71

Vergleichsoperator	Beispiel	Ergebnis	Bedeutung
<	3 < 9	True	kleiner als
>	7 > 2	True	größer als
<=	2 <= 2	True	kleiner gleich
>=	2 >= 7	False	größer gleich
==	9 == 1	False	ist gleich
!=	9 != 1	True	ungleich

Abbildung 63: Vergleichsoperatoren in Python mit Beispiel und Ergebnis (Woyand 2019), S. 71

Der Unterschied zwischen den Operatoren *is* und *==* wird in Abbildung 64 ersichtlich. Die Standardfunktion *id(object)* gibt für ein Objekt als Parameter eine Speicherbereich-spezifische Ganzzahl zurück (die ID). (Schäfer 2019), S. 14-15

Operator	Funktion
<i>is</i>	Prüft auf Gleichheit der Objekte, Objekte referenzieren den selben Speicherbereich
<i>==</i>	Prüft auf Gleichheit der Werte der Objekte, die Werte am jeweiligen Ort des Speicherbereichs sind gleich

Abbildung 64: Gegenüberstellung der Operatoren *is* und *==* in Python (Schäfer 2019), S. 15

Mithilfe der logischen Operatoren *and*, *or* und *not* sind komplexere Bedingungen möglich, siehe Abbildung 65. Logische Operatoren stehen in der Rangfolge unter den Vergleichsoperatoren und werden nach diesen ausgeführt. Wenn nur der Wert einer Variable geprüft wird, liefert *True* und jeder Wert außer 0 *True*. Kein Wert, 0, *None* und *False* liefern *False* als Ergebnis. (pythontutorial.net 2022) / (Woyand 2019), S. 71

Logischer Operator	Beispiel	Ergebnis	Bedeutung
and	1<2 and 3>0	True	und
or	1<2 or 5>7	True	oder
not	not 1<2	False	nicht

Abbildung 65: Logische Operatoren in Python mit Beispiel und Ergebnis (Woyand 2019), S. 71

Programmschleifen – auch Wiederholungsanweisungen, Schleifen oder Loops – beschreiben die wiederholte Ausführung eines Anweisungsblocks. Bei der *for*-Schleife läuft das Programm für jede Variable (Schleifenvariable) in einer Liste / einem Tuple einmal durch den eingerückten Anweisungsblock, siehe Abbildung 66. Es sind Beispiele für Listen mit eckigen Klammern [] und Tuple ohne Klammern vorhanden. Die Standardfunktion *range()* hat als mögliche Parameter Startwert, Stoppwert und Schrittweite. Wenn die Schrittweite negativ ist, bildet die Funktion den entsprechenden Dekrement. Das letzte Element der so erzeugten Liste ist dabei größer/gleich bzw. kleiner/gleich dem Stoppwert. (Woyand 2019), S. 72-75

for Variable in Liste: Anweisung_1 Anweisung_2 Anweisung_n	for i in 1,2,3,4: print(i,i*i)
for i in [1,2,3,4,5,6,7,8,9,10]: print(i,i*i)	for i in range(1,11): print(i,i*i)

>>> range(11,1,-2) [11, 9, 7, 5, 3]	>>> range(11,1,-3) [11, 8, 5, 2]
--	-------------------------------------

Abbildung 66: *for*-Schleife in Python mit Beispielen, u.a. Verwendung der *range()*-Funktion (Woyand 2019), S. 72-74, bearb.

Eine weitere Programmschleife in Python ist die *while*-Schleife (Bedingungsschleife), siehe Abbildung 67. (Woyand 2019), S. 77

while Bedingung: Anweisung_1 Anweisung_2 Anweisung_n	i = 1 while i<=4: print(i, i*i) i = i + 1
--	---

Abbildung 67: *while*-Schleife mit Beispiel in Python (Woyand 2019), S. 77

8.2.4 Datentypen in Python

In Python gibt es einfache Datentypen. Dazu gehören Ganzzahl (*Integer*, *class int*), Gleitpunktzahl (*Floating number*, *class float*), komplexe Zahl (*Complex number*, *class complex*), Zeichenkette (*String*, *class str*) und boolescher Wert (*Wahrheitswert*, *boolean value*, *class bool*). Datentypen sind in Python Objekttypen. Einen Überblick über wichtige Eigenschaften der numerischen Datentypen (alle einfachen Datentypen außer Zeichenkette) liefert Abbildung 68. (Woyand 2019), S. 31

Objekt	Genauigkeit und Wertebereich
<code>int</code>	Wertebereich nicht limitiert
<code>float</code>	64 bit double precision, nach IEEE 754 Standard
<code>bool</code>	<code>True</code> und <code>False</code>
<code>complex</code>	Komplexe Zahlen mit Real- und Imaginärteil

Abbildung 68: Eigenschaften numerischer Datentypen in Python (Schäfer 2019), S. 17

Ganzzahlen (*class int*) bestehen aus Ziffern und ggf. einem Vorzeichen (+/-). Die Länge bzw. Größe von Ganzzahlen ist in Python nicht begrenzt. Die Standardfunktion *int()* wandelt einen übergebenen Wert – wenn möglich – in eine Ganzzahl um. Das funktioniert für Strings (ausschließlich mit Ziffern) und Float-Werte. Nachkommastellen werden bei Float-Werten abgeschnitten. (Woyand 2019), S. 31

Gleitpunktzahlen (*class float*) dienen allgemein zur Darstellung reeller Zahlen (in Python mit 64-bit double precision). D.h. rationale Zahlen sind (wenn nicht zu lang) genau. Irrationale Zahlen (z.B. π , Eulersche Zahl e) sind in Python näherungsweise durch rationale Zahlen dargestellt. Das Trennzeichen in Python ist ein Punkt. Es ist eine Exponentialschreibweise – z.B. mit der Basis 10, e bzw. E – vorhanden. Der Exponent ist immer ganzzahlig. Es sind keine Leerzeichen zwischen den Zeichen in der Exponentialschreibweise erlaubt, siehe Abbildung 69. Eine Umwandlung in eine Gleitpunktzahl mit der Standardfunktion *float()* ist z.B. für Ganzzahlen und Strings (nur aus Ziffern) möglich. (Woyand 2019), S. 32-34

```
# Mehrere Wege zur Darstellung der Zahl 3000,0
x1 = 3000.0
x2 = 3e+3
x3 = 3.0e3
x4 = 3E3
x5 = 3.e+3
```

Abbildung 69: Verschiedene (Exponential)Schreibweisen einer Gleitpunktzahl in Python (Woyand 2019), S. 33

Die Darstellung komplexer Zahlen (*class complex*) aus Real- und Imaginärteil ist mit *j* / *J* als imaginäre Einheit oder mithilfe der Standardfunktion *complex()* möglich, siehe Abbildung 70. (Woyand 2019), S. 34

```
# Komplexe Zahlen
z1 = 3.0 - 5.0j
z2 = complex(3.0,-5.0)
z3 = 3 - 5j
```

Abbildung 70: Syntax zur Beschreibung komplexer Zahlen in Python (Woyand 2019), S. 34

Boolesche Werte (*class bool*) sind wahr (*True*) oder falsch (*False*). Die Großschreibung des ersten Buchstabens ist dabei in Python wichtig. In logischen Operationen bzw. Vergleichsoperationen gelten folgende Werte - neben *False* selbst - als *False* (sind *falsy*): *None*, leere Strings / Listen / Tuple / Dictionaries und die numerischen Werte *0.0* / *0* (*float* / *int*). (Schäfer 2019), S. 16

Zeichenketten (Strings, *class str*) stehen zwischen hochgestellten Anführungszeichen (double quotation marks, double quotes) wie in "*string*" oder zwischen Apostrophen (single quotation marks, single quotes) wie in '*string*'. Enthält ein String selbst solche Zeichen, ist zur Einschließung das jeweils andere Zeichen möglich, z.B. "*Das ist ein 'string'*" oder '*Das ist ein "string"*'. String ist ein sequentieller Datentyp, d.h. von links nach rechts hat jedes Zeichen einen Index (beginnend bei 0). Daher haben z.B. die Operatoren + und * eine einfache bzw. mehrfache Aneinanderreihung von mehreren / einem String(s) zur Folge (sog. concenation). Es gibt einige Standardfunktionen (auch) für Strings, z.B. *len(string)* zur Ermittlung der Anzahl der Zeichen oder *str()* zur Umwandlung anderer Datentypen in Strings. Für Zeichenketten gibt es viele nützliche Methoden. Einige wichtige sind nachfolgend genannt und erklärt.

string.count(substring) – gibt Anzahl von *substring* in *string* zurück

string.startswith(substring) / *string.endswith(substring)* – gibt *True* zurück, wenn *string* mit *substring* anfängt bzw. endet, sonst *False*

string.find(substring) – gibt Index des ersten Zeichens des ersten gefundenen *substring* in *string* zurück

string.replace(substring_1, substring_2, number) – erstellt eine Kopie von *string* und ersetzt *number*-mal *substring_1* mit *substring_2* (von links beginnend)

`string.isdigit()`, `string.isalpha()`, `string.isalnum()` - geben `True` zurück, wenn `string` nur aus Ziffern / Buchstaben / Ziffern und Buchstaben besteht, sonst `False`

`string.split(character)` – gibt durch `character` getrennte Teile von `string` als mehrere neue Strings zurück, siehe Abbildung 71.

```
x = "M 100 100 L 150 100"
a,b,c,d,e,f = x.split(" ")
b = int(b); c = int(c)
e = int(e); f = int(f)
print("Anfangspunkt: ", b,c)
print("Endpunkt: ", e,f)
```

Abbildung 71: Beispiel für Verwendung von String-Methode `string.split()` in Python (Woyand 2019), S. 40

(Woyand 2019), S. 35-40

Python beinhaltet komplexe Datentypen. Das sind Listen, Sets, Tuples und Dictionaries. Wichtige Operationen für sequentielle Datentypen (Listen, Tuples und Strings) sind in Abbildung 72 zusammengefasst. (Schäfer 2019), S. 17

Syntax	Operation
<code>a[i]</code>	Gibt <code>i</code> tes Element von Sequenz <code>a</code> zurück
<code>a[i:j]</code>	Gibt den Bereich <code>i</code> tes bis <code>(j-1)</code> tes Element von Sequenz <code>a</code> zurück
<code>a[i:j:k]</code>	Gibt den Bereich <code>i</code> tes bis <code>(j-1)</code> tes Element in Schritten von <code>k</code> von Sequenz <code>a</code> zurück
<code>a+b</code>	Verkettung von <code>a</code> und <code>b</code> , Rückgabe ist neue Datensequenz
<code>a+=b</code>	Fügt <code>b</code> an <code>a</code> an
<code>n*a</code>	Erzeugt neue Datensequenz mit <code>n</code> -fachem Inhalt
<code>e in a</code>	Prüft ob <code>e</code> in <code>a</code> enthalten ist, Rückgabewert <code>True</code> oder <code>False</code>
<code>e not in a</code>	Prüft ob <code>e</code> nicht in <code>a</code> enthalten ist, Rückgabewert <code>True</code> oder <code>False</code>

Abbildung 72: Operationen an sequentiellen Datentypen (Listen, Tuples, Strings) in Python (Schäfer 2019), S. 17

„Allgemein gesprochen handelt es sich bei Listen um eine geordnete Zusammenfassung von – im Prinzip – beliebigen Objekten. Diese werden mit eckigen Klammern eingeschlossen und durch Kommata getrennt. Eine Liste wird in Python als Objekt betrachtet und kann einer Variablen zugewiesen werden.“ Die Objekte in einer Liste können auch unterschiedlichen Typs sein. Wenn eine Liste eine/mehrere weitere Liste(n) enthält, ist deren Bezeichnung Subliste(n). Die Elemente einer Liste werden mit ihrem Index (beginnend bei 0) angesprochen, z.B. `list[i]`, siehe Abbildung 73. Listen sind veränderliche (mutable) Objekte. (Schäfer 2019), S. 17 / (Woyand 2019), S. 85-86

```
>>> a = [1,3,5,7,9]
>>> b = [2,"Hallo!", 12.5]
>>> c = [[2,4], ['a','b']]
>>> print(a[0],a[3])
1 7
>>> print(b[1])
Hallo!
>>> print(c[1], c[1][1])
['a', 'b'] b
```

```
moegliche_farben = ["rot", "grün", "blau"]
farbe = input("Bitte Farbe eingeben: ")
if farbe not in moegliche_farben:
    print("Diese Farbe ist nicht enthalten!")

moegliche_farben = ["rot", "grün", "blau"]
for i in moegliche_farben:
    j = moegliche_farben.index(i)
    print(j,i)
```

Abbildung 73: Beispiele für Listen und Verwendung in *if*-Anweisung und *for*-Schleife mit *not in*- und *in*-Operator in Python (Woyand 2019), S. 85-86

„Nachfolgend werden die wichtigsten Funktionen und Methoden zum Arbeiten mit Listen kurz vorgestellt.

len(list)

Gibt die Länge der Liste *list* bzw. die Anzahl der in der Liste enthaltenen Elemente zurück.

list.index(element)

Gibt den Index des Elements *element* innerhalb der Liste *list* zurück.

list.append(element)

Die Methode *append()* fügt an eine bestehende Liste *list* das Element *element* an (Anhängen eines Elements).

list.remove(element)

Die Methode *remove()* löscht das Element *element* aus der Liste *list*.

list.sort()

Mit dieser Methode wird die Liste *list* sortiert.

list.reverse()

Mit der Methode *reverse()* wird die Reihenfolge der Listenelemente umgekehrt.

Das letzte Element der Liste *list* wird zum ersten Element. Das erste Element wird zum letzten etc.

list.count(element)

Die Methode *count()* ermittelt, wie oft ein Element *element* in der Liste enthalten ist.

list.insert(index,element)

Mit der Methode *insert()* kann ein Element *element* in eine Liste *list* eingefügt werden. Das Einfügen geschieht an der Position *index*. Die restlichen Elemente der Liste werden angehängt: ihre Indizes verändern sich durch das Einfügen.“ (Woyand 2019), S. 87

Die Funktionen *max(list)* und *min(list)* geben die Extremwerte der Elemente einer Liste *list* zurück. Die Zuweisung eines neuen Wertes zu einem Element mit Index *i* überschreibt den bestehenden Wert, z.B. *list[i] = 3*. Weiterhin ermöglichen die

Operatoren + und * die Aneinanderreihung von Listen, ähnlich wie bei Strings. (Woyand 2019), S. 89

„Ein Tuple ist eine geordnete Zusammenfassung beliebiger Objekte (ganz ähnlich zur Liste). Im Gegensatz zu Listen sind Tuples jedoch unveränderlich [immutable, Anm. d. Verf.]. Es gibt also keine Methoden zum Löschen von Elementen oder zum Überschreiben von Elementen in Tuples. Die Elemente eines Tuples werden in runde Klammern eingeschlossen.“ Die Elemente werden durch Kommata voneinander getrennt. Das Setzen von Klammern ist optional, siehe Abbildung 74. Tuples haben Vorteile gegenüber Listen bei der Speicherung unveränderlicher (konstanter) Daten. Ein versehentliches Verändern ist nicht möglich. Das gilt allerdings ausschließlich für die Elemente selbst. Wenn diese Variablennamen / Referenzen auf veränderliche (mutable) Objekte (z.B. Listen, Strings) enthalten, ist weiterhin eine Änderung der Werte dieser Objekte möglich. (Schäfer 2019), S. 17-18 / (Woyand 2019), S. 95-96

```
>>> werte = (10,20,30)
>>> type(werte)
<class 'tuple'>
>>> zahlen = 10,20,30
>>> type(zahlen)
<class 'tuple'>
>>> werte==zahlen
True
```

Abbildung 74: Beispiele zur Syntax von Tuples in Python, mit und ohne runde Klammern (Woyand 2019), S. 95

„Ein Set (Menge) ist eine ungeordnete Zusammenfassung von Objekten, wobei jedes Objekt nur einmal darin vorkommen kann. Im Gegensatz zu Listen gibt es also keinen Index, der die Reihenfolge der Objekte im Set festlegt. Wir können zwar über eine Menge iterieren, z.B. mittels einer Schleife, dabei ist aber nicht festgelegt, in welcher Reihenfolge diese Objekte erscheinen. Auch können in Listen Objekte mehrfach vorkommen, bei Mengen (Sets) ist das nicht möglich. Sets können verändert werden. [...] Listen können keine Elemente von Mengen sein. Auf zwei Mengen können drei verschiedene Operationen (&, -, |) angewendet werden. In der folgenden Grafik [siehe Abbildung 75, Anm. d. Verf.] sind die beiden Mengen durch Kreise repräsentiert. Das Ergebnis der nachfolgend definierten Mengenverknüpfungen ist schraffiert dargestellt.“ Die Mengenoperationen erzeugen eine neue Menge, siehe Abbildung 76. Eine Schnittmenge (Durchschnitt) mit dem Operator & erstellt ein neues Set mit den in beiden Sets enthaltenen Elementen. Eine Differenzmenge mit dem Operator - erstellt ein neues Set mit den Elementen des ersten Sets ohne die Elemente der Schnittmenge mit dem zweiten Set. Eine Vereinigungsmenge mit dem Operator |

erstellt ein neues Set mit allen Elementen von zwei Sets. In beiden Sets vorkommende Elemente sind im neuen Set entsprechend einmal vorhanden. Die Syntax für die Erstellung eines Set ist z.B. `set([element_1, element_2, ... , element_n])`. Die Elemente haben keinen Index und die Reihenfolge der Elemente ist willkürlich. (Woyand 2019), S. 96-97

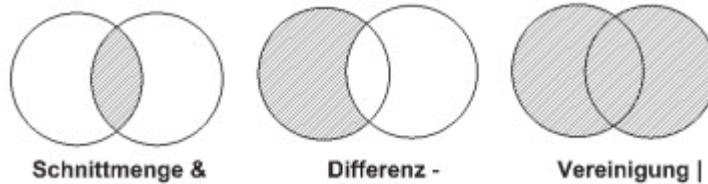


Abbildung 75: Mengenoperationen an zwei Sets (Mengen) in Python (Woyand 2019), S. 97

```
>>> Menge1 = set([12,34,60,12,72,34,90])
>>> print(Menge1)
set([72, 60, 34, 12, 90])
>>> Menge2 = set([17,90,34,34,112,9,12])
>>> print(Menge2)
set([34, 9, 12, 112, 17, 90])
>>> Menge1&Menge2
set([34, 12, 90])
>>> Menge1-Menge2
set([72, 60])
>>> Menge2-Menge1
set([112, 9, 17])
>>> Menge1|Menge2
set([34, 72, 9, 12, 112, 17, 90, 60])
>>> for i in Menge1:
    print(i)

72 60 34 12 90
```

Abbildung 76: Beispiel von Sets (Mengen) und Mengenoperationen in Python (Woyand 2019), S. 97

„Neben Listen sind Dictionaries die bedeutendsten Datenstrukturen in der Programmiersprache Python. Es handelt sich dabei um eine ungeordnete Sammlung von Schüssel-Wert-Paaren. Dictionaries werden auch als assoziative Arrays oder als assoziative Felder bezeichnet.“ Dabei enthält ein Dictionaries Datenpaare. Ein Datenpaar besteht aus einem Schlüsselwert (key value) und einem Datenwert (data value). Das Beispiel in Abbildung 77 sieht als Quellcode so aus: `kfz = {"DO":"Dortmund","E":"Essen","W":"Wuppertal"}`. „Die Datenwerte können im Prinzip beliebige Objekte sein. Ein Datenwert kann also auch aus einer Liste bestehen. Die Schlüsselwerte dürfen jedoch nicht aus Listen oder anderen Dictionaries bestehen.“ (Woyand 2019), S. 98-99

Für Dictionaries existieren in Python einige Methoden, z.B. `dict.keys()` und `dict.values()` mit Schlüssel- bzw. Datenwerten des Dictionaries als Rückgabewert. (Woyand 2019), S. 100

Schlüssel (Key Value)	Wert (Data Value)
DO	Dortmund
E	Essen
W	Wuppertal

Abbildung 77: Beispiel für Schlüsselwert – Datenwert – Paare eines Dictionaries in Python (Woyand 2019), S. 98

Für geordnete Datentypen (sequentielle Objekttypen, sequentielle Datenstrukturen) besteht die Möglichkeit des Slicings von Teilbereichen. In Python geht das mit Strings, Listen und Tuples, da jedes Element einen Index hat, siehe Abbildung 78. Slicing ermöglicht das Kopieren eines beliebig großen Teilbereiches einer solchen sequentiellen Datenstruktur, siehe Abbildung 79. (Woyand 2019), S. 105-016

Satz = "Stadt Wuppertal" # String
S t a d t W u p p e r t a l
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
Liste = [1, 4, 9, 16, 25, 36, 49, 64] # Liste
1 4 9 16 25 36 49 64
0 1 2 3 4 5 6 7
Kfz = ("W", "E", "DO", "BO" , "K", "D") # Tuple
"W" "E" "DO" "BO" "K" "D"
0 1 2 3 4 5

Abbildung 78: Beispiele für geordnete Datentypen (sequentielle Objekttypen) in Python – Strings, Listen und Tuples (Woyand 2019), S. 105

```
>>> Satz = "Stadt Wuppertal"
>>> Name = Satz[6:14]
>>> print(Name)
Wupperta
>>> Name = Satz[6:15]
>>> print(Name)
Wuppertal
>>> print(Satz[6:])
Wuppertal
>>> print(Satz[6:-1])
Wupperta
>>>
>>> Liste = [ 1, 4, 9, 16, 25, 36, 49, 64 ]
>>> print(Liste[0:3])
[1, 4, 9]
```

Abbildung 79: Beispiele von Slicing in Python (Woyand 2019), S. 106

Eine wichtige Vorgehensweise zur Verkürzung von Quellcode ist List Comprehension. „Die einfache Form der List Comprehension ist in allgemeiner Form: „Ausgabeliste = [Ausdruck for Element in Eingabeliste]““

Zwei Beispiele dazu verdeutlichen die Anwendung, siehe Abbildung 80. Der Ausdruck verwendet das jeweilige Element der Eingabeliste zur Ermittlung des Wertes eines Elements - mit gleichem Index - in der Ausgabeliste. (Woyand 2019), S. 107

```
>>> Ein = [ 1,5,10,16 ]
>>> Aus = [i+3 for i in Ein]
>>> Aus
[4, 8, 13, 19]
>>> Aus2 = [k**2 for k in Ein]
>>> Aus2
[1, 25, 100, 256]
>>> Rein = ["Kugel", "Zylinder", "Quader"]
>>> Raus = [len(x) for x in Rein]
>>> Raus
[5, 8, 6]
```

Abbildung 80: Beispiele für List Comprehension in Python (Woyand 2019), S. 108

„Die erweiterte, allgemeine Form der List Comprehensions wird durch eine zusätzliche if-Anweisung ergänzt:

Ausgabeliste = [Ausdruck for Element in Eingabeliste if Bedingung]“

Wenn die *if*-Bedingung erfüllt ist, wird der Wert des Ausdrucks an die Ausgabeliste angehängt. Wenn die *if*-Bedingung nicht erfüllt ist, geht die List Comprehension zur Prüfung der *if*-Bedingung des nächsten Element der Eingabeliste weiter. Beispiele verdeutlichen diese Vorgehensweise, siehe (Woyand 2019), S. 108

```
>>> Ein = range(10)
>>> Ein
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> Aus = [ x**3 for x in Ein if x%2==0 ]
>>> Aus
[0, 8, 64, 216, 512]
>>> Liste1 = [ 1,5,6,12,17]
>>> Liste2 = [ 2,5,6,19,12,23]
>>> Liste3 = [u for u in Liste1 if u in Liste2]
>>> Liste3
[5, 6, 12]
```

Abbildung 81: Beispiele für List Comprehension mit if-Anweisung in Python (Woyand 2019), S. 108

Für Listen gibt es die *iter()*-Funktion. Die Methode *next()* gibt bei Aufruf das nächste Element der iterierten Liste zurück, siehe Abbildung 82. Wenn die Liste vollständig durchlaufen ist, gibt die nächste Methode *next()* einen Fehler aus. (Woyand 2019), S. 109

```
>>> Staedte = ["Essen", "Dortmund", "Wuppertal"]
>>> k = iter(Staedte)
>>> next(k)
'Essen'
>>> next(k)
'Dortmund'
>>> next(k)
'Wuppertal'
>>> next(k)
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    next(k)
StopIteration
```

Abbildung 82: Beispiel für Verwendung von `iter`-Funktion und Methode `next()` in Python (Woyand 2019), S. 109

Für die Kombination von zwei oder mehr Listen in eine neue Liste mit Tuples gibt es die Funktion `zip()`, siehe Abbildung 83. (Woyand 2019), S. 110

```
>>> Staedte = ["Essen", "Dortmund", "Wuppertal"]
>>> Kfz = ["E", "DO", "W"]
>>> Ergebnis = zip(Staedte,Kfz)
>>> list(Ergebnis)
[('Essen', 'E'), ('Dortmund', 'DO'), ('Wuppertal', 'W')]
```

Abbildung 83: Beispiel für die Zusammenführung von zwei Listen mit `zip()`-Funktion in Python (Woyand 2019), S. 110

8.2.5 Objektorientierte Programmierung in Python

Eine Einleitung zu Objektorientierter Programmierung (OOP) ist in Unterkapitel 8.2.1 vorhanden. Die Syntax zur Definition einer Klasse in Python (Parameter *object* ist optional) ist:

```
class ClassName(object):
```

Anweisungsblock

Klassenattribute werden bei der Definition einer Klasse mitdefiniert, indem ihnen Werte zugewiesen werden. Damit übernehmen alle neuen Instanzen einer Klasse die Werte der Klassenattribute ihrer Klasse. Im Beispiel in Abbildung 84 sind das *Name*, *x* und *y*. Sie können weggelassen werden, um den Quellcode zu verkürzen. Die Definition von (dann normalen) Attributen erfolgt in diesem Fall bei Aufruf einer Methode. Im Beispiel ist das *object.vorgabe(self,n,x,y)* einer Instanz. Attribute und Methoden von Klassen und Objekten sind im selben Namensraum verfügbar, z.B. *Punkt.Name* oder *Punkt_1.vorgabe(Punkt_1,10,0)*. Ein direktes Aufrufen von Attributen auf diese Weise ist nicht empfohlen. Stattdessen besteht die Konvention, Methoden zum Ein- und Ausgeben (allg. *get()* und *set()*) von Attributen zu verwenden. Dieses Prinzip der OOP heißt Datenkapselung (encapsulation) oder Geheimnisprinzip (information hiding) und findet in Python Anwendung. Methoden schaffen definierte Schnittstellen zur Interaktion eines Objektes, z.B. mit anderen Objekten oder Funktionen. (Woyand 2019), S. 143-

147

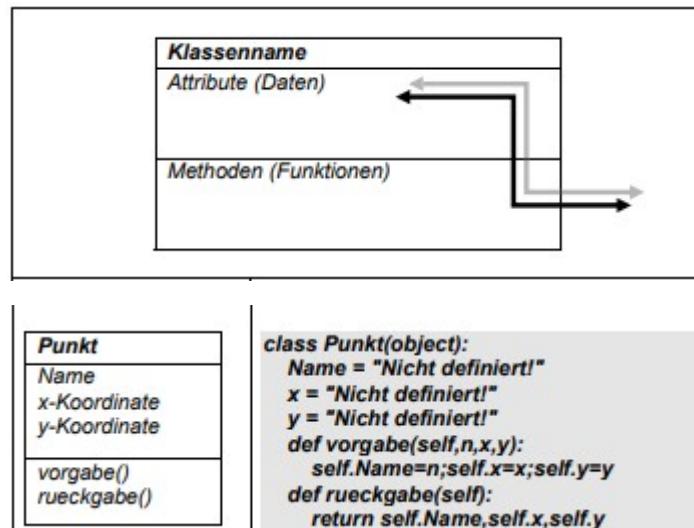


Abbildung 84: Schema zum Aufbau einer Klasse mit Beispiel in Python (Woyand 2019), S. 145, bearb.

Ein Variablenname in einer Zuweisung ist nicht identisch mit Objektname. Daher enthalten Klassen oft ein Attribut *name*. (Woyand 2019), S. 165

Der Aufruf (Initialisierung) einer Methode erfolgt folgendermaßen:
`object.method(parameter_1,...,parameter_n)`

Es gibt in Python einige Methoden mit vorgegebenen Namen. Diese sind durch jeweils zwei dem Methodennamen vor- und nachgestellte Unterstriche erkennbar.

Mit einem Konstruktor `__init__()` ist die Ausführung einer Methode direkt beim Erzeugen eines Objektes möglich, z.B. für die Zuweisung von Attributen mithilfe von übergebenen Parametern, siehe Abbildung 85.

```
class vektor(object):
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def ausgabe(self):
        print("X-Komponente: ",self.x)
        print("Y-Komponente: ",self.y)
```

Abbildung 85: Beispiel für die Verwendung eines Konstruktors zur Wertzuweisung der Attribute x und y eines Objektes der Klasse `vektor()` (Woyand 2019), S. 151

Ein Destruktor `__del__()` löscht das Objekt. Er führt optional einen Anweisungsblock aus, z.B. zum Ausgeben von Attributwerten oder einer Meldung.

(Woyand 2019), S. 149-151

Operatoren haben in Abhängigkeit der Datentypen / Objekttypen der Operanden unterschiedliche Bedeutungen. Für standardmäßige Klassen in Python wie String, Liste und Integer ist die Bedeutung der Operatoren definiert. Wenn ein Programmierer / eine Programmiererin Operatoren für Objekte einer eigenen Klasse anwenden möchte, ist eine Überladung dieser Operatoren (operator overloading) notwendig. (Woyand 2019), S. 152

Abbildung 86 zeigt die Überladung des Operators `+` an einem Beispiel. Dazu steht in der Definition der Klasse die Methode `__add__()`. Zwei mit dem Operator `+` verknüpfte Objekte der Klasse bilden (im Beispiel) ein neues Objekt der selben Klasse. Ein Programmierer / eine Programmiererin definiert die Vorgehensweise im Anweisungsblock der Methode, z.B. die Addition von Attributwerten. Die Methodennamen zur Überladung verschiedener Operatoren stehen in Abbildung 87. Es ist die Rückgabe von Werten beliebigen Objekttyps möglich. (Woyand 2019), S. 154-155

```

class vektor(object):
    def __init__(self,x=0,y=0):
        self.x = x
        self.y = y
    def __add__(self,other):
        x = self.x + other.x
        y = self.y + other.y
        return vektor(x,y)
    def ausgabe(self):
        print("(,"+str(self.x)+","+str(self.y)+")")

v1 = vektor(1,5)
v1.ausgabe()
v2 = vektor(3,-2)
v2.ausgabe()
v3 = v1 + v2
v3.ausgabe()

```

Abbildung 86: Beispiel der Überladung des Operators + für Objekte der Klasse `vektor()` in Python (Woyand 2019), S. 154

Operator	Methode
+	<code>__add__(self,other)</code>
-	<code>__sub__(self,other)</code>
*	<code>__mul__(self,other)</code>
/	<code>__div__(self,other)</code>
**	<code>__pow__(self,other)</code>
==	<code>__eq__(self, other)</code>
>=	<code>__ge__(self, other)</code>
>	<code>__gt__(self, other)</code>
<=	<code>__le__(self, other)</code>
<	<code>__lt__(self, other)</code>

Abbildung 87: Methoden zum Überladen von Operatoren in Python (Woyand 2019), S. 156

Vererbung (inheritance) von Klassen ist in Python möglich. „Bei der Vererbung übernimmt eine Klasse die Attribute und die Methoden einer anderen Klasse.“ Eine Klasse (Subklasse) erbt optional die Attribute und Methoden beliebig vieler Klassen (Oberklassen). Die Syntax dazu ist:

`class Subklasse(Oberklasse_1,...,Oberklasse_n):`

Attribute

Methoden

(Woyand 2019), S. 156, S. 159

8.2.6 Numerik mit Package NumPy

„NumPy ist ein Erweiterungsmodul für numerische Berechnungen. Mit ihm können Vektoroperationen mit hoher Geschwindigkeit ausgeführt werden. Weiterhin stellt NumPy grundlegende mathematische Werkzeuge für wissenschaftliche Anwendungen zur Verfügung.“ (Woyand 2019), S. 173

Die hohe Geschwindigkeit der Berechnung beruht auf vielen in C geschriebenen Bestandteilen von NumPy. Eine Anleitung zur Installation liefert die NumPy-Homepage (NumPy 2022) bei Klick auf GetStarted. Ansonsten ist die Verwendung einer Python-Distribution möglich, siehe Unterkapitel 8.2.2. Mit *import numpy* oder *import numpy as np* sind die Klassen und Funktionen aus dem Package verfügbar. (Klein 2019), S. 43-44

NumPy führt Array (Vektor, array) als komplexen Datentyp ein. Arrays ähneln Listen aus Standard-Python. Auf ein Array angewandte mathematische Funktionen aus NumPy (z.B. *sin()*, *cos()*, *exp()*, *log()*) geben ein neues Array zurück. Dabei berechnet die Funktion für jedes Element des Arrays im Ausdruck den Wert des zugehörigen Elements im neuen Array, siehe Abbildung 88. Das Beispiel zeigt die Überschneidung bei Funktionsnamen aus Numpy und dem *math*-Modul. Python wählt z.B. für *sin(array)* mit einem Array als Parameter automatisch die Funktion aus NumPy aus.

```
from math import *
zahlen = [ 1, 2, 3, 4, 5, 6 ]
ergebnis = []
for x in zahlen:
    y = sin(x)
    ergebnis.append(y)
```

```
from numpy import *
zahlen = array( [1, 2, 3, 4, 5, 6] )
ergebnis = sin(zahlen)
```

Abbildung 88: Gegenüberstellung von *sin()*-Funktion aus *math*-Modul und *numpy* für Liste und *numpy.array* in Python (Woyand 2019), S. 174

Die Syntax von Arrays ist *array([element_1, element_2,...,element_n], dtype=datatype)*. Der Schlüsselwert-Parameter *dtype* kann einen Datentyp für die Elemente vorgeben. Alle Elemente eines Arrays sind Objekte gleichen Typs. Datentyp und Größe eines Arrays sind automatisch festgelegt, wenn nicht durch den Programmierer/ die Programmiererin gegeben. Der Datentyp wird mit *array.dtype* ersichtlich (z.B. *int32*, *float64*).

Die Funktion *array()* erstellt auch Matrizen, siehe Abbildung 89. Die Methode *array.shape()* eines Arrays gibt ein Tupel mit dessen Anzahl von Zeilen und Spalten (rows and columns) zurück.

```
>>> M1 = array( [ [1, 2, 3], [4, 5, 6] ] ) # Array erklären
>>> M1
array([[1, 2, 3],
       [4, 5, 6]])
>>> M1.shape # Anzahl der Zeilen und Spalten
(2, 3) # abfragen
```

Abbildung 89: Beispiel für Erzeugung von Matrix mit `array()` aus NumPy in Python
(Woyand 2019), S. 175

Nachfolgend sind einige Funktionen aus NumPy erklärt. Es gibt Funktionen zur Erzeugung von verschiedenen Standardmatrizen, z.B. `zeros()` und `ones()`. Als Parameter nehmen die beiden Funktionen ein Tupel mit der Anzahl der Reihen und Spalten. Die Matrix wird dann je nach Funktion mit entsprechenden Werten ausgefüllt (hier 0 oder 1).

„Die Funktion `arange(s,e,i)` erlaubt es, eine Sequenz von Zahlen (auch Gleitpunktzahlen) mit aufsteigenden oder absteigenden Werten zu erstellen. Sie kann mit der Funktion `range()` verglichen werden, die allerdings nur ganzzahlige Argumente akzeptiert. Die Zahl `s` bedeutet den Startwert, `e` den Endwert und `i` das Inkrement. Aufgrund der Gleitpunktarithmetik kann man nicht genau voraussagen, wie viele Einträge ein Array haben wird, das durch diese Funktion entstanden ist. Eine Alternative ist die Funktion `linspace(s,e,n)`, bei der neben dem Start- und Endwert auch die Anzahl der gewünschten Array-Elemente `n` angegeben wird. Das hierfür notwendige Inkrement wird dann automatisch bestimmt.“ (Woyand 2019), S. 176 Ein Beispiel gibt Abbildung 90.

```
>>> M2 = zeros((3,3))
>>> M2
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> M3 = ones((3,4))
>>> M3
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
>>> x = zeros(5)
>>> x
array([ 0.,  0.,  0.,  0.,  0.])
```

$$\begin{aligned} >>> \text{time} = \text{arange}(0,5,0.5) \\ >>> \text{time} \\ \text{array}([0., 0.5, 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5]) \\ >>> t = \text{linspace}(0,5,11) \\ >>> t \\ \text{array}([0., 0.5, 1., 1.5, 2., 2.5, 3., 3.5, 4., 4.5, 5.]) \end{aligned}$$

Abbildung 90: Beispiele für Erzeugung von Standardmatrizen mit NumPy-Funktionen `zeros()` und `ones()` sowie Vektoren mit `arange()` und `linspace()` aus NumPy in Python (Woyand 2019), S. 176

Mathematische Operatoren (+, -, *, /, **) werden bei Operationen mit Arrays elementweise angewendet. Es gibt die NumPy-Funktion `dot(x, y)` für das Matrizenprodukt zweier Arrays. In Vergleichausdrücken mit Array(s) gibt der Ausdruck ein Array mit den für alle Elemente ermittelten booleschen Ausdrücken

zurück. Die Berechnung des Skalarproduktes (inneres Produkt) von zwei Arrays mit der NumPy-Funktion `inner(x, y)` ist ebenfalls möglich, siehe Abbildung 91.

Addition von zwei Arrays

```
>>> x = arange(1,6,1)
>>> y = arange(2,12,2)
>>> x
array([1, 2, 3, 4, 5])
>>> y
array([ 2,  4,  6,  8, 10])
>>> x+y
array([ 3,  6,  9, 12, 15], dtype=int32)
```

Multiplikation von zwei Arrays

```
>>> x = array([1,3,0,-1])
>>> y = array([2,0,5,3])
>>> x*y
array([ 2,  0,  0, -3], dtype=int32)
```

Addition/Subtraktion von Ganzzahl

```
>>> x = array([1,0,-2])
>>> x+5
array([6, 5, 3], dtype=int32)
>>> x-1
array([-1, -1, -3], dtype=int32)
```

Matrizenprodukt mit `dot()`

```
>>> X = array( [[1,2], [0,1]] )
>>> Y = array( [[1,0], [1,2]] )
>>> X
array([[1, 2],
       [0, 1]])
>>> Y
array([[1, 0],
       [1, 2]])
>>> Z= dot(X,Y)
>>> Z
array([[3, 4],
       [1, 2]])
```

Skalarprodukt mit `inner()`

```
>>> x = array([1,2,3])
>>> y = array([1,4,-5])
>>> z = inner(x,y)
>>> z
-6
```

Vergleichoperator

```
>>> x = arange(1,11,1) #1
>>> x
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>> y = x%3==0 #2
>>> y
array([False, False, True, False, False, True, False, True, False])
```

Abbildung 91: Beispiele für Array-Operationen aus NumPy in Python (Woyand 2019), S. 176-178

Aus dem mathematischen Gebiet der linearen Algebra sind einige Funktionen (zusätzlich zu `inner()` und `dot()`) in NumPy enthalten. Sie sind im Namespace `numpy.linalg` (bzw. `np.linalg`) zusammengefasst. Als wichtige Funktionen gibt es `linalg.solve(A, b)` und `linalg.inv(A)`, siehe Abbildung 92. Mit beiden Ansätzen ist die Lösung eines linearen Gleichungssystems möglich. `A` ist dabei die Koeffizientenmatrix und `b` der Spaltenvektor (Vektor der rechten Seite). `linalg.solve(A, b)` löst ein lineares Gleichungssystem direkt. `linalg.inv(A)` gibt die Inverse von `A` zurück, welche dann mit `b` in `dot(Ainv, b)` multipliziert wird.

```
A = array([[1,2], [3,4]])
b = array([[5],[11]])
y = linalg.solve(A,b)
```

```
A = array([[1,2], [3,4]])
b = array([[5], [11]])
Ainv = linalg.inv(A)
print(Ainv)
print()
y = dot(Ainv,b)
```

Abbildung 92: Beispiele für Lösung eines linearen Gleichungssystems und Inverse einer Matrix mit `numpy.linalg` in Python (Woyand 2019), S. 178-179

(Woyand 2019), S. 173-179

Wichtige Werkzeuge zum Arbeiten mit Arrays sind Indizierung und Teilbereichsoperationen (slicing). Beide funktionieren ähnlich wie bei Listen, vgl. Unterkapitel 8.2.4. Elemente von Arrays haben analog zu anderen sequentiellen Datentypen in Python (String, Liste und Tuple) einen Index. Ein Element wird mit $A[i][j]..[z]$ in einem ein- oder mehrdimensionalen Array A angesprochen. In NumPy ist eine weitere (speichertechnisch bessere) Möglichkeit mit $A[i, j, \dots, z]$ gegeben. i ist der Index der Zeile, j der Index der Spalte usw.

Slicing erzeugt in NumPy eine Sicht (view) auf das Original-Array (bei Listen entsteht eine Kopie). Daher sind Änderungen in der Sicht – im Gegensatz zu einer Kopie – auch im Original-Array wirksam. Für ein eindimensionales Array ist die Syntax `[start:stop:step]`, für ein zweidimensionales `[start_row:stop_row:step_row, start_column:stop_column:step_column]` usw.

Es gibt einige kurze Schreibweisen, diese sind größtenteils identisch mit dem allgemeinen Slicing in Python. Sie sind am eindimensionalen Array am einfachsten zu erklären und entsprechend auch bei mehrdimensionalen Arrays anwendbar. In Abbildung 93 ist der Defaultwert von `step=1` gültig und damit nur `start` und/oder `stop` und/oder `:` nötig zur Angabe des Bereichs. Wenn nur `:` verwendet wird, sind alle Elemente gemeint. Steht links von `:` kein Wert, so ist `start=0`. Steht rechts von `:` kein Wert, ist `stop=Index des letzten Elements`. Negative Indize zählen bei -1 startend vom letzten Element rückwärts.

```
S = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
print(S[2:5])
print(S[:4])
print(S[6:])
print(S[:])
```

Ausgabe:

```
[2 3 4]
[0 1 2 3]
[6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
```

Abbildung 93: Beispiel für Schreibweisen beim Slicing eindimensionaler Arrays aus NumPy in Python (Klein 2019), S. 57-58

(Klein 2019), S. 56-60

Die Funktion `identity()` bildet eine Identitätsmatrix (Einheitsmatrix), `eye()` eine beliebige Diagonalmatrix mit Einsen ab. Als Schlüsselwert-Parameter haben beide den Datentyp der Elemente (Standard ist `dtype=float`). (Klein 2019), S. 67-68

8.2.7 Datenvisualisierung mit Package Matplotlib

„Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.“ Auf der Homepage von Matplotlib sind Anleitungen, Dokumentation und Beispiele vorhanden. Bei Nutzung einer Python-Distribution wie z.B. Anaconda oder WinPython ist Matplotlib enthalten. (Matplotlib 2022)

Unter Getting Started sind einige einführende Informationen verfügbar. Dazu gehört eine Anleitung zur Installation mit pip oder conda. Die Erstellung eines 2D-Linien-Plots ist an einem Beispiel verdeutlicht, siehe Abbildung 94. Dazu wird im Programm zunächst aus dem Package *matplotlib* das Modul *pyplot* unter dem Kurznamen *plt* importiert. Weiterhin wird das Package *numpy* als *np* importiert zur Erzeugung der Arrays *x* und *y*. (Matplotlib 2022a)

```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)

fig, ax = plt.subplots()
ax.plot(x, y)
plt.show()
```

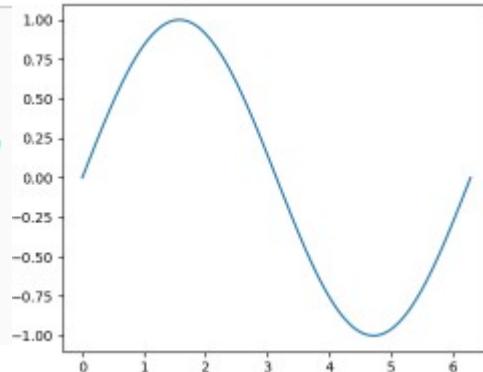


Abbildung 94: Beispiel für Plot einer Sinus-Funktion mit Matplotlib in Python
(Matplotlib 2022a)

Neben dem bereits vorgestellten 2D-Linien-Plot mit *ax.plot(x, y)* gibt es z.B. Scatter-Plot mit der Methode *ax.scatter(x, y)* und Bereichsdarstellung mit *ax.fill_between(x, y1, y2)*. (Matplotlib 2022b) Eine gute Übersicht der Grundlagen von Matplotlib findet sich in bereitgestellten Cheatsheets. (Matplotlib 2022c)

Abbildung 95 zeigt wesentliche Bestandteile eines Plots in Matplotlib. Grundlegende Begriffe sind Figure, Axes, Axis und Artist.

Artist – Alle Objekte einer Matplotlib-Grafik sind Artists. Dazu gehören Figure, Axes, Axis, Text- / Line2D- / collection- / Patch-Objekte. Beim Rendern von Figure erscheinen alle Artists im canvas (Zeichenfläche). Viele Artists sind Teil eines Axes-Objektes.

Figure (*plt.figure*) – Dieser Artist / dieses Objekt beinhaltet Axes (als Instanz einer Subklasse), titles, figure legends, colorbars und nested subfigures.

Axes (*ax*) – Ein Artist des Objekttyps Axes umfasst Axis-Objekte (*ax.xaxis*, *ax.yaxis* und ggf. *ax.zaxis*) und z.B. *ax.legend*, *ax.grid*, *ax.spines* und *ax.plot*.

Dazu kommen Methoden wie z.B. `ax.set_xlabel()`, `ax.set_ylabel()`, `ax.set_title()` usw. für die Veränderung der Eigenschaften dieser Objekte. `ax.plot()` nimmt als Parameter `numpy.array (np.array)`, `numpy.ma.masked_array (np.ma.masked_array)` und mit `numpy.asarray (np.asarray)` konvertierte Objekte.

Axis (`ax.xaxis`, `ax.yaxis` usw.) - Attribute der Axis-Objekte sind u.a ticks und ticklabel (Achseneneinteilung und beschriftung). Ein Locator-Objekt verwaltet die ticks, ein Formatter-Object beinhaltet Informationen und Anweisungen für die Werteintragung an den ticks (ticklabel strings).

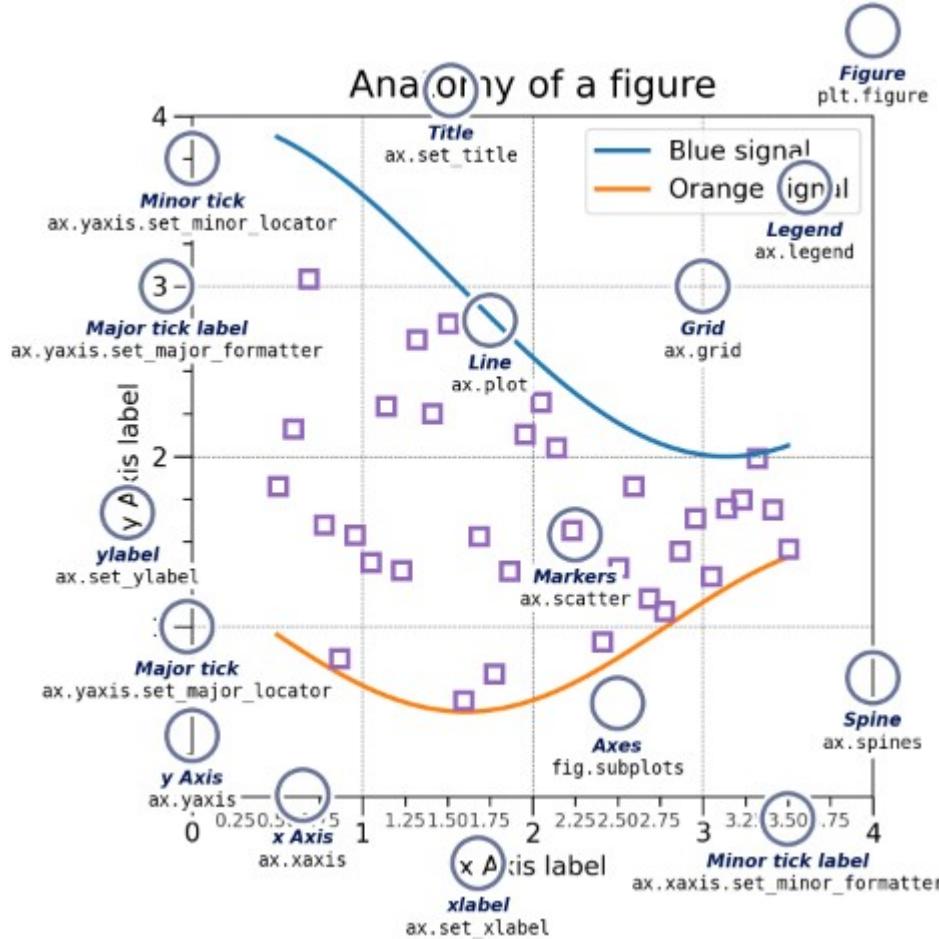


Abbildung 95: Grundlegende Objekte und Methoden einer Grafik mit Matplotlib in Python (Matplotlib 2022d)

Matplotlib ermöglicht drei Programmier-Stile (Coding styles) – objektorientiert, pyplot-Stil und Implementierung in eine GUI (z.B. mit Tk, siehe (Matplotlib 2022e) und Unterkapitel 8.2.8).

Für den allgemein empfohlenen objektorientierten Programmier-Stil nutzt der Anwender / die Anwenderin bevorzugt Methoden von Axes-Objekten, siehe Abbildung 96.

```
x = np.linspace(0, 2, 100) # Sample data.

# Note that even in the OO-style, we use `plt.pyplot.figure` to create the Figure.
fig, ax = plt.subplots(figsize=(5, 2.7), layout='constrained')
ax.plot(x, x, label='linear') # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic') # Plot more data on the axes...
ax.plot(x, x**3, label='cubic') # ... and some more.
ax.set_xlabel('x label') # Add an x-label to the axes.
ax.set_ylabel('y label') # Add a y-label to the axes.
ax.set_title("Simple Plot") # Add a title to the axes.
ax.legend(); # Add a Legend.
```

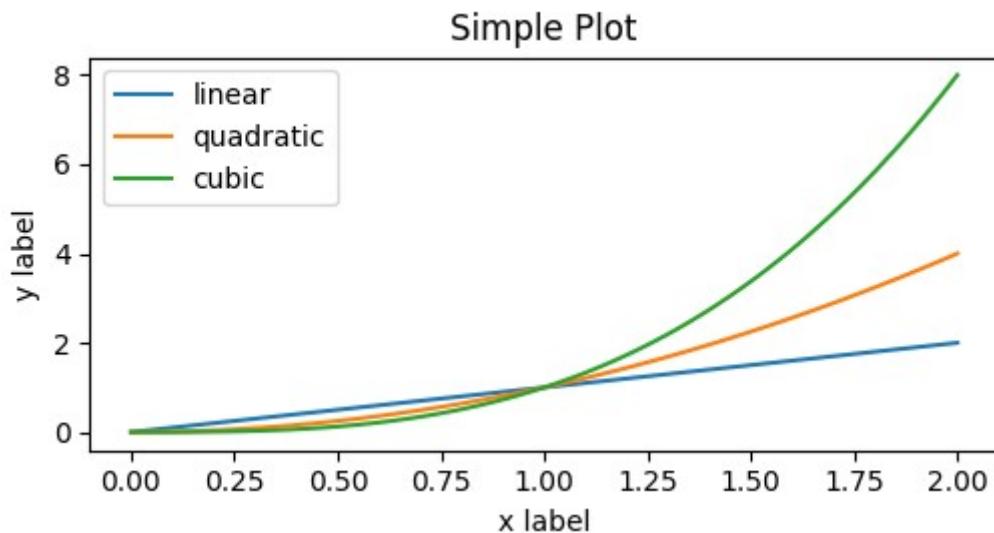


Abbildung 96: Beispiel für objektorientiert programmierten 2D-Linien-Plot mit Matplotlib in Python (Matplotlib 2022d)

(Matplotlib 2022d)

8.2.8 Graphical User Interface mit Package PySimpleGUI

PySimpleGUI ist ein Wrapper für die Ports Tkinter, PyQt, wxPython und Remi. Die Installation eines Ports erfolgt separat, z.B. Tkinter mit *pip install tk* in der command line (für die Masterarbeit wird Tkinter als Port verwendet). Die Installation von PySimpleGUI erfolgt mit *pip install pysimplegui* in der command line. Die Anweisung *import PySimpleGUI as psg* macht das Package im Programm-Modul verfügbar.

In Python bezeichnet der Begriff *widget* ein Element der GUI (button, label, window etc.). PySimpleGUI verwendet nested lists (Liste(n) in einer Liste) zur Positionierung von widgets. Zeilen und Spalten der nested list entsprechen Zeilen und Spalten im Fenster (window). PySimpleGUI ordnet diese automatisch an.

Eine nested list aller widgets (*layout*) wird einer Instanz der Klasse *psg.Window()* als Parameter übergeben. Diese nimmt als weiteren Positionsparameter einen Namen für das Fenster. Weiterhin ist die Übergabe vieler Schlüsselwert-Parameter zur Konfiguration möglich. PyCharm Community 2022 z.B. zeigt alle Schlüsselwert-Parameter mit einer kurzen Erklärung an.

Eine Interaktion mit einem Anwender / einer Anwenderin ist durch einen event loop in Form einer *while*-Schleife möglich. Die Methode *window_menu.read()* gibt bei Eingabe von Werten oder z.B. Drücken eines buttons diese Werte zurück und weist sie den vorgefertigten Variablen *event* und/oder *values* zu. Das geschieht so oft bzw. so lange, bis die Abbruchbedingung (*if*-Anweisung) erfüllt ist. Eine Erweiterung der Abbruchbedingung mit weiteren Bedingungen wie z.B. Drücken eines buttons ist möglich.

Als letzte Anweisung der Funktion *menu()* steht die Methode *window_menu.close()*. Wenn die Abbruchbedingung der *while*-Schleife erfüllt ist, geht das Programm zu dieser Zeile weiter und schließt das Fenster der GUI.

Abbildung 97 zeigt beispielhaft die erläuterten grundlegenden Elemente, Konzepte und Syntaxregeln einer GUI mit PySimpleGUI.

(Jones 2021)

Die offizielle Homepage von PySimpleGUI findet sich in (PySimpleGUI 2022). Sie enthält Anleitungen und Beispiele für die Anwendung und Gestaltung von GUIs mit PySimpleGUI. Eine weitere Möglichkeit für Informationen und Beispiele besteht in der Projektbeschreibung des Packages auf der PyPI-Website (PySimpleGUI 2022a).

```

import PySimpleGUI as psg

def menu():
    # Zeile für Filebrowser einfügen zum Suchen der Inputdatei
    layout = [[psg.Text('Input File'),
               psg.Input(size=(25, 1), enable_events=True,
                         key='-FOLDER-'),
               psg.FileBrowse(button_text='Browse',
                             file_types=((('ALL Files', '*.*'),
                                         ('Text', 'txt'))))]
              ], [psg.Button('Solve')]
    ]

    # Fenster erzeugen
    height_launcher, width_launcher = 256, 128
    window_menu = psg.Window('Querschnittswerteberechnung',
                             layout, margins=(height_launcher,
                                               width_launcher))

    # Eventloop einfügen zum kontrolliertem Beenden des Fensters
    while True:
        event, values = window_menu.read()
        if event == 'Exit' or event == psg.WIN_CLOSED:
            break
        window_menu.close()

    # Gesamtes Programm ausführen
if __name__ == '__main__':
    menu()

```

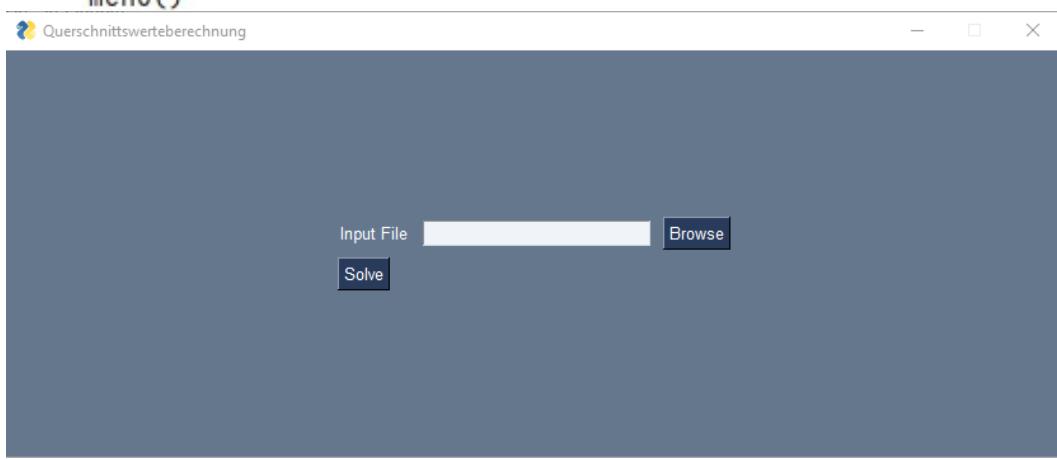


Abbildung 97: Beispiel für GUI mit widgets (Text, Input, FileBrowse, Button) mit PySimpleGUI in Python (erstellt mit PyCharm Community 2022)

8.2.9 Dokumentation mit Package MkDocs

MkDocs ist ein static site generator zur Erstellung von Projektdokumentationen. Auf der Homepage finden sich u.a. ein Installation Guide und ein User Guide. Die documentation source files sind in Markdown geschrieben. Markdown ist eine markup language (wie z.B. LaTeX) zur Formatierung von Dokumenten. (Christie 2014) / (Cone 2022)

Die Installation ist mit pip möglich. Die Eingabe von `mkdocs new my-project` in der command-line erstellt ein neues Projekt (mit einem selbstgewählten Namen statt `my-project`), siehe Abbildung 98. Mit `cd my-project` ist das Verzeichnis (Dateipfad, directory) in der command line ausgewählt. `mkdocs.yml` ist ein configuration file. Ein Ordner `docs` enthält documentation source files in Markdown, u.a. `index.md`.

Name	Kind
<code>mkdocs.yml</code>	YAML
<code>docs</code>	Folder
<code>index.md</code>	Markdown

Abbildung 98: Dateien und Ordner eines neu erstellten Projekts mit MkDocs
(Christie 2014a)

Die Eingabe von `$ mkdocs serve` in der command line (im Dateipfad der Dokumentation) startet einen dev-server mit einer Preview der Dokumentation. Diese aktualisiert automatisch bei Änderungen in der Dokumentation. Änderungen sind u.a. in einem Texteditor möglich.

Im configuration file ist z.B. der Eintrag eines themes möglich. Dazu steht in einer neuen Zeile z.B. `theme: readthedocs`. Mit Eingabe von `mkdocs -help` in der command line erscheinen weitere commands und Optionen.

Mit Eingabe von `mkdocs build` in der command line entsteht ein neues Verzeichnis mit Namen `site`, welches u.a. HTML-files enthält. Das ist die eigentliche Dokumentation. Diese kann hochgeladen und in eine Website eingebaut werden.

(Christie 2014a)

Grundlage der automatisierten Erstellung einer Dokumentation mit MkDocs ist die Verwendung von docstrings im Python-Skript. Dazu gibt es die PEP 257 – Docstring Conventions. Diese beschreiben generelle Syntaxregeln von

docstrings. Sie beinhalten keine markup language-spezifischen Syntax-Regeln innerhalb eines docstrings. Ein docstring ist eine Zeichenkette in dreifachen doppelten Anführungszeichen (""“ ..”“). Diese steht als erste Anweisung in Modulen, Funktionen, Klassen und Methoden. Dadurch wird ein docstring das `__doc__` -Attribut dieses Objekts. Auch Konstruktoren (`__init__` -Methode) erhalten einen docstring. Das gilt ebenfalls für `__init__.py` –Dateien zur Dokumentation des zugehörigen Package.

Einzelige docstrings beschreiben einfache Funktionen und Methoden. Abbildung 99 verdeutlicht Syntax und Inhalt. Ein Aufforderungssatz beschreibt Vorgang und Ergebnis.

```
def function(a, b):
    """Do X and return a list."""
```

Abbildung 99: Beispiel für einzeiligen docstring in Python (Goodger 2022)

Mehrzeilige docstrings beginnen mit einer einzeiligen Zusammenfassung des Objekts, so wie einzelige docstrings. Danach folgt eine Leerzeile (wichtig für automatisierte Dokumentation) und eine genauere Beschreibung. Alle Zeilen eines docstrings sind gleich weit eingerückt, siehe Abbildung 100.

```
def complex(real=0.0, imag=0.0):
    """Form a complex number.

    Keyword arguments:
    real -- the real part (default 0.0)
    imag -- the imaginary part (default 0.0)
    """
    if imag == 0.0 and real == 0.0:
        return complex_zero
    ...
```

Abbildung 100: Beispiel für mehrzeiligen docstring in Python (Goodger 2022)

Docstrings eines Programms (im Hauptmodul) beinhalten Funktion, Dateien, Optionen und Parameter (aus Schnittstellen). Insgesamt ist eine Erklärung des Aufbaus und der Funktionsweise des Programms enthalten.

Docstrings von Modulen listen Klassen, exceptions und Funktionen auf, wenn diese in anderen Modulen Verwendung finden. Jedes beschriebene Objekt erhält einen Einzeiler wie für einzeilige docstrings. Diese dürfen noch kürzer sein.

Docstrings von Funktionen und Methoden enthalten - nach einzeiliger Zusammenfassung und Leerzeile – Parameter (Positions-Parameter und Schlüsselwort-Parameter getrennt), Rückgabewert(e), Nebenwirkungen (side effects), exceptions und Bedingungen zum Aufruf.

Docstrings von Klassen beginnen wie bei Funktionen und Methoden mit einer einzeiligen Zusammenfassung und einer nachfolgenden Leerzeile. Danach folgt eine Auflistung aller öffentlichen (public) Methoden und instance variables. Instance variables sind Attribute, deren Wert-Zuweisung nicht in einer Methode stattfindet. Die Dokumentation eines Konstruktors (`__init__()`) erfolgt im docstring seiner Klasse. Wenn nach dem docstring einer Klasse eine Methode folgt (z.B. `__init__()`), ist eine Leerzeile als Abstand sinnvoll.

(Goodger 2022)

Eine automatisch erstellte Dokumentation eines Python-Programms benötigt neben dem Package MkDocs ein weiteres Hilfsmittel. Auf Github ist ein Programm dafür zum kostenlosen Download (mit Anleitung) vorhanden. (Github 2019) Weiterhin gibt es (Sphinx 2022) und (Read the Docs 2022) als Möglichkeiten. (Slatkin 2020), S. 452

Als Hinweis: In Python sind docstrings mit dem Attribut `__doc__` ausgebbar, z.B. in der Python-Shell, siehe Abbildung 101. (Slatkin 2020), S. 451

```
def palindrome(word):
    """True zurückgeben, falls das übergebene Wort
       ein Palindrom ist."""
    return word == word[::-1]
print(repr(palindrome.__doc__))
>>>
'True zurückgeben, falls das übergebene\n
Wort ein Palindrom ist.'
```

Abbildung 101: Beispiel für docstring-Ausgabe mit `__doc__` in Python (Slatkin 2020), S. 451

8.2.10 PEP 8 - Styleguide für Python

Der Pionier von Python – Guido van Rossum – liefert einige Empfehlungen und Richtlinien zum Gestalten von Quellcode. Das Ziel ist Lesbarkeit („Readability counts“) und Verständlichkeit. Das Mittel ist Konsistenz. Abweichende projektspezifische Richtlinien und Lesbarkeit gehen vor die Einhaltung der PEP 8 - Richtlinien. „One of Guido's key insights is that code is read much more often than it is written. The guidelines provided here are intended to improve the readability of code and make it consistent across the wide spectrum of Python code.“

Die PEP 8 – Richtlinien umfassen Themen wie Verwendung von Leerzeichen, Leerzeilen und Einrückungen, Kommentaren und Docstrings, Konventionen bei Namen sowie Empfehlungen für die Verwendung von Funktionen und Variablen.

PEP 8 empfiehlt 4 Leerzeichen als Einrückung zur Kennzeichnung von untergeordneten Anweisungen. Geht eine Anweisung über mehr als eine Zeile, ist eine größere Einrückungstiefe als die des untergeordneten Anweisungsblocks zu wählen, siehe Abbildung 102. Andersherum soll nicht mehr als eine Anweisung pro Zeile stehen.

```
# Aligned with opening delimiter.
foo = long_function_name(var_one, var_two,
                         var_three, var_four)

# Add 4 spaces (an extra level of indentation)
def long_function_name(
    var_one, var_two, var_three,
    var_four):
    print(var_one)

# Hanging indents should add a level.
foo = long_function_name(
    var_one, var_two,
    var_three, var_four)

# Add some extra indentation on the conditional
if (this_is_one_thing
    and that_is_another_thing):
    do_something()
```

Abbildung 102: Beispiele für Einrückung bei mehrzeiligen Anweisungen nach PEP 8 in Python (van Rossum 2022)

Mehrzeilige Listen schließen mit der Klammer in der nächsten Zeile, siehe Abbildung 103. Beide gezeigten Varianten sind möglich.

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
```

```
my_list = [
    1, 2, 3,
    4, 5, 6,
]
```

Abbildung 103: Beispiele für Position von schließenden Klammern von Listen nach PEP 8 in Python (van Rossum 2022)

Die maximale Zeichenanzahl in einer Zeile ist 79. Zeilen mit Kommentaren und Docstrings sind maximal 72 Zeichen lang. Anführungszeichen und Klammern führen in Python zur automatischen Erkennung eines Zeilenumbruchs. Das Setzen von zusätzlichen Klammern oder die Verwendung eines Backslash ist für mehrzeilige Anweisungen empfohlen.

```
with open('/path/to/some/file/you/want/to/read') as file_1, \
    open('/path/to/some/file/being/written', 'w') as file_2:
    file_2.write(file_1.read())
```

Abbildung 104: Beispiel für Zeilenumbruch mit Backslash nach PEP 8 in Python (van Rossum 2022)

Zeilenumbrüche in Anweisungen mit Operatoren finden direkt vor den Operatoren statt, siehe Abbildung 105.

```
# Correct:
# easy to match operators with operands
income = (gross_wages
          + taxable_interest
          + (dividends - qualified_dividends)
          - ira_deduction
          - student_loan_interest)
```

Abbildung 105: Beispiel für Zeilenumbruch in Klammerausdruck vor Operatoren nach PEP 8 in Python (van Rossum 2022)

Definitionen von Top-Level Funktionen und Klassen sind von (jeweils) zwei Leerzeilen umgeben. Definitionen von Methoden sind von einer Leerzeile umgeben. Zusätzliche Leerzeilen sind - sparsam eingesetzt - zur Kennzeichnung inhaltlich zusammenhängender / getrennter Anweisungen möglich.

import-Anweisungen stehen in separaten Zeilen. *from package import module* bzw. *from module import function* -Anweisungen sind in einer Zeile möglich, siehe Abbildung 106.

```
# Correct:
import os
import sys
```

```
# Correct:
from subprocess import Popen, PIPE
```

Abbildung 106: Beispiele für *import*-Anweisungen nach PEP 8 in Python (van Rossum 2022)

import-Anweisungen stehen am Anfang des Moduls / Python-Skripts. Davor erscheinen ggf. beschreibende Kommentare für das Modul / docstrings. Danach folgen Konstanten und globale Variablen. *import*-Anweisungen sind in drei Gruppen einzuteilen: „1. Standard library imports 2. Related third party imports 3. Local application / library specific imports“. Zwischen den Gruppen befinden sich Leerzeilen. Wenn möglich importiert die Anweisung das gesamte Modul. Die Verwendung von *from module import **-Anweisungen ist nicht empfohlen.

Die Verwendung von einfachen oder doppelten Anführungszeichen ist im Programm konsistent auszuführen. Für dreifache Anführungszeichen - wie z.B. bei Docstrings – sind doppelte Anführungszeichen üblich.

Leerzeichen dienen – sparsam eingesetzt – zur optischen Gliederung von Anweisungen (Zuweisungen, Ausdrücken, Operationen). Nach einem Komma oder Semikolon folgt ein Leerzeichen. Nach einem trailing comma folgt kein Leerzeichen, siehe Abbildung 107.

<pre># Correct: spam(ham[1], {eggs: 2})</pre>	<pre>foo = (0,) if x == 4: print(x, y); x, y = y, x</pre>
---	--

Abbildung 107: Beispiele für Leerzeichen nach Komma oder Semikolon nach PEP 8 in Python (van Rossum 2022)

Um einen Zuweisungsoperator ist genau ein Leerzeichen üblich, siehe Abbildung 108. Das gilt für alle Zuweisungsoperatoren (neben = sind das z.B. +=, -=), Vergleichsoperatoren (==, <, >, !=, <>, <=, >=, in, not in, is, is not) und logischen Operatoren (and, or, not). Die in der Rangfolge höchsten Operatoren erhalten ggf. zur optischen Gliederung keine umgebenden Leerzeichen. Ein Operator ist dabei immer auf beiden Seiten gleichmäßig mit/ohne Leerzeichen.

<pre># Correct: x = 1 y = 2 long_variable = 3</pre>	<pre>i = i + 1 submitted += 1 x = x*2 - 1 hypot2 = x*x + y*y c = (a+b) * (a-b)</pre>
---	--

Abbildung 108: Beispiel für Leerzeichen um Operatoren nach PEP 8 in Python (van Rossum 2022)

Für den Zuweisungsoperator = gibt es in Schlüsselwort-Parametern von Funktionen zwei Regeln. Bei der Zuweisung von *int* / *float* – Werten (Zahlenwerten) oder Variablen kommen keine Leerzeichen um =. Bei der Zuweisung von vorgegebenen Werten wie z.B. *None* oder *True* / *False* schon, siehe Abbildung 109.

```
# Correct:  
def complex(real, imag=0.0):  
    return magic(r=real, i=imag)
```

```
# Correct:  
def munge(sep: AnyStr = None): ...
```

Abbildung 109: Beispiele für Leerzeichen in Schlüsselwort-Parametern von Funktionen nach PEP 8 in Python (van Rossum 2022)

Trailing commas (angehängte Kommas ohne nachfolgenden Wert) dienen zur Erstellung von Tuples mit einem Wert. Zur eindeutigen Darstellung ist das Tuple in runde Klammern eingeschlossen, siehe Abbildung 110.

```
# Correct:  
FILES = ('setup.cfg',)
```

Abbildung 110: Beispiel für trailing comma und runde Klammern bei Erstellung von Tuple mit einem Wert nach PEP 8 in Python (van Rossum 2022)

Kommentare unterliegen einigen Richtlinien:

- Es befinden sich keine veralteten oder unnützen Kommentare im Quellcode (don't „state the obvious“).
- Kommentare sollen ganze Sätze sein bzw. beinhalten.
- Das erste Wort im Kommentar ist großgeschrieben (außer es ist ein im Programm oder in Python verwendeter Ausdruck, z.B. Funktionsname).
- Ein Satz schließt mit einem Punkt. Nach dem Punkt folgen zwei Leerzeichen.
- Die bevorzugte Programmiersprache von Kommentaren ist englisch.

Blockkommentare stehen über einem Anweisungsblock (gleich weit eingerückt) und beschreiben diesen. Jede Zeile startet mit einem Doppelkreuz (Hashtag, #). Einer Anweisung angehängte Kommentare (inline comments) sind sparsam zu verwenden. Der Abstand zur Anweisung beträgt mindestens zwei Leerzeichen. Sie geben zusätzliche, nützliche Informationen, siehe Abbildung 111.

```
x = x + 1          # Compensate for border
```

Abbildung 111: Beispiel für inline comment nach PEP 8 in Python (van Rossum 2022)

Die empfohlene Namensgebung (naming convention) in Python wird nachfolgend beschrieben. Es gibt folgende Stile der Namensgebung (naming styles):

- `b` (single lowercase letter)
 - `B` (single uppercase letter)
 - `lowercase`
 - `lower_case_with_underscores`
 - `UPPERCASE`
 - `UPPER_CASE_WITH_UNDERSCORES`
 - `CapitalizedWords` (or CapWords, or CamelCase – so named because of the bumpy look of its letters [4]). This is also sometimes known as StudlyCaps.
- Note: When using acronyms in CapWords, capitalize all the letters of the acronym. Thus `HTTPServerError` is better than `HttpServerError`.
- `mixedCase` (differs from CapitalizedWords by initial lowercase character!)
 - `Capitalized_Words_With_Underscores` (ugly!)

Abbildung 112: Allgemeine Stile der Namensgebung (naming styles) (van Rossum 2022)

Namen, die in einer Benutzeroberfläche / Anwendungsschnittstelle (Programmierschnittstelle, Application Programming Interface, API) sichtbar sind, folgen den Konventionen / Richtlinien von APIs statt PEP 8. I (kleines EI), O (großes Oh) und I (großes Ih) sind nicht als einzige Zeichen von Namen erlaubt. Die Verwechslungsgefahr mit 0 (null) und 1 (eins) ist zu groß.

Modulnamen benutzen `lowercase` oder `lower_case_with_underscores`. Packagename sind mit `lowercase` zu erstellen.

Klassennamen benutzen `CamelCase`. Dazu gehören auch Exception names. Wenn es sich bei der Exception um einen Fehler handelt, enthält sie die Endung `Error`.

Funktionen, Methoden und Variablen verwenden `lowercase` / `lower_case_with_underscores`. Bei Namenskollisionen haben Methoden- und Attributnamen ggf. einen / zwei vorgesetzte oder angehängte `underscore _`.

Der erste Parameter von Objektmethoden (instance methods) ist `self`, der erste von Klassenmethoden (class methods) `cls`.

Konstanten werden meist auf Top Level des Moduls definiert und verwenden `UPPERCASE` oder `UPPER_CASE_WITH_UNDERSCORES`.

Schließlich folgen in PEP 8 Tipps und Empfehlungen zum Programmieren selbst. Davon sind einige für die Masterarbeit nützliche Themen kurz erklärt.

Sequentielle Datentypen sollen mit der Methode `.join()` zusammengefügt werden (nicht mit Operator `+`). Vergleiche mit singletons (z.B. `None`) verwenden `is / is not` statt `== / !=` und `not ... is`, siehe Abbildung 113.

```
# Correct:
if foo is not None:
```

Abbildung 113: Beispiel für Vergleich mit singleton `None` nach PEP 8 in Python (van Rossum 2022)

Eine Definition einer Funktion mit `def` ist einem Lambda-Ausdruck (`lambda expression`) vorzuziehen, ggf. auch in einer Zeile, siehe Abbildung 114.

```
# Correct:
def f(x): return 2*x

# Wrong:
f = lambda x: 2*x
```

Abbildung 114: Beispiel für Verwendung von einzeiliger definierter Funktion statt Lambda-Ausdruck nach PEP 8 in Python (van Rossum 2022)

Eigene Exceptions werden von der Klasse `Exception` statt von `BaseException` abgeleitet. Fehlermeldungen enthalten Informationen zur Fehlerursache.

Bei Funktionen mit Rückgabewert sollen alle `return`-Anweisungen einen Rückgabewert haben (auch `None`), siehe Abbildung 115.

```
def foo(x):
    if x >= 0:
        return math.sqrt(x)
    else:
        return None
```

Abbildung 115: Beispiel für Rückgabewerte von Funktionen nach PEP 8 in Python (van Rossum 2022)

`.startswith()` und `.endswith()` sind für das Prüfen von Präfix und Suffix von Strings anzuwenden, nicht string slicing, siehe Abbildung 116.

```
# Correct:
if foo.startswith('bar'):

# Wrong:
if foo[:3] == 'bar':
```

Abbildung 116: Beispiel für Präfix-Prüfung nach PEP 8 in Python (van Rossum 2022)

Ein Vergleich von Objekttypen verwendet die Standardfunktion `isinstance()`, siehe Abbildung 117.

```
# Correct:
if isinstance(obj, int):
```

Abbildung 117: Beispiel für Objekttyp-Vergleich nach PEP 8 in Python (van Rossum 2022)

Die Prüfung, ob sequentielle Daten Werte enthalten, funktioniert wie in Abbildung 118.

```
# Correct:
if not seq:
if seq:
```

Abbildung 118: Beispiel für Prüfung auf Inhalt bei sequentiellen Daten nach PEP 8 in Python (van Rossum 2022)

Boolesche Werte benötigen ggf. keinen Operator (`==`, `!=`) in einer Bedingung, siehe Abbildung 119. `greeting` im Beispiel enthält einen booleschen Wert.

```
# Correct:
if greeting:

# Wrong:
if greeting == True:
```

Abbildung 119: Beispiele für Booleschen Wert in Bedingung nach PEP 8 in Python (van Rossum 2022)

Viele IDEs haben Möglichkeiten zur automatischen Überprüfung bzw. Einhaltung der PEP 8 – Richtlinien. „In Spyder, for example, enable Real time code analysis in Tools → Preferences → Editor → Code Introspection/Analysis and “Automatically remove trailing spaces when saving files” in in Tools → Preferences → Editor → Advanced Settings.“ Es bietet sich die Installation von Hilfstools wie Flake8 (Stapleton 2016) und/oder autopep8 (Python 2022a) an. (SciPy 2022)

8.2.11 Grundlagen des Software-Designs

Software Engineering liefert standardisierte Prozesse zum Design und zur Entwicklung von Software. Es enthält Themen wie Anforderungen, Systemanalyse, Systemdesign, Wartung, Updates, Softwareverteilung, Tests und Quellcode-Design. Das Ziel von Software Engineering ist die Optimierung des Managements großer Software-Pakete, der Skalierbarkeit, des Kostenmanagements, der Flexibilität und des Qualitätsmanagements.

Besonders bei großen Programmen ist ein standardisierter und geordneter Prozess von Vorteil. Flexibilität beinhaltet Möglichkeiten zur einfachen Veränderung und Anpassung der Software an neue oder ähnliche Probleme (statt einer Neuentwicklung). Kosten entstehen hauptsächlich durch Arbeitszeit der Programmierer/innen. Die Entwicklungszeit eines Software-Projekts ist daher möglichst kurz und effizient genutzt. Eine mögliche Erweiterung der Software mit Updates ist bei der Erstentwicklung zu berücksichtigen. Qualitätsmanagement ist bereits in der Entwicklungsphase wichtige für die Produktqualität.

Ein wichtiger Schritt ist die Aufteilung eines komplexen Systems in kleinere Problemstellungen. Die Lösung dieser Teilprobleme erfolgt unabhängig voneinander.

Die Reduzierung von Entwicklungskosten bzw. Arbeitsstunden erfordert einen Fokus auf wesentliche Software-Funktionalitäten und das Weglassen nicht unbedingt erforderlicher Arbeitsschritte. Eine effiziente Methodik zur Erstellung verschiedener Quellcode-Varianten und deren Vergleich führt zu einer weiteren Reduzierung des Zeitaufwands im Projekt.

Eine Anforderung an Software ist eine zuverlässige und wenig fehleranfällige Anwendung. Daher gehört bug-fixing (Wartung, maintenance) auch zum Software Engineering dazu.

(JavaTpoint 2021)

Der Begriff Software umfasst eine Menge an Programmen, Bedienungsanleitungen und zugehörige Dokumente / Dokumentation zur Beschreibung der Programme und ihrer Benutzung. Ein Software-Prozess beinhaltet Schritte zur Erstellung eines Software-Produkts:

1. Spezifikation – definiert Funktionsumfang und Einsatzbedingungen
2. Entwicklung – umfasst Erstellung der Software
3. Test – prüft auf fehlerfreie Ausführung der Funktionalitäten nach Spezifikation

-
- 4. Evolution – enthält Schritte zur Weiterentwicklung der Software und Anpassung an neue / geänderte Kunden-Anforderungen

Software-Prozess-Modelle sind Werkzeuge zur Planung eines Software-Prozesses. Das sind:

- 1. Workflow-Modell – zeigt menschliche Aktivitäten mit Eingaben, Ausgaben und Abhängigkeiten
- 2. Dataflow- / Activity-Modell – zeigt menschliche / automatisierte Aktivitäten zur Datentransformation (z.B. Spezifikation zu Entwicklung / Design), ein Dataflow-Modell ist ggf. genauer als ein Workflow-Modell
- 3. Rollen-Modell – zeigt Rollen und Verantwortungsbereiche beteiligter Personen

Es gibt einige generelle Modelle / Paradigmen der Software-Entwicklung:

- 1. Waterfall-Modell – arbeitet Schritte im Software-Prozess nacheinander ab (Spezifikation, Software design, Implementierung, Test, Betrieb und Wartung)
- 2. Evolutionary development – mischt Schritte im Software-Prozess, schafft schnell eine erste Version aus einer stark abstrahierten Spezifikation
- 3. Formal transformation – basiert auf einem mathematischen Modell / System als Spezifikation und leitet daraus mathematische Methoden für ein Programm ab
- 4. System assembly from reusable components – integriert bestehende Teile eines Systems in neues Software-Pakete

Im Software-Prozess ist auf kritische Faktoren zu achten:

- 1. Größe (size) – berücksichtigt wachsende Komplexität der Software, verursacht durch umfangreichere Kundenanforderungen
- 2. Qualität (quality) – benötigt Tests zur Verringerung der Fehleranzahl pro SLOC
- 3. Kosten (cost) – meint Kosten pro SLOC
- 4. Termin der Fertigstellung – umfasst Einhaltung gesetzter Termine

Der Begriff Software beinhaltet Programme, Dokumentation und Bedienungsanleitungen. Die Dokumentation besteht aus verschiedenen Dokumenten für Spezifikation, Design, Implementierung und Tests, siehe Abbildung 120.

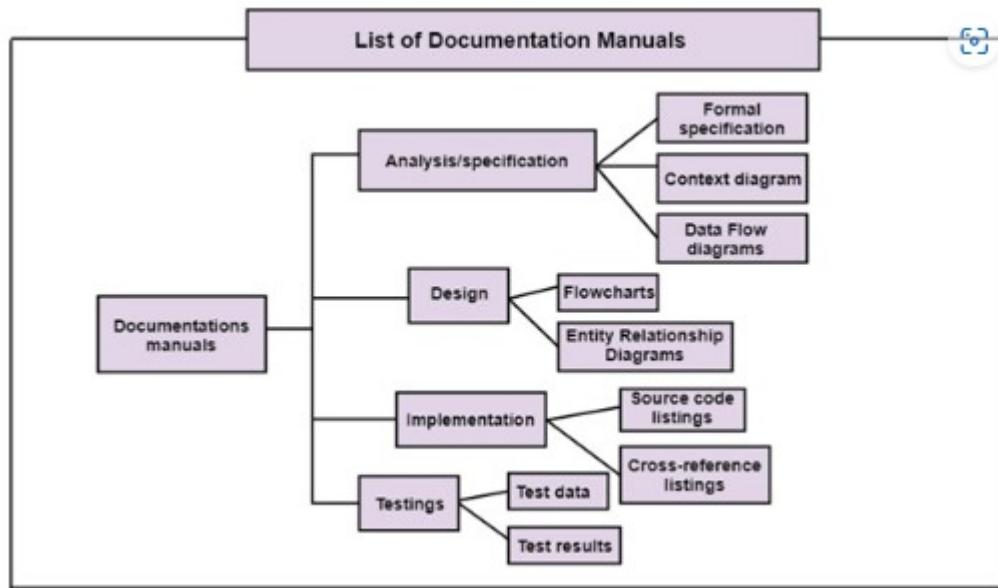


Abbildung 120: Einteilung von Dokumentation für Software nach Schritten eines Software-Prozesses

Bedienungsanleitungen bestehen aus verschiedenen Anleitungen, z.B. beginners guide, system overview und installation guide, siehe Abbildung 121.

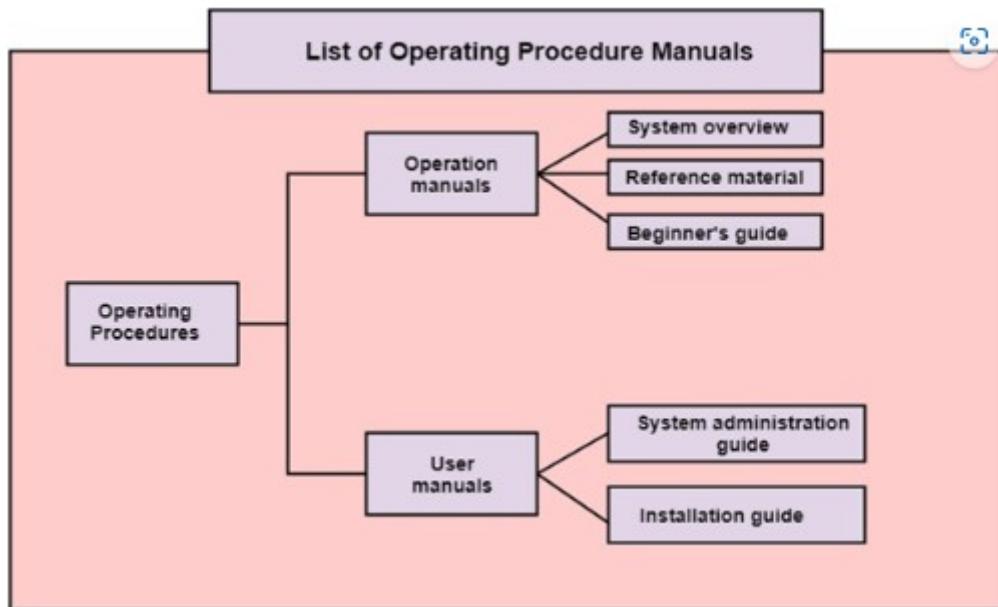


Abbildung 121: Einteilung von Bedienungsanleitungen für Software nach Inhalt

(Java Tpoint 2021a)

Winston Royce führte das Waterfall-Modell 1970 ein. Es besteht aus fünf Phasen eines Software Development Life Cycles (SDLC), siehe Abbildung 122. Die

Phasen orientieren sich an den Schritten im Software-Prozess (Spezifikation, Entwicklung, Test, Evolution). Das Waterfall-Modell empfiehlt eine Abarbeitung der Phasen in der gegebenen Reihenfolge (keine Vermischung oder Überlappung).

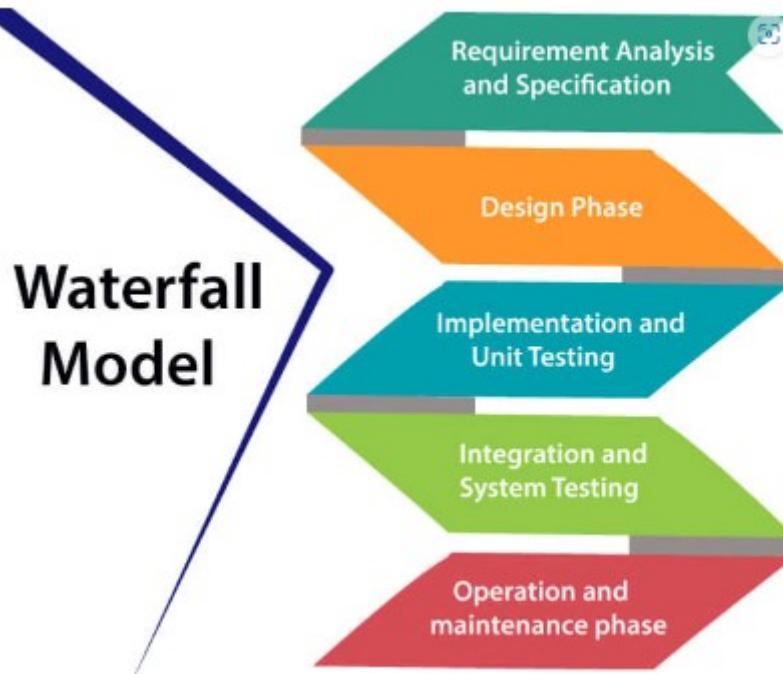


Abbildung 122: Phasen eines Software Development Life Cycle (SDLC) im Waterfall-Modell nach Winston Royce

Requirements analysis and specification phase – Beinhaltet Sammlung und Dokumentation der Kundenanforderungen. Die so erstellte Software Requirement Specification (SRS) entspricht einem Lastenheft im Maschinenbau-Entwicklungsprozess. Sie beschreibt Programmiersprachen-unabhängig das Problem sowie Erwartungen des Kunden bezüglich Funktionalität, Leistung und interfaces der Software. Sie enthält keine Informationen zu den konkreten (Teil-)Lösungen.

Design phase – Die SRS ist Grundlage zur Erstellung eines Software Design Document (SDD). Dieses beinhaltet abstrakte und detaillierte Beschreibungen der Software-Architektur. Ein SDD entspricht einem Pflichtenheft.

Implementation and unit testing – Diese Phase umfasst Umsetzung und getrennte Tests einzelner Funktionen, Klassen und Module und ihrer Interaktion untereinander. Dazu prüft der/die Programmierer/in Zwischenergebnisse, z.B. in Form von Rückgabewerten von Funktionen / Methoden oder Attributwerten von Instanzen.

Integration and System Testing – Diese Phase stellt die Kundenzufriedenheit und Qualität der Ergebnisse sicher und verringert anschließenden Wartungsaufwand. Es stehen Test der Interaktion von Modulen und des Gesamtsystems im Fokus. Operation and maintenance phase – Nach Auslieferung der Software an Anwender/innen beginnt diese Phase. Sie umfasst Benutzung und Wartung.

Anwendungsgebiete des Waterfall-Modells sind unter folgenden Bedingungen:

- Anforderungen stehen bei Projektbeginn fest und sind unveränderlich
- kleines Projekt
- verwendete Technologien und verfügbare Mittel stehen fest
- alle Ressourcen sind einsatzbereit und stehen zur Verfügung

Vorteile und Nachteile des Waterfall-Modells sind in Tabelle 7 zusammengefasst.

Tabelle 7 Vor- und Nachteile des Waterfall-Modells nach Winston Royce (JavaTpoint 2021b)

Vorteile	Nachteile
Einfache Umsetzung	Nicht geeignet für wichtige und komplexe Projekte
Benötigt wenig Ressourcen	Keine Änderung von Anforderungen möglich
Einfache, explizite und unveränderliche Formulierung von Anforderungen	Höheres Risiko, da Tests erst spät im Projekt erfolgen
Mit Terminierung der Phasen ist Projektfortschritt einfach messbar	
Abschlusstermin und Kosten des Projekts stehen zu Projektbeginn fest	

(JavaTpoint 2021b)

In der design phase eines SDLC steht die Erstellung eines SDD für die konkrete Umsetzung einer Software im Vordergrund. Dazu gibt es einige software design principles. Diese umfassen die Themen problem partitioning, abstraction, modularity und topdown / bottom up strategy.

Problem partitioning - Umfangreichere Probleme erfordern eine Einteilung in Teilprobleme. Das vereinfacht nachfolgende Schritte, wie z.B. Tests, Änderungen und Erweiterungen. Die Software wird insgesamt weniger kompliziert (verständlicher, strukturierter). Dafür wird sie aber ggf. komplexer

(umfangreicher), da Teilsysteme miteinander kommunizieren und interagieren müssen.

Abstraction – Abstraktion bedeutet eine Verringerung des Detaillierungsgrades der Betrachtung eines Problems oder Systems. D.h. verschiedene Funktionalitäten werden zusammengefasst (z.B. in einem Modul). Es stehen die Interaktionen, Ein- und Ausgaben der Betrachtungseinheit im Vordergrund. Interne Algorithmen sind bei diesem software design principle nicht wichtig. Es findet eine Unterscheidung in functional abstraction und data abstraction statt. Daraus leiten sich Function oriented design approaches und Object Oriented design approaches ab.

Modularity – Eine Einteilung eines Programms in Module erzeugt eine bessere Verständlichkeit und Übersichtlichkeit. Einzelne Module bilden die Grundlage für separate Arbeitspakete. Module haben optimalerweise folgende Eigenschaften:

- Module sind nahezu selbstständige Systeme und somit auch in anderen Programmen einsetzbar
- Module haben jeweils definierte Zielsetzungen und Aufgaben
- Module sind separat kompilier- und speicherbar
- Module sind einfach in der Anwendung
- Module haben möglichst wenige Schnittstellen nach außen (Komplexität innerhalb ist möglich)

Einige Vor- und Nachteile von modularity stehen sich in Tabelle 8 gegenüber.

Tabelle 8 Vor- und Nachteile von modularity im Software-Design

Vorteile	Nachteile
Ermöglicht Bearbeitung durch verschiedene Programmier/innen	Erfordert mehr SLOC und mehr Datentransfer
Ist Grundlage zur Erschaffung einer Bibliothek mit Modulen für verschiedene Programme	Schafft ggf. zusätzliche Probleme bei Interaktion verschiedener Module
Erleichtert das Laden eines Programms	Verlängert ggf. Ausführungszeit (run time), Kompilierungszeit und Ladezeiten
Liefert Termine zum Messen des Projektfortschritts	Erhöht ggf. Speicherplatzbedarf
Schafft Grundlage für einfache, systematische Tests	
Ergibt ein lesbaren, verständliches Programm	

Ein modularer Aufbau eines Programms heißt modular design. Ein Ziel und Qualitätsmerkmal von modular design ist functional independence. Funktionen erfüllen genau eine Aufgabe und interagieren möglichst wenig mit anderen Modulen. Das vereinfacht Implementierung, Tests, Fehlerfindung / Debugging, Wartung und Wiederverwendung in anderen Programmen. Cohesion und coupling sind zwei Begriffe zum Messen der Qualität eines Moduls. Cohesion beschreibt die Zusammenfassung von ähnlichen Funktionen / Funktionalitäten in ein Modul (wünschenswert). Coupling gibt die Abhängigkeiten (interdependence) zwischen Modulen an (möglichst wenige und einfache ist gut).

Modular design beinhaltet weiterhin information hiding. Daten innerhalb eines Moduls (oder Attribute einer Klasse) sind vor Zugriff von außen geschützt. Sie werden nur über definierte Schnittstellen (Funktionen, Methoden) verändert und weitergegeben. Das erleichtert die Zuordnung von Fehlermeldungen.

Generell gibt es für das Design eines Systems zwei Strategien, top down und bottom up. Top down beginnt beim Gesamtsystem und zerlegt dies – ggf. über mehrere Stufen – in Teilsysteme. Bottom up beginnt in der Stufe mit dem höchsten Detailgrad und baut daraus übergeordnete Teilsysteme bzw. das Gesamtsystem zusammen. Letzteres ist besonders für die Analyse bestehender Systeme von Vorteil.

(JavaTpoint 2021c)

Programmiersprachen-unabhängige Design-Prinzipien (design principles) enthalten Empfehlungen zum Design eines Systems, der Implementierung von Teilsystemen und der Gestaltung von Quellcode:

- Divide and conquer – ist allgemeine Problemlösungsstrategie; teilt komplexe und umfangreiche Probleme in Teilprobleme auf; löst Teilprobleme eins nach dem anderen (z.B. Aufteilung eines Programms zu → Packages → Module → Classes → Methoden → ggf. Funktionen, Beginn der Abarbeitung bei Funktionen und Methoden)
- Increase cohesion – Eine Gruppe (z.B. Package, Modul, Klasse) beinhaltet inhaltlich / thematisch zusammenhängende Objekte, Methoden und Funktionen, z.B. *math*-Package hat mathematische Funktionen
- Reduce coupling – Eine Gruppe hat möglichst wenige Interaktionen / Schnittstellen zu möglichst wenigen anderen Gruppen; Fehler sind bei viel coupling nicht eindeutig zuordenbar; führt langfristig zu weniger Aufwand und mehr Flexibilität und Quellcode-Qualität

- Increase abstraction – generalisiert Probleme und erschafft möglichst allgemeine Lösungen (konkret: Dreieck → abstrakt: geometrische Figur)
- Increase reusability – basiert auf Abstrahierung; investiert mehr Zeit bei Erstentwicklung zum Sparen von Zeit bei Weiterentwicklung / Änderung (z.B. soft coding statt hard coding, also Variablen und Funktionen statt konkreten Werten im Quellcode); vermeidet spezialisierte Lösungen
- Design for flexibility – basiert auf reduce coupling und increase abstraction; berücksichtigt Skalierbarkeit und mögliche Anpassungen an geänderte Anforderungen in der Zukunft
- Anticipate obsolescence – nutzt möglichst wenige externe Abhängigkeiten (external dependencies); d.h. möglichst wenig Verwendung von Software Dritter (third party software) als Hilfssysteme für eigene Software; vermeidet Versionen aus frühem Entwicklungsstadium, wenig bekannten Unternehmen, Software mit schlechter Dokumentation und Wartung
- Design for portability – designt Software für mögliche Nutzung auf verschiedenen Plattformen, Geräten und Betriebssystemen
- Design for testability – ist für komplexe Projekte / Systeme besonders wichtig; berücksichtigt bei Programmierung spätere Testmöglichkeiten (z.B. als unit test, functional test)
- Design defensively – geht nicht von Kenntnis der Software und Programmierkenntnissen eines/einer Anwenders / Anwenderin (user) aus; liefert gute Error messages (sagen genau was falsch läuft, bzw. liefern im Internet beschriebene Error messages); designt input / output eines Programms robust, d.h. Schnittstellen mit user sind intuitiv und einfach verwendbar

(Tech With Tim 2021)

Neben Syntax und Konzepten einer Programmiersprache ist Planung und Design / Architektur (Strukturierung) von Systemen wichtig. Die Vorgehensweise ist größtenteils Programmiersprachen-unabhängig. (Tech With Tim 2020), 0:00 Ein erster Schritt ist die Übersetzung eines Problems in ein Programm-Design. Die Erstellung von Klassen ist ein sinnvoller Anfang. (ebd.), 5:00 Dann folgt eine Zuordnung von Informationen aus dem Problem zu den erstellten Klassen. Informationen stellen somit Attribute oder Methoden einer Klasse dar. Jede Information erhält einen Datentyp. (ebd.), 9:30 Wenn Klassen gemeinsame Attribute aufweisen, ist ggf. die Verwendung einer Oberklasse sinnvoll (inheritance). (ebd.), 15:00

Im nächsten Schritt folgt die Beschreibung von Interaktionen (associations) zwischen Klassen bzw. deren Instanzen. Das bisher beschriebene Vorgehen liefert für das Beispiel der Kursbelegung an einer Schule als Ergebnis die Liste / Skizze in Abbildung 123. Eine Linie stellt associations zwischen Instanzen der verbundenen Klassen dar. Eine Zahl an einer Linie neben einer Klasse benennt die Anzahl von Instanzen, die Teil einer association sind bzw. sein können (* steht für unendlich). (ebd.), 19:05, 27:15 Linien mit Pfeil (zeigt auf Oberklasse) verdeutlichen inheritance. (ebd.), 25:15 Im Beispiel stellt enrol (unten rechts) eine association class dar. Eine association class verknüpft zwei andere Klassen (hier Student und Course) und hat für sich alleine keine Funktionalität. Sie wird benötigt, wenn mehrere Instanzen beider Klassen mit mehreren Instanzen der jeweils anderen Klasse assoziiert werden können (hier können mehrere Students den gleichen Course haben und ein Course ist mehreren Students zugeordnet). Enrol z.B. enthält die Attribute date und grade (Einschreibdatum und Note von Students in ihren Courses). (ebd.), 32:10 Ein zusätzliches Attribut in einer der beiden an einer association beteiligten Klassen beinhaltet assoziierte Instanzen der anderen Klasse (ggf. als Liste zur Speicherung mehrerer assoziierter Instanzen). (ebd.), 38:25

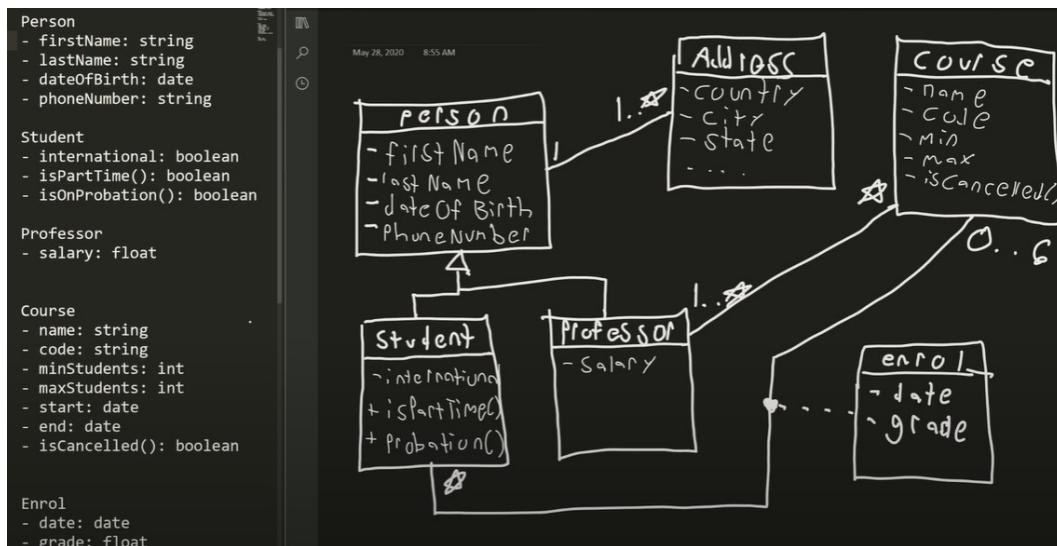


Abbildung 123: Übersetzung eines Problems in Klassen mit Attributen und Methoden (links) und associations zwischen Klassen (rechts) am Beispiel einer Kursbelegung an einer Schule (Tech With Tim 2020), 34:30

Der nächste Schritt umfasst die Erstellung von Quellcode für die beschriebenen Klassen. (Tech With Tim 2020a), 0:00 Die einfachste Klasse bildet den Anfang in der Programmierung. Sie hat die wenigsten / einfachsten associations, Attribute und Methoden (hier Adress, hat nur eine association und wenige Attribute).

Danach folgt die Reihenfolge der Programmierung von Klassen den associations wie einem Pfad (hier Person als zweites). (ebd.), 1:10

Eine einfache Vorgehensweise ist die Erstellung eines Moduls pro Klasse (*classname.py*). An erster Stelle steht die Definition eines Konstruktors (*def __init__()*) für die erste Klasse. Wenn bestimmte Attribute für die Instanziierung eines Objekts der Klasse erforderlich sind, erhält der Konstruktor Parameter und Zuweisungen dieser Parameter zu Attributen. Attribute, deren Instanziierung später erfolgen kann, sind nicht im Konstruktor enthalten. (ebd.), 2:50

Ein weiterer Begriff ist directional association (one way or two way). Im Beispiel von Klasse *Person* und Klasse *Address* benötigt eine Instanz von *Person* eine Übergabe von Attributwerten (*country*, *city*, *state*, *street*, etc.) von Instanz(en) von *Address*. Die Instanz(en) von *Address* erhält keine Information über assoziierte Person(en). Dieses Beispiel gehört zu one directional association, d.h. eine Klasse (hier *Person*) erhält view (Zugriff auf Attributwerte) einer anderen Klasse (hier *Address*). Für eine Übergabe von Werten als Parameter zwischen Klassen ist – wegen der Aufteilung in Module - ein Import einer assoziierten Klasse nötig, z.B. *from address import Address* im Modul *person*.

Die Benennung von Parametern ist intern und daher auch kurz möglich. Attributnamen sind eventuell extern sichtbar und daher nach PEP 8 – Richtlinie optimal, vgl. Unterkapitel 8.2.10. (ebd.), 4:10 Wenn Parameter übergeben werden, ist eine Prüfung auf passenden Datentyp sinnvoll, da Python dynamisch typisiert ist, vgl. Unterkapitel 8.2.4. Im Beispiel führen *if/elif/else*-Anweisungen eine Prüfung auf Datentyp und auf einfache oder mehrfache Übergabe von Instanzen der Klasse *Address* aus, siehe Abbildung 124. Wenn der Datentyp des Parameters nicht dem erforderlichen Datentyp entspricht, erscheint eine Fehlermeldung. (ebd.), 8:30

```

1 from address import Address
2
3 class Person:
4     def __init__(self, first, last, dob, phone, address):
5         self.first_name = first
6         self.last_name = last
7         self.date_of_birth = dob
8         self.phone = phone
9         self.addresses = []
10
11     if isinstance(address, Address):
12         self.addresses.append(address)
13     elif isinstance(address, list):
14         for entry in address:
15             if not isinstance(entry, Address):
16                 raise Error("Invalid Address...")
17
18         self.addresses = address
19     else:
20         raise Error("Invalid Address...")
21
22     def add_address(self, address):
23         if not isinstance(address, Address):
24             raise Error("Invalid Address...")
25
26         self.addresses.append(address)

```

Abbildung 124: Konstruktor einer Klasse *Person* mit Parametern, Attributen und Prüfung des Datentyps eines Parameters (*address*) am Beispiel einer Kursbelegung an einer Schule (Tech With Tim 2020), 14:30

Als nächstes steht im Beispiel die Klasse *Student* im Fokus. Die Vorgehensweise entspricht generell der Programmierung der Klassen *Person* und *Address*. Die erste Anweisung in der neuen Klasse ist wieder ein Konstruktor `def __init__()`. Eine Besonderheit hier ist inheritance, d.h. die Vererbung von Attributen und Methoden der Überklasse *Person*. Dieses berücksichtigt ein im Konstruktor der Subklasse *Student* eingeordneter `super().__init__()`-Konstruktor. Dieser übergibt die gleichen Parameter wie die Überklasse(n). Der Konstruktor der Subklasse enthält ebenfalls diese Parameter plus Parameter der Subklasse (hier Schlüsselwort-Parameter *international=false*). (ebd.), 14:45

Eine weitere Besonderheit der Klasse *Student* ist eine association mit der association class *Enroll*. Dazu wird ein Attribut *enrolled* mit einer zugewiesenen leeren Liste im Konstruktor eingefügt. Die Klasse *Enrolled* wird im Modul *enrolled* erstellt und *pass* als Anweisung eingefügt (damit Programm ausführbar bleibt). Eine Methode `add_enrollment(self, enroll)` mit Prüfung des Datentyps und

Fehlermeldung (analog zu Klasse *Student*) definiert eine Wertzuweisung zum Attribut *enrolled*. Konstruktor und Methode *add_enrollment()* der Person-Subklasse *Student* sind in Abbildung 125 sichtbar. (ebd.), 17:20

```

1 from person import Person
2 from enroll import Enroll
3
4 class Student(Person):
5     def __init__(self, first, last, dob, phone, address, international=False):
6         super().__init__(self, first, last, dob, phone, address)
7         self.international = international
8         self.enrolled = []
9
10    def add_enrollment(self, enroll):
11        if not isinstance(enroll, Enroll):
12            raise Error("Invalid Enroll...")
13
14        self.enrolled.append(enroll)

```

Abbildung 125: Konstruktor einer Subklasse *Student(Person)* mit *super()*-Konstruktor am Beispiel einer Kursbelegung an einer Schule (Tech With Tim 2020), 19:39

Der Begriff *pythonic* steht umgangssprachlich für die von Tim Peters formulierten Design-Prinzipien in Python (The Zen of Python):

„Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

Flat is better than nested.

Sparse is better than dense.

Readability counts.

Special cases aren't special enough to break the rules.

Although practicality beats purity.

Errors should never pass silently.

Unless explicitly silenced.

In the face of ambiguity, refuse the temptation to guess.

There should be one-- and preferably only one --obvious way to do it.

Although that way may not be obvious at first unless you're Dutch.

Now is better than never.

Although never is often better than *right* now.

If the implementation is hard to explain, it's a bad idea.

If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea -- let's do more of those!“ (Peters 2004)

8.2.12 Debugging, Errors und Exceptions in Python

In Python als dynamisch typisierte Programmiersprache sind Fehlermeldungen wegen falschen Datentyps möglichst zu Beginn der Programmausführung (z.B. bei Übergabe von Parametern) besonders wichtig. Ansonsten ist die Ursache eines Programm-Absturzes bei Aufruf z.B. von Attributen einer Klasse in Methoden, Funktionen oder anderen Klassen nicht einfach nachvollziehbar. (Tech With Tim 2020), 36:00 Eine weitere Absicherung gegen Bugs aufgrund falschen Datentyps stellen Typsignaturen dar. Diese sind in Python optional verfügbar, mit statischen Analysetools wie z.B. mypy (Installation mit pip möglich). Sie geben genauere Informationen über Fehler, z.B. Operationen mit ungeeigneten Datentyp(en). Ein Bug stellt dabei einen Fehler dar, der durch ungeeignete (z.B. falsch formatierte) Eingaben eines Anwenders / einer Anwenderin entsteht. (Slatkin 2020), S. 403, 484-486

Eine einfache und effektive Möglichkeit zum Debugging stellt die Standardfunktion *print()* dar. Damit ist die Überprüfung von Werten im Programm möglich (z.B. nach Übergabe von Parametern). Ein Wert einfachen Datentyps kann mit verschiedenen Syntax-Varianten (z.B. mit Formatelementen) in *print()* als String ausgegeben werden, siehe Abbildung 126 (vgl. Abbildung 57 und Abbildung 58 in Unterkapitel 8.2.3). (Slatkin 2020), S. 404-405

```
my_value = 'foo bar'
print(str(my_value))
print('%s' % my_value)
print(f'{my_value}')
print(format(my_value))
print(my_value.__format__('s'))
print(my_value.__str__())
>>>
foo bar
```

Abbildung 126: Syntax-Varianten zur Ausgabe von Werten einfachen Datentyps mit *print()* in Python (Slatkin 2020), S. 404

Die Verwendung der Standardfunktion *repr()* gibt für Strings und Zahlenwerte (Ganzzahl, Gleitpunktzahl) unterschiedliche Strings zurück. Sie ist damit zur Feststellung einfacher Datentypen geeignet, siehe Abbildung 127. (Slatkin 2020), S. 405-406

<pre>print(repr(5)) print(repr('5')) >>> 5 '5'</pre>	<pre>print('%r' % 5) print('%r' % '5')</pre>
---	--

Abbildung 127: Beispiel für Debugging mit *repr()* in Python (Slatkin 2020), S. 406

Abbildung 128 zeigt einige Anwendungen von Debugging mit `print()`. Dazu gehören Test von Funktionsaufrufen, Parameter- bzw. Variablenwerten und Bedingungen in `if`-Anweisungen (in 1. Probe). Weiterhin ist die Prüfung von Parametern in Funktionen sowie Prüfung von Schleifen (z.B. `for`-Schleife) möglich (2. Trace).

1 Probe

Add `print` statements. Use to:

- Check if a function is being called or not:

<code>def f(x, y): return x + 3*y</code>	→	<code>def f(x, y): print("HELLO FROM f") return x + 3*y</code>
--	---	--

- Check the value of a variable:

<code>y = 15 / x</code>	→	<code>print("x:", x) y = 15 / x</code>
-------------------------	---	--

- Check what happens at a conditional:

<code>if x > 5: y = 10 else: y = 3</code>	→	<code>if x > 5: print("x > 5") y = 10 else: print("x <= 5") y = 3</code>
--	---	---

2 Trace

Use multiple **probes** to understand code. Use to:

- Figure out where a value comes from:

<code>def f(a): g(a * 3) def g(b): for i in range(b): h(9-i) def h(c): print(10/c)</code>	→	<code>def f(a): print("a:", a) g(a * 3) def g(b): print("b:", b) for i in range(b): print("i:", i) h(9-i) def h(c): print("c:", c) print(10/c)</code>
---	---	---

(error if `c` is 0 in function `h`)

Abbildung 128: Übersicht Debugging mit `print()` in Python (Wellesley 2022)

Abbildung 129 zeigt weitere Debugging-Techniken, z.B. für komplexe mathematische Ausdrücke. Durch Zuweisung von Teilausdrücken zu mehreren Variablen ist eine genaue Lokalisierung eines Fehlers möglich (3. Unpack).

Weiterhin ist eine temporäre Vereinfachung von Quellcode möglich, um Fehlerquellen einzukreisen (4. Toggle und 5. Bisect). Das geschieht mit Auskommentieren (z.B. von optionalen Anweisungen oder Teilen einer Liste) und dem Einsatz von dummy values.

3 Unpack

Split up a complicated expression into multiple statements. Use this to:

- Isolate an error in a complex expression:

```
x = function(
    (a + 3*b)/(c * d),
    b / a
)
```

(ZeroDivisionError on line 1)

```
top = a + 3*b
bot = c * d
fst = top / bot
sec = b / a
x = function(fst, sec)
```

(ZeroDivisionError on line 4, so
a must be the problem)

4 Toggle

Turn a line of code into a comment. Use to:

- Disable (can later re-enable) optional code:

```
def f(a, b):
    print("R: ", a/b)
    return a + b + a
```

```
def f(a, b):
    #print("R: ", a/b)
    return a + b + a
```

- Temporarily replace broken code with a dummy value:

```
x = (3*y + 4*z)/w
```

```
#x = (3*y + 4*z)/w
x = 9
```

5 Bisect

Comment out half of your code to find the half that works, and then half of the broken part, etc., until you isolate an error. Use this to:

- Find missing brackets or commas:

```
pairs = [
    [0, 1],
    [10, 11],
    [20, 21],
    [30, 31],
]
```

(syntax error at end of file)

```
pairs = [
    # [0, 1],
    # [10, 11],
    [20, 21],
    [30, 31],
]
```

(works now, so error must be in
the commented zone)

Abbildung 129: Übersicht Debugging durch Aufteilung und Auskommentierung von Ausdrücken in Python (Wellesley 2022)

Ein weiteres Hilfsmittel ist Pythons integrierter interaktiver Debugger. In der IDE PyCharm Community 2022 ist z.B. unter Run ein Debugger enthalten. Eine Programmausführung erfolgt somit mit oder ohne Debugger. Ein Einfügen von breakpoints ist entweder in der IDE möglich oder durch Einfügen einer Anweisung `breakpoint()` an der gewünschten Stelle im Quellcode. Bei Programmausführung ist bei Unterbrechung am breakpoint eine Python-Shell verfügbar. Diese ermöglicht die Ausgabe von Werten (z.B. von lokalen Variablen) zum Zeitpunkt der Unterbrechung. (Slatkin 2020), S. 432-435

Ein Fehler (*error*) ist ein Programmabbruch in der Laufzeit. Er tritt beim Kompilieren (compile-time error, syntax error) oder beim Interpretieren (run-time error, exception) auf. Ein syntax error umfasst Fehler in den Regeln und Strukturen einer Programmiersprache. Dazu gehört z.B. Zeichensetzung. Exceptions treten bei nicht ausführbaren Operationen auf. Im Python-Interpreter sind viele Standard-*Error* (built-in exceptions) enthalten. Diese sind Subklassen von *Exception*. Wenn eine exception vom Interpreter nicht unter einen anderen *Error*-Typ fällt, gibt Python *RuntimeError* aus. Einige exceptions in Python sind z.B.

AssertionError – Ein assert statement `assert(Bedingung, Fehlerbeschreibung)` gibt den Wert *False* zurück. Optional wird eine Fehlerbeschreibung ausgegeben im *AssertionError*. (TechVidvan 2022a)

AttributeError – Eine Zuweisung oder ein Aufruf eines Attributs funktioniert nicht.

EOFError – Eine Eingabefunktion (z.B. `input()`) erreicht End Of File (EOF).

ImportError – Ein importiertes Modul ist nicht vorhanden.

IndexError – Eine Instanz eines sequentiellen Datentyps enthält kein Element mit aufgerufenem Index.

NameError – Eine Variable ist nicht im Namensraum vorhanden.

TypeError – Eine Operation oder Funktion erhält eine Instanz eines nicht geeigneten Datentyps.

ValueError – Ein Wert eines Parameters einer Funktion hat keinen akzeptierten Wert.

(TechVidvan 2022)

In Python ist die Erstellung eigener exceptions möglich. Der erste Schritt ist die Erzeugung einer neuen Subklasse mit *Exception* oder einer anderen exception-Klasse als Überklasse. Ein Beispiel für eine exception bei zu niedrigem Wert der Variable *temp* mit Fehlerbeschreibung liefert Abbildung 130.

```

1.  >>> class InvalidTemperature(Exception):
2.      def __init__(self, temp):
3.          self.temp=temp
4.      def __str__(self):
5.          return f'The temperature {self.temp} is low'

```

Abbildung 130: Beispiel für neue *Exception*-Subklasse in Python (TechVidvan 2022a)

Bei Auslösung einer exception (z.B. mit Schlüsselwort *raise*) gibt die Python-Shell eine Fehlermeldung aus. Für das Beispiel sieht diese wie in Abbildung 131 aus.

```

1.  >>> raise InvalidTemperature(30)
Traceback (most recent call last):
  File "<pyshell#283>", line 1, in <module>
    raise InvalidTemperature(30)
InvalidTemperature: The temperature 30 is low

```

Abbildung 131: Beispiel für Fehlermeldung im Fall einer selbsterstellten exception in Python (TechVidvan 2022a)

Eine einfache Möglichkeit für eine exception in Python ist mit *try / except* – Anweisungen, siehe Abbildung 132. Mithilfe des Moduls *sys* ist neben der *print()*-Ausgabe eines selbstgeschriebenen Fehlertextes auch eine Ausgabe des exception-Typs möglich.

```

1.  >>> import sys
2.  >>> try:
3.      nums[3]
4.  except:
5.      print('Out of index!')
6.      print(sys.exc_info())

```

```

Out of index!
(<class 'IndexError>, IndexError('list index out of
range'), <traceback object at 0x00000289E33498C8>)

```

Abbildung 132: Beispiel für *try / except* – Anweisung mit selbstdefinierter Fehlermeldung im Fall einer exception in Python (TechVidvan 2022a)

Mehrere *except* – Anweisungsblöcke sind möglich. Jedem *except* folgt ein / mehrere exception-Typ(en) (als Tuple). Die letzte *except* – Anweisung ist optional als generische Variante ohne exception-Typ möglich, siehe Abbildung 133.

```
>>> try:
    pass
except ZeroDivisionError:
    pass
except (IndexError, NameError):
    pass
except:
    pass
```

Abbildung 133: Beispiel für `try / except` – Anweisungen mit mehreren `except`-Anweisungsblöcken in Python (TechVidvan 2022a)

`try / except` – Anweisungen sind mit einer `finally` – Anweisung erweiterbar. Im Falle einer exception ist auf diese Weise die Ausführung von Anweisungen möglich, z.B. das Schließen einer Datei, siehe Abbildung 134.

```
1. >>> try:
2.     f=open('text.txt')
3.     f.write('Hello')
4. except:
5.     pass
6. finally:
7.     f.close()
```

Abbildung 134: Beispiel für `try / except` – Anweisungen mit `finally`-Anweisung in Python (TechVidvan 2022a)

Mit dem Schlüsselwort `raise` ist eine erzwungene exception möglich, wie bereits Abbildung 131 zeigt. Einer exception kann dabei ein Parameter mit zusätzlichem Ausgabetext übergeben werden, z.B. `TypeError('Das ist nicht der richtige Datentyp.')`.

`try / except` – Anweisungen sind auch mit einer `else`-Anweisung erweiterbar. Diese wird ausgeführt, wenn keine exception auftritt, siehe Abbildung 135.

<pre>1. >>> try: 2. print(1) 3. except: 4. print(2) 5. else: 6. print(3)</pre>	Output: <div style="border: 1px solid black; padding: 5px; width: fit-content; margin-left: 20px;">1 3</div>
---	--

Abbildung 135: Beispiel für `try / except` – Anweisungen mit anschließender `else`-Anweisung in Python (TechVidvan 2022a)

(TechVidvan 2022a)

8.3 Anhänge zu Material und Methoden

8.3.1 Berechnungsablauf „QUEBER_Version2016“

Nachfolgend ist die Berechnung vom Matlab MuPad Notebook „QUEBER_Version2016“ in Form eines (aufgeteilt dargestellten) Programmablaufplans (PAP, Flussdiagramm, flowchart) beschrieben. Zunächst folgt eine Legende zur Verständlichkeit der Symbole, siehe Abbildung 136. Diese sind teilweise erweitert bzw. modifiziert für eine eindeutige Darstellung. Der Programmablauf findet dabei von oben nach unten sowie von links nach rechts statt. (Wikipedia 2022c)

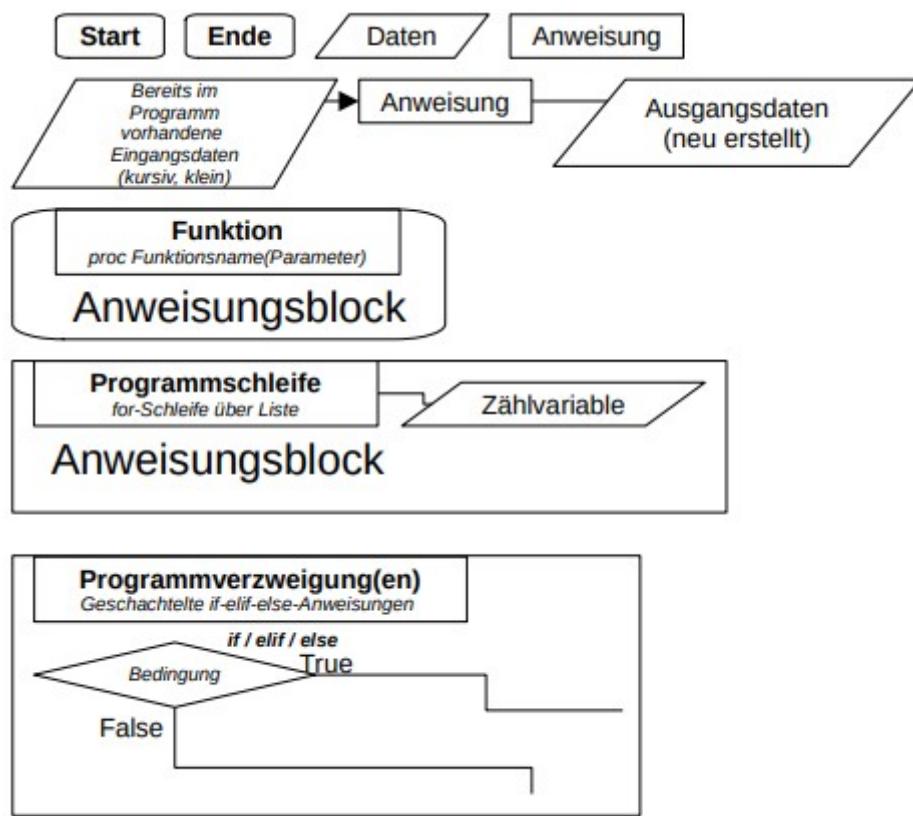


Abbildung 136: Legende mit Bedeutung von Symbolen in den Flowcharts für „QUEBER_Version2016“, vgl. (Wikipedia 2022c)

Abbildung 137 zeigt als Flowchart den Beginn des Programms mit Öffnen von *Eingabefile* und *Ausgabefile*. Danach folgt die Erstellung von Variablen wie *GMOD*, *NK* und *NM* und Vektoren bzw. Matrizen wie *KNOKO*, *STAB*, *STAB_OFF*, *DICKE* und *LANGE*. *KNOKO*, *STAB* und *DICKE* erhalten Eingabedaten zugewiesen. (Stanoev 2016)

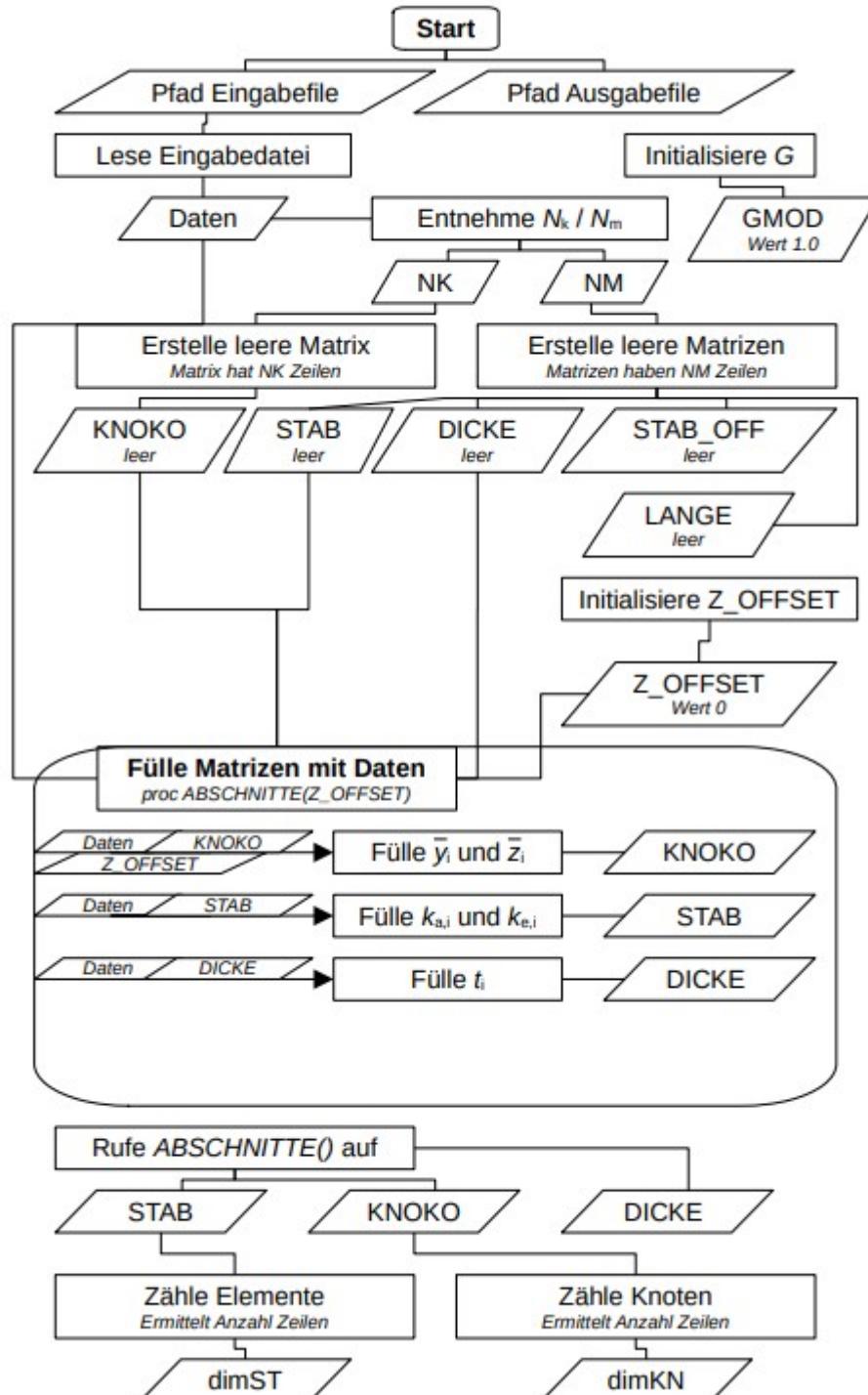


Abbildung 137: Flowchart zum „Vorbereiten der Eingabedaten“ (Lesen aus Textdatei und Sortieren) in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Abbildung 138 beinhaltet den Beginn der „Berechnung der Querschnittswerte“ mit Flächenmomenten, FSP S (Ys, Zs) und LANGE aus Unterkapitel 2.1.2. (ebd.)

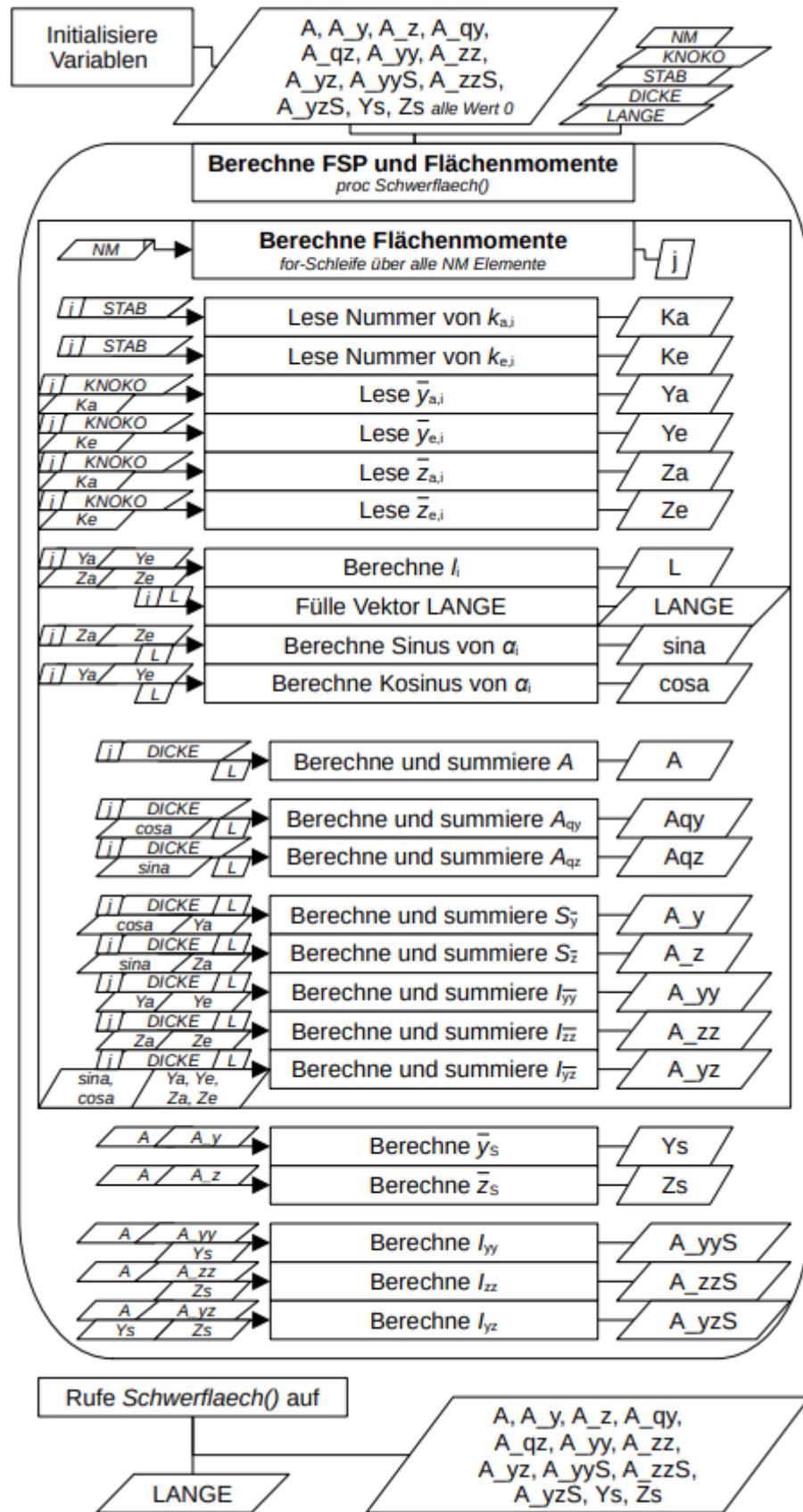


Abbildung 138: Flowchart zur Berechnung von grundlegenden Flächenmomenten und FSP in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Nach Unterkapitel 2.1.3 erfolgt die „Berechnung des Verdrehwinkels und der Hauptträgheitsachsen und Hauptträgheitsradien“, siehe Abbildung 139. (ebd.)

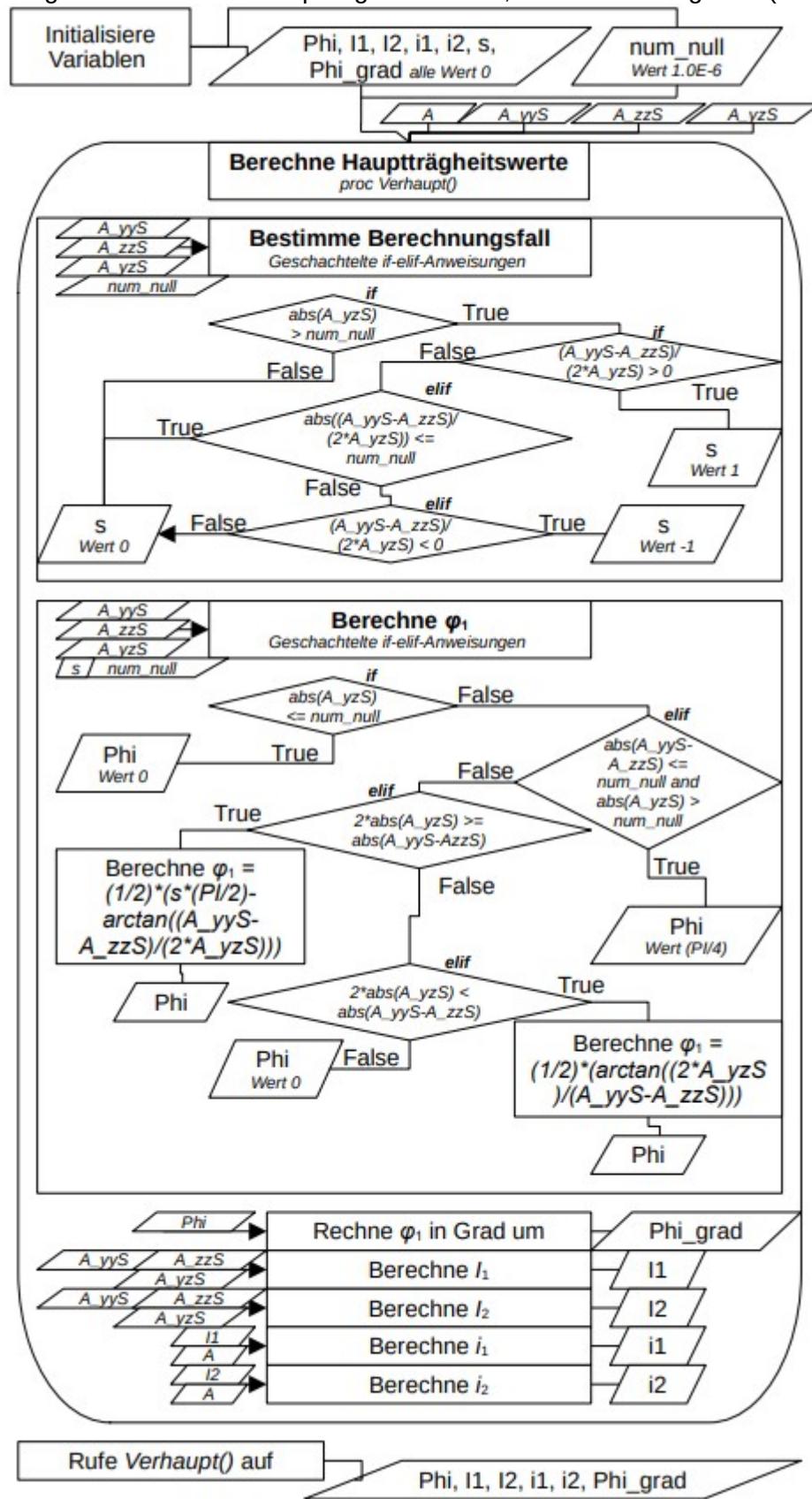


Abbildung 139: Flowchart Berechnung Verdrehwinkel und Hauptträgheitsachsen bzw. -radien in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Daran schließt sich die „Berechnung Polstrahl zum finiten Element“ RT bezogen auf O nach Gleichung 2.63 aus Unterkapitel 2.3.2 an, siehe Abbildung 140. (ebd.)

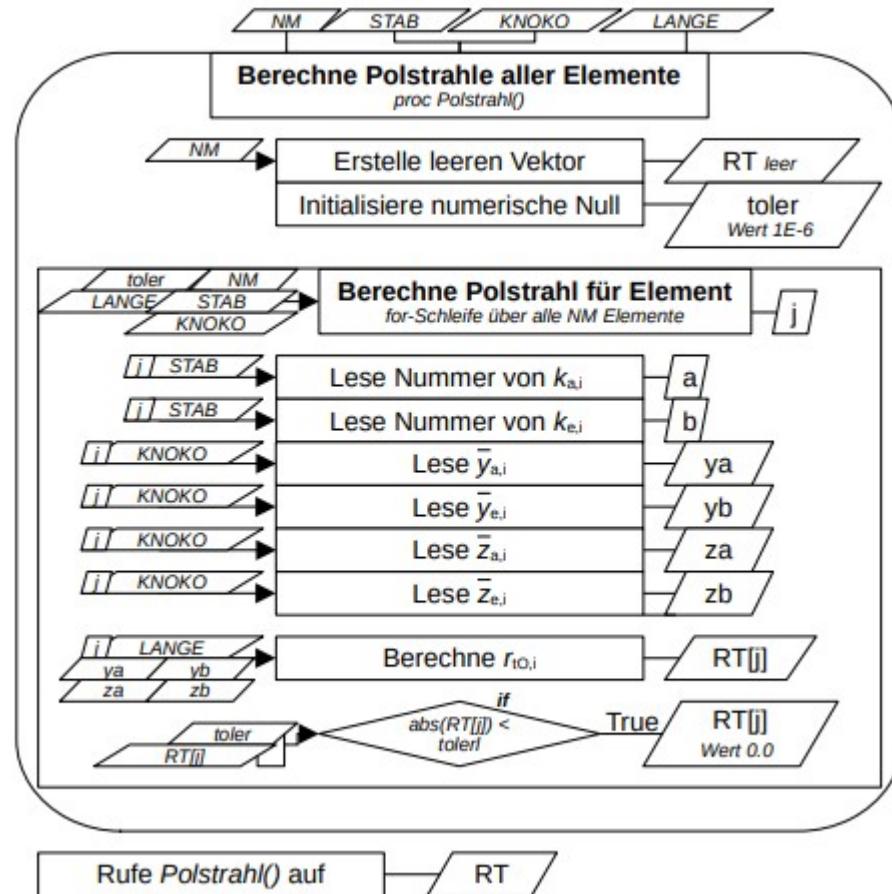


Abbildung 140: Flowchart zur Berechnung der Polstrahle aller Elemente in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Abbildung 141 zeigt den Beginn der „Prozedur zur Assemblierung der Systemsteifigkeitsmatrix/ Gesamtsteifigkeismatrix SS und des ‚rechte Seite‘-Vektors B“ im Programm. Zunächst wird der Systemlastvektor B aus den Elementlastvektoren PJ nach Gleichung 2.82 aus Unterkapitel 2.3.2 berechnet. (ebd.)

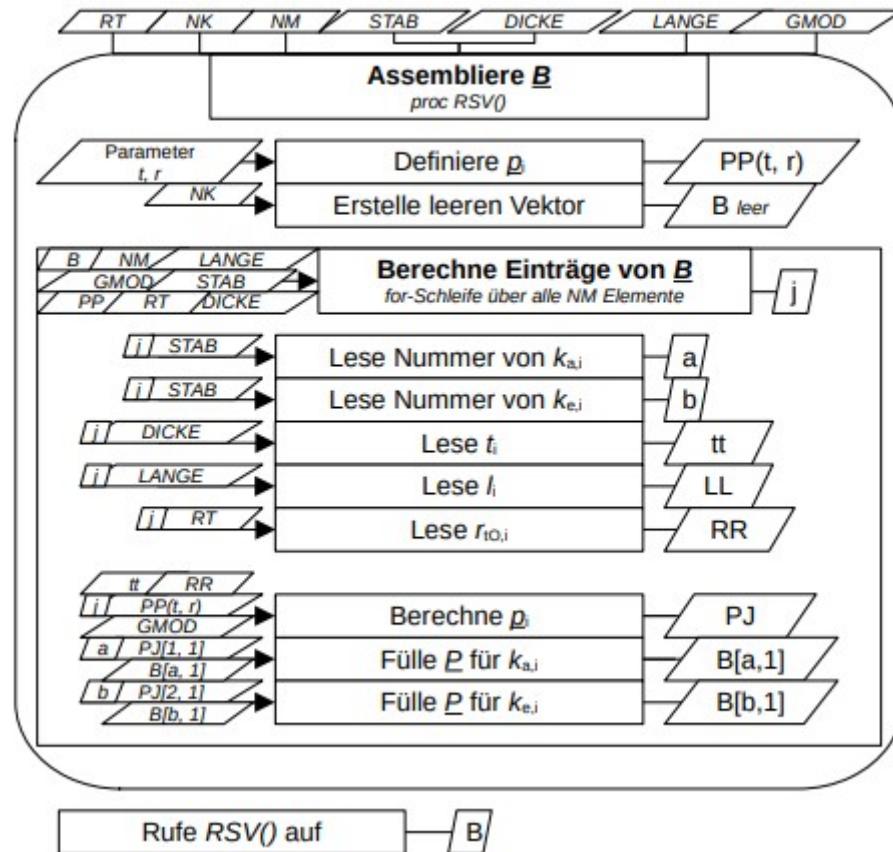


Abbildung 141: Flowchart zur Berechnung der Elementlastvektoren und des Systemlastvektors in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Abbildung 142 beinhaltet die Berechnung von Elementsteifigkeitsmatrizen KJ und deren Zusammenfassung zur Systemsteifigkeitsmatrix SS . Mit der daraus gebildeten reduzierten Matrix AA und dem reduzierten Lastvektor BB ist die Berechnung des Vektors $OMEG$ nach Gleichung 2.85 aus Unterkapitel 2.3.2 möglich. (ebd.)

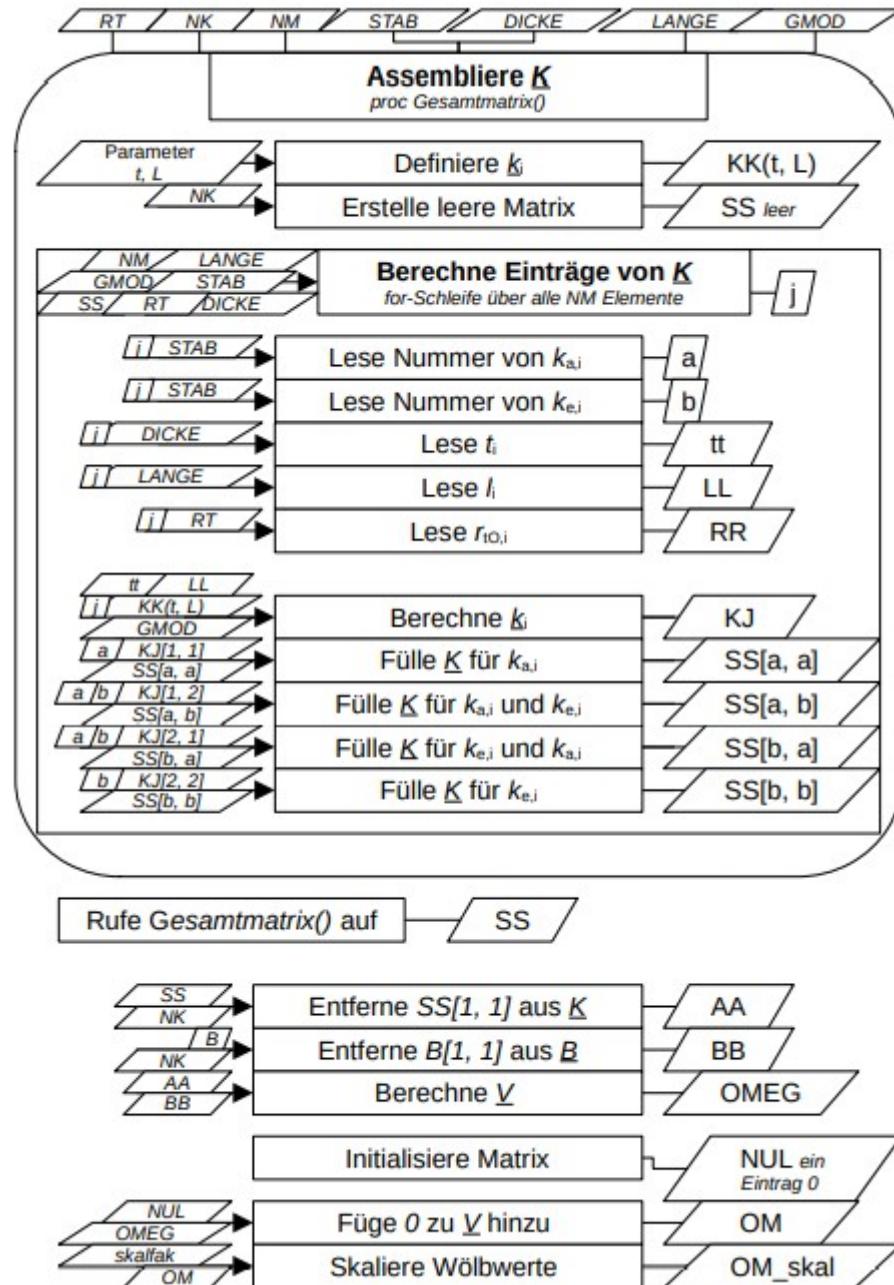


Abbildung 142: Flowchart zur Berechnung der Elementlastvektoren und des Systemlastvektors in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Die „Berechnung des St. Venantschen Torsionsträgheitsmomentes I_T “ erfolgt wie in Unterkapitel 2.3.6 beschrieben, siehe Abbildung 143. (ebd.)

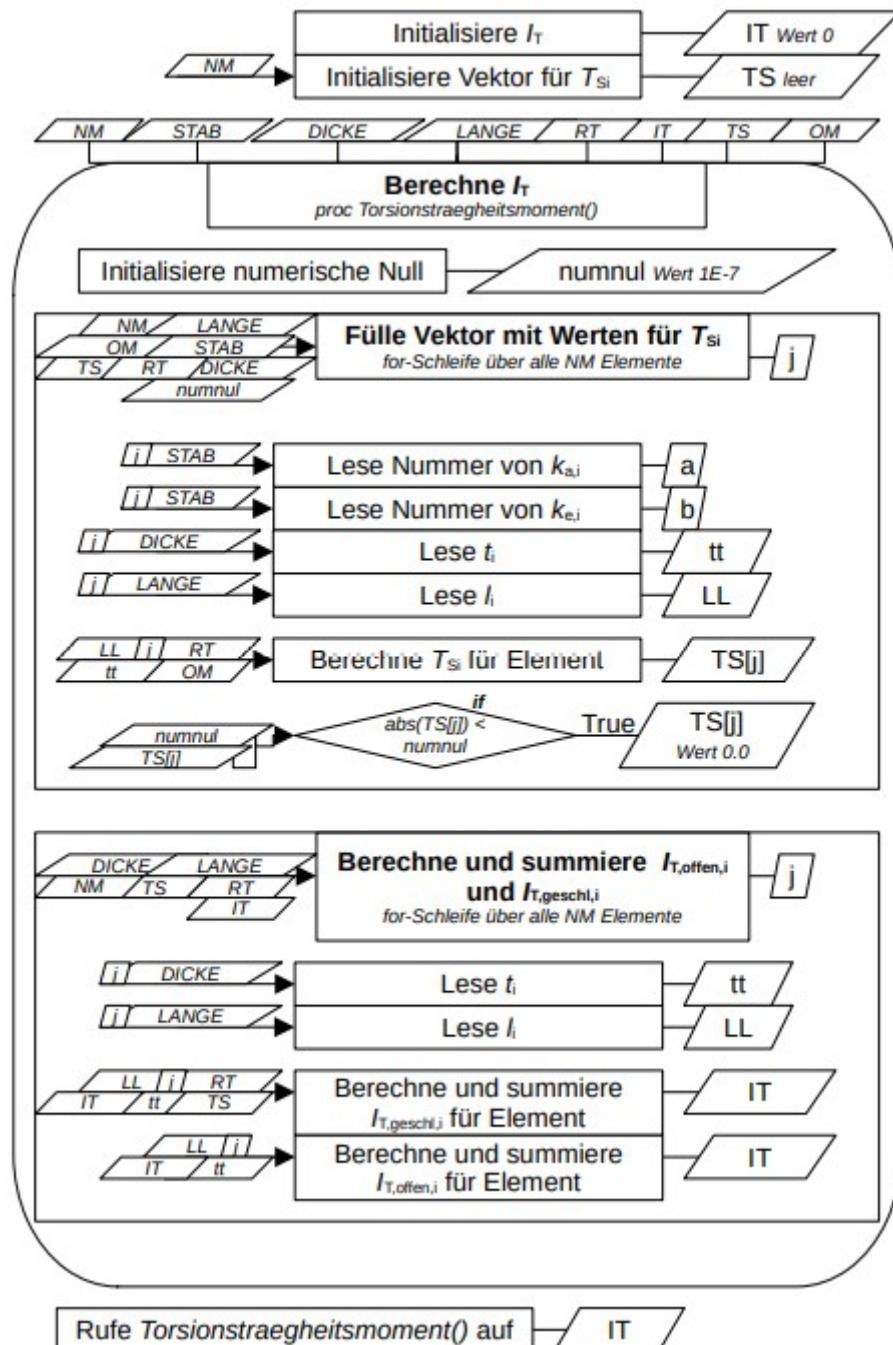


Abbildung 143: Flowchart zur Berechnung des Torsionsträgheitsmoments aus offenen und geschlossenen Anteilen in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Abbildung 144 zeigt die „Berechnung der Normierungskonstante OM_0“ und der „Woelbkoordinaten bezogen auf den Schwerpunkt S“. Gleichung 2.102 und Gleichung 2.103 aus Unterkapitel 2.3.5 beschreiben diese. (ebd.)

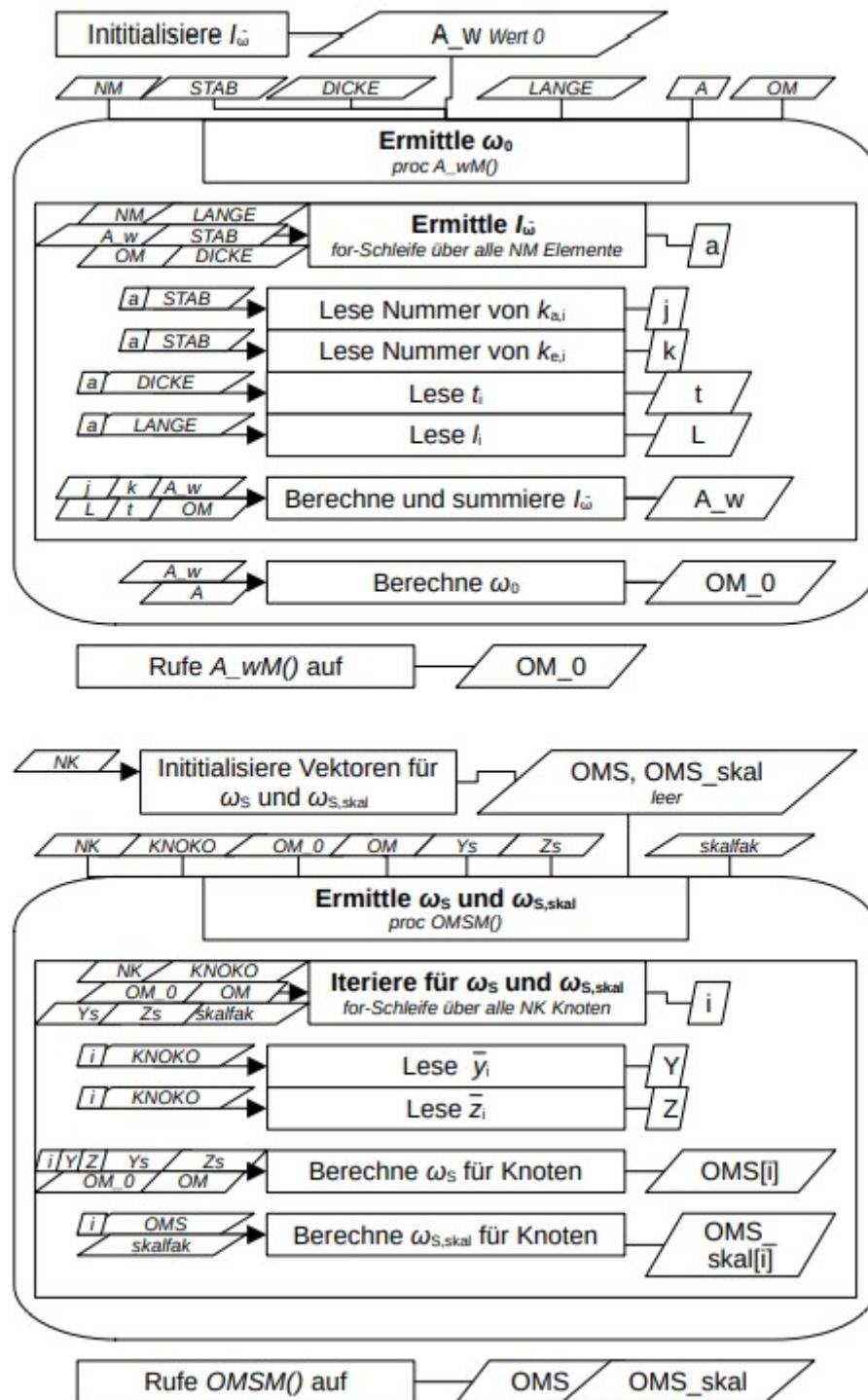


Abbildung 144: Flowchart zur Berechnung von Normierungskonstante und Wölfunktion bezogen auf S in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Die „Berechnung Schubmittelpunkt-Koordinaten“ erfolgt nach den Unterkapiteln 2.3.3 und 2.3.4, siehe Abbildung 145. (ebd.)

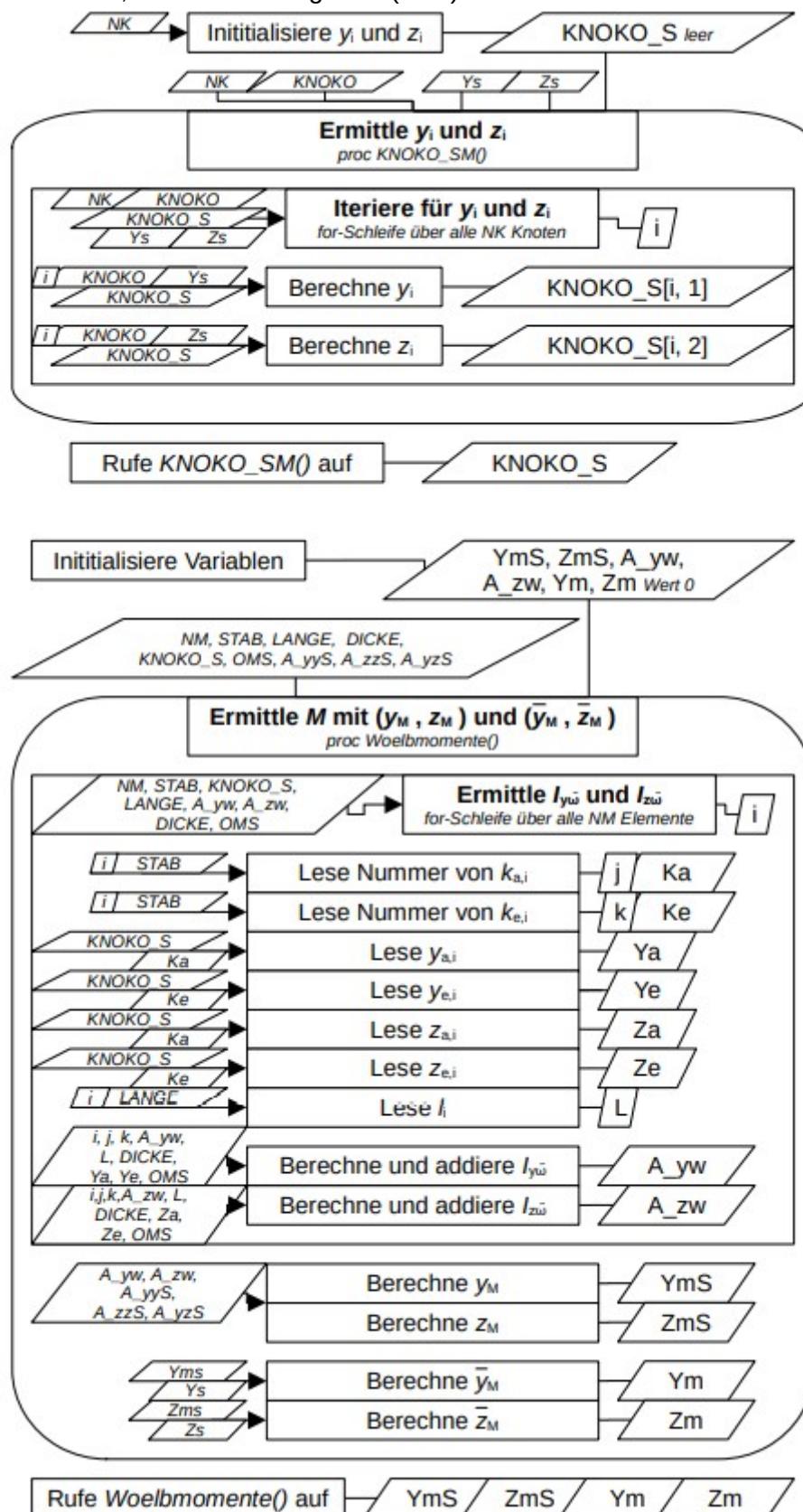


Abbildung 145: Flowchart zur Berechnung des SMP M in „QUEBER_Version2016“, vgl. (Stanoev 2016)

Abbildung 146 zeigt „Wölbkoordinaten bezogen auf den SM-Punkt M“ und „Wölbträgheitsmoment berechnen A_omomM“. Grundlage sind Gleichung 2.104 und Gleichung 2.106 aus Unterkapitel 2.3.5. (ebd.)

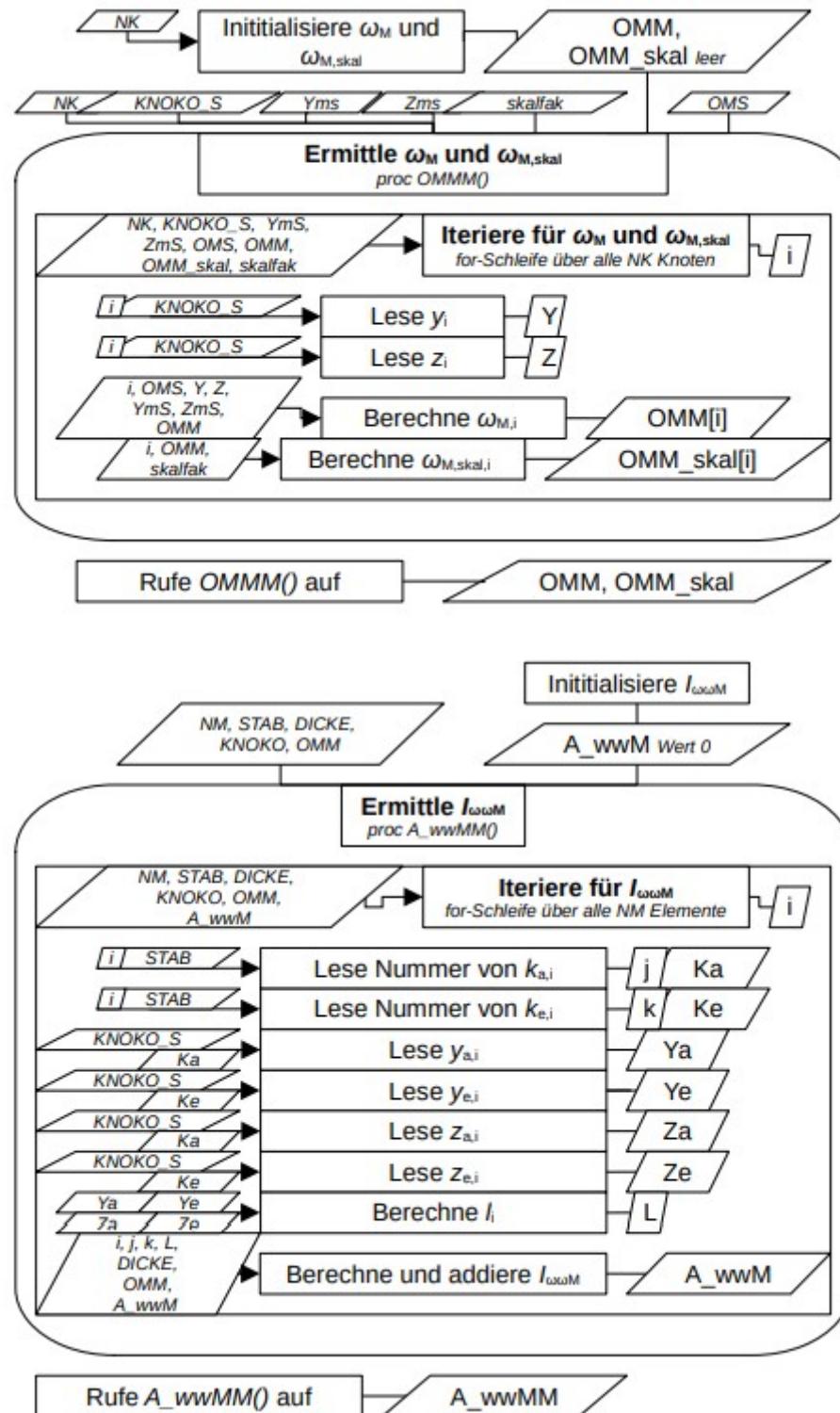


Abbildung 146: Flowchart zur Berechnung der Wölbfunktion bezogen auf M und des Wölbträgheitsmoments in „QUEBER_Version2016“, vgl. (Stanoev 2016)

8.3.2 Ausgabedatei von „QUEBER_Version2016“

Das nachfolgende Beispiel verwendet die Eingabedatei (Wunderlich k.A.).

„Universität Rostock : Fakultät für Maschinenbau und Schiffstechnik

Stiftungslehrstuhl für Windenergietechnik

COMPUTERPROGRAMM

Geometrische Werte dreieckiger Querschnitte

Version Dez. 2016

Autoren: Evgeni Stanoev / Sebastian Schwarz / Zhengkun Liu / Nan Li

Benutzer : LV_DwSS_Woelbfunktion

Profilbezeichnung : Profil_Wunderlich

Datum : DATUM

Uhrzeit : UHRZEIT Uhr

EINGABEDATEN

Anzahl Knoten : Abschnitte

--> 9 : 8

Knotenkoordinaten

Nr. Y Z

1 : -6.0 0

2 : 0 0

3 : 12.0 0

4 : 20.0 6.0

5 : 0 30.0

6 : -6.0 30.0

7 : 10.0 30.0

8 : 10.0 35.0

9 : 10.0 20.0

Polygonabschnitte

Nr. : A - E : Dicke

1 : 1 - 2 : 1.2
 2 : 2 - 3 : 1.6
 3 : 3 - 4 : 1.6
 4 : 2 - 5 : 1.2
 5 : 5 - 6 : 1.6
 6 : 5 - 7 : 1.4
 7 : 7 - 8 : 1.4
 8 : 7 - 9 : 1.4

QUERSCHNITTSWERTE

Werte im Ausgangskoordinatensystem

Koordinaten

geometrischer Schwerpunkt [m] Ys = 4.88455
 [m] Zs = 15.2317

Flächeninhalt [cm²] A = 123.0
 Schubflächenmoment Aqy [cm²] Aqy = 62.8
 Schubflächenmoment Aqz [cm²] Aqz = 66.6
 Statisches Moment [cm³] A_y = 600.8
 Statisches Moment [cm³] A_z = 1873.5
 Trägheitsmoment [cm⁴] A_yy = 7871.2
 Trägheitsmoment [cm⁴] A_zz = 48507.0
 Deviationsmoment [cm⁴] A_yz = 7843.0

Werte im Schwerpunktkoordinatensystem

Trägheitsmoment [cm⁴] A_yyS = 4936.56
 Trägheitsmoment [cm⁴] A_zzS = 19970.4
 Deviationsmoment [cm⁴] A_yzS = -1308.21

Verdrehwinkel [Grad] Phi = 4.93631
 Hauptträgheitsmoment1 [cm⁴] I1 = 20083.4
 Hauptträgheitsmoment2 [cm⁴] I2 = 4823.57

Haupttr#gheitsradius1 [cm] i1 = 12.7781

Haupttr#gheitsradius2 [cm] i2 = 6.26227

Torsiontr#gheitsmoment[cm^4] IT = 81.832

Koordinaten

Schubmittelpunkt [cm] Ym = -5.85488

bezogen auf Ausgangs-KS [cm] Zm = 9.48444

Koordinaten

Schubmittelpunkt [cm] YmS = -10.7394

bezogen auf Schwerpunkt-KS [cm] ZmS = -5.74726

W#lbtr#gheitsmoment [cm^6] A_wwM = 1369980.0

W#lbfunktion Omega

Knoten-Nr : Omega_Ursprung [cm^2]

1 : 0
2 : 6.21725e-14
3 : 9.14824e-14
4 : 72.0
5 : 3.48166e-13
6 : 180.0
7 : -300.0
8 : -250.0
9 : -400.0

Knoten-Nr : Omega_Schwerpkt [cm^2]

1 : -30.5366
2 : 60.8537
3 : 243.634
4 : 408.18
5 : -85.6829
6 : 2.92683

7 : -233.366
8 : -207.789
9 : -284.52

Knoten-Nr : Omega_Schubmittelpkt [cm²]

1 : -131.56
2 : -74.6535
3 : 39.1599
4 : 222.165
5 : 100.993
6 : 224.086
7 : -104.163
8 : -24.8881
9 : -262.711"

(Stanoev 2016) / (Wunderlich k.A.)

8.3.3 Quellcode der Programmerweiterung in C#

Im folgenden befindet sich der neue Quellcode als Erweiterung zu (Vent 2022), „Thesis_Interface“, QuerschnittGraph2D.

```
...neueVariante\Thesis_Interface\QuerschnittGraph2D.cs
214     // Darstellen der Wölfunktion bezogen auf Schubmittelpunkt
215     // @author Heinrich Vent, Masterarbeit 2022
216     private void Wölfunktion()
217     {
218         // Koordinaten des Anfangsknotens
219         double y_knoten_anfang;
220         double z_knoten_anfang;
221         // Wert der Wölfunktion am Anfangsknoten des Stabelements
222         i
223         double omm_stab_anfang;
224         // Koordinaten des Endknotens
225         double y_knoten_ende;
226         double z_knoten_ende;
227         // Wert der Wölfunktion am Endknoten des Stabelements i
228         double omm_stab_ende;
229         // Werte der Wölfunktion in darstellbarer Größenordnung
230         double SKALIERUNG_OMM = 1;
231         double omm_anfang_skaliert;
232         double omm_ende_skaliert;
233         // Differenz zwischen Koordinaten von Anfangs- und
234         // Endknoten
235         double dy_stab;
236         double dz_stab;
237         // Koordinaten der Wölfunktion an Anfangs- und Endknoten
238         double omm_y_anfang;
239         double omm_z_anfang;
240         double omm_y_ende;
241         double omm_z_ende;
242
243         double sina;
244         double cosa;
245         int umlauf;
246         double L;
247         vector Woelbwerte = new vector (2);
248
249         // Ermittle Vektor der Wölfunktion im Graph an Anfangs-/
250         // Endknoten
251         vector OM_VEC(double omm, double y_knoten, double z_knoten)
252         {
253             L = Math.Sqrt(Math.Pow(dy_stab, 2) + Math.Pow(dz_stab,
254                 2));
255             sina = dz_stab / L;
256             cosa = dy_stab / L;
257             double omm_dy = omm * sina;
258             double omm_dz = omm * cosa;
259             umlauf = 1;
260             if ((y_knoten_anfang*z_knoten_ende
261                 - y_knoten_ende*z_knoten_anfang) < 0)
262             {
263                 umlauf = -1;
264             }
265             Woelbwerte[0] = y_knoten + umlauf*omm_dy;
266             Woelbwerte[1] = z_knoten - umlauf*omm_dz;
267         }
268     }
```

```

...neueVariante\Thesis_Interface\QuerschnittGraph2D.cs
263         return Woelbwerte;
264     }
265
266     // Ermittle einen Punkt der Wölfunktion in Kette der OMM-
267     // Darstellung
268     // (Knotenreihenfolge: Anfangsknoten + OMM_Anfang +
269     // OMM_Ende)
270     int Knotennummer_Extremwert(int node)
271     {
272         int point_number = 3 * node + 1;
273         return point_number;
274     }
275
276     for (int i = 0; i < cross_section.nk; i++)
277     {
278
279         // Zuordnen der Koordinaten der Knotenpunkte
280         // vom Stabelement i
281         //-----
282         // Koordinaten des Anfangsknotens
283         y_knoten_anfang = cross_section.DATA1[i, 0];
284         z_knoten_anfang = cross_section.DATA1[i, 1];
285         // Wert der Wölfunktion am Anfangsknoten des
286         // Stabelements i
287         omm_stab_anfang = cross_section.OMM[i, 0];
288         // If-Anweisung für alle nk-1 Stabelemente, um
289         // Endknoten-
290         // Koordinaten
291         if (i + 1 < cross_section.nk)
292         {
293             y_knoten_ende = cross_section.DATA1[i + 1, 0];
294             z_knoten_ende = cross_section.DATA1[i + 1, 1];
295             omm_stab_ende = cross_section.OMM[i + 1, 0];
296         }
297         // else-Anweisung für die Wölfunktion am Endknoten des
298         // letzten Stabelement
299         else
300         {
301             y_knoten_ende = cross_section.DATA1[0, 0];
302             z_knoten_ende = cross_section.DATA1[0, 1];
303             omm_stab_ende = cross_section.OMM[0, 0];
304
305         }
306
307         // Berechnen der Koordinaten der Punkte der
308         // Wölfunktion
309         //
310         //-----
311         omm_anfang_skaliert = omm_stab_anfang * SKALIERUNG_OMM;
312         omm_ende_skaliert = omm_stab_ende * SKALIERUNG_OMM;
313
314         // Berechnung des Abstands von Anfangs- und Endknoten
315
316     }

```

```

...neueVariante\Thesis_Interface\QuerschnittGraph2D.cs
308          // in y- und z-Richtung
309          dy_stab = y_knoten_ende - y_knoten_anfang;
310          dz_stab = z_knoten_ende - z_knoten_anfang;

311          // Berechnung der Koordinaten der Wölfunktion
312          // am Anfangsknoten im Graphen
313          Woelbwerte = OM_VEC(omm_anfang_skaliert,
314          y_knoten_anfang,
315          z_knoten_anfang);
316          omm_y_anfang = Woelbwerte[0];
317          omm_z_anfang = Woelbwerte[1];

318          // Berechne Koordinaten der Wölfunktion am
319          // Anfangsknoten
320          Woelbwerte = OM_VEC(omm_ende_skaliert, y_knoten_ende,
321          z_knoten_ende);
322          omm_y_ende = Woelbwerte[0];
323          omm_z_ende = Woelbwerte[1];

324          // Einfügen der Punkte von Querschnitt und Wölfunktion
325          // in die dafür erstellte Serie
326          graphBereich.Series["Wölfunktion"].Points.
327          AddXY(y_knoten_anfang, z_knoten_anfang);
328          graphBereich.Series["Wölfunktion"].Points.
329          AddXY(omm_y_anfang, omm_z_anfang);
330          graphBereich.Series["Wölfunktion"].Points.
331          AddXY(omm_y_ende, omm_z_ende);
332      }
333
334      // Einfügen des ersten als letzten Punkt
335      graphBereich.Series["Wölfunktion"].Points.AddXY(
336          cross_section.DATA1[0, 0], cross_section.DATA1[0, 1]);
337
338      // Beschrifte Maximum und Minimum der Wölfunktion
339      // Ermitteln des Maximums und Minimums
340      double omm_max = Double.NegativeInfinity;
341      double omm_min = Double.PositiveInfinity;
342      int node_omm_max = 0;
343      int node_omm_min = 0;

344      for (int i = 0; i < cross_section.nk; i++)
345      {
346          if (cross_section.OMM[i, 0] > omm_max)
347          {
348              omm_max = cross_section.OMM[i, 0];
349              node_omm_max = i;
350          }
351          if (cross_section.OMM[i, 0] < omm_min)
352          {
353              omm_min = cross_section.OMM[i, 0];
354              node_omm_min = i;
355          }
356      }
357

```

```
...neueVariante\Thesis_Interface\QuerschnittGraph2D.cs
358         }
359
360         // Eintragen der Werte in die Graphik
361         int point_number_max = Knotennummer_Extremwert
362             (node_omm_max);
363         int point_number_min = Knotennummer_Extremwert
364             (node_omm_min);
365         string omm_max_string = Convert.ToString(Math.Round
366             (omm_max));
367         string omm_min_string = Convert.ToString(Math.Round
368             (omm_min));
369     }
370 }
```

9. Eidesstattliche Versicherung

Ich versichere eidesstattlich durch eigenhändige Unterschrift, dass ich die Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen sind, habe ich durch Angaben der Herkunft kenntlich gemacht. Dies gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet.

Ich erkläre ferner, dass ich die vorliegende Arbeit in keinem anderen Prüfungsverfahren als Prüfungsarbeit eingereicht habe oder einreichen werde.

Die eingereichte schriftliche Fassung ist identisch mit der elektronisch eingereichten Fassung.

Ich weiß, dass bei Abgabe einer falschen Versicherung die Prüfung als nicht bestanden zu gelten hat.

Rostock, den 21.11.2022

(Unterschrift)