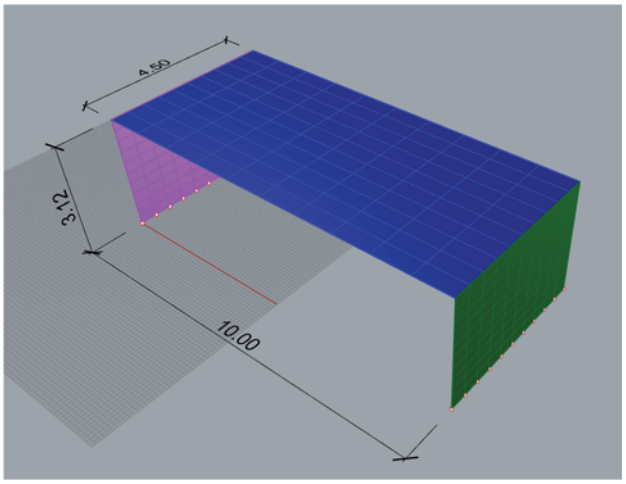
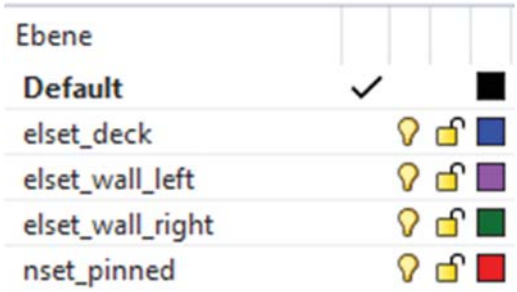
Anhang B: Tutorial Rahmen

An dieser Stelle wird anhand des Beispiels aus Kapitel 4.3.4 Schritt für Schritt gezeigt, wie die Eingabe für die Berechnung mit Compas FEA, Abaqus und dem Sandwichmodell erfolgt.

1. Modellbildung in Rhino

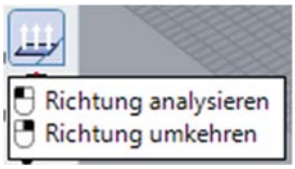
Als erstes muss die gewünschte Struktur in Rhino modelliert werden. Dafür werden Polygonnetze gezeichnet, die je einer separaten Ebene zugeordnet werden. Über den Namen dieser Ebene wird das Element später von Compas FEA angesteuert. **Gezeichnet wird in Millimeter**. Es ist immer darauf zu achten, dass die Schnittpunkte zweier sich berührender Elemente immer zu **100% exakt übereinanderliegen**. An den Schnittpunkten, wo eine Last oder ein Auflager sein soll, werden Punkte in einer neuen separaten Ebene gesetzt. Die Vermassung ist optional.

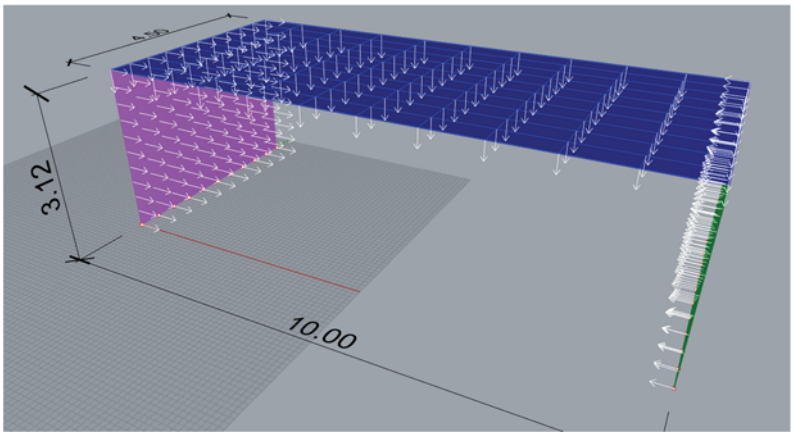




2. Die z-Achse definieren

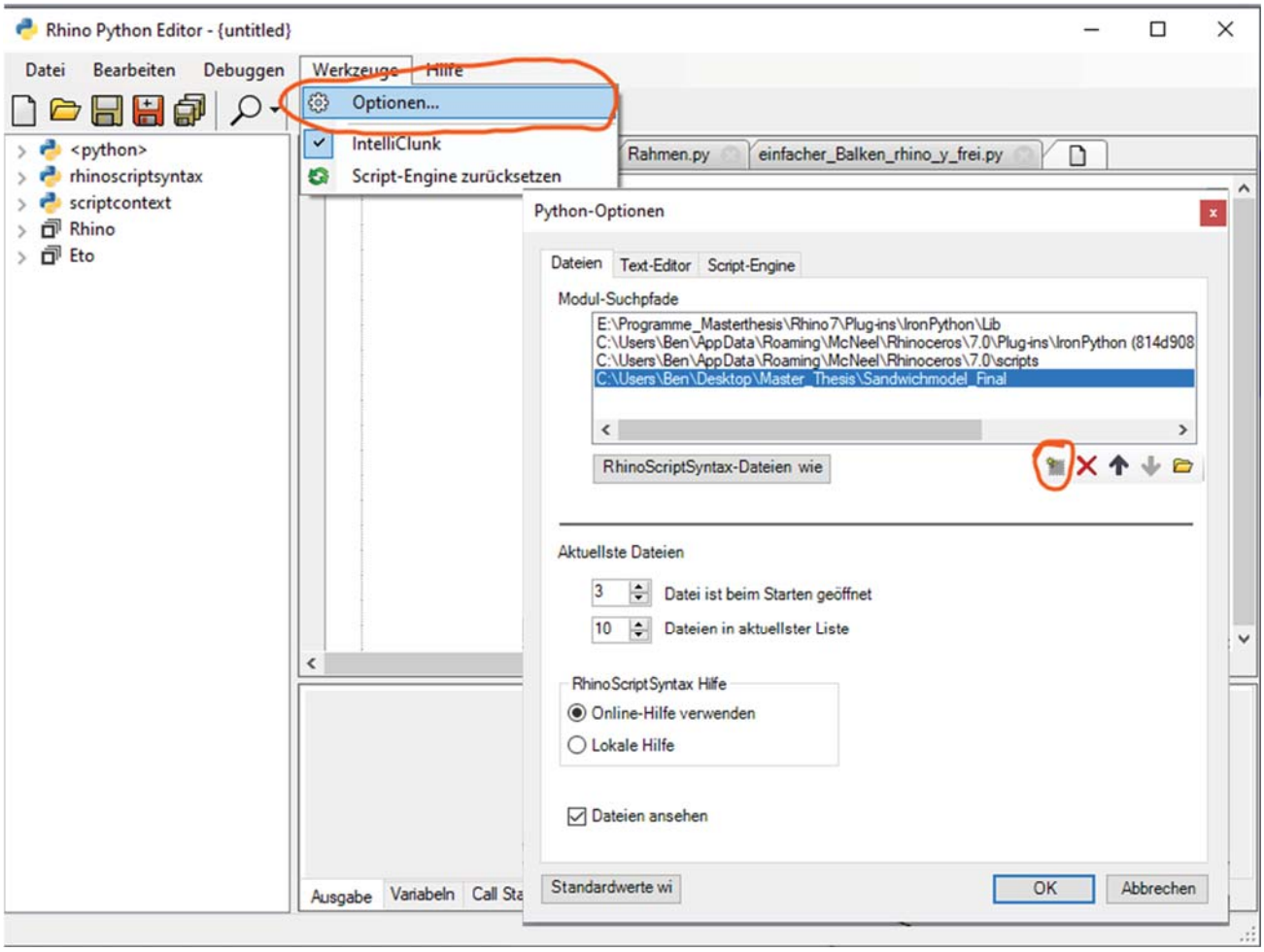
Wie in Kapitel 4.1.2 beschrieben wurde, muss die Definition der z-Achse in Rhino über die Normale erfolgen. Dazu werden über die Befehle «Richtung analysieren» und «Richtung umkehren» die Normalen der Polygonnetzte in die Richtungen der gewünschten späteren z-Achsen gesetzt.





3. Hinzufügen des Sandwichmodells im Suchpfad von Pythonskript

Um den Zugriff auf die Codes des Sandwichmodells zu ermöglichen, muss der Ordner, in dem die Module des Sandwichmodells gespeichert sind, in den Suchpfad von Pythonskript geladen werden. Diese Funktion kann über die Optionen im Pythonskript aufgerufen werden.



4. Importieren der benötigten Module

Vor Beginn der Definition der einzelnen Komponenten in Compas FEA müssen im Pythonskript die benötigten Module von Compas FEA **und dem Sandwichmodell** importiert werden.

```
from compas_fea.cad import rhino
from compas_fea.structure import ElasticIsotropic
from compas_fea.structure import ElementProperties as Properties
from compas_fea.structure import GeneralDisplacement
from compas_fea.structure import GeneralStep
from compas_fea.structure import GravityLoad
from compas_fea.structure import AreaLoad
from compas_fea.structure import PinnedDisplacement
from compas_fea.structure import ShellSection
from compas_fea.structure import Structure


import sandwichmodel_main as SMM
```

5. Allgemeine Definitionen

Als nächstes werden im Pythonskript allgemeine Rahmenbedingungen definiert. Dabei handelt es sich in diesem Falle um den Namen der Datei und dem Pfad, in welchem die Dateien gespeichert werden sollen. **Das Verzeichnis muss bereits existieren.**

```
#Allgemein
name = 'Rahmen'
path = 'C:/Temp/'
```

6. Structure

Nun wird das Objekt «structure» erstellt. Als Variable wird es «mdl» benannt.

```
# Structure
mdl = Structure(name=name, path=path)
```

7. Elemente und lokale Achsen

Jetzt können die Polygonnetze als «ShellElemente» aus den Ebenen in Rhino geladen werden. Danach werden die lokalen Achsen definiert. Dabei muss jeweils **nur ein Element pro «ShellElement» angepasst werden**. Es empfiehlt sich, die Berechnung zuerst ohne definierte Achsen durchlaufen zu lassen, um die Elementnummern aus Abaqus herauslesen zu können. In diesem Beispiel ist Element 150 in der linken Wand, Element 1 im Deck und Element 200 in der rechten Wand.

Die Achsen werden in globalen Komponenten eingegeben und sollten auf den Betrag 1 genormt sein. Diese Vektoren sind für die Berechnung der Bewehrungsrichtung und für die Funktion Achsenplotter wichtig, nur «ey» fliesst auch als Definition der «1-Achse» in Abaqus und somit in die Berechnung ein (siehe Kapitel 4.1.2). Dabei sollte «ez» die gleiche Richtung, wie in Punkt 2 dieses Tutorials definiert, aufweisen.

```
# Elements
rhino.add_nodes_elements_from_layers(mdl, mesh_type='ShellElement', layers=['elset_deck','el-
set_wall_left','elset_wall_right'])

mdl.elements[150].axes.update({'ex': [0, 0, 1], 'ey': [0, -1, 0], 'ez': [1, 0, 0]})
mdl.elements[1].axes.update({'ex': [1, 0, 0], 'ey': [0, -1, 0], 'ez': [0, 0, -1]})
mdl.elements[200].axes.update({'ex': [0, 0, -1], 'ey': [0, -1, 0], 'ez': [-1, 0, 0]})
```

8. Sets

Die vielen Punkte, die später die Linienauflager bilden, werden als «set» gruppiert.

```
# Sets
rhino.add_sets_from_layers(mdl, layers=['nset_pinned'])
```

9. Materialien

Es werden die gewünschten Materialien definiert. Dabei ist es wichtig, die richtigen Einheiten zu verwenden: E-Modul «E» in N/mm² und Dichte «p» in kg/mm³, die Querdehnzahl «v» ist einheitslos.

```
# Materials
mdl.add(ElasticIsotropic(name='mat_elastic', E=33700, v=0.0, p=2500/10**9))
```

10. Querschnitte

Als nächstes werden die Querschnitte als «ShellSections» definiert. Für «ShellSections» reicht die Definition der Elementhöhe «t» aus. Dies entspricht im Code vom Sandwichmodell der Variable «h».

```
# Sections
mdl.add(ShellSection(name='sec_deck', t=500))#[mm]
mdl.add(ShellSection(name='sec_wall_left', t=250))#[mm]
mdl.add(ShellSection(name='sec_wall_right', t=250))#[mm]
```

11. Properties / Eigenschaften

Nun werden die Elemente, Materialien und Querschnitte miteinander als Properties verknüpft.

```
# Properties
mdl.add(Properties(name='ep_deck', material='mat_elastic', section='sec_deck',
elset='elset_deck'))

mdl.add(Properties(name='ep_wall_left', material='mat_elastic', section='sec_wall_left',
elset='elset_wall_left'))

mdl.add(Properties(name='ep_wall_right', material='mat_elastic', section='sec_wall_right',
elset='elset_wall_right'))
```

12. Additional Properties / Zusätzliche Parameter (Neu mit Sandwichmodell)

Für die Benutzung des Sandwichmodells müssen an dieser Stelle **zusätzliche Parameter** für die einzelnen Properties definiert werden.

Dazu wird **zuerst ein leeres Dictionary «data»** erstellt. Als nächstes wird jede in Punkt 11 definierte Property mit zusätzlichen Parametern ergänzt. Dies wird über die Funktion «additionalproperty» ermöglicht. Wichtig ist, dass der Name der Property aus Punkt 11 und der Eingabewert «prop_name» eins zu eins übereinstimmen. Danach ist das Dictionary «data» gemäss Abb. 35 aufgebaut.

```
# Additional Properties
data = {} #generiert ein leeres dictionary fuer zusaetzliche Materialkennwerte

SMM.additionalproperty(data, prop_name = 'ep_deck', d_strich_bot = 40, d_strich_top = 40, fc_k =
30, theta_grad_kern = 45, fs_d=435, alpha_bot =0, beta_bot = 90, alpha_top =0, beta_top = 90)
#zusaetzliche Materialkennwerte werden gefuellt...

SMM.additionalproperty(data, prop_name = 'ep_wall_left', d_strich_bot = 40, d_strich_top = 40,
fc_k = 20, theta_grad_kern = 45, fs_d=435, alpha_bot = 0, beta_bot = 90, alpha_top = 0, beta_top =
90) #zusaetzliche Materialkennwerte werden gefuellt...

SMM.additionalproperty(data, prop_name = 'ep_wall_right', d_strich_bot = 40, d_strich_top = 40,
fc_k = 20, theta_grad_kern = 45, fs_d=435, alpha_bot = 0, beta_bot = 90, alpha_top = 0, beta_top =
90) #zusaetzliche Materialkennwerte werden gefuellt...
```

13. Verformungen

Es werden nun die Auflager als gehaltene, rotierbare Knotenauflager definiert. Diese werden für die Randbedingungen der FE-Analyse verwendet.

```
# Displacements
mdl.add([PinnedDisplacement(name='disp_pinned', nodes='nset_pinned')])
```

14. Lasten

Nun können im Pythonskript die Lasten definiert werden. In diesem Fall wird die Schwerkraft auf alle Elemente und eine Flächenlast nur auf die obere Decke modelliert. Bei den Lasten müssen die Einheiten und Richtungen besonders genau beachtet werden: Flächenlasten «AreaLoad» werden in [N/mm²] definiert, ihre Richtungen x,y,z entsprechen den **lokalen Richtungen**. *Knotenlasten «PointLoad» (hier nicht verwendet) werden in **globalen Richtungen** und in Newton [N] definiert!*

```
# Loads
mdl.add(GravityLoad(name='load_gravity', elements=['elset_deck','elset_wall_left',
'elset_wall_right']))

mdl.add(AreaLoad(name='load_pressure', elements='elset_deck', z=0.03, axes='local'))
```

15. Lastschritte

Nun werden die Verformungen und die Lasten in Lastschritte definiert und in einem nächsten Befehl die Reihenfolge der Lastschritte festgehalten. Die Verformungen dienen in diesem Beispiel den Randbedingungen.

```
# Steps
mdl.add([
    GeneralStep(name='step_bc', displacements=['disp_pinned'], nlgeom=False),
    GeneralStep(name='step_load', loads=['load_gravity','load_pressure'], nlgeom=False)
    ])

mdl.steps_order = ['step_bc','step_load']
```

16. Zusammenfassung

Optional kann das fertigdefinierte Objekt zusammengefasst und angezeigt werden.

```
# Summary
mdl.summary()
```

17. FE-Berechnung

Jetzt erfolgt die FE-Analyse. **Für das Sandwichmodell ist es wichtig, dass die Fields «sf» und «sm» berechnet werden**, da diese die benötigten Schnittkräfte beinhalten.

```
# Run
mdl.analyse_and_extract(software='abaqus', fields=['u','sf','sm'])
```

18. Plot der FE-Berechnungsresultate

Optional können die Resultate der FE-Berechnung für den gewünschten Lastschritt in Rhino geplottet werden. Compas FEA bietet viele zusätzliche Einstellungen zu den Plots an.

```
# Plot FE-results
rhino.plot_data(mdl, step='step_load', field='sm1',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sm2',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sm3',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sf4',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sf5',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sf1',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sf2',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='sf3',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='um',cbar_size=1)
```

19. Sandwichmodell Berechnung (Neu mit Sandwichmodell)

Nun kann die Anwendung des Sandwichmodells erfolgen. Dazu wird die Hauptfunktion des Sandwichmodells aufgerufen. Als Eingabewerte werden das Structure Objekt, die zusätzlichen Parameter und der zu berechnende Lastschritt eingegeben. Des Weiteren können Einstellungen zur Berechnung und Darstellung definiert werden.

```
# Sandwichmodell
SMM.Hauptfunktion(structure = mdl, data = data, step = 'step_load', Mindestbewehrung = False,
Druckzoneniteration = True, Schubnachweis = 'sia', code = 'sia', axes_scale = 100, plot_local_axes
= True, plot_reinf = True)
```

20. Plot der Sandwichmodellresultate (Neu mit Sandwichmodell)

Um die Resultate des Sandwichmodells in Rhino anzuzeigen, müssen diese geplottet werden. Da die Resultate in «structure.results» sind, können diese mit demselben Befehl wie die Resultate der FE-Berechnung (Punkt 18) geplottet werden.

```
# Plot SM-results
rhino.plot_data(mdl, step='step_load', field='as_xi_bot', cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='as_xi_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='as_eta_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='as_eta_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='as_z',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='CC_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='CC_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='k_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='k_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='t_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='t_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='psi_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='psi_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='Fall_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='Fall_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='m_cc_bot',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='m_cc_top',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='m_shear_c',cbar_size=1)
rhino.plot_data(mdl, step='step_load', field='m_c_total',cbar_size=1)
```

21. Anzeigen der Maximalwerte der Bewehrung (Neu mit Sandwichmodell)

Als Hilfsmittel werden optional mit der Funktion «max_values» am Schluss die maximalen Längsbewehrungen und deren Lage in der Befehlszeile angezeigt.

```
SMM.max_values(mdl,'step_load')
```

22. Ausführen des Pythonskripts

Zuletzt muss das Skript noch ausgeführt werden. Es empfiehlt sich vorher zu speichern. Es kommt immer wieder zu Abstürzen. In diesem Fall hilft meistens ein Schliessen und Neuöffnen von Rhino.