Python Praktikum zur Statistik einer Mischung

Schritt 1. Setup: Sollten Sie keine lokale Python Installation besitzen, gehen sie zu *replit.com* (https://replit.com/@frankrhein/Mixing-Statistics) und erstellen Sie zunächst einen kostenlosen Account. Klicken Sie anschließend den Button *Fork Repl* um eine Kopie der Entwicklungsumgebung und Dateien zu erzeugen. Sie sehen jetzt (hoffentlich) eine voll funktionsfähige Linux Umgebung. Auf der linken Seite sehen Sie den Datei-Explorer mit den jeweiligen Skripten. Auf der rechten Seite finden Sie eine Linux-Shell, über die wir die Skripte ausführen werden. Sollte ihnen etwas verloren gehen können Sie über *Tools* (links unten) jederzeit z. B. eine neue Shell öffnen.
Haben Sie eine lokale Python Distribution (z. B. Anaconda https://www.anaconda.com/) installiert, so benötigen Sie nicht unbedingt einen Replit Account. Die Dateien sind ebenfalls auf ILIAS verfügbar.

Schritt 2. Python 101: Öffnen sie die Datei `hello_world.py` und studieren Sie in Ruhe deren Inhalt. Um den Code auszuführen nutzen wir die Linux Shell. Führen Sie dazu folgenden Befehl aus:

```
python hello_world.py
```

Sie sollten nun die im Skript getätigten Ausgaben in der Shell lesen. Um den angefertigten Plot sehen zu können navigieren Sie links unter *Files/export* und öffnen die Datei `hello_world_testplot.pdf` über die drei Punkte bzw. die Option *Open tab*. Da wir noch öfter Plots visualisieren werden, bietet es sich an schon hier unser Interface etwas zu optimieren: Den neuen Tab können Sie z. B. in die untere rechte Browser-Hälfte ziehen, sodass ihr Interface in etwa so wie in Abbildung 1 aussieht.

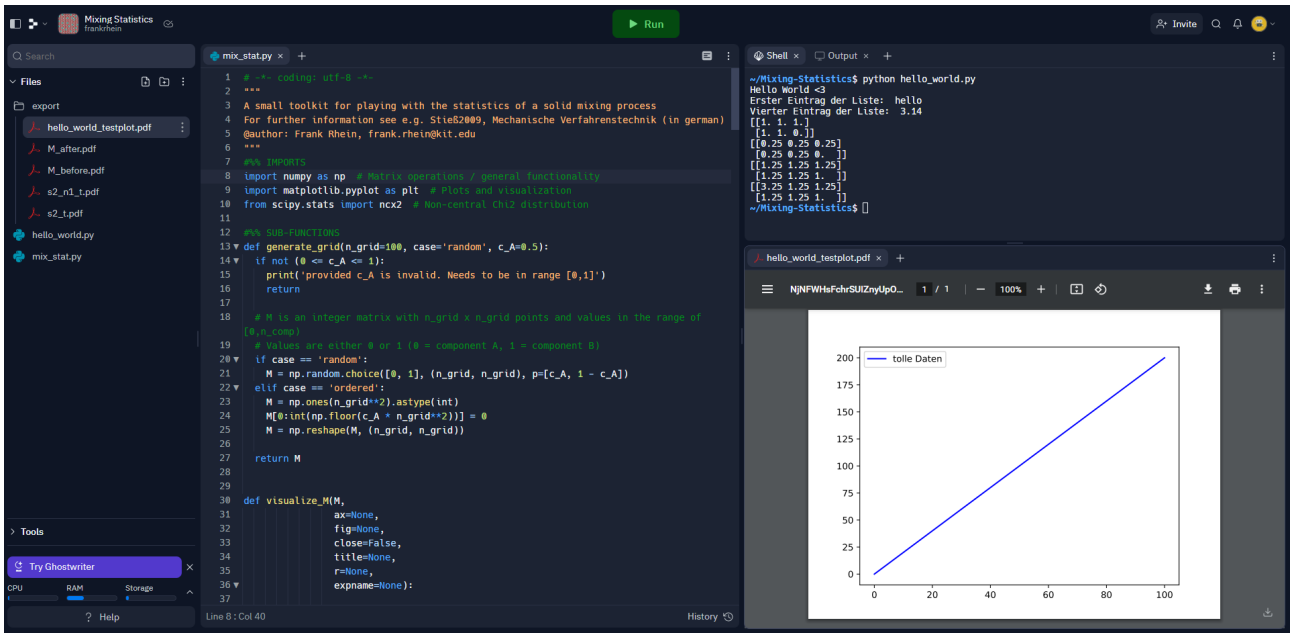

Abbildung 1: Beispielhaftes *Replit* Interface mit File-Browser (links), Code-Editor (mittig), Shell (oben rechts) und geöffneter .pdf (unten rechts).

Primäres Ziel dieser Übung soll es nicht sein, dass Sie sämtliche Befehle und Kniffe der Programmiersprache Python beherrschen, sondern dass Sie das bereitgestellte Skript verwenden können, ohne zu stark vom Code abgeschreckt zu sein. Um Python wirklich kennenzulernen,

gibt es eine Vielzahl herausragender Online-Tutorials, wie z.B. https://realpython.com/ oder https://www.learnpython.org/.

Schritt 3. Ein erster Eindruck: Wagen wir uns jetzt an das Skript `mix_stat.py`. Im Groben lässt sich die Datei in drei Bereiche einteilen:

1) Imports: Hier werden alle notwendigen zentralen Packages und Funktionen importiert
2) Sub-Functions: Es bietet sich an über dem "eigentlichen Code" die notwendigen Funktionen zu definieren, die wir später brauchen werden. Überspringen wir zunächst sämtliche Funktionen und kommen zum Abschnitt..
3) Main: Hier werden die eigentlichen Aktionen durchgeführt. Eingeleitet wird dieser Abschnitt mit dem Konstrukt

```
1  if __name__ == '__main__':
```

Durch diese Zeile wird der folgende Code nur dann ausgeführt, wenn das Programm direkt aufgerufen wird und eben nicht, wenn es z. B. von einem anderen Skript importiert wird. Dies erhöht die Flexibilität, da die definierten Unterfunktionen generell auch für andere Projekte zur Verfügung stehen.

Schritt 4. Der Main-Abschnitt: Das Main-Skript beginnt mit der Definition der notwendigen Parameter. Diese sind im Folgenden aufgeführt und orientieren sich an den Definitionen in den Folien. Tabelle 1 listet die Standardwerte der jeweiligen Parameter.

- `t_exp`: Zeit des Mischvorgangs in der Einheit "Zeitschritte". Was das genau bedeutet sehen wir später.
- `num_t_samples`: Anzahl an zeitlichen Probenahmen über dem Mischprozess (linear verteilt).
- `num_samples_per`: Anzahl an Proben pro Zeitschritt.
- `n_grid`: Gittergröße. Wir rechnen auf einem quadratischen 2D-Gitter, bei dem jeder Gitterpunkt einem diskreten Partikel entspricht. Die Gesamtanzahl an Partikeln beträgt folglich $\texttt{n\_grid}^2$.
- `Z`: Probengröße bzw. Anzahl an Einzelpartikeln pro Probe.
- `c_A`: Anzahlkonzentration Stoff A ($c_A \in [0, 1]$).
- `D`: Der "Diffusionskoeffizient" beschreibt die Geschwindigkeit unserer Durchmischung. Auch dazu später mehr.
- `alpha`: Vertrauensgrenze (in %) zur Berechnung des Konfidenzintervalls der Varianz.

t_exp	num_t_samples	num_samples_per	n_grid	Z	c_A	D	alpha
2000	15	20	50	50	0.5	$5e-4$	75

Tabelle 1: Standardwerte der allgemeinen Parameter.

Es folgen weitere Parameter, die für zusätzliche Untersuchungen zur Entmischung und zu Totzonen relevant sind. All diese Parameter laden dazu ein an ihnen herum zu spielen. Dem werden wir noch nachgehen, lesen wir jedoch zunächst weiter.

Schritt 5. Das "Experiment": Nach der Parameterdefinition finden Sie den folgenden Befehl im Main-Skript:

```
samples, M = perform_experiment(t_exp=t_exp, c_A=c_A, t_samples=t_samples
    , num_samples_per=num_samples_per, n_grid=n_grid, Z=Z, D=D,
    visualize_mix=True, deadzone=deadzone)
```

Hier wird die Unterfunktion `perform_experiment()` aufgerufen und alle notwendigen Parameter übergeben. Als Output erhalten wir zum einen den Array der die gezogenen Proben enthält, mit denen wir im Anschluss eine statistische Auswertung des Mischprozesses durchführen können. Zusätzlich wird die Mischung `M` zurück gegeben.
Springen wir nun zur Unterfunktion `perform_experiment()` und versuchen zu verstehen was hier passiert. Zunächst wird über den Befehl

```
M = generate_grid(n_grid=n_grid,case='ordered',c_A=c_A)
```

die Rechenmatrix bzw. die Mischung `M` initialisiert. `M` ist ein 2D-Array mit den Dimensionen $(n_{grid} \times n_{grid})$. Jede Position repräsentiert ein Partikel. Der Wert an der jeweiligen Stelle gibt an, um welches Partikel es sich handelt. 0 steht für ein Partikel des Materials A und 1 für ein Partikel des Materials B. Durch die Option `case='ordered'` geben wir an, dass wir eine geordnete Mischung (aufeinandergeschichtete Partikel) initialisieren möchten (vollständige Entmischung). In Kürze werden wir uns das System visualisieren.
In der Schleife

```
for i in range(t_exp):
  # Swap corresponding number of times
  if deadzone is None:
    M = swap_n_elements(M, num_swaps)
  else:
    M = swap_n_elements_dz(M, num_swaps, deadzone)

  # Time for some samples!
  if i in t_samples:
    for n in range(num_samples_per):
      tmp_sample, _ = pick_sample(M, Z)
      samples[cnt_t, n, :] = tmp_sample

    cnt_t += 1  # increase counter
```

erfolgt das eigentliche Misch-Experiment. Wir "simulieren" den Mischprozess dadurch, dass wir zu jedem Zeitschritt über die Funktion `swap_n_elements()` zufällig 2 Elemente der Matrix `M` auswählen und vertauschen (ignorieren wir zunächst die Unterscheidung bezüglich der Variablen `deadzone`). Dies modelliert letztlich eine diffusive Durchmischung des Systems. Dieses Vertauschen wird `num_swaps`-fach wiederholt, wobei sich diese Variable aus unserem Diffusionskoeffizienten D ableitet und die Geschwindigkeit unserer Durchmischung festlegt. Zu vorher, bzw. durch `t_samples` festgelegten Zeitpunkten werden zusätzlich Proben entnommen und im Array `samples` gespeichert. Kümmern wir uns zunächst nicht weiter darum, sondern visualisieren wir uns lieber wie die Mischung vor und nach unserem Mischexperiment aussieht. Führen wir hierzu das Skript aus, achten jedoch darauf, dass die Option `visualize_mix=True` beim Aufruf von `perform_experiment` übergeben wird. Dazu tippen wir folgenden Befehl in die Linux-Shell:

```
python mix_stat.py
```

Im Ordner *Files/export* sollten (nach kurzer Rechenzeit) zwei Dateien – `M_before.pdf` und `M_after.pdf` – erscheinen und zum Betrachten einladen. Die Abbildungen zeigen die Mischung vor und nach unserem Mischprozess und sollten in etwa wie in Abbildung 2 aussehen.

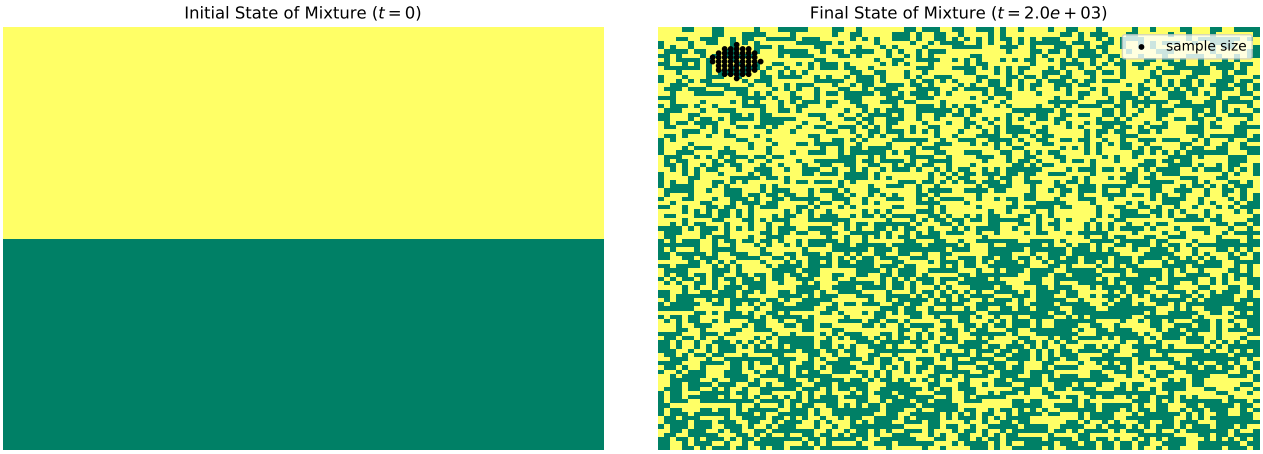

Abbildung 2: Die Rechenmatrix `M` vor und nach dem Mischexperiment. Die Mischzeit beträgt $t_{exp} =$ 2000 Zeitschritte.

Die schwarzen Punkte zeigen die Größe der jeweils gezogenen Stichprobe an. Dabei wird in der Funktion `pick_sample()` zunächst eine zufällige Position in `M` ausgewählt. Anschließend werden die `Z-1` nächst gelegenen Indizes ausgewählt, sodass sich eine Stichprobengröße von `Z` ergibt.
Wichtig: Entgegen der realen Praxis entnehmen wir die Probe nicht wirklich aus unserem Experiment, sondern analysieren ihre Zusammensetzung, ohne die Partikel aus der Matrix `M` zu entfernen. Wir brauchen uns also keine Sorgen zu machen, dass durch eine zu häufige und große Probennahme die Gesamtmischung aufgebraucht wird. In der Realität ist dies natürlich sehr wohl relevant!

Aufgabe 1: Verinnerlichen Sie sich das hier umgesetzte modellhafte Mischprinzip indem Sie den Parameter `t_exp` variieren. Beginnen Sie hierzu bei `t_exp=1` und erhöhen Sie die Mischzeit bis `t_exp=1e4`. Tipp: Um den letzten ausgeführten Befehl in der Shell nicht neu abtippen zu müssen, können Sie diesen auch durch die Pfeiltaste ↑ kopieren. Ohne eine statistische Auswertung der Proben durchzuführen: Was beobachten Sie qualitativ und was erwarten Sie für die folgende quantitative Auswertung hinsichtlich dem Verlauf der empirischen Varianz s^2?

Aufgabe 2: Setzen Sie `t_exp` zurück auf den Wert 2000 und untersuchen Sie wie sich der Prozess verhält wenn Sie die Zusammensetzung über `c_A` und den "Diffusionskoeffizienten" `D` variieren. Suchen Sie die Zeile in der Funktion `perform_experiment()`, in der `D` Verwendung findet und versuchen Sie zu verstehen was hier passiert. Wieso ist der Umweg über einen Diffusionskoeffizienten sinnvoll anstatt einfach eine absolute Anzahl an Tauschvorgängen pro Zeitschritt festzulegen?

Schritt 6. Probenauswertung: Nun ist es an der Zeit die entnommenen Proben zu analysieren. Hierzu ist es hilfreich zunächst zu verstehen wie der 3D `samples` Array aufgebaut ist. Finden Sie hierzu die Zeile

```
samples = np.ones((len(t_samples), num_samples_per, Z))
```

in der Funktion `perform_experiment()`.

Aufgabe 3: Versuchen Sie (auch unter Zuhilfenahme des darüber stehenden Kommentars) die Struktur dieser Variablen zu verstehen und visualisieren Sie diese auf Papier. Welche beiden numerischen Einträge erwarten Sie in dem Array?

Aufgabe 4: Ihnen liegt zum Zeitschritt t der unten dargestellte Array `samples[t,:,:]` vor. Er beinhaltet 4 Proben mit je 10 Partikeln (1: Material A, 0: Material B). Berechnen Sie händisch, d. h. mit Papier und Taschenrechner die empirische Varianz s_{n-1} sowie s_n für $P = c_A = 0.5$. Wie groß ist jeweils der Vertrauensbereich b^2 bzw. b^2_{n-1} für eine Vertrauensgrenze von 95 %? Berechnen Sie weiterhin die Varianz der völligen Entmischung σ_0 sowie die Varianz der stochastischen Homogenität σ_Z für die gewählte Probengröße.

```
In [100]: samples[t,:,:]
Out[100]:
array([[1., 1., 1., 1., 0., 1., 0., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 0.],
       [1., 1., 1., 1., 0., 1., 0., 0., 1., 1.],
       [0., 1., 0., 0., 0., 0., 1., 0., 1., 0.],
       [1., 1., 0., 0., 1., 0., 0., 1., 1., 1.]])
```

(Lösung: $s^2 = 0.068$, $s^2_{n-1} = 0.053$, $b^2 = 0.296$, $b^2_{n-1} = 0.299$, $\sigma_0 = 0.25$, $\sigma_{Z,50} = 0.025$)

Hervorragend! Da wir uns jetzt erarbeitet haben, wie eine einzelne Probe auszuwerten ist, schauen wir uns die Funktion `s2_from_samples()` an, die uns das mühsame Kopfrechnen abnimmt. Ausgehend von einem `samples` Array wird für jeden Zeitschritt, d. h.

```
for t in range(samples.shape[0]):
```

die empirische Varianz sowie der zugehörige Vertrauensbereich berechnet. Abhängig davon, ob die wahre Konzentration P an die Funktion übergeben wird oder nicht verändert sich die Berechnung leicht. Studieren Sie die programmierte Umsetzung und vergleichen Sie diese mit ihrem Vorgehen aus der vorherigen Aufgabe. Das tabellarische Ablesen der χ^2_S Werte wird von der Unterfunktion `return_chi2()` übernommen und braucht uns vorerst nicht weiter zu interessieren.
Um uns die Arbeit des Programms anzusehen, setzen wir die Variable `analyze_samples` auf `True`. Dies führt dazu, dass der folgende Programmabschnitt nach dem Misch-Experiment ausgeführt wird.

```
# Calculating s2 and b2 from samples (P is given)
s2, b2 = s2_from_samples(samples, conf=alpha, P=c_A)
```

```
# Calculating s2_n1 and b2_n1 from samples (P is NOT given)
s2_n1, b2_n1 = s2_from_samples(samples, conf=alpha, P=None)

# Calculating sigma_0 and sigma_Z
sig_0 = c_A * (1 - c_A)
sig_z = sig_0 * (1 / Z)

# Visualizing s2 with known true concentration P
ax, fig = visualize_s2_t(t_samples,s2,..)
ax, fig = visualize_s2_t(t_samples,b2,..)

fig.savefig(f'export/s2_t.pdf')

# Visualizing s2_n-1 without known true concentration P
ax, fig = visualize_s2_t(t_samples,s2_n1,..)
ax, fig = visualize_s2_t(t_samples,b2_n1,..)

fig.savefig(f'export/s2_n1_t.pdf')
```

Setzen Sie zusätzlich alle Parameter zurück auf die in Tabelle 1 dargestellten Werte und führen das Programm erneut aus. Zusätzlich zur Visualisierung der Mischung am Anfang und Ende (die wir übrigens mit der Option `perform_experiment(visualize_mix=False)` ausschalten können, um die Geschwindigkeit des Programms zu erhöhen) erscheinen nun ebenfalls die Mischgüteverläufe $s^2(t)$ bzw. $s^2_{n-1}(t)$ im Ordner *Files/export*, die in etwa wie in Abbildung 3 aussehen sollten. Die empirische Varianz an den diskreten Zeitpunkten ist durch die blauen Datenpunkte und der Vertrauensbereich durch die rot eingefärbte Fläche dargestellt. σ_0 und σ_Z sind als horizontale Linie ebenfalls abgebildet.

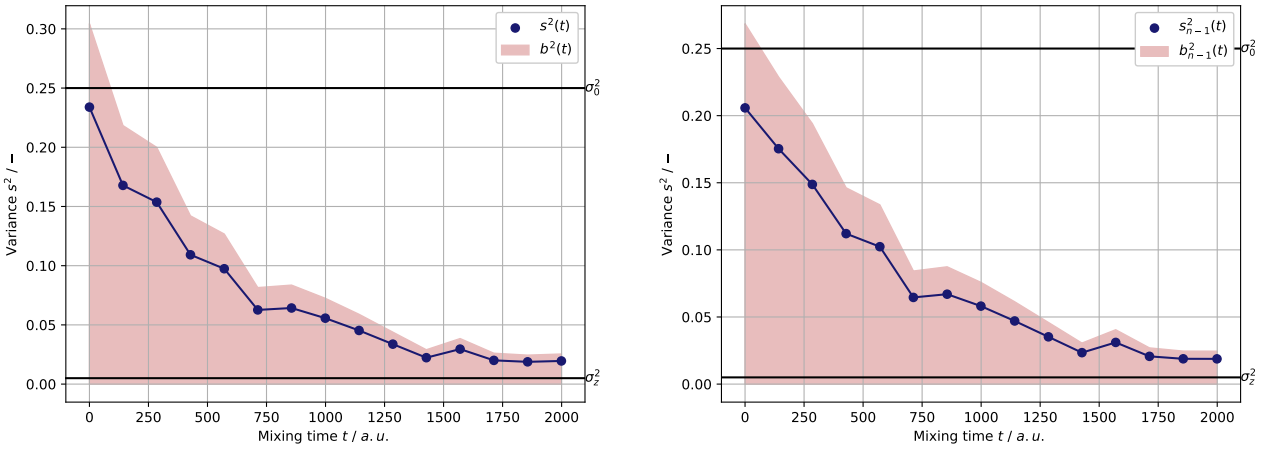

Abbildung 3: Simulierte Mischgüteverläufe $s^2(t)$ (links) und $s^2_{n-1}(t)$ (rechts).

Aufgabe 5: Wie werden sich die Parameter `num_t_samples`, `num_samples_per`, `Z`, `n_grid` und `alpha` auf den Mischgüteverlauf auswirken? Halten Sie ihre Vermutungen und Begründungen stichpunktartig fest.

Im Folgenden werden wir uns die Einflüsse der jeweiligen Parameter individuell ansehen. Dazu empfiehlt es sich die Parameter zwischen den Aufgaben wieder auf die Standardwerte aus Tabelle 1 zurück zu setzen.

Aufgabe 6: Variieren Sie den Parameter `num_samples_per`. Testen Sie zunächst eine deutlich niedrigere Anzahl an Proben (z. B. 3) und erhöhen Sie die Zahl anschließend drastisch (z. B. auf 1000). Was fällt ihnen auf?

Aufgabe 7: Untersuchen Sie die Parameter `Z` und `n_grid`. Wieso ist es sinnvoll beide Parameter zusammen zu betrachten und was sagt das Verhältnis der beiden letztlich aus?

Setzen Sie zunächst `Z` auf den Wert 1. Was passiert hier und wieso sehen Sie was sie sehen?

Eine Erhöhung von `Z` führt zu großen `samples` Arrays und einer spürbaren Verlangsamung auf der Cloud-basierten Hardware. Aus diesem Grund setzen Sie stattdessen `Z` zurück auf den Standardwert und reduzieren `n_grid` kontinuierlich, bis Sie $\texttt{n\_grid}^2 \approx \texttt{Z}$ erreichen. Was fällt ihnen auf?

Aufgabe 8: Was passiert, wenn Sie die Anzahl an zeitlichen Proben `num_t_samples` drastisch reduzieren (bis zum Minimum bei 2)? Basierend auf den bisher gesehenen Mischgüteverläufen: Wie würden Sie bei einem realen Prozess die zeitliche Probenahme gestalten, wenn Sie den Mischgüteverlauf möglichst exakt bestimmen möchten?

Schritt 7. Tot-Zonen. Setzen Sie erneut alle Parameter zurück auf die in Tabelle 1 dargestellten Werte. Im Folgenden wollen wir untersuchen wie sich ein nicht-ideales Verhalten des Mischers auf den Mischgüteverlauf auswirkt. Dazu setzen wir die Variable

```
deadzone = 0.1
```

um eine Tot-Zone in unserem Mischer zu simulieren, in der keinerlei Durchmischung auftritt. Der Zahlenwert (zwischen 0 und 1) legt die Größe dieser Zone fest.

Aufgabe 9: Was erwarten Sie, wie sich eine Variation dieses Parameters auf den Mischgüteverlauf auswirkt? Überpüfen Sie Ihre Vermutung! Betrachten Sie hierzu auch die Mischungen von und nach dem Experiment.

Schritt 8. Entmischung. Entfernen Sie zunächst sämtliche Tot-Zonen mit Hilfe von

```
deadzone = None
```

und widmen Sie sich den beiden nachfolgenden Code-Zeilen. In realen Anwendungen spielen Entmischungseffekte eine entscheidende Rolle, die z. B. durch Sedimentationseffekte oder Vibration während des Transports zustande kommen. Auch dies wollen wir mit einer (gravitationsgetriebenen) Diffusion vereinfacht modellieren. Setzen Sie hierzu zunächst den Diffusionskoeffizienten `D_demix` auf einen Wert größer 0 (z. B. `1e-5`) und lokalisieren Sie die Funktionen `perform_demixing()` bzw. `demix_n_elements()`.

Aufgabe 10: Versuchen Sie selbstständig zu verstehen wie der Entmischungsprozess hier (vereinfacht) simuliert wird. Vergleichen Sie hierzu beide angesprochene Funktionen mit ihren äquivalenten Funktionen des Mischvorgangs `perform_experiment()` bzw. `swap_n_elements()`.

Führen Sie anschließend verschiedene Experimente unter Variation der Parameter `D_demix` und `t_demix` durch. Visualisieren Sie sich hierzu die Mischungen vor und nach der Entmischung (*.pdf*-Dateien im Ordner *Files/export*). Analysieren Sie ebenfalls den resultierenden Mischgüteverlauf.

Aufgabe 11: Lassen Sie uns zum Abschluss des Praktikums die angewandten physikalischen Modelle des Misch- und Entmischvorgangs beurteilen. Für wie realisitisch halten Sie das ”diffusive” Mischmodell? Hinweis: Auch in realen Mischprozessen tritt eine Durchmischung aufgrund zufälliger Partikelbewegung auf, der sogenannte *dispersive* Transport. Wie könnten Sie den Algorithmus zum Positionswechsel verbessern, um die Realität besser abzubilden?

Berücksichtigt unser Modell Energieeintrag von außen (z. B. durch bewegte Behälter oder Mischwerkzeuge)? Denken Sie darüber nach, wie Sie z. B. eine Rotation des Behälters im Modell umsetzen könnten, um dadurch auch den sogenannten *konvektiven* Transport abzubilden.