Finanzperspektiven der EO

Technische Dokumentation

BSV MAS (sekretariat.mas@bsv.admin.ch)

27.10.2027

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument beschreibt die Implementation des Finanzperspektivenmodells der EO in der Programmiersprache R. Eine nicht-technische Zusammenfassung des Modellansatzes findet sich im Dokument „Modellbeschrieb der Finanzperspektiven der EO“. Um diese technische Dokumentation zu verstehen, ist es hilfreich, vorgängig den nicht-technischen Modellbeschrieb anzuschauen.

Das Finanzperspektivenmodell der EO besteht aus zwei Teilen. In einem ersten Teil werden die Daten, welche in Kapitel 2 beschrieben sind, aufbereitet. Das heisst, die Daten werden eingelesen, bei Bedarf in ein Format gemäss den tidy data Prinzipien umgewandelt, und danach in einem zentralen Ordner abgelegt.[1] Die Module hierzu sind in Kapitel 3 beschrieben. In einem zweiten Teil wird dann, basierend auf den im ersten Teil aufbereiteten Daten, das eigentliche Finanzperspektivenmodell berechnet. Die Module hierzu sind in Kapitel 4 beschrieben.

Das Programm ist so aufgebaut, dass es aus verschachtelten Modulen besteht. Im Falle des Teils zur Berechnung des Finanzperspektivenmodells (Kapitel 4) bedeutet dies, dass ein Hauptmodul zuerst auf ein Untermodul zugreift, das die aufbereiteten Daten einliest, und dann auf ein Untermodul, das die eigentlichen Berechnungen durchführt. Das Untermodul, das die Berechnungen durchführt ist wiederum aufgeteilt in ein Modul, das die Einnahmen der EO berechnet, und ein Modul, das die Ausgaben der EO berechnet. Diese Module sind wiederum aufgeteilt in Untermodulen nach Einnahmen- und Ausgabenkategorien.

1.1 Aufsetzen der Entwicklungsumgebung

Wir nehmen für die folgenden Erläuterungen an, dass der Ordner mit den Programmcodes mit *delfinverse* benennt und direkt auf dem Laufwerk C abgespeichert wird. Natürlich kann unter Anpassung des Grundpfades jeder beliebige Ordnername und Speicherort gewählt werden.

Um die Entwicklungsumgebung aufzusetzten genügt das folgende kurze Skript, das die Pfade definiert und die im Finanzperspektivenmodell verwendeten R-Packete lädt:

```
setwd("C:/delfinverse")

devtools::load_all("dinfra")
devtools::load_all("dinput")
devtools::load_all("delfin")
devtools::load_all("dmeasures")
devtools::load_all("doutput")
```

Die Packete enthalten durch das BSV entwickelte Programme für die folgenden Zwecke:

- `dinfra`: Grundprogramme, welche in allen Berechnungsschritten des Finanzperspektivenmodells immer wieder aufgerufen werden.
- `dinput`: Programme zur Aufbereitung der Input-Daten (vlg. Kapitel 3).
- `delfin`: Programme, welche den Kern des Finanzperspektivenmodells, also die Projektionen für die einzelnen Einnahmen- und Ausgabenpositionen berechnen (vgl. Kapitel 4).
- `dmeasures`: Programme zur Berechnung der Auswirkungen von Politikmassnahmen (bspw. enthält dieses Packet Module, zur Abschätzung der Kosten der EOG Revision).
- `doutput`: Programme zur optischen Aufbereitung des Outputs (bspw. die Finanzperspektiven-Übersichtstabellen, die auf der BSV-Website veröffentlicht werden).

1.2 Spezifikation der zu verwendenden Daten und Modellparameter

Die Inputdaten, respektive die Pfade zu diesen, sowie die zu verwendenden Modellparameter werden nicht direkt im R-Code, sondern „extern “ in einer .csv-Datei spezifiziert. Dies dient dazu, eine möglichst grosse Flexibilität bei der Modellierung zu erhalten, und zu verhindern, dass zum Testen von alternativen Parameterspezifikationen der Modellcode angepasst werden muss.

[1] https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

1.2.1 Spezifikation der zu verwendenden Daten

Bei der Ausführung des Codes zum aufbereiten der Input-Daten (Kapitel 3) wird die Datei `PARAM_INPUTS.csv` aufgerufen, welche angibt, welche Rohdaten eingelesen und aufbereitet werden sollen, und unter welchem Pfad diese Rohdaten abgelegt sind. Der Zweck von `PARAM_INPUTS` ist also einerseits, dass die einzulesenden Rohdaten in einer zentralen Datei ersichtlich sind, und andererseits, dass Änderungen an den einzulesenden Rohdaten einfach, d.h. ohne Anpassungen am Programmcode, vorgenommen werden können. Nachfolgend ein Auszug aus `PARAM_INPUTS.csv` (Stand Juni 2025) als Beispiel:

key	value
file_eo_abrechnung_m	sv_eo_m.xlsx
file_eo_abrechnung_fin	sv_eo_fin.xlsx
sheet_eo_abrechnung_prov	daten_prov
sheet_eo_abrechnung_def	daten

Wir sehen in der Tabelle die Angabe, in welcher Datei sich die EO-Abrechnung befindet, sowie die Angabe, in welchem Blatt des entsprechenden .xlsx die provisorische respektive die definitive EO-Abrechnung abgelegt sind.

1.2.2 Spezifikation der Modellparameter

Bei der Ausführung des Codes für die Berechnung der Finanzperspektiven (Kapitel 4) wird die Datei `PARAM_GLOBAL.csv` aufgerufen, welche angibt, mit welchen Parametern die Berechnungen des Finanzperspektivenmodells durchgeführt werden sollen. `PARAM_GLOBAL.csv` enthält nur die Parametern, die oft angepasst werden. Die restliche Parameter werden im Modul 4.3 festegelgt. Zusätzlich zu den Modellparametern kann in `PARAM_GLOBAL.csv` auch festgelegt werden, welche Grundlagedaten für die Berechnung der Finanzperspektiven verwendet werden sollen (bspw. welche Projektion für die Lohn- und Preisentwicklung oder welches BFS-Bevölkerungsszenario). Nachfolgend ein Auszug aus `PARAM_GLOBAL.csv` (Stand Oktober 2025) als Beispiel:

key	value
jahr_ende	2075
bev_scenario	A_00_2025
jahr_register	2024

In der Tabelle sehen wir in der Zeile `jahr_ende` die Angabe des Jahres, bis zu dem das Finanzperspektivenmodell berechnet werden soll. (Je länger der Projektionshorizont, desto länger die Rechnenzeit.) Die Zeile `bev_scenario` zeigt an, welches Bevölkerungsszenario für die Berechnung verwendet wird. Die Zeile `jahr_register` gibt das letzte Jahr an, für das die Daten aus dem EO-Register verwendet werden sollen – also das letzte Jahr, in dem noch nicht projiziert wird, sondern auf historischen Daten basiert wird.

Bei der Ausführung des Finanzperspektivenmodells kann optional auch eine Auswahl an Politikmassnahmen spezifiziert werden, welche bei den Berechnungen berücksichtigt werden sollen (der Modellteil zu den Politikmassnahmen ist in Kapitel 4.9 beschrieben). Diese werden in der Datei `PARAM_MASSNAHMEN_BASE.csv` spezifiziert. Diese Datei sieht wie folgt aus:

key	value
aktivierte_massnahmen	revision_betriebzlg_se_elt, revision_kzl, revision_verlaeng_msu_mut_spital, revision_verlaeng_vsu_mut_spital, revision_vsu_tot_kind, revision_bsu_kind_spital, eo_verlaeng_vsu, eo_verlaeng_msu
mass_f_1	revision_betriebzlg_se_elt, revision_kzl, revision_verlaeng_msu_mut_spital, revision_verlaeng_vsu_mut_spital, revision_vsu_tot_kind, revision_bsu_kind_spital
mass_f_2	eo_verlaeng_msu, eo_verlaeng_vsu

Wir sehen in der Tabelle in der ersten Zeile `aktivierte_massnahmen` die Politikmassnahmen, wessen Auswirkung im EO Finanzperspektivenmodell geschätzt werden soll. Die Zeilen `mass_f_1` und `mass_f_2` dienen lediglich der Strukturierung der Massnahmen bei der Darstellung in den Output-Dateien.

2 Beschrieb der verwendeten Input-Daten

Dokumentation zuletzt aktualisiert am 16.10.2025

In diesem Kapitel werden die Input-Daten beschrieben, auf welchen die Berechnungen der Ausgaben der EO beruhen. Wir verwenden Inputdaten aus dem Register der EO, Daten des BFS und EFV. Darüber hinaus verwenden wir Daten aus der Buchhaltung des BSV und Abrechnungsdaten der EO gemäss Compenswiss. Wir unterscheiden zwischen historischen Inputdaten und Projektionen.

2.1 Geburten

Um die Lebendgeburten pro Jahr zu bestimmen (`anzahl_geburten`) verwenden wir Daten aus der Statistik der natürlichen Bevölkerungsbewegung (BEVNAT) für die Jahre 2013 bis 2024 nach Geburtsjahr, Alter und Geschlecht der Eltern[2]. Die Daten werden nach Geburtsjahr des Kindes, Geschlecht des Elternteils sowie nach folgenden Altersgruppen aggregiert:

1. 15 bis 19 Jahre
2. 20 bis 24 Jahre
3. 25 bis 29 Jahre
4. 30 bis 34 Jahre
5. 35 bis 39 Jahre
6. 40 bis 44 Jahre
7. 45 Jahre und älter

Unsere Projektionen basieren auf den durch das BFS erstellten Szenarien zur Bevölkerungsbewegung.[3] Für die Modellierung des Mutterschaftsurlaubs verwenden wir Projektionen der Lebendgeburten aggregiert nach den oben genannten Altersgruppen und Jahr. Für den Urlaub des anderen Elternteils verwenden wir die Summe der Geburten pro Jahr.

Für die Berechnung eines Korrekturfaktors zur Hochrechnung der Vaterschaftsurlaube 2024 nutzen wir Daten zu Lebendgeburten pro Monat und Jahr der Statistik der natürlichen Bevölkerungsbewegung (BEVNAT).[4] Das Vorgehen ist weiter unten beschrieben.

[2] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/bevnat.html
[3] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/zukuenftige-entwicklung/schweiz-szenarien.html
[4] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/geburten-todesfaelle.assetdetail.36144426.html

Die Daten betreffen die folgenden Input-Files: `LEBENDGEBURTEN_FRAU`, `LEBENDGEBURTEN_MANN`, `LEBENDGEBURTEN_SZENARIO`, `LEBENDGEBURTEN_SZENARIO_ALTER_MUTTER`.

2.2 Daten zur Wohn- und Erwerbsbevölkerung

Wir quantifizieren den Bestand der männlichen Schweizer Bevölkerung (`bevendejahr`) in der Vergangenheit mit Daten zur ständigen Wohnbevölkerung (Ende Jahr) nach Alter und Geschlecht aus der Statistik der Bevölkerung und der Haushalte (STATPOP)[5](Jahre 2011 bis 2024). Um eine komplette Zeitreihe der historischen Entwicklungen zu erhalten ergänzen wir die STATPOP für die Jahre 2008 bis 2010 zudem mit Daten zur Synthesestatistik von Stand und Struktur der Bevölkerung (ESPOP)[6] des BFS.

Wir quantifizieren die historische Erwerbsquote der Frauen (`erwq`) für die Jahre 2013 bis 2024 mit Daten zur Erwerbsbevölkerung nach Alter und Geschlecht (in Anzahl Personen und in Vollzeitäquivalenten). Diese Quote berechnen wir, anhand des Vorgehens des BFS, direkt aus den Rohdaten der Schweizerischen Arbeitskräfteerhebung (SAKE)[7]. Wir aggregieren die Erwerbsquote pro Jahr und Altersgruppe.[8]

Unsere Projektionen für die beiden Variablen basieren auf den durch das BFS erstellten Wohnbevölkerungsszenarien[9] sowie den Szenarien der Erwerbsbevölkerung in Vollzeitäquivalenten[10] projizierten Entwicklungen. Diese Szenarien werden alle fünf Jahre aktualisiert, wobei Stand Oktober 2025 die letzte Aktualisierung 2025 stattgefunden hat. Beide Szenarien basieren auf dem Inländer-Konzept, das heisst, dass sie die Entwicklung der ständigen Wohnbevölkerung respektive die Erwerbsbevölkerung in Vollzeitäquivalenten innerhalb der ständigen Wohnbevölkerung abdecken. Wir basieren uns im Finanzperspektivenmodell durchgehend auf das Referenzszenario für die Wohnbevölkerung respektive die Erwerbsbevölkerung in Vollzeitäquivalenten.

Wir passen die Bevölkerungsszenarien auf das Niveau der letzten Beobachtung von STATPOP an:

$$BEVOELKERUNG_{t,a,s,n} = SZENARIO_{t,a,s,n} * \frac{BEV\_STATPOP_{T,a,s,n}}{SZENARIO_{T,a,s,n}}$$

Die Bevölkerung im Jahr t mit Alter a, Geschlecht s, Nationalität n. $BEVOELKERUNG_{t,a,s,n}$ entspricht der im BFS-Szenarion projizierten Bevölkerung $SZENARIO_{t,a,s,n}$ korrigiert um die Abweichung der Bevölkerung im letzten Beobachtungsjahr T von STATPOP $BEV\_STATPOP_{T,a,s,n}$ von der im BFS-Szenario projizierten Bevölkerung im selben Jahr $SZENARIO_{T,a,s,n}$.

Die Daten betreffen die folgenden Input-Files: `BEVOELKERUNG`

2.3 Daten zu GrenzgängerInnen

Wir quantifizieren die Wachstumsrate der GrenzgängerInnen basieren auf der prognostizierten Anzahl der GrenzgängerInnen (`anzahl_grenzganger`) der Szenarien zur Entwicklung der ausländischen GrenzgängerInnen.[11] Sie wurden vom BFS für uns nach Alter, Geschlecht und Jahr bereitgestellt.

Die Daten betreffen die folgenden Input-Files: `FRONTALIERS_SCEN`

2.4 Daten zur Lohn- und Preisentwicklung

Für die historische Lohnentwicklung verwenden wir Daten zum nominalen Schweizerischen Lohnindex (SLI)[12] mit Basis 1939=100 für den Zeitraum 1980 bis 2024. Unsere Projektionen für die Lohnentwicklung gemäss SLI respektive LIK basieren auf den durch die Eidgenössische Finanzverwaltung (EFV) projizierten Eckwerte

[5]https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/statpop.html
[6]https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/espop.html
[7]https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/erhebungen/sake.html
[8]Es werden dieselben Altersgruppen, wie für die Geburtendaten verwendet.
[9]https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/szenarien.html
[10]https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/erwerbstaetigkeit-arbeitszeit/erwerbsbevoelkerung/kuenftige-entwicklung-erwerbsbevoelkerung.html
[11]https://www.bfs.admin.ch/bfs/de/home/statistiken/kataloge-datenbanken.assetdetail.34969165.html
[12]https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/loehne-erwerbseinkommen-arbeitskosten/lohnindex.html

für die Finanzplanung bis 2029.[13] Zusätzlich zu den publizierten Eckwerten für die Finanzplanungsperiode (aktuelles Jahr und die kommenden 4 Jahre) stellt uns die EFV Projektionen für die SLI-Entwicklung für die Mittelfristperspektive, also die 5 Jahre nach den Finanzplanungsjahren, zur Verfügung. Für die Erstellung ihrer Projektionen stütz sich die ESTV einerseits auf die Prognosen der Expertengruppe Konjunkturprognose des Bundes, und andererseits auf die Mittelfristprognosen des Staatssekretariats für Wirtschaft SECO. Detaillierte Dokumentationen sind auf der Webseite der EFV, respektive auf Anfrage direkt bei der EFV, erhältlich.

Wir verwenden den SLI (`li`) und die Wachstumsrate des SLI in Prozent (`lohn`): $\frac{SLI_t - SLI_{t-1}}{SLI_{t-1}}$, um die Entwicklung der Taggelder der Dienstleistenden, des Mutterschaftsurlaubs und Urlaubs des anderen Elternteils zu modellieren.

Die Daten betreffen die folgenden Input-Files: `ECKWERTE_EXTENDED`

2.5 Daten basierend auf systembedingten Grössen

Bei der Berechnung einer EO-Entschädigung sind Mindestgarantien und Höchstbeträge zu respektieren. Die dazu benötigten Daten sind eindeutig ableitbar aus dem Höchstbetrag der Gesamtentschädigung nach Art. 16a EOG. Sie ändern sich also nur bei einer Änderung dieses Höchstbetrages, d.h. bei einer Anpassung der Erwerbsersatzordnung an die Lohnentwicklung. Eine solche kommt zustande, wenn das jährliche Lohnwachstum von 12% übersteigt und die letzte Anpassung mindestens 2 Jahre in der Vergangenheit liegt. In den für den Finanzhaushalt relevanten Jahren, seit 2008, gab es zwei Anpassungen; eine 2009 und eine 2023.

Für die Finanzhaushalte sind die folgenden Grössen relevant:

Parameter	Beschreibung	Berechnung	Betrag (2025)
gesamtentsch	Höchstbetrag der Gesamtentschädigung pro Tag	gesamtentsch	275.- CHF
eomax	Maximale Grundentschädigung pro Tag (ohne Kinder)	0.8 * gesamtentsch + 0.99	220.- CHF
eomin	Minimale Grundentschädigung pro Tag (ohne Kinder)	0.25 * gesamtentsch + 0.99	69.- CHF
kz	Kinderzulage pro Tag gemäss Art. 18 EOG	0.08 * gesamtentsch + 0.99	22.- CHF
bz	Betriebszulage pro Tag gemäss Art. 8 Abs. 2 EOG	0.27 * gesamtentsch + 0.99	75.- CHF

Die Modellierung der zukünftigen Minimal- und Maximalwerte erfolgt im Modul `mod_eomax`.

Daten betreffen die folgenden Input-Files: `EOMAX_hist`

2.6 Daten aus den Registern des BSV

Wir verwenden die folgenden Daten aus dem Register der Erwerbsersatzordnung.

2.6.1 Dienst

Für die Schätzung der Ausgaben für Dienstleistende verwenden wir Daten aus dem Register der Erwerbsersatzordnung für die Jahre 2010 bis 2024. Der Registerstand ist der 30. September 2025 für das Jahr 2024, für die Jahre davor entspricht er jeweils dem 30. April zwei Jahre später (t+2). Ausgangspunkt ist der Datensatz `apg_yyyy_etatjjjjmm_rajjjjmm`, in dem alle Leistungsansprüche separat erfasst sind. Leistungsansprüche eines Jahres sind aggregiert, wenn ein Leistungsanspruch in mehreren Tranchen ausgezahlt wird. Die Daten

[13] https://www.efv.admin.ch/efv/de/home/finanzberichterstattung/daten/eckwerte-finanzplanung.html

werden zunächst auf Personenebene, Leistungsart und Jahr des Leistungsanspruch aggregiert. Hier werden nur Beobachtungen mit positiven Leistungen und Leistungsdauern berücksichtigt. Wir unterscheiden zwischen den folgenden Leistungsarten (`kat_leist`):

- (100): Armee – Normaldienst, Gradänderungsdienst, Durchdiener Kader, Unterbrüche,
- (130): Armee – Rekrutierung,
- (110): Armee - Dienst als Rekrut,
- (210): Zivilschutz – Grundausbildung,
- (200): Zivilschutz – Dienstleistungen Mannschaft, Kader und Spezialisten, Kommandant/in,
- (410): Zivildienst mit Rekruteneinsatz,
- (400): Zivildienst – normale Dienstleistung,
- (300): Kaderbildung Jugend und Sport,
- (500): Jungschützenkurse.

Auf Personenebene nehmen wir folgende Bereinigung in den Daten vor:

- Wir berechnen die durchschnittliche Gesamtentschädigung pro Tag und Person als Gesamtentschädigung dividiert durch die Anzahl der geleisteten Tage. Die Höhe der durchschnittlichen Grundentschädigung pro Tag berechnen wir aus der durchschnittlichen Gesamtentschädigung abzüglich der Kinderzulagen pro Tag und Person.
- Die Kinderzulagen pro Tag und Person berechnen wir für Personen mit Kindern als Ansatz pro Kind (`kz`) · Anzahl Kinder, für Taggelder, die unter der maximalen Gesamtentschädigung liegen. Für Taggelder, die der maximalen Gesamtentschädigung entsprechen, d.h. plafoniert sind, errechnen wir die Kinderzulage als Differenz der maximalen Gesamtentschädigung und der maximalen Grundentschädigung. Dabei treffen wir die Annahme, dass Personen die auf der maximalen Grundentschädigung sind auch ohne Kinder dort wären. Die Zulage ist entsprechend 0 für Personen ohne Kinder.
- Die Kinderzulagen pro Leistungsbeziehenden berechnen wir als Kinderzulage (wie oben beschrieben) · Dauer der Leistung in Tagen.
- Die Betriebszulagen pro Leistungsbeziehenden berechnen wir für Personen mit dieser Zulage als Ansatz der Betriebszulage (`bz`) · Dauer der Leistung in Tagen. Für Personen ohne Betriebszulage ist die Zulage 0.
- Die Zulagen für Betreuungskosten werden im Register als Gesamtbetrag für die gesamte Leistungsdauer ausgewiesen. Wir übernehmen diese Summe. Im Anschluss werden die Daten nach Jahr des Leistungsanspruchs, Leistungsart und Alterskategorie aggregiert.

Wir verwenden die folgenden 7 Altersgruppen (`kat_alt`):

- (1): zwischen 17 und 19
- (2): zwischen 20 und 24
- (3): zwischen 25 und 29
- (4): zwischen 30 und 34
- (5): zwischen 35 und 39
- (6): zwischen 40 und 44
- (7): über 45

Wir berücksichtigen keine Dienstleistende, die jünger als 17 sind, da die Dienstpflicht erst Volljährige betrifft.

Wir berechnen für die gruppierten Daten nach Jahr, `kat_leist` und `kat_alt` folgende Input-Variablen:

- Anzahl Leistungsbeziehende (`dienstleistende`): als Summe der Leistungsbeziehenden.
- Anteil der Leistungsbeziehenden mit EO-Max (`anteil_eomax`): als Anteil der Leistungsbeziehenden mit der maximalen Grundentschädigung pro Tag an der Summe der Leistungsbeziehenden.
- Anteil der Leistungsbeziehenden mit EO-Min (`anteil_eomin`): als Anteil der Leistungsbeziehenden mit der minimalen Grundentschädigung pro Tag an der Summe der Leistungsbeziehenden.
- Anteil der Leistungsbeziehenden zwischen EO-Min und EO-Max (`anteil_zwminmax`): als Anteil der Leistungsbeziehenden mit einer Grundentschädigung zwischen der minimalen und maximalen Grundentschädigung pro Tag an der Summe der Leistungsbeziehenden.
- Durchschnittliche Dauer der Leistung in Tagen für Leistungsbeziehende mit EO-Max (`tage_eomax`): als Durchschnitt über die bezogenen Tage aller Leistungsbeziehenden mit der maximalen Grundentschädigung pro Tag.
- Durchschnittliche Dauer der Leistung in Tagen für Leistungsbeziehende mit EO-Min (`tage_eomin`): als Durchschnitt über die bezogenen Tage aller Leistungsbeziehenden mit der minimalen Grundentschädigung pro Tag.
- Durchschnittliche Dauer der Leistung in Tagen für Beziehende mit einem Tagesansatz zwischen dem Minimum und Maximum (`tage_zwminmax`): als Durchschnitt über die bezogenen Tage aller Leistungsbeziehenden zwischen der minimalen und maximalen Grundentschädigung pro Tag.
- Durchschnittliches Taggeld für Beziehende mit einem Grundentschädigung zwischen dem Minimum und Maximum pro Tag (`taggeld`): als durchschnittliche Grundentschädigung pro Tag, die zwischen die minimale und maximale Grundentschädigung fällt, gemittelt über alle bezogenen Tage.
- Summe der Kinderzulagen (`sum_kz`): als Summe der Kinderzulagen über alle Leistungsbeziehenden.
- Summe der Betriebszulagen (`sum_betriebsz`): als Summe der Betriebszulagen über alle Leistungsbeziehenden.
- Summe der Zulage für Betreuungskosten (`sum_betreuungsz`): als Summe der Zulagen für Betreuungskosten über alle Leistungsbeziehenden.

Wir berechnen zusätzlich die Summe der ausbezahlten Leistungen (`sum_taggelder`) aggregiert nach Jahr des Leistungsanspruchs und Leistungsart.

Die Daten betreffen die folgenden Input-Files: `REGISTER_DIENST`

2.6.2 Elternschaft

Für die Schätzung der Ausgaben für die Elternschaft verwenden wir Daten aus dem Register der Erwerbsersatzordnung. Für den Mutterschaftsurlaub verwenden wir Daten für die Jahre 2013 bis 2024, für den Vaterschafts- und Betreuungsurlaub für die Jahre 2021 bis 2024, für den Adoptionsurlaub für die Jahre 2022 bis 2024. Der Registerstand ist der 30. September 2025 für das Jahr 2024, für die Jahre davor entspricht er jeweils dem 30. April zwei Jahre später (t+2).

Ausgangspunkt ist das EO-Register (`apg_brut.apg_cpt`), indem alle Auszahlungen separat als Einträge enthalten sind, d.h. möglicherweise mehrere Einträge pro Person und Leistung. Dies ist insbesondere beim Mutterschaftsurlaub und Betreuungsurlaub der Fall, deren Leistungen i.d.R. in mehreren Tranchen ausbezahlt werden. Darüber hinaus gibt es mehrere Einträge pro Person und Leistung, wenn Personen mehrere Arbeitgeber haben. In einem ersten Schritt wir das Register auf die relevante Zeitperiode eingeschränkt. Einträge, die zum gleichen Leistungsanspruch gehören, werden über eine Business process ID verknüpft. Für Personen mit mehreren Arbeitgebern wird eine Korrektur in der Anzahl der entschädigten Tage vorgenommen, die zum gleichen Leistungsanspruch gehören, damit sie nicht mehrfach gezählt werden. Im Anschluss wird ein Datensatz erstellt, der nur einen Eintrag pro Leistungsbeziehende und Geburt bzw. Kind enthält. Entschädigte Tage und Leistungen, Tagesansätze und der vergütete Gesamtbetrag werden dafür aufsummiert.

Um sicherzustellen, dass Einträge für dieselbe Person über die Zeit richtig verlinkt werden können, verknüpfen wir das EO-Register mit dem Versichertenregister und nehmen, wo nötig Korrekturen in den AHV-Nummern der Eltern und Kinder sowie deren Geburtsdaten vor.

Für den Mutterschafts- und Vaterschaftsurlaub 2024 werden die Anzahl Leistungsbeziehenden, entschädigte Tage und der vergütete Gesamtbetrag mit einem Hochrechnungsfaktor korrigiert, weil bis zum Zeitpunkt der Auswertung (Stand Ende September des Folgejahres) noch nicht alle Leistungsansprüche vollständig im Register abgebildet sind. Dies betrifft nur Urlaube für Geburten nach Juni 2024 und Monate für die zu tiefe Geburten im EO-Register im Vergleich zur Statistik der natürlichen Bevölkerungsbewegung (BEVNAT) gezählt wurden.

Pro Person und Geburt/Kind berechnen wir anschliessend die Entschädigung pro Tag als Entschädigung für den gesamten Leistungszeitraum dividiert durch die Anzahl der entschädigten Tage.

2.6.2.1 Mutterschafts- und Vaterschaftsurlaub Für den Mutterschafts- und Vaterschaftsurlaub werden die Daten nach Geburtsjahr, Leistungsart, Domizil (Schweiz und Ausland) und Alterskategorie aggregiert. Wir verwenden dieselben 7 Altersgruppen (`kat_alt`), wie bei den Dienstleistenden. Wir berücksichtigen keine Leistungen von Müttern und Vätern, die jünger als 17 sind, da diese in der Regel keinen Anspruch auf eine EO-Leistung haben.

Wir berechnen für diese Gruppen folgende Input-Variablen:

- Anzahl der Leistungsbeziehenden mit EO-Max (`mutterschaftsurlaub_eomax`/ \ `vaterschaftsurlaub_eomax`): als Summe der Leistungsbeziehenden mit maximaler Entschädigung pro Tag.
- Anzahl der Leistungsbeziehenden unter EO-Max (`mutterschaftsurlaub_ohne_eomax`/ \ `vaterschaftsurlaub_ohne_e`
  als Summe der Leistungsbeziehenden mit einem Taggeld unter der maximalen Entschädigung pro Tag.
- Durchschnittliches Taggeld für Beziehende mit einer Entschädigung unter dem Maximum pro Tag (`taggeld`): als durchschnittliche Grundentschädigung pro Tag, gemittelt über alle Leistungsbeziehenden, deren Entschädigung unterhalb der maximalen Grundentschädigung liegt.
- Durchschnittliche Dauer der Leistung in Tagen (`tage`): als Durchschnitt über die bezogenen Tage aller Leistungsbeziehenden

Die Daten betreffen die folgenden Input-Files: `REGISTER_MSU`, `REGISTER_VSU`

2.6.2.2 Betreuungs- und Adoptionsurlaub Für den Betreuungsurlaub berechnen wir folgende Input-Variablen pro Jahr des Beginns des Leistungsanspruchs. Für den Adoptionsurlaub berechnen wir die Variablen pro Jahr der Adoption:

- Anzahl der Leistungsbeziehenden (`anzahl_urlaube`): als Summe der Leistungsbeziehenden
- Durchschnittliche Dauer der Leistung in Tagen (`tage`): als Durchschnitt über die bezogenen Tage aller Leistungsbeziehenden
- Anteil der von Frauen bezogenen Tage (`anteil_muetter`): als Anteil der von Frauen bezogenen Tagen an der Summe der bezogenen Tage aller Leistungsbeziehenden.
- Ausbezahlte Leistungen (`ausgaben`): als Summe der ausbezahlten Leistungen

Die Daten betreffen die folgenden Input-Files: `REGISTER_BSU`, `REGISTER_ADOPT`

2.7 Daten gemäss Abrechnungsjahr

Aus dem Register können auch Teile der Buchhaltung der EO repliziert werden, da für jede Auszahlung das Abrechnungsdatum erfasst ist. Allerdings sind nur die Leistungen und Rückerstattungen erfasst. Während in den Buchhaltungsdaten (EO-Abrechnung) keine Aufteilung nach Leistungsart möglich ist, können die Leistungen und Rückerstattungen so nach Leistungsart aufgeteilt werden. Den Daten gemäss Abrechnungsjahr entnehmen wir Angaben zu verbuchte EO-Leistungen in CHF pro gruppierte Leistungsart und Jahr.

Wir unterscheiden zwischen Dienst, Mutterschaftsurlaub, Urlaub des anderen Elternteils, Betreuungsurlaub und Adoptionsurlaub. Wir berechnen die Anteile der Ausgaben für die einzelnen Leistungsarten an den Gesamtausgaben (`anteil_dienst`, `anteil_msu`, `anteil_vsu`, `anteil_bsu`, `anteil_adopt`).

Die Daten betreffen die folgenden Input-Files: `EO_AUSGABEN_ANTEILE`

2.8 Daten aus der EO Abrechnung

Wir verwenden die Abrechnungsdaten der EO gemäss der von Compenswiss publizierten Jahresrechnung (Bilanz und der Erfolgsrechnung der EO) um allgemein anfallende Kosten zu schätzen. Diese stehen nach Abrechnungsjahr gemeinsam für alle Versicherungen zur Verfügung.[14] Es handelt sich um den Verwaltungsaufwand des EO-Fonds (`verw_kost`), die Abschreibungen von Rueckerstattungsforderungen (`abschr_rueckf`), den Beitragsanteil zulasten der EO (btr_ant_eo) und die Summe der Leistungen (`geld_leist`).

Die Daten betreffen die folgenden Input-Files: `EO_ABRECHNUNG_DEF`

3 Module zur Aufbereitung der Input-Daten

In diesem Kapitel werden die Module beschrieben, mithilfe welcher die Input-Daten für das Finanzperspektivenmodell EO eingelesen und aufbereitet werden. Die Datenaufbereitung für die Finanzperspektivenmodelle sämtlicher durch das BSV modellierten Sozialversicherungen (AHV, IV, EO, EL) erfolgt über ein gemeinsames Modul `prepare_input.R`. Dies ist dadurch begründet, dass vielfach die gleichen Input-Daten von allen Sozialversicherungen genutzt werden (bspw. Bevölkerungsstatistiken und -szenarien des BFS).[15]

3.1 Modul prepare_input.R

Dokumentation zuletzt aktualisiert am 14.07.2025

Das Hauptmodul zur Aufbereitung der Input-Daten für das Finanzperspektivenmodell der EO wird wie folgt aufgerufen:

```
path = "C:/delfinverse/data/PARAM_INPUTS.csv"
prepare_input(path)
```

`path` ist der Pfad zur `PARAM_INPUTS.csv` Datei, welche für die Berechnungen verwendet werden soll. `PARAM_INPUTS.csv` enthält Dateipfade zu den zu verwendenden Rohdaten (vgl. Kapitel 1.2.1). Bevor der obige Code ausgeführt werden kann, muss natürlich sichergestellt werden, dass der Ordner `.../data` existiert, und die Datei `PARAM_INPUTS.csv` enthält. Zudem darf der Ordner `.../data` ausser `PARAM_INPUTS.csv` vor dem Ausführen von `prepare_input` keine anderen Ordner oder Dateien enthalten.

Das Modul `prepare_input` enthält die folgenden Elemente:

```
function(path, path_out = file.path(dirname(path)),
                          overwrite = FALSE)
```

Das Modul nimmt den vorangehend spezifizierten Pfad `path` entgegen, und spezifiziert den Ordner auf welchen `path` verweist als `path_out`, also den Ordner, in welchen die aufbereiteten Daten am Ende des Moduls ausgelesen werden sollen. Zudem wird die Variable `overwrite` auf `FALSE` gesetzt, um zu verhindern, dass falls der Ordner `.../data` bereits aufbereitete Daten enthält, diese überschrieben werden.

Danach werden die Input-Parameter aus `PARAM_INPUTS.csv` eingelesen, und im Dataframe `PARAM_INPUTS` abgelegt:

[14] https://ar.compenswiss.ch/de_CH/konten/eo/erfolgsrechnung-der-eo

[15] In diesem Kapitel werden nur Untermodule von `prepare_input.R` beschrieben, welche Daten einlesen, die auch im Finanzperspektivenmodell EO verwendet werden. Die Module, welche Daten einlesen, die ausschliesslich von anderen Sozialversicherungen als der EO verwendet werden, werden hier nicht beschrieben.

```
# Parameter-File einlesen
PARAM_INPUTS <- read_param(path)
```

Es wird geprüft, auf welchem Betriebssystem das Skript läuft (z.B. lokal unter Windows oder in einem Linux-Container in der Cloud), basierend auf dem Arbeitsverzeichnis: Beginnt der Pfad nicht mit einem Buchstaben, wird davon ausgegangen, dass es sich um ein Linux-System handelt, und der Pfad zu den Rohdaten wird entsprechend gesetzt.

Dadurch kann `prepare_input` sowohl lokal als auch in der Workbench ausgeführt werden.

```
# Check the working directory and set the path based on the
# environment
if (!grepl("^[A-Za-z]", getwd())) {
    # If the working directory doesn't start with a letter,
    # assume Linux
    PARAM_INPUTS$raw_data <- "/data/appl-wb/01_raw_data"
}
```

Anschließend werden die Verzeichnispfade festgelegt, aus denen das Modul später die Roh-Input-Daten lädt.

```
# Pfade zu Input-Daten definieren
PARAM_INPUTS$path_allgemein_go <- paste0(PARAM_INPUTS$raw_data,
    "/allgemein/go/")
PARAM_INPUTS$path_allgemein_massnahmen <- paste0(PARAM_INPUTS$raw_data,
    "/allgemein/massnahmen/")

PARAM_INPUTS$path_ahv_go <- paste0(PARAM_INPUTS$raw_data, "/ahv/go/")
PARAM_INPUTS$path_ahv_massnahmen <- paste0(PARAM_INPUTS$raw_data,
    "/ahv/massnahmen/")
PARAM_INPUTS$path_avh_hila <- paste0(PARAM_INPUTS$raw_data, "/ahv/massnahmen/hila")

PARAM_INPUTS$path_iv_go <- paste0(PARAM_INPUTS$raw_data, "/iv/go/")
PARAM_INPUTS$path_iv_massnahmen <- paste0(PARAM_INPUTS$raw_data,
    "/iv/massnahmen/")

PARAM_INPUTS$path_eo_go <- paste0(PARAM_INPUTS$raw_data, "/eo/go/")
PARAM_INPUTS$path_eo_massnahmen <- paste0(PARAM_INPUTS$raw_data,
    "/eo/massnahmen/")

PARAM_INPUTS$path_el_go <- paste0(PARAM_INPUTS$raw_data, "/el/go/")
PARAM_INPUTS$path_el_massnahmen <- paste0(PARAM_INPUTS$raw_data,
    "/el/massnahmen/")

PARAM_INPUTS$path_ul_go <- paste0(PARAM_INPUTS$raw_data, "/ul/go/")
PARAM_INPUTS$path_ul_massnahmen <- paste0(PARAM_INPUTS$raw_data,
    "/ul/massnahmen/")

PARAM_INPUTS$path_rententab <- paste0(PARAM_INPUTS$raw_data,
    "/rententab")

PARAM_INPUTS$path_beitragstab <- paste0(PARAM_INPUTS$raw_data,
    "/beitragstab")

PARAM_INPUTS$path_eotab <- paste0(PARAM_INPUTS$raw_data, "/eotab")
```

Als nächstes werden die Pfade spezifiziert, unter welchen die aufbereiteten Input-Daten später im Modul

abgelegt werden sollen:

```
# Input-Ordner vorbereiten
path_out = file.path(dirname(path))

inp_path_allgemein_go <- file.path(path_out, "allgemein", "go")
inp_path_allgemein_massnahmen <- file.path(path_out, "allgemein",
    "massnahmen")

inp_path_ahv_go <- file.path(path_out, "ahv", "go")
inp_path_ahv_massnahmen <- file.path(path_out, "ahv", "massnahmen")

inp_path_iv_go <- file.path(path_out, "iv", "go")
inp_path_iv_massnahmen <- file.path(path_out, "iv", "massnahmen")

inp_path_eo_go <- file.path(path_out, "eo", "go")
inp_path_eo_massnahmen <- file.path(path_out, "eo", "massnahmen")

inp_path_el_go <- file.path(path_out, "el", "go")
inp_path_el_massnahmen <- file.path(path_out, "el", "massnahmen")

inp_path_ul_go <- file.path(path_out, "ul", "go")
inp_path_ul_massnahmen <- file.path(path_out, "ul", "massnahmen")

inp_path_rententab <- file.path(path_out, "rententab")

inp_path_beitragstab <- file.path(path_out, "beitragstab")

inp_path_eotab <- file.path(path_out, "eotab")
```

Die Funktion `file.path(...)` erzeugt Pfade in einem systemkonformen Format, das automatisch die passenden Trennzeichen für Linux, Windows oder macOS verwendet.

Es wird später für jede Sozialversicherung ein separater Ordner für die aufbereiteten Input-Daten erstellt. `path_eo_go` enthält beispielsweise den Pfad zum Ordner, in welchem Input-Daten für die Berechnungen der EO-Finanzperspektiven später abgespeichert werden sollen, und `path_eo_massnahmen` den Pfad zum Ordner, in welchem die Input-Daten für die Berechnung der Politikmassnahmen für EO abgespeichert werden sollen. Es gibt Input-Daten, welche für mehrere Finanzperspektivenmodelle verwendet werden, beispielsweise die Bevölkerungszahlen und -szenarien, oder die Projektionen zur Lohn- und Preisentwicklung. Um Duplikate zu vermeiden werden diese Input-Daten ausschliesslich im Ordner `path_allgemein_go` abegelegt.

Der Nachfolgende Code-Block stellt sicher, dass die Verschiedenen Ordner mit den Input-Daten nicht schon im Ordner `.../data` enthalten sind. Es handelt sich hierbei um ein Sicherheitscheck, der verhindern soll, dass bestehende Input-Daten versehentlich überschrieben werden:

```
ensure_path <- function(path) {
    # do not allow overwriting
    if (file.exists(path))
        stop(path, "already exists")
    if (!file.exists(path)) {
        dir.create(path, recursive = TRUE)
    }
}

ensure_path(inp_path_allgemein_go)
ensure_path(inp_path_allgemein_massnahmen)
```

```
ensure_path(inp_path_ahv_go)
ensure_path(inp_path_ahv_massnahmen)

ensure_path(inp_path_iv_go)
ensure_path(inp_path_iv_massnahmen)

ensure_path(inp_path_eo_go)
ensure_path(inp_path_eo_massnahmen)

ensure_path(inp_path_el_go)
ensure_path(inp_path_el_massnahmen)

ensure_path(inp_path_ul_go)
ensure_path(inp_path_ul_massnahmen)

ensure_path(inp_path_rententab)

ensure_path(inp_path_beitragstab)

ensure_path(inp_path_eotab)
```

Als nächstes werden die verschiedenen Rohdatenfiles eingelesen und aufbereitet. Die Funktionsweise dieser repetitiven Code-Elemente wird nachfolgend anhand der EO-spezifischen Rohdaten erläutert:

```
### EO: GELTENDE ORDNUNG ###

# Input EO Abrechnungsdaten
tl_input_eo_abrechnung <- mod_input_eo_abrechnung(PARAM_INPUTS = PARAM_INPUTS)

# Input Naissances, Population active EO
tl_mod_input_eo_ofs_hist <- mod_input_eo_ofs_hist(PARAM_INPUTS = PARAM_INPUTS)

# Input EO Register
tl_mod_input_register <- mod_input_eo_register(PARAM_INPUTS = PARAM_INPUTS)

# Input EOMAX
EOMAX_hist <- read.csv(file.path(PARAM_INPUTS$path_eo_go, PARAM_INPUTS$file_eomax),
    sep = ";")


eo_go <- c(tl_input_eo_abrechnung, tl_mod_input_eo_ofs_hist,
    tl_mod_input_register, list(EOMAX_hist = EOMAX_hist))

### EO: MASSNAHMEN ###

# Input der in xls berechneten EO-Massnahmen
eo_massnahmen <- mod_input_eo_massnahmen(PARAM_INPUTS = PARAM_INPUTS)
```

D.h. es werden nacheinander die verschiedenen für die Berechnung der EO-Finanzperspektiven relevanten Rohdaten eingelesen, wobei für jede Art von Rohdaten ein spezifisches Modul verwendet wird. Die EO-spezifischen Rohdaten für die Berechnung der EO-Finanzperspektiven nach geltender Ordnung werden danach in der Liste `eo_go` zusammengefasst. Ebenso werden die IV-spezifischen Rohdaten für die Berechnung der Massnahmen in der Liste `eo_massnahmen` zusammengefasst. Die einzelnen Module für das Einlesen der Rohdaten werden in den nachfolgenden Abschnitten dieses Kapitels erläutert.

Am Ende des Moduls `prepare_input.R` werden die eingelesenen und aufbereiteten Daten in den oben spezifizierten Ordnern abgelegt. Am Beispiel der Input-Daten der EO sieht der betreffende Codeteil wie folgt aus:

```
tidylist_write(eo_go, inp_path_eo_go)
tidylist_write(eo_massnahmen, inp_path_eo_massnahmen)
```

Die Funktion `tidy list_write` nimmt die in der tidy list `eo_go` spezifizierten Dataframes, und schreibt diese im .csv-Format in den Ordner `inp_path_eo_go` (siehe oben). Dasselbe geschieht mit den in `eo_massnahmen` spezifizierten Dataframes.

3.2 Modul mod_input_bev_bestand.R

Dokumentation zuletzt aktualisiert am 19.06.2025

Das Modul `mod_input_bev_bestand.R` liest die Daten für die Bevölkerungsbestände des BFS ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
BEV_BESTAND <- mod_input_bev_bestand(PARAM_INPUTS = PARAM_INPUTS)
```

Als erstes werden in`mod_input_bev_bestand.R` die verschiedenen Rohdateien mit den Bevölkerungsdaten eingelesen:

```
    #---------------------------------LOAD RData FILES (ESPOP, ERWBEV)-

    # Funktion zum Einlesen der RData-Dateien
    loadRData <- function(fileName) {
        temp_env <- new.env()
        load(fileName, envir = temp_env)
        temp_env[[ls(temp_env)[1]]]
    }

    # Lade die benötigten RData-Dateien
    ESPOP <- loadRData(file.path(PARAM_INPUTS$path_allgemein_go, "BEVOELKERUNG",
↪   PARAM_INPUTS$name_espop)) |>
      pivot_wider(
        names_from = variable,
        values_from = value
      ) |>
      arrange(jahr, sex, nat, alt)

    BEV_INLAENDER_EPT <- loadRData(file.path(PARAM_INPUTS$path_allgemein_go,
↪   "BEVOELKERUNG", PARAM_INPUTS$erwbev_bestand)) |>
      mutate(
# Berechnung von Erwerbsquote und Erwerbsquote in VZÄ, da diese auch für die
↪   Szenariodaten verfügbar sind
        erwq=erwbev/erwbev_and_nerwbev*100,
        erwqept=ept/erwbev_and_nerwbev*100
        ) |>
      select(-erwbev_and_nerwbev) # Variable mit Summe von Erwerbs- und
       ↪   Nichterwerbsbevölkerung gemäss SAKE entfernen (da nicht in Szenariodaten
       ↪   enthalten)
```

Die Funktion `loadRData` stellt sicher, dass die Daten im R-Format korrekt eingelesen werden. Darauffolgend werden nacheinander die Daten zu Beständen und Flüssen (bspw. Ein- und Auswanderung, Geburten, Todesfälle) der Wohnbevölkerung (Dataframe ESPOP) sowie zum Bestand der Inländer-Erwerbsbevölkerung

(BEV_INLAENDER_EPT) eingelesen.[16]

Als nächstes werden die Daten zu den Grenzgänger-Beständen eingelesen, wobei die Komplexität des Codes einzig daher rührt, dass die Rohdaten aus dem von uns gelieferten .xlsx-Format eingelesen werden müssen:

```
#------------------------------FRONTALIERS DATA-------------------------

# Funktion zum Einlesen der Sheets 'Männer' und 'Frauen'
process_sheet <- function(sheet_name, sex_label) {
    # Datei einlesen mit automatischer Vergabe eindeutiger
    # Spaltennamen
    data <- read_excel(file.path(PARAM_INPUTS$path_allgemein_go,
        "BEVOELKERUNG", PARAM_INPUTS$file_frontaliers), sheet = sheet_name,
        col_names = FALSE, .name_repair = "unique_quiet")

    # Spalte B entfernen (zweite Spalte)
    data <- data |>
        select(-2)

    # Spaltennamen setzen basierend auf der zweiten Zeile
    colnames(data) <- c("jahr", as.character(as.numeric(data[2,
        -c(1, ncol(data))])), "99")

    # Entferne erste und dritte Zeile
    data <- data[-c(1, 2, 3), ]

    # In tibble umwandeln und auf lange Form bringen
    data |>
        as_tibble() |>
        pivot_longer(cols = -jahr, names_to = "alt", values_to = "frontaliers") |>
        mutate(jahr = as.numeric(jahr), frontaliers = as.numeric(frontaliers),
            alt = as.numeric(alt), sex = sex_label, nat = "au")
}

# Daten aus beiden Sheets einlesen
FRONTALIERS <- bind_rows(process_sheet("Männer", "m"), process_sheet("Frauen",
    "f"))
```

Als nächstes werden die Daten zu den Saisonnier-Beständen ab 1971 eingelesen, wobei die Komplexität des Codes wiederum einzig daher rührt, dass die Rohdaten aus dem von uns gelieferten .xlsx-Format eingelesen werden müssen:

```
    #------------------------------SAISONNIERS DATA-------------------------

    # Lade und verarbeite die Saisonniers-Daten
    SAISONNIERS <- read_excel(
        file.path(PARAM_INPUTS$path_allgemein_go, "BEVOELKERUNG", PARAM_INPUTS$file_sm),
        sheet = "bestaende",
        skip = 10
    ) %>%
        as_tibble() %>%
        pivot_longer(
            cols = -jahr, # Alle Spalten außer "jahr"
```

[16]Bis 2010 wurden die Bestände und Flüsse der Wohnbevölkerung in der Synthesestatistik von Stand und Struktur der Bevölkerung (ESOPOP) abgebildet, ab 2010 in der Statistik der Bevölkerung und der Haushalte (STATPOP). Das BFS stellt die Daten aus den beiden Statistiken kombiniert bereit.

```
            names_to = "alt", # Alters-Spalte erstellen
            values_to = "saisonniers" # Werte in "saisonniers" speichern
        ) %>%
        mutate(
            alt = as.integer(gsub("age", "", alt, fixed = TRUE)), # "age" aus
            ↪  Alterswerten entfernen
            jahr = as.integer(jahr), # Jahr numerisch formatieren
            nat = "au" # Nationalität hinzufügen
        ) %>%
        crossing(sex = c("m", "f")) %>% # Geschlechter-Kombination
        filter(jahr >= 1971, alt <= 99) # Filtere relevante Jahre und Alter
```

Als nächstes werden die Daten zu den Beständen an freiwillig Versicherten eingelesen, wobei die Komplexität des Codes wiederum einzig daher rührt, dass die Rohdaten aus dem von uns gelieferten .xlsx-Format eingelesen werden müssen:

```
    #--------------------------------ASSURES FACULTATIFS DATA------------------

    # Importiere die Assures-Facultatifs-Daten
    tfile <- file.path(PARAM_INPUTS$path_allgemein_go, "BEVOELKERUNG",
↪   PARAM_INPUTS$file_af)

    # Bestimme relevante Sheets (Frauen und Männer)
    sheets_names_sel <- readxl::excel_sheets(tfile) %>%
        grep("^(f_|h_)", ., value = TRUE)

    # Funktion zum Lesen und Umformen von Excel-Daten
    af_fct_data <- function(tfile, sheet) {
        suppressMessages( # Suppress "New names" message
            read_excel(tfile, sheet, skip = 1) %>%
                as_tibble() %>%
                select(-TOTAL) %>%  # Entferne TOTAL-Spalte
                rename(jahr = "...1") %>%  # Benenne erste Spalte um
                pivot_longer(-jahr, names_to = "alt", values_to = "assures_facultatifs")
                ↪  %>%
                mutate(
                    alt = as.integer(alt),  # Wandle alt in Integer um
                    jahr = as.integer(jahr)  # Wandle jahr in Integer um
                ) %>%
                filter(alt <= 99) %>%  # Filtere Alterswerte bis 99
                arrange(alt)  # Sortiere nach Alter
        )
    }

    # Verarbeite und kombiniere die Assures-Facultatifs-Daten
    ASSURES_FACULTATIFS <- tibble(sheet = sheets_names_sel) %>%
        mutate(
            sex = ifelse(grepl("^f_", sheet), "f", "m"),  # Geschlecht aus Sheetname
            ↪  ableiten
            nat = ifelse(grepl("_ch", sheet), "ch", "au")  # Nationalität aus Sheetname
            ↪  ableiten
        ) %>%
        mutate(data = purrr::map(sheet, ~ af_fct_data(tfile, .x))) %>%  # Daten einlesen
        unnest(data) %>%  # Daten entpacken
        select(jahr, sex, nat, alt, assures_facultatifs)  # Spalten auswählen
```

Der nachfolgende Codeblock stellt sicher, dass für alle Variable (Variablen sind beispielsweise Grenzgänger, Saisonniers[17], Wohnbevölkerung), welche für ein gegebenes Jahr vorhanden sind, Daten für jedes Alter zwischen 0 und 99, Geschlecht, und jede Nationalität (Schweizer oder Ausländer) vorhanden sind. Wenn für ein Jahr, für welches Daten für die gegebene Variable vorhanden sind, in einer Alter-Geschlecht-Nationalität Zelle keine Werte geliefert wurden, wird dieser Wert auf 0 gesetzt. Dies stellt später sicher, dass später nicht-vorhandene Daten von 0-Werten unterscheiden werden können. Zusätzlich werden in nachfolgendem Codeblock auch alle Alter über 99 aggregiert und der Altersklasse 99 zugerechnet.

```
#---------------------------------ENSURE ALT RANGE 0-99---------------------

# Funktion zur Sicherstellung vollständiger Daten für alt von 0 bis 99 für jahre wo
↪  Daten vorhanden sind
ensure_alt_range <- function(df, value_cols) {
    # Ensure value_cols is treated as a vector
    value_cols <- as.vector(value_cols)

    # Create a complete grid with all combinations of jahr, sex, nat, and alt
    complete_grid <- expand_grid(
        jahr = unique(df$jahr),  # All unique years in the dataset
        sex = c("m", "f"),       # Both sexes
        nat = c("ch", "au"),     # Both nationalities
        alt = 0:99               # Age group from 0 to 99
    )

    # Summarize and complete the data
    df %>%
        mutate(alt = ifelse(alt >= 100, 99, alt)) %>% # Summiere alt >= 100 in alt ==
        ↪  99
        group_by(jahr, sex, nat, alt) %>%
        summarize(across(all_of(value_cols), \(x) sum(x, na.rm = TRUE)), .groups =
        ↪  "drop") %>% # Summiere Werte
        right_join(complete_grid, by = c("jahr", "sex", "nat", "alt")) %>% # Merge
        ↪  with complete grid
        mutate(across(all_of(value_cols), ~ replace_na(., 0))) # Replace NA with 0 in
        ↪  value_cols
}

# Wende die Normalisierung auf alle relevanten DataFrames an
ESPOP <- ensure_alt_range(ESPOP,
                  c("bevanfangj","bevendejahr","geburt","tod","einbuergerung",
                    "einwanderung","auswanderung","bereinigung"))

BEV_INLAENDER_EPT <- ensure_alt_range(BEV_INLAENDER_EPT, c("erwbev","ept", "erwq",
↪  "erwqept"))
FRONTALIERS <- ensure_alt_range(FRONTALIERS, "frontaliers")
SAISONNIERS <- ensure_alt_range(SAISONNIERS, "saisonniers")
ASSURES_FACULTATIFS <- ensure_alt_range(ASSURES_FACULTATIFS, "assures_facultatifs")
```

Der Nachfolgende Codeblock dient dazu, Fehler im Register für die freiwillig Versicherte[18] zu korrigieren. Konkret fehlen in den Jahren 2002 und 2004 Beobachtungen im Register. Daher werden die Werte für 2002 und 2004 aus den aufgerundeten Mittelwerten der Werte in der jeweiligen Geschlecht-Nationalität-Alter Zelle

[17] Der 'SAISONNIERS'-Dataframe ist für die AHV-Finanzperspektiven relevant, jedoch nicht für die der EO.
[18] Die freiwilig Versicherte sinf für die AHV Finanzperspektiven relevant, jedoch nicht für die der EO.

der angrenzenden Jahre (bspw. 2001 und 2003 für 2002) gebildet:

```
#---------------KORREKTUR REGISTERFEHLER ASSURES_FACULTATIFS----------------
# Correction des erreurs d'observations des registres pour
# les années 2002 et 2004 (non reported observations)

ASSURES_FACULTATIFS <- ASSURES_FACULTATIFS %>%
    group_by(sex, nat, alt) %>%
    # Pour 2002 et 2004, remplacer les valeurs par la
    # moyenne des observations de l'année précédente et
    # l'année suivante.
mutate(assures_facultatifs = if_else(jahr %in% c(2002, 2004),
    ceiling((lag(assures_facultatifs) + lead(assures_facultatifs))/2),
    assures_facultatifs)) %>%
    ungroup()
```

Zum Schluss werden alle Daten in einem Dataframe `BEV_BESTAND` zusammengefasst. Hierbei wird für die Jahre bis 2009 ESPOP für die Wohnbevölekrung und die Auswanderung verwendet, und ab 2010 STATPOP:

```
    #---------------------------------COMBINE DATA-----------------------------

    # Kombiniere die verschiedenen Datenquellen
    BEV_BESTAND <- ESPOP %>%
      full_join(BEV_INLAENDER_EPT, by = c("jahr", "sex", "nat", "alt")) %>%
      full_join(FRONTALIERS, by = c("jahr", "sex", "nat", "alt")) %>%
      left_join(SAISONNIERS, by = c("jahr", "sex", "nat", "alt")) %>% # Left-join für
      ↪  Saisonniers, da diese in Rohdaten bis 2065 auf 0 sind
      full_join(ASSURES_FACULTATIFS, by = c("jahr", "sex", "nat", "alt")) %>%
      arrange(jahr, sex, nat, alt)

    return(BEV_BESTAND = BEV_BESTAND)
```

3.3 Modul mod_input_bev_scenario.R

Dokumentation zuletzt aktualisiert am 19.06.2025

Das Modul `mod_input_bev_scenario.R` liest, analog zum Modul `mod_input_bev_bestand.R` für die Bevölkerungsbestände (vgl. Kapitel 3.2), die Daten für die Bevölkerungsszenarien des BFS ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
BEV_SCENARIO <- mod_input_bev_scenario(PARAM_INPUTS = PARAM_INPUTS)
```

Als erstes werden in`mod_input_bev_scenario.R` die verschiedenen Rohdateien mit den Bevölkerungsszenarien eingelesen:

```
    #---------------------------------LOAD RData FILES (ESPOP, STATPOP, ERWBEV)-

    # Funktion zum Einlesen der RData-Dateien
    loadRData <- function(fileName) {
        temp_env <- new.env()
        load(fileName, envir = temp_env)
        temp_env[[ls(temp_env)[1]]]
    }

    # Load Scenarios population
    SCENARIO_POP <- loadRData(file.path(
      PARAM_INPUTS$path_allgemein_go,
```

```
    "BEVOELKERUNG",
    PARAM_INPUTS$name_scenario_pop
  )) %>%
    filter(str_detect(scenario, "^(A-00|B-00|C-00)")) %>%
    mutate(scenario = str_replace_all(scenario, "-", "_")) %>% # Replace - with _
    select(-geburtnachnat)

  # Load Scenarios ept
  SCENARIO_EPT <- loadRData(file.path(
    PARAM_INPUTS$path_allgemein_go,
    "BEVOELKERUNG",
    PARAM_INPUTS$name_scenario_ept
  )) %>%
    filter(str_detect(scenario, "^(A-00|B-00|C-00)")) %>%
    mutate(scenario = str_replace_all(scenario, "-", "_")) %>% # Replace - with _
    rename(ept = erwbevept) %>%
    select(alt, jahr, nat, sex, scenario, erwbev, ept)
```

Die Funktion `loadRData` stellt sicher, dass die Daten im R-Format korrekt eingelesen werden. Darauffolgend werden nacheinander die Daten den Bevölkerungsszenarien (SCENARIO_POP)[19], sowie die Erwerbsbevölkerungsszenarien (SCENARIO_EPT)[20] eingelesen.

Als nächstes werden die Daten zu den Grenzgänger-Szenarien eingelesen:

```
SCENARIO_FRONTALIERS <- loadRData(file.path(PARAM_INPUTS$path_allgemein_go,
    "BEVOELKERUNG", PARAM_INPUTS$name_scenario_frontaliers)) %>%
    filter(str_detect(scenario, "^(A_00|B_00|C_00)"))
```

Der nachfolgende Codeblock stellt sicher, dass für alle Variable (Variablen sind beispielsweise Grenzgänger, Erwerbsbevölkerung, Wohnbevölkerung), welche für ein gegebenes Jahr vorhanden sind, Daten für jedes Alter zwischen 0 und 120, Geschlecht, und jede Nationalität (Schweizer oder Ausländer) vorhanden sind. Wenn für ein Jahr, für welches Daten für die gegebene Variable vorhanden sind, in einer Alter-Geschlecht-Nationalität Zelle keine Werte geliefert wurden, wird dieser Wert auf 0 gesetzt. Dies stellt später sicher, dass später nicht-vorhandene Daten von 0-Werten unterscheiden werden können. Zusätzlich werden in nachfolgendem Codeblock auch alle Alter über 120 aggregiert und der Altersklasse 120 zugerechnet.

```
  #---------------------------------ENSURE ALT RANGE 0-120---------------------

  # Funktion zur Sicherstellung vollständiger Daten für alt von 0 bis 120 für jahre wo
  ↪  Daten vorhanden sind
  ensure_alt_range <- function(df, value_cols) {

    # Create a complete grid with all combinations of existing jahr, sex, nat, alt, and
    ↪  scenario
    complete_grid <- df %>%
      select(jahr, scenario) %>%
      distinct() %>%
      expand_grid(
        sex = c("m", "f"),
        nat = c("ch", "au"),
        alt = 0:120
      )
```

[19] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/szenarien.html

[20] https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/erwerbstaetigkeit-arbeitszeit/erwerbsbevoelkerung/kuenftige-entwicklung-erwerbsbevoelkerung.html

```
    # Summarize, complete, and drop rows with all value_cols as NA
    df %>%
      mutate(alt = ifelse(alt >= 121, 120, alt)) %>%  # Gruppiere alt >= 121 zu 120
      group_by(jahr, sex, nat, alt, scenario) %>%
      summarize(across(all_of(value_cols), \(x) sum(x, na.rm = TRUE)), .groups =
      ↪  "drop") %>%
      right_join(complete_grid, by = c("jahr", "scenario", "sex", "nat", "alt")) %>%  #
      ↪  Ergänze fehlende Kombinationen innerhalb jedes scenario
      mutate(across(all_of(value_cols), \(x) replace_na(x, 0)))  # Ersetze NA-Werte
      ↪  durch 0
  }

  # Wende die Normalisierung auf alle relevanten DataFrames an
  SCENARIO_POP <- ensure_alt_range(SCENARIO_POP,
↪ c("bevanfangj","bevendejahr","geburt","tod","einbuergerung","einwanderung","auswanderung","geburtmu
  SCENARIO_EPT <- ensure_alt_range(SCENARIO_EPT, c("erwbev","ept","erwq","erwqept"))
  SCENARIO_FRONTALIERS <- ensure_alt_range(SCENARIO_FRONTALIERS, "frontaliers")
```

Zum Schluss werden alle Daten in einem Dataframe `SCENARIO_POP` zusammengefasst:

```
#--------------------------------COMBINE DATA-----------------------------

# Kombiniere die verschiedenen Datenquellen
BEV_SCENARIO <- SCENARIO_POP %>%
    full_join(SCENARIO_EPT, by = c("jahr", "sex", "nat", "alt",
        "scenario")) %>%
    full_join(SCENARIO_FRONTALIERS, by = c("jahr", "sex", "nat",
        "alt", "scenario")) %>%
    arrange(jahr, sex, nat, alt, scenario)

return(BEV_SCENARIO = BEV_SCENARIO)
```

3.4 Modul mod_input_estv.R

Dokumentation zuletzt aktualisiert am 19.06.2024

Das Modul `mod_input_estv.R` liest die Daten der individuellen Konten der ZAS (IK) ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
IK <- mod_input_ikregister(PARAM_INPUTS = PARAM_INPUTS)
```

Als Erstes wird die in `PARAM_INPUTS` spezifizierte Zeitspanne eingelesen, über welche die IK-Daten verfügbar sind:

```
# --- Loop Jahre 1997 bis zum mit jahr_ik gewuenschten Jahr
# ------------------#

all_years <- 1997:PARAM_INPUTS$jahr_ik
```

Als nächstes werden die IK-Daten eingelesen:

```
# --- Eingebettete Funktion, welche 1 Jahr IK Daten
# einliest -----------------#

read_one_year <- function(l_j) {
```

```
    ik_file <- paste0(PARAM_INPUTS$path_allgemein_go, "IK/ik_",
        l_j, PARAM_INPUTS$file_ik)
    z <- read.csv(ik_file, header = TRUE, sep = ";", stringsAsFactors = FALSE)
    as_tibble(z) %>%
        mutate(jahr = as.integer(l_j))
}

# Einlesen IK Daten
# ------------------------------------------------------------#

BEITRAG_IK <- bind_rows(lapply(all_years, read_one_year))
```

Hierfür wird zuerst die Funktion `read_one_year`, welche für ein gegebenes Jahr die CSV-Datei mit den IK-Daten dieses Jahres einliest. Danach wird die Funktion für jedes Jahr in `all_years` ausgeführt, und die Daten werden in `BEITRAG_IK` abgelegt.

Als nächstes wird die Kodierung und Benennung einiger Variablen so angepasst, dass sie mit den anderen verwendeten Daten im Finanzperspektivenmodell (bspw. den Bevölkerungsdaten und -szenarien) konsistent ist:

```
# --- Recodierung einzelner Variabler
# ---------------------------------------#

IK_TMP <- BEITRAG_IK %>%
    mutate(sex = recode(csex, `1` = "m", `2` = "f"), nat = recode(nation,
        `1` = "ch", `2` = "au")) %>%
    dplyr::select(-csex, -nation, -mcotsomZAS, -anzahlGEW) %>%
    rename(alt = alterP, personen = Anzahl, einkommen_ik = mrevsom,
        beitrag_ahvbsv = AHVCOT_BSV, beitrag_ivbsv = IVCOT_BSV,
        beitrag_eobsv = EOCOT_BSV)
# --- Return tidy df mit IK Daten
# ------------------------------------------#

IK <- IK_TMP %>%
    pivot_longer(c(beitrag_ahvbsv, beitrag_ivbsv, beitrag_eobsv),
        names_to = "vz", values_to = "beitrag_bsv") %>%
    mutate(vz = recode(vz, beitrag_ahvbsv = "ahv", beitrag_ivbsv = "iv",
        beitrag_eobsv = "eo")) %>%
    arrange(match(vz, c("ahv", "iv", "eo")))
```

Zum Abschluss werden die eingelesenen Daten wie folgt an `prepare_input.R` zurückgegeben:

```
return(IK = IK)
```

3.5 Modul mod_input_sv_beitragssatz.R

Dokumentation zuletzt aktualisiert am 19.06.2025

Das Modul `mod_input_sv_beitragssatz.R` liest die vollständige Historie der gültigen Sozialversicherungsbeitragssätze für beitragspflichtige Arbeitnehmer, Arbeitgeber, Selbständigerwerbende und Nichterwerbstätige ein. `mod_input_sv_beitragssatz.R` wird wie folgt in `prepare_input.R` aufgerufen:

```
SV_BEITRAGSSATZ <- mod_input_sv_beitragssatz(PARAM_INPUTS)
```

Das Modul besteht lediglich aus dem folgenden Code, mit welchem die Werte eingelesen und danach an `prepare_input.R` zurückgegeben werden:

```
#------ Lesen Inhalt Excel - sheet ------------------------------------#
SV_BEITRAGSSATZ <- read_excel(path = paste0(PARAM_INPUTS$path_allgemein_go,
    PARAM_INPUTS$file_sv_beitrag), sheet = PARAM_INPUTS$sheet_sv_beitrag,
    range = cell_limits(c(11, 1), c(NA, 20)), col_types = "numeric")
#------ Output -------------------------------------------------------#
return(SV_BEITRAGSSATZ = SV_BEITRAGSSATZ)
```

3.6 Modul mod_input_li.R

Dokumentation zuletzt aktualisiert am 19.06.2025

Das Modul `mod_input_li.R` liest den Lohnindex (gemäss nominalem Schweizerischen Lohnindex (SLI)). Es wird wie folgt in `prepare_input.R` aufgerufen:

```
LOHNINDEX <- mod_input_li(PARAM_INPUTS = PARAM_INPUTS)
```

Das Modul besteht lediglich aus dem folgenden Code, mit dem der in den ersten beiden Spalten des betreffenden Excel-Sheets enthaltene Indiz eingelesen und anschließend an prepare_input.R zurückgegeben wird.

```
path_lohn_index <- PARAM_INPUTS$path_allgemein_go
file_lohn_index <- PARAM_INPUTS$file_lohn_index
sheet_lohn_index <- PARAM_INPUTS$sheet_lohn_index

LOHNINDEX <- read_excel(file.path(path_lohn_index, file_lohn_index),
    sheet = sheet_lohn_index)[-(1:10), 1:2]
colnames(LOHNINDEX) <- c("jahr", "li")
LOHNINDEX <- LOHNINDEX %>%
    mutate(jahr = as.integer(jahr), li = as.numeric(li))

return(LOHNINDEX = LOHNINDEX)
```

3.7 Preisindex und Arbeitslosenquote

Anschließend werden der Preisindex (gemäß Landesindex der Konsumentenpreise, LIK) und die SECO-Arbeitslosenquote mit der Funktion `read_delim()` aus dem `readr`-Paket eingelesen. Die Ergebnisse werden als Data Frames mit den Namen `PREISINDEX` und `ARBEITSLOSENQUOTE` gespeichert.

```
# Input Preisindex
PREISINDEX <- read_delim(file.path(PARAM_INPUTS$path_allgemein_go,
    PARAM_INPUTS$file_preisindex), delim = ";", show_col_types = FALSE,
    trim_ws = TRUE)

# Input Preisindex
ARBEITSLOSENQUOTE <- read_delim(file.path(PARAM_INPUTS$path_allgemein_go,
    PARAM_INPUTS$file_arbeitslosenquote), delim = ";", show_col_types = FALSE,
    trim_ws = TRUE)
```

3.8 Modul mod_input_eckwerte.R

Dokumentation zuletzt aktualisiert am 20.06.2025

Das Modul `mod_input_eckwerte.R` liest die von der ESTV gelieferten Projektionen zur Entwicklung der Löhne und Preise ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
ECKWERTE <- mod_input_eckwerte(PARAM_INPUTS = PARAM_INPUTS)
```

Die Eckwerte werden zuerst eingelesen und alle Spalten ausser `stand`und `id` werden zu numerisch umwandeln.

```
  # --- Read file with eckwerte ------------------------------------------------

  ECKWERTE <- read_excel(
    path = paste0(PARAM_INPUTS$path_allgemein_go, PARAM_INPUTS$file_eckwerte),
    sheet = PARAM_INPUTS$sheet_eckwerte,
    range = readxl::cell_cols(1:12),
    col_types = c(rep("text", 12))
  )

  ECKWERTE <- ECKWERTE |>
    mutate(
      id = as.character(id),        # Wandle 'id' in Text um
      stand = as.character(stand), # Wandle 'stand' in Text um
      across(-c(id, stand), as.numeric) # Wandle alle anderen Spalten in numerisch um
    )
```

Anschliessend wird es überprüft, ob der Preisindex (`PREISINDEX`) aktuell genug ist. Wenn der Preisindex zu alt ist, erscheint eine Warnmeldung mit dem Hinweis, dass der INdex aktualisiert werden soll.

```
# --- Sicherstellen, dass tatsächliche
# Preisindex-Wachstumswerte verwendet werden ---- Dies ist
# insbesondere wichtig für den Bundesbeitrag im IV-Budeget,
# da dieser auf dem Wert des Preisindexes des Vorjahres
# basiert Bestimme das maximale Datum in ECKWERTE$stand und
# füge einen Monat hinzu (da Dezember-Eckwerte für Budget
# relevant)
max_stand <- max(dmy(ECKWERTE$stand))  # Konvertiere 'stand' in ein Datum
adjusted_date <- max_stand %m+% months(1)

# Bestimme das aktuelle Datum und den Tag des Monats
current_date <- Sys.Date()
current_day <- day(current_date)

# Überprüfe die Bedingung für die Ausgabe der Nachricht
if (year(adjusted_date) - 1 > max(PREISINDEX$jahr) & current_day >
    10) {
    message <- paste0("Preisindex ist Stand ", max(PREISINDEX$jahr),
        ". Stand sollte", max(PREISINDEX$jahr) + 1, " sein, bitte aktualisieren.")
    print(message)
}
```

Wenn der Preisindex aktuell ist, wird die prozentuale Änderung des Preisindex (con Vorjahr zu aktuellem Jahr) automatisch in die aktuelle `ECKWERTE`-Version eingefügt.

```
# Überprüfe die Bedingung für das Aktualisieren von
# ECKWERTE$preis
if (year(adjusted_date) - 1 == max(PREISINDEX$jahr)) {
    max_year <- max(PREISINDEX$jahr)

    # Berechne den neuen Preisindex
    preis_delta <- (PREISINDEX$lik_basis_1977[PREISINDEX$jahr ==
        max_year] - PREISINDEX$lik_basis_1977[PREISINDEX$jahr ==
```

```
        max_year - 1])/PREISINDEX$lik_basis_1977[PREISINDEX$jahr ==
        max_year - 1] * 100

    # Aktualisiere die Werte in ECKWERTE$preis
    ECKWERTE <- ECKWERTE |>
        mutate(preis = ifelse(jahr == max_year & year(dmy(ECKWERTE$stand) %m+%
            months(1)) - 1 == max_year, preis_delta, preis))
}
```

Anschliessend wird es noch überprüft, ob die aktuellsten Eckwerte zeitlich korrekt begginen (nämlich erst nach dem letzten Lohnindex-Jahr). Wenn nicht, kommt eine detaillierte Warnung mit Handlungsanleitung.

```
# Überprüfen, ob die letzten Eckwerte sich nicht mit dem
# letzten verfügbaren SLI überschneiden
if (min(ECKWERTE$jahr[dmy(ECKWERTE$stand) == max(dmy(ECKWERTE$stand)) &
    ECKWERTE$version == max(ECKWERTE$version[dmy(ECKWERTE$stand) ==
        max(dmy(ECKWERTE$stand))])]) <= max(LOHNINDEX$jahr)) {

    message <- paste0("Die aktuellsten Eckwerte mit ID ",
↪   first(ECKWERTE$id[dmy(ECKWERTE$stand) ==
        max(dmy(ECKWERTE$stand)) & ECKWERTE$version ==
        ↪  max(ECKWERTE$version[dmy(ECKWERTE$stand) ==
        max(dmy(ECKWERTE$stand))])]), " sind ab Jahr ",
        ↪  min(ECKWERTE$jahr[dmy(ECKWERTE$stand) ==
        max(dmy(ECKWERTE$stand)) & ECKWERTE$version ==
        ↪  max(ECKWERTE$version[dmy(ECKWERTE$stand) ==
        max(dmy(ECKWERTE$stand))])]), ", aber der letzte Wert vom SLI ist im Jahr ",
        max(LOHNINDEX$jahr), ". Das erste Jahr der aktuellsten Eckwerte sollte daher das
        ↪  Jahr ",
        max(LOHNINDEX$jahr) + 1, " sein. Bitte die letzten Eckwerte copy-pasten, Version
        ↪  auf ",
        max(ECKWERTE$version[dmy(ECKWERTE$stand) == max(dmy(ECKWERTE$stand))]) +
            1, " setzen, und das Jahr", max(LOHNINDEX$jahr),
        " entfernen.")
    print(message)
}
```

Zum Abschluss werden die vorbereitenden Daten wie folgt an `prepare_input.R` zurückgegeben:

```
# --- Return tidy df with eckwerte
# ----------------------------------------

return(ECKWERTE = ECKWERTE)
```

3.9 Verzinsung der Anlagen

Die Verzinsung der Anlagen wird aus der entsprechenden Datei mithilfe der Funktion read_delim() eingelesen:

```
ZINS_RAW <- read_delim(file.path(PARAM_INPUTS$path_allgemein_go,
    PARAM_INPUTS$file_zins), delim = ";", show_col_types = FALSE,
    trim_ws = TRUE)
```

3.10 Modul mod_input_eo_abrechnung.R

Dokumentation zuletzt aktualisiert am 16.10.2025

Das Modul `mod_input_eo_abrechnung` liest die Betriebsrechung der EO[21] ein. Es wird wie folgt in `prepare_input.R` aufgerugen:

```
# Input EO Abrechnungsdaten
tl_input_eo_abrechnung <- mod_input_eo_abrechnung(PARAM_INPUTS = PARAM_INPUTS)
```

Als Erstes werden die Daten aus der monatlichen EO-Abrechnung eingelesen und in ein geeignetes Format gebracht.

```
# Monatsdaten ab dem mass_db General open rectangle Upper
# left = A11, everything else unspecified

ABRECHNUNG_M <- read_excel(path = paste0(PARAM_INPUTS$path_eo_go,
    PARAM_INPUTS$file_eo_abrechnung_m), sheet = PARAM_INPUTS$sheet_eo_abrechnung_def,
    range = readxl::cell_limits(c(11, 1), c(NA, NA)))

# Identify empty lines column 1 : jahr column 2 : monat

ABRECHNUNG_M <- ABRECHNUNG_M %>%
    mutate(id = apply(.[, -c(1, 2)], MARGIN = 1, FUN = sum, na.rm = T)) %>%
    mutate(id = if_else(id == 0, 0, 1))
```

Als nächstes wird sichergestellt, dass nur Abrechnungsdaten von Jahren, für welche sämtliche Monatsdaten vorliegen, verwendet werden.

```
# Identify full years

id_full_years <- ABRECHNUNG_M %>%
    dplyr::select(jahr, monat, id) %>%
    group_by(jahr) %>%
    summarise(id_y = sum(id)) %>%
    ungroup()

ABRECHNUNG_M <- full_join(ABRECHNUNG_M, id_full_years, by = "jahr")

ABRECHNUNG_M_DEF <- ABRECHNUNG_M %>%
    filter(id_y == 12) %>%
    dplyr::select(-id, -id_y)
```

In einem nächsten Schritt werden die provisorischen Dezemberdaten eingelesen. Diese werden für die Erstellung der provisorischen EO-Abrechnung verwendet [22] Zu beachten ist, dass das Blatt `PARAM_INPUTS$sheet_eo_abrechnng_prov`, welches im ersten Codeblock eingelesen wird, lediglich die Abrechnungsdaten für den Monat Dezember enthält.

```
# Provisorische Monatsdaten ab dem mass_db General open
# rectangle Upper left = A11, everything else unspecified

ABRECHNUNG_M_PROV <- read_excel(paste0(PARAM_INPUTS$path_eo_go,
    PARAM_INPUTS$file_eo_abrechnung_m), sheet = PARAM_INPUTS$sheet_eo_abrechnung_prov,
```

[21] Die Zentrale Ausgleichsstelle ZAS stellt die Daten aus der Jahresrechnung t im April des Jahres t+1 bereit.

[22] In den Abrechungsdaten der ZAS sind die Ausgaben nicht nach Urlaubstyp aufgeschlüsselt. Die Verteilung nach Urlaubstyp erfolgt anhand des EO-Registers. Die Daten nach Abrechnungskajr können aus dem EO Register ab Mai ausgwertet werden. Die provisorische Abrechnung ist nicht sehr nutzvoll.

```
    range = readxl::cell_limits(c(11, 1), c(NA, NA)))

year_prov <- unique(ABRECHNUNG_M_PROV$jahr)

ABRECHNUNG_M_OHNE_DEZ <- ABRECHNUNG_M %>%
    filter(id_y >= 11) %>%
    filter((!(jahr %in% year_prov & monat == 12))) %>%
    dplyr::select(-id, -id_y)

ABRECHNUNG_M_PROV <- bind_rows(ABRECHNUNG_M_OHNE_DEZ, ABRECHNUNG_M_PROV) %>%
    arrange(jahr, monat)

ABRECHNUNG_M_ALL <- bind_rows(ABRECHNUNG_M_DEF, ABRECHNUNG_M_PROV,
    .id = "id")
```

Es werden also zuerst die provisorischen Dezember-Daten eingelesen und danach mit den definitiven Monatsdaten ohne Dezember-Daten im Dataframe `ABRECHUNG_M_PROV` kombiniert. Zum Schluss werden die provisorischen Monatsdaten `ABRECHNUNG_M_PROV` mit den definitiven Monatsdaten `ABRECHNUNG_M_DEF`im Dataframe `ABRECHNUNG_M_ALL` kombiniert, wobei die Variable `id` angibt, ob es sich um die definitiven Daten (`id==1`) oder die provisorischen Daten (`id==2`) handelt.

Als nächstes werden die Daten von Millionen CHF in CHF umgerechnet um nach Jahr aggregiert:

```
# Beitraege in Millionen Franken ausser id jahr monat

ABRECHNUNG_M_ALL <- ABRECHNUNG_M_ALL %>%
    mutate_at(.vars = colnames(ABRECHNUNG_M_ALL)[!(colnames(ABRECHNUNG_M_ALL) %in%
        c("id", "jahr", "monat"))], list(~. * 1e+06)) %>%
    mutate(jahr = as.integer(jahr), monat = as.integer(monat))

# --- Variables specific to eo
# ------------------------------------------------

ABRECHNUNG_M_ALL <- ABRECHNUNG_M_ALL

#-------------------------------------------------------------------------------

# --- Jahresdaten = Summe Monatsdaten
# ----------------------------------------

ABRECHNUNG <- ABRECHNUNG_M_ALL %>%
    dplyr::select(-monat) %>%
    mutate_all(list(~as.numeric(.))) %>%
    mutate_all(list(~if_else(is.na(.), 0, .))) %>%
    group_by(id, jahr) %>%
    summarise_all(sum) %>%
    ungroup()
```

Als nächstes werden die provisorischen und definitven Daten der EO-Jahresabrechnung eingelesen. Aus diesen Daten wird ausschliesslich der Stands des EO-Fonds (Kapital und flüssige Mittel) verwendet. Für die restliche Positionen der EO-Abrechnung werden die aus der Summe der monatlichen Abrechnungen errechneten Jahreswerte verwendet. Wir verwenden die monatlichen Abrechnung anstatt der jährlichen, da diese eine detaillierte Gruppierung der Ausgabepositionen enthalten:

```r
# --- Bilanzdaten
# ------------------------------------------------------------

# Kapital not available in the mass_db monthly data Kapital
# available in the mass_db yearly data

# Definitive Jahresdaten ab dem mass_db General open
# rectangle Upper left = A11, everything else unspecified
ABRECHNUNG_Y <- read_excel(paste0(PARAM_INPUTS$path_eo_go,
↪  PARAM_INPUTS$file_eo_abrechnung_fin),
    sheet = PARAM_INPUTS$sheet_eo_abrechnung_def, range = readxl::cell_limits(c(11,
        1), c(NA, NA)))

# Provisorische Jahresdaten ab dem mass_db General open
# rectangle Upper left = A11, everything else unspecified

ABRECHNUNG_Y_PROV <- read_excel(paste0(PARAM_INPUTS$path_eo_go,
    PARAM_INPUTS$file_eo_abrechnung_fin), sheet = PARAM_INPUTS$sheet_eo_abrechnung_prov,
    range = readxl::cell_limits(c(11, 1), c(NA, NA)))

year_prov <- unique(ABRECHNUNG_Y_PROV$jahr)

ABRECHNUNG_Y_OHNE_PROV <- ABRECHNUNG_Y %>%
    filter((!(jahr %in% year_prov)))

ABRECHNUNG_Y_PROV <- bind_rows(ABRECHNUNG_Y_OHNE_PROV, ABRECHNUNG_Y_PROV)

ABRECHNUNG_Y_ALL <- bind_rows(ABRECHNUNG_Y, ABRECHNUNG_Y_PROV,
    .id = "id")

# Beiträge in Millionen Franken

ABRECHNUNG_Y_ALL <- ABRECHNUNG_Y_ALL |>
    mutate(across(.cols = !c(id, jahr), .fns = ~as.numeric(.) *
        1e+06)) |>
    mutate(jahr = as.integer(jahr))

# --- Select subset of variables from yearly data
# ----------------------------

ABR_Y_SEL <- ABRECHNUNG_Y_ALL %>%
    dplyr::select(id, jahr, kap, fl_mtl) %>%
    filter(min(ABRECHNUNG_M_ALL$jahr) <= jahr & jahr <= max(ABRECHNUNG_M_ALL$jahr))

ABR_Y_SEL <- ABR_Y_SEL %>%
    mutate(id = as.numeric(id))

ABRECHNUNG <- left_join(ABRECHNUNG, ABR_Y_SEL, by = c("id", "jahr"))

# Subset of variables usied in delfin

ABRECHNUNG <- ABRECHNUNG %>%
    dplyr::select(names(ABRECHNUNG))
```

Demnäscht werden die Aberchnungsdaten in ein Dataframe für die provisorischen Abrechnungssdaten und ein eines für die definitiven Abrechnungsdaten aufgeteilt.

```
# Definitive EO Abrechnungen

EO_ABRECHNUNG_DEF <- ABRECHNUNG %>%
    filter(id == 1) %>%
    dplyr::select(-id)

# Provisorische EO Abrechnungen Bis 2016: definitive
# Jahresdaten Ab 2017: provisorische Jahresdaten

EO_ABRECHNUNG_PROV <- ABRECHNUNG %>%
    filter(id == 2) %>%
    dplyr::select(-id)
```

In den Abrechungsdaten der ZAS sind die Ausgaben nicht nach Urlaubstyp aufgeschlüsselt. Die Verteilung nach Urlaubstyp erfolgt anhand des EO-Registers. Diese Verteilung wird eingelesen und in der Dataframe `EO_AUSGABEN_ANTEILE` abgespeichert.

```
# Anteile der Ausgaben pro Leistungsart
EO_AUSGABEN_ANTEILE <- read_excel(paste0(PARAM_INPUTS$path_eo_go,
    PARAM_INPUTS$file_eo_ausgaben_anteile), sheet =
↪   PARAM_INPUTS$sheet_eo_ausgaben_anteile)
```

Zum Abschluss werden die provisorischen und definitiven Abrechnungsdaten sowie die Verteilung nach Urlaubstyp an `prepare_input.R` zurückgegeben:

```
# --- Output
# ------------------------------------------------------------------

return(list(EO_ABRECHNUNG_DEF = EO_ABRECHNUNG_DEF, EO_ABRECHNUNG_PROV =
↪   EO_ABRECHNUNG_PROV,
    EO_AUSGABEN_ANTEILE = EO_AUSGABEN_ANTEILE))
```

3.11 Modul mod_input_eo_ofs_hist.R

Dokumentation zuletzt aktualisiert am 16.10.2025

Dieses Modul dient der Aufbereitung historischer Daten über Lebendgeburten in der Schweiz, aggregiert nach dem Alter der Mutter oder des Vaters. Es wird wie folgt in `prepare_input.R` aufgerugen:

```
# Input Naissances
tl_mod_input_eo_ofs_hist <- mod_input_eo_ofs_hist(PARAM_INPUTS = PARAM_INPUTS)
```

Zuerst werden der Pfad zur Excel-Datei, die Namen der Arbeitsblätter sowie der einzulesende Zellbereich festgelegt.

```
path_eo <- PARAM_INPUTS$path_eo_go

file_f_lebendgeburten_hist <- PARAM_INPUTS$file_f_lebendgeburten_hist
path_f_hist <- paste0(path_eo, file_f_lebendgeburten_hist)
sheet_names <- excel_sheets(path_f_hist)
sheet_names <- sheet_names[-1]
range_f_lebendgeburten_hist <- PARAM_INPUTS$range_f_lebendgeburten_hist
```

Die Excel-Datei enthält ein Blatt pro Jahr. Diese werden so eingelesen, dass am Ende ein einheitlicher Da-

tensatz entsteht, der alle Blätter untereinander zusammenführt. Die Spalten werden dann noch umbenannt.

```
GEBURT_NACH_ALTER_MUTTER <- sheet_names |>
    set_names() |>
    map_dfr(~read_excel(path_f_hist, sheet = .x, range = range_f_lebendgeburten_hist) |>
        mutate(year = .x), .id = "sheet_name")

colnames(GEBURT_NACH_ALTER_MUTTER) <- c("jahr", "alt_mut", "anzahl",
    "year")
```

Im nächsten Schritt erfolgt die Bereinigung der Daten, wobei die Spalten alt_mut und jahr in numerisches Format konvertiert werden.

```
LEBENDGEBURTEN_FRAU <- GEBURT_NACH_ALTER_MUTTER |>
    select(-year) |>
    filter(alt_mut != "Total") |>
    mutate(alt_mut = case_when(alt_mut == "<=12" ~ "12", alt_mut ==
        ">=60" ~ "60", TRUE ~ alt_mut)) |>
    mutate(alt_mut = as.numeric(alt_mut)) |>
    mutate(jahr = as.numeric(jahr)) |>
    rename(alt = alt_mut, anzahl_geburten = anzahl)
```

Anschließend wird die Anzahl der Lebendgeburten nach Altersklassen der Mutter aggregiert, sodass sie den Alterkategorien der demografischen Szenarien des BFS entsprechen.

```
# Catégorie d'âge comme dans les scénarios
LEBENDGEBURTEN_FRAU <- LEBENDGEBURTEN_FRAU |>
    mutate(kat = case_when(alt >= 15 & alt <= 19 ~ 1, alt >=
        20 & alt <= 24 ~ 2, alt >= 25 & alt <= 29 ~ 3, alt >=
        30 & alt <= 34 ~ 4, alt >= 35 & alt <= 39 ~ 5, alt >=
        40 & alt <= 44 ~ 6, alt >= 45 ~ 7)) |>
    select(-alt) |>
    group_by(kat, jahr) |>
    mutate(anzahl_geburten = sum(anzahl_geburten)) |>
    unique() |>
    select(jahr, kat, anzahl_geburten)
```

Die gleichen Schritte werden wiederholt, jedoch so angepasst, dass sie der spezifischen Datenstruktur der Lebendgeburten in der Schweiz nach dem Alter des Vaters entsprechen.

```
# Naissances par age du père
file_m_lebendgeburten_hist <- PARAM_INPUTS$file_m_lebendgeburten_hist
path_m_hist <- paste0(path_eo, file_m_lebendgeburten_hist)
range_m_lebendgeburten_hist <- PARAM_INPUTS$range_m_lebendgeburten_hist
max_year_m_lebendgeburten_hist <- PARAM_INPUTS$max_year_m_lebendgeburten_hist

# Naissances par âge du père
LEBENDGEBURTEN_MANN <- read_excel(path_m_hist, range = range_m_lebendgeburten_hist,
    col_names = TRUE)


colnames(LEBENDGEBURTEN_MANN) <- c("alt", c(2007:max_year_m_lebendgeburten_hist))

LEBENDGEBURTEN_MANN <- LEBENDGEBURTEN_MANN |>
    slice(-1) |>
    pivot_longer(cols = !alt, names_to = "jahr", values_to = "anzahl_geburten")
```

```r
# Catégorie d'âge comme dans les scénarios
LEBENDGEBURTEN_MANN <- LEBENDGEBURTEN_MANN |>
    mutate(kat = case_when(alt >= 15 & alt <= 19 ~ 1, alt >=
        20 & alt <= 24 ~ 2, alt >= 25 & alt <= 29 ~ 3, alt >=
        30 & alt <= 34 ~ 4, alt >= 35 & alt <= 39 ~ 5, alt >=
        40 & alt <= 44 ~ 6, alt >= 45 ~ 7)) |>
    select(-alt) |>
    mutate(jahr = as.numeric(jahr)) |>
    group_by(kat, jahr) |>
    summarise(anzahl_geburten = sum(anzahl_geburten))
```

Schlussendlich werden die vorbereitende Dataframes an `prepare_input.R` zurückgegeben:

```r
return(list(LEBENDGEBURTEN_FRAU = LEBENDGEBURTEN_FRAU, LEBENDGEBURTEN_MANN =
↪  LEBENDGEBURTEN_MANN))
```

3.12 Modul mod_input_eo_register.R

Dokumentation zuletzt aktualisiert am 16.10.2025

Dieses Modul dient zum Lesen und Aufbereiten der Daten aus dem EO-Register. Es wird wie folgt in `prepare_input.R` aufgerugen:

```r
# Input EO Register
tl_mod_input_register <- mod_input_eo_register(PARAM_INPUTS = PARAM_INPUTS)
```

Zuerst werden die Pfade festgelegt, die zu den einzulesenden Daten führen und die Daten eingelesen.

```r
path_eo_go <- PARAM_INPUTS$path_eo_go
file_eo_register <- PARAM_INPUTS$file_eo_register
file_eo_register_beuadopt <- PARAM_INPUTS$file_eo_register_beuadopt
file_eo_dienst <- PARAM_INPUTS$file_eo_dienst
file_eo_dienst_ausgaben <- PARAM_INPUTS$file_eo_dienst_ausgaben
REGISTER <- read.csv(paste0(path_eo_go, file_eo_register), sep = ",")
REGISTER_BEUADOPT <- read.csv(paste0(path_eo_go, file_eo_register_beuadopt),
    sep = ",")
REGISTER_DIENST <- read.csv(paste0(path_eo_go, file_eo_dienst),
    sep = ";")
AUSGABEN_DIENST_PRO_LEISTUNGSART <- read.csv(paste0(path_eo_go,
    file_eo_dienst_ausgaben), sep = ";")
```

Danach werden die Daten pro Urlaubstyp geteilt und die Variablen umbenannt.

```r
# Mutterschaftsurlaube
# ------------------------------------------------------
REGISTER_MSU <- REGISTER |>
    filter(cgsp == 90) |>
    select(dom, jahr = year, kat = age_gr, mutterschaftsurlaub_ohne_eomax = nb_eo_umax,
        mutterschaftsurlaub_eomax = nb_eo_max, taggeld = mean_tagesansatz_umax,
        tage = mean_jour)

# Urlaube der anderen Elternteil
# ------------------------------------------
REGISTER_VSU <- REGISTER |>
    filter(cgsp == 91) |>
```

```
    select(dom, jahr = year, kat = age_gr, vaterschaftsurlaub_ohne_eomax = nb_eo_umax,
        vaterschaftsurlaub_eomax = nb_eo_max, taggeld = mean_tagesansatz_umax,
        tage = mean_jour)

# Betreuungsurlaube
# ------------------------------------------------------------
REGISTER_BEU <- REGISTER_BEUADOPT |>
    filter(cgsp == 92) |>
    select(jahr, anzahl_urlaube, tage, anteil_muetter, ausgaben)

# Adoptionsurlaube
# ------------------------------------------------------------
REGISTER_ADOPT <- REGISTER_BEUADOPT |>
    filter(cgsp == 93) |>
    select(jahr, anzahl_urlaube, tage, anteil_muetter, ausgaben)

# Dienst
# ------------------------------------------------------------------------
EO_DIENST_KAT <- REGISTER_DIENST |>
    rename(kat_leist = leistungsart, dienstleistende = n_dienstleistende,
        kat_alt = age_gr, taggeld = mean_tagesansatz_corr, tage_zwminmax =
        ↪  mean_tage_zwminmax,
        tage_eomin = mean_tage_eomin, tage_eomax = mean_tage_eomax) |>
    select(jahr, kat_leist, kat_alt, dienstleistende, sum_kz,
        sum_betriebsz, sum_betreuungsz, taggeld, anteil_zwminmax,
        anteil_eomax, anteil_eomin, tage_zwminmax, tage_eomin,
        tage_eomax)

AUSGABEN_DIENST_PRO_LEISTUNGSART <- AUSGABEN_DIENST_PRO_LEISTUNGSART |>
    select(jahr, kat_leist = leistungsart, sum_taggelder)

REGISTER_DIENST <- EO_DIENST_KAT |>
    full_join(AUSGABEN_DIENST_PRO_LEISTUNGSART, by = c("jahr",
        "kat_leist"))
```

Die vorbereitete Daten werden schlussendlich an `prepare_input.R`zurückgegeben.

```
return(list(REGISTER_MSU = REGISTER_MSU, REGISTER_VSU = REGISTER_VSU,
    REGISTER_ADOPT = REGISTER_ADOPT, REGISTER_BEU = REGISTER_BEU,
    REGISTER_DIENST = REGISTER_DIENST))
```

3.13 Modul mod_input_eo_massnahmen.R

Dokumentation zuletzt aktualisiert am 24.06.2025

Das Modul `mod_input_eo_massnahmen.R` liest die Daten ein, die für die Schätzung der Auswirkungen von Politikmassnahmen benötigt werden. Es wird wie folgt in `prepare_input.R`aufgerufen:

```
# Input der in xls berechneten EO-Massnahmen
eo_massnahmen <- mod_input_eo_massnahmen(PARAM_INPUTS = PARAM_INPUTS)
```

Das Modul liest die Ausgaben- respektive Einnahmevektoren der Politikmassnahmen, für welche die Berechnungen ausserhalb des EO Finanzperspektivenmodells durchgeführt wurden. Es unterscheidet zwischen Massnahmen, die Teil der EOG-REform sind und andere Massnahmen.

```
path_eo_massnahmen <- PARAM_INPUTS$path_eo_massnahmen

#------ Lesen Inhalt csv - sheet -------------------------------------#

EO_MASSN <- read_excel(file.path(paste0(path_eo_massnahmen,
↪  PARAM_INPUTS$file_eo_massnahmen)),
    sheet = PARAM_INPUTS$sheet_eo_massnahmen, col_names = TRUE)

EOG_REVISION_MASSN <- read_excel(file.path(paste0(path_eo_massnahmen,
    PARAM_INPUTS$file_eog_revision_massn)), sheet =
↪  PARAM_INPUTS$sheet_eog_revision_massn)
```

Die Dataframes werden schlussendlich an `prepare_input.R`zurückgegeben.

```
#------ Output ------------------------------------------------------------#
return(list(EO_MASSN = EO_MASSN, EOG_REVISION_MASSN = EOG_REVISION_MASSN))
```

4 Module zur Berechnung der Finanzperspektiven

In diesem Kapitel werden die Module beschrieben, mithilfe welcher das Finanzperspektivenmodell EO berechnet wird. Die hier beschriebenen Module lesen die in Kapitel 3 aufbereiteten Input-Daten sowie die Datei `PARAM_GLOVAL.csv` ein, führen die Berechnungen des Finanzperspektivenmodells durch, und lesen verschiedene Tabellen mit Resultaten aus. Hierzu zählen insbesondere auch die auf der Internetseite des BSV veröffentlichen Tabellen zu den finanziellen Perspektiven der EO[23].

4.1 Modul run_eo.R

Dokumentation zuletzt aktualisiert am 25.06.2025

Das Hauptmodul zur Berechnung des Finanzperspektivenmodells der EO wird wie folgt aufgerufen:

```
run_eo(path_param = path_param, path_inp = path_inp, path_out = path_out)
```

`path_param`ist der Pfad zum Ordner, welcher die zu verwendenden Modellparameter enthält, insbesondere auch die Datei `PARAM_GLOBAL.csv`. `path_inp` ist der Pfad zu den Inputdaten, und `path_out` ist der Pfad, wo die Resultate der Modellberechnungen schlussendlich abgelegt werden sollen.

Das Modul `run_eo`enthält die folgende Elemente:

```
if (length(path_param) > 1) {
    return(multi_run(path_param, run_eo, path_inp, path_out))
}
```

Dieser erste Codeteil ist für den Fall, dass unter `path_param` eine Liste von Pfade spezifiert wird, was dazu führt, dass die Finanzperspektivenberechnungen nacheinander für jede in `path_param` Container sprezifizierte `PARAM_GLOBAL` Datei ausgeführt werden.

Danach werde mit dem Modul `mod_read_data` (vgl. 4.2) der unter `path_param` abgelegte Parametercontainer sowie die unter `path_inp` abgelegten Input-Daten eingelesen. Mit `data_folders=c(älgemein", ëo")` wird angegeben, dass sowohl die Allgemein als auch die EO-spezifischen Input-Daten eingelesen werden sollen:

```
# Parameterwerte und Inputdaten einlesen
tl_inp <- mod_read_data(path_param, path_inp, data_folders = c("allgemein",
    "eo"))
```

[23] https://www.bsv.admin.ch/bsv/fr/home/assurances-sociales/eo-msv/finanzen.html

Im Frühjahr 2025 wurde die Struktur der Parametercontainer geändert, indem nur die Parameter, die mehrmals pro Jahr geändert werden, in der Datei `PARAM_GLOBAL.csv` enthalten sind. Die übrigen Parameter werden jetzt im Skript `mod_eo_param_global` (vgl. 4.3) festgelegt. Zuvor wurden die EO-spezifischen Parameter jedoch in der Datei `PARAM_EO.csv` festgelegt. Der folgende Codeteil sorgt dafür, dass die alten Parametercontainer mit der Datei `PARAM_EO.csv` weiterhin funktionieren.

```
# Folgender Codeblock dient lediglich dazu, alte Container
# mit PARAM_EO ausführbar zu halten.  Kann mittelfristig
# entfernt werden, da PARAM_EO in Mai 25 in PARAM_GLOBAL
# integriert wurde Prüfe, ob PARAM_EO in tl_inp existiert
if ("PARAM_EO" %in% names(tl_inp)) {

    # Hole den Datensatz als Liste (eine Zeile)
    param_eo <- tl_inp$PARAM_EO |>
        as.list()

    # Weise jeden Eintrag in PARAM_GLOBAL zu (überschreibe
    # oder füge hinzu)
    param_eo |>
        walk2(names(param_eo), \(wert, name) {
            tl_inp$PARAM_GLOBAL[[name]] <<- wert
        })
    rm(param_eo)
}
```

Als nächstes werden die Parameterwerte überprüft und default-Parameterwerte gesetzt:

```
# Modul zum überprüfen von Parameterwerten und setzten von
# default-Parameterwerten:
tl_inp$PARAM_GLOBAL <- mod_eo_param_global(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE = tl_inp$ECKWERTE, ESTV = tl_inp$ESTV, BEV_BESTAND = tl_inp$BEV_BESTAND,
    EO_AUSGABEN_ANTEILE = tl_inp$EO_AUSGABEN_ANTEILE, REGISTER_MSU = tl_inp$REGISTER_MSU,
    tl_inp = tl_inp)
```

Das Modul `mod_eo_param_global` dient dazu, default-Parameterwerte zu setzen, falls diese nicht in `PARAM_GLOBAL` spezifiert sind (vgl. 4.3).

Als nächtses wird das Modul `wrap_eo` (vgl. 4.4) aufgerufen, in welchem die einzelnen Ausgaben- und Einnahmenposition des Finanzperspektivenmodells berechnet werden:

```
tl_out_eo <- wrap_eo(tl_inp)
```

Zum Abschluss werden die Output-Dateien aufbereitet und in den vorangehend spezifizierten Output-Ordner geschrieben:

```
# output
# ------------------------------------------------------------------------

path_out_identifier <- mod_out_eo(path, path_out, tl_inp_eo = tl_inp,
    tl_out_eo)

return(path_out_identifier)
```

4.2 Modul mod_read_data.R

Dokumentation zuletzt aktualisiert am 25.06.2025

Das Modul `mod_read_data.R` liest den zu verwendenden Parametercontainer, sowie die Input-Daten ein. Es wird in `run_eo.R` wie folgt aufgerufen:

```
# Parameterwerte und Inputdaten einlesen
tl_inp <- mod_read_data(path_param, path_inp, data_folders = c("allgemein",
    "eo"))
```

Als erstes wird überprüft, dass die Pfade zu dem Parametercontainer (`path_param`) und zu den Input-Daten existiert.

```
# Prüfe, ob die Pfade existieren
if (!dir.exists(path_param)) stop("path_param existiert nicht, bitte Pfad überprüfen und
↪  anpassen.")
if (!dir.exists(path_inp)) stop("path_inp existiert nicht, bitte Pfad überprüfen und
↪  anpassen.")
```

Dann werden die Namen der .csv-Dateien mit den Parameterwerten in die Liste `csv_files_param` eingelesen, sowie die Namen der .csv-Dateien mit den Input-Daten in die Liste `csv_files_folders` eingelesen:

```
csv_files_param <- list.files(path = path_param, pattern = "^PARAM.*\\.csv$",
    full.names = TRUE, recursive = TRUE)

# Finde alle .csv-Dateien in den angegebenen Unterordnern
csv_files_folders <- map(data_folders, ~list.files(path = file.path(path_inp,
    .x), pattern = "\\.csv$", full.names = TRUE, recursive = TRUE)) %>%
    unlist()
```

Als nächstes werden die Funktionen definiert, mit welchem die einzelnen Parameter-Dateien (`process_file_param`) respektive die einzelnen Input-Dateien (`process_file_raw`) eingelesen werden:

```
  # Funktion für die 'PARAM'-Dateien mit Transformation
    process_file_param <- function(file) {
        cat("Reading: ", basename(file), "\n", sep = "")
        suppressMessages(
            read_delim(file, delim  = ";", show_col_types = FALSE, trim_ws = TRUE) # Lese
            ↪  Datei und lasse Spaltentypen automatisch bestimmen
         %>%
            select(1:2) %>%                                                  # Behalte nur
            ↪  die ersten zwei Spalten
            pivot_wider(names_from = "key", values_from = "value") %>%       #
            ↪  Transponiere die Daten
           { if (any(sapply(., is.character))) type_convert(.) else . }
        ) # Passe Spaltentypen an
        }

    # Funktion für die Ordner-Dateien ohne Transformation
    process_file_raw <- function(file) {
        cat("Reading: ", basename(file), "\n", sep = "")
        suppressMessages(
            read_delim(file, delim  = ";", show_col_types = FALSE, trim_ws = TRUE) %>% #
            ↪  Lese Datei und lasse Spaltentypen automatisch bestimmen
              { if (any(sapply(., is.character))) type_convert(.) else . }
↪   # Passe Spaltentypen an
        )
    }
```

Als nächstes werden die Parameter-Dateien und die Input-Dateien unter Verwendung der oben beschriebenen

Funktionen eingelesen und der Liste `tl_inp` hinzugefügt:

```
# Verarbeite alle Dateien und benenne sie nach Dateinamen
tl_inp <- list()  # Initialisiere die Liste

# Füge die verarbeiteten 'PARAM'-Dateien hinzu
tl_inp <- c(tl_inp, set_names(map(csv_files_param, process_file_param),
    basename(csv_files_param) %>%
        str_remove("\\.csv$")))

# Füge output-id zu PARAM_GLOBAL hinzu
tl_inp$PARAM_GLOBAL$identifier_number <- ffh_identifier_number("iv",
    path_param)

# Füge die unbearbeiteten Dateien aus den angegebenen
# Ordnern hinzu
tl_inp <- c(tl_inp, set_names(map(csv_files_folders, process_file_raw),
    basename(csv_files_folders) %>%
        str_remove("\\.csv$")))
```

Zum Schluss wird die Liste `tl_inp` an `run_eo` zurückgegeben:

```
return(tl_inp)
```

4.3 Modul mod_eo_param_global.R

Dokumentation zuletzt aktualisiert am 16.10.2025

Das Modul `mod_eo_param_global.R` setzt default-Parameterwerte in `PARAM_GLOBAL`. Es wird in `run_eo.R` wie folgt aufgerufen:

```
tl_inp$PARAM_GLOBAL <- mod_eo_param_global(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE = tl_inp$ECKWERTE, ESTV = tl_inp$ESTV, BEV_BESTAND = tl_inp$BEV_BESTAND,
    tl_inp = tl_inp)
```

Das Modul setzt Parameterwerte, für welche grundsätzlich ein Standardwert ("Default") existiert, für welche es aber in gewissen Fällen (bspw. Zwecks Backtesting oder Reproduktion von Berechnungen in der Vergangenheit) möglich sein soll, diese in der Datei `PARAM_GLOBAL.csv` anders zu wählen. Das Modul enthält folgende Code, welcher selbsterklärend sein sollte:

```
    ## DEFAULT PARAMETER SETZEN, UM AKTUELLSTE WERTE ZU VERWENDEN

    # Eckwerte:
    # Überprüfe, ob id_eckwerte existiert
    if (!"id_eckwerte" %in% names(PARAM_GLOBAL)) {

        # Finde den höchsten Wert von "laufjahr" und "version"
        id_selected <- ECKWERTE %>%
            filter(laufjahr == max(laufjahr)) %>%              # Finde den höchsten
            ↪  "laufjahr"
            filter(version == max(version))                    # Finde den höchsten
            ↪  "version"

        # Prüfe, ob mehrere verschiedene idestv existieren
        unique_id <- id_selected %>% distinct(id)

        if (nrow(unique_id) != 1) {
```

```
            stop("Mehrere verschiedene id im Dataframe ECKWERTE mit dem höchsten Wert von
            ↪  laufjahr und version gefunden. Code wird angehalten.")
        } else {
            # Setze id_estv in PARAM_GLOBAL auf den ausgewählten Wert
            PARAM_GLOBAL$id_eckwerte <- unique_id$id
        }
        rm(id_selected, unique_id)
    }

    # FHH für das mittlere Szenario (Basisszenario) berechnen, falls nicht festgelegt
    if(!"szenario_fhh" %in% names(PARAM_GLOBAL) || PARAM_GLOBAL$szenario_fhh == "mittel")
    ↪  {
      PARAM_GLOBAL$szenario_fhh <- "referenz"
    }

    # Setze bev_scenario auf B_00_2025, falls Szenario günstig ist und kein passender
    ↪  Wert vorhanden ist
    if (PARAM_GLOBAL$szenario_fhh == "hoch" &&
        (!"bev_scenario" %in% names(PARAM_GLOBAL) ||
↪   !startsWith(PARAM_GLOBAL$bev_scenario, "B_00"))) {
      PARAM_GLOBAL$bev_scenario <- "B_00_2025"
      warning("PARAM_GLOBAL$bev_scenario wurde auf B_00_2025 gesetzt, da
      ↪  PARAM_GLOBAL$szenario_fhh=='hoch'")
    }

    # Setze bev_scenario auf C_00_2025, falls Szenario ungünstig ist und kein passender
    ↪  Wert vorhanden ist
    if (PARAM_GLOBAL$szenario_fhh == "tief" &&
        (!"bev_scenario" %in% names(PARAM_GLOBAL) ||
↪   !startsWith(PARAM_GLOBAL$bev_scenario, "C_00"))) {
      PARAM_GLOBAL$bev_scenario <- "C_00_2025"
      warning("PARAM_GLOBAL$bev_scenario wurde auf C_00_2025 gesetzt, da
      ↪  PARAM_GLOBAL$szenario_fhh=='tief'")
    }

    # Folgender Code nimmt das Referenzszenario an, falls kein Szenario in PARAM_GLOBAL
    ↪  spezifiziert wurde.
    if (!"bev_scenario" %in% names(PARAM_GLOBAL)) {
      PARAM_GLOBAL$bev_scenario <- "A_00_2025"
    }

  # Letztes verfügbares Abrechnungsjahr nehmen, falls nicht in PARAM_GLOBAL.csv gesetzt
  if (!"jahr_abr" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_abr <- max(EO_AUSGABEN_ANTEILE$jahr)
  }

  # Letztes verfügbares EO-Registerjahr nehmen, falls nicht in PARAM_GLOBAL.csv gesetzt
  if (!"jahr_register" %in% names(PARAM_GLOBAL)) {
     PARAM_GLOBAL$jahr_register <- max(REGISTER_MSU$jahr)
  }

    # Preisbasis auf Abrechnungsjahr legen, falls nicht in PARAM_GLOBAL.csv gesetzt
    if (!"jahr_preisbasis" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$jahr_preisbasis <- PARAM_GLOBAL$jahr_abr
```

```
}

# Der Parameter pi_eckwerte (Auswahl des Preisindexes) sollte 1 sein, jedoch könnte
↪  er (bspw. für Backtestings) auf 0 gesetzt werden.
if(!"pi_eckwerte" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$pi_eckwerte <- 1
}

# Jahr des Beginns des Resultatevektors auf 1997 setzten, falls nicht in
↪  PARAM_GLOBAL.csv gesetzt
if(!"jahr_beginn" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_beginn <- 1997
}

# Bestimme Jahr für Bevölkerungsbestand, falls nicht gesetzt
if (!"jahr_bev" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_bev <- BEV_BESTAND |>
        filter(!is.na(bevendejahr)) |>
        summarise(max_jahr = max(jahr)) |>
        pull(max_jahr)
}

# Ende des Projektionshorizonts festlegen, falls nicht in PARAM_GLOBAL.csv gesetzt
if(!"jahr_ende" %in% names(PARAM_GLOBAL)) {
  match <- regmatches(PARAM_GLOBAL$bev_scenario, regexpr("[0-9]{4}$",
↪ PARAM_GLOBAL$bev_scenario))
  if (length(match) == 1) {
    PARAM_GLOBAL$jahr_ende <- as.numeric(match) + 50
  } else {
    print("L'année de publication des scénarios démographiques n'a pas été reconnue
    ↪  par mod_eo_param_global.R")
  }
}

if(!"light_output" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$light_output <- FALSE
}

# laufendes Jahr jahr_lj festlegen, falls nicht in PARAM_GLOBAL.csv gesetzt
if(!"jahr_lj" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_lj <- PARAM_GLOBAL$jahr_abr+1
}

# EO-Maximum im 2023 anpassen, falls nicht in PARAM_GLOBAL.csv gesetzt
if(!"jahr_adapt_eomax" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_adapt_eomax <- 2023
}

# notwendiger Lohnanstieg setzten, falls nicht in PARAM_GLOBAL.csv gesetzt
if(!"tcmax" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$tcmax <- 0.12
}

# Spalten, die im Output in Millionen ausgegeben werden definieren, falls nicht
↪  definieren
```

```
    if("spaltenmio" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$spaltenmio <- "eo_dienst_just, eo_msu_just, eo_vsu_just,
↪   eo_bsu_just, eo_adoption_just, eo_verw_kost_just, aus_total_ink_vk, eo_beitragsum,
↪   umlage_eo, kap_etr, erg_betr, kap_abr, fluss_mittel" # Spalten, die in den
↪   aufbereiteten Output-files (e.g., FHH_AHV.xlsx)  in Millionen Franken ausgegeben
↪   werden
    }

    # Spalten mit den indizes für den Output, falls nicht definieren
    if("spaltenidx" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$spaltenidx <- "bs_dienst, bs_msu, bs_vsu, bs_bsu, bs_adoption,
↪   bs_total, liquid_ausgaben"
    }

    # Spalten mit den indizes für den Output, falls nicht definieren
    if("cols_with_pc" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$cols_with_pc <- "eo_dienst_just, eo_msu_just, eo_vsu_just,
↪   eo_bsu_just, eo_adoption_just, aus_total_ink_vk, eo_beitragsum" # Spalten, für welche
↪   in den aufbereiteten Output-files (e.g., FHH_AHV.xlsx) Jahr-zu-Jahr Wachstumsraten
↪   angezeigt werden.
    }

    # output_massn_less_mio_dec_round setzten, falls nicht definieren
    if(!"output_massn_less_mio_dec_round" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$output_massn_less_mio_dec_round <- FALSE
    }

    # n_digit_less_zero setzten, falls nicht definieren
    if(!"n_digit_less_zero" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$n_digit_less_zero <- 3
    }

    # Massnahmen-Output auf 3 Stellen runden, falls nicht definieren
    if(!"n_dec_massn_round" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$n_dec_massn_round <- 3
    }

    # Massnahmen-Output bis zum gleichen Jahr wie FHH-Output, falls nicht definieren
    if(!"jahr_lastoutput_massn" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$jahr_lastoutput_massn <- PARAM_GLOBAL$jahr_lastoutput
    }

    # 10 Jahre nach Abrechnung in .xls FHH schattieren und runden, falls nicht definieren
    if(!"year_after_abr_shade" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$year_after_abr_shade <- 10
    }

    # Schattierungen in .xls FHH einfügen, falls nicht definieren
    if(!"year_shade" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$year_shade <- TRUE
    }

    # Schattierte Jahre in .xls FHH runden, falls nicht definieren
    if(!"round_shade" %in% names(PARAM_GLOBAL)) {
```

```
    PARAM_GLOBAL$round_shade <- TRUE
}

# Eckwerte für 5 Jahre anzeigen, falls nicht anders spezifiziert
if(!"long_eckwerte" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$long_eckwerte <- 5 # Parameter, der angibt, für wie viele Jahre die
↪ Lohn- und Preiswachstumsraten unten rechts in den FHH angezeigt werden sollen.
}

# file_eooutput setzten, falls nicht anders spezifiziert
if(!"file_eooutput" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$file_eooutput <- "FH_EO.xlsx"
}

# file_eooutput setzten, falls nicht anders spezifiziert
if (!"flag_param_eo_massn" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$flag_param_eo_massn <- if (
      "PARAM_MASSNAHMEN_BASE" %in% names(tl_inp) &&
      !is.null(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen)) {
        PARAM_GLOBAL$flag_param_massn
    } else {
        FALSE
    }
}


# footnote1_on setzten, falls nicht anders spezifiziert
if(!"footnote1_on" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$footnote1_on <- FALSE
}

# footnote1_on setzten, falls nicht anders spezifiziert
if(!"footnote2_on" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$footnote2_on <- FALSE
}

# option_go_staf setzten, falls nicht anders spezifiziert
if(!"option_go_staf" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$option_go_staf <- TRUE
}

# option_go_staf setzten, falls nicht anders spezifiziert
if(!"option_go_staf" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$option_go_staf <- TRUE
}

# ba_go_staf setzten, falls nicht anders spezifiziert
if(!"ba_go_staf" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$ba_go_staf <- 0.202
}

# ba_go_staf_jahr setzten, falls nicht anders spezifiziert
if(!"ba_go_staf_jahr" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$ba_go_staf_jahr <- 2020
```

```
    }

    # bs_go_staf setzten, falls nicht anders spezifiziert
    if(!"bs_go" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$bs_go <- 8.7
    }

    # bs_go_staf setzten, falls nicht anders spezifiziert
    if(!"bs_go_staf" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$bs_go_staf <- 8.7
    }

    # bs_go_staf_jahr setzten, falls nicht anders spezifiziert
    if(!"bs_go_staf_jahr" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$demomwstpzt_go_staf_jahr <- 2020
    }

    # demomwstpzt_go_staf_jahr setzten, falls nicht anders spezifiziert
    if(!"demomwstpzt_go_staf_jahr" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$demomwstpzt_go_staf_jahr <- 2020
    }

    # Anzahl Dezimalen für die Beitragserhöhung der Massnahmen
    if(!"n_dec_bs_massn_round" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$n_dec_bs_massn_round <- 2
    }


    #------ Output --------------------------------------------------------#
    return(PARAM_GLOBAL = PARAM_GLOBAL)
```

4.4 Modul wrap_eo.R

Dokumentation zuletzt aktualisiert am 25.08.2025

Das Modul zur Berechnung des Finanzperspektivenmodells der EO wird in `run_eo.R` wie folgt aufgerufen:

```
tl_out_eo <- wrap_eo(tl_inp)
```

Am Anfang des Moduls werden die EO-spezifischen Vorberechnungen durchgeführt.

```
# Vorbereitende Berechnungen
tl_vorb_berechn <- wrap_eo_vorb_berechn(tl_inp = tl_inp)
```

Das Modul `wrap_eo_vorb_berechn` wird in Kapitel 4.5 beschrieben.

Als nächstes wird das Modul aufgerufen, dass die Hauptberechnungen zum Finanzperspektivenmodell, also die Berechnung der einzelnen Einnahmen- und Ausgabenpositionen, durchführt:

```
# Hauptberechnung: Einnahmen und Ausgaben GO
tl_hauptberechnung <- wrap_eo_hauptberechnung(tl_inp = tl_inp,
    tl_vorb_berechn = tl_vorb_berechn)
```

Eine Beschreibung des Moduls `wrap_eo_hauptberechnung` findet sich in Kapitel 4.6.

Mit dem nachfolgenden Codeblock werden die Auswirkungen der im Parameter-Ordner ausgewählten Massnahmen berechnet:

```
# Berechnung Massnahmen
tl_massnahmen <- wrap_eo_massnahmen(tl_inp = tl_inp, tl_vorb_berechn = tl_vorb_berechn,
    tl_hauptberechnung = tl_hauptberechnung)
```

Das Modul `wrap_eo_massnahmen` wird in Kapitel 4.9 beschrieben.

Der darauffolgende Codeblock dient dazu, aus den einzelnen Ausgaben- und Einnahmenprojektionen der Hauptberechnungen sowie der Massnahmenberechnungen die Bilanz der EO zu berechnen:

```
# Berechnung Umlageergebnis und Bilanz
tl_ergebnisse <- wrap_eo_ergebnisse(tl_inp = tl_inp, tl_vorb_berechn = tl_vorb_berechn,
    tl_massnahmen = tl_massnahmen)
```

Eine Beschreibung des Moduls `wrap_eo_ergebnisse` findet sich in Kapitel 4.10.

Der nachfolgende Codeblock dient dazu, die nominalen Ergebnisse des Finanzperspektivenmodells in reale Grössen umzurechnen – also Werte zu Preisen des gewählten Basisjahres, typischerweise dem Jahr der letzten EO-Abrechnung. In `mod_eo_postprocessing` werden auch Anhand der EO Bilanz verschiedene Indikatoren berechnet.

```
# Postprocessing (Ergebnisse zu konstanten Preisen
# berechnen)
tl_postprocessing <- mod_eo_postprocessing(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EO_BILANZ = tl_ergebnisse$EO_BILANZ, EO_MASSNAHMEN = tl_massnahmen$EO_MASSNAHMEN,
    DISKONTFAKTOR = tl_vorb_berechn$DISKONTFAKTOR)
```

Das Modul `mod_eo_postprocessing` wird in Kapitel 4.11 beschrieben.

Anschliessend können die Berechnungsergebnisse an das Finanzperspektivenmodell (bzw. das Modul `run_eo.R`) zurückgegeben werden. Dies erfolgt durch den folgenden abschliessenden Codeblock des Moduls:

```
# Output

c(tl_vorb_berechn, tl_hauptberechnung, tl_massnahmen, tl_ergebnisse,
    tl_postprocessing)
```

4.5 Modul wrap_eo_vorb_berechn.R

Dokumentation zuletzt aktualisiert am 02.07.2025

Das Modul `wrap_eo_vorb_berech_eo.R` führt Vorberechnungen für Projektionsvariablen durch. Es wird in `wrap_eo.R` wie folgt aufgerufen:

```
# Vorbereitende Berechnungen
tl_vorb_berechn <- wrap_eo_vorb_berechn(tl_inp = tl_inp)
```

In einem ersten Schritt werden die Daten der EO-Abrechnung und der AHV-Abrechnung eingelesen (vgl. Kapitel **??**). Die AHV-Abrechnung wird ebenfalls eingelesen, da diese später zur Projektion der Lohnbeiträge verwendet wird:

```
# Abrechnung
tl_abrechnung <- mod_abrechnung(tl_inp, versicherung = c("ahv",
    "eo"))
```

Als nächstes werden die Bevölkerungsdaten eingelesen unf aufbereitet (vgl. Kapitel 4.5.2):

```
# Bevoelkerung
BEVOELKERUNG <- mod_population(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
```

```
    BEV_BESTAND = tl_inp$BEV_BESTAND, BEV_SCENARIO = tl_inp$BEV_SCENARIO)
```

Im nächsten Schritt werden die Daten zu den Grenzgängern aufbereitet:

```
# Grenzgänger
tl_grenzgaenger <- mod_eo_grenzgaenger(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    BEVOELKERUNG = BEVOELKERUNG)
```

Als nächstes werden die Eckwerte zur wirtschaftlichen Entwicklung aufbereitet:

```
# Berechnung Eckwerte
tl_eckwerte <- mod_eckwerte(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE = tl_inp$ECKWERTE, LOHNINDEX = tl_inp$LOHNINDEX,
    PREISINDEX = tl_inp$PREISINDEX)
```

Das Modul `mod_eckwerte`(vgl. Kapitel 4.5.4) fügt historische Daten zur Lohn- und Preisentwicklung mit den Projektionsdaten zusammen und erstellt so eine Zeitreihe für die Entwicklung dieser volkswirtschaftlichen Eckwerte.

Danach wird der Deflator relativ zum aktuellen Jahr, was in der Modellterminologie als Diskontfaktor bezeichnet wird, berechnet:

```
# Berechnung Diskontfaktor
DISKONTFAKTOR <- mod_diskontfaktor(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE_EXTENDED = tl_eckwerte$ECKWERTE_EXTENDED)
```

Das Modul `mod_diskontfaktor` (vgl. Kapitel 4.5.5) berechnet den Diskontfaktor so, dass sämtliche nominellen Grössen durch Multiplikation mit dem Diskontfaktor zu Preisen vom Preisbasis-Jahr (typischerweise das Jahr der letzten Abrechnung) umgerechnet werden (dh. der Diskontfaktor ist 1 im Preisbasis-Jahr, und bei positiver Inflation <1 in der Zukunft und > 1 in der Vergangenheit).

Als Nächstes wird im Modul `mod_eomax` (vgl. Kapitel 4.5.6) die Entwicklung des Höchstbetrags der Gesamtentschädigung gemäss Art. 16a Abs. 2 EOG[24] berechnet. Ebenso werden die minimale Tagesentschädigung für Dienstleistende sowie die maximale Tagesentschädigung für andere Urlaubstypen (Mutterschaftsurlaub, Urlaub des anderen Elternteils, Betreuungs- und Adoptionsurlaub) bestimmt.

```
# Berechnung EO-MAX
EOMAX <- mod_eomax(EOMAX_hist = tl_inp$EOMAX_hist, ECKWERTE_EXTENDED =
↪   tl_eckwerte$ECKWERTE_EXTENDED,
    PARAM_GLOBAL = tl_inp$PARAM_GLOBAL)
```

Das nachfolgende Modul berechnet den Sozialversicherungs-Beitragssatz für sämtliche Lohnbeiträge.

```
# Beitragssaetze
BEITRAGSSAETZE <- mod_beitragssaetze(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    SV_BEITRAGSSATZ = tl_inp$SV_BEITRAGSSATZ)
```

Das Modul `mod_beitragssaetze` wird in Kapitel **??** beschrieben.

Als Nächstes werden die erwartete Rendite der nicht flüssigen Fondsanlagen sowie der Zinssatz auf die IV-Schuld bei der AHV[25] eingelesen (vgl. Kapitel 4.5.8).

```
# Berechnung der nominellen Renten auf den nichtflüssigen
# Fondsanlagen
ZINS <- mod_zins(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL, ZINS_RAW = tl_inp$ZINS_RAW,
```

[24] Art. 16a EOG auf Fedlex

[25] Dieser Zinssatz ist für die EO zwar irrelevant, wird jedoch benötigt, da das Modul für alle Sozialversicherungen verwendet wird.

```
    ECKWERTE_EXTENDED = tl_eckwerte$ECKWERTE_EXTENDED)
```

Zum Schluss werden die Berechnungen an das Finanzperspektivenmodell (bzw. an das Modul `wrap_eo.R`) zurückgegeben. Dies geschiet mit der folgenden abschliessenden Codezeile des Moduls:

```
#-- Output

c(tl_abrechnung, tl_grenzgaenger, tl_eckwerte, list(BEVOELKERUNG = BEVOELKERUNG,
    ZINS = ZINS, DISKONTFAKTOR = DISKONTFAKTOR, EOMAX = EOMAX,
    BEITRAGSSAETZE = BEITRAGSSAETZE))
```

4.5.1 Modul mod_abrechnung.R

Dokumentation zuletzt aktualisiert am 09.07.2025

Das Modul `mod_abrechnung` erstellt eine List (`tl_abrechung`), in der für jede ausgewählte Versicherung die jeweilige Abrechnungsdatentabelle zusammengestellt wird. In diesem Fall werden sowohl die AHV-Abrechnung, die für die Projektion der Beitragssummen verwendet wird (siehe Kapitel 4.7.1), als auch die EO-Abrechnung bereitgestellt.

Das Modul wird in `wrap_eo_vorb_berechn` folgend aufgerufen:

```
# Abrechnung
tl_abrechnung <- mod_abrechnung(tl_inp, versicherung = c("ahv",
    "eo"))
```

Die Auswahl und Kombination der Daten hängt davon ab, ob provisorische Abrechnungswerte verwendet werden sollen (gesteuert durch den Parameter `abr_prov` in `PARAM_GLOBAL`).

Als erstes wird eine leere Liste `tl_abrechnung`vorbereitet, in der später die Abrechnungsaten für jede Versicherung (AHV und EO) abgelegt werden. Dann wird für jede Versicherung in der Liste `verischerung` geprüft, ob der Parameter `abr_prov` aktiviert ist (d.h. auf `TRUE` gesetzt wurde). Ist dies der Fall, so werden die definitiven Abrechnungswerte (mit dem Suffix `_ABRECHNUNG_DEF`) für alle Jahre vor dem festgelegten Abrechnungsjahr (`jahr_abr`) übernommen. Zusätzlich werden die provisorischen Abrechnungswerte (`_ABRECHNUNG_PROV`) ausschließlich für das Abrechnungsjahr berücksichtigt. Beide Teilmengen – die definitiven Werte der Vorjahre und die provisorischen Werte des aktuellen Jahres – werden anschließend mittels `rbind` zu einer vollständigen Abrechnungstabelle zusammengeführt.

Ist der Parameter abr_prov hingegen nicht aktiviert, werden ausschließlich die definitiven Abrechnungswerte verwendet – und zwar für sämtliche Jahre, einschließlich des Abrechnungsjahres.

```
tl_abrechnung <- list()

for (vers in versicherung) {
    if (tl_inp$PARAM_GLOBAL$abr_prov) {
        # Fuege DEF und PROV nach Bedingungen hinzu
        abrechnung_def <- tl_inp[[paste0(toupper(vers), "_ABRECHNUNG_DEF")]] %>%
            filter(jahr < tl_inp$PARAM_GLOBAL$jahr_abr)
        abrechnung_prov <- tl_inp[[paste0(toupper(vers), "_ABRECHNUNG_PROV")]] %>%
            filter(jahr == tl_inp$PARAM_GLOBAL$jahr_abr)

        # Kombiniere DEF und PROV
        tl_abrechnung[[paste0(toupper(vers), "_ABRECHNUNG")]] <- rbind(abrechnung_def,
            abrechnung_prov)
    } else {
        # Nutze nur DEF, wenn abr_prov nicht aktiviert ist
        tl_abrechnung[[paste0(toupper(vers), "_ABRECHNUNG")]] <-
↪   tl_inp[[paste0(toupper(vers),
```

```
            "_ABRECHNUNG_DEF")]]
    }

}
```

Die Abrechnungsdaten pro Versicherung werden abschließend als Liste (`tl_abrechnung`) an das Modul `wrap_eo_vorb_berechn` zurückgegeben.

```
# Output
# ------------------------------------------------------------------------

return(tl_abrechnung)
```

4.5.2 Modul mod_population.R

Dokumentation zuletzt aktualisiert am 09.07.2025

Das Modul `mod_population` wird im Skript `wrap_eo_vorb_berechn` wie folgt aufgerufen:

```
# Bevoelkerung
BEVOELKERUNG <- mod_population(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    BEV_BESTAND = tl_inp$BEV_BESTAND, BEV_SCENARIO = tl_inp$BEV_SCENARIO)
```

Die Funktion verlangt, neben `PARAM_GLOBAL` die folgende Argumente:

- `BEV_BESTAND`: Beobachtete Bevölkerung
- `BEV_SCENARIO`: Bevölkerungsszenarien

Zu Beginn des Skriptes werden die Alter über 99 Jahren in den Bevölkerungszenarien zum Alter 99 hinzugezählt. Dies, da die Bevölkerungsszenarien nachfolgend mit den Bevölkerungsbeständen verbunden werden und in der verwendeten Bevölkerungsstatistik STATPOP Alter über 99 im Alter 99 zusammengefasst werden:

```
  # BEV_SCENARIO deckt Alter 0-120 ab, BEV_BESTAND nur 0-99. BEV_SCENARIO auf Alter 0-99
  ↪  einschrC$nken.
  BEV_SCENARIO <- BEV_SCENARIO %>%
    mutate(alt = ifelse(alt >= 100, 99, alt)) %>%  # Gruppiere alt >= 100 zu 99
    group_by(jahr, sex, nat, alt, scenario) %>%
    summarize(across(all_of(c(
      "bevanfangj",
      "bevendejahr",
      "geburt",
      "tod",
      "einbuergerung",
      "einwanderung",
      "auswanderung",
      "geburtmutter",
      "erwbev",
      "ept",
      "erwq",
      "erwqept",
      "frontaliers")), \(x) sum(x, na.rm = TRUE)), .groups = "drop")
```

Als nächstes werden die Bevölkerungsbestände gemäss dem Parameter `jahr_bev`gesetzt, sofern dieser Parameter in `PARAM_GLOBAL` spezifiert wurde (ansonsten werden einfach die aktuellsten Bestände verwendet):

```
# Select
if ("jahr_bev" %in% names(PARAM_GLOBAL)) {
```

```
    BEV_BESTAND <- BEV_BESTAND %>%
        mutate(erwbev = ifelse(jahr > PARAM_GLOBAL$jahr_bev +
            1, NA, erwbev), ept = ifelse(jahr > PARAM_GLOBAL$jahr_bev +
            1, NA, ept), erwq = ifelse(jahr > PARAM_GLOBAL$jahr_bev +
            1, NA, erwq), erwqept = ifelse(jahr > PARAM_GLOBAL$jahr_bev +
            1, NA, erwqept), frontaliers = ifelse(jahr > PARAM_GLOBAL$jahr_bev +
            1, NA, frontaliers), assures_facultatifs = ifelse(jahr >
            PARAM_GLOBAL$jahr_bev + 1, NA, assures_facultatifs),
            bevanfangj = ifelse(jahr > PARAM_GLOBAL$jahr_bev,
                NA, bevanfangj), bevendejahr = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, bevendejahr), geburt = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, geburt), tod = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, tod), einbuergerung = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, einbuergerung), einwanderung = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, einwanderung), auswanderung = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, auswanderung), bereinigung = ifelse(jahr >
                PARAM_GLOBAL$jahr_bev, NA, bereinigung), )
}
```

Konkret wird für die Erwerbsbevölkerung, die Grenzgänger und die freiwilig Versicherten der Bestand bis zum Jahr `jahr_bev+1` verwendet, und für die Wohnbevölkerungswerte der Bestand bis zu `jahr_bev`. Die unterschiedlichen Bestände sind wie folgt begründet: Die Bestände der Erwerbsbevölkerung, die Grenzgänger, und die freiwilig Versicherten sind jeweils im März des Folgejahres verfügbar. Die Wohnbevölkerungbestände jeweils erst im August des Folgejahres. Im Zeitraum zwischen März und August in welchen auch die Publikation der Finanzperspektiven fällt, sind also die Bestände der Erwerbsbevölkerung, die Grenzgänger und die freiwilig Versicherten für ein Jahr länger verfügbar als die Wohnbevölkerungbestände. Daher wird davon ausgegangen, dass sich `jahr_bev`auf die Wohnbevölkerung bezieht, und dass die weiteren Bestände ein Jahr länger verfügbar sind.

Als nächstes werde aus dem Dataframe `BEV_BESTAND`sämtliche VAraiblen ausgewählt, für welche Szenariodaten verfügbar sind (also alle Variablen ausser `c("bevanfangj", "geburt", tod", ëinbuergerung", ëinwanderung", "bereinigung", ßaisonniers", assures_facultatifs")`):

```
    # Faktoren zur Justierung der Szenarien-Bestaende auf die letzten beobachteten
    ↪  Bestaende berechnen
    fractions_bevoelkerung_scen <- BEV_BESTAND |>
        select(-c("bevanfangj", "geburt", "tod", "einbuergerung", "einwanderung",
        ↪  "bereinigung", "saisonniers", "assures_facultatifs")) |>
        mutate(quelle="bestand") |>
        bind_rows(BEV_SCENARIO |>
                      filter(scenario == PARAM_GLOBAL$bev_scenario) |>
                      select(-scenario) |>
                      select(-c("bevanfangj", "geburt", "tod", "einbuergerung",
                      ↪  "einwanderung", "geburtmutter")) |>
                      mutate(quelle="scenario")
        ) |>
        pivot_longer(cols = bevendejahr:frontaliers, names_to = "variable", values_to =
        ↪  "value") |>  # Transformiere die Daten in ein langes Format
        filter(!is.na(value)) |>
        group_by(variable) |>
        filter(jahr == max(jahr[quelle == "bestand"])) |>
        ungroup() |>
        arrange(sex, nat, alt, variable, quelle) |>
        pivot_wider(names_from = quelle, values_from = value) |>  # Trenne die Szenarien
        ↪  in Spalten
```

```
        mutate(
            fraction = ifelse(
                scenario == 0 | is.na(scenario),
                1,
                bestand / scenario
            )
        ) |>  # Berechne den Bruch
        select(sex, nat, alt, variable, fraction)  # Waehle die relevanten Spalten aus
```

Für diese Variablen wird nun der Faktor berechnet, um welchen der Wert des Szenarios in der jeweiligen Alter-Geschlecht-Nationalität Zelle vom in einem gegebenen Jahr vom zuletzt beobachteten Wert abweicht. Die Berechnnung des Faktors entfällt , wenn der erste Wert des Szenarios direkt an den zuletzt beobachteten Wert anknüpft (bspw. im Jahr 2025 starten die Wohnbevölkerungsszenarien im Jahr 2024, und die letzten beobachteten Werte reichen bis 2023, die Justierung entfällt also). Dieser Faktor wird dann später genutzt, um einen Strukturbruch im Übergang von den beobachteten Werten auf das Szenario, auf Grund von zwischen dem Szenarionerstellungszeitpunkt und dem letzten beobachteten Jahr vom Szenario abweichenden Entwicklungen, zu verhindern [26].

Als nächste wird der Skalierungsfaktor für die freiwillig Versischerten berechnet (die Population an freiwillig Versicherten ist lediglich für den AHV Finanzhaushalt relevant. Dieser Codeteil kann also bei Betrachtung des EO Finanzhaushalts ignoriert werden):

```
fraction_assures_facultatifs <- BEV_BESTAND |>
    select(jahr, sex, nat, alt, bevendejahr, assures_facultatifs) |>
    mutate(fraction_assures_facultatifs = ifelse(bevendejahr !=
        0, assures_facultatifs/bevendejahr, 1)) |>
    filter(!is.na(assures_facultatifs), !is.na(bevendejahr)) |>
    filter(jahr %in% c(min(jahr), max(jahr))) %>%
    group_by(sex, nat, alt) |>
    summarize(fraction_assures_facultatifs_maxjahr =
    ↪  max(fraction_assures_facultatifs[jahr ==
        max(jahr)], na.rm = TRUE), fraction_assures_facultatifs_minjahr =
        ↪  max(fraction_assures_facultatifs[jahr ==
        min(jahr)], na.rm = TRUE), .groups = "drop")
```

Da für ie freiwillig Versichterten keine Szenarien existieren, wird der Bestand an freiwillig Versichterten anhand des Anteils deren an der Wohnbevölkerung im letzten beobachtete Jahr (`max(jahr)`), multipliziert mit der Wohnbevölkerungsentwicklung, fortgeschrieben. In obigem Codeblocak wird nur der Faktor berechnet, mit welchem (im Codeblock unten) die Wohnbevölkerung dann multipliziert werden soll. Genau gleich wird die Anzahl an freiwillig Versicherten für die Jahre, in welchen keine Registerdaten existieren (also die Jahre vor `min(jahr)`), anhand des Anteils der frewillig Versicherten in `min(jahr)` multipliziert mit der Wohnbevölkreung des entsprechenden Jahres geschätzt.

Analog wird auch ein Skalierungsfaktor für die Grenzgänger berechnet, wobei hier Szenarien existieren, und daher nur für die Jahre vor `min(jahr)` ein Faktor berechnet werden muss.

```
fraction_frontaliers <- BEV_BESTAND |>
    select(jahr, sex, nat, alt, bevendejahr, frontaliers) |>
    mutate(fraction_frontaliers = ifelse(bevendejahr != 0, frontaliers/bevendejahr,
        1)) |>
    filter(!is.na(frontaliers)) |>
    filter(jahr %in% min(jahr)) |>
    select(sex, nat, alt, fraction_frontaliers)
```

[26] Mathematisch ist das Vorgehen äquivalent dazu, die Wachstumsraten der Szenarien nach Alter-Geschlecht-Nationalität Zelle auf den letzten beobachteten Wert anzuwenden

Zum Schluss wird das Dataframe `BEVOELKERUNG` erstellt, welches die Daten zu Bevölkerungsbeständen und -Szenarien in einer durgehenden Zeitreihe kombiniert:

```
    BEVOELKERUNG <- BEV_BESTAND |>
        select(-c("bevanfangj", "geburt", "tod", "einbuergerung", "einwanderung",
        ↪  "bereinigung", "saisonniers", "assures_facultatifs")) |>
        mutate(quelle = "bestand") |>
        bind_rows(
            BEV_SCENARIO |>
                filter(scenario == PARAM_GLOBAL$bev_scenario) |>
                select(-scenario) |>
                select(-c("bevanfangj", "geburt", "tod", "einbuergerung", "einwanderung",
                ↪  "geburtmutter")) |>
                mutate(quelle = "scenario")
        ) |>
        pivot_longer(cols = bevendejahr:frontaliers, names_to = "variable", values_to =
        ↪  "value") |>  # Transformiere die Daten in ein langes Format
        filter(!is.na(value)) |>
        group_by(variable) |>
        filter(quelle == "bestand" | jahr > max(jahr[quelle == "bestand"])) |>
        ungroup() |>
        left_join(fractions_bevoelkerung_scen, by = c("sex", "nat", "alt", "variable"))
        ↪  |>  # Verknuepfe mit fractions
        mutate(value = ifelse(quelle == "scenario", value * fraction, value)) |>  #
        ↪  Multipliziere den Wert mit der entsprechenden fraction
        select(-fraction, -quelle) |>  # Entferne die fraction-Spalte
        filter(!is.na(value)) |>
        pivot_wider(names_from = "variable", values_from = "value") |>  # Transformiere
        ↪  zurueck ins weite Format
        left_join(BEV_BESTAND |> select(jahr, sex, nat, alt, saisonniers,
        ↪  assures_facultatifs)) |>
        left_join(fraction_assures_facultatifs) |>
        left_join(fraction_frontaliers) |>
        mutate(
            saisonniers = ifelse(jahr >= 2000 & is.na(saisonniers), 0, saisonniers),
            assures_facultatifs =  case_when(
                    jahr < 2000 & is.na(assures_facultatifs) ~ bevendejahr *
↪  fraction_assures_facultatifs_minjahr,
                    jahr > 2000 & is.na(assures_facultatifs) ~ bevendejahr *
↪  fraction_assures_facultatifs_maxjahr,
                    TRUE ~ assures_facultatifs
                ),
            frontaliers = ifelse(jahr<2000 & is.na(frontaliers), bevendejahr *
            ↪  fraction_frontaliers, frontaliers),
            auswanderung = auswanderung
        ) |>
        select(-fraction_assures_facultatifs_maxjahr,
        ↪  -fraction_assures_facultatifs_minjahr, -fraction_frontaliers) |>
        arrange(sex, nat, alt, jahr)
```

Hierfür werden in einem ersten Teil (Teil bis `pivot_wider(names_from = "variable", values_from = "values") |> )` die Werte des Bevölkerungsszenarios an die Bevölkerung-Bestandesdaten, für welche Szenario existieren, angehängt und jeweils mit den in `fractions_bevoelkerung_scen` ermittelten Skalierungsfaktoren multipliziert. Danach werden die Bevölkerungs-Bestandesdaten für die frewillig Versicherten und die Grenzgänger hinzugefügt, und es werden die fehlende Werte für die freiwillig Versicherten und die Grenz-

gänger durch Multiplikation der in `fraction_assures_facultatifsund` `fraction_frontaliers` Anteile mit der Wohnbevölkerung im jeweiligen Jahr berechnet. Die letzte if-Bedingung für die Auswanderung kann bei Betrachtung des EO FInanzhaushalts wiederum ignoriert werden, da diese mnur für den AHV Finanzhaushalt relevant ist.

Somit ist die Berechnung des `BEVOELKERUNG` Dataframes abgeschlossen, und ebendies kann an das Modul `wrap_eo_vorb_berechn` zurückgegeben werden:

```
# --- Output
# ------------------------------------------------------------------

return(BEVOELKERUNG = BEVOELKERUNG)
```

4.5.3 Modul mod_eo_grenzgaenger.R

Dokumentation zuletzt aktualisiert am 21.08.2025

Das Modul `mod_eo_grenzgaenger` dient dazu, die Daten zu den Grenzgängerinnen und Grenzgängern nach den im Modell zur Berechnung der finaziellen Perspektiven der EO verwendeten Alterskategorien aufzubereiten. Es wird in `wrap_eo_vorb_berechn` wie folgt aufgerufen:

```
# Grenzgänger
tl_grenzgaenger <- mod_eo_grenzgaenger(BEVOELKERUNG = BEVOELKERUNG)
```

Zuerst wählt das Modul die Grenzgängerinnen und Grenzgänger nach Alter, Nationalität und Geschlecht für jedes Jahr aus. Anschließend wird eine zusätzliche Variable erstellt, die die jeweilige Alterskategorie enthält. Schließlich wird die Anzahl der Grenzgängerinnen und Grenzgänger nach Alterskategorie, Geschlecht und Jahr aggregiert.

```
FRONTALIERS <- BEVOELKERUNG |>
    select(jahr, sex, nat, alt, frontaliers) |>
    filter(alt >= 15 & alt <= 49) |>
    mutate(kat = case_when(alt >= 15 & alt <= 19 ~ 1, alt >=
        20 & alt <= 24 ~ 2, alt >= 25 & alt <= 29 ~ 3, alt >=
        30 & alt <= 34 ~ 4, alt >= 35 & alt <= 39 ~ 5, alt >=
        40 & alt <= 44 ~ 6, alt >= 45 & alt <= 49 ~ 7)) |>
    group_by(kat, jahr, sex) |>
    summarize(anzahl_grenzganger = sum(frontaliers))
```

Diese aggregierten Daten werden in einem Data Frame mit dem Namen `FRONTALIERS` gespeichert und an das Modul für die vorbereitenden Berechnungen weitergegeben.

```
return(list(FRONTALIERS = FRONTALIERS))
```

4.5.4 Modul mod_eckwerte.R

Dokumentation zuletzt aktualisiert am 21.08.2025

Das Modul `mod_eckwerte.R` wird im Skript `wrap_eo_vorb_berechn` wie folgt aufgerufen:

```
# Berechnung Eckwerte
tl_eckwerte <- mod_eckwerte(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE = tl_inp$ECKWERTE, LOHNINDEX = tl_inp$LOHNINDEX,
    PREISINDEX = tl_inp$PREISINDEX)
```

Die Funktion verlangt die folgende Argumente:

- `ECKWERTE`: Dataaframe mit allen Projektionen zu den Eckwerten zur wirtsschaftlichen Entwicklung (insbesonedere der Lohn- und Presientwicklung) gemäss ESTV;

- `LOHNINDEX`: Historischer Lohnindex;
- `PREISINDEX`: Historischer Preisindex.

Zu beginn werden die gemäss `PARAM_GLOBAL.R` spezifierten Eckwerte extrahiert:

```
ECKWERTE_SCENARIO <- ECKWERTE %>%
    filter(id == PARAM_GLOBAL$id_eckwerte) %>%
    filter(jahr <= PARAM_GLOBAL$jahr_ende)
```

Danach werden drei Sicherheitsüberprüfungen durchgeführt, die sicherstellen, dass die Eckwerte korrekt spezifiziert sind:

```
if (PARAM_GLOBAL$jahr_abr < (min(ECKWERTE_SCENARIO$jahr) - 1)) {
    stop(paste0("ECKWERTE_SCENARIO empty for year ", PARAM_GLOBAL$jahr_abr))
}

if (min(ECKWERTE_SCENARIO$jahr) < PARAM_GLOBAL$jahr_abr) {
    stop(paste0("Eckwerte scenario PARAM_GLOBAL$id_eckwerte = ",
        PARAM_GLOBAL$id_eckwerte, " does not match PARAM_GLOBAL$jahr_abr = ",
        PARAM_GLOBAL$jahr_abr))
}

if (max(LOHNINDEX$jahr) < min(ECKWERTE_SCENARIO$jahr) - 1) {
    stop(paste0("Die Lohn-Daten in LOHNINDEX.csv gehen nur bis zum Jahr ",
        max(LOHNINDEX$jahr), ", und die ausgewählten Eckwerte ",
        PARAM_GLOBAL$id_eckwerte, " starten erst im Jahr ", min(ECKWERTE_SCENARIO$jahr),
        ". Zusätzlich das Jahr ", min(ECKWERTE_SCENARIO$jahr),
        " im Ordner 01_raw_data zu den Eckwerten ", PARAM_GLOBAL$id_eckwerte,
        " hinzufügen (copy-pasten von letzter Eckwerte-Version)."))
}
```

Danach werden für die Jahre ab 1979 bis zum Jahr vor dem kleinsten Jahr im Datensatz `ECKWERTE_SCENARIO` die jährlichen Wachstumsraten des Lohnindex (`lohn`) und des Preisindex (`preis`) in Prozent berechnet. Das Ergebnis wird als neuer Datensatz `ECKWERTE_HIST` gespeichert.

```
# Bereitstellen Eckwerte bis vor dem ersten Jahr in
# Eckwerte Scenario --------

ECKWERTE_HIST <- LOHNINDEX %>%
    full_join(PREISINDEX, by = "jahr") %>%
    filter(jahr >= 1979, jahr < min(ECKWERTE_SCENARIO$jahr)) %>%
    mutate(lohn = 100 * (li - lag(li))/lag(li), preis = 100 *
        (lik_basis_1977 - lag(lik_basis_1977))/lag(lik_basis_1977)) %>%
    dplyr::select(jahr, lohn, preis)
```

Als nächstes wird die vorangehend ausgewählte Eckwerte-Projektion ausgewählt:

```
# Bereitstellen Eckwerte des ausgewählten Szenarios
# --------------------------

ECKWERTE_GO <- ECKWERTE_SCENARIO %>%
    dplyr::select(jahr, lohn, preis)
```

Nun wird die Eckwerte Projektion für die fehleneden Jahre bis zum Ende des Projektionshorizonts erstellt:

```
  # Konstruktion der Eckwerte ab dem ausgewählten Szenario ----------------------
```

```
  if (last(ECKWERTE_GO$jahr) < PARAM_GLOBAL$jahr_ende) {
    ECKWERTE_PR <- crossing(
      jahr = (last(ECKWERTE_GO$jahr) + 1):PARAM_GLOBAL$jahr_ende,
      tail(
        ECKWERTE_GO %>%
          dplyr::select(-jahr),
        1
      )
    )
```

Hierbei wird zuerst geprüft, ob das letzte Jahr mit einer verfügbaren Eckwerte Projektion (`last(ECKWERTE_GO$jahr)`) kleiner ist als das letzte Jahr des Projektionshorizonts `PARAM_GLOBAL$jahr_ende`. Wenn dies der Fall ist, wird die Eckwerte Projektion vom Jahr (`last(ECKWERTE_GO$jahr)`) für alle Jahre bis `PARAM_GLOBAL$jahr_ende` verwendet. Stand Sommer 2025 sind Eckwerteprojektionen bis 2034 verfügbar, und das letzte Jahr des Projektionshorizonts ist 2075, d.h. die Eckwerte-Projektion für das Jahr 2034 wird für alle Jahre bis 2075 verwendet.

Als nächstes wird das Dataframe `ECWERTE_EXTENDED` aus den historischen und den projezierten Eckwerten gebildet, wobei das Datafrmae `ECKWERTE_PR` natürlich nur verwendet wird, wenn obige `if`-bedingung `TRUE` ist:

```
    ECKWERTE_EXTENDED <- bind_rows(
      ECKWERTE_HIST,
      ECKWERTE_GO,
      ECKWERTE_PR
    )
  } else {
    ECKWERTE_EXTENDED <- bind_rows(
      ECKWERTE_HIST,
      ECKWERTE_GO
    )
  }
```

Somit ist die Berechnung des `ECKWERTE_SCENARIO` sowie des `ECKWERTE_EXTENDED` Dataframes abgeschlossen, wodurch dies an das Modul `wrap_eo_vorb_berechn.R` zurück gegeben werden können:

```
# Output
# ------------------------------------------------------------------------

return(list(ECKWERTE_SCENARIO = ECKWERTE_SCENARIO, ECKWERTE_EXTENDED =
↪   ECKWERTE_EXTENDED))
```

4.5.5 Modul mod_diskontfaktor.R

Dokumentation zuletzt aktualisiert am 25.08.2025

`mod_diskontfaktor` wird im Skript `wrap_eo_vorb_berechn` wie folgt aufgerufen:

```
# Berechnung Diskontfaktor
DISKONTFAKTOR <- mod_diskontfaktor(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE_EXTENDED = tl_eckwerte$ECKWERTE_EXTENDED)
```

Die Funktion verlangt die folgende Argumente:

```
mod_diskontfaktor <- function(PARAM_GLOBAL,
                              ECKWERTE_EXTENDED
                              )
```

- `PARAM_GLOBAL`: Parameter, die zur Berechnung der Finanzperspektiven verwendet werden. Sie stammen aus der Datei `PARAM_GLOBAL.csv` sowie aus dem Modul `mod_eo_param_global.R` (vgl. 4.3).
- `ECKWERTE_EXTENDED`: Zeitreihe mit der historischen und der projezierten Lohn- und Preisentwicklung (vgl. 4.5.4).

Zu Beginn des Moduls wird überprüft, ob das Jahr, welches als Preisbasis gewählt wurde, innerhalb der jahre liegt, für welche in `ECKWERTE_EXTENDED` Werte für die Preisentwicklung vorhanden sind. Falls das nicht der Fall is wird die Ausführung angehalten und eine Fehlermeldung ausgegeben:

```
# check if jahr_preisbasis exists
if (!"jahr_preisbasis" %in% names(PARAM_GLOBAL)) {
    stop("A value must be provided for jahr_preisbasis in PARAM_GLOBAL")
}

# check for NA values
ECKWERTE_EXTENDED$preis[1] <- 0
if (any(is.na(ECKWERTE_EXTENDED$jahr))) {
    stop("There are NA values in ECKWERTE_EXTENDED$jahr")
}
if (any(is.na(ECKWERTE_EXTENDED$preis))) {
    stop("There are NA values in ECKWERTE_EXTENDED$preis")
}

# check if jahr_preisbasis is in the range provided by
# ECKWERTE_EXTENDED
jahr_preisbasis <- PARAM_GLOBAL$jahr_preisbasis
min_jahr <- min(ECKWERTE_EXTENDED$jahr)
max_jahr <- max(ECKWERTE_EXTENDED$jahr)
if (!PARAM_GLOBAL$jahr_preisbasis %in% min_jahr:max_jahr) {
    msg <- paste0("The value for jahr_preisbasis must be in the interval ",
        "[", min_jahr, ", ", max_jahr, "].")
    stop(msg)
}
```

Zunächst wird aus den erweiterten Eckwerten der Diskontfaktor berechnet, indem ein kumulativer Preisindex erstellt und dieser anschliessend auf das Basisjahr normiert wird.

```
# calculate DISKONTFAKTOR
DISKONTFAKTOR <- ECKWERTE_EXTENDED |>
    dplyr::mutate(preisindex = cumprod(1 + preis/100)) |>
    dplyr::mutate(diskontfaktor = preisindex[jahr == jahr_preisbasis]/preisindex) |>
    dplyr::select(jahr, diskontfaktor)
```

Somit ist die Berechnung des `DISKONTFAKTOR` Dataframes abgeschlossen, wodurch dieses an das Modul `wrap_eo_vorb_berechn` zurückgegeben werden kann:

```
return(DISKONTFAKTOR = DISKONTFAKTOR)
```

4.5.6 Modul mod_eomax.R

Dokumentation zuletzt aktualisiert am 25.08.2025

Dieses Modul projiziert die Entwicklung der minimalen und maximalen Beträge der Taggelder der EO gemäss dem Gesetz.

Die Funktion `mod_eomax` verlangt die folgenden Argumente:

```
mod_eomax <- function(EOMAX_hist,
                      ECKWERTE_EXTENDED,
                      PARAM_GLOBAL
                      )
```

- `EOMAX_hist`: Zeitreihe mit dem historischen Höchstbetrag der Gesamtentschädigung, genannt `gesamtentsch` und der maximalen Grundentschädigung für Eltern oder Dienstleistende ohne Kinder, genannt `eomax`.
- `ECKWERTE_EXTENDED`: Zeitreihe mit der historischen und der projezierten Lohn- und Preisentwicklung (vgl. 4.5.4)
- `PARAM_GLOBAL`: Parameter, die zur Berechnung der Finanzperspektiven verwendet werden. Sie stammen aus der Datei `PARAM_GLOBAL.csv` sowie aus dem Modul `mod_eo_param_global.R` (vgl. 4.3).

Aus `PARAM_GLOBAL` benutzten wir in dem Modul:

```
TCMAX <- PARAM_GLOBAL$tcmax
jahr_adapt_eomax <- PARAM_GLOBAL$jahr_adapt_eomax
jahr_end <- PARAM_GLOBAL$jahr_ende
```

- `TCMAX`: beträgt 12 % und entspricht der notwendigen Erhöhung des Lohnindexes, um eine Anpassung von den minimalen und maximalen EO-Beträgen vorzunehmen.
- `jahr_adapt_max`: das letzte Jahr, in dem die maximalen und minimalen EO-Beträge angepasst wurden. Derzeitiger Wert: 2023.
- `jahr_end`: das letzte Jahr, für welches Projektionen berechnet werden. Derzietiger Wert: 2075

Erstens wird die Datenstruktur vorbereitet, die zur Berechnung der Projektionen der maximalen und minimalen Beträge verwendet wird. Genauer gesagt werden die historischen Daten des Höchstbetrags der Gesamtentschädigung dem historischen und projizierten SLI beigefügt. Der Höchstbetrag der Gesamtentschädigung entspricht zunächst für alle Jahre nach `jahr_adapat_eomax` dem Betrag von `jahr_adapat_eomax` Diese werden später durch Projektionen ersetzt.

```
EOMAX <- EOMAX_hist |>
    full_join(LOHNINDEX |>
        select(jahr, lohnindex), by = "jahr") |>
    filter(jahr >= jahr_adapt_eomax) |>
    mutate(gesamtentsch = first(gesamtentsch))
```

Zur Berechnung der projizierten Höchstbeträge brauchen wir den SLI-Wert des Jahres, in dem die Höchstbeträge zuletzt erhöht wurden. Dieser wird extrahiert und als Ausgangswert für den Lohnindex (`lohnindex_t`) gespeichert. Er dient als Referenzwert für die folgenden Berechnungen.

```
lohnindex_t <- EOMAX |>
    filter(jahr == jahr_adapt_eomax) |>
    pull(lohnindex)
```

Mittels einer iterativen Schleife können wir nun die nächsten Maximalbeträge der Tagegelder vorhersagen.

```
for (i in jahr_adapt_eomax:jahr_end) {
    k <- i - jahr_adapt_eomax + 1
    diff <- (EOMAX[k, "lohnindex"] - lohnindex_t)/lohnindex_t
    if (diff >= TCMAX) {
        EOMAX[k, "gesamtentsch"] <- round((1 + diff) * EOMAX[k,
            "gesamtentsch"]/5) * 5  # rounds gesamtentsch to the nearest multiple of 5
        lohnindex_t <- EOMAX[k, "lohnindex"]
```

```
        EOMAX[k:jahr_end, "gesamtentsch"] <- EOMAX[k, "gesamtentsch"]
    }
}
```

Diese Schleife iteriert über jedes Jahr von `jahr_adapt_eomax` bis `jahr_end`.

`k` ist der Index, der verwendet wird, um die Zeilen im `EOMAX`-Datensatz zu adressieren. Dieser Index wird als Differenz zwischen dem aktuellen Jahr `i` und `jahr_adapt_eomax` berechnet, um den richtigen Index für die Zeile im Datensatz zu finden.

Danach wird die prozentuale Veränderung des Lohnindex im Vergleich zum Referenzjahr `jahr_adapt_eomax` berechnet.

```
diff <- (EOMAX[k, "lohnindex"] - lohnindex_t)/lohnindex_t
```

Falls die Bedingung `(diff >= TCMAX)` erfüllt ist, wird der `gesamtentsch`-Wert für das Jahr `i` angepasst. Dabei wird die Veränderung des Lohnindex (`diff`) auf den aktuellen Wert von `gesamtentsch` angewendet, und der Wert wird auf das nächste Vielfache von 5 gerundet.

```
EOMAX[k, "gesamtentsch"] <- round((1 + diff) * EOMAX[k, "gesamtentsch"]/5) *
    5
```

Wurde eine Anpassung vorgenommen, wird der Lohnindex für das Jahr `i` als neuer Referenzwert (`lohnindex_t`) gespeichert. Der gesamtentsch-Wert wird auf den neuen Wert gesetzt, der in k berechnet wurde.

```
lohnindex_t <- EOMAX[k, "lohnindex"]
EOMAX[k:jahr_end, "gesamtentsch"] <- EOMAX[k, "gesamtentsch"]
```

Schliesslich werden die in `EOMAX` enthaltenen Prognosen und die historischen Daten (`EOMAX_hist`) in einem einigen Datensatz zusammengefasst und die Projektionen für `eomax` und `eomin` berechnet. Es wird ausserdem eine Dummy-Variable erstellt, die den Wert 1 annimmt, wenn die Maximal- und Minimalbeträge angepasst werden, und sonst 0 ist.

```
EOMAX_prev <- EOMAX_hist |>
    filter(jahr <= jahr_abr) |>
    full_join(EOMAX_prev |>
        filter(jahr > jahr_abr) |>
        mutate(eomax = ceiling(0.8 * gesamtentsch)) |>
        select(jahr, gesamtentsch, eomax) |>
        filter(jahr <= jahr_end), by = c("jahr", "gesamtentsch",
        "eomax")) |>
    mutate(eomin = ceiling(0.25 * gesamtentsch)) |>
    mutate(wr_eomax = (gesamtentsch - lag(gesamtentsch))/lag(gesamtentsch),
        dummy_eomax = if_else(wr_eomax != 0, 1, 0))
```

Die Ergebnisse werden im Dataframe `EOMAX` gespeichert und somit dem Modul `wrap_eo_vorb_berechn` zurückgegeben:

```
return(EOMAX = EOMAX)
```

4.5.7 Modul mod_beitragssaetze.R

Dokumentation zuletz aktualisiert am 25.08.2025

Das Modul `mod_beitragssaetze` wird in `wrap_eo_vorb_berechn` wie folgt aufgerufen:

```
# Beitragssaetze
BEITRAGSSAETZE <- mod_beitragssaetze(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    SV_BEITRAGSSATZ = tl_inp$SV_BEITRAGSSATZ)
```

Die Funktion verwendet neben `PARAM_GLOBAL` das folgende Dataframe:

- `SV_BEITRAGSSATZ`: Dataframe mit allen historischen Beitragssätzen für Lohnbeiträge insbesondere auch dem EO-Beitragssatz.

Das Modul `mod_beitragssaetze.R` extrahiert den letzten verfügbaren gültigen historischen Beitragssatz und kopiert diesen für alle Jahre bis zum Ende des Projektionshorizonts. Danach fügt er die Zeitreihe mit den projezierten projizierten Beitragssätzen mit den historischen Beitragssätzen zusammen:

```
#------ Bereinigen und Gesamtsatz berechnen --------------------------#
    BEITRAGSSAETZE       <- SV_BEITRAGSSATZ %>%
      fill(-jahr, .direction = "down") %>%  #Potentiell fehlende Zellen mit letztem
      ↪  Bekannten Wert ersetzten
    select(jahr, ahv_vs_ag, iv_vs_ag, eo_vs_ag, alv_vs_ag) %>%
      mutate(across(c(-jahr), list(~if_else(is.na(.), 0, .)), .names = '{.col}') ) %>%
      # Fortschreibung der Saetze bis jahr_ende
      # keine Aenderung der Saetze bekannt, Werte aus letzten bekannten Jahr werden
      ↪  weitergefuehrt
      bind_rows(tibble(jahr = seq(max(SV_BEITRAGSSATZ$jahr, na.rm = TRUE) + 1,
      ↪  PARAM_GLOBAL$jahr_ende))) %>%
      fill(everything(), .direction = "down") %>%
      # Gesamtsatz, relevant fC<r AG-Anteil
      mutate(sv_vs_ag = ahv_vs_ag + iv_vs_ag + eo_vs_ag + alv_vs_ag)
```

Somit ist die Berechnung des `BEITRAGSSAETZE` Dataframes abgeschlossen, wodurch dieses an das Modul `wrap_eo_vorb_berechn.R` zurückgegeben werden kann:

```
#------ Output -------------------------------------------------------#
return(BEITRAGSSAETZE = BEITRAGSSAETZE)
```

4.5.8 Modul mod_zins.R

Dokumentation zuletzt aktualisiert am 25.08.2025

`mod_zins.R` bestimmt - ausgehend von der gewählten Preisentwicklung - für alle Jahre des Projektionszeitraums die Zinssätze, mit denen der EO-Fonds verzinst wird. Es wird im Skript `wrap_eo_vorb_berechn.R` wie folgt aufgerufen:

```
# Berechnung der nominellen Renten auf den nichtflüssigen
# Fondsanlagen
ZINS <- mod_zins(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL, ZINS_RAW = tl_inp$ZINS_RAW,
    ECKWERTE_EXTENDED = tl_eckwerte$ECKWERTE_EXTENDED)
```

Die Funktion verwendet neben `PARAM_GLOBAL` die folgende Dataframes:

- `ECKWERTE_EXTENDED`: Zeitreihe mit der historischen und der projizierten Lohn- und Preisentwicklung (vgl. ref{mod_eckwerte}),
- `ZINS_RAW`: Zinssätze für den EO-Fonds (und die IV-Schulden [27]).

Das Modul `mod_zins.R` extrahiert den Realzins für di nichtflüssigen Fondsanlagen und generiert aus diesem durch Addition der Inflation den Nominalzins. Zudem schreibt es die Zinssätze aus dem letzten Jahr in `ZINS_RAW` bis zum Ende des Prognosehorizonts fort:

[27] Nicht für die Berechnung der EO FHH verwendet, aber dieses Modul wird für alle Versicherungen genutzt.

```
ZINS <- ZINS_RAW |>
    filter(jahr >= PARAM_GLOBAL$jahr_abr) |>
    left_join(ECKWERTE_EXTENDED |>
        select(jahr, preis), by = "jahr") |>
    mutate(nominal_zins_fonds = (real_zins_fonds + preis)/100,
        nominal_zins_ivschuld = nominal_zins_ivschuld/100) |>
    select(jahr, nominal_zins_fonds, nominal_zins_ivschuld)

ZINS <- ZINS %>%
    bind_rows(tibble(jahr = seq(max(ZINS$jahr) + 1, PARAM_GLOBAL$jahr_ende),
        nominal_zins_fonds = ZINS %>%
            filter(jahr == max(jahr)) %>%
            pull(nominal_zins_fonds), nominal_zins_ivschuld = ZINS %>%
            filter(jahr == max(jahr)) %>%
            pull(nominal_zins_ivschuld))) %>%
    arrange(jahr)
```

Somit ist die Berechnung des `ZINS` Dataframes abgeschlossen, wodurch dieses an das Modul `wrap_eo_vorb_berechn.R` zurückgegeben werden kann:

```
#------ Output ------------------------------------------------------#

return(ZINS = ZINS)
```

4.6 Modul wrap_eo_hauptberechung.R

Dokumentation zuletzt aktualisiert am 02.07.2025

Das Modul zu den Hauptberechnung wird in `wrap_eo.R` wie folgt aufgerufen:

```
# Hauptberechnung: Einnahmen und Ausgaben GO
tl_hauptberechnung <- wrap_eo_hauptberechnung(tl_inp = tl_inp,
    tl_vorb_berechn = tl_vorb_berechn)
```

Als Erstes wird das Modul zur Berechnung der Einnahmenporjetktionen aufgerufen (vgl. Kapitel 4.7):

```
# Berechnung der Einnahmen der EO
tl_eo_einnahmen <- mod_eo_einnahmen(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    BEVOELKERUNG = tl_vorb_berechn$BEVOELKERUNG, IK = tl_inp$IK,
    ECKWERTE_EXTENDED = tl_vorb_berechn$ECKWERTE_EXTENDED, ECKWERTE_SCENARIO =
    ↪  tl_vorb_berechn$ECKWERTE_SCENARIO,
    EO_ABRECHNUNG = tl_vorb_berechn$EO_ABRECHNUNG, AHV_ABRECHNUNG =
    ↪  tl_vorb_berechn$AHV_ABRECHNUNG,
    DISKONTFAKTOR = tl_vorb_berechn$DISKONTFAKTOR, BEV_BESTAND = tl_inp$BEV_BESTAND,
    BEITRAGSSAETZE = tl_vorb_berechn$BEITRAGSSAETZE, ARBEITSLOSENQUOTE =
    ↪  tl_inp$ARBEITSLOSENQUOTE)
```

Darauffolgend wird das Modul zur Berechnung der Ausgabenprojektionen aufgerufen (vgl. Kapitel 4.8):

```
# Berechnen der Ausgaben der EO
tl_eo_ausgaben <- mod_eo_ausgaben(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    REGISTER_MSU = tl_inp$REGISTER_MSU, REGISTER_VSU = tl_inp$REGISTER_VSU,
    REGISTER_DIENST = tl_inp$REGISTER_DIENST, REGISTER_BEU = tl_inp$REGISTER_BEU,
    REGISTER_ADOPT = tl_inp$REGISTER_ADOPT, LEBENDGEBURTEN_FRAU =
    ↪  tl_inp$LEBENDGEBURTEN_FRAU,
    LEBENDGEBURTEN_MANN = tl_inp$LEBENDGEBURTEN_MANN, BEV_SCENARIO = tl_inp$BEV_SCENARIO,
```

```
    FRONTALIERS_SCEN = tl_vorb_berechn$FRONTALIERS_SCEN, BEVOELKERUNG =
    ↪  tl_vorb_berechn$BEVOELKERUNG,
    ECKWERTE_EXTENDED = tl_vorb_berechn$ECKWERTE_EXTENDED, EOMAX = tl_vorb_berechn$EOMAX,
    EO_ABRECHNUNG_DEF = tl_inp$EO_ABRECHNUNG_DEF, EO_AUSGABEN_ANTEILE =
    ↪  tl_inp$EO_AUSGABEN_ANTEILE,
    DISKONTFAKTOR = tl_vorb_berechn$DISKONTFAKTOR, STRUKTURFAKTOR =
    ↪  tl_eo_einnahmen$STRUKTURFAKTOR)
```

Zum Schluss werden die Ausgaben- und Einnahmenprojektionen an das Finanzperspektivenmodell (bzw. an das Modul `wrap_eo.R`) zurückgegeben. Dies geschieht mit der folenden abschliessenden Codezeile des Moduls:

```
# Output
return(c(tl_eo_ausgaben, tl_eo_einnahmen))
```

4.7 Modul mod_eo_einnahmen.R

Dokumentation zuletzt aktualisiert am 02.07.2025

Das Modul zu den Einnahmen wird in `wrap_eo_hauptberechnung.R` wie folgt aufgerufen:

```
tl_eo_einnahmen <- mod_eo_einnahmen(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    BEVOELKERUNG = tl_vorb_berechn$BEVOELKERUNG, IK = tl_inp$IK,
    ECKWERTE_EXTENDED = tl_vorb_berechn$ECKWERTE_EXTENDED, ECKWERTE_SCENARIO =
    ↪  tl_vorb_berechn$ECKWERTE_SCENARIO,
    EO_ABRECHNUNG = tl_vorb_berechn$EO_ABRECHNUNG, AHV_ABRECHNUNG =
    ↪  tl_vorb_berechn$AHV_ABRECHNUNG,
    DISKONTFAKTOR = tl_vorb_berechn$DISKONTFAKTOR, BEV_BESTAND = tl_inp$BEV_BESTAND,
    BEITRAGSSAETZE = tl_vorb_berechn$BEITRAGSSAETZE, ARBEITSLOSENQUOTE =
    ↪  tl_inp$ARBEITSLOSENQUOTE)
```

Zuerst wird in `mod_eo_einnahmen`das Modul zur Berechnung der Einnahmen aus Beiträgen von Arbeitgebern und Versicherten aufgerufen (vgl. Kapitel 4.7.1):

```
# Beitragssumme
tl_mod_beitragssumme <- mod_beitragssumme(PARAM_GLOBAL = PARAM_GLOBAL,
    BEVOELKERUNG = BEVOELKERUNG, BEV_BESTAND = BEV_BESTAND, IK = IK,
    ECKWERTE_EXTENDED = ECKWERTE_EXTENDED, ECKWERTE_SCENARIO = ECKWERTE_SCENARIO,
    AHV_ABRECHNUNG = AHV_ABRECHNUNG, BEITRAGSSAETZE = BEITRAGSSAETZE,
    ARBEITSLOSENQUOTE = ARBEITSLOSENQUOTE, versicherung = "eo",
    ABRECHNUNG = EO_ABRECHNUNG)
```

Danach wird sichergestellt, dass der Datensatz `EO-EINNAHMEN` für jedes Jahr zwischen `PARAM_GLOBAL$jahr_beginn` und `PARAM_GLOBAL$jahr_ende` die EO-Beitragssumme (`eo_beitragsum`) und die Lohnsumme enthält. Fehlende Werte werden dabei durch 0 ersetzt.

```
# Sichertstellen dass der Datensatz vollständig ist
EO_EINNAHMEN <- data.frame(jahr = PARAM_GLOBAL$jahr_beginn:PARAM_GLOBAL$jahr_ende) |>
    left_join(tl_mod_beitragssumme$BEITRAGSSUMME, by = "jahr") |>
    rename(eo_beitragsum = btr_vs_ag) |>
    mutate(across(everything(), ~replace_na(.x, 0))) |>
    left_join(tl_mod_beitragssumme$LOHNSUMME, by = "jahr")
```

Zum Schluss werden die Berechungen an das Finanzperspektivenmodell (bzw. an das Modul `wrap_eo_hauptberechnung`) zurückgegebn:

```
# Output
# ------------------------------------------------------------------------

return(list(EO_EINNAHMEN = EO_EINNAHMEN, STRUKTURFAKTOR =
↪  tl_mod_beitragssumme$STRUKTURFAKTOR))
```

4.7.1 Modul mod_beitragssumme.R

Dokumentation zuletzt aktualisiert am 26.08.2025

Dieses Modul dient zur Berechnung der Beitraggsumme der EO. Umd die Erläuterungen hier zu verstehen sollte vorgängig der Abschnitt zu den Beiträgen von Versicherteten und Arbeitnehmern im Dokument "Modelbeschrieb EO" gelesen werden.

Dieses Modul wird wie folgt in `mod_eo_einnahmen` aufgerufen:

```
# Beitragssumme
tl_mod_beitragssumme <- mod_beitragssumme(PARAM_GLOBAL = PARAM_GLOBAL,
    BEVOELKERUNG = BEVOELKERUNG, BEV_BESTAND = BEV_BESTAND, IK = IK,
    ECKWERTE_EXTENDED = ECKWERTE_EXTENDED, ECKWERTE_SCENARIO = ECKWERTE_SCENARIO,
    AHV_ABRECHNUNG = AHV_ABRECHNUNG, BEITRAGSSAETZE = BEITRAGSSAETZE,
    ARBEITSLOSENQUOTE = ARBEITSLOSENQUOTE, versicherung = "eo",
    ABRECHNUNG = EO_ABRECHNUNG)
```

Das Modul `mod_beitragssume` verwendet neben `PARAM_GLOBAL` die folgende Dataframes:

- `BEVOELKERUNG`: Aus diesem Dataframe werden die Zeitreihen für die Inländer-Erwerbsbevölkerung sowie die Grengänger verwendet, welche zusammen die Erwerbsbevölkerung im Inland ergeben.
- `BEV_BESTAND`: Diese Dataframe wird einzig dazu verwendet, zu bestimmen, wann die letzten verfügbaren Bestandesdaten für die Erwerbsbevölkerung vorliegen.
- `IK`: Die Daten gemäss die individuellen Konten der ZAS.
- `ECKWERTE_EXTENDED`: Daten zur Entwicklung der Preise und des SLI.
- `ECKWERTE_SCENARIO`: Aus diesem Dataframe nutzen wir die Projektionen zur Arbeitslosenquote und dem Wachstum der Erwerbstätigen gemäss BESTA für die Anpassungen in der kurzen Frist. Ausserdem wird das Dataframe dazu verwendet, zu bestimmen, welche Jahre durch das Eckwerte-Szenario der ESTV abgedeckt ist.
- `AHV_ABRECHNUNG`: Da das Modell für die Lohnbeiträge für alle Versicherungen (i.e. AHV, IV und EO) basierend auf die Beiträgen an die AHV geschätzt wird, und die Beiträge für die anderen Versicherungen dann durch Umrechnung des Beitragsatzes berechnet werden, wird hier die AHV Abrechnung benötigt.
- `EO_ABRECHNUNG`: Die EO Abrechnung wird verwendet, um die Abweichung der Projektionen von den Abrechungsdaten zu berechnung, und für diese zu korrigieren.
- `BEITRAGSSAETZE`: Die Beitragssätze für die AHV und EO werden benötigt, um bei der Schätzung des Modells für den Strukturfakot sowie der Projektion für allfällige Beitragssatzänderungen zu korrigieren.
- `ARBEITSLOSENQUOTE`: Die historische Arbeitslosenquote wird für die Anpassungen in der kurzen Frist verwendet (siehe `ECKWERTE_SCENARIO` oben).

Die Parameter `versicherung = eo` und `ABRECHNUNG = EO_ABRECHNUNG` stellen sicherm dass die Beiträge für die Erwerbsersatzordnung geschätzt werden (zur Schätzung der Beiträge für die AHV oder IV müsste die Versicherung entsprechend auf `ahv` oder `iv` gesetzt werden, respektive die `AHV_ABRECHNUNG` oder die `IV_ABRECHNUNG` dem `ABRECHNUNG` Dataframe zugewiesen werden.).

Das Modul `mod_beitragssumme.R` beginnt mit der folgenden Überprüfung:

```
## SAFETY-CHECKS

# Bestimme das maximale Jahr aus BEV_BESTAND, bei dem ept
# nicht NA ist
max_jahr_ept_bestand <- BEV_BESTAND |>
    filter(!is.na(ept)) |>
    summarize(max_jahr = max(jahr)) |>
    pull(max_jahr)

# Bestimme das minimale Jahr aus ECKWERTE_SCENARIO, bei dem
# besta nicht NA ist
min_jahr_besta_projektion <- ECKWERTE_SCENARIO |>
    filter(!is.na(besta)) |>
    summarize(min_jahr = min(jahr)) |>
    pull(min_jahr)

# Überprüfe die Bedingung und stoppe ggf. die Ausführung
# mit einer Fehlermeldung
if (min_jahr_besta_projektion > (max_jahr_ept_bestand + 1)) {
    stop(paste0("Erwerbsbevölkerung ist Stand ", max_jahr_ept_bestand,
        " und BESTA-Projektion startet in ", min_jahr_besta_projektion,
        ". Bitte Erwerbsbevölkerung auf Stand ", min_jahr_besta_projektion -
            1, " bringen, oder Jahr ", min_jahr_besta_projektion -
            1, " zu Eckwerten hinzufügen."))
}
```

Mit dieser Überprüfung wird sichergestellt, dass die beobachtete Erwerbsbevölkerung bis ins Jahr vor der ersten BESTA-Projektionen korrekt für kurzfristige, nicht von den BFS-Szenarien abgedeckte, Anpassungen auf dem Arbeitsmarkt korrigieren. Die Bedingung kann bei rechtzeitiger Datenaktualisierung immer erfüllt werden.

Danach wird noch überprüft, ob die Beschäftigungs- und Arbeitslosenquoteprognose mindestens für die zwie Jahre nach dem aktuellen Abrechnungsjahr vorhanden sind. Falls diese Bedingung nicht erfüllt ist, bricht mit einer Fehlermeldung der Code ab:

```
# Prüfen, ob Projektion für BESTA und ALQ mindestens für
# die 2 auf die Abrechnung folgenden Jahre vorhanden sind
# Filtere Zeilen ohne NA in 'besta' und 'arbeitslos',
# ermittle das maximale Jahr
max_jahr_alq_besta_projektion <- ECKWERTE_SCENARIO |>
    filter(!is.na(besta), !is.na(arbeitslos)) |>
    summarise(max_jahr = max(jahr, na.rm = TRUE)) |>
    pull(max_jahr)

# Überprüfe Bedingung und brich ggf. mit Fehlermeldung ab
if (max_jahr_alq_besta_projektion < PARAM_GLOBAL$jahr_abr + 2) {
    stop("In den ausgewählten Eckwerten ist die Projektion für die BESTA und/oder die
    ↪  Arbeislosenquote nicht für die 2 Jahre ab der aktuellen Abrechnung vorhanden.
    ↪  Dies führt zu fehlern in der Projektion der Lohnbeiträge. Bitte Eckwerte Input
    ↪  überprüfen und aktualisieren.")
}
```

Als nächstes wird überprüft, ob die BESTA-Beschäftigungsprognose Wachstumsraten von absolut mehr als 2% enthält:

```
# Auf grosse Veränderung bei der Beschäftigung überprüfen
max_besta <- ECKWERTE_SCENARIO |>
    summarize(max_besta = max(abs(besta), na.rm = TRUE)) |>
    pull(max_besta)

if (max_besta > 2) {
    message("Warnung: BESTA-Prognose sieht Absolutwert der Beschäftigungsveränderung von
    ↪  mehr als 2% vor. Überprüfen ob BESTA-Korrektur bei Lohnbeiträgen angemessen ist
    ↪  (siehe Dokumentation Lohnbeiträge).")
}
```

Ist dies der Fall wird eine Warnung ausgegeben, da dies auf eine volkswirtschaftlich ausserordentliche Situation hindeutet, bei welcher insbesondere überprüft werden soll, ob der angenommene direkte Übergang in t+2 vom (kurzfristigen) Erwebsbevölkerungswachstum gemäss BESTA-Projektion zum (mittel- bis langfristigen) Erwerbsbenölkerungswachstum gemäss BFS-Szenarien gerechfertigt ist. Es geht um die Frage, ob der Arbeitsmarkt nacht t+2 bereits wieder im Gleichgewicht ist, oder ob ein Rückführungspfad angenommen werden muss.

Es folgt das eigentliche Modell zur Schätzung der Beiträge. Hierbei wird als erstes die Zeitspanne definiert über welche der Struktufaktor geschätzt werden soll:

```
# PROJEKTION LOHNSUMME UND BEITRAGSSUMME

est_struktur_start = max(IK$jahr) - 19
est_struktur_end = max(IK$jahr)
```

Als nächstes wird die für die Schätzung des Modells massgebende Erwerbsbevölerunkg im Inland berechnet:

```
    # INLAND-ERWERBSBEVÖLKERUNG BERECHNEN
    EPT_INLAND <- BEVOELKERUNG %>%
        filter(jahr>=1996, alt>=18) %>%
        mutate(ept_inland=
                   case_when(
            alt<=65 & sex=="m" ~ ept+0.9*frontaliers, # Annahme: Männliche Grenzgänger
↪   arbeiten im Durchschnitt 90%
            alt<=65 & sex=="f" ~ ept+0.7*frontaliers, # Annahme: Weibliche Grenzgänger
↪   arbeiten im Durchschnitt 70%
            alt>65 & sex=="m" ~ 0.5*(ept+0.9*frontaliers), # Annahme: Lohnbeitragsquote
↪   der über 65-jährigen ist nur 50% der Lohnbeitragsquote der unter-65 Jährigen (wegen
↪   Beitragsbefreiung)
            alt>65 & sex=="f" ~ 0.5*(ept+0.7*frontaliers)
            )
        )  %>%
        group_by(jahr, sex) %>%
        summarize(ept_inland=sum(ept_inland, na.rm = TRUE))
```

Hierbei handelt es sich um die Kombination der Inländer-Erwerbsbevölkerung in Vollzeitäquivalenten (Variable `ept` im `BEVOELKERUNG` Dataframe) und der Grenzgänger (Variable `frontaliers` im `BEVOELKERUNG` Dataframe), wobei wir für die Grenzgänger nur über Daten und Szenarien zur Anzahl Personen, jedoch und nicht zur Anzahl Vollzeitäquivalenten, verfügen. Wir nehmen daher für die Konstruktion der Inländer Bevölkerung an, dass die männlichen Grenzgänger einen durchschnittlichen Beschäftigungsgrad von 90% aufweisen, und die weiblichen Granzgänger einen durchschnittlichen Beschäftigungsgrad von 70%[28]. Zusätzlich gilt die Annahme, dass sämtliche Inländer-Erwerbspersonen auch im Inland erwerbstätig sind, also dass

[28]Basierend auf Favre, Föllmi, Zweimüller (2021) Ëinkommensentwicklung von Grenzgängerinnen und Grenzgängern im Aufenthaltsverlauf.

es keine Grenzgänger von der Schweiz ins Ausland gibt (d.h. in der Schweiz leben und im Ausland arbeiten) gibt [29]. Diese Annahme, respektive das Ausmass, zu welchem diese Annahme nicht erfüllt ist achten wir gegeben det mutmasslich sehr geringen Anzahl von Grenzgänger von der Schweiz im Ausland als unbedeutend. Zusätzlich gehen wir davon aus, dass die Lohnbeitragsquote der über 65-Jährigen aufgrund der Beitragsbefreiung lediglich 50% jener der unter 65-Jährigen beträgt.

Anschließend wird für die letzten zwanzig verfügbaren Jahre der Individualkonten (IK) der Strukturfaktor getrennt nach Geschlecht und Jahr berechnet.

Zunächst werden die IK-Daten mit einem Korrekturfaktor angepasst, um die systematische Abweichung gegenüber den Beiträgen laut AHV-Abrechnung auszugleichen (die IK-Beiträge sind systematisch geringer als jene aus der AHV-Abrechnung).

Anschließend wird der Wert eines Beitragssatz-Prozents berechnet, indem die Beiträge durch 8.4% geteilt werden\footnote{Es wird mit 8.4% und nicht mit 8.7% gerechnet, da der tiefere Beitragssatz für Selbständigerwerbende berücksichtigt wird.}.

Daraufhin wird für jedes Jahr und Geschlecht der Strukturfaktor ermittelt. Dieser misst den Anteil der Veränderung des effektiven Beitragssatzes, der nicht durch die Entwicklung der Erwerbstätigkeit (`delta_ept`) oder durch die Lohnindexierung (`delta_sli`) erklärt werden kann:

$$Strukturfaktor_{t,s} = \frac{Beitragsprozent_{t,s}}{Beitragsprozent_{t-1,s} * (1 + \Delta ept_{t,s}) * (1 + \Delta sli_t)}$$

Das Ergebnis ist eine Zeitreihe geschätzter Strukturfaktoren nach Geschlecht für die letzten zwanzig verfügbaren IK-Jahre.

```
# STRUKTURFAKTOR BASIEREND AUF AHV-DATEN BERECHNEN
STRUKTURFAKTOR_SERIES <- IK |>
    filter(
        vz=="ahv",
        jahr>=est_struktur_start
        ) |>
    group_by(jahr, sex) |>
    summarize(
        ahv_beitraege=sum(beitrag_bsv, na.rm = TRUE),
        .groups = "drop"
        ) |>
    left_join(EPT_INLAND %>%
                  group_by(jahr, sex) %>%
                  summarize(ept_inland=sum(ept_inland, na.rm = TRUE))
              ) %>%
    left_join(ECKWERTE_EXTENDED) %>%
    left_join(AHV_ABRECHNUNG |> select(jahr, btr_vs_ag)) |>
    group_by(jahr) |> # Gruppieren nach Jahr
    mutate( # Korrektur IK-Daten um die Abweichung von den Beiträgen gemäss
    ↪  Abrechnung
        sum_ahv_beitraege = sum(ahv_beitraege, na.rm = TRUE), # Summe der Beiträge
        ↪  berechnen
        fraction = sum_ahv_beitraege / btr_vs_ag,    # Bruch berechnen
        ahv_beitraege = ahv_beitraege / fraction   # Beiträge durch den Bruch teilen
    ) |>
    ungroup() |> # Gruppierung entfernen
    left_join(BEITRAGSSAETZE %>% select(jahr, ahv_vs_ag)) |>
```

[29] Da das Modell lediglich auf Wachstumsraten der Erwerbsbevölkerung aufbaut ist die schwächere Annahme ausreichend, dass der Anteil an Grenzgänger von der Schweiz ins Ausland an der durch uns berechneten Inland-Erwerbsbevölkerung über die Zeit konstant ist.

```
        mutate(
            beitragsprozent=ahv_beitraege/(ahv_vs_ag*100)
        ) |>
        select(-sum_ahv_beitraege, -ahv_vs_ag, -btr_vs_ag, -fraction) |>
        group_by(sex) |> # Gruppieren nach sex
        mutate(
            delta_ept = (ept_inland / lag(ept_inland) - 1),
            delta_sli = ifelse(jahr==est_struktur_start,NA,lohn/100)
        ) |>
        mutate(

            ↪  strukturfaktor=beitragsprozent/(lag(beitragsprozent)*(1+delta_ept)*(1+delta_sli))-1
        ) |>
        group_by(sex)
```

Wenn der Parameter `szenario_fhh` auf `basis` gesetzt ist, berechnet der folgende Codeblock aus der Datenreihe der Strukturfaktoren (`STRUKTURFAKTOR_SERIES`) den geometrischen Mittelwert pro Geschlecht. Die Berechnung erfolgt nach folgender Formel:

$$\overline{strukturfaktor_s} = \left(\prod(1 + strukturfaktor_s)\right)^{\frac{1}{n}} - 1$$

Dabei steht n für die Anzahl gültiger (nicht fehlender) Werte. Anschließend wird das Ergebnis mit 100 multipliziert, um den durchschnittlichen Strukturfaktor in Prozent auszudrücken.

```
if (PARAM_GLOBAL$szenario_fhh == "basis") {

    # Berechne strukturfaktor direkt über Mittelwert
    STRUKTURFAKTOR <- STRUKTURFAKTOR_SERIES |>
        summarize(strukturfaktor = (prod(1 + strukturfaktor,
            na.rm = TRUE))^(1/sum(!is.na(strukturfaktor))) -
            1) |>
        mutate(strukturfaktor = strukturfaktor * 100)

}
```

Wenn der Parameter `szenario_fhh` nicht auf `basis` gesetzt ist [30], wird der Strukturfaktor nach Geschlecht mithilfe eines Bootstrapping-Verfahrens (10'000 Wiederholungen) als geometrisches Mittel geschätzt. Abhängig vom gewählten Szenario wird ein quatsilbasierter Wert verwendet: das 80. Perzentil für das Szenario `hoch`, das 20. Prenzentil für `tief`. Dadurch wird die Unsicherheit der Schätzung berücksichtigt und eine szenarienspezifische Modellierung ermöglicht.

```
 else {

        # Führe gruppiertes Bootstrapping durch
        set.seed(123)  # Setze Seed für Reproduzierbarkeit
        STRUKTURFAKTOR <- STRUKTURFAKTOR_SERIES |>
            group_split() |>
            purrr::map_dfr(function(df) {
                boot_fn <- function(data, indices) {
                    data_boot <- data[indices, ]
                    sf <- prod(1 + data_boot$strukturfaktor, na.rm = TRUE)^(1 /
↪   sum(!is.na(data_boot$strukturfaktor))) - 1
                    return(sf * 100)
```

[30] Das Modul wird für alle Versicherungen benutzt. Dieses Teil ist momentan für die EO nicht relevant.

```
            }

            boot_out <- boot::boot(
                data = df,
                statistic = boot_fn,
                R = 10000
            )

            sf_value <- dplyr::case_when(
                PARAM_GLOBAL$szenario_fhh=="hoch" ~ quantile(boot_out$t, 0.8),
                PARAM_GLOBAL$szenario_fhh=="tief" ~ quantile(boot_out$t, 0.2),
                TRUE ~ NA_real_
            )

            tibble::tibble(
                sex = unique(df$sex),
                strukturfaktor = sf_value
            )
        })
}
```

Nach dem den Struktufakotr geschätzt wurde, sind alle Elemente vorhanden um die Entwicklung der geschlechtsspezifische Beitragssummen (`btr_vs_ag`) zu berechnen. Dafür werden zuerst die IK Daten nach Verzicherung (in dem Fall `versicherung=eo`) und Abrechnungzeitraum gefiltert. Wir berechnen pro Jahr und Geschlecht der Summe der Beiträge:

```
BEITRAGSSUMME_SEX  <- IK |>
       filter(
           vz==versicherung,
           jahr>=min(ABRECHNUNG$jahr)
       ) |>
       group_by(jahr, sex) |>
       summarize(
           beitraege=sum(beitrag_bsv, na.rm = TRUE),
           .groups = "drop"
       ) |>
```

Zusätzliche Informationen (EPT, SLI, Abrechung, Arbeitslosenquote und BESTA) werden hinzugefügt und die Beiträge aus den IK werden skaliert, um mit den Abrechnungswerten übereinzustimmen:

```
full_join(EPT_INLAND %>%
                     filter(
                         jahr>=min(IK$jahr),
                         jahr>=min(ABRECHNUNG$jahr)
                         )
       ) %>%
       left_join(ECKWERTE_EXTENDED) %>%
       left_join(ABRECHNUNG |> select(jahr, btr_vs_ag)) |>
       group_by(jahr) |> # Gruppieren nach Jahr
       mutate( # Korrektur IK-Daten um die Abweichung von den Beiträgen gemäss
       ↪  Abrechnung
           sum_beitraege = sum(beitraege, na.rm = TRUE), # Summe der Beiträge berechnen
           fraction = sum_beitraege / btr_vs_ag,    # Bruch berechnen
           beitraege = beitraege / fraction,
           ept_inland_total=sum(ept_inland)
```

```
        ) |>
        ungroup() |>
        left_join(ARBEITSLOSENQUOTE) %>%
        left_join(ECKWERTE_SCENARIO %>%  select(jahr, besta, arbeitslos)) %>%
```

Anschließend werden die geschlechtspzifische Entwicklungen der Erwerbsbevölkerung und des SLI im Vergleich zum Vorjahr berechnet. Für die kurzfristige Projektion (`jahr <= PARAM_GLOBAL$jahr_abr + 3`) verwenden wir für die kommenden zwei Jahre die Beschäftigungsprognose des SECO anstelle der Erwerbsbevölkerungsszenarien des BFS.

Konkret wird angenommen, dass sich die Erwerbsbevölkerung entsprechend der Wachstumsrate der Beschäftigung zuzüglich der Veränderung der Arbeitslosenquote entwickelt. Der Einbezug der Arbeitslosenquote ist dadurch motiviert, dass auf Arbeitslosentaggelder Sozialversicherungsbeiträge erhoben werden und daher auch Taggeldbezüger zur beitragszahlenden Population zählen.

```
group_by(sex) %>%
        mutate(
            arbeitslosenquote=ifelse(is.na(arbeitslosenquote), arbeitslos,
            ↪  round(arbeitslosenquote,1)),
            delta_arbeitslosenquote=arbeitslosenquote-lag(arbeitslosenquote),
            besta_alq_factor = ifelse(jahr>max_jahr_ept_bestand &
            ↪  jahr<=PARAM_GLOBAL$jahr_abr+3 & !is.na(besta) & !is.na(arbeitslos),
            ↪  lag(ept_inland_total) / ept_inland_total * (1 + besta/100 +
            ↪  delta_arbeitslosenquote/100) - 1, 0),
            delta_ept = (ept_inland / lag(ept_inland) - 1),
            delta_sli = lohn/100
        ) |>
```

Der geschätzte Strukturfaktor wird auch noch hinzugefügt und die Beiträge werden anhand den verschiedenen Wachstumsfaktoren (EPT, SLI, Arbeitslosenquote und Strukturfaktor) projiziert:

```
        left_join(STRUKTURFAKTOR) |>
        mutate(
          beitraege =
                ifelse(jahr>= max(IK$jahr)+1,
                       beitraege[jahr == max(IK$jahr)] *
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,delta_ept, 0))) * #
                           ↪  Ersetze NA-Werte durch 0
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,delta_sli, 0))) *
                           ↪  # Ersetze NA-Werte durch 0
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,besta_alq_factor,
                           ↪  0))) *
                           (1+strukturfaktor/100)^(jahr-max(IK$jahr)),
                       beitraege)
            ) %>%
```

In einem weiteren Schritt werden die geschätzten Beitragssummen je Jahr aggregiert, um die Gesamtsumme der Beiträge pro Jahr zu berechnen. Diese dient als Grundlage für die anschließende Verteilung der Beiträge auf die einzelnen Geschlechter. Hierzu wird die geschätzte Beitragssumme jedes Geschlechts anteilig so angepasst, dass ihre Summe dem vorgegebenen Gesamtwert aus der Abrechnung entspricht. Für Jahre ab dem Abrechnungsjahr (`PARAM_GLOBAL$jahr_abr`) wird dazu der in der Abrechnung gemeldete Gesamtwert verwendet und proportional auf die geschätzten Beiträge aufgeteilt. Für frühere Jahre erfolgt die Verteilung entsprechend der jeweiligen jährlichen Beitragssumme.

Abschließend erfolgt eine Selektion der relevanten Variablen, sodass das Ergebnis eine Tabelle mit den final berechneten Beitragssummen (`btr_vs_ag`) nach Jahr und Geschlecht darstellt.

```
ungroup() |>
    group_by(jahr) |>
    mutate(sum_beitraege = sum(beitraege, na.rm = TRUE)) %>%
    group_by(sex) |>
    mutate(btr_vs_ag = ifelse(jahr >= PARAM_GLOBAL$jahr_abr,
        beitraege * btr_vs_ag[jahr == PARAM_GLOBAL$jahr_abr]/sum_beitraege[jahr ==
            PARAM_GLOBAL$jahr_abr], beitraege * btr_vs_ag/sum_beitraege)) |>
    select(jahr, sex, btr_vs_ag)
```

Dann wird noch der Effekt der Ausgleichmassnahmen der AHV21 auf die Erwerbsbeteiligung von Frauen aus den Übergangsgenerationen modelliert. Dazu wird zunächst ein Vektor mit Korrekturwerten (pro Jahr, von 2025 bis 2033) definiert. Diese Werte wurden bei der Schätzung des Effekts der Ausgleichsmassnahmen auf die Lohnbeiträge bei der Debatte der AHV21 berechnet und geben an, wie sich die Massnahmen in Millionen CHF auf die Lohnbeiträge auswirken.

```
### Effekt der Ausgleichsmassnahmen der AHV21 auf die
### Erwerbsbeteiligung der Frauen am Referenzalter
### berücksichtigen

# Der hier verwendete Vektor wurde bei der Schätzung des
# Effekts der Ausgleichsmassnahmen auf die Lohnbeiträge bei
# der Debatte der AHV21 berechnet. Aufgrund der geringen
# Auswirkungen wird er hier der Einfachheit halber direkt
# so verwendet.  Definiere Effekt-Vektor: Beschreibung: Der
# Effekt der Ausgleichsmassnahmen der AHV21 auf die
# Erwerbsbeteiligung von Frauen aus den
# Übergangsgenerationen. Effekt pro Lohnprozent.

ausgl_uebergangsgenerationen_ahv21 <- tibble(jahr = 2025:2033,
    delta = c(-7.011494253, -8.850574713, -10.8045977, -12.75862069,
        -13.10344828, -13.33333333, -13.44827586, -13.56321839,
        -13.67816092) * 1e+06  # Multipliziere mit 1000000, um von Mio. zu CHF zu
        ↪  gelangen
)
```

Da ab dem Abrechnungsjahr 2025 der Effekt bereits in der Basisrechnung enthalten ist, wird für die Folgejahre nur noch der zusätzliche marginale Effekt berücksichtigt. Konkret bedeutet das: fü das aktuelle Abrechnungsjahr wird der Effekt auf null gesetzt, und für spätere Jahre wird der jeweilige Korrekturwert um den bereits berücksichtigen Effekt reduziert:

```
# Berücksichtigen, dass ab Abrechnungsjahr 2025 der Effekt
# der Ausgleichsmassnahmen bereits in der Abrechnung
# enthalten ist, wodurch lediglich der marginale
# zusätzliche Effekt in den Folgejahre berücksichtigt
# werden soll:
if (PARAM_GLOBAL$jahr_abr >= 2025) {
    abr_wert <- ausgl_uebergangsgenerationen_ahv21 |>
        filter(jahr == PARAM_GLOBAL$jahr_abr) |>
        pull(delta)

    ausgl_uebergangsgenerationen_ahv21 <- ausgl_uebergangsgenerationen_ahv21 |>
        mutate(delta = case_when(jahr <= PARAM_GLOBAL$jahr_abr ~
            0, jahr > PARAM_GLOBAL$jahr_abr ~ delta - abr_wert,
            TRUE ~ delta))
}
```

Anschliessend werden die Korrekturwerte mit den entsprechenden Beitragssätzen verknüpft und auf die Betraigssumme der Frauen angewendet. Damit wird sichergestellt, dass die finanziellen Auswirkungen der Ausgleichmassnahmen im Modell korrekt in die Berechnung der Beiträge einfliessen. Das Ergebnis ist das Dataframe `BEITRAGSSUME_SEX` mit den final berechneten Beitragssummen (`btr_vs_ag`) nach Jahr und Geschlecht darstellt.

```
BEITRAGSSUMME_SEX <- BEITRAGSSUMME_SEX |>
    left_join(ausgl_uebergangsgenerationen_ahv21 |>
        left_join(BEITRAGSSAETZE |>
            select(jahr, beitragssatz = all_of(paste0(versicherung,
                "_vs_ag"))), by = "jahr") |>
        mutate(delta = delta * beitragssatz * 100), by = "jahr") |>
    mutate(btr_vs_ag = if_else(sex == "f" & !is.na(delta), btr_vs_ag +
        delta, btr_vs_ag)) |>
    select(jahr, sex, btr_vs_ag) |>
    arrange(jahr, sex)
```

Die Beitragssume ohne geschlechtspezifische Differenzierung wird ebenfalls in einem Dataframe (`BEITRAGSSUMME`) gespeichert:

```
BEITRAGSSUMME <- BEITRAGSSUMME_SEX %>%
    group_by(jahr) %>%
    summarize(btr_vs_ag = sum(btr_vs_ag))
```

Analog zur Beitragssumme wird das Verfahren im Anschluss auf die Projektion der Lohnsumme übertragen:

```
# PROJEKTION AHV-LOHNSUMME
    LOHNSUMME_SEX <- IK |>
        filter(
            vz=="ahv"
        ) |>
        group_by(jahr, sex) |>
        summarize(
            ahv_beitraege=sum(beitrag_bsv, na.rm = TRUE),
            ahv_lohnsumme=sum(einkommen_ik, na.rm = TRUE),
            .groups = "drop"
        ) |>
        full_join(EPT_INLAND %>%
                      filter(jahr>=min(IK$jahr)),
                  by = c("jahr", "sex")
        ) %>%
        left_join(ECKWERTE_EXTENDED, by = "jahr") %>%
        left_join(AHV_ABRECHNUNG |> select(jahr, btr_vs_ag), by = "jahr") |>
        group_by(jahr) |> # Gruppieren nach Jahr
        mutate( # Korrektur IK-Daten um die Abweichung von den Beiträgen gemäss
        ↪  Abrechnung
            sum_ahv_beitraege = sum(ahv_beitraege, na.rm = TRUE), # Summe der Beiträge
            ↪  berechnen
            fraction = sum_ahv_beitraege / btr_vs_ag,    # Bruch berechnen
            ahv_beitraege = ahv_beitraege / fraction,
            ahv_lohnsumme = ahv_lohnsumme / fraction   # Beiträge durch den Bruch teilen
        ) |>
        ungroup() |>
        left_join(ARBEITSLOSENQUOTE, by = "jahr") %>%
        left_join(ECKWERTE_SCENARIO %>%  select(jahr, besta, arbeitslos), by = "jahr")
        ↪  %>%
```

```
        group_by(sex) %>%
        mutate(
            arbeitslosenquote=ifelse(is.na(arbeitslosenquote), arbeitslos,
            ↪  round(arbeitslosenquote,1)),
            delta_arbeitslosenquote=arbeitslosenquote-lag(arbeitslosenquote),
            besta_alq_factor = ifelse(jahr<=max_jahr_ept_bestand | is.na(besta) |
            ↪  is.na(arbeitslos) | jahr>PARAM_GLOBAL$jahr_abr+3, 0, lag(ept_inland) /
            ↪  ept_inland * (1 + besta/100 + delta_arbeitslosenquote/100) - 1),
            delta_ept = (ept_inland / lag(ept_inland) - 1),
            delta_sli = lohn/100
        ) |>
        left_join(STRUKTURFAKTOR, by = "sex") |>
        mutate(
          ahv_beitraege =
                ifelse(jahr>=max(IK$jahr)+1,
                       ahv_beitraege[jahr == max(IK$jahr)] *
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,delta_ept, 0))) * #
                           ↪  Ersetze NA-Werte durch 0
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,delta_sli, 0))) *
                           ↪  # Ersetze NA-Werte durch 0
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,besta_alq_factor,
                           ↪  0))) *
                           (1+strukturfaktor/100)^(jahr-max(IK$jahr)),
                       ahv_beitraege),
          ahv_lohnsumme =
                ifelse(jahr>=max(IK$jahr)+1,
                       ahv_lohnsumme[jahr == max(IK$jahr)] *
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,delta_ept, 0))) * #
                           ↪  Ersetze NA-Werte durch 0
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,delta_sli, 0))) *
                           ↪  # Ersetze NA-Werte durch 0
                           cumprod((1 + ifelse(jahr>=max(IK$jahr)+1,besta_alq_factor,
                           ↪  0))) *
                           (1+strukturfaktor/100)^(jahr-max(IK$jahr)),
                       ahv_lohnsumme)
        ) %>%
        ungroup() |>
        group_by(jahr) |>
        mutate(sum_ahv_beitraege = sum(ahv_beitraege, na.rm = TRUE)) %>%
        group_by(sex) |>
        mutate(
          ahv_lohnsumme = ifelse(jahr>=PARAM_GLOBAL$jahr_abr,
                                 ahv_lohnsumme * btr_vs_ag[jahr == PARAM_GLOBAL$jahr_abr]
↪  / sum_ahv_beitraege[jahr == PARAM_GLOBAL$jahr_abr],
                                 ahv_lohnsumme * btr_vs_ag / sum_ahv_beitraege
            )
        ) |>
        select(jahr, sex, ahv_lohnsumme)

    ### Effekt der Ausgleichsmassnahmen der AHV21 auf die Erwerbsbeteiligung der Frauen
    ↪  am Referenzalter berücksichtigen. Siehe Codeblock weiter oben (vor Berechnung des
    ↪  Dataframe BEITRAGSSUMME) für Erläuterungen.

    # Addiere Effekt nur bei Frauen
```

```
    LOHNSUMME_SEX <- LOHNSUMME_SEX |>
      left_join(ausgl_uebergangsgenerationen_ahv21, by = "jahr") |>
      mutate(ahv_lohnsumme = if_else(sex == "f" & !is.na(delta), ahv_lohnsumme +
      ↪  delta*100, ahv_lohnsumme)) |> # delta hier mit 100 skalieren, da Effekt pro
      ↪  Lohnprozent in ausgl_uebergangsgenerationen_ahv21 enthalten ist.
      select(jahr, sex, ahv_lohnsumme) |>
      arrange(jahr, sex)

    LOHNSUMME <- LOHNSUMME_SEX %>%
        group_by(jahr) %>%
        summarize(ahv_lohnsumme=sum(ahv_lohnsumme))
```

Schlussendlich werden noch die Ergebnisse an das Modul `mod_eo_einnahmen` zurückgegeben:

```
# --- Output
# ------------------------------------------------------------------

return(list(BEITRAGSSUMME = BEITRAGSSUMME, LOHNSUMME = LOHNSUMME,
    BEITRAGSSUMME_SEX = BEITRAGSSUMME_SEX, LOHNSUMME_SEX = LOHNSUMME_SEX,
    STRUKTURFAKTOR = STRUKTURFAKTOR))
```

4.8 Modul mod_eo_ausgaben.R

Dokumentation zuletzt aktualisiert am 02.07.2025

Das Hauptmodul zur Berechnung der Ausgaben der EO braucht die folgende Argumente, welche in Kapitel 2 im Detail beschrieben wurden:

```
mod_eo_ausgaben <- function(PARAM_GLOBAL,
                            REGISTER_MSU,
                            REGISTER_VSU,
                            REGISTER_DIENST,
                            REGISTER_BEU,
                            REGISTER_ADOPT,
                            LEBENDGEBURTEN_FRAU,
                            LEBENDGEBURTEN_MANN,
                            BEV_SCENARIO,
                            FRONTALIERS,
                            BEVOELKERUNG,
                            ECKWERTE_EXTENDED,
                            EOMAX,
                            EO_ABRECHNUNG_DEF,
                            EO_AUSGABEN_ANTEILE,
                            DISKONTFAKTOR,
                            STRUKTURFAKTOR) {
```

- `PARAM_GLOBAL`: Parameter für die Berechnung, siehe Kapitel 4.3. Wir benutzen hier:
  - `jahr_abr`: letztes Jahr, für das Register- und Buchhaltungsdaten verfügbar sind.
  - `jahr_ende`: letztes Jahr, für das Prognosen berechnet werden.
- `REGISTER_MSU`: Daten aus dem EO-Register ab 2013 zum Mutterschaftsurlaub, gruppiert nach Jahr, Domizil und Altersgruppe der Mutter.
- `REGISTER_VSU`: Daten aus dem EO-Register ab 2021 zum Urlaub des anderen Elternteils, gruppiert nach Jahr, Domizil und Altersgruppe des anderen Elternteils.

- `REGISTER_DIENST`: Daten aus dem EO-Register ab 2008 zu Dienstleistenden, gruppiert nach Jahr, Leistungsart und Altersgruppe.
- `REGISTER_BEU`: Daten aus dem EO-Register ab 2021 zum Betreuungsurlaub.
- `REGISTER_ADOPT`: Daten aus dem EO-Register ab 2022 zum Adoptionsurlaub.
- `LEBENDGEBURTEN_FRAU`: Daten mit der historischen Anzahl Geburten pro Jahr und Altersgruppe der Mutter.
- `LEBENDGEBURTEN_MANN`: Daten mit der historischen Anzahl Geburten pro Jahr und Altersgruppe des Vaters.
- `LEBENDGEBURTEN_SZENARIO`: Daten zur zukünftigen Anzahl Geburten pro Jahr, aus dem demographischen Szenario des BFS.
- `BEV_SCENARIO`: Daten zur zukünftigen Anzahl Geburten gemäss BFS Szenario.
- `FRONTALIERS`: Daten zur historischen und zukünftigen Anzahl Grenzgänger nach Jahr und Altersgruppe.
- `BEVOELKERUNG`: Daten zur historisch und zukünftigen Bevölkerungsentwicklung des BFS.
- `ECKWERTE_EXTENDED`: historisches und prognostiziertes jährliches Wachstum des Lohnindex, Strukturfaktor und Preisindex.
- `EOMAX`: historische und zukünftige Entwicklung der maximalen und minimalen Taggelder.
- `EO_ABRECHNUNG_DEF`: Buchhaltungsdaten der EO.
- `EO_AUSGABEN_ANTEILE`: Anteil an den Gesamtausgaben nach Leistungsart, basierend auf dem EO-Register.
- `DISKONTFAKTOR`: Datensatz mit den Diskontierungsfaktoren im Vergleich zum Kalenderjahr von `jahr_preisbasis` in `PARAM_GLOBAL`.
- `STRUKTURAFKTOR`: Strukturfaktor, der die Lohnentwicklung abbildet, soweit sie nicht durch den SLI erklärt wird. Zur Schätzmethode siehe **??**.

Dieses Modul greift auf Untermodule zurück, die die Ausgaben nach Leistungsart (Dienst, Mutterschaftsurlaub, Urlaub des anderen Elternteils, Betreuungs- und Adoptionsurlaub) berechnen.

```
# Dienstleistende
# ------------------------------------------------------------

tl_mod_eo_dienst <- mod_eo_dienst(PARAM_GLOBAL, BEVOELKERUNG,
    REGISTER_DIENST, EOMAX, ECKWERTE_EXTENDED, EO_ABRECHNUNG_DEF,
    EO_AUSGABEN_ANTEILE)

DIENST_AUSGABEN <- tl_mod_eo_dienst$DIENST_AUSGABEN
DIENST_BENEF <- tl_mod_eo_dienst$DIENST_BENEF
DIENST_ANTEILE <- tl_mod_eo_dienst$DIENST_ANTEILE
DIENST_DAUER <- tl_mod_eo_dienst$DIENST_DAUER
DIENST_TAGGELD <- tl_mod_eo_dienst$DIENST_TAGGELD


# Mutterschaft
# ------------------------------------------------------------------

tl_mod_eo_msu <- mod_eo_msu(PARAM_GLOBAL, REGISTER_MSU, LEBENDGEBURTEN_FRAU,
    BEV_SCENARIO, FRONTALIERS, EOMAX, ECKWERTE_EXTENDED, EO_ABRECHNUNG_DEF,
    EO_AUSGABEN_ANTEILE, BEVOELKERUNG)
```

```
MSU_AUSGABEN <- tl_mod_eo_msu$MSU_AUSGABEN
MSU_BENEF <- tl_mod_eo_msu$MSU_BENEF
MSU_TAGGELD <- tl_mod_eo_msu$MSU_TAGGELD

# Anderer Elternteil
# ------------------------------------------------------------------

tl_mod_eo_vsu <- mod_eo_vsu(PARAM_GLOBAL, REGISTER_VSU, LEBENDGEBURTEN_MANN,
    BEV_SCENARIO, FRONTALIERS, EOMAX, ECKWERTE_EXTENDED, EO_ABRECHNUNG_DEF,
    EO_AUSGABEN_ANTEILE, STRUKTURFAKTOR)

VSU_AUSGABEN <- tl_mod_eo_vsu$VSU_AUSGABEN
VSU_BENEF <- tl_mod_eo_vsu$VSU_BENEF
VSU_TAGGELD <- tl_mod_eo_vsu$VSU_TAGGELD

# Betreuung
# ------------------------------------------------------------------

tl_mod_eo_betreuung <- mod_eo_betreuung(PARAM_GLOBAL, REGISTER_BEU,
    MSU_AUSGABEN, VSU_AUSGABEN, EO_ABRECHNUNG_DEF, EO_AUSGABEN_ANTEILE)

BSU_AUSGABEN <- tl_mod_eo_betreuung$BSU_AUSGABEN

# Adoption
# ------------------------------------------------------------------

tl_mod_eo_adoption <- mod_eo_adoption(PARAM_GLOBAL, REGISTER_ADOPT,
    MSU_AUSGABEN, VSU_AUSGABEN, EO_ABRECHNUNG_DEF, EO_AUSGABEN_ANTEILE)

ADOPTION_AUSGABEN <- tl_mod_eo_adoption$ADOPTION_AUSGABEN
```

Jedes dieser Untermodule wird im weiteren Verlauf dieser Dokumentation detailliert vorgestellt.

Die Verwaltungskosten werden anschliessend im folgenden Submodul berechnet:

```
# Verwaltungskosten
# ------------------------------------------------------------
tl_mod_eo_verwaltungskosten <- mod_eo_verwaltungskosten(PARAM_GLOBAL,
    EO_ABRECHNUNG_DEF, DISKONTFAKTOR)

EO_VERWALTUNGSKOSTEN <- tl_mod_eo_verwaltungskosten$EO_VERWALTUNGSKOSTEN
```

Die Ausgaben, die aus den oben genannten Untermodulen resultieren, werden in einem einzigen Datensatz formatiert und für die spätere Darstellung der Ergebnisse zusammengefügt (Erstellung der Tabelle der Finanzperspektiven).

```
MSU_AUSGABEN_1 <- MSU_AUSGABEN |>
    mutate(type = "msu") |>
    select(jahr, eo_total_just = aus_tot_mut, type)

DIENST_AUSGABEN_1 <- DIENST_AUSGABEN |>
    mutate(type = "dienst") |>
    select(jahr, eo_total_just = ausgaben_dienst, type)

VSU_AUSGABEN_1 <- VSU_AUSGABEN |>
```

```
    mutate(type = "vsu") |>
    select(jahr, eo_total_just = aus_tot_vsu, type)

BSU_AUSGABEN_1 <- BSU_AUSGABEN |>
    mutate(type = "bsu") |>
    select(jahr, eo_total_just = ausgaben_bsu, type)

ADOPTION_AUSGABEN_1 <- ADOPTION_AUSGABEN |>
    mutate(type = "adoption") |>
    select(jahr, eo_total_just = ausgaben_adopt, type)

EO_AUSGABEN_VERWALTUNGSKOSTEN_1 <- EO_VERWALTUNGSKOSTEN |>
    mutate(type = "verwaltungskosten") |>
    select(jahr, eo_total_just = verw_kost, type)

EO_AUSGABEN <- rbind(MSU_AUSGABEN_1, DIENST_AUSGABEN_1, VSU_AUSGABEN_1,
    BSU_AUSGABEN_1, ADOPTION_AUSGABEN_1, EO_AUSGABEN_VERWALTUNGSKOSTEN_1)
```

Schliesslich werden die Gesamtausgaben als Summe aller Ausgabentypen berechnet und im Datensatz `EO_AUSGABEN` mit den Ausgaben pro Urlaubsart gespeichert. Die Spalten werden am Ende umbenannt.

```
EO_AUSGABEN <- EO_AUSGABEN |>
    group_by(jahr) |>
    summarise(aus_total_ink_vk = sum(eo_total_just, na.rm = TRUE)) |>
    ungroup() |>
    left_join(EO_AUSGABEN, by = "jahr") |>
    pivot_wider(names_from = type, values_from = eo_total_just) |>
    select(c("jahr", "aus_total_ink_vk", "adoption", "bsu", "msu",
        "dienst", "vsu", "verwaltungskosten")) |>
    mutate(across(everything(), ~replace_na(.x, 0)))

lookup_table_colnames <- c(eo_dienst_just = "dienst", eo_msu_just = "msu",
    eo_vsu_just = "vsu", eo_bsu_just = "bsu", aus_total_ink_vk = "aus_total_ink_vk",
    jahr = "jahr", eo_adoption_just = "adoption", eo_verw_kost_just =
    ↪  "verwaltungskosten")

names(EO_AUSGABEN) <- names(lookup_table_colnames[match(names(EO_AUSGABEN),
    lookup_table_colnames)])
```

Am Ende werden noch die Dataframes gelistet, die an das Modul `mod_eo_hauptberechnung` zurückgegeben werden:

```
 # Output ------------------------------------------------------------------

  list(
    EO_AUSGABEN = EO_AUSGABEN,
    DIENST_AUSGABEN = DIENST_AUSGABEN,
    DIENST_BENEF = DIENST_BENEF,
    DIENST_ANTEILE = DIENST_ANTEILE,
    DIENST_DAUER = DIENST_DAUER,
    DIENST_TAGGELD = DIENST_TAGGELD,
    MSU_AUSGABEN = MSU_AUSGABEN,
    MSU_BENEF = MSU_BENEF,
    MSU_TAGGELD = MSU_TAGGELD,
    VSU_AUSGABEN = VSU_AUSGABEN,
```

```
    VSU_BENEF = VSU_BENEF,
    VSU_TAGGELD = VSU_TAGGELD,
    BSU_AUSGABEN = BSU_AUSGABEN,
    ADOPTION_AUSGABEN = ADOPTION_AUSGABEN)
}
```

4.8.1 Modul mod_eo_dienst.R

Dokumentation zuletzt aktualisiert am 26.08.2025

Dieser Modul berechnet die Ausgaben der EO für Dienstleistende.

Die Argumente der Funktion mod_eo_dienst sind folgende und sind weiter oben im Kapitel beschrieben:

```
mod_eo_dienst <- function(PARAM_GLOBAL,
                          BEVOELKERUNG,
                          REGISTER_DIENST,
                          EOMAX,
                          ECKWERTE_EXTENDED,
                          EO_ABRECHNUNG_DEF,
                          EO_AUSGABEN_ANTEILE) {
```

Dieses Modul ist ebenfalls in mehrere Untermodule unterteilt. Als erstes werden die Anzahl Dienstleistende im Modul `mod_eo_dienst_benef` prognostiziert.

```
# Anzahl Dienstleistende
tl_mod_eo_dienst_benef <- mod_eo_dienst_benef(PARAM_GLOBAL, BEVOELKERUNG,
    REGISTER_DIENST)

DIENST_BENEF <- tl_mod_eo_dienst_benef$DIENST_BENEF
```

Dann werden die durchschnittlichen Taggelder prognostiziert, die Dienstleistende bekommen, wenn ihre Entschädigung zwischen `eomax` und `eomin` liegt.

```
# Taggeld Entwicklung
tl_mod_eo_dienst_taggeld <- mod_eo_dienst_taggeld(PARAM_GLOBAL,
    REGISTER_DIENST, ECKWERTE_EXTENDED, EOMAX)

DIENST_TAGGELD <- tl_mod_eo_dienst_taggeld$DIENST_TAGGELD
```

Die Entwicklung der Verteilung der Dienstleistenden nach Höhe des Taggelder (`eomax`, `eomin` und `zwminmax`) wird ebenfalls in einem Untermodul modelliert.

```
# Anteile EOMIN, EOMAX, und indzwischen
tl_mod_eo_dienst_anteile <- mod_eo_dienst_anteile(PARAM_GLOBAL,
    REGISTER_DIENST, ECKWERTE_EXTENDED, EOMAX, TAGGELD_DIENST)

DIENST_ANTEILE <- tl_mod_eo_dienst_anteile$DIENST_ANTEILE
```

Die durchschnittliche Dauer jeder Dienstart und Taggeldkategorie wird im Submodul `mod_eo_dienst_dauer` berechnet. Dieses wird auch in `mod_eo_dienst` aufgerufen.

```
# Dauer (Anzhahl Tage)
tl_mod_eo_dienst_dauer <- mod_eo_dienst_dauer(PARAM_GLOBAL, REGISTER_DIENST)

DIENST_DAUER <- tl_mod_eo_dienst_dauer$DIENST_DAUER
```

Die Aufwendungen für Kinderzulagen, Betriebs- und Betreuungszulagen für Dienstleistende werden im Submodul `mod_eo_dienst_zulage` separat modelliert.

```
# Zulage
tl_zulage <- mod_eo_dienst_zulage(PARAM_GLOBAL, EOMAX, REGISTER_DIENST)

KZ <- tl_zulage$KZ
BETRIEBZ <- tl_zulage$BETRIEBZ
BETREUUNGZ <- tl_zulage$BETREUUNGZ
```

Wie im Dokument „Modellbeschrieb der Ausgabenseite der Finanzperspektiven der EO“ erklärt, werden die Ausgaben für die Leistungsart J+S-Leiterkurse (`kat_leist=300`) und Jungschützenleiterkurse (`kat_leist=500`) als Anteile der andere Dienstausgaben modelliert. Diese Anteile werden berechnet und die Anteile des Jahres `jahr_abr` für die Zukunft fortgeschrieben.

```
ANTEIL_300_500 <- REGISTER_DIENST |>
    group_by(jahr) |>
    summarise(andere_dienst_ausgaben = sum(sum_taggelder[kat_leist !=
        300 & kat_leist != 500], na.rm = TRUE), ausgaben_300 =
        ↪  sum(sum_taggelder[kat_leist ==
        300], na.rm = TRUE), ausgaben_500 = sum(sum_taggelder[kat_leist ==
        500], na.rm = TRUE)) |>
    mutate(anteil_300 = ausgaben_300/andere_dienst_ausgaben,
        anteil_500 = ausgaben_500/andere_dienst_ausgaben) |>
    select(jahr, anteil_300, anteil_500)

letzte_werte <- ANTEIL_300_500[ANTEIL_300_500$jahr == jahr_abr,
    c("anteil_300", "anteil_500")]
neue_jahren <- data.frame(jahr = (jahr_abr + 1):jahr_ende, anteil_300 =
↪  letzte_werte$anteil_300,
    anteil_500 = letzte_werte$anteil_500)

ANTEIL_300_500 <- rbind(ANTEIL_300_500, neue_jahren)
```

Die Dienstausgaben für alle Kategorien ausser J+S und Jungschützenleiterkursen werden ermittelt, indem für jede Alterskategorie und Leistungsart sowie für jedes Jahr die Anzahl der Dienstleistenden nach Taggeldkategorien aufgeschlüsselt wird. Diese Anzahl wird mit der entsprechenden durchschnittlichen Dauer und den durchschnittlichen Taggeldern multipliziert. Die resultierenden Beträge werden jährlich summiert. Kinder-, Betriebs- und Betreuungszulagen werden addiert. Am Ende werden auch die anteiligen Ausgaben für die Leistungsartn 300 und 500 hinzugefügt.

```
# Total
DIENST_AUSGABEN <- DIENST_BENEF |>
    left_join(DIENST_ANTEILE, by = c("jahr", "kat_alt", "kat_leist")) |>
    left_join(DIENST_TAGGELD, by = c("jahr", "kat_alt", "kat_leist")) |>
    left_join(DIENST_DAUER, by = c("jahr", "kat_alt", "kat_leist")) |>
    full_join(EOMAX |>
        filter(jahr >= min(REGISTER_DIENST$jahr)), by = "jahr") |>
    mutate(across(everything(), ~replace_na(.x, 0))) |>
    mutate(ausgaben_dienst = dienstleistende * anteil_eomax *
        tage_eomax * eomax + dienstleistende * anteil_eomin *
        tage_eomin * eomin + dienstleistende * anteil_zwminmax *
        tage_zwminmax * taggeld) |>
    group_by(jahr) |>
    summarise(ausgaben_dienst = sum(ausgaben_dienst, na.rm = TRUE)) |>
    left_join(KZ, by = "jahr") |>
```

```
    left_join(BETRIEBZ, by = "jahr") |>
    left_join(BETREUUNGZ, by = "jahr") |>
    group_by(jahr) |>
    summarise(ausgaben_dienst = ausgaben_dienst + kinderzulagen +
        betreuungszulagen + betriebszulagen) |>
    left_join(ANTEIL_300_500, by = "jahr") |>
    mutate(ausgaben_dienst = ausgaben_dienst * (1 + anteil_300 +
        anteil_500)) |>
    select(jahr, ausgaben_dienst)
```

Aufwendungen für Abschreibungen von Rückerstattungsforderungen, die mit den Ausgaben für Dienstleistende verbunden sind, werden anteilmässig aus der totalen Abschreibungen von Rückerstattungsforderungen von der Buchhaltung errechnet.

```
# Abschreibung von Rueckerstattungsforderungen
# (anteilmaessig)
ABSCHREIB_RUECKF <- EO_ABRECHNUNG_DEF |>
    select(jahr, abschr_rueckf) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_dienst), by = "jahr") |>
    mutate(abschreib_rueckf = anteil_dienst * abschr_rueckf) |>
    select(jahr, abschreib_rueckf)
```

Gleichermassen werden die Ausgaben aus dem Beitragsanteil zu Lasten der EO den Dienstleistenden zugeteilt.

```
# Beitragsanteil zu Lasten der EO (anteilmaessig)
BTR_ANT_EO <- EO_ABRECHNUNG_DEF |>
    select(jahr, btr_ant_eo) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_dienst), by = "jahr") |>
    mutate(btr_ant_eo = anteil_dienst * btr_ant_eo) |>
    select(jahr, btr_ant_eo)
```

Wir nehmen an, dass der Anteil dieser Kosten im Vergleich zu den anderen Dienstkosten konstant bleibt. Daher können wir sie mithilfe der Projektionen der anderen Dienstausgaben weiterschreiben.

```
  DIENST_AUSGABEN <- DIENST_AUSGABEN |>
    full_join(ABSCHREIB_RUECKF, by = "jahr") |>
    full_join(BTR_ANT_EO, by = "jahr") |>
    mutate(anteil_abschreib = abschreib_rueckf / ausgaben_dienst,
           anteil_btr = btr_ant_eo / ausgaben_dienst) |>
    fill(anteil_abschreib, anteil_btr, .direction = "down") |>
    mutate(btr_ant_eo = anteil_btr * ausgaben_dienst,
           abschreib_rueckf = anteil_abschreib * ausgaben_dienst,
           ausgaben_dienst =
             ausgaben_dienst + btr_ant_eo + abschreib_rueckf) |>
```

Schliesslich werden die Ausgaben bis zum Jahr jahr_abr angepasst, sodass die Gesamtsumme mit den Buchhaltungsdaten übereinstimmt. Diese Anpassung ist erforderlich, da wir im Rahmen des Dienstmodells alle Leistungsansprüche innerhalb eines Jahres berücksichtigen (d. h., das Jahr, in dem der Dienst erbracht wird, ist für unser Modell maßgeblich, nicht das Jahr, in dem die Leistungen ausgezahlt werden, das für die Abrechnung relevant ist).

```
full_join(EO_ABRECHNUNG_DEF |>
    select(jahr, geld_leist), by = "jahr") |>
    full_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_dienst), by = "jahr") |>
    mutate(ausgaben_dienst = if_else(jahr <= jahr_abr, geld_leist *
        anteil_dienst, ausgaben_dienst)) |>
    select(jahr, ausgaben_dienst)
```

4.8.1.1 Modul mod_eo_dienst_benef.R *Dokumentation zuletzt aktualisiert am 26.08.2025*

Dieses Modul dient zur Projektion der Anzahl der Beziehenden von EO-Entschädigungen im Dienst. Es braucht drei Argumente als Input, welche weiter oben beschrieben sind.

```
mod_eo_dienst_benef <- function(PARAM_GLOBAL,
                                BEVOELKERUNG,
                                REGISTER_DIENST) {
```

Zunächst bereitet das Modul die männliche Bevölkerung der Schweiz nach denselben Alterskategorien (`kat_alt`) wie in den Input-Daten aus dem Register auf und berechnet ihr jährliches Wachstum nach Alterskategorie.

```
BEVOELKERUNG_KAT <- BEVOELKERUNG |>
    filter(jahr >= min(REGISTER_DIENST$jahr) & sex == "m" & nat ==
        "ch") |>
    group_by(jahr, alt) |>
    summarise(bevendejahr = sum(bevendejahr)) |>
    filter(alt >= 15 & alt <= 55) |>
    mutate(kat_alt = case_when(alt >= 15 & alt < 20 ~ 1, alt >=
        20 & alt < 25 ~ 2, alt >= 25 & alt < 30 ~ 3, alt >= 30 &
        alt < 35 ~ 4, alt >= 35 & alt < 40 ~ 5, alt >= 40 & alt <
        45 ~ 6, alt >= 45 & alt < 50 ~ 7)) |>
    group_by(kat_alt, jahr) |>
    summarise(bevendejahr = sum(bevendejahr)) |>
    ungroup() |>
    group_by(kat_alt) |>
    arrange(jahr) |>
    filter(jahr >= jahr_register) |>
    mutate(wr_bev = bevendejahr/first(bevendejahr) - 1) |>
    ungroup() |>
    select(jahr, kat_alt, wr_bev)
```

Anschliessend wird ein Datensatz erstellt, indem die prognostizierte Anzahl der Begünstigten gespeichert wird. Dazu wählen wir für jede Alterskategorie und jede Art von Dienstleistung (`kat_leist`) die Anzahl der Begünstigten des Jahres `jahr_register` als Ausgangspunkt (`dienstleistende_abr`). Wir verlängern den Datensatz, so dass er von `jahr_register` bis `jahr_ende` reicht.

```
DIENST_BENEF_REG <- REGISTER_DIENST |>
    filter(jahr == jahr_register) |>
    select(jahr, kat_alt, kat_leist, dienstleistende) |>
    group_by(kat_alt, kat_leist) |>
    arrange(jahr) |>
    mutate(dienstleistende_reg = last(dienstleistende)) |>
    select(-jahr, -dienstleistende) |>
    expand_grid(tibble(jahr = jahr_register:jahr_ende))
```

Schliesslich lassen wir die Zahl der Dienstleistenden pro Alterskategorie und Leistungsart mit den Wachs-

tumsraten pro Altersgruppe der Schweizer Bevölkerung wachsen.

Wir schließen J+S- und Jungschützenleiterkurse sowie Dienstleistende aus, deren Alterskategorie oder Leistungskategorie `NA` ist.

```
DIENST_BENEF <- REGISTER_DIENST |>
    select(jahr, kat_alt, kat_leist, dienstleistende) |>
    full_join(DIENST_BENEF_REG, by = c("kat_alt", "kat_leist",
        "jahr")) |>
    full_join(BEVOELKERUNG_KAT |>
        expand_grid(tibble(kat_leist = unique(REGISTER_DIENST$kat_leist))) |>
        filter(jahr >= jahr_register), by = c("jahr", "kat_alt",
        "kat_leist")) |>
    filter(!(is.na(kat_leist) | kat_leist %in% c(300, 500)) &
        !is.na(kat_alt)) |>
    group_by(kat_alt, kat_leist) |>
    mutate(dienstleistende = if_else(jahr > max(REGISTER_DIENST$jahr),
        dienstleistende_reg * (1 + wr_bev), dienstleistende)) |>
    select(jahr, kat_leist, kat_alt, dienstleistende)
```

Somit ist die Berechung der Anzhal Dienstleistende feritg und das Dataframe `DIENST_BENEF` wird an das Modul `mod_eo_dienst` zurückgegeben.

4.8.1.2 Modul mod_eo_dienst_taggeld.R *Dokumentation zuletzt aktualisiert am 26.08.2025.*

Dieses Modul dient der Berechnung der durchschnittlichen Entschädigung für Dienstpersonen, die ein Taggeld erhalten, das zwischen dem maximalen und minimalen Taggeld liegt.

Die Funktion `mod_eo_dienst_taggeld` braucht die folgende Argumente:

```
mod_eo_dienst_taggeld <- function(PARAM_GLOBAL,
                                  REGISTER_DIENST,
                                  ECKWERTE_EXTENDED,
                                  EOMAX) {
```

Zunächst werden die Registerdaten gefiltert, um J+S- und Jungschützenleiterkurse auszuschliessen und nur die notwendigen Variablen beizubehalten. Die Daten für welche die Alterskategorie unbekannt ist, werden ausgeschlossen.

```
DIENST_KAT <- REGISTER_DIENST |>
    filter(kat_leist != 300 & kat_leist != 500 & !is.na(kat_alt)) |>
    select(jahr, kat_leist, kat_alt, dienstleistende, anteil_zwminmax,
        tage_zwminmax, taggeld)
```

Danach wird, pro Alterskategorie und Leistungsart, das Wachstum des durchschnittlichen Taggeld der Dienstleistenden berechnet, die nicht die maximale oder minimale Entschädigung erhalten.

```
  WR_TAGGELD <- DIENST_KAT |>
    mutate(n_dienstleistende_zwminmax = dienstleistende * anteil_zwminmax) |>
    group_by(kat_alt, kat_leist) |>
    arrange(jahr) |>
    mutate(wr_taggeld = taggeld /lag(taggeld) - 1) |>
    ungroup() |>
```

Für jedes Jahr wird eine gewichtete durchschnittliche Wachstumsrate für alle Alterskategorie und Leistungsarten berechnet.

```
group_by(jahr) |>
    summarise(weighted_mean_taggeld = sum(wr_taggeld * n_dienstleistende_zwminmax *
        tage_zwminmax, na.rm = TRUE)/sum(n_dienstleistende_zwminmax *
        tage_zwminmax, na.rm = TRUE)) |>
    select(jahr, wr_taggeld = weighted_mean_taggeld)
```

Das jährliche Wachstum des SLI wird ab 2008 aus der Tabelle ECKWERTE_EXTENDED extrahiert.

```
# Compute yearly growth rate of SLI
WR_SLI <- ECKWERTE_EXTENDED |>
    select(jahr, lohn) |>
    mutate(wr_sli = lohn/100) |>
    filter(jahr >= min(DIENST_KAT$jahr))
```

Für die Schätzung der Entwicklung des Taggelds, werden die Jahre 2020 und 2021 (bedingt durch COVID und den besonderen SLI) aus den Daten entfernt. Anschliessend wird ein lineares Modell geschätzt, in dem das durchschnittliche Wachstum des Taggelds (`wr_taggeld`) auf die SLI-Wachstumsrate (`wr_sli`) und die Dummy-Variable (`dummy_eomax`) regressiert wird. Letztere bildet die Jahre ab, in denen die minimale und maximale Entschädigung geändert wurde.

```
# Merge taggeld and SLI yearly growth rate into a single
# data frame, add dummy eomax and take 2021 out, since the
# SLI of that year problematic is.  2020 is influenced by
# COVID, which impacted the military activities- 2020 is
# therefore also taken out of the sample.
WR_lm <- WR_TAGGELD |>
    full_join(EOMAX |>
        select(jahr, dummy_eomax) |>
        filter(jahr >= min(DIENST_KAT$jahr)), by = "jahr") |>
    full_join(WR_SLI, by = "jahr") |>
    filter(jahr != 2020 & jahr != 2021)


# Fit linear model:
model <- lm(wr_taggeld ~ wr_sli + dummy_eomax, data = WR_lm)
```

Mit dem folgenden Command, kriegt man die Ergebnisse der Regression:

```
summary(model)
```

```
##
## Call:
## lm(formula = wr_taggeld ~ wr_sli + dummy_eomax, data = WR_lm)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0043582 -0.0006111  0.0002611  0.0010579  0.0048241
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.003352   0.002012  -1.666  0.12672
## wr_sli       0.886116   0.227894   3.888  0.00302 **
## dummy_eomax  0.044370   0.003549  12.502 1.99e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 0.002794 on 10 degrees of freedom
##   (51 osservazioni eliminate a causa di valori mancanti)
## Multiple R-squared:  0.9713, Adjusted R-squared:  0.9656
## F-statistic: 169.2 on 2 and 10 DF,  p-value: 1.947e-08
```

Das Regressionsmodell untersucht den Zusammenhang zwischen dem Wachstum der Taggelder (`wr_taggeld`) und dem Wachstum des Lohnindexes (`wr_sli`), unter Kontrolle einer Dummy-Variable (`dummy_eomax`).

Die Ergebnisse zeigen:

- Der Koeffizient für den Lohnindex (`wr_sli`) beträgt 0.883 und ist mit einem p-Wert von 0.076 auf dem 10%-Niveau schwach signifikant. Das bedeutet, dass ein Anstieg des Lohnindexes tendenziell mit einem unterproportionalen Anstieg der Taggelder einhergeht. -> Dieses Resultat ist ökonomisch plausibel, da die Taggelder einem Deckelungsmechanismus unterliegen. Dadurch steigen sie langsamer als die Löhne, weshalb ein Koeffizient kleiner als 1 zu erwarten ist.
- Die Dummy-Variable (`dummy_eomax`) hat einen geschätzten Koeffizienten von 0.0444 und ist hochsignifikant ($p < 0.001$). Dies weist darauf hin, dass in den Perioden, in denen diese Dummy-Variable den Wert 1 annimmt (z. B. bei einem speziellen Ereignis oder einer politischen Anpassung), die Taggelder im Durchschnitt um rund 4,4 Prozentpunkte höher sind als in den übrigen Perioden.
- Das Bestimmtheitsmaß (R^2 = 0.971) zeigt, dass das Modell rund 97 % der Variation in den Taggeldern erklärt. Damit weist das Modell eine sehr gute Anpassung an die Daten auf.
- Die Residuen sind relativ klein und gleichmäßig verteilt, was auf eine gute Modellpassung hindeutet.

Basierend auf diesem Modell (`model`), wird die Vorhersage des durchschnitllichen Wachstums der durchschnittlichen Entschädigung (`wr_taggeld`) für die Jahre nach `jahr_register` mit dem `predict()`Befehl generiert. `newdata = pic(everything())` sorgt dafür, dass alle Spalten aus den gefilterten Daten (`WR_lm`) als Eingabewerte verwendet werden.

Schliesslich wird die Funktion `bind_rows(data_last_abr)` verwendet, um die Vorhersagen (`predict_data`) mit den letzten bekannten Daten (`data_last_year_regsiter`) zu kombinieren.

```r
# Predict the growth rate of the indemnity and combine the
# last known data point with predictions
data_last_year_regsiter <- WR_lm |>
    filter(jahr == jahr_register)

predict_data <- WR_lm |>
    filter(jahr > jahr_register) |>
    mutate(wr_taggeld = predict(model, newdata = pick(everything()))) |>
    bind_rows(data_last_year_regsiter)
```

Um die Entwicklung der Taggelder nach Alterskategorie und Leistungsart zu berechnen, wird zunächst ein Datensatz erstellt, der alle Kombinationen der relevanten Alterskategorien und Leistungsarten zusammen mit der prognostizierten Wachstumsrate enthält.

```r
# Add Leistungsart und age categories
kat_alt <- unique(DIENST_KAT$kat_alt)
kat_leist <- unique(DIENST_KAT$kat_leist)
kat_leist <- kat_leist[!kat_leist %in% c(300, 500)]
predict_data <- tidyr::crossing(kat_alt = kat_alt, kat_leist = kat_leist,
    predict_data)
```

Wir verwenden für jede Alterskategorie und Leistungsart die durchschnittlichen Taggelder des Jahres `jahr_register` als Ausgangspunkt und schreiben diese anhand der prognostizierten durchschnittlichen Wachstumsrate (`wr_taggeld`) fort. Die Daten werden dafür nach `kat_leist` (Leistungsart) und `kat_alt`

(Alterskategorie) gruppiert und innerhalb jeder Gruppe aufsteigend sortiert. Für die Fortschreibung benutzen wir die `accumulate()`-Funktion:

- Eine neue Spalte taggeld wird erstellt, die die akkumulierte Entschädigung enthält:
- `taggeld[1]`: Der erste Wert der Entschädigung in jeder Gruppe wird als Ausgangspunkt genommen.
- `accumulate(wr_taggeld[-1], .init = taggeld[1], ~ .x * (1 + .y))`: Die Funktion `accumulate` berechnet eine kumulierte Akkumulation der Taggelder:
  - `wr_taggeld[-1]`: Alle Werte der Entschädigungsrate wr_taggeld, ausser dem ersten.
  - `.init = taggeld[1]`: Der Startwert für die Akkumulation ist der erste Wert von taggeld.
  - `~ .x * (1 + .y)`: Die Akkumulation wird berechnet, indem jeder Wert mit der vorherigen Entschädigung multipliziert und die Entschädigungsrate `(.y)` berücksichtigt wird.
  - `[-1]`: Das erste Element der Akkumulation (der Initialwert) wird entfernt, um die Akkumulation ab dem zweiten Jahr zu starten.

Die berechneten akkumulierten Taggelder werden mit den ursprünglichen Entschädigungsdaten aus dem `DIENST_KAT`-Datensatz zusammengeführt.

```
# Calculate indemnity accumulation for the military
DIENST_TAGGELD <- predict_data |>
    filter(jahr >= jahr_register) |>
    full_join(REGISTER_DIENST |>
        select(jahr, kat_alt, kat_leist, taggeld) |>
        filter(jahr == jahr_register & kat_leist != 300 & kat_leist !=
            500) |>
        drop_na(), by = c("jahr", "kat_alt", "kat_leist")) |>
    group_by(kat_leist, kat_alt) |>
    arrange(jahr) |>
    mutate(taggeld = c(taggeld[1], accumulate(wr_taggeld[-1],
        .init = taggeld[1], ~.x * (1 + .y))[-1])) |>
    select(jahr, kat_alt, kat_leist, taggeld) |>
    filter(jahr > jahr_register) |>
    bind_rows(DIENST_KAT |>
        select(jahr, kat_alt, kat_leist, taggeld))
```

Damit sind die Projektionen der durchschnittlichen Taggelder pro Alterskategorie und Leistungsart für Dienstleistende, die nicht die maximale oder die minimale Entschädigung bekommen, abgeschlossen und der Dataframe `DIENST_TAGGELD` wird an das Modul `mod_eo_dienst` zurückgegeben:

```
# --Output----------------------------------------------------------------

return(list(DIENST_TAGGELD = DIENST_TAGGELD))
```

4.8.1.3 Modul mod_eo_dienst_anteile.R *Dokumentation zuletzt aktualisiert am 16.10.2025*

Dieses Modul dient zur Prognose der Entwicklung des Anteils der Dienstleistenden, die die maximale Entschädigung (`anteil_eomax`), die Mindestentschädigung (`anteil_eomin`) oder eine Entschädigung in Höhe von 80% ihres vordienstlichen Einkommens (`anteil_zwminmax`) erhalten.

Es benötigt folgende Argumente:

```
mod_eo_dienst_anteile <- function(PARAM_GLOBAL,
                                  REGISTER_DIENST,
                                  ECKWERTE_EXTENDED,
                                  EOMAX,
```

```
                                    TAGGELD_DIENST) {
```

Zunächst bereiten wird die Daten vor, die wir benötigen, indem wir J & S und die Jugschützenleiterkurse sowie alle Variablen ausschliessen, die für dieses Modul nicht erforderlich sind.

```
# Prepare EO_DIENST dataset by selecting required columns
# and filtering to take 'Jungschützenkurse' and 'J+S' out
DIENST_KAT <- REGISTER_DIENST |>
    filter(kat_leist != 300 & kat_leist != 500 & !is.na(kat_alt)) |>
    select(jahr, kat_leist, kat_alt, dienstleistende, anteil_zwminmax,
        anteil_eomax, anteil_eomin)
```

Wir beginnen damit, Prognosen bezüglich des Anteils der Dienstleistenden zu erstellen, die die maximale Entschädigung erhalten. Dafür wird wird die jährliche Wachstumsrate (`wr_anteil_eomax`) des Anteils der Dienstleistenden, die die maximale Entschädigung pro Alterskategorie (`kat_alt`) und Leistungsart (`kat_leist`) und pro Periode - nähmlich ob es sich um ein Jahr mit einer Anpassung der maximalen Entschädigung oder um ein Jahr ohne Anpassung der Betrag handelt - wie folgt berechnet:

```
# Anteil EOMAX
# ------------------------------------------------------------------
# Compute yearly growth rate of the share of men getting
# EOMAX
WR_ANTEIL_EOMAX <- DIENST_KAT |>
    group_by(jahr, kat_alt, kat_leist) |>
    summarise(anteil_eomax = sum(anteil_eomax * dienstleistende,
        na.rm = TRUE)/sum(dienstleistende, na.rm = TRUE)) |>
    ungroup() |>
    group_by(kat_alt, kat_leist) |>
    arrange(jahr) |>
    mutate(wr_anteil_eomax = (anteil_eomax - lag(anteil_eomax))/lag(anteil_eomax)) |>
    select(jahr, kat_alt, wr_anteil_eomax) |>
    full_join(EOMAX |>
        select(jahr, dummy_eomax), by = "jahr") |>
    filter(jahr > min(DIENST_KAT$jahr) & jahr <= max(DIENST_KAT$jahr)) |>
    select(jahr, kat_alt, kat_leist, wr_anteil_eomax, dummy_eomax) |>
    group_by(dummy_eomax, kat_alt, kat_leist) |>
    summarise(wr_anteil_eomax = mean(wr_anteil_eomax, na.rm = TRUE)) |>
    mutate(wr_anteil_eomax = if_else(is.infinite(wr_anteil_eomax) |
        wr_anteil_eomax == -1 | is.na(wr_anteil_eomax), 0, wr_anteil_eomax))
```

Wir verwenden für jede Alterskategorie und Leistungsart die durchschnittlichen Anteile der Dienstleistenden aus dem Jahr `jahr_register`, die die maximale Entschädigung erhalten, als Ausgangspunkt und berechnen dann den zukünftigen `anteil_eomax` mithilfe der `accumulate()`-Funktion und des jährlichen Wachstums (`wr_anteil_eomax`).

```
# Calculate indemnity accumulation for the military
ANTEIL_EOMAX <- expand.grid(jahr = min(DIENST_KAT$jahr):jahr_ende,
    kat_alt = unique(DIENST_KAT$kat_alt), kat_leist = unique(DIENST_KAT$kat_leist)) |>
    left_join(DIENST_KAT |>
        select(jahr, kat_leist, kat_alt, anteil_eomax), by = c("jahr",
        "kat_alt", "kat_leist")) |>
    left_join(EOMAX |>
        select(jahr, dummy_eomax)) |>
    left_join(WR_ANTEIL_EOMAX) |>
    group_by(kat_alt, kat_leist) |>
```

```
    arrange(jahr) |>
    mutate(anteil_eomax = c(anteil_eomax[1], accumulate(wr_anteil_eomax[-1],
        .init = anteil_eomax[1], ~.x * (1 + .y))[-1])) |>
    select(jahr, kat_alt, kat_leist, anteil_eomax) |>
    filter(jahr > jahr_register) |>
    bind_rows(DIENST_KAT |>
        select(jahr, kat_alt, kat_leist, anteil_eomax)) |>
    mutate(anteil_eomax = if_else(anteil_eomax > 1, 1, anteil_eomax))
```

Die Daten werden nach `kat_leist` (Leistungsart) und `kat_alt` (Alterskategorie) gruppiert, sodass jede Gruppe separat behandelt wird. Sie werden innerhalb jeder Gruppe nach Jahr aufsteigend sortiert. Eine neue Spalte taggeld wird erstellt, die die akkumulierte Entschädigung enthält:

- `anteil_eomax[1]`: Der erste Wert der Anteile Dienstleistende bei `eomax` in jeder Gruppe wird als Ausgangspunkt genommen.
- `accumulate(wr_anteil_eomax[-1], .init = anteil_eomax[1], ~ .x * (1 + .y))`: Hier wird die Variable `wr_anteil_eomax` (die prognostizierte Wachstumsrate) verwendet, um den `anteil_eomax` in jedem Jahr zu berechnen, basierend auf der Wachstumsrate des vorherigen Jahres. Das accumulate berechnet schrittweise die Werte, indem es den vorherigen Wert mit der Wachstumsrate multipliziert.
  - `.init = anteil_eomax[1]` gibt den Startwert der Berechnung an, und `~ .x * (1 + .y)` bedeutet, dass der vorherige Wert `(.x)` mit dem Wachstumsfaktor `(1 + .y)`, (wobei `.y` die Wachstumsrate ist) multipliziert wird.
  - `[-1]`: entfernt das erste Element, da es sich bereits im urprünglichen Wert von `anteil_eomax[1]` befindet.

Die berechneten akkumulierten Daten werden mit den ursprünglichen Entschädigungsdaten aus dem `DIENST_KAT`-Datensatz zusammengeführt.

Dann wird überprüft, ob die Projektion der Höhe der Entschädigungen für Personen, die ein Taggeld unter EOMAX erhalten, mit den vorhergesagten Anteilen übereinstimmt. Dazu wird für pro Jahr, Leistungsart und Alterskategorie überprüft, ob der durchschnittliche Entschädigungsbetrag den Betrag der maximalen Entschädigung übersteigt. Wenn dies der Fall ist, wird `anteil_eomax` entsprechend auf 1 gestezt.

```
# Make sure that the share are consistent with the daily
# indemnities
DIENST_TAGGELD_EOMAX <- DIENST_TAGGELD |>
    full_join(EOMAX |>
        filter(jahr >= min(DIENST_TAGGELD$jahr)) |>
        select(jahr, eomax), by = "jahr")

ANTEIL_EOMAX <- ANTEIL_EOMAX |>
    full_join(DIENST_TAGGELD_EOMAX, by = c("jahr", "kat_alt",
        "kat_leist")) |>
    mutate(anteil_eomax = if_else(is.na(taggeld), anteil_eomax,
        if_else(taggeld >= eomax, 1, anteil_eomax))) |>
    select(jahr, kat_alt, kat_leist, anteil_eomax)
```

Damit sind die Projektionen des Anteils der Dienstleistenden, die die maximale Entschädigung erhalten (`anteil_eomax`), für jede Alterskategorie und Leistungsart abgeschlossen.

Der Anteil der Dienstleistenden, die die Mindestentschädigung erhalten, sollte pro Alterskategorie und Leistungsart weitgehend konstant bleiben, da er insbesondere von der Art der Dienstleistung (z. B. Rekruten) oder dem Alter der Teilnehmer abhängt (die Jüngsten haben noch nicht gearbeitet). Aus diesem Grund schreiben wir den Anteil der Dienstleistenden, die eine Mindestentschädigung erhalten, für jede Alterskategorie und Leistungsart als Durchschnitt der letzten drei Jahre dieses Anteils fort.

```
# Anteil EOMIN
# ------------------------------------------------------------------
MEAN_EOMIN <- DIENST_KAT |>
    group_by(kat_alt, kat_leist) |>
    arrange(jahr) |>
    mutate(mean_anteil_eomin = mean(tail(anteil_eomin, 3), na.rm = TRUE)) |>
    select(kat_alt, kat_leist, mean_anteil_eomin) |>
    unique()

MEAN_EOMIN <- tidyr::crossing(MEAN_EOMIN, jahr = (jahr_register +
    1):jahr_ende)

ANTEIL_EOMIN <- DIENST_KAT |>
    select(jahr, kat_alt, kat_leist, anteil_eomin) |>
    full_join(MEAN_EOMIN, by = c("jahr", "kat_alt", "kat_leist")) |>
    mutate(anteil_eomin = if_else(jahr > jahr_register, mean_anteil_eomin,
        anteil_eomin)) |>
    select(jahr, kat_alt, kat_leist, anteil_eomin)
```

Schliesslich wird der Anteil Dienstleistende, die weder die minimale noch die maximale Entschädigung erhalten, als der verbleibender Anteil berechnet, indem der Anteil `anteil_eomax` und der Anteil `anteil_eomin` für jede Gruppe von 1 subtrahiert werden. Es wird noch überprüft, dass die Summe der vorhergesagten `anteil_eomax` und `anteil_eomin`nicht grösser als 1 ist. Falls er der Fall ist, wird der Anteil `anteil_eomax` so korriiert, dass er gleich `1-anteil_eomin` ist.

```
# Anteil zwischen EOMIN & EOMAX
# ------------------------------------------------
DIENST_ANTEILE <- ANTEIL_EOMAX |>
    full_join(ANTEIL_EOMIN, by = c("jahr", "kat_alt", "kat_leist")) |>
    mutate(anteil_zwminmax = if_else(1 - anteil_eomax - anteil_eomin >
        0, 1 - anteil_eomax - anteil_eomin, 0), anteil_eomax = if_else(1 -
        anteil_eomax - anteil_eomin > 0, anteil_eomax, 1 - anteil_eomin)) |>
    filter(jahr > jahr_register) |>
    bind_rows(DIENST_KAT |>
        select(jahr, kat_leist, kat_alt, anteil_eomax, anteil_eomin,
            anteil_zwminmax))
```

Somit ist die Berechnung des Dataframes `DIENST_ANTEILE` abgeschlossen, und es kann an das Modul `mod_eo_dienst` zurückgegeben werden:

```
# --Output--------------------------------------------------------------------

return(list(DIENST_ANTEILE = DIENST_ANTEILE))
```

4.8.1.4 Modul mod_eo_dienst_dauer.R *Dokumentation zuletzt aktualisiert am 27.08.2025*

Dieses Modul dienst zur Berechnung der duchschnittlichen Dauer jeder Art von Service. Zuerst werden die Daten des Registers gefiltert, um nur die für dieses Modul erforderlichen Daten zu behalten.

```
# Prepare EO_DIENST data set by selecting required columns
# and filtering to take 'Jungschützenkurse' and 'J+S' out
DIENST_KAT <- REGISTER_DIENST |>
    filter(kat_leist != 300 & kat_leist != 500) |>
    select(jahr, kat_leist, kat_alt, dienstleistende, tage_eomax,
        tage_zwminmax, tage_eomin, anteil_eomax, anteil_eomin,
```

```
        anteil_zwminmax)
```

Da die Dienstzeit hauptsächlich von der Art des Dienstes (`kat_leist`) und der Taggeldeskategorie (`eomax, eomin, zwminmax`) und nicht vom Alter der dienstleistenden Person abhängt, aggregieren wir die Daten (vor allem die Dauer des Dienstes) nach der Art des Dienstes und Taggeldkategorie.

```
# Aggregate the data
DIENST_AGG_BY_LEISTUNG <- DIENST_KAT |>
    group_by(jahr, kat_leist) |>
    summarise(tage_eomax = sum(tage_eomax * dienstleistende *
        anteil_eomax, na.rm = TRUE)/sum(dienstleistende * anteil_eomax,
        na.rm = TRUE), tage_eomin = sum(tage_eomin * dienstleistende *
        anteil_eomin, na.rm = TRUE)/sum(dienstleistende * anteil_eomin,
        na.rm = TRUE), tage_zwminmax = sum(tage_zwminmax * dienstleistende *
        anteil_zwminmax, na.rm = TRUE)/sum(dienstleistende *
        anteil_zwminmax, na.rm = TRUE)) |>
    ungroup()
```

Nach der geltenden Ordnung besteht kein Grund zu der Annahme, dass die Dauer des Dienstes (je nach Art der Dienstleistung und Entschädigung) in Zukunft stark variieren wird. Daher nehmen wir die durchschnittliche Dauer der letzten drei Jahre als Prognose.

```
# Compute the mean over the last 3 years
MEAN_DAUER <- DIENST_AGG_BY_LEISTUNG |>
    group_by(kat_leist) |>
    arrange(jahr) |>
    mutate(mean_dauer_eomax = mean(tail(tage_eomax, 3)), mean_dauer_eomin =
    ↪  mean(tail(tage_eomin,
        3)), mean_dauer_zwminmax = mean(tail(tage_zwminmax, 3))) |>
    select(kat_leist, tage_eomax = mean_dauer_eomax, tage_eomin = mean_dauer_eomin,
        tage_zwminmax = mean_dauer_zwminmax) |>
    unique()
```

Wir erweitern die Tabelle, die diese Durchschnittsdauer enthält, durch eine Kombination der Jahre (zwischen `jahr_register+1`und `jahr_ende`) und der einzigarten `kat_alt`.

```
# Add jahr and kat_alt
kat_alt <- unique(DIENST_KAT$kat_alt[!is.na(DIENST_KAT$kat_alt)])

MEAN_DAUER <- tidyr::crossing(MEAN_DAUER, jahr = (jahr_register +
    1):jahr_ende, kat_alt = kat_alt)
```

Schliesslich werden die historischen Daten und die Prognosen in einem einzigen Datensatz (`DIENST_DAUER`) zusammengefügt und an das Modul `mod_eo_dienst_dauer` zurückgegeben.

```
DIENST_DAUER <- DIENST_KAT |>
    filter(!is.na(kat_alt)) |>
    select(jahr, kat_leist, kat_alt, tage_eomax, tage_zwminmax,
        tage_eomin) |>
    bind_rows(MEAN_DAUER)

# Output------------------------------------------------------------------------
return(list(DIENST_DAUER = DIENST_DAUER))
```

4.8.1.5 Modul mod_eo_dienst_zulage.R *Dokumentation zuletzt aktualisiert am 27.08.2025*

Dieses Modul berechnet die Ausgaben für die Kinderzulagen, Betriebszulagen und Zulagen für Betreuungskosten.

Es benötigt die folgende Argumente:

```
mod_eo_dienst_zulage <- function(PARAM_GLOBAL,
                                 EOMAX,
                                 REGISTER_DIENST) {
```

Diese sind im Kapitel 4.8 beschrieben.

Wie auch für den Rest des Dienstmodells, werden die Ausgaben für J&S und Jungschützenleiterkurse ausgeschlossen, da sie separat modelliert werden.

```
# Filter J&S und Jungenschutz out
REGISTER_DIENST <- REGISTER_DIENST |>
    filter(!(is.na(kat_leist) | kat_leist %in% c(300, 500)))
```

Anschliessend wird eine prozentuale Wachstumsrate für `eomax` berechnet. Das letzte Registersjahr dient als Referenzjahr für den Wert der maximalen Entschädigung.

```
# Growth of EOMAX (compared to jahr_register)
WR_EOMAX <- EOMAX |>
    arrange(jahr) |>
    mutate(ref_eomax = eomax[jahr == jahr_register], wr_eomax = (eomax -
        ref_eomax)/ref_eomax)
```

Um die Kinderzulagen fortzuschreiben, werden die Registerdaten zunächst auf die relevanten Variablen eingeschränkt. Anschliessend werden die Kinderzulagen über alle im Datensatz verbliebenen Leistungsarten pro Jahr summiert. Die summierten Zulagen desletzten Registersjahres werden zunächst für die Vorhersagezeitraum übernommen. Anschliessend wird die prozentuale Wachstumsrate der maximalen Entschädigung an die Daten gemerged und für die Vorhersagejahre auf die Kinderzulagen angewendet.

```
# Sum of Kinderzulagen, grows with EOMAX
KZ <- REGISTER_DIENST |>
    select(jahr, kat_alt, kat_leist, sum_kz) |>
    group_by(jahr) |>
    summarise(kinderzulagen = sum(sum_kz)) |>
    bind_rows(tibble(jahr = (jahr_register + 1):jahr_ende)) |>
    arrange(jahr) |>
    fill(kinderzulagen, .direction = "downup") |>
    ungroup() |>
    left_join(WR_EOMAX |>
        select(jahr, wr_eomax), by = "jahr") |>
    arrange(jahr) |>
    mutate(kinderzulagen = if_else(jahr > jahr_register, kinderzulagen[jahr ==
        jahr_register] * (1 + wr_eomax), kinderzulagen)) |>
    select(jahr, kinderzulagen)
```

Die Betriebszulagen werden in gleicher Form fortgeschrieben. Die Registerdaten werden zunächst auf die relevanten Variablen eingeschränkt. Die Betriebszulagen werden über alle im Datensatz verbliebenen Leistungsarten pro Jahr summiert. Die summierten Zulagen desletzten Registersjahres werden zunächst für die Vorhersagezeitraum übernommen. Anschliessend wird die prozentuale Wachstumsrate der maximalen Entschädigung an die Daten gemerged und für die Vorhersagejahre auf die Betriebszulagen angewendet.

```
# Sum of Betriebzulagen, grows with EOMAX
BETRIEBZ <- REGISTER_DIENST |>
    select(jahr, kat_alt, kat_leist, sum_betriebsz) |>
```

```
    group_by(jahr) |>
    summarise(betriebszulagen = sum(sum_betriebsz)) |>
    bind_rows(tibble(jahr = (jahr_register + 1):jahr_ende)) |>
    arrange(jahr) |>
    fill(betriebszulagen, .direction = "downup") |>
    ungroup() |>
    left_join(WR_EOMAX |>
        select(jahr, wr_eomax), by = "jahr") |>
    arrange(jahr) |>
    mutate(betriebszulagen = if_else(jahr > jahr_register, betriebszulagen[jahr ==
        jahr_register] * (1 + wr_eomax), betriebszulagen)) |>
    select(jahr, betriebszulagen)
```

Auch die Fortschreibung der Zulagen für Betreuungskosten erfolgt nach der gleichen Logik. Die Registerdaten werden zunächst auf die relevanten Variablen eingeschränkt. Die Zulagen für Betreuungskosten werden über alle im Datensatz verbliebenen Leistungsarten pro Jahr summiert. Die summierten Zulagen desletzten Registersjahres werden zunächst für die Vorhersagezeitraum übernommen. Anschliessend wird die prozentuale Wachstumsrate der maximalen Entschädigung an die Daten gemerged und für die Vorhersagejahre auf die Zulagen für Betreuungskosten angewendet.

```
# Sum of Betreuungszulage
BETREUUNGZ <- REGISTER_DIENST |>
    select(jahr, kat_alt, kat_leist, sum_betreuungsz) |>
    group_by(jahr) |>
    summarise(betreuungszulagen = sum(sum_betreuungsz)) |>
    bind_rows(tibble(jahr = (jahr_register + 1):jahr_ende)) |>
    arrange(jahr) |>
    fill(betreuungszulagen, .direction = "downup") |>
    ungroup() |>
    left_join(WR_EOMAX |>
        select(jahr, wr_eomax), by = "jahr") |>
    arrange(jahr) |>
    mutate(betreuungszulagen = if_else(jahr > jahr_register,
        betreuungszulagen[jahr == jahr_register] * (1 + wr_eomax),
        betreuungszulagen)) |>
    select(jahr, betreuungszulagen)
```

Schlussendlich werden die Dataframes `KZ`, `BETRIEBZ` und `BETREUUNGZ` an das Modul `mod_eo_dienst` zurückgegeben.

```
#---Output--------------------------------------------------------------------
return(list(KZ = KZ, BETRIEBZ = BETRIEBZ, BETREUUNGZ = BETREUUNGZ))
```

4.8.2 Modul mod_eo_msu.R

Dokumentation zuletzt aktualisiert am 27.08.2025

Dieses Modul berechnet die Ausgaben im Zusammenhang mit dem Mutterschaftsurlaub. Es besteht aus verschiedenen Untermodulen, um die Anzahl der in der Schweiz und im Ausland wohnhaften Leistungsempfänger sowie die durchschnittliche Höhe der Entschädigung für Frauen, die nicht `eomax` erhalten. Es benötigt die folgende Argumente:

```
mod_eo_msu <- function(PARAM_GLOBAL,
                       REGISTER_MSU,
                       LEBENDGEBURTEN_FRAU,
                       BEV_SCENARIO,
```

```
                        FRONTALIERS,
                        EOMAX,
                        ECKWERTE_EXTENDED,
                        EO_ABRECHNUNG_DEF,
                        EO_AUSGABEN_ANTEILE,
                        BEVOELKERUNG) {
```

Diese sind im Kapitel 4.8 beschrieben.

Zuerst wird das Modul aufgerufen, das die Anzahl Mutterschaftsurlaube mit Domizil in der Schweiz prognostiziert.

```
# Number of maternity leaves for Swiss residents
tl_eo_msu_benef_ch <- mod_eo_msu_benef_ch(PARAM_GLOBAL, REGISTER_MSU,
    LEBENDGEBURTEN_FRAU, BEV_SCENARIO, EOMAX, BEVOELKERUNG)
```

Die Ergebnisse dieses Modules werden im Datensatz `MSU_BENEF_CH` gespeichert.

```
MSU_BENEF_CH <- tl_eo_msu_benef_ch$MSU_BENEF_CH
```

Anschliessend wird das Modul, das die Anzahl der Mutterschaftsurlaube mit Domizil im Ausland prognostiziert aufgerufen und die Ergebnisse im Datensatz `MSU_BENEF_AUS` gespeichert.

```
# Number of maternity leaves for non-Swiss residents
tl_eo_msu_benef_au <- mod_eo_msu_benef_au(PARAM_GLOBAL, REGISTER_MSU,
    FRONTALIERS, MSU_BENEF_CH)

MSU_BENEF_AU <- tl_eo_msu_benef_au$MSU_BENEF_AU
```

Die Ergebnisses dieser beiden Module werden in einem einzigen Datensatz gespeichert, der `MSU_BENEF` genannt wird.

```
# Number of maternity leaves
MSU_BENEF <- MSU_BENEF_CH |>
    full_join(MSU_BENEF_AU)
```

Zunächst wird das Untermodul abgerufen, das die Entschädigungsbeträge für Frauen berechnet, die nicht `eomax` erhalten. Die Ergebnisse sind im Datensatz `MSU_TAGGELD` gespeichert.

```
# Daily indemnity for those not getting the maximal
# benefits
tl_eo_msu_taggeld <- mod_eo_msu_taggeld(PARAM_GLOBAL, REGISTER_MSU,
    EOMAX, ECKWERTE_EXTENDED)

MSU_TAGGELD <- tl_eo_msu_taggeld$MSU_TAGGELD
```

Bevor wir die Kosten für den Mutterschaftsurlaub berechnen, setzen wir den geschätzten Anteil der Mütter, die `eomax` erhalten auf 1, falls das durchschnittliche Taggeld pro Alterskategorie, Domizil und Jahr grösser gleich dem maximalen Taggeld ist.

Für jede Alterskategorie, Domizil und Jahr, werden die Anzahl Mutterschaftsurlaube gemäss der errechneten Anteile verteilt, die `eomax` und eine niedrige Entschädigung erhalten.

Die Ausgaben werden dann zunächst pro Alterskategorie, Domizil, Taggeldkategorie (`eomax`oder nicht) und Jahr berechnet, indem die Anzahl der Beziehenden mit dem durchschnittlichen Taggeldbetrag und 98 Tagen multipliziert wird. Die Ergebnisse werden anschliessend pro Jahr aufsummiert, um die jährlichen Ausgaben zu ermitteln.

```
# Maternity Expenses
MSU_AUSGABEN <- MSU_BENEF |>
    left_join(MSU_TAGGELD, by = c("jahr", "kat", "dom")) |>
    full_join(EOMAX |>
        select(jahr, eomax), by = "jahr") |>
    drop_na() |>
    group_by(jahr, kat, dom) |>
    mutate(anteil_eomax = if_else(taggeld >= eomax, 1, anteil_eomax),
        mutterschaftsurlaub_eomax = mutterschaftsurlaub * anteil_eomax,
        mutterschaftsurlaub_ohne_eomax = mutterschaftsurlaub *
            (1 - anteil_eomax), tot_msu_aus = 98 * mutterschaftsurlaub_ohne_eomax *
            taggeld + 98 * mutterschaftsurlaub_eomax * eomax) |>
    group_by(jahr) |>
    summarize(aus_tot_mut = sum(tot_msu_aus, na.rm = TRUE))
```

Aufwendungen für Abschreibungen von Rückerstattungsforderungen, die mit den Ausgaben für Mutterschaftsurlaube verbunden sind, werden als Anteil der totalen Abschreibungen von Rückerstattungsforderungen errechnet.

```
# Abschreibung von Rueckerstattungsforderungen
# (anteilmaessig)
ABSCHREIB_RUECKF <- EO_ABRECHNUNG_DEF |>
    select(jahr, abschr_rueckf) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_msu), by = "jahr") |>
    mutate(abschreib_rueckf = anteil_msu * abschr_rueckf) |>
    select(jahr, abschreib_rueckf)
```

Der Beitragsanteil zulasten der EO, der mit Mutterschaftsurlauben einhergeht, wird als Anteil des totalen Beitragsanteils errechnet.

```
# Beitragsanteil zu Lasten der EO (anteilmaessig)
BTR_ANT_EO <- EO_ABRECHNUNG_DEF |>
    select(jahr, btr_ant_eo) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_msu), by = "jahr") |>
    mutate(btr_ant_eo = anteil_msu * btr_ant_eo) |>
    select(jahr, btr_ant_eo)
```

Wir nehmen an, dass der Anteil dieser Kosten im Vergleich zu den anderen Mutterschaftsurlaubskosten konstant bleibt und schreiben sie mithilfe der Projektionen der anderen Mutterschaftsurlaubskosten weiter. Die Totalen Ausgaben für den Mutterschaftsurlaub werden als Summe der errechneten Kosten `aus_tot_mut`, der Abschreibungen von Rückerstattungsforderungen und des Beitragsanteils zulasten der EO errechnet.

```
  # Add Abschreibung und Beitragsanteil zu den MSU-Ausgaben
  MSU_AUSGABEN <- MSU_AUSGABEN |>
    full_join(ABSCHREIB_RUECKF, by = "jahr") |>
    full_join(BTR_ANT_EO, by = "jahr") |>
    mutate(anteil_abschreib = abschreib_rueckf / aus_tot_mut,
           anteil_btr = btr_ant_eo / aus_tot_mut) |>
    fill(anteil_abschreib, anteil_btr, .direction = "down") |>
    mutate(btr_ant_eo = anteil_btr * aus_tot_mut,
           abschreib_rueckf = anteil_abschreib * aus_tot_mut,
           aus_tot_mut =
```

```
            aus_tot_mut + btr_ant_eo + abschreib_rueckf) |>
```

Schliesslich werden die Ausgaben bis zum Jahr `jahr_abr` so angepasst, dass die Gesamtsumme mit den Buchhaltungsdaten (`geld_leist`) übereinstimmt. Etwaige Abweichungen kommen z.B. bei Leistungen für die Elternschaft davon, dass sich der Beginn des Leistungsanspruchs (Geburt eines Kindes) nicht unbedingt mit dem Abrechnungszeitpunkt übereinstimmt.

```
full_join(EO_ABRECHNUNG_DEF |>
    select(jahr, geld_leist), by = "jahr") |>
    full_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_msu), by = "jahr") |>
    mutate(aus_tot_mut = if_else(jahr <= jahr_abr, geld_leist *
        anteil_msu, aus_tot_mut)) |>
    select(jahr, aus_tot_mut)
```

4.8.2.1 Modul mod_eo_msu_benef_ch.R *Dokumentation zuletzt aktualisiert am 16.10.2025*

Dieses Modul schätzt die Anzahl der Mutterschaftsurlaube für Frauen mit Wohnsitz in der Schweiz. Die nötigen Argumente sind:

```
mod_eo_msu_benef_ch <- function(PARAM_GLOBAL,
                                REGISTER_MSU,
                                LEBENDGEBURTEN_FRAU,
                                BEV_SCENARIO,
                                EOMAX,
                                BEVOELKERUNG) {
```

Zuerst werden die Registerdaten gefiltert, sodass nur Frauen mit Wohnsitz in der Schweiz (`dom` = 1) im Sample bleiben.

```
MSU_CH <- REGISTER_MSU |>
    filter(dom == 1)
```

Anschliessend werden die zukünftigen jährlichen Wachstumsraten der Geburten pro Altersgruppe der Mutter gemäss den demografischen Szenarien des BFS berechnet.

```
# Calculate annual growth rates for births
SCENARIO_WR_LEBENDGEBURTEN <- BEV_SCENARIO |>
    filter(scenario == PARAM_GLOBAL$bev_scenario & geburtmutter !=
        0) |>
    mutate(kat = case_when(alt >= 15 & alt <= 19 ~ 1, alt >=
        20 & alt <= 24 ~ 2, alt >= 25 & alt <= 29 ~ 3, alt >=
        30 & alt <= 34 ~ 4, alt >= 35 & alt <= 39 ~ 5, alt >=
        40 & alt <= 44 ~ 6, alt >= 45 ~ 7)) |>
    group_by(kat, jahr) |>
    summarize(anzahl_geburten = sum(geburtmutter, na.rm = TRUE)) |>
    arrange(jahr) |>
    filter(jahr >= max(LEBENDGEBURTEN_FRAU$jahr)) |>
    mutate(wr_geburten = anzahl_geburten/anzahl_geburten[1] -
        1) |>
    filter(jahr != max(LEBENDGEBURTEN_FRAU$jahr)) |>
    ungroup() |>
    select(jahr, kat, wr_geburten)
```

Um die BFS-Geburtenszenarien nach Alterskategorie der Mutter mit der letzten Beobachtungen zu verknüpfen, wird der durchschnitt den letzen 3 Beobachtungen als Anfangspunkt benutzt.

```
LEBENDGEBURTEN_hist <- LEBENDGEBURTEN_FRAU |>
    select(jahr, kat, anzahl_geburten) |>
    filter(!is.na(kat)) |>
    group_by(kat) |>
    arrange(kat, desc(jahr)) |>
    mutate(anzahl_geburten_hist = mean(head(anzahl_geburten,
        3))) |>
    ungroup()
```

Die Wachstumsraten der Szenarien werden nun auf diese Ausgangspunkte angewendet.

```
# Adapts scenarios to the year of the accounting year
LEBENDGEBURTEN_SZENARIO_ALTER_MUTTER_fc <- SCENARIO_WR_LEBENDGEBURTEN |>
    filter(jahr > max(LEBENDGEBURTEN_FRAU$jahr)) |>
    select(jahr, kat, wr_geburten) |>
    full_join(LEBENDGEBURTEN_hist |>
        select(kat, anzahl_geburten_hist) |>
        unique(), by = "kat") |>
    mutate(anzahl_geburten = anzahl_geburten_hist * (1 + wr_geburten)) |>
    select(jahr, kat, anzahl_geburten)
```

Um eine vollständige Zeitreihe der Geburten nach Altersgruppe der Mutter zu erhalten, fügen wir die historischen Daten und die Projektionen in derselben Tabelle mit der Bezeichnung `LEBENDGEBURTEN_fc` zusammen.

```
LEBENDGEBURTEN_fc <- LEBENDGEBURTEN_FRAU |>
    filter(!is.na(kat)) |>
    full_join(LEBENDGEBURTEN_SZENARIO_ALTER_MUTTER_fc, by = c("jahr",
        "kat", "anzahl_geburten")) |>
    arrange(jahr)
```

Aus dem BFS-Erwerbsbevölkerungszenario wird die Erwerbsquote in der Schweiz pro Geschlecht und Alterskategorie berechnet (`erwerbsquote`). Dafür verwenden wir die alterspezifische Erwerbsquote und Erwerbsbevölkerung und berechnen zunächst die Nichterwerbsbevölkerung. Um die aggregierte Erwerbsquote auf Ebene Altersgruppen zu berechnen verwenden wir die folgende Formel $Erwerbsquote = \frac{Erwerbsbevölkerung}{Erwerbsbevölkerung + Nichterwerbsbevölkerung}$.

Dieses Vorgehen stellt sicher, dass die altersgruppenspezifische Erwerbsquote richtig gewichtet ist.

```
  # Compute labor force participation rate of women by age group
  ERWERBSBEVOELKERUNG <- BEVOELKERUNG  |>
      filter(alt >= 15, alt <= 49, !is.na(erwbev))  |>
      mutate(
          nerwbev = (1 - (erwq / 100)) / (erwq / 100) * erwbev, # Nichterwerbsbevölkerung
          ↪  ausrechnen, Formel erwq=erwbev/(erwbev+nerwbev) nutzen (umformen)
          kat = case_when(
              alt >= 15 & alt <= 19 ~ 1,
              alt >= 20 & alt <= 24 ~ 2,
              alt >= 25 & alt <= 29 ~ 3,
              alt >= 30 & alt <= 34 ~ 4,
              alt >= 35 & alt <= 39 ~ 5,
              alt >= 40 & alt <= 44 ~ 6,
              alt >= 45 & alt <= 53 ~ 7 #53 weil es gibt weniger als 10 Geburten mit
↪  einer Mutter über 53
              )
          ) |>
```

```
      group_by(jahr, sex, kat) |>
            summarise(
                nerwbev = sum(nerwbev, na.rm = TRUE),
                erwbev = sum(erwbev, na.rm = TRUE)
                ) |>
      ungroup() |>
      mutate(erwerbsquote = erwbev / (erwbev + nerwbev))
```

Hier wird angenommen, dass Frauen, die 54 Jahre oder älter sind, nicht relevant sind, um die Entwicklung der Erwerbsquote der Mütter abzubilden.

Der Datensatz für die Prognose der Anzahl Mutterschaftsurlaube wird vorbereitet. Dazu werden die historischen Daten der Anzahl Mutterschaftsurlaube und die historischen Daten und Prognosen der Anzahl Geburten und der Erwersquote von Frauen in der Schweiz nach Altersgruppe in `MSU_CH_fc` zusammengetragen. Der Logarithmus der Anzahl Mutterschaftsurlaube und Geburten wird berrechnet.

```
# Prepare MSU_CH_fc dataset, union of register, births
# according to mother's age and activity rates for female
MSU_CH_fc <- MSU_CH |>
    full_join(LEBENDGEBURTEN_fc, by = c("jahr", "kat")) |>
    full_join(ERWERBSBEVOELKERUNG |>
        filter(sex == "f") |>
        select(jahr, kat, erwerbsquote), by = c("jahr", "kat")) |>
    filter(jahr >= min(REGISTER_MSU$jahr)) |>
    mutate(mutterschaftsurlaub = mutterschaftsurlaub_ohne_eomax +
        mutterschaftsurlaub_eomax) |>
    mutate(log_mutterschaftsurlaub = log(mutterschaftsurlaub),
        log_anzahl_geburten = log(anzahl_geburten))
```

Die Anzahl Mutterschaftsurlaube in der Zukunft wird mit der Hilfe der Funktion `impute_lm()` aus dem `impute`-Packet geschätzt. Der Schätzwert basiert auf einem, linearen Modell, das die Variable `log_mutterschaftsurlaub` in Bezug auf `log_annzahl_geburten` und `erwerbsquote` erklärt.

```
MSU_BENEF_CH <- MSU_CH_fc |>
    impute_lm(log_mutterschaftsurlaub ~ log_anzahl_geburten +
        erwerbsquote | kat) |>
    ungroup() |>
    mutate(mutterschaftsurlaub = ifelse(jahr > jahr_register,
        exp(log_mutterschaftsurlaub), mutterschaftsurlaub)) |>
    select(jahr, kat, mutterschaftsurlaub)
```

Der Ausdruck `|kat` bedeutet, dass die Imputation innerhalb der verschiedenen Alterskategorien durchgeführt wird (eine Art von Gruppierung).

Auf den Wert von `log_mutterschaftsuralub` wird die Exponentialfunktion angewendet, um die tatsächliche Anzahl Mutterschaftsurlaube zu erreichen.

Danach gibt es ein auskommentiertes Stück Code, das erlaubt die Statistiken und Ergebnisse der Regression zu sehen:

```
# Fit - uncomment to see R^2 values, coefficients, etc.
library(broom)
library(purrr)

model_summaries <- MSU_CH_fc |>
    filter(!is.na(log_mutterschaftsurlaub), !is.na(log_anzahl_geburten),
        !is.na(erwerbsquote)) |>
```

```
    group_by(kat) |>
    nest() |>
    mutate(model = map(data, ~lm(log_mutterschaftsurlaub ~ log_anzahl_geburten +
        erwerbsquote, data = .x)), glance = map(model, broom::glance),
        tidy = map(model, broom::tidy))

# View R², p-value, etc. per group
model_summaries |>
    select(kat, glance) |>
    unnest(glance) |>
    select(kat, r.squared, adj.r.squared, p.value)
# Coefficients
a <- model_summaries |>
    select(kat, tidy) |>
    unnest(tidy)
print(a, n = 21)
```

Die Ergebnisse sind folgenden:

```
## # A tibble: 7 x 4
## # Groups:   kat [7]
##     kat r.squared adj.r.squared       p.value
##   <dbl>     <dbl>         <dbl>         <dbl>
## 1     1     0.765         0.713 0.00148
## 2     2     0.982         0.978 0.0000000137
## 3     3     0.984         0.981 0.00000000792
## 4     4     0.974         0.968 0.0000000722
## 5     5     0.975         0.969 0.0000000635
## 6     6     0.960         0.951 0.000000503
## 7     7     0.940         0.926 0.00000322
```

```
## # A tibble: 21 x 6
## # Groups:   kat [7]
##      kat term                estimate std.error statistic       p.value
##    <dbl> <chr>                  <dbl>     <dbl>     <dbl>         <dbl>
##  1     1 (Intercept)            1.25     1.85       0.678 0.515
##  2     1 log_anzahl_geburten    1.12     0.337      3.32  0.00899
##  3     1 erwerbsquote          -6.52     6.42      -1.01  0.337
##  4     2 (Intercept)            1.73     0.458      3.78  0.00432
##  5     2 log_anzahl_geburten    1.06     0.0674    15.7   0.0000000765
##  6     2 erwerbsquote          -3.63     0.989     -3.67  0.00517
##  7     3 (Intercept)            0.225    0.525      0.428 0.679
##  8     3 log_anzahl_geburten    0.898    0.0384    23.4   0.00000000226
##  9     3 erwerbsquote           0.495    0.332      1.49  0.171
## 10     4 (Intercept)           -0.478    0.726     -0.658 0.527
## 11     4 log_anzahl_geburten    0.943    0.0760    12.4   0.000000580
## 12     4 erwerbsquote           0.936    0.174      5.39  0.000437
## 13     5 (Intercept)           -3.47     0.758     -4.58  0.00133
## 14     5 log_anzahl_geburten    1.24     0.0900    13.7   0.000000243
## 15     5 erwerbsquote           0.977    0.402      2.43  0.0381
## 16     6 (Intercept)           -4.43     1.23      -3.60  0.00575
## 17     6 log_anzahl_geburten    1.46     0.185      7.87  0.0000253
## 18     6 erwerbsquote           0.211    0.544      0.387 0.708
## 19     7 (Intercept)           -1.99     0.941     -2.12  0.0634
## 20     7 log_anzahl_geburten    1.31     0.143      9.16  0.00000736
```

```
## 21     7 erwerbsquote          -0.374    1.44      -0.259 0.801
```

Wir berechnen die Anzahl der Mutterschaftsurlaube in der Vergangenheit aus der Summe der Urlaube mit und ohne maximaler Entschädigung. Auch der Anteil mit maximaler Entschädigung wird berechnet.

```
# Compute anteil_eomax
MSU_CH_anteil_eomax <- MSU_CH |>
    mutate(mutterschaftsurlaub = mutterschaftsurlaub_eomax +
        mutterschaftsurlaub_ohne_eomax, anteil_eomax =
↪   mutterschaftsurlaub_eomax/mutterschaftsurlaub) |>
    select(jahr, kat, anteil_eomax, mutterschaftsurlaub_eomax,
        mutterschaftsurlaub) |>
    ungroup()
```

Um die Entwicklung des Anteils an Mutterschaftsurlauben mit maximaler Entschädigung vorherzusagen, berechnen wir zunächst die durchschnittliche Wachstumsrate pro Alterskategorie auf Basis der Registerdaten. Dabei unterscheiden wir zwischen den Jahren, in denen der Höchstbetrag angepasst wurde, und den Jahren ohne Anpassung (mithilfe der Variable `dummy_eomax`).

```
# Compute growth rate in anteil EOMAX, as well as a dummy
# variable for EOMAX increase
WR_ANTEIL_EOMAX <- MSU_CH_anteil_eomax |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_anteil = (anteil_eomax - lag(anteil_eomax))/lag(anteil_eomax)) |>
    ungroup() |>
    full_join(EOMAX |>
        select(jahr, dummy_eomax), by = "jahr") |>
    filter(jahr > min(MSU_CH$jahr) & jahr <= max(MSU_CH$jahr)) |>
    select(jahr, kat, wr_anteil, dummy_eomax) |>
    group_by(dummy_eomax, kat) |>
    summarize(wr_anteil = mean(wr_anteil, na.rm = TRUE)) |>
    mutate(wr_anteil = if_else(is.na(wr_anteil) | wr_anteil ==
        -1 | is.infinite(wr_anteil) | kat == 2, 0, wr_anteil))
```

Für die Projektion des Anteils an Mutterschaftsurlauben mit EOmax erstellen wir eine Tabelle mit allen Kombination von Alterskategorien und Jahren, vom ersten Jahr in `MSU_CH_anteil_eomax` bis `jahr_ende`. Dieser Tabelle werden die historischen Daten des EO-Registers (`MSU_CH_anteil_eomax`), historischen und zukünfigten Werte der Dummy-Variable `dummy_eomax` und die errechneten durchschnittliche Wachstumsraten (`wr_anteil`) der Vergangenheit hinzugefügt. Wir nehmen an, dass jährlichen Wachstumsraten pro Alterskategorie in der Zukunft gleich der errechneten durchschnittlichen Wachstumsraten der Vergangenheit sein werden. Sie unterscheiden sich für Perioden mit und ohne Anpassung in `eomax`.

Mithilfe dieser Tabelle (`MSU_ANTEIL_EOMAX_CH`) und der Funktion `accumulate()` aus dem `purrr`-Paket, wird der Wert von `anteil_eomax` fortgeschrieben. Die Funktion `accumulate()` berechnet sukzessive, wie sich `anteil_eomax` von Jahr zu Jahr verändert.

```
  # Calculate the "anteil_eomax" thanks to the accumulation of the yearly growth
  # rate "wr_anteil"
  MSU_ANTEIL_EOMAX_CH <- expand_grid(
      jahr = min(MSU_CH_anteil_eomax$jahr):jahr_ende, # Jahre von der aktuellen Range bis
      ↪  2070
      kat = unique(MSU_CH_anteil_eomax$kat)       # Kategorien beibehalten
  ) |>
      left_join(MSU_CH_anteil_eomax |>
                  select(-mutterschaftsurlaub_eomax))  |>
      left_join(EOMAX |>
```

```
            select(jahr, dummy_eomax)) |>
    left_join(WR_ANTEIL_EOMAX) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(
        anteil_eomax = c(
            anteil_eomax[jahr==PARAM_GLOBAL$jahr_register],
            accumulate(wr_anteil[-1],
                       .init = anteil_eomax[1],
                       ~ .x * (1 + .y))[-1]
        )) |>
    mutate(anteil_eomax=case_when(
        anteil_eomax > 1 ~ 1,
        anteil_eomax < 0 ~ 0,
        TRUE ~ anteil_eomax
    )) |>
    select(jahr, kat, anteil_eomax)
```

- `wr_anteil[-1]`: der ersten Wert wird ignoriert, da dieser bereits in der Initialisierung (`.init = anteil_eomax[1]`) berücksichtigt wird.
- `.init = anteil_eomax[1]`: Der Startwert für die Berechnung wird auf den Wert `anteil_eomax` im Jahr `jahr_abr` gesetzt.
- `~ .x * (1 + .y)`: Dies ist eine Funktion, die angibt, wie der kumulierte Wert berechnet wird. In diesem Fall werden die Werte `anteil_eomax` durch die Multiplikation des vorheirgen Wertes von `anteil_eomax`mit (1 + dem jeweiligen Wachstumsrat des Anteils) projiziert.
- `[-1]`: Nach der Berechnung der kumulativen Werte wird das erste Element entfernt, weil es bereits als Initialwert gesetzt wurde.

Schliesslich wird sichergestellt, dass die Anteile zwischen 0 und 1 liegen.

Wir haben nun alle notwendigen Elemente, um die zukünftige Anzahl Mutterschaftsurlaube mit maximaler Entschädigung (`mutterschaftsurlaub_eomax`) und die Anzahl Mutterschaftsurlaube mit geringerer Entschädigung (`mutterschaftsurlaub_ohne_eomax`) zu berechnen. Die Ergebnisse werden im Dataframe `MSU_BENEF_CH` abgespeichert und an das Modul `mod_eo_msu` zurückgegeben.

```
# Divide the maternity leaves between EOMAX and under EOMAX
MSU_BENEF_CH <- MSU_BENEF_CH |>
    left_join(MSU_ANTEIL_EOMAX_CH, by = c("jahr", "kat")) |>
    mutate(mutterschaftsurlaub_eomax = ifelse(!is.na(anteil_eomax),
        mutterschaftsurlaub * anteil_eomax, 0)) |>
    mutate(mutterschaftsurlaub_ohne_eomax = mutterschaftsurlaub -
        mutterschaftsurlaub_eomax, dom = 1) |>
    select(jahr, kat, dom, mutterschaftsurlaub, mutterschaftsurlaub_eomax,
        mutterschaftsurlaub_ohne_eomax, anteil_eomax)

# Output
# ------------------------------------------------------------------------

return(list(MSU_BENEF_CH = MSU_BENEF_CH))
```

4.8.2.2 Modul mod_eo_msu_benef_au.R *Dokumentation zuletzt aktualisiert am 27.08.2025*

Dieses Modul prognostiziert die Anzahl der Mutterschaftsurlaube für Frauen mit Wohnsitz im Ausland. Die nötigen Argumente sind:

```
mod_eo_msu_benef_au <- function(PARAM_GLOBAL,
                                REGISTER_MSU,
                                FRONTALIERS,
                                MSU_BENEF_CH) {
```

Zuerst werden die Registerdaten gefiltert, sodass nur Frauen mit Wohnsitz im Ausland (dom = 2) übrig bleiben. Anschliessend wird das Grenzgängerszenario gefiltert, um nur die Grenzgängerinen zu behalten.

```
# Filter the foreign maternity leave
MSU_AU <- REGISTER_MSU |>
    filter(dom == 2) |>
    mutate(mutterschaftsurlaub = mutterschaftsurlaub_ohne_eomax +
        mutterschaftsurlaub_eomax, anteil_eomax =
↪   mutterschaftsurlaub_eomax/(mutterschaftsurlaub))

# Filter the boreder worker scenarios by sex
FRONTALIERS_SCEN_F <- FRONTALIERS |>
    filter(sex == "f") |>
    select(-sex)
```

Wir berechnen die Wachstumsrate der Gesamtzahl der Grenzgängerinen pro Alterskategorie im Vergleich zum Jahr `jahr_register`.

```
# Calculation of growth of frontier workers compared to the
# 'Abrechnungsjahr'
FRONTALIERS_WR_SCEN <- FRONTALIERS_SCEN_F |>
    filter(jahr >= jahr_register) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_grenz = (anzahl_grenzganger -
    ↪  first(anzahl_grenzganger))/first(anzahl_grenzganger)) |>
    ungroup()
```

Wir lassen die Anzahl Mutterschaftsurlaube pro Alterskategorie mit dem Wachstum der Grenzgängerinnen wachsen. Wir nehmen als Initialwert den Durchschnitt der Anzahl Mutterschaftsurlaube der letzten 3 Jahren für jede Alterskategorie.

```
# Take the mean of the last three numbers of maternity
# leave among frontier workers as starting point
MSU_AU_mean_reg <- MSU_AU %>%
    group_by(kat) %>%
    arrange(jahr, .by_group = TRUE) %>%
    mutate(mutterschaftsurlaub_reg = mean(tail(mutterschaftsurlaub,
        3), na.rm = TRUE)) %>%
    ungroup()

# Let the mean number of maternity leave grow with the
# frontier workers growth factor
MSU_BENEF_AU <- MSU_AU |>
    full_join(FRONTALIERS_WR_SCEN, by = c("jahr", "kat")) |>
    full_join(MSU_AU_mean_reg |>
        select(kat, mutterschaftsurlaub_reg) |>
        unique(), by = "kat") |>
    mutate(mutterschaftsurlaub = ifelse(jahr > jahr_register,
        mutterschaftsurlaub_reg * (1 + wr_grenz), mutterschaftsurlaub)) |>
    select(jahr, kat, mutterschaftsurlaub)
```

Um die prognostizierte Anzahl Mutterschaftsurlaube im Ausland nach Entschädigungshöhe (`eomax` oder nicht) zu verteilen, lassen wir den `anteil_eomax` im Ausland für jede Alterskategorie ab dem Jahr `jahr_register` mit dem Wachstum dieses Anteils in der Schweiz steigen. Diese Wachstumsrate unterscheidet sich für Perioden mit und ohne Anpassung von `eomax`.

```
# Compute the part of maternity leave abroad which gets the
# maximum indemnity We take only the last year because of
# the EO MAX increase in 2023.
MSU_AU_register <- MSU_AU |>
    filter(jahr == jahr_register) |>
    select(kat, anteil_eomax_reg = anteil_eomax)

# Growth factor of the anteil for mother living in
# Switzerland
ANTEIL_EOMAX <- MSU_BENEF_CH |>
    select(jahr, kat, anteil_eomax) |>
    filter(jahr >= jahr_register) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_anteil_eomax = (anteil_eomax - first(anteil_eomax))/first(anteil_eomax)) |>
    ungroup() |>
    select(jahr, kat, wr_anteil_eomax)

# Let the percent of mother leaving aborad getting EOMAX
# growing like the percent of mother leaving in CH getting
# EOMAX
MSU_AU_anteil_eomax <- ANTEIL_EOMAX |>
    full_join(MSU_AU_register, by = "kat") |>
    group_by(kat, jahr) |>
    mutate(anteil_eomax = anteil_eomax_reg * (1 + wr_anteil_eomax)) |>
    ungroup() |>
    select(jahr, kat, anteil_eomax) |>
    filter(jahr > jahr_register)
```

Man kann nun die Anzahl der Mutterschafturlaube mit Domizil im Ausland berechnen, die eine maximale Entschädigung und eine Entschädigung unterhalb dieser erhalten. Die Kategorien 1 wird ausgeschlossen, da es 2024 keinen Mutterschaftsurlaub für diese Alterskategorien mit Domizil Ausland gab.

```
# Separate the total number of maternity leaves between the
# one getting the max indemnity and the one getting a
# smaller indemnity in the future
MSU_BENEF_AU_fc <- MSU_BENEF_AU |>
    full_join(MSU_AU_anteil_eomax, by = c("kat", "jahr")) |>
    group_by(kat, jahr) |>
    mutate(mutterschaftsurlaub_eomax = ifelse(!is.na(anteil_eomax) &
        kat %in% c(2:7), mutterschaftsurlaub * anteil_eomax,
        0)) |>
    mutate(mutterschaftsurlaub_ohne_eomax = mutterschaftsurlaub -
        mutterschaftsurlaub_eomax) |>
    ungroup() |>
    select(jahr, kat, mutterschaftsurlaub, mutterschaftsurlaub_eomax,
        mutterschaftsurlaub_ohne_eomax, anteil_eomax) |>
    filter(jahr > max(MSU_AU$jahr))
```

Schliesslich werden die historischen Daten aus dem Register und die Projektionen in einem Datensatz zusammengefügt und an das Modul `mod_eo_msu` zurückgegeben.

```
# Join the forecast and the past observations
MSU_BENEF_AU <- MSU_AU |>
    select(jahr, kat, mutterschaftsurlaub, mutterschaftsurlaub_eomax,
    ↪  mutterschaftsurlaub_ohne_eomax,
        anteil_eomax) |>
    bind_rows(MSU_BENEF_AU_fc) |>
    mutate(dom = 2)

# Output
# ------------------------------------------------------------------------

return(list(MSU_BENEF_AU = MSU_BENEF_AU))
```

4.8.2.3 Modul mod_eo_msu_taggeld.R *Dokumentation zuletzt aktualisiert am 27.08.2025*

Dieses Modul dient der Projektion der durchschnittlichen Taggelder von Müttern, die nicht die maximale Entschädigung erhalten. Es verwendet die folgende Argumente:

```
mod_eo_msu_taggeld <- function(PARAM_GLOBAL,
                               REGISTER_MSU,
                               EOMAX,
                               ECKWERTE_EXTENDED)  {
```

Zuerst werden die benötigten Registerdaten ausgewählt. Sie unterscheiden die Höhe der Taggelder nach Jahr, Alterskategorie und Domizil (Schweiz oder Ausland).

```
# Prepare TAGGELD data set by selecting required columns
MSU_TAGGELD <- REGISTER_MSU |>
    select(jahr, kat, dom, taggeld, mutterschaftsurlaub_ohne_eomax,
        mutterschaftsurlaub_eomax)
```

Mit den Daten der, in der Schweiz wohnhaften Mütter wird eine durchschnittliche jährliche Wachstumsrate der Tagegelder über alle Altersgruppen berechnet, gewichtet mit der Anzahl der Leistungsempfänger pro Altersgruppe, die nicht das maximale Tagegeld erhalten.

```
# Calculate weighted growth rate of 'taggeld' for MSU_CH
# data
WR_TAGGELD_TOT <- MSU_TAGGELD |>
    filter(dom == 1) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_taggeld = (taggeld - lag(taggeld))/lag(taggeld)) |>
    ungroup() |>
    group_by(jahr) |>
    summarise(weigthed_wr_taggeld = sum(wr_taggeld * mutterschaftsurlaub_ohne_eomax,
        na.rm = TRUE)/sum(mutterschaftsurlaub_ohne_eomax, na.rm = TRUE))
```

Die durchschnittlichen Wachstumsraten werden zu einer Tabelle hinzugefügt, die jede Kombination der Jahre vom Anfangsjahr des Registers (`min(REGISTER_MSU$jahr)`) bis `jahr_ende` sowie jeder Alterskategorie (`unique(REGISTER_MSU$kat)`) enthält. Zusätzlich wird der Tabelle eine Spalte mit den jährlichen historischen und prognostizierten Wachstumsraten des Schweizer Lohnindexes (`wr_li` aus der Tabelle `ECKWERTE_EXTENDED`) und eine Spalte mit der Variable `dummy_eomax`, die die Anpassungsjahre der Maximalentschädigung abbildet, hinzugefügt.

```r
# Merge growth rates for 'WR' dataset including lohnindex,
# weighted employment, and taggeld growth
WR <- expand_grid(jahr = min(REGISTER_MSU$jahr):jahr_ende, kat =
↪  unique(REGISTER_MSU$kat)) |>
    full_join(ECKWERTE_EXTENDED |>
        filter(jahr >= min(MSU_TAGGELD$jahr)) |>
        select(jahr, wr_li = lohn), by = "jahr") |>
    full_join(WR_TAGGELD_TOT |>
        select(jahr, weigthed_wr_taggeld)) |>
    full_join(EOMAX |>
        select(jahr, dummy_eomax)) |>
    filter(jahr >= min(MSU_TAGGELD$jahr)) |>
    mutate(wr_li = wr_li/100)  # Since lohn wachstum is in percent in ECKWERTE_EXTENDED
```

Wir ignorieren das Jahr 2021, für das der Schweizer Lohnindex problematisch ist, bevor wir eine lineare Regression schätzen, die die Wachstumsrate der Tagegelder in Abhängigkeit von der Wachstumsrate des SLI (`wr_li`) und `dummy_eomax` modelliert.

```r
WR_lm <- WR |>
    filter(jahr != 2021)

model <- lm(weigthed_wr_taggeld ~ wr_li + dummy_eomax, data = WR_lm)
```

Mit dem folgenden Befehl, kann man sich die Ergebnisse der Regression anschauen:

```r
summary(model)
```

```
##
## Call:
## lm(formula = weigthed_wr_taggeld ~ wr_li + dummy_eomax, data = WR_lm)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.007179 -0.003471  0.000000  0.002862  0.007947
##
## Coefficients:
##               Estimate Std. Error t value             Pr(>|t|)
## (Intercept) -0.0007764  0.0010791  -0.719                0.474
## wr_li        1.0884571  0.1227255   8.869    0.000000000000294 ***
## dummy_eomax  0.0605714  0.0019256  31.456 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.003954 on 74 degrees of freedom
##   (357 osservazioni eliminate a causa di valori mancanti)
## Multiple R-squared:  0.966,  Adjusted R-squared:  0.9651
## F-statistic:  1050 on 2 and 74 DF,  p-value: < 0.00000000000000022
```

Der Koeffizient für den Lohnindex (`wr_li`) beträgt 0.972 und ist hochsignifikant ($p < 0.001$). -> Das bedeutet, dass sich die Taggelder nahezu im gleichen Verhältnis wie die Löhne entwickeln. Ein Anstieg des Lohnindexes um 1 % geht im Durchschnitt mit einem Anstieg der Taggelder um rund 0,97 % einher.

Dieser Wert liegt leicht unter 1, was – wie du richtig anmerkst – ökonomisch plausibel ist: Da die Taggelder gedeckelt sind, steigen sie tendenziell langsamer als die Löhne, insbesondere in den oberen Einkommenssegmenten.

Der Koeffizient der Dummy-Variable (`dummy_eomax`) beträgt 0.0618 und ist ebenfalls hochsignifikant. -> In den Perioden, in denen dieser Dummy aktiv ist (wegen einer Anpassung der maximalen Tagesentschädigung), liegen die Taggelder im Durchschnitt um rund 6,2 Prozentpunkte höher als sonst.

Der Intercept (Achsenabschnitt) ist mit -0.000026 insignifikant ($p \approx 0.99$) und kann vernachlässigt werden. Das deutet darauf hin, dass ohne Lohnveränderung und ohne Sondereffekt keine systematische Veränderung der Taggelder vorliegt.

Das Bestimmtheitsmaß (R^2 = 0.966) zeigt eine sehr hohe Erklärungskraft: rund 97 % der Variation der gewichteten Taggelder werden durch das Modell erklärt.

Die Residuen sind klein und symmetrisch verteilt, was auf eine gute Modellpassung hindeutet.

Basierend auf diesem Modell (`model`), wird die Vorhersage des durchschnittlichen Wachstums der Taggelder (`wr_taggeld`) für die Jahre nach `jahr_register` mit dem `predict()`Befehl generiert. `newdata = pic(everything())` sorgt dafür, dass alle Spalten aus den gefilterten Daten (`WR_lm`) als Eingabewerte verwendet werden. Schliesslich wird die Funktion `bind_rows(data_last_year_register)` verwendet, um die Vorhersagen (`predict_data`) mit den letzten bekannten Daten (`data_last_year_register`) zu kombinieren.

```
# Predict and combine the last known data point with
# predictions
data_last_year_register <- WR_lm |>
    filter(jahr == jahr_register)

predict_data <- WR_lm |>
    filter(jahr > jahr_register) |>
    mutate(weigthed_wr_taggeld = predict(model, newdata = pick(everything()))) |>
    bind_rows(data_last_year_register)
```

Die Prognosen für das durchschnittliche Wachstum der Tagegelder werden für jedes Jahr mit den beiden möglichen Domizil-Ländern (1 = Schweiz, 2 = Ausland) kombiniert.

```
# add domicile to predict_data
DOM <- expand.grid(jahr = (jahr_register + 1):jahr_ende, dom = unique(MSU_TAGGELD$dom))

predict_data_dom <- predict_data |>
    full_join(DOM, by = "jahr") |>
    mutate(dom = ifelse(is.na(dom), 1, dom))
```

Für das Jahr `jahr_register` wird der NA-Wert von `dom` auf 1 gesetzt, da die Wachstumsraten auf den Daten der in der Schweiz ansässigen Mütter basieren.

Wir verwenden für jede Alterskategorie und jedes Domzil die durchschnittlichen Taggelder des Jahres `jahr_register` als Ausgangspunkt und schreiben diese anhand der prognostizierten durchschnittlichen Wachstumsrate (`weigthed_wr_taggeld`) fort. Die Daten werden dafür nach `kat_alt` (Alterskategorie) und `dom` (Domizil) gruppiert und innerhalb jeder Gruppe aufsteigend sortiert.Für die Fortschreibung benutzen wir die `accumulate()`-Funktion:

- Eine neue Spalte taggeld wird erstellt, die die akkumulierte Entschädigung enthält:
- `taggeld[1]`: Der erste Wert der Entschädigung in jeder Gruppe wird als Ausgangspunkt genommen.
- `accumulate(weigthed_wr_taggeld[-1], .init = taggeld[1], ~ .x * (1 + .y))`: Die Funktion `accumulate` berechnet die kumulierten Taggelder:
  - `weigthed_wr_taggeld[-1]`: Alle Werte der Wachstumsrate wr_taggeld, ausser dem Ersten.
  - `.init = taggeld[1]`: Der Startwert für die Akkumulation ist der erste Wert von taggeld.

- `~ .x * (1 + .y)`: Die Akkumulation wird berechnet, indem jeder Wert mit der vorherigen Entschädigung multipliziert und die Wachstumsrate `(.y)` berücksichtigt wird.
- `[-1]`: Das erste Element der Akkumulation (der Initialwert) wird entfernt, um die Akkumulation ab dem zweiten Jahr zu starten.

```
# Calculate 'taggeld' accumulation for MSU_TAGGELD_CH
MSU_TAGGELD_pred <- predict_data_dom |>
    filter(jahr >= jahr_register) |>
    full_join(MSU_TAGGELD |>
        select(jahr, kat, dom, taggeld) |>
        filter(jahr == jahr_register), by = c("kat", "jahr",
        "dom")) |>
    group_by(kat, dom) |>
    arrange(jahr) |>
    mutate(taggeld = c(taggeld[1], accumulate(weigthed_wr_taggeld[-1],
        .init = taggeld[1], ~.x * (1 + .y))[-1])) |>
    select(jahr, kat, dom, taggeld)
```

Die berechneten kumulierten Taggelder werden mit den historischen Taggeldern aus dem `MSU_TAGGELD`-Datensatz zusammengeführt und an das Modul `mod_eo_msu` zurückgegeben.

```
# Bind the projections with the historical data
MSU_TAGGELD <- MSU_TAGGELD_pred |>
    filter(jahr > last(MSU_TAGGELD$jahr)) |>
    full_join(MSU_TAGGELD |>
        select(jahr, kat, dom, taggeld)) |>
    arrange(jahr)


# Output
# ------------------------------------------------------------------------
return(list(MSU_TAGGELD = MSU_TAGGELD))
```

4.8.3 Modul mod_eo_vsu.R

Dokumentation zuletzt aktualisiert am 28.08.2025

Dieses Modul berechnet die Ausgaben im Zusammenhang mit dem Urlaub des anderen Elternteils. Es benötigt die folgende Argumente:

```
mod_eo_vsu <- function(PARAM_GLOBAL,
                       REGISTER_VSU,
                       LEBENDGEBURTEN_MANN,
                       BEV_SCENARIO,
                       FRONTALIERS,
                       EOMAX,
                       ECKWERTE_EXTENDED,
                       EO_ABRECHNUNG_DEF,
                       EO_AUSGABEN_ANTEILE,
                       STRUKTURFAKTOR
                       ) {
```

Diese sind im Kapitel 4.8 beschrieben.

Das Modul besteht aus verschiedenen Untermodulen, welche die Anzahl der in der Schweiz und im Ausland wohnhaften Leistungsempfänger sowie die durchschnittliche Höhe der Entschädigung für Eltern, die nicht

eomaxerhalten, schätzen.

Zuerst wird das Modul, das die Anzahl Urlaube des anderen Elternteils mit Domizil in der Schweiz prognostiziert aufgerufen. Es benötigt die folgenden Inputs.

```
# Number of paternity leaves for Swiss residents
tl_eo_vsu_benef_ch <- mod_eo_vsu_benef_ch(PARAM_GLOBAL, REGISTER_VSU,
    LEBENDGEBURTEN_MANN, BEV_SCENARIO, EOMAX, ECKWERTE_EXTENDED,
    STRUKTURFAKTOR)
```

Die Ergebnisse dieses Modules werden im Datensatz VSU_BENEF_CH gespeichert.

```
VSU_BENEF_CH <- tl_eo_vsu_benef_ch$VSU_BENEF_CH
```

Das Modul, das die Anzahl der Urlaube des anderen Elternteils mit Domizil im Ausland prognostiziert, wird anschliessend aufgerufen und die Ergebnisse im Datensatz VSU_BENEF_AU gespeichert.

```
# Number of paternity leaves for non-Swiss residents
tl_eo_vsu_benef_au <- mod_eo_vsu_benef_au(PARAM_GLOBAL, REGISTER_VSU,
    FRONTALIERS, VSU_BENEF_CH)

VSU_BENEF_AU <- tl_eo_vsu_benef_au$VSU_BENEF_AU
```

Die Ergebnisse dieser beiden Module werden gemeinsam in einem Datensatz gespeichert, der VSU_BENEF genannt wird.

```
# Number of paternity leaves
VSU_BENEF <- VSU_BENEF_CH |>
    full_join(VSU_BENEF_AU)
```

Anschliessend wird das Untermodul abgerufen, das die Taggelder für Eltern berechnet, die nicht eomax erhalten. Die Ergebnisse sind im Datensatz VSU_TAGGELD gespeichert.

```
# Daily indemnity for those not getting the maximal
# benefits
tl_eo_vsu_taggeld <- mod_eo_vsu_taggeld(PARAM_GLOBAL, REGISTER_VSU,
    EOMAX, ECKWERTE_EXTENDED, STRUKTURFAKTOR)

VSU_TAGGELD <- tl_eo_vsu_taggeld$VSU_TAGGELD
```

Bevor wir die Kosten für den Urlaub des anderen Elternteils berechnen, setzen wir den Anteil der Eltern, die eomax erhalten auf 1, falls das geschätzte durchschnittliche Taggeld pro Alterskategorie, Domizil und Jahr grösser gleich dem maximalen Taggeld ist. Für jede Alterskategorie, jedes Domizil und Jahr, werden die Anzahl Urlaube mit eomax und einer niedrigeren Entschädigung errechnet. Die Ausgaben werden dann zunächst pro Alterskategorie, Domizil, Taggeldkategorie (eomaxoder nicht) und Jahr berechnet, indem die Anzahl der Beziehenden mit dem durchschnittlichen Taggeld (oder eomax) und 14 Tagen multipliziert wird. Die Ergebnisse werden anschliessend pro Jahr aufsummiert, um die jährlichen Ausgaben zu ermitteln.

```
# Paternity leaves expenses
VSU_AUSGABEN <- VSU_BENEF |>
    left_join(VSU_TAGGELD, by = c("jahr", "kat", "dom")) |>
    full_join(EOMAX |>
        select(jahr, eomax), by = "jahr") |>
    drop_na() |>
    group_by(jahr, kat, dom) |>
    mutate(anteil_eomax = if_else(taggeld >= eomax, 1, anteil_eomax),
        vaterschaftsurlaub_eomax = vaterschaftsurlaub * anteil_eomax,
        vaterschaftsurlaub_ohne_eomax = vaterschaftsurlaub *
```

```
            (1 - anteil_eomax), tot_vsu_aus = 14 * vaterschaftsurlaub_ohne_eomax *
            taggeld + 14 * vaterschaftsurlaub_eomax * eomax) |>
    group_by(jahr) |>
    summarize(aus_tot_vsu = sum(tot_vsu_aus, na.rm = TRUE))
```

Aufwendungen für Abschreibungen von Rückerstattungsforderungen, die mit den Ausgaben für Urlaube des anderen Elternteils verbunden sind, werden anteilmässig aus der totalen Abschreibungen von Rückerstattungsforderungen von der Buchhaltung errechnet.

```
# Abschreibung von Rueckerstattungsforderungen
# (anteilmaessig)
ABSCHREIB_RUECKF <- EO_ABRECHNUNG_DEF |>
    select(jahr, abschr_rueckf) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_vsu), by = "jahr") |>
    mutate(abschreib_rueckf = anteil_vsu * abschr_rueckf) |>
    select(jahr, abschreib_rueckf)
```

Gleichermassen werden die Ausgaben aus dem Beitragsanteil zu Lasten der EO dem Urlaub des anderen Elternteils zugeteilt.

```
# Beitragsanteil zu Lasten der EO (anteilmaessig)
BTR_ANT_EO <- EO_ABRECHNUNG_DEF |>
    select(jahr, btr_ant_eo) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_vsu), by = "jahr") |>
    mutate(btr_ant_eo = anteil_vsu * btr_ant_eo) |>
    select(jahr, btr_ant_eo)
```

Wir nehmen an, dass der Anteil dieser Kosten im Vergleich zu den anderen Ausgaben für den Urlaub des anderen Elternteils konstant bleibt und schreiben sie mithilfe der Projektionen der anderen Ausgaben weiter. Die Totalen Ausgaben für den Urlaub des anderen Elternteils werden als Summe der errechneten Kosten `aus_tot_vsu`, der Abschreibungen von Rückerstattungsforderungen und des Beitragsanteils zulasten der EO errechnet.

```
# Add Abschreibung und Beitragsanteil zu den VSU-Ausgaben
VSU_AUSGABEN <- VSU_AUSGABEN |>
    full_join(ABSCHREIB_RUECKF, by = "jahr") |>
    full_join(BTR_ANT_EO, by = "jahr") |>
    mutate(anteil_abschreib = abschreib_rueckf/aus_tot_vsu, anteil_btr =
    ↪  btr_ant_eo/aus_tot_vsu) |>
    fill(anteil_abschreib, anteil_btr, .direction = "down") |>
    mutate(btr_ant_eo = anteil_btr * aus_tot_vsu, abschreib_rueckf = anteil_abschreib *
        aus_tot_vsu, aus_tot_vsu = aus_tot_vsu + btr_ant_eo +
        abschreib_rueckf) |>
    full_join(EO_ABRECHNUNG_DEF |>
        select(jahr, geld_leist), by = "jahr") |>
    full_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_vsu), by = "jahr") |>
    mutate(aus_tot_vsu = if_else(jahr <= jahr_abr, geld_leist *
        anteil_vsu, aus_tot_vsu)) |>
    select(jahr, aus_tot_vsu)
```

Schliesslich werden die Ausgaben bis zum Jahr `jahr_abr` so angepasst, dass die Gesamtsumme mit den

Buchhaltungsdaten übereinstimmt. Etwaige Abweichungen kommen z.B. bei Leistungen für die Elternschaft davon, dass sich der Beginn des Leistungsanspruchs (Geburt eines Kindes) nicht unbedingt mit dem Abrechnungszeitpunkt übereinstimmt.

Somit ist die Berechnung der Ausgaben des Urlaubs füd den anderen Elternteil abgeschlossen und die Dataframes `VSU_AUSGABEN`, `VSU_BENEF` und `VSU_TAGGELD` können an das Modul `mod_eo_ausgaben` zurückgegebn werden:

```
#---Output--------------------------------------------------------------------

return(list(VSU_AUSGABEN = VSU_AUSGABEN, VSU_BENEF = VSU_BENEF,
    VSU_TAGGELD = VSU_TAGGELD))
```

4.8.3.1 Modul mod_eo_vsu_benef_ch.R *Dokumentation zuletzt aktualisiert am 28.08.2025*

Dieses Modul schätzt die Anzahl der Urlaube des anderen Elternteils mit Wohnsitz in der Schweiz. Die benötigen Argumente sind:

```
mod_eo_vsu_benef_ch <- function(PARAM_GLOBAL,
                                REGISTER_VSU,
                                LEBENDGEBURTEN_MANN,
                                BEV_SCENARIO,
                                EOMAX,
                                ECKWERTE_EXTENDED,
                                STRUKTURFAKTOR
                                ) {
```

Zuerst werden die Registerdaten gefiltert, sodass nur Eltern mit Wohnsitz in der Schweiz (`dom` = 1) im Datensatz verbleiben. Die Summe der Urlaube des anderen Elternteils werden als Summe der Urlaube mit maximalen und darunter liegenden Taggeldern berechnet. Darüber hinaus wird der Anteil Urlaube berechnet, welcher die maximale Entschädigung bekommt (`anteil_eomax`) .

```
# filter for the Swiss paternity leave
VSU_CH <- REGISTER_VSU |>
    filter(dom == 1) |>
    mutate(vaterschaftsurlaub = vaterschaftsurlaub_ohne_eomax +
        vaterschaftsurlaub_eomax, anteil_eomax =
↪   vaterschaftsurlaub_eomax/vaterschaftsurlaub)
```

Anschliessend werden die historische Anzahl Geburten pro Jahr agreggiert. Wir brauchen die Alterskategorie nicht, da die BFS Szenarien keine Information über des Vatersalters enthalten.

```
# Number of historical births per year (Age categorie not
# relevant, as we don't have the father's age in the
# BEV_SCENARIO)
ANZAHL_GEBURTEN <- LEBENDGEBURTEN_MANN |>
    group_by(jahr) |>
    summarise(anzahl_geburten = sum(anzahl_geburten, na.rm = TRUE),
        .groups = "drop")
```

Um die Szenarien mit den historischen Geburten zu verknüpfen, verwenden wir für das Jahr nach der letzten bekannten Anzahl an Geburten (`max(LEBENDGEBURTEN_MANN$jahr) + 1`) den Durchschnitt der letzten drei Jahre. Dieses zusätzliche Jahr wird mit den historischen Geburten im Dataframe `ANZAHL_GEBURTEN` zusammengeführt.

```
# Mean of the number of birth of the last three years as
# starting point
```

```
STARTING_POINT <- ANZAHL_GEBURTEN |>
    arrange(jahr) |>
    mutate(anzahl_geburten = mean(tail(anzahl_geburten, 3))) |>
    mutate(jahr = max(LEBENDGEBURTEN_MANN$jahr) + 1) |>
    unique()

ANZAHL_GEBURTEN <- ANZAHL_GEBURTEN |>
    full_join(STARTING_POINT)
```

Anschließend werden die Wachstumsrate aus dem BFS-Szenario berechnet. Die Wachstumsraten werden mit den Geburten pro Alterskategorie des anderen Elternteils kombiniert.

```
# Compute future growth rate of births from OFS Scenarios
LEBENDGEBURTEN_SZENARIO_WR <- BEV_SCENARIO |>
    filter(scenario == PARAM_GLOBAL$bev_scenario) |>
    group_by(jahr) |>
    summarize(anzahl_geburten = sum(geburt, na.rm = TRUE)) |>
    filter(jahr >= max(ANZAHL_GEBURTEN$jahr)) |>
    arrange(jahr) |>
    mutate(wr_geburten = anzahl_geburten/first(anzahl_geburten) -
        1) |>
    select(jahr, wr_geburten)

WR_ANZAHL_GEBURTEN <- ANZAHL_GEBURTEN |>
    full_join(LEBENDGEBURTEN_SZENARIO_WR) |>
    mutate(anzahl_geburten_start = anzahl_geburten[jahr == max(ANZAHL_GEBURTEN$jahr)],
        anzahl_geburten = if_else(jahr <= max(ANZAHL_GEBURTEN$jahr),
            anzahl_geburten, anzahl_geburten_start * (1 + wr_geburten))) |>
    select(jahr, anzahl_geburten) |>
    mutate(anzahl_geburten_start = anzahl_geburten[jahr == jahr_register],
        wr_geburten = anzahl_geburten/anzahl_geburten_start -
            1) |>
    filter(jahr >= jahr_register)

# Add age categories
KAT <- expand.grid(jahr = min(WR_ANZAHL_GEBURTEN$jahr):max(WR_ANZAHL_GEBURTEN$jahr),
    kat = unique(VSU_CH$kat))

LEBENDGEBURTEN_SZENARIO_WR_KAT <- WR_ANZAHL_GEBURTEN |>
    full_join(KAT, by = "jahr")
```

Die Anzahl Urlaube des anderen Elternteils entwickelt sich für den Prognosezeitraum ntsprechend der Wachstumsrate der Geburten gemäß dem BFS-Szenario. Startpunkt ist die Anzahl der Urlaube pro Alterskategorie im letzten Jahr, für das Registerdaten verfügbar sind.

```
# Take the last historical data point of paternity leave
# per age category of the other parent
VSU_CH_HIST <- VSU_CH |>
    group_by(kat) |>
    arrange(jahr, .by_group = TRUE) |>
    summarise(vaterschaftsurlaub_register = tail(vaterschaftsurlaub,
        1))

# Let it grow like the scenarios
VSU_CH_fc <- LEBENDGEBURTEN_SZENARIO_WR_KAT |>
```

```
    filter(jahr > jahr_register) |>
    select(jahr, wr_geburten, kat) |>
    full_join(VSU_CH_HIST) |>
    mutate(vaterschaftsurlaub = vaterschaftsurlaub_register *
        (1 + wr_geburten)) |>
    select(jahr, kat, vaterschaftsurlaub) |>
    mutate(dom = 1)
```

Die historische Anzahl der Urlaube des anderen Elternteils wird mit den Prognosen kombiniert, um die vollständige Zeitreihe zu erstellen.

```
# Join forecast and historical paternity leave by other
# parents' age category
VSU_BENEF_CH <- VSU_CH |>
    full_join(VSU_CH_fc, by = c("jahr", "kat", "vaterschaftsurlaub")) |>
    arrange(jahr) |>
    select(jahr, kat, vaterschaftsurlaub)
```

Um die Urlaube des anderen Elternteils auf die Empfänger der Maximalentschädigung und die anderen aufteilen zu können, wird der Anteil der Eltern, die eine Maximalentschädigung erhalten geschätzt. Dazu berechnen wir für die Vergangenheit zunächst eine durchschnittliche jährliche Wachstumsrate des Anteils der Empfänger, die eine Maximalentschädigung erhalten, gewichtet mit der Anzahl der Empfänger pro Altersgruppe. Wir verwenden keine Daten der Altersgruppe 1, da im Jahr 2024 kein Empfänger dieser Kategorie die maximale Entschädigung erhalten hat.

```
  # Modelisation of the part of paternity leaves, which get EOMAX
  # Compute the weighted mean of the annual growth of the part of father getting
  # EOMAX
  VSU_CH_anteil_eomax <- VSU_CH |>
    mutate(anteil_eomax = vaterschaftsurlaub_eomax  / (vaterschaftsurlaub )) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_anteil_eomax =
             (anteil_eomax - lag(anteil_eomax)) / lag(anteil_eomax)) |>
    ungroup() |>
    filter(kat !=1 ) |> # kat 1 has year(s) without any father by eomax, which is then
    ↪  problematic when computing the mean.
    group_by(jahr) |>
    summarise(weighted_wr_anteil =
                sum(wr_anteil_eomax * vaterschaftsurlaub, na.rm = TRUE) /
                sum(vaterschaftsurlaub, na.rm = TRUE))
```

Die durchschnittliche Lohnwachstumsrate wird anhand der Entwicklung der Schweizer Löhne und des Strukturfaktors berechnet.

```
# Compute the yearly growth of wages and structure
strukturfaktor <- STRUKTURFAKTOR %>%
    filter(sex == "m") %>%
    pull(strukturfaktor)

WR <- ECKWERTE_EXTENDED |>
    mutate(wr_sli_struktur = (lohn + strukturfaktor + lohn *
        strukturfaktor)/100) |>
    select(jahr, wr_sli_struktur)
```

Anhand der Daten aus dem Jahr 2023 (Jahr, in dem der Höchstbetrag der Entschädigung als letztes angepasst

wurde) berechnen wir welcher Teil der Wachstumsrate in diesem Jahr nicht durch die Lohnentwicklung erklärt werden kann. Dieser Teil wird als `effekt_eomax` der EOmax-Erhöhung zugesprochen.

```
# Compute the effekt of EOMAX with the data from 2023
EFFEKT_EOMAX <- VSU_CH_anteil_eomax |>
    select(jahr, weighted_wr_anteil) |>
    filter(jahr == 2023) |>
    full_join(WR |>
        filter(jahr == 2023), by = "jahr") |>
    mutate(effekt_eomax = weighted_wr_anteil - wr_sli_struktur)

effekt_eomax <- EFFEKT_EOMAX$effekt_eomax
```

Für den Prognosezeitraum berechnen wir das Wachstum im Anteil der Eltern mit maximaler Entschädigung aus der Summe des Lohnwachstums (`wr_sli_struktur`) und der Auswirkungen der Anpassung der Höchstbeträge, für die Jahre, für die eine solche Anpassung prognostiziert wurde (`dummy_eomax * effekt_eomax`).

```
# Prepare the data needing to compute the prediction of
# anteil_eomax
WR <- WR |>
    right_join(EOMAX |>
        select(jahr, dummy_eomax), by = "jahr") |>
    mutate(wr_anteil_eomax = wr_sli_struktur + dummy_eomax *
        effekt_eomax) |>
    filter(jahr >= jahr_register) |>
    select(jahr, wr_anteil_eomax) |>
    crossing(kat = 1:7)
```

Bevor wir den Anteil an Eltern mit maximaler Entschädigung fortschreiben, berechnen wir zunächst den Anteil der Eltern, die im Jahr `jarh_register` die maximale Entschädigung erhalten, pro Altersgruppe. Diese Werte gelten als Ausgangspunkte.

```
VSU_CH_register <- VSU_CH |>
    mutate(anteil_eomax = vaterschaftsurlaub_eomax/(vaterschaftsurlaub_eomax +
        vaterschaftsurlaub_ohne_eomax)) |>
    filter(jahr == jahr_register) |>
    select(jahr, kat, anteil_eomax)
```

Mithilfe dieser Ausgangspunkte (`VSU_CH_register`) und der Funktion `accumulate()` aus dem `purrr`-Paket, wird der Wert von `anteil_eomax` fortgeschrieben. Die Funktion `accumulate()` berechnet sukzessive, wie sich `anteil_eomax` von Jahr zu Jahr verändert.

```
# Calculate the 'anteil_eomax' thanks to the accumulation
# of the yearly growth rate wr_anteil_eomax
VSU_ANTEIL_EOMAX <- WR |>
    filter(jahr > jahr_register) |>
    full_join(VSU_CH_register |>
        select(jahr, kat, anteil_eomax), by = c("jahr", "kat")) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(anteil_eomax = c(anteil_eomax[1], accumulate(wr_anteil_eomax[-1],
        .init = anteil_eomax[1], ~.x * (1 + .y))[-1])) |>
    select(jahr, kat, anteil_eomax) |>
    filter(jahr > jahr_register)
```

- `wr_anteil_eomax[-1]`: der erste Wert wird ignoriert, da dieser bereits in der Initialisierung (`.init = anteil_eomax[1]`) berücksichtigt wird.

- `.init = anteil_eomax[1]`: Der Startwert für die Berechnung wird auf den ersten Wert `anteil_eomax` gesetzt.
- `~ .x * (1 + .y)`: Dies ist eine anonyme Funktion, die angibt, wie der kumulierte Wert berechnet wird. In diesem Fall werden die Werte `anteil_eomax` durch die Multiplikation des vorherigen Wertes von `anteil_eomax`mit (1 + dem jeweiligen Wachstumsrate des Anteils) projiziert.
- `[-1]`: Nach der Berechnung der kumulativen Werte wird das erste Element entfernt, weil es bereits als Initialwert gesetzt wurde.

Wir haben nun alle notwendigen Elemente, um die Anzahl Urlaube des anderen Elternteils mit maximaler Entschädigung (`vaterschaftsurlaub_eomax`) und die Anzahl Urlaube des anderen Elternteils mit geringerer Entschädigung (`vaterschaftsurlaub_ohne_eomax`) zu prognostizieren und mit den historischen Daten aus dem Register in einem Datensatz (`VSU_BENEF_CH`) zu speichern.

```
# Compute thanks to the anteil_eomax predicted above the
# number of paternity leave getting EOMAX and the one
# getting below.
VSU_BENEF_CH_fc <- VSU_BENEF_CH |>
    left_join(VSU_ANTEIL_EOMAX, by = c("kat", "jahr")) |>
    mutate(vaterschaftsurlaub_eomax = vaterschaftsurlaub * anteil_eomax,
        vaterschaftsurlaub_ohne_eomax = vaterschaftsurlaub -
            vaterschaftsurlaub_eomax, dom = 1) |>
    select(dom, jahr, kat, vaterschaftsurlaub, vaterschaftsurlaub_eomax,
        vaterschaftsurlaub_ohne_eomax, anteil_eomax)

VSU_BENEF_CH <- VSU_CH |>
    select(dom, jahr, kat, vaterschaftsurlaub, vaterschaftsurlaub_eomax,
        vaterschaftsurlaub_ohne_eomax, anteil_eomax) |>
    full_join(VSU_BENEF_CH_fc |>
        filter(jahr > jahr_register), by = c("dom", "jahr", "kat",
        "vaterschaftsurlaub", "vaterschaftsurlaub_eomax",
        ↪  "vaterschaftsurlaub_ohne_eomax",
        "anteil_eomax"))
```

Der Datensatz `VSU_BENEF_CH` wird schlussendlich noch an das Modul `mod_eo_vsu` zurückgegeben.

```
# Output
# ------------------------------------------------------------------------
return(list(VSU_BENEF_CH = VSU_BENEF_CH))
```

4.8.3.2 Modul mod_eo_vsu_benef_au.R *Dokumentation aktualisiert am 28.08.2025*

Dieses Modul prognostiziert die Anzahl der Urlaube des anderen Elternteils für Eltern mit Wohnsitz im Ausland. Die nötigen Argumente sind:

```
mod_eo_vsu_benef_au <- function(PARAM_GLOBAL,
                                REGISTER_VSU,
                                FRONTALIERS,
                                VSU_BENEF_CH) {
```

Zuerst werden die Registerdaten auf Eltern mit Wohnsitz im Ausland (dom = 2) eingegrenzt. Es wird die Summe aller Urlaube, (unabhängig von der Höhe der Entschädigung), benannt `vaterschaftsurlaub`, sowie der Anteil Eltern, die die maximale Entschädigung erhalten (`anteil_eomax`) berechnet.

```
# Filter the register to get only the paternity leave of
# the frontiers workers into account:
VSU_AU <- REGISTER_VSU |>
    filter(dom == 2) |>
```

```
    select(dom, jahr, kat, vaterschaftsurlaub_eomax, vaterschaftsurlaub_ohne_eomax) |>
    mutate(vaterschaftsurlaub = vaterschaftsurlaub_ohne_eomax +
        vaterschaftsurlaub_eomax, anteil_eomax =
↪   vaterschaftsurlaub_eomax/vaterschaftsurlaub)
```

Anschliessend wird das Grenzgängerszenario gefiltert, um nur männliche Grenzgänger für die Jahre >= `jahr_register` zu behalten.

```
# Filter the frontier worker scenario to get only the men
FRONTALIERS_SCEN_M <- FRONTALIERS |>
    filter(sex == "m" & jahr >= jahr_register) |>
    select(-sex)
```

Es wird eine Wachstumsrate der Anzahl Grenzgänger im Vergleich zu Jahr `jahr_register` berechnet.

```
# Calculation of growth of frontier workers compared to the
# 'Registersjahr'
FRONTALIERS_WR_SCEN <- FRONTALIERS_SCEN_M |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_grenz = (anzahl_grenzganger/first(anzahl_grenzganger) -
        1)) |>
    ungroup()
```

Für die Prognose der zukünftigen Anzahl Urlaube von Eltern mit Domizil Ausland, nehmen wir den Durchschnitt der Anzahl Urlaube des anderen Elternteils pro Altersgruppe der letzten 3 Jahren als Initialwert und lassen sie mit der Wachstumsrate der Grenzgänger wachsen.

```
# Take the mean numbers of paternity leave among frontier
# workers of the last 3 years availabe as starting point
VSU_AU_mean_reg <- VSU_AU |>
    group_by(kat) |>
    arrange(jahr, .by_group = TRUE) |>
    mutate(vaterschaftsurlaub_reg = mean(tail(vaterschaftsurlaub,
        3))) |>
    select(-dom)

# Let it grow with the growth factor of the frontier
# workers
VSU_BENEF_AU <- VSU_AU |>
    full_join(FRONTALIERS_WR_SCEN, by = c("jahr", "kat")) |>
    full_join(VSU_AU_mean_reg |>
        select(kat, vaterschaftsurlaub_reg) |>
        unique(), by = "kat") |>
    mutate(vaterschaftsurlaub = ifelse(jahr > max(REGISTER_VSU$jahr),
        vaterschaftsurlaub_reg * (1 + wr_grenz), vaterschaftsurlaub)) |>
    select(jahr, kat, vaterschaftsurlaub)
```

Ähnlich wie bei den Müttern nehmen wir an, dass der Anteil der im Ausland lebenden anderen Elterteile mit Maximalentschädigung im Jahr `jahr_register` auf dieselbe Weise wächst wie der, der in der Schweiz lebenden anderen Elterteile mit Maximalentschädigung (Siehe Kapitel **??**).

```
# Compute the part of paternity leave abroad which gets the
# maximum indemnity We take only the last year because of
# the EO MAX increase in 2023.
anteil_eomax_au <- VSU_AU |>
```

```
    mutate(anteil_eomax_register = vaterschaftsurlaub_eomax/vaterschaftsurlaub) |>
    filter(jahr == jahr_register) |>
    select(kat, anteil_eomax_register)


# Growth factor of the percentage of father leaving in
# Switzerland and getting EOMAX
ANTEIL_EOMAX <- VSU_BENEF_CH |>
    select(jahr, kat, anteil_eomax) |>
    filter(jahr >= jahr_register) |>
    group_by(kat) |>
    arrange(jahr) |>
    mutate(wr_anteil_eomax = (anteil_eomax/first(anteil_eomax) -
        1)) |>
    ungroup() |>
    select(jahr, kat, wr_anteil_eomax)

VSU_AU_anteil_eomax <- ANTEIL_EOMAX |>
    full_join(anteil_eomax_au, by = "kat") |>
    group_by(jahr, kat) |>
    mutate(anteil_eomax = anteil_eomax_register * (1 + wr_anteil_eomax)) |>
    ungroup() |>
    select(jahr, kat, anteil_eomax) |>
    filter(jahr > jahr_register)
```

Man kann nun die Anzahl der Urlaube des anderen Elternteils mit Domizil im Ausland berechnen, die die maximale Entschädigung erhalten, und die Anzahl derer, die eine Entschädigung unterhalb dieser erhalten. Dafür wenden wir die prognostizierten Anteile auf die Summe der Urlaube im Ausland an.

```
# Separate the total number of paternity leaves between the
# one getting the max and the one getting less in the
# future
VSU_BENEF_AU_fc <- VSU_BENEF_AU |>
    full_join(VSU_AU_anteil_eomax, by = c("jahr", "kat")) |>
    group_by(jahr, kat) |>
    mutate(vaterschaftsurlaub_eomax = vaterschaftsurlaub * anteil_eomax,
        vaterschaftsurlaub_ohne_eomax = vaterschaftsurlaub -
            vaterschaftsurlaub_eomax, dom = 2) |>
    ungroup() |>
    select(dom, jahr, kat, vaterschaftsurlaub, vaterschaftsurlaub_eomax,
        vaterschaftsurlaub_ohne_eomax, anteil_eomax) |>
    filter(jahr > jahr_register)
```

Schliesslich fügen wir die historischen Daten und die Projektionen in einem einzigen Datensatz `VSU_BENEF_AU` zusammen.

```
# Join the forecast and the past observations
VSU_BENEF_AU <- VSU_AU |>
    select(dom, jahr, kat, vaterschaftsurlaub, vaterschaftsurlaub_eomax,
        vaterschaftsurlaub_ohne_eomax, anteil_eomax) |>
    bind_rows(VSU_BENEF_AU_fc)
```

Somit ist die Berechnung des Anzahl Urlaub der anderen Elternteil abgeschlossen und das Dataframe `VSU_BENEF_AU` wird an das Modul `mod_eo_vsu` zurückgegeben.

```
# Output
# ------------------------------------------------------------------------
return(list(VSU_BENEF_AU = VSU_BENEF_AU))
```

4.8.3.3 Modul mod_eo_vsu_taggeld.R *Dokumentation zuletzt aktualisiert am 28.08.2025*

Dieses Modul dient zur Berechnung des durchschnittlichen Taggelds für Eltern, die nicht den Höchstbetrag erhalten. Folgende Argumente sind notwendig:

```
mod_eo_vsu_taggeld <- function(PARAM_GLOBAL,
                               REGISTER_VSU,
                               EOMAX,
                               ECKWERTE_EXTENDED,
                               STRUKTURFAKTOR
                               ) {
```

Zuerst werden die Registerdaten gefiltert, um nur die notwendigen Variablen zu behalten.

```
# Prepare VSU_CH and VSU_AU datasets by filtering and
# selecting required columns
VSU_TAGGELD <- REGISTER_VSU |>
    select(jahr, kat, dom, taggeld, vaterschaftsurlaub_ohne_eomax)
```

Anschliessend wird die jährliche Entwicklung des durchschnittlichen Lohnniveaus anhand der Entwicklung des Schweizer Lohnindexes (`lohn`) und der Entwicklung des Strukturfaktors (`struktur`) berechnet.

```
# Annahme: Das durchschnittliche Taggeld entwickelt sich
# entsprechend dem Wachstum des SLI sowie struktureller
# Veränderungen - konsistent mit der Entwicklung der
# Beiträge.
strukturfaktor <- STRUKTURFAKTOR %>%
    filter(sex == "m") %>%
    pull(strukturfaktor)

EINK_ENTWICKLUNG <- ECKWERTE_EXTENDED |>
    mutate(wr_eink = (lohn + strukturfaktor + lohn * strukturfaktor)/100)
```

Wir verkfüpfen diese Lohnentwicklung mit allen möglichen Kombinationen von `EINK_ENTWIKCLUNG$jahr` und `dom`.

```
DOM <- expand.grid(jahr = EINK_ENTWICKLUNG$jahr, dom = unique(VSU_TAGGELD$dom))

WR_EINK_ENTWICKLUNG <- EINK_ENTWICKLUNG |>
    select(jahr, wr_eink) |>
    full_join(DOM, by = "jahr")
```

Wir berechnen für jedes Jahr und Domizil die durchschnittliche Wachstumsrate des Taggeldes der Eltern, die nicht eomax erhalten, gewichtet mit der Anzahl der Leistungsempfänger nach Altersgruppen.

```
# Schätzung der Effekt der Erhöhung von EOMAX
WR_TAGGELD <- VSU_TAGGELD |>
    group_by(kat, dom) |>
    arrange(jahr) |>
    mutate(wr_taggeld = (taggeld - lag(taggeld))/lag(taggeld)) |>
    ungroup() |>
    group_by(jahr, dom) |>
```

```
    summarise(weigthed_wr_taggeld = sum(wr_taggeld * vaterschaftsurlaub_ohne_eomax,
        na.rm = TRUE)/sum(vaterschaftsurlaub_ohne_eomax, na.rm = TRUE))
```

Die Lohn- und Gehaltsentwicklung sowie die Entwicklung der Taggelder werden in einem Datensatz zusammengefasst.

```
WR_TAGGELD <- WR_TAGGELD |>
    full_join(WR_EINK_ENTWICKLUNG, by = c("jahr", "dom"))
```

Die Auswirkungen einer Anpassung der Höchstbeträge werden auf Basis des Jahres 2023 geschätzt, da dies das einzige Jahr ist, für das Daten vorliegen, die von einer Anpassung betroffen sind. Unter der Annahme, dass die Entwicklung der Entschädigungen eng mit der Entwicklung des Schweizer Lohnniveaus verknüpft ist, besteht der Effekt einer Anpassung in der Differenz zwischen der Entwicklung der Entschädigungen und jener der Löhne im Jahr 2023.

```
EFFEKT_EOMAX <- WR_TAGGELD |>
    filter(jahr == jahr_adapt_eomax) |>
    mutate(effekt_eomax = weigthed_wr_taggeld - wr_eink) |>
    ungroup() |>
    select(dom, effekt_eomax)
```

Wir verknüpfen die Auswirkungen einer Anpassung der Höchstbeträge (`effekt_eomax`) mit allen möglichen Kombinationen der Jahre von `jahr_register` bis `jahr_ende`, der Alterskategorien (`kat`) und des Domizils (`dom`). Dies erlaubt uns eine effiziente Handhabung der Daten – `effekt_eomax` hat aber überall denselben Wert.

```
FRAME_EOMAX <- expand.grid(jahr = jahr_register:jahr_ende, kat = 1:7,
    dom = unique(VSU_TAGGELD$dom))

EFFEKT_EOMAX <- FRAME_EOMAX |>
    full_join(EFFEKT_EOMAX)
```

Die Wachstumsrate des durchschnittlichen Tagegelds für Eltern, die nicht das maximale Tagegeld erhalten (`wr_taggeld`), entspricht den erwarteten Einkommenswachstumsraten (`wr_eink`) in den Jahren, in denen die Höchstbeträge nicht angepasst werden. In den Jahren, in denen eine Anpassung vorgesehen ist (abbgebildet mit `dummy_eomax`), hängt die Entwicklung der Taggelder sowohl von der Einkommensentwicklung als auch vom geschätzten Effekt der Anpassung (`effekt_eomax`) ab.

```
TAGGELD <- VSU_TAGGELD |>
    select(jahr, kat, dom, taggeld) |>
    full_join(WR_TAGGELD |>
        select(-weigthed_wr_taggeld) |>
        expand_grid(tibble(kat = 1:7))) |>
    full_join(EOMAX |>
        select(jahr, dummy_eomax) |>
        filter(jahr >= jahr_register), by = "jahr") |>
    full_join(EFFEKT_EOMAX, by = c("jahr", "kat", "dom")) |>
    filter(jahr >= jahr_register) |>
    group_by(jahr, kat, dom) |>
    mutate(wr_taggeld = wr_eink + effekt_eomax * dummy_eomax) |>
    ungroup()
```

Mithilfe diesen jährlichen Wachstumsraten (`wr_taggeld`) und der Funktion `accumulate()` können wir die durchschnittlichen Taggelder des anderen Elternteils fortschreiben. Wir gruppieren die Daten zunächst nach Alterskategorie (`kat`) und Domizil (`dom`). Das bedeutet, dass alle folgenden Berechnungen innerhalb jeder Kombination dieser beiden Variablen durchgeführt werden. Wir sortieren die Daten nach der Variablen `jahr`,

um sicherzustellen, dass alle Berechnungen für das Tagegeld korrekt über die Jahre hinweg durchgeführt werden. Danach werden die Taggelder berechnet:

- Der erste Wert des taggeld bleibt unverändert (`taggeld[1]`), um als Startwert zu dienen.
- Ab dem zweiten Jahr wird das Taggeld kumulativ berechnet, basierend auf den Werten von `wr_taggeld` (Zuwachsrate) und dem letzten berechneten Tagegeld.
- Die `accumulate`-Funktion berechnet diese kumulative Veränderung, indem sie für jedes Jahr das Tagegeld mit der Wachstumsrate `(1 + .y)` multipliziert. Das erste Element der kumulierten Liste wird entfernt (`[-1]`), da es redundant ist.

```
# Calculate 'taggeld' accumulation for VSU_TAGGELD_CH
VSU_TAGGELD <- TAGGELD |>
    group_by(kat, dom) |>
    arrange(jahr) |>
    mutate(taggeld = c(taggeld[1], accumulate(wr_taggeld[-1],
        .init = taggeld[1], ~.x * (1 + .y))[-1])) |>
    select(jahr, kat, dom, taggeld)
```

Schließlich werden die Registerdaten und die Taggeldprognosen in einem Datensatz `VSU_TAGGELD` zusammengefasst, je nach Domizil (`dom`) und Alterskategorie (`kat`) des Elternteils. Dieser Datensatz wird in das Modul `mod_eo_vsu` zurückgegeben.

```
# Join past values and forecasts
VSU_TAGGELD <- VSU_TAGGELD |>
    filter(jahr > last(REGISTER_VSU$jahr)) |>
    full_join(REGISTER_VSU |>
        select(jahr, kat, dom, taggeld)) |>
    arrange(jahr)


# Output
# ------------------------------------------------------------------------
return(list(VSU_TAGGELD = VSU_TAGGELD))
```

4.8.4 Modul mod_eo_betreuung.R

Dokumentation zuletzt aktualisiert am 28.08.2025

Dieses Modul berechnet die Ausgaben im Zusammenhang mit dem Betreuungsurlaub. Folgende Argumente werden verwendet:

```
mod_eo_betreuung <- function(PARAM_GLOBAL,
                             REGISTER_BEU,
                             MSU_AUSGABEN,
                             VSU_AUSGABEN,
                             EO_ABRECHNUNG_DEF,
                             EO_AUSGABEN_ANTEILE) {
```

Zuerst teilen wir die Kosten für den Betreuungsurlaub auf, basierend darauf, welches Geschlecht den Urlaub in Anspruch genommen hat – entweder die Mutter bzw. Mütter - im Falle gleichgeschlechtlicher Paare - oder der Vater. Erstere werden mit `ausgaben_adopt_mut` bezeichnet, letztere mit `ausgaben_adopt_vat.`

```
# Ausgaben Betreuungsurlaub
AUSGABEN_BSU_HIST <- REGISTER_BEU |>
    select(jahr, ausgaben_bsu = ausgaben)
```

```
# Distribution of historic expenses between mothers and
# other relatives
AUSGABEN_BSU_HIST <- AUSGABEN_BSU_HIST |>
    left_join(REGISTER_BEU |>
        select(jahr, anteil_muetter), by = "jahr") |>
    mutate(ausgaben_bsu_mut = anteil_muetter * ausgaben_bsu,
        ausgaben_bsu_vat = ausgaben_bsu - ausgaben_bsu_mut)
```

Dann berechnen wir das projizierte Wachstum der geschätzten Ausgaben für Mutterschaftsurlaub und für den Urlaub des anderen Elternteils.

```
# Growth rate of maternity leave expenditure
WR_MUT <- MSU_AUSGABEN |>
    filter(jahr >= jahr_register) |>
    arrange(jahr) |>
    mutate(wr_mut = (aus_tot_mut - first(aus_tot_mut))/aus_tot_mut) |>
    select(jahr, wr_mut)

# Growth rate of paternity leave expenditure
WR <- VSU_AUSGABEN |>
    filter(jahr >= jahr_register) |>
    arrange(jahr) |>
    mutate(wr_vat = (aus_tot_vsu - first(aus_tot_vsu))/aus_tot_vsu) |>
    select(jahr, wr_vat) |>
    right_join(WR_MUT, by = "jahr")
```

Als Ausgangspunkte für die Prognosen verwenden wir den Durchschnitt der letzten zwei Jahren der geschlechtsspezifischen Ausgaben für den Betreuungsurlaub.

```
# Mean of the last two years as starting point for the
# projections
AUSGABEN_BSU_reg <- AUSGABEN_BSU_HIST |>
    arrange(jahr) |>
    slice_tail(n = 2) |>
    mutate(ausgaben_bsu_mut_reg = mean(ausgaben_bsu_mut), ausgaben_bsu_vat_reg =
    ↪  mean(ausgaben_bsu_vat)) |>
    select(ausgaben_bsu_mut_reg, ausgaben_bsu_vat_reg) |>
    unique()
```

Wir wenden auf diese Ausgangswerte die Wachstumsraten der Ausgaben für den Mutterschaftsurlaub bzw. für den Urlaub des anderen Elternteils an, um die Ausgabenprognosen für den Betreuungsurlaub zu erstellen.

```
# Projections with the maternity and paternity leave growth
# rate
BSU_AUSGABEN <- AUSGABEN_BSU_HIST |>
    full_join(WR, by = "jahr") |>
    mutate(ausgaben_bsu_mut = if_else(jahr <= jahr_register,
        ausgaben_bsu_mut, AUSGABEN_BSU_reg$ausgaben_bsu_mut_reg *
            (1 + wr_mut)), ausgaben_bsu_vat = if_else(jahr <=
        jahr_register, ausgaben_bsu_vat, AUSGABEN_BSU_reg$ausgaben_bsu_vat_reg *
        (1 + wr_vat)), ausgaben_bsu = ausgaben_bsu_mut + ausgaben_bsu_vat) |>
    select(jahr, ausgaben_bsu)
```

Aufwendungen für Abschreibungen von Rückerstattungsforderungen, die mit den Ausgaben für Betreuungsurlaube verbunden sind, werden als Anteil der totalen Abschreibungen von Rückerstattungsforderungen errechnet.

```
# Abschreibung von Rueckerstattungsforderungen
# (anteilmaessig)
ABSCHREIB_RUECKF <- EO_ABRECHNUNG_DEF |>
    select(jahr, abschr_rueckf) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_bsu), by = "jahr") |>
    mutate(abschreib_rueckf = anteil_bsu * abschr_rueckf) |>
    select(jahr, abschreib_rueckf)
```

Der Beitragsanteil zulasten der EO, der mit Betreuungsurlauben einhergeht, wird als Anteil des totalen Beitragsanteils errechnet.

```
# Beitragsanteil zu Lasten der EO (anteilmaessig)
BTR_ANT_EO <- EO_ABRECHNUNG_DEF |>
    select(jahr, btr_ant_eo) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_bsu), by = "jahr") |>
    mutate(btr_ant_eo = anteil_bsu * btr_ant_eo) |>
    select(jahr, btr_ant_eo)
```

Wir nehmen an, dass der Anteil dieser Kosten im Vergleich zu den anderen Betreuungsurlaubskosten konstant bleibt und schreiben sie mithilfe der Projektionen der anderen Betreuungsurlaubskosten weiter. Die totalen Ausgaben für den Betreuungsurlaub werden als Summe der errechneten Kosten `ausgaben_beu`, der Abschreibungen von Rückerstattungsforderungen und des Beitragsanteils zulasten der EO errechnet.

```
# Add Abschreibung und Beitragsanteil zu den
# Betreuungsausgaben
BSU_AUSGABEN <- BSU_AUSGABEN |>
    full_join(ABSCHREIB_RUECKF, by = "jahr") |>
    full_join(BTR_ANT_EO, by = "jahr") |>
    mutate(anteil_abschreib = abschreib_rueckf/ausgaben_bsu,
        anteil_btr = btr_ant_eo/ausgaben_bsu) |>
    fill(anteil_abschreib, anteil_btr, .direction = "down") |>
    mutate(btr_ant_eo = anteil_btr * ausgaben_bsu, abschreib_rueckf = anteil_abschreib *
        ausgaben_bsu, ausgaben_bsu = ausgaben_bsu + btr_ant_eo +
        abschreib_rueckf) |>
    full_join(EO_ABRECHNUNG_DEF |>
        select(jahr, geld_leist), by = "jahr") |>
    full_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_bsu), by = "jahr") |>
    mutate(ausgaben_bsu = if_else(jahr <= jahr_abr, geld_leist *
        anteil_bsu, ausgaben_bsu)) |>
    select(jahr, ausgaben_bsu)
```

Schliesslich werden die Ausgaben bis zum Jahr `jahr_abr` so angepasst, dass die Gesamtsumme mit den Buchhaltungsdaten (`geld_leist`) übereinstimmt. Etwaige Abweichungen kommen z.B. bei Leistungen für die Elternschaft davon, dass sich der Beginn des Leistungsanspruchs (Hospitalisierung des Kindes) nicht unbedingt mit dem Abrechnungszeitpunkt übereinstimmt.

Somit ist die Berechnung der Ausgaben für den Betreuungsurlaub abgeschlossen und die Ergebnisse `BSU_AUSGABEN` werden an das Modul `mod_eo_ausgaben` zurückgegeben.

```
#---Output--------------------------------------------------------------------
```

```
return(list(BSU_AUSGABEN = BSU_AUSGABEN))
```

4.8.5 Modul mod_eo_adoption.R

Dokumentation zuletzt aktualisiert am 28.08.2025

Dieses Modul dient zur Schätzung der Ausgaben für den Adoptionsurlaub und ist ähnlich aufgebaut wie `mod_eo_betreuung.R`. Es benötigt folgende Argumente:

```
mod_eo_adoption <- function(PARAM_GLOBAL,
                            REGISTER_ADOPT,
                            MSU_AUSGABEN,
                            VSU_AUSGABEN,
                            EO_ABRECHNUNG_DEF,
                            EO_AUSGABEN_ANTEILE) {
```

Zunächst verteilen wir die Kosten für den Adoptionsurlaub basierend darauf, welches Geschlecht den Urlaub in Anspruch genommen hat – entweder die Mutter bzw. Mütter - im Falle gleichgeschlechtlicher Paare - oder der Vater. Erstere werden mit `ausgaben_adopt_mut` bezeichnet, letztere mit `ausgaben_adopt_vat.`

```
# Ausgaben Adoptionsurlaub
AUSGABEN_ADOPT_HIST <- REGISTER_ADOPT |>
    select(jahr, ausgaben_adopt = ausgaben)


# Select the part of leave taken by the mother
AUSGABEN_ADOPT_HIST <- AUSGABEN_ADOPT_HIST |>
    left_join(REGISTER_ADOPT |>
        select(jahr, anteil_muetter), by = "jahr") |>
    mutate(ausgaben_adopt_mut = anteil_muetter * ausgaben_adopt,
        ausgaben_adopt_vat = ausgaben_adopt - ausgaben_adopt_mut)
```

Anschliessend berechnen wir das prognostizierte jährliche Wachstum der zuvor geschätzten Ausgaben für den Mutterschaftsurlaub sowie für den Urlaub des anderen Elternteils.

```
# Compute the growth factor of the maternity and paternity
# leave expenses
WR_MUT <- MSU_AUSGABEN |>
    filter(jahr >= jahr_register) |>
    arrange(jahr) |>
    mutate(wr_mut = (aus_tot_mut - first(aus_tot_mut))/aus_tot_mut) |>
    select(jahr, wr_mut)

WR <- VSU_AUSGABEN |>
    filter(jahr >= jahr_register) |>
    arrange(jahr) |>
    mutate(wr_vat = (aus_tot_vsu - first(aus_tot_vsu))/aus_tot_vsu) |>
    select(jahr, wr_vat) |>
    right_join(WR_MUT, by = "jahr")
```

Als Ausgangspunkte für die Prognosen verwenden wir die geschlechtsspezifischen Ausgaben für den Adoptionsurlaub des letzten Registersjahres (`jahr_register`) .

```
# Starting points for the projections
AUSGABEN_ADOPT_reg <- AUSGABEN_ADOPT_HIST |>
    filter(jahr == jahr_register) |>
```

```
    select(ausgaben_adopt_mut_reg = ausgaben_adopt_mut, ausgaben_adopt_vat_reg =
    ↪  ausgaben_adopt_vat)
```

Wir wenden die Wachstumsraten der Ausgaben für den Mutterschafts- bzw. den Urlaub des anderen Elternteils auf diese Ausgangswerte an, um die prognostizierten Ausgaben für den Adoptionsurlaub zu erhalten.

```
# Projections with the maternity and paternity leave growth
# rate
ADOPTION_AUSGABEN <- AUSGABEN_ADOPT_HIST |>
    full_join(WR, by = "jahr") |>
    mutate(ausgaben_adopt_mut = if_else(jahr <= jahr_register,
        ausgaben_adopt_mut, AUSGABEN_ADOPT_reg$ausgaben_adopt_mut_reg *
            (1 + wr_mut)), ausgaben_adopt_vat = if_else(jahr <=
        jahr_register, ausgaben_adopt_vat, AUSGABEN_ADOPT_reg$ausgaben_adopt_vat_reg *
        (1 + wr_vat)), ausgaben_adopt = ausgaben_adopt_mut +
        ausgaben_adopt_vat) |>
    select(jahr, ausgaben_adopt)
```

Aufwendungen für Abschreibungen von Rückerstattungsforderungen, die mit den Ausgaben für Adoptionsurlaube verbunden sind, werden als Anteil der totalen Abschreibungen von Rückerstattungsforderungen errechnet.

```
# Abschreibung von Rueckerstattungsforderungen
# (anteilmaessig)
ABSCHREIB_RUECKF <- EO_ABRECHNUNG_DEF |>
    select(jahr, abschr_rueckf) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_adopt), by = "jahr") |>
    mutate(abschreib_rueckf = anteil_adopt * abschr_rueckf) |>
    select(jahr, abschreib_rueckf)
```

Der Beitragsanteil zulasten der EO, der mit Adoptionsurlauben einhergeht, wird als Anteil des totalen Beitragsanteils errechnet.

```
# Beitragsanteil zu Lasten der EO (anteilmaessig)
BTR_ANT_EO <- EO_ABRECHNUNG_DEF |>
    select(jahr, btr_ant_eo) |>
    filter(jahr >= min(EO_AUSGABEN_ANTEILE$jahr)) |>
    right_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_adopt), by = "jahr") |>
    mutate(btr_ant_eo = anteil_adopt * btr_ant_eo) |>
    select(jahr, btr_ant_eo)
```

Wir nehmen an, dass der Anteil dieser Kosten im Vergleich zu den anderen Kosten des Adoptionsurlaubs konstant bleibt und schreiben ihn mithilfe der Projektionen der anderen Adoptionsurlaubskosten weiter. Die totalen Ausgaben für den Adoptionsurlaub werden als Summe der errechneten Kosten `ausgaben_adopt`, der Abschreibungen von Rückerstattungsforderungen und des Beitragsanteils zulasten der EO errechnet.

```
# Add Abschreibung und Beitragsanteil zu den
# Adoptionsausgaben
ADOPTION_AUSGABEN <- ADOPTION_AUSGABEN |>
    full_join(ABSCHREIB_RUECKF, by = "jahr") |>
    full_join(BTR_ANT_EO, by = "jahr") |>
    mutate(anteil_abschreib = abschreib_rueckf/ausgaben_adopt,
        anteil_btr = btr_ant_eo/ausgaben_adopt) |>
```

```
    fill(anteil_abschreib, anteil_btr, .direction = "down") |>
    mutate(btr_ant_eo = anteil_btr * ausgaben_adopt, abschreib_rueckf = anteil_abschreib
    ↪  *
        ausgaben_adopt, ausgaben_adopt = ausgaben_adopt + btr_ant_eo +
        abschreib_rueckf) |>
    full_join(EO_ABRECHNUNG_DEF |>
        select(jahr, geld_leist), by = "jahr") |>
    full_join(EO_AUSGABEN_ANTEILE |>
        select(jahr, anteil_adopt), by = "jahr") |>
    mutate(ausgaben_adopt = if_else(jahr <= jahr_abr, geld_leist *
        anteil_adopt, ausgaben_adopt)) |>
    select(jahr, ausgaben_adopt)
```

Schliesslich werden die Ausgaben bis zum Jahr `jahr_abr` so angepasst, dass die Gesamtsumme mit den Buchhaltungsdaten (`geld_leist`) übereinstimmt. Etwaige Abweichungen kommen z.B. bei Leistungen für die Elternschaft davon, dass sich der Beginn des Leistungsanspruchs (Adoption des Kindes) nicht unbedingt mit dem Abrechnungszeitpunkt übereinstimmt.

Somit ist die Berechnung der Adoptionsausgaben abgeschlossen und die Ergebnisse werden mit dem foglenden Befehl an das Modul `mod_eo_ausgaben` zurückgegeben werden.

```
#---Output--------------------------------------------------------------------

return(list(ADOPTION_AUSGABEN = ADOPTION_AUSGABEN))
```

4.8.6 Modul mod_eo_verwaltungskosten.R

Dokumentaiton zuletzt aktualisiert am 03.09.2025

Dieses Modul schätzt die zukünftigen Verwaltungskosten. Es bentötigt die folgende Argumente:

```
mod_eo_verwaltungskosten <- function(PARAM_GLOBAL,
                                     EO_ABRECHNUNG_DEF,
                                     DISKONTFAKTOR) {
```

Die Verwaltungskosten (zu realen Preisen) werden als konstant angenommen und basieren auf dem Durchschnitt der Kosten der letzten drei Jahre. Zunächst wird der Durchschnitt der Verwaltungskosten der letzten drei Jahre berechnet.

```
mean_verwaltungskosten <- EO_ABRECHNUNG_DEF |>
    arrange(jahr) |>
    slice_tail(n = 3) |>
    summarise(mean_verwaltungskosten = mean(verw_kost)) |>
    pull(mean_verwaltungskosten)
```

Die zukünfitgen realen Verwaltungskosten entsprechen diesem Durchschnitt (`mean_verwaltungskosten`). Um die nominalen Verwaltungskosten zu berechnen, wird dieser Wert jedoch noch durch den Diskontfaktor (`diskontfaktor`) geteilt.

```
EO_VERWALTUNGSKOSTEN <- EO_ABRECHNUNG_DEF |>
    select(jahr, verw_kost) |>
    bind_rows(tibble(jahr = (jahr_abr + 1):jahr_ende, verw_kost =
    ↪  mean_verwaltungskosten)) |>
    right_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(verw_kost = verw_kost/diskontfaktor) |>
    select(jahr, verw_kost)
```

Die Ergebnisse werden an das Modul `mod_eo_ausgaben` zurückgegeben:

```
#---Output--------------------------------------------------------------------

return(list(EO_VERWALTUNGSKOSTEN = EO_VERWALTUNGSKOSTEN))
```

Somit ist die Berechnung der Ausgaben der EO gemäss geltender Ordnung abgeschlossen.

4.9 Modul wrap_eo_massnahmen.R

Dokumentation zuletzt aktualisiert am 03.09.2025

Das Modul zu den Massnahmen dient dazu, die Auswirkungen von Politikmassnahmen auf die EO abzuschätzen. Es wird in `wrap_eo` wie folgt aufgerufen:

```
# Berechnung Massnahmen
tl_massnahmen <- wrap_eo_massnahmen(tl_inp = tl_inp, tl_vorb_berechn = tl_vorb_berechn,
    tl_hauptberechnung = tl_hauptberechnung)
```

Als erstes werden die Einnahmen und Ausgaben aus `wrap_eo_hauptberechnung` geladen und zwei leere Dataframes erstellt, in welchen später die Auswirkungen von Massnahmen gespeichert werden:

```
# Liste und Dataframe für die Resultate initialisieren
tl_opt <- list()
EO_AUSGABEN <- tl_hauptberechnung$EO_AUSGABEN
EO_EINNAHMEN <- tl_hauptberechnung$EO_EINNAHMEN
EO_MASSNAHMEN <- data.frame()
```

Danach folgt die Prüfung, ob die Auswirkungen von Massnahmen in die Finanzperspektiven-Berechnungen mit einbezogen werden sollen:

```
  if (tl_inp$PARAM_GLOBAL$flag_param_massn &&
      !is.null(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen)) { # Überprüfen, ob
      ↪  Massnahmen spezifiziert sind und berechnet werden sollen
```

Wenn dies nicht der Fall ist, wird das Modul abgeschlossen und es werden die leeren Dataframes zurückgegeben:

```
  }

  c(
    tl_opt,
    list(
      EO_MASSNAHMEN = EO_MASSNAHMEN,
      EO_AUSGABEN = EO_AUSGABEN,
      EO_EINNAHMEN = EO_EINNAHMEN
    )
  )
```

Falls Massnahmen spezifiert wurden, wird die Massnahmen-Inputliste `tl_inp_massn` erstellt, welche die Input-Dataframes, die Dataframes der vorbereitenden Vorberechnugen und die Dataframes der Hauptberechnungen enthält:

```
tl_inp_massn <- c(tl_inp, tl_vorb_berechn, tl_hauptberechnung)
```

Danach werden in der EO-Ausgabe-Berechnung aus den Hauptberechnungen die Spalte mit den Totalausgaben `aus_total_ink_vk` entfernt, da die Totalausgaben neu berechnet werden, um die Effekte der Maßnahmen zu berücksichtigen:

```
# Totalausgaben von EO_AUSGABEN entfernen, da Totalausgaben
# neu berechnet werden müssen
EO_AUSGABEN <- EO_AUSGABEN %>%
    select(-aus_total_ink_vk)
```

Die spezifierten Massnahmen, die berücksichtigt werden sollen (sogenannte `aktivierte_massnahmen`) werden in einer Liste `massnahmen` abgelegt:

```
massnahmen <- separate_at_comma(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen)
```

Danach wird die Funktion spezifiziert, mit Hilfe welcher die Berechnungen der Auswirkungen der internen Massnahmen durchgeführt werden:

```
run_massnahmen <- function(massnahme_name, data) {
    funktion_name <- paste0("mod_opt_", massnahme_name)

    if (!exists(funktion_name)) {
        stop("Folgendes Massnahmen-Modul nicht in dmeasures gefunden: ",
            funktion_name)
    }

    # Call the module function with the data argument
    massnahme_effekte <- do.call(get(funktion_name), list(data))

    # Rename the data frames within massnahme_effekte
    names(massnahme_effekte) <- paste0(names(massnahme_effekte),
        "..", massnahme_name)

    # Return the modified massnahme_effekte
    return(massnahme_effekte)
}
```

Die Funktion `run_massnahmen` nimmt in `massnahme_name` den Namen der Massnahme entgegen, für welche die Auswirkungen berechnet werden sollen, und in `data` die Input-Daten, welche für die Berechnungen genutzt werden sollen. Als erstes überprüft die Funktion, ob eine Funktion (respektiv ein Modul) zum Berechnen der Auswirkungen existiert. Wenn dies nicht der Fall ist wird eine Fehlermeldung ausgegeben, und ansonsten werden die Auswirkungen der Massnahme in `massnahme_name` berechnet und in die Liste `massnahme_effekte` abgelegt. Danach wird an den Namen der Dataframes in der Liste `massnahme_effekte` der Name der betreffenden Massnahmen angefügt. Dies dient dazu, später die Dataframes mit den Massnahmeneffekte den richtigen Massnahmen zuzuordnen. Dananch wird die Liste `massnahme_effekte` zurückgegebn. Die Funktion wird mit dem `lappy` Befehl so aufgerufen, wodurch für jedes Element der Liste `massnahmen` (also für jede spezifierte Massnahme, die berechnet werden soll) `run_massnahmen` unter Verwendung der Daten in `tl_inp_massn` ausgeführt wird. Der umhüllende `do.call` dient lediglich dazu sicherzustellen, dass die resultierende Liste `tl_massnahme_effekte` direkt die Dataframes aus den Listen der verschiedenen Massnahmen-Berechnungen enthält, anstatt die Listen selbst.

```
# Apply the function and store the result in
# tl_massnahme_effekte
tl_massnahme_effekte <- do.call(c, lapply(massnahmen, run_massnahmen,
    data = tl_inp_massn))
```

Als nächstes werden die Dataframes mit den Massnahmen-Effekte aus `tl_massnahme_effekte` ausgelesen und in die Liste `tl_DELTAS_MASSNAHMEN` gelegt:

```
# Auswahl der Dataframes aus tl_massnahme_effekte, die mit
# 'DELTA' beginnen
```

```
tl_DELTAS_MASSNAHMEN <- tl_massnahme_effekte[grepl("^DELTA",
    names(tl_massnahme_effekte))]
```

Als nächstes werden für jedes Dataframe in `tl_DELTAS_MASSNAHMEN` die Massnahmen-Effekte ausgelesen und im langen Format in das Dataframe `DELTAS_MASSNAHMEN` gelegt:

```
# Prüfen und Umwandeln der Dataframes sowie Zeilenweise
# Verbinden
DELTAS_MASSNAHMEN <- lapply(names(tl_DELTAS_MASSNAHMEN), function(df_name) {
    df <- tl_DELTAS_MASSNAHMEN[[df_name]]

    # Prüfen, ob alle Spalten in IV_ABRECHNUNG existieren
    if (!all(colnames(df) %in% c(colnames(EO_AUSGABEN), colnames(EO_EINNAHMEN)))) {
        stop(sprintf("\033[31mFehler: Dataframe %s enthält Spalten, die weder in
        ↪  EO_AUSGABEN noch in EO_EINNAHMEN enthalten sind.\033[0m",
            df_name))
    }

    # Umwandeln in Long-Format und 'massnahme' hinzufügen
    df_long <- df %>%
        pivot_longer(cols = -jahr, names_to = "konto", values_to = "wert") %>%
        mutate(massnahme = sub(".*\\.\\.", "", df_name))  # massnahme aus dem Namen
        ↪  extrahieren

    return(df_long)
}) %>%
    bind_rows()  # Alle Dataframes zeilenweise verbinden
```

Hierbei wird als erstes überprüft, ob die Spaltennamen im entsprechenden Dataframe `df` auch in der EO-Ausgaben oder in der EO-Einnahmen enthalten sind. Zweck dieser Überprüfung ist es, sicherzustellen, dass alle finanziellen Auswirkungen der Massnahmen auch tatsächlich eineer Position im Finanzperspektivenmodell zugewiesen werden können. Als nächstes wird `df` in das lange Format gebracht, was heisst, dass die Spaltennahmen (welche die von der Massnahme betroffenen Positionen der EO-Abrechnung enthalten) in die Spalte `konto` kopiert werden und die Massnahmen-Effekte im entsprechenden Jahr in die Spalte `wert`.

Zum Abschluss der Berechungen für die Massnahmen werde auch noch die Dataframes it den Auswirkungen der Massnahmen, die nicht die EO-Abrechnung betreffen, also die `OPT`-Dataframesm ausgelesen:

```
# Auswahl der Dataframes aus tl_massnahme_effekte, die mit
# 'OPT' beginnen
tl_opt <- tl_massnahme_effekte[grepl("^OPT", names(tl_massnahme_effekte))]
```

Danach wird noch sichergestellt, dass `DELTAS_MASSNAHMEN` keine NA-Werte enthält:

```
# Überprüfe auf NA-Werte in der Spalte 'wert'
if (any(is.na(DELTAS_MASSNAHMEN$wert))) {
    stop("Massnahme-Schätzung enthält NA-Wert. Massnahme-Module überprüfen.")
}
```

Anschließend werden die Auswirkungen der Maßnahmen nach Kategorie – Einnahmen oder Ausgaben – unterschieden. Für jede Position der EO-Finanzperspektiven (`konto`) werden die jährlichen Änderungen bei den Ausgaben (`veraend_ausg`) und Einnahmen (`veraend_einn`) berechnet, indem die entsprechenden Spalten summiert werden.

```
# Massnahmen-Effekte in breites Format bringen und
# Spaltenbenennung anpassen
```

```
EO_MASSNAHMEN <- DELTAS_MASSNAHMEN |>
    mutate(konto = case_when(konto %in% colnames(EO_AUSGABEN) ~
        paste0("ausgaben..", massnahme), konto %in% colnames(EO_EINNAHMEN) ~
        paste0("einnahmen..", massnahme), TRUE ~ NA_character_)) |>
    filter(!is.na(konto)) |>
    group_by(jahr, konto) |>
    summarise(wert = sum(wert, na.rm = TRUE), .groups = "drop") |>
    pivot_wider(names_from = konto, values_from = wert, values_fill = 0)
```

Dann wird die Gesamtwirkung (`wirkung_total`) als Differenz von Einnahmen und Ausgaben bestimmt.

```
# Totale Wirkung der Massnahmen berechnen
EO_MASSNAHMEN <- EO_MASSNAHMEN |>
    mutate(veraend_ausg = rowSums(EO_MASSNAHMEN[, grep("^ausgaben\\.\\.",
        colnames(EO_MASSNAHMEN)), drop = FALSE], na.rm = TRUE),
        veraend_einn = rowSums(EO_MASSNAHMEN[, grep("^einnahmen\\.\\.",
            colnames(EO_MASSNAHMEN)), drop = FALSE], na.rm = TRUE),
        wirkung_total = veraend_einn - veraend_ausg)
```

Danach werden die EO Ausgaben pro Leistungsart und die Totalausgaben (`aus_total_ink_vk`) unter Berücksichtigung der Effekte der Massnahmen werden pro Jahr neu berechnet.

```
      # EO_AUSGABEN unter Berücksichtigung der Effekte der Massnahmen neu berechnen
      EO_AUSGABEN <- EO_AUSGABEN |>
        pivot_longer(cols = -jahr, names_to = "konto", values_to = "wert") |>  # Bring to
        ↪  long format
        bind_rows(
          DELTAS_MASSNAHMEN |> filter(konto %in% colnames(EO_AUSGABEN))  # Append
↪  DELTAS_MASSNAHMEN
        ) |>
        group_by(jahr, konto) |>
        summarise(wert = sum(wert, na.rm = TRUE), .groups = "drop")

      EO_AUSGABEN <- EO_AUSGABEN |>
        group_by(jahr) |>
        summarise(wert = sum(wert, na.rm = TRUE)) |> # Totalausgaben neu berechnen
        mutate(konto = "aus_total_ink_vk") |>
        bind_rows(EO_AUSGABEN) |>  # Aggregate by jahr & konto
        pivot_wider(names_from = konto, values_from = wert, values_fill = 0)  # Reshape
        ↪  back to wide format
```

Analog zu den Ausgaben werden die Einnahmen mit der Effekte der Massnahmen auch pro Jahr und `konto` summiert.

```
  # EO_EINNAHMEN unter Berücksichtigung der Effekte der Massnahmen neu berechnen
      EO_EINNAHMEN <- EO_EINNAHMEN |>
        pivot_longer(cols = -jahr, names_to = "konto", values_to = "wert") |>  # Bring to
        ↪  long format
        bind_rows(
          DELTAS_MASSNAHMEN |> filter(konto %in% colnames(EO_EINNAHMEN))  # Append
↪  DELTAS_MASSNAHMEN
        ) |>
        group_by(jahr, konto) |>
        summarise(wert = sum(wert, na.rm = TRUE), .groups = "drop") |>  # Aggregate by
        ↪  jahr & konto
```

```
        pivot_wider(names_from = konto, values_from = wert, values_fill = 0)  # Reshape
        ↪  back to wide format
```

Die Ergebnisse werden schlussendlich an das Modul `wrap_eo.R` zurückgegeben:

```
c(tl_opt, list(EO_MASSNAHMEN = EO_MASSNAHMEN, EO_AUSGABEN = EO_AUSGABEN,
    EO_EINNAHMEN = EO_EINNAHMEN))
```

4.10 Modul wrap_eo_ergebnisse.R

Dokumentation zuletzt aktualisiert am 17.09.2025

Das Modul `wrap_eo_ergebnisse.R` dient dazu, die Erfolgsrechnung inklusive Umlageergebnis, und die Bilanz inklusive Kapital der EO zu berechnen. Es wird in `wrap_eo.R` wie folgt aufgerufen:

```
# Berechnung Umlageergebnis und Bilanz
tl_ergebnisse <- wrap_eo_ergebnisse(tl_inp = tl_inp, tl_vorb_berechn = tl_vorb_berechn,
    tl_massnahmen = tl_massnahmen)
```

In `wrap_eo_ergebnisse` wird das Modul, das die Bilanz der EO für jedes zukünftige Jahr berechnet (vgl. 4.10.1) aufgerufen:

```
# Berechnung von Ertrag der Anlagen, Betriebsergebnis und
# Fondsstand EO
EO_BILANZ <- mod_eo_bilanz(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EO_AUSGABEN = tl_massnahmen$EO_AUSGABEN, EO_EINNAHMEN = tl_massnahmen$EO_EINNAHMEN,
    ZINS = tl_vorb_berechn$ZINS, EO_ABRECHNUNG = tl_vorb_berechn$EO_ABRECHNUNG)
```

Sobald dieses Modul ausgeführt ist wird der resultierende Output an das Finanzperspektivenmodell, respektive das Modul `wrap_eo` zurückgegeben:

```
# Output
c(list(EO_BILANZ = EO_BILANZ))
```

4.10.1 Modul mod_eo_bilanz.R

Dokumentation zuletzt aktualisiert am 17.09.2025

`mod_eo_bilanz.R` wird im Skript `wrap_eo_ergebnisse` wie folgt aufgerufen:

```
# Berechnung von Ertrag der Anlagen, Betriebsergebnis und
# Fondsstand EO
EO_BILANZ <- mod_eo_bilanz(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EO_AUSGABEN = tl_massnahmen$EO_AUSGABEN, EO_EINNAHMEN = tl_massnahmen$EO_EINNAHMEN,
    ZINS = tl_vorb_berechn$ZINS, EO_ABRECHNUNG = tl_vorb_berechn$EO_ABRECHNUNG)
```

Die Funktion verwendet neben `PARAM_GLOBAL` die folgende Dataframes:

- `EO_AUSGABEN`: Sämtliche Ausgabeprojektionen, inklusive der Ausgaben auf Grund von Politikmassnahmen,
- `EO_EINNAHMEN`: Sämtliche Einnahmeprojektionenm mit Ausnahme des Zinsertrages (dieser kann erst nach Berechnung des Fondsstandprojektion ermittelt werden), inklusive der Einnahmen auf Grund von Politikmassnahmen.
- `ZINS`: Der für das Anlagekapital angenommene Zinssatz
- `EO_ABRECHNUNG`: Die vergangene Einnahmen und Ausgaben gemäss EO-Abrechnung.

Zuerst wird gepfrüft, ob das Abrechnungsjahr 2024 ist, und wenn ja, wird der Wert des Kapitales korrigiert um die Auswirkung der neuen Rechnunglegung IPSAS zu berücksichtigen.

```
# Falls Abrechnungsjahr 2024 ist: Auswirkung der neuen
# Rechnungslegung (IPSAS) berücksichtigen: Reduktion der
# Eigenmittel um 256 Mio. CHF (Restatement)
if (PARAM_GLOBAL$jahr_abr == 2024) {
    EO_ABRECHNUNG <- EO_ABRECHNUNG %>%
        mutate(kap = ifelse(jahr == 2024, kap - 255729982, kap))
}
```

In einem nächsten Schritt werden die Einnahmen- und Ausgabenprojektionen mit der EO-Abrechnung zusammengefügt:

```
# Bilanz aus Einnahme- und Ausgabevektor und Abrechnung
# erstellen
EO_BILANZ <- tibble(jahr = PARAM_GLOBAL$jahr_beginn:PARAM_GLOBAL$jahr_ende) %>%
    left_join(EO_AUSGABEN, by = "jahr") %>%
    left_join(EO_EINNAHMEN, by = "jahr") %>%
    mutate_all(list(~as.numeric(.))) %>%
    mutate(umlage_eo = eo_beitragsum - aus_total_ink_vk) %>%
    left_join(EO_ABRECHNUNG %>%
        select(jahr, kap, anl_erg, erg_betr, fl_mtl) %>%
        rename(kap_abr = kap, fluss_mittel = fl_mtl, kap_etr = anl_erg),
        by = "jahr") %>%
    left_join(ZINS %>%
        select(jahr, nominal_zins_fonds), by = "jahr")
```

Hierzu wird das projizierte Umlageergebnis der EO berechnet, indem die projizierten Totalausgaben (einschließlich Verwaltungskosten) von den Lohnbeitragsprojektionen subtrahiert werden. Zusätzlich wird das DataFrame `ZINS` angefügt, das für jedes Projektionsjahr den in diesem Jahr gültigen Zinssatz für die Anlageposition enthält.

Danach wird der Anteil der nicht-flüssigen Anlagen in der Vergangenheit bestimmt:

```
    # Anteil des Fonds, der nicht flüssige Mittel oder Anlagen ist, berechnen (für
    ↪  Schätzung von Zinsertrag unten)
    EO_BILANZ <- EO_BILANZ |>
        mutate(ant_ausgaben_nichtanlagen = (lag(kap_abr) -
        ↪  lag(fluss_mittel))/aus_total_ink_vk) |> # Annahme: Vor Jahresende wird dem
        ↪  Fonds der Betrag entnommen, welcher für die Deckung der laufenden Ausgaben
        ↪  des kommenden Jahres benötigt wird (wird laut Brief von Compenswiss an
        ↪  Bundesrat vom Dezember 2024 so gemacht).
        mutate(
            ant_ausgaben_nichtanlagen = if_else(
                jahr > PARAM_GLOBAL$jahr_abr,
                mean(
                    ant_ausgaben_nichtanlagen[
                        between(jahr, PARAM_GLOBAL$jahr_abr-9, PARAM_GLOBAL$jahr_abr) #
                        ↪  Anteil für zukünftige Jahre als Durchschnittlicher Anteil
                        ↪  seit 2023
                    ],
                    na.rm = TRUE
                ),
                ant_ausgaben_nichtanlagen
            )
```

```
        )
```

Um die Konstistenz mit vergangenen Abrechnungen zu bewahren (in welchen keine Rechnungsabgrenzung gemacht wurde), wird die passive Rechnungsabgrenzung für die Berechnung des Anteils zum Fondsvermögen hinzugefügt, bevor die flüssige Mittel und Anlagen vom Fondsvermögen abgezogen werden. Dies hat auch eine logische Begründung: Rechnungsabgrenzung ist rein buchhalterisch, und hat kein direkten Einfluss auf die Geldflüsse, welche zur Bestimmung des Anteils des Fonds, welcher nicht angelegt werden kann (respektive in flüssigen Mitteln gehalten werden kann). Daher wird die Rechungsbagrenzung bei der Abschätzung des Anteils der Ausgaben, die nicht angelegt sind, abgezogen. Mit letzten `mutate`-Befehl des Codeblocks wird der mittlere Anteil der Ausgaben, die nicht-Anlagen sind, der vergangenen 10 Jahre für die Jahre der Zukunft eingefügt (was weiter unten zur Berechnung der flüssigen Mittel und Anlagen genutzt wrid).

Als nächstes folgt eine `for`-Schleife, welche zum Ziel hat, anhand der jeweiligen Erträge und Aufwände in einem Jahr durch rekursive Fortschreibung das Umlageergebenis, den Zinsertrag, sowie den Stand des EO-Kapital (Fondsstand) zu berechnen. Zur Erläuterung wird die Schleife nachfolgend Schritt-für-Schritt beschrieben. Als erstes wird das Anlageergebnis berechnet, also die in einem Projektionsjahr erwartete Anlagerendite sowie das resultierende Betriebsergebnis:

```
    ## Rekursive Berechnung des Zinsertrags und des Fondsstands
    for(l_j in (PARAM_GLOBAL$jahr_abr + 1):PARAM_GLOBAL$jahr_ende) {

        EO_BILANZ <- EO_BILANZ %>%
            mutate(
                # Berechne Kapitalertrag
                kap_etr = if_else(
                    jahr == l_j,
                    (lag(fluss_mittel) + umlage_eo / 2) * nominal_zins_fonds,
                    kap_etr
                ),
                erg_betr = if_else(
                    jahr == l_j,
                    umlage_eo + kap_etr,
                    erg_betr
                ),
```

Wir nehmen an, dass die flüssigen Mittel und Anlagen sowie die hälfte des Umlageergebnis verzinst werden. Damit wird dann das Bertriebergebnis ermittelt.

In einem letzten Schritt wird nun der Stand des EO-Fonds sowie der Stand der flüssigen Mittel und Anlagen ermittelt:

```
                kap_abr = if_else(
                    jahr == l_j,
                    lag(kap_abr) + erg_betr,
                    kap_abr
                ),
                fluss_mittel = case_when(
                    jahr == l_j & jahr != PARAM_GLOBAL$jahr_ende ~ kap_abr -
↪   ant_ausgaben_nichtanlagen * lead(aus_total_ink_vk),
                    jahr == l_j & jahr == PARAM_GLOBAL$jahr_ende ~ kap_abr -
↪   ant_ausgaben_nichtanlagen * aus_total_ink_vk, # diesjährige Ausgaben verwenden im
↪   letzten Projektionsjahr
                    TRUE ~ fluss_mittel
                )
            )
    }
```

Für die Berechnung der flüssigen Mittel und Anlagen wird angenommen, dass die Compenswiss einen Teil der erwarteten Ausgaben vom nächsten Jahr für die Deckung der laufenden Ausgaben zurücklegt und nicht investiert. Dies ist das vorgehen von Compenswiss, gemäss dem Brief der Compenswiss an den Bundesrat vom Dezember 2024.

Zum Schluss wird die resultierende EO-Bilanz an das Finanzperspektivenmodell (respektive an das Modul `wrap_eo_ergebnisse`) zurückgegeben:

```
return(EO_BILANZ = EO_BILANZ)
```

4.11 Modul mod_eo_postprocessing.r

Dokumentation zuletzt aktualisiert am 17.09.2025

Das Modul `mod_eo_postprocessing` dient dazu, die im Finanzperspektivenmodell berechneten nominalen Werte in Werte zu konstanten Preisen umzurechnen und die Indikatoren zu berechnen, die im EO-Finanzhaushalt angezeigt werden (vgl. 4.11.1). Es wird wie folgt im `wrap_eo.R` aufgerufen:

```
# Postprocessing (Ergebnisse zu konstanten Preisen
# berechnen)
tl_postprocessing <- mod_eo_postprocessing(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EO_BILANZ = tl_ergebnisse$EO_BILANZ, EO_MASSNAHMEN = tl_massnahmen$EO_MASSNAHMEN,
    DISKONTFAKTOR = tl_vorb_berechn$DISKONTFAKTOR)
```

Es wird der folgende Code aufgerufen:

```
#--- Berechnen Indikatoren und Zusammenfügen zum Finanzhaushalt --------------
# Diskontierung und Berechnung der Beitragssätze
EO_FHH_real <- EO_BILANZ %>%
    diskontierung(DISKONTFAKTOR) %>%
    left_join(mod_eo_indices(.)$EO_INDICES, by = "jahr")
```

Die Funktion `diskontierung(DISKONTFAKTOR)` nimmt lediglich sämtliche Werte in EO_BILANZ ausser der Spalte `jahr`, und dividiert diese durch den in `DISKONTFAKTOR` angegebenen Deflator für das entsprechende Jahr:

```
diskontierung <- function(DATA, DISKONTFAKTOR, askontierung = FALSE) {
    colnames(DISKONTFAKTOR) <- c("jahr", "diskontfaktor")

    DATA_DISKONTIERT <- DATA %>%
        left_join(DISKONTFAKTOR, by = "jahr") %>%
        mutate(diskontfaktor = if (askontierung) {
            1/diskontfaktor
        } else {
            diskontfaktor
        }) %>%
        mutate_at(vars(-jahr, -diskontfaktor), list(~. * diskontfaktor)) %>%
        dplyr::select(-diskontfaktor)

    return(DATA_DISKONTIERT)
}
```

Die Indikatoren, die mit der Funktion `mod_eo_indices` berechnet wurden, werden zusätzlich hinzugefügt.

Die Indikatoren werden auch berechet für die nominale Version:

```
# Berechnung der Beitragssätze
EO_FHH_nom <- EO_BILANZ %>%
    left_join(mod_eo_indices(.)$EO_INDICES, by = "jahr")
```

Zuletzt wird noch der Bestand der flüssigen Mittel und der Anlagen des Ausgleichfonds in Prozent der Totalausgaben berechnet, da dieser Wert darf in der Regel nicht unter 50% sinken [31]:

```
# Berechnung der Fondsstand in Prozent der Ausgaben
EO_FHH_real <- EO_FHH_real %>%
    mutate(liquid_ausgaben = fluss_mittel/aus_total_ink_vk) %>%
    mutate_at(vars("liquid_ausgaben"), list(~round(. * 100, digits = 1)))

EO_FHH_nom <- EO_FHH_nom %>%
    mutate(liquid_ausgaben = fluss_mittel/aus_total_ink_vk) %>%
    mutate_at(vars("liquid_ausgaben"), list(~round(. * 100, digits = 1)))
```

Falls gewisse politische Massnahmen berechnet wurden, werden sie noch diskontiert. Die Ergebnisse werden dann zurück an das Modul `wrap_eo.R` zurückgegeben:

```
#--- Output -----------------------------------------------------------------
if (PARAM_GLOBAL$flag_param_eo_massn) {
    EO_MASSNAHMEN_nom <- EO_MASSNAHMEN
    EO_MASSNAHMEN_real <- EO_MASSNAHMEN |>
        diskontierung(DISKONTFAKTOR)
    tl_out <- tidylist(EO_MASSNAHMEN_real, EO_FHH_real, EO_MASSNAHMEN_nom,
        EO_FHH_nom)
} else {
    tl_out <- tidylist(EO_FHH_nom, EO_FHH_real)
}
```

4.11.1 Modul mod_eo_indices.R

Dokumentation zuletzt aktualisiert am 18.09.2025

Das Modul `mod_eo_indices` dient dazu, die Indikatoren, die im EO-Finanzhaushalt gezeigt werden zu berechnen. Die berechnete Indikatoren sind:

- Der Beitragssatz für Dienstleistende in Prozent der AHV-Lohnsumme,
- Der Beitragssatz für Mutterschatfsurlaub in Prozent der AHV-Lohnsumme,
- Der Beitragssatz für Vaterschaftsurlaub in Prozent der AHV-Lohnsumme,
- Der Beitragssatz für Betreuungsurlaub in Prozent der AHV-Lohnsumme,
- Der Beitragssatz für Totalausgaben (ohne Verwaltungskosten) in Prozent der AHV-Lohnsumme.

```
# --- Berechnen der Indikatoren
# ----------------------------------------------
colnames_eofhh <- names(EO_FHH)  # Variables in EO_FHH

EO_INDICES <- EO_FHH %>%
    mutate(bs_dienst = if ("eo_dienst_just" %in% colnames_eofhh) {
        eo_dienst_just/ahv_lohnsumme
    } else {
        NA_real_
    }, bs_msu = if ("eo_msu_just" %in% colnames_eofhh) {
```

[31] gemäss Art. 28 al.2 EOG

```
        eo_msu_just/ahv_lohnsumme
    } else {
        NA_real_
    }, bs_vsu = if ("eo_vsu_just" %in% colnames_eofhh) {
        eo_vsu_just/ahv_lohnsumme
    } else {
        NA_real_
    }, bs_bsu = if ("eo_bsu_just" %in% colnames_eofhh) {
        eo_bsu_just/ahv_lohnsumme
    } else {
        NA_real_
    }, bs_adoption = if ("eo_adoption_just" %in% colnames_eofhh) {
        eo_adoption_just/ahv_lohnsumme
    } else {
        NA_real_
    })

EO_INDICES_TOT <- EO_INDICES |>
    select(starts_with("bs")) |>
    mutate(across(everything(), ~replace_na(., 0))) |>
    # 3. Calculer la somme de toutes les colonnes par ligne
mutate(bs_total = rowSums(across(everything())), jahr = EO_INDICES$jahr) |>
    select(jahr, bs_total) |>
    right_join(EO_INDICES, by = "jahr")
```

Die Beitragssätze pro Urlaubstyp werden berechnet, indem man für jedes Jahr die relevanten Ausgaben durch die Lohnsumme teilt. Der Beitragssatz für die Totalausgaben entspricht einfach der Summe der Beitragssätze pro Urlaubstyp.

Schlussendlich werden noch alle Beitragssätze gerundet und als Prozent dargestellt:

```
#--- Darstellung in Prozent -------------------------------------------------

EO_INDICES <- EO_INDICES_TOT %>%
    # Scale and round all columns matching 'bs_'
mutate(across(matches("bs_"), ~round(.x * 100, 2))) %>%
    # Keep only 'jahr' and the processed 'bs_' columns
select(jahr, starts_with("bs_"))
```

Das Dataframe mit den Indikatoren wird schlussendlich an das Modul `mod_eo_postprocessing` zurückgegeben:

```
#--- Output -----------------------------------------------------------------

return(list(EO_INDICES = EO_INDICES))
```