Finanzperspektiven der EL

Technische Dokumentation

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

26. August 2025

Inhaltsverzeichnis

1 Einleitung

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

Das Finanzperspektivenmodell der EL 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 EL berechnet, und ein Modul, das die Ausgaben der EL berechnet. Diese Module sind wiederum aufgeteilt in Untermodulen nach Einnahmen- und Ausgabenposten.

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 ein Modul zur Abschätzung der Kosten einer 13. EL-Rente).
- `doutput`: Programme zur optischen Aufbereitung des Outputs (bspw. die Finanzperspektiven-Übersichtstabellen, die auf der BSV-Website veröffentlicht werden).

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

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.

1.2.1 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 Rohndaten 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 November 2025) als Beispiel:

key	value
file_eckwerte	eckwerte.xlsx
sheet_eckwerte	eckwerte
name_espop	espop.Rdata

Wir sehen in der Tabelle die Angabe, in welcher Datei sich die volkswirtschaftlichen Eckwerte der ESTV befinden, und in welcher Datei sich die Wohnbevölkerungsbestände gemäss STATPOP respektive ESPOP befinden.

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` erlaubt es also, auf einen Blick nachzuvollziehen unter welchen Parameterannahmen das Finanzperspektivenmodell berechnet wird, und diese Parameterannahmen ohne Änderungen am Code anzupassen. 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 September 2024) als Beispiel:

key	value
jahr_abr	2024
jahr_modelldaten	2024
flag_param_massn	FALSE
years_zu_abgaenge	2019;2020;2021;2022;2023

Wir sehen in der Tabelle in der Zeile `jahr_abr` die Angabe des Jahres der letzten EL-Abrechnung, und in der Zeile `jahr_modelldaten` die Angabe des letzten verfügbaren Registerstandes für die Schätzung der Modellparameter. `flag_param_massn` gibt an, ob die Auswirkungen von Politikmassnahmen geschätzt werden sollen (siehe unten). In `years_zu_abgaenge` ist angegeben, welche Jahre für die Schätzung der Modellparameter Zugangsrate und Abgangsrate verwendet werden sollen (siehe auch Kapitel 4.7).

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 4.10 beschrieben). Diese werden in der Datei `PARAM_MASSNAHMEN_BASE.csv` spezifiziert. Eine Testversion dieser Datei sieht wie folgt aus:

key	value
aktivierte_massnahmen_int	el_test_massn_int
aktivierte_massnahmen_ext	test_massn_ext
mass_f_1	test_massn_ext
mass_f_2	el_test_massn_int

Wir sehen in der Tabelle in der ersten Zeile `aktivierte_massnahmen_int` die Politikmassnahmen, wessen Auswirkung im EL Finanzperspektivenmodell geschätzt werden soll. Die Zeile `aktivierte_massnahmen_ext` enthält die Massnahmen, für welche ein vorberechneter Vektor mit jährlichen Auswirkungend er Massnahmen eingelesen werden soll. Die Zeilen `mass_f_1` und `mass_f_` dienen lediglich der Strukturierung der Massnahmen bei der Darstellung in den Output-Dateien.

2 Beschrieb der verwendeten Input-Daten

Dokumentation zuletzt aktualisiert am 24.04.2025

In diesem Kapitel werden die Input-Daten beschrieben, auf welchen die Berechnungen des Finanzperspektivenmodell EL beruhen.

2.1 Daten zu Wohn- und Erwerbsbevölkerung, und Grenzgänger

Wir verwenden Daten zur ständigen Wohnbevölkerung nach Alter und Geschlecht aus der Statistik der Bevölkerung und der Haushalte (STATPOP)[2] des BFS. Um eine komplette Zeitreihe der historischen Entwicklungen zu erhalten ergänzen wir die STATPOP zudem mit Daten zur Synthesestatistik von Stand und Struktur der Bevölkerung (ESOPOP)[3] des BFS. Diese Daten erlauben uns, den IST-Bestand der Wohnbevölkerung nachzuvollziehen, und werden im Finanzperspektivenmodell unter anderem zur Abschätzung des Anteils der Wohnbevölkerung, die in einem Jahr neu eine Invalidenrente erhält, verwendet. Zusätzlich nutzen wir Daten zur Erwerbsbevölkerung nach Alter und Geschlecht (in Anzahl Personen und in Vollzeitäquivalenten). Diese Daten berechnen wir, anhand des Vorgehens des BFS, direkt aus den Rohdaten der Schweizerischen Arbeitskräfteerhebung (SAKE).[4] Ebenfalls nutzen wir die Daten zu den Grenzgängern nach Alter und Geschlecht aus der Grenzgängerstatistik.[5]

Unsere Projektionen für die EL zur AHV basieren auf den durch das BFS in den Wohnbevölkerungsszenarien[6].

2.2 Daten zur Lohn- und Preisentwicklung

Wir verwenden Daten zur historischen Lohn- und Preisentwicklung des BFS. Für die Lohnentwicklung basieren wir uns auf Daten zum nominalen Schweizerischen Lohnindex (SLI) mit Basis 1939=100[7]. Für die

[2] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/statpop.html
[3] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/espop.html
[4] https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/erhebungen/sake.html
[5] https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/erhebungen/ggs.html
[6] https://www.bfs.admin.ch/bfs/de/home/statistiken/bevoelkerung/erhebungen/szenarien.html
[7] https://www.bfs.admin.ch/bfs/de/home/statistiken/arbeit-erwerb/loehne-erwerbseinkommen-arbeitskosten/lohnindex.html

Preisentwicklung basieren wir uns auf den Landesindex der Konsumentenpreise (LIK) mit Basis 1977=100[8].[9]

Unsere Projektionen für die Lohn- und Preisentwicklung gemäss SLI respektive LIK basieren auf den durch die Eidgenössische Finanzverwaltung (EFV) projizierten Eckwerte für die Finanzplanung.[10] Zusätzlich zu den publizierten Eckwerten für die Finanzplanungsperiode (aktuelles Jahr und die kommenden 4 Jahre) stellt uns die EFV Projektionen für die SLI und LIK 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 bei der EFV unter https://www.efv.admin.ch/efv/de/home/finanzberichterstattung/daten/eckwerte-finanzplanung.html, respektive auf Anfrage direkt bei der EFV, erhältlich.

2.3 Daten aus dem EL-Register

2.3.1 Registre des prestations complémentaires (RPC)

Le registre des prestations complémentaires (RPC) contient les cas actifs des PC pour un mois donné, état du mois suivant. Il est basé sur les annonces faites mensuellement par les cantons à la Centrale de Compensation, qui gère la création du registre. Toutes les informations nécessaires au calcul d'un cas PC (différents types de revenus et de dépenses, fortune, structure du ménage, etc.) sont transmises par les organes PC et les pools, et une sélection de ces informations est faite pour construire le RPC.

La plausibilité des données communiquées par les organes PC et pools est contrôlée par le registre des PC. En fonction du résultat de cette vérification, les données sont intégrées dans le registre sans réserve, avec réserves ou ne sont pas enregistrées.

Le concept d'échange de données pour le registre des prestations complémentaires (RPC) détermine les informations que les organes d'exécution sont tenus de transmettre régulièrement au registre central des PC. Les directives relatives au registre des prestations complémentaires (D-RPC) définissent et précisent les données à fournir et abordent également divers aspects spécifiques en relation avec les transmissions de données.

2.3.2 Données du RPC pour le modèle financier

Les données individuelles du registre des prestations complémentaires à l'AVS et à l'AI sont agrégées par année, assurance, âge, sexe et lieu d'habitation (à domicile ou en home) pour le modèle financier. Les données par cas sont utilisées (el_faelle), et sont agrégées selon les caractéristiques de l'ayant droit (Ansprechperson). L'annexe A.1 présente le script SAS qui sert à préparer les données du RPC pour le modèle financier.

Le tableau ci-dessous présente la manière dont les variables sont définies dans le jeu de données.

Nom de la variable	Descriptif	Détails
annee	Année	
csg1	Sexe	1 = homme, 2 = femme, 9 = inconnu
lsa1	Âge	999 = âge inconnu
assurance	Assurance (AVS ou AI)	1 = AVS, 3 = AI
in_jahr	Nombre de cas	

[8] https://www.bfs.admin.ch/bfs/de/home/statistiken/preise/landesindex-konsumentenpreise/indexierung.assetdetail.32767557.html

[9] Die Verwendung der Basisjahre folgt den gesetztlichen Anforderungen für die Berechnung des Mischindexes für die Bestimmung der Minimalrenten. Grundsätzlich verwenden wir das Jahresmittel des LIK zur Bestimmung der Teuerung. Eine Ausnahme bildet die Berechnung der Minimalrentenentwicklung, bei welcher wir entsprechend der Praxis in der Vergangenheit bis ins Jahr 2017 den LIK Stand Dezmeber (respektive die jährliche Veränderung des LIK Stand Dezember) für die Bestimmung der für die Entwicklung der Minimalrente massgeblichen Teuerung nutzen, und ab 2017 das Jahresmittel des LIK, respektive die jährliche Verändungsrate des Jahresmittels des LIK.

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

Nom de la variable	Descriptif	Détails
mbop_exsi	Montants des PC pour les besoins vitaux	Total du montant des PC pour les cas vivant à domicile, part estimée des coûts pour les besoins vitaux pour les cas en home.
is_new_jahr	Nombre de nouveaux cas par rapport à l'année précédente	Un cas est compté comme nouveau si l'année précédente il n'était pas dans le système PC ou s'il était un cas PC d'une autre assurance.
mbop_exsi_neu	Montants des PC pour les besoins vitaux des nouveaux cas	Nouveaux cas selon définition de « is_new_jahr »
heim_pers	Nombre de cas vivant en home	Les personnes mariées vivant en home sont comptées comme des cas séparés (contrairement aux personnes mariées vivant à domicile).
heim_mehrkosten_mbop	Montants des PC pour les coûts supplémentaires liés au séjour en home.	Pour les cas vivant à domicile, le montant est nul. Pour les cas vivant en home, le montant correspond à la différence entre le montant total des PC et le montant pour les besoins vitaux (de la variable « mbop_exsi »)
heim_pers_neu	Nombre de nouveaux cas vivant en home par rapport à l'année précédente	Un cas est compté comme nouveau si l'année précédente il ne vivait pas en home, ou si l'année précédente il vivait en home mais a changé d'assurance.
heim_mehrkosten_mbop_neu	Montants des PC pour les coûts supplémentaires liés au séjour en home des nouveaux cas	Nouveaux cas selon définition de « heim_pers_neu »

2.4 Daten aus der EL Abrechnung

Les cantons livrent chaque année le montant des dépenses pour les PC selon les catégories suivantes : PC à l'AI et PC à l'AVS, PC périodiques et frais d'invalidité et de maladie. Il s'agit de données comptables.

Aus den Daten der EL Abrechnung verwenden wir insbesondere die Informationen über die Höhe der Ausgaben für Krankheits- und Behinderungskosten, sowie die Verwaltungskosten.

3 Module zur Aufbereitung der Input-Daten

In diesem Kapitel werden die Module beschrieben, mithilfe welcher die Input-Daten für das Finanzperspektivenmodell EL eingelesen und aufbereitet werden. Die Datenaufbereitung für die Finanzperspektivenmodelle sämtlicher durch das BSV modellierten Sozialversicherungen (AHV, EL, 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).[11]

3.1 Modul prepare_input.R

Dokumentation zuletzt aktualisiert am 24.04.2025

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

Das Hauptmodul zur Aufbereitung der Input-Daten für das Finanzperspektivenmodell der EL 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:

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

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

```
# 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$path_iv_go <- paste0(PARAM_INPUTS$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")
```

Es wird später für jede Sozialversicherung ein separater Ordner für die aufbereiteten Input-Daten erstellt. `path_iv_go` enthält beispielsweise den Pfad zum Ordner, in welchem Input-Daten für die Berechnungen der EL-Finanzperspektiven später abgespeichert werden sollen, und `path_iv_massnahmen` den Pfad zum Ordner, in welchem die Input-Daten für die Berechnung der Politikmassnahmen für EL AHV 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 overwrite == FALSE
    if (!overwrite && file.exists(path))
        stop(path, "already exists")
    if (!file.exists(path)) {
        dir.create(path, recursive = TRUE)
    }
    file.remove(list.files(path, full.names = 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 EL-spezifischen Rohdaten erläutert:

```
### EL: GELTENDE ORDNUNG ###

# EL Abrechnung
EL_ABRECHNUNG <- mod_input_el_abrechnung(PARAM_INPUTS = PARAM_INPUTS)

# EL Quote
EL_QUOTEN <- mod_input_el_quoten(PARAM_INPUTS = PARAM_INPUTS)

# EL Modelldaten
EL_MODELLDATEN <- mod_input_el_modelldaten(PARAM_INPUTS = PARAM_INPUTS)

# RENTENBESTAND IV (Als Input für 'Population at Risk' für
# EL zur IV)
RENTENBESTAND_IV <- read_delim(file.path(PARAM_INPUTS$path_el_go,
    "RENTENBESTAND_IV.csv"), delim = ";", show_col_types = FALSE,
    trim_ws = TRUE)


el_go <- c(list(EL_ABRECHNUNG = EL_ABRECHNUNG, EL_QUOTEN = EL_QUOTEN,
    EL_MODELLDATEN = EL_MODELLDATEN, RENTENBESTAND_IV = RENTENBESTAND_IV))

### EL: MASSNAHMEN ###

el_massnahmen <- mod_input_massnahmen_el(PARAM_INPUTS)
```

D.h. es werden nacheinander die verschiedenen für die Berechnung der EL-Finanzperspektiven relevanten Rohdaten eingelesen, wobei für jede Art von Rohdaten ein spezifisches Modul verwendet wird. Die EL-spezifischen Rohdaten für die Berechnung der EL-Finanzperspektiven nach geltender Ordnung werden danach in der Liste `el_go` zusammengefasst. Ebenso werden die EL-spezifischen Rohdaten für die Berechnung der Massnahmen in der Liste `el_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 EL sieht der betreffende Codeteil wie folgt aus:

```
tidylist_write(el_go, inp_path_el_go)
tidylist_write(el_massnahmen, inp_path_el_massnahmen)
```

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

3.2 Modul mod_input_bev_bestand.R

Dokumentation zuletzt aktualisiert am 08.07.2025

Bemerkung: Dieses Modul beinhaltet die Aufbereitung sämtlicher Bevölkerungsdaten, die in den Finanzperspektiven des BSV genutzt werden. Der Grund ist, dass dieses Modul auch von den Finanzperspektivenmodellen der AHV, IV und EO verwendet wird, welche zusätzliche Bevölkerungsdaten verwenden. Für das EL-Modell sind lediglich die Daten zur Wohnbevölkerung Ende Jahr (nachfolgende Variable `bevendejahr`) relevant. Sämtliche anderen nachfolgend diskutierten Bevölkerungsvariablen werden nicht verwendet.

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.[12]

[12]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.

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"
            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, 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.[13] 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 untescherschieden 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","bereinig
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 Versicherten 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 der angrenzenden Jahre (bspw. 2001 und 2003 für 2002) gebildet:

[13] Die Unterscheidung der Nationalitäten ist für die EL nicht relevant, wird aber hier trotzdem gemacht, da diese für die AHV relevant ist und die Datenaufbereitung für alle Sozialversicherungen gemeinsam gemacht wird. Für die EL werden dann später im Code Schweizer und Ausländer aggregiert, womit die Nationalitätsunterscheidung entfällt.

```
#---------------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 08.07.2025

Bemerkung: Dieses Modul beinhaltet die Aufbereitung sämtlicher Bevölkerungsdaten, die in den Finanzperspektiven des BSV genutzt werden. Der Grund ist, dass dieses Modul auch von den Finanzperspektivenmodellen der AHV, IV und EO verwendet wird, welche zusätzliche Bevölkerungsdaten verwenden. Für das EL-Modell sind lediglich die Daten zur Wohnbevölkerung Ende Jahr (nachfolgende Variable `bevendejahr`) relevant. Sämtliche anderen nachfolgend diskutierten Bevölkerungsvariablen werden nicht verwendet.

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)[14], sowie die Erwerbsbevölkerungsszenarien (SCENARIO_EPT)[15] 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.[16] 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 untescherschieden werden können. Zusätzlich werden in nachfolgendem Codeblock auch alle Alter über 120 aggregiert und der Altersklasse 120 zugerechnet.

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

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

[16] Die Unterscheidung der Nationalitäten ist für die EL nicht relevant, wird aber hier trotzdem gemacht, da diese für die AHV relevant ist und die Datenaufbereitung für alle Sozialversicherungen gemeinsam gemacht wird. Für die EL werden dann später im Code Schweizer und Ausländer aggregiert, womit die Nationalitätsunterscheidung entfällt.

```
    #---------------------------------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
        )

      # 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_eckwerte.R

Dokumentation zuletzt aktualisiert am 18.11.2024

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)
```

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

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

ECKWERTE <- read_excel(paste0(PARAM_INPUTS$path_allgemein_go,
    PARAM_INPUTS$file_eckwerte), sheet = PARAM_INPUTS$sheet_eckwerte,
    range = readxl::cell_cols(1:12))

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

return(ECKWERTE = ECKWERTE)
```

3.5 Modul mod_input_minimalrente.R

Dokumentation zuletzt aktualisiert am 14.01.2025

Das Modul `mod_input_minimalrente.R` liest die historische Zeitreihe zur Höhe der Minimalrente ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
MINIMALRENTE <- mod_input_minimalrente(PARAM_INPUTS = PARAM_INPUTS)
```

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

```
# Einlesen der Minimalrenten
# ------------------------------------------------#

MINIMALRENTE <- read_excel(paste0(PARAM_INPUTS$path_allgemein_go,
    PARAM_INPUTS$file_minimalrente), sheet = PARAM_INPUTS$sheet_minimalrente,
    range = readxl::cell_limits(c(11, 1), c(NA, NA))) %>%
    mutate(minimalrente = r_min/12) %>%
    select(jahr, minimalrente)

# Return tidylist mit Minimalrenten
# ---------------------------------------#

return(MINIMALRENTE = MINIMALRENTE)
```

3.6 Modulmod_input_el_abrechnung.R

Das Modul `Modulmod_input_el_abrechnung.R` liest die Abrechnungsdaten der EL ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
EL_ABRECHNUNG <- mod_input_el_abrechnung(PARAM_INPUTS = PARAM_INPUTS)
```

Als Erstes werden die Daten aus dem entsprechenden Excel-File eingelesen:

```
# Lies Datei ohne Spaltennamen ein
raw <- read_excel(file.path(PARAM_INPUTS$path_el_go, "sv_el_fin.xlsx"),
    sheet = "daten", col_names = FALSE)
```

Als nächstes werden die Daten aufbereitet:

```
    # Setze Spaltennamen aus Zeile 11
    colnames(raw) <- raw[11, ] |> as.character()

    # Entferne die ersten 11 Zeilen und wähle relevante Spalten
    EL_ABRECHNUNG <- raw |>
        slice(-(1:11)) |>  # Entferne Header und leere Zeilen
        select(jahr, kant_kz, aus_ahv:aus_tot, bund_ahv:verw_tot) |>  # Wähle gewünschte
        ↪  Spalten
        mutate(
            jahr = as.integer(jahr),  # Konvertiere 'jahr' in Integer
            kant_kz = as.character(kant_kz),  # Konvertiere 'kant_kz' in Character
            across(-c(jahr, kant_kz), as.numeric)  # Konvertiere restliche Spalten in
            ↪  numerisch
        ) %>%
        filter(kant_kz=="CH") %>%
        select(-kant_kz) %>%
        mutate(
            exis_ahv=bund_ahv*8/5,
            heim_ahv=kant_ahv-kk_ahv-bund_ahv*(3/5),
            exis_iv=bund_iv*8/5,
            heim_iv=kant_iv-kk_iv-bund_iv*(3/5)
        ) %>%
        filter(jahr>=2008)

    ## output
    return(EL_ABRECHNUNG = EL_ABRECHNUNG)
```

Hierzu werden zuerst die ersten 11 Zeilen des Excel-Files entfernt, da diese Beschriebe des Excel-Files enthalten und keine Daten. Danach werden die relevanten Spalten ausgewählt, und es wird sichergestellt, dass die Variablen richtig formatiert sind. Als nächstes wird mit `filter(kant_kz==CH")` sichergestellt, dass nur die gesamtschweizerischen Daten ausgewählt werden (das Abrechnungsfile enthält auch Daten nach Kantonen). Danach werden die Bundes- und Kantonsanteile gemäss den gesetztlichen Vorgaben berechnet. Zum Schluss werden die aufbereiteten Daten an `prepare_input.R` zurückgegeben.

3.7 Modul mod_input_el_quoten.R

Das Modul `mod_input_el_quoten.R` liest die Grundlagedaten zur Berechnung der EL-Quoten nach Versicherung ein, und berechnet die EL-Quoten. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
EL_QUOTEN <- mod_input_el_quoten(PARAM_INPUTS = PARAM_INPUTS)
```

Als Erstes werden die Daten aus dem entsprechenden Excel-File eingelesen:

```
# Lies Datei ohne Spaltennamen ein
raw <- read_excel(file.path(PARAM_INPUTS$path_el_go, "sv_el_quoten.xlsx"),
    sheet = "daten", col_names = FALSE)
```

Als nächstes werden die EL-Quoten nach Versicherung berechnet:

```
    # Setze Spaltennamen aus Zeile 11
    colnames(raw) <- raw[11, ] |> as.character()

    # Entferne die ersten 11 Zeilen und wähle relevante Spalten
    EL_QUOTEN <- raw |>
        slice(-(1:11)) |>  # Entferne Header und leere Zeilen
        select(jahr, class1:class2_t, bez, rent) |>  # Wähle gewünschte Spalten
        mutate(
            jahr = as.integer(jahr),
            bez = as.numeric(bez),
            rent = as.numeric(rent),
            across(-c(jahr, bez, rent), as.character)  # Konvertiere restliche Spalten in
            ↪  character
        ) %>%
        filter(class1=="Total") %>%
        mutate(
            vers=case_when(
                class2_t=="EL zur AV" | class2_t=="EL zur HV" ~ "AHV",
                class2_t=="EL zur IV" ~ "IV",
                TRUE ~ "Total"
            )
        ) %>%
        group_by(jahr, vers) %>%
        summarize(
            bez=sum(bez),
            rent=sum(rent)
        ) %>%
        mutate(
            el_quote=bez/rent
        ) %>%
        select(jahr, vers, el_quote)

    ## output
    return(EL_QUOTEN = EL_QUOTEN)
```

Hierzu werden zuerst die ersten 11 Zeilen des Excel-Files entfernt, da diese Beschriebe des Excel-Files enthalten und keine Daten. Danach werden die relevanten Spalten ausgewhält, und es wird sichergestellt, dass die Variablen richtig formatiert sind. Als nächstes werden die Daten zur Alters- und der Versicherung zusammengefasst (`class2_t==ËL zur AV" | class2_t==ËL zur HV" ~ ÄHV"`) und die für die El-Quote relevanten Daten zu der Anzahl EL-Bezüger (`bez`) und der Versichertenpopulation, welcher der Anzal der Rentenbeziehenden in der entsprechenden Versicherung AHV oder IV entspricht (`rent`) werden aggregiert (`group_by` und `summarize` Befehl). Zum Schluss werden die El-Quoten berechnet, und die aufbereiteten Daten werden an `prepare_input.R` zurückgegeben.

3.8 Modul mod_input_el_modelldaten.R

Das Modul `mod_input_el_modelldaten.R` liest die aufbereiteten Registerdaten zur Berechnung des EL-Modells ein. Es wird wie folgt in `prepare_input.R` aufgerufen:

```
EL_MODELLDATEN <- mod_input_el_modelldaten(PARAM_INPUTS = PARAM_INPUTS)
```

Das Modul besteht aus einem einzigen Codeblock, in welchem die aufbereiteten Registerdaten eingelesen und aufbereitet werden:

```
EL_MODELLDATEN <- read_delim(file.path(PARAM_INPUTS$path_el_go,
    "daten_tomaet_1998_2024_v2.csv"), delim = ";", show_col_types = FALSE,
    trim_ws = TRUE) %>%
    select(jahr = annee, sex = csg1, alt = lsa1, vers = assurance,
        exis_pers = in_jahr_Sum, exis_pers_zugaenge = is_new_jahr_Sum,
        exis_chf = mbop_exsi_Sum, exis_chf_zugaenge = mbop_exsi_neu_Sum,
        heim_pers = heim_pers_Sum, heim_pers_zugaenge = heim_pers_neu_Sum,
        heim_chf = heim_mehrkosten_mbop_Sum, heim_chf_zugaenge =
        ↪  heim_mehrkosten_mbop_neu_Sum) %>%
    filter(jahr >= 2008) %>%
    mutate(across(where(is.numeric), ~replace_na(.x, 0)), sex = case_when(sex ==
        1 ~ "m", sex == 2 ~ "f", TRUE ~ "unknown"), vers = case_when(vers ==
        1 | vers == 2 ~ "AHV", vers == 3 ~ "IV", TRUE ~ "unknown")) %>%
    group_by(jahr, alt, sex, vers) %>%
    summarize(exis_pers = sum(exis_pers, na.rm = TRUE), exis_pers_zugaenge =
    ↪  sum(exis_pers_zugaenge,
        na.rm = TRUE), exis_chf = sum(exis_chf, na.rm = TRUE),
        exis_chf_zugaenge = sum(exis_chf_zugaenge, na.rm = TRUE),
        heim_pers = sum(heim_pers, na.rm = TRUE), heim_pers_zugaenge =
        ↪  sum(heim_pers_zugaenge,
            na.rm = TRUE), heim_chf = sum(heim_chf, na.rm = TRUE),
        heim_chf_zugaenge = sum(heim_chf_zugaenge, na.rm = TRUE))
```

Zuerst werden die Variablennahmen durch Umbenennung von der Register-Terminologie auf die Terminologie des Finanzperspektivenmodells angepasst. Danach werden die Daten zur Alten- und Hinterlassenversicherung (`vers==1 | vers==2`) aggregiert, was mit dem `group_by` und dem nachfolgenden `summarize` Befehl geschieht.

Zum Schluss werden die aufbereiteten Daten an `prepare_input.R` zurückgegeben:

```
return(EL_MODELLDATEN = EL_MODELLDATEN)
```

4 Module zur Berechnung der Finanzperspektiven

In diesem Kapitel werden die Module beschrieben, mithilfe welcher das Finanzperspektivenmodell EL berechnet wird. Die hier beschriebenen Module lesen die in Kapitel 3 aufbereiteten Input-Daten sowie die Datei `PARAM_GLOBAL.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 veröffentlichten Tabellen zu den finanziellen Perspektiven der EL.[17]

[17] https://www.bsv.admin.ch/bsv/de/home/sozialversicherungen/ergaenzungsleistungen/finanzen.html

4.1 Modul run_el.R

Dokumentation zuletzt aktualisiert am 28.04.2025

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

```
run_el(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_output` ist der Pfad, wo die Resultate der Modellberechnungen schlussendlich abgelegt werden sollen.

Das Modul `run_iv` enthält die folgenden Elemente:

```
run_el <- function(path_param, path_inp, path_out) {
```

Zuerst werden mit dem Modul `mod_read_data` (vgl. Kapitel 4.2) der unter `path_param` abgelegte Parametercontainer sowie die unter `path_inp` abgelegten Input-Daten eingelesen. Mit `data_folders=c(ällgemein", ël")` wird angegeben, dass sowohl die Allgemeinen als auch die EL-spezifischen Input-Daten eingelesen werden sollen:

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

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

```
# Modul zum überprüfen von Parameterwerten und setzten von
# default-Parameterwerten:
tl_inp$PARAM_GLOBAL <- mod_el_param_global(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE = tl_inp$ECKWERTE, EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG,
    EL_MODELLDATEN = tl_inp$EL_MODELLDATEN)
```

Das Modul `mod_el_param_global` dient dazu, default-Parameterwerte zu setzten, falls diese nicht in `PARAM_GLOBAL` spezifiziert sind (vgl. Kapitel 4.3).

Als nächstes wird das Modul `wrap_el` (vgl. Kapitel 4.4) aufgerufen, in welchem die einzelnen Ausgaben- und Einnahmenpositionen des Finanzperspektivenmodells berechnet werden:

```
# Modellberechnungen durchführen
tl_out_el <- wrap_el(tl_inp)
```

Zum Abschluss werden die Output-Dateien aufbereitet und in den vorangehend spezifizierten Output-Ordner geschrieben. Dies geschieht mit dem Modul `mod_out_el.R`. Da dieses Modul lediglich berechnete Daten zusammenträgt und in Excel-Files schreibt, und selber keine Berechnungen vornimmt, ist es nicht Teil dieser Dokumentation:

```
path_out_identifier <- mod_out_el(path_param, path_out, tl_inp,
    tl_out_el)

path_out_identifier
```

4.2 Modul mod_read_data.R

Dokumentation zuletzt aktualisiert am 14.01.2025

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

```
tl_inp <- mod_read_data(path_param, path_inp, data_folders = c("allgemein",
    "el"))
```

Als Erstes werden in `mod_read_data.R` 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:

```
# Finde alle .csv-Dateien, die mit 'PARAM' beginnen
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 welchen die einzelnen Paramter-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_iv` zurückgeben:

```
return(tl_inp)
```

4.3 Modul mod_el_param_global.R

Dokumentation zuletzt aktualisiert am 24.04.2025

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

```
tl_inp$PARAM_GLOBAL <- mod_el_param_global(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE = tl_inp$ECKWERTE, EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG,
    EL_MODELLDATEN = tl_inp$EL_MODELLDATEN)
```

Das Modul setzt Parameterwerte optional fest, für welche grundsätzlich ein Standardwert ("Default") existiert, für welche es aber in gewissen Fällen (bspw. Zwecks Backtestings oder Reproduktion von Berechnungen in der Vergangenheit) möglich sein soll, diese in der Datei `PARAM_GLOBAL.csv` anders zu wählen. Hierbei erfolgt zuerst immer eine Prüfung, ob der entsprechende Parameterwert nicht schon in `tl_inp$PARAM_GLOBAL` vorhanden ist (was der Fall ist, wenn der entsprechende Parameter in `PARAM_GLOBAL.csv` spezifiziert wurde):

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

    # ECKWERTE
    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 id 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)
}

# EL-ABRECHNUNGSJAHR
# auf Jahr der letzten verfügbaren Abrechnung setzten, falls nicht anders angegeben
if(!"jahr_abr" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_abr <- max(EL_ABRECHNUNG$jahr)
}

# EL-MODELLDATENJAHR
# auf Jahr der letzten verfügbaren Modelldaten setzten, falls nicht anders angegeben
if(!"jahr_modelldaten" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_modelldaten <- max(EL_MODELLDATEN$jahr)
}

# JAHR RENTENREGISTER
# Jahr auf PARAM_GLOBAL$jahr_modelldaten setzten. Irrelevant, da nur zur Berechnung
↪  einer im EL-Modell nicht benutzten Variable in mod_rentenentwicklung genutzt.
if(!"jahr_rr" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_rr <- PARAM_GLOBAL$jahr_modelldaten
}

# PREISBASIS
# auf Abrechnungsjahr legen, falls nicht in PARAM_GLOBAL gesetzt
if(!"jahr_preisbasis" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_preisbasis <- PARAM_GLOBAL$jahr_abr
}

# JAHR ENDE
# letztes Projektionsjahr setzen, falls nicht in PARAM_GLOBAL gesetzt
if(!"jahr_ende" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$jahr_ende <- 2070
}
# Schattierung für .xls-FHH deaktivieren, falls nicht anders spezifiziert
if(!"year_shade" %in% names(PARAM_GLOBAL)) {
    PARAM_GLOBAL$year_shade <- FALSE
```

```
    }
    # Schattierung ab Projektionsjahr jahr_abr+11 in .xls-FHH, falls nicht anders
    ↪  spezifiziert
    if(!"year_after_abr_shade" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$year_after_abr_shade <- 10
    }
    # Schattierte Werte in .xls-FHH nicht runden, falls nicht anders spezifiziert
    if(!"round_shade" %in% names(PARAM_GLOBAL)) {
        PARAM_GLOBAL$round_shade <- FALSE
    }
```

Damit kann das ergänzte `PARAM_GLOBAL`-Dataframe an `run_el.R` zurückgegeben werden. Dies geschieht mit der folgenden abschliessenden Codezeile des Moduls:

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

4.4 Modul wrap_el.R

Dokumentation zuletzt aktualisiert am 25.11.2024

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

```
tl_out_el <- wrap_el(tl_inp)
```

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

```
tl_el_vorb_berechn <- wrap_vorb_berechn_el(tl_inp = tl_inp)
```

Das Modul `wrap_vorb_berechn_el` ist in Kapitel 4.5 beschrieben.

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

```
tl_el_hauptberechnung <- wrap_el_hauptberechnung(tl_inp = tl_inp,
    tl_el_vorb_berechn = tl_el_vorb_berechn)
```

Das Modul `wrap_el_hauptberechnung` ist in Kapitel 4.6 beschrieben.

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

```
tl_el_massnahmen <- wrap_el_massnahmen(tl_inp = tl_inp, tl_el_vorb_berechn =
↪  tl_el_vorb_berechn,
    tl_el_hauptberechnung = tl_el_hauptberechnung)
```

Das Modul `wrap_el_massnahmen` ist in Kapitel 4.10 beschrieben.

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

```
tl_el_bilanz <- wrap_el_bilanz(tl_inp = tl_inp, tl_el_vorb_berechn = tl_el_vorb_berechn,
    tl_el_hauptberechnung = tl_el_hauptberechnung, tl_el_massnahmen = tl_el_massnahmen)
```

Das Modul `wrap_el_bilanz` ist in Kapitel 4.11 beschrieben.

Somit können die Berechnungen an das Finanzperspektivenmodell (respektive das Modul `run_el.R`) zurückgegeben werden. Dies geschieht mit der folgenden abschliessenden Codezeile des Moduls:

```
c(tl_el_vorb_berechn, tl_el_hauptberechnung, tl_el_massnahmen,
    tl_el_bilanz)
```

4.5 Modul wrap_vorb_berechn_el.R

Dokumentation zuletzt aktualisiert am 24.04.2025

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

```
tl_el_vorb_berechn <- wrap_vorb_berechn_el(tl_inp = tl_inp)
```

In einem ersten Schritt werden die Bevölkerungsdaten eingelesen und aufbereitet (vgl. Kapitel 4.14):

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

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 4.15) 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 4.16) 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 die Entwicklung der Minimalrente berechnet:

```
# Berechnung Rentenentwicklung gemaess Rentenanpassungen

RENTENENTWICKLUNG <- mod_rentenentwicklung(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE_EXTENDED = tl_eckwerte$ECKWERTE_EXTENDED, MINIMALRENTE =
    ↪  tl_inp$MINIMALRENTE,
    PREISINDEX = tl_inp$PREISINDEX)
```

Hierbei wird im Modul `mod_rentenentwicklung` (vgl. Kapitel 4.17) zuerst die für die Berechnung der Minimalrente massgebliche Entwicklung des Mischindexes gemäss der Entwicklung des nominalen Schweizerischen Lohnindexes (SLI) und der Preisentwicklung berechnet, und danach die daraus folgende zukünftige Entwicklung der Minimalrente berechnet (vgl. Kapitel 4.17).

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

```
#------ Output ------------------------------------------------------------#
c(list(BEVOELKERUNG = BEVOELKERUNG, DISKONTFAKTOR = DISKONTFAKTOR,
    RENTENENTWICKLUNG = RENTENENTWICKLUNG))
```

4.6 Modul wrap_el_hauptberechnung.R

Dokumentation zuletzt aktualisiert am 24.04.2025

Das Modul zu den Hauptberechnungen wird in `wrap_el.R` wie folgt aufgerufen:

```
tl_el_hauptberechnung <- wrap_el_hauptberechnung(tl_inp = tl_inp,
    tl_el_vorb_berechn = tl_el_vorb_berechn)
```

Als Erstes wird das Modul zur Berechnung der Projizierten Ausgaben für die Existenzsicherung und die heimbedingten Mehrkosten aufgerufen (vgl. Kapitel 4.7):

```
tl_el_exis_heim <- mod_el_exis_heim(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_MODELLDATEN = tl_inp$EL_MODELLDATEN, EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG,
    EL_QUOTEN = tl_inp$EL_QUOTEN, RENTENBESTAND_IV = tl_inp$RENTENBESTAND_IV,
    BEVOELKERUNG = tl_el_vorb_berechn$BEVOELKERUNG, RENTENENTWICKLUNG =
    ↪  tl_el_vorb_berechn$RENTENENTWICKLUNG,
    DISKONTFAKTOR = tl_el_vorb_berechn$DISKONTFAKTOR)
```

Darauffolgend wird das Modul zur Berechnung der projizierten Krankheits- und Behinderungskosten aufgerufen (vgl. Kapitel 4.8):

```
tl_el_krank_behin <- mod_el_krank_behin(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_AHV_FIN = tl_el_exis_heim$EL_AHV_FIN, EL_IV_FIN = tl_el_exis_heim$EL_IV_FIN,
    EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG)
```

Danach wird das Modul zur Berechnung der projizierten Verwaltungskosten aufgerufen (vgl. Kapitel 4.9):

```
tl_el_verwaltung <- mod_el_verwaltung(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_AHV_FIN = tl_el_exis_heim$EL_AHV_FIN, EL_IV_FIN = tl_el_exis_heim$EL_IV_FIN,
    EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG)
```

Zum Schluss werden die Ausgabenprojektionen zur EL-Bilanz nach geltender Ordnung zusammengefügt, und es werden gemäss den gestzlichen Bestimmungen die Bundes- und Kantonsanteile an die periodisch EL berechnet. Danach werden die Projektionen an das Finanzperspektivenmodell (respektive das Modul `wrap_el.R`) zurückgegeben:

```
EL_BILANZ_GO <- tl_el_exis_heim$EL_AHV_FIN %>%
    left_join(tl_el_exis_heim$EL_IV_FIN) %>%
    left_join(tl_el_verwaltung$VERW_AHV) %>%
    left_join(tl_el_verwaltung$VERW_IV) %>%
    left_join(tl_el_krank_behin$KK_AHV) %>%
    left_join(tl_el_krank_behin$KK_IV) %>%
    mutate(bund_ahv = exis_ahv * 5/8, kant_ahv = heim_ahv + exis_ahv *
        3/8, bund_iv = exis_iv * 5/8, kant_iv = heim_iv + exis_iv *
        3/8)

c(list(EL_BILANZ_GO = EL_BILANZ_GO, EL_AHV = tl_el_exis_heim$EL_AHV,
    EL_IV = tl_el_exis_heim$EL_IV, EL_QUOTEN_PROJ = tl_el_exis_heim$EL_QUOTEN_PROJ))
```

4.7 Modul mod_el_exis_heim.R

Dokumentation zuletzt aktualisiert am 24.04.2025

Das Modul zu den Ausgaben für Existenzsicherung und heimbedingte Mehrkosten wird in `wrap_el_hauptberechnung.R` wie folgt aufgerufen:

```
tl_el_exis_heim <- mod_el_exis_heim(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_MODELLDATEN = tl_inp$EL_MODELLDATEN, EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG,
    EL_QUOTEN = tl_inp$EL_QUOTEN, RENTENBESTAND_IV = tl_inp$RENTENBESTAND_IV,
    BEVOELKERUNG = tl_el_vorb_berechn$BEVOELKERUNG, RENTENENTWICKLUNG =
    ↪  tl_el_vorb_berechn$RENTENENTWICKLUNG,
    DISKONTFAKTOR = tl_el_vorb_berechn$DISKONTFAKTOR)
```

Da wir im Modell reale Entwicklungen aus der Vergangenheit in die Zukunft fortschreiben wollen, werden zu Beginn sämtliche monetären Grössen deflationiert. Dazu verwenden wir den in mod_diskontfaktor.R (vgl. Kapitel 4.16) berechneten Deflator.

```
    MINIMALRENTE <- RENTENENTWICKLUNG |>
        left_join(DISKONTFAKTOR) |>
        mutate(
            minimalrente=minimalrente*diskontfaktor,
            dminrente=minimalrente/lag(minimalrente)-1
        ) |>
        select(jahr, dminrente)

    EL_ABRECHNUNG_REAL <- EL_ABRECHNUNG |>
        left_join(DISKONTFAKTOR) |>
        mutate(across(-jahr, ~ .x * diskontfaktor)) |>
        select(-diskontfaktor)

    EL_MODELLDATEN_REAL <- EL_MODELLDATEN |>
        filter(jahr<=PARAM_GLOBAL$jahr_modelldaten) |> # Nur Daten bis zum gewünschten
        ↪  Registerstand auswählen
        left_join(DISKONTFAKTOR) |>
```

```
        mutate(across(contains("_chf"), ~ .x * diskontfaktor)) |>
        select(-diskontfaktor)
```

Als nächstes extrahieren wir die Altersgruppen, welche ausserhalb des Altersbereich liegen, welcher vom AHV respektive vom IV-Modell abgedeckt ist. Die Idee ist, dass wir für diese Altersgruppen später die Ausgabenentwicklung separat fortschreiben. Das Ausschliessen der Altersgruppen nach den untenstehenden Kriterien hat keinen relevanten Einfluss auf die Resultate, da die Ausgaben für EL an die ausgeschlossenen Altersgruppen im Jahr 2024 lediglich zwischen 1% und 4% der Gesamtausgaben ausmachen, und dadurch keinen relevanten Effekt auf die projizierte Kostenentwicklung haben.

```
    OUT_OF_AGERANGE <- EL_MODELLDATEN_REAL |>
        filter((vers=="AHV" & ((sex=="f" & alt<62)  | (sex=="m" & alt<63) | alt>=100)) |
        ↪  (vers=="IV" & (alt<=17 | ((sex=="f" & alt>=64) | (sex=="m" & alt>=65))))) |>
        mutate(
            # Berechne Verhältnisse und verhindere Division durch 0
            exis_chf = if_else(exis_pers == 0, NA, exis_chf / exis_pers),
            exis_chf_zugaenge = if_else(exis_pers_zugaenge == 0, NA, exis_chf_zugaenge /
            ↪  exis_pers_zugaenge),
            heim_chf = if_else(heim_pers == 0, NA, heim_chf / heim_pers),
            heim_chf_zugaenge = if_else(heim_pers_zugaenge == 0, NA, heim_chf_zugaenge /
            ↪  heim_pers_zugaenge)
        )
```

Als nächstes werden die in `PARAM_GLOBAL` spezifizierten Jahre, über welche die Zu- und Abgangsraten, respektive das Wachstum der durchschnittlichen EL, geschätzt werden sollen.

```
# Jahre auswählen, für welche die Wachstumsrate der
# Durchschnitts-EL geschätzt werden soll
years_el_wachstum <- str_split(PARAM_GLOBAL$years_el_wachstum,
    ";") |>
    unlist() |>
    as.numeric()

# Jahre auswählen, für welche die Zugangs- und Abgangsraten
# in der EL geschätzt werden sollen
years_zu_abgaenge <- str_split(PARAM_GLOBAL$years_zu_abgaenge,
    ";") |>
    unlist() |>
    as.numeric()

# Prüfe, ob years_zu_abgaenge ein Jahr >= 2025 enthält
if (any(years_zu_abgaenge >= 2025)) {
    warning("Warnung: Die Zu- und Abgangsraten beinhalten ein Jahr nach Einführung der
    ↪  AHV21, respektive dem Inkrafttreten des höheren Rentenalters für Frauen. Dadurch
    ↪  können die Zu- und Abgangsraten für Frauen zwischen 62 und 65 verzerrt sein.
    ↪  Bitte Modellcode überprüfen und allenfalls Jahre nach 2025 bei der Schätzung der
    ↪  Zu- und Abgangsraten für Frauen ausschliessen (Dataframe FORTSCHREIBUNG_AHV sowie
    ↪  Projektionsloop).")
}
```

Die Überprüfung im letzten Codeblock dient dazu sicherzustellen, dass sofern Registerdaten ab dem Jahr 2025 für die Parameterschätzung verwendet werden, die Auswirkung auf die Parameter und die Projektion für

die Frauen überprüft wird. Derzeit ist das Modell so spezifiziert, dass der Effekt der Rentenaltererhöhung im Rahmen der AHV21 basierend auf der Annahme berücksichtigt wird, dass die Modellparameter ohne AHV21 geschätzt wurden.

Damit sind alle Daten vorbereitet für die Berechnung des EL-Modells zur Projektion der Ausgaben für Existenzsicherung und heimbedingte Mehrkosten. Die Modellierung ist identisch für das AHV-Modell und für das IV-Modell, mit Ausnahmender relevanten Versichertenpopulation und daher der Altersrange, welche vom Modell abgedeckt ist (IV: 18-65 Jahre, AHV: 62-99 Jahre).

Die nachfolgenden Erläuterungen anahand des Codes zur Berechnung des Modells für die EL zur AHV gelten daher analog für die Berechnung der EL zur IV.

In einem ersten Schritt wird die relevante Versichertenpopulation extrahiert:

```
BEVOELKERUNG_AHV <- BEVOELKERUNG |>
    select(jahr, sex, alt, bevendejahr) |>
    group_by(jahr, sex, alt) |>
    summarize(bevendejahr = sum(bevendejahr, na.rm = TRUE))
```

Für die AHV gehen wir davon aus, dass die Versichertenpopulation durch die Wohnbevölkerung gut approximiert wird, was auf Grund der AHV-Bezugsquote von nahezu 100% eine gute Approximation darstellen sollte. Wir extrahieren hier daher die Wohnbevölkerung nach Alter und Geschlecht aus dem `BEVOELERKUNG`-Dataframe.

Für die IV nehmen wir die (beobachtete und im IV-Finanzperspektivenmodell projizierte) Anzahl Beziehenden von IV-Renten als Proxy für die Versichertenpopulation:

```
    BEVOELKERUNG_IV <- RENTENBESTAND_IV |>
        select(jahr, sex, alt, bestand_personen) |>
        left_join(
            RENTENBESTAND_IV |>
                select(jahr, sex, alt, bestand_personen) |>
                filter(alt == 25) |>
                select(jahr, sex, rentenbestand_iv_25 = bestand_personen),
            by = c("jahr", "sex")
        ) |>
        mutate(bevendejahr = ifelse(alt >= 18 & alt <= 25, rentenbestand_iv_25,
        ↪  bestand_personen)) |> # Bestand an 25-jährigen für 18-24 jährige verwenden,
        ↪  da aufgrund verzögerungen bei Rentenzusprache die Gruppe zu klein
        select(-rentenbestand_iv_25)  |>
        select(jahr, sex, alt, bevendejahr)
```

Für die Alter 18 bis 25 Jahre ersetzten wir die Population der IV-Rentenbeziehenden durch Population der Rentenbeziehenden mit Alter 25 Jahre. Dies hat zwei Gründe: 1. Bezieht sich die Population der Rentenbeziehenden in `RENTENBESTAND_IV` auf die Rentenbeziehenden, die in einem gegebenen Jahr tatsächliche eine Rente ausbezahlt erhalten haben (und nicht auf das tatsächliche Rentenanspruchsdatum, dass im Schnitt 1-2 Jahre zurückliegt), wodurch in `RENTENBESTAND_IV` die Anzahl Personen mit Rentenanspruch insbesondere bei jungen Erwachsenen unterschätzt wird. 2. Besteht Anspruch auf EL auch für Taggeldbezüger mit Bezugsdauer über 6 Monaten. Diese sind nicht in `RENTENBESTAND_IV` enthalten, wobei lange Taggeldbezüge auf Grund von erstmaligen beruflichen Ausbildungen insbesondere bei jungen Erwachsenen verbreitet sind.

Nachfolgend wieder die Erläuterungen anhand des Modells für die EL zur AHV:

```
    EL_AHV <- EL_MODELLDATEN_REAL |>
        anti_join(OUT_OF_AGERANGE |> select(jahr, alt, sex, vers)) |>
```

```
        filter(vers=="AHV") |>
        mutate(
            exis_pers_zugaenge = ifelse((alt==62 & sex=="f") | (alt==63 & sex=="m"),
            ↪  exis_pers, exis_pers_zugaenge),
            exis_chf_zugaenge = ifelse((alt==62 & sex=="f") | (alt==63 & sex=="m"),
            ↪  exis_chf, exis_chf_zugaenge),
            heim_pers_zugaenge = ifelse((alt==62 & sex=="f") | (alt==63 & sex=="m"),
            ↪  heim_pers, heim_pers_zugaenge),
            heim_chf_zugaenge = ifelse((alt==62 & sex=="f") | (alt==63 & sex=="m"),
            ↪  heim_chf, heim_chf_zugaenge)
        )  |>
        mutate(
            # Berechne Verhältnisse und verhindere Division durch 0
            exis_chf = if_else(exis_pers == 0, NA, exis_chf / exis_pers),
            exis_chf_zugaenge = if_else(exis_pers_zugaenge == 0, NA, exis_chf_zugaenge /
            ↪  exis_pers_zugaenge),
            heim_chf = if_else(heim_pers == 0, NA, heim_chf / heim_pers),
            heim_chf_zugaenge = if_else(heim_pers_zugaenge == 0, NA, heim_chf_zugaenge /
            ↪  heim_pers_zugaenge)
        ) |>
        select(-vers)
```

Dieser Codeteil berechnet die Zugänge in die EL-zur AHV pro Jahr. Da das Modell für die EL zur AHV erst mit dem AHV-Vorbezugsalter von 62 Jahren für Frauen (Stand 2025) respektive 63 Jahren für Männer beginnt, wird für diese Alter die Variable mit der Anzahl Personen (`_pers_zugaenge`) respektive der Totalausgaben (`_chf_zugaenge`) gleichgesetzt mit dem Endbestand. Zwar sind im Sinne der Daten nicht alle Personen und Ausgaben „neu“ (beispielsweise weil jemand von EL zur IV zu EL zur AHV wechselt, oder von zu Hause ins Heim zieht), jedoch schon im Sinne des Modells, weshalb diese Anpassung für eine korrekte Identifikation der Modellparameter (Zugangsrate, Entwicklung der Durchschnittlichen EL der Zugänge) notwendig ist.

Im zweiten Teil des Codes werden die Frankenbeträge, welche als Total in der jeweiligen Alter-Geschlecht-Jahr Zelle enthalten sind, durch Division mit der Anzahl Personen in der jeweiligen Zelle in Frankenbeträgen pro Person umgewandelt.

Der nächste Codeblock dient dazu, Anhand der Bestände und der Zugänge die Abgänge aus der EL zu identifizieren, sowie die Veränderungen der durchschnittlichen EL über die Jahre zu berechnen:

```
EL_AHV <- EL_AHV |>
    filter(jahr != min(EL_AHV$jahr)) |>
    full_join(EL_AHV |>
        filter(jahr != max(EL_AHV$jahr)) |>
        mutate(alt = alt + 1, jahr = jahr + 1) |>
        rename(exis_pers_vj = exis_pers, exis_chf_vj = exis_chf,
            heim_pers_vj = heim_pers, heim_chf_vj = heim_chf) |>
        select(jahr, sex, alt, exis_pers_vj, exis_chf_vj, heim_pers_vj,
            heim_chf_vj)) |>
    left_join(EL_AHV |>
        mutate(jahr = jahr + 1) |>
        rename(exis_pers_zugaenge_vj = exis_pers_zugaenge, heim_pers_zugaenge_vj =
        ↪  heim_pers_zugaenge,
            exis_chf_zugaenge_vj = exis_chf_zugaenge, heim_chf_zugaenge_vj =
            ↪  heim_chf_zugaenge) |>
        select(jahr, sex, alt, exis_pers_zugaenge_vj, heim_pers_zugaenge_vj,
```

```
            exis_chf_zugaenge_vj, heim_chf_zugaenge_vj)) |>
    mutate_if(is.numeric, ~coalesce(., 0)) |>
    mutate(exis_abgaenge = exis_pers_vj + exis_pers_zugaenge -
        exis_pers, heim_abgaenge = heim_pers_vj + heim_pers_zugaenge -
        heim_pers) |>
    mutate(exis_chf_ohnezugaenge = (exis_pers * exis_chf - exis_pers_zugaenge *
        exis_chf_zugaenge)/(exis_pers - exis_pers_zugaenge),
        delta_exis_chf_ohnezugaenge = exis_chf_ohnezugaenge/exis_chf_vj -
            1, delta_exis_chf_zugaenge = exis_chf_zugaenge/exis_chf_zugaenge_vj -
            1, heim_chf_ohnezugaenge = (heim_pers * heim_chf -
            heim_pers_zugaenge * heim_chf_zugaenge)/(heim_pers -
            heim_pers_zugaenge), delta_heim_chf_ohnezugaenge =
↪   heim_chf_ohnezugaenge/heim_chf_vj -
            1, delta_heim_chf_zugaenge = heim_chf_zugaenge/heim_chf_zugaenge_vj -
            1) |>
    arrange(jahr, sex, alt)
```

Mit dem ersten full-join Befehl werden die Beziehenden sowie die durchschnittlichen Frankenbeträge der gleichen Personen (die zu diesem Zeitpunkt ein Jahr jünger waren, weshalb zu deren Alter 1 dazu gezählt wird) aus dem Vorjahr (`_vj`) angefügt. Der nachfolgende left-join fügt die Vorjahres-Zugänge gleichen Alters und deren durchschnittlichen Frankenbeträge an. Der `mutate_if`-Befehl ersetzt NA-Werte im Dataframe mit 0. Dies betrifft lediglich die 100-jährige, welche mit dem full_join-Befehl an das Dataframe angefügt wurden. Da das Modell mit 99 Jahren endet ist es gewollt, dass die Bestände der 100-jährigen = 0 sind (damit ergibt sich dann auch die Abgangsrate von 1 für die 99-jährigen). Im nachfolgenden Befehl werden die Abgänge berechnet. Die Berechnung folgt der Logik, dass der Bestand im aktuellen Jahr dem Bestand im Vorjahr zuzüglich der Zugänge abzüglich der Abgänge entsprechen muss, was umgeformt die im Code dargestellte Gleichung für die Abgänge ergibt. Zum Schluss werden die Wachstumsraten der durchschnittlichen EL des Bestandes (`_ohnezugaenge`) sowie der durchschnittlichen EL der Zugänge (`_zugaenge`) berechnet.

Mit Hilfe der vorangehend berechneten Variablen lassen sich die aggregierten Wachstumsraten der Durchschnitts-EL für den Bestand sowie die Zugänge berechnen:

```
AHV_AGG_GROWTH <- EL_AHV |>
    group_by(jahr) |>
    mutate(exis_pers_bestand = exis_pers - exis_pers_zugaenge,
        heim_pers_bestand = heim_pers - heim_pers_zugaenge) |>
    summarise(exis_chf_zugaenge = weighted.mean(exis_chf_zugaenge,
        exis_pers_zugaenge_vj, na.rm = TRUE), exis_chf_zugaenge_vj =
↪   weighted.mean(exis_chf_zugaenge_vj,
        exis_pers_zugaenge_vj, na.rm = TRUE), exis_chf_ohnezugaenge =
↪   weighted.mean(exis_chf_ohnezugaenge,
        exis_pers_bestand, na.rm = TRUE), exis_chf_ohnezugaenge_vj =
↪   weighted.mean(exis_chf_vj,
        exis_pers_bestand, na.rm = TRUE), heim_chf_zugaenge =
↪   weighted.mean(heim_chf_zugaenge,
        heim_pers_zugaenge_vj, na.rm = TRUE), heim_chf_zugaenge_vj =
↪   weighted.mean(heim_chf_zugaenge_vj,
        heim_pers_zugaenge_vj, na.rm = TRUE), heim_chf_ohnezugaenge =
↪   weighted.mean(heim_chf_ohnezugaenge,
        heim_pers_bestand, na.rm = TRUE), heim_chf_ohnezugaenge_vj =
↪   weighted.mean(heim_chf_vj,
        heim_pers_bestand, na.rm = TRUE)) |>
    mutate(delta_exis_chf_zugaenge_total = exis_chf_zugaenge/exis_chf_zugaenge_vj -
```

```
        1, delta_exis_chf_ohnezugaenge_total =
        ↪  exis_chf_ohnezugaenge/exis_chf_ohnezugaenge_vj -
        1, delta_heim_chf_zugaenge_total = heim_chf_zugaenge/heim_chf_zugaenge_vj -
        1, delta_heim_chf_ohnezugaenge_total =
        ↪  heim_chf_ohnezugaenge/heim_chf_ohnezugaenge_vj -
        1) |>
    select(jahr, delta_exis_chf_zugaenge_total, delta_exis_chf_ohnezugaenge_total,
        delta_heim_chf_zugaenge_total, delta_heim_chf_ohnezugaenge_total)
```

Hierzu werden zuerst die gewichteten Mittelwerte für die verschiedenen Frankenbeträge gebildet. Die Frankenbeträge bei den Zugängen werden im `summarise`-Befehl jeweils mit der Anzahl Zugänge in der Vorperiode gewichtet. Damit wird die Zunahme der Frankenbeträge bei konstanter Struktur der Zugänge gerechnet (im Sinne eines Laspeyres-Indexes). Dies ist so gewollt, da der Einfluss einer Veränderten Struktur (beispielsweise vermehrte Zugänge in höherem Alter mit höherer durchschnittlichen EL) im Modell durch die Fortschreibung nach Kohorten berücksichtigt wird, und daher sonst eine doppelte Berücksichtigung dieses Effektes erfolgen würde. Bei den Bestandes-Frankenbeträge wird nach dem Bestand ohne Zugänge gewichtet, also nach der Anzal Beziehenden in der Zelle, die sowohl in der Vorperiode als auch in der aktuellen Periode eine EL Bezogen haben (was wiederum mit einem Laspeyres-Index konsistent ist).

Mit Hilfe dieser aggregierten Wachstumsraten können dann die Regressionsmodelle zur Bestimmung der Wachstumsrate der Durchschnitts-EL geschätzt werden:

```
## Regressionsmodell zur Entwicklung der Durchschnitts-EL
## schätzen

AHV_MODELTRAINING <- AHV_AGG_GROWTH |>
    filter(jahr %in% years_el_wachstum) |>
    left_join(MINIMALRENTE)

# Fit the regression model
model_exis_zugaenge_AHV <- lm(delta_exis_chf_zugaenge_total ~
    dminrente, data = AHV_MODELTRAINING)
summary(model_exis_zugaenge_AHV)
model_heim_zugaenge_AHV <- lm(delta_heim_chf_zugaenge_total ~
    1, data = AHV_MODELTRAINING)
summary(model_heim_zugaenge_AHV)
model_exis_ohnezugaenge_AHV <- lm(delta_exis_chf_ohnezugaenge_total ~
    dminrente, data = AHV_MODELTRAINING)
summary(model_exis_ohnezugaenge_AHV)
model_heim_ohnezugaenge_AHV <- lm(delta_heim_chf_ohnezugaenge_total ~
    1, data = AHV_MODELTRAINING)
summary(model_heim_ohnezugaenge_AHV)
```

Konkret werden zuerst die gemäss `years_el_wachstum` ausgewählten Jahre aus dem `AHV_AGG_GROWTH`-Dataframe extrahiert. Danach werden die Regressionsmodelle geschätzt, wobei angenommen wird, dass die Höhe der durchschnittlichen EL für Existenzsicherung sich mit einer Konstante zuzüglich einem Faktor proportional zum Minimalrentenwachstum (`dminrente`) entwickelt. Dies ist damit begründet, dass die Höhe des Lebensbedarfs in der EL an den Mischindex, und daher die Minimalrente, gekoppelt ist, und wir für die Minimalrentenentwicklung über eine Projektion verfügen (abgeleitet aus der projizierten Lohn- und Preisentwicklung). Für die heimbedingten Mehrkosten beinhaltet das Modell lediglich eine Konstante. Diese konstante entpricht dann dem durchschnittlichen Wachstum der durchschnittlichen heimbedingten Mehrkosten über die Jahre in `years_el_wachstum`.[18] Anhand dieser Regressionskoeffizienten kann dann weiter

[18] Eine OLS-Regression (welche dem lm-Befehl zu Grunde liegt) auf eine konstante und keine weiteren Kovariaten ergibt für

unten das Wachstum der Durchschnitts-Frankenbeträge in die Zukunft projiziert werden.

Zum Schluss erfolgt der Codeteil zur Berechnung der Zugangs- und Abgangsraten für die Projektion der Anzahl Beziehenden:

```
EL_AHV <- EL_AHV |>
    left_join(BEVOELKERUNG_AHV |>
        mutate(jahr = jahr + 1, alt = alt + 1)) |>
    mutate(exis_zugangsrate = exis_pers_zugaenge/(bevendejahr -
        exis_pers_vj), exis_abgangsrate = case_when(exis_pers_vj ==
        0 ~ 0, TRUE ~ exis_abgaenge/exis_pers_vj), heim_zugangsrate =
        ↪  heim_pers_zugaenge/(bevendejahr -
        heim_pers_vj), heim_abgangsrate = case_when(heim_pers_vj ==
        0 ~ 0, TRUE ~ heim_abgaenge/heim_pers_vj)) |>
    select(-contains("_vj"), -bevendejahr) |>
    mutate(across(everything(), ~replace(., is.infinite(.), NA)))

# Add the predicted values as a new column
FORTSCHREIBUNG_AHV <- EL_AHV |>
    filter(jahr %in% years_zu_abgaenge) |>
    group_by(alt, sex) |>
    summarize(exis_zugangsrate = mean(exis_zugangsrate, na.rm = TRUE),
        exis_abgangsrate = mean(exis_abgangsrate, na.rm = TRUE),
        heim_zugangsrate = mean(heim_zugangsrate, na.rm = TRUE),
        heim_abgangsrate = mean(heim_abgangsrate, na.rm = TRUE)) |>
    select(alt, sex, exis_zugangsrate, exis_abgangsrate, heim_zugangsrate,
        heim_abgangsrate)
```

Im ersten Codeblock werden die Zu- und Abgangsraten bestimmt. Die Zugangsrate für Personen mit Alter=alt im Jahr t ist definiert als die Zugänge von Personen mit Alter=alt im Jahr t dividiert durch die Versichertenpopulation Ende Jahr=t-1 mit Alter=alt-1, abzüglich der Anzahl Personen mit Alter=alt-1 die schon im Jahr t-1 eine EL bezogen haben. Die Abgangsrate für Personen mit Alter=alt im Jahr t ist definiert als die Anzahl Abgänge von Personen mit Alter=alt im Jahr=t dividiert durch den Bestand Ende Jahr t-1 mit Alter=alt-1. Im zweiten Codeblock werden dann die Mittelwerte über die in `years_zu_abgaenge` gebildet, welche für die Projektion genutzt werden. Im Gegensatz zu den Frankenbeträgen oben nehmen wir bei den Zu- und Abgängen an, dass diese in der Zukunft konstant bleiben. Dies ist insbesondere der Tatsache geschuldet, dass es keinen klaren Trend gibt in den Zu- und Abgangsraten (während bei den Frankenbeträgen ein klarer Aufwärtstrend verzeichnet wird). Zusätzlich unterliegen die Zu- und Abgangsraten auf Grund deren detaillierten Darstellung nach Alter und Geschlecht einer hohen Volatilität, was die Schätzung eines Trends deutlich erschwert.

Im nächsten for-Loop wird die eigentliche Projektion sämtlicher Bestände und Durchschnitts-CHF durchgefürt. Der Loop wird nachfolgend Schritt-für-Schritt erklärt:

```
  for(lJahr in (PARAM_GLOBAL$jahr_modelldaten+1):PARAM_GLOBAL$jahr_ende) {

      EL_AHV_NEU <- EL_AHV |>
          filter(jahr == lJahr - 1,
                 alt<=max_ahv_alt-1
                 ) |>
          select(jahr, alt, sex, exis_pers, exis_chf, heim_pers, heim_chf) |>
```

die Konstante einen Schätzer, der dem arithmetischen Mittel der abhängigen Variable entspricht. Diese Implementation wird hier der einfacheren und transparenteren direkten Berechnung des arithmetischen Mittels bevorzugt, da es weiter unten erlaubt, die Fortschreibung für das Wachstum der Existenzsicherung und der heimbedingten Mehrkosten gleich zu gestalten.

```
            full_join(BEVOELKERUNG_AHV |>
                          filter((sex=="f" & alt>=61)  | (sex=="m" & alt>=62),
                          ↪  jahr==lJahr - 1)
            )  |>
            arrange(sex, alt) |>
            mutate(jahr=jahr+1,
                   alt=alt+1
            ) |>
            left_join(EL_AHV |> select(jahr, alt, sex, exis_chf_zugaenge,
            ↪  heim_chf_zugaenge) |> mutate(jahr=jahr+1))
```

Das neue Dataframe `EL_AHV_NEU` soll die Projektion für die Bestände und Durchschnitts-CHF im `lJahr` enthalten, was im Ersten Durchlauf des Loops dem Ersten Jahr nach dem letzten verfügbaren Registerjahr entspricht. Dazu wird zuerst der Bestand an Beziehenden und deren Durchschnitts-CHF aus dem Vorjahr (was im ersten Durchlauf des Loops dem letzten verfügbaren Registerjahr, und danach rekursiv jeweils der im vorangehenden Loopdurchlauf berechneten Projektion) ausgelesen. Danach wird mit einem full_join die Versichertenpopuation mit entsprechendem Alter am Ende des Jahres angefügt, und die Personen werden durch Erhöhung des Jahres und des Alters um 1 um 1 Jahr älter gemacht. Danach werden die Durchschnitts-EL der Zugänge aus dem Jahr lJahr+1 angefügt. Damit enthält `EL_AHV_NEU` eine erste rohe Schätzung der Anzahl Personen und deren Durchschnitts-EL im `lJahr`. Als nächstes wird die Schätzung so bearbeitet, dass Zu- und Abgänge, sowie die Wachstumsrate der Durchschnitts-EL über die Zeit berücksichtigt werden.

Dazu werden die Zu- und Abgangsraten aus `FORTSCHREIBUNG_AHV` angefügt. Bevor diese verwendet werden können, müssen sie um den Effekt der AHV-21 korrigiert werden. Dies geschieht in dem nachfolgenden Codeblock:

```
            left_join(FORTSCHREIBUNG_AHV |>
                          # AHV-21
                          # Idee: Zwischen 2025 und 2028 verschieben sich die Zugänge und
                          ↪  Abgänge bei den Frauen zwischen dem Alter 62 und 65 ein
                          ↪  Jahr nach hinten.
                          # Verschiebung nach hinten auch bei Alter 65 da sich zeigt,
                          ↪  dass auch mit Rentenalter + 1 viele Eintritte in die EL
                          ↪  erfolgen
                          # Möglicherweise wird EL-Antrag erst nach erreichen des
                          ↪  Rentenalters eingereicht, und daher sind viele Personen bei
                          ↪  EL-Eintritt schon
                          # ein Jahr in Rente.
                          left_join(FORTSCHREIBUNG_AHV |>
                                        filter(sex=="f" & alt>=62 & alt<=65) |>
                                        mutate(alt=alt+1) |>
                                        rename(
                                            exis_zugangsrate_ahv21=exis_zugangsrate,
                                            exis_abgangsrate_ahv21=exis_abgangsrate,
                                            heim_zugangsrate_ahv21=heim_zugangsrate,
                                            heim_abgangsrate_ahv21=heim_abgangsrate
                                        )
                          ) |>
                          mutate(
                              exis_zugangsrate = case_when(
                                  sex == "f" & alt == 62 & lJahr>=2025 ~ exis_zugangsrate
↪  * (1-min(1, (lJahr - 2024) / 4)),
                                  sex == "f" & (alt == 63 | alt == 64 | alt==65) &
↪  lJahr>=2025 ~ exis_zugangsrate * (1-min(1, (lJahr - 2024) / 4)) +
↪  exis_zugangsrate_ahv21 * min(1, (lJahr - 2025) / 4),
```

```
                                sex == "f" & alt == 66 & lJahr>=2025 ~ exis_zugangsrate
↪  + exis_zugangsrate_ahv21 * min(1, (lJahr - 2025) / 4),
                                TRUE ~ exis_zugangsrate
                            ),
                            exis_abgangsrate = case_when(
                                sex == "f" & (alt == 63 | alt == 64 | alt==65 |
↪  alt==66) & lJahr>=2025 ~ exis_abgangsrate * (1-min(1, (lJahr - 2025) / 4)) +
↪  exis_abgangsrate_ahv21 * min(1, (lJahr - 2025) / 4),
                                TRUE ~ exis_abgangsrate
                            ),
                            heim_zugangsrate = case_when(
                                sex == "f" & alt == 62 & lJahr>=2025 ~ heim_zugangsrate
↪  * (1-min(1, (lJahr - 2024) / 4)),
                                sex == "f" & (alt == 63 | alt == 64 | alt==65) &
↪  lJahr>=2025 ~ heim_zugangsrate * (1-min(1, (lJahr - 2024) / 4)) +
↪  heim_zugangsrate_ahv21 * min(1, (lJahr - 2025) / 4),
                                sex == "f" & alt == 66 & lJahr>=2025 ~ heim_zugangsrate
↪  + heim_zugangsrate_ahv21 * min(1, (lJahr - 2025) / 4),
                                TRUE ~ heim_zugangsrate
                            ),
                            heim_abgangsrate = case_when(
                                sex == "f" & (alt == 63 | alt == 64 | alt==65 |
↪  alt==66) & lJahr>=2025 ~ heim_abgangsrate * (1-min(1, (lJahr - 2025) / 4)) +
↪  heim_abgangsrate_ahv21 * min(1, (lJahr - 2025) / 4),
                                TRUE ~ heim_abgangsrate
                            )
                        )
            ) |>
            left_join(MINIMALRENTE) |>
            ungroup() |>  # Remove grouping before mutation
            mutate(delta_exis_chf_zugaenge = predict(model_exis_zugaenge_AHV, newdata =
            ↪  cur_data())) |>
            mutate(delta_exis_chf_ohnezugaenge = predict(model_exis_ohnezugaenge_AHV,
            ↪  newdata = cur_data())) |>
            mutate(delta_heim_chf_zugaenge = predict(model_heim_zugaenge_AHV, newdata =
            ↪  cur_data())) |>
            mutate(delta_heim_chf_ohnezugaenge = predict(model_heim_ohnezugaenge_AHV,
            ↪  newdata = cur_data()))
```

Der ganze Codeblock dient lediglich dazu, die Zugangs-und Abgangsraten für die Effekte der AHV-21 Reform zu korrigieren. Spezifisch nehmen wir an, dass die Zugangs- und Abgangsraten bei den Frauen sich mit der schrittweisen Erhöhung des Rentenalters zwischen 2025 und 2028 parallel schrittweise um ein Jahr nach hinten verschieben und so den Männern angleichen. Während dies bereits einen relativ umfangreichen Codeblock in Anspruch nimmt, so ist dies immer noch bewusst vereinfachend gehalten. So wird beispielsweise bewusst ausser Acht gelassen, dass für die sogenannten Übergangsjahrgänge bis 1969 immer noch ein Vorbezug ab 62 Jahren möglich ist. Diese Vereinfachung wird gemacht, da sie nur einen vernachlässigbaren Effekt auf die Projektion der Gesamtausgaben hat (via die Unterschätzung des Bestandes der 62-jährigen Frauen für die Jahre 2025-2031).

Im nachfolgenden Code wird die Entwicklung der durchschnitts-EL projiziert:

```
            left_join(MINIMALRENTE) |>
            ungroup() |>  # Remove grouping before mutation
```

```
            mutate(delta_exis_chf_zugaenge = predict(model_exis_zugaenge_AHV, newdata =
            ↪  cur_data())) |>
            mutate(delta_exis_chf_ohnezugaenge = predict(model_exis_ohnezugaenge_AHV,
            ↪  newdata = cur_data())) |>
            mutate(delta_heim_chf_zugaenge = predict(model_heim_zugaenge_AHV, newdata =
            ↪  cur_data())) |>
            mutate(delta_heim_chf_ohnezugaenge = predict(model_heim_ohnezugaenge_AHV,
            ↪  newdata = cur_data()))
```

Hierzu wird zuerst das Dataframe `MINIMALRENTE`, welches die jährlichen (reale) Veränderungsraten der Minimalrente enthält. Diese wird dann von den nachfolgenden predict()-Funktionen in Kombination mit den vorangehend geschätzten Modellparametern genutz, um eine Projektion für das Wachstum der Durchschnitts-ELs in der Existenzsicherung zu berechnen.

Im nachfolgenden Code werden die Zu- und Abgänge, und die daraus folgenden Bestände projiziert:

```
# converting NA to zero
EL_AHV_NEU[is.na(EL_AHV_NEU)] <- 0

EL_AHV_NEU <- EL_AHV_NEU |>
    mutate(exis_pers_zugaenge = exis_zugangsrate * (bevendejahr -
        exis_pers), exis_abgaenge = exis_abgangsrate * exis_pers,
        exis_chf_zugaenge = exis_chf_zugaenge * (1 + delta_exis_chf_zugaenge),
        exis_chf = ifelse(exis_pers + exis_pers_zugaenge - exis_abgaenge ==
            0, 0, ((exis_chf * (1 + delta_exis_chf_ohnezugaenge)) *
            (exis_pers - exis_abgaenge) + exis_chf_zugaenge *
            exis_pers_zugaenge)/(exis_pers + exis_pers_zugaenge -
            exis_abgaenge)), exis_pers = exis_pers + exis_pers_zugaenge -
            exis_abgaenge, heim_pers_zugaenge = heim_zugangsrate *
            (bevendejahr - heim_pers), heim_abgaenge = heim_abgangsrate *
            heim_pers, heim_chf_zugaenge = heim_chf_zugaenge *
            (1 + delta_heim_chf_zugaenge), heim_chf = ifelse(heim_pers +
            heim_pers_zugaenge - heim_abgaenge == 0, 0, ((heim_chf *
            (1 + delta_heim_chf_ohnezugaenge)) * (heim_pers -
            heim_abgaenge) + heim_chf_zugaenge * heim_pers_zugaenge)/(heim_pers +
            heim_pers_zugaenge - heim_abgaenge)), heim_pers = heim_pers +
            heim_pers_zugaenge - heim_abgaenge)
```

Als Erstes werden NA-Werte in `EL_AHV_NEU` durch 0 ersetzt. Dies dient dem Zweck, die Bestände, die vor der Berechnung im Nachfolgenden Codeblock die Interpretation der Vorjahresbestände der ein Jahr jüngeren haben, für die Altersgruppen am Anfang des Modellhorizonts, also 62 Jahre bei Frauen und 63 Jahre bei Männern, auf 0 zu setzten.

Im zweiten Codeblock werden dann zuerst die Zugänge in die Existenzsicherung `exis_pers_zugaenge` berechnet, welche sich aus der Zugangsrate multipliziert mit der ein Jahr jüngeren Versichertenpopulation `bevendejahr` am Ende des Vorjahres abzüglich der ein Jahr jüngeren Beziehenden am Ende des Vorjahres ergibt. Als nächstes werden die Abgänge aus der Existenzsicherung `exis_abgaenge` berechnet, welche sich aus dem Produkt der Abgangsrate und der ein Jahr jüngeren Beziehenden am Ende des Vorjahres ergeben. Als nächstes folgt die Projektion der durchschnittlichen EL der Zugänge in die Existenzsicherung `exis_chf_zugaenge`, welche sich aus dem Produkt der Existenzischerung der gleichaltrigen im Vorjahr und eins plus der vorangehend geschätzten Veränderungsrate der durchschnittlichen EL in der Existenzsicherung ergibt. Die etwas komplexere Berechnung für die durchschnittliche EL in der Existenzsicherung insgesamt (also Zugaene und ohne Zugaenge zusammen) `exis_chf` folgt als nächstes. Hierzu wird zuerst überprüft, ob

die jeweilige Alters-Geschlecht-Jahr Zelle überhaupt Personen enthält, und falls dies der Fall ist ergibt sich `exis_chf` aus der gewichteten Summe der durchschnittlichen EL des Bestandes ohne Zugänge `exis_chf *(1 + delta_exis_chf_ohnezugaenge)` und der durchschnittlichen EL der Zugänge `exis_chf_zugaenge`, wobei `(exis_pers-exis_abgaenge)` der Anzahl Personen im Bestand ohne Zugänge und daher dem Gewicht von `exis_chf *(1 + delta_exis_chf_ohnezugaenge)`, und `exis_pers_zugaenge` der Anzahl Zugänge und daher dem Gewicht von `exis_chf_zugaenge` entspricht. Zuletzt werden die Personen im Bestand berechnet `exis_pers`, welche sich aus der ein Jahr jüngeren Personen im Bestand am Ende des Vorjahres, zuzüglich der Zugänge, abzüglich der Abgänge, ergibt. Mit dieser Gleichung wechselt somit die Interpretation der Variable `exis_pers`, welche vorher den Bestand der ein Jahr jüngeren am Ende des Vorjahres darstellt, und nachher den Bestand ende diesen Jahres. Die danach folgenden Berechnungen für die `_heim` Variablen sind analog zu den hier erläuterten für die `_exis` Variablen.

Damit ist die Projektion für sämtliche Bestände und durchschnitts-ELs für das `lJahr` dieses Loopdurchlaufs abgeschlossen, und das Dataframe `EL_AHV_NEU` wird bereinigt und an `EL_AHV` angefügt.

```
        EL_AHV_NEU <- EL_AHV_NEU |>
            select(jahr, alt, sex, exis_pers, exis_chf, exis_pers_zugaenge,
            ↪  exis_chf_zugaenge, exis_abgaenge, heim_pers, heim_chf,
            ↪  heim_pers_zugaenge, heim_chf_zugaenge, heim_abgaenge)

        #------ Step 3b: an Ergebnistabelle anfügen ---------------------------#
        EL_AHV   <- EL_AHV %>%
            rbind(., EL_AHV_NEU)
    }
```

Nachdem der Loop die Projektionen für sämtliche Jahre des Projektionshorizonts erstellt hat, bleibt nur noch der Code zur Aggregation der projizierten Werte, der Berücksichtigung der Werte ausserhalb der vom Modell abgedeckten Alter, zur Justierung auf die letzte EL-Abrechnung, und zur Projektion der EL-Quote. Dieser wird nachfolgend erläutert.

Als erstes werden die projizierten Werte nach Jahr und Geschlecht gemittelt (durchschnitts-EL) respektive aggregiert (Beziehende):

```
EL_AHV_YEARLY <- EL_AHV |>
    group_by(jahr, sex) |>
    summarize(exis_chf = weighted.mean(exis_chf, exis_pers, na.rm = TRUE),
        heim_chf = weighted.mean(heim_chf, heim_pers, na.rm = TRUE),
        exis_pers = sum(exis_pers, na.rm = TRUE), heim_pers = sum(heim_pers,
            na.rm = TRUE))
```

Danach wir die Projektion für die Alter ausserhalb der vom Modell abgedeckten Alter erstellt:

```
    AHV_OUT_OF_AGERANGE <- OUT_OF_AGERANGE |>
        filter(vers=="AHV") |>
        group_by(jahr) |>
        summarize(
            outofagerange_exis_chf=sum(exis_chf*exis_pers, na.rm = TRUE),
            outofagerange_heim_chf=sum(heim_chf*heim_pers, na.rm = TRUE)
        ) |>
        full_join(MINIMALRENTE |> filter(jahr>=PARAM_GLOBAL$jahr_modelldaten+1) |>
        ↪  mutate(dminrente=cumprod(1+dminrente))) |>
        mutate(
            # Ersetze outofagerange_exis_chf, falls jahr >=
            ↪  PARAM_GLOBAL$jahr_modelldaten+1
```

```
            outofagerange_exis_chf = ifelse(jahr >= PARAM_GLOBAL$jahr_modelldaten+1,
            ↪  mean(outofagerange_exis_chf[jahr %in%
            ↪  (PARAM_GLOBAL$jahr_modelldaten-2):PARAM_GLOBAL$jahr_modelldaten], na.rm =
            ↪  TRUE)*dminrente, outofagerange_exis_chf),
            outofagerange_heim_chf = ifelse(jahr >= PARAM_GLOBAL$jahr_modelldaten+1,
            ↪  mean(outofagerange_heim_chf[jahr %in%
            ↪  (PARAM_GLOBAL$jahr_modelldaten-2):PARAM_GLOBAL$jahr_modelldaten], na.rm =
            ↪  TRUE)*dminrente, outofagerange_heim_chf)
        ) |>
        select(-dminrente)
```

Dazu wird die Summe der an nicht vom Modell abgedeckten Alter ausbezahlten EL der letzten 3 Jahre gemittelt, und mit dem Wachstum der Minimalrente fortgeschrieben. Da diese lediglich 1-4% der Totalausgaben ausmachen, wird hier auf eine genauere Modellierung verzichtet.

Als nächstes werden die Ausgaben pro Jahr berechnet:

```
EL_AHV_FIN <- EL_AHV_YEARLY |>
    group_by(jahr) |>
    summarize(sum_exis_chf = sum(exis_chf * exis_pers, na.rm = TRUE),
        sum_heim_chf = sum(heim_chf * heim_pers, na.rm = TRUE)) |>
    left_join(AHV_OUT_OF_AGERANGE) |>
    mutate(sum_exis_chf = sum_exis_chf + outofagerange_exis_chf,
        sum_heim_chf = sum_heim_chf + outofagerange_heim_chf) |>
    select(-outofagerange_exis_chf, -outofagerange_heim_chf) |>
    ungroup() |>
    left_join(EL_ABRECHNUNG_REAL |>
        select(jahr, exis_ahv, heim_ahv)) |>
    mutate(sum_exis_chf_just = ifelse(jahr >= PARAM_GLOBAL$jahr_abr,
        sum_exis_chf * (exis_ahv[jahr == PARAM_GLOBAL$jahr_abr]/sum_exis_chf[jahr ==
            PARAM_GLOBAL$jahr_abr]), exis_ahv), sum_heim_chf_just = ifelse(jahr >=
        PARAM_GLOBAL$jahr_abr, sum_heim_chf * (heim_ahv[jahr ==
        PARAM_GLOBAL$jahr_abr]/sum_heim_chf[jahr == PARAM_GLOBAL$jahr_abr]),
        heim_ahv)) |>
    select(jahr, sum_exis_chf_just, sum_heim_chf_just) |>
    rename(exis_ahv = sum_exis_chf_just, heim_ahv = sum_heim_chf_just) |>
    filter(jahr > PARAM_GLOBAL$jahr_abr) |>
    bind_rows(EL_ABRECHNUNG_REAL |>
        filter(jahr <= PARAM_GLOBAL$jahr_abr) |>
        select(jahr, exis_ahv, heim_ahv)) |>
    arrange(jahr)
```

Hierzu werden die jährlichen Ausgaben aus `EL_AHV_YEARLY` mit den jährlichen Ausgaben aus `AHV_OUT_OF_AGERANGE` aufsummiert. Danach werden die projizierten Werte ab dem Abrechnungsjahr mit um die Abweichung der projizerten Werte von der EL-Abrechnung im letzten Abrechnungsjahr korrigiert (left_join(EL_ABRECHNUNG_REAL und nachfolgender mutate-Befehl)). Damit ist die finale Projektiond er Ausgaben für Existenzsicherung und die heimbdeingten Mehrkosten (nach geltender Ordnung) erstellt.

Die einzige im Modul verbleibende Berechnung ist die der EL-Quote. Diese wird in den nachfolgenden zwei Codeblöcken vorgenommen:

```r
    # EL-Quote gemäss Abrechnung, um danach modellberechnete Quoten zu justieren
    el_quote_ahv_ref <- EL_QUOTEN |>
        filter(jahr == max(EL_QUOTEN$jahr), vers == "AHV") |>
        pull(el_quote)

    # Berechne EL-Quote unter Verwendung des Referenzwerts
    EL_AHV_QUOTE <- EL_AHV |>
        filter(alt >= 65) |>
        left_join(BEVOELKERUNG_AHV, by = join_by(jahr, alt, sex)) |>
        group_by(jahr) |>
        summarise(
            # Summiere
            exis_pers = sum(exis_pers, na.rm = TRUE),
            bevendejahr = sum(bevendejahr, na.rm = TRUE),
            .groups = "drop"
        ) |>
        mutate(
            # Bestimme Referenzquote
            ratio = sum(exis_pers[jahr == max(EL_QUOTEN$jahr)]) / sum(bevendejahr[jahr ==
            ↪  max(EL_QUOTEN$jahr)]),
            # Berechne EL-Quote
            el_quote = (exis_pers / bevendejahr) * (el_quote_ahv_ref / ratio)
        ) |>
        select(jahr, el_quote) |>
        filter(jahr>max(EL_QUOTEN$jahr)) |>
        bind_rows(EL_QUOTEN |> filter(vers=="AHV" & jahr<=max(EL_QUOTEN$jahr)) |>
        ↪  select(jahr, el_quote)) |>
        rename(ahv=el_quote) |>
        arrange(jahr)
```

Hierzu wrid zuerst die EL-Quote gemäss Abrechnung im letzten Jahr für welches die Daten des EL-Registers verfügbar sind aus `EL_QUOTEN` ausgelesen. Im zweiten Codeblcok wird dann die Projektion der EL-Quote gemäss den in `EL_AHV` projizierten Bestände berechnet, wobei nur Alter ab 65 Jahren berücksichtigt werden (da für diese Alter die Annahme gut erfüllt ist dass Wohnbevölkerung $\subseteq$ AHV-Beziehende, nicht jedoch für die Vorbezugs-Alter vor 65). Zum Schluss wird die anahand von `EL_AHV` projizerte EL-Quote um die Abweichung von der tatsächlichen EL-Quote im letzten Jahr, für welches die EL-Quote gemäss Abrechnung verfügbar ist, korrigiert.[19]

Nachdem die analogen Berechnungen für die IV durchgeführt wurden, werden die Resultate an das Finanzperspektivenmodell (respektive das Modul `wrap_el_hauptberechnung.R`) zurückgegeben. Dies geschieht mit der folgenden abschliessenden Codezeile des Moduls:

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

[19]Die EL-Quoten gemäss der EL-Statistik werden berechnet, indem die Anzal der EL-Beziehenden durch die *tatsächliche* Anzal der AHV-Beziehenden geteilt wird. Wir verwenden die Wohnbevölkerung über 65 Jahre als Proxy für die AHV-Beziehenden, wodurch eine kleine Abweichung zwischen der EL-Quote gemäss Modelldaten und gemäss EL-Statistik entsteht. Unter der Annahme, dass die relative Abweichung zwischen der Modelldaten-Quote und der Quote gemäss EL-Statistik in der Zukunft konstant bleibt, ist dieses Vorgehen valide. Wir sehen keinen Anhaltspunkt, dass diese Annahme grob verletzt ist. Die Abweichungen zwischen Modelldaten-Quote und der Quote gemäss EL-Statistik beträgt 1.4 %-Punkte in 2024 für die AHV, und 5,5 %-Punkte für die IV, wobei die Modelldaten-Quoten in beiden Fällen tiefer liegen als die Quoten gemäss EL-Statistik. Bei der IV ist die Abweichung grösser, da Rentenbeziehende im Ausland auch in der im Modell verwendeten Versichertenpopulation enthalten sind (jedoch diese keinen Anspruch auf EL haben, und daher nicht zur in der EL-Statistik verwendeten Versichertenpopulation zählen).

```
return(list(EL_AHV_FIN = EL_AHV_FIN, EL_IV_FIN = EL_IV_FIN, EL_QUOTEN_PROJ =
↪  EL_QUOTEN_PROJ,
    EL_AHV = EL_AHV, EL_IV = EL_IV))
```

4.8 Modul mod_el_krank_behin.R

Dokumentation zuletzt aktualisiert am 27.04.2025

Das Modul zur Berechnung der Ausgaben für Krankheits- und Behinderungskosten wird in `wrap_el_hauptberechnung.R` wie folgt aufgerufen:

```
tl_el_krank_behin <- mod_el_krank_behin(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_AHV_FIN = tl_el_exis_heim$EL_AHV_FIN, EL_IV_FIN = tl_el_exis_heim$EL_IV_FIN,
    EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG, DISKONTFAKTOR =
    ↪  tl_el_vorb_berechn$DISKONTFAKTOR)
```

In einem ersten Schritt werden die Jahre ausgewählt, über welche das Wachstum der Krankheits- und Behinderungskosten geschätzt werden soll. Dafür werden die gleichen Jahre verwendet wie für das Wachstum der Durchschnitts-EL bei der Existenzsicherung und den heimbedingten Mehrkosten:

```
# Jahre auswählen, für welche die Wachstumsrate der
# Durchschnitts-EL geschätzt werden soll. Diese Jahre
# werden hier auch als Jahre für die Schätzung des
# Proportionalitätsfaktors verwendet.
years_el_wachstum <- str_split(PARAM_GLOBAL$years_el_wachstum,
    ";") |>
    unlist() |>
    as.numeric()
```

In einem nächsten Schritt werden die Daten der EL-Abrechnung sowie die projizierten Ausgaben für Existenzsicherung und heimbedingte Mehrkosten deflationiert:

```
# Konvertiere alle Eingabedaten in reale Werte
EL_ABRECHNUNG_REAL <- EL_ABRECHNUNG |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-jahr, ~.x * diskontfaktor)) |>
    select(-diskontfaktor)

EL_AHV_FIN_REAL <- EL_AHV_FIN |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-jahr, ~.x * diskontfaktor)) |>
    select(-diskontfaktor)

EL_IV_FIN_REAL <- EL_IV_FIN |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-jahr, ~.x * diskontfaktor)) |>
    select(-diskontfaktor)
```

Als nächstes werden die Proportionalitätsfaktoren für die Wachstumsraten der Krankheits- und Behinderungskosten in der AHV und in der IV geschätzt, wobei angenommen wird, dass diese Kosten proportional

zur Summe der projizierten Ausgaben für Existenzsicherung und heimbedingte Mehrkosten wachsen:[20]

```
    # Schätze Modelle auf realen Werten basierend auf Summe (exis + heim) ohne
    ↪  Achsenabschnitt
    model_ahv <- EL_ABRECHNUNG_REAL |>
        arrange(jahr) |>
        mutate(
            sum_exis_heim_ahv = exis_ahv + heim_ahv,
            sum_exis_heim_ahv_wachstum = sum_exis_heim_ahv / lag(sum_exis_heim_ahv) - 1,
            kk_ahv_wachstum = kk_ahv / lag(kk_ahv) - 1
        ) |>
        filter(jahr %in% years_el_wachstum) |>
        drop_na() |>
        lm(kk_ahv_wachstum ~ 0 + sum_exis_heim_ahv_wachstum, data = _)

    model_iv <- EL_ABRECHNUNG_REAL |>
        arrange(jahr) |>
        mutate(
            sum_exis_heim_iv = exis_iv + heim_iv,
            sum_exis_heim_iv_wachstum = sum_exis_heim_iv / lag(sum_exis_heim_iv) - 1,
            kk_iv_wachstum = kk_iv / lag(kk_iv) - 1
        ) |>
        filter(jahr %in% years_el_wachstum) |>
        drop_na() |>
        lm(kk_iv_wachstum ~ 0 + sum_exis_heim_iv_wachstum, data = _)
```

Als nächstes werden, unter Verwendung der Regressionskoeffizienten aus obigen Modellen in Kombination mit den projizierten Ausgaben für Existenzsicherung und heimbedingte Mehrkosten, die Projektionen für die Krankheits- und Behinderungskosten erstellt:

```
# Berechne kk_ahv (real)
KK_AHV_REAL <- EL_AHV_FIN_REAL |>
    filter(jahr >= PARAM_GLOBAL$jahr_abr, jahr <= PARAM_GLOBAL$jahr_ende) |>
    arrange(jahr) |>
    mutate(sum_exis_heim_ahv = exis_ahv + heim_ahv, sum_exis_heim_ahv_wachstum =
    ↪  sum_exis_heim_ahv/lag(sum_exis_heim_ahv) -
        1, kk_ahv_wachstum_pred = predict(model_ahv, newdata = cur_data()),
        kk_ahv_wachstum_pred = replace_na(kk_ahv_wachstum_pred,
            0), kk_ahv = accumulate(kk_ahv_wachstum_pred, ~.x *
            (1 + .y), .init = EL_ABRECHNUNG_REAL |>
            filter(jahr == PARAM_GLOBAL$jahr_abr) |>
            pull(kk_ahv))[-1]) |>
    select(jahr, kk_ahv) |>
    bind_rows(EL_ABRECHNUNG_REAL |>
        filter(jahr < PARAM_GLOBAL$jahr_abr) |>
```

[20] Für die Fortschreibung wurde das Wachstum der Krankheits- und Behinderungskosten (KK-AHV) auf das Wachstum der Gesamtausgaben für Existenzsicherung und heimbedingte Mehrkosten regressiert. Dabei kommt ein lineares Regressionsmodell ohne Achsenabschnitt zum Einsatz.

Im Unterschied zur einfachen Mittelwertbildung der periodenspezifischen Quotienten (z. B. $\text{Wachstum}_{\text{KK-AHV}}/\text{Wachstum}_{\text{Ausgaben}}$) ermöglicht die Regression eine robuste Schätzung des durchschnittlichen Zusammenhangs zwischen den beiden Wachstumsgrössen. Die Methode minimiert die quadrierten Abweichungen zwischen beobachtetem und modelliertem Wert und verhindert dadurch, dass einzelne Jahre mit extremen Werten (insbesondere bei sehr kleinen Nennern) das Ergebnis übermässig beeinflussen. Diese Überlegung liegt sowohl der Verwendung der Regression in diesem Fall, als auch bei der nachfolgenden Verwendung der Methode in diesem Modul, sowie im Modul mod_el_verwaltung.R, zugrunde.

```
        select(jahr, kk_ahv)) |>
    arrange(jahr)

# Berechne kk_iv (real)
KK_IV_REAL <- EL_IV_FIN_REAL |>
    filter(jahr >= PARAM_GLOBAL$jahr_abr, jahr <= PARAM_GLOBAL$jahr_ende) |>
    arrange(jahr) |>
    mutate(sum_exis_heim_iv = exis_iv + heim_iv, sum_exis_heim_iv_wachstum =
    ↪  sum_exis_heim_iv/lag(sum_exis_heim_iv) -
        1, kk_iv_wachstum_pred = predict(model_iv, newdata = cur_data()),
        kk_iv_wachstum_pred = replace_na(kk_iv_wachstum_pred,
            0), kk_iv = accumulate(kk_iv_wachstum_pred, ~.x *
            (1 + .y), .init = EL_ABRECHNUNG_REAL |>
            filter(jahr == PARAM_GLOBAL$jahr_abr) |>
            pull(kk_iv))[-1]) |>
    select(jahr, kk_iv) |>
    bind_rows(EL_ABRECHNUNG_REAL |>
        filter(jahr < PARAM_GLOBAL$jahr_abr) |>
        select(jahr, kk_iv)) |>
    arrange(jahr)
```

Zum Schluss werden die projizierten Krankheits- und Behinderungskosten von real in nominal konvertiert, und an das Modul `wrap_el_hauptberechnungen.R` zurückgegeben:

```
# Konvertiere kk_ahv und kk_iv zurück in nominale Werte
KK_AHV <- KK_AHV_REAL |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(kk_ahv = kk_ahv/diskontfaktor) |>
    select(jahr, kk_ahv)

KK_IV <- KK_IV_REAL |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(kk_iv = kk_iv/diskontfaktor) |>
    select(jahr, kk_iv)


# --- Output
# ------------------------------------------------------------------
return(list(KK_AHV = KK_AHV, KK_IV = KK_IV))
```

4.9 Modul mod_el_verwaltung.R

Dokumentation zuletzt aktualisiert am 21.07.2025

Das Modul zur Berechnung der Ausgaben für Verwaltungskosten wird in `wrap_el_hauptberechnung.R` wie folgt aufgerufen:

```
mod_el_verwaltung <- function(
        PARAM_GLOBAL,
        EL_AHV_FIN,
        EL_IV_FIN,
        EL_ABRECHNUNG,
```

```
        DISKONTFAKTOR
) {
```

In einem ersten Schritt werden die Jahre ausgewählt, über welche das Wachstum der Verwaltungskosten geschätzt werden soll. Dafür werden die gleichen Jahre verwendet wie für das Wachstum der Durchschnitts-EL bei der Existenzsicherung und den heimbedingten Mehrkosten:

```
# Jahre auswählen, für welche die Wachstumsrate der
# Durchschnitts-EL geschätzt werden soll. Diese Jahre
# werden hier auch als Jahre für die Schätzung des
# Proportionalitätsfaktors verwendet.
years_el_wachstum <- str_split(PARAM_GLOBAL$years_el_wachstum,
    ";") |>
    unlist() |>
    as.numeric()
```

In einem nächsten Schritt werden die Daten der EL-Abrechnung sowie die projizierten Ausgaben für Existenzsicherung und heimbedingte Mehrkosten deflationiert:

```
# Konvertiere alle Eingabedaten in reale Werte
EL_ABRECHNUNG_REAL <- EL_ABRECHNUNG |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-jahr, ~.x * diskontfaktor)) |>
    select(-diskontfaktor)

EL_AHV_FIN_REAL <- EL_AHV_FIN |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-jahr, ~.x * diskontfaktor)) |>
    select(-diskontfaktor)

EL_IV_FIN_REAL <- EL_IV_FIN |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-jahr, ~.x * diskontfaktor)) |>
    select(-diskontfaktor)
```

Als nächstes werden die Proportionalitätsfaktoren für die Wachstumsraten der Verwaltungskosten in der AHV und in der IV geschätzt, wobei angenommen wird, dass diese Kosten proportional zur Summe der projizierten Ausgaben für Existenzsicherung und heimbedingte Mehrkosten wachsen:

```
    model_verw_ahv <- EL_ABRECHNUNG_REAL |>
        arrange(jahr) |>
        mutate(
            sum_exis_heim_ahv = exis_ahv + heim_ahv,
            sum_exis_heim_ahv_wachstum = sum_exis_heim_ahv / lag(sum_exis_heim_ahv) - 1,
            verw_ahv_wachstum = verw_ahv / lag(verw_ahv) - 1
        ) |>
        filter(jahr %in% years_el_wachstum) |>
        drop_na() |>
        lm(verw_ahv_wachstum ~ 0 + sum_exis_heim_ahv_wachstum, data = _)

    model_verw_iv <- EL_ABRECHNUNG_REAL |>
```

```
        arrange(jahr) |>
        mutate(
            sum_exis_heim_iv = exis_iv + heim_iv,
            sum_exis_heim_iv_wachstum = sum_exis_heim_iv / lag(sum_exis_heim_iv) - 1,
            verw_iv_wachstum = verw_iv / lag(verw_iv) - 1
        ) |>
        filter(jahr %in% years_el_wachstum) |>
        drop_na() |>
        lm(verw_iv_wachstum ~ 0 + sum_exis_heim_iv_wachstum, data = _)
```

Als nächstes werden, unter Verwendung der Regressionskoeffizienten aus obigen Modellen in Kombination mit den projizierten Ausgaben für Existenzsicherung und heimbedingte Mehrkosten, die Projektionen für die Verwaltungskosten erstellt:

```
# Berechne VERW_AHV_REAL
VERW_AHV_REAL <- EL_AHV_FIN_REAL |>
    filter(jahr >= PARAM_GLOBAL$jahr_abr, jahr <= PARAM_GLOBAL$jahr_ende) |>
    arrange(jahr) |>
    mutate(sum_exis_heim_ahv = exis_ahv + heim_ahv, sum_exis_heim_ahv_wachstum =
    ↪  sum_exis_heim_ahv/lag(sum_exis_heim_ahv) -
        1, verw_ahv_wachstum_pred = predict(model_verw_ahv, newdata = cur_data()),
        verw_ahv_wachstum_pred = replace_na(verw_ahv_wachstum_pred,
            0), verw_ahv = accumulate(verw_ahv_wachstum_pred,
            ~.x * (1 + .y), .init = EL_ABRECHNUNG_REAL |>
                filter(jahr == PARAM_GLOBAL$jahr_abr) |>
                pull(verw_ahv))[-1]) |>
    select(jahr, verw_ahv) |>
    bind_rows(EL_ABRECHNUNG_REAL |>
        filter(jahr < PARAM_GLOBAL$jahr_abr) |>
        select(jahr, verw_ahv)) |>
    arrange(jahr)

# Berechne VERW_IV_REAL
VERW_IV_REAL <- EL_IV_FIN_REAL |>
    filter(jahr >= PARAM_GLOBAL$jahr_abr, jahr <= PARAM_GLOBAL$jahr_ende) |>
    arrange(jahr) |>
    mutate(sum_exis_heim_iv = exis_iv + heim_iv, sum_exis_heim_iv_wachstum =
    ↪  sum_exis_heim_iv/lag(sum_exis_heim_iv) -
        1, verw_iv_wachstum_pred = predict(model_verw_iv, newdata = cur_data()),
        verw_iv_wachstum_pred = replace_na(verw_iv_wachstum_pred,
            0), verw_iv = accumulate(verw_iv_wachstum_pred, ~.x *
            (1 + .y), .init = EL_ABRECHNUNG_REAL |>
            filter(jahr == PARAM_GLOBAL$jahr_abr) |>
            pull(verw_iv))[-1]) |>
    select(jahr, verw_iv) |>
    bind_rows(EL_ABRECHNUNG_REAL |>
        filter(jahr < PARAM_GLOBAL$jahr_abr) |>
        select(jahr, verw_iv)) |>
    arrange(jahr)
```

Zum Schluss werden die projizierten Verwaltungskosten von real in nominal konvertiert, und an das Modul `wrap_el_hauptberechnungen.R` zurückgegeben:

```r
# Konvertiere die Resultate zurück in nominale Werte
VERW_AHV <- VERW_AHV_REAL |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(verw_ahv = verw_ahv/diskontfaktor) |>
    select(jahr, verw_ahv)

VERW_IV <- VERW_IV_REAL |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(verw_iv = verw_iv/diskontfaktor) |>
    select(jahr, verw_iv)

# --- Output
# ------------------------------------------------------------------
return(list(VERW_AHV = VERW_AHV, VERW_IV = VERW_IV))
```

4.10 Modul wrap_el_massnahmen.R

Dokumentation zuletzt aktualisiert am 27.04.2025

Das Modul zu den Massnahmen dient dazu, die Auswirkungen von Politikmassnahmen auf die EL abzuschätzen (vgl. auch Kapitel 1.2.2). Es wird in `wrap_el.R` wie folgt aufgerufen:

```r
tl_el_massnahmen <- wrap_el_massnahmen(tl_inp = tl_inp, tl_el_vorb_berechn =
↪  tl_el_vorb_berechn,
    tl_el_hauptberechnung = tl_el_hauptberechnung)
```

Als erstes werden zwei leere Dataframes erstellt, in welchen später die Auswirkungen von Massnahmen gespeichert werden:

```r
# Liste und Dataframe für die Resultate initialisieren
tl_opt <- list()
DELTAS_MASSNAHMEN <- data.frame()
```

Danach folgt die Prüfung, ob die Auswirkungen von Massnahmen in die Finanzperspektiven-Berechnungen mit einbezogen werden sollen:

```r
if (tl_inp$PARAM_GLOBAL$flag_param_massn) {
```

Wenn dies nicht der Fall ist (also `tl_inp$PARAM_GLOBAL$flag_param_massn==FALSE` ist), wird das Modul abgeschlossen und es werden die leeren Dataframes zurückgegeben:

```r
#------ Output ------------------------------------------------------------#
c(tl_opt, list(DELTAS_MASSNAHMEN = DELTAS_MASSNAHMEN  # Füge DELTAS_MASSNAHMEN hinzu
))
```

Falls Massnahmen spezifiziert wurden (also `tl_inp$PARAM_GLOBAL$flag_param_massn==TRUE` ist), wird die Massnahmen-Inputliste `tl_inp_massn` erstellt, welche die Input-Dataframes sowie die Dataframes der Vor- und Hauptberechnungen enthält:

```
tl_inp_massn <- c(tl_inp, tl_el_vorb_berechn, tl_el_hauptberechnung)
```

Als nächstes wird überprüft, ob im Massnahmen-Parameterfile (vgl. Kapitel 1.2.2) Massnahmen spezifiziert sind, die innerhalb des Finanzperspektivenmodells berechnet werden sollen, also sogenannte „interne“ Massnahmen.

```
      ## INTERNE MASSNAHMEN BERECHNEN (falls solche ausgewählt sind)
      if (!is.na(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen_int)) {
```

Wenn dies der Fall ist wir eine Liste der Massnahmen ausgelesen, und in `massn_int` abgelegt:

```
massn_int <- separate_at_comma(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen_int)
```

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 (respektive 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 Massnahme angefügt. Dies dient dazu, später die Dataframes mit den Massnahmeneffekte den richtigen Massnahmen zuzuordnen. Danach wird die Liste `massnahme_effekte` zurückgegeben.

Die Funktion wird mit dem `lappy` Befehl so aufgerufen, wodurch für jedes Element der Liste `massn_int` (also für jede spezifizierte Massnahme, die innerhalb des EL-Finanzperspektivenmodells 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:

```r
# Apply the function and store the result in
# tl_massnahme_effekte
tl_massnahme_effekte <- do.call(c, lapply(massn_int, run_massnahmen,
    data = tl_inp_massn))
```

Am Ende dieses Kapitels wird anhand des Beispiels des Moduls `mod_opt_el_test_massn_int.R` erläutert, wie die Module für die Massnahmen-Berechnungen aufgebaut sind.

Als nächstes werden die Dataframes mit den Massnahmen-Effekte aus `tl_massnahme_effekte` ausgelesen und in die Liste `tl_DELTAS_MASSNAHMEN` gelegt:

```r
# 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 `DELTA_MASSNAHMEN` gelegt:

```r
# 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 EL_BILANZ_GO existieren
    if (!all(colnames(df) %in% colnames(tl_inp_massn$EL_BILANZ_GO))) {
        stop(sprintf("\033[31mFehler: Dataframe %s enthält Spalten, die nicht in
        ↪  EL_BILANZ_GO 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 EL-Abrechnung enthalten sind. Zweck dieser Überprüfung ist es, sicherzustellen, dass alle finanziellen Auswirkungen der Massnahmen auch tatsächlich einer 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 EL-Abrechnung enthalten) in die Spalte `konto` kopiert werden und die Massnahmen-Effekte im entsprechenden Jahr in die Spalte `wert`.

Zum Abschluss der Berechnungen für die internen Massnahmen werden auch noch die Dataframes mit den Auswirkungen der Massnahmen, die nicht die EL-Abrechnung betreffen, also die `OPT`-dataframes, ausgelesen (vgl. Kapitel 4.10.1):

```
# Auswahl der Dataframes aus tl_massnahme_effekte, die mit
# 'OPT' beginnen
tl_opt <- tl_massnahme_effekte[grepl("^OPT", names(tl_massnahme_effekte))]
```

Als nächstes wird überprüft, ob Massnahmen spezifiziert sind, deren Auswirkungen ausserhalb des EL-Finanzperspektivenmodells berechnet wurden (sogenannte „externe“ Massnahmen):

```
      ## EXTERNE MASSNAHMEN HINZUFÜGEN (falls solche ausgewählt sind)
      if (!is.na(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen_ext)) {
```

Ist dies der Fall, wird eine Liste mit den externen Massnahmen ausgelesen:

```
massn_ext <- separate_at_comma(tl_inp$PARAM_MASSNAHMEN_BASE$aktivierte_massnahmen_ext)
```

Als nächstes werden die ausgewählten Massnahmen aus `tl_inp_massn$MASSN_EXT` extrahiert, und es wird überprüft, ob die `konto`, die die Massnahmen betreffen sollen, auch in der EL-Abrechnung enthalten sind. Wenn dies der Fall ist werden die Massnahmen-Effekt `DELTAS_MASSNAHMEN` angefügt.

```
# Filtere MASSN_EXT nach aktiven Massnahmen
massn_ext_selected <- tl_inp_massn$MASSN_EXT |>
    filter(massnahme %in% massn_ext)

# Extrahiere eindeutige Kontonamen
massn_konten <- unique(massn_ext_selected$konto)

# Prüfe, ob alle Konten in EL_BILANZ_GO vorhanden sind
if (!all(massn_konten %in% colnames(tl_inp_massn$EL_BILANZ_GO))) {
    stop(sprintf("\033[31mFehler: Dataframe %s enthält Spalten, die nicht in EL_BILANZ_GO
    ↪  enthalten sind.\033[0m",
        "MASSN_EXT"))
}

# Zeilen an DELTAS_MASSNAHMEN anhängen
DELTAS_MASSNAHMEN <- bind_rows(DELTAS_MASSNAHMEN, massn_ext_selected)  # Daten zu
↪  DELTAS_MASSNAHMEN hinzufügen
```

Zum Schluss wird noch sichergestellt, dass `DELTA_MASSNAHMEN` keine NA-Werte enthält, und die Liste `tl_opt` sowie das Dataframe `DELTAS_MASSNAHMEN` werden an das Finanzperspektivenmodell, repsektive das Modul `wrap_el.R` zurückgegeben:

```
# Überprüfe auf NA-Werte in der Spalte 'wert'
if (any(is.na(DELTAS_MASSNAHMEN$wert))) {
    stop("Massnahme-Schätzung enthält NA-Wert. Massnahme-Module überprüfen.")
}

#------ Output --------------------------------------------------------#
c(tl_opt, list(DELTAS_MASSNAHMEN = DELTAS_MASSNAHMEN  # Füge DELTAS_MASSNAHMEN hinzu
))
```

4.10.1 mod_opt_el_test_massn_int.R

`mod_opt_el_test_massn_int.R` dient lediglich dazu aufzuzeigen, wie die Massnahmen-Module aufgebaut sind.

Die Massnahmen-Module werden mit einer Funktion aufgerufen, die als einziges Argument die Liste `tl_inp_massn` enthält:

```
mod_opt_el_test_massn_int <- function(tl_inp_massn) {
```

Danach werden die zu verwendenden Dataframes aus der Liste `tl_inp_massn` extrahiert, und es werden die Berechnungen durchgeführt. Dieser Teil folgt keiner zwingenden Struktur:

```
PARAM_GLOBAL <- tl_inp_massn$PARAM_GLOBAL
PARAM_EL_TEST_MASSN_INT <- tl_inp_massn$PARAM_EL_TEST_MASSN_INT

DELTA_TEST_INT <- tibble(jahr = 2023:2070, heim_iv = PARAM_EL_TEST_MASSN_INT$wert_mio *
    1e+06, kant_iv = PARAM_EL_TEST_MASSN_INT$wert_mio * 1e+06)

DELTA_TEST_INT <- DELTA_TEST_INT %>%
    mutate(heim_iv = case_when(jahr < PARAM_EL_TEST_MASSN_INT$jahr_beginn ~
        0, TRUE ~ heim_iv), kant_iv = case_when(jahr <
        ↪  PARAM_EL_TEST_MASSN_INT$jahr_beginn ~
        0, TRUE ~ kant_iv))

OPT_TEST_INT <- tibble(jahr = 2023:2070, betroffene_personen =
↪  PARAM_EL_TEST_MASSN_INT$wert_mio *
    5)

OPT_TEST_INT <- OPT_TEST_INT %>%
    mutate(betroffene_personen = case_when(jahr < PARAM_EL_TEST_MASSN_INT$jahr_beginn ~
        0, TRUE ~ betroffene_personen))
```

Zum Schluss werden die Resultate der Berechnungen zurückgegeben. Hierbei ist es wichtig, zwei Arten von Dataframes zu unterscheiden:

1. Die `DELTA`-Dataframes, welches die finanziellen Auswirkungen der Massnahmen pro Jahr enthält Ein `DELTA`-Dataframe enthält die Spalte `jahr` sowie eine oder mehrere Spalten mit Ausgaben- oder Einnahmepositionen aus der EL-Abrechnung. In vorliegendem Fall enthält `DELTA` die Spalten `jahr`, `heim_iv` und `kant_iv`. Generell bei allen Massnahmen ist wichtig, dass die ausgewiesenen Effekte auf die Ausgaben den Effekten auf die Einnahmen entsprechen.
2. Eines oder mehrere `OPT`-Dataframes, welches die nicht-finanziellen Auswirkungen der Massnahme pro Jahr enthält. Das `OPT`-Dataframe enthält die Spalte `jahr` sowie eine oder mehrere Spalten mit Auswirkungen. In vorliegendem Fall enhält `OPT` die Spalten `jahr` und `betroffene_personen`, also die Anzahl der von der Massnahme betroffenen Personens.

```
return(list(DELTA_TEST_INT = DELTA_TEST_INT, OPT_TEST_INT = OPT_TEST_INT))
```

4.11 Modul wrap_el_bilanz.R

Dokumentation zuletzt aktualisiert am 28.04.2025

Das Modul zur Berechnung der EL-Bilanz wird in wird in `wrap_el.R` wie folgt aufgerufen:

```
tl_el_bilanz <- wrap_el_bilanz(tl_inp = tl_inp, tl_el_vorb_berechn = tl_el_vorb_berechn,
    tl_el_hauptberechnung = tl_el_hauptberechnung, tl_el_massnahmen = tl_el_massnahmen)
```

Hier werden lediglich die Module zur Einbindung der Effekte der Massnahmen, sowie das Modul zur Berechnung des EL-Finanzhaushalts aufgerufen, welche in den folgenden zwei Kapiteln Beschrieben sind (vgl. Kapitel 4.12 und 4.13).

```
tl_el_massnahmeneffekte <- mod_el_massnahmeneffekte(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG, DISKONTFAKTOR =
    ↪  tl_el_vorb_berechn$DISKONTFAKTOR,
    EL_BILANZ_GO = tl_el_hauptberechnung$EL_BILANZ_GO, DELTAS_MASSNAHMEN =
    ↪  tl_el_massnahmen$DELTAS_MASSNAHMEN)

tl_el_fhh <- mod_el_fhh(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL, EL_BILANZ =
↪  tl_el_massnahmeneffekte$EL_BILANZ,
    EL_QUOTEN_PROJ = tl_el_hauptberechnung$EL_QUOTEN_PROJ, DISKONTFAKTOR =
    ↪  tl_el_vorb_berechn$DISKONTFAKTOR)


c(tl_el_massnahmeneffekte, tl_el_fhh)
```

4.12 Modul mod_el_massnahmeneffekte.R

Dokumentation zuletzt aktualisiert am 28.04.2025

Das Modul zur zur Einbindung der Effekte der Massnahmen in wird in `wrap_el_bilanz.R` wie folgt aufgerufen:

```
tl_el_massnahmeneffekte <- mod_el_massnahmeneffekte(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    EL_ABRECHNUNG = tl_inp$EL_ABRECHNUNG, DISKONTFAKTOR =
    ↪  tl_el_vorb_berechn$DISKONTFAKTOR,
    EL_BILANZ_GO = tl_el_hauptberechnung$EL_BILANZ_GO, DELTAS_MASSNAHMEN =
    ↪  tl_el_massnahmen$DELTAS_MASSNAHMEN)
```

Zur Vorbereitung wird die EL-Bilanz nach geltender Ordnung extrahiert, um später die Effekte der Massnahmen in den entsprechenden Zellen zu addieren. Zudem werden zwei leere Dataframes erstellt, in welchen die Effekte der Massnahmen gespeichert werden:

```
# Bilanz nach Geltender Ordnung in EL_BILANZ ablegen, um
# danach Massnahmen-Effekte zu addieren (falls es solche
# gibt)
EL_BILANZ <- EL_BILANZ_GO

# Leere dataframes mit Massnahme-Effekte erstellen (bleiben
# nur leer, falls keine Massnahmen ausgewählt sind)
DELTAS_EINN_AUSG_NOM <- data.frame()
DELTAS_EINN_AUSG_REAL <- data.frame()
```

Alle nachfolgenden Berechnungen werden nur durchgeführt, falls gemäss `PARAM_GLOBAL` die Effekte der Massnahmen berücksichtigt werden sollen, was mit der folgenden if-Bedingung sichergestellt wird:

```
    if (PARAM_GLOBAL$flag_param_massn && nrow(DELTAS_MASSNAHMEN) > 0) {
```

Falls die if-Bedingung `TRUE` ist, wird zuerst überprüft, ob die Massnahmen lediglich Konti betreffen, die auch Teil der EL-Bilanz sind, also ob keine falschen Konti in den Massnahmen angegeben wurden:

```
# Überprüfen, ob alle Werte von 'konto' in
# DELTAS_MASSNAHMEN in den Spaltennamen von EL_BILANZ_GO
# enthalten sind
konti_in_el_bilanz <- DELTAS_MASSNAHMEN$konto %in% colnames(EL_BILANZ_GO)

if (!all(konti_in_el_bilanz)) {
    stop("Massnahmen betreffen Konti, die nicht in der EL-Abrechnung enthalten sind.
    ↪  Bitte in den Massnahmen-Modulen die Effekte einer Spalte in EL_BILANZ_GO
    ↪  zuweisen.")
}
```

Ist dies der Fall werden die Effekte aller Massnahmen nach dem betroffenen Konto der EL-Bilanz aggregiert, und zum entsprechenden Konto der EL-Bilanz dazugezühlt:

```
# Aggregiere DELTAS_MASSNAHMEN
aggregated_DELTAS_MASSNAHMEN <- DELTAS_MASSNAHMEN |>
    group_by(konto, jahr) |>
    summarise(wert_sum = sum(wert), .groups = "drop")

# Iteriere über aggregierte Massnahmen
for (row in seq_len(nrow(aggregated_DELTAS_MASSNAHMEN))) {
    konto <- aggregated_DELTAS_MASSNAHMEN[row, "konto"][[1]]
    jahr <- aggregated_DELTAS_MASSNAHMEN[row, "jahr"][[1]]
    wert_sum <- aggregated_DELTAS_MASSNAHMEN[row, "wert_sum"][[1]]

    EL_BILANZ[EL_BILANZ$jahr == jahr, konto] <- EL_BILANZ[EL_BILANZ$jahr ==
        jahr, konto] + wert_sum
}
```

Als nächstes wird das Dataframe `DELTAS_EINN_AUSG_NOM` gebildet, in welchem die nach Massnahme aggregierten Effekte der Massnahmen auf die Einnahmen und die Ausgaben, abgelegt werden sollen. Diese Aggregation dient später zur Darstellung der Massnahmeeffekte nach Massnahmen, beispielsweise im Output-File `MASSNAHMEN_1.xlsx`:

```
## AUSGABEN- UND EINNAHMEEFFEKTE DER MASSNAHMEN PRO
## MASSNAHME AGGREGIEREN Erstelle leeres Dataframe
DELTAS_EINN_AUSG_NOM <- data.frame(jahr = unique(DELTAS_MASSNAHMEN$jahr))  #
↪  Initialisiere mit Jahr-Spalte
```

Nachfolgend werden die Effekte auf die Einnahmen und die Ausgaben separat aggregiert. Diese Aggregation wird separat durchgeführt, damit später bei der optischen Aufbereitung für die Ausgabe der Excel-Files zu den Massnahmeneffekte unterschieden werden kann, was Einnahme- und Ausgabeeffekte sind.

```
# Bearbeitung für Einnahmen
for (massnahme in unique(DELTAS_MASSNAHMEN$massnahme)) {
```

```
    # Filtere Konti, die Einnahmen in EL_BILANZ_GO sind
    valid_konti <- DELTAS_MASSNAHMEN %>%
        filter(massnahme == !!massnahme & konto %in% c("bund_ahv",
            "bund_iv", "kant_ahv", "kant_iv"))

    # Prüfe, ob es gültige Konti gibt
    if (nrow(valid_konti) > 0) {
        # Summiere über gültige Konti
        summarized <- valid_konti %>%
            group_by(jahr) %>%
            summarise(sum_einnahmen = sum(wert, na.rm = TRUE),
                .groups = "drop")

        # Füge die Spalte zu DELTAS_EINN_AUSG_NOM hinzu
        DELTAS_EINN_AUSG_NOM <- DELTAS_EINN_AUSG_NOM %>%
            left_join(summarized, by = "jahr") %>%
            rename_with(~paste0("einnahmen..", massnahme), "sum_einnahmen")
    }
}

# Bearbeitung für Ausgaben
for (massnahme in unique(DELTAS_MASSNAHMEN$massnahme)) {
    # Filtere Konti, die Ausgaben in EL_BILANZ_GO sind
    # (also alle, die nicht jahr oder Einnahmen sind)
    valid_konti <- DELTAS_MASSNAHMEN |>
        filter(massnahme == !!massnahme & !konto %in% c("jahr",
            "bund_ahv", "bund_iv", "kant_ahv", "kant_iv"))

    # Prüfe, ob es gültige Konti gibt
    if (nrow(valid_konti) > 0) {
        # Summiere über gültige Konti
        summarized <- valid_konti %>%
            group_by(jahr) %>%
            summarise(sum_ausgaben = sum(wert, na.rm = TRUE),
                .groups = "drop")

        # Füge die Spalte zu DELTAS_EINN_AUSG_NOM hinzu
        DELTAS_EINN_AUSG_NOM <- DELTAS_EINN_AUSG_NOM %>%
            left_join(summarized, by = "jahr") %>%
            rename_with(~paste0("ausgaben..", massnahme), "sum_ausgaben") %>%
            arrange(jahr)
    }
}
```

Es ist erkennbar, dass die Berechnungen für die Einnahme- und Ausgabeeffekte analog sind. Nachfolgende Erläuterungen daher lediglich am Beispiel der Einnahmeeffekte. Als erstes werden die Effekte ausgelesen, die Einnahmekonti der EL-Bilanz betreffen, und in der Liste `valid_konti` abgelegt. Danach werden im Falle, dass überhaut Einnahmekonti der EL-Bilanz betroffen sind (`nrow(valid_konti) > 0`) die Effekte pro Massnahme über die verschiedenen Einnahmekonti hinweg aggregiert. Zum Schluss werden die Effekte dem Dataframe `DELTAS_EINN_AUSG_NOM` zugewiesen, wobei mit der Umbenennung der Spalte `sum_ausgaben` in `paste0(äusgaben..", massnahme)` sichergestellt wird, dass die Massnahmeneffekte später der entsprechenden Massnahme zugewiesen werden können.

Zum Schluss werden die Massnahmen-Effekte auch in reale Werte umgerechnet, und die Resultate werden

an das Modul `wrap_el_bilanz.R` zurückgegeben:

```
    # Ersetze alle NA-Werte in DELTAS_EINN_AUSG_NOM durch 0
    DELTAS_EINN_AUSG_NOM[is.na(DELTAS_EINN_AUSG_NOM)] <- 0

    DELTAS_EINN_AUSG_REAL <- DELTAS_EINN_AUSG_NOM |>
        left_join(DISKONTFAKTOR, by = "jahr") |>
        mutate(across(
            -jahr,
            ~ .x * diskontfaktor
        )) |>
        select(-diskontfaktor)
    }


    #------ Output --------------------------------------------------------#
    return(
        list(
            EL_BILANZ = EL_BILANZ,
            DELTAS_EINN_AUSG_NOM = DELTAS_EINN_AUSG_NOM,
            DELTAS_EINN_AUSG_REAL = DELTAS_EINN_AUSG_REAL
        )
    )
```

4.13 Modul mod_el_fhh.R

Dokumentation zuletzt aktualisiert am 28.04.2025

Das Modul für die Aufbereitung der EL-Finanzhaushalte `mod_el_fhh.R` wir in `wrap_el_bilanz.R` wie folgt aufgerufen:

```
tl_el_fhh <- mod_el_fhh(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL, EL_BILANZ =
↪  tl_el_massnahmeneffekte$EL_BILANZ,
    EL_QUOTEN_PROJ = tl_el_hauptberechnung$EL_QUOTEN_PROJ, DISKONTFAKTOR =
    ↪  tl_el_vorb_berechn$DISKONTFAKTOR)
```

Der Zweck des Moduls ist, die Finanzhaushaltsdaten für den Finanzhausahlt der EL zur AHV und zur IV, sowie den gesamt-Finanzhaushalt der EL, aufzuteilen, und diese auch in reale Werte umzurechnen. Die in dieser Aufbereitung erstellten Dataframes sind die Grundlage für die Ausgabe-Excel `FH_EL.xlsx` und `FH_EL_NOM.xlsx`:

```
EL_ZU_AHV_BILANZ_NOM <- EL_BILANZ |>
    select(jahr, contains("_ahv")) |>
    left_join(EL_QUOTEN_PROJ |>
        select(jahr, ahv)) |>
    rename(el_quote = ahv) %>%
    mutate(aus_ahv = bund_ahv + kant_ahv, per_ahv = exis_ahv +
        heim_ahv, bundesanteil_ausgaben = bund_ahv/(exis_ahv +
        heim_ahv)) |>
    rename_with(~str_remove(.x, "_ahv"), .cols = -c(jahr, el_quote,
        bundesanteil_ausgaben))
```

```
EL_ZU_AHV_BILANZ <- EL_ZU_AHV_BILANZ_NOM |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-c(jahr, el_quote, bundesanteil_ausgaben),
        ~.x * diskontfaktor)) |>
    select(-diskontfaktor)

EL_ZU_IV_BILANZ_NOM <- EL_BILANZ |>
    select(jahr, contains("_iv")) |>
    left_join(EL_QUOTEN_PROJ |>
        select(jahr, iv)) |>
    rename(el_quote = iv) %>%
    mutate(aus_iv = bund_iv + kant_iv, per_iv = exis_iv + heim_iv,
        bundesanteil_ausgaben = bund_iv/(exis_iv + heim_iv)) |>
    rename_with(~str_remove(.x, "_iv"), .cols = -c(jahr, el_quote,
        bundesanteil_ausgaben))

EL_ZU_IV_BILANZ <- EL_ZU_IV_BILANZ_NOM |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-c(jahr, el_quote, bundesanteil_ausgaben),
        ~.x * diskontfaktor)) |>
    select(-diskontfaktor)

EL_TOT_BILANZ_NOM <- EL_BILANZ <- EL_BILANZ |>
    mutate(exis_tot = exis_ahv + exis_iv, heim_tot = heim_ahv +
        heim_iv, verw_tot = verw_ahv + verw_iv, kk_tot = kk_ahv +
        kk_iv, bund_tot = bund_ahv + bund_iv, kant_tot = kant_ahv +
        kant_iv) |>
    select(jahr, ends_with("_tot")) |>
    left_join(EL_QUOTEN_PROJ |>
        select(jahr, total)) |>
    rename(el_quote = total) %>%
    mutate(aus_tot = bund_tot + kant_tot, per_tot = exis_tot +
        heim_tot, bundesanteil_ausgaben = bund_tot/(exis_tot +
        heim_tot)) |>
    rename_with(~str_remove(.x, "_tot"), .cols = -c(jahr, el_quote,
        bundesanteil_ausgaben))

EL_TOT_BILANZ <- EL_TOT_BILANZ_NOM |>
    left_join(DISKONTFAKTOR, by = "jahr") |>
    mutate(across(-c(jahr, el_quote, bundesanteil_ausgaben),
        ~.x * diskontfaktor)) |>
    select(-diskontfaktor)
```

Als abschliessende automatisierte Qualitätssicherung wird überprüft, ob im totalen EL-Finanzhaushalt die Gesamtausgaben den Gesamteinnahmen entsprechen, und wenn dies der Fall ist, werden die Daten an `wrap_el_bilanz.R` zurückgegeben:

```
check_balance <- EL_TOT_BILANZ |>
    mutate(check = (exis + heim) - (bund + kant)) |>
    filter(abs(check) > 0.01)

if (nrow(check_balance) > 0) {
    stop("Die Ausgaben in einem Jahr entsprechen nicht den Einnahmen. Grund ist, dass bei
    ↪  mindestens einer Massnahme nur die Mehrausgaben berechnet werden, aber nicht die
    ↪  Finanzierzung dieser Mehrausgaben. Bitte bei Massnahmen Vektor(-en) mit der
    ↪  Finanzierung der Massnahme (Bund, Kanton, oder beide) hinzufügen.")
```

```
}


# --- Output
# ------------------------------------------------------------------

return(list(EL_ZU_AHV_BILANZ_NOM = EL_ZU_AHV_BILANZ_NOM, EL_ZU_IV_BILANZ_NOM =
↪  EL_ZU_IV_BILANZ_NOM,
    EL_TOT_BILANZ_NOM = EL_TOT_BILANZ_NOM, EL_ZU_AHV_BILANZ = EL_ZU_AHV_BILANZ,
    EL_ZU_IV_BILANZ = EL_ZU_IV_BILANZ, EL_TOT_BILANZ = EL_TOT_BILANZ))
```

4.14 Modul mod_population.R

Dokumentation zuletzt aktualisiert am 14.08.2025

Bemerkung: Dieses Modul beinhaltet die Aufbereitung sämtlicher Bevölkerungsdaten, die in den Finanzperspektiven des BSV genutzt werden. Der Grund ist, dass dieses Modul auch von den Finanzperspektivenmodellen der AHV, IV und EO verwendet wird, welche zusätzliche Bevölkerungsdaten verwenden. Für das EL-Modell sind lediglich die Daten zur Wohnbevölkerung Ende Jahr (nachfolgende Variable `bevendejahr`) relevant. Sämtliche anderen nachfolgend diskutierten Bevölkerungsvariablen werden nicht verwendet.

`mod_population.R` wird im Skript `wrap_el_vorb_berechn.R` wie folgt aufgerufen:

```
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 folgenden Argumente:

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

Zu beginn des Skripts werden die Alter über 99 Jahren in den Bevölkerungsszenarien zum Alter 99 hinzugezählt. Dies, da die Bevölkerungsszenarien nachfolgend mit den Bevölkerungsbeständen verbunden werden, und in der von uns 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
  ↪  einschränken.
  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","einwanderu
    ↪  \(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` spezifiziert wurde (ansonsten werden einfach die aktuellsten Bestände verwendet):

```
    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 freiwillig Versicherten der Bestand bis zum Jahr `jahr_bev` 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 freiwillig Versicherten sind jeweils im März des Folgejahres verfügbar. Die Wohnbevölkerungsbestände 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 freiwillig Versicherten für ein Jahr länger verfügbar als die Wohnbevölkerungsbestände. Daher wird davon ausgegangen, dass sich `jahr_bev` auf die Wohnbevölkerung bezieht, und dass die weiteren Bestände ein Jahr länger verfügbar sind.

Als nächstes werden aus den Dataframes `BEV_BESTAND` und `BEV_SCENARIO` sämtliche Variablen ausgewählt, welche für die Spätere bearbeitung mit den Szenariodaten verknüpft werden sollen (also alle Variablen ausser `c("bevanfangj", "geburt", tod", ëinbuergerung", ëinwanderung", "bereinigung", ßaisonniers", ässures_facultatifs")`):

```
    # Faktoren zur Justierung der Szenarien-Bestände auf die letzten beobachteten
    ↪  Bestände 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)  # Wähle 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 Berechnung des Faktors entfällt, wenn der erste Wert des Szenarios direkt an den zuletzt beobachteten Wert anknüpft (bspw. im Jahr 2025 starten die Wohnbevölkerungsszeanrien 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 Szenarioerstellungszeitpunkt und dem letzten beobachteten Jahr vom Szenario abweichenden Entwicklungen, zu verhindern.[21]

Als nächstes wird der Skalierungsfaktor für die freiwillig Versicherten berechnet (die Population an freiwillig Versicherten ist lediglich für den AHV Finanzhaushalt relevant. Dieser Codeteil kann also bei Betrachtung des EL 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 die freiwillig Versicherten keine Szenarien existieren, wird der Bestand an freiwillig Versicherten anhand des Anteils deren an der Wohnbevölkerung im letzten beobachteten Jahr (`max(jahr)`), multipliziert mit der Wohnbevölkerungsentwicklung, fortgeschrieben.In obigem Codeblock 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 freiwillig Versicherten in `min(jahr)` multipliziert mit der Wohnbevölkerung 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 (dieser Teil ist wiederum nur für den AHV Finanzhaushalt relevant, da der EL Finanzhaushalt keine Grenzgägner-Daten vor 2010 verwendet):

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

```
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)
```

Zum Schluss wird das Dataframe `BEVOELKERUNG` erstellt, welches die Daten zu Bevölkerungsbeständen und -Szenarien in einer durchgehenden 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"))
        ↪  |>  # Verknüpfe 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
        ↪  zurück 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 = "value") |>`) die Werte des Bevölkerungsszenarios an die Bevölkerungs-Bestandesdaten, für welche Szenariodaten existieren, angehängt und jeweils mit den in `fractions_bevoelkerung_scen` ermittelten Skalierungsfaktoren multipliziert. Danach werden die Bevölkerungs-Bestandesdaten für die freiwillig Versicherten und die Grenzgänger hinzugefügt, und es werden die fehlenden Werte für die frewillig Versicherten und die Grenzgänger durch Multiplikation der in `fraction_assures_facultatifs` und `fraction_frontaliers` Anteile mit der Wohnbevölkerung im jeweiligen Jahr berechnet. Die letzte if-Bedingung für die Auswanderung kann bei Betrachtung des EL Finanzhaushalts wiederum ignoriert werden, da diese nur für den AHV Finanzhaushalt relevant ist.

Somit ist die Berechnung des `BEVOELKERUNG` Dataframes abgeschlossen, und ebendieses kann an das Modul `wrap_el_vorb_berechn.R` zurückgegeben werden:

```
return(BEVOELKERUNG = BEVOELKERUNG)
```

4.15 Modul mod_eckwerte.R

Dokumentation zuletzt aktualisiert am 14.08.2025

`mod_eckwerte.R` wird im Skript `wrap_el_vorb_berechn.R` wie folgt aufgerufen:

```
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 folgenden Argumente:

```
mod_eckwerte <- function(PARAM_GLOBAL,
                         ECKWERTE,
                         LOHNINDEX,
                         PREISINDEX
                         ) {
```

- `ECKWERTE`: Datafrme mit allen Projektionen zu den Eckwerten zur wirtschaftlichen Entwicklung (insbesondere der Lohn- und Preisentwicklung) gemäss ESTV.
- `LOHNINDEX`: Historischer Lohnindex.
- `PREISINDEX`: Historischer Preisindex.

Zu beginn werden die gemäss `PARAM_GLOBAL` spezifizierten Eckwerte extrahiert:

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

Danach werden zwei 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))
}
```

In einem nächsten Schritt erfolgt die Berechnung des jährlichen Lohn- und Preiswachstums:

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

ECKWERTE_HIST <- LOHNINDEX %>%
    full_join(PREISINDEX) %>%
    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 fehlenden 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$jahr)` tiefer ist als das letzte Jahr des Projektionshorizonts `PARAM_GLOBAL$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 2024 sind Eckwerteprojektionen bis 2034 verfügbar, und das letzte Jahr des Projektionshorizonts ist 2070, d.h. die Eckwerte-Projektion für das Jahr 2034 wird für alle Jahre bis 2070 verwendet.

Als nächstes wird das Dataframe ECKWERTE_EXTENDED aus den historischen un den projizierten Eckwerten gebildet, wobei das Dataframe 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_EXTENDEDDataframes abgeschlossen, wodurch diese an das Modul wrap_el_vorb_berechn.R zurückgegeben werden können:

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

4.16 Modul mod_diskontfaktor.R

Dokumentation zuletzt aktualisiert am 25.11.2024

mod_diskontfaktor.R wird im Skript wrap_el_vorb_berechn.R wie folgt aufgerufen:

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

Die Funktion verlangt die folgenden Argumente:

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

- ECKWERTE_EXTENDED: Zeitreihe mit der historischen und der projizierten Lohn- und Preisentwicklung (vgl. Kapitel 4.15)

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 ist 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)
}
```

Somit erfolgt die Berechnung des Preisdeflators, welcher in der Modellterminologie als Diskontfaktor bezeichnet 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_el_vorb_berechn.R` zurückgegeben werden kann:

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

4.17 Modul mod_rentenentwicklung.R

Dokumentation zuletzt aktualisiert am 14.08.2025

`mod_rentenentwicklung` wird im Skript `wrap_el_vorb_berechn.R` wie folgt aufgerufen:

```
RENTENENTWICKLUNG <- mod_rentenentwicklung(PARAM_GLOBAL = tl_inp$PARAM_GLOBAL,
    ECKWERTE_EXTENDED = tl_eckwerte$ECKWERTE_EXTENDED, MINIMALRENTE =
    ↪  tl_inp$MINIMALRENTE,
    PREISINDEX = tl_inp$PREISINDEX)
```

Die Funktion verlangt die folgenden Argumente:

```
mod_rentenentwicklung <- function(PARAM_GLOBAL,
                                  ECKWERTE_EXTENDED,
                                  MINIMALRENTE,
                                  PREISINDEX
                                  ) {
```

- `ECKWERTE_EXTENDED`: Zeitreihe mit der historischen und der projizierten Lohn- und Preisentwicklung (vgl. Kapitel 4.15)
- `MINIMALRENTE`: Historische Zeitreihe mit der im jeweiligen Jahr gemäss Verordnung gültigen Minimalrente.
- `PREISINDEX`: Historischer Preisindex.

Als Erstes wird die relevante Wachstumsrate des Preisindexes für die Minimalrentenentwicklung gemäss Gesetz/Praxis in der Vergangenheit berechnet (vor 2017 Preisindex Ende Jahr, danach Jahresmittel Preisindex, ohne Rebasierung):

```
PREISWACHSTUM <- PREISINDEX %>%
    mutate(preisindex = ifelse(jahr < 2017, lik_dez_basis_1977,
        lik_basis_1977), preis_mischindex = 100 * (preisindex -
        lag(preisindex))/lag(preisindex)) %>%
    select(jahr, preis_mischindex)
```

Danach wird anhand der projizierten Lohn- und Preisentwicklung, welche im Dataframe `ECKWERTE_EXTENDED` enthalten ist, die Entwicklung der für die Minimalrentenberechnung relevanten Indizes (insbesondere des Mischindexes) projiziert. Eine Erläuterung zu der Berechnungsformel findet sich in den Hintergrunddokumentationen zum Mischindex (auf Anfrage beim Bereich MATH des BSV erhältlich):

```
  RENTENENTWICKLUNG <- ECKWERTE_EXTENDED |>
      left_join(PREISWACHSTUM) %>%
      mutate(preis=ifelse(jahr<=2017,preis_mischindex,preis)) %>%
      select(-preis_mischindex) %>%
      mutate(
          lohn = if_else(is.na(lohn), 0, lohn), # NA-Werte ersetzen
          preis = if_else(is.na(preis), 0, preis), # NA-Werte ersetzen
          lientw = cumprod(1 + lohn / 100), # Indexentwicklung berechnen
          pientw = cumprod(1 + preis / 100), # Indexentwicklung berechnen
          lohnindex = round(1004 * lientw), # Lohnindex berechnen
          preisindex = round(104.1 * pientw, 1), # Preisindex berechnen
          licomp = round(lag(lohnindex) / 10.04, 4), # Lohnkomponente berechnen
          picomp = round(lag(preisindex) / 1.041, 4), # Preiskomponente berechnen
          mischindex = round((licomp + picomp) / 2, 4), # Mischindex berechnen
          minimalrente_rd = 5 * round(5.5 * mischindex / 5) # Gerundete Minimalrente
          ↪  berechnen
      ) |>
```

Als nächstes wird die historische Minimalrente angefügt, und berücksichtigt, dass die Anpassung der Minimalrente nur alle zwei Jahre erfolgt:

```
      left_join(MINIMALRENTE, by = "jahr") |>
      mutate(
          rentenanpassung = case_when(
              jahr <= max(MINIMALRENTE$jahr, na.rm = TRUE) ~ if_else(minimalrente -
↪  lag(minimalrente) > 0, 1, 0),
              jahr == max(MINIMALRENTE$jahr, na.rm = TRUE)+1 ~ if_else(lag(minimalrente)
↪  - lag(minimalrente, 2) > 0, 0, 1),
              TRUE ~ jahr %% 2
          ), # Anpassungsrhythmus bestimmen
          minimalrente_pj = pmax(
```

```
            rentenanpassung * minimalrente_rd,
            lag(rentenanpassung * minimalrente_rd, default = 0)
        ), # Projektierte Minimalrente berechnen
        minimalrente = case_when(
            jahr <= max(MINIMALRENTE$jahr, na.rm = TRUE) ~ minimalrente,
            jahr == max(MINIMALRENTE$jahr, na.rm = TRUE)+1 ~
↪  ifelse(rentenanpassung==1,minimalrente_pj,lag(minimalrente)),
            TRUE ~ minimalrente_pj
        ) # Finalisierte Minimalrente setzen
    ) %>%
    mutate(
        dminimalrente = (minimalrente - lag(minimalrente)) / lag(minimalrente, default
        ↪  = 0), # Wachstumsrate berechnen
        rentenentwicklung = cumprod(1 + if_else(jahr > PARAM_GLOBAL$jahr_rr,
        ↪  dminimalrente, 0)) # Rentenentwicklung berechnen
    ) %>%
    select(-c(preis, lohn, dminimalrente, minimalrente_pj, minimalrente_rd, licomp,
    ↪  picomp))
```

die variable `rentenanpassung` wird so erstellt, dass sie `=1` ist für alle Jahre bis `last_year`, in welchem eine Rentenanpassung erfolgte, und danach in allen ungeraden Jahren (was Stand 2025 dem Anpassungsrythmus entspricht). Rentenanpassung wird dann für die Erstellung der Minimalrentenprojektion verwendet, wobei `minimalrente_pj` der weiter oben für das entsprechende Jahr projizierten Minimalrente entspricht falls es sich um ein Rentenanpassungsjahr handelt, und sonst der Minimalrentenprojektion für das Vorjahr.

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

```
return(RENTENENTWICKLUNG = RENTENENTWICKLUNG)
```

A Anhang

A.1 Aufbereitung der Daten aus dem EL-Register in SAS

Les variables du registre RPC sont décrites dans le fichier excel “Documentation_RPC_extrait_model_financier” et les variables des données préparées à l’aide du programme ci-dessous pour le modèle financier sont décrites dans le chapitre 2.3.2. Die aus der folgenden Aufbereitung resultierenden Daten werden mit dem in Kapitel 3.8 beschriebenen Modul in das Finanzperspektivenmodell eingelesen.

```
Aide à la lecture: Les éléments entre /*   */ ou entre *   ; sont des commentaires.


/* Préparation des données pour le modèle des finances PC */
/* modèle TOMAET */
/* Le but du programme est d'agréger les données du registre des PC selon les catégories
↪  utiles au modèle:
/* âge, sexe, assurance, lieu de vie (domicile/home) */

* 23.05.2025 Kma: adaptation du code pour les années dès 2014;


/* Marche à suivre ajout nouvelle année:
Partie A.
- ajouter le montant des Lebensbedarf: soit le même que l'année d'avant, soit un nouveau
- modifier l'année dans l'appel de la macro data_mfpc_cas
Partie B. + C.
Changer la date dans le nom du fichier créé
*/


/* Structure du script

A. Préparation des données
La préparation des données se fait dans une première boucle pour les années 2014 à 2018
↪  et dans une deuxième boucle presque identique pour les années
2019 et suivantes (actuellement 2024). Il y a seulement une différence dans la source et
↪  le nom de la base de données initiale, le reste est identique.
Dans chaque boucle, les données de l'année sont d'abord préparées, puis celles de l'année
↪  précédente.
Une partie des commentaires ne sont présents qu'une fois, dans la première boucle et pour
↪  l'année en cours.

B. Mettre ensemble les deux "morceaux" créés à l'étape A.

C. Enregistrer les données

*/



/* A. Préparation des données */
```

```
* loop 2014-2018;

rsubmit;

%macro data_mfpc_cas_98_18(annee_debut, annee_fin);

    %do annee=&annee_debut. %to &annee_fin.;

    %let annee_prec=%eval(&annee. - 1);

* 1. données sources;

* données année;

* merge avec le nap (noavs pseudonymisé) de référence, c'est-à-dire un numéro unique dans
↪  le temps;
proc sql;
create table el_faelle_comp_&annee._12 as
select a.*, b.napref as napref_ra
from pc.el_faelle_&annee. as a left join ra.rafixe as b
on a.nap1=b.nap;
quit;

* création des données;
data el_faelle_&annee._12 (keep=napref_ra cswo csg1 lsa1 ceref mbop_exsi mbop
↪  heim_mehrkosten_mbop in_jahr assurance);
set el_faelle_comp_&annee._12;

in_jahr=1;
assurance=.;
if csre1 in (1 2 6) then assurance=1;
/* if csre1=2 then assurance=2; */ /* on décide de mettre AV et AS ensemble pour le
↪  moment */
if csre1 in (3 4 5) then assurance=3;

if cgpr1 in (54 55 74 75) and lsa1 > 17 then lsa1 = 17; /* les personnes majeures avec
↪  une rente pour enfant lié à la rente d'un parent sont considéré comme mineurs pour ne
↪
pas les confondre avec les jeunes adultes à l'AI qui ont une rente eux-même (rente
↪  principale) */

heim_mehrkosten_mbop=mbop-mbop_exsi;

run;

* données année précédente;
proc sql;
create table el_faelle_comp_&annee_prec._12 as
select a.*, b.napref as napref_ra
from pc.el_faelle_&annee_prec. as a left join ra.rafixe as b
on a.nap1=b.nap;
quit;
```

```
data el_faelle_&annee_prec._12 (keep=napref_ra assurance_vorjahr in_vorjahr
↪  cswo_vorjahr);
set el_faelle_comp_&annee_prec._12;
heim_mehrkosten_mbop=mbop-mbop_exsi;
in_vorjahr=1;
assurance_vorjahr=.;
if csre1 in (1 2 6) then assurance_vorjahr=1;
/* if csre1=2 then assurance_vorjahr=2; */ /* on met AV et AS ensemble pour le moment */
if csre1 in (3 4 5) then assurance_vorjahr=3;
cswo_vorjahr=cswo;
run;

* mettre ensemble;
proc sort data=el_faelle_&annee._12 out=el_faelle_&annee._12; by napref_ra; run;
proc sort data=el_faelle_&annee_prec._12 out=el_faelle_&annee_prec._12; by napref_ra;
↪  run;

data elf_aed_&annee. ;
merge el_faelle_&annee._12 (in=a) el_faelle_&annee_prec._12 (in=b) ;
by napref_ra;
if a and napref_ra ne .;

* création des variables pour le modèle financier;
is_new_jahr=.;
if (in_vorjahr ne 1) or (assurance ne assurance_vorjahr) /* or (cswo ne cswo_vorjahr)*/
↪  then is_new_jahr=1; /* nouveau dans les PC ou dans l'assurance mais pas dans le lieu
↪  de domicile */
mbop_exsi_neu=.;
if is_new_jahr=1 then mbop_exsi_neu=mbop_exsi;

if (cswo = 2 and cswo_vorjahr ne 2) or (cswo = 2 and assurance ne assurance_vorjahr) then
↪  new_heim=1; /* si la personne habite nouvellement en home ou si elle est en home et a
↪  changé d'assurance */
heim_mehrkosten_mbop_neu=.;
if new_heim=1 then heim_mehrkosten_mbop_neu=heim_mehrkosten_mbop;
heim_pers=.;
if cswo=2 then heim_pers=1;
heim_pers_neu=.;
if new_heim=1 then heim_pers_neu=1;
run;


* 2. Préparation du tableau;
proc tabulate data=elf_aed_&annee. out=daten_tomaet_test1_&annee._temp;
var in_jahr is_new_jahr mbop_exsi mbop heim_mehrkosten_mbop mbop_exsi_neu
↪  heim_mehrkosten_mbop_neu heim_pers heim_pers_neu;
class csg1 lsa1 assurance;
table csg1*lsa1*assurance, in_jahr*SUM is_new_jahr*SUM mbop_exsi*SUM mbop_exsi_neu*SUM
↪  heim_mehrkosten_mbop*SUM heim_mehrkosten_mbop_neu*SUM heim_pers*SUM
↪  heim_pers_neu*SUM;
run;

data daten_tomaet_test1_&annee. (drop=_TYPE_ _PAGE_ _TABLE_);
```

```
set daten_tomaet_test1_&annee._temp;
annee=&annee.;
run;

%end;


/* fusion des tableaux des différentes années */
data daten_tomaet_test1_&annee_debut._&annee_fin.;
set daten_tomaet_test1_&annee_debut.-daten_tomaet_test1_&annee_fin. ;
run;

/* charger les données */
proc download data=daten_tomaet_test1_&annee_debut._&annee_fin.;
run;

%mend;

endrsubmit;

rsubmit;
%data_mfpc_cas_98_18(annee_debut=1998, annee_fin=2018);
endrsubmit;



* loop 2018 et après;

rsubmit;

%macro data_mfpc_cas(annee_debut, annee_fin);

    %do annee=&annee_debut. %to &annee_fin.;

    %let annee_prec=%eval(&annee. - 1);

* 1. données sources;

* données année;
proc sql;
create table el_faelle_comp_&annee._12 as
select a.*, b.napref as napref_ra
from rpc.el_faelle_&annee._12 as a left join ra.rafixe as b
on a.nap1=b.nap;
quit;

data el_faelle_&annee._12 (keep=napref_ra cswo csg1 lsa1 ceref mbop_exsi mbop
↪  heim_mehrkosten_mbop in_jahr assurance);
set el_faelle_comp_&annee._12;

in_jahr=1;
assurance=.;
if csre1 in (1 2 6) then assurance=1;
```

```
/* if csre1=2 then assurance=2; */ /* on décide de mettre AV et AS ensemble pour le
↪  moment */
if csre1 in (3 4 5) then assurance=3;

if cgpr1 in (54 55 74 75 ) and lsa1 > 17 then lsa1 = 17; /* les personnes majeures avec
↪  une rente pour enfant lié à la rente d'un parent sont considéré comme mineurs pour ne
↪
pas les confondre avec les jeunes adultes à l'AI car ils ont une rente eux-même */

heim_mehrkosten_mbop=mbop-mbop_exsi;

run;

* données année précédente;
proc sql;
create table el_faelle_comp_&annee_prec._12 as
select a.*, b.napref as napref_ra
from rpc.el_faelle_&annee_prec._12 as a left join ra.rafixe as b
on a.nap1=b.nap;
quit;

data el_faelle_&annee_prec._12 (keep=napref_ra assurance_vorjahr in_vorjahr
↪  cswo_vorjahr);
set el_faelle_comp_&annee_prec._12;
heim_mehrkosten_mbop=mbop-mbop_exsi;
in_vorjahr=1;
assurance_vorjahr=.;
if csre1 in (1 2 6) then assurance_vorjahr=1;
/* if csre1=2 then assurance_vorjahr=2; */ /* on met AV et AS ensemble pour le moment */
if csre1 in (3 4 5) then assurance_vorjahr=3;
cswo_vorjahr=cswo;
run;

* mettre ensemble;
proc sort data=el_faelle_&annee._12 out=el_faelle_&annee._12; by napref_ra; run;
proc sort data=el_faelle_&annee_prec._12 out=el_faelle_&annee_prec._12; by napref_ra;
↪  run;

data elf_aed_&annee. ;
merge el_faelle_&annee._12 (in=a) el_faelle_&annee_prec._12 (in=b) ;
by napref_ra;
if a and napref_ra ne .;

is_new_jahr=.;
if (in_vorjahr ne 1) or (assurance ne assurance_vorjahr) /* or (cswo ne cswo_vorjahr)*/
↪  then is_new_jahr=1; /* nouveau dans les PC ou dans l'assurance mais pas dans le lieu
↪  de domicile */
mbop_exsi_neu=.;
if is_new_jahr=1 then mbop_exsi_neu=mbop_exsi;

if (cswo = 2 and cswo_vorjahr ne 2) or (cswo = 2 and assurance ne assurance_vorjahr) then
↪  new_heim=1; /* si la personne habite nouvellement en home ou si elle est en home et a
↪  changé d'assurance */
```

```
heim_mehrkosten_mbop_neu=.;
if new_heim=1 then heim_mehrkosten_mbop_neu=heim_mehrkosten_mbop;
heim_pers=.;
if cswo=2 then heim_pers=1;
heim_pers_neu=.;
if new_heim=1 then heim_pers_neu=1;
run;


* 2. Préparation du tableau;
proc tabulate data=elf_aed_&annee. out=daten_tomaet_test1_&annee._temp;
var in_jahr is_new_jahr mbop_exsi mbop heim_mehrkosten_mbop mbop_exsi_neu
↪  heim_mehrkosten_mbop_neu heim_pers heim_pers_neu;
class csg1 lsa1 assurance;
table csg1*lsa1*assurance, in_jahr*SUM is_new_jahr*SUM mbop_exsi*SUM mbop_exsi_neu*SUM
↪  heim_mehrkosten_mbop*SUM heim_mehrkosten_mbop_neu*SUM heim_pers*SUM
↪  heim_pers_neu*SUM;
run;

data daten_tomaet_test1_&annee. (drop=_TYPE_ _PAGE_ _TABLE_);
set daten_tomaet_test1_&annee._temp;
annee=&annee.;
run;

%end;


/* fusion des tableaux des différentes années */
data daten_tomaet_test1_&annee_debut._&annee_fin.;
set daten_tomaet_test1_&annee_debut.-daten_tomaet_test1_&annee_fin. ;
run;

/* charger les données */
proc download data=daten_tomaet_test1_&annee_debut._&annee_fin.;
run;

%mend;

endrsubmit;

rsubmit;
%data_mfpc_cas(annee_debut=2019, annee_fin=2024);
endrsubmit;



/* B. Mettre ensemble les tableaux des deux boucles */
data daten_tomaet_2014_2024;
set daten_tomaet_test1_2014_2018 daten_tomaet_test1_2019_2024;
run;
```

```
/* C. Export */
proc export data=daten_tomaet_2014_2024
  outfile="O:\MASS\09_mathprod\09_el\documentation\daten_tomaet_2014_2024_v2.csv"
  DBMS=csv
  replace;
  delimiter=";";
  ;
run;
```