Import packages

In [1]:
```
using DataFrames, Plots, Statistics, JSON, Distributed
plotlyjs();
```

The WebIO Jupyter extension was not detected. See the WebIO Jupyter integration documentation for more information.

Prepare parallell processing - import code on all cores

The problem is simulated on 31 2.20 GHz processors running in parallel. TODO: Test on faster processors.

In [2]:
```
const numcores = 31

if nprocs() < numcores
    addprocs(numcores - nprocs())
end

@show nprocs();
```

```
nprocs() = 31
```

In [198…
```
@everywhere include(joinpath(dirname(pwd()),"jgrc/TuLiPa/src/TuLiPa.jl"));
```

In [199…
```
@everywhere include("JulES/src/JulES.jl");
```

Starting point and scenario settings

- Start model year 2025 and weather scenario 1981
- Series simulation 30 years with moving horizons and two day time steps.
- 7 weather years considered for uncertainty (always the next 7 years, TODO: replace with scenario generation from all 30 scenarios)
- Scenarios are phased in after two days
  - First two days have perfect information from starting point scenario
  - Next 5 weeks combines starting point scenario and scenario X. Smooth transition.
  - After 5 weeks the starting point scenario is fully phased out.

In [219…
```
datayear = 2025
scenarioyearstart = 1981
scenarioyearstop = 2011
horizonstart = 1981
numscen = 7

tnormal = FixedDataTwoTime(getisoyearstart(datayear),getisoyearstart(horizonstart))

phaseinoffsetdays = 2
phaseinoffset = Millisecond(Day(phaseinoffsetdays)) # phase in straight away from second st
phaseindelta = Millisecond(Week(5)) # Phase in the second stage scenario over 5 weeks
phaseinsteps = 5 # Phase in second stage scenario in 5 steps
tphasein = PhaseinTwoTime(getisoyearstart(datayear),getisoyearstart(horizonstart), getisoy
```

Price prognosis models uses the Thema dataset

This is a simplified version of the dataset NVE uses for its long-term power market analyses (for example https://www.nve.no/energi/analyser-og-statistikk/langsiktig-kraftmarkedsanalyse/). The dataset consist of:

- Detailed thermal units, aggregated hydro (Res, RoR, PHS), inelastic wind and solar, demands (inelastic with rationing price), storages (hydro and battery), transmission (ATC) and price areas (endogenous and exogenous).
- Levels in 2021, 2025, 2030, 2040 and 2050 (e.g. commodity price levels, installed production/transmission/storage capacities etc.)
- Profiles (e.g. availability profiles for transmission or production units, commodity price profiles, weather scenario profiles 1981-2010 for demand, solar, wind and inflow etc.)

We cannot publish the full dataset, but many of the profiles for wind, solar, hydro and demand can be downloaded from https://www.nve.no/energi/analyser-og-statistikk/vaerdatasett-for-kraftsystemmodellene/.

NB!

- In this demo we use the model year 2025. This scenario was made a couple of years ago and results should be considered outdated. A lot has happened in the power market since then. In addition the dataset is simplified.

In [220…

```
sti_themadata = "data_fra_thema"

themastructure = json_to_elements(sti_themadata, "dataset_thema.json")
themaseries = json_to_elements(sti_themadata, "tidsserier_thema.json")

elements = vcat(themaseries,themastructure)
addscenariotimeperiod!(elements, scenarioyearstart, scenarioyearstop);
```

Initialize and run price prognosis models

- Three levels of details: Long, medium and short term problems (run sequentially)
- Seven scenarios (run in parallell)
- Inputs:
  - Power market representation / Thema-data. (And how to aggregate it)
  - Problem length and time resolution
    - Long: 5 years long horizon with 6-weekly resolution for hydro, divided into 4 dynamic load blocks for power
    - Med: 54 week long horizon with weekly resolution for hydro, divided into 4 dynamic load blocks for power
    - Short: Week long horizon with daily resolution for hydro, and two-hourly for power
- Outputs:
  - Long, medium and short term prices for use in stochastic subsystem models.
  - Storage values (water values) from medium for use as end condition in stochastic subsystem models.
  - Thermal end states from short for use as end condition in market clearing model

In [221…

```
# Set horizons for price prognosis models
# Long
longhorizonend = getisoyearstart(horizonstart + 5)
longhydroperiodduration = Millisecond(Week(6))
longpowerparts = 6

longrhsdata = StaticRHSAHData("Power", datayear, scenarioyearstart, scenarioyearstop)
```

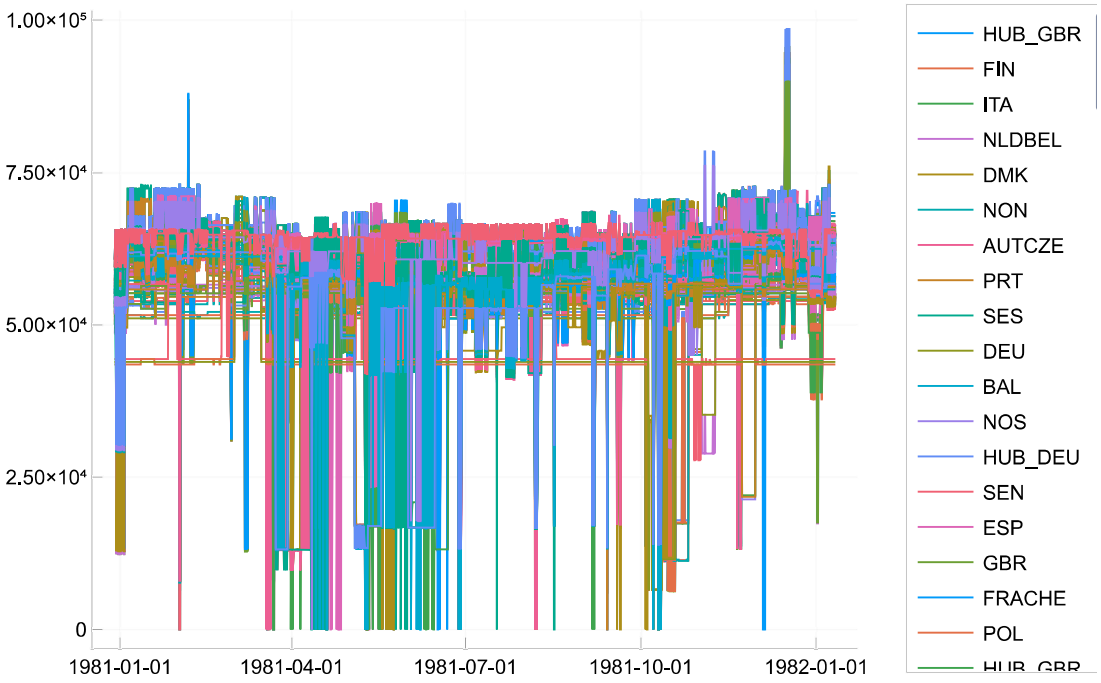
```
longmethod = KMeansAHMethod()
longclusters = 4
longunitduration = Millisecond(Hour(6))

# Medium
medweeklength = 54
medhorizonend = getisoyearstart(horizonstart) + Week(medweeklength)
medhydroperiodduration = Millisecond(Day(7)); @assert medweeklength % Int(longhydroperiodd
medpowerparts = 7

medrhsdata = DynamicRHSAHData("Power")
medmethod = KMeansAHMethod()
medclusters = 4
medunitduration = Millisecond(Hour(4))

# Short
shortweeklength = 1
shorthorizonend = getisoyearstart(horizonstart) + Week(shortweeklength)
shorthydroperiodduration = Millisecond(Day(1)); @assert medweeklength % shortweeklength ==
shortpowerparts = 12

# Preallocate storage for problems and results on different cores. Use package DistributedA
# Problems are built, updated, solved, and stored on a specific core. Moving a problem betw
longprobs = distribute([HiGHS_Prob() for i in 1:numscen])
medprobs = distribute([HiGHS_Prob() for i in 1:numscen], longprobs)
shortprobs = distribute([HiGHS_Prob() for i in 1:numscen], longprobs)

# Results are moved between cores. These are much smaller than longprobs/medprobs/shortprob
medprices = distribute([Dict() for i in 1:numscen], longprobs)
shortprices = distribute([Dict() for i in 1:numscen], longprobs)
medendvaluesobjs = distribute([EndValues() for i in 1:numscen], longprobs)
nonstoragestates = distribute([Dict{StateVariableInfo, Float64}() for i in 1:numscen], long

# Organise inputs and outputs
probs = (longprobs, medprobs, shortprobs)
longinput = (longhorizonend, longhydroperiodduration, longrhsdata, longmethod, longcluster
medinput = (medhorizonend, medhydroperiodduration, medrhsdata, medmethod, medclusters, med
shortinput = (shorthorizonend, shorthydroperiodduration, shortpowerparts, "short")
allinput = (numcores, elements, horizonstart, tnormal, tphasein, phaseinoffsetdays)
output = (medprices, shortprices, medendvaluesobjs, nonstoragestates)

# Initialize price prognosis models and run for first time step. Run scenarios in parallel
@time pl_prognosis_init!(probs, allinput, longinput, medinput, shortinput, output)
```

```
 24.840786 seconds (3.10 M allocations: 117.803 MiB)
```

Out[221]:
```
Task (done) @0x00000000bba6bc20
```

Plot medium term prices for the 7 scenarios

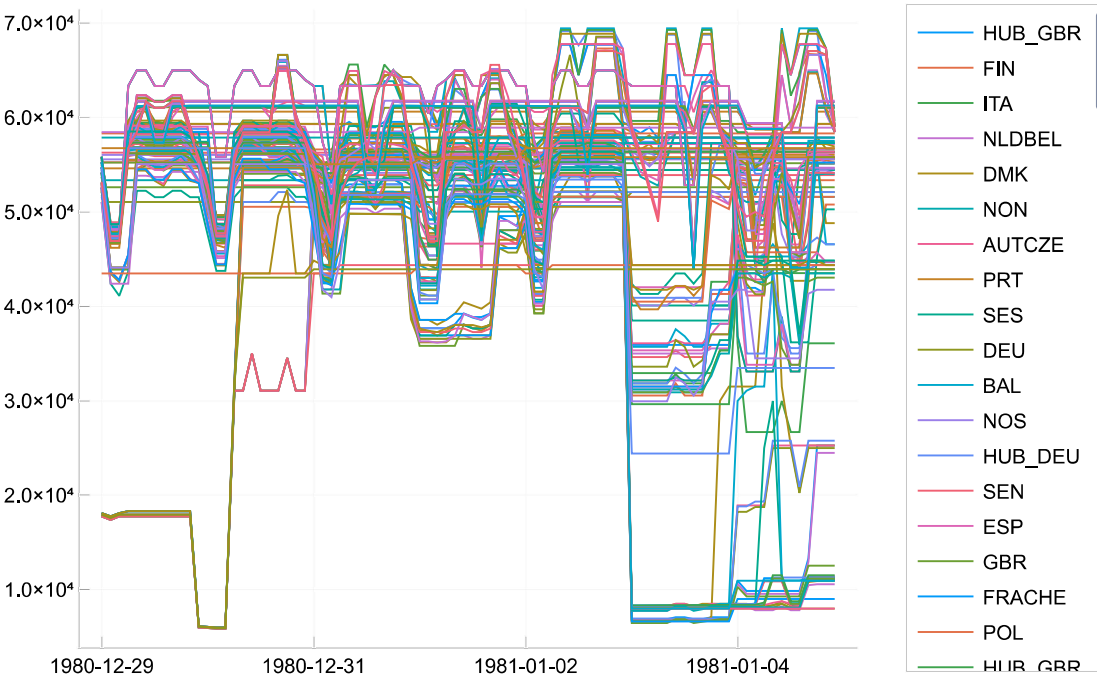
In [222…
```
scenario = 1
index = collect(medprices[scenario]["steprange"])
prices = medprices[scenario]["matrix"]
labels = [name for name in medprices[scenario]["names"]]
p = plot(index,prices,label=reshape(labels, 1, length(labels)),legend=:outertopright)

for scenario in 2:numscen
    prices = medprices[scenario]["matrix"]
    labels = [name for name in medprices[scenario]["names"]]
    plot!(p,index,prices,label=reshape(labels, 1, length(labels)),legend=:outertopright)
end
display(p)
```



Plot short term prices for the 7 scenarios

In [223…

```
scenario = 1
index = collect(shortprices[scenario]["steprange"])
prices = shortprices[scenario]["matrix"]
labels = [name for name in shortprices[scenario]["names"]]
p = plot(index,prices,label=reshape(labels, 1, length(labels)),legend=:outertopright)

for scenario in 2:numscen
    prices = shortprices[scenario]["matrix"]
    labels = [name for name in shortprices[scenario]["names"]]
    plot!(p, index,prices,label=reshape(labels, 1, length(labels)),legend=:outertopright)
end
display(p)
```



Stochastic subsystem models and market clearing model uses more detailed data

Thema dataset (explained above) combined with detailed hydropower for Norway and Sweden. The full detailed dataset contains for Nortwestern Europe:

- 32 price areas (9 exogen)
- 73 transmission lines (19 with ramping)
- 162 demands
- 88 batteries (normal, and representing demand response and V2G)
- 294 thermal plants (228 with start up costs)
- 100 wind and solar plants (aggregated)
- 1482 hydropower modules
  - 965 with production
  - 43 with pumps (includes PHS)
  - 998 with reservoirs
  - 788 restrictions (environmental, reservoir curves and ramping)
  - 90 PQ-curves (mostly Sweden)

NVEs dataset for the hydropower system in 2022 is open, but we have not published datasets for 2025/2030/2040/2050 since it would reveal investment plans. The dataset exist in several formats:

- Aggregated (for Res and RoR) production capacity, reservoir capacity, inflow and inflow profiles per price areas from https://www.nve.no/energi/analyser-og-statistikk/vaerdatasett-for-kraftsystemmodellene/
- Detailed watercourse descriptions from https://www.nve.no/energi/energisystem/vannkraft/modell-av-det-norske-vannkraftsystemet/. The dataset exist in two DETD-formats (per EMPS area (also includes rest modules for small-scale hydro) or per watercourse), and simplified in an Excel-format. The dataset used in this demo is derived from the excel-format with some differences:
  - Every water balance has its own module (i.e. a module with both regulated and unregulated inflow is split into two modules).
  - It has pq-kurves and environmental restrictions
  - It has ramping restrictions for hydropower release (only used in market clearing).
- The inflow series for the detailed dataset can be found at https://www.nve.no/vann-og-vassdrag/hydrologiske-data/historiske-data/historiske-vannfoeringsdata-til-produksjonsplanlegging/.

In [224…

```
# Read dataset for stochastic and market clearing models
sti_themadata = "data_fra_thema"
tst = JSON.parsefile(joinpath(sti_themadata, "dataset_thema_excl_hydro_nose.json")) # Europ
themastructure = getelements(tst)
tse = JSON.parsefile(joinpath(sti_themadata, "tidsserier_thema.json"))
themaseries = getelements(tse, sti_themadata)

dse = JSON.parsefile("data_fra_dynmodell/tidsserier_detd.json")
detdseries = getelements(dse);
dst = JSON.parsefile("data_fra_dynmodell/dataset_detd.json")
detdstructure = getelements(dst);

detailedelements = vcat(themaseries,themastructure,detdseries,detdstructure)
addscenariotimeperiod!(detailedelements, scenarioyearstart, scenarioyearstop);
```

Get water values for detailed hydro (DETD) from aggregated hydro (Thema)

- With different representations of watercourses in different problems, we need a mapping

In [225…
```
# Mapping between aggregated and detailed storages
detailedrescopl = JSON.parsefile("data_fra_dynmodell/magasin_elspot1.json")

# detailed dataset has reservoirs for SE4, aggregated does not, TODO: Improve aggregation/m
for k in keys(detailedrescopl)
    if detailedrescopl[k] == "SE4"
        detailedrescopl[k] = "SE3"
    end
end

# Get dictionary with each detailed reservoir and their water value for each scenario
# TODO: Detailed run-of-river reservoirs get water value from aggregated reservoir hydro
function getendvaluesdicts(endvaluesobjs, detailedrescopl::Dict)
    endvaluesdicts = Dict[]
    for endvaluesobj in endvaluesobjs
        instance = [getinstancename(getid(obj)) for obj in endvaluesobj.objects]
        endvalues = endvaluesobj.values
        endvaluesdict = Dict(instance .=> endvalues)

        for (k,v) in detailedrescopl
            endvaluesdict["Reservoir_" * k] = endvaluesdict["Reservoir_" * v * "_hydro_res
        end
        push!(endvaluesdicts, endvaluesdict)
    end

    return endvaluesdicts
end
medendvaluesdicts = getendvaluesdicts(medendvaluesobjs, detailedrescopl);
```

Stochastic subsystem models with Benders decomposition

- Each unique storage system is solved as a two-stage linear program with Benders decomposition
- The subsystems are optimized against exogen prices from the price prognosis scenarios
- We group storage systems into two. These are solved with different degree of detail
  - Med-term - mostly hydro
    - 52 week long horizon, weekly resolution, price from medium term price prognosis
  - Short-term - mostly batteries and PHS
    - Week long horizon, hourly resolution, price from short term price prognosis
- Outputs:
  - Cuts or storage values (battery or hydro) for use as end condition in market clearing problem

In [226…
```
# Cut parameters
maxcuts = 13 # preallocate fixed number of cuts, no cut selection
lb = -1e10 # lower bound of the future value in the first iteration
reltol = 0.0001 # relative tolerance

# Parameters for stochastic subsystem problems
shorttotaldays = shortweeklength*7
medtotaldays = (medweeklength-2)*7 # we reuse prices for two weeks, so have to be two weeks
shortstartstorage = 50
medstartstorage = 65
medpriceslocal = convert(Vector{Dict}, medprices)
shortpriceslocal = convert(Vector{Dict}, shortprices)
medendvaluesdicts = getendvaluesdicts(medendvaluesobjs, detailedrescopl)
storageinfo = (shortstartstorage, medstartstorage, medendvaluesdicts)
shortterminputs = (detailedelements, shorttotaldays, numscen, horizonstart, phaseinoffsetda
medterminputs = (detailedelements, medtotaldays, numscen, horizonstart, phaseinoffsetdays,

ustoragesystemobjects = []
ushorts = []
```

```
# Make modelobjects for short-term subsystems
@time makemastersubobjects!(shortterminputs, ustoragesystemobjects, ushorts)
# Make modelobjects for medium-term subsystems
@time makemastersubobjects!(medterminputs, ustoragesystemobjects, ushorts)

# Distribute subsystems with inputs and outputs on different cores
storagesystemobjects, shorts = distribute_subsystems(ustoragesystemobjects, ushorts)
masters = distribute([HiGHS_Prob() for i in 1:length(storagesystemobjects)], storagesystem
subs = distribute([[] for i in 1:length(storagesystemobjects)], storagesystemobjects)
states = distribute([Dict{StateVariableInfo, Float64}() for i in 1:length(storagesystemobj
cuts = distribute([SimpleSingleCuts() for i in 1:length(storagesystemobjects)], storagesys
storagesystems = distribute([Dict() for i in 1:length(storagesystemobjects)], storagesyste

# Initialize subsystem problems and run for first time step. Run subsystems in parallell
@time pl_stochastic_init!(numcores, storagesystemobjects, shorts, masters, subs, states, c
```

```
  3.463325 seconds (12.42 M allocations: 494.330 MiB, 17.29% gc time)
  3.247567 seconds (12.53 M allocations: 503.512 MiB)
 86.066231 seconds (412.27 k allocations: 11.856 MiB)
```

Out[226]:
```
Task (done) @0x00000000b12387d0
```

Market clearing problem

- Deterministic LP
- Two days, two-hour resolution, full detail (hydro and thermal)
- Inputs:
  - Detailed power market representation
  - Water values and battery storage values from stochastic subsystem problems
  - Thermal end condition from short term price prognosis model
- Outputs:
  - Two day detailed power market simulation (price, production, reservoir etc...)
  - Start state for next time step (reservoirs and other state variables)

In [227…
```
# Bring data to local core
cutslocal = convert(Vector{SimpleSingleCuts}, cuts)
nonstoragestateslocal = convert(Vector{Dict}, nonstoragestates)

# Initialize market clearing problem and run for first time step
@time clearing, nonstoragestatesmean, clearingendvaluesdict, varendperiod = clearing_init(
```

```
  4.405481 seconds (3.96 M allocations: 221.829 MiB)
```

In [228…
```
# Initialize start states for next time step, also mapping to Thema storages and max capac
startstates, enekvglobaldict = startstates_init(clearing, detailedrescopl, longprobs[1], t
```

Initialize and collect results: Prices and start states (e.g. reservoirs)

In [229…
```
# Initialize and collect prices and start states
price = Dict()
powerhorizonix = argmax(getnumperiods(h) for h in clearing.horizons)
getareaprices!(price, clearing, clearing.horizons[powerhorizonix], tnormal)
areanames = price["names"]
ix = collect(price["steprange"])
pricematrix = price["matrix"]

statenames = collect(keys(startstates))
statematrix = collect(values(startstates));
```

Collect more detailed results

- Prices, supply, demand, transmission and hydro storage levels
- TODO: Customize results more, now assumes same time resolution for all hydropower reservoirs

- TODO: Convert hydro reservoir levels to energy

In [230…

```
function init_results(clearing, modelobjects, resultobjects, numperiods_powerhorizon, numpe
    # Order result objects into lists
    powerbalances = []
    rhsterms = []
    rhstermbalances = []
    plants = []
    plantbalances = []
    plantarrows = Dict()
    demands= []
    demandbalances = []
    demandarrows = Dict()
    hydrostorages = []

    for obj in resultobjects # regret using Ids rather than the whole object

        # Powerbalances
        if obj isa BaseBalance
            if getinstancename(getid(getcommodity(obj))) == "Power"
                if !isexogen(obj)
                    push!(powerbalances, getid(obj))
                    for rhsterm in getrhsterms(obj)
                        push!(rhsterms,getid(rhsterm))
                        push!(rhstermbalances,getid(obj))
                    end
                end
            end
        end

        # Hydrostorages
        if obj isa BaseStorage
            if getinstancename(getid(getcommodity(getbalance(obj)))) == "Hydro"
                push!(hydrostorages,getid(obj))
            end
        end

        # Supply and demands
        if obj isa BaseFlow
            # The type of supply or demand can be found based on the arrows
            arrows = getarrows(obj)

            # Simple supplies and demands
            powerarrowbool = [(getid(getcommodity(getbalance(arrow))) == Id("Commodity", "P
            powerarrows = arrows[powerarrowbool]
            if sum(powerarrowbool) == 1
                if isingoing(powerarrows[1])
                    push!(plants,getid(obj))
                    push!(plantbalances,getid(getbalance(powerarrows[1])))
                elseif !isingoing(powerarrows[1])
                    push!(demands,getid(obj))
                    push!(demandbalances,getid(getbalance(powerarrows[1])))
                end
            end

            # Transmissions
            if sum(powerarrowbool) == 2
                for arrow in arrows
                    balance = getbalance(arrow)
                    if getid(getcommodity(balance)) == Id("Commodity", "Power")
                        if isingoing(arrow) && (balance in resultobjects)
                            push!(plants,getid(obj))
                            push!(plantbalances,getid(balance))
                            if isexogen(balance)
                                plantarrows[getid(obj)] = arrow
                            end
                        elseif !isingoing(arrow) && (balance in resultobjects)
                            push!(demands,getid(obj))
```

```
                            push!(demandbalances,getid(balance))
                            if isexogen(balance)
                                demandarrows[getid(obj)] = arrow
                            end
                        end
                    end
                end
            end


            # Supplies with SegmentedArrows (hydropower with PQ-kurves)
            pqarrowbool = [arrow isa SegmentedArrow for arrow in arrows]
            pqarrows = arrows[pqarrowbool]
            if sum(pqarrowbool) == 1
                if isingoing(pqarrows[1])
                    push!(plants,getid(obj))
                    push!(plantbalances,getid(getbalance(pqarrows[1])))
                end
            end
        end

        # Aggregated supplies (thermal power plants aggregated into one or more equivalent
        # TODO: Result should be a sum of all clusters, not separated
        if obj isa BaseAggSupplyCurve
            instance = getinstancename(getid(obj))
            concept = getconceptname(getid(obj))
            balance = getbalance(obj)
            for c in 1:getnumclusters(obj)
                newname = string(instance,"_",c)
                push!(plants,Id(concept,newname))
                push!(plantbalances,getid(balance))
            end
        end
    end

    prices, rhstermvalues, production, consumption, hydrolevels = get_results(clearing, pow

    return prices, rhstermvalues, production, consumption, hydrolevels, powerbalances, rhst
end

function update_results(clearing, oldprices, oldrhstermvalues, oldproduction, oldconsumptio

    prices, rhstermvalues, production, consumption, hydrolevels = get_results(clearing, pow

    prices = vcat(oldprices, prices)
    rhstermvalues = vcat(oldrhstermvalues, rhstermvalues)
    production = vcat(oldproduction, production)
    consumption = vcat(oldconsumption, consumption)
    hydrolevels = vcat(oldhydrolevels, hydrolevels)

    return prices, rhstermvalues, production, consumption, hydrolevels
end

function get_results(clearing, powerbalances, rhsterms, plants, plantbalances, plantarrows

    # Matrices to store results per time period, scenario and object
    prices = zeros(numperiods_powerhorizon, length(powerbalances))
    rhstermvalues = zeros(numperiods_powerhorizon, length(rhsterms))
    production = zeros(numperiods_powerhorizon, length(plants))
    consumption = zeros(numperiods_powerhorizon, length(demands))
    hydrolevels = zeros(numperiods_hydrohorizon, length(hydrostorages))

    for j in 1:numperiods_powerhorizon

        # Timefactor transform results from GWh to GW/h regardless of horizon period durat
        timefactor = periodduration_power/Millisecond(3600000)

        # For powerbalances collect prices and rhsterms (like inelastic demand, wind, solar
        for i in 1:length(powerbalances)
```

```
            prices[j, i] = -getcondual(clearing, powerbalances[i], j)/1000 # from €/GWh to
            for k in 1:length(rhsterms)
                if hasrhsterm(clearing, powerbalances[i], rhsterms[k], j)
                    rhstermvalues[j, k] = getrhsterm(clearing, powerbalances[i], rhsterms[
                end
            end
        end

        # Collect production of all plants
        for i in 1:length(plants) # TODO: Balance and variable can have different horizons
            concept = getconceptname(plants[i])
            if concept != AGGSUPPLYCURVE_CONCEPT
                arrows = getarrows(modelobjects[plants[i]])
                pqarrowbool = [arrow isa SegmentedArrow for arrow in arrows]
                pqarrows = arrows[pqarrowbool]
                if sum(pqarrowbool) == 1
                    production[j, i] = 0
                    for k in 1:length(getconversions(pqarrows[1]))
                        segmentid = getsegmentid(pqarrows[1], k)
                        production[j, i] += getvarvalue(clearing, segmentid, j)*abs(getcon
                    end
                else
                    if isexogen(modelobjects[plantbalances[i]])
                        # TODO: Balance and variable can have different horizons
                        arrow = plantarrows[plants[i]]
                        horizon = gethorizon(arrow)
                        conversionparam = _getcontributionparam(arrow)
                        querytime = getstarttime(horizon, j, t)
                        querydelta = gettimedelta(horizon, j)
                        conversionvalue = getparamvalue(conversionparam, querytime, queryd
                        production[j, i] = getvarvalue(clearing, plants[i], j)*conversionv
                    else
                        production[j, i] = getvarvalue(clearing, plants[i], j)*abs(getconc
                    end
                end
            else
                production[j, i] = getvarvalue(clearing, plants[i], j)*abs(getconcoeff(cle
            end
        end

        # Collect demand of all demands
        for i in 1:length(demands) # TODO: Balance and variable can have different horizons
            if isexogen(modelobjects[demandbalances[i]])
                arrow = demandarrows[demands[i]]
                horizon = gethorizon(arrow)
                conversionparam = _getcontributionparam(arrow)
                querytime = getstarttime(horizon, j, t)
                querydelta = gettimedelta(horizon, j)
                conversionvalue = getparamvalue(conversionparam, querytime, querydelta)
                consumption[j, i] = getvarvalue(clearing, demands[i], j)*conversionvalue/t
            else
                consumption[j, i] = getvarvalue(clearing, demands[i], j)*abs(getconcoeff(c
            end
        end
    end

    # Collect hydro storage levels
    for j in 1:numperiods_hydrohorizon
        for i in 1:length(hydrostorages)
            hydrolevels[j, i] = getvarvalue(clearing, hydrostorages[i], j)/1000 # Gm3 TODO
        end
    end

    return prices, rhstermvalues, production, consumption, hydrolevels
end
```

Out[230]: get_results (generic function with 1 method)

In [231…

```
numperiods_powerhorizon = 24
numperiods_hydrohorizon = 8
periodduration_power = Millisecond(Hour(2));
periodduration_hydro = Millisecond(Hour(6));
clearingobjects = Dict(zip([getid(obj) for obj in clearing.objects],clearing.objects))
resultobjects = getpowerobjects(clearingobjects,["NO2"]); # only collect results for one ar
# resultobjects = clearing.objects # collect results for all areas
prices, rhstermvalues, production, consumption, hydrolevels, powerbalances, rhsterms, rhst
```

Simulate next time steps (16 steps = 32 days)

- Simulate next time steps, store results and plot results

In [232…

```
# Only update medterm and longterm every 8 days (med/long prices and med/long watervalues c
skipmed = Millisecond(Day(2))

steps = 16;
# steps = (getisoyearstart(1982) - getisoyearstart(scenarioyearstart)).value/Millisecond(Da
```

In [233…

```
step = 2
tnormal += Day(2)
tphasein = PhaseinTwoTime(getdatatime(tnormal), getscenariotime(tnormal), getscenariotime(t
display(tnormal)

totaltime = @elapsed while step <= steps

    # Deterministic long/mid/short - calculate scenarioprices
    @time pl_prognosis!(numcores, longprobs, medprobs, shortprobs, medprices, shortprices,

    # Stochastic sub systems - calculate storage value
    if skipmed.value == 0
        medpriceslocal = convert(Vector{Dict}, medprices)
        medendvaluesdicts = getendvaluesdicts(medendvaluesobjs, detailedrescopl)
    end
    shortpriceslocal = convert(Vector{Dict}, shortprices)

    @time pl_stochastic!(numcores, masters, subs, states, cuts, startstates, medpricesloca

    # Market clearing
    cutslocal = convert(Vector{SimpleSingleCuts}, cuts)
    nonstoragestateslocal = convert(Vector{Dict}, nonstoragestates)

    # Try reusing cuts in market clearing, if non-optimal solution retry with fresh cuts
    try
        @time startstates = clearing!(clearing, startstates, cutslocal, clearingendvaluesd

        updateareaprices!(price, clearing, clearing.horizons[powerhorizonix], tnormal)
        ix = vcat(ix, collect(price["steprange"]))
        pricematrix = vcat(pricematrix, price["matrix"])
        statematrix = hcat(statematrix, collect(values(startstates)))

        prices, rhstermvalues, production, consumption, hydrolevels = update_results(clear

        step += 1
        tnormal += Day(2)
        tphasein = PhaseinTwoTime(getdatatime(tnormal), getscenariotime(tnormal), getscena
        display(tnormal)

        skipmed += Millisecond(Day(2))
        if skipmed > Millisecond(Day(6))
            skipmed = Millisecond(0)
        end
    catch
        println("Non-optimal market clearing, calculating new cuts for time step (could als
        skipmed = Millisecond(0)
    end
```

```
end

display(string("The simulation took: ", totaltime/60, " minutes"))
display(string("Time usage per timestep: ", totaltime/steps, " seconds"))
```

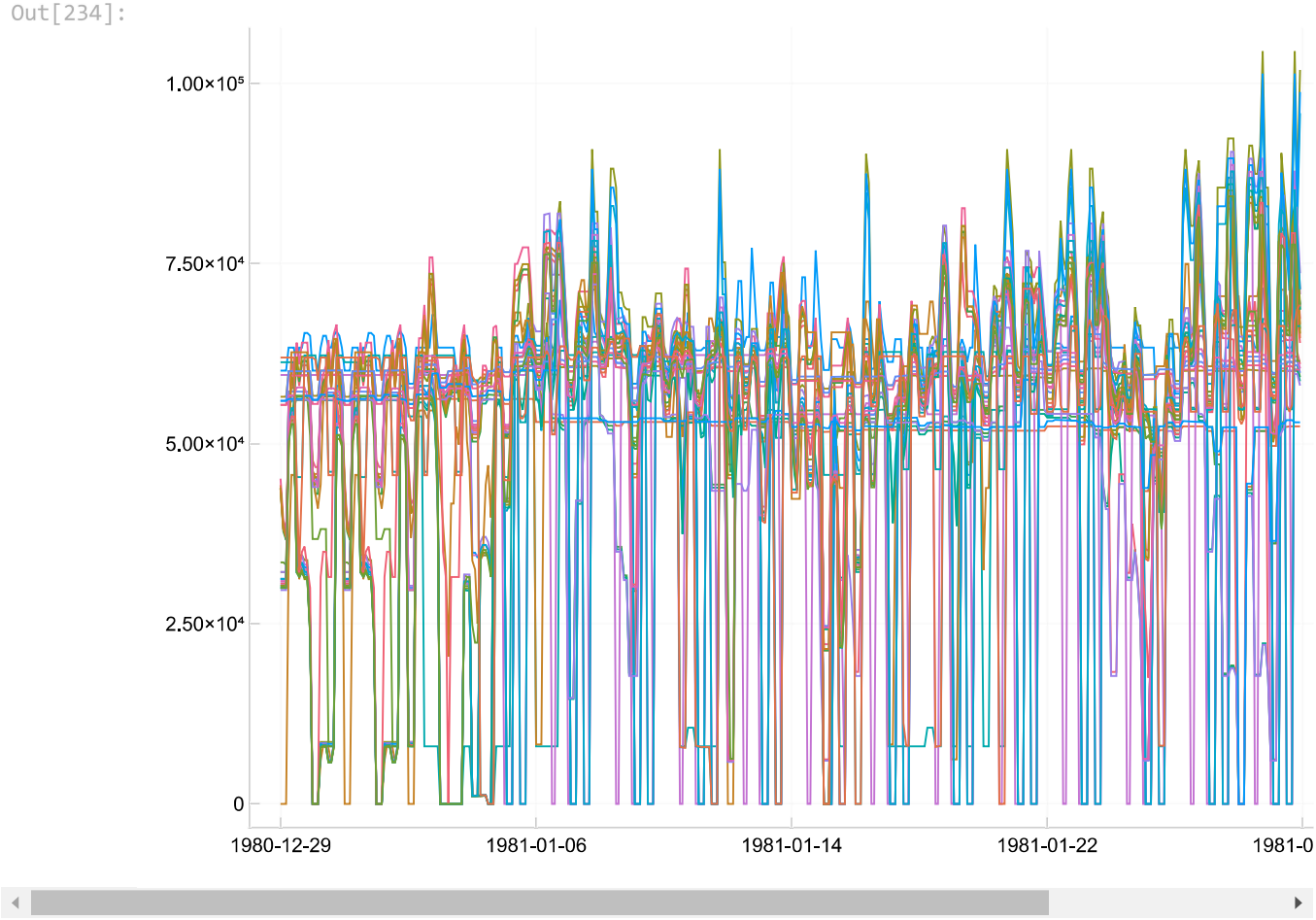
```
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1980-12-31T00:00:00"))
  6.883573 seconds (75.81 k allocations: 5.101 MiB)
  5.999406 seconds (526.70 k allocations: 17.165 MiB)
  6.848061 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-02T00:00:00"))
  1.523430 seconds (75.70 k allocations: 5.100 MiB)
  0.348360 seconds (526.54 k allocations: 17.162 MiB)
 12.482530 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-04T00:00:00"))
  1.636714 seconds (79.73 k allocations: 5.368 MiB, 80.41% gc time)
  5.347590 seconds (526.68 k allocations: 17.163 MiB)
 10.564286 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-06T00:00:00"))
  7.835753 seconds (75.82 k allocations: 5.106 MiB)
 26.457638 seconds (527.14 k allocations: 17.174 MiB)
 13.141320 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-08T00:00:00"))
  1.471588 seconds (75.60 k allocations: 5.092 MiB)
  1.582393 seconds (526.74 k allocations: 17.177 MiB)
  7.982458 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-10T00:00:00"))
  7.441561 seconds (75.77 k allocations: 5.095 MiB)
  0.494595 seconds (526.54 k allocations: 17.161 MiB)
 11.294219 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-12T00:00:00"))
  1.762676 seconds (75.71 k allocations: 5.100 MiB)
  0.570465 seconds (526.55 k allocations: 17.162 MiB)
 18.012947 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-14T00:00:00"))
  3.763816 seconds (75.71 k allocations: 5.104 MiB)
 18.942986 seconds (527.02 k allocations: 17.178 MiB)
 11.815193 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-16T00:00:00"))
  2.039319 seconds (75.62 k allocations: 5.092 MiB)
  0.342542 seconds (526.54 k allocations: 17.162 MiB)
 12.989299 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-18T00:00:00"))
  1.979819 seconds (75.72 k allocations: 5.102 MiB)
  0.365452 seconds (526.54 k allocations: 17.162 MiB)
 13.002922 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-20T00:00:00"))
  2.118496 seconds (75.62 k allocations: 5.095 MiB)
  0.350516 seconds (526.54 k allocations: 17.162 MiB)
  9.207951 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-22T00:00:00"))
  4.035766 seconds (75.61 k allocations: 5.092 MiB)
 18.917217 seconds (526.92 k allocations: 17.169 MiB)
  8.575473 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-24T00:00:00"))
  1.989106 seconds (76.33 k allocations: 5.145 MiB)
  0.353261 seconds (526.53 k allocations: 17.160 MiB)
 10.539182 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-26T00:00:00"))
  1.899608 seconds (75.71 k allocations: 5.103 MiB)
  0.333435 seconds (526.54 k allocations: 17.161 MiB)
 10.384904 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-28T00:00:00"))
  1.303387 seconds (75.59 k allocations: 5.092 MiB)
  0.339495 seconds (526.54 k allocations: 17.160 MiB)
  9.825352 seconds (508.42 k allocations: 26.846 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-30T00:00:00"))
"The simulation took: 4.98155437 minutes"
"Time usage per timestep: 18.6808288875 seconds"
```

Plot resulting prices

In [234…

```
plot(ix,pricematrix,label=reshape(areanames, 1, length(areanames)),size=(800,500),legend=:
```

Out[234]:



Comments:

- Throughout the testing we have achieved the wanted price volatility in the thermal dominated part of the dataset (Western Europe). On the other hand, the Nordics have had very flat prices due to too much flexibility in the hydropower system. In this demo we have therefore added hydropower production ramping restrictions in an attempt to reduce the flexibility of the run-of-river hydropower plants. This results in much more price volatiliy, but at a big computational cost.
- Ramping restrictions on transmission lines has the same effect. Without ramping restrictions (hydro and transmission) the computational time is down to around 9 seconds in total per time step. This is promising considering the big dataset, and the list of possible optimization we have in mind. It is also always possible to clear the market for 24 hours instead of 48 hours like now, which would reduce the computational time substantially. It is also interesting what these computational times would be with a commercial solver.
- We will try different configurations of ramping restrictions, and also test if time delays can achieve the same effects at a lower computation cost. Considering unavailability of hydropower or reserve market obligations, should also decrease the flexibility of the hydropower system.

Postprocess detailed results

- Combine fixed contributions (e.g. wind, solar and demand) together with supply and demand variables
- Make time axis for price, supply/demand and reservoir levels

In [235…

```
# Only keep rhsterms that have at least one value (TODO: Do the same for sypply and demands
rhstermtotals = dropdims(sum(rhstermvalues,dims=1),dims=1)
rhstermsupplyidx = []
rhstermdemandidx = []

for k in 1:length(rhsterms)
    if rhstermtotals[k] > 0
        push!(rhstermsupplyidx, k)
    elseif rhstermtotals[k] < 0
        push!(rhstermdemandidx, k)
    end
end

# Put rhsterms together with supplies and demands
rhstermsupplyvalues = rhstermvalues[:,rhstermsupplyidx]
rhstermdemandvalues = rhstermvalues[:,rhstermdemandidx]*-1

rhstermsupplynames = [getinstancename(rhsterm) for rhsterm in rhsterms[rhstermsupplyidx]]
rhstermsupplybalancenames = [split(getinstancename(r), "PowerBalance_")[2] for r in rhsterm
rhstermdemandnames = [getinstancename(rhsterm) for rhsterm in rhsterms[rhstermdemandidx]]
rhstermdemandbalancenames = [split(getinstancename(r), "PowerBalance_")[2] for r in rhsterm

supplynames = [[getinstancename(plant) for plant in plants];rhstermsupplynames]
supplybalancenames = [[split(getinstancename(p), "PowerBalance_")[2] for p in plantbalances
supplyvalues = hcat(production,rhstermsupplyvalues)

demandnames = [[getinstancename(demand) for demand in demands];rhstermdemandnames]
demandbalancenames = [[split(getinstancename(p), "PowerBalance_")[2] for p in demandbalanc
demandvalues = hcat(consumption, rhstermdemandvalues)

# Prepare for plotting results
hydronames = [getinstancename(hydro) for hydro in hydrostorages]
powerbalancenames = [split(getinstancename(powerbalance), "PowerBalance_")[2] for powerbala

# Time
x1 = [getisoyearstart(horizonstart) + periodduration_power*(t-1) for t in 1:first(size(supp
x2 = [getisoyearstart(horizonstart) + periodduration_hydro*(t-1) for t in 1:first(size(hydr
```

In [236…

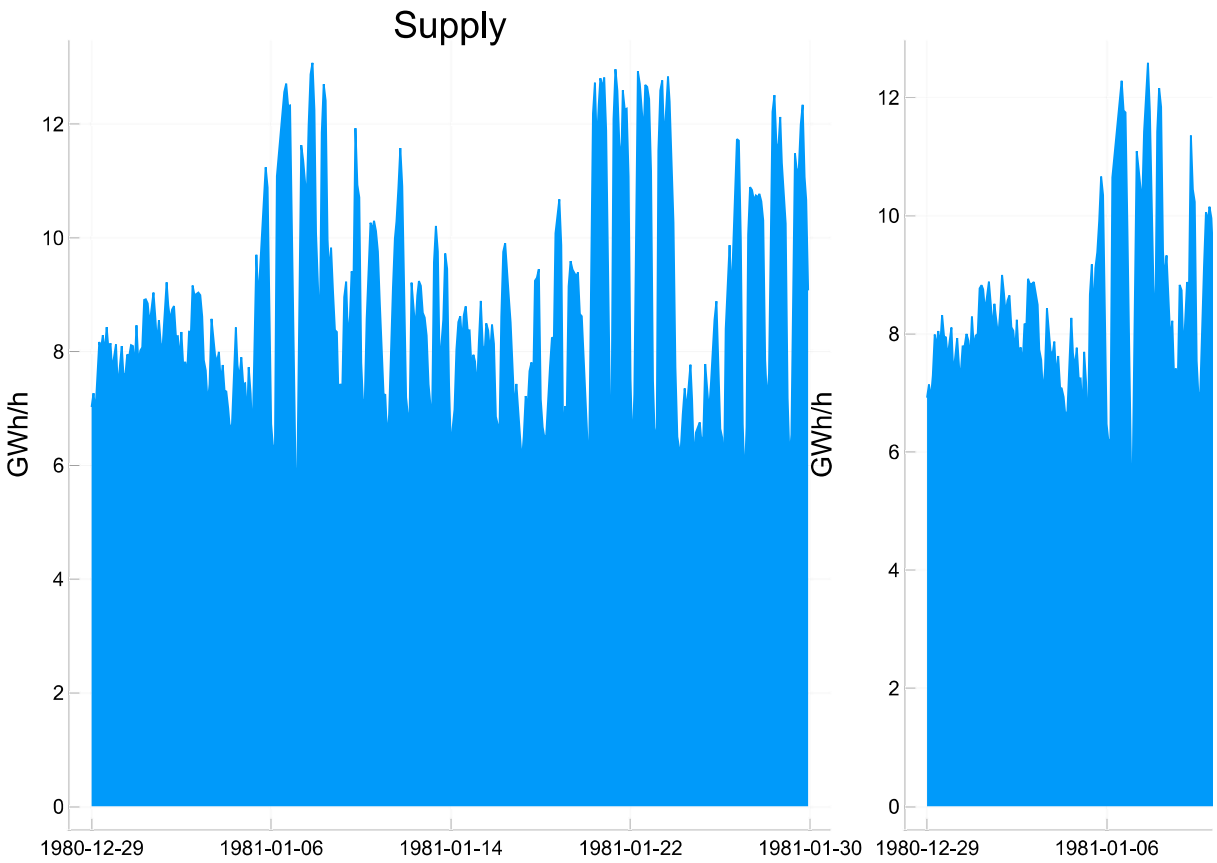
```
# # Store results
# data = Dict()
# data["areanames"] = areanames
# data["pricematrix"] = pricematrix
# data["priceindex"] = ix

# data["statenames"] = statenames
# data["statematrix"] = transpose(statematrix)
# data["stateindex"] = [getisoyearstart(horizonstart) + Day(2*(i-1)) for i in 1:first(size

# data["supplyvalues"] = supplyvalues
# data["supplynames"] = supplynames
# data["supplybalancenames"] = supplybalancenames

# open("JulES_2030_81.json", "w") do f
#     write(f, JSON.json(data))
# end

# areaprices = rename!(DataFrame(prices, :auto),powerbalancenames)
# areaprices[!,:time] = x1
# CSV.write("JulES_results\\prices.csv", areaprices)

# demand = rename!(DataFrame(demandvalues, :auto),demandnames)
# demand[!,:time] = x1
# demand = stack(demand,Not(:time))
# demandcopl = DataFrame(variable=demandnames, area=demandbalancenames)
# demand = leftjoin(demand, demandcopl, on=:variable)
# CSV.write("JulES_results\\demand.csv", demand)
```

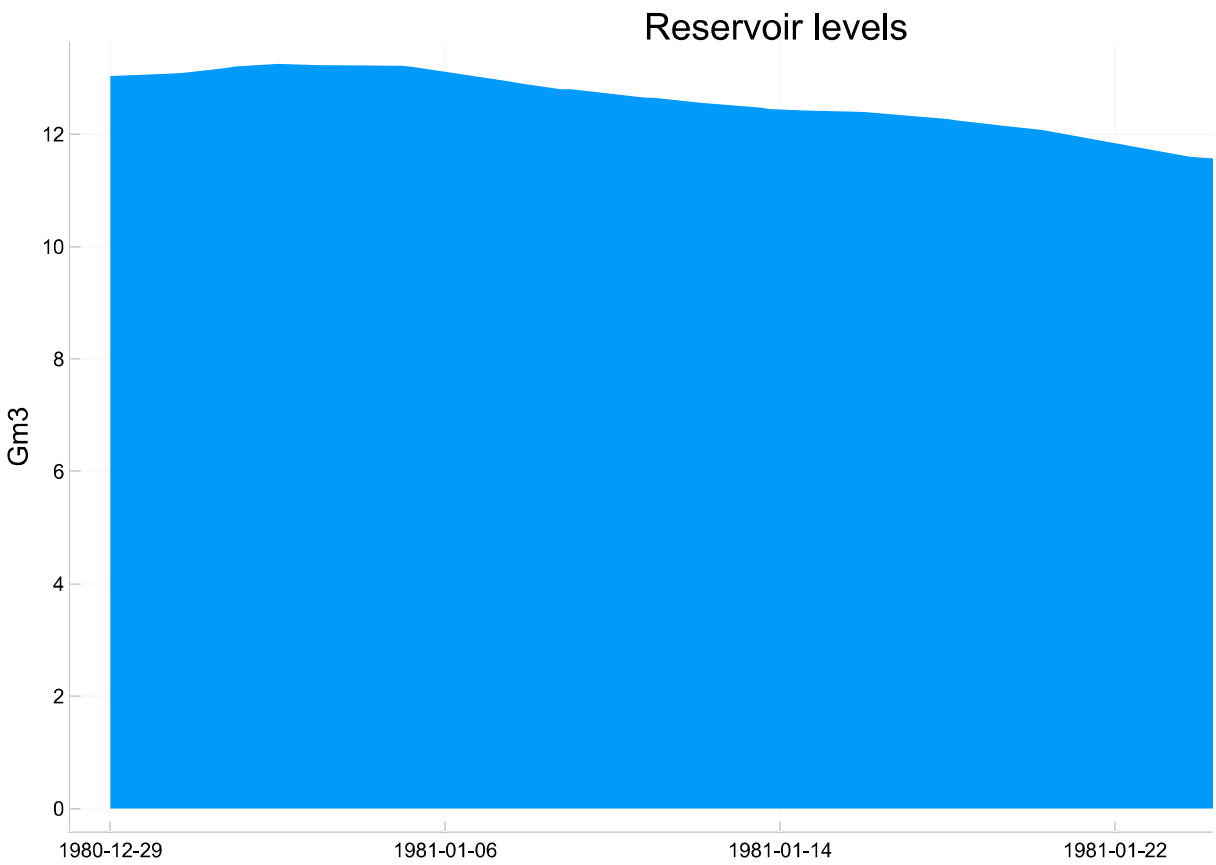
```
# supply = rename!(DataFrame(supplyvalues, :auto),supplynames)
# supply[!,:time] = x1
# supply = stack(supply,Not(:time))
# supplycopl = DataFrame(variable=supplynames, area=supplybalancenames)
# supply = leftjoin(supply, supplycopl, on=:variable)
# CSV.write("JulES_results\\supply.csv", supply)

# hydro = rename!(DataFrame(hydrolevels, :auto),hydronames)
# hydro[!,:time] = x2
# CSV.write("JulES_results\\hydro.csv", hydro);
```

In [237…

```
# Plot prices
display(plot(x1, prices, labels=reshape(powerbalancenames,1,length(powerbalancenames)), si

# # Plot supplies and demands
# supplychart = areaplot(x1, supplyvalues,labels=reshape(supplynames,1,length(supplynames),
# demandchart = areaplot(x1, demandvalues,labels=reshape(demandnames,1,length(demandnames),
# display(plot([supplychart,demandchart]...,layout=(1,2),size=(1600,500)))
supplychart = areaplot(x1, sum(supplyvalues,dims=2),title="Supply", ylabel = "GWh/h")
demandchart = areaplot(x1, sum(demandvalues,dims=2),title="Demand", ylabel = "GWh/h")
display(plot([supplychart,demandchart]...,layout=(1,2),size=(900,500)))

# Plot storages
# display(areaplot(x2, hydrolevels,labels=reshape(hydronames,1,length(hydronames)),size=(80
display(areaplot(x2, dropdims(sum(hydrolevels,dims=2),dims=2),labels="Total reservoirs",si


# Plot list of yearly mean production and demand for each supply/demand
meandemand = dropdims(mean(demandvalues,dims=1),dims=1)
meanproduction = dropdims(mean(supplyvalues,dims=1),dims=1)
supplydf = sort(DataFrame(Supplyname = supplynames, Yearly_supply_TWh = meanproduction*8.7
demanddf = sort(DataFrame(Demandname = demandnames, Yearly_demand_TWh = meandemand*8.76),[
supplydf[!,:ID] = collect(1:length(supplynames))
demanddf[!,:ID] = collect(1:length(demandnames))
joineddf = select!(outerjoin(supplydf,demanddf;on=:ID),Not(:ID))
# show(joineddf,allcols=true, allrows=true)

# Check that total supply equals total demand
show(combine(joineddf, [:Yearly_supply_TWh, :Yearly_demand_TWh] .=> sum∘skipmissing))
```


Prices
€/MWh
70
65
60
55
50
45
1980-12-29
1981-01-06
1981-01-14
1981-01-22


Supply
GWh/h
12
10
8
6
4
2
0
1980-12-29
1981-01-06
1981-01-14
1981-01-22
1981-01-30
GWh/h
12
10
8
6
4
2
0
1980-12-29
1981-01-06



1×2 DataFrame

Row	Yearly_supply_TWh_sum_skipmissing	Yearly_demand_TWh_sum_skipmissing
	Float64	Float64
1	78.0786	75.584

- Yearly_supply_TWh and Yearly_demand_TWh is the mean production timed with 8736.
- Transmission into the area is added to supply, while transmission out is added to demand.
- The reason why the supply does not match the demand is that the filtering does not split up watercourses where hydropower plants are in different areas. The supply side (of NO2) therefore contains 14 hydropower plants in NO1 and NO5.

In [ ]: