## 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 14 2.20 GHz processors running in parallel

In [2]:
```
const numcores = 14

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

@show nprocs();

@everywhere include(joinpath(dirname(pwd()),"jgrc/TuLiPa/src/TuLiPa.jl"))
@everywhere include("JulES/src/JulES.jl");
```

```
nprocs() = 14
```

## 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 [3]:
```
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 s
phaseindelta = Millisecond(Week(5)) # Phase in the second stage scenario over 5 wee
phaseinsteps = 5 # Phase in second stage scenario in 5 steps
tphasein = PhaseinTwoTime(getisoyearstart(datayear),getisoyearstart(horizonstart),
```

## 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 [4]:
```
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 [5]:
```
# Set horizons for price prognosis models
# Long
longhorizonend = getisoyearstart(horizonstart + 5)
longhydroperiodduration = Millisecond(Week(6))
longpowerparts = 6

longrhsdata = StaticRHSAHData("Power", datayear, scenarioyearstart, scenarioyearsto
longmethod = KMeansAHMethod()
longclusters = 4
longunitduration = Millisecond(Hour(6))

# Medium
medweeklength = 54
medhorizonend = getisoyearstart(horizonstart) + Week(medweeklength)
medhydroperiodduration = Millisecond(Day(7)); @assert medweeklength % Int(longhydro
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 % shortweekle
shortpowerparts = 12

# Preallocate results, distributed on cores
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)

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:numsce

# Organise inputs and outputs
probs = (longprobs, medprobs, shortprobs)
longinput = (longhorizonend, longhydroperiodduration, longrhsdata, longmethod, long
medinput = (medhorizonend, medhydroperiodduration, medrhsdata, medmethod, medcluste
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 p
@time pl_prognosis_init!(probs, allinput, longinput, medinput, shortinput, output)
```

```
 43.805738 seconds (3.75 M allocations: 184.031 MiB, 0.20% gc time, 2.74% compilat
ion time)
```

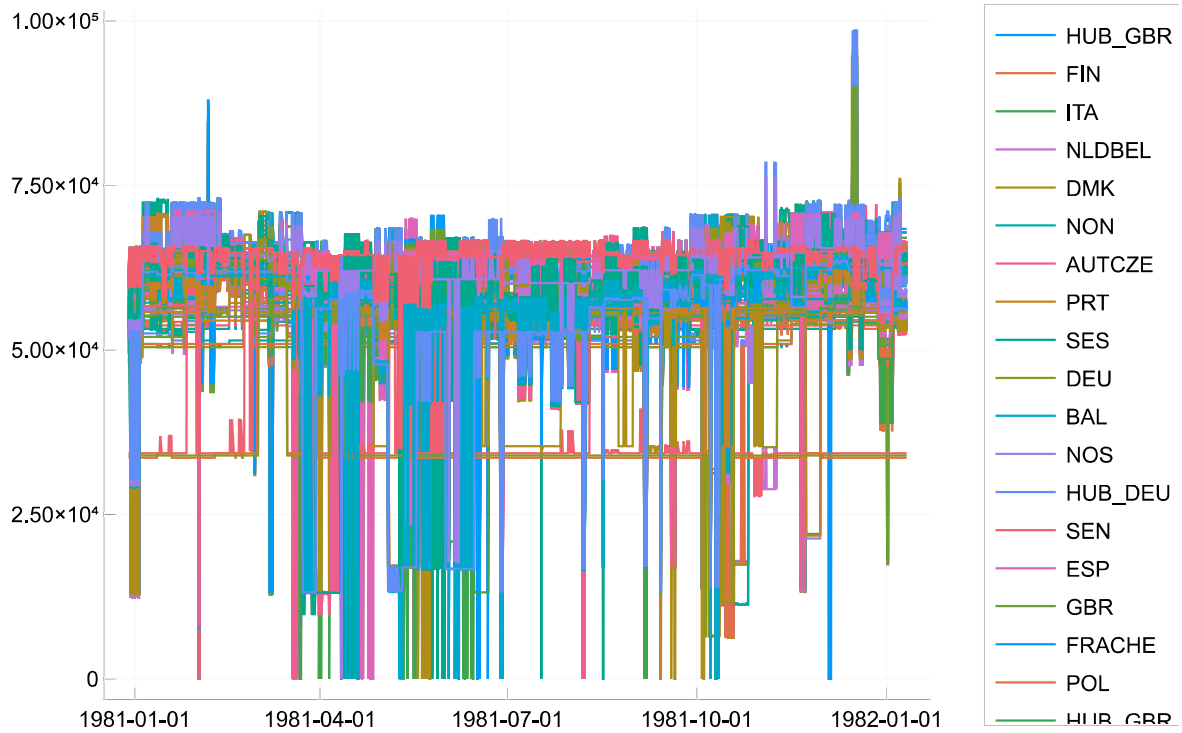
Out[5]:
```
Task (done) @0x0000000098f73e10
```

## Plot medium term prices for the 7 scenarios

In [6]:
```
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=:outertoprigl

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=:outertopr
end
display(p)
```

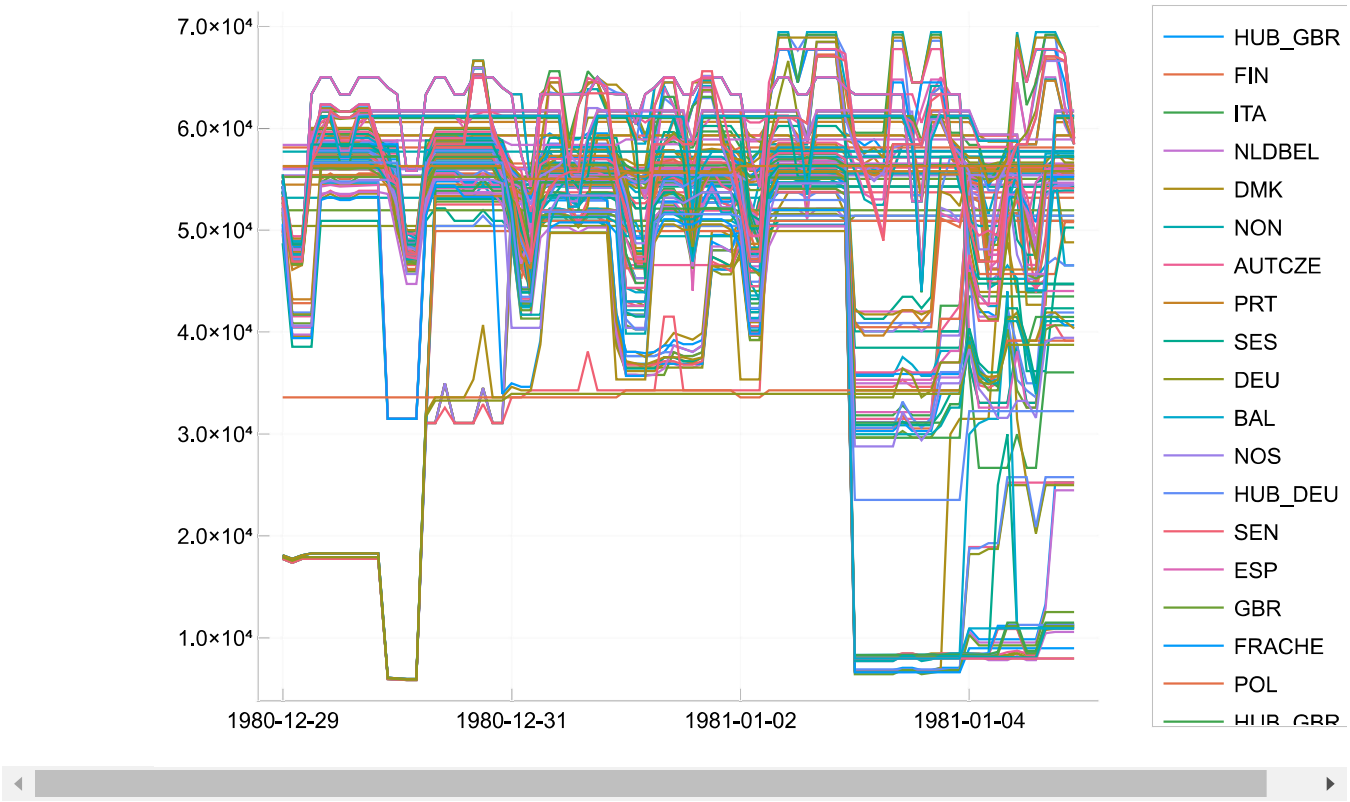



## Plot short term prices for the 7 scenarios

In [7]:

```
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=:outertoprigl

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=:outertop
end
display(p)
```




## Stochastic subsystem models and market clearing model uses more detailed data

Thema dataset (explained above) combined with detailed hydropower from for Norway and 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
- 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 [8]:
```
# 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")
```

```
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 [9]:
```
detailedrescopl = JSON.parsefile("data_fra_dynmodell/magasin_elspot1.json")
for k in keys(detailedrescopl) # quick fix, detailed dataset has reservoirs for SE4
    if detailedrescopl[k] == "SE4"
        detailedrescopl[k] = "SE3"
    end
end

scenario = 1
endvaluescopl = Dict()
instance = [getinstancename(getid(obj)) for obj in medendvaluesobjs[scenario].obje
endvalues = medendvaluesobjs[scenario].values
endvaluescopl = Dict(instance .=> endvalues)

for (k,v) in detailedrescopl
    endvaluescopl["Reservoir_" * k] = endvaluescopl["Reservoir_" * v * "_hydro_res
end
```

## 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 [10]:
```
# 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 t
shortstartstorage = 50
medstartstorage = 65
storageinfo = (shortstartstorage, medstartstorage, endvaluescopl)
shortterminputs = (detailedelements, shorttotaldays, numscen, horizonstart, phasein
medterminputs = (detailedelements, medtotaldays, numscen, horizonstart, phaseinoffs

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)], storag
subs = distribute([[] for i in 1:length(storagesystemobjects)], storagesystemobject
states = distribute([Dict{StateVariableInfo, Float64}() for i in 1:length(storagesy
cuts = distribute([SimpleSingleCuts() for i in 1:length(storagesystemobjects)], stc
storagesystems = distribute([Dict() for i in 1:length(storagesystemobjects)], stora

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

```
 11.533659 seconds (23.26 M allocations: 1.066 GiB, 2.77% gc time, 74.50% compilat
ion time)
  3.094866 seconds (11.98 M allocations: 485.123 MiB, 6.09% gc time, 2.63% compila
tion time)
 63.126281 seconds (893.38 k allocations: 49.954 MiB, 0.44% compilation time)
```

Out[10]:
```
Task (done) @0x0000000078de6100
```

## 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 [11]:
```
# 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
clearing, clearingstorages, nonstoragestatesmean, clearingendvaluesdict = clearing_
```

In [12]:
```
# Initialize start states for next time step, also mapping to Thema storages and ma
startstates, enekvglobaldict = startstates_init(clearing, detailedrescopl, longprol

# 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));
```

## Move to the next time step

In [13]:
```
tnormal += Day(2)
tphasein = PhaseinTwoTime(getdatatime(tnormal), getscenariotime(tnormal), getscena
```

## Run price prognosis models for new time step

In [14]:
```
@time pl_prognosis!(numcores, longprobs, medprobs, shortprobs, medprices, shortpri
```

```
  3.745796 seconds (133.46 k allocations: 7.489 MiB, 1.52% compilation time)
```

Out[14]:
```
Task (done) @0x0000000097a4e2f0
```

## Run stochastic sub systems for new time step

In [15]:
```
medpriceslocal = convert(Vector{Dict}, medprices)
shortpriceslocal = convert(Vector{Dict}, shortprices)
medendvaluesdict = getmedendvaluesdict(medendvaluesobjs)

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

```
 13.893554 seconds (540.69 k allocations: 29.703 MiB, 1.02% compilation time)
```

Out[15]:
```
Task (done) @0x0000000098c1f650
```

## Run market clearing model for new time step

In [16]:
```
cutslocal = convert(Vector{SimpleSingleCuts}, cuts)
nonstoragestateslocal = convert(Vector{Dict}, nonstoragestates)

@time startstates = clearing!(clearing, clearingstorages, startstates, cutslocal, 

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

```
  1.478443 seconds (1.33 M allocations: 43.751 MiB, 23.52% compilation time)
```

## Simulate next time steps

- Simulate next 24 time steps and store results

In [17]:
```
# Only update medterm and longterm every 14 days (med/long prices and med/long wate
skipmed = Millisecond(0)
tnormal += Day(2)
tphasein = PhaseinTwoTime(getdatatime(tnormal), getscenariotime(tnormal), getscena

steps = 24
# steps = (getisoyearstart(scenarioyearstop) - getisoyearstart(scenarioyearstart))
totaltime = @elapsed for i in 1:steps
# totaltime = @elapsed while getscenariotime(tnormal) < getisoyearstart(scenarioyea
```

```
    display(tnormal)
    # Deterministic long/mid/short - calculate scenarioprices
    @time pl_prognosis!(numcores, longprobs, medprobs, shortprobs, medprices, shor

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

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

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

    @time startstates = clearing!(clearing, clearingstorages, startstates, cutsloc

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

    skipmed += Millisecond(Day(2))
    if skipmed > Millisecond(Day(14))
        skipmed = Millisecond(0)
    end

    tnormal += Day(2)
    tphasein = PhaseinTwoTime(getdatatime(tnormal), getscenariotime(tnormal), gets
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("1981-01-02T00:00:00"))
  3.250508 seconds (31.73 k allocations: 2.177 MiB)
 12.420668 seconds (73.22 k allocations: 4.185 MiB)
  1.003028 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-04T00:00:00"))
  1.794800 seconds (31.71 k allocations: 2.175 MiB)
  0.442913 seconds (72.89 k allocations: 4.173 MiB)
  0.879865 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-06T00:00:00"))
  1.549399 seconds (31.70 k allocations: 2.175 MiB)
  0.791747 seconds (72.90 k allocations: 4.174 MiB)
  0.960900 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-08T00:00:00"))
  1.341306 seconds (31.79 k allocations: 2.183 MiB)
  0.425484 seconds (72.89 k allocations: 4.173 MiB)
  0.925187 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-10T00:00:00"))
  1.198269 seconds (31.79 k allocations: 2.183 MiB)
  1.215532 seconds (72.91 k allocations: 4.175 MiB)
  0.952580 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-12T00:00:00"))
  1.553951 seconds (31.80 k allocations: 2.183 MiB)
  0.582069 seconds (73.07 k allocations: 4.185 MiB)
  0.936429 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-14T00:00:00"))
```

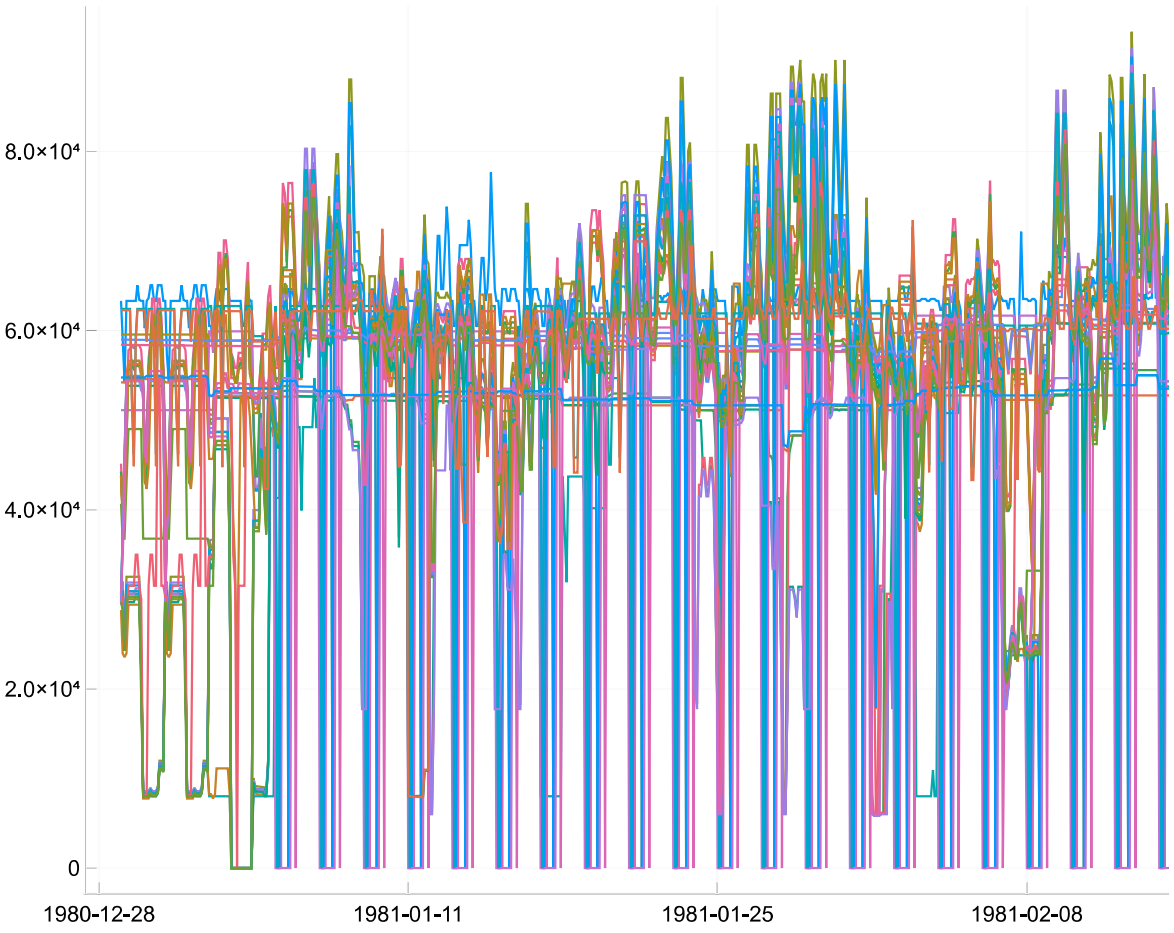
```
  1.872522 seconds (31.71 k allocations: 2.175 MiB)
  0.636750 seconds (72.89 k allocations: 4.173 MiB)
  1.084081 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-16T00:00:00"))
  2.199703 seconds (31.72 k allocations: 2.177 MiB)
  0.428263 seconds (72.89 k allocations: 4.173 MiB)
  1.169123 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-18T00:00:00"))
  4.024766 seconds (31.85 k allocations: 2.186 MiB)
 13.128075 seconds (73.24 k allocations: 4.186 MiB)
  1.007780 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-20T00:00:00"))
  2.080653 seconds (31.82 k allocations: 2.185 MiB)
  0.484461 seconds (72.89 k allocations: 4.173 MiB)
  0.990579 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-22T00:00:00"))
  2.316381 seconds (31.73 k allocations: 2.177 MiB)
  0.419515 seconds (72.89 k allocations: 4.173 MiB)
  0.917005 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-24T00:00:00"))
  2.043056 seconds (31.72 k allocations: 2.176 MiB)
  0.484071 seconds (72.89 k allocations: 4.173 MiB)
  1.184006 seconds (1.02 M allocations: 26.250 MiB, 14.96% gc time)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-26T00:00:00"))
  1.706002 seconds (31.96 k allocations: 2.183 MiB)
  0.424702 seconds (72.89 k allocations: 4.173 MiB)
  1.058249 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-28T00:00:00"))
  1.285755 seconds (31.70 k allocations: 2.175 MiB)
  0.453153 seconds (72.89 k allocations: 4.173 MiB)
  0.955979 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-01-30T00:00:00"))
  1.792949 seconds (31.81 k allocations: 2.185 MiB)
  0.457751 seconds (72.89 k allocations: 4.175 MiB)
  1.067862 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-01T00:00:00"))
  2.113679 seconds (31.72 k allocations: 2.176 MiB)
  0.461590 seconds (72.90 k allocations: 4.175 MiB)
  1.099243 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-03T00:00:00"))
  4.282540 seconds (31.85 k allocations: 2.186 MiB)
 12.842496 seconds (73.24 k allocations: 4.187 MiB)
  1.104493 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-05T00:00:00"))
  2.108224 seconds (31.72 k allocations: 2.176 MiB)
  0.505923 seconds (75.53 k allocations: 4.378 MiB, 29.15% gc time)
  1.149739 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-07T00:00:00"))
  2.309945 seconds (31.82 k allocations: 2.185 MiB)
  0.464392 seconds (72.89 k allocations: 4.175 MiB)
  1.116108 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-09T00:00:00"))
  2.297468 seconds (31.82 k allocations: 2.185 MiB)
  0.467817 seconds (72.89 k allocations: 4.173 MiB)
  1.174768 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-11T00:00:00"))
  2.464477 seconds (31.73 k allocations: 2.177 MiB)
  0.454079 seconds (72.89 k allocations: 4.173 MiB)
  1.086572 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-13T00:00:00"))
  2.316104 seconds (31.82 k allocations: 2.185 MiB)
  0.502919 seconds (72.89 k allocations: 4.173 MiB)
  1.008278 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-15T00:00:00"))
```

```
  2.167772 seconds (31.72 k allocations: 2.177 MiB)
  0.433954 seconds (72.89 k allocations: 4.173 MiB)
  1.001513 seconds (1.02 M allocations: 26.095 MiB)
FixedDataTwoTime(DateTime("2024-12-30T00:00:00"), DateTime("1981-02-17T00:00:00"))
  2.171829 seconds (31.82 k allocations: 2.185 MiB)
  0.512882 seconds (72.89 k allocations: 4.175 MiB)
  1.057305 seconds (1.02 M allocations: 26.095 MiB)
"The simulation took: 2.2046029616666667 minutes"
"Time usage per timestep: 5.511507404166667 seconds"
```

## Plot resulting prices

In [19]:
```
plot(ix,pricematrix,label=reshape(areanames, 1, length(areanames)),size=(800,500),
```

Out[19]:




## Save results

In [18]:
```
# 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:fi

# open("JulES_2030_81_10.json", "w") do f
#     write(f, JSON.json(data))
# end
```

In [ ]: