In [17]:
```julia
using DataFrames, Plots, Statistics, JSON, Clp
plotlyjs()
include(joinpath(dirname(pwd()),"src/TuLiPa.jl")); # using Dates, JuMP, HiGHS, CSV,
```

Demo 6 - Two-stage stochastic hydro with Benders decomposition

This demo solves the same problem as demo 5, but this time we use Benders decomposition.

The guide below describes the method used in this demo. We use Benders decomposition for a two-stage minimization problem.

- https://jump.dev/JuMP.jl/stable/tutorials/algorithms/benders_decomposition/

The problem is split into a master problem (V1, first stage problem) and subproblems (V2, second stage scenarios).

- The master problem has a future value variable (θ) in the objective function, which represents the weighted objective value of the subproblems (V2(x)). The future value variable in the master problem is restricted by Benders cuts generated from the subproblems.
  - Benders cut: θ - $\bar{\pi}$ x >= $V2(\bar{x})$ - $\bar{\pi}$ $\bar{x}$ where
    - θ is the future value variable
    - $\bar{\pi}$ is the water value of the end reservoir (solution from subproblems)
    - x is the end reservoir variable
    - V2($\bar{x}$) is the weighted objective value of the subproblems (solution from subproblems)
    - $\bar{x}$ is the end reservoir from the previous solved master problem (solution from previous master problem)
- The subproblems use the end reservoirs ($\bar{x}$) from the master problem as start reservoirs.
- This is an iterative process where the future value variable converges towards the weighted subproblem objective value
- The iteration is stopped when the relative difference between lb = θ and ub = V2($\bar{x}$) is under a tolerance factor.

Make modelobjects for the master and subproblems

In [2]:
```julia
function getdataelements()
    # Read dataelements from json-files
    sti_dynmodelldata = "dataset_vassdrag"
    price = JSON.parsefile("priceDMK.json")
    detdprice = getelements(price);
    tidsserie = JSON.parsefile(joinpath(sti_dynmodelldata, "tidsserier_detd.json"))
    detdseries = getelements(tidsserie, sti_dynmodelldata);
    dst = JSON.parsefile(joinpath(sti_dynmodelldata, "dataset_detd_AURLAND_H.json"
    detdstructure = getelements(dst);

    elements = vcat(detdseries,detdprice,detdstructure)
```

```julia
    # Select which scenarios to include from the time-series
    scenarioyearstart = 1981
    scenarioyearstop = 1996 # price series only goes to 1995
    push!(elements, getelement(TIMEPERIOD_CONCEPT, "ScenarioTimePeriod", "Scenario
            ("Start", getisoyearstart(scenarioyearstart)), ("Stop", getisoyearstar

    # Add an exogenous price area that the plants and pumps can interact with. All
    push!(elements, getelement(BALANCE_CONCEPT, "ExogenBalance", "PowerBalance_NO5
            (COMMODITY_CONCEPT, "Power"),
            (PRICE_CONCEPT, "PriceDMK")))
    return elements
end

function makemodelobjects(elements, weeks::Int,offset::Union{Offset,Nothing})
    # Add horizons to the dataset
    hydro_horizon = SequentialHorizon(weeks, Hour(168); offset)
    power_horizon = SequentialHorizon(7*weeks, Hour(24); offset)
    push!(elements, getelement(COMMODITY_CONCEPT, "BaseCommodity", "Power",
            (HORIZON_CONCEPT, power_horizon)))
    push!(elements, getelement(COMMODITY_CONCEPT, "BaseCommodity", "Hydro",
            (HORIZON_CONCEPT, hydro_horizon)))

    # Generate modelobjects from dataelements and add boundary conditions to storag
    return getmodelobjects(elements)
end;
```

In [3]:

```julia
# Total problem length is 105 weeks = approx 2 years, and first stage problem is e
totalweeks = 105
firstweeks = 8

# Read dataelements that will be in both the first stage and second stage problems
elements = getdataelements()

# Make modelobjects for first stage problem
firstobjects = makemodelobjects(copy(elements), firstweeks, nothing)

# Make modelobjects for 10 second stage scenarios. Each scenario start eight weeks
numscen = 10

secondobjects = []
for i in 1:numscen
    scenarioyearstart = 1981+i-1 # Scenarios start in 1981, 1982, etc...
    offset = IsoYearOffset(scenarioyearstart, MsTimeDelta(Week(firstweeks)))
    push!(secondobjects, makemodelobjects(copy(elements), totalweeks-firstweeks, o
end
```

Problem start time

In [4]:

```julia
# Problem start time
datayear = getisoyearstart(2025)
scenarioyear = getisoyearstart(1981)

# We use a PhaseinTwoTime to get smooth transitions between the first stage and se
phaseinoffset = Millisecond(Week(firstweeks)) # phase in straight away from second
phaseindelta = Millisecond(Week(26)) # Phase in the second stage scenario over hal
phaseinsteps = 25 # Phase in second stage scenario in 25 steps

t = PhaseinTwoTime(datayear, scenarioyear, scenarioyear, phaseinoffset, phaseindel
```

Initialize masterproblem

In [5]:
```
# Get cutobjects
function getcutobjects(modelobjects)
    cutobjects = Vector{Any}()
    for (id,obj) in modelobjects
        if hasstatevariables(obj)
            if length(getstatevariables(obj)) > 1
                error("Not supported")
            else
                push!(cutobjects,obj)
            end
        end
    end
    return cutobjects
end

# Initialize cuts
function initialize_cuts!(modelobjects, cutobjects, maxcuts, lb, numscen)
    # Make a cutid
    cutid = Id(BOUNDARYCONDITION_CONCEPT,"StorageCuts")

    # Probability of each subproblem / second stage scenario
    probabilities = [1/numscen for i in 1:numscen]

    # Make cut modelobject
    cuts = SimpleSingleCuts(cutid, cutobjects, probabilities, maxcuts, lb)
    modelobjects[cutid] = cuts
    return cuts
end

# Initialize list of the statevariables that connects the master and subproblem. Fo
# variables (here reservoirs), we store the state variables of the master problem,
# (end storage in master and start storage in sub). (We do not need to store the he
# subproblems seperately, since they have the same id and index as the initial stat
function getstatevariables(cutobjects::Vector{Any})
    states = Dict{StateVariableInfo, Float64}()

    for obj in cutobjects
        for statevariable in getstatevariables(obj)
            states[statevariable] = 0.0
        end
    end
    return states
end

maxcuts = 13 # preallocate fixed number of cuts, no cut selection
lb = -1e10 # lower bound of the future value in the first iteration
cutobjects = getcutobjects(firstobjects)
cuts = initialize_cuts!(firstobjects, cutobjects, maxcuts, lb, numscen);
states = getstatevariables(cutobjects) # state variables in master and subs for bou

# Build problem and update parameters
master = HiGHS_Prob(firstobjects)
update!(master, t)

# Set start reservoir as a percentage of capacity
function setstartstorage!(prob, start, percentage)
    for obj in prob.objects
        if obj isa Storage

            dummydelta = MsTimeDelta(Millisecond(0))
```

```
            startreservoir = getparamvalue(getub(obj), start, dummydelta)*percenta

            ingoingstates = Dict{StateVariableInfo, Float64}()
            for statevariable in getstatevariables(obj)
                ingoingstates[statevariable] = startreservoir
            end

            setingoingstates!(prob, ingoingstates)
        end
    end
end

function setendstorage!(prob, endtime, percentage)
    for obj in prob.objects
        if obj isa Storage

            dummydelta = MsTimeDelta(Millisecond(0))
            endreservoir = getparamvalue(getub(obj), endtime, dummydelta)*percenta

            outgoingstates = Dict{StateVariableInfo, Float64}()
            for statevariable in getstatevariables(obj)
                outgoingstates[statevariable] = endreservoir
            end

            setoutgoingstates!(prob, outgoingstates)
        end
    end
end

percentage = 65
setstartstorage!(master, t, percentage)
```

Solve masterproblem and query results

In [6]:
```
solve!(master)
getobjectivevalue(master)
```

Out[6]:
```
-1.0097083907807703e10
```

In [7]:
```
lb = getvarvalue(master, getfuturecostvarid(cuts),1)
```

Out[7]:
```
-1.0e10
```

Initialize, solve and query subproblems

In [8]:
```
# Transfer master problem end reservoir to subproblems (start reservoir)
function transferboundarystorage!(master, sub, states)
    states = getoutgoingstates!(master, states)
    setingoingstates!(sub, states)
end

subs = [HiGHS_Prob(secondobject) for secondobject in secondobjects] # initialize su
ub = 0 # upper bound is a sum of weighted subproblem objective values
cutparameters = Vector{Tuple{Float64, Dict{StateVariableInfo, Float64}}}() # preal

for (i,sub) in enumerate(subs)

    update!(sub, t) # update parameters given problem start time of scenario

    transferboundarystorage!(master, sub, states) # set start reservoir
```

```
    setendstorage!(sub, t + MsTimeDelta(Week(totalweeks-firstweeks)), percentage) #

    solve!(sub)

    ub += getobjectivevalue(sub)
    push!(cutparameters, getcutparameters(sub, states))
end

ub /= numscen # subproblems are weighted equally
```

Out[8]:

```
-2.3253547626743326e8
```

Check convergence

The convergence criteria calculates a relative difference from the upper and lower bound of the future cost, and compares it to a relative tolerance.

- The lower bound is the future cost from the master problem.
- The upper bound is the weighted objective values from the subproblems.

In [9]:

```
reltol = 0.0001 # relative tolerance
display(abs((lb-ub)/ub)) # relative difference
display(abs((lb-ub)/ub) < reltol) # convergence?
```

```
42.00419084655818
false
```

Update cuts

In [10]:

```
updatecuts!(master, cuts, cutparameters)
```

Iterate until convergence

In [14]:

```
@time while abs((lb-ub)/lb) > reltol
    solve!(master)
    lb = getvarvalue(master, getfuturecostvarid(cuts),1)
    masterlb = getobjectivevalue(master)
    cx = masterlb - lb

    masterub = 0
    ub = 0
    for (i,sub) in enumerate(subs)

        transferboundarystorage!(master, sub, states)

        solve!(sub)

        ub += getobjectivevalue(sub)
        cutparameters[i] = getcutparameters(sub, states)
    end
    ub /= numscen
    masterub = min(masterub,ub+cx)
    updatecuts!(master, cuts, cutparameters)
    display(string("Cut: ", cuts.cutix))
    display("Master lb and ub:")
    display(masterlb)
    display(masterub)
    display("Future value lb and ub, and relative difference:")
    display(lb)
```

```julia
    display(ub)
    display((ub-lb)/lb)
#     plot_results()
end
```

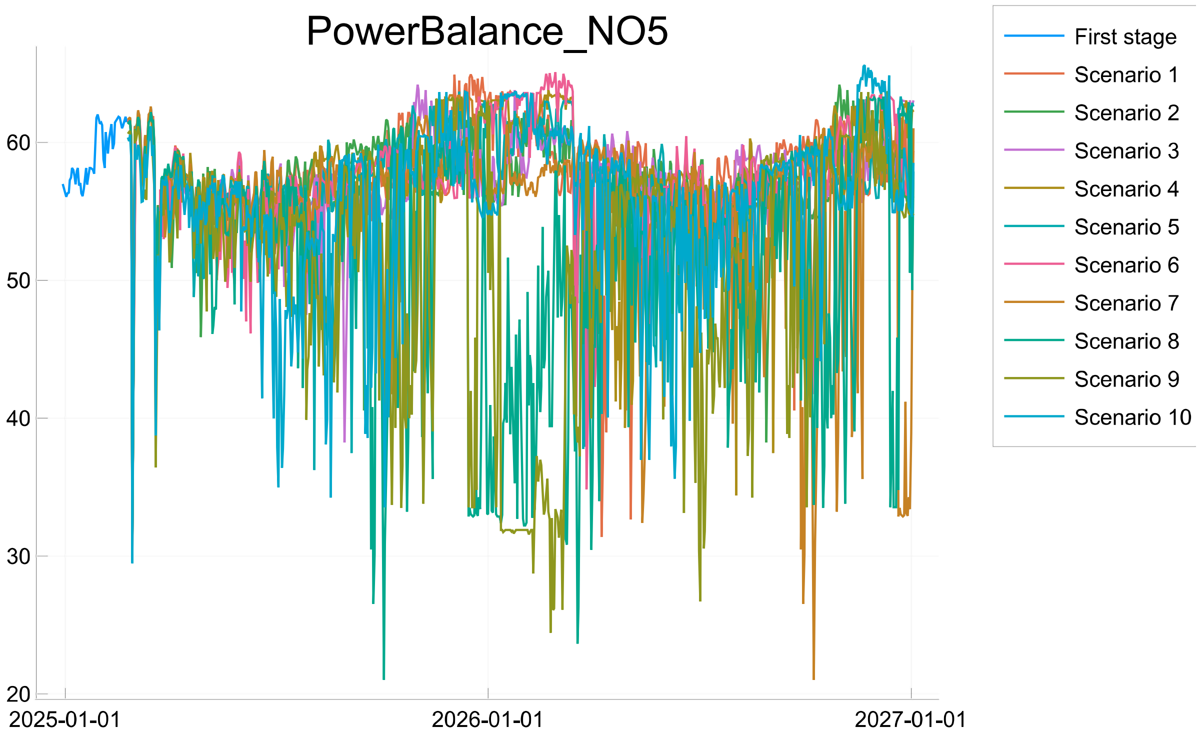
```
"Cut: 2"
"Master lb and ub:"
-3.4664865994264245e8
-3.4238840839009553e8
"Future value lb and ub, and relative difference:"
-3.395481248463214e8
-3.352878732937745e8
-0.012546826917318708
"Cut: 3"
"Master lb and ub:"
-3.445445738941054e8
-3.4355800566390884e8
"Future value lb and ub, and relative difference:"
-2.833639703879706e8
-2.823774021577741e8
-0.0034816290470724483
"Cut: 4"
"Master lb and ub:"
-3.4434578872186124e8
-3.4388711145929605e8
"Future value lb and ub, and relative difference:"
-2.93615235491081e8
-2.931565582285158e8
-0.001562171192506557
"Cut: 5"
"Master lb and ub:"
-3.440836129535467e8
-3.4388712264462703e8
"Future value lb and ub, and relative difference:"
-3.08812985400187e8
-3.0861649509126735e8
-0.0006362760577086446
"Cut: 6"
"Master lb and ub:"
-3.440405860821565e8
-3.4396550555580825e8
"Future value lb and ub, and relative difference:"
-3.058362375226578e8
-3.057611569963096e8
-0.0002454925778462433
"Cut: 7"
"Master lb and ub:"
-3.439760259255129e8
-3.4392871795046616e8
"Future value lb and ub, and relative difference:"
-2.9790786409265804e8
-2.978605561176113e8
-0.0001588006922571192
"Cut: 8"
"Master lb and ub:"
-3.4397360972046846e8
-3.4394975370726174e8
"Future value lb and ub, and relative difference:"
-3.00384316726293 9e8
-3.003604607130872e8
-7.941830474610838e-5
```

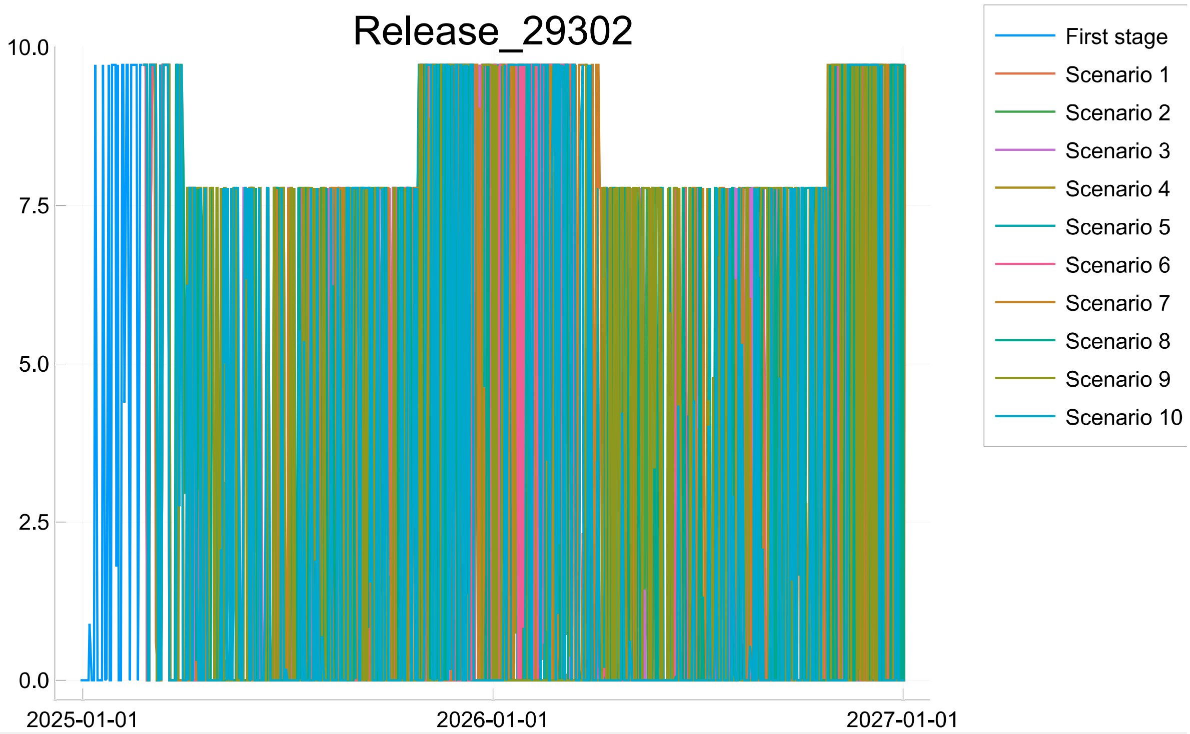
Plot some results

In [15]:
```
function plot_var(master, subs, id, datayear)
    obj = firstobjects[id]
    horizon = gethorizon(obj)

    x = [datayear + getstartduration(horizon, t) for t in 1:getnumperiods(horizon)
    y = [getvarvalue(master, id, t) for t in 1:getnumperiods(horizon)]
    plot(x,y,label="First stage", title=getinstancename(id))

    for i in 1:numscen
        obj = secondobjects[i][id]
        horizon = gethorizon(obj)
        x1 = [datayear + Millisecond(Week(firstweeks)) + getstartduration(horizon,
        y1 = [getvarvalue(subs[i], id, t) for t in 1:getnumperiods(horizon)]
        plot!(x1,y1,label=string("Scenario ", i),legend=:outertopright)
    end

    display(plot!())
end

function plot_price(master, subs, id, t)
    obj = firstobjects[id]
    horizon = gethorizon(obj)
    price = getprice(obj)
    datayear = getdatatime(t)

    probtimes = [t + getstartduration(horizon, j) for j in 1:getnumperiods(horizon
    x = [datayear + getstartduration(horizon, j) for j in 1:getnumperiods(horizon)
    y = [getparamvalue(price, probtimes[j], gettimedelta(horizon, j)) for j in 1:g
    plot(x,y,label="First stage",title=getinstancename(id),legend=:outertopright)

    for i in 1:numscen
        obj = secondobjects[i][id]
        horizon = gethorizon(obj)
        probtimes1 = [getoffsettime(t,getoffset(horizon)) + getstartduration(horiz
        x1 = [datayear + Millisecond(Week(firstweeks)) + getstartduration(horizon,
        y1 = [getparamvalue(price, probtimes1[j], gettimedelta(horizon, j)) for j
        plot!(x1,y1,label=string("Scenario ", i))
    end

    display(plot!())
end

function plot_results()
    concept = BALANCE_CONCEPT
    instance = "PowerBalance_NO5"
    id = Id(concept, instance)

    plot_price(master, subs, id, t)

    concept = STORAGE_CONCEPT
    instance = "Reservoir_29302"
    id = Id(concept, instance)

    plot_var(master, subs, id, datayear)

    concept = FLOW_CONCEPT
    instance = "Release_29302"
    id = Id(concept, instance)

    plot_var(master, subs, id, datayear)
end
```

Out[15]: plot_results (generic function with 1 method)

In [16]:
```
plot_results()
```




Release_29302
10.0
7.5
5.0
2.5
0.0
2025-01-01
2026-01-01
2027-01-01
First stage
Scenario 1
Scenario 2
Scenario 3
Scenario 4
Scenario 5
Scenario 6
Scenario 7
Scenario 8
Scenario 9
Scenario 10

In [ ]: