



# Applied optimization with JuMP at SINTEF

Truls Flatberg



# Outline

- Background, SINTEF, Julia
- How to work with large scale, complex optimization models
- Illustrative examples showing two different approaches
- Learning, experiences and how to progress



SINTEF

ONE OF EUROPE'S LARGEST **INDEPENDENT**  
**RESEARCH ORGANISATIONS**

**367 million**

EUR turnover

**2200**

employees

**6400**

projects

**3300**

customers

INTERNATIONAL

**57 million EUR**

NATIONALITIES

**80**

PUBLICATIONS (INCL. DISSEMINATION)

**6200**

CUSTOMER SATISFACTION

**4,6 / 5**



# Optimization modeling @SINTEF

- Developing optimization models for industrial use and economic analysis
- Historically the primary tool has been FICO Xpress using the Mosel modelling language
- Other modeling tools: GAMS, AMPL, Pyomo
- Several models in daily operation across several industries (oil and gas, aluminium)
- Economic models for long term analysis (energy, infrastructure, regional, national and global focus)



SINTEF

# Why Julia and JuMP?

- Specialized algebraic modelling languages typically lack support for:
  - Easy modularization and code reuse
  - Support for multiple solvers
  - Ecosystem of surrounding packages for file IO, database access, plotting
- Commercial modeling tools are costly, and some are tied to a specific solver
- Research funding (EU and Research Council of Norway) tending towards more openly available models and software
- Julia and JuMP gradually introduced since 2020



# Open packages released by SINTEF

- SparseVariables: <https://github.com/sintefore/SparseVariables.jl>
- UnitJuMP: <https://github.com/trulsf/UnitJuMP.jl>
- TimeStruct: <https://github.com/sintefore/TimeStruct.jl>
- EnergyModelsX: <https://github.com/orgs/EnergyModelsX>
- PiecewiseAffineApprox: <https://github.com/sintefore/PiecewiseAffineApprox.jl>

For a short introduction to the packages see presentations at JuMP-Dev 2022, 2023 and 2024 (talks by Lars Hellemo and Julian Straus).



SINTEF

# How to handle complex optimization models

- Clear separation between input data and optimization modelling
- Multiple dispatch to control formulation and configuration
- Separate module(s) for the optimization model and associated data structures
- Automatic test routines
- Documentation
  
- Recommended reading
  - [https://jump.dev/JuMP.jl/stable/tutorials/getting\\_started/design\\_patterns\\_for\\_larger\\_models/](https://jump.dev/JuMP.jl/stable/tutorials/getting_started/design_patterns_for_larger_models/)



# Modeling approaches

- Single script – monolithic
- Julia supports more modular approaches, but the model is still "global" by nature
  
- What is the API of an optimization model?
- How can a user easily extend model functionality?
- How can we combine multiple models?





SINTEF

# Example 1: logistics of crushed stone and excavated masses

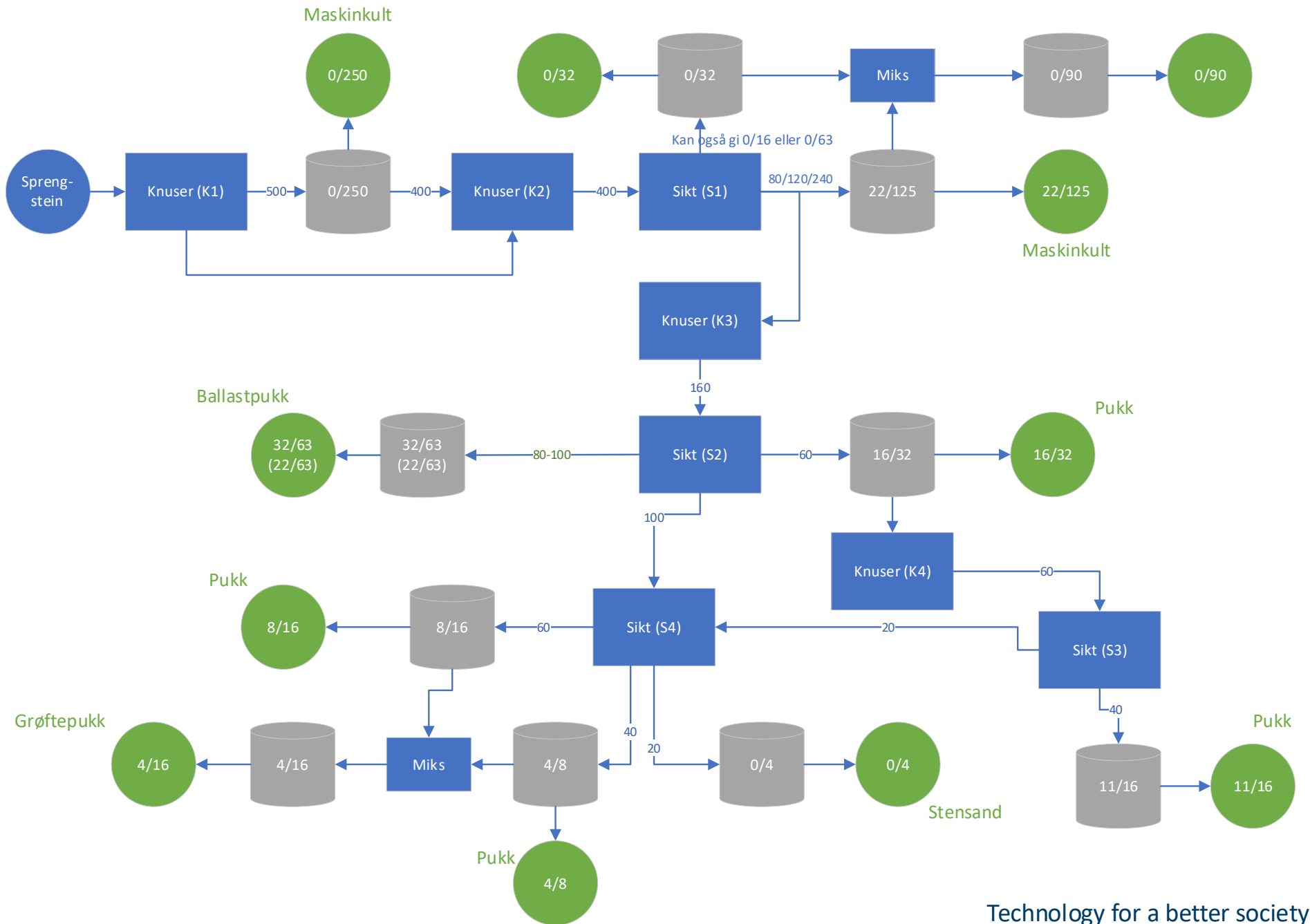




SINTEF

# Approach using JuMP

- **UnitJuMP** to ensure correct and flexible use of physical units
- **TimeStruct** for general time structures
- **SparseVariables** to allow dynamic generation of variables with sparse structure
- **MultiObjectiveAlgorithms** to balance economic and environmental aspects
- Separate packages for the optimization **modeling** and for setting up an **instance** based on input data





SINTEF

# Model variables

- Separate structure to collect all model variables
- Easier overview of available variables (including code completion)
- Using SparseVariables to allow dynamic variable creation
- Using UnitJuMP to include physical units

```
struct ModelVariables
    flow
    flow_mode
    n_trips
    usage_trip
    storage
    prod
    consume
    emit::Dict{Emission, IndexedVarArray}
    fuel::Dict{Fuel, IndexedVarArray}
    epd_flow::Dict{EPDIndicator, IndexedVarArray}
    usage
    usage_mode
```



SINTEF

UnitJuMP



```
@variable(m, flow[start =  $\mathcal{N}$ , stop =  $\mathcal{N}$ , product =  $\mathcal{P}$ , period =  $\mathcal{T}$ ] >= 0, massunit; container = IndexedVarArray)
@variable(m, flow_mode[node =  $\mathcal{N}$ , mode =  $\mathcal{D}$ , product =  $\mathcal{P}$ , period =  $\mathcal{T}$ ] >= 0, massunit; container = IndexedVarArray)

@variable(m, n_trips[trips =  $\mathcal{S}$ , period =  $\mathcal{T}$ ] >= 0; container = IndexedVarArray)
@variable(m, usage_trip[node =  $\mathcal{N}$ , trips =  $\mathcal{S}$ , period =  $\mathcal{T}$ ] >= 0, timeunit; container = IndexedVarArray)
```



SparseVariables



SINTEF

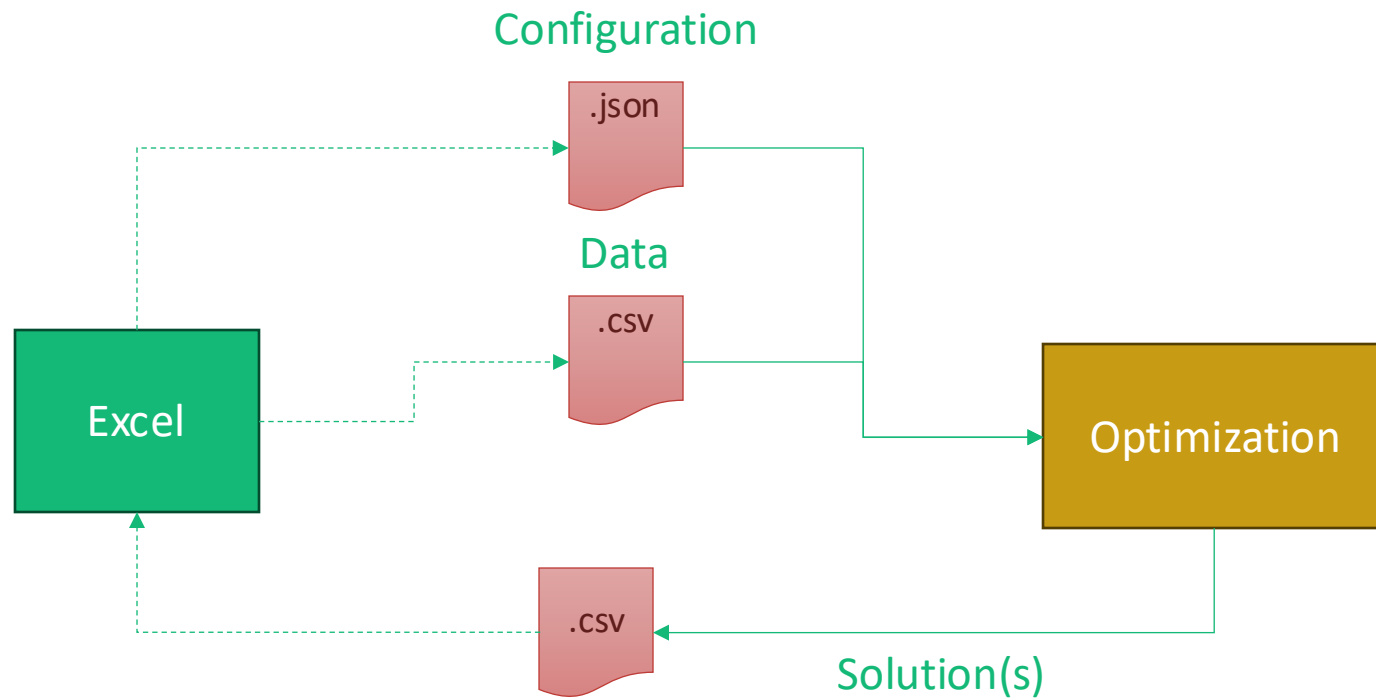
# Data structures

```
function variables_node(mv, i::Market, t)
    for p in products(i)
        insertvar!(mv.sale, i, p, t)
        if !isnothing(price(i, p, t))
            insertvar!(mv.sale_income, i, p, t)
        end
        if allow_slack_min(i, p)
            insertvar!(mv.slack_demand, i, p, t)
            insertvar!(mv.penalty_cost, i, p, t)
        end
    end
end
```

- Each node in the network is responsible for creating its own variables, constraints and objective contributions
- Dispatching on node types

```
function massbalance(m, mv, i::Market, t, prev)
    for p in products(i)
        ctr = @constraint(m, mv.sale[i, p, t] == sum(mv.flow[:, i, p, t]))
        set_name(ctr, "mass_bal_market($i,$p,$t)")
    end
end
```

# Industrial deployment



- Julia model distributed as a self-contained executable using PackageCompiler
- 382 MB zipped file



SINTEF



## MassePlan (v1.0) - resultat

Løsningskatalog: C:\\_project\CircMass\Test\Prosjekt\results

Velg...

### Mulige løsninger

Løsning	Pris total [kr]	Pris transport [kr]	Pris produkt [kr]	Avgift innkjørt [kr]	EPD [kgCO2e]
1	1 630 260	357 000	1 149 510	123 750	51 743
2	1 650 775	377 515	1 149 510	123 750	52 743
3	1 713 982	440 722	1 149 510	123 750	55 712
4	1 872 027	443 767	1 304 510	123 750	57 163

Løsning:

	Masser inn	Masser ut	Total
Volum [tonn]	9 670	2 750	12 420
EPD A1-A3 [kgCO2e]	23 788		23 788
EPD A4 [kgCO2e]	22 237	5 718	27 955
Pris produkt [kr]	1 149 510		1 149 510
Pris transport [kr]	300 372	56 628	357 000
Avgift innkjørt [kr]		123 750	
Pris total [kr]	1 449 882	180 378	1 630 260

### Masser inn

Produkt	Fra	Til	Volum [tonn]	EPD A1-A3 [kgCO2e]	EPD A4 [kgCO2e]	Pris produkt [kr]	Pris transport [kr]
22125	Lørenskog	Prosjekt	7 750	19 065	18 817	999 750	263 586
32	Bjønndalen	Prosjekt	1 920	4 723	3 421	149 760	36 786





SINTEF

# Experiences and learnings

- The use of physical units can be a bit cumbersome, but can catch errors in modeling at an early stage
- Custom structures for holding model variables can be beneficial
- Multi objective for "free" 😊
- Deployment at customer can be a considerable challenge (but not necessarily related to Julia/JuMP)



SINTEF

# Example 2: Locating hydrogen infrastructure in the Lofoten Islands



Photo by Holly Rowland/CC-BY 2.0

Technology for a better society



SINTEF

# Approach using JuMP

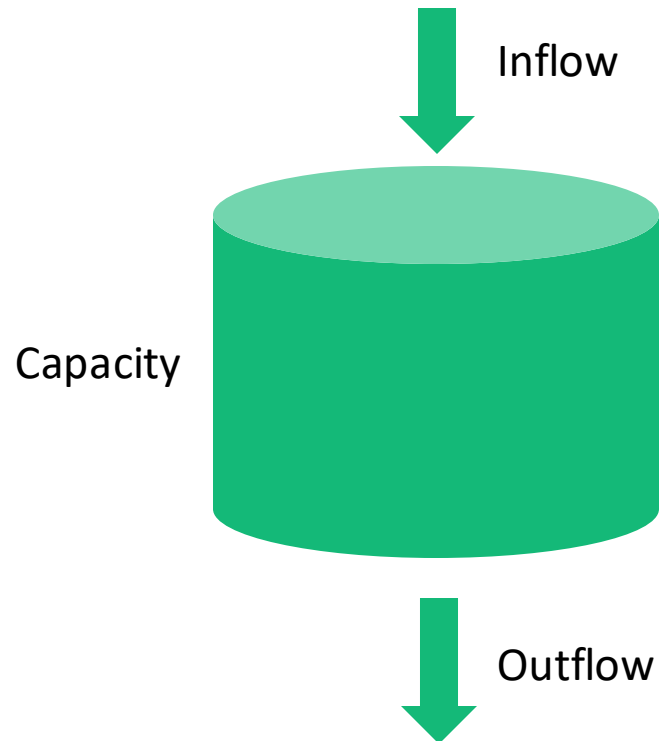
- **TimeStruct** for flexible time structures
- **Storages** for general modeling of storage inventory (in combination with TimeStruct)
- **PiecewiseAffineApprox** to add convex piecewise linear approximations of a set of points to optimization models
- **EnergyModelsInvestement** for general modeling of investment decisions



# Submodels as separate packages with well-defined APIs

- Inspired by the approach of ModelingToolkit/Modelica and Plasmio
- Create a standard library of components that can be combined through connectors
- Components can be added to an optimization model producing the required submodel

# Storages



- Storage as a high-level object that can be directly added to a JuMP model
- Builds upon TimeStruct
- Exposes a small interface:

```
export init_storage, add_storage!  
export set_inflow, set_outflow, set_capacity
```

- Tracking of storage levels is internal to the module



SINTEF

# Usage example

```
periods = SimpleTimes(3, 1)
model = JuMP.Model(HiGHS.Optimizer)
@variable(model, cap >= 0)
@objective(model, Min, cap)

storage = Storages.init_storage(model, "Lager", Storages.Cyclic(), periods)

set_capacity(storage, periods, cap)
for t in periods
    set_outflow(storage, t, 1)
end
set_inflow(storage, first(periods), 3)

Storages.add_storage!(model, storage, StorageData(init_stock_level = FixedProfile(2)))

optimize!(model)
```



SINTEF

# Generated model

```
Min cap
Subject to
  _init_stock_level_Lager[sc1] == 2
  _stock_level_Lager[t1] - _init_stock_level_Lager[sc1] == 2
  -_stock_level_Lager[t1] + _stock_level_Lager[t2] == -1
  -_stock_level_Lager[t2] + _stock_level_Lager[t3] == -1
  -_stock_level_Lager[t3] + _init_stock_level_Lager[sc1] == 0
  -cap + _init_stock_level_Lager[sc1] <= 0
  -cap + _stock_level_Lager[t1] <= 0
  -cap + _stock_level_Lager[t2] <= 0
  -cap + _stock_level_Lager[t3] <= 0
  cap >= 0
  _stock_level_Lager[t1] >= 0
  _stock_level_Lager[t2] >= 0
  _stock_level_Lager[t3] >= 0
  _init_stock_level_Lager[sc1] >= 0
```



SINTEF

# Experiences and learnings

- Separate modules for different components and aspects allow for easy reuse
- Hiding internal model implementation reduces complexity
- Composability in Julia is excellent
- Still experimenting, more tutorials and best-practice are welcome





SINTEF

# Summary and wishlist

- Julia and JuMP has been well received among all involved colleagues
- JuMP documentation and tutorials are very good
- JuMP has been robust and stable
- Much to gain from working with an open-source approach (code and knowledge sharing)
- Wishlist
  - Use of physical units as an integral part of JuMP
  - Model debugger (IIS detection, relaxation, activating/deactivating constraints)
  - Efficient handling of large scale and sparse models and associated solution information



SINTEF

Technology for a better society