

Optimization for Electric Engineering

Application to Energy Management of Micro-Grids and Electric Transportation

06-09 October 2019

Vincent Reinbold

Preliminaries

Presentation

Vincent Reinbold
Associate Professor

*Laboratoire Génie électrique et électronique de Paris (GeePs) (UMR 8507
CNRS / CentraleSupelec - Universites UPMC et UPSud) 11 rue Joliot-Curie -
Plateau de Moulon 91192 Gif sur Yvette Cedex - France*

vincent.reinbold@geeps.centralesupelec.fr

Academic Background

- Master's degree in Electrical Engineering



2008 - 2011

📍 Grenoble Institute of Technology - France

Academic Background

- **PhD**



Oct. 2011 - Oct. 2014

📍 Environment and Transportation Laboratory , IFSTTAR Lyon, FR
Electrical Engineering Laboratory (G2eLAB), Grenoble, France

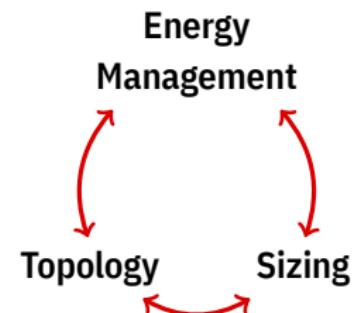
Academic Background

- PhD

Oct. 2011 - Oct. 2014

📍 Environment and Transportation Laboratory , IFSTTAR Lyon, FR
Electrical Engineering Laboratory (G2eLAB), Grenoble, France

"Sizing methodology of an electrical machine for hybrid electrical vehicles - joint optimization of components sizing and energy management"



Academic Background

- Postdoctoral research



Nov. 2014 - August 2015

📍 Electrical Engineering Laboratory (G2eLAB), Grenoble, France

"Interoperability and co-simulation for eco-buildings conception" (Project ANR INTENSE)

Academic Background

- Postdoctoral research

 August 2015 - Feb. 2016

 Electrical systems planning research laboratory - Florianópolis, Brazil

Granted from the European *ELECON project*

"Mixed integer linear programming for the optimization of the energy management and sizing of micro-grids"

Academic Background

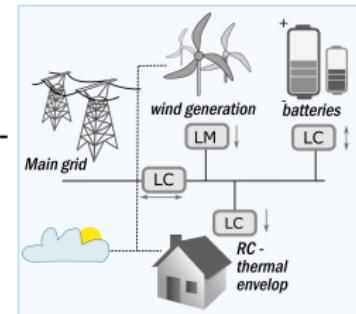
- Postdoctoral research

📅 August 2015 - Feb. 2016

📍 Electrical systems planning research laboratory - Florianópolis, Brazil

Granted from the European *ELECON project*

- Linear Programming and micro-grids
- Use of the building thermal structure for Active Demand Response
- Comparison Linear vs. Non-Linear approaches



Academic Background

- **Postdoctoral research**

KU Leuven, Building Physics Section - Leuven
Energy Ville - Genk



March 2016 - May 2018

- Modeling and **co-simulation** of District Energy Systems,
- **Cooperation** within the Internationale Energy Agency.

Form of the lecture

- Lecture in English (and Portuguese) $\approx 20h$ ($4h \times 5$)
- Interactions, Questions, Feedback in ENG, PT or FR
- Multiple choice and open questions on
<https://www.wooclap.com/UFPBCEAR>

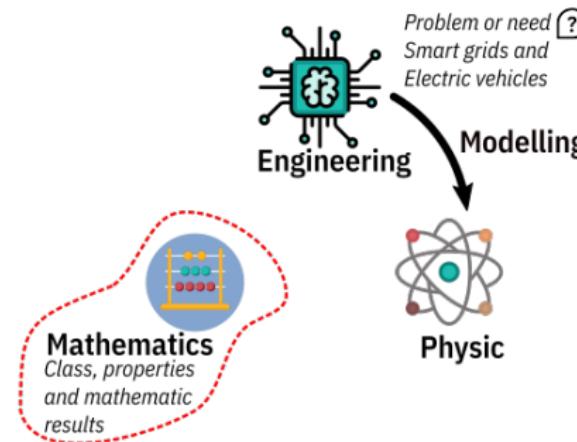
Some prerequisites

- Elementary Mathematical Analysis (derivative, gradient, etc.) (Part 1)
- Elementary Informatics (Part 1)
- Modelling Languages (All parts)
(One of the following : Matlab, Python, Scilab, AMPL)
- Basic Electrical Engineering (Part 2)
- Basic Mechanics (Part 3)

Objectives of the lecture

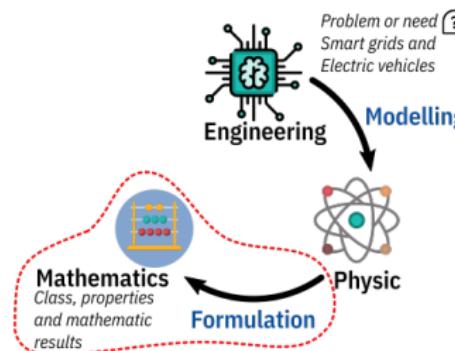
- **Identify optimization classes**

- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)
- Be able to apply those methods to Electric Engineering (Application to micro-grids and EV)



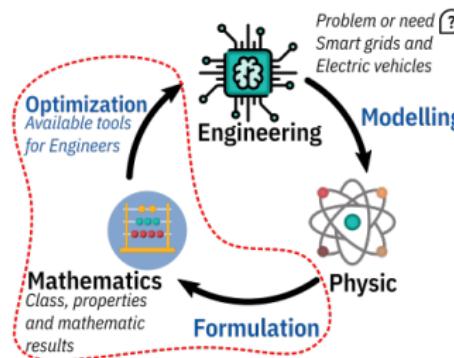
Objectives of the lecture

- Identify optimization classes
- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)
- Be able to apply those methods to Electric Engineering (Application to micro-grids and EV)



Objectives of the lecture

- Identify optimization classes
- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)
- Be able to apply those methods to Electric Engineering (Application to micro-grids and EV)



Objectives of the lecture

- Identify optimization classes
- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)
- Be able to apply those methods to Electric Engineering (Application to micro-grids and EV)

Content

PART 1: Mathematical Optimization

PART 2: Python & Pyomo Tutorial

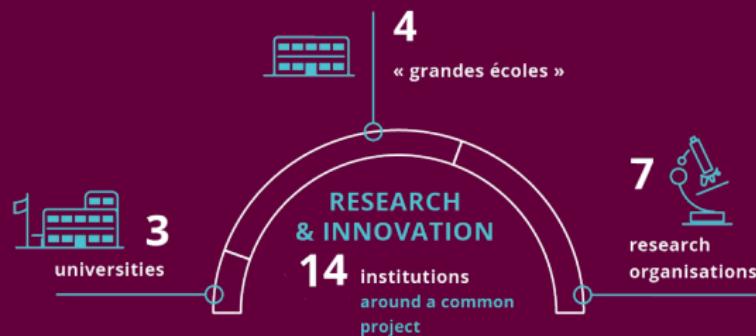
PART 3: Micro-Grids

Paris-Saclay University



A Leading University at the Heart
of a Large-Scale Ecosystem of
Innovation

A large, stylized lowercase letter "e" is positioned on the right side of the slide. Above it, a lowercase letter "i" is partially visible, also featuring decorative dots.



14 founding members

for

1 research university

A world-class cluster

Université Paris-Saclay

€ 1.5 bln

Plan Campus
(academic buildings)

€ 2.5 bln

Infrastructure

EPAPS Paris-Saclay cluster authority

€ 1 bln

+ € 10 bln

Private investment

New Grand Paris Express Metro Line

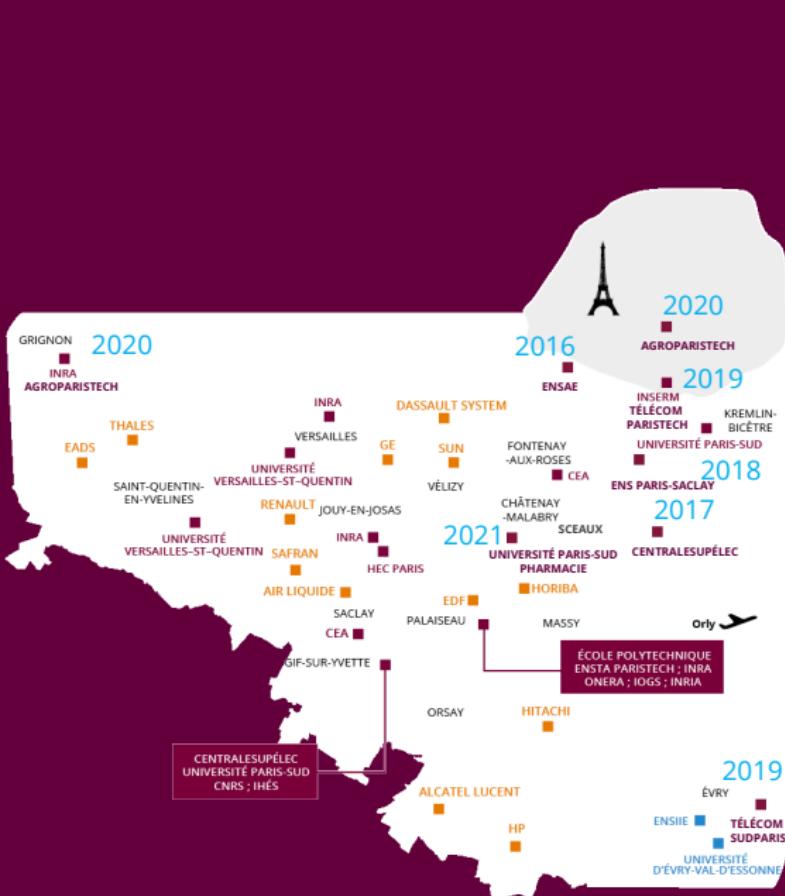
Société du Grand Paris

€ 1.5 bln

+ € 4.5 bln

other Greater Paris
lines

**A key stakeholder
for the
Greater Paris
Project**



A large-scale ecosystem
in constant transformation

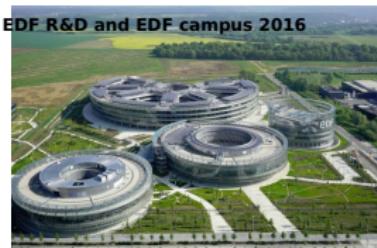
An aerial photograph of the EDF Research Center. The complex features four large, interconnected circular buildings with glass facades and metallic roofs, arranged in a cross-like pattern. The buildings are set against a backdrop of green fields, trees, and a road network. The sky is clear and blue.

2016 EDF Research Center

New buildings for higher education institutions (2016-2020)



New research facilities and centers (2016-2019)



Enhancement of Campus life



Learning Center
2018



Wellness and sports Center 2018



Live-Work-Play 2016



Metro Line 18
Beginning of works 2017



Dedicated bus lines



Learning Centre

1,200 seats – 10,000 sq.m – 2 km of shelf space
Collaborative workrooms, auditorium
Electronic access to millions of publications





2018
**delivery of 2
outdoor sports
fields**

2019
**the biggest
sports facilities
project in
France**

A green
Campus
close to
Paris with
outstanding
natural
reserves



Economic Leaders



13% of French R&D

**167 ERC projects
(FP7&H2020)**

148 ERC grantees
1st in France
3rd in Europe

**10 Fields
Medallists**

ARWU 2018 Subject Ranking



ARWU Academic Ranking of World Universities
Projection 2019

15th - 20th



65,000
students
 9,000
in master degree

 2017-2018 Academic year

- 90,000 applications
- 9,000 master students in the mutualized programs



4,600
PhD students



8 schools



45 master programs
46 master tracks
0

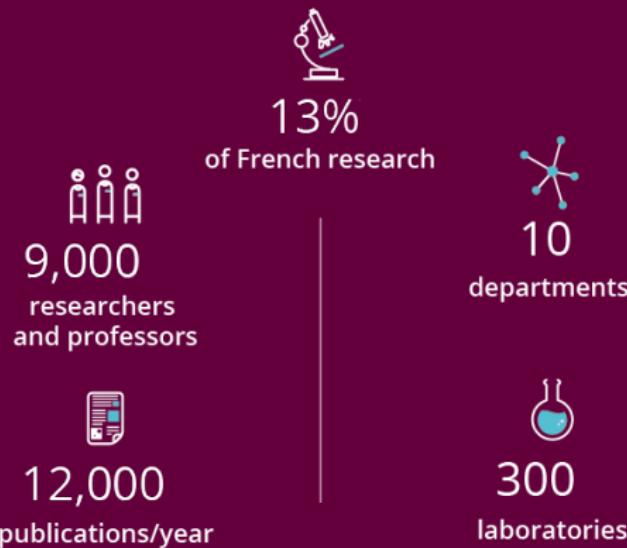


20 doctoral schools

1,300 PhD/year
(support from LABEX, LIDEX, IDI...)

International Master's Students: 38 %
International PhD students: 42%

Innovative Cutting-Edge Academic Programmes

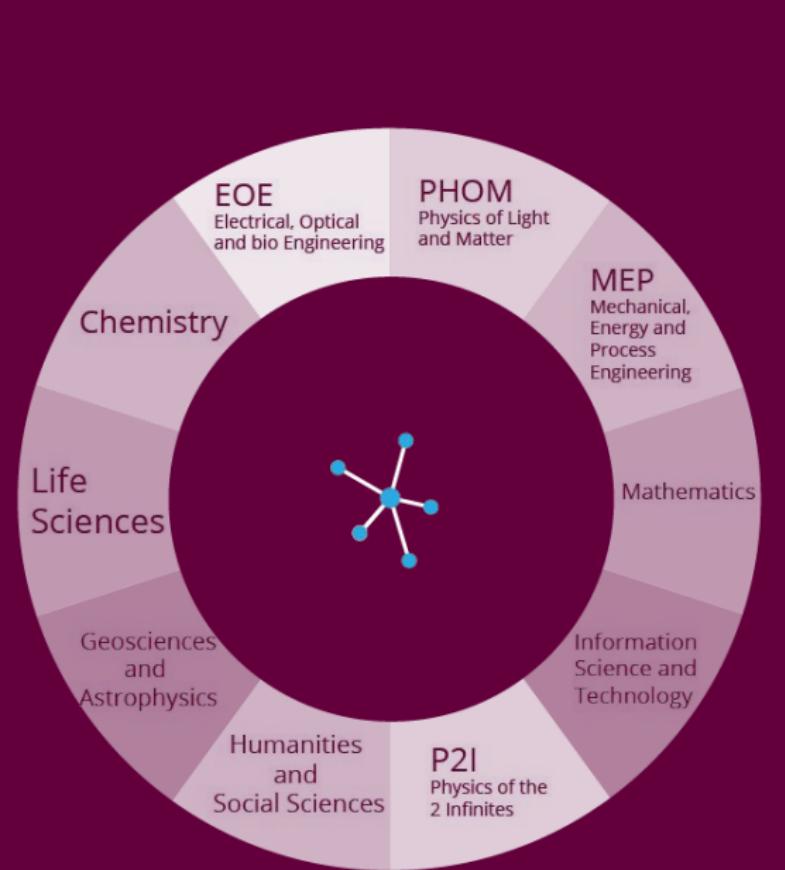


Major IDEX actions: new institutes,
interdisciplinary projects,
chairs,



A common identifier
for all scientific publications
laboratory, institution, Université Paris-Saclay, address

Intensive Research



10 Research departments

s

Interdisciplina
ry institutes
and centres

- **Systems Engineering**- LASIPS
- **Drugs and Therapeutic Innovation** – LERMIT
- **Hadamard Mathematics Laboratory** - LMH
- **Nanosciences** – NanoSaclay
- **Physics: Atoms, Light, Matter** – PALM
- **Physics of the Infinites and Origins** – P2IO
- **Saclay Plant Sciences** – SPS
- **Chemistry of Multifunctional Molecular Architecture and Materials** – CHARMMMAT
- **Biodiversity, Agroecosystems, Society, Climate** – BASC
- **Economics and Decision Sciences** – ECODEC
- **Distributed Data, Programs and Architecture** - DIGICOSME

11 strategic interface programs **(Labex)**

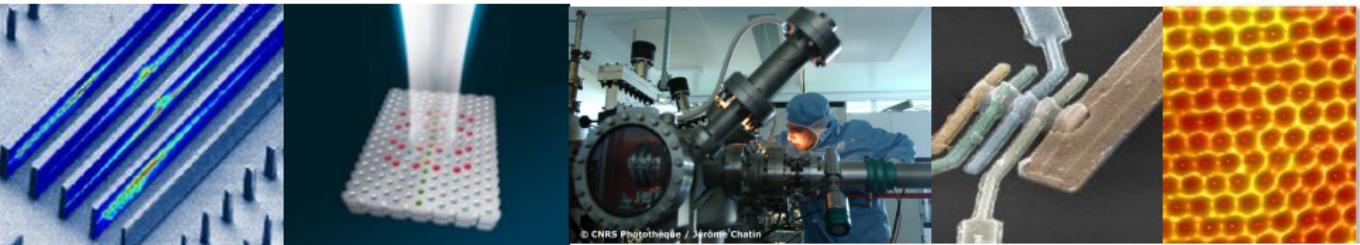


C2N
Center for
Nanoscience
and
Nanotechnology



www.c2n.universite-paris-saclay.fr/en

Nanoscience and Nanotech: C2N lab & research group



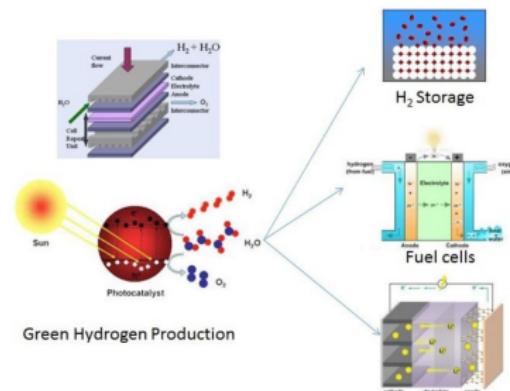
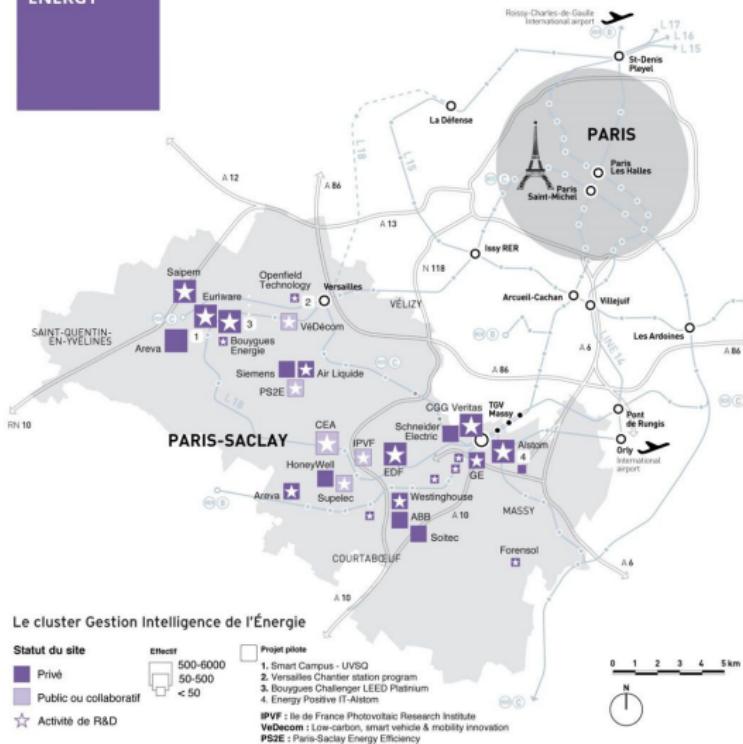
- **410 people**
- **18,000 sq. m**
- **2,800sq. m clean room**
- Large nanotech facility to:
 - develop new advanced tools for nanotechnologies
 - launch these tools on the market in partnership with leading manufacturers in the field



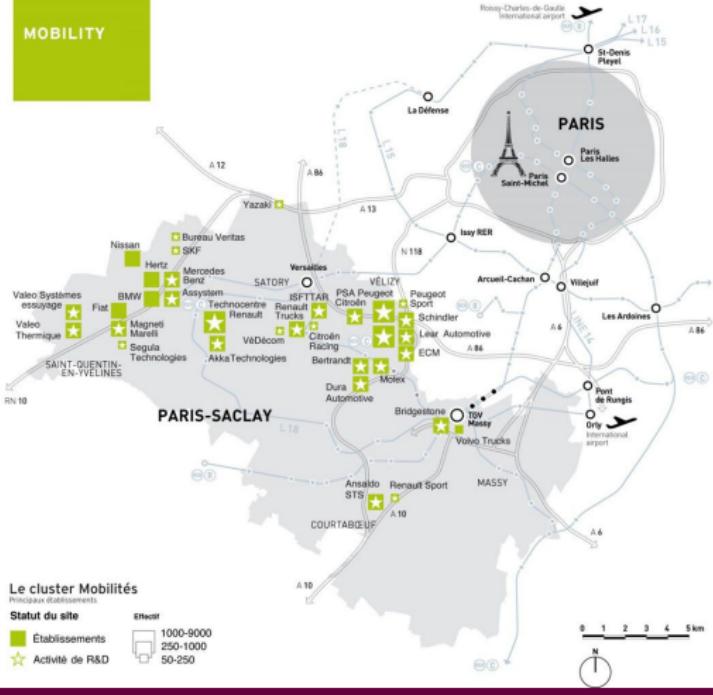
2 cutting edge labs:
IEF (Fundamental
Electronics Institute)
and LPN (Photonics
and Nanostructures
Lab)

ENERGY

ENERGY



MOBILITY



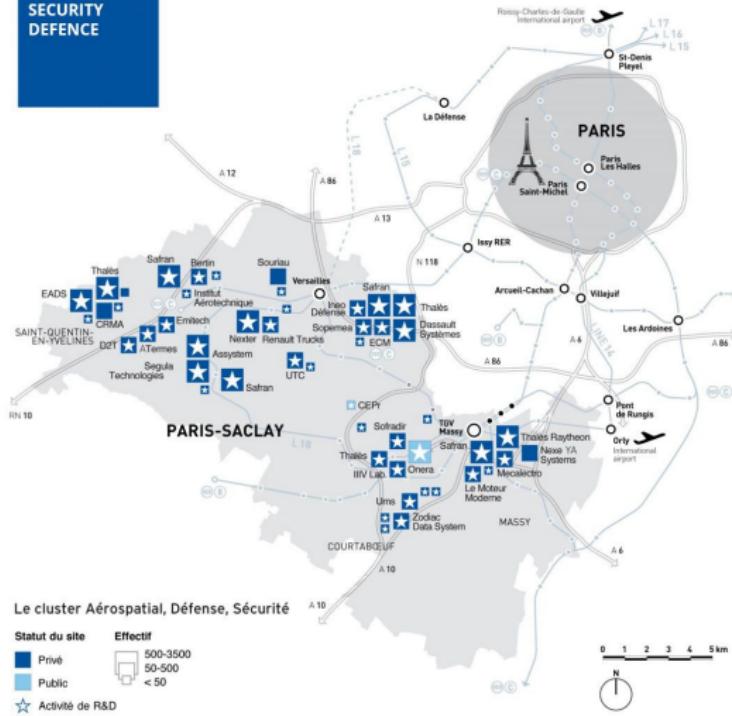
MOBILITY

Development of Autonomous Vehicles on the Paris Saclay territory



université
PARIS-SACLAY

AEROSPACE
SECURITY
DEFENCE



The Aerospace Network of UPSaclay



AIRBUS

 **SAFRAN**
AEROSPACE · DEFENCE · SECURITY

université
PARIS-SACLAY

- Over 50 UPSaclay active international partnerships (and more than 400 through UPSaclay's members)
 - Master's tracks taught in English: 15 %
 - International Master's Students: 38 %
 - International PhD students: 42%
 - 45 CNRS International Associated Labs (more than 25 % of all French LIAs)



A university open to the world

International agreements signed by Université Paris-Saclay (2016-2019)



➤ For Master's students

International Master's Scholarships

120 Incoming full master's scholarships /

150 Outgoing research internships (2-6 months)

➤ For PhD students

□ Over 350 full PhD grants / year

□ **ADI programme** - Additional UPSaclay grants for PhD cotutelles with strategic partners

27 in 2019

□ **CIFRE** = co-funded by businesses

□ **Chateaubriand Fellowships**

For PhD students from US universities

4-9 months research at a UPSaclay lab

Co-funded with the French Embassy in the US

➤ For researchers

□ **Jean d'Alembert Fellowships**

Programme for junior and senior scientists

6-12 month stays: 10 laureates / call

Any field – from any country

51 laureates since 2014

➤ Also - Funding for **international summer schools, scientific workshops, and short stays at UPSaclay and abroad** for **Master's and PhD students**

Fostering international mobility



Master's programmes in English at Université Paris-Saclay



Over 120 Master's programmes taught in English

38 Master's programmes exclusively in English

66 Master's programmes in both languages

Schools

- Basic Sciences
- Biology, Medicine and Pharmacy
- Biodiversity, Agriculture and Food, Society, Environment
- Engineering, Information Science and Technology
- Law and Political Science
- Social Sciences
- Sports Sciences and Human Movement



Master's programmes in English at Université Paris-Saclay

TRACKS



BASIC SCIENCES

- Chemistry (4)
- Earth and Planetary Sciences, Environment (5)
- Mathematics & Applications (6)
- Physics (10)

BIOLOGY, MEDICINE & PHARMACY

- Life Sciences and Health (18)
- Pharmaceutical Science (3)
- Public Health (3)

BIODIVERSITY, AGRICULTURE & FOOD, SOCIETY, ENVIRONMENT

- Agrosciences, Environment, Territory, Landscape, Forest (1)
- Environmental, Energy and Transport Economics (3)
- Nutrition and Food Science (4)

LAW AND POLITICAL SCIENCE

- Business Law (3)
- International and European Law (2)

ENGINEERING, INFORMATION SCIENCE & TECHNOLOGY

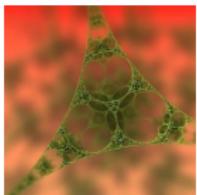
- Bioinformatics (1)
- Computer Science (11)
- Electrical and Control Engineering (8)
- Energy (3)
- Materials Science and Engineering (2)
- Mechanics (5)
- Nuclear Energy (7)

SOCIAL SCIENCES

- Economics (2)
- Finance (1)
- Innovation, Enterprise and Society (4)
- Social Sciences (2)
- Strategic Management (3)

2019

Short Programmes and Summer Schools in English at Université Paris-Saclay



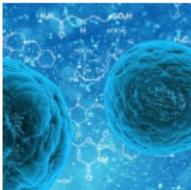
A variety of thematic fields and audiences

For

Bachelor, Master's & PhD students
Post-docs, researchers
Professionals...



Aeronautics
Agriculture
Arctic Studies
Artificial Intelligence
Chemistry
Climate Sciences
Computer Science
Condensed Matter
Physics
Condensed State Physics
Genomics
High Energy Physics



Industry
Information Technology
Integrative Biology &
Ecology
Life Sciences
Mathematics
Molecular Scale Physics
Nanotechnology
Quantum Technologies
Science Management
Specialized Plant
Metabolites

Welcoming International Students and Researchers at Université Paris-Saclay



e-International Welcome Office

Dedicated Website and Web app to help international students and researchers with administrative procedures

- ❑ Individualized roadmap based on each person's specific situation (student, researcher, nationality, country of origin, duration of stay...)
- ❑ Visa application, health insurance, bank account, residence permit, academic fees, accommodation, financial support...
- ❑ Information about cultural life in Paris and at UPSaclay



GATE platform (*Welcome Desk for International Talents*)

One **centralized office** for all administrative procedures to guide international talents living and working in the Paris-Saclay area upon their arrival and stay in France

Université Paris-Saclay Buddy Program - Fostering induction of international students

- For all new international students
- Matching new international students with local students
- Support for admin procedures
- Social and cultural activities



[www.universite-paris-saclay.fr/
en](http://www.universite-paris-saclay.fr/en)
<https://www.universite-paris-saclay.fr/en/e-international-welcome-office>



@UnivParisSaclay



UParisSaclay



Université Paris-Saclay



Universite_Paris_Saclay



Thank you !

Part 1 - Introduction to Mathematical Optimization for Electrical Engineering

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

1 - Introduction

■ General definition

- Importance of Convexity
- Continuous vs Discrete
- Finite vs Infinite Dimensional Optimization

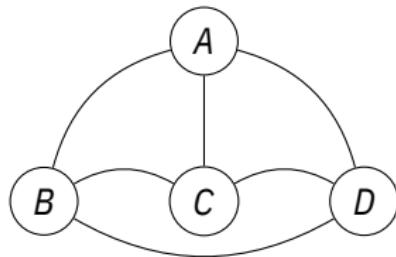
2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

General definition

Introductory example



Drilling an electronic board : Optimal path problem

n	T_{search} (3 GHz ^a)	T_{search} (Planck ^b)
10	10 ms	
15	1 h	
19	1 an	
27	$8 \times \text{Universe age}$	
35	-	5 ms
40	-	12 years

^a1 op. every $\frac{1}{3} \times 10^{-9}$ s

^b $5,391 \times 10^{-44}$ s

Table: Finding best solution by compute every possibilities : "brute force"

Definition

Goal :

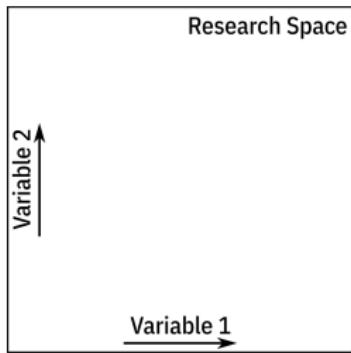
Find $\bar{x} \in \mathbb{X}$ such that
$$\begin{cases} f(\bar{x}) = \min f(x) \\ x \in \mathbb{X} \\ g(x) \leq 0 \\ h(x) = 0 \end{cases}$$

where f, g, h are given functions, and \mathbb{X} is a known set.

Vocabulary

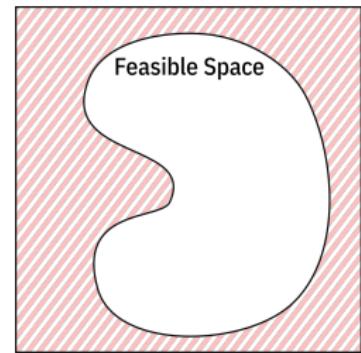
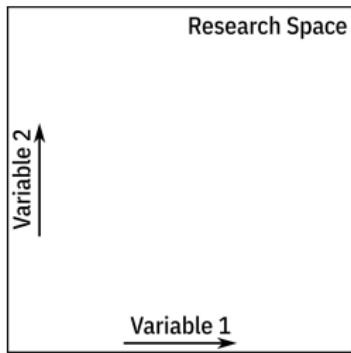
Objective function, Inequality and Equality Constraints, Variable, Optimal Solution, Feasible Set

Geometric representation (2D)



Geometric representation (2D) of the feasible set, the objective function contours and the optimal solution

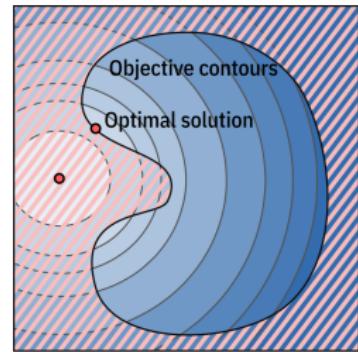
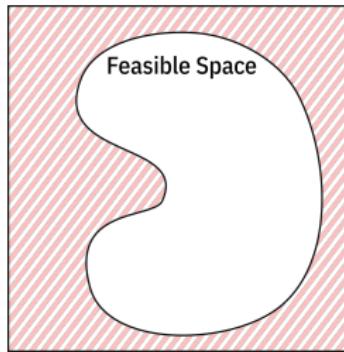
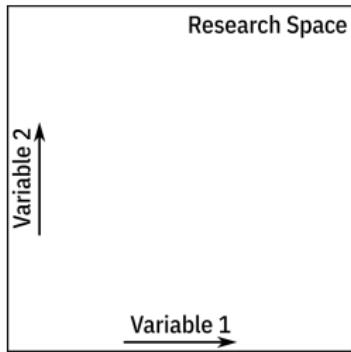
Geometric representation (2D)



Geometric representation (2D) of the feasible set, the objective function contours and the optimal solution

General definition

Geometric representation (2D)



Geometric representation (2D) of the feasible set, the objective function contours and the optimal solution

General definition

What is the use of optimization ?

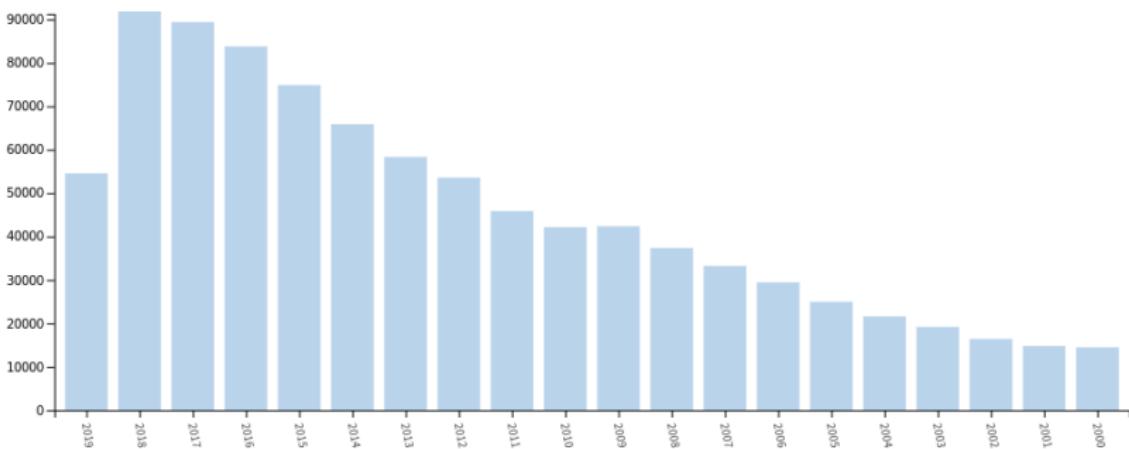


Treemap of records for topic : Optimization and publication years :

2010-2019. source: WebofScience

General definition

What is the use of optimization ?



Bar Graph of Engineering Electrical Electronics records for topic : Optimization and publication years : 2000-2019. source: WebofScience

What optimization problems are most commonly encountered ?



Q. 1

- Transportation, logistics, distribution and stocks
- **Resources allocation** (equipment, energy, raw material)
- Task scheduling (industry)
- Optimal control (motor, chemistry)
- Wire routing
- **Curve fitting**, Neural Networks and Deep Learning
- Production and distribution of Electricity

What optimization problems are most commonly encountered ?



Q. 1

- Transportation, logistics, distribution and stocks
- **Resources allocation** (equipment, energy, raw material)
- Task scheduling (industry)
- Optimal control (motor, chemistry)
- Wire routing
- **Curve fitting**, Neural Networks and Deep Learning
- **Production and distribution of Electricity**

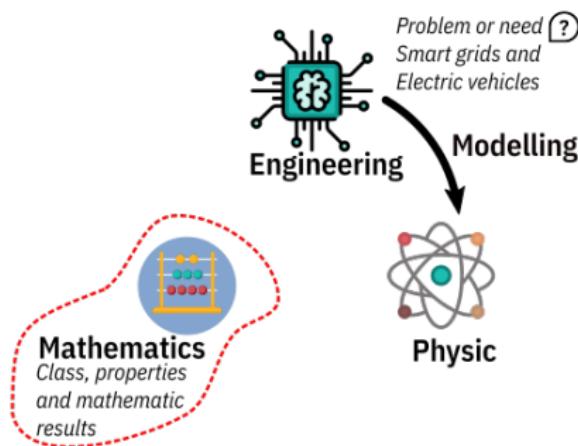
What is the use of optimization ?

- Improve a solution (objective function or constraints satisfaction)
- Give alternative solutions
- Help decision taking
- Give sensibility and robustness feedbacks
- Automatized repetitive tasks

General definition

Objectives of this part

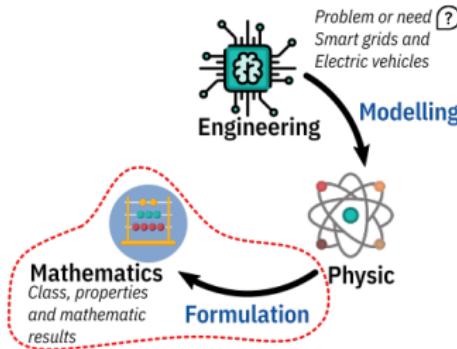
- Identify optimization classes
- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)



General definition

Objectives of this part

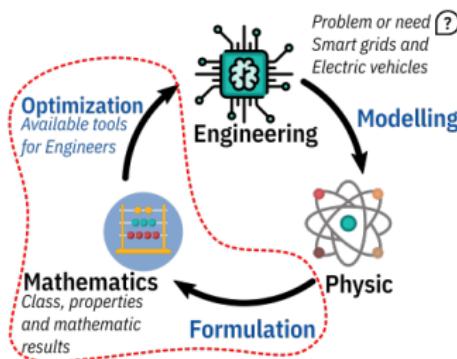
- Identify optimization classes
- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)



General definition

Objectives of this part

- Identify optimization classes
- Formulate a problem from a real engineering problem
- Discover available tools for engineers (you)



1 - Introduction

- General definition
- **Importance of Convexity**
- Continuous vs Discrete
- Finite vs Infinite Dimensional Optimization

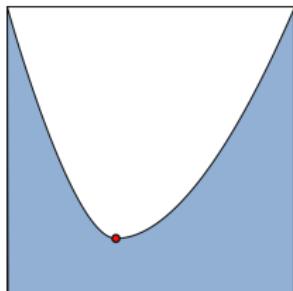
2 - Class of problems

3 - Heuristics & Multi-Objective

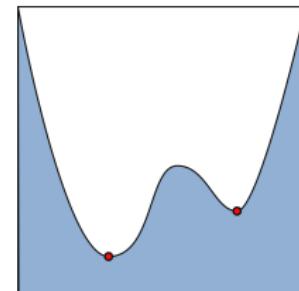
4 - Model and formulation

Importance of Convexity

Importance of Convexity



Convex function : 1 optimal solution



Non-convex function : 1 optimal, 1 sub-optimal solutions

Convex Problems

- Minimum = Global Minimum
- Local information ($\Delta f = 0$) = Global information
- Solvers are efficient

Non-convex Problems

- $\Delta f = 0$: sub-optimal
- Solver may or may not converge to sub-optimal or unfeasible points

How to detect convexity ?

Definition

A function $f : Q \rightarrow \mathbb{R}$ defined on a nonempty subset Q of \mathbb{R}^n and taking real values is called convex, if :

- the domain Q of the function is convex;
- $\forall x, y \in Q, \lambda \in [0, 1]$

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

Operations that conserve convexity

- Weighted sum of convex funct. $\lambda f + \mu g, \lambda, \mu \in \mathbb{R}^{+*}$
- Affine substitution. $f(Ax + b)$
- Upper bounds of convex funct. $\sup f(.)$

NSC for smooth functions

- f is differentiable and f' is monotonically non-decreasing
- f is twice-differentiable and f'' is non-negative

How to detect convexity ?

Definition

A function $f : Q \rightarrow \mathbb{R}$ defined on a nonempty subset Q of \mathbb{R}^n and taking real values is called convex, if :

- the domain Q of the function is convex;
- $\forall x, y \in Q, \lambda \in [0, 1]$

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

Operations that conserve convexity

- Weighted sum of convex funct. $\lambda f + \mu g, \lambda, \mu \in \mathbb{R}^{+*}$
- Affine substitution. $f(Ax + b)$
- Upper bounds of convex funct. $\sup f(.)$

NSC for smooth functions

- f is differentiable and f' is monotonically non-decreasing
- f is twice-differentiable and f'' is non-negative

1 - Introduction

- General definition
- Importance of Convexity
- **Continuous vs Discrete**
- Finite vs Infinite Dimensional Optimization

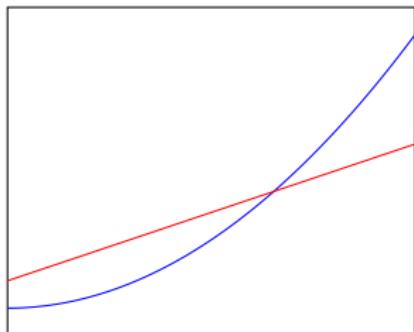
2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

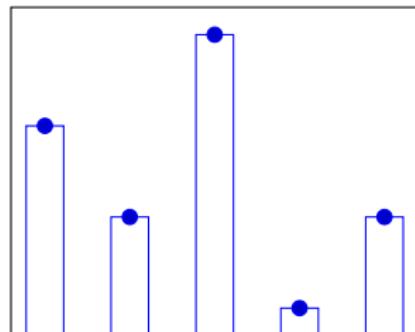
Finite Dimensional Optimisation

Continuous



$$x \in \mathbb{R}^n$$

Discrete



$$x \in \mathbb{N}^n$$

Discontinuities in Electric Engineering



Q. 2

Discontinuities in Electric Engineering



Q. 2

Actuators

Nbr. poles

Architecture

Material Properties

Energy Management

On/Off equipment

Charging/Discharging (batteries)

Hybrid modes

Power Elec.

Reverse/Forward

Nbr. of modules /

IGBT

Rooting

Finite vs Infinite Dimensional Optimization

1 - Introduction

- General definition
- Importance of Convexity
- Continuous vs Discrete
- Finite vs Infinite Dimensional Optimization

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

Infinite Dimensional Optimisation

There is an infinite number of variables.

Examples

- Dynamic system & Trajectory
Command & Control, Shortest path,
Battery SOC, etc.
- Surface & Shape
Actuator, machine shape, wings,
etc.

Useful tips

- Embedded Solver (for ODE)
- Discretization of time - Meshing of space

Infinite Dimensional Optimisation

There is an infinite number of variables.

Examples

- Dynamic system & Trajectory
Command & Control, Shortest path,
Battery SOC, etc.
- Surface & Shape
Actuator, machine shape, wings,
etc.

Usual tips

- Embedded Solver (for ODE)
- Discretization of time Meshing of space

Infinite Dimensional Optimisation

There is an infinite number of variables.

(*Dyn.Syst.*)

$$\begin{cases} \text{find } \min J(x, u) \\ J(x, u) = \int_{t_1}^{t_2} L(x(t), u(t), t) dt \\ \dot{x} = f(x, u) \\ x(t_1) = x_0 \end{cases}$$

Examples

- Dynamic system & Trajectory
Command & Control, Shortest path, Battery SOC, etc.
- Surface & Shape
Actuator, machine shape, wings, etc.

Usual tips

- Embedded Solver (for ODE)
- Discretization of time Meshing of space

Infinite Dimensional Optimisation

There is an infinite number of variables.

(*Dyn.Syst.*)

$$\begin{cases} \text{find } \min J(x, u) \\ J(x, u) = \int_{t_1}^{t_2} L(x(t), u(t), t) dt \\ \dot{x} = f(x, u) \\ x(t_1) = x_0 \end{cases}$$

Examples

- Dynamic system & Trajectory
Command & Control, Shortest path, Battery SOC, etc.
- Surface & Shape
Actuator, machine shape, wings, etc.

Usual tips

- Embedded Solver (for ODE)
- Discretization of time Meshing of space

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

Linear Programming

1 - Introduction

2 - Class of problems

- Linear Programming
- Integer Programming
- Mixed Integer Linear Programming
- Non-Linear Programming

3 - Heuristics & Multi-Objective

4 - Model and formulation

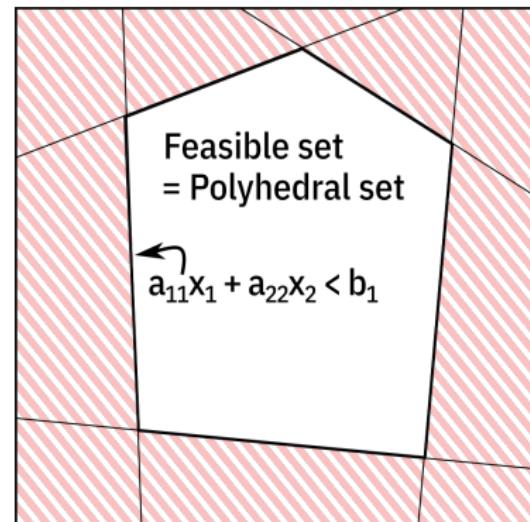
Linear Programming

Linear Programming

Formulation

$$(LP) \left\{ \begin{array}{l} \min c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

where $c^T = (c_1, c_2, \dots) \in \mathbb{R}^n$ is the given cost vector, $x = (x_1, x_2, \dots)^T \in \mathbb{R}^n$ is the variable vector, $A \in \mathbb{R}^{n \times m}$ and $C \in \mathbb{R}^{n \times k}$ are given matrix and $b \in \mathbb{R}^m$, $d \in \mathbb{R}^k$ are given vectors.



2D geometric representation of a LP - Feasible Set

Linear Programming

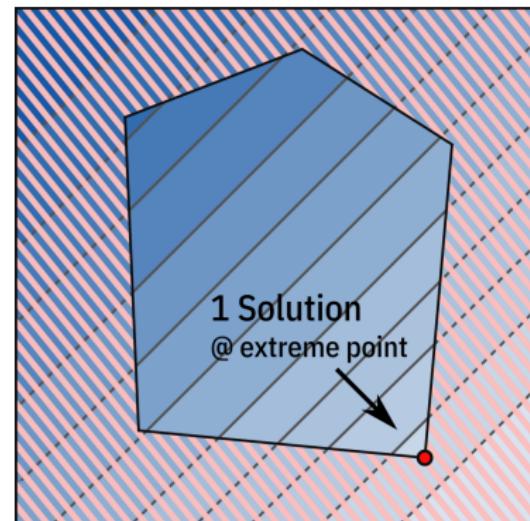
Linear Programming

Formulation

$$(LP) \left\{ \begin{array}{l} \min c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

Notes

Problem has one solution and seems well formulated. Only one solution at extreme point.



*2D geometric representation of a LP -
Solution set is an extreme point*

Linear Programming

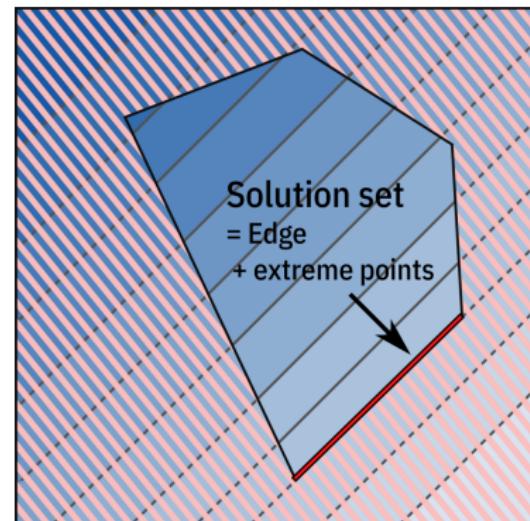
Linear Programming

Formulation

$$(LP) \left\{ \begin{array}{l} \min c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

Notes

Problem has many solution and seems well formulated. Edge and extreme points are solutions.



*2D geometric representation of a LP -
Solution Set is an edge*

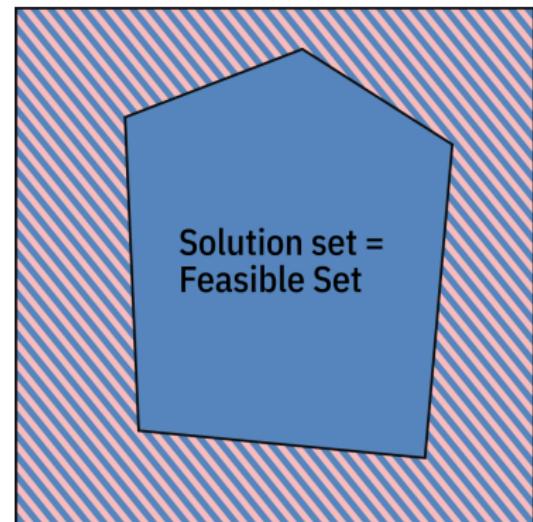
Linear Programming

Formulation

$$(LP) \left\{ \begin{array}{l} \min c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

Notes

The Objective function is constant, all feasible solution is optimal.



*2D geometric representation of a LP-
Solution Set is the Feasible Set*

Linear Programming

Formulation

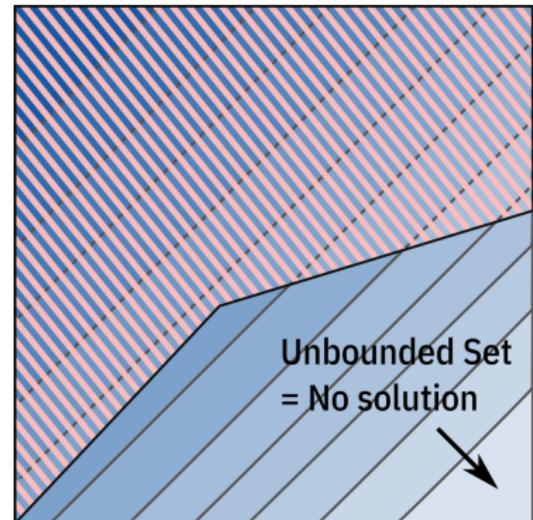
$$(LP) \left\{ \begin{array}{l} \min c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

Notes

The Problem is not constrained enough.

The Feasible set is not bounded.

The Problem might be not well formulated.



2D geometric representation of a LP - Unbounded Feasible Set

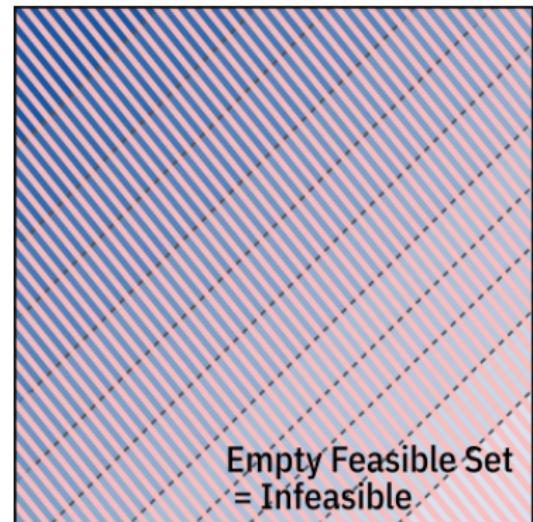
Linear Programming

Formulation

$$(LP) \left\{ \begin{array}{l} \min c^T x \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

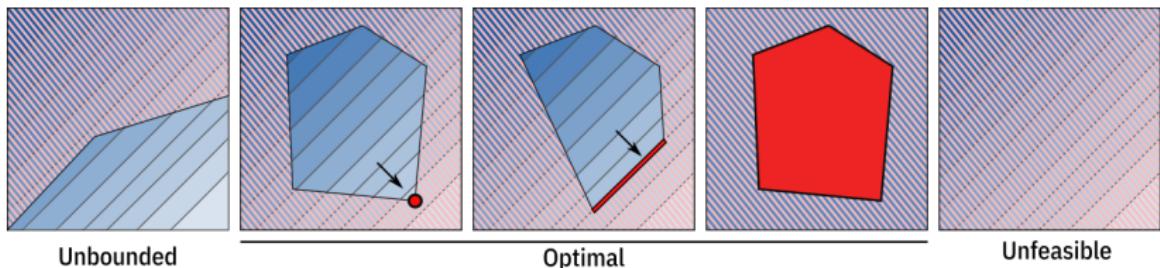
Notes

- The Problem is over constrained.
- The Feasible set is empty.
- The Problem might be not well formulated.



2D geometric representation of a LP - Empty Feasible Set

Linear Programming



2D geometric representation of a LP - All cases

Conclusion

Except in some degenerated cases. If looking for extrema, only consider extrem points !

c.f. Simplex algorithm

Integer Programming

1 - Introduction

2 - Class of problems

- Linear Programming
- Integer Programming
- Mixed Integer Linear Programming
- Non-Linear Programming

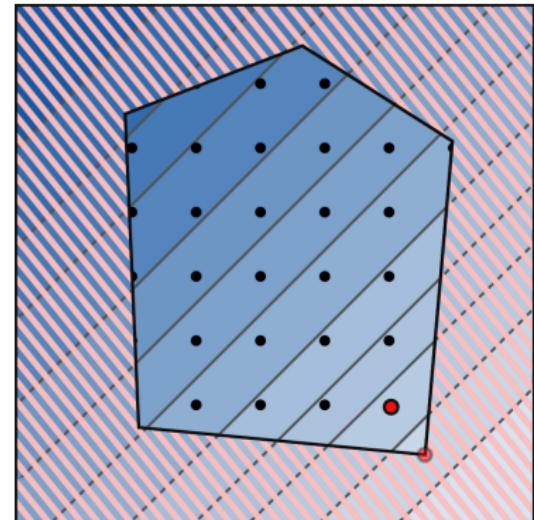
3 - Heuristics & Multi-Objective

4 - Model and formulation

Integer Programming

Formulation

$$(IP) \left\{ \begin{array}{l} \min c^T u \\ u \in \mathbb{Z}^n \\ Du \leq e \\ Fu = g \end{array} \right.$$



*2D geometric representation of a
resolvable IP*

Mixed Integer Linear Programming

1 - Introduction

2 - Class of problems

- Linear Programming
- Integer Programming
- Mixed Integer Linear Programming
- Non-Linear Programming

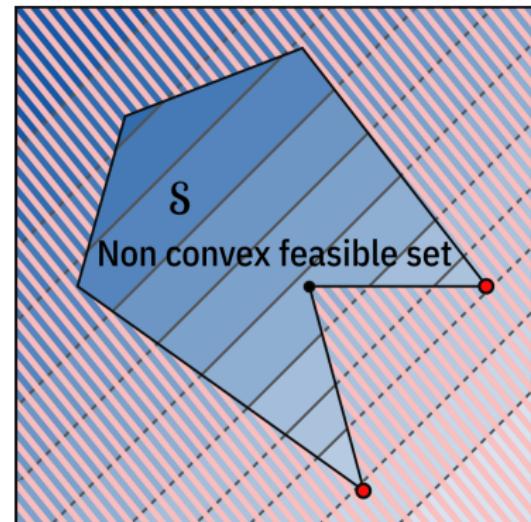
3 - Heuristics & Multi-Objective

4 - Model and formulation

Mixed Integer Linear Programming

Suppose a Feasible set as in Fig.

- Not a Convex Set
- Not a LP



2D representation of a non-convex feasible set.

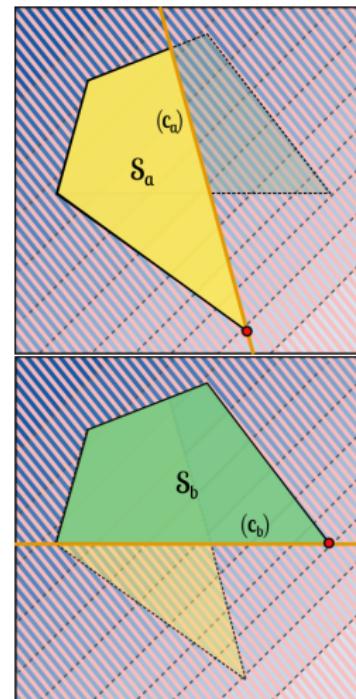
Mixed Integer Linear Programming

But there are many tricks !

- Union of two Convex Set
- Introduction of Integer variable
- Linearisation techniques

General Formulation

$$(MILP) \left\{ \begin{array}{l} \min c^T u \\ x \in \mathbb{R} \times \cdots \times \mathbb{R} \times \mathbb{Z} \times \cdots \times \mathbb{Z} \\ Ax \leq b \\ Cx = d \end{array} \right.$$



1 - Introduction

2 - Class of problems

- Linear Programming
- Integer Programming
- Mixed Integer Linear Programming
- Non-Linear Programming

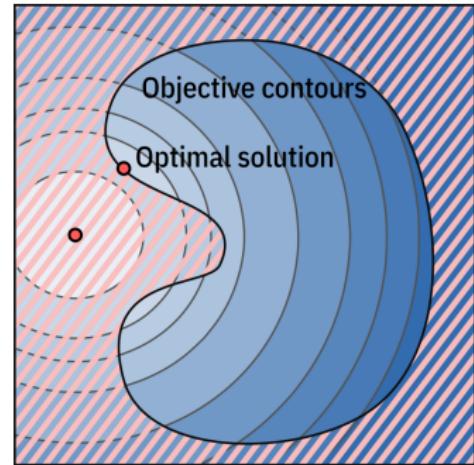
3 - Heuristics & Multi-Objective

4 - Model and formulation

Non-Linear Programming

Large class of problems. Depends on :

- Convexity
- Differentiability (1st & 2nd)
- Linearity of g



Non-Linear Programming

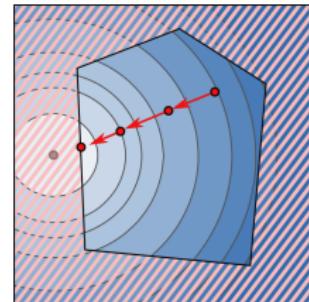
Non-Linear Programming

Solving Non-Linear Programs

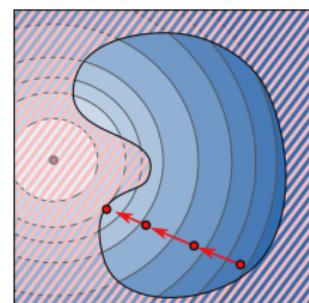
Main Idea

- Iterative algorithm **based on Gradient** (and Hessian)
- Final conditions : **Karush–Kuhn–Tucker (KKT)**

$$\nabla f(x^*) + \sum_i \mu_i \nabla h_i(x^*) + \sum_j \lambda_j \nabla g_j(x^*) = 0 \quad (1)$$



(a) convex



(b) non-convex

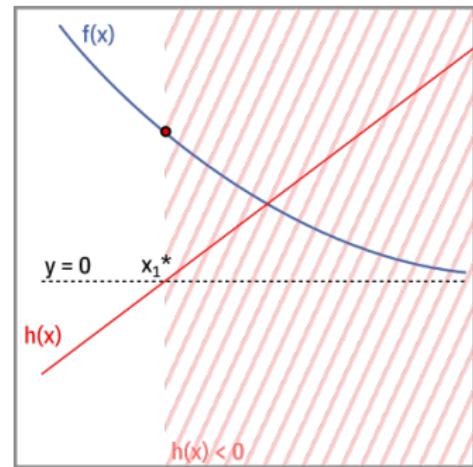
Take home results and properties

$$\nabla f(x^*) + \mu \nabla h(x^*) = 0$$

$$\Rightarrow \mu = -\frac{\nabla f(x^*)}{\nabla h(x^*)}$$

Notes

- $\mu \neq 0$: Active constraint
- μ : Sensibility



*Illustration of KKT condition
(1D)*

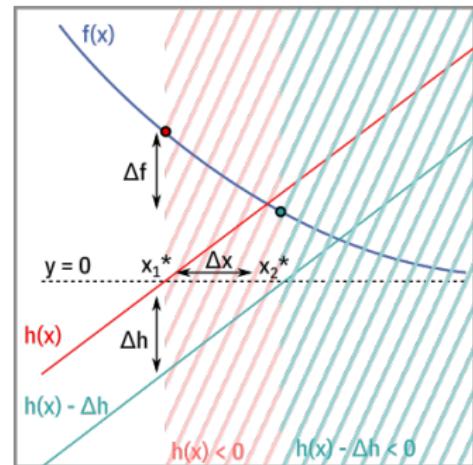
Take home results and properties

$$\nabla f(x^*) + \mu \nabla h(x^*) = 0$$

$$\Rightarrow \mu = -\frac{\nabla f(x^*)}{\nabla h(x^*)}$$

Notes

- $\mu \neq 0$: Active constraint
- μ : Sensibility



*Illustration of KKT condition
(1D)*

Non-Linear Programming

Quadratic Programming

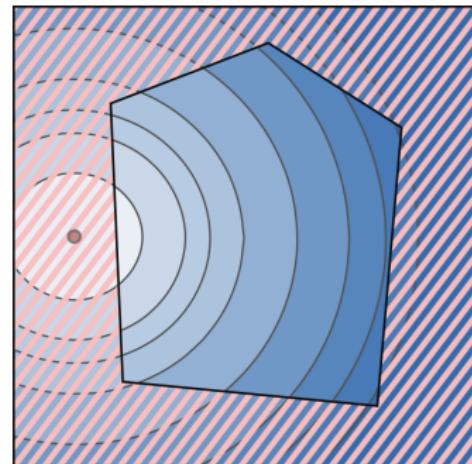
Formulation & Geometric Representation

Formulation

$$(QP) \left\{ \begin{array}{l} \min x^T Qx \\ x \in \mathbb{R}^n \\ Ax \leq b \\ Cx = d \end{array} \right.$$

Notes

Convex iff $\nabla^2 f \geq 0$ i.e. Q is positive semi-definite



Synthesise

Class	Conditions	Convergence
LP	$x \in \mathbb{R}, f, g, h$ linear (1)	Global
IP	$u \in \mathbb{Z}, f, g, h$ linear (2)	Global
MILP	(1) & (2)	Global
NLP	$x \in \mathbb{R}$	No result ^a
QP	Q is SDP ^b	Global
Convex NLP	f, h convex, g linear	Global
Non-convex NLP	$x \in \mathbb{R}, f, g, h$ non-convex (3)	No result ^a
MINLP	(3) + $u \in \mathbb{Z}$	No result ^a

^a May diverge or converge to unfeasible, local or global solution. Depends on the initial solution.

^b Positive Semi-definite



Q. 3

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

Heuristic methods

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

- Heuristic methods
- Multi-Objective Optimization

4 - Model and formulation

Heuristic methods

Heuristic and Meta-heuristic

Heuristics

- Rule based / Self-discovery
- No general properties

Meta-Heuristics

- Computer based
- Iterative
- Intensification and Diversification

Famous Algorithms

- Simulated Annealing
- **Genetic Algorithms**
- Ant Colony Optimization
- Bee Algorithms
- Particle Swarm Optimization

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

- Heuristic methods
- Multi-Objective Optimization

4 - Model and formulation

Multi-Objective

Procurando um bom compromisso

New formulation find $P = \{x' \in S \mid \{x'' \in S \mid x'' \succ x', x'' \neq x'\} = \emptyset\}$

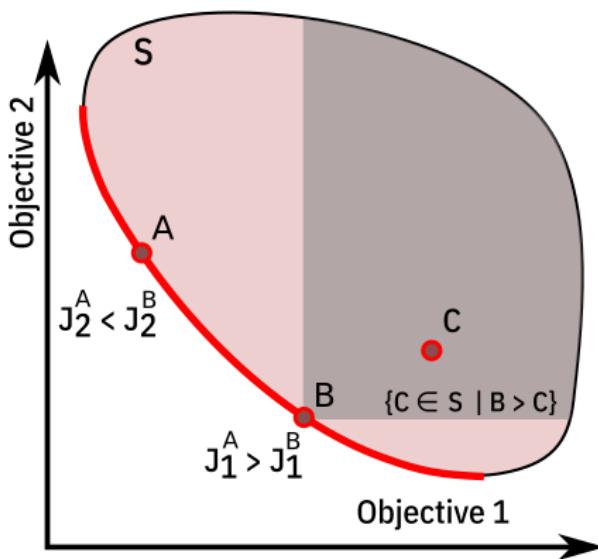
When is it encountered ?

- More than one objective
- Contradictory objectives

Examples

- Economical vs Ecological costs
- Economical costs vs Performances
- Volume or weight vs Efficiency
- Life Cycle Analysis (≈ 10 criteria)

Pareto-optimal Solution Set



Representation of solution in the 2D-objectives space.

NSGA-II

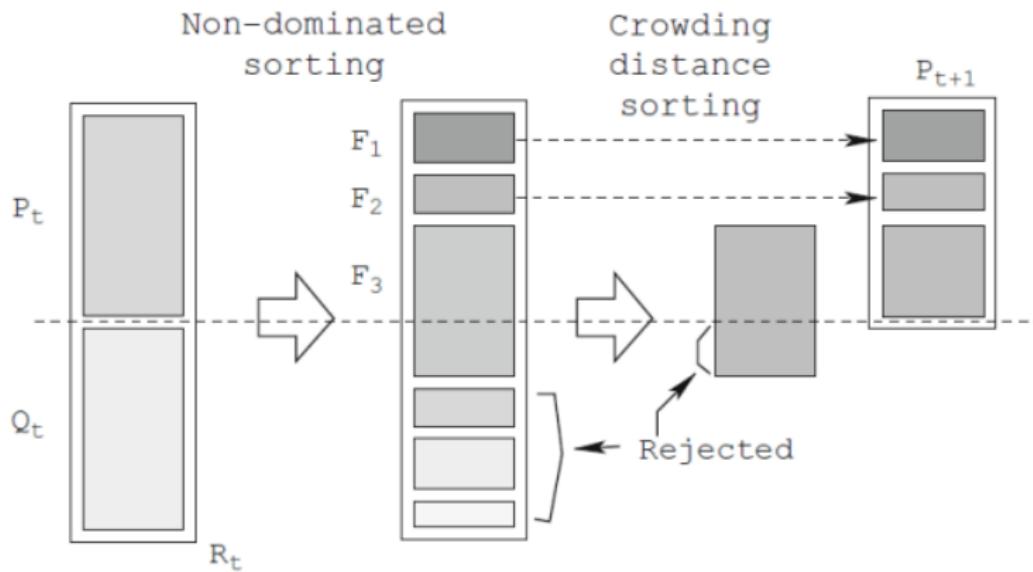
Vocabulary

- Variable : gene
- Solution (set of variable) : Chromosome
- Set of solution : Population

Idea

- Based on Evolution Theory and generations
- Intensification : Crossing between parents
- Diversification : Chromosome Mutation

NSGA-II



Reformulation with 1 objective

Method #1 :

Weighted sum

Algorithm

- Solve $\min_x \alpha f_1(x) + (1 - \alpha)f_2(x)$
with $\alpha \in [0, 1]$
- Change α

Method #2 :

ϵ -constraint

Algorithm

- Solve $\min_x f_1(x)$ s.t. $f_2(x) < \epsilon$
with $\epsilon \in \mathbb{R}$
- Change ϵ

Example

Economic vs. Ecological cost ($85\text{€}/t_{CO_2}$)

Economic vs. comfort

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

1 - Introduction

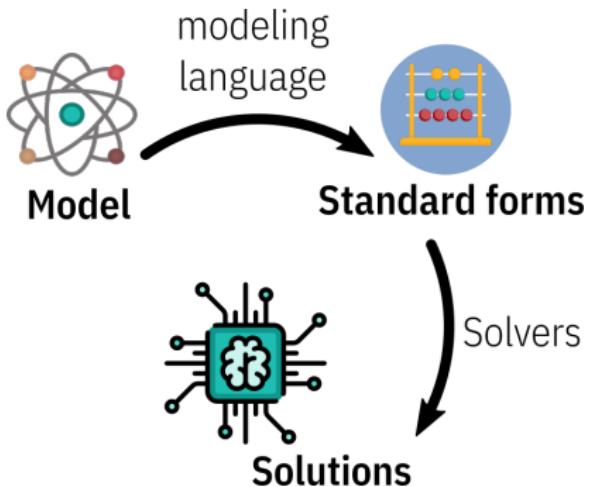
2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

- Modelling and Modelling Languages
- Rules and Tricks to Formulate Optimization Problems
- Available tools

Computer based Modelling



Computer based modelling

```

1 m = AbstractModel()
2 m.N = Set()
3 m.A = Set(within=m.N*m.N)
4
5 m.s = Param(within=m.N)
6 m.t = Param(within=m.N)
7 m.c = Param(m.A)
8
9 m.f = Var(m.A, within=NonNegativeReals)
10
11 @m.Objective()
12 def total_rule(m):
13     return sum(m.f[i,j] for (i, j) in m.A if j == value(
14         m.t))
15
16 @m.Constraint(A)
17 def limit_rule(m, i, j):
18     return m.f[i,j] <= m.c[i, j]

```

Rules and Tricks to Formulate Optimization Problems

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

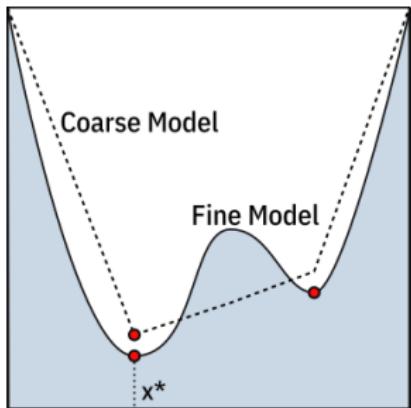
4 - Model and formulation

- Modelling and Modelling Languages
- Rules and Tricks to Formulate Optimization Problems
- Available tools

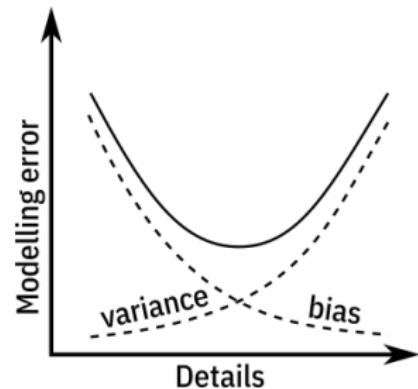
Rules and Tricks to Formulate Optimization Problems

Compromising accuracy and simplicity

O inimigo do bom é o melhor



Tendencies are most important



Details in the model decrease the bias, but may increase the overall error.

Road to Salvador

Modelling Tricks

- Absolute value, Min/Max, Logical Conditions (**IF, AND, OR**)
- Piecewise linear functions
- $\min(\max(f_i(x)))$
- Product of binary variables
- Product binary \times continuous variable
- Ratio objective functions
- Robust and stochastic optimization

Example : the Big M method

$$\begin{cases} y = |x| \\ x \in \mathbb{R} \\ y \in \mathbb{R}^+ \end{cases} \Rightarrow \begin{cases} s^+ \in \mathbb{R}, s^- \in \mathbb{R} \\ u \in \{0, 1\} \\ x = s^+ - s^- \\ y = s^+ + s^- \\ s^+ \leq Mu \\ s^- \leq M(1 - u) \end{cases}$$

Modelling Tricks

Choice among several options

Choose n solutions between m possibilities

$$\sum_m u_m \leq n$$

If | Then statement

Suppose two binary variables u and v

$$\begin{cases} \text{if } u == 1 \\ \text{then } v = 1 \quad \Rightarrow v \geq u \\ u, v \in \{0, 1\} \end{cases}$$

Rules and Tricks to Formulate Optimization Problems

Modelling Tricks

Simple conditions

At most one of A, B,...,H

$$a + b + c + d + e + f + g + h \leq 1$$

Exactly two of A, B,...,H

$$a + b + c + d + e + f + g + h = 2$$

If A then B

$$b \geq a$$

Not B

$$\bar{b} = 1 - b$$

If A then not B

$$a + b \leq 1$$

If not A then B

$$a + b \geq 1$$

If A then B, and if B then A

$$a = b$$

If A then B and C

$$b \geq a \text{ and } c \geq a$$

If A then B or C

$$b + c \geq a$$

If B or C then A

$$a \geq b \text{ and } a \geq c$$

If B and C then A

$$\text{or alternatively: } a \geq \frac{1}{2} \cdot (b + c)$$

$$a \geq b + c - 1$$

If two or more of B, C, D or E then A

$$a \geq \frac{1}{3} \cdot (b + c + d + e - 1)$$

If M or more of N projects (B, C, D, ...) then A

$$a \geq \frac{b+c+d+\dots-M+1}{N-M+1}$$

Source : FICO Xpress [?]

Modelling Tricks

minmax problems

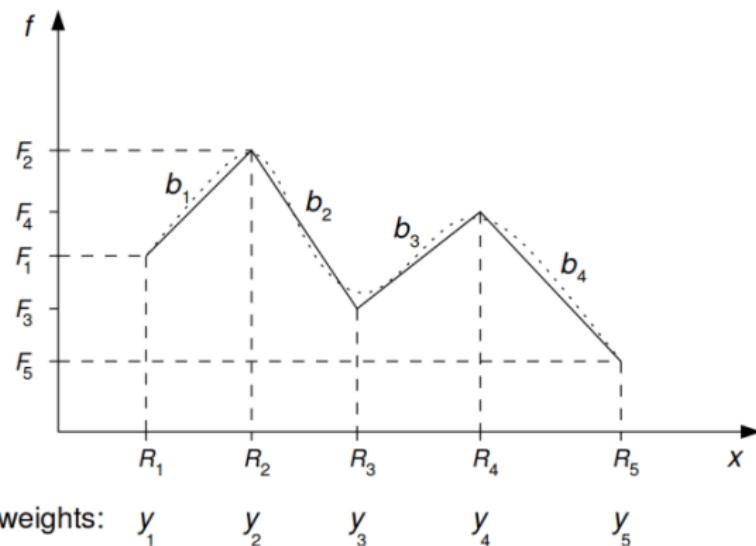
Minimizing the worst case scenario

$$\text{Minimize } \max\{t_1, t_2, \dots, t_n\} \Rightarrow \left\{ \begin{array}{l} \text{Minimize } s \\ \text{s.t. } s \geq t_1 \\ \quad s \geq t_2 \\ \quad \vdots \\ \quad s \geq t_n \end{array} \right.$$

Rules and Tricks to Formulate Optimization Problems

Modelling Tricks

Special Ordered Sets (type 2)



Source : FICO Xpress [?]

Other Considerations

- Size of the problem (dimensions, and definition set)
- Conditioning
- Specific problems (decomposition) : Specific solvers

Available tools

1 - Introduction

2 - Class of problems

3 - Heuristics & Multi-Objective

4 - Model and formulation

- Modelling and Modelling Languages
- Rules and Tricks to Formulate Optimization Problems
- Available tools

Available tools

Tools

AIMMS<https://www.aimmms.com/>**AMPL**<https://ampl.com>**FICO Xpress**<https://www.fico.com/en/products/fico-xpress-optimization>**Gurobi**<https://www.gurobi.com/>**Matlab**<https://fr.mathworks.com/products/optimization.html>**COIN-OR**<https://www.coin-or.org/>**Python**<https://pyomo.readthedocs.io/en/latest/>**Julia**<https://github.com/JuliaOpt/JuMP.jl>*Optimization Toolbox**Collection of open-source projects**Pyomo, PuLP, etc.**JuMP*

Available tools

AMPL

```
1 set N;  
2 set F;  
3 param c {F} > 0;  
4 param a {N,F} >= 0;  
5 param v {F} >= 0;  
6 param f_min {F} >= 0;  
7 param f_max {j in F} >= f_min[j];  
8 param n_min {N} >= 0;  
9 param n_max {i in N} >= n_min[i];  
10 param V_max >= 0;  
11 var x {j in F} >= n_min[j], <= n_max[j];
```

```
1 minimize Total_Cost: sum {j in F} c[j] * x[j];  
2  
3 subject to Diet {i in N}:  
4   n_min[i] <= sum {j in F} a[i,j] * x[j] <= n_max[i];  
5 subject to Volume :  
6   0 <= sum {j in F} v[j] * x[i] <= V_max;  
7
```

<https://ampl.com/>

Available tools

Matlab

OPT toolbox

Class	functions
LP	linprog
QP	quadprog
NLP Unconstrained	fminunc
NLP Constrained	fmincon, fminbnd, fseminf
Least-Squares	lsqlin , lsqlin, lsqnonlin, lsqcurvefit
Binary IP	bintprog

Table: List of available Matlab functions and problem classes.

GUI : optimtool Source : Optimization Toolbox :

<https://fr.mathworks.com/help/optim/index.html>

Part 2 - Introduction to Python and Numerical Optimization using Pyomo

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes

1 - Pyomo Tutorial

Python tutorial

ReinboldV / PyomoGallery
forked from Pyomo/PyomoGallery

Code Pull requests Projects Wiki Security Insights Settings

A collection of Pyomo examples

Manage topics

71 commits 1 branch 0 releases 7 contributors View license

Branch: master • New pull request Create new file Upload files Find file Clone or download

This branch is 27 commits ahead of Pyomo:master.

Pull request Compare

REINBOLD	Vincent	ignore_pycache_file	Latest commit 40c5d4e 10 days ago
Lecture_UFPB_2019		reorganization of structure	10 days ago
PyomoTutorial		reorganization of structure	10 days ago
Pythontutorial		reorganization of structure	10 days ago
binder		adding binder configuration layout	10 days ago
.gitignore		ignore_pycache_file	10 days ago
LICENSE		Updating license to FreeBSD	4 years ago
README.md		Update README.md	16 days ago
environment.yml		Update environment.yml	16 days ago

README.md

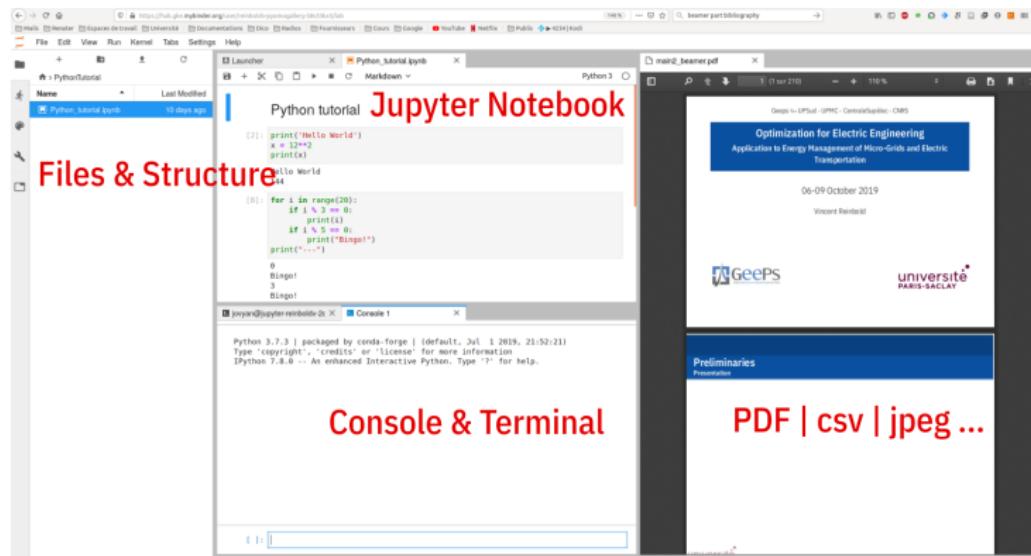
build status Launch binder

PyomoGallery

Go to

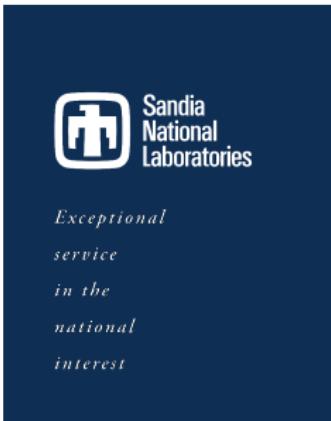
<https://github.com/ReinboldV/PyomoGallery> and click on launch|binder logo or go to <https://mybinder.org/v2/gh/ReinboldV/PyomoGallery/master?urlpath=lab/tree/index.ipynb>

Jupyter notebook



1 - Pyomo Tutorial

- Overview of Pyomo
- Python tutorial
- Pyomo tutorial
- Non-linear Optimization using Pyomo



1. Overview of Pyomo



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Optimization Modeling

Goal:

- Provide a natural syntax to describe mathematical models
- Formulate large models with a concise syntax
- Separate modeling and data declarations
- Enable data import and export in commonly used formats

Impact:

- Robustly model large systems
- Integrated automatic differentiation for complex nonlinear models

Examples:

- AMPL, GAMS, AIMMS, ...
- OptimJ, FlopCPP, PuLP, JuMP, ...
- **Pyomo: A Python-based optimization modeling package**

Overview

- Three really good questions:
 - Why another Algebraic Modeling Language (AML)?
 - Why Python?
 - Why open-source?
- Pyomo: Team overview and collaborators / users
- Where to find more information...

Pyomo Overview

Idea: a Pythonic framework for formulating optimization models

- Keep a natural syntax even though it is Python
- Meet goals of modeling languages (concise, separate model and data)
- And...
- Provide an **open-source** tool for problem solving and research
- Allow for complex workflows through rich programming capabilities already available with Python

Highlights:

- Python provides a clean, intuitive syntax
- Python scripts provide a flexible context for exploring the structure of Pyomo models

```
# simple.py
from pyomo.environ import *

M = ConcreteModel()
M.x1 = Var()
M.x2 = Var(bounds=(-1,1))
M.x3 = Var(bounds=(1,2))
M.o = Objective(
    expr=M.x1**2 + (M.x2*M.x3)**4 + \
        M.x1*M.x3 + \
        M.x2*sin(M.x1+M.x3) + M.x2)
```

Pyomo Overview

- Equation-oriented models
 - User must provide ***all*** model equations explicitly (e.g., glass-box)
 - Not an optimizer around existing simulators (e.g., not black-box)
- Classes of problems and solvers:
 - Discrete Optimization (MIP): Gurobi, CPLEX, CBC, GLPK
 - Nonlinear Optimization (NLP): IPOPT, KNITRO, ... (Any AMPL or GAMS)
 - Mixed-Integer Nonlinear Optimization (MINLP): Baron, COUENNE
- Included packages and capabilities:
 - Pyomo Core: Basic modeling and optimization capabilities
 - Pyomo Blocks: Object-oriented (hierarchical) models
 - Pyomo.DAE: Modeling and optimization of dynamic systems (DAE and PDE)
 - Pyomo.GDP: Generalized disjunctive programming
 - PySP: Optimization under uncertainty
 - Pyomo.Bilevel, Pyomo.MPEC, ...

Pyomo Overview: Two focus areas

- **Flexibility (built on Python)**

- Ease of Prototyping and new workflows
- Higher-level interfaces and applications *through Python*
- Data import & manipulation, plotting, UI development

- **Solution Efficiency**

- Fast solution of large-scale problems (efficient interfaces with solvers)
- **HPC Ready:** Parallel execution of many optimization problems and parallel decomposition of large, structured problems
- Reduced end-to-end solution time

More than just mathematical modeling



Scripting

- Construct models using native Python data
- Iterative analysis of models leveraging Python functionality
- Data analysis and visualization of optimization results

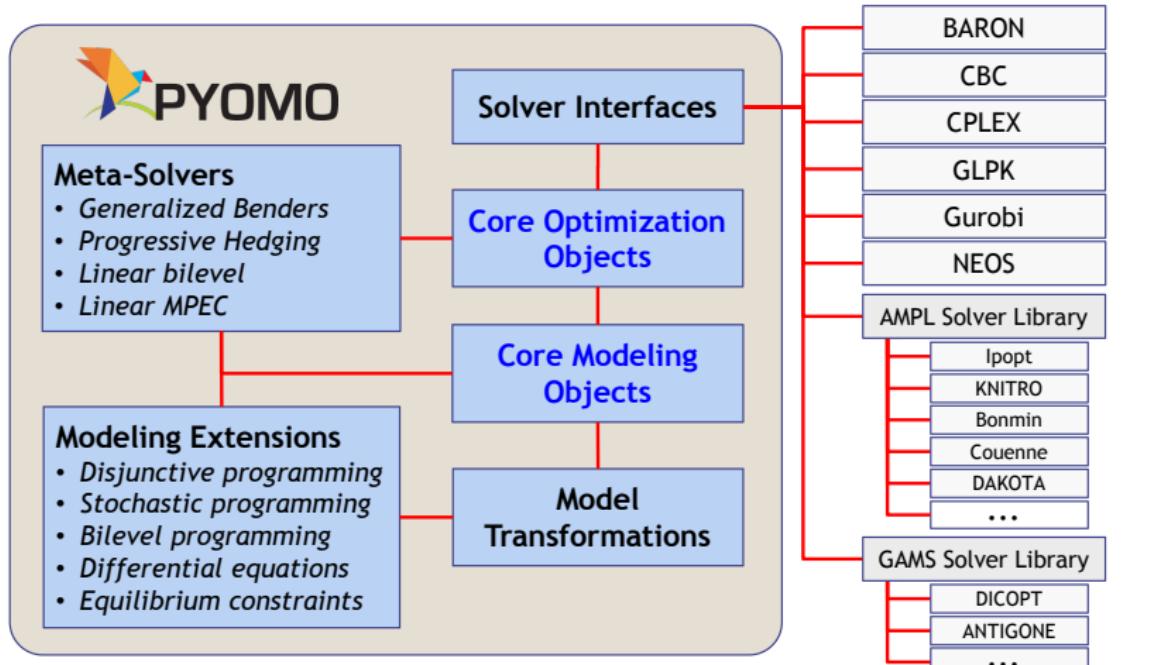
Model transformations (a.k.a. reformulations)

- Automate generation of one model from another
- Leverage Pyomo's object model to apply transformations sequentially
- E.g.: DAE -> discretized model, GDP -> Big M

Meta-solvers

- Integrate scripting and/or transformations into optimization solver
- Leverage power of Python to build “generic” capabilities
- E.g.: progressive hedging, SP extensive form -> MIP

Pyomo at a Glance



Why Model in Python?

- **Full-Featured Language**
 - Language features includes functions, classes, looping, namespaces, etc
 - Introspection facilitates the development of generic algorithms
 - Python's clean syntax facilitates rapid prototyping
- **Open Source License**
 - No licensing issues w.r.t. the language itself
- **Extensibility and Robustness**
 - Highly stable and well-supported
- **Support and Documentation**
 - Extensive online documentation and several excellent books
 - Long-term support for the language is not a factor
- **Standard Library**
 - Includes a large number of useful modules
- **Portability**
 - Widely available on many platforms

Why Open Source?

- Transparency and reliability
- Foster community involvement
 - Extend the modeling language
 - Develop new solvers / algorithms
 - Interface with additional external utilities
 - “Stone Soup” model
- Flexible licensing
 - Pyomo released under 3-clause BSD license
 - No restrictions on deployment or commercial use

Who Uses Pyomo?



- Students
 - Rose-Hulman, UC Davis, Purdue, U Texas, Iowa State, Notre Dame, NPS, ...
- Researchers
 - Government laboratories
 - Sandia National Labs, Lawrence Livermore National Lab, Los Alamos National Lab, National Energy Technology Lab, National Renewable Energy Lab, Federal Energy Regulation Commission
 - Universities
 - Carnegie Mellon, UC Davis, Texas A&M, Rose-Hulman, U. Texas, USC, GMU, Iowa State, NCSU, NTNU, Notre Dame, Imperial College, U Washington, NPS, U de Santiago de Chile, U Pisa, ...
 - Companies... yes.
- Software Projects
 - IDAES – Optimization of energy production systems
 - Minpower – Power systems toolkit
 - SolverStudio – Excel plugin for optimization modeling
 - TEMOA – Energy economy optimization models
 - WNTR – Resilience analysis for water distribution systems

For More Information

See the Pyomo homepage

- www.pyomo.org



The Pyomo homepage provides a portal for:

- Online documentation
- Installation instructions
- Help information
- Developer links

GitHub:

- github.com/Pyomo



Parallel stochastic programming
in PySP >

```
set Scenario := Bedienungszeitraum  
  prioritaetsreihenfolge  
  Anzahlverkaufskategorien 3  
  
set StageVariables(Stichtagstag) := Quantitaetkaufkategorie  
  Quantitaetverkaufskategorie  
  
<code>  
model.JetztUndMorgen := Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorm = Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorv = Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorvv = Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorvvv = Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorvvvv = Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorvvvvv = Parameter(1..2000, elternmodell=OptModell)  
model.JetztUndVorvvvvvv = Parameter(1..2000, elternmodell=OptModell)</code>
```

What Is Pyomo?

Pyomo is a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities.

[Read More](#)

Installation

The easiest way to install Pyomo is to use pip. Pyomo also needs access to optimization solvers.

[Read more](#)

Latest: Pyomo 4.4

Docs and Examples

Pyomo documentation and examples are available online. Pyomo is also described in book and journal publications.

[Read more](#)

Acknowledgments

The Pyomo project would not be where it is without the generous contributions of numerous people and organizations.

[Read More](#)

Getting Help

Users and developers provide help online:

- [Questions on StackExchange](#)
- [Discussions on Pyomo Forum](#)

[ASK A QUESTION](#)

Who Uses Pyomo?

Pyomo is used by researchers to solve complex real-world applications.

[Read More](#)

Acknowledgements

- William Hart (SNL)
- Jean-Paul Watson (SNL)
- John Siirola (SNL)
- Francisco Munoz (SNL, UAI)
- Bethany Nicholson (CMU, SNL)
- Prof. David L. Woodruff (UC Davis)
- Prof. Roger Wets (UC Davis)
- Prof. Carl D. Laird (Purdue, SNL)
- Gabe Hackebeil (U.Mich.)

1 - Pyomo Tutorial

- Overview of Pyomo
- **Python tutorial**
- Pyomo tutorial
- Non-linear Optimization using Pyomo

Why Python?

- Interpreted language
- Intuitive syntax
- Object-oriented
- Simple, but extremely powerful
- Lots of built-in libraries and third-party extensions
- Shallow learning curve and many resources
- Dynamic typing
- Integration with C/Java

Python Versions: 2.x vs 3.x

- Python 3.0 was released in 2008
 - Included significant backward incompatibilities
 - Initial adoption was slow
 - But, community is moving on to Python 3.0
- Pyomo support:
 - Python 2.7
 - Very stable; patches have included package updates to support Python 3.x compatibility
 - Python 3.5, 3.6
 - Very stable, but marginally slower than 2.7

We try to stick to “universal” syntax that will work in both 2.x and 3.x



<https://pythonclock.org>

Overview

- interactive "shell"
- basic types: numbers, strings
- container types: lists, dictionaries, tuples
- variables
- control structures
- functions & procedures
- classes & instances
- modules
- exceptions
- files & standard library

Interactive Shell

- Great for learning the language
- Great for experimenting with the library
- Great for testing your own modules
- Two variations:
 - IDLE (GUI)
 - `python` (command line)
- Type statements or expressions at prompt:

```
>>> print( "Hello, world" )  
Hello, world  
>>> x = 12**2  
>>> x/2  
72  
>>> # this is a comment
```

Python Program



- To write a program, put commands in a file

```
# hello.py
print( "Hello, world" )
x = 12**2
print( x )
```

- Execute on the command line

```
C:\Users\me> python hello.py
Hello, world
144
```



Python Variables

- No need to declare
- Need to assign (initialize)
 - use of uninitialized variable raises exception
- Not typed

```
greeting = 34.2
if friendly:
    greeting = "hello world"
else:
    greeting = 12**2
print( greeting )
```

- ***Everything*** is a "variable":
 - Even functions, classes, modules

Control Structures



```
if condition:  
    statements  
elif condition:  
    statements ...  
else:  
    statements  
  
while condition:  
    statements  
  
for var in sequence:  
    statements  
  
        break  
        continue
```

Note: Spacing matters!

Control structure scope dictated by indentation



Grouping Indentation

In Python:

```
for i in range(20):
    if i % 3 == 0:
        print(i)
    if i % 5 == 0:
        print("Bingo!")
print("---")
```

In C:

```
for (i = 0; i < 20; i++)
{
    if (i % 3 == 0) {
        printf("%d\n", i);
        if (i % 5 == 0) {
            printf("Bingo!\n");
        }
    }
    printf("---\n");
}
```

Numbers

- The usual suspects
 - 12, 3.14, 0xFF, 0377, (-1+2)*3/4**5, abs(x), x <= 5
- C-style shifting & masking
 - 1<<16, x&0xff, x|1, ~x, x^y
- Integer division truncates
 - Python 2.x
 - `1/2 → 0, 1./2. → 0.5, float(1)/2 → 0.5`
 - `from __future__ import division`
 - » `1/2 → 0.5`
 - Python 3.x
 - `1/2 → 0.5`
- Long (arbitrary precision), complex
 - `2L**100 → 1267650600228229401496703205376L`
 - In Python 2.2 and beyond, `2**100` does the same thing
 - `1j**2 → (-1+0j)`

Strings

- "hello"+"world" "helloworld" *# concatenation*
 - "hello"*3 "hellohellohello" *# repetition*
 - "hello"[0] "h" *# indexing*
 - "hello"[-1] "o" *# (from end)*
 - "hello"[1:4] "ell" *# slicing*
 - len("hello") 5 *# size*
 - "hello" < "jello" True *# comparison*
 - "e" in "hello" True *# search*
-
- "escapes: \n etc, \033 etc, \if etc"
 - 'single quotes' """triple quotes"""\ r"raw strings"

Lists

- Flexible arrays, *not* linked lists
 - `a = [99, "bottles of beer", ["on", "the", "wall"]]`
- Same operators as for strings
 - `a+b, a*3, a[0], a[-1], a[1:], len(a)`
- Item and slice assignment
 - `a[0] = 98`
 - `a[1:2] = ["bottles", "of", "beer"]`
-> [98, "bottles", "of", "beer", ["on", "the", "wall"]]
 - `del a[-1]`
-> [98, "bottles", "of", "beer"]

List Operations



```
>>> a = range(5)          # [0,1,2,3,4]
>>> a.append(5)           # [0,1,2,3,4,5]
>>> a.pop()               # [0,1,2,3,4]
5
>>> a.insert(0, 42)        # [42,0,1,2,3,4]
>>> a.pop(0)               # [0,1,2,3,4]
42
>>> a.reverse()            # [4,3,2,1,0]
>>> a.sort()               # [0,1,2,3,4]
```

Dictionaries

- Hash tables, "associative arrays"
 - `d = {"wood": "float", "witch": "nose"}`
- Lookup:
 - `d["wood"]` # -> "*float*"
 - `d["back"]` # *raises KeyError exception*
- Delete, insert, overwrite:
 - `del d["wood"]` # {"witch": "nose"}
 - `d["duck"] = "float"` # {"duck": "float", "witch": "nose"}
 - `d["witch"] = "burn"` # {"duck": "float", "witch": "burn"}

Dictionary Operations

- Keys, values, items:

- `d.keys()` → ["duck", "witch"]
- `d.values()` → ["float", "burn"]
- `d.items()` → [("duck", "float"), ("witch", "burn")]

Note: These actually return lists in Python 2.x, and generators in Python 3.x.

- Presence check:

- `d.has_key("duck")` # True
- `d.has_key("spam")` # False
- "duck" in d # True

- Values of any type; keys almost any

- `{"age": 43,
("hello", "world"): 1,
42: "answer",
"flag": ["red", "white", "blue"] }`

Dictionary Details

- Keys must be **immutable**:
 - numbers, strings, tuples of immutables
 - these cannot be changed after creation
 - keys are hashed (to ensure fast lookup)
 - lists or dictionaries cannot be used as keys
 - these objects can be changed "in place"
 - no restrictions on values
- Keys will be listed in **arbitrary order**
 - key hash values are in an arbitrary order
 - that numeric keys are returned sorted is an artifact of the implementation and *is not guaranteed*

References vs.Copies

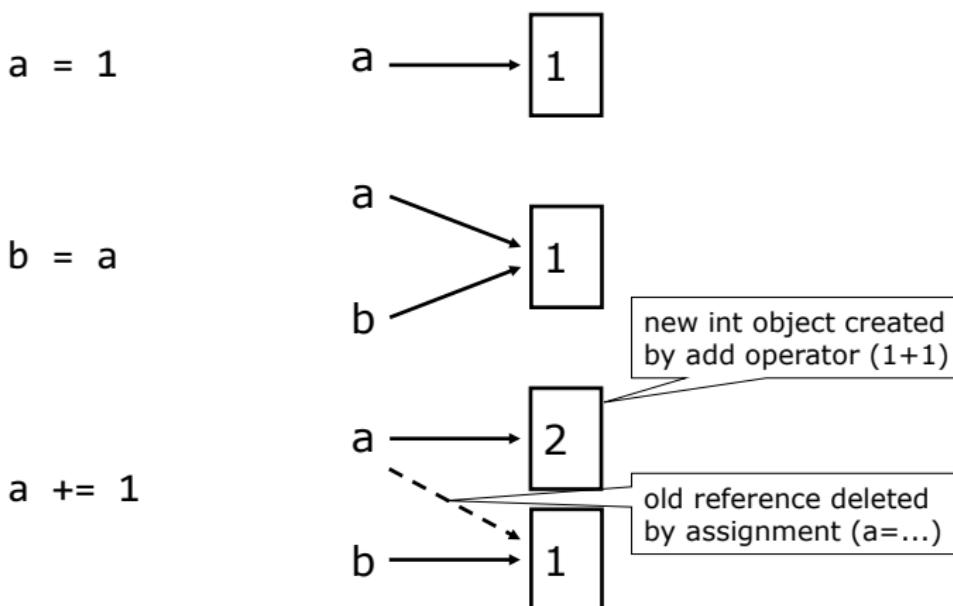
- Assignment manipulates references
 - `x = y` **does not make a copy** of `y`
 - `x = y` makes `x` **reference** the object `y` references
- Reference values can be modified!

```
>>> a = [1, 2, 3]
>>> b = a
>>> a.append(4)
>>> print(b)
[1, 2, 3, 4]
```

- Copied objects are distinct

```
>>> import copy
>>> c = copy.copy(a)
>>> a.pop()
>>> print(c)
[1, 2, 3, 4]
```

Working with *immutable* data



Working with objects

```
a = [1, 2, 3]
```

a →



```
b = a
```

a →

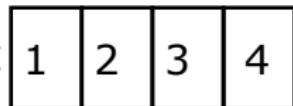
b →



```
a.append(4)
```

a →

b →



Functions / Procedures



```
def name(arg1, arg2, ...):
    """documentation"""\n    # optional doc string\n    statements\n\n    return expression      # from function\n    return                 # from procedure (returns None)
```

Example

```
def gcd(a, b):  
    """greatest common divisor"""  
    while a != 0:  
        a, b = b%a, a    # parallel assignment  
    return b
```

```
>>> gcd.__doc__  
'greatest common divisor'
```

```
>>> gcd(12, 20)  
4
```

Modules

- Collection of stuff in *foo.py* file

- functions, classes, variables

- Importing modules:

```
import re
print( re.match("[a-z]+", s) )
from re import match
print( match("[a-z]+", s) )
```

- Import with rename:

```
import re as regex
from re import match as m
```

Major Python Packages

- Matplotlib
 - 2D plotting library
 - <http://matplotlib.org/>
- Pandas
 - Data structures and analysis
 - <http://pandas.pydata.org/>
- NetworkX, Plotly, ...
- SciPy
 - Scientific Python for mathematics and engineering
 - <http://www.scipy.org>
- Numpy
 - Numeric array package
 - <http://www.numpy.org/>
- Ipython
 - Interactive Python shell
 - <http://ipython.org/>

Resources

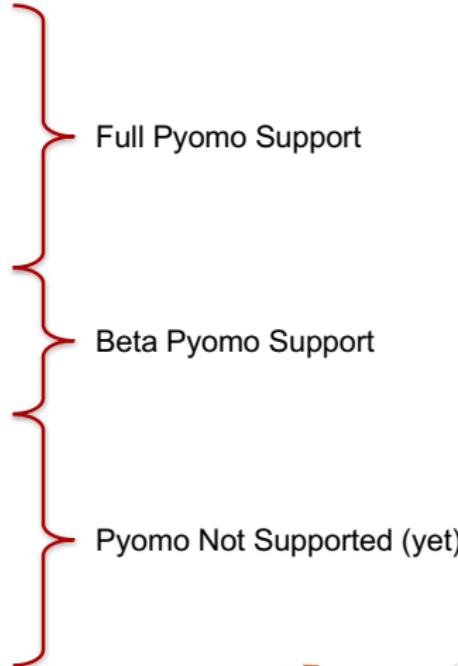
- Software Carpentry
 - <http://software-carpentry.org/>
- Python webpage
 - <http://www.python.org>
- Books
 - Python Essential Reference (4th Edition), David Beazley, 2009
 - Python in a Nutshell, Alex Martelli, 2003
 - Python Pocket Reference, 4th Edition, Mark Lutz, 2009
 - ...

Acknowledgements

- William Hart
- Ted Ralphs
- John Siirola
- Carl Laird
- Bethany Nicholson
- Dave Woodruff
- Guido van Rossum

OTHER MATERIAL

Python Implementations

- Cpython
 - C Python interpreter
 - <https://www.python.org/downloads/>
 - SciPy Stack
 - <http://www.scipy.org/install.html>
 - Anaconda: Linux/MacOS/MS Windows
 - PyPy
 - A Python interpreter written in Python
 - <http://pypy.org/>
 - Jython
 - Java Python interpreter
 - <http://www.jython.org/>
 - IronPython
 - .NET Python interpreter
 - <http://ironpython.net/>
- 
- Full Pyomo Support
- Beta Pyomo Support
- Pyomo Not Supported (yet)

Tuples

- `key = (lastname, firstname)`
- `point = x, y, z` *# parentheses optional*
- `x, y, z = point` *# unpack*
- `lastname = key[0]` *# index tuple values*
- `singleton = (1,)` *# trailing comma!!!*
 # (1) → integer!
- `empty = ()` *# parentheses!*

- Tuples vs. lists
 - tuples immutable
 - lists mutable

Classes

```
class name(object):  
    """documentation"""  
    statements
```

Most, *statements* are method definitions:

```
def name(self, arg1, arg2, ...):  
    ...
```

May also be *class variable* assignments

Example

```
class Stack(object):
    """A well-known data structure..."""
    def __init__(self):          # constructor
        self.items = []
    def push(self, x):
        self.items.append(x)      # the sky is the limit
    def pop(self):
        x = self.items[-1]        # what if it's empty?
        del self.items[-1]
        return x
    def empty(self):
        return len(self.items) == 0 # Boolean result
```

Example (cont'd)

- To create an instance, simply call the class object:

```
x = Stack()          # no 'new' operator!
```

- To use methods of the instance, call using dot notation:

```
x.empty()           # -> 1
x.push(1)           # [1]
x.empty()           # -> 0
x.push("hello")     # [1, "hello"]
x.pop()             # -> "hello" # [1]
```

- To inspect instance variables, use dot notation:

```
x.items            # -> [1]
```

Class/Instance Variables



```
class Connection(object):
    verbose = 0                      # class variable

    def __init__(self, host):
        self.host = host      # instance variable

    def debug(self, v):
        self.verbose = v      # make instance variable!

    def connect(self):
        if self.verbose:      # class or instance variable?
            print("connecting to %s" % (self.host,))
```



Instance Variable Rules

- On use via instance (`self.x`), search order:
 - (1) instance, (2) class, (3) base classes
 - this also works for method lookup
- On assignment via instance (`self.x = ...`):
 - always makes an instance variable
- Class variables "default" for instance variables
- But...!
 - mutable *class* variable: one copy *shared* by all
 - mutable *instance* variable: each instance its own

Catching Exceptions



```
def foo(x):
    return 1/x

def bar(x):
    try:
        print( foo(x) )
    except ZeroDivisionError as message:
        print("Can't divide by zero: %s" % message)

bar(0)
```



Try-Finally: Cleanup

```
f = open(file)
try:
    process_file(f)
finally:
    f.close()          # always executed
print("OK")           # executed on success only
```

Raising Exceptions



```
raise IndexError
```

```
raise IndexError("k out of range")
```

```
raise IndexError, "k out of range"
```

this only works in Python 2.x!

```
try:
```

something

```
except:
```

catch everything

```
    print( "Oops" )
```

```
    raise
```

reraise



More on Exceptions

- User-defined exceptions
 - subclass `Exception` or any other standard exception
- Note: in older versions of Python exceptions can be strings
- Last caught exception info:
 - `sys.exc_info() == (exc_type, exc_value, exc_traceback)`
- Printing exceptions: `traceback` module

File Objects

- `f = open(filename[, mode[, bufsize]])`
 - mode can be "r", "w", "a" (like C stdio); default "r"
 - append "b" for text translation mode
 - append "+" for read/write open
 - bufsize: 0=unbuffered; 1=line-buffered; buffered
- methods:
 - `read([nbytes])`, `readline()`, `readlines()`
 - `write(string)`, `writelines(list)`
 - `seek(pos[, how])`, `tell()`
 - `flush()`, `close()`
 - `fileno()`

Highlights from the Standard Library



- Core:
 - os, sys, string, getopt, StringIO, struct, pickle, json, csv, collections, math, tempfile, itertools, ...
- Regular expressions:
 - re module; Perl-5 style patterns and matching rules
- Internet:
 - socket, rfc822, httplib, htmlllib, ftplib, smtplib, ...
- Miscellaneous:
 - pdb (debugger), profile+pstats
 - Tkinter (Tcl/Tk interface), audio, *dbm, ...

1 - Pyomo Tutorial

- Overview of Pyomo
- Python tutorial
- Pyomo tutorial**
- Non-linear Optimization using Pyomo



3. Pyomo Fundamentals



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Learning Objectives

- Goals, benefits, drawbacks of Pyomo
- Basic understanding of Python
- Create and solve optimization problems within Pyomo
 - Vars, Constraints, Objectives
 - Sets, Parameters
- Basic scripting, workflow, and programming capabilities
 - Changing data, multiple solutions, importing data, plotting, reporting
 - Pieces to build your own workflows

Simple Modeling Example: Classic Knapsack Problem



- Given a set of items A, with weight and value (benefit)
- Goal: select a subset of these items
 - Maximize the benefit
 - Under weight limit

Item (A)	Weight (w)	Benefit (b)
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11

Max weight: 14

Symbol	Meaning
A	set of items available for purchase
b_i	benefit of item i
w_i	weight of item i
W_{\max}	max weight that can be carried
x_i	discrete variable (select i or not)

Simple Modeling Example: Classic Knapsack Problem



- Given a set of items A, with weight and value (benefit)
- Goal: select a subset of these items
 - Maximize the benefit
 - Under weight limit

Item (A)	Weight (w)	Benefit (b)
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11

Max weight: 14

Symbol	Meaning
A	set of items available for purchase
b_i	benefit of item i
w_i	weight of item i
W_{\max}	max weight that can be carried
x_i	discrete variable (select i or not)

$$\begin{aligned} \max_x \quad & \sum_{i \in A} b_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x_i \in \{0,1\} \quad \forall i \in A \end{aligned}$$

Simple Modeling Example: Classic Knapsack Problem



- Variables
- Objectives
- Constraints
- Sets
- Parameters

Item (A)	Weight (w)	Benefit (b)
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11

Max weight: 14

Symbol	Meaning
A	set of items available for purchase
b_i	benefit of item i
w_i	weight of item i
W_{\max}	max weight that can be carried
x_i	discrete variable (select i or not)

$$\begin{aligned} \max_x \quad & \sum_{i \in A} b_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x_i \in \{0,1\} \quad \forall i \in A \end{aligned}$$

Anatomy of a Concrete Pyomo Model



$$\begin{aligned} \max_x \quad & \sum_{i \in A} b_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x_i \in \{0,1\} \quad \forall i \in A \end{aligned}$$

Item	Weight	Value
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11

```
from pyomo.environ import *

A = ['hammer', 'wrench', 'screwdriver', 'towel']
b = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

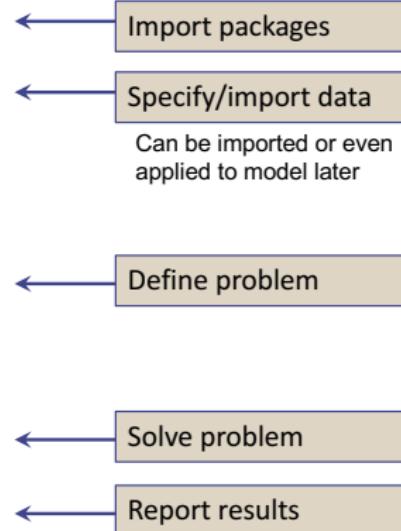
model = ConcreteModel()
model.x = Var( A, within=Binary )

model.value = Objective(
    expr = sum( b[i]*model.x[i] for i in A),
    sense = maximize )

model.weight = Constraint(
    expr = sum( w[i]*model.x[i] for i in A) <= W_max )

opt = SolverFactory('glpk')
result_obj = opt.solve(model, tee=True)

model.pprint()
```



Pyomo imports and *namespaces*

- Pyomo objects exist within the `pyomo.environ` namespace:

```
import pyomo.environ
model = pyomo.environ.ConcreteModel()
```

- ...but this gets verbose. To save typing, we will import the core Pyomo classes into the main namespace:

```
from pyomo.environ import *
model = ConcreteModel()
```

- To clarify Pyomo-specific syntax in this tutorial, we will highlight Pyomo symbols in green

- Note: We violate good Python practice in this tutorial. One should:

```
import pyomo.environ as pe
model = pe.ConcreteModel()
```

Getting Started: the *Model*

```
from pyomo.environ import *  
  
model = ConcreteModel()
```

Every Pyomo model starts with this; it tells Python to load the Pyomo Modeling Environment

Create an instance of a *Concrete* model

- Concrete models are immediately constructed
- Data must be present *at the time* components are defined

Local variable to hold the model we are about to construct

- While not required, by convention we use “model”

Populating the Model: *Variables*

Components can take a variety of keyword arguments when constructed.

```
model.a_variable = Var(within = NonNegativeReals)
```

The name you assign the object to becomes the object's name, and must be unique in any given model.

"within" is optional and sets the variable domain ("domain" is an alias for "within")

Several pre-defined domains, e.g., "Binary"

Populating the Model: *Variables*

Components can take a variety of keyword arguments when constructed.

```
model.a_variable = Var(within = NonNegativeReals)
```

The name you assign the object to becomes the object's name, and must be unique in any given model.

"within" is optional and sets the variable domain ("domain" is an alias for "within")

Several pre-defined domains, e.g., "Binary"

```
model.a_variable = Var(bounds = (0, None))
```

```
model.a_variable = Var(initialize = 42.0)
```

```
model.a_variable = Var(initialize=42.0, bounds=(0, None))
```

Defining the *Objective*

```
model.x = Var( initialize=-1.2, bounds=(-2, 2) )  
model.y = Var( initialize= 1.0, bounds=(-2, 2) )  
  
model.obj = Objective(  
    expr= (1-model.x)**2 + 100*(model.y-model.x**2)**2,  
    sense= minimize )
```

If “sense” is omitted, Pyomo assumes minimization

“expr” can be an expression, or any function-like object that returns an expression

Note that the Objective expression is not a *relational expression*

Defining the Problem: *Constraints*

```
model.a = Var()  
model.b = Var()  
model.c = Var()  
model.c1 = Constraint(  
    expr = model.b + 5 * model.c <= model.a )
```

$$b + 5c \leq a$$

“expr” can be an expression,
or any function-like object
that returns an expression

```
model.c2 = Constraint(expr = (None, model.a + model.b, 1))
```

$$a + b \leq 1$$

“expr” can also be a tuple:

- 3-tuple specifies (LB, expr, UB)
- 2-tuple specifies an equality constraint.

Higher-dimensional components

- (Almost) All Pyomo components can be *indexed*

```
A = [1, 2, 5]
```

```
model.x = Var(A)
```

- All non-keyword arguments are assumed to be *indices*
- Individual indices may be multi-dimensional (e.g., a list of pairs)

- E.g., Indexed variables

```
A = [1, 2, 5]
```

```
B = ['wrench', 'hammer']
```

```
model.x = Var(A)
```

```
model.y = Var(A, B)
```



The indexes are any iterable object,
e.g., list or Set

- **Note:** while indexed variables look like matrices, they are **not**.
 - In particular, we do not support matrix algebra (yet...)

List Comprehensions

```
model.IDX = range(10)
model.a = Var()
model.b = Var(model.IDX)
model.c1 = Constraint(
    expr = sum(model.b[i] for i in model.IDX) <= model.a )
```

Python *list comprehensions* are very common for working over indexed variables and nicely parallel mathematical notation:

$$\sum_{i \in IDX} b_i \leq a$$

Putting it all together: Concrete Knapsack

```
# knapsack.py
from pyomo.environ import *

A = ['hammer', 'wrench', 'screwdriver', 'towel']
b = {'hammer':8, 'wrench':3, 'screwdriver':6, 'towel':11}
w = {'hammer':5, 'wrench':7, 'screwdriver':4, 'towel':3}
W_max = 14

model = ConcreteModel()
model.x = Var( A, within=Binary )
model.value = Objective(
    expr = sum( b[i]*model.x[i] for i in A ),
    sense = maximize )

model.weight = Constraint(
    expr = sum( w[i]*model.x[i] for i in A ) <= W_max )

opt = SolverFactory('glpk')
result_obj = opt.solve(model, tee=True)
model.pprint()
```

Import packages

Specify/import data

Create model object

Define variable x

Define objective

Define constraint

Create solver

Solve the problem

Print results

Putting it all together: Concrete Knapsack



```
# knapsack.py
from pyomo.environ import *
...
result_obj = opt.solve(model, tee=True)
model.pprint()
```

```
> python knapsack.py
```

```
1 Set Declarations
    x_index : Dim=0, Dimen=1, Size=4, Domain=None, Ordered=False, Bounds=None
        ['hammer', 'screwdriver', 'towel', 'wrench']
1 Var Declarations
    x : Size=4, Index=x_index
        Key      : Lower : Value : Upper : Fixed : Stale : Domain
            hammer : 0 : 1.0 : 1 : False : False : Binary
            screwdriver : 0 : 1.0 : 1 : False : False : Binary
            towel : 0 : 1.0 : 1 : False : False : Binary
            wrench : 0 : 0.0 : 1 : False : False : Binary
1 Objective Declarations
    value : Size=1, Index=None, Active=True
        Key : Active : Sense   : Expression
        None : True : maximize : 8*x[hammer] + 3*x[wrench] + 6*x[screwdriver] + 11*x[towel]
1 Constraint Declarations
    weight : Size=1, Index=None, Active=True
        Key : Lower : Body                                     : Upper : Active
        None : -Inf : 5*x[hammer] + 7*x[wrench] + 4*x[screwdriver] + 3*x[towel] : 14.0 : True
4 Declarations: x_index x value weight
```

Putting it all together: Concrete Knapsack



```
# knapsack.py
from pyomo.environ import *
...
result_obj = opt.solve(model, tee=True)
model.display()
```

```
> python knapsack.py
```

Variables:

```
x : Size=4, Index=x_index
    Key      : Lower : Value : Upper : Fixed : Stale : Domain
        hammer :    0 :  1.0 :    1 : False : False : Binary
        screwdriver :   0 :  1.0 :    1 : False : False : Binary
        towel :    0 :  1.0 :    1 : False : False : Binary
        wrench :   0 :  0.0 :    1 : False : False : Binary
```

Objectives:

```
value : Size=1, Index=None, Active=True
    Key  : Active : Value
    None :  True  : 25.0
```

Constraints:

```
weight : Size=1
    Key  : Lower : Body : Upper
    None :  None  : 12.0 : 14.0
```

Pyomo Fundamentals: Exercises #1

- 1.1 **Knapsack example:** Solve the knapsack problem shown in the tutorial using your IDE (e.g., Spyder) or the command line: `> python knapsack.py`. Which items are acquired in the optimal solution? What is the value of the selected items?
- 1.2 **Knapsack with improved printing:** The `knapsack.py` example shown in the tutorial uses `model pprint()` to see the value of the solution variables. Starting with the code in `knapsack.print.incomplete.py`, complete the missing lines to produce formatted output. Note that the Pyomo `value` function should be used to get the floating point value of Pyomo modeling components (e.g., `print(value(model.x[i]))`). Also print the value of the items selected (the objective), and the total weight. (A solution can be found in `knapsack.print.soln.py`.)
- 1.3 **Changing data:** If we were to increase the value of the wrench, at what point would it become selected as part of the optimal solution? (A solution can be found in `knapsack.wrench.soln.py`.)
- 1.4 **Loading data from Excel:** In the knapsack example shown in the tutorial slides, the data is hardcoded at the top of the file. Instead of hard-coding the data, use Python to load the data from a different source. You can start from the file `knapsack.pandas.excel.incomplete.py`. (A solution that uses pandas to load the data from Excel is shown in `knapsack.pandas.excel.soln.py`.)
- 1.5 **NLP vs MIP:** Solve the knapsack problem with IPOPT instead of glpk. (Hint: switch `glpk` to `ipopt` in the call `SolverFactory`. Print the solution values for `model.x`. What happened? Why?)

More Complex Example: Warehouse Location



- Determine the set of P warehouses chosen from the set W of candidates to minimize the cost of serving all customers C
- Here $d_{w,c}$ is the cost of serving customer c from warehouse at location w .

$$\min \sum_{w \in W, c \in C} d_{w,c} x_{w,c}$$

$$s.t. \quad \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C$$

y_w : discrete variable
(location w selected or not)

$$x_{w,c} \leq y_w \quad \forall w \in W, c \in C$$

$x_{w,c}$: fraction of demand of customer c served by warehouse w

$$\sum_{w \in W} y_w = P$$

$$0 \leq x \leq 1 \quad y \in \{0,1\}^{|W|}$$

More Complex Example: Warehouse Location



- Determine the set of P warehouses chosen from the set W of candidates to minimize the cost of serving all customers C
- Here $d_{w,c}$ is the cost of serving customer c from warehouse at location w .

$$\min \sum_{w \in W, c \in C} d_{w,c} x_{w,c} \quad (\text{minimize total cost})$$

$$s.t. \quad \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C \quad (\text{guarantee all customers served})$$

$$x_{w,c} \leq y_w \quad \forall w \in W, c \in C \quad (\text{customer } c \text{ can only be served from warehouse } w \text{ if warehouse } w \text{ is selected})$$

$$\sum_{w \in W} y_w = P \quad (\text{select } P \text{ warehouses})$$

$$0 \leq x \leq 1 \quad y \in \{0,1\}^{|W|}$$

Key difference?

Knapsack

$$\begin{aligned} \max_x \quad & \sum_{i \in A} v_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x \in \{0,1\}^{|A|} \end{aligned}$$

Warehouse Location

$$\begin{aligned} \min \quad & \sum_{w \in W, c \in C} d_{w,c} x_{w,c} \\ \text{s.t.} \quad & \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C \\ & x_{w,c} \leq y_w \quad \forall w \in W, c \in C \\ & \sum_{w \in W} y_w = P \\ & 0 \leq x \leq 1 \quad y \in \{0,1\}^{|W|} \end{aligned}$$

Key difference: Scalar vs Indexed Constraint



Knapsack

$$\begin{aligned} \max_x \quad & \sum_{i \in A} v_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x \in \{0,1\}^{|A|} \end{aligned}$$

$$w_{\text{hammer}}x_{\text{hammer}} + \dots + w_{\text{wrench}}x_{\text{wrench}} \leq W_{\max}$$

Warehouse Location

$$\begin{aligned} \min \quad & \sum_{w \in W, c \in C} d_{w,c} x_{w,c} \\ \text{s.t.} \quad & \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C \\ & x_{w,c} \leq y_w \quad \forall w \in W, c \in C \\ & \sum_{w \in W} y_w = P \\ & 0 \leq x \leq 1 \quad y \in \{0,1\}^{|W|} \end{aligned}$$



Key difference: Scalar vs Indexed Constraint



Knapsack

$$\begin{aligned} \max_x \quad & \sum_{i \in A} v_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x \in \{0,1\}^{|A|} \end{aligned}$$

$$w_{\text{hammer}} x_{\text{hammer}} + \dots + w_{\text{wrench}} x_{\text{wrench}} \leq W_{\max}$$

$$x_{\text{Har},\text{NYC}} + x_{\text{Mem},\text{NYC}} + x_{\text{Ash},\text{NYC}} = 1$$

$$x_{\text{Har},\text{LA}} + x_{\text{Mem},\text{LA}} + x_{\text{Ash},\text{LA}} = 1$$

$$x_{\text{Har},\text{Chi}} + x_{\text{Mem},\text{Chi}} + x_{\text{Ash},\text{Chi}} = 1$$

$$x_{\text{Har},\text{Hou}} + x_{\text{Mem},\text{Hou}} + x_{\text{Ash},\text{Hou}} = 1$$

Warehouse Location

$$\begin{aligned} \min \quad & \sum_{w \in W, c \in C} d_{w,c} x_{w,c} \\ \text{s.t.} \quad & \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C \end{aligned}$$

$$x_{w,c} \leq y_w \quad \forall w \in W, c \in C$$

$$\begin{aligned} \sum_{w \in W} y_w &= P \\ 0 \leq x &\leq 1 \quad y \in \{0,1\}^{|W|} \end{aligned}$$

w \ c	NYC	LA	Chicago	Houston
Harlingen	1956	1606	1410	330
Memphis	1096	1792	531	567
Ashland	485	2322	324	1236

Key difference: Scalar vs Indexed Constraint



Knapsack

$$\begin{aligned} \max_x \quad & \sum_{i \in A} v_i x_i \\ \text{s.t.} \quad & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x \in \{0,1\}^{|A|} \end{aligned}$$

Warehouse Location

$$\begin{aligned} \min \quad & \sum_{w \in W, c \in C} d_{w,c} x_{w,c} \\ \text{s.t.} \quad & \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C \\ & x_{w,c} \leq y_w \quad \forall w \in W, c \in C \\ & \sum_{w \in W} y_w = P \\ & 0 \leq x \leq 1 \quad y \in \{0,1\}^{|W|} \end{aligned}$$

- Knapsack problem: All constraints are singular (i.e., scalar)
- Warehouse location problem:
 - Multiple constraints defined over indices
- Pyomo used the concept of *construction rules* to specify indexed constraints



Indexed Constraints: Construction Rules



$$\sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C$$

```
W = ['Harlingen', 'Memphis', 'Ashland']
```

```
C = ['NYC', 'LA', 'Chicago', 'Houston']
```

```
model.x = Var(W, C, bounds=(0,1))
```

```
model.y = Var(W, within=Binary)
```

```
def one_per_cust_rule(m, c):  
    return sum(m.x[w,c] for w in W) == 1
```

```
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)
```

Particular index is passed as argument to the rule

For indexed constraints, you provide a “rule” (function) that returns an expression (or tuple) for each index.

Rule is called once for every entry in C!



Construction Rules: Multiple indices

$$x_{w,c} \leq y_w \quad \forall w \in W, c \in C$$

```

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)
def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)
  
```

Each dimension of the indices is a separate argument to the rule

Rule is called once for every entry w in W crossed with every entry c in C !

Construction Rules: Complex logic

$$x_i = \begin{cases} \cos(y_i), & i \text{ even} \\ \sin(y_i), & i \text{ odd} \end{cases} \quad \forall \{i \in N : i > 0\}$$

```
def complex_rule(model, i):
    if i == 0:
        return Constraint.Skip
    elif i % 2 == 0:
        return model.x[i] == cos(model.y[i])
    return model.x[i] == sin(model.y[i])
model.complex_constraint = Constraint(N, M, rule=complex_rule)
```

Constraint rules can contain complex logic on the indices and other parameters, but NOT on the value of model variables.

More on Construction Rules

- Components are constructed in declaration order
 - The instructions for *how* to construct the object are provided through a function, or *rule*
 - Pyomo calls the rule for each component index
 - *Rules* can be provided to virtually all Pyomo components
- Naming conventions
 - the component name prepended with “_” ($c4 \rightarrow _c4$)
 - the component name with “_rule” appended ($c4 \rightarrow c4_rule$)
 - each rule is called “rule” (Python implicitly overrides each declaration)
- Abstract models (discussed later) construct the model in two passes:
 - Python parses the model declaration
 - creating “empty” Pyomo components in the model
 - Pyomo loads and parses external data

Putting it all together: Warehouse Location



- Determine the set of P warehouses chosen from the set W of candidates to minimize the cost of serving all customers C
- Here $d_{w,c}$ is the cost of serving customer c from warehouse at location w .

$$\min \sum_{w \in W, c \in C} d_{w,c} x_{w,c}$$

$$s.t. \quad \sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C$$

$$x_{w,c} \leq y_w \quad \forall w \in W, c \in C$$

$$\sum_{w \in W} y_w = P$$

$$0 \leq x \leq 1 \quad y \in \{0,1\}^{|W|}$$

W	C	NYC	LA	Chicago	Houston
Harlingen		1956	1606	1410	330
Memphis		1096	1792	531	567
Ashland		485	2322	324	1236

$$P = 2$$

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956,
     . . .
     ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Putting It All Together: Warehouse Location



$$x_{w,c} \quad \forall w \in W, c \in C$$
$$y_w \quad \forall w \in W$$

```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956,
     . . .
     ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

$$\min_{x,y} \sum_{w \in W, c \in C} d_{w,c} x_{w,c}$$

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

$$\sum_{w \in W} x_{w,c} = 1 \quad \forall c \in C$$

Putting It All Together: Warehouse Location



$$x_{w,c} \leq y_w \quad \forall w \in W, c \in C$$

```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956,
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956,
     . . .
     ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

$$\sum_{w \in W} y_w = P$$

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Putting It All Together: Warehouse Location



```
# warehouse_location.py: Warehouse location determination problem
from pyomo.environ import *

model = ConcreteModel(name="(WL)")

W = ['Harlingen', 'Memphis', 'Ashland']
C = ['NYC', 'LA', 'Chicago', 'Houston']
d = {('Harlingen', 'NYC'): 1956, \
      . . .
      ('Ashland', 'Houston'): 1236 }
P = 2

model.x = Var(W, C, bounds=(0,1))
model.y = Var(W, within=Binary)

def obj_rule(m):
    return sum(d[w,c]*m.x[w,c] for w in W for c in C)
model.obj = Objective(rule=obj_rule)

def one_per_cust_rule(m, c):
    return sum(m.x[w,c] for w in W) == 1
model.one_per_cust = Constraint(C, rule=one_per_cust_rule)

def warehouse_active_rule(m, w, c):
    return m.x[w,c] <= m.y[w]
model.warehouse_active = Constraint(W, C, rule=warehouse_active_rule)

def num_warehouses_rule(m):
    return sum(m.y[w] for w in W) <= P
model.num_warehouses = Constraint(rule=num_warehouses_rule)

SolverFactory('glpk').solve(model)

model.pprint()
```

Pyomo Fundamentals: Exercises #2

- 2.1 Knapsack problem with rules:** Rules are important for defining indexed constraints, however, they can also be used for single (i.e. scalar) constraints. Starting with `knapsack.py`, reimplement the model using rules for the objective and the constraints. (A solution can be found in `knapsack_rules.soln.py`.)
- 2.2 Integer formulation of the knapsack problem:** Consider again, the knapsack problem. Assume now that we can acquire multiple items of the same type. In this new formulation, x_i is now an integer variable instead of a binary variable. One way to formulate this problem is as follows:

$$\begin{aligned} & \max_{q,x} \sum_{i \in A} v_i x_i \\ \text{s.t. } & \sum_{i \in A} w_i x_i \leq W_{\max} \\ & x_i = \sum_{j=0}^N j q_{i,j} \quad \forall i \in A \\ & 0 \leq x \leq N \\ & q_{i,j} \in \{0, 1\} \quad \forall i \in A, j \in \{0..N\} \end{aligned}$$

Starting with `knapsack.rules.py`, implement this new formulation and solve. Is the solution surprising? (A solution can be found in `knapsack.integer.soln.py`.)

Decorator notation

- Reduce redundancy in *rule definitions* using *Decorators*

```
@model.Constraint(W, C)
def warehouse_active(m, w, c):
    return m.x[w,c] <= m.y[w]
```

- General notation

```
@<object>. <Component>(indices, ..., keywords=...)
def my_thing(m, index, ...):
    return # ...
```

Is equivalent to

```
def my_thing(m, index, ...):
    return # ...
<object>.my_thing = <Component>(indices, ..., rule=my_thing, keywords=...)
```

Note: this syntax *only* works for defining the *rule* keyword.

Parameters

- Pyomo models can be built using standard python numeric types (e.g., int, float) for all constants/data
 - This is how we have done examples so far.
- Pyomo also supports a parameter component (Param)
 - Keeps data documented on the model
 - Allows for validation of data, default values, and changes in data without the need to rebuild entire model
 - Allows Abstract model definitions (declare model, apply data later)

Provide an (initial) value of 42 for the parameter

- Scalar numeric values

```
model.a_parameter = Param( initialize = 42 )
```

Parameters

- Indexed numeric values

```
model.a_param_vec = Param( IDX,  
                           initialize = data,  
                           default = 0 )
```

Providing “default” allows the initialization data to only specify the “unusual” values

“data” must be a dictionary(*) of index keys to values because all sets are assumed to be *unordered*

(*) – actually, it must define `__getitem__()`, but that only really matters to Python geeks

- Mutable Parameters

```
model.a_parameter = Param( initialize = 42,  
                           mutable = True )
```

Indicates to Pyomo that you may want to change this parameter later.

Generating and Managing Indices: Sets

- Any iterable object can be an index, e.g., lists:
 - `IDX_a = [1,2,5]`
 - `DATA = {1: 10, 2: 21, 5:42};`
`IDX_b = DATA.keys()`
- Sets: objects for managing multidimensional indices
 - `model.IDX = Set(initialize = [1,2,5])`

Note: capitalization matters:
`Set` = Pyomo class
`set` = native Python set

Like indices, Sets can be initialized from any iterable

- `model.IDX = Set([1,2,5])`

Note: This does not mean what you think it does.
This creates a 3-member *indexed set*, where each set is *empty*.

Sequential Indices: *RangeSet*

- Sets of sequential integers are common
 - `model.IDX = Set(initialize=range(5))`
 - `model.IDX = RangeSet(5)`

Note: RangeSet is 1-based.
This gives [1, 2, 3, 4, 5]

Note: Python range is 0-based.
This gives [0, 1, 2, 3, 4]

- You can provide lower and upper bounds to RangeSet
 - `model.IDX = RangeSet(0, 4)`

This gives [0, 1, 2, 3, 4]

Manipulating Sets

- Creating sparse sets

```
model.IDX = Set( initialize=[1,2,5] )
def lower_tri_filter(model, i, j):
    return j <= i
model.LTRI = Set( initialize = model.IDX * model.IDX,
                  filter = lower_tri_filter )
```

The filter should return *True* if the element is in the set; *False* otherwise.

- Sets support efficient higher-dimensional indices

```
model.IDX = Set( initialize=[1,2,5] )
model.IDX2 = model.IDX * model.IDX
```

This creates a *virtual*
2-D “matrix” Set

Sets also support union (&), intersection (|),
difference (-), symmetric difference (^)

Higher-Dimensional Sets and Flattening

- Higher-dimensional sets contain multiple indices per element

```
model.IDX = Set( initialize=[1,2,5] )
model.IDX2 = model.IDX * model.IDX

tuples = list()
for i in model.IDX:
    for j in model.IDX:
        tuples.append( (i,j) )
model.IDXT = Set( initialize=tuples )
```

- $\text{IDX2} == \text{IDXT} == [(1,1), (1,2), (1,5), (2,1), (2,2), (2,5), (5,1), (5,2), (5,5)]$

- Higher-dimensional sets and rules

```
def c5_rule(model, i, j, k):
    return model.a[i] + model.a[j] + model.a[k] <= 1
model.c5 = Constraint( model.IDX2, model.IDX, rule=c5_rule )
```

Each dimension of each index is a separate argument to the rule

Set ordering

- By default, Sets are *unordered* (just like Python sets and dictionaries)

```
model.IDX = Set( initialize=[1,2,5] )  
for i in model.IDX: ← No specific order guaranteed for this loop  
    print(i) (e.g., 1,5,2 ... 5,1,2 ...)  
  
model.x = Var(model.IDX)  
for k in model.x: ← This is also true for variables  
    print(value(model.x[i])) indexed by unordered sets.
```

```
model.IDX0 = Set( initialize=[1,2,5], ordered=True )
```

Use the keyword argument `ordered=True` to guarantee fixed order.

Other Modeling Components

- Pyomo supports “list”-like indexed components (useful for meta-algorithms and addition of cuts)

```
model.a = Var()  
model.b = Var()  
model.c = Var()  
model.limits = ConstraintList()  
  
model.limits.add(30*model.a + 15*model.b + 10*model.c <= 100)  
model.limits.add(10*model.a + 25*model.b + 5*model.c <= 75)  
model.limits.add(6*model.a + 11*model.b + 3*model.c <= 30)
```

“add” adds a single new constraint to the list.
The constraints need not be related.

Expression performance tips

- For reasons that are beyond this tutorial, the following can be VERY slow in Pyomo:

- ```
ans = 0
for i in m.INDEX:
 ans = ans + m.x[i]
```

- Recommended alternatives are

- ```
ans = 0
for i in m.INDEX:
    ans += m.x[i]
```
 - ```
sum(m.x[i] for i in m.INDEX)
```

- *Note that this is likely to change in Pyomo 5.6, where all three forms will be relatively equivalent.*

- To report the construction of individual Pyomo components, call:

- ```
pyomo.util.timing.report_timing()
```

- before building your model.

Pyomo Fundamentals: Exercises #3

3.1 Changing Parameter values: In the tutorial slides, we saw that a parameter could be specified to be `mutable`. This tells Pyomo that the value of the parameter may change in the future, and allows the user to change the parameter value and resolve the problem without the need to rebuild the entire model each time. We will use this functionality to find a better solution to an earlier exercise. Considering again the knapsack problem, we would like to find when the wrench becomes valuable enough to be a part of the optimal solution. Create a Pyomo `Parameter` for the value of the items, make it mutable, and then write a loop that prints the solution for different wrench values. Start with the file `knapsack.mutable.parameter.incomplete.py`. (A solution for this problem can be found in `knapsack.mutable.parameter.soln.py`.)

3.2 Integer cuts: Often, it can be important to find not only the “best” solution, but a number of solutions that are equally optimal, or close to optimal. For discrete optimization problems, this can be done using something known as an integer cut. Consider again the knapsack problem where the choice of which items to select is a discrete variable $x_i \forall i \in A$. Let x_i^* be a particular set of x values we want to remove from the feasible solution space. We define an integer cut using two sets. The first set S_0 contains the indices for those variables whose current solution is 0, and the second set S_1 consists of indices for those variables whose current solution is 1. Given these two sets, an integer cut constraint that would prevent such a solution from appearing again is defined by,

$$\sum_{i \in S_0} x[i] + \sum_{i \in S_1} (1 - x[i]) \geq 1.$$

Starting with `knapsack.rules.py`, write a loop that solves the problem 5 times, adding an integer cut to remove the previous solution, and printing the value of the objective function and the solution at each iteration of the loop. (A solution for this problem can be found in `knapsack.integer.cut.soln.py`)

Pyomo Fundamentals: Exercises #3

3.3 Putting it all together with the lot sizing example: We will now write a complete model from scratch using a well-known multi-period optimization problem for optimal lot-sizing adapted from Hagen et al. (2001) shown below.

$$\min \sum_{t \in T} c_t y_t + h_t^+ I_t^+ + h_t^- I_t^- \quad (1)$$

$$\text{s.t. } I_t = I_{t-1} + X_t - d_t \quad \forall t \in T \quad (2)$$

$$I_t = I_t^+ - I_t^- \quad \forall t \in T \quad (3)$$

$$X_t \leq P y_t \quad \forall t \in T \quad (4)$$

$$X_t, I_t^+, I_t^- \geq 0 \quad \forall t \in T \quad (5)$$

$$y_t \in \{0, 1\} \quad \forall t \in T \quad (6)$$

Our goal is to find the optimal production X_t given known demands d_t , fixed cost c_t associated with active production in a particular time period, an inventory holding cost h_t^+ and a shortage cost h_t^- (cost of keeping a backlog) of orders. The variable y_t (binary) determines if we produce in time t or not, and I_t^+ represents inventory that we are storing across time period t , while I_t^- represents the magnitude of the backlog. Note that equation (4) is a constraint that only allows production in time period t if the indicator variable $y_t=1$.

Write a Pyomo model for this problem and solve it using glpk using the data provided below. You can start with the file `lot_sizing_incomplete.py`. (A solution is provided in `lot_sizing.soln.py`.)

Parameter	Description	Value
c	fixed cost of production	4.6
I_0^+	initial value of positive inventory	5.0
I_0^-	initial value of backlogged orders	0.0
h^+	cost (per unit) of holding inventory	0.7
h^-	shortage cost (per unit)	1.2
P	maximum production amount (big-M value)	5
d	demand	[5, 7, 6.2, 3.1, 1.7]

Other Topics



Solving models: *the pyomo command*



- `pyomo` (`pyomo.exe` on Windows):

- Constructs model and passes it to an (external) solver

```
pyomo solve <model_file> [<data_file> ...] [options]
```

- Installed to:

- `[PYTHONHOME]\Scripts` [Windows; `C:\Python27\Scripts`]
 - `[PYTHONHOME]/bin` [Linux; `/usr/bin`]

- Key options (*many* others; see `--help`)

<code>--help</code>	Get list of all options
<code>--help-solvers</code>	Get the list of all recognized solvers
<code>--solver=<solver_name></code>	Set the solver that Pyomo will invoke
<code>--solver-options="key=value[...]"</code>	Specify options to pass to the solver as a space-separated list of keyword-value pairs
<code>--stream-solver</code>	Display the solver output during the solve
<code>--summary</code>	Display a summary of the optimization result
<code>--report-timing</code>	Report additional timing information, including construction time for each model component

Abstract Modeling



Center for Computing Research

Concrete vs. Abstract Models

- Concrete Models: data first, then model
 - 1-pass construction
 - All data must be present before Python starts processing the model
 - Pyomo will construct each component in order at the time it is declared
 - Straightforward logical process; easy to script.
 - Familiar to modelers with experience with GAMS
- Abstract Models: model first, then data
 - 2-pass construction
 - Pyomo stores the basic model declarations, but does not construct the actual objects
 - Details on how to construct the component hidden in functions, or *rules*
 - e.g., it will declare an indexed variable “x”, but will not expand the indices or populate any of the individual variable values.
 - At “creation time”, data is applied to the abstract declaration to create a concrete instance (components are still constructed in declaration order)
 - Encourages generic modeling and model reuse
 - e.g., model can be used for arbitrary-sized inputs
 - Familiar to modelers with experience with AMPL

Data Sources

- Data can be imported from “.dat” file

- Format similar to AMPL style
 - Explicit data from “param” declarations
 - External data through “load” declarations:

- Excel

```
load ABCD.xls range=ABCD : Z=[A, B, C] Y=D ;
```

- Databases

```
load "DBQ=diet.mdb" using=pyodbc query="SELECT FOOD, cost,  
    f_min, f_max from Food" : [FOOD] cost f_min f_max ;
```

- External data overrides “initialize=” declarations

Abstract p-Median (pmedian.py, 1)

```
from pyomo.environ import *

model = AbstractModel()

model.N = Param( within=PositiveIntegers )
model.P = Param( within=RangeSet( model.N ) )
model.M = Param( within=PositiveIntegers )

model.Locations = RangeSet( model.N )
model.Customers = RangeSet( model.M )

model.d = Param( model.Locations, model.Customers )

model.x = Var( model.Locations, model.Customers, bounds=(0.0, 1.0) )
model.y = Var( model.Locations, within=Binary )
```

Abstract p-Median (pmedian.py, 2)

```
def obj_rule(model):
    return sum( model.d[n,m]*model.x[n,m]
               for n in model.Locations for m in model.Customers )
model.obj = Objective( rule=obj_rule )

def single_x_rule(model, m):
    return sum( model.x[n,m] for n in model.Locations ) == 1.0
model.single_x = Constraint( model.Customers, rule=single_x_rule )

def bound_y_rule(model, n,m):
    return model.x[n,m] - model.y[n] <= 0.0
model.bound_y = Constraint( model.Locations, model.Customers,
                            rule=bound_y_rule )

def num_facilities_rule(model):
    return sum( model.y[n] for n in model.Locations ) == model.P
model.num_facilities = Constraint( rule=num_facilities_rule )
```

Abstract p-Median (pmedian.dat)

```
param N := 3;  
  
param M := 4;  
  
param P := 2;  
  
param d: 1      2      3      4 :=  
        1  1.7    7.2   9.0   8.3  
        2  2.9    6.3   9.8   0.7  
        3  4.5    4.8   4.2   9.3 ;
```

In Class Exercise: Abstract Knapsack



$$\begin{aligned} \max \quad & \sum_{i=1}^N v_i x_i \\ s.t. \quad & \sum_{i=1}^N w_i x_i \leq W_{\max} \\ & x_i \in \{0,1\} \end{aligned}$$

Item	Weight	Value
hammer	5	8
wrench	7	3
screwdriver	4	6
towel	3	11

Max weight: 14

Syntax reminders:

```
AbstractModel()  
Set( [index, ...], [initialize=list/function] )  
Param( [index, ...], [within=domain], [initialize=dict/function] )  
Var( [index, ...], [within=domain], [bounds=(Lower,upper)] )  
Constraint( [index, ...], [expr=expression/rule=function] )  
Objective( sense={maximize/minimize},  
           expr=expression/rule=function )
```



Abstract Knapsack: *Solution*



```
from pyomo.environ import *

model      = AbstractModel()
model.ITEMS = Set()
model.v     = Param( model.ITEMS, within=PositiveReals )
model.w     = Param( model.ITEMS, within=PositiveReals )
model.W_max = Param( within=PositiveReals )
model.x     = Var( model.ITEMS, within=Binary )

def value_rule(model):
    return sum( model.v[i]*model.x[i] for i in model.ITEMS )
model.value = Objective( rule=value_rule, sense=maximize )

def weight_rule(model):
    return sum( model.w[i]*model.x[i] for i in model.ITEMS ) \
        <= model.W_max
model.weight = Constraint( rule=weight_rule )
```

Abstract Knapsack: *Solution Data*

```
set ITEMS := hammer wrench screwdriver towel ;  
  
param: v w :=  
    hammer      8 5  
    wrench      3 7  
    screwdriver 6 4  
    towel       11 3;  
  
param w_max := 14;
```

1 - Pyomo Tutorial

- Overview of Pyomo
- Python tutorial
- Pyomo tutorial
- Non-linear Optimization using Pyomo



5. Nonlinear Problems



Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Nonlinear problems are easy...



Nonlinear problems are easy...



... to write in Pyomo (correct formulation and solution is another story)

Nonlinear: Supported expressions

Operation	Operator	Example
multiplication	*	expr = model.x * model.y
division	/	expr = model.x / model.y
exponentiation	**	expr = (model.x+2.0)**model.y
in-place multiplication ¹	*=	expr *= model.x
in-place division ²	/=	expr /= model.x
in-place exponentiation ³	**=	expr **= model.x

```

model = ConcreteModel()
model.r = Var()
model.h = Var()

def surf_area_obj_rule(m):
    return 2 * math.pi * m.r * (m.r + m.h)
model.surf_area_obj = Objective(rule=surf_area_obj_rule)

def vol_con_rule(m):
    return math.pi * m.h * m.r**2 == 355
model.vol_con = Constraint(rule=vol_con_rule)
  
```

Nonlinear: Supported expressions

Operation	Function	Example
arccosine	acos	expr = acos(model.x)
hyperbolic arccosine	acosh	expr = acosh(model.x)
arcsine	asin	expr = asin(model.x)
hyperbolic arcsine	asinh	expr = asinh(model.x)
arctangent	atan	expr = atan(model.x)
hyperbolic arctangent	atanh	expr = atanh(model.x)
cosine	cos	expr = cos(model.x)
hyperbolic cosine	cosh	expr = cosh(model.x)
exponential	exp	expr = exp(model.x)
natural log	log	expr = log(model.x)
log base 10	log10	expr = log10(model.x)
sine	sin	expr = sin(model.x)
square root	sqrt	expr = sqrt(model.x)
hyperbolic sine	sinh	expr = sinh(model.x)
tangent	tan	expr = tan(model.x)
hyperbolic tangent	tanh	expr = tanh(model.x)

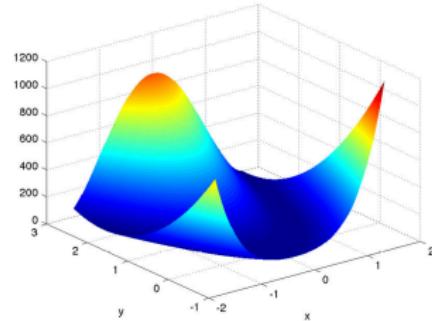
Caution: Always use the intrinsic functions that are part of the Pyomo package.

```
from pyomo.environ import * # imports, e.g., pyomo versions of exp, log, etc.)
from math import *           # overrides the pyomo versions with math versions
```

Example: Rosenbrock function

$$\min_{x,y} f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

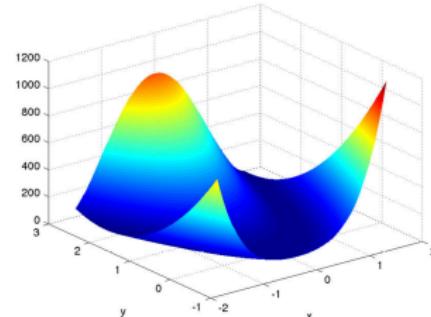
- Minimize the rosenbrock function using Pyomo and IPOPT
- Initialize at $x=1.5, y=1.5$



Example: Rosenbrock function

$$\min_{x,y} f(x,y) = (1-x)^2 + 100(y-x^2)^2$$

- Minimize the rosenbrock function using Pyomo and IPOPT
- Initialize at $x=1.5, y=1.5$



```
# rosenbrock.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var(initialize=1.5)
model.y = Var(initialize=1.5)

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

SolverFactory('ipopt').solve(model, tee=True)
model.pprint()
```

Nonlinear: Exercises

1.1 Alternative Initialization: Effective initialization can be critical for solving nonlinear problems, since they can have several local solutions and numerical difficulties. Solve the Rosenbrock example using different initial values for the x variables. Write a loop that varies the initial value from 2.0 to 6.0, solves the problem, and prints the solution for each iteration of the loop. (A solution for this problem can be found in `rosenbrock_init.soln.py`.)

1.2 Evaluation errors: Consider the following problem with initial values $x=5$, $y=5$.

$$\begin{aligned} \min_{x,y} \quad & f(x,y) = (x - 1.01)^2 + y^2 \\ \text{s.t.} \quad & y = \sqrt{x - 1.0} \end{aligned}$$

- (a) Starting with `evaluation_error.incomplete.py`, formulate this Pyomo model and solve using IPOPT. You should get a list of errors from the solver. Add the IPOPT solver option `solver.options['halt_on_ampl_error']='yes'` to find the problem. (Hint: error output might be ordered strangely, look up in the console output.) What did you discover? How might you fix this? (A solution for this can be found in `evaluation_error.soln.py`.)
- (b) Add bounds $x \geq 1$ to fix this problem. Resolve the problem. Comment on the number of iterations and the quality of solution. (Note: The problem still occurs because $x \geq 1$ is not enforced exactly, and small numerical values still cause the error.) (A solution for this can be found in `evaluation_error.bounds.soln.py`.)
- (c) Think about other solutions for this problem. (e.g., $x \geq 1.001$). (A solution for this can be found in `evaluation_error.bounds2.soln.py`.)

Nonlinear: Exercises

1.3 Alternative Formulations: Consider the following problem with initial values $x=5$, $y=5$.

$$\begin{aligned} \min_{x,y} \quad & f(x,y) = (x - 1.01)^2 + y^2 \\ \text{s.t.} \quad & \frac{x-1}{y} = 1 \end{aligned}$$

Note that the solution to this problem is $x=1.005$ and $y=0.005$. There are several ways that the problem above can be reformulated. Some examples are shown below. Which ones do you expect to be better? Why? Starting with `formulation_incomplete.py` finish the Pyomo model for each of the formulations and solve with IPOPT. Note the number of iterations and quality of solutions. What can you learn about problem formulation from these examples?

(a) (A solution for this can be found in `formulation_1.soln.py`.)

$$\begin{aligned} \min_{x,y} \quad & f(x,y) = (x - 1.01)^2 + y^2 \\ \text{s.t.} \quad & \frac{x-1}{y} = 1 \end{aligned}$$

(b) (A solution for this can be found in `formulation_2.soln.py`.)

$$\begin{aligned} \min_{x,y} \quad & f(x,y) = (x - 1.01)^2 + y^2 \\ \text{s.t.} \quad & \frac{x}{y+1} = 1 \end{aligned}$$

(c) (A solution for this can be found in `formulation_3.soln.py`.)

$$\begin{aligned} \min_{x,y} \quad & f(x,y) = (x - 1.01)^2 + y^2 \\ \text{s.t.} \quad & y = x - 1 \end{aligned}$$

(d) Bounds and initialization can be very helpful when solving nonlinear optimization problems. Starting with `formulation_incomplete.py` resolve the original problem, but add bounds, $y \geq 0$. Note the number of iterations and quality of solution, and compare with what you found in 1.2 (a). (A solution for this can be found in `formulation_4.soln.py`.)

Nonlinear: Exercises

1.4 Reactor design problem (Hart et al., 2017; Bequette, 2003): In this example, we will consider a chemical reactor designed to produce product B from reactant A using a reaction scheme known as the Van de Vusse reaction:



Under appropriate assumptions, F is the volumetric flowrate through the tank. The concentration of component A in the feed is c_{Af} , and the concentrations in the reactor are equivalent to the concentrations of each component flowing out of the reactor, given by c_A , c_B , c_C , and c_D .

If the reactor is too small, we will not produce sufficient quantity of B, and if the reactor is too large, much of B will be further reacted to form the undesired product C. Therefore, our goal is to solve for the reactor volume that maximizes the outlet concentration for product B.

The steady-state mole balances for each of the four components are given by,

$$\begin{aligned} 0 &= \frac{F}{V} c_{Af} - \frac{F}{V} c_A - k_1 c_A - 2k_3 c_A^2 \\ 0 &= -\frac{F}{V} c_B + k_1 c_A - k_2 c_B \\ 0 &= -\frac{F}{V} c_C + k_2 c_B \\ 0 &= -\frac{F}{V} c_D + k_3 c_A^2 \end{aligned}$$

The known parameters for the system are,

$$c_{Af} = 10 \frac{\text{gmol}}{\text{m}^3} \quad k_1 = \frac{5}{6} \text{ min}^{-1} \quad k_2 = \frac{5}{3} \text{ min}^{-1} \quad k_3 = \frac{1}{6000} \frac{\text{m}^3}{\text{mol min}}.$$

Formulate and solve this optimization problem using Pyomo. Since the volumetric flowrate F always appears as the numerator over the reactor volume V , it is common to consider this ratio as a single variable, called the space-velocity SV . (A solution to this problem can be found in `reactor_design.soln.py`)

OTHER MATERIAL

Introduction to IPOPT

$$\begin{array}{ll}
 \min_{\underline{x}} & f(\underline{x}) \quad \xleftarrow{\hspace{1cm}} \text{Objective Function} \\
 \text{s.t.} & c(\underline{x}) = 0 \quad \xleftarrow{\hspace{1cm}} \text{Equality Constraints} \\
 & d^L \leq d(\underline{x}) \leq d^U \quad \xleftarrow{\hspace{1cm}} \text{Inequality Constraints} \\
 & x^L \leq \underline{x} \leq x^U \quad \xleftarrow{\hspace{1cm}} \text{Variable Bounds}
 \end{array}$$

$$\underline{x} \in \mathbb{R}^n$$

$$f(\underline{x}) : \mathbb{R}^n \mapsto \mathbb{R}$$

$$c(\underline{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m$$

$$d(\underline{x}) : \mathbb{R}^n \mapsto \mathbb{R}^p$$

- $f(\underline{x}), c(\underline{x}), d(\underline{x})$
 - general nonlinear functions (non-convex?)
 - Smooth (C^2)
- The \underline{x} variables are continuous
 - $\underline{x}(x-1)=0$ for discrete conditions really doesn't work

Introduction to IPOPT

$$\begin{array}{ll}
 \min_{\underline{x}} & f(\underline{x}) \quad \leftarrow \text{Cost/Profit, Measure of fit} \\
 \text{s.t.} & c(\underline{x}) = 0 \quad \leftarrow \text{Physics of the system} \\
 & d^L \leq d(\underline{x}) \leq d^U \quad \leftarrow \text{Physical, Performance,} \\
 & x^L \leq \underline{x} \leq x^U \quad \leftarrow \text{Safety Constraints}
 \end{array}$$

$$\underline{x} \in \mathbb{R}^n$$

$$f(\underline{x}) : \mathbb{R}^n \mapsto \mathbb{R}$$

$$c(\underline{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m$$

$$d(\underline{x}) : \mathbb{R}^n \mapsto \mathbb{R}^p$$

- $f(\underline{x}), c(\underline{x}), d(\underline{x})$
 - general nonlinear functions (non-convex?)
 - Smooth (C^2)
- The \underline{x} variables are continuous
 - $\underline{x}(x-1)=0$ for discrete conditions really doesn't work

Large Scale Optimization

- Gradient Based Solution Techniques

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & c(x) = 0 \\ & x > 0 \end{array}$$



$$\nabla f(x) + \nabla c(x)^T \cdot \lambda = 0$$

$$c(x) = 0$$

$$x \geq 0$$

$$(x \geq 0, z \geq 0)$$

Newton Step

$$\begin{bmatrix} W_k & \nabla c(x_k) \\ \nabla c(x_k)^T & 0 \end{bmatrix} \begin{pmatrix} \Delta x \\ \Delta \lambda \end{pmatrix} = - \begin{bmatrix} \nabla f(x_k) + \nabla c(x_k)^T \lambda_k \\ c(x_k) \end{bmatrix}$$

$$(W_k = \nabla_{xx}^2 \mathcal{L} = \nabla_{xx}^2 f(x_k) + \nabla_{xx}^2 c(x_k) \lambda)$$

Active-set Strategy

Interior Point Methods

Original NLP

$$\begin{array}{ll} \min_x & f(x) \\ \text{s.t.} & c(x) = 0 \\ & x \geq 0 \end{array}$$

Barrier NLP

$$\begin{array}{ll} \min_x & f(x) - \mu \cdot \sum_i \ln(x_i) \\ \text{s.t.} & c(x) = 0 \end{array}$$

- Initialize $x_0 > 0, \mu_0 > 0, \text{ Set } l \leftarrow 0$
- Solve Barrier NLP for μ_l
- Decrease the barrier parameter $\mu_{l+1} < \mu_l$
- Increase $l \leftarrow l + 1$

as $x \rightarrow 0, \ln(x) \rightarrow \infty$

as $\mu \rightarrow 0, x^*(\mu) \rightarrow x^*$

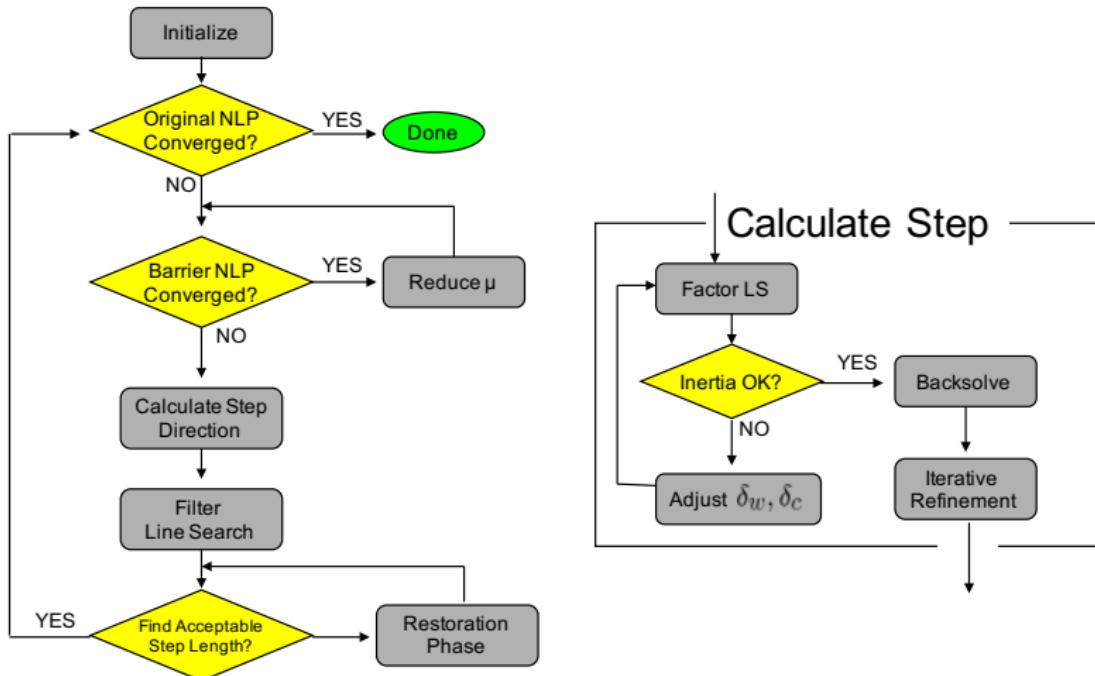
Fiacco & McCormick (1968)

IPOPT: Other Considerations



- Regularization:
 - If certain convexity criteria are not satisfied at a current point, IPOPT may need to regularize. (This can be seen in the output.)
 - We do NOT want to see regularization at the final iteration (solution).
 - Can be an indicator of poor conditioning.
- Globalization:
 - IPOPT uses a filter-based line-search approach
 - Accepts the step if sufficient reduction is seen in objective or constraint violation
- Restoration Phase:
 - Minimize constraint violation
 - Regularized with distance from current point
 - Similar structure to original problem (reuse symbolic factorization)

IPOPT Algorithm Flowsheet



IPOPT Output

```
*****
This program contains Ipopt, a library for large-scale nonlinear optimization.
Ipopt is released as open source code under the Eclipse Public License (EPL).
For more information visit http://projects.coin-or.org/Ipopt
*****
```

```
This is Ipopt version 3.11.7, running with linear solver ma27.
```

```
Number of nonzeros in equality constraint Jacobian...: 2
Number of nonzeros in inequality constraint Jacobian.: 0
Number of nonzeros in Lagrangian Hessian.....: 1

Total number of variables.....: 2
    variables with only lower bounds: 0
    variables with lower and upper bounds: 1
    variables with only upper bounds: 0
Total number of equality constraints.....: 1
Total number of inequality constraints.....: 0
    inequality constraints with only lower bounds: 0
    inequality constraints with lower and upper bounds: 0
    inequality constraints with only upper bounds: 0
```

```
iter      objective      inf_pr      inf_du   lg(mu) ||d|| lg(rg) alpha_du alpha_pr  ls
0  5.0000000e-01  2.50e-01  5.00e-01  -1.0  0.00e+00  -  0.00e+00  0.00e+00  0
```

IPOPT Output

iter	objective	inf_pr	inf_du	lg(mu)	d	lg(rg)	alpha_du	alpha_pr	ls
0	5.0000000e-01	2.50e-01	5.00e-01	-1.0	0.00e+00	-	0.00e+00	0.00e+00	0
1	2.4298076e-01	2.33e-01	7.67e-01	-1.0	5.20e-01	-	7.73e-01	9.52e-01h	1
2	2.6898113e-02	7.23e-05	4.09e-04	-1.7	2.16e-01	-	1.00e+00	1.00e+00h	1
3	1.8655807e-04	1.83e-04	7.86e-05	-3.8	2.68e-02	-	1.00e+00	9.97e-01f	1
4	1.8250072e-06	1.23e-12	2.22e-16	-5.7	1.85e-04	-	1.00e+00	1.00e+00h	1
5	-1.7494097e-08	8.48e-13	0.00e+00	-8.6	1.84e-06	-	1.00e+00	1.00e+00h	1

Number of Iterations....: 5

	(scaled)	(unscaled)
Objective.....	-1.7494096510394117e-08	-1.7494096510394117e-08
Dual infeasibility.....	0.0000000000000000e+00	0.0000000000000000e+00
Constraint violation....	8.4843243541854463e-13	8.4843243541854463e-13
Complementarity.....	2.5050549017950606e-09	2.5050549017950606e-09
Overall NLP error.....	2.5050549017950606e-09	2.5050549017950606e-09

- iter: iterations (codes)
- objective: objective
- Inf_pr: primal infeasibility (constraints satisfied? current constraint violation)
- Inf_du: dual infeasibility (am I optimal?)
- lg(mu): log of the barrier parameter, mu
- ||d||: length of the current stepsize
- lg(rg): log of the regularization parameter
- alpha_du: stepsize for dual variables
- alpha_pr: stepsize for primal variables
- ls: number of line-search steps

Exit Conditions

- Successful Exit
- Successful Exit with regularization at solution
- Infeasible
- Unbounded

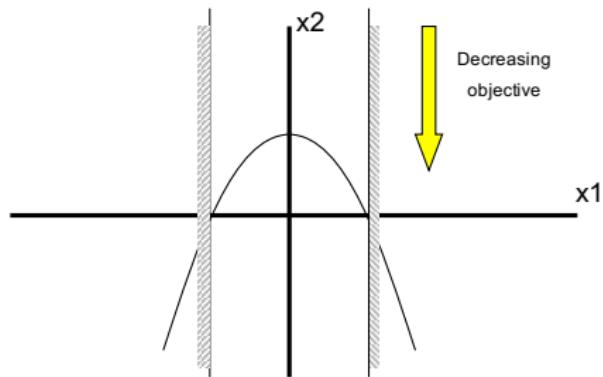
Exit Conditions: Successful

$$\min x_2$$

$$-x_2 = x_1^2 - 1$$

$$-1 \leq x_1 \leq 1$$

Initialize at $(x_1=0.5, x_2=0.5)$



Exit Conditions: Successful

```

iter      objective     inf_pr     inf_du lg(mu) ||d|| lg(rg) alpha_du alpha_pr  ls
0  5.000000e-01 2.50e-01 5.00e-01  -1.0 0.00e+00   - 0.00e+00 0.00e+00  0
1  2.4298076e-01 2.33e-01 7.67e-01  -1.0 5.20e-01   - 7.73e-01 9.52e-01h 1
2  2.6898113e-02 7.23e-05 4.09e-04  -1.7 2.16e-01   - 1.00e+00 1.00e+00h 1
3  1.8655807e-04 1.83e-04 7.86e-05  -3.8 2.68e-02   - 1.00e+00 9.97e-01f 1
4  1.8250072e-06 1.23e-12 2.22e-16  -5.7 1.85e-04   - 1.00e+00 1.00e+00h 1
5 -1.7494097e-08 8.48e-13 0.00e+00  -8.6 1.84e-06   - 1.00e+00 1.00e+00h 1

Number of Iterations....: 5

                           (scaled)                      (unscaled)
Objective.....: -1.7494096510367012e-08  -1.7494096510367012e-08
Dual infeasibility....: 0.000000000000000e+00  0.000000000000000e+00
Constraint violation....: 8.4843243541854463e-13 8.4843243541854463e-13
Complementarity.....: 2.5050549017950606e-09 2.5050549017950606e-09
Overall NLP error.....: 2.5050549017950606e-09 2.5050549017950606e-09

...
EXIT: Optimal Solution Found.

Ipopt 3.11.1: Optimal Solution Found

*** soln
x1 = 1.0
x2 = -1.7494096510367012e-08

```

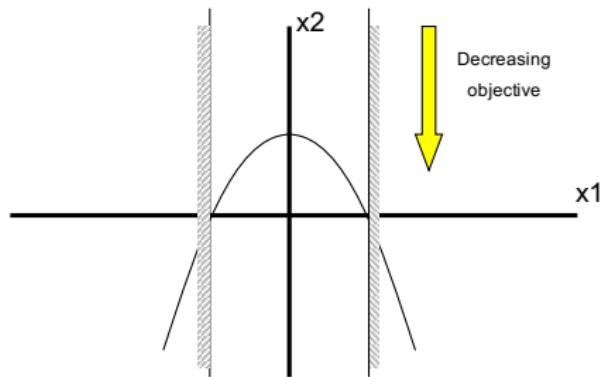
Exit Conditions: Successful w/ Regularization

$$\min x_2$$

$$-x_2 = x_1^2 - 1$$

$$-1 \leq x_1 \leq 1$$

Initialize at $(x_1=0.0, x_2=2.0)$



Exit Conditions: Successful w/ Regularization



```
iter    objective    inf_pr    inf_du lg(mu) ||d|| lg(rg) alpha_du alpha_pr ls
  0  2.000000e+00 1.00e+00 0.00e+00 -1.0 0.00e+00   - 0.00e+00 0.00e+00  0
  1  1.000000e+00 0.00e+00 1.00e-04 -1.7 1.00e+00 -4.0 1.00e+00 1.00e+00h 1
  2  1.000000e+00 0.00e+00 0.00e+00 -3.8 0.00e+00  0.9 1.00e+00 1.00e+00  0
  3  1.000000e+00 0.00e+00 0.00e+00 -5.7 0.00e+00  0.5 1.00e+00 1.00e+00T 0
  4  1.000000e+00 0.00e+00 0.00e+00 -8.6 0.00e+00  0.9 1.00e+00 1.00e+00T 0
```

Number of Iterations....: 4

	(scaled)	(unscaled)
Objective.....	1.000000000000000e+00	1.000000000000000e+00
Dual infeasibility.....	0.000000000000000e+00	0.000000000000000e+00
Constraint violation....	0.000000000000000e+00	0.000000000000000e+00
Complementarity.....	2.5059035596800808e-09	2.5059035596800808e-09
Overall NLP error.....	2.5059035596800808e-09	2.5059035596800808e-09

...

EXIT: Optimal Solution Found.

Iopt 3.11.1: Optimal Solution Found

```
*** soln
x1 = 0.0
x2 = 1.0
```



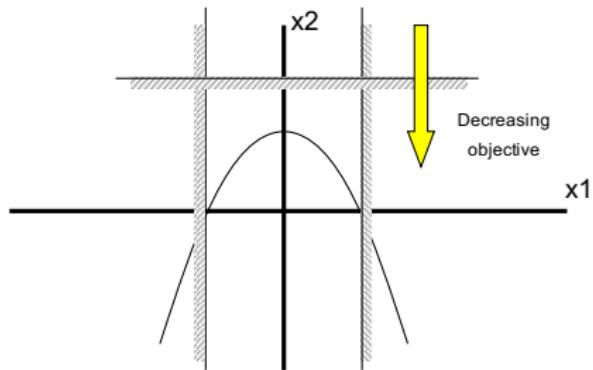
Exit Conditions: Infeasible

$$\min -x_2$$

$$\text{s.t. } -x_2 = x_1^2 - 1$$

$$-1 \leq x_1 \leq 1$$

$$x_2 \geq 2$$



Exit Conditions: Infeasible

```

iter    objective    inf_pr    inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
...
 5  2.000016e+00 1.00e+00 4.74e+03 -1.0 3.47e+01   - 2.79e-02 4.16e-04h 7
 6r 2.000016e+00 1.00e+00 1.00e+03  0.0 0.00e+00   - 0.00e+00 4.44e-07R 3
 7r 2.001000e+00 1.01e+00 1.74e+02  0.0 8.73e-02   - 1.00e+00 1.00e+00f 1
 8r 2.0010010e+00 1.00e+00 1.32e-03  0.0 8.73e-02   - 1.00e+00 1.00e+00f 1
 9r 2.0000080e+00 1.00e+00 5.18e-03 -2.1 6.12e-03   - 9.94e-01 9.99e-01h 1
iter    objective    inf_pr    inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
 10r 2.000000e+00 1.00e+00 3.73e-06 -4.7 7.99e-06   - 1.00e+00 1.00e+00f 1
 11r 2.000000e+00 1.00e+00 1.79e-07 -7.1 2.85e-08   - 1.00e+00 1.00e+00f 1

```

Number of Iterations....: 11

	(scaled)	(unscaled)
Objective.....	1.9999999800009090e+00	1.9999999800009090e+00
Dual infeasibility.....	1.000000002321485e+00	1.000000002321485e+00
Constraint violation.....	9.9999998000090895e-01	9.9999998000090895e-01
Complementarity.....	9.0909091652062654e-10	9.0909091652062654e-10
Overall NLP error.....	9.9999998000090895e-01	1.000000002321485e+00

...

EXIT: Converged to a point of local infeasibility. Problem may be infeasible.

Ipopt 3.11.1: Converged to a locally infeasible point. Problem may be infeasible.

WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt 3.11.1\x3a Converged to a locally infeasible point. Problem may be infeasible.

```

*** soln
x1 = -6.353194883662875e-12
x2 = 2.0

```

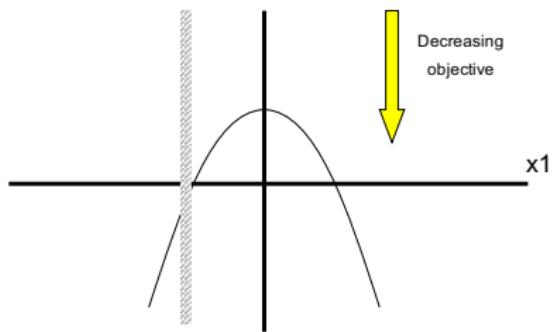
Exit Conditions: Unbounded

$$\min -x_2$$

$$\text{s.t. } -x_2 = x_1^2 - 1$$

$$-1 \leq x_1$$

Initialize at $(x_1=0.5, x_2=0.5)$



Exit Conditions: Unbounded

```

iter    objective    inf_pr    inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
...
45 -2.2420218e+11 1.00e+04 1.00e+00  -1.7 1.25e+19 -19.1 3.55e-08 7.11e-15f 48
46 -2.2420225e+11 1.00e+04 1.00e+00  -1.7 3.75e+19 -19.6 1.20e-08 1.78e-15f 50
47 -2.2420229e+11 1.00e+04 1.00e+00  -1.7 1.25e+19 -19.1 1.00e+00 3.55e-15f 49
48 -3.7503956e+19 1.57e+27 8.36e+09  -1.7 3.75e+19 -19.6 1.18e-08 1.00e+00w 1
49 -1.3750923e+20 3.92e+26 2.09e+09  -1.7 1.00e+20 -20.0 1.00e+00 1.00e+00w 1

```

Number of Iterations....: 49

	(scaled)	(unscaled)
Objective.....	-1.3750923074037683e+20	-1.3750923074037683e+20
Dual infeasibility.....	2.0888873315629249e+09	2.0888873315629249e+09
Constraint violation....	3.9209747283936173e+26	3.9209747283936173e+26
Complementarity.....	3.1115099971882619e+03	3.1115099971882619e+03
Overall NLP error.....	3.9209747283936173e+26	3.9209747283936173e+26

EXIT: Iterates diverging; problem might be unbounded.

Ipopt 3.11.1: Iterates diverging; problem might be unbounded.

WARNING - Loading a SolverResults object with a warning status into model=unknown; message from solver=Ipopt 3.11.1\x3a Iterates diverging; problem might be unbounded.

```
*** soln
x1 = 0.5
x2 = 0.5
```

IPOPT Options

- Solver options can be set through scripts (and the pyomo command line)
- `print_options_documentation yes`
 - Outputs the complete set of IPOPT options (with documentation and their defaults)
- `mu_init`
 - Sets the initial value of the barrier parameter
 - Can be helpful to make this smaller when initial guesses are known to be good
- `bounds_push`
 - By default, IPOPT pushes the bounds a little further out.
 - This can be set to remove this behavior
 - E.g., $\text{sqrt}(x), x \geq 0$
- `linear_solver`
 - Set the linear solver that will be used for the KKT system
 - Significantly better performance with HSL (MA27) over default MUMPS
- `print_user_options`
 - Print options set and whether or not they were used
 - Helpful to detect mismatched options

IPOPT Options

```
# rosenbrock_options.py: A Pyomo model for the Rosenbrock problem
from pyomo.environ import *

model = ConcreteModel()
model.x = Var()
model.y = Var()

def rosenbrock(m):
    return (1.0-m.x)**2 + 100.0*(m.y - m.x**2)**2
model.obj = Objective(rule=rosenbrock, sense=minimize)

solver = SolverFactory('ipopt')
solver.options['mu_init'] = 1e-4
solver.options['print_user_options'] = 'yes'
solver.options['ma27_pivtol'] = 1e-4
solver.solve(model, tee=True)

print()
print('*** Solution *** :')
print('x:', value(model.x))
print('y:', value(model.y))
```

IPOPT Options

```
ma27_pivtol=0.0001
print_user_options=yes
mu_init=0.0001
ma27_pivtol=0.0001
print_user_options=yes
mu_init=0.0001
```

List of user-set options:

Name	Value	used
ma27_pivtol	= 0.0001	no
mu_init	= 0.0001	yes
print_user_options	= yes	yes

```
*****
This program contains Ipopt, a library for large-scale nonlinear optimization.
Ipopt is released as open source code under the Eclipse Public License (EPL).
For more information visit http://projects.coin-or.org/Ipopt
*****
```

NOTE: You are using Ipopt by default with the MUMPS linear solver.
Other linear solvers might be more efficient (see Ipopt documentation).

This is Ipopt version 3.11.1, running with linear solver mumps.



Other Considerations

- Linear Solvers
 - Performance of IPOPT is HIGHLY dependent on the linear solver selected
 - The version of IPOPT installed in this workshop uses a freely distributable linear solver, MUMPS.
 - Performance of IPOPT with the Harwell Subroutine Library suite can be significantly better
 - MA27 (HSL) is **free** for personal and commercial use, but not **distributable**
 - Recommend downloading and installing Coin-HSL Archive (MA27) from HSL
 - Web search: "HSL for IPOPT"
- Variable Initialization
 - Proper initialization of nonlinear problems can be critical for effective solution.
 - Strategies include:
 - Using understood physics or past successful solutions
 - Solving simpler problem(s) first, progressing to more difficult

Other Considerations

- Undefined Evaluations
 - Many mathematical functions have a valid domain, and evaluation outside that domain causes errors
 - Add appropriate bounds to variables to keep them inside valid domain
 - Note that solvers use first and second derivatives. While $\text{sqrt}(x)$ is valid at $x=0$, $1/\text{sqrt}(x)$ is not
- Problem Scaling
 - Scale model to avoid variables, constraints, derivatives with different scales.
- Formulation Matters

Part 3 - Micro-Grids

1 - Introduction

- Stakes of Micro-Grids Integration
- Energy Management of the Micro-Grid

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

- Modelling Components of the Micro-grid
- Building an objective function

4 - Project on Pyomo

5 - Annexes

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes

Stakes of Micro-Grids Integration

1 - Introduction

- Stakes of Micro-Grids Integration
- Energy Management of the Micro-Grid

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

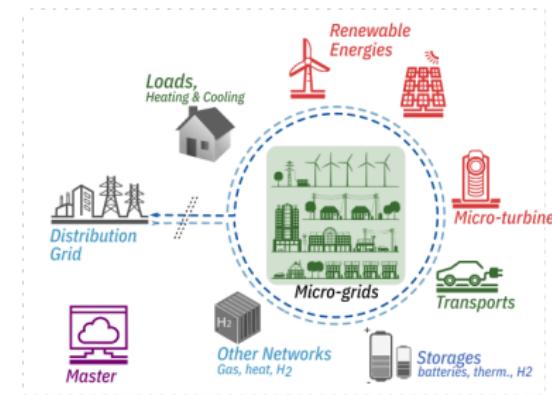
4 - Project on Pyomo

5 - Annexes

Stakes of Micro-Grids Integration

Main Aspects

- **Renewable Energies**
Integration, Techno., Sizing, Topology
- **Energy Quality & Security**
Comfort, Access, Privacy
- **Energy Management**
Economical and Ecological Assessment



Bottlenecks

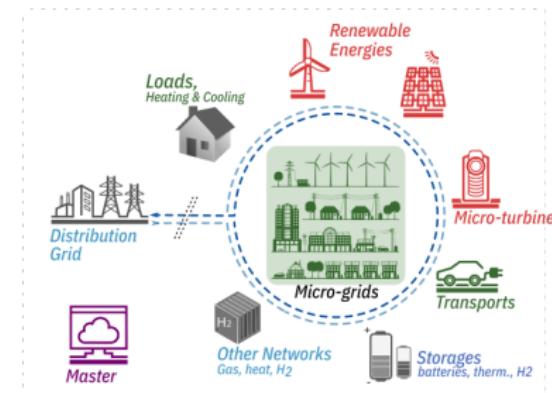
- (Multi-physics)
- Time Scales ($10^{-3} \rightarrow 5.10^8$ s)
- Geographic Scales
- Uncertainties

Micro-Grid : cluster of Renewable Sources, Storage & Charges, connected or isolated from the Main Grid

Stakes of Micro-Grids Integration

Main Aspects

- **Renewable Energies**
Integration, Techno., Sizing, Topology
- **Energy Quality & Security**
Comfort, Access, Privacy
- **Energy Management**
Economical and Ecological Assessment

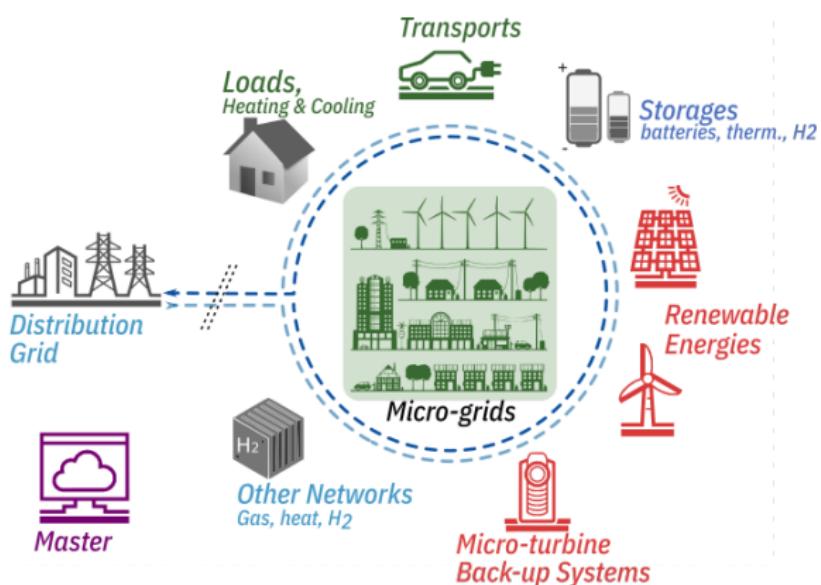


Bottlenecks

- (Multi-physics)
- Time Scales ($10^{-3} \mapsto 5.10^8 s$)
- Geographic Scales
- Uncertainties

Micro-Grid : cluster of Renewable Sources, Storage & Charges, connected or isolated from the Main Grid

Micro-Grids Issues

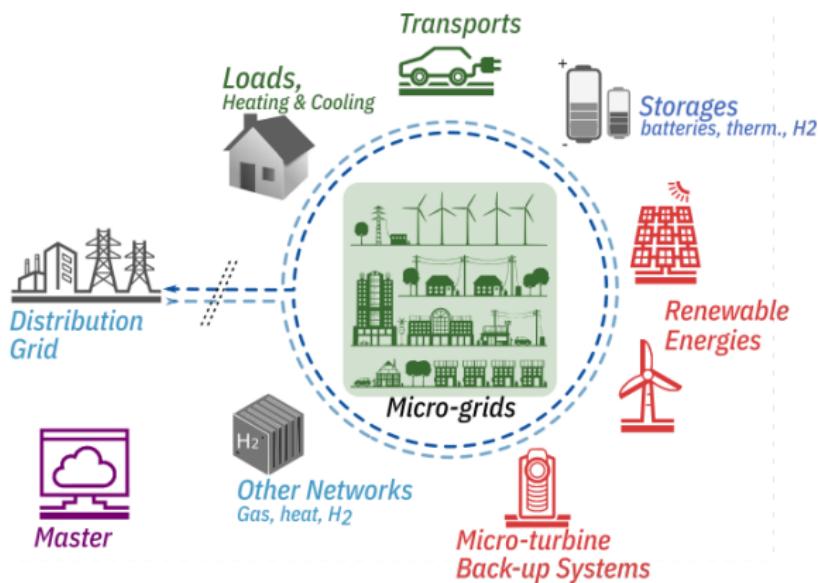


Main Issues

- Decentralised Power Production
- Loads
- Storage
- Distribution grids
- Centralized Master

Stakes of Micro-Grids Integration

Micro-Grids Issues



Main Issues

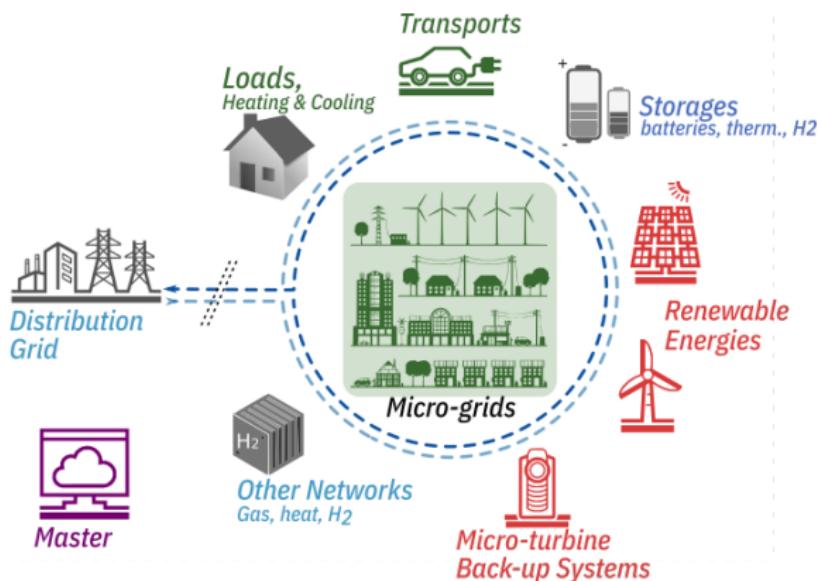
Decentralised Power Production

- ② Intermittence
- ② Incertitudes
- ② Management
- ② Sizing
- ② Technology

- Loads
- Storage
- Distribution grids
- Centralized Master

Stakes of Micro-Grids Integration

Micro-Grids Issues

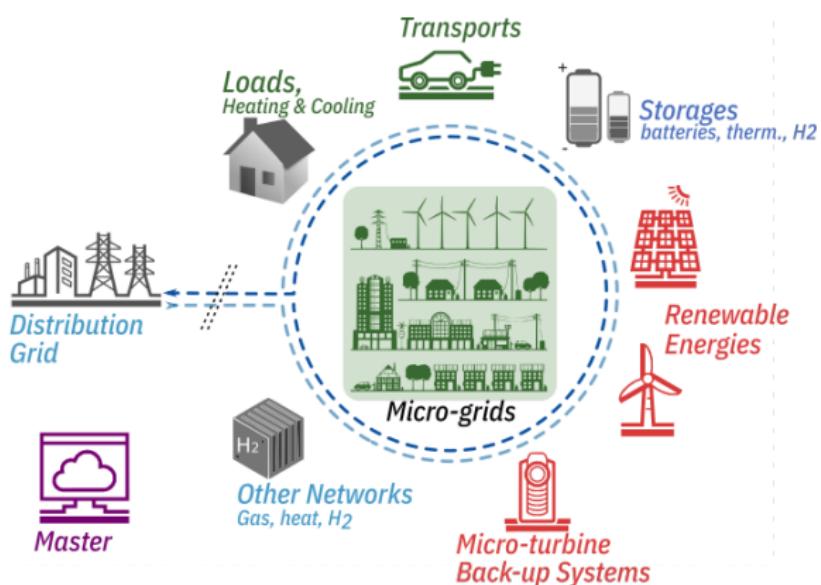


Main Issues

- Decentralised Power Production
- Loads
 - ② Peak Shaving
 - ② Demand Response
 - ② Interface H/S
- Storage
- Distribution grids
- Centralized Master

Stakes of Micro-Grids Integration

Micro-Grids Issues

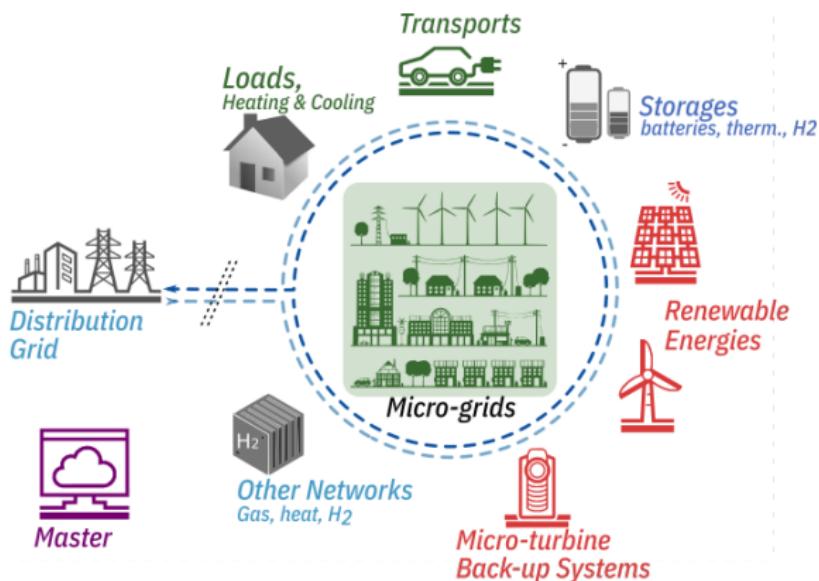


Main Issues

- Decentralised Power Production
- Loads
- Storage
- Management
- Sizing
- Technologies
- Distribution grids
- Centralized Master

Stakes of Micro-Grids Integration

Micro-Grids Issues

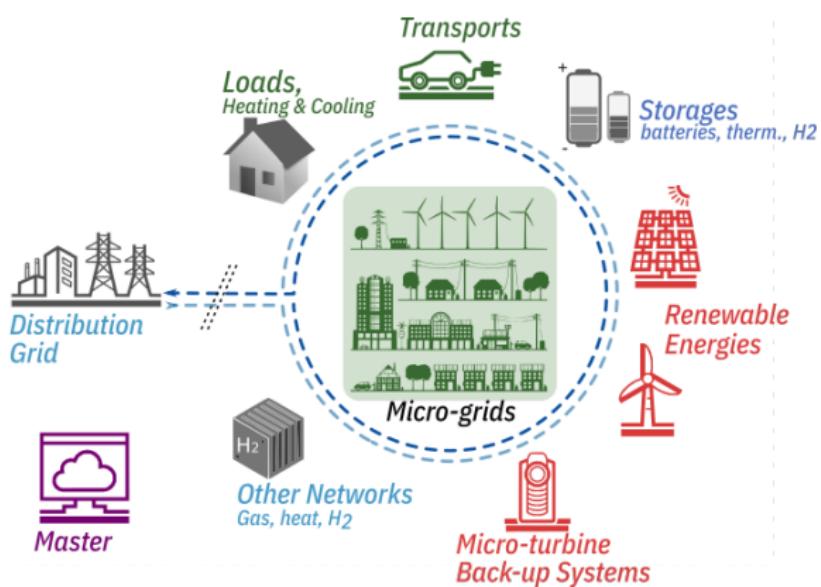


Main Issues

- Decentralised Power Production
- Loads
- Storage
- Distribution grids
- ② Economics
- ② Communications
- ② Contracts and sizing
- Centralized Master

Stakes of Micro-Grids Integration

Micro-Grids Issues



Main Issues

- Decentralised Power Production
- Loads
- Storage
- Distribution grids
- Centralized Master
- ② Interfaces, H/S & S/S
- ② forecast & (Deep) learning
- ② Management

1 - Introduction

- Stakes of Micro-Grids Integration
- Energy Management of the Micro-Grid

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes

Energy Management of the Micro-Grid

Data Management Module

Demand Side Management (DSM)

State Estimation

Energy Management or Dispatch

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes



Introduction to Blocks: Structured Modeling in Pyomo



U.S. DEPARTMENT OF
ENERGY



NASA
National Aeronautics and Space Administration



NCCR
National Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What's the problem with Math Programming?



$$\text{Minimize : } \sum_t \sum_g (c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}) \quad (1)$$

$$\text{S.t. } \theta^{\min} \leq \theta_{\text{net}} \leq \theta^{\max}, \quad \forall n, c, t \quad (2)$$

$$\sum_{\forall k(n,.)} P_{kct} - \sum_{\forall k(.,n)} P_{kct} + \sum_{\forall g(n)} P_{g0t} = d_{nt},$$

$\forall n, c = 0$, transmission contingency states c, t

$$\sum_{\forall k(n,.)} P_{kct} - \sum_{\forall k(.,n)} P_{kct} + \sum_{\forall g(n)} P_{gct} = d_{nt},$$

$\forall n, \text{ generator contingency states } c, t$

$$P_{kc}^{\min} N1_{kc} z_{kt} \leq P_{kct} \leq P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall k, c, t \quad (4)$$

$$B_k(\theta_{\text{net}} - \theta_{\text{mct}}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \geq 0, \quad \forall k, c, t \quad (5a)$$

$$B_k(\theta_{\text{net}} - \theta_{\text{mct}}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \leq 0, \quad \forall k, c, t \quad (5b)$$

$$P_g^{\min} N1_{gc} u_{gt} \leq P_{gct} \leq P_g^{\max} N1_{gc} u_{gt}, \quad \forall g, c, t \quad (6)$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall g, t \quad (7)$$

$$\sum_{q=t-UT_g+1}^t v_{g,q} \leq u_{g,t}, \quad \forall g, t \in \{UT_g, \dots, T\} \quad (8)$$

$$\sum_{q=t-DT_g+1}^t w_{g,q} \leq 1 - u_{g,t}, \quad \forall g, t \in \{DT_g, \dots, T\} \quad (9)$$

$$P_{g0t} - P_{g0,t-1} \leq R_g^+ u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall g, t \quad (10)$$

$$P_{g0,t-1} - P_{g0,t} \leq R_g^- u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall g, t \quad (11)$$

$$P_{gct} - P_{g0,t} \leq R_g^+, \quad \forall g, c, t \quad (12)$$

$$P_{g0,t} N1_{gc} - P_{gct} \leq R_g^-, \quad \forall g, c, t \quad (13)$$

$$0 \leq v_{g,t} \leq 1, \quad \forall g, t \quad (14)$$

$$0 \leq w_{g,t} \leq 1, \quad \forall g, t \quad (15)$$

$$u_{g,t} \in \{0, 1\}, \quad \forall g, t \quad (16)$$

Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010



What's the problem with Math Programming?



$$\begin{aligned} \text{Minimize : } & \sum_t \sum_g (c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt}) \\ \text{S.t. } & \theta^{\min} \leq \theta_{nct} \leq \theta^{\max}, \quad \forall n, c, t \\ & \sum_{k \in k(n,.)} P_{kct} - \sum_{k \in k(.,n)} P_{kct} + \sum_{g \in g(n)} P_{g0t} = d_{nt}, \\ & \forall n, \quad c = 0, \text{ transmission contingency states } c, \\ & \sum_{k \in k(n,.)} P_{kct} - \sum_{k \in k(.,n)} P_{kct} + \sum_{g \in g(n)} P_{gct} = d_{nt}, \\ & \forall n, \text{ generator contingency states } c, t \\ & P_{kc}^{\min} N1_{kc} z_{kt} \leq P_{kct} \leq P_{kc}^{\max} N1_{kc} z_{kt}, \quad \forall k, c, t \\ & B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + (2 - z_{kt} - N1_{kc}) M_k \geq 0, \quad \forall k, c, t \\ & B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - (2 - z_{kt} - N1_{kc}) M_k \leq 0, \quad \forall k, c, t \\ & P_g^{\min} N1_{gc} u_{gt} \leq P_{gct} \leq P_g^{\max} N1_{gc} u_{gt}, \quad \forall g, c, t \\ & v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall g, t \\ & \sum_{q=t-UT_g+1}^t v_{g,q} \leq u_{g,t}, \quad \forall g, t \in \{UT_g, \dots, T\} \\ & \sum_{q=t-DT_g+1}^t w_{g,q} \leq 1 - u_{g,t}, \quad \forall g, t \in \{DT_g, \dots, T\} \\ & P_{g0t} - P_{g0,t-1} \leq R_g^+ u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall g, t \\ & P_{g0,t-1} - P_{g0,t} \leq R_g^- u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall g, t \\ & P_{gct} - P_{g0,t} \leq R_g^+, \quad \forall g, c, t \\ & P_{g0,t} N1_{gc} - P_{gct} \leq R_g^-, \quad \forall g, c, t \\ & 0 \leq v_{g,t} \leq 1, \quad \forall g, t \\ & 0 \leq w_{g,t} \leq 1, \quad \forall g, t \\ & u_{g,t} \in \{0, 1\}, \quad \forall g, t \end{aligned}$$

- The “solution” for Unit Commitment + Transmission Switching + N-1 reliability



Hedman, et al., "Co-Optimization of Generation Unit Commitment and Transmission Switching With N-1 Reliability," *IEEE Trans Power Systems*, 25(2), pp.1052-1063, 2010



The challenge: MP is dense and subtle

Minimize :

$$\sum_t \sum_g (c_g P_{g0t} + c_g^{SU} v_{gt} + c_g^{SD} w_{gt})$$

S.t.

$$\theta^{\min} \leq \theta_{nct} \leq \theta^{\max}, \quad \forall n, c, t$$

$$\sum_{\forall k(n,.)} P_{kct} - \sum_{\forall k(.,n)} P_{kct} + \sum_{\forall g(n)} P_{g0t} = d_{nt},$$

$$\forall n, \quad c = 0, \quad \text{transmission contingency states } c, t$$

$$\sum_{\forall k(n,.)} P_{kct} - \sum_{\forall k(.,n)} P_{kct} + \sum_{\forall g(n)} P_{gct} = d_{nt},$$

$$\forall n, \quad \text{generator contingency states } c, t$$

$$P_{kc}^{\min} |N1_{kc}| z_{kt} \leq P_{kcet} \leq P_{kc}^{\max} |N1_{kc}| z_{kt}, \quad \forall k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} + [(2 - z_{kt}) - |N1_{kc}|] M_k \geq 0, \quad \forall k, c, t$$

$$B_k(\theta_{nct} - \theta_{mct}) - P_{kct} - [(2 - z_{kt}) - |N1_{kc}|] M_k \leq 0, \quad \forall k, c, t$$

$$P_g^{\min} |N1_{gc}| u_{gt} \leq P_{gct} \leq P_g^{\max} |N1_{gc}| u_{gt}, \quad \forall g, c, t$$

$$v_{g,t} - w_{g,t} = u_{g,t} - u_{g,t-1}, \quad \forall g, t$$

$$\sum_{q=t-UT_g+1}^t v_{g,q} \leq u_{g,t}, \quad \forall g, t \in \{UT_g, \dots, T\}$$

$$\sum_{q=t-DT_g+1}^t w_{g,q} \leq 1 - u_{g,t}, \quad \forall g, t \in \{DT_g, \dots, T\}$$

$$P_{g0t} - P_{g0,t-1} \leq R_g^+ u_{g,t-1} + R_g^{SU} v_{g,t}, \quad \forall g, t$$

$$P_{g0,t-1} - P_{g0,t} \leq R_g^- u_{g,t} + R_g^{SD} w_{g,t}, \quad \forall g, t$$

$$P_{gct} - P_{g0,t} \leq R_g^+, \quad \forall g, c, t$$

$$P_{g0,t} |N1_{gc}| - P_{gct} \leq R_g^-, \quad \forall g, c, t$$

$$0 \leq v_{g,t} \leq 1, \quad \forall g, t$$

$$0 \leq w_{g,t} \leq 1, \quad \forall g, t$$

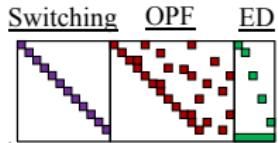
$$u_{g,t} \in \{0, 1\}, \quad \forall g, t$$

To a first approximation:

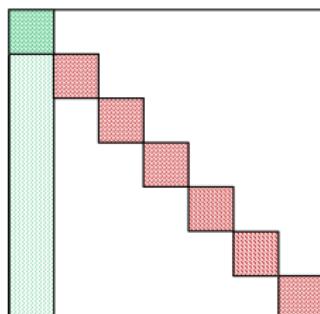
- DCOPF
- Economic dispatch
- Unit commitment
- Transmission switching
- N-1 contingency

(Nonobvious) Inherent structure

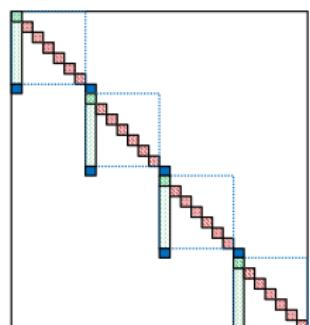
(Sparsity pattern in model constraint matrix)



N-1 Economic Dispatch

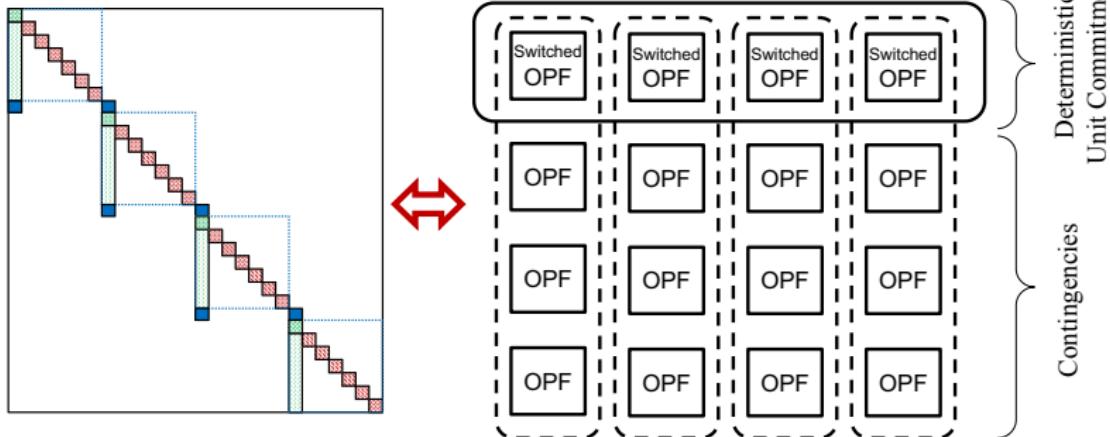


Unit Commitment



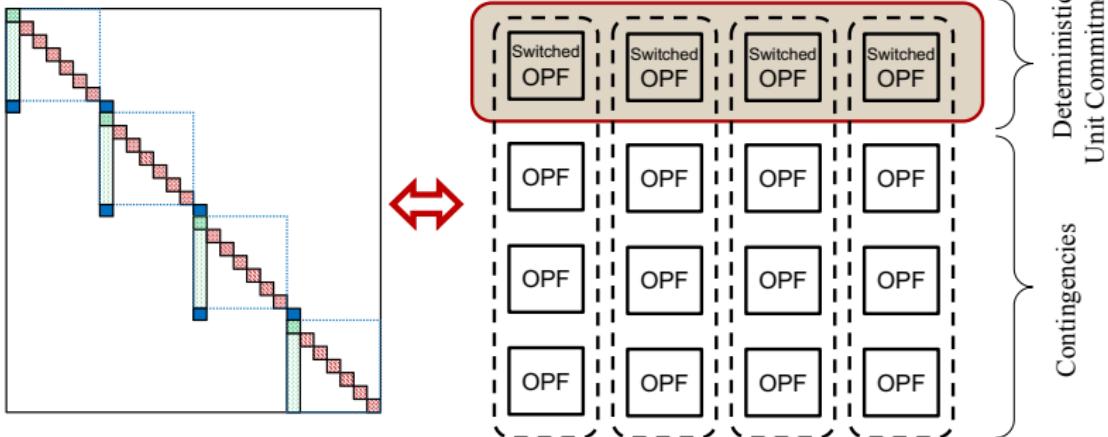
UC + N-1 + Switching block structure

- “2-D” grid of linked optimal power flow models

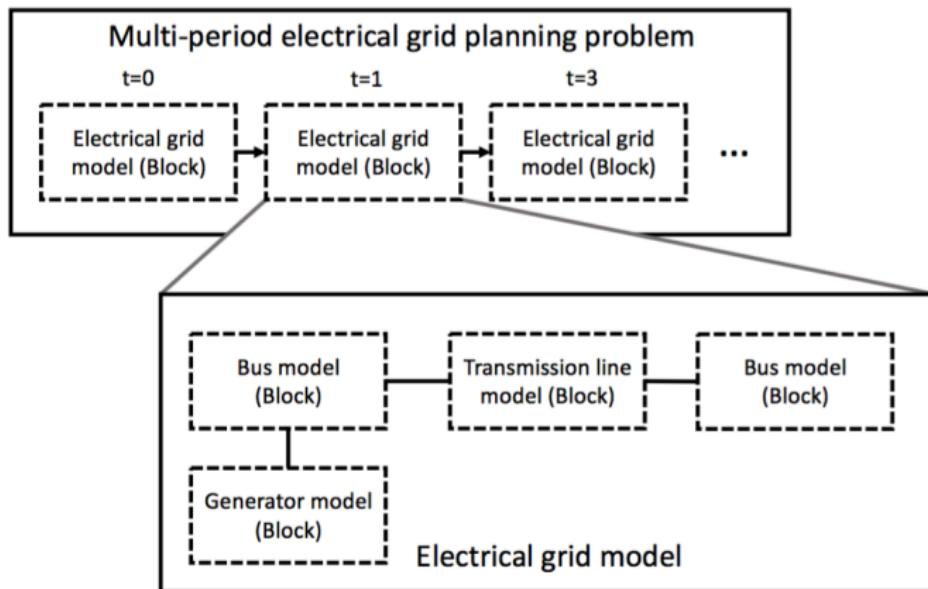


UC + N-1 + Switching block structure

- “2-D” grid of linked optimal power flow models



Object-oriented model construction



Pyomo Blocks

- The Pyomo "Block" allows us to construct hierarchical models
- A Block can be treated in much the same way as a model
 - Modeling components can be added to blocks (e.g., Var, Constraint)
 - Blocks can be added to other Blocks

```
model = ConcreteModel()
model.x = Var()
model.P = Param(initialize=5)
model.S = RangeSet(model.P)
```

Blocks provide *namespacing*

- Each block defines its own namespace

```
print(model.x.local_name)      # x
print(model.x.name)            # x
print(model.b.x.local_name)    # x
print(model.b.x.name)          # b.x
print(model.b.b.x.local_name)  # x
print(model.b.b.x.name)        # b.b.x
```

- Blocks can be created (and “solved”) and later added to a parent model

```
new_b = Block()
new_b.x = Var()
new_b.P = Param(initialize=5)
new_b.I = RangeSet(10)

model = ConcreteModel()
model.b = new_b
model.x = Var(model.b.I)
```

Indexed objects and blocks

- Blocks may be indexed (and defined by rules)

```
model = ConcreteModel()
model.P = Param(initialize=3)
model.T = RangeSet(model.P)

def xyb_rule(b, t):
    b.x = Var()
    b.I = RangeSet(t)
    b.y = Var(b.I)
    b.c = Constraint(expr = b.x == 1.0 - sum(b.y[i] for i \
        in b.I))
model.xyb = Block(model.T, rule=xyb_rule)
```

- Components within blocks may also be indexed

```
def xyb_rule(b, t):
    b.x = Var()
    b.I = RangeSet(t)
    b.y = Var(b.I, initialize=1.0)
    def _b_c_rule(_b):
        return _b.x == 1.0 - sum(_b.y[i] for i in _b.I)
    b.c = Constraint(rule=_b_c_rule)
model.xyb = Block(model.T, rule=xyb_rule)
```

Accessing objects within blocks

- Accessing components within Blocks: looping

```
for t in model.xyb:  
    for i in model.xyb[t].y:  
        print("%s %f" % (model.xyb[t].y[i], value(model.xyb[t].y[i]))
```

- Slicing

- Pyomo supports a special "slice-like" wildcard notation
 - ":" is a wildcard matching a single index
 - "..." is a wildcard that matches 0 or more indices

```
for v in model.xyb[:,].y[:]:  
    print("%s %f" % v, value(v))
```

Advanced slicing: a "weird" model



```
from pyomo.environ import *

m = ConcreteModel()
m.I = Set(initialize=[1,2,3])
@m.Block(m.I, m.I)
def block(b, i, j):
    b_id = 10*(3*i+j)

    @b.Set(dimen=None)
    def IDX(b):
        return [ tuple(
            x*10**y for y in range(j))
            for x in range(i) ]

    @b.Block(b.IDX)
    def subblock(b, *x):
        b.x = Var(initialize=b_id + sum(x))

for v in m.block[...].subblock[...].x:
    print("%-25s= %s" % (v, value(v)))

block[1,1].subblock[0].x = 40
block[1,2].subblock[0,0].x= 50
block[1,3].subblock[0,0,0].x= 60
block[2,1].subblock[0].x = 70
block[2,1].subblock[1].x = 71
block[2,2].subblock[1,10].x= 91
block[2,2].subblock[0,0].x= 80
block[2,3].subblock[0,0,0].x= 90
block[2,3].subblock[1,10,100].x= 201
block[3,1].subblock[2].x = 102
block[3,1].subblock[0].x = 100
block[3,1].subblock[1].x = 101
block[3,2].subblock[1,10].x= 121
block[3,2].subblock[0,0].x= 110
block[3,2].subblock[2,20].x= 132
block[3,3].subblock[2,20,200].x= 342
block[3,3].subblock[0,0,0].x= 120
block[3,3].subblock[1,10,100].x= 231
```



Advanced slicing: explicit indices



```
for v in m.block[2,...].subblock[...].x:  
    print("%-25s= %s" % (v, value(v)))  
  
block[2,1].subblock[0].x = 70  
block[2,1].subblock[1].x = 71  
block[2,2].subblock[1,10].x= 91  
block[2,2].subblock[0,0].x= 80  
block[2,3].subblock[0,0,0].x= 90  
block[2,3].subblock[1,10,100].x= 201  
  
for v in m.block[2,:].subblock[...].x:  
    print("%-25s= %s" % (v, value(v)))  
  
block[2,1].subblock[0].x = 70  
block[2,1].subblock[1].x = 71  
block[2,2].subblock[1,10].x= 91  
block[2,2].subblock[0,0].x= 80  
block[2,3].subblock[0,0,0].x= 90  
block[2,3].subblock[1,10,100].x= 201  
  
block[1,1].subblock[0].x = 40  
block[1,2].subblock[0,0].x= 50  
block[1,3].subblock[0,0,0].x= 60  
block[2,1].subblock[0].x = 70  
block[2,1].subblock[1].x = 71  
block[2,2].subblock[1,10].x= 91  
block[2,2].subblock[0,0].x= 80  
block[2,3].subblock[0,0,0].x= 90  
block[2,3].subblock[1,10,100].x= 201  
block[3,1].subblock[2].x = 102  
block[3,1].subblock[0].x = 100  
block[3,1].subblock[1].x = 101  
block[3,2].subblock[1,10].x= 121  
block[3,2].subblock[0,0].x= 110  
block[3,2].subblock[2,20].x= 132  
block[3,3].subblock[2,20,200].x= 342  
block[3,3].subblock[0,0,0].x= 120  
block[3,3].subblock[1,10,100].x= 231
```



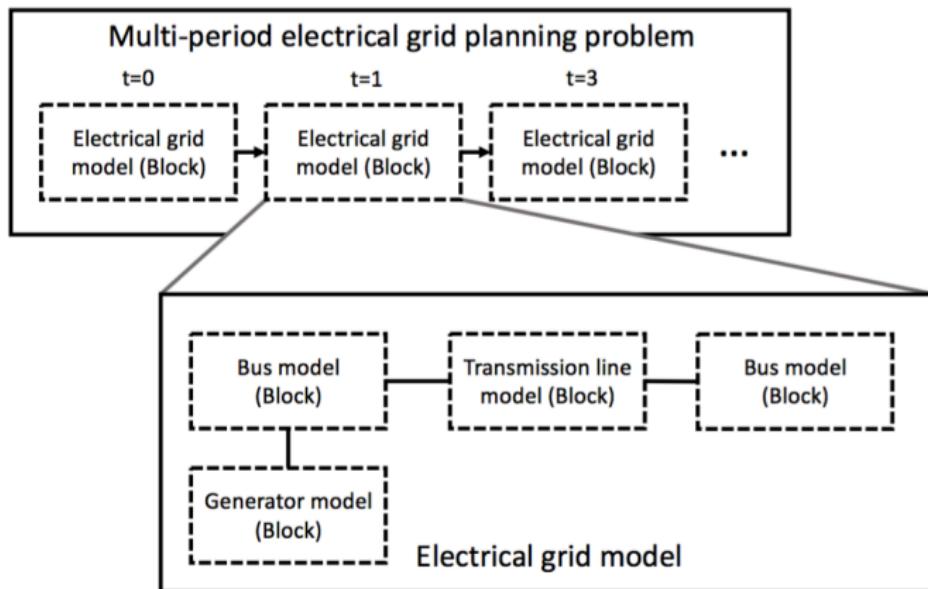
Advanced slicing: jagged sets



```
for v in m.block[...].subblock[1,...].x:  
    print("%-25s= %s" % (v, value(v)))  
  
block[2,1].subblock[1].x = 71  
block[2,2].subblock[1,10].x= 91  
block[2,3].subblock[1,10,100].x= 201  
block[3,1].subblock[1].x = 101  
block[3,2].subblock[1,10].x= 121  
block[3,3].subblock[1,10,100].x= 231  
  
for v in m.block[...].subblock[1,:].x:  
    print("%-25s= %s" % (v, value(v)))  
  
block[2,2].subblock[1,10].x= 91  
block[3,2].subblock[1,10].x= 121  
  
for v in m.block[...].subblock[...,:,1].x:  
    print("%-25s= %s" % (v, value(v)))  
  
block[2,2].subblock[1].x= 71  
block[3,2].subblock[1].x= 101  
  
block[1,1].subblock[0].x = 40  
block[1,2].subblock[0,0].x= 50  
block[1,3].subblock[0,0,0].x= 60  
block[2,1].subblock[0].x = 70  
block[2,1].subblock[1].x = 71  
block[2,2].subblock[1,10].x= 91  
block[2,2].subblock[0,0].x= 80  
block[2,3].subblock[0,0,0].x= 90  
block[2,3].subblock[1,10,100].x= 201  
block[3,1].subblock[2].x = 102  
block[3,1].subblock[0].x = 100  
block[3,1].subblock[1].x = 101  
block[3,2].subblock[1,10].x= 121  
block[3,2].subblock[0,0].x= 110  
block[3,2].subblock[2,20].x= 132  
block[3,3].subblock[2,20,200].x= 342  
block[3,3].subblock[0,0,0].x= 120  
block[3,3].subblock[1,10,100].x= 231
```



Back to object-oriented modeling



(Complete!) Solution

```
import pyomo.environ as pe
import dcopf_decl as dcopf
from uc_example_data import data

model = pe.ConcreteModel()
model.T = pe.Set(initialize=range(2), ordered=True)
model.G = pe.Set(initialize=sorted(data['gens'].keys()), ordered=True)

# create blocks of the dcopf model for each time period
def period_rule(b, t):
    return dcopf.create_dcopf_model(data[t])
model.period = pe.Block(model.T, rule=period_rule)

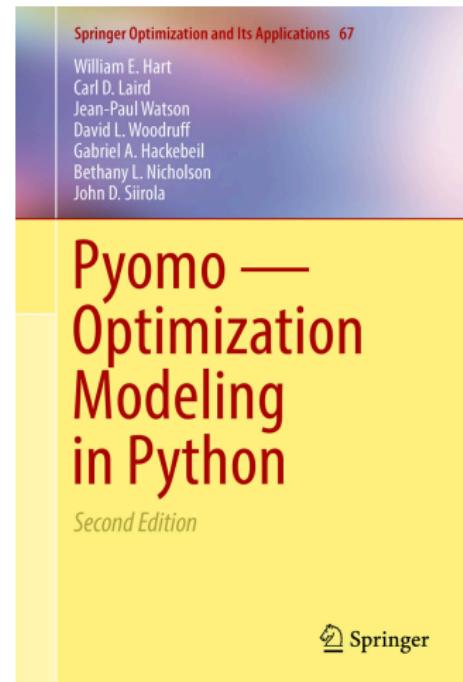
# create the ramping constraints between time periods
def ramp_con_rule(m, t, g):
    if t == m.T.first():
        return pe.Constraint.Skip
    return (-15.0, m.period[t-1].pg[g] - m.period[t].pg[g], 15.0)
model.ramp_con = pe.Constraint(model.T, model.G, rule=ramp_con_rule)

# create the new objective (sum over time)
model.period[:].objective.deactivate()
model.obj = pe.Objective(expr=sum(model.period[:].objective.expr))

# solve the new multi-period model
solver = pe.SolverFactory('ipopt')
solver.solve(model, tee=True)

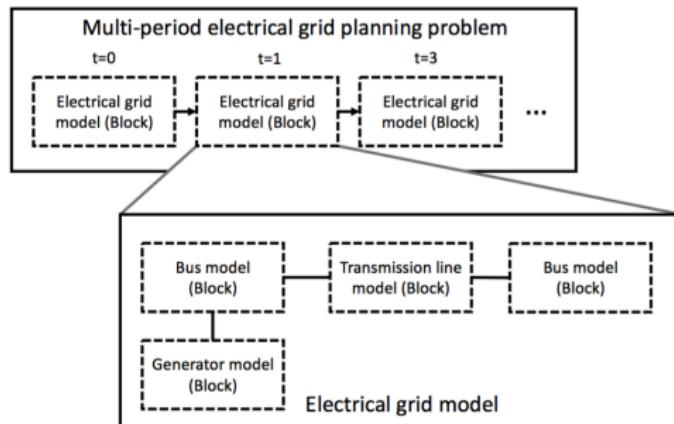
# print the solution
model.period[:].pg.display()
```

Other references



Blocks Summary

- Useful to create object-oriented models
 - (e.g, engineering equipment, physical concepts)
- Useful for other hierarchical structure (temporal, scenario)
- Basis for numerous other extensions (*more in later units*)



1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes

Modelling Components of the Micro-grid

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

- Modelling Components of the Micro-grid
- Building an objective function

4 - Project on Pyomo

5 - Annexes

Modelling Components of the Micro-grid

Modelling Loads

- Critical

$$p_c(t) = p_c^*(t)$$

The consumed critical power is equal to the predicted critical power.

Examples : Servers, Specific machines in hospitals, elevators, ventilation, etc.

Modelling Components of the Micro-grid

Modelling Loads

- Critical

$$p_c(t) = p_c^*(t)$$

- Programmables

- Power Profile

$$u_p \in \{0; 1\} \quad \sum_t u(t) = 1 \quad p_{pp}(t) = \sum_i p_{pp}^*(i).u_p(t - i)$$

The consumed power is following an estimated power profile p_{pp}^* that can be scheduled. The profile is triggered when $u_p(t) = 1$. Example : Washing Machine, Charge of Electric Vehicle.

Modelling Loads

- Critical

$$p_c(t) = p_c^*(t)$$

- Programmables

- Power Profile

$$u_p \in \{0; 1\} \quad \sum_t u(t) = 1 \quad p_{pp}(t) = \sum_i p_{pp}^*(i) \cdot u_p(t - i)$$

- Energy

$$\int_{t_1}^{t_2} p_{pe}(t) \cdot dt = E_{ep}^*$$

The amount of consumed energy within the set $t \in [t_1, t_2]$ is equal to the estimated energy E_{pp}^ . Example : Charge of Electric Vehicle.*

Modelling Components of the Micro-grid

Modelling Loads

- Critical

$$p_c(t) = p_c^*(t)$$

- Programmables

- Power Profile

$$u_p \in \{0; 1\} \quad \sum_t u(t) = 1 \quad p_{pp}(t) = \sum_i p_{pp}^*(i) \cdot u_p(t-i)$$

- Energy

$$\int_{t_1}^{t_2} p_{pe}(t) \cdot dt = E_{ep}^*$$

- Curtailable

$$p_d(t) - u_d(t) \cdot p_d^*(t) = 0 \quad u_d \in \{0; 1\}$$

The estimated power p_d^ may be curtailed (turned off). Example : non-critical loads, lights, cooling, computers.*

Modelling productions

- Photovoltaic panels (or windmills)

$$p_{pv}(t) = (\text{or } \leq) p_{pv}^*(t)$$

The PV production is equal to the estimated power p_{pv}^ on a given horizon.*

- Specific sources¹ : Fuel Cells, Micro-turbines, etc.

* denotes a prediction

Modelling storage systems

Constraints

$$e(0) = e_0 \in \mathbb{R}^+, \quad e(H) = e_f \in \mathbb{R}^+$$

$$e_{min}(t) \leq e(t) \leq e_{max}(t)$$

$$\frac{de(t)}{dt} = \left(p_c(t) \cdot \eta_c - \frac{p_d(t)}{\eta_d} \right)$$

$$p_c(t) - u(t) \cdot p_{cmax} \leq 0$$

$$p_d(t) + u(t) \cdot p_{dmax} \leq p_{dmax}$$

Contributions to the objective function: *May include maintenance, replacement, recycling, investment or environmental costs.*

Modelling storage systems

Reserve Constraint

$$\int_t^{t+H} p_c^*(t) \geq e(t) - e_{min}, H \in \mathbb{R}^+$$

* denotes a prediction

Modelling storage systems

Reserve Constraint

$$\int_t^{t+H} p_c^*(t) \geq e(t) - e_{min}, H \in \mathbb{R}^+$$

In case of grid breakdown, the critical power will be entirely supplied by the storage, over a horizon H (usually some hours).

* denotes a prediction

Main Grid Connection

Contribution to the cost

$$J_{\mathbb{E}}^g = \int_t^{t+H} (c_g^+(t) \cdot p_g^+(t) - c_g^-(t) \cdot p_g^-(t)) \cdot dt \quad \text{where } c_g \text{ is in } \mathbb{E}/(kW.h)$$

$$J_{CO_2}^g = \int_t^{t+H} (m_{CO_2}^g(t) \cdot p_g^+(t) - m_{CO_2}^{\mu g}(t) \cdot p_g^-(t)) \cdot dt \quad \text{where } m_{CO_2} \text{ is in } kgCO_2/(kW.h)$$

$+$: buying from the grid, $-$: selling to the grid

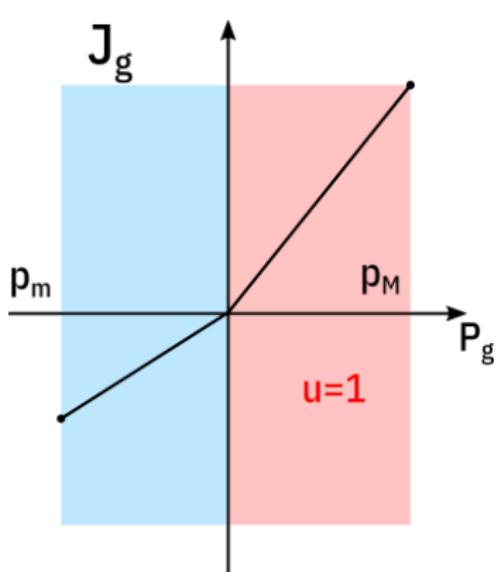
$$c_g^+(t), c_g^-(t) \in \mathbb{R}, \quad p_g^+(t) \in [0 ; p_M] \quad p_g^-(t) \in [0 ; p_m]$$

 Q. 5  Q. 6

Modelling Components of the Micro-grid

Main Grid Connection

Convex case



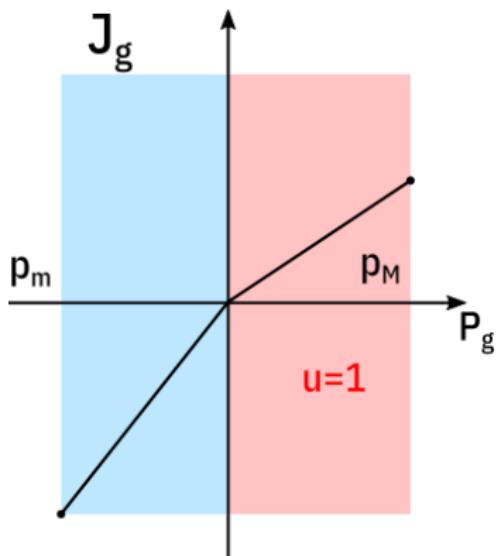
$+$: buying from the grid, $-$: selling to the grid

- J_g is convex with respect to $p_g^+(t)$ and $p_g^-(t)$ if $c_g^+(t) \geq c_g^-(t), \forall t$.
- No binary variable needed

Modelling Components of the Micro-grid

Main Grid Connection

Non-convex case



- J_g is non-convex, i.e.
 $\exists t, c_g^+(t) < c_g^-(t)$
- Introduce a binary variable u

$$\begin{aligned} &\text{if } c_g^+(t) < c_g^-(t) \\ &u(t) \in \{0 ; 1\} \\ &p_g^+(t) \leq pM.u(t) \\ &p_g^-(t) \leq pm.(1 - u(t)) \end{aligned}$$

+ : buying from the grid, - : selling to the grid

Building an objective function

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

- Modelling Components of the Micro-grid
- Building an objective function

4 - Project on Pyomo

5 - Annexes

Building an objective function

Unique Objective function

Ecological or Economical

$$J = \sum_b J_b = J_{grid} + J_{stock} + J_{prod} + J_{load} + \dots$$

This can include , for each component of the μ -grid :

- investment
- maintenance
- replacement
- use
- recycling
- ageing, etc.

Building an objective function

multi-Objective

Ecological, Economical, Comfort, Compactness, etc.

Method #1 :
Weighted sum

Algorithm

- Solve

$$\min_x \alpha J_{CO2}(x) + (1 - \alpha) f_{\in}(x)$$

with $\alpha \in [0, 1]$

- Change α

Method #2 :
 ϵ -constraint

Algorithm

- Solve

$$\min_x f_{CO2}(x) \text{ s.t. } f_{\in}(x) < \epsilon$$

with $\epsilon \in \mathbb{R}$

- Change ϵ

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes

1 - Introduction

2 - Structured Modelling in Pyomo

3 - Optimal Energy Management of the Micro-Grid

4 - Project on Pyomo

5 - Annexes

List of Frames I

- 2 Preliminaries
- 3 Academic Background
- 4 Form of the lecture
- 5 Some prerequisites
- 6 Objectives of the lecture
- 7 Content
- 8 Paris-Saclay University
- 10 Introductory example
- 11 Definition
- 12 Geometric representation (2D)
- 13 What is the use of optimization ?
- 14 What is the use of optimization ?
- 15 What optimization problems are most commonly encountered ?
- 16 What is the use of optimization ?
- 17 Objectives of this part
- 18 Importance of Convexity
- 19 How to detect convexity ?
- 20 Finite Dimensional Optimisation
- 21 Discontinuities in Electric Engineering
- 22 Infinite Dimensional Optimisation
- 23 Linear Programming
- 25 Integer Programming
- 26 Mixed Integer Linear Programming
- 27 Mixed Integer Linear Programming
- 28 Non-Linear Programming

List of Frames II

- 29 Non-Linear Programming
- 30 Take home results and properties
- 31 Quadratic Programming
- 32 Synthesise
- 33 Heuristic and Meta-heuristic
- 34 Multi-Objective
- 35 Pareto-optimal Solution Set
- 36 NSGA-II
- 37 NSGA-II
- 38 Reformulation with 1 objective
- 39 Computer based Modelling
- 40 Compromising accuracy and simplicity
- 41 Modelling Tricks
- 42 Modelling Tricks
- 43 Modelling Tricks
- 44 Modelling Tricks
- 45 Modelling Tricks
- 46 Other Considerations
- 47 Tools
- 48 AMPL
- 49 Matlab
- 51 Python tutorial
- 52 Jupyter notebook
- 55 Micro-Grids Issues

List of Frames III

- 57 Data Management Module
- 58 Demand Side Management (DSM)
- 59 State Estimation
- 60 Energy Management or Dispatch
- 61 Modelling Loads
- 62 Modelling productions
- 63 Modelling storage systems
- 64 Modelling storage systems
- 65 Main Grid Connection
- 66 Main Grid Connection
- 67 Main Grid Connection
- 68 Unique Objective function
- 69 multi-Objective
- 70 List of Frames