Build your own gas turbine performance model in Python

Tutorial of Basics

ASME IGTI TE 2025 **Aircraft Engine Committee**

Dr. Wilfried Visser

Oscar Kogenhop MSc

TU Delft
Delft University of Technology

EPCOR
AIRFRANCE INDUSTRIES KLM

Overview

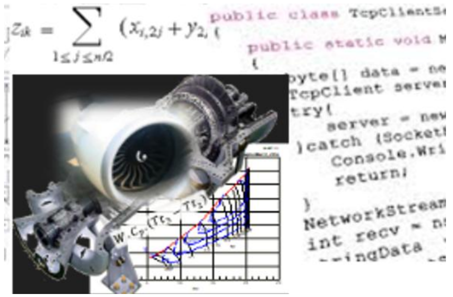
- Introduction
- System performance modelling
  - Propulsion and power systems
- Python software implementation
  - Example case: jet engine performance model
  - Demonstration
- Concluding remarks

For whom is this tutorial?

Introduction

- Engineers, scientists, researchers, students
  - With interest in system simulation of propulsion and power systems

  - Wanting to learn how system models work
  - Needing models with new functionalities
    - Coupling with other models



  - Having basic coding skills

Key focus areas

Introduction

- System modelling *today: gas turbines*
- Object orientation
- Python programming language
  - Open-source libraries
    - Aero-thermal
    - Numerical methods
    - Data handling
    - Visualization

```python
class CodeExample():
    def printStatement(self):
        print('Hello World!')

def main():
    classEx = CodeExample()
    classEx.printStatement()
if __name__ == "__main__":
    main()
```

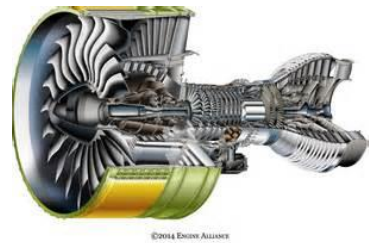
System performance model

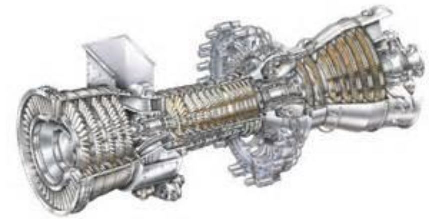
- System of interacting components
  - Relationships among components
  - Component models (usually 0-D, 1-D …)
- Equations
  - conservation of mass, energy, momentum, other…
  - mostly non-linear algebraic/differential
- Component operating points
  - Defined by state variables
- Solution of set of equations
  - State variable vector representing valid system operating point
  - Steady-state or transient

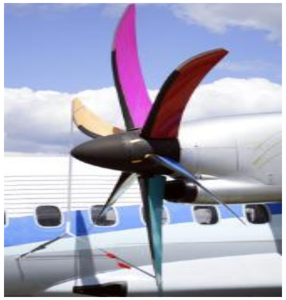
Propulsion and power systems

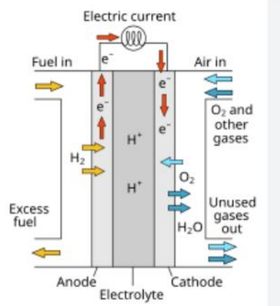
System performance modeling

- Gas turbines
  - Aero engines: jet engines, turboprop, turboshaft
  - Land based engines: turboshaft
- Propellers and rotors
- Hybrid elements
  - Electric motors
  - Generators
  - Fuel cells
  - Batteries
- Alternative/sustainable fuels





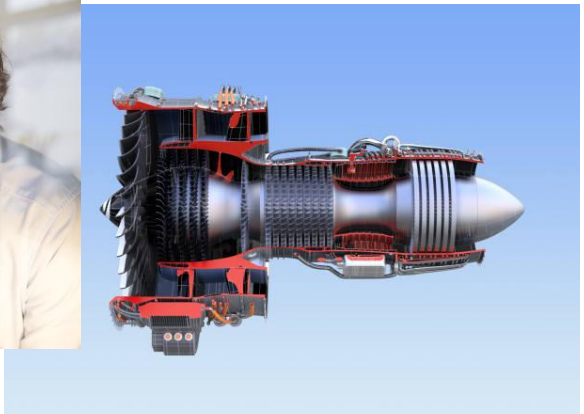




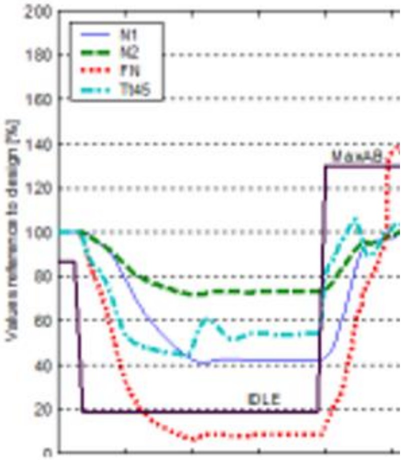
Propulsion and power system performance prediction

System performance modeling

- Design studies



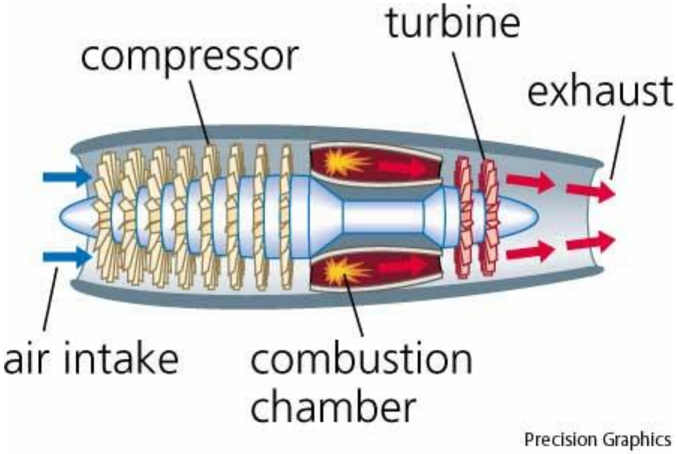
- Existing designs
  - off-design performance



Gas turbine system performance models

System performance modeling

- Component models
  - intake, fans/compressors, combustors, turbines, mixers, exhausts,
  - Component maps



- State variables
  - Rotor speeds, mass flows, pressure ratios, …
- Equations
  - Mass flow in = mass flow out (at component intersections, engine stations)
  - Power balance : PWcompressor = PWturbine
- Input
  - Fuel flow, TIT, inlet P and T…
- Output
  - Thrust FN, Power, Fuel flow, Efficiency, P, T at engine stations…

Design point simulation

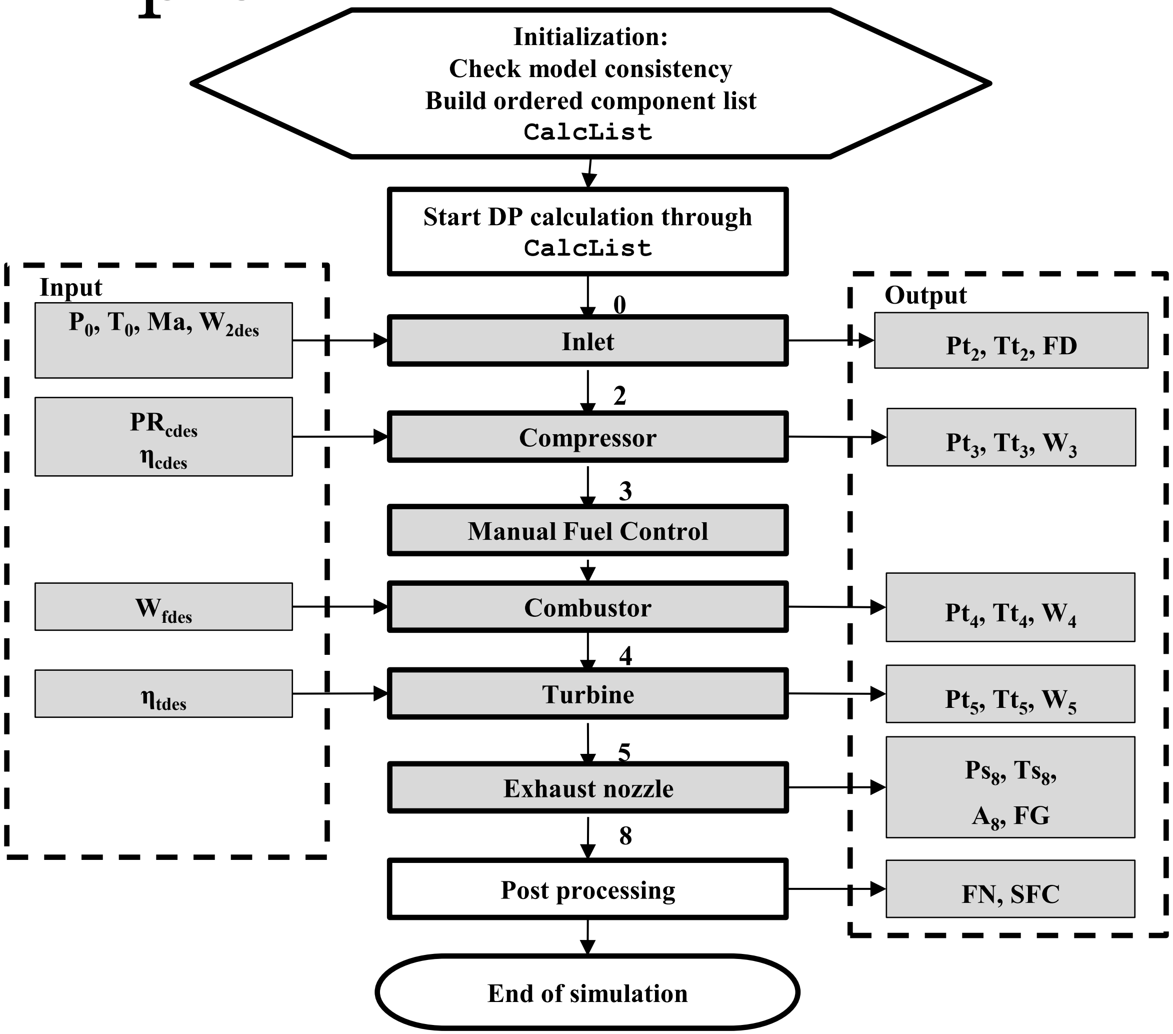
System performance modeling

- Design studies
- Effects of cycle design parameters
  - e.g. PR, BPR, W, efficiencies
- Varying dimensions
- Simple
  - straight forward inlet to exhaust calculation
  - conservation of mass, energy, momentum
- More complex
  - solving equations
  - Constraints
  - AI methods for optimization

Design point calculation example

Design point simulation

- Single spool Turbojet



Off-design ('OD') simulation

System performance modeling

- Analyze effects of operating conditions on performance
- Fixed geometry and dimensions

**except for variable geometry features like VSVs, IGVs, Bleed valves*

- Requires initially calculated Design point
  - 'cycle reference point' CRP

- Predict how the operating point changes (from CRP) with operating conditions deviating from the design conditions
  - Steady-state equilibrium
  - Transient (change in time, e.g. accel/decel of rpm's, heat transfer)

Equations and numerical methods

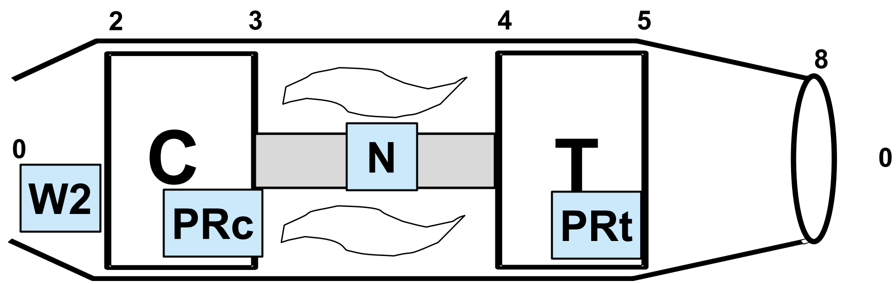
Off-design simulation

- Solve equations for a gas turbine engine steady-state or transient OD operating point
- State variables
  - representing the operating point
  - minimal number of independent (or 'tearing') variables
  - other parameters can be directly/explicitly calculated from the state variable values
- Find the OD operating point = find state variable values
  - Satisfying all conservation equations
  - Residuals ('errors') must all be 0
- Single solution
  - n 'error equations' with n unknown 'state variables'
  - Calculate all dependent other parameter values

Turbojet example - states

Off-design simulation

- Required: 4 states S1..4
  - S1 = W2
  - S2 = PRc*
  - S3 = N
  - S4 = PRt



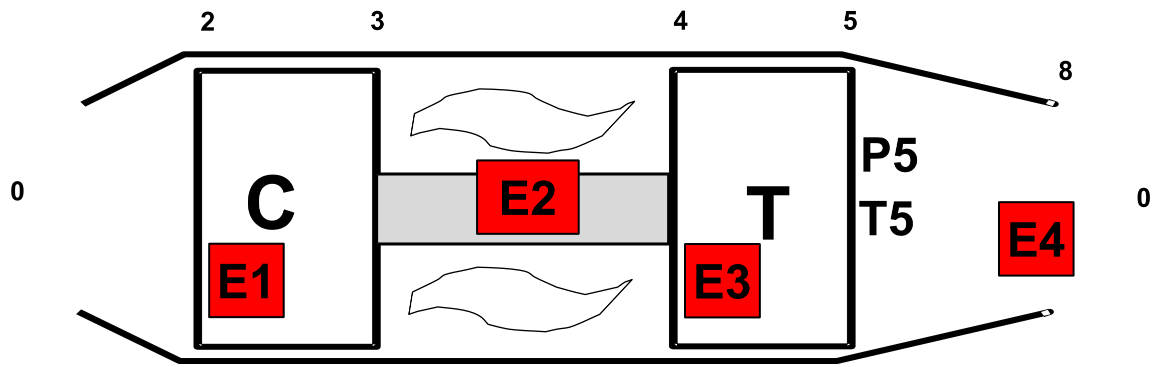
- From the 4 states all other parameters can be calculated

**Instead of PR, usually Beta is used*

Turbojet example - equations

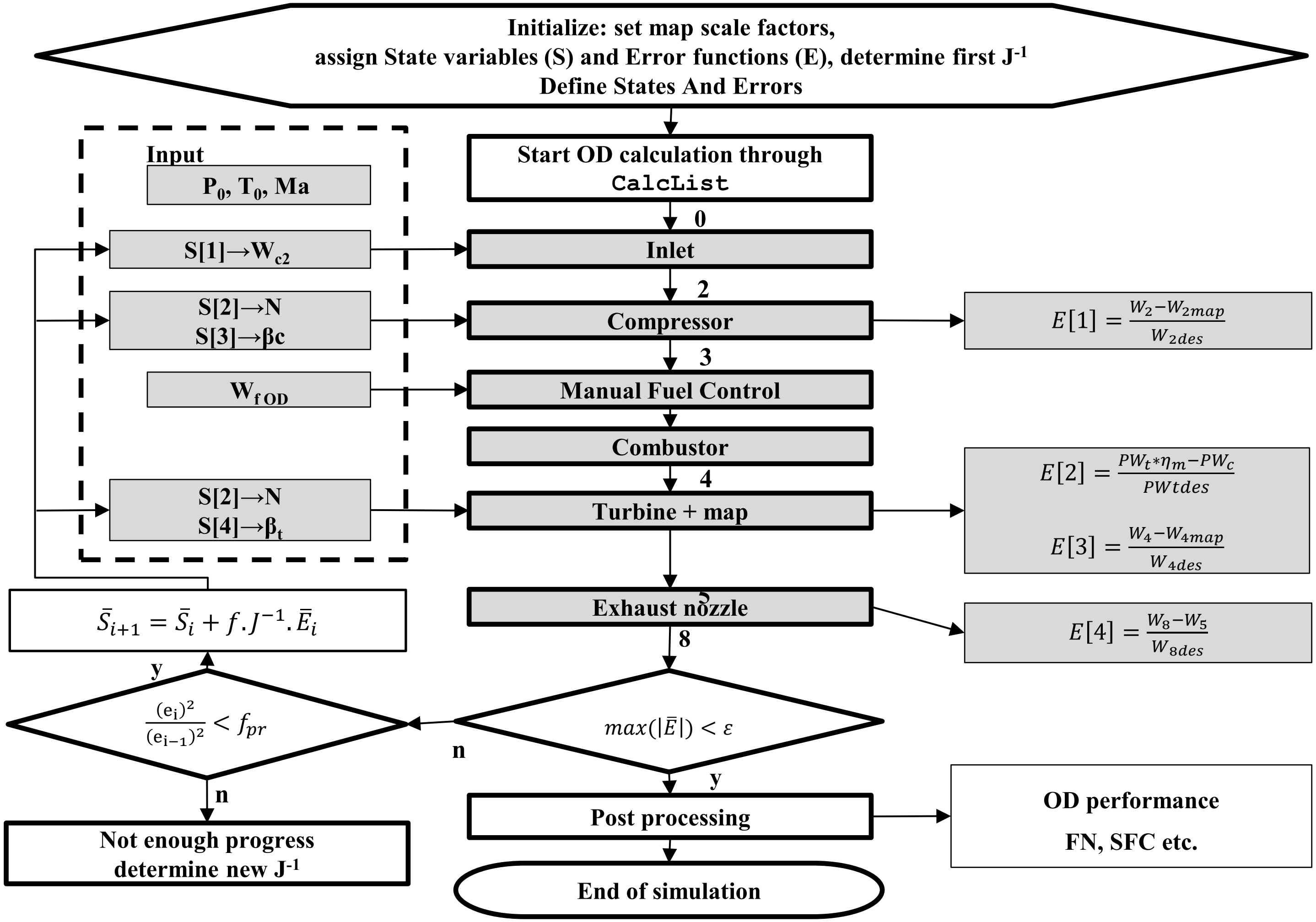
Off-design simulation

- Satisfying conservation equations
- To find the 4 unknown states we need 4 equations
  - E1 = Inlet mass flow - compressor mass flow (map) = 0
  - E2 = Compressor power + turbine power = 0
  - E3 = Combustor exit mass flow - turbine mass flow (map) = 0
  - E4 = Turbine mass flow - nozzle mass flow = 0



Off-design iteration flow chart - turbojet

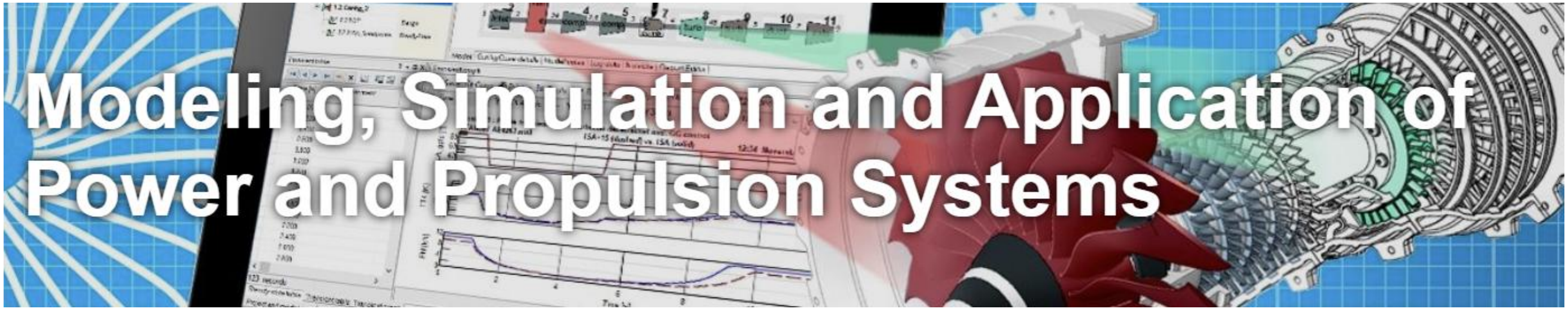
Off design simulation



Recommended literature

- W. Visser “Generic Analysis Methods for Gas Turbine Engine Performance: the development of the gas turbine simulation program GSP”, 6 January 2015, Delft University of Technology, ISBN 978-94-6259-492-0 (available from https://repository.tudelft.nl/islandora/object/uuid%3Af95da308-e7ef-47de-abf2-aedbfa30cf63).
- H. I. H. Saravanamuttoo, G. F. C. Rogers and H. Cohen, *Gas Turbine Theory*, 5th ed.: Pearson Education, 2001.
- Several authors and contributors, ‘AVT-018 Performance Prediction and Simulation of Gas Turbine Engine Operation’, NATO Research and Technology Organization 2003 RTO-TR-044
- Walsh P.P, Fletcher P., Gas turbine Performance,
  ISBN-10: 063206434X ISBN-13 : 978-0632064342
- Visser, W.P.J. and Broomhead M.J., 2000, ‘GSP, a generic object-oriented gas turbine simulation environment’, ASME paper 2000-GT-2, presented at the ASME TURBO EXPO 2000, 8-11 May 2000, Munich, Germany,
- J. Kurzke, "Advanced User-friendly Gas Turbine Performance Calculations on a Personal Computer", 95-GT-147, presented at the ASME IGTI Turbo Expo, 1995.`
- J. Kurzke, "How to get Component Maps for Aircraft Gas Turbine Performance Calculations", ASME Paper 96-GT-164, 1996.
- W. P. J. Visser and I. D. Dountchev, "Modeling Thermal Effects on Performance of small Gas Turbines", GT2015-42744, to be presented at the ASME IGTI Turbo Expo 2015, Montreal, Canada, 2015.

Delft University online learning



https://online-learning.tudelft.nl/courses/modeling-simulation-and-application-of-power-and-propulsion-systems/

New propulsion system model elements

System performance modeling

- Sustainable fuels
- Electric systems
- Heat exchangers
- Hybrid systems (gas turbines, fuel cells, batteries)
- Generators, Electric motors driving fans and propellers

- So, there is a need for
  - Extensions in modeling and simulation tools
  - ***Or, build tools from scratch ?***

From scratch

System performance modeling

- Things to think about
  - Will others use my code?
  - Will code be used later?
    - For other propulsion systems / engines?
    - Flexibility
  - Costs / Benefits of making the tool generic?
    - Add a user-friendly GUI?
    - Or just run from the (Python) interpreter
  - ?...
- Preparing your project
  - Select language, development platform
  - Design software architecture

Why from scratch?

System performance modeling

- Pros
  - Full control and flexibility
  - "I know what I am doing"
  - Coupling with other models (1..4 D models)

- Cons
  - A lot of work
    - Validation
    - Code maintenance / documentation
  - Code accessibility / readability to others
    - Especially 'if you leave the company'

Software implementation

Do you really want to start from scratch?

Language / development environment

Software implementation

- Python
  - Open-source programming language
  - Ready to use libraries for many engineering problems
  - Widely used by students, young engineers and scientists
  - Readable code and simple syntax
  - Supports Object Orientation (OO)
  - Cross-Platform Compatibility (Windows, Linux, IOS)
  - Con’s? -> memory usage and performance limitations
- Development environments (IDE)
  - Microsoft Visual Studio Code (‘VS Code’), PyCharm

Python versus previous experience

Software implementation

- Gas turbine Simulation Program GSP
  - Developed at
    - Netherlands Aerospace Centre NLR
    - Delft University of Technology
  - Object oriented, GUI
  - 4GL development environment, 30 years of legacy code
  - ~120,000 lines of code (excluding library code)

- Python "GSPy"
  - using open-source libraries

~ 1000 - 4000 lines of code !!!!

Working with Python models

Software implementation

- Run code from development environment (no GUI)
- Specify input in code
  - Example: specify Ambient conditions and a number of fuel flow steps

```
# create Ambient conditions object (to set ambient/inlet/flight conditions)
#                               Altitude, Mach, dTs,    Ps0,    Ts0
# None for Ps0 and Ts0 means values are calculated from standard atmosphere
fsys.Ambient = TAmbient('Ambient', 0, 0, 0,   0,   None,   None)

# create a control (controlling inputs to the system model)
# components like the combustor retrieve inputs like fuel flow
# input or combustor exit temperature
fsys.Control = TControl('Control', '', 0.38, 0.38, 0.06, -0.01)
```

- Adapt / extend code where needed
- Alternatives: separate input text files, or GUI (lot of extra work)

Python for system modeling

Software implementation

- Object Oriented architecture
- Main program/model file
- Using open-source libraries
- Component model implementations
- Output of simulation results

Object Orientation (OO)

Software implementation

- Encapsulation
  - An object holds both data and 'methods' (procedures, functions) working with these and data, and also external data
  - Object class (class is 'type' of object)
- Inheritance
  - An object can inherit data and methods from a 'parent class'
- Polymorphism
  - An object variable can represent objects from different classes inheriting from the same parent
- Abstraction
  - Representing the essential features without concerning about the background details

Why object orientation?

Software implementation

- Essential to build a flexible modular framework
- Rapidly implement system models with different configurations using same building blocks ('objects')
- Maintain overview ('encapsulation')
- Avoid code duplication
- Enhance code maintainability

Why object orientation?

Software implementation

- Example implementing system models with different configurations

1. Build a turbojet model
   - Inlet-compressor-combustor-turbine-exhaust nozzle

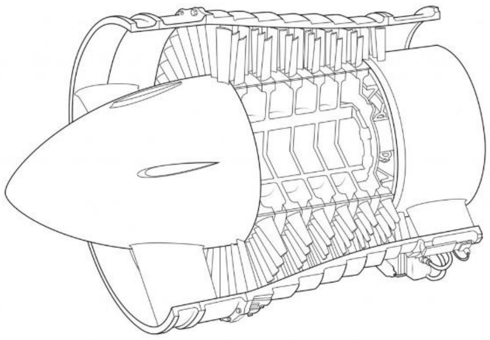
2. Build a turbofan model
   - Use the same elements as used in the turbojet model, add others
   - Without moving large chunks of code around..

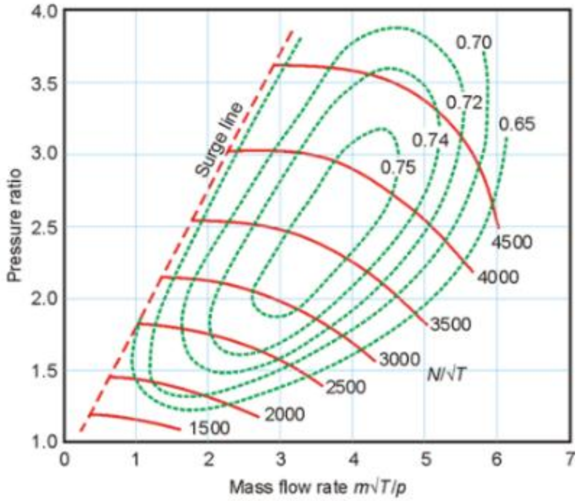
… applying inheritance, polymorphism and abstraction !

Encapsulation example

Object orientation

- Compressor component model
  - compressor object class holds
    - W, Beta (PR), Eta, N
    - functions to calculate the compression process
    - Compressor performance map
    - Functions to communicate with the system model and other component models
    - Calculate equation *residuals from states*





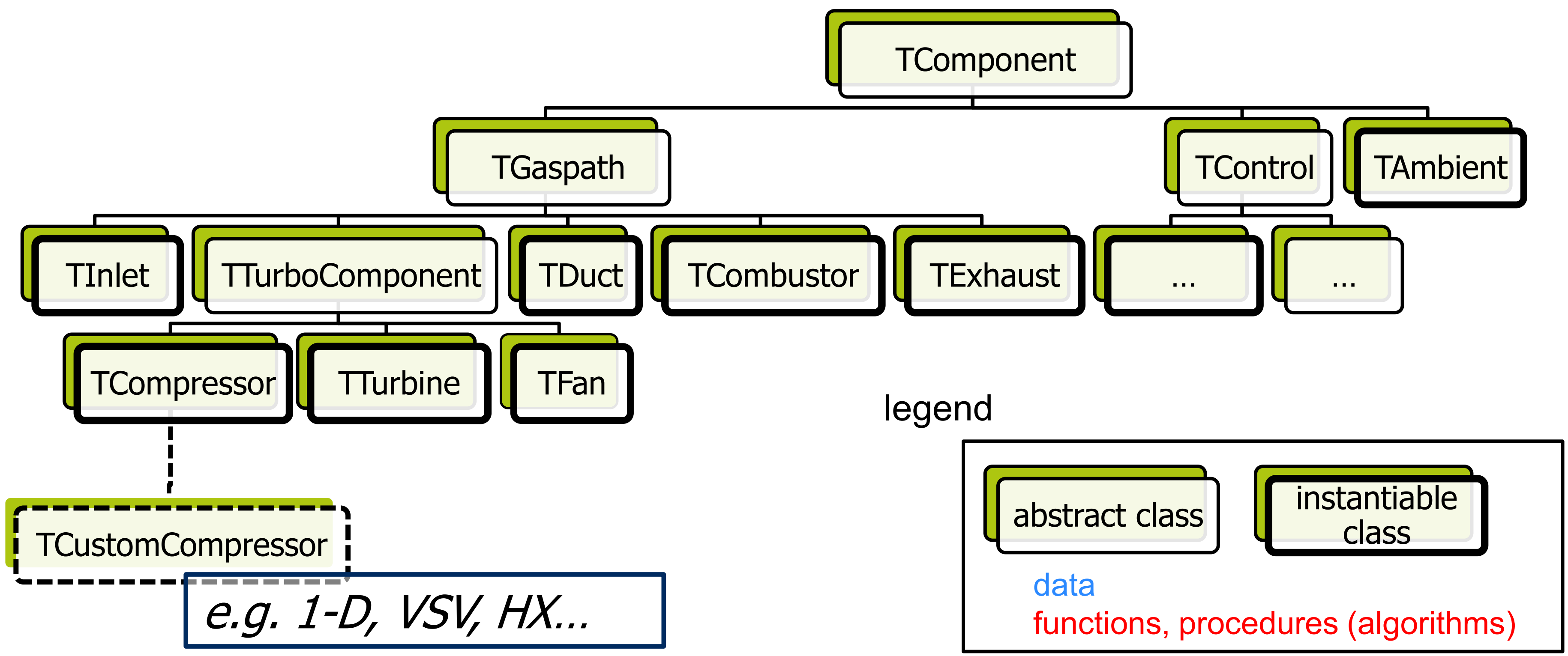
Polymorphism/abstraction

Object orientation

- Generic component model class object
  - *'abstract'* object that can represent any child object
  - has a generic 'Run' method (procedure or function)
  - Child classes inherit and 'override' the Run procedure with calculations simulating the actual process in the component
- System model
  - List of objects (of instantiable classes)
    - E.g. Inlet-compressor-combustor-turbine-exhaust
    - System model simulation subsequently calls the 'Run' methods of all objects in the list, in this example calculating from inlet through exhaust….

Inheritance example GSPy

Object Orientation



Python Polymorphism code example

Object orientation

```
# create a turbojet model list of component models
turbojet = [TInlet(      'Inlet1',       19.9, 1                    ),
            TCompressor('compressor1', 1,    16540, 6.92, 0.825),
            TCombustor( 'combustor1',  0.38, 0,    1, 1          ),
            TTurbine(   'turbine1',    1,   16540, 0.88          ),
            TDuct(      'exhaustduct', 1                          ),
            TExhaust(   'exhaust1',    1, 1, 1                    )]
```

Class   Name   Design/object data

......

```
# comp is the abstract component object
for comp in system_model:
    comp.Run(Mode, PointTime)
```

Model definition alternative

Software implementation

```
 # Generic gas turbine components
  inlet1    = TInlet('Inlet1',       '',            0,2,   19.9, 1    )
  duct1     = TDuct('exhduct',       '',            5,7,   1.0        )
  exhaust1 = TExhaust('exhaust1',   '',            7,8,9, 1, 1, 1, 1 )
  # **************** Combustor ***************************************************
  # fuel input
  combustor1 = TCombustor('combustor1', '', 3, 4, 0.38, None, 1, 1, None,      43031, 1.9167, 0, '')

  compressor1 = TCompressor('compressor1','compmap.map' , 2, 3, 1, 16540, 0.825, 1, 0.75   , 6.92, 'GG')
  turbine1 =    TTurbine(   'turbine1'   ,'turbimap.map', 4, 5, 1, 16540, 0.88 , 1, 0.50943, 0.99, 'GG’)



    fsys.systemmodel = [inlet1,
                        compressor1,
                        combustor1,
                        turbine1,
                        duct1,
                        exhaust1]
```

Python open-source libraries

Software implementation

- Ready to use libraries
  - Standard atmosphere *aero-calc*
  - Thermodynamics *cantera*
    - Compression, expansion in compressors, turbine, exhausts, nozzles
    - Combustion chemical reactor model
  - Multidimensional arrays *numpy*
  - Numerical methods *scipy*
    - 'root' generic solver ('Krylov' method)
  - Output data export
    - Export to .csv *pandas*
      - Make graphs in Excel
    - Map graphs with operating curves *matplotlib*

Aero-calc library

Software implementation

- Aero-calc functions used for calculation of ambient/flight conditions from altitude, flight Mach, dT
  - alt2temp
  - alt2press
  - temp2speed_of_sound

```
if self.Tsa == None:
    # Tsa not defined, use standard atmosphere
    self.Tsa = ac.std_atm.alt2temp(self.Altitude, alt_units='m', temp_units='K')
    # for standard atmosphere, use dTs if defined
    if self.dTs != None:
        self.Tsa = self.Tsa + self.dTs
if self.Psa == None:
    # Ps0 not defined, used standard atmosphere
    self.Psa = ac.std_atm.alt2press(self.Altitude, alt_units='m', press_units='pa')
self.Tta = self.Tsa * ( 1 + 0.2 * self.Macha**2)
self.Pta = self.Psa * ((self.Tta/self.Tsa)**3.5)
# set values in the Gas_Ambient phase object conditions
self.Gas_Ambient.TPY = self.Tta, self.Pta, fg.s_air_composition_mass
self.V = self.Macha * ac.std_atm.temp2speed_of_sound(self.Tsa, speed_units = 'm/s', temp_units = 'K')
```

Cantera library

Software implementation

- Open-source suite of tools for problems involving chemical kinetics, thermodynamics, and transport processes
- https://cantera.org/index.html
- Jetsurf mechanism used for
  - Gas path transport
  - Compression/expansion
  - Combustion

https://web.stanford.edu/group/haiwanglab/JetSurF/JetSurF2.0/Index.html

Using Cantera (ct) **gas model**

Software implementation

- define gas model

```
gas = ct.Solution('jetsurf.yaml')
```

- Set T, P, composition (mass mixing ratios)

```
gas.TPY = 1200, 101325, 'CH4:1, O2:2, N2:7.52'
```

- Or set P and S like gas.PSY = ...

- Get T, H, S or P

```
P = gas.P       S = gas.S
T = gas.T       H = gas.H
```

Using Cantera (ct) **mass flow/transport**

Software implementation

- Define a quantity (mass flow) of gas

GasIn = ct.Quantity(gas, mass flow)

- Set entropy or enthalpy and pressure using

GasIn.SP = entropy value, pressure value

GasIn.HP = enthalpy value, pressure value

- Isentropic compression of GasIn mass flow (with pressure ratio PR)

```
Sin = GasIn.s
Pout = GasIn.P*PR
GasOut.SP = Sin, Pout # get GasOut at constant s and higher P
Hisout = GasOut.phase.enthalpy_mass # isentropic exit specific enthalpy
```

Using Cantera (ct) **combustion**

Software implementation

- chemical equilibrium reactor model

- 2 options for fuel specification

User specified composition (single species)

T and P

or

User specified LHV (mix of 'unknown' species, e.g. Kerosine, Jet-A)

T and P

average H/C and O/C ratio of the fuel molecules

- In both cases end-combustion H, T, P and composition can be calculated

User specified fuel composition

Cantera - combustion

fuel = ct.Quantity(fg.gas) *define fuel*

fuel.mass = fuel mass flow

fuel.TPY = T, P, FuelComposition *define T, P, composition*

FuelComposition is a string variable, e.g.:
'C12H26:1` or a 5 to 1 mixture : 'C12H26:5, C8H12:1'

GasOut = GasIn + fuel *mix fuel with air*

GasOut.equilibrate('HP') *chemical equilibrium combustion end state*

T4 = GasOut.T *get combustion end T (or H, S etc)*

User specified LHV

Cantera - combustion

- More complicated
- Cannot use Equilibrate function
- Need to calculate chem. equilibrium end composition from
  - Air composition
  - Fuel H/C and O/C ratios
  - Mole fractions book keeping (converting between mass – moles)

Numpy and Scipy

Software implementation

- https://docs.scipy.org/doc/
- Numpy
  - multidimensional array operations
- Scipy
  - solving system model non-linear algebraic/differential equations
  - using **Root** function
    - finding root of a vector function
      > i.e. error vector as function of the state vector
      > result is a valid system steady-state or transient point
    - newton-Raphson like iteration
    - several solver methods
  - Most suitable solver method ‘Krylov’

Solver code

Software implementation

- ***Do_Run*** runs all component model code and returns the vector of equation errors (residuals)

```
def Do_Run(Mode, PointTime, states):
    fsys.states = states.copy()
    fsys.reinit_system()
    fsys.Ambient.Run(Mode, PointTime)
    fsys.Control.Run(Mode, PointTime)
    for comp in fsys.systemmodel:
        comp.Run(Mode, PointTime)
    return fsys.errors
```

- ***residuals*** is the vector function (returns all 0's if states is valid OP)

```
def residuals(states):
    # residuals will return residuals of system conservation equations,
    # schedules, limiters etc. the residuals are the errors returned by Do_Run
    # test with GSP final performan with 0.3 kg/s fuel at ISA static
    # states = [+9.278E-01,  +9.438E-01,  +8.958E-01,  +1.008E+00]
    return Do_Run(Mode, inputpoints[ipoint], states)
```

- ***fsys.states*** = the initial value of the state vector
- ***solution*** = the state vector representing a valid operating point

```
solution = root(residuals, fsys.states, method='krylov', options = options)
```

Main program for turbojet DP & OD simulation

Software implementation

(key code lines only)

Create Oper.conds.

```
fsys.Ambient = TAmbient('Ambient', 0, 0, 0,   0,   None,   None)
fsys.Control = TControl('Control', '', 0.38, 0.38, 0.06, -0.01)
# system model with gas path components in calculation order
```

Model creation & definition

```
fsys.systemmodel = [TInlet('Inlet1',      '',     0, 2,   19.9, 1     ),
                    TCompressor('compressor1','compmap.map' , 2, 3, 1, 16540, 0.825, 1, 0.75   , 6.92, 'GG'),
                    TCombustor('combustor1', '', 3, 4, 0.38, None, 1, 1, None,      43031, 1.9167, 0, ''),
                    TTurbine(   'turbine1'   ,'turbimap.map', 4, 5, 1, 16540, 0.88 , 1, 0.50943, 0.99, 'GG'),
                    TDuct('exhduct',      '',     5, 7,   1.0         ),
                    TExhaust('exhaust1',  '',     7,8,9, 1, 1, 1, 1 )]
```

Procedure running all comps.

```
def Do_Run(Mode, PointTime, states):
    for comp in fsys.systemmodel:
        comp.Run(Mode, PointTime)
    return fsys.errors
    # run the system model Design Point (DP) calculation for CRP
```

DP simulation

```
fsys.Ambient.SetConditions('DP', 0, 0, 0, None, None)
Do_Run('DP', 0, fsys.states)  # in DP always fsys.states = [1, 1, 1, 1,...]
    # series of OD points
```

OD simulation

using root to Find OD points

```
inputpoints = fsys.Control.Get_OD_inputpoints()
fsys.Ambient.SetConditions('OD', 0, 0, 0, None, None)
def residuals(states):
    return Do_Run('OD', inputpoints[ipoint], states)
fsys.reinit_states_and_errors()
for ipoint in inputpoints:
    solution = root(residuals, fsys.states, method='krylov', {})
```

Passing gas conditions in Python

Software implementation

- `comp.Run(` is run in order given in `fsys.systemmodel`
- `fsys.systemmodel` list must be ordered such that
  - component gas path upstream conditions
  - other component/control parameters needed for "Run"

  have been calculated

- Basic component "Run" code takes inlet conditions from *gaspath_conditions[…] dictionary*

```
# use dictionary for gas path conditions oriented by gas
path station number
gaspath_conditions = {}


def Run(self, Mode, PointTime):
…
self.GasIn = fsys.gaspath_conditions[self.stationin]
…
fsys.gaspath_conditions[self.stationout] = self.GasOut
```

Component model implementations

Software implementation

- Implementation of component performance maps
- Inlets
- Turbo machinery
  - Compressors
  - Fans
  - Turbines
- Combustors
- Ducts
- Exhaust nozzles and diffusers

Abstract TGaspath class

```
def Run(self, Mode, PointTime):
    self.GasIn = fsys.gaspath_conditions[self.stationin]

    if Mode == 'DP':
        # create GasInDes, GasOut cantera Quantity (GasIn already created)
        self.GasInDes = ct.Quantity(self.GasIn.phase, mass = self.GasIn.mass)
        self.GasOut = ct.Quantity(self.GasIn.phase, mass = self.GasIn.mass)
        self.Wdes = self.GasInDes.mass
        self.Wcdes = self.Wdes * fg.GetFlowCorrectionFactor(self.GasInDes)
        self.Wc = self.Wcdes
    else:
        self.GasOut.TPY = self.GasIn.TPY
        self.GasOut.mass = self.GasIn.mass

    fsys.gaspath_conditions[self.stationout] = self.GasOut
    return self.GasOut
```

TInlet run code

Component model implementations

- TInlet is descending from TGaspath class
- Overriding TGaspath component ***Run*** code

```
...
self.GasOut.TP = self.GasIn.T, self.GasIn.P * self.PR
....
```

States and errors in component models

Software implementation

- The system model requires an equal number of
  - state variables ('states')
  - and error functions (equation residuals)
- Component model code
  - initializes states and/or errors
  - depending on the type of component
  - See turbojet OD turbojet simulation example slides
    - 4 states - 4 errors
- Component map input parameters typically become states:
  - Nc and Beta for a compressor

TInlet code - adding a state variable

Component model implementation

```
def Run(self, Mode, PointTime):
    if Mode == 'DP':
        fsys.gaspath_conditions[self.stationin].mass = self.Wdes
    super().Run(Mode, PointTime)
    self.GasIn.TP = self.GasIn.T, self.GasIn.P
    if Mode == 'DP':
        self.GasIn.mass = self.Wdes
        self.wcdes = self.GasIn.mass * fg.GetFlowCorrectionFactor(self.GasIn
        self.wc = self.wcdes
        self.PR = self.PRdes
        fsys.states = np.append(fsys.states, 1)
        self.istate_wc = fsys.states.size-1   # add state for corrected inle
    else:
        self.wc = fsys.states[self.istate_wc] * self.wcdes
        self.GasIn.mass = self.wc / fg.GetFlowCorrectionFactor(self.GasIn)
        self.GasOut.TP = self.GasIn.T, self.GasIn.P * self.PRdes
        # this inlet has constant PR, no OD PR yet (use manual input in code
        self.PR = self.PRdes
    self.GasOut.TP = self.GasIn.T, self.GasIn.P * self.PR
    self.GasOut.mass = self.GasIn.mass
    self.RD = self.GasIn.mass * fsys.Ambient.V
    # add ram drag to system level ram drag (note that multiple inlets may e
    fsys.RD = fsys.RD + self.RD
    return self.GasOut
```

Adding state variable

TCompressor - adding 2 states & 1 error

Software implementation

```python
def Run(self, Mode, PointTime):
    super().Run(Mode, PointTime)
    if Mode == 'DP':
        self.PW = fu.Compression(self.GasIn, self.GasOut, self.PRdes, self.Etades)
        self.shaft.PW_sum = self.shaft.PW_sum - self.PW

        self.map.ReadMapAndSetScaling(self.Ncdes, self.Wcdes, self.PRdes, self.Etades)

        # add states and errors
        if self.SpeedOption != 'CS':
            fsys.states = np.append(fsys.states, 1)
            self.istate_n = fsys.states.size-1
            self.shaft.istate = self.istate_n
        fsys.states = np.append(fsys.states, 1)
        self.istate_beta = fsys.states.size-1
        # error for equation GasIn.wc = wcmap
        fsys.errors = np.append(fsys.errors, 0)
        self.ierror_wc = fsys.errors.size-1
        # calculate parameters for output
        self.PR = self.PRdes
    else:
```

Adding state variables

Adding error variable

Compressor - Off-design mode

Component model implementation

States and errors at work….

N from state var.

Beta from state var.

Compress. calc

error var. calc

```
else:
    if self.SpeedOption != 'CS':
        self.N = fsys.states[self.istate_n] * self.Ndes
    self.Nc = self.N / fg.GetRotorspeedCorrectionFactor(self.GasIn)

    self.Wc, self.PR, self.Eta = self.map.GetScaledMapPerformance(self.Nc, fsys.states[self.istate_beta])

    self.PW = fu.Compression(self.GasIn, self.GasOut, self.PR, self.Eta)

    self.shaft.PW_sum = self.shaft.PW_sum - self.PW
    self.W = self.Wc / fg.GetFlowCorrectionFactor(self.GasIn)
    fsys.errors[self.ierror_wc ] = (self.W - self.GasIn.mass) / self.Wdes

    # set out flow rate to W according to map
    # may deviate from self.GasIn.mass during iteration: this is to propagate the effect of mass flow error
    # to downstream components for more stable convergence in the solver (?)
    self.GasOut.mass = self.W

return self.GasOut
```

Implementation of other components

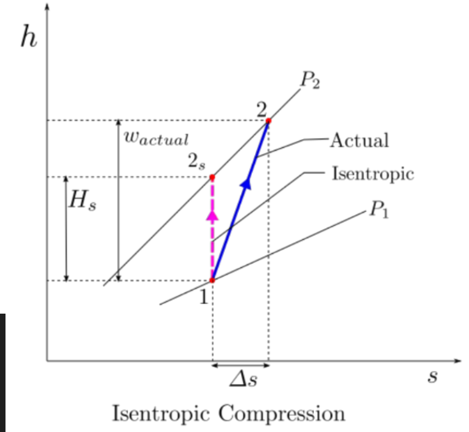
Component model implementation

- Generic functions
  - Compression
  - Expansion
- See code for
  - Combustion (also see slides on Cantera)
  - Duct: simple pressure loss
  - Exhaust nozzle: isentropic expansion
  - Control
  - Ambient conditions (see slides on Aerocalc)

Compression code

Component model implementation

- Generic function used for calculating compressor or fan end conditions
  - Use Cantera for isentropic compression
  - Eta isentropic

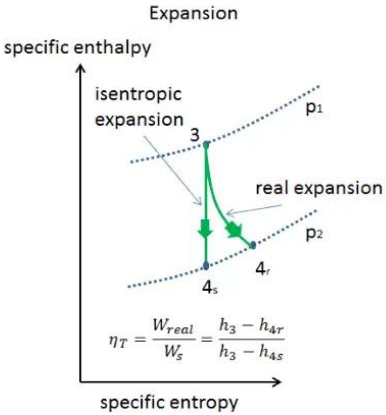



```python
def Compression(GasIn: ct.Quantity, GasOut: ct.Quantity, PR, Etais):
    Sin = GasIn.s
    Pout = GasIn.P*PR
    GasOut.SP = Sin, Pout # get GasOut at constant s and higher P
    Hisout = GasOut.phase.enthalpy_mass # isentropic exit specific enthalpy
    Hout = GasIn.phase.enthalpy_mass + (Hisout - GasIn.phase.enthalpy_mass) / Etais
    GasOut.HP = Hout, Pout
    PW = GasOut.H - GasIn.H
    return PW
```

Turbine expansion code

Component model implementation

- Generic function used for calculating turbine end conditions
  - Use Cantera for isentropic expansion
  - Eta isentropic




```python
def TurbineExpansion(GasIn: ct.Quantity, GasOut: ct.Quantity, PR, Etais):
    Pout = GasIn.P / PR
    GasOut.SP = GasIn.entropy_mass, Pout
    final_enthalpy_is = GasOut.enthalpy_mass
    # eta_is = (initial_enthalpy - final_enthalpy) / (initial_enthalpy - final_enthalpy_is)
    final_enthalpy = GasIn.enthalpy_mass - (GasIn.enthalpy_mass - final_enthalpy_is) * Etais
    GasOut.HP = final_enthalpy, Pout
    PW = GasIn.H - GasOut.H
    return PW
```

Component characteristics or “maps”

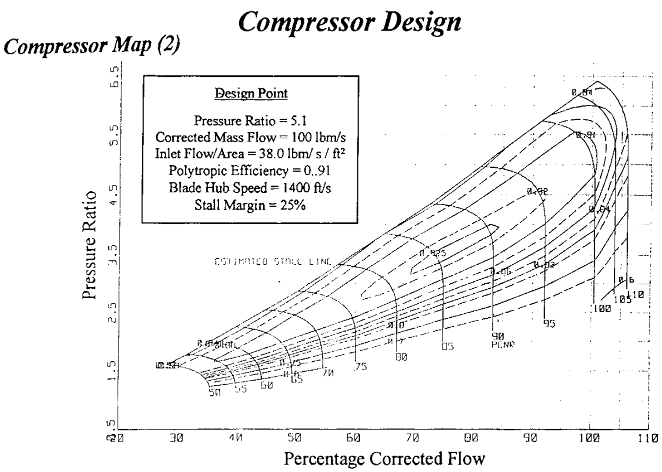
Component model implementation

- Tabulated relations of performance parameters
- Applicable to various component models
- Examples
  - Simple: non-linear pressure loss relationship
  - Complex: turbo machinery maps:
    - fans, compressors, turbines
- Turbomachinery
  - Parameters ‘corrected’ or ‘reduced’ to ISA
  - Nc, Wc, PR, Eta, (Re)

Turbomachinery maps

Component model implementation

- GSP and Gasturb map format
- Relations among
  - Nc, Wc, PR, Eta, (Re)



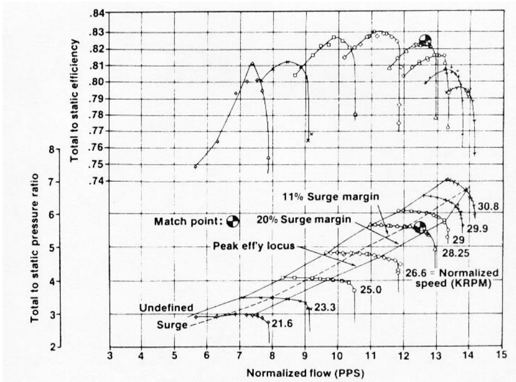
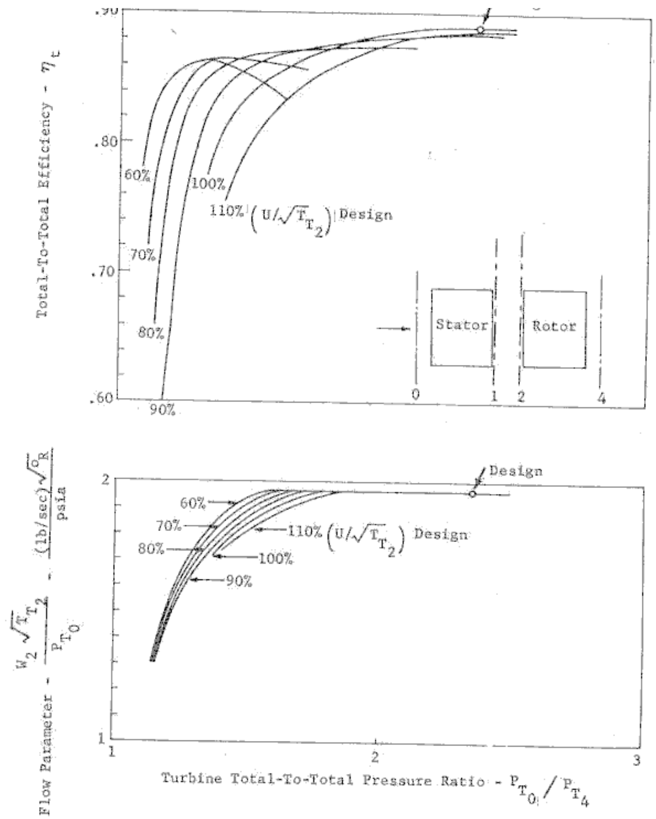


Fig. 2 Medium Pressure Ratio Centrifugal Compressor 5.5 PR Match of 7:1 PR Design.

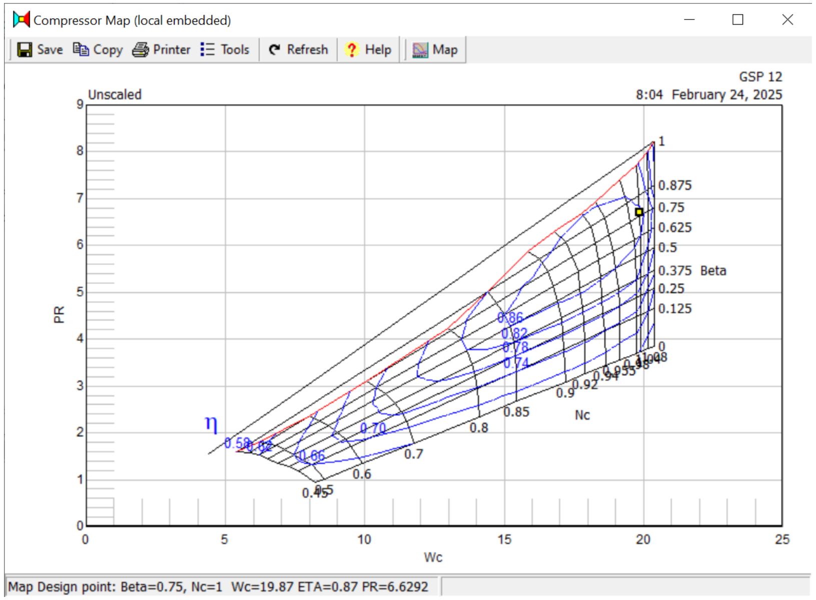


Compressor maps **Turbine map**

Maps in component models

Component model implementation

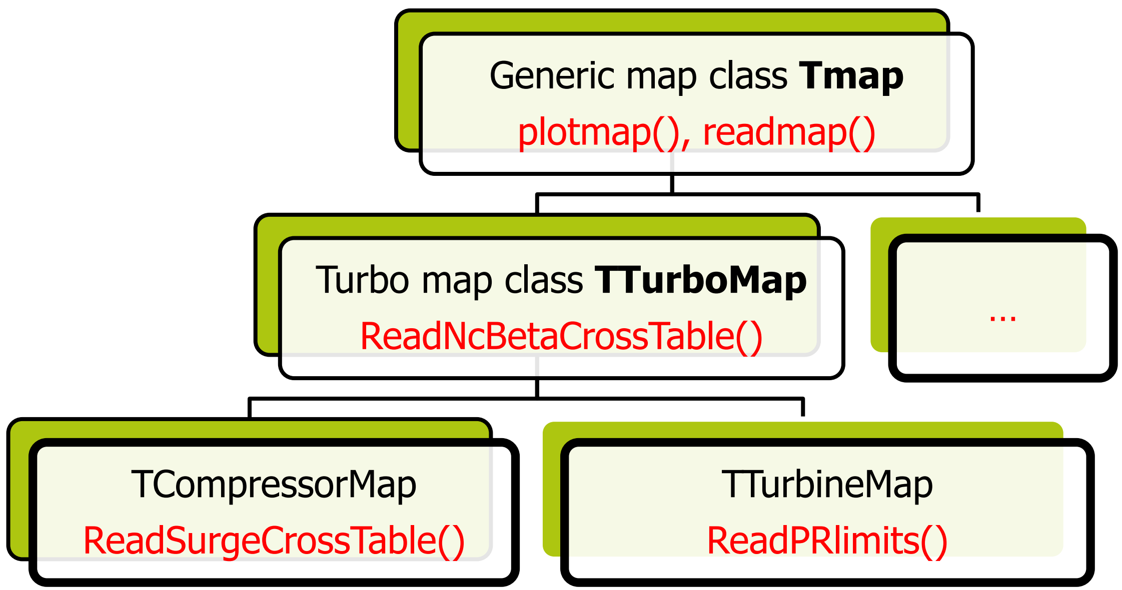
- 1 or more map parameters define component operating point
- For turbomachinery
  - 2 parameters define operating point
    - Nc
    - Beta (auxiliary parameter)

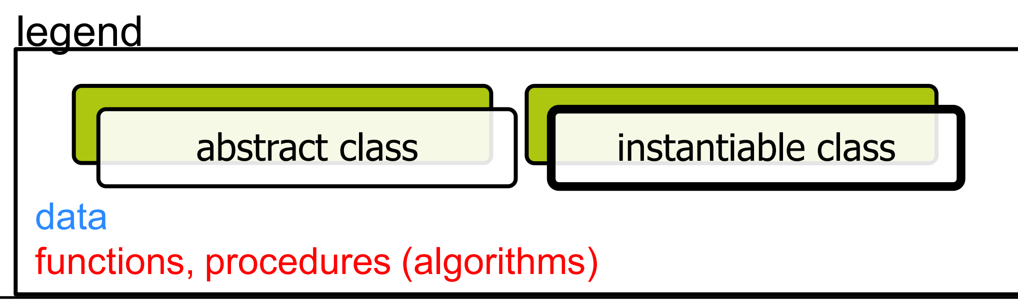


- With values for Nc and Beta, PR, Wc and Eta can be obtained from lookup tables
  - SciPy *RegularGridInterpolator* cubic interpolation

Turbomachinery map classes

Component model implementations





Plotting turbomachinery maps

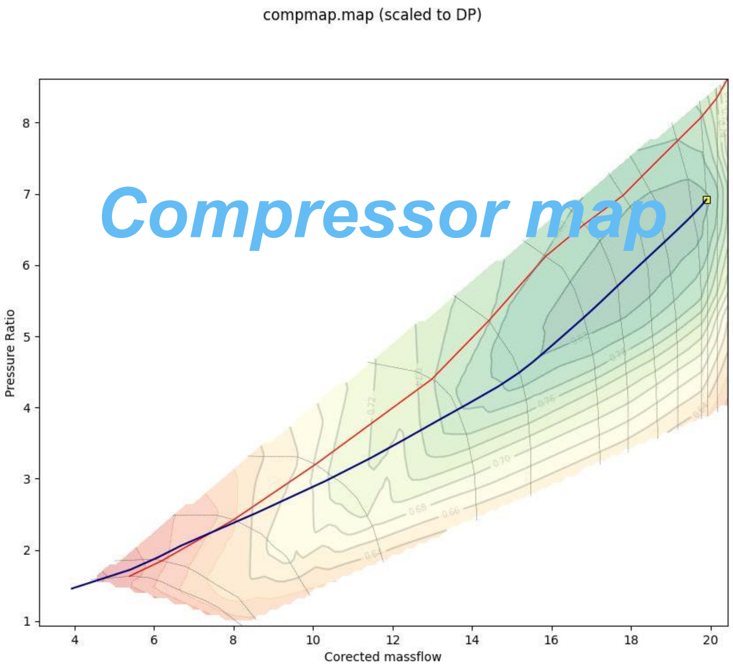
Component model implementation

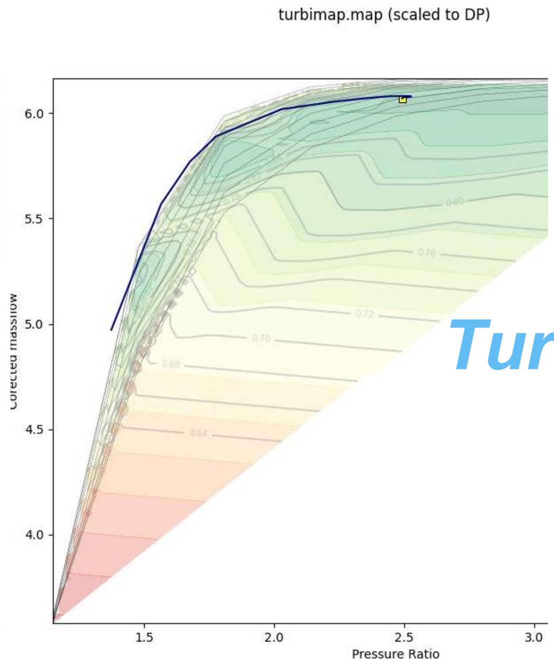
- Consider the following data when plotting
  - Pressure ratio, Mass flow, Spool speed
  - Efficiency contours
  - Limits
    - Surge line for compressor
- Design point / CRP (cycle reference point)
- Map scaling to CRP
  - For OD operating lines (transient or steady state)
- Use standard plot libraries! matplotlib

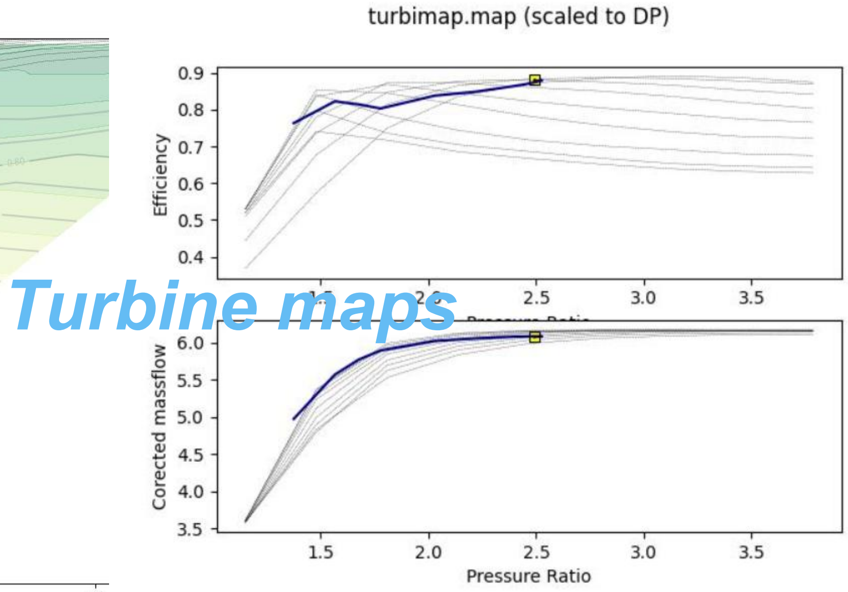
Turbomachinery map plot examples

Component model implementations

- Few lines of code!
- OD operating curves in scaled maps







Python implementation - output

Component model implementation

- Generate global output table *pandas*
  - Code output per component
  - System level output (PW, FN, Wf, W2 etc.)
  - Export to .csv
  - Make graphs in Excel (or separate Python plot program)
- Component maps
  - Graphical output *matplotlib*
  - Compressors, fans, turbines, …
  - Draw operating lines in scaled maps

Demonstration

- Running single spool turbojet model
  - Environment Microsoft VScode
  - Design + Off-design to IDLE at ISA SL
- 2 spool Turbofan
  - Design + Off-design at Ma 0.8 / 5,000 m

Using the GSPy code

Software implementation

- https://github.com/wvisser1958/GSPy *
- Limited set of component model classes
- With current open-source code you can
  - Run Design and off-design simulations of gas turbines
  - Define other engine configurations
    - Turbofan, turboshaft etc.
  - Other fuels
    - Easily adapt fuel in combustor component
- This code is the groundwork for you continue developing your own specific models

** The code is provided under the Apache open-source license*

Extending the baseline code - 1

Software implementation

- Component model extensions
  - Variable geometry
  - Bleed
  - Turbine cooling
  - Polytropic efficiencies (design studies)
  - 1-D model code
  - Emission models
- Transient simulation
  - Extending the conservation equations (error function in the code)

Extending the baseline code - 2

Software implementation

- Adding component model classes
  - Fan / splitter
  - Mixer
  - Recuperator / heat exchanger
  - Turbogenerator
  - Secondary air flows (bleed flow, cooling flows)
- Additional *Hybrid* propulsion system elements
  - Generators / Loads
  - electric motors
  - propulsors
  - batteries,
  - fuel cells

Extending the baseline code - 3

Software implementation

- Extend user interface
  - Extend output Graphs, tables, export
  - Add GUI
- Coupling with other models
  - API (Application Programming Interface)
  - Running *GSPy* code via API
  - *GSPy* controlling execution of other models
- Fidelity
  - 1-D component models
    - Turbomachinery
    - Combustors

Conclusion

- GSPy open-source code available from
  - https://github.com/wvisser1958/GSPy
- This code is mostly the groundwork!
  - It's up to you to further refine it!
  - Extensions, numerical stability, maps, fidelity, couplings etc.
- Contact authors for questions
  - Dr. Wilfried Visser w.p.j.Visser@tudelft.nl
  - Oscar Kogenhop MSc oscar.kogenhop@epcor.nl

Questions?