



# Modular Initiator Modelling of Engines

A component-based approach

H.A. Mulder BSc.

Technische Universiteit Delft



# MODULAR INITIATOR MODELLING OF ENGINES

A COMPONENT-BASED APPROACH

by

**H.A. Mulder BSc.**

in partial fulfillment of the requirements for the degree of

**Master of Science**  
in Aerospace Engineering

at the Delft University of Technology,  
to be defended publicly on Tuesday August 21st, 2018 at 09:00 AM.

Supervisor:	Dr. ir. R. Vos	
Thesis committee:	Prof. dr. A. G. Rao,	TU Delft
	Dr. ir. R. Vos,	TU Delft
	Dr. ir. J.M.J.E Van Campen,	TU Delft

An electronic version of this thesis is available at <http://repository.tudelft.nl/>.

## ABSTRACT

Over the past decades, the turbofan engine has established itself as the prime propulsor for airliners. However, recent developments towards propfans, unducted fans and even hybrid propulsion indicate a possible shift away from this well-established turbofan engine architecture for future aircraft designs.

In order to enable the incorporation of these new engine design choices in the Aircraft Design Initiator of the Delft University of Technology Flight Performance and Propulsion group, a modular engine design, mass and performance estimation tool called MIME (Modular Initiator Model for Engines) has been created. The modular nature of this tool offers the freedom to piece together any engine architecture from the available modules, extending the modelling capabilities for engines within the Initiator. For these pieced-together engines, the model is then able to estimate not only the thrust, fuel consumption and mass, but also the engine dimensions and many details about specific components, including stage and blade numbers, internal geometry and component thrust contributions. Even though the current set of available modules is limited to standard turbofan components, the flexibility and extendability of the core framework allows for future-proofing engine modelling within the Initiator.

For three testcases performed on the isolated MIME model, the on-design thrust and fuel consumption estimates were shown to have reasonable accuracy, while the engine mass is significantly overestimated due to an unresolved error in spool speed estimation. As a consequence, a full Initiator run of a testcase using only the MIME engine design and performance estimations yielded reasonable outcomes, while the same case failed to converge when engine mass estimation was also done through MIME due to the snowball effect of the mass overestimation.

Even though the current code is not perfect and there is room for improvement in the current component modules, the modularity and expandability of the code does promise to significantly expand the engine modelling capabilities of the Initiator.



# PREFACE

Carrying out this thesis project has been a long quest, as quests are supposed to be: full of trying times, difficult questions and initially incorrect answers. [1]

During this quest, I was fortunate enough to have the guidance and supervision of dr. ir. Roelof Vos, to whom I owe much gratitude. Also, dr. ir. Matteo Pini was kind enough to point me back to the right path when I strayed too far.

A quest such as this puts a significant strain on not only its undertaker, but also on those around him. Therefore, I would like to thank my loving parents for supporting me throughout my life up to this point in general, and during my studies and this thesis project in particular. I would also like to thank my sister and other family members for providing me with the motivation to carry this project through.

Finally, I would like to thank my girlfriend for her patience, understanding, motivational remarks, gentle and not-so-gentle encouragements and feedback and for putting up with my frustations, motivational lapses, thesis-induced depressions and everything else. I would not have been able to do this without her support.

*H.A. Mulder BSc.  
Delft, June 2018*



# CONTENTS

List of Figures	ix
List of Tables	xi
<b>I Modular Initiator Modelling of Engines</b>	<b>1</b>
<b>1 Introduction</b>	<b>3</b>
1.1 Problem statement	3
1.2 Thesis programming objectives	4
1.3 Report outline	5
<b>2 Modular Initiator Model of Engines modelling approach</b>	<b>7</b>
2.1 Overall model structure	7
2.2 Component sizing and on-design performance approach	8
2.3 Mass estimation approach	9
<b>3 Theoretical considerations for engine components</b>	<b>11</b>
3.1 Axial compressor	11
3.1.1 On-design computations	11
3.1.2 Mass estimation calculations	16
3.2 Axial turbine	17
3.2.1 On-design calculations	17
3.2.2 Mass estimation	20
3.3 Fan	20
3.3.1 On-design calculations	20
3.3.2 Mass estimation	22
3.4 Duct	22
3.5 Combustor	23
3.5.1 On-design calculations	23
3.5.2 Mass estimation	25
3.6 Inlet	25
3.6.1 On-design calculations	26
3.6.2 Mass estimation	26
3.7 Nozzle	26
3.7.1 On-design calculations	27
3.7.2 Mass estimation	28
3.8 Shaft	28
3.9 Flow	29
3.10 Frame	29
3.11 Lower-level components	29
3.11.1 Blade	29
3.11.2 Disk	31
3.11.3 Inlet guide vanes	32
3.12 Per-component thrust	32
<b>4 Code verification and validation</b>	<b>33</b>
4.1 Verification	33
4.1.1 Stations	33
4.1.2 Blades	34
4.1.3 Disks	36
4.1.4 Inlets	36

4.1.5	Ducts . . . . .	37
4.1.6	Nozzles . . . . .	37
4.1.7	Combustors . . . . .	37
4.1.8	Compressor stages . . . . .	38
4.1.9	Turbine stages . . . . .	39
4.1.10	Compressors . . . . .	39
4.1.11	Turbines . . . . .	40
4.1.12	Fans . . . . .	40
4.1.13	Shafts . . . . .	41
4.1.14	Flows . . . . .	41
4.2	Validation . . . . .	42
4.2.1	Methods . . . . .	42
4.2.2	Testcases . . . . .	42
4.2.3	Results . . . . .	44
<b>5</b>	<b>Full Initiator case study</b>	<b>51</b>
5.1	Aircraft description . . . . .	51
5.2	Baseline simulation . . . . .	51
5.3	MIME simulations . . . . .	53
5.3.1	MIME for engine design only . . . . .	53
5.3.2	MIME for engine design and mass estimation . . . . .	53
5.4	Comparison . . . . .	55
<b>6</b>	<b>Results and conclusions</b>	<b>59</b>
6.1	Software results . . . . .	59
6.2	Case study results . . . . .	59
6.2.1	Isolated case studies . . . . .	59
6.2.2	Full aircraft case study . . . . .	60
6.3	Conclusions . . . . .	60
<b>7</b>	<b>Recommendations</b>	<b>61</b>
7.1	Improvements on current program . . . . .	61
7.1.1	General architecture . . . . .	61
7.1.2	Component-specific improvements . . . . .	61
7.2	Capability extensions . . . . .	62
7.2.1	Off-design performance estimation . . . . .	62
7.2.2	Addition of components . . . . .	62
<b>II</b>	<b>Code report</b>	<b>63</b>
<b>8</b>	<b>Introduction</b>	<b>65</b>
<b>9</b>	<b>Requirements, Architecture and Interfaces</b>	<b>67</b>
9.1	Interfaces . . . . .	67
9.2	Requirements . . . . .	67
9.3	Architecture . . . . .	68
<b>10</b>	<b>Class descriptions</b>	<b>69</b>
10.1	The AeroEngine class . . . . .	69
10.1.1	Properties . . . . .	70
10.1.2	Methods . . . . .	70
10.2	Axialcompressor . . . . .	77
10.2.1	Properties . . . . .	79
10.2.2	Methods . . . . .	79
10.3	Axialturbine . . . . .	85
10.3.1	Properties . . . . .	85
10.3.2	Methods . . . . .	86

10.4	<a href="#">Axcompressorstage</a>	87
10.4.1	<a href="#">Properties</a>	87
10.4.2	<a href="#">Methods</a>	89
10.5	<a href="#">Axturbinestage</a>	94
10.5.1	<a href="#">Properties</a>	96
10.5.2	<a href="#">Methods</a>	96
10.6	<a href="#">Fan_MIME</a>	99
10.6.1	<a href="#">Properties</a>	99
10.6.2	<a href="#">Methods</a>	101
10.7	<a href="#">Inlet</a>	102
10.7.1	<a href="#">Properties</a>	103
10.7.2	<a href="#">Methods</a>	103
10.8	<a href="#">Combustor</a>	104
10.8.1	<a href="#">Properties</a>	104
10.8.2	<a href="#">Methods</a>	105
10.9	<a href="#">Nozzle</a>	106
10.9.1	<a href="#">Properties</a>	106
10.9.2	<a href="#">Methods</a>	106
10.10	<a href="#">Duct</a>	108
10.10.1	<a href="#">Properties</a>	108
10.10.2	<a href="#">Methods</a>	108
10.11	<a href="#">Blade</a>	109
10.11.1	<a href="#">Properties</a>	109
10.11.2	<a href="#">Methods</a>	110
10.12	<a href="#">Disk</a>	111
10.12.1	<a href="#">Properties</a>	111
10.12.2	<a href="#">Methods</a>	112
10.13	<a href="#">Flow</a>	112
10.13.1	<a href="#">Properties</a>	113
10.13.2	<a href="#">Methods</a>	113
10.14	<a href="#">Shaft</a>	113
10.14.1	<a href="#">Properties</a>	114
10.14.2	<a href="#">Methods</a>	114
10.15	<a href="#">Station</a>	114
10.15.1	<a href="#">Properties</a>	116
10.15.2	<a href="#">Methods</a>	116
10.16	<a href="#">Shaftstation</a>	116
10.16.1	<a href="#">Properties</a>	117
10.16.2	<a href="#">Methods</a>	117
10.17	<a href="#">InletGuideVane</a>	117
10.17.1	<a href="#">Properties</a>	117
10.17.2	<a href="#">Methods</a>	118
10.18	<a href="#">Frame</a>	118
10.18.1	<a href="#">Properties</a>	118
10.18.2	<a href="#">Methods</a>	119
10.19	<a href="#">Superclasses</a>	119
10.19.1	<a href="#">Component</a>	119
10.19.2	<a href="#">Aerocomponent</a>	120
<b>11</b>	<b><a href="#">Integration into the Initiator</a></b>	<b>121</b>
11.1	<a href="#">Goals</a>	121
11.2	<a href="#">Implementation</a>	121
11.3	<a href="#">Potential improvements</a>	122
<b>12</b>	<b><a href="#">Guide for expansion</a></b>	<b>123</b>
12.1	<a href="#">Creating a MIME module</a>	123
12.2	<a href="#">Connecting the module</a>	123



**Bibliography**

**125**

# LIST OF FIGURES

2.1	Engine build-up schematic	8
2.2	Engine design and sizing workflow	8
3.1	14-stage axial compressor of a Wright J65 engine	12
3.2	Axial compressor design procedure	13
3.3	Velocity triangles for a typical compressor stage	13
3.4	Axial compressor stage rotor design procedure	14
3.5	Axial compressor stage stator design procedure	14
3.6	Compressor mass estimation components	16
3.7	3-stage axial turbine of a J79 engine	17
3.8	Axial turbine design procedure	18
3.9	Axial turbine stage design procedure	20
3.10	Fan of a GE90 engine	21
3.11	Fan radius progression for constant tip radius	22
3.12	Maximum angle definition for ducts	23
3.13	Various combustor types	24
3.14	Combustor design procedure	25
3.15	Nozzle of a CFM56 engine	27
3.16	Bleed air ducts	29
3.17	Compressor blades	30
3.18	Typical turbine blade profile	31
3.19	Disk approximation methods	32
4.1	CFM56-3B1 mass breakdown from MIME model	45
4.2	Trent 556 mass breakdown from MIME model	46
4.3	Orpheus Mk. 803 mass breakdown from MIME model	46
4.4	Overlay of CFM56-3B1 actual flow path with MIME estimate	47
4.5	Comparison between MIME flow path estimate and actual engine section for CFM56-3B1	48
4.6	Comparison between MIME flow path estimate and actual engine section for Trent 556	49
5.1	Design convergence process of the baseline A320-200 Initiator computation	51
5.2	Geometry of the baseline A320-200 Initiator computation	52
5.3	Mass distribution of the baseline A320-200 Initiator computation	52
5.4	Class 2 mass estimation according to Torenbeek of the baseline A320-200 Initiator computation	52
5.5	Engine total mass composition of the baseline A320-200 Initiator computation	53
5.6	Design convergence process of the A320-200 Initiator computation with MIME engine design calculations	53
5.7	Geometry of the A320-200 Initiator computation with MIME engine design calculations	54
5.8	Mass distribution of the A320-200 Initiator computation with MIME engine design calculations	54
5.9	Class 2 mass estimation according to Torenbeek of the A320-200 Initiator computation with MIME engine design calculations	54
5.10	Engine total mass composition of the A320-200 Initiator computation with MIME engine design calculations	55
5.11	Design convergence process of the A320-200 Initiator computation with MIME engine design and mass estimation calculations	55
5.12	Mass distribution of the A320-200 Initiator computation with MIME engine design and mass estimation calculations	56
5.13	Class 2 mass estimation according to Torenbeek of the A320-200 Initiator computation with MIME engine design and mass estimation calculations	56

5.14 Engine total mass composition of the A320-200 Initiator computation with MIME engine design and mass estimation calculations . . . . .	56
10.1 Class diagram for the AeroEngine class. . . . .	69
10.2 AeroEngine constructor workflow flowchart . . . . .	71
10.3 AeroEngine buildAeroEngine component creation workflow flowchart . . . . .	72
10.4 AeroEngine buildInlet inlet creation workflow flowchart . . . . .	73
10.5 order_engine_array components re-ordering function workflow flowchart . . . . .	73
10.6 AeroEngine onDesignCalc() on-design calculation workflow flowchart . . . . .	76
10.7 AeroEngine massCalc() mass calculation workflow flowchart . . . . .	77
10.8 Class diagram for the Axialcompressor class. . . . .	78
10.9 Axialcompressor buildstages stage (re-)creation workflow flowchart . . . . .	80
10.10 Axialcompressor aerocalc_Pini2() on-design calculation workflow flowchart . . . . .	82
10.11 Axialcompressor calcstages on-design stage calculation workflow flowchart . . . . .	84
10.12 Class diagram for the Axialturbine class. . . . .	85
10.13 Axialturbine aerocalc_Pini() on-design turbine calculation workflow flowchart . . . . .	86
10.14 Class diagram for the Axcompressorstage class. . . . .	88
10.15 Axcompressorstage buildstage() stage (re-)creation workflow flowchart . . . . .	90
10.16 Axcompressorstage rotorcalc_Pini_phi() on-design rotor calculation workflow flowchart . . . . .	91
10.17 Axcompressorstage statorcalc_Pini_phi() on-design stator calculation workflow flowchart . . . . .	93
10.18 Class diagram for the Axturbinestage class. . . . .	95
10.19 Class diagram for the Fan_MIME class. . . . .	100
10.20 Class diagram for the Inlet class. . . . .	102
10.21 Class diagram for the Combustor class. . . . .	104
10.22 Class diagram for the Nozzle class. . . . .	106
10.23 Class diagram for the Duct class. . . . .	108
10.24 Class diagram for the Blade class. . . . .	110
10.25 Class diagram for the Disk class. . . . .	112
10.26 Class diagram for the Flow class. . . . .	113
10.27 Class diagram for the Shaft class. . . . .	114
10.28 Class diagram for the Station class. . . . .	115
10.29 Class diagram for the Shaftstation class. . . . .	117
10.30 Class diagram for the InletGuideVane class. . . . .	117
10.31 Class diagram for the Frame class. . . . .	118
10.32 Class diagram for the Component superclass. . . . .	119
10.33 Class diagram for the Aerocomponent superclass. . . . .	120

# LIST OF TABLES

3.1	Characteristic values for various fuels . . . . .	24
4.1	Compressor blade (supersonic) verification results . . . . .	34
4.2	Compressor blade (subsonic) verification results . . . . .	35
4.3	Turbine blade verification results . . . . .	35
4.4	Disk verification results . . . . .	36
4.5	Inlet verification results . . . . .	36
4.6	Combustor verification results . . . . .	37
4.7	Axial compressor stage verification results . . . . .	38
4.8	Axial turbine stage verification results . . . . .	39
4.9	Axial compressor verification results . . . . .	40
4.10	Axial turbine verification results . . . . .	40
4.11	Testcase-engine inputs . . . . .	43
4.12	Results for the MIME model . . . . .	44
4.13	Additional results from MIME for CFM56-3B1 . . . . .	44
4.14	Additional results from MIME for Trent 556 . . . . .	45
4.15	Additional results from MIME for Orpheus Mk.803 . . . . .	45
4.16	Per-component results for CFM56 . . . . .	47
4.17	HPT and LPT breakdown results for CFM56 . . . . .	50
10.1	Contents of default GE90 and CFM56 options structures . . . . .	74
10.2	Material properties set by the Component-class <code>getdensity()</code> method . . . . .	120





# LISTS OF SYMBOLS AND ABBREVIATIONS

## Abbreviations

CFD	Computational Fluid Dynamics
IGV	Inlet Guide Vane
ISA	International Standard Atmosphere
MIME	Modular Initiator Model for Engines

**Definitions**

$\Lambda$	$1 - \frac{C_{\text{wavg}}}{U}$	—
$\lambda$	$\frac{\Delta C_w}{U}$	—
$\phi$	$\frac{V_{\text{ax}}}{U}$	—

**Greek Symbols**

$\alpha$	Absolute flow angle	rad
$\alpha_2$	Relative angle at stator (compressor) or rotor (turbine) inflow	rad
$\alpha_3$	Relative angle at stator (compressor) or rotor (turbine) outflow	rad
$\beta$	Relative flow angle	rad
$\beta_1$	Relative angle at rotor (compressor) or stator (turbine) inflow	rad
$\beta_2$	Relative angle at rotor (compressor) or stator (turbine) outflow	rad
$\delta$	Blade deviation angle	rad
$\Delta C_w$	Whirl velocity jump	$\frac{m}{s}$
$\Delta T$	Temperature deviation	°
$\Delta t_{comp}$	Compressor total temperature jump	K
$\Delta t_{stage}$	Stage total temperature jump	K
$\eta$	Efficiency	-
$\gamma$	Ratio of specific heats	-
$\gamma$	Stagger angle	rad
$\iota$	Incidence angle	rad
$\kappa_1$	Leading edge angle	rad
$\kappa_3$	Trailing edge angle	rad
$\Lambda$	Degree of reaction	-
$\lambda$	Work coefficient	-
$\mu$	Dynamic viscosity	Pa · s
$\nu$	Kinematic viscosity	$\frac{m^2}{s}$
$\omega$	Rotational velocity	$\frac{rad}{s}$
$\phi$	Camber angle	rad
$\phi$	Flow coefficient	-
$\Psi$	Zweifel coefficient	-
$\rho$	Density	$\frac{kg}{m^3}$
$\sigma_{bp}$	Blade pull stress	Pa
$\sigma_r$	Rotor solidity	-
$\sigma_s$	Stator solidity	-
$\sigma_y$	Yield stress	Pa
$\tau_y$	Yield shear stress	Pa

**Roman Symbols**

$a$	Sonic velocity	$\frac{m}{s}$
AR	Aspect ratio	–
BPR	Bypass Ratio	–
$c$	Blade chord	$m$
$c_{ax}$	Axial chord	$m$
$c_p$	Specific heat capacity at constant pressure	$\frac{J}{kg \cdot K}$
$C_{w1}$	Inflow absolute whirl velocity	$\frac{m}{s}$
$C_{w_{avg}}$	Average whirl velocity	$\frac{m}{s}$
$D_{mean}$	Mean diameter	$m$
DF	Diffusion factor	–
$DH_{crit}$	Critical De Haller number	–
$D_{hub}$	Hub diameter	$m$
$E$	Young's modulus	Pa
$E_{kin}$	Kinetic energy	J
$F$	Thrust	N
$F_g$	Gross thrust	N
$F_{comp}$	Component "thrust"	N
$h_{cruise}$	Cruise altitude	$m$
$h_{blade}$	Blade height	$m$
$\frac{H}{T}$	Hub-to-tip ratio	–
$K$	Blade volume factor	–
$l_{stage}$	Stage length	$m$
$M$	Mach number	–
$\dot{m}_f$	Fuel mass flow rate	$\frac{kg}{s}$
$M_{cruise}$	Cruise Mach number	–
$\dot{m}_{in}$	Inflow massflow rate	$\frac{kg}{s}$
$m_{hw}$	Hardware mass	kg
$N_{blades}$	Number of blades	–
$N_{stages}$	Number of stages	-
$o$	Throat opening	$m$
$p_0$	Ambient pressure	Pa
$P_{comp}$	Compressor power	W
$p_{max}$	Maximum total pressure	Pa

$PR_{\text{stage}}$	Stage total pressure ratio	-
$PR_{\text{comp}}$	Compressor total pressure ratio	-
$P_{\text{req}}$	Required power	W
$p_t$	Total pressure	Pa
$R$	Specific gas constant	$\frac{\text{J}}{\text{kg}\cdot\text{K}}$
$Re$	Reynolds number	-
RPM	Rotational velocity in revolutions per minute	$\frac{\text{rev}}{\text{min}}$
$r_{\text{tip}}$	Tip radius	m
$S$	Sutherland's constant	-
$s$	Blade spacing	$m$
$T$	Static temperature	K
$t$	Thickness	m
$T_{t_4}$	Turbine inlet total temperature	K
$T$	Torque	Nm
TR	Taper ratio	-
$t_{\text{res}}$	Residence time	s
$T_t$	Total temperature	K
$\vec{U}$	Rotor velocity vector	$\frac{\text{m}}{\text{s}}$
$V_{\text{axout}}$	Axial outflow velocity	$\frac{\text{m}}{\text{s}}$
$V_{\text{ax}}$	Axial velocity	$\frac{\text{m}}{\text{s}}$
$V_{\text{blade}}$	Blade volume	$\text{m}^3$
$\vec{V}$	Absolute velocity vector	$\frac{\text{m}}{\text{s}}$
$\vec{W}$	Relative velocity vector	$\frac{\text{m}}{\text{s}}$
$W_{w_1}$	Inflow relative whirl velocity	$\frac{\text{m}}{\text{s}}$





# I

## MODULAR INITIATOR MODELLING OF ENGINES



# 1

## INTRODUCTION

Turbine engines in various shapes and forms have formed the backbone of commercial aviation for the past six or seven decades, and are likely to remain a dominant factor in powering aircraft for at least the near future. Even though the use of turbine engines has remained a constant factor over the years, the sub-types and engine architectures have changed dramatically throughout history, and are expected to continue evolving in the decades to come. Revolutions have regularly taken place in the fields of engine architectures: from the original single-spool turbojets of Sir Frank Whittle and Hans-Joachim Pabst von Ohain in the late thirties, to the addition of fans and ever-increasing bypass flows, to turboprops and -shafts, up to the current developments in propfans, electric and hybrid or boosted engine technologies [2] involving generators and electric motors.

It can thus be seen that for every application and every era, there is a different engine type, architecture or lay-out, each with their own characteristics. This also has an impact on the aircraft design the engine is intended for. For example, increasing bypass-ratios increase engine diameters, impacting landing gear length and wing lay-out, small differences in fuel consumption have a large impact on aircraft mass and thus on the sizing of lifting surfaces, and differences in mass of the engine itself impact amongst others wing structure design due to changed bending relief loads.

In determining what the impact of engine design decisions is on the complete aircraft design, it is essential that different engine characteristics can be compared on an equal basis, and as such are generated from a similar process. To facilitate this, a flexible system is required, which can generate engine characteristics for a large number of different configurations on the same basis, in order to allow for an equal comparison.

### 1.1. PROBLEM STATEMENT

Different aircraft designs have different characteristics. These are determined by many different design choices, which are often highly interconnected. This is the case on both a complete-aircraft level as well as on a component-level. In order to structure the way these choices are made, and provide guidance to the aircraft design process, several authors have published conceptual aircraft design methods, such as [3], [4] and [5]. These are generally initially focussed on estimation of lift, drag, aircraft mass and fuel capacity, after which the focus shifts to the positioning and mass estimation of major components and (sub-)systems. These estimations are often made based on empirical relations and rules of thumb.

While most of these methods can be used as a cookbook-like recipe, their methods can also be automated and interconnected in an iterative fashion. Amongst others, this is how the Aircraft Design Initiator [6–8] of Delft University of Technology (TU Delft) its Flight Performance and Propulsion (FPP) group started out. This program allows for the automated conceptual design of an aircraft, based on several top-level requirements as well as some user-defined design decisions. The former includes requirements such as range, cruise conditions and payload, while the latter includes parameters such as wing, tail and cabin configurations as well as engine number, type and bypass ratio. The Initiator is currently often used to compare different aircraft concepts and the impact of and interaction between design choices and new techniques, as it is relatively easy to generate different aircraft designs with the same requirements, but different (combinations of) design choices in an automated fashion.

The engines of an aircraft can have a large impact on the total aircraft design, as they dictate or strongly

influence such parameters as fuel consumption (and thereby required fuel mass), wing bending loads, flight performance and of course simply the aircraft empty mass. As the Initiator is often used to evaluate the impact of novel techniques, and as there are many current developments in the field of novel propulsion concept and engines, it is vital that the effects of these new techniques are accurately modelled.

The Initiator estimates several characteristics of an engine: its cruise performance (thrust and fuel consumption), its mass and its climb performance. Currently however, the Initiator is only capable of modelling the cruise performance of two types of engines: either a twin-spool axial-flow turbofan or a twin-spool axial-flow turboprop engine, which in the case of the turboprop is just a turbofan with the fan replaced by a propeller. Furthermore, these engines are mostly hardcoded in terms of engine architecture and specifications, with the effectively only user-controlled parameters being the bypass-ratio BPR and turbine inlet total temperature  $T_{t_4}$ . Using this system, it is impossible to simulate for example hybrid engines, or any other configuration, such as a geared turbofan, propfan, turboshaft, et cetera. Climb performance is estimated using an almost completely hardcoded three-spool turbofan (only the turbine inlet temperature is carried over), regardless of the engine type. Engine mass is estimated from a database interpolation, based on cruise thrust, over either a database of turbofan or turboprop engines. This again illustrates the limited flexibility and inapplicability of the current method to novel engine configurations.

A possible means of improving this is to look at the engine as a collection of interconnected (and user-selectable) parts, instead of a single unit with only one or two selectable parameters. This “component-based” approach is both more physically accurate<sup>1</sup>, as well as more flexible, as such a method could allow for the evaluation of the effects and interactions of all components with each other, instead of making assumptions on a pre-defined design. Ideally, this component-based calculation is also used for estimation of the engine mass, as the sum of the mass of all parts, as well as the off-design (climb) performance. This report therefore focusses on the creation of such a component-based method, specifically for use within the Initiator.

This method is then used to find what benefits such a method could bring to the Initiator, answering the main research question “*How can we size engines in terms of geometry, mass and performance for different engine architectures based on the same Top-Level Engine Requirements, technology level parameters and design strategy?*”. This research question can be subdivided into several subquestions, which will be answered first, in order to generate an answer to the main question. These include

1. *How do the estimations of mass, thrust and specific fuel consumption of the Modular Initiator Model for Engines compare to the actual values for typical engines?*
2. *How do the estimations of additional engine characteristics such as stage or blade numbers, shaft speeds and geometry compare to actual values for typical engines?*
3. *What implications can be observed when using a modular engine model within the full aircraft design process?*
4. *How can this modular engine design method be translated into a software environment?*

## 1.2. THESIS PROGRAMMING OBJECTIVES

As stated, the objective of this thesis is to create a component-based engine model for use within the Initiator framework. This model should be modular, in order to allow flexibility, but should also be able to serve as a drop-in replacement for the currently used engine models.<sup>2</sup> In order to do this, the following characteristics of the engine should be determined:

**Cruise performance** The cruise (on-design) performance comprises the thrust and specific fuel consumption at cruise altitude and Mach number, and serves as the design condition. Within the Initiator, this is used as input for, amongst others, the range calculations and to achieve force balance.

**Engine mass** The engine dry mass is the mass of all the engine components, excluding for example any effects of (trapped) liquids. Within the Initiator, this mass is used for determining wing loads, as well as the entire aircraft mass and center-of-gravity locations.

<sup>1</sup>For an arbitrary engine

<sup>2</sup>As the more flexible engine design approach requires more information about the engine lay-out from the user than is currently provided within the Initiator, this implies that it is essential that a default condition is in place to fill in these information gaps if present.

Therefore, calculation functions for each of these parameters should be written, with in- and outputs defined in such a way as to require minimal modifications to the code of the Initiator itself. This should allow drop-in replacement of the current methods, in order to have a minimal impact on any current Initiator implementations and developments.

Ideally, also the off-design performance (e.g. during take-off and climb) would also be modelled by this method. Even though the system was set up with this in mind, due to time constraints these calculations have not been implemented yet.

As can be seen from the previously given list, cruise performance is the main engine design driver and should therefore, together with the user-specified information on engine lay-out, be used to size the engine components. The mass estimation and off-design performance of the engine are then determined based on these component specifications.

The final method should be flexible in terms of engine architecture and at least comparable in terms of accuracy, while method speed is expected to suffer, however this should not become excessive.

### 1.3. REPORT OUTLINE

Part I of this report focusses on the methodologies and theoretical basis of the followed procedures and used equations for the Modular Initiator Model for Engines.<sup>3</sup> Chapter 2 describes the choices and reasoning behind the chosen structure of the program and its interfaces.

Chapter 3 describes the design methods and equations used for designing the various components and sub-components currently implemented in the Modular Initiator Model for Engines (MIME).

For a code to be of any use, it needs to be shown that it runs without issues and that its outcomes both make sense and are valid. The methods used to investigate this, as well as their results, are described in Chapter 4. Chapter 5 describes a comparison case study for both the current Initiator engine model and MIME fully integrated into the Initiator.

Results are presented on the performance of the code in general in Chapter 6. Conclusions are then drawn from these results on this particular implementation of the modular modelling approach, the desirability of this approach for the purposes of the Initiator and in general.

Finally, as a thesis will never be a closed and perfect story, ending “happily ever after”, but always will leave something to be desired or improved, Chapter 7 provides a list of recommendations for further research. This includes both recommendations regarding improvements to the currently implemented components, codes and methods, as well as recommendations for further capability extensions of the system.

---

<sup>3</sup>A detailed description of all involved classes, methods and workflows is presented in Part II.



# 2

## MODULAR INITIATOR MODEL OF ENGINES MODELLING APPROACH

Before actually coding any program, or even starting to work out the exact equations for a problem, it is important to first determine which approach will be taken to get to a solution. This dictates the structure of the program, as the chosen workflows have a large impact on the order of calculations, and thereby mostly determining which variables are taken as inputs, and which are left to be calculated. Therefore, the overall structure and approach of the Modular Initiator Model for Engines (MIME) is laid out in this chapter.

First, Section 2.1 describes which structure is chosen for representing an engine, and which considerations played a part in making this decision. Next, the chosen general procedures for obtain an engine design, on-design performance and mass estimation are elaborated upon in Sections 2.2 and 2.3.

### 2.1. OVERALL MODEL STRUCTURE

From the outset, the goal of MIME has been to provide a modular, expandable and flexible model of any turbine engine, however in principle it should theoretically also be expandable to for example ramjet engines. In order to be able to model a wide range of engines, several different approaches could be used. As a very coarse approximation, simple database calls or response surfaces could be used to estimate the properties of a complete engine in one go, such as [9]. However, as shown by [10], these “complete-engine” methods have relatively poor accuracy and, although they are fast, do not incorporate any detailed information on the engine layout, which for the purposes of MIME is explicitly required in order to capture the effects of design decisions on propulsion details. On the other hand, more detailed component-based methods, often coupled to CFD tools, are in use with engine manufacturers, including MTU [11], Pratt & Whitney [12] and Rolls-Royce [13], however these tools are much more detailed than required for this purpose, as well as only for in-house use. The exception is the WATE weight estimation tool from NASA [14], however this is again coupled to an external gas path design routine, which is too detailed for the current purposes.

In the end, it was decided to build MIME as a component-based model, with each main component in the engine having a separate model, which can be coupled arbitrarily to form a complete engine. As the Initiator is programmed almost solely in MATLAB<sup>1</sup>, the decision was made to also create MIME as a MATLAB code. Due to the modular nature of the program, it is set up as an object-oriented code.

Each type of component has its own class. These classes are constructed in such a way as to have a mostly unified method for calling them; each component class has a single function which is called by the AeroEngine top-level object when evaluated for a certain function (e.g. mass calculation, on-design performance estimation, et cetera). As all (through-flow) components tend to depend on at least an inflow gas state, shaft RPM and/or power consumption, or have to write outflow conditions or power requirements, the appropriate associated component objects are usually fed into these functions as well.

Gas states before and after components are stored in Station objects, which are contained in Flow objects. The former a.o. allows for the flow conditions at a certain station to always be consistent, and also allows for easy storage and retrieval of both gas conditions at a specific location as well as additional information. The containment with the Flow allows for easy separation and ordering of the various Stations,

<sup>1</sup>As well as considering the author's proficiencies in programming languages



using a division which is also physically present, and also allows for additional calculations to be performed, such as for example convergence, division and sourcing of bleed air.

For this initial version of MIME, only the basic components of a general turbofan engine are included. These include classes for the modelling of axial compressors and turbines, fans, inlets, nozzles and simple combustors, as well as shafts and engine frames. These components are sufficient to model turbofans, turbojets and theoretically also straight jets.<sup>2</sup> The components currently making up MIME and the way in which they are nested within each other are roughly given in Figure 2.1.

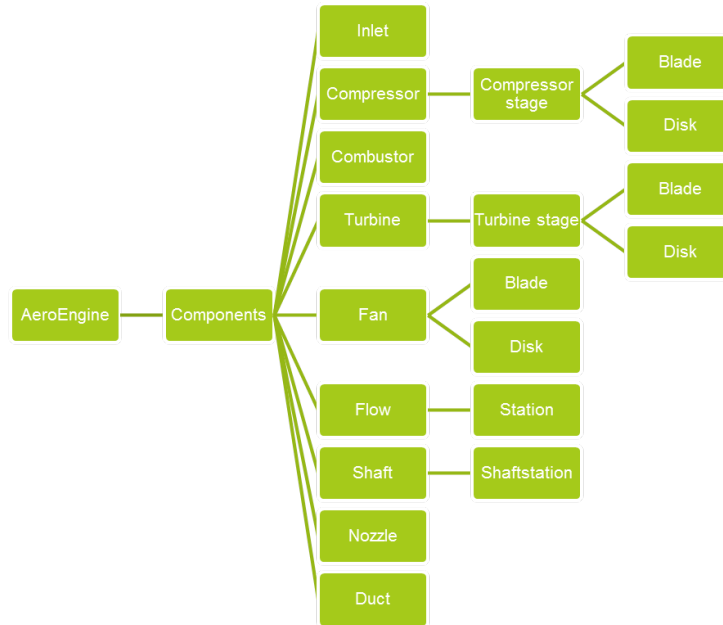


Figure 2.1: Engine build-up schematic

The top-level `AeroEngine` object represents the engine, containing all the components, but also serves as the single point of contact for the Initiator. Upon creation of an engine, it handles the conversion of the user specifications from the Initiator input, as well as supplements any possibly incomplete data. When the engine performance or mass is needed, it also handles these calculations, based on its components.

## 2.2. COMPONENT SIZING AND ON-DESIGN PERFORMANCE APPROACH

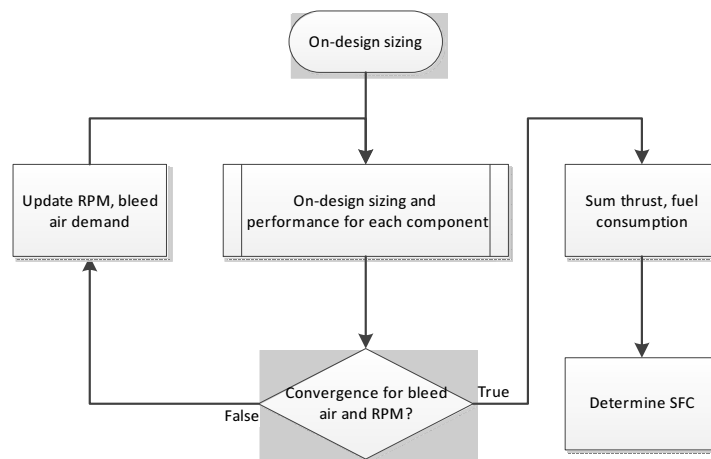


Figure 2.2: Engine design and sizing workflow

<sup>2</sup>As most pure jets do not function below  $M = 1$ , while the current versions of both the inlet and nozzle models are not suited for supersonic flows, this is not yet a practical implementation.

The design and on-design performance estimation is based on iterative calculation of all engine components, with each (aerodynamic) component taking the outflow conditions of its preceding component and processing them to produce its own output, similar to the way a particle flowing through the engine would encounter each component in turn. All pre- and post-component flow conditions are stored in `Station` objects within `Flow`-class objects, modelling the flowpaths through the engine. If requirements are imposed on a preceding component, these are taken into account during the next iteration, but are also already compensated for in the current iteration, in order to never yield an inconsistent system. Iteration is continued until both the shaft speeds and bleed air supplies have converged, as indicated schematically in Figure 2.2. This method of modelling was chosen due to the fact that this both guarantees that the inflow conditions of the following component match the outflow conditions of the preceding component, as well as being relatively simple to implement for a large possible variation in the number and type of components.

An alternative method would have been a scheme with a.o. all inflow parameters as additional control variables and the errors between the in- and outflow variables of all components as additional minimisation objectives. This method, although relatively robust and more suited for parallel computing, was deemed to add too much complexity, especially considering the required flexibility, leading to variable-size design vectors and other potential complications.

As there may be multiple assumptions used in the design of a certain component which are updated during the calculation of later components, calculation of the entire components array is repeated until these assumptions have converged. One of the clearest examples of this process is the bleed airflow supply and demand. The demand is only fully established after all the turbines have been fully calculated, however the bleed air required needs to be sourced from the compressor, which was already calculated earlier. For a non-negligible difference between bleed air supply and demand, a re-iteration is thus required. Another point where this influence is present is the shaft, where the rotational speed has a value based on the maximum velocity limits of the various components.

For the on-design performance calculation, the component requirements and inflow conditions are used to design and size the parameters of the individual components. For example, for a nozzle, the nozzle areas and diameters are determined based on the design condition incoming pressure, massflow rate and the design condition ambient conditions. Within the on-design calculation, the engine design is thus also being determined and therefore is flexible within this calculation stage! This truly reflects the iterative nature of the actual engine design process.

### 2.3. MASS ESTIMATION APPROACH

While the saying may suggest differently, the MIME approach to estimating engine mass is to simply compute the masses of all individual components and add them up, to obtain the whole mass as the sum of its part masses. As also concluded by [10], this seems the most flexible and reliable method for estimating the masses of different engines. While other options such as database extrapolation are very accurate for engines highly similar to those in the database, their accuracy plummets very fast with deviating designs. As novel propulsion concepts often do not have a large amount of pre-existing highly similar reference engines, the database method is unsuitable. Also, creating and selecting separate reference engine databases for each specific engine architecture is not really feasible.

The mass estimation methods are mostly derived from the WATE method as described by [14] and [15], with some methods also partially inspired by [16]. Most of these specific methods are described later in Chapter 3.



# 3

## THEORETICAL CONSIDERATIONS FOR ENGINE COMPONENTS

While Chapter 2 described the general structure of the program, the overall workflow concepts and the considerations guiding these, this can be seen mostly separate from the internal workflows and computational strategies of the individual components, except for the interfaces.<sup>1</sup>

This chapter will treat the used design principles and equations for the various components which are currently supported. These include the turbomachinery components such as the axial compressor and its stages in Section 3.1, the axial turbine and associated stages in Section 3.2 and the fan in Section 3.3. Throughflow components are also described, including the duct in Section 3.4, the combustor in Section 3.5, the inlet in Section 3.6 and the nozzle in Section 3.7. The governing equations and design principles of data transport components will be elaborated upon next, such as the shaft in Section 3.8 and the Flow class in Section 3.9. Finally, some of the more lower level components are included, such as the engine frames in Section 3.10, the turbomachinery blades in Section 3.11.1, the turbomachinery rotor disks in Section 3.11.2 and the inlet guide vanes in Section 3.11.3.

### 3.1. AXIAL COMPRESSOR

The axial compressor has been one of the core components of almost any turbofan or turbojet engine for the last decades, as it allows for large massflows and is relatively easy to stack, as can be seen in Figure 3.1. Even though it tends to allow a smaller pressure increase compared to a radial compressor in a single stage, it is much easier to stage. It therefore almost exclusively appears as a multi-stage component in modern engines, with each stage (consisting of a rotor and a stator) functioning in a similar manner.

#### 3.1.1. ON-DESIGN COMPUTATIONS

From this, it can already be deduced that the number of stages  $N_{\text{stage}}$  in a compressor is highly dependent on the required compressor total pressure ratio  $PR_{\text{comp}}$ . Therefore, it is determined during the design procedure (which is depicted in Figure 3.2) by dividing the total required energy increase  $\Delta T_{\text{tcomp}}$  by the maximum generated for a single stage  $\Delta T_{\text{tstage}}$ , as suggested by [17, Sec. 5.7]. Other possible approaches include specifying the number of stages and the overall compressor pressure ratio to obtain the required stage energy increase, or specifying the number of stages with the maximum stage energy increment, leading to the overall compressor total pressure ratio being obtained. As the compressor total pressure ratio is a key defining feature, while the stage energy increase also has its limits, these latter two methods are deemed unsuitable for use within MIME.

An axial compressor functions by adding energy to the flow over a rotor, in the form of an increase in (absolute) velocity. This velocity increase is then mostly cancelled out over a stator, which slows down the flow, increasing the pressure. Due to this adverse pressure gradient, it is important that the velocity differences

<sup>1</sup>Even though there is virtually no direct interaction between the top-level object and the individual components except for the chosen interface functions and variable, the determination of these interfaces does determine the available variables for computation, which of course does heavily influence the way the individual components function internally.

<sup>2</sup>Source: <https://commons.wikimedia.org/wiki/File:Axialkompressor.jpg>, retrieved 2018-06-09

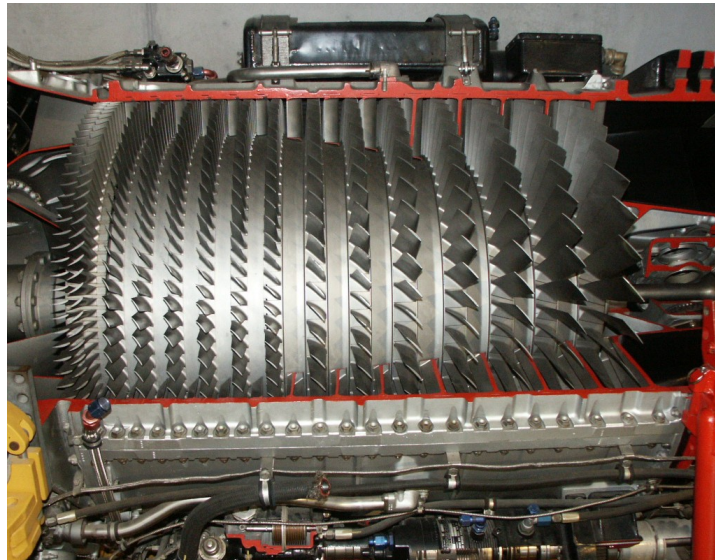


Figure 3.1: 14-stage axial compressor of a Wright J65 engine.<sup>2</sup> Note: flow from right to left.

over compressor blades are not excessive, as otherwise flow separation and blade stall might occur. This is one of the limiting factors in designing the compressor, and is characterised by the De Haller criterion, which states that the velocity ratio over a blade relative to that same blade should not drop below  $DH_{crit} = 0.72$ .<sup>3</sup> [18, p. 558]

As the addition of energy to the flow to increase its pressure occurs through a mechanism involving velocities, as described earlier, this implies that velocities and their relations play a major part in the design of a compressor. To more easily define the relations between the various velocities and components at a certain location, so-called velocity triangles are used, which show the relations between the absolute velocity vector  $\vec{V}$ , the rotor velocity vector  $\vec{U}$  and the velocity vector relative to the rotor  $\vec{W} = \vec{V} - \vec{U}$  at a given location, as shown in Figure 3.3. These velocity triangles can be easily scaled, leading to several non-dimensional coefficients being used to describe them, including the flow coefficient  $\phi$ , which is defined as  $\phi = \frac{V_{ax}}{U}$ , the work coefficient  $\lambda = \frac{\Delta C_w}{U}$  and the degree of reaction  $\Lambda = 1 - \frac{C_{w,avg}}{U}$ . These coefficients completely define the absolute and relative flow angles  $\alpha$  and  $\beta$  throughout the compressor.

Of these characteristic coefficients, the work coefficient  $\lambda$  is selected as the control variable, as this variable has a direct relation with the energy increase in the flow. The flow coefficient  $\phi$  is assumed to be bounded between  $0.3 \leq \phi \leq 0.9$ <sup>4</sup> [18, p. 628], and is initially assumed to be determined by the local flow conditions, only being adjusted when needed. If the rotor inflow velocities are not specified, the degree of reaction is initially assumed as  $\Lambda = 0.5$ , yielding a so-called Parson's stage, where the increase in static temperature is equally distributed over the rotor and stator. For a specified rotor inflow velocity,  $\Lambda$  may be adjusted to still be able to ensure consistency. For a stage with a known flow coefficient  $\phi$ , degree of reaction  $\Lambda$  and work coefficient  $\lambda$ , the required absolute rotor inflow angle  $\alpha_1$  is calculated using

$$\tan \alpha_1 = \frac{-\Lambda - \frac{\lambda}{4} + 1}{\phi} \quad (3.1)$$

For a specified inflow angle,  $\Lambda$  is found by rewriting this equation. From the absolute inflow angle  $\alpha_1$  and the (inflow<sup>5</sup>) axial velocity  $V_{ax}$ , the inflow absolute whirl velocity component  $C_{w1}$  can be computed, which is related to the inflow relative whirl velocity component  $W_{w1}$  according to

$$W_{w1} = C_{w1} - U \quad (3.2)$$

From this, the relative inflow angle  $\beta_1$  can be computed, as can the inflow absolute and relative velocities  $V_1$

<sup>3</sup>However, [17, p. 197] states that values as low as  $DH_{crit} = 0.69$  may still be acceptable.

<sup>4</sup>[19, Slide 4] gives bounds of  $0.4 \leq \phi \leq 1.1$ , however in personal contact, Dr. Pini indicated a trend towards lower flow coefficients, leading to the selection of the lower set of bounds.

<sup>5</sup>For now, this method assumes a constant axial velocity over each stage.

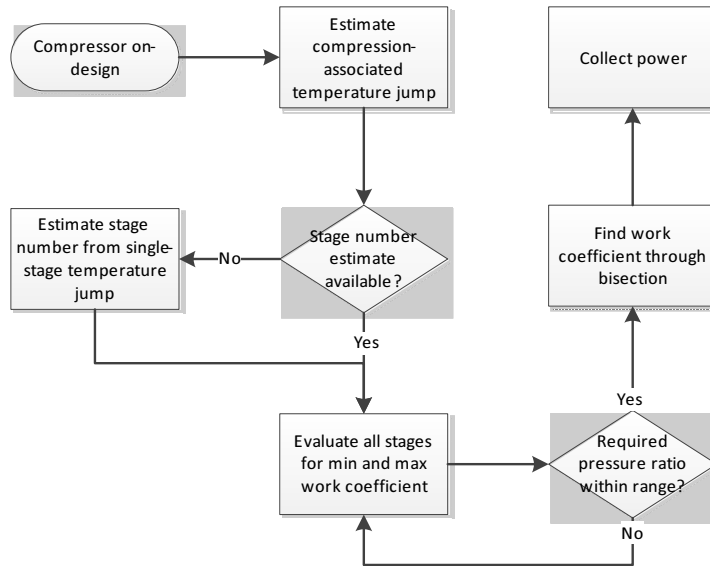


Figure 3.2: Axial compressor design procedure

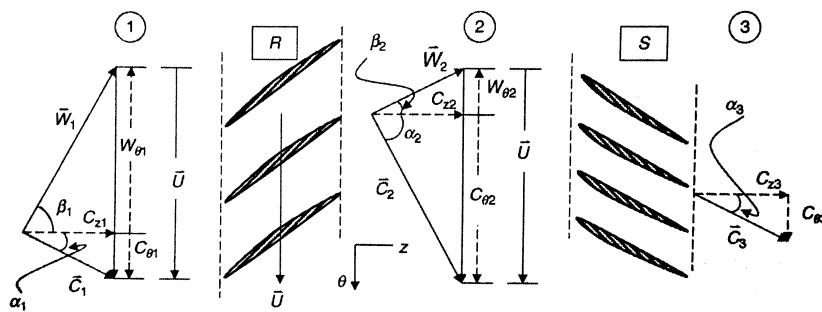


Figure 3.3: Velocity triangles for a typical compressor stage [18, Fig. 8.9]

and  $W_1$ . The post-rotor absolute whirl velocity  $C_{w2}$  is computed from  $C_{w1}$ ,  $\lambda$  and  $U$  according to

$$\begin{aligned} C_{w2} &= C_{w1} + \Delta C_w \\ &= C_{w1} + \lambda \cdot U \end{aligned} \quad (3.3)$$

Assuming a constant axial velocity over the stage, this then again allows for calculation of the post-rotor absolute and relative angles and velocities  $\alpha_2$ ,  $\beta_2$ ,  $V_2$  and  $W_2$ . The stator outflow angle  $\alpha_3$  of a given stage is finally assumed to be completely specified by (and due to the stage stacking equal to) the required inflow angle  $\alpha_1$  of the following stage.<sup>6</sup> This stage calculation procedure is depicted schematically in Figures 3.4 and 3.5.

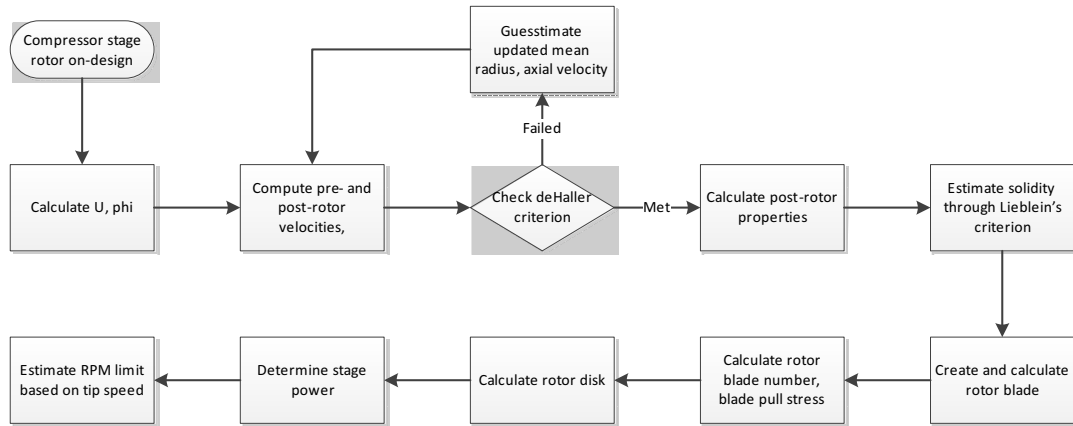


Figure 3.4: Axial compressor stage rotor design procedure

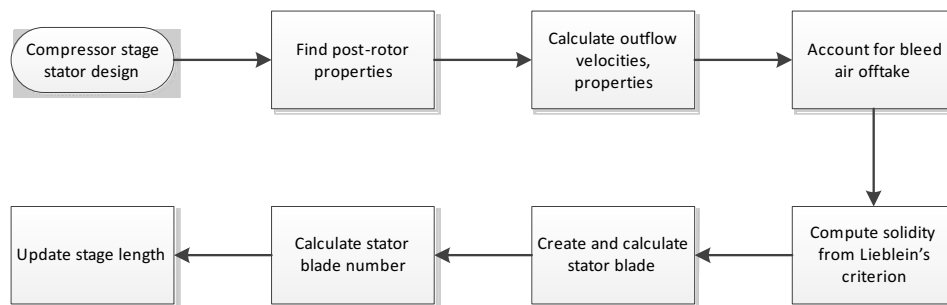


Figure 3.5: Axial compressor stage stator design procedure

As stated earlier, the deceleration over a rotor should not be excessive, in order to avoid the occurrence of flow separation, which would cause unnecessary losses, as well as possibly damaging vibrations. Therefore, the DeHaller number  $DH$  of the rotor is computed using

$$DH = \frac{W_2}{W_1} \quad (3.4)$$

which can then be checked against the critical value of  $DH_{crit} = 0.72$ . [18, p. 558]

If this criterion is not met, the ratio between these relative velocities must thus be adjusted, which can be done by altering any of the non-dimensional coefficients mentioned earlier:  $\lambda$ ,  $\Lambda$  and  $\phi$ . As  $\lambda$  was selected earlier as the control variable<sup>7</sup> it is illogical to try and modify it to just try to meet the DeHaller criterion. Trying to vary the flow coefficient  $\phi$  is more convenient, which can be done through its direct coupling to the axial velocity  $V_{ax}$  and rotor velocity  $U$ , the latter of which in turn is a direct linear function of the mean radius  $r_{mean}$  and the RPM. While the latter of these is specified by the shaft, the former is relatively freely adjustable.

<sup>6</sup>In case this is the last stage of the compressor, it may also be set to the required inflow angle of the following component. If no required outflow angle is specified by either the following stage or component, the stage outflow angle  $\alpha_3$  is simply assumed to be equal to the inflow angle of that stage  $\alpha_1$ .

<sup>7</sup>As it is defined as the control variable for the entire compressor,  $\lambda$  is assumed to be varied on the compressor level and therefore kept constant for all stages during the same calculation.

Assuming the axial flow velocity  $V_{ax}$  remains unchanged, making a small change to the mean radius will yield a new  $\phi$  (and in case of a fixed inflow angle  $\alpha_1$  also a new  $\Lambda$ ), yielding a new DeHaller number.

Using purely these relations, there is no guarantee that the outcome will yield a positive hub radius, which is a requirement for any annular component. In order to remedy this when it occurs, the axial velocity is increased slightly. This should decrease the required flow area, and therefore increase the hub radius as the annulus is sized around the mean radius. This way, not only is the hub radius kept positive, but also sufficiently large to allow the drive shaft to fit through.

Thermodynamically, the post-rotor total temperature  $T_{t2}$  is calculated from the various local whirl velocities according to [18, Eq. 8.23]

$$T_{t2} = T_{t1} + U \cdot \frac{C_{w2} - C_{w1}}{c_p} \quad (3.5)$$

This then allows for the computation of a rotor total temperature ratio, which can be translated into a total pressure ratio, leading to a post-rotor total pressure  $p_{t2}$ , based on the stage polytropic efficiency  $\eta$ , according to [17, Eq. 2.15]

$$p_{t2} = p_{t1} + \left( \frac{T_{t2}}{T_{t1}} \right)^{\frac{\eta \gamma}{\gamma - 1}} \quad (3.6)$$

These total properties are assumed to remain constant over the stator. Static properties are computed from the total properties and local velocities at each station.

In order to say something about the number and spacing of the rotor and stator blades, the so-called Lieblein's criterion [20, Slide 14] is used to determine the solidity of rotor and stator  $\sigma_r$  and  $\sigma_s$  based on the flow angles relative to the blade and the so-called diffusion factor DF, according to

$$\begin{aligned} \sigma_r &= \left| \cos \beta_1 \cdot \frac{\tan \beta_1 - \tan \beta_2}{2 \cdot \left( DF - 1 + \frac{\cos \beta_1}{\cos \beta_2} \right)} \right| \\ \sigma_s &= \left| \cos \alpha_2 \cdot \frac{\tan \alpha_2 - \tan \alpha_3}{2 \cdot \left( DF - 1 + \frac{\cos \alpha_2}{\cos \alpha_3} \right)} \right| \end{aligned} \quad (3.7)$$

For calculating the blade chord length, the blade Reynolds number is assumed to have a typical value of  $Re = 0.5 \cdot 10^6$  [18, Pg. 628], which is then used to determine the chord length. For this, however the kinematic viscosity  $\nu$  at the blade location is also required, which is calculated using [21, Eq. 2.81]

$$\nu = \frac{\mu_0 \left( \frac{T}{T_0} \right)^{\frac{3}{2}} \cdot \frac{T_0 + S}{T + S}}{\rho} \quad (3.8)$$

where  $T$  is the blade row inflow static temperature,  $\rho$  is the blade row inflow static density,  $S = 110$  is Sutherland's constant for air,  $T_0 = 288.16$  is the reference temperature for air and  $\mu_0 = 1.7894 \cdot 10^{-6}$  is the reference viscosity. Further details about the calculation of blade parameters are given in Section 3.11.1.

When the rotor blade dimensions and mass are known, the pulling stresses they exert on the rotor disk  $\sigma_{bp}$  can be estimated using [15, Eq. 7]

$$\sigma_{bp} = \frac{12 \cdot \rho_{blade} \cdot \left( \frac{RPM}{60} \cdot r_{tip} \right)^2}{g \cdot TR} \cdot \left( \frac{1 - \left( \frac{H}{T} \right)^2}{2} + \frac{TR}{12} \cdot \left( 1 - \frac{H}{T} \right) \cdot \left( 1 + 3 \cdot \frac{H}{T} \right) \right) \quad (3.9)$$

This value can then be used to size the rotor disk, more details on which are provided in Section 3.11.2.

The power required by a compressor  $P_{comp}$  is determined from the observed total temperature jump for each stage. As the total temperature is assumed to only change over the rotor, and assuming that any tapped bleed air is taken off after the compression is fully complete, it is computed using

$$P_{comp} = c_p \cdot \dot{m}_1 \cdot (T_{t2} - T_{t1}) \quad (3.10)$$

Stage length is estimated by summing the lengths of the rotor and stator blade rows (calculated as their axial chords  $c_{ax}$  multiplied by a spacing factor), as well as the length of any required rows of IGV's and/or inflow correction ducts.

Finally, the compressor limit RPM is estimated based on a maximum tip Mach number, according to

$$RPM_{lim} = 60 \cdot \frac{\sqrt{(M_{lim} \cdot a)^2 - V_{ax}^2}}{2 \cdot \pi \cdot r_{tip}} \quad (3.11)$$



This assumption is however not perfect, as this does not take into account the now-modified flow angle, which in response is likely to induce variations in the axial velocity. However, for lack of a better method, this equation is used, completing the set of equations used in the compressor design and on-design performance estimation calculations.

### 3.1.2. MASS ESTIMATION CALCULATIONS

The estimation of the mass of a compressor is based on the methods used by [15] and [16] (as illustrated in Figure 3.6), estimating the mass of a compressor as the sum of the masses of its stages. In turn, each stage is again split up into various components, such as the rotor and stator blades, rotor disk, casings, as well as as possibly required IGV's and/or an inflow correction duct. Also, a mass contribution for the inter-stage coupling hardware is taken into account.

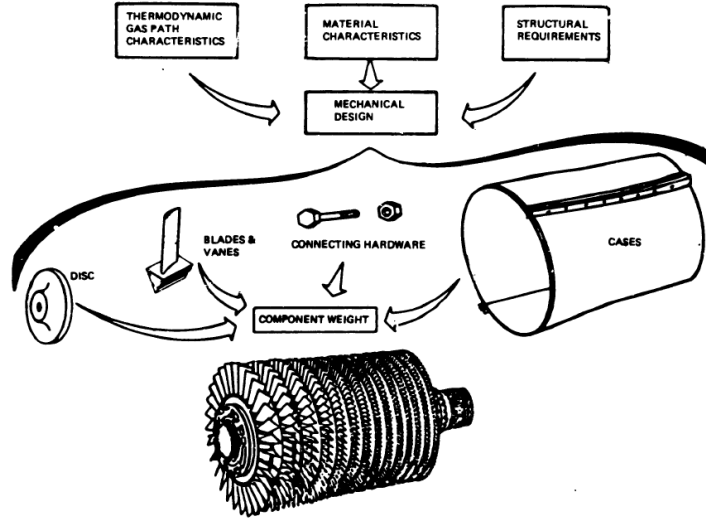


Figure 3.6: Compressor mass estimation components [15, Fig. 6]

The mass contributions of the rotor and stator blades are found by taking the masses of the individual blades (see Section 3.11.1 for more details on how these are obtained) and multiplying this with the number of rotor and stator blades present in the stage. The mass of the rotor disk is also dependent on its design, with its calculation being detailed in Section 3.11.2. This also goes for the IGV's and ducts.

The coupling hardware mass  $m_{hw}$ , which are the bolts, nuts and spacers coupling the stage to the next, is modelled as a 0.075" (= 1.905 [mm]) thick ring, located at 75% of the hub radius, according to [15, Eq. 8]

$$m_{hw} = 0.75 \cdot \pi \cdot D_{hub} \cdot 0.001905 \cdot l_{stage} \cdot \rho_{mat} \quad (3.12)$$

The casing is assumed to be a straight thin-walled cylinder, the thickness of which is sized based on three different criteria: manufacturability, pressure containment and blade containment. In order to keep the casing manufacturable and handleable, a minimum thickness of 1 [mm] is primarily assumed. Next, it is checked whether the pressure containment criterion yields a larger thickness  $t_{pres}$ , based on the standard boiler equation

$$t_{pres} = \frac{p_{max} \cdot D_{max}}{2 \cdot \sigma_y} \quad (3.13)$$

The final criterion is formed by the requirement for blade containment, which according to [22, 810] must be ensured for fan, compressor and turbine blades, yielding a minimum containment thickness  $t_{cont}$ . For this, first the kinetic energy in a loose blade  $E_{kin}$  is estimated, assuming this as the kinetic energy of a particle at the mean blade radius with the blade mass and a purely linear velocity

$$E_{kin} = \frac{1}{2} \cdot m_{blade} \cdot \left( 2 \cdot \pi \cdot r_m \cdot \frac{RPM}{60} \right)^2 \quad (3.14)$$

Next, this kinetic energy  $E_{kin}$  is used to determine the required thickness for ensuring blade containment  $t_{cont}$  according to [16, Eq. 3.83]

$$t_{cont} = \frac{0.4 \cdot E_{kin} \cdot E}{\sigma_y^2 \cdot h_{blade} \cdot c_{blade}} \quad (3.15)$$

where  $E$  is the Young's modulus of the casing material (assumed equal to the blade material),  $\sigma_y$  is the casing material yield strength,  $h_{\text{blade}}$  is the blade height and  $c_{\text{blade}}$  is the blade chord.

For a high blade kinetic energy (either due to a high RPM, large mean radius and/or a high blade mass), this criterion may yield very thick casings. In order to avoid this, it is also possible to divide the casing into two parts, creating a blade-material casing with a thickness purely sized for the pressure criterion, as well as a kevlar wrapping around it, which takes care of the blade containment criterion. Its thickness is also determined using Equation 3.15, however now with  $E$  and  $\sigma_y$  as the values for kevlar.

With the thicknesses determined, the casing mass  $m_{\text{case}}$  itself is calculated separately for the stator and rotor cases, according to

$$\begin{aligned} m_{\text{case}_r} &= \pi \cdot D_{\text{max}} \cdot l_{\text{blade}_r} \cdot t_{\text{case}_r} \cdot \rho \\ m_{\text{case}_s} &= \pi \cdot D_{\text{max}} \cdot l_{\text{blade}_s} \cdot t_{\text{case}_s} \cdot \rho \end{aligned} \quad (3.16)$$

where  $l_{\text{blade}}$  is the (axial) length of the rotor or stator blade row,  $t_{\text{case}}$  is the maximum of the found required thicknesses (handling and pressure for the stator, handling, pressure and containment for the rotor) and  $\rho$  is the density of the case material.

The stage mass  $m_{\text{stage}}$  is then assumed according to

$$m_{\text{stage}} = N_{\text{blade}_s} \cdot m_{\text{blade}_s} + N_{\text{blade}_r} \cdot m_{\text{blade}_r} + m_{\text{disk}} + m_{\text{hw}} + m_{\text{case}_s} + m_{\text{case}_r} + m_{\text{IGV}} + m_{\text{ductin}} \quad (3.17)$$

with the total sum of all stage masses being assumed as the mass of the complete compressor.

## 3.2. AXIAL TURBINE

Like the axial compressor, the axial turbine is a core part of most turbine engines, having similar advantages over its radial counterpart as mentioned for the compressor. It is for most turbine engines the sole method for generating the power required by the compressors and fan. An example is shown in Figure 3.7.

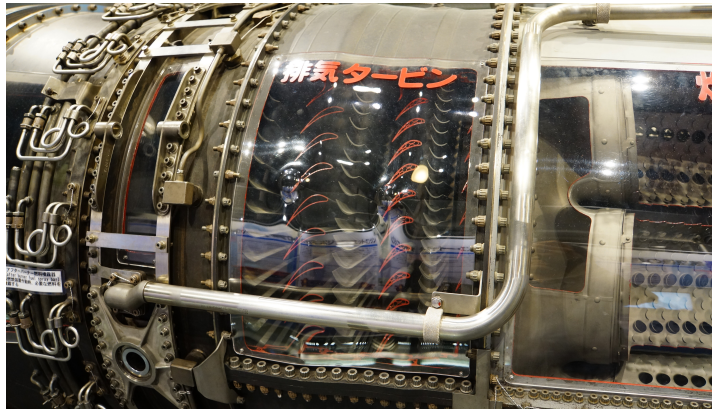


Figure 3.7: 3-stage axial turbine of a J79 engine<sup>8</sup>

An axial turbine consists of one or more stages, each consisting of a stator and rotor row. The order of stators and rotors are reversed with respect to the compressor.

### 3.2.1. ON-DESIGN CALCULATIONS

The total power to be generated by a turbine  $P_{\text{req}}$  is calculated from the total power requirement on its shaft, which also provides the rotational velocity. This required power is deducted from the energy of the flow through the turbine, and can therefore be translated into an expected drop in total temperature  $\Delta T_{\text{test}}$  according to

$$\Delta T_{\text{test}} = \frac{P_{\text{req}}}{\eta \cdot \dot{m}_{\text{in}} \cdot c_p} \quad (3.18)$$

where  $\eta$  is the polytropic efficiency,  $\dot{m}_{\text{in}}$  is the incoming massflow rate and  $c_p$  is the specific heat for constant pressure expansion of the incoming flow. Comparing this value with the actual total temperature drop found

<sup>8</sup>Source: [https://commons.wikimedia.org/wiki/File:J79-IHI-11A\\_turbojet\\_engine\(cutaway\\_model\)\\_turbine\\_section\\_right\\_side\\_view\\_at\\_JASDF\\_Hamamatsu\\_Air\\_Base\\_Publication\\_Center\\_November\\_24,\\_2014-18.jpg](https://commons.wikimedia.org/wiki/File:J79-IHI-11A_turbojet_engine(cutaway_model)_turbine_section_right_side_view_at_JASDF_Hamamatsu_Air_Base_Publication_Center_November_24,_2014-18.jpg), retrieved on 2018-06-10

for a single stage for an average work coefficient<sup>9</sup> allows for the estimation of the number of required stages. This procedure is visualised in Figure 3.8.

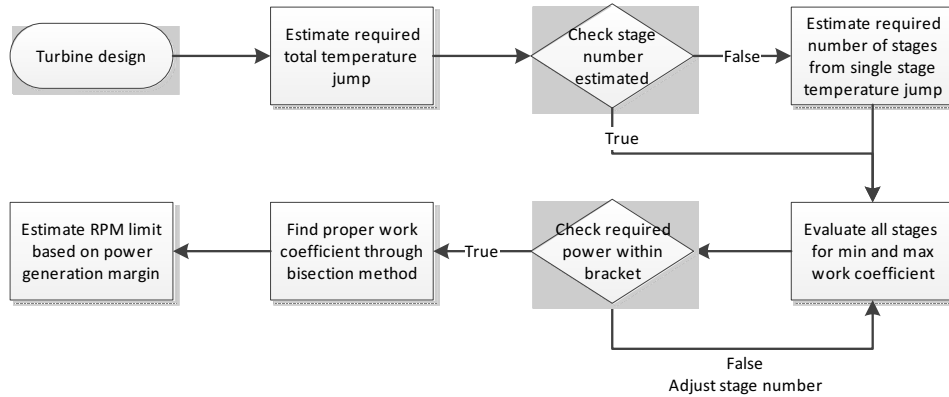


Figure 3.8: Axial turbine design procedure

Like with the compressor, velocity triangles are used to visualise the flow conditions through the turbine stages, the shapes of which are fully specified by three non-dimensional coefficients: the flow coefficient  $\phi = \frac{V_{ax}}{U}$ , the work coefficient  $\lambda = \frac{\Delta C_w}{U}$  and the degree of reaction  $\Lambda = 1 - \frac{C_{wavg}}{U}$ .

An often-made assumption ([18, p. 754]) is that of a constant axial velocity through the turbine.<sup>10</sup> Therefore, the initial attempt at a turbine stage design routine assumed the work coefficient  $\lambda$  (used as the control variable) and the degree of reaction  $\Lambda = 0.5$ <sup>11</sup>, while computing the flow coefficient  $\phi$  from the axial inflow velocity  $V_{ax}$  and the rotor mean-radius velocity  $U$ , which is computed based on the shaft RPM and the inflow mean radius  $r_{m1}$  according to

$$U = 2 \cdot \pi \cdot r_m \cdot \frac{\text{RPM}}{60} \quad (3.19)$$

Combining the definitions of these coefficients, expressions are then derived for the pre- and post-rotor absolute whirl velocity components  $C_{w2}$  and  $C_{w3}$ , yielding

$$C_{w2} = C_{wavg} + \frac{\Delta C_w}{2} = (1 - \Lambda) \cdot U + \frac{\lambda}{2} \cdot U \quad (3.20)$$

$$C_{w3} = C_{wavg} - \frac{\Delta C_w}{2} = (1 - \Lambda) \cdot U - \frac{\lambda}{2} \cdot U \quad (3.21)$$

By using the definition of the relative whirl velocity components  $W_w = C_w - U$  and the Pythagorean theorem  $V = \sqrt{V_{ax}^2 + C_w^2}$ , the absolute and relative velocities can now be calculated.

However, in case of a specified absolute rotor outflow angle  $\alpha_3$ , the rotor outflow whirl velocity is already defined as  $C_{w3} = V_{ax} \cdot \tan \alpha_3$ , requiring a recomputation of  $\Lambda$  to keep a consistent system, assuming  $U$  and  $\lambda$  are kept constant. This is done by rewriting Equation 3.21, yielding  $\Lambda = 1 - \frac{C_{w3}}{U} - \frac{\lambda}{2}$ , after which the corresponding  $C_{w2}$  is computed using Equation 3.20.

As [19, Slide 4] provides some boundaries for a.o.  $\phi$ , this provides a means of checking whether the found flow coefficient is reasonable and acceptable. If the obtained value of  $\phi$  lies outside of these bounds, it must therefore be concluded that either  $V_{ax}$  or  $U$  must be modified. As the shaft RPM is assumed fixed throughout the turbine calculation, this then leads to the conclusion that either the inflow axial velocity  $V_{ax}$  or the rotor mean radius  $r_m$  must be adjusted to obtain an acceptable flow coefficient for the calculations.

Initially, it was chosen to adjust the mean radius, and keep the axial inflow velocity constant, in order to obtain an acceptable value for  $\phi$  if this was originally found to be out of bounds. However, this approach failed, as the thus-obtained absolute and relative velocities were extremely high, yielding highly supersonic stator and rotor in- and outflow velocities. As [18, Sec. 10.8] lists a.o. subsonic rotor relative exit velocities as hard design constraints, it became clear that this approach did not yield satisfactory results.

<sup>9</sup>As the actual single-stage temperature drop may also contain a total temperature change contribution due to cooling flows and may therefore be larger than that purely due to power extraction, the number of stages is estimated for an average  $\lambda$  instead of the maximum allowable value.

<sup>10</sup>At least in the initial design phases. In actual turbines, if not constant, the velocities will generally increase over the turbine stages.

<sup>11</sup>Assuming a Parson stage, except for possibly the last stage due to outflow angle constraints.

Therefore, some modifications to the procedure were based on the approach described in [18, Sec. 10.7], where the axial inflow velocity is determined based on an assumed stator outflow Mach number  $M_2$ , which is given there as  $M_2 > 1$  for the first stage of a turbine<sup>12</sup> and  $M_2 < 1$  for any subsequent stages in that same turbine. Initially, in order to still try and keep the axial velocity constant over the turbine, this method is only used for determining the axial velocity at the first stage, keeping it constant for the rest of the turbine, with any possible modifications for  $\phi$  being done by altering  $U$ . However,  $M_2 < 1$  must be selected for this, even though it is the first stage, as energy is extracted in the turbine, decreasing the temperature and therefore also the local sonic velocity. Therefore, for the same velocity and similar values of the flow and work coefficients  $\phi$  and  $\lambda$ , the local Mach numbers of subsequent stages will always be higher, resulting in  $M_2 < 1$  for later stages being unobtainable if  $M_1 > 1$  for the first stage with constant axial velocity. For a given rotor velocity  $U$ , the axial velocity  $V_{ax}$  is estimated using

$$\begin{aligned} T_2 &= \frac{T_{t1}}{1 + \frac{\gamma-1}{2} \cdot M_2^2} \\ a_2 &= \sqrt{\gamma \cdot R \cdot T_2} \\ V_2 &= M_2 \cdot a_2 \\ V_{ax} &= \sqrt{V_2^2 - U^2 \cdot \left(1 - \Lambda + \frac{\lambda}{2}\right)^2} \end{aligned} \quad (3.22)$$

However, this calculation simply assumes that the specified  $U$  will yield a valid answer, which might not be the case, either due to the flow coefficient  $\phi$  exceeding its acceptable limits or due to the whirl velocity component term  $(U_2 \cdot (1 - \Lambda + \frac{\lambda}{2}))$  exceeding the desired local absolute velocity  $V_2$ . If either of these is the case, then the rotor velocity  $U$  should also be recalculated, leading to Equation 3.22 rewritten in terms of a specified  $\phi$  instead of  $U$ , yielding

$$V_{ax} = \frac{V_2}{\sqrt{1 + \frac{(1 - \Lambda + \frac{\lambda}{2})^2}{\phi^2}}} \quad (3.23)$$

where  $\phi$  is assumed as the minimum limit value if the whirl component exceeded  $V_2$  or the previously obtained flow coefficient was too low, and  $\phi$  is assumed as the upper limit value if the previously obtained value was too high. After this,  $U$  is then recomputed from  $V_{ax}$  and  $\phi$ , as is the mean radius  $r_m$  which follows from it. This procedure is illustrated in Figure 3.9.

With the velocities obtained, the contributions of possibly present cooling flows for the stator and/or rotor are computed, yielding new values for the post-rotor and post-stator total temperatures  $T_{t2}$  and  $T_{t3}$ , massflow rates  $\dot{m}_2$  and  $\dot{m}_3$  and the post-rotor and post-stator gas compositions. This is done based on the massflow rate, total temperature and gas composition of the uncooled flow at that point, and those of the bleed air used for cooling. The post-rotor uncooled total temperature  $T_{t3,uncooled}$  is obtained from the cooled post-stator total temperature  $T_{t2}$  and the pre- and post-rotor whirl velocities  $C_{w2}$  and  $C_{w3}$  using the Euler equation [18, Eq. 8.23]

$$T_{t3,uncooled} = T_{t2} + U \cdot \frac{C_{w3} - C_{w2}}{c_p} \quad (3.24)$$

Based on this total temperature jump, a rotor total temperature ratio can be estimated, which in turn can be translated to a total pressure ratio, leading to a value for the post-rotor total pressure  $p_{t3}$  according to

$$p_{t3} = p_{t2} \cdot \left( \frac{T_{t3,uncooled}}{T_{t2}} \right)^{\frac{\eta \gamma}{\gamma-1}} \quad (3.25)$$

From the uncooled total temperature jump over the rotor, the power generated by the stage  $P_{stage}$  is calculated according to

$$P_{stage} = (T_{t2} - T_{t3,uncooled}) \cdot c_p \cdot \dot{m}_2 \quad (3.26)$$

Like the compressor blades, the turbine blades are sized based on an assumed throat Reynolds number  $Re = 5 \cdot 10^5$ . More detailed on turbine blade sizing and the estimation of the associated solidities  $\sigma_r$  and  $\sigma_s$  are given in Section 3.11.1. From these solidities, the blade numbers  $N_{blade_s}$  and  $N_{blade_r}$  are estimated according to

$$N_{blade} \geq 2 \cdot \pi \cdot r_m \cdot \frac{\sigma}{c} \quad (3.27)$$

<sup>12</sup>This to ensure the stator is choked, allowing operation over a range of operating conditions.

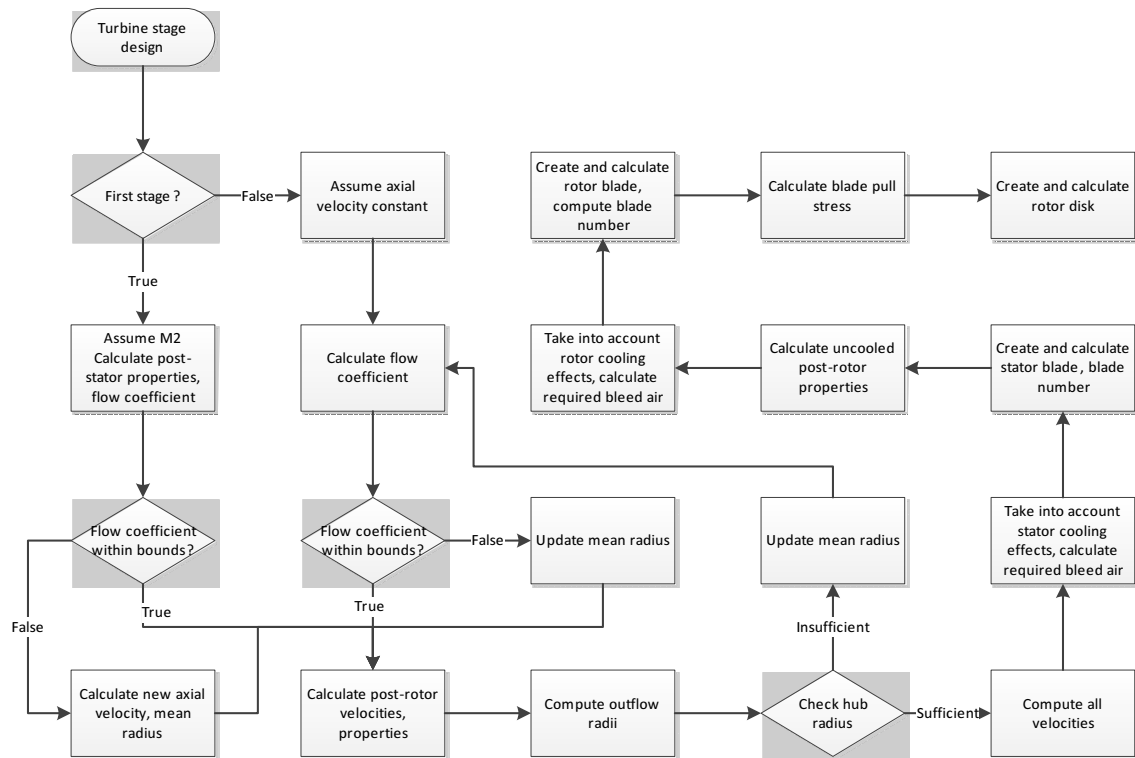


Figure 3.9: Axial turbine stage design procedure

The rotor disk blade pull stress  $\sigma_{bp}$  is then computed from [16, Eq. 3.58]

$$\sigma_{bp} = N_{blade_r} \cdot m_{blade_r} \cdot r_m \cdot \frac{(2 \cdot \pi \cdot \frac{RPM}{60})^2}{\pi \cdot 2 \cdot r_{hub} \cdot c_{blade_r}} \quad (3.28)$$

The rotor disk is then sized as described in Section 3.11.2. Finally, the RPM limit is taken as the lowest value found for all of the stages.

### 3.2.2. MASS ESTIMATION

The mass estimation of an axial turbine is done completely analogous to the process for an axial compressor, as described in Section 3.1.2. However, due to the higher temperatures present in the turbine compared to the compressor, the actual values for e.g. yield strength  $\sigma_y$  are likely to be lower than the assumed values, which are usually obtained for the material at room temperature. While not currently implemented, this does form a potential later improvement for the code.

## 3.3. FAN

The fan (as depicted in Figure 3.10) is the part of a turbofan engine which performs the first compression on all air flowing through the engine, as well as splitting the incoming airflow into a core and bypass outflow. The ratio between the amount of air going through the engine core (the core or “hot” flow) and the amount of air diverted past the engine core (the bypass or “cold” flow) is called the bypass ratio or BPR, and is defined according to  $BPR = \frac{\dot{m}_{bypass}}{\dot{m}_{core}}$ .

Within MIME, the fan is assumed to be a single-stage component.

### 3.3.1. ON-DESIGN CALCULATIONS

The fan is assumed to have two separately specified total pressure ratios: one for the core and one for the bypass flow. This implies that each section must be evaluated separately, both on the rotor as well as on the stators.

<sup>13</sup>Source: [https://commons.wikimedia.org/wiki/File:GE90\\_dsc04644.jpg](https://commons.wikimedia.org/wiki/File:GE90_dsc04644.jpg), retrieved on 2018-06-10



Figure 3.10: Fan of a GE90 engine<sup>13</sup>

Again, the assumption is made that the total properties only change over the rotor, and therefore remaining constant over the stator. For both the core and bypass flows, the post-rotor total pressures  $p_{t_{rc}}$  and  $p_{t_{rbp}}$  can be found by multiplying the inflow total pressure with the respective core and bypass total pressure ratios  $PR_c$  and  $PR_{bp}$ . The post-rotor total temperatures  $T_{t_{rc}}$  and  $T_{t_{rbp}}$  then follow directly from the pressure ratios according to

$$\begin{aligned} T_{t_{rc}} &= T_{t_{in}} \cdot PR_c^{\frac{\gamma-1}{\eta\gamma}} \\ T_{t_{rbp}} &= T_{t_{in}} \cdot PR_{bp}^{\frac{\gamma-1}{\eta\gamma}} \end{aligned} \quad (3.29)$$

The flow split is assumed to happen between the rotor and stators, with the core and bypass massflows  $\dot{m}_c$  and  $\dot{m}_{bp}$  being calculated by

$$\begin{aligned} \dot{m}_c &= \dot{m}_{in} \cdot \frac{1}{BPR+1} \\ \dot{m}_{bp} &= \dot{m}_{in} \cdot \frac{BPR}{BPR+1} \end{aligned} \quad (3.30)$$

Power consumption of the rotor is estimated as the sum of the power consumptions of both rotor sections, with each being calculated according to

$$P = \dot{m} \cdot c_p \cdot (T_{out} - T_{in}) \quad (3.31)$$

Based on the massflow divisions, the radius dividing the core and bypass massflows at the fan inflow can be computed, which in turn can be used to define the core and bypass inflow mean radii. For the fan, several design options are available besides the standard constant mean radius, as it is also possible to instead specify a factor by which either the core root radius, the mean radii or the bypass tip radius changes over the fan (which may also simply equal 1 for keeping that value constant), based on the approach by [16]. The appropriate associated mean radii must then be estimated using an iterative process. For example, Figure 3.11 illustrates the case where the tip diameter remains constant.

As this implies a non-constant mean radius may occur over the rotor sections and stators, the rotor section in- and outflow rotational velocities  $U_{in}$  and  $U_r$  will not necessarily be equal either, with each being estimated according to [16, Eq. 3.22]. This then allows for estimation of the post-rotor whirl velocity component  $C_{wr}$  for that specific rotor section by rewriting [16, Eq. 3.24] into

$$C_{wr} = \frac{\frac{P}{\dot{m}} + U_{in} \cdot C_{w_{in}} \cdot \frac{r_{in}}{r_{min_{section}}}}{U_{r_{out}}} \quad (3.32)$$

From this, the post-rotor absolute velocities and angles can then be calculated using the assumption that there is no change in axial velocity occurring over the rotor. With the stator outflows angles being either specified or assumed, the velocities at all points within the fan can then be calculated.



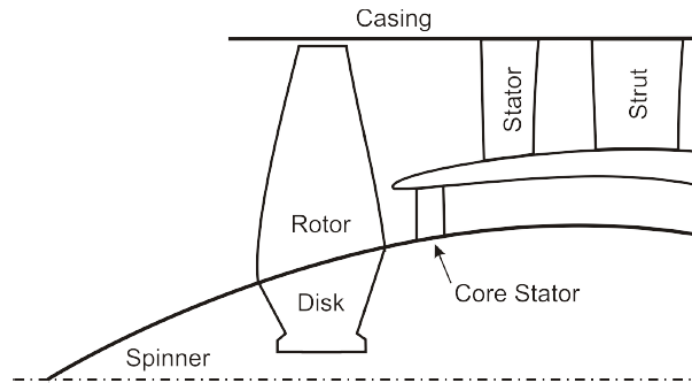


Figure 3.11: Fan radius progression for constant tip radius [16, Fig. 3.9]

The solidities of the stators and rotor  $\sigma$  are estimated using the diffusion factor DF through [16, Eq. 3.39]

$$\sigma = \frac{|C_{w_{out}} - C_{w_{in}}|}{V_{in} \cdot (DF - 1 + \frac{V_{out}}{V_{in}})} \quad (3.33)$$

However, as the rotor would now possibly yield two different values for the number of blades, the solidity for the core flow rotor section is assumed for the rotor.

The limit RPM is finally estimated based on a limit rotor Mach number, according to

$$\text{RPM}_{\text{lim}} = \sqrt{M_{\text{lim}}^2 \cdot a^2 - V_{\text{ax}}^2} \cdot \frac{60}{2 \cdot \pi \cdot r_{\text{tip}}} \quad (3.34)$$

### 3.3.2. MASS ESTIMATION

The mass of a fan is estimated by summing the mass contributions of the fan rotor blades, core and bypass flow stator blades, rotor disk, casings, nosecone and coupling hardware. The latter is modelled as a 0.075"  $\approx$  1.9 [mm] thick ring at 75% of the disk radius (equal to the rotor hub radius), similar to the method for the axial compressor and turbine stages, using Equation 3.12.

The casing mass contributions are divided into 3 segments: the casing of the rotor and the casings of both stators. As done for the axial compressor and turbine stages, these are each individually sized for containment of the internal pressure and/or minimum thickness<sup>14</sup>, as well as for containment of a detached rotorblade for the rotor casing, as described in Section 3.1.2.

A component which does not occur in an axial compressor stage is the nosecone. This is assumed to be made of 5 [mm] thick fiberglass [16, Pg. 75], assumed to have a density of  $\rho_{\text{fiberglass}} \approx 1500 \left[ \frac{\text{kg}}{\text{m}^3} \right]$ . Its length is estimated by assuming the same slope as that of the fan rotor hub radius, with a maximum of twice the rotor length.

### 3.4. DUCT

Ducts are those parts within the engine which surround the flow between two points, while not performing any work on the flow. They are mainly used for adjusting the mean radius and/or axial velocity of the flow between components.

During calculation of the post-duct flow conditions, it is assumed that no change in the total temperature occurs, as a duct is assumed to not perform any work on the flow, and is also assumed to not add or remove heat from the flow. However, it is assumed that a total pressure ratio drop may occur over the duct, which if present is specified by a duct total pressure ratio. If no value is specified, it is assumed that no pressure loss occurs and so both total pressure and temperature are conserved over the duct. With the post-duct total properties, mean radius and axial velocity known, the outflow static properties are computed according to the standard isentropic relations. From these results, the outflow area and radii then follow directly.

While in reality, any duct (or in fact any through-flow component in general) will have a smooth flow path, with any changes in flow direction and/or radius being smoothed out in order to minimise losses, the duct model of MIME simply models ducts as straight tapered annuli for the sake of simplicity. While smoothing

<sup>14</sup>Whichever is greater.

through the use of polynomial functions was considered, this was rejected due to the introduction of additional complexities in mass estimation, as well as the introduction of extra variables. The additional level of detail was deemed to be not worth the effort as the possible improvements were likely to not weigh up to the error margins in most other components, nor be accurate due to the required assumptions involved in the used variables.

For estimation of the duct length  $l_{\text{duct}}$ , a maximum allowable angle  $\alpha_{\text{max}}$  is assumed. For a non-constant mean radius  $r_{\text{mean}}$ , this is applied to the mean radii to yield an estimate for the duct length, according to

$$l_{\text{duct}} = \left| \frac{r_{\text{min}} - r_{\text{out}}}{\tan \alpha_{\text{max}}} \right| \quad (3.35)$$

If the mean radius is however constant over the duct (implying only the axial velocity and therefore the hub and tip radii change), the maximum allowable angle is imposed on the in- and outflow tip radii instead, as illustrated in Figure 3.12.

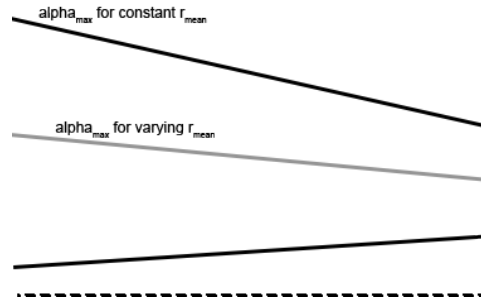


Figure 3.12: Maximum angle definition for ducts

Duct material is assumed to be dependent on the maximum total temperature  $T_{t_{\text{max}}}$  encountered, with steel being chosen for higher temperatures ( $T_{t_{\text{max}}} > 700^\circ\text{F} \approx 371^\circ\text{C}$ , [15]), while for lower temperatures titanium is assumed. Duct wall thickness is estimated based on the maximum internal total pressure  $p_{t_{\text{max}}}$  according to the boiler equations for the maximum duct diameter, while the wall volumes are approximated as thin-walled cylinders of the average hub and tip diameters. Multiplication of these volumes with the material density results in the duct mass estimate.

## 3.5. COMBUSTOR

The combustor or combustion chamber is a part in the engine where fuel is added and combusted, usually at elevated pressures, to increase the energy in the flow. While early jet engines, such as those of Sir Frank Whittle, used so-called can combustors, where several cylindrical combustion chambers are arranged in parallel, development later shifted to can-annular (annular outer casing, can-type liner) combustors, with purely annular combustors being common in modern jet engines.

These various combustor types are depicted in Figure 3.13. For the reasons mentioned earlier, it was decided to initially model a simple annular combustion chamber. Further detail may be added by future contributors.

### 3.5.1. ON-DESIGN CALCULATIONS

The computations for the thermodynamic and chemical performance of the combustor are taken from the current Initiator implementation's `combustion.m` function, which supports three types of fuel: kerosene, hydrogen and methane. For all of these, characteristic values are defined according to Table 3.1.

The required fuel flow  $\dot{m}_f$  and post-combustor flow composition are then computed by first determining the required jump in total temperature  $\Delta T_t = T_{t_{\text{out}}} - T_{t_{\text{in}}}$ , as well as the  $\Delta Q = c_{p_f} \cdot T_{t_{\text{fuel}}} - c_{p_d} \cdot T_{\text{ref}}$ . Using these

<sup>15</sup>Source: [https://commons.wikimedia.org/wiki/File:Cannular\\_combustor\\_on\\_a\\_Pratt\\_%26\\_Whitney\\_JT9D\\_turbofan.jpg](https://commons.wikimedia.org/wiki/File:Cannular_combustor_on_a_Pratt_%26_Whitney_JT9D_turbofan.jpg), retrieved on 2018-06-10

<sup>16</sup>Source: [https://commons.wikimedia.org/wiki/File:Royal\\_Military\\_Museum\\_Brussels\\_2007\\_297.JPG](https://commons.wikimedia.org/wiki/File:Royal_Military_Museum_Brussels_2007_297.JPG), retrieved on 2018-06-10

<sup>17</sup>Source: [https://commons.wikimedia.org/wiki/File:Sectioned\\_combustor\\_of\\_Atar\\_turbojet\\_engine\\_\(2\).jpg](https://commons.wikimedia.org/wiki/File:Sectioned_combustor_of_Atar_turbojet_engine_(2).jpg), retrieved on 2018-06-10



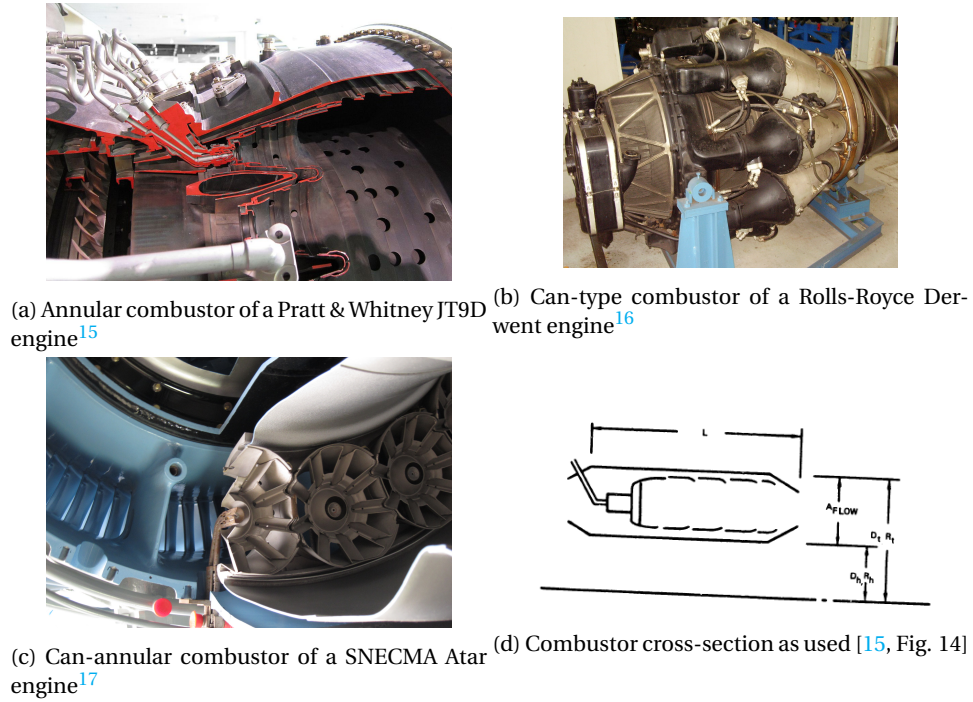


Figure 3.13: Various combustor types

Table 3.1: Characteristic values for various fuels

	Kerosene	Hydrogen	Methane	Unit
$Q_f$	42.80	120.00	50.00	$\left[\frac{MJ}{kg}\right]$
$H_v$	38.75	N/A	N/A	$\left[\frac{kJ}{kg}\right]$
$c_{p_f}$	2039.85	variable	variable	[-]
$c_{p_d}$	2039.85	14304.59	2224.75	[-]
$m_M$	167.31	2.02	16.04	$\left[\frac{kg}{mol}\right]$
$T_{ref}$	298.15	298.15	298.15	[K]

two quantities, the fuel massflow rate  $\dot{m}_f$  is computed using

$$\dot{m}_f = \frac{\dot{m}_{in} \cdot c_p \cdot \Delta T_t}{\eta \cdot (Q_f + \Delta Q - \frac{H_v}{m_M})} \quad (3.36)$$

where  $c_p$  is that of the resulting gas mixture.<sup>18</sup> As both the incoming gas composition and massflow rate as well as the fuel composition and massflow rate are then known, the outflow massflow rate  $\dot{m}_{out}$  and gas composition can then be calculated. For reference, the combustor power  $P_{comb}$  is also computed according to

$$P_{comb} = \eta \cdot c_{p_{out}} \cdot \dot{m}_f \cdot (T_{t_{out}} - T_{t_{in}}) \quad (3.37)$$

In order to estimate the length of the combustor, a residence time  $t_{res}$  parameter is specified, which is then multiplied with the average of the in- and outflow axial velocities to obtain an estimate of the combustor length.<sup>19</sup> Finally, based on the set axial outflow velocity or Mach number, the outflow area is calculated. The entire procedure is depicted schematically in Figure 3.14.

<sup>18</sup>Therefore, solving this equation is done iteratively, as described in Section 10.8.

<sup>19</sup>This is a very rough method, as the residence time is not actually the time spent in the entire combustor, but rather in the flame area itself. Also, the velocities through a combustor are often far from linear and vary greatly, slowing down near the burner and speeding up near the outflow.

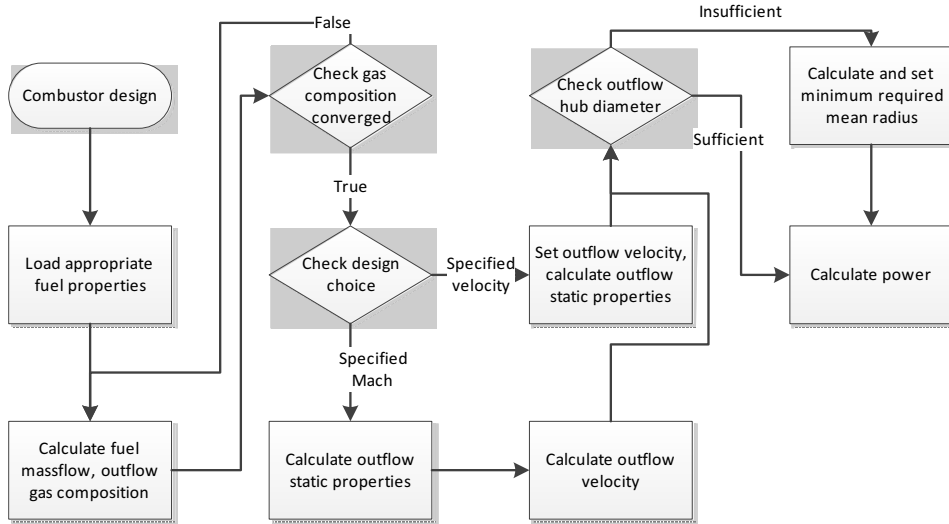


Figure 3.14: Combustor design procedure

### 3.5.2. MASS ESTIMATION

As mentioned earlier, the combustor is assumed to be of the annular type. This then allows for modelling of the combustor as an annular duct with inner and outer casing walls, liners to allow for proper combustion and a mass contribution for the domes, fuel nozzles, et cetera. The casing dimensions are derived from the in- and outflow area and an assumption of a constant mean radius over the combustor.<sup>20</sup> From the casing diameters, also the diameters of the combustor liners are computed, which are assumed to be at a distance of 20% of the passage height from the inner and outer walls. [14]

The thickness of the casings  $t_{\text{casing}}$  is assumed to be equal for the inner and outer casing and is determined based on the boiler equation

$$t_{\text{casing}} = \frac{p_{\text{max}} \cdot D_{\text{outer}}}{2 \cdot \sigma_y} \quad (3.38)$$

where  $p_{\text{max}}$  is the highest total pressure encountered in the combustor,  $D_{\text{outer}}$  is the maximum outer diameter of the outer casing and  $\sigma_y$  is the material yield strength.<sup>21</sup> The liner thickness is assumed to be a constant  $t_{\text{liner}} = 0.055'' \approx 1.4$  [mm]. [14] This then leads to the estimation of the masses of the different casings and liners according to

$$m = \pi \cdot l_{\text{comb}} \cdot D \cdot t \cdot \rho \quad (3.39)$$

where  $m$  is the mass of that casing,  $l_{\text{comb}}$  is the combustor length,  $D$  and  $t$  are the diameter and thickness of that casing, and  $\rho$  is the casing material density.

For the mass contribution of the domes and fuel nozzles et cetera, an estimate is made using [14, Eq. 17]

$$m_{\text{dome}} = 0.0106 \cdot l_{\text{comb}} \cdot (r_{\text{tip}}^2 - r_{\text{hub}}^2) \quad (3.40)$$

where  $m_{\text{dome}}$  is the mass estimate of dome, fuel nozzles, manifold et cetera in [lbm] and  $l_{\text{comb}}$ ,  $r_{\text{tip}}$  and  $r_{\text{hub}}$  are the combustor length, hub and tip diameter in [ft].

## 3.6. INLET

The inlet is the part of the engine which allows ambient air to be sucked into the engine. As the Initiator specifies the fan inflow diameter (and therefore, for a normal turbofan engine, the inlet outflow diameter) as an input for the engine design routing in order to define the engine size (as opposed to the design thrust level, which might have been a more obvious choice), the inflow duct therefore also defines the massflow rate going through the engine at the design condition.

<sup>20</sup>This assumption is made for simplicity only; in reality many combustors may feature a change in mean radius. As the mass of the combustor is relatively low compared to that of other components, the resulting error is deemed acceptable.

<sup>21</sup>The used material yield strength is the standard value, which is defined at room temperature. However, at the elevated temperatures of the combustor, the yield strength tends to be lower, so ideally, a temperature-dependent function is used for the yield strength.

While multiple inlet styles exist for different speed regimes and integration challenges, the only type of inlet currently supported is a round pitot-style inlet as used on virtually all airliners, for which the outflow diameter, in- and outflow hub-to-tip ratios<sup>22</sup> and outflow Mach number are specified.

### 3.6.1. ON-DESIGN CALCULATIONS

When air passes through the inlet, it is assumed that no energy is added to the flow, keeping the total temperature  $T_{\text{tout}} = T_{\text{tin}}$  constant with respect to the ambient conditions. The outflow static temperature  $T_{\text{out}}$  is then computed from the specified axial outflow Mach number  $M_{\text{out}}$ , which defaults to a typical value of  $M_{\text{out}} = 0.6$  [16, pg. 71] if not specified, according to

$$T_{\text{out}} = \frac{T_{\text{tout}}}{1 + \frac{\gamma-1}{2} \cdot M_{\text{out}}^2} \quad (3.41)$$

The actual axial outflow velocity  $V_{\text{axout}}$  is then computed from the newly found static temperature through the sonic velocity and the assumption of purely axial in- and outflow using

$$V_{\text{axout}} = M_{\text{axout}} \cdot \sqrt{\gamma \cdot R \cdot T_{\text{out}}} \quad (3.42)$$

Outflow total and static pressures can then be found according to

$$p_{\text{tout}} = p_{\text{amb}} \cdot \left( 1 + \eta \cdot \frac{V_{\text{in}}^2}{2 \cdot c_p \cdot T_{\text{in}}} \right)^{\frac{\gamma}{\gamma-1}} \quad (3.43)$$

$$p_{\text{out}} = p_{\text{tout}} \cdot \left( \frac{T_{\text{tout}}}{T_{\text{out}}} \right)^{\frac{-\gamma}{\gamma-1}} \quad (3.44)$$

The outflow area  $A_{\text{out}}$  and outflow mean radius  $r_{\text{mout}}$  are then computed from the outflow diameter  $D_{\text{out}}$  and the outflow specified hub-to-tip ratio  $\left(\frac{H}{T}\right)_{\text{out}}$  according to

$$A_{\text{out}} = \pi \cdot \frac{D_{\text{out}}^2 \cdot \left( 1 - \left(\frac{H}{T}\right)_{\text{out}} \right)^2}{4} \quad (3.45)$$

$$r_{\text{mout}} = D_{\text{out}} \cdot \frac{1 + \left(\frac{H}{T}\right)_{\text{out}}}{4} \quad (3.46)$$

The on-design massflow rate through the inlet (and therefore the engine)  $\dot{m}$  is then computed using

$$\dot{m} = \rho_{\text{out}} \cdot A_{\text{out}} \cdot V_{\text{axout}} \quad (3.47)$$

With the outflow mean radius  $r_{\text{mout}}$  and the massflow rate  $\dot{m}$  known, the inflow area  $A_{\text{in}}$  can be found and then translated into a mean inflow radius  $r_{\text{min}}$ , depending on the specified inflow hub-to-tip ratio  $\left(\frac{H}{T}\right)_{\text{in}}$ , using

$$\begin{aligned} A_{\text{in}} &= \frac{\dot{m}}{\rho_{\text{amb}} \cdot V_{\text{amb}}} \\ r_{\text{min}} &= \frac{\sqrt{\frac{A_{\text{in}}}{\pi}}}{2 \cdot \sqrt{1 - \left(\frac{H}{T}\right)_{\text{in}}^2}} \cdot \left( \left(\frac{H}{T}\right)_{\text{in}} + 1 \right) \end{aligned} \quad (3.48)$$

### 3.6.2. MASS ESTIMATION

The mass estimation of an inlet can be a relatively tricky undertaking, as its design does not purely rely on specifications of the engine itself. For example, the inlets of the CFM56-7B's on the Boeing 737-NG series have a flattened lower lip, which is due to ground clearance reasons. Also, the design is dependent on both the on- and off-design engine running conditions. A small introduction into the design complexities of aero-engine inlets may for example be found in [23, Ch. 37].

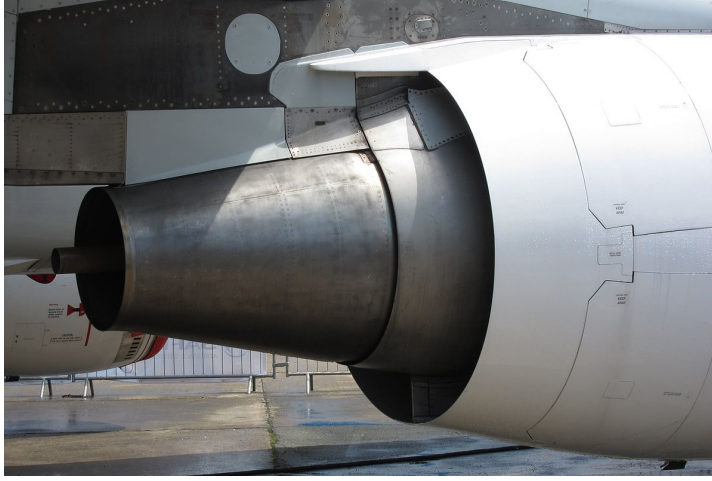
Therefore, as the inlet can be seen as a relatively separate part from the rest of the engine, which may also vary in appearance between aircraft, its mass estimation is not yet included in MIME.

## 3.7. NOZZLE

For a jet engine, the nozzle is responsible for the main contribution to the thrust. It expands the flow to ambient pressure (or as close to it as possible), increasing its velocity. The nozzle in MIME is currently modelled as a standard, fixed-geometry, convergent nozzle for subsonic purposes, an example of which is depicted in Figure 3.15.

<sup>22</sup>These are used e.g. to account for the presence of spinners et cetera, or may also be set to 0 for completely circular cross-sections (e.g. no hub radius).

<sup>23</sup>Source: [https://commons.wikimedia.org/wiki/File:Exhaust\\_nozzles\\_of\\_a\\_CFM56\\_turbofan.jpg](https://commons.wikimedia.org/wiki/File:Exhaust_nozzles_of_a_CFM56_turbofan.jpg), retrieved on 2018-06-10

Figure 3.15: Nozzle of a CFM56 engine<sup>23</sup>

### 3.7.1. ON-DESIGN CALCULATIONS

A nozzle can be choked or unchoked, depending on whether the pressure ratio over the nozzle is high enough. In order to determine whether this is the case, the critical pressure ratio  $PR_{\text{crit}}$  needs to be computed and compared to the actual pressure ratio  $PR$ , according to

$$PR_{\text{crit}} = \frac{1}{\left(1 - \frac{\gamma-1}{\eta(\gamma+1)}\right)^{\frac{\gamma}{\gamma-1}}} \quad (3.49)$$

$$PR = \frac{p_{\text{tin}}}{p_{\text{amb}}} \quad (3.50)$$

If the nozzle pressure ratio exceeds the pressure ratio  $PR > PR_{\text{crit}}$ , the nozzle is assumed to be choked, implying that the Mach number in the nozzle throat is supersonic. Therefore, the static outflow temperature and pressure must be equal to the critical temperature  $T_c$  and pressure  $p_c$  according to

$$T_c = T_{\text{tin}} \cdot \frac{2}{\gamma+1} \quad (3.51)$$

$$p_c = p_{\text{tin}} \cdot \frac{1}{PR_{\text{crit}}} \quad (3.52)$$

As the nozzle is choked and thus  $M_{\text{throat}} = 1$ , the outflow axial velocity must then be equal to the sonic velocity, given by

$$V_{\text{ax,out}} = a_{\text{out}} = \sqrt{\gamma \cdot R \cdot T_c} \quad (3.53)$$

Assuming the outflow gas behaves like a perfect gas, the outflow density  $\rho_{\text{out}}$  is computed from the ideal gas law according to

$$\rho_{\text{out}} = \frac{p_c}{R \cdot T_c} \quad (3.54)$$

which allows for calculation of the nozzle outflow area  $A_{\text{out}}$  using

$$A_{\text{out}} = \frac{\dot{m}}{\rho \cdot V_{\text{ax,out}}} \quad (3.55)$$

For a choked nozzle, the flow is not expanded to the ambient static pressure, as stated earlier. This implies that the engine thrust  $F$  consists of not only a component due to the increased momentum of the flow, but also a component due to the pressure thrust, leading to the thrust  $F$  and gross thrust  $F_g$  being calculated according to

$$\begin{aligned} F &= \dot{m} \cdot (V_{\text{ax,out}} - V_{\text{amb}}) + A_{\text{out}} \cdot (p_c - p_{\text{amb}}) \\ F_g &= \dot{m} \cdot V_{\text{ax,out}} + A_{\text{out}} \cdot (p_c - p_{\text{amb}}) \end{aligned} \quad (3.56)$$

If however the actual pressure ratio over the nozzle remains below the critical pressure ratio  $PR \leq PR_{\text{crit}}$ , the nozzle is assumed to remain unchoked, with the flow being expanded completely to the ambient pressure. The outflow static temperature  $T_{\text{out}}$  is assumed to be calculated from the nozzle inflow total temperature  $T_{\text{tin}}$  and the nozzle pressure ratio  $PR$  according to

$$T_{\text{out}} = T_{\text{tin}} \cdot \left(1 - \eta \cdot \left(1 - \frac{1}{PR^{\frac{\gamma}{\gamma-1}}}\right)\right) \quad (3.57)$$

This then leads to the estimation of the outflow axial velocity  $V_{ax_{out}}$  based on this temperature difference according to

$$V_{ax_{out}} = \sqrt{2 \cdot c_p \cdot T_{in} \cdot \eta \cdot \left(1 - \frac{1}{PR^{\frac{\gamma-1}{\gamma}}}\right)} \quad (3.58)$$

As the flow is now assumed to be fully expanded, the nozzle outflow overpressure no longer exists, so the pressure thrust term drops out of the thrust calculation, yielding the calculation of the thrust and gross thrust  $F$  and  $F_g$  according to

$$\begin{aligned} F &= \dot{m} \cdot (V_{ax_{out}} - V_{amb}) \\ F_g &= \dot{m} \cdot V_{ax_{out}} \end{aligned} \quad (3.59)$$

Regardless of whether or not the nozzle is choked, the outflow dimensions are based on the calculated outflow area, and the assumption of a constant inner diameter.

### 3.7.2. MASS ESTIMATION

In order to estimate its mass, the nozzle is assumed to be shaped like a conical duct, with a wall thickness of  $t = \frac{1}{10}'' \approx 2.54$  [mm]. For flow total temperatures exceeding  $700^\circ F \approx 644.3$  [K], a steel nozzle is assumed, while for lower temperatures, titanium is selected as the material of choice. [14]

The nozzle mass  $m_{nozzle}$  is then estimated by multiplying the material density  $\rho_{mat}$  with the wall thickness  $t$  and the areas of the outer and inner (if applicable) nozzle surfaces  $A$ , which are estimated according to

$$\begin{aligned} A_{outer} &= \pi \cdot (D_{tin} + D_{tout}) \cdot \sqrt{l_{nozzle}^2 + (D_{tin} - D_{tout})^2} \\ A_{inner} &= \pi \cdot (D_{hin} + D_{hout}) \cdot \sqrt{l_{nozzle}^2 + (D_{hin} - D_{hout})^2} \end{aligned} \quad (3.60)$$

## 3.8. SHAFT

Shafts are used to connect power-producing and power-consuming rotating components. The most notable of these are the compressors and turbines, however this category also includes for example fans, propellers, gearboxes, electrical motors and generators and auxiliary equipment power offtakes. Even though, except for the fan, these components are not yet incorporated into the MIME code, the code is prepared to be able to handle them.

Within turbine engines, shafts usually take the form of concentric hollow tubes, which are often loaded on their extremities, with forces assumed to be transferred within the components as well. Therefore, the shafts are assumed to run from the aft-most end of the forward-most component to the front of the aft-most component on the shaft. For simplicity, shafts within the MIME code are assumed to be straight with a constant radius and thickness. Also, for obvious reasons, the rotational velocity of the shaft is assumed to be equal over the entire shaft, and is assumed to be equal to the lowest maximum allowable RPM of all components on the shaft.

Mechanical losses in power transmission may occur due to a variety of reasons, and are taken into account by incorporating a value for the mechanical efficiency  $\eta_{mech}$ , giving the ratio between the power demand and supply.

In order to calculate the required diameter of the shaft, the maximum torque on the shaft  $T$  first needs to be calculated. This is done by taking the total power transferred through the shaft  $P$  and dividing it by the shaft rotation  $\omega$ , yielding  $T = \frac{P}{\omega}$ . For a shaft which does not have any other shaft running through it, the outer and inner diameters  $D_o$  and  $D_i$  can then be estimated based on a rewritten version of [15, Eq. 22] and an assumed hub-to-tip ratio  $\frac{H}{T}$  according to

$$\begin{aligned} D_o &= \left( \frac{16 \cdot T}{\pi \cdot \tau_{yield} \cdot \left(1 - \left(\frac{H}{T}\right)^4\right)} \right)^{\frac{1}{3}} \\ D_i &= D_o \cdot \frac{H}{T} \end{aligned} \quad (3.61)$$

If however the inner diameter is already defined due to the presence of another shaft inside the shaft, the outer diameter is calculated instead by numerically solving [15, Eq. 22]

$$\tau_{yield} = \frac{16 \cdot T \cdot D_o}{\pi \cdot (D_o^4 - D_i^4)} \quad (3.62)$$

With the inner and outer diameters as well as the length known, the volume of the tube  $V_{shaft}$  is fully defined, and can be easily translated into a shaft mass  $m_{shaft}$  by multiplying it with the material density  $\rho$ .

### 3.9. FLOW

While the flows are no physical components of the engine, especially the bleed air flow is an important part of the engine sizing, as it governs the off-take and supply of bleed air for cooling (and potentially other) purposes.

As bleed air has to flow from its source to its point of use, the pressure at the source needs to be at least as high as the pressure at the point of use. Therefore, different source mechanisms can be used. The simple way is to source the bleed air at the point of highest (cold) pressure in the engine, being right after the final compressor stage. However, this does imply that the pressure may be significantly higher than required, yielding losses. Therefore, in actual engines, it is often visible that bleed air for different uses is sourced from different locations, as can be seen in Figure 3.16.



Figure 3.16: Bleed air ducts<sup>24</sup>

In order to determine these locations, for each bleed air demand, the supply with the least amount of margin over the demand pressure<sup>25</sup> is selected and assigned, in order to reduce the amount of energy put into needlessly compressing the bleed air further than required.

### 3.10. FRAME

While the separate components of the engine make up the bulk of the engine mass, they do require something to keep them in their proper relative positions. In some cases, this is simply the preceding and/or following component, however in some places frames are present. This may include support for shaft bearings or other components. Due to the absence of information on the actual loads on these components in the design phase applicable to the Initiator, it is very hard, if not impossible, to perform a mechanical design procedure for these parts. Therefore, a method is used based on a correlation with the flow tip radius at the frame location and the type of frame.

[15, Section 1.1.7] recognises 4 different types of frames: single bearing frames with and without power off-take, turbine frame and intermediate or burner frames, for each of which frame mass estimation relations are given in [15, Fig. 17] as a function of the square of the local tip radius. This was stated to yield acceptable results at that time and, for lack of a better method, is used in the MIME code as well.

### 3.11. LOWER-LEVEL COMPONENTS

Next to the earlier-described top-level components, there are several objects which recur as components of some of these top-level components. These include the blades for the axial turbines and compressors, the rotor disks and the inlet guide vanes.

#### 3.11.1. BLADE

Blades are essential components of axial-flow turbomachinery parts. They change the direction of the flow in order to allow for the addition or subtraction of energy to or from the flow. While the in- and outflow angles

<sup>24</sup>Source: [https://www.shell.nl/over-ons/venster/eerder-verschenen/de-toekomst-van-elektrisch-vliegen/\\_jcr\\_content/pagePromo/image.img.800.jpeg/1498664796537/shell-venster-elektrisch.jpeg](https://www.shell.nl/over-ons/venster/eerder-verschenen/de-toekomst-van-elektrisch-vliegen/_jcr_content/pagePromo/image.img.800.jpeg/1498664796537/shell-venster-elektrisch.jpeg), retrieved on 2018-06-10

<sup>25</sup>Optionally increased by absolute and/or relative loss margins.



of each blade are determined at the level of the compressor and turbine stages, the shapes and dimensions of the blades are found at the blade level.

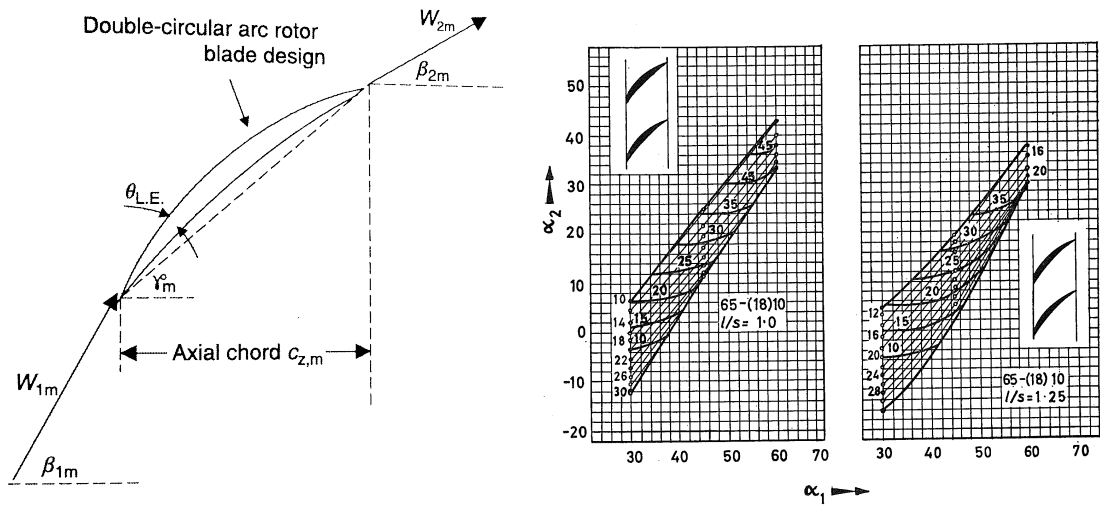
### COMPRESSOR BLADES

For a compressor blade, these are estimated through the method laid out by [18], based on a given passage height  $h_{\text{passage}}$ , mean radius  $r_m$ , thickness-to-chord ratio  $\frac{t}{c}$ , solidity  $\sigma$ , typical Reynolds number  $Re_{\text{typ}}$ , kinematic viscosity  $\nu$  and the Mach number  $M_{\text{in}}$ , inflow velocity  $V_{\text{in}}$  and in- and outflow angles  $\beta_1$  and  $\beta_2$  relative to the blade.<sup>26</sup>

The blade dimensions are given by the blade height  $h_{\text{blade}}$  and blade chord  $c$ . The blade height is assumed to be equal to the passage height  $h_{\text{blade}} = h_{\text{passage}}$ , while the blade chord is estimated based on the typical Reynolds number, according to

$$c = \frac{Re_{\text{typ}} \cdot \nu}{V_{\text{in}}} \quad (3.63)$$

The compressor blade shape is assumed to depend on the blade inflow relative Mach number. For transsonic and supersonic blades ( $M \geq 0.8$ ), the blade shape is assumed to be a double-circular arc (DCA), while for lower Mach numbers, a NACA 65-series airfoil is assumed. Both of these are shown in Figure 3.17.



(a) Double-Circular Arc (DCA) compressor blade airfoil [18, Pg. 633]

(b) NACA65 compressor blade cascade data [24, Fig. 3.9]

Figure 3.17: Compressor blades

For a DCA airfoil, the airfoil incidence angle  $\iota$  [rad] is assumed to be equal to the mean  $\frac{t}{c}$  ratio of the blade. The blade deviation angle  $\delta$  [rad] is now estimated through Carter's rule [18, Eq. 8.208]

$$\delta = \frac{\beta_2 - \beta_1}{4 \cdot \sqrt{\sigma}} \quad (3.64)$$

to which another  $2^\circ$  is added in case of supersonic inflow ( $M \geq 1$ ). From these angles, the leading edge angle  $\kappa_1$ , trailing edge angle  $\kappa_2$ , camber angle  $\phi$  and stagger angle  $\gamma$  are now obtained according to

$$\begin{aligned} \kappa_1 &= \beta_1 - \iota \\ \kappa_2 &= \beta_2 - \delta \\ \phi &= \kappa_1 - \kappa_2 \\ \gamma &= \beta_1 - \iota - \frac{\phi}{2} \end{aligned} \quad (3.65)$$

The axial chord  $c_{\text{ax}}$  can then be determined through its definition  $c_{\text{ax}} = c \cdot \cos \gamma$ , which after the multiplication with a clearance factor yields the length of this stator or rotor segment.

If a NACA 65-series airfoil is assumed instead, the appropriate airfoil is selected based on the graphed data as presented in [24, Fig. 3.9]. From these, the nearest solidity is selected, for which the angle of attack  $\alpha$

<sup>26</sup>This implies that, for a rotor, these values are those in the relative frame of reference, while for a stator, which does not rotate, the frame of reference relative to the stator is equal to the absolute frame of reference.

yielding the highest stall margins for the given blade inflow and outflow angles  $\beta_1$  and  $\beta_2$  is selected. From this, the stagger angle  $\gamma$  is then calculated according to

$$\gamma = \beta_1 - \alpha \quad (3.66)$$

after which the axial chord  $c$  and section length are calculated similar to those of the DCA airfoil.

The compressor blade aspect ratio AR can then be determined according to

$$AR = \frac{h_{blade}}{c} \quad (3.67)$$

The blade volume  $V_{blade}$  finally is approximated by a rectangular box, according to

$$V_{blade} = h_{blade} \cdot c \cdot \left(c \cdot \frac{t}{c}\right) \quad (3.68)$$

The blade mass is then estimated by multiplying this volume with the blade material density.

### TURBINE BLADES

The calculation of turbine blades is also done according to the methods of [18], however for this calculation the solidity  $\sigma$  is an output of the calculation instead of a prerequisite. An example of typical turbine blades is given in Figure 3.18.

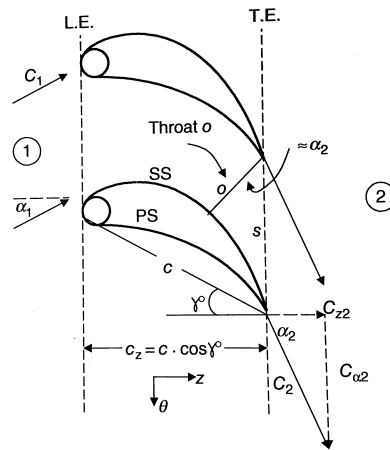


Figure 3.18: Typical turbine blade profile [18, Fig. 10-53]

An “optimal” solidity  $\sigma_{opt}$  is first calculated from an assumed Zweifel coefficient  $\Psi$  and the blade relative in- and outflow angles  $\beta_1$  and  $\beta_2$  using [18, Eq. 10.52]

$$\sigma_{opt} = \frac{2}{\Psi} \cdot \frac{\cos \beta_2}{\cos \beta_1} \cdot \sin |\beta_1 - \beta_2| \quad (3.69)$$

This is then corrected by dividing by the cosine of an estimated stagger angle  $\gamma$  [18, Eq. 10.94] to yield the actual blade solidity  $\sigma$ .

Next, the throat opening size  $o$  is estimated based on the given Reynolds number  $Re_{typ}$ , kinematic viscosity  $\nu$  and relative outflow velocity  $V_2$ , which is then converted to a blade spacing  $s$  by dividing by the cosine of the outflow angle  $\beta_2$ . In case of supersonic outflow ( $M_2 \geq 1$ ), this is multiplied by the sonic throat area ratio  $A^*$ . The chord  $c$  is then estimated by multiplying the solidity  $\sigma$  and the spacing  $s$ , due to the definition of the solidity  $\sigma = \frac{c}{s}$ . Using the stagger angle, the blade chord  $c$  can then be translated into an axial chord  $c_{ax}$ , which leads to a blade row length estimate when taking into account a clearance factor.

Blade aspect ratio AR and volume are determined analogous to those for compressor blades. However, for cooled blades, the volume is decreased by 20% to account for the coolant flow channels within the blades. [15, p. 27]

### 3.11.2. DISK

Disks are used to mount the rotor blades to each other and to the shaft powering them. In order to do this, they are mainly sized based on the blade pull stresses applied to them by the rotating blades, as well as the



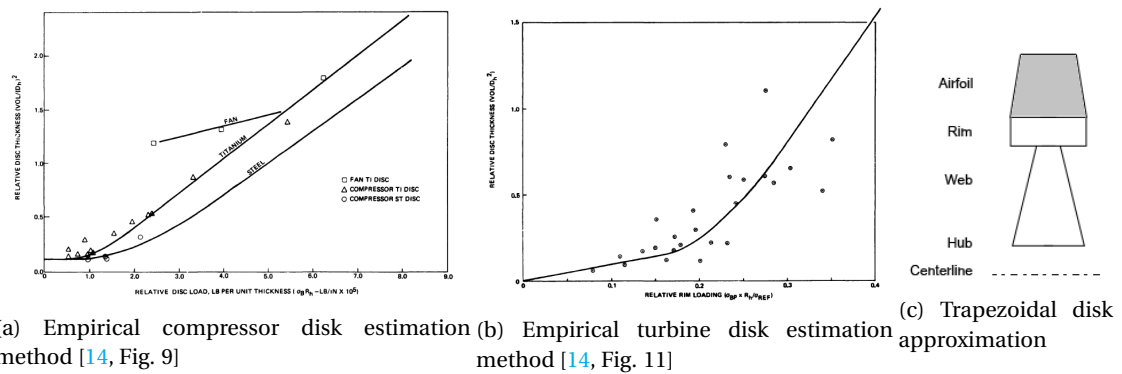


Figure 3.19: Disk approximation methods

internal stresses due to the rotating mass of the disk itself. While several disk design strategies are described in literature (e.g. [25]), for MIME the chart-based methods of [14, Fig. 9, 11] and trapezoidal disk approximation of [15, Sec. 1.2.3] are implemented. These are illustrated in Figure 3.19.

The former is an empirical method, yielding approximations for the disk volume based on its type, material and the product of the blade pull stress  $\sigma_{bp}$  and the blade hub radius  $r_h$ .

The trapezoidal disk approximation is based on a rim of  $10\%r_h$  thick and a trapezoidal web between this rim and the shaft driving the disk. The width of the rim is assumed equal to the axial rotor blade chord, while the trapezoidal web tip and hub widths are sized based on allowable stresses of respectively 75% and 50% of the yield strength.

### 3.11.3. INLET GUIDE VANES

Inlet guide vanes are used to correct the inflow angle of a component if the incoming flow angle does not conform to the required value. Effectively, they are a row of stator blades, and their calculation is therefore done similar to a compressor stator row.

## 3.12. PER-COMPONENT THRUST

Instead of looking at thrust as something only produced in a nozzle, it is also possible to look at the contribution to thrust of each component, in terms of the changes in pressure, area, massflow rate and velocity over that component, as presented in [26, Chapter 20]. This then suggests the component thrust  $F_{comp}$  to be calculated according to

$$F_{comp} = \Delta(V_{ax} \cdot \dot{m}) + \Delta(p \cdot A) \quad (3.70)$$

This works for any straight duct (or constant area  $A$ ). However, when looking at the control volume of for example a compressor, which typically has a smaller exhaust area compared to its inflow area, the resulting forces on the surfaces other than the in- and outflow areas no longer cancel out. Assuming these pressure forces are purely caused by ambient pressure, the solution is then to subtract the ambient pressure  $p_0$  from all the pressure terms, yielding

$$F_{comp} = V_{ax_{out}} \cdot \dot{m}_{out} - V_{ax_{in}} \cdot \dot{m}_{in} + (p_{out} - p_0) \cdot A_{out} - (p_{in} - p_0) \cdot A_{in} \quad (3.71)$$

# 4

## CODE VERIFICATION AND VALIDATION

For any piece of software to be useful in evaluation, it needs to be checked that its outcomes are logical and reasonable, as well as provide a realistic model. The former is called verification, and is described in Section 4.1, while the latter is called validation and is done in Section 4.2.

### 4.1. VERIFICATION

Verification is the process of checking to see that the code is able to handle different inputs, as well as generates data which seems reasonable. Due to the modular nature of the code, this has been a long and continuous process, undertaken mostly during the coding, debugging and rewriting. This is due to the fact that invalid inputs (e.g. complex masses or velocities, which may be caused by invalid outputs of a preceding module) should be caught and fixed in order to prevent cascading effects or complete model failures, while invalid and/or nonsensical outputs usually indicate code bugs or, worse, invalidities of the used methods. As a result, many inputs of both the model and the modules are first checked to see whether they are sensible and otherwise return to a default estimate.

While a full-blown, thorough verification study would methodically test all the extreme combinations of all inputs to try and break the model, the high levels of interactivity as well as the large number of possible combinations of both inputs and modules would make this an extremely lengthy process.

Therefore, as an intermediate solution, most modules are evaluated separately for some characteristic values, to check whether they yield reasonable results. This is done starting with the station property-calculations and the most low-level components (blades and disks) and moving up to the higher-level and/or more complex components.<sup>1</sup>

#### 4.1.1. STATIONS

The stations are mainly used to store the flow properties at a given location, and calculate missing properties based on the known values. As a verification check, a `Station` with standard ISA ambient static conditions ( $T = 288.15$  [K],  $p = 101325$  [Pa]) is created first. The standard dry air composition of 75.47% N<sub>2</sub>, 23.2% O<sub>2</sub>, 1.28% Ar, 0.046% CO<sub>2</sub> and 0% H<sub>2</sub>O by mass is assumed, yielding a specific heat ratio of  $\gamma = 1.4$ , specific gas constant  $R = 287.03$ , sonic velocity of  $a = 340.3$  [ $\frac{m}{s}$ ] and (static) density  $\rho = 1.225$  [ $\frac{kg}{m^3}$ ]. These values are as expected and therefore check out.

Next, axial and whirl velocities of  $V_{ax} = 4$  [ $\frac{m}{s}$ ] and  $C_w = 3$  [ $\frac{m}{s}$ ] are assigned, while keeping the static properties constant, yielding a Mach number of  $M = 0.015$  [-]. Total properties of  $T_t = 288.1624$  [K] and  $p_t = 101340$  [Pa] are then found, both as expected slightly higher compared to the static properties. As expected, the static density remains the same due to the static properties remaining constant, leading to an estimated required flow area of  $A = 1.02$  [m<sup>2</sup>] for a  $\dot{m} = 5$  [ $\frac{kg}{s}$ ] massflow rate, again in line with expectations. For a mean flow annulus radius of  $r_m = 0.5$  [m], this yields annulus hub and tip radii of  $r_h = 0.338$  [m] and  $r_m = 0.662$  [m], as expected.

This data, matching the values as obtained from manual calculations, as well as the general behaviour seen during the method creation, debugging and testing, suggests the internal workings of the stations yields

<sup>1</sup>This approach is chosen due to the more complex components sometimes being made up of lower-level components.

reasonable results.

#### 4.1.2. BLADES

For the blades, performing verification and/or validation through hand calculations is no longer feasible, however still some sanity checks can be made. As computation is done differently for compressor and turbine blades, these are also treated separately here.

##### COMPRESSOR BLADES

For testing purposes, a compressor Blade with the properties of the rotor blade as described in the example on [18, pg. 633]<sup>2</sup> is first created, yielding an axial chord length (the most relevant comparison parameter) of  $c_{ax} = 3.78$  [cm], compared to the  $c_{ax} = 3.44$  [cm] from the example. These values seem to be in reasonable agreement with each other. A blade mass  $m_{blade} = 0.0709$  [kg] is obtained when assuming a blade volume factor  $K = 0.120$  and steel as the blade material, which seems to be a reasonable value for a blade with outer dimensions of  $29.4 \times 67.9 \times 4.4$  [mm] and mounting provisions.

Table 4.1: Compressor blade (supersonic) verification results

Case	$c_{ax}$ [cm]	$m_{blade}$ [kg]
Baseline	3.78	0.0709
110% blade height	3.78	0.0780
110% mean radius	3.78	0.0709
110% $\frac{t}{c}$ -ratio	3.80	0.078
110% solidity	3.79	0.0709
110% Reynolds number	4.16	0.0858
110% kinematic viscosity	4.16	0.0858
110% Mach number	3.78	0.0709
110% inflow velocity	3.44	0.0586
110% inflow angle	3.38	0.0709
110% outflow angle	3.59	0.0709

To evaluate the sensitivity to changes in input, the input values are given modified inputs, the results of which are given in Table 4.1. A 10% increase in blade height yields a similar increase in blade mass, however with a negligible impact on blade chord, while a similar increase in mean radius affects neither of these parameters significantly. This seems logical considering the methods used for blade sizing. A 10% increased thickness-to-chord ratio yields a similar-sized increase in blade mass, however also a small increase ( $\approx 0.5\%$  for this case) in axial chord length. This is due to the thickness-to-chord ratio being used as an approximation for the blade angle of attack, and therefore influencing the stagger angle estimation which in turn affects the axial chord length estimation. In a similar fashion, the solidity input also very slightly influences the axial chord ( $\Delta c_{ax} \approx 0.26\%$  for  $\Delta \sigma = 10\%$ ) due to its role in the deviation angle estimation.

A 10% increase in Reynolds' number yields a logical similar increase in axial chord, but the square of that in terms of mass, which makes sense due to both chord and the (absolute) thickness (through the chord) being estimated based on this parameter, and are therefore both increasing by this factor. The same obviously goes for the kinematic viscosity. A pure increase in relative inflow Mach number (without modifying the velocity, Reynolds' number or viscosity) has virtually no effect on axial chord or blade mass, due to its value being used solely for switching blade shape assumptions and boundary layer thickening. A 10% increase in velocity however decreases the chord length (and therefore thickness) with approximately 9% and the mass with the square of that value. Again, this is logical, as for a similar Reynolds' number and a higher velocity, a smaller typical length will result. Finally, while not affecting the blade mass, the axial chord decreases for an increase in inflow angle (which for this case yields a larger amount of flow turning), while decreasing less for an increased outflow angle (decreasing the amount of flow turning over the blade). This is due to the average of the leading- and trailing-edge angles increasing in both cases, yielding a larger stagger angle and therefore smaller axial chord, even though the blade chord itself remains unchanged.

However, as for a subsonic blade a different calculation path is followed, a blade is also compared with

<sup>2</sup>  $h_{blade} = 2.94$  [cm],  $r_m = 36.77$  [cm],  $\frac{t}{c} = 6.5\%$ ,  $\sigma = 1$ ,  $Re = 3 \cdot 10^5$ ,  $\nu = 8.54 \cdot 10^{-5} \left[ \frac{m^2}{s} \right]$ ,  $M_{in,rel} = 1.12$ ,  $V_{in,rel} = 377.4 \left[ \frac{m}{s} \right]$ ,  $\beta_{in} = 63.43^\circ$ ,  $\beta_{out} = 51.6^\circ$

the (subsonic) stator blade from the same example.<sup>3</sup> This yields a blade axial chord length of  $c_{ax} = 11.5$  [cm] and blade mass of  $m_{blade} = 0.229$  [kg]. Additionally, the found blade angle of attack is  $\gamma = 21.9^\circ$  with a NACA 65-(18)10 profile, matching the outcome in the example. The large chord length is due to the combination of Reynolds' number, viscosity and (lower) velocity, which also explains the relatively high blade mass.

Table 4.2: Compressor blade (subsonic) verification results

Case	$c_{ax}$ [cm]	$m_{blade}$ [kg]
Baseline	11.8	0.2294
110% blade height	11.8	0.2523
110% mean radius	11.8	0.2294
110% $\frac{t}{c}$ -ratio	11.8	0.2523
110% solidity	11.8	0.2294
110% Reynolds number	13.0	0.2776
110% kinematic viscosity	13.0	0.2776
110% Mach number	11.8	0.2294
110% inflow velocity	10.7	0.1896
110% inflow angle	11.9	0.2294
110% outflow angle	11.8	0.2294

As can be seen from Table 4.2, responses to increases in blade height, mean radius, thickness-to-chord ratio, Reynolds' number, viscosity, Mach number and relative velocity are similar to those for the DCA-profile, while changes to solidity result solely in a change to axial chord length due to the different resulting stagger angle. Similarly, an increase in inflow angle or a decrease in outflow angle tends to lead to lower angles of attack. As an increase in inflow angle tends to partially cancel out this lower angle of attack, this effect tends to be most pronounced for a change in outflow angle.

#### TURBINE BLADES

For the turbine blade, again a blade is modelled as described in an example, in this case the first stage stator blade from [18, Example 10.7].<sup>4</sup> This yields a blade with axial chord length  $c_{ax} = 1.1$  [cm], a mass of  $m_{blade} = 0.0107$  [kg] and a calculated solidity of  $\sigma = 1.15$ . While the first two values are not given in the example, it does quote a value for the solidity of  $\sigma = 1.125$ , which seems close enough. The assumed stagger angles are identical at  $\gamma = 40.9^\circ$ , and while the resulting values for the axial chord and mass seem relatively low, this is likely also due to the used Reynolds' number being at the lower end of the typical spectrum.<sup>5</sup>

Table 4.3: Turbine blade verification results

Case	$\sigma$ [-]	$m_{blade}$ [kg]	$c_{ax}$ [cm]	$\gamma$ [deg]
Baseline	1.146	0.0107	1.10	40.89°
110% blade height	1.146	0.0117	1.10	40.89°
110% mean radius	1.146	0.0107	1.10	40.89°
110% Reynolds number	1.146	0.0129	1.21	40.89°
110% kinematic viscosity	1.146	0.0129	1.21	40.89°
110% Mach number	1.146	0.0112	1.13	40.89°
110% specific heat ratio	1.146	0.0107	1.10	40.89°
110% outflow velocity	1.146	0.0088	1.00	40.89°
110% inflow angle	1.146	0.0107	1.10	40.89°
110% outflow angle	1.117	0.0153	1.16	48.32°

As can be seen in Table 4.3, of these parameters, increasing the blade height only significantly affects the blade mass estimate, which follows directly, while increasing the mean radius affects none of these parameters. As for the compressor blades, changing the Reynolds' number or viscosity by themselves purely affects

<sup>3</sup> Blade height, mean radius, Reynolds' number, kinematic viscosity and thickness-to-chord ratio assumed to be the same as those for the rotor,  $\sigma = 1.25$ ,  $M = 0.587$ ,  $V_{rel} = 209.8 \left[ \frac{m}{s} \right]$ ,  $\beta_{in} = 36.4^\circ$ ,  $\beta_{out} = 0^\circ$ .

<sup>4</sup>  $h_{blade} = 3.197$  [cm],  $r_m = 40$  [cm],  $Re = 1 \cdot 10^5$  (assumed),  $\nu \approx 4.78 \cdot 10^{-5} \left[ \frac{m^2}{s} \right]$ ,  $\gamma = 1.3$ ,  $M_{throat,rel} = 1.1$ ,  $V_{throat,rel} = 757.79 \left[ \frac{m}{s} \right]$ ,  $\beta_{in} = 0^\circ$ ,  $\beta_{out} = 60^\circ$

<sup>5</sup> According to [18, Eq. 10.97],  $10^5 \leq Re \leq 10^6$  is typical.

the chord (by an equal factor) and blade mass (by the square of the change). Increasing the throat Mach number does not affect the solidity estimate or stagger angle, but does affect the chord, as the ratio between blade spacing (from which the chord is determined) and throat opening is corrected by the throat area ratio for  $M > 1$ , which is the case for this specific blade. Modifying the specific heat ratio decreases the chord and mass estimate values, however the magnitude of these changes is negligible. Like for the compressor blades, an increased throat velocity causes a decreased blade chord and mass, again due to the chord being determined through the Reynolds' number, albeit less directly for a turbine blade.

Increasing the inflow angle (for this case) yields a lower solidity, axial chord and mass and a higher stagger angle. As the increased inflow angle implies a decreased flow turning over the blade and a higher average flow angle, the flow angles directly affect the solidity estimate, the effect of which then cascades through the other parameters. Increasing the outflow angle has a similar effect on the solidity and stagger angle, however leads to an increase in mass and axial chord, which makes sense, as the amount of flow turning required now increases.

### 4.1.3. DISKS

To verify the disk, a disk is designed for the compressor rotor blade used for the blade verification. From its source [18, Example 8.8], it can be found that this disk spins with an RPM of  $8770 \left[ \frac{\text{rev}}{\text{min}} \right]$  and has an outer diameter (equal to the blade hub diameter) of  $D = 0.706 \text{ [m]}$ . The disk is assumed to be made of steel, and to contain 33 blades. Using Equation 3.28, a blade pull stress of  $\sigma_{\text{bp}} = 4.818 \text{ [MPa]}$  is assumed. This results in a rotor disk with a mass of  $m_{\text{disk}} = 12.02 \text{ [kg]}$ . This seems a reasonable estimate.

Table 4.4: Disk verification results

Case	$m_{\text{disk}} \text{ [kg]}$
Baseline	12.02
110% RPM <sup>6</sup>	12.02
110% blade pull stress	12.03
110% diameter	14.66
Titanium	6.58

As shown in Table 4.4, increasing the RPM, blade pull stress and/or diameter all yield increases in mass of the disk, while switching to titanium as the material almost halves the blade mass, due to the decreased density and increased allowable stresses.

### 4.1.4. INLETS

In order to verify the inlet calculation, an Inlet is created with a 1 [m] outflow diameter, 95% efficiency and a specified outflow Mach number of  $M_{\text{out}} = 0.6$ . It is then fed a flow containing an ambient conditions station with ISA sea-level conditions and a velocity of  $V = 280 \left[ \frac{\text{m}}{\text{s}} \right]$ . From this, it generates an outflow station with equal total temperature and a total pressure which is 2.35% lower, corresponding to the outcome of Equation 3.43. This makes sense, as no energy is added or removed, from which it follows that the total temperature cannot change, leaving a decrease of total pressure as the only option to account for the inlet efficiency. Massflow remains constant, while velocity decreases, static density increases and area decreases.

Table 4.5: Inlet verification results

Case	$T_{\text{out}} \text{ [K]}$	$p_{\text{out}} \text{ [Pa]}$	$\dot{m} \left[ \frac{\text{kg}}{\text{s}} \right]$	$V_{\text{out}} \left[ \frac{\text{m}}{\text{s}} \right]$	$\rho_{\text{out}} \left[ \frac{\text{kg}}{\text{m}^3} \right]$	$A_{\text{out}} \text{ [m}^2\text{]}$
Baseline	288.1	98944.1	149.5	197.2	1.005	0.754
90% efficiency	288.1	96603.4	146.0	197.2	0.982	0.754
110% outflow diameter	288.1	98944.1	180.9	197.2	1.005	0.912
110% outflow Mach	288.1	98944.1	157.7	215.4	0.9708	0.754

For a decreased efficiency, as can be seen in Table 4.5, the total outflow pressure decreases, as does the massflow. For an increase in outflow diameter, the massflow and area increase quadratically, while the other

<sup>6</sup>The RPM increase in Table 4.4 is a pure increase in RPM, so assuming all other inputs on the disk remain constant. In reality, for the same blade, an increased RPM will also increase blade pull stress and therefore lead to a heavier blade.

properties remain constant, as expected. For an increase in outflow Mach number, the massflow also increases, while the static properties decrease. Again, this is expected for an increased axial velocity and constant total properties. All considered, this seems to indicate the inlet functions as desired.

#### 4.1.5. DUCTS

Creating a new Duct with pressure ratio<sup>7</sup>  $PR_{\text{duct}} = 1$  and feeding it with a consistent inflow station yields an effectively renamed and renumbered copy of the inflow station, as expected. However, when reducing the pressure ratio to  $PR_{\text{duct}} = 95\%$ , the total and static pressures drop accordingly, resulting in a lower density and higher through-flow area, as expected for a velocity which remains constant.

If (for the  $PR_{\text{duct}} = 1$ ) a required outflow mean radius is imposed, the outflow station radii change accordingly, with all other properties remaining constant. If however a required outflow axial velocity is imposed which is larger than the inflow axial velocity, the static properties drop, while the total properties remain unaffected, as they should. The combination of decreased static density and increased axial velocity leads to a lower outflow area, implying a convergent duct, consistent with general subsonic flow theory. While the mean radius remains unaffected (unless also imposed), the hub and tip outflow radii respectively increase and decrease to match.

#### 4.1.6. NOZZLES

For the nozzle verification, a Nozzle with a 95% efficiency is created and fed a Flow containing an ambient Station with sea-level conditions and an axial velocity of  $V_{\text{ax}} = 280 \left[ \frac{\text{m}}{\text{s}} \right]$ , as well as a nozzle inflow Station with equal total temperature, 30% higher total pressure, a massflow rate of  $\dot{m} = 150 \left[ \frac{\text{kg}}{\text{s}} \right]$ , a purely axial velocity of  $V_{\text{ax}} = 150 \left[ \frac{\text{m}}{\text{s}} \right]$  and a mean radius  $r_{\text{m}} = 0.3 \text{ [m]}$ . Based on these inputs, it finds a choked nozzle, generating net and gross thrust values of  $F = 8.76 \text{ [kN]}$  and  $F_{\text{g}} = 50.76 \text{ [kN]}$ . This implies that the difference between these two must be equal to  $\dot{m} \cdot V_{\text{amb}} = 280 \cdot 150 = 42 \text{ [kN]}$ , which checks out. The net thrust must then be composed of a pressure and a velocity component. The former is found to be slight as  $F_{\text{p}} = (p_{\text{out}} - p_{\text{amb}}) \cdot A_{\text{out}} = (104624 - 101325) \cdot 0.339 = 1.12 \text{ [kN]}$ , while the jet net thrust should then be equal to  $F_{\text{j}} = \dot{m} \cdot (V_{\text{out}} - V_{\text{amb}}) = 150 \cdot (330.9 - 280) = 7.635 \text{ [kN]}$ . Again, this checks out.

Assigning a lower efficiency to the nozzle immediately leads to a lower produced thrust, again as expected.

#### 4.1.7. COMBUSTORS

The combustor is verified, like many others, by checking whether the observed trends make sense. In order to do this, a baseline Combustor is created, initially set to provide a  $T_{\text{t,out}} = 1000 \text{ [K]}$  outflow total temperature with an efficiency of  $\eta = 0.95$  and a total pressure ratio of  $PR = 0.95$ . This object is then fed an inflow station with ISA sea-level ambient conditions, purely axial velocity of  $V_{\text{ax}} = 150 \left[ \frac{\text{m}}{\text{s}} \right]$ , massflow rate of  $\dot{m}_{\text{in}} = 150 \left[ \frac{\text{kg}}{\text{s}} \right]$  and mean radius of  $r_{\text{m}} = 0.5 \text{ [m]}$ . From these inputs, a combustor with a mass of  $m = 20.84 \text{ [kg]}$  and a length of  $l = 0.23 \text{ [m]}$  results, requiring a fuel massflow rate of  $\dot{m}_{\text{f}} = 3.12 \left[ \frac{\text{kg}}{\text{s}} \right]$  with an axial outflow velocity of  $V_{\text{ax,out}} = 801.8 \left[ \frac{\text{m}}{\text{s}} \right]$ .

Table 4.6: Combustor verification results

Case	$m_{\text{comb}} \text{ [kg]}$	$\dot{m}_{\text{f}} \left[ \frac{\text{kg}}{\text{s}} \right]$	$l_{\text{comb}} \text{ [m]}$	$V_{\text{out}} \left[ \frac{\text{m}}{\text{s}} \right]$
Baseline	21.0	3.123	0.226	301.8
110% outflow total temperature	21.8	3.643	0.233	315.6
110% outflow "residence time"	23.1	3.123	0.249	301.8
90% pressure ratio	21.3	3.123	0.226	301.8
90% efficiency	21.0	3.484	0.226	301.7

As shown in Table 4.6, a 10% increase in required outflow total temperature increases the required fuel massflow rate with approximately 17%, which seems logical, as not only the inflow gas needs to be heated further, but also the added fuel requires an increased amount of energy to heat up, both due to the higher temperature as well as due to the increased fuel massflow, creating a snowball effect. The combustor outflow axial velocity also increases, as it is calculated based on a given outflow Mach number, which yields a higher

<sup>7</sup>The duct pressure ratio is the mechanism used for taking into account possible losses over the duct. It is therefore *not* an isentropic pressure ratio.



velocity for a higher temperature and associated sonic velocity. This then logically translates into a larger length (as this is based on the “residence time” and the average of the in- and outflow velocities), which in turn leads to a larger mass, which is also due to the larger outflow area (due to the larger massflow and lower static density). As this “residence time” parameter<sup>8</sup> is only used for estimating combustor length, it is therefore seen to only affect the length (and consequently mass) estimates in a direct relation.

Decreasing the total pressure ratio over the combustor only affects the mass, besides the outflow total and static conditions, which increases slightly due to the larger outflow diameter, which is a consequence of the lower static outflow density. Similarly, decreasing the efficiency mostly only affects the required fuel massflow rate. While the change in fuel massflow rate is non-negligible, the change to the overall massflow is in the order of 0.1%, and therefore it is entirely according to expectations that no significant changes to the other outflow properties are observed.

Again, all of this behaviour is explainable and as expected, implicitly verifying the combustor calculations.

#### 4.1.8. COMPRESSOR STAGES

For the compressor stage verification, a single stage is created and fed a flow containing an inflow station with ambient conditions and a  $V_{ax} = 150 \left[ \frac{m}{s} \right]$  velocity for a massflow of  $\dot{m} = 150 \left[ \frac{kg}{s} \right]$  with a mean radius of  $r_{mean} = 0.5 [m]$ . Also, the spool speed is set to  $RPM = 4000 \left[ \frac{rev}{min} \right]$  and a shaft diameter of  $D_{shaft} = 0.3 [m]$  is provided. The stage itself is created with an efficiency of  $\eta = 0.95$ , work coefficient  $\lambda = 0.7$  and degree of reaction  $\Lambda = 0.5$ . It is assumed to be neither a first nor a last stage.

The result is a compressor stage with a pressure ratio of  $PR_{stage} = 1.40$ , requiring  $P_{req} = 4.61 [MW]$  of power, consisting of 166 stator and 107 rotor blades (mean-line solidities of respectively  $\sigma_s = 3.54$  and  $\sigma_r = 2.00$ ) and having an estimated mass of  $m_{stage} = 180.4 [kg]$ .

Table 4.7: Axial compressor stage verification results

Case	PR [-]	$P [MW]$	$N_{blade_s} [-]$	$N_{blade_r} [-]$	$\sigma_s [-]$	$\sigma_r [-]$	$m_{stage} [kg]$
Baseline	1.40	4.61	166	107	3.54	2.00	180.4
110% massflow	1.40	5.07	166	107	3.54	2.00	195.1
110% mean radius	1.33	3.88	166	107	2.56	1.64	194.7
110% inflow axial velocity	1.40	4.61	166	107	2.48	1.61	150.2
90% efficiency	1.35	4.61	166	107	3.54	2.00	191.6
110% work coefficient	1.44	5.07	166	107	5.68	2.23	170.4
110% spool speed	1.33	3.88	166	107	2.56	1.64	209.8

For an increased massflow, the pressure ratio, blade numbers and solidities remain equal, as expected, however both the required power and stage mass increase. Due to the increased amount of air being compressed, this is logical, as the blade height and mass increase, in turn increasing the blade pull stresses, as well as casing masses. An increase in provided mean inflow radius also increases the stage mass, however decreases the stage pressure ratio and power consumption. This is due to the actual rotor mean radius being decreased due to allowable limits for certain parameters being exceeded.

An increase in inflow axial velocity does not affect the pressure ratio or power consumption of the stage, nor the number of blades, however it does lower the resulting solidities and the stage mass. The latter is likely due to the decreased passage height (and therefore blade size) associated with the same massflow at a higher velocity.

A decrease in the assumed stage efficiency, while not affecting the stage power requirements, blade numbers or solidities, does affect the resulting pressure ratio and stage mass, both adversely. The former can be explained by a lower efficiency causing less pressure rise for the same change in flow velocity, while the latter is a logical consequence of the lower gas outflow density due to this fact, implying a larger outflow passage dimension and therefore heavier blades as well as bigger casings. An increased work coefficient however does cause an increase in the pressure ratio, as well as an increased required power consumption. The mass decreases, likely due to the lower outflow passage height.

Finally, an increase in the provided RPM decreases the obtained pressure ratio and power consumption and increases the stage mass. The former is again due to a flow-coefficient-limit-imposed decrease in mean

<sup>8</sup>N.B. Not the actual residence time in the current module implementation, just the time taken to traverse the combustor length with the average of the in- and outflow velocities. Implementing the actual residence time requires a more detailed combustor model and is left to a future contributor.

radius, while the latter is due to the increased blade pull stresses due to the larger rotational velocity and the additional case masses due to the additional blade kinetic energy which needs to be contained.

#### 4.1.9. TURBINE STAGES

For the turbine stage verification, a single turbine stage is set up, similar to that for the compressor stage verification. The inflow conditions are however changed to a total pressure of  $p_{\text{tin}} = 5$  [bar] and  $T_{\text{tin}} = 546.3$  [K]. The used work coefficient is however now initially assumed to be  $\lambda = 1$ . This yields a stage generating  $P_{\text{gen}} = 6.58$  [MW], with 478 stator and 448 rotor blades (mean-line solidities of respectively  $\sigma_s = 1.15$  and  $\sigma_r = 1.15$ ) and a total stage mass of  $m_{\text{stage}} = 1563$  [kg].

Table 4.8: Axial turbine stage verification results

Case	$P$ [MW]	$N_{\text{blade}_s}$ [-]	$N_{\text{blade}_r}$ [-]	$\sigma_s$ [-]	$\sigma_r$ [-]	$m_{\text{stage}}$ [kg]
Baseline	6.58	478	448	1.15	1.15	1563.1
110% massflow	7.24	478	448	1.15	1.15	1689.5
110% mean radius	7.96	517	477	1.15	1.15	2289.8
110% inflow axial velocity	6.58	522	488	1.15	1.15	1448.5
90% efficiency	6.58	478	460	1.15	1.15	1524.8
110% spool speed	7.96	470	434	1.15	1.15	1998.5
110% work coefficient	7.24	474	441	1.15	1.19	1644.4

From Table 4.8, it can be seen that, as for the compressor stage, increasing the massflow purely increases the generated power and stage mass, as expected for the same reasons. Increasing the mean inflow radius also increases the generated power (as expected due to the higher local rotor velocity, which increases the absolute velocity jump due to its role in the non-dimensionalisation of the coefficients such as  $\lambda$ ) and stage mass, both due to the increased number of blades as well as the increased stage size and associated blade-pull stresses and casing areas. On the other hand, increasing the axial velocity does not influence the power generation, however does change the blade numbers and stage mass, increasing the former while decreasing the latter. This is due to the dependence of both on the chord length, which is determined based on a typical Reynolds' number, and therefore decreases with increasing velocity.

A decrease in stage efficiency yields an increase in the number of rotor blades, as well as a decrease in the stage mass, while keeping the same generated power. Again, a lower efficiency leads to a higher pressure drop over the rotor, in turn causing a lower outflow density. For an increased spool speed, the generated power increases due to the same reasoning as presented for the inflow radius. Due to the increased encountered velocities over the blades combined with the increased temperature and pressure drops, the blade chords will increase only slightly, decreasing the blade numbers, while the stage mass increases, primarily due to the increased blade pull stress induced by the higher spool speed.

When the shaft diameter is increased above the originally found hub diameter, the rotor mean radius is updated to still allow for the shaft to pass within the blade hub, increasing the rotor mean radius and therefore generating the same effects, as expected. Finally, increasing the work coefficient increases the produced power, and slightly reduces the blade count, while increasing the stage mass, as seems intuitive.

As indicated, this behaviour is explainable and as expected, therefore indicating a code which is functioning correctly.

#### 4.1.10. COMPRESSORS

Assuming proper functioning of the compressor stages, as can be assumed from the stage verification done earlier, the proper workings of the compressor is now verified, again by evaluation of an arbitrary compressor and checking the trends for changes to the inputs. Therefore, a baseline compressor is created, with a set efficiency of  $\eta = 0.9$  and an overall total pressure ratio of  $PR = 5$ . This is then fed a Flow containing an inflow station with ISA sea-level ambient conditions, purely axial velocity of  $V_{\text{ax}} = 150$  [ $\frac{\text{m}}{\text{s}}$ ], massflow rate of  $\dot{m}_{\text{in}} = 150$  [ $\frac{\text{kg}}{\text{s}}$ ] and mean radius of  $r_m = 0.5$  [m] as well as a Shaft running at  $3000$  [ $\frac{\text{rev}}{\text{min}}$ ]. This yields a 7-stage compressor with a length of  $l = 67.8$  [cm] and mass of  $m = 698.6$  [kg], consuming 28.90 [MW], providing a fivefold pressure increase with an associated outflow total temperature of  $T_{\text{out}} = 478.7$  [K].

When bumping up the efficiency to  $\eta = 95\%$ , all of these properties decrease significantly, providing a smaller, lighter, less powerfull and lower loss-inducing machine, as expected for an increased efficiency. However, when a total pressure ratio of  $PR = 6.5$  is prescribed, the mass, length, required power and number of



Table 4.9: Axial compressor verification results

Case	$m_{\text{comp}}$ [kg]	$l_{\text{comp}}$ [m]	$P$ [MW]	$N_{\text{stage}}$ [-]
Baseline	699	0.678	28.9	7
$\eta = 95\%$	582	0.590	26.8	6
PR = 5.5	665	0.644	31.0	7
PR = 6.5	715	0.706	35.1	8

stages all increase, again as expected. For some intermediate values however (as can be seen in Table 4.9), the mass decreases for increasing pressure ratio while the number of stages has not yet increased, which is explainable again through the chord-length determination. For higher pressure ratios and an equal number of stages, higher work coefficients are required, implying more flow turning and therefore higher local velocities, and thus shortening chord lengths for equal Reynolds numbers, influencing blade and disc masses. Also, local flow coefficient and/or DeHaller limits may have been reached, implying resizing of the mean radius and/or axial velocity at that location, again affecting both mass and length.

This shows behaviour of the compressor is in line with the expectations and intuitive trends, and explainable when otherwise.

#### 4.1.11. TURBINES

The verification process for turbines also relies on the output responses to design input changes for an arbitrary turbine. For this purpose, a generic turbine is created with a 90% efficiency. This is then supplied with a Flow containing an inflow Station with a total pressure of  $p_t = 10$  [bar], total temperature  $T_t = 1000$  [K], purely axial velocity of  $V_{\text{ax}} = 150$  [ $\frac{\text{m}}{\text{s}}$ ], massflow rate of  $\dot{m}_{\text{in}} = 150$  [ $\frac{\text{kg}}{\text{s}}$ ] and mean radius of  $r_m = 0.5$  [m]. It is also supplied with a Shaft with  $3000$  [ $\frac{\text{rev}}{\text{min}}$ ] to be driven, containing a Shaftstation of a  $P = 10$  [MW] consumer. This results in a single-stage turbine with an estimated length of  $l = 53.9$  [cm], mass of  $m = 2048$  [kg] and total outflow temperature of  $T_{t,\text{out}} = 941.2$  [K]. While this is definitely too large and heavy for a single-stage machine, this is a known issue, with most of the length being due to a long inflow correction duct for the mean radius, and the mass being mostly due to the heavy disc, caused by high blade pull stresses.

Table 4.10: Axial turbine verification results

Case	$m_{\text{turb}}$ [kg]	$l_{\text{turb}}$ [m]	$P$ [MW]	$N_{\text{stage}}$ [-]	$T_{t,\text{out}}$ [K]
Baseline	2049	0.539	10.0	1	941
200% power	4577	0.546	20.0	2	882
110% massflow	2082	0.554	10.0	1	947
110% mean radius	2039	0.489	10.0	1	941
110% inflow velocity	2049	0.539	10.0	1	941

Doubling the required power to  $P_{\text{req}} = 20$  [MW] also causes the turbine to switch to a two-stage design. While the length changes only marginally (due to the majority of the length being taken up by the duct), the mass does more than double. The outflow total temperature decreases, as expected from a higher energy offtake from the flow. Increasing the massflow rate through the turbine also increases the mass and length, both due to the larger passage height, while decreasing the total temperature decrease over the turbine. This is logical, as less energy has to be removed per kg of air for a higher massflow rate and equal power generation. As the inflow mean radius is overridden by the code due to the exceedence of certain criteria, an increase to the inflow mean radius causes only a significant change to the total turbine length, as the duct has to take up less of this discrepancy. For the same override reasons, varying the axial inflow velocity also does not affect the turbine properties in any significant way. In this case, the length is also unaffected, as the sizing of the duct length for radius correction is more critical in this case.

Summarising, while the turbine model does show signs of excessive mean radii, the other trends and thermodynamic performance do seem on par.

#### 4.1.12. FANS

Verification of the fan is also done through the checking of the effects of input variations on output parameters. For this, a fan is created with a bypass ratio of  $\text{BPR} = 5$ , outflow Mach number  $M_{\text{out}} = 0.3$ ,  $\eta = 90\%$

efficiency and a pressure ratio of  $PR = 1.3$ . A  $\dot{m} = 150 \left[ \frac{\text{kg}}{\text{s}} \right]$  massflow inflow Station is provided with ambient ISA sea-level conditions, an axial inflow velocity of  $V_{\text{axin}} = 50 \left[ \frac{\text{m}}{\text{s}} \right]$  and mean inflow radius  $r_m = 0.5 \text{ [m]}$ . This results in a 655 [kg] fan with core and bypass flowpath lengths of  $l_c = 35.9 \text{ [cm]}$  and  $l_b = 41.7 \text{ [cm]}$  and requiring a power of  $P_{\text{req}} = 3.77 \text{ [MW]}$ .

For an increased inflow massflow, the fan mass and flowpath lengths increase, as does the required power. All of this makes intuitive sense, as more air needs to be processed, requiring more power and a larger machine. A bigger machine can also be realised by enlarging the mean inflow radius, which however now causes a shorter and lighter design, with the power requirement remaining equal. This is caused by the annulus height decreasing for increasing mean radius and constant massflow rate, leading to smaller and lighter blades. A similar effect is seen when increasing the inflow axial velocity, which also causes a smaller annulus height through the reduced flow area.

When increasing the pressure ratio, the mass and power consumption increase as well, as expected. A slight decrease in length is also observed, which may be explained through the increased density and therefore smaller blades. An increase in bypass ratio causes a small increase in mass, as well as very small changes to the core and bypass flow path lengths, again due to forementioned reasons. Power consumption does not change, as the same total mass is still compressed with the same ratio.

As expected, an increase in fan efficiency causes a lighter and (slightly) shorter machine, which above all consumes less power. Finally, increasing the outflow Mach number decreases the fan mass, due to the smaller stator blades.

All in all, the fan module seems to behave as expected.

#### 4.1.13. SHAFTS

As the shaft is a relatively simple component in terms of coding, its verification is also relatively straightforward. As the power distribution is handled by the components which are coupled to the shaft, the only calculations performed by the shaft are the calculation of the new spool speed, the dimensioning and mass estimation. In order to verify this, a test Shaft is created, which is given the standard speed of  $RPM = 3000 \left[ \frac{\text{rev}}{\text{min}} \right]$  and initially assumed to be the innermost shaft, with an inner-to-outer-radius ratio of  $\frac{r_i}{r_o} = \frac{3}{4}$ . Two stations are added to this shaft, each supplying or consuming  $P = 10 \text{ [MW]}$  of power and having different limit spool speeds. A Flow with corresponding stations was also created to allow for the calculation of the  $x$ -locations of the shaft stations. Running the design calculation reset the spool speed to the lowest of the limits, and provided the shaft length as the difference between the two shaft stations in  $x$ -direction, as expected. The created steel shaft got outer and inner diameters of respectively  $D_o = 12.2 \text{ [cm]}$  and  $D_i = 9.9 \text{ [cm]}$ , yielding a shaft wall thickness of  $t = 1.65 \text{ [cm]}$  to convey the given power at the given speed over the calculated length, yielding a shaft mass of  $m = 23.9 \text{ [kg]}$  for a shaft length of  $l = 0.5 \text{ [m]}$ . Running the calculation for a similar but non-innermost shaft using this results as the prescribed inner diameter, yields the exact same answers for the thickness and outer diameter, implying at least a consistent calculation.

Running the calculations for an increased RPM reduces the diameters of the resulting shaft for a fixed radius ratio, which is as expected, as for equal power, an increased angular velocity should yield a decrease in torque, reducing required material thickness for equal stress levels. For a prescribed inner diameter, an increase in this prescribed value yields a lower thickness, which is again logical due to torque being a product of radius and force. Also intuitively, increasing the amount of transmitted power increases the required thickness and/or radii, depending on the prescribed properties.

#### 4.1.14. FLOWS

As the Flow object only performs calculations when modelling a bleed flow, and is considered as a pure container otherwise, the bleed air redistribution method is the only one which needs verification. In order to do this, a Flow with 4 Stations is created, two of which are bleed air suppliers and two of which are consumers. Of the two suppliers, both are capable of supplying cooling flow at sufficiently low temperatures, however only one has sufficient pressure to supply both of the consumers. After running the bleed calculation method, all the required bleed air offtake is sourced from the final (highest-pressure) bleed air supply station, with the bleed air total temperature of this supply station being transferred to the demand stations, as was intended, suggesting correct operation.

## 4.2. VALIDATION

Validation of a modular model is preferably done in two phases: first on a module-by-module basis, validating the outcomes of the individual modules, followed by validation of the full model, validating the interaction between the modules. However, due to a lack of validation data for the individual modules combined with the high levels of interactivity between them, the first phase is not feasible in this case. Therefore, the used method of validation for MIME is based purely on the comparison of the outcomes with known data for several engines of different types.

### 4.2.1. METHODS

In order to validate the MIME method, several test cases are evaluated for their on-design<sup>9</sup> performance. In order to be able to purely evaluate the engine model performance, the validation is performed on an isolated version<sup>10</sup> of MIME.

For each of the test cases, the required inputs are first set up based on the known information. The model is then run, after which the engine physical and performance characteristics are obtained, as well as the evaluation time for the model runs. The results are then compared to the known engine data in Section 4.2.3.

### 4.2.2. TESTCASES

In order to test the functioning of the code for various cases, it was decided to test the code for three different engines. Each of these cases was chosen to determine a specific performance criterion.

The first selected testcase is the CFM56-3B, as the CFM56 is a standard turbofan and a relatively common engine, being a popular engine option on both the Boeing 737 and the Airbus A320 families. Its evaluation should provide a baseline. The second testcase is the Rolls-Royce Trent 556, as found on the Airbus A340. While it is still a standard turbofan, it does have a different architecture, being a 3-spool engine. The final testcase is the Bristol Siddeley Orpheus engine. Even though it is quite old, it still provides interesting information on versatility, as it is a turbojet, and therefore well and truly different in architecture from the other two.

#### TESTCASE 1: CFM INTERNATIONAL CFM56-3B1

The CFM International CFM56 is a very common turbofan engine, which has powered aircraft ranging from the Douglas DC-8 up to the Airbus A320, Boeing 737NG and KC-135R Stratotanker. The specific version used for this testcase is the CFM56-3B1, which saw use on the Boeing 737-300 and -400.

**CFM56 validation data** The CFM56-3B1 is a twin-spool high-bypass axial throughflow turbofan engine with an annular combustion chamber. It consists of a single-stage fan, a 3-stage low-pressure compressor (LPC), a 9-stage high-pressure compressor (HPC), a single-stage high-pressure turbine (HPT) and a 4-stage low-pressure turbine (LPT). Its dimensions are  $2364 \times 2018 \times 1817$  [mm], with a dry-engine mass of 1966 [kg]. [27] The fan consists of 38 titanium blades, has a diameter of 1524 [mm] and a maximum RPM of  $5490 \left[ \frac{\text{rev}}{\text{min}} \right]$ . The LPC and HPC blades are also titanium, except for the rotor blades of the final 6 HPC stages and all the HPC stator blades, which are made of steel. The maximum HPC RPM is  $15183 \left[ \frac{\text{rev}}{\text{min}} \right]$ . The combustor is fully annular. The HPT has a single stage, with both stator and rotor being air-cooled, while the LPT consists of 4 stages. The engine has a bypass ratio of  $\text{BPR} = 5.0$ , an overall pressure ratio of  $\text{OPR} = 27.5$ , a maximum cruise thrust  $F_{\text{max}} = 4650$  [lb]  $\approx 20.69$  [kN] and a cruise specific fuel consumption of  $\text{SFC} = 18.55 \left[ \frac{\text{g}}{\text{kN}\cdot\text{s}} \right]$ .<sup>11</sup>

**CFM56 inputs** The MIME model requires a minimum set of inputs for each component. All of these inputs for the CFM56 (as well as for the other testcase engines) are presented in Table 4.11.

As can be seen when comparing this table to the validation data presented previously, not all data was known. A CFM56-3  $\text{PR}_{\text{fan}} = 1.64$  was estimated based on a less reliable source<sup>12</sup>, however determining the LPC and HPC pressure ratios is more difficult. Therefore, these are estimated through the assumption of a constant total temperature increase per stage, which is then transformed back based on the number of stages per compressor. The turbine inlet temperature was also estimated based on the same source, for lack of a

<sup>9</sup>The exception is the last testcase, as only the take-off performance (at sea-level) is known. Therefore, the model is instead used to evaluate the take-off performance and engine characteristics for this case.

<sup>10</sup>The model is called directly instead of through its Initiator implementations.

<sup>11</sup>Source: [https://janex.ihs.com/Janes/Display/jae\\_0631-jae](https://janex.ihs.com/Janes/Display/jae_0631-jae), retrieved on 2018-04-20

<sup>12</sup>Source: <http://www.propfan.net/turbodir.html>, retrieved on 2018-04-27

Parameter	CFM56-3B1	Trent 556	Orpheus Mk.803	Unit
$D$	1.524	2.474	0.823	[m]
$h$	10667	10667	0	[m]
$M$	0.8	0.82	0.6	[-]
$PR_{fan_c}$	1.64	1.48	N/A	[-]
$PR_{fan_{bp}}$	1.64	1.48	N/A	[-]
$PR_{LPC}$	2.57	N/A	N/A	[-]
$PR_{IPC}$	N/A	8.66	N/A	[-]
$PR_{HPC}$	6.53	2.82	4.14	[-]
$T_{t_4}$	1535	1500	913	[K]
$PR_{cc}$	0.95	0.95	0.95	[-]
BPR	5	7.6	N/A	[-]

Table 4.11: Testcase-engine inputs

better source. The combustor pressure ratio was unknown, and therefore an assumption was made of a 5% loss in total pressure over the combustor.

Even though MIME does allow for specific efficiencies for specific components to be set, however, as no information is known about the individual component efficiencies, no input is provided for these, leaving MIME to default to its internal default values.

### TESTCASE 2: ROLLS-ROYCE TRENT 556

The Rolls-Royce Trent 556 is an engine from the Trent 500 three-spool engine series, which is powering the Airbus A340-600 aircraft. Again, the found validation data is presented first, followed by the inputs deduced from it.

**Trent 556 validation data** As stated, the Trent 556 is a three-spool, high-bypass axial throughflow turbofan engine. Its fan has a diameter of 2474 [mm] with a bypass ratio of  $BPR = 7.6$ , yielding a massflow rate of  $\dot{m} = 879.5 \frac{\text{kg}}{\text{s}}$ . Unlike the CFM56, the post-fan compressor is mounted on a separate shaft from the fan and is therefore dubbed the Intermediate-pressure compressor, consisting of 8 stages. The high-pressure compressor consists of 6 stages, yielding an overall pressure ratio of  $OPR = 36.3$ . The HPT consists of a single stage with cooled CMSX-4 single-crystal blades, while the single-stage IPT only has stator blades made of CMSX-4. The low-pressure turbine has 5 stages. The engine has a dry mass of 4835 [kg] and a cruise thrust of  $F = 47.71$  [kN], along with a cruise specific fuel consumption  $SFC_{cruise} = 15.26 \left[ \frac{\text{g}}{\text{kN}\cdot\text{s}} \right]$ .<sup>13</sup>

**Trent 556 inputs** As pressure ratios over the individual compressor (nor over the fan) are known, these are estimated based on the same constant total temperature jump assumption as used for the CFM56. Another unknown for the Trent 556 engine is the cruise turbine inlet total temperature  $T_{t_4}$ , which is therefore estimated to be approximately  $T_{t_4} = 1500$  [K]. Again, no information is known about the individual component efficiencies, so no input is provided for these, leaving MIME to default to its internal default values.

### TESTCASE 3: BRISTOL SIDDELEY ORPHEUS Mk.803

The Bristol Siddeley Orpheus is a single-shaft turbojet, of which the BOr.3 or Mk 803 version was used to power the Fiat G91. As for the previous cases, after the presentation of the obtained validation data from literature, the used inputs of both models are presented.

**Orpheus validation data** The Orpheus Mk 803 has a single shaft, to which a seven-stage axial compressor is mounted, with the stator and rotor blades being made of steel or aluminium alloy, and a massflow of  $\dot{m} = 39.4 \left[ \frac{\text{kg}}{\text{s}} \right]$  passing through. The combustor is a can-annular type with seven steel cans. The turbine consists of a single stage, containing 125 rotor blades. The engine has a diameter of  $D = 823$  [mm], a length of  $l = 1916$  [mm] and a dry mass of  $m = 374$  [kg]. Only take-off performance parameters are known, yielding a thrust of  $F = 22.24$  [kN] and associated specific fuel consumption of  $SFC = 30.59 \left[ \frac{\text{g}}{\text{kN}\cdot\text{s}} \right]$  at ISA sea-level conditions.<sup>14</sup> According to [28], its compressor pressure ratio is  $PR = 4.14$ .

<sup>13</sup>Source: [https://janex.ihs.com/Janex/Display/jae\\_a003-jae](https://janex.ihs.com/Janex/Display/jae_a003-jae), retrieved on 2018-04-20

<sup>14</sup>Source: [https://janex.ihs.com/Janex/Display/jae\\_0250-jae](https://janex.ihs.com/Janex/Display/jae_0250-jae), retrieved on 2018-04-20

**Orpheus inputs** For the Orpheus, several issues arose when creating the inputs. While the sea-level height is not a problem, the method used by MIME defines the massflow through the Mach number and area at the inlet outflow through  $\dot{m} = A \cdot \rho \cdot V$  and then back-translating this into a required intake area for the ambient velocity, which causes errors if this ambient velocity  $V = 0$ . Therefore the Mach number was set to  $M = 0.6$  instead of  $M = 0$ , the new value chosen to reflect the default post-inlet Mach number setting of MIME.

### 4.2.3. RESULTS

The model runs described earlier yield results which can be evaluated in several ways. In this section, the absolute performance of the MIME code is described, and the observed errors explained.

As described earlier, the absolute performance of the MIME model is evaluated for three different engines. The found discrepancies are then explained as far as possible.

The outcomes of the MIME model for the three testcase engines are presented in Table 4.12. From this table, it becomes clear that the thrust is generally overestimated, with the SFC estimate being reasonably accurate for all except the Orpheus. The mass estimates can be seen to be mostly grossly out of line.

	$F$ [kN]			SFC [ $\frac{\text{g}}{\text{kN}\cdot\text{s}}$ ]			$m$ [kg]		
	Ref	MIME	$\epsilon$	Ref	MIME	$\epsilon$	Ref	MIME	$\epsilon$
CFM56-3B1	20.7	23.6	+14.0%	18.6	15.0	-19.4%	1966	5144	+161.6%
Trent 556	47.7	58.1	+21.7%	15.3	16.6	+8.4%	4835	5695	+17.8%
Orpheus Mk.803	22.2	37.3	+68.0%	30.6	41.8	+36.6%	374	950	+154.0%

Table 4.12: Results for the MIME model

For the thrust values, the main deviation present is likely due to the used value of the turbine inflow temperature  $T_{t_4}$ . As the used values are maximum values (at least for the Trent and Orpheus), which are usually not attained during cruise flight, this likely causes an overestimation of the cruise thrust. The MIME code determines engine massflow based on post-inlet diameter and an estimated post-inlet Mach number, leading to the ambient Mach number mostly having an effect on the inlet sizing and difference between gross and net thrust values. In this case, too high assumed efficiencies as well as an incorrect assumption of the post-inlet hub-to-tip ratio may also lead to higher predicted thrust levels, the former through an overestimation of the engine efficiency, showing in an underestimated SFC and the latter through an overestimation of the engine massflow, increasing absolute thrust levels without affecting SFC. Also, the difference in ambient static pressure for the  $M_{\text{amb}} = 0$  and  $M_{\text{amb}} = 0.6$  cases is not taken into account, yielding a intake total pressure and therefore a higher pressure at the combustor, yielding more efficient combustion and a potentially higher post-turbine flow energy, increasing both the thrust and SFC estimations.

However, the more detailed model of MIME also allows for several additional variables to be checked, including the shaft speeds, the number of stages in each compressor and turbine and the number of blades on a fan stage. These results are given in Tables 4.13, 4.14<sup>15</sup> and 4.15 along with their reference values if available.

	$N_{\text{stage}}$		$N_{\text{blade}_{\text{rotor}}}$		RPM	
	Ref	MIME	Ref	MIME	Ref	MIME
Fan	1	1	38	17	5490	5497
LPC	3	2			5490	5497
HPC	9	5			15183	27687
HPT	1	1			15183	27687
LPT	4	3			5490	5497

Table 4.13: Additional results from MIME for CFM56-3B1

From these tables, it becomes clear that the number of required stages is generally underestimated, seen most clearly in the CFM56 HPC, the Trent 556 compressors and the Trent 556 LPT. The Trent 556 IPT is however a notable exception. Also, while the RPM of shafts with a fan mounted to them are relatively close to the reference values (usually slightly lower), the other shaft RPM values generally deviate significantly from the reference values.

<sup>15</sup>Rotor speed values as cited by [29] for maximum continuous rotor speeds.

	$N_{stage}$		RPM	
	Ref	MIME	Ref	MIME
Fan	1	1	3479	3363
IPC	8	5	8654	10644
HPC	6	1	12569	34629
HPT	1	1	12569	34629
IPT	1	2	8654	10644
LPT	5	3	3479	3363

Table 4.14: Additional results from MIME for Trent 556

	$N_{stage}$	
	Ref	MIME
Compressor	7	3
Turbine	1	1

Table 4.15: Additional results from MIME for Orpheus Mk.803

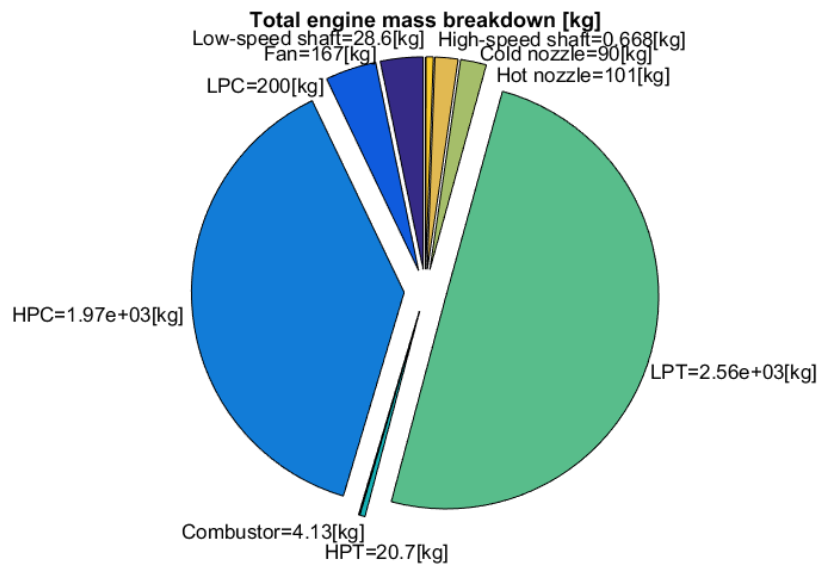


Figure 4.1: CFM56-3B1 mass breakdown from MIME model

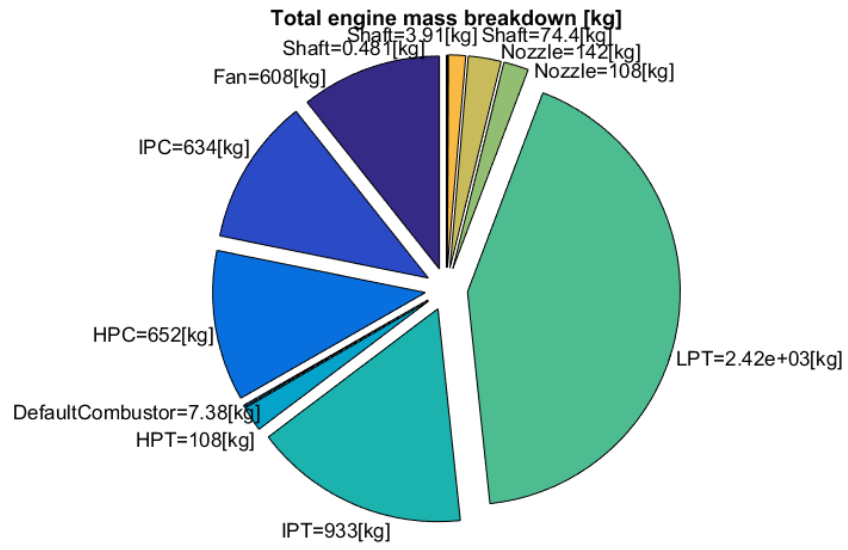


Figure 4.2: Trent 556 mass breakdown from MIME model

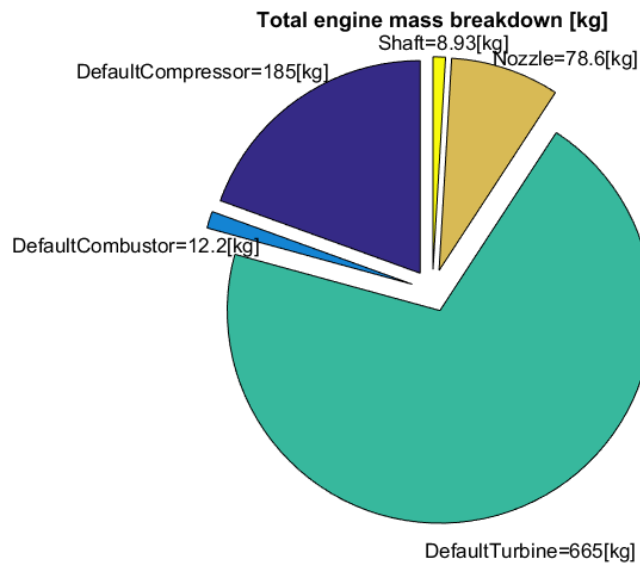


Figure 4.3: Orpheus Mk. 803 mass breakdown from MIME model



From Table 4.12, the excessive mass estimates also stand out. Looking into the mass build-up of the engines, as shown in Figures 4.1, 4.2 and 4.3, it clearly shows that the masses of the final turbines are unexpectedly large, which is mostly due to a huge mass allocation for the turbine discs of these turbines. For example, the mass estimate for the Trent 556 3rd stage LPT rotor disk alone is 1069 [kg]. This can be traced back to excessive blade pull stresses (30.4 [MPa] for that specific disk), caused by too large radii at these turbine stages.

The evaluation times for the MIME model are quite long, with evaluation of the thrust and SFC values taking approximately 58.8s, 66.4s and 12.2s on the author's laptop<sup>16</sup> with other applications running in the background.

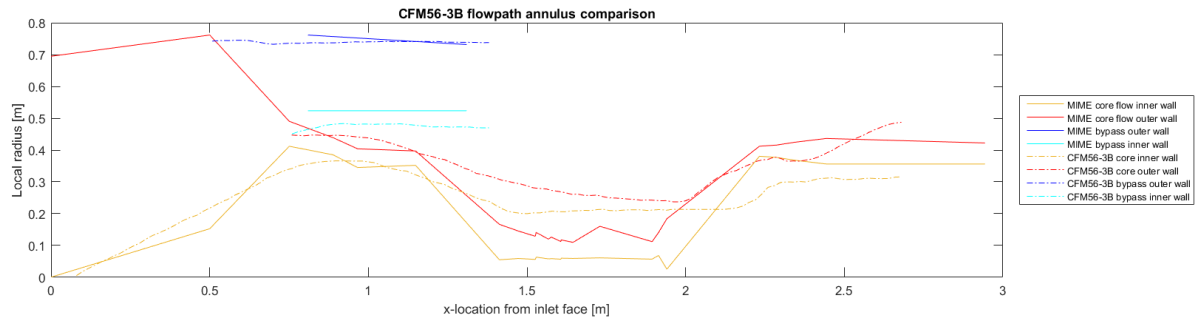


Figure 4.4: Overlay of CFM56-3B1 actual flow path with MIME estimate

Some of these figures include the computed flow paths, formed from the local flow station radii in sequence. These are shown for the CFM56 and Trent 556 in comparison with actual engine section images in Figures 4.5 and 4.6. In Figure 4.4, the actual and MIME-estimated flowpaths for the CFM56 are overlaid, with the fan face locations aligned.<sup>19</sup> Upon comparison, it can be seen that the contractions around the high-speed sections is more severe in the model than found in reality, which is expected when considering the too high spool speeds found. Also, from Figure 4.4, it becomes clear that the length of the high-pressure compressor is underestimated. It is interesting to see that, regardless of the lack of modelling of non-constant-mean-radius combustors and turbines, the LPT outflow mean radius is still corresponding to the actual value.


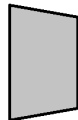







									
	Inlet	Fan	LPC	HPC	Combus-tor	HPT	LPT	Nozzle (hot)	Nozzle (cold)
$l$ [m]	0.500	0.251	0.321	0.500	0.250	0.047	0.503	0.500	0.500
$m$ [kg]	0	166.695	199.651	1967.819	4.131	20.658	2564.266	100.995	90.009
$N_{\text{stage}}$ [-]	N/A	1	2	5	N/A	1	3	N/A	N/A
$P$ [kW]	N/A	5770	2094	6277	983	6296	7830	N/A	N/A
RPM [ $\frac{\text{rev}}{\text{min}}$ ]	N/A	5497	5497	27687	N/A	27687	5497	N/A	N/A
$F_{\text{comp}}$ [kN]	0.269	20.400	6.430	12.200	2.530	4.270	-17.400	-1.890	-3.020
$r_{\text{min}}$ [m]	0.348	0.457	0.451	0.375	0.084	0.084	0.104	0.396	0.396
$r_{\text{mout}}$ [m]	0.457	0.451 <sup>20</sup>	0.375	0.084	0.084	0.104	0.396	0.396	0.389
$\frac{p_{\text{out}}}{p_{\text{in}}}$ [m]	0.990	1.640	2.567	6.473	0.950	0.420	0.239	0.986	0.986

Table 4.16: Per-component results for CFM56

<sup>16</sup>Specifications: HP Elitebook 8570w with Intel i7-3610QM 2.3GHz CPU, 8GB RAM running Windows 7 64-bit

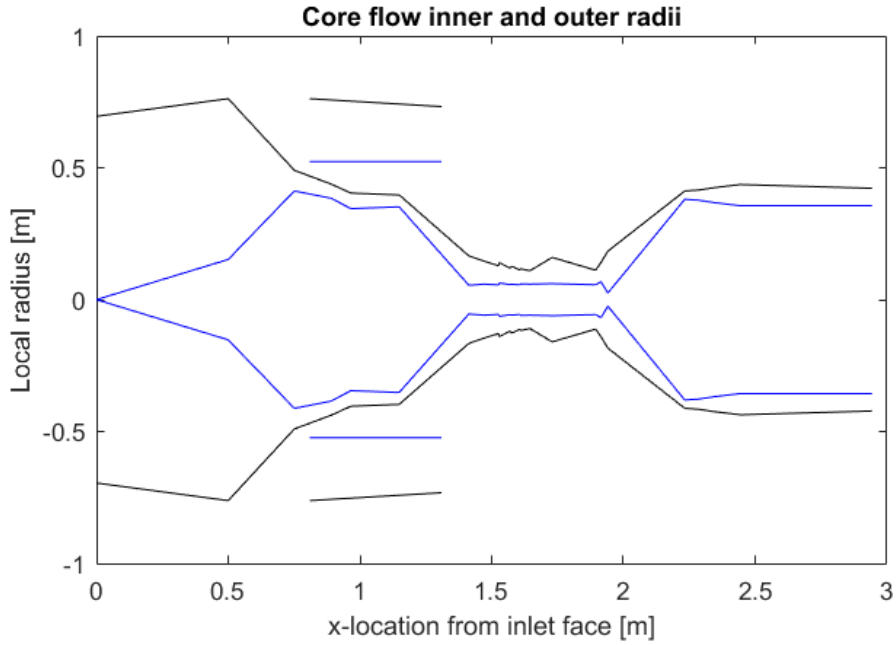
<sup>17</sup>Source: [http://lessonslearned.faa.gov/BritishMidlands092/BMI\\_engine\\_pop\\_up.htm](http://lessonslearned.faa.gov/BritishMidlands092/BMI_engine_pop_up.htm), retrieved on 2018-05-02

<sup>18</sup>Source: <https://www.flickr.com/photos/rolls-royceplc/14334403531/in/album-72157644997543445/>, retrieved on 2018-05-02

<sup>19</sup>While the actual flowpath does include the spinner geometry, it neither includes inlet cowling nor nozzles. Sadly, lacking suitable source data for the Trent 556 and Orpheus, this overlay could only be done for the CFM56.

<sup>20</sup>Fan outflow mean radius is that of the core outflow.





(a) CFM56-3B1 flow path estimate of MIME simulation

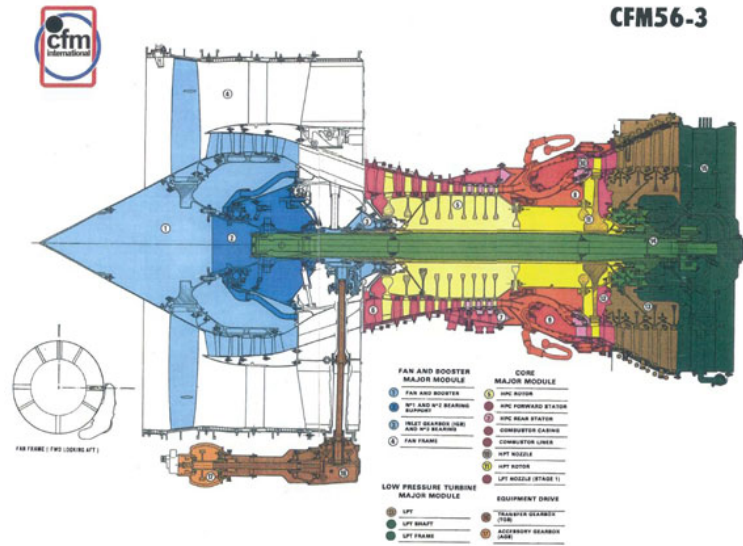
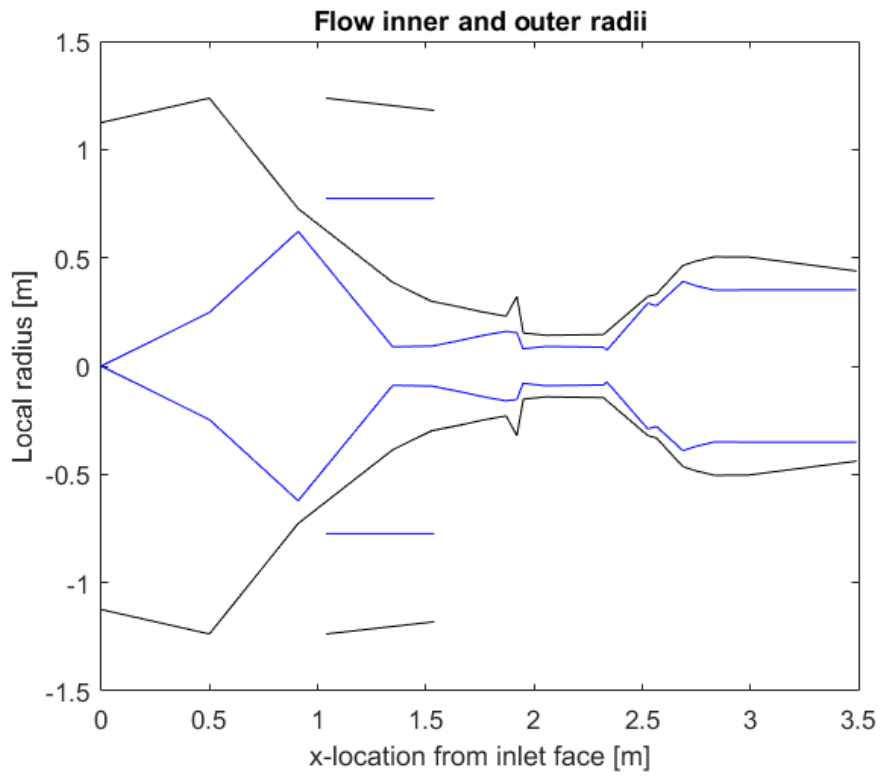
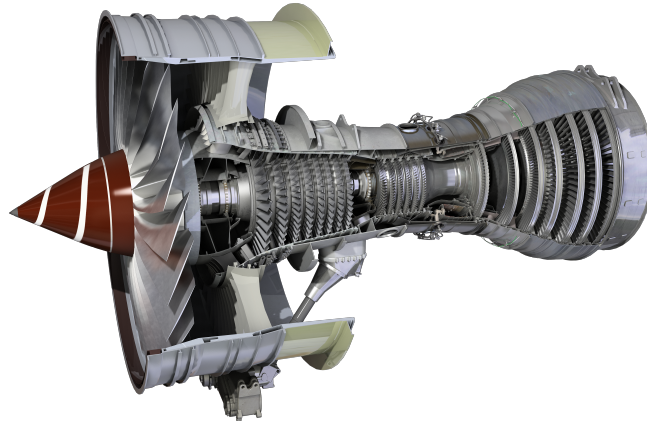
(b) CFM56-3B1 section schematic<sup>17</sup>

Figure 4.5: Comparison between MIME flow path estimate and actual engine section for CFM56-3B1



(a) Trent 556 flow path estimate of MIME simulation



(b) Trent 556 section<sup>18</sup>

Figure 4.6: Comparison between MIME flow path estimate and actual engine section for Trent 556


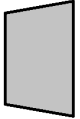

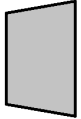


						
	Inflow correction duct	HPT Stage 1	Inflow correction duct	LPT Stage 1	LPT Stage 2	LPT Stage 3
$l$ [m]	0.020	0.027	0.292	0.052	0.059	0.099
$m$ [kg]	0.051	20.607	2.390	455.207	692.608	1414.060
$N_{blade_s}$ [-]	N/A	50	N/A	188	141	91
$N_{blade_r}$ [-]	N/A	38	N/A	164	121	93
$\sigma_{bp_r}$ [MPa]	N/A	1024	N/A	12.000	18.000	36.500
$\sigma_s$ [-]	N/A	1.07	N/A	1.850	1.430	1.330
$\sigma_r$ [-]	N/A	1.09	N/A	1.430	1.430	1.970
$P$ [kW]	N/A	6294.420	N/A	2710.300	2710.300	2409.16
$r_{min}$ [m]	0.084	0.104	0.104	0.396	0.396	0.396
$r_{mout}$ [m]	0.104	0.104	0.396	0.396	0.396	0.396
$F_{comp}$ [kN]	15.065	-10.798	-8.542	-1.824	-3.592	-3.470
$\frac{p_{t_{out}}}{p_{t_{in}}}$ [m]	1.000	0.420	1.000	0.645	0.610	0.607

Table 4.17: HPT and LPT breakdown results for CFM56

To show more visually how much new data is generated on each component, Tables 4.16 and 4.17 provide nice examples of the detail level generated. For example, the erroneous-looking positive component-thrust of the HPT in Table 4.16 can be seen from Table 4.17 to be caused solely by the inflow correction duct, which effectively already functions as a nozzle, increasing axial velocity and as such causing a high component thrust.

# 5

## FULL INITIATOR CASE STUDY

In this chapter, a case study for a full design of an A320-200 using the Initiator is performed, in order to evaluate the impact of the MIME model on the full aircraft design. Even though this is a conventional aircraft with standard engines which should be properly modelled by the current Initiator engine model already, comparing the performance for this aircraft should provide a baseline on the impact of the new MIME model on all aspects of the full aircraft design and calculations.

A short description of the A320-200 and its engines is given in Section 5.1, including the reference data of the aircraft. Next, the outputs and calculation characteristics of the baseline Initiator code for this aircraft are presented in Section 5.2. The same is then done for the Initiator using the MIME code for engine performance estimation instead in Section 5.3. Finally, the outcomes for both runs are compared in Section 5.4.

### 5.1. AIRCRAFT DESCRIPTION

According to [30], the A320-200 is a single-aisle aircraft with an overall length of  $l = 37.57$  [m] and a wingspan of 34.09 [m]. It has an operational empty mass of  $m_{OE} = 42100$  [kg], with a maximum take-off mass  $m_{TO_{max}} = 73500$  [kg] and maximum fuel mass of  $m_{f_{max}} = 24240$  [kg].

### 5.2. BASELINE SIMULATION

Complete design convergence was reached in 1544.43 [s] after 8 iterations, as indicated in Figure 5.1. This yields the aircraft geometry as shown in Figure 5.2.

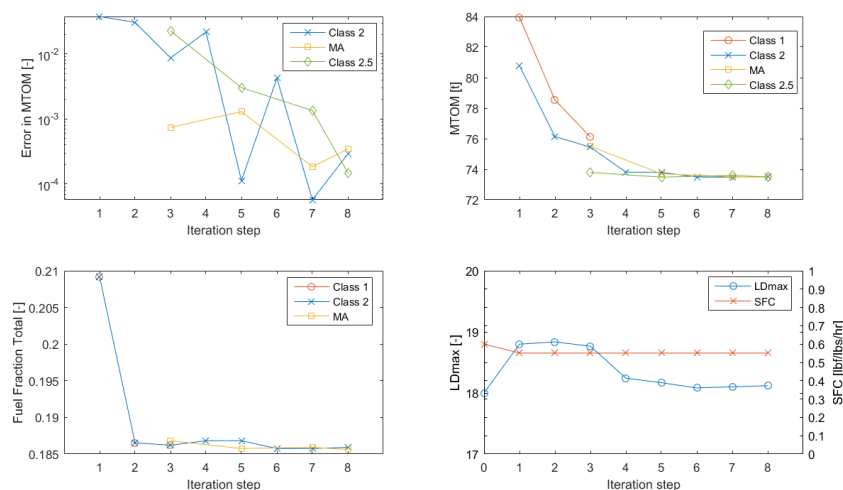


Figure 5.1: Design convergence process of the baseline A320-200 Initiator computation

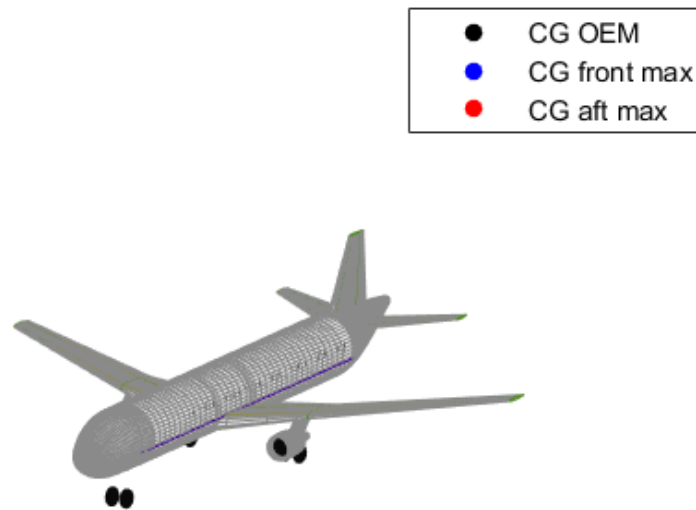


Figure 5.2: Geometry of the baseline A320-200 Initiator computation

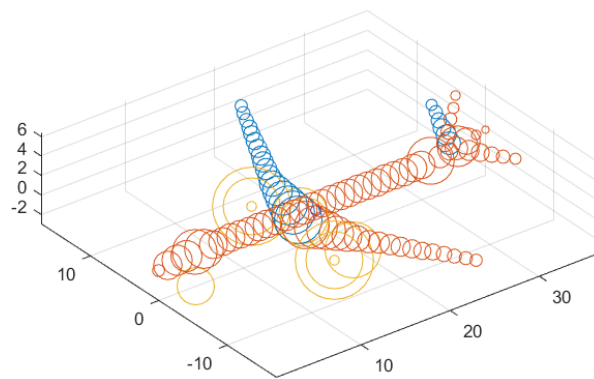


Figure 5.3: Mass distribution of the baseline A320-200 Initiator computation

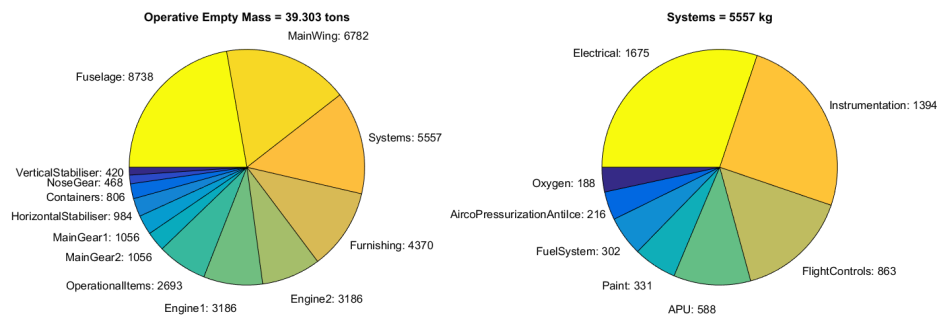


Figure 5.4: Class 2 mass estimation according to Torenbeek of the baseline A320-200 Initiator computation

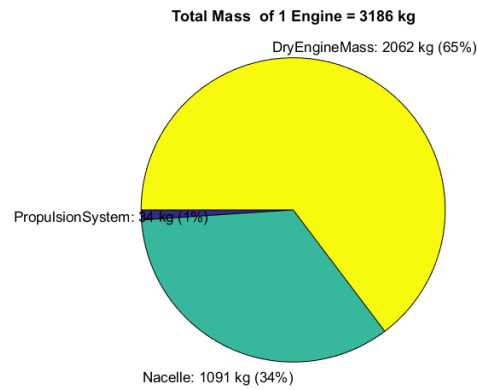


Figure 5.5: Engine total mass composition of the baseline A320-200 Initiator computation

### 5.3. MIME SIMULATIONS

To evaluate the performance of the MIME engine design and mass estimation methods separately, the Initiator A320-200 model is first run with only the MIME engine design method active (mass estimation is still done through the “old” database method), after which a second run is done which uses MIME for both the engine design as well as the mass estimation.

#### 5.3.1. MIME FOR ENGINE DESIGN ONLY

For the first run, using only the MIME design calculations with the database-based mass estimation, the design convergence was achieved in 10154.15 [s] after 10 iterations, as shown in Figure 5.6.

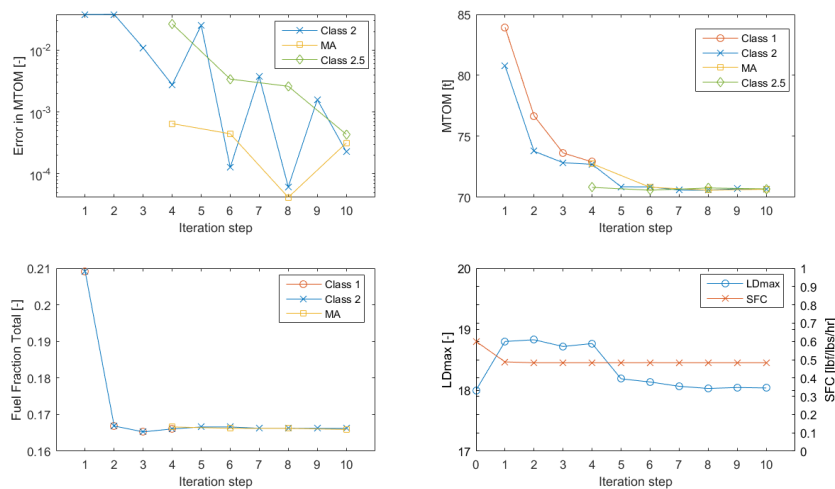


Figure 5.6: Design convergence process of the A320-200 Initiator computation with MIME engine design calculations

From Figure 5.6, it can be seen that the maximum take-off mass estimate is a little lower compared to the actual values, with the fuel fraction also significantly lower. The operative empty mass, of which the build-up can be seen in Figure 5.9, deviates by about 10% from the reference value of  $m_{OE} = 42100$  [kg] with an estimate of  $m_{OE} = 38.400$  [kg].

#### 5.3.2. MIME FOR ENGINE DESIGN AND MASS ESTIMATION

When performing an Initiator run using MIME for both the engine design calculations as well as the engine mass estimation, the design convergence fails during the 9th iteration. The cause for this can clearly be seen in Figure 5.11, as the too-large engine mass estimate requires more lift, increasing drag, increasing the

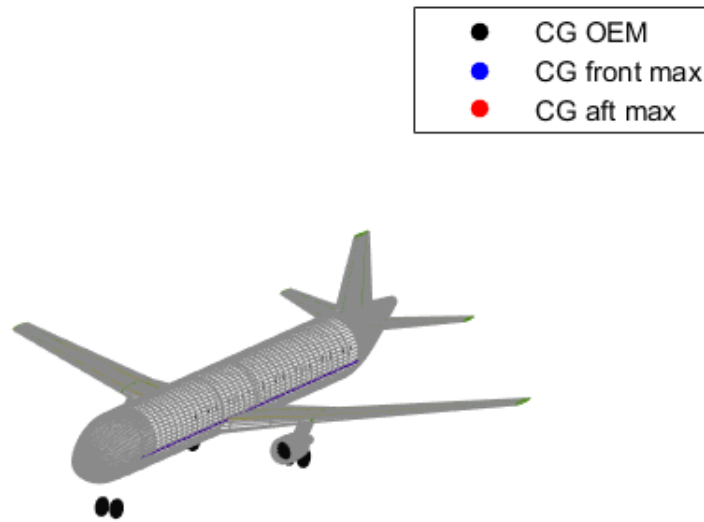


Figure 5.7: Geometry of the A320-200 Initiator computation with MIME engine design calculations

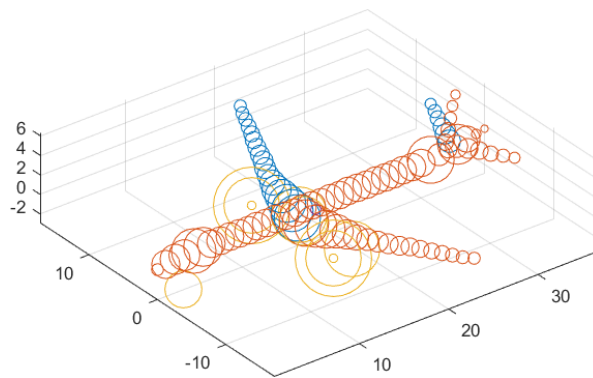


Figure 5.8: Mass distribution of the A320-200 Initiator computation with MIME engine design calculations

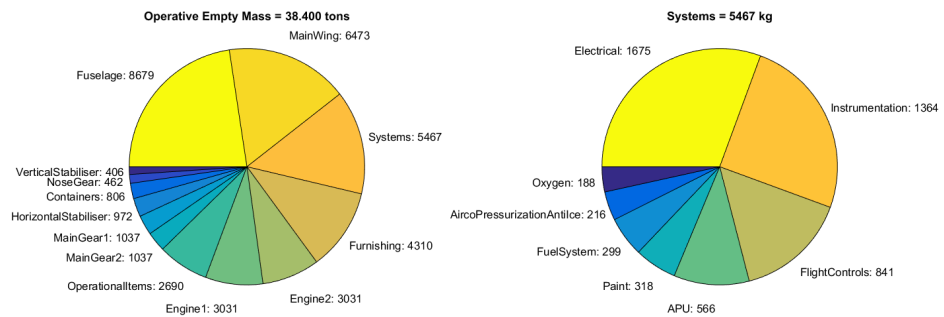


Figure 5.9: Class 2 mass estimation according to Torenbeek of the A320-200 Initiator computation with MIME engine design calculations

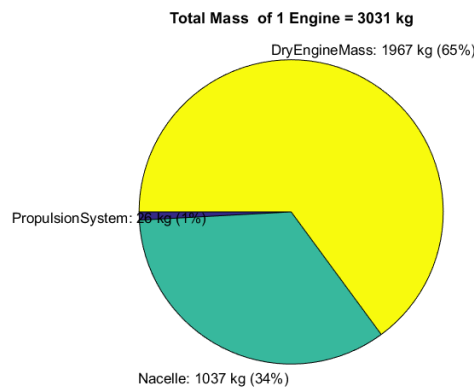


Figure 5.10: Engine total mass composition of the A320-200 Initiator computation with MIME engine design calculations

amount of required thrust and therefore engine size and mass. This creates a snowball effect which is clearly visible in the MTOM<sup>1</sup> convergence, and ultimately causes the convergence to fail on the wing and thrust loading computation.

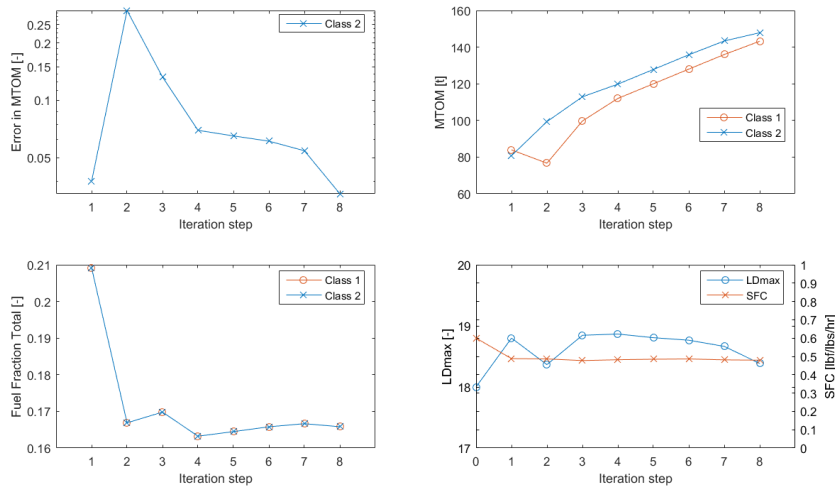


Figure 5.11: Design convergence process of the A320-200 Initiator computation with MIME engine design and mass estimation calculations

Other visible effects include the increase tail size and mass, visible from the mass distribution of Figure 5.12 and the breakdown in Figure 5.13, which also show the absurd values of the estimated engine masses with respect to other aircraft components.

### 5.4. COMPARISON

When comparing the current Initiator engine model with the MIME code results, the first clear remark is that using MIME as intended, so for both engine design as well as mass estimation, currently does not yet work for this case. The deviations in the engine mass estimation (with underlying causes as discussed in Section 4.2.3) are simply too big for the Initiator to deal with. Due to the snowballing effect mentioned earlier, the excessive mass estimations cause large increases in other aircraft parameters, which in turn call for an even larger engine, eventually causing insurmountable issues for other sizing modules (in this case the wing and thrust loading calculation). Therefore, the rest of this comparison will focus purely on the case where the MIME code is used for engine design only.

<sup>1</sup>Maximum Take-Off Mass



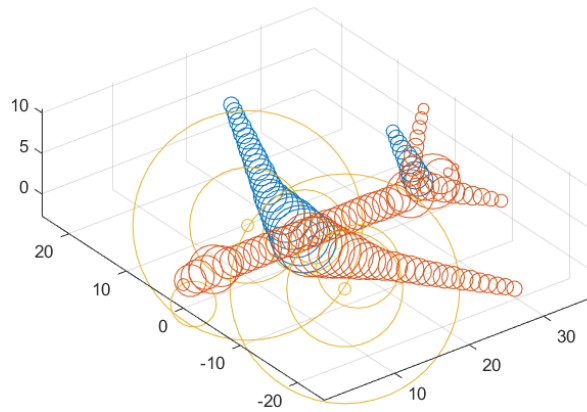


Figure 5.12: Mass distribution of the A320-200 Initiator computation with MIME engine design and mass estimation calculations

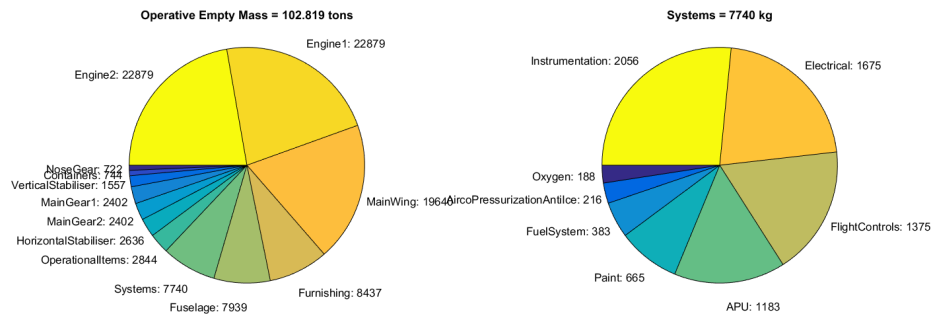


Figure 5.13: Class 2 mass estimation according to Torenbeek of the A320-200 Initiator computation with MIME engine design and mass estimation calculations

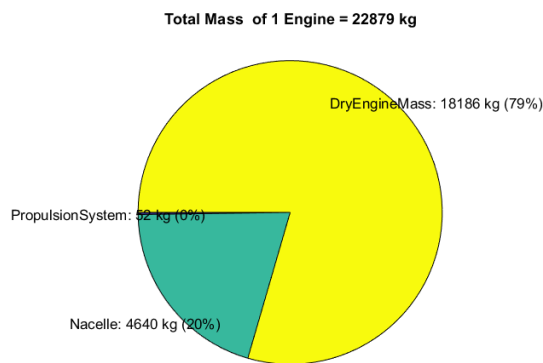


Figure 5.14: Engine total mass composition of the A320-200 Initiator computation with MIME engine design and mass estimation calculations

Another clear issue is the computation time, which has gone up significantly when using the MIME code. This is both due to the model being significantly slower by itself, as already noted in Chapter 4, as well as due to the model being created and re-created within an iterative loop in order to size the fan diameter to yield the required thrust.<sup>2</sup> Also, the Initiator requires more iterations to converge when using the MIME model for unknown reasons.

Comparing the maximum take-off masses, the MIME model run is further off the actual value than the run with the current engine model. Upon closer inspection of Figures 5.4 and 5.9, this is mostly due to the slightly smaller engine masses. These are likely caused by differences in the obtained diameter.<sup>3</sup>

---

<sup>2</sup>A possible solution would be to only modify the required fan diameter and then re-use the original model with this updated value, however for unknown reasons this causes unexpected convergence issues.

<sup>3</sup>While MIME treats the engine as the fan face diameter, the Initiator engine model assumes it as the inlet lip diameter. While the diameter itself does not directly influence the mass calculation, it does work through by changing nacelle mass, drag, and so on, also creating a snowball effect.



# 6

## RESULTS AND CONCLUSIONS

At the end of each project, the outcomes should be listed and evaluated, in order to lead to some conclusions. As this project can be split up into a software creation part, as well as an evaluation of the suitability of this software for its intended purpose, the results for both of these fields are presented separately. The results of the creation of the MIME code are presented and discussed in Section 6.1, while the results of the case studies outcomes are treated in Section 6.2. Finally, conclusions will be drawn regarding these results, also with respect to the original objectives and research questions, in Section 6.3.

### 6.1. SOFTWARE RESULTS

This thesis has resulted in the Modular Initiator Model for Engines MATLAB code, as described in this report. It is a modular, expandable and flexible framework for modelling engines in on-design conditions thermodynamically and mechanically, intended specifically for use within the Aircraft Design Initiator of Delft University of Technology's Flight Performance and Propulsion group. Like the engine model currently implemented in the Initiator, the engine design is based completely on the required on-design performance. The MIME code has been created with modules modelling subsonic intakes, fans, multi-stage axial compressors and turbines, annular combustors, annular subsonic nozzles, concentric shafts and bleed air flows, and is prepared for additional off-take or supply of bleed air or shaft power, as well as the integration of other components in the future, such as radial turbomachinery components, propellers, gearboxes or hybrid-electric motors and/or generators. The current modules may also be updated to provide more accurate and/or detailed modelling, and the entire method may later be updated with off-design performance estimation, which might then also be taken into account in the design loop. Due to time constraints however, these upgrades are left to future researchers.

### 6.2. CASE STUDY RESULTS

The effects of the MIME code have been evaluated through case studies for both the isolated engine code as well as a full aircraft design case study.

#### 6.2.1. ISOLATED CASE STUDIES

As described in Section 4.2.3, the obtained results for thrust and specific fuel consumption are in the right range, however not yet extremely accurate. Mass estimations are mostly overshoot with a sizable margin. When looking into more detail, it can be seen that the found spool speeds of the various shafts which do not have fans mounted to them tends to be too high. Through the used equations and definitions, this translates into higher work per stage, therefore less stages and more efficient turbomachinery, as well as higher stresses on rotor disks, causing excessive engine mass. Taking into account that this too high spool speed is caused by an ineffective calculation for the limit spool speeds for compressor and turbine stages, this single issue is likely to cause most if not all of the observed issues.

### 6.2.2. FULL AIRCRAFT CASE STUDY

From Chapter 5, it can be seen that the full MIME implementation (engine design calculations as well as mass estimation) produces results which, at least for the case studied, prohibit the Initiator from converging. The excessively high mass estimations, which were already mentioned previously, have a too big snowballing impact on the total aircraft design.

Looking purely at the engine design calculations for this specific case, the results are again within a reasonable range. The impact of engine thrust, dimensions and mass is also clearly visible in the sizing of other aircraft components, such as the vertical tailplane.

Regardless of whether or not the mass calculation is used, the computation time for the engine model and/or the number of complete MIME model rebuilds should be decreased significantly, as Initiator evaluation times currently become unacceptably high when using MIME.

If the mass estimation and computation time issues are solved, the MIME model promises to be a reasonable competitor to the currently used model, however with the distinction that the MIME model is fully expandable and extendable and therefore more flexible compared to the current Initiator engine model, as well as more detailed, however at the cost of computational loads.

## 6.3. CONCLUSIONS

The main research question of this thesis was defined as “*How can we size engines in terms of geometry, mass and performance for different engine architectures based on the same Top-Level Engine Requirements, technology level parameters and design strategy?*”. This main question was then subdivided into several subquestions, which will be answered first, in order to generate an answer to the main question. These questions are:

1. *How do the estimations of mass, thrust and specific fuel consumption of the Modular Initiator Model for Engines compare to the actual values for typical engines?*
2. *How do the estimations of additional engine characteristics such as stage or blade numbers, shaft speeds and geometry compare to actual values for typical engines?*
3. *What implications can be observed when using a modular engine model within the full aircraft design process?*
4. *How can this modular engine design method be translated into a software environment?*

Question 4 is answered through the produced MIME code as described in Part II. Questions 1 and 2 are addressed by the isolated case studies, where it is shown that, while the errors currently are significant, they are consistent and do all seem to stem from 1 or 2 specific issues (incorrect RPM limits and  $T_{t_{\max}}$  instead of  $T_{t_{\text{cruise}}}$ ). Using the outcomes of Chapter 5, the answer to Question 3 is found to be that the implications of a proper modular engine modelling tool are likely important. Even though the current MIME code is far from perfect, its imperfections show the impact the various engine parameters and properties have on the rest of the aircraft. From this, it can be concluded that poorly modelled engines will likely lead to a poorly modelled aircraft design, thereby strengthening the case for a good modular engine model, which is capable of accurately modelling a wide range of engines.

From these answers to these subquestions, the main research question can now be answered, yielding that the MIME code is a possible tool for solving this problem as, even though it can certainly benefit from updates and expansions, its framework provides a means for comparing engine designs of varying architecture on the same basis, both in terms of inputs and of calculations.

# 7

## RECOMMENDATIONS

Even though a project has ended, this does not mean that there is nothing left to be desired. For this project, these recommendations for further work can be split up in three main parts. First of all, no piece of software is perfect, both due to the tunnelvision and lack of fresh perspectives all long-time developers will eventually experience, as well as the inevitable lack of specific knowledge which will occasionally occur, or simply a lack of time and/or resources for implementing specific features, which to some possible improvements which might be made later. These are described in Section 7.1. Also, being set up as a modular code, there are many suggestions for other components which could be implemented, extending the capability of MIME. These are presented in Section 7.2.

### 7.1. IMPROVEMENTS ON CURRENT PROGRAM

The improvements to the current code can be subdivided into two categories: those related to the general code architecture and/or overarching calculation procedures, and those related to specific components and/or modules.

#### 7.1.1. GENERAL ARCHITECTURE

Several improvements benefit the MIME code and/or its integration in general.

**Cache for inputs** One of the possible improvements to the program architecture can be sought in storing the results for a given calculation, and building a response surface to generate improved estimates for both the individual component and top-level starting points. This may decrease the required number of iterations on both component- and top-level, increasing the code speed. However, due to the large number of inputs, this may also result in the required interpolations taking more time than they would save.

**Re-using previous results** At the moment, a new MIME model is created for every set of inputs the Initiator provides. However, as this is often only a small change (for the current Initiator implementation only changes in fan face diameter), it may be possible to update solely this parameter on an already converged engine model, which should allow for faster convergence upon recalculation compared to starting from scratch again.

**Initiator inputs** The MIME model provides much more flexibility than is currently used with the current inputs to the Initiator. Therefore, the Initiator settings files and MIME implementation may be adapted to take advantage of these extended capabilities, taking care to still maintain backward compatibility.

**Material properties** Several material properties which are used are actually temperature-dependent in reality. As the local temperatures are known, this may be incorporated into the calculations.

#### 7.1.2. COMPONENT-SPECIFIC IMPROVEMENTS

While all modules can be improved in many ways, the list below provides some of the most critical improvements.

**Inlet** The inlet mass estimation is currently completely left out of the code. This is due to a lack of sufficiently reliable design parameters and constraints for this part. However, when off-design performance is also taken into account, it may be possible to say more about the different required operating conditions of the inlet, and therefore also about the required geometry, in order to be able to generate at least a reasonably accurate estimate of the inlet mass and dimensions.

**Spool speeds** The compressor and turbine stage estimations for the limit spool speeds can be improved, as currently an undesired feedback seems to be present, boosting spool speeds and dragging down radii.

**Blade pull stress** Currently, excessive blade pull stresses cause extremely heavy rotor discs, which have a detrimental effect on the engine mass estimation. It may be possible to include a coupling between blade pull stress and/or disk thickness and the mean radius and/or limit RPM values.

**Combustor** The combustor geometry and mass estimation is currently extremely coarse, and certainly requires improvements. This also goes for the estimation of the outflow (static) conditions.

## 7.2. CAPABILITY EXTENSIONS

Besides improvements to the already existing functions of the code, it is also interesting to consider upgrades to the code which extend the capabilities, either by adding new components to widen the range of modellable engines, or by adding the calculation of other engine properties, such as off-design performance.

### 7.2.1. OFF-DESIGN PERFORMANCE ESTIMATION

Adding off-design performance estimation methods allows for all the properties of the engine used within the Initiator to be taken from the same model. Also, this may aid in updating the sizing, as actual engine sizing is dependent on off-design as well as on-design considerations, with only the on-design part being considered in the current MIME code.

### 7.2.2. ADDITION OF COMPONENTS

It may also be beneficial to add modules for additional components to the MIME code. For example, these may include:

**Propeller** To allow for the modelling of a turboprop engine, a propeller module is required.

**Gearbox** In order to model geared turbofans, turboprops or other geared engines, a gearbox module is needed. This should likely act as a consumer on one Shaft, and a supplier on another, however whether the gear ratio should be specified to determine the shaft speeds, or the other way around, needs to be looked at.

**Electric motor** For modelling of hybrid engines, an electric motor module may be of interest for the case where electric power is used to boost an engine during (certain phases of) flight (or even to model purely electric propulsors).

**Generator** For modelling the effects of electric power off-take from the engine, regardless of whether this power is used for aircraft systems or for powering other propulsors, a generator module may be created.

**Radial compressor/turbine** For modelling smaller and more compact motors, modules modelling radial compressors and turbines may be developed.

# II

## CODE REPORT





# 8

## INTRODUCTION

While Part I has focussed on the theoretical basis behind the MIME code, this part focusses on the actual code, the exact workflows and the considerations behind pure coding decisions. This includes workflow diagrams of various functions, class diagrams of the involved classes, and a coding guide.

In Chapter 9, the requirements for the code and the external interfaces are discussed. Next, Chapter 10 gives a detailed description of all the classes included in the MIME code. Chapter 11 then covers the integration of the MIME code within the Initiator and finally, Chapter 12 provides a short guide for expanding MIME with e.g. new components, such as radial turbines, electric generators, motors or bleed air consumers.



# 9

## REQUIREMENTS, ARCHITECTURE AND INTERFACES

The MIME code was written specifically for integration within the Initiator. This has a large influence on the desired, available and possible input and output, which is described in Section 9.1. The Initiator integration, along with some other considerations, also impose important requirements on the MIME code, which are described in Section 9.2. Finally, the resulting code architecture is described in 9.3.

### 9.1. INTERFACES

The MIME code was originally conceived as a drop-in replacement for the `Turbofan_Initiator` function, with initially the same assumed in- and outputs. This `Turbofan_Initiator` function yields the engine thrust  $F$  and fuel massflow  $\dot{m}_f$  as a function of engine (fan face) diameter  $D$ , cruise altitude  $h_{\text{cruise}}$  and Mach number  $M_{\text{cruise}}$ , fan core and bypass pressure ratios  $PR_{\text{fancore}}$  and  $PR_{\text{fanbp}}$ , low- and high-pressure compressor pressure ratios  $PR_{\text{LPC}}$  and  $PR_{\text{HPC}}$ , turbine inlet total temperature  $T_{t4}$ , combustion chamber pressure ratio  $PR_{\text{CC}}$ , bypass ratio BPR and the number of high- and low-pressure turbine stages  $N_{\text{stageHPT}}$  and  $N_{\text{stageLPT}}$ , along with a set of hardcoded efficiencies. Of these values, the diameter  $D$  is used by the Initiator as control variable, while the rest is obtained from the aircraft `Settings` file, except for the BPR and the engine type, which are sourced from the aircraft `Input` file.

These values can be split up into several different groups. The cruise conditions ( $h_{\text{cruise}}$ ,  $M_{\text{cruise}}$ ) will always be used, as they determine the operating conditions of the engine, which are required regardless of the engine type or lay-out. The pressure ratios, efficiencies, bypass ratio and turbine inlet temperature are dependent on the engine type and lay-out, as they characterise the various individual components. Technically, this also goes for the diameter (a defining property of the inlet), however as it is used as the control variable, it is considered separately. The specified numbers of turbine stages have become completely obsolete, as these are computed from the required power consumption and turbomachinery performance limits (see Sections 3.2 and 10.3).

The MIME code is also intended to (partially) replace the `getEngineWeight` function, specifically the database call which estimates dry engine mass from the engine cruise thrust. For this, the interface which will be kept is the output of the dry engine mass, while the computation will now be based on the earlier sized engine.

### 9.2. REQUIREMENTS

As the code is intended to be integrated in the Initiator, this sets several requirements on it. Primarily, it must be relatively fast. so its level of detail must be relatively low, as the Initiator is intended to be a relatively fast-acting code for designing complete aircraft, of which the engines are only one part. Also, as the Initiator is a.o. used for studying the impact of design choices on complete aircraft performance, the code must be flexible, and ideally be able to design different engines using similar design methodologies, as to ensure any found differences are due to the design choice, instead of due to discrepancies between the used different methods.

In order to allow for relatively easy further development and future-proofing, it is also essential that it is easy and straightforward to improve and expand. The current developments towards more (or even com-

pletely) electric aircraft are a good example for this: any engine model which aims to be even slightly future-proof must incorporate electric components or at least be prepared to, imposing a large degree of required flexibility on the code architecture.

### 9.3. ARCHITECTURE

The flexibility and expandability requirements have lead to a modular code, based on a single engine-modelling object (the `AeroEngine` class), containing many component-modelling objects. Each of these component models have independent performance calculation and design methods, based purely on their interfaces with other components and initially specified requirements, allowing for consistent calculation methods for components of equal type.

The performance and design calculations are controlled by methods of the top-level object, which calls the appropriate methods of the individual component objects. Information is conveyed between the components through flow-condition-describing flow `Station` objects, contained in `Flow` objects which model the gas flowpaths through the engine, and through `Shaftstation` objects in the `Shaft` objects, which convey information on required or generated power, shaft RPM and possible limitations.

The modelled components have masses, dimensions and positions, allowing for determination of the complete engine total mass and dimensions.<sup>1</sup>

---

<sup>1</sup>Theoretically, the fact that component masses and locations are known should also allow for an estimation of the center of gravity of the engine. This has however not yet been implemented.

# 10

## CLASS DESCRIPTIONS

This chapter describes each class generated by the MIME program, explaining its purpose, properties and methods. This also includes class diagrams and workflow diagrams for the various functions. While most of the theoretical basis behind the different calculation methods was already presented in Chapters 2 and 3, the actual implementations are described in this chapter.

### 10.1. THE AEROENGINE CLASS

AeroEngine class diagram

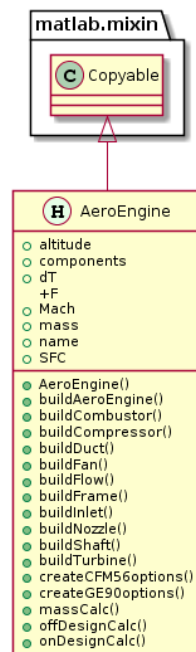


Figure 10.1: Class diagram for the AeroEngine class.

The goal of the AeroEngine class is to model an air-breathing engine using a component-based model, yielding information on engine mass and on-design performance.

An AeroEngine-class object (as shown in Figure 10.1) is used to describe the engine, with all the objects describing the engine components stored within the components property array. In this array, the components are listed in their required order of calculation. Each of these components is again represented by an object, of which there are various types. These include “actual” aerodynamic components, such as Inlet-, Comburst- and Nozzle-class objects, but may also include “gathered” aerodynamic components such as

an `AxialCompressor`- or `AxialTurbine`-class object, which contain multiple sub-components (for example `AxCompressorStage`-class objects). Some array members also represent non-aerodynamic components, such as `Frame`- or `Shaft`-class objects. Special members within this array are the `Flow`- and `Shaft`-class objects. While the `Shaft`-class also still represents an actual engine component, the `Flow` does not. Both however contain arrays of `Station`- or `Shaftstation`-class objects. The former class contains information on flow-states at a certain location, such as massflow rate, gas composition, velocities, flow area and total and static properties, while the latter class defines the position and power supply or demand of a component. Most of these classes are subclasses of the `Component` class or its sub-class `AeroComponent`.

### 10.1.1. PROPERTIES

As can be seen in Figure 10.1, the `AeroEngine` class has several properties. Besides the standard `name`, `mass` (in [kg]) and `debug` properties, these include the `components` array, which contains all the objects which model the engine components in the proper order. The properties `Mach`, `altitude` and `dT` represent the design conditions, being the design point Mach number  $M$ , altitude  $h$  (in [m]) and the temperature deviation from the ISA conditions  $\Delta T$  (in [K]). Finally, the class contains the properties `F` and `SFC`, which are used as containers for structures containing respectively the thrust and gross thrust (in [kN]) or SFC (in [ $\frac{g}{N \cdot s}$ ]) and fuel massflow (in [ $\frac{kg}{s}$ ]), along with the conditions  $(M, h, \Delta T)$  for which they were found.<sup>1</sup>

### 10.1.2. METHODS

The `AeroEngine` class has a number of methods associated with it. Besides its constructor `AeroEngine()`, these include the `buildAeroEngine()` method which handles the conversion of options into different components, the `buildCombustor()`, `buildCompressor()`, `buildDuct()`, `buildFan()`, `buildFlow()`, `buildFrame()`, `buildInlet()`, `buildNozzle()`, `buildShaft()` and `buildTurbine()` methods, which effectively wrap the constructor methods of the relevant classes to ensure they get the requirement information. The static methods `createCFM56options()` and `createGE90options()` are used to create options structures for pre-defined engines. Finally, the `massCalc()`, `onDesignCalc()` and `offDesignCalc()` methods are used to obtain the mass, engine design and on- and off-design performance of the engine.

#### THE AEROENGINE() AND BUILDAEROENGINE() METHODS

As already mentioned in Section 1.2, the Modular Initiator Model for Engines requires more information about the engine from the user than was the case in the current Initiator engine estimations, and also stores the generated engine design, which did not happen in the old system. In order to allow for this, an `AeroEngine`-class object is generated, with all its components, to conform to the specifications given by the Initiator user inputs. The workflow of the `AeroEngine` object constructor is shown in the flowchart of Figure 10.2, while the workflow of the `buildAeroEngine` function, which converts the options structure to a set of components, is shown in the flowchart of Figure 10.3.

Figure 10.2 clearly shows that the `AeroEngine` constructor function is purely intended to ensure that the basic information is present, before calling for the construction of all the components and putting them in a structured order. The constructor requires two inputs: a name and an options structure. After setting the name of the new engine to the specified value (or a default name, if not specified), it checks whether the provided options structure is not empty. If it is empty, the constructor replaces it by the options structure of a default engine. If it is non-empty, it checks whether it contains more than just an engine name. If only an engine type name is provided, the code checks whether it has a preset options structure for this engine type, and loads it if this is the case. If this is not the case, or if more than one options field is given, the code retains the provided structure. Next, it checks whether design condition (cruise) altitude, Mach number and ISA temperature deviation are specified, and defaults them to  $h_{\text{cruise}} = 10000$  [m],  $M_{\text{cruise}} = 0.85$  and  $\Delta T = 0^\circ$  if not given. It then calls the `buildAeroEngine` function to actually convert the data from the options into engine components. After the components have all been constructed, it calls the `order_engine_array` function to ensure all components are ordered according to their required order of calculation.

In contrast, as can be seen from Figure 10.3, the `buildAeroEngine` method is intended to actually convert the options into the components array. It assumes that any `AeroEngine` at least requires at least an `Inlet`, `Combustor`, `Nozzle`, `Flow` and `Shaft` object. This is due to the fact that the most simple jet engine (an intake, combustion chamber and nozzle) requires at least these components, except for the `Shaft`. The reason the

<sup>1</sup>This is done in preparation for incorporation of off-design performance calculation. For purely on-design calculations, these properties will thus have only one entry.

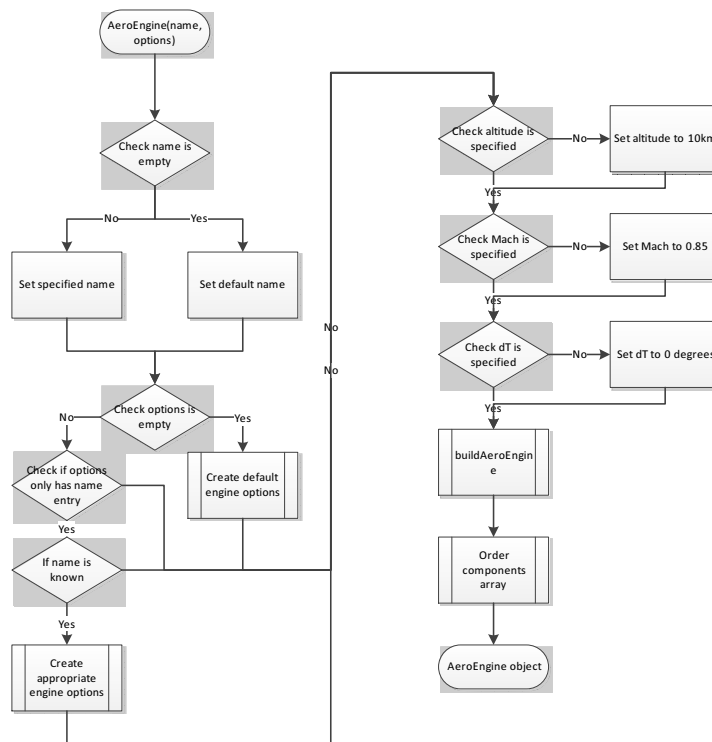


Figure 10.2: AeroEngine constructor workflow flowchart

Shaft is also included is due to the fact that it may cause issues if this component is accidentally left out of any other, more complicated, engine configuration. The `buildAeroEngine` method then checks for the presence of fields for these and all other supported components within the `options` structure, each of which may also be an array. If a component is considered essential, but does not have an associated field in the `options` structure, the code first creates an appropriate default field. Then, if a field is present for a given component, the corresponding `build*` method of the `AeroEngine` object is called with each of the field array entries. If the fields for all supported components have been processed this way, the method ends.

#### THE `BUILDINLET()` METHOD

In order to illustrate the functioning of the `build*` methods called during the `buildAeroEngine` method, the `buildInlet` workflow, as depicted in Figure 10.4, is discussed here. It is important to emphasize that these `build*` methods are not the component constructor methods, but wrap around them, in order to provide a centralised location for handling deficiencies in component specifications.

As shown in Figure 10.4, the `buildInlet` method starts by checking whether all the options required by the `Inlet`-class constructor are present in the provided part of the `options` structure. If any of these are missing, they are substituted by default reasonable values. Next, the `Inlet`-class constructor is called to create the inlet component, after which its `setHT_in` and `setHT_out` methods are called to set the in- and outflow hub-to-tip-ratios to the specified values. Finally, the newly created object is added to the `AeroEngine` components array, after which the method ends.

#### THE `CREATEGE90OPTIONS` AND `CREATECFM56OPTIONS` METHODS

If no `options` structure is provided, or only the name is specified as “GE90” or “CFM56”, pre-defined `options` structures are used, which are created by the `createGE90options` and `createCFM56options` methods. The contents of these created structures are given in Table 10.1, and are mostly sourced from [15, 31–33].

#### THE `ORDER_ENGINE_ARRAY()` FUNCTION

The `order_component_array` function, as shown in Figure 10.5, is used to ensure the components array is ordered in the correct way for the other calculations. While not a direct method of the `AeroEngine` class, the function is only used in this context and is therefore discussed here. The desired correct order starts with all aerodynamic component objects (e.g. `Inlet`, `AxialCompressor`, `Combustor`), followed by the `Shaft` and



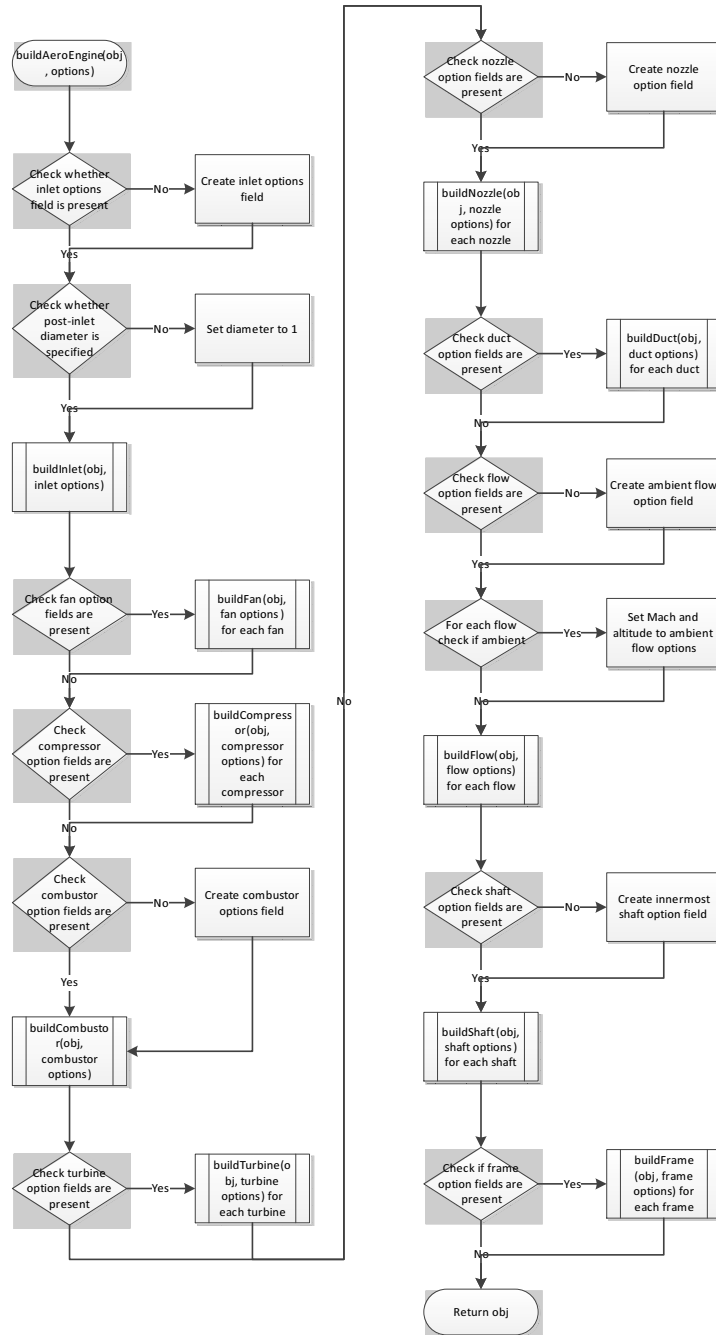


Figure 10.3: AeroEngine buildAeroEngine component creation workflow flowchart

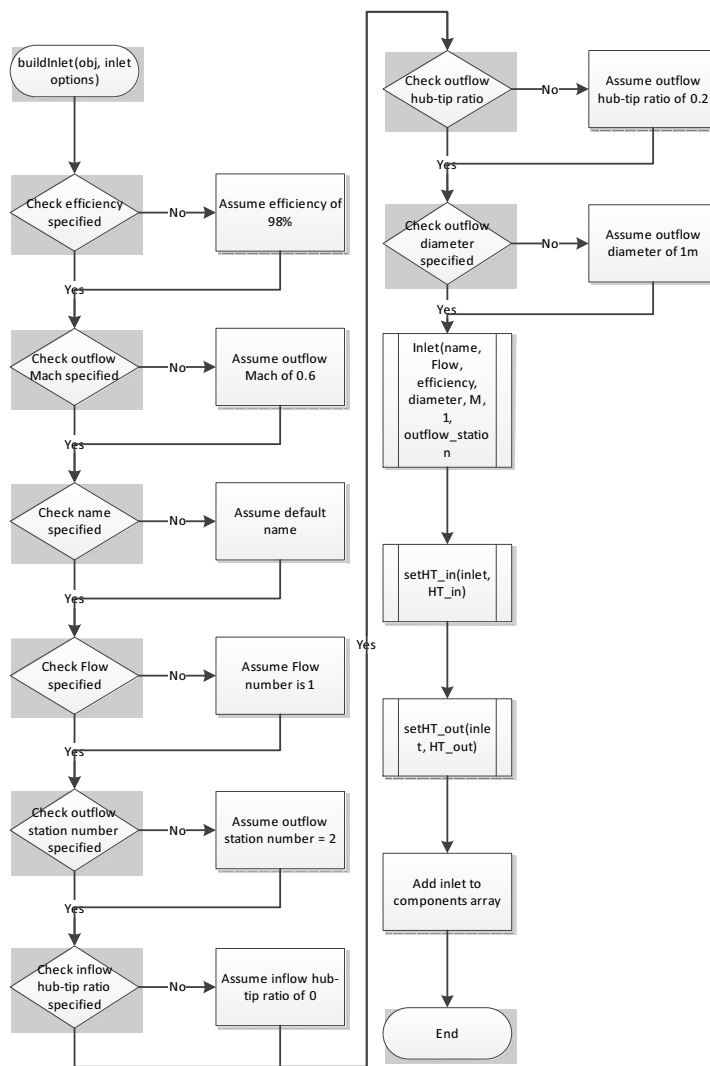


Figure 10.4: AeroEngine buildInlet inlet creation workflow flowchart

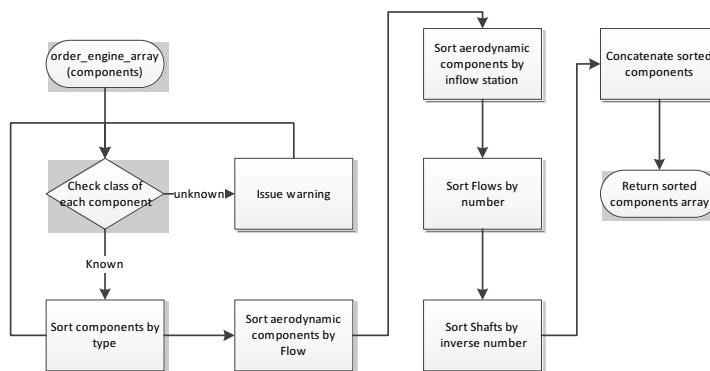


Figure 10.5: order\_engine\_array components re-ordering function workflow flowchart

Table 10.1: Contents of detail GE90 and CFM56 options structures

Field	GE90	CFM56	Field	GE90	CFM56
options.name	GE90	CFM56	options.duct(1).name	Core duct	Core duct
options.Mach	0.85	0.8	options.duct(1).flow	1	1
options.altitude	10668	10668	options.duct(1).PR	1	1
options.inlet.name	Intake	Intake	options.duct(1).stat_in	5	5
options.inlet.diameter	3.124	1.520	options.duct(1).stat_out	7	7
options.inlet.eta	0.98		options.nozzle(1).name	Core nozzle	Core nozzle
options.inlet.flow	1	1	options.nozzle(1).eta	0.95	0.95
options.inlet.stat_out	2	2	options.nozzle(1).flow	1	1
options.inlet.HT_in	0	0	options.nozzle(1).stat_in	7	7
options.inlet.HT_out	0.29	0.29	options.nozzle(1).stat_out	8	8
options.fan.name	Fan	Fan	options.duct(2).name	Bypass duct	Bypass duct
options.fan.flow	1	1	options.duct(2).flow	2	2
options.fan.flow_bp	2	2	options.duct(2).PR	1	1
options.fan.shaft	2	2	options.duct(2).stat_in	13	13
options.fan.stat_in	2	2	options.duct(2).stat_out	17	17
options.fan.stat_out	21	21	options.nozzle(2).name	Bypass nozzle	Bypass nozzle
options.fan.stat_out_bp	13	13	options.nozzle(2).eta	0.95	0.95
options.fan.PRC	1.65	1.685	options.nozzle(2).flow	2	2
options.fan.PRD	1.65	1.685	options.nozzle(2).stat_in	17	17
options.fan.BPR	8.1	5	options.nozzle(2).stat_out	18	18
options.fan.eta	0.93	0.9	options.flow(1).name	Core flow	Core flow
options.fan.material	titanium	titanium	options.flow(1).flow	1	1
options.fan.dimrel	tip	tip	options.flow(1).Mach	options.Mach	options.Mach
options.fan.dimrel_fac	1	1	options.flow(1).altitude	options.altitude	options.altitude
options.compressor(1).name	LPC	LPC	options.flow(1).hasAmbient	true	true
options.compressor(1).flow	1	1	options.flow(2).name	Bypass flow	Bypass flow
options.compressor(1).stat_in	21	21	options.flow(2).flow	2	2
options.compressor(1).stat_out	25	25	options.flow(3).name	Bleed air flow	Bleed air flow
options.compressor(1).shaft	2	2	options.flow(3).flow	99	99
options.compressor(1).eta	0.91	0.931	options.shaft(1).name	High-speed shaft	High-speed shaft
options.compressor(1).PR	1.14	1.935	options.shaft(1).shaft	1	1
options.compressor(1).material	steel	steel	options.shaft(1).eta	0.99	0.99
options.compressor(2).name	HPC	HPC	options.shaft(1).spacing	0.00508	0.00508
options.compressor(2).flow	1	1	options.shaft(1).innermost	false	false
options.compressor(2).stat_in	25	25	options.shaft(2).name	Low-speed shaft	Low-speed shaft
options.compressor(2).stat_out	3	3	options.shaft(2).shaft	2	2
options.compressor(2).shaft	1	1	options.shaft(2).eta	0.99	0.99
options.compressor(2).eta	0.91	0.903	options.shaft(2).spacing	0.00508	0.00508
options.compressor(2).PR	21.5	9.369	options.shaft(2).innermost	true	true
options.compressor(2).material	steel	steel	options.shaft(2).radius_ratio	0.75	0.75
options.compressor(2).alpha_out	0	0	options.frame(1).name	Fan frame	Fan frame
options.combustor.name	Combustor	Combustor	options.frame(1).type	SBF	SBF
options.combustor.flow	1	1	options.frame(1).flow	1	1
options.combustor.stat_in	3	3	options.frame(1).station	21	21
options.combustor.stat_out	4	4	options.frame(2).name	Inter-compressor frame	Turbine outflow frame
options.combustor.tt_out	1380	1578	options.frame(2).type	IF	TF
options.combustor.eta	0.99	0.9827	options.frame(2).flow	1	1
options.combustor.PR	0.95	0.94	options.frame(2).station	25	5
options.combustor.fueltype	1	1	options.frame(3).name	Burner frame	
options.combustor.t_res	0.001	0.001	options.frame(3).type	IF	
options.turbine(1).name	HPT	HPT	options.frame(3).flow	1	
options.turbine(1).flow	1	1	options.frame(3).station	3	
options.turbine(1).stat_in	4	4	options.frame(4).name	Turbine outflow frame	
options.turbine(1).stat_out	45	45	options.frame(4).type	TF	
options.turbine(1).shaft	1	1	options.frame(4).flow	1	
options.turbine(1).eta	0.93	0.903	options.frame(4).station	5	
options.turbine(1).V_ax_out	100				
options.turbine(2).name	LPT	LPT			
options.turbine(2).flow	1	1			
options.turbine(2).stat_in	45	45			
options.turbine(2).stat_out	5	5			
options.turbine(2).shaft	2	2			
options.turbine(2).eta	0.93	0.8851			
options.turbine(2).V_ax_out	150				
options.turbine(2).alpha_out	0	0			

Flow objects, and finally the Frame objects.<sup>2</sup> The aerodynamic components are ordered by their associated Flow number (ascending), and, within that, by their inflow Station identifier [34]<sup>3</sup>, again in ascending order and therefore physically speaking in the order in which they are encountered by a flow particle. This ensures that the thermodynamic calculations for the various objects do not have errors between the outflow conditions of the preceding component and the inflow conditions of the following component. The Shaft objects are ordered from the inside out, as the shafts are dimensioned based on the outer diameter of the shaft directly inside of them<sup>4</sup>, and there are no other interlinks between the different Shaft objects which influence the required calculation order. The Flow objects are ordered according to their ascending numbers (according to convention this yields an order from the core flow going outward), with the bleed air Flow object being the last. The Frame objects are again sorted based on their associated station identifier, as no information on  $x$ -location is known yet upon engine initialisation.

The `order_engine_array` function first separates all component array entries into separate arrays according to these groups, discarding any which do not fit any of these descriptions. It then sorts them according to the rules laid out previously. Finally, it concatenates all the individual sorted arrays, again according to the previously stated order, to form a newly sorted components array, which the function then finally returns.

With the entire AeroEngine object created and its components array filled and sorted, the object is now ready for the actual calculations to be performed on it.

#### THE `onDesignCalc()` METHOD

The on-design calculation is performed by calling the `onDesignCalc()` method of the AeroEngine object. During this on-design calculation, the engine components are sized for the on-design condition, and the on-design performance (SFC, net and gross thrusts  $F$  and  $F_g$ ) is estimated. This is done by iterating over the components, with each of them calculating their respective component properties and post-component flow states, as illustrated in Figure 10.6. Essentially, the `onDesignCalc()` is a structured way of calling all the individual component `aeroCalc()` methods with the required additional information, for as long as needed to obtain a converged design, doing one final iteration when convergence is achieved.

The `onDesignCalc()` method starts by initialising a counter, after which it iterates over the entries of the components array until the counter exceeds the array length. For virtually all components (except the bleed Flow), a similar procedure is followed. First, the class of the component is determined, after which the required other components are found. These often include Flow or Shaft classes, as information is required on the incoming flow state and velocity, contained in the appropriate Station of the relevant Flow, and on the RPM of the Shaft. Both are also often required to be able to write away the outflow conditions to the proper Station, as well as the power production or off-take and RPM upper limit to the proper Shaftstation. Additionally, the bleed Flow is required for compressors and turbines, as this determines the amount of bleed air off-take for the compressors, and bleed air supply conditions for the turbines. In some cases, some other components are also required, such as the preceding component, or the bypass Flow. With all of the required components found, these are passed onto the component `aeroCalc()` method<sup>5</sup>, along with the original component object being evaluated. For most components, the code then moves on to the next one.

The Flow and Shaft objects are however the exceptions to this. In case of the Shaft, it depends on whether the shaft is the innermost shaft or not. In the latter case, first the outer diameter of the next shaft inward is used with the required spacing to determine the required inner diameter. Afterwards, the calculations for the Shaft are done in an analogous fashion to the other components. Finally, a check is performed on whether the original and recalculated spool speeds are sufficiently close together, setting a greenlight variable accordingly.

In the case of the Flow object, no other components are required, and only if the Flow represents the bleed air flow, any action is performed. For this Flow, the `calcBleed()` method is run first, which not only calculates a new distribution of bleed air supply and demand, but also returns a value representing the error in the original bleed air supply massflow estimation. If this error does not remain below 0.5%, a greenlight boolean is set to false, and the code resets the counter variable to 1, causing a re-iteration of all calculations from the top of the components array with the new bleed air distribution. If the error does remain below 0.5%,

<sup>2</sup>This is due to the way the `onDesignCalc()` method is set up. See Section 2.2 for more details.

<sup>3</sup>The Station identifier, even though it is fully numeric, is treated as a string rather than a number, due to the fact that for example station "25" is used to indicate a station between stations "2" and "3", rather than after both.

<sup>4</sup>The inner shaft cannot be sized using this method, and is therefore sized using a fixed hub-tip-ratio. See Section 3.8 for further details.

<sup>5</sup>For more information on these methods, see their specification in Chapter 3.

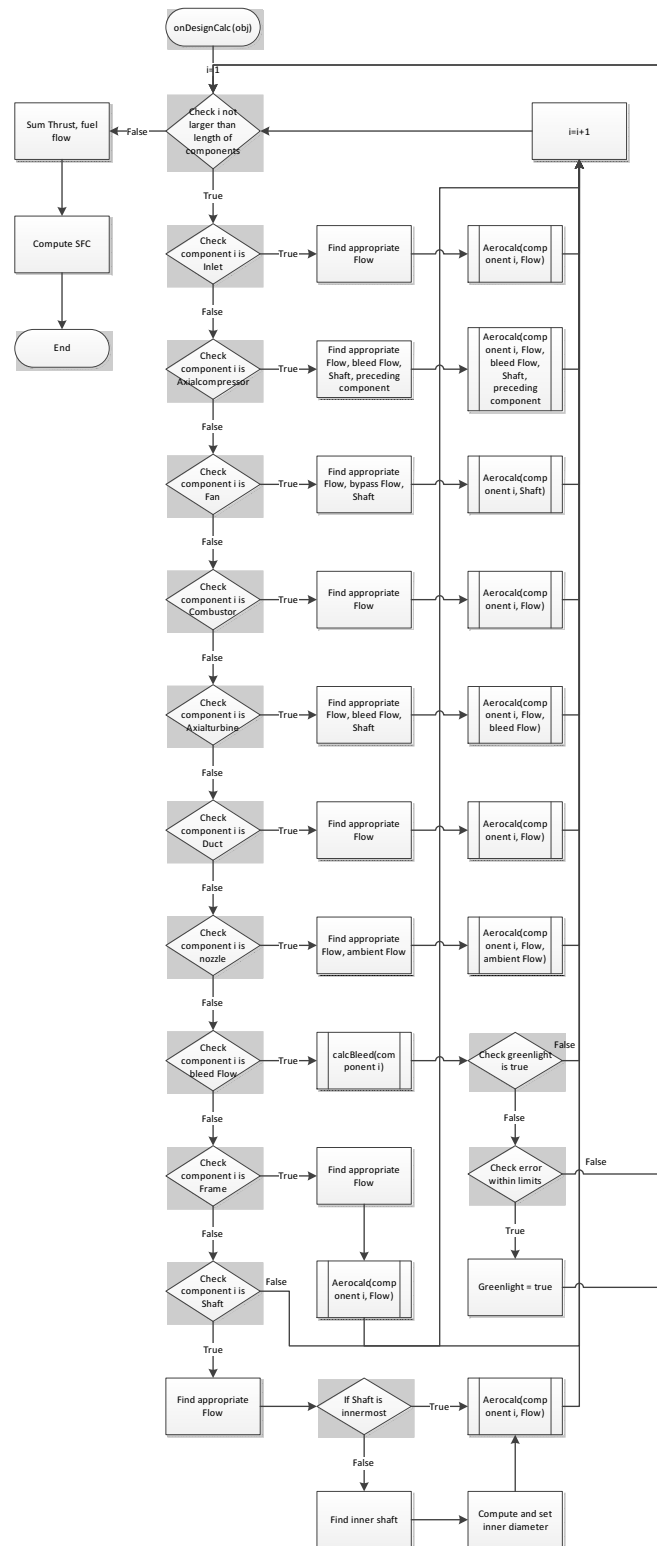


Figure 10.6: AeroEngine onDesignCalc() on-design calculation workflow flowchart

the greenlight boolean is set to true, but the code does reset the counter to 1 to cause a final re-iteration.<sup>6</sup> If both the bleed air and spool speed greenlight booleans are both found true and the error still remains below the threshold, the counter is simply incremented, allowing for continuation of the calculation and allowing the code to eventually exit the loop.

When the value of the counter variable finally exceeds the length of the components array, implying that all components have been calculated and converged, net and gross thrust values are extracted from all `Nozzle` components and added to obtain the total engine net and gross thrust values. Similarly, the fuel flow is extracted from each `Combustor` and summed to yield the total engine fuel flow  $\dot{m}_f$ . By dividing this fuel massflow by the net total thrust, the engine on-design SFC is then determined. Finally, these parameters are written to the properties of the `AeroEngine` object, concluding the on-design calculation procedure.

### THE MASSCALC() METHOD

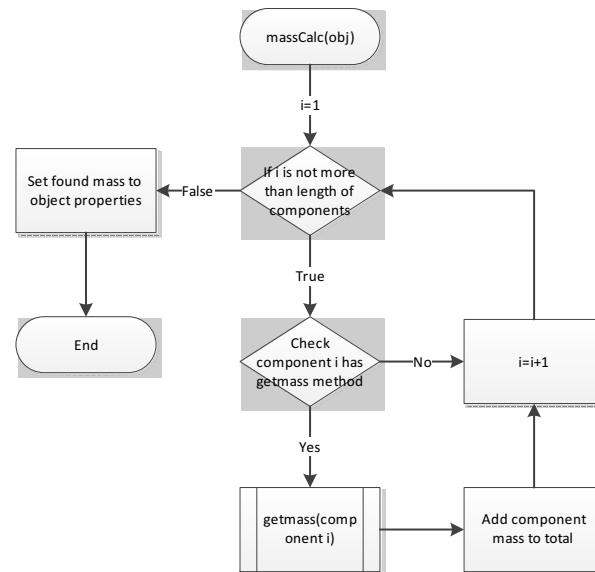


Figure 10.7: `AeroEngine` `massCalc()` mass calculation workflow flowchart

As the model is broken down into separate components in order to be able to say something about the whole by evaluating the sum of its parts (as per the philosophy behind a.o. the WATE method [14–16]), it is logical that the mass estimation method also follows this model. As can be seen from Figure 10.7, the `massCalc()` method works by going through the complete `components` array, checking for each entry whether that object has a `getmass()` method<sup>7</sup>. If it does, it calls this method, which is intended to calculate the mass of that object. It then adds the newly calculated mass of this component to a running total.

When all entries of the `components` array have been investigated in this manner, the thus-obtained total dry engine mass is assumed to be a good approximation of the total engine mass, and its value is therefore set as the `mass` property of the `AeroEngine` object.<sup>8</sup>

## 10.2. AXIALCOMPRESSOR

The `Axialcompressor` class serves as a model for an axial through-flow compressor with possibly multiple stages, each of which are represented as an `Axcompressorstage` object (see Section 10.4). As shown in Figure 10.8, the `Axialcompressor` is a subclass of `AeroComponent`, implying its status as model of a through-flow component.

While the class is mostly used as a container for its stages, it also handles important parts of the design and performance calculations itself.

<sup>6</sup>This is done to rule out effects of other inter-component interactions, as well as ensure convergence in case of engines not requiring bleed air cooling.

<sup>7</sup>For details on these methods for individual components, see Chapter 3.

<sup>8</sup>Further improvements may include the use of empirical-factor-based contributions of not-modelled engine components on top of the now calculated base engine mass, such as used in [5] and many others.

### Axialcompressor class diagram

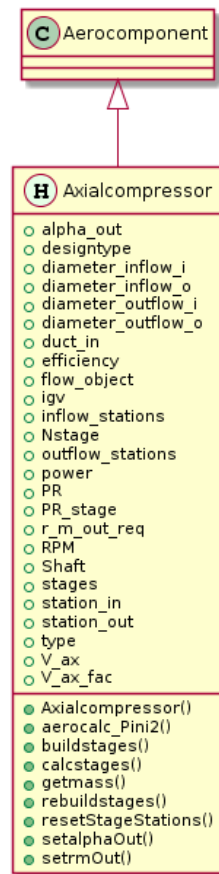


Figure 10.8: Class diagram for the Axialcompressor class.

### 10.2.1. PROPERTIES

As can be seen from Figure 10.8, besides the standard properties conveyed from the `Aerocomponent` super-class, the class has many other properties, which can be divided into descriptive values, design settings and container properties.

The descriptive properties are mainly used for storing output from the calculations, either for use in future calculations of the object or for reference by other objects. These properties include for example the `diameter_inflow_i`, `diameter_inflow_o`, `diameter_outflow_i` and `diameter_outflow_o` properties, which give the inner and outer diameters (in [m]) of the in- and outflow stations of the compressor. The `Nstage` property gives the number of stages present within the compressor, while the `inflow_stations` and `outflow_stations` properties are arrays with the station numbers of all the inflow and outflow stations of all stages and possible other objects within the compressor. The power property represents the required power (in [W]) for on-design operation of the compressor. Finally, `V_ax` is the axial outflow velocity in  $\left[\frac{\text{m}}{\text{s}}\right]$ .

The design setting properties are those which are primarily used to determine the design and/or performance of the compressor. These properties include `alpha_out`, which is the required compressor outflow angle in [rad], the `designtype` and `design` properties, which regulate the assumed mean radius progression over the compressor (currently unused) and the `efficiency`, which is the polytropic efficiency of the compressor stages  $\eta$ . The `PR` property contains the target total pressure ratio over the compressor, while the `r_m_out_req` property states the mean outflow radius (in [m]) required by the following component if applicable. The `RPM` is the rotational speed (in  $\left[\frac{\text{rev}}{\text{min}}\right]$ ) used for the design calculations of the compressor, the `Shaft` property contains the number of the corresponding `Shaft` object and the `station_in` and `station_out` properties contain the in- and outflow station identifiers. Finally, the `V_ax_fac` property sets the ratio between the in- and outflow axial velocity.

Some properties also serve as containers for other objects.<sup>9</sup> The most obvious one is the `stages` property, which is an array containing all the `Axcompressorstage` objects which make up the compressor, however, there are also several others. The `igv` and `duct_in` contain `InletGuideVane` and `Duct` classes, if these are used for the correction of inflow conditions on the compressor level, and otherwise are left empty. Finally, the `flow_obj` contains a reference to the `Flow`-class object modelling the flow through the compressor.

### 10.2.2. METHODS

Also from Figure 10.8, it can be seen that the `Axialcompressor` class defines 9 methods. These are its constructor `Axialcompressor()`, the stage creation methods `buildstages()` and `rebuildstages()`<sup>10</sup>, the stage number change garbage handler `resetStageStations()`, the compressor on-design performance and design routing `aerocalc_Pini2()`, the individual stage calculation wrapper `calcstages()`, the mass estimation function `getmass()` and the desired outflow angle and mean radius setting method `setalphaOut()` and `setrmOut()`. The latter two purely are used to set their respective properties and have no further use.

#### THE AXIALCOMPRESSOR() CONSTRUCTOR

Upon the creation of the `Axialcompressor` component, the constructor initially sets input values such as the compressor name, the identifiers of the associated `Flow` and `Shaft` objects, the in- and outflow station identifiers, the compressor polytropic efficiency  $\eta$ , the required pressure ratio `PR`, the outflow required axial velocity  $V_{\text{axout}}$ , the compressor material<sup>11</sup> and the initial estimate of the number of stages<sup>12</sup>. Next, based on the value of the `type` property of the `Axialcompressor`, it is determined whether the design should have a constant hub, mean or tip diameter, if the geometry is controlled.<sup>13</sup> Finally, it calls the `buildstages` method on the object, which is used to construct the `Axcompressorstage` objects and thereby fill the `stages` array of the `Axialcompressor`.

#### THE BUILDSTAGES() METHOD

The `buildstages()` method, as depicted in Figure 10.9, (re-)constructs the stages based on the properties of the `Axialcompressor` object, and is also used to rebuild the stages after for example a change in number of

<sup>9</sup>Actually, all of these contain references to the relevant objects, as all of these classes are set up as handle classes.

<sup>10</sup>The `rebuildstages()` function simply clears out the `stages` array, and then calling the `buildstages()` method, and is therefore not elaborated upon further.

<sup>11</sup>All parts of the compressor are assumed to be made of the same material.

<sup>12</sup>At the moment, this value is effectively obsolete, as the required number of stages will be estimated later if not set, and then recalculated, as described in Section 10.2.2. It therefore purely serves as a starting point.

<sup>13</sup>At the moment, this function is obsolete, as a relic from past code versions. It is however still required to be passed on later to the `Axcompressorstage` constructor.



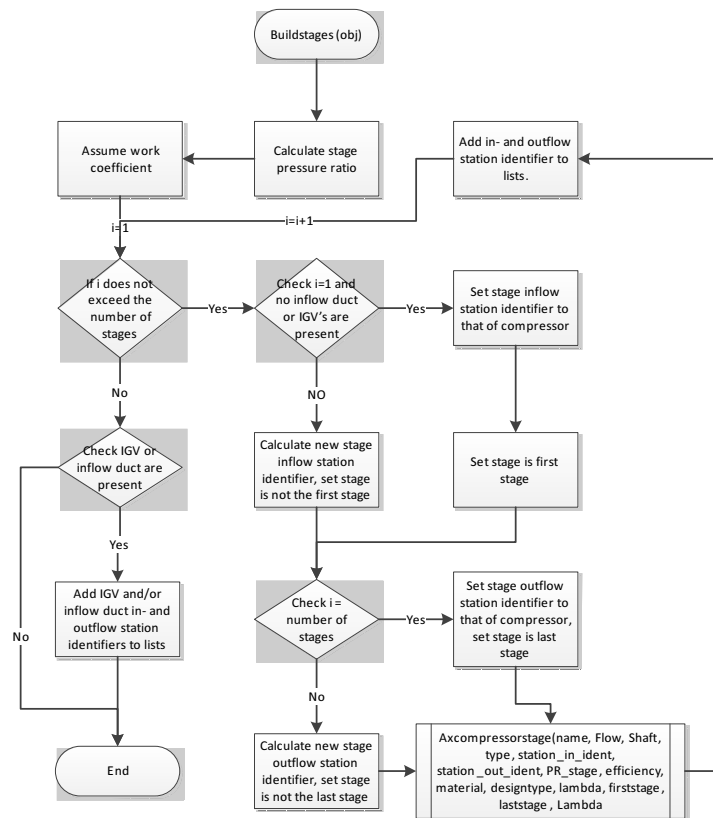


Figure 10.9: Axialcompressor buildstages stage (re-)creation workflow flowchart

stages. It starts by estimating the total pressure ratio per stage  $PR_{\text{stage}}$  based on the compressor overall total pressure ratio  $PR$  and the assumption of equal pressure ratios over all stages, yielding

$$PR_{\text{stage}} = PR^{\frac{1}{N_{\text{stages}}}} \quad (10.1)$$

It then assumed a work coefficient  $\lambda = 1$ , after which it starts a loop to create each individual stage.

In this loop, the method first checks whether the stage has the same inflow conditions as the compressor. This is the case when the stage is both the first stage and there are no inflow correction ducts or inlet guide vanes (IGV's) present for the total compressor.<sup>14</sup> If this is the case, the inlet station identifier for this stage is assumed to be equal to the inlet station identifier of the `Axialcompressor` object itself. Also, the `firststage` boolean is set to true. If however the stage is not the first stage, and/or inflow correction ducts or IGV's are present, the `firststage` boolean is set to false and the inflow station identifier is determined using

$$ID_{\text{stage}_{\text{in}}} = 100 \cdot ID_{\text{in}} + i - 1 + \text{IGV} + \text{Duct} \quad (10.2)$$

where  $ID_{\text{stage}_{\text{in}}}$  is the stage inflow station identifier,  $ID_{\text{in}}$  is the `Axialcompressor` inflow station identifier,  $i$  is the stage number, and `IGV` and `Duct` have the value of 1 if they are present, and 0 otherwise.<sup>15</sup> This naming system will hold up to 98 stages<sup>16</sup>, assuming that the compressor outflow station identifier is the inflow identifier plus 1. For both cases, the degree of reaction  $\Lambda$  is currently set to  $\Lambda = 0.5$ .

Next, it is checked whether the stage is the last stage of the compressor. If this is the case, the stage outflow station identifier is set equal to that of the compressor, and the `laststage` boolean is set to true for this stage. If the stage is not the last stage, the `laststage` boolean is false, and the stage outflow station identifier is determined using

$$ID_{\text{stage}_{\text{out}}} = 100 \cdot ID_{\text{in}} + i + \text{IGV} + \text{Duct} \quad (10.3)$$

<sup>14</sup>The latter two conditions are currently obsolete, as the requirements for both of these have since been transferred to the stages themselves, and their creation and presence is handled internally.

<sup>15</sup>As an example, the inflow station identifier of the 5th stage of a compressor with inflow station identifier 2 and both an inflow correction duct as well as IGV's will become  $ID_{\text{stage}_{\text{in}}} = 100 \cdot 2 + 5 - 1 + 1 + 1 = 206$ .

<sup>16</sup>Assuming the currently used maximum of 25 stages per compressor or turbine, this is sufficient.

which can be observed to yield a 1 higher outcome compared to Equation 10.2.

With the stage in- and outflow station identifiers and other stage-dependent parameters determined, the `Axcompressorstage` constructor is called, and the resulting object is stored as the  $i$ th entry in the `Axialcompressor` `stages` array. As the final act of this loop, the stage inflow and outflow identifiers are added to lists kept as object properties, which are later used in different methods to allow for garbage collection.

After the method has looped over all stages in this manner, the in- and outflow stations of any present IGV or inflow duct in the `Axialcompressor` object are also added to this list. This is the final act of the `buildstages` method.

#### THE `RESETSTAGESTATIONS()` METHOD

The `resetStageStations()` method acts as a garbage collector, removing any orphaned `Station` objects, which may appear upon a decrease in the number of stages and/or the removal of IGV's or inflow ducts. Upon calling of the `resetStageStations()` method, it first checks whether one of these conditions is the case.

If so, the method creates lists of expected inflow and outflow station identifiers associated with the new number of stages. Next, for the to-be-removed stages, the station identifiers of any associated stage-level IGV's and inflow ducts are also added to the lists of current in- and outflow station identifiers which are stored in the `inflow_stations` and `outflow_stations` properties.

The lists of new and old stations are then compared, with any station with an identifier appearing in the latter but not in the former being added to a to-be-destroyed list. The stations on this list are then deleted from both the core and bleed `Flow` objects.

Finally, the `rebuildstages()` method is called to clear and re-fill the compressor `stages` property array.

#### THE `AEROCALC_PINI2()` METHOD

The on-design calculation of the `Axialcompressor` is done by calling its `aerocalc_Pini2()` method, which needs the `Axialcompressor` object, the core flow and bleed flow `Flow` components, the appropriate `Shaft` component and the preceding component as arguments. Essentially, this method first guesstimates the required number of stages, if this is not already specified, based on the performance of a single stage. For this estimate, the method then checks whether the required pressure ratio lies within the achievable range, defined by the minimum and maximum allowable work coefficients, by calculating for these two conditions using the `calcstages()` method. If needed, the number of stages is adjusted until this is the case. The method then finds the appropriate value for the work coefficient to yield the required pressure ratio using a bisection method.

As can be seen in Figure 10.10, the method initially defines acceptable ranges for the flow and work coefficients, requiring that  $0.6 \leq \lambda \leq 1.2$  and  $0.4 \leq \phi \leq 1.1$ . [19, Sl. 4] Next, the expected outflow total conditions are estimated using the isentropic relations, yielding

$$\begin{aligned} p_{t_{out}} &= PR \cdot p_{t_{in}} \\ T_{t_{out}} &= T_{t_{in}} \cdot PR^{\frac{\gamma-1}{\gamma}} \end{aligned} \quad (10.4)$$

The estimated total temperature jump  $\Delta T_{t_{req}} = T_{t_{out}} - T_{t_{in}}$  over the compressor can then be determined. If the outflow angle  $\alpha_3$  is specified, the required outflow whirl velocity is also determined at this point. If no outflow angle is specified, the outflow whirl velocity is assumed to be equal to the inflow whirl velocity.

Next, as long as no previous estimate for the number of stages is available, a first-order estimate is made of the number of stages required, by determining the total temperature jump generated by a single stage for the maximum work coefficient  $\lambda_{max}$ <sup>17</sup> using the `rotorcalc_Pini_phi()` and `statorcalc_Pini_phi()` methods of the stage, dividing the required total temperature jump  $\Delta T_{t_{req}}$  by the thus-obtained value and rounding upwards. Also, this yields information on the required inflow angle and mean radius, which is communicated to the preceding component for use in the next iteration of the complete engine. In order to avoid unreasonable numbers of compressor stages, the number of stages must stay within the range of  $1 \leq N_{stages} \leq 25$ , and is reset to the violated boundary if needed. To ensure no `Station` objects become orphaned after this change of stage numbers and sufficient `Axcompressorstage` objects are present, the `resetStageStations()` method is then called.

With the first-order stage number estimate available, the method now checks whether this number of stages is actually capable of providing the required pressure ratio within the specified work coefficient limits.

<sup>17</sup>The maximum value is chosen as this should yield the minimum required number of stages. If this should prove insufficient later on, the number will be adjusted at that point.

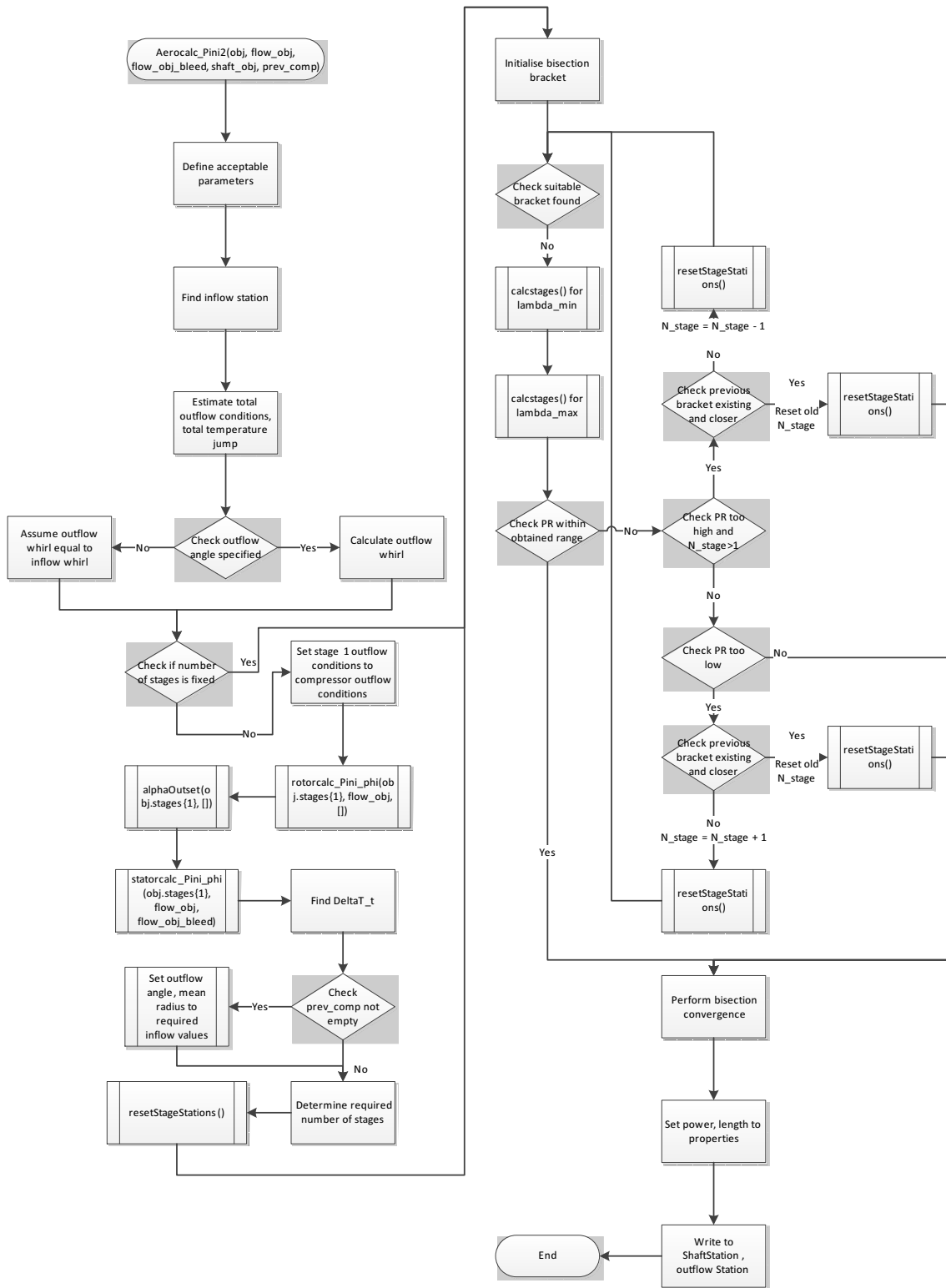


Figure 10.10: Axialcompressor aerocalc\_Pini2() on-design calculation workflow flowchart

The compressor is evaluated for  $\lambda_{\min}$  and  $\lambda_{\max}$  using the `calcstages()` method, after which it is checked whether the resulting errors for both have opposing signs, signifying that the desired pressure ratio is achievable for the assumed number of stages. If this is the case, the method continues onward. If however both errors are negative, this indicates that the generated pressure ratio was too small, and the estimated number of stages must therefore be increased. The number of stages is thus increased by 1, `resetStageSations()` is called, and the compressor is re-evaluated. If on the other hand both errors are positive, indicating a too large pressure increase, the number of stages is decreased by one<sup>18</sup> before being re-evaluated. In order to avoid oscillating behaviour, both the change in stage number as well as the lowest absolute error is saved upon a stage number change. When a new stage number change seems to be required, it is first checked whether the current error is not larger in magnitude than the previous error. If this would be the case, the stage number will revert back to the previous value and the method continues onward, breaking the loop, as this indicates oscillating behaviour.

Next, with a suitable (or as-close-as-possible) value for the number of stages found, a bisection convergence is performed to find the proper value of the work coefficient  $\lambda$ , again using the `calcstages()` method to obtain the resulting pressure ratio, RPM limit and power consumption. Convergence is assumed to be achieved for a pressure ratio error of  $\leq 1\%$ . If the pressure ratio error is larger, but both the upper and lower bounds can be seen to yield a same-sign error value, the work coefficient at the lowest-error boundary is assumed to be the closest possible match, and is evaluated once more to obtain the RPM limits and power consumption, before breaking out of the loop.

Upon convergence, the compressor power consumption and total length are written to the `Axialcompressor` object properties, with the limit RPM and power consumption also being written away to the appropriate `Shaftstation`, before ending the method.

#### THE CALCSTAGES() METHOD

The `calcstages()` method, a flowchart of which is depicted in Figure 10.11, is used to calculate the performance of a compressor for a given work coefficient  $\lambda$ . If present, it first calls the `aerocalc()` method on the compressor inflow duct, after which it starts calculating the separate stages in turn. For each stage, the outflow axial velocity  $V_{axout}$ , work coefficient  $\lambda$  and the shaft RPM are first updated to the proper values using the `outVaxset()`, `lambdaset()` and `RPMset()` methods of the `Axcompressorstage`. This also goes for the shaft diameter  $D_{shaft}$ , which is taken from the `Shaft` and set to the stages using their `shaftDiaset()` method. Next, the stage rotor is calculated by calling the `rotorcalc_Pini_phi()` method of the stage. If this stage is the first stage and no IGV's are present, the inflow angle is also specified in the call to this method, as neither the IGV's or previous stage stators are available to adjust this value. This is followed by a call to the `updateDshaft()` method of the `rotordisk` of the stage, in order to update it with the diameter of the shaft.

Next, the stator or IGV (if any) preceding the stage is calculated. If the stage is not the first, the outflow angle  $\alpha_3$  and mean radius  $r_{mout}$  of the preceding stator are set to the desired inflow values of the just calculated rotor using the `alphaOutset()` and `rmoutset()` methods, after which the `statorcalc_Pini_phi()` method is called on the preceding stator. If the stage is the first of the compressor, and IGV's are present, its mean radius is set in the same fashion, and the required outflow angle is simply specified as an argument in the IGV `aerocalc()` method.

The power produced by the stage is now added to a power runner variable. Also, the stage rpm limit is checked against the already obtained rpm limit. If it is smaller, it replaces the previous value, if it is larger, the previous value remains.

After all stages have been treated in this manner, the final stator is calculated using the compressor outflow angle and mean radius settings. Finally, the obtained pressure ratio is calculated by dividing the outflow total pressure by the inflow value, and the error is obtained by subtracting the target value from the obtained value, after which is it divided by the target to obtain a relative value.

#### THE GETMASS() METHOD

The `Axialcompressor` `getmass()` mass estimation method is relatively simple, looping over all stages as well as any possibly present IGV's and inflow ducts and calling their individual `getmass()` methods to obtain their masses, summing them and returning the found total compressor mass.

<sup>18</sup>In some cases, it may occur that the pressure ratio is too small for even a single-stage compressor. In that case, decreasing the number of stages is not possible, so the method continues onward anyway, accepting a too high pressure ratio from this compressor. Even though it decreases the modelling accuracy, it will not yield an inconsistent design, just an over-performing one.

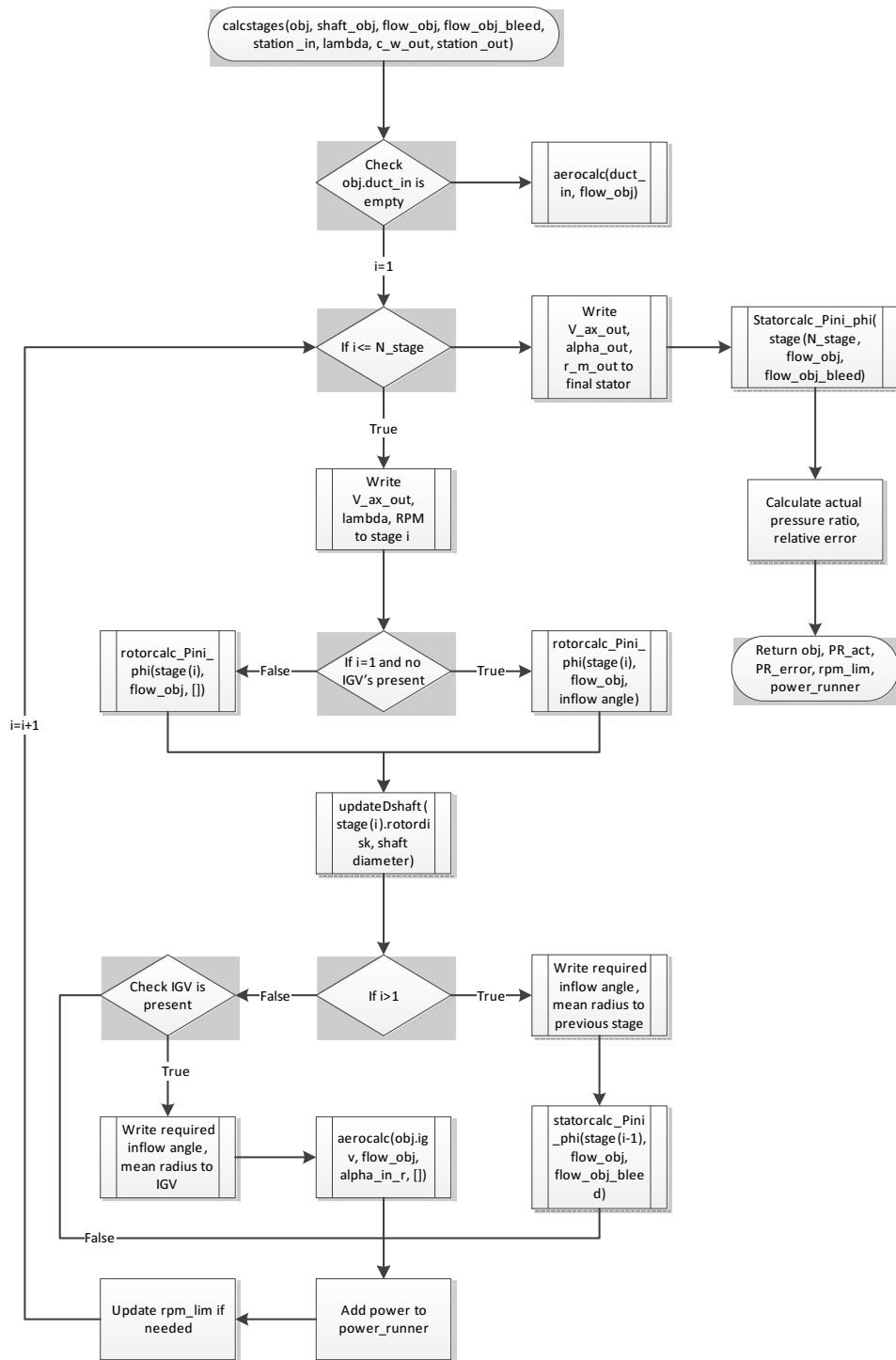


Figure 10.11: Axialcompressor calcstages on-design stage calculation workflow flowchart

## 10.3. AXIALTURBINE

Axialturbine class diagram

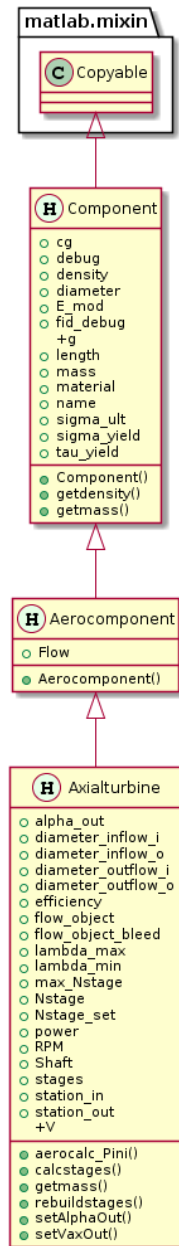


Figure 10.12: Class diagram for the Axialturbine class.

The Axialturbine class serves as a model for an axial through-flow turbine with possibly multiple stages.

### 10.3.1. PROPERTIES

Like the Axialcompressor class, the Axialturbine class is a subclass of the Aerocomponent superclass. Due to their similarities, many properties of the Axialcompressor class also re-appear in the Axialturbine class. These are not discussed here any further.

One of the “new” properties is the `flow_object_bleed` property, which contains (the handle of) the bleed Flow object. The `lambda_min` and `lambda_max` properties contain the allowable upper and lower limits for the work coefficient  $\lambda$ , while the `max_Nstage` gives the maximum allowed number of stages. Finally, the `Nstage_set` boolean states whether the number of stages is already specified at the start of the calculation,

however is currently practically superfluous.

### 10.3.2. METHODS

The `Axialturbine` class contains several methods. Besides its constructor `Axialturbine()`, it contains a stage-creating `rebuildstages()` method, the design and on-design performance calculation method `aerocalc_Pini()`, the individual stage calculation routine `calcstages()` and the mass calculation method `getmass()`. Finally, the `setAlphaOut()` and `setVaxOut()` methods are purely used for updating the properties defining the required outflow absolute angle and axial velocity, and are therefore not described in more detail.

#### THE AXIALTURBINE() CONSTRUCTOR

The `Axialturbine()` constructor is relatively simple, setting the turbine name, associated `Flow` and `Shaft` identifiers, in- and outflow `Station` identifiers, polytropic efficiency  $\eta$  and initial number of stages estimate  $N_{\text{stages}}$  to the appropriate `Axialturbine` properties. Next, it builds the stages, and sets the `firststage` property of the first stage.

#### THE REBUILDSTAGES() METHOD

The `rebuildstages()` method is intended to update the number of stages and the contents of the `stages` property array, as well as to function as a garbage collector for any orphaned `Station` objects.

After updating the `Nstage` property, the method creates lists of associated bleed and core flow `Station` identifiers for both the old and the new number of stages. After comparing these lists, any station appearing for the old situation which does not re-occur in the new situation is deleted.

Finally, if the new and old number of stages do not match, the `stages` property array is repopulated with `Axturbinstage` objects. The outflow angle of the last stage is set to the required turbine outflow angle using its `alphaOutset()` function, and the `firststage` property of each stage is set to the appropriate value using the `setFirstStage()` method.

#### THE AEROCALC\_PINI() METHOD

The `aerocalc_Pini()` method is effectively used to determine the proper number of turbine stages  $N_{\text{stages}}$  and work coefficient  $\lambda$  in order to yield the required amount of generated power.

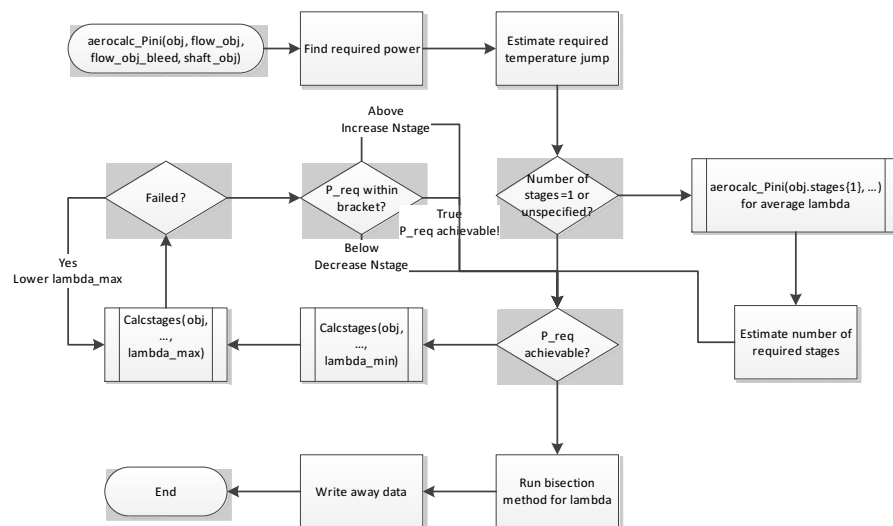


Figure 10.13: `Axialturbine aerocalc_Pini()` on-design turbine calculation workflow flowchart

As shown in Figure 10.13, the method starts similar to its compressor counterparts, however besides searching for just the relevant `Station` objects from the `Flows`, also all `Shaftstation` objects on the relevant `Shaft` are checked, from which the total required power is determined by summing the required power and subtracting the total generated power of all other `Shaftstations`. The resulting total required power  $P_{\text{req}}$  is then translated into an estimated total temperature jump  $\Delta T_{\text{test}}$  using

$$\Delta T_{\text{test}} = \frac{P_{\text{req}}}{\eta \cdot \dot{m}_{\text{in}} \cdot c_p} \quad (3.18)$$

Next, if the number of stages  $N_{\text{stages}}$  is undefined or  $N_{\text{stages}} = 1$  (which is the default value), the first stage is evaluated by itself to estimate the single-stage total temperature jump, and through that guesstimate the required number of stages. For this calculation, the mean value of the upper and lower work coefficient limits is used, as the total temperature jump which will be found during this calculation is likely to be higher than the drop due to work extraction, as cooling is also taken into account. The determined number of stages is then checked for reasonability, resulting in too low ( $N_{\text{stages}} < 1$ ) or too high ( $N_{\text{stages}} > 25$ ) values being excluded, with the stage number being set to the violated boundary. Finally, the `rebuildstages()` method is used to ensure the correct `Axturbinestage` and `Station` objects are created and removed.

Next, the feasibility of achieving the required power with the selected number of stages is checked. A calculation over all stages using the `calcstages` method is done first for the lower work coefficient limit  $\lambda_{\text{min}} = 2$  [19, Sl. 4], followed by an evaluation for the upper work coefficient  $\lambda_{\text{max}} = 6$  [19, Sl. 4].<sup>19</sup> If the  $P_{\text{gen}} > P_{\text{req}}$  for both limits, the number of stages is decreased by 1, while if  $P_{\text{gen}} < P_{\text{req}}$  for both cases, the number of stages is increased, after which the limits are re-evaluated. Again, like the compressor, provisions are incorporated to avoid oscillations, as well as to avoid invalid stage numbers.

With a suitable (or as-close-as-possible) number of stages found, a bisection method is now used, calling the `calcstages()` method with varying values for  $\lambda$ , in order to find the proper work coefficient corresponding to  $P_{\text{gen}} = P_{\text{req}}$ , with a maximum power generation error of 1%, and a minimum bracket size of  $0.5\% \cdot \lambda$  as exit conditions.

After this  $\lambda$  has been found, the obtained generator power is set to the `Axialturbine` object's properties, and the lowest RPM limit of all stages is written along with the generated power to the appropriate `Shaftstation`, ending the method.

#### THE `CALCSTAGES()` METHOD

The `calcstages()` method is used to bundle up all calculations for calculating turbine performance for a given number of stages  $N_{\text{stages}}$  and work coefficient  $\lambda$ . It first assigns the proper RPM,  $D_{\text{shaft}}$  and  $\lambda$  values to each stage, after which for each stage individually, the stage `aerocalc_Pini()` method is called, its rotordisk inner diameter is updated with the outer diameter of the shaft, and its generated power and length are added to the running total. Upon evaluation of all stages, the generated power is compared to the required power, and both an absolute and relative error value are computed. The method finally returns these values, along with the actual generated power value and the lowest encountered stage RPM limit.

#### THE `GETMASS()` METHOD

The `getmass()` mass estimation method for the `Axialturbine` is relatively simple: it loops over all the `Axturbinestage` objects of its `stages` array, calling their `getmass()` methods and adding the returned values to a running total. This summation is then set as the total turbine mass.

## 10.4. AXCOMPRESSORSTAGE

The `Axcompressorstage` class is a model for a single axial compressor stage, intended as a component of an `Axialcompressor` object, and should not be used separately.

### 10.4.1. PROPERTIES

As the class models a through-flow component, it again is a sub-class of the `Aerocomponent` superclass. Next to the properties inherited through this mechanism, Figure 10.14 shows that the `Axcompressorstage` class has many more. Some of these are purely used to store the results of the design and/or performance calculation for reference in future calculations or by other objects, some are used to store design or calculation inputs, and some are used to store other objects.

Among the characteristics-storing properties are `alpha_in_r`, `alpha_in_s`, `alpha_out_r` and `alpha_out_s`, which respectively give the rotor and stator absolute in- and outflow angles in [rad], while the relative rotor in- and outflow angles are stored as `beta_in_r` and `beta_out_r`. The in- and outflow hub and tip diameters (in [m]) are stored in the `diameter_inflow_i`, `diameter_inflow_o`, `diameter_outflow_i` and `diameter_outflow_o` properties, while the `HT_ratio` property gives the inflow hub-to-tip ratio. The numbers of blades on the stage rotor and stator are stored in the `Nbladerot` and `Nbladestat` properties,

<sup>19</sup>For some excessive combinations of too high work coefficients and too high numbers of stages, the code may throw an error due to too much energy being extracted from the flow, yielding negative absolute outflow temperatures and/or pressures, which clearly would be an issue. Therefore, these errors are caught if occurring, after which a re-evaluation is done for a slightly lower maximum work coefficient. This is continued until a working upper limit is found.



## Axcompressorstage class diagram

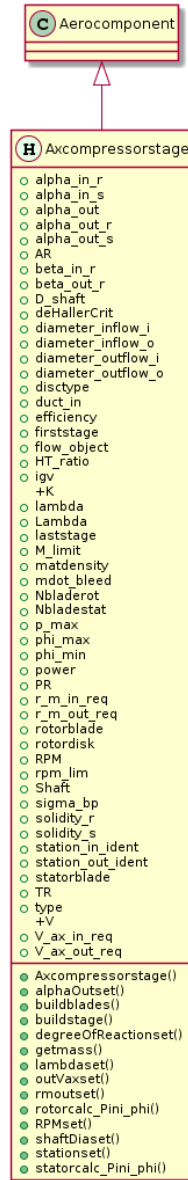


Figure 10.14: Class diagram for the Axcompressorstage class.

while the maximum encountered total pressure (in [Pa]) is stored in the `p_max` property and the power property gives the power requirements of the stage in [W]. While also initially specified as a target value, the `PR` property contains the actually obtained total pressure ratio over the stage, with the `r_m_in_req` property containing the mean inflow radius required by the stage, `rpm_lim` containing the stage maximum allowable rotational velocity, `sigma_bp` stating the rotor blade pull stress on the rotor disk in [Pa] and `solidity_r` and `solidity_s` containing the solidities of rotor and stator. Finally, the `V_ax_in_req` property gives the required rotor inflow axial velocity in  $\left[\frac{\text{m}}{\text{s}}\right]$ , if applicable.

The properties used primarily for storing settings include the `alpha_out` property, which specifies the stage outflow angle in [rad] if needed. The `AR` property specifies the initial assumption for the blade aspect ratio, the `deHallerCrit` property specifies the used value for the deHaller criterion, the `type` and `disctype` properties set the type of the compressor and its rotor disc, while the `efficiency` stores the polytropic efficiency used for the calculations. The `firststage` and `laststage` properties are booleans indicating whether or not the stage is the first and/or last of the compressor. The property `K` contains a blade volume factor, used for mass estimation, while the `lambda` and `Lambda` properties store the work coefficient and degree of reaction<sup>20</sup> used. The `M_limit` property indicates the limit Mach number used for calculation of the limit RPM, while `mdot_bleed` sets the post-stage bleed offtake massflow in  $\left[\frac{\text{kg}}{\text{s}}\right]$ . The range of allowed flow coefficient  $\phi$  values is defined by the `phi_min` and `phi_max` properties, while the required outflow mean radius and axial velocity (if applicable) are specified in the `r_m_out_req` and `V_ax_out_req` properties. The stage rotor rotational velocity, which is taken from the shaft, is also stored in the `RPM` property, the `Shaft` property indicates the number of the shaft associated with the compressor, while the in- and outflow station identifiers are stored in `station_in_ident` and `station_out_ident`. Finally, the blade taper ratio is set using the `TR` property, and the `V` property defines the compressor axial (outflow) velocity.

Some properties are also purely intended as containers for (references to) other objects. These include the `duct_in` and `igv` properties, which contain `Duct`- and/or `InletGuideVane`-class objects to correct the inflow conditions if necessary. The object describing the flow through the stage is stored in the `flow_object` property, while the objects modelling the blades and rotor disk are stored in the `rotorblade`, `statorblade` and `rotordisk` properties.

### 10.4.2. METHODS

As can be seen from Figure 10.14, the `Axcompressorstage` class defines or re-defines several methods. Construction of the object itself is handled by its constructor `Axcompressorstage()` and the `buildstage()` and `buildblades()` methods, while the design and on-design performance calculations are performed using the `rotorcalc_Pini_phi()` and `statorcalc_Pini_phi()` methods. Estimation of the stage mass is done through the `getmass()` method. Finally, there are also several `*set()` methods, which are purely used to set a given input property to the provided value. Therefore, these are not worked out further here.

#### THE AXCOMPRESSORSTAGE() CONSTRUCTOR

The `Axcompressorstage` constructor, similar to the constructor of the `Axialcompressor` class, almost purely sets the inputs to the object properties. These include the object name, the identifiers of the associated `Flow` and `Shaft` objects, the compressor type, the in- and outflow station identifiers, the stage pressure ratio `PR`, polytropic efficiency  $\eta$ , axial outflow velocity  $V_{\text{ax,out}}$ , stage material, compressor design type<sup>21</sup>, stage work coefficient  $\lambda$ , first and last stage indication booleans and the degree of reaction  $\Lambda$ . Based on the specified compressor type, the stage's blade aspect ratios `AR` are initially set to guesstimated values of  $\text{AR} = 4$  for a normal compressor. The constructor then calls the `buildstage()` method of the `Axcompressorstage` object to actually convert these properties into something useable.

#### THE BUILDSTAGE() METHOD

The `buildstage()` method, as shown in Figure 10.15, initially checks the compressor type to set a blade volume factor  $K$ . This factor is later used to relate blade height and aspect ratio to blade volume, as described by [15, Eq. 3]. For compressors, its value depends on the blade hub-to-tip ratio  $\frac{H}{T}$ , being set to  $K = 0.120$  for  $\frac{H}{T} \leq 0.8$  and  $K = 0.120 + 0.04 \cdot \left(\frac{H}{T} - 0.8\right)$  for  $\frac{H}{T} > 0.8$ . [15, Eq. 4] With this factor known, the `buildblades()` method is called on the `Axcompressorstage` object to create the objects representing the stator and rotor blades. Then for both the stator and the rotor, the solidity  $\sigma$  is estimated. If the outflow axial velocity  $V_{\text{ax,out}}$

<sup>20</sup>The degree of reaction  $\Lambda$ , while usually assumed as an input, is sometimes also recalculated (see Section 3.1).

<sup>21</sup>Currently unused

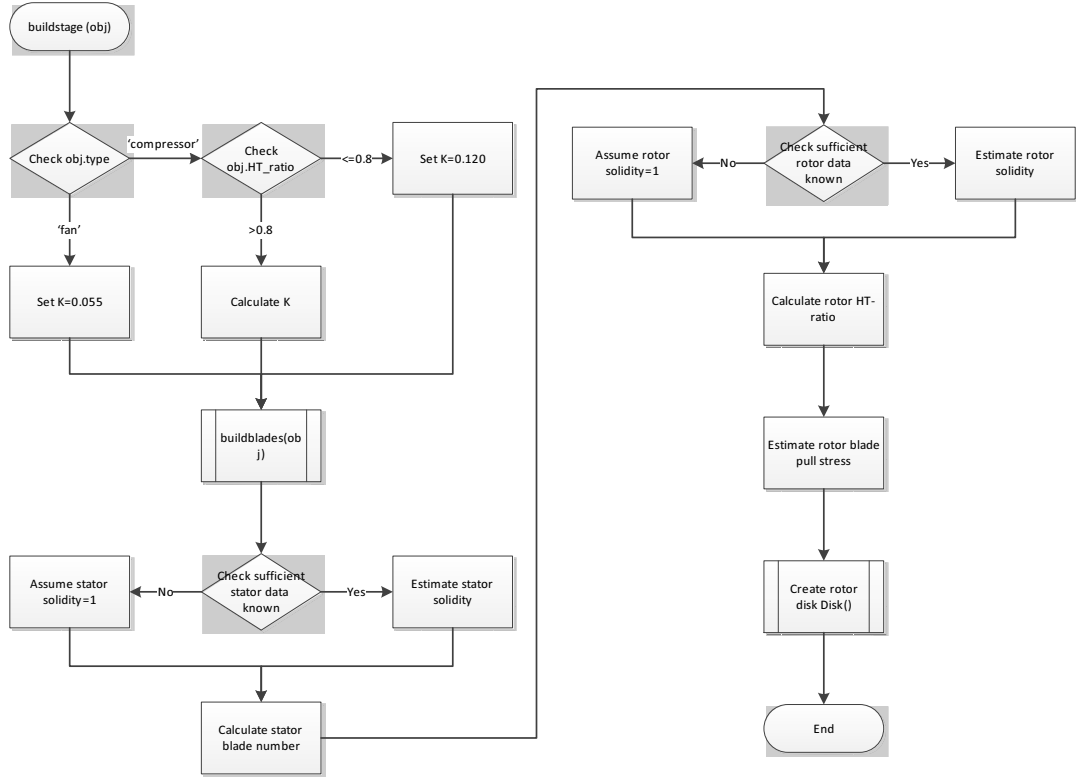


Figure 10.15: Axcompressorstage buildstage() stage (re-)creation workflow flowchart

and the relevant angles ( $\beta_1$  and  $\beta_2$  for the rotor,  $\alpha_2$  and  $\alpha_3$  for the stator) are known, the solidity is calculated based on an estimated diffusion factor  $DF = 0.45$  [18, p. 628] using Equation 3.7, which is a rewritten form of [35, Eq. 5.18]

$$\sigma_s = \frac{1}{\left(DF - 1 + \frac{\cos \alpha_2}{\cos \alpha_3}\right) \cdot 2} \cdot \frac{1}{\cos \alpha_2 \cdot (\tan \alpha_2 - \tan \alpha_3)}$$

$$\sigma_r = \frac{1}{\left(DF - 1 + \frac{\frac{V_{ax}}{\cos \beta_2}}{\frac{V_{ax}}{\cos \beta_1}}\right) \cdot 2 \cdot \frac{V_{ax}}{\cos \beta_1}} \cdot \frac{1}{|V_{ax} \cdot (\tan \alpha_2 - \tan \alpha_1)|}$$
(10.5)

If any of these variables are not (yet) available, the solidity of the stator or rotor is estimated as  $\sigma = 1$ . With the solidity obtained or assumed, the number of blades  $N_{\text{blades}}$  in the stator or rotor are then calculated by using the definition of the solidity  $\sigma = \frac{c}{s}$  (with  $c$  being the blade chord and  $s$  the blade spacing) and the known blade hub diameter  $D_{\text{hub}}$ , yielding

$$N_{\text{blades}} \geq \pi \cdot D_{\text{hub}} \cdot \frac{\sigma}{c}$$
(10.6)

Next, the rotor hub-to-tip ratio is recalculated, after which the blade pull stress  $\sigma_{\text{bp}}$  of the rotor blades on the rotor disk is estimated. This is done using [15, Eq. 7], which is based on the RPM, the mean diameter  $D_{\text{mean}}$ , material density  $\rho$ , blade taper ratio TR and rotor hub-to-tip-ratio  $\frac{H}{T}$ , according to Equation 3.9

$$\sigma_{\text{bp}} = \frac{12 \cdot \rho_{\text{blade}} \cdot \left(\frac{\text{RPM}}{60} \cdot r_{\text{tip}}\right)^2}{g \cdot \text{TR}} \cdot \left( \frac{1 - \left(\frac{H}{T}\right)^2}{2} + \frac{\text{TR}}{12} \cdot \left(1 - \frac{H}{T}\right) \cdot \left(1 + 3 \cdot \frac{H}{T}\right) \right)$$
(3.9)

With this variable known, the rotor Disk can now also be created by calling its constructor, as described in Section 10.12.

#### THE BUILDBLADES() METHOD

The buildblades() method of the Axcompressorstage class first checks whether any information is known about the rotor and stator inner and outer diameters. If this is not the case, these values are approximated by assuming a linear change in radii over the length of the stage, with the rotor and the stator also being assumed equal in length. Next, the stator and rotor blades are created by calling the Blade class constructors

(see Section 10.11) with the blade names, inner and outer diameters, aspect ratio,  $K$ -factor and material, and storing the resulting Blade objects in the statorblade and rotorblade properties of the stage. Finally, the stage length is estimated by summing the lengths of both blades.

#### THE ROTORCALC\_PINI\_PHI() METHOD

The on-design calculations for the compressor stage rotor and stator have been split up into two functions, as the stator calculation needs information of the next rotor stage in order to determine the desired outflow stator outflow angle. The rotor calculation is performed by the `rotorcalc_Pini_phi()` method, while the stator calculation is done using the `statorcalc_Pini_phi()` method.

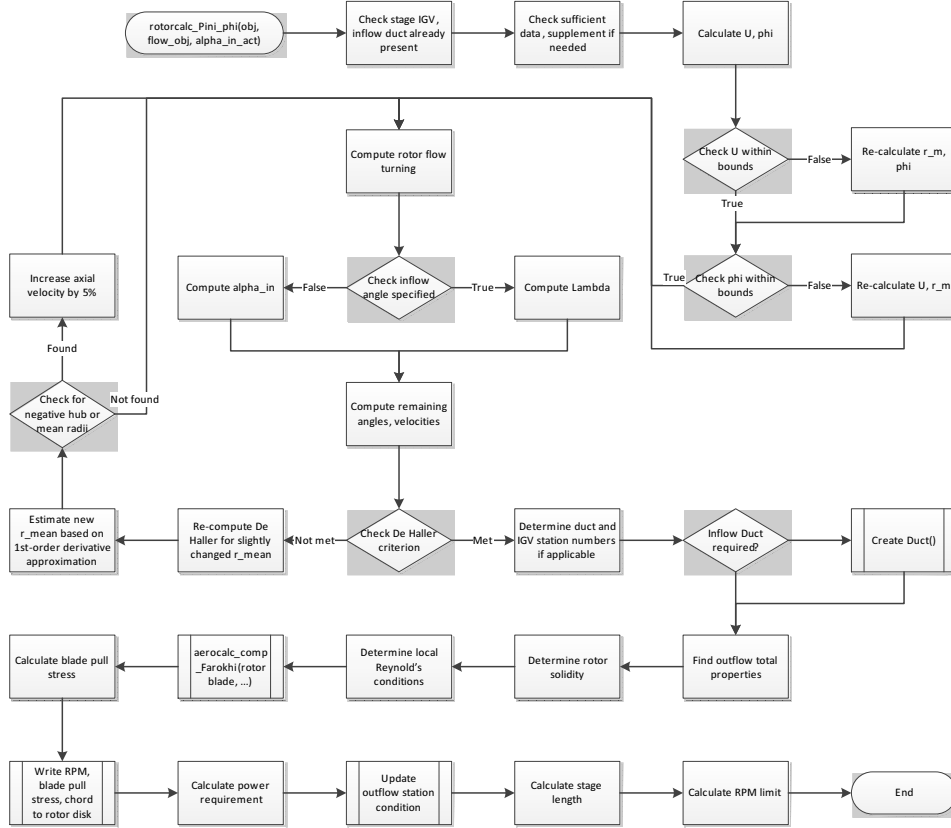


Figure 10.16: Axcompressorstage rotorcalc\_Pini\_phi() on-design rotor calculation workflow flowchart

The `rotorcalc_Pini_phi()` method, as depicted in Figure 10.16, performs the sizing of the rotor. In essence, this method first determines whether the inflow conditions yield an acceptable design and, if not, then changes the inflow mean radius or axial velocity until an acceptable design is obtained. An inflow correction duct is then added if needed, after which the outflow properties are determined, followed by the blade and disk characteristics and power requirements.

The inputs for the `rotorcalc_Pini_phi()` method are the stage object, the corresponding Flow object and the inflow angle  $\alpha_1$ . The latter may also be left empty, to signify the rotor is allowed to determine the inflow angle itself. After checking whether the stage already has IGV's and/or an inflow correction duct, the method searches the Flow for the appropriate in- and outflow Station objects. It also check whether the degree of reaction  $\Lambda$ , the inflow angle  $\alpha_1$  and the required outflow radius  $r_{m_{outreq}}$  are specified and otherwise defaults them to  $\Lambda = 0.5$ ,  $r_{m_{out}} = r_{m_{in}}$  and an empty variable for  $\alpha_1$ .

Next, based on the inflow mean radius  $r_{mean_{in}}$  and the shaft RPM, the mean-line rotor velocity  $U_{mean}$ <sup>22</sup> and flow coefficient  $\phi$  are computed using

$$U_{mean} = r_{m_{in}} \cdot 2 \cdot \pi \cdot \frac{RPM}{60} \quad (10.7)$$

$$\phi = \frac{V_{ax}}{U_{mean}} \quad (10.8)$$

<sup>22</sup>The mean radius is assumed not to change over the rotor, therefore the value for  $U_{mean}$  is also assumed constant over the rotor.

Some sensibility tests are then performed on these values. First of all, as [18, p. 628] states that  $U_{\text{mean}} \leq 550 \left[ \frac{\text{m}}{\text{s}} \right]$  should be the case,  $U_{\text{mean}}$  is reset to this limit if exceeded, after which a new required inflow mean radius  $r_{\text{min,req}}$  is computed using a rewritten form of Equation 10.7, and the associated  $\phi$  is also recalculated using Equation 10.8. The flow coefficient is then also checked to be within the bounds of  $0.4 \leq \phi \leq 1.1$  [19, Sl. 4].<sup>23</sup> Again, if a limit is exceeded, the flow coefficient is set to that limit value, after which  $U_{\text{mean}}$  is updated through a rewritten Equation 10.8, and then translated into an updated required inflow mean radius through a rewritten Equation 10.7.

Next, the method enters an iteration loop for as long as the DeHaller criterion has not been met. The first step is to determine the amount of flow turning  $\Delta C_w$  from the work coefficient  $\lambda$  and the current value for the mean-line rotor velocity  $U_{\text{mean}}$  using

$$\Delta C_w = \lambda \cdot U_{\text{mean}} \quad (10.9)$$

If no inflow angle  $\alpha_1$  has been specified, it is estimated based on the degree of reaction  $\Lambda$ ,  $\lambda$  and  $\phi$ , according to

$$\tan \alpha_1 = \frac{-\Lambda - \frac{\lambda}{4} + 1}{\phi} \quad (10.10)$$

If however  $\alpha_1$  is specified, Equation 10.10 is rewritten to yield the degree of reaction  $\Lambda$ .

With  $\alpha_1$  and  $\Lambda$  thus determined, the pre- and post-rotor absolute and relative velocities can now be calculated. First, the pre- and post-rotor absolute and relative whirl velocity components  $C_{w1}$ ,  $C_{w2}$ ,  $W_{w1}$  and  $W_{w2}$  are determined using

$$\begin{aligned} C_{w1} &= V_{\text{ax,in}} \cdot \tan \alpha_1 \\ W_{w1} &= C_{w1} - U_{\text{mean}} \\ C_{w2} &= C_{w1} + \Delta C_w \\ W_{w2} &= C_{w2} - U_{\text{mean}} \end{aligned} \quad (10.11)$$

From these whirl velocities, the absolute and relative pre- and post-rotor flow angles  $\alpha_2$ ,  $\beta_1$  and  $\beta_2$  can now be calculated by taking the tangent of the whirl component divided by the axial velocity, as can the absolute and relative velocities  $V_1$ ,  $V_2$ ,  $W_1$  and  $W_2$  through the Pythagorean theorem.

Next, the DeHaller criterion is tested. This criterion [18, p. 628]<sup>24</sup> states that

$$\text{DH} = \frac{W_2}{W_1} \geq \text{DH}_{\text{crit}} = 0.72 \quad (10.12)$$

If this criterion is met, the code exits the loop and continues on. If this criterion is not met however, this indicates a too large decrease in velocity over the rotor blade, which may cause the boundary layer of the blade to separate under the adverse pressure gradient. To solve this, there are several options. One could increase the inflow relative velocity  $W_1$  for a constant  $\Delta C_w$ , implying an increase of the inflow whirl velocity  $C_{w1}$ , inflow axial velocity  $V_{\text{ax}}$  (coupled when inflow angle is specified) or decrease the mean rotational velocity  $U_{\text{mean}}$ , or alternatively decrease the  $\Delta C_w$ , implying a decrease in either the work coefficient  $\lambda$  or the mean rotational velocity  $U_{\text{mean}}$ . As  $\lambda$  is used as the control variable, it is undesirable to change its value within the process. It can also be seen that decreasing  $U_{\text{mean}}$  would have a positive effect through both mechanisms, so therefore, the choice was made to vary the mean radius  $r_{\text{mean}}$  (which due to the fixed RPM directly and solely influences  $U_{\text{mean}}$ ).

Therefore, a new, shifted, value for  $r_{\text{mean}}$  is assumed, for which new values of  $U_{\text{mean}}$ ,  $\phi$ ,  $\Delta C_w$ ,  $\alpha_1$  or  $\Lambda$  (depending on whether the inflow angle was specified), and all the velocities are determined. This then leads to a new estimate for the DeHaller number. By taking the difference between the DeHaller number error values for the original and shifted case, and dividing this by the step in  $r_{\text{mean}}$ , an error slope is estimated, which is used to estimate a new value for  $r_{\text{mean}}$ .<sup>25</sup> The method then re-calculates  $U_{\text{mean}}$ , and also makes an estimate of the inflow hub radius  $r_{\text{hub}}$ .<sup>26</sup> If this value ends up positive and larger than the shaft radius, the method iterates the DeHaller criterion loop again using the new  $r_{\text{mean}}$  and associated  $U_{\text{mean}}$  and  $\phi$ . If this however turns out to be negative or smaller than the shaft radius, the inflow axial velocity is increased by 5%, and  $r_{\text{mean}}$  defaults back to that of the inflow station, after which the DeHaller criterion loop is re-iterated with these values.

When the method drops out of the DeHaller criterion loop, indicating that a set of variables has been found for which the DeHaller criterion has been met, a Duct is added to the stage if the required and present

<sup>23</sup>It is interesting to note that [18, p. 628] cites an acceptable range of  $0.3 \leq \phi \leq 0.9$  instead.

<sup>24</sup>It is interesting to note that [17, p. 197] states an acceptable DeHaller number of 0.69.

<sup>25</sup>Some limitations on the maximum size of the jump relative to the original  $r_{\text{mean}}$  are set for stability reasons.

<sup>26</sup>Assuming a constant mean radius and increasing density, the inflow station will have the largest area, and is therefore chosen as the most critical point to check.

inflow conditions ( $r_{\text{mean}}$ ,  $V_{\text{ax}}$ ) do not match. If an inflow correction duct however is no longer necessary but was already present, both the obsolete Duct and the now-orphaned associated Station are deleted.

Next, the post-rotor total flow properties  $T_{t_2}$  and  $p_{t_2}$  are estimated, using [18, Eq. 8.23] and [17, Eq. 2.15]

$$\begin{aligned} T_{t_2} &= T_{t_1} + \frac{U_{\text{mean}} \cdot C_{w_2} - U_{\text{mean}} \cdot C_{w_1}}{c_p} \\ p_{t_2} &= p_{t_1} \left( \frac{T_{t_2}}{T_{t_1}} \right)^{\frac{\gamma-1}{\gamma \eta}} \end{aligned} \quad (10.13)$$

The Lieblein criterion [20, Sl. 8] is then used to estimate the rotor solidity  $\sigma_r$  based on an assumed Diffusion Factor  $DF = 0.45$  [18, p. 628] according to

$$\sigma_r = \cos \beta_1 \cdot \frac{\tan \beta_1 - \tan \beta_2}{2 \cdot (DF - 1 + \frac{\cos \beta_1}{\cos \beta_2})} \quad (10.14)$$

In order to allow estimation of the rotor blade characteristics, also the inflow kinematic viscosity  $\nu_1$  and a typical Reynolds number  $Re_{\text{typ}} = 5 \cdot 10^5$  [18, p. 628] are required. The kinematic viscosity is determined using Sutherland's equation [21, Eq. 2.81] and assuming air as the working fluid, according to

$$\begin{aligned} \mu_1 &= \mu_0 \cdot \left( \frac{T_1}{T_0} \right)^{\frac{3}{2}} \cdot \frac{T_0 + S}{T_1 + S} \\ \nu_1 &= \frac{\mu_1}{\rho_1} \end{aligned} \quad (10.15)$$

where  $\mu_0 = 1.7894 \cdot 10^{-5} \left[ \frac{\text{kg}}{\text{m} \cdot \text{s}} \right]$  is the ISA reference viscosity,  $T_0 = 288.16 [\text{K}]$  is the ISA reference temperature and  $S = 110 [\text{K}]$  is the Sutherland temperature for air. Together with the annulus inflow passage height, guesstimated hub and tip thickness-to-chord ratios of 10% and 3% [18, p.632], the solidity, inflow relative velocity and Mach number and relative blade angles, these are passed to the rotor Blade's `aerocalc_comp_Farokhi()` method, which handles calculation of the blade properties. With the blade properties known, the rotor disk properties can then be estimated. First, the blade pull stress is estimated using Equation 3.9, after which this value, along with the RPM and axial blade chord, are passed to the `Disk` object.

Finally, the stage power consumption  $P_{\text{stage}}$  is determined from

$$P_{\text{stage}} = c_p \cdot \dot{m}_{\text{in}} \cdot (T_{t_2} - T_{t_1}) \quad (10.16)$$

and the RPM limit is estimated based on a maximum rotor tip Mach number  $M_{\text{lim}}$  according to

$$\text{RPM}_{\text{lim}} = 60 \cdot \frac{\sqrt{(M_{\text{lim}} \cdot a)^2 - V_{\text{ax}}^2}}{2 \cdot \pi \cdot r_{\text{tip}}} \quad (10.17)$$

#### THE STATORCALC\_PINI\_PHI() METHOD

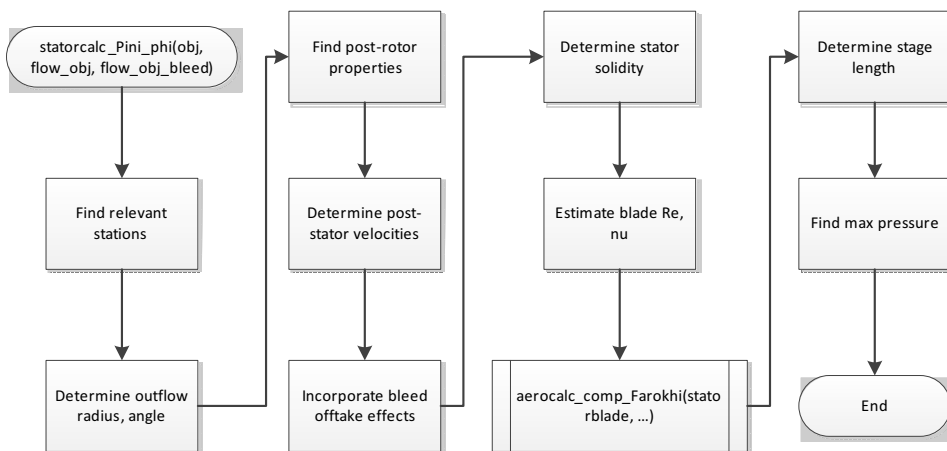


Figure 10.17: Axcompressorstage statorcalc\_Pini\_phi() on-design stator calculation workflow flowchart

The `statorcalc_Pini_phi()` method, as shown in Figure 10.17, is intended to handle calculation of the stage stator section. With in- and outflow conditions specified by the preceding and following rotors, this method is relatively simple.

After the method has started by finding the relevant stations in the core and bleed Flow objects, the outflow mean radius  $r_{\text{mean}_{\text{out}}}$  and angle  $\alpha_3$  are determined. If the required outflow mean radius from the following rotor lies within a reasonable range<sup>27</sup> from the preceding rotor outflow mean radius, it is set to the required value. If however these bounds are exceeded, the outflow mean radius is set to the violated boundary and the following stage is left to solve the discrepancy with an inflow correction duct. If no required mean outflow radius is specified, the stage outflow mean radius value from the previous computation is used if available. Otherwise, it defaults to the stage inflow mean radius. The outflow angle  $\alpha_3$  is set to the specified value, or, if no value is specified, to the stage inflow angle  $\alpha_1$ . Also, part of the computation from the `rotorcalc_Pini_phi()` method is repeated in order to obtain the required post-rotor flow conditions data.

Next, the post-stator velocities are determined from the specified axial velocity and required outflow angle. With these velocities known, and the assumption of no change to the total properties over the stator, the new outflow conditions are now written to the outflow Station. If there is any bleed air massflow demand placed on this stage, this is deducted from the incoming massflow rate  $\dot{m}_{\text{in}}$ , with the new massflow rate written to the outflow Station.

The stator solidity  $\sigma_s$  is then computed using Lieblein's criterion (Equation 3.7) and a diffusion factor  $DF = 0.45$ . The stator blade properties are computed next by calling the stator `Blade_aerocalc_comp_Farokhi()` method, along with a typical blade Reynolds number  $Re_{\text{typ}} = 5 \cdot 10^5$  and the kinematic viscosity, calculated according to Equation 3.8.

Using the newly obtained properties of the stator, a new stage length is calculated and the location of the outflow Station is updated.

#### THE GETMASS() METHOD

The `Axcompressorstage_getmass()` method is slightly more involved, as it involves calculating masses of casings and coupling hardware, as well as getting the masses for the `Blade` and `Disk` objects using their own `getmass()` methods.

The contribution for the stage coupling hardware<sup>28</sup>  $m_{\text{hw}}$  is modelled as a 0.075"  $\approx 0.0019$  [m] thick ring located at 75% of the hub radius over the entire length of the stage according to [15, Eq. 8], yielding

$$m_{\text{hw}} = 0.75 \cdot D_{\text{hub}} \cdot \pi \cdot 0.0019 \cdot l_{\text{stage}} \cdot \rho_{\text{mat}} \quad (10.18)$$

Next, the thickness of the casings is determined. Multiple sizing conditions are present here: the casing must be able to deal with the internal pressure, however blade containment in case of a rotor failure is also a requirement [22, 810] for a casing. The thickness required for pressure containment  $t_{\text{pres}}$ <sup>29</sup> is found through the simple boiler equations, using

$$t_{\text{pres}} = \frac{p_{\text{max}} \cdot D_{\text{max}}}{2 \cdot \sigma_y} \quad (10.19)$$

while for the rotorblade containment criterion, the kinetic energy of a loose blade  $E_{\text{kin}}$  needs to be estimated first, before the blade containment thickness  $t_{\text{cont}}$  can be determined using [16, Eq. 3.83], yielding

$$\begin{aligned} E_{\text{kin}} &= \frac{1}{2} \cdot m_{\text{blade}_r} \cdot \left( 2 \cdot \pi \cdot r_m \cdot \frac{\text{RPM}}{60} \right)^2 \\ t_{\text{cont}} &= \frac{0.4 \cdot E_{\text{kin}} \cdot E}{\sigma_y^2 \cdot h_{\text{blade}_r} \cdot c_{\text{blade}_r}} \end{aligned} \quad (10.20)$$

where  $E$  is the casing material Young's modulus.<sup>30</sup>

The stator and rotor `Blade`, rotor `Disc`, casing and hardware masses are then added, with the masses of the IGV and inflow duct also being calculated and added to the total if present.

## 10.5. AXTURBINESTAGE

The `Axturbinestage` class is a model for a single axial turbine stage. It is intended as a component of an `Axialturbine` object, and should not be used separately.

<sup>27</sup>This is assumed to be the case when the required mean outflow radius lies between the hub and tip radius of the preceding rotor mean outflow radius.

<sup>28</sup>These are the nuts, bolts, spacers et cetera connecting e.g. one stage to the next.

<sup>29</sup>A minimum thickness is also imposed on this value, ensuring the resulting thickness always exceeds 1 [mm].

<sup>30</sup>As described in Section 3.1.2, for too large resulting thicknesses, the code switches to a multi-material casing.

### Axturbinestage class diagram

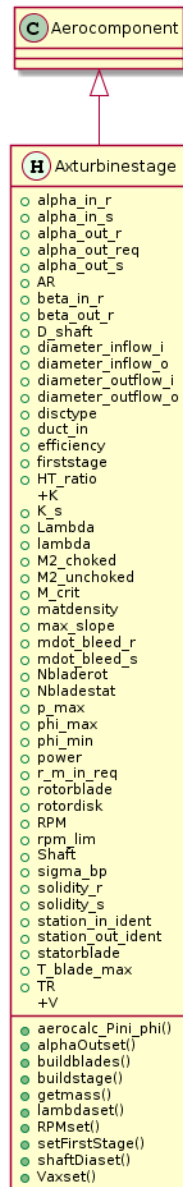


Figure 10.18: Class diagram for the Axturbinestage class.



### 10.5.1. PROPERTIES

Again, as shown in Figure 10.18, the component is a sub-class of the `Aerocomponent` superclass, due to it being a through-flow component, and therefore inherits all of its properties. It however also defines many others of its own.

Most of these properties are descriptive, allowing the use of results by future calculations or by other objects. These include the `alpha_in_r`, `alpha_in_s`, `alpha_out_r` and `alpha_out_s` properties, which represent the rotor and stator in- and outflow absolute angles<sup>31</sup> and `beta_in_r` and `beta_out_r`, which represent the relative rotor in- and outflow angles, all in [rad]. The stage in- and outflow inner and outer diameters (in [m]) are given by the `diameter_inflow_i`, `diameter_inflow_o`, `diameter_outflow_i` and `diameter_outflow_o`, while the `HT_ratio` property defines the stage outflow hub-to-tip ratio. The required bleed massflow rates (in  $\left[\frac{\text{kg}}{\text{s}}\right]$ ) for cooling the rotor and stator are stored in the `mdot_bleed_r` and `mdot_bleed_s` properties, while the rotor and stator blade numbers are given by the `Nbladerot` and `Nbladestat` properties. The maximum total pressure in [Pa] is stored in the `p_max` property. If a certain mean inflow radius is required, this value (in [m]) is stored in the `r_m_in_req` property, while the limit rotational speed in  $\left[\frac{\text{rev}}{\text{min}}\right]$  is stored in the `rpm_lim` property. The rotor and stator solidities are stored in the `solidity_r` and `solidity_s` properties, while the blade pull stress (in [Pa]) is given by the `sigma_bp` property.

Some properties are also used as design or calculation inputs instead, such as the `alpha_out_req` property, which contains the required outflow angle in [rad], if applicable, or the assumed blade aspect ratio `AR`. The type of rotor disk used for the estimation of the rotor disk mass is stored in the `disctype` property, while the turbine polytropic efficiency  $\eta$  is stored in the `efficiency` properties. The `firststage` property is a boolean determining whether the stage is the first stage or not, while the `K` and `K_s` properties contain the blade volume factors for the rotor and stator. The degree of reaction  $\Lambda$  and work coefficient  $\lambda$  of the stage are controlled by the `Lambda` and `lambda` properties. Several Mach numbers are defined as defaults, for a choked nozzle, an unchoked nozzle and a critical Mach number for calculation of the limit rotational velocity as `M2_choked`, `M2_unchoked` and `M_crit`. The density of the stage material is given by the `matdensity` property, while the maximum mean radius slope (in [rad]) over the stator is stored in the `max_slope` property. The `phi_min` and `phi_max` properties contain the boundaries for the allowable range of the flow coefficient  $\phi$ , while the power property contains the to-be-generated power in [W]. The stage rotational velocity in  $\left[\frac{\text{rev}}{\text{min}}\right]$  is given in the `RPM` property, the identifier of the associated `Shaft` object is stored in the `Shaft` property, while the `station_in_ident` and `station_out_ident` contain the in- and outflow station identifiers. The maximum blade temperature in [K] is given by `T_blade_max`, the assumed blade taper ratio is given by the `TR` property and the axial outflow velocity is specified by the property `V` in  $\left[\frac{\text{m}}{\text{s}}\right]$ .

Finally, some properties are also purely used for containing (references to) other objects, including the `statorblade`, `rotorblade` and `rotordisk` properties, which contain the stator and rotor `Blade` and `Disk` objects. The `duct_in` object contains the inflow `Duct` object, if applicable.

### 10.5.2. METHODS

The `Axturbinestage` class also defines or redefines some methods, as again can be seen from Figure 10.18. These include the `Axturbinestage()` constructor and the `buildstage()` and `buildblades()` methods, which are used to build or re-build the `Disk` and `Blade` objects. The `aerocalc_Pini_phi()` method is used for the design and on-design performance calculations, while the mass estimation is done through the `getmass()` method. Finally, the `alphaOutset()`, `lambdaset()`, `RPMset()`, `Vaxset()` and `setFirstStage()` methods are purely used for setting the required outflow angle  $\alpha_3$ , work coefficient  $\lambda$ , stage rotational velocity, outflow axial velocity  $V_{ax}$  and whether the stage is the first stage, and are therefore not worked out further.

#### THE AXTURBINESTAGE() CONSTRUCTOR

The `Axturbinestage()` method is relatively straightforward, first setting the input name, identifiers of the associated `Flow` and `Shaft` objects as well as the in- and outflow `Stations` and the polytropic efficiency  $\eta$  to the appropriate object properties. The stage material is then assumed to be steel, after which the `buildstage()` method is called to further construct the stage.

#### THE BUILDSTAGE() METHOD

The `buildstage()` method starts by setting the rotor and stator blade volume factors  $K = 0.195$  [14] and  $K_s = 0.144$  [15, pg. 27], after which the `buildblades()` method is called to obtain the blade properties. With

<sup>31</sup>For the calculations done here, `alpha_out_s` and `alpha_in_r` are assumed to be equal.

these known, the stator and rotor blade solidities  $\sigma_s$  and  $\sigma_r$  are estimated, based on which the stator and rotor blade numbers are estimated.

Next, the blade pull stress is estimated using Equation 3.28, which then allows for (re-)creation of the rotor Disk object.

#### THE BUILDBLADES () METHOD

The `buildblades ()` method starts by estimating the average stator and rotor hub and tip diameters, assuming a piecewise linear change in diameter. The stator and rotor Blade objects are then (re-)created, after which it is checked whether the rotor and/or stator blades are cooled. If this is the case, the applicable Blade object `cooled` boolean property is set to true, or to false otherwise. Finally, it sums the stator and rotor lengths to estimate the total stage length.

#### THE AEROCALC\_PINI\_PHI () METHOD

The `aerocalc_pini_phi ()` method is used to perform the on-design performance and design calculations of the `Axturbinestage`.

The method starts by finding the relevant in- and outflow Station objects, as well as the stator and rotor bleed supply Stations. The method then checks whether or not the stage is the first stage, based on which the method for determining the axial and rotational velocities is chosen.

If the stage is the first stage, the nozzle outflow Mach number  $M_2$  is assumed, after which the local static temperature  $T_2$ , sonic velocity  $a_2$ , absolute velocity  $V_2$  and axial velocity  $V_{ax}$  are computed according to Equation 3.22. The flow coefficient  $\phi = \frac{V_{ax}}{U}$  is then computed and compared with the given limit values. If it is found to exceed the allowed range, the axial velocity  $V_{ax}$  is recomputed according to Equation 3.23 with the flow coefficient set to the violated limit, from which also an updated value for  $U$  follows, which is then translated into a required inflow mean radius  $r_{mreq}$ .

If however the stage is not the first stage, the axial velocity is assumed to be equal to the stage inflow axial velocity  $V_{ax} = V_{axin}$ . Again, the flow coefficient  $\phi$  is computed and checked, however, if found out of bounds, it is now purely the rotational velocity  $U$  which is recomputed for the violated limit  $\phi$  from its definition  $\phi = \frac{V_{ax}}{U}$ , while the axial velocity  $V_{ax}$  is kept constant.

Next, the rotor whirl velocity change  $\Delta C_w$  is determined from the rotor rotational velocity  $U$  and the work coefficient definition  $\lambda = \frac{\Delta C_w}{U}$ . If the stage outflow angle  $\alpha_3$  is left free, the degree of reaction is assumed to be  $\Lambda = 0.5$  and the pre- and post-rotor absolute whirl velocity components  $C_{w2}$  and  $C_{w3}$  are computed according to Equations 3.20 and 3.21.

If however the outflow angle  $\alpha_3$  is prescribed,  $C_{w3}$  is already fully defined through  $\alpha_3$  and the assumption of constant axial velocity, therefore leading to a calculation of  $\Lambda$  and  $C_{w2}$  using rewritten versions of these equations, yielding

$$\begin{aligned}\Lambda &= 1 - \frac{C_{w3}}{U} - \frac{\lambda}{2} \\ C_{w2} &= U \cdot \left(1 - \Lambda + \frac{\lambda}{2}\right)\end{aligned}\quad (10.21)$$

However, recomputing  $\Lambda$  this way may lead to a too low degree of reaction (e.g.  $\Lambda < 0$ ), which needs to be checked. If this is found to be the case, the degree of reaction is set to  $\Lambda = 0$ , with the work coefficient  $\lambda$  and the post-stator whirl velocity  $C_{w2}$  being recalculated<sup>32</sup> according to

$$\begin{aligned}\lambda &= 2 \cdot \left(1 - \frac{C_{w3}}{U} - \Lambda\right) \\ C_{w2} &= U \cdot \left(1 - \Lambda + \frac{\lambda}{2}\right)\end{aligned}\quad (10.22)$$

After all absolute whirl velocity components have been determined this way, the relative whirl velocities and absolute and relative velocities and angles are calculated. The stage outflow hub radius is then estimated. If this is negative or smaller than the shaft radius, an updated mean radius estimate is made based on the found outflow area and minimum required hub radius, after which the calculation is re-iterated for the same axial velocity until a sufficient hub radius is obtained. If the found required axial velocity and/or the required mean radius are not equal to the inflow conditions, an inflow correction duct is added to adjust these.

Next, the impact of the stator cooling bleed air is evaluated using the pre-existing `flowMix ()` function, yielding a new post-stator total temperature, gas composition and massflow rate  $\dot{m}$ . Next, the post-stator

<sup>32</sup>Even though  $\lambda$  is the control variable and should therefore not be tweaked during the calculation, it can be seen that for a specified outflow angle of  $\alpha_3 = 0^\circ$ , the only remaining variable in Equation 10.21 which has any impact on  $\Lambda$  is  $\lambda$ .

static properties are calculated, along with the annulus area and hub and tip radii. If the stator inflow total temperature exceeds the maximum blade temperature, as set by the stage `T_blade_max` property, the required amount of cooling bleed flow (to be used in the next iteration) is computed using the `pre-existing coolingFlow2()` function. This value is then set to the appropriate bleed flow `Station`.

The stator blade Zweifel coefficient is then assumed as  $\Psi_s = 1$ , as well as a throat typical Reynolds number  $Re_{typ} = 5 \cdot 10^5$ , and a kinematic viscosity  $\nu_{throat}$  is estimated. These properties are then used to calculate the stator blade properties and solidity  $\sigma_s$  using the `Blade aerocalc_turb_Farokhi()` method. This solidity is then used to estimate the number of stator blades according to Equation 3.27.

With the stator now calculated, the decrease in total temperature over the rotor due to the power extraction is calculated from the change in whirl velocity  $C_{w3} - C_{w2}$  over the rotor, according to

$$T_{t_{uncooled}} = T_{ts} + \frac{U}{c_p} \cdot (C_{w3} - C_{w2}) \quad (10.23)$$

After which the post-rotor total pressure is computed using

$$p_{tr} = p_{ts} \cdot \left( \frac{T_{t_{uncooled}}}{T_{ts}} \right)^{\frac{\eta \cdot \gamma}{\gamma - 1}} \quad (10.24)$$

Using the thus-obtained uncooled post-rotor total temperature, along with the bleed air massflow and properties, the cooled post-rotor total temperature  $T_{tr}$ , massflow  $\dot{m}_r$  and gas composition are now computed, again using the `flowMix()` function.

The rotor blade Zweifel coefficient  $\Psi_r = 1$  and typical Reynolds number  $Re_{typ} = 5 \cdot 10^5$  are now assumed and the rotor throat kinematic viscosity  $\nu_t$  is estimated, after which the rotor solidity  $\sigma_r$  and blade number are estimated analogous to those of the stator. Again, if the rotor inflow total temperature is found to exceed the maximum allowable blade temperature, the required bleed air cooling massflow is estimated using the `coolingFlow2()` function, for use in the next iteration.

With the stator and rotor properties now determined, the stage power production  $P_{gen}$  is estimated according to<sup>33</sup>

$$P_{gen} = \dot{m}_s \cdot c_p \cdot (T_{ts} - T_{t_{uncooled}}) \quad (10.25)$$

The rotor hub-to-tip-ratio  $\left(\frac{H}{T}\right)_{rot}$  is now re-computed, after which the blade pull stress  $\sigma_{bp}$  is estimated, according to [16, Eq. 3.58]

$$\sigma_{bp} = N_{blade_r} \cdot m_{blade_r} \cdot \frac{r_m \cdot (2 \cdot \pi \cdot \frac{RPM}{60})^2}{2 \cdot \pi \cdot r_{hub} \cdot c_r} \quad (10.26)$$

With this value known, the rotor `Disk` is now created and calculated and the stage length estimate is updated.

Finally, the stage  $RPM_{lim}$  is determined based on a maximum allowed rotor inflow Mach number (default value  $M_{max} = 0.75$  [17, p. 327]), from which a maximum flow velocity  $V_{2_{max}}$  is computed. This in turn is used with the axial flow velocity  $V_{ax}$  (assumed unchanged) to find a maximum relative rotor inflow angle  $\beta_{2_{lim}}$ , which is combined with the actual outflow angle  $\alpha_3$  and required power extraction (in the form of the uncooled total temperature jump) to yield the limit rotor velocity  $\omega_{lim}$  and  $RPM_{lim}$  using

$$\begin{aligned} \omega_{lim} &= \frac{(T_{ts} - T_{t_{uncooled}}) \cdot c_p}{2 \cdot r_m} - \frac{V_{ax} \cdot \tan \alpha_3}{r_m} - \tan \beta_{2_{lim}} \cdot V_{ax} \\ RPM_{lim} &= \frac{\omega_{lim}}{2 \cdot \pi} \cdot 60 \end{aligned} \quad (10.27)$$

With this calculation, the method ends.

#### THE GETMASS() METHOD

The `getmass()` method of the `Axturbinestage` object is a little more involved, as the turbine is assumed to consist of several different components, including stator and rotor blades and a rotor disk, but also contributions for coupling hardware, casings and possibly ducts and/or IGV's.

The method starts by calling `getmass()` methods of the rotor and stator `Blade` and the rotor `Disk`, in order to establish their masses, however the results are not directly added. The contribution for the stage coupling hardware<sup>34</sup>  $m_{hw}$  is modelled according to [15, Eq. 8] as a  $0.075'' \approx 0.0019$  [m] thick ring located at 75% of the hub radius over the entire length of the stage, yielding

$$m_{hw} = 0.75 \cdot D_{hub} \cdot \pi \cdot 0.0019 \cdot l_{stage} \cdot \rho_{mat} \quad (10.28)$$

<sup>33</sup>The used massflow rate is the post-stator cooled massflow rate, as it is assumed that the rotor cooling massflow does not take part in the power generation.

<sup>34</sup>This includes the nuts, bolts, spacers et cetera connecting one stage to the next.

Next, the thickness of the casings is determined. Multiple sizing conditions are present here: the casing must be able to deal with the internal pressure, however blade containment in case of a rotor failure is also a requirement [22, 810] for a casing. The thickness required for pressure containment  $t_{\text{pres}}$ <sup>35</sup> is found through the simple boiler equations, using

$$t_{\text{pres}} = \frac{p_{\text{max}} \cdot D_{\text{max}}}{2 \cdot \sigma_y} \quad (10.29)$$

while for the rotorblade containment criterion, the kinetic energy of a loose blade  $E_{\text{kin}}$  needs to be estimated first, before the blade containment thickness  $t_{\text{cont}}$  can be determined using [16, Eq. 3.83], yielding

$$\begin{aligned} E_{\text{kin}} &= \frac{1}{2} \cdot m_{\text{blade}_r} \cdot \left( 2 \cdot \pi \cdot r_m \cdot \frac{\text{RPM}}{60} \right)^2 \\ t_{\text{cont}} &= \frac{0.4 \cdot E_{\text{kin}} \cdot E}{\sigma_y^2 \cdot h_{\text{blade}_r} \cdot c_{\text{blade}_r}} \end{aligned} \quad (10.30)$$

where  $E$  is the casing material Young's modulus. As the blade kinetic energy can be quite high, which in turn may lead to ridiculous wall thickness requirements, the method checks whether the calculated thickness with the regular turbine material yields an acceptable thickness.<sup>36</sup> If this is not the case, a multi-material casing is assumed, with the turbine material casing being sized purely for the pressure requirement, while the blade containment criterion is met by a kevlar overwrap.<sup>37</sup>

The total casing mass is determined by assuming a casing with pressure thickness around the stator, and the most critical calculated thickness (or thicknesses) around the rotor, and adding the two mass contributions. Then, by adding this casing mass, the coupling hardware mass contribution, the rotor Disk mass and the rotor and Blade masses, multiplied by their respective blade numbers, the total stage mass estimate is obtained. If an inflow Duct is present within the stage, its `getmass()` method is called as well and the resulting mass added to the total stage mass estimate. Finally, returning the obtained stage mass estimate ends the method.

## 10.6. FAN\_MIME

The `Fan_MIME` class models a fan stage, which splits its outflow into 2 different flows, yielding a core and bypass flow.

### 10.6.1. PROPERTIES

As can be seen in Figure 10.19, the `Fan` class contains many properties besides those inherited from its `Aerocomponent` superclass. In terms of controlling properties, the most obvious of these is the bypass ratio BPR, which is the ratio of the bypass to the core massflow rates  $\frac{\dot{m}_{\text{bp}}}{\dot{m}_{\text{c}}}$ . Another controlling property is the inflow hub-to-tip ratio `HT_ratio`. The property `V` contains the required axial outflow velocity component (in  $\left[\frac{\text{m}}{\text{s}}\right]$ ), while the core and bypass pressure ratios are stored in the `PRC` and `PRD` properties. The fan efficiency  $\eta$  is contained in the `efficiency` property, while the `dimrel` and `dimrel_fac` properties are used for setting the controlled radius and its corresponding ratio over the fan. Finally, the required outflow Mach number and the core and bypass absolute outflow angles are stored in the `M_out`, `core_outflow_alpha` and `bypass_outflow_alpha` properties.

Several more or less standard properties for through-flow rotating components include the shaft RPM, the shaft identifier `Shaft`, the inflow station identifier `station_in`, as well as those for the core and bypass outflow stations `station_out` and `station_out_bp`. As a second `Flow` object is required to model the bypass outflow, this flow identifier is stored in the `Flow_bp` property.

Some properties also contain settings which are usually left unchanged, such as the rotor disc type `disctype` (set by default to `compFan`). Other properties in this category are the rotor and stator assumed blade aspect ratios `AR` and `AR_s` (default to `AR = 2.5` and `AR_s = 3.5` [16, pg. 72]), the blade volume factor `K = 0.055` [15], blade taper ratio `TR = 0.55` [17, pg. 346], allowable tip Mach number `M_limit = 1.5` [17, pg. 195] and the limit deHaller number `deHallerCrit = 0.69` [17, p. 197].

Many design outputs are also stored as properties, including the maximum allowable spool speed `rpm_lim` in  $\left[\frac{\text{rev}}{\text{min}}\right]$ , the fan power consumption power in [W], the various in- and outflow hub and tip diameters, the solidities `solidity_s_c`, `solidity_s_bp` and `solidity_r` and the blade numbers `Nbladestat_c`, `Nbladestat_bp` and `Nbladerot`. The absolute and relative in- and outflow angles of rotor and stators are

<sup>35</sup>A minimum thickness is also imposed on this value, ensuring the resulting thickness always exceeds 1 [mm].

<sup>36</sup>Currently, this critical value is set at 5 [cm].

<sup>37</sup>This approach is taken from [16, Sec. 3.3.6.4].

### Fan\_MIME class diagram

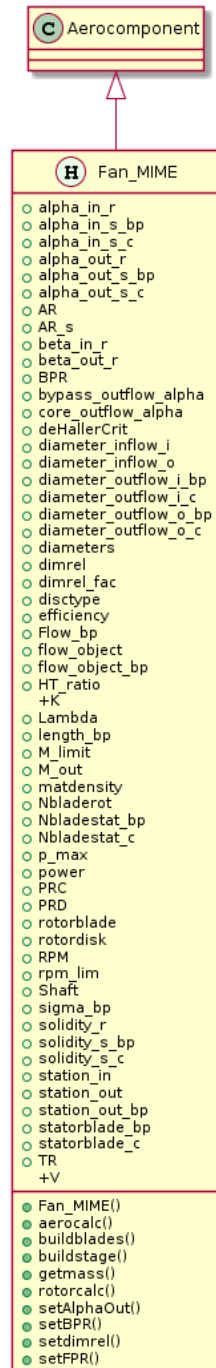


Figure 10.19: Class diagram for the Fan\_MIME class.

stored in [rad] as `alpha_in_s_c`, `alpha_out_s_c`, `alpha_in_s_bp`, `alpha_out_s_bp`, `alpha_in_r`, `alpha_out_r`, `beta_in_r` and `beta_out_r`. The blade pull stress  $\sigma_{bp}$  due to the rotor blades is stored as `sigma_bp`, while the degree of reaction is stored as `Lambda`, the length of the bypass path (rotor and bypass stator row lengths summed in [m]) and the maximum encountered total pressure in [Pa] is stored as `p_max`.

Finally, some container properties are present, including the `rotordisk`, `rotorblade`, `statorblade_c` and `statorblade_bp` properties, which contain the `Disk` and `Blade` object modelling these parts. Also, for ease of reference, the core and bypass `Flow` objects (or more precisely the handles of these objects) are also stored in the `flow_object` and `flow_object_bp` properties.

### 10.6.2. METHODS

The `Fan` class defines several methods. Besides its constructor `Fan()`, these are the `buildstage()` and `buildblades()` methods, the `getmass()` mass estimation method, the `aerocalc()` and `rotorcalc()` on-design performance and design calculation methods and the `setdimrel()` and `setAlphaOut()` setting update methods. The latter are single-action methods and thus not elaborated upon further. The other methods are discussed in more detail below.

#### THE FAN() CONSTRUCTOR METHOD

The `Fan()` constructor takes the fan name, identifiers of the core and bypass `Flow` objects, the `Shaft`, inflow and core and bypass outflow `Stations`, the core and bypass pressure ratios, the bypass ratio, efficiency, fan material and outflow Mach number as inputs. Optionally, it can also be specified which dimension is controlled over the fan (hub, tip or mean radius), the ratio between the in- and outflow values of this dimension, as well as the desired outflow angles for the core and bypass flows in [rad]. If these are not specified, they default to the tip radius being kept constant (a ratio of 1) and  $0^\circ$  outflow angles for both the core and bypass flows. All of these values are then set to the `Fan` properties, after which the `buildstage()` method is called to convert these properties into something resembling a fan.

#### THE BUILDSTAGE() METHOD

The `buildstage()` method requires only the object it is called upon as input. It starts by calling the `buildblades()` method to create the rotor and stator `Blade` objects. It then checks whether the rotor solidity  $\sigma_r$  has a positive non-zero value, for which case the solidity  $\sigma_r$ , rotor blade chord  $c_r$  and mean radius  $r_{mean}$  are used to find the number of rotor blades  $N_{blade}$ . If the solidity is not (yet) known, is negative or 0, the number of blades is estimated as the minimum number of 2 blade required for a balanced rotor.

The rotor hub-to-tip ratio is then found, based on which the blade pull stresses of the rotor blades on the rotor disk  $\sigma_{bp}$  are estimated using [15, Eq. 7]. This in turn is then used to create the rotor `Disk`.

Finally, the core and bypass stator solidities are checked and their respective blade numbers calculated similar to those of the rotor.

#### THE BUILDBLADES() METHOD

The `buildblades()` method requires only the object it is called upon as input. It first checks whether the diameters of the fan are known, and otherwise assumes some defaults, based on which the rotor and stator `Blade` objects are constructed. Based on their properties, the core and bypass fan lengths are then estimated by summing the length of the rotor row and that of the appropriate stator row.

#### THE AEROCALC() METHOD

The inputs of the `aerocalc()` method are, besides the object it is called upon, the (handles to the) core and bypass `Flow` as well as the `Shaft` objects. From these `Flows`, the in- and outflow `Stations` are retrieved first. With these found, the `rotorcalc()` method is called to perform the calculations over the rotor. From the thus-obtained post-rotor properties, the post-stator properties for both the core and bypass flows are then obtained, leading to the absolute outflow angles and velocities, which are then written to the outflow stations. The outflow mean radii are then computed, based on the assumption that the core hub and bypass tip radii remain constant after the rotor. The results are again written to the outflow stations.

Next, based on an assumed hub diffusion factor  $DF_{hub} = 0.6$  [16, pg. 57], the solidities of the rotor and stators are estimated through the use of [16, Eq. 3.39], which yields the pitch-to-chord ratio, or the inverse of the solidity.

The limit spool speed is then estimated based on a maximum tip Mach number. The axial velocity is vectorially subtracted from the corresponding limit tip velocity, the result of which is translated through the rotor tip diameter to a maximum allowable spool speed.



The `buildstage()` method is then called to translate the found properties into properly sized `Blade` and `Disk` objects. The found power and spool speed limit are written to the shaft, after which the outflow `Stations` are assigned updated values for their  $x$ -location based on the fan core and bypass lengths and the  $x$ -location of the inflow `Station`. Finally, the maximum total pressure within the fan is set to the `p_max` property.

#### THE ROTORCALC() METHOD

The `rotorcalc()` method needs the object it is called upon, as well as the inflow `Station` and `Shaft` objects, as inputs. It starts by calculating the post-rotor total properties from the pressure ratios and the derived temperature ratios. The massflow division is then computed based on the bypass ratio, followed by the power required corresponding to the core and bypass total temperature increases. Based on the massflow division and a uniform inflow assumption, an inflow “division” radius can also be computed, allowing for the estimation of the inflow mean radii of both the core and bypass flows.

Next, depending on which radius is controlled, the method enters a design loop, first determining the specified outflow radius. The post-rotor static properties and area are then estimated using [16, Eq. 3.22, 3.24, 3.7] for an estimate of the mean radius, after which this mean radius estimate is updated, until this has converged. This is done outside-in for a controlled tip diameter, and inside-out for a controlled hub diameter.

The resulting radii, total properties and massflows are then returned by the method.

#### THE GETMASS() METHOD

The `getmass()` method requires only the object it is called upon as input. It first defines several values, such as a maximum single-material thickness  $t_{\text{switch}}$ , a minimum material thickness  $t_{\text{min}}$  and the mechanical properties  $E$ ,  $\sigma_y$  and  $\rho$  of Kevlar.

The coupling hardware mass contribution  $m_{\text{hw}}$  is then estimated according to Equation 10.18. [15, Eq. 8] Likewise, the pressure and blade containment material thicknesses and masses of the casings are also computed analogous to those for the `Axcompressorstage` as described in Section 10.4.2, with the only difference being the presence of two stator casings instead of just one.

One other additional mass contribution is also taken into account for the Fan, which is the nosecone or spinner. This is assumed to be a straight cone with a maximum length of twice the rotor blade row length, which is made of 5 [mm] thick fiberglass. Finally, all of these mass contributions are summed to obtain the mass estimate of the entire Fan.

## 10.7. INLET

The `Inlet` class is a model of a pitot inlet for an air-breathing engine.

**Inlet class diagram**

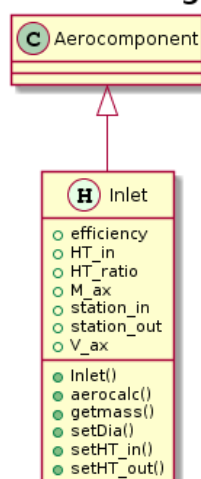


Figure 10.20: Class diagram for the `Inlet` class.

### 10.7.1. PROPERTIES

The `Inlet` class is, as indicated in Figure 10.20, a subclass of the `Aerocomponent` (and thereby of the `Component`) class, and has relatively few non-inherited properties due to the model simplicity. Almost all of these properties can be considered design properties, as they include the efficiency of the inlet, the in- and outflow hub-to-tip ratios `HT_in` and `HT_ratio`, the design axial outflow Mach number `M_ax` and the in- and outflow station identifiers `station_in` and `station_out`. The `diameter` property (inherited from the `Component` superclass) is given an additional meaning, as it is used as a setting variable defining the inlet outflow outer diameter.<sup>38</sup> Finally, the `V_ax` property is a more descriptive property, as it contains the axial outflow velocity at design condition.

### 10.7.2. METHODS

Besides the `Inlet()` constructor method, the class (re-)defines several other methods. These include the aerodynamic performance estimation method `aerocalc()`, the mass estimation method `getmass()` and the in- and outflow hub-to-tip-ratio modification functions `setHT_in()` and `setHT_out()`. While the first three methods will each be covered below, the last two will not be, as they purely exist to allow (re-)setting of these variables.

#### THE INLET() CONSTRUCTOR METHOD

The constructor method of the `Inlet` component is very simple. It simply assigns all the given inputs (the object name, associated `Flow` identifier, efficiency, outflow diameter, axial Mach number and in- and outflow `Station` identifiers) to the properties of the newly created object.

#### THE AEROCALC() METHOD

The `aerocalc()` method determines the on-design post-inlet properties, as well as the inlet design, according to the methods laid out in Section 3.6.1.

First, the appropriate in- and outflow `Station` objects are obtained from the associated `Flow` object. As no energy is assumed to be added to the flow, the total temperature is assumed to remain constant over the inlet ( $T_{t_{out}} = T_{t_{in}}$ ), with any losses due to a non-100% efficiency being accounted for in the outflow total pressure  $p_{t_{out}}$  according to

$$p_{t_{out}} = p_{in} \cdot \left( 1 + \frac{\eta \cdot V_{ax_{in}}^2}{2 \cdot c_p \cdot T_{in}} \right)^{\frac{\gamma}{\gamma-1}} \quad (10.31)$$

The outflow static temperature  $T_{out}$  is calculated from the outflow total temperature  $T_{t_{out}}$  and the specified axial outflow Mach number  $M_{ax_{out}}$  using the standard isentropic relations, after which the axial outflow velocity  $V_{ax_{out}}$  is estimated by multiplying  $M_{ax_{out}}$  and the outflow sonic velocity  $a_{out} = \sqrt{\gamma \cdot R \cdot T_{out}}$ . Finally, the outflow static pressure  $p_{out}$  is computed from the total-to-static temperature ratio and the outflow total pressure  $p_{t_{out}}$ .

All of these parameters are then set to the outflow `Station` object (which is created if not yet in existence). The outflow area  $A_{out}$  is now calculated based on the specified outflow outer diameter  $D_{tip_{out}}$  and hub-to-tip ratio  $\left(\frac{H}{T}\right)_{out}$  according to

$$A_{out} = \pi \cdot \left( \left( \frac{D_{tip_{out}}}{2} \right)^2 - \left( \left( \frac{H}{T} \right)_{out} \cdot \frac{D_{tip_{out}}}{2} \right)^2 \right) \quad (10.32)$$

With the mean outflow radius  $r_{m_{out}}$  by definition then being given by

$$r_{m_{out}} = D_{tip_{out}} \cdot \frac{1 + \left(\frac{H}{T}\right)_{out}}{4} \quad (10.33)$$

The intake massflow  $\dot{m}$  is then calculated by multiplying the outflow static density  $\rho_{out}$  with the outflow area  $A_{out}$  and the axial outflow velocity  $V_{ax_{out}}$ , after which all of these parameters are then used to update the outflow `Station`, with the calculated massflow also being written to the inflow `Station`. Finally, the inflow mean radius is also calculated and written to the the inflow `Station` according to

$$r_{m_{in}} = \frac{\sqrt{\frac{A_{in}}{\pi}}}{2 \cdot \sqrt{1 - \left(\frac{H}{T}\right)_{in}^2}} \cdot \left( 1 + \left(\frac{H}{T}\right)_{in} \right) \quad (10.34)$$

This ends the method.

<sup>38</sup>For a turbofan engine, this is equal to the fan face diameter.



### THE GETMASS () METHOD

While the `getmass()` method of the `inlet` does exist, it is set to return a mass of 0 [kg], as estimating the inlet mass is heavily dependent on many more design decisions and constraints than are included within the MIME code at this time. Therefore, estimating inlet mass is something which is deemed outside the scope of the MIME code for now.

## 10.8. COMBUSTOR

Combustor class diagram

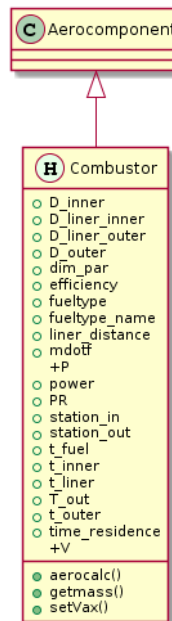


Figure 10.21: Class diagram for the Combustor class.

The Combustor class serves as a rough model of an annular combustion chamber. Its thermodynamic calculations are mostly taken from the current Initiator engine implementation.

### 10.8.1. PROPERTIES

As can be seen from Figure 10.21, the Combustor class is a subclass of the Aerocomponent class. It however also defines several properties which are not inherited from this superclass. These properties may be outputs or design inputs. Properties from the latter category include the `time_residence` property, which defines the time in [s] a particle would spend in the combustor when moving through it at the average of the in- and outflow axial velocities, an optional specified outflow axial velocity  $V$  in  $\left[\frac{m}{s}\right]$ , the outflow total temperature  $T_{out}$ , the total pressure ratio over the combustor  $PR$  and the combustor `efficiency`. The selection of the fuel type is stored as an integer (1, 2 or 3) in the `fueltype` property. As for most through-flow components, also the in- and outflow Station identifiers are stored, which is done in the `station_in` and `station_out` properties.

Output properties include the various diameters  $D_{inner}$ ,  $D_{outer}$ ,  $D_{liner\_inner}$ ,  $D_{liner\_outer}$  and thicknesses  $t_{inner}$  and  $t_{outer}$ . The combustor maximum total pressure in [Pa] is stored as  $P$ , while the found fuel massflow rate in  $\left[\frac{kg}{s}\right]$  is stored as `mdotf`. The generated power is stored as `power`.

Several properties also contain scarcely-changed settings, such as the liner thickness  $t_{liner} = 0.055'' \approx 1.4$  [mm] [15] and the liner spacing `liner_distance`, set to 20% of the burner passage height [15]. Another such setting is the fuel temperature, which defaults to  $t_{fuel} = 288.15$  [K]. Finally, the `dim_par` string property sets which criterion is used for sizing, and is set by default to `M05`, indicating an outflow Mach number of  $M_{out} = 0.5$  is used for sizing.

### 10.8.2. METHODS

The `Combustor` class defines four methods. Besides its constructor `Combustor()`, these include the on-design performance and design method `aerocalc()`, the mass estimation method `getmass()` and the `setVax()` method, which is used to update the required axial outflow velocity setting. As the latter only performs this one function, it is not elaborated upon further here.

#### THE COMBUSTOR() CONSTRUCTOR METHOD

The `Combustor()` constructor requires the name, Flow identifier, outflow total temperature  $T_{t4}$ , residence time parameter, fuel type selector, in- and outflow `Station` identifiers, total pressure ratio PR and combustor efficiency  $\eta$  as inputs. It first sets these parameters to the appropriate properties, after which it assumes the combustor material is steel, for which the properties are retrieved using the `Component` class's `getdensity()` method. For ease of reference, the `fueltype_name` property is also set to correspond to the selected `fueltype`.

#### THE AEROCALC() METHOD

For the `aerocalc()` method, several constants are first defined based on the value of `fueltype`. Next, after the inflow `Station` is found, a convergence loop is started to obtain the desired outflow total temperature and fuel massflow rate required, as the thermodynamic gas properties change with the gas composition, which changes due to the addition of combustion products. Therefore, convergence is assumed for a change of  $< 0.1\%$  of all the gas composition components.

Next, based on the selected sizing criterion, the outflow static properties are computed. Several options are available, including a fixed outflow Mach number of  $M_{out} = 0.5$  or  $M_{out} = 1$ , a constant flow area, constant axial velocity or a set axial outflow velocity. For the fixed outflow Mach number, the static temperature and pressure are computed first using the isentropic relations, after which the outflow velocity is computed from the local speed of sound. Combustor length is then estimated based on the residence time and the average in- and outflow velocities, after which all properties are written to the outflow `Station`. Finally, it is checked that the outflow hub radius is still positive and non-zero. If this is found to no longer be the case, a constant hub radius is assumed instead of a constant hub radius, allowing for the calculation of a new mean outflow radius, which is then assigned instead.

For the constant area assumption, the function

$$\begin{aligned}
 0 &= \rho_{out} \cdot V_{axout} \cdot A_{out} - \dot{m}_{out} \\
 \rho_{out} &= \frac{p_{out}}{R \cdot T_{out}} \\
 p_{out} &= \frac{p_{t_{out}}}{\left(1 + \frac{\gamma-1}{2} \cdot M^2\right)^{\frac{\gamma}{\gamma-1}}} \\
 T_{out} &= T_{t_{out}} - \frac{V_{axout}^2}{2 \cdot c_p} \\
 M &= \frac{V_{axout}}{\sqrt{\gamma \cdot R \cdot T_{out}}}
 \end{aligned} \tag{10.35}$$

is solved numerically to find an estimate for the axial outflow velocity  $V_{ax}$ . With the length estimated similar to that for the specified outflow Mach case, the properties are then written to the outflow `Station`.

For both the constant and specified axial velocity cases, the new outflow axial velocity is simply written to the outflow `Station` along with the total properties, from which the internal calculations of the `Station` object later find the static properties. Both the combustor length estimate as well as the handling of negative or zero outflow hub radii are done analogous to those for the specified outflow Mach cases.

Finally, combustor power is estimated, and the casing and liner diameters are computed, concluding the `aerocalc()` method.

#### THE GETMASS() METHOD

The `Combustor getmass()` method functions similar to that of the `Duct` class, first computing an (outer) wall thickness based on the maximum encountered total pressure through the boiler equations. The inner wall is assumed to have this same thickness. By multiplying this thickness with the surface area of the inner and outer casings and the material density, the estimate for the casing masses is found.

However, besides the `Duct`-like casings, the combustor also has several other mass contributions. These include the masses of the liners, which are computed similar to the casing masses, however now using the standard liner thickness (default  $t_{liner} \approx 1.4$  [mm]). Also, an empirical mass contribution for the fuel domes, nozzles et cetera is taken into account according to [15, Eq. 20].

Summing these 5 contributions of inner and outer casing, inner and outer liner and other hardware yields the mass estimate for the combustor.

## 10.9. NOZZLE

Nozzle class diagram

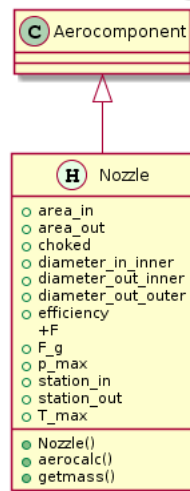


Figure 10.22: Class diagram for the `Nozzle` class.

The `Nozzle` class models a simple annular nozzle, with an assumed constant hub radius.

### 10.9.1. PROPERTIES

Like virtually all throughflow-component-modelling objects within MIME, the `Nozzle` class is a subclass of the `Aerocomponent` and `Component` classes, as can be seen in Figure 10.22. Therefore, most basic properties are inherited from these classes, but some additional properties are added in the `Nozzle` class definition.

While most of these properties are descriptive in nature, there are three which are used to provide some calculation inputs. These are the in- and outflow station identifiers, stored in the `station_in` and `station_out` properties, as well as the nozzle efficiency  $\eta$ , which is stored in the `efficiency` property.

Descriptive properties include the nozzle net and gross thrusts, stored in [N] in the `F` and `F_g` properties, whether or not the nozzle is choked as a boolean property `choked` and the maximum encountered total temperature and pressure properties `p_max` and `T_max` in [Pa] and [K]. Finally, the various diameters and flow areas are stored in `diameter_in_inner`, `diameter_out_inner`, `diameter_out_outer`, `area_in` and `area_out`.

### 10.9.2. METHODS

The `Nozzle` class defines three methods: its constructor `Nozzle()`, the design and on-design performance calculation method `aerocalc()` and the mass estimation method `getmass()`. Each of these is worked out individually below.

#### THE NOZZLE() CONSTRUCTOR METHOD

The `Nozzle()` method is used to create a new `Nozzle`-class object, and requires the new object name, the identifiers of the relevant `Flow` and in- and outflow `Station` objects and the nozzle efficiency. It sets these values to the appropriate properties, and defaults the nozzle length to  $l_{\text{nozzle}} = 0.5$  [m]. This concludes the method.

#### THE AEROCALC() METHOD

The `aerocalc()` method is used to obtain the design of the nozzle, as well as its on-design performance and post-nozzle conditions. In order to do this, it required the `Nozzle` object, the `Flow` object describing the flow through the nozzle and the `Flow` object containing the ambient conditions `Station` as its inputs.<sup>39</sup>

The method starts by searching for the in- and outflow `Station` objects in the `Flow` describing the flow through the nozzle, as well as the ambient conditions `Station` in the appropriate `Flow`. If no outflow `Station` was found, this is created by cloning the inflow `Station`, followed by renaming and renumbering.

<sup>39</sup>It may be the case that the `Flow` object containing the `Station` with the ambient conditions is the same as the one describing the flow through the nozzle. In that case, both inputs should be provided with the same object.

Next, the actual and critical pressure ratios  $PR_{\text{act}}$  and  $PR_{\text{crit}}$  are determined according to

$$\begin{aligned} PR_{\text{act}} &= \frac{p_{\text{tin}}}{p_{\text{amb}}} \\ PR_{\text{crit}} &= \frac{1}{\left(1 - \frac{\gamma-1}{\eta \cdot (\gamma+1)}\right)^{\frac{\gamma}{\gamma-1}}} \end{aligned} \quad (10.36)$$

These values are then compared to determine whether or not the nozzle is choked. This is assumed to be the case if  $PR_{\text{act}} > PR_{\text{crit}}$ . For a choked nozzle, the static outflow temperature and pressure are assumed to be equal to the critical outflow temperature and pressure  $T_{\text{crit}}$  and  $p_{\text{crit}}$ , which are calculated through

$$\begin{aligned} T_{\text{crit}} &= T_{\text{tin}} \cdot \frac{2}{\gamma+1} \\ p_{\text{crit}} &= \frac{p_{\text{tin}}}{PR_{\text{crit}}} \end{aligned} \quad (10.37)$$

Through the ideal gas law, this then allows for estimation of the outflow static density  $\rho$ . The axial outflow velocity is calculated by calculating the sonic velocity  $a$  at the throat, which is equal to the axial outflow velocity for a choked nozzle by definition, according to

$$a = \sqrt{\gamma \cdot R \cdot T_{\text{crit}}} \quad (10.38)$$

Based on this velocity, the outflow static density and the massflow  $\dot{m}$ , the outflow area  $A_{\text{out}}$  is then calculated according to

$$A_{\text{out}} = \frac{\dot{m}}{\rho \cdot a} \quad (10.39)$$

This then allows for the calculation of the net and gross thrust values  $F$  and  $F_g$  according to

$$\begin{aligned} F &= \dot{m} \cdot (a - V_{\text{amb}}) + A_{\text{out}} \cdot (p_{\text{crit}} - p_{\text{amb}}) \\ F_g &= \dot{m} \cdot a + A_{\text{out}} \cdot (p_{\text{crit}} - p_{\text{amb}}) \end{aligned} \quad (10.40)$$

The outflow Station is then updated with these static properties and the axial outflow velocity.

If however the nozzle is unchoked, the total-to-static temperature drop associated with full expansion to ambient pressure is estimated first according to

$$T_{\text{tin}} - T_{\text{out}} = \Delta T = \eta \cdot T_{\text{tin}} \cdot \left(1 - \left(\frac{1}{PR_{\text{act}}}\right)^{\frac{\gamma-1}{\gamma}}\right) \quad (10.41)$$

This is then converted to an axial outflow velocity according to

$$V_{\text{axout}} = \sqrt{2 \cdot c_p \cdot \Delta T} \quad (10.42)$$

Due to the full expansion, the pressure terms are now absent, so the net and gross thrust values  $F$  and  $F_g$  are now calculated according to

$$\begin{aligned} F &= \dot{m} \cdot (a - V_{\text{amb}}) \\ F_g &= \dot{m} \cdot a \end{aligned} \quad (10.43)$$

Again, the outflow Station is updated with the static properties and axial velocity.

If however the pressure ratio over the nozzle is found to be  $PR_{\text{ax}} \leq 1$ , it follows that the nozzle would have to perform compression, which is unlikely. Therefore, in this case, the nozzle is assumed to be absent and given a length of 0. Still, a thrust calculation is performed, for which the same equations are used as for the case of the choked nozzle, however as the ambient pressure now exceeds the nozzle outflow static pressure, this net backpressure will now detract from the calculated thrust values.

Finally, the outflow mean radius is updated to such a value to ensure the outflow inner diameter is equal to the inflow inner diameter, and the  $x$ -location of the outflow Station is updated.

#### THE GETMASS() METHOD

The `getmass()` method is used to estimate the nozzle mass contribution. It assumed the nozzle as consisting of a thin-walled conical outer (as well as optionally an inner) casing, with a thickness of  $t = 0.1'' \approx 2.5$  [mm]. [15]

The nozzle material is established first, being assumed as steel for maximum encountered total temperatures exceeding 700°F and titanium otherwise. The material properties are then set through the `getdensity()`. The casing masses are then calculated according to

$$m_{\text{casing}} = \rho \cdot t \cdot \pi \cdot (D_{\text{in}} + D_{\text{out}}) \cdot \sqrt{l^2 + (D_{\text{in}} - D_{\text{out}})^2} \quad (10.44)$$

The mass of both casings (if present) is then summed to estimate the total nozzle mass, concluding the method.

## 10.10. DUCT

The Duct class is intended to model ducts between components, which may be needed to adjust radii and/or velocities. Duct-class objects may be used as components of axial compressor and/or turbine stages, but can also be used as top-level components themselves.

**Duct class diagram**

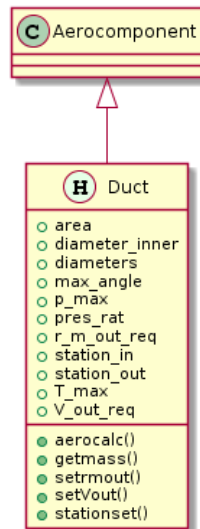


Figure 10.23: Class diagram for the Duct class.

### 10.10.1. PROPERTIES

Like all aerodynamic through-flow components, the Duct class is a subclass of the Aerocomponent class, as shown in Figure 10.23. Its non-inherited properties are few, and can be divided into design and descriptive parameters.

The design parameters include the `max_angle` property, which contains the maximum allowable tip or mean radius slope (in radians, default value  $\alpha_{\max} = 45^\circ$ ) over the duct. Another of these is the `pres_rat` property, which specifies the total pressure ratio<sup>40</sup> over the duct. If the Duct is used to obtain a certain outflow mean radius, this is specified in its `r_m_out_req` property. If a certain outflow axial velocity should be obtained, this is stored in the `V_out_req` property. If either of these two are left empty, it is assumed that that parameter should remain unchanged over the duct. Finally, the `station_in` and `station_out` properties contain the identifiers for the associated in- and outflow Station objects.

Descriptive properties include the `area`, containing the outflow area in [m<sup>2</sup>], the `diameter_inner` property, which contains the smallest hub diameter of the duct and the `diameters` property structure. The latter contains the in- and outflow inner and outer diameters. Finally, the `p_max` contains the maximum total pressure (in [Pa]) present within the duct and the `T_max` property contains the duct maximum present total temperature ([K]).

### 10.10.2. METHODS

The Duct class has several associated methods. These include its constructor `Duct()`, the design and on-design post-component conditions calculation method `aerocalc()` and the mass estimation method `getmass()`. Also, the `setrmout()`, `setVout()` and `stationset()` methods are present in order to set the required outflow mean radius, outflow axial velocity and in- and outflow station identifiers. These methods are however not described in detail here, as they purely set the appropriate properties based on their inputs.

#### THE DUCT () CONSTRUCTOR METHOD

The `Duct()` method is used to create a Duct-class object. It takes the name to be given to the new object, the identifier of the associated Flow and in- and outflow Station objects, as well as the desired total pressure

<sup>40</sup>This pressure ratio is used to take into account losses, and therefore purely acts upon the total pressure ratio. The total temperature ratio will remain 1, due to the assumption of no energy transfer between the flow and the duct.

ratio over the duct, and sets the appropriate properties of the Duct to these values. The length property of the duct is then defaulted to a value of  $l_{\text{default}} = 0.5$  [m], concluding the method.

#### THE AEROCALC() METHOD

The `aerocalc()` method is used to size the duct for on-design conditions, as well as to estimate the post-duct flow conditions.

It starts by searching for the in- and outflow `Station` objects in the `Flow` object provided as input. In case no outflow `Station` is found, one is constructed, initially as a clone of the inflow `Station`<sup>41</sup> with an updated name and number.

Next, the outflow `Station` mean radius and axial velocity are updated using the `Station`'s `updateRmean()` and `updateV()` methods. If these values are specified within the `Duct`'s properties, these values are used, otherwise the values are taken from the inflow `Station` object. The total pressure ratio is then taken into account, by multiplying it with the inflow total pressure, yielding the outflow total pressure. The outflow total temperature is assumed to be equal to the inflow total temperature (no addition of heat in the duct).

The duct length  $l_{\text{duct}}$  is then calculated based on the maximum allowable angle property `max_angle` and the difference in mean or tip radius, depending on whether the mean radius changes over the duct. Based on this value, the outflow `Station`'s `x_loc` property is updated.

Finally, the duct outflow area and maximum encountered total pressure and temperature are stored in the properties, and the various diameters are calculated, concluding the method.

#### THE GETMASS() METHOD

The `getmass()` method estimates the mass of the duct, by modelling it as an inner and outer thin-walled casing.

The method starts by setting the duct material, by checking whether the maximum encountered total temperature exceeds the switch temperature of  $T_{\text{switch}} = 700^\circ\text{F} \approx 644$  [K]. If it does, the duct is assumed to be made of steel, otherwise the duct material is assumed to be titanium. The material properties are then loaded through the `getdensity()` method.

The required wall thickness  $t$  is then estimated based on the maximum encountered total pressure, diameter and the material yield strength using

$$t = p_{\text{max}} \cdot \frac{D_{\text{max}}}{2 \cdot \sigma_y} \quad (10.45)$$

Both casings are assumed to have this thickness, even though the inner casing is loaded in compression, for which the buckling scenario may be more critical. The volumes of both cases are then estimated by assuming them as thin-walled straight cylinders with diameters equal to the mean inner and outer diameters, after which these volumes are added and multiplied with the density, yielding the mass estimate of the duct and concluding the method.

## 10.11. BLADE

The `Blade` class models the rotor and stator blades within axial turbomachinery components, such as compressor and turbine stages. As these are very different in terms of blade shape as well as calculation methods, this implies specific methods are present for the various types of blades.

### 10.11.1. PROPERTIES

In essence, the `Blade` class has only a few additional properties other than those inherited from its `Component`. The include the inner and outer diameter (in [m]) of the annulus in which the blade is located, the blade height (again in [m], should be equal to half the difference between the annulus diameters), the blade aspect ratio `AR`, the blade volume factor `K`, the blade volume in [m<sup>3</sup>] and the blade chord length in [m]. Additionally, a chord clearance property is present (set by default to 0.17), expressing the additional clearance assumed behind or in front of the blade as a perunage of the chord length, as well as a boolean indicating whether or not the blade is cooled (and therefore has cooling channels running through it).

<sup>41</sup>For a duct with no prescribed pressure ratio, outflow mean radius and axial outflow velocity, the properties will not change, so this is a good starting point.

Blade class diagram

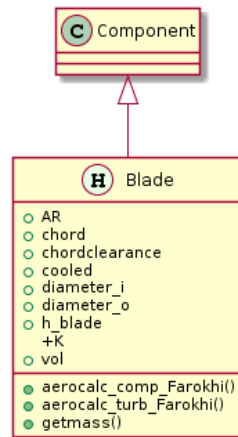


Figure 10.24: Class diagram for the Blade class.

### 10.11.2. METHODS

As stated, the Blade class is intended to model both compressor and turbine blades, which tend to be rather different. Therefore, separate on-design calculation methods are present for both types. In addition, a mass estimation method is present, as well as the mandatory constructor.

#### THE BLADE () CONSTRUCTOR METHOD

The Blade () constructor creates a Blade object from a name, inner and outer diameter, aspect ratio, blade volume factor and blade material choice. After setting these values to the appropriate properties, the getmass () method is called to get an initial estimate of the blade mass.

#### THE AEROCALC\_COMP\_FAROKHI () COMPRESSOR BLADE DESIGN METHOD

For a compressor blade, the sizing is performed using the aerocalc\_comp\_Farokhi () method, which is based on the methods as layed out in [18]. In order to perform this calculation, the method requires the blade height, mean radius, thickness-to-chord ratios<sup>42</sup>, solidity, blade Reynolds number, kinematic viscosity as well as the inflow Mach number, velocity and in- and outflow angles relative to the blade.

The blade chord is estimated based on the given Reynolds number, kinematic viscosity and inflow velocity. Next, based on the relative blade Mach number, the method assumes the blade to have either a Double-Circular-Arc (DCA, for  $M \geq 0.8$ ) or a NACA65-series airfoil. Depending on this choice, a calculation path is chosen.

For a DCA airfoil, this starts by assuming the mean blade thickness-to-chord ratio as the leading edge incidence angle  $\iota$ , which is followed by the blade deviation angle  $\delta$  using Carter's rule, supplemented with a 2° shock correction for  $M \geq 1$ , according to

$$\delta = \frac{\beta_{\text{out}} - \beta_{\text{in}}}{2 \cdot \sqrt{\sigma}} \quad (10.46)$$

The leading and trailing edge angles are then found by respectively subtracting the incidence and deviation angles from the in- and outflow angles, after which the camber angle is estimated by subtracting the trailing-edge angle from the leading-edge angle. Finally, the stagger angle  $\gamma$  is found by subtracting half this camber angle and the incidence angle from the inflow angle. This then allows for the determination of the axial chord length from the stagger angle and the chord, after which the total blade (row) length including clearance is found by multiplying this axial chord by unity plus the clearance factor.

For a NACA65-series profile, the method uses data extracted from the graphs of [24]. It starts by selecting the closest solidity in the available dataset. It then loops through all datasets with this solidity and checks whether the inflow angle lies within the range of that dataset. If so, it finds the maximum and minimum values for the outflow angle for this inflow angle within that dataset, after which this range is checked against the actual outflow angle. If the range is found to be appropriate, the corresponding angle of attack for this

<sup>42</sup>If two ratios (e.g. hub and tip values) are given, the average is taken, while for a single value, this value is used. If left empty, a default of  $\frac{t}{c} = 6.5\%$  is assumed.



combination of in- and outflow angles within this dataset is found through interpolation. Finally, the stall margins on the inflow angle are checked as well, after which the angle and these margins are written to a `found_points` matrix. If no viable points were found, the search is extended to the next closest solidity value, until a solidity yields viable points or the entire dataset has been exhausted.

This matrix with viable points is then checked for the point with the highest mean value of its stall margins, which is selected. The corresponding angle of attack and inflow angle are then subtracted from each other to find a value for the stagger angle, which is then also translated into an axial chord and blade row length.

Finally, regardless of the airfoil type, the inner and outer diameters, blade aspect ratio and volume are re-calculated and updated, after which the blade mass is re-estimated using the `getmass()` method.

#### THE AEROCALC\_TURB\_FAROKHI () TURBINE BLADE DESIGN METHOD

The turbine blade design procedure is also based on [18], and differs from the compressor blade strategy in many ways, among which the fact that the blade solidity is now an output instead of an input. The method requires the `Blade` object itself, the blade height, mean radius, throat Reynolds number, kinematic viscosity, Mach number and specific heat ratio, the outflow velocity and in- and outflow angles relative to the blade and the Zweifel coefficient as inputs.

The method first determines the optimal solidity  $\sigma_{\text{opt}}$  based on the in- and outflow angles  $\beta_{\text{in}}$  and  $\beta_{\text{out}}$  and the Zweifel coefficient  $\Psi$  using [18, Eq. 10.52]:

$$\sigma_{\text{opt}} = \frac{2}{\Psi} \cdot \frac{\cos \beta_{\text{out}}}{\cos \beta_{\text{in}}} \cdot \sin |\beta_{\text{in}} - \beta_{\text{out}}| \quad (10.47)$$

By dividing this optimal solidity by an approximation of the stagger angle (based on the mean of the velocity vectors), the “actual” solidity  $\sigma$  is estimated.

Next, the size of the throat opening  $o$  is estimated from the throat Reynolds number, kinematic viscosity and outflow velocity. If the flow is subsonic, the blade spacing  $s$  is then found through [18, Eq. 10.95]

$$s = \frac{o}{\cos \beta_{\text{out}}} \quad (10.48)$$

If the flow is supersonic, this value is corrected through multiplication with the sonic area ratio according to [18, Eq. 2.94]. With the spacing and solidity now known, the chord length is calculated through the definition of the solidity ( $\sigma = \frac{c}{s}$ ). Through the stagger angle approximation made earlier, the axial chord and blade row lengths are then estimated analogous to those for the compressor blades. The annulus diameters, blade aspect ratio and volume are then recalculated and updated (with the volume being reduced by 20% to account for cooling flow channels if the blade is cooled [15, p. 27]), after which the blade mass is recalculated as the final action of this method.

As output, this method returns the found solidity  $\sigma$ .

#### THE GETMASS () MASS ESTIMATION METHOD

Mass estimation of the blade is done in a relative short manner, with the blade volume  $V$  being estimated first through

$$V = K \cdot \frac{h_{\text{blade}}^3}{\text{AR}^2} \quad (10.49)$$

Again, this volume is decreased by 20% if the blade `cooled` property is true, after which the volume is multiplied by the blade material density to obtain the mass, which is then set to the appropriate property and returned.

## 10.12. Disk

The `Disk` class provides a model for rotor disks of axial turbomachinery components.

### 10.12.1. PROPERTIES

The `Disk` has only a few properties besides those inherited from its `Component` superclass. These include the identifier for the associated `Shaft` component, the disc type setting, the operating RPM, the blade pull stress, a setting for the calculation method used, the value of the shaft outer diameter and the axial chord length of the mounted blades.



Disk class diagram

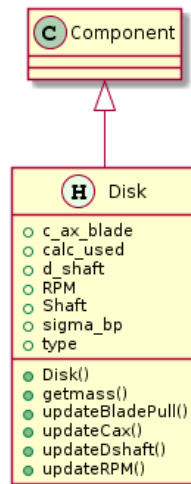


Figure 10.25: Class diagram for the Disk class.

### 10.12.2. METHODS

While most of the Disk methods are used purely for setting properties, such as the `updateRPM()`, `updateBladePull()`, `updateCax()` and `updateDshaft()` methods for setting respectively operating speed, blade pull stress, axial chord of mounted blades and diameter of the powering shaft, it also contains a constructor method as well as a mass estimation method.

#### THE DISK() CONSTRUCTOR

The `Disk()` constructor method takes the name of the to-be created disk, the shaft identifier, disk type, spool speed (in  $[\frac{\text{rev}}{\text{min}}]$ ), applied blade pull stress in [Pa], disk outer diameter in [m] and disk material, and sets these to the appropriate properties.

#### THE GETMASS() MASS ESTIMATION METHOD

The `getmass()` method takes the Disk object as its sole input, and switches its calculation path based on the `calc_used` property, which can either be WATE1977 (chart-based, default) or OnatKlees1979 (trapezoidal disc approximation, optional).

For the WATE1977 calculation, the blade pull stress, diameter and material yield stress are first converted to imperial units, after which the blade volume in  $[\text{in}^3]$  is estimated by interpolating the appropriate dataset from [14, Fig. 9, 11]. This is then converted back to SI units, and multiplied by the density to find the mass estimate for the disk.

If however, the trapezoidal disk approximation is used (as described in [15, Sec. 1.2.3]), the code starts by determining the rim inner radius, initially assumed as 90% of the disk outer radius. If however this yields a rim thickness of  $t_{\text{rim}} < \frac{3}{4}$ " , the rim inner radius is recalculated to yield a rim with this minimum thickness. This rim is then assumed to have a constant width equal to the axial chord length of the mounted blade, with its center of gravity assumed to be located at the mean radius of this rim. This then allows for the calculation of the volume and mass of this rim.

Using this mass, along with the rotational velocity, blade pull stress and an allowable stress of 75% ultimate stress, the trapezoidal web outer thickness  $t_o$  is estimated using

$$t_o = \frac{\sigma_{\text{bp}} \cdot r_o \cdot c_{\text{axblade}} + m_{\text{rim}} \cdot r_{\text{cogrim}} \cdot 2 \cdot \pi \cdot \left(\frac{\text{RPM}}{60}\right)^2}{r_{\text{rim}} \cdot \sigma_{\text{allow}}} \quad (10.50)$$

The web inner thickness  $t_i$  is then found by numerically solving (using `fzero()`) for an allowable stress equal to 50% of the ultimate stress. This also leads to an estimate of the web mass, which is added to the rim mass to yield the total disk mass.

## 10.13. FLOW

The Flow class does not model any physical component, but is a container for all stations on a certain flow path through the engine. It is also used to perform calculations on the distribution of bleed flow supply and

### Flow class diagram

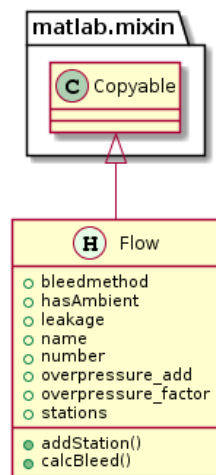


Figure 10.26: Class diagram for the Flow class.

demand.

#### 10.13.1. PROPERTIES

The properties of the Flow class, as shown in Figure 10.26, include the name and identifier of the specific flow, an array used to contain the various Station objects within it, a leakage factor to model bleed massflow losses, a boolean signifying whether or not an ambient station is present, two different factors (multiplication and addition) for determining required overpressure and a setting for where the bleed air should be sourced from (either by default the final compressor stage or optionally the first suitable station).

#### 10.13.2. METHODS

The Flow object has only three methods: its constructor, the addStation() method which simply appends a new Station to the end of the stations array and a calcBleed() method, used to perform (re-)distribution of the bleed air demand over the supply stations.

##### THE FLOW() CONSTRUCTOR

The constructor of the Flow class is very simple, as it simply takes the name, identifying number and an array containing Station objects as input, and assigns these to the appropriate properties of the object.

##### THE CALCBLEED() METHOD

The calcBleed() method is used to redistribute the bleed air demand over the possible supply station, and return the absolute and relative changes in the required supply to allow for determination of convergence.

If the bleed air should be sourced from the final compressor stage (which should be the bleed air supply Station with the highest total pressure), this station is found first.

Next, for each demand station, the optimal supply station is found by checking the local total pressure of the supply station exceeds that at the demand station, corrected by the relative and absolute required overpressure values. The required massflow of the demand station is then added to the to-be-supplied massflow of the optimal or last supply station, and total temperatures and local gas compositions from the supply station are transferred to the demand station.

Finally, the errors in massflow for each supply station are determined as absolute and relative numbers, with the maxima of both being returned by the method.

## 10.14. SHAFT

The Shaft models the shaft which connects the rotating power-producing and -consuming components, and also serves as a container for the shaft stations.

### Shaft class diagram

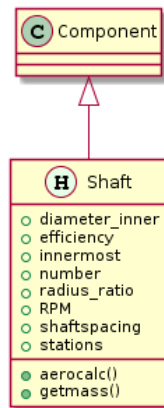


Figure 10.27: Class diagram for the `Shaft` class.

#### 10.14.1. PROPERTIES

As can be seen in Figure 10.27, the `Shaft` class has some properties besides those inherited from the `Component` superclass. These include the shaft identifier, the spool speed in  $\left[\frac{\text{rev}}{\text{min}}\right]$ , the efficiency, an array for holding the `Shaftstation` objects, a radius ratio used for sizing the innermost shaft, an inner diameter value, which can be both a sizing input (for a non-innermost shaft) or an output, a boolean signifying whether the shaft is the innermost shaft of the engine and a value for the radial spacing required between this and another concentric shaft.

#### 10.14.2. METHODS

The `Shaft` class has only three methods: its constructor, the `aerocalc()` method used within the design and performance iteration and the mass estimation method `getmass()`.

##### THE `SHAFT()` CONSTRUCTOR

The constructor method requires a name for the shaft, a shaft identifier number, a power transmission efficiency and a value for the required shaft spacing. These are set to the properties of the `Shaft` object, which is then returned.

##### THE `AEROCALC()` METHOD

The `aerocalc()` method is used to update the shaft speed and size the shaft diameters, requiring both the `Shaft` object itself as well as the `Flow` object in which it is mounted.

The method starts by finding the lowest RPM limit of all the `Shaftstation` and setting this as the new RPM of the shaft. It also updates the  $x$ -locations of all of these stations by scouring the `Flow` for the associated `Station`, from which the location is copied to the `Shaftstation`. The length of the shaft is then defined as the difference between the lowest and highest  $x$ -locations on the shaft.

Next, the total power supplied to the shaft is found, after which it is converted to a torque by dividing it by the spool speed in  $\left[\frac{\text{rad}}{\text{s}}\right]$ . If the shaft is the innermost, it is then sized based on the set radius ratio and yield shear stress, while otherwise the outer diameter is based on the prescribed inner diameter and yield shear stress.

##### THE `GETMASS()` METHOD

The mass estimation first redoes the sizing part of the `aerocalc` method, after which the shaft mass is estimated by modelling the shaft as a uniform thin-walled tube, of which the volume is calculated and multiplied by the density of the shaft material.

### 10.15. STATION

The `Station` is used to store the flow properties at a certain point in the engine. It also allows for the calculation of any missing properties.

Station class diagram

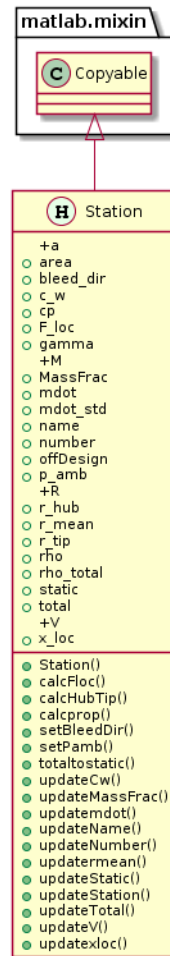


Figure 10.28: Class diagram for the Station class.

### 10.15.1. PROPERTIES

The `Station` has multiple properties, required to model the gas and flow properties at a certain location. These include the static and total temperatures and pressures (both contained in a structure), the flow area, total and static densities, axial and whirl velocities, absolute and standardised massflow rate,  $c_p$ ,  $R$  and  $\gamma$  values, gas composition, Mach number and local sonic velocity. Also some other parameters are stored, such as a station name, station identifier number, annulus mean, hub and tip radii and location of the station in  $x$ -direction. The `p_amb` value stores the ambient pressure in [Pa], while the `F_loc` property contains the sum of the local flow impuls and pressure forces. Finally, a property is present for indicating the bleed air flow direction (supply or demand) for a bleed flow `Station`, as well as a yet unused `offDesign` property, intended for storing off-design conditions when this will be implemented.

### 10.15.2. METHODS

While the `Station` class contains many methods, most of these are used just to update certain properties, after which the property calculations are usually rerun to obtain a consistent set of properties at that station. The methods in which actual calculations happen are the `totaltostatic()`, `calcprop()`, `calcHubTip()` `calcFloc()` methods.

#### THE TOTALTOSTATIC() METHOD

The `totaltostatic()` method is used to calculate static properties from a given set of total properties. It first calls the `getGasProp()` function using the total temperature  $T_t$  and gas composition to obtain the specific gas constant  $R$ , the specific heat ratio  $\gamma$  and the  $c_p$ . Then, as long as a local velocity is specified, it first calculates the local absolute velocity  $V_{abs}$ , which is then used to obtain the static temperature  $T$  using

$$T = T_t - \frac{V_{abs}^2}{2 \cdot c_p} \quad (10.51)$$

This static temperature is then used to find the local sonic velocity  $a$ , using which the local Mach number is calculated. The static pressure  $p$  is then found through

$$p = \frac{p_t}{\left(1 + \frac{\gamma-1}{2} \cdot M^2\right)^{\frac{\gamma}{\gamma-1}}} \quad (10.52)$$

Next, the static and total densities are calculated using the ideal gas law, after which the flow area is defined based on the massflow rate, static density and axial velocity. The hub and tip radii are then updated through the `calcHubTip()` method. Finally, the standardised massflow rate is computed using

$$\dot{m}_{std} = \dot{m} \cdot \sqrt{\frac{R \cdot T_t}{R_{std} \cdot T_{std}}} \cdot \frac{p_{std}}{p_t} \quad (10.53)$$

#### THE CALCPROP() METHOD

The `calcprop()` method performs the same function as the `totaltostatic()` method, however without performing the conversion between total and static properties, instead assuming these as already being correct. Calculation of the standardised massflow is also left out.

#### THE CALCHUBTIP() METHOD

The `calcHubTip()` method calculated the hub and tip radii of an annulus assuming the flow area and mean radius are known. This is done through the equations

$$\begin{aligned} h &= \frac{A}{2 \cdot \pi \cdot r_m} \\ r_h &= r_m - \frac{h}{2} \\ r_t &= r_m + \frac{h}{2} \end{aligned} \quad (10.54)$$

#### THE CALCFLUC() METHOD

This method is intended to compute the sum of the impuls and pressure force at this station, which is done through

$$F_{loc} = V_{ax} \cdot \dot{m} + A \cdot (p - p_{amb}) \quad (10.55)$$

## 10.16. SHAFTSTATION

The `Shaftstation` contains information on power transfer to or from the shaft at a certain location, as well as on the local maximum allowed rotational velocities et cetera.

## Shaftstation class diagram

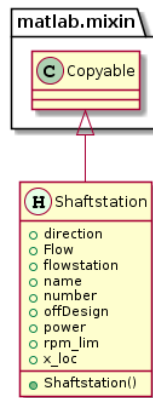


Figure 10.29: Class diagram for the Shaftstation class.

## 10.16.1. PROPERTIES

The Shaftstation has only a few properties. These include the name and identifier of the station, the amount of power transferred at that location (in [W], always positive!) and whether this is supplied or consumed, an  $x$ -location, the identifiers of the corresponding Station and Flow, the limit spool speed of the connected component (in [RPM]) and a currently unused `offDesign` property, intended for storing the properties of future off-design calculations.

## 10.16.2. METHODS

Except for its constructor, the Shaftstation class does not have any other methods.

## 10.17. INLETGUIDEVANE

## InletGuideVane class diagram

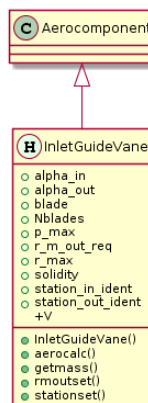


Figure 10.30: Class diagram for the InletGuideVane class.

The InletGuideVane class is used as a component of an axial turbomachinery stage if needed, in order to correct the inflow angle if required.

## 10.17.1. PROPERTIES

As the InletGuideVane essentially acts like a stator, it has similar properties. These include the solidity, number of blades, axial outflow velocity, in- and outflow station identifiers, in- and outflow angles, a container for the Blade object and an optional required outflow mean radius. Additionally, it contains properties for storing the maximum encountered total pressure and radius in the component.

### 10.17.2. METHODS

The methods included in this class are its constructor, the `aerocalc()` on-design performance and design method, the `getmass()` mass estimation method and 2 methods for modifying the required outflow mean radius and associated station identifiers. As the latter two only set these parameters, these are not discussed in detail any further.

#### THE `INLETGUIDEVANE()` CONSTRUCTOR

The constructor requires the name of the new object, the identifiers of the relevant `Flow` and in- and outflow `Station` objects, as well as optionally information on the required outflow angle, axial velocity and material. These properties are set to the newly created object, which is then returned.

#### THE `AEROCALC()` METHOD

The `aerocalc()` method is intended to perform the on-design performance and sizing of the inlet guide vane. In order to do this, it takes the `InletGuideVane` object, the `Flow` object in which its in- and outflow stations are located, the required outflow angle (if any) and axial outflow velocity (if any). First, the appropriate `Station` objects are found from the `Flow`. Next, the outflow whirl velocity is calculated from the desired outflow angle and axial velocity. As the inlet guide vane is modelled as a lossless stator blade row, this then fully specifies the outflow station conditions, by copying over the inflow total conditions to the outflow station and updating the axial and whirl velocities.

Next, a blade solidity of  $\sigma = 1$  is assumed, and the inflow Mach number, kinematic viscosity and relative velocity are estimated, based on which a new blade is created and calculated using the `aerocalc_comp_Farokhi()` method. The number of blades is then calculated from the chord length, mean radius and solidity, as well as the inlet guide vane blade row length. This in turn is used to update the outflow station  $x$ -location as the final action of this method.

#### THE `GETMASS()` METHOD

The `getmass()` method is very simple for the `InletGuideVane` class, effectively containing just the stator parts of the `Axcompressorstage` class `getmass()` method. First, the blade mass is obtained from the `Blade` object, after which a minimum thickness and a thickness for containment of the internal pressure are calculated. The highest of the two is then selected, based upon which the casing mass is estimated. Adding this to the blade mass multiplied by the number of blades yields the mass estimate, which is then returned.

## 10.18. FRAME

Frame class diagram

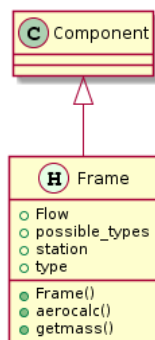


Figure 10.31: Class diagram for the `Frame` class.

The `Frame` class models the support frames of the engine, which do contribute to the engine mass estimation, however do not have any direct aerodynamic or thermodynamic functions.

### 10.18.1. PROPERTIES

As seen from Figure 10.31, the `Frame` class is a subclass of `Component`, and additionally defines four new properties. These include a string array containing allowable frame type strings, a property containing the string setting the type of frame and the identifiers of the `Station` at which the frame is present and its `Flow`.

### 10.18.2. METHODS

Besides its constructor, the Frame class has two other methods, the `aerocalc()` method for design purposes and the `getmass()` method for mass estimation.

#### THE AEROCALC() METHOD

As the Frame class does not perform any aerodynamic or thermodynamic function, the `aerocalc()` method is used purely to retrieve the frame dimensions. In order to do this, it requires the relevant Frame and Flow objects as inputs, after which the latter is searched for the Station at which the frame is located. When this is located, the local tip radius is doubled and set as the frame diameter, ending the method.

#### THE GETMASS() METHOD

As loads on the frame are unknown, making actual frame design impossible, the frame mass is estimated according to [15, Fig. 17]. Based on whether the frame is classified as a single-bearing frame, with or without power take-off, a turbine frame or an intermediate frame type, the appropriate piecewise linear relations from this figure are used to estimate the frame mass.

## 10.19. SUPERCLASSES

As mentioned in multiple class descriptions, most classes are subclasses of the Component and/or AeroComponent classes. These are created to contain some standard properties for almost every component, and are described below.

### 10.19.1. COMPONENT

Component class diagram

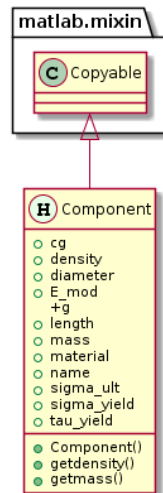


Figure 10.32: Class diagram for the Component superclass.

The Component superclass is intended to provide the basic properties and methods which almost every physical component needs. It is not intended for use by itself.

#### PROPERTIES

These properties include a name (mostly used for easy identification), a mass (in [kg]), length and diameter (all in [m]) and some material properties. Besides a string defining this material, these properties include the density (in  $\left[\frac{\text{kg}}{\text{m}^3}\right]$ ), yielding and ultimate tensile stresses, yield shear stress and Young’s modulus (all in [Pa]). Finally, it defines the value for the gravitational constant  $g$ , as well as a  $cg$  property intended as a vector indicating the absolute location of the component’s center of gravity, however this is currently still unused.

#### METHODS

Three methods are defined for the Component class. The `Component()` constructor method does nothing except for creating a Component-class object and setting the name property to the provided value. Similarly simple, the `getmass()` method simply returns the value set to the object’s mass property.



The `getdensity()` method is the only method which is more than a placeholder or backstop, setting the material properties based on the contents of the material string property, according to Table 10.2.

material	steel	titanium	
Density $\rho$	8050	4420	$\frac{\text{kg}}{\text{m}^3}$
Tensile yield stress $\sigma_y$	$250 \cdot 10^6$	$830 \cdot 10^6$	[Pa]
Tensile ultimate stress $\sigma_{\text{ult}}$	$400 \cdot 10^6$	$900 \cdot 10^6$	[Pa]
Shear yield stress $\tau_y$	$\frac{1}{2} \cdot \sigma_y$	$\frac{1}{2} \cdot \sigma_y$	[Pa]
Young's modulus $E$	$200 \cdot 10^9$	$105 \cdot 10^9$	[Pa]

Table 10.2: Material properties set by the Component-class `getdensity()` method

### 10.19.2. AEROCOMPONENT

Aerocomponent class diagram

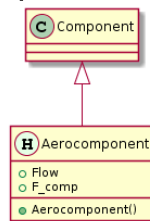


Figure 10.33: Class diagram for the Aerocomponent superclass.

The `Aerocomponent` superclass is a subclass of the `Component` class, adding information on associated `Flow` objects to the properties. Currently, as shown in Figure 10.33, this purely includes properties for the identifier of the associated `Flow` and to store the component thrust `F_loc`, as well as a constructor method `Aerocomponent()` which, after calling the `Component()` constructor with the name of the new component, sets this `Flow` identifier property to the provided value as well.

# 11

## INTEGRATION INTO THE INITIATOR

After the MIME code was written and found to function separately, the next step is to integrate it into the Initiator. This chapter will describe the challenges, chosen solutions and other considerations involved with this process.

### 11.1. GOALS

While the MIME code is intended to be as flexible as possible, simulating a variation of engines, the convergence of the Initiator is currently only built to create engines with preset properties, only varying the fan diameter. In order to avoid redefining the Initiator convergence methods regarding engines at this point, using MIME as a drop-in replacement is desirable, as already mentioned in Section 9.1.

As it has been shown that the MIME method does not yet function perfectly, especially regarding the engine mass estimation, it is desirable that the use of the MIME method for engine design and engine mass estimation remain selectable options, besides the current engine modelling methods.

### 11.2. IMPLEMENTATION

In order to implement the MIME method, changes were made to the definition of the existing `EngineModel` class of the Initiator, the associated `run.m` file and the `getEngineWeight()` method of the `Class2WeightEstimation` analysis module. To the former, a new property `MIME_model` was added, in which the `AeroEngine` object resulting from the MIME design calculations can be stored for future reference.

In the `EngineModel`'s `run.m` file, the value of the new setting `UseMIMEmodel`, which was added to the `ProgramSettings.xml`, is now also retrieved with the other engine-related settings. For the `TurboFan`-type engines, if this value is true, the new `MIMEwrapper()` function is called instead of the original `Turbofan_Initiator()` function, both for the standard and small-deviation calculations.

This `MIMEwrapper()` function has the same in- and outputs as the `TurbofanInitiator()` function, with the addition of an in- and output for the respectively already known and generated `AeroEngine` objects, and the exception of the no-longer-used turbine stage numbers. From the inputs, an options structure is constructed with all the provided properties. If the input for the existing `AeroEngine` object is empty, a new object is constructed, while otherwise, the diameter, bypass ratio and fan pressure ratios<sup>1</sup> are checked and updated if required. The `onDesignCalc()` method is then called on the new or updated object, after which the thrust, fuel massflow rate and the object itself are returned.

For the `getEngineWeight()` method, the changes are limited. For the case where the engine type is defined as a `TurboFan`<sup>2</sup>, the values of the `UseMIMEmodel` and `UseMIMEmass` program settings are retrieved. If both are true, the `EngineModel` object is retrieved and checked whether its `MIME_model` property is empty.<sup>3</sup> If so, or if any of the mentioned settings are empty or false, the original database-based dry engine mass calculation is used. If all of these are true and the `MIME_model` property contains a previously generate `AeroEngine`

<sup>1</sup>These parameters are the only ones possibly being modified by the Initiator during a convergence calculation.

<sup>2</sup>As this is the only case for which the MIME model is implemented at the moment.

<sup>3</sup>This may occur as the `Class2WeightEstimation` module, of which the `getEngineWeight()` function is a part, is run before the `EngineModel` analysis module.

object, its `massCalc()` method is called to obtain the MIME engine mass estimate, which is then used as the dry mass for the rest of the engine mass calculations already present.

The MIME code itself is stored in a sub-folder of the `EngineModel` folders of both the `External` as well as the `AnalysisModules` folders. While the latter place is the more logical location for storing engine model code, for some reason, the code in the former location is used.

### 11.3. POTENTIAL IMPROVEMENTS

In terms of implementation, the most important improvement is to allow for engine lay-out definition directly from the settings files, instead of still building the same twin-spool engine architecture. Ideally, the `options` structure itself would be defined in (a series of) settings.

Another potential improvement to the implementation would be to give a MIME-modelled engine an engine type of its own (or alternatively have the type be dependent on the contents of the MIME engine model).

# 12

## GUIDE FOR EXPANSION

While MIME is intended as a modular and therefore very versatile and future-proof code, its current limited assortment of available components keeps it from fulfilling its full potential. As Section 7.2 contains a list of future modules which could possibly be added to the code, this chapter gives a short guide for how to implement a new module: which methods are required, which interfaces are available and which classes and functions should be modified.

### 12.1. CREATING A MIME MODULE

In order to create a new module for MIME, the first step (of course after theoretically working out how the module should function) is to create a class. Depending on the type of component being modelled, this should be a subclass of the `Component` (general components, not associated with specific `Flow` objects, such as a gearbox or generator) or `Aerocomponent` (components associated with a certain `Flow`, such as compressors, splitters, bleed air offtakes) class. If neither of these fit, make sure to use at least the `handle` and `matlab.mixin.copyable` property.

In terms of methods, the class should contain at least a constructor, the `getmass()` method<sup>1</sup> and an `aerocalc()` or similar on-design method.

The `getmass()` method is called during mass estimation, and should only require the object on which it is being called as input. It should calculate the mass of the component it models, set this to the proper property of the object and also return this value.

An on-design method is called during the engine design and on-design performance calculation loop, and may require more inputs, most typically `Flow` or `Shaft` objects, from and to the `Stations` and `Flowstations` of which the old and new in- and outflow properties are transferred. During programming, it is good to keep in mind that outflow stations sometimes may not yet exist and need to be created in that case. Please check the current MIME code for examples.

### 12.2. CONNECTING THE MODULE

After the class is created, it needs to be integrated within MIME. For this, several steps need to be taken.<sup>2</sup> First, the component needs to be created and placed correctly. Next, the component methods need to be properly called when needed.

In order to create the component object, the class file first needs to be located in the same folders as the other MIME classes, after which a new creation method (`build*(obj, options_comp)`) for this class should be added to the `AeroEngine` object. This creation method should take in a structure of options specific to the object, fill in any blanks and then call the class constructor to actually return the object. The calling of this method upon encountering the associated fields in the `options` structure should then be added into the `buildAeroEngine()` method of the `AeroEngine` class.

The next function to be modified is the `order_engine_array()` function. Depending on when the module is supposed to be run, the class name should be added to one of the groups defined on lines 28-48.

<sup>1</sup>For `Component` subclasses, this is already defined as a method returning the `mass` property value, however for most classes, it is desirable to redefine this method.

<sup>2</sup>Depending on the nature of the module, not all of these may be required. This should be obvious to the programmer.

Finally, the module needs to be added to the `AeroEngine`'s `onDesignCalc()` method. Within the design loop, a check for the object class should be added, within which the object's on-design calculation method should be called, as well as all required other object and/or information should be found and provided. This may include relevant `Flow` and/or `Shaft` objects, or even the object representing the preceding or next component. Browsing through the current code should provide sufficient inspiration. As a final action, the counter is incremented by 1.

When a `getmass()` method is defined for the class, there is no need to code the new module into the `AeroEngine`'s `massCalc()` method separately, as this will call the `getmass()` method of all objects from the `components` array which have one.

## BIBLIOGRAPHY

- [1] M. Python, *Monty python - "the bridge of death" quote-a-long*, .
- [2] A. W. X. Ang, A. G. Rao, T. Kanakis, and W. Lammen, *Performance analysis of an electrically assisted propulsion system for a short-range civil aircraft*, *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering* (2018), 10.1177/0954410017754146, <https://doi.org/10.1177/0954410017754146> .
- [3] J. Roskam, *Airplane design Part V: component weight estimation* (Roskam Aviation and Engineering Corporation, 1985).
- [4] D. P. Raymer, *Aircraft Design: a conceptual approach* (American Institute of Aeronautics and Astronautics, 1992).
- [5] E. Torenbeek, *Synthesis of subsonic airplane design* (Delft University Press, 1982).
- [6] R. Elmendorp, R. Vos, and G. La Rocca, *A conceptual design and analysis method for conventional and unconventional airplanes*, in *ICAS 2014: Proceedings of the 29th Congress of the International Council of the Aeronautical Sciences, St. Petersburg, Russia, 7-12 September 2014* (International Council of Aeronautical Sciences, 2014).
- [7] G. La Rocca, *Knowledge based engineering techniques to support aircraft design and optimization*, Ph.D. thesis, Delft University of Technology (2011).
- [8] R. Vos and J. van Dommelen, *A conceptual design and optimization method for blended-wing-body aircraft*, in *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference, Honolulu, Hawaii, 23-26 April 2012* (American Institute of Aeronautics and Astronautics, 2012).
- [9] A. R. Byerley, A. J. Rolling, and K. W. Van Treuren, *Estimating gas turbine engine weight, costs, and development time during the preliminary aircraft engine design process*, in *Proceedings of ASME Turbo Expo 2013: Turbine Technical Conference and Exposition, June 3-7, 2013, San Antonio, Texas, USA* (ASME, 2013).
- [10] P. Lolis, P. Giannakakis, V. Sethi, A. Jackson, and P. Pilidis, *Evaluation of aero gas turbine preliminary weight estimation methods*, *The Aeronautical Journal* **118**, 625 (2014).
- [11] P. Jeschke, J. Kurzke, R. Schaber, and C. Riegler, *Preliminary gas turbine design using the multidisciplinary design system mopeds*, *Journal of Engineering for Gas Turbines and Power* **126**, 258 (2004).
- [12] P. Prado, Y. Panchenko, J.-Y. Trepanier, and C. Tribes, *Preliminary multidisciplinary design optimization system: A software solution for early gas turbine conception*, in *Proceedings of GT2005 ASME Turbo Expo 2005: Power for Land, Sea and Air, June 6-9, 2005, Reno-Tahoe, Nevada, USA* (ASME, 2005).
- [13] A. Karl, R. Hansen, and M. Pfitzner, *Computer based optimisation and automation of analysis and design processes in aero engine development*, in *Proceedings of Turbo Expo 2002: ASME TURBO EXPO 2002: LAND, SEA AND AIR, 3-6 June 2002, Amsterdam* (ASME, 2002).
- [14] R. Pera, E. Onat, G. Klees, and E. Tjonneland, *A Method to Estimate Weight and Dimensions of Aircraft Gas Turbine Engines - Final Report*, Tech. Rep. CR135170 (National Aeronautics & Space Administration, 1977).
- [15] E. Onat and G. Klees, *A Method to Estimate Weight and Dimensions of Large and Small Aircraft Gas Turbine Engines - Final Report*, Tech. Rep. CR-159481 (National Aeronautics & Space Administration, 1979).
- [16] P. Lolis, *Development of a Preliminary Weight Estimation Method for Advanced Turbofan Engines*, Ph.D. thesis, Cranfield University School of Engineering (2014).

- [17] H. Saravanamuttoo, G. Rogers, H. Cohen, and P. Straznicky, *Gas Turbine Theory*, sixth ed. (Pearson Education Limited, 2009).
- [18] S. Farokhi, *Aircraft Propulsion*, 2nd ed. (John Wiley & Sons Ltd, 2014).
- [19] M. Pini, *Ae4206 turbomachinery lecture 7: Turbines*, Lecture slides (2016).
- [20] M. Pini, *Ae4206 turbomachinery lecture 8: Compressors*, Lecture slides (2016).
- [21] R. Vos and S. Farokhi, *Introduction to Transonic Aerodynamics* (Springer Science+Business Media Dordrecht, 2015).
- [22] Anonymous, *Certification Specifications and Acceptable Means of Compliance for Engines CS-E Amendment 4*, Tech. Rep. (European Aviation Safety Agency, 2015).
- [23] E. Obert, R. Slingerland, D. Leusink, T. van den Berg, and J. Koning, *Aerodynamic Design of Transport Aircraft* (Ios Press, 2009).
- [24] J. Horlock, *Axial Flow Compressors: Fluid Mechanics and Thermodynamics* (Butterworths Scientific Publications, 1958).
- [25] M. T. Tong, I. Halliwell, and L. J. Ghosn, *A computer code for gas turbine engine weight and disk life estimation*, *Journal of Engineering for Gas Turbines and Power* **126**, 265 (2004).
- [26] Anonymous, *The Jet Engine* (Rolls-Royce plc, 1986).
- [27] Anonymous, *EASA TYPE-CERTIFICATE DATA SHEET CFM International SA CFM56-2 & CFM56-3 series engines*, Tech. Rep. E.066 (European Aviation Safety Agency, 2008).
- [28] Anonymous, *Orpheus*, FLIGHT International, 28 (1969).
- [29] Anonymous, *EASA TYPE-CERTIFICATE DATA SHEET Rolls/Royce plc RB211 Trent 500 Series Engines*, Tech. Rep. E.060 (European Aviation Safety Agency, 2007).
- [30] *Airbus a320*, (2018).
- [31] Anonymous, *The ge90 - an introduction*, Retrieved on 2017-08-16 from [https://web.stanford.edu/~cantwell/AA283\\_Course\\_Material/GE90\\_Engine\\_Data.pdf](https://web.stanford.edu/~cantwell/AA283_Course_Material/GE90_Engine_Data.pdf).
- [32] D. A. R. Martins, *Off-Design Performance Prediction of the CFM56-3 Aircraft Engine*, Master's thesis, Instituto Superior Técnico, Lisboa (2015).
- [33] M. A. York, W. W. Hoburg, and M. Drela, *Turbofan engine sizing and tradeoff analysis via signomial programming*, *Journal of Aircraft*, 1 (2017).
- [34] Anonymous, *Gas Turbine Engine Performance Station Identification and Nomenclature*, Tech. Rep. ARP755A (Society of Automotive Engineers, 1974).
- [35] S. Dixon and C. Hall, *Fluid Mechanics and Thermodynamics of Turbomachinery*, seventh ed. (Butterworth-Heinemann, 2014).