FAST-(OAD)-GA

unknown

Feb 23, 2023

CONTENTS

For a description of models used in FAST-(OAD)-GA, you may see the *model documentations*.

Note: Models in FAST-(OAD)-GA are still a work in progress.

CHAPTER ONE

CONTENTS

1.1 License

GNU GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Preamble

The GNU General Public License is a free, copyleft license for

software and other kinds of works.

The licenses for most software and other practical works are designed

to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not

price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you

these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether

gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps:

(1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains

that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents.

States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and

modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of

works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this

License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work

in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based

on the Program.

To "propagate" a work means to do anything with it that, without

permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other

parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices"

to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work

for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other

than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all

the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users

can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that

same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of

copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not

convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under

the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological

measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid

circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you

receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey,

and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to

produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

> a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
>
> b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".
>
> c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
>
> d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent

works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms

of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

> a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
>
> b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in

the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded

from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any

tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods,

procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or

specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a

requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this

License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option

remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you

add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

> a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
>
> b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
>
> c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or
>
> d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
>
> e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
>
> f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further

restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you

must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the

form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly

provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your

license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is

reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the

licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or

run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically

receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an

organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the

rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this

License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims

owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free

patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express

agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license,

and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or

arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within

the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting

any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or

otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have

permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of

the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the

Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future

versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different

permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY

APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING

WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided

above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

1.2 Contributors

- Aurelien REYSSET <aurelien.reysset@insa-toulouse.fr>
- Florent LUTZ <Florent.LUTZ2@isae-supaero.fr>
- Aitor BUSTEROS RAMOS <Aitor.BUSTEROS-RAMOS@student.isae-supaero.fr>
- Lucas REMOND <Lucas.REMOND@isae-supaero.fr>

1.3 Changelog

1.3.1 Version 1.2.0

- Made use of the @functools.lru_cache function in order to save time on function that read csv files see #188
- Reworked some of the solvers to save some time and deleted some that were not useful see #195
- Added yml as a viable input for the block analysis
- Development of high wing models by pulling the work of @fomra during his summer 2022 internship
- Implementation of twist and dihedral in logitudinal aerodynamics see #202
- Changed the function of the Cl=f(curve) to match a new reference wing Cl see #202

1.3.2 Version 1.1.0

- Modified the Xfoil interface to allow for single AoA launch see #182
- Modification of the usage of the FlightPoint dataclass so that it is possible to add new fields more easily see #187
- Added a module for the computation of more aerodynamic derivatives see #185

1.3.3 Version 1.0.4

- Added compressibility corrections and reynolds correction in the BEMT code
- Change in the propeller code organization
- Added a new representation of the propeller performance under the form of Ct and Cp graphs
- Centralized all post_processing function in a post-processing API
- Added an option for the path to airfoil files

1.3.4 Version 1.0.3

- Removed unnecessary use of matplotlib in post_processing functions

1.3.5 Version 1.0.2

- Updated binder requirements

1.3.6 Version 1.0.1

- Changed the homepage address in the *pyproject.toml*

1.3.7 Version 1.0.0

- Added an analytical model for wing mass estimation
- Most models are now registered using the submodel feature from FAST-OAD-core
- Added an analytical model for fuselage mass estimation
- Dependencies were updated to reflected the separation between core and models in FAST-OAD
- Generalized the usage of shape_by_conn option when possible
- Added propeller installation effects
- Added link to Binder-hosted notebooks
- Sped up the computation of propeller performances, of aircraft performances and of IC engine fuel consumption
- Added a model for turboprop fuel consumption computation
- Added a new mission modules solves all the FlightPoints at once instead of using a time step approach
- Added Daher TBM900 as a reference aircraft

1.3.8 Version 0.1.4-beta

- Changed the variables that define nacelle position, they are now arrays
- When reading the .csv polar Mach number within a given precision are now read
- Variable descriptions were added
- Minor model error corrections
- Like the **generate_xml_file** was added which creates a default input file that matches what is need in **generate_configuration_file**
- Added polar computation
- Added payload-range diagram computation
- Use of the mission builder feature in FAST-OAD-GA is now possible
- Changed the name of some variables to make the use of the mission builder possible namely: data:geometry:propulsion:count, data:geometry:propulsion:layout, data:geometry:propulsion:y_ratio

1.3.9 Version 0.1.3-beta

- NACA .csv polar files replaced to correct xfoil_polar.py read issues
- Correction of security issues using **exec** and **eval** commands

1.3.10 Version 0.1.2-beta

- Unitary tests based on converged OAD aircraft .XML
- OAD process (integration test and API) switched to VLM method to work on linux/mac os
- Minor changes in Notebooks

1.3.11 Version 0.1.0-beta

- First beta-release

1.4 fastga

1.4.1 fastga package

Subpackages

fastga.command package

Subpackages

Submodules

fastga.command.api module

Module contents

fastga.configurations package

Module contents

fastga.models package

Subpackages

fastga.models.aerodynamics package

Subpackages

fastga.models.aerodynamics.airfoil_folder package

Module contents

fastga.models.aerodynamics.components package

Subpackages

fastga.models.aerodynamics.components.fuselage package

Submodules

fastga.models.aerodynamics.components.fuselage.compute_cm_alpha_fus module

fastga.models.aerodynamics.components.fuselage.compute_cn_beta_fuselage module

fastga.models.aerodynamics.components.fuselage.compute_cy_beta_fuselage module

Module contents

fastga.models.aerodynamics.components.ht package

Submodules

fastga.models.aerodynamics.components.ht.compute_cl_beta_ht module

fastga.models.aerodynamics.components.ht.compute_cl_pitch_rate_ht module

fastga.models.aerodynamics.components.ht.compute_cl_roll_rate_ht module

fastga.models.aerodynamics.components.ht.compute_cm_pitch_rate_ht module

fastga.models.aerodynamics.components.ht.downwash_gradient module

Module contents

fastga.models.aerodynamics.components.vt package

Submodules

fastga.models.aerodynamics.components.vt.compute_cl_alpha_vt module

fastga.models.aerodynamics.components.vt.compute_cl_beta_vt module

fastga.models.aerodynamics.components.vt.compute_cl_roll_rate_vt module

fastga.models.aerodynamics.components.vt.compute_cl_yaw_rate_vt module

fastga.models.aerodynamics.components.vt.compute_cn_beta_vt module

fastga.models.aerodynamics.components.vt.compute_cn_roll_rate_vt module

fastga.models.aerodynamics.components.vt.compute_cn_yaw_rate_vt module

fastga.models.aerodynamics.components.vt.compute_cy_beta_vt module

Module contents

fastga.models.aerodynamics.components.wing package

Submodules

fastga.models.aerodynamics.components.wing.compute_cl_beta_wing module

fastga.models.aerodynamics.components.wing.compute_cl_pitch_rate_wing module

fastga.models.aerodynamics.components.wing.compute_cl_roll_rate_wing module

fastga.models.aerodynamics.components.wing.compute_cl_yaw_rate_wing module

fastga.models.aerodynamics.components.wing.compute_cm_pitch_rate_wing module

fastga.models.aerodynamics.components.wing.compute_cn_roll_rate_wing module

fastga.models.aerodynamics.components.wing.compute_cn_yaw_rate_wing module

fastga.models.aerodynamics.components.wing.compute_cy_beta_wing module

Module contents

Submodules

fastga.models.aerodynamics.components.airfoil_lift_curve_slope module

fastga.models.aerodynamics.components.cd0 module

fastga.models.aerodynamics.components.cd0_fuselage module

fastga.models.aerodynamics.components.cd0_ht module

fastga.models.aerodynamics.components.cd0_lg module

fastga.models.aerodynamics.components.cd0_nacelle module

fastga.models.aerodynamics.components.cd0_other module

fastga.models.aerodynamics.components.cd0_total module

fastga.models.aerodynamics.components.cd0_vt module

fastga.models.aerodynamics.components.cd0_wing module

fastga.models.aerodynamics.components.compute_L_D_max module

fastga.models.aerodynamics.components.compute_cl_aileron module

fastga.models.aerodynamics.components.compute_cl_alpha_dot module

fastga.models.aerodynamics.components.compute_cl_beta module

fastga.models.aerodynamics.components.compute_cl_extreme module

fastga.models.aerodynamics.components.compute_cl_extreme_htp module

fastga.models.aerodynamics.components.compute_cl_extreme_wing module

fastga.models.aerodynamics.components.compute_cl_pitch_rate module

fastga.models.aerodynamics.components.compute_cl_roll_rate module

fastga.models.aerodynamics.components.compute_cl_rudder module

fastga.models.aerodynamics.components.compute_cl_yaw_rate module

fastga.models.aerodynamics.components.compute_cm_alpha_dot module

fastga.models.aerodynamics.components.compute_cm_pitch_rate module

fastga.models.aerodynamics.components.compute_cn_aileron module

fastga.models.aerodynamics.components.compute_cn_beta module

fastga.models.aerodynamics.components.compute_cn_roll_rate module

fastga.models.aerodynamics.components.compute_cn_rudder module

fastga.models.aerodynamics.components.compute_cn_yaw_rate module

fastga.models.aerodynamics.components.compute_cy_beta module

fastga.models.aerodynamics.components.compute_cy_roll_rate module

fastga.models.aerodynamics.components.compute_cy_rudder module

fastga.models.aerodynamics.components.compute_cy_yaw_rate module

fastga.models.aerodynamics.components.compute_effective_efficiency_prop module

fastga.models.aerodynamics.components.compute_equilibrated_polar module

fastga.models.aerodynamics.components.compute_non_equilibrated_polar module

fastga.models.aerodynamics.components.compute_reynolds module

fastga.models.aerodynamics.components.compute_vn module

fastga.models.aerodynamics.components.elevator_aero module

fastga.models.aerodynamics.components.figure_digitization module

fastga.models.aerodynamics.components.high_lift_aero module

fastga.models.aerodynamics.components.hinge_moments_elevator module

fastga.models.aerodynamics.components.mach_interpolation module

Module contents

fastga.models.aerodynamics.external package

Subpackages

fastga.models.aerodynamics.external.openvsp package

Subpackages

fastga.models.aerodynamics.external.openvsp.openvsp3201 package

Module contents

Submodules

fastga.models.aerodynamics.external.openvsp.compute_aero module

fastga.models.aerodynamics.external.openvsp.compute_aero_slipstream module

fastga.models.aerodynamics.external.openvsp.openvsp module

Module contents

fastga.models.aerodynamics.external.propeller_code package

Submodules

fastga.models.aerodynamics.external.propeller_code.compute_propeller_aero module

fastga.models.aerodynamics.external.propeller_code.compute_propeller_coefficient_map module

fastga.models.aerodynamics.external.propeller_code.propeller_core module

Module contents

fastga.models.aerodynamics.external.vlm package

Submodules

fastga.models.aerodynamics.external.vlm.compute_aero module

fastga.models.aerodynamics.external.vlm.vlm module

Module contents

fastga.models.aerodynamics.external.xfoil package

Subpackages

fastga.models.aerodynamics.external.xfoil.xfoil699 package

Module contents

Submodules

fastga.models.aerodynamics.external.xfoil.xfoil_group module

fastga.models.aerodynamics.external.xfoil.xfoil_polar module

Computation of the airfoil aerodynamic properties using Xfoil.

class `fastga.models.aerodynamics.external.xfoil.xfoil_polar.`**`XfoilPolar`**`(**kwargs)`

Bases: `openmdao.components.external_code_comp.ExternalCodeComp`

Runs a polar computation with XFOIL and returns the 2D max lift coefficient.

Intialize the ExternalCodeComp component.

Parameters `**kwargs` (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`initialize()`
Perform any one-time initialization run at instantiation.

`setup()`
Declare inputs and outputs.

Available attributes: name pathname comm options

`check_config(logger)`
Perform optional error checks.

Parameters `logger` (*object*) – The object that manages logging output.

`compute(inputs, outputs)`
Run this component.

User should call this method from their overriden compute method.

Parameters

- `inputs` (*Vector*) – Unscaled, dimensional input variables read via inputs[key].
- `outputs` (*Vector*) – Unscaled, dimensional output variables read via outputs[key].

Module contents

Module for OpenMDAO-embedded XFOIL

Module contents

fastga.models.aerodynamics.unitary_tests package

Submodules

fastga.models.aerodynamics.unitary_tests.dummy_engines module

fastga.models.aerodynamics.unitary_tests.test_beechcraft_76 module

fastga.models.aerodynamics.unitary_tests.test_cirrus_sr22 module

fastga.models.aerodynamics.unitary_tests.test_daher_tbm900 module

fastga.models.aerodynamics.unitary_tests.test_functions module

fastga.models.aerodynamics.unitary_tests.test_partenavia_p68 module

Module contents

Submodules

fastga.models.aerodynamics.aero_center module

fastga.models.aerodynamics.aerodynamics module

fastga.models.aerodynamics.aerodynamics_high_speed module

fastga.models.aerodynamics.aerodynamics_low_speed module

fastga.models.aerodynamics.aerodynamics_stability_derivatives module

class fastga.models.aerodynamics.aerodynamics_stability_derivatives.**AerodynamicsStabilityDerivatives**(**/

Bases: openmdao.core.group.Group

Computes the aircraft aerodynamic derivatives that are not used for the sizing of the aircraft in cruise or low speed conditions or both depending of the user choice. It is meant to provide the aerodynamic derivatives necessary for the study of the stability of the aircraft. Can be run outside of the sizing loop for some time gain.

Set the solvers to nonlinear and linear block Gauss–Seidel by default.

Parameters ****kwargs** (*dict*) – dict of arguments available here and in all descendants of this Group.

initialize()

Perform any one-time initialization run at instantiation.

setup()

Build this group.

This method should be overidden by your Group's method. The reason for using this method to add subsystem is to save memory and setup time when using your Group while running under MPI. This avoids the creation of systems that will not be used in the current process.

You may call 'add_subsystem' to add systems to this group. You may also issue connections, and set the linear and nonlinear solvers for this group level. You cannot safely change anything on children systems; use the 'configure' method instead.

Available attributes: name pathname comm options

fastga.models.aerodynamics.compute_polar module

fastga.models.aerodynamics.constants module

fastga.models.aerodynamics.load_factor module

Module contents

fastga.models.geometry package

Subpackages

fastga.models.geometry.geom_components package

Subpackages

fastga.models.geometry.geom_components.fuselage package

Subpackages

fastga.models.geometry.geom_components.fuselage.components package

Submodules

fastga.models.geometry.geom_components.fuselage.components.compute_fuselage_depth module

fastga.models.geometry.geom_components.fuselage.components.compute_fuselage_dimensions module

fastga.models.geometry.geom_components.fuselage.components.compute_fuselage_volume module

fastga.models.geometry.geom_components.fuselage.components.compute_fuselage_wet_area module

Module contents

Submodules

fastga.models.geometry.geom_components.fuselage.compute_fuselage module

fastga.models.geometry.geom_components.fuselage.constants module

Module contents

fastga.models.geometry.geom_components.ht package

Subpackages

fastga.models.geometry.geom_components.ht.components package

Submodules

fastga.models.geometry.geom_components.ht.components.compute_ht_chords module

fastga.models.geometry.geom_components.ht.components.compute_ht_distance module

fastga.models.geometry.geom_components.ht.components.compute_ht_efficiency module

fastga.models.geometry.geom_components.ht.components.compute_ht_mac module

fastga.models.geometry.geom_components.ht.components.compute_ht_sweep module

fastga.models.geometry.geom_components.ht.components.compute_ht_volume_coefficient module

fastga.models.geometry.geom_components.ht.components.compute_ht_wet_area module

Module contents

Submodules

fastga.models.geometry.geom_components.ht.compute_horizontal_tail module

fastga.models.geometry.geom_components.ht.constants module

Module contents

fastga.models.geometry.geom_components.landing_gears package

Submodules

fastga.models.geometry.geom_components.landing_gears.compute_lg module

Module contents

fastga.models.geometry.geom_components.nacelle package

Submodules

fastga.models.geometry.geom_components.nacelle.compute_nacelle_dimension module

fastga.models.geometry.geom_components.nacelle.compute_nacelle_position module

Module contents

fastga.models.geometry.geom_components.propeller package

Subpackages

fastga.models.geometry.geom_components.propeller.components package

Submodules

fastga.models.geometry.geom_components.propeller.components.compute_propeller_installation_effect module

fastga.models.geometry.geom_components.propeller.components.compute_propeller_position module

Module contents

Submodules

fastga.models.geometry.geom_components.propeller.compute_propeller module

fastga.models.geometry.geom_components.propeller.constants module

Module contents

fastga.models.geometry.geom_components.vt package

Subpackages

fastga.models.geometry.geom_components.vt.components package

Submodules

fastga.models.geometry.geom_components.vt.components.compute_vt_chords module

fastga.models.geometry.geom_components.vt.components.compute_vt_distance module

fastga.models.geometry.geom_components.vt.components.compute_vt_mac module

fastga.models.geometry.geom_components.vt.components.compute_vt_sweep module

fastga.models.geometry.geom_components.vt.components.compute_vt_wet_area module

Module contents

Submodules

fastga.models.geometry.geom_components.vt.compute_vertical_tail module

fastga.models.geometry.geom_components.vt.constants module

Module contents

fastga.models.geometry.geom_components.wing package

Subpackages

fastga.models.geometry.geom_components.wing.components package

Submodules

fastga.models.geometry.geom_components.wing.components.compute_wing_b50 module

fastga.models.geometry.geom_components.wing.components.compute_wing_l1_l4 module

fastga.models.geometry.geom_components.wing.components.compute_wing_l2_l3 module

fastga.models.geometry.geom_components.wing.components.compute_wing_mac module

fastga.models.geometry.geom_components.wing.components.compute_wing_sweep module

fastga.models.geometry.geom_components.wing.components.compute_wing_toc module

fastga.models.geometry.geom_components.wing.components.compute_wing_wet_area module

fastga.models.geometry.geom_components.wing.components.compute_wing_x module

fastga.models.geometry.geom_components.wing.components.compute_wing_x_absolute module

fastga.models.geometry.geom_components.wing.components.compute_wing_y module

fastga.models.geometry.geom_components.wing.components.compute_wing_z module

Module contents

Submodules

fastga.models.geometry.geom_components.wing.compute_wing module

fastga.models.geometry.geom_components.wing.constants module

Module contents

fastga.models.geometry.geom_components.wing_tank package

Submodules

fastga.models.geometry.geom_components.wing_tank.compute_mfw_advanced module

fastga.models.geometry.geom_components.wing_tank.compute_mfw_simple module

Module contents

Submodules

fastga.models.geometry.geom_components.compute_total_area module

Module contents

fastga.models.geometry.profiles package

Submodules

fastga.models.geometry.profiles.get_profile module

Airfoil reshape function.

`fastga.models.geometry.profiles.get_profile.`**`get_profile`**(*airfoil_folder_path: Optional[str] = None, file_name: Optional[str] = None, thickness_ratio=None, chord_length=None*) → *fastga.models.geometry.profiles.profile.Profile*

Reads profile from indicated resource file and returns it after resize

Parameters

- **`file_name`** – name of resource (ex: "naca23012.af")
- **`thickness_ratio`** –
- **`chord_length`** –

Returns Profile object.

`fastga.models.geometry.profiles.get_profile.`**`genfromtxt`**(*file_name: Optional[str] = None*) → pandas.core.frame.DataFrame

fastga.models.geometry.profiles.profile module

Management of 2D wing profiles.

`class` `fastga.models.geometry.profiles.profile.`**`Coordinates2D`**(*x, y*)

Bases: `tuple`

Create new instance of Coordinates2D(x, y)

`property x`

Alias for field number 0

`property y`

Alias for field number 1

`class` `fastga.models.geometry.profiles.profile.`**`Profile`**(*chord_length: float = 0.0*)

Bases: `object`

Class for managing 2D wing profiles.

`property thickness_ratio: float`

thickness-to-chord ratio

set_points(*x: Sequence*, *z: Sequence*, *keep_chord_length: bool = True*, *keep_relative_thickness: bool = True*)

Sets points of the 2D profile.

Provided points are expected to be in order around the profile (clockwise or anti-clockwise).

Parameters

- **x** – in meters
- **z** – in meters
- **keep_relative_thickness** –
- **keep_chord_length** –

get_mean_line() → pandas.core.frame.DataFrame

Point set of mean line of the profile.

DataFrame keys are 'x' and 'z', given in meters.

get_relative_thickness() → pandas.core.frame.DataFrame

Point set of relative thickness of the profile.

DataFrame keys are 'x' and 'thickness' and are relative to chord_length. 'x' is form 0. to 1.

get_upper_side() → pandas.core.frame.DataFrame

Point set of upper side of the profile.

DataFrame keys are 'x' and 'z', given in meters.

get_lower_side() → pandas.core.frame.DataFrame

Point set of lower side of the profile.

DataFrame keys are 'x' and 'z', given in meters.

get_sides() → pandas.core.frame.DataFrame

Point set of the whole profile

Points are given from trailing edge to trailing edge, starting by upper side.

Module contents

Different functions available.

fastga.models.geometry.unitary_tests package

Submodules

fastga.models.geometry.unitary_tests.dummy_engines module

Test module for geometry functions of cg components.

class fastga.models.geometry.unitary_tests.dummy_engines.**DummyEngineBE76**(*max_power: float*, *design_altitude_propeller: float*, *fuel_type: float*, *strokes_nb: float*, *prop_layout: float*)

Bases: *fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion*

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float

Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float

Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.geometry.unitary_tests.dummy_engines.**DummyEngineWrapperBE76**

Bases: fastoad.model_base.propulsion.IOMPropulsionWrapper

setup(*component: openmdao.core.component.Component*)

Defines the needed OpenMDAO inputs for propulsion instantiation as done in get_model()

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion

This method defines the used IPropulsion subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

class fastga.models.geometry.unitary_tests.dummy_engines.**DummyEngineSR22**(*max_power: float, design_altitude_propeller: float, fuel_type: float, strokes_nb: float, prop_layout: float*)

Bases: fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See FlightPoint for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

thrust_is_regulated tells if a flight point should be computed using thrust_rate (when False) or thrust (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if thrust_is_regulated is not defined, the considered input will be the defined one between thrust_rate and thrust (if both are provided, thrust_rate will be used)
- if thrust_is_regulated is True or False (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, thrust_is_regulated is a sequence or array, thrust_rate and thrust should be provided and have the same shape as thrust_is_regulated:code:. The method will consider for each element which input will be used according to thrust_is_regulated.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float
Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)
Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)
Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.geometry.unitary_tests.dummy_engines.**DummyEngineWrapperSR22**
Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in `get_model()`

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

class fastga.models.geometry.unitary_tests.dummy_engines.**DummyEngineTBM900**(*max_power: float*, *design_altitude_propeller: float*, *prop_layout: float*)
Bases: `fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`

Dummy engine model returning nacelle dimension

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() *-> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)*

Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float

Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.geometry.unitary_tests.dummy_engines.**DummyEngineWrapperTBM900**
Bases: fastoad.model_base.propulsion.IOMPropulsionWrapper

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in *get_model()*

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used IPropulsion subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

fastga.models.geometry.unitary_tests.test_beechcraft_76 module

fastga.models.geometry.unitary_tests.test_cirrus_sr22 module

fastga.models.geometry.unitary_tests.test_daher_tbm900 module

fastga.models.geometry.unitary_tests.test_partenavia_p68 module

Module contents

Submodules

fastga.models.geometry.constants module

Constants for the geometry submodels.

fastga.models.geometry.geometry module

Module contents

Estimation of global geometry components.

fastga.models.handling_qualities package

Subpackages

fastga.models.handling_qualities.tail_sizing package

Submodules

fastga.models.handling_qualities.tail_sizing.compute_balked_landing_limit module

fastga.models.handling_qualities.tail_sizing.compute_to_rotation_limit module

fastga.models.handling_qualities.tail_sizing.constants module

Constants for the tail area update submodels.

fastga.models.handling_qualities.tail_sizing.update_ht_area module

fastga.models.handling_qualities.tail_sizing.update_tail_areas module

Computation of tail areas w.r.t. HQ criteria.

`class fastga.models.handling_qualities.tail_sizing.update_tail_areas.`**`UpdateTailAreas`**`(`***`*kwargs`*`)`

Bases: `openmdao.core.group.Group`

Computes areas of vertical and horizontal tail.

- Horizontal tail area is computed so it can balance pitching moment of aircraft at rotation speed.
- Vertical tail area is computed so aircraft can have the Cnbeta in cruise conditions and (for bi-motor) maintain trajectory with failed engine @ 5000ft.

Set the solvers to nonlinear and linear block Gauss–Seidel by default.

Parameters **`**kwargs`** (*`dict`*) – dict of arguments available here and in all descendants of this Group.

`initialize``()`

Perform any one-time initialization run at instantiation.

`setup``()`

Build this group.

This method should be overidden by your Group's method. The reason for using this method to add subsystem is to save memory and setup time when using your Group while running under MPI. This avoids the creation of systems that will not be used in the current process.

You may call 'add_subsystem' to add systems to this group. You may also issue connections, and set the linear and nonlinear solvers for this group level. You cannot safely change anything on children systems; use the 'configure' method instead.

Available attributes: name pathname comm options

fastga.models.handling_qualities.tail_sizing.update_vt_area module

Module contents

fastga.models.handling_qualities.unitary_tests package

Submodules

fastga.models.handling_qualities.unitary_tests.dummy_engines module

Test module for geometry functions of cg components.

class fastga.models.handling_qualities.unitary_tests.dummy_engines.**DummyEngineBE76**(*max_power: float, design_altitude_propeller: float, fuel_type: float, strokes_nb: float, prop_layout: float*)

Bases: `fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float
Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)
Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)
Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.handling_qualities.unitary_tests.dummy_engines.**DummyEngineWrapperBE76**
Bases: fastoad.model_base.propulsion.IOMPropulsionWrapper

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in *get_model()*

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static **get_model**(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used IPropulsion subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

class fastga.models.handling_qualities.unitary_tests.dummy_engines.**DummyEngineSR22**
Bases: *fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion*

Dummy engine model returning nacelle dimensions height-width-length-wet_area.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)
Computes Specific Fuel Consumption according to provided conditions.

See FlightPoint for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

thrust_is_regulated tells if a flight point should be computed using thrust_rate (when False) or thrust (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if thrust_is_regulated is not defined, the considered input will be the defined one between thrust_rate and thrust (if both are provided, thrust_rate will be used)
- if thrust_is_regulated is True or False (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, thrust_is_regulated is a sequence or array, thrust_rate and thrust should be provided and have the same shape as thrust_is_regulated:code:. The method will consider for each element which input will be used according to thrust_is_regulated.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float

Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_sl_thrust() → float

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float

Computes max available power on one engine.

Returns the maximum available power in W

`class fastga.models.handling_qualities.unitary_tests.dummy_engines.DummyEngineWrapperSR22`
Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in `get_model()`

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

`static get_model`(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

`class fastga.models.handling_qualities.unitary_tests.dummy_engines.DummyEngineTBM900`
Bases: *`fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`*

Dummy engine model returning nacelle dimensions height-width-length-wet_area.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)
Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float
Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)
Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)
Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_sl_thrust() → float

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class
`fastga.models.handling_qualities.unitary_tests.dummy_engines.`**DummyEngineWrapperTBM900**
Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in *get_model()*

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

fastga.models.handling_qualities.unitary_tests.test_beechcraft_76 module

fastga.models.handling_qualities.unitary_tests.test_cirrus_sr22 module

fastga.models.handling_qualities.unitary_tests.test_daher_tbm900 module

Module contents

Submodules

fastga.models.handling_qualities.compute_static_margin module

fastga.models.handling_qualities.handling_qualities module

Module contents

fastga.models.load_analysis package

Subpackages

fastga.models.load_analysis.unitary_tests package

Submodules

fastga.models.load_analysis.unitary_tests.dummy_engines module

fastga.models.load_analysis.unitary_tests.test_beechcraft_76 module

fastga.models.load_analysis.unitary_tests.test_cirrus_sr22 module

fastga.models.load_analysis.unitary_tests.test_daher_tbm900 module

Module contents

fastga.models.load_analysis.wing package

Submodules

fastga.models.load_analysis.wing.aerodynamic_loads module

fastga.models.load_analysis.wing.aerostructural_loads module

fastga.models.load_analysis.wing.constants module

Constants for wing loads models.

fastga.models.load_analysis.wing.loads module

class fastga.models.load_analysis.wing.loads.**WingLoads**(***kwargs*)
Bases: openmdao.core.group.Group

Set the solvers to nonlinear and linear block Gauss–Seidel by default.

Parameters ****kwargs** (*dict*) – dict of arguments available here and in all descendants of this Group.

setup()
Build this group.

This method should be overidden by your Group's method. The reason for using this method to add subsystem is to save memory and setup time when using your Group while running under MPI. This avoids the creation of systems that will not be used in the current process.

You may call 'add_subsystem' to add systems to this group. You may also issue connections, and set the linear and nonlinear solvers for this group level. You cannot safely change anything on children systems; use the 'configure' method instead.

Available attributes: name pathname comm options

fastga.models.load_analysis.wing.structural_loads module

Module contents

Module contents

fastga.models.loops package

Subpackages

fastga.models.loops.unitary_tests package

Submodules

fastga.models.loops.unitary_tests.test_loops module

Module contents

fastga.models.loops.wing_area_component package

Submodules

fastga.models.loops.wing_area_component.update_wing_area module

Computation of wing area depending on which criteria between geometric and aerodynamic is the most constraining.

class fastga.models.loops.wing_area_component.update_wing_area.**UpdateWingArea**(***kwargs*)
Bases: openmdao.core.explicitcomponent.ExplicitComponent

Computes needed wing area to:

- have enough lift at required approach speed
- be able to load enough fuel to achieve the sizing mission.

Store some bound methods so we can detect runtime overrides.

Parameters `**kwargs` (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`setup()`
Declare inputs and outputs.

Available attributes: name pathname comm options

`compute`(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials`(*inputs*, *partials*, *discrete_inputs=None*)
Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **partials** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.

fastga.models.loops.wing_area_component.wing_area_cl_equilibrium module

fastga.models.loops.wing_area_component.wing_area_loop_cl_simple module

fastga.models.loops.wing_area_component.wing_area_loop_geom_adv module

fastga.models.loops.wing_area_component.wing_area_loop_geom_simple module

Module contents

Submodules

fastga.models.loops.constants module

Constants for the loop computations.

fastga.models.loops.update_wing_area_group module

Computation of wing area and wing area related constraints.

class `fastga.models.loops.update_wing_area_group.`**`UpdateWingAreaGroup`**(***kwargs*)

Bases: `openmdao.core.group.Group`

Groups that gather the computation of the updated wing area, chooses the biggest one and computes the constraints based on the new wing area.

Set the solvers to nonlinear and linear block Gauss–Seidel by default.

Parameters ****kwargs** (*dict*) – dict of arguments available here and in all descendants of this Group.

`initialize`()

Perform any one-time initialization run at instantiation.

`setup`()

Adding the update groups, the selection of the maximum and the constraints.

fastga.models.loops.update_wing_position module

Computation of wing position.

class `fastga.models.loops.update_wing_position.`**`UpdateWingPosition`**(***kwargs*)

Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`setup`()

Declare inputs and outputs.

Available attributes: name pathname comm options

`compute`(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials`(*inputs*, *partials*, *discrete_inputs=None*)

Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **partials** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.

Module contents

fastga.models.performances package

Subpackages

fastga.models.performances.mission package

Subpackages

fastga.models.performances.mission.mission_components package

Submodules

fastga.models.performances.mission.mission_components.climb module

fastga.models.performances.mission.mission_components.cruise module

fastga.models.performances.mission.mission_components.descent module

fastga.models.performances.mission.mission_components.reserve module

fastga.models.performances.mission.mission_components.taxi module

Module contents

Submodules

fastga.models.performances.mission.constants module

Constants for the geometry submodels.

fastga.models.performances.mission.dynamic_equilibrium module

fastga.models.performances.mission.mission module

fastga.models.performances.mission.mission_builder_prep module

fastga.models.performances.mission.takeoff module

Module contents

fastga.models.performances.mission_vector package

Subpackages

fastga.models.performances.mission_vector.initialization package

Submodules

fastga.models.performances.mission_vector.initialization.initialize module

fastga.models.performances.mission_vector.initialization.initialize_airspeed module

fastga.models.performances.mission_vector.initialization.initialize_airspeed_derivatives module

fastga.models.performances.mission_vector.initialization.initialize_altitude module

fastga.models.performances.mission_vector.initialization.initialize_cg module

class `fastga.models.performances.mission_vector.initialization.initialize_cg.`**`InitializeCoG`**(***kwargs*)

Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes the center of gravity at each time step.

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

initialize()

Perform any one-time initialization run at instantiation.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

fastga.models.performances.mission_vector.initialization.initialize_gamma module

fastga.models.performances.mission_vector.initialization.initialize_horizontal_speed module

class `fastga.models.performances.mission_vector.initialization.initialize_horizontal_speed.`**`InitializeHo`**

Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Initializes the horizontal airspeed at each time step.

Store some bound methods so we can detect runtime overrides.

Parameters **`**kwargs`** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`initialize()`
Perform any one-time initialization run at instantiation.

`setup()`
Declare inputs and outputs.

Available attributes: name pathname comm options

`setup_partials()`
Declare partials.

This is meant to be overridden by component classes. All partials should be declared here since this is called after all size/shape information is known for all variables.

`compute`(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **`inputs`** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **`outputs`** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **`discrete_inputs`** (*dict or None*) – If not None, dict containing discrete input values.
- **`discrete_outputs`** (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials`(*inputs*, *partials*, *discrete_inputs=None*)
Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **`inputs`** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **`partials`** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **`discrete_inputs`** (*dict or None*) – If not None, dict containing discrete input values.

fastga.models.performances.mission_vector.initialization.initialize_time_and_distance module

Module contents

fastga.models.performances.mission_vector.mission package

Submodules

fastga.models.performances.mission_vector.mission.compute_time_step module

class `fastga.models.performances.mission_vector.mission.compute_time_step.`**`ComputeTimeStep`**(***kwargs*)
Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes the time step size for the energy consumption later.

Store some bound methods so we can detect runtime overrides.

Parameters `**kwargs` (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`initialize()`
Perform any one-time initialization run at instantiation.

`setup()`
Declare inputs and outputs.

Available attributes: name pathname comm options

`setup_partials()`
Declare partials.

This is meant to be overridden by component classes. All partials should be declared here since this is called after all size/shape information is known for all variables.

`compute`(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- `inputs` (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- `outputs` (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- `discrete_inputs` (*dict or None*) – If not None, dict containing discrete input values.
- `discrete_outputs` (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials`(*inputs*, *partials*, *discrete_inputs=None*)
Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- `inputs` (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- `partials` (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- `discrete_inputs` (*dict or None*) – If not None, dict containing discrete input values.

fastga.models.performances.mission_vector.mission.dep_equilibrium module

fastga.models.performances.mission_vector.mission.energy_consumption_preparation module

fastga.models.performances.mission_vector.mission.equilibrium module

fastga.models.performances.mission_vector.mission.mission_core module

fastga.models.performances.mission_vector.mission.no_dep_effect module

fastga.models.performances.mission_vector.mission.performance_per_phase module

fastga.models.performances.mission_vector.mission.propulsion_via_id module

fastga.models.performances.mission_vector.mission.reserve_energy module

class `fastga.models.performances.mission_vector.mission.reserve_energy.`**`ReserveEnergy`**(***kwargs*)
Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes the fuel consumed during the reserve phase.

Store some bound methods so we can detect runtime overrides.

Parameters **`**kwargs`** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`setup`()
Declare inputs and outputs.

Available attributes: name pathname comm options

`setup_partials`()
Declare partials.

This is meant to be overridden by component classes. All partials should be declared here since this is called after all size/shape information is known for all variables.

`compute`(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **`inputs`** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **`outputs`** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **`discrete_inputs`** (*dict or None*) – If not None, dict containing discrete input values.
- **`discrete_outputs`** (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials`(*inputs*, *partials*, *discrete_inputs=None*)
Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **`inputs`** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **`partials`** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **`discrete_inputs`** (*dict or None*) – If not None, dict containing discrete input values.

fastga.models.performances.mission_vector.mission.sizing_energy module

class `fastga.models.performances.mission_vector.mission.sizing_energy.`**`SizingEnergy`**(***kwargs*)
Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes the fuel consumed during the whole sizing mission.

Store some bound methods so we can detect runtime overrides.

Parameters **`**kwargs`** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`setup`()
Declare inputs and outputs.

Available attributes: name pathname comm options

`setup_partials()`
Declare partials.

This is meant to be overridden by component classes. All partials should be declared here since this is called after all size/shape information is known for all variables.

`compute(`*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*`)`
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials(`*inputs*, *partials*, *discrete_inputs=None*`)`
Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **partials** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.

fastga.models.performances.mission_vector.mission.thrust_taxi module

fastga.models.performances.mission_vector.mission.update_mass module

class `fastga.models.performances.mission_vector.mission.update_mass.UpdateMass(`***kwargs*`)`
Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Update mass for next iteration.

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`initialize()`
Perform any one-time initialization run at instantiation.

`setup()`
Declare inputs and outputs.

Available attributes: name pathname comm options

`setup_partials()`
Declare partials.

This is meant to be overridden by component classes. All partials should be declared here since this is called after all size/shape information is known for all variables.

`compute(`*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*`)`
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

compute_partials(*inputs*, *partials*, *discrete_inputs=None*)

Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **partials** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.

Module contents

Submodules

fastga.models.performances.mission_vector.constants module

fastga.models.performances.mission_vector.full_mission module

fastga.models.performances.mission_vector.mission_vector module

fastga.models.performances.mission_vector.to_csv module

Module contents

fastga.models.performances.payload_range package

Submodules

fastga.models.performances.payload_range.payload_range module

Module contents

fastga.models.performances.unitary_tests package

Submodules

fastga.models.performances.unitary_tests.dummy_engines module

fastga.models.performances.unitary_tests.test_beechcraft_76 module

fastga.models.performances.unitary_tests.test_cirrus_sr22 module

fastga.models.performances.unitary_tests.test_daher_tbm900 module

Module contents

Module contents

fastga.models.propulsion package

Subpackages

fastga.models.propulsion.fuel_propulsion package

Subpackages

fastga.models.propulsion.fuel_propulsion.basicIC_engine package

Subpackages

fastga.models.propulsion.fuel_propulsion.basicIC_engine.unitary_tests package

Submodules

fastga.models.propulsion.fuel_propulsion.basicIC_engine.unitary_tests.test_basicIC_engine module

fastga.models.propulsion.fuel_propulsion.basicIC_engine.unitary_tests.test_openmdao_engine module

Module contents

Submodules

fastga.models.propulsion.fuel_propulsion.basicIC_engine.basicIC_engine module

fastga.models.propulsion.fuel_propulsion.basicIC_engine.exceptions module

Exceptions for basicIC_engine package.

exception `fastga.models.propulsion.fuel_propulsion.basicIC_engine.exceptions.` **`FastBasicICEngineInconsistentInputParametersError`**

Bases: `Exception`

Raised when provided parameter combination is incorrect.

fastga.models.propulsion.fuel_propulsion.basicIC_engine.openmdao module

Module contents

Provides a parametric model for turbofan:

- as a pure Python
- as OpenMDAO modules.

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop package

Subpackages

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.unitary_tests package

Submodules

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.unitary_tests.test_basicTP_engine module

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.unitary_tests.test_openmdao_engine module

Module contents

Submodules

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.basicTP_engine module

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.exceptions module

Exceptions for basicIC_engine package.

exception `fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.exceptions.`**`FastBasicICEngineInconsistentInputParametersError`**

Bases: `Exception`

Raised when provided parameter combination is incorrect.

exception `fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.exceptions.`**`FastBasicTPEngineImpossibleTurbopropGeometry`**

Bases: `Exception`

Raised when the geometry of the turboprop can't be computed.

exception `fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.exceptions.`**`FastBasicTPEngineUnknownLimit`**

Bases: `Exception`

Raised when an unknown limit given to the turboprop.

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop.openmdao module

Module contents

Provides a parametric model for turboprop as a pure Python and as OpenMDAO modules.

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map package

Subpackages

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.unitary_tests package

Submodules

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.unitary_tests.test_basicTP_engine_mapped module

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.unitary_tests.test_daher_tbm900 module

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.unitary_tests.test_openmdao_engine module

Module contents

Submodules

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.basicTP_engine_constructor module

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.basicTP_engine_mapped module

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.exceptions module

Exceptions for basicIC_engine package.

exception `fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.exceptions.`**`FastBasicICEngineInconsistentInputParametersError`**

Bases: `Exception`

Raised when provided parameter combination is incorrect.

fastga.models.propulsion.fuel_propulsion.basicTurbo_prop_map.openmdao module

Module contents

Provides a parametric model for turboprop as a pure Python and as OpenMDAO modules.

Submodules

fastga.models.propulsion.fuel_propulsion.base module

Base classes for fuel-consuming propulsion models.

class fastga.models.propulsion.fuel_propulsion.base.**AbstractFuelPropulsion**

Bases: *fastga.models.propulsion.propulsion.IPropulsionCS23*, abc.ABC

Propulsion model that consume any fuel should inherit from this one.

In inheritors, compute_flight_points() is expected to define "sfc" and "thrust" in computed FlightPoint instances.

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float

Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

abstract compute_dimensions()

Computes propulsion sub-components dimensions.

class fastga.models.propulsion.fuel_propulsion.base.**FuelEngineSet**(*engine: fastga.models.propulsion.propulsion.IPropulsion*(*engine_count*)

Bases: *fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion*

Class for modelling an assembly of identical fuel engines.

Thrust is supposed equally distributed among them.

Parameters

- **engine** – the engine model
- **engine_count** –

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See FlightPoint for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **`flight_points`** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

`compute_max_power`(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes max available power on one engine.

Returns the maximum available power in W

`compute_weight`()

Computes total propulsion mass.

Returns the total uninstalled mass in kg

`compute_dimensions`()

Computes propulsion sub-components dimensions.

`compute_drag`(*mach*, *unit_reynolds*, *wing_mac*)

Computes nacelle drag force for out of fuselage engine.

Parameters

- **`mach`** – mach at which drag should be calculated
- **`unit_reynolds`** – unitary Reynolds for calculation
- **`wing_mac`** – wing MAC length in m

Returns drag force cd0*wing_area

Module contents

Submodules

fastga.models.propulsion.dict module

class `fastga.models.propulsion.dict.`**`DynamicAttributeDict`**(**args*, ***kwargs*)

Bases: `dict`

A dictionary class where keys can also be used as attributes.

The keys that can be used as attributes are defined using decorators *`AddKeyAttribute`* or `SetKeyAttributes`.

They can also be used as keyword arguments when instantiating this class.

Note: Using this class as a dict is useful when instantiating another dict or a pandas DataFrame, or instantiating from them. Direct interaction with DynamicAttributeDict instance should be done through attributes.

Example:

```
>>> @AddKeyAttributes({"foo": 0.0, "bar": None, "baz": 42.0})
... class MyDict(DynamicAttributeDict):
...     pass
...

>>> d = MyDict(foo=5, bar="aa")
>>> d.foo
5
>>> d.bar
'aa'
>>> d.baz  # returns the default value
42.0
>>> d["foo"] = 10.0  # can still be used as a dict
>>> d.foo  # but change are propagated to/from the matching attribute
10.0
>>> d.foo = np.nan  # setting None or numpy.nan returns to default value
>>> d["foo"]
0.0
>>> d.foo # But the attribute will now return the default value
0.0
>>> d.bar = None  # If default value is None, setting None or numpy.nan deletes the␣
↪key.
>>> # d["bar"]  #would trigger a key error
>>> d.bar # But the attribute will return None
```

Parameters

- **args** – a dict-like object where all keys are contained in `attribute_keys`
- **kwargs** – argument keywords must be names contained in `attribute_keys`

class `fastga.models.propulsion.dict.`**AddKeyAttribute**(*attr_name*, *default_value=None*, *doc=None*)

Bases: `object`

A decorator for a dict class that adds a property for accessing the matching dict item.

The getter and the setter of the property are defined. Setting None or np.nan when setting the property will delete the dict key, so that next calls to the getter will return default_value.

Calling AddKeyAttribute for an already defined key will redefine the default value.

The "attribute_keys" property is created in decorated class for returning the list of attributes that have been defined by AddKeyAttribute or by *AddKeyAttributes*.

Parameters

- **attr_name** – the dict key that will be paired to a property
- **default_value** – the default value that will be returned if dict has not the attr_name as key

class `fastga.models.propulsion.dict.`**AddKeyAttributes**(*attribute_definition: Union[dict, Iterable[str]]*)

Bases: `object`

A decorator for a dict class that adds properties for accessing the matching dict item.

This class simply does several call of *AddKeyAttribute*.

Parameters **attribute_definition** – the list of keys that will be attributes. If it is a dictionary, the values are the associated default values. If it is a list or a set, default values will be None.

fastga.models.propulsion.propulsion module

Base module for propulsion models.

class fastga.models.propulsion.propulsion.**IPropulsionCS23**

Bases: fastoad.model_base.propulsion.IPropulsion

Interface that should be implemented by propulsion models.

abstract **compute_weight**() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

abstract **compute_max_power**(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes max available power on one engine.

Returns the maximum available power in W

abstract **compute_dimensions**() *-> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)*

Computes propulsion dimensions.

Returns (height, width, length, wet area) of nacelle in m or m^2

abstract **compute_drag**(*mach: Union[float, numpy.array]*, *unit_reynolds: Union[float, numpy.array]*, *wing_mac: float*) → Union[float, numpy.array]

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

class fastga.models.propulsion.propulsion.**BaseOMPropulsionComponent**(***kwargs*)

Bases: openmdao.core.explicitcomponent.ExplicitComponent, abc.ABC

Base class for OpenMDAO wrapping of subclasses of IEngineForOpenMDAO.

Classes that implements this interface should add their own inputs in setup() and implement *get_wrapper()*.

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

initialize()

Perform any one-time initialization run at instantiation.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)
Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

abstract static get_wrapper() → fastoad.model_base.propulsion.IOMPropulsionWrapper
This method defines the used `IOMPropulsionWrapper` instance.

Returns an instance of OpenMDAO wrapper for propulsion model

Module contents

Package for propulsion modules.

fastga.models.weight package

Subpackages

fastga.models.weight.cg package

Subpackages

fastga.models.weight.cg.cg_components package

Subpackages

fastga.models.weight.cg.cg_components.a_airframe package

Submodules

fastga.models.weight.cg.cg_components.a_airframe.a1_wing_cg module

fastga.models.weight.cg.cg_components.a_airframe.a2_fuselage_cg module

fastga.models.weight.cg.cg_components.a_airframe.a3_tail_cg module

fastga.models.weight.cg.cg_components.a_airframe.a4_flight_control_cg module

fastga.models.weight.cg.cg_components.a_airframe.a5_landing_gear_cg module

Module contents

fastga.models.weight.cg.cg_components.b_propulsion package

Submodules

fastga.models.weight.cg.cg_components.b_propulsion.b1_engine_cg module

Estimation of engine(s) center of gravity.

class fastga.models.weight.cg.cg_components.b_propulsion.b1_engine_cg.**ComputeEngineCG**(***kwargs*)

Bases: openmdao.core.explicitcomponent.ExplicitComponent

Engine(s) center of gravity estimation

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

fastga.models.weight.cg.cg_components.b_propulsion.b2_fuel_lines_cg module

Estimation of fuel lines center of gravity.

class fastga.models.weight.cg.cg_components.b_propulsion.b2_fuel_lines_cg.**ComputeFuelLinesCG**(***kwargs*)

Bases: openmdao.core.explicitcomponent.ExplicitComponent

Fuel lines center of gravity estimation

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]

- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

fastga.models.weight.cg.cg_components.b_propulsion.b3_tank_cg module

fastga.models.weight.cg.cg_components.b_propulsion.b_cg module

Module contents

fastga.models.weight.cg.cg_components.c_systems package

Submodules

fastga.models.weight.cg.cg_components.c_systems.c1_power_systems_cg module

fastga.models.weight.cg.cg_components.c_systems.c2_life_support_systems_cg module

fastga.models.weight.cg.cg_components.c_systems.c3_navigation_systems_cg module

fastga.models.weight.cg.cg_components.c_systems.c4_recording_systems_cg module

Module contents

fastga.models.weight.cg.cg_components.d_furniture package

Submodules

fastga.models.weight.cg.cg_components.d_furniture.d2_passenger_seats_cg module

Module contents

Submodules

fastga.models.weight.cg.cg_components.constants module

Constants for the CoG submodels.

fastga.models.weight.cg.cg_components.global_cg module

fastga.models.weight.cg.cg_components.loadcase module

fastga.models.weight.cg.cg_components.max_cg_ratio module

Estimation of maximum center of gravity ratio.

class fastga.models.weight.cg.cg_components.max_cg_ratio.**ComputeMaxMinCGRatio**(***kwargs*)

Bases: openmdao.core.explicitcomponent.ExplicitComponent

Extrema center of gravity ratio estimation

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

fastga.models.weight.cg.cg_components.payload module

fastga.models.weight.cg.cg_components.ratio_aft module

Module contents

fastga.models.weight.cg.unitary_tests package

Submodules

fastga.models.weight.cg.unitary_tests.dummy_engines module

Test module for geometry functions of cg components.

class fastga.models.weight.cg.unitary_tests.dummy_engines.**DummyEngineBE76**(*max_power: float, design_altitude_propeller: float, fuel_type: float, strokes_nb: float, prop_layout: float*)

Bases: `fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

compute_drag(*mach: Union[float, numpy.array], unit_reynolds: Union[float, numpy.array], wing_mac: float*) → Union[float, numpy.array]

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.weight.cg.unitary_tests.dummy_engines.**DummyEngineWrapperBE76**
Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in `get_model()`

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static **get_model**(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

class fastga.models.weight.cg.unitary_tests.dummy_engines.**DummyEngineSR22**(*max_power: float*, *design_altitude_propeller: float*, *fuel_type: float*, *strokes_nb: float*, *prop_layout: float*)
Bases: `fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)
Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

compute_drag(*mach: Union[float, numpy.array]*, *unit_reynolds: Union[float, numpy.array]*, *wing_mac: float*) → Union[float, numpy.array]

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float

Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float

Computes max available power on one engine.

Returns the maximum available power in W

class `fastga.models.weight.cg.unitary_tests.dummy_engines.`**DummyEngineWrapperSR22**

Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)

Defines the needed OpenMDAO inputs for propulsion instantiation as done in `get_model()`

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **`component`** –

`static get_model`(*inputs*) → fastoad.model_base.propulsion.IPropulsion

This method defines the used `IPropulsion` subclass instance.

Parameters **`inputs`** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

`class fastga.models.weight.cg.unitary_tests.dummy_engines.`**`DummyEngineTBM900`**(*fuel_type: float*, *prop_layout: float*)

Bases: *`fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`*

Dummy engine model returning nacelle aerodynamic drag force.

`compute_flight_points`(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **`flight_points`** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

`compute_weight`() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

`compute_dimensions`() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

`compute_drag`(*mach: Union[float, numpy.array], unit_reynolds: Union[float, numpy.array], wing_mac: float*) → Union[float, numpy.array]

Computes nacelle drag force for out of fuselage engine.

Parameters

- **`mach`** – mach at which drag should be calculated

- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.weight.cg.unitary_tests.dummy_engines.**DummyEngineWrapperTBM900**
Bases: fastoad.model_base.propulsion.IOMPropulsionWrapper

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in *get_model()*

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters component –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used IPropulsion subclass instance.

Parameters inputs – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

fastga.models.weight.cg.unitary_tests.test_beechcraft_76 module

fastga.models.weight.cg.unitary_tests.test_cirrus_sr22 module

fastga.models.weight.cg.unitary_tests.test_daher_tbm900 module

Module contents

Submodules

fastga.models.weight.cg.cg module

fastga.models.weight.cg.cg_variation module

class fastga.models.weight.cg.cg_variation.**InFlightCGVariation**(***kwargs*)

Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes the coefficient necessary to the calculation of the cg position at any point of the DESIGN flight.

Store some bound methods so we can detect runtime overrides.

Parameters ****kwargs** (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

Module contents

fastga.models.weight.mass_breakdown package

Subpackages

fastga.models.weight.mass_breakdown.a_airframe package

Subpackages

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components package

Submodules

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_additional_bending_material_ module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_additional_bending_material_ module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_bulkhead_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_cone_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_doors_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_engine_support_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_floor_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_insulation_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_nlg_hatch_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_shell_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_windows_mass module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.compute_wing_fuselage_connection_m module

fastga.models.weight.mass_breakdown.a_airframe.fuselage_components.update_fuselage_mass module

Module contents

fastga.models.weight.mass_breakdown.a_airframe.wing_components package

Submodules

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_lower_flange module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_misc_mass module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_primary_mass module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_ribs_mass module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_secondary_mass module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_skin_mass module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_upper_flange module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.compute_web_mass module

fastga.models.weight.mass_breakdown.a_airframe.wing_components.update_wing_mass module

Module contents

Submodules

fastga.models.weight.mass_breakdown.a_airframe.a1_wing_weight module

fastga.models.weight.mass_breakdown.a_airframe.a1_wing_weight_analytical module

fastga.models.weight.mass_breakdown.a_airframe.a2_fuselage_weight module

fastga.models.weight.mass_breakdown.a_airframe.a2_fuselage_weight_analytical module

fastga.models.weight.mass_breakdown.a_airframe.a3_tail_weight module

fastga.models.weight.mass_breakdown.a_airframe.a4_flight_control_weight module

fastga.models.weight.mass_breakdown.a_airframe.a5_landing_gear_weight module

fastga.models.weight.mass_breakdown.a_airframe.a7_paint_weight module

fastga.models.weight.mass_breakdown.a_airframe.constants module

Constants for airframe mass submodels.

fastga.models.weight.mass_breakdown.a_airframe.sum module

Module contents

fastga.models.weight.mass_breakdown.b_propulsion package

Submodules

fastga.models.weight.mass_breakdown.b_propulsion.b1_2_oil_weight module

Estimation of engine and associated component weight.

class fastga.models.weight.mass_breakdown.b_propulsion.b1_2_oil_weight.**ComputeOilWeight**(***kwargs*)

Bases: openmdao.core.explicitcomponent.ExplicitComponent

Weight estimation for motor oil.

Based on a statistical analysis. See [WHM17].

Store some bound methods so we can detect runtime overrides.

Parameters **kwargs (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

initialize()

Perform any one-time initialization run at instantiation.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

fastga.models.weight.mass_breakdown.b_propulsion.b1_engine_weight module

fastga.models.weight.mass_breakdown.b_propulsion.b2_fuel_lines_weight module

fastga.models.weight.mass_breakdown.b_propulsion.b3_unusable_fuel_weight module

fastga.models.weight.mass_breakdown.b_propulsion.constants module

Constants for propulsion mass submodels.

fastga.models.weight.mass_breakdown.b_propulsion.sum module

Module contents

fastga.models.weight.mass_breakdown.c_systems package

Submodules

fastga.models.weight.mass_breakdown.c_systems.c1_power_systems_weight module

fastga.models.weight.mass_breakdown.c_systems.c2_life_support_systems_weight module

fastga.models.weight.mass_breakdown.c_systems.c3_avionics_systems_weight module

fastga.models.weight.mass_breakdown.c_systems.c4_recording_systems_weight module

fastga.models.weight.mass_breakdown.c_systems.constants module

Constants for systems mass submodels.

fastga.models.weight.mass_breakdown.c_systems.sum module

Module contents

fastga.models.weight.mass_breakdown.d_furniture package

Submodules

fastga.models.weight.mass_breakdown.d_furniture.constants module

Constants for furniture mass submodels.

fastga.models.weight.mass_breakdown.d_furniture.d2_passenger_seats_weight module

fastga.models.weight.mass_breakdown.d_furniture.sum module

Module contents

fastga.models.weight.mass_breakdown.unitary_tests package

Submodules

fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines module

Test module for geometry functions of mass breakdown components.

class `fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines.`**`DummyEngineBE76`**(*max_power: float*, *design_altitude_propelle float*, *fuel_type: float*, *strokes_nb: float*, *prop_layout: float*)

Bases: `fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

compute_drag(*mach*, *unit_reynolds*, *wing_mac*)

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float

Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float

Computes max available power on one engine.

Returns the maximum available power in W

class fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines.DummyEngineWrapperBE76

Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)

Defines the needed OpenMDAO inputs for propulsion instantiation as done in `get_model()`

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion

This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

class fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines.DummyEngineSR22(*max_power: float, design_altitude_propeller: float, fuel_type: float, strokes_nb: float, prop_layout: float*)

Bases: `fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion`

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See `FlightPoint` for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

`thrust_is_regulated` tells if a flight point should be computed using `thrust_rate` (when False) or `thrust` (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if `thrust_is_regulated` is not defined, the considered input will be the defined one between `thrust_rate` and `thrust` (if both are provided, `thrust_rate` will be used)
- if `thrust_is_regulated` is `True` or `False` (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.

- if there are several flight points, `thrust_is_regulated` is a sequence or array, `thrust_rate` and `thrust` should be provided and have the same shape as `thrust_is_regulated:code:`. The method will consider for each element which input will be used according to `thrust_is_regulated`.

Parameters **`flight_points`** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

`compute_weight``()` → float
Computes total propulsion mass.

Returns the total uninstalled mass in kg

`compute_dimensions``()` *-> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)*
Computes propulsion sub-components dimensions.

`compute_drag``(`*mach*, *unit_reynolds*, *wing_mac*`)`
Computes nacelle drag force for out of fuselage engine.

Parameters

- **`mach`** – mach at which drag should be calculated
- **`unit_reynolds`** – unitary Reynolds for calculation
- **`wing_mac`** – wing MAC length in m

Returns drag force cd0*wing_area

`get_consumed_mass``(`*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*`)` → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **`flight_point`** –
- **`time_step`** –

Returns the consumed mass in kg

`compute_max_power``(`*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*`)` → float
Computes max available power on one engine.

Returns the maximum available power in W

`class fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines.`**`DummyEngineWrapperSR22`**
Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

`setup``(`*component: openmdao.core.component.Component*`)`
Defines the needed OpenMDAO inputs for propulsion instantiation as done in *`get_model()`*

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **`component`** –

`static` **`get_model`**`(`*inputs*`)` → fastoad.model_base.propulsion.IPropulsion
This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

class fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines.**DummyEngineTBM900**(*fuel_type: float*, *prop_layout: float*)

Bases: *fastga.models.propulsion.fuel_propulsion.base.AbstractFuelPropulsion*

Dummy engine model returning nacelle aerodynamic drag force.

compute_flight_points(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*)

Computes Specific Fuel Consumption according to provided conditions.

See FlightPoint for available fields that may be used for computation. If a DataFrame instance is provided, it is expected that its columns match field names of FlightPoint (actually, the DataFrame instance should be generated from a list of FlightPoint instances).

Note: About thrust_is_regulated, thrust_rate and thrust

thrust_is_regulated tells if a flight point should be computed using thrust_rate (when False) or thrust (when True) as input. This way, the method can be used in a vectorized mode, where each point can be set to respect a **thrust** order or a **thrust rate** order.

- if thrust_is_regulated is not defined, the considered input will be the defined one between thrust_rate and thrust (if both are provided, thrust_rate will be used)
- if thrust_is_regulated is True or False (i.e., not a sequence), the considered input will be taken accordingly, and should of course be defined.
- if there are several flight points, thrust_is_regulated is a sequence or array, thrust_rate and thrust should be provided and have the same shape as thrust_is_regulated:code:. The method will consider for each element which input will be used according to thrust_is_regulated.

Parameters **flight_points** – FlightPoint or DataFram instance

Returns None (inputs are updated in-place)

compute_weight() → float

Computes total propulsion mass.

Returns the total uninstalled mass in kg

compute_dimensions() -> (<class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>, <class 'float'>)

Computes propulsion sub-components dimensions.

compute_drag(*mach: Union[float, numpy.array]*, *unit_reynolds: Union[float, numpy.array]*, *wing_mac: float*) → Union[float, numpy.array]

Computes nacelle drag force for out of fuselage engine.

Parameters

- **mach** – mach at which drag should be calculated
- **unit_reynolds** – unitary Reynolds for calculation
- **wing_mac** – wing MAC length in m

Returns drag force cd0*wing_area

get_consumed_mass(*flight_point: fastoad.model_base.flight_point.FlightPoint*, *time_step: float*) → float
Computes consumed mass for provided flight point and time step.

This method should rely on FlightPoint fields that are generated by :meth: *compute_flight_points*.

Parameters

- **flight_point** –
- **time_step** –

Returns the consumed mass in kg

compute_max_power(*flight_points: Union[fastoad.model_base.flight_point.FlightPoint, pandas.core.frame.DataFrame]*) → float
Computes max available power on one engine.

Returns the maximum available power in W

class
`fastga.models.weight.mass_breakdown.unitary_tests.dummy_engines.`**DummyEngineWrapperTBM900**
Bases: `fastoad.model_base.propulsion.IOMPropulsionWrapper`

setup(*component: openmdao.core.component.Component*)
Defines the needed OpenMDAO inputs for propulsion instantiation as done in *get_model()*

Use *add_inputs* and *declare_partials* methods of the provided *component*

Parameters **component** –

static get_model(*inputs*) → fastoad.model_base.propulsion.IPropulsion
This method defines the used `IPropulsion` subclass instance.

Parameters **inputs** – OpenMDAO input vector where the parameters that define the propulsion model are

Returns the propulsion model instance

fastga.models.weight.mass_breakdown.unitary_tests.test_beechcraft_1900 module

fastga.models.weight.mass_breakdown.unitary_tests.test_beechcraft_76 module

fastga.models.weight.mass_breakdown.unitary_tests.test_cirrus_sr22 module

fastga.models.weight.mass_breakdown.unitary_tests.test_daher_tbm900 module

fastga.models.weight.mass_breakdown.unitary_tests.test_maxwell_x57 module

fastga.models.weight.mass_breakdown.unitary_tests.test_partenavia_p68 module

Module contents

Submodules

fastga.models.weight.mass_breakdown.constants module

Constants for the mass breakdown submodels.

fastga.models.weight.mass_breakdown.mass_breakdown module

fastga.models.weight.mass_breakdown.payload module

fastga.models.weight.mass_breakdown.update_mlw_and_mzfw module

Main component for mass breakdown.

`class fastga.models.weight.mass_breakdown.update_mlw_and_mzfw.UpdateMLWandMZFW(**kwargs)`

Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes Maximum Landing Weight and Maximum Zero Fuel Weight from Overall Empty Weight and Maximum Payload.

Store some bound methods so we can detect runtime overrides.

Parameters `**kwargs` (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

`setup()`

Declare inputs and outputs.

Available attributes: name pathname comm options

`compute(inputs, outputs, discrete_inputs=None, discrete_outputs=None)`

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

`compute_partials(inputs, partials, discrete_inputs=None)`

Compute sub-jacobian parts. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **partials** (*Jacobian*) – sub-jac components written to partials[output_name, input_name]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.

fastga.models.weight.mass_breakdown.update_mtow module

Main component for mass breakdown.

class fastga.models.weight.mass_breakdown.update_mtow.**UpdateMTOW**(***kwargs*)

Bases: `openmdao.core.explicitcomponent.ExplicitComponent`

Computes Maximum Take-Off Weight from Maximum Zero Fuel Weight and fuel weight.

Store some bound methods so we can detect runtime overrides.

Parameters **kwargs (*dict of keyword arguments*) – Keyword arguments that will be mapped into the Component options.

setup()

Declare inputs and outputs.

Available attributes: name pathname comm options

compute(*inputs*, *outputs*, *discrete_inputs=None*, *discrete_outputs=None*)

Compute outputs given inputs. The model is assumed to be in an unscaled state.

Parameters

- **inputs** (*Vector*) – unscaled, dimensional input variables read via inputs[key]
- **outputs** (*Vector*) – unscaled, dimensional output variables read via outputs[key]
- **discrete_inputs** (*dict or None*) – If not None, dict containing discrete input values.
- **discrete_outputs** (*dict or None*) – If not None, dict containing discrete output values.

Module contents

Submodules

fastga.models.weight.constants module

Constants for the weight submodels.

fastga.models.weight.weight module

Weight computation (mass and CG).

class fastga.models.weight.weight.**Weight**(***kwargs*)

Bases: `openmdao.core.group.Group`

Computes masses and Centers of Gravity for each part of the empty operating aircraft, among these 5 categories: airframe, propulsion, systems, furniture, crew

This model uses MTOW as an input, as it allows to size some elements, but resulting OWE do not aim at being consistent with MTOW.

Consistency between OWE and MTOW can be achieved by cycling with a model that computes MTOW from OWE, which should come from a mission computation that will assess needed block fuel.

Set the solvers to nonlinear and linear block Gauss–Seidel by default.

Parameters `**kwargs` (*dict*) – dict of arguments available here and in all descendants of this Group.

`initialize()`
Perform any one-time initialization run at instantiation.

`setup()`
Build this group.

This method should be overidden by your Group's method. The reason for using this method to add subsystem is to save memory and setup time when using your Group while running under MPI. This avoids the creation of systems that will not be used in the current process.

You may call 'add_subsystem' to add systems to this group. You may also issue connections, and set the linear and nonlinear solvers for this group level. You cannot safely change anything on children systems; use the 'configure' method instead.

Available attributes: name pathname comm options

Module contents

Submodules

fastga.models.options module

Module for management of options and factorizing their definition.

Module contents

This package contains the OAD models of FAST-OAD.

These models are based on following references:

fastga.source_files package

Module contents

fastga.utils package

Subpackages

fastga.utils.postprocessing package

Subpackages

fastga.utils.postprocessing.load_analysis package

Submodules

fastga.utils.postprocessing.load_analysis.analysis_and_plots_la module

Defines the analysis and plotting functions for postprocessing of load analysis.

`fastga.utils.postprocessing.load_analysis.analysis_and_plots_la.force_repartition_diagram`(*aircraft_file_path: str*, *name=''*, *fig=None*, *file_formatter=No* → plotly.graph_objs

Returns a figure plot of the force repartition on the wing. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: force repartition diagram.

`fastga.utils.postprocessing.load_analysis.analysis_and_plots_la.shear_diagram`(*aircraft_file_path: str*, *name=''*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget.F

Returns a figure plot of the shear repartition on the wing. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: force repartition diagram.

`fastga.utils.postprocessing.load_analysis.analysis_and_plots_la.rbm_diagram`(*aircraft_file_path: str*, *name=''*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget.Fig

Returns a figure plot of the root bending moment repartition on the wing. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot

- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: force repartition diagram.

Module contents

fastga.utils.postprocessing.propeller package

Submodules

fastga.utils.postprocessing.propeller.analysis_and_plots_propeller module

Defines the analysis and plotting functions for postprocessing of propeller performances.

fastga.utils.postprocessing.propeller.analysis_and_plots_propeller.**propeller_efficiency_map_plot**(*aircraft_*
str,
file_form
sea_leve
→
plotly.gr

Returns a contour plot of the propeller efficiency maps as they are used in FAST-OAD-GA.

Parameters

- **aircraft_file_path** – path of data file
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :param sea_level: boolean to choose whether to plot the sea level maps or the cruise level map. :return: propeller efficiency map.

fastga.utils.postprocessing.propeller.analysis_and_plots_propeller.**propeller_coeff_map_plot**(*aircraft_file_pa*
str,
name='',
fig=None,
file_formatter=
→
plotly.graph_ol

Returns a two subplot figure of the thrust and power coefficient of the propeller. Different figure can be superposed by providing an existing fig. Each figure can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: thrust and power coefficient graphs

Module contents

fastga.utils.postprocessing.unitary_tests package

Subpackages

fastga.utils.postprocessing.unitary_tests.data package

Module contents

Submodules

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots module

Tests for analysis and plots functions

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_aircraft_geometry_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_evolution_diagram_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_compressibility_effect_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_cl_wing_diagram_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_cg_lateral_diagram_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_mass_breakdown_bar_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_mass_breakdown_sun_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_drag_breakdown_diagram_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_payload_range_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots.**test_aircraft_polar_plot**()
    Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_la module

Tests for analysis and plots functions

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_la.**test_force_repartition_diagram**()
Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_la.**test_shear_diagram**()
Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_la.**test_rbm_diagram**()
Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_prop module

Tests for analysis and plots functions

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_prop.**test_efficiency_map_plot**()
Basic tests for testing the plotting.

fastga.utils.postprocessing.unitary_tests.test_analysis_and_plots_prop.**test_coefficient_map_plot**()
Basic tests for testing the plotting.

Module contents

Submodules

fastga.utils.postprocessing.analysis_and_plots module

Defines the analysis and plotting functions for postprocessing.

fastga.utils.postprocessing.analysis_and_plots.**aircraft_geometry_plot**(*aircraft_file_path: str*, *name=''*, *fig=None*, *plot_nacelle: bool = True*, *file_formatter=None*) → plotly.graph_objs._figurewidget.FigureWidg

Returns a figure plot of the top view of the wing. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **plot_nacelle** – boolean to turn on or off the plotting of the nacelles
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: wing plot figure.

`fastga.utils.postprocessing.analysis_and_plots.`**`evolution_diagram`**(*aircraft_file_path: str*, *name=''*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget.FigureWidget

Returns a figure plot of the V-N diagram of the aircraft. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: V-N plot figure.

`fastga.utils.postprocessing.analysis_and_plots.`**`compressibility_effects_diagram`**(*aircraft_file_path: str*, *name: str = ''*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget

Returns a figure plot of the evolution of the lift curve slope with Mach number.

Parameters

- **aircraft_file_path** – path of the aircraft data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: Cl_alpha distribution with Mach number.

`fastga.utils.postprocessing.analysis_and_plots.`**`cl_wing_diagram`**(*aircraft_file_path: str*, *name: str = ''*, *prop_on: bool = False*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget.FigureWidget

Returns a figure plot of the CL distribution on the semi-wing.

Parameters

- **aircraft_file_path** – path of the aircraft data file
- **name** – name to give to the trace added to the figure
- **prop_on** – boolean stating if the rotor is on or off (for single propeller plane)
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: Cl distribution figure along the span.

fastga.utils.postprocessing.analysis_and_plots.**cg_lateral_diagram**(*aircraft_file_path: str*, *name=''*, *fig=None*, *file_formatter=None*, *color=None*) → plotly.graph_objs._figurewidget.FigureWidget

Returns a figure plot of the lateral view of the plane. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **color** – color that we give to the aft, empty and fwd CGs of the aircraft
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :return: wing plot figure.

fastga.utils.postprocessing.analysis_and_plots.**mass_breakdown_bar_plot**(*aircraft_file_path: str*, *name=None*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget.FigureWi

Returns a figure plot of the aircraft mass breakdown using bar plots. Different designs can be superposed by providing an existing fig. Each design can be provided a name.

Parameters

- **aircraft_file_path** – path of data file
- **name** – name to give to the trace added to the figure
- **fig** – existing figure to which add the plot
- **file_formatter** – the formatter that defines the format of data file. If not provided, default format will be assumed.

Returns bar plot figure.

fastga.utils.postprocessing.analysis_and_plots.**mass_breakdown_sun_plot**(*aircraft_file_path: str*, *file_formatter=None*)

Returns a figure sunburst plot of the mass breakdown. On the left a MTOW sunburst and on the right a OWE sunburst.

Parameters

- **aircraft_file_path** – path of data file
- **file_formatter** – the formatter that defines the format of data file. If not provided, default format will be assumed.

Returns sunburst plot figure.

fastga.utils.postprocessing.analysis_and_plots.**drag_breakdown_diagram**(*aircraft_file_path: str*, *file_formatter=None*) → plotly.graph_objs._figurewidget.FigureWidg

Return a plot of the drag breakdown of the wing in cruise conditions.

`fastga.utils.postprocessing.analysis_and_plots.`**`payload_range`**(*aircraft_file_path: str*, *name=''*, *fig=None*, *file_formatter=None*) → plotly.graph_objs._figurewidget.FigureWidget

Returns a figure plot of the payload range diagram of the plane. Different designs can be superposed by providing an existing fig. Each design can be provided a name. :param aircraft_file_path: path of data file :param name: name to give to the trace added to the figure :param fig: existing figure to which add the plot :param file_formatter: the formatter that defines the format of data file. If not provided, default format will be assumed. :return: payload range figure.

`fastga.utils.postprocessing.analysis_and_plots.`**`aircraft_polar`**(*aircraft_file_path: str*, *name=None*, *fig=None*, *file_formatter=None*, *equilibrated=False*) → plotly.graph_objs._figurewidget.FigureWidget

Returns a figure plot of the polar of the plane. Different designs can be superposed by providing an existing fig. Each design can be provided a name. The value obtained for the finesse for the equilibrated drag polar is quite low.

Parameters

- **`aircraft_file_path`** – path of data file
- **`name`** – name to give to the trace added to the figure
- **`fig`** – existing figure to which add the plot
- **`file_formatter`** – the formatter that defines the format of data file. If not provided,

default format will be assumed. :param equilibrated: boolean stating if the polar plotted is the equilibrated one or not :return: plane polar figure.

fastga.utils.postprocessing.post_processing_api module

Module contents

fastga.utils.resource_management package

Submodules

fastga.utils.resource_management.copy module

`fastga.utils.resource_management.copy.`**`copy_resource_from_path`**(*source: str*, *resource: str*, *target_path*)

Copies the indicated resource file to provided target path.

If target_path matches an already existing folder, resource file will be copied in this folder with same name. Otherwise, target_path will be the used name of copied resource file

Parameters

- **`source`** – source of the resource to copy
- **`resource`** – name of the resource to copy
- **`target_path`** – file system path

Module contents

Submodules

fastga.utils.warnings module

A module for FAST-OAD-GA specific warnings.

exception fastga.utils.warnings.**FASTOADGAWarning**

Bases: `UserWarning`

Base class for FAST-OAD-GA warning.

name = 'warn_fast_oad_ga'

exception fastga.utils.warnings.**VariableDescriptionWarning**

Bases: *fastga.utils.warnings.FASTOADGAWarning*

Warning class for warnings in the generation of variable_descriptions.txt

name = 'warn_variable_descriptions'

Module contents

This package contains various utilities.

Submodules

fastga.conftest module

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[EAS] EASA. *Certification Specifications and Acceptable Means of Compliance for Normal, Utility, Aerobatic, and Commuter Category Aeroplanes CS-23 Amendment 4, July 2015*. https://www.easa.europa.eu/.

[Gud13] Snorri Gudmundsson. *General aviation aircraft design: Applied Methods and Procedures*. Butterworth-Heinemann, 2013.

[HJimenezEchavarria+22] José D Hoyos, Jesús H Jiménez, Camilo Echavarría, Juan P Alvarado, and Germán Urrea. Aircraft propeller design through constrained aero-structural particle swarm optimization. *Aerospace*, 9(3):153, 2022.

[Int18] ASTM International. *ASTM F3116 / F3116M-18e2, Standard Specification for Design Loads and Conditions*. West Conshohocken, PA, 2018. www.astm.org.

[NC10] Leland M Nicolai and Grant E Carichner. *Fundamentals of aircraft and airship design, Volume 1–Aircraft Design*. American Institute of Aeronautics and Astronautics, 2010.

[Ray12] Daniel P Raymer. Aircraft design: a conceptual approach (aiaa education series). *Reston, Virginia*, 2012.

[Ros85a] Jan Roskam. *Airplane Design: Part 5-Component Weight Estimation*. DARcorporation, 1985.

[Ros85b] Jan Roskam. *Airplane Design: Part 6-Preliminary Calculation of Aerodynamic, Thrust and Power Characteristics*. DARcorporation, 1985.

[WHM17] Douglas P Wells, Bryce L Horvath, and Linwood A McCullers. The flight optimization system weights estimation method. 2017.

[YJ83] **missing institution in yamauchi:1983**

PYTHON MODULE INDEX

f

INDEX

D

E

F

G

N

P

R

S

U

V

W

X

Y