Mod5 – A Matlab Class Wrapper for MODTRAN 5

Introduction

MODTRAN (www.modtran.org) is a software code for computation of electromagnetic radiative transfer in the Earth's atmosphere. MODTRAN is controlled through the use of an input text file having a strict format. This file is referred to as 'tape5' and may have the extension .tp5 or .ltn. It is inconvenient to create or edit this file manually and many users of MODTRAN have software tools to create and edit tape5. These tools include various Graphical User Interfaces (GUIs) such as PcModWin from Ontar Corporation (www.ontar.com) and MODO from Rese (www.rese.ch).

The Matlab class wrapper described here, *Mod5*, is not a GUI. It provides a set of functions (called "methods" in object-oriented programming terminology) that ease the handling of MODTRAN cases. These include methods to read the tape5 file format into a *Mod5* class "instance" or "object", alter the case instance described in the file, write the tape5 format back to a file, run MODTRAN on the case and read various results from MODTRAN output files back into Matlab. A number of plotting methods are also available.

Requirements for Mod5

The class wrapper was written for MODTRAN 5 and requires Matlab 2009a or later. MODTRAN itself is not included with the distribution. The MODTRAN executable is obtainable from Ontar Corporation.

Licensing

The *Mod5* class software is at a beta testing level and may still contain significant bugs. The conditions of use are specified by the BSD license (the standard open-source license used at The Mathworks for public contributions). See the file BSDlicence.txt included in the download archive for details. Please report any bugs and problems to the author (dgriffith@csir.co.za).

Installation

Unpack the archive in a convenient directory. Add the directory to your Matlab path. Explore the *Mod5Examples* script file by opening it in the Matlab editor.

MODTRAN Inputs

MODTRAN has a large number of inputs in tape5 and these are described in detail in the MODTRAN 5 User's Manual. The manual can be found online and is the definitive reference for the meaning of the various MODTRAN inputs.

The sheer number of MODTRAN inputs can be daunting for the new user. However, to get your first case running, there is only a small subset of inputs actually required.

The chief outputs of MODTRAN (described in more detail below) are the spectral transmittance and spectral radiance of lines-of-sight (LOS or "paths") passing through the Earth's atmosphere under the conditions described by the inputs. MODTRAN can also compute the direct normal solar (or lunar) irradiance (DNI) at a particular point in the atmosphere. The solar DNI is useful for solar energy applications amongst others.

The chief modes of execution for MODTRAN are as follows:

a) Compute LOS spectral transmission.
b) Compute LOS radiance excluding scattered shortwave (solar/lunar) radiation.
c) Compute LOS radiance including scattered shortwave (solar/lunar) radiation.
d) Compute solar/lunar DNI.

MODTRAN inputs are organized (within tape5) into lines of data referred to, for historical reasons, as "cards". The mode of execution mentioned above is specified by the input parameter IEMSCT which appears on Card 1.

Within the LOS radiance modes it is possible to select whether you wish to include path radiance contributions due to light scattered into the path coming from the Sun or Moon (c), or deal only with the thermally emitted portion (b). It is also possible to specify whether you wish to compute only single-scatted solar/lunar radiation or to include multiple scatter components. These additional components could come via the surface of the Earth or from atmospheric components such as aerosols (solid or liquid particles suspended in the atmosphere).

Examples

Examples of using the *Mod5* wrapper are to be found in the Matlab script file *Mod5Examples.m*, included in the download archive. Each example is given in a cell within the script file. The examples are to be run individually by clicking in the example and using the "Evaluate Cell" button on the standard Matlab editor toolbar. Note that every example will clear all workspace variables and close all plots. Look over the code in the example before executing it.

The first time that the `Run` method is executed on a *Mod* instance, i.e. the first time that MODTRAN is called, a dialog will appear requesting that you point the class to your MODTRAN executable file (preferably MOD4v1r1.EXE), wherever that is installed on your computer. This should only be required once as the class will then save the location. However, if you move MODTRAN or reinstall to another location, you will have to do this again. You can explicitly point *Mod5* to the MODTRAN executable by calling the static method:

```
>> Mod5.SetMODTRANExe;
```

MODTRAN Outputs

MODTRAN writes a number of output files, depending on certain input parameters, including the following:

tape6

The tape6 (or .tp6) output file is always produced for a normal MODTRAN run. This file contains a large amount of information about the run, the atmospheric profiles and results for radiance and transmittance. This file should be inspected to ensure that your MODTRAN run executed as desired. Currently there is no function in *Mod5* to read data from tape6.

tape7

The tape7 (or .tp7) file contains only the radiance, transmittance or irradiance results, one block per run, starting with header information about the run and terminated with the value -9999. When a case is run using the `Run` method, tape7 data is read back into the *Mod5* instance into a property called tp7. This is a structure with a set of fields that depends on the data that MODTRAN has written to the tape7 file.

tape7.scn

The tape7.scn (or .7sc) file (if written by MODTRAN at all) contains the spectrally convolved data – that is data for which the spectral resolution has been downgraded. This data will be read back into the *Mod5* instance into a property called sc7.

channels.out

The channels.out (or .chn) file is written if spectral channel filters have been attached to the case. If so, the content of this file is read into the class property called chn.

Other MODTRAN Output Files

There are currently no methods in *Mod5* for reading output from tape6, and clrates (cooling rates).

Mod5 Methods

Methods are functions for creating and manipulating *Mod5* objects.

Constructor Method

There are two ways of creating a *Mod5* instance. These are to read the case from a tape5 format file (.ltn and .tp5 files follow this format) or to create the case from scratch. To read a case from a file in the current directory, the call to the constructor method looks as follows:

```
>> MyCase = Mod5('AnExistingCase.tp5');
```

To get a file selection dialog, call the constructor with the empty string:

```
>> MyCase = Mod5('');
```

To create a case from scratch such as in example 2, it is necessary first to obtain an empty *Mod5* object. This is done by calling the constructor without any parameters:

```
>> MyEmptyCase = Mod5;
```

Once you have an empty case, the MODTRAN variables in the case instance can be set in conformance with MODTRAN rules (see example 2).

Super-Cases and Sub-Cases

Often, documentation will refer to “sub-cases” or “super-cases”. A “super-case” is merely a *Mod5* object having more than one sub-case (i.e. it is a vector or matrix of sub-cases). All sub-cases in a *Mod5* instance will be executed in a single run of MODTRAN when the `Run` method is used. A scalar *Mod5* instance has a single sub-case.

It is possible to replicate a *Mod5* instance to create a super-case using the `Replicate` method. Do not use the built-in `repmat` Matlab function. To create a row vector of 5 identical sub-cases, use the following:

```
>> MyReplicatedCase = MyCase.Replicate([1 5]);
```

Once you have the vector of sub-cases, you can change selected MODTRAN inputs for each of the 5 sub-cases using standard Matlab indexing.

It is also possible to extract a sub-set of sub-cases from a super-case using the `Extract` method, for example the following will extract sub-cases 2 and 3:

```
>> MySelectedCase = MyCase.Extract([2 3]);
```

Do not use horizontal or vertical concatenation on *Mod5* objects, as this will result in a super-case with incorrect internal indexing and IRPT flags. Instead, use the `Merge` method.

```
>> MySuperCase = MyCase1.Merge(MyCase2);
```

Writing Cases

To write out a tape5 format file use:

```
>> MyCase.Write('MyCaseFile.tp5');
```

Or to get a file/save dialog, simply:

```
>> MyCase.Write;
```

Setting MODTRAN Parameters

The MODTRAN input parameters are all available as *Mod5* properties. The properties of a sub-case are set using standard Matlab syntax, for example:

```
>> MyCase(iSubCase).MODTRN = 'M'; % Set the MODTRAN Spectral Band Model
```

To set a number of MODTRAN variables for a sub-case in a single function call use the `Set` method. See example 3 for use of this method.

Running MODTRAN

Once a *Mod5* (be it a scalar case or a super-case) instance has been constructed it is a simple matter to run MODTRAN on the case.

```
>> MyCase = MyCase.Run;
```

The first time the `Run` method is executed, *Mod5* will present a file selection dialog, and you must select the MODTRAN executable to be used. *Mod5* was developed and tested with MODTRAN 5.2.0. Other versions may not work with this wrapper.

If run from the command line, MODTRAN outputs some text indicating progress of the computation. Unfortunately this output is not available to Matlab in real time. However, some of this text (at least the tail end) will be captured in the *Mod5* property `MODSays`. If the run appears to have aborted, the `Run` method will issue a warning. The MODTRAN tape6 (or CaseName.tp6) file should be examined to determine what went wrong. The `MODSays` output is only attached to the first sub-case of a super-case.

The tape5 (.tp5) as well as all the MODTRAN output files associated with a *Mod5* instance will be written to the same directory in which the MODTRAN executable is located. This can result in accumulation of a large number of files in that directory over time.

To save the input (.tp5) and all output data to a specific directory, use the following:

```
>> MyCase.Save(SaveDirectory);
```

This will also save a .mat file with the *Mod5* object in it. If no directory is specified, the files will be moved to the directory set using the `SetCasePath` method. To save to you current working directory use:

```
>> MyCase.Save(pwd);
```

To delete the files created in the MODTRAN executable directory associated with a particular case, use the `Purge` method.

```
>> MyCase.Purge;
```

Plotting Results

There are several methods for plotting various data items, including MODTRAN outputs. See the following plotting methods:

```
MyCase.PlotTp7  - plots selectable MODTRAN tape7 outputs
MyCase.PlotSc7  - plots selectable convolved MODTRAN tape7 outputs
MyCase.PlotChn  - plots channel (.chn) MODTRAN outputs
MyCase.PlotAtm  - plots user-defined atmospheric data
```

There are also static methods for plotting albedo (lambertian reflectance) data and filter data. See the relevant sections below.

Help

More detailed documentation on the class, properties and methods can be obtained using:

```
>> doc Mod5
```

Note that the documentation generated by `doc` will list the properties in alphabetical order (rather than the MODTRAN Card order). Properties in all upper case are MODTRAN inputs. The list of methods appears at in a table at the bottom, with hyperlinks to method help text.

Spectral Response Filters and Channels

In many applications, e.g. Earth observation, the sensor or camera will respond only to certain wavelengths. The sensor may respond to a fairly broad range of spectral wavelengths of light, but is usually more sensitive to certain wavelengths. Raw MODTRAN results, on the other hand, have very high spectral resolution. MODTRAN can calculate the so-called “band radiance” or band-averaged transmittance relevant to such cameras which may have multiple spectral channels. Example 2 illustrates creation of the sensor spectral response filters (SRF) and getting MODTRAN to compute the band radiance for each of the filters. It is important to distinguish the SRF concept from the convolution (or “slit”) instrument functions used by MODTRAN to degrade the spectral resolution of transmission and radiance results. The difference should become clear in the examples.

By creating small variations on the SRF, the sensitivity of the camera output to variations in the channel filters can be explored.

The following static methods are available for working with spectral channel filters. A static method is one that is associated with the class but does not take a class instance as an input. Static functions must be called with the name of the class as a prefix.

```
Mod5.ReadFlt   - Read a MODTRAN .flt spectral filter file
Mod5.CreateFlt - Create a synthetic set of spectral filters
Mod5.WriteFlt  - Write out a MODTRAN .flt filter set
Mod5.PlotFlt   - Plot a set of MODTRAN spectral channel filters
```

```
Mod5.ReadFltFromSensorML - Extract a set of spectral channel
definitions from a SensorML file (Sensor Markup Language)
```

A filter set that is read from a .flt file, or created using `CreateFlt` can be attached to a *Mod5* instance using:

```
>> MyCase = MyCase.AttachFlt(MyFilter);
```

This should be sufficient to get MODTRAN to perform the band calculation and the results will be attached to the class instance (in the property 'chn') when MODTRAN is run.

Warning : Attaching of channel filters and albedo data (see next section) should, if possible, be the last thing done before running a case. Incorrect or unintended results could occur if `Replicate`, `Extract` or `Merge` are used after attaching filter data to a case.

Using Albedo Data

MODTRAN can model the effect of different lambertian spectral reflectance (albedo) at the target or boundary surface. This is used in radiance cases (IEMSCT = 1 or 2), by setting the SURREF property to 'LAMBER' on Card 1 and then pointing MODTRAN to the spectral albedo file (property SALBFL on Card 4L1), and choosing an albedo curve contained in the file using the CSALB property on Card 4L2. There are a number of *Mod5* methods for working with spectral albedo data, as follows:

```
Mod5.ReadAlb   - Read a MODTRAN format spectral albedo file
Mod5.CreateAlb - Create a spectral albedo curve
Mod5.WriteAlb  - Write out a MODTRAN spectral albedo file
Mod5.PlotAlb   - Plot a set of spectral albedo curves
Mod5.ReadAlbFromASD - Read albedo data from an ASD text file
Mod5.ReadAlbFromUSGS - Read albedo data from a USGS file
Mod5.MixAlb    - Mix several albedos in a weighted mean
Mod5.InterpAlb - Interpolate albedo data to other wavelengths
Mod5.AttachAlb - Attach albedo data to sub-cases
```

MODTRAN has a compile-time parameter called MWVSRF that defines the maximum number of albedo spectral points. When reading albedo data from external sources, care must be taken not to exceed the maximum. Normally, MWVSRF = 50, and it is necessary to recompile MODTRAN to increase this maximum. Use the function `InterpAlb` to reduce the number of points in your data. Take care that interpolation fully captures the spectral details in the albedo curve. This can be done by using `PlotAlb` to plot the original and reduced data together (See Example 6).

Warning : Attaching of albedo data and channel filters (see previous section) should, if possible, be the last thing done before running a case. Incorrect or unintended results could occur if `Replicate`, `Extract` or `Merge` are used after attaching albedo data to a case.

Sensitivity Studies

Sensitivity studies are one of the most important applications of MODTRAN. For example, you may wish to know what happens to radiative transfer when there are fluctuations in atmospheric conditions. You would then set up a series of MODTRAN runs (a sensitivity series) in which the MODTRAN results are compared when the input of interest is altered.

Mod5 includes a function called `CreateSeries` that facilitates generation of such series. For example, suppose you wish to know by how much atmospheric transmission varies as you scale the water vapor column (i.e. multiply the water vapor content at all altitudes by a certain factor). Suppose also that you would also like to get an idea of what effect the increased humidity might have on aerosol transmission due to particle growth. MODTRAN has a string input called H2OSTR that allows scaling of the water vapor column and a flag called H2OAER that controls whether or not the aerosols are affected by the change in water vapor. Suppose further that you are interested in a 10% water vapor reduction, a 10% increase and a 20% increase. The following command will create a suitable series based on an existing *Mod5* instance called BaseCase (which must be scalar).

```
>> WatVapSeries = BaseCase.CreateSeries('H2OSTR', {'0.9','1.1','1.2'}, …
                                                  'H2OAER', {'t', 'f'});
```

It is important to note that `CreateSeries` generates all combinations of the input parameters. The above example creates an array of 3 by 2 sub-cases with the H2OSTR water vapor scaling factor varying down the columns and the H2OAER flag changing from 't' in the first column to 'f' in the second column.

More than two parameters can be specified and an extra dimension is added to the matrix of sub-cases for every parameter. The total number of sub-cases is the product of the number of values given for each of the parameters, so the number of sub-cases goes up rapidly with more parameters and values.

Notes

The following are important considerations:

a) If a particular card is required in a case, all of the parameters on the card must be set, even if the parameters in question are not used. This is a restriction of the wrapper itself, which will, hopefully, be removed in a future version.
b) MODTRAN has a large number of input parameters, and exhaustive testing of the wrapper class was not possible. Significant bugs may still lurk, and the user must be very critical with regard to outputs. Please report any bugs you may find to the author.
c) The wrapper has only been tested on a PC under Windows XP. The wrapper may not work on other operating systems.

d) Integrity checking of many of the MODTRAN parameters has not yet been implemented (‘set’ methods). Only parameters on the main cards are validated in any way.

Acknowledgements

The `CreateFlt` method uses the function `srf_generate` written by Jason Brazile. Meena Lysko, Tami Mudau and others have assisted with testing of the wrapper.

Please notify the author of any bugs or if you would like to make a contribution.
Derek Griffith <dgriffith@csir.co.za>