Pratmo Box Model Interface API

Version 1.2.0, September 28, 2015

Table of Contents

The Pratmo Box Model API

The Pratmo Box Model (Pratmo++) API is used to model the photo-chemistry occurring at stratospheric and mesospheric heights. The model evaluates the production and loss terms from approximately 200 chemical and photo-chemical reactions occurring in the middle atmosphere and searches for a consistent solution in each box by equating the rate of change of a species with its production and loss terms.

The calculation is achieved by adjusting the number density of 29-30 reactive species using a backward Euler root finding algorithm. There are also ~80 a-priori species and terms used to represent the background atmospheric state, long–lived species and photo-dissociation production rates. Atmospheric state and long-lived species are modeled using various climatologies while photo-dissociation rates are calculated using an internal radiative transfer model. The internal radiative transfer model is based upon vertical optical depth calculations adjusted for solar zenith angles using a spherical atmosphere and has an additional correction for Rayleigh and albedo scattering.

The Pratmo box model was originally written in Fortran by Chris McLinden from code developed by Prather et al. The software provided here is a C++ version of the McLinden code. The code has been re-written but has been shown to give identical results. We have updated the code to run in a modern multi-threaded environment and we have tried to provide the user with more access and flexibility in configuring the inputs and outputs.

Our group originally developed this code to model the diurnal variation of NO2 in the stratosphere and apply this diurnal variation as a correction to our NO2 limb profiles measured from space. The code can be used for other goals but please keep in mind that our primary purpose has been to capture the NOx diurnal cycle. Keen users should review all of the chemical reactions built into the Pratmo model to ensure they are appropriate for their application.

ISKCLIMATOLOGY Interface

The Pratmo box model is encapsulated in an ISKClimatology interface created with '**PRATMOBOXMODEL**'. The Pratmo box model climatology solves the pratmo box model for an entire 24 hour period at a set of user defined latitudes.

The climatology interpolates the retrieved pratmo profiles in latitude, longitude, height and time. The model is configured by default to use 7 internal box models to span 35° of latitude in 5° steps. Consequently it can be quite slow when first initialized during a call to *UpdateCache*, taking between 5 and 20 seconds.

NO2 Scaling

The pratmo box model provides a scaling feature for NO2 profiles whereby the returned NO2 value at any time and location is scaled by the ratio of $(actual\ NO2)/(pratmo\ NO2)$ at a specific location. The actual NO2 and specific location are provided by the user and by default are disabled. Scaling is only expected to work properly over the altitude range 0 to 70 km.

Example

```
climate = ISKClimatology('PRATMOBOXMODEL')
climate.UpdateCache( [60,0,0, 56732.5 );
 [ok, value] = climate.GetParameter( SKCLIMATOLOGY_NO2_CM3(), [60, 12, 35000.0, 56732.5]);
```

The PRATMOBOXMODEL climatology supports the following properties,

Properties

Albedo	*Scalar*	Sets the albedo used by the Pratmo box model in its internal radiative transfer engine. Should be a value between 0 and 1.0.
NumNominalTimeSteps	*Scalar*	Sets the number time steps evaluated in a 24 hour period. The default is 35. Do not set this value too low (< 20) or convergence problems will occur. The value has been tested up to 1200. The value is internally converted to an integer.
NumDaysForConvergence	*Scalar*	Sets the number of days to be repeated for convergence. The default is 40. The value is internally converted to an integer.
OzoneIsLongLived	*Scalar*	Sets the treatment of ozone. A non-zero value lets ozone be treated as a long-lived species. A value of zero treats

		ozone as a radical species and is fitted in the analysis.
AddCustomLocalSolarTime	*Scalar*	Allows the user to add a specific local solar time to the time steps used by the Pratmo box model. This is useful if the user wishes to ensure there are solutions at a specific local solar time. Solar time is 12.00 at local noon, ~18.00 at dusk, 0.00 at midnight and ~06.00 at dawn. The value must be specified in the range 0-24.
AddCustomSolarZenithAngle	*Scalar*	Allows the user to add the time of a specific solar angle to the time steps used by the Pratmo box model. This is useful if the user wishes to ensure there are solutions at a specific solar zenith angle. The angle must be in the range 0-180. Custom solar zenith angles not encountered by a specific solution of the pratmo model are internally cached for other runs but are ignored by the current model solution.
LatitudeResolution	*Scalar*	The user may specify the latitude spacing between the array of pratmo solutions used in the climatology. The default is 5.0 degrees
NumLatitudeProfiles	*Scalar*	The user may specify the number of pratmo solutions used in this climatology. This allows the user to track and interpolate systematic latitudinal changes in the pratmo box model solution. The array of latitudes is centred on the reference point used in *UpdateCache* and are spaced by the value used in the last call to *LatitudeResolution*.
BoxHeights	*Array*	An ascending array of heights expressed in meters. The pratmo solution is solved at all of these heights. The pratmo box model is suitable for altitudes between the tropopause and ~60 km altitude. The default is a ~2 km spacing derived from a climatological pressure grid.
AtmosphericHeights	*Array*	An ascending array of heights expressed in meters. The pratmo solution uses this grid for the internal radiative transfer. The set of heights should cover the region 0 to 80 km. The default is a ~2

		km spacing derived from a climatological pressure grid.
NO2ScalingLocation	*Array*	Allows the user to specify the scaling location for NO2 number density. The array must be a 4 element array of (latitude, longitude, height, mjd). Note that the height field is ignored. This option must be used in conjunction with *SetTrueNO2Climatology*. The default value is the location used in the first call to *UpdateCache.*
Climatology *XXX*	*Object*	Allows the user to set an internal climatology used by Pratmo. The value of *XXX* should be the name of the internal climatology being replaced. See Pratmo_BoxModel::SetClimatology for a list of acceptable values for *XXX*.
SetTrueNO2Climatology	*Object*	An ISKClimatology object that provides a profile of NO2 that is considered truth. The pratmo box model will use this profile to scale all of its NO2 values. The climatology must support SKCLIMATOLOGY_NO2_CM3 between 0 and 100 km inclusive.

Installation Package	Pratmo Box Model
External Dependencies	None.

Back to ISKClimatology
Back to Table of Contents

Pratmo_BoxModel

The software object used to model diurnal cycles of various species in the middle atmosphere. The model provides default climatologies based upon the original Fortran code and also permits the user to provide their own climatologies of various components of atmospheric state. The user can define both the altitudes and local times in the diurnal cycle at which solutions are required. We recommend users keep their height requests between 0 and 80 km.

The Pratmo_BoxModel creates variables in 5 specific sections:

1. Atmospheric Section
2. Radical Section
3. Long-Lived Section
4. Miscellaneous Section
5. Photo-dissociation Section.

Variables in all sections can be read by the user after a solution has been calculated. Only variables in the Radical section are calculated by Pratmo using a balance of rate of change with production/loss. The variables in the other 4 sections are calculated by Pratmo during initialization as inputs to the solution for the radicals.

Ozone can be treated as either a radical, in which case its value is solved, or it can be treated as long-lived, in which case its value is held constant.

Pratmo_BoxModel::Pratmo_BoxModel

Constructs the Pratmo_BoxModel object with default settings based upon the original Fortran source code. By default, O3 is considered a long-lived species.

C++

```
Pratmo_BoxModel::Pratmo_BoxModel(
);
```

Matlab

```
model = Pratmo_BoxModel()
```

Python

```
model = Pratmo_BoxModel.Pratmo_BoxModel()
```

Back to Table of Contents

Pratmo_BoxModel::AddCustomLocalSolarTime

Allows the user to add specific Local Solar Times to the list of times internally generated by Pratmo. Useful if the user wants to ensure that Pratmo generates a solution at a specific local solar time.

C++

```
bool Pratmo_BoxModel::AddCustomLocalSolarTime (
    double                          lst
);
```

Matlab

```
ok = model.AddCustomLocalSolarTime( lst );
```

Python

```
ok = model.AddCustomLocalSolarTime( lst );
```

Parameters:

`double` ***lst***

The specific local solar time (0.0 to 24) in hours to be added. 0 is midnight, 6 is nominal dawn, 12 is noon and 18 is nominal dusk.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::AddCustomSolarZenithAngle

Allows the user to add the times of specific Solar Zenith Angles to the list of times internally generated by Pratmo. Useful if the user wants to ensure that Pratmo generates a solution at a specific solar zenith angle. Note that a specific solar zenith angle will occur before and after noon and both times are inserted into the processing.

C++

```
bool Pratmo_BoxModel::AddCustomSolarZenithAngle (
    double                          sza,
);
```

Matlab

```
[ok] = model. AddCustomSolarZenithAngle ( sza );
```

Python

```
ok = model. AddCustomSolarZenithAngle ( sza );
```

Parameters:

`double` ***sza***

The Solar Zenith Angle of the two times to be inserted into the processing. The solar zenith angle must be between 0 and 180. The code will internally store but ignore solar zenith angles which are out of the possible range of values for a given location on a given date.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::BoxHeights

Returns the altitudes of the box heights used in Pratmo

C++

```
const std::vector<double>& Pratmo_BoxModel::BoxHeights (
) const;
```

Matlab

```
[heights,ok] = model.BoxHeights();
```

Python

```
[heights,ok] = model.BoxHeights();
```

Parameters:

`const std::vector<double>&` ***heights***

The array of heights in meters.

Back to Table of Contents

Pratmo_BoxModel::GetMaxStoichiometryConvergence

Returns the maximum iteration convergence errors for each box altitude and time step during the last call to *Solve*. The convergence errors indicate the fractional change of the value in the last iteration within Pratmo.

C++

```
bool Pratmo_BoxModel::GetMaxConvergenceErrors (
    nx2dArray<double>*          values
);
```

Matlab

```
[values,ok] = model.GetMaxConvergenceErrors ();
```

Python

```
(values,ok) = model.GetMaxConvergenceErrors ()
```

Parameters:

`nx2dArray<double>*` ***values***

The 2d array of convergence error values. The size of the array is *BoxHeights() x LocalSolarTimes()*. Typical convergence values are better than 1.0E-10 although these can be significantly degraded when numerical round off errors become important.

`bool` ***ok / return value***

Back to Table of Contents

Pratmo_BoxModel::GetMaxStoichiometryErrors

Returns the maximum stoichiometry convergence errors for each box altitude and time step during the last call to *Solve*. The stoichiometry errors indicate how closely the equation that relates the rate of change of number density to production and loss terms was solved.

C++

```
bool Pratmo_BoxModel::GetMaxStoichiometryErrors(
    nx2dArray<double>*          values
);
```

Matlab

```
[values,ok] = model.GetMaxStoichiometryErrors ();
```

Python

```
(values,ok) = model.GetMaxStoichiometryErrors ()
```

Parameters:

`nx2dArray<double>*` ***values***

The 2d array of stoichiometry error values. The size of the array is *BoxHeights() x LocalSolarTimes()*. Typical convergence values are better than 1.0E-10 although these can be significantly degraded when numerical round off errors become important.

`bool` ***ok / return value***

Back to Table of Contents

Pratmo_BoxModel::GetSpeciesConvergenceProfile

Returns the profile of fractional differences between the start and end values for a given radical species during the last call to *Solve*. This value indicates how closely the solution has achieved stability. The code will try to exceed the convergence set in the last call to *SetDailyConvergenceRatio,* typically around 0.5%, but the code may fail to reach the desired goal, especially at lower altitudes.

C++

```
std::vector<double>
Pratmo_BoxModel::GetSpeciesConvergenceProfile (
    std::string                 radicalname
);
```

Matlab

```
[values] = model. GetSpeciesConvergenceProfile (radicalname);
```

Python

```
(values) = model. GetSpeciesConvergenceProfile (radicalname)
```

Parameters:

`std::string` ***radicalname***

The name of the radical species for which the convergence profile is required.

`std::vector<double>` ***return value***

Returns the array of convergence fractions for the given species. One entry is provided for each altitude specified by the box heights (see *BoxHeights*).

Back to Table of Contents

Pratmo_BoxModel::GetSpecies

Returns the values calculated for the requested species for each box altitude and time step during the last call to *Solve*. Pratmo supports over 110 species, the names of which can be found by calling *GetVariableNames*. A few important values are 'NO2', 'NO', 'NOY' and 'M' where M is the atmospheric number density.

C++

```
bool Pratmo_BoxModel::GetSpecies (
    std::string                 name,
    nx2dArray<double>*          values
);
```

Matlab

```
[values,ok] = model.GetSpecies( name );
```

Python

```
(values,ok) = model.GetSpecies( name )
```

Parameters:

`std::string` ***name***
: The name of the variable to return. Must be capitalized.

`nx2dArray<double>*` ***values***
: The 2d array of values. The size of the array is *BoxHeights() x LocalSolarTimes()*.

Returns the radiance of the specified entry in the 2-D radiance array calculated by the last call to *CalculateRadiance*. It will return NaN if the index is invalid.

Back to Table of Contents

Pratmo_BoxModel::GetVariableNames

Return the names of all the variables stored inside the pratmo model. The user can select variables by the internal regions used within Pratmo or can provide an empty *regionfilter* string to select all regions.

C++

```
std::vector<std::string> Pratmo_BoxModel::GetVariableNames (
    std::string                 regionfilter
);
```

Matlab

```
[names,ok] = model.GetVariableNames( regionfilter );
```

Python

```
(names,ok) = model.GetVariableNames( regionfilter );
```

Parameters:

`std::string` ***regionfilter***

A string used to select the names of variables in the 5 regions of the pratmo model. An empty string will return the names in all regions. The function will return variables for a given region if the Id code given in the table below can be found in the string.

Id	Description	Comment
A	Atmospheric Section	Constant
R	Radical Section	Changed by Pratmo
L	Long-Lived Section	Constant
M	Miscellaneous Section	mixed
J	Photo-dissociation Section	Changed by Pratmo

For example, "ARL" will return variables in the Atmospheric, Radical and Long-Lived sections.

`std::vector<std::string>` ***return value***

The array of variables names. These names can be used in calls to *GetSpecies*.

Back to Table of Contents

Pratmo_BoxModel::LocalSolarTimes

Returns the array of time steps used in pratmo as an array of local solar times. The local solar time varies between 0 and 24 where 0 is midnight, 6 is nominal dawn, 12 is noon and 18 is nominal dusk. The pratmo engine only saves information for the last day processed, (see *SetNumDaysForConvergence*).

C++

```
std::vector<double> Pratmo_BoxModel::LocalSolarTimes (
) const;
```

Matlab

```
[lst,ok] = model.LocalSolarTimes ();
```

Python

```
[lst,ok] = model.LocalSolarTimes ();
```

Parameters:

`std::vector<double>` ***lst***

The array of local solar times used within Pratmo.

Back to Table of Contents

Pratmo_BoxModel::SetAlbedo

Sets the albedo used by the internal radiative transfer engine.

C++

```
bool Pratmo_BoxModel::SetAlbedo (
    double                          albedo
);
```

Matlab

```
ok = model. SetAlbedo( albedo );
```

Python

```
ok = model. SetAlbedo( albedo );
```

Parameters:

`double` ***albedo***

The albedo to be used in the internal radiative transfer engine. This will be a value between 0 and 1. The default is 0.3

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::SetAtmosphericHeights

Sets the heights within the pratmo box model that will be used to cache all atmospheric species and will be used by the internal radiative transfer engine. This will typically go from 0 to 80 km in 1 or 2 km steps. The default is the height of the 40 pressure levels within the internal atmospheric density model.

C++

```
bool Pratmo_BoxModel::SetAtmosphericHeights (
    const std::vector<double>& heights
);
```

Matlab

```
ok = model. SetAtmosphericHeights ( heights );
```

Python

```
ok = model. SetAtmosphericHeights ( heights );
```

Parameters:

`const std::vector<double>&` ***heights***

The array of heights in meters climatology that will be used for the internally cached atmospheric state and radiative transfer calculations. The heights should be specified in ascending order.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::SetBoxHeights

Sets the heights in meters of the photochemical boxes used within the Pratmo box model. The chemical and photochemical composition of the atmosphere is calculated at this array of heights. Typical values range from 0 to 60 km in 1 or 2 km steps. The default is the height of the 25 pressure levels evenly distributed between ~10 and ~60 km in the internal atmospheric density model.

C++

```
bool Pratmo_BoxModel::SetBoxHeights(
    const std::vector<double>&  heights
);
```

Matlab

```
ok = model.SetBoxHeights(heights);
```

Python

```
ok = model.SetBoxHeights(heights);
```

Parameters:

`const std::vector<double>&` ***heights***

The heights of the boxes in meters. These should be entered in ascending order. This array defines the first dimension of species retrieved through subsequent calls to *GetSpecies* after calling *Solve*.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::SetClimatology

Used to change an internal climatology within the Pratmo box model.

C++

```
bool Pratmo_BoxModel::SetClimatology(
    std::string                 name,
    ISKClimatology*             climatology,
);
```

Matlab

```
ok = model.SetClimatology ( name, climatology);
```

Python

```
ok = model.SetClimatology ( name, climatology);
```

Parameters:

`std::string` ***name***

The name of the climatology within Pratmo that is being changed. A list of valid, case-sensitive, names is given in the table below.

`ISKClimatology&` ***climatology***

The climatology of the *species* being changed. This **ISKClimatology** instance should support the *CLIMATOLOGY_HANDLE* specific for each species indicated in the table below by this number density.

`bool` ***ok / return value***

Returns true if the underlying object is valid.

Climatologies Supported by Pratmo

The following table lists all of the climatologies that can be changed within Pratmo by the user through the **SetClimatology** function. The first column identifies the case-sensitive string Pratmo uses to identify the climatology. The second column provides a brief description and the 3rd column states which *CLIMATOLOGY HANDLE* the climatology object must support.

Name	Description	ISKClimatology instance must support
'T'	Temperature profile in Kelvins	SKCLIMATOLOGY_TEMPERATURE_K
'P'	Pressure profile in Pascals.	SKCLIMATOLOGY_PRESSURE_PA
'DM'	Air number density in molecules/cm3	SKCLIMATOLOGY_AIRNUMBERDENSITY_CM3
'JH2O'	H2O photodissociation. Default was calculated by Chris McLinden.	SKCLIMATOLOGY_JH2O
'ASA'	Aerosol Surface Area in µm2/cm3	SKCLIMATOLOGY_AEROSOLSURFACEAREA_UM2PerCM3
'O3'	Mixing ratio profile.	SKCLIMATOLOGY_O3_VMR
'N2O'	Mixing ratio profile.	SKCLIMATOLOGY_N2O_VMR
'NOY'	Mixing ratio profile.	SKCLIMATOLOGY_NOY_VMR
'CO2'	Mixing ratio profile. Default is constant 314 ppm.	SKCLIMATOLOGY_CO2_VMR
'N2'	Mixing ratio profile. Default is constant 0.78084.	SKCLIMATOLOGY_N2_VMR
'O2'	Mixing ratio profile. Default is constant 0.20948.	SKCLIMATOLOGY_O2_VMR
'CO'	Mixing ratio profile. Default is 30 ppb.	SKCLIMATOLOGY_CO_VMR
'CF2CL2'	Mixing ratio profile. Default is 100 ppt.	SKCLIMATOLOGY_CF2CL2_VMR
'CFCL3'	Mixing ratio profile. Default is 100 ppt.	SKCLIMATOLOGY_CFCL3_VMR
'CCL4'	Mixing ratio profile. Default is 100 ppt.	SKCLIMATOLOGY_CCL4_VMR
'MECL'	Mixing ratio profile. Default is 100 ppt.	SKCLIMATOLOGY_MECL_VMR
'NH3'	Mixing ratio profile. Default is 0.	SKCLIMATOLOGY_NH3_VMR
'C5H8'	Mixing ratio profile. Default is 0.	SKCLIMATOLOGY_C5H8_VMR
'CH3BR'	Mixing ratio profile. Default is 10 ppt.	SKCLIMATOLOGY_CH3BR_VMR
'XXX'	Mixing ratio profile. Default is 0.	SKCLIMATOLOGY_XXX_VMR
'H2'	Mixing ratio profile. Default is 0.5 ppm	SKCLIMATOLOGY_H2_VMR
'CH4'	Mixing ratio profile. Default is from an n2o scaling.	SKCLIMATOLOGY_CH4_VMR
'H2O'	Mixing ratio profile. Default is from an n2o scaling	SKCLIMATOLOGY_H2O_VMR
'BRY'	Mixing ratio profile. Default is from an n2o scaling	SKCLIMATOLOGY_BRY_VMR
'CLY'	Mixing ratio profile. Default is from an n2o scaling	SKCLIMATOLOGY_CLY_VMR
'CH3CL'	Mixing ratio profile. Default is from an n2o scaling	SKCLIMATOLOGY_CH3CL_VMR

Back to Table of Contents

Pratmo_BoxModel::SetNominalTimeSteps

Sets the number of time steps used in the Pratmo model. Pratmo splits a 24 hour interval into a sequence of this many time steps. The time steps are evenly spaced in the cosine of solar zenith angle as this gives better temporal resolution than equally spaced time steps in the dawn/dusk sectors where many chemicals change rapidly. Please note that the number is nominal and the box model usually adds a few more points to deal with various issues during its execution of *Solve*. Users can also add extra time steps by calling *AddCustomLocalSolarTime* or *AddCustomSolarZenithAngle*.

C++

```
bool Pratmo_BoxModel::SetNominalTimeSteps(
    size_t                          numtimesteps
);
```

Matlab

```
ok = model.SetNominalTimeSteps ( numtimesteps );
```

Python

```
ok = model.SetNominalTimeSteps ( numtimesteps );
```

Parameters:

`size_t` ***numtimesteps***
: The number of time steps used in a 24 hour period in the Pratmo model. The default is 48.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::SetNumDaysForConvergence

Sets the number of days to run the pratmo model. The pratmo model internally saves only the last 24 hour period and must run for several days to ensure the 24 hour period solution has converged.

C++

```
bool Pratmo_BoxModel::SetNumDaysForConvergence(
    size_t                          numdays
);
```

Matlab

```
ok = model.SetNumDaysForConvergence( numdays );
```

Python

```
ok = model.SetNumDaysForConvergence( numdays );
```

Parameters:

`size_t` ***numdays***
: The number of days to run the Pratmo model. The default is 7.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::SetDailyConvergenceRatio

Sets the fractional convergence criteria for the values of radicals at the start and end of each day. Pratmo will continue to iterate through days until all radicals meet or exceed this convergence ratio or until the maximum number of days is encountered. The value can have a significant impact upon performance. The default value is set to 0.5% (i.e. 5.0E-03). Note that pratmo struggles to meet even this requirement at lower altitudes.

C++

```
bool Pratmo_BoxModel::SetDailyConvergenceRatio (
    double                          ratio
);
```

Matlab

```
ok = model.SetDailyConvergenceRatio ( ratio );
```

Python

```
ok = model.SetDailyConvergenceRatio ( ratio );
```

Parameters:

`double` ***ratio***

The desired daily convergence ratio to be used on the next call to Solve. The default is 5.0E-03.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::SetOzoneLongLived

Instructs Pratmo to either treat ozone as a long lived species (default) or as a radical species.

C++

```
bool Pratmo_BoxModel::SetOzoneLongLived (
    bool                          islonglived
);
```

Matlab

```
ok = model. SetOzoneLongLived ( islonglived );
```

Python

```
ok = model. SetOzoneLongLived (islonglived);
```

Parameters:

`bool` ***islonglived***

If true (not equal to zero) then ozone is treated as long lived else ozone is treated as a radical.

`bool` ***ok / return value***

Returns true if successful

Back to Table of Contents

Pratmo_BoxModel::Solve

Uses the current configuration to solve the pratmo stoichiometry at the given latitude, longitude and time. The code will normally execute in a few seconds using default settings but may take substantially longer if the configuration demands it. The code is multi-threaded and tries to use all cpu resources. You may find other applications are sluggish during execution.

C++

```
int Pratmo_BoxModel::Solve (
    double                  latitude,
    double                  longitude,
    double                  mjd
);
```

Matlab

```
numbad = model.Solve( latitude, longitude, mjd);
```

Python

```
numbad = model.Solve( latitude, longitude, mjd );
```

Parameters:

`double` ***latitude***
: The latitude (-90 to +90) at which to execute pratmo.

`double` ***longitude***
: The longitude (-180 to +360) at which to execute pratmo.

`double` ***mjd***
: The modified Julian date (UTC) at which to execute pratmo. 2014-01-01 00:00:00 UTC corresponds to an mjd of 56658.0

`int` ***return value***
: Returns the number of box heights that failed to converge. A negative value implies an internal error within the pratmo model. A value of 0 implies all box heights successfully converged. A value greater than 0 indicates the number of box heights that failed to converge. Beware that normal execution of pratmo can returns heights which are not fully converged. The user can check the convergence of specific species with calls to *GetSpeciesConvergenceProfile*

Back to Table of Contents

Pratmo_BoxModel::SZA

Returns the array of time steps used in pratmo as an array of local solar zenith angles. The local solar zenith angles varies between 0 and 180. The pratmo engine only saves information for the last day processed.

C++

```
const std::vector<double> Pratmo_BoxModel::SZA (
) const;
```

Matlab

```
[sza,ok] = model.SZA ();
```

Python

```
[sza,ok] = model.SZA ();
```

Parameters:

`const std::vector<double>` ***sza***

The array of local solar angles used within Pratmo. This will include valid custom solar zenith angles added by the user but exclude invalid solar zenith angles added by the user.

Back to Table of Contents

Pratmo_BoxModel:: ASA_To_Extinction

A helper function used to convert aerosol surface area expressed in $\mu m^2 cm^{-2}$ to extinction per centimeter at a specified wavenumber. This code uses a log normal distribution of H_2S0_4 sulphate (75% by weight) aerosol. The mode radius (default 0.08 μm) and mode width (default 1.6) are set by the last call to ASA_RG_Width. This function is only available in Matlab. Please note that this function does not change any settings within the Pratmo model.

Matlab

```
[extinction] = model.ASA_To_Extinction( wavenumber, asa);
```

Parameters:

double ***wavenumber***

The wavenumber (cm-1) at which to perform the conversion.

double ***asa***

The aerosol surface area in $\mu m^2 cm^{-2}$.

double ***extinction/return value***

Returns the extinction per cm of the specified aerosol surface area at the specified wavenumber. A value of NaN value implies an internal error.

Back to Table of Contents

Pratmo_BoxModel:: Extinction_To_ASA

A helper function used to convert aerosol extinction to aerosol surface area at a specified wavenumber. This is the reciprocal function to ASA_To_Extinction. The code uses a log normal distribution of H_2S0_4 sulphate (75% by weight) aerosol. The mode radius (default 0.08 μm) and mode width (default 1.6) are set by the last call to ASA_RG_Width. This function is only available in Matlab. Please note that this function does not change any settings within the Pratmo model.

Matlab

```
[asa] = model.Extinction_To_ASA( wavenumber, extinction );
```

Parameters:

double ***wavenumber***

The wavenumber (cm-1) at which to perform the conversion.

double ***extinction***

Returns the extinction per cm of the specified aerosol surface area at the specified wavenumber. A value of NaN value implies an internal error.

double ***asa/return value***

Returns the aerosol surface area in $\mu m^2 cm^{-2}$. A value of NaN value implies an internal error

Back to Table of Contents

Pratmo_BoxModel:: ASA_RG_Width

Sets the mode width and mode radius used by the two aerosol surface area conversion functions, ASA_To_Extinction and Extinction_To_ASA. The aerosol surface area helper functions use a log normal distribution of H_2S0_4 sulphate (75% by weight) aerosol. The mode radius by default is 0.08 μm and mode width by default is 1.6. This function is only available in Matlab. Please note that this function does not change any settings within the Pratmo model.

Matlab

```
[ok] = model.ASA_RG_Width( moderadius, modewidth );
```

Parameters:

`double` ***moderadius***

The mode radius in μm to be used in subsequent aerosol area helper functions.

`double` ***modewidth***

The mode width to be used in subsequent aerosol area helper functions.

`bool` ***ok/return value***

Returns true if successful.

Back to Table of Contents

Installation

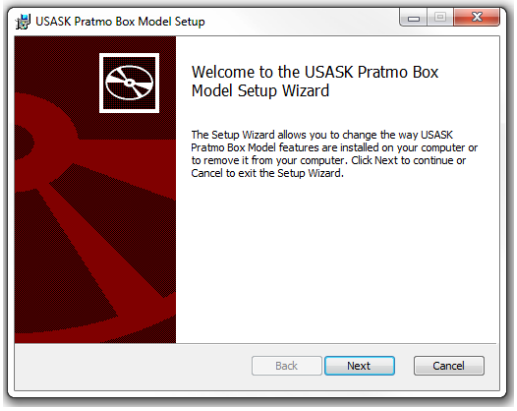
A simple *click and install* installation script is available for 64 bit windows machines. Linux binary installations are not yet available. Linux users must currently checkout the code base and build all the necessary packages which is beyond the scope of this document. We do build and run the code on various Linux systems including the Canadian High Performance Computing Facility at WestGrid so if you must have a Linux system then drop us an email and we can probably point you in the right direction.

Windows x64 Installation

The Pratmo_BoxModel code was built with the Visual Studio 2012 C++ compiler. Installation requires the following components:

1. Visual Studio 2012 Redistributables.
2. SasktranIF from our website
3. Pratmo Box Model from our website

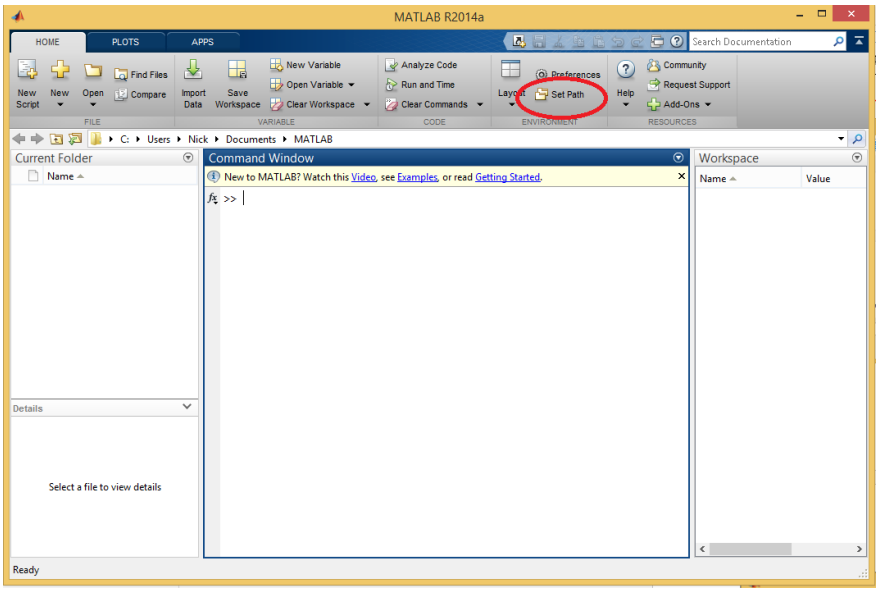
SasktranIF and the Pratmo Box Model can be downloaded from our website using the **Software** tab. Note the various installers will require administrator privilege.



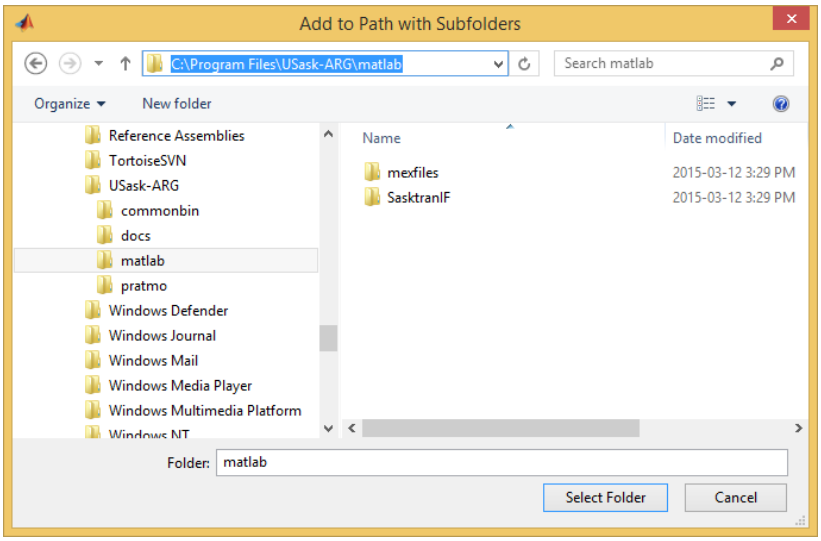
Click through the various stages and install the software. After installation you need to manually configure Matlab paths if you wish to use the Matlab interfaces, see next section.

Matlab Extra Installation and Configuration

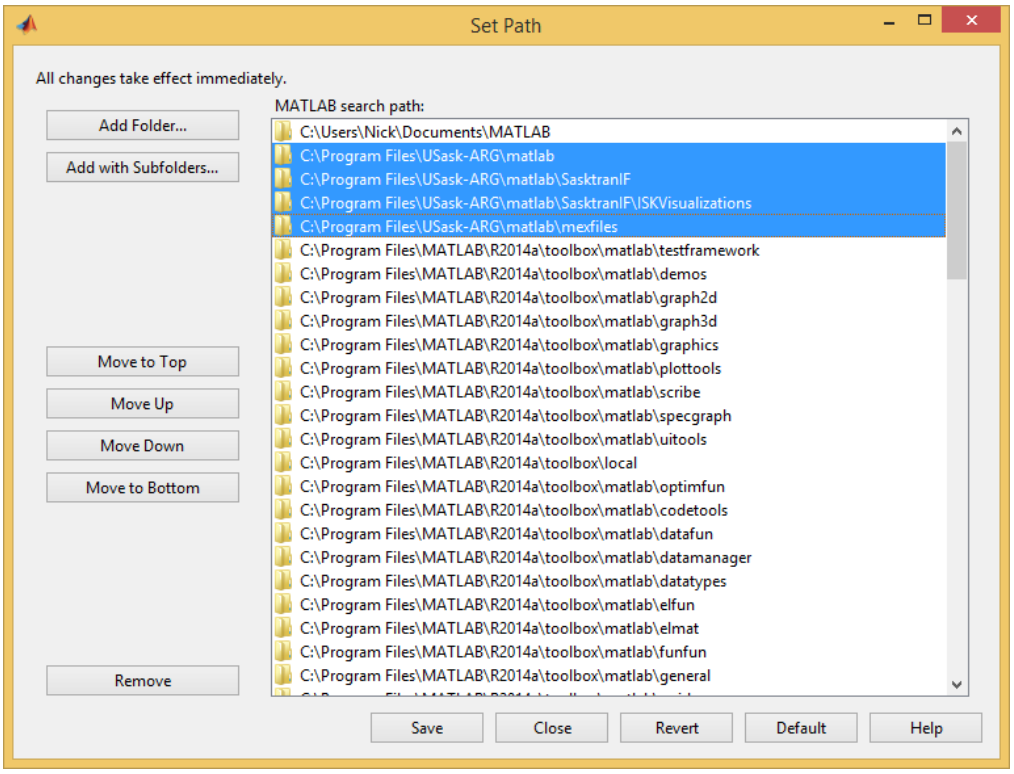
Our installation script installs Matlab software but does not update the Matlab search path. This must be done manually. Start Matlab and select **Set Path**



Select option **Add with Subfolders** and use the directory navigation dialog



to add directory `C:\Program Files\USask-ARG\matlab` and its sub-folders (including the pratmo_boxmodel directory) to the Matlab search path. Note that you will have more folders than shown above. It is okay to select all of the folders under the Matlab folder.



Make sure you **Save** the settings. Note you may have to launch Matlab with Administrator privilege if you cannot save the path (right click on the Matlab launch button and choose *Run As Administrator* option)

Appendix A: Pratmo Theory:

The pratmo model is a photochemical model. The rate of change of the number density of a species, n_i is given by,

$$\frac{dn_i(t)}{dt} = P_i(n_1, n_2 \dots n_m, t) - L_i(n_1, n_2 \dots n_m, t)$$

Where P and L are production and loss rates and there are m unknowns. This is a standard equation of the form

$$\frac{dy}{dt} = f(y, t)$$

Which can be solved by the backward (implicit) Euler technique,

$$\frac{(y_{t+\Delta t} - y_t)}{\Delta t} - f(y_{t+\Delta t}, t + \Delta t) = 0$$

Or for our case

$$\frac{n_i(t + \Delta t) - n_i(t)}{\Delta t} - [P_i(\boldsymbol{n}(t + \Delta t), t + \Delta t) - L_i(\boldsymbol{n}(t + \Delta t), t + \Delta t)] = 0$$

This equation can be solved for $n_i(t)$ using a Newton-Raphson multivariate problem: given,

$$\begin{aligned} &f_1(x_1, x_2 \dots x_m) = 0 \\ &f_2(x_1, x_2 \dots x_m) = 0 \\ &\vdots \\ &f_m(x_1, x_2 \dots x_m) = 0 \end{aligned}$$

Then this can be iteratively solved by using the Newton-Raphson substitution,

$$\boldsymbol{x} \Leftarrow \boldsymbol{x} - J^{-1} f(\boldsymbol{x})$$

Where J is the Jacobian

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_m} \\ \vdots & \dots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_m} \end{pmatrix}$$

In this case,

$$f_i \equiv \frac{n_i(t + \Delta t) - n_i(t)}{\Delta t} - [P_i(\boldsymbol{n}(t + \Delta t), t + \Delta t) - L_i(\boldsymbol{n}(t + \Delta t), t + \Delta t)]$$

and $\boldsymbol{x}$ is given by $\boldsymbol{n}(t + \Delta t)$.

Convergence Criteria

The pratmo application uses an initial estimate of $\boldsymbol{n}(t+\Delta t)$ to calculate f and J. The inverse equation is solved and the value of $\boldsymbol{n}(t+\Delta t)$ is updated. The process is repeated until the solution has converged.

Convergence is tested by examining how close f is to zero. We normalize the comparison by dividing f by $P+L$ and demanding that the ratio be less than 1.0E-10. In practice we find that pratmo covers such a diverse range of number densities and production/loss rates that the numerical precision of the machine (64 bit doubles) occasionally comes into play and we have to use less stringent criteria.

In general the convergence limit is given by

$$\frac{n(t+\Delta t)-n(t)}{P+L}-\frac{(P-L)\,\Delta t}{P+L} < 10^{-10}$$

In the above equation, $n(t+\Delta t)$ and $n(t)$ are usually two very large numbers whose difference is taken. This difference gets smaller as Δt gets smaller and is more susceptible to numerical round-off error. Terms P and L are calculated quite precisely from the product of reactions rates and number densities and do not normally suffer from numerical precision.

For a concrete example consider an example we encountered in pratmo relating to the equations for BrCl in the nighttime sector. Under these conditions we had $P = 0.00001$, $L = 0.0000$, $\Delta t = 0.001$, $n(t+\Delta t) = 86456.6782364$ and $n(t) = 86456.6782364$. The value of the production term, $P.\Delta t$, was lower than the numerical precision of $n(t+\Delta t)$ and $n(t)$. The net result was that the LHS of the equation evaluates to 0 while the RHS evaluates to 0.001. Thus the numerical precision of the machine dramatically altered the reported precision from around 1.0E-10 to 1.0E-03.

We can quantify the numerical round-off error into the convergence calculation by introducing the round off error, $\Delta e = n(t)\epsilon$

$$\frac{n(t+\Delta t)-n(t)+\Delta e}{P+L}=\frac{(P-L)\,\Delta t}{P+L}$$

If we are very close to a solution where numerical round-off is an issue then $n(t+\Delta t) = n(t)$ and we get

$$\Delta e = (P-L)\,\Delta t$$

And the best convergence accuracy we can hope to achieve is given by,

$$\text{Convergence Limit} = \frac{\Delta e}{(P - L)\,\Delta t}$$