The MOSTLEE Model

A Model Of Sources, Transport, Loss, and Emission in Exospheres

Dr. Matthew H. Burger

Last Updated: August 1, 2012

Contents

List of Tables

List of Figures

1 ✓ AtomicData/get_gvalue_3.1.pro

Summary

Creates a g-value structure using saved atomic data.

Calling Procedure

```
gval = get_gvalue, atom, a, path=path
```

Inputs

1 atom = species being modeled
2 a = heliocentric distance
3 path = path where g-values are saved (optional)

Outputs

Function returns a structure with the fields:

- species = same as the atom input
- a = same as the input. Units: AU.
- wavelength = pointer to an array of length *nlam* with wavelengths of resonance transitions. Units: Å.
- v = pointer to an array of length *nvel* with the radial velocities at which the g-value was calculated Units: km cm^{-2} s^{-1}.
- g = pointer to an array of size $nvel \times nlam$ with g-values as function of radial velocity for each resonant transition scaled to distance a . Units: photons s^{-1}.
- radaccel = pointer to an array of size *nvel* with the radiation acceleration at each wavelength. See radiation_pressure and Figure 1. Units: km s^{-1}.
- If the species is not found, a warning is printed and the g-value is set to 0.

Model Procedures Called

None.

Notes

1 The g-values are saved in `$BASEPATH/Work/AtomicData/g-values/'`
2 The structure saved in the g-value files are created with `set_up_gvals`.
3 See Table 1 for list of species that have g-values implemented in the code.
4 Figure **??** shows g-value at 1 AU as function of radial velocity for each species listed in Table 1.

This Page Last Updated

12 December 2011.

Species	Wavelengths (Å)	Reference
C	1560, 1657	Killen et al. (2009)
Ca^{+}	3934, 3969	Killen et al. (2009)
Ca	2722, 4227, 4567	Killen et al. (2009)
H	1215	Killen et al. (2009)
He	584	Killen et al. (2009)
K	4045	Killen et al. (2009)
Mg^{+}	2083, 2796	Killen et al. (2009)
Mg	2026, 2852,	Killen et al. (2009)
Mn	2795, 2799, 2802	XXXXX
Na	3303, 5891, 5897	Killen et al. (2009)
O	1303	Killen et al. (2009)
OH	3081, 3092	Killen et al. (2009)
S	1807, 1820	Killen et al. (2009)
Ti	3187, 3193, 3204, 3342, 3371, 3371, 3636, 3644	XXXXX

Table 1: Available g-values

2 ✓Data/SystemConstants_2.0.pro

Summary

Creates the `SystemConsts` and `DipoleConsts` structures from data stored in the *!consts* system variables.

Calling Procedure

`SystemConstants, planet, SystemConsts, DipoleConsts`

Inputs

- planet: String containing planet name. For purposes of the model, *planet* is defined as the center of the modeled system. Therefore, "sun" and "pluto" are valid planets.

Outputs

- SystemConsts: Structure containing the planetary system constants The fields in the structure are given in Table 4. The values for each object are given in Table 6.
- DipoleConsts: Structure containing the planetary dipole constants The fields in the structure are given in Table 5. The values for each object are given in Table 6.

Model Procedures Called

None.

Notes

1 The values used in this routine are saved in the files `Data/PhysicalData/PlanetaryConstants.dat` and `Data/PhsyicalData/DipoleConstants.dat`.
2 Data taken from NASA Solar System Dynamics and Wikipedia: Sun.

This Page Last Updated

9 December 2011.

3 ✓Display/determine_image_rotation_4.0.pro

Summary

Computes the rotation matrix for an image given a sub-observer longitude, sub-observer latitude, and north pole rotation angle, or a time and observer location.

Calling Procedure

```
M = determine_image_rotation, input, format
```

Inputs

1 input = input structure
2 format = format structure

Outputs

Returns the rotation matrix from model coordinates to the observer's viewpoint.

Model Procedures Called

1 `relative_position`
2 `utc2et`
3 `rotationmatrix`
4 `rotation`

Notes

1 There are two ways to specify the observing geometry:
   1.1 Sub-Observer point:
   - format.geometry.SubObsLongitude: radians dawnward of sub-solar point
   - format.geometry.SubObsLatitude: radians northward of sub-solar point
   - format.geometry.PolarAngle: north pole rotation angle radians counter-clockwise (?) from north.

   1.2 Time and Observer location:
   - format.geometry.time: string time in ISOC format (YYYY-MM-DDTHH:MM:SS.S or YYY-DOYTHH:MM:SS.S)
   - format.geometry.observer: NAIF observer location (e.g., 'Earth', 'MESSENGER')

2 The rotation matrix is determined by the rotation angle (θ) of the sun direction ($\hat{\mathbf{x}}_\odot$) to the observer direction ($\hat{\mathbf{x}}_{obs}$) around the axis perpendicular to both vectors ($\hat{\mathbf{a}}$):

$$\hat{\mathbf{a}} = \hat{\mathbf{x}}_\odot \times \hat{\mathbf{x}}_{obs} \tag{1}$$

$$\cos\theta = \hat{\mathbf{x}}_\odot \cdot \hat{\mathbf{x}}_{obs} \tag{2}$$

$$M = \begin{pmatrix} a_x^2 + (1-a_x)^2\cos\theta & a_xa_y(1-\cos\theta) - a_z\sin\theta & a_xa_z(1-\cos\theta) + a_y\sin\theta \\ a_xa_y(1-\cos\theta) + a_z\sin\theta & a_y^2 + (1-a_y^2)\cos\theta & a_ya_z(1-\cos\theta) - a_x\sin\theta \\ a_xa_z(1-\cos\theta) - a_y\sin\theta & a_ya_z(1-\cos\theta) + a_x\sin\theta & a_z^2 + (1-a_z^2)\cos\theta \end{pmatrix} \tag{3}$$

3 If specifying Sub-Observer point, then:

$$\hat{\mathbf{x}}_{\odot} = (0, -1, 0) \quad (4)$$
$$\hat{\mathbf{x}}_{obs} = (\sin\lambda\cos\mu, -cos\lambda\cos\mu, \sin\mu) \quad (5)$$

where $\lambda = input.geometry.SubObsLongitude$ and $\mu = input.geometry.SubObsLatitude$

4 If specifying Time and Observer location, then $\hat{\mathbf{x}}_{\odot}$ and $\hat{\mathbf{x}}_{obs}$ are determined with `relative_position` using SPICE.

5 If $format.geometry.PolarAngle \neq 0$, then a rotation is taken around the new y-axis.

This Page Last Updated

13 December 2011.

4 Display/display_hull.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

5 Display/display_model_image_4.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

6 Display/emission_measure_2.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

7 Display/model_view.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

8 Display/produce_density_4.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

9 Display/produce_image_4.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

10 Display/produce_los_4.12.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

11 Display/produce_results_4.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

12 Display/produce_voronoi_image_4.3.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

13 Display/quick_look.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

14 Display/read_resultformat_4.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

15 Display/results_common_4.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

16 Display/results_density_4.5.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

17 Display/results_functions_4.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

18 Display/results_kd_tree_4.3.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

19 Display/results_packet_weighting_4.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

20 Display/results_voronoi_4.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

21 ✓Forces/accel_3.1.pro

Summary

Computes the acceleration on a packet due to the specified forces. Possible forces are gravity, radiation pressure, and Lorentz (not working).

Calling Procedure

```
a = accel(loc, input, magcood, which)
```

Inputs

1 loc = packet location structure
2 input = input structure
3 magcoords = magnetic coordinate structure (which includes out_of_shadow). See xyz_to_magcoord.
4 which = array specifying which objects are included in the calculation

Outputs

Function returns a structure with field dvdt, which is an $n \times 3$ array with a_x, a_y, a_z in units of $\mathrm{R_{plan}}$ $\mathrm{s^{-2}}$.

Model Procedures Called

1 gravity
2 radiation_pressure
3 Lorentz

Notes

1 Total acceleration:

$$\mathbf{a} = \mathbf{a}_{grav} + \mathbf{a}_{radpres} + \mathbf{a}_{lor} \tag{6}$$

2 The input structure fields `input.Forces.gravity`, `input.Forces.radpres`, and `input.Forces.Lorentz` are used to specify which forces are included in the calculations.
3 The Lorentz force, which only affects ions, does not currently work properly.

This Page Last Updated

12 December 2011.

22 ✓Forces/gravity_3.1.pro

Summary

Computes acceleration due to gravity from each object.

Calling Procedure

```
agrav = gravity(loc, input)
```

Inputs

1 loc = packet location structure
2 input = input structure

Outputs

Function returns acceleration due to gravity as a $n \times 3$ array with a_x, a_y, a_z in units of $\mathrm{R_{plan}\ s^{-2}}$.

Model Procedures Called

1 locmoon: gives the location of each object as viewed by each packet.

Notes

1 Gravity computed by:

$$\mathbf{a} = \sum_{i=0}^{nobj} \frac{GM}{r_i^2}\hat{\mathbf{r}}_\mathbf{i} \quad (7)$$

$$r_i = \sqrt{(x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2} \quad (8)$$

which is given in component form by:

$$a_x = \sum_{i=0}^{nobj} \frac{GM_i(x-x_i)}{r_i^3} \quad (9)$$

$$a_y = \sum_{i=0}^{nobj} \frac{GM_i(y-y_i)}{r_i^3} \quad (10)$$

$$a_z = \sum_{i=0}^{nobj} \frac{GM_i(z-z_i)}{r_i^3} \quad (11)$$

(12)

where the coordinates of each object (planet and satellites) are $\mathbf{r_i} = (x_i, y_i, z_i)$ are computed by `locmoon`, GM_i are stored in the *SystemConsts* variable, and the sum is only performed for objects which *input.geometry.include* = 1.

This Page Last Updated

12 December 2011

23 Forces/lorentz_2.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

24 ✓Forces/radiation_pressure_3.1.pro

Summary

Returns the radiation acceleration as function of radial velocity.

Calling Procedure

```
radiation_pressure, loc, out_of_shadow
```

Inputs

1 loc = packet location structure
2 out_of_shadow = boolean array specifying whether each packet is in the planet's shadow. *Does not currently compute satellite shadow*

Outputs

Function returns acceleration due to radiation pressure as a $n \times 3$ array with a_x, a_y, a_z in units of $\mathrm{R_{plan}}\ \mathrm{s}^{-2}$.

Model Procedures Called

None.

Notes

1 The radiation pressure is computed by:

$$\mathbf{a}_{rad}(v_r) = \sum_i \frac{hg(v_r)}{m\lambda_i}\hat{y} \tag{13}$$

where $h = 6.6260690 \times 10^{-27}$ erg s^{-1} is Planck's constant, g is the g-value as a function of radial velocity relative to the sun v_r, m is the atomic mass, and λ_i is each resonant transition for the species in question.
2 The radiation acceleration is actually calculated by the function get_gvalue and stored in the *stuff* structure in modeldriver (in the fields *stuff.radpres_v* and *stuff.radpres_const*). This is done to speed things up.
3 $\mathbf{a}_{rad}$ is directed entirely in the $+\hat{y}$ direction. I currently assume the planets' equatorial (rotational) planes are not tilted relative to their orbital planes.
4 Similarly, v_{rad} is the velocity in the $\hat{y}$ direction (which is computed relative to the planet) + *stuff.vrplanet*. *stuff.vrplanet* is computed by planet_dist and saved into the *stuff* structure in modeldriver.
5 Because the radiation acceleration is computed at discrete values by get_gvalue, the radiation acceleration for each packet is computed by linear interpolation between values.
6 If v_r is outside the velocity range in *stuff.radpres_v*, then v_r is assumed to be either the maximum or minimum value of *stuff.radpres_v*, as appropriate.
7 $\mathbf{a_{rad}} = 0$ in the shadow of a planet or satellite.
8 See Table 1 for list of species that have g-values implemented in the code.
9 Figure 1 shows radiation acceleration as function of radial velocity at 1 AU for each species listed in Table 1.

Things to do

1 Does not currently compute shadowing for satellites. If radiation pressure is turned on for a system with multiple objects, program will stop.
2 The radiation acceleration for K is wrong because I do not have the g-values for the strongest lines.

This Page Last Updated

12 December 2011.

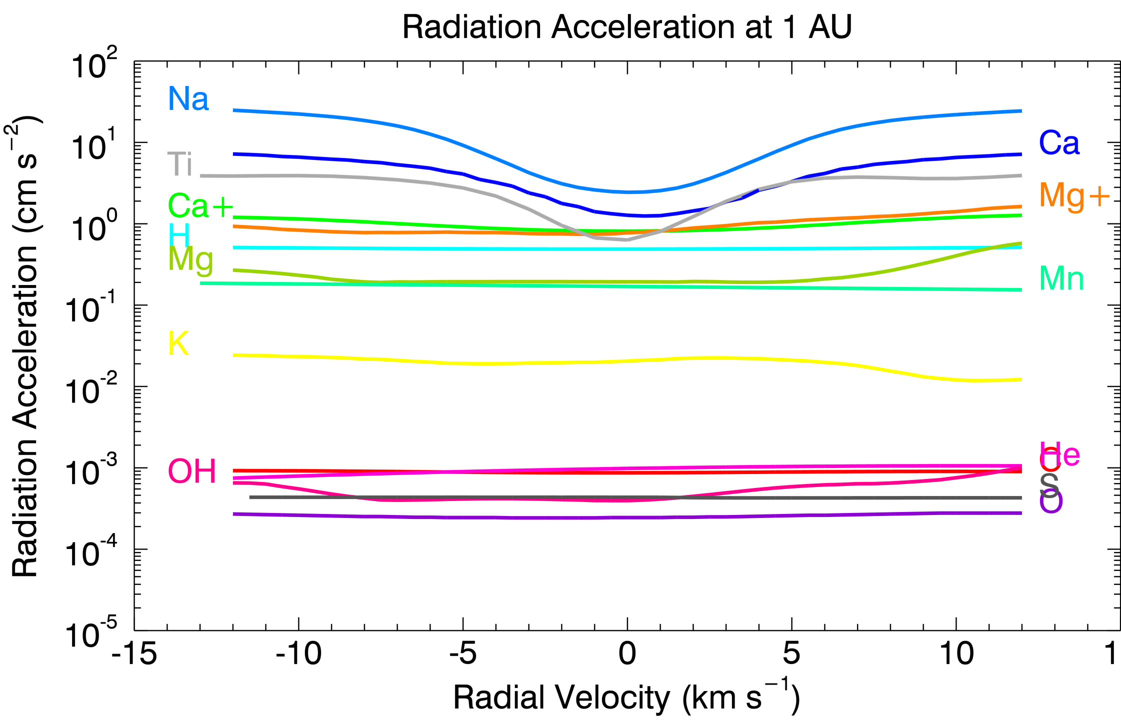

Figure 1: Radiation acceleration as function of velocity for species with available g-values.

25 Integrator/driver_3.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

26 Integrator/rk5_3.3.pro

Summary

Computes a Runge-Kutta step and estimates the error.

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

1 Based on Numerical Recipies, 3rd Edition, Ch. 17.2

2 The general form of a fifth-order Runge-Kutta formula is (NR, p. 912):

$$k_1 = hf(x_n, y_n) \tag{14}$$
$$k_2 = hf(x_n + c_2h, y_n + a_{21}k_1) \tag{15}$$
$$k_3 = hf(x_n + c_3h, y_n + a_{31}k_1 + a_{32}k_2) \tag{16}$$
$$k_4 = hf(x_n + c_4h, y_n + a_{41}k_1 + a_{42}k_2 + a_{43}k_3) \tag{17}$$
$$k_5 = hf(x_n + c_5h, y_n + a_{51}k_1 + a_{52}k_2 + a_{53}k_3 + a_{54}k_4) \tag{18}$$
$$k_6 = hf(x_n + c_6h, y_n + a_{61}k_1 + a_{62}k_3 + a_{63}k_3 + a_{64}k_4 + a_{65}k_5) \tag{19}$$
$$y_{n+1} = y_n + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4 + b_5k_5 + b_6k_6 \tag{20}$$

3 The coefficients are given in Table 2

4 (x_n, y_n) are the values at the beginning of the interval. (x_{n+1}, y_{n+1}) are the values at the end. Actually, there are 7 equations for each y_n corresponding to the 7 output values.

Table 2: 5th Order Runge-Kutta Coefficeints

5 Solving this set of differential equations:

$$\frac{dx}{dt} = v_x \tag{21}$$

$$\frac{dy}{dt} = v_y \tag{22}$$

$$\frac{dz}{dt} = v_z \tag{23}$$

$$\frac{dv_x}{dt} = F_x(x, y, z, v_x, v_y, v_z) \tag{24}$$

$$\frac{dv_y}{dt} = F_y(x, y, z, v_x, v_y, v_z) \tag{25}$$

$$\frac{dv_z}{dt} = F_z(x, y, z, v_x, v_y, v_z) \tag{26}$$

$$f = f_0 e^{-\nu(x,y,z,v_x,v_y,v_z)t} \tag{27}$$

This Page Last Updated

27 Lifetimes/JupiterPlasma_3.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

28 Lifetimes/SaturnPlasma_2.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

29 Lifetimes/ionization_rate_3.3.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

30 Lifetimes/lifetime_setup_3.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

31 Lifetimes/load_plasma_2.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

32 Lifetimes/xyz_to_magcoord_3.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

33 ModelIO/BennaPrecipitationFilename_1.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

34 ✓ModelIO/MercuryModelEndTime_1.0.pro

Summary

Computes the appropriate value of *input.options.endtime* for a Mercury model run as a function of true anomaly angle. The appropriate value is set at 4× the photoionization lifetime.

Calling Procedure

```
endtime = MercuryModelEndTime(atoms, taa)
```

Inputs

1 atoms = species to look at
2 taa = true anomaly angle in radians

Outputs

Function returns 4× the photoionization lifetime of each species at the requested true anomaly angles.

Model Procedures Called

1 planet_dist: determines Mercury's distance from the sun as function of true anomaly.
2 search_atomicdata [not in the manual yet]

Notes

None.

This Page Last Updated

12 December 2011

35 ModelIO/combine_iterations_3.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

36 ModelIO/compare_inputs_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

37 ModelIO/extract_distribution_3.3.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

38 ModelIO/extract_parameter_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

39 ModelIO/inputs_restore_3.3.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

40 ModelIO/make_generic_input_3.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

41 ModelIO/make_model_header_3.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

42 ModelIO/modeloutput_search_3.5.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

43 ModelIO/output_filename_3.5.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

44 ✓ModelIO/print_inputs_3.1.pro

Summary
Prints the contents of an input structure to the screen.

Calling Procedure
`print_inputs, input, printarr=printarr`

Inputs
1 input: either an input structure or the name on an inputfile

Outputs
1 printarr: contains a strarr with each of the input fields

Model Procedures Called
1 inputs_restore

Notes
None.

This Page Last Updated
9 December 2011.

45 SourceDistributions/PSD_distribution_1.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

46 SourceDistributions/SO2exosphere_distribution_3.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

47 SourceDistributions/SourceFlux_1.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

48 SourceDistributions/SourceRate_1.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

49 SourceDistributions/add_perturbation_2.2.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

50 SourceDistributions/angular_distribution_3.1.pro

Summary

Determines the initial angular distribution for packets.

Calling Procedure

```
angular_distribution, input, output, npack, seed
```

Inputs

1 input = input structure
2 output = output structure with *output.vx0*=initial speed distribution.
3 npack = number of packets
4 seed = seed for random number generator

Outputs

1 output = output structure with *output.vx0*, *output.vy0*, and *output.vz0* defined in units of $R_{plan}\ s^{-1}$

Model Procedures Called

1 `random_nr`
2 `RandomDeviates_1d`

Notes

1 First step: randomly choose altitude (θ) and azimuth (ϕ) angles.
  1.1 Altitude is measured from surface tangent ($\theta = 0$) to surface normal ($\theta = \pi/2$).
  1.2 The zero point for azimuth angle is not clear to me. It is possible to limit the azimuth angle, but I'm not sure exactly how the zero point is defined at each point on the surface. It is better to use the default ($0 \leq \phi \leq 2\pi$.
  1.3 Three options for *angulardist.type*: `radial`, `isotropic`, `costheta`
  1.4 Radial ejection
    1.4.1 $\theta = \pi/2$ for all packets
    1.4.2 $\phi = 0$ for all packets (azimuth angle is technically undefined for normal ejection.
  1.5 Isotropic ejection
    1.5.1 Azimuth angles are evenly distributed from $0 \rightarrow 2\pi$.
    1.5.2 $\sin\theta$ is evenly distributed between $0 \rightarrow 1$ for distributions that start at the surface (2π ejection into outward-facing hemisphere) or $-1 \rightarrow 1$ for ejection into 4π str.
  1.6 costheta ejection
    1.6.1 Azimuth angles are evenly distributed from $0 \rightarrow 2\pi$.
    1.6.2 $f(\sin\theta) = \sin^n\theta$

This Page Last Updated

20 January 2012

51 SourceDistributions/charge_exchange_perturbation_2.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

52 SourceDistributions/exosphere_distribution_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

53 SourceDistributions/show_veldist_1.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

54 ✓SourceDistributions/source_distribution_3.3.pro

Summary

Determines the initial positions and velocities for each packet. This is a driver program which calls the procuedures to determine initial spatial, speed, and angular distributions of packets.

Calling Procedure

`source_distribution, input, npack, seed, output=output`

Inputs

1 input: `input` structure
2 npack: number of packets in the output
3 seed: seed for random number generator (optional)

Outputs

1 output: an `output` structure (see below)

Model Procedures Called

1 random_nr
2 surface_distribution
3 torus_distribution
4 exosphere_distribution
5 SO2exosphere_distribution
6 PSD_distribution
7 speed_distribution
8 angular_distribution
9 add_perturbation
10 locmoon

Notes

Procuedure used:

1 Create the `output` structure (see Table 3 for description of fields). This just sets up the structure put does not put any values into the fields.
2 `output.f0` = 1 for all packets initially.
3 Set the output time:
    - If `input.options.at_once = 1`, then `output.time` is set to `input.options.endtime` for all packets.
    - If `input.options.at_once = 0`, then `output.time` is set to a random value between 0 and `input.options.endtime`.
4 Set the initial locations for packets. Possible distributions allowed in `input.SpatailDist.type` are:
    - *surface*: starts all packets at a sphere located at `input.geometry.startpoint`'s surface or a uniform exobase. See surface_distribution
    - *torus*: starts all packets in a uniform torus around `input.geometry.startpoint`. See torus_distribution

- *exosphere*: starts all the packets in an exosphere over the surface of `input.geometry.startpoint` See exosphere_distribution
- *SO2 exosphere*: Uses Vincent Dol's exospheric source model. See SO2exosphere_distribution
- *PSD distribution*: Approximation to the ion-enhanced PSD distribution described by Burger et al. (2010). See PSD_distribution

These routines set `output.x0`, `output.y0`, and `output.z0` in a coordinate system centered on `input.geometry.startpoint`. A transformation to the `input.geometry.planet`-cenetered coordinate system is made later, if necessary.

5 Set the initial speed distribution. See speed_distribution. This routine sets the intitial speed (and possibly the components of the speed) in units of R_{plan} s^{-1}. Satellite orbital velocity is not included yet.

6 Set the angular disitribution, if necessary. See angular_distribution. This routine sets `output.vx0`, `output.vy0`, and `output.vz0` (if it has not already been done) in a coordinate system centered on `input.geometry.startpoint`. A transformation to the `input.geometry.planet`-cenetered coordinate system is made later, if necessary.

7 Add a perturbation to the intial velocities, if requested. See add_perturbation.

8 Make the conversion from the `input.geometry.startpoint`-centered system to the `input.geometry.planet`-cenetered system if necessary.

8.1 Move the packets out to their starting distance and rescale from $R_{st.pt.}$ to R_{plan}:

$$x = x_0 \times \texttt{SystemConsts.radius} \tag{28}$$
$$y = y_0 \times \texttt{SystemConsts.radius} + \texttt{SystemConsts.a} \tag{29}$$
$$z = z_0 \times \texttt{SystemConsts.radius} \tag{30}$$

This places each packet at the satellite's orbital distance at orbital phase $= 0°$ (local midnight).

8.2 Add in the orbital velocity, if `input.options.motion=1`:

$$v_x = vx_0 \tag{31}$$
$$v_y = vy_0 - \texttt{SystemConsts.orbvel} \tag{32}$$
$$v_z = vz_0 \tag{33}$$

For a packet at orbital phase $= 0°$, the direction of orbital motion is in the $-\hat{y}$ direction.

8.3 Determine `output.phi0`, the orbital phase of the satellite at the start time of each packet (`output.time` seconds ago). See locmoon.

8.4 Rotate the packets to the proper location in planet-centered coordinates:

$$output.x = x\cos\phi_0 - y\sin\phi_0 \tag{34}$$
$$output.y = x\sin\phi_0 + y\cos\phi_0 \tag{35}$$
$$output.z = z \tag{36}$$
$$output.vx = v_x\cos\phi_0 - v_y\sin\phi_0 \tag{37}$$
$$output.vy = v_x\sin\phi_0 + v_y\cos\phi_0 \tag{38}$$
$$output.vz = v_z \tag{39}$$

9 `output.x0`, etc., are in startpoint-centered coordinates. These are not used in any of the model computations.

10 `output.x`, etc., are in planet-centered coordinates. These are used in all of the model computations.

This Page Last Updated

20 December 2011.

55 SourceDistributions/speed_distribution_3.3.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

56 SourceDistributions/speed_dists_2.1.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

57 SourceDistributions/surface_distribution_3.2.pro

Summary

Distribute packets about a sphere with radius r = `input.SpatialDist.exobase`

Calling Procedure

`surface_distribution, input, output, npack, seed`

Inputs

1 input: `input` structure
2 output: `output` structure defined in source_distribution (Table 3)
3 npack: number of packets in the output
4 seed: seed for random number generator (optional)

Outputs

1 output: `output.x0`, `output.y0`, and `output.z0` fields are populated.

Model Procedures Called

1 RandomDeviates_2d
2 random_nr

Notes

There are several ways to use this option:

1 Use a predefined map file:
  1.1 The following lines must be included in the *input* file.
  ```
  SpatialDist.type = surface
  SpatialDist.use_map = 1
  SpatialDist.mapfile = path_to_mapfile
  ```
  1.2 *mapfile* is an IDL savefile with a variable called `sourcemap` which is a structure defined by:
  ```
  sourcemap = {longitude:ptr_new(longitude), latitude:ptr_new(latitude), $
    map:ptr_new(map)}
  ```
  where:
  - `longitude` is an array of longitude (λ) values in radians. See Section **??** for definition of longitude. The acceptable range of values is $0 \leq \lambda \leq 2\pi$. I generally set this to:
    `longitude = fltarr(361)*!dtor}`
  - `latitude` is an array of latitude (μ) values in radians. The acceptable range of values is $-\pi/2 \leq \mu \leq \pi/2$. I generally set this to
    `latitude = fltarr(181)*!dtor - !pi/2.`
  - `map` is an array of size (n_λ, n_μ) containing the relative source flux (particles per unit area) as a function of λ and μ. This does not need to be absolute flux since it is normalized to the total number of packets ejected.

2 Uniform ejection within a specified range
  2.1 The following lines must be included in the input file:
  ```
  SpatialDist.type = surface
  ```

```
SpatialDist.use_map = 0
```

The following lines are optional:

```
SpatialDist.longitude0 = minlong
SpatialDist.longitude1 = maxlong
SpatialDist.latitude0 = minlat
SpatialDist.latitude1 = maxlat
```

2.2 `minlong` (λ_{min}) and `maxlong` (λ_{max}) are the minimum and maximum longitude values with $0 \leq \lambda_{min}, \lambda_{max} \leq 2\pi$.
- If $\lambda_{min} < \lambda_{max}$, then packets are randomly distributed with $\lambda_{min} \leq \lambda < \lambda_{max}$ (Figure 2a).
- If $\lambda_{min} > \lambda_{max}$, then packets are randomly distributed with $\lambda_{max} \leq \lambda < \lambda_{max} + 2\pi$ (Figure 2b).
- If $\lambda_{min} = \lambda_{max}$ then all packets are set to $\lambda = \lambda_{min}$.

2.3 `minlat` (μ_{min}) and `maxlat` (μ_{max}) are the minimum and maximum values of latitude with $\pi/2 \leq \mu_{min} \leq \mu_{max} \leq \pi/2$.

2.4 Default values:

$$\begin{aligned}
\lambda_{min} &= 0 \\
\lambda_{max} &= 2\pi \\
\mu_{min} &= -\pi/2 \\
\mu_{max} &= \pi/2
\end{aligned}$$

3 Choosing random values:

3.1 Longitude values are evenly distributed between λ_{min} and λ_{max}, so we can choose values with:

$$\lambda = \lambda_{min} + (\lambda_{max} - \lambda_{min})R \tag{40}$$

where R is a random deviate from a uniform distribution with $0 \leq R < 1$.

3.2 Latitude values are not evenly distributed; however, $\sin\mu$ are, so we can choose values with:

$$\sin\mu = \sin\mu_{min} + (\sin\mu_{max} - \sin\mu_{min})R \tag{41}$$

where R is a random number as above and $-1 \leq \sin\mu < 1$.

4 If `input.geometry.StartPoint = input.geometry.planet` then:
- local noon: $\lambda = 0°$, $(x, y, z) = (0, -1, 0)$
- local dusk: $\lambda = 90°$, $(x, y, z) = (1, 0, 0)$
- local midnight: $\lambda = 180°$, $(x, y, z) = (0, 1, 0)$
- local dawn: $\lambda = 270°$, $(x, y, z) = (-1, 0, 0)$

$$\begin{aligned}
x_0 &= r_0 \sin\lambda \cos\mu && (42) \\
y_0 &= -r_0 \cos\lambda \sin\mu && (43) \\
z_0 &= r_0 \sin\mu && (44)
\end{aligned}$$

where r_0 = `input.SpatialDist.exobase`

5 If `input.geometry.StartPoint` $\neq$ `input.geometry.planet` then:
- sub-planet: $\lambda = 0°$, $(x, y, z) = (0, -1, 0)$
- leading side: $\lambda = 90°$, $(x, y, z) = (1, 0, 0)$

- anti-planet: $\lambda = 180°$, $(x, y, z) = (0, 1, 0)$
- trailing side: $\lambda = 270°$, $(x, y, z) = (-1, 0, 0)$

$$x_0 = -r_0 \sin\lambda \cos\mu \tag{45}$$
$$y_0 = -r_0 \cos\lambda \cos\mu \tag{46}$$
$$z_0 = r_0 \sin\mu \tag{47}$$

This Page Last Updated

21 December 2011.

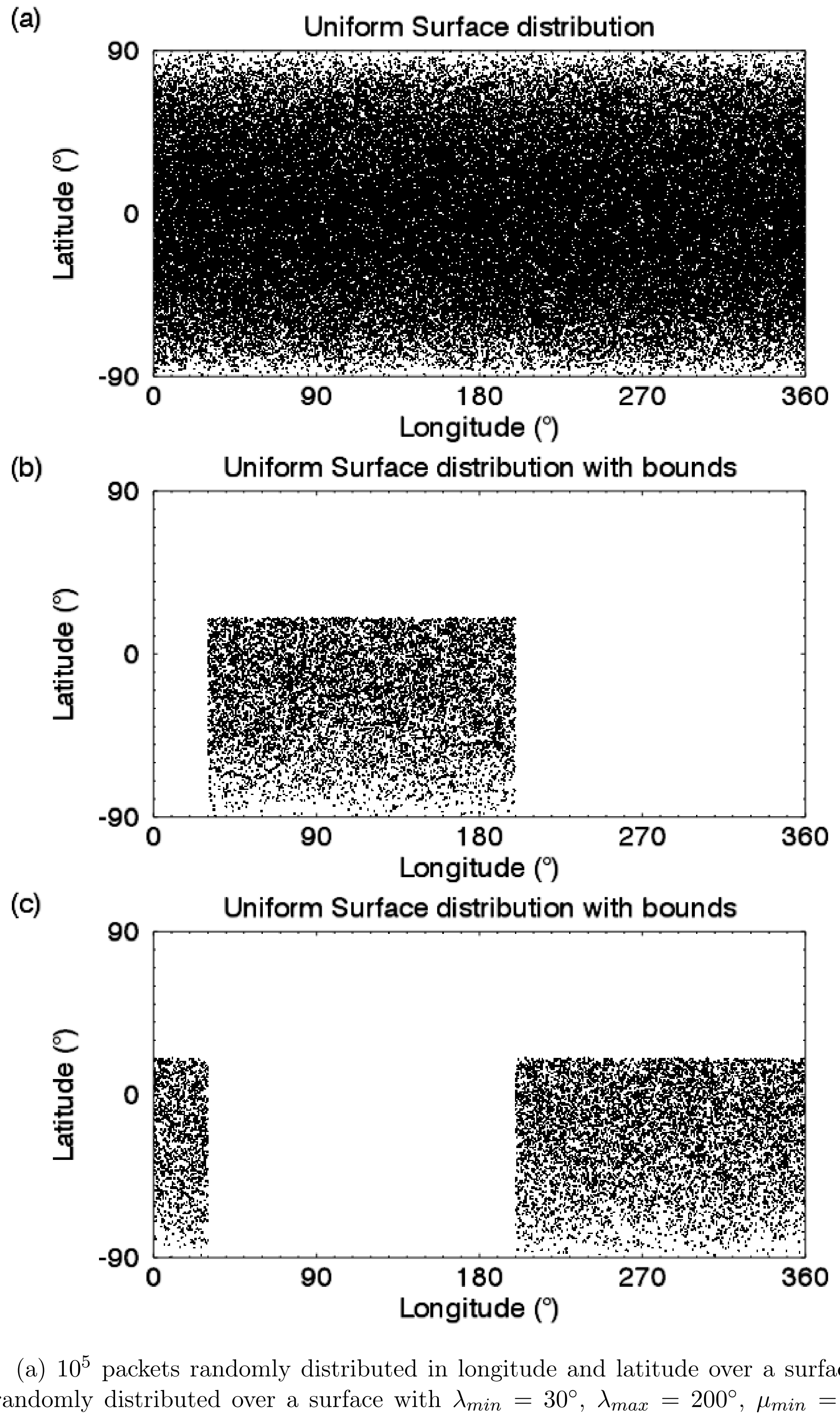

Figure 2: (a) 10^5 packets randomly distributed in longitude and latitude over a surface. (b) 10^4 packets randomly distributed over a surface with $\lambda_{min} = 30°$, $\lambda_{max} = 200°$, $\mu_{min} = -90°$, and $\mu_{max} = 20°$. (c) 10^4 packets randomly distributed over a surface with $\lambda_{min} = 200°$, $\lambda_{max} = 30°$, $\mu_{min} = -90°$, and $\mu_{max} = 20°$.

58 SourceDistributions/surface_temperature_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

59 SourceDistributions/torus_distribution_2.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

60 loc_operations_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

61 ✓locmoon_1.0.pro

Summary

Calculates the coordinates of each moon given a "final" orbital longitude and a time difference. Determines where a moon was *time* seconds ago.

Calling Procedure

`locmoon, time, theta0, radius, orbrate, x=x, y=y, z=z, ang=ang`

Inputs

- time: time before moon was at *theta0* (seconds)
- theta0: final orbital longitude of each moon (radians). Corresponds to the position at *time=0*.
- radius: orbital radius of each moon (R_{plan})
- orbrate: angular speed of each moon (radians/sec)

Outputs

- x: x-position of each moon relative to the planet *time* seconds it was before it was at *theta0* (R_{plan})
- y: y-position of each moon relative to the planet *time* seconds it was before it was at *theta0* (R_{plan})
- z: z-position of each moon relative to the planet *time* seconds it was before it was at *theta0* (R_{plan})
- ang: orbital longitude *time* seconds ago (radians)

Model Procedures Called

None.

Notes

- *time* and *theta0* are arrays of length npackets. radius and orbrate are arrays with length nobj (number of objects in the system which are included in the model run).
- The outputs are arrays of size (npackets x nobj) which give the location (in model coordinates) of each satellite as seen by each packet.
- $z \equiv 0 \rightarrow$ I have assumed satellites orbit in the planetary equatorial planes.
- Equations used:

$$\theta = -tR + \theta_0 \tag{48}$$
$$x = -r \sin\theta \tag{49}$$
$$y = r \cos\theta \tag{50}$$

where θ is *ang*, t is *time*, R is *orbrate* θ_0 is *theta0*, and r is *radius*.

This Page Last Updated

9 December 2011

62 ✓model_common_blocks_3.0.pro

Summary

Initializes the common blocks used in the model

Calling Procedure

`model_common_blocks`

Inputs

None.

Outputs

None.

Model Procedures Called

None.

Notes

Three common blocks are defined:

1 constants
- SystemConsts: See Table 4
- DipoleConsts: See Table 5
- stuff: See modeldriver

2 ratecoefs
- kappa: See modeldriver

3 plasma
- plasma: See modeldriver
- plasmahot: See modeldriver

This Page Last Updated

9 December 2011

63 model_findpackets_4.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

64 modeldriver_3.9.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

65 modelstreamlinesB_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

66 modelstreamlines_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

67 modstreamA_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

68 modstreamB_3.0.pro

Summary

Calling Procedure

Inputs

Outputs

Model Procedures Called

Notes

This Page Last Updated

69 ✓planet_dist_2.0.pro

Summary

Calculates the distance and radial velocity of a planet relative to the sun as a function of true anomaly angle.

Calling Procedure

`planet_dist, taa, SystemConsts, distance=distance, velocity=velocity`

Inputs

1 taa: array containing true anomaly angles in radians
2 SystemConsts: a SystemConstants structure.

Outputs

1 distance: distance of the planet from the sun. Units: AU
2 velocity: radial velocity (dr/dt) of the planet from the sun. Units: km s^{-1}.

Model Procedures Called

None.

Notes

1 The radial velocity is only computed for Mercury and Pluto. Otherwise, it is assumed to be 0. This is to avoid having to model a specific true anomaly for Jupiter and Saturn where dr/dt is small.
2 Equations:

$$r = \frac{a(1+\epsilon)}{1+\epsilon\cos\phi} \tag{51}$$

$$v_r = \frac{dr}{dt} \tag{52}$$

where a is the semi-major axis, ϵ is the orbital eccentricity, and ϕ is the true anomaly angle. While computing r is straight forward, computing v_r is not as that requires knowing how ϕ varies with time.
3 Procedure used (from wikipedia):
3.1 The *mean anomaly* is given by:

$$M = \frac{2\pi t}{P} \tag{53}$$

where P is the orbital period and t is time (t/P is the orbital phase)
3.2 The *eccentric anomaly* is the angle from perihelion to the center of the ellipse to the planet:

$$M = E - \epsilon \sin E \tag{54}$$

where E, the eccentric anomaly, is determined numerically.
3.3 The true anomaly angle is given by:

$$\tan(\phi/2) = \sqrt{\frac{1+\epsilon}{1-\epsilon}}\tan(E/2) \tag{55}$$

which can easily be inverted to compute ϕ.

3.4 To get $\phi(t)$, I compute ϕ at 1000 points discrete points along the orbit and numerically compute the derivative dr/dt. This gives ϕ, r, and dr/dt as functions of time (converting from orbital phase to time units using Kepler's Third Law, $P^2 = a^3$, to get the orbital period). I compute r and $drdt$ at the requested true anomalies by linear interpolation.

This Page Last Updated

12 December 2011

References

Burger et al. 2010.

Killen et al. 2009.

Field	Data Type	Unit	Description
`x0`	pointer to dblarr	R_{obj}	initial x position
`y0`	pointer to dblarr	R_{obj}	initial y position
`z0`	pointer to dblarr	R_{obj}	initial z position
`f0`	pointer to dblarr	–	initial fractional content
`vx0`	pointer to dblarr	$R_{plan}\ s^{-1}$	initial v_x
`vy0`	pointer to dblarr	$R_{plan}\ s^{-1}$	initial v_y
`vz0`	pointer to dblarr	$R_{plan}\ s^{-1}$	initial v_z
`phi0`	pointer to dblarr	radians	initial orbital phase (local time)
`totalsource`	double	R_{obj}	total of `f0`
`time`	pointer to dblarr	sec	integration time for each packet
`x`	pointer to dblarr	R_{plan}	final x position
`y`	pointer to dblarr	R_{plan}	final y position
`z`	pointer to dblarr	R_{plan}	final z position
`frac`	pointer to dblarr	–	final fractional content
`vx`	pointer to dblarr	$R_{plan}\ s^{-1}$	final v_x
`vy`	pointer to dblarr	$R_{plan}\ s^{-1}$	final v_y
`vz`	pointer to dblarr	$R_{plan}\ s^{-1}$	final v_z
`lossfrac`	pointer to dblarr	–	fraction lost to ionization
`hitfrac`	pointer to dblarr	–	fraction hitting objects
`ringfrac`	pointer to dblarr	–	fraction hitting Saturn's rings
`leftfrac`	pointer to dblarr	–	fraction escaping beyond *input.options.outeredge*
`deposition`	structure	–	map of surface deposition (Table **??**)
`loss_info`	structure	–	ionization and dissociation reactions used (Table **??**)
`sourcefile`	pointer to strarr	–	origin of this output file

Table 3: `output` fields.

Field	Data Type	Unit	Description
`Planet`	string	–	Planet name
`rPlan`	double	km	Planetary radius
`aPlan`	double	AU	Planetary semi-major axis
`epsPlan`	double	–	Planetary eccentricity
`Objects`	pointer to strarr	–	Object (planet and satellite) names (1)
`GM`	pointer to dblarr	${R_{plan}}^3$ s^{-2}	Gravitational constant × Planetary mass (1,2)
`radius`	pointer to dblarr	R_{plan}	Radius of each object relative to planetary radius (1)
`a`	pointer to dblarr	R_{plan}	semi-major axis of each object relative to plantary radius (1)
`eps`	pointer to dblarr	–	eccentricity of each object (1)
`orbvel`	pointer to dblarr	km s^{-1}	orbital velocity of satellites (2)
`period`	pointer to dblarr	s	orbital period of satellites (1)
`orbrate`	pointer to dblarr	s^{-1}	orbital frequency of satellites (= 2π/period) (1)

Table 4: `SystemConsts` fields. Notes: (1) The length of these arrays is (number of satellites) + 1. The first element refers to the planet and is 0, except for radius (where it is 1). (2) G = 6.67428×10^{-8} cm^{-3} s^{-2} g^{-1}

Field	Data Type	Unit	Description
`strength`	double	G R_{plan}^3	Dipole field strength
`tilt`	double	radians	Dipole tilt
`lam3`	double	radians	Longitude of tilt
`offset`	double	R_{plan}	Dipole offset
`offlong`	double	radians	Longitude of offset
`offlat`	double	radians	Latitude of offset
`period`	double	hours	Dipole period
`magrat`	double	hours^{-1}	Dipole rotation frequency (2π/period)

Table 5: `DipoleConsts` fields.

Object	**orbits**	**radius**	**mass**	**a**	ϵ	**rot period**	**orb period**
Sun	—	6.955×10^5	1.9891×10^{30}	0	0	601.2	0.
Mercury	Sun	2439.7	0.330104×10^{24}	0.387098	0.205630	1407.51	87.97
Venus	Sun	6051.8	4.86732×10^{24}	0.723	0.0068	5832.45	244.70
Earth	Sun	6378.14	5.97219×10^{24}	1.0	0.0167	23.93	365.25
Mars	Sun	3396.2	0.641693×10^{24}	1.524	0.0934	24.62	687.02
Jupiter	Sun	71492.	1898.13×10^{24}	5.203	0.0483	9.9250	4333.
Saturn	Sun	60268.	568.319×10^{24}	9.529	0.0560	10.7625	10743.
Uranus	Sun	25559.	86.8103×10^{24}	19.19	0.0461	17.24	30700.
Neptune	Sun	24764.	102.410×10^{24}	30.09	0.0097	16.11	60280.
Pluto	Sun	1151.	0.01309×10^{24}	39.24	0.2482	153.29	90130.
Moon	Earth	1737.10	7.3477e22	384400	0.554	655.728	27.322
Io	Jupiter	1821.6	8.9319e22	421800.	0.0041	42.456	1.769
Europa	Jupiter	1560.8	4.8e22	671100.	0.0094	85.224	3.551
Ganymede	Jupiter	2163.2	1.4819e23	1070400.	0.0013	171.72	7.155
Callisto	Jupiter	2410.3	1.0759e23	1882700.	0.0074	400.56	16.69
Mimas	Saturn	198.2	0.4e20	185539.	0.0196	22.608	0.942
Enceladus	Saturn	252.1	1.1e20	238037.	0.0047	31.344	1.370
Tethys	Saturn	533.0	6.2e20	294672.	0.0001	45.312	1.888
Dione	Saturn	561.7	11e20	377415.	0.0022	65.688	2.737
Rhea	Saturn	764.3	23e20	527068.	0.0010	108.432	4.518
Titan	Saturn	2575.5	1350e20	1221865.	0.0288	478.800	15.95
Hyperion	Saturn	135.0	5.58e18	1500934.	0.0232	510.720	21.28
Iapetus	Saturn	735.6	18e20	3560851.	0.0293	1903.92	79.33
Phoebe	Saturn	106.6	8.292e18	12947913.	0.1634	13207.2	550.30
Charon	Pluto	593.	1.62e21	19600.	0.	153.294	6.38725
Nix	Pluto	68.	2e18	48708.	0.	596.544	24.856
Hydra	Pluto	84.	2e18	64749.	0.	916.944	38.206

Table 6: Solar System Parameters