Fast transit time computation

Eric Agol (UW/CfA), Kat Deck (MIT), Matt Holman (CfA), David Nesvorny (SWRI)

1 Introduction

The routine `ttv_fast.for` is a code written for computing transit times as fast as possible. Since it is used for integrating short timescale dynamics (over several years, or typically tens to hundreds of dynamical times), for observed planetary systems (which, since we observe these systems, presumably are dynamically stable), it is not concerned with stability of the system or concerned with close approaches. The code is kept as short and simple as possible to make it easy to read and interpret, and it avoids trying to compute the times of transit at high accuracy (which isn't necessary given the typical transit timing error bars from Kepler, yet can end up costing a lot in CPU time). A more extensive description of the code can be found in Deck, Agol, Holman & Nesvorny (2014, arXiv:1234.5678). Also, this Fortran version was written separately, and hence has some minor differences in how it carries out the computation, and one major difference: it uses democratic heliocentric coordinates rather than Jacobi coordinates for the integration.

2 Symplectic integrator specifics and speed-up

The code uses a first-order symplectic integration scheme with canonical heliocentric coordinates as described in Wisdom (2006, AJ, 131, 2294; based on many other papers). Canonical heliocentric coordinates, which use heliocentric positions and barycentric velocities, were chosen since most multi-planet systems with TTVs orbit a single star, and the planet masses are much less than the stellar mass. Canonical heliocentric coordinates have a minimal number of coordinate transformations necessary to compute times of transit since the coordinates of each planet are referred to that of the star; only the velocity components need to be transformed from barycentric to heliocentric when carrying out the interpolation to find a time of transit. There are symplectic correctors written in canonical heliocentric coordinates (also summarized in Wisdom 2006), so this can give additional accuracy if required (in addition to taking a smaller time step), but at the cost of much more CPU time. The integrator takes a fixed time step, Δt, and we use an operator splitting of drift(Δt/2) kick(Δt) drift(Δt/2), where the kick includes the barycentric correction. To save on computation time, we combine successive Δt/2 drifts into a single Δt drift; then, when a transit has been identified, we drift back -Δt/2 to the end of a time step for the three time steps near the time of transit.

We find that it is not necessary to apply the inverse corrector at each transit time; this saves significantly on CPU time. This results in an error in transit

time that is much smaller than the typical Kepler error bars, and is also smaller than errors caused by our Keplerian interpolation scheme for finding the times of transit. However, we do find that applying a single 3rd-order corrector (Wisdom 2006) at the start of the integration leads to improved accuracy of the derivation of the times of transit. This is due to a correction of the phase of the orbit which manifests itself over long timescales; the inverse corrector is not necessary since the phase error between symplectic time steps are small.

We find that taking symplectic integrator time steps of 1/20 of the shortest orbital period is sufficient to reach timing errors of high enough accuracy for the Kepler data. Longer time steps cause a significant growth of inaccuracy over the timescale of the simulation, while shorter time steps cause longer simulation times.

An additional feature that saves on computing time is that we store the change in eccentricity anomaly, ΔE, from prior time steps. We use the three prior symplectic time steps to do a quadratic extrapolation to the current time step; this gives a very accurate starting estimate for ΔE during a Δt drift. When applying the corrector at the beginning, or when taking a -Δt/2 drift when a transit has occurred, we use the standard starting estimate of ΔE from Danby.

3 Transit time finding

The times of transit are found approximately by interpolating the three nearest steps once a transit has been found, which is flagged when the sign of the sky velocity (speed of the planet with respect to the star, projected onto the sky) changes from negative to positive, indicating that the planet and star have gone from approaching one another to separating. The interpolation proceeds as follows: 1) Once a transit time of a particular planet is flagged as having occurred between two time steps, an initial guess at the time is computed from linear interpolation of the sky velocity of the planet between the bracketing time steps; 2) The orbits of the three nearest time steps are extrapolated by a Keplerian drift of the heliocentric coordinates of the planet to the middle time step (so perturbations due to the other planets are ignored). These three sets of coordinates at the mid time are then fit with a quadratic function in time to interpolate to the transit time (when v_{sky}=0). 3) The root-finding approach of Fabrycky (2010) is used in finding the time of transit, using these interpolated coordinates with a Keplerian drift to transform to the mid-point time. This procedure proves to be fast and accurate enough for the problems we have considered so far, giving timing errors of a few seconds. Using the quadratic interpolation of the three nearest symplectic time steps is similar to the linear interpolation scheme of Nesvorny et al. (2013, arXiv:1034.4283) for KOI 142.01.

Once a time of transit is found, the projected sky separation of the planet and star (impact parameter), as well as the projected sky velocity (which determines

the duration of transit) is computed at the midpoint of transit.

4 Code technicalities

The subroutine `ttv_fast` uses physical units with GM in AU^3/day^2, length scales in AU, time in days, and velocity in AU/day. The input required for the code is a vector of parameters that contain the masses of the bodies, as well as the cartesian coordinates for the positions and velocities of all bodies in barycentric coordinates; the code then converts these to canonical heliocentric coordinates. The cartesian coordinates are set up so that transits occur in the x-y plane (the observer views the system along the z-plane). So, typically, the y-values are nearly zero for an edge-on transiting planet system. The choice of cartesian coordinates was used to minimize the number of transformations that occur within the code, to avoid any ambiguity that occurs in trying to interpret orbital elements (for example, does the longitude of periastron refer to that of the planet or of the star? are the orbital elements Jacobi, heliocentric, or barycentric? etc...). This coordinate conversion can occur in wrapper routines that call `ttv_fast`. The vector is arranged such that the first seven elements are the mass and barycentric position and velocity of the star (in code units), the second seven are the same for the innermost planet, and so on. The number of planets is specified in an include file, `'nbody_include.for'`, which contains a parameter `nbody` which is the total number of bodies to be integrated (and in this case equals the number of planets plus one for the star). This include file must be edited before compilation of the code in order to specify the correct number of bodies in each subroutine.

The radial velocity (in AU/day) is computed at the times specified in the file `rvdata.txt`. The first line of the file gives the number of RV times; the subsequent lines give the times at which the RV is to be computed (which we compute via Keplerian extrapolation of all of the planets' motions from the time step just after the RV time is passed).

If a transit occurs within two time steps of the initial time of the integration, the code can miss computing the transit due to the way transits are flagged (this should be a rare occurrance).

5 Speed and accuracy test

We tested this integrator on the outer five planets of the KOI 351 system, and find that it produces transit times accurate to about <1 seconds over 1500 days; more than sufficient for the typical 1-10 minute precision of timing measurements from Kepler for this system. Comparison with Bulirsch-Stoer integrators on the same processor indicate that it is about an order of magnitude faster.