A Practical Method for Solving the Kepler Equation

Marc A. Murison
U.S. Naval Observatory, Washington, DC
murison@usno.navy.mil

6 November, 2006

Abstract

We summarize and show a practical yet fast method, optimized with respect to cpu time, that numerically solves the Kepler equation.

Subject headings: celestial mechanics—two-body problem—Kepler equation

The XML version of this document is available on the web at
http://www.alpheratz.net/murison/dynamics/twobody/KeplerIterations_summary.xml
The PDF version of this document is available on the web at
http://www.alpheratz.net/murison/dynamics/twobody/KeplerIterations_summary.pdf

1 Introduction

The Kepler equation relates the linearly advancing time to the nonlinear relative angular position between two bodies m_1 and m_2 in Keplerian motion about their center of mass. It is

$$E\left(t\right) - e\sin E\left(t\right) = M\left(t\right) \tag{1}$$

where E is the eccentric anomaly, e is the orbital eccentricity, and $M\left(t\right) = n\left(t - \tau\right)$ is the mean anomaly, where $n = \sqrt{G(m_1 + m_2)/a^3}$ is the two-body mean motion with τ being the time of pericenter passage and a the semimajor axis of the orbital ellipse. See Figure 1.

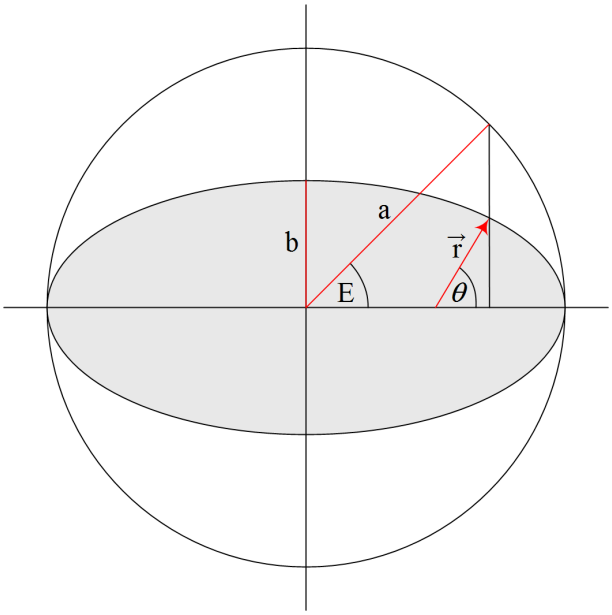

Figure 1: The geometric relationship between the eccentric anomaly E and the true anomaly θ.

We will outline here the development of a numerical method for solving the Kepler equation. Ideally, a method should be practical in the sense that it should simultaneously try to (1) minimize the cpu time spent solving for E and (2) minimize the programming complexity of the procedure. These two requirements tend to oppose each other, so we shall arrive at a suitable compromise. Fortunately, the compromise shown here is in fact ideal. A more detailed and comlete development will appear in another paper. The emphasis here is on immediate practicality.

The Kepler equation is transcendental, which means solutions must be found iteratively. Thus, any numerical procedure will have two tasks. First is the iterative loop wherein some refining algorithm is repeated until a satisfactory convergence is achieved. Generally, the higher the order of the algorithm, the fewer iterative passes needed. However, higher order brings with it a mushrooming expression complexity, which costs cpu time. Thus, for whatever algorithm one chooses, a certain (usually low) order will result in minimum cpu expenditure. The second task is to choose a starting value for the iterative loop. The better the initial approximation, the faster the loop will converge. The starting value method need not be the same as, or even remotely similar to, the iteration method. Similar to the iteration algorithm, there will be an ideal order for a given starting value approximation method that minimizes the cpu cost.

In what follows, we present a particularly simple starting value method, followed by what turns out to be a fast iteration method. In Section 5 we show results from work elsewhere that indicates the choice of order for each method summarized here is ideal, followed by a fast and fool-proof procedure that uses these methods. Fortunately, it is surprisingly simple and easy to adapt to any numerical computing language.

2 A Starting Value Method

Since we must solve the Kepler equation iteratively, it seems reasonable that the more accurate the starting value we feed to the iterative loop the better, at least until expression complexity becomes objectionable. Write the Kepler equation as

$$E = M + e\sin E \tag{2}$$

In the limit of zero eccentricity, we just have $E = M$. Thus, (2) suggests a simple iterative scheme for improving the starting approximation $E = M$. Write

$$E_k = M + e\sin E_{k-1} \tag{3}$$

where $E_0 = M$. We can then iterate the recursive expression (3) to as many orders in e as desired. For example, the third-order approximation is

$$E = M + e\sin M + e^2\sin M\cos M + \frac{1}{2}e^3\sin M\left(3\cos^2 M - 1\right) \tag{4}$$

An third-order pseudocode procedure for (4), optimized with respect to calculation time, is

```
KeplerStart3 := proc(e,M)
    local t33, t35, t34;
    t34 := e^2;
    t35 := e*t34;
    t33 := cos(M);
    return M+(-1/2*t35+e+(t34+3/2*t33*t35)*t33)*sin(M);
end proc;
```

3 An Iteration Method

Since (1) is a transcendental equation, it must be solved iteratively, whether numerically or analytically. So we must search for an expression that, when given some E that is in error, returns an approximation that reduces the error. It must also converge. To that end, write eq. (1) in the form

$$f(x) = x - e\sin x - M \tag{5}$$

where the solution of $f(x) = 0$ is $x = E$. Let $\varepsilon \equiv x - E$ be the error in the approximation of E given by x. Then a Taylor expansion about $x = E$ yields

$$f(E) = f(x - \varepsilon) = x - e\sin x - M - (1 - e\cos x)\,\varepsilon + \frac{1}{2}\varepsilon^2 e\sin x - \frac{1}{6}\varepsilon^3 e\cos x + \ldots \tag{6}$$

assuming ε is small.

We may solve the first-order term of (6) for ε to get

$$\varepsilon = \frac{x - e\sin x - M}{1 - e\cos x} \tag{7}$$

We can use this as the kernel of a first-order iterative scheme. Suppose we start with an initial guess, $x = x_0$. Then $x_1 = x_0 + \varepsilon$ should be a better approximation for E than x_0, and so on. Thus, we posit the first-order iterative procedure

$$\varepsilon_{n+1} = \frac{x_n - e\sin x_n - M}{1 - e\cos x_n} \tag{8}$$

where the choice of starting value for x_0 will be discussed below. We have in (8) a single-step first-order iterative method for estimating $E_{n+1} = E_n - \varepsilon_n$. A pseudocode procedure for (8) is

```
eps1 := proc(e,M,x)
    return (x-e*sin(x)-M)/(1-e*cos(x));
end proc;
```

At second order, (6) in Horner form is

$$f(x - \varepsilon) = x - e\sin x - M - \left(1 - e\cos x - \frac{1}{2}e\sin x \cdot \varepsilon\right)\varepsilon \tag{9}$$

We may rearrange $f(x-\varepsilon)=0$ given by (9) into the suggestive form

$$\varepsilon = \frac{x - e\sin x - M}{1 - e\cos x - \frac{1}{2}\varepsilon e\sin x} \tag{10}$$

Hence, let us create a second-order iterative scheme by writing, in analogy to (8),

$$\varepsilon_{n+1} = \frac{x_n - e\sin x_n - M}{1 - e\cos x_n - \frac{1}{2}\varepsilon_n e\sin x_n} \tag{11}$$

We can create a two-step iterative procedure out of this by first calculating ε_n given by (8) and then ε_{n+1} given by (11). We can also just skip the intermediate step by substituting (7) directly for ε_n in (11). Then the single-step iteration is

$$\varepsilon_{n+1} = \frac{x_n - e\sin x_n - M}{1 - e\cos x_n - \frac{1}{2}e\sin x_n \frac{x_n - e\sin x_n - M}{1-e\cos x_n}} \tag{12}$$

An optimized pseudocode procedure for (12) is

```
eps2 := proc(e,M,x)
        local t1, t2, t3;
        t1 := -1+e*cos(x);
        t2 := e*sin(x);
        t3 := -x+t2+M;
        return t3/(1/2*t3*t2/t1+t1);
end proc;
```

The third-order approximation of $f(E)=f(x-\varepsilon)$ in Horner form is

$$f(x-\varepsilon) = x - e\sin x - M - \left(1 - e\cos x - \left(\frac{1}{2}e\sin x - \frac{1}{6}e\cos x \cdot \varepsilon\right)\varepsilon\right)\varepsilon \tag{13}$$

Setting this to zero and solving for the "top-level" ε on the right-hand side, we have

$$\varepsilon_{n+1} = \frac{x_n - e\sin x_n - M}{1 - e\cos x_n - \frac{1}{2}\left(e\sin x_n - \frac{1}{3}e\cos x_n \cdot \varepsilon_n\right)\varepsilon_n} \tag{14}$$

We may use (12) for ε_n in (14) for a two-step method, or (8) for ε_n in (11) and then that result for ε_n in (14) for a three-step method. Alternatively, we can just substitute (12) for ε_n directly in into (14) for a one-step third-order method. An optimized pseudocode representation for the latter is

```
eps3 := proc(e,M,x)
        local t1, t2, t3, t4, t5, t6;
        t1 := cos(x);
        t2 := -1+e*t1;
        t3 := sin(x);
        t4 := e*t3;
        t5 := -x+t4+M;
```

```
    t6 := t5/(1/2*t5*t4/t2+t2);
    return t5/((1/2*t3 - 1/6*t1*t6)*e*t6+t2);
end proc;
```

One may continue in this fashion to higher orders.

4 Converting between True and Eccentric Anomaly

If one needs to work with true anomaly θ rather than eccentric anomaly, the conversion can be derived from inspection of Figure 1. From the figure, the magnitude of the position vector can be written

$$r = a(1 - e\cos E) \tag{15}$$

Again from inspection of Figure 1, we may write

$$a\cos E = r\cos\theta + ae \tag{16}$$

From (16) one can then easily derive

$$\cos\theta = \frac{\cos E - e}{1 - e\cos E} \quad \text{and} \quad \sin\theta = \frac{\sqrt{1 - e^2}\sin E}{1 - e\cos E} \tag{17}$$

Inverting eqs. (17), we can go the other direction with

$$\cos E = \frac{e + \cos\theta}{1 + e\cos\theta} \quad \text{and} \quad \sin E = \frac{\sqrt{1 - e^2}\sin\theta}{1 + e\cos\theta} \tag{18}$$

Hence, we can, if necessary, use our iterative procedures to solve the Kepler equation for E, then use (17) to convert to θ.

5 Summary: A Useful Numerical Method

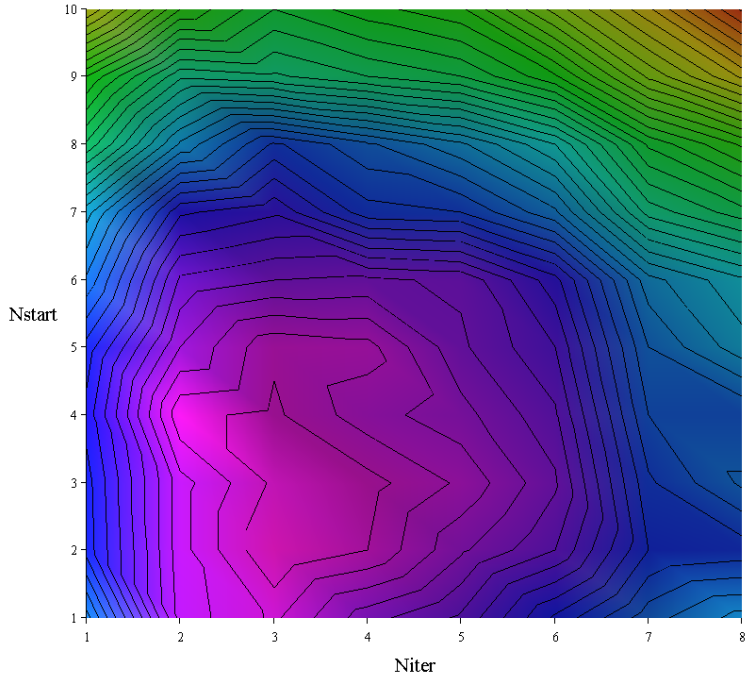

Figure 2: Contours, as a function of starting value order (Nstart) and iteration order (Niter), of the amount of cpu time expended to perform, at each $(N_{\text{iter}}, N_{\text{start}})$ point, 160,000 solutions of the Kepler equation on an evenly-spaced 400×400 grid over the domain $\{\mathbb{R} \times \mathbb{R} : e \in [0, 1)\,, M \in [0, \pi]\}$. It appears that third order for each method is near-optimal.

Extensive numerical tests (see Figure 2) indicate that third order for both starting value and iteration, using the methods shown previously, is the most optimal with respect to time spent calculating.

Here then is a procedure to numerically solve the Kepler equation that makes use of the optimized third-order iteration and starting value methods:

```
KeplerSolve := proc( e, M, tol:=1.0e-14 )
        local dE, E, E0, Mnorm, count;
        global Estart3, eps3;
        Mnorm := fmod(M,2*Pi);
        E0 := KeplerStart3(e,Mnorm);
        dE := tol + 1;
        count := 0;
        while dE > tol do
                E := E0 - eps3(e,Mnorm,E0);
                dE := abs(E-E0);
                E0 := E;
                count := count + 1;
                if count=100 then
                        print “Astounding! KeplerSolve failed to converge!”;
                        break;
                end if;
        end do;
        return E;
end proc;
```