# The Non-forced Spherical Pendulum: Semi-numerical Solutions

Richard J. Mathar*
*Max-Planck Institute of Astronomy, Königstuhl 17, 69117 Heidelberg, Germany*
(Dated: June 16, 2022)

This manuscript deals with semi-numerical solutions of the equations of motion of the spherical pendulum. This pendulum has some azimuthal velocity and non-vanishing angular momentum. The cord restricts the motion to the surface of a sphere. We derive the system of coupled differential equations for the radial and azimuthal angles from the Lagrangian, solve them in terms of Elliptic Integrals, and provide a C++ program to print trajectories as a function of time utilizing the GNU scientific library. [vixra:1909.0201]

PACS numbers: 45.20.Jj

## I. COORDINATE SYSTEM

### A. Cartesian Coordinates and Angles

An ideal point mass is suspended from a pendulum of cord length $l$. The mass has Cartesian coordinates $x, y, z$, $z$ pointing upwards, and we define the origin of these coordinates, the point $x = y = z = 0$, to be the point of suspension. We assume the pendulum restricts the mass position to the sphere at distance $l$ from the suspension:

$$x^2 + y^2 + z^2 = l^2. \tag{1}$$

The constraint introduces two spherical angles (measured in radians): The first, $\varphi$, is the angle between the cord and the vertical of the pole. The second is an azimuth angle $\lambda$,

$$\tan\lambda = \frac{y}{x}. \tag{2}$$

$\lambda$ defines the instantaneous plane of the motion. The standard set of coordinate transformations reads:

$$\sin\varphi = \frac{\sqrt{x^2+y^2}}{l}. \tag{3}$$

$$\cos\varphi = -z/l; \quad z = -l\cos\varphi. \tag{4}$$

$$\sin\lambda = \frac{y}{\sqrt{x^2+y^2}}; \cos\lambda = \frac{x}{\sqrt{x^2+y^2}}; \tag{5}$$

$$x = l\sin\varphi\cos\lambda; \tag{6}$$

$$y = l\sin\varphi\sin\lambda. \tag{7}$$

### B. Time Derivatives

The derivatives of the Cartesian coordinates with respect to time — indicated by an overdot — are

$$\dot{x} = l\dot{\varphi}\cos\lambda\cos\varphi - l\dot{\lambda}\sin\varphi\sin\lambda. \tag{8}$$

$$\dot{y} = l\dot{\varphi}\sin\lambda\cos\varphi + l\dot{\lambda}\sin\varphi\cos\lambda. \tag{9}$$

$$\dot{z} = l\dot{\varphi}\sin\varphi. \tag{10}$$

The squared velocity is

$$\dot{x}^2 + \dot{y}^2 + \dot{z}^2 = l^2(\dot{\lambda}^2\sin^2\varphi + \dot{\varphi}^2). \tag{11}$$

* https://www.mpia-hd.mpg.de/~mathar

## II. LAGRANGIAN

### A. Energies

We set the zero of the potential energy $V$ at the origin of the Cartesian Coordinates and assume that a homogeneous gravitational potential:

$$V = mgz. \tag{12}$$

$m$ is the mass at the end of the pendulum and $g$ the acceleration. The Kinetic Energy $K$ is proportional to the square of the velocity $v$:

$$K = \frac{1}{2}mv^2 = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2 + \dot{z}^2). \tag{13}$$

### B. Euler-Lagrange equations

The Lagrangian of the system the difference of $K$ and $V$ [1–4],

$$\mathcal{L} = K - V = \frac{1}{2}ml^2(\dot{\lambda}^2 \sin^2\varphi + \dot{\varphi}^2) + mgl\cos\varphi. \tag{14}$$

The Euler-Lagrange equations of the generalized coordinates $\varphi$ and $\lambda$ are

$$\frac{\partial\mathcal{L}}{\partial\varphi} = \frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\varphi}}; \tag{15}$$

$$\frac{\partial\mathcal{L}}{\partial\lambda} = \frac{d}{dt}\frac{\partial\mathcal{L}}{\partial\dot{\lambda}}, \tag{16}$$

explicitly with (14)

$$\frac{1}{2}ml^2\dot{\lambda}^2 2\sin\varphi\cos\varphi - mgl\sin\varphi = \frac{d}{dt}[\frac{1}{2}ml^2 2\dot{\varphi}]; \tag{17}$$

$$\therefore \dot{\lambda}^2\sin\varphi\cos\varphi - \frac{g}{l}\sin\varphi = \frac{d}{dt}\dot{\varphi}. \tag{18}$$

$$0 = \frac{d}{dt}[\frac{1}{2}ml^2\sin^2\varphi 2\dot{\lambda}]. \tag{19}$$

The only topic of this manuscript is to solve these two coupled differential equations for $\varphi(t)$ and $\lambda(t)$.

### C. Conical Pendulum

There is one particular solution where $\ddot{\lambda} = \dot{\varphi} = 0$ such that

$$\dot{\lambda}^2\cos\varphi = g/l. \tag{20}$$

This system is known as the conical pendulum because the mass swings on a circle of constant distance to the pole such that the forces exerted by the cord and by the gravitation are keeping $\varphi$ and $\dot{\lambda}$ constant in time.

## III. DIFFERENTIAL EQUATIONS OF MOTION

### A. Separation of canonical variables

A first integral of the second order differential equation for $\ddot{\lambda}$ from (19) is:

$$\sin^2\varphi\dot{\lambda} = L_0. \tag{21}$$

This represents the conservation of angular momentum [5]. ($l^2\dot{\lambda}^2\sin^2\varphi$ is the squared tangential velocity $\dot{x}^2+\dot{y}^2$, and $l^2\sin^2\varphi = x^2+y^2$ is the squared distance to the pole. So the product $m^2l^4\sin^4\varphi\dot{\lambda}^2$ is the squared angular momentum, and the equation above is up to constants the angular momentum.) It also implicitly says that the general solutions avoids $\varphi = 0$, the point of lowest potential, because that would force $L_0$ to fall to an abrupt and intermediate non-continuous zero at that point—unless $L_0 = 0$, the planar pendulum.

The second order differential equation (18) is

$$\ddot{\varphi} - \dot{\lambda}^2\sin\varphi\cos\varphi + \frac{g}{l}\sin\varphi = 0. \tag{22}$$

We follow the usual approach for differential equations that do not contain the first derivative [6, §9.1.2.2] [7, §19] [8] and multiply by $2\dot{\varphi}$ to obtain a first order differential equation:

$$2\dot{\varphi}\ddot{\varphi} - 2\dot{\lambda}^2\dot{\varphi}\sin\varphi\cos\varphi + \frac{2g}{l}\dot{\varphi}\sin\varphi = 0. \tag{23}$$

$$\frac{d}{dt}[\dot{\varphi}^2] - 2\dot{\lambda}^2\dot{\varphi}\sin\varphi\cos\varphi + \frac{2g}{l}\dot{\varphi}\sin\varphi = 0. \tag{24}$$

Elimination of $\dot{\lambda}^2$ via (21) decouples the two differential equations:

$$\frac{d}{dt}[\dot{\varphi}^2] - 2L_0^2\dot{\varphi}\frac{\cos\varphi}{\sin^3\varphi} + \frac{2g}{l}\dot{\varphi}\sin\varphi = 0. \tag{25}$$

$$\frac{d}{dt}[\dot{\varphi}^2] - 2L_0^2\frac{d}{dt}[-\frac{1}{2\sin^2\varphi}] - \frac{2g}{l}\frac{d}{dt}[\cos\varphi] = 0. \tag{26}$$

The format now resembles the radial equations in a gravitational potential where the squared angular momentum modifies the effective radial potential. The constant of integration is written as a function of $\varphi_0$ at some reference point in time:

$$\dot{\varphi}^2 + L_0^2\frac{1}{\sin^2\varphi} - \frac{2g}{l}\cos\varphi = \dot{\varphi}_0^2 + L_0^2\frac{1}{\sin^2\varphi_0} - \frac{2g}{l}\cos\varphi_0. \tag{27}$$

Starting at the upper angular limit, the angle drops, $\dot{\varphi} < 0$, and its cosine increases, $(d/dt)\cos\varphi > 0$. Starting from the lower limit, the signs are opposite, so both signs of that quadratic equation of $\dot{\varphi}$ are relevant,

$$\dot{\varphi} = \mp\sqrt{\dot{\varphi}_0^2 + L_0^2\frac{1}{\sin^2\varphi_0} - \frac{2g}{l}\cos\varphi_0 - L_0^2\frac{1}{\sin^2\varphi} + \frac{2g}{l}\cos\varphi}. \tag{28}$$

This differential equation is separable:

$$\mp\int_{\varphi_0}\frac{d\varphi}{\sqrt{\dot{\varphi}_0^2 + L_0^2\frac{1}{\sin^2\varphi_0} - \frac{2g}{l}\cos\varphi_0 - L_0^2\frac{1}{\sin^2\varphi} + \frac{2g}{l}\cos\varphi}} = \int_0 dt = t. \tag{29}$$

We solve the homogeneous equation, i.e., we shift the time variable such that $\dot{\varphi}_0 = 0$, measuring time from one of the extreme positions of the motions at maximum or minimum amplitude $\varphi$:

$$\mp\int_{\varphi_0}\frac{d\varphi}{\sqrt{\cos\varphi - \frac{lL_0^2}{2g}\frac{1}{\sin^2\varphi} - \cos\varphi_0 + \frac{lL_0^2}{2g}\frac{1}{\sin^2\varphi_0}}} = \sqrt{\frac{2g}{l}}t. \tag{30}$$

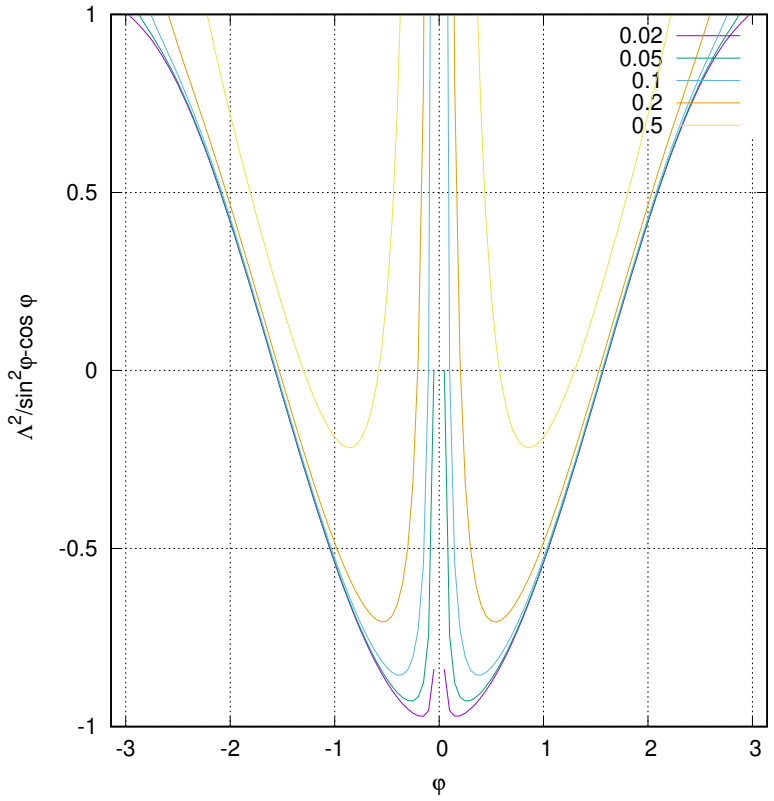


FIG. 1. The discriminant $D(\varphi) = D(2\pi - \varphi)$ as a function of $\varphi$ for five values of $\Lambda^2$.

Define a unitless time $T \equiv \sqrt{2g/l}t$ and the unitless angular momentum

$$\Lambda^2 \equiv lL_0^2/(2g) \tag{31}$$

— which means, define the time derivative in (21) also on the new scale — to reduce the radial integral to

$$\mp \int_{\varphi_0} \frac{d\varphi}{\sqrt{\cos\varphi - \Lambda^2 \frac{1}{\sin^2\varphi} - \cos\varphi_0 + \Lambda^2 \frac{1}{\sin^2\varphi_0}}} = T. \tag{32}$$

The discriminant

$$D(\varphi) \equiv \Lambda^2/\sin^2\varphi - \cos\varphi \tag{33}$$

is periodic with period $2\pi$, illustrated in Figure 1, with poles where the sine is zero, i.e., where $\varphi$ is a multiple of $\pi$. (The exception is the planar pendulum where $\Lambda = 0$.) The minimum is where $(d/d\varphi)D = 0$, the conical pendulum:

$$\sin^4\hat{\varphi} = 2\Lambda^2\cos\hat{\varphi}. \tag{34}$$

The is a quartic equation in $\cos\hat{\varphi}$ and could be solved directly. A suitable series expansion is

$$\sin^2\hat{\varphi} = (\sqrt{2}\Lambda) - \frac{1}{4}(\sqrt{2}\Lambda)^2 - \frac{1}{32}(\sqrt{2}\Lambda)^3 + \frac{7}{2048}(\sqrt{2}\Lambda)^5 + \frac{1}{512}(\sqrt{2}\Lambda)^6 + \frac{39}{65536}(\sqrt{2}\Lambda)^7 - \frac{1045}{8388608}(\sqrt{2}\Lambda)^9 - \frac{11}{131072}(\sqrt{2}\Lambda)^{10} - \frac{7735}{268435456}(\sqrt{2}\Lambda)^{11} + \frac{121923}{17179869184}(\sqrt{2}\Lambda)^{13} + \cdots; \tag{35}$$

$$\hat{\varphi} = 2^{1/4}\sqrt{\Lambda}[1 + \frac{\sqrt{2}}{24}\Lambda - \frac{7}{320}\Lambda^2 - \frac{65\sqrt{2}}{3584}\Lambda^3 - \frac{1045}{73728}\Lambda^4 - \frac{1785\sqrt{2}}{720896}\Lambda^5 + \frac{14973}{6815744}\Lambda^6 + \frac{153439\sqrt{2}}{62914560}\Lambda^7 + \cdots]; \tag{36}$$

$$\sin\hat{\varphi} = 2^{1/4}\sqrt{\Lambda}\left[1 - \frac{\sqrt{2}}{8}\Lambda - \frac{3}{64}\Lambda^2 - \frac{3\sqrt{2}}{512}\Lambda^3 + \frac{35}{8192}\Lambda^4 + \cdots\right]; \tag{37}$$

$$\cos\hat{\varphi} = 1 - \frac{\sqrt{2}}{2}\Lambda + \frac{\sqrt{2}}{32}\Lambda^3 + \frac{1}{32}\Lambda^4 + \frac{9\sqrt{2}}{1024}\Lambda^5 - \frac{55\sqrt{2}}{1634}\Lambda^7 + \cdots . \tag{38}$$

The value $D(\hat{\varphi})$ at that minimum is

$$D(\hat{\varphi}) = -1 + (\sqrt{2}\Lambda) + \frac{1}{8}(\sqrt{2}\Lambda)^2 + \frac{1}{32}(\sqrt{2}\Lambda)^3 + \frac{1}{128}(\sqrt{2}\Lambda)^4 + \frac{3}{2048}(\sqrt{2}\Lambda)^5 - \frac{11}{65536}(\sqrt{2}\Lambda)^7 - \frac{3}{32768}(\sqrt{2}\Lambda)^8 - \frac{221}{8388608}(\sqrt{2}\Lambda)^9 + \frac{1311}{268435456}(\sqrt{2}\Lambda)^{11} + \frac{13}{4194304}(\sqrt{2}\Lambda)^{12} - \cdots \tag{39}$$

$D(\hat{\varphi})$ is negative if $\Lambda$ is in the range

$$0 \le \Lambda^2 < \frac{2}{3^{3/2}} \approx 0.384900179. \tag{40}$$

### B. Equilibrium Points

If the value of $D(\varphi_0) - D(\varphi)$ becomes zero, the pendulum has reached a point of zero radial velocity, either a point at maximum amplitude or a point of closest swing by the pole. The initial conditions of the individual trajectory are set by shifting the plots of Figure 1 up or down until the ordinate value becomes zero at the value of $\varphi_0$ that is selected to be an equilibrium point. $D(\varphi) - D(\varphi_0)$ is essentially the energy $E$ at the equilibrium points. The trajectory is swinging forward and backward staying at one of the curves of Fig. 1.

Thus having fixed $\varphi_0$ and $D(\varphi_0) \equiv D_0$, an intermediate task is to find the other root(s) of the equation

$$\cos\varphi - \Lambda^2/\sin^2\varphi + D_0 = 0, \tag{41}$$

the second equilibrium point. Multiplied by $\sin^2\varphi$, this yields a cubic equation for the cosine,

$$\cos^3\varphi + \cos^2\varphi D_0 - \cos\varphi + \Lambda^2 - D_0 = 0. \tag{42}$$

Some comments of solving this without recourse to complex arithmetic are added in Appendix B. An overview of the solutions is given in Figure 3. If $D_0 > 0$, some parts of the trajectory may push the mass transiently through negative values of the cosine, i.e., above the horizon of the suspension.

The Cardano form of the requirement for three real solutions of a cubic polynomial reads in the case (42)

$$\Lambda^4 + 4D_0(D_0^2 - 9)\Lambda^2/27 - 4(1 - D_0^2)^2/27 < 0, \tag{43}$$

and is illustrated in Figure 2.

The substitution

$$\cos\varphi \equiv \theta \tag{44}$$

rewrites the integral (32) as an elliptic integral [9, 10][11, (17.4.67)][12, 3.131.5][13, (235.00)].

$$\mp \int_{\varphi_0} \frac{d\varphi}{\sqrt{\cos\varphi - \Lambda^2/\sin^2\varphi + D_0}} = T; \tag{45}$$

$$\pm \int_{\theta_0} \frac{d\theta}{\sqrt{-\theta^3 - D_0\theta^2 + \theta - \Lambda^2 + D_0}} = T. \tag{46}$$

The three roots $\cos\varphi_u = \theta_u$, $\cos\varphi_l = \theta_l$, and $\cos\varphi_s = \theta_s$ of the cubic equation of the cosine are fixed with the algebra indicated above and sorted numerically as $\theta_s < \theta_l \le \theta_u$. Vieta's laws state $\theta_s + \theta_l + \theta_u = -D_0$, $\theta_u\theta_l\theta_s = D_0 - \Lambda^2$ [11, 3.8.2]. Time $T$ is an Elliptic Integral of the First Kind [14, §10.1]:

$$\pm \int_{\theta_l} \frac{d\theta}{\sqrt{-(\theta - \theta_l)(\theta - \theta_u)(\theta - \theta_s)}} = T. \tag{47}$$

$$\pm F(\phi, k) = \frac{1}{2}\sqrt{\theta_u - \theta_s}\,T, \tag{48}$$

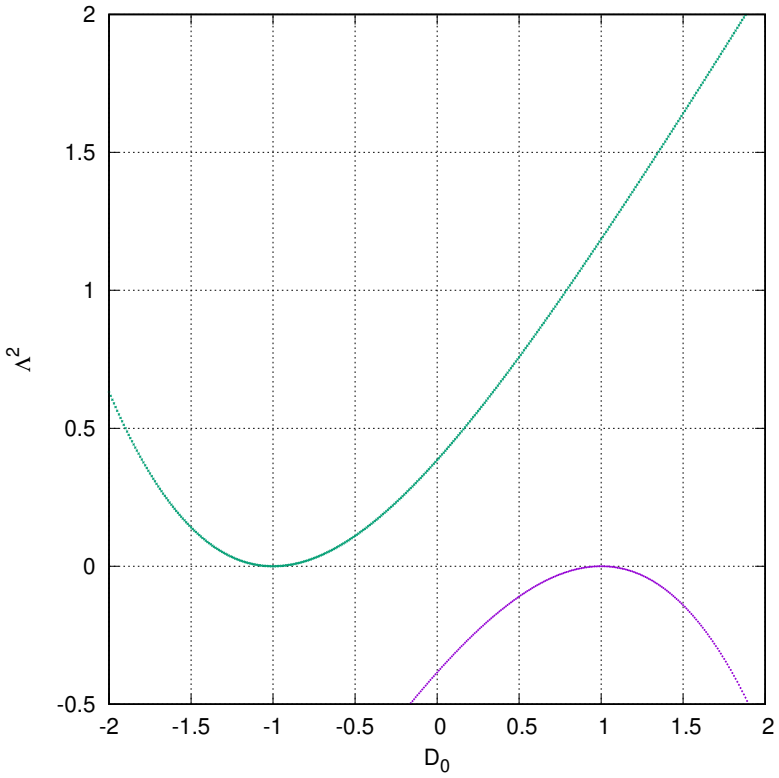


FIG. 2. The parameter region where the cubic equation has 3 real-valued roots is where $\Lambda^2 \geq 0$, and where $\Lambda^2$ is below the green and above he magenta line. The two lines are the two solutions setting the left hand side of the biquadratic equation (43) to zero.

for $\theta_l < \theta \leq \theta_u$, where

$$\sin^2 \phi = \frac{(\theta_u - \theta_s)(\theta - \theta_l)}{(\theta_u - \theta_l)(\theta - \theta_s)}, \tag{49}$$

$$k^2 = \frac{\theta_u - \theta_l}{\theta_u - \theta_s}, \tag{50}$$

are phase and modulus of the Elliptic Integral of the First Kind. (The phase $\phi$ is not to be confused with the distance $\varphi$ of the pendulum to the pole.) Integration between the two equilibrium points $\theta_l$ and $\theta_u$ yields the quarter period $P/4$ as a Complete Integral of the First Kind:

$$K(k) \equiv F(\pi/2, k) = \frac{1}{2}\sqrt{\theta_u - \theta_s} P/4. \tag{51}$$

(The nomenclature is here adapted to the plane pendulum, where the mass returns to the same farthest position after *two* passages through the lowest point.) Numerical evaluation of the Elliptic Integrals gives the curves of Figure 4.

Eq. (48) can be used to tabulate $T$ (time) as a function of angles $\varphi$ for all values in the ranges set by $\theta_l$ and $\theta_u$ by computing the Elliptic Integrals as a function of phases $\phi$. The Jacobian Elliptic Functions tabulate $\varphi$ as a function of time [13, §120.01] by computing

$$\sin \phi = \mathrm{sn}\left(\frac{1}{2}\sqrt{\theta_u - \theta_s} T, k\right), \tag{52}$$

then extracting $\theta$ by solving the linear equation (49)

$$\theta = \frac{\theta_l - \theta_s k^2 \sin^2 \phi}{1 - k^2 \sin^2 \phi} = \theta_l + (\theta_l - \theta_s)\frac{k^2 \sin^2 \phi}{1 - k^2 \sin^2 \phi} \tag{53}$$

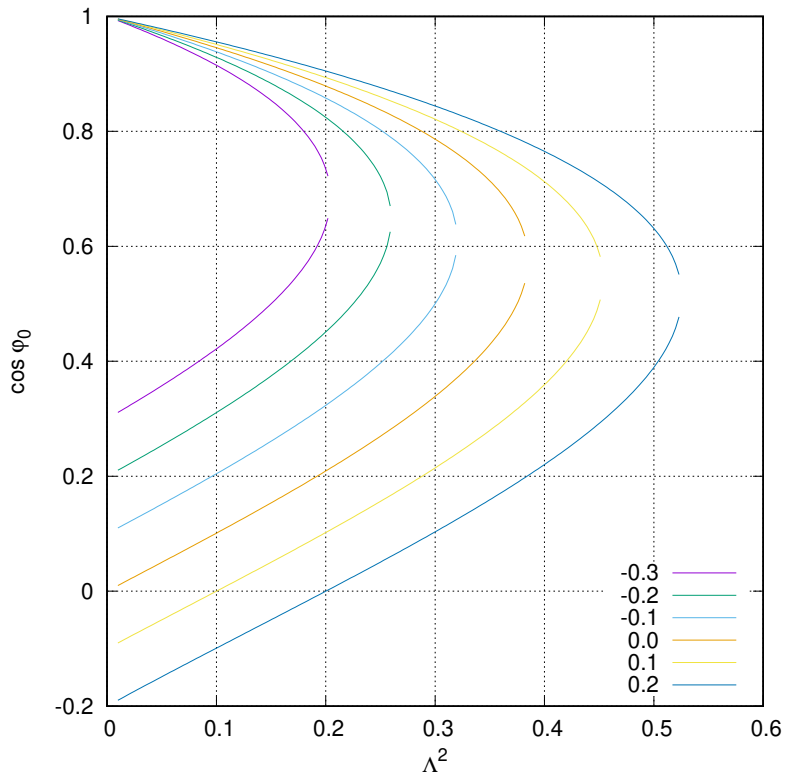


FIG. 3. The cosines of the two equilibrium positions, $\theta_u$ and $\theta_l$, as a function of $\Lambda^2$ for 6 different values of $D_0$ in the range of $-0.3$ to 0.2. The range of $D_0$ for which solutions exist depends on $\Lambda$, see Eq. (39).

and then $\varphi = \cos^{-1}\theta$. For small times power series are known [13, 907.01][11, 16.22.1][14, §10.3][15, A06028],[16, 17]:

$$\operatorname{sn}(z,k) = z - \left(1+k^2\right)\frac{z^3}{3!} + \left(1+14k^2+k^4\right)\frac{z^5}{5!} - \cdots \tag{54}$$

## IV. TRAJECTORY

### A. Of the Altitude

The differential equation of $\cos\varphi$ (basically proportional to the altitude $z$ or the gravitational energy $V$) allows to generate its power series as a function of $T$ recursively. Let the tick mark denote derivatives with respect to $T$.

$$\frac{d}{dT}\cos\varphi = -\sin\varphi\varphi'. \tag{55}$$

$$\frac{d^2}{dT^2}\cos\varphi = -\cos\varphi(\varphi')^2 - \sin\varphi\varphi'' \tag{56}$$

Then (28) means

$$\frac{d}{dT}\varphi = \mp\sqrt{\cos\varphi - \Lambda^2/\sin^2\varphi + D_0}, \tag{57}$$

and (22) means

$$\frac{d^2}{dT^2}\varphi = \Lambda^2\frac{\cos\varphi}{\sin^3\varphi} - \frac{1}{2}\sin\varphi. \tag{58}$$

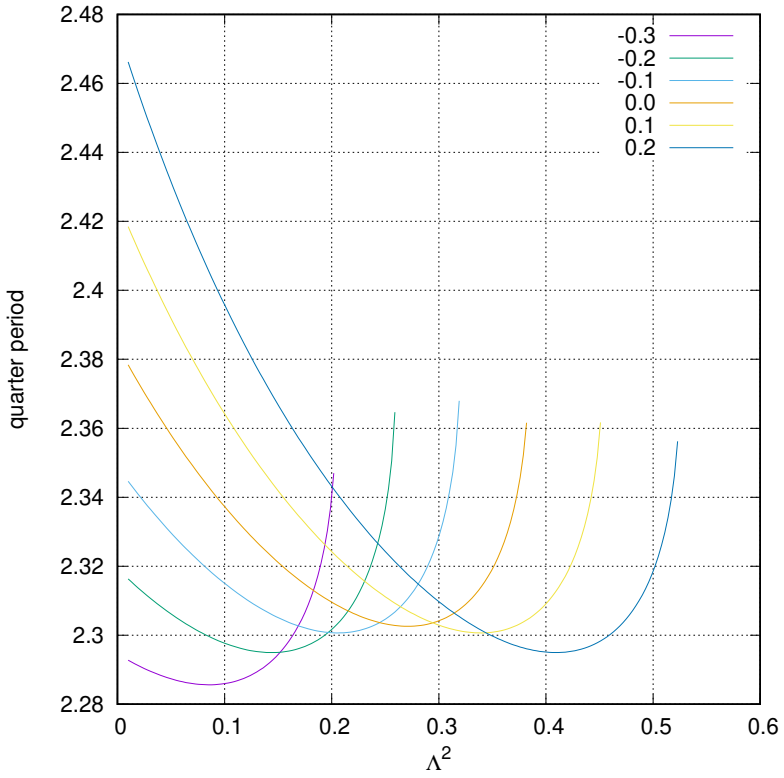


FIG. 4. The quarter period $P/4$ as a function of $\Lambda^2$ for the same 6 different signed values of $D_0$ as in Figure 3.

Insertion of these two equations eliminates $\Lambda$ and $\varphi$:

$$\frac{d^2}{dT^2}\cos\varphi = \frac{1}{2} - \frac{3}{2}\cos^2\varphi - D_0\cos\varphi. \tag{59}$$

We bootstrap the series expansion of $\cos\varphi$ as a function of $T$ by repeated integration of (59) from $\varphi_0 = \varphi_u$ or $\varphi_l$. The iteration starts with the observation that the linear term vanishes, $(d/dT)\cos\varphi_0 = -\sin\varphi_0 \cdot 0 = 0$. The quadratic order is copied from (59):

$$\cos\varphi = \cos\varphi_0 + \frac{1}{2}(\frac{1}{2} - \frac{3}{2}\cos^2\varphi_0 - D_0\cos\varphi_0)T^2 + O(T^3). \tag{60}$$

By defining power series coefficients $c_i$,

$$\cos\varphi = \cos\varphi_0 + \frac{1 - 3\cos^2\varphi_0 - 2D_0\cos\varphi_0}{4}T^2\sum_{i\geq 0} c_i(T/2)^{2i}, \tag{61}$$

the $c_i$ are recursively computable, rooted at $c_0 = 1$, as

$$c_i = -(3\cos\varphi_0 + D_0)\frac{c_{i-1}}{(i+1/2)(i+1)} - \frac{1 - 3\cos^2\varphi_0 - 2D_0\cos\varphi_0}{2}\frac{3}{(i+1/2)(i+1)}\sum_{j=0}^{i-2} c_j c_{i-2-j}. \tag{62}$$

[This is equivalent to plugging (54) into (53).]

Rephrasing the power series as a function of time measured in units of the quarter period — which is a matter of multiplying the coefficients with powers of $P/4$ — this can be written as a sum of Chebyshev Polynomials Of The First Kind of $4T/P$, and is equivalent to a Fourier time series of $\cos\varphi$ [18].

### B. Harmonic Analysis

In numerical practice Eq. (52) is not evaluated on a point-by-point basis but evaluated as a Fourier series of time [14, §10.3][11, 16.23.1][13, 908.01]:

$$\operatorname{sn}(2Kz,k)=\frac{2\pi}{Kk}\sum_{m=0}^{\infty}\frac{q^{m+1/2}}{1-q^{2m+1}}\sin[(2m+1)\pi z], \tag{63}$$

where $K$ is the Complete Elliptic Integral (51) playing the role of the period, $K'\equiv K/\sqrt{1-k^2}$ a complementary value, and $q\equiv\exp(-\pi K'/K)$ Jacobi's Nome [13, 1050].

### C. Evolution of the Azimuth

The first order differential equation (21) for the speed of the azimuth plane reads

$$\sin^2\varphi\lambda'=\Lambda \tag{64}$$

in the unitless time scale $T$. At the turning points the values of $\lambda'$ are known by isolating $\Lambda/\sin^2\varphi$ in (41)

$$\lambda'_0=\frac{1}{\Lambda}\frac{\Lambda^2}{\sin^2\varphi_0}=\frac{1}{\Lambda}(\cos\varphi_0+D_0)=\frac{1}{\Lambda}(\theta_0+D_0). \tag{65}$$

Assuming that the radial angle $\varphi$ or its cosine $\theta$ are known functions of time, (64) is a separable differential equation

$$\lambda=\Lambda\int\frac{1}{1-\theta^2(T)}dT. \tag{66}$$

The kernel is a geometric series for $\theta^2$ and via (53) a power series for $k^2\sin^2\phi$:

$$\begin{aligned}\frac{1}{1-\theta^2}&=\frac{(1-k^2\sin^2\phi)^2}{(1+\theta_l-k^2\sin^2\phi-\theta_sk^2\sin^2\phi)(1-\theta_l-k^2\sin^2\phi+\theta_sk^2\sin^2\phi)}\\
&=\frac{1}{1-\theta_l^2}\frac{(1-k^2\sin^2\phi)^2}{(1-\frac{1+\theta_s}{1+\theta_l}k^2\sin^2\phi)(1-\frac{1-\theta_s}{1-\theta_l}k^2\sin^2\phi)}\\
&=\frac{1}{1-\theta_l^2}(1-k^2\sin^2\phi)^2\left[\frac{1}{1-\frac{(1-\theta_s)(1+\theta_l)}{(1-\theta_l)(1+\theta_s)}}\frac{1}{1-\frac{1+\theta_s}{1+\theta_l}k^2\sin^2\phi}+\frac{1}{1-\frac{(1+\theta_s)(1-\theta_l)}{(1+\theta_l)(1-\theta_s)}}\frac{1}{1-\frac{1-\theta_s}{1-\theta_l}k^2\sin^2\phi}\right]\\
&=\frac{1}{1-\theta_l^2}\left[1+2\theta_l(\theta_l-\theta_s)\frac{k^2}{(1-\theta_l^2)}\sin^2\phi-(\theta_s+3\theta_s\theta_l^2-3\theta_l-\theta_l^3)(\theta_l-\theta_s)\frac{k^4}{(1-\theta_l^2)^2}\sin^4\phi\right.\\
&\qquad\left.+2(2\theta_l\theta_s-\theta_l^2-1)(\theta_s\theta_l^2+\theta_s-2\theta_l)(\theta_l-\theta_s)\frac{k^6}{(1-\theta_l^2)^3}\sin^6\phi+\cdots\right].\end{aligned} \tag{67}$$

The individual $\sin\phi$ are periodic in $T$ according to (63),

$$\sin\phi=\frac{2\pi}{Kk}\sum_{m=0}^{\infty}\frac{q^{m+1/2}}{1-q^{2m+1}}\sin\left[(2m+1)\pi\frac{\sqrt{\theta_u-\theta_s}}{4K}T\right], \tag{68}$$

so the powers $\sin^{2n}\phi$ are constant plus periodic terms $\propto\cos(m\pi\frac{\sqrt{\theta_u-\theta_s}}{4K}T)$ [12, 1.320.1]. After integrating (66), $\lambda(T)$ is a linear function of $T$ plus harmonic overtones of the same frequencies as $\cos\varphi(T)$.

Mixing $dT/d\theta=1/\sqrt{-(\theta-\theta_u)(\theta-\theta_l)(\theta-\theta_s)}$ into (66) yields [19]

$$\lambda=\frac{1}{2}\Lambda\int_{\theta_l}\left[\frac{1}{1+\theta}+\frac{1}{1-\theta}\right]d\theta\frac{1}{\sqrt{-(\theta-\theta_u)(\theta-\theta_l)(\theta-\theta_s)}}. \tag{69}$$

These are known Elliptic Integrals [13, 235.17,339.04,336]

$$\int_{\theta_l} d\theta \frac{1}{(\theta \mp 1)\sqrt{-(\theta-\theta_u)(\theta-\theta_l)(\theta-\theta_s)}} = \frac{2/\sqrt{\theta_u-\theta_s}}{\theta_l \mp 1}\frac{k^2}{\alpha^2}\left[F(\phi,k) + (\frac{\alpha^2}{k^2}-1)\Pi(\phi,\alpha^2,k)\right] \tag{70}$$

with

$$\alpha^2 \equiv \frac{(\theta_u-\theta_l)(\theta_s \mp 1)}{(\theta_u-\theta_s)(\theta_l \mp 1)}. \tag{71}$$

## V. SUMMARY

The system of differential equations (28) and (21) for the radial angle $\varphi$ and azimuth $\lambda$ of the Spherical Pendulum has been transformed into a system of equations for $\theta \equiv \cos\varphi$ and $\lambda$ and their derivatives for a unitless time $T$. Decoupling the equations gives the solution (52)–(53) for the $\theta(T)$ and (69) for the $\lambda(T)$ in terms of Elliptic Integrals of the first and third kind.

### Appendix A: Translating Sets of Initial Conditions

The initial conditions of the trajectory may be specified by initial position vector $(x, y, z)$ and initial velocity vector $(\dot{x}, \dot{y}, \dot{z})$. The transition to the dimensionless control parameters of our description may be installed as follows: $l$ is the Euclidean distance via (1). The initial angles $\varphi$ and $\lambda$ are obtained from (4) and (5). $\dot{\lambda}$ is computed by the time derivative of (2),

$$\frac{1}{\cos^2\lambda}\dot{\lambda} = \frac{\dot{y}x - y\dot{x}}{x^2}. \tag{A1}$$

$\dot{\varphi}$ is computed by the time derivative of (4),

$$\dot{\varphi}\sin\varphi = \dot{z}/l. \tag{A2}$$

$L_0$ follows from (21), then $\Lambda^2$ from (31), then $D_0$ from (41).

The initial position has 3 coordinates, but the constraint to a sphere of radius $l$ reduces these to 2 parameters, and since the system is invariant with respect to a rotation around the pole axis (in the spirit of the Noether Theorem the 'cause' of the conservation of angular momentum), only $z$ is relevant. The initial velocity has 3 coordinates, but since the motion is bound to the sphere surface, it must be orthogonal to the initial position: $x\dot{x} + y\dot{y} + z\dot{z} = 0$, so 2 velocity parameters are independent. These $1 + 2$ independent coordinates have been transfused to the 2 parameters $\Lambda^2$ and $D_0$ above. The remaining third piece of information hides as the time $T$ since the transit through one of the equilibrium points — or equivalently the amplitudes $\varphi_0$ of these equilibrium points.

The total energy $E = K + V$ is

$$\frac{2E}{ml^2} = \dot{\lambda}^2\sin^2\varphi + \dot{\varphi}^2 - \frac{2g}{l}\cos\varphi = \frac{2g}{l}\left(\frac{\Lambda^2}{\sin^2\varphi} + \varphi'^2 - \cos\varphi\right). \tag{A3}$$

It is constant in time and known from the initial conditions. The equilibrium angles $\varphi_0$ are found by setting $\varphi' = 0$ and solving the emerging cubic equation for $\cos\varphi_0$.

### Appendix B: Antipodal Equilibrium Point

#### 1. Long Division

If $\cos\varphi_0$ is known, the other roots of the cubic polynomial (42) are found by long division of through $\cos\varphi - \cos\varphi_0$. It remains to solve the the quadratic equation

$$\cos^2\varphi + [\cos\varphi_0 + D_0]\cos\varphi - \sin^2\varphi_0 + \cos\varphi_0 D_0 = 0. \tag{B1}$$

According to (41), $\cos\varphi_0 + D_0 > 0$, so we mainly search for for the positive upper sign in

$$\cos\varphi = -\frac{\cos\varphi_0 + D_0}{2} \pm \frac{1}{2}\sqrt{[\cos\varphi_0 + D_0]^2 + 4\sin^2\varphi_0 - 4\cos\varphi_0 D_0} \tag{B2}$$

This establishes the roots $\cos\varphi_u$, $\cos\varphi_l$ and a spurious solution $\cos\varphi_s < -1$ of (42).

### 2. Regularization

Removal of the quadratic term with the standard binomial compensation formula translates Equation (42) of the equilibrium angles into

$$\left[\cos\varphi+\frac{D_0}{3}\right]^3-\left[1+\frac{D_0^2}{3}\right]\left[\cos\varphi+\frac{D_0}{3}\right]+\Lambda^2-\frac{2}{3}D_0=0. \tag{B3}$$

The coefficient in front of the remaining linear term is set to unity by dividing all terms through the 3/2th power of the coefficient of the linear term — which is well defined because positive:

$$\left[\frac{\cos\varphi+\frac{D_0}{3}}{\sqrt{1+D_0^2/3}}\right]^3-\frac{\cos\varphi+\frac{D_0}{3}}{\sqrt{1+D_0^2/3}}+\frac{\Lambda^2-\frac{2}{3}D_0}{(1+D_0^2/3)^{3/2}}=0. \tag{B4}$$

This is a cubic equation of the form (42) at a scaled-shifted unknown $(\cos\varphi+D_0/3)/\sqrt{1+D_0^2/3}$ as if $D_0=0$, as if searching for the roots of the discriminant $D(\varphi)$ itself, and we only need to treat

$$\cos^3\varphi-\cos\varphi+\Lambda^2=0, \tag{B5}$$

a cubic equation for $\cos\varphi$. The roots can be written in terms of cubic roots of functions of $\Lambda$ [11, 3.8.2] All three roots $\cos\varphi^\uparrow$, $\cos\varphi^\downarrow$ and $\cos\varphi^\dagger$ are real if (40) is satisfied; the are related by Vieta's formula [6, 1.6.2.3]: $\cos\varphi^\dagger=-\cos\varphi^\uparrow-\cos\varphi^\downarrow<0$. We sort the others by $\varphi^\downarrow<\varphi^\uparrow$. The larger of these roots of $D$ can be written as a Gaussian Hypergeometric Function [20]

$$\begin{aligned}\cos\varphi^\uparrow&=\Lambda^2+\Lambda^6+3\Lambda^{10}+12\Lambda^{14}+55\Lambda^{18}+273\Lambda^{22}+\cdots\\&=\Lambda^2\sum_{i\geq0}\alpha_i^\uparrow\Lambda^{4i}=\Lambda^2{}_2F_1\left(\begin{matrix}2/3,1/3\\3/2\end{matrix}\,\middle|\,27\Lambda^4/4\right)\end{aligned} \tag{B6}$$

where recursively

$$2i(2i+1)\alpha_i^\uparrow=3(3i-1)(3i-2)\alpha_{i-1}^\uparrow. \tag{B7}$$

In the limit $\Lambda^2\to2/3^{3/2}$ we have $27\Lambda^4/4\to1$ and

$${}_2F_1\left(\begin{matrix}2/3,1/2\\3/2\end{matrix}\,\middle|\,27\Lambda^4/4\right)\overset{\Lambda^2\to2/3^{3/2}}{\longrightarrow}3/2; \tag{B8}$$

$$\cos\varphi^\uparrow\overset{\Lambda^2\to2/3^{3/2}}{\longrightarrow}\frac{1}{\sqrt{3}}. \tag{B9}$$

The smaller of these roots of $D$ is at [21]

$$\begin{aligned}\cos\varphi^\downarrow&=-\frac{1}{2}\cos\varphi^\uparrow+\sqrt{1-\frac{3}{4}\cos^2\varphi^\uparrow}\\&=1-\frac{1}{2}\Lambda^2-\frac{3}{8}\Lambda^4-\frac{1}{2}\Lambda^6-\frac{105}{128}\Lambda^8-\frac{3}{2}\Lambda^{10}-\frac{3003}{1024}\Lambda^{12}+\cdots\\&={}_2F_1\left(\begin{matrix}1/6,-1/6\\1/2\end{matrix}\,\middle|\,27\Lambda^4/4\right)-\frac{1}{2}\Lambda^2{}_2F_1\left(\begin{matrix}1/3,2/3\\3/2\end{matrix}\,\middle|\,27/4\Lambda^4\right).\end{aligned} \tag{B10}$$

By setting $\varphi^\uparrow=\varphi^\downarrow=\hat{\varphi}$, i.e. solving cojointly the Equations (34) and

$$D(\hat{\varphi})=0\Rightarrow\sin^2\varphi\cos\varphi=\Lambda^2 \tag{B11}$$

we arrive at (40).

## Appendix C: Free Ballistic Fall

The model of the mass motion described in this manuscript keeps the mass at constant distance $l$ from the point of suspension. It describes a mass rolling frictionless inside a sphere shell of radius $l$. This is not strictly equivalent to a pendulum realized with a cord, because that cord can pull the mass towards the pole, but cannot push it away. The vertical acceleration of the model mass is obtained by Eqs. (10), (22) and (27):

$$\ddot{z} = l\ddot{\varphi}\sin\varphi + l\cos\varphi\dot{\varphi}^2 = -g + 3g\cos^2\varphi + 2gD_0\cos\varphi. \tag{C1}$$

Free fall sets in, and the mass 'violates' the spherical constraint set forth by the Lagrangian, if it is at an 'overhead' position $\varphi > \pi/2$ — which requires $D_0 > 0$ — and if $\ddot{z} > -g$, that means if

$$D_0 < \frac{3}{2}|\cos\varphi|. \tag{C2}$$

Eq. (41) confirms that this inequality is fulfilled at the farther equilibrium point if $\cos\varphi_u < 0$ and $\Lambda^2 < \frac{1}{2}|\cos\varphi_u|\sin^2\varphi_u$. So a pendulum with a real cord would leave the sphere shell under conditions of that kind.

## Appendix D: C++ implementation

This section contains the source code of a reference implementation which can be compiled with

```
g++ -o spherPend -O2 spherPend.cxx SpherPend.cxx ByrdEllip.cxx -lm -lgsl -lgslcblas
```

It requires the GNU Scientific Library. On some Linux systems the `-lgslcblas` linker option is not needed. The executable can then be called with

`spherPend -x` *x0* `-z` *z0* `-v` *v0* `[-N` *Nperiod*`]` `[-s` *samppp* `]` `[-g` *accel*`]` `[-d]`

The parameter *x0* is the x-coordinate in units of meter at time $t = 0$; *z0* is the z-coordinate in units of meter at time $t = 0$; *v0* is the velocity in units of meter per second at time $t = 0$, *Nperiod* is the number of periods to trace (default 2), *samppp* is the number of sampling points per period (default 200). The optional *accel* defines the acceleration constant of the gravitation in units of meters per second squared (default 9.80665). The optional switch `-d` switches from the computation by the representation of the angles via Elliptic Integrals to a numerical solution of the two differential equation using a predictor scheme up to the fourth order differentials $d^4\theta/dT^4$ and $d^4\lambda/dT^4$. (This is essentially a debugging option to compare the two numerical approaches.)

The executable can then alternatively be called with

`spherPend -D` $D_0$ `-L` $\Lambda$ `[-N` *Nperiod*`]` `[-s` *samppp* `[-d]`

to plot the trajectory given the two unitless parameters $D_0$ and $\Lambda$. Mixing the command line options of these two variants of calling the program yields undefined results.

Trajectories start at $y = 0$ with $\dot{x} = \dot{z} = 0$. The program prints snapshots of positions as a function of time. Each line of the output contains the time $t$ in SI units, the unitless time $T$, the Cartesian coordinates $x$, $y$ and $z$ in SI units, and the angles $\varphi$ and $\lambda$ in radians.

### 1. SpherPend.h

```
#pragma once
/*!*****************************************
* @file
* @brief Declarations of the member functions of the SpherPend class.
*/


/*!*****************************************
* @brief SpherPend solves the coupled equations of motions for a spherical pendulum
* @since 2022-06-12
* @author Richard J. Mathar
*/
class SpherPend {
public:
    /**
    * @brief unitless angular momentum
    */
    double Lambda ;
```

```
    /**
    * @brief unitless energy parameter in the radial equation
    */
    double D0 ;

    /** The main parameter of the Elliptic Functions of the first kind
    */
    double k ;



    /** chord length
    */
    double l ;


    /* the sqrt(l/2g) that converts from SI to
    * unitless variables
    */
    double siFactor ;

        SpherPend(const double L, const double Dzero) ;

        SpherPend(const double xStart, const double zStart, const double velStart, const double g=9.80665) ;

    void tableTraject(double periods, int samplPer, bool numer) ;

    double qperiod() ;

    bool checkCubic() ;
protected:
        void solveTurns() ;


    /** the 3 real-valued roots for the cosine(phi)
    */
    double thetaRoots[3] ;

        static void solveCubic(const double b, const double c, const double d, double roots[3]) ;

        static void solveCubic(const double p, const double q, double roots[3]) ;

private:
} ;
```

## 2. SpherPend.cxx

```
/*!*****************************************************
* @file
* @brief SpherPend is a class to solve the equations of motions of the spherical pendulum.
*
*/

#include <cstdlib>
#include <cmath>
#include <iostream>

/* headers of the GNU scientific library
* of https://www.gnu.org/software/gsl
*/
#include <gsl/gsl_sf_elljac.h>
#include <gsl/gsl_sf_ellint.h>
#include <gsl/gsl_poly.h>

#include "SpherPend.h"
#include "ByrdEllip.h"

using namespace std ;

/** Ctor given the two master parameters.
* @param[in] L The unitless angular momentum, capital-Lambda.
* @param[in] Dzero The unitless master parameter for the diff equation of the radial amplitude
* @author Richard J. Mathar
*/
SpherPend::SpherPend(const double L, const double Dzero)
{
    Lambda =L ;
    D0 = Dzero ;
    siFactor = 1.0 ;
```

```
    /* adjust chord length such that the siFactor remains 1.
    */
    l = 2*9.80665 ;

    /* initialize the three values of thetaRoots[0..2]
    */
    if ( checkCubic() )
    {
        solveTurns () ;
        k = sqrt((thetaRoots[2]-thetaRoots[1])/(thetaRoots[2]-thetaRoots[0])) ;
    }
} /* ctor */

/** Ctor given the start position and velocity.
* @param[in] xStart x coordinate in meters at time 0
* @param[in] zStart z coordinate in meters at time 0
*    Note that the ystart parameter is always zero.
* @param[in] velStart velocity at time 0 in meters per second
* @param[in] g gravitational acceleration in meters per second squared.
* @since 2022-06-16
* @author Richard J. Mathar
*/
SpherPend::SpherPend(const double xStart, const double zStart, const double velStart, const double g)
{

    /* chord length = sqrt(x^2+y^2+z^2)
    */
    l = hypot(xStart,zStart) ;

    siFactor = sqrt(0.5*l/g) ;

    /* initial velocity is ydot0 at chord length l which gives an
    * angular velocity of lambda dot = ydot0/l
    * Initial angle phi is sin phi = xStart/l at time zero.
    */
    double sinphiStart = xStart/l ;

    /* sin^2(phi) at time 0
    */
    double sinphiStartSq = pow(sinphiStart,2.) ;

    /* lambda dot a time zero  in radians per second.
    */
    double lambdadotStart = velStart/l ;

    /* L0 is product sin^2(phi)*lambda do
    */
    double L0 = sinphiStartSq*lambdadotStart ;

    /* unitless Lambda
    */
    Lambda = L0*sqrt(l/2/g) ;

    /* D(phi) = Lambda^2/sin^2phi -cos (phi) where cos(phi)=sqrt(1-sin^2phi)
    */
    D0 = pow(Lambda/sinphiStart,2.0)-sqrt(1.-sinphiStartSq) ;

    /* initialize the three values of thetaRoots[0..2]
    */
    if ( checkCubic() )
    {
        solveTurns () ;
        k = sqrt((thetaRoots[2]-thetaRoots[1])/(thetaRoots[2]-thetaRoots[0])) ;
    }
} /* ctor */

#define SPHER_PEND_SWAP_2(a,b) {const double tmp = b; b=a ; a= tmp ; } ;

/** Solve a cubic equation assuming 3 real roots.
* The cubic equation is x^3+b*x^2+c*x+d=0.
* @param[in] d coefficient of absolute term.
* @param[in] c coefficient of linear term.
* @param[in] b coefficient of quadratic term.
* @param[out] roots The 3 real-valued solutions.
*   If there are no 3 real-valued solutions, the values are undefined.
* @author Richard J. Mathar
*/
void SpherPend::solveCubic(const double b, const double c, const double d, double roots[3])
{
#if 0
    /* Pederstrian approach without the GSL: solve the reduced equation first
    */
    double p = c-b*b/3.0 ;
```

```
    double q = b*(2.*b*b/9.0-c)/3.0 +d ;
    solveCubic(p,q,roots) ;
    /* shift the solutions of the reduced equation to the solutions of the
    * equation with the quadratic term
    */
    for (int i =0 ; i < 3; i++)
        roots[i] -= b/3.0 ;

    /* sort the 3 values algebraically
    */
    if ( roots[0] > roots[1])
        SPHER_PEND_SWAP_2(roots[0], roots[1])

    if ( roots[1] > roots[2])
        SPHER_PEND_SWAP_2(roots[1], roots[2])

    if ( roots[0] > roots[1])
        SPHER_PEND_SWAP_2(roots[0], roots[1])
#else
    /* since we use the GSL anyway, we may as well use it here
    */
    gsl_poly_solve_cubic(b,c,d, &roots[0],&roots[1],&roots[2] ) ;
#endif

} /* solveCubic */

#undef SPHER_PEND_SWAP_2

/** Solve a cubic equation with Cardano's formula.
* The cubic equation is t^3+p*t+q=0.
* @param[in] p coefficient of the linear term.
* @param[in] q coefficient of absolute term.
* @param[out] roots The 3 real-valued solutions.
*   If there are no 3 real-valued solutions, the values are not touched.
* @author Richard J. Mathar
*/
void SpherPend::solveCubic(const double p, const double q, double roots[3])
{
    const double disc = 27.*q*q+4.*p*p*p ;
    if ( disc < 0)
    {
        const double negpthird = sqrt(-p/3.0) ;
        const double cphase = acos( 1.5*q/p/negpthird ) ;
        for(int i=0 ; i < 3 ; i++)
            roots[i] = 2.0*negpthird*cos((cphase-2.*M_PI*i)/3.) ;
    }
} /* solveCubic */

/** Determine the theta-values of the turning points from the cubic equation
* @author Richard J. Mathar
*/
void SpherPend::solveTurns()
{
    solveCubic(D0, -1.0, Lambda*Lambda-D0, thetaRoots) ;
} /* solveCubic */

/** compute the quarter period in scaled units of time T
* @return 2*K(k)/sqrt(theta_u-theta_s) .
*/
double SpherPend::qperiod()
{
    return 2.*gsl_sf_ellint_Kcomp(k,GSL_PREC_DOUBLE) /sqrt(thetaRoots[2]-thetaRoots[0]) ;

} /* qperiod */

/** validate that the cubic characteristic equation has 3 real roots.
* @author Richard J. Mathar
* @return true if this is a realistic system with 3 real roots of the radial equation.
*/
bool SpherPend::checkCubic()
{
    /* parameters of the coefficients are
    * t^3 + D0 t^2 - t + Lambda^2-D0=0. a=1, b=D0, c=-1, d=L^2-D0.
    * The two prameters of the Cardan form are
    * p = c-b*b/3.0 = -1-D0^2/3 ;
    * q = 2*b^3/27 - b*c/3 +d = 2 D0^3/27 +D0/3 +L^2-D0
    *    = 2 D0^3/27 -2 D0/3 +L^2 ;
    */
    const double p = -1. -D0*D0/3.0 ;
    const double q = 2.*D0*(D0*D0/9.0-1.)/3.0 + Lambda*Lambda ;

    const double disc = 27.*q*q+4.*p*p*p ;
    /* need a negative value of 27q^2+4p^3 for 3 real-valued roots
```

```
    */
    return (disc < 0.0) ;
} /* checkCubic */

/** Tabulate the trajectory
* Apart from comment lines tha tstart with the hash sign the
* columns are:
*    time t in units of seconds
*    unitless time T
*    x coordinate in meters
*    y coordinate in meters
*    z coordinate in meters
*    angle phi in radians
*    angle lambda in radians
* @param[in] periods The number of periods to trace
* @param[in] samplPer The number of samples plotted in each period
* @param[in] numer Flag to trigger numerical solution.
*   If the flag is 'true', the differential equations will be
*   solved numerically with a preditor method of 4th order in
*   the derivatives. If the flag is 'false', the solution
*   with the Elliptic Integrals is printed.
*
*/
void SpherPend::tableTraject(double periods, int samplPer, bool numer)
{
    /* compute the length of the quarter period in unitless time
    */
    double Pqart = qperiod() ;

    cout << "# D0 = " << D0 << " Lambda = " << Lambda << endl ;
    cout << "# t T x y z phi lambda" << endl ;

    /* 3 cartesian coordinates x, y, z
    */
    double cartCoor[3] ;
    if ( numer)
    {
        /* if this flag is raised, the differential
        * equations are solved by a standard numerical predictor
        * integration up to 4th order derivatives as a function of time. This is merely a
        * numerical check that the main implementation is consistent with this
        */
        if ( thetaRoots[0] < thetaRoots[1]-1.e-5)
        {
            double theta = thetaRoots[1] ;
            double lambda = 0.0 ;
            const double deltaT = 1.0/samplPer*4*Pqart ;
            /* todo: relate T not to T but to periods... factor
            */
            for(double tsca=0.0 ; tsca < periods ; tsca += 1.0/samplPer)
            {
                double T= tsca*4*Pqart ;

                int lambdaTurns ;
                lambda = std::remquo(lambda, 2.*M_PI, & lambdaTurns) ;

                const double phi = acos(theta) ;
                cartCoor[0] = l * sin( phi)*cos(lambda) ;
                cartCoor[1] = l * sin( phi)*sin(lambda) ;
                cartCoor[2] = -l * theta ;
                std::cout
                    << T*siFactor
                    << " " << T
                    << " " << cartCoor[0]
                    << " " << cartCoor[1]
                    << " " << cartCoor[2]
                    << " " << phi
                    << " " << lambda
                    << std::endl ;

                /* dtdT[n] is the n'th derivative d^n theta/dT^n
                */
                double dtdT[5] ={0.,0.,0.,0.,0.};
                dtdT[1] = sqrt( theta*(1.-(theta+D0) *theta) -Lambda*Lambda+D0 ) ;
                /* value under the square root may be slightly negative by rounding
                */
                if ( isnan(dtdT[1]) )
                    dtdT[1] =0. ;

                /* sign of the first derivative flips after each quarter period
                * and the square root above was always taken with the positive sign.
                * write T=alpha*P/4 and + for alpha<1 - for alpha<2 + foralph<3 tec
                */
```

```
            const int Nqper = T/Pqart ;
            if ( Nqper %2 )
                dtdT[1] *= -1. ;

            /* 2nd derivative d theta^2/dT^2 = 1/2-3/2*theta^2 -D0*theta
            */
            dtdT[2] = 0.5-theta*(D0+1.5*theta) ;

            /* 3rd derivative d theta^3/dT^3 = (-3*theta-D0)*theta'
            */
            dtdT[3] = (-3.0*theta-D0)*dtdT[1] ;

            /* 4th derivative d theta^4/dT^4 = -3*theta'^2-(3*theta+D0)*theta''
            */
            dtdT[4] = -3.0*pow(dtdT[1],2.0)-(3*theta+D0)*dtdT[2] ;

            /* main numerator of lambda derivatives, 1/(1-theta^2)
            */
            const double invt2 = 1./(1.-theta*theta) ;

            /* first derivatie d lambda/d T= Lambda/(1-theta^2)
            */
            double dlambdadT[5] = {0.,0.,0.,0.,0.} ;
            dlambdadT[1] = Lambda* invt2 ;

            /* 2nd derivative d lambda/d T= Lambda*2*theta *theta'/(1-theta^2)^2
            * = 2*theta*theta'*lambda'/(1-theta^2)
            */
            dlambdadT[2] = 2.*theta*dtdT[1]*dlambdadT[1] * invt2 ;

            /* 3rd derivative
            * 2*Lambda*[(theta'^2+theta theta')/(1-theta^2)^2 + (2theta theta')^2/(1-theta^2)^3 ]
            */
            dlambdadT[3] = 2.*Lambda*( pow(dtdT[1],2.0)+theta*dtdT[2]
                +invt2*pow(2.*theta*dtdT[1],2.0) )
                *pow(invt2,2.) ;

            /* 4th derivative
            */
            dlambdadT[4] = 2.*Lambda*(
                24.*pow(theta*dtdT[1],3.0)*pow(invt2,2.)
                +12.*theta*pow(dtdT[1],3.0)*invt2
                +12.*pow(theta,2.0)*dtdT[1]*dtdT[2]*invt2
                +3.*dtdT[1]*dtdT[2]
                +theta*dtdT[3])
                *pow(invt2,2.) ;

            /* update theta and lambda with the standard
            * initial terms of a power series of deltaT.
            */
            theta += dtdT[1]*deltaT
                + dtdT[2]*pow(deltaT,2.)/2.0
                +dtdT[3]*pow(deltaT,3.)/6.
                +dtdT[4]*pow(deltaT,4.)/24.
                ;

            lambda += dlambdadT[1]*deltaT
                + dlambdadT[2]*pow(deltaT,2.)/2.0
                + dlambdadT[3]*pow(deltaT,3.)/6.0
                + dlambdadT[4]*pow(deltaT,4.)/24.0
                ;
        }
    }
}
else
{
    const double deltaT = 1.0/samplPer*4*Pqart ;

    /* complete lambda over a quarter period, which is
    * the linear drift of the lambda angle derived from
    * the complete elliptic integrals.
    */
    const double lambdacomp = Lambda*(
        ByrdEllip::B23517comp(-1.,thetaRoots[2],thetaRoots[1],thetaRoots[0],1)
        -
        ByrdEllip::B23517comp(1.,thetaRoots[2],thetaRoots[1],thetaRoots[0],1)
        )/2. ;

    /* loop over periods of time (i.e. full periods of radial equation)
    */
    for(double tsca=0.0 ; tsca < periods ; tsca += 1.0/samplPer)
    {
        double T= tsca*4*Pqart ;
```

```
            double u = 0.5*sqrt(thetaRoots[2]-thetaRoots[0])*T ;

            double sn, cn, dn ;
            /* gather the Jacobian Elliptic functions at m=k^2
            */
            gsl_sf_elljac_e(u, k*k, & sn, & cn, & dn) ;

            /* now sin phi =sn, and theta derived from there
            */
            double k2sin2phi = pow(k*sn,2.0) ;
            double theta = thetaRoots[1]+(thetaRoots[1]-thetaRoots[0])*k2sin2phi/(1-k2sin2phi) ;

            /* slice tsca in quarter periods 0, 1, 2, 3, 4,...
            * in 0..1 lambda is the formula value, in 1..2 it is 2 times the complete
            * value minus the  formula, in 2..3 it is 2 times the complete vlue plus
            * the formula and so on.
            */
            int qperiods = T/Pqart ;


            double lambda = Lambda*(
            ByrdEllip::B23517(-1.,thetaRoots[2],thetaRoots[1],thetaRoots[0],theta,1)
            -
            ByrdEllip::B23517(1.,thetaRoots[2],thetaRoots[1],thetaRoots[0],theta,1)
            )/2. ;

            if ( qperiods % 2)
            {
                /* quarter period 1,3,5,... */
                lambda = lambdacomp *(1+qperiods)-lambda ;
            }
            else
            {
                /* quarter period 0,2,4,... */
                lambda += lambdacomp *qperiods ;
            }

            int lambdaTurns ;
            lambda = std::remquo(lambda, 2.*M_PI, & lambdaTurns) ;

            const double phi = acos(theta) ;
            cartCoor[0] = l * sin( phi)*cos(lambda) ;
            cartCoor[1] = l * sin( phi)*sin(lambda) ;
            cartCoor[2] = -l * theta ;
            std::cout
                << T*siFactor
                << " " << T
                << " " << cartCoor[0]
                << " " << cartCoor[1]
                << " " << cartCoor[2]
                << " " << phi
                << " " << lambda
                << std::endl ;
        }
    }
} /* tableTraject */
```

## 3. ByrdEllip.h

```
#pragma once
/*!*********************************************
* @file
* @brief Declarations of the member functions of the ByrdEllip class.
*/


/*!*****************************************
* @brief Elliptic integrals of the Byrd-Friedmann book implemented via the GSL.
* @since 2022-06-15
* @author Richard J. Mathar
*/
class ByrdEllip {
public:

    static double B23517(const double p, const double a, const double b, const double c, const double y, int m) ;
    static double B23517comp(const double p, const double a, const double b, const double c, int m) ;

protected:
    static double B33904(const double alphasq, const double ksq, const double phi, const int m) ;
```

```
    static double B33904comp(const double alphasq, const double ksq, const int m) ;

    static double B336(const double alphasq, const double ksq, const double phi, const int m) ;
    static double B336comp(const double alphasq, const double ksq, const int m) ;

private:
} ;
```

## 4. ByrdEllip.cxx

```
/*!*****************************************************
* @file
* @brief ByrdEllip computes an integral of the Byrd-Friedmann book related to the spherical pendulum
*
*/

#include <cmath>
#include <iostream>

/* headers of the GNU scientific library
* of https://www.gnu.org/software/gsl
*/
#include <gsl/gsl_sf_elljac.h>
#include <gsl/gsl_sf_ellint.h>

#include "ByrdEllip.h"

using namespace std ;

/* Integral dx 1/(x-p)^m/sqrt[(a-x)(x-b)(x-c)] in formula (235.17)
*   Requires that a>=y>b>c and m=0 or 1.
* @param p constant in the linear factor
* @param a constant in the square root
* @param b constant in the square root
* @param c constant in the square root
* @param y Upper limit of the rgion of integration
* @param m Power in the denominator
* @return integral _b^y 1/(t-p)^m/sqrt[(a-t)(t-b)(t-c)]
*   If the requirements are not fulfilled, zero is returned.
* @todo throw an applicable parameter exception.
* @since 2022-05-15
*/
double ByrdEllip::B23517(const double p, const double a, const double b, const double c, const double y, int m)
{
    if ( a < y || y<= b || b<= c)
        /* invalid parameter region */
        return 0. ;

    /* constants prior to equation 235.00
    */
    const double g=2./sqrt(a-c) ;
    /* k^2
    */
    const double ksq = (a-b)/(a-c) ;
    /* angle phi in radians
    */
    const double phi= asin( sqrt((a-c)*(y-b)/(a-b)/(y-c))) ;
    /* alpha ^2
    */
    double alphasqr ;
    switch(m)
    {
    case 0:
        /* this is actually 235.00 , a special case of 339.04 */
        return g*gsl_sf_ellint_F(phi,sqrt(ksq),GSL_PREC_DOUBLE) ;
    case 1:
        alphasqr = (a-b)*(c-p)/(a-c)/(b-p) ;
        return g/(b-p)*B33904(alphasqr,ksq,phi,m) ;
        /* 235.17 */
    default:
        alphasqr = (a-b)*(c-p)/(a-c)/(b-p) ;
        return g/pow(b-p,m)*B33904(alphasqr,ksq,phi,m) ;
    }
} /* B23517 */

/* Complete integral dx 1/(x-p)^m/sqrt[(a-x)(x-b)(x-c)] in formula (235.17)
*   Requires that a>=y>b>c and m=0 or 1.
* @param p constant in the linear factor
* @param a constant in the square root
```

```
* @param b constant in the square root
* @param c constant in the square root
* @param m Power in the denominator
* @return integral _b^y 1/(t-p)^m/sqrt[(a-t)(t-b)(t-c)]
*   If the requirements are not fulfilled, zero is returned.
* @todo throw an applicable parameter exception.
* @since 2022-05-15
*/
double ByrdEllip::B23517comp(const double p, const double a, const double b, const double c, int m)
{
    if ( a < b || b<= c)
        /* invalid parameter region */
        return 0. ;

    /* constants prior to equation 235.00
    */
    const double g=2./sqrt(a-c) ;
    const double ksq = (a-b)/(a-c) ;
    double alphasqr ;
    switch(m)
    {
    case 0:
        /* this is actually 235.00 , a special case of 339.04 */
        return g*gsl_sf_ellint_Kcomp(sqrt(ksq),GSL_PREC_DOUBLE) ;
    case 1:
        alphasqr = (a-b)*(c-p)/(a-c)/(b-p) ;
        return g/(b-p)*B33904comp(alphasqr,ksq,m) ;
        /* 235.17 */
    default:
        alphasqr = (a-b)*(c-p)/(a-c)/(b-p) ;
        return g/pow(b-p,m)*B33904comp(alphasqr,ksq,m) ;
    }
} /* B23517comp */

/**
* @brief The integral (339.04) in the Byrd-Friedmann book.
* @param alphasq The alpha^2
* @param ksq The k^2
* @param phi The phase (radians, upper limit in the Legendre forms)
* @param m integer parameter in the exponent
* @since 2022-05-15
*/
double ByrdEllip::B33904(const double alphasq, const double ksq, const double phi, const int m)
{
    double Sm=0.0 ;
    /* binomial coefficient (m,j) initialized at j=0
    */
    double mchoosej = 1.0 ;
    for(int j=0 ; j <= m ; j++)
    {
        Sm += pow(alphasq/ksq-1.0,j) *mchoosej *B336(alphasq,ksq,phi,j) ;
        /* upgrade binomial(m,j) to binomial(m,j+1)
        * for the next loop
        */
        mchoosej *= (double)(m-j)/(j+1.0) ;
    }
    /* multiply by k^(2m)/alpha^(2m)
    */
    return Sm *pow(ksq/alphasq,m) ;
} /* B33904 */

/**
* @brief The complete integral (339.04) in the Byrd-Friedmann book.
* @param alphasq The alpha^2
* @param ksq The k^2
* @param m integer parameter in the exponent
* @since 2022-05-15
*/
double ByrdEllip::B33904comp(const double alphasq, const double ksq, const int m)
{
    double Sm=0.0 ;
    /* binomial coefficient (m,j) initialized at j=0
    */
    double mchoosej = 1.0 ;
    for(int j=0 ; j <= m ; j++)
    {
        Sm += pow(alphasq/ksq-1.0,j) *mchoosej *B336comp(alphasq,ksq,j) ;
        /* upgrade binomial(m,j) to binomial(m,j+1)
        * for the next loop
        */
        mchoosej *= (double)(m-j)/(j+1.0) ;
    }
    /* multiply by k^(2m)/alpha^(2m)
```

```
    */
    return Sm *pow(ksq/alphasq,m) ;
} /* B33904comp */

/**
* @brief The integral V_m defined prior to (336.00) in the Byrd-Friedmann book.
* @param[in] alphasq The alpha^2
* @param[in] ksq The k^2
* @param[in] phi The angle phi as one form of the upper limit
* @param[in] m integer parameter in the exponent
*      This is not the m-parameter of the Abramowitz-Stegun notation
* @return integral^phi d phi/[(1-alphasq*sin^2 phi)^m*sqrt(1-k^2*sin^2 phi)]
* @since 2022-05-15
*/
double ByrdEllip::B336(const double alphasq, const double ksq, const double phi, const int m)
{
    /* sn(u,k), cn(u,k) and dn(u,k) as a vector
    */
    double sncndn[3] ;
    const double k = sqrt(ksq) ;
    const double u = gsl_sf_ellint_F(phi,k,GSL_PREC_DOUBLE) ;

    if ( m >= 2)
    {
        /* the values are not needed for m=0,1 so we skip
        * the evaluation in these cases, even if that may confuse some compilers.
        */
        gsl_sf_elljac_e(u,ksq,& sncndn[0], & sncndn[1], & sncndn[2]) ;
    }

    switch(m)
    {
    case 0:
        /* equation 336.00
        */
        return u ;
    case 1:
        /* equation 336.01
        */
        /* the GSL uses the paramater n = - alpha^2 as the last of the three
        */
        return gsl_sf_ellint_P(phi,k,-alphasq,GSL_PREC_DOUBLE) ;
    case 2:
        /* equation 336.02. This case and the default are actually never called
        * in conjunction with the solution of the spherical pendulum.
        * The GSL uses the parameter n = - alpha^2 as the last of the three
        */
        return ( alphasq*gsl_sf_ellint_E(phi,k,GSL_PREC_DOUBLE)
            +(ksq-alphasq)*u
            +(2*alphasq*(ksq+1.0)-alphasq*alphasq-3*ksq)*gsl_sf_ellint_P(phi,k,-alphasq,GSL_PREC_DOUBLE)
            -alphasq*alphasq*sncndn[0]*sncndn[1]*sncndn[2]/(1-alphasq*sncndn[0]*sncndn[0])
            )
            /2./(alphasq-1.0)/(ksq-alphasq) ;

    default:
        /* equation 336.03 after setting m->m-3
        */
        return ( (2*m-5) *ksq *B336(alphasq,ksq,phi,m-3)
            +2*(m-2) * (alphasq*(ksq+1)-3*ksq) *B336(alphasq,ksq,phi,m-2)
            +(2*m-3) *(alphasq*(alphasq-2*ksq-2)+3*ksq) *B336(alphasq,ksq,phi,m-1)
            +alphasq*alphasq*sncndn[0]*sncndn[1]*sncndn[2]/pow(1-alphasq*sncndn[0]*sncndn[0],m+2.)
            )
            /2./(m-1.0)/(1.0-alphasq)/(ksq-alphasq) ;
    }
} /* B336 */

/**
* @brief The integral V_m defined prior to (336.00) in the Byrd-Friedmann book.
* @param[in] alphasq The alpha^2
* @param[in] ksq The k^2
* @param[in] m integer parameter in the exponent
*      This is not the m-parameter of the Abramowitz-Stegun notation
* @return integral^phi d phi/[(1-alphasq*sin^2 phi)^m*sqrt(1-k^2*sin^2 phi)]
* @since 2022-05-15
*/
double ByrdEllip::B336comp(const double alphasq, const double ksq, const int m)
{
    const double k = sqrt(ksq) ;
    const double u = gsl_sf_ellint_Kcomp(k,GSL_PREC_DOUBLE) ;


    switch(m)
    {
```

```
    case 0:
        /* equation 336.00
        */
        return u ;
    case 1:
        /* equation 336.01
        */
        /* the GSL uses the paramater n = - alpha^2 as the last of the three
        */
        return gsl_sf_ellint_Pcomp(k,-alphasq,GSL_PREC_DOUBLE) ;
    case 2:
        /* equation 336.02
        */
        /* the GSL uses the parameter n = - alpha^2 as the last of the three
        * Difference to the inomplete case is that cn(u)=0 and one term vanishes
        */
        return ( alphasq*gsl_sf_ellint_Ecomp(k,GSL_PREC_DOUBLE)
            +(ksq-alphasq)*u
            +(2*alphasq*(ksq+1.0)-alphasq*alphasq-3*ksq)*gsl_sf_ellint_Pcomp(k,-alphasq,GSL_PREC_DOUBLE)
            )
            /2./(alphasq-1.0)/(ksq-alphasq) ;

    default:
        /* equation 336.03 after setting m->m-3
        * Difference to the inomplete case is that cn(u)=0 and one term vanishes
        */
        return ( (2*m-5) *ksq *B336comp(alphasq,ksq,m-3)
            +2*(m-2) * (alphasq*(ksq+1)-3*ksq) *B336comp(alphasq,ksq,m-2)
            +(2*m-3) *(alphasq*(alphasq-2*ksq-2)+3*ksq) *B336comp(alphasq,ksq,m-1)
            )
            /2./(m-1.0)/(1.0-alphasq)/(ksq-alphasq) ;
    }
} /* B336comp */
```

## 5. spherPend.cxx

```
/*!*****************************************************
* @file
* @brief SpherPend tabulates the trajectory of a spherical pendulum as a function of time.
*
*/

#include <unistd.h>
#include <cmath>
#include <iostream>

#include <gsl/gsl_sf_ellint.h>
#include "SpherPend.h"

using namespace std ;



void usage(char *argv0)
{
    cout << "usage: " << argv0 << "-x x0 -z z0 -v v0 [-N nperiods] [-s nsamples] [-g accg] [-d]" << endl ;
    cout << "usage: " << argv0 << "-D D0 -L Lambda [-N nperiods] [-s nsamples] [-d]" << endl ;
}

/**
*/
int main(int argc, char *argv[])
{
    /* acceleration in meters per second squared. Default
    * is the normal acceleration on the Earth's surface
    */
    double g = 9.80665 ;

    /* unit less parameter in the radial equation of the radial angle
    */
    double D0 ;
    /* unit less parameter of the angular momentum
    */
    double Lambda =0.0;

    /* number of periods to trace/print
    */
    int N=2 ;

    /* number of samples (refinement) along the trajectory in each period
```

```
    */
    int samp=200 ;

    /** false means derive trajectories via Elliptic Integrals,
    * true means use numerical intgration scheme to integrate
    * the differential equations in steps defined by samp.
    */
    bool debug=false ;

    /* option character
    */
    char oc ;

    /** initial conditions: two parameters of position
    * and one parameter of velocity. Trajectories start
    * horizontally at y=0.
    */
    double xStart, zStart, velStart ;

    while (  (oc=getopt(argc,argv,"D:L:N:s:dx:z:v:g:")) != -1 )
    {
        switch(oc)
        {
        case 'D' :
            D0 = atof( optarg) ;
            break ;
        case 'L' :
            Lambda = atof( optarg) ;
            break ;
        case 'N' :
            N = atoi( optarg) ;
            break ;
        case 'd' :
            debug = true ;
            break ;
        case 'g' :
            g = atof(optarg) ;
            break ;
        case 's' :
            samp = atoi( optarg) ;
            break ;
        case 'v' :
            velStart = atof(optarg) ;
            break ;
        case 'x' :
            xStart = atof(optarg) ;
            break ;
        case 'z' :
            zStart = atof(optarg) ;
            break ;
        case '?' :
            std::cerr << "Invalid command line option " << optopt << std::endl ;
            break ;
        }
    }



    SpherPend *p ;

    if ( Lambda > 0.0)
        p = new SpherPend(Lambda,D0) ;
    else
        p = new SpherPend(xStart, zStart, velStart, g) ;

    if ( p->checkCubic() )
    {
        if ( p->Lambda == 0.0)
            std:cerr << "Cannot solve the planar pendulum" << endl ;
        else
            p->tableTraject(N,samp,debug) ;
        delete p ;
        return 0 ;
    }
    else
    {
        cerr << "Invalid parameter domain" << endl ;
        delete p ;
        return 1 ;
    }
} /* main */
```

[1] E. Horozov, J. reine angew. Math. **408**, 114 (1990).
[2] E. Horozov, Phys. Lett. A **173**, 279 (1993).
[3] P. H. Richter, H. R. Dullin, H. Waalkens, and J. Wiersig, J. Phys. Chem. **100**, 19124 (1996).
[4] R. D. Peters, Phys. Rev. A **38**, 5352 (1988).
[5] I. Castro, I. Castro-Infantes, and J. Castro-Infantens, arXiv:2111.00458 (2021).
[6] I. N. Bronshtein, K. A. Semendyayev, G. Musiol, and H. Muehlig, *Handbook of Mathematics*, 5th ed. (Springer, 2007).
[7] F. Ayres, *Differentialgleichungen*, Schaum's Outline (McGraw-Hill, New York, 1979).
[8] K. R. Meyer, Am. Math. Monthly **108**, 729 (2001).
[9] A. Weinstein, Am. Math. Monthly **49**, 521 (1942).
[10] M. S. Albert and J. E. Marsden, Fields Inst. Commun. **8** (1995).
[11] M. Abramowitz and I. A. Stegun, eds., *Handbook of Mathematical Functions*, 9th ed. (Dover Publications, New York, 1972).
[12] I. Gradstein and I. Ryshik, *Summen-, Produkt- und Integraltafeln*, 1st ed. (Harri Deutsch, Thun, 1981).
[13] P. F. Byrd and M. D. Friedman, *Handbook of Elliptical Integrals for Engineers and Physicists*, 2nd ed., Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen, Vol. 67 (Springer, Berlin, Göttingen, 1971).
[14] W. Magnus, F. Oberhettinger, and R. P. Soni, eds., *Formulas and Theorems for the Special Functions of Mathematical Physics*, 3rd ed., Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen, Vol. 52 (Springer, Berlin, Heidelberg, 1966).
[15] O. E. I. S. Foundation Inc., (2022), https://oeis.org/.
[16] A. Fransén, Math Comp. **37**, 475 (1981).
[17] G. Viennot, J. Combin. Theory A **29**, 121 (1980).
[18] G. J. Cooper, Comp. J. **10**, 94 (1967).
[19] H. R. Dullin, J. Diff. Equat. **254**, 2942 (2013).
[20] M. L. Glasser, J. Comp. Appl. Math. **118**, 169 (2000).
[21] H. F. Sandham, Quart. J. Math. **7**, 153 (1956).