Least Action Principle
and the $n$-Body Problem

Robert J. Vanderbei

February 6, 2002
Seminar on Celestial Mechanics
Program in Applied and Computational Mathematics
Princeton University

Operations Research and Financial Engineering, Princeton University
http://www.princeton.edu/~rvdb
1 Least Action Principle

Given: \( n \) bodies.

Let: 
\( m_j \) denote the mass and 
\( z_j(t) \) denote the position in \( \mathbb{R}^2 = \mathbb{C} \) of body \( j \) at time \( t \).

Action Functional:

\[
A = \int_0^{2\pi} \left( \sum_j \frac{m_j}{2} \|\dot{z}_j\|^2 + \sum_{j,k:k<j} \frac{m_j m_k}{\|z_j - z_k\|} \right) dt.
\]
2 Equation of Motion

First Variation:

\[ \delta A = \int_0^{2\pi} \left( \sum_j \sum_\alpha m_j \dot{z}_j^\alpha \delta z_j^\alpha - \sum_{j,k:k<j} \sum_\alpha m_j m_k \frac{(z_j^\alpha - z_k^\alpha)(\delta z_j^\alpha - \delta z_k^\alpha)}{\|z_j - z_k\|^3} \right) dt \]

\[ = - \int_0^{2\pi} \sum_j \sum_\alpha \left( m_j \ddot{z}_j^\alpha + \sum_{k:k\neq j} m_j m_k \frac{z_j^\alpha - z_k^\alpha}{\|z_j - z_k\|^3} \right) \delta z_j^\alpha dt \]

Setting first variation to zero, we get:

\[ m_j \ddot{z}_j^\alpha = - \sum_{k:k\neq j} m_j m_k \frac{z_j^\alpha - z_k^\alpha}{\|z_j - z_k\|^3}, \quad j = 1, 2, \ldots, n, \quad \alpha = 1, 2 \]

Note: If \( m_j = 0 \) for some \( j \), then the first order optimality condition reduces to \( 0 = 0 \), which is not the equation of motion for a massless body.
3 Second Variation

\[ \delta^2 A = \int_0^{2\pi} \sum_j \sum_\alpha \left( \dot{z}_j^\alpha \right)^2 \, dt \]

\[ + 3 \int_0^{2\pi} \sum_{j,k:j \neq k} \sum_\alpha,\beta \frac{(z_j^\alpha - z_k^\alpha)(z_j^\beta - z_k^\beta)(\delta z_j^\beta - \delta z_k^\beta)}{\|z_j - z_k\|^5} \delta z_j^\alpha \, dt \]

\[ - \int_0^{2\pi} \sum_{j,k:j \neq k} \sum_\alpha \frac{\delta z_j^\alpha - \delta z_k^\alpha}{\|z_j - z_k\|^3} \delta z_j^\alpha \, dt \]
4 Periodic Solutions

We assume solutions can be expressed in the form

\[ z_j(t) = \sum_{k=-\infty}^{\infty} \gamma_k e^{ikt}, \quad \gamma_k \in \mathbb{C}. \]

Writing with components

\[ z_j(t) = (x_j(t), y_j(t)) \quad \text{and} \quad \gamma_k = (\alpha_k, \beta_k), \]

we get

\[ x(t) = a_0 + \sum_{k=1}^{\infty} (a_k^c \cos(kt) + a_k^s \sin(kt)) \]

\[ y(t) = b_0 + \sum_{k=1}^{\infty} (b_k^c \cos(kt) + b_k^s \sin(kt)) \]

where

\[ a_0 = \alpha_0, \quad a_k^c = \alpha_k + \alpha_{-k}, \quad a_k^s = \beta_{-k} - \beta_k, \]
\[ b_0 = \beta_0, \quad b_k^c = \beta_k + \beta_{-k}, \quad b_k^s = \alpha_k - \alpha_{-k}. \]

The variables \( a_0, a_k^c, a_k^s, b_0, b_k^c, \) and \( b_k^s \) are the decision variables in the optimization model.
5 The AMPL Model

param N := 3;  # number of masses
param n := 15;  # number of terms in Fourier series representation
param m := 100;  # number of terms in numerical approx to integral

param theta {j in 0..m-1} := j*2*pi/m;

param a0 {i in 0..N-1} default 0; param b0 {i in 0..N-1} default 0;
var as {i in 0..N-1, k in 1..n} := 0; var bs {i in 0..N-1, k in 1..n} := 0;
var ac {i in 0..N-1, k in 1..n} := 0; var bc {i in 0..N-1, k in 1..n} := 0;

var x {i in 0..N-1, j in 0..m-1}
  = a0[i]+sum {k in 1..n} ( as[i,k]*sin(k*theta[j]) + ac[i,k]*cos(k*theta[j]) )
var y {i in 0..N-1, j in 0..m-1}
  = b0[i]+sum {k in 1..n} ( bs[i,k]*sin(k*theta[j]) + bc[i,k]*cos(k*theta[j]) )

var xdot {i in 0..N-1, j in 0..m-1}
  = if (j<m-1) then (x[i,j+1]-x[i,j])*m/(2*pi) else (x[i,0]-x[i,m-1])*m/(2*pi);
var ydot {i in 0..N-1, j in 0..m-1}
  = if (j<m-1) then (y[i,j+1]-y[i,j])*m/(2*pi) else (y[i,0]-y[i,m-1])*m/(2*pi);

var K {j in 0..m-1} = 0.5*sum {i in 0..N-1} (xdot[i,j]^2 + ydot[i,j]^2);

var P {j in 0..m-1}
  = - sum {i in 0..N-1, ii in 0..N-1: ii>i} 1/sqrt((x[i,j]-x[ii,j])^2 + (y[i,j]-y[ii,j])^2);

minimize A: (2*pi/m)*sum {j in 0..m-1} (K[j] - P[j]);
let {i in 0..N-1, k in 1..n} as[i,k] := 1*(Uniform01()-0.5);
let {i in 0..N-1, k in 1..n} ac[i,k] := 1*(Uniform01()-0.5);
let {i in 0..N-1, k in n..n} bs[i,k] := 0.01*(Uniform01()-0.5);
let {i in 0..N-1, k in n..n} bc[i,k] := 0.01*(Uniform01()-0.5);
solve;
7  Choreographies and the Ducati

The previous AMPL model was used to find many choreographies in the equimass $n$-body problem and the stable Ducati solution to the 3-body problem.
8 Limitations of the Model

- The infinite sum gets truncated to a finite sum. This amounts to adding constraints. Hence, the solution might be suboptimal. That is, the trajectory obtained might not satisfy the equations of motion.

- Masses must be positive.

- Model can’t solve 2-body problem w/ eccentricity (see next slide).
Elliptic Solutions to the 2-Body Problem

An ellipse with semimajor axis $a$, semiminor axis $b$, and having its left focus at the origin of the coordinate system is given parametrically by:

$$x(t) = f + a \cos t, \quad y(t) = b \sin t,$$

where $f = \sqrt{a^2 - b^2}$ is the distance from the focus to the center of the ellipse.

However, this is not the trajectory of a mass in the 2-body problem. Such a mass will travel faster around one focus than around the other. We need to introduce a time-change function $\theta(t)$:

$$x(t) = f + a \cos \theta(t), \quad y(t) = b \sin \theta(t).$$

This function $\theta$ must be increasing and must satisfy $\theta(0) = 0$ and $\theta(2\pi) = 2\pi$.

The optimization model can be used to find (a discretization of) $\theta(t)$ automatically by changing `param theta` to `var theta` and adding appropriate monotonicity and boundary constraints.
Using an eccentricity $e = f/a = 0.0167$ and appropriate Sun and Earth masses, we can find a periodic Hill-Type satellite trajectory in which the satellite orbits the Earth once per year.
Sensitivity Analysis

Let

\[ \xi^*(t) = \begin{bmatrix} x^*(t) \\ v^*(t) \end{bmatrix} \]

be a solution to

\[ \dot{\xi} = A(\xi) \]

where

\[
A \left( \begin{bmatrix} x(t) \\ v(t) \end{bmatrix} \right) = \begin{bmatrix} v(t) \\ a(x(t)) \end{bmatrix}
\]

and

\[
a(x) = \begin{bmatrix} a_1(x) \\ \vdots \\ a_n(x) \end{bmatrix}
\]

and

\[
a_j(x) = - \sum_{k: k \neq j} \frac{x_j - x_k}{\|x_j - x_k\|^2}, \quad j = 1, 2, \ldots, n.
\]
Consider a nearby solution $\xi(t)$:

$$
\dot{\xi}(t) = A(\xi(t)) \\
\approx A(\xi^*(t)) + A'(\xi^*(t))(\xi(t) - \xi^*(t)) \\
= \dot{\xi}^*(t) + A'(\xi^*(t))(\xi(t) - \xi^*(t)).
$$

Put $\Delta \xi = \xi - \xi^*$. Then

$$
\Delta \dot{\xi} = A'(\xi^*(t))\Delta \xi.
$$

A finite difference approximation yields

$$
\Delta \xi(t + h) = \Delta \xi(t) + hA'(\xi^*(t))\Delta \xi(t) \\
= (I + hA'(\xi^*(t))) \Delta \xi(t).
$$

Iterating around one period, we get:

$$
\Delta \xi(T) = \left( \prod_{i=0}^{n-1} (I + hA'(\xi^*(t_i))) \right) \Delta \xi(0),
$$

where $h = T/n$ and $t_i = iT/n$. 
Uninteresting Perturbations—Invariants

Perturbations associated with invariants of the physical law are unimportant in calculating $\Delta \xi(T)$:

$$
\begin{bmatrix}
\Delta x \\
\Delta v
\end{bmatrix} =
\begin{bmatrix}
e_1 \\
e_1 \\
e_1 \\
0 \\
0 \\
0
\end{bmatrix}
+ 
\begin{bmatrix}
e_2 \\
e_2 \\
e_2 \\
0 \\
0 \\
0
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
e_1 \\
e_1 \\
e_1 \\
e_1
\end{bmatrix}
\begin{bmatrix}
0 \\
0 \\
e_2 \\
e_2 \\
e_2 \\
e_2
\end{bmatrix}
\begin{bmatrix}
Rx_1 \\
Rx_2 \\
Rx_3 \\
Rv_1 \\
Rv_2 \\
Rv_3
\end{bmatrix}
\begin{bmatrix}
-3v_1 + 2x_1 \\
-3v_1 + 2x_1 \\
-3v_1 + 2x_1 \\
-3a_1 - v_1 \\
-3a_2 - v_2 \\
-3a_3 - v_3
\end{bmatrix}
$$

where $R = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$.

These directions correspond to translation(2), moving frame(2), rotation, and dilation. Dilation is explained on the next page.

All quantities are evaluated at $t = 0$.

Vector $e_i$ denotes the $i$-th unit vector in $\mathbb{R}^2$. 
Dilation

Consider spatial dilation by $\rho$ together with a temporal dilation by $\theta$:

$$y_j(t) = \rho x_j(t/\theta).$$

Given that the $x_j$'s are a solution, it is easy to check that

$$\ddot{y}_j(t) = -\frac{\rho^3}{\theta^2} \sum_{k \neq j} \frac{y_j(t) - y_k(t)}{\|y_j(t) - y_k(t)\|^2}.$$

Hence, if mass is to remain fixed, we must have that $\rho^3 = \theta^2$:

$$y_j(t) = \rho x_j(t/\rho^{3/2}) \quad \dot{y}_j(t) = \rho^{-1/2}v_j(t/\rho^{3/2}).$$

To find the perturbation direction corresponding to this dilation, we differentiate with respect to $\rho$ at $\rho = 1$:

$$\left. \frac{d}{d\rho} \begin{bmatrix} \rho x_j(t/\rho^{3/2}) \\ \rho^{-1/2}v_j(t/\rho^{3/2}) \end{bmatrix} \right|_{\rho=1} = \begin{bmatrix} -\frac{3}{2}v_j + x_j \\ -\frac{3}{2}a_j - \frac{1}{2}v_j \end{bmatrix}.$$
Projection

Put

\[
P = \begin{bmatrix}
e_1 & e_2 & 0 & 0 & Rx_1 & (-3v_1 + 2x_1)/2 \\
e_1 & e_2 & 0 & 0 & Rx_2 & (-3v_2 + 2x_2)/2 \\
e_1 & e_2 & 0 & 0 & Rx_3 & (-3v_3 + 2x_3)/2 \\
0 & 0 & e_1 & e_2 & Rv_1 & (-3a_1 - v_1)/2 \\
0 & 0 & e_1 & e_2 & Rv_2 & (-3a_2 - v_2)/2 \\
0 & 0 & e_1 & e_2 & Rv_3 & (-3a_3 - v_3)/2
\end{bmatrix}.
\]

For checking stability, we project any initial perturbation onto the null space of \( P^T \) using

\[
\Pi = I - P(P^TP)^{-1}P^T.
\]

From the fact that \( x_1 + x_2 + x_3 = 0 \) and \( v_1 + v_2 + v_3 = 0 \), it follows that all columns of \( P \) are mutually orthogonal except for the 5-th and 6-th columns. Hence, \( P^TP \) is not a purely diagonal matrix.
We now focus on:

\[ \Delta \xi(T) = \left( \prod_{i=0}^{n-1} \left( I + hA'(\xi^*(t_i)) \right) \right) \left( I - P(P^TP)^{-1}P^T \right) \Delta \xi(0), \]

Stability: all eigenvalues of

\[ \Lambda \Pi = \left( \prod_{i=0}^{n-1} \left( I + hA'(\xi^*(t_i)) \right) \right) \left( I - P(P^TP)^{-1}P^T \right) \]

must be at most one in magnitude.
### Numerical Results—Euler Stable Orbits

<table>
<thead>
<tr>
<th>Name</th>
<th>( \text{max}(\lambda_i(\Lambda)) )</th>
<th>( \text{max}(\lambda_i(\Lambda\Pi)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lagrange2</td>
<td>1.383</td>
<td>1.362</td>
</tr>
<tr>
<td>FigureEight3</td>
<td>1.228</td>
<td>4.220</td>
</tr>
<tr>
<td>Ducati3</td>
<td>1.105</td>
<td>3.885</td>
</tr>
<tr>
<td>Hill15</td>
<td>1.444</td>
<td>2.403</td>
</tr>
<tr>
<td>DoubleDouble5</td>
<td>12.298</td>
<td>12.298</td>
</tr>
<tr>
<td>DoubleDouble10</td>
<td>1.404</td>
<td>5.948</td>
</tr>
<tr>
<td>DoubleDouble20</td>
<td>1.890</td>
<td>1.890</td>
</tr>
</tbody>
</table>
## Numerical Results—Euler Unstable Orbits

<table>
<thead>
<tr>
<th>Name</th>
<th>$\text{max}(\lambda_i(\Lambda))$</th>
<th>$\text{max}(\lambda_i(\Lambda\Pi))$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lagrange3</td>
<td>81.630</td>
<td>81.630</td>
</tr>
<tr>
<td>OrthQuasiEllipse4</td>
<td>18.343</td>
<td>18.343</td>
</tr>
<tr>
<td>Rosette4</td>
<td>1.873</td>
<td>4.449</td>
</tr>
<tr>
<td>Braid4</td>
<td>727.508</td>
<td>711.811</td>
</tr>
<tr>
<td>Trefoil4</td>
<td>41228.515</td>
<td>41213.852</td>
</tr>
<tr>
<td>FigureEight4</td>
<td>221.642</td>
<td>194.095</td>
</tr>
<tr>
<td>FoldedTriLoop4</td>
<td>74758.355</td>
<td>74675.092</td>
</tr>
<tr>
<td>PlateSaucer4</td>
<td>3653.210</td>
<td>3653.210</td>
</tr>
<tr>
<td>BorderCollie4</td>
<td>188.235</td>
<td>188.052</td>
</tr>
<tr>
<td>Trefoil5</td>
<td>$1.913e+8$</td>
<td>$1.917e+8$</td>
</tr>
<tr>
<td>FigureEight5</td>
<td>2223.137</td>
<td>2223.457</td>
</tr>
</tbody>
</table>
Midpoint Integrator (using a Spring)

\[ \ddot{x} = -x \]

Given: \( x(0), v(0) \)

Compute:

\[ a(0) = -x(0) \]
\[ v(h/2) = v(0) + (h/2)a(0) \]

For \( t = h, 2h, \ldots \)

\[ a(t) = -x(t) \]
\[ v(t + h/2) = v(t - h/2) + ha(t) \]
\[ x(t + h) = x(t) + hv(t + h/2) \]

<table>
<thead>
<tr>
<th>( t )</th>
<th>( x )</th>
<th>( v )</th>
<th>( a )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>1.000</td>
<td>0.000</td>
<td>-1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.050</td>
</tr>
<tr>
<td>0.1</td>
<td>0.995</td>
<td></td>
<td>-0.995</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.150</td>
</tr>
<tr>
<td>0.2</td>
<td>0.980</td>
<td></td>
<td>-0.980</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.248</td>
</tr>
<tr>
<td>0.3</td>
<td>0.955</td>
<td></td>
<td>-0.955</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.343</td>
</tr>
<tr>
<td>0.4</td>
<td>0.921</td>
<td></td>
<td>-0.921</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-0.435</td>
</tr>
<tr>
<td>0.5</td>
<td>0.877</td>
<td></td>
<td>-0.877</td>
</tr>
</tbody>
</table>
if (integrator == MIDPOINT) {
    for (j=0; j<n; j++) {
        p[j].x += p[j].vx * dt;
        p[j].y += p[j].vy * dt;
    }
    for (j=0; j<n; j++) {
        p[j].ax = 0; p[j].ay = 0;
        for (i=0; i<n; i++) {
            if (i != j) {
                double r3 = dist3(p[i], p[j]);
                if (r3<r03) r3=r03;
                p[j].ax -= G * p[i].m * (p[j].x - p[i].x)/r3;
                p[j].ay -= G * p[i].m * (p[j].y - p[i].y)/r3;
            }
        }
    }
    for (j=0; j<n; j++) {
        p[j].vx += p[j].ax * dt;
        p[j].vy += p[j].ay * dt;
    }
}