AE4879 Mission Geometry and Orbit Design

Assignment 12: Exponential Sinusoids

EXPO-4

Simon Billemont

March 22, 2011

Assignment 12: Exponential Sinusoids

This assignment deals with low thrust transfer orbits. This is done my implementing Exponential Sinusoids, and fitting these to the Lambert Problem (Lambert's Orbital Boundary-Value Problem). The solution is then and transfer orbit, where the thrust in in the same direction as the objects velocity vector.

1. Exponential Sinusoids

Exponential Sinusoids are a class of functions that represent a spiral in the r, θ plane. The functions are determined by eq 1 and a set of costants; $k_0, k_1, k_2, q, \varphi$.

$$r = k_0 \cdot exp(q \cdot \theta + k_1 \cdot \sin(k_2 \cdot \theta + \phi)) \tag{1}$$

From now on, we will be working the the pure exponential sinusoids, meaning that $q = 0$. Furthermore tangential thrusting is assumed. This simplifies the equations needed to solve the targeting problem. What is given by the Lambert problem, is the start, end positions and the time of flight between the two points. From this, the total angle between the start and end point can be determined.

$$\psi = \frac{\vec{r_1} \cdot \vec{r_2}}{|\vec{r_1}||\vec{r_2}|} + N \cdot 2\pi \tag{2}$$

Where N is the amount of revolutions wanted to reach the target position. What is also needed is the standard gravitational parameter of the central body μ. For out solar system this is for the sun: $\mu_{sun} = 1.32712 \cdot 10^{20} m^3/s^2$. Then also an assumption on k_2 is needed. For multiple revolutions one can use eq 3 to guess k_2 (otherwise $k_2 = 1/12$ is a good start guess).

$$k_2 = \frac{1}{2N} \tag{3}$$

From this set of knowns and assumptions, a set of general solutions can be found. These solutions are dependent on a parameter γthat determines the travel time from $\vec{r_1}$ to $\vec{r_2}$. Using this γ, one can find an exponential sinusoid, with the parameter given in eq 4 - 7 (see [1] slide 51-52).

$$sign(k_1) = sign\left[ln\left(\frac{|\vec{r_1}|}{|\vec{r_2}|}\right) + \frac{\tan\gamma \sin(k_2\psi)}{k_2}\right] \tag{4}$$

$$k_1^2 = \left[\frac{ln\left(\frac{|\vec{r_1}|}{|\vec{r_2}|}\right) + \frac{\tan\gamma\sin(k_2\psi)}{k_2}}{1 - \cos(k_2\psi)}\right]^2 + \frac{\tan^2\gamma}{{k_2}^2} \tag{5}$$

$$\phi = \cos^{-1}\frac{\tan\gamma}{k_1 k_2} \tag{6}$$

$$k_0 = \frac{|\vec{r_1}|}{exp(k_1 \sin\phi)} \tag{7}$$

Revolutions	γ_{opt}	k_0	k_1	k_2	ϕ
$N = 0$	0.2402	$1.91 \cdot 10^{12}$	$-3,89$	$0,08$	$2,43$
$N = 1$	$-1,0864$	$1.61 \cdot 10^{41}$	$-72,81$	$0,08$	$1,25$
$N = 2$	$-1,0513$	$1.25 \cdot 10^{25}$	$-38,32$	$0,08$	$0,99$
$N = 3$	$-1,0426$	$1.46 \cdot 10^{19}$	$-27,59$	$0,08$	$0,73$
$N = 4$	$-1,0394$	$4.54 \cdot 10^{15}$	$-22,88$	$0,08$	$0,47$
$N = 5$	$-1,0378$	$1.06 \cdot 10^{13}$	$-20,78$	$0,08$	$0,21$

Table 1: Optimal results for $\vec{r_1} =< 1,0,0 >$ AU; $\vec{r_2} =< 0,1.5,0 >$ AU; $\Delta T = 0.35$ year

Then these parameters translate into the time of flight required ([1] slide 32) :

$$tof = \int_0^{\psi} \sqrt{\frac{r^3 \left(\tan^2 \gamma + k_1 {k_2}^2 s + 1\right)}{\mu}} d(\theta) \tag{8}$$

$$\text{with } s = \sin(k_2 \psi + \varphi) \tag{9}$$

Then a using a root finding technique, one can find the root of the tof - dT, yielding the wanted transfer orbit. To limit the search space for γ, it can be used that γ only exists for a small domain. This domain is determined by (see [1] slide 53):

$$D = [\gamma_{min}, \gamma_{max}] \tag{10}$$

$$\gamma_{min} = \tan^{-1}\left[\frac{k_2}{2}\left(-ln\left(\frac{|\vec{r_1}|}{|\vec{r_2}|}\right)\cot\left(\frac{k_2\psi}{2}\right) - \sqrt{\delta}\right)\right] \tag{11}$$

$$\gamma_{max} = \tan^{-1}\left[\frac{k_2}{2}\left(-ln\left(\frac{|\vec{r_1}|}{|\vec{r_2}|}\right)\cot\left(\frac{k_2\psi}{2}\right) + \sqrt{\delta}\right)\right] \tag{12}$$

$$\text{with } \delta = \frac{2(1-\cos(k_2\psi))}{{k_2}^4} - ln^2\frac{|\vec{r_1}|}{|\vec{r_2}|} \tag{13}$$

The results of this process are then all the exponential sinusoid parameters that define a unique orbit from $\vec{r_1}$ to $\vec{r_2}$ in the prescribed time of dT. Furthermore, one can use the exponential sinusoid parameter to find the required ΔV needed for the maneuver.

2. Simple transfer orbit example

- **Reproduce the time-of-flight curve S1/12 [1, 1.5, p/2, 0] as on slide 57 [1] .**
- **Determine the transfer options for N = 0, 1, 2 that satisfy the boundary conditions r1 = 1 AU, r2 = 1.5 AU, TOF = 0.35 yr and k2 = 1/12.**

As an simple exercise, the transfer orbit from $\vec{r_1} =< 1,0,0 >$ AU to $\vec{r_2} =< 0,1.5,0 >$ AU is considered. When we plot the solution sets, fig 1 is created. It shows the time of flight vs the gamma parameter. This is done for multiple revolutions (N) around the center body (the sun in this case).

Next, the optimal trajectory for a given travel time of $\Delta T = 0.35$ years is computed. This again for multiple revolutions. The results are summarized in table 1. Furthermore,

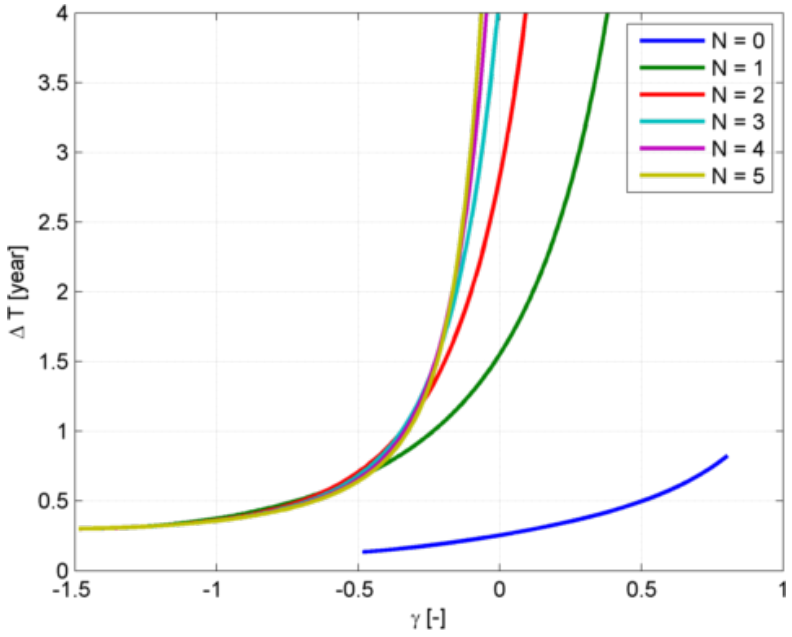

Figure 1:

for $N = [1; 2]$, a plot of the transfer orbit has been made (in the orbit plane). This is shown in fig 2.

References

[1] R. Noomen, *AE4-879 Exponential Sinusoids V3.3*, TUDelft Lecture Slides, 2010.

[2] D. Austin and W. Dickinson. (2009) Spherical easel. A spherical drawing program. [Online]. Available: http://merganser.math.gvsu.edu/easel/

[3] MathWorks. (2010a) Matlab 7.11. Natick, MA.

[4] J. R. Wertz, *Orbit & Constellation Design & Management*, second printing ed. El Segundo, California: Microcosm Press, 2009.

[5] E. W. Weisstein. (2010) Trigonometry. MathWorld. [Online]. Available: http://mathworld.wolfram.com/Trigonometry.html

[6] S. Eddins. (2010, Nov) Matlab xunit test framework. MATLAB Central File Exchange. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/22846

Additional information

Estimated work time:

$\sim$ 3h Studying theory + $\sim$ 5h making assignment + $\sim$ 3h writing report = $\sim$ 11h

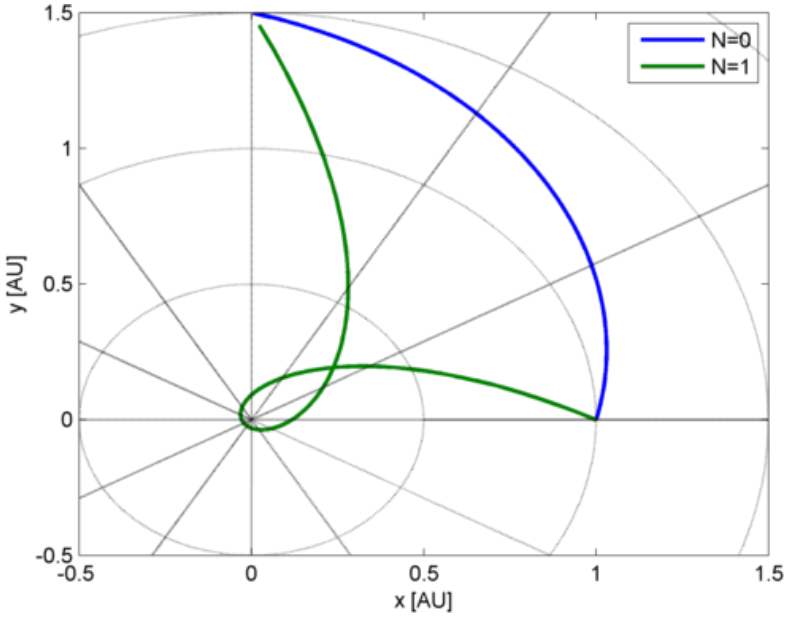

Figure 2:

Made by

Simon Billemont

Stud Nr: 1387855

s.billemont@student.tudelft.nl

License and notices

Build using the be.angelcorp.libs libraries:

- be.angelcorp.libs.math (Git version; 223d616ed6634c2ed5d8787eb895e1472d7b48aa)
- be.angelcorp.libs.celest (Git version; ee6b173f6464966bae0cc03c42be4e86f7ad0bde)

Version history

Version 1: Initial document

A. Source code

The code written to implement the three described optimizers was written in JAVA SE 6, with a MATLAB 7.11 (2010b)[3] backend over jmi (JAVA MATLAB interface). The relevant files are listed below:

Listing 1: TestExposin: caption

```
/**
 * Copyright (C) 2011 Simon Billemont <aodtorusan@gmail.com>
 *
 * Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported
 * (CC BY-NC 3.0) (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
 *        http://creativecommons.org/licenses/by-nc/3.0/
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package be.angelcorp.tudelft.ae4879.exposins;

import be.angelcorp.libs.celest.maneuvers.targeters.exposin.ExpoSin;
import be.angelcorp.libs.celest.maneuvers.targeters.exposin.ExpoSinSolutionSet;
import be.angelcorp.libs.celest.maneuvers.targeters.exposin.ExpoSinTrajectory;
import be.angelcorp.libs.celest.stateVector.CartesianElements;
import be.angelcorp.libs.math.functions.ExponentialSinusoid;
import be.angelcorp.libs.math.linear.Vector3D;
import be.angelcorp.libs.math.matlab.MPlot2;
import be.angelcorp.libs.util.mlink.MLink;
import be.angelcorp.libs.util.physics.Length;
import be.angelcorp.libs.util.physics.Time;

/**
 *
 * Plots the time of flight for a set of exposin trajectories for a different set of revolutions.
 * <p>
 * Based on: TUDelft AE4-879 lecture slides on exposin by Ron Noomen.
 * </p>
 *
 * @author Simon Billemont, aodtorusan@gmail.com
 *
 */
public class TestExposin {

    public static void main(String[] args) throws Exception {
        MLink.get(); // Start initializing matlab
        MPlot2 graph = new MPlot2(); // Container for the gamma vs dt plot that will be made
        for (int n = 0; n <= 5; n++) {
            /* Create a new exposin problem */
            ExpoSin expo = new ExpoSin(
                    new CartesianElements( // Start state vector
                            new Vector3D(Length.convert(1, Length.AU), 0, 0), Vector3D.ZERO),
                    new CartesianElements( // End state vector R: <0, 5 AU, 0> V: <0, 0, 0>
                            new Vector3D(0, Length.convert(1.5, Length.AU), 0), Vector3D.ZERO),
                    Time.convert(0.35, Time.year)); // TOF, Not required
            expo.assumeK2(1. / 12); /* Sets exposin k2 */
            expo.setN(n); /* Sets the revolutions we should make */

            /* Plot all of the the solutions in the solution set */
            /* meaning plot "time of flight" in function of gamma */
            ExpoSinSolutionSet solutions = expo.getSolutionSet();
            graph.addFunction(
                    solutions.multiply(Time.convert(1, Time.second, Time.year)), // plot normalized tof
                    solutions.getDomain() // Plot the entire gamma domain
            );

            /* Get the optimal exposin trajectory (where tf is the wanted value) */
            ExpoSinTrajectory trajectory = expo.getTrajectory();
            ExponentialSinusoid f = trajectory.getExposin();

            /* Output exposin parameter details */
            System.out.println(String.format("\t\tN=%d & %.2e & %.2f & %.2f & %.2f \\\\ \\hline",
                    n, f.getK0(), f.getK1(), f.getK2(), f.getPhi()));
        }

        /* Make the matlab plot */
        graph.makePlot().get();
        /* Make the plot fancyer and add a legend */
        MLink.get().eval("set(gca, 'xlim',[-1.5,1], 'ylim',[0,4]);grid on;" +
                "legend('N = 0', 'N = 1', 'N = 2', 'N = 3', 'N = 4', 'N = 5')");
    }

}
```

Listing 2: ExpoSin: caption

```
/**
 * Copyright (C) 2011 Simon Billemont <aodtorusan@gmail.com>
 *
 * Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported
 * (CC BY-NC 3.0) (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
 *        http://creativecommons.org/licenses/by-nc/3.0/
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package be.angelcorp.libs.celest.maneuvers.targeters.exposin;

import org.apache.commons.math.MathException;

import be.angelcorp.libs.celest.body.CelestialBody;
import be.angelcorp.libs.celest.constants.SolarConstants;
import be.angelcorp.libs.celest.maneuvers.targeters.TPBVP;
import be.angelcorp.libs.celest.stateVector.StateVector;
import be.angelcorp.libs.math.linear.Vector3D;

import com.lyndir.lhunath.lib.system.logging.Logger;

/**
 *
 * Exponential sinusoid solution (low thrust solution) to the Lambert problem ( a two-point boundary
 * value problem, TPBVP). It assumes a constant low thrust by the spacecraft in the direction of the
 * instantaneous velocity vector.
 *
 * @author simon
 *
 */
public class ExpoSin extends TPBVP {

    private static final Logger logger      = Logger.get(ExpoSin.class);

    /**
     * Standard gravitational parameter of the center body (body with the strongest influence on the
     * satellite). Default is the standard gravitational parameter of the sun.
     * <p>
     * <b>Unit: [m<sup>3</sup>/s<sup>2</sup>]</b>
     * </p>
     */
    private CelestialBody       center      = SolarConstants.body;
    /**
     * Amount of rotations to perform around the center body in order to arrive at r2.
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    private int                 N           = 0;
    /**
     * Exposin parameter k2 (Winding parameter). This must be assumed. Set using:
     * <ul>
     * <li>{@link ExpoSin#assumeK2()} Set the k2 using this.N as guide</li>
     * <li>{@link ExpoSin#assumeK2(int)} Set the k2 using a given amount of revolutions as guide</li>
     * <li>{@link ExpoSin#assumeK2(double)} Set the k2 directly</li>
     * </ul>
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    private double              assumeK2    = 1. / 12;

    /**
     * Exposin problem with the given boundary conditions
     *
     * @param r1
     *            StateVector that defines the starting position [m]
     * @param r2
     *            StateVector that defines the wanted end position [m]
     * @param dT
     *            Time between the start position and end position [s]
     */
    public ExpoSin(StateVector r1, StateVector r2, double dT) {
        super(r1, r2, dT);
    }

    /**
     * Set the k2 parameter (winding parameter) used in the exposin solution directly.
     *
     * @param assumeK2
     *            New value for k2 [-]
     */
    public void assumeK2(double assumeK2) {
        this.assumeK2 = assumeK2;
    }
```

```
    /**
     * Set the k2 parameter (winding parameter) used in the exposin solution by inferring it from the set
     * amount of revolutions around the center body.
     */
    public void assumeK2FromN() {
        assumeK2FromN(N);
    }

    /**
     * Set the k2 parameter (winding parameter) used in the exposin solution by inferring it from the
     * given amount of revolutions around the center body.
     *
     * @param N
     *            Revolutions to assume [N]
     */
    public void assumeK2FromN(int N) {
        this.assumeK2 = 1. / (2 * N);
    }

    /**
     * Get all the possible exposin solutions for reaching r2 from r1 (without time constraignt). It is a
     * function of a generic parameter gamma.
     *
     * @return All possible exposin solutions
     */
    public ExpoSinSolutionSet getSolutionSet() {
        Vector3D r1vec = this.r1.toCartesianElements().R;
        Vector3D r2vec = this.r2.toCartesianElements().R;
        double r1 = r1vec.getNorm();
        double r2 = r2vec.getNorm();

        double psi = Math.acos(r1vec.dot(r2vec) / (r1 * r2));
        double theta = psi + 2 * Math.PI * N;

        return new ExpoSinSolutionSet(r1, r2, dT, assumeK2, theta, center.getMu());
    }

    @Override
    public ExpoSinTrajectory getTrajectory() throws MathException {
        ExpoSinSolutionSet solutions = getSolutionSet();
        double gammaOptimal = solutions.getOptimalSolution();
        logger.dbg("Gamma optimal: %f", gammaOptimal);

        return new ExpoSinTrajectory(solutions.getExpoSin(gammaOptimal), solutions.getThetaMax(),
                gammaOptimal, center);
    }

    /**
     * Set the center celesital body
     */
    public void setCenter(CelestialBody center) {
        this.center = center;
    }

    /**
     * Set the amount of revolutions that are to be made around the center body (default:0)
     *
     * @param n
     *            Nr of revolutions
     */
    public void setN(int n) {
        N = n;
    }
}
```

Listing 3: ExpoSinSolutionSet: caption

```
/**
 * Copyright (C) 2011 Simon Billemont <aodtorusan@gmail.com>
 *
 * Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported
 * (CC BY-NC 3.0) (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
 *        http://creativecommons.org/licenses/by-nc/3.0/
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package be.angelcorp.libs.celest.maneuvers.targeters.exposin;

import org.apache.commons.math.FunctionEvaluationException;
import org.apache.commons.math.MathException;
import org.apache.commons.math.analysis.ComposableFunction;
import org.apache.commons.math.analysis.solvers.UnivariateRealSolverFactoryImpl;
import org.apache.commons.math.optimization.OptimizationException;

import be.angelcorp.libs.math.functions.ExponentialSinusoid;
import be.angelcorp.libs.math.functions.domain.Domain;
```

```java
/**
 * Creates a solution set for a given exposin problem. It contains all of the possible trajectories from
 * r1 to r2, without the time constraint. For the optimal solution, it solves the time of flight to match
 * the set dT.
 *
 * @author Simon Billemont
 *
 */
public class ExpoSinSolutionSet extends ComposableFunction {

    /**
     * Computes the exposin K0 parameter
     */
    private static double getK0(double r1, double k1, double phi) {
        double k0 = r1 * (Math.exp(-k1 * Math.sin(phi)));
        return k0;
    }

    /**
     * Computes the exposin K1 parameter
     */
    private static double getK1(double log, double tan_gamma, double k2, double theta, double gamma) {
        double signK1 = (log + (tan_gamma / k2) * Math.sin(k2 * theta))
                / (1 - Math.cos(k2 * theta));
        double k1 = Math.signum(signK1)
                * Math.sqrt(Math.pow(signK1, 2) + (Math.pow(Math.tan(gamma), 2)) / (k2 * k2));
        return k1;
    }

    /**
     * Computes the exposin phi parameter
     */
    private static double getPhi(double tan_gamma, double k1, double k2) {
        double phi = Math.acos(tan_gamma / (k1 * k2));
        return phi;

    }

    /**
     * Domain (of gamma) over which this function has valid solutions.
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    private Domain  domain;

    /**
     * Start radius of the spacecraft
     * <p>
     * <b>Unit: [m]</b>
     * </p>
     */
    private double  r1;
    /**
     * Wanted travel time between r1 and r2
     * <p>
     * <b>Unit: [s]</b>
     * </p>
     */
    private double  dT;
    /**
     * Cached value of log(r1 / r2)
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    private double  log;
    /**
     * Cached value of the exposin parameter k2
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    private double  k2;
    /**
     * Total traveled angle between r1 and r2
     * <p>
     * <b>Unit: [rad]</b>
     * </p>
     */
    private double  theta;
    /**
     * Standard gravitational parameter of the center body.
     * <p>
     * <b>Unit: [m<sup>3</sup>/s<sup>2</sup>]</b>
     * </p>
     */
    private double  mu_center;

    public ExpoSinSolutionSet(double r1, double r2, double dT, double k2, double theta, double mu_center) {
        this.r1 = r1; // Store stuff
        this.dT = dT;
        this.k2 = k2;
        this.theta = theta;
```

```
        this.mu_center = mu_center;
        log = Math.log(r1 / r2); // cache value as its commenly needed

        /* Compute the domain of gamma [gamma_min, gamma_max] */
        double delta = (2 * (1 - Math.cos(k2 * theta))) / Math.pow(k2, 4) - (log * log);
        double gamma_min = Math.atan((k2 / 2)
                * (-log * (1 / Math.tan(k2 * theta / 2)) - Math.sqrt(delta)));
        double gamma_max = Math.atan((k2 / 2)
                * (-log * (1 / Math.tan(k2 * theta / 2)) + Math.sqrt(delta)));

        domain = new Domain(gamma_min, gamma_max);
    }

    /**
     * Domain of gamma over which this function yields valid results
     */
    public Domain getDomain() {
        return domain;
    }

    /**
     * Get the exponential sinusiod trajectory that is linked to a given gamma value.
     */
    public ExponentialSinusoid getExpoSin(double gamma) {
        double tan_gamma = Math.tan(gamma);
        /* Compute the exposin parameters */
        double k1 = getK1(log, tan_gamma, k2, theta, gamma);
        double phi = getPhi(tan_gamma, k1, k2);
        double k0 = getK0(r1, k1, phi);
        return new ExponentialSinusoid(k0, k1, k2, 0, phi);
    }

    /**
     * Compute the function of gamma for which the travel time is the set dT.
     *
     * @return The optimal gamma value
     * @throws OptimizationException
     */
    public double getOptimalSolution() throws OptimizationException {
        double gamma = Double.NaN;
        /* Try different solvers to find a root of the f(gamma) = tof - dT */
        try {
            gamma = new UnivariateRealSolverFactoryImpl().newBrentSolver().
                    solve(this.add(-dT), domain.lowerBound, domain.upperBound, 0);
        } catch (Exception e) {
        }
        if (gamma == Double.NaN)
            /* We could not solve the problem to tell the user */
            throw new OptimizationException(new MathException("Could not find root solution"));
        return gamma; // return the optimam gamma value
    }

    public double getThetaMax() {
        return theta;
    }

    /**
     * Set the wanted travel time between r1 and r2
     *
     * @param dT
     *            Wanted travel time [s]
     */
    public void setdT(double dT) {
        this.dT = dT;
    }

    /**
     * {@inheritDoc} Find the time of flight for a given value of gamma.
     */
    @Override
    public double value(double gamma) throws FunctionEvaluationException {
        double tan_gamma = Math.tan(gamma);
        /* Compute the exposin parameters */
        double k1 = getK1(log, tan_gamma, k2, theta, gamma);
        double phi = getPhi(tan_gamma, k1, k2);
        double k0 = getK0(r1, k1, phi);
        /* Formulate the tof equation */
        ExpoSinTOF tof = new ExpoSinTOF(k0, k1, k2, phi, theta, mu_center);
        double y = tof.value(gamma); // Return the actual tof value
        return y;
    }

}
```

Listing 4: ExpoSinTOF: caption

```
/**
 * Copyright (C) 2011 Simon Billemont <aodtorusan@gmail.com>
 *
 * Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported
 * (CC BY-NC 3.0) (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
```

```
 *        http://creativecommons.org/licenses/by-nc/3.0/
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package be.angelcorp.libs.celest.maneuvers.targeters.exposin;

import org.apache.commons.math.FunctionEvaluationException;
import org.apache.commons.math.analysis.ComposableFunction;
import org.apache.commons.math.analysis.UnivariateRealFunction;
import org.apache.commons.math.analysis.integration.LegendreGaussIntegrator;
import org.apache.commons.math.analysis.integration.UnivariateRealIntegrator;

import be.angelcorp.libs.math.functions.ExponentialSinusoid;

public class ExpoSinTOF extends ComposableFunction {

    /**
     * Exposin k1 parameter
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    protected final double              k1;
    /**
     * Exposin k2 parameter
     * <p>
     * <b>Unit: [-]</b>
     * </p>
     */
    protected final double              k2;
    /**
     * Exposin phi parameter
     * <p>
     * <b>Unit: [rad]</b>
     * </p>
     */
    protected final double              phi;
    /**
     * End angle for theta (total rotation that is executed)
     * <p>
     * <b>Unit: [rad]</b>
     * </p>
     */
    protected final double              theta_f;
    /**
     * Standard gravitational parameter of the center body.
     * <p>
     * <b>Unit: [m<sup>3</sup>/s<sup>2</sup>]</b>
     * </p>
     */
    protected final double              mu;

    /**
     * Basic exponential sinusoid, for which the time of flight can be computed (in function of gamma).
     * Note this only hold valif information for the same gamma where the exposin parameters where
     * constructed with.
     * <p>
     * <b>Unit: tof(&gamma;) = [s]</b>
     * </p>
     */
    protected ExponentialSinusoid       expo;
    /**
     * Function integrator (used to integrate from theta is 0 to theta_f.
     */
    private UnivariateRealIntegrator    integrator;

    /**
     * Create the time of flight function for a given set of exposin parameters
     *
     * @param k0
     *            Exposin param
     * @param k1
     *            Exposin param
     * @param k2
     *            Exposin param
     * @param phi
     *            Exposin param
     * @param theta_final
     *            Total rotation angle of the exposin
     * @param mu
     *            Standard gravitation parameter of the center body
     */
    public ExpoSinTOF(double k0, double k1, double k2, double phi, double theta_final, double mu) {
        this.k1 = k1;
        this.k2 = k2;
        this.phi = phi;
        this.theta_f = theta_final;
        this.mu = mu;
        /* Make an exposin of the given parameters */
        expo = new ExponentialSinusoid(k0, k1, k2, 0, phi);
```

```
        /* Make a new function intgrator */
        integrator = new LegendreGaussIntegrator(5, 100);
    }

    /**
     * {@inheritDoc}
     * <p>
     * Note you should use the same gamma as was used to make the exposin parameters
     * </p>
     */
    @Override
    public double value(final double gamma) throws FunctionEvaluationException {
        /* The d(theta)/dt equation */
        UnivariateRealFunction theta_dot = new UnivariateRealFunction() {
            @Override
            public double value(double theta) throws FunctionEvaluationException {
                /* d(theta)/dt = sqrt(r^3 (tan^2(g) + k1 k2^2 s +s)/ mu) */
                double r = expo.value(theta);
                double s = Math.sin(k2 * theta + phi);
                double theta_dot = Math.sqrt(
                        (Math.pow(r, 3) / mu)
                                * (Math.pow(Math.tan(gamma), 2) + k1 * k2 * k2 * s + 1));
                return theta_dot;
            }
        };
        try {
            return integrator.integrate(theta_dot, 0, theta_f);
        } catch (Exception e) {
            return Double.POSITIVE_INFINITY; /* wrap the error and deal with it later */
        }
    }
}
```

Listing 5: ExpoSinTrajectory: caption

```
/**
 * Copyright (C) 2011 Simon Billemont <aodtorusan@gmail.com>
 *
 * Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported
 * (CC BY-NC 3.0) (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
 *        http://creativecommons.org/licenses/by-nc/3.0/
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package be.angelcorp.libs.celest.maneuvers.targeters.exposin;

import org.apache.commons.math.ConvergenceException;
import org.apache.commons.math.FunctionEvaluationException;
import org.apache.commons.math.analysis.ComposableFunction;
import org.apache.commons.math.analysis.UnivariateRealFunction;
import org.apache.commons.math.analysis.integration.LegendreGaussIntegrator;

import be.angelcorp.libs.celest.body.CelestialBody;
import be.angelcorp.libs.celest.stateVector.CartesianElements;
import be.angelcorp.libs.celest.stateVector.StateVector;
import be.angelcorp.libs.celest.trajectory.Trajectory;
import be.angelcorp.libs.math.functions.ExponentialSinusoid;
import be.angelcorp.libs.math.linear.Vector3D;

public class ExpoSinTrajectory extends Trajectory {

    private ExponentialSinusoid exposin;
    private double              thetaMax;
    private double              gamma;
    private CelestialBody       center;

    public ExpoSinTrajectory(ExponentialSinusoid exposin, double thetaMax, double gamma,
            CelestialBody center) {
        this.exposin = exposin;
        this.thetaMax = thetaMax;
        this.gamma = gamma;
        this.center = center;
    }

    @Override
    public StateVector evaluate(double t) throws FunctionEvaluationException {
        UnivariateRealFunction thetaDot = new ComposableFunction() {
            @Override
            public double value(double theta) throws FunctionEvaluationException {
                double r = exposin.value(theta);
                double s = Math.sin(exposin.getK2() * theta + exposin.getPhi());
                double thetaDot = (center.getMu() / Math.pow(r, 3))
                        * (1 / (Math.pow(Math.tan(gamma), 2) + exposin.getK1() * exposin.getK2()
                                * exposin.getK2() * s + 1));
                thetaDot = Math.sqrt(thetaDot);
                return thetaDot;
            }
```

```
        };
        double theta;
        try {
            theta = new LegendreGaussIntegrator(5, 200).integrate(thetaDot, 0, t);
        } catch (ConvergenceException e) {
            throw new FunctionEvaluationException(e, t);
        }
        double r = exposin.value(theta);
        return new CartesianElements(
                new Vector3D(r * Math.cos(theta), r * Math.sin(theta), 0), Vector3D.ZERO);
    }

    public ExponentialSinusoid getExposin() {
        return exposin;
    }

    public double getGamma() {
        return gamma;
    }

    public double getThetaMax() {
        return thetaMax;
    }

}
```

Listing 6: TPBVP: caption

```
/**
 * Copyright (C) 2011 Simon Billemont <aodtorusan@gmail.com>
 *
 * Licensed under the Creative Commons Attribution-NonCommercial 3.0 Unported
 * (CC BY-NC 3.0) (the "License"); you may not use this file except in
 * compliance with the License. You may obtain a copy of the License at
 *
 *        http://creativecommons.org/licenses/by-nc/3.0/
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package be.angelcorp.libs.celest.maneuvers.targeters;

import org.apache.commons.math.MathException;

import be.angelcorp.libs.celest.stateVector.StateVector;
import be.angelcorp.libs.celest.trajectory.Trajectory;

/**
 *
 * Base class for the Lambert problem, meaning finding the transfer arc for a two-point boundary value
 * problem (TPBVP). Here the start and end position are known, as well as the time that the satellite is
 * to travel between the two points.
 *
 * @author Simon Billemont
 *
 */
public abstract class TPBVP {

    /**
     * Travel time between r1 and r2
     * <p>
     * <b>Unit: [s]</b>
     * </p>
     */
    protected double        dT;
    /**
     * Start position
     * <p>
     * <b>If converted to Cartesian coordinates; unit: [m]</b>
     * </p>
     */
    protected StateVector   r1;
    /**
     * Targeted end position
     * <p>
     * <b>If converted to Cartesian coordinates; unit: [m]</b>
     * </p>
     */
    protected StateVector   r2;

    /**
     * Construct a TPBVP problem with known, start/end points and the travel time between r1 and r2.
     *
     * @param r1
     *            Start position
     * @param r2
     *            End position
     * @param dT
     *            Travel time
     */
```

```
    public TPBVP(StateVector r1, StateVector r2, double dT) {
67      this.r1 = r1;
        this.r2 = r2;
        this.dT = dT;
    }

72  /**
     * Find the optimal trajectory according to this targeted between r1->r2 for the given travel time.
     *
     * @return An optimal trajectory
     * @throws MathException
77   */
    public abstract Trajectory getTrajectory() throws MathException;

}
```