A JAVA SOFTWARE FOR AIRCRAFT FLIGHT DYNAMICS CALCULATION

APPENDIX

RELATORE

CH.MO PROF. ING.
AGOSTINO DE MARCO

CANDIDATO

GIUSEPPE SCARLATELLA

N35/678

ANNO ACCADEMICO 2013/2014

STABILITY DERIVATIVES CALCULATOR

```java
package newproj;

public class StabilityDerivativesCalc {

    public enum Propulsion { CONSTANT_TRUST, CONSTANT_POWER, CONSTANT_MASS_FLOW, RAMJET }

    public static double calcDynamicPressure(double rho0, double u0) {

        return 0.5*rho0*Math.pow(u0,2);

    }

    public static double calcMassVehicleParameter(double rho0, double surf, double mass,
            double cbar) {

        return 2*mass/(rho0*surf*cbar);

    }


    //////////////////////          LONGITUDINAL DYNAMICS          //////////////////////

    // Longitudinal Dimensional Stability Derivatives

    /**
     *   calculates the dimensional stability derivative of force component X with respect
     to "u",
     *   divided by the mass, for Constant Thrust (appropriate for jet aircraft or
     *   for unpowered flight)
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cd0 drag coefficient at null incidence (Cdº) of the aircraft
     * @param m0 Mach number
     * @param cdM0 drag coefficient with respect to Mach (CdMº) of the aircraft
     * @return XªuCT dimensional derivative [s^(-1)]
     */
    public static double calcX_u_CT (double rho0, double surf, double mass, double u0,
    double q0,
            double cd0, double m0, double cdM0) {

        return -q0*surf*(2*cd0 + m0*cdM0)/(mass*u0);

    }

    /**
     *   calculates the dimensional stability derivative of force component X with respect
     to "u",
     *   divided by the mass, for Constant Power (appropriate for propeller aircraft
     *   with automatic pitch control and constant-speed propeller)
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cd0 drag coefficient at null incidence (Cdº) of the aircraft
     * @param m0 Mach number
     * @param cdM0 drag coefficient with respect to Mach (CdMº) of the aircraft
     * @param cl0 lift coefficient at null incidence (Clº) of the aircraft
     * @param gamma0 ramp angle
     * @return XªuCP dimensional derivative [s^(-1)]
     */
    public static double calcX_u_CP (double rho0, double surf, double mass, double u0,
    double q0,
            double cd0, double m0, double cdM0, double cl0, double gamma0) {
```

```
        double gamma0_rad = Math.toRadians(gamma0); // angolo di salita (in rad)
        return -q0*surf*(3*cd0+cl0*Math.tan(gamma0_rad)+m0*cdM0)/(mass*u0);

    }

    /**
     *   calculates the dimensional stability derivative of force component X with respect
     to "w",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cl0 lift coefficient at null incidence (Clº) of the aircraft
     * @param cdAlpha0 linear drag gradient (CdAlphaº) of the aircraft
     * @return Xªw dimensional derivative [s^(-1)]
     */
    public static double calcX_w (double rho0, double surf, double mass, double u0, double q0,
            double cl0, double cdAlpha0) {


        return q0*surf*(cl0-cdAlpha0)/(mass*u0);

    }

    /**
     *   calculates the dimensional stability derivative of force component Z with respect
     to "u",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param m0 Mach number
     * @param cl0 lift coefficient at null incidence (Clº) of the aircraft
     * @param clM0 lift coefficient with respect to Mach (ClMº) of the aircraft
     * @return Zªu dimensional derivative [s^(-1)]
     */
    public static double calcZ_u (double rho0, double surf, double mass, double u0, double
    q0, double m0,
            double cl0, double clM0) {


        return -q0*surf*(2*cl0 + Math.pow(m0,2)*clM0/(1-Math.pow(m0,2)) )/(mass*u0);

    }

    /**
     *   calculates the dimensional stability derivative of force component Z with respect
     to "w",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param m0 Mach number
     * @param cd0 drag coefficient at null incidence (Cdº) of the aircraft
     * @param clAlpha0 linear lift gradient (ClAlphaº) of the aircraft
     * @return Zªw dimensional derivative [s^(-1)]
     */
    public static double calcZ_w (double rho0, double surf, double mass, double u0, double
    q0, double m0,
            double cd0, double clAlpha0) {
```

```java
        return -q0*surf*(cd0 + clAlpha0 )/(mass*u0);

    }

    /**
     *   calculates the dimensional stability derivative of force component Z with respect
     to "w_dot",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param cbar mean aerodynamic chord
     * @param clAlpha_dot0 linear lift gradient time derivative (ClAlpha_dotº) of the aircraft
     * @return Zªw_dot adimensional derivative
     */
    public static double calcZ_w_dot (double rho0, double surf, double mass, double cbar,
            double clAlpha_dot0) {

        return -clAlpha_dot0/(2*calcMassVehicleParameter(rho0, surf, mass, cbar));

    }

    /**
     *   calculates the dimensional stability derivative of force component Z with respect
     to "q",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cbar mean aerodynamic chord
     * @param clQ0 lift coefficient with respect to q (ClQº) of the aircraft
     * @return Zªq dimensional derivative [s^(-1)]
     */
    public static double calcZ_q (double rho0, double surf, double mass, double u0, double
    q0, double cbar,
            double clQ0) {

        return -u0*clQ0/(2*calcMassVehicleParameter(rho0, surf,mass, cbar));

    }

    /**
     *   calculates the dimensional stability derivative of pitching moment M with respect
     to "u",
     *   divided by the longitudinal moment of inertia Iyy
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cbar mean aerodynamic chord
     * @param m0 Mach number
     * @param iYY longitudinal moment of inertia  (IYY)
     * @param cM_m0 pitching moment coefficient with respect to Mach number
     * @return cMªu dimensional derivative [m^(-1) * s^(-1)]
     */
    public static double calcM_u (double rho0, double surf, double mass, double u0, double
    q0, double cbar,
            double m0, double iYY, double cM_m0) {


        return q0*surf*cbar*m0*cM_m0/(iYY*u0);
```

```
    }

    /**
     *   calculates the dimensional stability derivative of pitching moment M with respect
     to "w",
     *   divided by the longitudinal moment of inertia Iyy
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cbar mean aerodynamic chord
     * @param iYY longitudinal moment of inertia  (IYY)
     * @param cMAlpha0 pitching moment coefficient with respect to Alpha (CmAlpha°) of the
     aircraft
     * @return calcMªw dimensional derivative [m^(-1) * s^(-1)]
     */
    public static double calcM_w (double rho0, double surf, double mass, double u0, double
    q0, double cbar,
            double iYY, double cMAlpha0) {


        return q0*surf*cbar*cMAlpha0/(iYY*u0);

    }

    /**
     *   calculates the dimensional stability derivative of pitching moment M with respect
     to "w_dot",
     *   divided by the longitudinal moment of inertia Iyy
     * @param rho0 air density
     * @param mass total mass
     * @param surf wing area
     * @param cbar mean aerodynamic chord
     * @param iYY longitudinal moment of inertia  (IYY)
     * @param cMAlpha_dot0 pitching moment coefficient with respect to Alpha (CmAlpha°) of
     the aircraft_dot
     * @return calcMªw_dot dimensional derivative [m^(-1)]
     */
    public static double calcM_w_dot (double rho0, double surf, double cbar,
            double iYY, double cMAlpha_dot0) {

        return rho0*surf*Math.pow(cbar, 2)*cMAlpha_dot0/(4*iYY);


    }

    /**
     *   calculates the dimensional stability derivative of pitching moment M with respect
     to "q",
     *   divided by the longitudinal moment of inertia Iyy
     * @param rho0 air density
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param surf wing area
     * @param cbar mean aerodynamic chord
     * @param iYY longitudinal moment of inertia  (IYY)
     * @return calcMªq dimensional derivative [s^(-1)]
     */
    public static double calcM_q (double rho0, double mass, double u0, double q0, double
    surf, double cbar,
            double iYY, double cMq) {

        return rho0*u0*surf*Math.pow(cbar, 2)*cMq/(4*iYY);

    }
```

```
    // Longitudinal Dimensional Control Derivatives

    /**
     *   calculates the dimensional control derivative of force component X with respect to
     "delta_T",
     *   divided by the mass, for Constant Thrust (appropriate for jet aircraft or
     *   for unpowered flight)
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cTfix thrust coefficient at a fixed point ( U0 = u , delta_T = 1 )
     * @param kv scale factor of the effect on the propulsion due to the speed
     * @return Xªdelta_T_CT dimensional derivative [m * s^(-2)]
     */
    public static double calcX_delta_T_CT (double rho0, double surf, double mass, double u0,
    double q0,
            double cTfix, double kv) {


        return -q0*surf*(cTfix+kv*Math.pow(u0, -2))/(mass);

    }

    /**
     *   calculates the dimensional control derivative of force component X with respect to
     "delta_T",
     *   divided by the mass, for Constant Power (appropriate for propeller aircraft
     *   with automatic pitch control and constant-speed propeller)
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cTfix thrust coefficient at a fixed point ( U0 = u , delta_T = 1 )
     * @param kv scale factor of the effect on the propulsion due to the speed
     * @return Xªdelta_T_CP dimensional derivative [m * s^(-2)]
     */
    public static double calcX_delta_T_CP (double rho0, double surf, double mass, double u0,
    double q0,
            double cTfix, double kv) {


        return -q0*surf*(cTfix+kv*Math.pow(u0, -3))/(mass);

    }

    /**
     *   calculates the dimensional control derivative of force component X with respect to
     "delta_T",
     *   divided by the mass, for Constant Mass Flow propulsion
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cTfix thrust coefficient at a fixed point ( U0 = u , delta_T = 1 )
     * @param kv scale factor of the effect on the propulsion due to the speed
     * @return Xªdelta_T_CMF dimensional derivative [m * s^(-2)]
     */
    public static double calcX_delta_T_CMF (double rho0, double surf, double mass, double
    u0, double q0,
            double cTfix, double kv) {
```

```
        return -q0*surf*(cTfix+kv*Math.pow(u0, -1))/(mass);

    }

    /**
     *   calculates the dimensional control derivative of force component X with respect to
     "delta_T",
     *   divided by the mass, for Ramjet propulsion
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cTfix thrust coefficient at a fixed point ( U0 = u , delta_T = 1 )
     * @param kv scale factor of the effect on the propulsion due to the speed
     * @return Xªdelta_T_RJ dimensional derivative [m * s^(-2)]
     */
    public static double calcX_delta_T_RJ (double rho0, double surf, double mass, double u0,
    double q0,
            double cTfix, double kv) {


        return -q0*surf*(cTfix+kv*Math.pow(u0, 0))/(mass);

    }

    /**
     *   calculates the dimensional control derivative of force component X with respect to
     "delta_T",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param clDelta_T lift coefficient with respect to delta_T (ClDelta_Tº) of the aircraft
     * @return Zªdelta_T dimensional derivative [m * s^(-2)]
     */
    public static double calcZ_delta_T (double rho0, double surf, double mass, double u0,
    double q0,
            double clDelta_T) {

        return 0;

    }

    /**
     *   calculates the dimensional control derivative of force component Z with respect to
     "delta_E",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param clDelta_E lift coefficient with respect to delta_E (ClDelta_Eº) of the
     aircraft
     * @return Zªdelta_E dimensional derivative [m * s^(-2)]
     */
    public static double calcZ_delta_E (double rho0, double surf, double mass, double u0,
    double q0,
            double clDelta_E) {

        return -q0*surf*(1/mass)*clDelta_E;

    }
```

```
    /**
     *   calculates the dimensional control derivative of pitching moment M with respect to
     "delta_T",
     *   divided by the longitudinal moment of inertia Iyy
     * @param rho0 air density
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param surf wing area
     * @param cbar mean aerodynamic chord
     * @param iYY longitudinal moment of inertia  (IYY)
     * @return calcMªdelta_T dimensional derivative [s^(-2)]
     */
    public static double calcM_delta_T (double rho0, double surf, double cbar, double u0,
    double q0,
            double iYY, double cMDelta_T) {

        return 0;

    }

    /**
     *   calculates the dimensional control derivative of pitching moment M with respect to
     "delta_E",
     *   divided by the longitudinal moment of inertia Iyy
     * @param rho0 air density
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param surf wing area
     * @param cbar mean aerodynamic chord
     * @param iYY longitudinal moment of inertia  (IYY)
     * @return calcMªdelta_E dimensional derivative [s^(-2)]
     */
    public static double calcM_delta_E (double rho0, double surf, double cbar, double u0,
    double q0,
            double iYY, double cMDelta_E) {

        return q0*surf*cbar*cMDelta_E/(iYY);

    }

    // Longitudinal Matrices

    /**
     *   generates the longitudinal coefficients matrix [A_Lon] of linearized equations of
     dynamics
     * @param propulsion_system propulsion regime type
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param cbar mean aerodynamic chord
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param m0 Mach number
     * @param gamma0 ramp angle
     * @param theta0_rad Euler angle [rad] (assuming gamma0 = theta0)
     * @param iYY longitudinal moment of inertia  (IYY)
     * @param cd0 drag coefficient at null incidence (Cdº) of the aircraft
     * @param cdM0 drag coefficient with respect to Mach (CdMº) of the aircraft
     * @param cdAlpha0 linear drag gradient (CdAlphaº) of the aircraft
     * @param cl0 lift coefficient at null incidence (Clº) of the aircraft
     * @param clAlpha0 linear lift gradient (ClAlphaº) of the aircraft
     * @param clAlpha_dot0 linear lift gradient time derivative (ClAlpha_dotº) of the aircraft
     * @param clQ0 lift coefficient with respect to q (ClQº) of the aircraft
     * @param cMAlpha0 pitching moment coefficient with respect to Alpha (CmAlphaº) of the
     aircraft
```

```java
     * @param cMAlpha0_dot pitching moment coefficient time derivative (CmAlpha_dot°) of the
     aircraft
     * @param cM_m0 pitching moment coefficient with respect to Mach number
     * @param cMq pitching moment coefficient with respect to q
     * @return matrix [A_Lon]
     */
    public static double[][] build_A_Lon_matrix (Propulsion propulsion_system,
            double rho0, double surf, double mass, double cbar, double u0, double q0,
            double cd0, double m0, double cdM0, double cl0, double clM0, double cdAlpha0,
            double gamma0, double theta0_rad, double clAlpha0, double clAlpha_dot0,
            double cMAlpha0, double cMAlpha0_dot, double clQ0, double iYY, double cM_m0,
            double cMq) {

        double [][] aLon = new double [4][4];

        // Propulsion type in the Xªu calculation
        switch (propulsion_system)
            {
            case CONSTANT_TRUST:
                aLon [0][0] = calcX_u_CT (rho0, surf, mass, u0, q0, cd0, m0, cdM0);
                break;
            case CONSTANT_POWER:
                aLon [0][0] = calcX_u_CP (rho0, surf, mass, u0, q0, cd0, m0, cdM0, cl0,
                gamma0);
                break;
            default:
                aLon [0][0] = calcX_u_CT (rho0, surf, mass, u0, q0, cd0, m0, cdM0);
                break;
            }

        double k = calcM_w_dot (rho0, surf, cbar, iYY, cMAlpha0_dot)/(1 - calcZ_w_dot (rho0,
        surf,
                    mass, cbar, clAlpha_dot0));

        // Construction of the Matrix [A Lon]
        aLon [0][1] = calcX_w(rho0, surf, mass, u0, q0, cl0, cdAlpha0);

        aLon [0][2] = 0;

        aLon [0][3] = -(9.8100)*Math.cos(theta0_rad);

        aLon [1][0] = calcZ_u (rho0, surf, mass, u0, q0, m0, cl0, clM0)/(1 - calcZ_w_dot
        (rho0, surf,
                mass, cbar, clAlpha_dot0));

        aLon [1][1] = calcZ_w (rho0, surf, mass, u0, q0, m0, cd0, clAlpha0)/(1 - calcZ_w_dot
        (rho0, surf,
                mass, cbar, clAlpha_dot0));

        aLon [1][2] = (calcZ_q (rho0, surf, mass, u0, q0, cbar, clQ0) + u0)/(1 - calcZ_w_dot
        (rho0, surf,
                mass, cbar, clAlpha_dot0));

        aLon [1][3] = -(9.8100)*Math.sin(theta0_rad)/(1 - calcZ_w_dot (rho0, surf,
                mass, cbar, clAlpha_dot0));

        aLon [2][0] = calcM_u (rho0, surf, mass, u0, q0, cbar, m0, iYY, cM_m0) + k*calcZ_u
        (rho0, surf,
                mass, u0, q0, m0, cl0, clM0);

        aLon [2][1] = calcM_w (rho0, surf, mass, u0, q0, cbar, iYY, cMAlpha0) + k*calcZ_w
        (rho0, surf,
                mass, u0, q0, m0, cd0, clAlpha0);

        aLon [2][2] = calcM_q (rho0, mass, u0, q0, surf, cbar, iYY, cMq) + k * ( calcZ_q
        (rho0, surf, mass,
                u0, q0, cbar, clQ0)  + u0);
```

```
        aLon [2][3] = -k*(9.8100)*Math.sin(theta0_rad);

        aLon [3][0] = 0;

        aLon [3][1] = 0;

        aLon [3][2] = 1;

        aLon [3][3] = 0;

        return aLon;
    }

    /**
     * generates the longitudinal control coefficients matrix [B_Lon] of linearized
     equations of dynamics
     * @param propulsion_system propulsion regime type
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param cbar mean aerodynamic chord
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cd0 drag coefficient at null incidence (Cd°) of the aircraft
     * @param m0 Mach number
     * @param cdM0 drag coefficient with respect to Mach (CdM°) of the aircraft
     * @param cl0 lift coefficient at null incidence (Cl°) of the aircraft
     * @param cdAlpha0 linear drag gradient (CdAlpha°) of the aircraft
     * @param gamma0 ramp angle
     * @param theta0_rad Euler angle [rad] (assuming gamma0 = theta0)
     * @param clAlpha0 linear lift gradient (ClAlpha°) of the aircraft
     * @param clAlpha_dot0 linear lift gradient time derivative (ClAlpha_dot°) of the aircraft
     * @param cMAlpha0 pitching moment coefficient with respect to Alpha (CmAlpha°) of the
     aircraft
     * @param cMAlpha0_dot pitching moment coefficient time derivative (CmAlpha_dot°) of the
     aircraft
     * @param clQ0 lift coefficient with respect to q (ClQ°) of the aircraft
     * @param iYY longitudinal moment of inertia  (IYY)
     * @param cM_m0 pitching moment coefficient with respect to Mach number
     * @param cMq pitching moment coefficient with respect to q
     * @param cTfix thrust coefficient at a fixed point ( U0 = u , delta_T = 1 )
     * @param kv scale factor of the effect on the propulsion due to the speed
     * @param clDelta_T lift coefficient with respect to delta_T (ClDelta_T°) of the aircraft
     * @param clDelta_E lift coefficient with respect to delta_E (ClDelta_E°) of the aircraft
     * @param cMDelta_T pitching moment coefficient with respect to delta_T (CMDelta_T°) of
     the aircraft
     * @param cMDelta_E pitching moment coefficient with respect to delta_E (CMDelta_E°) of
     the aircraft
     * @return
     */
    public static double[][] build_B_Lon_matrix (Propulsion propulsion_system, double rho0,
            double surf, double mass, double cbar, double u0, double q0, double cd0, double
            m0, double cdM0,
            double cl0, double cdAlpha0, double gamma0, double theta0_rad, double clAlpha0,
            double clAlpha_dot0, double cMAlpha0, double cMAlpha_dot0, double clQ0, double
            iYY,
            double cM_m0, double cMq, double cTfix, double kv, double clDelta_T, double
            clDelta_E,
            double cMDelta_T, double cMDelta_E) {

        double [][] bLon = new double [4][2];

        // Propulsion type in the Xªdelta_T calculation
        switch (propulsion_system)
        {
        case CONSTANT_TRUST:
```

```
            bLon [0][0] = calcX_delta_T_CT (rho0, surf, mass, u0, q0, cTfix, kv);
            break;
        case CONSTANT_POWER:
            bLon [0][0] = calcX_delta_T_CP (rho0, surf, mass, u0, q0, cTfix, kv);
            break;
        case CONSTANT_MASS_FLOW:
            bLon [0][0] = calcX_delta_T_CMF (rho0, surf, mass, u0, q0, cTfix, kv);
            break;
        case RAMJET:
            bLon [0][0] = calcX_delta_T_RJ (rho0, surf, mass, u0, q0, cTfix, kv);
            break;
        default:
            bLon [0][0] = calcX_delta_T_CT (rho0, surf, mass, u0, q0, cTfix, kv);
            break;
        }

        // Coefficient kª calculation
        double k = calcM_w_dot (rho0, surf, cbar, iYY, cMAlpha_dot0)/(1 - calcZ_w_dot (rho0,
        surf,
                    mass, cbar, clAlpha_dot0));

        // Construction of the Matrix [B Lon]
        bLon [0][1] = 0;

        bLon [1][0] = calcZ_delta_T (rho0, surf, mass, u0, q0, clDelta_T) / (1 - calcZ_w_dot
        (rho0, surf,
                mass, cbar, clAlpha_dot0));

        bLon [1][1] = calcZ_delta_E (rho0, surf, mass, u0, q0, clDelta_E) / (1 - calcZ_w_dot
        (rho0, surf,
                mass, cbar, clAlpha_dot0));

        bLon [2][0] = calcM_delta_T (rho0, surf, cbar, u0, q0, iYY, cMDelta_T) + k *
        calcZ_delta_T (rho0, surf,
                mass, u0, q0, clDelta_T);

        bLon [2][1] = calcM_delta_E (rho0, surf, cbar, u0, q0, iYY, cMDelta_E) + k *
        calcZ_delta_E (rho0, surf,
                mass, u0, q0, clDelta_E);

        bLon [3][0] = 0;

        bLon [3][1] = 0;

        return bLon;

    }


    //////////////////////       LATERAL-DIRECTIONAL DYNAMICS       //////////////////////

    // Lateral-Directional Dimensional Stability Derivatives

    /**
     *   calculates the dimensional stability derivative of force component Y with respect
     to "beta",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cyBeta lateral force coefficient with respect to beta (CyBeta) of the aircraft
     * @return Yªbeta dimensional derivative [m * s^(-2)]
     */
    public static double calcY_beta (double rho0, double surf, double mass, double u0,
    double q0,
```

```
            double cyBeta) {

        return q0*surf*cyBeta/(mass);

    }

    /**
     *   calculates the dimensional stability derivative of force component Y with respect
     to "p",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param bbar wingspan
     * @param cyP lateral force coefficient with respect to p (CyP) of the aircraft
     * @return Yªp dimensional derivative [m * s^(-1)]
     */
    public static double calcY_p (double rho0, double surf, double mass, double u0, double q0,
            double bbar, double cyP) {

        return q0*surf*bbar*cyP/(2*u0*mass);

    }

    /**
     *   calculates the dimensional stability derivative of force component Y with respect
     to "r",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param bbar wingspan
     * @param cyR lateral force coefficient with respect to r (CyR) of the aircraft
     * @return Yªr dimensional derivative [m * s^(-1)]
     */
    public static double calcY_r (double rho0, double surf, double mass, double u0, double q0,
            double bbar, double cyR) {

        return q0*surf*bbar*cyR/(2*u0*mass);

    }

    /**
     *   calculates the dimensional stability derivative of rolling moment L with respect to
     "beta",
     *   divided for the lateral-directional moment of inertia I_xx
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iXX lateral-directional moment of inertia I_xx
     * @param cLBeta rolling moment coefficient with respect to beta (CLBeta) of the aircraft
     * @return Lªbeta dimensional derivative [s^(-2)]
     */
    public static double calcL_beta (double rho0, double surf, double bbar, double iXX,
            double u0, double q0, double cLBeta) {

        return q0*surf*bbar*cLBeta/(iXX);

    }

    /**
```

```
     *   calculates the dimensional stability derivative of rolling moment L with respect to
     "p",
     *   divided for the lateral-directional moment of inertia I_xx
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iXX lateral-directional moment of inertia I_xx
     * @param cLP rolling moment coefficient with respect to a p (CLP) of the aircraft
     * @return Lªp dimensional derivative [s^(-1)]
     */
    public static double calcL_p (double rho0, double surf, double bbar, double iXX,
            double u0, double q0, double cLP) {

        return q0*surf*bbar*(bbar/(2*u0))*cLP/(iXX);

    }

    /**
     *   calculates the dimensional stability derivative of rolling moment L with respect to
     "r",
     *   divided for the lateral-directional moment of inertia I_xx
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iXX lateral-directional moment of inertia I_xx
     * @param cLR rolling moment coefficient with respect to a r (CLR) of the aircraft
     * @return Lªr dimensional derivative [s^(-1)]
     */
    public static double calcL_r (double rho0, double surf, double bbar, double iXX,
            double u0, double q0, double cLR) {

        return q0*surf*bbar*(bbar/(2*u0))*cLR/(iXX);

    }

    /**
     *   calculates the dimensional stability derivative of yawing moment N with respect to
     "beta",
     *   divided for the lateral-directional moment of inertia I_zz
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param cNBeta yawing moment coefficient with respect to a beta (CNBeta) of the aircraft
     * @return Nªbeta dimensional derivative [s^(-2)]
     */
    public static double calcN_beta (double rho0, double surf, double bbar, double iZZ,
            double u0, double q0, double cNBeta) {

        return q0*surf*bbar*cNBeta/(iZZ);

    }

    /**
     *   calculates the dimensional stability derivative of yawing moment N with respect to
     "p",
     *   divided for the lateral-directional moment of inertia I_zz
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
```

```java
     * @param q0 dynamic pressure
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param cNP yawing moment coefficient with respect to p (CNP) of the aircraft
     * @return Nªp dimensional derivative [s^(-1)]
     */
    public static double calcN_p (double rho0, double surf, double bbar, double iZZ,
            double u0, double q0, double cNP) {

        return q0*surf*bbar*(bbar/(2*u0))*cNP/(iZZ);

    }

    /**
     *   calculates the dimensional stability derivative of yawing moment N with respect to
     "r",
     *   divided for the lateral-directional moment of inertia I_zz
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param cNR yawing moment coefficient with respect to p (CNP) of the aircraft
     * @return Nªr dimensional derivative [s^(-1)]
     */
    public static double calcN_r (double rho0, double surf, double bbar, double iZZ,
            double u0, double q0, double cNR) {

        return q0*surf*bbar*(bbar/(2*u0))*cNR/(iZZ);

    }

    // Lateral-Directional Dimensional Control Derivatives

    /**
     *   calculates the dimensional control derivative of force component Y with respect to
     "delta_A",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cyDelta_A lateral force coefficient with respect to delta_A (CyDelta_A) of the
     aircraft
     * @return Yªdelta_A dimensional derivative [m * s^(-2)]
     */
    public static double calcY_delta_A (double rho0, double surf, double mass, double u0,
    double q0,
            double cyDelta_A) {

        return q0*surf*cyDelta_A/(mass);

    }

    /**
     *   calculates the dimensional control derivative of force component Y with respect to
     "delta_R",
     *   divided by the mass
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param cyDelta_R lateral force coefficient with respect to delta_R (CyDelta_R) of the
     aircraft
     * @return Yªdelta_R dimensional derivative [m * s^(-2)]
```

```java
     */
    public static double calcY_delta_R (double rho0, double surf, double mass, double u0,
    double q0,
            double cyDelta_R) {

        return q0*surf*cyDelta_R/(mass);

    }

    /**
     *   calculates the dimensional control derivative of rolling moment L with respect to
     "delta_A",
     *   divided for the lateral-directional moment of inertia I_xx
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iXX lateral-directional moment of inertia I_xx
     * @param cLDelta_A rolling moment coefficient with respect to a delta_A (CLDelta_A) of
     the aircraft
     * @return Lªdelta_A dimensional derivative [s^(-2)]
     */
    public static double calcL_delta_A (double rho0, double surf, double bbar, double iXX,
            double u0, double q0, double cLDelta_A) {

        return q0*surf*bbar*cLDelta_A/(iXX);

    }

    /**
     *   calculates the dimensional control derivative of rolling moment L with respect to
     "delta_R",
     *   divided for the lateral-directional moment of inertia I_xx
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iXX lateral-directional moment of inertia I_xx
     * @param cLDelta_R rolling moment coefficient with respect to a delta_R (CLDelta_R) of
     the aircraft
     * @return Lªdelta_R dimensional derivative [s^(-2)]
     */
    public static double calcL_delta_R (double rho0, double surf, double bbar, double iXX,
            double u0, double q0, double cLDelta_R) {

        return q0*surf*bbar*cLDelta_R/(iXX);

    }

    /**
     *   calculates the dimensional control derivative of yawing moment N with respect to
     "delta_A",
     *   divided for the lateral-directional moment of inertia I_zz
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param cNDelta_A yawing moment coefficient with respect to delta_A (CNDelta_A) of the
     aircraft
     * @return Nªdelta_A dimensional derivative [s^(-2)]
     */
    public static double calcNªdelta_A (double rho0, double surf, double bbar, double iZZ,
            double u0, double q0, double cNDelta_A) {
```

```
        return q0*surf*bbar*cNDelta_A/(iZZ);

    }

    /**
     *   calculates the dimensional control derivative of yawing moment N with respect to
     "delta_R",
     *   divided for the lateral-directional moment of inertia I_zz
     * @param rho0 air density
     * @param surf wing area
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param cNDelta_R yawing moment coefficient with respect to delta_R (CNDelta_R) of the
     aircraft
     * @return Nªdelta_R dimensional derivative [s^(-2)]
     */
    public static double calcN_delta_R (double rho0, double surf, double bbar, double iZZ,
            double u0, double q0, double cNDelta_R) {

        return q0*surf*bbar*cNDelta_R/(iZZ);

    }

    // Lateral-Directional Matrices

    /**
     *   generates the lateral-directional coefficients matrix [A_LD] of linearized
     equations of dynamics
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param cbar mean aerodynamic chord
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param theta0_rad Euler angle [rad] (assuming gamma0 = theta0)
     * @param iXX lateral-directional moment of inertia I_xx
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param iXZ lateral-directional product of inertia I_xz
     * @param cyBeta lateral force coefficient with respect to beta (CyBeta) of the aircraft
     * @param cyP lateral force coefficient with respect to p (CyP) of the aircraft
     * @param cyR lateral force coefficient with respect to r (CyR) of the aircraft
     * @param cLBeta rolling moment coefficient with respect to beta (CLBeta) of the aircraft
     * @param cLP rolling moment coefficient with respect to a p (CLP) of the aircraft
     * @param cLR rolling moment coefficient with respect to a r (CLR) of the aircraft
     * @param cNBeta yawing moment coefficient with respect to a beta (CNBeta) of the aircraft
     * @param cNP yawing moment coefficient with respect to p (CNP) of the aircraft
     * @param cNR yawing moment coefficient with respect to r (CNR) of the aircraft
     * @return matrix [A_LD]
     */
    public static double[][] build_A_LD_matrix (double rho0, double surf, double mass,
            double cbar, double bbar, double u0, double q0, double theta0_rad, double iXX,
            double iZZ,
            double iXZ, double cyBeta, double cyP, double cyR, double cyDelta_A,
            double cyDelta_R, double cLBeta, double cLP, double cLR, double cLDelta_A,
            double cLDelta_R, double cNBeta, double cNP, double cNR, double cNDelta_A,
            double cNDelta_R) {

        double [][] aLD = new double [4][4];

        // Inertia coefficient calculation
        double i1 = iXZ/iXX;
        double i2 = iXZ/iZZ;
```

```
        // Primed Derivatives calculation
        double Y_beta_1 = calcY_beta (rho0, surf, mass, u0, q0, cyBeta);
        double Y_p_1 = calcY_p (rho0, surf, mass, u0, q0, bbar, cyP);
        double Y_r_1 = calcY_r (rho0, surf, mass, u0, q0, bbar, cyR);
        double L_beta_1 = (calcL_beta (rho0, surf, bbar, iXX, u0, q0, cLBeta) +
                i1*calcN_beta (rho0, surf, bbar, iZZ, u0, q0, cNBeta))/(1-i1*i2);
        double L_p_1 = (calcL_p (rho0, surf, bbar, iXX, u0, q0, cLP) +
                i1*calcN_p (rho0, surf, bbar, iZZ, u0, q0, cNP))/(1-i1*i2);
        double L_r_1 = (calcL_r (rho0, surf, bbar, iXX, u0, q0, cLR) +
                i1*calcN_r (rho0, surf, bbar, iZZ, u0, q0, cNR))/(1-i1*i2);
        double N_beta_1 = (i2*calcL_beta (rho0, surf, bbar, iXX, u0, q0, cLBeta) +
                calcN_beta (rho0, surf, bbar, iZZ, u0, q0, cNBeta))/(1-i1*i2);
        double N_p_1 = (i2*calcL_p (rho0, surf, bbar, iXX, u0, q0, cLP) +
                calcN_p (rho0, surf, bbar, iZZ, u0, q0, cNP))/(1-i1*i2);
        double N_r_1 = (i2*calcL_r (rho0, surf, bbar, iXX, u0, q0, cLR) +
                calcN_r (rho0, surf, bbar, iZZ, u0, q0, cNR))/(1-i1*i2);

        // Construction of the Matrix [A_LD]
        aLD [0][0] = N_r_1;

        aLD [0][1] = N_beta_1;

        aLD [0][2] = N_p_1;

        aLD [0][3] = 0;

        aLD [1][0] = Y_r_1/u0 - 1;

        aLD [1][1] = Y_beta_1/u0;

        aLD [1][2] = Y_p_1/u0;

        aLD [1][3] = (9.8100)*Math.cos(theta0_rad)/u0;

        aLD [2][0] = L_r_1;

        aLD [2][1] = L_beta_1;

        aLD [2][2] = L_p_1;

        aLD [2][3] = 0;

        aLD [3][0] = Math.sin(theta0_rad)/Math.cos(theta0_rad);

        aLD [3][1] = 0;

        aLD [3][2] = 1;

        aLD [3][3] = 0;

        return aLD;
    }

    /**
     *   generates the lateral-directional control coefficients matrix [B_LD] of linearized
     equations of dynamics
     * @param rho0 air density
     * @param surf wing area
     * @param mass total mass
     * @param cbar mean aerodynamic chord
     * @param bbar wingspan
     * @param u0 speed of the aircraft
     * @param q0 dynamic pressure
     * @param iXX lateral-directional moment of inertia I_xx
     * @param iZZ lateral-directional moment of inertia I_zz
     * @param iXZ lateral-directional product of inertia I_xz
     * @param cyDelta_A lateral force coefficient with respect to delta_A (CyDelta_A) of the
```

```
     aircraft
     * @param cyDelta_R lateral force coefficient with respect to delta_R (CyDelta_R) of the
     aircraft
     * @param cLDelta_A rolling moment coefficient with respect to a delta_A (CLDelta_A) of
     the aircraft
     * @param cLDelta_R rolling moment coefficient with respect to a delta_R (CLDelta_R) of
     the aircraft
     * @param cNDelta_A yawing moment coefficient with respect to a delta_A (CLDelta_A) of
     the aircraft
     * @param cNDelta_R yawing moment coefficient with respect to a delta_R (CLDelta_R) of
     the aircraft
     * @return matrix [B_LD]
     */
    public static double[][] build_B_LD_matrix (double rho0, double surf, double mass,
            double cbar, double bbar, double u0, double q0, double iXX, double iZZ, double
            iXZ,
            double cyDelta_A, double cyDelta_R, double cLDelta_A, double cLDelta_R,
            double cNDelta_A, double cNDelta_R) {

        double [][] bLD = new double [4][2];

        // Inertia coefficient calculation
        double i1 = iXZ/iXX;
        double i2 = iXZ/iZZ;

        //Primed Derivatives calculation
        double Y_delta_A_1 = calcY_delta_A (rho0, surf, mass, u0, q0, cyDelta_A);
        double Y_delta_R_1 = calcY_delta_R (rho0, surf, mass, u0, q0, cyDelta_R);
        double L_delta_A_1 = (calcL_delta_A (rho0, surf, bbar, iXX, u0, q0, cLDelta_A) +
                i1*calcNªdelta_A (rho0, surf, bbar, iZZ, u0, q0, cNDelta_A))/(1-i1*i2);
        double L_delta_R_1 = (calcL_delta_R (rho0, surf, bbar, iXX, u0, q0, cLDelta_R) +
                i1*calcN_delta_R (rho0, surf, bbar, iZZ, u0, q0, cNDelta_R))/(1-i1*i2);
        double N_delta_A_1 = (i2*calcL_delta_A (rho0, surf, bbar, iXX, u0, q0, cLDelta_A) +
                calcNªdelta_A (rho0, surf, bbar, iZZ, u0, q0, cNDelta_A))/(1-i1*i2);
        double N_delta_R_1 = (i2*calcL_delta_R (rho0, surf, bbar, iXX, u0, q0, cLDelta_R) +
                calcN_delta_R (rho0, surf, bbar, iZZ, u0, q0, cNDelta_R))/(1-i1*i2);

        // Construction of the Matrix [B_LD]
        bLD [0][0] = N_delta_A_1;

        bLD [0][1] = N_delta_R_1;

        bLD [1][0] = Y_delta_A_1/u0;

        bLD [1][1] = Y_delta_R_1/u0;

        bLD [2][0] = L_delta_A_1;

        bLD [2][1] = L_delta_R_1;

        bLD [3][0] = 0;

        bLD [3][1] = 0;

        return bLD;
    }

}
```

DYNAMIC STABILITY CALCULATOR

```java
package newproj;

    import org.apache.commons.math3.linear.Array2DRowRealMatrix;

    //import java.text.DecimalFormat;
    //import java.util.Arrays;

    import org.apache.commons.math3.linear.EigenDecomposition;
    import org.apache.commons.math3.linear.MatrixUtils;
    import org.apache.commons.math3.linear.RealMatrix;
    import org.apache.commons.math3.linear.RealVector;

public class DynamicStabilityCalculator {

    /**
     *   calculates EigenValues from a square matrix 4x4 (such as [A_Lon] and [A_LD])
     *   and puts them in a matrix 4x2 (a single line contains a specific eigenvalue
     *   in the form [ lambda(i)_Re , lambda(i)_Img ])
     * @param aMatrix
     * @return lambda_Matrix
     */
    public static double[][] buildEigenValuesMatrix (double aMatrix[][]) {

        RealMatrix aLonRM = MatrixUtils.createRealMatrix(aMatrix);
        EigenDecomposition aLonDecomposition = new EigenDecomposition(aLonRM);
        double[] reEigen = aLonDecomposition.getRealEigenvalues();
        double[] imgEigen = aLonDecomposition.getImagEigenvalues();

        double [][] lambda_Matrix = new double [4][2];

        for (int i=0 ; i < 4 ; i++) {
            lambda_Matrix[i][0] = reEigen [i];
            lambda_Matrix[i][1] = imgEigen [i];
        }

        return lambda_Matrix;
    }

    /**
     *   calculates i° EigenVector from a square matrix 4x4 (such as [A_Lon] and [A_LD])
     * @param aMatrix
     * @param index
     * @return
     */
    public static RealVector buildEigenVector (double aMatrix[][], int index) {
        RealMatrix aRM = new Array2DRowRealMatrix(aMatrix);
        EigenDecomposition eigDec = new EigenDecomposition(aRM);

        RealVector eigVec = eigDec.getEigenvector(index);

        return eigVec;
    }

    /**
     *   calculates Damping Coefficient
     * @param sigma - lambda(i)_Re
     * @param omega - lambda(i)_Img
     * @return Damping Coefficient
     */
    public static double calcZeta (double sigma, double omega) {

        return Math.sqrt( 1 / ( 1 + Math.pow( omega/sigma , 2 )));
    }

    /**
     *   calculates Natural Frequency
     * @param sigma - lambda(i)_Re
```

```java
     * @param omega - lambda(i)_Img
     * @return Natural Frequency [s^(-1)]
     */
    public static double calcOmega_n (double sigma, double omega) {

        return -(sigma)/calcZeta(sigma,omega);
    }

    /**
     *   calculates Period
     * @param sigma - lambda(i)_Re
     * @param omega - lambda(i)_Img
     * @return Period [s]
     */
    public static double calcT (double sigma, double omega) {

        return 2*Math.PI / ( calcOmega_n (sigma,omega) *
                Math.sqrt( 1 - Math.pow(calcZeta (sigma,omega) , 2)));
    }

    /**
     *   calculates Halving Time
     * @param sigma - lambda(i)_Re
     * @param omega - lambda(i)_Img
     * @return Halving Time [s]
     */
    public static double calct_half (double sigma, double omega) {

        return Math.log(2) / (calcOmega_n (sigma,omega) * calcZeta (sigma,omega));
    }

    /**
     *   calculates number of cycles to Halving Time
     * @param sigma - lambda(i)_Re
     * @param omega - lambda(i)_Img
     * @return number of cycles to Halving Time
     */
    public static double calcN_half (double sigma, double omega) {

        return calct_half (sigma,omega) / calcT (sigma,omega);
    }

}
```

FLIGHT DYNAMICS MANAGER

```
package newproj;

import java.io.File;
import java.io.FileInputStream;
import java.text.DecimalFormat;

import org.apache.commons.math3.linear.RealVector;
import org.apache.poi.ss.usermodel.Cell;
import org.apache.poi.ss.usermodel.Row;
import org.apache.poi.ss.usermodel.Sheet;
import org.apache.poi.ss.usermodel.Workbook;
import org.apache.poi.ss.usermodel.WorkbookFactory;

import newproj.StabilityDerivativesCalc.Propulsion;


public class FlightDynamicsManager {

    Propulsion propulsion_system = Propulsion.CONSTANT_TRUST;  // propulsion regime type

    double rho0;                                  // air density
    double surf;                                  // wing area
    double mass;                                  // total mass
    double cbar;                                  // mean aerodynamic chord
    double bbar;                                  // wingspan
    double u0;                                    // speed of the aircraft
    double q0;                                    // dynamic pressure
    double m0;                                    // Mach number
    double gamma0;                                // ramp angle
    double theta0_rad = Math.toRadians(gamma0); // Euler angle [rad] (assuming gamma0 =
    theta0)
    double iXX;                                   // lateral-directional moment of inertia (IXX)
    double iYY;                                   // longitudinal moment of inertia  (IYY)
    double iZZ;                                   // lateral-directional moment of inertia (IZZ)
    double iXZ;                                   // lateral-directional product of inertia
    (IXZ)
    double cd0;                                   // drag coefficient at null incidence (Cdº)
    of the aircraft
    double cdAlpha0;                              // linear drag gradient (CdAlphaº) of the
    aircraft
    double cdM0;                                  // drag coefficient with respect to Mach
    (CdMº) of the aircraft
    double cl0;                                   // lift coefficient at null incidence (Clº)
    of the aircraft
    double clAlpha0;                              // linear lift gradient (ClAlphaº) of the
    aircraft
    double clAlpha_dot0;                          // linear lift gradient time derivative
    (ClAlpha_dotº) of the aircraft
    double clM0;                                  // lift coefficient with respect to Mach
    (ClMº) of the aircraft
    double clQ0;                                  // lift coefficient with respect to q (ClQº)
    of the aircraft
    double clDelta_T;                             // lift coefficient with respect to delta_T
    (ClDelta_Tº) of the aircraft
    double clDelta_E;                             // lift coefficient with respect to delta_E
    (ClDelta_Eº) of the aircraft
    double cMAlpha0;                              // pitching moment coefficient with respect
    to Alpha (CmAlphaº) of the aircraft
    double cMAlpha_dot0;                          // pitching moment coefficient time
    derivative (CmAlpha_dotº) of the aircraft
    double cM_m0;                                 // pitching moment coefficient with respect
    to Mach number
    double cMq;                                   // pitching moment coefficient with respect
    to q
    double cMDelta_T;                             // pitching moment coefficient with respect
    to delta_T (CMDelta_Tº) of the aircraft
    double cMDelta_E;                             // pitching moment coefficient with respect
```

```
    to delta_E (CMDelta_E°) of the aircraft
    double cTfix;                           // thrust coefficient at a fixed point ( U0
    = u , delta_T = 1 )
    double kv;                              // scale factor of the effect on the
    propulsion due to the speed
    double cyBeta;                          // lateral force coefficient with respect to
    beta (CyBeta) of the aircraft
    double cyP;                             // lateral force coefficient with respect to
    p (CyP) of the aircraft
    double cyR;                             // lateral force coefficient with respect to
    r (CyR) of the aircraft
    double cyDelta_A;                       // lateral force coefficient with respect to
    delta_A (CyDelta_A) of the aircraft
    double cyDelta_R;                       // lateral force coefficient with respect to
    delta_R (CyDelta_R) of the aircraft
    double cLBeta;                          // rolling moment coefficient with respect
    to beta (CLBeta) of the aircraft
    double cLP;                             // rolling moment coefficient with respect
    to a p (CLP) of the aircraft
    double cLR;                             // rolling moment coefficient with respect
    to a r (CLR) of the aircraft
    double cLDelta_A;                       // rolling moment coefficient with respect
    to a delta_A (CLDelta_A) of the aircraft
    double cLDelta_R;                       // rolling moment coefficient with respect
    to a delta_R (CLDelta_R) of the aircraft
    double cNBeta;                          // yawing moment coefficient with respect to
    a beta (CNBeta) of the aircraft
    double cNP;                             // yawing moment coefficient with respect to
    p (CNP) of the aircraft
    double cNR;                             // yawing moment coefficient with respect to
    r (CNR) of the aircraft
    double cNDelta_A;                       // yawing moment coefficient with respect to
    delta_A (CNDelta_A) of the aircraft
    double cNDelta_R;                       // yawing moment coefficient with respect to
    delta_R (CNDelta_R) of the aircraft

    double x_u_CT;                  // dimensional derivative of force component X with
    respect to "u" for Constant Thrust
    double x_u_CP;                  // dimensional derivative of force component X with
    respect to "u" for Constant Power
    double x_w;                     // dimensional derivative of force component X with
    respect to "w"
    double x_w_dot;                 // dimensional derivative of force component X with
    respect to "w_dot"
    double x_q;                     // dimensional derivative of force component X with
    respect to "q"
    double z_u;                     // dimensional derivative of force component Z with
    respect to "u"
    double z_w;                     // dimensional derivative of force component Z with
    respect to "w"
    double z_w_dot;                 // dimensional derivative of force component Z with
    respect to "w_dot"
    double z_q;                     // dimensional derivative of force component Z with
    respect to "q"
    double m_u;                     // dimensional derivative of pitching moment M with
    respect to "u"
    double m_w;                     // dimensional derivative of pitching moment M with
    respect to "w"
    double m_w_dot;                 // dimensional derivative of pitching moment M with
    respect to "w_dot"
    double m_q;                     // dimensional derivative of pitching moment M with
    respect to "q"

    double x_delta_T_CT;            // dimensional control derivative of force component
    X with respect to "delta_T" for Constant Thrust
    double x_delta_T_CP;            // dimensional control derivative of force component
    X with respect to "delta_T" for Constant Power
```

```
    double x_delta_T_CMF;              // dimensional control derivative of force component
    X with respect to "delta_T" for Constant Mass Flow
    double x_delta_T_RJ;               // dimensional control derivative of force component
    X with respect to "delta_T" for RamJet
    double x_delta_E;                  // dimensional control derivative of force component
    X with respect to "delta_E"
    double z_delta_T;                  // dimensional control derivative of force component
    Z with respect to "delta_T"
    double z_delta_E;                  // dimensional control derivative of force component
    Z with respect to "delta_E"
    double m_delta_T;                  // dimensional control derivative of pitching moment
    M with respect to "delta_T"
    double m_delta_E;                  // dimensional control derivative of pitching moment
    M with respect to "delta_E"

    double y_beta;                     // dimensional derivative of force component Y with
    respect to "beta"
    double y_p    ;                    // dimensional derivative of force component Y with
    respect to "p"
    double y_r    ;                    // dimensional derivative of force component Y with
    respect to "r"
    double l_beta;                     // dimensional derivative of rolling moment L with
    respect to "beta"
    double l_p    ;                    // dimensional derivative of rolling moment L with
    respect to "p"
    double l_r    ;                    // dimensional derivative of rolling moment L with
    respect to "r"
    double n_beta;                     // dimensional derivative of yawing moment N with
    respect to "beta"
    double n_p    ;                    // dimensional derivative of yawing moment N with
    respect to "p"
    double n_r    ;                    // dimensional derivative of yawing moment N with
    respect to "r"

    double y_delta_A;                  // dimensional control derivative of force component
    Y with respect to "delta_A"
    double y_delta_R;                  // dimensional control derivative of force component
    Y with respect to "delta_R"
    double l_delta_A;                  // dimensional control derivative of rolling moment
    L with respect to "delta_A"
    double l_delta_R;                  // dimensional control derivative of rolling moment
    L with respect to "delta_R"
    double n_delta_A;                  // dimensional control derivative of yawing moment N
    with respect to "delta_A"
    double n_delta_R;                  // dimensional control derivative of yawing moment N
    with respect to "delta_R"

    double [][] aLon = new double [4][4];       // longitudinal coefficients [A_Lon] matrix
    double [][] bLon = new double [4][2];       // longitudinal control coefficients [B_Lon]
    matrix
    double [][] aLD = new double [4][4];        // lateral-directional coefficients [A_LD]
    matrix
    double [][] bLD = new double [4][2];        // lateral-directional control coefficients
    [B_LD] matrix

    double[][] lonEigenvaluesMatrix = new double [4][2];    // longitudinal eigenvalues matrix
    double[][] ldEigenvaluesMatrix = new double [4][2];     // lateral-directional
    eigenvalues matrix

    RealVector eigLonVec1;                      // longitudinal 1st eigenvector
    RealVector eigLonVec2;                      // longitudinal 2nd eigenvector
    RealVector eigLonVec3;                      // longitudinal 3rd eigenvector
    RealVector eigLonVec4;                      // longitudinal 4th eigenvector
    RealVector eigLDVec1;                       // lateral-directional 1st eigenvector
    RealVector eigLDVec2;                       // lateral-directional 2nd eigenvector
    RealVector eigLDVec3;                       // lateral-directional 3rd eigenvector
    RealVector eigLDVec4;                       // lateral-directional 4th eigenvector
```

```
    double zeta_SP;                              // Short Period mode damping coefficient
    double zeta_PH;                              // Phugoid mode damping coefficient
    double omega_n_SP;                           // Short Period mode natural frequency
    double omega_n_PH;                           // Phugoid mode natural frequency
    double period_SP;                            // Short Period mode period
    double period_PH;                            // Phugoid mode period
    double t_half_SP;                            // Short Period mode halving time
    double t_half_PH;                            // Phugoid mode halving time
    double N_half_SP;                            // Short Period mode number of cycles to
    halving time
    double N_half_PH;                            // Phugoid mode number of cycles to halving
    time

    double zeta_DR;                              // Dutch-Roll mode damping
    coefficient
    double omega_n_DR;                           // Dutch-Roll mode natural frequency
    double period_DR;                            // Dutch-Roll mode period
    double t_half_DR;                            // Dutch-Roll mode halving time
    double N_half_DR;                            // Dutch-Roll mode number of cycles to
    halving time



    public FlightDynamicsManager() {

    }



    public void calculateAll() {
        // Formats numbers up to 4 decimal places
        DecimalFormat df = new DecimalFormat("#,###,##0.0000");

        ////////////////////          LONGITUDINAL DYNAMICS          \\\\\\\\\\\\\\\\\\\\

        // Calculates the longitudinal Stability and Control DERIVATIVES \\
        x_u_CT  = StabilityDerivativesCalc.calcX_u_CT(rho0, surf, mass, u0, q0, cd0, m0,
        cdM0);
        x_u_CP  = StabilityDerivativesCalc.calcX_u_CP(rho0, surf, mass, u0, q0, cd0, cM_m0,
        cdM0, cl0, gamma0);
        x_w     = StabilityDerivativesCalc.calcX_w(rho0, surf, mass, u0, q0, cl0, cdAlpha0);
        x_w_dot = 0;
        x_q     = 0;
        z_u     = StabilityDerivativesCalc.calcZ_u(rho0, surf, mass, u0, q0, cM_m0, cl0,
        clM0);
        z_w     = StabilityDerivativesCalc.calcZ_w(rho0, surf, mass, u0, q0, cM_m0, cd0,
        clAlpha0);
        z_w_dot = StabilityDerivativesCalc.calcZ_w_dot(rho0, surf, mass, cbar, clAlpha_dot0);
        z_q     = StabilityDerivativesCalc.calcZ_q(rho0, surf, mass, u0, q0, cbar, clQ0);
        m_u     = StabilityDerivativesCalc.calcM_u(rho0, surf, mass, u0, q0, cbar, cdM0,
        iYY, cM_m0);
        m_w     = StabilityDerivativesCalc.calcM_w(rho0, surf, mass, u0, q0, cbar, iYY,
        cMAlpha0);
        m_w_dot = StabilityDerivativesCalc.calcM_w_dot(rho0, surf, cbar, iYY, cMAlpha_dot0);
        m_q     = StabilityDerivativesCalc.calcM_q(rho0, mass, u0, q0, surf, cbar, iYY, cMq);

        x_delta_T_CT  = StabilityDerivativesCalc.calcX_delta_T_CT (rho0, surf, mass, u0, q0,
        cTfix, kv);
        x_delta_T_CP  = StabilityDerivativesCalc.calcX_delta_T_CP (rho0, surf, mass, u0, q0,
        cTfix, kv);
        x_delta_T_CMF = StabilityDerivativesCalc.calcX_delta_T_CMF (rho0, surf, mass, u0,
        q0, cTfix, kv);
        x_delta_T_RJ  = StabilityDerivativesCalc.calcX_delta_T_RJ (rho0, surf, mass, u0, q0,
        cTfix, kv);
        x_delta_T_CT  = StabilityDerivativesCalc.calcX_delta_T_CT (rho0, surf, mass, u0, q0,
        cTfix, kv);
```

```
        x_delta_E     = 0;
        z_delta_T     = StabilityDerivativesCalc.calcZ_delta_T (rho0, surf, mass, u0, q0, clDelta_T);
        z_delta_E     = StabilityDerivativesCalc.calcZ_delta_E (rho0, surf, mass, u0, q0, clDelta_E);
        m_delta_T     = StabilityDerivativesCalc.calcM_delta_T (rho0, surf, cbar, u0, q0, iYY, cMDelta_T);
        m_delta_E     = StabilityDerivativesCalc.calcM_delta_E (rho0, surf, cbar, u0, q0, iYY, cMDelta_E);

        // Prints out the LONGITUDINAL STABILITY AND CONTROL DERIVATIVES LIST \\

        System.out.println("_________________________________________________________________\n");
        System.out.println("LONGITUDINAL STABILITY DERIVATIVES: \n");
        System.out.println(" Xªu_CT   = " + df.format(x_u_CT));
        System.out.println(" Xªu_CP   = " + df.format(x_u_CP));
        System.out.println(" Xªw      = " + df.format(x_w));
        System.out.println(" Xªw_dot = " + df.format(x_w_dot));
        System.out.println(" Xªq      = " + df.format(x_q));
        System.out.println(" Zªu      = " + df.format(z_u));
        System.out.println(" Zªw      = " + df.format(z_w));
        System.out.println(" Zªw_dot = " + df.format(z_w_dot));
        System.out.println(" Zªq      = " + df.format(z_q));
        System.out.println(" Mªu      = " + df.format(m_u));
        System.out.println(" Mªw      = " + df.format(m_w));
        System.out.println(" Mªw_dot = " + df.format(m_w_dot));
        System.out.println(" Mªq      = " + df.format(m_q));
        System.out.println("\n\nLONGITUDINAL CONTROL DERIVATIVES: \n");
        System.out.println(" Xªdelta_T_CT   = " + df.format(x_delta_T_CT));
        System.out.println(" Xªdelta_T_CP   = " + df.format(x_delta_T_CP));
        System.out.println(" Xªdelta_T_CMF = " + df.format(x_delta_T_CMF));
        System.out.println(" Xªdelta_T_RJ   = " + df.format(x_delta_T_RJ));
        System.out.println(" Xªdelta_E      = " + df.format(x_delta_E));
        System.out.println(" Zªdelta_T      = " + df.format(z_delta_T));
        System.out.println(" Zªdelta_E      = " + df.format(z_delta_E));
        System.out.println(" Mªdelta_T      = " + df.format(m_delta_T));
        System.out.println(" Mªdelta_E      = " + df.format(m_delta_E)+"\n");

        // Generates and prints out the [A_Lon] and [B_Lon] MATRICES \\

        System.out.println("_________________________________________________________________\n");
        System.out.println("MATRIX [A_LON]: \n");

        aLon = StabilityDerivativesCalc.build_A_Lon_matrix (propulsion_system, rho0, surf, mass, cbar, u0, q0, cd0, m0, cdM0, cl0,
                clM0, cdAlpha0, gamma0, theta0_rad,clAlpha0, clAlpha_dot0, cMAlpha0,
                cMAlpha_dot0, clQ0, iYY, cM_m0, cMq);


        System.out.println(df.format(aLon[0][0])+"\t\t"+df.format(aLon[0][1])+"\t\t"+df.format(aLon[0][2])+"\t\t"+df.format(aLon[0][3])+"\n");

        System.out.println(df.format(aLon[1][0])+"\t\t"+df.format(aLon[1][1])+"\t\t"+df.format(aLon[1][2])+"\t\t"+df.format(aLon [1][3])+"\n");

        System.out.println(df.format(aLon[2][0])+"\t\t"+df.format(aLon[2][1])+"\t\t"+df.format(aLon[2][2])+"\t\t"+df.format(aLon [2][3])+"\n");
        System.out.println(aLon[3][0]+"\t\t"+aLon[3][1]+"\t\t"+aLon[3][2]+"\t\t"+aLon[3][3]+"\n");


        System.out.println("_________________________________________________________________\n");
        System.out.println("MATRIX [B_LON]: \n");
```

```
        bLon = StabilityDerivativesCalc.build_B_Lon_matrix (propulsion_system, rho0, surf,
        mass, cbar, u0, q0, cd0, m0, cdM0, cl0,
                cdAlpha0, gamma0, theta0_rad, clAlpha0, clAlpha_dot0, cMAlpha0,
                cMAlpha_dot0, clQ0, iYY, cM_m0, cMq, cTfix,
                kv, clDelta_T, clDelta_E, cMDelta_T, cMDelta_E);

        System.out.println(df.format(bLon[0][0])+"\t\t"+df.format(bLon[0][1])+"\n");
        System.out.println(df.format(bLon[1][0])+"\t\t"+df.format(bLon[1][1])+"\n");
        System.out.println(df.format(bLon[2][0])+"\t\t"+df.format(bLon[2][1])+"\n");
        System.out.println(bLon[3][0]+"\t\t"+bLon[3][1]+"\n");

        // Generates and prints out the Eigenvalues of [A_Lon] matrix \\
        lonEigenvaluesMatrix = DynamicStabilityCalculator.buildEigenValuesMatrix(aLon);


        System.out.println("_________________________________________________________________\
        n");
        System.out.println("LONGITUDINAL EIGENVALUES\n");
        System.out.println("  SHORT PERIOD: "+df.format(lonEigenvaluesMatrix[0][0])+" ±
        j"+df.format(lonEigenvaluesMatrix[0][1])+"\n");
        System.out.println("  PHUGOID:      "+df.format(lonEigenvaluesMatrix[2][0])+" ±
        j"+df.format(lonEigenvaluesMatrix[2][1])+"\n");

        // Generates and prints out the EigenVectors of [A_Lon] matrix \\

        System.out.println("_________________________________________________________________\
        n");
        System.out.println("LONGITUDINAL EIGENVECTORS:\n");

        eigLonVec1 = DynamicStabilityCalculator.buildEigenVector(aLon, 0);
        eigLonVec2 = DynamicStabilityCalculator.buildEigenVector(aLon, 1);
        eigLonVec3 = DynamicStabilityCalculator.buildEigenVector(aLon, 2);
        eigLonVec4 = DynamicStabilityCalculator.buildEigenVector(aLon, 3);

        System.out.println("EigenVector 1 = " + eigLonVec1);
        System.out.println("EigenVector 2 = " + eigLonVec2);
        System.out.println("EigenVector 3 = " + eigLonVec3);
        System.out.println("EigenVector 4 = " + eigLonVec4+"\n");

        // Generates and prints out all the characteristics for longitudinal SHORT PERIOD
        and PHUGOID modes \\
        zeta_SP =
        DynamicStabilityCalculator.calcZeta(lonEigenvaluesMatrix[0][0],lonEigenvaluesMatrix[0]
        [1]);
        zeta_PH =
        DynamicStabilityCalculator.calcZeta(lonEigenvaluesMatrix[2][0],lonEigenvaluesMatrix[2]
        [1]);
        omega_n_SP =
        DynamicStabilityCalculator.calcOmega_n(lonEigenvaluesMatrix[0][0],lonEigenvaluesMatrix
        [0][1]);
        omega_n_PH =
        DynamicStabilityCalculator.calcOmega_n(lonEigenvaluesMatrix[2][0],lonEigenvaluesMatrix
        [2][1]);
        period_SP =
        DynamicStabilityCalculator.calcT(lonEigenvaluesMatrix[0][0],lonEigenvaluesMatrix[0][1]
        );
        period_PH =
        DynamicStabilityCalculator.calcT(lonEigenvaluesMatrix[2][0],lonEigenvaluesMatrix[2][1]
        );
        t_half_SP =
        DynamicStabilityCalculator.calct_half(lonEigenvaluesMatrix[0][0],lonEigenvaluesMatrix[
        0][1]);
        t_half_PH =
        DynamicStabilityCalculator.calct_half(lonEigenvaluesMatrix[2][0],lonEigenvaluesMatrix[
        2][1]);
        N_half_SP =
        DynamicStabilityCalculator.calcN_half(lonEigenvaluesMatrix[0][0],lonEigenvaluesMatrix[
```

```
0][1]);
N_half_PH =
DynamicStabilityCalculator.calcN_half(lonEigenvaluesMatrix[2][0],lonEigenvaluesMatrix[
2][1]);


System.out.println("_________________________________________________________________\
n");
System.out.println("SHORT PERIOD MODE CHARACTERISTICS\n");
System.out.println("Zeta_SP                           = "+df.format(zeta_SP)+"\n");
System.out.println("Omega_n_SP                        = "+df.format(omega_n_SP)+"\n");
System.out.println("Period                            = "+df.format(period_SP)+"\n");
System.out.println("Halving Time                      = "+df.format(t_half_SP)+"\n");
System.out.println("Number of cycles to Halving Time = "+df.format(N_half_SP)+"\n\n");

System.out.println("PHUGOID MODE CHARACTERISTICS\n");
System.out.println("Zeta_PH                           = "+df.format(zeta_PH)+"\n");
System.out.println("Omega_n_PH                        = "+df.format(omega_n_PH)+"\n");
System.out.println("Period                            = "+df.format(period_PH)+"\n");
System.out.println("Halving Time                      = "+df.format(t_half_PH)+"\n");
System.out.println("Number of cycles to Halving Time = "+df.format(N_half_PH)+"\n");

////////////////////          LATERAL-DIRECTIONAL DYNAMICS
\\\\\\\\\\\\\\\\\\\\

// Calculates the lateral-directional Stability and Control DERIVATIVES \\
y_beta = StabilityDerivativesCalc.calcY_beta (rho0, surf, mass, u0, q0, cyBeta);
y_p    = StabilityDerivativesCalc.calcY_p (rho0, surf, mass, u0, q0, bbar, cyP);
y_r    = StabilityDerivativesCalc.calcY_r (rho0, surf, mass, u0, q0, bbar, cyR);
l_beta = StabilityDerivativesCalc.calcL_beta (rho0, surf, bbar, iXX, u0, q0, cLBeta);
l_p    = StabilityDerivativesCalc.calcL_p (rho0, surf, bbar, iXX, u0, q0, cLP);
l_r    = StabilityDerivativesCalc.calcL_r (rho0, surf, bbar, iXX, u0, q0, cLR);
n_beta = StabilityDerivativesCalc.calcN_beta (rho0, surf, bbar, iZZ, u0, q0, cNBeta);
n_p    = StabilityDerivativesCalc.calcN_p (rho0, surf, bbar, iZZ, u0, q0, cNP);
n_r    = StabilityDerivativesCalc.calcN_r (rho0, surf, bbar, iZZ, u0, q0, cNR);
y_delta_A = StabilityDerivativesCalc.calcY_delta_A (rho0, surf, mass, u0, q0,
cyDelta_A);
y_delta_R = StabilityDerivativesCalc.calcY_delta_R (rho0, surf, mass, u0, q0,
cyDelta_R);
l_delta_A = StabilityDerivativesCalc.calcL_delta_A (rho0, surf, bbar, iXX, u0, q0,
cLDelta_A);
l_delta_R = StabilityDerivativesCalc.calcL_delta_R (rho0, surf, bbar, iXX, u0, q0,
cLDelta_R);
n_delta_A = StabilityDerivativesCalc.calcNªdelta_A (rho0, surf, bbar, iZZ, u0, q0,
cNDelta_A);
n_delta_R = StabilityDerivativesCalc.calcN_delta_R (rho0, surf, bbar, iZZ, u0, q0,
cNDelta_R);

// Prints out the LATERAL-DIRECTIONAL STABILITY AND CONTROL DERIVATIVES LIST \\

System.out.println("_________________________________________________________________\
n");
System.out.println("LATERAL-DIRECTIONAL STABILITY DERIVATIVES: \n");
System.out.println(" Yªbeta = " + df.format(y_beta));
System.out.println(" Yªp    = " + df.format(y_p));
System.out.println(" Yªr    = " + df.format(y_r));
System.out.println(" Lªbeta = " + df.format(l_beta));
System.out.println(" Lªp    = " + df.format(l_p));
System.out.println(" Lªr    = " + df.format(l_r));
System.out.println(" Nªbeta = " + df.format(n_beta));
System.out.println(" Nªp    = " + df.format(n_p));
System.out.println(" Nªr    = " + df.format(n_r));
System.out.println("\n\nLATERAL-DIRECTIONAL CONTROL DERIVATIVES: \n");
System.out.println(" Yªdelta_A = " + df.format(y_delta_A));
System.out.println(" Yªdelta_R = " + df.format(y_delta_R));
System.out.println(" Lªdelta_A = " + df.format(l_delta_A));
System.out.println(" Lªdelta_R = " + df.format(l_delta_R));
```

```
        System.out.println(" Nªdelta_A = " + df.format(n_delta_A));
        System.out.println(" Nªdelta_R = " + df.format(n_delta_R)+"\n");

        // Generates and prints out the [A_Ld] and [B_Ld] MATRICES \\

        System.out.println("_________________________________________________________________\
n");
        System.out.println("MATRIX [A_LD]: \n");

        aLD = StabilityDerivativesCalc.build_A_LD_matrix (rho0, surf, mass, cbar, bbar, u0,
q0,
                theta0_rad, iXX, iZZ, iXZ, cyBeta, cyP, cyR, cyDelta_A, cyDelta_R, cLBeta,
                cLP, cLR, cLDelta_A, cLDelta_R, cNBeta, cNP, cNR, cNDelta_A, cNDelta_R);


        System.out.println(df.format(aLD[0][0])+"\t\t"+df.format(aLD[0][1])+"\t\t"+df.format(a
LD[0][2])+"\t\t"+df.format(aLD[0][3])+"\n");

        System.out.println(df.format(aLD[1][0])+"\t\t"+df.format(aLD[1][1])+"\t\t"+df.format(a
LD[1][2])+"\t\t"+df.format(aLD [1][3])+"\n");

        System.out.println(df.format(aLD[2][0])+"\t\t"+df.format(aLD[2][1])+"\t\t"+df.format(a
LD[2][2])+"\t\t"+df.format(aLD [2][3])+"\n");
        System.out.println(aLD[3][0]+"\t\t"+aLD[3][1]+"\t\t"+aLD[3][2]+"\t\t"+aLD
[3][3]+"\n");


        System.out.println("_________________________________________________________________\
n");
        System.out.println("MATRIX [B_LD]: \n");

        bLD = StabilityDerivativesCalc.build_B_LD_matrix (rho0, surf, mass, cbar, bbar, u0,
q0,
                iXX, iZZ, iXZ, cyDelta_A, cyDelta_R, cLDelta_A, cLDelta_R, cNDelta_A,
                cNDelta_R);

        System.out.println(df.format(bLD[0][0])+"\t\t"+df.format(bLD[0][1])+"\n");
        System.out.println(df.format(bLD[1][0])+"\t\t"+df.format(bLD[1][1])+"\n");
        System.out.println(df.format(bLD[2][0])+"\t\t"+df.format(bLD[2][1])+"\n");
        System.out.println(bLD[3][0]+"\t\t"+bLD[3][1]+"\n");

        // Generates and prints out the Eigenvalues of [A_Ld] matrix \\
        ldEigenvaluesMatrix = DynamicStabilityCalculator.buildEigenValuesMatrix(aLD);


        System.out.println("_________________________________________________________________\
n");
        System.out.println("LATERAL-DIRECTIONAL EIGENVALUES\n");
        System.out.println("  ROLL:       "+df.format(ldEigenvaluesMatrix[2][0])+"\n");
        System.out.println("  DUTCH-ROLL: "+df.format(ldEigenvaluesMatrix[0][0])+" ±
j"+df.format(ldEigenvaluesMatrix[0][1])+"\n");
        System.out.println("  SPIRAL:     "+df.format(ldEigenvaluesMatrix[3][0])+"\n");

        // Generates and prints out the EigenVectors of [A_Ld] matrix \\

        System.out.println("_________________________________________________________________\
n");
        System.out.println("LATERAL-DIRECTIONAL EIGENVECTORS:\n");

        eigLDVec1 = DynamicStabilityCalculator.buildEigenVector(aLD, 0);
        eigLDVec2 = DynamicStabilityCalculator.buildEigenVector(aLD, 1);
        eigLDVec3 = DynamicStabilityCalculator.buildEigenVector(aLD, 2);
        eigLDVec4 = DynamicStabilityCalculator.buildEigenVector(aLD, 3);

        System.out.println("EigenVector 1 = " + eigLDVec1);
        System.out.println("EigenVector 2 = " + eigLDVec2);
        System.out.println("EigenVector 3 = " + eigLDVec3);
```

```
        System.out.println("EigenVector 4 = " + eigLDVec4+"\n");

        // Generates and prints out all the characteristics for lateral-directional
        DUTCH-ROLL mode \\
        zeta_DR =
        DynamicStabilityCalculator.calcZeta(ldEigenvaluesMatrix[0][0],ldEigenvaluesMatrix[0][1
        ]);
        omega_n_DR =
        DynamicStabilityCalculator.calcOmega_n(ldEigenvaluesMatrix[0][0],ldEigenvaluesMatrix[0
        ][1]);
        period_DR =
        DynamicStabilityCalculator.calcT(ldEigenvaluesMatrix[0][0],ldEigenvaluesMatrix[0][1]);
        t_half_DR =
        DynamicStabilityCalculator.calct_half(ldEigenvaluesMatrix[0][0],ldEigenvaluesMatrix[0]
        [1]);
        N_half_DR =
        DynamicStabilityCalculator.calcN_half(ldEigenvaluesMatrix[0][0],ldEigenvaluesMatrix[0]
        [1]);


        System.out.println("_________________________________________________________________\
        n");
        System.out.println("DUTCH-ROLL MODE CHARACTERISTICS\n");
        System.out.println("Zeta_DR                          = "+df.format(zeta_DR)+"\n");
        System.out.println("Omega_n_DR                       = "+df.format(omega_n_DR)+"\n");
        System.out.println("Period                           = "+df.format(period_DR)+"\n");
        System.out.println("Halving Time                     = "+df.format(t_half_DR)+"\n");
        System.out.println("Number of cycles to Halving Time = "+df.format(N_half_DR)+"\n\n");

    }


    public void readDataFromExcelFile(File excelFile, int sheetNum) {

        // Formats numbers up to 4 decimal places
        DecimalFormat df = new DecimalFormat("#,###,##0.0000");

        try {
            System.out.println("Input file: " + excelFile.getAbsolutePath());
            FileInputStream fis = new FileInputStream(excelFile);
            Workbook wb = WorkbookFactory.create(fis);
            Sheet ws = wb.getSheetAt(sheetNum);
            int rowNum = ws.getLastRowNum() + 1;
            System.out.println("rows number: " + rowNum);

            if(sheetNum == 0){
                System.out.println("---------------------------------------------------\n");
                System.out.println("\n\n BOEING 747 /// Flight Condition (2) ");

                System.out.println("__________________________________________________________
                _______\n");
                System.out.println("DATA LIST: \n");
            }
            else if (sheetNum == 1){
                System.out.println("---------------------------------------------------\n");
                System.out.println("\n\n BOEING 747 /// Flight Condition (5) ");

                System.out.println("__________________________________________________________
                _______\n");
                System.out.println("DATA LIST: \n");
            }

            for (int i = 0 ; i < rowNum ; i++) {
                Row row = ws.getRow(i);
                int colNum = ws.getRow(0).getLastCellNum();
                for (int j = 0 ; j < colNum-2 ; j++) {
```

```
                    Cell cell = row.getCell(j);
                    String value = cellToString(cell);
                    switch (sheetNum){
                    ///////////// 1st sheet /////////////
                    case 0:
                        if ((i == 1) && (j == 1)) {
                            propulsion_system = Propulsion.valueOf(value);
                            switch (propulsion_system)
                            {
                            case CONSTANT_TRUST:
                                System.out.println(" PROPULSION SYSTEM: CONSTANT TRUST \n");
                                break;
                            case CONSTANT_POWER:
                                System.out.println(" PROPULSION SYSTEM: CONSTANT POWER \n");
                                break;
                            case CONSTANT_MASS_FLOW:
                                System.out.println(" PROPULSION SYSTEM: CONSTANT MASS FLOW
                                \n");
                                break;
                            case RAMJET:
                                System.out.println(" PROPULSION SYSTEM: RAMJET \n");
                                break;
                            default:
                                System.out.println(" PROPULSION SYSTEM: CONSTANT TRUST \n");
                                break;
                            }
                        }

                        if ((i == 2) && (j == 1)) {
                            rho0 = Double.parseDouble(value);
                            System.out.println(" rho0          = " + rho0);
                        }

                        if ((i == 3) && (j == 1)) {
                            surf = Double.parseDouble(value);
                            System.out.println(" surf          = " + surf);
                        }

                        if ((i == 4) && (j == 1)) {
                            mass = Double.parseDouble(value);
                            System.out.println(" mass          = " + mass);
                        }

                        if ((i == 5) && (j == 1)) {
                            cbar = Double.parseDouble(value);
                            System.out.println(" cbar          = " + cbar);
                        }

                        if ((i == 6) && (j == 1)) {
                            bbar = Double.parseDouble(value);
                            System.out.println(" bbar          = " + bbar  );
                        }

                        if ((i == 7) && (j == 1)) {
                            u0 = Double.parseDouble(value);
                            System.out.println(" u0            = " + u0);
                            q0 = StabilityDerivativesCalc.calcDynamicPressure(rho0, u0);
                            System.out.println(" q0            = " + df.format(q0));
                        }


                        if ((i == 8) && (j == 1)) {
                            m0 = Double.parseDouble(value);
                            System.out.println(" m0            = " + m0);
                        }

                        if ((i == 9) && (j == 1)) {
```

```
            gamma0 = Double.parseDouble(value);
            System.out.println(" gamma0        = " + gamma0);
        }

        if ((i == 10) && (j == 1)) {
            theta0_rad  = Double.parseDouble(value);
            System.out.println(" theta0_rad    = " + theta0_rad );
        }

        if ((i == 11) && (j == 1)) {
            iXX = Double.parseDouble(value);
            System.out.println(" iXX           = " + iXX);
        }

        if ((i == 12) && (j == 1)) {
            iYY = Double.parseDouble(value);
            System.out.println(" iYY           = " + iYY);
        }

        if ((i == 13) && (j == 1)) {
            iZZ = Double.parseDouble(value);
            System.out.println(" iZZ           = " + iZZ);
        }

        if ((i == 14) && (j == 1)) {
            iXZ = Double.parseDouble(value);
            System.out.println(" iXZ           = " + iXZ);
        }

        if ((i == 15) && (j == 1)) {
            cd0 = Double.parseDouble(value);
            System.out.println(" cd0           = " + cd0);
        }

        if ((i == 16) && (j == 1)) {
            cdAlpha0  = Double.parseDouble(value);
            System.out.println(" cdAlpha0      = " + cdAlpha0 );
        }

        if ((i == 17) && (j == 1)) {
            cdM0  = Double.parseDouble(value);
            System.out.println(" cdM0          = " + cdM0 );
        }

        if ((i == 18) && (j == 1)) {
            cl0 = Double.parseDouble(value);
            System.out.println(" cl0           = " + cl0);
        }

        if ((i == 19) && (j == 1)) {
            clAlpha0 = Double.parseDouble(value);
            System.out.println(" clAlpha0      = " + clAlpha0);
        }

        if ((i == 20) && (j == 1)) {
            clAlpha_dot0 = Double.parseDouble(value);
            System.out.println(" clAlpha_dot0 = " + clAlpha_dot0);
        }

        if ((i == 21) && (j == 1)) {
            clM0 = Double.parseDouble(value);
            System.out.println(" clM0          = " + clM0);
        }

        if ((i == 22) && (j == 1)) {
            clQ0 = Double.parseDouble(value);
            System.out.println(" clQ0          = " + clQ0);
```

```
            }

            if ((i == 23) && (j == 1)) {
                clDelta_T = Double.parseDouble(value);
                System.out.println(" clDelta_T     = " + clDelta_T);
            }

            if ((i == 24) && (j == 1)) {
                clDelta_E = Double.parseDouble(value);
                System.out.println(" clDelta_E     = " + clDelta_E);
            }

            if ((i == 25) && (j == 1)) {
                cMAlpha0 = Double.parseDouble(value);
                System.out.println(" cMAlpha0      = " + cMAlpha0);
            }

            if ((i == 26) && (j == 1)) {
                cMAlpha_dot0 = Double.parseDouble(value);
                System.out.println(" cMAlpha_dot0 = " + cMAlpha_dot0);
            }

            if ((i == 27) && (j == 1)) {
                cM_m0 = Double.parseDouble(value);
                System.out.println(" cM_m0         = " + cM_m0);
            }

            if ((i == 28) && (j == 1)) {
                cMq = Double.parseDouble(value);
                System.out.println(" cMq           = " + cMq);
            }

            if ((i == 29) && (j == 1)) {
                cMDelta_T = Double.parseDouble(value);
                System.out.println(" cMDelta_T     = " + cMDelta_T);
            }

            if ((i == 30) && (j == 1)) {
                cMDelta_E = Double.parseDouble(value);
                System.out.println(" cMDelta_E     = " + cMDelta_E);
            }

            if ((i == 31) && (j == 1)) {
                cTfix = Double.parseDouble(value);
                System.out.println(" cTfix         = " + cTfix);
            }

            if ((i == 32) && (j == 1)) {
                kv = Double.parseDouble(value);
                System.out.println(" kv            = " + kv);
            }

            if ((i == 33) && (j == 1)) {
                cyBeta = Double.parseDouble(value);
                System.out.println(" cyBeta        = " + cyBeta);
            }

            if ((i == 34) && (j == 1)) {
                cyP = Double.parseDouble(value);
                System.out.println(" cyP           = " + cyP);
            }

            if ((i == 35) && (j == 1)) {
                cyR = Double.parseDouble(value);
                System.out.println(" cyR           = " + cyR);
            }
```

```
        if ((i == 36) && (j == 1)) {
            cyDelta_A = Double.parseDouble(value);
            System.out.println(" cyDelta_A      = " + cyDelta_A);
        }

        if ((i == 37) && (j == 1)) {
            cyDelta_R = Double.parseDouble(value);
            System.out.println(" cyDelta_R      = " + cyDelta_R);
        }

        if ((i == 38) && (j == 1)) {
            cLBeta = Double.parseDouble(value);
            System.out.println(" cLBeta         = " + cLBeta);
        }

        if ((i == 39) && (j == 1)) {
            cLP = Double.parseDouble(value);
            System.out.println(" cLP            = " + cLP);
        }

        if ((i == 40) && (j == 1)) {
            cLR = Double.parseDouble(value);
            System.out.println(" cLR            = " + cLR);
        }

        if ((i == 41) && (j == 1)) {
            cLDelta_A = Double.parseDouble(value);
            System.out.println(" cLDelta_A      = " + cLDelta_A);
        }

        if ((i == 42) && (j == 1)) {
            cLDelta_R = Double.parseDouble(value);
            System.out.println(" cLDelta_R      = " + cLDelta_R);
        }

        if ((i == 43) && (j == 1)) {
            cNBeta = Double.parseDouble(value);
            System.out.println(" cNBeta         = " + cNBeta);
        }

        if ((i == 44) && (j == 1)) {
            cNP = Double.parseDouble(value);
            System.out.println(" cNP            = " + cNP);
        }

        if ((i == 45) && (j == 1)) {
            cNR = Double.parseDouble(value);
            System.out.println(" cNR            = " + cNR);
        }

        if ((i == 46) && (j == 1)) {
            cNDelta_A = Double.parseDouble(value);
            System.out.println(" cNDelta_A      = " + cNDelta_A);
        }

        if ((i == 47) && (j == 1)) {
            cNDelta_R = Double.parseDouble(value);
            System.out.println(" cNDelta_R      = " + cNDelta_R);
        }

        break;

        ///////////// 2nd sheet /////////////
    case 1:
        if ((i == 1) && (j == 1)) {
            propulsion_system = Propulsion.valueOf(value);
            switch (propulsion_system)
```

```
        {
        case CONSTANT_TRUST:
            System.out.println(" PROPULSION SYSTEM: CONSTANT TRUST \n");
            break;
        case CONSTANT_POWER:
            System.out.println(" PROPULSION SYSTEM: CONSTANT POWER \n");
            break;
        case CONSTANT_MASS_FLOW:
            System.out.println(" PROPULSION SYSTEM: CONSTANT MASS FLOW
            \n");
            break;
        case RAMJET:
            System.out.println(" PROPULSION SYSTEM: RAMJET \n");
            break;
        default:
            System.out.println(" PROPULSION SYSTEM: CONSTANT TRUST \n");
            break;
        }
    }

    if ((i == 2) && (j == 1)) {
        rho0 = Double.parseDouble(value);
        System.out.println(" rho0         = " + rho0);
    }

    if ((i == 3) && (j == 1)) {
        surf = Double.parseDouble(value);
        System.out.println(" surf         = " + surf);
    }

    if ((i == 4) && (j == 1)) {
        mass = Double.parseDouble(value);
        System.out.println(" mass         = " + mass);
    }

    if ((i == 5) && (j == 1)) {
        cbar = Double.parseDouble(value);
        System.out.println(" cbar         = " + cbar);
    }

    if ((i == 6) && (j == 1)) {
        bbar = Double.parseDouble(value);
        System.out.println(" bbar         = " + bbar  );
    }

    if ((i == 7) && (j == 1)) {
        u0 = Double.parseDouble(value);
        System.out.println(" u0           = " + u0);
        q0 = StabilityDerivativesCalc.calcDynamicPressure(rho0, u0);
        System.out.println(" q0           = " + df.format(q0));
    }


    if ((i == 8) && (j == 1)) {
        m0 = Double.parseDouble(value);
        System.out.println(" m0           = " + m0);
    }

    if ((i == 9) && (j == 1)) {
        gamma0 = Double.parseDouble(value);
        System.out.println(" gamma0       = " + gamma0);
    }

    if ((i == 10) && (j == 1)) {
        theta0_rad  = Double.parseDouble(value);
        System.out.println(" theta0_rad   = " + theta0_rad );
    }
```

```
if ((i == 11) && (j == 1)) {
    iXX = Double.parseDouble(value);
    System.out.println(" iXX          = " + iXX);
}

if ((i == 12) && (j == 1)) {
    iYY = Double.parseDouble(value);
    System.out.println(" iYY          = " + iYY);
}

if ((i == 13) && (j == 1)) {
    iZZ = Double.parseDouble(value);
    System.out.println(" iZZ          = " + iZZ);
}

if ((i == 14) && (j == 1)) {
    iXZ = Double.parseDouble(value);
    System.out.println(" iXZ          = " + iXZ);
}

if ((i == 15) && (j == 1)) {
    cd0 = Double.parseDouble(value);
    System.out.println(" cd0          = " + cd0);
}

if ((i == 16) && (j == 1)) {
    cdAlpha0  = Double.parseDouble(value);
    System.out.println(" cdAlpha0     = " + cdAlpha0 );
}

if ((i == 17) && (j == 1)) {
    cdM0  = Double.parseDouble(value);
    System.out.println(" cdM0         = " + cdM0 );
}

if ((i == 18) && (j == 1)) {
    cl0 = Double.parseDouble(value);
    System.out.println(" cl0          = " + cl0);
}

if ((i == 19) && (j == 1)) {
    clAlpha0 = Double.parseDouble(value);
    System.out.println(" clAlpha0     = " + clAlpha0);
}

if ((i == 20) && (j == 1)) {
    clAlpha_dot0 = Double.parseDouble(value);
    System.out.println(" clAlpha_dot0 = " + clAlpha_dot0);
}

if ((i == 21) && (j == 1)) {
    clM0 = Double.parseDouble(value);
    System.out.println(" clM0         = " + clM0);
}

if ((i == 22) && (j == 1)) {
    clQ0 = Double.parseDouble(value);
    System.out.println(" clQ0         = " + clQ0);
}

if ((i == 23) && (j == 1)) {
    clDelta_T = Double.parseDouble(value);
    System.out.println(" clDelta_T    = " + clDelta_T);
}

if ((i == 24) && (j == 1)) {
```

```
            clDelta_E = Double.parseDouble(value);
            System.out.println(" clDelta_E     = " + clDelta_E);
        }

        if ((i == 25) && (j == 1)) {
            cMAlpha0 = Double.parseDouble(value);
            System.out.println(" cMAlpha0      = " + cMAlpha0);
        }

        if ((i == 26) && (j == 1)) {
            cMAlpha_dot0 = Double.parseDouble(value);
            System.out.println(" cMAlpha_dot0 = " + cMAlpha_dot0);
        }

        if ((i == 27) && (j == 1)) {
            cM_m0 = Double.parseDouble(value);
            System.out.println(" cM_m0         = " + cM_m0);
        }

        if ((i == 28) && (j == 1)) {
            cMq = Double.parseDouble(value);
            System.out.println(" cMq           = " + cMq);
        }

        if ((i == 29) && (j == 1)) {
            cMDelta_T = Double.parseDouble(value);
            System.out.println(" cMDelta_T     = " + cMDelta_T);
        }

        if ((i == 30) && (j == 1)) {
            cMDelta_E = Double.parseDouble(value);
            System.out.println(" cMDelta_E     = " + cMDelta_E);
        }

        if ((i == 31) && (j == 1)) {
            cTfix = Double.parseDouble(value);
            System.out.println(" cTfix         = " + cTfix);
        }

        if ((i == 32) && (j == 1)) {
            kv = Double.parseDouble(value);
            System.out.println(" kv            = " + kv);
        }

        if ((i == 33) && (j == 1)) {
            cyBeta = Double.parseDouble(value);
            System.out.println(" cyBeta        = " + cyBeta);
        }

        if ((i == 34) && (j == 1)) {
            cyP = Double.parseDouble(value);
            System.out.println(" cyP           = " + cyP);
        }

        if ((i == 35) && (j == 1)) {
            cyR = Double.parseDouble(value);
            System.out.println(" cyR           = " + cyR);
        }

        if ((i == 36) && (j == 1)) {
            cyDelta_A = Double.parseDouble(value);
            System.out.println(" cyDelta_A     = " + cyDelta_A);
        }

        if ((i == 37) && (j == 1)) {
            cyDelta_R = Double.parseDouble(value);
            System.out.println(" cyDelta_R     = " + cyDelta_R);
```

```
                        }

                        if ((i == 38) && (j == 1)) {
                            cLBeta = Double.parseDouble(value);
                            System.out.println(" cLBeta         = " + cLBeta);
                        }

                        if ((i == 39) && (j == 1)) {
                            cLP = Double.parseDouble(value);
                            System.out.println(" cLP            = " + cLP);
                        }

                        if ((i == 40) && (j == 1)) {
                            cLR = Double.parseDouble(value);
                            System.out.println(" cLR            = " + cLR);
                        }

                        if ((i == 41) && (j == 1)) {
                            cLDelta_A = Double.parseDouble(value);
                            System.out.println(" cLDelta_A      = " + cLDelta_A);
                        }

                        if ((i == 42) && (j == 1)) {
                            cLDelta_R = Double.parseDouble(value);
                            System.out.println(" cLDelta_R      = " + cLDelta_R);
                        }

                        if ((i == 43) && (j == 1)) {
                            cNBeta = Double.parseDouble(value);
                            System.out.println(" cNBeta         = " + cNBeta);
                        }

                        if ((i == 44) && (j == 1)) {
                            cNP = Double.parseDouble(value);
                            System.out.println(" cNP            = " + cNP);
                        }

                        if ((i == 45) && (j == 1)) {
                            cNR = Double.parseDouble(value);
                            System.out.println(" cNR            = " + cNR);
                        }

                        if ((i == 46) && (j == 1)) {
                            cNDelta_A = Double.parseDouble(value);
                            System.out.println(" cNDelta_A      = " + cNDelta_A);
                        }

                        if ((i == 47) && (j == 1)) {
                            cNDelta_R = Double.parseDouble(value);
                            System.out.println(" cNDelta_R      = " + cNDelta_R);
                        }

                        break;
                    }

                }
            }
        }

        catch(Exception ioe) {
            ioe.printStackTrace();
        }


    }

    public static String cellToString(Cell cell) {
```

```
        int type;
        Object result = null;
        type = cell.getCellType();

        switch (type) {

        case Cell.CELL_TYPE_NUMERIC: // numeric value in Excel
        case Cell.CELL_TYPE_FORMULA: // precomputed value based on formula
            result = cell.getNumericCellValue();
            break;
        case Cell.CELL_TYPE_STRING: // String Value in Excel
            result = cell.getStringCellValue();
            break;
        case Cell.CELL_TYPE_BLANK:
            result = "";
        case Cell.CELL_TYPE_BOOLEAN: //boolean value
            result = cell.getBooleanCellValue();
            break;
        case Cell.CELL_TYPE_ERROR:
        default:
            throw new RuntimeException("There is no support for this type of
            cell");
        }

        if (result == null)
            return "";
        else
            return result.toString();
    }

    public static void main(String[] args) {


        FlightDynamicsManager theObj = new FlightDynamicsManager();


        System.out.println("---------------------------------------------------");
        System.out.println("Reading input data file (excel format)");
        String inputFileName = "AIRCRAFT_DATA.xlsx";
        File excelFile = new File (inputFileName) ;


        ////// select the excel sheet you want to read \\\\\\

                        int sheetNumber = 0;


        if (excelFile.exists()){

            System.out.println("File " + inputFileName + " found.");

            System.out.println("\n %%% start reading from file %%% ");

            // Read all data from file
            theObj.readDataFromExcelFile(excelFile, sheetNumber);

            System.out.println("\n %%% end of reading from file %%%");

            theObj.calculateAll();

        }
        else {
            System.out.println("File " + inputFileName + " not found.");
        }

    }
```

```
}
```