Assignments

Paper: Orbital mechanics MCD560

Name: Amit Mondal

Admission no. : 23DP0044

Prob. 1.solution of the universal Kepler's equation using Newton's method

Prob. 2. Calculation of the Lagrange coefficients f and g and their time derivatives

Prob .3. Gauss's method of preliminary orbit determination with iterative improvement

Prob. 4. calculation of the state vector of the spacecraft/planets for Bi-elliptic transfer

```c
#include <stdio.h>
#include <math.h>

#define EPSILON 1e-10 // Tolerance for convergence
#define MAX_ITER 1000 // Maximum number of iterations

// Function to calculate f(E)
double f(double E, double e, double M) {
    return E - e * sin(E) - M;
}

// Derivative of f(E)
double f_prime(double E, double e) {
    return 1 - e * cos(E);
}

// Newton's method for solving Kepler's equation
double kepler(double M, double e) {
    double E = M; // Initial guess

    for (int i = 0; i < MAX_ITER; i++) {
        double next_E = E - f(E, e, M) / f_prime(E, e);

        // Check for convergence
        if (fabs(next_E - E) < EPSILON)
            return next_E;

        E = next_E;
    }

    // If no convergence, return NaN
    return NAN;
}

int main() {
    double M, e;
    printf("Enter mean anomaly (M): ");
    scanf("%lf", &M);
    printf("Enter eccentricity (e): ");
    scanf("%lf", &e);

    // Call kepler function
    double E = kepler(M, e);

    if (!isnan(E)) {
        printf("Eccentric anomaly (E) = %lf\n", E);
    } else {
        printf("Unable to find solution.\n");
    }

    return 0;
}

```

Output:

```
Enter mean anomaly (M): 30
Enter eccentricity (e): 0.33
Eccentric anomaly (E) = 29.674777

--------------------------------
Process exited after 18.06 seconds with return value 0
Press any key to continue . . .
```

```c
#include <stdio.h>
#include <math.h>

#define PI 3.14159265358979323846

// Function to calculate the eccentric anomaly E using Newton's method
double eccentric_anomaly(double M, double e) {
    double E = M; // Initial guess

    for (int i = 0; i < 1000; i++) {
        double next_E = E - (E - e * sin(E) - M) / (1 - e * cos(E));

        // Check for convergence
        if (fabs(next_E - E) < 1e-10)
            return next_E;

        E = next_E;
    }

    // If no convergence, return NaN
    return NAN;
}

// Function to calculate f and g and their time derivatives
void lagrange_coefficients(double M, double e, double a, double mu, double *f, double *g, doub
    double E = eccentric_anomaly(M, e);
    double r = a * (1 - e * cos(E));
    double sqrt_a_mu = sqrt(a * mu);

    *f = 1 - r / a * cos(E);
    *g = r / sqrt_a_mu * sin(E);
    *fdot = -sqrt(mu / a) * sin(E) / (1 - e * cos(E));
    *gdot = sqrt(a / mu) * (cos(E) - e);
}

int main() {
    double M, e, a, mu;
    printf("Enter mean anomaly (M): ");
    scanf("%lf", &M);
    printf("Enter eccentricity (e): ");
    scanf("%lf", &e);
    printf("Enter semi-major axis (a): ");
    scanf("%lf", &a);
    printf("Enter gravitational parameter (mu): ");
    scanf("%lf", &mu);

    double f, g, fdot, gdot;
    lagrange_coefficients(M, e, a, mu, &f, &g, &fdot, &gdot);

    printf("f = %lf\n", f);
    printf("g = %lf\n", g);
    printf("fdot = %lf\n", fdot);
    printf("gdot = %lf\n", gdot);

    return 0;
```

```
}

```

Output

```
Enter mean anomaly (M): 30

Enter eccentricity (e): .35

Enter semi-major axis (a): 45

Enter gravitational parameter (mu): 1.33

f = 1.200124

g = -6.088773

fdot = 0.158445

gdot = -3.128146


--------------------------------

Process exited after 92.95 seconds with return value 0

Press any key to continue . . .
```

```c
#include <stdio.h>
#include <math.h>

#define N_OBSERVATIONS 3 // Number of observations
#define G 6.67430e-11    // Gravitational constant
#define M_EARTH 5.972e24 // Mass of the Earth

typedef struct {
    double t; // Time of observation
    double r[3]; // Position vector
} Observation;

double dot_product(double v1[], double v2[]) {
    double result = 0;
    for (int i = 0; i < 3; i++) {
        result += v1[i] * v2[i];
    }
    return result;
}

void cross_product(double v1[], double v2[], double result[]) {
    result[0] = v1[1] * v2[2] - v1[2] * v2[1];
    result[1] = v1[2] * v2[0] - v1[0] * v2[2];
    result[2] = v1[0] * v2[1] - v1[1] * v2[0];
}

void gauss_method(Observation obs[], double r[], double v[], double tol, int max_iter) {
    // Initial guess for position vector
    double r0[3];
    for (int i = 0; i < 3; i++) {
        r0[i] = r[i];
    }

    // Iterative improvement
    for (int iter = 0; iter < max_iter; iter++) {
        double delta_r[3] = {0};

        // Calculate delta_r
        for (int i = 0; i < N_OBSERVATIONS; i++) {
            double rho_i = sqrt(pow(obs[i].r[0] - r0[0], 2) + pow(obs[i].r[1] - r0[1], 2) + po
            double f = obs[i].t - dot_product(obs[i].r, r0) / rho_i;
            for (int j = 0; j < 3; j++) {
                delta_r[j] += f * (obs[i].r[j] / rho_i);
            }
        }

        // Update r0
        for (int i = 0; i < 3; i++) {
            r0[i] += delta_r[i];
        }

        // Check for convergence
        double delta_mag = sqrt(pow(delta_r[0], 2) + pow(delta_r[1], 2) + pow(delta_r[2], 2));
        if (delta_mag < tol) {
            break;
```

```
        }
    }

    // Final position vector
    for (int i = 0; i < 3; i++) {
        r[i] = r0[i];
    }
}

int main() {
    // Observations (time in seconds, position vector in meters)
    Observation obs[N_OBSERVATIONS] = {
        {0, {1500e3, 1500e3, 1000e3}},
        {3600, {-1500e3, -1500e3, -1000e3}},
        {7200, {0, 0, 0}}
    };

    // Initial guess for position and velocity vectors
    double r[3] = {1000e3, 1000e3, 1000e3};
    double v[3] = {0};

    // Set tolerance and maximum iterations for Gauss method
    double tol = 1e-9;
    int max_iter = 1000;

    // Perform Gauss method
    gauss_method(obs, r, v, tol, max_iter);

    // Output final position vector
    printf("Final position vector (meters): (%lf, %lf, %lf)\n", r[0], r[1], r[2]);

    return 0;
}

```

Output

Final position vector (meters): (23650954.748252, 23650954.748252, 16100636.498835)

```
--------------------------------
Process exited after 0.04542 seconds with return value 0
Press any key to continue . . .
```

```c
#include <stdio.h>
#include <math.h>

#define G 6.67430e-11    // Gravitational constant
#define M_EARTH 5.972e24 // Mass of the Earth

typedef struct {
    double r; // Distance from center of the Earth
    double v; // Velocity magnitude
    double theta; // Angle (in radians) with respect to the positive x-axis
} StateVector;

StateVector calculate_bi_elliptic_transfer(double r_initial, double r_final, double delta_v1,
    StateVector sv;

    // Calculate velocity at the initial orbit
    double v_initial = sqrt(G * M_EARTH * (2 / r_initial - 1 / (r_initial + r_final)));

    // Calculate state vector after the first burn
    double v1x = delta_v1 * cos(theta);
    double v1y = delta_v1 * sin(theta);
    sv.r = r_initial;
    sv.v = sqrt(v_initial * v_initial + delta_v1 * delta_v1);
    sv.theta = atan2(v1y, v_initial + v1x);

    // Calculate state vector after the second burn
    double v_final = sqrt(G * M_EARTH * (2 / r_final - 1 / (r_initial + r_final)));
    double v2x = delta_v2 * cos(M_PI - theta);
    double v2y = delta_v2 * sin(M_PI - theta);
    sv.r = r_final;
    sv.v = sqrt(v_final * v_final + delta_v2 * delta_v2);
    sv.theta = atan2(v2y, v_final + v2x);

    // Calculate state vector after the third burn
    double v3x = delta_v3 * cos(theta);
    double v3y = delta_v3 * sin(theta);
    sv.r = r_final;
    sv.v = sqrt(v_final * v_final + delta_v3 * delta_v3);
    sv.theta = atan2(v3y, v_final + v3x);

    return sv;
}

int main() {
    // Initial and final radii of the orbits (in meters)
    double r_initial = 200000; // Initial orbit radius (e.g., 200 km)
    double r_final = 400000;   // Final orbit radius (e.g., 400 km)

    // Delta v values for the three burns (in m/s)
    double delta_v1 = 200; // Delta v for first burn
    double delta_v2 = 400; // Delta v for second burn
    double delta_v3 = 200; // Delta v for third burn

    // Angle (in radians) between initial and final radii
    double theta = M_PI / 4; // Example angle of 45 degrees
```

```

    // Calculate state vector for bi-elliptic transfer
    StateVector sv = calculate_bi_elliptic_transfer(r_initial, r_final, delta_v1, delta_v2, de

    // Output state vector
    printf("State vector after bi-elliptic transfer:\n");
    printf("Radius: %.2f meters\n", sv.r);
    printf("Velocity: %.2f m/s\n", sv.v);
    printf("Angle (with respect to positive x-axis): %.2f radians\n", sv.theta);

    return 0;
}

```

Output

State vector after bi-elliptic transfer:

Radius: 400000.00 meters

Velocity: 36450.93 m/s

Angle (with respect to positive x-axis): 0.00 radians

Process exited after 0.04185 seconds with return value 0

Press any key to continue . . .