Загрузим данные, переведем градусы в радианы, отфильтруем по первому столбцу, что бы получить данные, когда самолет находился в покое, а так :
введем необхоимые константы и начальные условия:

In [16]:
```
A = load('./imu.dat');
B = load('./trj.dat');
A(:, 2:4) .*= (pi / 180);
B(:, 2:3) .*= (pi / 180);
B(:, 8:10) .*= (pi / 180);
A_init = A(A(:, 1) < 150, :);
R_earth = 6371000;
a = 6378137.0; b = 6356752.0; e2 = 6.6943799901413 * 10 ** -3;
delta_t = 0.01;
phi_0 = B(1, 2); lambda_0 = B(1, 3); h_0 = B(1, 4);
u = 2 * pi / 86164.090530833;
```

Реализуем функцию для подсчета g при заданном угле φ:

In [17]:
```
function [g] = g_phi(phi, h, a, e2)
    g = 9.78030 * (1 - 2 * h / a + 3 * e2 * sin(phi) ** 2 / 4);
endfunction
```

Реализуем функцию начальной выставки для получения матрицы ориентации $L(0)$:

In [18]:
```
function [L] = initial_calibration(A, phi, h, a, e2, u)
    L = zeros(3, 3);
    means = mean(A(:,2:end));
    L(:, 3) = means(4:end) / g_phi(phi, h, a, e2);
    L(:, 2) = (means(1:3) - (L(:, 3)' * u * sin(phi))) / (u * cos(phi));
    L(:, 1) = cross(L(:, 2), L(:, 3))';

    L(:, 3) /= norm(L(:, 3), 2);
    L(:, 2) /= norm(L(:, 2), 2);
    L(:, 1) /= norm(L(:, 1), 2);
endfunction
```

Получим матрицу $L(0)$ и посчитаем ее определитель:

In [19]:
```
L_0 = initial_calibration(A_init, phi_0, h_0, a, e2, u);
disp(L_0), disp(det(L_0))
```

```
   9.1843e-01  -3.9559e-01   3.1000e-08
   3.9559e-01   9.1843e-01   1.9332e-04
  -7.6502e-05   1.0950e-03   1.0000e+00
 1.00000
```

Реализуем функции необходимые для реализации итеративного процесса:

- matrix_multiplier() - функция для подсчета матричного множителя при численном интегрировании кинематических уравнений Пуассона:
$$E + \frac{sin(||\omega||\Delta t)}{||\omega||}\hat{\omega} + \frac{1 - cos(||\omega||\Delta t)}{||\omega||}\hat{\omega}^2$$
- create_matrix() - функция, которая по заданному ветору ω строит матрицу следующего вида:
$$\hat{\omega} = \begin{pmatrix} 0 & \omega_3 & -\omega_2 \\ -\omega_3 & 0 & \omega_1 \\ \omega_2 & -\omega_1 & 0 \end{pmatrix}$$
- u_x() - функция, которая по заданному углу φ строит вектор вида:
$$(0, u\cos\phi, u\sin\phi)^T$$
- g_x() - функция, которая по заданному углу φ и высоте h строит вектор вида:
$$(0, 0, -g)^T$$
- R_E() - функция, которая по заданному углу φ вычисляет значение вида:
$$\frac{a}{\sqrt{1 - e^2\sin^2\varphi}}$$
- R_N() - функция, которая по заданному углу φ вычисляет значение вида:
$$\frac{a(1 - e^2)}{\left(\sqrt{1 - e^2\sin^2\varphi}\right)^3}$$
- Omega() - функция, которая по заданному углу φ, вектору скорости V и высоте h строит вектор вида:

$$\left(\frac{-V_2}{R_N+h}, \frac{V_1}{R_E+h}, \frac{V_1 tg\varphi}{R_E+h}\right)^T$$

In [20]:
```
function [res] = matrix_multiplier(w, w_matrix, delta_t)
    res = (eye(3) + (sin(norm(w, 2) * delta_t) / norm(w, 2)) * w_matrix +
        ((1 - cos(norm(w, 2) * delta_t)) / (norm(w, 2) ** 2)) * (w_matrix ** 2));
endfunction

function [res] = create_matrix(w)
    res = [0, w(3), -w(2); -w(3), 0, w(1); w(2), -w(1), 0];
endfunction

function [res] = u_x(phi, u)
    res = [0; u * cos(phi); u * sin(phi)];
endfunction

function [res] = g_x(phi, h, a, e2)
    res = [0; 0; -g_phi(phi, h, a, e2)];
endfunction

function [res] = R_E(phi, a, e2)
    res = a / ((1 - e2 * (sin(phi) ** 2)) ** 0.5);
endfunction

function [res] = R_N(phi, a, e2)
    res = a * (1 - e2) / ((1 - e2 * (sin(phi) ** 2)) ** 1.5);
endfunction

function [res] = Omega(V, phi, h, a, e2)
    res = [-V(2) / (R_N(phi, a, e2) + h); V(1) / (R_E(phi, a, e2) + h); V(1) * tan(phi) / (R_E(phi, a, e2) + h)];
endfunction
```

Запишем так же формулы для пересчета необходимых величин:

$$A_z(t_{i+1}) = \left(E + \frac{sin(||\omega_z(t_i)||\Delta t)}{||\omega_z(t_i)||}\hat{\omega}_z(t_i) + \frac{1 - cos(||\omega_z(t_i)||\Delta t)}{||\omega_z(t_i)||}\hat{\omega}_z^2(t_i)\right) A_z(t_i)$$
$$A_x(t_{i+1}) = \left(E + \frac{sin(||\omega_x(t_i)||\Delta t)}{||\omega_x(t_i)||}\hat{\omega}_x(t_i) + \frac{1 - cos(||\omega_x(t_i)||\Delta t)}{||\omega_x(t_i)||}\hat{\omega}_x^2(t_i)\right) A_x(t_i) \text{ , где}$$
$$\omega_x(t_i) = \Omega_x(t_i) + u_x(t_i)$$
$$L(t_{i+1}) = A_z(t_i)A_x^T(t_i)$$
$$V(t_{i+1}) = V(t_i) + ((\hat{\Omega}_x(t_i) + \hat{u}_x(t_i))V(t_i) + g_x(t_i) + L(t_i)^T f_z(t_i))(t_{i+1} - t_i)$$
$$\varphi(t_{i+1}) = \varphi(t_i) + \frac{V_2(t_i)}{R_N(t_i) + h(t_i)}(t_{i+1} - t_i)$$
$$\lambda(t_{i+1}) = \lambda(t_i) + \frac{V_1(t_i)}{(R_E(t_i) + h(t_i))\cos\varphi(t_i)}(t_{i+1} - t_i)$$
$$h(t_{i+1}) = h(t_i) + V_3(t_i)(t_{i+1} - t_i)$$

И, наконец, запустим наш процесс, записывая координаты, полученные на каждой итерации, в массив:

In [47]:
```
L_0 = initial_calibration(A_init, phi_0, h_0, a, e2, u);
A_x_0 = eye(3);
A_z_0 = L_0;
phi_0 = B(1, 2); lambda_0 = B(1, 3); h_0 = B(1, 4);
V_0 = [0; 0; 0];

coordinates = [phi_0, lambda_0, h_0];

for row = A'
    t = row'(1);
    if mod(t, 100.0) == 0
        disp(t)
    endif
    w_z = row'(2:4)';
    w_z_matrix = create_matrix(w_z);
    A_z = matrix_multiplier(w_z, w_z_matrix, delta_t) * A_z_0;
    w_x = u_x(phi_0, u) + Omega(V_0, phi_0, h_0, a, e2);
    w_x_matrix = create_matrix(w_x);
    A_x = matrix_multiplier(w_x, w_x_matrix, delta_t) * A_x_0;
    L = A_z * A_x';
    f_z = row'(5:7)';
    V = (V_0 + ((create_matrix(Omega(V_0, phi_0, h_0, a, e2) + 2 * u_x(phi_0, u))) * V_0 +
        g_x(phi_0, h_0, a, e2) + L_0' * f_z) * delta_t);
    phi = phi_0 + V_0(2) / (R_N(phi_0, a, e2) + h_0) * delta_t;
    lambda = lambda_0 + V_0(1) / ((R_E(phi_0, a, e2) + h_0) * cos(phi_0)) * delta_t;
    h = h_0 + V_0(3) * delta_t;
    coordinates(end + 1, :) = [phi, lambda, h];
    A_z_0 = A_z; A_x_0 = A_x; L_0 = L; V_0 = V; phi_0 = phi; lambda_0 = lambda; h_0 = h;
endfor;
```

```
100
200
300
400
500
600
700
800
```

Напишем функцию преобразования координат в евклидовы:

In [48]:
```
function [x, y, z] = convert(coord, R_earth)
    x = (R_earth + coord(:, 3)).*cos(coord(:, 1)).*cos(coord(:, 2));
    y = (R_earth + coord(:, 3)).*cos(coord(:, 1)).*sin(coord(:, 2));
    z = (R_earth + coord(:, 3)).*sin(coord(:, 1));
endfunction
```

Преобразуем координаты и нарисуем результат:

In [50]:
```
[X11, X12, X13] = convert(B(:, 2:4), R_earth);
[X21, X22, X23] = convert(coordinates, R_earth);
plot3(X11, X12, X13, X21, X22, X23)
```