

f and g Series

The **closed** f and g expressions which were developed to express position and velocity vectors at some time t_o as linear combinations of position and velocity vectors at another time t may be replaced by f and g **series** expressions. Since

$$\mathbf{r} = f(\mathbf{r}_o, \mathbf{v}_o, t)\mathbf{r}_o + g(\mathbf{r}_o, \mathbf{v}_o, t)\mathbf{v}_o$$

we may expand \mathbf{r} in a Taylor series around $t = t_o$

$$\mathbf{r} = \sum_{n=0}^{\infty} \frac{(t - t_o)^n}{n!} \mathbf{r}_o^{(n)}$$

where

$$\mathbf{r}_o^{(n)} = \left. \frac{d^n \mathbf{r}}{dt^n} \right|_{t=t_o}$$

Since $\mathbf{r}^{(n)}$ is a vector in the $\mathbf{r} - \mathbf{v}$ plane it can be written as

$$\mathbf{r}^{(n)} = F_n \mathbf{r} + G_n \mathbf{v}$$

The next derivative is given by

$$\mathbf{r}^{(n+1)} = \dot{F}_n \mathbf{r} + F_n \mathbf{v} + \dot{G}_n \mathbf{v} + G_n \ddot{\mathbf{r}}$$

Since

$$\ddot{\mathbf{r}} = -u \mathbf{r}$$

where

$$u \equiv \frac{\mu}{r^3}$$

then

$$\mathbf{r}^{(n+1)} = (\dot{F}_n - uG_n) \mathbf{r} + (F_n + \dot{G}_n) \mathbf{v}$$

So there are recurrence relations given by

$$F_{n+1} = \dot{F}_n - uG_n$$

$$G_{n+1} = F_n + \dot{G}_n$$

When $n = 0$

$$\mathbf{r}^{(0)} = F_o \mathbf{r} + G_o \mathbf{v}$$

\Rightarrow

$$F_o = 1$$

$$G_o = 0$$

$$F_1 = \dot{F}_o - u G_o = 0$$

$$G_1 = F_o + \dot{G}_o = 1$$

$$F_2 = \dot{F}_1 - u G_1 = -u$$

$$G_2 = F_1 + \dot{G}_1 = 0$$

\vdots

To continue the development of the series coefficients we need expressions for the derivatives of u that can be used recursively. Let

$$u \equiv \frac{\mu}{r^3}$$

$$p \equiv \frac{1}{r^2}(\mathbf{r} \cdot \mathbf{v})$$

$$q \equiv \frac{1}{r^2}(\mathbf{v} \cdot \mathbf{v}) - u$$

Then since $\mathbf{r} \cdot \mathbf{v} = r \dot{r}$

$$\dot{u} = -3 \frac{\mu}{r^4} \dot{r} = -3 \frac{\mu}{r^3} \frac{r \dot{r}}{r^2} = -3up$$

$$\dot{p} = \frac{1}{r^2}(\mathbf{v} \cdot \mathbf{v}) + \frac{1}{r^2}(\mathbf{r} \cdot \ddot{\mathbf{r}}) - \frac{1}{r^3} \dot{r}(\mathbf{r} \cdot \mathbf{v}) = \frac{1}{r^2} v^2 - u - \frac{1}{r^4}(\mathbf{r} \cdot \mathbf{v})^2 = q - 2p^2$$

$$\dot{q} = \frac{2}{r^2}(\mathbf{v} \cdot \ddot{\mathbf{r}}) - \frac{2}{r^3} \dot{r}(\mathbf{v} \cdot \mathbf{v}) - \dot{u} = -\frac{2}{r^2} u (\mathbf{r} \cdot \mathbf{v}) - \frac{2}{r^4} (\mathbf{r} \cdot \mathbf{v})(\mathbf{v} \cdot \mathbf{v}) - \dot{u}$$

$$\dot{q} = -2p(u + q + u) + 3up = -p(u + 2q)$$

Continuing with the recursion formulas

$$F_3 = \dot{F}_2 - uG_2 = -\dot{u} = 3up$$

$$G_3 = F_2 + \dot{G}_2 = -u$$

$$\begin{aligned} F_4 &= \dot{F}_3 - uG_3 = 3\dot{u}p + 3u\dot{p} + u^2 \\ &= u(u - 15p^2 + 3q) \end{aligned}$$

$$G_4 = F_3 + \dot{G}_3 = 3up + 3up = 6up$$

\vdots

and so on until the desired number of terms is reached. A computer program to produce recursion formulas is desirable to prevent errors. These formulas are used to determine the f and g series.

$$\begin{aligned} \mathbf{r} &= \sum_{n=0}^{\infty} \frac{(t - t_o)^n}{n!} \mathbf{r}_o^{(n)} \\ &= \sum_{n=0}^{\infty} \frac{(t - t_o)^n}{n!} \left[F_n \mathbf{r} + G_n \mathbf{v} \right]_{t=t_o} \\ &= \left(\sum_{n=0}^{\infty} \frac{\tau^n}{n!} F_n \right) \mathbf{r}_o + \left(\sum_{n=0}^{\infty} \frac{\tau^n}{n!} F_n \right) \mathbf{v}_o \end{aligned}$$

where

$$\tau = t - t_o$$

then

$$f(\mathbf{r}_o, \mathbf{v}_o, t) = \sum_{n=0}^{\infty} \frac{\tau^n}{n!} [F_n]_{t=t_o}$$

$$g(\mathbf{r}_o, \mathbf{v}_o, t) = \sum_{n=0}^{\infty} \frac{\tau^n}{n!} [G_n]_{t=t_o}$$

The f and g series are expressed

$$f = 1 - \frac{1}{2}u_o\tau^2 + \frac{1}{2}u_op_o\tau^3 + \frac{1}{24}u_o(u_o - 15p_o^2 + 3q_o)\tau^4 + \dots$$

$$g = \tau - \frac{1}{6}u_o\tau^3 + \frac{1}{4}u_op_o\tau^4 + \dots$$

where u_o, p_o , and q_o are u, p , and q at $t = t_o$.

Prediction Problem

The prediction problem may be quickly solved using the f and g series if the time interval, τ , is not too large. Given the position, \mathbf{r}_1 , and velocity, \mathbf{v}_1 , vector at some time t_1 the position at a later time t_2 is given by

$$\mathbf{r}_2 = f(\mathbf{r}_1, \mathbf{v}_1, t_2 - t_1)\mathbf{r}_1 + g(\mathbf{r}_1, \mathbf{v}_1, t_2 - t_1)\mathbf{v}_1$$

Intercept Problem

If two positions are known this formula may be rearranged and solved to find the unknown velocity at $t = t_1$ and thus **determine the intercept orbit**

$$\mathbf{v}_1 = \frac{\mathbf{r}_2 - f(\mathbf{r}_1, \dot{\mathbf{r}}_1, \tau)\mathbf{r}_1}{g(\mathbf{r}_1, \dot{\mathbf{r}}_1, \tau)}$$

The method employed is to guess an initial value for $\mathbf{v}_1 = \dot{\mathbf{r}}_1$ to compute f and g . Then use the above equation to compute a new value for \mathbf{v}_1 and use this value to compute a new f and g . Continue this iteration until the vector \mathbf{v}_1 converges. The method converges quickly and there is the advantage of no ambiguity.

Orbit Determination

The f and g series gives a method to determine an initial orbit from optical sightings. Obtain line of sight unit vectors at three different observation times, t_1, t_2, t_3 by measuring right ascension α_i and declination δ_i . The line-of-sight unit vector from the observer to the object is given by

$$\mathbf{L}_i = \cos \delta_i \cos \alpha_i \mathbf{I} + \cos \delta_i \sin \alpha_i \mathbf{J} + \sin \delta_i \mathbf{K}$$

The fundamental vector triangle is

$$\mathbf{r} = \rho \mathbf{L} + \mathbf{R}$$

where \mathbf{r} is the vector from the center of mass to the object, \mathbf{R} is the vector from the center of mass to the observer, and ρ is the length of the vector from the observer to the object.

We expand the vector \mathbf{r} in terms of the f and g series evaluated at t_2

$$\mathbf{r} = f(\mathbf{r}_2, \mathbf{v}_2, t - t_2) \mathbf{r}_2 + g(\mathbf{r}_2, \mathbf{v}_2, t - t_2) \mathbf{v}_2$$

Since we can write

$$f_i = f(\mathbf{r}_2, \mathbf{v}_2, t_i - t_2) \text{ and } g_i = g(\mathbf{r}_2, \mathbf{v}_2, t_i - t_2)$$

there are nine equations in nine unknowns : $\mathbf{r}_2(3), \mathbf{v}_2(3), \rho_1, \rho_2, \rho_3$

$$f_1 \mathbf{r}_2 + g_1 \mathbf{v}_2 = \rho_1 \mathbf{L}_1 + \mathbf{R}_1$$

$$\mathbf{r}_2 = \rho_2 \mathbf{L}_2 + \mathbf{R}_2$$

$$f_3 \mathbf{r}_2 + g_3 \mathbf{v}_2 = \rho_3 \mathbf{L}_3 + \mathbf{R}_3$$

We can eliminate the ρ_i by cross multiplying the i^{th} equation by \mathbf{L}_i .

$$f_1 \mathbf{L}_1 \times \mathbf{r}_2 + g_1 \mathbf{L}_1 \times \mathbf{v}_2 = \mathbf{L}_1 \times \mathbf{R}_1$$

$$\mathbf{L}_2 \times \mathbf{r}_2 = \mathbf{L}_2 \times \mathbf{R}_2$$

$$f_3 \mathbf{L}_3 \times \mathbf{r}_2 + g_3 \mathbf{L}_3 \times \mathbf{v}_2 = \mathbf{L}_3 \times \mathbf{R}_3$$

Since there are now only six equations we may expand the cross products and drop the \mathbf{K} components to have six equations in six unknowns. This equation matrix may be economically solved by a Gauss-Jordan scheme.

$$\begin{aligned} f_1 L_{1z}x - f_1 L_{1x}z + g_1 L_{1z}\dot{x} - g_1 L_{1x}\dot{z} &= R_{1x}L_{1z} - R_{1z}L_{1x} \\ -f_1 L_{1z}y + f_1 L_{1y}z - g_1 L_{1z}\dot{y} + g_1 L_{1y}\dot{z} &= R_{1z}L_{1y} - R_{1y}L_{1z} \\ L_{2z}x - L_{2x}z &= R_{2x}L_{2z} - R_{2z}L_{2x} \\ -L_{2z}y + L_{2y}z &= R_{2z}L_{2y} - R_{2y}L_{2z} \\ f_3 L_{3z}x - f_3 L_{3x}z + g_3 L_{3z}\dot{x} - g_3 L_{3x}\dot{z} &= R_{3x}L_{3z} - R_{3z}L_{3x} \\ -f_3 L_{3z}y + f_3 L_{3y}z - g_3 L_{3z}\dot{y} + g_3 L_{3y}\dot{z} &= R_{3z}L_{3y} - R_{3y}L_{3z} \end{aligned}$$

The iterative procedure is as follows :

1. Known values are $\mathbf{L}_1, \mathbf{L}_2, \mathbf{L}_3, \mathbf{R}_1, \mathbf{R}_2, \mathbf{R}_3$
2. Estimate the magnitude of \mathbf{r}_2 .
3. Compute $u_2 = \frac{\mu}{\mathbf{r}_2^3}$
4. Compute f_1, g_1, f_3, g_3 using f and g terms independent of p_2 and q_2
5. Solve the six equations for the unknowns $x, y, z, \dot{x}, \dot{y}, \dot{z}$ (\mathbf{r}_2 and \mathbf{v}_2)
6. Compute new values of u_2, p_2, q_2 from \mathbf{r}_2 and \mathbf{v}_2
7. Compute f_1, g_1, f_3, g_3 to the desired accuracy
8. Loop back to step #5 and continue until the process converges to the correct values of \mathbf{r}_2 and \mathbf{v}_2 .

This method will converge with no ambiguity if the time between observations is not too large. It also has an advantage over the method of Laplace in that the observer may lie in the plane of the orbit. The method does have some divergent behavior as the time intervals approach the

limits. This undesirable behavior may be damped by allowing only a small percentage of change from one set of \mathbf{r}_2 and \mathbf{v}_2 to the next.

This method was found in the reference:

Fundamentals of Astrodynamics, Bate, Mueller, White

Here is a routine in C/C++ to calculate an eighth order f and g series, fr and gr , given a position and velocity vector, $rr1$ and $vv1$, and a time, $delt$, advanced from the epoch of $rr1$ and $vv1$.

// F and G series

```
void f8g(double rr1[], double vv1[], double delt, double& fr, double& gr)
{
    double r, u, p, q, p2, p4, q2, u2;
    double f[9], g[9];
    r = norm(rr1);
    u = 1 / (r * r * r);
    p = dot(rr1, vv1) / (r * r);
    q = dot(vv1, vv1) / (r * r) - u;
    p2 = p * p;
    p4 = p2 * p2;
    q2 = q * q;
    u2 = u * u;
    f[0] = 1;
    f[1] = 0;
    f[2] = -u / 2;
    f[3] = p * u / 2;
    f[4] = u * (u - 3 * (5 * p2 - q)) / 24;
    f[5] = -p * u * (u - 7 * p2 + 3 * q) / 8;
    f[6] = -u * (u2 - 6 * (35 * p2 - 4 * q) * u
        + 45 * (21 * p4 - 14 * p2 * q + q2)) / 720;
    f[7] = p * u * (u2 - 2 * (25 * p2 - 7 * q) * u
        + 5 * (33 * p4 - 30 * p2 * q + 5 * q2)) / 80;
    f[8] = u * (u2 * u - 9 * (245 * p2 - 13 * q) * u2
        + 27 * (1925 * p4 - 910 * p2 * q + 41 * q2) * u
        - 315 * (429 * p4 * p2 - 495 * p4 * q
        + 135 * p2 * q2 - 5 * q2 * q)) / 40320;
    g[0] = 0;
    g[1] = 1;
    g[2] = 0;
    g[3] = -u / 6;
    g[4] = p * u / 4;
    g[5] = u * (u - 9 * (5 * p2 - q)) / 120;
```

```

g[6]= - p * u * (u - 2 * (7 * p2 - 3 * q)) / 24;
g[7]= - u * (u2 - 18 * (35 * p2 - 3 * q) * u
      + 225 * (21 * p4 - 14 * p2 * q + q2)) / 5040;
g[8]= p * u * (u2 - 4 * (25 * p2 - 6 * q) * u
      + 15 * (33 * p4 - 30 * p2 * q + 5 * q2)) / 320;
fr = 1 + delt * delt * (f[2] + delt * (f[3] + delt * (f[4]
      + delt * (f[5] + delt * (f[6] + delt * (f[7] + delt * f[8])))))));
gr = delt * (1 + delt * delt * (g[3] + delt * (g[4] + delt * (g[5]
      + delt * (g[6] + delt * (g[7] + delt * g[8])))))));
}

```

It will also be necessary to perform a Gaussian Elimination on the equation matrix to solve for the components of the position and velocity vectors at t_2 . I found an excellent algorithm in a numerical analysis text for scaled partial pivoting. The following code is my implementation of the algorithm in C++. Note: the number 7 is the column dimension for this particular application. The code can be adjusted to have this dimension declared as a **constant int** col = 7; and all the other numbers in the algorithm could be derived from that constant. (i.e. replace every 6,5,4 by col - 1, col - 2, col - 3).

```

// Gaussian elimination
void rref(double m[][7], double b[]) // scaled partial pivoting
{
    int i,j,k, ROW;
    double s[6], bin, mult;

    // calculate scale factors

    for(i = 0; i < 6; i++)
    {
        s[i] = abs(m[i][0]);
        for(j = 1; j < 6; j++)
            if(s[i] < abs(m[i][j]))
            {
                s[i] = abs(m[i][j]);
            }
    } // end for i

    // swap rows according to scale

    for(j = 0; j < 5; j++)
    {
        ROW = j;
        for(i = j + 1; i < 6; i++)

```



```

{
    if(abs(m[ROW][j] / s[ROW]) < abs(m[i][j] / s[i]))
        ROW = i;
}
if(ROW != j)
{
    for(k = j; k < 7; k++)    // swap rows
    {
        bin = m[j][k];
        m[j][k] = m[ROW][k];
        m[ROW][k] = bin;
    }
    bin = s[j];                // swap scales
    s[j] = s[ROW];
    s[ROW] = bin;
} // end if

// forward elimination

for(i = j + 1; i < 6; i++)
{
    mult = m[i][j] / m[j][j];
    for(k = j + 1; k < 7; k++)
    {
        m[i][k] = m[i][k] - mult * m[j][k];
    }
    m[i][j] = 0;
} // end for i
} // end for j

// test for singular matrix

bin = 1;
for(i = 0; i < 6; i++)
{
    bin *= m[i][i];
}
if(bin == 0)
{
    printf("Singular matrix");
    exit(0);
}

```

```
// back substitution
```

```
b[5] = m[5][6] / m[5][5];  
for( $i = 4$ ;  $i \geq 0$ ;  $i--$ )  
{  
    bin = 0;  
    for( $k = i + 1$ ;  $k < 6$ ;  $k++$ )  
        bin = bin + m[i][k] * b[k];  
    b[i] = (m[i][6] - bin) / m[i][i];  
}  
} // end rref
```