Fast Computation of Values and Derivatives of Rational Cubic Polynomials

July 24, 2015

1 Background and Notation

Given an image and a set of n points within the image (tie points) that we know the true spatial locations of, we would like to fit a ratio of cubic polynomials that model the relationship between the tie points and their true spatial locations. Let the location of the ith tie point in an image be parameterized by the ordered pair (r_i, s_i) and let the true spatial location of the tie point be (x_i, y_i, z_i). Then we would like to fit functions f_r and f_s that are ratios of cubic polynomials of x, y, and z, i.e. we would like for all $i \leq n$

$$f_r(x_i, y_i, z_i) \approx r_i, \quad f_s(x_i, y_i, z_i) \approx s_i.$$

if we overload our notation so that x,y, z, s, and r are vectors in $\mathbf{R}^n$ then we can write

$$f_r(x, y, z) = r, \quad f_s(x, y, z) = s.$$

f_r and f_i are ratios of cubic polynomials, which we will write as:

$$f_r(x, y, z) = \frac{p_1(x, y, z)}{p_2(x, y, z)}, \quad f_s(x, y, z) = \frac{p_3(x, y, z)}{p_4(x, y, z)}$$

where each $p_i(x, y, z)$ denotes a (vector) cubic polynomial of x,y, and z, i.e.

$$\begin{aligned} p_1(x, y, z) = & a_1 + \\ & a_2 x + a_3 y + a_4 z + \\ & a_5 x^2 + a_6 xy + a_7 xz + a_8 y^2 + a_9 yz + a_{10} z^2 + \\ & a_{11} xyz + a_{12} x^2 y + a_{13} x^2 z + a_{14} x^3 + a_{15} y^2 x + \\ & a_{16} y^2 z + a_{17} y^3 + a_{18} z^2 x + a_{19} z^2 y + a_{20} z^3. \end{aligned}$$

where all multiplication and power-taking is elementwise on the vectors. So we denote the 20-vector of coefficients for p_1 as a, and the vectors of coefficients for p_2, p_3, and p_4 as b, c, and d respectively.

2 Problem

We would like to fit the two functions f_s and f_r, i.e. we solve the problem

$$\min_{a,b} \quad (1/2)\|f_r(x, y, z) - r\|_2^2 \tag{1}$$

and similarly for f_s

$$\min_{c,d} \quad (1/2)\|f_s(x,y,z) - s\|_2^2. \tag{2}$$

a, b, c, d enter in through f_r and f_s although they are not listed as parameters of the functions. *Note:* This is kind of backwards, because x, y and z should be thought about as *given* since they are the data of our problem and f_r and f_s should be thought about as functions of the coefficients instead of the data.

These are non-linear, non-convex least squares problems, and cannot be solved with traditional conved optimization methods.

3 Evaluating the Functions

To solve (1) and (2), most COTS solvers need to evaluate the functions f_r and f_s and their derivatives many times. This is exacerbated by the fact that some solvers compute derivatives numerically, which can require several function evaluations to compute each requested derivative. Thus the final speed of a solver is heavily dependent on the speed of evaluating the functions and their derivatives. We now describe a fast method for this.

Let $P \in \mathbf{R}^{n\times 20}$ be a matrix that is the cubic polynomial expansion of the vectors x, y, and z. More concretely,

$$P = [1, x, y, z, x^2, xy, xz, y^2, yz, z^2, xyz, x^2y, x^2z, x^3, y^2z, y^2z, y^3, z^2x, z^2y, z^3]$$

where all multiplication and power-taking is elementwise on the vectors. This allows us to write

$$p_1(x,y,z) = Pa, \quad p_2(x,y,z) = Pb, \quad p_3(x,y,z) = Pc, \quad p_4(x,y,z)Pd$$

and thus

$$f_r = Pa/Pb, \quad f_s = Pc/Pd$$

where the division is elementwise. Note that the matrix P need only be computed once. Here is some matlab code:

Listing 1: cubic_poly.m

```
function [ P ] = cubic_poly(x,y,z)
%CUBIC_POLY computes a matrix of all cubic combinations
%of the three supplied vectors
    P = [ones(size(x)) x y z x.*y x.*z y.*z x.^2 y.^2 z.^2 ...
        (x.*y).*z (x.^2).*y (x.^2).*z x.^3 (y.^2).*x (y.^2).*z ...
        y.^3 (z.^2).*x (z.^2).*y z.^3];
end
```

Listing 2: rp.m

```
function [f] = rp(x,y,z,a,b)
    P = cubic_poly(x,y,z);
    p1 = P*a;
    p2 = P*b;
    f = (p1./p2);
end
```

We would actually like to change this code subtly. To normalize the coefficients, we fix the first coefficient in the denominator to be 1, i.e. $b_1 = 1$ and $d_1 = 1$. Furthermore instead of simply evaluating the rational polynomial, we would like to evaluate the loss functions $f_r - r$ and $f_s - s$. So let us modify b and d so that $b, d \in \mathbf{R}^{19}$. Then the matlab becomes:

Listing 3: A better rp.m

```
function [f] = rp(x,y,z,a,b,r)
    P = cubic_poly(x,y,z);
    b = [1; b];
    p1 = P*a;
    p2 = P*b;
    f = (p1./p2) - r ;
end
```

This vectorized formulation can be accelerated with BLAS or GPU linear algebra routines. Furthermore the matrix P can be computed once and stored, so that every function evaluation requires only two matrix-vector multiplies and one elementwise vector division.

4 Evaluating Derivatives

We would also like a fast way to evalute the Jacobian of f_r and f_s. First we compute them analytically. Let f be a vector rational cubic polynomial that depends on data x, y, z and coefficients a, b. Then

$$\frac{\partial f_i}{\partial a_j} = P_{ij} / P_j b$$

where P_j is the jth row of P as a row vector. This suggests a quick way to compute the derivatives of each f_i with respect to each a_j:

$$J_a(f) = P / (Pb\mathbf{1}^T)$$

where J_a is the Jacobian with respect to the coefficients a, the division is elementwise matrix division, and $\mathbf{1}$ is a vector of 20 ones.

We can do something similar for the derivatives with respect to the coefficients in the denominator using the chain rule.

$$\frac{\partial f_i}{\partial b_j} = -\frac{P_i a}{(P_i b)^2} \left(\frac{\partial P_i b}{\partial b_j} \right) = -P_{ij} \frac{P_i a}{(P_i b)^2}$$

P_i is the ith row of P as a row vector. We can thus compute the Jacobian with respect to b as

$$J_b(f) = -P \left(\frac{Pa}{(Pb)^2} \mathbf{1}^T \right)$$

where the division is elementwise vector division, and the multiplication by P is elementwise matrix multiplication.

Here is matlab code that implements this:

Listing 4: rp_jacobian.m

```
function [J] = rp_jacobian(x,y,z,a,b)
```

```
%RP_JACOBIAN Calculates jacobian of a rational polynomial
    P = cubic_poly(x,y,z);
    b = [1;b];
    p1 = P*a;
    p2 = P*b;
    Ja = P./(p2*ones(1,20));
    Jb = P(:,2:20).*((-p1./((p2).^2))*ones(1,19));
    J=[J1 J2];
end
```

5 Additional Considerations

Since many times the Jacobian and function value will both be needed, it is best to combine their calculations of P and the polynomials. The following matlab code impliments this:

Listing 5: Final rp.m

```
function [f,J] = rp(x,y,z,s,a,b)
%RP Calculates value and jacobian of a rational polynomial
    P = cubic_poly(x,y,z);
    b = [1;b];
    p1 = P*a;
    p2 = P*b;
    f = (p1./p2)-s;
    if nargout > 1
        J1 = P./(p2*ones(1,20));
        J2 = P(:,2:20).*((-p1./((p2).^2))*ones(1,19));
        J=[J1 J2];
    end
end
```

This code uses `nargout` to check if the Jacobian is requested.