Practical Advice for Non-linear Least Squares Problems

Niclas Börlin

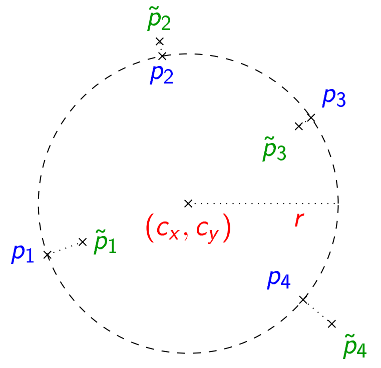
5DA001 Non-linear Optimization

Function call structure

Problem independent	Problem specific
	1 Calculate a starting approximation x_0.
2 Repeat for $k = 0, 1, \ldots$	
	3 Calculate residual $r(x_k)$ and Jacobian $J(x_k)$.
4 Calculate termination criteria.	
5 Calculate search direction p_k.	
6 Calculate step length α_k.	**6** Compute $f(x_k + \alpha_k p_k)$.
7 Calculate $x_{k+1} = x_k + \alpha_k p_k$.	

Toy problem: Circle fitting

Given a number of points $\tilde{p}_i = [\tilde{x}_i, \tilde{y}_i]^T$, $i = 1, 2, \ldots, m$, "find the circle that fits the points best in the least squares sense".



Objective function

Step 1: Decide what to minimize.

Minimize the squared sum of Euclidean distances between each measured point $\tilde{p}_i$ and the closest point p_i on the circle, i.e.

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} \|p_i - \tilde{p}_i\|^2.$$

Models and parameters

Step 2: Formulate a mathematical model of the object and determine the unknowns.

Step 2a: Formulate a *local model* for each "term" of the least squares sum.

A point (x_i, y_i) on a circle with center (c_x, c_y) and radius r can be modelled as

$$G_i(x) = \begin{bmatrix} x_i \\ y_i \end{bmatrix} = \begin{bmatrix} c_x \\ c_y \end{bmatrix} + r \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}$$

for some "phase angle" θ_i.

Step 2b: Formulate a *global model* describing a vector with all terms of the least squares sum.

$$G(x) = \begin{bmatrix} G_1(x) \\ G_2(x) \\ \vdots \\ G_m(x) \end{bmatrix}.$$

Models and parameters

Step 2c: Determine which parameters to put in the vector of unknowns, and in what order.

$$x = \begin{bmatrix} c_x \\ c_y \\ r \\ \theta_1 \\ \dots \\ \theta_m \end{bmatrix},$$

where c_x, c_y, r are the "global" parameters, corresponding to the circle we wish to find and the θ_i are "local" parameters, corresponding to one point each.

Step 2d: Implement the model function.

Step 2e: **Verify the model and implementation by calculating $G(x)$ for realistic values of x.**

The residual

Step 3a: Implement the residual $r(x)$ as "model minus data".

$$r(x; d) = G(x) - d, \text{ where } d = \begin{bmatrix} \tilde{p}_1 \\ \tilde{p}_2 \\ \vdots \\ \tilde{p}_m \end{bmatrix} \text{ contain the "measurements".}$$

Step 3b: **Reality check. Verify that $r(x^*; G(x^*)) = 0$.**

The Jacobian

Step 4a: Derive an analytical expression for the Jacobian. Use symbolic tools, e.g. Maple, if necessary.

Step 4b: Implement the Jacobian.

$$J(x; d) = \begin{bmatrix} 1 & 0 & \cos\theta_1 & -r\sin\theta_1 & & \\ 0 & 1 & \sin\theta_1 & r\cos\theta_1 & & \\ \vdots & \vdots & \vdots & & \ddots & \\ 1 & 0 & \cos\theta_m & & & -r\sin\theta_m \\ 0 & 1 & \sin\theta_m & & & r\cos\theta_m \end{bmatrix}.$$

Step 4c: **Compare the implemented Jacobian with a numerical approximation.**

$$J(x; d) \approx \begin{bmatrix} \frac{r(x+\epsilon_1)-r(x-\epsilon_1)}{2\epsilon} & \dots & \frac{r(x+\epsilon_n)-r(x-\epsilon_n)}{2\epsilon} \end{bmatrix},$$
$$\epsilon_1 = \begin{bmatrix} \epsilon & 0 & \dots & 0 \end{bmatrix}^T, \ \epsilon_n = \begin{bmatrix} 0 & \dots & 0 & \epsilon \end{bmatrix}^T.$$

Step 4d: **Verify that $J(x)$ has full rank for a general x.**

Convergence check on synthetic, error-free data

Step 5: Tests on perfect data. Use a realistic x^* and generate error-free "measurements", i.e. $d = G(x^*)$.

Step 5a: **Call the optimization method with $x_0 = x^*$ and verify that it returns x^* as the solution after maximum 1 iteration.**

Step 5b: **Generate starting approximations x_0 as pertubations of the true solution x^*. Verify convergence from a reasonable large region.**

Pertubation sensitivity of solution

Step 5: Tests on data with known errors.

Step 5a: Use a realistic x^* and generate "measurements" with an added measurement error, i.e.

$$d = G(x^*) + \varepsilon,\ \varepsilon \in N(0, \sigma^2).$$

where σ^2 is chosen to give errors of a reasonable size.

Step 5b: Solve the optimization problem with $x_0 = x^*$. Call the solution to the new problem $\hat{x}$.

Step 5c: Compare the solution of the perturbed problem $\hat{x}$ with the solution of the original problem x^*.

Step 5d: Repeat steps 5a–5c and analyze the deviation of $\hat{x}$ from x^*.

Step 5e: Repeat steps 5a-5d with $x_0 \neq x^*$, i.e. perturb the starting approximation also.

Starting approximation calculation

Step 6a: Construct a function for the starting approximation that (optimally) relies only on measurements.

$$\boxed{c_x^0 = \overline{\tilde{x}_i},\ c_y^0 = \overline{\tilde{y}_i},\ r_0 = \sqrt{\overline{(\tilde{x}_i - c_x^0)^2 + (\tilde{y}_i - c_y^0)^2}},\ \theta_{0i} = \tan^{-1}\frac{\tilde{y}_i - c_y^0}{\tilde{x}_i - c_x^0}}$$

Step 6b: Check the quality of the starting approximating function on error-free data.

Step 6b.1: Generate realistic measurements d without any errors.

Step 6b.2: Use the function in Step 6a to determine x_0 from d and solve the optimization problem. Call the solution $\hat{x}$.

Step 6b.3: Compare $\hat{x}$ with x^*.

Step 6c: Repeat steps 6b.1 to 6b.3 on data with realistic errors.

Sensitivity analysis

	"Truth"		"Real world"
1	Start with a "true" parameter vector $x^* = [c_x, c_y, r, \theta_1, \ldots, \theta_m]$.		
2	Calculate points p_i on the circle.		
3	Generate simulated measurements $\tilde{p}_i = p_i + \epsilon_i,\ \epsilon_i \in N(0, \sigma^2)$.		
		4	Construct a starting approximation x_0 from the measurements.
		5	Solve the parameter estimation problem. Call the solution $\hat{x}$.
		6a	Study the deviation of $\hat{x}$ for repeated simulations. Determine precision, repeatability.
6b	Compare the true parameter vector x^* with the estimated $\hat{x}$. Determine accuracy.		

Suggest code structure - model function

```
function pts=circle_g(c,r,theta)
%CIRCLE_G Circle example model function.
%
%pts=circle_G(c,r,theta)
%c     - center of circle.
%r     - circle radius.
%theta - angle parameter for points i on circle, i=1,2,...,m.
%pts   - points on the circle.

% Number of points.
m=length(theta);

if (slow_but_readable)
    % Preallocate result to avoid memory fragmentation and improve speed.
    pts=zeros(2,m);
    for i=1:m
        % Calculate position of each point on the circle.
        pts(:,i)=c+r*[cos(theta(i));sin(theta(i))];
    end
else % faster
    pts=repmat(c,1,m)+r*[cos(theta);sin(theta)];
end
```

Suggest code structure - residual/jacobian function

```
function [r,J,JJ]=circle_r(x,m,b)
%CIRCLE_R Circle example residual/jacobian function.
%
%[r,J,JJ]=circle_f(x,m,b)
%x  - parameter vector [cx,cy,r,theta1,...,thetam].
%     cx,cy  - center of circle.
%     r      - circle radius.
%     thetai - parameter for point i on circle, i=1,2,...,m.
%m  - number of points.
%b  - 2xm matrix with measured points.
%r  - residual function G(x)-d.
%J  - jacobian of f w.r.t. x.
%JJ - numerical approximation of J.

% Unpack parameter vector.
base=1;
[ixc,base]=pindex(2,base); % indices for c elements
[ixr,base]=pindex(1,base); % index for r
[ixt,base]=pindex(m,base); % indices for theta elements

c=reshape(x(ixc),2,1);
r=reshape(x(ixr),1,1);
theta=reshape(x(ixt),m,1);
```

```
pts=circle_g(c,r,theta); % Call model function.
f=pts(:)-b(:);           % Unroll vector and subtract data.

if (nargout>1) % Want Jacobian too.
    % Slow but readable
    % Preallocate result to avoid memory fragmentation and improve speed.
    J=zeros(2*m,m+3);
    for i=1:m
        rows=(i-1)*2+[1:2]; % Rows in J corresponding to point m.

        % Calculate partial derivatives corresponding to each point.
        J(rows(1),1)=1;                     % dFx/dcx
                                            % dFx/dcy=0
        J(rows(1),3)=cos(theta(i));         % dFx/dr
        J(rows(1),3+i)=r*(-sin(theta(i))); % dFx/dthetai=0 except for i=k.

                                            % dFy/dcx=0
        J(rows(2),2)=1;                     % dFy/dcy
        J(rows(2),3)=sin(theta(i));         % dFy/dr
        J(rows(2),3+i)=r*cos(theta(i));     % dFy/dthetai=0 except for i=k.
    end
end

if (nargout>2) % Want numerical approximation, too.
    JJ=jacapprox(mfilename,x,1e-6,{m,b});
end
```