VIO 第三章作业

曾卓

05 2021

1 参数估计

1.1 请绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图

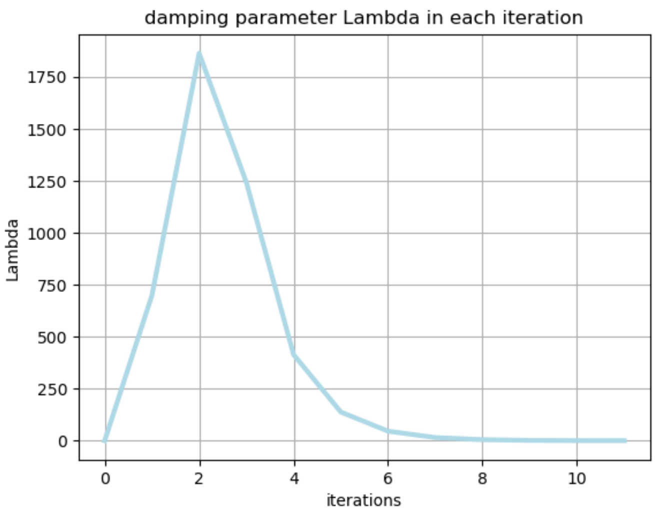

图 1: 阻尼因子 μ 随着迭代变化的曲线图

1.2 将曲线函数改成 $y = ax^2 + bx + c$，请修改样例代码中残差计算，雅克比计算等函数，完成曲线参数估计

修改残差项和雅可比即可

residual_(0) = abc(0)*x_*x_ + abc(1)*x_ + abc(2) - y_;

jaco_abc « x_ * x_, x_, 1;

```
Test CurveFitting start...
iter: 0 , chi= 61493.7 , Lambda= 0.001
iter: 1 , chi= 91.3952 , Lambda= 0.000333333
iter: 2 , chi= 91.395 , Lambda= 0.000222222
problem solve cost: 0.157363 ms
   makeHessian cost: 0.065848 ms
-------After optimization, we got these parameters :
10.6107 19.6183 9.99517
-------ground truth:
10 20 10
```

图 2: 参数估计结果

1.3 实现其他更优秀的阻尼因子策略，并给出实验对比

对 $y = e^{(ax^2+bx+c)}$ 进行拟合

原代码采用的是类似 Nielsen 的方法，但是对 alpha 进行了一定的限制 (<=2/3)，避免其过大

参数拟合结果如下：

```
Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 30015.5 , Lambda= 699.051
iter: 2 , chi= 13421.2 , Lambda= 1864.14
iter: 3 , chi= 7273.96 , Lambda= 1242.76
iter: 4 , chi= 269.255 , Lambda= 414.252
iter: 5 , chi= 105.473 , Lambda= 138.084
iter: 6 , chi= 100.845 , Lambda= 46.028
iter: 7 , chi= 95.9439 , Lambda= 15.3427
iter: 8 , chi= 92.3017 , Lambda= 5.11423
iter: 9 , chi= 91.442 , Lambda= 1.70474
iter: 10 , chi= 91.3963 , Lambda= 0.568247
iter: 11 , chi= 91.3959 , Lambda= 0.378832
problem solve cost: 0.485304 ms
   makeHessian cost: 0.264057 ms
-------After optimization, we got these parameters :
0.941939  2.09453 0.965586
-------ground truth:
1 2 1
```

采用 ppt 中的 Marquardt 的方法

```
//// method from Marquardt, 1963
double rho = (currentChi_ - tempChi) / scale;
if (rho > 0.75 && isfinite(tempChi)) {
    currentLambda_ /= 3;
    currentChi_ = tempChi;
    return true;
} else if(rho < 0.25) {
    currentLambda_ *= 2;
}
return false;
```

结果：效果欠佳，结束迭代的原因应该是错误次数过多，所以必须把 false_cnt 改成更大的值。

```
Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 16035.7 , Lambda= 349.525
problem solve cost: 0.756021 ms
   makeHessian cost: 0.047206 ms
-------After optimization, we got these parameters :
0.720116  0.87538  1.13788
-------ground truth:
1 2 1
```

采用 Gavin 论文中的第一种方法

```
//// method 1 from Gavin
//// need to change the delta_x and scale with additional diag(Hessian_)
double L = 2.;
if (rho > 0 && isfinite(tempChi)){
    currentLambda_ = max(currentLambda_/ L, 1e-7);
    currentChi_ = tempChi;
    return true;
}else{
    currentLambda_ = min(currentLambda_ * L , 1e7);
    return false;

}
```

结果：效果还行，但是 Lambda 很快便减小为一个很小的值，使其接近与 GN 算法，应该在调整参数 L 和初始值后有进一步提升的空间。

```
Test CurveFitting start...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 9425.37 , Lambda= 8.192
iter: 2 , chi= 5610.47 , Lambda= 4.096
iter: 3 , chi= 2597.3 , Lambda= 2.048
iter: 4 , chi= 1222.02 , Lambda= 1.024
iter: 5 , chi= 900.344 , Lambda= 0.512
iter: 6 , chi= 810.124 , Lambda= 0.256
iter: 7 , chi= 686.698 , Lambda= 0.128
iter: 8 , chi= 498.461 , Lambda= 0.064
iter: 9 , chi= 284.067 , Lambda= 0.032
iter: 10 , chi= 144.796 , Lambda= 0.016
iter: 11 , chi= 107.834 , Lambda= 0.008
iter: 12 , chi= 102.832 , Lambda= 0.004
iter: 13 , chi= 99.4849 , Lambda= 0.002
iter: 14 , chi= 95.7602 , Lambda= 0.001
iter: 15 , chi= 92.9187 , Lambda= 0.0005
iter: 16 , chi= 91.6785 , Lambda= 0.00025
iter: 17 , chi= 91.4185 , Lambda= 0.000125
iter: 18 , chi= 91.3965 , Lambda= 6.25e-05
iter: 19 , chi= 91.3959 , Lambda= 3.125e-05
problem solve cost: 0.680182 ms
   makeHessian cost: 0.437259 ms
-------After optimization, we got these parameters :
0.942151  2.09422  0.96569
-------ground truth:
1 2 1
```

采用 Gavin 论文中的第二种方法

```
//// method 2 from Gavin
// currentLambda_ *= maxDiagonal;
double temp_bx = b_.transpose() * delta_x_;
double alpha = temp_bx / ((tempChi - currentChi_) / 2 + 2 * temp_bx);
cout << "alpha is : " << alpha << endl;
alpha = min(alpha, 1e-1);
//// compute rho(alpha * delta_x_)
//// roll back to compute Chi with alpha * delta_x_
//// rest procedure is same as before
RollbackStates();
delta_x_ *= alpha;
UpdateStates();
tempChi = 0.0;
for (auto edge: edges_) {
    edge.second->ComputeResidual();
    tempChi += edge.second->Chi2();
}
scale = 0;
scale = delta_x_.transpose() * (currentLambda_ * delta_x_ + b_);
scale += 1e-3;
double rho = (currentChi_ - tempChi) / scale;
if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
{
    currentLambda_ = max(currentLambda_/(1 + alpha), 1e-7);
    currentChi_ = tempChi;
    return true;
} else {
    currentLambda_ += abs(tempChi - currentChi_) / (2 * alpha);
    return false;
}
```

结果：效果欠佳，alpha 在初值 0，0，0 下几乎为零，改用其它初值结果也不理想，应该还有点问题需要完善。

2 公式推导，根据课程知识，完成 F,G 中如下两项的推导过程

$$\begin{pmatrix}\alpha_{k+1}\\ \theta_{k+1}\\ \beta_{k+1}\\ b^a_{k+1}\\ b^g_{k+1}\end{pmatrix} = F\begin{pmatrix}\delta\alpha_k\\ \delta\theta_k\\ \delta\beta_k\\ \delta b^a_k\\ \delta b^g_k\end{pmatrix} + G\begin{pmatrix} n^a_k\\ n^g_k\\ n^a_{k+1}\\ n^g_{k+1}\\ n_{b^a_k}\\ n_{b^g_k}\end{pmatrix}$$

$$f_1:\quad \alpha_{k+1} = \alpha_k + \beta_k\,\delta t + \frac{1}{2} a\,\delta t^2$$

$$a = \frac{1}{2}\left(q_{b_i b_k}\left(a^{b_k} - b^a_k\right) + q_{b_i b_{k+1}}\left(a^{b_{k+1}} - b^a_k\right)\right)$$

$$① \quad \alpha_{k+1} + \delta\alpha_{k+1} = \alpha_k + \beta_k\delta t + \frac{1}{4}\Big(q_{b_i b_k}\left(a^{b_k} - b^a_k\right) + \boxed{\overset{A}{q_{b_i b_k} \otimes \begin{bmatrix}1\\ \frac{1}{2}\underline{w}\,\delta t\end{bmatrix}\left(a^{b_{k+1}} - b^a_k\right)}}\Big)\delta t^2$$

$$② \quad \underline{w} = \frac{1}{2}\left(w^{b_{k+1}} - b^g_k - \delta b^g_k + w^{b_k} - b^g_k - \delta b^g_k\right) = w - \delta b^g_k$$

$$③ \quad \therefore A = \left[q_{b_i b_k} \otimes \begin{bmatrix}1\\ \frac{1}{2}(w - \delta b^g_k)\delta t\end{bmatrix}\left(a^{b_{k+1}} - b^a_k\right)\right]$$

$$④ \quad = \left[R_{b_i b_k}\cdot \exp\left((w - \delta b^g_k)^\wedge \delta t\right)\left(a^{b_{k+1}} - b^a_k\right)\right]$$

$$⑤ \quad = \left[\underline{R_{b_i b_k} \exp(w^\wedge \delta t)}\cdot \underline{\exp\left(-\left(J_r(w\delta t)\,\delta b^g_k\right)^\wedge \delta t\right)}\left(a^{b_{k+1}} - b^a_k\right)\right]$$

$$⑥ \quad = \left[\underline{R_{b_i b_{k+1}}}\ \underline{\left(I - \left(J_r(w\delta t)\,\delta b^g_k\right)^\wedge \delta t\right)}\left(a^{b_{k+1}} - b^a_k\right)\right]$$

$$⑦ \quad \therefore \frac{\partial \alpha_{k+1}}{\partial b^g_k} = \lim_{\delta b^g_k \to 0}\frac{\delta\alpha_{k+1}}{\delta b^g_k} = \lim_{\delta b^g_k \to 0}\frac{\frac{1}{4}\left[R_{b_i b_{k+1}}\left(-\left(J_r(w\delta t)\,\delta b^g_k\right)^\wedge \delta t\left(a^{b_{k+1}} - b^a_k\right)\right]\delta t^2\right.}{\delta b^g_k}$$

$$⑧ \quad = \frac{1}{4}R_{b_i b_{k+1}}\left(a^{b_{k+1}} - b^a_k\right)^\wedge \delta t^2\,\underset{\approx I}{J_r(w\delta t)}\,\delta t \approx -\frac{1}{4}R_{b_i b_{k+1}}\left(a^{b_{k+1}} - b^a_k\right)^\wedge \delta t^2(-\delta t)$$

（和 f_{15} 很像，分别是速度 at，位移 $\frac{1}{2}at^2$ 差了一个 $\frac{1}{2}t$ 而已）

图 3: f_{15} 推导结果

$$
\begin{pmatrix}\alpha_{k+1}\\ \theta_{k+1}\\ \beta_{k+1}\\ b^a_{k+1}\\ b^g_{k+1}\end{pmatrix} = F\cdot\begin{pmatrix}\delta\alpha_k\\ \delta\theta_k\\ \delta\beta_k\\ \delta b^a_k\\ \delta b^g_k\end{pmatrix} + G\begin{pmatrix}n^a_k\\ n^g_k\\ n^a_{k+1}\\ n^g_{k+1}\\ n_{b^a}\\ n_{b^g}\end{pmatrix}
$$

$$
g_1:\ \alpha_{k+1} = \alpha_k + \beta_k\,\delta t + \frac{1}{2}a\,\delta t^2
$$

$$
a = \frac{1}{2}\left(q_{b_i b_k}\left(a^{b_k} - b^a_k\right) + q_{b_i b_{k+1}}\left(a^{b_{k+1}} - b^a_k\right)\right)
$$

$$
\alpha_{k+1} + \delta\alpha_{k+1} = \alpha_k + \beta_k\,\delta t + \frac{1}{4}\left[q_{b_i b_k}\left(a^{b_k} - b^a_k\right) + \overbrace{\boxed{q_{b_i b_k}\otimes\begin{bmatrix}1\\ \frac{1}{2}w\delta t\end{bmatrix}\left(a^{b_{k+1}} - b^a_k\right)}}^{A}\right]\delta t^2
$$

$$
w = \frac{1}{2}\left[\left(\bar{w}_k + n^g_k - b^g_k\right) + \left(\bar{w}_{k+1} - b^g_k\right)\right] = w + \frac{1}{2}n^g_k
$$

$$
\begin{aligned}
A &= R_{b_i b_k}\cdot \exp\left[\left(w + \tfrac{1}{2}n^g_k\right)^\wedge \delta t\right]\cdot\left(a^{b_{k+1}} - b^a_k\right)\\
&= R_{b_i b_k}\ \exp\left[(w\delta t)^\wedge\right]\cdot \exp\left[\left(J_r(w\delta t)\cdot\tfrac{1}{2}n^g_k\right)^\wedge \delta t\right]\cdot\left(a^{b_{k+1}} - b^a_k\right)\\
&= R_{b_i b_{k+1}}\cdot\left(I + \left(J_r(w\delta t)\cdot\tfrac{1}{2}n^g_k\right)^\wedge \delta t\right)\left(a^{b_{k+1}} - b^a_k\right)
\end{aligned}
$$

$$
\therefore \frac{\partial \alpha_{k+1}}{\partial n^g_k} = \frac{\partial\, \frac{1}{4}R_{b_i b_{k+1}}\cdot\left(J_r(w\delta t)\cdot\frac{1}{2}n^g_k\right)^\wedge \delta t\,\left(a^{b_{k+1}} - b^a_k\right)\delta t^2}{\partial n^g_k}
$$

$$
\begin{aligned}
&= -\frac{1}{4}R_{b_i b_{k+1}}\left(a^{b_{k+1}} - b^a_k\right)^\wedge \overbrace{J_r(w\delta t)}^{\approx I}\cdot\delta t^2\cdot\left(\tfrac{1}{2}\delta t\right)\\
&\approx -\frac{1}{4}R_{b_i b_{k+1}}\left(a^{b_{k+1}} - b^a_k\right)^\wedge \delta t^2\cdot\left(\tfrac{1}{2}\delta t\right)
\end{aligned}
$$

图 4: g_{12} 推导结果

3 证明式 9

$$
\left(J^T J + \mu I\right)\Delta x = -J^T f \quad,\quad F' = \left(J^T f\right)^T
$$

通常 EVD 是 $X = V\Lambda V^{-1}$，但是由于 $J^T J$ 是 symmetric, semi-positive definite. 可以证明此时特征值可以选取为相互正交的，(即为SVD)

即 $V^{-1} = V^T \quad \therefore J^T J = V\Lambda V^T,\ VV^T = I$

$$
\left(V\Lambda V^T + \mu V V^T\right)\Delta x = -F'^T
$$

$$
V\left(\Lambda + \mu I\right)V^T\Delta x = -F'^T
$$

$$
\Delta x = -V\left(\Lambda + \mu I\right)^{-1}V^T F'^T
$$

展开右式 即得

$$
\begin{aligned}
\Delta x &= -\sum_{j=1}^{n}\frac{1}{\lambda_j + \mu}\cdot v_j\,\underbrace{v_j^T\cdot F'^T}_{\text{标量，位置无所谓}}\\
&= -\sum_{j=1}^{n}\frac{v_j^T\cdot F'^T}{\lambda_j + \mu}\cdot v_j
\end{aligned}
$$