一. 代码修改

样例代码给出了使用 LM 算法来估计曲线 $y = exp(ax^2 + bx + c)$ 参数 a, b, c 的完整过程。

1.1 请绘制样例代码中的阻尼因子 μ 随着迭代变化的曲线图

答:

阻尼因子 μ 是 Levenberg-Maquardt 法中用于限制迭代步长的参数, 引入 μ 的原因是 Gauss-Newton 法中采用的近似二阶泰勒展开只在展开点附近有较好的近似效果, 因此必须限制每次迭代过程中的步长 Δx。

具体每一次迭代中的 μ 以 currentLambda_ 这个成员变量出现在 problem.cc 文件的 bool Problem::Solve(int iterations) 中。

它的初始化在成员函数 void Problem::ComputeLambdaInitLM() 具体迭代计算在成员函数 bool Problem::IsGoodStepInLM()。因此只需要在 while() 循环中加入相关的存储或者绘图语句就可以满足题目要求。

```
while (!stop && (iter < iterations)) {
    std::cout << "iter: " << iter << " , chi= " << currentChi_
              << " , Lambda= " << currentLambda_<< std::endl;

    //*********************modified beginning**********************
    ofstream lambda_data("../../data/lambda_data.txt", ios_base::app);
    lambda_data << iter << "\t" << currentLambda_ << endl;
    lambda_data.close();
    //*********************modified ending**********************
...
}
```

程序运行结果如下 (为了节省篇幅, 中间几步的数值结果这里就不放了, 图中有所表示):

```
Test CurveFitting start ...
iter: 0 , chi= 36048.3 , Lambda= 0.001
iter: 1 , chi= 30015.5 , Lambda= 699.051
...
```

```
 iter: 12 , chi= 91.3959 , Lambda= 0.252554
problem solve cost: 2.01802 ms
makeHessian cost: 0.898437 ms
--------After optimization, we got these parameters :
0.941842  2.09467 0.965537
--------ground truth:
 1.0,  2.0,  1.0
```

上述代码将 μ 的每次迭代数据存入文件夹`CurveFittingLM/data/lambda_data`文件夹内，存储格式是 n 行两列，第一列是迭代次数，第二列是 lambda(即 μ) 的值。然后再用 python matplotlib.plot 库将折线图画出，画图程序`draw_data.py`如下所示：

```python
#!/usr/bin/python
import matplotlib.pyplot as plt
filename = "lambda_data.txt"
X, Y = [],[]
for line in open(filename, 'r'):
    value = [float(s) for s in line.split()]
    X.append(value[0])
    Y.append(value[1])

plt.plot(X, Y, 'b--')
plt.plot(X, Y, 'ro')
plt.title("lambda data")
plt.xlabel("iterations")
plt.ylabel("lambda")
plt.savefig("./lambda_line_chart")
plt.show()
```

运行 python 程序之后得到折线图，即为题目要求的 LM 阻尼因子 μ 随着迭代变化的曲线图，结果如图 1 所示：

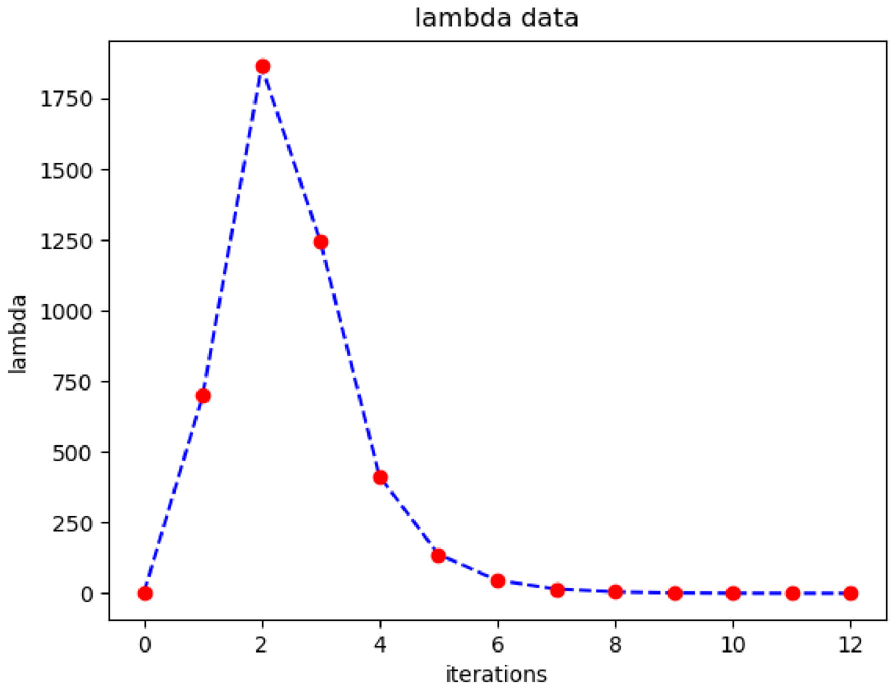

图 1: line chart of lambda

从图中可以看出，μ 的值先增大，后减小，即迭代步长 Δx 先增大，后减小。

1.2 将曲线函数改成 $y = ax^2 + bx + c$

修改样例代码中的残差计算、雅克比计算函数等，完成曲线参数估计。

答:

残差模块的更改只需要将原函数 $y = exp(ax^2 + bx + c)$ 更改为 $y = ax^2 + bx + c$，然后再减去观测值 y 即可 (在类中是成员变量 y_).

$y = ax^2 + bx + c$ 分别对 a, b, c 求导得到新的雅克比函数 $[x^2, x, 1]^\top$。然后用新的雅克比函数替换原程序中相应位置即可。关键代码如下所示:

```
// 计算曲线模型误差
virtual void ComputeResidual() override
{
    Vec3 abc = verticies_[0]->Parameters();  // 估计的参数
    // residual_(0) = std::exp( abc(0)*x_*x_ + abc(1)*x_ + abc(2) ) - y_; // 构建残差
```

```
    residual_(0) = abc(0)*x_*x_ + abc(1)*x_ + abc(2) - y_; // 构建残差
}

// 计算残差对变量的雅克比
// 对a, b, c求导，而不是对x求导
virtual void ComputeJacobians() override
{
    Vec3 abc = verticies_[0]->Parameters();
    double y = abc(0)*x_*x_ + abc(1)*x_ + abc(2) ;

    // 误差为1维，状态量 3 个，所以是 1x3 的雅克比矩阵
    Eigen::Matrix<double, 1, 3> jaco_abc;
    // jaco_abc << x_ * x_ * exp_y, x_ * exp_y , 1 * exp_y;
    jaco_abc << x_ * x_ , x_, 1 ;
    jacobians_[0] = jaco_abc;
}
```

程序运行结果如下所示：

```
Test CurveFitting start ...
iter: 0 , chi= 315486 , Lambda= 1.95033
iter: 1 , chi= 91.4713 , Lambda= 0.650111
iter: 2 , chi= 91.3951 , Lambda= 0.216704
iter: 3 , chi= 91.395 , Lambda= 0.144469
problem solve cost: 1.01164 ms
makeHessian cost: 0.509846 ms
-------After optimization, we got these parameters :
 1.00611  1.96185 0.995133
-------ground truth:
1.0,  2.0,  1.0
```

阻尼因子 μ(lambda) 随迭代步变化的折线图如图 2 所示：

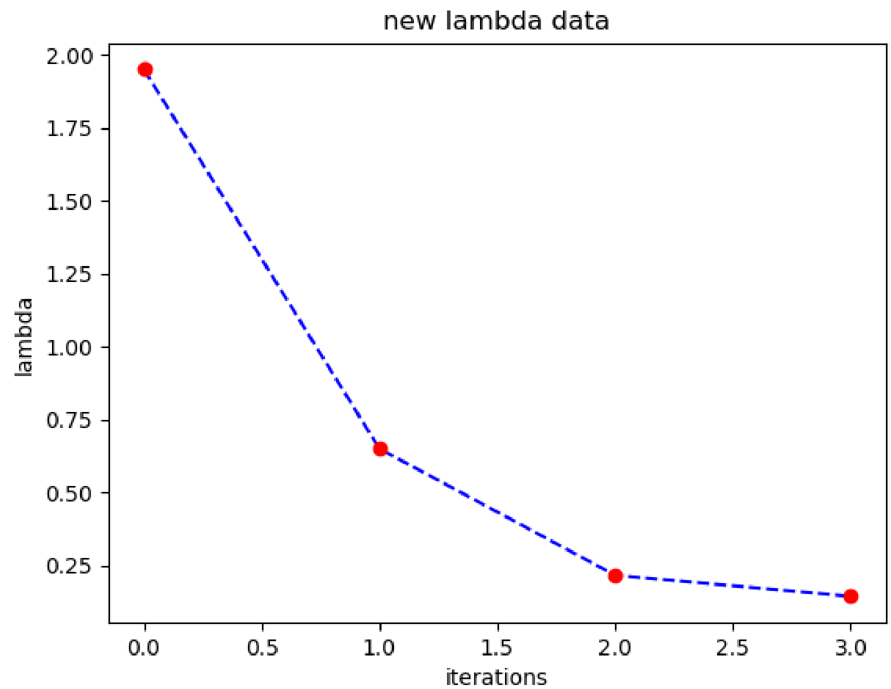

图 2: line chart of new_lambda

可以看出，其迭代步数较少，程序运行时间也比较短，可能因为这个函数比较简单，求解起来更加容易。

1.3 实现其他阻尼因子策略，并给出实验对比

阻尼因子策略参考论文 [1]4.1.1 节。该节总结了 3 种阻尼因子更新策略，其更新算法如下：

4.1.1 Initialization and update of the L-M parameter, λ, and the parameters $\mathbf{p}$

In `lm.m` users may select one of three methods for initializing and updating λ and $\boldsymbol{p}$.

1. $\lambda_0 = \lambda_o$; λ_o is user-specified [8].
   use eq'n (13) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (16) for ρ
   if $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h}$; $\lambda_{i+1} = \max[\lambda_i / L_\downarrow, 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \min[\lambda_i L_\uparrow, 10^7]$;
2. $\lambda_0 = \lambda_o \max\left[\mathsf{diag}[\boldsymbol{J}^\mathsf{T}\boldsymbol{W}\boldsymbol{J}]\right]$; λ_o is user-specified.
   use eq'n (12) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   $\alpha = \left(\left(\boldsymbol{J}^\mathsf{T}\boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)^\mathsf{T} \boldsymbol{h}\right) / \left(\left(\chi^2(\boldsymbol{p}+\boldsymbol{h}) - \chi^2(\boldsymbol{p})\right)/2 + 2\left(\boldsymbol{J}^\mathsf{T}\boldsymbol{W}(\boldsymbol{y} - \hat{\boldsymbol{y}}(\boldsymbol{p}))\right)^\mathsf{T}\boldsymbol{h}\right)$;
   if $\rho_i(\alpha\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow p + \alpha\boldsymbol{h}$; $\lambda_{i+1} = \max[\lambda_i/(1+\alpha), 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \lambda_i + |\chi^2(\boldsymbol{p}+\alpha\boldsymbol{h}) - \chi^2(\boldsymbol{p})|/(2\alpha)$;
3. $\lambda_0 = \lambda_o \max\left[\mathsf{diag}[\boldsymbol{J}^\mathsf{T}\boldsymbol{W}\boldsymbol{J}]\right]$; λ_o is user-specified [9].
   use eq'n (12) for $\boldsymbol{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   if $\rho_i(\boldsymbol{h}) > \epsilon_4$: $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{h}$; $\lambda_{i+1} = \lambda_i \max[1/3, 1 - (2\rho_i - 1)^3]$; $\nu_i = 2$;
   otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\nu_{i+1} = 2\nu_i$;

只实现了 algorithm 1 和 algorithm 3，algorithm 2 总是在第一次迭代后结束，看到作业讲

评里面，algorithm 2 的效果并不是很强，就不想 debug 了，休息一下，哈哈。算法 1 和算法 3 的运行结果下所示：

```
Algorithm 1===========================
Test CurveFitting start ...
 iter: 0 , chi= 36048.3 , Lambda= 0.001
 iter: 1 , chi= 28146.3 , Lambda= 196.84
 ...
 iter: 9 , chi= 91.3959 , Lambda= 0.00608628
problem solve cost: 1.77909 ms
makeHessian cost: 0.872804 ms
————————After optimization, we got these parameters :
0.941867  2.09463 0.965551
————————ground truth:
 1.0,  2.0,  1.0

Algorithm 3===========================
Test CurveFitting start ...
 iter: 0 , chi= 36048.3 , Lambda= 0.001
 iter: 1 , chi= 30015.5 , Lambda= 699.051
 ...
 iter: 12 , chi= 91.3959 , Lambda= 0.252554
problem solve cost: 2.01802 ms
makeHessian cost: 0.898437 ms
————————After optimization, we got these parameters :
0.941842  2.09467 0.965537
————————ground truth:
 1.0,  2.0,  1.0
```

总结：Algorithm 1 的效果比 Algorithm 3 略好，迭代次数更少，求解所用时间也更少，但两种方法对状态变量的估计结果是差不多的。两种方法的 lambda 随迭代次数变化折线图如图 3 所示：

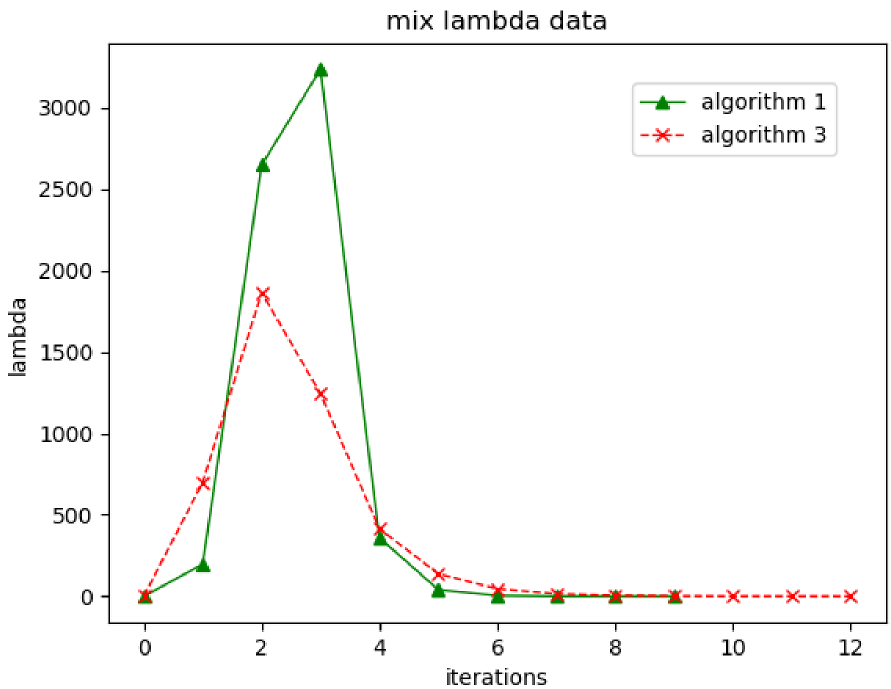

图 3: mix line chart of lambda using different algorithms

二. 公式推导

题目: 根据课程知识, 完成 F, G 中如下两项的推导过程:

$$1)f_{15} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta b_k^g} = -\frac{1}{4}(R_{b_i b_{k+1}}[(a^{b_{k+1}} - b_k^a)]_\times \delta t^2)(-\delta t)$$

$$2)g_{12} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta n_k^g} = -\frac{1}{4}(R_{b_i b_{k+1}}[(a^{b_{k+1}} - b_k^a)]_\times \delta t^2)(\frac{1}{2}\delta t)$$

答:

需要用到的公式:

$$\begin{aligned} a &= \frac{1}{2}(q_{b_i b_k}(a^{b_k} - b_k^a) + q_{b_i b_{k+1}}(a^{b_{k+1}} - b_k^a)) \\ &= \frac{1}{2}(q_{b_i b_k}(a^{b_k} - b_k^a) + q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a)) \end{aligned} \tag{1}$$

$$\begin{aligned}\alpha_{b_ib_{k+1}} &= \alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{2}a\delta t^2 \\ &= \alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{2}(\frac{1}{2}(q_{b_ib_k}(a^{b_k} - b_k^a) + q_{b_ib_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t \end{bmatrix}(a^{b_{k+1}} - b_k^a)))\delta t^2\end{aligned} \tag{2}$$

(1)f_{15}:

$$\begin{aligned}f_{15} &= \frac{\partial \alpha_{b_ib_{k+1}}}{\partial \delta b_k^g} \\ &= \frac{\partial \frac{1}{4} q_{b_ib_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t \end{bmatrix} \otimes \begin{bmatrix} 1 \\ -\frac{1}{2}\delta b_k^g \delta t \end{bmatrix}(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta b_k^g} \\ &= \frac{\partial \frac{1}{4} q_{b_ib_{k+1}} \otimes \begin{bmatrix} 1 \\ -\frac{1}{2}\delta_k^g \delta t \end{bmatrix}(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta b_k^g} \\ &= \frac{\partial \frac{1}{4} R_{b_ib_{k+1}} exp([-\delta b_k^g \delta t]_\times)(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta b_k^g} \\ &= \frac{\partial \frac{1}{4} R_{b_ib_{k+1}} (I + [-\delta b_k^g \delta t]_\times)(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta b_k^g} \\ &= \frac{\partial \frac{1}{4} R_{b_ib_{k+1}} [(a^{b_{k+1}} - b_k^a)\delta t^2]_\times (-\delta b_k^g \delta t)}{\partial \delta b_k^g} \\ &= \frac{1}{4}(R_{b_ib_{k+1}}[(a^{b_{k+1}} - b_k^a)]_\times \delta t^2)(-\delta t)\end{aligned} \tag{3}$$

(2)g_{12}，根据公式 (2)，有:

$$\begin{aligned}
g_{12} &= \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta n_k^g} \\
&= \frac{\partial \frac{1}{4} q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega \delta t \end{bmatrix} \otimes \begin{bmatrix} 1 \\ \frac{1}{4}\delta n_k^g \delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a)\delta t^2}{\partial n_k^g} \\
&= \frac{\partial \frac{1}{4} q_{b_i b_{k+1}} \otimes \begin{bmatrix} 1 \\ \frac{1}{4} n_k^g \delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a)\delta t^2}{\partial n_k^g} \\
&= \frac{\partial \frac{1}{4} R_{b_i b_{k+1}} exp(I + [\frac{1}{2} n_k^g \delta t]_\times)(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial n_k^g} \\
&= \frac{\partial - \frac{1}{4} R_{b_i b_{k+1}} [(a^{b_{k+1}} - b_k^a)\delta t^2]_\times (\frac{1}{2} n_k^g \delta t)}{\partial n_k^g} \\
&= -\frac{1}{4}(R_{b_i b_{k+1}} [(a^{b_{k+1}} - b_k^a)]_\times \delta t^2)(\frac{1}{2}\delta t)
\end{aligned} \tag{4}$$

三. 证明 PPT 中公式 (9)

PPT 中公式 (9):

$$\Delta x_{lm} = -\sum_{j=1}^{n} \frac{v_j^\top F'^\top}{\lambda_j + \mu} v_j$$

证明: 根据 LM method 中阻尼因子 μ 的引入，有:

$$\begin{aligned}
(J^\top J + \mu I)\Delta x_{lm} &= (V \Lambda V^\top + \mu I)\Delta x_{lm} \\
&= (V(\Lambda + \mu I)V^\top)\Delta x_{lm} \\
&= -J^\top f = -F'^\top
\end{aligned} \tag{5}$$

可得：

$$\begin{aligned}
\Delta x_{lm} &= -V(\lambda+\mu I)^{-1}V^{\top}F'^{\top} \\
&= -\begin{bmatrix} v_1 v_2 \dots v_n \end{bmatrix}\begin{bmatrix} \frac{1}{\lambda_1+\mu} & & \cdots & \\ & \frac{1}{\lambda_2+\mu} & \cdots & \\ & & \ddots & \\ & & & \frac{1}{\lambda_n+\mu} \end{bmatrix}\begin{bmatrix} v_1^{\top} \\ v_2^{\top} \\ \vdots \\ v_n^{\top} \end{bmatrix} F'^{\top} \\
&= -\begin{bmatrix} v_1 v_2 \dots v_n \end{bmatrix}\begin{bmatrix} \frac{v_1^{\top}F'^{\top}}{\lambda_1+\mu} \\ \frac{v_2^{\top}F'^{\top}}{\lambda_2+\mu} \\ \vdots \\ \frac{v_n^{\top}F'^{\top}}{\lambda_n+\mu} \end{bmatrix} \\
&= -(\frac{v_1^{\top}F'^{\top}}{\lambda_1+\mu}v_1 + \frac{v_2^{\top}F'^{\top}}{\lambda_2+\mu}v_2 + \cdots + \frac{v_n^{\top}F'^{\top}}{\lambda_n+\mu}v_n) \\
&= -\sum_{j=1}^{n}\frac{v_j^{\top}F'^{\top}}{\lambda_j+\mu}v_j
\end{aligned} \tag{6}$$

证毕。

参考文献

[1] Gavin, H. P. (2019). The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems. Department of Civil and Environmental Engineering, Duke University, 1-19.