深蓝学院
shenlanxueyuan.com

第三次作业分享

主讲人 尤海荣

作业1

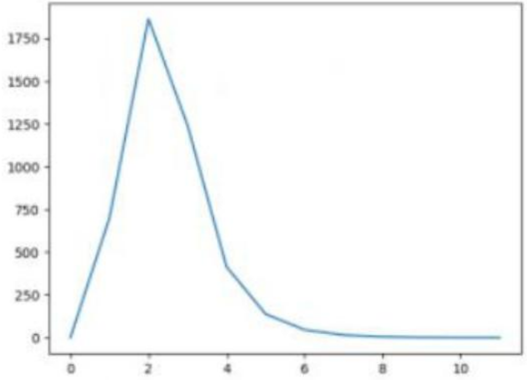
作业

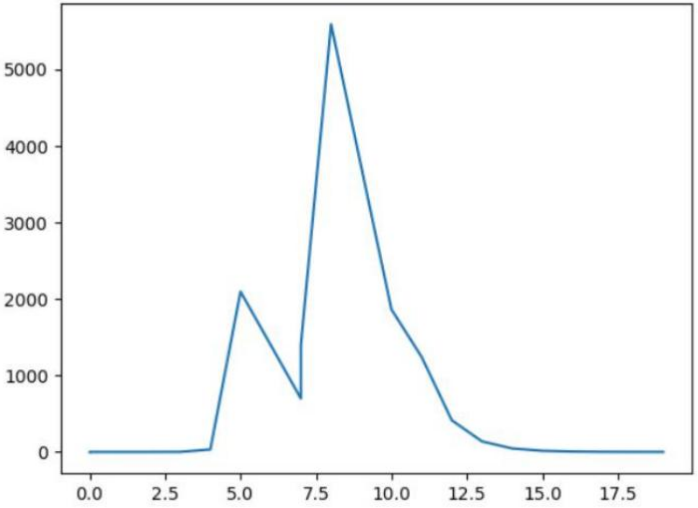
1 样例代码给出了使用 LM 算法来估计曲线 $y = \exp(ax^2 + bx + c)$ 参数 a, b, c 的完整过程。

1. 请绘制样例代码中 LM 阻尼因子 μ 随着迭代变化的曲线图
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。
3. 实现其他更优秀的阻尼因子策略，并给出实验对比（选做，评优秀），策略可参考论文[a] 4.1.1 节。

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

细节之处:不仅需要画出代码中打印的正确的lambda，还需要将IsGoodStepInLm()函数返回flase失败的lambda也要画出来。





1.1 初始值tau对lambda的影响

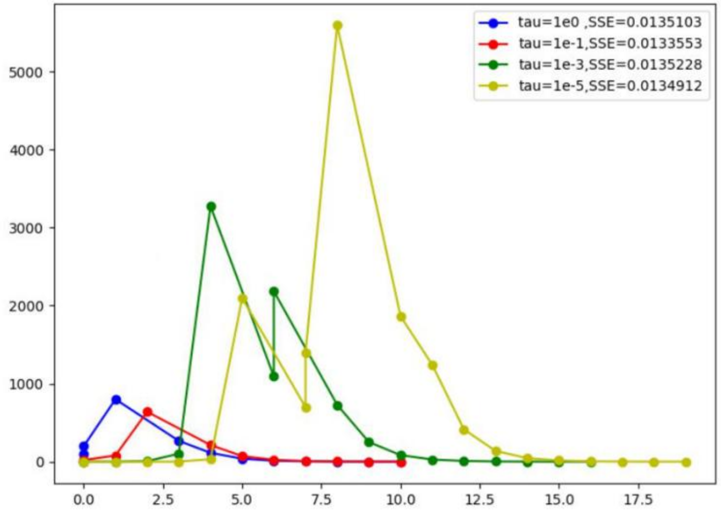
```cpp
void Problem::ComputeLambdaInitLM()
{
    ni_ = 2.;
    currentLambda_ = -1.;
    currentChi_ = 0.0;

    for (auto edge : edges_)
    {
        currentChi_ += edge.second->Chi2();
    }
    if (err_prior_.rows() > 0)
        currentChi_ += err_prior_.norm();

    stopThresholdLM_ = 1e-6 * currentChi_; // 迭代条件为 误差下降 1e-6

    double maxDiagonal = 0;
    ulong size = Hessian_.cols();// 3

    for (ulong i = 0; i < size; ++i)
    {
        maxDiagonal = std::max(fabs(Hessian_(i, i)), maxDiagonal);/
    }
    double tau = 1e-5;
    currentLambda_ = tau * maxDiagonal;
}
```



1.2 将指数函数换成二次函数，实现曲线函数参数估计

需要修改的核心代码：

1.Main函数中观测方程修改为：

```
// 观测 y
double y = a*x*x + b*x + c + n;
```

2.修改残差计算函数为：

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

3.修改雅克比计算函数为：

```
// 计算残差对变量的雅克比
virtual void ComputeJacobians() override
{
    Vec3 abc = verticies_[0]->Parameters();
    double exp_y = std::exp( abc(0)*x_*x_ + abc(1)*x_ + abc(2) );

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

1.2 将指数函数换成二次函数，实现曲线函数参数估计

深蓝学院 shenlanxueyuan.com

注意：
本题结果真实值为：a=1，b=2，c=1；若采用原始参数，拟合结果较差，可通过以下操作进行改进：
1.增加数据点数，如N=1000（原始N=100）
2.增大步长以增大数据范围，如x=i/10（原始x=i/100）
3.减小噪声均方差，如w_sigma=0.01（原始w_sigma=0.1）

这里我尝试了将N改为1000：

```
youhairong@youhairong-Legion-Y7000P-2019-PG0:~/文档/CurveFitting_LM/build/app$ .
/testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 974.658 , Lambda= 6.65001
iter: 2 , chi= 973.881 , Lambda= 2.21667
iter: 3 , chi= 973.88 , Lambda= 1.47778
problem solve cost: 1.47013 ms
   makeHessian cost: 1.18132 ms
-------After optimization, we got these parameters :
0.999588   2.0063 0.968786
-------ground truth:
1.0,  2.0,  1.0
```

1.3 实现其他更新阻尼因子策略：

论文中有三种 阻尼因子策略，如下：

1. $\lambda_0 = \lambda_o$; λ_o is user-specified [8].
   use eq'n (13) for $\mathbf{h}_{\text{lm}}$ and eq'n (16) for ρ
   if $\rho_i(\mathbf{h}) > \epsilon_4$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{h}$; $\lambda_{i+1} = \max[\lambda_i / L_\downarrow, 10^{-7}]$;
   otherwise: $\lambda_{i+1} = \min[\lambda_i L_\uparrow, 10^7]$;

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

3. $\lambda_0 = \lambda_o \max\left[\text{diag}[\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J}]\right]$; λ_o is user-specified [9].
   use eq'n (12) for $\mathbf{h}_{\text{lm}}$ and eq'n (15) for ρ
   if $\rho_i(\mathbf{h}) > \epsilon_4$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{h}$; $\lambda_{i+1} = \lambda_i \max[1/3, 1 - (2\rho_i - 1)^3]$; $\nu_i = 2$;
   otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\nu_{i+1} = 2\nu_i$;

1.3 实现其他更新阻尼因子策略：

策略1中使用论文中式13，16更新，不同于另两种策略，除IsGoodStepInLM()还需修改AddLambdatoHessianLM()，RemoveLambdaHessianLM()

```
void Problem::AddLambdatoHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    for (ulong i = 0; i < size; ++i) {
        Hessian_(i, i) *= (1.+currentLambda_);
    }
}

void Problem::RemoveLambdaHessianLM() {
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    // TODO:: 这里不应该减去一个，数值的反复加减容易造成数值精度出问题？而应该保存叠加lambda前的值，在这里直接赋值
    for (ulong i = 0; i < size; ++i) {
        Hessian_(i, i) /= (1.+currentLambda_);
    }
}
```

1.3 实现其他更新阻尼因子策略：

深蓝学院 shenlanxueyuan.com

```
bool Problem::IsGoodStepInLM() {

    // recompute residuals after update state
    // 统计所有的残差
    double tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }

    ulong size = Hessian_.cols();
    MatXX diag_hessian(MatXX::Zero(size, size));
    for (ulong i = 0; i < size; ++i) {
        diag_hessian(i, i) = Hessian_(i, i);
    }

    double scale = 0;
    scale = delta_x_.transpose() * (currentLambda_ * diag_hessian * delta_x_ + b_);
    scale += 1e-3;    // make sure it's non-zero :)
```

1.3 实现其他更新阻尼因子策略：

```
    scale += 1e-3;    // make sure it's non-zero :)

    double rho = (currentChi_ - tempChi) / scale;
//    std::cout << "rho = " << rho << std::endl;

    double L_down = 9.0;
    double L_up = 11.0;

    if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
    {
        currentLambda_ = std::max(currentLambda_ / L_down, 1e-7);
        currentChi_ = tempChi;
        return true;
    } else {
        currentLambda_ = std::min(currentLambda_ * L_up, 1e7);
        return false;
    }
}
```

1.3 实现其他更新阻尼因子策略：


```
Test CurveFitting start...
iter: 0, chi= 36048.3, Lambda= 1
iter: 1, chi= 34219.5, Lambda= 13.4444
iter: 2, chi= 1141.81, Lambda= 1.49383
iter: 3, chi= 531.043, Lambda= 0.165981
iter: 4, chi= 365.945, Lambda= 0.0184423
iter: 5, chi= 133.522, Lambda= 0.00204915
iter: 6, chi= 99.5329, Lambda= 0.000227683
iter: 7, chi= 91.9421, Lambda= 2.52981e-05
iter: 8, chi= 91.3974, Lambda= 2.8109e-06
iter: 9, chi= 91.3959, Lambda= 3.12322e-07
problem solve cost: 2.7422 ms
makeHessian cost: 1.70599 ms
-------After optimization, we got these parameters :
0.941903  2.09458 0.965571
-------ground truth:
1.0,  2.0,  1.0
```

1.3 实现其他更新阻尼因子策略：

策略2

```
bool Problem::IsGoodStepInLM() {

    // recompute residuals after update state
    // 统计所有的残差
    double tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }
//    cout << "currentChi_:" << currentChi_ << endl;
//    cout << "tempChi:" << tempChi << endl;

    double Numerator = b_.transpose() * delta_x_;
    double alpha = Numerator / ((currentChi_ - tempChi)/2. + 2.*Numerator);
    alpha = std::max(alpha, 1e-1);
//    cout << "Numerator:" << Numerator << endl;
//    cout << "alpha:" << alpha << endl;

    RollbackStates();
    delta_x_ *=alpha;
    UpdateStates();
```

1.3 实现其他更新阻尼因子策略：

深蓝学院 shenlanxueyuan.com

策略2

```
    tempChi = 0.0;
    for (auto edge: edges_) {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }

    double scale = 0;
    scale = delta_x_.transpose() * (currentLambda_ * delta_x_ + b_);
    scale += 1e-3;    // make sure it's non-zero :)
    double rho = (currentChi_ - tempChi) / scale;
//    std::cout << "rho = " << rho << std::endl;

    if (rho > 0 && isfinite(tempChi))   // last step was good, 误差在下降
    {
        currentLambda_ = std::max(currentLambda_ / (1.+alpha), 1e-7);
        currentChi_ = tempChi;
        return true;
    } else {
        currentLambda_ += abs(currentChi_ - tempChi)/(2.*alpha);
        return false;
    }
}
```

1.3 实现其他更新阻尼因子策略：

```
Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 19.95
iter: 1 , chi= 3.21386e+06 , Lambda= 13.3
iter: 2 , chi= 3.21386e+06 , Lambda= 9.35928
iter: 3 , chi= 3.21386e+06 , Lambda= 7.42248
iter: 4 , chi= 3.21386e+06 , Lambda= 6.56868
iter: 5 , chi= 3.21386e+06 , Lambda= 6.08372
iter: 6 , chi= 3.21386e+06 , Lambda= 5.72601
iter: 7 , chi= 3.21386e+06 , Lambda= 5.43387
iter: 8 , chi= 3.21386e+06 , Lambda= 5.18617
iter: 9 , chi= 3.21386e+06 , Lambda= 4.97185
iter: 10 , chi= 3.21386e+06 , Lambda= 4.78366
iter: 11 , chi= 3.21386e+06 , Lambda= 4.61649
iter: 12 , chi= 3.21386e+06 , Lambda= 4.46657
iter: 13 , chi= 3.21386e+06 , Lambda= 4.33102
iter: 14 , chi= 3.21386e+06 , Lambda= 4.20763
iter: 15 , chi= 3.21386e+06 , Lambda= 4.09464
iter: 16 , chi= 3.21386e+06 , Lambda= 3.99062
iter: 17 , chi= 3.21386e+06 , Lambda= 3.89442
iter: 18 , chi= 3.21386e+06 , Lambda= 3.80508
iter: 19 , chi= 3.21386e+06 , Lambda= 3.72182
iter: 20 , chi= 3.21386e+06 , Lambda= 3.64396
iter: 21 , chi= 3.21386e+06 , Lambda= 3.57093
iter: 22 , chi= 3.21386e+06 , Lambda= 3.50224
iter: 23 , chi= 3.21386e+06 , Lambda= 3.43747
iter: 24 , chi= 3.21386e+06 , Lambda= 3.37625
iter: 25 , chi= 3.21386e+06 , Lambda= 3.31828
iter: 26 , chi= 3.21386e+06 , Lambda= 3.26325
iter: 27 , chi= 3.21386e+06 , Lambda= 3.21094
iter: 28 , chi= 3.21386e+06 , Lambda= 3.16113
iter: 29 , chi= 3.21386e+06 , Lambda= 3.1136
problem solve cost: 40.5925 ms
   makeHessian cost: 32.5209 ms
-------After optimization, we got these parameters :
 0.91409  1.83397 0.881693
-------ground truth:
1.0,  2.0,  1.0
```

作业2

2 公式推导，根据课程知识，完成 $\mathbf{F},\mathbf{G}$ 中如下两项的推导过程：

$$\mathbf{f}_{15}=\frac{\partial\alpha_{b_ib_{k+1}}}{\partial\delta\mathbf{b}_k^g}=-\frac{1}{4}(\mathbf{R}_{b_ib_{k+1}}[(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)]_\times\delta t^2)(-\delta t)$$

$$\mathbf{g}_{12}=\frac{\partial\alpha_{b_ib_{k+1}}}{\partial\mathbf{n}_k^g}=-\frac{1}{4}(\mathbf{R}_{b_ib_{k+1}}[(\mathbf{a}^{b_{k+1}}-\mathbf{b}_k^a)]_\times\delta t^2)(\frac{1}{2}\delta t)$$

作业2

$$f_{15} = \frac{\partial(\alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{2}a\delta t^2)}{\partial \delta b_k^g}$$

$$= \frac{\partial(\frac{1}{2}a\delta t^2)}{\partial \delta b_k^g}$$ 前两项与导数无关

$$= \frac{\partial\left(\frac{1}{2}\frac{1}{2}\left(q_{b_ib_k}(a^{b_k} - b_k^a) + q_{b_ib_{k+1}}(\alpha^{b_{k+1}} - b_k^\alpha)\right)\delta t^2\right)}{\partial \delta b_k^g}$$

$q_{b_ib_k}(\alpha^{b_k} - b_k^a)$ 与导数无关

$$= \frac{1}{4}\frac{\partial\left(q_{b_ib_{k+1}}(\alpha^{b_{k+1}} - b_k^\alpha)\right)\delta t^2}{\partial \delta b_k^g}$$

$$= \frac{1}{4}\frac{\partial q_{b_ib_{k+1}} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t\end{bmatrix} \otimes \begin{bmatrix} 1 \\ -\frac{1}{2}\delta b_k^g\delta t\end{bmatrix}(\alpha^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta b_k^g}$$

$$= \frac{1}{4}\frac{\partial R_{b_ib_{k+1}}\exp([-\delta b_k^g\delta t]_\times)(\alpha^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta b_k^g}$$

$$= \frac{1}{4}\frac{\partial R_{b_ib_{k+1}}(I + [-\delta b_k^g\delta t]_\times)(\alpha^{b_{k+1}} - b_k^\alpha)\delta t^2}{\partial \delta b_k^g}$$

$$= \frac{1}{4}\frac{\partial -R_{b_ib_{k+1}}([(\alpha^{b_{k+1}} - b_k^\alpha)\delta t^2]_\times)(-\delta b_k^g\delta t)}{\partial \delta b_k^g}$$

$$= -\frac{1}{4}\left(R_{b_ib_{k+1}}[(\alpha^{b_{k+1}} - b_k^\alpha)]_\times\delta t^2\right)(-\delta t)$$

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

推导：
$$g_{12} = \frac{\partial(\alpha_{b_ib_k} + \beta_{b_ib_k}\delta t + \frac{1}{2}a\delta t^2)}{\partial \delta n_k^g}$$

$$= \frac{1}{4}\frac{\partial q_{b_ib_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\omega\delta t\end{bmatrix} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}\delta n_k^g\delta t\end{bmatrix}(\alpha^{b_{k+1}} - b_k^\alpha)\delta t^2}{\partial \delta n_k^g}$$

$$= \frac{1}{4}\frac{\partial R_{b_ib_{k+1}}\exp([\frac{1}{2}\delta n_k^g\delta t]_\times)(\alpha^{b_{k+1}} - b_k^\alpha)\delta t^2}{\partial \delta n_k^g}$$

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

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

$$= -\frac{1}{4}\left(R_{b_ib_{k+1}}[(\alpha^{b_{k+1}} - b_k^\alpha)]_\times\delta t^2\right)\left(\frac{1}{2}\delta t\right)$$

作业3

3 证明式(9)。

阻尼因子初始值的选取

阻尼因子 μ 大小是相对于 $\mathbf{J}^\top\mathbf{J}$ 的元素而言的。半正定的信息矩阵 $\mathbf{J}^\top\mathbf{J}$ 特征值 $\{\lambda_j\}$ 和对应的特征向量为 $\{\mathbf{v}_j\}$。对 $\mathbf{J}^\top\mathbf{J}$ 做特征值分解分解后有：$\mathbf{J}^\top\mathbf{J} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^\top$ 可得：

$$\Delta \mathbf{x}_{\mathrm{lm}} = -\sum_{j=1}^{n} \frac{\mathbf{v}_j^\top \mathbf{F}'^\top}{\lambda_j + \mu} \mathbf{v}_j \tag{9}$$

所以，一个简单的 μ_0 初始值的策略就是：

$$\mu_0 = \tau \cdot \max\left\{\left(\mathbf{J}^\top\mathbf{J}\right)_{ii}\right\}$$

通常，按需设定 $\tau \sim [10^{-8}, 1]$。

作业3

深蓝学院
shenlanxueyuan.com

$$(J^TJ+\mu I)\Delta x_{lm} = -J^Tf$$

$$(V\Lambda V^T+\mu I)\Delta x_{lm} = -J^Tf$$

$$(V(\Lambda+\mu I)V^T)\Delta x_{lm} = -J^Tf$$

左右同乘V^T $$(\Lambda+\mu I)V^T\Delta x_{lm} = -V^TJ^Tf$$

$F(x)=(J^Tf)^T$ $$(\Lambda+\mu I)V^T\Delta x_{lm} = -V^TF'(x)^T$$

$=(J^Tf)^T_{1*m*m*1=1\times1}$ $$V^T\Delta x_{lm} = -(\Lambda+\mu I)^{T}V^TF'(x)^T$$

$$V^T\Delta x_{lm} = -\begin{bmatrix} \frac{1}{\lambda_1+\mu} & 0 & 0 & \cdots \\ 0 & \frac{1}{\lambda_2+\mu} & 0 & \cdots \\ \cdots & \cdots & \cdots & \frac{1}{\lambda_j+\mu} \end{bmatrix} V^TF'(x)^T$$

$$\Delta x_{lm} = -[v_1 \ldots v_2]\begin{bmatrix} \frac{1}{\lambda_1+\mu} & 0 & 0 & \cdots \\ 0 & \frac{1}{\lambda_2+\mu} & 0 & \cdots \\ 0 & \cdots & \cdots & \frac{1}{\lambda_j+\mu} \end{bmatrix}\begin{bmatrix} v_1 \\ \cdots \\ v_2 \end{bmatrix}F'(x)^T$$

$$\Longrightarrow \quad \Delta x_{lm} = -\sum_{j=1}^{n}\frac{v_j^TF'^T}{\lambda_j+\mu}v_j$$

深蓝学院
shenlanxueyuan.com

感谢各位聆听！

Thanks for Listening