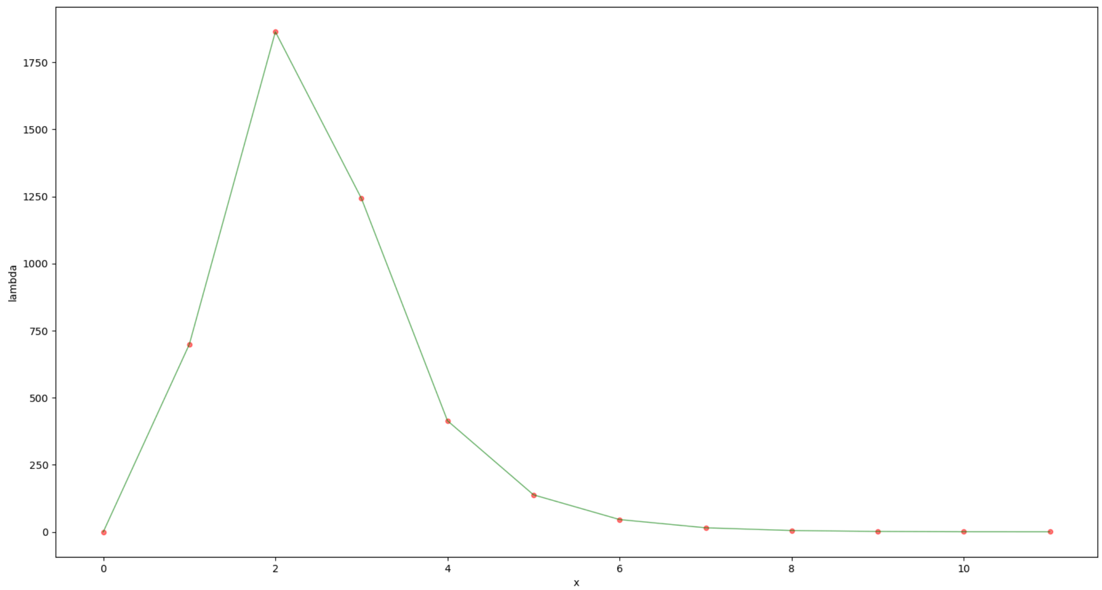
1.LM算法相关

1.1 绘制阻尼因子μ随迭代变化曲线图




```
# 绘制变化曲线的python程序
import matplotlib.pyplot as plt

x_list = list(range(0, 12))
lambda_list = [0.001, 699.051, 1864.14, 1242.76,
    414.252, 138.084, 46.028, 15.3427, 5.11423, 1.70474, 0.568247, 0.378832]

plt.figure('lambda iteration')
ax = plt.gca()

ax.set_xlabel('x')
ax.set_ylabel('lambda')

ax.scatter(x_list, lambda_list, c='r', s=20, alpha=0.5)
ax.plot(x_list, lambda_list, c='g', linewidth=1, alpha=0.6)

plt.show()
```

1.2 改曲线函数，修改代码中残差计算，实现曲线参数估计

修改CurveFitting.cpp代码中部分如下：

```
// main函数部分:
double x = i / 100.;
double n = noise(generator);
// 观测 y
// double y = std::exp( a*x*x + b*x + c ) + n;
double y = a * x * x + b * x + c + n;

// ComputeResidual函数部分:
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

// ComputeJacobians函数部分:
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;
```

（一）观测数量的讨论：

同学们可以尝试一下采样点数N取不同值，看看拟合参数结果会如何变化。当N较小比如取100时，拟合出来a=1.61039，b=1.61853，c=0.995178，和真值相差较大。当N取较大时，拟合出来参数会逐渐收敛于真值。

比如，将采样数据点N增大为1000，得到结果如下：

```
xwl@xwl-Inspiron-15-7000-Gaming:~/Documents/VSLAM-fundamentals-and-VIO-learning/L12/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: 7.17463 ms
   makeHessian cost: 5.5162 ms
-------After optimization, we got these parameters :
0.999588   2.0063 0.968786
-------ground truth:
1.0,  2.0,  1.0
```

（二）初值的讨论：

再次取1000个点，这次考虑将初值(0.0 0.0 0.0)修改为(0.9 2.1 0.9)，即给待估计参数一个较好的初值，再次运行结果如下。与上面的1000点的结果对比，发现参数估计精度没有提升。推测原因是：我们拟合的曲线函数较简单，且只有一个极小值点，因此初值的选择并不会使得优化至错误的极小值，所以最终的精度更多地取决于观测数据的噪声。但是，较好的初值带来的好处便是，迭代次数的减少、耗时的减少，当然更快地收敛。

在面对真正复杂的函数时，好的初始值还是很有必要的。

```
xwl@xwl-Inspiron-15-7000-Gaming:~/Documents/VSLAM-fundamentals-and-VIO-learning/L12/CurveFitting_LM/build/app$ ./testCurv

Test CurveFitting start...
iter: 0 , chi= 16792 , Lambda= 0.001
iter: 1 , chi= 974.673 , Lambda= 0.000111111
iter: 2 , chi= 973.881 , Lambda= 1.23457e-05
problem solve cost: 5.99562 ms
   makeHessian cost: 4.91672 ms
-------After optimization, we got these parameters :
0.999547  2.00672 0.968013
-------ground truth:
1.0,  2.0,  1.0
```

1.3 实现更优秀的阻尼因子策略，给出实验对比

论文《The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems》中有三种阻尼因子策略，如下图所示：

1. $\lambda_0 = \lambda_o$; λ_o is user-specified [8].
   use eq'n (13) for $\mathbf{h}_{\mathsf{lm}}$ 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. $\lambda_0 = \lambda_o \max\left[\mathsf{diag}[\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J}]\right]$; λ_o is user-specified.
   use eq'n (12) for $\mathbf{h}_{\mathsf{lm}}$ and eq'n (15) for ρ
   $\alpha = \left(\left(\mathbf{J}^\mathsf{T}\mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}))\right)^\mathsf{T}\mathbf{h}\right) / \left((\chi^2(\mathbf{p}+\mathbf{h}) - \chi^2(\mathbf{p}))/2 + 2\left(\mathbf{J}^\mathsf{T}\mathbf{W}(\mathbf{y} - \hat{\mathbf{y}}(\mathbf{p}))\right)^\mathsf{T}\mathbf{h}\right)$;
   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. $\lambda_0 = \lambda_o \max\left[\mathsf{diag}[\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J}]\right]$; λ_o is user-specified [9].
   use eq'n (12) for $\mathbf{h}_{\mathsf{lm}}$ 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: $\lambda_{i+1} = \lambda_i \nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

原代码中采用的是第3种策略，即Nielsen策略。下面在原代码problem.cc基础上实现论文中第1种阻尼因子更新策略，修改代码为：

```
void Problem::ComputeLambdaInitLM()
{
    currentChi_ = 0.0;
    // TODO:: robust cost chi2
    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 倍
    currentLambda_ = 1e-3;
}

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) += currentLambda_ * Hessian_(i, i);
    }
}

void Problem::RemoveLambdaHessianLM()
{
    ulong size = Hessian_.cols();
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
    for (ulong i = 0; i < size; ++i)
    {
        Hessian_(i, i) /= 1.0 + currentLambda_;
    }
}

bool Problem::IsGoodStepInLM()
{
    // 统计所有的残差
    double tempChi = 0.0;
    for (auto edge : edges_)
    {
        edge.second->ComputeResidual();
        tempChi += edge.second->Chi2();
    }
    // compute rho
    assert(Hessian_.rows() == Hessian_.cols() && "Hessian is not square");
```

```
    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 = delta_x_.transpose() *
        (currentLambda_ * diag_hessian * delta_x_ + b_);
    double rho = (currentChi_ - tempChi) / scale;
    // update currentLambda_
    double epsilon = 0.0;
    double L_down = 9.0;
    double L_up = 11.0;
    if (rho > epsilon && isfinite(tempChi))
    {
        currentLambda_ = std::max(currentLambda_ / L_down, 1e-7);
        currentChi_ = tempChi;
        return true;
    }
    else
    {
        currentLambda_ = std::min(currentLambda_ * L_up, 1e7);
        return false;
    }
}
```

代码运行结果为：

```
xwl@xwl-Inspiron-15-7000-Gaming:~/Documents/VSLAM-fundamentals-and-VIO-learning/L12/CurveFitting_LM/build/app$ ./testCurveFitting

Test CurveFitting start...
iter: 0 , chi= 3.21386e+06 , Lambda= 0.001
iter: 1 , chi= 1001.43 , Lambda= 0.000111111
iter: 2 , chi= 973.884 , Lambda= 1.23457e-05
iter: 3 , chi= 973.88 , Lambda= 1.37174e-06
problem solve cost: 5.90161 ms
   makeHessian cost: 4.76683 ms
-------After optimization, we got these parameters :
0.999589  2.00629 0.968815
-------ground truth:
1.0,  2.0,  1.0
```

同学们可以自行比较一下两种方法的精度、迭代次数、运行耗时等性能，在实际运用的过程中合理选择使用哪种方法。

2.公式推导

1.推导f_{15}

这里需要注意两点：

- 对谁加扰动，扰动项就直接跟在该变量后面

- 只有旋转变量加扰动才是乘一个微小旋转，其他均是加号

  推导过程如下：

$$f_{15} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial \delta b_k^g} = \frac{1}{4} \frac{\partial q_{b_i b_k} \otimes \begin{bmatrix} 1 \\ \frac{1}{2}(\omega - \delta b_k^g)\delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a) \cdot \delta t^2}{\partial \delta b_k^g}$$

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

$$= \frac{1}{4} \frac{\partial R_{b_i b_k} \left(\exp([\omega \delta t]_\times) \exp([-J_r(\omega\delta t)\delta b_k^g \delta t]_\times)(a^{b_{k+1}} - b_k^a) \delta t^2\right.}{\partial \delta b_k^g}$$

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

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

当 $\omega\delta t$ 极小时，$J_r(\omega\delta t) = I$，即有

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

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

2.推导g_{12}

$$g_{12} = \frac{\partial \alpha_{b_i b_{k+1}}}{\partial n_k^g} = \frac{1}{4} \frac{\partial q_{b_i b_k} \begin{bmatrix} 1 \\ \frac{1}{2}(\omega + \frac{1}{2}\delta n_k^g)\delta t \end{bmatrix} (a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta n_k^g}$$

$$= \frac{1}{4} \frac{\partial R_{b_i b_k} \cdot \exp\left(\left[(\omega + \frac{1}{2}\delta n_k^g)\delta t\right]_\times\right)(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta n_k^g}$$

$$= \frac{1}{4} \frac{\partial R_{b_i b_k} \exp\left([\omega \delta t]_\times\right) \exp\left(\left[J_r(\omega\delta t)\frac{1}{2}\delta n_k^g \delta t\right]_\times\right)(a^{b_{k+1}} - b_k^a)\delta t^2}{\partial \delta n_k^g}$$

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

$$= -\frac{1}{4}\left(R_{b_i b_{k+1}}\left[(a^{b_{k+1}} - b_k^a)\right]_\times \delta t^2\right)\left(\underbrace{J_r(\omega\delta t)}_{\text{近似为 } I}\frac{1}{2}\delta t\right)$$

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

3.证明式(9)

$$(J^TJ+\mu I)\Delta x_{LM}=-J^Tf \quad , \text{when } \mu \geq 0$$

由于半正定矩阵 J^TJ 是实对称矩阵，其特征值为 $\{\lambda_i\}$，对应特征向量为 $\{v_i\}$，则有 $J^TJ=V\Lambda V^T$，V 为 J^TJ 特征向量组成的特征矩阵，且各向量之间相互正交。Λ 为 J^TJ 的特征值组成的对角矩阵，另，$F'(x)=(J^Tf)^T$

所以：

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

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

$$(V\Lambda V^T+V\mu IV^T)\Delta x_{LM}=-F'^T$$

$$[V(\Lambda+\mu I)V^T]\Delta x_{LM}=-F'^T$$

其中：$V = [v_1 \ v_2 \ \cdots \ v_n]$ $V^T = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix}$

$$\Lambda + \mu I = \begin{bmatrix} \lambda_1+\mu & & & \\ & \lambda_2+\mu & & \\ & & \ddots & \\ & & & \lambda_n+\mu \end{bmatrix}$$

所以：

$$[v_1 \ v_2 \cdots v_n] \begin{bmatrix} \lambda_1+\mu & & & \\ & \lambda_2+\mu & & \\ & & \ddots & \\ & & & \lambda_n+\mu \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix} \Delta x_{lm} = -F'^T$$

由于 V，$\Lambda+\mu I$，V^T 均可逆，且 $V^{-1} = V^T$，有：

$$\Delta x_{lm} = -\sum_{i=1}^{n} \frac{v_i^T F'^T}{\lambda_i + \mu} v_i$$

，证毕