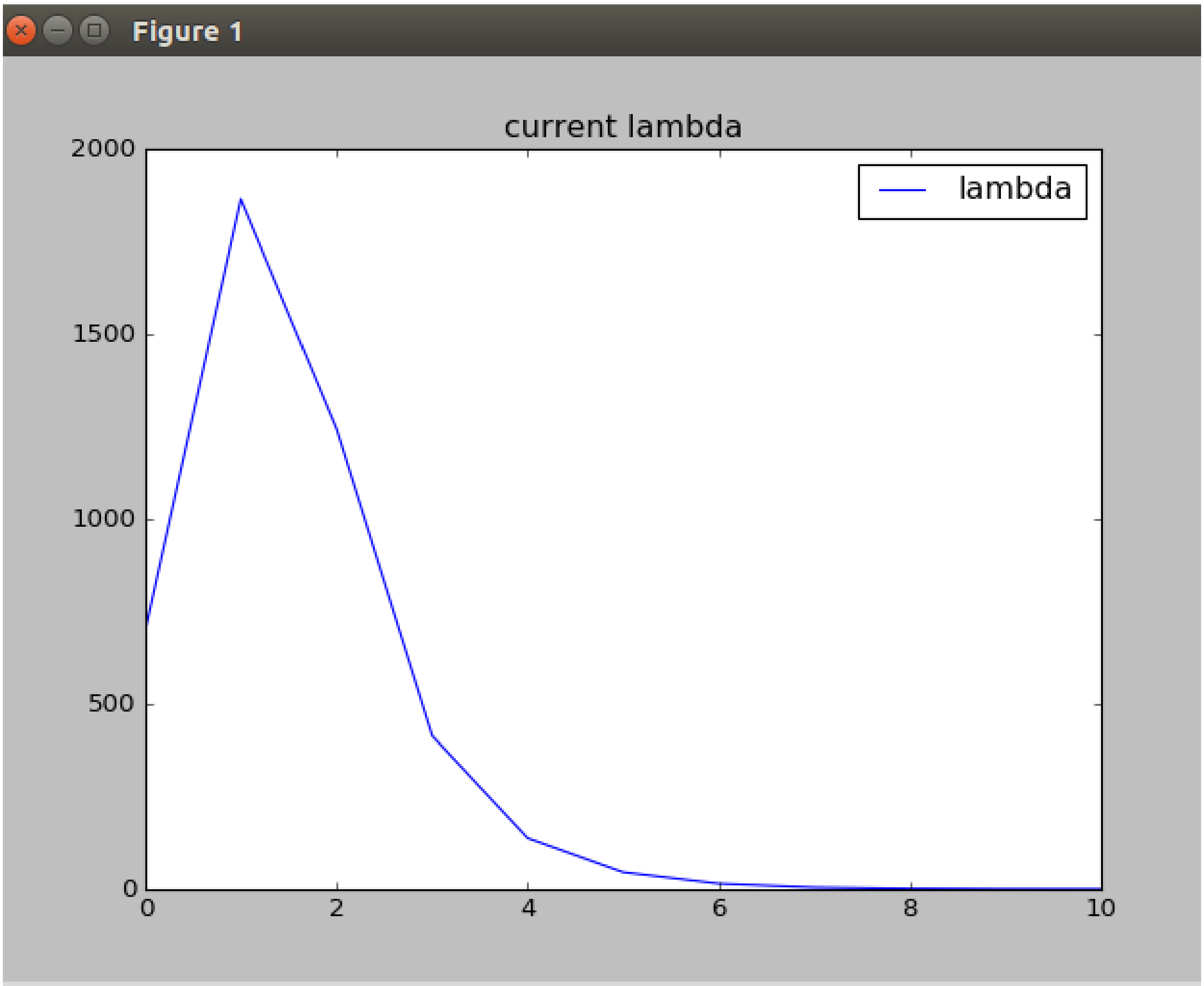
ex.1

1.使用LM算法估计曲线$y = \exp(ax^2 + bx + c)$的参数,绘制出阻尼因子的变化曲线，当噪声参数为1.0时，估计的参数误差较大，此处将噪声参数设为0.1，曲线上升阶段是最速下降法，后半段是高斯牛顿法：



2.使用LM算法估计曲线$y = ax^2 + bx + c$的参数,绘制出阻尼因子的变化曲线，发现噪声方差太大时，估计的参数误差较大，设定噪声方差为0.1。

2.1重新计算了残差和雅各比：

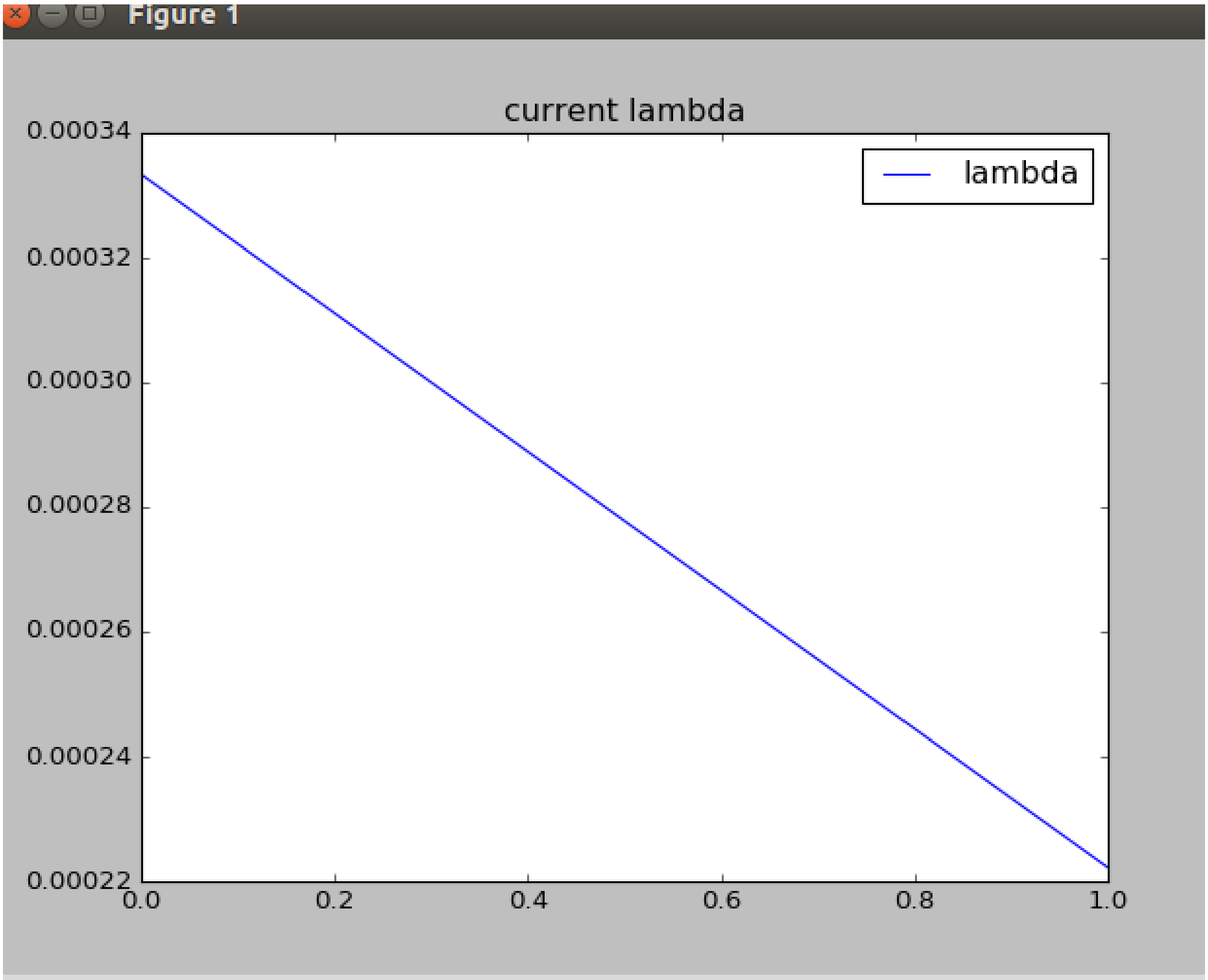
```cpp
// 计算曲线模型误差
virtual void ComputeResidual() override
{
    Vec3 abc = verticies_[0]->Parameters();  // 估计的参数
    //residual_(0) = std::exp( abc(0)*x_*x_ + abc(1)*x_ + abc(2) ) - y_;  // 构建残差
    residual_(0) = abc(0)*x_*x_ + abc(1)*x_ + abc(2)  - y_;  // 构建残差
}

// 计算残差对变量的雅克比
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_ * exp_y, x_ * exp_y , 1 * exp_y;
    jaco_abc << x_* x_, x_, 1;
    jacobians_[0] = jaco_abc;
}
```

2.2 阻尼因子变化曲线只迭代了3次便收敛，但是有一定的误差，可以跟下面另外一种阻尼更新方式对比：

```
Test CurveFitting start...
iter: 0 , chi= 614.937 , Lambda= 0.001
iter: 1 , chi= 0.913952 , Lambda= 0.000333333
iter: 2 , chi= 0.91395 , Lambda= 0.000222222
Writing lambda to csv file succeed!
problem solve cost: 0.465367 ms
   makeHessian cost: 0.263073 ms
-------After optimization, we got these parameters :
 1.06107  1.96183 0.999517
-------ground truth:
1.0,  2.0,  1.0
```



3.示例代码中采用阻尼因子更新策略是方式3，我这里采用方式2作为阻尼因子的更新策略来估计曲线$y = ax^2 + bx + c$的参数，参考文章The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems：

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: $\lambda_{i+1} = \lambda_i \nu_i$; $\nu_{i+1} = 2\nu_i$;

实现的关键代码如下：

3.1 计算缩放因子和更新状态

```
-
// compute alpha
// 统计所有的残差
double tempChi = 0.0;
for (auto edge: edges_) {
    edge.second->ComputeResidual();
    tempChi += edge.second->Chi2();
}
double alpha = b_.transpose() * delta_x_;
alpha_ = alpha / ((tempChi - currentChi_) / 2 + 2*alpha);

// 更新状态量  X = X + alpha*X
UpdateStates();
```

```
void Problem::UpdateStates() {
    for (auto vertex: verticies_) {
        ulong idx = vertex.second->OrderingId();
        ulong dim = vertex.second->LocalDimension();
        VecX delta = delta_x_.segment(idx, dim);

        // 所有的参数 x 叠加一个增量  x_{k+1} = x_{k} + alpha*delta_x
        //vertex.second->Plus(delta);
        vertex.second->Plus(alpha_*delta);
    }
}
```

3.2 使用上述方式2更新阻尼因子：

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

    // 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();
    }

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

3.3 运行结果和阻尼因子变化曲线，迭代了12次收敛，有一定的误差：

```
Test CurveFitting start...
iter: 0 , chi= 614.937 , Lambda= 0.001
iter: 1 , chi= 154.423 , Lambda= 0.000666667
iter: 2 , chi= 39.2916 , Lambda= 0.000444444
iter: 3 , chi= 10.5084 , Lambda= 0.000296296
iter: 4 , chi= 3.31259 , Lambda= 0.000197531
iter: 5 , chi= 1.51361 , Lambda= 0.000131687
iter: 6 , chi= 1.06387 , Lambda= 8.77915e-05
iter: 7 , chi= 0.951429 , Lambda= 5.85277e-05
iter: 8 , chi= 0.92332 , Lambda= 3.90184e-05
iter: 9 , chi= 0.916293 , Lambda= 2.60123e-05
iter: 10 , chi= 0.914536 , Lambda= 1.73415e-05
iter: 11 , chi= 0.914097 , Lambda= 1.1561e-05
iter: 12 , chi= 0.913987 , Lambda= 7.70735e-06
Writing lambda to csv file succeed!
problem solve cost: 1.00309 ms
   makeHessian cost: 0.569093 ms
-------After optimization, we got these parameters :
 1.06081  1.96136 0.999273
-------ground truth:
1.0,  2.0,  1.0
```

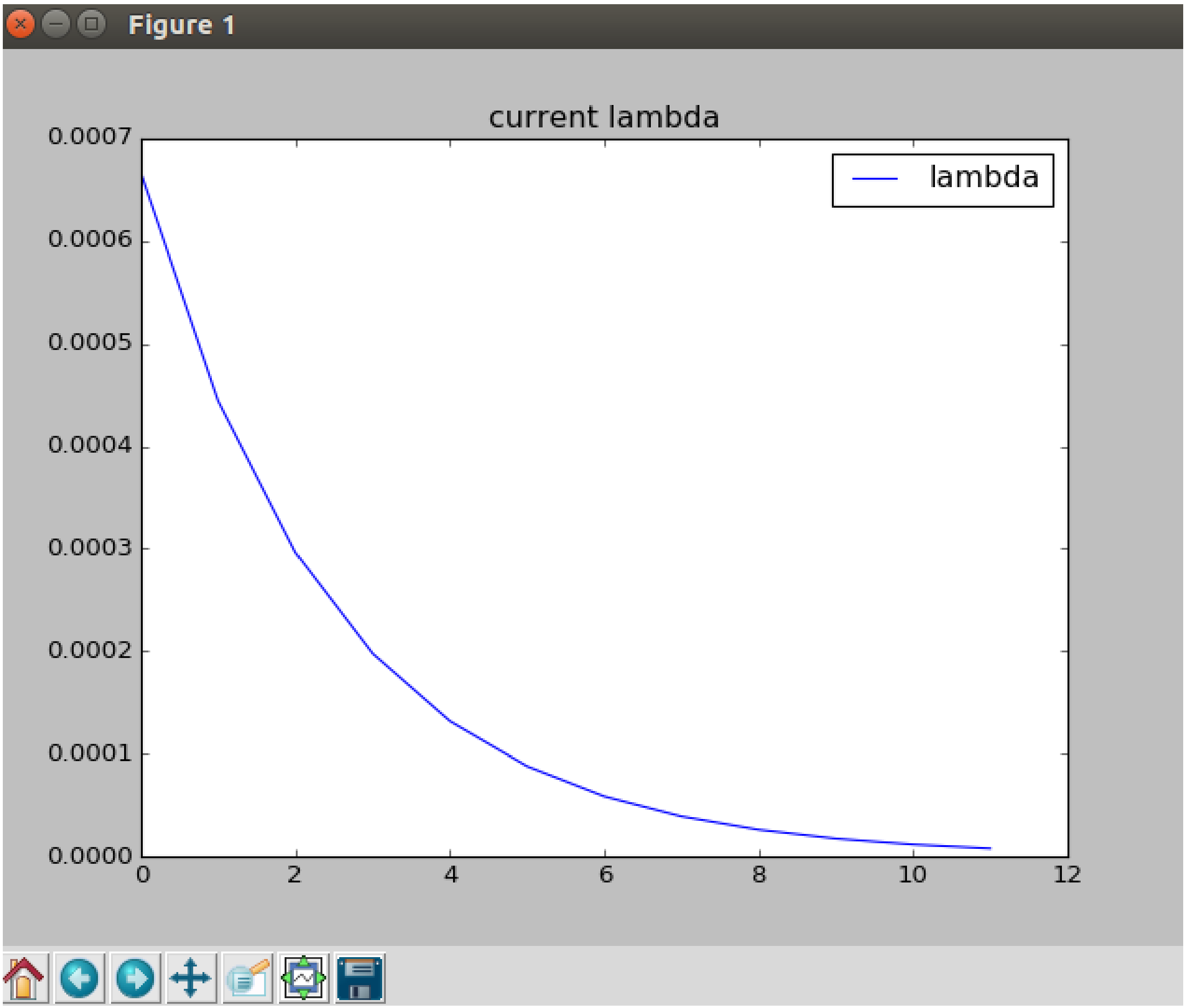


Figure 1
current lambda
lambda
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
2
4
6
8
10
12