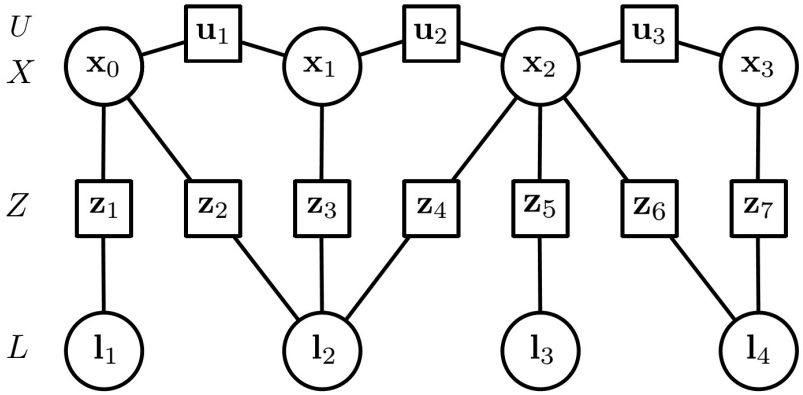
g2o简介

一.问题描述



(来源：Course On SLAM)

目标，求

$$X^*, L^* = argmax P(x_0) \prod_{i=1}^{M} P(x_i \mid x_{i-1}, u_i) \prod_{k=1}^{K} P(z_k \mid x_{i_k}, l_{j_k})$$

或者令

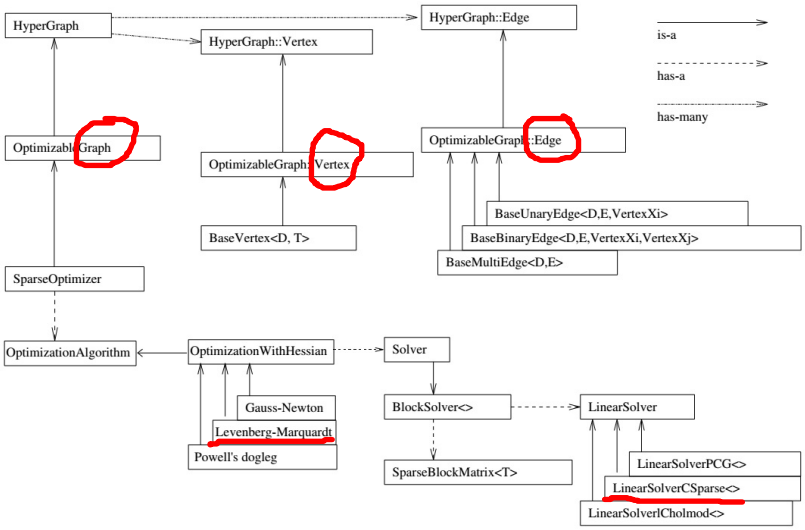
$$e_k(x_{i_k-1}, x_{i_k}) = f_{i_k}(x_{i_k-1}, u_{i_k}) - x_{i_k}$$

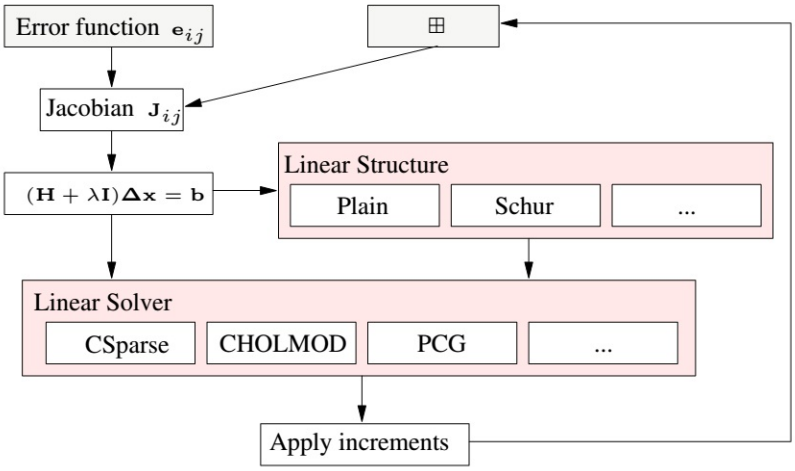
$$e_k(x_{i_k}, l_{j_k}) = h_k(x_{i_k}, l_{j_k}) - z_k$$

求

$$X^*, L^* = argmin \sum e_k(x_i, x_j)^T \Omega e_k(x_i, x_j) + \sum e_k(x_i, l_j)^T \Omega e_k(x_i, l_j)$$

g2o框架





(来源：g2o论文)

$$
\begin{aligned}
&\mathbf{F}_k(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}) \\
&\quad = \quad \mathbf{e}_k(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x})^T\boldsymbol{\Omega}_k\mathbf{e}_k(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}) \\
&\quad \simeq \quad (\mathbf{e}_k+\mathbf{J}_k\boldsymbol{\Delta}\mathbf{x})^T\,\boldsymbol{\Omega}_k\,(\mathbf{e}_k+\mathbf{J}_k\boldsymbol{\Delta}\mathbf{x}) \\
&\quad = \quad \underbrace{\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{e}_k}_{\mathbf{c}_k}+2\underbrace{\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{J}_k}_{\mathbf{b}_k}\boldsymbol{\Delta}\mathbf{x}+\boldsymbol{\Delta}\mathbf{x}^T\underbrace{\mathbf{J}_k^T\boldsymbol{\Omega}_k\mathbf{J}_k}_{\mathbf{H}_k}\boldsymbol{\Delta}\mathbf{x} \\
&\quad = \quad \mathbf{c}_k+2\mathbf{b}_k\boldsymbol{\Delta}\mathbf{x}+\boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}_k\boldsymbol{\Delta}\mathbf{x}
\end{aligned}
$$

解方程

$$H\Delta x^* = -b$$

当landmarks >> poses，Schur补：

$$\begin{pmatrix} H_{pp} & H_{pl} \\ H_{pl}^T & H_{ll} \end{pmatrix} \begin{pmatrix} \Delta x_p^* \\ \Delta x_l^* \end{pmatrix} = \begin{pmatrix} -b_p \\ -b_l \end{pmatrix}$$

两边分别乘以

$$\begin{pmatrix} 1 & -H_{pl}H_{ll}^{-1} \\ 0 & 1 \end{pmatrix}$$

得到

$$\begin{pmatrix} H_{pp} - H_{pl}H_{ll}^{-1}H_{pl}^T & 0 \\ H_{pl}^T & H_{ll} \end{pmatrix} \begin{pmatrix} \Delta x_p^* \\ \Delta x_l^* \end{pmatrix} = \begin{pmatrix} -b_p + H_{pl}H_{ll}b_l \\ b_l \end{pmatrix}$$

第一个方程：

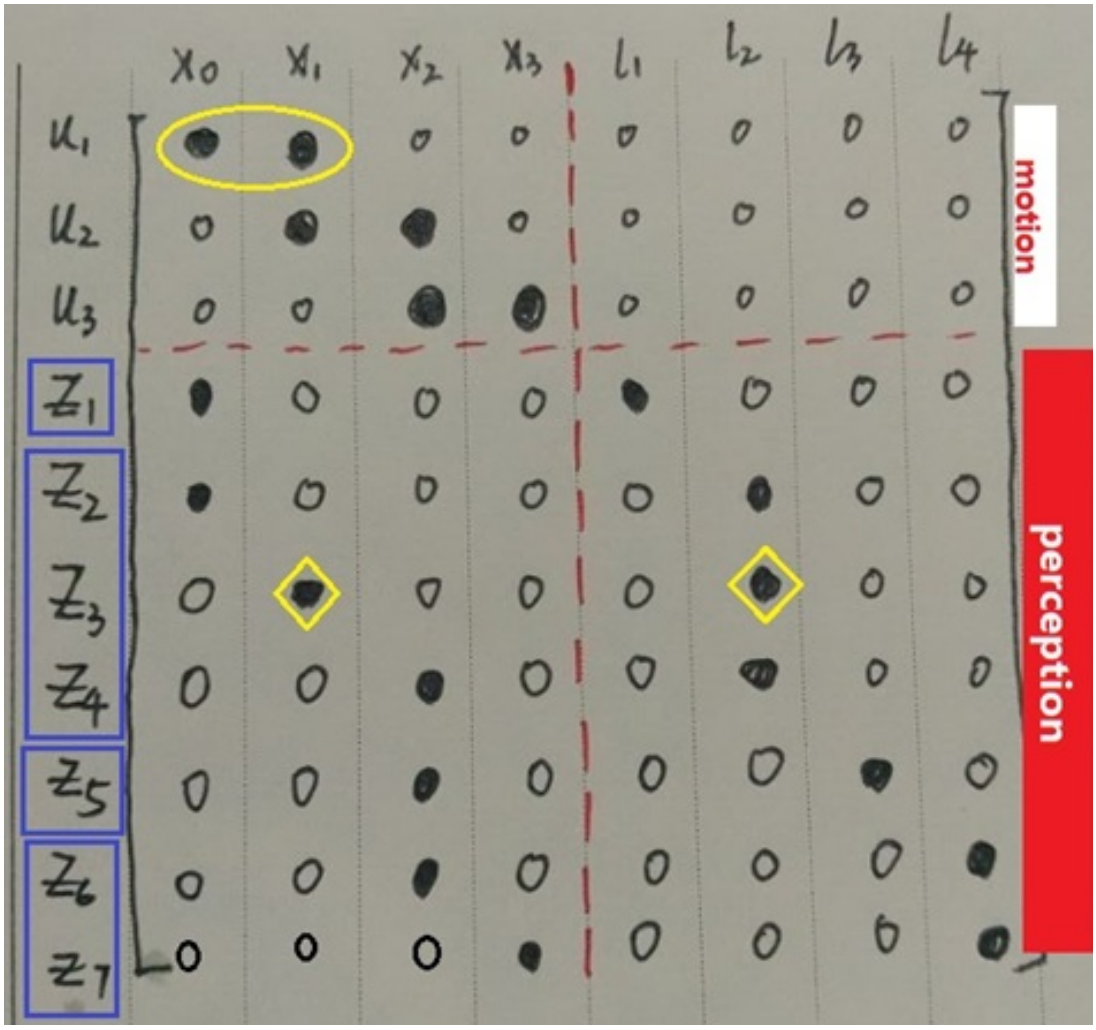
$$(H_{pp} - H_{pl}H_{ll}^{-1}H_{pl}^T)\Delta x_p^* = -b_p + H_{pl}H_{ll}^{-1}b_l$$

第二个方程

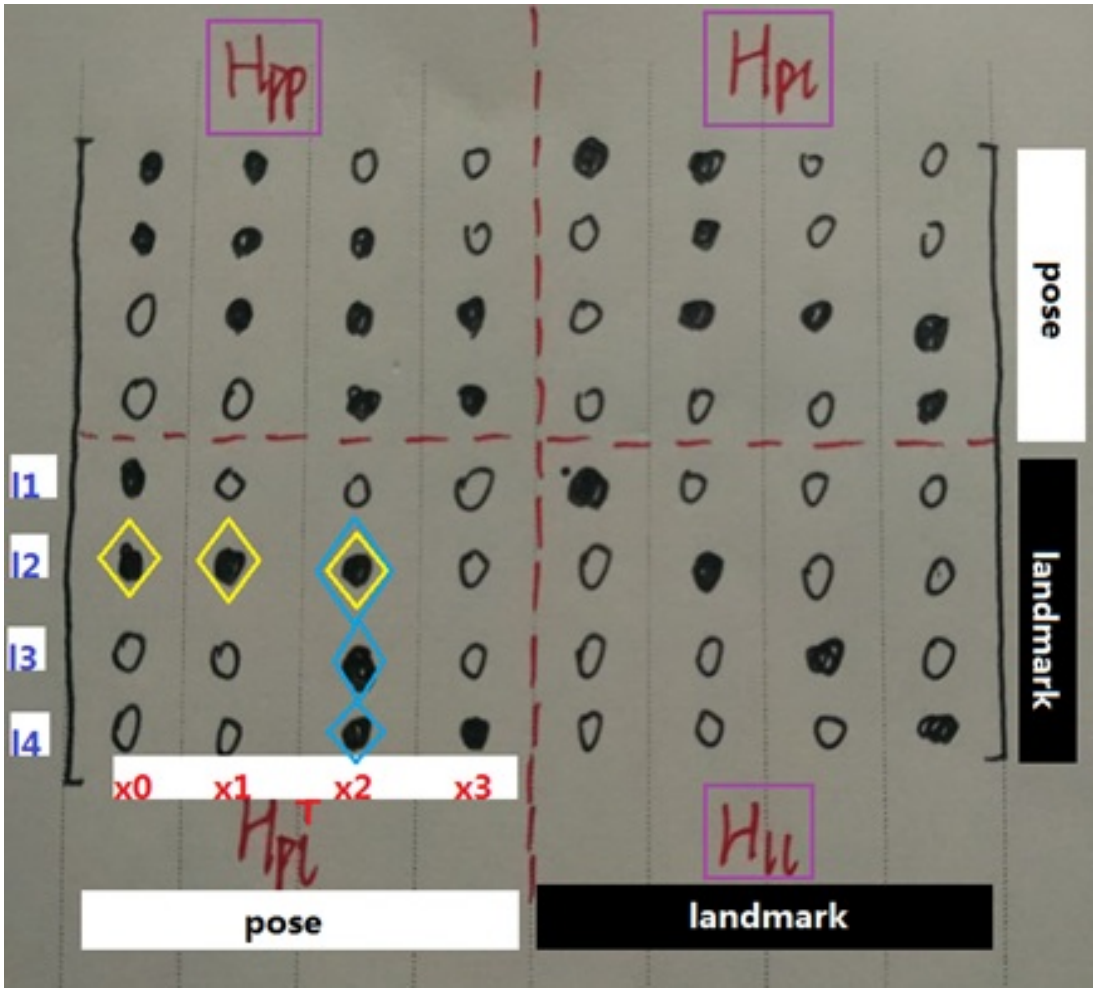
$$H_{ll}\Delta x_l^* = -b_l - H_{pl}^T\Delta x_p^*$$

Jacobian和Hessian的形式

Jacobian矩阵：



Hessian矩阵：



代码

一个简单的例子：tutorial_slam2d

```
int numNodes = 300;
Simulator simulator;
simulator.simulate(numNodes, sensorOffsetTransf);

/********************************************************************************
 * creating the optimization problem
 ********************************************************************************/
// dimension of the pose and landmark vertices (-1 if variable)
typedef BlockSolver< BlockSolverTraits<-1, -1> >  SlamBlockSolver;
typedef LinearSolverCSparse<SlamBlockSolver::PoseMatrixType> SlamLinearSolver;

// allocating the optimizer
SparseOptimizer optimizer;
SlamLinearSolver* linearSolver = new SlamLinearSolver();
linearSolver->setBlockOrdering(false);
SlamBlockSolver* blockSolver = new SlamBlockSolver(linearSolver);
OptimizationAlgorithmLevenberg* solver = new OptimizationAlgorithmLevenberg(blockSolve
r);
```

```
optimizer.setAlgorithm(solver);

// add the parameter representing the sensor offset
ParameterSE2Offset* sensorOffset = new ParameterSE2Offset;
sensorOffset->setOffset(sensorOffsetTransf);
sensorOffset->setId(0);
optimizer.addParameter(sensorOffset);

// adding the odometry to the optimizer
// first adding all the vertices
cerr << "Optimization: Adding robot poses ... ";
for (size_t i = 0; i < simulator.poses().size(); ++i)
{
  const Simulator::GridPose& p = simulator.poses()[i];
  const SE2& t = p.simulatorPose;
  VertexSE2* robot =  new VertexSE2;
  robot->setId(p.id);
  robot->setEstimate(t);
  optimizer.addVertex(robot);
}
cerr << "done." << endl;

// second add the odometry constraints
cerr << "Optimization: Adding odometry measurements ... ";
for (size_t i = 0; i < simulator.odometry().size(); ++i)
{
  const Simulator::GridEdge& simEdge = simulator.odometry()[i];

  EdgeSE2* odometry = new EdgeSE2;
  odometry->vertices()[0] = optimizer.vertex(simEdge.from);
  odometry->vertices()[1] = optimizer.vertex(simEdge.to);
  odometry->setMeasurement(simEdge.simulatorTransf);
  odometry->setInformation(simEdge.information);
  optimizer.addEdge(odometry); // second add the odometry constraints
}
cerr << "done." << endl;

// add the landmark observations
cerr << "Optimization: add landmark vertices ... ";
for (size_t i = 0; i < simulator.landmarks().size(); ++i)
{
  const Simulator::Landmark& l = simulator.landmarks()[i];
  VertexPointXY* landmark = new VertexPointXY;
  landmark->setId(l.id);
  landmark->setEstimate(l.simulatedPose);
```

```
    landmark->setMarginalized(true);   //Schur!!!!!
    optimizer.addVertex(landmark);
  }
  cerr << "done." << endl;

  cerr << "Optimization: add landmark observations ... ";
  for (size_t i = 0; i < simulator.landmarkObservations().size(); ++i)
  {
    const Simulator::LandmarkEdge& simEdge = simulator.landmarkObservations()[i];
    EdgeSE2PointXY* landmarkObservation =  new EdgeSE2PointXY;
    landmarkObservation->vertices()[0] = optimizer.vertex(simEdge.from);
    landmarkObservation->vertices()[1] = optimizer.vertex(simEdge.to);
    landmarkObservation->setMeasurement(simEdge.simulatorMeas);
    landmarkObservation->setInformation(simEdge.information);
    landmarkObservation->setParameterId(0, sensorOffset->id());
    optimizer.addEdge(landmarkObservation); // add landmark observations
  }
  cerr << "done." << endl;


  /*********************************************************************************
   * optimization
   ********************************************************************************/

  // dump initial state to the disk
  optimizer.save("tutorial_before.g2o");

  // prepare and run the optimization
  // fix the first robot pose to account for gauge freedom
  VertexSE2* firstRobotPose = dynamic_cast<VertexSE2*>(optimizer.vertex(0));
  firstRobotPose->setFixed(true);

  // prepare and run the optimization
  optimizer.setVerbose(true);

  cerr << "Optimizing" << endl;
  optimizer.initializeOptimization();
  optimizer.optimize(10);
  cerr << "done." << endl;

  optimizer.save("tutorial_after.g2o");

  // freeing the graph memory
  optimizer.clear();

  // destroy all the singletons
```

```
Factory::destroy();
OptimizationAlgorithmFactory::destroy();
HyperGraphActionLibrary::destroy();
```

注：`firstRobotPose->setFixed(true);`

> Local Accuracy **and Global** Consistency **for** Efficient Visual SLAM
>
> If one proceeds exactly as explained above, one would realise that the matrix $J^T\Lambda_z J$ is singular. This is caused by our decision to optimise over all poses $T1, \ldots, Tm$ and all points $y1, \ldots, yn$. To get a better understanding of this, let us first consider bundle adjustment using a calibrated stereo rig; i.e. the observations z are three dimensional and the prediction function is $\hat{z}stereo$. Now, let us assume that $\bar{T}_1, \ldots, \bar{T}_m, \bar{y}_1, \ldots, \bar{y}_n$ is the optimal solution to the BA minimization problem where the reprojection error of each point in each frame is minimal. It is important to note that all point measurements are relative, so that the solution does not depend on the absolute frame of reference. If we were now to apply a general rigid body transformation $A \in SE(3)$ to all parameters, the reprojection error would still be minimal. In other words, $A\bar{T}_1, \ldots, \bar{T}_m, A\bar{y}_1, \ldots, A\bar{y}_n$ is another optimal solution to the problem. Therefore, we do have a six dimensional solution space, and thus we say that there is a gauge freedom of six dimensions. That is the reason why the approximated Hessian is rank deficient. We can remove the gauge freedom by fixing six parameters, e.g. the first pose. Thus, we would optimize over the poses $T_2, \ldots, T_m$ and all points $y_1, \ldots, y_n$ while keeping T_1 fixed which now defines the reference frame.

四件事：
1. 设置优化算法OptimizationAlgorithmLevenberg和线性求解器LinearSolverCSparse
2. 设置vertices(poses and landmarks) 和 edges (odometry constraints and landmark observations)
3. `optimizer.initializeOptimization();`
4. optimizer.optimize(10);

部分关键数据和函数

optimizer.optimize(10)中的重要函数：

```
//OptimizationAlgorithm::SolverResult OptimizationAlgorithmLevenberg::solve(int iteration, bool online)
result = _algorithm->solve(i, online);
```

> 1. bool ok = _solver->buildStructure();
> 2. _optimizer->computeActiveErrors();
> 3. double currentChi = _optimizer->activeRobustChi2();
> 4. _solver->buildSystem();
> 5. _currentLambda = computeLambdaInit();
> 6. solver->setLambda(_currentLambda, **true**);
> 7. bool ok2 = _solver->solve();
> 8. _optimizer->update(_solver->x());
> 9. _optimizer->computeActiveErrors();
> 10. tempChi = _optimizer->activeRobustChi2();
> 11. Lambda Update

1. bool ok = _solver->buildStructure();

稀疏矩阵的基本技巧：不存储零元素，避免与零元素做加减乘除运算。

```
SparseBlockMatrix<PoseMatrixType>* _Hpp;
SparseBlockMatrix<LandmarkMatrixType>* _Hll;
SparseBlockMatrix<PoseLandmarkMatrixType>* _Hpl;

//第一个方程左边的系数矩阵 $(H_{pp}-H_{pl}H_{ll}^{-1}H_{pl}^{T})$
SparseBlockMatrix<PoseMatrixType>* _Hschur;

//$H_{ll}^{-1}$
SparseBlockMatrixDiagonal<LandmarkMatrixType>* _DInvSchur;

//$H_{pl}$
SparseBlockMatrixCCS<PoseLandmarkMatrixType>* _HplCCS;
SparseBlockMatrixCCS<PoseMatrixType>* _HschurTransposedCCS;
```

SparseBlockMatrix数据结构：

```
typedef std::map<int, SparseMatrixBlock*> IntBlockMap;
std::vector <IntBlockMap> _blockCols;
```

$$\begin{bmatrix} 3*2 & 3*2 & 3*9 \\ 3*2 & 3*2 & 3*9 \\ 2*2 & 2*2 & 2*9 \\ 4*9 & 4*2 & 4*9 \end{bmatrix}$$

_rowBlockIndices: [3, 6, 8, 12]
_colBlockIndices: [2, 4, 13]
例如block(0,0) -> 3 * 2, block(1,1) -> (6 – 3) * (4 – 2), block(3,2) -> (12 – 8) * (13 – 4)

SparseBlockMatrixCCS数据结构：

```
typedef std::unordered_map<int, MatrixType*> SparseColumn;
std::vector<SparseColumn> _blockCols;     ///< the matrices stored in CCS order
```

稀疏矩阵的存储：Compressed Column Storage (CCS)
比三元组更少的存储空间，处理大规模稀疏矩阵的首选数据结构。

$$A = \begin{bmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{bmatrix}.$$

val： 10 3 3 9 7 8 4 8 8…9 2 3 13 -1
row：1 2 4 2 3 5 6 3 4…5 6 2 5 6
col： 1 4 8 10 13 17 20
其中，val存储非0元素的值，row存储非0元素所在行数，col中[1,4)表示val中的前三个非0值在第一列，[4,8)表示val的第4个到第7个非0值在第2列。

buildStructure：配置稀疏矩阵的结构

> 1.allocate the diagonal on Hpp and Hll
> 2.create the structure in Hpp, Hll and in Hpl

以配置Hpp和Hll矩阵的对角为例：

```
// allocate the diagonal on Hpp and Hll
  int poseIdx = 0;
  int landmarkIdx = 0;
  for (size_t i = 0; i < _optimizer->indexMapping().size(); ++i)
{
    OptimizableGraph::Vertex* v = _optimizer->indexMapping()[i];
    if (! v->marginalized())//pose
{
      //assert(poseIdx == v->hessianIndex());
      PoseMatrixType* m = _Hpp->block(poseIdx, poseIdx, true);
      if (zeroBlocks)
        m->setZero();
      v->mapHessianMemory(m->data()); //将节点的Jacobian矩阵和系统矩阵H中的_Hpp对应的块关联起来
      ++poseIdx;
    }
else  //if v->marginalized() is true ,  v is a landmark
{
      LandmarkMatrixType* m = _Hll->block(landmarkIdx, landmarkIdx, true);
      if (zeroBlocks)
        m->setZero();
      v->mapHessianMemory(m->data());
      ++landmarkIdx;
    }
  }
```

> 3._Hpl->fillSparseBlockMatrixCCS(*_HplCCS); //copy into CCS structure*
> *4. _Hschur->fillSparseBlockMatrixCCSTransposed(*_HschurTransposedCCS); //copy as transposed into a CCS structure

时间消耗
pose: 100
landmarknum: 345
edges: 1647
buildStructure耗时：0.003703

2. _optimizer->computeActiveErrors();

takes the current estimate of the active vertices and for each active edge calls

时间消耗:
pose: 100
landmarknum: 345
edges: 1647
在一次迭代中：
time= 0.005264
computeActiveErrors耗时：0.000077

3. double currentChi = _optimizer->activeRobustChi2();

```
double SparseOptimizer::activeRobustChi2() const
{
  Vector3D rho;
  double chi = 0.0;
  for (EdgeContainer::const_iterator it = _activeEdges.begin(); it != _activeEdges.end(); ++it
)
  {
    const OptimizableGraph::Edge* e = *it;
    if (e->robustKernel()) //是否采用robustKernel
    {
      e->robustKernel()->robustify(e->chi2(), rho);
      chi += rho[0];
    }
    else
      chi += e->chi2();
  }
  return chi;
}
```

4. _solver->buildSystem();

1.计算每个edge的Jacobian，以二元边（假设两个节点是i和j）为例，就是 `_jacobianOplusXi` 和 `_jacobianOplusXj`。

> 注:如果没有定义这个边的linearizeOplus(),就会调用数值求导。（数值求导比较慢，效果。。。）
> 以orbslam，large_with_loop为例：
> (1).时间
> 跟踪运动模型时间统计：
> median trackwithMotion time: 0.015711(raw:0.003939)
> mean trackwithMotion time: 0.024777(raw:0.004080)
> 跟踪参考关键帧时间统计：
> median trackwithReference time: 0.0169646(raw:0.016892)

> mean trackwithReference time: 0.0192095(raw:0.016892)
> 跟踪局部地图时间统计：
> median localtrack time: 0.0121695(raw:0.008775)
> mean localtrack time: 0.0256535(raw:0.005065)
> LocalMapping时间统计：
> median localmapping time: 0.422528(raw:0.255696)
> mean localmapping time: 0.409984(raw:0.221395)
> (2).精度：
> absolute_translational_error.rmse 0.643333
> m (raw:0.32865391505)
> absolute_translational_error.mean 0.615756
> m (raw:0.319955279678)
> absolute_translational_error.median 0.622965
> m (raw:0.320324762114)
> absolute_translational_error.std 0.186338
> m (raw:0.0751133469098)
> absolute_translational_error.min 0.204926 m(raw: 0.115329009601)
> absolute_translational_error.max 1.215240 m(raw:0.508809133756)

```
e->linearizeOplus(jacobianWorkspace); // jacobian of the nodes' oplus (manifold)
```

2.然后计算每个边对应Jacobian矩阵，得到Hessien矩阵*H*

```
e->constructQuadraticForm();
```

实现为：

```
template <int D, typename E, typename VertexXiType, typename VertexXjType>
void BaseBinaryEdge<D, E, VertexXiType, VertexXjType>::constructQuadraticForm()
{
  VertexXiType* from = static_cast<VertexXiType*>(_vertices[0]);
  VertexXjType* to   = static_cast<VertexXjType*>(_vertices[1]);

  // get the Jacobian of the nodes in the manifold domain
  const JacobianXiOplusType& A = jacobianOplusXi();
  const JacobianXjOplusType& B = jacobianOplusXj();

  bool fromNotFixed = !(from->fixed());
  bool toNotFixed = !(to->fixed());

  if (fromNotFixed || toNotFixed) {
```

```
#ifdef G2O_OPENMP
    from->lockQuadraticForm();
    to->lockQuadraticForm();
#endif

    const InformationType& omega = _information;

    Eigen::Matrix<double, D, 1, Eigen::ColMajor> omega_r = - omega * _error;

    if (this->robustKernel() == 0) {
      if (fromNotFixed)
      {
        Eigen::Matrix<double, VertexXiType::Dimension, D, Eigen::ColMajor> AtO = A.transpos
e() * omega;
        from->b().noalias() += A.transpose() * omega_r;
        from->A().noalias() += AtO*A;
        if (toNotFixed )
        {
          if (_hessianRowMajor) // we have to write to the block as transposed
            _hessianTransposed.noalias() += B.transpose() * AtO.transpose();
          else
            _hessian.noalias() += AtO * B;
        }
      }
      if (toNotFixed)
      {
        to->b().noalias() += B.transpose() * omega_r;
        to->A().noalias() += B.transpose() * omega * B;
      }
    }
    else
    { // robust (weighted) error according to some kernel
      double error = this->chi2();
      Vector3D rho;
      this->robustKernel()->robustify(error, rho);
      InformationType weightedOmega = this->robustInformation(rho);
      //std::cout << PVAR(rho.transpose()) << std::endl;
      //std::cout << PVAR(weightedOmega) << std::endl;
      omega_r *= rho[1];
      if (fromNotFixed)
     {
        from->b().noalias() += A.transpose() * omega_r;
        from->A().noalias() += A.transpose() * weightedOmega * A;

        if (toNotFixed ) {
```

```
        if (_hessianRowMajor) // we have to write to the block as transposed
          _hessianTransposed.noalias() += B.transpose() * weightedOmega * A;
        else
          _hessian.noalias() += A.transpose() * weightedOmega * B;
      }
    }
    if (toNotFixed)
    {
      to->b().noalias() += B.transpose() * omega_r;
      to->A().noalias() += B.transpose() * weightedOmega * B;
    }
  }
#ifdef G2O_OPENMP
  to->unlockQuadraticForm();
  from->unlockQuadraticForm();
#endif
  }
}
```

3.构建系统的b矩阵

---

buildSystem时间消耗：

> pose: 100
> landmarknum: 345
> edges: 1647
>
> 在一次迭代中：
> time= 0.005264
> buildSystem耗时：0.003492

5. _currentLambda = computeLambdaInit();

```
double OptimizationAlgorithmLevenberg::computeLambdaInit() const
{
  if (_userLambdaInit->value() > 0)
    return _userLambdaInit->value();
  double maxDiagonal=0.;
  for (size_t k = 0; k < _optimizer->indexMapping().size(); k++) {
    OptimizableGraph::Vertex* v = _optimizer->indexMapping()[k];
    assert(v);
    int dim = v->dimension();
    for (int j = 0; j < dim; ++j){
      maxDiagonal = std::max(fabs(v->hessian(j,j)),maxDiagonal);
```

```
    }
  }
  return _tau*maxDiagonal;
}
```

6. solver->setLambda(_currentLambda, true);

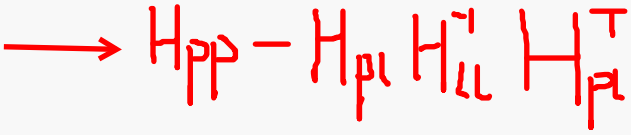
$$H = H + \lambda I$$

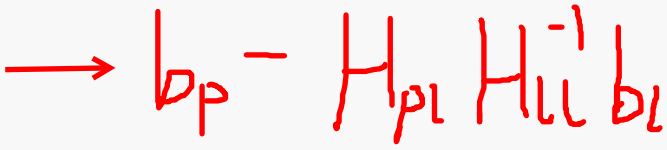
7. bool ok2 = _solver->solve();

```
template <typename Traits>
bool BlockSolver<Traits>::solve()
{
  //cerr << __PRETTY_FUNCTION__ << endl;
  if (! _doSchur)
  {
    double t=get_monotonic_time();
    bool ok = _linearSolver->solve(*_Hpp, _x, _b);
    G2OBatchStatistics* globalStats = G2OBatchStatistics::globalStats();
    if (globalStats)
    {
      globalStats->timeLinearSolver = get_monotonic_time() - t;
      globalStats->hessianDimension = globalStats->hessianPoseDimension = _Hpp->cols();
    }
    return ok;
  }

  // schur thing

  // backup the coefficient matrix
  double t=get_monotonic_time();

  // _Hschur = _Hpp, but keeping the pattern of _Hschur
  _Hschur->clear();
  _Hpp->add(_Hschur);

  //_DInvSchur->clear();
  memset (_coefficients, 0, _sizePoses*sizeof(double));
# ifdef G2O_OPENMP
# pragma omp parallel for default (shared) schedule(dynamic, 10)
# endif
  for (int landmarkIndex = 0; landmarkIndex < static_cast<int>(_Hll->blockCols().size()); ++l
andmarkIndex)
  {
```

```
    const typename SparseBlockMatrix<LandmarkMatrixType>::IntBlockMap& marginalizeCo
lumn = _Hll->blockCols()[landmarkIndex];
    assert(marginalizeColumn.size() == 1 && "more than one block in _Hll column");

    // calculate inverse block for the landmark
    const LandmarkMatrixType * D = marginalizeColumn.begin()->second;
    assert (D && D->rows()==D->cols() && "Error in landmark matrix");
    LandmarkMatrixType& Dinv = _DInvSchur->diagonal()[landmarkIndex];
    Dinv = D->inverse();
    LandmarkVectorType  db(D->rows());
    for (int j=0; j<D->rows(); ++j)
    {
      db[j]=_b[_Hll->rowBaseOfBlock(landmarkIndex) + _sizePoses + j];
    }
    db=Dinv*db;

    assert((size_t)landmarkIndex < _HplCCS->blockCols().size() && "Index out of bounds");
    const typename SparseBlockMatrixCCS<PoseLandmarkMatrixType>::SparseColumn& la
ndmarkColumn = _HplCCS->blockCols()[landmarkIndex];

    for (typename SparseBlockMatrixCCS<PoseLandmarkMatrixType>::SparseColumn::const
_iterator it_outer = landmarkColumn.begin();
         it_outer != landmarkColumn.end(); ++it_outer)
      {
      int i1 = it_outer->row;

      const PoseLandmarkMatrixType* Bi = it_outer->block;
      assert(Bi);

      PoseLandmarkMatrixType BDinv = (*Bi)*(Dinv);
      assert(_HplCCS->rowBaseOfBlock(i1) < _sizePoses && "Index out of bounds");
      typename PoseVectorType::MapType Bb(&_coefficients[_HplCCS->rowBaseOfBlock(i1)]
, Bi->rows());
#     ifdef G2O_OPENMP
      ScopedOpenMPMutex mutexLock(&_coefficientsMutex[i1]);
#     endif
      Bb.noalias() += (*Bi)*db;

      assert(i1 >= 0 && i1 < static_cast<int>(_HschurTransposedCCS->blockCols().size()) &&
 "Index out of bounds");
      typename SparseBlockMatrixCCS<PoseMatrixType>::SparseColumn::iterator targetColu
mnIt = _HschurTransposedCCS->blockCols()[i1].begin();

      typename SparseBlockMatrixCCS<PoseLandmarkMatrixType>::RowBlock aux(i1, 0);
      typename SparseBlockMatrixCCS<PoseLandmarkMatrixType>::SparseColumn::const_it
```

```
erator it_inner = lower_bound(landmarkColumn.begin(), landmarkColumn.end(), aux);
      for (; it_inner != landmarkColumn.end(); ++it_inner)
      {
        int i2 = it_inner->row;
        const PoseLandmarkMatrixType* Bj = it_inner->block;
        assert(Bj);
        while (targetColumnIt->row < i2 /*&& targetColumnIt != _HschurTransposedCCS->bloc
kCols()[i1].end()*/)
          ++targetColumnIt;
        assert(targetColumnIt != _HschurTransposedCCS->blockCols()[i1].end() && targetColu
mnIt->row == i2 && "invalid iterator, something wrong with the matrix structure");
        PoseMatrixType* Hi1i2 = targetColumnIt->block;//_Hschur->block(i1,i2);
        assert(Hi1i2);
        (*Hi1i2).noalias() -= BDinv*Bj->transpose();
      }
    }
  }
  //cerr << "Solve [marginalize] = " <<  get_monotonic_time()-t << endl;

  // _bschur = _b for calling solver, and not touching _b
  memcpy(_bschur, _b, _sizePoses * sizeof(double));
  for (int i=0; i<_sizePoses; ++i)
  {
    _bschur[i]-=_coefficients[i];
  }

  G2OBatchStatistics* globalStats = G2OBatchStatistics::globalStats();
  if (globalStats)
  {
    globalStats->timeSchurComplement = get_monotonic_time() - t;
  }

  t=get_monotonic_time();
  //解第一个方程
  bool solvedPoses = _linearSolver->solve(*_Hschur, _x, _bschur);
  if (globalStats) {
    globalStats->timeLinearSolver = get_monotonic_time() - t;
    globalStats->hessianPoseDimension = _Hpp->cols();
    globalStats->hessianLandmarkDimension = _Hll->cols();
    globalStats->hessianDimension = globalStats->hessianPoseDimension + globalStats->hes
sianLandmarkDimension;
  }
  //cerr << "Solve [decompose and solve] = " <<  get_monotonic_time()-t << endl;

  if (! solvedPoses)
    return false;
```





```
  // _x contains the solution for the poses, now applying it to the landmarks to get the new par
t of the
  // solution;
  // 解第二个方程
  double* xp = _x;
  double* cp = _coefficients;

  double* xl=_x+_sizePoses;
  double* cl=_coefficients + _sizePoses;
  double* bl=_b+_sizePoses;

  // cp = -xp
  for (int i=0; i<_sizePoses; ++i)
    cp[i]=-xp[i];

  // cl = bl
  memcpy(cl,bl,_sizeLandmarks*sizeof(double));

  // cl = bl - Bt * xp
  //Bt->multiply(cl, cp);
  _HplCCS->rightMultiply(cl, cp);

  // xl = Dinv * cl
  memset(xl,0, _sizeLandmarks*sizeof(double));
  _DInvSchur->multiply(xl,cl);
  //_DInvSchur->rightMultiply(xl,cl);
  //cerr << "Solve [landmark delta] = " <<  get_monotonic_time()-t << endl;

  return true;
}
```

其中，解方程组：

```
_linearSolver->solve(*_Hschur, _x, _bschur)// _Hschur*_x = _bschur
```

ORBSLAM中使用的两类求解器，默认不重排序：

> 1. LinearSolverDense
> PoseOptimization 和 OptimizeSim3

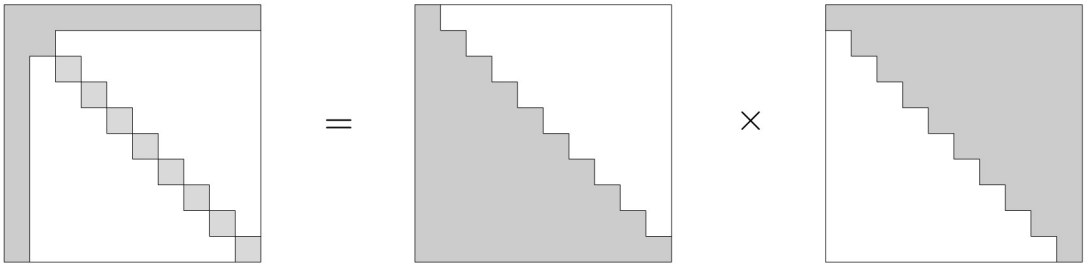
```
    bool solve(const SparseBlockMatrix<MatrixType>& A, double* x, double* b)
    {
```

```
      int n = A.cols();
      int m = A.cols();

      MatrixXD& H = _H;
      if (H.cols() != n) {
        H.resize(n, m);
        _reset = true;
      }
      if (_reset) {
        _reset = false;
        H.setZero();
      }

      // copy the sparse block matrix into a dense matrix
      int c_idx = 0;
      for (size_t i = 0; i < A.blockCols().size(); ++i) {
        int c_size = A.colsOfBlock(i);
        assert(c_idx == A.colBaseOfBlock(i) && "mismatch in block indices");

        const typename SparseBlockMatrix<MatrixType>::IntBlockMap& col = A.blockCols()[i
];
        if (col.size() > 0)
        {
          typename SparseBlockMatrix<MatrixType>::IntBlockMap::const_iterator it;
          for (it = col.begin(); it != col.end(); ++it)
          {
            int r_idx = A.rowBaseOfBlock(it->first);
            // only the upper triangular block is processed
            if (it->first <= (int)i)
            {
              int r_size = A.rowsOfBlock(it->first);
              H.block(r_idx, c_idx, r_size, c_size) = *(it->second);
              if (r_idx != c_idx) // write the lower triangular block
                H.block(c_idx, r_idx, c_size, r_size) = it->second->transpose();
            }
          }
        }

        c_idx += c_size;
      }

      // solving via Cholesky decomposition
      VectorXD::MapType xvec(x, m);
      VectorXD::ConstMapType bvec(b, n);
      //Compute / recompute the LDLT decomposition A = L D L^* = U^* D U of matrix
      _cholesky.compute(H);
```

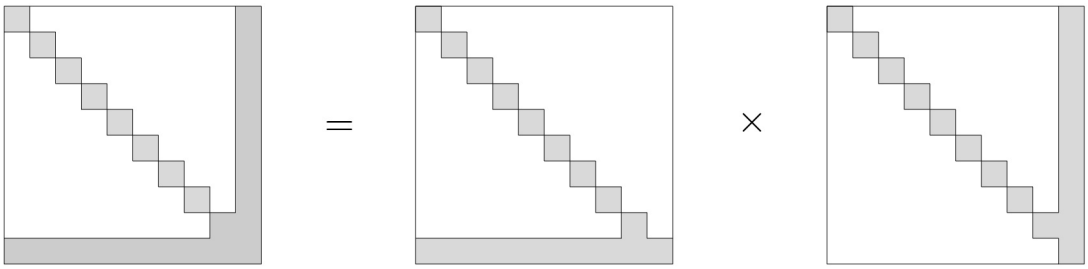
```
        if (_cholesky.isPositive())
        {
        /*
        template<typename Rhs >
        const internal::solve_retval < LDLT, Rhs >  solve (const MatrixBase< Rhs > &b) const
        Returns
        a solution x of $ A x = b $ using the current decomposition of A.
        This function also supports in-place solves using the syntax x = decompositionObject.so
lve(x) .

        This method just tries to find as good a solution as possible. If you want to check wheth
er a solution exists or if it is accurate, just call this function to get a result and then compute t
he error of this result, or use MatrixBase::isApprox() directly, for instance like this:

        bool a_solution_exists = (A*result).isApprox(b, precision);
        This method avoids dividing by zero, so that the non-existence of a solution doesn't by it
self mean that you'll get inf or nan values.

        More precisely, this method solves $ A x = b $ using the decomposition $ A = P^T L D L
^* P $ by solving the systems $ P^T y_1 = b $, $ L y_2 = y_1 $, $ D y_3 = y_2 $, $ L^* y_4 =
 y_3 $ and $ P x = y_4 $ in succession. If the matrix $ A $ is singular, then $ D $ will also be
singular (all the other matrices are invertible). In that case, the least-square solution of $ D y_
3 = y_2 $ is computed. This does not mean that this function computes the least-square solut
ion of $ A x = b $ is $ A $ is singular.
        */
          xvec = _cholesky.solve(bvec);
          return true;
        }
        return false;
      }
```

> 2. LinearSolverEigen ：
> BundleAdjustment，LocalBundleAdjustment和
> OptimizeEssentialGraph
> 主要过程：
> 1.计算置换矩阵P，减少分解带来的fill in。Replase A by PAP^T and b by Pb。

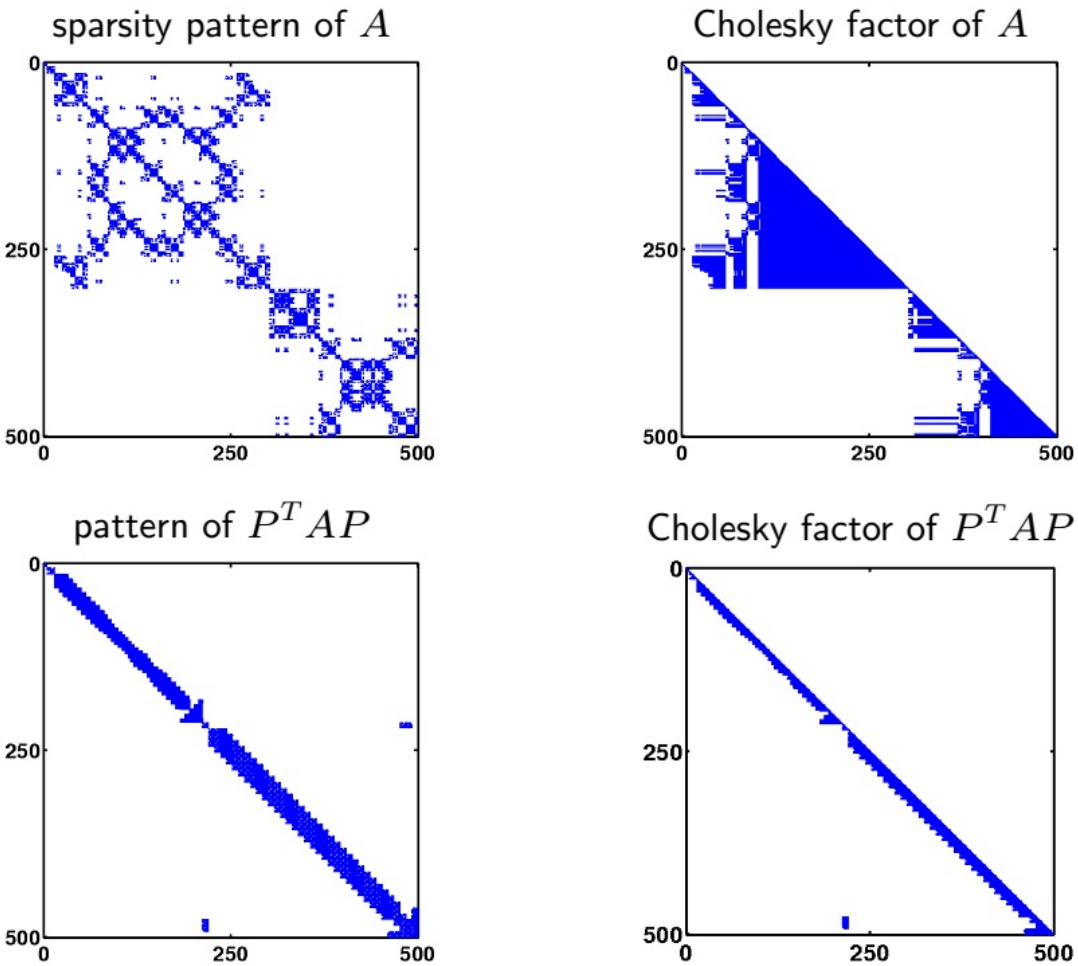
排序前的LU分解（Fill in！！！！！！！）：



排序后的LU分解：



排序前后的分解对比：



> 2.符号分解（Build the data structures necessary to compute and store factor）：
> 依据原矩阵的非零元分布(pattern),计算分解后因子矩阵的非零

> pattern，方便执行后续的数值分解。
> （消去树：`THE ROLE OF ELIMINATION TREES IN SPARSE FACTORIZATION`）
> 3.数值分解(Perform the factorization itselt, 这个阶段最耗时)：
> 4.解方程(Solve)

```
//Eigen::SimplicialLDLT<SparseMatrix, Eigen::Upper> _cholesky;
bool solve(const SparseBlockMatrix<MatrixType>& A, double* x, double* b)
{
  if (_init)
    _sparseMatrix.resize(A.rows(), A.cols());
  fillSparseMatrix(A, !_init);
  if (_init)
  {
  // compute the symbolic composition once
  // 只算一次符号分解，computeSymbolicDecomposition中默认不排序
  /*
  * compute the symbolic decompostion of the matrix only once.
  * Since A has the same pattern in all the iterations, we only
  * compute the fill-in reducing ordering once and re-use for all
  * the following iterations.
  */
    computeSymbolicDecomposition(A);
  }
  _init = false;

  double t=get_monotonic_time();

  /*factorize函数：
  Performs a numeric decomposition of matrix
  The given matrix must has the same sparcity than the matrix on which the symbolic deco
mposition has been performed.
  */
  _cholesky.factorize(_sparseMatrix);

  if (_cholesky.info() != Eigen::Success) { // the matrix is not positive definite
    if (_writeDebug) {
      std::cerr << "Cholesky failure, writing debug.txt (Hessian loadable by Octave)" << std::
endl;
      A.writeOctave("debug.txt");
    }
    return false;
  }
  // Solving the system
  VectorXD::MapType xx(x, _sparseMatrix.cols());
```

```
    VectorXD::ConstMapType bb(b, _sparseMatrix.cols());
    xx = _cholesky.solve(bb);
    G2OBatchStatistics* globalStats = G2OBatchStatistics::globalStats();
    if (globalStats) {
      globalStats->timeNumericDecomposition = get_monotonic_time() - t;
      globalStats->choleskyNNZ = _cholesky.matrixL().nestedExpression().nonZeros() + _spa
rseMatrix.cols(); // the elements of D
    }
    return true;
  }
```

其中的computeSymbolicDecomposition函数：

```
  /**
   * compute the symbolic decompostion of the matrix only once.
   * Since A has the same pattern in all the iterations, we only
   * compute the fill-in reducing ordering once and re-use for all
   * the following iterations.
   */
  void computeSymbolicDecomposition(const SparseBlockMatrix<MatrixType>& A)
  {
    double t=get_monotonic_time();
    //不排序,默认是这个，Course On SLAM也认为不需要再排序了，排序也不太会影响稀疏
性
    if (! _blockOrdering)
    {
      printf("_blockOrdering is false In computeSymbolicDecomposition\n");
      /*
      analyzePattern函数：
      Performs a symbolic decomposition on the sparcity of matrix.
      This function is particularly useful when solving for several problems having the same st
ructure.
      */
      _cholesky.analyzePattern(_sparseMatrix);
    }
    //排序
    else
    {
      // block ordering with the Eigen Interface
      // This is really ugly currently, as it calls internal functions from Eigen
      // and modifies the SparseMatrix class
      //置换矩阵P
      Eigen::PermutationMatrix<Eigen::Dynamic,Eigen::Dynamic> blockP;
      {
        // prepare a block structure matrix for calling AMD
```

```
        //三元组结构
        std::vector<Triplet> triplets;
        for (size_t c = 0; c < A.blockCols().size(); ++c)
        {
          const typename SparseBlockMatrix<MatrixType>::IntBlockMap& column = A.blockC
ols()[c];
          for (typename SparseBlockMatrix<MatrixType>::IntBlockMap::const_iterator it = col
umn.begin(); it != column.end(); ++it)
      {
            const int& r = it->first;
            if (r > static_cast<int>(c)) // only upper triangle
              break;
            triplets.push_back(Triplet(r, c, 0.));
          }
        }
        // call the AMD ordering on the block matrix.
        // Relies on Eigen's internal stuff, probably bad idea
        SparseMatrix auxBlockMatrix(A.blockCols().size(), A.blockCols().size());
        auxBlockMatrix.setFromTriplets(triplets.begin(), triplets.end());
        //Eigen::SimplicialLDLT<SparseMatrix, Eigen::Upper>()
        typename CholeskyDecomposition::CholMatrixType C;
        C = auxBlockMatrix.selfadjointView<Eigen::Upper>();
        //Approximate Minimum Degree Reordering
        Eigen::internal::minimum_degree_ordering(C, blockP);
      }

      int rows = A.rows();
      assert(rows == A.cols() && "Matrix A is not square");

      // Adapt the block permutation to the scalar matrix
      PermutationMatrix scalarP;
      scalarP.resize(rows);
      int scalarIdx = 0;
      for (int i = 0; i < blockP.size(); ++i) {
        //a reference to the stored array representing the permutation.
        const int& p = blockP.indices()(i);
        int base  = A.colBaseOfBlock(p);
        int nCols = A.colsOfBlock(p);
        for (int j = 0; j < nCols; ++j)
          //indices() : a reference to the stored array representing the permutation.
          scalarP.indices()(scalarIdx++) = base++;
      }
      assert(scalarIdx == rows && "did not completely fill the permutation matrix");
      // analyze with the scalar permutatio $PHP^{T}$
      _cholesky.analyzePatternWithPermutation(_sparseMatrix, scalarP);
```

```
  }
  G2OBatchStatistics* globalStats = G2OBatchStatistics::globalStats();
  if (globalStats)
    globalStats->timeSymbolicDecomposition = get_monotonic_time() - t;
}
```

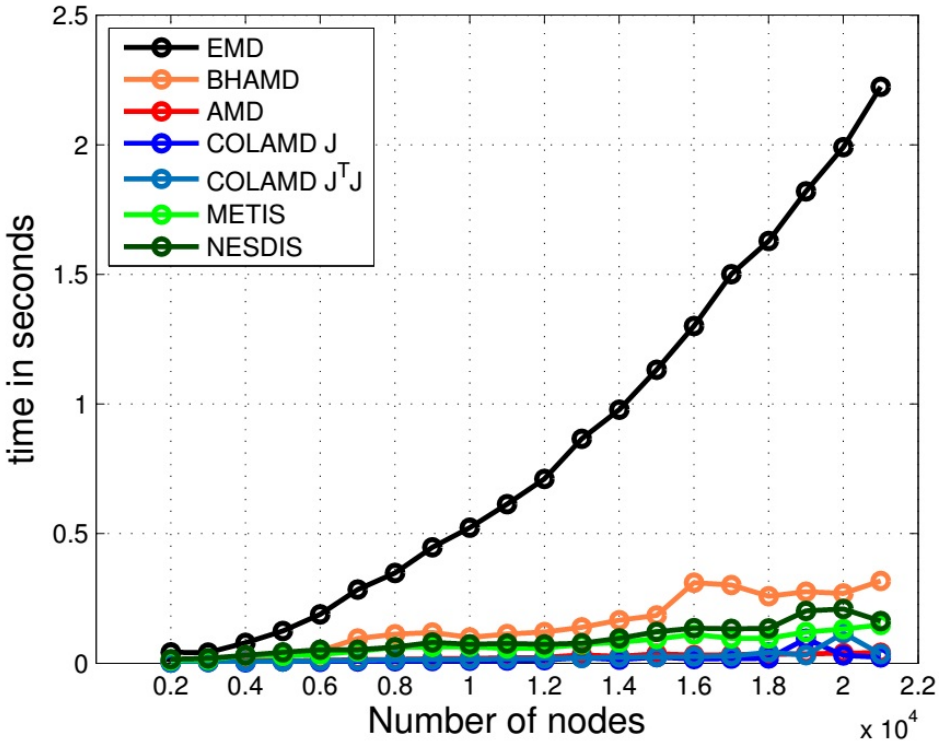
LinerSolveCSparse与LinerSolveEigen类似，默认_blockOrdering = true;：

```
bool solve(const SparseBlockMatrix<MatrixType>& A, double* x, double* b)
{
  fillCSparse(A, _symbolicDecomposition != 0);
  // perform symbolic cholesky once
  if (_symbolicDecomposition == 0)
  {
    computeSymbolicDecomposition(A);
  }
  // re-allocate the temporary workspace for cholesky
  if (_csWorkspaceSize < _ccsA->n) {
    _csWorkspaceSize = 2 * _ccsA->n;
    delete[] _csWorkspace;
    _csWorkspace = new double[_csWorkspaceSize];
    delete[] _csIntWorkspace;
    _csIntWorkspace = new int[2*_csWorkspaceSize];
  }

  double t=get_monotonic_time();
  // _x = _b for calling csparse
  if (x != b)
    memcpy(x, b, _ccsA->n * sizeof(double));
  /*
  int cs_cholsolsymb(const cs *A, double *b, const css* S, double* x, int* work)
  {
    csn *N ;
    int n, ok ;
    if (!CS_CSC (A) || !b || ! S || !x) {
      fprintf(stderr, "%s: No valid input!\n", __PRETTY_FUNCTION__);
      assert(0); // get a backtrace in debug mode
      return (0) ;     // check inputs
    }
    n = A->n ;
    N = cs_chol_workspace (A, S, work, x) ;                    // numeric Cholesky factorization
    if (!N) {
      fprintf(stderr, "%s: cholesky failed!\n", __PRETTY_FUNCTION__);
      //assert(0);
    }
```

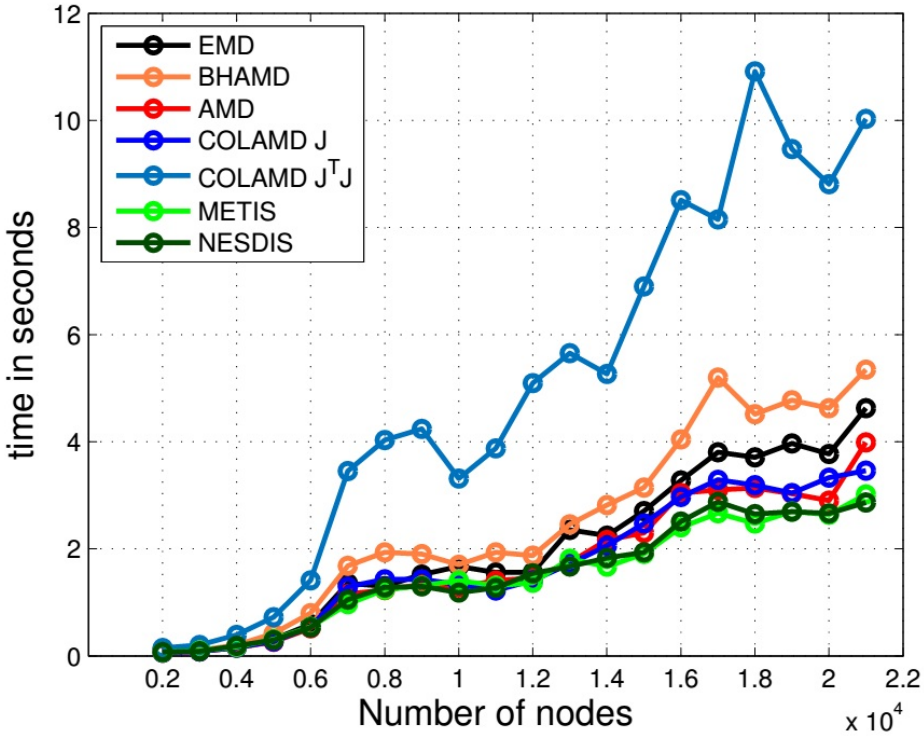
```
      ok = (N != NULL) ;
      if (ok)
      {
    //Solve $PAP^{T} Px = Pb =LL^{T}Px=Pb$
    //解方程
        cs_ipvec (S->pinv, b, x, n) ;   // x = P*b
        cs_lsolve (N->L, x) ;           // x = L\x
        cs_ltsolve (N->L, x) ;          // x = L'\x
        cs_pvec (S->pinv, x, b, n) ;    // b = P'*x
      }
      cs_nfree (N) ;
      return (ok) ;
    }
    */
    int ok = csparse_extension::cs_cholsolsymb(_ccsA, x, _symbolicDecomposition, _csWor
kspace, _csIntWorkspace);
    if (! ok) {
      if (_writeDebug) {
        std::cerr << "Cholesky failure, writing debug.txt (Hessian loadable by Octave)" << std::
endl;
        csparse_extension::writeCs2Octave("debug.txt", _ccsA, true);
      }
      return false;
    }

    G2OBatchStatistics* globalStats = G2OBatchStatistics::globalStats();
    if (globalStats){
      globalStats->timeNumericDecomposition = get_monotonic_time() - t;
      globalStats->choleskyNNZ = static_cast<size_t>(_symbolicDecomposition->lnz);
    }

    return ok != 0;
  }
```

在Variable reordering strategies for SLAM(Pratik Agarwal and Edwin Olson)中，不同排序方法的对比:

> 各种算法的重排序时间：



各种算法的solve时间：



结论：This experiment provides evidence that small modifications to minimum-degree type variable ordering algorithms may not be able to achieve significant improvements. An experiment like this cannot be conclusive, but it does suggest that different ideas may be required to improve performance.

Matlab中$x = H \setminus b$的反斜线求解器，自动判断矩阵类型，选择不同的求解算法：

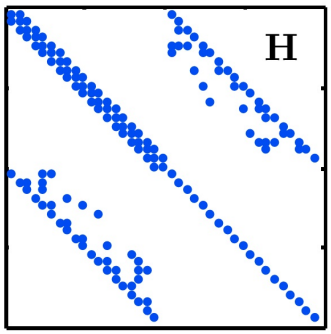
表 3-3 “反斜线”求解器的内部算法框架

矩阵 A(按如下优先顺序)	求 解 算 法
稀疏矩阵，且为对角阵	右端项元素除以矩阵对角元
较稠密的带状矩阵	部分选主元的带状矩阵 LU 分解与回代(LAPACK 软件包)
上三角或下三角矩阵	回代法或前代法
对三角矩阵作行排列形成的矩阵	重排序后用回代或前代法
对称矩阵，且对角线元素大于零	通过 Cholesky 分解算法检查正定性，对稠密矩阵和稀疏矩阵分别用 LAPACK 软件包和 CHOLMOD 软件包. 若成功再回代求解，若稠密且不正定，使用选主元的对称矩阵求解算法(LAPACK 软件包)
稠密的上黑森伯格(Hessenberg)矩阵	执行高斯消去变为上三角矩阵再回代求解

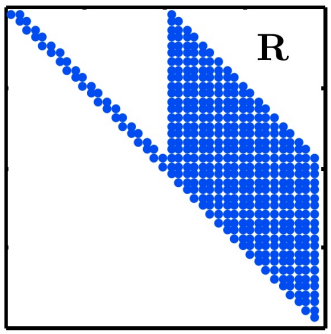
矩阵 A(按如下优先顺序)	求 解 算 法
一般的稀疏方阵	针对稀疏矩阵的直接解法(UMFPACK 软件包)
一般的稠密方阵	部分主元 LU 分解(LAPACK 软件包)
不是方阵	通过矩阵 QR 分解得到最小二乘解，分稠密矩阵与稀疏矩阵两种情况

然而在Course On SLAM中：

不采用Schur，排序之前：

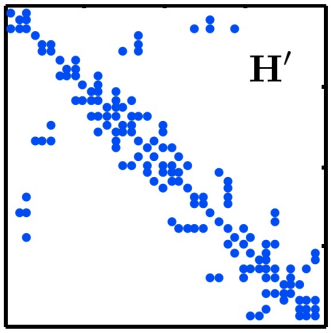


157

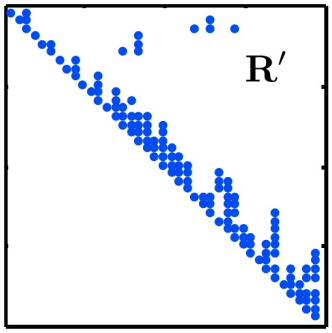


438

不采用Schur，排序之后：

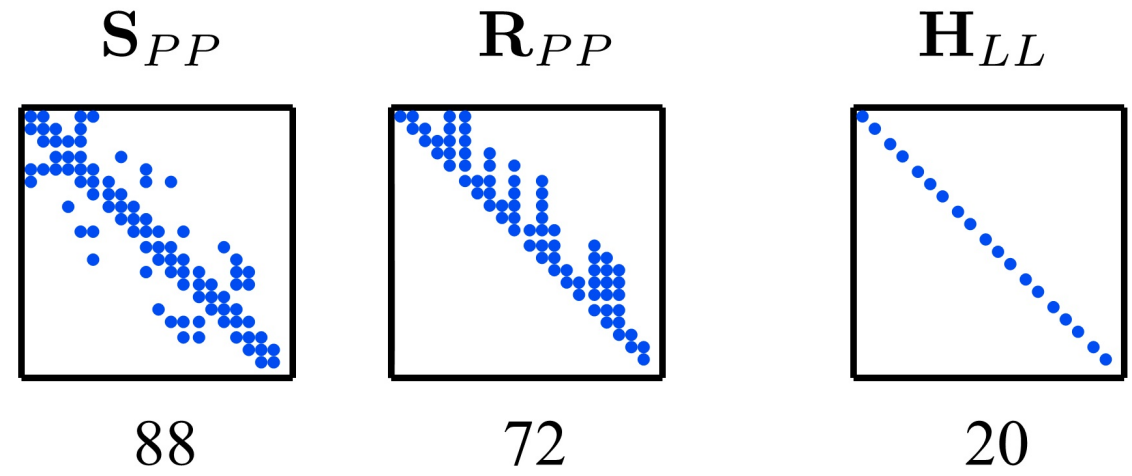


157

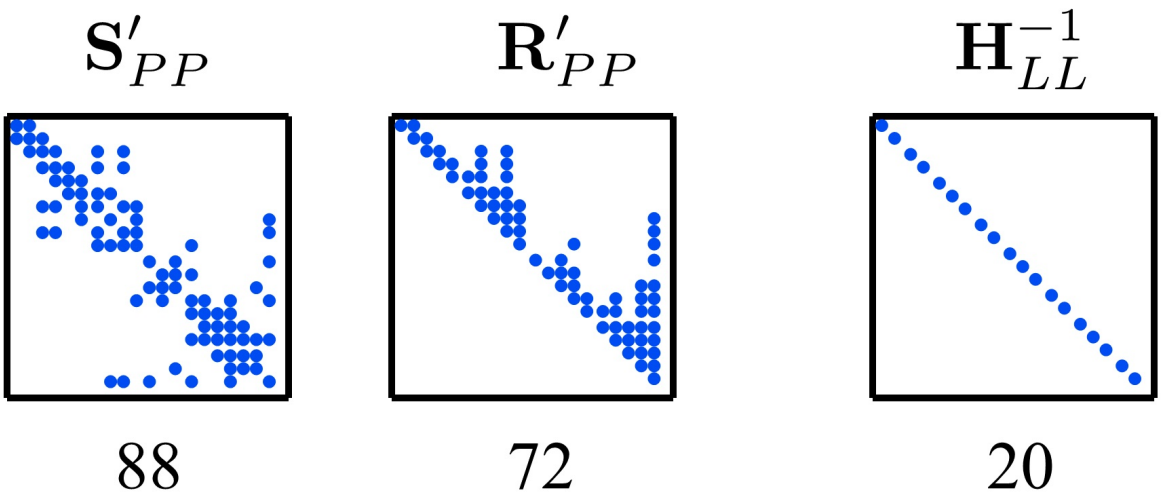


132

采用Schur，排序之前：



采用Schur，排序之后：



可以看出，采用Schur补再排序，前后变化并不明显。但是相比没有采用Schur补的方法，问题规模要小很多。

时间消耗方面：

pose: 100
landmarknum: 345
edges: 1647

在一次迭代中：
time= 0.005264

solve()函数
准备数据(Schur补)耗时：0.001012
解第1个方程耗时：0.000325
解第二个方程耗时：0.000099

> 注:LinerSolveEigen 和 LinerSolveCSparse在solve()中的表现非常接近

8. _optimizer->update(_solver->x());

9. _optimizer->computeActiveErrors();

10. tempChi = _optimizer->activeRobustChi2();

11. Lambda Update

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

if (rho>0 && g2o_isfinite(tempChi)){ // last step was good
  double alpha = 1.-pow((2*rho-1),3);
  // crop lambda between minimum and maximum factors
  alpha = (std::min)(alpha, _goodStepUpperScale);
  //_goodStepLowerScale = 1./3.;
  double scaleFactor = (std::max)(_goodStepLowerScale, alpha);
  _currentLambda *= scaleFactor;
  _ni = 2;
  currentChi=tempChi;
  _optimizer->discardTop();
} else {
  _currentLambda*=_ni;
  _ni*=2;
  _optimizer->pop(); // restore the last state before trying to optimize
}
```

LM中Lambda的计算：

(The Levenberg-Marquardt method for nonlinear least squares curve-fitting problems)

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. $\lambda_0 = \lambda_o \max\left[\text{diag}[\mathbf{J}^\mathsf{T}\mathbf{W}\mathbf{J}]\right]$; 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 $\rho_i(\alpha\mathbf{h}) > \epsilon_4$: $\mathbf{p} \leftarrow \mathbf{p} + \alpha\mathbf{h}$; $\lambda_{i+1} = \max\left[\lambda_i/(1+\alpha), 10^{-7}\right]$;
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 $\rho_i(\mathbf{h}) > \epsilon_4$: $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{h}$; $\lambda_{i+1} = \lambda_i \max\left[1/3, 1 - (2\rho_i - 1)^3\right]$; $\nu_i = 2$;
otherwise: $\lambda_{i+1} = \lambda_i \nu_i$; $\quad \nu_{i+1} = 2\nu_i$;

In Local Accuracy and Global Consistency for Efficient Visual SLAM .P97

> Indeed, a sufficient numerical stability of even rank-deficient problems can be archived by applying a robust variant of Cholesky — such as the pivoted $LTDL$ decomposition used in the Eigen matrix library — and the Levenberg-Marquardt damping term, called Tikhonov regularisation (Tikhonov & Arsenin, 1977) in this context.

---

g2o + ceres:

> https://github.com/RainerKuemmerle/g2o/blob/master/g2o/examples/bal/bal_example.cpp
>
> https://github.com/JzHuai0108/ORB_SLAM/blob/master/g2o_types/IMU_constraint.cpp

例子的数据：

> pose:3
> landmarknum:27
> Add odometry measurements：[0–16] [16–25]
>
> Add landmark observations：[0–1] [0–2] [0–3] [0–4] [0–5] [0–6] [0–7] [0–8] [0–9] [0–10] [0–11] [0–12] [0–13] [0–14] [0–15]
> Add landmark observations：[16–17] [16–5] [16–7] [16–8] [16–9] [16–10] [16–18] [16–11] [16–19] [16–13] [16–20] [16–14] [16–21] [16–15] [16–22] [16–23] [16–24]
> Add landmark observations：[25–7] [25–8] [25–9] [25–19] [25–12] [25–13] [25–20] [25–14] [25–21] [25–22] [25–26] [25–24] [25–27] [25–28] [25–29]

_Hpp:
RBI: 3 3 6 9
CBI: 3 3 6 9
BLOCK: 0 0
7.056e+12 -599.377 -372.554
-599.377 7.056e+12 9300.66
-372.554 9300.66 7.056e+12
BLOCK: 0 1
-424.95 608.389 0
391.094 -9972.78 0
388.083 -9855.62 -820.702
BLOCK: 1 1
7.056e+12 -361.667 -728.556
-361.667 7.056e+12 9786.72
-728.556 9786.72 7.056e+12
BLOCK: 1 2
-400.132 -38.4342 0
-32.8619 -9999.87 0
-33.4301 -10152.1 -820.702
BLOCK: 2 2
7.056e+12 33.076 -1088.67
33.076 7.056e+12 -1361.9
-1088.67 -1361.9 7.056e+12

_Hll:
RBI: 27 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54
CBI: 27 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54
BLOCK: 0 0
7.056e+12 0
0 7.056e+12
BLOCK: 1 1
7.056e+12 -2.38838e-05
-2.38838e-05 7.056e+12
BLOCK: 2 2
7.056e+12 0
0 7.056e+12
BLOCK: 3 3

7.056e+12 5.24476e-06
5.24476e-06 7.056e+12
BLOCK: 4 4
7.056e+12 -3.82672e-05
-3.82672e-05 7.056e+12
BLOCK: 5 5
7.056e+12 2.91285e-05
2.91285e-05 7.056e+12
BLOCK: 6 6
7.056e+12 8.90826e-06
8.90826e-06 7.056e+12
BLOCK: 7 7
7.056e+12 4.47407e-05
4.47407e-05 7.056e+12
BLOCK: 8 8
7.056e+12 -7.32163e-06
-7.32163e-06 7.056e+12
BLOCK: 9 9
7.056e+12 -2.88327e-05
-2.88327e-05 7.056e+12
BLOCK: 10 10
7.056e+12 4.52602e-05
4.52602e-05 7.056e+12
BLOCK: 11 11
7.056e+12 0
0 7.056e+12
BLOCK: 12 12
7.056e+12 -7.95032e-05
-7.95032e-05 7.056e+12
BLOCK: 13 13
7.056e+12 -2.85862e-05
-2.85862e-05 7.056e+12
BLOCK: 14 14
7.056e+12 0
0 7.056e+12
BLOCK: 15 15
7.056e+12 -4.69052e-05
-4.69052e-05 7.056e+12
BLOCK: 16 16

7.056e+12 0
0 7.056e+12
BLOCK: 17 17
7.056e+12 -1.5704e-05
-1.5704e-05 7.056e+12
BLOCK: 18 18
7.056e+12 8.6073e-05
8.6073e-05 7.056e+12
BLOCK: 19 19
7.056e+12 -0.000112155
-0.000112155 7.056e+12
BLOCK: 20 20
7.056e+12 -4.43331e-05
-4.43331e-05 7.056e+12
BLOCK: 21 21
7.056e+12 -3.91929e-05
-3.91929e-05 7.056e+12
BLOCK: 22 22
7.056e+12 5.46622e-05
5.46622e-05 7.056e+12
BLOCK: 23 23
7.056e+12 -4.98747e-05
-4.98747e-05 7.056e+12
BLOCK: 24 24
7.056e+12 -8.34771e-05
-8.34771e-05 7.056e+12
BLOCK: 25 25
7.056e+12 -5.08193e-05
-5.08193e-05 7.056e+12
BLOCK: 26 26
7.056e+12 3.84388e-05
3.84388e-05 7.056e+12

_Hpl:
RBI: 3 3 6 9
CBI: 27 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54
BLOCK: 0 0
-399.91 -8.47297

8.47302 -399.91
-279.607 949.083
BLOCK: 0 1
-399.91 -8.47299
8.47304 -399.91
-24.2135 909.255
BLOCK: 0 2
-399.91 -8.47302
8.47298 -399.91
-621.345 607.728
BLOCK: 0 3
-399.91 -8.47303
8.473 -399.91
-431.124 227.54
BLOCK: 0 4
-399.91 -8.47302
8.47302 -399.91
550.851 328.426
BLOCK: 1 4
-399.646 -16.8267
16.8267 -399.646
540.9 723.508
BLOCK: 0 5
-399.91 -8.47296
8.47298 -399.91
-988.833 -151.825
BLOCK: 0 6
-399.91 -8.47301
8.47299 -399.91
-233.165 -107.996
BLOCK: 1 6
-399.646 -16.8268
16.8268 -399.646
-243.116 287.086
BLOCK: 2 6
-399.655 -16.6127
16.6127 -399.655
-260.131 692.82
BLOCK: 0 7

-399.91 -8.47301
8.47301 -399.91
138.121 131.383
BLOCK: 1 7
-399.646 -16.8268
16.8268 -399.646
128.17 526.465
BLOCK: 2 7
-399.655 -16.6127
16.6127 -399.655
111.155 932.199
BLOCK: 0 8
-399.91 -8.47299
8.47302 -399.91
226.512 162.506
BLOCK: 1 8
-399.646 -16.8268
16.8268 -399.646
216.561 557.588
BLOCK: 2 8
-399.655 -16.6127
16.6127 -399.655
199.546 963.322
BLOCK: 0 9
-399.91 -8.47304
8.47304 -399.91
687.145 4.47241
BLOCK: 1 9
-399.646 -16.8268
16.8268 -399.646
677.194 399.555
BLOCK: 0 10
-399.91 -8.47302
8.47298 -399.91
-539.502 -497.286
BLOCK: 1 10
-399.646 -16.8268
16.8267 -399.646
-549.453 -102.204

BLOCK: 0 11
-399.91 -8.473
8.47297 -399.91
294.945 -321.408
BLOCK: 2 11
-399.655 -16.6127
16.6127 -399.655
267.979 479.409
BLOCK: 0 12
-399.91 -8.47302
8.47304 -399.91
700.608 -208.982
BLOCK: 1 12
-399.646 -16.8268
16.8268 -399.646
690.657 186.1
BLOCK: 2 12
-399.655 -16.6127
16.6127 -399.655
673.642 591.834
BLOCK: 0 13
-399.91 -8.473
8.473 -399.91
-281.472 -736.139
BLOCK: 1 13
-399.646 -16.8268
16.8268 -399.646
-291.423 -341.057
BLOCK: 2 13
-399.655 -16.6126
16.6127 -399.655
-308.439 64.6771
BLOCK: 0 14
-399.91 -8.47302
8.47298 -399.91
568.225 -756.868
BLOCK: 1 14
-399.646 -16.8268
16.8268 -399.646

558.274 -361.785
BLOCK: 1 15
-399.646 -16.8267
16.8267 -399.646
138.568 860.372
BLOCK: 1 16
-399.646 -16.8268
16.8268 -399.646
-676.41 172.259
BLOCK: 1 17
-399.646 -16.8268
16.8268 -399.646
-160.258 149.604
BLOCK: 2 17
-399.655 -16.6127
16.6127 -399.655
-177.273 555.338
BLOCK: 1 18
-399.646 -16.8268
16.8267 -399.646
-858.045 -307.784
BLOCK: 2 18
-399.655 -16.6127
16.6126 -399.655
-875.061 97.95
BLOCK: 1 19
-399.646 -16.8267
16.8268 -399.646
-156.089 -592.707
BLOCK: 2 19
-399.655 -16.6126
16.6127 -399.655
-173.104 -186.973
BLOCK: 1 20
-399.646 -16.8268
16.8268 -399.646
713.711 -412.832
BLOCK: 2 20
-399.655 -16.6127

16.6127 -399.655
696.696 -7.09846
BLOCK: 1 21
-399.646 -16.8267
16.8268 -399.646
-517.113 -677.842
BLOCK: 1 22
-399.646 -16.8267
16.8268 -399.646
539.247 -668.975
BLOCK: 2 22
-399.655 -16.6127
16.6126 -399.655
522.232 -263.241
BLOCK: 2 23
-399.655 -16.6126
16.6127 -399.655
-57.9455 -317.201
BLOCK: 2 24
-399.655 -16.6126
16.6127 -399.655
704.703 -462.504
BLOCK: 2 25
-399.655 -16.6126
16.6127 -399.655
-279.441 -747.594
BLOCK: 2 26
-399.655 -16.6127
16.6127 -399.655
-13.3921 -987.003

_Hschur:
RBI: 3 3 6 9
CBI: 3 3 6 9
BLOCK: 0 0
7.056e+12 -599.377 -372.554
-599.377 7.056e+12 9300.66
-372.554 9300.66 7.056e+12
BLOCK: 0 1

-424.95 608.389 1.00176e-07
391.094 -9972.78 1.04209e-07
388.083 -9855.62 -820.702
BLOCK: 1 1
7.056e+12 -361.667 -728.556
-361.667 7.056e+12 9786.72
-728.556 9786.72 7.056e+12
BLOCK: 0 2
-1.36026e-07 2.76982e-09 4.32249e-08
-2.76981e-09 -1.36026e-07 2.10258e-07
4.53481e-08 -6.31985e-08 -9.03968e-08
BLOCK: 1 2
-400.132 -38.4342 3.13858e-08
-32.8619 -9999.87 1.9391e-07
-33.4301 -10152.1 -820.702
BLOCK: 2 2
7.056e+12 33.076 -1088.67
33.076 7.056e+12 -1361.9
-1088.67 -1361.9 7.056e+12