Incomplete Cholesky Preconditioner

Andrew Selle, Ron Fedkiw

July 10, 2004

1 Preconditioned Conjugate Gradient

For more detail on PCG see *Matrix Computations*, Golub and Van Loan pg. 534 (section 10.3.1). We wish to solve $A\vec{x} = \vec{b}$. Instead we'll solve $\tilde{A}\tilde{\vec{x}} = \tilde{\vec{b}}$ which we call our preconditioned system. The idea is to select a C that looks like A. Then we could solve $C^{-1}A\vec{x} = C^{-1}\vec{b}$. Obviously, if $C = A$ then $A^{-1}A\vec{x} = I\vec{x} = \vec{x} = \vec{b}$. Instead, so we can simplify the actual algorithm, we choose $\tilde{A} = C^{-1}AC^{-1}$ and $\tilde{\vec{x}} = C\vec{x}$ and $\tilde{\vec{b}} = C^{-1}b$. We want $\tilde{A}$ to be symmetric positive definite like A. Golub defines $M = C^2 = A$ as a "preconditioner" matrix. Note that $C =$"$\sqrt{A}$" exists because A is symmetric positive definite.

The algorithm assumes that $\vec{b}$ is zero in the null space, which makes sense because b is in the range of Ax. PCG can do this by simply reprojecting with `Enforce_Compatibility`, but this is not the best way. `PROJECTION` can be smarter about this in the case of fluids, for example handling Neumann subregions by adjusting only elements on the boundaries. Note that $\vec{x}$ may have a null space component, since PCG never uses $\vec{x}$ directly. It is only used to compute $\vec{r} = \vec{b} - A\vec{x}$ where A kills that part of $\vec{x}$. In fact the only place where the null space component can return is from the solve of $M\vec{z} = \vec{r}$ and that is why we explicitly reproject it after that solve.

$k = 1$
$\vec{r} = \vec{b} - A\vec{x}$
while $\|\vec{r}\|_\infty > \varepsilon$
  solve $M\vec{z} = \vec{r}$ — M is incomplete cholesky factor $LL^T\vec{z} = \vec{r}$
  Project out null space component of z
  if $k = 1$ **then** $\beta = 0$ **else** $\beta = \vec{r}_{k-1} \cdot \vec{z}_{k-1} / \vec{r}_{k-2} \cdot \vec{z}_{k-2}$
  $\vec{p}_k = \vec{z}_{k-1} + \beta_k \vec{p}_{k-1}$
  $\alpha_k = \vec{r}_{k-1}^T \vec{z}_{k-1} / \vec{p}_k^T A\vec{p}_k$
  $\vec{x}_k = \vec{x}_{k-1} + \alpha_k \vec{p}_k$
  $\vec{r}_k = \vec{r}_{k-1} - \alpha_k A\vec{p}_k$
  $k = k + 1$

2 Incomplete Cholesky Factorization

Besides "looking like" A, the "preconditioner" M should also be easy to form and invert. Cholesky factorization produces $LL^T = LU = M$ which can be solved by two cycles of back substitution. The algorithm as in Golub and Van Loan pg. 145 define outer product version:

for $k = 1 : n$
  $A(k,k) = \sqrt{A(k,k)}$
  $A(k+1:n,k) = A(k+1:n,k)/A(k,k)$
  for $j = k+1 : n$
    $A(j:n,j) = A(j:n,j) - A(j:n,k)A(j,k)$

Incomplete cholesky only writes to an element i,j if $A(i,j)$ is non-zero. Thus the incomplete Cholesky factorization is formed by:

for $k = 1 : n$
  $A(k,k) = \sqrt{A(k,k)}$
  for $i = k+1 : n$
    if $A(i,k) \neq 0$ **then** $A(i,k) = A(i,k)/A(k,k)$
  for $j = k+1 : n$
    for $i = j : n$
      if $A(i,j) \neq 0$ **then** $A(i,j) = A(i,j) - A(k,i)A(j,k)$

After the kth iteration of the matrix we get a $L_0 L_0^T$ block like

$$M = L_0 L_0^T = \left(\begin{array}{c|c} \sqrt{a(k,k)} & 0 \\ \hline \frac{a_{k+1,k}}{\sqrt{a_{k,k}}} & \\ \frac{a_{k+2,k}}{\sqrt{a_{k,k}}} & A_{k+1,k+1} - \frac{col_k col_k^T}{a_{k,k}} \\ \vdots & \\ \frac{a_{n,k}}{\sqrt{a_{k,k}}} & \end{array}\right) \left(\begin{array}{c|ccc} \sqrt{a(k,k)} & \frac{a_{k,k+1}}{\sqrt{a_{k,k}}} & \frac{a_{k,k+2}}{\sqrt{a_{k,k}}} & \cdots \quad \frac{a_{k,n}}{\sqrt{a_{k,k}}} \\ \hline 0 & & A_{k+1,k+1} - \frac{col_k col_k^T}{a_{k,k}} & \end{array}\right)$$

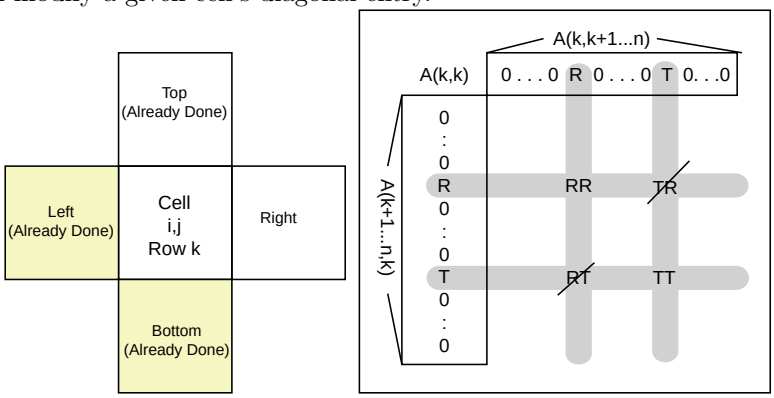
But we would rather not compute the square root because it is expensive. So instead of $L_0 L_0^T$ for our preconditioner we will refactor it to be LU where $L = \frac{1}{\sqrt{a_{k,k}}} L_0$ and $U = \sqrt{a_{k,k}} L_0$ so we get:

$$M = LU = \left(\begin{array}{c|c} 1 & 0 \\ \hline \frac{a_{k+1,k}}{a_{k,k}} & \\ \frac{a_{k+2,k}}{a_{k,k}} & A_{k+1,k+1} - \frac{col_k col_k^T}{a_{k,k}} \\ \vdots & \\ \frac{a_{n,k}}{a_{k,k}} & \end{array}\right) \left(\begin{array}{c|ccc} a(k,k) & a_{k,k+1} & a_{k,k+2} & \cdots \quad a_{k,n} \\ \hline 0 & & A_{k+1,k+1} - \frac{col_k col_k^T}{a_{k,k}} & \end{array}\right)$$

2.1 Uniform Grid

For the case of the uniform grid this new factorization can be made fast. In particular, we know that we will not update for any part of the matrix that is

not in the block $A(k:n, k:n)$. In particular, the coupling associated with the bottom and with the left are not in the block. That implies that the column and row used in the outer product will consist of only two nonzero entries, the right and the top. Thus, the only place where the outer product can be nonzero is at these positions (the crossings in the figure). But, the crossings on the off-diagonals are zero entries in the original matrix because right and top are decoupled so they will not be filled. This is not true in the general case (like for an octree grid), but in the uniform grid case it allows us to update only the diagonals. Alternatively, we can think of the reverse and consider which elements can modify a given diagonal. In that case only the left and bottom will modify a given cell's diagonal entry.



We can represent both L and U in one $n \times n$ matrix because L's diagonal is all ones. For efficiency we actually represent the diagonal of the Cholesky factorization as the reciprocal. Note C_{r}, C_{c}, C_{t} are the right, center, and top values for a given matrix row for the cell i, j. The code to form the LU matrices for 2d becomes:

$C = A$
for $k = 1 : n$
  $i, j \in \mathbb{Z}^+$ s.t. $k = i + jN$
  $1/[C_{\mathrm{c}}(i,j) = A(k,k) - C_{\mathrm{r}}(i-1,j)^2 C_{\mathrm{c}}(i-1,j) - C_{\mathrm{t}}(i,j-1)^2 C_{\mathrm{c}}(i,j-1)]$

When doing the backsubstitution solves during the $Mz = b$ step of PCG, we see that having the reciprocal diagonal allows us to multiply instead of divide by the diagonal element during a row solve.

2.2 Sparse Matrix

In a general sparse matrix we cannot make the simplifying assumption of only modifying the diagonal. Moreover the representation of the sparsity biases certain accesses to be computationally cheaper. Our implementation keeps each row of the matrix in a sparse vector which keeps an array of indices and values. Thus, traversing the columns of a single row is very cheap. Traversing down a column is expensive. Thus, we reorder the Cholesky formation process to only go through each row once.

2.3 Modified Cholesky

Incomplete Cholesky is known to perform poorly as the grid is refined. To solve this problem, the modified Cholesky scheme is employed, where the elements of the outer product that are not filled in (due to the sparsity pattern of A) are added to diagonal of that row instead - usually diminished by a factor of about .95.

for $k = 1 : n$
  $A(k,k) = \sqrt{A(k,k)}$
  for $i = k+1 : n$
    if $A(i,k) \neq 0$ **then** $A(i,k) = A(i,k)/A(k,k)$
  for $j = k+1 : n$
    for $i = j : n$
      if $A(i,j) \neq 0$ **then** $A(i,j) = A(i,j) - A(k,i)A(j,k)$
      else $A(i,i) = A(i,i) - A(k,i)A(j,k)$

More detail can be found in Chan and Van Der Vorst *Approximate and Incomplete Factorizations.*