Overview of Templates

Jack Dongarra

Computer Science Department
University of Tennessee

and

Mathematical Sciences Section
Oak Ridge National Laboratory

(http://www.netlib.org/utk/people/JackDongarra.html)

Who needs numerical software?

- **Kinds of users and their priorities:**
  - **Engineers and scientists generally, who make 12,000 requests/day to netlib and keep NAG and IMSL in business:**
    1. **Easy interface, hidden details**
    2. **Reliability**
    3. **Speed**
  - **HPCC community, solving biggest and hardest problems:**
    1. **Speed**
    2. **Access to details for tuning**
    3. **Reliable enough for their application**
  - **Teachers and Students**
    1. **Ease of understanding**
    2. **Access to some details for learning**
- **Can we please everyone with one "medium"?**

WHY ITERATIVE METHODS?

When direct methods too expensive

When multigrid not feasable

— iterative methods may do —

ADVANTAGE OF ITERATIVE METHODS:

modest in time per step

modest in memory requirements

DISADVANTAGE: not robust

ILLUSTRATION

Finite element space model (Simon '89)

Realistic model: $n = 5 \star 10^9$

Direct Solver at 1 TFLOP: $520,040$ years

(exploitable parallelism)

Conjugate Gradients: $\sim$ 500 iteration steps

With 1 TFLOP speed: 30 minutes

(parallelism exploitable?)

ITERATIVE METHODS - BACKGROUND

$Ax = b$, write: $A = I - (I - A)$

$$(I - (I - A))\,x = b$$

⇓

BASIC ITERATION METHOD:

$$\begin{aligned} x_i &= b + (I - A)x_{i-1} \\ &= x_{i-1} + r_{i-1} \\ &= x_{i-2} + r_{i-2} + r_{i-1} \end{aligned}$$

BASIC ITERATION METHOD

$$x_i = x_0 + r_0 + r_1 + \cdots + r_{i-1}$$

from $x_i = x_{i-1} + r_{i-1}$

$A\times \Rightarrow Ax_i = Ax_{i-1} + Ar_{i-1}$

$$\begin{aligned} b - \quad \Rightarrow \quad \underbrace{b - Ax_i}_{r_i} &= \underbrace{b - Ax_{i-1}}_{r_{i-1}} - Ar_{i-1} \\ &= r_{i-1} - Ar_{i-1} \\ &= (I - A)r_{i-1} \end{aligned}$$

$$\begin{aligned} x_i &= x_0 + r_0 + (I - A)r_0 + \cdots + (I - A)^{i-1}r_0 \\ &= x_0 + \left[r_0, Ar_0, A^2r_0, \ldots, A^{i-1}r_0\right] \end{aligned}$$

Iterative Methods

- No single iterative method that can solve any given sparse linear system in reasonable time and with reasonable memory requirements.
- Rate of convergence depends on certain properties of the system, and these properties are often unknown in practice.
- For many very large linear systems, iterative methods are at this moment the only choice.
- Necessary to have a variety of iterative methods available.
- Through Templates the user is guided into the world of iterative methods, with only a minimum of linear algebra knowledge required.
- Using Templates the user can build software at several levels of sophistication, depending on requirements and goals.

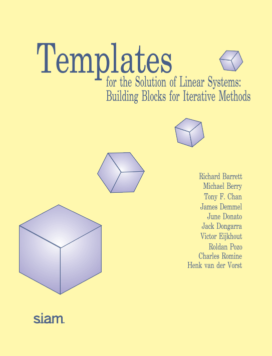
Iterative Methods

- The iteration schemes consist of one or two matrix vector products, a couple of inner-products and a few vector updates.
- The algorithms also allow for some form of preconditioning, which means that in each step a linear system $Mz = r$ has to be solved, where M has to be specified by the user.
- These operations are pretty well understood, and it is easy to see how the algorithms work.
- These basic iteration Templates may be used for simple computational experiments in order to get familiarized with the basic iteration methods and their mutual differences.
- The user is given extensive advice in the manual on how to expand the basic codes to reliable software for practical problems.

Motivation from Iterative Methods

Algorithm sketches for

- Gauss Siedel, Jacobi SOR, SSOR, etc.
- PCG, Bi-CG, QMR, GMRES, etc.
- Preconditioners
- Domain Decomposition
- Parallelism



Templates
for the Solution of Linear Systems:
Building Blocks for Iterative Methods

Richard Barrett
Michael Berry
Tony F. Chan
James Demmel
June Donato
Jack Dongarra
Victor Eijkhout
Roldan Pozo
Charles Romine
Henk van der Vorst

siam

Templates for Sparse Matrix Computations

- **Alternate to software**
- **Describe the basic features of the algorithms**
- **Language not so important**
- **Communication between disciplines**
- **Clear documentation**
- **Test cases**
- **Provide understanding of the results**
- **Allow customization**
- **Serve as a pedagogical role**
- **Retain the delicate numerical details**
- **Describe parallel opportunities**

A Template for an Algorithm Includes

1. A high-level description of the algorithm.
2. A description of when it is effective, including conditions on the input, and estimates of the time, space or other resources required. If there are natural competitors, they will be referenced.
3. A description of available refinements and user-tunable parameters, as well as advice on when to use them.
4. Pointers to complete or partial implementations, perhaps in several languages or for several architectures (each parallel architecture). These implementation expose those details suitable for user-tuning, and hide the others.
5. Numerical examples, on a common set of examples, illustrating both easy cases and difficult cases.
6. Trouble shooting advice.
7. Pointers to texts or journal articles for further information.

In addition to individual templates, there will be a *decision tree* to help steer the user to the right algorithm, or subset of possible algorithms, based on a sequence of questions about the nature of the problem to be solved.

Building Blocks
for Iterative Solution of Linear Systems

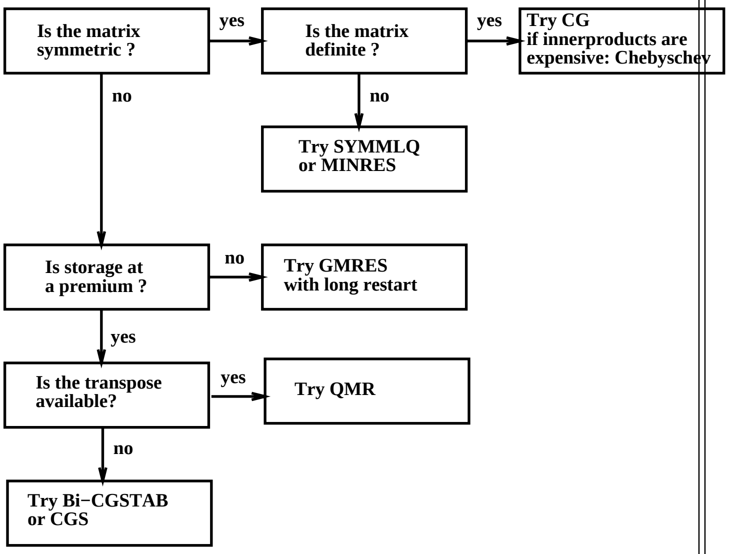
- **Algorithm sketches for**
  - **Algorithm**
    - **Jacobi, Gauss-Seidel, SOR**
    - **CG, Precond-CG, CG-Squared, Bi-CG**
    - **Bi-CG-Stab, QMR, GMRES, Chebychev**
  - **Matlab script**
  - **Fortran code in a 2-D array**
  - **Fortran code using reverse commmunication**
  - **Preconditioners**
    - **Jacobi, SSOR, Incomplete Factoriation, Polynominal**
  - **Convergence tests**
  - **Performance Summary**
  - **Sparse Data Structures**
  - **Hints on parallel implementation**
  - **Available from SIAM or http://www.netlib.org/templates/index.html**

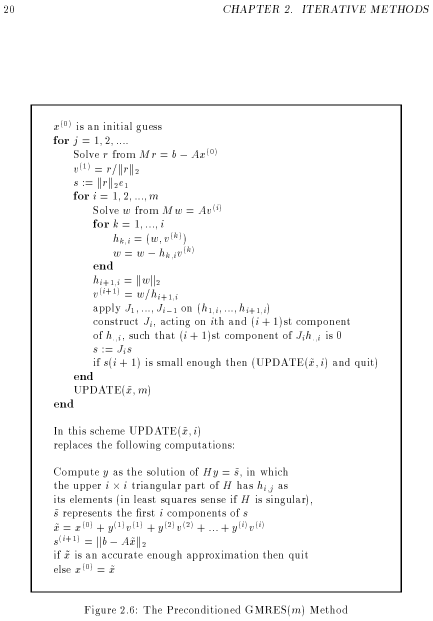
Three Potential User Communities for such Tools

- **The "HPCC" (High Performance Computing and Communication) community consists of those scientists trying to solve the largest, hardest applications in their fields. They desire high speed, access to algorithmic details for performance tuning, and reliability for their problem.**
- **Engineers and scientists generally desire easy-to-use, reliable software, that is also reasonably efficient.**
- **Students and teachers want simple but generally effective algorithms, which are easy to explain and understand.**

- **Matrix data type:**
  - **General Dense.**
  - **Banded Dense.**
  - $\boxed{\textit{General Sparse.}}$

    Preferred Operations (in order of decreasing power):
    - ○ $\boxed{\textit{Solve } (A - \sigma I)x = b}$
    - ○ **Solve** $Ax = b$
    - ○ **Multiply** $x = A \cdot b$
    - ○ **Don't know.**
  - **Sparse with Special Structure.**
  - **Dense but compactly representable.**
  - **Other.**
- **Machine type:**
  - **Serial.**
  - **Parallel.**
    - ∗ **Shared Memory.**
    - ∗ **Distributed Memory.**


Is the matrix symmetric ?
yes
Is the matrix definite ?
yes
Try CG if innerproducts are expensive: Chebyschev
no
Try SYMMLQ or MINRES
no
Is storage at a premium ?
no
Try GMRES with long restart
yes
Is the transpose available?
yes
Try QMR
no
Try Bi−CGSTAB or CGS

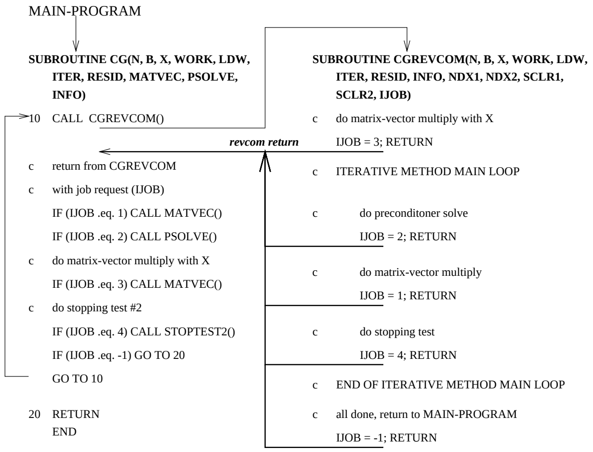

20 CHAPTER 2. ITERATIVE METHODS

```
$x^{(0)}$ is an initial guess
for $j = 1, 2, ....$
    Solve $r$ from $Mr = b - Ax^{(0)}$
    $v^{(1)} = r/\|r\|_2$
    $s := \|r\|_2 e_1$
    for $i = 1, 2, ..., m$
        Solve $w$ from $Mw = Av^{(i)}$
        for $k = 1, ..., i$
            $h_{k,i} = (w, v^{(k)})$
            $w = w - h_{k,i}v^{(k)}$
        end
        $h_{i+1,i} = \|w\|_2$
        $v^{(i+1)} = w/h_{i+1,i}$
        apply $J_1, ..., J_{i-1}$ on $(h_{1,i}, ..., h_{i+1,i})$
        construct $J_i$, acting on $i$th and $(i+1)$st component
        of $h_{.,i}$, such that $(i+1)$st component of $J_i h_{.,i}$ is 0
        $s := J_i s$
        if $s(i+1)$ is small enough then (UPDATE($\tilde{x}, i$) and quit)
    end
    UPDATE($\tilde{x}, m$)
end

In this scheme UPDATE($\tilde{x}, i$)
replaces the following computations:

Compute $y$ as the solution of $Hy = \tilde{s}$, in which
the upper $i \times i$ triangular part of $H$ has $h_{i,j}$ as
its elements (in least squares sense if $H$ is singular),
$\tilde{s}$ represents the first $i$ components of $s$
$\tilde{x} = x^{(0)} + y^{(1)}v^{(1)} + y^{(2)}v^{(2)} + ... + y^{(i)}v^{(i)}$
$s^{(i+1)} = \|b - A\tilde{x}\|_2$
if $\tilde{x}$ is an accurate enough approximation then quit
else $x^{(0)} = \tilde{x}$
```

Figure 2.6: The Preconditioned GMRES(m) Method



MAIN-PROGRAM

```
SUBROUTINE CG(N, B, X, WORK, LDW,
     ITER, RESID, MATVEC, PSOLVE,
     INFO)
10   CALL  CGREVCOM()

c    return from CGREVCOM
c    with job request (IJOB)
     IF (IJOB .eq. 1) CALL MATVEC()
     IF (IJOB .eq. 2) CALL PSOLVE()
c    do matrix-vector multiply with X
     IF (IJOB .eq. 3) CALL MATVEC()
c    do stopping test #2
     IF (IJOB .eq. 4) CALL STOPTEST2()
     IF (IJOB .eq. -1) GO TO 20
     GO TO 10

20   RETURN
     END
```

revcom return

```
SUBROUTINE CGREVCOM(N, B, X, WORK, LDW,
     ITER, RESID, INFO, NDX1, NDX2, SCLR1,
     SCLR2, IJOB)
c    do matrix-vector multiply with X
     IJOB = 3; RETURN
c    ITERATIVE METHOD MAIN LOOP

c         do preconditoner solve
          IJOB = 2; RETURN

c         do matrix-vector multiply
          IJOB = 1; RETURN

c         do stopping test
          IJOB = 4; RETURN
c    END OF ITERATIVE METHOD MAIN LOOP

c    all done, return to MAIN-PROGRAM
     IJOB = -1; RETURN

     STOP
     END
```

Note 1: Diagram only represents reverse communication interactions.

Note 2: CGREVCOM() logic to resume execution at appropriate place not shown.

```
*     First time call always init.
      IJOB = 1

 1    CONTINUE

          CALL CGSREVCOM(N, B, X, WORK, LDW, ITER, RESID, INFO,
     $                   NDX1, NDX2, SCLR1, SCLR2, IJOB)

*         On a return from REVCOM() we use the table
*         to figure out what is reqd.
          IF (IJOB .eq. -1) THEN
*           revcom wants to terminate, so do it.
            GOTO 2
          ELSEIF (IJOB .eq. 1) THEN
*           call matvec.
            CALL MATVEC(SCLR1, WORK(NDX1), SCLR2, WORK(NDX2))
          ELSEIF (IJOB .eq. 2) THEN
*           call solve.
            CALL PSOLVE(WORK(NDX1), WORK(NDX2))
          ELSEIF (IJOB .eq. 3) THEN
*           call matvec with X.
            CALL MATVEC(SCLR1, X, SCLR2, WORK(NDX2))
         ELSEIF (IJOB .EQ. 4) THEN
*           do stopping test 2
*           if FirstTime, set info  so that BNRM2 is computed.
            IF( FTFLG ) INFO = -1
            CALL STOPTEST2(N, WORK(NDX1), B, BNRM2, RESID, TOL, INFO)
            FTFLG = .FALSE.
         ENDIF
*        done what revcom asked, set IJOB & go back to it.
         IJOB = 2
         GOTO 1
*     come here to terminate
 2    CONTINUE
*
      RETURN
*     End of CGS
      END
```

```
function [x, error, iter, flag] = cg(A, x, b, M, max_it, tol)
%  -- Iterative template routine --
%     Details of this algorithm are described in "Templates for the
%     Solution of Linear Systems: Building Blocks for Iterative
%     Methods", Barrett, et al, SIAM Publications, 1993.
%     (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
%  [x, error, iter, flag] = cg(A, x, b, M, max_it, tol)
% cg.m solves the symmetric positive definite linear system Ax=b
% using the Conjugate Gradient method with preconditioning.
% input   A        REAL symmetric positive definite matrix
%         x        REAL initial guess vector
%         b        REAL right hand side vector
%         M        REAL preconditioner matrix
%         max_it   INTEGER maximum number of iterations
%         tol      REAL error tolerance
% output  x        REAL solution vector
%         error    REAL error norm
%         iter     INTEGER number of iterations performed
%         flag     INTEGER: 0 = solution found to tolerance
%                           1 = no convergence given max_it
  flag = 0;  iter = 0;                       % initialization
  bnrm2 = norm( b );
  if  ( bnrm2 == 0.0 ), bnrm2 = 1.0; end
  r = b - A*x;
  error = norm( r ) / bnrm2;
  if ( error < tol ) return, end
  for iter = 1:max_it                        % begin iteration
     z  = M \ r;
     rho = (r'*z);
     if ( iter > 1 ),                        % direction vector
        beta = rho / rho_1;
        p = z + beta*p;
     else
        p = z;
     end
     q = A*p;
     alpha = rho / (p'*q );
     x = x + alpha * p;                     % update approximation vector
     r = r - alpha*q;                       % compute residual
     error = norm( r ) / bnrm2;             % check convergence
     if ( error <= tol ), break, end
     rho_1 = rho;
  end
  if ( error > tol ) flag = 1; end          % no convergence
% END cg.m
```

Yes, we can please everyone!

- **We propose that**
  - **NHSE - National High Performance Software Exchange**
  - **GAMS - Guide to Available Math Software**
  - **Netlib**
  - **Templates**
  - **NA textbooks**
  - **Annotated bibliographies**
  - **Computing environments á la Matlab or Maple**
  - **On-line compute servers (comp-bots) for bigger problems**

should all be part of one seamless whole.

Starting vector: ...

Operate: ...

Vector operations: ...

Reorthogonalization: Were the arithmetic exact, this had not been needed, but in finite precision orthogonality is lost as soon as one eigenvalue converges. The choices are:

1. *Full:* Simple to implement see ??? below. Iterations will need successively more work as j grows. Preferred choice in shifted and inverted case when several eigenvalues converge in few steps j.
2. *Selective:* The most elaborate scheme, necessary when convergence is slow and several eigenvalues are sought. See below ??? !
3. *Local:* Used for huge matrices, when it is difficult to store the whole basis V_j. Advisable only when one or two extreme eigenvalues are sought. We make sure that v_{j+1} is orthogonal to v_{j-1} and v_j by subtracting $r = r - v_{j-1}(v_{j-1}^* r); r = r - v_j(v_j^* r)$ once in this step.

Test for convergence: ...

Approximate eigenvalues and eigenvectors: ...

Algorithm: *Symmetric single vector Lanczos*

```
1. Start with $r$ starting vector. Compute $b_0 = \|r\|_2$
2. for $j = 1, 2, \ldots,$ until Convergence,
    1. Normalize $v_j = r/b_{j-1}$
    2. Operate $r = Av_j$
    3. Subtract $r = r - v_{j-1}b_{j-1}$
    4. Compute $a_j = v_j^* r$
    5. Subtract $r = r - v_j a_j$
    6. Reorthogonalize if necessary
    7. Compute norm $b_j = \|r\|_2$
    8. Compute approximate eigenvalues $\theta_i$
    9. Test for convergence
3. Compute approximate eigenvectors
```

End.

Symmetric Lanczos template

Theory: The symmetric Lanczos algorithm is applicable to the eigenvalue problem,

$$Ax = \lambda x \ ,$$

where A is a symmetric matrix operator. It starts with a vector x and builds up an orthogonal basis V_j of the Krylov subspace,

$$K^j(A,x) = \operatorname{span}\{x, Ax, A^2x, \ldots, A^{j-1}x\} \ ,$$

one column at a time. In each step just one matrix vector multiplication

$$y = Ax$$

is needed. In the basis V, the operator A is represented by a tridiagonal matrix,

$$T_{j,j} = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{j-1} \\ & & b_{j-1} & a_j \end{bmatrix} ,$$

which is also built up one column at a time, using the basic recursion,

$$AV_j = V_{j+1}T_{j+1,j} \ .$$

- **Matrix data type:**
    - **General Dense.**
    - **Banded Dense.**
    - *General Sparse.*

      Preferred Operations (in order of decreasing power):
        - *Solve* $(A - \sigma I)x = b$
        - **Solve** $Ax = b$
        - **Multiply** $x = A \cdot b$
        - **Don't know.**
    - **Sparse with Special Structure.**
    - **Dense but compactly representable.**
    - **Other.**

- **Machine type:**
    - **Serial.**
    - **Parallel.**
        - **Shared Memory.**
        - **Distributed Memory.**

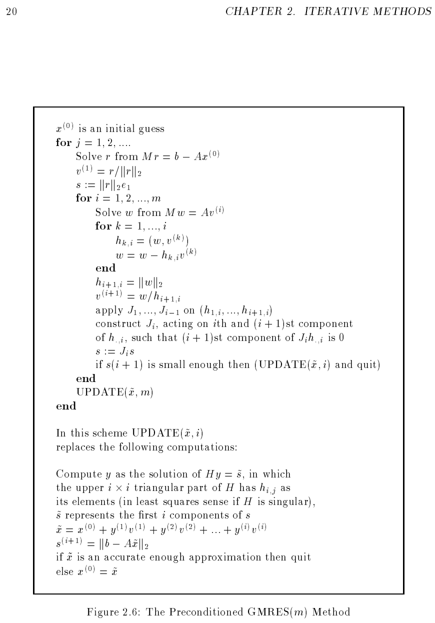
Symmetric Subtree

- **Desired Eigenvalue Information:**
  - **All eigenvalues.**
  - $\boxed{\textit{Some eigenvalues}}$**:**
    - ∗ **Compute λ_i through λ_j where $\lambda_1 \le \cdots \le \lambda_n$**
    - ∗ $\boxed{\textit{Eigenvalues in } [\alpha, \beta].}$
    - ∗ **Other.**

- **Desired Eigenvector Information Desired?:**
  - $\boxed{\textit{Yes.}}$
  - **No**

Top of Eigentemplates Decision Tree

- **Problem type:**
  - $\boxed{A - \lambda I,\ A = A^*}$
    - ∗ $A = H^*H$, H **given (point to SVD).**
    - ∗ $\boxed{\textit{Other (}\text{Symmetric Subtree}\textit{).}}$
  - **SVD of** A
  - $A - \lambda I$, **general** A
  - $A - \lambda B$, $A = A^*, B = B^*$ **and positive definite**
  - $A - \lambda B$, A **and** B **square**
  - **Other**
  - **Don't know**



20 *CHAPTER 2. ITERATIVE METHODS*

```
$x^{(0)}$ is an initial guess
for $j = 1, 2, ....$
    Solve $r$ from $Mr = b - Ax^{(0)}$
    $v^{(1)} = r/\|r\|_2$
    $s := \|r\|_2 e_1$
    for $i = 1, 2, ..., m$
        Solve $w$ from $Mw = Av^{(i)}$
        for $k = 1, ..., i$
            $h_{k,i} = (w, v^{(k)})$
            $w = w - h_{k,i} v^{(k)}$
        end
        $h_{i+1,i} = \|w\|_2$
        $v^{(i+1)} = w/h_{i+1,i}$
        apply $J_1, ..., J_{i-1}$ on $(h_{1,i}, ..., h_{i+1,i})$
        construct $J_i$, acting on $i$th and $(i+1)$st component
        of $h_{.,i}$, such that $(i+1)$st component of $J_i h_{.,i}$ is 0
        $s := J_i s$
        if $s(i+1)$ is small enough then (UPDATE($\tilde{x}, i$) and quit)
    end
    UPDATE($\tilde{x}, m$)
end

In this scheme UPDATE($\tilde{x}, i$)
replaces the following computations:

Compute $y$ as the solution of $Hy = \tilde{s}$, in which
the upper $i \times i$ triangular part of $H$ has $h_{i,j}$ as
its elements (in least squares sense if $H$ is singular),
$\tilde{s}$ represents the first $i$ components of $s$
$\tilde{x} = x^{(0)} + y^{(1)} v^{(1)} + y^{(2)} v^{(2)} + ... + y^{(i)} v^{(i)}$
$s^{(i+1)} = \|b - A\tilde{x}\|_2$
if $\tilde{x}$ is an accurate enough approximation then quit
else $x^{(0)} = \tilde{x}$
```

Figure 2.6: The Preconditioned GMRES(m) Method

A Practical Solution: Meta-Libraries

- **separate the data implementation details out of the fundamental algorithms**

- **describe library algorithms in terms of abstract base classes (inheritance)**

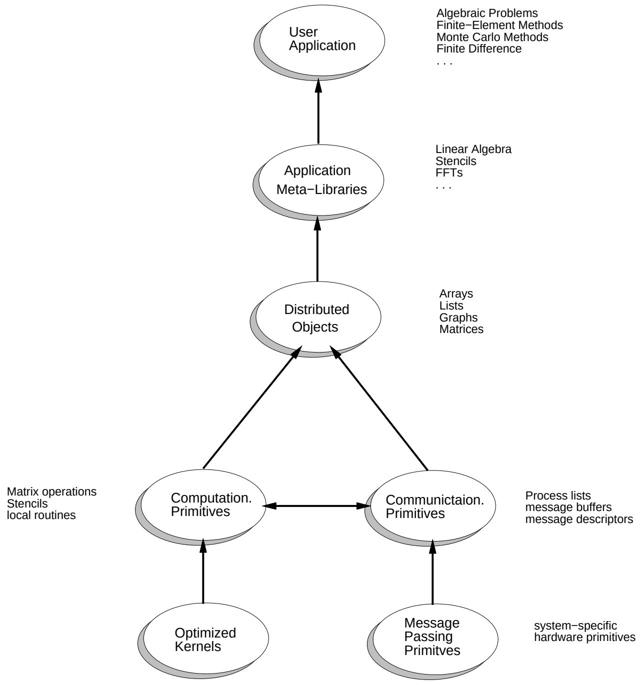
- **specify only the interface**

- **utilize late-binding (polymorphism) to determine data distributions and implementations**

Need good design to choose proper objects and abstractions to balance

- **performance**
- **reusability**
- **generality**
- **control**
- **flexibility**

Software Hierachy



Templates for Sparse Matrix Computations

- Alternate to software
- Describe the basic features of the algorithms
- Language not so important
- Communication between disciplines
- Clear documentation
- Test cases
- Provide understanding of the results
- Allow customization
- Serve as a pedagogical role
- Retain the delicate numerical details
- Describe parallel opportunities

Building Blocks for Iterative Solution of Linear Systems

- Some users want algorithms which are fast for their particular application even if not reliable as general methods.
- They want the tools to customize iterative methods for solving the linear systems that arise in their specific problem.
- Offer reusable software templates
  - Algorithm
    - Jacobi, Gauss-Seidel, SOR
    - CG, Precond-CG, CG-Squared, Bi-CG
    - Bi-CG-Stab, QMR, GMRES, Chebychev
  - Matlab script
  - Fortran code in a 2-D array
  - Preconditioners
  - Convergence tests
  - Performance Summary
  - Data Structures
  - Hints on parallel implementation

Preconditioned Conjugate Gradient Method for $Ax = b$

```
Initial $r^{(0)} = b - Ax^{(0)}$
for $i = 1, 2, \ldots$
    solve $Mz^{(i-1)} = r^{(i-1)}$
    $\rho_{i-1} = r^{(i-1)^T} z^{(i-1)}$
    if $i = 1$
        $p^{(1)} = z^{(0)}$
    else
        $\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
        $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)} = Ap^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)^T} q^{(i)}$
    $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$
    $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$
    check convergence;
end
```

```
r = b - A*x;
for (int i=1; i<maxiter; i++){
    z = M.solve(r);
    rho = r * z;
    if (i==1)
       p = z;
    else{
       beta = rho1/ rho0;
       p = z + p * beta;
    }
    q = A*p;
    alpha = rho1 / (p*q);
    x += alpha * p;
    r -= alpha * q;
    if (norm(r)/normb < tol) break;
}
```

OO Sparse Matrix Libraries

- **motivated by "Building Blocks for Iterative Solution of Linear Systems"**
- **based on Level 3 Sparse BLAS interface**
- **support distributed formats: compressed row / column**
- **implementing parallel Level 3 Sparse BLAS**
- **inital development of conventional sparse matrix classes**
  - **compressed column (Harwell/Boeing)**
  - **compressed row**
  - **jagged diagonal storage**
  - **compressed diagonal storage**
  - **block compressed row/col storage**
  - **skyline storage**

OO Sparse Matrix Libraries (contd.)

- **Iterative algorithms**
  - Stationary Methods (SOR),
  - Conjugate Gradient (CG),
  - Generalized Minimal Residual (GMRES),
  - Minimum Residual (MINRES),
  - Quasi-Minimal Residual (QMR),
  - Chebyshev Iteration (Cheb),
  - Conjugate Gradient Squared (CGS),
  - Biconjugate Gradient (BiCG),
  - Biconjugate Gradient Stabilized (Bi-CGSTAB)
- **Preconditioners are user-defined**
  - **some basic stationary preconditioners provided**
  - **application specific**
  - **more difficult to parallelize**
- **Current Development**
  - **Phase I: development of conventional sparse matrix classes:**
  - **Phase II: development of shared memory sparse matrix**
  - **Phase III: development of distributed memory sparse matrix objects with Level 3 Sparse BLAS**

A Simple Example

$$\frac{\|Ax-b\|_\infty}{N\|A\|_\infty\|x\|_\infty\epsilon} < 1$$

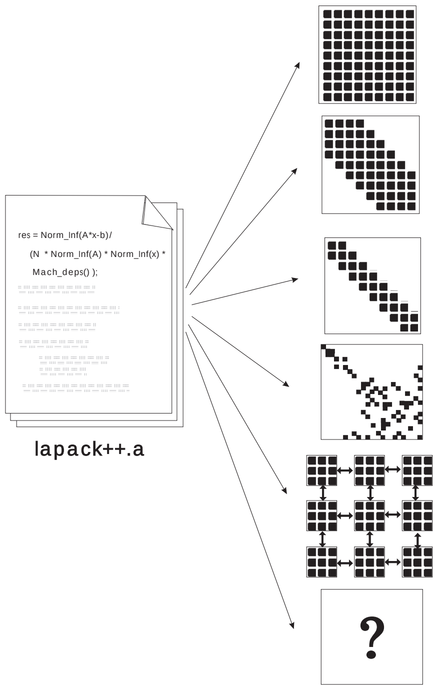
F77 LAPACK / BLAS

```
      DCOPY(N, AXMB, 1, B, 1)
      DGEMV('No Transpose', N, N, 1.0, A, N, X, 1, -1.0, AXMB, 1)
      DO 100 I=1,N
100   R(I) = DASUM(N, A(I), N)
      INDEX = IDAMAX(N, R, 1)
      RES = DASUM(N,AXMB,1) / ( N* R(INDEX) * DASUM(N,X)
     &         * DLMACH('e'))
```

LAPACK++

```
  res =          Norm_Inf(A*x-b) /
         (N * Norm_Inf(A) * Norm_Inf(x) * Mach_deps() );
```

One Single Lapack++ Library



Conventional LAPACK Storage Formats

Hermitian

Storage	Hermitian matrix A	Storage in array A
Upper	$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ \bar{a}_{01} & a_{11} & a_{12} & a_{13} \\ \bar{a}_{02} & \bar{a}_{12} & a_{22} & a_{23} \\ \bar{a}_{03} & \bar{a}_{13} & \bar{a}_{23} & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} \\ * & a_{11} & a_{12} & a_{13} \\ * & * & a_{22} & a_{23} \\ * & * & * & a_{33} \end{matrix}$
Lower	$\begin{pmatrix} a_{00} & \bar{a}_{10} & \bar{a}_{20} & \bar{a}_{30} \\ a_{10} & a_{11} & \bar{a}_{21} & \bar{a}_{31} \\ a_{20} & a_{21} & a_{22} & \bar{a}_{32} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & * & * & * \\ a_{10} & a_{11} & * & * \\ a_{20} & a_{21} & a_{22} & * \\ a_{30} & a_{31} & a_{32} & a_{33} \end{matrix}$

Triangular

Storage	Triangular matrix A	Storage in array A
Upper	$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ & a_{11} & a_{12} & a_{13} \\ & & a_{22} & a_{23} \\ & & & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & a_{01} & a_{02} & a_{03} \\ * & a_{11} & a_{12} & a_{13} \\ * & * & a_{22} & a_{23} \\ * & * & * & a_{33} \end{matrix}$
Lower	$\begin{pmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\begin{matrix} a_{00} & * & * & * \\ a_{10} & a_{11} & * & * \\ a_{20} & a_{21} & a_{22} & * \\ a_{30} & a_{31} & a_{32} & a_{33} \end{matrix}$

Banded

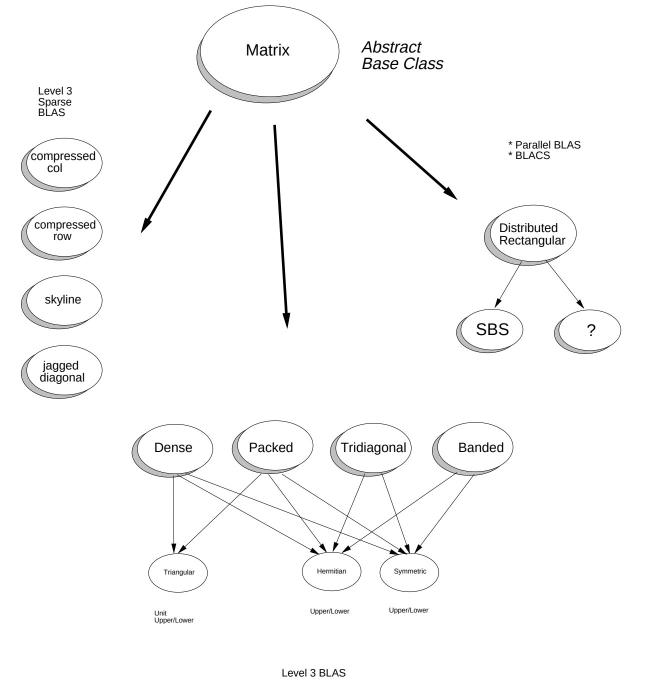
Band matrix A	Band storage in array AB
$\begin{pmatrix} a_{00} & a_{01} & & & \\ a_{10} & a_{11} & a_{12} & & \\ a_{20} & a_{21} & a_{22} & a_{23} & \\ & a_{31} & a_{32} & a_{33} & a_{34} \\ & & a_{42} & a_{43} & a_{44} \end{pmatrix}$	$\begin{matrix} * & a_{01} & a_{12} & a_{23} & a_{34} \\ a_{00} & a_{11} & a_{22} & a_{33} & a_{44} \\ a_{10} & a_{21} & a_{32} & a_{43} & * \\ a_{20} & a_{31} & a_{42} & * & * \end{matrix}$

Triangular Packed

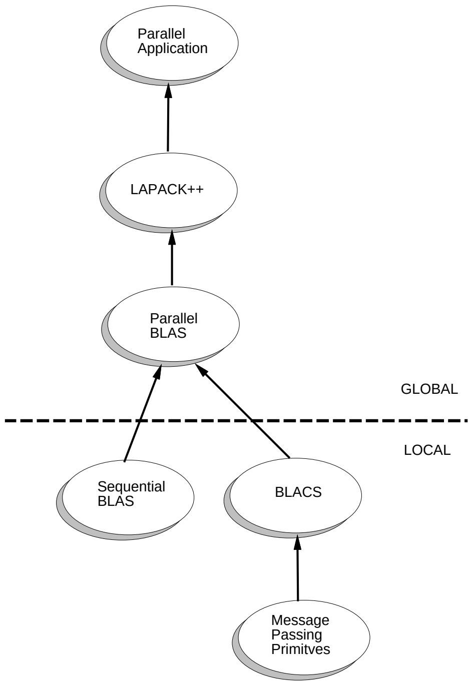
Storage	Triangular matrix A	Packed storage in array AP
Upper	$\begin{pmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ & a_{11} & a_{12} & a_{13} \\ & & a_{22} & a_{23} \\ & & & a_{33} \end{pmatrix}$	$a_{00}\ \underbrace{a_{01}\,a_{11}}\ \underbrace{a_{02}\,a_{12}\,a_{22}}\ \underbrace{a_{03}\,a_{13}\,a_{23}\,a_{33}}$
Lower	$\begin{pmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ a_{30} & a_{31} & a_{32} & a_{33} \end{pmatrix}$	$\underbrace{a_{00}\,a_{10}\,a_{20}\,a_{30}}\ \underbrace{a_{11}\,a_{21}\,a_{31}}\ \underbrace{a_{22}\,a_{32}}\ a_{33}$

Orthogonal $Q = \underbrace{H_1 H_2 \ldots H_q}_{\text{Householder vectors}}$

Linear Algebra Classes



Object Oriented Software Framework

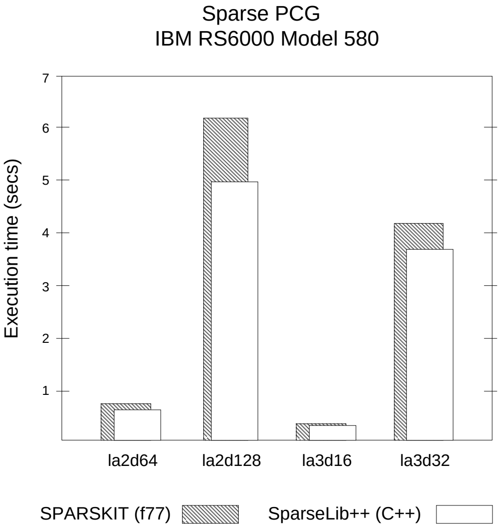


Iterative Methods Meta-Library (IML++)

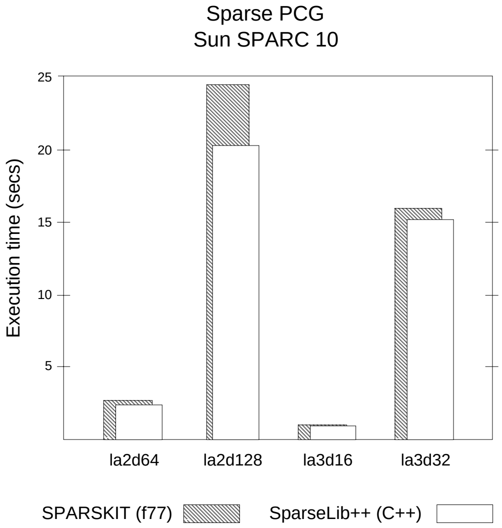
- **Stationary Methods (SOR)**
- **Conjugate Gradient (CG)**
- **Generalized Minimal Residual (GMRES)**
- **Minimum Residual (MINRES)**
- **Quasi-Minimal Residual (QMR)**
- **Chebyshev Iteration (Cheb)**
- **Conjugate Gradient Squared (CGS)**
- **Biconjugate Gradient (BiCG)**
- **Biconjugate Gradient Stabilized (Bi-CGSTAB)**

SparseLib++ Matrix Classes

- coordinate
- compressed column (Harwell/Boeing)
- compressed row
- jagged diagonal storage
- compressed diagonal storage
- block compressed row/col storage
- skyline storage

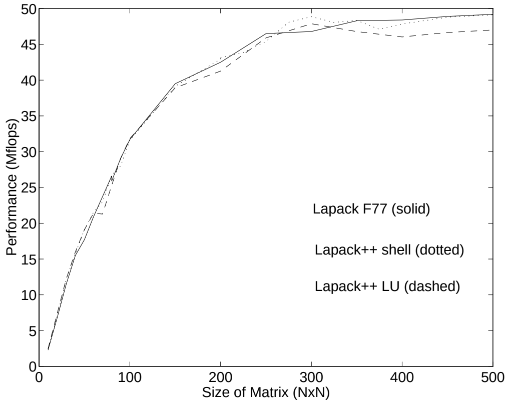


PCG using compressed row matrices; optimized Fortran (xlf -O) Sparskit and C++ (xlC -O) SparseLib++ (Sparse Blas).



PCG using compressed row matrices; optimized Fortran (f77 -O) Sparskit and C++ (g++ -O) SparseLib++ (Sparse Blas).

What does the C++ design cost?



Performance of LAPACK++ LU factorization on the IBM RS/6000 Model 550 workstation, using GNU g++ v. 2.3.1 and BLAS routines from the IBM ESSL library.

Level 3 Sparse Blas
Large (Duff, Marrone, Radicatti)
(S. Carney, M. Heroux, G. Li, and K. Wu)

- **sparse matrix products,**

$$C \leftarrow \alpha \; op(A) \; B + \beta C$$

- **solution of triangular systems,**

$$C \leftarrow \alpha D \; op(A)^{-1} \; B + \beta C$$

- **reordering of a sparse matrix (permutations),**

$$A \leftarrow A \; op(P)$$

- **conversion of one data format to another,**

$$A' \leftarrow A$$

α and β are scalars
B and C are dense (multiple vectors)
where D is a (block) diagonal matrix,
A and A' are sparse
$op(A)$ is either A or A^T.

- **framework for extending data structures**

Object Oriented interface

```
XYYYMM( TRANSA, M, N, K, ALPHA, args(A),
        B, LDB, BETA, C, LDC, WORK, LWORK)
```

Point entry:

COO	**Coordinate**
CSC	**Comressed sparse column**
CSR	**Compressd sparse row**
DIA	**Sparse diagonal**
ELL	**Ellpack/Itpack**
JAD	**Jagged diagonal**
SKY	**Skyline**
Block entry:	
BCO	**Block coordinate**
BSC	**Block comressed sparse column**
BSR	**Block compressd sparse row**
BDI	**Block sparse diagonal**
BEL	**Block Ellpack/Itpack**
VBR	**Variable block compressed sparse row**

TEMPLATES – REFERENCES

- Templates software and documentation for $Ax = b$ can be obtained via:
  - WWW: `http://www.netlib.org/templates`,
  - (anonymous) ftp `ftp.netlib.org`:
    - cd templates; get index
  - email `netlib@www.netlib.org` with the message:

    `send index from templates`

- R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994.