Some code exercises in Fortran 90

Presented by Dr. Ronan Dupont
July 1, 2025

Interview for the Position:

Postdoctoral Position in Numerical Linear Algebra
Nagoya University, Japan

Moonshot R&D Program – "Backcasting digital system by super-dimensional state engineering"

Supervised by Dr. Shao-Liang Zhang and Dr. Tomohiro Sogabe

Table des Matières

Conjugate Gradient for $Ax = b$ (SPD matrix)

Algorithm (Mathematical Form) - **Input:** Initial guess x_0 (approximate or 0).

$r_0 := b - Ax_0$
if $\|r_0\|$ small **then return** x_0
end if
$p_0 := r_0$
$k := 0$
repeat
  $\alpha_k := \frac{r_k^\top r_k}{p_k^\top A p_k}$
  $x_{k+1} := x_k + \alpha_k p_k$
  $r_{k+1} := r_k - \alpha_k A p_k$
  if $\|r_{k+1}\|$ small **then break**
  end if
  $\beta_k := \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$
  $p_{k+1} := r_{k+1} + \beta_k p_k$
  $k := k + 1$
until convergence
return x_{k+1}

Initialization of Conjugate Gradient

Initialization - **Input:** Initial guess x_0 (approximate or 0).

1: $r_0 := b - Ax_0$
2: **if** $\|r_0\|$ small **then return** x_0
3: **end if**
4: $p_0 := r_0$
5: $k := 0$

```
r = b - MATMUL(A, x) ! Initial residual
r_norm = SQRT(SUM(r**2))
PRINT *, "initial residual norm = ", r_norm
p = r
k = 0
max_iter = 1000
```

Main Loop of Conjugate Gradient

Main loop.

1: **repeat**
2: $\quad \alpha_k := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A \mathbf{p}_k}$
3: $\quad \mathbf{x}_{k+1} := \mathbf{x}_k + \alpha_k \mathbf{p}_k$
4: $\quad \mathbf{r}_{k+1} := \mathbf{r}_k - \alpha_k A \mathbf{p}_k$
5: $\quad$ **if** $\|\mathbf{r}_{k+1}\|$ small **then break**
6: $\quad$ **end if**
7: $\quad \beta_k := \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$
8: $\quad \mathbf{p}_{k+1} := \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
9: $\quad k := k + 1$
10: **until** convergence
11: **return** $\mathbf{x}_{k+1}$

```
1  DO k = 1, max_iter ! Main loop
2      alpha = DOT_PRODUCT(r, r) / DOT_PRODUCT(p, MATMUL(A, p))
3      x = x + alpha * p ! Update solution
4      r_new = r - alpha * MATMUL(A, p) ! Update residual
5      r_norm_new = SQRT(SUM(r_new**2)) ! Update residual
6      IF (r_norm_new < 1.0e-6) EXIT ! Go out of the loop
7      beta = DOT_PRODUCT(r_new, r_new) / DOT_PRODUCT(r, r) ! Beta
8      p = r_new + beta * p ! Update search direction
9      r = r_new ! Update residual for next iteration
10     r_norm = r_norm_new ! Update residual for next iteration
11 ENDDO
12 PRINT *, MATMUL(A, x) - b
```

Sparse with COO Matrix, Can also be parallelized using OPENMP

```
! Definition of the hollow matrix (COO)
! 3x3 tridiagonal matrix:
! 4 -1 0
!-1 4 -1
! 0 -1 4
VAL = (/ 4.0, -1.0, -1.0, 4.0, -1.0, -1.0, 4.0 /)
ROW = (/ 1, 1, 2, 2, 2, 3, 3 /)
COL = (/ 1, 2, 1, 2, 3, 2, 3 /)
B = (/ 1.0, 2.0, 3.0 /)
X = (/ 1.0, 1.0, 1.0 /)
```

```
CALL SPMV(N, NNZ, VAL, ROW, COL, P, AP)
R = B - AP ! Initialization
ALPHA = DOT_PRODUCT(R, R) / DOT_PRODUCT(P, AP) ! Loop
```

```
SUBROUTINE SPMV(N, NNZ, VAL, ROW, COL, V, RES)
  INTEGER, INTENT(IN) :: N, NNZ
  REAL(KIND=KIND(0.DO)), INTENT(IN) :: VAL(NNZ), V(N)
  INTEGER, INTENT(IN) :: ROW(NNZ), COL(NNZ)
  REAL(KIND=KIND(0.DO)), INTENT(OUT) :: RES(N)
  INTEGER :: I

  RES = 0.0
  DO I = 1, NNZ
    RES(ROW(I)) = RES(ROW(I)) + VAL(I) * V(COL(I))
  END DO
END SUBROUTINE SPMV
```

Lanczos Algorithm for Hermitian A

Input: Hermitian $A \in \mathbb{C}^{n\times n}$, v_1 unit norm, max steps m (default: $m = n$).

$v_1 \in \mathbb{C}^n$ such that $\|v_1\| = 1$
$w_1' := Av_1$
$\alpha_1 := v_1^* w_1'$
$w_1 := w_1' - \alpha_1 v_1$
for $j = 2$ to m **do**
    $\beta_j := \|w_{j-1}\|$
    if $\beta_j = 0$ **then**
        choose v_j orthogonal to $v_1, \ldots, v_{j-1}$ with $\|v_j\| = 1$
    else
        $v_j := w_{j-1}/\beta_j$
    end if
    $w_j' := Av_j - \beta_j v_{j-1}$
    $\alpha_j := v_j^* w_j'$
    $w_j := w_j' - \alpha_j v_j$
end for
return $V = [v_1, \ldots, v_m]$, $T = V^* AV$ (real symmetric tridiagonal)

Initialization of the Lanczos Algorithm

Initialization.

1: $v_1 \in \mathbb{C}^n$ such that $\|v_1\| = 1$
2: $w_1' := Av_1$
3: $\alpha_1 := v_1^* w_1'$
4: $w_1 := w_1' - \alpha_1 v_1$

```
1 ! Initial vector (random, then normalized)
2 CALL RANDOM_NUMBER(V(:,1))
3 norm = SQRT(SUM(V(:,1)**2))
4 V(:,1) = V(:,1) / norm
5
6 ! Initial step
7 w_prime = MATMUL(A, V(:,1))
8 alpha(1) = DOT_PRODUCT(w_prime, V(:,1))
9 w = w_prime - alpha(1) * V(:,1)
```

Main Loop of the Lanczos Algorithm

Main loop.

1: **for** $j = 2$ to m **do**
2: $\beta_j := \|w_{j-1}\|$
3: **if** $\beta_j = 0$ **then**
4: choose v_j orthogonal to $v_1, \ldots, v_{j-1}$ with $\|v_j\| = 1$
5: **else**
6: $v_j := w_{j-1}/\beta_j$
7: **end if**
8: $w'_j := Av_j - \beta_j v_{j-1}$
9: $\alpha_j := v_j^* w'_j$
10: $w_j := w'_j - \alpha_j v_j$
11: **end for**
12: **return** $V = [v_1, \ldots, v_m]$, $T = V^* AV$ (real symmetric tridiagonal)

```
1  DO j = 2, m
2      beta(j) = SQRT(SUM(w**2))
3      IF (beta(j) /= 0.0D0) THEN
4          V(:,j) = w / beta(j)
5      ELSE
6          PRINT *, 'beta(', j, ') = 0. Lanczos stops.  Because Gram-
                Schmidt requieres too much computation.'
7          EXIT
8      END IF
9      w_prime = MATMUL(A, V(:,j)) - beta(j) * V(:,j-1)
10     alpha(j) = DOT_PRODUCT(w_prime, V(:,j))
11     w = w_prime - alpha(j) * V(:,j)
12 END DO
```

Construction of the Tridiagonal Matrix

Construction of the Tridiagonal Matrix.

1: **return** $V = [v_1, \ldots, v_m]$, $T = V^* A V$ (real symmetric tridiagonal)

```
1 ! BUILD TRIDIAGONAL MATRIX T
2 T = 0.0D0
3 DO i = 1, m
4     T(i,i) = alpha(i)
5     IF (i < m) THEN
6         T(i,i+1) = beta(i+1)
7         T(i+1,i) = beta(i+1)
8     END IF
9 END DO
```

Givens QR Factorization without Explicit Q

Input: $A \in \mathbb{R}^{m \times n}$

Initialize: $indexI = 0$, $indexJ = 0$, $C = 0$, $S = 0$
for $i = 1$ to n **do**
  for $j = i + 1$ to m **do**
    $c := \frac{A(i,i)}{\sqrt{A(i,i)^2 + A(j,i)^2}}$
    $s := \frac{A(j,i)}{\sqrt{A(i,i)^2 + A(j,i)^2}}$
    $A(i,:) := cA(i,:) + sA(j,:)$
    $A(j,:) := -sA(i,:) + cA(j,:)$
    $indexI(j,i) := i$, $indexJ(j,i) := j$
    $C(j,i) := c$, $S(j,i) := s$
  end for
end for
return $R = A$, $indexI$, $indexJ$, C, S

Explicit Construction of Q

Input: *indexI*, *indexJ*, C, S

Initialize $Q = I$
for $i = 1$ to n **do**
    for $j = i + 1$ to m **do**
        if $indexI(j, i) > 0$ **then**
            $c := C(j, i)$, $s := S(j, i)$
            $Q(:, i) := cQ(:, i) + sQ(:, j)$
            $Q(:, j) := -sQ(:, i) + cQ(:, j)$
        end if
    end for
end for
return Q

Iterative QR with Givens Rotations

Iterative QR algorithm with convergence check.

1: Initialize: A, tol, max_iter, $Q = I$
2: **for** $iter = 1$ to max_iter **do**
3: Compute Givens QR: $A = R$, store $indexI$, $indexJ$, C, S
4: Construct Q using stored rotations
5: $A := RQ$
6: Compute $\|A - \text{diag}(A)\|_F^2$
7: **if** off-diagonal norm $< tol$ **then**
8: **return** approximate eigenvalues $A(i, i)$
9: **end if**
10: **end for**

Iterative QR with Givens Rotations

```
1  DO iter = 1, max_iter
2    R = A
3    Q = 0.0D0
4    Q(i,i) = 1.0D0
5    DO i = 1, n ! QR by Givens
6      DO j = i+1, n
7        denom = SQRT(R(i,i)**2 + R(j,i)**2)
8        cc = R(i,i)/denom
9        ss = R(j,i)/denom
10       tmp_row = cc*R(i,:) + ss*R(j,:)
11       R(j,:) = -ss*R(i,:) + cc*R(j,:)
12       R(i,:) = tmp_row
13       ! Store rotation
14       indexI(j,i)=i; indexJ(j,i)=j
15       C(j,i)=cc; S(j,i)=ss
16     END DO
17   END DO
18   DO i = 1, n ! Apply rotations to Q
19     DO j = i+1, n
20       IF (indexI(j,i) > 0) THEN
21         tmp_row = cc*Q(:,i) + ss*Q(:,j)
22         Q(:,j) = -ss*Q(:,i) + cc*Q(:,j)
23         Q(:,i) = tmp_row
24       END IF
25     END DO
26   END DO
27   A = MATMUL(R, Q)
28   norm_offdiag = 0.0D0
29   DO i = 1, n
30     DO j = 1, n
31       IF (i /= j) norm_offdiag += A(i,j)**2
32     END DO
33   END DO
34   IF (SQRT(norm_offdiag) < tol) EXIT ! Check convergence
35 END DO
```

Symmetric QR Algorithm with Givens Rotations for tridiagonal matrices

Can be also optimized for tridiagonal matrices.