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 $A\mathrm{x} = \mathrm{b}$ (SPD matrix)

Algorithm (Mathematical Form) - **Input:** Initial guess $\mathbf{x_0}$ (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:  $k := 0$
6:  **repeat**
7:      $\alpha_k := \frac{r_k^\top r_k}{p_k^\top A p_k}$
8:      $x_{k+1} := x_k + \alpha_k p_k$
9:      $r_{k+1} := r_k - \alpha_k A p_k$
10:     **if** $\|r_{k+1}\|$ small **then break**
11:     **end if**
12:     $\beta_k := \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$
13:     $p_{k+1} := r_{k+1} + \beta_k p_k$
14:     $k := k + 1$
15: **until** convergence
16: **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: $k := 0$
```

```fortran
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:     $\alpha_k := \frac{\mathbf{r}_k^\top \mathbf{r}_k}{\mathbf{p}_k^\top A \mathbf{p}_k}$
3:     $x_{k+1} := x_k + \alpha_k p_k$
4:     $r_{k+1} := r_k - \alpha_k A p_k$
5:     if $\|r_{k+1}\|$ small then break
6:     end if
7:     $\beta_k := \frac{\mathbf{r}_{k+1}^\top \mathbf{r}_{k+1}}{\mathbf{r}_k^\top \mathbf{r}_k}$
8:     $p_{k+1} := r_{k+1} + \beta_k p_k$
9:     $k := k + 1$
10: until convergence
11: return $x_{k+1}$
```

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

Sparse with COO Matrix, Can also be parallelized using OPENMP

```fortran
! 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 /)
```

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

```fortran
SUBROUTINE SPMV(N, NNZ, VAL, ROW, COL, V, RES)
  INTEGER, INTENT(IN) :: N, NNZ
  REAL(KIND=KIND(0.D0)), INTENT(IN) :: VAL(NNZ), V(N)
  INTEGER, INTENT(IN) :: ROW(NNZ), COL(NNZ)
  REAL(KIND=KIND(0.D0)), 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 $A \in \mathbb{C}^{n\times n}$, $\mathbf{v_1}$ unit norm, max steps m (default: $m = n$).

```
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: $w_1 := w_1' - \alpha_1 v_1$
5: for $j = 2$ to $m$ do
6:     $\beta_j := \|w_{j-1}\|$
7:     if $\beta_j = 0$ then
8:         choose $v_j$ orthogonal to $v_1, \dots, v_{j-1}$ with $\|v_j\| = 1$
9:     else
10:        $v_j := w_{j-1}/\beta_j$
11:    end if
12:    $w_j' := Av_j - \beta_j v_{j-1}$
13:    $\alpha_j := v_j^* w_j'$
14:    $w_j := w_j' - \alpha_j v_j$
15: end for
16: return $V = [v_1, \dots, v_m]$, $T = V^* AV$ (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: $w_1 := w_1' - \alpha_1 v_1$
```

```fortran
! Initial vector (random, then normalized)
CALL RANDOM_NUMBER(V(:,1))
norm = SQRT(SUM(V(:,1)**2))
V(:,1) = V(:,1) / norm

! Initial step
w_prime = MATMUL(A, V(:,1))
alpha(1) = DOT_PRODUCT(w_prime, V(:,1))
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 $v_1, \dots, v_{j-1}$ 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 $V = [v_1, \dots, v_m]$, $T = V^* AV$ (real symmetric tridiagonal)
```

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

Construction of the Tridiagonal Matrix

Construction of the Tridiagonal Matrix.

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

```fortran
! BUILD TRIDIAGONAL MATRIX T
T = 0.0D0
DO i = 1, m
    T(i,i) = alpha(i)
    IF (i < m) THEN
        T(i,i+1) = beta(i+1)
        T(i+1,i) = beta(i+1)
    END IF
END DO
```

Givens QR Factorization without Explicit Q

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

```
 1: Initialize: $indexI = 0,\ indexJ = 0,\ C = 0,\ S = 0$
 2: for $i = 1$ to $n$ do
 3:     for $j = i+1$ to $m$ do
 4:         $c := \frac{A(i,i)}{\sqrt{A(i,i)^2 + A(j,i)^2}}$
 5:         $s := \frac{A(j,i)}{\sqrt{A(i,i)^2 + A(j,i)^2}}$
 6:         $A(i,:) := cA(i,:) + sA(j,:)$
 7:         $A(j,:) := -sA(i,:) + cA(j,:)$
 8:         $indexI(j,i) := i,\ indexJ(j,i) := j$
 9:         $C(j,i) := c,\ S(j,i) := s$
10:     end for
11: end for
12: return $R = A,\ indexI,\ indexJ,\ C,\ S$
```

Explicit Construction of Q

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

```
1: Initialize $Q = I$
2: for $i = 1$ to $n$ do
3:     for $j = i + 1$ to $m$ do
4:         if $indexI(j, i) > 0$ then
5:             $c := C(j, i)$, $s := S(j, i)$
6:             $Q(:, i) := cQ(:, i) + sQ(:, j)$
7:             $Q(:, j) := -sQ(:, i) + cQ(:, j)$
8:         end if
9:     end for
10: end for
11: 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 $\|A - \mathrm{diag}(A)\|_F^2$
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

```fortran
DO iter = 1, max_iter
  R = A
  Q = 0.0D0
  Q(i,i) = 1.0D0
  DO i = 1, n ! QR by Givens
    DO j = i+1, n
      denom = SQRT(R(i,i)**2 + R(j,i)**2)
      cc = R(i,i)/denom
      ss = R(j,i)/denom
      tmp_row = cc*R(i,:) + ss*R(j,:)
      R(j,:) = -ss*R(i,:) + cc*R(j,:)
      R(i,:) = tmp_row
      ! Store rotation
      indexI(j,i)=i; indexJ(j,i)=j
      C(j,i)=cc; S(j,i)=ss
    END DO
  END DO
  DO i = 1, n ! Apply rotations to Q
    DO j = i+1, n
      IF (indexI(j,i) > 0) THEN
        tmp_row = cc*Q(:,i) + ss*Q(:,j)
        Q(:,j) = -ss*Q(:,i) + cc*Q(:,j)
        Q(:,i) = tmp_row
      END IF
    END DO
  END DO
  A = MATMUL(R, Q)
  norm_offdiag = 0.0D0
  DO i = 1, n
    DO j = 1, n
      IF (i /= j) norm_offdiag += A(i,j)**2
    END DO
  END DO
  IF (SQRT(norm_offdiag) < tol) EXIT ! Check convergence
END DO
```

Symmetric QR Algorithm with Givens Rotations for tridiagonal matrixs

Can be also optimized for tridiagonal matrixs.