Matrix- und Vektoroperationen

1. **Arbeiten mit vollbesetzten Matrizen:**
   Wegen der starren Feldvereinbarungen mit konstanten Dimensionen wird oft in Fortran eine pseudo-dynamische Speicherverwaltung verwendet, indem der Programmierer selbst die Felder innerhalb des zusammenhängend vereinbarten Speicherplatzes verteilt[1]:

   - **Hauptprogramm**: vereinbart ein großes Feld als Platzhalter, greift nicht auf einzelne Matrixelemente zu. (Das Hauptprogramm **lässt** die Unterprogramme für sich arbeiten!)

     ```
     PARAMETER (MAX_S = 50 000 000)
     DIMENSION S( MAX_S )
     ```

     Eine *„Matrix“* wird grundsätzlich nur in Unterprogrammen verwendet, und zwar mit variabler Dimension, z.B.

     ```
     SUBROUTINE READMAT (N,A)
     REAL A(N,N)
     ```

   - Nachdem die aktuelle Dimension `n` der $n \times n$-Matrix bekannt ist (z.B. durch Nutzereingabe oder aus einer Datei eingelesen), kann das Unterprogramm zum Einlesen oder Belegen der Matrixelemente gerufen werden:

     ```
     n=...       bzw.   READ(*,*) n
     call READMAT(n,S)
     ```

   - **Zur Erinnerung:** Der aktuelle Parameter `S` liefert dem Unterprogramm nur die Startadresse eines Feldes, die Vereinbarung im Unterprogramm bestimmt, wie der Speicherinhalt zu deuten ist.
     Braucht das Programm etwa noch zwei Vektoren x und y, so nimmt man diese ebenfalls aus dem „Reservoir“ `S`:

     ```
     jA = 1              ! Startindex der Matrix
     jX = jA+n*n         ! n*n Elemente fuer A reservieren
     jY = jX+n           ! n Elemente fuer X reservieren
     jFrei = jY+n        ! n Elemente fuer Y reservieren
     if (jFrei .GT. MAX_S) ... Fehler ...
     ```

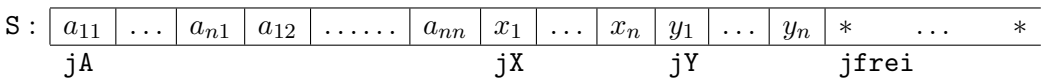
     und ruft dann ein Unterprogramm der Art

     ```
     SUBROUTINE AXMUL (N,A,X,Y)
     REAL A(N,N), X(N), Y(N)
     ```

     folgendermaßen auf:

     ```
     call AXMUL ( n, S(jA), S(jX), S(jY) )
     ```

   Der Inhalt des Feldes `S` wird dadurch folgendermaßen interpretiert:

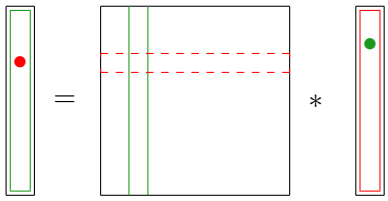
   

2. Als Übungsaufgabe sollen verschiedene Versionen zur **Multiplikation einer Matrix mit einem Vektor** ($y := Ax$ und $y := A^\top x$) verglichen werden:

[1] bei Fortran 90 besteht dazu keine Notwendigkeit mehr, aber diese Methode kann in Einzelfällen trotzdem aus Effizienzgründen und hinsichtlich Portabilität vorteilhaft sein.

(1) „normale“ Variante, d.h. 2 Schleifen (i und j) für $y_i = \sum_j a_{ij}x_j$ (bzw. $a_{ji}x_j$).
Dabei kann man deutliche Rechenzeitunterschiede beobachten.

(2) Verwendung der Unterprogramme für Vektoroperationen `vsaxpy` und `scapr` aus vorigen Übungen, indem man beachtet, dass eine Matrix spaltenweise gespeichert ist, d. h. mit `A(1,J)` wird die Startadresse des *Vektors* „Spalte j von A“ (bzw. „Zeile j von $A^\top$“) angegeben.



Offensichtlich kann man nun jede der beiden Vektoroperationen günstig zur Realisierung einer der beiden Matrix-Vektor-Multiplikationen Ax bzw. $A^\top x$ verwenden, die dann nur noch eine einfache Schleife enthält, in der die entsprechende Vektoroperation gerufen wird:
$y = Ax$ kann als Linearkombination der Spalten von A ($\sum x_j A_{*j}$) dargestellt werden, $y = A^\top x$ ist der Vektor aus den Skalarprodukten der Zeilen von $A^\top$ mit x, d.h. $\left(\langle A^\top_{i*}, x\rangle = \langle (A_{*i})^\top, x\rangle\right)$.

3. **Arbeit mit speziell strukturierten Matrizen**

- Wenn von **symmetrischen Matrizen** nur das obere (*untere*) Dreieck gespeichert wird, um knapp die Hälfte des Speicherplatzes zu sparen, muss die Matrix als lineares Feld (d.h. mit nur **einem** Index) vereinbart und verarbeitet werden. Wir betrachten die drei Varianten
  (a) Speicherung des oberen Dreiecks zeilenweise (*unteres Dreieck spaltenweise*)
  (b) Speicherung des oberen Dreiecks spaltenweise (*unteres Dreieck zeilenweise*)
  (c) Speicherung des oberen Dreiecks in diagonaler Richtung, beginnend mit der Hauptdiagonalen
- Für jede (bzw. mindestens eine) der genannten Varianten ist ein Unterprogramm zu schreiben, das (als Testbeispiel) die Hilbertmatrix ($a_{ij} = 1.0/(i+j-1)$) in dieser Speicherform auf dem formalen Parameter `A` zurückgibt.

  `SUBROUTINE SETMAT_Z(N,A)` $\leftarrow$ $a_{11}, \ldots, a_{1n}, a_{22}, \ldots, a_{2n}, \ldots, a_{nn}$
  `SUBROUTINE SETMAT_S(N,A)` $\leftarrow$ $a_{11}, a_{12}, a_{22}, a_{13}, \ldots, a_{33}, \ldots, a_{nn}$
  `SUBROUTINE SETMAT_D(N,A)` $\leftarrow$ $a_{11}, \ldots, a_{nn}, a_{12}, \ldots, a_{n-1,n}, \ldots, a_{1n}$
- Für jede Variante ist das zugehörige Unterprogramm zur Matrix-Vektor-Multiplikation zu schreiben und zu testen. Die Multiplikation ist so durchzuführen, dass auf jedes Matrixelement nur jeweils einmal zugegriffen wird (in der Reihenfolge ihrer Anordnung im Speicher).

```
SUBROUTINE AXMUL_Z (N,A,X,Y)
SUBROUTINE AXMUL_S (N,A,X,Y)
SUBROUTINE AXMUL_D (N,A,X,Y)
```

- Zum Testen soll das Hauptprogramm folgendes realisieren:
  – Die Dimension `N` wird vom Nutzer abgefragt (wiederholte Abfrage, Programm-Ende erst bei Eingabe von $N \leq 0$).
  – Für kleine Dimensionen (`N`< 10) ist der Ergebnisvektor auszugeben (Kontrolle der Rechnung).
  – Für große Dimensionen (`N`≈ 5000) soll die Rechenzeit verglichen werden (auch mit der Variante für allgemeine vollbesetzte Matrizen, s.o.).