```fortran
MODULE REAL_PRECISION  ! module for 64-bit arithmetic
  INTEGER, PARAMETER:: R8=SELECTED_REAL_KIND(13)
END MODULE REAL_PRECISION

MODULE SPLINES
  USE REAL_PRECISION
  IMPLICIT NONE

CONTAINS

SUBROUTINE FIT_SPLINE(BREAKPOINTS, VALUES, KNOTS, COEFFICIENTS, STATUS)
  ! Subroutine for computing a linear combination of B-splines that
  ! interpolates the given function value (and function derivatives)
  ! at the given breakpoints.
  !
  ! INPUT:
  !   BREAKPOINTS(N) -- The increasing real-valued locations of
  !                     the breakpoints for the interpolating spline.
  !   VALUES(N,C)    -- VALUES(I,J) contains the (J-1)st derivative at
  !                     BREAKPOINTS(I) to be interpolated.
  !
  ! OUTPUT:
  !   KNOTS(N*C+2*C)    -- The nondecreasing real-valued locations
  !                        of the knots for the B-spline basis.
  !   COEFFICIENTS(N*C) -- The coefficients for the B-splines
  !                        that define the interpolating spline.
  !   STATUS -- Integer representing the subroutine execution status:
  !       0    Successful execution.
  !       1    SIZE(BREAKPOINTS) is less than 1.
  !       2    SIZE(VALUES) is less then 1.
  !       3    SIZE(VALUES,1) does not equal SIZE(BREAKPOINTS).
  !       4    Bad SIZE(KNOTS), should be size N*C + 2*C.
  !       5    Bad SIZE(COEFFICIENTS), should be N*C.
  !       6    Elements of BREAKPOINTS are not strictly increasing.
  !       7    The computed spline does not match the provided VALUES
  !            and this fit should be disregarded. This arises when
  !            the scaling of function values and derivative values
  !            causes the resulting linear system to have a
  !            prohibitively large condition number.
  !     >10    10 plus the info flag as returned by DGBSV from LAPACK.
  !
  !
  ! DESCRIPTION:
  !
  !   This subroutine uses a B-spline basis to interpolate given
  !   function values (and derivative values) at unique breakpoints.
  !   The interpolating spline is returned in terms of KNOTS and
  !   COEFFICIENTS that define the underlying B-splines and the
  !   corresponding linear combination that interpolates given data
  !   respectively. This function uses the subroutine EVAL_BSPLINE to
  !   evaluate the B-splines at all knots and the LAPACK routine
  !
  !     DGBSV
  !
  !   to compute the coefficients of all component B-splines. The
  !   difference between the provided function (and derivative) values
  !   and the actual values produced by this code can vary depending
  !   on the spacing of the knots and the magnitudes of the values
  !   provided. When the condition number of the intermediate linear
  !   system grows prohibitively large, this routine may fail to
  !   produce a correct set of coefficients and return STATUS code 7.
  !   For very high levels of continuity, or when this routine failes,
  !   a Newton form of polynomial representation should be used instead.
  !
  REAL(KIND=R8), INTENT(IN),  DIMENSION(:)   :: BREAKPOINTS
  REAL(KIND=R8), INTENT(IN),  DIMENSION(:,:) :: VALUES
  REAL(KIND=R8), INTENT(OUT), DIMENSION(SIZE(VALUES)+2*SIZE(VALUES,2)) :: KNOTS
  REAL(KIND=R8), INTENT(OUT), DIMENSION(SIZE(VALUES)) :: COEFFICIENTS
  INTEGER, INTENT(OUT) :: STATUS
  ! Local variables.
  INTEGER, DIMENSION(SIZE(COEFFICIENTS)) :: IPIV
  REAL(KIND=R8), DIMENSION(1 + 3*(2*SIZE(VALUES,2)-1), SIZE(VALUES)) :: AB
  REAL(KIND=R8) :: MAX_ERROR
  INTEGER :: C, K, N, NC, DERIV, DEGREE, STEP, &
       FIRST_BREAK, FIRST_ROW, FIRST_KNOT, &
       LAST_BREAK,  LAST_ROW,  LAST_KNOT
  ! LAPACK subroutine for solving banded linear systems.
  EXTERNAL :: DGBSV

  ! Define some local variables for notational convenience.
  N = SIZE(BREAKPOINTS) ! number of breakpoints
  C = SIZE(VALUES,2)    ! number of continuity conditions
  NC = SIZE(VALUES)     ! number of coefficients
  K = NC + 2*C          ! number of knots
  DEGREE = 2*C - 1      ! degree of the B-splines
  STATUS = 0            ! execution status

  ! Check the shape of incoming arrays.
  IF (N .LT. 1) THEN
     STATUS = 1
     RETURN
  ELSE IF (NC .LT. 1) THEN
     STATUS = 2
     RETURN
  ELSE IF (SIZE(VALUES,1) .NE. N) THEN
     STATUS = 3
     RETURN
  ELSE IF (SIZE(KNOTS) .NE. NC+2*C) THEN
     STATUS = 4
     RETURN
  ELSE IF (SIZE(COEFFICIENTS) .NE. NC) THEN
     STATUS = 5
     RETURN
  END IF

  ! Verify that BREAKPOINTS are increasing.
  DO STEP = 1, N - 1
     IF (BREAKPOINTS(STEP) .GE. BREAKPOINTS(STEP+1)) THEN
        STATUS = 6
        RETURN
     END IF
  END DO

  ! Copy over the knots that will define the B-spline representation.
  ! Each knot will be repeataed C times to maintain the necessary
  ! level of continuity for this spline.
  KNOTS(1:2*C) = BREAKPOINTS(1)
  DO STEP = 2, N-1
     KNOTS(STEP*C+1 : (STEP+1)*C) = BREAKPOINTS(STEP)
  END DO
  ! Assign the last knot to exist a small step outside the supported
  ! interval to ensure the B-spline basis functions are nonzero at the
  ! rightmost breakpoint.
  KNOTS(K-DEGREE:) = BREAKPOINTS(N) + &
       BREAKPOINTS(N) * SQRT(EPSILON(BREAKPOINTS(N)))

  ! The next block of code evaluates each B-spline and it's
  ! derivatives at all breakpoints. The first and last elements of
  ! BREAKPOINTS will be repeated DEGREE+1 times and each internal
  ! breakpoint will be repeated C times. As a result, each B-spline
  ! will have nonzero values for at most three breakpoints when
  ! computing function value and C-1 derivatives. The coefficients for
  ! the B-spline basis are determined by a linear solve. In all, each
  ! B-spline basis function will have at most 3*C nonzero values (in
  ! each column) and there will be N*C rows.
  !
  ! For example, a C^1 interpolating spline over three breakpoints
  ! will match function value and first derivative at each breakpoint
  ! requiring six fourth order (third degree) B-splines each composed
  ! from five knots. Below, the six B-splines are numbered (first
  ! number, columns) and may be nonzero at the three breakpoints
  ! (middle letter, rows) for each function value (odd rows, terms end
  ! with 0) and first derivative (even rows, terms end with 1). The
  ! linear system will look like:
  !
```

```
.
!       B-SPLINE VALUES AT BREAKPOINTS     SPLINE           VALUES
!        1st  2nd  3rd  4th  5th  6th    COEFICIENTS
!
!      _                              _     _   _           _    _
!     |                                |   |     |          |      |
!   B |  1a0  2a0  3a0  4a0            |   |  1  |          |  a0  |
!   R |  1a1  2a1  3a1  4a1            |   |  2  |          |  a1  |
!   E |  1b0  2b0  3b0  4b0  5b0  6b0  |   |  3  |   ===    |  b0  |
!   A |  1b1  2b1  3b1  4b1  5b1  6b0  |   |  4  |   ===    |  b1  |
!   K |            3c0  4c0  5c0  6c0  |   |  5  |          |  c0  |
!   S |            3c1  4c1  5c1  6c0  |   |  6  |          |  c1  |
!     |_                              _|   |_   _|          |_    _|
!
! Notice this matrix is banded with lower / upper bandwidths equal
! to (one less than the maximum number of breakpoints for which a
! spline takes on a nonzero value) times (the number of continuity
! conditions) minus (one). In general KL = KU = DEGREE = 2*C - 1.

! Initialize all values in AB to zero.
AB(:,:) = 0_R8
! Evaluate all B-splines at all breakpoints (walking through rows).
DO STEP = 1, NC
   ! Compute indices of the first and last knot for the current B-spline.
   FIRST_KNOT = STEP
   LAST_KNOT  = STEP + 2*C
   ! Compute the row indices in "A" that would be accessed.
   FIRST_ROW = ((STEP-1)/C - 1) * C + 1
   LAST_ROW  = FIRST_ROW + 3*C - 1
   ! Only two breakpoints will be covered for the first C B-splines
   ! and the last C B-splines.
   IF   (STEP .LE. C)  FIRST_ROW = FIRST_ROW + C
   IF (STEP+C .GT. NC) LAST_ROW = LAST_ROW - C
   ! Compute the indices of the breakpoints that will be nonzero.
   FIRST_BREAK = FIRST_ROW / C + 1
   LAST_BREAK =  LAST_ROW  / C
   ! Convert the "i,j" indices in "A" to the banded storage scheme.
   ! The mapping is looks like   AB[KL+KU+1+i-j,j] = A[i,j]
   FIRST_ROW = 2*DEGREE+1 + FIRST_ROW - STEP
   LAST_ROW  = 2*DEGREE+1 + LAST_ROW  - STEP
   ! Evaluate this B-spline, computing function value and derivatives.
   DO DERIV = 0, C-1
      ! Place the evaluations into a block of a column in AB, shift
      ! according to which derivative is being evaluated and use a
      ! stride determined by the continuity (number of derivatives).
      AB(FIRST_ROW+DERIV:LAST_ROW:C,STEP) = &
           BREAKPOINTS(FIRST_BREAK:LAST_BREAK)
      CALL EVAL_BSPLINE(KNOTS(FIRST_KNOT:LAST_KNOT), &
           AB(FIRST_ROW+DERIV:LAST_ROW:C,STEP), STATUS, D=DERIV)
      ! ^ Correct usage is inherently enforced, only extrapolation
      !   warnings will be produced by this call. These
      !   extrapolation warnings are expected because underlying
      !   B-splines may not support the full interval.
   END DO
END DO
! Copy the VALUES into the COEFFICIENTS (output) variable.
DO STEP = 1, C
   COEFFICIENTS(STEP::C) = VALUES(:,STEP)
END DO
! Call the LAPACK subroutine to solve the banded linear system.
CALL DGBSV(NC, DEGREE, DEGREE, 1, AB, SIZE(AB,1), IPIV, &
     COEFFICIENTS, NC, STATUS)
! Check for errors in the execution of DGBSV, (this should not happen).
IF (STATUS .NE. 0) THEN
   STATUS = STATUS + 10
   RETURN
END IF
! Check to see if the linear system was correctly solved by looking at
! the difference between prouduced B-spline values and provided values.
MAX_ERROR = SQRT(SQRT(EPSILON(1.0_R8)))
DO DERIV = 0, C-1
   ! Reuse the first row of AB as scratch space (the first column
   ! might not be large enough, but the first row certainly is).
   AB(1,1:N) = BREAKPOINTS(:)
   ! Evaluate this spline at all breakpoints and Ignore the STATUS
     ! outcome because correct usage is already enforced.
     CALL EVAL_SPLINE(KNOTS, COEFFICIENTS, AB(1,1:N), STATUS, D=DERIV)
     ! Check the maximum difference between the provided values and
     ! the reproduced values by the interpolating spline.
     IF (MAXVAL(ABS(AB(1,1:N) - VALUES(:,DERIV+1))) .GT. MAX_ERROR) THEN
        STATUS = 7
        RETURN
     END IF
  END DO
END SUBROUTINE FIT_SPLINE


SUBROUTINE EVAL_SPLINE(KNOTS, COEFFICIENTS, XY, STATUS, D)
  ! Evaluate a spline construed with FIT_SPLINE. Similar interface
  ! to EVAL_BSPLINE. Evaluate D derivative at all XY, result in XY.
  !
  ! INPUT:
  !   KNOTS(N+2*C)    -- The nondecreasing real-valued locations of the
  !                      breakpoints for the underlying B-splines,
  !                      where "C" is the continuity level of the
  !                      spline plus one (I.e., C^1 spline -> C = 2).
  !   COEFFICIENTS(N) -- The coefficients assigned to each B-spline
  !                      that underpins this interpolating spline.
  !
  ! INPUT / OUTPUT:
  !   XY(Z) -- The locations at which the spline is evaluated on
  !            input, on output holds the value of the spline with
  !            KNOTS and COEFFICIENTS evaluated at the given locations.
  !
  ! OUTPUT:
  !   STATUS -- Integer representing subroutine exeuction status.
  !        0    Successful execution.
  !        1    Extrapolation warning, some X are outside of spline support.
  !        2    KNOTS contains at least one decreasing interval.
  !        3    KNOTS has size less than or equal to 1.
  !        4    KNOTS has an empty interior (KNOTS(1) = KNOTS(N+2*C)).
  !        5    Invalid COEFFICEINTS, size smaller than or equal to KNOTS.
  !
  ! OPTIONAL INPUT:
  !   D [= 0]  --  The derivative to take of the evaluated spline.
  !                When negative, this subroutine integrates the spline.
  !                The higher integrals of this spline are capped at
  !                the rightmost knot, using constant-valued extrapolation.
  !
  !
  ! DESCRIPTION:
  !
  !    This subroutine serves as a convenient wrapper to the
  !    underlying calls to EVAL_BSPLINE that need to be made to
  !    evaluate the full spline. Internally this uses a matrix-vector
  !    multiplication of the B-spline evaluations with the assigned
  !    coefficients. This requires O(Z*N) memory, meaning single XY
  !    points should be evaluated at a time when memory-constrained.
  !
  REAL(KIND=R8), INTENT(IN), DIMENSION(:) :: KNOTS, COEFFICIENTS
  REAL(KIND=R8), INTENT(INOUT), DIMENSION(:) :: XY
  INTEGER, INTENT(IN), OPTIONAL :: D
  INTEGER, INTENT(OUT) :: STATUS
  ! Local variables.
  INTEGER :: DERIV, STEP, ORDER
  REAL(KIND=R8), DIMENSION(SIZE(XY), SIZE(COEFFICIENTS)) :: VALUES

  ! Check for various size-related errors.
  IF (SIZE(KNOTS) .LE. 1) THEN
     STATUS = 3
  ELSE IF (SIZE(COEFFICIENTS) .LE. SIZE(KNOTS)) THEN
     STATUS = 5
  END IF
  ! Check for valid (nondecreasing) knot sequence.
  DO STEP = 1, SIZE(KNOTS)-1
     IF (KNOTS(STEP) .GT. KNOTS(STEP+1)) THEN
        STATUS = 2
        RETURN
```

```fortran
      END IF
   END DO
   ! Check to make sure the support interval has positive size.
   IF (KNOTS(1) .EQ. KNOTS(SIZE(KNOTS))) THEN
      STATUS = 4
      RETURN
   END IF

   ! Compute the ORDER (number of knots minus one) for each B-spline.
   ORDER = SIZE(KNOTS) - SIZE(COEFFICIENTS)
   ! Assign the local value of the optional derivative "D" argument.
   set_derivative : IF (PRESENT(D)) THEN ; DERIV = D
   ELSE ; DERIV = 0
   END IF set_derivative

   ! Evaluate all splines at all the X positions.
   DO STEP = 1, SIZE(COEFFICIENTS)
      IF (KNOTS(STEP) .EQ. KNOTS(STEP+ORDER)) CYCLE
      ! ^ If this internal B-spline has no support, skip it.
      VALUES(:,STEP) = XY(:)
      CALL EVAL_BSPLINE(KNOTS(STEP:STEP+ORDER), VALUES(:,STEP), &
           STATUS, D=DERIV)
      ! ^ Correct usage is inherently enforced, only extrapolation
      !   warnings will be produced by this call. These
      !   extrapolation warnings are expected because underlying
      !   B-splines may not support the full interval.
   END DO
   ! Set the EXTRAPOLATION status flag.
   IF ((MINVAL(XY(:)) .LT. KNOTS(1)) .OR. &
        (MAXVAL(XY(:)) .GE. KNOTS(SIZE(KNOTS)))) THEN
      STATUS = 1
   ELSE
      STATUS = 0
   END IF
   ! Store the values into Y as the weighted sums of B-spline evaluations.
   XY(:) = MATMUL(VALUES(:,:), COEFFICIENTS(:))
 END SUBROUTINE EVAL_SPLINE


 SUBROUTINE EVAL_BSPLINE(KNOTS, XY, STATUS, D)
   ! Subroutine for evaluating a B-spline with provided knot sequence.
   !
   ! INPUT:
   !   KNOTS(N) -- The nondecreasing sequence of break points for the B-spline.
   !
   ! INPUT / OUTPUT:
   !   XY(Z) -- The locations at which the B-spline is evaluated on
   !            input, on output holds the value of the B-spline with
   !            prescribed knots evaluated at the given X locations.
   !
   ! OPTIONAL INPUT:
   !   D [= 0]  --  The derivative to take of the evaluated B-spline.
   !                When negative, this subroutine integrates the B-spline.
   !
   ! OUTPUT:
   !   STATUS -- Execution status of this subroutine on exit.
   !       0    Successful execution.
   !       1    Extrapolation warning, some points were outside of knots.
   !       2    Invalid knot sequence (not entirely nondecreasing).
   !       3    Invalid size for KNOTS (less than or equal to 1).
   !
   ! DESCRIPTION:
   !
   !    This function uses the recurrence relation defining a B-spline:
   !
   !      B_{K,1}(X)   =   1     if KNOTS(K) <= X < KNOTS(K+1),
   !                       0     otherwise,
   !
   !    where K is the knot index, I = 2, ..., N-MAX(D,0)-1, and
   !
   !                               X - KNOTS(K)
   !      B_{K,I}(X) =      ------------------------- B_{K,I-1}(X)
   !                         KNOTS(K+I-1) - KNOTS(K)
   !
   !                             KNOTS(K+I) - X
   !                     +  ------------------------- B_{K+1,I-1}(X).
   !                         KNOTS(K+I) - KNOTS(K+1)
   !
   !    All of the intermediate steps (I) are stored in a single block
   !    of memory that is reused for each step.
   !
   !    The computation of the integral of the B-spline proceeds from
   !    the above formula one integration step at a time by adding a
   !    duplicate of the last knot, raising the order of all
   !    intermediate B-splines, summing their values, and dividing the
   !    sums by the width of the supported interval and the integration
   !    coefficient.
   !
   !    For the computation of the derivative of the B-spline, the
   !    continuation of the standard recurrence relation is used that
   !    builds from I = N-D, ..., N-1 as
   !
   !                           (I-1) B_{K,I-1}(X)
   !      B_{K,I}(X) =      -------------------------
   !                         KNOTS(K+I-1) - KNOTS(K)
   !
   !                           (I-1) B_{K+1,I-1}(X)
   !                     -  -------------------------.
   !                         KNOTS(K+I) - KNOTS(K+1)
   !
   !     The final B-spline is right continuous, has nonzero value and
   !     derivatives on [KNOTS(1), KNOTS(N) everywhere except at the
   !     last knot, at which it is both left and right continuous.
   !
   REAL(KIND=R8), INTENT(IN), DIMENSION(:) :: KNOTS
   REAL(KIND=R8), INTENT(INOUT), DIMENSION(:) :: XY
   INTEGER, INTENT(OUT) :: STATUS
   INTEGER, INTENT(IN), OPTIONAL :: D
   ! Local variables.
   REAL(KIND=R8), DIMENSION(SIZE(XY), SIZE(KNOTS)) :: VALUES
   INTEGER :: K, N, DERIV, ORDER, STEP
   REAL(KIND=R8) :: DIV_LEFT, DIV_RIGHT, LAST_KNOT
   ! Assign the local value of the optional derivative "D" argument.
   set_derivative : IF (PRESENT(D)) THEN ; DERIV = D
   ELSE ; DERIV = 0
   END IF set_derivative
   ! Store local useful variable.
   N = SIZE(KNOTS)
   ORDER = N - 1
   LAST_KNOT = KNOTS(N)
   STATUS = 0
   ! Check for valid knot sequence.
   IF (N .LE. 1) THEN
      STATUS = 3
      RETURN
   END IF
   DO K = 1, N-1
      IF (KNOTS(K) .GT. KNOTS(K+1)) THEN
         STATUS = 2
         RETURN
      END IF
   END DO
   ! Check for extrapolation, set status if it is happening, but continue.
   IF ((MINVAL(XY(:)) .LT. KNOTS(1)) .OR. (MAXVAL(XY(:)) .GE. LAST_KNOT)) &
        STATUS = 1
   ! If this is a large enough derivative, we know it is zero everywhere.
   IF (DERIV+1 .GE. N) THEN
      XY(:) = 0.0_R8
      RETURN
   ! ---------------- Performing standard evaluation ------------------
   ! This is a standard B-spline with multiple unique knots, right continuous.
   ELSE
      ! Initialize all values to 0.
      VALUES(:,:) = 0.0_R8
      ! Assign the first value for each knot index.
      first_b_spline : DO K = 1, ORDER
         IF (KNOTS(K) .EQ. KNOTS(K+1)) CYCLE
```

```
      ! Compute all right-continuous order-1 B-spline values.
      WHERE ( (KNOTS(K) .LE. XY(:)) .AND. (XY(:) .LT. KNOTS(K+1)) )
         VALUES(:,K) = 1.0_R8
      END WHERE
   END DO first_b_spline
END IF

! Compute the remainder of B-spline by building up from the first.
! Omit the final steps of this computation for derivatives.
compute_spline : DO STEP = 2, N-1-MAX(DERIV,0)
   ! Cycle over each knot accumulating the values for the recurrence.
   DO K = 1, N - STEP
      ! Enforce nonzero divisors, intervals with 0 width add 0 value to the B-spline.
      DIV_LEFT = (KNOTS(K+STEP-1) - KNOTS(K))
      DIV_RIGHT = (KNOTS(K+STEP) - KNOTS(K+1))
      ! Compute the B-spline recurrence relation (cases based on divisor).
      IF (DIV_LEFT .GT. 0) THEN
         IF (DIV_RIGHT .GT. 0) THEN
            VALUES(:,K) = &
                 ((XY(:) - KNOTS(K))      / DIV_LEFT)  * VALUES(:,K) + &
                 ((KNOTS(K+STEP) - XY(:)) / DIV_RIGHT) * VALUES(:,K+1)
         ELSE
            VALUES(:,K) = &
                 ((XY(:) - KNOTS(K))      / DIV_LEFT)  * VALUES(:,K)
         END IF
      ELSE
         IF (DIV_RIGHT .GT. 0) THEN
            VALUES(:,K) = &
                 ((KNOTS(K+STEP) - XY(:)) / DIV_RIGHT) * VALUES(:,K+1)
         END IF
      END IF
   END DO
END DO compute_spline

! -------------------- Performing integration ----------------------
integration_or_differentiation : IF (DERIV .LT. 0) THEN
   ! Integrals will be nonzero on [LAST_KNOT, \infty).
   WHERE (LAST_KNOT .LE. XY(:))
      VALUES(:,N) = 1.0_R8
   END WHERE
   ! Loop through starting at the back, raising the order of all
   ! constituents to match the order of the first.
   raise_order : DO STEP = 1, ORDER-1
      DO K = N-STEP, ORDER
         DIV_LEFT = (LAST_KNOT - KNOTS(K))
         DIV_RIGHT = (LAST_KNOT - KNOTS(K+1))
         IF (DIV_LEFT .GT. 0) THEN
            IF (DIV_RIGHT .GT. 0) THEN
               VALUES(:,K) = &
                    ((XY(:) - KNOTS(K))  / DIV_LEFT)  * VALUES(:,K) + &
                    ((LAST_KNOT - XY(:)) / DIV_RIGHT) * VALUES(:,K+1)
            ELSE
               VALUES(:,K) = &
                    ((XY(:) - KNOTS(K))  / DIV_LEFT)  * VALUES(:,K)
            END IF
         ELSE
            IF (DIV_RIGHT .GT. 0) THEN
               VALUES(:,K) = &
                    ((LAST_KNOT - XY(:)) / DIV_RIGHT) * VALUES(:,K+1)
            END IF
         END IF
      END DO
   END DO raise_order

   ! Compute the integral(s) of the B-spline.
   compute_integral : DO STEP = 1, -DERIV
      ! Do a forward evaluation of all constituents.
      DO K = 1, ORDER
         DIV_LEFT = (LAST_KNOT - KNOTS(K))
         DIV_RIGHT = (LAST_KNOT - KNOTS(K+1))
         IF (DIV_LEFT .GT. 0) THEN
            IF (DIV_RIGHT .GT. 0) THEN
               VALUES(:,K) = &
                    ((XY(:) - KNOTS(K)) / DIV_LEFT) * VALUES(:,K) + &
                    ((LAST_KNOT - XY(:)) / DIV_RIGHT) * VALUES(:,K+1)
            ELSE
               VALUES(:,K) = ((XY(:) - KNOTS(K)) / DIV_LEFT) * VALUES(:,K)
            END IF
         ELSE
            IF (DIV_RIGHT .GT. 0) THEN
               VALUES(:,K) = ((LAST_KNOT - XY(:)) / DIV_RIGHT) * VALUES(:,K+1)
            END IF
         END IF
      END DO
      ! Sum the constituent functions at each knot (from the back).
      DO K = ORDER, 1, -1
         VALUES(:,K) = (VALUES(:,K) + VALUES(:,K+1))
      END DO
      ! Divide by the degree plus the integration coefficient.
      VALUES(:,1) = VALUES(:,1) / (ORDER-1+STEP)
      ! Rescale then integral by its width.
      VALUES(:,1) = VALUES(:,1) * (LAST_KNOT - KNOTS(1))
      ! Extend the previous two computations if more integrals need
      ! to be computed after this one.
      IF (STEP+DERIV .LT. 0) THEN
         VALUES(:,2:) = VALUES(:,2:) / (ORDER-1+STEP)
         DO K = 2, N
            VALUES(:,K) = VALUES(:,K) * (LAST_KNOT - KNOTS(K))
         END DO
      END IF
   END DO compute_integral

! ------------------ Performing differentiation --------------------
ELSE IF (DERIV .GT. 0) THEN
   ! Compute the derivative of the B-spline (if D > 0).
   compute_derivative : DO STEP = N-DERIV, ORDER
      ! Cycle over each knot, following the same structure with the
      ! derivative computing relation instead of the B-spline one.
      DO K = 1, N-STEP
         ! Assure that the divisor will not cause invalid computations.
         DIV_LEFT = (KNOTS(K+STEP-1) - KNOTS(K))
         DIV_RIGHT = (KNOTS(K+STEP) - KNOTS(K+1))
         ! Compute the derivative recurrence relation.
         IF (DIV_LEFT .GT. 0) THEN
            IF (DIV_RIGHT .GT. 0) THEN
               VALUES(:,K) =  (STEP-1) * (&
                    VALUES(:,K) / DIV_LEFT - VALUES(:,K+1) / DIV_RIGHT )
            ELSE
               VALUES(:,K) =  (STEP-1) * (VALUES(:,K) / DIV_LEFT)
            END IF
         ELSE
            IF (DIV_RIGHT .GT. 0) THEN
               VALUES(:,K) =  (STEP-1) * ( - VALUES(:,K+1) / DIV_RIGHT )
            END IF
         END IF
      END DO
   END DO compute_derivative
END IF integration_or_differentiation

! Assign the values into the "Y" output.
XY(:) = VALUES(:,1)
END SUBROUTINE EVAL_BSPLINE

END MODULE SPLINES
```