Cost Functions, Derivatives and Inverses
for Flux Flow Superconductors

Aleksandar Donev, Corey Musolf
July 2000

1 Module FF_Cost_Functions

This module is the only input concerning the elemental arc costs that the user needs to provide to the SSCNO library. The following is an encoding by Corey of a specific function suitable for certain superconducting materials, which we call FF for flux-flow. I have provided some FWEB macros and such to make life easier for Corey, and I recommend others to use a similar structure:

"WEAVE.f90" 1.0.0.1 ≡

MODULE FF_Cost_Functions
  USE Precision
  USE Error_Handling
  USE System_Monitors
  USE Network_Data_Structures
  USE Network_Data_Types
  USE FF_Cost_Parameters
  IMPLICIT NONE
  PUBLIC :: FFElementalCosts    // The elemental cost function(s)
PRIVATE
  INTEGER, SAVE, PUBLIC :: ff_elemental_costs_timer = -1    // An internal timer
CONTAINS
  { FFElementalCosts 1.1.0.3}
END MODULE FF_Cost_Functions
1.1 Cost Function and Related Derivatives Calculation

"WEAVE.f90" 1.1.0.1 ≡
@m ARRAY_SYNTAX 0
@m LOOP_SYNTAX 0
@m OPTIMIZED_LOOP_SYNTAX

This function will accept either the currents or the voltages through a set of arcs and return either the flows, voltages, resistances or costs (dissipated powers) for a set of arcs in the index range arcs_indices(1) : arcs_indices(2). I present three different versions for this relatively complicated cost function. The first one uses Fortran 90 array syntax and calls a scalar macro with array arguments (ARRAY_SYNTAX). The second one calls a scalar macro from inside a DO loop (LOOP_SYNTAX), which slows things down because of conditionals inside the loop. The third version (OPTIMIZED_LOOP_SYNTAX) is the most optimized one and calls the scalar macro inside a loop, but now puts the conditionals around (hoists invariant expressions out of loops, as certain compilers will do automatically with LOOP_SYNTAX).

For this particular function, here is how much time a typical run of SSCNO spent evaluating the cost functions:

ARRAY_SYNTAX 1.9 seconds
LOOP_SYNTAX 2.7 seconds
OPTIMIZED_LOOP_SYNTAX 1.6 seconds

So in this case the difference between optimized loop and array syntax is not that great, mostly because we do not use any array temporaries. But as you can see, one should avoid LOOP_SYNTAX:

Here is the body of the procedure FFElementalCosts:

⟨FFElementalCosts 1.1.0.3⟩ ≡
SUBROUTINE FFElementalCosts (cost_function, arguments_status, tolerance, arcs_indices,
    arcs_flows, arcs_voltages, arcs_resistances, arcs_costs)
    USE Precision
    USE Network_Data_Types
    _TYPE (Network_SC_Cost), INTENT (INOUT), OPTIONAL :: cost_function    // class in F2x
    CHARACTER (LEN = 4), INTENT (IN) :: arguments_status     // For example "F00D"
    REAL (KIND = r_wp), INTENT (IN) :: tolerance
    INTEGER (KIND = i_wp), DIMENSION (2), INTENT (IN) :: arcs_indices
    REAL (KIND = r_wp), DIMENSION (arcs_indices(1) :), INTENT (INOUT), OPTIONAL :: arcs_flows,
    arcs_voltages, arcs_resistances, arcs_costs

2
LOGICAL :: known_flow, known_voltage  // Indicators—I do not care about warm starts here
CHARACTER :: status  // A temporary
INTEGER (KIND = i_wp) :: arc  // A counter
REAL (KIND = r_wp) :: eps, temp  // The precision and a temporary for the flow
eps = MAX(EPSILON(1.0_r_wp), tolerance)  // To avoid divisions by zero
CALL StartTimer(ff_elemental_costs_timer)  // Start the internal timer

@#if ARRAY SYNTAX
  _FFElementalCost_array(arcs_flows, arcs_voltages, arcs_resistances, arcs_costs, Jc(ars), Xi(ars),
  J(ars), L(ars), L(ars))
  // Call the function macro with array arguments and array syntax
@#else
  LOOP SYNTAX
    DO arc = arcs_indices1, arcs_indices2  // A big DO loop
      _FFElementalCost_loop(arcs_flows, arcs_voltages, arcs_resistances, arcs_costs,
      Jc(ars), Xi(ars), J(ars), L(ars))
      // Call the macro with scalar arguments from inside a loop
    END DO
  @#end
  CALL StopTimer(ff_elemental_costs_timer)  // Stop the internal timer
END SUBROUTINE FFElementalCosts

This code is used in section 1.0.0.1.

1.2 Macros for Arc Cost Parameters

These macros access the array arcs_cost_parameters to return the critical current, width, resistance etc.:
1.3 The Flux Flow (FF) Superconductor Voltage-Current Characteristic

This is Corey’s encoding of the FF cost function. The following macro can either work with scalar or with array arguments. I have made suggestions of how to turn these into array statements without any explicit extra coding. This requires making any temporaries, in this case i, arrays as well. For this reason, try to avoid temporaries as much as possible (I did this here by avoiding the ABS function via a macro call), or simply recode these cost functions with explicit DO loops. Here for example arc_cost_parameters, is used as temporary storage for this calculation to store the flow value. I recommend that whoever is using this library learns Fortran 90 array syntax, so I use it here as an example:

"WEAVE.f90" 1.3.0.1 ≡
@m _FFElementalCost_array(flow, voltage, resistance, cost, Ic, Xi, R, C, I)

/* Flow calculation */
status = arguments_status_1:1
known_flow = (status ≡ 'P')
IF ((¬known_flow) ∧ status ≠ 'D') THEN    // Calculate the flow from the voltage
    flow = SIGN(1.0,wp, voltage) * flow(Abs(voltage), Ic, Xi, R, C)
END IF

/* Potential calculation */
status = arguments_status_2:2
known_voltage = (status ≡ 'P')
IF ((¬known_voltage) ∧ status ≠ 'D') THEN    // Calculate the voltage from the flow
    voltage = SIGN(1.0,wp, flow) * voltage(Abs(flow), Ic, Xi, R, C)
END IF

/* Resistance calculation */
status = arguments_status_3:3
IF (status ≠ 'P' ∧ status ≠ 'D') THEN    // Calculate the resistance from the flow or the voltage
    IF (arguments_status_1:1 ≡ 'P') THEN    // Flow was the input
        resistance = _resistance_Abs(flow), Ic, Xi, R, C)
    ELSE IF (arguments_status_2:2 ≡ 'P') THEN    // The voltage was the input
        resistance = _resistance_Abs(voltage), Ic, Xi, R, C)
    ELSE    // Should never occur!!!
        resistance = 0.0,wp
    END IF
END IF

/* Cost calculation */
status = arguments_status_4:4
IF (status ≠ 'P' ∧ status ≠ 'D') THEN    // Calculate the cost from the flow or the voltage
    IF (known_flow) THEN    // Flow is known
        cost = _cost_Abs(flow), Ic, Xi, R, C)
    ELSE IF (known_voltage) THEN    // Only the voltage is known
        I = _flow_Abs(voltage), Ic, Xi, R, C)
        cost = _cost_I, Ic, Xi, R, C)
END IF

4
ELSE  // Should never occur!!!
cost = 0.0
END IF
END IF

1.3.1 Auxillary Scalar/Array Macros

These are the actual macros that calculate \( I(V) \) and \( V(I) \). They are encoded so that they only work with positive arguments, i.e., \texttt{ABS} or \texttt{SIGN} should be applied to them if needed when calling the macro. Please note that the FWEB symbol \$\%\$ is needed here as it makes the inlined expressions that can be embedded inside other expressions (so one can say for example \((\text{flow}(\text{voltage}, Ic, Xi, R, C))^2\) and still get legal Fortran code:

"\texttt{WEAVE.f90} 1.3.1.1 \equiv
\begin{align*}
\texttt{m} \_\text{flow}(V, Ic, Xi, R, C) &= (2 \times V) \times (C - Ic) \\
\texttt{m} \_\text{voltage}(I, Ic, Xi, R, C) &= 0.5 \times \text{R} \times (\text{sqrt}((I - Ic)^2 + Xi^2) + I - C) \\
\texttt{m} \_\text{resistance}(I, Ic, Xi, R, C) &= 0.5 \times \text{R} \times (1 + (I - Ic) \times \text{sqrt}((I - Ic)^2 + Xi^2)) \\
\texttt{m} \_\text{cost}(I, Ic, Xi, R, C) &= 0.25 \times \text{R} \times ((2V)^2 + (I - 2Ic) \times C + (I - Ic) \times \text{sqrt}((I - Ic)^2 + Xi^2) + X_i^2 \times \text{log}(I - Ic + \text{sqrt}((I - Ic)^2 + Xi^2)) - \text{log}(C - Ic))
\end{align*}

1.4 Loop-Based Elemental Costs

In order to use array syntax, as the above example demonstrates, one must make sure all used temporaries are arrays of conformable shape and that all user functions involved are \texttt{ELEMENTAL}. This may sometimes not be OK, for example, if there are too many temporaries involved to waste space on them. So the following will often work just as good if not better from the array-based version. Notice that it is the same as above only with an explicitly added \texttt{DO} loop. There are some tricky things in FWEB with macro arguments, so here I do not pass \( Ic, Xi, R, C \) as an argument to the macro:

"\texttt{WEAVE.f90} 1.4.0.1 \equiv
\begin{align*}
\texttt{m} \_\text{FFEElementalCost}\_\text{loop}(\text{flow}, \text{voltage}, \text{resistance}, \text{cost}) \\
& \hspace{1cm} /* Flow calculation: */ \\
& \hspace{1cm} \text{status} = \text{arguments.status}_{1:1} \\
& \hspace{1cm} \text{known\_flow} = (\text{status} \equiv \text{'F'}) \\
& \hspace{1cm} \text{if} ((\sim \text{known\_flow} \land \text{status} \neq \text{'D'})) \text{ then} \hspace{1cm} // \text{Calculate the flow from the voltage} \\
& \hspace{2cm} \text{do arc} = \text{arcs\_indices}_{1}, \text{arcs\_indices}_{2} \\
& \hspace{2cm} \text{flow\_arc} = \text{SIGN}(1.0, \text{voltage}_{\text{arc}}) \times \text{flow}(\text{ABS}(\text{voltage}_{\text{arc}}), Jc(\text{arc}), Xi(\text{arc}), \rho(\text{arc}), \rho(\text{arc})) \\
& \hspace{1cm} \text{end do}
\end{align*}
known_flow = \mathcal{T}
END IF

/* Potential calculation */
status = arguments.status_{1:2}
known_voltage = (status \equiv \mathcal{F})
IF \((\neg known\_voltage) \land status \neq \mathcal{D}) \) THEN // Calculate the voltage from the flow
  DO arc = arcs_indices_1, arcs_indices_2
    voltage_{arc} = \text{SIGN}(1.0,_{wp, flow_{arc}}) \cdot \text{voltage}(\text{ABS}(flow_{arc}), \mathcal{J}(arc), \mathcal{X}(arc), \mathcal{R}(arc), \mathcal{C}(arc))
  END DO
known\_voltage = \mathcal{T}
END IF

/* Resistance calculation */
status = arguments.status_{3:4}
IF \((status \neq \mathcal{F} \land status \neq \mathcal{D}) \) THEN // Calculate the resistance from the flow or the voltage
  IF \(\text{arguments.status}_{1:1} \equiv \mathcal{F}) \) THEN // Flow is known
    DO arc = arcs_indices_1, arcs_indices_2
      resistance_{arc} = \text{resistance}_{L}(\text{ABS}(flow_{arc}), \mathcal{J}(arc), \mathcal{X}(arc), \mathcal{R}(arc), \mathcal{C}(arc))
    END DO
  ELSE IF \(\text{arguments.status}_{2:2} \equiv \mathcal{F}) \) THEN // Only the voltage is known
    DO arc = arcs_indices_1, arcs_indices_2
      resistance_{arc} = \text{resistance}_{V}(\text{ABS}(voltage_{arc}), \mathcal{J}(arc), \mathcal{X}(arc), \mathcal{R}(arc), \mathcal{C}(arc))
    END DO
  ELSE // Should never occur!!!
    resistance = 0.0,_{wp}
  END IF
END IF

/* Cost calculation */
status = arguments.status_{4:4}
IF \((status \neq \mathcal{F} \land status \neq \mathcal{D}) \) THEN // Calculate the cost from the flow or the voltage
  IF \(\text{known\_flow}) \) THEN // Flow is known
    DO arc = arcs_indices_1, arcs_indices_2
      cost_{arc} = \text{cost}(\text{ABS}(flow_{arc}), \mathcal{J}(arc), \mathcal{X}(arc), \mathcal{R}(arc), \mathcal{C}(arc))
    END DO
  ELSE IF \(\text{known\_voltage}) \) THEN // Only the voltage is known
    DO arc = arcs_indices_1, arcs_indices_2
      temp = flow(\text{ABS}(voltage_{arc}), \mathcal{J}(arc), \mathcal{X}(arc), \mathcal{R}(arc), \mathcal{C}(arc)) // A scalar temporary
      cost_{arc} = \text{cost}(temp, \mathcal{J}(arc), \mathcal{X}(arc), \mathcal{R}(arc), \mathcal{C}(arc))
    END DO
  ELSE // Should never occur!!!
    cost = 0.0,_{wp}
  END IF
END IF
@m CASE_TYPE TYPE
@m _TYPE TYPE
@m _NULL > NULL()
@m PRIVATE PRIVATE
@m _SIZE(array, _kind,...)
    $IFSEL (0, 0, INT(SIZE(array), KIND=_kind), INT(SIZE(array,#), KIND=_kind))
@m _MAXLOC(array, _kind,...)
    $IFSEL (0, 0, INT(MAXLOC(array), KIND=_kind), INT(MAXLOC(array,#), KIND=_kind))
@m _MINLOC(array, _kind,...)
    $IFSEL (0, 0, INT(MINLOC(array), KIND=_kind), INT(MINLOC(array,#), KIND=_kind))
@m _LBOUND(array, _kind,...) $IFSEL (0, 0, INT(LBOUND(array, DIM=1), KIND=_kind),
                              INT(LBOUND(array, #), KIND=_kind))
@m _UBOUND(array, _kind,...) $IFSEL (0, 0, INT(UBOUND(array, DIM=1), KIND=_kind),
                               INT(UBOUND(array, #), KIND=_kind))
@m _GenericInterface(generic_name,...)
    INTERFACE generic_name MODULE PROCEDURE #.
    END Declare INTERFACE generic_name
@m _Declare i_word(...) INTEGER :: #.
@m _Declare i_wp(...) INTEGER (KIND = i_wp) :: #.
@m _Declare r_wp(...) REAL (KIND = r_wp) :: #.
@m _Declare r_sp(...) REAL (KIND = r_sp) :: #.
@m _Declare r_dp(...) REAL (KIND = r_dp) :: #.
@m _FullExtent(_rank) :&DO (DIM, 2, _rank) { ; }&DO
@m _VarSequence(_variable, _start, _end)
    _variable##_start :&DO (DIM, _start + 1, _end) { , _variable##DIM }
@m _NestedLoopStart(_variable, _array, _rank, _kind)
    _DO (DIM, _rank, 1, -1) { _DO _variable##DIM = _LBOUND(_array, _kind, DIM = DIM),
                           _UBOUND(_array, _kind, DIM = DIM) }&DO
@m _NestedLoopEnd(_rank) _DO (DIM, 1, _rank) { END _DO }
@m _Dummy(...)
@m _DisplayArray(message, array)
    IF (SIZE(array) <= 20) THEN
        WRITE(message, print_unit, "(A)" message
        WRITE(message, print_unit, "(205,2)" array
    END IF