1. Implementation of symbolic expressions
   1. symL is an attribute of L that needs to be input by the user.
      1. There are checks to make sure symL is indeed the symbolic version of L.
      2. Principle: Functionalities of Julia Functions >= Functionalities of SymPy. If p\_k has no SymPy representation, the only consequence should be that outputs by functions that take L as arugment has no symbolic expression. E.g., we allow L.pFunctions and L.symL.pFunctions to differ.
   2. If a function can produce both symbolic and non-symbolic outputs, the output type is controlled by get\_output(…; symbolic = true/false). If a function produces only symbolic outputs, it is usually called get\_symOutput().
2. Unit tests
   1. Tested the algorithm to generate adjoint for constant and variable (polynomial) coefficients
   2. Tolerance for checking float equality:
      1. 1e-05 \* mean(x, y).
      2. 1e-05 \* mean(norm(A,2), norm(B,2)) where norm() is p-norm for matrices with p=2
3. Capstone report
   1. Introduction:
      1. Project goals
      2. Where am I in the timeline. What does what I’ve done so far mean to the project.
         1. E.g., construct valid adjoint is the first step in the algorithm.
   2. Preliminary background material as relevant to the algorithm to construct adjoint
      1. Coddington & Levinson chapter 11
      2. Definitions, concepts, key results, etc
   3. Outline of algorithm (theoretical)
      1. Key steps and justifications
   4. Brief discussion on implementation
      1. Platform: Julia
      2. Main structs/objects/types: LinearDifferentialOperator, VectorBoundaryForm
      3. Notable features: Symbolic expression to keep track of functions
   5. Next steps:
      1. Transform pairs in Smith-Fokas