PARAMERERIZED TOPOLOGY — CIRCUITS

FREE VARIABLES AND CONDITIONAL OPERATORS IN COMPUTER PROGRAMS

- **One of the most important characteristics of computer programs is that they ordinarily contain inputs (free variables) and conditional operations.**
- **Genetic algorithms and other techniques of genetic and evolutionary computation are typically used to search for an optimal (or near-optimal) solution to a particular single instance of a problem.**
    - EXAMPLE: Find the numerical values for the real and imaginary parts of the two complex roots of a particular quadratic equation, such as $3x^2 + 4x + 5$, a separate run is required to solve each different instance of the problem (e.g., $10x^2 + 2x + 1$).

FREE VARIABLES AND CONDITIONAL OPERATORS

- **Free variables enable a single program to produce different outputs based on the particular values of its free variables.**

- **Conditional operations enable a single program to execute alternative sequences of steps based on the particular values of the free variables.**

- **Conditional operations and free variables together potentially enable a single program to solve all instances of a problem (instead of just one instance of the problem). Thus, a computer program containing free variables and conditional operations has the ability to solve all quadratic equations — $ax^2 + bx + c$**

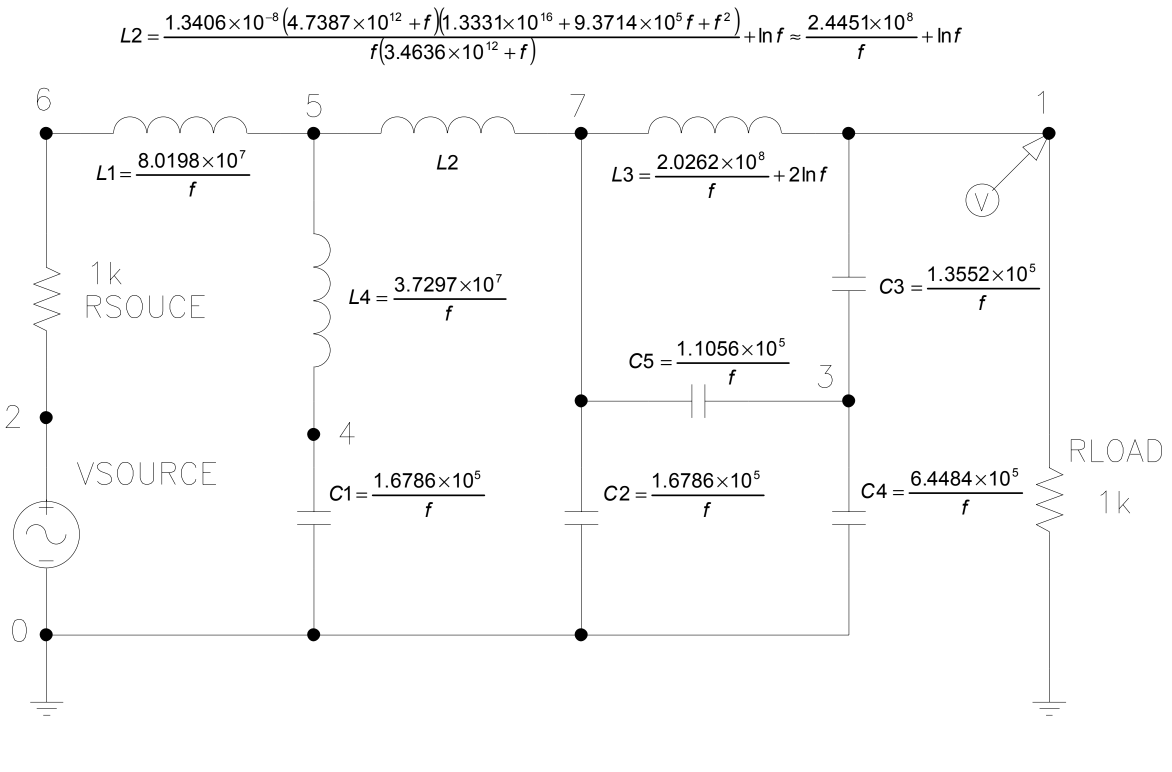
PARAMERERIZED TOPOLOGIES

- **You may really want the *general* solution to $ax^2 + bx + c$**
- **And, you may also want to automatically create both the *topology* of an entity (e.g., a circuit) along with the *equations specifying the values of the components* in the entity.**

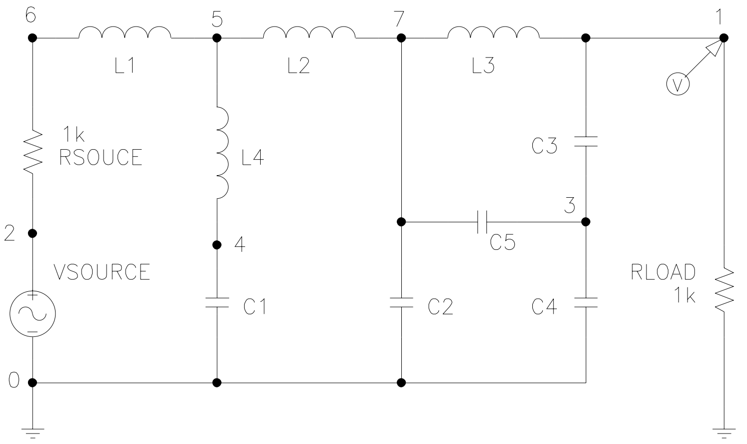
VARIABLE CUTOFF LOWPASS FILTER

"GENERALIZED" PARAMETERIZED LOWPASS FILTER

• You may want a "general solution" to the lowpass filter problem. That is, you may want a filter whose passband ends at frequencies such as *f* = 1,000, 1,780, 3,160, 5,620, 10,000, 17,800, 31,600, 56,200, 100,000 Hz


$L2 = \frac{1.3406\times10^{-8}(4.7387\times10^{12}+f)(1.3331\times10^{16}+9.3714\times10^{5}f+f^{2})}{f(3.4636\times10^{12}+f)}+\ln f \approx \frac{2.4451\times10^{8}}{f}+\ln f$
6
5
7
1
$L1=\frac{8.0198\times10^{7}}{f}$
L2
$L3=\frac{2.0262\times10^{8}}{f}+2\ln f$
V
1k
RSOUCE
$L4=\frac{3.7297\times10^{7}}{f}$
$C3=\frac{1.3552\times10^{5}}{f}$
$C5=\frac{1.1056\times10^{5}}{f}$
3
2
4
VSOURCE
RLOAD
$C1=\frac{1.6786\times10^{5}}{f}$
$C2=\frac{1.6786\times10^{5}}{f}$
$C4=\frac{6.4484\times10^{5}}{f}$
1k
0

VARIABLE CUTOFF LOWPASS FILTER



BBB2291

$$L1 = \frac{8.0198\times10^{7}}{f}$$

$$L2 = \frac{1.3406\times10^{-8}\left(4.7387\times10^{12}+f\right)\left(1.3331\times10^{16}+9.3714\times10^{5}f+f^{2}\right)}{f\left(3.4636\times10^{12}+f\right)}+\ln f \approx \frac{2.4451\times10^{8}}{f}+\ln f$$

$$L3 = \frac{2.0262\times10^{8}}{f}+2\ln f$$

$$L4 = \frac{3.7297\times10^{7}}{f}$$

$$C1 = \frac{1.6786\times10^{5}}{f}$$

$$C2 = \frac{1.6786\times10^{5}}{f}$$

$$C3 = \frac{1.3552\times10^{5}}{f}$$

$$C4 = \frac{6.4484\times10^{5}}{f}$$

$$C5 = \frac{1.1056\times10^{5}}{f}$$

VALUE-SETTING SUBTREES (ARITHMETIC-PERFORMING SUBTREES)

• The numerical value for each capacitor or inductor is established by a value-setting subtree (arithmetic-performing subtrees) containing perturbable numerical values, arithmetic operations, and the free variable *f* representing the frequency of the end of the desired filter's passband.

• The terminal set, $\mathsf{T}_{\mathsf{aps}}$, for the value-setting subtrees is

$\mathsf{T}_{\mathsf{aps}} = \{\Re, \mathsf{F}\}$,

where $\Re$ denotes perturbable numerical values.

VALUE-SETTING SUBTREE — CONTINUED

• In generation 0, each perturbable numerical value is set, individually and separately, to a random value in the range from 10^{-5} and 10^{5}.

• In later generations, this perturbable numerical value is perturbed by a Gaussian probability distribution.

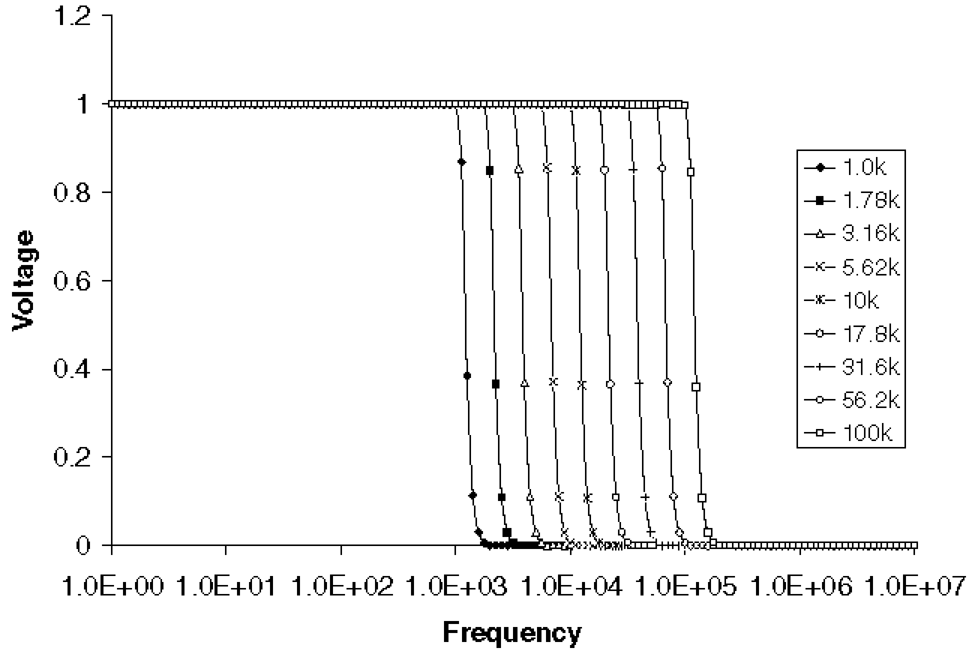
• A constrained syntactic structure maintains a function and terminal set for the value-setting subtrees (arithmetic-performing subtrees) and the (different) function and terminal set for all other parts of the program tree.

• The function set, F_{aps}, for the arithmetic-performing subtrees is

F_{aps} = {ADD_NUMERIC, SUB_NUMERIC, MUL_NUMERIC, DIV_NUMERIC, REXP, RLOG}

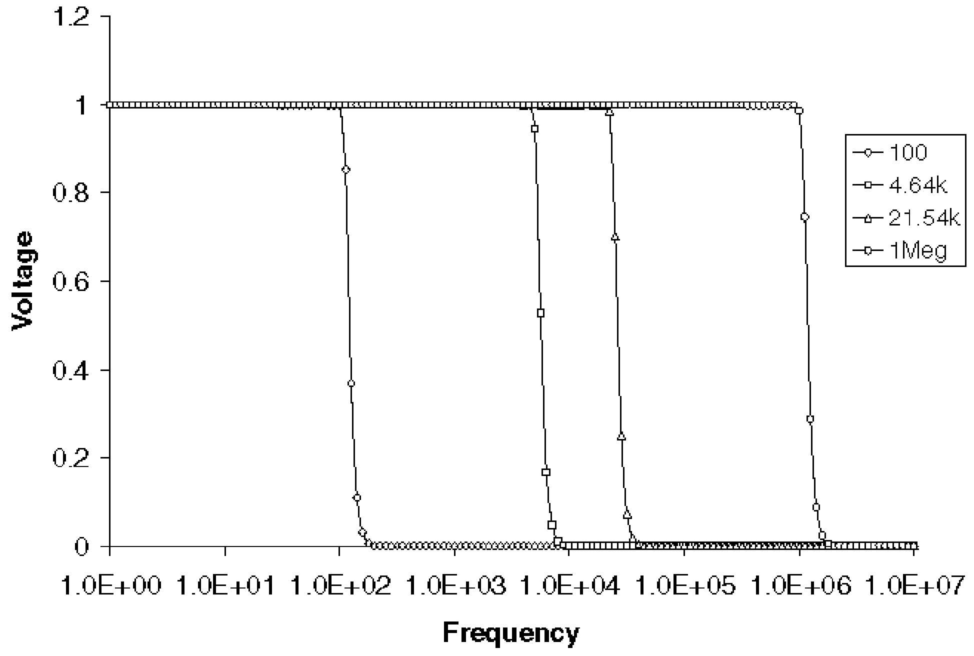
VARIABLE CUTOFF LOWPASS FILTER

BEHAVIOR IN THE FREQUENCY DOMAIN FROM GENERATION 78 FOR EACH OF NINE VALUES OF THE FREE VARIABLE *F*



VARIABLE CUTOFF LOWPASS FILTER — CONTINUED

FOUR OUT-OF-SAMPLE VALUES OF FREQUENCY *F*



LOWPASS - HIGHPASS FILTER USING CONDITIONAL DEVELOPMENTAL OPERATOR

- **The numerical value for each capacitor or inductor is established by a value-setting subtree containing a conditional developmental operator, perturbable numerical values, arithmetic operations, and the free variables F1 representing the 1 of the passband and F2 representing the boundary of the stopband.**
- **Function set, F_{aps}, for arithmetic-performing subtrees is**

F_{aps} = {IFGTZ_DEVELOPMENTAL, +, -, *, %, REXP, RLOG}.

LOWPASS - HIGHPASS FILTER USING CONDITIONAL DEVELOPMENTAL OPERATOR — CONTINUED

- **If the first (numerical) argument of the three-argument `IFGTZ_DEVELOPMENTAL` operator evaluates its second (developmental) argument (but not its third argument) if its first (numerical argument) is greater than zero; otherwise, this operator evaluates its third (developmental) argument (but not its second argument).**
- **Function set, F_{rpb}, for rest of result-producing branch**

F_{rpb} = {`L`, `C`, `SERIES`, `PARALLEL0`, `FLIP`, `NOP`, `PAIR_CONNECT_0`, `PAIR_CONNECT_1`, `THREE_GROUND_0`, `THREE_GROUND_1`, `ADF0`, `ADF1`, `ADF2`, `ADF3`, `ADF4`}

LOWPASS - HIGHPASS FILTER USING CONDITIONAL DEVELOPMENTAL OPERATOR — CONTINUED

- **Terminal set, T_{aps}, for arithmetic-performing subtrees**

$T_{aps} = \{\Re, \texttt{F1}, \texttt{F2}\}$,

- **A constrained syntactic structure maintains a function and terminal set for the arithmetic-performing subtrees and a different function and terminal set for rest of program tree.**
- **Terminal set, T_{rpb}, for rest of result-producing branch**

$T_{rpb} = \{\text{END}, \text{SAFE\_CUT}\}$.

- **The terminal set, T_{adf}, for each automatically defined function (other than their arithmetic-performing subtrees) is**

$T_{adf} = \{\text{END}, \text{SAFE\_CUT}\}$

LOWPASS - HIGHPASS FILTER USING CONDITIONAL DEVELOPMENTAL OPERATOR

• The two free variables, `F1` and `F2`, may assume the values (10,000, 20,000) or (20,000, 10,000). For each of these two combinations, SPICE is instructed to perform an AC small signal analysis and report the circuit's behavior over five decades with each decade being divided into 20 parts.

• Fitness is the sum, over all two combinations of values of the free variables and over all 101 fitness cases associated with each combination, of the absolute weighted deviations

$$F(t) = \sum_{k=1}^{2} \sum_{i=0}^{100} (W(d(f_i), f_i) d(f_i))$$

ONE OF THE RESULT-PRODUCING BRANCHES OF A PACE-SETTING INDIVIDUAL FROM GENERATION 1

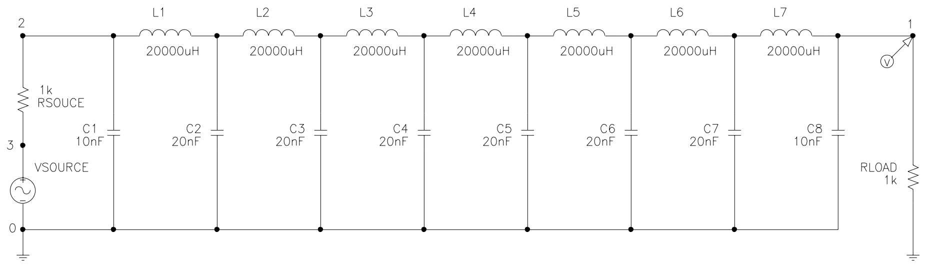
```
(FLIP
  (IFGTZ_DEVELOPMENTAL
    (* F1 (- F1 F2))
    (IFGTZ_DEVELOPMENTAL
      (* F2 F2)
      (L F2 END)
      (FLIP END))
    (C (DIVIDE_NUMERIC F2 F2) (PARALLEL0
END END END END))))
```

- **The first argument of the first `IFGTZ_DEVELOPMENTAL` operator in this branch computes `F1 - F2` and multiplies this difference by a quantity (`F1`) that is always positive. If the result is positive (i.e., a highpass filter is desired), the second `IFGTZ_DEVELOPMENTAL` operator is executed.**

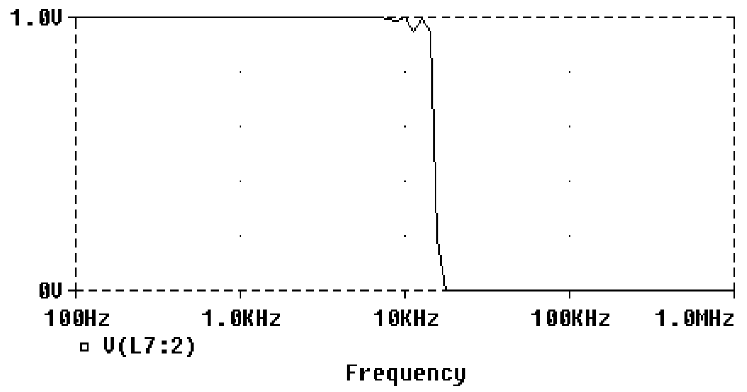
ONE OF THE RESULT-PRODUCING BRANCHES OF A PACE-SETTING INDIVIDUAL FROM GENERATION 1 — CONTINUED

- **Since the first argument of this second `IFGTZ_DEVELOPMENTAL` operator squares `F2`, the component-creating `L` function is unconditionally executed.**
- **However, if `F1 - F2` is non-positive, then the component-creating `C` function is unconditionally executed.**

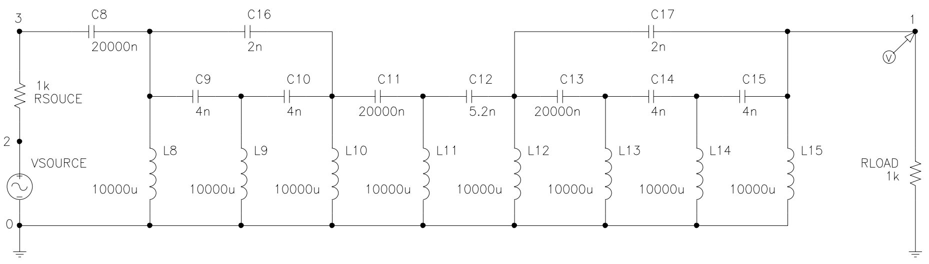
BEST-OF-RUN INDIVIDUAL FROM GENERATION 47 FOR CASE WHEN INPUTS CALL FOR LOWPASS FILTER — I. E., THE TWO FREE VARIABLES, F1 AND F2, ARE (10,000, 20,000)



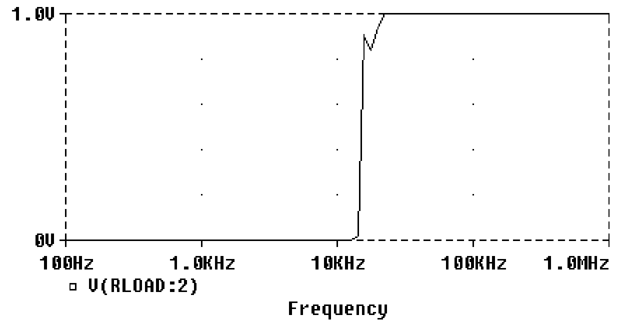
FREQUENCY DOMAIN BEHAVIOR OF BEST-OF-RUN INDIVIDUAL WHEN INPUTS CALL FOR LOWPASS FILTER



BEST-OF-RUN INDIVIDUAL FROM GENERATION 47 FOR CASE WHEN INPUTS CALL FOR HIGHPASS FILTER — I.E. THE TWO FREE VARIABLES, F1 AND F2, ARE (20,000, 10,000)



FREQUENCY DOMAIN BEHAVIOR OF BEST-OF-RUN INDIVIDUAL WHEN INPUTS CALL FOR HIGHPASS FILTER



VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS

- **We are seeking a single parameterized outcome that yields two functionally and topologically different circuits (lowpass versus highpass filters) depending on the particular values of two free variables and the boundaries of the passband and the stopband may vary over the range between 1,000 Hz and 100,000 Hz**

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

• **Fitness is the sum, over all 18 combinations of values of the free variables, F1 and F2, and over all 101 fitness cases associated with each combination, of the absolute weighted deviation between the actual value of the voltage that is produced by the circuit at the probe point VOUT and the target value for voltage (0 or 1 volt). Specifically,**

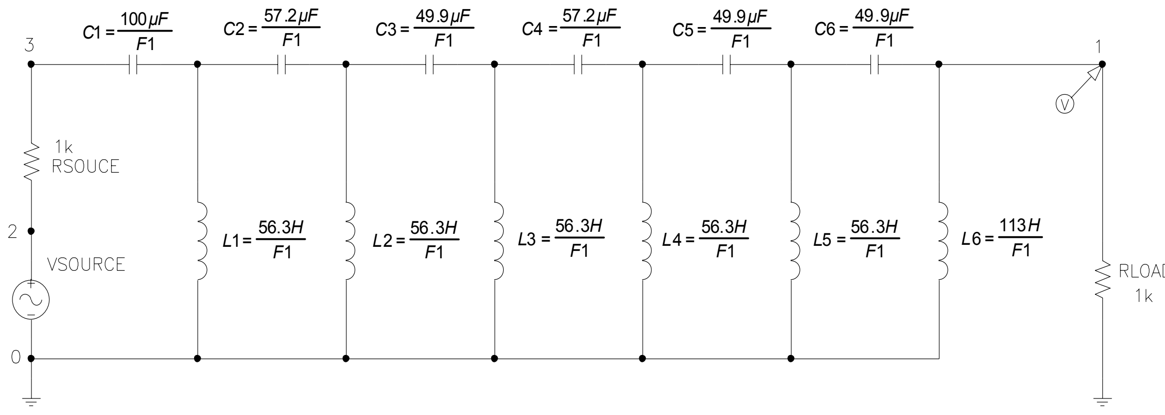
$$F(t) = \sum_{k=1}^{2} \sum_{j=1}^{9} \sum_{i=0}^{100} (W(d(f_i), f_i) d(f_i))$$

where *fi* is the frequency of fitness case *i*; *d(x)* is the absolute value of the difference between the target and observed values at frequency *x*; and *W*(*y*,*x*) is the weighting for difference *y* at frequency *x*. A smaller value of fitness is better.

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

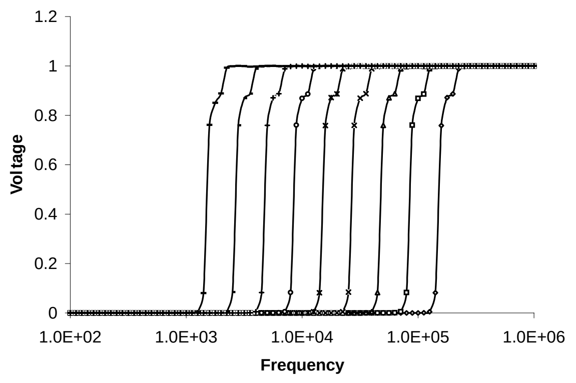
VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

- **Best-of-run circuit from generation 93 when inputs call for a highpass filter (i.e., `F1` > `F2`). Genetic programming produced this circuit's overall topology as well as the 12 mathematical expressions (each containing the free variables `F1` and `F2`) for specifying the component values for the circuit's six inductors and six capacitors.**



VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

FREQUENCY DOMAIN BEHAVIOR OF BEST-OF-RUN CIRCUIT WHEN INPUTS CALL FOR A HIGHPASS FILTER

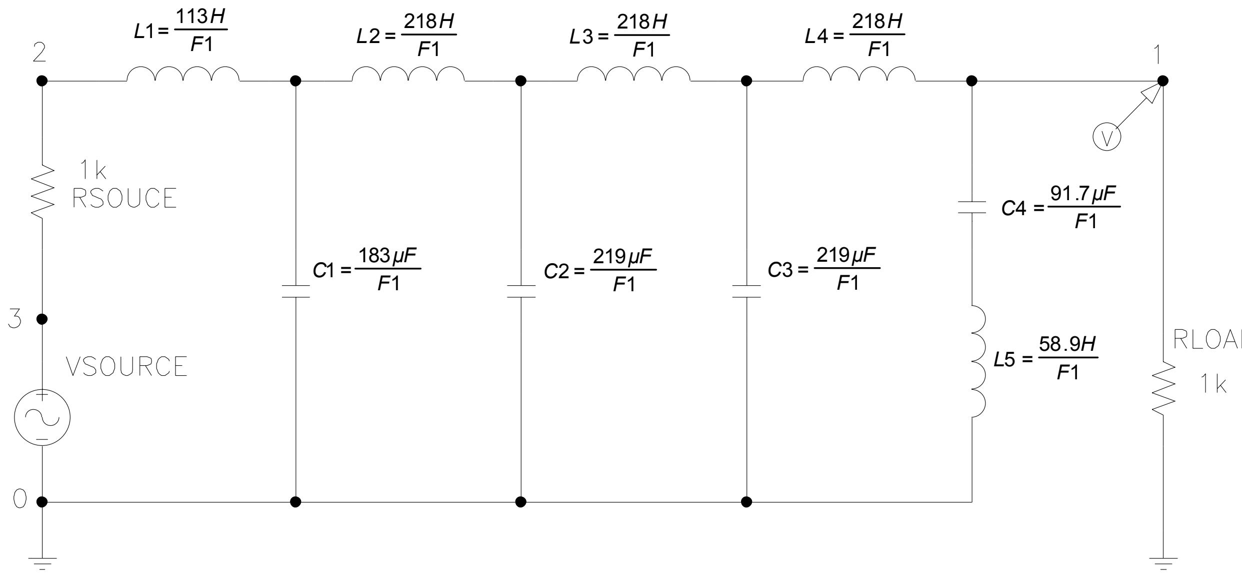


VALUES FOR HIGHPASS FILTER

Start of transition band	C1	C2 and C4	C3, C5, and C6	L1, L2, L3, L4, and L5	L6
1,000	100.0	57.2	49.9	56,300	100,000
1,778	56.2	32.1	28.1	31,700	63,300
3,162	31.6	18.1	15.8	17,800	35,600
5,623	17.8	10.2	8.88	10,000	20,000
10,000	10.0	5.72	4.99	5,630	11,300
17,782	5.62	3.21	2.81	3,170	6,330
31,622	3.16	1.81	1.58	1,780	3,560
56,234	1.78	1.02	0.888	1,000	2,000
100,000	1.0	0.572	0.499	563	1,130

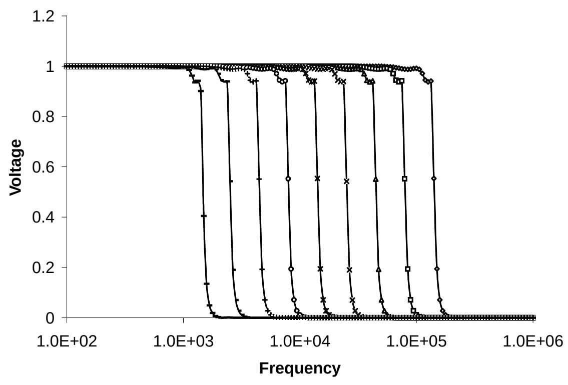
VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

- **Best-of-run circuit from generation 93 when inputs call for a lowpass filter. Genetic programming produced this circuit's overall topology as well as the nine mathematical expressions (each containing the free variables `F1` and `F2`) for specifying the component values for the circuit's five inductors and four capacitors.**



VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

FREQUENCY DOMAIN BEHAVIOR OF BEST-OF-RUN CIRCUIT WHEN INPUTS CALL FOR A LOWPASS FILTER.

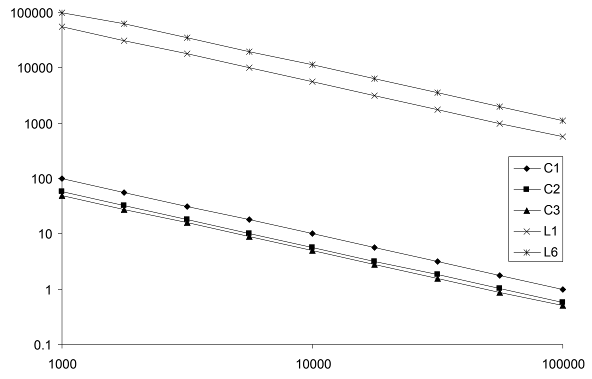


VALUES FOR A LOWPASS FILTER

Start of transition band	L1	L2, L3, and L4	L5	C1	C2 and C3	C4
1,000	100,000	200,000	58,900	183	219	91.7
1,778	63,400	123,000	33,100	103	123	51.6
3,162	35,600	69,000	18,600	58	69.2	29.0
5,623	20,000	38,800	10,500	32.6	38.9	16.3
10,000	11,300	21,800	5,890	18.3	21.9	9.17
17,782	6,340	12,300	3,310	10.3	12.3	5.16
31,622	3,560	6,900	1,860	5.8	6.92	2.90
56,234	2,000	3,880	1,050	3.26	3.89	1.63
100,000	1,130	2,180	589	1.83	2.19	0.917

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

COMPARISON OF COMPONENT VALUES FOR HIGHPASS FILTER



$$C1 = \frac{100{,}000nF}{F1} = \frac{100\mu F}{F1}$$

$$C2, C4 = \frac{57{,}200nF}{F1} = \frac{57.2\mu F}{F1}$$

$$C3, C5, C6 = \frac{49{,}900nF}{F1} = \frac{49.9\mu F}{F1}$$

$$L1, L2, L3, L4, L5 = \frac{56{,}300{,}000\mu H}{F1} = \frac{56.3H}{F1}$$

$$L6 = \frac{113{,}000{,}000\mu H}{F1} = \frac{113H}{F1}$$

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

Six similar plots for the parameterized lowpass filter reveal that the component values for the six distinct components for all nine frequencies also lie along straight lines. The following formulae (all of which have the frequency `F1` in the denominator) represent the relationships between the component values and frequencies in the parameterized lowpass filter.

$$L1 = \frac{113{,}000{,}000 \mu H}{F1} = \frac{113 H}{F1}$$

$$L2, L3, L4 = \frac{218{,}000{,}000 \mu H}{F1} = \frac{218 H}{F1}$$

$$L5 = \frac{58{,}900{,}000 \mu H}{F1} = \frac{58.9 H}{F1}$$

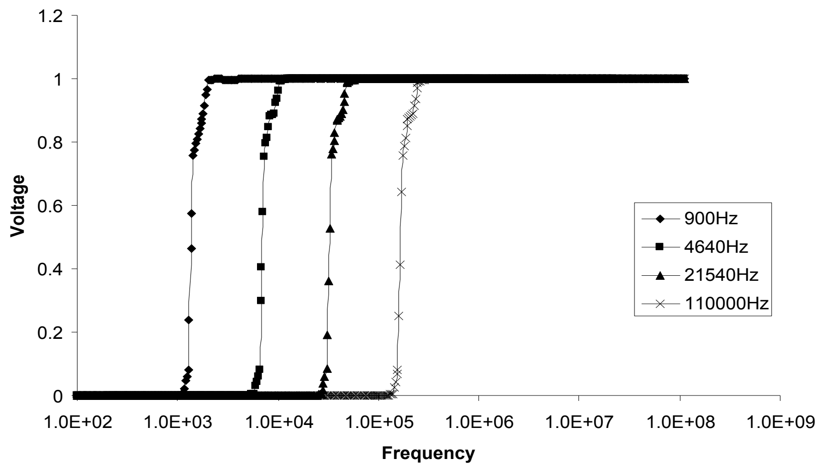
$$C1 = \frac{183{,}000 nF}{F1} = \frac{183 \mu F}{F1}$$

$$C2, C3 = \frac{219{,}000 nF}{F1} = \frac{219 \mu F}{F1}$$

$$C4 = \frac{91{,}700 nF}{F1} = \frac{91.7 \mu F}{F1}$$

VARIABLE-CUTOFF LOWPASS/HIGHPASS FILTER CIRCUIT USING BOTH FREE VARIABLES AND CONDITIONAL DEVELOPMENTAL OPERATORS — CONTINUED

FREQUENCY DOMAIN BEHAVIOR OF BEST-OF-RUN CIRCUIT FOR FOUR OUT-OF-SAMPLE VALUES OF FREQUENCY WHEN INPUTS CALL FOR A HIGHPASS FILTER



- **Similar results for lowpass filter**

FIVE NEW TECHNIQUES FOR 5 NEW TECHNQIUES FOR AUTOMATIC CIRCUIT SYNTHESIS THAT WE USE ON THE NEXT 2 PROBLEMS INVOLVING PARAMETERIZED TOPOLOGIES FOR CIRCUITS

- **New `NODE` Function for Connecting Distant Points**
- **Symmetry-Breaking Procedure using Geometric Coordinates**
- **Depth-First Evaluation**
- **New `TWO_LEAD` Function for Inserting Two-Leaded Components**
- **New `Q` Transistor-Creating Function**

NEW `NODE` FUNCTION FOR CONNECTING DISTANT POINTS

- **Most practical circuits require connections that cannot be achieved in a totally planar arrangement.**
- **Previous work concerning the automatic synthesis of electrical circuits employed the `VIA` and `PAIR_CONNECT` functions to connect distant points in a developing circuit.**
- **Although runs of genetic programming using these earlier functions were successful in automatically creating circuits, we have long believed that these functions were inefficient.**

NEW NODE FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **The premise behind the crossover operation is that individuals with relatively high fitness are likely to contain some substructures which, when recombined, may create offspring with even higher fitness.**

NEW NODE FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **In the genetic algorithm operating on fixed-length strings, the conventional crossover operation recombines a contiguous sub-string of characters from one parent's chromosome with a contiguous sub-string of characters from the second parent's chromosome.**
    - Over many generations, combinations of characters that are located close to each other on the chromosome string tend to be differentially preserved in the face of the inherently disruptive crossover operation.
    - Thus, shorter contiguous substrings are preserved to a greater degree than longer ones. That is, local substructures are preserved in the face of the (often) disruptive nature of the crossover operation.

NEW NODE FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **In genetic programming, the conventional crossover operation recombines a subtree from one parent's program tree with the second parent's program tree.**
  - Over many generations, functions and terminals that are nearby in the tree tend to be differentially preserved.
  - In particular, smaller subtrees are preserved to a greater degree than larger ones.

NEW NODE FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **The `VIA` and `PAIR_CONNECT` functions have the disadvantage that when a subtree of one circuit-constructing program tree is swapped with a subtree of another circuit-constructing program tree, the connectivity of points within both the crossover fragment and the remainder is, almost always, dramatically altered in an extremely disruptive way.**
  - That is, crossover usually significantly disrupts the nature of the preexisting connections formed by the `VIA` and `PAIR_CONNECT` functions within a local area of the developing circuit.
  - These local structures may have contributed to the individual's comparatively high fitness and to the

individual's being selected to participate in the genetic operation in the first place.

- To the extent that crossover dramatically alters the characteristics of the swapped genetic material, it acquires the characteristics of the mutation operation and its effectiveness in solving the problem is consequently reduced to the lesser level delivered by the random mutation operation. • It would be desirable that local substructures be preserved during the evolutionary process.

NEW `NODE` FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **This perceived problem concerning the `VIA` and `PAIR_CONNECT` functions is addressed by introducing a new two-argument function (called "`NODE`").**
- **The two-argument `NODE` function provides a way to connect distant points in a developing circuit to each other.**
- **The `NODE` function replaces one modifiable wire (or modifiable component) with a series composition consisting of one modifiable wire, a temporary port that can potentially be connected to other point(s) in the circuit, and a second modifiable wire.**

NEW NODE FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **Prior to the execution of the developmental process, the circuit-constructing program tree is examined to identify the set of NODE functions that are not ancestors of any NODE function higher in the program tree.**
- **The NODE functions in this set are called "top-most NODE functions."**
- **For each top-most NODE function, all the temporary ports (if any) associated with the NODE functions that are ancestors of a particular top-most NODE function are connected to the port of the top-most NODE function (subsequent to the execution of the developmental process).**
- **Any temporary ports that do not become connected by this process are removed.**

- **The `NODE` function takes two arguments, one for each of the construction-continuing subtrees associated with the two modifiable wires.**

NEW NODE FUNCTION FOR CONNECTING DISTANT POINTS — CONTINUED

- **When the crossover operation moves any subtree within the subtree rooted by a particular top-most NODE function into another circuit-constructing program tree, the connectivity of the ports contained in the moved subtree remains intact.**
- **Moreover, the connectivity of the ports in the unmoved remainder of the original program tree also remains intact.**
- **We believe that this new approach encourages the preservation of building blocks and thereby increases the efficiency of the overall search.**

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES

- **In previous work, a unique consecutive part number (internally assigned during the developmental process when a component is first inserted into the developing circuit) is used to break symmetries involving the behavior of certain circuit-constructing functions (such as the transistor-creating functions and parallel division functions)**
  - When a transistor is inserted into a developing circuit, the correspondence between the three leads of the transistor (its base, collector, and emitter) and the three points in the developing circuit to which these leads are connected is defined by referring to the unique consecutive part numbers of neighboring components.

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES — CONTINUED

- **The overall circuit-constructing program tree changes throughout the run.**
    - Thus, when neighboring parts change, the behavior of the transistor-creating and parallel division functions is dramatically altered in a rather abrupt way.
    - These abrupt changes in behavior may disrupt the local structures that contribute to the individual's relatively high fitness and to the individual's selection to participate in genetic operations in the first place.
    - To the extent that the genetic operations do not preserve locality, they lose some of their effectiveness (and, in the extreme case, come to resemble blind random search).

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES — CONTINUED

- **This perceived problem is addressed by breaking symmetry using geometric coordinates that are assigned to each node in the developing circuit (instead of using the unique consecutive parts number).**
- **We start by assigning geometric coordinates to the nodes to which each modifiable wire in the embryo is connected.**
    - For example, if there is one modifiable wire in the embryo, the positive end is defined to be at coordinate location (0, 0), and its negative end is defined to be at coordinate location (1, 0).

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES — CONTINUED

- The coordinate locations of new nodes that are created by functions that entail symmetry breaking (and those created by all other functions) are defined in terms of the coordinate locations of the existing nodes.
- This is accomplished by recursively dividing the preexisting modifiable wires into smaller and smaller new wires.
- As an example, a series division performed on the one modifiable wire just described would create three new modifiable wires and would create two new nodes: one at (1/3, 0) and the other at (2/3, 0).

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES — CONTINUED

- In this way, the behavior of the functions that in our previous approach relied on the unique consecutive (circuit-wide) part number is now defined in terms of information that is local to the region of the circuit where the symmetry-breaking is performed.
- We believe that this new approach encourages the preservation of building blocks and thereby increases the efficiency of the crossover operation.

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES — CONTINUED

- **We use a new parallel division function in conjunction with this new approach. The first argument of the five-argument `PARALLEL_NEW` function specifies the direction in which the parallel division is to be performed.**
  - This first argument is one of the following two terminals: `UP_OR_LEFT` or `DOWN_OR_RIGHT`.
  - The terminal `UP_OR_LEFT` means that the parallel division will be performed upwards if the current modifiable wire or component is horizontal, but left if it is vertical.

SYMMETRY-BREAKING PROCEDURE USING GEOMETRIC COORDINATES — CONTINUED

- The terminal `DOWN_OR_RIGHT` means that the parallel division will be performed downwards if the current modifiable wire or component is horizontal, but right if it is vertical.
- Note that if all the modifiable wires in the embryo are either horizontal or vertical, all subsequent wires and components will be either horizontal or vertical.
- The remaining four arguments of the `PARALLEL_NEW` function are the construction-continuing subtrees that correspond to the four modifiable wires created by this function.

DEPTH-FIRST EVALUATION

- **Previous work used a breath-first order of evaluation during the developmental process.**
- **However, we have come to believe that this choice is not as efficient as the alternative (i.e., depth-first evaluation).**
- **When a subtree is evaluated using a depth-first order of evaluation in the developmental process, all the component-creating and topology-modifying functions in the subtree are evaluated before any other part of the overall program tree is evaluated. That is, the entire subtree is fully evaluated before the developmental process moves on to another subtree.**

DEPTH-FIRST EVALUATION — CONTINUED

- **Conversely, when a subtree is evaluated using a breath-first order of evaluation in the developmental process, the evaluation of each function in the subtree is followed (in general) by the evaluation of functions (usually many) that are not part of the subtree.**

DEPTH-FIRST EVALUATION — CONTINUED

- **The genetic algorithm and genetic programming work on the principle that there is something about the structure of a relatively fit individual that contributes to the individual's fitness.**
    - In genetic programming, the crossover operation creates new individuals by recombining subtrees of relatively fit individuals.
    - Thus, it seems reasonable that the crossover operation would be more efficient if the parts of a program that are moved by the crossover operation (i.e., subtrees) more closely corresponded to the sequence of component-creating and topology-modifying functions.

NEW `TWO_LEAD` FUNCTION FOR INSERTING TWO-LEADED COMPONENTS

- **In previous work, two-leaded components, (such as resistors, capacitors, and inductors) were inserted into the developing circuit by means of separate component-creating functions (e.g., the `R`, `C`, and `L` functions).**
    - Each of these functions has a construction-continuing subtree as one of its arguments.
    - Thus, with these functions, any crossover or mutation that changes one of these functions necessarily changes the construction-continuing subtree.
    - It may be advantageous for the evolutionary process to be able to change one two-leaded component into another without changing the construction-continuing subtree.

NEW `TWO_LEAD` FUNCTION FOR INSERTING TWO-LEADED COMPONENTS — CONTINUED

- **The new three-argument `TWO_LEAD` function replaces one modifiable wire (or component) with a series composition consisting of one modifiable wire, a two-leaded component, and a second modifiable wire.**

NEW `TWO_LEAD` FUNCTION FOR INSERTING TWO-LEADED COMPONENTS — CONTINUED

- **The first argument of the `TWO_LEAD` function specifies the identity and numerical value (sizing) of the component.**
    - The first argument of the `TWO_LEAD` function is a subtree whose root is always one of the following new one-argument component-creating functions: `R_NEW`, `C_NEW`, or `L_NEW`.
    - Each of these three new functions takes a value-setting subtree as its argument.
    - The value-setting subtree may be either a single perturbable numerical value or an arithmetic-performing subtree (in particular, an arithmetic-performing subtree containing free variables).

NEW `TWO_LEAD` FUNCTION FOR INSERTING TWO-LEADED COMPONENTS — CONTINUED

- **The second and third arguments of the `TWO_LEAD` function are the construction-continuing subtrees that correspond to the two modifiable wires created by this function.**

- **The important point is that the first argument of the `TWO_LEAD` function does not possess a construction-continuing subtree.**
    - As a result, the identify of the two-leaded component may be changed (by crossover or mutation) without changing either construction-continuing subtree.
    - That is, a two-leaded component can be changed in a localized way.

NEW Q TRANSISTOR-CREATING FUNCTION

- **Similarly, it may be advantageous for the evolutionary process to be able to change the orientation of a transistor or the transistor model without changing the construction-continuing subtrees associated with the insertion of the transistor into the developing circuit.**

- **The new six-argument Q function inserts a transistor into a developing circuit, with both the model and orientation of the transistor specified as parameters.**

NEW `Q` TRANSISTOR-CREATING FUNCTION — CONTINUED

- **The first argument of the `Q` function specifies the transistor model that is to be used. The set of available transistor models is specific to the problem at hand.**

- **The second argument establishes which end (polarity) of the preexisting modifiable wire will be bifurcated (if necessary) in inserting the transistor.**
    - It can take on the values `BIFURCATE_POSITIVE` or `BIFURCATE_NEGATIVE`.

NEW `Q` TRANSISTOR-CREATING FUNCTION — CONTINUED

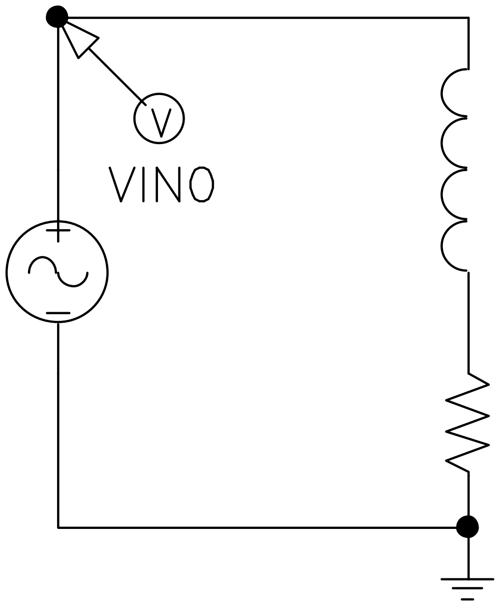
- **The third argument of the `Q` function specifies which of six possible permutations of the transistor's three leads is to be used.**
    - It can take on the values `B_C_E`, `B_E_C`, `C_B_E`, `C_E_B`, `E_B_C`, and `E_C_B`. For example, `C_B_E` causes the first point to be connected to the transistor's collector; the second point to be connected to the transistor's base; and the third point to be connected to the transistor's emitter.
    - As previously mentioned, the geometric coordinates define an underlying order of the three points to which the transistor's base, collector, and emitter may be connected.
    - Considering the second and third argument together, there are 12 possible ways of inserting a transistor.

NEW Q TRANSISTOR-CREATING FUNCTION — CONTINUED

- **The remaining three arguments of the Q function are the construction-continuing subtrees that correspond to the three modifiable wires created by this function.**

ZOBEL NETWORK WITH TWO FREE VARIABLES

- **An audio power amplifier driving a loudspeaker may be modeled, at first approximation, as an inductor L1 and resistor R1 situated in series between the incoming signal and ground.**



ZOBEL NETWORK — CONTINUED

- **When the loudspeaker is prone to instability, it may be desirable to make the reactive load appear to be a purely resistive load to the driving source.**
- **This may be accomplished by adding additional circuitry to the LR circuit.**
- **Thus, the problem is to find the topology of the unspecified additional circuitry as well as the sizing of each component of this additional circuitry such that the original LR circuit plus the additional circuitry together appear to the driving source to be a purely resistive load.**
- **The desired solution to this problem must be parameterized (i.e., general) because the sizing of at least some of the additional components will necessarily depend on the particular value, L_1, of the original inductor L1 and the particular value, R_1, of the original resistor R1.**

ZOBEL NETWORK — CONTINUED

• Boutin (2002) describes a mathematical solution to this problem in the form of what is called a *Zobel network*.

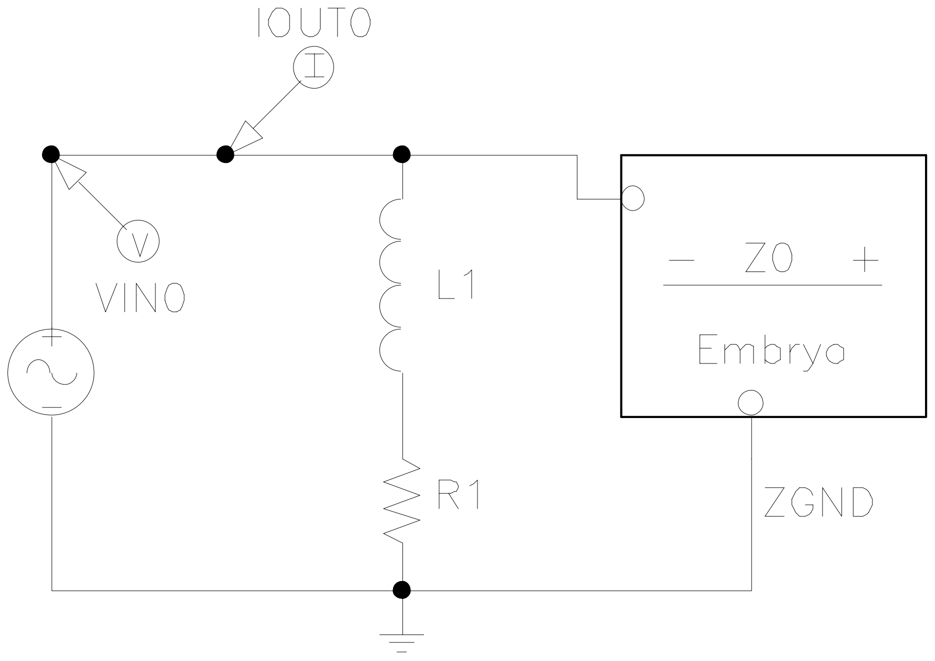
• BEcause the original LR circuit plus the added circuitry must together have the behavior of a resistor with resistance $R_{overall}$, then the current flowing through the circuit must (according to Ohm's law) be $V_{in}/R_{overall}$, where V_{in} is the voltage of the incoming signal.

• If, for example, the driving source is a step function rising from 0 to V_{max} at time t=t_{rise}, then the current of the original LR circuit plus the added circuitry must be a step function rising from 0 to $V_{max}/R_{overall}$ at time t=t_{rise}.

• The behavior of the original LR circuit starts at 0 and rises exponentially (with time constant L_1/R_1) toward its final value.

PREPARATORY STEPS FOR THE ZOBEL NETWORK PROBLEM WITH TWO FREE VARIABLES

INITIAL CIRCUIT



• **Initial circuit consists of an embryo and a test fixture**

• **The floating embryo consists of a single modifiable wire Z0 that is not initially connected to the circuit's input(s) or output(s)..**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

PROGRAM ARCHITECTURE

- **Because there must be one result-producing branch in the program tree for each modifiable wire in the embryo and there is one modifiable wire in the embryo, the architecture of each circuit-constructing program tree has one result-producing branch.**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FUNCTION SET

- **A constrained syntactic structure enforces the use of one function set for the arithmetic-performing subtrees, another for the first argument of the `TWO_LEAD` function, and yet another for all other parts of the program tree.**
- **The function set, F_{aps}, for the arithmetic-performing subtrees is**

F_{aps} = {+, −, *, %, RLOG, EXP}.

- **Because resistors, inductors, and capacitors may be used in this problem, the function set, $F_{two\text{-}lead}$, for the first argument of the `TWO_LEAD` function is**

$F_{two\text{-}lead}$ = {R_NEW, L_NEW, C_NEW}.

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FUNCTION SET — CONTINUED

- **The function set, F_{ccs}, for each construction-continuing subtree is**

F_{ccs} = {`TWO_LEAD`, `SERIES`, `PARALLEL_NEW`, `NODE`, `TWO_GROUND`, `THREE_GROUND`, `INPUT_0`}.

- **Note that because the input source is a voltage source and the probe point measures current at the same point, there is no need for a separate output probe point in this particular test fixture (and hence no need for the `OUTPUT_0` function in the function set).**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

TERMINAL SET

- **A constrained syntactic structure enforces the use of one terminal set for the value-setting subtrees, another for the first argument of the `PARALLEL_NEW` function, and yet another for all other parts of the program tree.**
- **In this problem, there are two free variables. They represent the inductance L_1 (called "`L1`" below) of the original inductor L1 and the resistance R_1 (called "`R1`") of the original resistor R1.**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

TERMINAL SET — CONTINUED

• The component value for each component possessing a parameter is established by an arithmetic-performing subtree that may contain perturbable numerical values, arithmetic operations, and the two free variables (`L1` and `R1`).

• The terminal set, T_{aps}, for the arithmetic-performing subtree for component-creating functions is

$T_{aps} = \{\Re_p, \texttt{L1}, \texttt{R1}\}$,

where $\Re_p$ denotes a perturbable numerical value.

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

TERMINAL SET — CONTINUED

• A constrained syntactic structure specifies the terminals that may appear as the first argument of the `PARALLEL_NEW` function. The terminal set, $T_{parallel}$, for the first argument of the `PARALLEL_NEW` function is

$T_{parallel}$ = {`UP_OR_LEFT`, `DOWN_OR_RIGHT`}.

• The terminal set, T_{ccs}, for each construction-continuing subtree is

T_{ccs} = {`END`, `SAFE_CUT`}.

• In problems involving parameterized topologies, the nonlinear mapping can potentially impede the evolutionary process and complicate the post-run analysis of mathematical expressions containing free variables. Thus it is not used for this problem.

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FITNESS MEASURE

- **When there are free variables in a problem, it is necessary to ascertain the behavior and characteristics of each candidate individual for a representative sample of values of each of the free variables.**
- **Multiple combinations of values of the free variables force generalization of the to-be-evolved circuit.**
- **The free variable for the inductance, L_1, of inductor L1 is `L1`). `L1` ranges over six values, namely 1.0, 2.5, 6.3, 15.8, 39.8, and 100.0 microhenrys.**
- **The free variable for the resistance, R_1, of resistor R1 is `R1`. `R1` ranges over six values, namely 10, 25, 63, 158, 398, and 1,000 Ohms.**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FITNESS MEASURE — CONTINUED

- **Thus, there are a total of 36 fitness cases, each representing a combination of values of the two free variables (L_1 and R_1).**
- **The input signal (driving source) for a particular one of the 36 fitness cases is**

$$1-e^{-\frac{t}{\tau}},$$

where the time constant, τ, is L_1/R_1. This curve has an initial amplitude of 0 Volts, rises with a time constant τ, and has a potential maximum amplitude of 1 Volt.
- **SPICE is instructed to perform a transient (time-domain) analysis for 10*L*/*R* seconds. There are 100 time steps. The current at the current probe, IOUT0, is reported for 101 times.**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FITNESS MEASURE — CONTINUED

- The contribution to fitness of each of the 36 fitness cases is equalized by weighting each fitness case by $R_{overall}$. In this connection, recall that a maximum input voltage of 1 Volt corresponds to a maximum desired current of $1/R_{overall}$.
- The number of hits is defined as the number of fitness cases for which the absolute error is less than or equal to 5% of the reciprocal of the value of $R_{overall}$ for that fitness case.

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FITNESS MEASURE — CONTINUED

- **The fitness measure is designed to encourage two types of parsimony.**
- **The first type of parsimony is based on the familiar idea of counting the circuit's components.**
- **The second type of parsimony, called *equational parsimony*, is the total number of mathematical functions in the arithmetic-performing subtrees. This type of parsimony is often appropriate when parameterized topologies are being used.**

PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

FITNESS MEASURE — CONTINUED

• For individuals scoring less than 36 hits, fitness is the sum, for each of the 101 time steps associated with each of the 36 fitness cases, of the product of $R_{overall}$ (the equalizing factor for the fitness case) and the weighted absolute value of the difference between the current at the current probe point, IOUT0, and the current value of $V_{in}/R_{overall}$ for the fitness case involved. The absolute value of the difference is weighted by 1.0 if the absolute error is less than or equal to 5% of the reciprocal of the value of $R_{overall}$ for that fitness case, but 10. 0 otherwise.

• For individuals scoring 36 hits, fitness is one trillionth of the sum of the total number of components (including the fixed components in the test fixture) and one tenth

of the total number of mathematical functions in the arithmetic-performing subtrees.

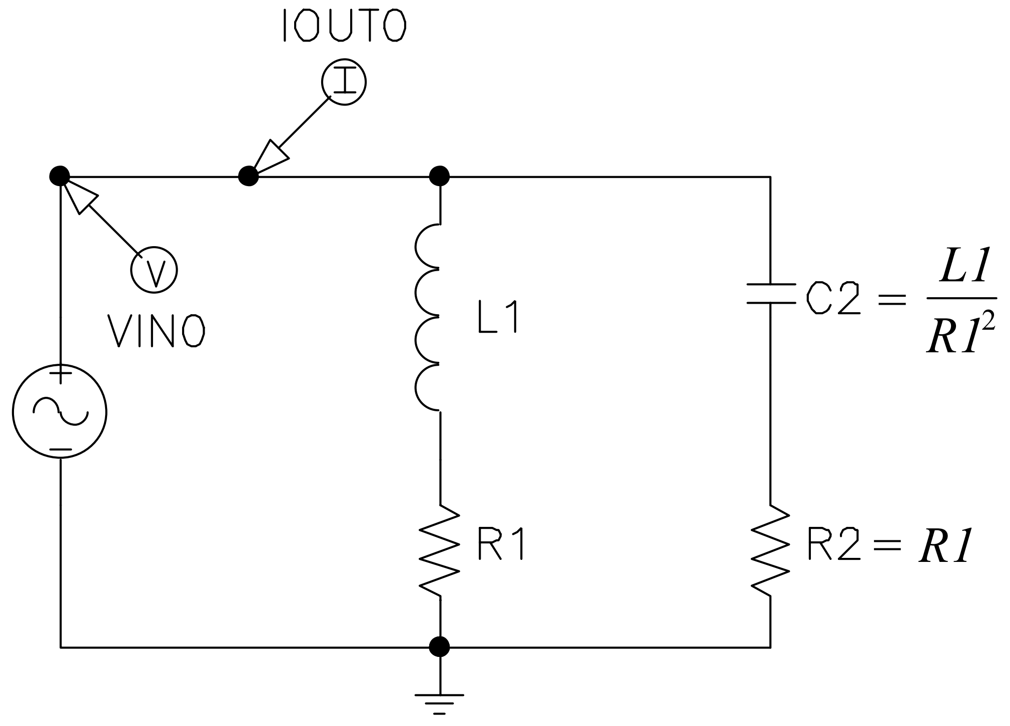
PREPARATORY STEPS FOR THE ZOBEL NETWORK — CONTINUED

CONTROL PARAMETERS

- **The population size is 500,000**

RESULTS — ZOBEL NETWORK

BEST-OF-RUN PARAMETERIZED TOPOLOGY (GEN 15)



- **Genetic programming produced this circuit's overall topology as well as the two mathematical expressions (containing the free variables L_1 and R_1) for specifying the parameter values for new capacitor C2 and new resistor R2.**

RESULTS — ZOBEL NETWORK — CONTINUED

- **The circuitry added by genetic programming consists a new capacitor C2 with component value**

$$L_1/R_1^2$$

and a new resistor R2 with component value

$$R_1.$$

- **These values for the new components in the genetically evolved circuit are exactly those that Boutin (2000) derived by mathematical analysis.**
- **The addition of the second shunt consisting of an appropriately valued new capacitor and an appropriately valued new resistor makes the overall load appear to be purely resistive to the driving source.**

RESULTS — ZOBEL NETWORK — CONTINUED

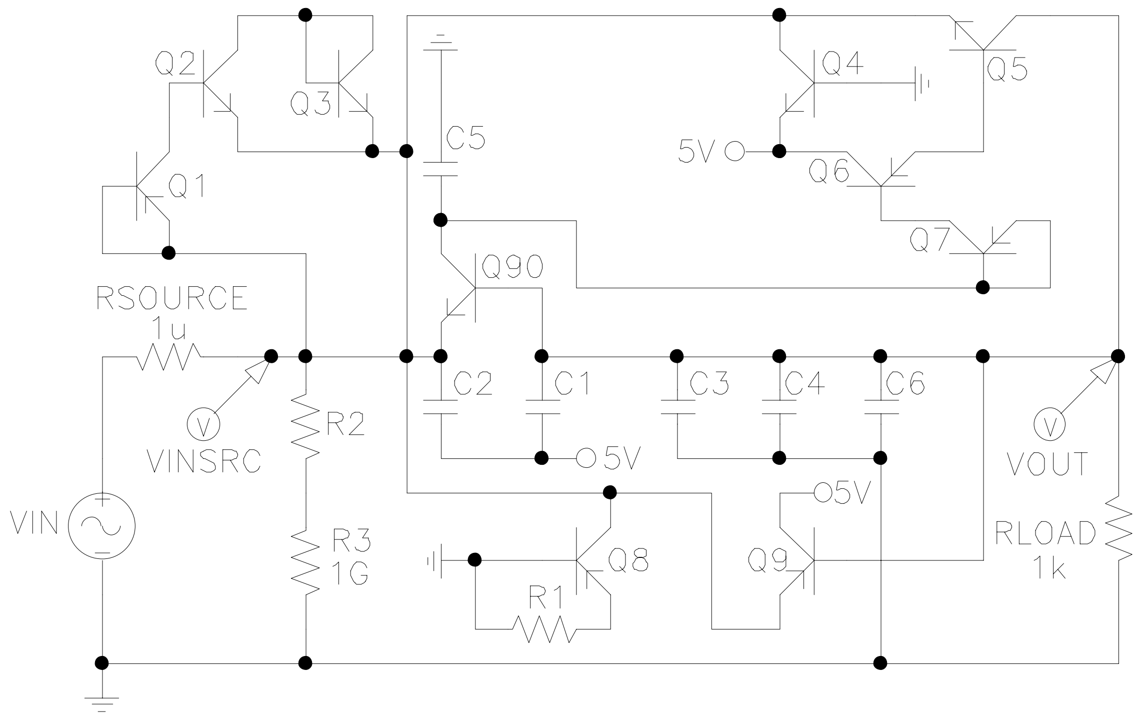
- **Genetic programming automatically created all the following in the single run that produced the additional circuitry:**
  - the topology of the additional circuitry, including
    - the total number of added components (two),
    - the type of each added component (i.e., one new capacitor and one new resistor),
    - all the connections between the circuit's added components, preexisting components, and accessible points of the test fixture,
  - the sizing of the additional circuitry, including
    - one mathematical expression containing both of the problem's free variables (L_1 and R_1) for establishing the sizing of the added capacitor C2, and
    - another mathematical expression containing one of the problem's free variables (R_1) for establishing the sizing of the added resistor R2.

ACTIVE LOWPASS FILTER WITH A FREE VARIABLE FOR THE PASSBAND BOUNDARY

- **The problem in this section is to evolve a one-input parameterized lowpass filter composed of transistors, capacitors, and resistors whose passband boundary is specified by a free variable.**

RESULTS — ACTIVE LOWPASS FILTER WITH A FREE VARIABLE FOR THE PASSBAND BOUNDARY

BEST-OF-RUN INDIVIDUAL — GENERATION 100



RESULTS — ACTIVE LOWPASS FILTER WITH A FREE VARIABLE FOR THE PASSBAND BOUNDARY

BEST-OF-RUN INDIVIDUAL — GENERATION 100

- **Has six capacitors and two resistors whose values are specified by the expressions below involving the free variable, *f*.**

$R_1 = f$

$R_2 = NLM(2\log f)$

$C_1 = NLM((\log f)^2 + \log f)$

$C_2 = NLM((\log f)^2 + \log f)$

$C_3 = NLM(\log f - (\log f)^2 + 9.6707)$

$C_4 = NLM(2\log f)$

$C_5 = NLM(5(\log f)^2 - 14.509)$

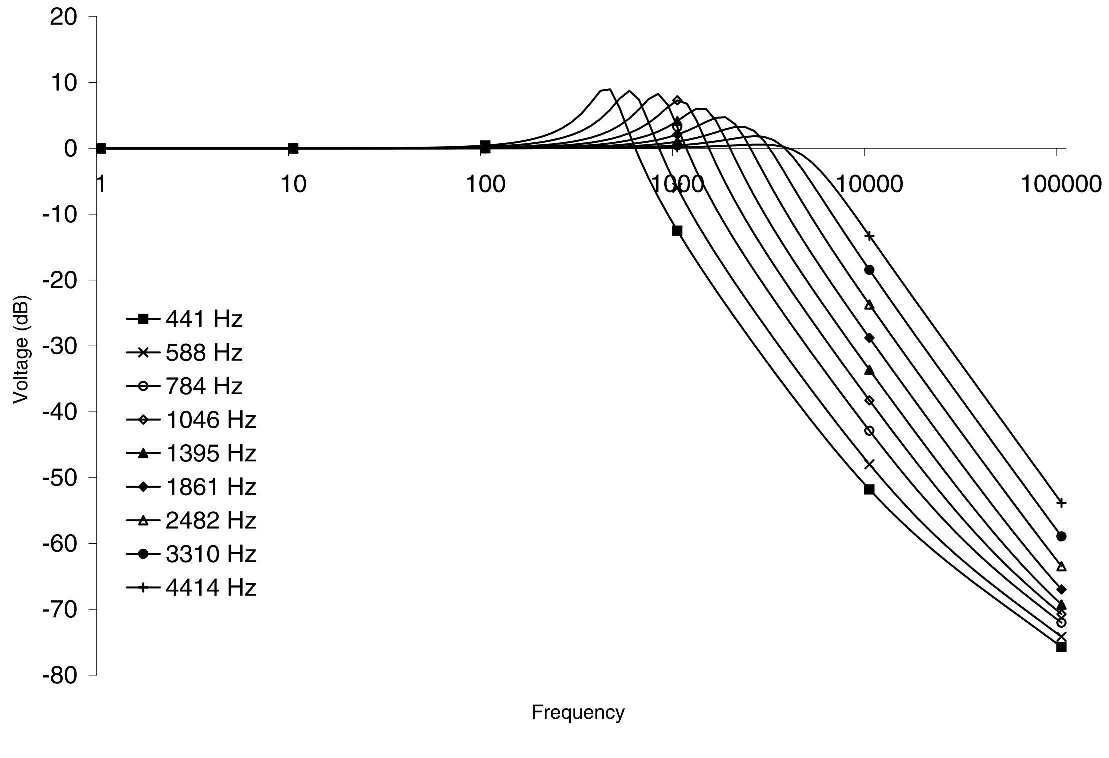
$C_6 = NLM(3.6900 + (\log f)^2 - \log f)$

RESULTS — ACTIVE LOWPASS FILTER WITH A FREE VARIABLE FOR THE PASSBAND BOUNDARY

- **Genetic programming automatically created all the following in the single run that produced the parameterized circuit of figure 10.14:**
  - the circuit's topology, including
    - the total number of components (18) in addition to those of the test fixture,
    - the type of each component (i.e., 10 transistors, six capacitors, and two resistors),
    - all the connections between the circuit's components, accessible points of the test fixture, and the power source, and
  - the circuit's sizing as expressed by eight mathematical expressions containing the problem's free variable (*f*) for establishing the sizing of the circuit's components (R1, R2, C1, C2, C3, C4, C5, and C6).

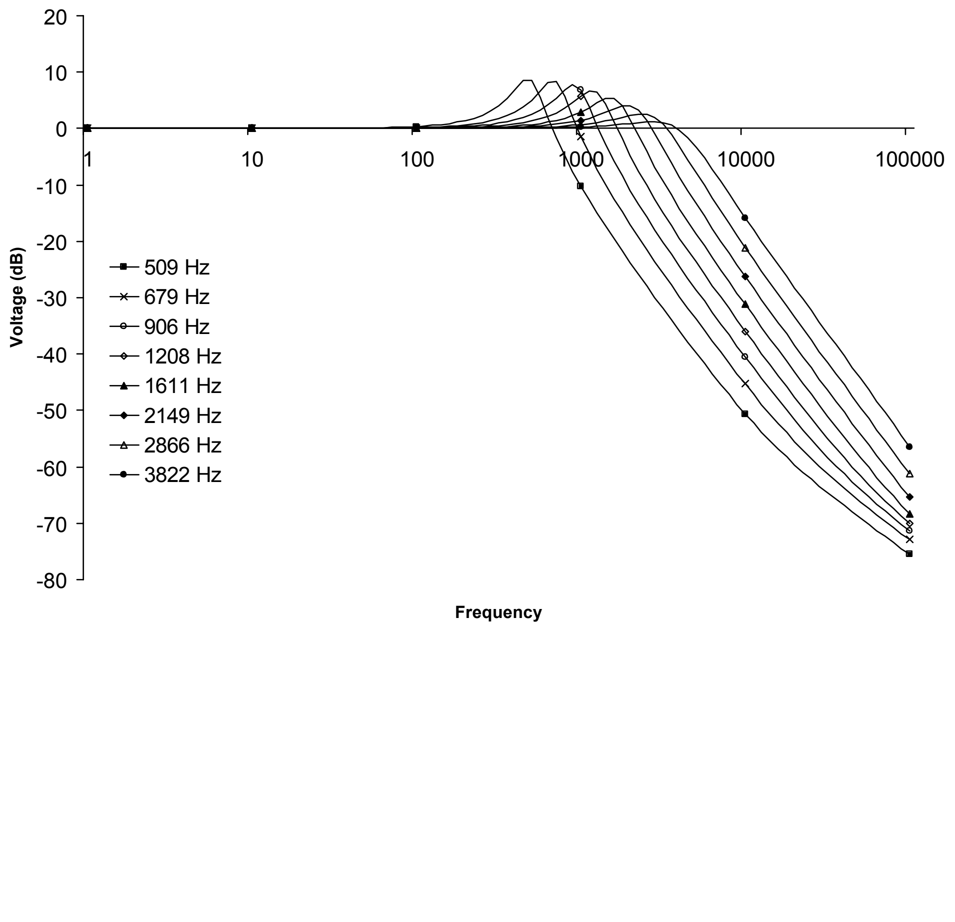
RESULTS — ACTIVE LOWPASS FILTER WITH A FREE VARIABLE FOR THE PASSBAND BOUNDARY

FREQUENCY RESPONSE OF THE BEST-OF-RUN CIRCUIT FROM GENERATION 100 FOR THE 9 IN-SAMPLE VALUES OF FREQUENCY (0.316 DB AVERAGE ABSOLUTE ERROR)



RESULTS — ACTIVE LOWPASS FILTER WITH A FREE VARIABLE FOR THE PASSBAND BOUNDARY

FREQUENCY RESPONSE FOR THE BEST-OF-RUN CIRCUIT FROM GENERATION 100 FOR THE 8 OUT-OF-SAMPLE FREQUENCIES (0.42 DB AVERAGE ABSOLUTE ERROR)


Voltage (dB)
Frequency
509 Hz
679 Hz
906 Hz
1208 Hz
1611 Hz
2149 Hz
2866 Hz
3822 Hz