ALDCC simulator

1.0.0

Generated by Doxygen 1.16.1

Chapter 1

Details

ALDCCsim is a C++ command-line tool designed to solve linear electrical circuits using Modified Nodal Analysis (MNA). The solver supports resistors, independent voltage sources, and independent current sources. It also features a bisection-based optimization engine to find specific component values required to achieve a desired circuit output (voltage, current, or power).

Below is a full description of the program focusing on Analysis, Internal Specification and Testing. For External Specification read the readme in root directory `../README.md`

1.1 Problem and analysis

Based on the provided code and the task description image, the core problem is the development of a software simulation tool capable of analyzing linear direct current (DC) circuits. The fundamental objective is to parse a textual representation of a circuit, commonly referred to as a netlist, and calculate the steady-state operating point for every component. This involves determining the voltage across, current through, and power dissipated by each resistor, voltage source, and current source in the network.

The mathematical foundation selected to solve this problem is Modified Nodal Analysis (MNA). This approach is necessary because the circuit elements include independent voltage sources, which standard nodal analysis cannot easily handle without supernodes. MNA addresses this by expanding the system of linear equations to solve simultaneously for node potentials and the currents flowing through voltage sources. The program can construct an augmented matrix where rows correspond to Kirchhoff's Current Law (KCL) for each node and the constitutive equations for voltage sources, solving the system Ax=b using Gaussian elimination with partial pivoting to ensure numerical stability.

Several improvements can be made on top of what the task specifies. I decided to add a simple parallel voltage source check and a check for nodes with only current sources connected since these configurations result in the circuit being indeterminate. Obviously in some cases we need the output to be more precise than others and too high precision at all times results in the output being less readable. This is why I also added a third command line switch allowing the user to specify precision of output parameters.

The task mentions only solving a system with known element data and outputting currents, voltages and power balance, but in real use cases I found that I have to perform the opposite almost just as often. This is why I additionally implemented search mode which uses bisection to find some element parameter knowing one of the currents, voltages or power balances. The program still isn't suitable for solving more elaborate problems, but this add-on makes its applicability significantly higher.

I also decided to replace the `I` current source symbol from the task decription with `J` as it is more commonly used and can't be misinterpretted as current.

1.2 Internal specification

1.2.1 Global Context

The simulation state is aggregated into a central data structure, Context, which serves as the primary data transfer object between modules.

- Element Storage: All circuit components are stored in a `std::vector<[Element](@ref Element)>`, where each Element contains its type, connection nodes, value, and computed results (voltage, current, power).
- Equation System: The MNA system is represented by an augmented matrix (`std::vector<std::vector<double>> equations`), where the number of unknowns corresponds to the maximum node index plus the number of independent voltage sources.
- Bisection State: To support the optimization mode, the context holds a pointer to a variable_elem (the component being tuned) and a GivenValue structure defining the target metric (desired_value) and type (v_type).

1.2.2 Mathematical Core

1.2.2.1 Modified Nodal Analysis (MNA)

The solver constructs a linear system `Ax=b` to determine the circuit state.

Equation Formulation:

- KCL Equations: Generated for every non-ground node, summing currents leaving the node.
- Constitutive Equations: Generated for voltage sources, introducing the source current as an additional unknown.
- Ground Reference: The potential of Node `1` is explicitly constrained to zero.

Matrix Solution:

- The system is solved using Gaussian elimination with partial pivoting to maximize numerical stability before decoding the results into node potentials and source currents.

1.2.2.2 Bisection Optimization

When a variable element is detected, the bisect module wraps the MNA solver in an iterative loop.

- Monotonicity Check: The variable element is perturbed by EPSILON (2E-10) to determine the sign of the derivative of the output with respect to the input.
- Bound Search: The search interval is iteratively doubled until the target value is bracketed by the output range.
- Convergence: The algorithm performs a fixed number of iterations (BISECTION_PREC = 50) to narrow the interval, updating the output_value in the Context after every step.

1.2.3 Validation and Error Handling

The system includes pre-solve validation to reject topologically invalid circuits:

- Indeterminate Nodes: Detects nodes connected exclusively to current sources, which result in undefined node potentials.
- Parallel Voltage Sources: Identifies voltage sources sharing the same node pair, which creates a singular matrix due to indeterminate loop currents.
- Singular Matrices: The Gaussian elimination process throws a runtime error if inconsistent or dependent equations (e.g., 0 = non-zero) are encountered.

1.2.4 Output Generation

The simulator writes a comprehensive report to the specified output file.

- Element Table: Lists every element's type, nodes, and calculated Voltage (U), Current (I), and Power (P).
- Potentials: Lists the calculated electric potential (V) for every node in the circuit.
- Formatting: All floating-point values are rounded to the user-specified precision (defaulting to 4 decimal places) using a scaling factor, with specific handling to prevent negative zero outputs.

1.3 Testing - Basic Mode

1.3.1 Example 1

Command: `ALDCCsim.exe -i data.txt -o solution.txt -p 5`
data.txt:

```
E 1 2 4
R 2 3 16
R 1 3 16
R 3 4 8
R 1 4 8
J 1 4 6
```

> Note: Because we marked the ground node on the left as node `1`, we can do the same on the right - they are effectively the same node.

solution.txt:

```
 Element |      U[V]     |      I[A]     |      P[W]
---------+---------------+---------------+---------------
   E12   |             4 |      -0.83333 |       -3.3333
   R23   |        13.333 |       0.83333 |        11.111
   R13   |        17.333 |        1.0833 |        18.778
   R34   |        15.333 |        1.9167 |        29.389
   R14   |        32.667 |        4.0833 |        133.39
   J14   |        32.667 |             6 |           196

 Node |      V[V]
------+---------------
  1   |             0
  2   |             4
  3   |        17.333
  4   |        32.667
```

1.3.2 Example 2

Command: `ALDCCsim.exe -i data.txt -o solution.txt`
data.txt:

```
R 2 1 4
E 3 2 20
R 3 1 12
R 1 4 4
E 1 6 0
R 3 6 2
R 6 4 4
R 5 4 8
J 5 6 3
```

Note: In this circuit we can model an ideal ammeter using a `0V` voltage source.

solution.txt:

```
 Element |      U[V]     |      I[A]     |      P[W]
---------+---------------+---------------+---------------
   R21   |            -14 |          -3.5 |             49
   E32   |             20 |           3.5 |             70
   R31   |              6 |           0.5 |              3
   R14   |             -6 |          -1.5 |              9
   E16   |              0 |           1.5 |              0
   R36   |              6 |             3 |             18
   R64   |             -6 |          -1.5 |              9
   R54   |             24 |             3 |             72
   J56   |             30 |             3 |             90

 Node |      V[V]
------+---------------
  1   |             0
  2   |            14
  3   |            -6
  4   |            -6
  5   |           -30
  6   |             0
```

1.4 Testing - Search Mode

1.4.1 Example 1

Command: `ALDCCsim.exe -i data.txt -o solution.txt`
data.txt:

```
E 1 2 10
R 2 3 x
R 3 1 50 U 5
```

solution.txt:

```
Answer: x = 50

 Element |      U[V]     |      I[A]     |      P[W]
---------+---------------+---------------+---------------
   E12   |             10 |            0.1 |              1
   R23   |             -5 |          -0.1 |            0.5
   R31   |             -5 |          -0.1 |            0.5

 Node |      V[V]
------+---------------
  1   |             0
  2   |            10
  3   |             5
```

1.4.2 Example 2

Command: `ALDCCsim.exe -i data.txt -o solution.txt -p 7`
data.txt:

```
E 5 1 x
R 1 2 3.8
R 2 3 3.8
J 4 2 1.2
R 5 4 3.8
E 5 3 18.5 P 4.9
```

solution.txt:

```
Answer: x = 12

 Element |      U[V]     |      I[A]     |      P[W]
```

```
---------+---------------+---------------+---------------
   E51   |            12 |          -1.5 |           -17
   R12   |           5.6 |           1.5 |            8.2
   R23   |             1 |          0.26 |           0.27
   J42   |            22 |           1.2 |             26
   R54   |          -4.6 |          -1.2 |            5.5
   E53   |            18 |          0.26 |            4.9

 Node |      V[V]
------+---------------
  1   |             0
  2   |           5.6
  3   |           6.6
  4   |           -16
  5   |           -12
```

The same example with higher precision:

Command: `ALDCCsim.exe -i data.txt -o solution.txt -p 5`

data.txt:

```
E 5 1 x
R 1 2 3.8
R 2 3 3.8
J 4 2 1.2
R 5 4 3.8
E 5 3 18.5 P 4.9
```

solution.txt:

```
Answer: x = 11.927

 Element |      U[V]     |      I[A]     |      P[W]
---------+---------------+---------------+---------------
   E51   |        11.927 |       -1.4649 |       -17.471
   R12   |        5.5665 |        1.4649 |        8.1541
   R23   |        1.0065 |       0.26486 |       0.26658
   J42   |        22.054 |           1.2 |        26.464
   R54   |         -4.56 |          -1.2 |         5.472
   E53   |          18.5 |       0.26486 |           4.9

 Node |      V[V]
------+---------------
  1   |             0
  2   |        5.5665
  3   |         6.573
  4   |       -16.487
  5   |       -11.927
```

1.4.3 Example 3

Command: `ALDCCsim.exe -i data.txt -o solution.txt -p 7`

data.txt:

```
J 3 1 x P 5.5
R 1 2 1
R 2 3 5
R 2 3 20
```

solution.txt:

```
Answer: x = 1.048809

 Element |      U[V]     |      I[A]     |      P[W]
---------+---------------+---------------+---------------
   J31   |      5.244044 |      1.048809 |           5.5
   R12   |     -1.048809 |     -1.048809 |           1.1
   R23   |     -4.195235 |    -0.8390471 |          3.52
   R23   |     -4.195235 |    -0.2097618 |          0.88

 Node |      V[V]
------+---------------
  1   |             0
  2   |     -1.048809
  3   |     -5.244044
```

Chapter 2

Directory Hierarchy

2.1 Directories

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Directory Documentation

6.1 docs Directory Reference

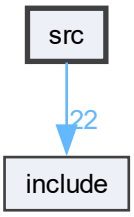
6.2 include Directory Reference

Files

- file bisect.hpp
- file cli.hpp
- file constants.hpp
- file context.hpp
- file element.hpp
- file file_io.hpp
- file solver.hpp
- file util.hpp

6.3 src Directory Reference

Directory dependency graph for src:



Files

- file bisect.cpp
- file cli.cpp
- file file_io.cpp
- file main.cpp
- file solver.cpp
- file util.cpp

Chapter 7

Namespace Documentation

7.1 cli Namespace Reference

Contains functions and structures for command-line argument parsing.

Classes

- struct Input

  Holds the parsed command-line input values.

Enumerations

- enum class Switch { none = 0 , in = 'i' , out = 'o' , prec = 'p' }

  Defines the valid command-line switches for the program.

Functions

- Input parse (int argc, char ∗argv[ ])

  Parses the command line arguments to extract input file, output file, and precision.

7.1.1 Detailed Description

Contains functions and structures for command-line argument parsing.

7.1.2 Enumeration Type Documentation

7.1.2.1 Switch

`enum class cli::Switch [strong]`

Defines the valid command-line switches for the program.

Enumerator

none	
in	
out	
prec	

7.1.3 Function Documentation

7.1.3.1 parse()

```
Input cli::parse (
            int argc,
            char * argv[])
```

Parses the command line arguments to extract input file, output file, and precision.
It checks for the '-i', '-o', and '-p' switches and reads the subsequent argument. It also prints usage instructions if no arguments are provided. Throws a std::runtime_error if a switch is expected but not found, or if the required input/output filenames are missing.

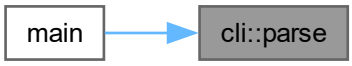
Parameters

argc	The number of command-line arguments.
argv	The array of command-line argument strings.

Returns

Input A struct containing the parsed input filename, output filename, and precision.

Here is the caller graph for this function:



7.2 constants Namespace Reference

Global constants and configuration variables for the circuit solver.

Functions

- double prec_factor ()

Variables

- uint32_t PRECISION = 4
- constexpr double ZERO_TRESHOLD = 1e-12
- constexpr double EPSILON = 2e-10
- constexpr uint32_t BISECTION_PREC = 50

7.2.1 Detailed Description

Global constants and configuration variables for the circuit solver.

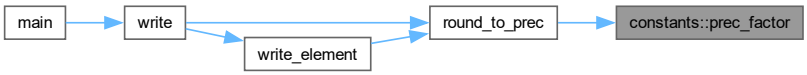
7.2.2 Function Documentation

7.2.2.1 prec_factor()

```
double constants::prec_factor ()  [inline]
```

Here is the call graph for this function:

Here is the caller graph for this function:



7.2.3 Variable Documentation

7.2.3.1 BISECTION_PREC

```
uint32_t constants::BISECTION_PREC = 50  [constexpr]
```

7.2.3.2 EPSILON

```
double constants::EPSILON = 2e-10  [constexpr]
```

7.2.3.3 PRECISION

```
uint32_t constants::PRECISION = 4  [inline]
```

7.2.3.4 ZERO_TRESHOLD

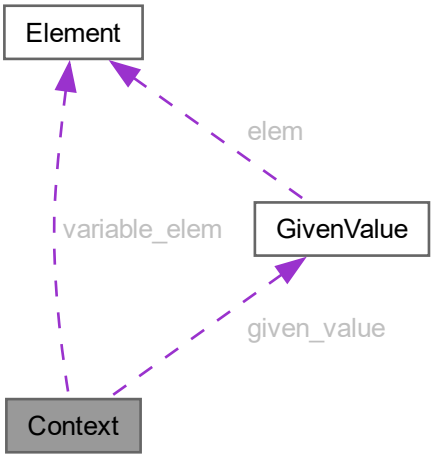
```
double constants::ZERO_TRESHOLD = 1e-12  [constexpr]
```

Chapter 8

Class Documentation

8.1 Context Struct Reference

A structure holding all runtime data and state for the circuit analysis.
`#include <context.hpp>`
Collaboration diagram for Context:



Public Attributes

- uint8_t max_node = 0
- uint8_t number_of_unknowns = 0
- std::vector< Element > elements
- std::vector< double > potentials
- Element * variable_elem = nullptr
- GivenValue given_value
- double output_value = 0
- std::unordered_map< uint16_t, uint8_t > v_source_currents_map
- std::vector< std::vector< double > > equations

8.1.1 Detailed Description

A structure holding all runtime data and state for the circuit analysis.

This includes the circuit elements, the maximum node index, the MNA system's equations, and state variables for the bisection algorithm.

8.1.2 Member Data Documentation

8.1.2.1 elements

```
std::vector<Element> Context::elements
```

8.1.2.2 equations

```
std::vector<std::vector<double> > Context::equations
```

8.1.2.3 given_value

```
GivenValue Context::given_value
```

8.1.2.4 max_node

```
uint8_t Context::max_node = 0
```

8.1.2.5 number_of_unknowns

```
uint8_t Context::number_of_unknowns = 0
```

8.1.2.6 output_value

```
double Context::output_value = 0
```

8.1.2.7 potentials

```
std::vector<double> Context::potentials
```

8.1.2.8 v_source_currents_map

```
std::unordered_map<uint16_t, uint8_t> Context::v_source_currents_map
```

8.1.2.9 variable_elem

```
Element* Context::variable_elem = nullptr
```

The documentation for this struct was generated from the following file:

- include/context.hpp

8.2 Element Struct Reference

Represents a single circuit component and its solved parameters.

`#include <element.hpp>`

Public Member Functions

- bool is_directed_outwards (const uint8_t node) const

  Checks if the element's defined direction points away from a given node.
- int8_t sign_of_current (const uint8_t node) const

  Gets the sign of the element's current with respect to a given node in KCL.
- uint8_t other_node (const uint8_t node) const

  Finds the node at the other end of the element.

Public Attributes

- ElementType type
- std::array< uint8_t, 2 > nodes
- double value
- double voltage
- double current
- double power

8.2.1 Detailed Description

Represents a single circuit component and its solved parameters.

8.2.2 Member Function Documentation

8.2.2.1 is_directed_outwards()

```
bool Element::is_directed_outwards (
            const uint8_t node) const  [inline]
```

Checks if the element's defined direction points away from a given node.
For sources, this indicates the current is flowing from `nodes[0]` to `nodes[1]`.

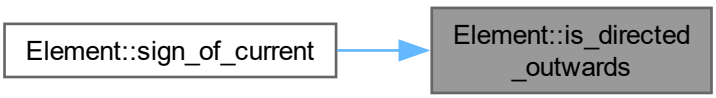
Parameters

node	The node to check against.

Returns

bool True if the element's `nodes[0]` is the given node.

Here is the caller graph for this function:



8.2.2.2 other_node()

```
uint8_t Element::other_node (
            const uint8_t node) const  [inline]
```

Finds the node at the other end of the element.

Parameters

node	The known node.

Returns

uint8_t The index of the connecting node.

8.2.2.3 sign_of_current()

```
int8_t Element::sign_of_current (
            const uint8_t node) const  [inline]
```

Gets the sign of the element's current with respect to a given node in KCL.
Current is defined as positive when leaving the node (+1) and negative when entering the node (-1).

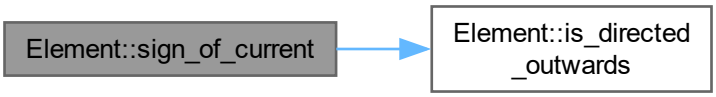
Parameters

node	The node of interest.

Returns

int8_t 1 if current is directed out of the node, -1 if directed into the node.

Here is the call graph for this function:



8.2.3 Member Data Documentation

8.2.3.1 current

```
double Element::current
```

8.2.3.2 nodes

```
std::array<uint8_t, 2> Element::nodes
```

8.2.3.3 power

```
double Element::power
```

8.2.3.4 type

```
ElementType Element::type
```

8.2.3.5 value

```
double Element::value
```

8.2.3.6 voltage

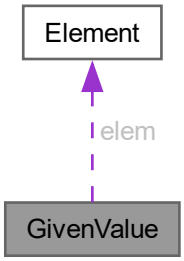
```
double Element::voltage
```

The documentation for this struct was generated from the following file:

- include/element.hpp

8.3 GivenValue Struct Reference

Holds information about the desired circuit output for the bisection method.
`#include <element.hpp>`
Collaboration diagram for GivenValue:



Public Attributes

- Element * elem
- ValueType v_type
- double desired_value

8.3.1 Detailed Description

Holds information about the desired circuit output for the bisection method.

8.3.2 Member Data Documentation

8.3.2.1 desired_value

```
double GivenValue::desired_value
```

8.3.2.2 elem

```
Element* GivenValue::elem
```

8.3.2.3 v_type

```
ValueType GivenValue::v_type
```

The documentation for this struct was generated from the following file:

- include/element.hpp

8.4 cli::Input Struct Reference

Holds the parsed command-line input values.
`#include <cli.hpp>`

Public Attributes

- std::string ifname = ""
- std::string ofname = ""
- uint8_t prec = 4

8.4.1 Detailed Description

Holds the parsed command-line input values.

8.4.2 Member Data Documentation

8.4.2.1 ifname

```
std::string cli::Input::ifname = ""
```

8.4.2.2 ofname

```
std::string cli::Input::ofname = ""
```

8.4.2.3 prec

```
uint8_t cli::Input::prec = 4
```

The documentation for this struct was generated from the following file:

- include/cli.hpp

Chapter 9

File Documentation

9.1 docs/doxygen_detailed_documentation.md File Reference

9.2 include/bisect.hpp File Reference

```
#include "context.hpp"
```

Functions

- void update_bisection_output (Context &ctx)

  Updates the output value for the bisection algorithm.
- int8_t find_derivative_sign (Context &ctx)

  Determines the sign of the derivative of the output value with respect to the variable element value.
- double find_bound (Context &ctx, int8_t derivative_sign)

  Finds an initial search bound for the bisection method.
- void bisect (Context &ctx)

  Implements the bisection method to find the variable element's value that yields the desired output value.

9.2.1 Function Documentation

9.2.1.1 bisect()

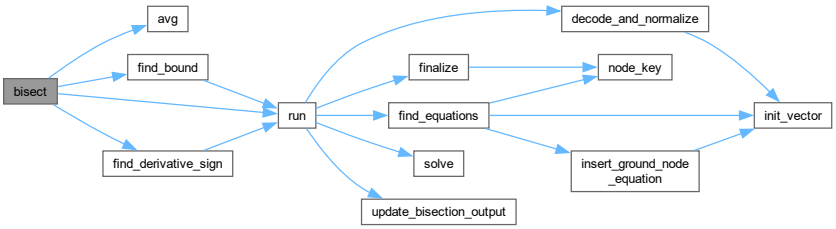
```
void bisect (
            Context & ctx)
```

Implements the bisection method to find the variable element's value that yields the desired output value.
It iteratively narrows the search interval (`range[0]` and `range[1]`) based on the sign of the difference between the current output and the desired output, taking into account the derivative sign. The iteration runs for 'constants::↩ BISECTION_PREC' steps.

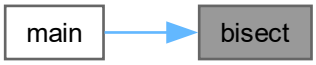
Parameters

ctx	The Context object containing all circuit data.

Here is the call graph for this function:



Here is the caller graph for this function:



9.2.1.2 find_bound()

```
double find_bound (
            Context & ctx,
            int8_t derivative_sign)
```

Finds an initial search bound for the bisection method.
The function iteratively doubles an initial value until the sign of the difference between the current output and the desired output flips, indicating the solution is within the bounded range.

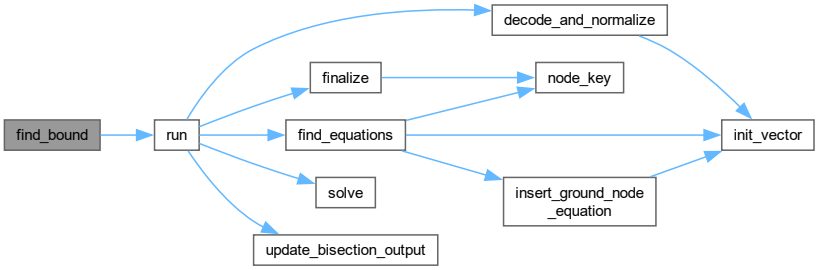
Parameters

ctx	The Context object containing all circuit data.
derivative_sign	The sign of the output value's derivative (1 or -1).

Returns

double The calculated upper bound for the variable element's value.

Here is the call graph for this function:



Here is the caller graph for this function:

9.2.1.3 find_derivative_sign()

```
int8_t find_derivative_sign (
            Context & ctx)
```

Determines the sign of the derivative of the output value with respect to the variable element value.
It checks how the `ctx.output_value` changes when the `ctx.variable_elem->value` is perturbed by a small value (`constants::EPSILON`). This determines the monotonic relationship used in the bisection.

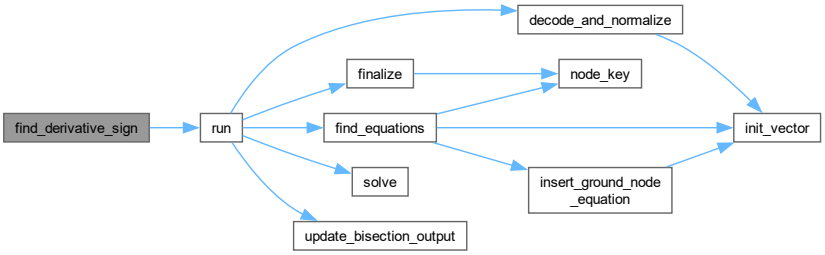
Parameters

ctx	The Context object containing all circuit data.

Returns

int8_t 1 if the derivative is positive, -1 if negative.

Here is the call graph for this function:



Here is the caller graph for this function:

9.2.1.4 update_bisection_output()

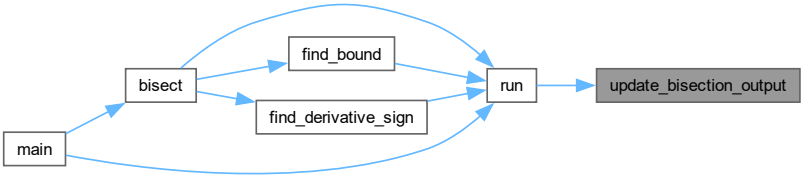
```
void update_bisection_output (
            Context & ctx)
```

Updates the output value for the bisection algorithm.
Based on the `ValueType` specified in `ctx.given_value`, this sets `ctx.output_value` to the voltage, current, or power of the monitored element. This function is called after a full circuit run to check the result.

Parameters

ctx	The Context object containing all circuit data.

Here is the caller graph for this function:



9.3 bisect.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "context.hpp"
00003
00013 void update_bisection_output(Context& ctx);
00014
00024 int8_t find_derivative_sign(Context& ctx);
00025
00037 double find_bound(Context& ctx, int8_t derivative_sign);
00038
00048 void bisect(Context& ctx);
```

9.4 include/cli.hpp File Reference

```
#include <cstdint>
#include <string>
```

Classes

- struct cli::Input

  Holds the parsed command-line input values.

Namespaces

- namespace cli

  Contains functions and structures for command-line argument parsing.

Enumerations

- enum class cli::Switch { cli::none = 0 , cli::in = 'i' , cli::out = 'o' , cli::prec = 'p' }

  Defines the valid command-line switches for the program.

Functions

- Input cli::parse (int argc, char ∗argv[ ])

  Parses the command line arguments to extract input file, output file, and precision.

9.5 cli.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <cstdint>
00003 #include <string>
00008 namespace cli {
00009
00013     enum class Switch {
00014         none = 0,
00015         in = 'i',
00016         out = 'o',
00017         prec = 'p'
00018     };
00019
00023     struct Input {
00024         std::string ifname = "";
00025         std::string ofname = "";
00026         uint8_t prec = 4;
00027     };
00028
00041     Input parse(int argc, char* argv[]);
00042 }
```

9.6 include/constants.hpp File Reference

```
#include "util.hpp"
```

Namespaces

- namespace constants

  Global constants and configuration variables for the circuit solver.

Functions

- double constants::prec_factor ()

Variables

- uint32_t constants::PRECISION = 4
- constexpr double constants::ZERO_TRESHOLD = 1e-12
- constexpr double constants::EPSILON = 2e-10
- constexpr uint32_t constants::BISECTION_PREC = 50

9.7 constants.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "util.hpp"
00003
00008 namespace constants {
00009     inline uint32_t PRECISION = 4;
00010     inline double prec_factor() { return power(10, PRECISION); }
00011     constexpr double ZERO_TRESHOLD = 1e-12;
00012     constexpr double EPSILON = 2e-10;
00013     constexpr uint32_t BISECTION_PREC = 50;
00014 }
```

9.8 include/context.hpp File Reference

```
#include <cstdint>
#include <unordered_map>
#include <vector>
#include "element.hpp"
```

Classes

- struct Context

  A structure holding all runtime data and state for the circuit analysis.

9.9 context.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <cstdint>
00003 #include <unordered_map>
00004 #include <vector>
00005
00006 #include "element.hpp"
00007
00015 struct Context {
00016     uint8_t max_node = 0;
00017     uint8_t number_of_unknowns = 0;
00018     std::vector<Element> elements;
00019     std::vector<double> potentials;
```

```
00020
00021     Element* variable_elem = nullptr;
00022     GivenValue given_value;
00023     double output_value = 0;
00024
00025     std::unordered_map<uint16_t, uint8_t> v_source_currents_map;
00026     std::vector<std::vector<double» equations;
00027 };
```

9.10 include/element.hpp File Reference

```
#include <array>
#include <cstdint>
```

Classes

- struct Element

  Represents a single circuit component and its solved parameters.
- struct GivenValue

  Holds information about the desired circuit output for the bisection method.

Enumerations

- enum ElementType { v_source = 'E' , c_source = 'J' , resistor = 'R' }

  Enumeration of supported circuit element types.
- enum class ValueType { voltage = 'U' , current = 'I' , power = 'P' }

  Enumeration of the measurable values that can be the target for the bisection algorithm.

9.10.1 Enumeration Type Documentation

9.10.1.1 ElementType

`enum ElementType`

Enumeration of supported circuit element types.
The values are set to the single-character representation used in the input file.

Enumerator

v_source	
c_source	
resistor	

9.10.1.2 ValueType

`enum class ValueType [strong]`

Enumeration of the measurable values that can be the target for the bisection algorithm.
The values are set to the single-character representation used in the input file.

Enumerator

voltage	
current	
power	

9.11 element.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <array>
00003 #include <cstdint>
00004
00010 enum ElementType {
00011     v_source = 'E',
00012     c_source = 'J',
00013     resistor = 'R'
00014 };
00015
00020 struct Element {
00021     ElementType type;
00022     std::array<uint8_t, 2> nodes;
00023     double value;
00024
00025     double voltage;
00026     double current;
00027     double power;
00028
00036     bool is_directed_outwards(const uint8_t node) const {
00037         return nodes[0] == node;
00038     }
00039
00047     int8_t sign_of_current(const uint8_t node) const {
00048         return static_cast<int8_t>(2 * is_directed_outwards(node) - 1);    // 1 if current leaves, -1
      when enters
00049     }
00050
00057     uint8_t other_node(const uint8_t node) const {
00058         return nodes[0] == node? nodes[1] : nodes[0];
00059     }
00060 };
00061
00067 enum class ValueType {
00068     voltage = 'U',
00069     current = 'I',
00070     power = 'P'
00071 };
00072
00077 struct GivenValue {
00078     Element* elem;
00079     ValueType v_type;
00080     double desired_value;
00081 };
```

9.12 include/file_io.hpp File Reference

```
#include <string>
#include "context.hpp"
```

Functions

- bool read (Context &ctx, const std::string &filename)

  Reads circuit element data from an input file.
- void write_element (std::ofstream &file, const Element &elem)

  Writes the solved parameters (voltage, current, power) for a single element to the output file.
- void write (const Context &ctx, const std::string &filename, bool do_bisection)

  Writes the final results of the circuit analysis to the output file.

9.12.1 Function Documentation

9.12.1.1 read()

```
bool read (
            Context & ctx,
            const std::string & filename)
```

Reads circuit element data from an input file.

Parses each line for element type, node 1, node 2, and element value. It also detects the presence of a variable element (missing value) and a desired output value for the bisection method. Updates the `ctx.max_node`. Throws a std::runtime_error on failure to read, too many variable elements, or mismatched solving procedure (variable element vs. given value).

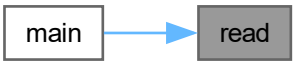
Parameters

ctx	The Context object to store the parsed data.
filename	The name of the input file.

Returns

bool True if a variable element and desired output were found, indicating bisection is needed.

Here is the caller graph for this function:



9.12.1.2 write()

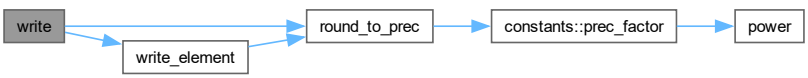
```
void write (
            const Context & ctx,
            const std::string & filename,
            bool do_bisection)
```

Writes the final results of the circuit analysis to the output file.
Includes the element table (U, I, P) and the node potential table (V). If bisection was performed, it also writes the final value of the variable element. Applies rounding to the specified precision.

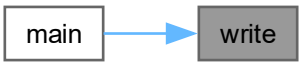
Parameters

ctx	The Context object containing the elements and bisection data.
filename	The name of the output file.
do_bisection	Boolean flag indicating if bisection was performed.

Here is the call graph for this function:



Here is the caller graph for this function:



9.12.1.3 write_element()

```
void write_element (
            std::ofstream & file,
            const Element & elem)
```

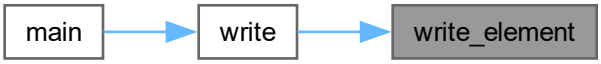
Writes the solved parameters (voltage, current, power) for a single element to the output file.
The output is formatted into columns with alignment and rounding applied.

Parameters

file	The output file stream.
elem	The Element object to write.

Here is the call graph for this function:

Here is the caller graph for this function:



9.13 file_io.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <string>
00003
00004 #include "context.hpp"
00005
00019 bool read(Context& ctx, const std::string& filename);
00020
00029 void write_element(std::ofstream& file, const Element& elem);
```

```
00030
00042 void write(const Context& ctx, const std::string& filename, bool do_bisection);
```

9.14 include/solver.hpp File Reference

```
#include "context.hpp"
```

Functions

- void run (Context &ctx, bool do_bisection)

  Runs the main circuit analysis procedure.
- void find_equations (Context &ctx)

  Constructs the system of linear equations for the circuit using Modified Nodal Analysis (MNA).
- void solve (Context &ctx)

  Solves the system of linear equations stored in the context using Gaussian elimination with partial pivoting.
- std::pair< std::vector< double >, std::vector< double > > decode_and_normalize (Context &ctx)

  Decodes the results from the reduced row echelon form of the equations.
- void finalize (Context &ctx, const std::vector< double > &potentials, const std::vector< double > &currents)

  Calculates and updates the voltage, current, and power for every circuit element.

9.14.1 Function Documentation

9.14.1.1 decode_and_normalize()

```
std::pair< std::vector< double >, std::vector< double > > decode_and_normalize (
            Context & ctx)
```

Decodes the results from the reduced row echelon form of the equations.
Extracts the solved node potentials and voltage source currents from the matrix. It also checks for and handles rows representing identity and throws an error if the circuit is contradictory or not fully determinate.

Parameters

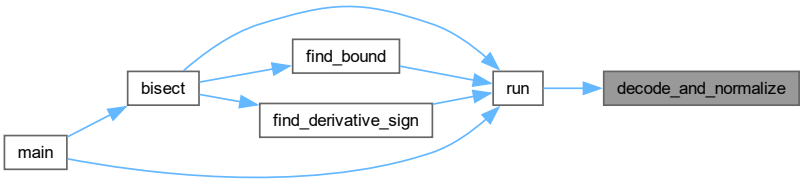
ctx	The Context object containing all circuit data.

Returns

std::pair<std::vector<double>, std::vector<double>> A pair of vectors: the first contains node potentials, the second contains voltage source currents.

Here is the call graph for this function:

Here is the caller graph for this function:



9.14.1.2 finalize()

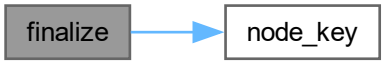
```
void finalize (
            Context & ctx,
            const std::vector< double > & potentials,
            const std::vector< double > & currents)
```

Calculates and updates the voltage, current, and power for every circuit element.
Uses the solved node potentials and voltage source currents to determine the final operating parameters for all resistors, current sources, and voltage sources.

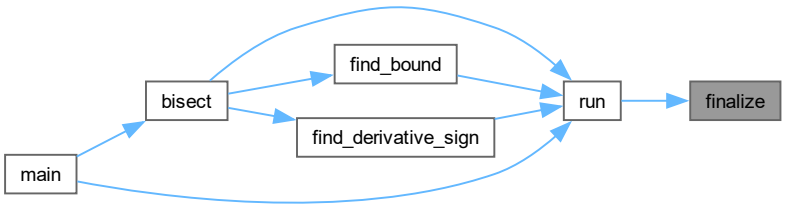
Parameters

ctx	The Context object containing all circuit data.
potentials	A vector of solved node potentials.
currents	A vector of solved voltage source currents.

Here is the call graph for this function:



Here is the caller graph for this function:



9.14.1.3 find_equations()

```
void find_equations (
            Context & ctx)
```

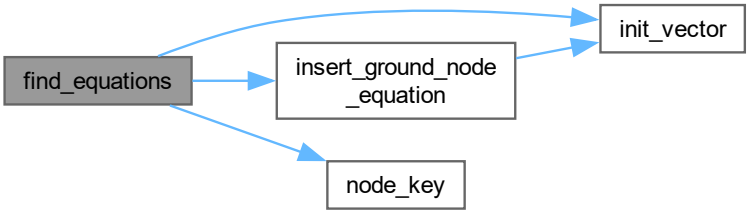
Constructs the system of linear equations for the circuit using Modified Nodal Analysis (MNA).
This function creates equations based on:

1. Ground node (Node 1) potential is zero.
2. Voltage sources (introducing a new unknown for every voltage source current).
3. Kirchhoff's Current Law (KCL) for every non-ground node.

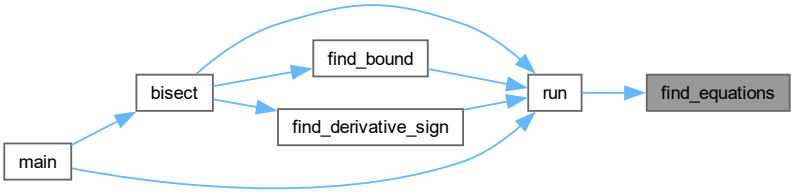
Parameters

ctx	The Context object containing all circuit data.

Here is the call graph for this function:



Here is the caller graph for this function:



9.14.1.4 run()

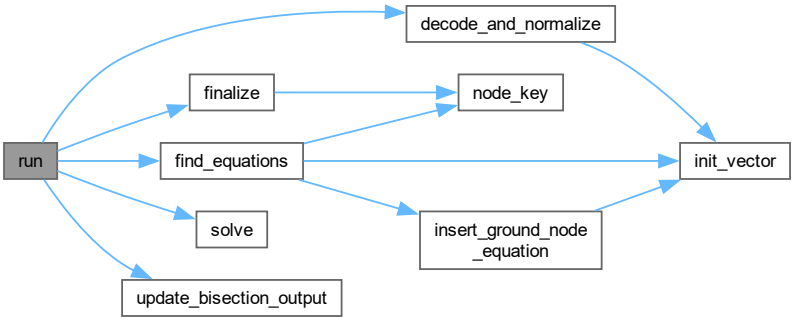
```
void run (
            Context & ctx,
            bool do_bisection)
```

Runs the main circuit analysis procedure.
Finds all necessary nodal and source equations, solves the resulting system of linear equations and decodes the results into node potentials and source currents. If bisection is enabled, it updates the output value for the bisection algorithm.

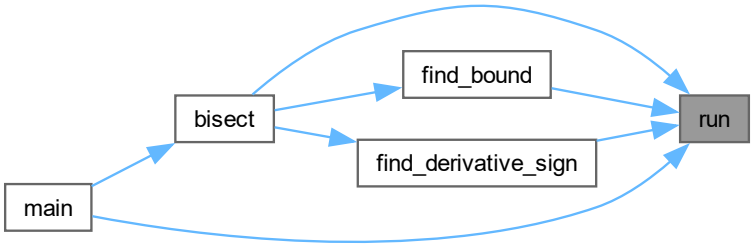
Parameters

ctx	The Context object containing all circuit data.
do_bisection	Boolean flag indicating whether to perform bisection updates.

Here is the call graph for this function:



Here is the caller graph for this function:



9.14.1.5 solve()

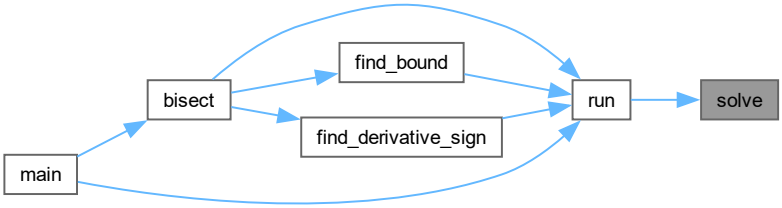
```
void solve (
            Context & ctx)
```

Solves the system of linear equations stored in the context using Gaussian elimination with partial pivoting.
The equations are represented as an augmented matrix stored in `ctx.equations`. After solving, the matrix is in reduced row echelon form. Throws a std::runtime_error if contradictory equations are found.

Parameters

ctx	The Context object containing all circuit data.

Here is the caller graph for this function:



9.15 solver.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include "context.hpp"
00003
00015 void run(Context& ctx, bool do_bisection);
00016
00027 void find_equations(Context& ctx);
00028
00038 void solve(Context& ctx);
00039
00051 std::pair<std::vector<double>, std::vector<double» decode_and_normalize(Context& ctx);
00052
00063 void finalize(Context& ctx, const std::vector<double>& potentials, const std::vector<double>&
      currents);
```

9.16 include/util.hpp File Reference

```
#include <cstdint>
#include <vector>
#include "element.hpp"
```

Functions

- uint8_t get_number_of_unknowns (const std::vector< Element > &elements, uint8_t max_node)

  Calculates the total number of unknowns in the MNA system.
- void init_vector (std::vector< double > &vector, uint8_t n)

  Initializes a vector of doubles with a specified number of zeros.
- void insert_ground_node_equation (std::vector< std::vector< double > > &equations, uint8_t unknowns)

  Inserts the equation for the ground node (Node 1) into the system.
- uint16_t node_key (uint8_t node_0, uint8_t node_1)

  Creates a unique 16-bit key for a node pair, independent of the order.
- bool current_source_only_node (const std::vector< Element > &elements, uint8_t max_node)

  Checks if there is a node connected only to current sources.
- bool parallel_voltage_sources (const std::vector< Element > &elements)

  Checks for voltage sources connected in parallel.
- double avg (double a, double b)

  Calculates the average (arithmetic mean) of two double-precision numbers.
- constexpr double power (double base, int exp)

  Calculates the base raised to the power of the exponent.
- double round_to_prec (double value)

  Rounds a double-precision value to the specified number of significant digits.

9.16.1 Function Documentation

9.16.1.1 avg()

```
double avg (
            double a,
            double b)
```

Calculates the average (arithmetic mean) of two double-precision numbers.

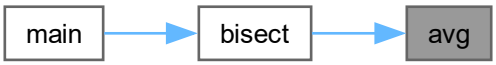
Parameters

a	The first number.
b	The second number.

Returns

double The average of a and b.

Here is the caller graph for this function:



9.16.1.2 current_source_only_node()

```
bool current_source_only_node (
            const std::vector< Element > & elements,
            uint8_t max_node)
```

Checks if there is a node connected only to current sources.
This condition makes the node's voltage indeterminate, as only KCL equations (which govern current) apply, not Ohm's law (which relates voltage and current).

Parameters

elements	A constant reference to the vector of circuit elements.
max_node	The highest node index.

Returns

bool True if such a node exists, false otherwise.

Here is the caller graph for this function:

9.16.1.3 get_number_of_unknowns()

```
uint8_t get_number_of_unknowns (
            const std::vector< Element > & elements,
            uint8_t max_node)
```

Calculates the total number of unknowns in the MNA system.
The number of unknowns is equal to the maximum node index (for node potentials) plus the number of voltage sources (for voltage source currents).

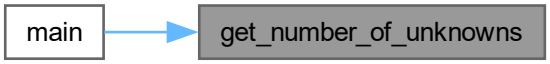
Parameters

elements	A constant reference to the vector of circuit elements.
max_node	The highest node index in the circuit.

Returns

uint8_t The total number of unknowns.

Here is the caller graph for this function:



9.16.1.4 init_vector()

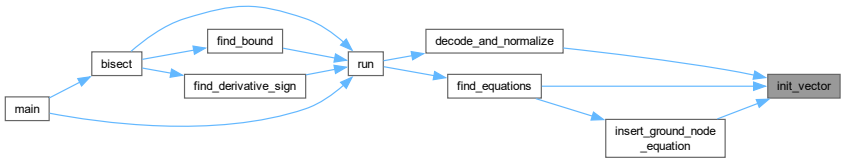
```
void init_vector (
            std::vector< double > & vector,
            uint8_t n)
```

Initializes a vector of doubles with a specified number of zeros.

Parameters

vector	The vector to be initialized.
n	The number of zeros to append.

Here is the caller graph for this function:



9.16.1.5 insert_ground_node_equation()

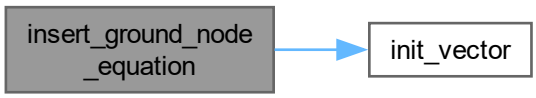
```
void insert_ground_node_equation (
            std::vector< std::vector< double > > & equations,
            uint8_t unknowns)
```

Inserts the equation for the ground node (Node 1) into the system.

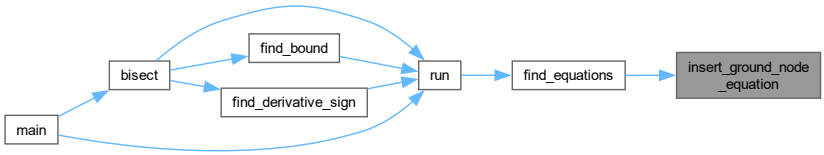
Parameters

equations	The vector of equation rows (augmented matrix).
unknowns	The total number of unknowns in the system.

Here is the call graph for this function:



Here is the caller graph for this function:



9.16.1.6 node_key()

```
uint16_t node_key (
            uint8_t node_0,
            uint8_t node_1)
```

Creates a unique 16-bit key for a node pair, independent of the order.
The key is constructed by shifting the greater node to the upper 8 bits and the smaller node to the lower 8 bits.

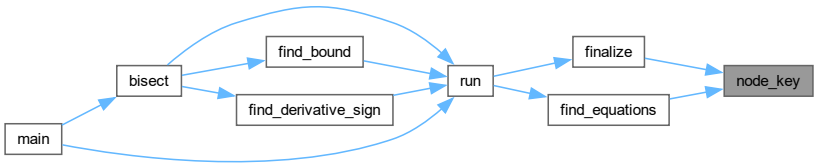
Parameters

node_0	The first node index.
node_1	The second node index.

Returns

uint16_t The unique node key.

Here is the caller graph for this function:



9.16.1.7 parallel_voltage_sources()

```
bool parallel_voltage_sources (
            const std::vector< Element > & elements)
```

Checks for voltage sources connected in parallel.
Parallel voltage sources (connected to the same two nodes) lead to an indeterminate current in the branches, making the system unsolvable.

Parameters

elements	A constant reference to the vector of circuit elements.

Returns

bool True if parallel voltage sources are found, false otherwise.

Here is the caller graph for this function:

9.16.1.8 power()

```
double power (
            double base,
            int exp)  [constexpr]
```

Calculates the base raised to the power of the exponent.
This is a `constexpr` implementation of the power function for use in compile-time calculations.

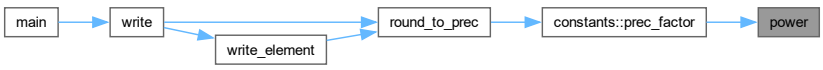
Parameters

base	The number to be raised to a power.
exp	The integer exponent.

Returns

double The result of base to the power of exp.

Here is the caller graph for this function:



9.16.1.9 round_to_prec()

```
double round_to_prec (
            double value)
```

Rounds a double-precision value to the specified number of significant digits.
Uses `constants::prec_factor()` to scale the value for rounding. Special handling to ensure negative zero is not returned.

Parameters

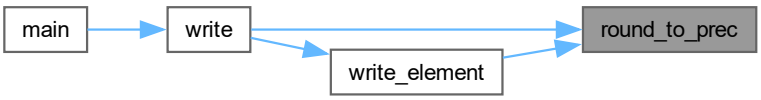
value	The value to be rounded.

Returns

double The rounded value.

Here is the call graph for this function:

Here is the caller graph for this function:



9.17 util.hpp

Go to the documentation of this file.

```
00001 #pragma once
00002 #include <cstdint>
00003 #include <vector>
00004
00005 #include "element.hpp"
00006
00017 uint8_t get_number_of_unknowns(const std::vector<Element>& elements, uint8_t max_node);
00018
00025 void init_vector(std::vector<double>& vector, uint8_t n);
00026
00033 void insert_ground_node_equation(std::vector<std::vector<double»& equations, uint8_t unknowns);
00034
00045 uint16_t node_key(uint8_t node_0, uint8_t node_1);
00046
00057 bool current_source_only_node(const std::vector<Element>& elements, uint8_t max_node);
00058
00068 bool parallel_voltage_sources(const std::vector<Element>& elements);
00069
00077 double avg(double a, double b);
00078
00089 constexpr double power(double base, int exp) {
00090     double result = 1.0;
00091     while (exp-- > 0) result *= base;
00092     return result;
00093 }
00094
00104 double round_to_prec(double value);
```

9.18 src/bisect.cpp File Reference

```
#include "bisect.hpp"
#include "element.hpp"
#include "constants.hpp"
#include "solver.hpp"
#include "util.hpp"
```

Functions

- void update_bisection_output (Context &ctx)

  Updates the output value for the bisection algorithm.
- int8_t find_derivative_sign (Context &ctx)

  Determines the sign of the derivative of the output value with respect to the variable element value.
- double find_bound (Context &ctx, int8_t derivative_sign)

  Finds an initial search bound for the bisection method.
- void bisect (Context &ctx)

  Implements the bisection method to find the variable element's value that yields the desired output value.

9.18.1 Function Documentation

9.18.1.1 bisect()

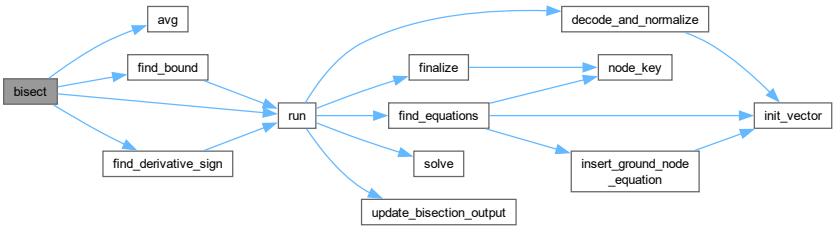
```
void bisect (
            Context & ctx)
```

Implements the bisection method to find the variable element's value that yields the desired output value.
It iteratively narrows the search interval (`range[0]` and `range[1]`) based on the sign of the difference between the current output and the desired output, taking into account the derivative sign. The iteration runs for 'constants::↩ BISECTION_PREC' steps.

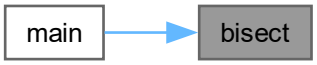
Parameters

ctx	The Context object containing all circuit data.

Here is the call graph for this function:



Here is the caller graph for this function:



9.18.1.2 find_bound()

```
double find_bound (
            Context & ctx,
            int8_t derivative_sign)
```

Finds an initial search bound for the bisection method.
The function iteratively doubles an initial value until the sign of the difference between the current output and the desired output flips, indicating the solution is within the bounded range.

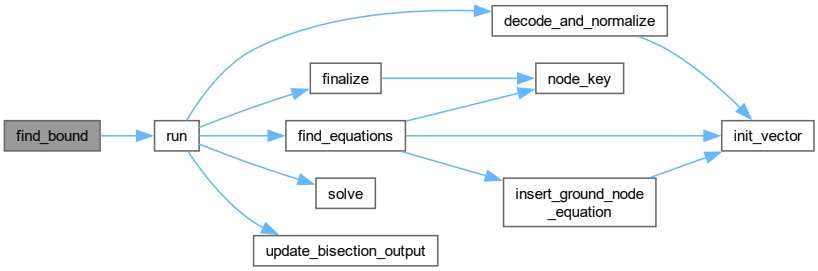
Parameters

ctx	The Context object containing all circuit data.
derivative_sign	The sign of the output value's derivative (1 or -1).

Returns

double The calculated upper bound for the variable element's value.

Here is the call graph for this function:



Here is the caller graph for this function:

9.18.1.3 find_derivative_sign()

```
int8_t find_derivative_sign (
            Context & ctx)
```

Determines the sign of the derivative of the output value with respect to the variable element value.
It checks how the `ctx.output_value` changes when the `ctx.variable_elem->value` is perturbed by a small value (`constants::EPSILON`). This determines the monotonic relationship used in the bisection.

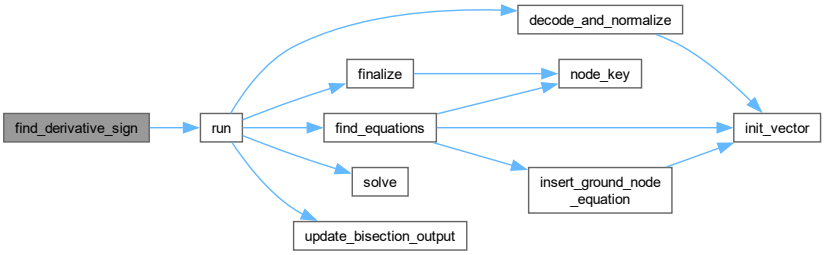
Parameters

ctx	The Context object containing all circuit data.

Returns

int8_t 1 if the derivative is positive, -1 if negative.

Here is the call graph for this function:



Here is the caller graph for this function:

9.18.1.4 update_bisection_output()

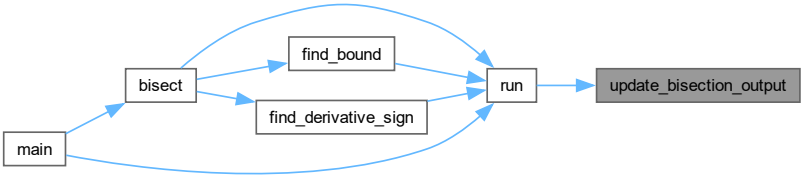
```
void update_bisection_output (
            Context & ctx)
```

Updates the output value for the bisection algorithm.
Based on the `ValueType` specified in `ctx.given_value`, this sets `ctx.output_value` to the voltage, current, or power of the monitored element. This function is called after a full circuit run to check the result.

Parameters

ctx	The Context object containing all circuit data.

Here is the caller graph for this function:



9.19 src/cli.cpp File Reference

```
#include "cli.hpp"
#include <cstdint>
#include <iostream>
#include <stdexcept>
#include <string>
```

Namespaces

- namespace cli

  Contains functions and structures for command-line argument parsing.

Functions

- Input cli::parse (int argc, char *argv[])

  Parses the command line arguments to extract input file, output file, and precision.

9.20 src/file_io.cpp File Reference

```
#include <fstream>
#include <iomanip>
#include <sstream>
#include <cmath>
#include "file_io.hpp"
#include "constants.hpp"
```

Functions

- bool read (Context &ctx, const std::string &filename)

  Reads circuit element data from an input file.
- void write_element (std::ofstream &file, const Element &elem)

  Writes the solved parameters (voltage, current, power) for a single element to the output file.
- void write (const Context &ctx, const std::string &filename, bool do_bisection)

  Writes the final results of the circuit analysis to the output file.

9.20.1 Function Documentation

9.20.1.1 read()

```
bool read (
            Context & ctx,
            const std::string & filename)
```

Reads circuit element data from an input file.
Parses each line for element type, node 1, node 2, and element value. It also detects the presence of a variable element (missing value) and a desired output value for the bisection method. Updates the `ctx.max_node`. Throws a std::runtime_error on failure to read, too many variable elements, or mismatched solving procedure (variable element vs. given value).

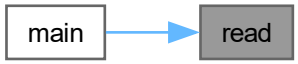
Parameters

ctx	The Context object to store the parsed data.
filename	The name of the input file.

Returns

bool True if a variable element and desired output were found, indicating bisection is needed.

Here is the caller graph for this function:



9.20.1.2 write()

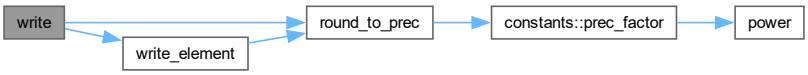
```
void write (
            const Context & ctx,
            const std::string & filename,
            bool do_bisection)
```

Writes the final results of the circuit analysis to the output file.
Includes the element table (U, I, P) and the node potential table (V). If bisection was performed, it also writes the final value of the variable element. Applies rounding to the specified precision.

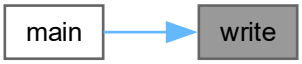
Parameters

ctx	The Context object containing the elements and bisection data.
filename	The name of the output file.
do_bisection	Boolean flag indicating if bisection was performed.

Here is the call graph for this function:



Here is the caller graph for this function:



9.20.1.3 write_element()

```
void write_element (
            std::ofstream & file,
            const Element & elem)
```

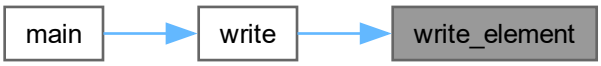
Writes the solved parameters (voltage, current, power) for a single element to the output file.
The output is formatted into columns with alignment and rounding applied.

Parameters

file	The output file stream.
elem	The Element object to write.

Here is the call graph for this function:

Here is the caller graph for this function:



9.21 src/main.cpp File Reference

```
#include "util.hpp"
#include "solver.hpp"
#include "file_io.hpp"
#include "bisect.hpp"
#include "context.hpp"
#include "cli.hpp"
#include "constants.hpp"
#include <fstream>
```

Functions

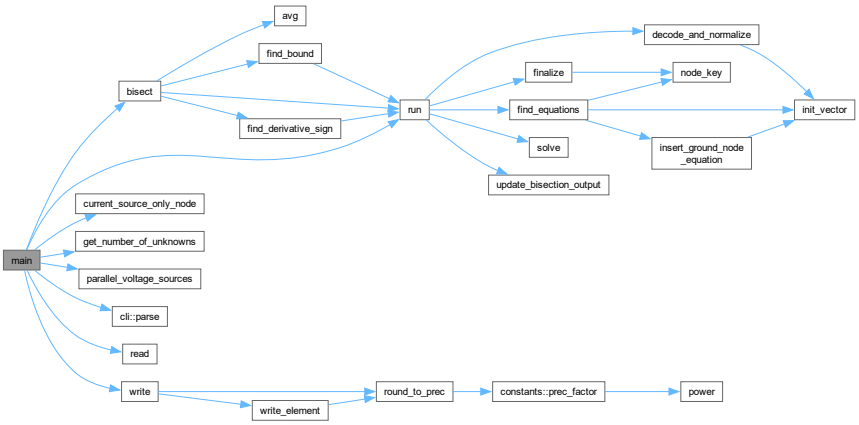
- int main (int argc, char ∗argv[ ])

9.21.1 Function Documentation

9.21.1.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

Here is the call graph for this function:



9.22 src/solver.cpp File Reference

```
#include <array>
#include <cmath>
#include <cstdint>
#include <fstream>
#include <ranges>
#include <vector>
#include "util.hpp"
#include "solver.hpp"
#include "bisect.hpp"
#include "constants.hpp"
```

Functions

- void run (Context &ctx, bool do_bisection)

  Runs the main circuit analysis procedure.
- void find_equations (Context &ctx)

  Constructs the system of linear equations for the circuit using Modified Nodal Analysis (MNA).
- void solve (Context &ctx)

  Solves the system of linear equations stored in the context using Gaussian elimination with partial pivoting.
- std::pair< std::vector< double >, std::vector< double > > decode_and_normalize (Context &ctx)

  Decodes the results from the reduced row echelon form of the equations.
- void finalize (Context &ctx, const std::vector< double > &potentials, const std::vector< double > &currents)

  Calculates and updates the voltage, current, and power for every circuit element.

9.22.1 Function Documentation

9.22.1.1 decode_and_normalize()

```
std::pair< std::vector< double >, std::vector< double > > decode_and_normalize (
            Context & ctx)
```

Decodes the results from the reduced row echelon form of the equations.
Extracts the solved node potentials and voltage source currents from the matrix. It also checks for and handles rows representing identity and throws an error if the circuit is contradictory or not fully determinate.

Parameters

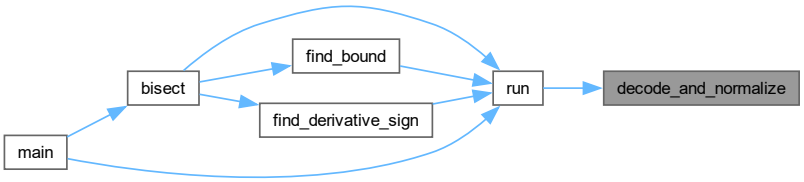
ctx	The Context object containing all circuit data.

Returns

std::pair<std::vector<double>, std::vector<double>> A pair of vectors: the first contains node potentials, the second contains voltage source currents.

Here is the call graph for this function:

Here is the caller graph for this function:



9.22.1.2 finalize()

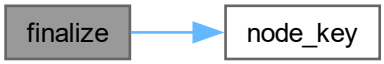
```
void finalize (
            Context & ctx,
            const std::vector< double > & potentials,
            const std::vector< double > & currents)
```

Calculates and updates the voltage, current, and power for every circuit element.
Uses the solved node potentials and voltage source currents to determine the final operating parameters for all resistors, current sources, and voltage sources.

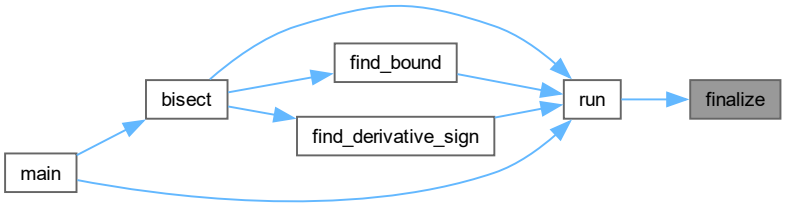
Parameters

ctx	The Context object containing all circuit data.
potentials	A vector of solved node potentials.
currents	A vector of solved voltage source currents.

Here is the call graph for this function:



Here is the caller graph for this function:



9.22.1.3 find_equations()

```
void find_equations (
            Context & ctx)
```

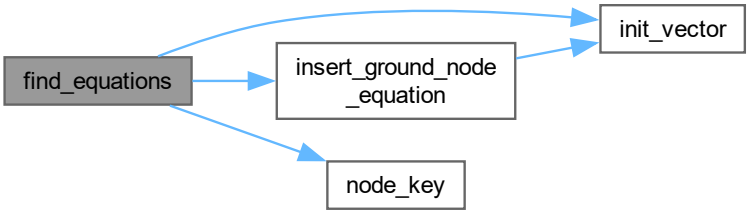
Constructs the system of linear equations for the circuit using Modified Nodal Analysis (MNA).
This function creates equations based on:

1. Ground node (Node 1) potential is zero.
2. Voltage sources (introducing a new unknown for every voltage source current).
3. Kirchhoff's Current Law (KCL) for every non-ground node.

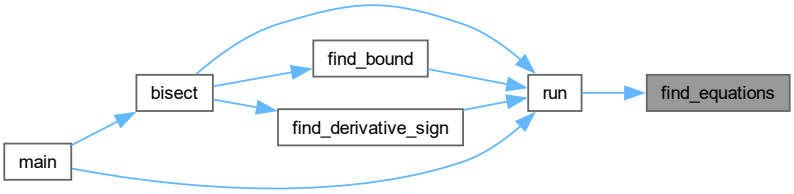
Parameters

ctx	The Context object containing all circuit data.

Here is the call graph for this function:



Here is the caller graph for this function:



9.22.1.4 run()

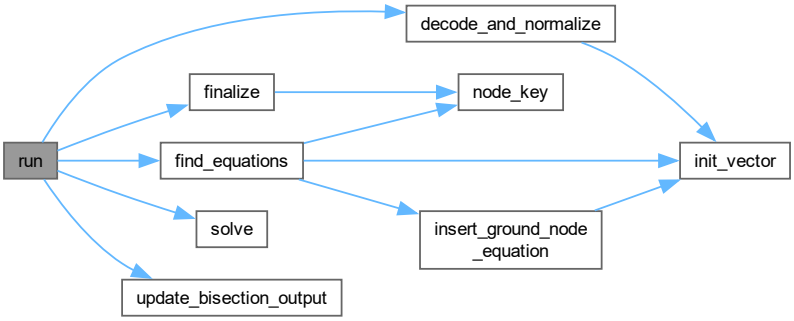
```
void run (
            Context & ctx,
            bool do_bisection)
```

Runs the main circuit analysis procedure.
Finds all necessary nodal and source equations, solves the resulting system of linear equations and decodes the results into node potentials and source currents. If bisection is enabled, it updates the output value for the bisection algorithm.

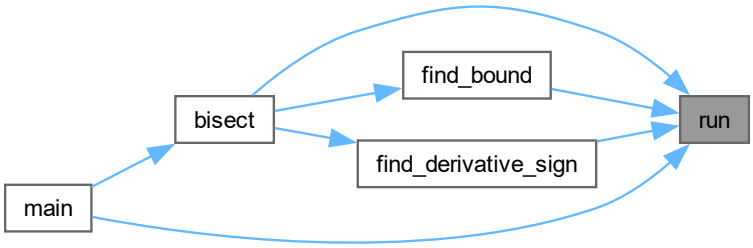
Parameters

ctx	The Context object containing all circuit data.
do_bisection	Boolean flag indicating whether to perform bisection updates.

Here is the call graph for this function:



Here is the caller graph for this function:



9.22.1.5 solve()

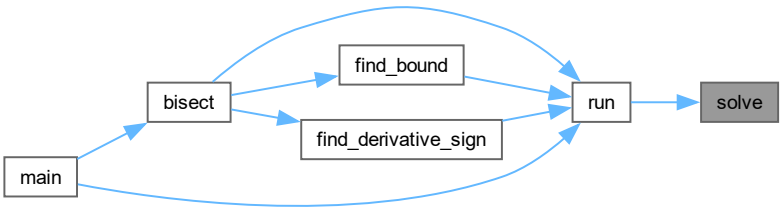
```
void solve (
            Context & ctx)
```

Solves the system of linear equations stored in the context using Gaussian elimination with partial pivoting.
The equations are represented as an augmented matrix stored in `ctx.equations`. After solving, the matrix is in reduced row echelon form. Throws a std::runtime_error if contradictory equations are found.

Parameters

ctx	The Context object containing all circuit data.

Here is the caller graph for this function:



9.23 src/util.cpp File Reference

```
#include "util.hpp"
#include <cmath>
#include "element.hpp"
#include <cstdint>
#include <vector>
#include "constants.hpp"
```

Functions

- uint8_t get_number_of_unknowns (const std::vector< Element > &elements, const uint8_t max_node)

  Calculates the total number of unknowns in the MNA system.
- void init_vector (std::vector< double > &vector, const uint8_t n)

  Initializes a vector of doubles with a specified number of zeros.
- void insert_ground_node_equation (std::vector< std::vector< double > > &equations, const uint8_t unknowns)

  Inserts the equation for the ground node (Node 1) into the system.
- uint16_t node_key (const uint8_t node_0, const uint8_t node_1)

  Creates a unique 16-bit key for a node pair, independent of the order.
- bool current_source_only_node (const std::vector< Element > &elements, const uint8_t max_node)

  Checks if there is a node connected only to current sources.
- bool parallel_voltage_sources (const std::vector< Element > &elements)

  Checks for voltage sources connected in parallel.
- double avg (double a, double b)

  Calculates the average (arithmetic mean) of two double-precision numbers.
- double round_to_prec (double value)

  Rounds a double-precision value to the specified number of significant digits.

9.23.1 Function Documentation

9.23.1.1 avg()

```
double avg (
            double a,
            double b)
```

Calculates the average (arithmetic mean) of two double-precision numbers.

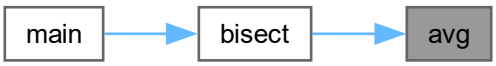
Parameters

a	The first number.
b	The second number.

Returns

double The average of a and b.

Here is the caller graph for this function:



9.23.1.2 current_source_only_node()

```
bool current_source_only_node (
            const std::vector< Element > & elements,
            uint8_t max_node)
```

Checks if there is a node connected only to current sources.
This condition makes the node's voltage indeterminate, as only KCL equations (which govern current) apply, not Ohm's law (which relates voltage and current).

Parameters

elements	A constant reference to the vector of circuit elements.
max_node	The highest node index.

Returns

bool True if such a node exists, false otherwise.

Here is the caller graph for this function:

9.23.1.3 get_number_of_unknowns()

```
uint8_t get_number_of_unknowns (
            const std::vector< Element > & elements,
            uint8_t max_node)
```

Calculates the total number of unknowns in the MNA system.
The number of unknowns is equal to the maximum node index (for node potentials) plus the number of voltage sources (for voltage source currents).

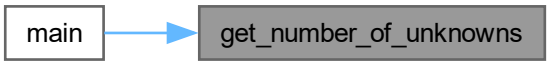
Parameters

elements	A constant reference to the vector of circuit elements.
max_node	The highest node index in the circuit.

Returns

uint8_t The total number of unknowns.

Here is the caller graph for this function:



9.23.1.4 init_vector()

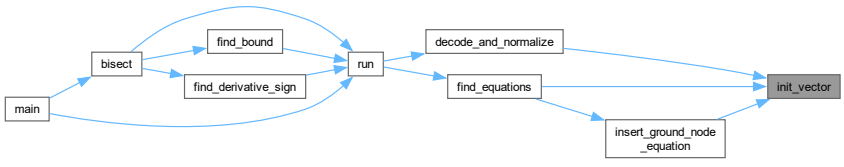
```
void init_vector (
            std::vector< double > & vector,
            uint8_t n)
```

Initializes a vector of doubles with a specified number of zeros.

Parameters

vector	The vector to be initialized.
n	The number of zeros to append.

Here is the caller graph for this function:



9.23.1.5 insert_ground_node_equation()

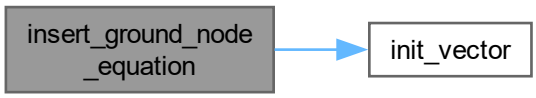
```
void insert_ground_node_equation (
            std::vector< std::vector< double > > & equations,
            uint8_t unknowns)
```

Inserts the equation for the ground node (Node 1) into the system.

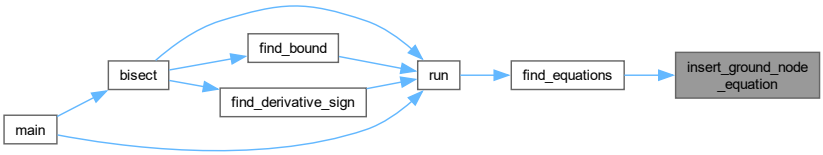
Parameters

equations	The vector of equation rows (augmented matrix).
unknowns	The total number of unknowns in the system.

Here is the call graph for this function:



Here is the caller graph for this function:



9.23.1.6 node_key()

```
uint16_t node_key (
            uint8_t node_0,
            uint8_t node_1)
```

Creates a unique 16-bit key for a node pair, independent of the order.
The key is constructed by shifting the greater node to the upper 8 bits and the smaller node to the lower 8 bits.

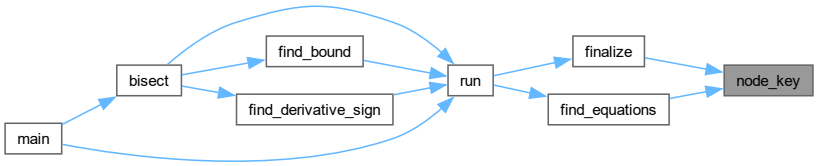
Parameters

node↩ _0	The first node index.
node↩ _1	The second node index.

Returns

uint16_t The unique node key.

Here is the caller graph for this function:



9.23.1.7 parallel_voltage_sources()

```
bool parallel_voltage_sources (
            const std::vector< Element > & elements)
```

Checks for voltage sources connected in parallel.
Parallel voltage sources (connected to the same two nodes) lead to an indeterminate current in the branches, making the system unsolvable.

Parameters

elements	A constant reference to the vector of circuit elements.

Returns

bool True if parallel voltage sources are found, false otherwise.

Here is the caller graph for this function:

9.23.1.8 round_to_prec()

```
double round_to_prec (
            double value)
```

Rounds a double-precision value to the specified number of significant digits.
Uses `constants::prec_factor()` to scale the value for rounding. Special handling to ensure negative zero is not returned.

Parameters

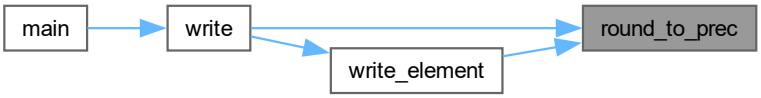
value	The value to be rounded.

Returns

double The rounded value.

Here is the call graph for this function:

Here is the caller graph for this function:



Index