# ALDCC simulator

1.0.0

Generated by Doxygen 1.16.1

# Chapter 1

# Directory Hierarchy

## 1.1 Directories

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# Chapter 5

# Directory Documentation

## 5.1 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

## 5.2 src Directory Reference

### Files

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

# Chapter 6

# Namespace Documentation

## 6.1 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.*

### 6.1.1 Detailed Description

Contains functions and structures for command-line argument parsing.

### 6.1.2 Enumeration Type Documentation

#### 6.1.2.1 Switch

```
enum class cli::Switch  [strong]
```

Defines the valid command-line switches for the program.

**Enumerator**

| | |
|---|---|
| none | |
| in | |
| out | |
| prec | |

### 6.1.3 Function Documentation

#### 6.1.3.1 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.

## 6.2 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

## 6.2.1 Detailed Description

Global constants and configuration variables for the circuit solver.

## 6.2.2 Function Documentation

### 6.2.2.1 prec_factor()

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

## 6.2.3 Variable Documentation

### 6.2.3.1 BISECTION_PREC

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

### 6.2.3.2 EPSILON

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

### 6.2.3.3 PRECISION

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

### 6.2.3.4 ZERO_TRESHOLD

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

# Chapter 7

# Class Documentation

## 7.1 Context Struct Reference

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

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

#### Public Attributes

- uint8_t max_node = 0
- uint8_t number_of_unknowns = 0
- std::vector< Element > elements
- 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

### 7.1.1 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.

### 7.1.2 Member Data Documentation

#### 7.1.2.1 elements

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

#### 7.1.2.2 equations

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

### 7.1.2.3 given_value

```
GivenValue Context::given_value
```

### 7.1.2.4 max_node

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

### 7.1.2.5 number_of_unknowns

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

### 7.1.2.6 output_value

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

### 7.1.2.7 v_source_currents_map

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

### 7.1.2.8 variable_elem

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

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

- include/context.hpp

## 7.2 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

### 7.2.1 Detailed Description

Represents a single circuit component and its solved parameters.

### 7.2.2 Member Function Documentation

#### 7.2.2.1 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.

#### 7.2.2.2 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.

#### 7.2.2.3 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.

### 7.2.3 Member Data Documentation

#### 7.2.3.1 current

```
double Element::current
```

#### 7.2.3.2 nodes

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

#### 7.2.3.3 power

```
double Element::power
```

#### 7.2.3.4 type

```
ElementType Element::type
```

#### 7.2.3.5 value

```
double Element::value
```

#### 7.2.3.6 voltage

```
double Element::voltage
```

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

- include/element.hpp

## 7.3 GivenValue Struct Reference

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

```
#include <element.hpp>
```

**Public Attributes**

- Element ∗ elem
- ValueType v_type
- double desired_value

### 7.3.1 Detailed Description

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

### 7.3.2 Member Data Documentation

#### 7.3.2.1 desired_value

```
double GivenValue::desired_value
```

#### 7.3.2.2 elem

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

#### 7.3.2.3 v_type

```
ValueType GivenValue::v_type
```

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

- include/element.hpp

## 7.4 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

### 7.4.1 Detailed Description

Holds the parsed command-line input values.

### 7.4.2 Member Data Documentation

#### 7.4.2.1 ifname

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

#### 7.4.2.2 ofname

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

#### 7.4.2.3 prec

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

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

- include/cli.hpp

# Chapter 8

# File Documentation

## 8.1 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.*

### 8.1.1 Function Documentation

#### 8.1.1.1 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. |

#### 8.1.1.2 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.

#### 8.1.1.3 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.

#### 8.1.1.4 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. |

## 8.2 bisect.hpp

Go to the documentation of this file.

```
#pragma once
#include "context.hpp"

void update_bisection_output(Context& ctx);

int8_t find_derivative_sign(Context& ctx);

double find_bound(Context& ctx, int8_t derivative_sign);

void bisect(Context& ctx);
```

## 8.3 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.*

## 8.4 cli.hpp

Go to the documentation of this file.

```
#pragma once
#include <cstdint>
#include <string>
namespace cli {

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

    struct Input {
        std::string ifname = "";
        std::string ofname = "";
        uint8_t prec = 4;
    };

    Input parse(int argc, char* argv[]);
}
```

## 8.5 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

## 8.6 constants.hpp

Go to the documentation of this file.

```
#pragma once
#include "util.hpp"

namespace constants {
    inline uint32_t PRECISION = 4;
    inline double prec_factor() { return power(10, PRECISION); }
    constexpr double ZERO_TRESHOLD = 1e-12;
    constexpr double EPSILON = 2e-10;
    constexpr uint32_t BISECTION_PREC = 50;
}
```

## 8.7 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.*

## 8.8 context.hpp

Go to the documentation of this file.

```
#pragma once
#include <cstdint>
#include <unordered_map>
#include <vector>

#include "element.hpp"

struct Context {
    uint8_t max_node = 0;
    uint8_t number_of_unknowns = 0;
    std::vector<Element> elements;

    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;
};
```

## 8.9 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.*

### 8.9.1 Enumeration Type Documentation

#### 8.9.1.1 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 | |

#### 8.9.1.2 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 | |

## 8.10 element.hpp

Go to the documentation of this file.

```
#pragma once
#include <array>
#include <cstdint>

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

struct Element {
    ElementType type;
    std::array<uint8_t, 2> nodes;
    double value;

    double voltage;
    double current;
    double power;

    bool is_directed_outwards(const uint8_t node) const {
        return nodes[0] == node;
    }

    int8_t sign_of_current(const uint8_t node) const {
        return static_cast<int8_t>(2 * is_directed_outwards(node) - 1);    // 1 if current leaves, -1
when enters
```

```
    }

    uint8_t other_node(const uint8_t node) const {
        return nodes[0] == node? nodes[1] : nodes[0];
    }
};

enum class ValueType {
    voltage = 'U',
    current = 'I',
    power = 'P'
};

struct GivenValue {
    Element* elem;
    ValueType v_type;
    double desired_value;
};
```

## 8.11 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::vector< double > &potentials, const std::string &filename, bool do_bisection)

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

### 8.11.1 Function Documentation

#### 8.11.1.1 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.

#### 8.11.1.2 write()

```
void write (
            const Context & ctx,
            const std::vector< double > & potentials,
            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. |
| *potentials* | The vector of solved node potentials. |
| *filename* | The name of the output file. |
| *do_bisection* | Boolean flag indicating if bisection was performed. |

#### 8.11.1.3 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. |

## 8.12 file_io.hpp

Go to the documentation of this file.

```
#pragma once
#include <string>

#include "context.hpp"

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

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

void write(const Context& ctx, const std::vector<double>& potentials, const std::string& filename,
      bool do_bisection);
```

## 8.13 include/solver.hpp File Reference

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

### Functions

- std::vector< double > 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.*

## 8.13.1 Function Documentation

### 8.13.1.1 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.

### 8.13.1.2 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. |

#### 8.13.1.3 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. |

#### 8.13.1.4 run()

```
std::vector< double > 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. |

**Returns**

std::vector<double> A vector of solved node potentials.

#### 8.13.1.5 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. |

## 8.14 solver.hpp

Go to the documentation of this file.

```
#pragma once
#include "context.hpp"

std::vector<double> run(Context& ctx, bool do_bisection);

void find_equations(Context& ctx);

void solve(Context& ctx);

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

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

## 8.15 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.*

### 8.15.1 Function Documentation

#### 8.15.1.1 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.

#### 8.15.1.2 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.

#### 8.15.1.3 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.

#### 8.15.1.4 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. |

#### 8.15.1.5 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. |

#### 8.15.1.6 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.

#### 8.15.1.7 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.

#### 8.15.1.8 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.

#### 8.15.1.9 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.

## 8.16 util.hpp

Go to the documentation of this file.

```
#pragma once
#include <cstdint>
#include <vector>

#include "element.hpp"

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

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

void insert_ground_node_equation(std::vector<std::vector<double»& equations, uint8_t unknowns);

uint16_t node_key(uint8_t node_0, uint8_t node_1);

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

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

double avg(double a, double b);

constexpr double power(double base, int exp) {
    double result = 1.0;
    while (exp-- > 0) result *= base;
    return result;
}

double round_to_prec(double value);
```

## 8.17 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.*

### 8.17.1 Function Documentation

#### 8.17.1.1 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. |

#### 8.17.1.2 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.

#### 8.17.1.3 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.

#### 8.17.1.4 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. |

## 8.18 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.*

## 8.19 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::vector< double > &potentials, const std::string &filename, bool do_bisection)

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

### 8.19.1 Function Documentation

#### 8.19.1.1 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.

#### 8.19.1.2 write()

```
void write (
            const Context & ctx,
            const std::vector< double > & potentials,
            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. |
| *potentials* | The vector of solved node potentials. |
| *filename* | The name of the output file. |
| *do_bisection* | Boolean flag indicating if bisection was performed. |

#### 8.19.1.3 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. |

## 8.20 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[ ])

### 8.20.1 Function Documentation

#### 8.20.1.1 main()

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

## 8.21 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**

- std::vector< double > 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.*

### 8.21.1 Function Documentation

#### 8.21.1.1 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.

#### 8.21.1.2 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. |

#### 8.21.1.3 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. |

#### 8.21.1.4 run()

```
std::vector< double > 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. |

**Returns**

std::vector<double> A vector of solved node potentials.

#### 8.21.1.5 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. |

## 8.22 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.*

## 8.22.1 Function Documentation

### 8.22.1.1 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.

#### 8.22.1.2 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.

#### 8.22.1.3 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.

#### 8.22.1.4 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. |

#### 8.22.1.5 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. |

#### 8.22.1.6 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.

#### 8.22.1.7 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.

#### 8.22.1.8 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.

# Index