ΠΑΝΕΠΙΣΤΗΜΙΟ
ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

Τίτλος διπλωματικής

Διπλωματική εργασία

Ονοματεπώνυμο φοιτητή

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Όνομα εργαστηρίου

Πολυτεχνική Σχολή

Επιβλέπων:

Ημερομηνία

UNIVERSITY OF
WESTERN MACEDONIA

Thesis title

Diploma Thesis

Student name

Department of Electrical and Computer Engineering

Laboratory name

Faculty of Engineering

Advisor:

Date

Περίληψη

Κείμενο περίληψης
Λέξεις κλειδιά:

Abstract

Abstract text
Keywords:

Ευχαριστίες

Ευχαριστίες.

Acknowledgements

Your acknowledgements text is here.

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

Κατάλογος Εικόνων

Κατάλογος Αλγορίθμων

Περιεχόμενα

Ακρωνύμια

AI Artificial Intelligence. 8

CPU Central Processing Unit. 8, 9

LP Linear Programming. 8

Κεφάλαιο 1

Τίτλος κεφαλαίου

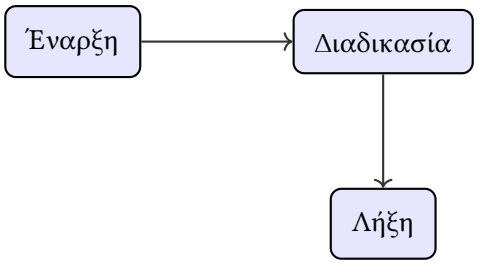
1.1 Τίτλος

Θεώρημα 1.1 (Όνομα θεωρήματος)**.** *Περιγραφή*

Δοκιμαστικό κείμενο Central Processing Unit (CPU).

Λήμα 1.2. *Για κάθε* $x \in \mathbb{R}$*, ισχύει* $x^2 \geq 0$*.*

1.2 Τίτλος



Εικόνα 1.2.1: Σύντομη περιγραφή

Κεφάλαιο 2

Τίτλος κεφαλαίου

2.1 Τίτλος

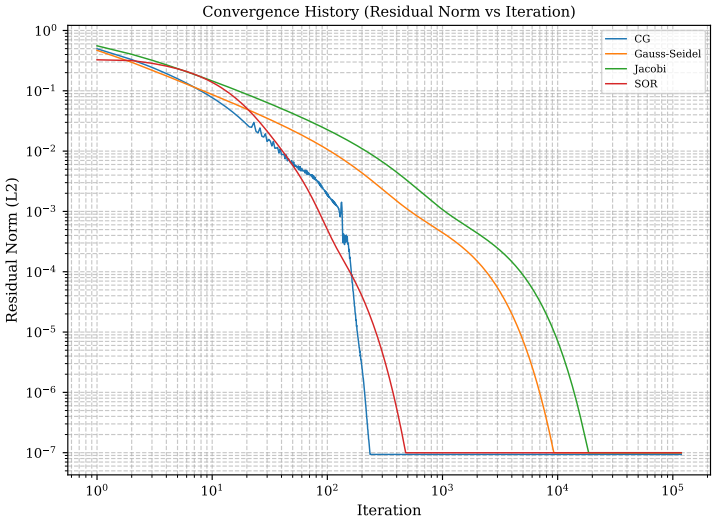
Παράδειγμα αναφοράς: [1]. Παράδειγμα αναφοράς: [2].

Αλγόριθμος 2.1 – Όνομα αλγορίθμου - Algorithm name

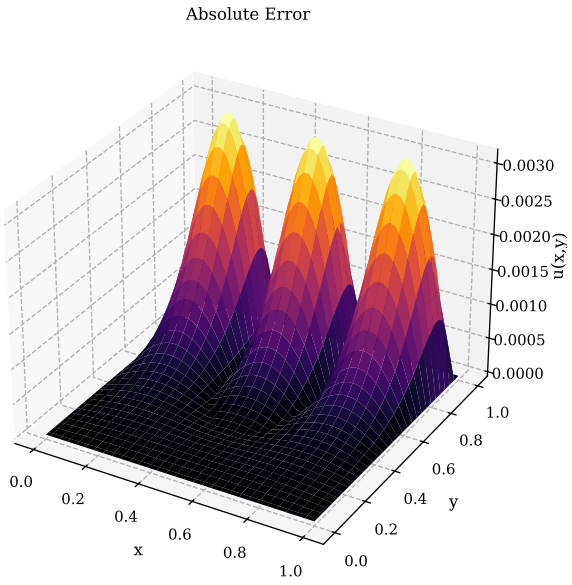
Require: Implication graph and a conflict at the current decision level
Ensure: Learned clause and backjump level

```
1:  function ANALYZE_CONFLICT
2:      if current_decision_level() == 0 then
3:          return −1
4:      end if
5:      cl ← find_conflicting_clause()
6:      while not stop_criterion_met(cl) do
7:          lit ← choose_literal(cl)
8:          var ← variable_of_literal(lit)
9:          ante ← antecedent(var)
10:         cl ← resolve(cl, ante, var)
11:     end while
12:     add_clause_to_database(cl)
13:     back_dl ← clause_asserting_level(cl)
14:     return back_dl
15: end function
```

2.2 Τίτλος



Σχήμα 2.2.1: Σύντομη περιγραφή



Εικόνα 2.2.1: Σύντομη περιγραφή

N	CG	Gauss-Seidel	Jacobi	SOR
0	8	14	64	35
1	18	39	324	62
2	28	64	784	87
3	38	90	1444	112
4	48	114	2304	136
5	58	139	3364	161
6	68	163	4624	225
7	78	187	6084	304
8	88	211	7614	388
9	98	235	9260	479
10	108	258	11048	577
11	118	282	12977	682
12	128	305	15043	794
13	138	328	17244	913
14	148	351	19577	1039
15	158	374	22041	1172
16	168	398	24633	1311
17	178	421	27352	1457
18	188	444	30196	1609
19	198	467	33163	1768

Πίνακας 2.2.1: Σύντομη περιγραφή

Παράρτημα Α΄

Κώδικας

```python
def hello(name):
    print("Hello", name)
```

```cpp
#ifndef SOR_HPP
#define SOR_HPP

#include "utils/SolverLog.hpp"
#include "solvers/config.h"
template<typename Vector>
struct SOR
{
    using Scalar       = typename Vector::Scalar;
    using SparseMatrix = Eigen::SparseMatrix<Scalar, Eigen::RowMajor>;

    double              tol       = DEFAULT_TOL;
    int                 max_iters = MAX_ITERS;
    double              omega;
    std::string         name      = "SOR";
    SolverLog<Vector>   log;
    Vector              final_solution;

    template<typename System>
```

```
20      SOR (System system) : omega(system.omega_)
21      {
22          log.system_dim     = system.A.rows();
23          max_iters          = static_cast<int>(10 *
            ↪  std::sqrt(log.system_dim));
24          log.max_iterations = max_iters;
25          log.tolerance      = tol;
26          log.solver_name    = name;
27      }
28
29      template<typename System>
30      void solve(System& system)
31      {
32          const auto& A         = system.A;
33          const auto& b         = system.b;
34                auto& u         = system.u;
35
36          std::cout << "max_iters: " << max_iters << '\n';
37
38          double sum1, sum2;
39
40          double b_norm    = b.norm();
41          double res       = (A * u - b).norm() / b_norm;
42
43          if (res <= tol)
44          {
45              this->final_solution = u;
46              log.final_solution   = this->final_solution;
47              log.converged = 1;
48              return;
49          }
50
51          Vector inv_diag = A.diagonal().cwiseInverse();
52
53          for (int k = 0; k < max_iters; k++)
54          {
55              for (int i = 0; i < A.rows(); ++i)
56              {
57                  double sum = 0;
```

```
58
59                  for (typename SparseMatrix::InnerIterator it(A, i);
                    ↪  it; ++it)
60                  {
61                      int j = it.col();
62
63                      if (j != i)
64                      {
65                          sum += it.value() * u[j];
66                      }
67                  }
68                  u[i] = (1 - omega) * u[i] + omega * (inv_diag[i] * (b[i]
                    ↪  - sum));
69              }
70
71              res = (A * u - b).norm() / b.norm();
72
73              log.num_of_iterations++;
74              log.res_per_iteration.push_back(res);
75
76              if (res <= tol)
77              {
78                  this->final_solution = u;
79                  log.final_solution   = this->final_solution;
80                  log.converged = 1;
81                  return;
82              }
83          }
84          this->final_solution = u;
85          log.final_solution   = this->final_solution;
86          return;
87      }
88  };
89
90
91  #endif // SOR_HPP
```

```cpp
#ifndef CONJUGATE_GRADIENT_HPP
#define CONJUGATE_GRADIENT_HPP

#include "utils/SolverLog.hpp"
#include "solvers/config.h"
template< typename Vector>
struct ConjugateGradient
{
    double              tol        = DEFAULT_TOL;
    int                 max_iters = 1e6;
    std::string         name       = "CG";
    SolverLog<Vector>   log;
    Vector              final_solution;

    ConjugateGradient ()
    {
        log.tolerance      = tol;
        log.max_iterations = max_iters;
        log.solver_name    = name;
    }

    template<typename System>
    void solve(System& system)
    {
        const auto&  A = system.A;
        const auto&  b = system.b;
        auto&        u = system.u;
        log.system_dim = A.rows();

        std::cout << "max_iters: " << max_iters << '\n';

        Vector r      = b - A * u; // initial residual
        double b_norm = b.norm();
        double r_norm = r.norm();

        if (r_norm / b_norm <= tol)
        {
            this->final_solution = u;
```

```
39              log.final_solution   = this->final_solution;
40              log.converged        = 1;
41              return;
42          }
43
44          Vector d = r; // initial search direction
45          Vector Ad(A.rows());
46
47          for (int k = 0; k < max_iters; k++)
48          {
49              // std::cout << "---------------------- iter. " << k+1 << "
                ↪ ----------------------\n";
50              Ad.noalias() = A * d;
51
52              double alpha      = ((r.transpose() * r) / (d.transpose() *
                ↪ Ad)).coeff(0); // step size
53              double r_prev_dot = (r.transpose() * r).coeff(0); // to
                ↪ calculate beta
54
55              u.noalias() += alpha * d;
56              r.noalias() -= alpha * Ad;
57
58              r_norm = r.norm();
59
60              log.num_of_iterations++;
61              log.res_per_iteration.push_back(r_norm / b_norm);
62
63              if (r_norm / b_norm <= tol)
64              {
65                  log.converged        = 1;
66                  this->final_solution = u;
67                  log.final_solution   = this->final_solution;
68                  return;
69              }
70
71              double beta = r.dot(r) / r_prev_dot;
72              d           = r + beta * d; // update direction
73          }
74          this->final_solution = u;
```

```
75          log.final_solution    = this->final_solution;
76          return;
77      }
78  };
79
80
81  #endif // CONJUGATE_GRADIENT_HPP
```

Βιβλιογραφία

[1] V. Balabanov and J.-H. R. Jiang, "Unified qbf certification and its applications," *Form. Methods Syst. Des.*, vol. 41, p. 45–65, Aug. 2012.

[2] Γιάννης Παπαδόπουλος, *Μαθηματικά για Επιστήμονες Υπολογιστών*. Εκδόσεις Πανεπιστημίου Μακεδονίας, 2020.