Listing 1: Experiment

```

//
// ExpiFramework
// Excperiment.cs
//
// Copyright © 2022 Nikolai Tiunin. All rights reserved.
//

using NecProblemFramework;
using PAASolveFramework;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Numerics;

namespace Expi
{
    class Experiment
    {
        string nec;
        string package;
        string solver;
        double theta;
        double phi;
        string name;
        string compare;
        bool draw;
        NecUtilities.In.Parser parser = new NecUtilities.In.Parser();
        NecProblemBuilder problemBuilder = new NecProblemBuilder();
        ProblemWritter problemWritter = new ProblemWritter();
        Dictionary<string, object> parameters;

        string[] PreferredOutputOrder = new string[] {
            "Nec", "Theta", "Phi", "p", "q", "force", "z", "de_status", "
                de_record_iteration", "de_time", "grad_time"
        };

        public class PackageResult
        {
            public string Path;
            public OptimizationResult Result;

            public PackageResult(string Path, OptimizationResult Result)
            {
                this.Path = Path;
                this.Result = Result;
            }
        }

        public class PackageResolver
        {
```

```
52              string directory;
53              string name;
54              NecUtilities.In.Model model;
55              Problem problem;
56              ProblemWritter problemWritter = new ProblemWritter();
57              public PackageResolver(string directory, NecUtilities.In.Model
                     model, Problem problem)
58              {
59                  this.directory = directory;
60                  this.model = model;
61                  this.problem = problem;
62              }
63
64              public PackageResult Resolve(string package, string solver,
                    Dictionary<string, object> parameters)
65              {
66                  string solvedNec = null;
67                  var result = UsePackage(package, solver, parameters, ref
                        solvedNec);
68                  var writter = new OptimizationResultWritter();
69                  writter.Write(result, $"{directory}\\solution.txt");
70                  WriteSolution(result.GetComplexSolution(), solvedNec);
71                  var packageResult = new PackageResult(solvedNec, result);
72                  return packageResult;
73              }
74
75              private OptimizationResult UsePackage(string package, string
                    solver, Dictionary<string, object> parameters, ref string
                    solvedNec)
76              {
77                  switch (package.ToLower())
78                  {
79                      case "gams":
80                          return UseGamsPackage(solver, parameters, ref
                                solvedNec);
81                      default:
82                          return UseCustomPackage(package, solver,
                                parameters, ref solvedNec);
83                  }
84              }
85
86              public void WriteSolution(Complex[] solution, string fileName)
87              {
88                  if (solution == null)
89                  {
90                      return;
91                  }
92                  var clone = model.Copy();
93                  var sources = clone.Sources;
94                  for (var i = 0; i < sources.Length; i++)
95                  {
96                      sources[i].Value = solution[i];
97                  }
98                  File.WriteAllText(fileName, clone.ToString());
```

```
99              }
100
101             private OptimizationResult UseGamsPackage(string solver,
                    Dictionary<string, object> parameters, ref string
                    solvedNec)
102             {
103                 solvedNec = $"{directory}\\solved_by_gms.nec";
104                 var reslim = 1000;
105                 if (parameters.ContainsKey("reslim"))
106                 {
107                     reslim = (int)parameters["reslim"];
108                 }
109
110                 var input = $"{directory}\\{name}.gms";
111                 var gamsWritter = new GAMSWritter();
112                 gamsWritter.Write(problem, input, problem.analyticSolution
                        , reslim);
113                 var gams = new GAMSSolver(solver);
114                 var result = gams.Solve(input);
115                 return result;
116             }
117
118             private void WriteGams()
119             {
120                 var input = $"{directory}\\problem.gms";
121                 var gamsWritter = new GAMSWritter();
122                 var reslim = 1000;
123                 gamsWritter.Write(problem, input, problem.analyticSolution
                        , reslim);
124             }
125
126             private OptimizationResult UseCustomPackage(string packageName
                    , string solver, Dictionary<string, object> parameters,
                    ref string solvedNec)
127             {
128                 solvedNec = $"{directory}\\solved_by_{packageName}_{solver
                        }.nec";
129                 var package = new CustomSolver(packageName, solver,
                        parameters);
130                 var problemFile = $"{directory}\\problem.dat";
131                 problemWritter.WriteBin(problem, problemFile);
132                 WriteGams();
133                 var result = package.Solve(problemFile);
134                 return result;
135             }
136         }
137
138         public Experiment(string nec, string package, string solver,
                double theta, double phi, string compare, bool draw,
                Dictionary<string, object> parameters)
139         {
140             this.nec = nec;
141             this.package = package;
142             this.solver = solver;
```

```
143             this.theta = theta;
144             this.phi = phi;
145             this.name = nec.Replace(".nec", "");
146             this.compare = compare;
147             this.parameters = parameters;
148             this.draw = draw;
149         }
150 
151         public void Solve(bool rerunNec)
152         {
153             var paaFolder = $"{name}_results";
154             Console.WriteLine(paaFolder);
155             var model = parser.Parse(nec);
156             var frequency = model.Frequency.frequency;
157             var frequesncyFolder = $"{paaFolder}\\{frequency}MHz";
158             var directionFolder = $"{frequesncyFolder}\\{theta}-{phi}";
159             Directory.CreateDirectory(directionFolder);
160             Console.WriteLine(directionFolder);
161             var analyticNec = $"{directionFolder}\\analytic.nec";
162             Console.WriteLine("Run nec");
163             var problem = problemBuilder.Build(model, theta, phi,
                    frequesncyFolder, rerunNec);
164             Utils.Symmetrize(problem);
165             ExtendingAnalyze(model, problem, $"{frequesncyFolder}\\report.
                    txt");
166             Console.WriteLine("Problem files ready");
167             WriteMatrix(problem.A, $"{directionFolder}\\A.txt", 10);
168             for (var i = 0; i < problem.B.Length; i++)
169             {
170                 WriteMatrix(problem.B[i], $"{directionFolder}\\B{i + 1}.
                        txt", 15);
171             }
172             var resolver = new PackageResolver(directionFolder, model,
                    problem);
173             resolver.WriteSolution(problem.analyticSolution, analyticNec);
174             var result = resolver.Resolve(package, solver, parameters);
175             if (draw)
176             {
177                 MakeDiagrams(result.Path, analyticNec, compare,
                        directionFolder, rerunNec, frequency);
178             }
179             CollectResult(result.Result);
180         }
181 
182         private void CollectResult(OptimizationResult result)
183         {
184             var filename = "batch_results.txt";
185             var text = " ";
186             var pars = ComposeOutputLine(result);
187             if (File.Exists(filename) == false)
188             {
189                 foreach(var par in pars)
190                 {
191                     text += $"\t{par.Key}";
```

```
192                 }
193                 text += "\n ";
194             }
195             foreach (var par in pars)
196             {
197                 text += $"\t{HumanReadableValue(par)}";
198             }
199             text += "\n";
200             File.AppendAllText(filename, text);
201         }
202 
203         private string HumanReadableValue(KeyValuePair<string, object>
                pair)
204         {
205             if (pair.Value is string)
206             {
207                 return $"\"{pair.Value}\"";
208             }
209             if (pair.Value is double)
210             {
211                 return $"{pair.Value}".Replace(",", ".");
212             }
213             if (pair.Value is int)
214             {
215                 if (pair.Key == "de_status")
216                 {
217                     switch (pair.Value)
218                     {
219                         case 0:
220                             return "Normal Completion";
221                         case 1:
222                             return "Iterations Limit";
223                         case 2:
224                             return "Record estimation";
225                         default:
226                             return "Unknown";
227                     }
228                 }
229             }
230             if (pair.Value is double[])
231             {
232                 var array = (double[])pair.Value;
233                 var text = "";
234                 foreach (var value in array)
235                 {
236                     text += $"{value} ".Replace(",", ".");
237                 }
238                 return text;
239             }
240             return $"{pair.Value}";
241         }
242 
243         private KeyValuePair<string, object>[] ComposeOutputLine(
                OptimizationResult result)
```

```
244         {
245             var dictionary = new Dictionary<string, object>();
246             foreach (var pair in parameters)
247             {
248                 dictionary[pair.Key] = pair.Value;
249             }
250             foreach(var pair in result.output)
251             {
252                 dictionary[pair.Key] = pair.Value;
253             }
254             dictionary["z"] = -result.z;
255             dictionary["Nec"] = nec;
256             dictionary["Theta"] = theta;
257             dictionary["Phi"] = phi;
258             var pairs = dictionary.ToArray();
259             Array.Sort(pairs, (lhs, rhs) => {
260                 var res = GetPriority(lhs.Key) - GetPriority(rhs.Key);
261                 if (res == 0)
262                 {
263                     return lhs.Key.CompareTo(rhs.Key);
264                 }
265                 return res;
266             });
267             return pairs;
268         }
269 
270         private int GetPriority(string key)
271         {
272             var index = Array.IndexOf(PreferredOutputOrder, key);
273             if (index < 0)
274             {
275                 return PreferredOutputOrder.Length;
276             }
277             return index;
278         }
279 
280         private void ExtendingAnalyze(NecUtilities.In.Model model, Problem
                problem, string reportFileName)
281         {
282             var file = new StreamWriter(reportFileName);
283             var segs = 0;
284             foreach (var wire in model.Wires)
285             {
286                 segs += wire.seg;
287             }
288             if (segs > 11000)
289             {
290                 file.WriteLine($"                    : {segs}.");
291             }
292             else
293             {
294                 file.WriteLine($"                     : {segs}.");
295             }
296             Utils.Symmetrize(problem);
```

```
297             var Bsum = problem.Bsum;
298             double[][] grid = new double[Bsum.Height / 2][];
299             for (var i = 0; i < Bsum.Height / 2; i++)
300             {
301                 grid[i] = new double[Bsum.Width / 2];
302                 for (var j = 0; j < Bsum.Width / 2; j++)
303                 {
304                     grid[i][j] = Bsum[i, j];
305                 }
306             }
307             var eigenValues = Utils.EigenValues(Bsum, 50);
308             if (eigenValues.Has((value) => value < 0))
309             {
310                 eigenValues = Utils.EigenValues(Bsum, 100);
311             }
312             if (eigenValues.Has((value) => value < 0))
313             {
314                 eigenValues = Utils.EigenValues(Bsum, 1000);
315             }
316             if (eigenValues.Has((value) => value < 0))
317             {
318                 file.WriteLine("    $B_{\\Sigma}$                    .
                                                          .");
319                 file.Close();
320                 return;
321             }
322             var lamdaMin = eigenValues.First;
323             var lamdaMax = eigenValues.Last;
324 
325             file.WriteLine("    $B_{\\Sigma}$
                                                  :\\\\ " +
326                 "$\\lambda_{max} = " + $"{lamdaMax}".Replace(",", ".") + "
                        $, \\\\" +
327                 "$\\lambda_{min} = " + $"{lamdaMin}".Replace(",", ".") + "
                        $.\\\\");
328 
329             var cond = lamdaMax / lamdaMin;
330 
331             file.WriteLine("                $B_{\\Sigma}     " + $"{cond}".
                    Replace(",", ".") + "$.");
332             var radius = Math.Sqrt(problem.A.Width / 2 / lamdaMin);
333             file.WriteLine("                    $r = " + $"{radius}".Replace
                    (",", ".") + "$.");
334             file.Close();
335             problem.Radius = radius;
336         }
337 
338         private void WriteMatrix(Matrix M, string fileName, int itemWidth)
339         {
340             var file = new StreamWriter(fileName);
341             for (var i = 0; i < M.Height; i++)
342             {
343                 for (var j = 0; j < M.Width; j++)
344                 {
```

```
                    file.Write($"{MatrixFormat(M[i, j], itemWidth)} ");
                }
                file.WriteLine();
            }
            file.Close();
        }

        private string MatrixFormat(double value, int width)
        {
            var radix = Math.Pow(10, width);
            var rounded = ((long)(value * radix)) / (1.0 * radix);
            var str = $"{rounded}";
            if (str.Length > width)
            {
                return $"{str.Substring(0, width - 3)}...";
            }
            while (str.Length < width)
            {
                str += " ";
            }
            return str;
        }

        private void WriteSolution(NecUtilities.In.Model model, Complex[]
            solution, string fileName)
        {
            if (solution == null) {
                return;
            }
            var clone = model.Copy();
            var sources = clone.Sources;
            for (var i = 0; i < sources.Length; i++)
            {
                sources[i].Value = solution[i];
            }
            File.WriteAllText(fileName, clone.ToString());
        }

        private OptimizationResult SolveWithGams(Problem problem, string
            folder, int reslim = 1000)
        {
            var input = $"{folder}\\{name}.gms";
            var gamsWritter = new GAMSWritter();
            gamsWritter.Write(problem, input, problem.analyticSolution,
                reslim);
            var solver = new GAMSSolver(this.solver);
            var writter = new OptimizationResultWritter();
            var result = solver.Solve(input);
            writter.Write(result, $"{folder}\\solution.txt");
            return result;
        }

        private void MakeDiagrams(string solvedNec, string analyticNec,
            string compareNec, string folder, bool rerunNec, double
```

```
            frequency)
395         {
396             var inputsList = new List<PatternInput>();
397             if (compareNec != null && compareNec.Length > 0)
398             {
399                 inputsList.Add(PatternInput("Single", "#909090", "2px",
                        compareNec, frequency));
400             }
401             var verticalPlaneTool = new VerticalPlanSVGTool();
402             var horisontalPlaneTool = new HorisontalPlanSVGTool();
403             inputsList.Add(PatternInput("Analytic", "#606060", "3px",
                    analyticNec, frequency));
404             inputsList.Add(PatternInput("PAA", "#000000", "3px", solvedNec
                    , frequency));
405             var inputs = inputsList.ToArray();
406             var verticalPatternsInput = new VerticalPlanPatternsInput(phi,
                     -90, 90, 1, "", inputs);
407             var horisontalPatternsInput = new HorisontalPlanPatternsInput(
                    theta, 0, 359, 1, "", inputs);
408             verticalPlaneTool.VerticalPlane(
409                 $"vertical_plane.svg",
410                 verticalPatternsInput,
411                 $"{folder}\\vertical_plane.svg"
412             );
413             horisontalPlaneTool.HorisontalPlane(
414                 $"horisontal_plane.svg",
415                 horisontalPatternsInput,
416                 $"{folder}\\horisontal_plane.svg"
417             );
418         }
419
420         private PatternInput PatternInput(string caption, string color,
                string width, string nec, double frequency)
421         {
422             var input = nec;
423             PrepareForDiagram(input, input, frequency);
424             var output = nec.Replace(".nec", ".out");
425             var pattern = Utils.BeamPattern(input, output, true);
426             return new PatternInput(pattern, caption, color, width);
427         }
428
429         private void PrepareForDiagram(string input, string output, double
                 frequency)
430         {
431             var model = new NecUtilities.In.Parser().Parse(input);
432             var rp = model.RadiationPattern;
433             rp.Phi0 = 0;
434             rp.PhiNumber = 360;
435             rp.PhiInc = 1;
436             rp.ThetaNumber = 91;
437             rp.Theta0 = 0;
438             rp.ThetaInc = 1;
439             model.Frequency.frequency = frequency;
440             var file = new StreamWriter(output);
```

```
441                 file.Write(model.ToString());
442                 file.Close();
443             }
444         }
445 }
```