

1 The Component Structure 2 of a Self-Adapting Numerical Software 3 System

4 Victor Eijkhout, Erika Fuentes,¹ Thomas Eidson,² and
5 Jack Dongarra¹

Received ■; revised ■; accepted ■

6 Self-Adapting Numerical Software (SANS) systems aim to automate some of
7 the laborious human decision making involved in adapting numerical algo-
8 rithms to problem data, network conditions, and computational platform. In
9 this paper we describe the structure of a SANS system that tackles auto-
10 matic algorithm choice, based on dynamic inspection of the problem data.
11 We describe the various components of such a system, and their interfaces.

12 **KEY WORDS:** Linear system solving; component frameworks; adaptive sys-
13 tems.

14 1. INTRODUCTION

15 The process of arriving at an efficient numerical solution of problems in
16 applied physics, chemistry, etc., involves numerous decision by a numeri-
17 cal expert. Attempts to automate such decisions (see Ref. 1 for a recent
18 overview) distinguish three levels:

- 19 • algorithmic decision;
20 • management of the parallel environment;
21 • processor-specific tuning of kernels.

22 This paper addresses the top level, where algorithm choices are made
23 dynamically based on the problem data. We describe the architecture of

¹ University of Tennessee, Knoxville TN, USA.

² Old Dominion University, Norfolk, VA, USA.

24 such a Self-Adapting Numerical Software (SANS) system for algorithm
25 choice, paying particular attention to the formalization of various inter-
26 faces between modules in the system. We will not go into the modeling
27 techniques that build up the heuristics of the ‘intelligence’ of the system.
28 An introduction to this subject can be found in Ref. 1.

29 **2. SYSTEM COMPONENTS**

30 A SANS system has the following large scale building blocks:

- 31 • Application,
- 32 • Analysis Modules,
- 33 • Intelligent switch,
- 34 • Numerical libraries or components,
- 35 • Database,
- 36 • Modeler.

37 We will discuss each of these, devoting particular attention to their inter-
38 faces.

39 **2.1. The Application**

40 The problem to be solved by a SANS system typically derives from a phys-
41 ics, chemistry, etc., application. This application would normally call a library
42 routine, picked and parametrized by an application expert. Absent such an
43 expert, the application calls the SANS routine that solves the problem.

44 For maximum ease of use, then, the API of the SANS routine should
45 be largely similar to the library call it replaces. However, this ignores the
46 issue that we may want the application to pass application metadata to the
47 SANS system. Other application questions to be addressed relate to the
48 fact that we may call the SANS system repeatedly on data that varies only
49 a little between instances. The paradigmatic example here is the sequence
50 of linear systems to be solved in the course of a nonlinear (Newton) pro-
51 cess. In such cases we want to limit the effort expended by the Analysis
52 Modules.

53 The solution to both problems is to extend the notion of problem
54 data to a ‘dataset’, which can contain application metadata, as well as
55 knowledge about the context of the system call. Components accepting
56 such datasets as input are said to have an ‘extended interface’. We will go
57 into the matter of this below; see Section 3.3.

58 **2.2. Analysis Modules**

59 Analysis modules have a two-level structure of categories and ele-
60 ments inside the categories. Categories are mostly intended to be concep-
61 tual, but they can also be dictated by practical considerations.

- 62 In the case of linear algebra problems, conceptual categories are
63 • pertaining to the nonzero structure of matrices (for sparse prob-
64 lems);
65 • norm-like properties (including diagonal dominance);
66 • spectral properties.

67 As an example of a nonconceptual categories, one can imagine the set of
68 elements required by a certain algorithm.

69 An analysis element can either be computed exactly or approximately.
70 For instance, the nonzero structure of a matrix can be computed exactly
71 at very little cost, but bounds on the spectrum will in practice only be
72 approximated. For some approximations, the degree of confidence can be
73 quantified, but in other cases one can at best indicate what algorithm was
74 used to compute them.

75 The output interface of the modules is defined by our standard for
76 numerical metadata.

77 The input interface is slightly more complicated. Here we remark that
78 modules have to accept the same kind of data as the numerical compo-
79 nents do, so we can adopt the extended interface here too.

80 **2.3. Intelligent Switch**

81 The intelligent switch determines which library code to apply to the
82 problem. However, the method choice can be a composite decision, where
83 certain stages can be considered preliminary transformations of the prob-
84 lem. Since such a transform maps the original problem to another, for
85 which other numerical metadata applies, the switch can choose to rerun
86 the analysis modules. This approach is expensive but likely to be accurate.

87 The alternative is to use only the initial metadata, and decide all
88 transforms together. Of course, certain transforms leave certain categories
89 of metadata invariant. For instance, scaling a matrix leaves the sparsity
90 structure intact.

91 **2.4. Numerical Components**

92 In order to make numerical library routines more managable, we
93 embed them in a component framework. This will also introduce a level
94 of abstraction, as there need not be a one-to-one relation between library
95 routines and components. In fact, we will define two kinds of compo-
96 nents:

- 97 • library components are uniquely based on library routines, but they
98 carry a specification in the numerical adaptivity language (Section
99 3.2) that describes their applicability;

100 • ‘numerical component’ are based one or more library routines, and
101 having an extended interface (Section 3.3) that accomodates passing
102 numerical metadata.

103 This distinction allows us to make components corresponding to the spe-
104 cific algorithm level (‘Incomplete LU with drop tolerance’) and generic
105 (‘preconditioner’).

106 *2.4.1. Transform Components*

107 In certain problem domains it may be possible to pick a routine (or
108 component in our framework) that solves the stated problem by itself.
109 However, in other cases the solution is effected by the interplay of vari-
110 ous pieces of software, such as the preconditioner and iterative method in
111 iterative linear system solution.

112 We can take this modularity one step further by introducing ‘trans-
113 form components’ which map one problem into another. Examples here
114 would be permutations or scalings of matrix problems, prior to choosing
115 the preconditioner and iterative method.

116 Presumably there is a choice of mappings, so we need to pass numer-
117 ical metadata to a transform component. In fact, a transform will have
118 largely the same extended interface as other numerical components.

119 Like numerical components, a transform can be queried as to the
120 numerical metadata that is needed for determination of the mapping
121 choice.

122 Applying the transforms is under control of the intelligent switch.

123 **2.5. Database**

124 The database of a SANS system contains information that couples
125 problem features to method performance. While problem features can be
126 standardized (this is numerical metadata), method performance is very
127 much dependent on the problem area and the actual algorithm.

128 As an indication of some of the problems in defining method perfor-
129 mance, consider linear system solvers. The performance of a direct solver
130 can be characterized by the amount of memory and the time it takes.
131 The amount of memory here is strongly variable between methods, and
132 should perhaps be normalized by the memory needed to store the prob-
133 lem. For iterative solvers, the amount of memory is usually limited to
134 a small multiple of the problem memory, and therefore of less concern.
135 However, in addition to the time to solution, one could here add a mea-
136 sure such as “time to a certain accuracy”, which is interesting if the linear
137 solver is used in a nonlinear solver context. There is no counterpart to this

138 measure in direct solvers, other than the trivial measure that the time to
139 any accuracy is the same.

140 **2.6. Modeler**

141 The intelligence in a SANS system resides in two components: the
142 intelligent switch which makes the decisions, and the modeler which draws
143 up the rules that the switch applies. The modeler draws on the database
144 of problem characteristics (as laid down in numerical metadata) to make
145 rules express in an ‘adaptivity specification language’ (Section 3.2).

146 **3. INTERFACES**

147 **3.1. Numerical Metadata**

148 Numerical metadata is data associated with the numerical data. This
149 can either be

- 150 • derived metadata: information derived from the numerical data, or
- 151 • application metadata: facts known a priori and normally not passed
152 from the application to the library routines.

153 In the NMD library, described in Ref. 2 we have standardized the API
154 to these data. This also defines the interface between the analysis modules
155 and the intelligent switch.

156 **3.1.1. Derived Metadata**

157 Derived numerical metadata comprises such categories as

- 158 • structural metadata, relating to the nonzero structure of a sparse
159 matrix;
- 160 • norm-like properties, including diagonal dominance; these first two
161 categories are typically cheaply computable up to reasonable round-
162 off;
- 163 • spectral information, giving some estimate of the spectrum or sin-
164 gular values of a matrix;
- 165 • other measures of a matrix such as departure from normality; these
166 last two measures can not be computed exactly at a reasonable cost,
167 but estimates—though more expensive than for the first two cate-
168 gories—can be obtained at a cost that is still justifiable as part of
169 preprocessing.

170 In our paper⁽²⁾ we proposed a core repertoire of numerical metadata cat-
171 egories, but the NMD language definition allows extension. For instance,

172 one could introduce a category to account for the quantities measured in
173 Ref. 3.

174 **3.1.2. Application Metadata**

175 Application metadata is numerical metadata that derives from knowl-
176 edge of the application. Typical examples are

- 177 • grid properties;
- 178 • nature of the problem;
- 179 • properties of the operator if PDE.

180 Such information can be useful to an intelligent system (for instance,
181 knowing positive definiteness obviates the need to infer this fact) but is
182 usually dropped because the interface between application and numerics
183 has no way of passing it.

184 **3.2. Adaptivity Specification Language**

185 A language for specifying the rules that control intelligent choice of
186 algorithms is still a topic of research. Such a language will be used as the
187 interface between the modeler and the intelligent switch, where the mod-
188 eler extracts rules from the database, and the switch applies them to spe-
189 cific data.

190 An adaptivity specification language can also be used in numerical
191 components. One can envision library components coming equipped with
192 suitably formulated rules describing their applicability. This adds a seman-
193 tic side to the interface specification of components.

194 **3.3. Extended Interfaces**

195 The extended interface captures more the semantics than the syntax:
196 it contains

- 197 • routine parameters if the component is based on a single routines;
198 missing parameters are filled with default or determined values;
- 199 • the union of all routine parameters if the component contains more
200 than one routine; in this case parameters can be specified for any
201 and all, only the relevant ones are used;
- 202 • numerical metadata: the presence of this makes it possible for the
203 component to be intelligent and choose between the wrapped rou-
204 tines.

205 We also enhance the output side of components. In the case of numer-
206 ical components this allows for performance data to be returned; with

207 transform components this allows metadata to be returned, describing the
208 transformed problem.

209 **4. CONCLUSION**

210 We have outlined precise definitions of the modules and interfaces
211 in a SANS system for algorithm choice. Some of these interfaces have
212 been formally defined in our research, others are in a process of being
213 developed.

214 **ACKNOWLEDGMENTS**

215 This research was supported in part by NSF NGS program under
216 grant ACI-0203984 and in part by the AMS/MICS office of the DOE
217 under contract DE-ACO5-00OR22725 with UT-Battelle, LLC., and in part
218 by the Los Alamos Computer Science Institute through the subcontract
219 # R71700J-29200099 from Rice University.

220 **REFERENCES**

- 221 1. J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, R. C. Whaley,
222 and K. Yelick, Self Adapting Linear Algebra Algorithms and Software, in: *Proc. of the*
223 *IEEE* (2004).
- 224 2. V. Eijkhout and E. Fuentes, A proposed standard for numerical metadata. Technical
225 Report ICL-UT 03-02, Innovative Computing Laboratory, University of Tennessee,
226 2003. *Poster presentation at Supercomputing* (2003).
- 227 3. S. T. Xu, E. J. Lee, and J. Zhang, An Interim Analysis Report on Preconditioners and
228 Matrices, *Technical Report* 388-03, University of Kentucky, Lexington; Department of
229 Computer Science (2003).