

Automating scientific and engineering computing

Elisha Sacks*

Computer Science Department
Princeton University
Princeton, NJ 08544

Leo Joskowicz

IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Abstract

This paper describes research on automating scientific and engineering computing by combining domain knowledge, mathematical theory, artificial intelligence techniques, and numerical software. We present two case studies: kinematic analysis of mechanisms and phase space analysis of dynamical systems. The case studies illustrate our general strategy of identifying an important task domain, formalizing the domain knowledge and analysis tools of experts, and incorporating the formalization into a useful problem solver. The problem solvers could not work without a robust, high-level interface to conventional numerical software. We describe an initial interface that manages a root finder, a continuation package, an ordinary differential equation integrator, and a Lyapunov exponent calculator.

Introduction

Our longterm research goal is to automate scientific and engineering computing. Scientists and engineers spend much of their time developing programs, running them on large data sets, and interpreting the outputs. The broader and deeper the problem, the harder these tasks become. We aim to develop programs that shoulder a significant proportion of the work. The programs should combine mathematical theory, domain knowledge, numerical software, computer graphics, and artificial intelligence. Mathematical theory provides abstract concepts and reasoning methods. Domain knowledge provides the basis for selecting solution methods and effective simplifying assumptions. Numerical packages provide quantitative information about mathematical models. Computer graphics helps scientists and engineers visualize complicated objects and processes. Artificial intelligence provides methods for modeling, incorporating heuristics, and managing assumptions and defaults.

Existing software packages support some aspects of scientific and engineering computing. However, they are restricted in the class of problems they can solve

and the quality of results they produce. By and large, these packages produce large amounts of quantitative data for specific problems, leaving their interpretation to the user. Our goal is to provide the user with a concise, qualitative description of the problem solution along with any desired quantitative information. Qualitative descriptions identify the key aspects of the problem, provide an abstract understanding of the solution, and organize the quantitative information into a coherent representation.

Our strategy for automating scientific and engineering knowledge is to identify important task domains, formalize the analysis tools of experts, and incorporate the formalization into useful problem solvers. So far, we have tackled phase space analysis of dynamical systems and kinematic analysis and simulation of mechanisms.

Dynamical systems

Phase space analysis determines the steady-state behavior of dynamical systems, such as ordinary differential equations and maps. Dynamical systems provide a rich language for modeling evolving physical systems, such as chemical reactions, circuits, human physiology, plasma flow, space vehicles, and weather patterns. Standard numerical integration packages can solve specific equations with exact initial conditions. But realistic models contain imprecisely specified parameters and imprecisely observable initial conditions. A complete analysis must provide a compact, insightful description of all possible solutions. Experts analyze dynamical systems through a combination of theoretical dynamics, numerical simulation, symbolic algebra, and geometrical reasoning. We have developed an analysis program for one-parameter planar dynamical systems that performs comparably to experts. The program constructs phase diagrams, which represent the steady-state behavior over a range of initial conditions, and bifurcation diagrams, which represent the changes in steady-state behavior over a range of parameter values. It displays the diagrams symbolically and graphically. The program has solved an open problem in cardiology [Sacks and Widman, 1992] and corrected mistaken

*Supported by the National Science Foundation under grant No. IRI-9008527 and by an IBM grant.

From: AAAI Technical Report FS-92-01. Copyright © 1992, AAAI (www.aaai.org). All rights reserved.
analyses in major journals [Sacks, 1991a]. We are extending it to larger systems and integrating it with specialized visualization software.

Mechanism kinematics

Kinematic analysis determines the constraints on the behavior of a rigid part mechanism (such as gears, cylinder locks, clutches, and transmissions) imposed by the shapes of its parts and by the contacts among them. It provides qualitative and quantitative information about the working of the mechanism that is essential for many tasks of mechanical engineering, including kinematic simulation, design generation and validation, and catalog construction. The computational complexity of kinematic analysis depends on the shapes and motion types of the parts of the mechanism, and is in general intractable.

We studied how engineers efficiently analyze large mechanisms despite the exponential worst-case complexity of the task. We observed that they simplify the task by building mechanisms modularly and by restricting the shapes, motions, and interactions of parts. They derive the kinematics of a complex mechanism by decomposing it into manageable subassemblies, deriving the kinematics of the subassemblies, and composing the results. The most common subassemblies are sets of parts linked by permanent joints, called linkages, and sets of parts that move along fixed spatial axes, called fixed-axes mechanisms. Fixed-axes mechanisms decompose further into pairs of interacting parts, called kinematic pairs.

We systematized and automated the engineering strategy. We defined a class of mechanisms for which kinematic analysis is feasible and implemented an efficient kinematic analysis algorithm for this class. The feasible class contains linkages, fixed-axes assemblies, and mechanisms with linkage and fixed-axes subassemblies. The precise definition appears in our journal article [Joskowicz and Sacks, 1991]. We demonstrated that the feasible class covers most mechanisms by surveying some 2500 mechanisms from a mechanical engineering encyclopedia [Artobolevsky, 1979]. We tested the correctness and efficiency of the kinematic analysis program on a dozen mechanisms from the encyclopedia. In each case, it output the correct answer in under a minute.

We worked within the configuration space (CS) representation of mechanical engineering. Intuitively, the CS of a mechanism is the space of non-overlapping configurations of its parts. It partitions into regions of uniform part contacts separated by boundaries where part contacts change, called a region diagram. Each region is specified by equality and inequality constraints that express part contacts. The regions define the operating modes of the mechanism. Mode transitions occur when the configuration shifts between adjacent regions. Each path through CS defines a possible behavior of the mechanism, which is realizable under an appropri-

ate driving motion. The region diagram specifies the behaviors that are realizable under all driving motions.

The kinematic analysis program constructs the region diagram of a feasible mechanism by degrees of freedom modeling, subassembly analysis, and composition. The input is a geometric description (boundary representation) and an initial configuration (position and orientation) of the parts. Modeling decomposes the mechanism into linkages and kinematic pairs and finds the degrees of freedom of the fixed-axes parts. Subassembly analysis constructs region diagrams for the linkages and the kinematic pairs. Composition constructs the mechanism region diagram by composing the subassembly region diagrams.

We also developed a kinematic simulation program that derives the behavior of a mechanism for a given driving motion. The program traces the CS path that the mechanism traverses under the driving motion, constructing the regions along the path by subassembly analysis and composition. It also simulates external forces and frictions, using a simple model of dynamics that captures their steady-state effect without the conceptual and computational cost of dynamical simulation. It outputs a concise, symbolic interpretation of the behavior of the mechanism and a realistic, three-dimensional animation. The program covers most mechanisms in the engineering encyclopedia, including ones with complex part shapes, varying part contacts, and multiple driving motions. It runs much faster than the analysis program because it generates only a portion of the region diagram, less than 1% in our test cases [Sacks and Joskowicz, 1992].

Analysis of a feeder

We illustrate the kinematics programs on a mechanism that feeds blocks from a stack onto a processing table (Figure 1). The driving motion rotates the driver shaft, which moves the link, which slides the piston left and right. Each time the piston slides left, one block drops onto the table due to gravity. Each time it slides right, it pushes the bottom block onto the table.

The inputs are the part specifications and initial configurations, the gravitational forces on the blocks, and the motion "driver rotates counterclockwise". Each part is specified by its shape and motion type: fixed, fixed-axes, or linkage. Fixed-axes parts move along fixed spatial axes, whereas linkage parts need not. The fixed parts form the frame, the fixed-axes parts form fixed-axes subassemblies, and the linkage parts along with the connected fixed-axes parts form linkages. In the feeder, the driver mounting, magazine, and processing table form the frame, the driver, link, pins, and piston form a linkage, and the frame, driver, piston, and blocks form a fixed-axes subassembly.

The analysis program constructs the feeder region diagram. Degrees of freedom modeling infers that the driver rotates around its mounting, that the piston translates horizontally, and that each block translates

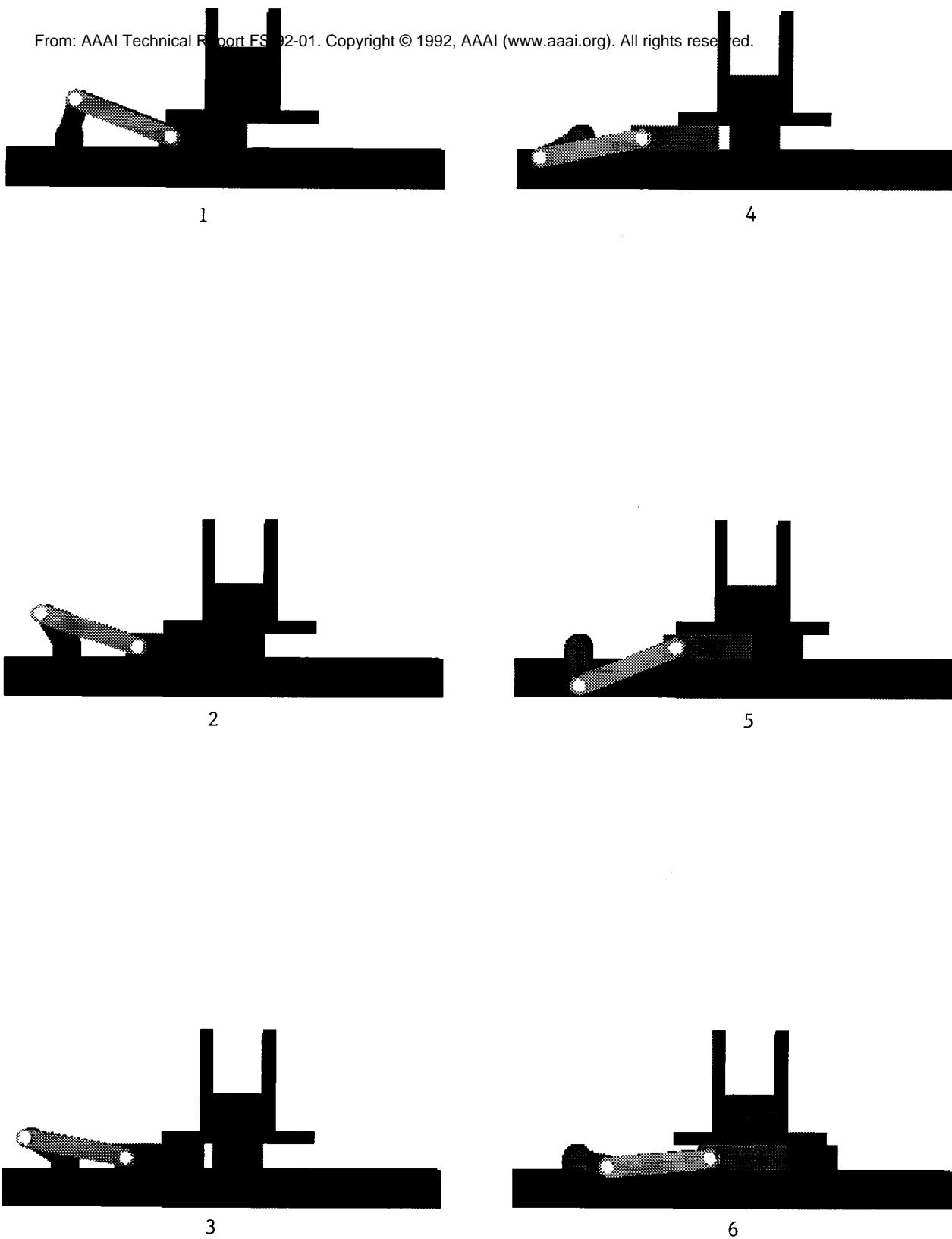


Figure 1: Configurations from a kinematic simulation of the feeder.

horizontally and vertically. Subassembly analysis derives the interactions between the pairs of blocks, between the piston and the blocks, and within the linkage. For example, it finds that rotating the driver slides the piston left and right, and that the piston supports the bottom block in the initial configuration. Composition produces a global region diagram containing 2115 regions.

The simulator derives the CS path that the mechanism traverses when the driver rotates and gravity acts on the blocks. The path goes through 48 of the 2115 global regions. Figure 1 shows one configuration from each of the first six segments in the path, which represent the first cycle of the feeder. Figure 2 shows the program's textual interpretation of the motion within the segments. Segment 1 lies in the initial region. The contact between the piston and the bottom of block 1 prevents the blocks from dropping. The program constructs a path segment in which the driver rotates, the link moves, the piston retracts, and the other parts do not move. The segment ends when the piston moves out from under block 1, causing a contact change. In segment 2, gravity causes the blocks to drop onto the table. In segment 3, the driver moves the piston left. In segment 4, the driver moves the piston right until it touches block 1. In segment 5, the contact between the piston and the side of block 1 enables the piston to push the block to the right. In segment 6, block 1 breaks contact with block 2 and continues right. The cycle repeats until the magazine empties.

Intelligent numerical software

The task domains of scientific and engineering computing share a need for better numerical software. Current numerical software places many burdens on the user, making it hard to use and error prone. For example, suppose someone wishes to find the roots of a polynomial using a typical numerical library. He must find the appropriate subroutine in the index, learn its input/output format, and write a program that passes the polynomial to the subroutine in its format and returns the roots in his format. He must then compile, link, and run the program. Finally, he must assess the output. If the roots seem unreasonable, he must look for programming and interface errors. If the program appears correct, he must look for numerical errors or for peculiarities in the polynomial that violate the assumptions of the subroutine.

We are developing a smart interface that makes numerical libraries easy to use and reliable by automating the relevant mathematical theory, symbolic algebra, and programming formats [Sacks, 1991b]. The current interface manages a root finder, a continuation package, an ordinary differential equation integrator, and a Lyapunov exponent calculator. We plan to extend it to domains such as optimization, differential geometry, and partial differential equations. The interface accepts a high-level problem description, se-

Segment 1

Input motion: DRIVER rotates around its z axis.
 DRIVER makes PISTON translate along its x axis.
 DRIVER rotates from $cd=0.0$ to $cd=2.1268$.
 PISTON translates from $xp=10.0$ to $xp=5.0$.
 B1 is stationary at $xb1=12.0$; $yb1=1.0$.
 B2 is stationary at $xb2=12.0$; $yb2=3.0$.
 B3 is stationary at $xb3=12.0$; $yb3=5.0$.

Segment 2

Input motion: B3 translates along its y axis.
 B3 makes B2 translate along its y axis.
 B2 makes B1 translate along its y axis.
 DRIVER is stationary at $cd=2.1268$.
 PISTON is stationary at $xp=5.0$.
 B1 translates from $yb1=1.0$ to $yb1=-1.0$.
 B2 translates from $yb2=3.0$ to $yb2=1.0$.
 B3 translates from $yb3=5.0$ to $yb3=3.0$.

Segment 3

Input motion: DRIVER rotates around its z axis.
 DRIVER makes PISTON translate along its x axis.
 DRIVER rotates from $cd=2.1268$ to $cd=3.1416$.
 PISTON translates from $xp=5.0$ to $xp=4.0$.
 B1 is stationary at $xb1=12.0$; $yb1=-1.0$.
 B2 is stationary at $xb2=12.0$; $yb2=1.0$.
 B3 is stationary at $xb3=12.0$; $yb3=3.0$.

Segment 4

Input motion: DRIVER rotates around its z axis.
 DRIVER makes PISTON translate along its x axis.
 DRIVER rotates from $cd=-3.1416$ to $cd=-2.1268$.
 PISTON translates from $xp=4.0$ to $xp=5.0$.
 B1 is stationary at $xb1=12.0$; $yb1=-1.0$.
 B2 is stationary at $xb2=12.0$; $yb2=1.0$.
 B3 is stationary at $xb3=12.0$; $yb3=3.0$.

Segment 5

Input motion: DRIVER rotates around its z axis.
 DRIVER makes PISTON translate along its x axis.
 PISTON makes B1 translate along its x axis.
 DRIVER rotates from $cd=-2.1268$ to $cd=-0.7227$.
 PISTON translates from $xp=5.0$ to $xp=9.0$.
 B1 translates from $xb1=12.0$ to $xb1=16.0$.
 B2 is stationary at $xb2=12.0$; $yb2=1.0$.
 B3 is stationary at $xb3=12.0$; $yb3=3.0$.

Segment 6

Input motion: DRIVER rotates around its z axis.
 DRIVER makes PISTON translate along its x axis.
 PISTON makes B1 translate along its x axis.
 DRIVER rotates from $cd=-0.7227$ to $cd=0.0$.
 PISTON translates from $xp=9.0$ to $xp=10.0$.
 B1 translates from $xb1=16.0$ to $xb1=17.0$.
 B2 is stationary at $xb2=12.0$; $yb2=1.0$.
 B3 is stationary at $xb3=12.0$; $yb3=3.0$.

Figure 2: Computer-generated summary of the first cycle of the feeder.

From: AAAI Technical Report FS-92-01. Copyright © 1992, AAAI (www.aaai.org). All rights reserved.
lects the appropriate numerical subroutine, programs
the problem in subroutine format, runs the program,
corrects for numerical errors and special conditions,
and returns the output in a high-level format. For ex-
ample, it accepts a system of equations in symbolic for-
mat, programs the equations and the Jacobian matrix,
runs a Newton-Rhapson subroutine, eliminates spuri-
ous roots, and returns the remaining roots. The inter-
face benefits human users and other programs. Hu-
mans benefit primarily from the ease of use. Programs
benefit primarily from the reliability of the output,
since they often misbehave badly when given incorrect
data.

Conclusions

Our research demonstrates the importance of automati-
ing the reasoning skills and domain knowledge of sci-
entists and engineers and integrating it with software
for scientific computing. The resulting programs per-
form at the level of human experts in two important
domains of scientific and engineering computing. We
believe the same strategy applies to many other sci-
entific computing tasks. We plan to study the relevance
of our strategy for other tasks and the feasibility of
transferring expertise between domains.

References

- I. Artobolevsky. *Mechanisms in Modern Engineering Design*, volume 1–4. MIR Publishers, Moscow, 1979. English translation.
- Leo Joskowicz and Elisha Sacks. Computational kine-
matics. *Artificial Intelligence*, 51:381–416, 1991.
- Elisha Sacks and Leo Joskowicz. Automated modeling
and interpretation in mechanism analysis. *Computer-
Aided Design*, 1992. in press.
- Elisha Sacks and Lawrence Widman. Nonlinear heart
model predicts the range of heart rates for electrical
alternans in pericardial effusion. Technical Report
CS-TR-365-92, Princeton University, 1992.
- Elisha Sacks. Automatic analysis of one-parameter
planar ordinary differential equations by intelli-
gent numerical simulation. *Artificial Intelligence*,
48(1):27–56, 1991.
- Elisha Sacks. A smart interface for numerical soft-
ware. In *Proceedings of the 8th Israeli Symposium
on Artificial Intelligence and Computer Vision*, 1991.
appears as Princeton CS-TR-341-91.